GlobalServerAppGuide

Выпуск 0.0.33

"Бизнес Технологии" ООО

сент. 13, 2024

Оглавление

Часть I

Введение

Глава 1

Предисловие

Добро пожаловать в `Руководство разработчика GlobalFramework for Postgresql`.

Цель руководства предоставить информацию о разработке в `GlobalFramework`. Руководство так или иначе касается всех аспектов работы с данной технологией.

1.1 На кого ориентировано руководство

Руководство ориентировано на инженеров-разработчиков, а также выпускников (студентов старших курсов) классических и технических университетов и других вузов, имеющих подготовку по программированию и продолжающих специализироваться в разработке корпоративных информационных систем на базе Global3 Framework.

1.2 Программные средства для работы

Обязательные инструменты:

- IntelliJ IDEA
  Среда разработки
- Liberica JDK 8
- DBeaver
  Среда написания sql запросов
- `Global Server` (поставляется отдельно)
  Сервер приложения
- `GlobalFramework Starter Kit` (поставляется отдельно)
  Набор шаблонов и файлов для упрощенной настройки рабочего места
- git

Рекомендуемые инструменты:

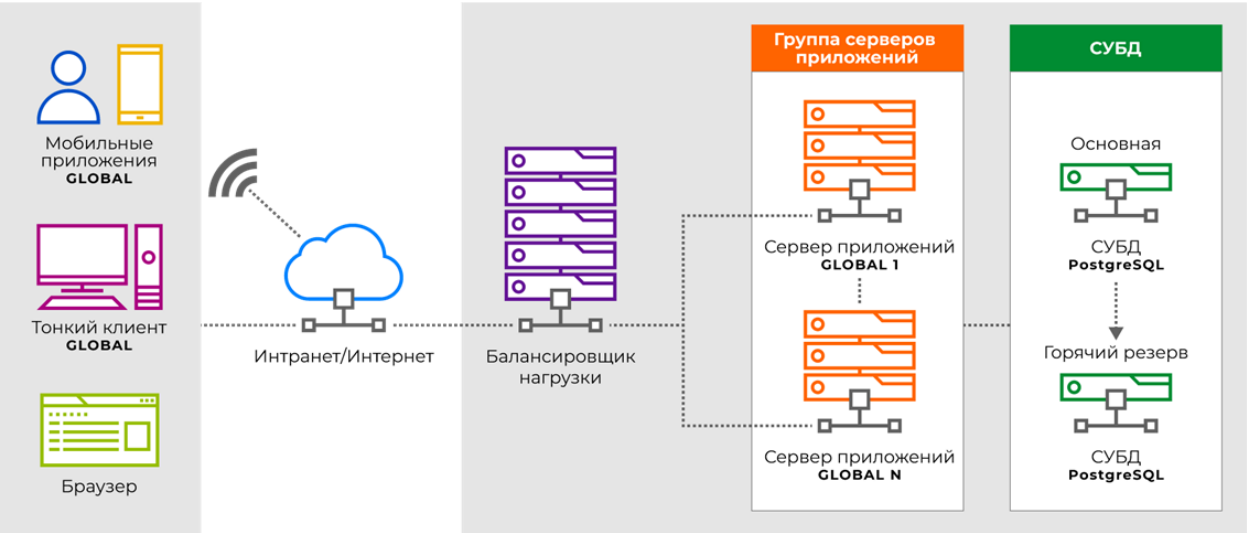
- Notepad++
  Удобный быстрый текстовый редактор
- VS Code
  Среда разработки для работы с контейнерами и написания документации
- Chrome
- Adobe Acrobat Reader DC
- PuTTY
  Доступ к контейнеру через ssh в виде командной строки
- Winscp
  Доступ к контейнерам через ssh в виде файлового менеджера
- TortoiseSVN
  Удобное средство работы с `svn`(система контроля версий)

Глава 2

Обзор

Global3-FrameWork является мощным и удобным инструментом разработки кроссплатформенных решений.

2.1 Архитектура системы



2.1.1 Тонкий клиент

Тонкий клиент подключается к серверу приложений в терминальном режиме. Что позволяет работать на как на компьютерах так и на планшетах через канал с низкой пропускной способностью. Это возможно так как в терминальном режиме пользователь получает только минимально необходимый для работы набор данных. К примеру если пользователь открыл список содержащий 10 000 записей. К нему на эран будет отправлена только видимая часть и кэш примерно 100-200 записей (зависит от размеров экрана)

Тонкий клиент может работать как веб приложение в любом браузере поддерживающим стандарт `html 5.0`. Или поставляться в виде отдельного приложения на базе chromium

В браузер может быть установлено дополнительное расширение, позволяющее тонкому клиенту напрямую работать с файлами, оборудованием и крипто провайдерами. В случае наличия такого расширения, работа в браузере неотличима от работы в нативном приложении.

2.1.2 Мобильный клиент

Существует линейка мобильных приложений позволяющая использовать полный перечень возможностей мобильного устройства. Для реализации специализированных приложений, GlobalFramework предоставляет возможность написание web сервисов, посредством которых возможно взаимодействие с нативным мобильным приложением. А так же набор библиотек облегчающих написание мобильного приложения.

2.1.3 Балансировщик нагрузки

Балансировщик нагрузки распределяет пользователей между серверами приложений в кластере.

2.1.4 Кластер серверов приложений

Нагрузка горизонтально масштабируется между серверами приложений.

Сервера приложений объединяются в кластер для возможности централизованного управления и мониторинга.

2.1.5 Решение

Набор скомпилированных библиотек устанавливаемых в сервер приложения. Решение предоставляет весь перечень прикладной бизнес логики необходимый для работы сервера приложения.

2.1.6 Кластер баз данных

Кластеризация базы данных поддерживается штатными средствами `postgreSQL`. В случае синхронной репликации горячей копии, возможно перенаправление на нее запросов на чтение.

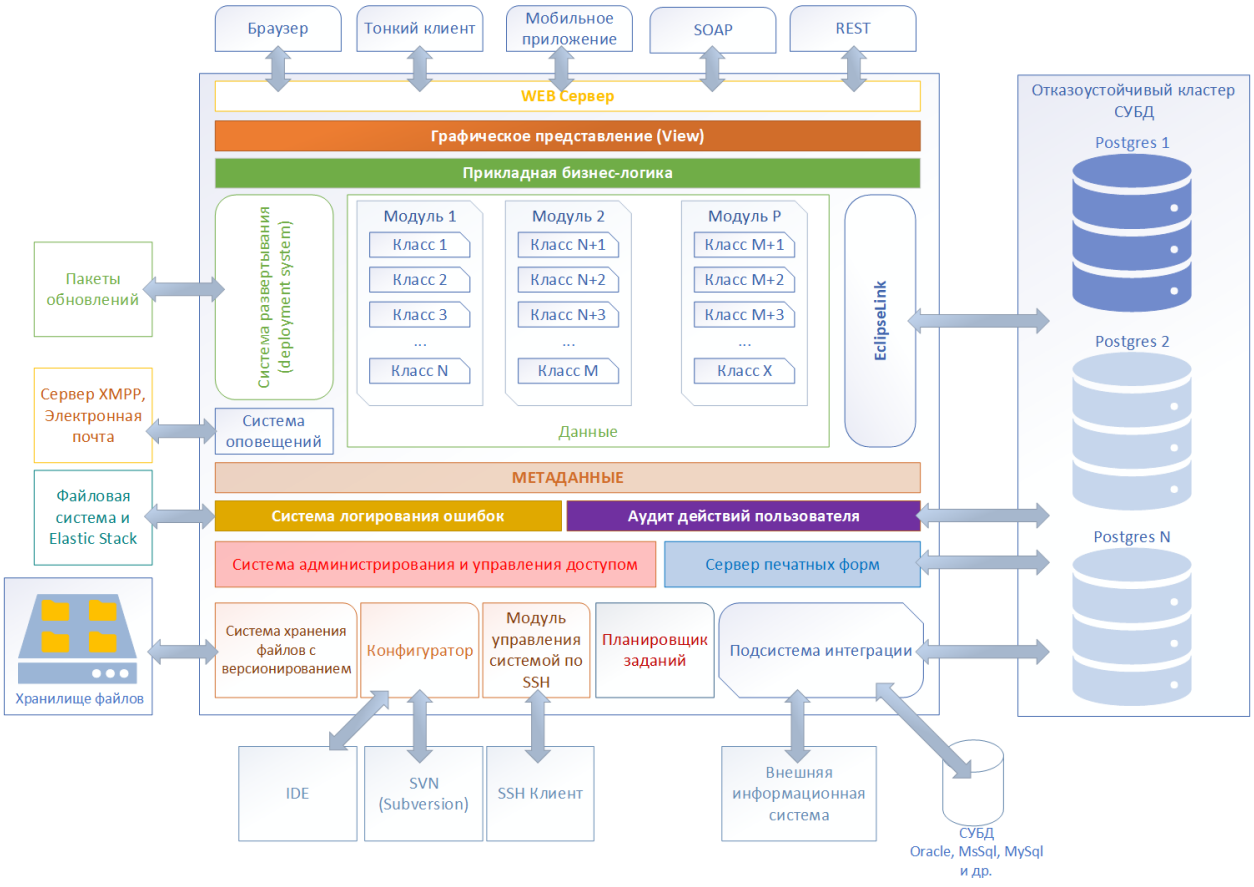
2.2 Уровни разработки

Разделение бизнес логики по уровням позволяет существенно повысить повторное использование кода между пользовательскими интерфейсами и фоновыми сервисами(интеграция, выполнение задач по расписанию и т.д.). Система Global состоит из следующих уровней:

- Интерактивная бизнес логика
  Отвечает за ввод/вывод информации и обработку действий пользователя.
- Автономная бизнес логика
  Осуществляет основную обработку данных.
- Хранение данных
  Реализован СУБД

Фреймворк предоставляет удобные инструменты для взаимодействие между уровнями.

2.3 Принципиальная схема



где:

- Графическое представление
  Вид интерфейса определяется по средством декларации пользовательского интерфейса. Реакция на пользовательские действия программируется на уровне интерактивной бизнес логики.

- Прикладная бизнес логика
  Располагается по большей части в слои автономной бизнес логики. Прикладная бизнес логика группируется по модулям, классам и пакетам
- Метаданные
  Объекты автономной и интерактивной бизнес логике регистрируются в базе данных, в слои мета данных, что позволяет выполнять администрирование и кэширования элементов бизнес логики при необходимости
- Система администрирования
  Определяет права доступа пользователя к интерактивной и автономной бизнес логики
- Система развертывания
  Отвечает за создание и обновление схемы базы данных

2.4 Основные понятия

2.4.1 Класс

Определяет правила хранения и обработки таблицы базы данных.

Класс позволяет существенно ускорить разработку бизнес логики ориентированную на работу с данными. Программисту достаточно обЪявить перечень атрибутов класса чтобы за счет кода генерации получить набор готовых сервисов:

- Интеграцию с `Orm`
  - Pojo объект для хранения данных в кэше
  - Aro объект интеграции pojo в фреймворк
- Каркас автономной логики(`Api`)
  `scala` класс с постфиксом `Api`
- Каркас интерактивной логики(`Avi`)
  `scala` класс с постфиксом `Avi`
- Каркас декларации пользовательского интерфейса(`Avm`)
  `xml` файл с расширением `avm.xml`
- Прикладные сервисы

Подробнее смотри:

- *Классы*
- *Сервисы класса*
- *Тип объекта*

2.4.2 Бизнес объект

Бизнес-объект (БО) - объединение нескольких классов и их коллекций в группу для более удобного манипулирования ими при работе с кэшем и конфигурировании вспомогательных сервисов.

Бизнес объект позволяет:

- Массово загружать данные в транзакционный кэш
  Для бизнес объекта можно указать стратегию загрузки данных существенно уменьшающую количество запросов в базу данных. Так как запросы пойдут не по каждому объекту а по каждому классу бизнес объекта.
- Настраивать права доступа
  По бизнес объекту создается административный объект на котором можно массово выдать привилегии для всех классов бизнес объекта
- Управлять электронной подписью
  Можно настроить правила подписи всего бизнес объекта включая не только шапку но и все вложенные коллекции.
- Настраивать интеграцию и репликацию

2.4.3 Пакет

Позволяют сгруппировать общую бизнес логику в контексте сессии приложения. Пакет является `scala` классом с постфиксом `Pkg`

2.4.4 Выборка

Выборка определяет правило получения, отображение данных и обеспечивает взаимодействие с пользователем. Выборки содержат основную часть интерактивной бизнес логики.

Выборка определяет:

- способ получения данных
- способ отображения данных пользователю
- обработки пользовательских действий

Подробнее смотри:

- *Выборка*

2.4.5 Модуль

Неделимая часть решения. Набор модулей определяет доступную функциональность развернутую в сервере приложения.

Модуль может зависеть от других модулей. В таком случаи такие зависимости так же должны быть установлены в решения для обеспечения корректной работы решения.

2.4.6 Приложение

Приложение является коренной выборкой рабочего сеанса пользователя и задает:

- Главное меню
  Содержит перечень форм, отчетов и операций
- Панель быстрого доступа
  Содержит операции быстрого доступа, а так же поля задающие контекст работы рабочего сеанса, например рабочий период

Пользователь при входе в систему выбирает требуемое ему для работы приложение.

Перечень приложений доступных для пользователя задается администратором системы.

2.4.7 Сессия приложения

Сессия приложения создается на поток прикладной бизнес логики. Сессия приложения предоставляет доступ к сессией базы данных, EclipseLink кэшу, серверу приложения. А так же содержит необходимые инъекции зависимости для работы прикладной бизнес логики.

Сессия приложения создается:

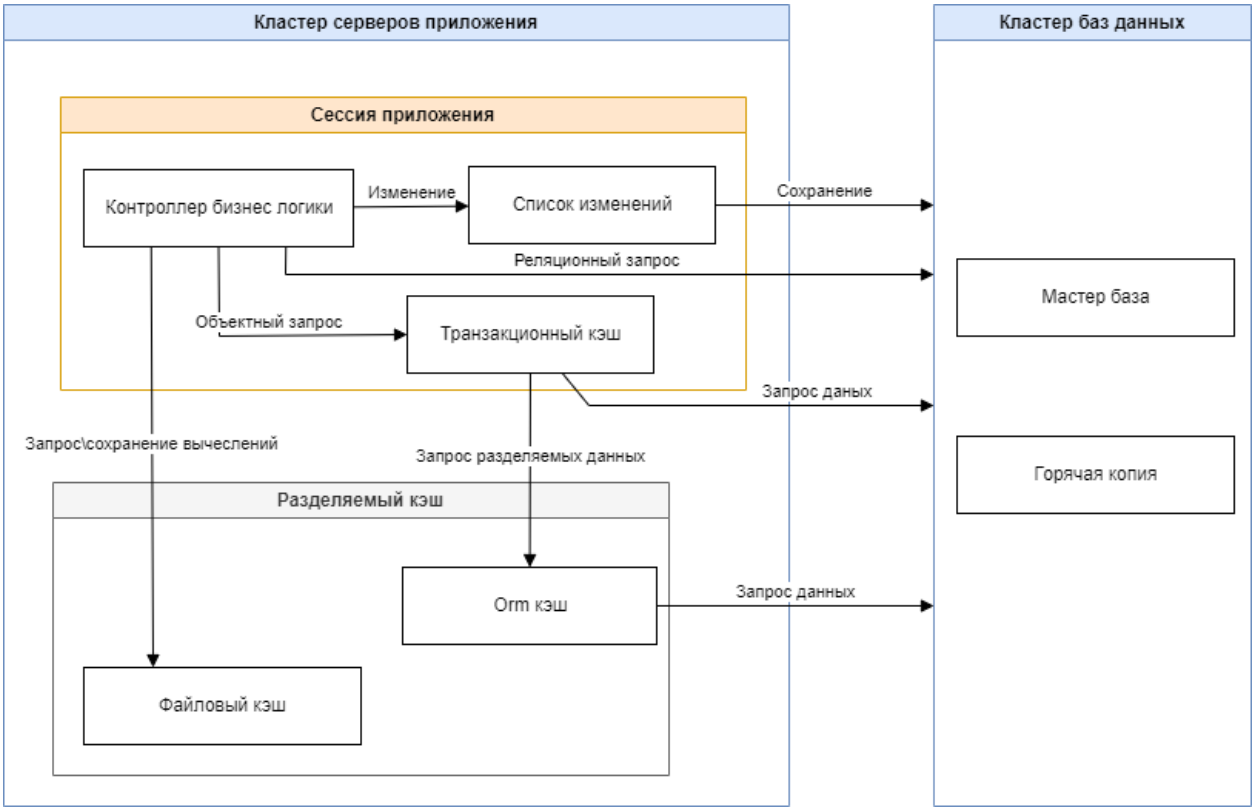
- на каждую mdi форму открытую в приложении
- на rest запрос к серверу приложения

> **Примечание:** Для ускорение rest запросов возможна настройка пула сессий

Подробнее смотри:

- *Сессия приложения*

2.5 Взаимодействие с данными



где:

- *Объектный запрос*
  Запрос данных через `JQPL`, данные запрошенные на данном языке попадают в транзакционный кэш что позволяет их редактировать.

  Применяется к примеру в карточке документа.

- *Реляционный запрос*
  Запрос в базу данных на языке sql, позволяет использовать аналитические функции, и сложную фильтрацию.

  Применяется к примеру в списках.

- *Транзакционный кэш*
  Перечень запрошенных объектов в текущей сессии.

- *Orm кэш*
  Кэш разделяемых объектов. На классе можно включить режим кэширование в разделяемом кэше.

  Применяется на классах типизации, для снижения нагрузки на базу.

- *Файловый кэш*
  Повторяемые вычисления можно кэшировать в файловом кэше.

2.5.1 Объектное взаимодействие

Объектное взаимодействие реализовано на основе библиотеки EclipseLink которая отвечает за низкоуровневое взаимодействие между сервером приложений и базой данный через механизмы Orm.

Объектное взаимодействие позволяет повысить быстродействие системы за счет:

- Кэширования данных
- Применения ленивых коллекций
- Пакетных (Bulk) запросов, которые обеспечивают:
  - Сбор данных по бизнес-объекту
  - Обновление\вставку\удаление

За счет работы с объектами через объектный кэш возможно реализация прикладных сервисов, таких как аудит.

2.5.2 Реляционные запросы

При необходимости возможно взаимодействие с базой данных на прямую по средством язык `sql`.

Глава 3

Начало работы с фреймворком

3.1 Требования к аппаратному обеспечению рабочего места разработчика

Характеристика сервера	Рекомендуемые параметры
ОС	Windows 10 и выше
CPU	8 ядер 3.5 GHz или более
Оперативная память	16 – 32 Gb или более
Память	SSD 20 Gb свободного места
Монитор	разрешение 1920×1080

Использование виртуальной машины может заметно снизить производительность при сборке и отладки проекта

3.2 Создание новой БД для проекта

Для создания базы данных используется команда `CREATE DATABASE`, после которой указывается название базы данных.

Для выполнения запросов будем использовать графический клиент pgAdmin, хотя также можно использовать консольный клиент psql.

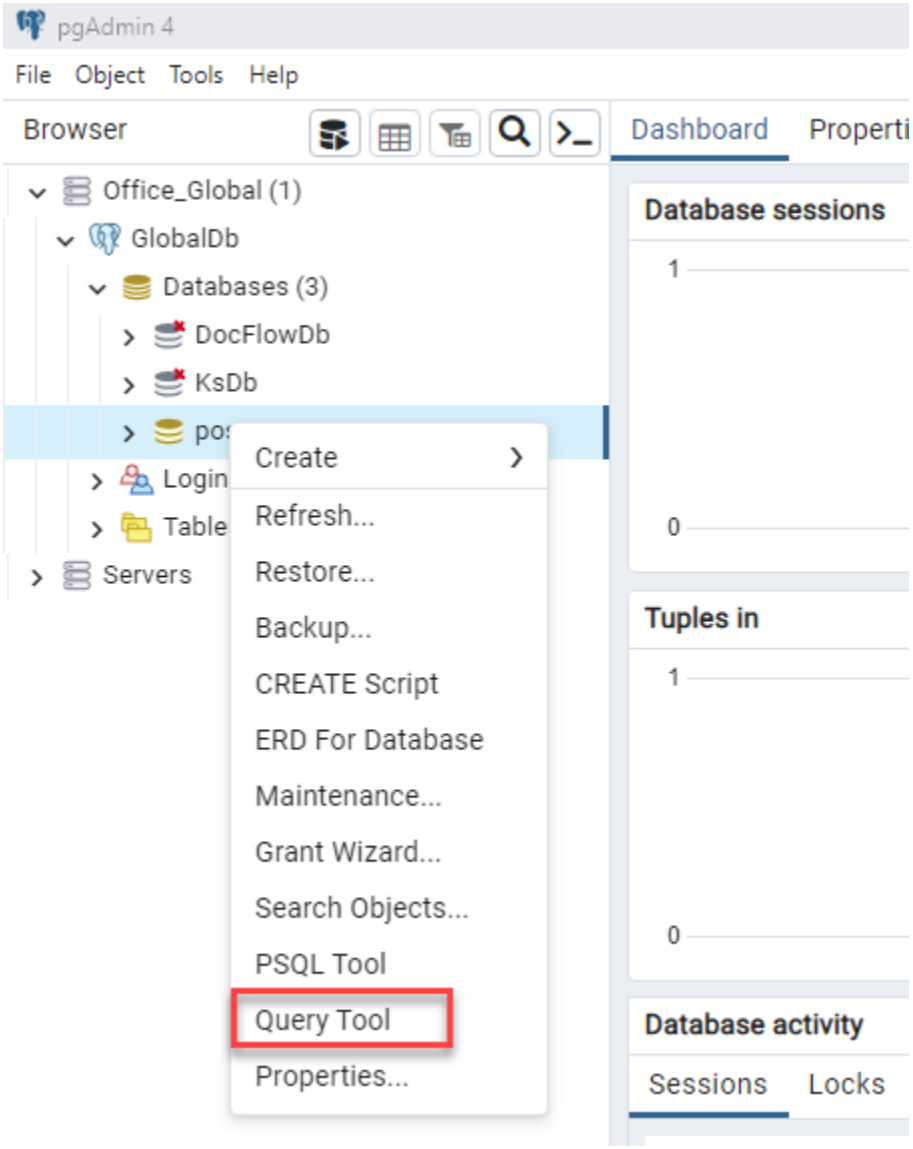
Создадим для проекта нового пользователя бд и базу данных, например:

```
Логин: sampleproject
Пароль: sampleproject

БД: sampleproject
```

Чтобы создать нового пользователя и новую БД:

1. Откройте pgAdmin.
2. Подключитесь под суперпользователем к СУБД.
3. В левой части программы выберите базу данных `postgres`
4. Выделите ее и правой кнопкой мыши запустите инструмент Query Tool



3.2.1 Создание пользователя

В центральной части программы откроется поле для ввода кода SQL.

1. В окно запроса введите следующий код:

```
CREATE ROLE sampleproject WITH
    LOGIN
    NOSUPERUSER
    NOCREATEDB
```

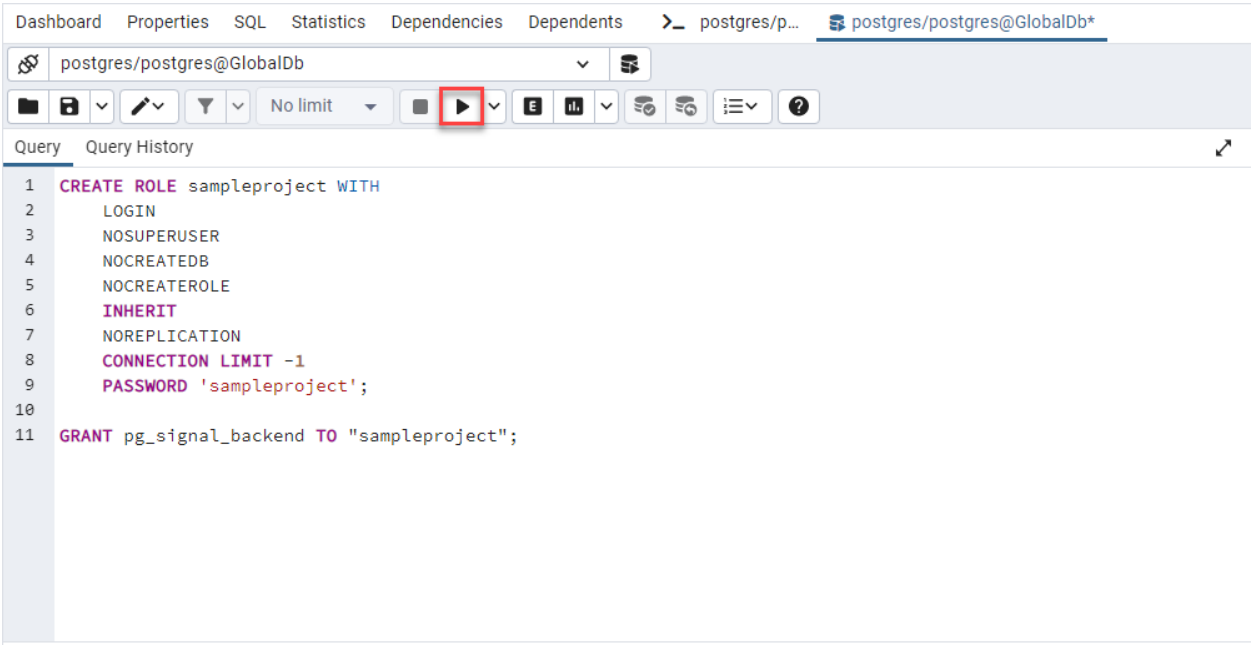
(continues on next page)

(продолжение с предыдущей страницы)

```
    NOCREATEROLE
    INHERIT
    NOREPLICATION
    CONNECTION LIMIT -1
    PASSWORD 'sampleproject';

GRANT pg_signal_backend TO "sampleproject";
```

2. Для создания пользователя выполним операцию `Execute/refresh`.



3.2.2 Создание БД

1. В окно запроса введите следующий код:

```
CREATE DATABASE "sampleproject"
    WITH
    OWNER = "sampleproject"
    ENCODING = 'UTF8'
    CONNECTION LIMIT = -1;
```

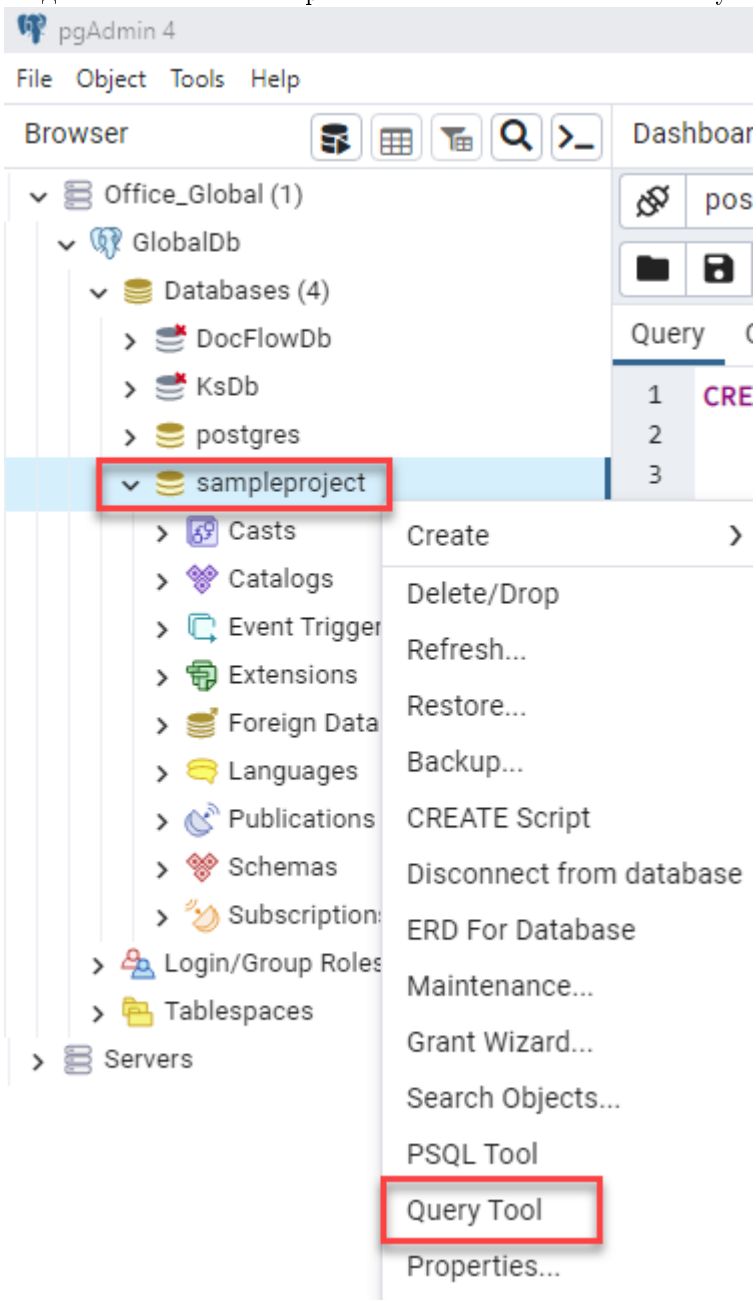
2. Для создания выполните операцию `Execute/refresh`.

3.2.3 Подключение расширений

К созданной БД необходимо подключить расширения, которые использует система `Global`.

Для этого:

1. Обновите список баз
2. В левой части программы выберите базу данных `sampleproject`
3. Выделите ее и правой кнопкой мыши запустите инструмент Query Tool



4. В окно запроса введите следующий код:

```
CREATE EXTENSION if not exists plpgsql;
CREATE EXTENSION if not exists fuzzystrmatch;
CREATE EXTENSION if not exists pg_trgm;
CREATE EXTENSION if not exists pg_stat_statements;
CREATE EXTENSION if not exists "uuid-ossp";
CREATE EXTENSION if not exists dict_xsyn;
CREATE EXTENSION if not exists ltree;
```

5. Для подключения расширений выполните операцию `Execute/refresh`.

3.3 Настройка рабочего места

3.3.1 Установка

1. Установите IntelliJ IDEA Community Edition
2. Установите в IDE IntelliJ IDEA SCALA Plugin
3. Установите Global3FrameworkStarterKit
4. Распакуйте архив с дистрибутивом сервера приложений `globalserver.zip` в директорию `C:\Global3se`
5. Скорипуйте проект в директорию `C:\Projects\projectName`

3.3.2 Настройка переменных окружения Windows

Добавьте переменные окружения Windows:

- `G3_HOME = C:\Global3se`
  Путь к каталогу, в который разархивирован дистрибутив Global 3SE
- `G3_PUBLISH = C:\Global3se\application`
  Путь к каталогу, в который будет производиться сборка проекта

3.3.3 Настройка проекта

Настройку проекта можно сделать в ручном и автоматическом режиме с помощью gsf-cli - утилиты командной строки Global System Framework

Развертывание вручную

1. Откройте в IDE `Intellij Idea` проект
   Для примера будем использовать путь: `C:\Projects\SampleProject\application`
2. Откройте файл `C:\Projects\SampleProject\application\project\orm\config\persistence.xml`
3. Поменяйте параметры соединения с БД Postgres проекта для `persistence-unit pgdev`: \

```
<property name="eclipselink.jdbc.url" value="jdbc:postgresql://pgProject:5432/
↪sampleProject"/>
<property name="eclipselink.jdbc.driver" value="org.postgresql.Driver"/>
<property name="eclipselink.jdbc.user" value="sampleproject"/>
<property name="eclipselink.jdbc.password" value="sampleproject"/>
```

Где:
`eclipselink.jdbc.url` – строка соединения
`eclipselink.jdbc.driver` – драйвер jdbc
`eclipselink.jdbc.user` – пользователь бд
`eclipselink.jdbc.password` – пароль

Конфигурация Global Server

Перед запуском Global Server необходимо добавить информацию о проекте в конфигурационный файл. Для этого:

1. Откройте файл `C:\Global3se\application\config\global3.config.xml` в текстовом редакторе
2. В раздел `databases` добавьте новую базу данных

```
<?xml version="1.0"?>
<database alias="SampleProject"
            driver="org.postgresql.Driver"
            schema="PUBLIC"
            url="jdbc:postgresql://pg12:5432/sampleProject"
            connectionType="proxyShared"
            authenticationType="btk">
    <users>
        <!--user определяет схему-->
        <user name="sampleProject" password="sampleProject"/>
    </users>
    <metaManager mode="Xml"
                defaultNamespace="ru.bitec.app.btk"
                sbtName="SampleProject_sbt"/>
    <eclipseLink persistenceUnitName="pgdev" autoCommit="false"/>
</database>
```

3. В раздел `sbts` добавьте конфигурацию sbt

```
<sbt name="SampleProject_sbt"
    source="C:\Projects\SampleProject\application"
    sourceMode="Dev"
    sbtMode="External"
    gitPullOnSbtStart="false"
    binaryFolder="c:\Global3se\application\appbin"
>
</sbt>
```

4. В разделе `development` включить режим конфигуратора

```
<development>
    <!--enabled - режим отладки. Делает видимым панель отладки в главном
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        меню приложения-->
    <debugMode enabled="true"/>
    <!--enabled - Доступность флага "Конфигуратор" в диалоге
        подключения-->
    <configurator enabled="true"/>
</development>
```

gsf-cli - утилита командной строки Global System Framework

gsf-cli - утилита командной строки Global System Framework предназначена для автоматизации работы разработчика

Инструкция по развертыванию проекта в автоматическом режиме Командная утилита G3Server

3.4 Прикладной проект

Прикладной проект собирает конечное решение из набора прикладных модулей. Собранное решение разворачивается не сервере приложения для работы с пользователями.

3.4.1 Структура проекта

- Проект(Application)
  - `build.sbt`
    Настройка сборки проекта
  - project
    Дополнительные настройки проекта
  - `модуль`
    - `src.main.java`
    - `src.main.scala`
      - `ru.bitec.app.[модуль]`
        Файлы модуля компилируемые языком scala
    - `src.main.resources`
      - `META-INF\module-info.xml`
        Мета данные о модуле
      - `ru.bitec.app.[модуль]`
        Файлы модуля не требующие компиляции
    - build.sbt
      Файл сборки модуля
  - `модуль_N`
  - `модуль_N+1`

Внимание: `ru.bitec.app.[moduleName]` – это **отдельные** каталоги!

Точка в именах директорий запрещена! IDE Intellij Idea в обозревателе проекта группирует каталоги через точку для удобства просмотра

3.4.2 Модуль

Каждый модуль представлен как отдельная директория со строго заданной системой подкаталогов.

Для включения модуля в сборку необходимо объявить его и включить в агрегацию в корневом build.sbt проекта application.

Модуль представляет собой некоторую неделимую совокупность функциональности системы, которая может быть включена в то или инной прикладной проект.

Каждый объект, зарегистрированный в системе, принадлежит какому-либо модулю.

Основными составляющими модуля являются классы, выборки и серверные методы, группированные в пакеты.

Структура каталогов модуля

`[moduleName]`
Директория модуля разбита на строгую иерархию:

- `[moduleName]/src/main/java/ru/bitec/app/[moduleName]`
  Содержит java сущности модуля: pojo классов, перечисления и т.д. Чаще всего сущности создаются автоматически генератором кода.
- `[moduleName]/src/main/resources/META-INF`
  Содержит метаинформацию модуля: Описание, версию, зависимости, перечень приложений.
- `[moduleName]/src/main/resources/orm`
  Содержит индексные файлы ORM.
- `[moduleName]/src/main/resources/ru/bitec/app/[moduleName]`
  Содержит файлы моделей Odm, Orm, Dvm, Avm
- `[moduleName]/src/main/scala/ru/bitec/app/[moduleName]`
  Содержит файлы прикладной бизнес-логики и контроллера view, пакетов: Dvi, Dpi, Api, Avi, Pkg
- `[moduleName]/src/test/scala/ru/bitec/app/[moduleName]`
  Содержит юнит-тесты прикладных методов

Зависимости

Модуль может использовать или требовать наличия некоторого набора других модулей. Это требование выражается в том, что объекты, или функциональность, заложенная в модуль, не будут работать в случае отсутствия в системе другого, требуемого модуля.

Зависимости позволяют увеличить повторное использование кода, за счет выделение частно используемого функционала в низкоуровневые модули.

Базовые модули:

- gtk
  Обеспечивает интеграцию с сервером приложения

- btk
  Основная функциональность фреймворка, доступная всем более высокоуровневым модулям.

Эти модули не требуют для своей работы наличия других модулей, являясь платформой системы.

Зависимость модулей описывается в файле настроек сборки build.sbt для модуля.

Подключение внешних библиотек

Допускается подключение к прикладным модулям внешних библиотек (`*.jar`). В файле build.sbt прикладного проекта необходимо прописать зависимость модуля от этих библиотек.

```
lazy val bts = project.
  dependsOn(btk).
  settings(CommonSetting.setting: _*).
  settings(
  libraryDependencies ++= Seq(
    "org.apache.httpcomponents" % "httpmime" % "4.5.2",
    "org.apache.httpcomponents" % "httpcore" % "4.4.5",
    "org.apache.httpcomponents" % "httpclient" % "4.5.2" )
  )
```

Если подключаемая библиотека опубликована в нестандартном репозитории, необходимо объявить путь к этому репозиторию.

```
lazy val btk = project.
  dependsOn(gtk).
  settings(commonSettings: _*).
  settings(
    resolvers += "nuiton-maven" at
      "https://nexus.nuiton.org/nexus/content/groups/releases/",
    libraryDependencies += "nl.knaw.dans.common" % "dans-dbf-lib" % "1.0.0-beta-10"
  )
```

Если объявить зависимость с двумя символами «%», в качестве разделителя:

`"nl.knaw.dans.common" %% "dans-dbf-lib" % "1.0.0-beta-10"`

к имени библиотеки будет добавляться суффикс с версией scala.

Все библиотеки зависимостей из списка «libraryDependencies» копируются в каталог `.commonlib`.

За формирование каталога `.commonlib` отвечают sbt-задания `publishLibDependencies` и `publishDevDependencies`, выполняемые в зависимости от режима сборки проекта автоматически или вручную.

Внимание: При выполнении sbt задачи `publishDevDependencies` в Intellij Idea может выдаваться ошибка вида:

```
[error] download error: Caught javax.net.ssl.SSLHandshakeException:
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to
find valid certification path to requested target
(sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to
```

```
find valid certification path to requested target) while downloading
https://ivy.global-system.ru/
```

Ошибка проверки возникает из-за того, что в хранилище доверенных корневых сертификатов Java нет CA-сертификата промежуточного сервера для let's encrypt.

Исправить ошибку можно следующим образом:

1. Добавить сертификат в стандартное хранилище доверенных CA-сертификатов
   Для Java приложений сгенерировать новый CA-сертификат R3 для Let's Encrypt можно с помощью утилиты keytool.
   Java Keytool — это инструмент командной строки, который может генерировать пары открытый ключ / закрытый ключ и сохранять их в хранилище ключей. Исполняемый файл утилиты распространяется вместе с Java SDK (или JRE)
   пример использования:

   ```
   keytool -importcert -file mycertfile.pem -keystore cacerts -alias "Alias"
   ```

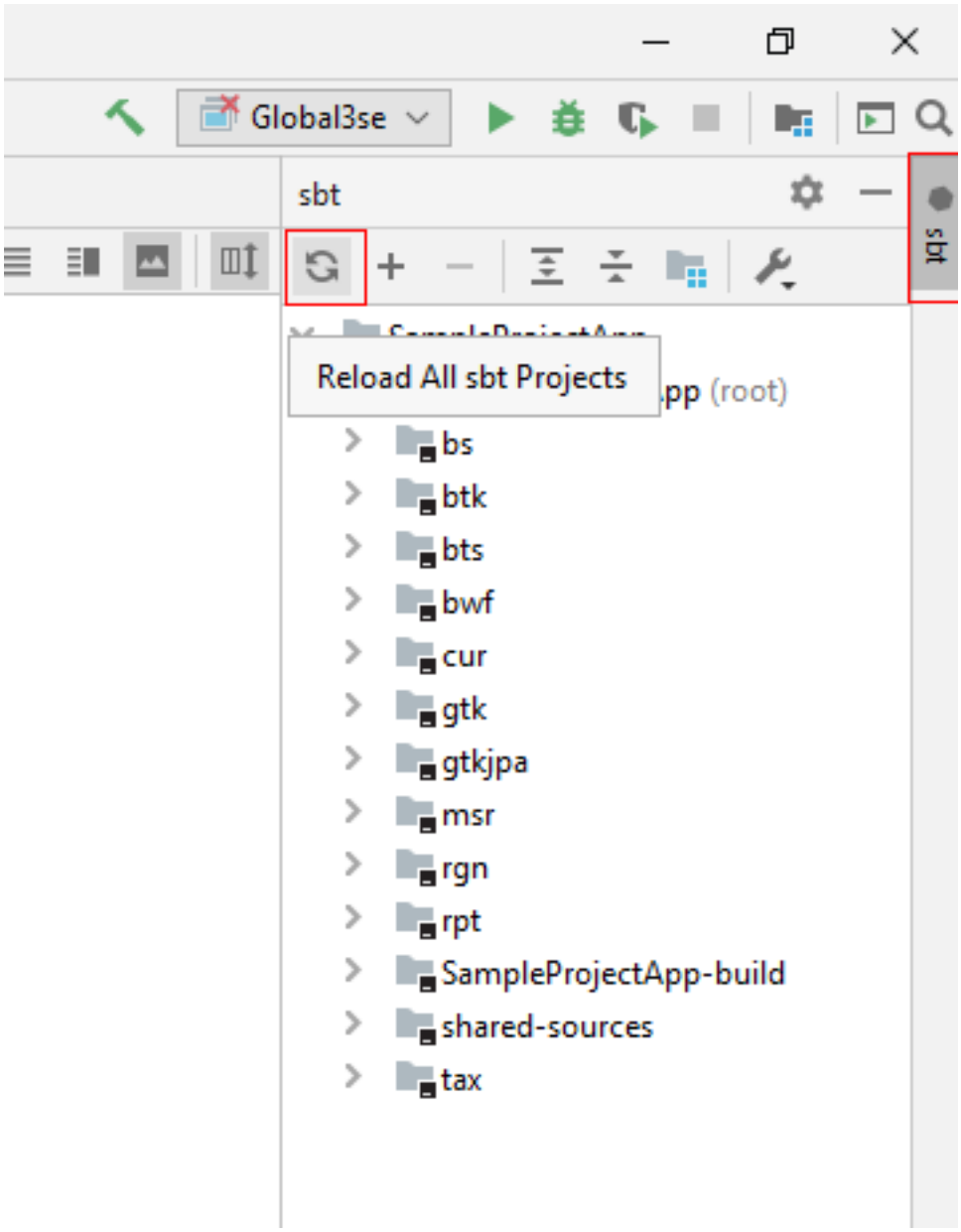
   Стандартный пароль хранилища: changeit

2. Заменяем хранилище
   `C:\Program Files\Java\jdk[версия]\jre\lib\security\cacerts` файлом из архива `Global3FrameworkStarterKit.zip`

3.5 Первая сборка проекта

Когда проект открывается впервые IDE Intellij Idea проиндексирует его состав и выполнит обновление sbt проекта. После этого необходимо выполнить агрегацию библиотек проекта и собрать проект.

1. Дождитесь окончания инициализации проекта и индексации исходного кода
2. Обновите sbt проект



3. Выполните агрегацию библиотек проекта
   Для этого запустите на панели sbt задачу `publishDevDependencies`
   При этом sbt задача просканирует проект на наличие зависимостей и сформирует в корне проекта каталог `.commonlib` со всеми необходимыми библиотеками.

> **Внимание:** При выполнении sbt задачи `publishDevDependencies` в Intellij Idea может выдаваться ошибка вида:
> `[error] download error: Caught javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target (sun.security.validator.ValidatorException: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to`

```
find valid certification path to requested target) while downloading
https://ivy.global-system.ru/
```

Ошибка проверки возникает из-за того, что в хранилище доверенных корневых сертификатов Java нет CA-сертификата промежуточного сервера для let's encrypt.
Для исправления ошибки замините хранилище `C:\Program Files\Java\jdk\[версия]\jre\lib\security\cacerts` файлом из архива `Global3FrameworkStarterKit.zip`

4. Выполнить сборку проекта: Build | Build Project

3.6 Развертывание первоначальных данных

Для генерации начальных данных в БД проекта необходимо провести успешную сборку проекта и запустить утилиту создания таблиц.

1. Если требуется сборка проекта, выполните команду `Build project`
   Пункт меню Intellij Idea: `Build > Build Project`
2. После сборки проекта запустите генерацию таблиц
   В обозревателе проекта Intellij Idea вызовите контекстное меню на корневой директории проекта `application` пункт `External Tolls > Generate tables`
   При этом происходит создание таблиц схемы, установка начальных данных и создается пользователь admin с полными правами(пароль admin)

После этого конфигурация готова для разработки.

3.7 Запуск приложения

Для запуска и отладки приложения вам необходимо ознакомиться с основными принципами запуска и отладки в idea

Для отладки проекта:

1. Выберите в выпадающем списке конфигураций запуска `Global 3 SE`
   Данная конфигурация создается автоматически дополнительным плагином. Конфигурация появляется при следующем запуске idea. В случаи если плагин не установлен, конфигурацию для запуска необходимо создать вручную.
2. Выполните Debug `Global 3 SE`
3. Подключитесь к запущенному серверу
   Для этого откройет браузер по адресу `http://localhost:8080/`

При этом в консоли будет выводиться лог работы сервера. При запуске не должно быть сообщений с типом `[ERROR]` или стеков ошибок.

3.8 Регистрация

При первом запуске система Global Postgres потребует зарегистрировать БД Для регистрации базы данных.

1. Сформируйте файл запроса лицензии
   Для формирования файла запроса лицензии необходимо нажать на кнопку «Сформировать и скачать файл запроса лицензии».
2. Запросите лицензионный файл
   Сформированный файл запроса лицензии вместе с заявкой на регистрацию отправьте по электронной почте контактному лицу ООО «Бизнес-Технологии». В заявке рекомендуется указать имена модулей, которые будут созданы.
3. Зарегистрируйте лицензию
   Для этого в диалоге требования регистрации нажмите на кнопку `Установить лицензионный файл`, и выберете полученный файл в открывшемся диалоге.

3.9 Отладка приложений

Для отладки прикладного кода:

1. Откройте файл с кодом для отладки
2. Установите точку остановки на требуемой строке
3. Откройте приложение в браузере и выполните отлаживаемое действие
   Выполнение программы будет остановлено на отмеченной строке.

3.9.1 Отладка бизнес-логики

Для отладки и тестирования кода Dpi/Api-классов можно воспользоваться Unit-тестами. Для этого, в каталоге с тестами соответствующего модуля создайте класс, унаследованный от `ru.bitec.app.gtk.eclipse.api.ApiTest`.

Смотрите пример: `ru.bitec.app.cur.Cur_CurrencyTest`

3.9.2 Перекомпиляция кода без перезапуска сервера

При отладке приложений, в большинстве случаев, нет необходимости перезапуска веб-сервера при внесении изменений в код любого модуля проекта Application. Для применения изменений необходимо:

1. Выключите кэш
   Выполните пункт меню `Файл \ Использовать кэш` в отлаживаемом приложении
2. Скомпилируйте изменённый модуль
   `Compile module` в контекстном меню модуля или `Build project` на панели инструментов. В отдельных случаях можно компилировать отдельный класс.
3. Обновите приложение
   В главном меню отлаживаемого приложения выполнить `Файл \ SBT \ Обновить и переоткрыть текущую форму`
4. Повторите отлаживаемое действие.

3.9.3 Отладка выборки

Для отладки пользовательских интерфейсов предусмотрен специальный инструмент `Selection debug window`. Вызвать его можно в любом месте системы с помощью сочетания клавиш `ctrl+alt+shift+W`.

Глава 4

Языки разработки

Основными языками разработки являются

- scala
  Для программирования автономной и интерактивной бизнес логики.
- jexl
  Для программирования динамических скриптов.
- PL/pgSQL
  Для обработки данных в БД

4.1 Scala

Язык разработки используемый для программирования бизнес логики сервера приложения.

- Основы языка
- Гайд по коллекциям
  Может быть полезен, так как данные библиотеки очень удобны при манипуляциях со строками, к примеру операции сортировки и фильтрации
- Руководство scala: Типы коллекций
- Выбор последовательности

Прикладные разработчики в основном используют процедурную парадигму, что позволяет избежать сложного порога вхождения со стороны разработчиков `java`

При этом `scala` увеличивает производительность программистов за счет:

- Наличие удобной библиотеки коллекций
- Инлайн классов
  Позволяет использовать специализированную арифметику при работе с null типами, при этом не перегружая сборщик мусора

- Интерполяции строк
  Облегчает работу с sql запросами
- Трейтов
  Позволяют реализовать наследование отображений
- Прозрачного механизма инъекций зависимости

```
//При этом автоматически пробрасывается контекст сессии базы данных.
//Использование фабрик позволяет прозрачно перекрывать контроллеры бизнес логики
//в итоговом решении при необходимости
SomeApi().load(id)
```

- Фабрик контроллеров
  За счет того что обращения к любому контроллеру идет не статично а через фабрику(`SomeApi()`) возможно динамическое аспектирование(к примеру запись макросов) и проектное расширение кода(подмена базового контроллера его
  проектным наследником)

4.1.1 Обработка исключений

При обработки исключений нельзя перехватывать системные сообщения. Системные исключения всегда должны быть проброшены на верх. Это в том числе позволяет при необходимости прерывать сессию.

Для упрощения корректной обработки исключений введены вспомогательные функции и классы.

Объявление исключений

Прикладные исключения объявляются только как наследники от класса AppException или от потомков класса AppException.

При объявлении класса исключения нужно создать класс, и фабрику.

```
class ApiException(params:AnyRef) extends AppException(params)
object ApiException extends ExceptionFactory(new ApiException(_))
```

Выбрасывания исключения

Для выбрасывания исключения используется конструкция

```
throw AppException(args)
```

Внимание: Все прикладные исключения должны создаваться только от фабрики.

Перехват исключений

Перехватываться могут только прикладные исключения, если необходима принудительная очистка ресурсов в любом случае используйте секцию `finally`

Пример:

```
try App{
  println("start")
  throw AppException("Ups")
}catch {
  case e:AppException =>
    e.raise("Ups2")
}finally{
  println("end")
}
```

Внимание:

- Используйте `try app`
  Это позволит гарантировано получать `AppException`
- При необходимости продолжить выбрасывания сообщения используйте конструкцию `e.raise`
- При необходимости выбросить новый тип сообщения используйте конструкцию `e.raise(ApiException,"Ups2")`

Данная практика позволит избегать потери стека.

Конвертация java исключений в прикладные исключения

Все прикладные исключения являются наследниками от `AppException`.

При необходимости типизированной обработки java исключений в прикладном коде, необходимо выполнять конвертацию java исключений в прикладные исключения, что делается в функции:

```
ru.bitec.app.gtk.exception.App#apply
```

4.2 Jexl скрипты

Для выполнения динамического кода, существует jexl расширение.

Возможно выполнить jexl скрипт из:

- Ssh консоли

  ```
  login user/password@db
  jexl
  doSomeThing()
  /
  ```

- Api

```
JexlScript().eval(“doSomeThing”)
```

- Бизнес приложения, если у пользователя есть привелегии администратора
  Используйте пункт меню
  `Сервис > Инструменты > Выполнить jexl скрипт`

4.2.1 Обработка исключений в jexl

Jexl в ядре не поддерживает исключения, механизм исключений реализован через аннотации

пример:

```
@begin{
  logInfo("ok")
}@exception function(e){
  rollback()
} end
```

Для бросания исключения можно воспользоваться командой `raise`:

```
raise(“Текст исключения”);
```

4.2.2 Добавленные методы

- `lpad`
  Дополнение строки символами слева
- `rpad`
  Дополнение строки символами справа
- `flush()`
  Выполняет session.flush
- `commit()`
  Выполняет session. commit
- `rollback()`
  Выполняет session. rollback
- `raise`
  Бросает исключение AppException с указанным текстом
- `nvl`
  Возвращает первый параметр, если он не null, иначе второй параметр. Поддерживает nullable-типы
- `isNull`
  Проверка на null указанного параметра
- `isNotNull`
  Проверка, что параметр не null
- `foreachRop`
  Обход записей, возвращенных объектным запросом, например `byParent`
- `pgArrayToNLongList`
  Преобразует массив Long в список NLong

Примечание: Для просмотра списка поддерживаемых методов смотрите:\

`ru.bitec.app.gtk.jexl.session.JexlScriptContextExtension#ScopeBuilder`

4.2.3 Работа с датами

Методы работы с датами:

- `sysDate()`
  Текущая дата
- `truncDate`
  Дата на начало дня от переданной
- `truncYear`
  Дата на начало года от переданной
- `toDate`
  Переводит строку в дату по указанному формату. Если формат не указан, то используется стандартный. Стандартный формат `dd.MM.yyyy` и `dd.MM.yyyy HH:mm:ss`
- `toString`
  Переводит дату в строку по указанному формату. Если формат не указан, то используется стандартный. Стандартный формат `dd.MM.yyyy` и `dd.MM.yyyy HH:mm:ss`
- `+,-`
  Используются для изменения даты, операторы добавляют или отнимают от указанной даты количество дней. Например `sysDate() + 1` вернет дату на один день больше текущей

4.2.4 Работа с sql

Для работы с sql используется команды

- `sql(sqlText:String)`
  Для запросов на чтение
- `tsql(sqlText:String)`
  Для запросов на запись

Основные функции:

- `execute`
  Выполняет sql комманду
- `asList`
  Выполняет запрос, и возвращает список строк
- `asSingle`
  Возвращает запись
- `foreach`
  Принимает на вход функцию для потоковой обработки запроса

Пример

```
var l= sql("select 1 d").asList()
for(r:l){
  logInfo(r.d)
}
```

4.2.5 Работа с массивами объектов навигации (Rop)

Для обхода записей, возвращенных функцией byParent используется метод toJRops Основные функции:

- `asList`
  Возвращает список строк
- `asSingle`
  Возвращает запись
- `foreach`
  Принимает на вход функцию для потоковой обработки запроса

Пример

```
var l= toJRops(Btk_SomeEntityApi.byParent(rop)).asList()
for(r:l){
  logInfo(r.id);
  logInfo(r.sSystemName)
}
```

4.2.6 Работа в контексте выборки

Контекст выборки (JexlSelScript) – расширяет контекст бизнес-логики возможностью работы с выборками и пользовательским интерфейсом. Используется в тех случаях, когда необходимо получить данные из полей для пользовательского ввода. Пример методов, доступных из jexl-скрипта контекста выборки:

- `varExists`
  Проверка наличия переменной в выборке(если не найдено, пойдет поиск в мастер-выборках)
- `selfVarExists`
  Проверка наличия переменной только в текущей выборке
- `setVar`
  Установить значение переменной(если не найдено, пойдет поиск в мастер-выборках)
- `setSelfVar`
  Установить значение переменной только в текущей выборке
- `getVar`
  Получить значение переменной текущей выборки(если не найдено, пойдет поиск в мастер-выборках)
- `getSelfVar`
  Получить значение переменной только из текущей выборки
- `addVar`
  Добавить переменную в выборку с установкой значения

- newForm
  Создание форм

Пример:

```
  //Получаем значение из атрибута "id" выборки, приводим его к типу NString и пишем в␣
↪переменную idTree
  var idTree = getVar("id").asNString()

  //Получаем атрибут DGLOBALENDDATE из мастер-выборки
  var dEndDate = getVar("super$DGLOBALENDDATE").asNDate()

  //Вызываем метод из пакета с передачей параметров
  //Обратите внимание, для обращения к Api или пакетам используются их короткие␣
↪имена(без скобок)
  var fltCond = Act_UniversalReportPkg.getUniFilterCondByIdTree(idTree, dEndDate)

  //Вызывается ещё один метод, возвращающий объект scala-класса immutable.Map[NString,␣
↪Any]
  var filters = Act_UniversalReportPkg.getFilterValues(idTree)

  /*Определение Map-ы внутри jexl-скрипта. Отметим, что Map внутри jexl и объект scala-
↪класса
    Map (неважно mutable или immutable) - это разные объекты. Наиболее важным
    фактом является то, что передать scala-объект Map, полученный в предыдущей строке,
    в пакетный метод, принимающий scala-объект Map, в jexl-скрипте напрямую нельзя,
    именно поэтому приходится получать значения из переменной filters, перезаписывать
    их в jexl-Map param и потом передавать param в scala-метод.
  var param = {"flt_idDepOwner" : filters['flt_idDepOwner'],
               "flt_idAcc" : filters['flt_idAcc'],
               "flt_dFrom" : filters['flt_dFrom'],
               "flt_dTo" : filters['flt_dTo'],
               "flt_idAdjustMethod" : filters['flt_idAdjustMethod'],
               "flt_idAccCor" : filters['flt_idAccCor'],
               "uniFilterCondition_dz": fltCond}
  */
  //Открываем новую форму с переданными параметрами
  Act_TransAvi.defList().newForm().params(param).open()
```

4.2.7 Контекстная справка

В выборке выполнения jexl-скрипта есть возможность открытия выборки контекстной помощи, при нажатии на операцию «Помощь». В этой выборке в левой части поле ввода команд помощи, а права отображает справку.

Если методы справки вызывать из ssh-консоли, то результат будет напечатан в консоль.

Команды

- `help`
  Справка по глобальным методам;
- `listObj`
  Список доступных Api и Pkg
- `describe`
  Описание Api иPkg

4.3 PL/pgSQL

Язык обработки бизнес логики на стороне базы данных. Применяется в случаи если выполнение логики на сервере приложений не рационально в следствие высокой нагрузки на сеть или диск. Типичными примерами является:

- Выполнение агрегаций
- Построения аналитических запросов
- При фильтрации и сортировки больших данных

4.4 Требования к масштабируемости

При проектировании архитектуры приложения необходимо стремиться, чтобы горизонтальный рост системы не приводил к нелинейному увеличению потребностей в ресурсах.

Типичным узким местом в такой ситуации является увеличение запросов в базу данных при увеличении данных, когда код написан таким образом что при обработки бизнес логики на каждую строчку данных идет запрос в базу данных.

Типовыми вариантами сохранения масштабируемости методов являются:

- Работа через объектный кэш с возможностью его предварительной загрузки
  При этом кэш должен загружаться batch запросами, что значительно уменьшает нагрузку на базу
- Массовая обработка объектов
  При этом обработка разбивается на порции, каждая порция данных вытаскивается и сохраняется batch запросами

4.5 Общие принципы наименования

Global3-FrameWork является регистро-зависимым. Это означает что имена сущностей, свойств, методов и т.д. нужно указывать строго так, как они объявлены.

> **Предупреждение:** Btk_Class и BTK_Class это разные сущности с точки зрения фреймворка

При написании кода очень важно соблюдать общие правила, это позволяет улучшить взаимодействие между людьми и повысить эффективность.

Общие принципы по работе с кодом:

- Помните! Код чаще читается, чем пишется, поэтому не экономьте на понятности и чистоте кода ради скорости набора.
- Не используйте подчеркивание для отделения слов внутри идентификаторов. Подчеркивание используется только после имени модуля.
- Старайтесь не использовать сокращения лишний раз, помните о тех, кто читает код.
- Старайтесь делать имена идентификаторов как можно короче (но не в ущерб читабельности). Главное, чтобы смысл идентификатора был понятен в используемом контексте.
- Когда придумываете название для нового наименования, старайтесь не использовать имена, потенциально или явно конфликтующие со стандартными идентификаторами.
- Предпочтительно использовать имена, которые ясно и четко описывают предназначение и/или смысл сущности.
- Старайтесь использовать имена с простым написанием. Их легче читать и набирать. Избегайте (в разумных пределах) использования слов с двойными буквами, сложным чередованием согласных. Прежде, чем остановиться в выборе имени, убедитесь, что оно легко пишется и однозначно воспринимается на слух. Если оно с трудом читается, и вы ошибаетесь при его наборе, возможно, стоит выбрать другое.

4.5.1 Конвенция для разработки на scala

Стандартный код именуется в соответствии с руководством по языку

4.6 Сокращения

В основном сокращения использовать либо в горячих методах, либо в объектах базы данных, которые имеют ограничения.

Старайтесь не использовать аббревиатуры или неполные слова в идентификаторах, если только они не являются общепринятыми. Например, пишите getClass, а не getCls. (Исключением являются базовые понятия модуля, на которые имеет смысл придумать аббревиатуры, т.к. изначально сложно предсказать каким количеством коллекций они могут обладать, что может привести к вынужденным сокращениям в дальнейшем)

Не используйте акронимы, если они не общеприняты в области информационных технологий.

Широко распространенные акронимы используйте для замены длинных фраз. Например, UI вместо User Interface или Olap вместо On-line Analytical Processing.

Если имеется идентификатор длиной менее трех букв, являющийся сокращением, то его записывают заглавными буквами. Имена длиннее двух букв записывайте в стиле Паскаль Xml, xmlDocument.

Если, все-таки, сокращение необходимо из-за ограничений в длине наименования или по другим причинам, старайтесь сокращать наименее информативные части в имени и наиболее часто используемые (легко запомнить). К примеру, в таблице настроек есть поле idEnergyVolWithLostParamType -Тип показателя расхода электроэнергии с потерями. Что из этого можно сократить:

Energy - так как мы находимся в модуле энерго-учета, это поле обладает не большой информативностью и его можно сократить до En, по крайней мере, в контексте энерго-учета, легко вспомнить, что это электроэнергия.

VolWithLost — это основной смысл настройки, и его сокращать нельзя ни в коем случае

ParamType - тип параметра, так как в таблице настроек лежат в основном аналитики параметра учета, то на них можно придумать аббревиатуры. Название легко будет восстановить, помня о возможных аналитиках параметра учета.

В итоге, получаем сокращение idEnVolWithLostPT, с которым намного проще работать, чем с idEnergyVwltParamType

4.7 Scala типы для работы с данными

Для удобной обработки данных язык скала расширен null типами. Null типы, добавляют null логику к стандартным java типам, а так же расширяют их функциональность.

Задачи null типов:

1. Определить стандартные операции
2. Исключить `Null pointer exception`
3. Сделать арифметику более компактной

Для большей наглядности достаточно взглянуть на методы типов, к примеру для `NNumber`:

```
def round(value: NLong): NNumber = {
  if (this.isNull.isTrue || value.isNull.isTrue) this else NNumber(this.get.
↪setScale(value.get.intValue(), RoundingMode.HALF_UP))
}
```

Альтернативой null типам в scala является класс `Option`, однако в нем отсутствуют специализированные методы под каждый тип.

Пример использования:

```
//Null типы
val a1 = 1.nn
val b1 = None.nn
assert(
  (a1+b1).isNull
)
//Option
val a2 = Some(1)
val b2: Option[Int] = None
assert(
  (for(a<-a2;b<-b2) yield a+b).isEmpty
)
```

Любая колонка в таблице обрабатывается соответствующим ей null типом. При этом, происходит перегрузка стандартных операторов языка, создавая, таким образом, dsl расширение.

Для создания констант используются соответствующие фабрики:

4.7.1 NLong

Используется для работы с идентификаторами.

Фабрика: nl

4.7.2 NNumber

Используется для работы с числами

Фабрика: nn

4.7.3 NGid

Используется для работы с глобальными идентификаторами

Фабрика: ng

4.7.4 NDate

Используется для работы с датами.

Фабрика: nd

4.7.5 NString

Используется для работы со строками.

Фабрика: ns

4.7.6 NBigDecimal

Используется для работы с номерами. Внимание, обязательно используйте BigDecimal для выполнения расчетов. Использование чисел с двоичной арифметикой, может приводить к ошибкам.

Фабрика: nn

4.7.7 NDuration

Используется для арифметики дат, к примеру:

```
dvNewDate = dvDate + (10.hours+5.mins)
```

Фабрики: seconds, minutes, hours, second, minute, hour

4.8 Наименование переменных для nullable типов и атрибутов

Необходимо очень четко прослеживать, где работа идет с dsl, а где со стандартным типом. Наиболее яркий пример возможной, это операция сравнения.

```
If ( (idvSome1 === idvSome2 ) &&(nvNumber > 0.nn) {

}
```

В данном примере, если не использовать null логику, операция nvNumber `>0.nn` выдаст исключение, при nvNumber равном null. Поэтому, использование null логики требует ввести эквивалент операции сравнения.

Так как, невозможно перегрузить оператор `==`, если он возвращает другой тип или используется значимый класс. Создан стандартный оператор `===`.

Внимание: Очень легко перепутать `===` и `==` поэтому использование расширенных стандартов наименования для null типов обязательно.

Правила наименование переменной:

```
[Variable][a][t][Scope][Name][Suffix]
```

где:

- `[Variable]`
  Определяет тип
  - `n` - Число
  - `s` - Строка
  - `j` - Строка в формате json
  - `d` - Дата
  - `r`,`x` - Запись
  - `u`,`cur` - Курсор
  - `l`,`blob` - Бинарные данные (bytea, blob)
  - `c` - Символьные данные (text, clob)
  - `b` - Булево значение
  - `id`- Идентификатор
  - `gid` - Глобальный идентификатор
- `[a]`
  Определяет является ли переменная последовательностью
- `[t]`
  Определяет пользовательский тип
- `[Scope]`
  Область действия
  - `v` - Переменная процедуры

  - `g` - Переменная пакета (Как различать константы)
  - `p` - параметр процедуры

- `[Name]`
  Имя переменной
- `[Suffix]`
  Суффикс
  - `_dz` - Системные атрибуты
  - `_z` - Проектные атрибуты

4.8.1 Примеры задания параметров

- `dStart` - дата начала в таблице
- `dvStart` - дата начала переменной процедуры
- `dgMaxDate` - максимальная дата переменной пакета которую нельзя менять.
- `tvdaDate` - тип коллекции дат объявленный в процедуре
- `davDate` - коллекция дат объявленная в процедуре
- `uvStudents` - курсор объявленный в процедуре.

4.9 Наименования Scala-пакетов

Всегда в нижнем регистре.

Правильно:

- `ru.bitec.deepspace`
- `ru.bitec.deep_space`

Не правильно:

- `ru.bitec.deepSpace`

4.10 Правила наименования сущностей

4.10.1 Наименование классов

```
[Имя модуля]_[Аббревиатура логического пакета][Системное имясущности]
```

Так как, имя класса привязано к таблице, необходимо согласовывать наименование классов, с наименованием таблицы, что накладывает ограничение на имена.

4.10.2 Наименование пакетов

По аналогии с классами:

```
[Имя модуля]_[Аббревиатура логического пакета][Наименование сущности][pkg]
```

4.10.3 Наименование процедур и функций

Используйте глаголы или комбинацию глагола и существительных и прилагательных для имен методов.

Старайтесь избегать неоднозначных глаголов:

Не правильно:

- `checkItem` - проверить элемент
  Абсолютно не сообщает, что произойдет в результате отработки данного метода, толи будет выдано исключения, толи возращен результат,

Правильно:

- `validateItem` - гарантировать корректность элемента
  в случае несоответствия генерирует исключение
- `isItemValid` - возвращает истину если элемент корректен
- `itemErrCode` - Возвращает код ошибки
- `deleteItem` - удалить элемент
- `createOrder` - создать заказ

4.10.4 Наименование параметра

Из имени и типа параметра должны быть понятны его назначение и смысл.

Не усложняйте методы параметрами, которые, возможно, будут использоваться в будущих версиях.

Если в будущем понадобится новый параметр, можно использовать перегрузку методов и значения по умолчанию.

4.10.5 Наименование временных таблиц

```
[Имя модуля]_[Аббревиатура логического пакета][Системное имя][gtt]
```

4.10.6 Наименование выборок

```
[Имя модуля]_[Аббревиатура логического пакета][Системное имя]
```

Данный способ наименование позволяет, более легко выполнять рефакторинг, при перемещении выборки между пакетами.

4.10.7 Наименование отображений

```
[Системное имя]_[группа зависимости]
```

Где:

- `[группа зависимости]`
  Опциональный параметр в случаи если данные выборки имеют внешнии зависимости

Примеры зависимостей:

- idDoc
  Выборка отображает детализацию по документу
- TurnOver
  Детализация к оборотной ведомости

Не следует называть группу зависимости в несоответствии с тем, что реально происходит, так как это вводит в заблуждения, к примеру, название idDoc говорит о том, что можно использовать эту детализацию в любых выборках для детализации документа, однако, если запрос заточен под конкретную выборку мастера, это невозможно.

4.10.8 Формат комментария в систему контроля версий

```
format ::= (attention | label)? text
  attention ::= add | rem | fix
  label ::= (feat| test| doc | err ..)*
  text ::= Краткий текст комментария. dp (при наличии)
  dp ::= #id - Документ поддержки
```

4.11 Стандартные сокращения

4.11.1 Odm (Object domain model)

Доменная модель сущности. Используется для описания метаданных классов фреймворка: Обозначение, наименование, тип класса, атрибутный состав, ссылочность и т.д.

4.11.2 Orm (Object-Relational Mapper)

Объектно-реляционное представление сущности. Сопоставляет объектную доменную модель и реляционную базу данных. Используется в EclipseLink

4.11.3 Pojo

Java объект на сервере приложения. Хранит значения атрибутов объекта класса (строки таблицы БД) в оперативной памяти.

4.11.4 Dpi

Доменная автономная бизнес логика (создается кода генератором). Обеспечивает для сущности базовую обвязку методами фреймворка.

4.11.5 Dvi

Доменная интерактивная бизнес логика (создается кода генератором). Обеспечивает для сущности базовые методы работы с пользовательским интерфейсом.

4.11.6 Api

Прикладная автономная бизнес-логика по классу. Содержит методы прикладной обработки данных для конкретной сущности.

4.11.7 Pkg

Автономная бизнес логика. Содержит методы прикладной обработки данных. В отличии от api не привязана к какой либо конкретной сущности.

4.11.8 Avi

Прикладная интерактивная бизнес логика. Содержит методы работы с пользовательским интерфейсом.

4.11.9 Dvm (Domain view markup)

Доменная разметка выборки (создается автоматически). Шаблон базового представления сущности в системе.

4.11.10 Avm (Application view markup)

Прикладная разметка выборки. Описывает базовое представление сущности в системе.

4.11.11 Ata (Application table)

Прикладная таблица. Определение таблицы на языке Scala. Что позволяет использовать подсказчик кода, и статическую проверку при написании объектных запросов.

4.11.12 Aro (Application row)

Прикладная строка. Определяет правила взаимодействия с сущностью (pojo) в EclipseLink. Предоставляет адаптированный для фреймворка способ взаимодействия с объектами EclipseLink.

4.11.13 Rop (Row provider)

Провайдер строки. Используется для гарантированного доступа к строке (Aro) через кэш. При переходе к редактору rop проверяет наличие строки в кэше, и, если ее нет загружает строку из базы.

Глава 5

Сессия приложения

Сессия приложения создается на поток прикладной бизнес логики и предоставляет доступ к сессии базы данных, EclipseLink кэшу, серверу приложения. А так же содержит необходимые инъекции зависимости для работы прикладной бизнес логики.

Сессия приложения создается:

- на каждую mdi форму открытую в приложении
- на rest запрос к серверу приложения

Примечание: Для ускорение rest запросов возможна настройка пула сессий

Сессия приложения не привязана к сессии базы данных. Сессия базы данных выделяется на запрос либо на транзакцию.

Каждый контроллер прикладной бизнес логики содержит контекст сессии который представляет из себя фасад для быстрого доступа к сессии приложения и стандартным языковым примитивам фреймворка.

5.1 Сессия базы данных

Физическое соединение с базой.

Фреймворк использует механизм коротких транзакций в БД. Сессия БД выделяется для получения или изменения данных и возвращается в пул после окончания работы с данными.

Число прикладных сессий не равно числу соединений с БД.

Сессия базы данных выделяется из пула при:

- Начале транзакции
- Выполнении запроса из прикладного кода.

5.2 Рабочее пространство (workspace)

Рабочее пространство содержит транзакционный кэш чтения, а так же перечень изменений для отправки в базу данных.

Рабочее пространство создается при любом действии с базой данных. Чтение, создание\удаление\обновление. И удаляется при commit или rollback.

5.2.1 Точки сохранения

При создании точки сохранения, происходит сброс всех изменений из рабочего пространства в базу данных. При откате к точке сохранения, рабочее пространство удаляется.

5.3 Пользовательский сеанс

Пользовательский сеанс начинается в момент открытия формы в новой сессии, и заканчивается при закрытии формы.

Примечание: При открытии модального окна в той же сессии, пользовательские блокировки работают в сеансе дочерней формы.

Пользовательские сеансы можно просмотреть: `Меню приложения > помощь > блокировки`

5.4 Пользовательская блокировка

Пользовательская блокировка позволяет блокировать бизнес-объект вне транзакции базы данных. Такая блокировка работает в разрезе пользовательского сеанса. Элементы заблокированного бизнес-объекта не могут быть изменены каким-либо другим пользовательским сеансом.

Пользовательскую блокировку можно снять, при этом все заблокированные сессией объекты становятся доступны для блокирования.

Подробнее можно почитать здесь.

5.5 Логические блокировки

Логические блокировки являются транзакционными, и снимаются на commit или rollback.

Для получения блокировки используется метод `Btk_LogicLockPkg.request()`, в который передается имя блокировки.

Сессия при получении блокировки будет ожидать фиксацию или откат данных другой сессии, которая уже получила эту блокировку до этого.

5.6 Оптимистические блокировки

По умолчанию класс использует оптимистическую блокировку, для этого в таблице создается служебное поле `nVersion_dz`. В случае, если версия загруженного в транзакционный кэш перед изменением объекта отличается от версии в строке таблицы базы данных при сохранении изменений, возникает ошибка сохранения.

Преимущества оптимистической блокировки:

- Не требует начало транзакции базы данных

Недостатки оптимистической блокировки:

- Ошибка проявляется только при сохранении изменений в базу
  В момент возникновения ошибки, сложнее отловить ее источник.
- Возможно появления лишней нагрузки на базу данных
  От момента правки строки, до момента возникновение ошибки может пройти существенный набор действий который в итоге будет откачен.

5.7 Пессимистическая блокировка

Транзакционная блокировка с помощью которой можно заблокировать строку базы данных. При необходимости данная блокировка может быть выполнена с помощью:

- *ATSql запроса*

  Совет: Смотрите Явные блокировки

- *OQuery* с опцией `forUpdate`
- `Btk_Pkg().lockObjects`

Преимущества пессимистической блокировки

- Конфликт определяется в момент выполнения блокировки

Недостатки пессимистической блокировки:

- Требует транзакции базы данных

5.8 Бизнес логика

На каждый контроллер бизнес логики создается объект в контексте сессии приложения. Таким образом бизнес логика работает в контексте сессии что позволяет:

- Хранить переменные объекта в разрезе сессии
- Выполнять объектные запросы в базу данных с учетом сессионного кэша
- Работать в контексте транзакции сессии

5.8.1 Ключевые стратегии создания масштабируемого решения

1. Минимизируйте количество запросов к базе данных
   Для этого используйте:
   - bulk загрузку кэша
   - транзакционные индексы
   - разделяемые классы
   - кэширование запросов для разделяемых классов
2. Ограничивайте размер транзакции
   Для этого:
   - разбивайте бизнес логику на части
     К примеру разбейте массив бизнес объектов на порции для обработки.
   - Используйте `commitwork`
     Это позволяет сбросить данные в базу без завершения транзакции, при этом происходит очистка оперативной памяти.

5.8.2 Api

Содержит бизнес логику по обработке данных в таблице класса(сущности). Каждому классу соответствуют два файла с кодом бизнес логики данного класса:

- `Xxx_XxxxDpi`
  Содержит авто формируемый прикладной код. Данный код пересоздаётся системой формирования кода и не должен меняться разработчиками
- `Xxx_XxxxApi`
  Содержит прикладной код, изменяемый разработчиками

Пример API

```
class Bs_GdsCostDeviationTypeApi extends Bs_GdsCostDeviationTypeDpi[Bs_
↪GdsCostDeviationTypeAro, Bs_GdsCostDeviationTypeApi, Bs_GdsCostDeviationTypeAta] {
  override protected def entityAta: Bs_GdsCostDeviationTypeAta = Bs_
↪GdsCostDeviationTypeAta

  def register(spCode: NString
               , spCaption: NString
               , spDescription: NString): Unit = {
    val ropSelf = get(findByMnemoCode(spCode)).getOrElse {
      insert() :/ { rop =>
        setsCode(rop, spCode)

        rop
      }
    }

    setsCaption(ropSelf, spCaption)
    setsDescription(ropSelf, spDescription)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  }
}
```

5.8.3 Pkg

Пакеты создаются разработчиками по необходимости. Пакеты содержат методы бизнес-логики, которые не возможно отнести к какому-либо классу.

```
class Xxx_XxxxPkg extends Pkg{}
object Xxx_XxxxPkg extends PkgFactory[Xxx_XxxxPkg]
```

Глава 6

Взаимодействие с базой данных

6.1 Объектные запросы

Кроссплатформенные запросы, которые выполняются на уровне объектов класса.

При выполнении запроса идет обращение к базе данных, если по классу не настроено кэширование запросов.

Пример запроса

```
new OQuery(Bs_GoodsAta.Type) {
  where(t.sSystemName === spMnemoCode)
}
```

`OQuery` предоставляет подмножество JPQL и используется для выбора данных по классу объектный кэш.

6.1.1 Методы

- `where`
  Условие запроса.
- `orderBy`
  Выражение для сортировки результата
- `batchAll`
  Массовая загрузка объектов. Возвращает строки с прогруженными записями всех коллекций этого класса.
- `batchIn`
  Выполнение запроса с указанным перечнем коллекций, в которых требуется прогрузить данные.
- `forUpdate`
  Выполнение запроса с блокированием вернувшихся записей

- `forUpdateNoWait`
  Выполнение запроса с блокирование вернувшихся записей, без ожидания разблокирования, если записи уже заблокированы другой сессией.
- `tryCacheQueryResults`
  Попытаться закешировать результат запроса. Смотри пункт «Кеширование»
- `unique`
  Говорит, что запрос возвращает одну уникальную запись. Позволяет использовать cache-index'ы, указанные в orm класса

6.1.2 Объектный запрос с помощью выражений EclipseLink

Довольно полезный функционал, расширяющий использование объектных запросов. который позволяет запрашивать данные из заранее неизвестных классов (когда заранее неизвестен тип Ata). Класс:`ru.bitec.app.gtk.eclipse.query.ElExpOQuery`

Пример:

```
new ElExpOQuery(api).forWhere(
  _.get("sSystemName").equal("test".ns.get)
).addOrdering(
  _.get("sCaption")
)
```

смотри примеры использования: `ru.bitec.app.btk.ElExpOQueryTest`

6.1.3 Объектный запрос большого списка

```
in.ru.bitec.app.btk.Btk_QueryPkg#largeInQuery
```

смотри примеры использования: `ru.bitec.app.btk.Btk_QueryPkgTest`

6.1.4 Кеширование объектных запросов

Кеширование запросов работает только для классов с разделяемым режимом кеширования(`Shared`).

Кэширование по полю

Кеширование через cache-index'ы указанные в orm класса. Такой запрос должен возвращать одну строку и дополняется командой `unique()`. Например для атрибутов мнемокода класса в orm формируется запись:

```
<cache-index>
  <column-name>SSYSTEMNAME</column-name>
</cache-index>
```

Запрос выглядит следующим образом:

```
new OQuery(entityAta.Type) {
  unique()
  where(t.sSystemName*=== spMnemoCode)
}
```

Кэширование объектных запросов

Кэширование объектных запросов возможно по требованию, в случае, если класс настроен для сохранения в разделяемом кэше.

Чтобы включить кэширования запроса:

1. Добавьте в запрос опцию `tryCacheQueryResults()`.
   Результат такого запроса будет кэширован, если транзакция не находится в режиме редактирования разделяемых объектов.

Пример запроса:

```
new OQuery(entityAta.Type) {
  tryCacheQueryResults ()
  where(t.sSystemName === spMnemoCode)
}
```

Внимание: Объектный запрос не отображает удаленные строки из текущего рабочего пространства, однако он не способен отслеживать изменения в `where` условии. Если в текущем рабочем пространстве был изменен мнемокод, то объектный запрос вернет данные без учета этих изменений.

Для гарантированного получения согласованных изменений в объектном запросе необходимо вызывать перед ним `flush`.

6.2 Транзакционный индекс

Позволяет получить, согласованный с изменениями, перечень строк из базы данных по значению индексируемого атрибута.

Индекс подгружает данные из базы данных по мере обращения к ключам индекса, а так же отслеживает изменения в текущем рабочем пространстве, для получение согласованного набора строк. Это позволяет получить актуальный список строк и значение их атрибутов, даже когда изменения в кэше не сброшены в базу данных.

Примечание: В отличии от объектного запроса транзакционный индекс видит любые изменения в рабочем пространстве по индексируемому полю.

Пример объявления:

```
lazy val idxidParent = TxIndex(Btk_GroupAta.Type)(_.idParentGroup)
```

Методы

- `byKey`
  Возвращает итератор по ключу индекса.
- `refreshByKey`
  Возвращает итератор по ключу индекса с обновлением из базы данных.
- `queryKeys`
  Кеширование ключей индекса.
- `forPartition`
  Открывает секцию для массового обновления индекса. Используется для прозрачного массового обновления после очистки транзакционного кэша. Секции могут быть вложенными друг в друга, в таком случае ключи суммируются.

6.3 Реляционные запросы

Для обработки реляционных запросов в основном используется методы на базе anorm.

Для более удобного использования в контекст бизнес логики добавлены дополнительные функции.

Внимание: Реляционные запросы полностью игнорируют изменения в рабочем пространстве. Если требуется выполнить согласованный запрос используйте метод `session.flush()`

6.3.1 ASQL

Выполнение запроса на чтение

6.3.2 ATSQL

Выполнение запроса с изменением данных или блокировками

6.3.3 ASelect

Выполнение запросов на чтение\запись с большим кол-вом колонок.

Пример:

```
for (rv <- new ASelect {
  val nParentLevel = asInt("nParentLevel")
  val gidParent = asString("gidParent")
  val gidChild = asString("gidChild")
  val idParent = asLong("idParent")
  SQL"""
  select nParentLevel,gidParent,gidChild,idParent,idChild from table
  """
}) {
  println(rv.nParentLevel())
  //запрос поля без его предварительного объявления
  println(rv.get("idChild").asNLong())
}
```

6.3.4 Типичные ошибки

Некорректное использование запросов на чтение

Внимание: Даже если такой код работает в тестовом случае, это будет приводить к ошибкам в продакшене.

Пример неверного запроса:

```
ASQL"select * from table where id=$id for no key update".execute()
```

Так как данный запрос не регистрирует транзакцию, он может быть выполнен в режиме автокоммита, что фактически делает его бесмысленным, так как блокировка снимается сразу же после выполнения запроса.

```
ASQL"update table set a=$v where id=$c".execute()
```

Данный запрос может быть выполнен в текущей транзакции, а может быть не выполнен в транзакции на чтение, поэтом становиться абсолютно не предсказуемое поведение на откат, и контроль ссылочной. Что легко может наводить ошибки на бизнес логику.

Внимание: Внимание, все запросы на изменения данных должны быть транзакционными:

```
ATSQL"select * from table where id=$id for no key update".execute()

ATSQL"update table set a=$v where id=$c".execute()
```

Если вам необходимо выполнить запрос в отдельной транзакции, всегда создавайте атомарные транзакции, помните, даже если в вашем тестовом случаи транзакция не начата, это не значит что она не может быть начата в другом контексте.

Некорректное использование блокировок

```
ATSQL"select * from reference where id=$id for update".execute()
```

Данная блокировка заблокирует работу всех документов по справочнику, так как блокируется не только запись, но и все внешние ключи.

Всегда пишите `for no key update`, если у вас нет четкого обоснования необходимости иной блокировки.

Некорректное формирование запроса

Будьте осторожнее с подставлением значений в строку (интерполяции) при формировании sql запроса.

```
ATSQL(s"select * from reference where gid=$gid for update")
ASQL(s"select * from reference where gid=$gid for update")
new ASelect {
  //...
  SQL(s"select * from reference where gid=$gid for update")
}
```

В данном примере значение gid неправильно подставится в sql выражение т.к. технически подстановка идет в обычную строку никто не догадается правильно конвертировать данные в правильную для sql форму

```
ATSQL(s"select * from reference where gid={gid} for update").on(Symbol("gid") -> gid)
ASQL(s"select * from reference where gid={gid} for update").on(Symbol("gid") -> gid)
new ASelect {
  //...
  SQL(s"select * from reference where gid={gid} for update")
    on(Symbol("gid") -> gid/*, другие параметры*/)
}
```

Для случаев когда вам нужно собрать запрос из обычной строки используйте замену символов `on(Symbol("символ для замены") -> ваша переменная, ...)`

```
ATSQL"select * from reference where gid=$gid for update"
ASQL"select * from reference where gid=$gid for update"
new ASelect {
  //...
  SQL"select * from reference where gid=$gid for update"
}
```

Тут все будет правильно подставлено

6.4 Работа с большими данными

Postgresql не поддерживает bulk insert\select\update, более того коммит закроет любой открытый курсор. Это делает невозможным обрабатывать большие объемы данных частями на сервере. Так как очень длинные транзакции перегревают wal и в конечном счете сильно роняют общую производительность сервера. Необходимо выполнять партиционирование на сервере приложения.

Замеры производительности показывают, наиболее производительным средством перегонки данных является sql комманда postgresql «copy».

Сравнительная таблица производительности:

Операция 100000 записей	Приблизительное время(ms) на сервере
Вставка записей одним sql в базе	214
Общее время запроса\вставки через copy	728
Вставка записей одним sql в базе при наличии индексов	2785
Общее время вставки через массовый flush при наличии индексов из сервера в базу	19562
Обще время вставки через flush по одной записи при наличии индексов из сервера в базу	156710
Запрос данных через ResultSet	396
Запрос данных через команду copy	330

Примечание: На десктопе разница между вставкой через copy и вставкой прямым sql может достигать различий в 10 раз. Финальные тесты производительности нужно замерять исключительно на серверном оборудовании.

Вставка через flush по одно записи может работать в 1000 раз медленнее, особенно это актуально для данных с аудитом.

Вставка через batch на чистом sql сравнима по порядку с copy хоть и медленнее.

Массовая вставка на чистом sql должна быть раза в 4 быстрее за счет отсутствия затрат на кэша.

Таким образом наиболее высокую производительность при массовой работе с данными может обеспечивают следующие алгоритмы:

6.4.1 Массовая вставка

Через команду copy

6.4.2 Массовый update\delete

- Сохранение данных в файл через команду copy
- Для каждой пачки
  - Начало транзакции
  - Копирование данных в gtt
  - Выполнение dml операции
  - Завершение транзакции

Внимание: Транзакция не должна завершаться до конца обработки пачки, иначе это приведет к потере данных в gtt(при завершении транзакции сессия отпускается)

6.4.3 Размер транзакции

Постгресс по своей архитектуре менее зависим от длинны транзакции(смотри наполнение базы данных). Таким образом дробление нет потребности в дроблении на мелкие транзакции.

Однако слишком длинные транзакции негативно влияют на базу, происходят следующие негативные эффекты:

- блокировка данных
- блокируется сборщик мусора, что может привести к падению производительности запросов на dead row,и к распуханию файлов

Таким образом на сервисных операциях, если они не превышают несколько гигабайтов(зависит от базы), разбиением транзакций можно пренебречь.

На оперативных транзакциях, желательно чтобы они не занимали больше 10 минут(зависит от частоты транзакций и нагрузки на базу) и не вызывали блокировок.

Часть II

Классы

Глава 7

Класс

Определяет правила хранения и обработки таблицы базы данных.

Класс позволяет существенно ускорить разработку бизнес логики ориентированную на работу с данными. Программисту достаточно объявить перечень атрибутов класса чтобы за счет кодо-генерации получить набор готовых сервисов.

Перечень генерируемых элементов:

- *Доменная автономная бизнес логика*(`Dpi`)
  Содержит код для автономной бизнес логики
- *Каркас прикладной автономной логики*(`Api`)
  `scala` класс с окончанием `Api`, в котором пишется автономная бизнес логика для работы с классом. Наследуется от `Dpi`
- *Доменная интерактивная бизнес логика*(`Dvi`)
- *Каркас прикладной интерактивной логики*(`Avi`)
  `scala` класс с окончанием `Avi`, в котором пишется интерактивная бизнес логика. Наследуется от `Dvi`
- *Доменная разметка выборки*(`dvm.xml`)
  Содержит сгенерированную по умолчанию разметку выборки.
- *Каркас прикладной декларации пользовательского интерфейса*(`Avm`)
  `xml` файл с расширением `avm.xml`, в котором пишется разметка выборки
- Интеграция с `Orm`
  - `Pojo` объект для хранения данных в кэше
  - `Aro` объект интеграции pojo в фреймворк

Примечание: При кодо-генерации обычно создаются два элемента разных типов:

- `D` - Domain
  Доменный элемент всегда перезаписывается при кодо-генерации и содержит бизнес логику для подключения сервисов.

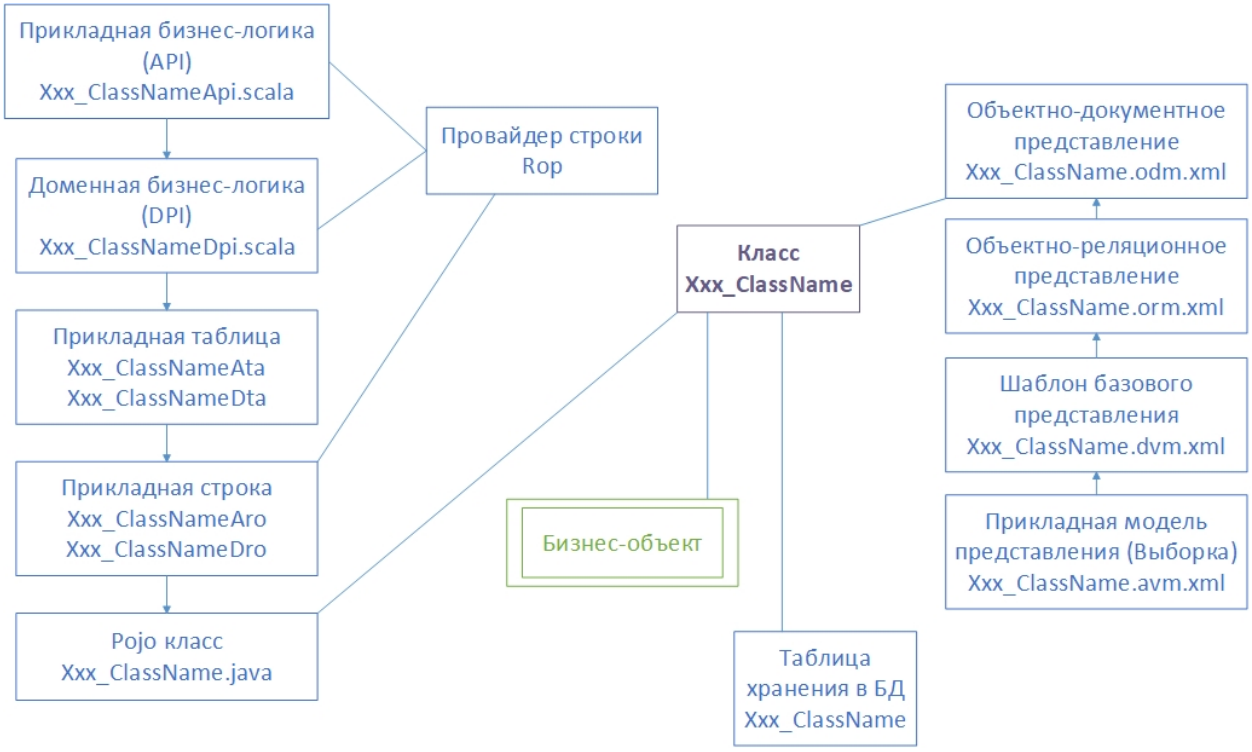
- `A` - Application
  Прикладной элемент не изменяется при кодо-генерации и служит для хранения бизнес логики написанной программистом вручную. Прикладной элемент наследуется от доменного элемента.

Доступные прикладные сервисы:

- Аудит
  Аудит предназначен для фиксации различных событий при работе пользователей в системе
- Администрирование
  Позволяет управлять доступом к автономной и интерактивной бизнес логике за счет выдачи прав на привилегии.
- Универсальная фильтрация
  Позволяет пользователю строить комплексную фильтрацию списка классов на уровне базы данных. В универсальном фильтре можно использовать как поля самого класса и его коллекций так и поля классов на которые есть ссылки.
- Авто нумерация
  Механизм выдачи номеров объектам класса, данный механизм позволяет переопределять алгоритм выдачи номеров на проекте.
- Копирование объектов
  Кода генерация бизнес логики для копирование объектов
- Группировка
  Группировка используется для систематизации хранения объектов и удобства восприятия пользователем, так же группировка позволяет массово управлять характеристиками и настройками объектов класса.
- Сервис прикрепленных файлов
  Позволяет прикреплять к объектам класса произвольные файлы
- Поиск по шаблону
  возможность поиска объектов класса по частичному или полному совпадению введенного текса со значениями полей объекта или его заголовка и мнемокода
- Объектные характеристики
  Возможность добавлять произвольные поля в класс на проекте
- Генерация штрих-кодов объекта Механизм генерации штрих кодов объектам класса при создании
- Подписи объектов для печати
  Используется при печати отчетов. Позволяет формировать в печатной форме список лиц с местом для подписи
- Полнотекстовый поиск
  Возможность класса осуществлять быстрый поиск по значению атрибутов класса

7.1 Схема окружения

Окружение класса создается в момент кода генерации:



7.2 Общие сведения о классах

Класс сущности(далее просто класс) определяет хранилище совокупности объектов(строк), имеющих одинаковые характеристики, подчиняющихся общим настройкам и операциям, функционирующих в рамках единой логики.

Класс содержит набор атрибутов, атрибут может быть:

- Простым значением
  Используется для ввода и хранения значения определенного типа данных
- Ссылочным
  Используется для выбора значения атрибута из справочника или другого множества
- Переменной ссылочности
  Используется для ссылочности на любую таблицу в системе
- Ссылочным на класс
  Таблица классов является служебной таблицей мета данных хранящей весь перечень классов в решении.

Класс должен иметь уникальное системное имя и наименование.

Правила наименования класса:

- системное имя должно задаваться на латинице
- Имя должно быть в формате `{Module}_{Name}` где:

  - `Module` - Имя модуля
  - `Name` - Имя класса
    Имя класса должно быть в единственном числе, именительном падеже

Пример:

- `Lbr_Book`

7.3 Типы данных

В система имеет специализированный набот простых и объектных типов для удобной обработки данных. Объектные типы при необходимости интегрированны в контекст сессии что позволяет обеспечить высокую произоводительность системы за счет минимизации операций сериализации\десириализации.

Основные типы данных, используемые для атрибутов класса:

- Целое число
- Дробное число
- Строка
- Дата
- Ссылка (на объект заданного класса)
- Ссылка на класс
- Переменная ссылка (на объект произвольного класса)
- Глобальный идентификатор gid
- Json контейнер

7.3.1 Простые типы

Простыми типами являются: Число, строка, дата

Тип	postgresql	odm	Рекомендация по использованию
Строка фиксированной длинны	varchar	Varchar	Текст до 4000 символов
Строка переменной длинны	text	Text	Текст до 15 мегабайт
Число	number	Number	Целое или дробное число
Дата	date	Date	Дата, дата и время

7.3.2 Ссылочные типы

Данные классов могут быть связаны между собой. Для организации связей между классами используются специальные типы.

Ссылка и переменная ссылка

Ссылка на объект и переменная ссылка используются для организации ссылочности объектов одного класса на объекты другого класса (или объекты своего собственного класса).

Тип `ссылка` может хранить объекты одного класса, который задан в настройках.

Тип `переменная ссылка` хранит ссылку на объект любого класса.

Ссылка на класс

Хранит ссылку на класс. Обычно используется совместно с типом `переменная ссылка` для хранения класса, объект которого содержится в переменной ссылке.

Глобальный идентификатор gid

Gid является уникальным идентификатором в рамках системы. Переменная ссылка на gid является альтернативой системе переменной ссылочности из двух атрибутов (ссылка на класс + переменная ссылка на объект). Для организации переменной ссылки через gid используется один атрибут.

Формат ссылки gid

Глобальный идентификатор состоит из строки:

```
gid :== idClass \ id[@ idNode]
```

Где:

- idClass – идентификатор класса
  Ссылка на объект в таблице btk_class
- id – идентификатор
- idNode – идентификатор удаленного нода
  В текущей базе, задается для объекта созданного на удаленном узле (если используется механизм репликации на уровне системы)

7.3.3 Json контейнер

Json контейнер – это расширение объекта класса NoSQL нотацией в реляционной СУБД. Контейнер не имеет жесткой, заранее определенной схемы и основан на множестве пар «ключ-значение». Это позволяет использовать его как динамическое расширение объекта. Для добавления новых данных в контейнер не требуется перекомпиляция кода или изменение структуры СУБД.

7.4 Супертипы классов

Для проектирования существует несколько разновидностей класса, с помощью которых можно реализовать модель с необходимой бизнес-логикой. Эти разновидности класса называют супертипами.

Супертипы задают стандартное поведение для класса, а также задают стандартную обвязку методами.

Основные супертипы:

- `reference` - Справочник
  Справочник — это прикладной объект, который позволяет хранить данные, имеющие одинаковую структуру и списочный характер. Пример: Справочник физ. Лиц; Места хранения; справочник ТМЦ.
- `document` - Документ
  Документ – это прикладной объект, который хранит данные о событиях или операциях на предприятии. Пример: Заявка на отгрузку; Приходная накладная; Акт сверки. Документ обычно имеет атрибут состояние, который отражает его жизненный цикл.
- `collection` - Коллекция
  Коллекции представляют собой классы, объекты которых не имеют права на самостоятельное существование и могут быть созданы только для объектов других классов. Коллекции применяются в качестве табличных частей документов или логических развязок между классами. Добавление коллекций в бизнес объект позволяет массово загружать данные в объектный кэш, что минимизирует нагрузку на базу данных. Так же возможно обход элементов коллекции по родителю без транзакционного индекса, что уменьшает нагрузку на процессор.
- `vcollection` – Переменная коллекция
  Переменная коллекция расширяет возможности обычных коллекций и может ссылается на родителя переменной ссылкой. Это требуется, когда для нескольких классов используется одинаковая коллекция.
- `journal` - Журнал
  Журнал – это особый тип классов, приспособленный для хранения большого количества записей. Такие классы имеют ограниченную функциональную обвязку ядровыми методами фреймворка. Это позволяет увеличить быстродействие при работе с журналом. Примеры: записи по потребности ТМЦ на заказ в разрезе документов; журнал трудоемкости в разрезе операций и т.д.
- `trait` – Трейт
  Абстрактный класс-предок, не имеющий собственной структуры хранения. Такой класс содержит общую логику нескольких классов-потомков и является частью механизма повторного использования кода.
- `mixin` – Миксин (класс-примесь)
  Миксин – это особый вид классов, которые служат для хранения данных из разных классов. Используются для построения общих списочных форм различных диалогов подбора в пользовательских интерфейсах, а также для удобства обработки данных в прикладной бизнес-логике. Миксин позволяет объединить несколько разных таблиц вместе что дает возможность использовать внешние ключи и индексы на данное объединение.

7.5 Бизнес-объект

Бизнес-объект (БО) - объединение нескольких классов и их коллекций в группу для более удобного манипулирования ими при работе с кэшем и конфигурировании вспомогательных сервисов.

Бизнес объект позволяет:

- Массово загружать данные в транзакционный кэш
  Для бизнес объекта можно указать стратегию загрузки данных существенно уменьшающую количество запросов в базу данных. Так как запросы пойдут не по каждому объекту а по каждому классу бизнес объекта.
- Настраивать права доступа
  По бизнес объекту создается административный объект на котором можно массово выдать привилегии для всех классов бизнес объекта
- Управлять электронной подписью
  Можно настроить правила подписи всего бизнес объекта включая не только шапку но и все вложенные коллекции.
- Настраивать интеграцию и репликацию

7.5.1 Навигация по бизнес-объекту

Навигацией является последовательное посещение элементов бизнес-объекта сверху вниз. При навигации перемещение между объектами идет по кэшу, при этом обеспечивается автоматическая дозагрузка объектов в кэш по необходимости.

В процессе навигации объекты не блокируются и могут при необходимости быть вытолкнуты из кэша, что вызовет автоматическую дозагрузку (обновление).

Примеры навигации:

```
val empApi = EmployeeApi()
empApi.load(7452) :/ { id =>
  println(id.id)
  for(idDes <- AddressApi.byParent(id)){
    println(s"address: city=${ idDes.get(_.city)}")
  }
}
```

7.6 Навигация в рабочем пространстве

Объекты загружаются из базы или из кросс – сессионного кэша, при загрузке происходит пессимистическая либо оптимистическая блокировка. Объекты находятся в рабочем пространстве до момента коммита. В момент коммита рабочее пространство очищается.

При навигации можно производить модификацию объектов. Api гарантирует корректную навигацию по мастер деталям без необходимости flush и clean кэша или немедленной загрузки коллекций.

7.7 Массовая загрузка объектов.

При массовой загрузке объектов. Происходит минимизация обращений к базе данных. То есть при обходе в обычном режиме 3-х уровневого бизнес-объекта произойдет n+2 запросов, где n количество деталей 2-го уровня, 1 запрос, на запрос мастер объекта, 1 запрос на запрос коллекций 2-го уровня. Однако если объект запросить с помощью массового запроса, то при его обходе произойдет всего 3 запроса. Что может ускорить навигацию по объектам более чем в 10 раз.

Пример массового запроса:

```
for (rv <- new OQuery(Stk_WarrantAta.Type){
  where (t.id in idap)
  batchAll()
}){}
```

Примечание: Объектные запросы активно расходуют оперативную память. Это накладывает ограничение на использование их в процедурах бизнес-логики. Обычно объектные запросы используются для организации пользовательского интерфейса (редактирование одного объекта с коллекциями. Справочники, документы и т.д.), а для программирования внутренних процедур бизнес-логики используются SQL запросы в БД.

7.8 Работа с провайдерами строк

Провайдер строки - Rop используется для работы со строкой данных (Aro), загруженных в рабочее пространство, обеспечивая гарантию того, что при доступе к строке данная строка будет находиться в рабочем пространстве.

Метод получения rop:

```
thisApi().load(идентификатор.asNLong)
```

Примеры сеттеров в файле выборки:

```
val rop = thisRop
thisApi().setidContras(rop,getVar("super$id").asNLong)
```

Работа с rop в API:

```
for (ropGrade <- new OQuery(entityAta.Type){
  where (t.idGdsGrade === idpGdsGrade)
}) {
  setidGdsGrade(ropGrade, None.nl)
}
```

Работа с AnyRop(rop неизвестного типа):

```
anyRop match {
  case Btk_GroupApi(ropGroup) =>
    ropGroup.get(_.sCaption)
  case Btk_ClassApi(ropClass) =>
    ropClass.get(_.sName)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  case _ => throw AppException("Ожидали роп группы или класса")
}

// получить из списка только ропы определенного класса
ropaAny
  .collect {
    case Btk_GroupApi(ropGroup) => ropGroup
  }
```

7.9 Оптимистическая блокировка

Так как система Global Postgres ориентирована на работу с короткими транзакциями, фреймворк по умолчанию включает для классов оптимистическую блокировку.

Принцип работы оптимистической блокировки:

- При загрузке строки в кэш, запоминается версия изменения
- Если строка изменяется, увеличивается версия изменения.
- При сохранении в базу происходит проверка того, что старая версия изменения соответствует версии изменения в базе данных, если это не так, то выдается ошибка применения изменений.

Для отключения оптимистической блокировки в классе необходимо указать свойство:

```
<class useOptimisticLocking="false"/>
```

Примечание: Для хранения версии изменений используется служебное поле `nVersion_dz`

7.10 Коллекции

Коллекцией является сущность, объекты которой не могут существовать без ссылки на объект владелец. Классы коллекций объявляются в Odm сущности-владельце.

В бизнес-объекте предполагается использование ленивых коллекций. Элементы коллекции загружаются по необходимости, удаление добавление происходит в фоновых коллекциях и не требует немедленного запроса к базе данных.

Связывание сущностей владельца и коллекции производится путём объявления элемента `collection` в секции `collections`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
      name="Bs_Brigade" caption="Бригада"
      cardEditor.representation="Card"
      listEditor.representation="List"
      viewOptions.openCardType="mdi"
      supertype="reference">
        <attributes>
                <attr name="id"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        attribute-type="Long"
        caption="Идентификатор"
        order="-1"
        type="basic"
        isVisible="false"/>
              <attr name="idClass"
        attribute-type="Long"
        caption="idClass"
        order="-2"
        type="basic"
        isVisible="false"/>
              <attr name="gid"
        attribute-type="Varchar"
        isVisible="false"/>
              <attr name="sCode"
        attribute-type="Varchar"
        caption="Код"
        order="10"
        isMnemoCode="true"
        type="basic"
        isVisible="true">
                      <autonum id="1">
                              <dimension name="Dim1"/>
                      </autonum>
              </attr>
              <attr name="sCaption"
        attribute-type="Varchar"
        caption="Наименование"
        order="20"
        isHeadLine="true"
        type="basic"
        isVisible="true"/>
              <attr name="idDepartment"
        attribute-type="Long"
        caption="Подразделение"
        order="30"
        type="refObject"
        ref.class="Bs_Department"/>
              <attr name="idForeman"
        attribute-type="Long"
        caption="Бригадир"
        order="40"
        type="refObject"
        ref.class="Bs_Employee"/>
              <attr name="idMaster"
        attribute-type="Long"
        caption="Мастер"
        order="50"
        type="refObject"
        ref.class="Bs_Employee"/>
      </attributes>
      <collections>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                <collection name="Bs_BrigadeStaff"
                ref.attr="idBrigade"
                cascadeOnDelete="true"/>
        </collections>
        <dbData>
                <script name="dataInstall" version="1">
                        <install>Bs_BrigadeApi.dataInstall()</install>
                </script>
        </dbData>
</class>
```

Для класса коллекции указывается супертип «collection», это гарантирует, наследование scala-классов от необходимых системных классов.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
       name="Bs_BrigadeStaff"
       caption="Состав бригады"
       cardEditor.representation="Card"
       listEditor.representation="List"
       viewOptions.openCardType="mdi"
       supertype="collection">
        <attributes>
                <attr name="id"
                      attribute-type="Long"
                      caption="Идентификатор"
                      order="-1"
                      type="basic"
                      isVisible="false"/>
                <attr name="idClass"
                      attribute-type="Long"
                      caption="idClass"
                      order="-2"
                      type="basic"
                      isVisible="false"/>
                <attr name="gid"
                      attribute-type="Varchar"
                      isVisible="false"/>
                <attr name="idEmployee"
                      attribute-type="Long"
                      caption="Сотрудник"
                      order="10"
                      type="refObject"
                      ref.class="Bs_Employee"
                      isMnemoCode="true"/>
                <attr name="idBrigade"
                      attribute-type="Long"
                      caption="Бригада"
                      order="20"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                    type="refObject"
                    ref.class="Bs_Brigade"
                    genListCollectionRep="true"/>
              <attr name="dStart"
                    attribute-type="Date"
                    caption="Дата начала"
                    order="30"
                    type="basic"
                    editorType="datePick"/>
              <attr name="dEnd"
                    attribute-type="Date"
                    caption="Дата окончания"
                    order="40"
                    type="basic"
                    editorType="datePick"/>
              <attr name="bisForeman"
                    attribute-type="Number"
                    caption="Является бригадиром"
                    order="50"
                    type="basic"
                    editorType="check"/>
        </attributes>
</class>
```

7.10.1 Формирование кода

При формировании кода сущности-владельца, производится пересоздание кода всех коллекций. Dpi коллекции будет унаследован от ChildApi.

```
trait Xxx_XxxxDpi[T] extends ChildApi[java.lang.Long, ARO, API]
```

Dvi будет унаследован от CollectionAvi.

```
trait Xxx_XxxxDvi extends CollectionAvi
```

Основные методы для работы:

```
//создание по владельцу, возвращает rop созданного объекта
Api().insertByParent(ropMaster)
//загрузка всей коллекции по владельцу, возвращает обходчик записей отфильтрованных по␣
↪rop предка
Api().byParent(ropMaster)
Api().byParent(idMaster)
//удаление по rop объекта
Api().delete(rop)
```

7.10.2 Отображения-детали

Для формирования отображений выборки, данные которых ограничены по значению ссылочного поля, необходимо в odm-файле, для соответствующего ссылочного атрибута указать свойство genListCollectionRep=»true». Будет сформировано отображение List_{attr}.

```
<attr name="idBrigade"
      attribute-type="Long"
      caption="Бригада"
      order="20"
      type="refObject"
      ref.class="Bs_Brigade"
      genListCollectionRep="true"/>
```

7.10.3 Переменная ссылочность

Часто при проектировании бизнес-логики необходимо хранить в атрибуте коллекции ссылки на объекты нескольких классов. В этом случае необходимо хранить не только идентификатор ссылочного объекта, но и идентификатор ссылочного класса.

Для реализации переменной ссылочности, в Odm коллекции необходимо создать атрибут способный ссылаться на поле «gid» мастер-объекта.

```
<attr name="gidSrcXxxxxx"
      attribute-type="String"
      type="refAnyObject"
      genListCollectionRep="true"/>
```

Примечание: У классов, на объекты которых может ссылаться коллекция с переменной ссылочностью обязательно должен существовать атрибут «gid». Рекомендуется индексировать данное поле.

7.10.4 Каскадное удаление

Если у коллекции включено свойство «Каскадное удаление» (по умолчанию включено),

```
<collections>
  <collection cascadeOnDelete="true"
              name="Bs_PersonIdentityDoc"
              ref.attr="idPerson"/>
  <collection cascadeOnDelete="true"
              name="Bs_PersonProf"
              ref.attr="idPerson"/>
</collections>
```

то в dpi мастера формируется вызов метода delete из коллекций.

Для подключенных `vcollection` (переменных коллекций) код удаления необходимо писать вручную в api:

```
override def delete(rop: ApiRop): Unit = {
  for (crop <- Bs_BankAccApi().byParent(rop)) {
    Bs_BankAccApi().delete(crop)
  }
  for (crop <- Bs_DefSettlerAddressApi().byParent(rop)) {
    Bs_DefSettlerAddressApi().delete(crop)
  }
  for (crop <- Bs_SettlerAddressApi().byParent(rop)) {
    Bs_SettlerAddressApi().delete(crop)
  }
  super.delete(rop)
}
```

7.11 Механизмы наследования

Во фреймворке нет полноценного классического наследования классов сущностей. Понятие «наследование» используется в качестве описания логической связи двух классов.

Вместо классического наследования фреймворк использует миксины и трейты.

7.11.1 Миксин (Mixin)

Примесь (англ. Mix-in) — элемент языка программирования (обычно класс или модуль), реализующий какое-либо чётко выделенное поведение. Используется для уточнения поведения других классов, не предназначен для порождения самостоятельно используемых объектов.

В системе Global3 Postgres миксин – это специализированный класс, каждый объект (запись в таблице) которого соответствует одному объекту подключенного класса (записи в таблице). Mixin-класс может иметь не ограниченное число подключенных классов. Первичным ключом для mixin-объектов является поле gidRef. Значение поля gidRef равно значению поля gid объекта подключенного класса и заполняется в момент создания объектов.

В Api миксина реализуются общие методы, которые могут использоваться для всех подключенных классов.

Миксин может содержать произвольное число атрибутов. Атрибуты, имена которых совпадают с именами атрибутов подключенных классов (наследников), будут автоматически заполняться при установке значений в атрибуты.

Примечание: Все scala-классы, соответствующие миксину наследуются от системных классов D-ветви (имеют префикс «D»). Остальные scala-классы, соответствующие обычным сущностям, наследуются от системных классов S-ветви (имеют префикс «S»)

Для миксинов метод получения rop по `gidRef` называется `loadByGid`, а также реализован метод аналогичный методу get для SApi

```
getByGid(gidpRef: NGid): Option[ApiRop]
```

Создание mixin-класса

В Odm файле необходимо объявить атрибут `gidRef`.

```
<attr name="gidRef" attribute-type="Varchar"/>
```

У класса указать свойство «supertype=”mixin”».

```
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
      name="Xxx_ClassName" supertype="mixin"/>
```

Формирование кода

При формировании кода для mixin-класса, Dpi будет унаследован от `MixinApi`.

```
trait Xxx_XxxxDpi[T] extends MixinApi[java.lang.Long, ARO, API]
```

Dvi будет унаследован от `AppMixinAvi`.

```
trait Xxx_XxxDvi extends AppMixinAvi
```

Основные методы для работы:

```
//создание по гор мастера, возвращает гор миксина
Api().insertByParent(ropMaster)
//загрузка миксина по gidRef, возвращает гор миксина
Api().loadByGid(gidRef)
//удаления по gidRef
Api().deleteByKey(gidRef)
```

Объявление подключенных классов (классов-наследников)

Для связывания класса с миксином необходимо в Odm файле указать имена классов-миксинов.

```
<mixins>
  <mixin name="Bs_Settler" isDpiManaged="true"/>
</mixins>
```

Формирование кода

При формировании кода сущности в Dpi будет добавлен код:

- Вставки миксин-объекта в методе `insert`
- Удаления миксин-объекта в методе `delete`
- Установки значений атрибутов миксина
  Код будет добавлен в сеттера атрибутов имена которых совпадают с именами атрибутов сущности, в сеттерах атрибутов сущности.

При необходимости управления генерацией миксина вручную необходимо отключить формирование кода в Dpi с помощью настройки `isDpiManaged`.

По умолчанию по всем подключённым миксинам генерируется код в Dpi

Системные миксины

Функционально не отличаются от прикладных миксинов. Основной системный миксин это `Btk_Object`.

При формировании кода для классов с `supertype="document"` и `supertype="reference"`, этот миксин автоматически подключается к классу. Допускается отключение миксина Btk_ Object путём указания свойства в Odm файле:

```
<reflection isEnabled="false"/>
```

Пример смотрите в классе `Btk_Group`

Ссылочность на миксин

Для атрибутов, ссылочных на миксин, необходимо указывать тип `refAnyObject`:

```
<attr name="gidSettler"
      attribute-type="Varchar"
      caption="Контрагент"
      type="refAnyObject"
      isVisible="false"
      order="20"
      ref.class="Bs_Settler"/>
```

Для возможности автоматической генерации HL и MC атрибутов, сеттеров этих атрибутов и добавления их в `selectStatement` и `onRefreshExt` необходимо указать в настройке `ref.class` класс миксина для ссылки.

Важно: Для миксинов по ссылочным атрибутам не генерируются внешние ключи. Вместо этого генерируются индексы.

7.11.2 Трейт (Trait)

`Trait` — это механизм обеспечения повторного использования кода между классами. В отличии от наследования класс может содержать несколько трейтов.

Трейт не имеет объектов, собственной структуры хранения данных и визуального представления.

Создание трейта:

1. Создайте Odm файл
2. укажите свойство `supertype="trait"`

При формировании кода будут созданы только Dpi и Api файлы. Dpi будет содержать только объявления сеттеров и геттеров.

Наследование сущности от трейта

Для наследования сущности от трейта, добавьте в Odm свойство класса `with="Trait_Name"`

Сущность, наследуемая от трейта, должна иметь все атрибуты, объявленные в трейте.

Перекрытие и порядок вызовов методов

Рассмотрим код трейта и наследуемого от него класса:

```
trait Gs3_TraitApi[T] extends Gs3_TraitDpi[T] {
  override def setFNumber(rop: ApiRop, value: NNumber): Unit = {
    Logger.Factory.get(getClass).info("Gs3_TraitApi.setFNumber")
    super.setFNumber(rop, value)
  }
}

class Gs3_DescendantApi extends Gs3_DescendantDpi[T] with Gs3_TraitApi[T] {
  override def setFNumber(rop: ApiRop, value: NNumber): Unit = {
    Logger.Factory.get(getClass).info("Gs3_DescendantApi.setFNumber")
    super.setFNumber(rop, value)
  }
}
```

В результате выполнения метода `Gs3_DescendantApi.setFNumber()` последовательность вызовов будет следующей:

1. `Gs3_DescendantApi.setFNumber`
2. `Gs3_TraitApi.setFNumber`
3. `Gs3_DescendantDpi.setFNumber`

7.12 Пример разметки класса

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0" name="Bs_
↪GdsCostDeviationType"
       caption="Виды отклонений в стоимости ТМЦ"
       cardEditor.representation="Card" listEditor.representation="List"
       viewOptions.openCardType="mdi" supertype="reference">
    <attributes>
        <attr name="id" attribute-type="Long" caption="Идентификатор" order="-1" type=
↪"basic" isVisible="false"/>
        <attr name="idClass" attribute-type="Long" caption="idClass" order="-2" type=
↪"basic" isVisible="false"/>
        <attr name="gid" attribute-type="Varchar" isVisible="false"/>
        <attr name="sCode" attribute-type="Varchar" caption="Код" order="10" isMnemoCode=
↪"true" type="basic"
              isRequired="true" isVisible="true"/>
        <attr name="sCaption" attribute-type="Varchar" caption="Наименование" order="20"␣
↪isHeadLine="true" type="basic"
              isRequired="true" isVisible="true"/>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        <attr name="sDescription" attribute-type="Varchar" caption="Описание" order="30"␣
↪type="basic" editorType="memo"
              isVisible="true">
            <descriptionColumn/>
        </attr>
    </attributes>
</class>
```

Глава 8

Сервисные возможности для классов

Возможности классов расширяются сервисным окружением, которое постоянно совершенствуется.

8.1 Служебные атрибуты

Для сервисных возможностей фреймворк добавляет в таблицы служебные атрибуты. Служебные атрибуты заканчиваются постфиксом `_dz`.

Пример:

- `nVersion_dz`
- `screateuser_dz`
- `dcreatedate_dz`

Внимание: Разработчикам запрещено создавать и изменять такие атрибуты.

8.1.1 Отображение состояния сессии в форме

Состояние сессии можно определить по цвету системных операций `Сохранить` и `Откатить` на тулбаре формы.

Все состояния формы перечислены в таблице:

Пользовательская блокировка	Данные для отправки в базу данных	Кнопка «Сохранить»	Кнопка «Откатить»
Нет	Нет	не активна	не активна
Нет	Есть	активна	активна(крест)
Есть	Нет	не активна	активна(круг)
Есть	Есть	активна	активна(крест)

8.2 Автонумерация

Автонумерация – процесс автоматической выдачи значения для атрибута класса. В основе лежит выдача порядкового номера (числа), которое может быть преобразовано в значение по маске.

Автоматическая нумерация объектов системы позволяет устанавливать для созданных объектов уникальные коды. При этом создаваемые коды последовательно увеличиваются.

Существует два режима работы автонумерации:

- Без заполнения пропусков
- С заполнением пропусков

8.2.1 Определение разреза автонумерации

Автонумерация в пределах значений одного или нескольких атрибутов класса называется автонумерацией с разрезом. Значение разреза может вычисляться по правилу, указанному в теге «expression», если выражение не указано, будет использоваться значение атрибута.

Например, в течении одного дня выдаются номера последовательно, при наступлении следующего дня номера начинают выдаваться заново. Такой разрез называется разрез по дате.

8.2.2 Настройка автонумерации без разреза

Для автонумерации без разреза в odm файле класса указываем для атрибута тип `autonum`, далее в блоке `autonum` задаем имя разреза «dimension»

```
<attr name="sNoDep"
      caption="Автонумерация без разреза"
      attribute-type="Varchar"
      type="autoNum"
      isHeadLine="true">
        <autonum id="1">
                <mask>lpad(counter,3,"0")</mask>
                <dimension name="dim1"/>
        </autonum>
</attr>
```

8.2.3 Настройка автонумерации с разрезом

Для настройки разрезов `dimension`

1. Задайте уникальные имена
2. Укажите атрибуты разреза
   Разрезы указываются из атрибутов класса
3. При необходимости задайте порядковый номер
   Если его не указывать, то они будут обрабатываться в алфавитном порядке.

Количество разрезов должно быть >= 1.

Указание нескольких разрезов:

```
<attr name="sTwoDepExpr"
                caption="Автонумерация с 2 разрезами"
                attribute-type="Varchar"
                type="autoNum">
        <autonum id="3">
                <mask>lpad(counter,3,"0")</mask>
                <dimension name="nNumber"
                                attr="nNumber"
                                order="10" />
                <dimension name="dDate_dim"
                                attr="dDate"
                                order="20">
                        <expression>truncDate(dimValue)</expression>
                </dimension>
        </autonum>
</attr>
```

Если в настройках автонумерации указан разрез, то автоматически включено заполнение пропусков. Отключить заполнение пропусков можно через свойство `isHoleFill`

```
<attr name="sOneDepNoHoleFill"
      caption="Автонумерация с 1 разрезом без заполнения пропусков"
      attribute-type="Varchar"
      type="autoNum"
      isHeadLine="true">
        <autonum id="4"
                 isHoleFill="false">
                <dimension name="dDate"
                           attr="dDate"
                           order="10"/>
        </autonum>
</attr>
```

8.2.4 Общие настройки

Возможно использовать маски для вывода значения счетчика. Значение, указанное в элементе mask, будет передано в обработчик Jexl-скриптов.

Параметр, заменяемый на значение счетчика, называется `counter`. В Jexl добавлены методы:

- `lpad(string, size, padChar)` Добавление строки символами слева
- `rpad(string, size, padChar)` Добавление строки символами справа
- `truncDate(date)` Получение даты на начало дня от переданной
- `truncYear(date)` Получение даты на начало года от переданной

В jexl скрипте можно обращаться к api и pkg объектам.

К значениям атрибутов разреза возможно применение выражений, они указываются в элементе `expression` для `dimension`.

На каждый разрез может быть не больше одного выражения.

Значение, указанное в элементе `expression`, будет передано в обработчик Jexl-скриптов, параметр, заменяемый на значение атрибута разреза, называется `dimValue`.

```
<attr name="sRegNumber" attribute-type="Varchar" caption="Per. №"  order="230" type=
↪"autoNum" isVisible="false" isReadOnly="true">
        <autonum>
                <mask>
      var sOTCode = Btk_ObjectTypeApi.load(self.idObjectType()).getByAttrName(\"sCode\");
↪
      if (sOTCode eq \"OperDecisionPrj\"
          || sOTCode eq \"ProductDecisionPrj\") {
            counter + \"-Г\"
      } else {
        counter + \"-Г.\" + Bs_PrjVerApi.load(self.idPrjVer()).getByAttrName(\"sCode\")
      }
    </mask>
                <dimension name="idObjectType" attr="idObjectType" order="1">
                        <expression>
          if (dimValue == null) {
            null
          } else {
            var sCode = Btk_ObjectTypeApi.load(dimValue).getByAttrName(\"sCode\");
            if (sCode eq \"OperDecisionPrj\"
                || sCode eq \"OperDecisionWrk\"
                || sCode eq \"ProductDecisionPrj\"
                || sCode eq \"ProductDecisionWrk\"
                ) OL
                else null;
          }
        </expression>
                </dimension>
        </autonum>
</attr>
```

Для обращения к полям класса при вычислении маски используется переменная `self`, которая является объектом типа Aro, поэтому для получения значения атрибута необходимо добавлять в конце скобки (т.к. это метод). Например:

```
self.sCaption()
```

После генерации кода необходимо будет выполнить метод `*Dpi.autoNumCreateTable`.

Если с момента последнего выполнения данного метода были изменены данные автонумерации (имена разрезов, типы данных разрезов, количество разрезов), то таблицы и индексы будут пересозданы.

Так же генерация окружения автонумерации выполняется перед инициализацией первоначальных данных (init data), которая вызывается при обновлении проектного jar-кода или при выполнении генерации таблиц в Intellij Idea : `External Tolls > Generate Tables`

Присвоение значения автонумерации происходит в момент сохранения транзакции.

При ручной установке значения автонумерующегося атрибута (при работе из интерфейса) будет задан вопрос:

«Значение присваивается автоматически в момент сохранения, ввод данных отключит автонумерацию. Продолжить?»

Положительный ответ на вопрос установит значение и отключит автонумерацию объекта. Для повторного включения автонумерации нужно сбросить значение атрибута.

8.2.5 Автонумерация в миксинах

Возможно задание автонумерации в миксине. Атрибуты, которые объявлены как автонумерующиеся в миксине, но при этом как обычные в целевом классе, получат настройки автонумерации + сквозные таблицы хранения значений автонумерации для всех классов, использующих данный миксин. Если атрибут объявлен как автонумерующийся и в миксине, и в классе, то будет использована настройка класса.

8.2.6 Создание окружения автонумерации

Для создания окружения автонумерации по классу:

1. Зайдите в приложение «Настройка системы»
2. Откройте список классов
3. Перевидите фокус на целевой класс
   Для данного класса должен стоять признак «Есть автонумерация»
4. Выполните операцию «Создать структуры автонумерации»
   Данная операция находится под молоточками.
   При этом произойдет вызов метода `*Dpi.autoNumCreateTable`

Для создания окружения автонумерации по атрибуту класса:

1. Откройте карточку класса
2. Переведите фокус на целевой атрибут
3. Выполните операцию «Создать структуру автонумерации»
   Данная операция вызывает метод `*Dpi.autoNumCreateTableByAttr`

8.2.7 Проектное переопределение

Для индивидуальной настройки работы автонумерации реализован механизм проектного переопределения. Вся настройка производится из пользовательского интерфейса без необходимости изменения исходного кода и перекомпиляции.

Проектные настройки хранятся в БД и не сбрасываются при установке обновлений.

Для выполнения проектного переопределения:

1. Зайдите в карточку класса
2. Выберете автонумерующийся атрибут
3. Откройте закладку «Настройки автонумерации»;
4. На панели открытой закладки нажмите кнопку «Переопределить»
5. Настройте разрезы и маску;
6. Выполните операцию создания окружения нумерации.

Настройка счетчиков и данных разрезов.

Если автонумерация без разрезов, то она и без заполнения пропусков. Счетчик такой автонумерации хранится в сиквенсе.

Автонумерация с разрезами хранит свои данные в двух таблицах:

- Таблица разрезов
  В этой таблице на каждый уникальный набор значений разрезов хранится значение счетчика, маска, признак, что есть пропуски
- Таблица пропусков
  Для каждого разреза (ссылка на таблицу разрезов) хранит информацию о имеющихся пропусках. В виде пар значений: «значение от» и «значение до»

Таким образом для изменения текущего значения счетчика для автонумерации без разрезов достаточно изменить значение сиквенса.

Для автонумерации с разрезами чтобы изменить счетчик нужно установить значение в таблице разрезов. Для настройки:

1. Зайдите в карточку класса
2. Выберете автонумерующийся атрибут
3. Откройте закладку «Настройки автонумерации»
4. Откройте закладку «Данные автонумерации»
5. Настройте значение счетчика:
6. Если автонумерация без разрезов, то на детальной закладке будет карточка с одним атрибутом, который показывает текущее значение сиквенса.
7. Если автонумерация с разрезом, то на детальной закладке будут отображены данные из таблицы разрезов и таблицы пропусков. Для изменения счетчика в нужном разрезе указывается значение поле «Максимальное значение». Если разреза нет, то его можно создать. Если удалить разрез, то автонумерация начнется с 1 для этого разреза.

8.3 Копирование объектов

При создании класса, при необходимости генерируется операция копирования данный метод позволяет создать новый объект скопировав значение атрибутов и содержание его коллекций. При этом происходит глубокое копирование, то есть для коллекций создаются новые строчки. Так же возможно добавить в один объект данные из другого объекта.

Копирование управляется свойствами в метаданных (все по умолчанию = true). Возможность копирования объекта можно настроить на уровне всего класса, и на каждый атрибут. Например, запретив копирование нескольким атрибутам в классе.

На уровне класса копирование настраивается свойством `isCopyObjectEnabled`

На уровне атрибута свойством `isCopyInCopyObject`

В случае, если на классе включено копирование:

В мастер-выборке `EntityAvi#Default` создана операция «Копировать», пустая и не активная по умолчанию. Для классов с включенным свойством `isCopyObject` операция будет содержать исполняемый код, в отображении `List` операция будет активна.

В отображении `Card` создается операция `copyObjectCard`, которая создает новый объект и копирует в него данные по `CardRep.IdItemSharp`. Для параметра мастер-объекта используется переменная `idBOParent#`.

Настройка копирования в Odm для класса:

```
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
       name="Mct_OperCard"
       caption="Справочник операционных карт"
       cardEditor.representation="Card"
       listEditor.representation="List"
       viewOptions.openCardType="mdi"
       supertype="reference"
       isCopyObjectEnabled="true"
       attachType="simple"/>
```

Настройка копирования для атрибута класса:

```
<attr name="sCreateUser"
      attribute-type="Varchar"
      type="basic"
      isCopyInCopyObject="false"/>
```

При создании метода `copyObject` в мастер-классе в метод будут добавлены вызовы `copyObject` из коллекций если для этих классов включено копирование.

Если не указано свойство `isCopyInCopyObject` атрибут будет копироваться в методе `copyObject`. По умолчанию не копируются системные атрибуты:

- `id`
- `gid`
- `gidRef`
- `sImpExpKey_dz`
- `nImpExpFlag_dz`
- `sMnemoCode_dz`
- `sHeadLine_dz`
- атрибуты связи с мастер-сущностью.

8.3.1 Параметры метода copyObject

- Для миксинов
  `(gidFrom: NGid, gidTo: NGid, gidParent: NGid)`
- Для V-коллекций
  `(idFrom: NLong, idTo: NLong, gidParent: NGid)`
- Для всех остальных
  `(idFrom: NLong, idTo: NLong, gidParent: NLong)`

Метод `copyObject` создает объект (если `gidTo`/`idTo` не передан) с помощью dpi метода, установка значений происходит через Dpi-сеттеры.

8.4 Группировка

Группировка используется для систематизации хранения объектов и удобства восприятия пользователем, так же группировка позволяет массово управлять характеристиками и настройками объектов класса.

Дерево групп представляет собой иерархическую структуру, потомки наследуют характеристики предков с возможностью переопределения.

Для включения группировки класса

1. Откройте Odm для класса
2. Задайте `group.type` - тип группировки
   - single
     Объект может входить только в одну группу
   - multi
     Объект может входить в несколько групп
3. Задайте group.root - системное имя корня группировки класса

```
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
      name="Bs_Contras"
      caption="Контрагент"
      supertype="reference"
      cardEditor.representation="MainCard"
      listEditor.representation="MainRoList"
      roListEditor.representation="ROList"
      viewOptions.openCardType="mdi"
      group.type="multi"
      group.root="Group3" />
```

4. Добавьте в набор коллекций класса развязку групп

```
<var-collection name="Btk_ObjectGroup"
                ref.attr="gidSrcObject"/>
```

Для подключения группировки, у класса обязательно должен присутствовать глобальный идентификатор (атрибут `gid`).

Для управления группами в карточке объекта можно добавить закладку. Для этого:

1. Откройте avm класса
2. Добавьте закладку

```
<tabItems isVisible="true">
  <tabItem selection="gtk-ru.bitec.app.btk.Btk_ObjectGroupAvi"
           representation="List_Object"
           id="2"
           caption="Группы объекта"/>
</tabItems>
```

Связь объекта с группой осуществляется через класс Btk_ObjectGroup, денормализованное дерево групп объектов хранится в Btk_FlatObjectGroup. Эти таблицы могут использоваться в запросах для фильтрации объектов или организации прикладной бизнес-логики.

8.4.1 Интеграция с выборкой

При включении группировки в список объектов класса добавляются операции по работе с группами.

Для того чтобы открыть панель группировки для класса:

1. Откройте список объектов
2. Выполните операцию «Открыть группы»
   При этом слева откроется панель с деревом групп.

При включенной панели группировки в основной части выборки будут показываться объекты, находящиеся в текущей группе или ее потомках. Новые объекты будут создаваться в выбранной группе.

8.5 Сервис прикрепленных файлов

Сервис прикрепленных файлов позволяет прикреплять к объектам класса произвольные файлы, которые сохраняются в специальном хранилище на сервере и сопоставляются с конечным объектом-владельцем.

8.5.1 Файловое хранилище

Global3-Framework позволяет классифицировать места хранения файлов по функциональной принадлежности. Для этой цели используется справочник файловых хранилищ системы.

Файловое хранилище может быть локальным или сетевым. Локальное хранилище располагается в локальной директории сервера приложений. Сетевое хранилище использует SMB протокол.

Перед работой с файлами необходимо настроить файловые хранилища. Для открытия списка настроек:

1. Откройте приложение `Настройка системы`
2. Выполните пункт меню `Сущности > Файловые хранилища`

Минимально в системе должно быть настроено два файловых хранилища:

- `Default` – хранилище по умолчанию
- `btkAttach` – хранилище прикрепленных файлов

Для минимизации роста объема файлового хранилища используется специальный алгоритм хранения, позволяющий сопоставлять с одним физическим файлом несколько объектов системы:

- Файл в хранилище загружается только один раз
- копирование объектов системы, связанных с файлом, не создают
- физическую копию файла, а создают символическую ссылку на файл (`btk_attachitem`).

Существует два режима работы прикрепленных файлов:

- Простой
- Версионный

8.5.2 Простое хранение файлов

Позволяет работать с прикрепленными файлами без возможности хранения истории изменений файла. Этот режим является частным случаем версионного хранения с одной версией. Версия у файла всегда остается одна, любое изменение приводит к перезаписи файла версии.

Для режима простого хранения файлов доступны операции:

- Прикрепить файл
  В хранилище добавляется новый файл, загруженный пользователем, а в закладке отображается запись, соответствующая этому файлу, отмеченная как основная.
- Удаление
  Удаляется запись из закладки. В случае, если запись отмечена как основная, произойдет удаление файла из хранилища.

  Внимание: Операция удаления записи из хранилища необратима

- Скачать файл
  Из хранилища скачивается файл с именем и расширением, указанным в записи.
- Сделать файл основным
  Для записи устанавливается флаг «Основной». Для записи, которая была основной до этого, флаг снимается.

При копировании родительской записи, так же копируются записи прикрепленных файлов, без флага «Основной».

8.5.3 Версионное хранение файлов

Версионирование прикрепленных файлов позволяет сохранять историю изменений. Если включен режим версионирования, любое изменение файла приводит к созданию новой версии.

В режиме версионного хранения доступны операции обычного хранения, а также ряд дополнительных:

- Добавить новую версию
  К записи добавляется новая версия. При этом предыдущая версия остается в хранилище.
- Удалить последнюю версию
  Из хранилища удаляется последняя версия файла. В случае, если версия только одна, будет предложено удалить запись.

  Внимание: Операция удаления записи из хранилища необратима

- Отобразить историю изменений файла
  Открывается список с версиями для текущей записи. Из списка так же доступны операции удаление и скачивания для каждой версии.

При копировании родительской записи, так же копируются записи прикрепленных файлов с версиями, без флага `Основной`.

8.5.4 Настройка

По умолчанию сервис прикрепленных файлов для класса отключен. Для подключения необходимо в odm файле в настройке класса указать тип хранения – обычный или версионный

```
<class xmlns="http://www.global-system.ru/xsd/global3-class-1.0"
       name="Gds_ControlDoc"
       caption="Документ контроля"
       cardEditor.representation="Card"
       listEditor.representation="List"
       viewOptions.openCardType="mdi"
       supertype="document"
       attachType="versioned"/>
```

При генерации avm файла в отображение `List` и `Card`, а также для классов с иерархией в отображение `tree` добавится закладка «Прикрепленные файлы»

8.6 Поиск по шаблону

Поиск по шаблону - серверная возможность поиска объектов класса по частичному или полному совпадению введенного текса со значениями полей объекта или его заголовка и мнемокода.

Поиск регистронезависимый.

8.6.1 Алгоритм работы поиска

1. Определяются поля, по которым требуется осуществлять поиск, и их приоритет.
2. Выполняются последовательно запросы пока не получен результат в виде одной записи
    1. по всем полям в порядке приоритета ищутся значения по полному совпадению введенного текста (=)
    2. по всем полям в порядке приоритета ищутся значения по частичному совпадению введенного текста (`like`)
3. Возвращается результат
   Результатом поиска является первый запрос который вернул одну запись, или первый запрос который вернул данные.

По умолчанию для класса включен поиск по шаблону, и осуществляется по служебным полям `sheadline_dz` и `smnemocode_dz`. Сначала по заголовку, потом по мнемокоду.

8.6.2 Настройка поиска для класса

В odm-файле для класса добавить тег `patternSearch`

```
<class>
        <patternSearch isActive="true"
                       headLineOrder="100"
                       mnemoCodeOrder="200"
                       searchType="startsWith"
                       indexHlMc="true"/>
</class>
```

Описание настроек:

- `isActive`
  Активность поиска по шаблону на классе. Если выключено, то функция поиска по шаблону будет возвращать 0 записей. Умолчательное значение = true
- `headLineOrder`
  Порядок поиска по служебному полю `sheadline_dz`. Умолчательное значение = 1. Чем меньше значение, тем выше приоритет описка по этому полю. Поля с наименьшими значениями порядкового номера обрабатываются в первую очередь.
- `mnemoCodeOrder`
  Порядок поиска по служебному полю `smnemocode_dz`. Умолчательное значение = 2. Логика обработки как в `headLineOrder`
- `searchType`
  Тип поиска:
  - по началу слова(по умолчанию)
    like с правым %
  - по вхождению
    like с правым и левым %
- `indexHlMc`
  Индексировать поля `sheadline_dz` и `smnemocode_dz`. При включенном свойстве будут созданы индексы по этим поля в верхнем регистре.

8.6.3 Настройка поиска по атрибутам

1. Откройте odm файл
2. Добавьте тэг patternSearch для нужного атрибута
   Если тег добавлен, то атрибут участвует в поиске по шаблону.

```
<attr name="sText"
      attribute-type="Text"
      type="basic"
      order="130"
      caption="Text">
  <patternSearch order="10"
         searchType="contains"/>
</attr>
```

Описание настроек:

- `order`
  Порядковый номер поиска по полю этого атрибута.
  Нумерация сквозная по классу, т.е. можно настроить, чтобы атрибут обрабатывался раньше заголовка и мнемокода.
- `searchType`
  Тип поиска:
  - по началу слова(по умолчанию)
    like с правым %
  - по вхождению
    like с правым и левым %

8.6.4 Перекрытие метода поиска

Метод реализован в одном из рутовых API:

```
ru.bitec.app.gtk.eclipse.rdb.SEntityBaseApiImpl#findByPattern
```

В наследниках его можно перекрыть, и написать свою логику поиска по шаблону.

8.7 Объектные характеристики

Характеристика – это качественное или количественное свойство объекта. Характеристиками в системе являются специальные атрибуты, которые содержат дополнительную пользовательскую информацию. Система может хранить характеристики двумя способами:

- Хранение в json контейнере объекта
  Этот способ является основным. Представляет собой NOSql расширение строки таблицы, что позволяет добавлять характеристики динамически, без пересборки проекта и обновления схемы базы данных.
- Хранение в виде атрибута класса с признаком «объектная характеристика» \
  В некоторых случаях характеристики используются в высоконагруженной бизнес-логике, что не позволяет использовать json контейнер. Для снижения накладных расходов доступа к данным характеристика может хранится в колонке таблицы класса. При добавлении характеристик, которые хранятся в атрибуте класса требуется синхронизация схемы БД.

8.7.1 Интеграция с выборкой

Для вывода характеристик используется универсальное отображение объектных характеристик. За необходимость генерации отображения отвечает настройка в Odm `objectAttrCardType` Варианты настройки:

- `None`
  Объектные характеристики не используются, отображение `Card_ObjectAttr` не формируется
- `Simple`
  Объектные характеристики настраиваются в классе или на типе объекта. Если у класса есть тип объекта, то характеристики настраиваются на типах объекта этого класса. Если у класса нет типа объекта, то выводятся все объектные характеристики.
- `Group`
  Объектные характеристики настраиваются для групп
- `GroupAndObjectType`
  Объектные характеристики настраиваются для групп и дополняются характеристиками, настроенными на типе объекта.

Отсутствие настройки равнозначно

```
objectAttrCardType="simple"
```

В отображение объектных характеристик выводятся значения атрибутов, настроенных в качестве объектных характеристик класса. Характеристики хранятся в системном json атрибуте `jObjAttrs_dz`, а также в атрибутах класса с признаком: `isObjectAttr="true"`

Так же в этом отображении выводятся и `универсальные характеристики` (См. главу `Универсальные характеристики` в Документация по модулю btk)

8.7.2 Настройка характеристик на проекте

Пользователь может добавить необходимые характеристики в карточке класса. Для этого:

1. Откройте приложение `Настройка системы`
2. Выполните пункт меню `Сущности > классы`
3. Перейдите на закладку атрибуты
4. При необходимости задайте значение по умолчанию для уже созданных атрибутов
5. Создайте атрибут объектной характеристики

Примечание: Имена атрибутов должны соответствовать соглашению по именам атрибутов:

- Строковые начинаются с `s`. Например, `sStringAttrName`
- Числовые начинаются с `n`. Например, `nNumberAttrName`
- Даты начинаются с `d`. Например, `dDateAttrName`
- Логические начинаются с `b`. Например, `bBooleanAttrName`

---

6. Сбросьте кэш выборки
   Для этого снимите галочку с в пункте меню `Сервис > Управление решением > Использовать кэш метаданных выборок`
7. Сбросьте кэш настройки
   Для на текущей закладке атрибутов выполните операцию `Сбросить кеш настройки`

8.7.3 Значения по умолчанию

Значения по умолчанию заполняются при создании объекта. Если класс с группировкой, то значения по умолчанию подставляются из настроек основной группы.

Для базовых атрибутов класса значение по умолчанию указывается в Odm, и устанавливается в Dpi при вставке. Для таких атрибутов есть возможность переопределить значение, указанное в Odm.

Проектное переопределение

В карточке класса в списке атрибутов нужно установить значение в поле «Значение по умолчанию». Для сброса переопределения:

- в списке атрибутов выполнить операцию «Удалить переопределение значения по умолчанию»

Программное переопределение

Программное переопределение возможно в методе

```
ru.bitec.app.btk.Btk_AttributeApi#overrideDefaultValue
```

Правила указания значений

1. Для ссылочных полей указывается ID ссылочного объекта
2. Для атрибутов переменной ссылочности указывается GID ссылочного объекта
3. Для числовых разделителем целой части и дробной является точка.
4. Дата указывается в стандартном формате. dd.MM.yyyy HH:mm:ss. Если указать «sysdate», то значением по умолчанию будет текущая дата на момент вставки объекта.

8.7.4 Програмная установка значений json-характеристик

В бизнес-логике:

```
val rop = Bs_GoodsApi.load(209851.nl);
//установка ссылочной характеристики
Bs_GoodsApi().setAttrValue(rop, "idGradeJson", 77016.nl);
```

В jexl-скрипте:

```
var rop = Bs_GoodsApi.load(209851L);
//установка ссылочной характеристики
Bs_GoodsApi.setAttrValue(rop, "idGradeJson", 77016L);
```

8.8 Аудит

Аудит предназначен для фиксации различных событий при работе пользователей в системе: вставка, изменение, удаление, выполнение операций, и т.д.

Аудит ведется в рамках бизнес-объектов и классов, входящих в БО. Для нужд аудита используется отдельная схема в БД, Для БО в схеме `Aud` формируется таблица с именем `<Имя класса-шапки БО>_dzAud` в табличном пространстве, указанном в настройке модуля btk `Табличное пространство аудита` (по умолчанию `pg_default`), в которой хранятся данные аудита.

Внимание: Если в классе включен аудит, а в классе-шапке БО, в который входит этот класс – не включен или у него не синхронизирована структура аудита, аудит не будет сохраняться. Класс-шапка БО для объекта класса определяется через системное поле `gidRoot_dz`

Настройка:

1. Зайдите в приложение `Настройка системы`
2. Откройте список классов
   Пункт меню `Сущности > Классы`
3. Выберите требуемый класс
4. В закладке `Характеристики` установите признак `Вести аудит`
5. Если класс не является шапкой БО, включите аудит для шапки БО
6. Для класса шапки БО выполните операцию `Создать оболочку аудита`
   При этом произойдет создание таблицы аудита в схеме аудита

7. Сбросьте `shared кэш`
   Для этого выполните пункт меню `Сервис > Управление решением > Очистить кэш ORM(shared кэш)`

> **Внимание:** Для того чтобы аудит вступил в силу, пользователь должен переоткрыть форму

Просмотр данных аудита:

- В карточке объекта по пункту меню: `Информация > Аудит объекта`
- В списке объектов класса из контекстного меню `Информация > Аудит`
- Общий список аудита `Настройка системы \ Аудит > Аудит`

Методы работы с аудитом:

- `Btk_AuditPkg().isEnabledAudit`
  Признак, что аудит ведется
- `Btk_AuditPkg().isEnabledAudit()`
  Сеттер, для изменения признака ведения аудита в сессии
- `Btk_AuditPkg().insertRow`
  создать запись в аудите.

8.8.1 Перенос существующих таблиц аудита в указанное табличное пространство:

Для обновления табличного пространства:

1. Зайдите в приложение `Настройка системы`
2. Откройте настройки модуля btk
   Пункт меню `Настройки и сервисы > Настройки модулей системы > Общие настройки модулей`
3. Выберите настройки модуля btk
4. В настройке `Табличное пространство аудита` укажите желаемое табличное пространство
5. По операции под молоточками `Скачать файл со скриптом переноса таблиц аудита` скачайте скрипт sql и запустите на выполнение

8.9 Автоматическая генерация штрих-кодов объекта

Сервисная возможность при вставке объекта класса формировать ему штрихкод.

Для этого в Odm в теге Class есть свойство `barCodeObjectTypeName` в качестве значения указывается системное имя объектов класса `Bs_BarCodeObjectType`.

Для регистрации новой записи используется метод

```
Bs_BarCodeObjectTypeApi().register
```

8.9.1 Описание работы маски

В случае, если в маске указано: `001-[counter]`

При этом:

- Длина штрихкода составляет 10 символов.
- Следующее значение счетчика: 12

Итоговый штрихкод:

```
001-000012
```

8.10 Подписи объектов для печати

Сервисная возможность указания типа подписей объекта. Используется при печати отчетов. Позволяет формировать в печатной форме список лиц с местом для подписи.

Список подписей настраивается в коллекции для печатной формы. Список может быть переопределен у объекта. Для хранения подписей печатных бланков используется json-контейнер `jSign_dz`.

Приоритет отображения данных по типу подписи:

1. Данные по типу подписи, хранимые в атрибуте объекта класса `jSign_dz` (формируется для всех классов)
2. Данные из настроек подписей в отчете.

При печати отчета, если удалось определить сохраненные подписи объекта, то будут переданы настройки подписей на самом отчете.

Кроме этого, в базовых справочниках реализована возможность формирования комиссий, состав которых может быть передан в отчет. Подробная настройка комиссий будет рассмотрена в разделе «Прикладная типизация объекта класса»

Данные подписей передаются в параметр `SIGNDATA_DZ`. Представляют собой json-объект с ключами:

- `idComission` – идентификатор комиссии.
  Определяется через тип объекта и подразделение.
- `data` – массив подписей объекта
  Каждый элемент представляет собой json-объект, на каждый тип подписи. Описание полей json-объекта подписи:
  - `idBlankSignType` – ИД типа подписи
  - `idBlankSignTypeMC` – системное имя типа подписи
  - `idBlankSignTypeHL` – наименование типа подписи
  - `sPosition` – должность
  - `sFIO` – Фамилия Имя Отчество
  - `sBasisDocument` – Документ-основание
  - `dDate` – дата подписи
  - `idDepartment` – ИД подразделения
  - `idDepartmentMC` – код подразделения

- `idDepartmentHL` – наименование подразделения
- `nOrder` – Порядковый номер подписи комиссии

Для работы с атрибутом `jSign_dz` реализован API в пакете `Bs_ObjectSignPkg`

В случае, если для класса задан `тип объекта` для работы с подписями можно воспользоваться стандартной закладкой `ru.bitec.app.bs.sign.Bs_ObjectSignAvi.List_Master`

8.11 Полнотекстовый поиск

Сервисная возможность класса осуществлять быстрый поиск по значению атрибутов класса.

Данные классов хранятся в индексе полнотекстового поиска в схеме fts:

- `Btk_FtsReg` – индекс объектов классов
- `Btk_FtsWords` – индекс используемых слов.

Форма поиска доступна в главном меню приложений, в меню «Сервис»

8.11.1 Настройка

Для включения полнотекстового поиска в Odm установить свойство класса `fts.Enabled` в значение `true`

По умолчанию все строковые значимые/автонумерующиеся/вычисляемые, а также ссылочные на объект и атрибуты переменной ссылочности с указанным ссылочным классом участвуют в полнотекстовом поиске.

Для переопределения поиска по атрибуту используется свойство `fts.Enabled`

8.11.2 Обновление индекса

После обновления кода на проекте и вызова ssh-команды `init Data` происходит анализ классов. Если настройки класса изменились (класс стал индексируемым или добавились новые индексируемые атрибуты, которые до этого не индексировались. Новые атрибуты не учитываются). Происходит планирование обновления индекса для всех объектов класса.

При изменении объектов классов и сохранении в БД (session.flush()), которые индексируются, или на которые ссылаются объекты индексируемых классов, эти объекты добавляются в лог измененных объектов индекса.

Значения ссылочных полей хранятся в строковом виде, и определяются как значение поля `sHeadLine_dz` ссылочного объекта.

В системе зарегистрировано задание «Синхронизация полнотекстового индекса», которое обновляет индекс. Это задание анализирует таблицу измененных объектов и таблицу измененных классов, сохраняет эти данные в общую очередь на индексацию, вычисляет и обновляет данные индекса.

8.11.3 Команды строки поиска

Формат описания:

- `Введенная строка`
  Логика обработки

Команды:

- `Цепь`
  Найдет текст, в котором присутствует слово цепь, цепи, цепью и тд.
- `цепь звено`
  Надет текст, в котором присутствуют оба слова и их склонения
- `Цепь И звено`
  Тоже самое, что и цепь звено. Пробел считается оператором «И»
- `Цепь & звено`
  Тоже самое, что и цепь звено
- `Цепь ИЛИ звено`
  Надет текст, в котором присутствует одно из или оба слова и их склонения
- `Цепь | звено`
  Тоже самое, что и цепь ИЛИ звено
- `Цепь И (Звено ИЛИ Кольцо)`
  Найдет текст, в котором есть слово цепь и одно из слов звено или кольцо
- `Цепь#40`
  Надет текст, в котором есть похожие на цепь слова, например, цеп. Число после # показывает точность поиска, от 1 до 100. Где 100 – полное совпадение
- `#Цепь`
  Тоже самое, что и Цепь#40
- `Цеп*`
  Найдет текст, слова в котором начинаются на цеп. Например, цепь, цепная и тд.

8.12 Пообъектный доступ

Сервисная возможность разграничения доступа на уровне объектов класса.

Права выдаются ролям и пользователям.

Пользователи, замещающие других, обладают правами замещаемых пользователей.

Права на объекты выдаются в виде группы доступа (`Btk_RlsAccessGroup`), на каждый объект для пользователя или роли может быть только одна группа.

8.12.1 Управление правами

Для управления правами в интерфейсах выборок используется библиотека `Btk_RlsLib`, и ее методы:

- `manageAccess`
  Операция для стандартных выборок, вызывает интерфейс настройки прав для всех выделенных строк. Определяет объекты через поля `gid` или `id`/`idClass`
- `manageAccessByGid`
  Открывает интерфейс настройки прав по одному переданному объекту
- `manageAccessByGidList`
  Открывает интерфейс настройки прав по перечню объектов

Если в выборке администрируются одновременно несколько объектов, то изначально будут показаны пользователи и роли, обладающие правами на все объекты и одинаковыми группами доступа.

Для выдачи прав в бизнес-логике используются методы пакета Btk_RlsPkg:

- `grantToUsers` выдача прав пользователям
- `revokeFromUsers` – отзыв прав у пользователей
- `grantToRoles` – выдача прав ролям
- `revokeFromRoles` – отзыв прав у ролей

8.12.2 Удаление администрируемых объектов.

При удалении объектов, для которых используется пообъектный доступ, необходимо выполнить очистку структур доступа. Для этого используются методы пакета Btk_RlsPkg:

- `clearByObject`
- `clearByObjectList`

8.12.3 Проверка наличия прав на объекты

Для проверки прав в запросах можно использовать таблицу `Btk_RlsUserRightsFlat`, при построении запроса необходимо использовать `exists` по `gidObj` и `idUser`

Для проверки прав в бизнес-логике используются методы пакета Btk_RlsPkg:

- `getAccessByObject`
- `getAccessByObjectList`

8.13 Сервис универсальных коллекций

Для работы системы реализована возможность прикрепления произвольной коллекции к любому классу системы, при этом не создавая связи между модулями и не привлекая к настройке программиста.

Для этого во всех Dpi методах удаления вызывается

```
Btk_UniversalTabPkg().deleteManualCollections(rop.gid)
```

При удалении записи в которой есть коллекции доступно 2 варианта действий:

- Каскадное удаление
  Удаляются все записи коллекций, ссылающиеся на удаляемую запись
- Генерация ошибки
  При наличии записей в коллекции, ссылающихся на удаляемую, поднимается ошибка

8.13.1 Подключение из приложения

1. Откройте приложение `Настройка системы`
2. Выполните пункт меню `Сущности > Классы`
3. Откройте карточку нужного класса
4. Перейдите на закладку `Коллекции`
   На данной закладке отображаются все коллекции класса, а также есть возможность добавить универсальную коллекцию
5. При необходимости добавьте универсальную коллекцию
   При добавлении требуется указать класс и атрибут ссылочности.

   > **Внимание:** Ссылочность возможна только для атрибутов с типом данных `Long` и типом `RefObject` с указанным классом для ссылочности или с типом данных `Varchar` и типом `RefAnyObject`. Ссылочность поддерживается в том числе и для проектных атрибутов и объектных характеристик, хранящихся в json контейнере `jObjAttrs_dz`

6. При необходимости настройте «Действие при удалении»
7. При необходимости «Удалите коллекцию»
   Удаление из списка доступно только для коллекций, добавленных вручную
8. Сбросьте кэш Btk_ClassCollection является классом попадающим в кэш, после подключения новых коллекций необходим сброс объектного кэша

> **Внимание:** При подключении для типа объекта универсальных закладок обязательно убедитесь, что для класса указана универсальная коллекция, если она необходима

8.13.2 Подключение из прикладного кода

В некоторых случаях необходимо подключение универсальной коллекции из прикладного кода, для этого можно воспользоваться методом

```
Btk_ClassCollectionApi().register(
  idpBtkClass: NLong,
  idpRefClass: NLong,
  idpRefAttr: NLong,
  spDelAction: NString
): ApiRop
```

Доступные действия содержатся в объекте

```
Btk_ClassCollectionApi
```

8.14 Денормализация классов-деревьев

Зачастую для древовидных классов требуется вспомогательный класс денормализации, который обеспечивает более быстрый доступ к данным по иерархичным связям, чем рекурсивные запросы.

Для работы системы с денормализацией древовидных классов реализован трейт (`trait`) `TreeDenormApi` (`ru.bitec.app.btk.denormalization.TreeDenormApi`). Данный трейт предоставляет функциональность создания/обновления/удаления денормализации по объектам из класса-дерева.

Пример: Пусть название класса-дерева будет `Btk_TreeExample`.

Для подключения денормализации необходимо создать класс, хранящий денормализацию.

Обычно в таком классе содержатся атрибуты:

- `id`
- `idClass`
- `idParent`
- `idChild`
- `nParentLevel`.

Для даного примера название такого класса `Btk_TreeExampleDenormalization`.

После создания класса `Btk_TreeExampleDenormalization` необходимо подключить к его Api трейт `TreeDenormApi`.

Затем имплементировать абстрактный метод `masterTableName`, который должен возвращать имя таблицы хранения класса-дерева `Btk_TreeExample`.

Пример синтаксиса подключения:

```
class Btk_TreeExampleDenormalizationApi extends
  Btk_TreeExampleDenormalizationDpi[
    Btk_TreeExampleDenormalizationAro,
    Btk_TreeExampleDenormalizationApi,
    Btk_TreeExampleDenormalizationAta
  ]
  with TreeDenormApi[
    EntityAbst,java.lang.Long,
    Btk_TreeExampleDenormalizationAro
  ]

  override def masterTableName: NString = "Btk_TreeExample"
```

Если названия полей в вашем классе хранения денормализации отличаются от обычных имён, то имеется возможность переопределения методов геттеров названий полей:

```
def idParentFieldName = "idParent"
def idChildFieldName = "idChild"
def nParentLevelFieldName = "nParentLevel"
```

Далее нужно запланировать событие заполнения денормализации. Обычно это делают в Api основного класса (`Btk_TreeExample`) в методе установки ссылки на предка предположим `setidParent` и в методе `insert` на случай, когда родительская запись остается пустой. В таком методе нужно вызвать

```
ru.bitec.app.btk.denormalization.TreeDenormApi#updateDenormAfterFlush
```

Метод `updateDenormAfterFlush` накапливает все записи, которые ему передавали что позволяте обновить денормализацию после сброса данных в БД.

Также необходимо удалять денормализацию при удалении записи из основного класса. Обычно это делают в методе `delete`. В нём нужно вызвать метод

```
ru.bitec.app.btk.denormalization.TreeDenormApi#deleteDenormBeforeFlush
```

`deleteDenormBeforeFlush` накапливает все записи, которые ему передавали что позволяет удалять по ним денормализацию перед сохранением в БД.

Также возможно обновление/удаление денормализации непосредственно в момент редактирования, а не после сохранения, однако такое использование замедлит работу сервиса, и оно не рекомендовано. Методы прямого обновления/удаления:

```
ru.bitec.app.btk.denormalization.TreeDenormApi#updateDenorm
ru.bitec.app.btk.denormalization.TreeDenormApi#deleteByObject
```

Примечание: Реализацию можно посмотреть в `Bs_DepartmentTreeApi` в связке к `Bs_DepartmentApi`.

8.15 Настройки приложения

Настройки приложения позволяют задовать глобальные константы Для получения значения настройки используйте функцию

```
Btk_AppPropertiesTypeApi.GetbValue(
  sType //- Системное имя настройки
  idObjectType //Тип объекта
) //возвращает true или false
```

8.15.1 Btk_AppPropertiesType

Таблица настройки приложения, содержит поля:

- `sSysName` - Системное имя
- `sCaption` - Наименование
- `sDescription` - Описание
- `idModule` - Модуль

8.15.2 Btk_AppProperties

Таблица значений настроек приложения, содержит поля:

- `idType` - Настройка
- `idObjectType` - Тип объекта
- `bValue` - Значение boolean
- `nValue` - Значение number
- `sValue` - Значение string
- `dValue` - Значение date

8.16 Вставка изображений в прикреплённый файлы типов word и pdf

Для вставки изображений в прикрепляемые файлы документа необходимо настроить нужные изображения в коллекции-расширении(Настройки вставки изображений) к типу объекта этого документа.

В коллекции доступны следующие настройки:

- Активность Данное поле отвечает будет ли вставленные изображения в прикрепляемый файл
- Печатная форма В поле указывается печатная форма с типом jasper и форматом png, которой будет передан только один аргумент «IDDOC» - id документа.
- Изображение Файл изображение в формате png

Примечание: В случае указания печатной формы как источник изображения файл изображения будет удалён. Если сначала будет указана печатка то при загрузке файла изображения, удалена будет ссылка на печатную форму.

- Положение изображения по осям XY Поля отвечают за положение изображение внутри PDF документа, точкой отсчёта является нижний левый угол
- Ширина и высота изображение Поля отвечают за размер изображения Как только в коллекции-расширении настройки изображений будет активное изображение для вставки, в отображении прикреплённых файлов к документу данного типа объекта станут активны операции для вставки данных изображений в один документ или вовсе сразу.

8.17 Справочник параметров

Для создания локальных коллекций с параметрами класса реализован трейт (`trait`) `Prm_ParamCalcApi` (`ru.bitec.app.prm.techparam.Prm_ParamCalcApi`) и абстрактное отображение `Prm_CalcTechOpParamAbsAvi` (`ru.bitec.app.prm.techparam.Prm_CalcTechOpParamAbsAvi`).

8.17.1 Prm_ParamCalcApi

В трейте реализована функциональность заполнения параметров и их значений.

Обычно в коллекции содержатся атрибуты:

- `idTechParameter` - Ссылка на параметр Prm_TechParameter
- `idRefTable` - Ссылка на таблицу значений параметров Prm_RefTable
- `nValue` - Числовое значение
- `sValue` - Строковое значение
- `dValue` - Значение даты
- `bManualInput` - Параметр заполнен вручную

Для использования функционала необходимо в Api нужного класса отнаследовать данный трейт и переопределить геттеры(сеттеры) параметров, Api родительского класса. Пример:

```
class Mnf_MCOperParamApi extends Mnf_MCOperParamDpi[Mnf_MCOperParamAro, Mnf_
↪MCOperParamApi, Mnf_MCOperParamAta] with Prm_ParamCalcApi[EntityAbst, java.lang.Long,␣
↪Mnf_MCOperParamAro] {
  override protected def entityAta: Mnf_MCOperParamAta = Mnf_MCOperParamAta

  override lazy val apiParent = Mnf_MCOperApi().asInstanceOf[SApi[AnyRef, Aro[_ <:␣
↪AnyRef]]]

  override def getsValue(rop: ApiRop): NString = rop.get(_.sValue)
  override def getnValue(rop: ApiRop): NNumber = rop.get(_.nValue)
  override def getdValue(rop: ApiRop): NDate = rop.get(_.dValue)
  override def getidValue(rop: ApiRop): NLong = rop.get(_.idValue)

  override def setidTechParam(rop: ApiRop, value: NLong): Unit = setidTechParameter(rop,␣
↪value)
  override def getidTechParam(rop: ApiRop): NLong = rop.get(_.idTechParameter)

  override def getidRefTable(rop: ApiRop): NLong = rop.get(_.idRefTable)

  override def getidCalcFormulaByParent(ropParent: AnyRop): NLong = ropParent.
↪asInstanceOf[Mnf_MCOperApi#ApiRop].get(_.idCalcFormula)

  override def isManualInput(rop:ApiRop): Boolean = rop.get(_.bManualInput).toBoolean

}
```

Если в вашем классе не используются какие-либо из этих атрибутов, то их сеттеры и геттеры можно переопределить на Unit() и null-значение соответственно.

Функция `isManualInput` используется для выбора параметров, значения которых будут рассчитаны в функции `refreshParamValByParent`.

Дополнительные возможности:

- `setParamValueRichAnyRef` - простановка значения параметра RichAnyRef(value)
- `getParamValue` - получение значения параметра
- `clearAllValues` - очищает значения параметра

- `fillByParent` - заполнение параметров по формуле от родителя
- `getsParamsCodeAndValueByParent` - возвращает строку со значениями параметров вида (код параметра: значение)

8.17.2 Prm_CalcTechOpParamAbsAvi

Абстрактная Avi содержит в себе два функциональных трейта:

- `Default_Calc` - Для расчета по формуле, параметры не хранятся
- `SavedParamsCalc` - Для коллекций с хранимыми значениями, можно использовать в паре с трейтом Prm_ParamCalcApi

Default_Calc

Примечание: Пример использования - `Mct_TechOpParamAvi.List_idOperCardOperationForCalculate`

В интерфейсе: `Управление данными об изделии (Судостроение) -> Технология -> Справочник операционных карт (Mct_OperCardAvi.list) -> Закладка Операции -> Проверка формул по параметрам`

Для использования необходимо отнаследоваться от данного отображения и переопределить методы `getFormula` (получение формулы) и `getParams` (получение параметров).

SavedParamsCalc

Аналогично трейту Prm_ParamCalcApi переопределяются сеттеры значений, а также функции `onRefresh`, `onRefreshItem` и `getAdditionalInfo`

Примечание: Примеры использования:

`ru.bitec.app.mnf.manufCard.Mnf_MCOperTestParamAvi.List_idMcOper`

`ru.bitec.app.mct.techproc.Mct_TechProcNormGroupParamAvi.List_idTechProcNorm`

Необязательная функция `refreshValues` - заполнение параметров (по умолчанию active = false) - для вызова функции заполнения параметров refreshParamValByParent трейта Prm_ParamCalcApi и/или своего необходимого функционала

Глава 9

Тип объекта

Сервис предоставляет набор стандартных возможностей, доступных для всех типизируемых классов. Основным назначением типа объекта является хранение настроек по документу, которые могут быть переопределены на проекте.

Основные настройки:

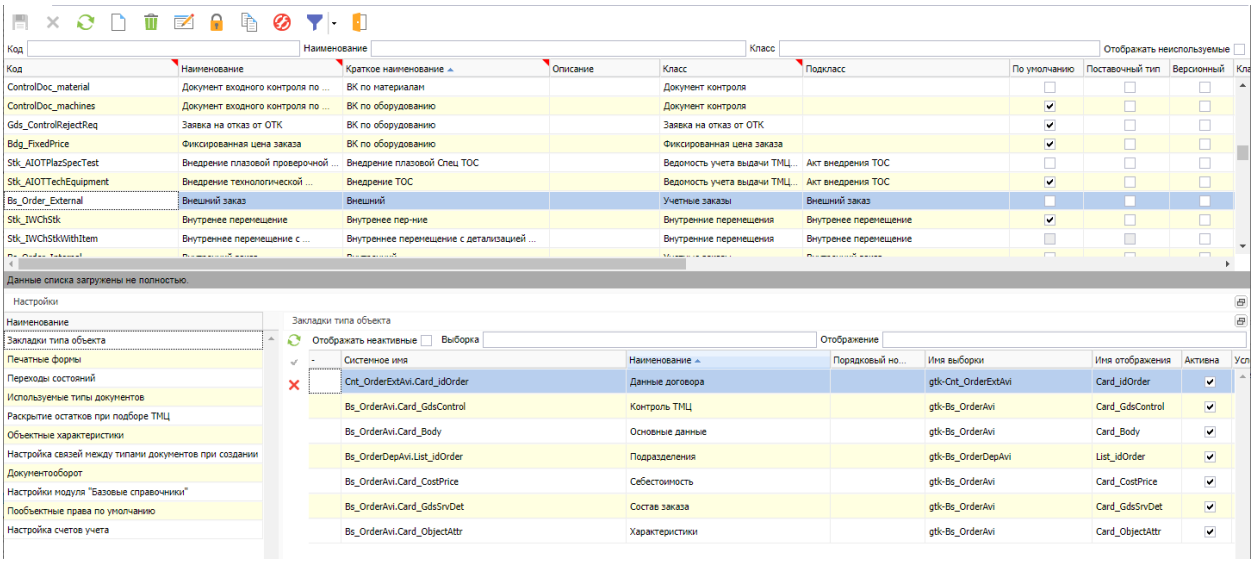
- Настройка закладок
- Настройка печатных форм
- Настройка переходов состояний
- Настройка объектных характеристик

Также предусмотрены механизмы для подключения настроек для модуля или подсистемы.

Тип объекта может менять поведение документа в зависимости от бизнес процесса в котором данный документ участвует. В случае, если необходимо запрограммировать поведение документа в зависимости от типа объект следует использовать системное имя подкласса.

Внимание: Использовать системные имена типов объекта в коде нельзя, так как типы объектов могут быть добавлены на проекте.

На базе типа объекта реализовано множество сервисных возможностей, позволяющих тонко настраивать типизированные документы не привлекая программистов.



9.1 Подключение типа объекта к классу

Примечание: Для супертипа document (Документ) ссылка на тип `idObjectType` создается автоматически

Пример объявления атрибута:

```
<attr name="idObjectType"
      attribute-type="Long"
      caption="Тип объекта"
      order="50"
      type="refObject"
      isVisible="true"
      ref.class="Btk_ObjectType"/>
```

Система имеет общий справочник типов объекта (`Настройка системы \ Сущности > Типы объектов > Типы объектов`)

Можно регистрировать типы объекта в коде *Регистрация в коде*

Объекты класса типизируются классом-настройкой `Btk_ObjectType`. В данной настройке можно задать несколько типов для одного класса системы, а также назначить тип по умолчанию. Обязательным условием для типизации объектов является наличие в типизируемом классе атрибута-ссылки на `Btk_ObjectType`. Атрибут обязательно должен называться `idObjectType`. Это имя зарезервировано фреймворком и не может использоваться для других целей.

9.2 Подкласс

Подкласс – это логическая типизация для разделения бизнес-логики в рамках одного класса. Подклассы добавляются разработчиками и на них можно завязываться в коде. Несколько типов объекта может быть привязано к одному подклассу

Система имеет общий справочник подклассов (`Настройка системы \ Сущности > Типы объектов > Подклассы`), в котором настраивается:

- Код
- Наименование
- Суперкласс (Класс к которому будет привязан данный подкласс)

Можно регистрировать подклассы в коде *Регистрация в коде*

Примечание: Системные имена подклассов можно использовать в коде.

9.3 Стандартные настройки

9.3.1 Закладки типа объектов

Для типизированного объекта доступна возможность управления выводом закладок.

Система имеет общий справочник закладок (`Настройка системы \ Сущности > Закладки детализации`), в котором настраивается:

- имя выборки
- отображение
- наименование
- условие вывода
- иконка
- перечень классов, для которых возможен вывод закладки.

Можно регистрировать закладки в коде *Регистрация в коде*

Связь с типом объекта осуществляется по списку допустимых классов. Чтобы включить отображение закладки для типа объекта нужно:

1. Откройте `Настройка системы \ Сущности > Типы объектов > Типы объектов`
2. Выберите настройку `Закладки типа объектов`
3. Отобразите все закладки
   Для этого на панели фильтрации закладок снимите флажок `Отображать неактивные`
4. Выберите требуемую закладку
5. Установите на ней флажок «Активна»

Можно регистрировать закладки для типа в коде *Регистрация в коде*

По умолчанию в разметке выборки Avm сервис настройки закладок не используется. Программист должен указать для нужного отображения динамическое получение закладок от типа объекта.

Существует два варианта вывода детализации фрейма: в виде закладок и в виде списка

Детализация виде закладок

Набор закладок отображается либо снизу либо справа от карточки или списка

Для подключения в Avm нужно указать выборку с динамическими закладками

```
<tabComposer>
        <frame filter.isVisible="false">
                <card>
                        <layout>
                                <hBox>
                                        <attr name="sNumber"/>
                                        <attr name="sCaption"/>
                                        <attr name="idPrjVerHL"/>
                                </hBox>
                                <hBox>
                                        <attr name="idDepOwnerHL"/>
                                        <attr name="idObjectTypeHL"/>
                                        <attr name="idStateHL"/>
                                </hBox>
                        </layout>
                </card>
        </frame>
        <tabItems isVisible="true"
                  selection="gtk-ru.bitec.app.btk.Btk_ObjectTypeTabAvi"
                  representation="List_Tab"
                  selection.selectionAttr="SSELECTIONNAME"
                  selection.representationAttr="SREPRESENTATIONNAME"
                  selection.captionAttr="SCAPTION"
                  selection.imageIndexAttr="NIMAGE"
                  selection.paramsAttr="JSONPARAMS"

                        />
</tabComposer>
```

Детализация в виде списка закладок

Набор закладок отображается в виде списка закладок, справа от которого выводится содержимое текущий закладки. Для подключения в Avm нужно указать выборку с динамическим содержимым.

```
<dynamicComposer>
        <frame filter.isVisible="false"
               header.isWide="true"
               toolBar.isCaptionsVisible="true">
                <card useMasterData="false"/>
        </frame>
        <dynamicItems masterAlign="top">
                <dynamicItem align="client"
                             representation="DynList"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                selection="gtk-ru.bitec.app.btk.Btk_ObjectTypeTabAvi"/>
        </dynamicItems>
</dynamicComposer>
```

9.3.2 Печатные формы

Позволяет настраивать список печатных форм для типа объекта. Настроенные печатные формы будут доступны под операцией печати для карточки объекта.

Печатные формы добавляются из общего справочника и могут быть активированы для типа объекта. Доступ к формированию печатной формы может регулироваться состоянием объекта.

9.3.3 Переходы состояний

Состояния класса позволяют управлять бизнес-логикой объекта: доступ на редактирование атрибутов, возможность выполнения операций в зависимости от состояний и т.д.

Настройка состояний подразумевает наличие в классе атрибута, ссылочного на состояние и типа объекта.

```
<attr name="idState"
      attribute-type="Long"
      caption="Состояние"
      order="40"
      type="refObject"
      ref.class="Btk_ClassState"/>
```

Для класса, имеющего атрибут, ссылочный на состояние, необходимо определить перечень возможных для него состояний.

Список состояний можно настроить в карточке класса, либо вручную, через метод `*Api().regState` *Регистрация в коде*. При создании состояний необходимо помнить, что у класса должно присутствовать ровно одно начальное состояние.

Возможные переходы между состояниями настраиваются для типа объекта на закладке `Переходы состояний`. Можно регистрировать переходы состояний для типа в коде *Регистрация в коде*.

Сеттеры состояний никак не проверяют эти переходы, они влияют только на выпадающий список состояний.

Выборка для выпадающего списка состояний класса, в которые возможен переход от текущего: `gtk-ru.bitec.app.btk.Btk_ClassStateAvi#Lookup_Class`

Пример разметки атрибута в Avm:

```
<attr name="idStateHL" caption="Состояние" order="30.1" isLastInLine="false">
    <editor>
        <lookup lookupQuery="gtk-ru.bitec.app.btk.Btk_ClassStateAvi#Lookup_Class"
                changeableAttr="idState"
                isLookupLazyLoad="true" lookupKeyAttr="id" lookupListAttr="sHeadLine"/> ␣
↪
    </editor>
    <card isControlWidthFixed="true" controlWidth="60"/>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    <ref class="Btk_ClassState"/>
</attr>
```

Методы для работы с состояниями:

```
/**
* Метод ищет состояние по сист. имени и сист. имени класса, к которому оно относится
* @param spSystemName  Сист. имя состояния
* @param spMasterClass Сист. имя класса, к которому относится состояние
* @param bpRaiseErr    При значении = 1.пп будут создаваться исключения, если не найден␣
↪класс, либо нужное состояние
*/
Btk_ClassStateApi().findByNameAndClass(spSystemName:NString, spMasterClass:NString,␣
↪bpRaiseErr:NNumber = 0.nn): NLong

/**
* Возвращает начальное состояние для класса
* @param idpMasterClass идентификатор класса
*/
Btk_ClassStateApi().getClassStartState(idpMasterClass: NLong) :NLong

/**
    * Возвращает номер состояния класса по его id
    */
Btk_ClassStateApi().getOrder(idp: NLong): NNumber
```

В прикладном коде не всегда удобно проверять ссылку на состояние, поэтому у каждого состояния есть номер. Фреймворк позволяет сохранять номер состояния в отдельном атрибуте.

Если в классе объявлен специальный атрибут с признаком `isStateMC=true`, фреймворк автоматически будет сохранять в него номер текущего состояния.

```
<attr name="idStateMC"
      attribute-type="Number"
      caption="Состояние"
      order="51"
      type="basic"
      isVisible="false"
      isStateMC="true" />
```

История состояний

Для классов, имеющих атрибут состояния (`idState`) по умолчанию, ведется история состояний. Данные хранятся в классе `Btk_ObjectStateHistory`. Через dpi-сеттер состояния создаются записи в журнале изменения состояний.

Для отключения истории состояний необходимо воспользоваться атрибутом `isStateHistoryEnabled` тега `class`.

Для просмотра истории состояния объекта:

1. Откройте «Информацию об объекте»

2. Перейдите в закладку «История состояний» Откроется универсальная закладка `Btk_ObjectStateHistoryAvi.List_GidObj`, которая отображает историю состояний объекта. Работает от супер-параметра, в котором указан gid-объекта.

Возможных параметры закладки `Btk_ObjectStateHistoryAvi.List_GidObj`:

- `sGidAttrName` - имя атрибута, в котором хранится gid-объекта из мастер выборки.
  По умолчанию gid.
- `bWithStack` - признак необходимости отображения стека вызова.
  Доступные значения: 0, 1. По умолчанию 0.

Пример использования параметров, при подключении закладки через разметку выборки:

```
<tabItem selection="gtk-Btk_ObjectStateHistoryAvi"
         representation="List_GidObj"
         caption="История состояний">
    <params>
        <param name="sGidAttrName"
               value="gid"
               dataType="String"/>
        <param name="bWithStack"
               value="1"
               dataType="Number"/>
    </params>
</tabItem>
```

Настройка вызова процедур на переход между состояниями

В разрезе типа объекта возможно настроить перечень процедур, которые будут выполнены при переходе из одного состояния в другое.

Есть несколько типов событий, на которые можно настроить вызов произвольных процедур:

- `На вход в состояние` - будут вызваны, если это состояние будет конечным при любом переходе. Настраивается в детализации к состоянию.
- `На выход из состояния` - будут вызваны, если это состояние будет начальным при любом переходе. Настраивается в детализации к состоянию.
- `На переход` - будут вызываны, если будет осуществлен именно этот переход. Настраивается в детализации к переходу.

Порядок вызова событий:

1. Выполнение процедур на выход из состояния
2. Выполнение процедур на переход
3. Выполнение процедур на вход в состояние

Доступные переменные в `jexl`-скрипте:

- `rop` - `rop` объекта, для которого была вызвана смена состояния.
- `idOldState` - старое состояние объекта
- `idNewState` - новое состояние объекта

9.3.4 Объектные характеристики

Позволяет настроить вывод объектных характеристик на каждый тип объекта. На закладке характеристик учитывается настройка объектных характеристик на тип объекта.

9.3.5 Пообъектные права по умолчанию

На закладке можно настроить права доступа по умолчанию для пообъектного режима доступа. Настройка происходит в списки с атрибутами:

- Пользователь
- Роль
- Вид роли ОФС
- Группа доступа

9.4 Подключение индивидуальных настроек объекта

Настройки типа объекта могут быть расширены индивидуальными классами настроек. Такие настройки могут включать или отключать бизнес-логику, управлять фреймами, настраивать права доступа и т.д. в зависимости от поставленной задачи. Индивидуальная настройка добавляются в список настроек для типа документа.

Интерфейс доступен из приложения `Настройка системы`, меню: `Сущности > Типы объектов > Настройка детализации типов объектов`.

При создании собственной настройки для типа объекта можно указать список классов, у которых данная настройка должна выводиться. Если список классов пуст, то настройка выводится у всех типов объекта.

Также можно указать список подклассов, для которых требуется выводить настройку. Если список подклассов не указан, ограничения по подклассам не применяются.

9.5 Регистрация из кода

9.5.1 Регистрация закладок

Описание метода `Btk_TabApi().register`:

```
/**
    * Процедура для регистрации новой строки
    *
    * @param spSystemName  Необязательный параметр - Системное имя
    * @param spCaption     Название закладки
    * @param spSel         Выборка
    * @param spRep         Отображение
    * @param idprefClass   Необязательный параметр - Класс на который регистрируется␣
↪закладка
    * @param spDescription Необязательный параметр - Описание
    * @param spCondition   Необязательный параметр - Условие появления
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    * @param spManualFunction Необязательный параметр - Функция получения списка␣
↪закладок
    * @param idpDataClass  Необязательный параметр - Класс отображаемых в закладке␣
↪данных(для универсальных коллекций)
    * @param spMasterAttr  Необязательный параметр - Атрибут с мнемокодом атрибута,␣
↪через который класс из закладки будет ссылаться на мастера
    * @param npImage       Необязательный параметр - Картинка по умолчанию
    */
```

Пример ручной настройки Api класса `Stm_ActInApi.scala`:

```
def regTab(): Unit = {
  session.commit()
  session.setDefaultUOWEditType(RWSharedUOWET)

  Btk_TabApi().register(
    spCaption = "Товарные позиции"
    , spSel = "gtk-Stm_ActInDetAvi"
    , spRep = "List_idDoc"
    , idprefClass = idClass
  )

  Btk_TabApi().register(
    spCaption = "Складские ордера"
    , spSel = "gtk-Stk_WarrantInAvi"
    , spRep = "LinkDocs"
    , idprefClass = idClass
  )
  session.commit()
}
```

Пример регистрации скрипта в тэге dbData для odm класса `Stm_ActIn`:

```
<script name="regTab" version="1">
    <install>Stm_ActInApi.regTab();</install>
</script>
```

9.5.2 Регистрация состояний

Описание метода `Btk_ClassStateApi().register`:

```
/**
* Метод регистрирует состояние для класса по сист. имени. Если остальные атрибуты не␣
↪переданы, они не изменяются
* Переданные значения параметров записываются в объект, если он найден по id класса и␣
↪сист. имени состояния
* @param idpMasterClass Класс, к которому относится состояние
* @param spSystemName   Сист. имя состояния - должно быть уникально в рамках класса, к␣
↪которому относится состояние
* @param spCaption      Необязательный параметр - наименование
* @param bpStartState   Необязательный параметр - признак стартового состояния
```

(continues on next page)

(продолжение с предыдущей страницы)

```
* @param npOrder        Необязательный параметр - порядковый номер состояния в классе
*/
```

Кроме того есть перегруженная версия метода, где первый параметр – сист.имя класса

```
/**
* @param spMasterClass  Мнемокод класса, для которого регистрируем состояния
*/
```

Пример ручной настройки Api класса `Stm_ActInApi.scala`:

```
def regState: Unit = {
  session.commit()
  session.setDefaultUOWEditType(RWSharedUOWET)
  //Регистрация состояний
  Btk_ClassStateApi().register(
    idpMasterClass = idClass,
    spSystemName = "Forming",
    spCaption = "Формируется",
    bpStartState = 1.nn,
    npOrer = 100.nn)
  session.flush() //чтоб было начальное состояние
  Btk_ClassStateApi().register(
    idpMasterClass = idClass,
    spSystemName = "Executing",
    spCaption = "Исполняется",
    bpStartState = 0.nn,
    npOrer = 200.nn)
  Btk_ClassStateApi().register(
    idpMasterClass = idClass,
    spSystemName = "Done",
    spCaption = "Выполнена",
    bpStartState = 0.nn,
    npOrer = 300.nn)
  session.commit()
}
```

Пример регистрации скрипта в тэге dbData для odm класса `Stm_ActIn`:

```
<script name="regState" version="1">
    <install>Stm_ActInApi.regState();</install>
</script>
```

9.5.3 Регистрация подкласса, типа объекта, закладок для типа и переходов состояний

Пример ручной настройки Api класса Stm_ActInApi.scala:

```
def regObjectType(bpNeedRefresh: Boolean = true): Unit = {
  session.commit()
  Btk_Pkg().setRWSharedUOWEditType()
  val idvInvoice = Btk_SubClassApi().register("invoiceIn", "Приходная накладная",␣
↪idClass, "").get(_.id)
  session.flush()

  if (bpNeedRefresh || Btk_ObjectTypeApi().findByMnemoCodeAndClass("Stm_ActInInvoice",␣
↪idClass).isNull) {
    //регистрируем тип
    val idvObjectType = Btk_ObjectTypeApi().register(
      spCode = "Stm_ActInInvoice".ns
      , spCaption = "Приходная накладная".ns
      , spShortCaption = "Приходная накладная".ns
      , idpRefClass = idClass
      , idpSubClass = idvInvoice
      , bpIsDefault = 1.nn
    )
    //ищем закладку и привязываем к типу
    var idvTab = Btk_TabApi().findByMnemoCode("Stm_ActInDetAvi.List_idDoc")
    if (idvTab.isNotNull)
      Btk_ObjectTypeTabApi().registerTab(
        idpObjectType = idvObjectType
        , idpTab = idvTab
        , spCaption = "Товарные позиции"
        , npOrder = 10.nn
      )

    idvTab = Btk_TabApi().findByMnemoCode("Stk_WarrantInAvi.LinkDocs")
    if (idvTab.isNotNull)
      Btk_ObjectTypeTabApi().registerTab(
        idpObjectType = idvObjectType
        , idpTab = idvTab
        , spCaption = "Складские ордера"
        , npOrder = 20.nn
      )
    //регистрируем переходы состояний для типа
    Btk_StateChangeApi().registerForObjectType(idvObjectType, List(
      "Forming" -> "Executing",
      "Executing" -> "Done",
      "Executing" -> "Forming",
      "Done" -> "Forming",
      "Done" -> "Executing"
    ))
  }
  session.commit()
}
```

Пример регистрации скрипта в тэге dbData для odm класса Stm_ActIn:

```
<script name="regObjectType" version="1">
    <depends>
        <dep on="Stm_ActIn.regState"/>
        <dep on="Stm_ActIn.regTab"/>
    </depends>
    <install>Stm_ActInApi.regObjectType(false);</install>
</script>
```

9.6 Общая схема классов

Часть III

Выборки

Глава 10

Выборка

Выборка определяет правило получения, отображение данных и обеспечивает взаимодействие с пользователем. Выборки содержат основную часть интерактивной бизнес логики.

Выборка определяет:

- Способ получения данных
- Способ отображения данных пользователю
- Бизнес логику обработки пользовательских действий

Выборка может создаваться от класса с использованием кодо-генерации или вручную.

Пользовательский интерфейс приложения является совокупностью экземпляров отображений выборок.

10.1 Отображения

Отображение – группирует бизнес логику выборки в зависимости от способа представления данных.

Стандартные отображения для класса:

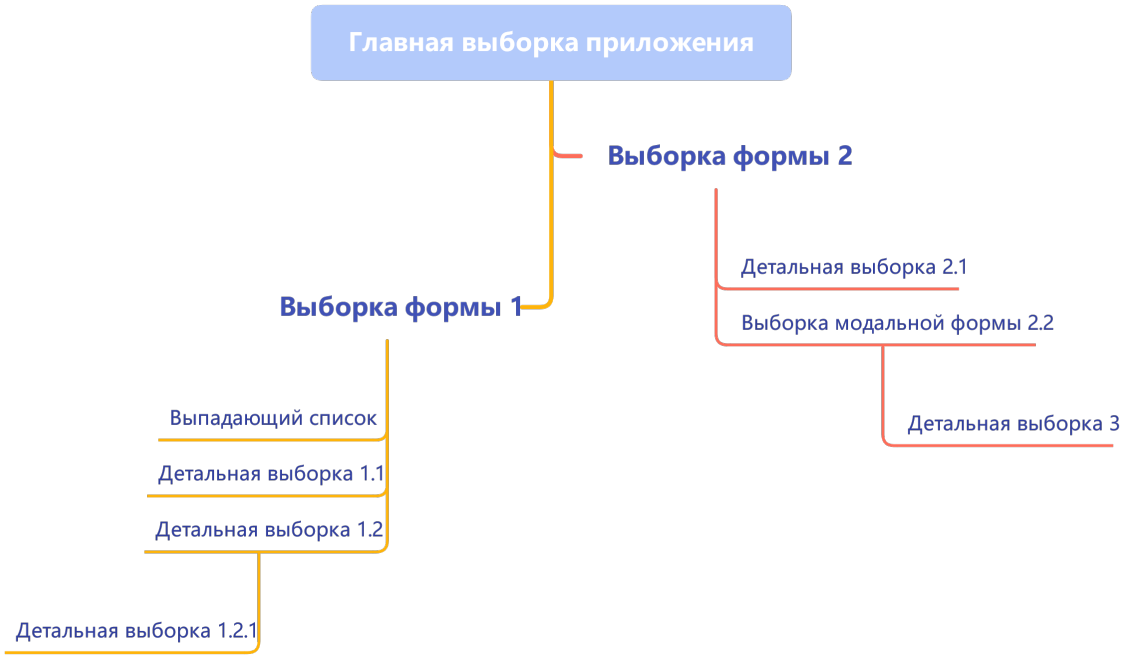
- `Default` – отображение по умолчанию
  В данном отображении определена функциональность общая для данных выборки. Остальные отображения выборки обычно наследуют это отображение.
- `List` – отображение для списка объектов класса
- `Card` – отображение для карточки объекта
- и т.д.

10.2 Отношение экземпляров выборок

Экземпляры выборок, открытые в приложении, связаны друг с другом в отношении `мастер-деталь`. Это означает, что одна выборка всегда является подчиненной к другой выборке. Подчиненная выборка имеет доступ к параметрам и данным мастер выборки.

Корнем выборок экземпляра приложения – является выборка главного меню. Все остальные выборки являются подчиненными к ней.

Выборка являющаяся корнем формы имеет доступ к данным главного меню. Экземпляры выборок внутри формы также выстраиваются в иерархии `мастер-деталь`, в соответствии с бизнес логикой отображения.



10.3 Доступность параметров главных выборок в подчиненных выборках

Получение значения параметра осуществляется с помощью метода `getVar()` и преобразованием результата к нужному типу.

Иерархическая структура создаваемых объектов (выборок) позволяет обеспечивать доступность и передачу параметров по отношению «мастер-деталь», а также производить автоматическое обновление при изменении используемых параметров. Параметры экземпляров выборок – это, прежде всего, атрибуты (поля) их датасета. Также это могут быть дополнительно созданные параметры, или специальные параметры фреймов.

Параметры мастер-выборок автоматически становятся доступны в детальных выборках. При этом, главная выборка является общим, наиболее высокоуровневым мастером, поэтому в ней имеет смысл создавать наиболее общие параметры, используемые всем приложением.

Обращение к параметру непосредственного мастера осуществляется прибавлением к имени параметра префикса `super$`. При этом, если необходимо обратиться к параметру мастер-выборки своей собственной мастер-выборки (т.е. перепрыгнуть через один экземпляр выборки), необходимо использовать префикс `super$super$`. В случае, если при любом из обращений к параметру он не будет найден, произойдет автоматический поиск параметра по дереву отношений «мастер-деталь» выборок вплоть до тех пор, пока мы не найдем этот параметр, или не дойдем до главной выборки приложений. Поэтому, при создании глобальных параметров на уровне главной выборки приложения, для них имеет смысл давать уникальное имя, а затем обращаться к ним из любого места приложения используя один единственный префикс `super$`.

Пример получения строкового параметра из мастер-выборки:

```
getVar("super$sCaption").asNString
```

10.4 Передача параметров в выборку

При открытии выборки, в нее можно передать карту с дополнительными параметрами, используя метод `params()`.

Пример кода:

```
Bs_GoodsAvi.defCard.newForm().params(Map(CardRep.IdItemSharp -> idvGds, CardRep.
↪EditingType -> EditingType.edit)).open()
```

В случае такой передачи параметров, они становятся доступны в открытом отображении.

10.5 Открытие выборок в различных режимах

При открытии выборки в любом из режимов:

- newForm().open()
- newForm().openModal()
- newForm().openLookup()

необходимо учитывать, что родителем для открывшейся выборки будет являться ***_MainMenu, а сама открывшаяся выборка будет главной формой (selection.isMainOnForm = true). Поэтому открывать выборки, в которых установлены зависимости от параметров (`"super$..."`) родительской выборки, без самой родительской выборки нельзя, тк никаким способом в открывающуюся выборку не получится передать подобные параметры.

10.6 Разметка выборки

Фреймворк использует систему декларативной разметки выборок. Это позволяет ускорить разработку визуальных форм т.к. разработчику не требуется заниматься низкоуровневым программированием.

Разметка выборки задает правила отображения данных выборки и ее дочерних элементов в `xml` файле, с расширение `.avm.xml`.

Ключевые элементы разметки:

- Отображения

- Компоновка формы
- Фрейм
  Задает тип представляемых данных (список, дерево и тд.)
- Атрибуты
- Операции
- Фильтры

10.6.1 Разметка карточки. Контейнеры

Существует два режима разметки отображения:

1. Разметка на основе настроек атрибутов - старый режим разметки, основанный на порядковом номере и поле `isLastInLine` атрибутов.
2. Разметка на основе контейнеров - новый режим, основанный на разметке с помощью контейнеров, указанных внутри тэга `<card>`. \

Разметка атрибутов может быть задана с помощью следующих контейнеров:

- *hBox* горизонтальный контейнер: объекты внутри располагаются друг за другом горизонтально.
- *vBox* - вертикальный контейнер: объекты внутри располагаются друг за другом вертикально.
- *vGroup* - группа объединения: по сути тот же вертикальный контейнер, но визуально обводит содержимое рамкой и может иметь наименование.
- *vSection* - сворачиваемая секция: вертикальный контейнер, который имеет наименование и функцию сворачивания, которая скрывает содержимое. Можно указать, как по умолчанию будет отображаться секция - свернутой или развернутой.

Контейнеры могут быть вложены друг в друга.

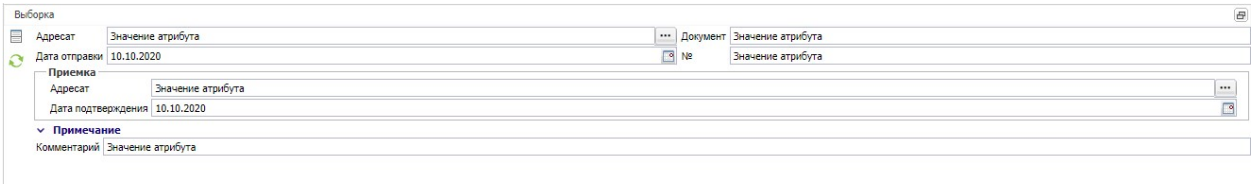
Пример разметки с помощью контейнеров:

```
<representation editMode="edit" name="Card" stdFilter.isAvailable="false">
        <layout>
            <simpleComposer>
                <frame filter.isVisible="false">
                    <card>
                        <layout>
                            <vBox>
                                <hBox>
                                    <vBox>
                                        <attr name="idCustomerHL"/>
                                        <attr name="dOutDate"/>
                                    </vBox>
                                    <vBox>
                                        <attr name="gidSrcDoc" visible="true"/>
                                        <attr name="nOrder"/>
                                    </vBox>
                                </hBox>
                                    <vGroup caption="Приемка">
                                        <attr name="idCustomerHL"/>
                                        <attr name="dExecuteDate"/>
                                    </vGroup>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                            </vBox>
                            <vSection caption="Примечание" collapsed="false">
                                <attr name="sComment" caption="Комментарий"/>
                            </vSection>
                        </layout>
                    </card>
                </frame>
            </simpleComposer>
        </layout>
    <attributes/>
<operations/>
</representation>
```



10.6.2 Использование шаблонов в разметке.

Файлы разметки поддерживают язык шаблонов Thymeleaf. Шаблоны позволяют вставлять в выборку повторно используемые фрагменты. К примеру использовать общий фильтр в разных отображениях.

Для использование шаблонов языка шаблонов необходимо подключить в выборку пространство имен: `xmlns:th="http://www.global-system.ru/xsd/global3-view-template-1.0"`

10.6.3 Наследование разметки

Для наследования разметки другой выборки необходимо:

1. Подключить пространство имен языка шаблонов
2. Добавить к тегу view указание о наследовании другой разметки `th:extends="Some_Other.avm.xml"`

Полный пример xml с наследованием:

```
<view xmlns="http://www.global-system.ru/xsd/global3-view-1.0"
      xmlns:th="http://www.global-system.ru/xsd/global3-view-template-1.0"
      name="RplTst_AllDbTypes" th:extends="RplTst_AllDbTypes.dvm.xml">
  <representation name="Default"/>
</view>
```

Алгоритм наследования выборки:

1. При отсутствие скопировать отображения предка в наследника
   Копирование происходит в случаи если в наследники нет данного отображения
2. Произвести наследование отображений
   Наследование производится в случаи если и в предке и в потомке есть отображение с тем же именем.
3. Выполнить оставшиеся команды языка разметки

Алгоритм наследования отображений:

1. При отсутствие скопировать
   - levelGroups
   - bandGroups
   - layout
   - filter
2. Добавить отсутствующие атрибуты
3. Добавить отсутствующие операции

10.7 Операции

Выборка взаимодействует с программным окружением системы с помощью операций. Операция – это некая обработка того или иного события, вызванного пользователем или каким-то процессом. Операции бывают служебными, выполняя логику обработки событий, или пользовательскими, срабатывающими в результате явного действия пользователя.

10.7.1 Классификация операций

- *Базовые операции*
  Операции, предназначенные для работы с набором данных. Они отвечают за добавление, удаление, обновление записей, получения данных, навигацию по датасету и т.п. Эти операции, как правило, уже определены на уровне базовых выборок-предков. Некоторые из этих операций срабатывают автоматически на выполнение определенных пользовательских действий (`BeforeEdit` при начале редактирования, `AfterEdit` при завершении, `CheckWorkAbility` при перемещении с записи на запись)
- *Служебные операции*
  Операции, выполняющие строго определенные вспомогательные действия
- *Системные операции*
  Операции, создающиеся автоматически
- *Сеттеры*
  Операции предназначены для изменения значений атрибутов (полей) выборки. При вводе значения в какое-то поле формы вызывается связанная с данным полем операция (сеттер), которая по умолчанию вызывает серверные методы изменения значений атрибутов класса. Сеттеры для атрибутов класса создаются автоматически.
- *Операции фильтров*
  Операции, необходимые для работы с фильтрами.
- *Пользовательские операции* Операции, создаваемые разработчиком самостоятельно и позволяющие осуществлять любой набор необходимых операций, не только осуществляемых над данными, но и являющихся целиком клиентскими (например, вызов другой формы, сообщения и т.п.).

10.7.2 Базовые операции выборки

С точки зрения автоматического срабатывания операций на различные системные действия, ключевым параметром операций является их системное имя. Базовые и служебные операции выборки имеют предопределенные системные имена.

onRefresh

Операция, загружающая данные в датасет. При повторном вызове производит обновление всего набора данных (`refresh`).

onRefreshItem

Операция обновления одной (текущей) записи.

onRefreshExt

Операция с дополнительным запросом к базе для получения полей-заголовков ссылочных объектов. Вызывается только для объектных запросов в onRefresh/onRefreshItem.
Для использования в запросе атрибутов класса необходимо передать их в блоке `with`. При этом обязательно для НЕ `Long` атрибутов нужно явно указывать тип в аннотации `/*@...*/`: \

- Для `Varchar` атрибутов - «String», «NString» или «varchar», например - `/*@NString*/`
- Для `Number` атрибутов - «Number», например - `/*@Number*/`

В противном случае атрибуты попытаются преобразоваться к `bigint` типу.

beforeEdit

Операция, выполняемая перед началом редактирования записи для выполнения подготовительных действий перед началом редактирования. Автоматически выполняется в момент начала внесения изменений в объект.

afterEdit

Операция финальной проверки ввода записи. Срабатывает автоматически при попытке перейти с отредактированной записи на другую запись выборки или при закрытии выборки.

insert

Операция вставки новой записи в выборку.

delete

Операция удаления записи.

checkWorkAbility

Операция, предназначенная для осуществления проверок изменения состояния операций выборок, доступности редактирования ее атрибутов. Вызывается автоматически при переходе с записи на запись, при открытии выборки, а также после выполнения операции, для которой выставлен флаг «выполнять операцию применимости после выполнения операции».

onLoadMeta

Операция, срабатывающая при загрузке метаданных. Т.к. метаданные загружаются только один раз, при первом открытии экземпляра выборки на фрейме, то и данная операция срабатывает только один раз.

onUnloadMeta

Операция, срабатывающая при выгрузке метаданных.

beforeOpen

Операция, срабатывающая перед открытием выборки. Данная операция рекомендуется для создания дополнительных параметров, используемых в выборке.

afterOpen

Операция, срабатывающая сразу после открытия выборки. Необходимо использовать для переопределения наименований столбцов выборки и т.д.

onShow

Операция, срабатывающая сразу после того, как отрисован интерфейс формы и были созданы элементы формы.

onControllerCreated

Операция, срабатывающая сразу после того, был создан фрейм отображения выборки. Эта операция аналогична OnShow, но срабатывает в том числе и при открытии интерфейсных элементов формы, которые были скрыты при открытии формы (например, закладки, которые не были первыми в перечне закладок).

10.7.3 Служебные операции выборки

saveForm

Операция сохраняет данные формы.

cancelForm

Операция отменяет изменения на форме.

closeFormOk

Операция закрытия формы с подтверждением выбора

closeFormCancel

Операция закрытия формы по кнопке «Выход» или по кнопке закрытия окна «крестик»

beforeCloseForm

Операция срабатывает перед закрытием формы.

afterCloseForm

Операция срабатывает после закрытия формы.

onCloseFormQuery

Операция вызывается в начале процесса закрытия формы и позволяет отменить закрытие.

10.7.4 Предопределенные операции выборок

applyUniFilter

Операция применяет настроенные условия фильтрации, выполняется перезапрос данных с наложенными условиями фильтрации.

resetUniFilter

Операция отменяет все наложенные на выборку фильтры

clearUniFilter

Операция отчищает все условия фильтрации.

showAuditObject

Операция вызывает окно аудита выполняемых пользователями действий в БД, с наложенным фильтром по текущему объекту.

showAboutObject

Операция открывает окно системной информации об объекте.

copyObject

Операция копирования объекта

cardEdit

Операция открытия объекта в карточке, предварительно запросив, какую использовать выборку и отображение.

allowEdit

Разрешение/запрещение редактирования объектов в списке. По умолчанию операция доступна для редактируемых списков.

showTab

Операция открытия/закрытия детальной части

10.7.5 Сеттеры изменения атрибутов

Операции установки значений атрибутов (так называемые сеттеры) используются для того, чтобы установить определенное значение в поле записи объекта, соответствующее его типу. Имя сеттера состоит из префикса `set` и имени атрибута `attributeName`: `setattributeName`.

10.7.6 Специальные операции контролов

onFocusedFieldChanged

Это специальная операция, срабатывающая после перехода фокуса ввода в списке или дереве с одного столбца на другой, а также при переходе с контрола на контрол в карточке или панели фильтров.

onFocusedCellChanged

Это специальная операция, срабатывающая после перехода фокуса ввода в списке с одной ячейки на другую.

onFrameActivated

Специальная операция, срабатывающая при переходе фокуса ввода в грид, дерево, карточку.

10.7.7 Способы визуализации операций

Существует 3 основных способа визуального отображения операций:

- На тулбаре фрейма
- В выпадающем (контекстном) списке (popup меню)
- В качестве пункта главного меню (доступно только для выборок приложений главного меню)

Для настройки отображения операции каждым из способов существует собственная настройка при помощи соответствующих свойств операции.

10.8 Диалоги

Диалоги позволяют задавать вопросы пользователю и получать от него ответы на эти вопросы.

Для доступа к методам диалогов используется специальный объект `ru.bitec.app.gtk.gl.Dialogs`, который доступен в `Avi` через переменную `dialogs`.

Кроме стандартных диалогов, есть возможность открыть форму в режиме выбора (`openLookup`) и после обработать выбранное пользователем значение, что так же из себя представляет специализированный вид диалогов.

10.8.1 Стандартные диалоги

showMessage

Выводит на экран текст в модальном окне.

Пример использования:

```
dialogs.showMessage("Текст, который увидит пользователь")
```

showMsgDialog

Открывает диалог указанного типа с сообщением и указанным перечнем кнопок.

Возможные типы диалогов см. в `ru.bitec.app.gtk.gl.msgdlg.MsgDlg`

Возможные типы кнопок см. в `ru.bitec.app.gtk.gl.msgdlgbutton.MsgDlgButtong`

Пример использования:

```
val dlgRes = dialogs.showMsgDialog(
  //тип диалога "Подтверждение"
  MsgDlg.confirmation,
  //текст диалога
  "Включить автонумерацию?",
  //список кнопок: Да, Нет
  List(MsgDlgButton.yes,MsgDlgButton.no)
)
if (dlgRes == MsgDlgButton.yes) {
  //обработка нажатия на кнопку "Да"
} else if (dlgRes == MsgDlgButton.no) {
  //обработка нажатия на кнопку "Нет"
}
```

showButtonsDialog

Открывает диалог с произвольными кнопками.

Позволяет формировать диалоги, кнопки которых будут не из перечня стандартных кнопок.

Пример использования:

```
dialogs.showButtonsDialog(
  caption = "",
  text = "Удалить форму или убрать ссылку на вид отчетности?",
  colCount = 2,
  buttons = List(List("Удалить"), List("Убрать ссылку")),
  imageCollectionName = "ToolBarPrimaryHot",
  focusButtonNumber = 0L
) match {
  case 0L =>
    //обработка нажатия кнопки "Удалить"
  case 1L =>
    //обработка нажатия кнопки "Убрать ссылку"
}
```

showPromptDialog

Открывает диалог с запросом ввода строки.

Пример использования:

```
val svDialogRes = dialogs.showPromptDialog("Заголовок окна", "Текст диалога", "Значение
↪по умолчанию").ns

if (svDialogRes.isNotNullOrEmpty) {
  //обработка введенного значения
}
```

showConfirmDialog

Открывает диалог с типом «Подтверждение» и кнопками «Да» и «Нет»

Пример использования:

```
val dialogRes = dialogs.showConfirmDialog("Текст диалога")
if (dialogRes){
  //обработка случая, когда пользователь нажал "Да"
}
```

showInfoForm

Отображает информационное сообщение с переданным текстом, без возможности его закрытия пользователем.

Скрывается при вызове метода `hideInfoForm`.

Применяется, когда в бизнес-логике выполняется продолжительная задача, и требуется проинформировать пользователя об ее выполнении.

Пример использования:

```
dialogs.showInfoForm("Внимание, идет загрузка Госреестра типов СИ")
try {
  //продолжительное действие
  thisApi().download()
} finally {
  dialogs.hideInfoForm()
}
```

hideInfoForm

Скрывает информационное сообщение, см. описание метода `showInfoForm`

withInfoForm

Метод включает в себя открытие информационного окна, и его закрытие после завершение выполнения переданной функции.

Пример из описания метода `showInfoForm`, можно переписать используя этот метод:

```
dialogs.withInfoForm("Внимание, идет загрузка Госреестра типов СИ") {
  //продолжительное действие
  thisApi().download()
}
```

showEditPaintStyleDialog

Вызывает диалог настройки стилей раскраски текстовых полей.

10.8.2 Форма в режиме выбора значения

Одной из типичных задач является предложить выбрать пользователю какой-либо объект и затем сделать обработку его выбора. Например, чаще всего ссылочные поля открывают для выбора список объектов ссылочного класса, а после выбора пользователем значения устанавливают его в текущей форме.

Для реализации такого типа диалогов используется синтаксис открытия формы в режиме `openLookup`.

Пример открытия формы и обработки выбора пользователя:

```
//создаем новую форму
val data = Btk_ClassAvi.listForChoose().newForm()
  //указываем список полей, которые требуется получить из формы
  .results("id" :: "sCaption" :: Nil)
  //открываем форму в режиме выбора
  .openLookup()

//проверяем, что пользователь подтвердил выбор
if (data.getLookupResult eq LookupResult.ok) {
  //получаем значение первого поля из списка results
  val id = data.getData(1, 0)
  //получаем значение второго поля из списка results
  val sCaption = data.getData(1, 1)
}
```

Выбор нескольких строк (мультиселект)

Для того, чтобы в форме в режиме выбора пользователь мог выбрать несколько строк используется опция `useMultiSelect`.

Пример открытия формы в режиме мультиселекта и обработка выбранных значений:

```
//создаем новую форму
val data = Btk_ClassAvi.listForChoose().newForm()
  //указываем список полей, которые требуется получить из формы
  .results("id" :: "sCaption" :: Nil)
  //указываем возможность выбора нескольких строк
  .useMultiSelect
  //открываем форму в режиме выбора
  .openLookup()

//проверяем, что пользователь подтвердил выбор
if (data.getLookupResult eq LookupResult.ok) {
  //цикл по выбранным строкам
  for (i <- 1 to data.size) {
    //получаем значение первого поля из списка results для строки i
    val id = data.getData(i, 0)
    //получаем значение второго поля из списка results для строки i
    val sCaption = data.getData(i, 1)
  }
}
```

10.9 Фрейм

Фреймы — это средство представления информации в выборке.

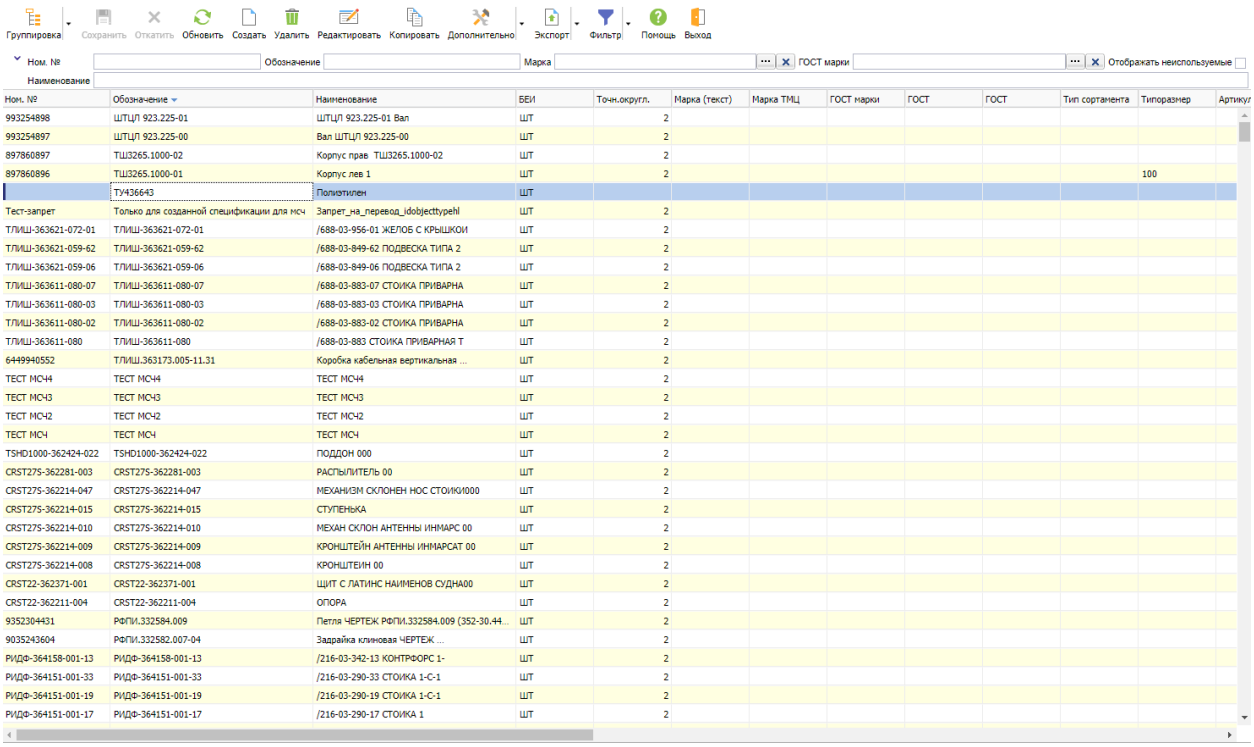
Существующие фреймы по способу представления информации можно разделить на несколько видов:

- для отображения информации в виде карточки объекта
- для отображения списка объектов в табличной форме
- для отображения списка объектов в виде древовидной структуры
- для отображения инфографики
- специализированные (для отображения графической информации, доступа к файлам и т.д.)

10.9.1 Основные типы фреймов

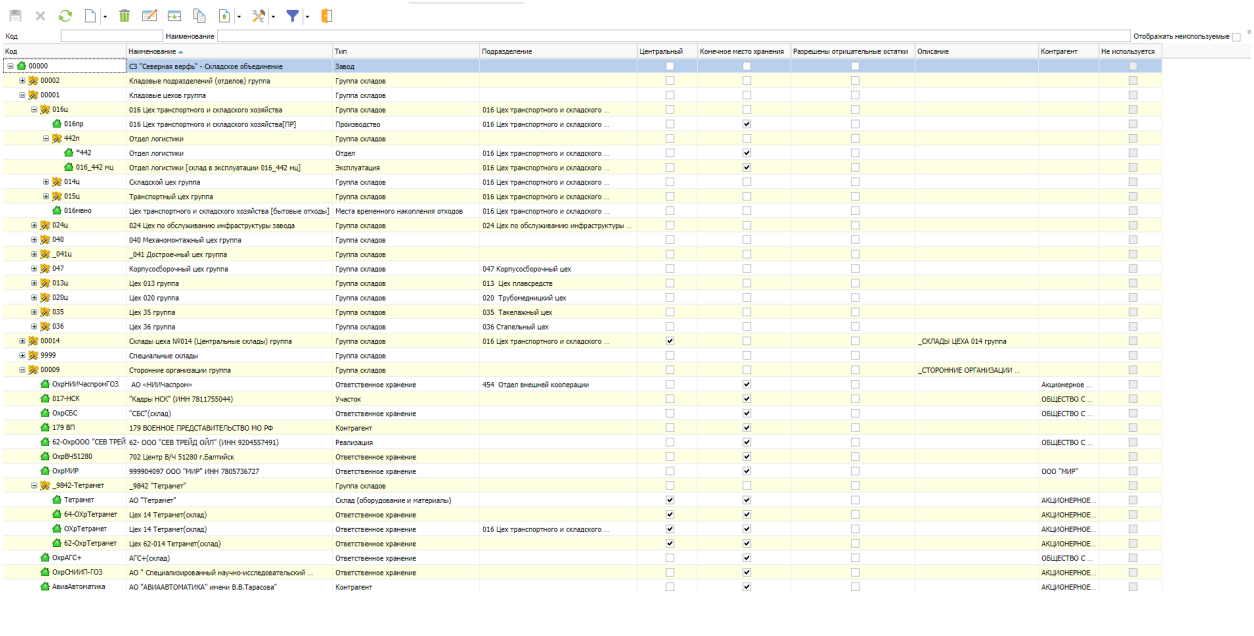
grid

Фрейм для вывода списка объектов класса в табличной форме



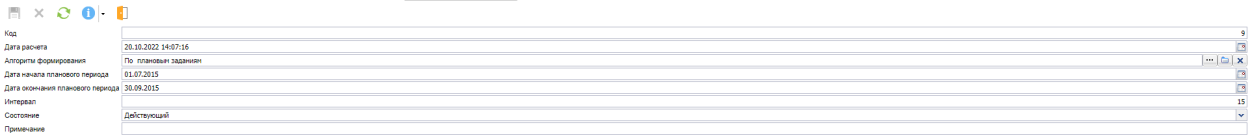
tree

Фрейм для вывода списка объектов класса в древовидной форме



card

Фрейм для вывода всех атрибутов конкретного объекта (Карточка)

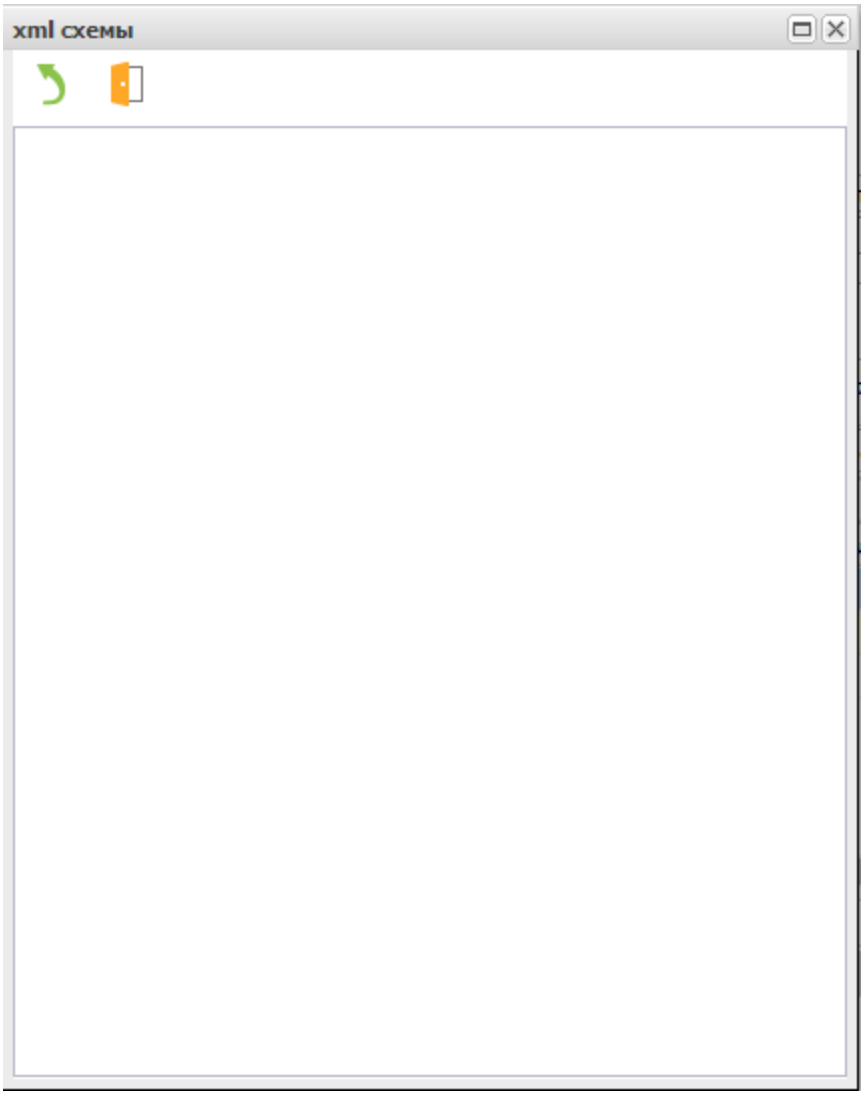


tab

Фрейм с закладками. Выборка фрейма является источником списка закладок. Так же, закладки могут быть указаны статически, в xml-разметке.

memo

Фрейм для ввода большого количества текстовой информации (Memo)



html

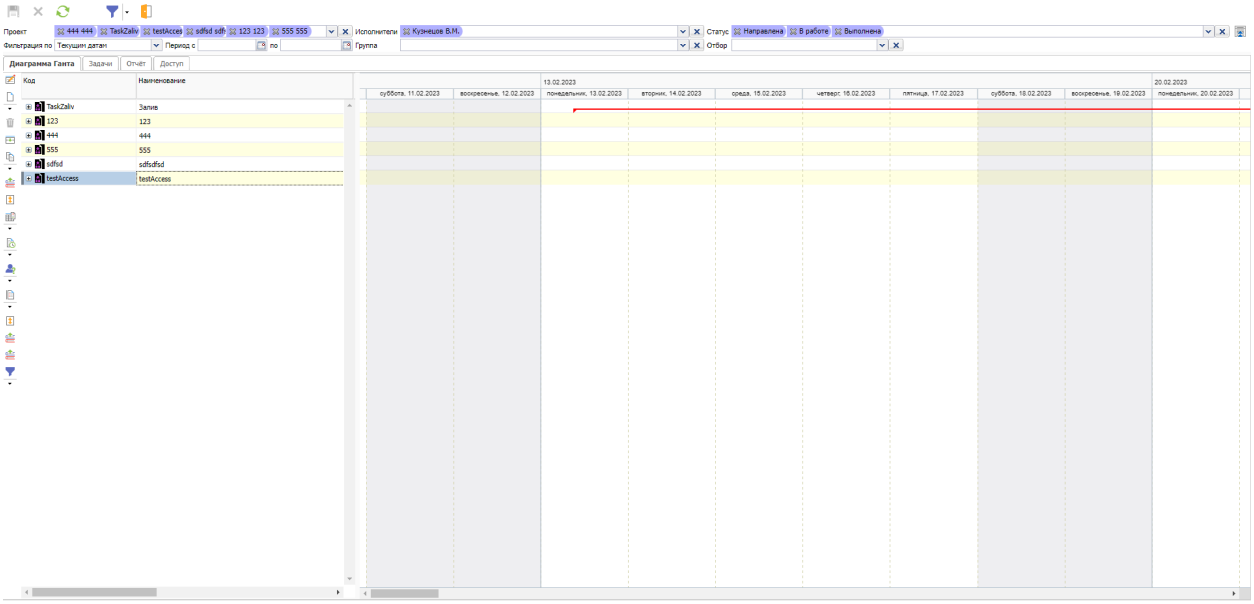
Фрейм, отображающий HTML

image

Изображение

gantt

Фрейм, для построения диаграмм Ганта.

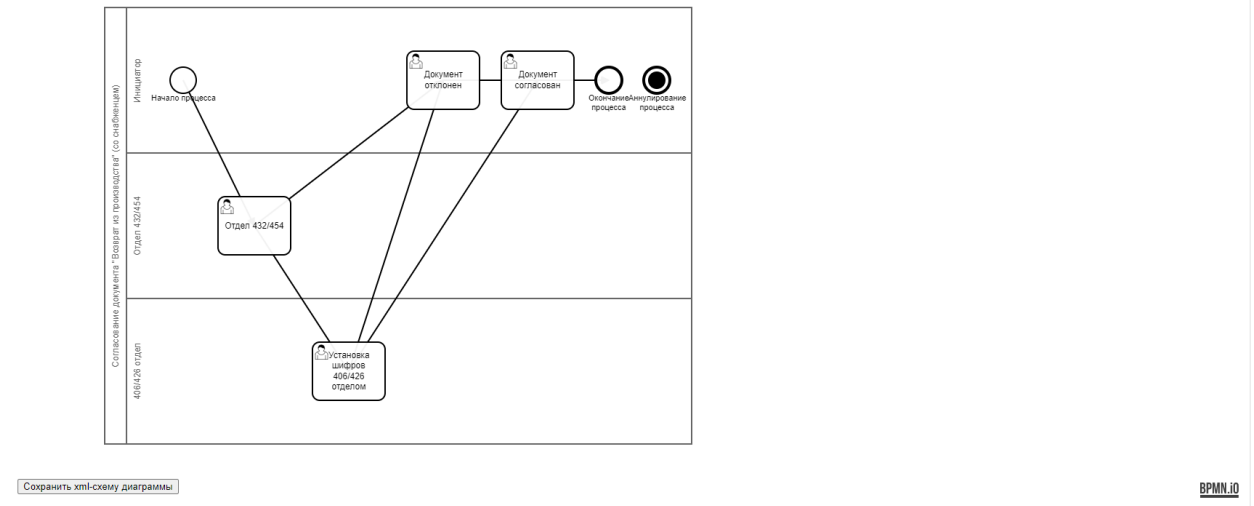


olap

Фрейм для построения сводных (OLAP) таблиц

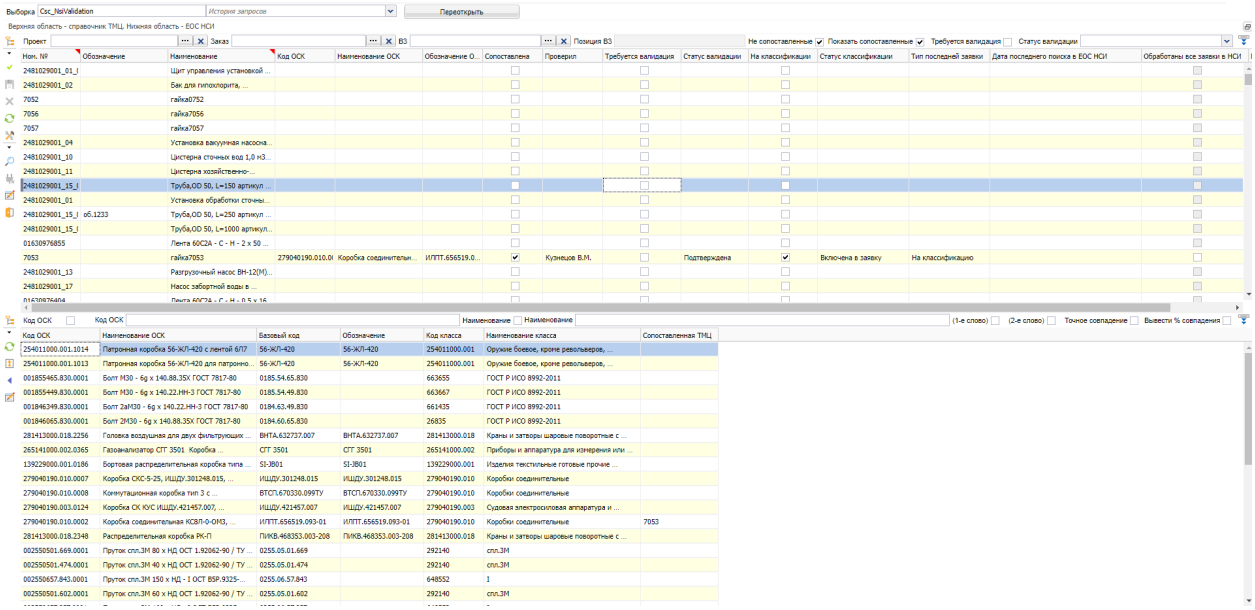
bpmn

Фрейм редактора диаграмм бизнес-процессов. Поддерживает диаграммы BPMN версии 2.0



gridPanel

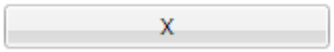
Позволяет размещать детальные фреймы в виде таблицы, для указания ширины и высоты ячеек которой можно использовать как абсолютные, так и относительные величины.



10.9.2 Основные типы редакторов

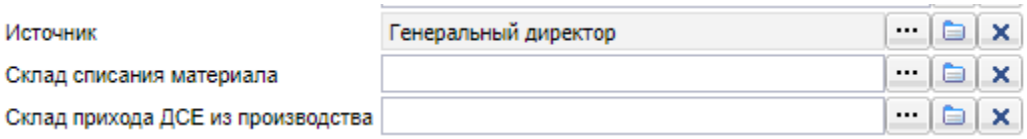
button

Редактор - Кнопка. При нажатии выполнится сеттер соответствующего атрибута.



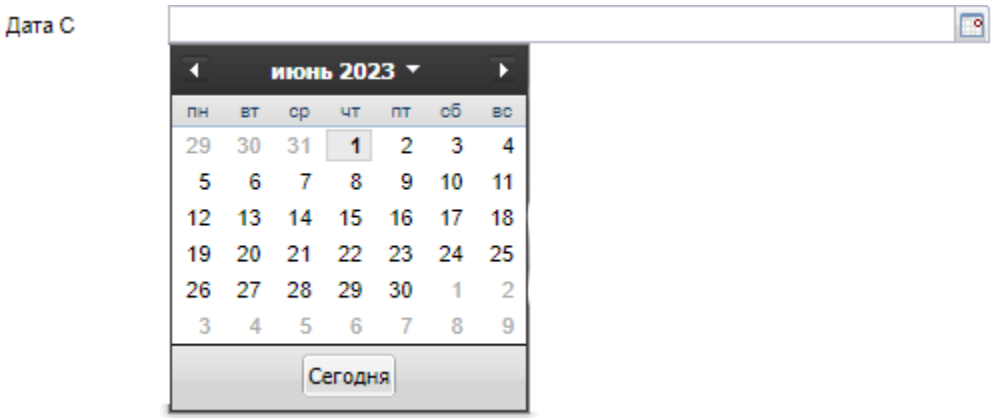
buttonsEdit

Редактор в строке с произвольными кнопками. Если ни одной кнопки для отображения не задано, по умолчанию будет отображена кнопка lookup.



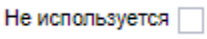
calendar

Редактор - Календарь.



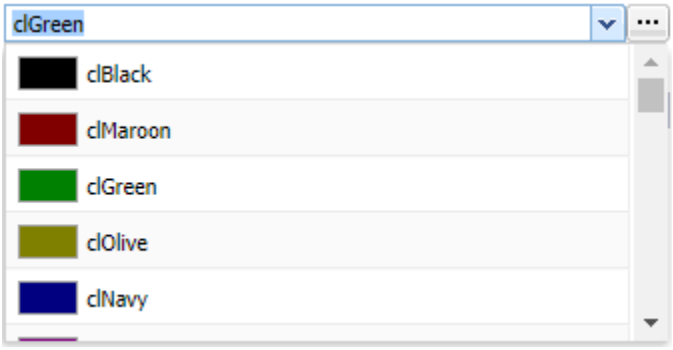
check

Редактор - Чекбокс (Галка).



colorPick

Редактор - Выбор цвета.



combo

Редактор - Фиксированный выпадающий список.

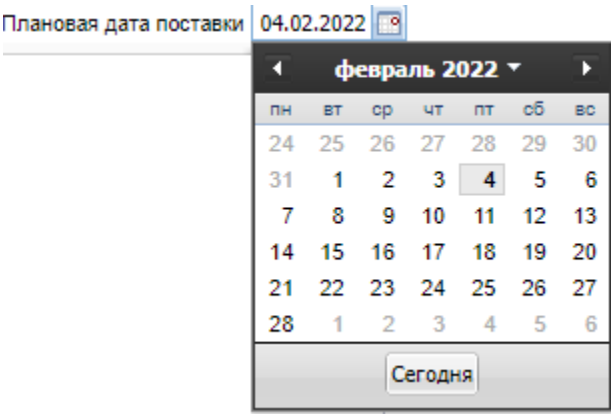
currency

Денежный редактор.

Сумма себестоимости в базовой ва...
1 305 000,00
1 305 000,00
380 833,33
380 833,33
319 000,00
120 000,00
120 000,00
120 000,00
116 087,50
104 583,33
101 750,00
100 000,00
96 000,00
90 000,00
90 000,00

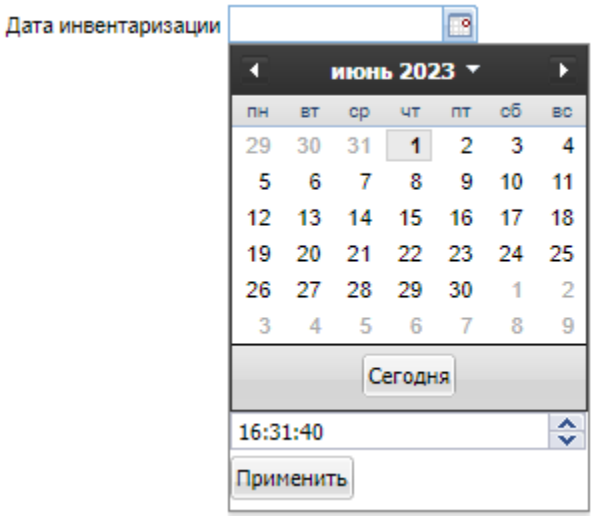
datePick

Редактор даты.



dateTimePick

Редактор даты и времени.



edit

Однострочный редактор текста.

editButton

Редактор: Редактор в строке с кнопкой.

hotKey

Редактор ввода комбинации горячих клавиш.

hyperLink

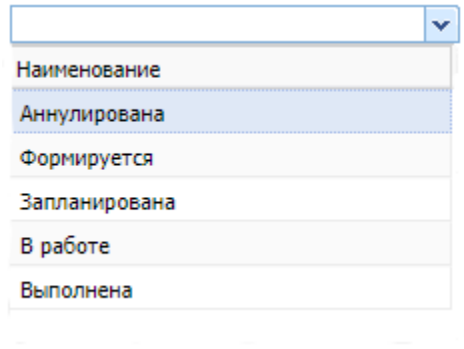
Редактор адреса эл.почты и гипер-ссылок.

icon

Редактор - изображение. Используется для списка и дерева. Отображает изображение из коллекции изображений.

lookup

Редактор - Выпадающий список по запросу. Источником элементов могут являться: выборка или SQL-запрос, указанные в свойствах.

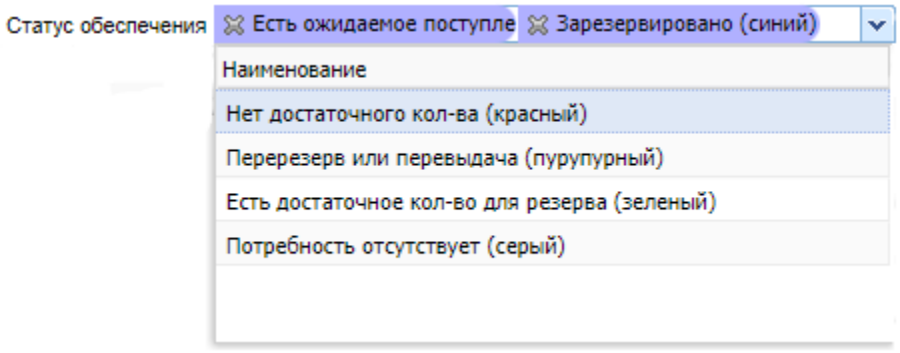


memo

Многострочный редактор. Предназначен для редактирования многострочных текстовых значений без разметки.

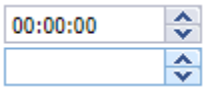
tagLookup

Редактор - Выпадающий список по запросу с возможностью множественного выбора.



timePick

Редактор времени.

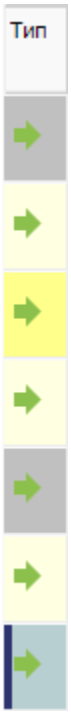


editPassword

Редактор ввода пароля.

imageCollection

Редактор - выпадающий список изображений.



10.10 Мультиселект

Мультиселект - возможность выбрать несколько строк в списках или деревьях.

Для включения мультиселекта в отображении укажите в теге `grid` или `tree` свойство `isMultiSelectEnabled="true"`.

Пример разметки:

```
<representation name="List">
  <layout>
    <simpleComposer>
      <frame>
        <grid isMultiSelectEnabled="true"/>
      </frame>
     </simpleComposer>
  </layout>
</representation>
```

Пример обработки выделенных строк пользователем и получения значений атрибутов этих строк:

```
//цикл по выделенным строкам
for (i <- 0 until selection.selectedRecordsCount()) {
  //получение значения поля с типом NLong
  val id = NLong.fromAny(selection.selectedValueByName("id", i))
  //получение значения поля с типом NNumber
  val nOrder = NNumber.fromAny(selection.selectedValueByName("nOrder", i))
  //получение значения поля с типом NGid
  val gid = NGid(selection.selectedValueByName("gid", i).asInstanceOf[String])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  //получение значения поля с типом NString
  val sCaption = selection.selectedValueByName("sCaption", i).asInstanceOf[String].ns
  //получение значения поля с типом NDate
  val dDate = NDate(selection.selectedValueByName("dDate", i).asInstanceOf[JDate])
}
```

Примечание: Для формы в режиме выбора мультиселект регулируется опцией `useMultiSelect` при создании формы. Подробнее в главе `Диалоги`

10.11 Настройка стилей

Стиль - набор свойств, включающий в себя:

- `Color` - Цвет заднего фона в формате `$00BBGGRR` или Global константой цвета.
- `FontColor` - Цвет шрифта в формате `$00BBGGRR` или Global константой цвета.
- `FontSize` - Размер шрифта натуральным числом.
- `FontItalic` - Курсив. Значение `0` - нет курсива, `1` - есть курсив.
- `FontBold` - Жирность. Значение `0` - не жирный шрифт, `1` - жирный шрифт.
- `FontUnderLine` - Подчеркивание. Значение `0` - нет подчёркивания шрифта, `1` - есть подчёркивание шрифта.
- `FontStrikeOut` - Перечеркивание. Значение `0` - нет перечеркивания шрифта, `1` - есть перечеркивание шрифта.

Стиль задаётся строкой формата:

```
"param1=value1;param2=value2;param3=value3;...;paramn=valuen"
```

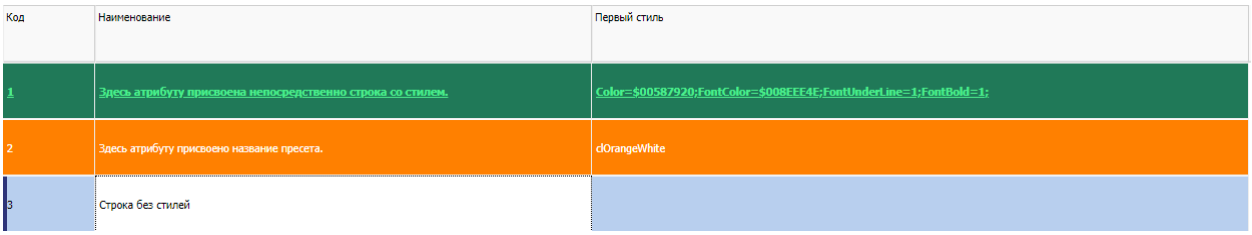
Пресеты стилей хранятся в классе `Btk_Registry`, их можно использовать для типовых задач.

Стили можно применять в отображении к строкам и столбцам.

10.11.1 Применение стиля к строке

Чтобы применить стиль к строке, нужно в теге `representation` присвоить свойству `rowStyleAttr` имя атрибута, в котором будет храниться либо готовая строка со стилем, либо системное имя пресета.

```
<representation name="List" editMode="notEdit" rowStyleAttr="sStyle1">
        <attributes>
            <attr name="sCode" caption="Код" editorType="edit" order="10" isRequired=
↪"true"/>
            <attr name="sCaption" caption="Наименование" editorType="edit" order="20"␣
↪isRequired="true"/>
            <attr name="sStyle1" caption="Первый стиль" editorType="edit" order="40"/>
        </attributes>
    </representation>
```

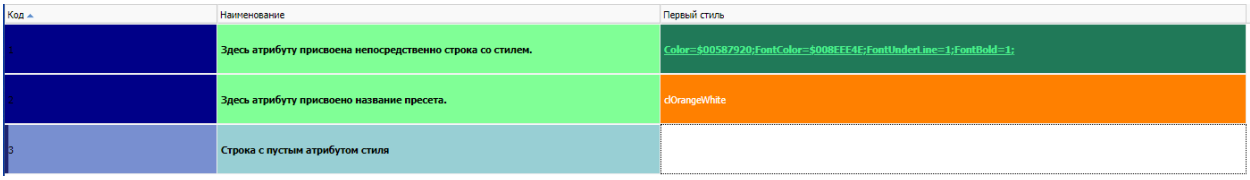


10.11.2 Применение стиля к столбцу

Чтобы применить стиль к столбцу, нужно в теге `attr` создать тег `style` и в нём использовать либо свойство `name`, либо `attr`. В свойстве `name` указывается непосредственно строка стиля или системное имя пресета, в `attr` указывается атрибут, где хранится либо строка, либо системное имя.

```
<representation name="List" editMode="notEdit">
        <attributes>
            <attr name="sCode" caption="Код" editorType="edit" order="10" isRequired=
↪"true">
                <style name="Color=$00880000"/>
            </attr>

            <attr name="sCaption" caption="Наименование" editorType="edit" order="20"␣
↪isRequired="true">
                <style name="BDG_URTotalRow"/>
            </attr>
            <attr name="sStyle1" caption="Первый стиль" editorType="edit" order="40">
                <style attr="sStyle1"/>
            </attr>
        </attributes>
    </representation>
```



Также стиль можно менять в avi используя `selection`

```
selection.attrs("Имя атрибута, на который мы хотим применить стиль").styleAttributeName␣
↪= "Имя атрибута со строкой стиля или с системным именем пресета"
```

10.12 Создание строки стиля. StyleBuilder

StyleBuilder - класс, позволяющий работать со стилем не как со строкой, а как с объектом, в котором хранятся параметры. Помимо этого в StyleBuilder хранятся цветовые и шрифтовые константы для удобства использования.

Внутри класса хранится изменяемый словарь, который хранит пары `название параметра GS -> (значение параметра, название параметра CSS)`. Для его корректной работы есть вспомогательный

неизменяемый словарь, хранящий пары `название параметра GS -> название параметра CSS`. Все манипуляции со словарём происходят через сеттеры и геттеры. Для создания результирующей строки тоже есть отдельный метод.

Изначально класс создан для стилей, которые используются в Global, но класс также поддерживает создание результирующей строки и для CSS.

10.12.1 Создание экземпляра StyleBuilder

Для использования StyleBuilder нужно сначала создать его экземпляр. Можно создать как и пустой стиль

```
val builder = StyleBuilder()
builder.build() //Результирующая строка пустая
```

так и взять за основу уже существующую строку стиля

```
val stringStyle = "color=$00FF4400;font=arial;fontsize=15".ns
val builder = StyleBuilder(stringStyle)
builder.build() //В результирующей строке будут все те же параметры с теми же значениями,
                //что и в stringStyle, но возможно параметры будут расположены в другом
↪порядке.
```

10.12.2 Работа с цветовыми параметрами

Для работы с цветовыми параметрами внутри StyleBuilder создан класс Color. Методы работы с цветовыми параметрами принимают на вход и возвращают экземпляры этого класса. На вход в сеттер можно подавать обычную строку типа String, она будет неявно преобразована в экземпляр Color.

При создании от строки Color пытается преобразовать цвет в HEX, если цвет невозможно преобразовать - бросается исключение. Color умеет преобразовывать строки типа:

- `$00BBGGRR`
- `$BBGGRR`
- `#RRGGBB`
- Наименование GS константы

Из Color можно получить строку обратно в формате HEX или GS кода:

```
val color = Color.Red //В Color есть константы Gs цветов
//ИЛИ
val color = Color("$000000FF")
//ИЛИ
val color = Color("$0000FF")
//ИЛИ
val color = Color("#FF0000")
//ИЛИ
val color = Color("clRed")
//-------------------------------
color.getGsHex // Вернётся $000000FF
color.getHex // Вернётся #FF0000
```

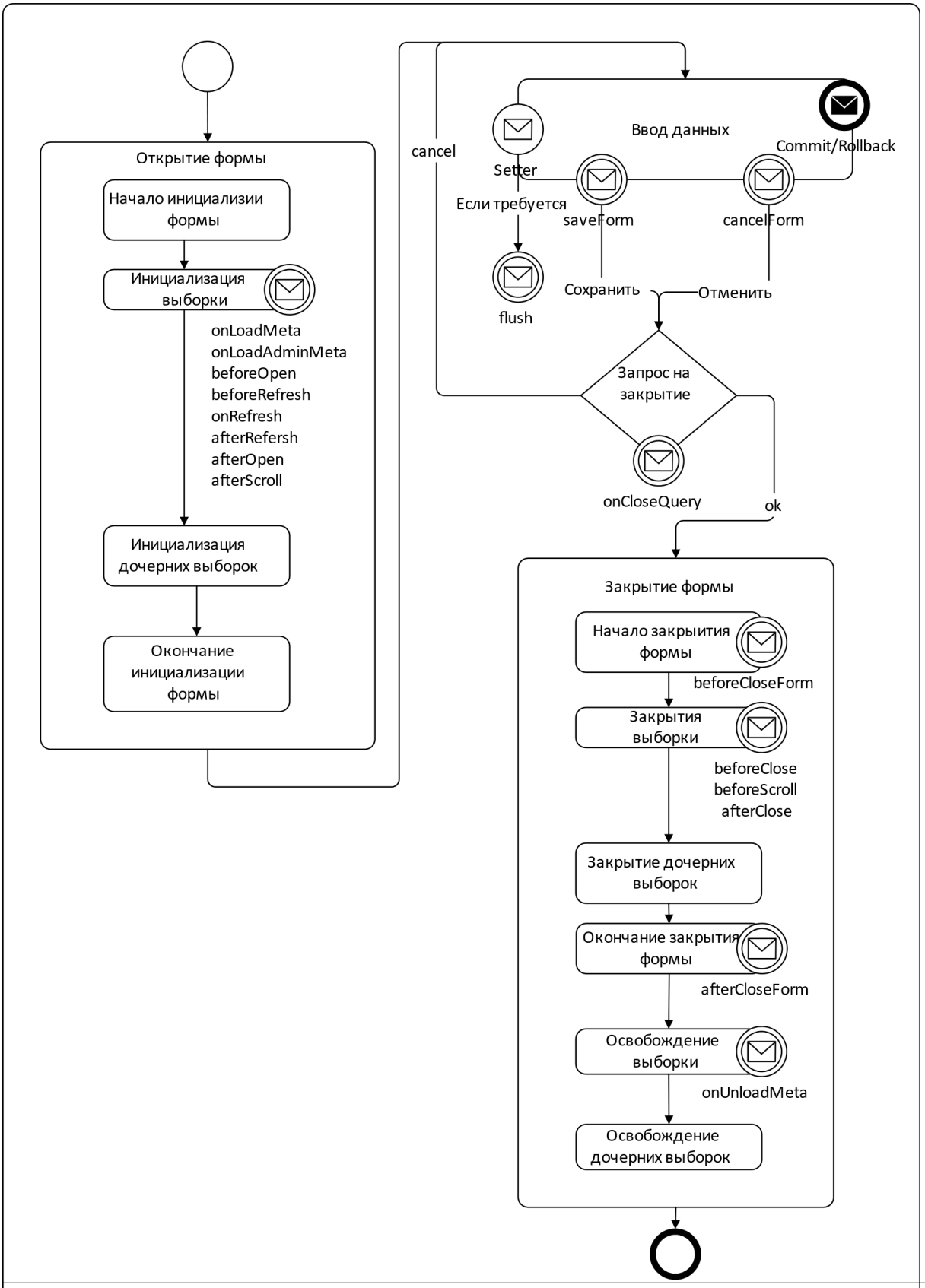
Если в Color подать пустую строку - это считается как отсутствие цвета.

10.12.3 Примеры

```
val style = StyleBuilder()
  .setBackgroundColor(Color.Blue)
  .setFontColor("#FFAAFF")
  .setFont(Font.Impact)
  .setFontUnderLine(1.nn)
  .build() // style = "color=$00FF0000;fontcolor=$00FFAAFF;fontunderline=1;
↪fontname=Impact"

val style2 = StyleBuilder(style)
  .setBackgroundColor(Color("")) // Убрать параметр цвета заднего фона
  .setFontColor("$00FFFFFF")
  .setFont("") // Убрать параметр шрифта
  .setFontUnderLine(NNumber()) // Убрать параметр нижнего подчёркивания
  .setFontBold(1.nn)
  .buildCSS // style2 ="color: #FFFFFF;
            //          font-weight: bold;"
```

10.13 Жизненный цикл формы



10.14 Иконки

Фреймворк поддерживает три варианта подключения иконок:

1. Из ресурсов сервера (основной режим)
   Коллекции изображений берутся из каталога с ресурсами `[G3_HOME]/resources/imagecollection`.
   Каждый подкаталог является коллекцией, вложенные файлы - элементы коллекции, номера которых соответствуют именам файлов.
2. Из базы данных
   Коллекции изображений загружаются из таблицы `btk_component`, поле `clobdataxml`
3. Подключение иконки по url
   Для иконки указывается явный url

10.14.1 Использование иконок из java-ресурсов модуля

Файлы изображений размещаются в ресурсных каталогах соответствующего модуля.

На скриншоте приведён пример структуры каталогов с ресурсами модуля BTK. Коллекции изображений размещены в каталоге ресурсов `../ru/bitec/app/btk/images`. Каждой коллекции соответствует одноимённый каталог.

Рассмотрим структуру каталога `ru/bitec/app/btk/images/toolbar`.

- В корне каталога размещаются файлы изображений с минимальным разрешением 16x16 точек.
- В подкаталоге `ru/bitec/app/btk/images/toolbar/24x24` размещаются файлы изображений с разрешением 24x24 точек. В случае необходимости могут быть добавлены каталоги изображений с большим разрешением.
- Подкаталог `ru/bitec/app/btk/images/toolbar/disabled` содержит обесцвеченные копии изображений коллекции toolbar, необходимые для отображения неактивных операций на панелях управления.

Пример указания изображения для операции с использованием аннотации:

```
@Oper(
  caption = "Род",
  headOperation = "references",
  imageUri ="ru/bitec/app/gs3/images/toolbar/61.png")
def open_Bs_Kind_RoList(): Unit = {
  Bs_KindAvi.roList().newForm().open()
}
```

При одновременном указании свойств `imageIndex` и `indexUri`, второе имеет приоритет.

10.15 Частично-загружаемые деревья

10.15.1 Введение

Частично-прогружаемое дерево – это такой вид отображения, где список в форме дерева подгружается из базы только для тех записей, которые раскрыты в дереве.

Частично прогружаемые деревья используются для отображений, в которых заведомо известно, что будет огромное количество записей, например `Mct_Structure`.

10.15.2 Настройка avm файла

Для отключения полной загрузки дерева в avm файле в настройках отображения необходимо указать свойство

```
fetchAllTree="false"
```

Помимо этого, в настройках фрейма необходимо указать наименование атрибута, отвечающего за отображение наличия у записи потомков

```
<tree idAttr="gid"
      idParentAttr="gidParent"
      hasChildrenAttr="bHasChild"
/>
```

10.15.3 Настройка refresh

Для отображения необходимых записей нужно получить список `idAttr` и `idParentAttr`, по которым будут загружаться данные из базы. Для этого нужно воспользоваться значениями параметров выборки `OPENNODEIDARRAY` и `OPENNODEID`

- `OPENNODEIDARRAY`
  Параметр выборки, хранящий в себе список открытых нод через запятую в формате `ftString`
- `OPENNODEID`
  Параметр выборки, передающий `idAttr` открываемой ноды в формате `ftString`

`OPENNODEIDARRAY` и `OPENNODEID` – системные параметры. Пользовательская установка значений этих параметров запрещена

10.15.4 Пример

Пример запроса для `Mct_StructureAvi#StructureTree`, находящегося в пакете `Mct_StructurePkg.getOnRefreshBs`

```
--n.1.- открытые узлы
with  nodes as (
  SELECT distinct tVal.tVal as gid
    FROM regexp_split_to_table(
          concat_ws(
             ', '
            ,:OpenNodeIdArray#
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            ,:OpenNodeIdArray
          )
          ,', '
        ) tVal
    where :OpenNodeId is null -- узлы нужны только при рефреше
  union
  select :gidParent as gid
  where :gidCurrentGid# is not null
),
-- фильтрация объектов для вывода в дерево
gidFlt as (
  select h.gid
        ,h.gidParent
    from (
      -- п.2.  блок рефреша узлов "RefreshNodes"
      select t.gid
            ,case when ll.id is null then null else l.gidParent end as gidParent
        from mct_structure t
        join mct_structurelink l on l.gidstructure=t.gid
        left join mct_structurelink ll on (
                    l.gidparent=ll.gidstructure and
                    l.idviewtype=ll.idviewtype
                  )
        where t.idPrjVer = :flt_idPrjVer
          and l.idviewtype=:flt_idViewType
          and l.gidparent in (select n.gid from nodes n)
      -- Блок OnRefresh
      union all
      --  п.3. 1. Раскрываем заказ - корневые записи
      select t.gid
            ,null as gidParent
        from mct_structure t
        join mct_structurelink l on l.gidstructure=t.gid
        l.idviewtype=ll.idviewtype
    )
      --left join mct_structure tt on (l.gidparent=tt.gid)
    where t.idPrjVer = :flt_idPrjVer
      and l.idviewtype=:flt_idViewType
      and not exists (
          select 1
            from mct_structurelink ll
           where ll.gidstructure=l.gidparent
             and ll.idviewtype=:flt_idViewType)
           and :OpenNodeId is null
      --
      union all
  --п.4. - 2. Раскрываем потомков - по нажатию ноды
  select l.gidstructure as gid
        ,l.gidparent
    from mct_structurelink l
   where l.idviewtype = :flt_idViewType
     and l.gidparent = :OpenNodeId
```

(continues on next page)

(продолжение с предыдущей страницы)

```
) h
--n.5.
SELECT
       t.id
      ,t.idClass
      ,t.gid
      ,tt.gidParent
      ,(select COALESCE(max(1),0) from mct_structurelink l where
      l.gidparent=t.gid) as bHasChild
      ,t.gidDoc
      ,t.gidDocVer
      ,t.gidSourceObj
      ,t.idPrjVer
      ,t1.sCode as idPrjVerHL
      ,t.idEskd
      ,t2.sCaption as idEskdHL
      ,t.idPosType
      ,t3.sCaption as idPosTypeHL
      ,t.sCode
      ,t.sCaption
      ,t.sSysName
  FROM Mct_Structure t
  join gidFlt tt on t.gid=tt.gid
  LEFT JOIN Bs_PrjVer t1 on t.idPrjVer = t1.id
  LEFT JOIN Mct_Eskd t2 on t.idEskd = t2.id
  LEFT JOIN Mct_PosType t3 on t.idPosType = t3.id
  LEFT JOIN Bs_Goods t4 on t.idGds = t4.id
  LEFT JOIN Msr_MeasureItem t5 on t.idMsr = t5.id
  LEFT JOIN Mct_OrderSheet t6 on t.idOrderSheet = t6.id
```

- *n.1*
  Получение таблицы значений открытых нод с учетом открытия ноды для новой созданной записи уровнем ниже
- *n.2*
  Получение записей, которые входят в открытые ноды
- *n.3*
  Получение корневых записей дерева
- *n.4*
  Получение записей, которые входят в открытую ноду – блок, срабатывающий при первом раскрытии ноды
- *n.5*
  Основной запрос для получения необходимых колонок.

Так же важно обратить внимание, что в основном запросе присутствуют изменения:

- Добавлен атрибут bHasChild
  Признак наличие дочерних элементов.
  Текст запроса атрибута:

```
(select COALESCE(max(1),0) from mct_structurelink l where
  l.gidparent=t.gid) as bHasChild
```

- Добавлено ограничение получения записей по отфильтрованному списку

```
join gidFlt tt on t.gid = tt.gid
```

10.15.5 Загрузка дочерних записей на открытие

Если необходимо открыть выборку с уже прогруженными дочерними записями, можно при открытии передать в качестве параметров список нод для прогрузки. И в качестве параметра, используемого в **refresh**-е использовать его. Однако, в таком случае все ноды будут закрыты и их нужно будет открывать вручную.

Примечание: Изменение параметров, участвующих в запросе, спровоцирует операцию **refresh**!

Для установки параметров без вызова **refresh** необходимо воспользоваться конструкцией

```
try {
  selection.ignoreParamChange = true
  setVar("gidParent", curGidParent)
  setVar("gidCurrentGid#", thisRop().gid)
}
finally {
  selection.ignoreParamChange = false
}
```

10.16 Пример разметки выборки

```
<?xml version="1.0"?>
<view xmlns="http://www.global-system.ru/xsd/global3-view-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.global-system.ru/xsd/global3-view-1.0"
      name="Bs_GdsCostDeviationType">

    <representation name="Default" doubleClickOperation.createFormMode="CardEdit"
                    doubleClickOperation.lookupMode="CloseFormOK"
                    caption="Виды отклонений в стоимости ТМЦ">
        <filter name="Bs_GdsCostDeviationTypeFilter">
            <macros name="DefFltReferenceMacro">
                <condition logicalOperator="and" id="shownotused"
                           isExpression="true"
                           expression="(:filter$flt_bShowNotUsed = 1
                                        or
                                       (coalesce(t.bnotactive,0) = 0
                                        and (t.dexpirydate is null or t.dexpirydate >␣
↪current_date)
                                        )
                                       )">
                    <filterAttr name="flt_bShowNotUsed" attribute-type="Long" caption=
↪"Отображать неиспользуемые"
                                isLastInLine="false" order="10" defaultValue="0"␣
↪editorType="check"/>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                </condition>
            </macros>
        </filter>
        <layout>
            <simpleComposer>
                <frame filter.isVisible="true" toolBar.isVisible="true">
                    <grid/>
                </frame>
            </simpleComposer>
        </layout>
        <attributes>
            <attr name="id" caption="Идентификатор" isVisible="false" editorType="edit"␣
↪order="-1"/>
            <attr name="idClass" caption="idClass" isVisible="false" editorType="edit"␣
↪order="-2"/>
            <attr name="gid" isVisible="false" editorType="edit"/>
            <attr name="sCode" caption="Код" editorType="edit" order="10" isRequired=
↪"true"/>
            <attr name="sCaption" caption="Наименование" editorType="edit" order="20"␣
↪isRequired="true"/>
            <attr name="sDescription" caption="Описание" editorType="memo" order="30"/>
            <attr name="bNotActive" caption="Не используется" editorType="check"/>
            <attr name="dExpiryDate" caption="Дата окончания использования" editorType=
↪"datePick"/>
        </attributes>

        <operations>

        </operations>

    </representation>

    <representation name="List" editMode="notEdit">
    </representation>

    <representation name="Card" editMode="edit" stdFilter.isAvailable="false">
        <layout>
            <simpleComposer>
                <frame filter.isVisible="false">
                    <card/>
                </frame>
            </simpleComposer>
        </layout>
    </representation>

    <representation name="Lookup" editMode="notEdit">
    </representation>

</view>
```

Глава 11

Сервис группового редактирования

11.1 Основные положения

Операция группового редактирования объявлена в `EntityAvi` в отображение `Default`, что означает, что она доступна в каждой выборке.

Что бы включить операцию группового редактирования, необходимо в вашем отображении переопределить операцию, так как изначально она выключена:

```
@Oper(
    active = true,
    headOperation = "extraOperation"
)
override def groupEditAttr(): Unit = super.groupEditAttr()
```

Для включения возможности группового редактирования на атрибуте, необходимо в odm (Объектно-документном представлении) или avm (Прикладную модель представления) добавить флаг isGroupEditAvailable:

```
<!-- odm -->
<attr name="dDoc" attribute-type="Date" caption="Дата док-та" order="60" type="basic"␣
↪isVisible="true"
      editorType="datePick" isHeadLine="true" isCopyInCopyObject="false"␣
↪isGroupEditAvailable="true" />
```

```
<!-- avm -->
<attr name="dDoc" caption="Дата док-та" editorType="datePick" order="60"␣
↪isGroupEditAvailable="true" />
```

Для json-атрибутов, при регистрации так же можно включить групповое редактирование (изначально оно выключено):

```
Btk_AttributeApi().registerJsonAttr(
    idpClass = idClass,
    spSystemName = "dCreateDoc".ns,
    spCaption = "Дата создания док-та",
    spAttrType = AttributeTypes.Long.toString,
    spType = AttrTypes.Basic.toString,
    bpGroupEditAvailable = 1.nn
)
```

Если необходимо включить групповое редактирование не изменяя проектный код, необходимо зайти в приложение «Настройки системы» далее Сущности->Классы. В списке атрибутов в столбце «Групповое редактирование атрибутов» поставить галочку.

11.1.1 Редактируемые объекты

Редактируемые объект представляет из себя ссылку на провайдер строки определённого класса. Для классов в отображение происходит поиск атрибута с наименование `id`, для миксинов `gidRef`. Если ваше отображение не привязано к классу, то в нём происходит поиск по следующим атрибутам: `id`, `gid`, `gidRef`. Если ни один из вышеперечисленных атрибутов не найден, будет выброшено исключение.

11.1.2 Свой список изменяемых объектов

Если необходимо передать свой список изменяемых атрибутов, необходимо переопределить операцию ггруппового редактирования следующим образом:

```
@Oper(
    active = true,
    headOperation = "extraOperation"
)
override def groupEditAttr(): Unit = {
    val idList = (0 until selection.selectedRecordsCount()).map { idx =>
      // Вместо id -> idTask
      NLong.fromAny(selection.selectedValueByName("idTask", idx))
    }.toList

    Btk_GroupEditPkg().execByRep(
        baseRep = this,
        listObjects = idList
    )
}
```

11.2 Установка групп

Если на классе установлена возможность группировки объектов Руководство разработчика: Сервисные возможности для классов # Группировка, то в отображении группового редактирования появится закладка «Установка групп», на которой можно задать группы для выделеных записей.

11.3 Дополнительные закладки для группового редактирования

Для отображения группового редактирования есть возможность добавить новые закладки для дополнительной обработки объектов. Для этого необходимо переопределить операцию группового редактирования, в которой вызывается `Btk_GroupEditPkg().execByRep`, с параметром `addTabs`, который принимает последовательность объектов закладок для данного отображения.

Закладки представлены в виде кейс класса `GroupEdit_TabsRow`, который принимает в себя данные об отображении и функцию обработки объекта.

11.3.1 Функция обработки объекта

Данная функция принимает в себя один параметр в виде кейс класса `GroupEdit_TabsRowParam`, который содержит в себе Сессия выполнения группового редактирования и редактируемый объект, который может быть представлен в виде NGid или NLong.

Глава 12

Универсальный фильтр

12.1 Основные положения

Операции универсального фильтра объявлены в `EntityAvi`, что означает, что он доступен в каждой выборке.

Вся фильтрация построена на создании объектов scala-классов, которые хранятся в памяти, сохранение настроек осуществляется в таблицу базы данных в виде json.

Управляет универсальной фильтрацией объект, который хранится в переменной `fltManager`. Для каждого экземпляра отображения создается свой экземпляр менеджера фильтра.

По умолчанию в фильтре для фильтрации доступны атрибуты выборки, а так же класса, который определяется через Avi-функцию `thisApi()` .

Сохраненная настройка не обращает внимание на изменившиеся свойства атрибутов и прочее. Она использует те настройки атрибутов, которые были в момент добавления этого атрибута с панели. Что бы изменилось поведение атрибута в настройке, необходимо его удалить, затем снова добавить и сохранить настройку.

12.1.1 Физические атрибуты и выражения фильтрации

- Атрибуты в макросе представлены их «физическими атрибутами». По умолчанию физический атрибут равен системному имени атрибута, но существуют атрибуты где это не так.Примером может служить технический атрибут класса «IdObject#», имеющий физическое имя «ID» (или gidRef для Mixin-ов и пр.). В макросе такой атрибут всегда представлен как ID.
- У атрибутов фильтра есть понятия выражения фильтрации, в котором указывается произвольное sql-выражение, для определения значения этого атрибута. Например, такие выражения используются в объектных характеристиках, хранящихся в json

12.1.2 Активация фильтра в выборке

Для включения возможности фильтрации через универсальный фильтр в отображении необходимо активировать операцию uniFilter. Так же удостовериться в активности операций onApplyFilter, onFilterFinalize, onFilterInit

Внимание: Фильтр не работает в выборках с объектными запросами

12.1.3 HL-атрибуты

HL – атрибуты (далее IdAttrHL) считаются ссылочными атрибутами и в фильтрации не участвуют, фильтрация осуществляется по их id-атрибутам (далее IdAttr).

- Если в выборке присутствует IdAttrHL, но отсутствует IdAttr, то фильтрация по атрибуту IdAttrHL будет недоступна.
- Все расширенные настройки требуется указывать на атрибуте IdAttr
- Ссылочность определяется:
  - Через тег `<ref>`, например: `<ref class="Rpl_IntOutSession"/>`
  - через одноименный атрибут класса этой выборки

12.1.4 Связь выборки и класса

Для выборки класс определяется через Avi-функцию `thisApi()`

12.1.5 Связь атрибута выборки и атрибута класса

Если у выборки определен класс, то для атрибута выборки его сопоставленным атрибутом класса будет атрибут с таким же системным именем.

12.1.6 Тип редактора атрибута

Тип редактора атрибута в большинстве случаев такой же, как указан в выборке, исключения:

- Выпадающие списки открываются через `редактор с тремя точками`
- Даты фильтруются через интерфейс «Фильтрации по периоду»
- Ссылочные атрибуты через `редактор с тремя точками`
- Специфичные типы сравнения (множественность) изменяют редактор на `редактор с тремя точками` .

Для атрибутов класса тип редактора определяется:

- Если указан в odm, то этот тип редактора.
- Иначе определяется от типа атрибута и его типа данных.

12.1.7 Коллекции класса

Для выборок с классами и для классов в перечне доступных для фильтрации атрибутов могут быть позиции с красными иконками, которые позволяют осуществлять фильтрации по коллекциям. Коллекции доступные для фильтрации определяются через odm- файл в теге `<collections>`.

Так же в карточке класса на закладке «Коллекции универсального фильтра» есть возможность добавить дополнительные коллекции для фильтрации. Например, подключить какой-то класс, который не входит в бизнес-объект, но имеет ссылки на текущий.

На этой же закладке можно переопределить и стандартные коллекции класса:

- отключить фильтрацию по коллекции, сняв признак «Активность»
- переименовать коллекцию, изменив поле «Наименование»

12.2 Доступные элементы для фильтрации

12.2.1 Типы доступных элементов

- `Атрибут класса/выборки` – доступен в случае генерации доступных атрибутов по классу или выборке
- `Коллекции` - добавляются при генерации по классу
- `Объектные характеристики, хранящиеся в json` :
  - Фильтрация:
    - Осуществляется через выражение, которое получает значение атрибута из json
  - Заполнение на панели доступных атрибутов:
    - Все об. Характеристики класса или класса, с которым сопоставлена выборка.
- `Универсальные характеристики` :
  - Фильтрация:
    - Для единичных - осуществляется через выражение, которое получает значение атрибута из json
    - Для множественных - формирует выражение через exists или not exists, по аналогии с фильтрацией по коллекциям.
  - Заполнение на панели доступных атрибутов:
    - Все универсальные характеристики, объявленные на:
      - группах класса, если у класса включена `Группировка`
      - типах объекта класса
- `Постоянные/технические атрибуты класса`
  - Заполнение на панели доступных атрибутов:
    - Всегда, при раскрытии узла, ссылочного на класс
    - Всегда, если выборка связана с классом
  - Атрибуты
    - sHeadLine# (Заголовок)

* sMnemoCode# (Мнемокод)
* dModifyDate# (Дата изменения)
* dCreateDate# (Дата создания)
* sModifyUser# (Изменивший пользователь)
* sCreateUser# (Создатель)
* IdGroup# (Группа) – заполняется только для классов, имеющих группировку. Позволяет фильтроваться по группе
* idObject# (Объект) – специальный атрибут, позволяющий фильтроваться по идентификатору(или gidRef) объекта

12.2.2 Первичная генерация по выборке

В операции `lazyInitFilter` происходит заполнение доступных атрибутов по выборке. Ниже рассмотренные варианты.

Если у выборки указан класс, то формируются атрибуты по этому классу. Атрибутами выборки считаются все атрибуты, присутствующие в дата-сете. Типы данных таких атрибутов также определяются их типами данных в дата-сете. Атрибуты выборки будут сгенерированы только, если выборка активна (загружен дата-сет). Сформированные классу атрибуты, присутствующие в выборке, будут дополнены свойствами из выборки (тип редактора, ссылочность и тд)

12.2.3 Генерация по классу

Для класса формируются:

- Атрибуты класса (кроме idClass), с не отключенным свойством доступности в фильтре

```
<attr name="bError">
    <uniFilter isActive="false"/>
</attr>
```

- Все объектные характеристики, настроенные пользователями и хранящимися в jObjAttrs_dz
- Технические атрибуты (заголовок, дата создания, объект и тд)
- Коллекции, перечисленные в odm и в карточке класса на закладке «Коллекции универсального фильтра»
- Универсальные характеристики, объявленные:
  - на группах класса, если в классе включен сервис группировки
  - на `типах объекта` класса, если сервис группировки не включен для класса.

Примечание: Для управления формированием атрибутов класса или добавления новых используется специальная точка расширения.

Подробности в главе `Расширенная настройка - Класс`

12.2.4 Атрибуты переменной ссылочности или ссылочные на миксин.

Атрибуты переменной ссылочности фильтруются через их раскрытие в дереве доступных элементов. Такие атрибуты раскрываются на классы, доступные к фильтрации по этому атрибуту

- Атрибут ссылочный на миксин, фильтруется по всем классам, к которым подключен этот миксин
- Атрибут, ссылочный на один класс, сразу фильтруется по этому классу
- Атрибут, у которого настроен перечень ссылочных классов, фильтруется по всем классам, указанных в этом перечне. Пример настройки:

```
<attr name="gidRefDocument" attribute-type="Varchar" caption="Документ" order="20"␣
↪type="refAnyObject">
    <uniFilter>
        <refAnyObject>
            <ref name="Wf_Doc"/>
            <ref name="Cnt_Contract"/>
        </refAnyObject>
    </uniFilter>
</attr>
```

- Атрибут с ненастроенной ссылочностью фильтруется как значимый. Приоритет обработки ссылочности:

1. Ссылочный на класс в refClass в теге uniFilter
2. Ссылочный на перечень классов в теге refAnyObject в теге unitFilter
3. Ссылочный на класс в refClass в настройке самого атрибута.

12.3 Формирование макроса

Макрос формируется в операции onApplyFilter. По умолчанию в фильтре создается главная группа, имеющая признак bisMainGroup = 1 и системное имя MainGroup, по ней и формируются макросы.

Каждая фильтруема коллекция будет представлять Exists в макросе.

12.3.1 Режим onlyWhere

Стандартный режим работы фильтра, в результате выдается текст одного макроса, который будет представлять собой exists, примером может служить:

```
Exists (
    select 1
      from bs_goods t#2
     where t#2.sName = '1'
       and t#2.id = t.id
    )
```

12.3.2 Обработка сервером приложений сформированного макроса

На операции onApplyFilter сформированный макрос передается серверу, чтобы он наложил условие на запрос данных выборки:

```
selection.filters().server.alias = mainGroupAlias
selection.filters().server.condition = fltMacro.where
selection.filters().server.isEnabled = true
```

12.4 Передача условий фильтра через параметры выборки

Для формирования условия используется объект класса FltCondition. В качестве имен атрибутов и коллекций можно использовать:

- Системное имя – в таком случае атрибут или коллекция ищется на первом уровне дерева доступных элементов.
- Путь – позволяет добавлять условия со вложенных уровней доступных элементов. Например, по атрибуту класса, на который ссылается атрибут верхнего уровня. Путь можно получить через дебаггер (атрибут sPath), стоя на доступном к фильтрации элементе. Пример использования:

```
val fltCondition = FltCondition()

//условия группы "отбор"
fltCondition.withMainGroup{mainGroup =>
    //добавляем условие по атрибутам выборки
    mainGroup.addAttr("nNumber", 1.nn)
    //условие "Заполнен"
    mainGroup.addIsNotNullAttr("sCaption")
    //добавляем условие по ссылочному атрибуту с раскрытием его на подуровни
    mainGroup.addAttr("attr:idPurchaseDirection/attr:sCaption", "643")

    //добавляем условие по атрибуту класса выборки, который отсутствует в дата-сете
    mainGroup.cl.addAttr("sCreateUser_dz", "admin")

    //добавляем условие по атрибуту даты
    mainGroup.cl.addDateInterval("dCreateDate_dz", "10.02.2020".nd, "10.02.2021".nd)

    //условие "в списке"
    mainGroup.addAttr("idRef", Seq(1.nl, 2.nl, 3.nl))

    //условие "Не в списке"
    mainGroup.addAttr("idRef", Seq(1.nl, 2.nl, 3.nl), FltAttributeCompareKind.notInList)

    //добавляем группу "Или"
    mainGroup.addOrGroup{orGr =>
        orGr.addAttr("sCaption", "123")
        orGr.addAttr("sCaption", "321")
    }

    //добавляем условие по коллекции
    mainGroup.withCollection("Btk_SomeCollection") {col =>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        col.addAttr("sCaption","someValue")
    }
}

//добавляем условие по другой рутовой группе
fltCondition.withRootGroup("groupName"){group =>
  //условия по атрибутам и коллекциям
  //...
}

//передаем условие в выборку
Btk_ClassAvi.defList.newForm().params(Map("uniFilterCondition_dz" -> fltCondition)).
↪open()
```

Внимание: Если нужно добавить условия по атрибуту класса выборки, который отсутствует в дата-сете, и объединить их в отдельную группу «И»/группу «ИЛИ», необходимо добавить .cl непосредственно перед добавлением соответствующей группы, а не самого атрибута.

Корректный пример добавления группы:

```
fltCondition.withMainGroup { mainGroup =>
  mainGroup.cl.addOrGroup { orGr =>
    orGr.addAttr("attr:id/attr:jAttrDataDeb_sCaption", "123")
    orGr.addAttr("attr:id/attr:jAttrDataCr_sCaption", "123")
  }
}
```

Следующий код не будет работать (ни при компиляции, ни при работе Вы не получите сообщение об ошибке, но переданное подобным образом условие функционировать не будет):

```
fltCondition.withMainGroup { mainGroup =>
  mainGroup.addOrGroup { orGr =>
    orGr.cl.addAttr("attr:id/attr:jAttrDataDeb_sCaption", "123")
    orGr.cl.addAttr("attr:id/attr:jAttrDataCr_sCaption", "123")
  }
}
```

Примечание: Узнать путь до определенного атрибута универсального фильтра можно при помощи отладчика (Ctrl+Alt+Shift+w). Заметим, что при формировании в коде пути до ссылочного атрибута необходимо указывать строку пути полностью, например, attr:idPurchaseDirection/attr:sCaption. Для обычных атрибутов указывается сокращенный путь, например, строка mainGroup.addAttr(«nNumber», 1.nn) добавляет условие фильтрации по атрибуту attr:nNumber.

12.5 Описание scala-классов, используемых фильтром

Во всех отображениях присутствует переменная менеджер фильтров `fltManager`, хранящая объект, создающийся в операции `onFilterInit`.

Менеджер управляет фильтром, загружает/выгружает настройки, формирует макрос. Основные классы фильтра находятся в пакете `ru.bitec.app.btk.flt`

12.5.1 Avi-методы выборки

- `onFilterInit` – событие инициализации фильтра. Создается фильтр-менеджер `fltManager`
- `onApplyFilter` – первичное применение настройки по умолчанию при отрытии выборки. Формирование макроса, применение макроса.
- `onFilterFinalize` – уничтожение фильтра.
- `uniFilter` – операция фильтра, если ее активировать, то будет возможность фильтрации.
- `lazyInitFilter` – ленивая инициализация. Вызывается при обращении к фильтру. Формирует главную группу фильтра.

12.6 Расширенная настройка

12.6.1 Класс

Разметка класса (odm.xml)

Для атрибута доступен тег uniFilter:

- `isActive` - по умолчанию включено, определяет наличие этого атрибута в перечне доступных для фильтрации
- `refClass` – ссылочный класс, перекрывает настройку ссылочности самого атрибута(указанного в теге `attr`)
- тег `refAnyObject` – позволяет настроить перечень классов, на который ссылается атрибут переменной ссылочности

Пример настройки атрибута класса в `odm.xml`:

```
<attr name="bError" attribute-type="Number" caption="С ошибками" order="70" type="basic"␣
↪editorType="check" defaultValue="0">
    <uniFilter isActive="false" refClass = " Bs_Goods">
        <refAnyObject>
            <ref name="Bs_Goods"/>
        </refAnyObject>
    </uniFilter>
    <booleanColumn/>
</attr>
```

Управление атрибутами в бизнес-логике

Для того чтобы управлять атрибутами в бизнес-логике используется точка расширения `Btk_Ext.afterBuildFltEntityMeta`, которая позволяет изменять атрибуты, которые формируются по умолчанию, а так же добавлять свои собственные.

Для управления атрибутами используются методы пакета `ru.bitec.app.btk.flt.meta.Btk_FltMetaAttributePkg`.

Ниже приведен пример, в котором json-атрибуту добавляется признак, что он может быть раскрыт в дереве атрибутов, и формируются атрибуты, которые отобразятся при его раскрытии. Каждый такой атрибут будет формировать условие, которое будет получать значение из json-контейнера.

```
def afterBuildFltEntityMeta(fltMetaEntity: FltMetaEntity, fltManager: FltManager): Unit␣
↪= {
  //проверка, что переданный класс - класс, для которого нужно добавить логику
  if (fltMetaEntity.name == "Bs_Goods") {
    //определяем, что json-атрибут доступен по умолчанию.
    val typeSizeAttrOpt = fltMetaEntity.attrMap.get("jTypeSizeAttrs".toLowerCase)
    if (typeSizeAttrOpt.isDefined) {
      val typeSizeAttr = typeSizeAttrOpt.get
      //ставим признак, что он может быть раскрыт в дереве
      Btk_FltMetaAttributePkg().setCanExpand(typeSizeAttr, true)

      //формируем атрибуты-потомки
      for (rvx <- new OQuery(Gds_TypeSizeCharacteristicAta.Type) {
      }) {
        //создаем значимые атрибуты, которые хранятся в json-контейнере
        val fltAttr = Btk_FltMetaAttributePkg().buildBasicJObjectAttr(
          systemName = rvx.get(_.sCode),
          caption = rvx.get(_.sCaption),
          dbType = AttributeTypes.Number,
          isBoolean = false,
          jsonColumnName = "jTypeSizeAttrs"
        )

        //устанавливаем созданному атрибуту потомка.
        Btk_FltMetaAttributePkg().setParentTree(fltAttr, typeSizeAttr)

        //добавляем атрибут в общий перечень атрибутов.
        fltMetaEntity.attrMap(fltAttr.systemName.toLowerCase) = fltAttr
      }
    }
  }
}
```

12.6.2 Коллекции

Для классов реализована возможность настройки отображения коллекций в УФ. Настройка находится в карточке класса на закладке «Коллекции универсального фильтра». Позволяет подключать дополнительные коллекции, или переопределять существующие.

Если признак «Активность» снят, то такая коллекция не будет отображаться в фильтре. Коллекция метаданных считается переопределенной, если существует запись, у которой ссылочный класс, и атрибуты соединения равны коллекции из мета данных класса.

Для коллекций атрибут мастера – id, для v-коллекций – gid.

Для регистрации записей в настройку используется метод `ru.bitec.app.btk.class_.flt.Btk_ClassFltCollectionApi#register`

12.6.3 Выборка

В avm выборки для атрибута доступен тег `uniFilter`, имеющий свойства:

- `isActive` – если снят, то такой атрибут не будет отображаться в списке доступных для фильтрации. По умолчанию включено.
- `isSelectionAttr` – если снят, то такой атрибут будет фильтроваться как атрибут класса (через join с таблицей класса по id), если у выборки нет класса, то атрибут не доступен для фильтрации. По умолчанию включено.
- `refSelection` – позволяет явно указать имя выборки, которая будет открываться при выборе значения в фильтре. Имеет приоритет перед свойством `refSelection` тега `ref`
- `refRepresentation` – позволяет явно указать имя отображения, которое будет открываться при выборе значения в фильтре. Имеет приоритет перед свойством `refRepresentation` тега `ref`

12.6.4 Создание дополнительных групп фильтрации

Для фильтрации сложных запросов, или подключения фильтрации не связанной с фильтруемым классом через коллекции или прямую ссылочность, есть возможность создания дополнительных корневых групп фильтрации.

Для этого требуется:

1. инициализировать новую группу;
2. сформировать макрос фильтрации по этой группе;
3. полученный текст подставить в основной запрос onRefresh

Ниже приведен пример создания дополнительной группы, которая позволяет фильтровать по условиям, наложенным на класс `ТМЦ`

1. На операции `lazyInitFilter` инициализируем дополнительную группу:

```
override def lazyInitFilter(): Unit = {
  if (!fltManager.isPopulatedRootGroup) {
    Btk_FltPkg().createRootGroupByClass(fltManager, "Bs_Goods", "Bs_Goods")
  }
  super.lazyInitFilter()
}
```

2. На `onApplyFiler` получаем текст фильтрации по группе:

```
override def onApplyFilter(): Unit = {
  super.onApplyFilter()
  //проверяем, что группы инициализированы и фильтр активен в выборке
  if (fltManager.isPopulatedRootGroup && fltManager.isActive) {
    val alias = "tt"
    val macros = Btk_FltPkg().generateMacroByGroup(fltManager, "Bs_Goods", alias)
    //если наложено условие по этой группе, то формируем текст макроса фильтрации,␣
↪который будет применим к запросу данных нашей выборки
    if (macros.hasFilter) {
      selection.setMacro(
         "GdsMacro",
         s"""
          exists (
            select 1
              from bs_goods $alias
             where $alias.id = t.idGds -- join к атрибуту из дата-сета (t.idGds)
               and ${macros.where} -- условия по этой групше
          )
         """
     )
    } else {
      //иначе делаем макрос без наложения условий
      selection.setMacro("GdsMacro", "1=1")
    }
  }
}
```

3. В операции `onRefresh` используем установленный в `onApplyFiler` макрос:

   - Через метод `prepareSelectStatement`

```
override protected def onRefresh: Recs = {
  prepareSelectStatement("&GdsMacro")
}
```

   - В тексте sql-запроса

```
override protected def onRefresh: Recs = {
  """
    select .....
      from .....
     where ....
       and &GdsMacro
  """
}
```

12.6.5 Связь панели стандартного фильтра с универсальным фильтром

Сервисная возможность указать атрибутам панели стандартного фильтра режим, при котором их изменение будет отображаться в универсальном фильтре, а изменение универсального фильтра будет приводить к изменению значений на панели фильтра. Так же при сохранении настроек универсального фильтра будут сохранены и настройки атрибутов стандартного фильтра (только атрибутов, для которых указана связь.)

Основным принципом является то, что атрибуты стандартного фильтра задаются по определенным правилам, в сеттерах этих атрибутов используются специальные библиотечные методы.

Если атрибут на панели стандартного фильтра выведен без возможности смены типа сравнения, то в универсальном фильтре ему так же нельзя будет изменить тип сравнения.

Существует 3 режима связи атрибута стандартного фильтра и универсального:

- `Через панель доступных для фильтрации атрибутов`
  Режим, при котором атрибуты стандартного и универсального фильтра полностью интегрированы. Ограничение по таким атрибутам накладывается через макрос универсального фильтра
- `Произвольный атрибут`
  Режим, при котором ограничения по такому атрибуту накладываются программистом по средствам макросов, формируемых на каждый атрибут.
- `Режим "Только сохранение"`
  Режим, при котором значения стандартного фильтра только транслируется в универсальный, без возможности его изменения в универсальном фильтре. Ограничения по таким атрибутам накладываются по правилам стандартного фильтра

Добавление атрибута через панель доступных для фильтрации атрибутов УФ

Если вам требуется вывести на панель стандартного фильтра один из атрибутов, по которым позволяет фильтровать УФ, то можно воспользоваться способом создания атрибута по его пути.

Фильтрация по атрибутам коллекций также доступа в этом режиме.

Настройка

1. Откройте интерфейс универсального фильтра выборки, для которой хотите создать новый атрибут на панели фильтров
2. Откройте панель доступных атрибутов
3. Найдите доступный атрибут, условия по которому хотите накладывать через панель стандартного фильтра
4. Выполните контекстную операцию `Сформировать текст разметки стандартного фильтра`. Операция доступа супер-пользователям или пользователям, обладающих объектной привилегий `Формирование текста разметки связей стандартного фильтра` пакета `Btk_FltPkg`
5. В открывшемся мастере укажите:
   - `системное имя атрибута стандартного фильтра`
     По умолчанию flt_<Имя атрибута УФ>
   - `наименование`
     По умолчанию наименование атрибута УФ

- `порядковый номер`
  Порядковый номер для атрибута стандартного фильтра.
- `с условием сравнения`
  Если признак установлен, то будет сформированы атрибуты стандартного фильтра, позволяющие изменять тип сравнения.

6. Подтвердите выбор в мастере
7. Из открывшегося окна скопируйте в `avm.xml` и `Avi` сформированный текст

Создание ссылочного атрибута с типом редактора «Выпадающий список»

1. Выполните действия описанные в главе `Настройка`.
2. Смените тип данных атрибута на `Long`
3. Сделайте атрибут не видимым
4. Добавьте `hl`-атрибут, в котором настройте выпадающий список.

Описание xml-разметки

Для атрибута стандартного фильтра добавляется тег `uniFilter`, в котором указываются следующие `xml-атрибуты`:

- `attrPath`
  Путь атрибута универсального фильтра. Это уникальный путь атрибута, по которому можно однозначно определить атрибут в УФ. Если это атрибут коллекции, то перед его путем добавляется путь до коллекции.
- `attribute-type`
  Тип данных атрибута
- `conditionAttr`
  Имя атрибута стандартного фильтра с типом сравнения. Содержит имя атрибута, который будет отвечать за смену типа сравнения. Если не указан, то тип сравнения для этого атрибута нельзя будет изменить в универсальном фильтре.
- `conditionType`
  Тип сравнения по умолчанию. Если не указан, то будет использоваться стандартный алгоритм определения типа сравнения. В качестве значений используется системное имя объекта `ru.bitec.app.btk.flt.selected.attribute.FltAttributeCompareKind`
- `defaultValue`
  Значение по умолчанию
- `defaultValueType`
  Тип значения по умолчанию
- `ref.selection`
  Имя выборки, которая будет открываться для выбора значений ссылочных атрибутов
- `ref.representation`
  Имя отображения выборки, которое будет открываться для выбора значений ссылочных атрибутов

Добавление произвольного атрибута

Иногда требуется добавить в универсальный фильтр произвольный атрибут, которого нет среди доступных для фильтрации.

Настройка

1. Добавьте в `avm.xml` для нужного атрибута тег `uniFilter` и настройте нужные свойства
2. Добавьте в `Avi` сеттеры для атрибутов:
   - `атрибут значения`:

```
@Setter()
def setflt_someValue(event: SetterEvent): Unit = {
  Btk_FltLib().setStdFilterValue(event)
}
```

   - `атрибут типа сравнения`:

```
@Setter()
def setflt_someCondition(event: SetterEvent): Unit = {
  Btk_FltLib().setStdFilterConditionType(event)
}
```

3. Добавьте ограничение в запрос выборки
   Для добавления ограничения добавьте в операции `onRefresh` использование макросов,сформированных по каждому из атрибутов. Имя макроса указывается в свойстве `macros` тега `uniFilter`.

   Например:

```
override def onRefresh: Recs = {
  prepareSelectStatement(
    s"""&flt_stdMacroName -- условие по атрибуту, имя макроса которого равно flt_
↪stdMacroName
       """
  )
}
```

   Пример сложного условия с под-запросами. В этом примере, сначала проверяется есть ли наложенное ограничение по атрибуту, и если оно есть, то добавляется под-запрос

```
override def onRefresh: Recs = {
  prepareSelectStatement(
    s"""
      ${if (fltManager.hasStdAttrCondition("flt_someAttrName")) {
        """exists (
              select 1
                from Some_class tt
               where tt.id = t.idSomeAttr -- связь между основной коллекцией и␣
↪таблицей   подзапроса
                 and &flt_stdMacroName -- макрос с сформированным ограничением по␣
↪атрибуту
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            )"""
        } else {
          ""
        }}
    """
  )
}
```

Описание xml-разметки

Для атрибута стандартного фильтра добавляется тег `uniFilter`, в котором указываются следующие `xml-атрибуты`:

- `attribute-type`
  Тип данных атрибута
- `type`
  Тип ссылочности атрибута. Доступные значения:
  - `basic`
    Значимый
  - `boolean`
    Булево
  - `refObject`
    Ссылочный на объект
  - `refState`
    Ссылочный на состояние
  - `refGroup`
    Ссылочный на группу
- `expression`
  `sql-выражение` для получения значения атрибута. Например, `tt.idGs`. Будет использовано при формировании макроса атрибута
- `macros`
  Имя макроса, в который будет установлено сформированное `sql-выражение` по этому атрибуту Пример выражения: `tt.idGds = 42`. В качестве правого операнда используется выражение указанное в `expression`
- `valueMacros`
  Имя макроса, в который будет установленно `sql-выражение` этого атрибута, но содержащее только часть, относящуюся к значению. Например, `42`. Если в значении указан интервал, то будет использовано выражение вида `(10, 20)`, где первое значение - начало интервала, второе значение - конец. Если одна из границ не задана, то будет указано `null`
- `ref.class`
  Ссылочный класс. Используется для атрибутов:
  - `Ссылочный на объект`
    Содержит имя класса, объекты которого будут доступны для выбора
  - `Ссылочный на группу` или `Ссылочный на состояние`\ Содержит имя класса, которым будет ограничен выбор групп и состояний

- `conditionAttr`
  Имя атрибута стандартного фильтра с типом сравнения. Содержит имя атрибута, который будет отвечать за смену типа сравнения. Если не указан, то тип сравнения для этого атрибута нельзя будет изменить в универсальном фильтре.
- `conditionType`
  Тип сравнения по умолчанию. Если не указан, то будет использоваться стандартный алгоритм определения типа сравнения. В качестве значений используется системное имя объекта `ru.bitec.app.btk.flt.selected.attribute.FltAttributeCompareKind`
- `defaultValue`
  Значение по умолчанию
- `defaultValueType`
  Тип значения по умолчанию
- `ref.selection`
  Имя выборки, которая будет открываться для выбора значений ссылочных атрибутов
- `ref.representation`
  Имя отображения выборки, которое будет открываться для выбора значений ссылочных атрибутов

Режим «Только сохранение»

Режим, при котором значение атрибута стандартного фильтра транслируется в универсальный, без возможности изменения значений в интерфейсе универсального фильтра.

Такой подход позволяет сохранять значения стандартного фильтра в настройках универсального.

Настройка

1. Настройте атрибут стандартного фильтра используя *базовый подход*
2. Для атрибута добавьте тег `uniFilter`, установите свойство `saveOnly="true"`
3. Добавьте сеттер для фильтра

```
@Setter()
def setflt_someValue(event: SetterEvent): Unit = {
  Btk_FltLib().setStdFilterValue(event)
}
```

Принудительная отправка значения в универсальный фильтр

Иногда требуется заполнять атрибуты стандартного фильтра не вызывая их сеттер. Т.к. на сеттерах выполняется синхронизация значения стандартного фильтра и универсального, то такой подход требует ручного указания, что значения какого-либо атрибута требуется отправить в универсальный фильтр.

Для этого можно воспользоваться методом `Btk_FltLib.submitStdFilterValue`.

Пример использования:

```
@Setter()
def setflt_idClassHl(event: SetterEvent): Unit = {
  event.editButton match {
    case EditButton.lookup =>
      Btk_DialogLib().withLookupIdMulti(Btk_ClassAvi.listForChoose()) { ids =>
        val idss = ids.mkString(";")
        val hls = ids.map(id => Btk_ClassApi().getMnemoCode(id)).mkString(";")

        selection.clientSetSelfVar("flt_idClass", idss) // установка значения другого
↪фильтра без вызова сеттера
        Btk_FltLib().submitStdFilterValue("flt_idClass") // отправка значения
↪стандартного фильтра в универсальный
        selection.clientSetSelfVar("flt_idClassHl", hls)
      }
    case _ =>
  }
  Btk_FltLib().setStdFilterValue(event) // значение фильтра, для которого был вызван
↪сеттер, будет отправлено в универсальный в этом вызове
}
```

Управление возможностью редактирования значения атрибута в универсальном фильтре.

Если требуется запретить редактировать значение в универсальном фильтре, то можно воспользоваться методом `Btk_FltStdLinkLib.setUniFilterReadOnly`. Часто это используется в связке вместе с блокирование атрибута и на панели фильтрации.

```
val bool = true

selection.filters().std.attrs("flt_someAttrName").isReadOnly = bool
Btk_FltStdLinkLib().setUniFilterReadOnly("flt_someAttrName", bool)
```

Примечание: Не работает для атрибутов, работающих в режиме `saveOnly`. Для таких атрибутов значение всегда не редактируется.

Значение по умолчанию

Значение по умолчанию указывается в теге `uniFilter` в свойстве `defaultValue`. Кроме самого значения может быть указан и его тип.

Тип значения по умолчанию указывается в свойстве `defaultValueType` и может принимать значения:

- `Constant`
  Константа. Если тип значения по умолчанию не указан, то он считается константой.
- `Jexl`
  `Jexl`-скрипт.

Константное значение

В свойстве `defaultValue` указывается константное значение. Например:

- Для числовых атрибутов `123`
- Для строковых `abc`
- Для даты `20.12.2023 10:30:42` Если требуется указать множественное значение, то указывается массив значений, разделенных запятой
- Для числовых атрибутов `[123,321]`
- Для строковых `[abc,cde]` Если указывается интервальное значение, то значения указывается массив, содержащий 2 значения. Первое значение - начало интервала, второе значение - конец. Если для одной из границ интервала не задано значение, то оставляется пустая строка:
- `[20.12.2023 10:30:42,20.12.2023 10:30:42]`
- `[,20.12.2023 10:30:42]`

Jexl-скрипт

В свойстве `defaultValue` указывается `jexl-скрипт`, который будет выполнен в контексте выборки для вычисления значения по умолчанию.

Если требуется указать множественное значение, то `jexl-скрипт` должен вернуть массив значений.

Если требуется указать интервальное значение, то `jexl-скрипт` должен вернуть массив, содержащий 2 значения. Первое значение - начало интервала, второе значение - конец.

Передача значений фильтра через параметры выборки

Для передачи значений используется тот же механизм, что и для *передачи значений в универсальный фильтр*

В качестве имен атрибутов используются имена атрибутов стандартного фильтра, которые являются атрибутами значений.

Пример:

```
val fltCondition = FltCondition().withStdLinkGroup{std =>
  std.addAttr("flt_idClass", Btk_ClassApi().findByMnemoCode("Btk_Class"))
  std.addAttr("flt_bBool", 1.nn)
}

Some_Avi.defList().newForm().params(Map("uniFilterCondition_dz" -> fltCondition)).open()
```

Получение значения атрибута фильтра

Если в прикладной логике для какого-то либо атрибута стандартного фильтра требуется получить его значение, особенно это актуально для ссылочных атрибутов, необходимо воспользоваться `библиотечным` методом `ru.bitec.app.btk.flt.Btk_FltStdLinkLib#getAttrValue`, в который требуется передать имя атрибута `стандартного фильтра`

Пример использования:

```
val fltCond = Btk_FltStdLinkLib().getAttrValue("flt_idBisObj")
//проверяем, что условие активно.
if (fltCond.isActive) {
  //проверяем, что выбран множественный тип сравнения
  if (fltCond.compareKind.isMultiValue) {
    //конвертируем к массиву и обходим значения
    fltCond.value.asNLongArray.foreach{v =>
      println(v)
    }
  } else if (fltCond.compareKind.isInterval) {
    //сравнение "В интервале"
    val interval = fltCond.value.value.asInstanceOf[FltLongInterval]
    println(interval.from)
    println(interval.to)
  } else {
    //единичное значение
    println(fltCond.value.asNLong)
  }
}
```

12.7 Формирование макроса

12.7.1 Строковый атрибут

- `Содержит` - накладывает условие по регистронезависимому вхождению строки.

Пример макроса:

```
upper(t.sCaption) like upper('%'||'value'||'%')
```

- `Начинается с` - накладывает условие на регистронезависимое начало строки.

Пример макроса:

```
upper(t.sCaption) like upper('value'||'%')
```

- `Заканчивается на` - накладывает условие на регистронезависимое окончание строки.

Пример макроса:

```
upper(t.sCaption) like upper('%'||'value')
```

- `Равно` - накладывает условие на регистронезависимое совпадение строки.

Пример макроса:

```
upper(t.sCaption) = upper('value')
```

- `Не равно` - накладывает условие на регистронезависимое несовпадение строки.

Пример макроса:

```
upper(t.sCaption) <> upper('value')
```

- `В списке` - накладывает условие на регистронезависимое вхождение в список.

Пример макроса:

```
upper(t.sCaption) in (upper('value'))
```

- `Не в списке` - накладывает условие на регистронезависимое отсутствие в списке.

Пример макроса:

```
upper(t.sCaption) not in (upper('value'))
```

- `Заполнено` - накладывает условие на наличие значения.

Пример макроса:

```
t.sCaption is not null
```

- `Не заполнено` - накладывает условие на отсутствие значения.

Пример макроса:

```
t.sCaption is null
```

- `Не содержит` - накладывает условие на регистронезависимое отсутствие подстроки.

Пример макроса:

```
upper(t.sCaption) not like upper('%'||'value'||'%')
```

- `В интервале` - накладывает условие на вхождение в заданный интервал.

Пример макроса:

```
upper(t.sCaption) >= upper('value') and upper(t.sCaption) <= upper('value')
```

12.7.2 Числовой атрибут

- `Равно` - накладывает условие на точное совпадение числового значения.

Пример макроса:

```
t.nNumber = value
```

- `Не равно` - накладывает условие на несовпадение числового значения.

Пример макроса:

```
t.nNumber <> value
```

- `В списке` - накладывает условие на вхождение числового значения в список.

Пример макроса:

```
t.nNumber in (value)
```

- `Не в списке` - накладывает условие на отсутствие числового значения в списке.

Пример макроса:

```
t.nNumber not in (value)
```

- `Заполнено` - накладывает условие на наличие числового значения.

Пример макроса:

```
t.nNumber is not null
```

- `Не заполнено` - накладывает условие на отсутствие числового значения.

Пример макроса:

```
t.nNumber is null
```

- `В интервале` - накладывает условие на вхождение числового значения в заданный интервал.

Пример макроса:

```
t.nNumber >= value and t.nNumber <= value
```

- `Меньше` - накладывает условие на то, что числовое значение меньше заданного.

Пример макроса:

```
t.nNumber < 1
```

- `Меньше или равно` - накладывает условие на то, что числовое значение меньше или равно заданному.

Пример макроса:

```
t.nNumber <= 1
```

- `Больше` - накладывает условие на то, что числовое значение больше заданного.

Пример макроса:

```
t.nNumber > 1
```

- `Больше или равно` - накладывает условие на то, что числовое значение больше или равно заданному.

Пример макроса:

```
t.nNumber >= 1
```

Внимание: Для типа Long виды сравнения: Меньше, Меньше или равно, Больше, Больше или равно - не доступны!

12.7.3 Булевый атрибут

- `Равно` - накладывает условие на точное совпадение булевого значения.

Пример макроса:

```
and t.bBool = 1
```

- `Не равно` - накладывает условие на несовпадение булевого значения.

Пример макроса:

```
and t.bBool <> 1
```

- `Заполнено` - накладывает условие на наличие числового значения.

Пример макроса:

```
t.bBool is not null
```

- `Не заполнено` - накладывает условие на отсутствие булевого значения.

Пример макроса:

```
t.bBool is null
```

12.7.4 Дата

- `Равно` - накладывает условие на точное совпадение даты.

Пример макроса:

```
t.dDate = to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- `Не равно` - накладывает условие на несовпадение даты.

Пример макроса:

```
t.dDate <> to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- `Заполнено` - накладывает условие на наличие даты.

Пример макроса:

```
t.dDate is not null
```

- `Не заполнено` - накладывает условие на отсутствие даты.

Пример макроса:

```
t.dDate is null
```

- `В интервале` - накладывает условие на вхождение даты в заданный интервал.

Пример макроса:

```
t.dDate >= to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS') and t.dDate <= to_
↪timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- Меньше - накладывает условие на то, что дата меньше заданной.

Пример макроса:

```
t.dDate < to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- Меньше или равно - накладывает условие на то, что дата меньше или равна заданной.

Пример макроса:

```
t.dDate <= to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- Больше - накладывает условие на то, что дата больше заданной.

Пример макроса:

```
t.dDate > to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

- Больше или равно - накладывает условие на то, что дата больше или равна заданной.

Пример макроса:

```
t.dDate >= to_timestamp('31.12.2000 00:00:00','DD.MM.YYYY HH24:MI:SS')
```

12.7.5 Json атрибут

- Заполнено - накладывает условие на наличие json значения.

Пример макроса:

```
t.jJson is not null
```

- Не заполнено - накладывает условие на отсутствие json значения.

Пример макроса:

```
t.jJson is null
```

12.7.6 Blob атрибут

- Заполнено - накладывает условие на наличие числового значения.

Пример макроса:

```
t.lBlob is not null
```

- Не заполнено - накладывает условие на отсутствие булевого значения.

Пример макроса:

```
t.lBlob is null
```

12.7.7 Clob атрибут

- По аналогий, как с `Строковым атрибутом`

12.7.8 Ссылочный атрибут

- По аналогий, как с `Числовым атрибутом`

Глава 13

MDA-таблица

13.1 Основные положения

`MDA-таблица` (Multi Dimension Analysis) - средство позволяющее реализовать список, управляемый пользователем с возможностью анализа данных и их группировки.

Основной особенностью `MDA-таблиц` является работа с данным вне базы данных. При первом запросе данных, будет выполнен запрос к базе данных, далее если не изменялся порядок сортировки данных, то запросы к базе не выполняются.

Для проектирования `MDA-таблицы` разработчик определяет запрос данных, и описывает колонки этого запроса, часть колонок могут быть `измерениями`, другие колонки - `показатели`.

- `Измерения` - колонки, по которым данные группируются
- `Показатели` - колонки, к которым применяются функции агрегации

Пользователь имеет возможность управлять сортировкой данных, порядком вывода колонок, способом агрегации значений, выводить под-итоги. Сохранять эти настройки в виде приватных и публичных, назначать настройку по умолчанию.

Построение и отображение данных делится на две части:

- `Модель запроса` - описывает правила получения данных из `БД`, `измерения` и `показатели`
- `Представление данных` - описывает правила сортировки, агрегации и порядок вывода данных на экран.

13.2 Модель запроса

Описывает правила получения данных, измерения и показатели.

Класс ru.bitec.app.btk.sel.mda.model.MdaModel.

Для построения модели создан класс-builder ru.bitec.app.btk.sel.mda.model.MdaModelBuilder

13.2.1 Текст запроса

Текст запроса - строка, в которой содержится текст sql-запроса. Этот запрос будет получать исходные данные из базы данных.

Для использования связанных переменных используется синтаксис AnormSql. Пример связанной переменной:

```
select ...
    from ...
where t.id = {id}
```

Для указания значений связанных переменных используется метод on класса ru.bitec.app.btk.sel.mda.model.MdaModelBuilder

Полный пример использования запроса со связанными переменными:

```
val model = MdaModelBuilder()
    .setQuery( () =>
    s"""
        select ...
            from ...
        where t.id = {id}
        """
    )
    .on(() => Seq(
    "id" -> 100.nl
    ))
    .build()
```

13.2.2 Измерения

Определяют поля по которым доступна группировка и сортировка

Класс ru.bitec.app.btk.sel.mda.model.Dimension

Значимое измерение

Измерение, представляющее собой единичную колонку запроса

Пример создания:

```
val model = MdaModelBuilder()
    .addDimension(NumberColumn("nNum"), "Число")
    .addDimension(StringColumn("sString"), "Строка")
    .addDimension(DateColumn("dDate"), "Дата")
```

Первым параметром передается объект `колонка`, который описывает тип данных и имя колонки в запросе данных

Ссылочное измерение

Измерение, представляющее из себя 2 колонки запроса, одна из которых содержит `id` объекта, а вторая - `headline`.

Используется если в качестве измерения используется какой- либо объект с прямой ссылочностью.

В отображаемых данных колонка с `id` будет невидима.

Примечание: Важно для ссылочных полей создавать такие измерения, т.к. они позволят фильтроваться в универсальном фильтре по такому измерению как по ссылочному

Пример создания:

```
val model = MdaModelBuilder()
  .addRefDimension("idRefObject", "idRefObjectHl", "Ссылочный объект", "Some_RefClassName
↪")
```

Параметры в порядке следования:

- имя колонки с идентификатором
- имя колонки с отображаемым значением
- Отображаемое имя
- Имя класса, на который ссылается измерение

Измерение переменной ссылочности

Измерение, представляющее из себя 2 колонки запроса, одна из которых содержит `gid` объекта, а вторая - `headline`.

Используется если в качестве измерения используется какой- либо объект, на который ссылочность организована через `gid`

В отображаемых данных колонка с `gid` будет невидима.

Примечание: Важно для ссылочных полей создавать такие измерения, т.к. они позволят фильтроваться в универсальном фильтре по такому измерению как по ссылочному

Пример создания:

```
val model = MdaModelBuilder()
  .addAnyRefDimension("gidRefObject", "gidRefObjectHl", "Ссылочный объект", List("Some_
↪RefClassName1", "Some_RefClassName2"))
```

Параметры в порядке следования:

- имя колонки с идентификатором
- имя колонки с отображаемым значением
- Отображаемое имя
- Список классов, на который ссылается измерение

13.2.3 Показатели

Определяют поля по которым доступна агрегация значений

Класс `ru.bitec.app.btk.sel.mda.model.Dimension`

Простой показатель

Показатель, представляющий собой единичную колонку запроса

Пример создания:

```
val model = MdaModelBuilder()
    .addMeasure(NumberColumn("nSum"), "Сумма")
```

Первым параметром передается объект `колонка`, который описывает тип данных и имя колонки в запросе данных

Составной показатель

Так же как и `простой показатель` представляет собой единичную колонку запроса, но имеет дополнительное `измерение`, в разрезе которого должна производиться агрегация. Например, сумма в договоре, где дополнительным измерением является валюта.

Пример создания:

```
val model = MdaModelBuilder()
  .addCompositeMeasure(
    NumberColumn("nSum"),
    Dimension(StringColumn("sMsr"), "Валюта"),
    "Сумма"
  )
```

Параметры в порядке следования:

- `колонка`, которая описывает тип данных и имя колонки в запросе данных
- `измерение` - дополнительное измерение, в рамках которого требуется агрегация
- отображаемое имя

13.2.4 Колонки

Модель обладает перечнем колонок, которые были использованы при создании измерений и значений.

Класс ru.bitec.app.btk.sel.mda.model.Column

Колонки определяют:

- тип данных
- имя колонки в запросе данных
- тип редактора

Типы колонок:

- NumberColumn
  Числовая колонка
- BooleanColumn
  Булева колонка. Наследник от числовой, с типом редактора чекбокс
- MoneyColumn
  Денежная колонка. Наследник от числовой, с типом редактора Денежный редактор
- LongColumn
  Целочисленная колонка
- StringColumn
  Строковая колонка
- DateColumn
  Колонка с типом дата, с типом редактора редактор даты
- DateTimeColumn
  Колонка с типом дата и время. Наследник от DateColumn, с типом редактора Редактор даты и времени

Стандартный тип редактора может быть переопределен, используя метод setEditorType, который принимает строку, имеющую формат настройки динамического типа редактора.

Пример:

```
NumberColumn("nNum").setEditorType("EditorType=etEdit")
```

13.3 Представление данных

Отвечает за настройки отображения данных в MDA-таблице:

- Порядок сортировки
- Вывод промежуточных итогов
- Агрегация значений показателей
- Порядок вывода колонок на экран
- Пользовательское переопределение наименования колонки
- Изменение стилей строк итогов/подытогов
- Размеры, видимость колонок

- Параметры настройки

Класс `ru.bitec.app.btk.sel.mda.view.MdaView`.

Представлением управляет пользователь, используя пользовательский интерфейс настройки вывода колонок, а так же сохраняя и загружая настройки.

13.3.1 Измерения

Определяет настройки сортировки, группировки и вывода промежуточных итогов.

13.3.2 Показатели

Определяет показатели, по которым будет выполнена агрегация

13.3.3 Атрибуты

Определяет порядок вывода колонок на экран и их наименование

13.3.4 Описание формируемых атрибутов выборки

На каждый подобранный `атрибут представления` в выборке формируются следующие атрибуты:

- `idValue[<имя колонки>]` - невидимый атрибут, будет содержать `id` для `ссылочных измерений` или `gid` для `измерений переменной ссылочности`.
- `value[<имя колонки>]` - видимый атрибут, в котором будет отображаться значение, видимое пользователя. Для `ссылочных измерений` или `измерений переменной ссылочности` в этот атрибут выводится значение колонки, объявленной как `headline`
- `sEditor[<имя колонки>]` - невидимый атрибут, управляющий типом редактора
- `sStyle[<имя колонки>]` - невидимый атрибут, управляющий стилем раскраски ячейки.

13.4 Подключение MDA-таблицы к прикладной выборке

13.4.1 Реализация отображения сгруппированных данных

Для подключения к прикладной выборке сервиса `MDA-таблиц` необходимо:

1. Унаследовать выборку от `ru.bitec.app.btk.sel.mda.Btk_MdaAbsAvi`
2. Создать отображение, если требуется
3. Унаследовать нужное отображение от отображения `Grid`
4. Определить модель запроса данных, переопределив метод `buildModel`, используя строителя модели `ru.bitec.app.btk.sel.mda.model.MdaModelBuilder`
5. Переопределить метод `getSourceDataRep`, который будет определять выборку для открытия исходных данных.

13.4.2 Реализация отображения исходных данных

Для реализации выборки, которая будет отображать `исходные данные` (данные которые вернул запрос из базы данных), необходимо:

1. Унаследовать выборку от `ru.bitec.app.btk.sel.mda.Btk_MdaAbsAvi`
2. Создать отображение, если требуется
3. Унаследовать нужное отображение от отображения `SourceDataGrid`

13.4.3 Управление типом редактора

Тип редактора для колонки определяется в момент описания `модели запроса данных`. Каждый тип колонки имеет свой тип редактора по умолчанию, для его переопределения можно воспользоваться методом `setEditorType`

13.4.4 Интеграция с универсальным фильтром

Универсальный фильтр анализирует `модель запроса данных` и на все `показатели` и `измерения` позволяет наложить условия фильтрации. Условие фильтрации будет наложено поверх результат запроса, описанного в `модели`.

Правила формирования доступных для фильтрации атрибутов:

- для `ссылочных измерений` создается ссылочный атрибут фильтрации
- для `измерений переменной ссылочности` создается атрибут фильтрации переменой ссылочности
- для всех остальных `измерений` и `показателей` создаются значимые атрибуты фильтрации в соответствие с типом данных

13.4.5 Создание стандартного фильтра

Стандартный фильтр создается по тем же правилам, что и для обычной выборки.

13.4.6 Использование макросов

Для использования макроса выборки в тексте запроса необходимо получить его текст, используя метод `selection.getMacro`, и добавить его в нужное место запроса.

Пример:

```
val model = MdaModelBuilder()
    .setQuery(() =>
    s"""
        select ...
            from ...
        where ${selection.getMacro("SomeMacrosName")}
        """
    )
```

13.5 Описание основных методов

- `extendDynMeta` - позволяет добавить в перечень атрибутов выборки свои атрибуты.
- `extendDynRow` - позволяет установить дополнительные значения в формируемую строку выборки
- `buildModel` - метод, который должен переопределить разработчик, для описания `модели данных`
- `mdaModel` - возвращает `модель данных`
- `mdaView` - возвращает `представление данных`
- `needGenChooseCheckBox` - признак, что требуется формировать чекбоксы для выбора строк
- `forSelectedRows` - позволяет обойти выбранные через чекбоксы строки.
- `getSourceDataRep` - метод, который должен переопределить разработчик, для указания выборки, которая будет открыта из операции `Открыть исходные данные`
- `filterRow` - позволяет отфильтровать записи
- `refreshBySql` - метод, который принудительно делает перезапрос данных из базы данных, и обновляет выборку

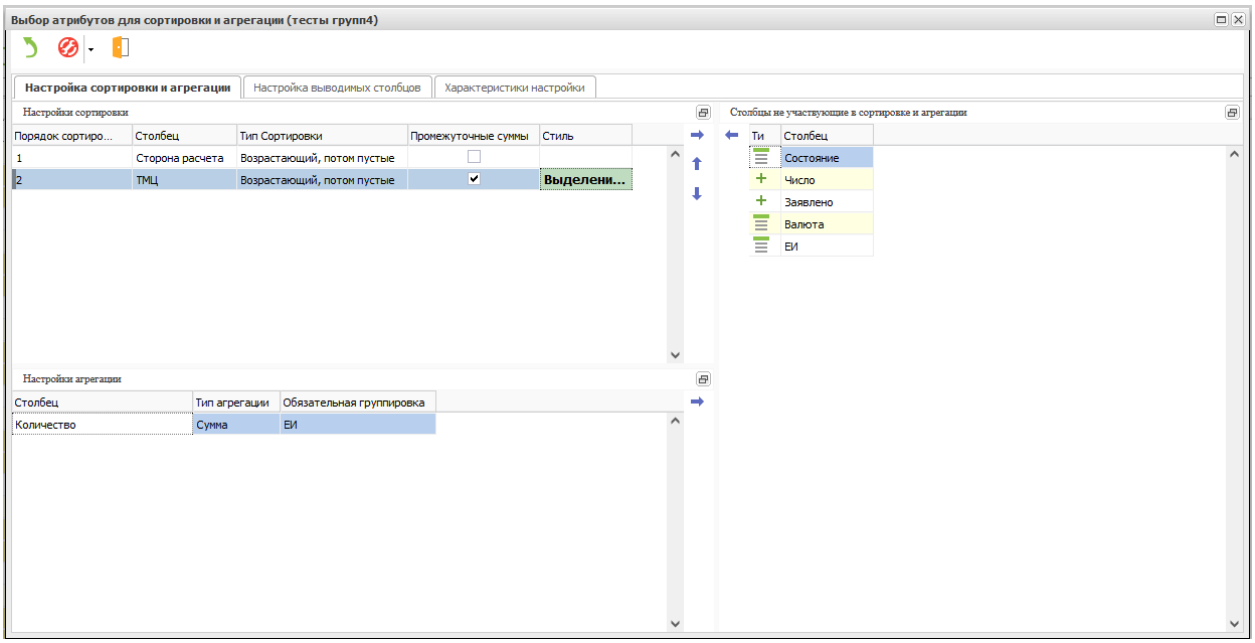
13.6 Описание основных операций

- `openSourceData` - `Открыть исходные данные`
  Открывает выборку, указанную в методе `getSourceDataRep`, в которой будут отображены исходные данные для выбранных строк
- `chooseAll` - `Выбрать все записи`
  Для всех записей проставляет чекбокс выбора
- `clearAllChosen` `Снять выбор со всех записей`
  Для всех записей снимет чекбокс выбора
- `chooseSelected` `Выбрать выделенные записи`
  Для всех выделенных записей проставляет чекбокс выбора
- `clearSelected` `Снять выбор с выделенных записей`
  Для всех выделенных записей снимет чекбокс выбора

13.7 Описание настройки атрибутов

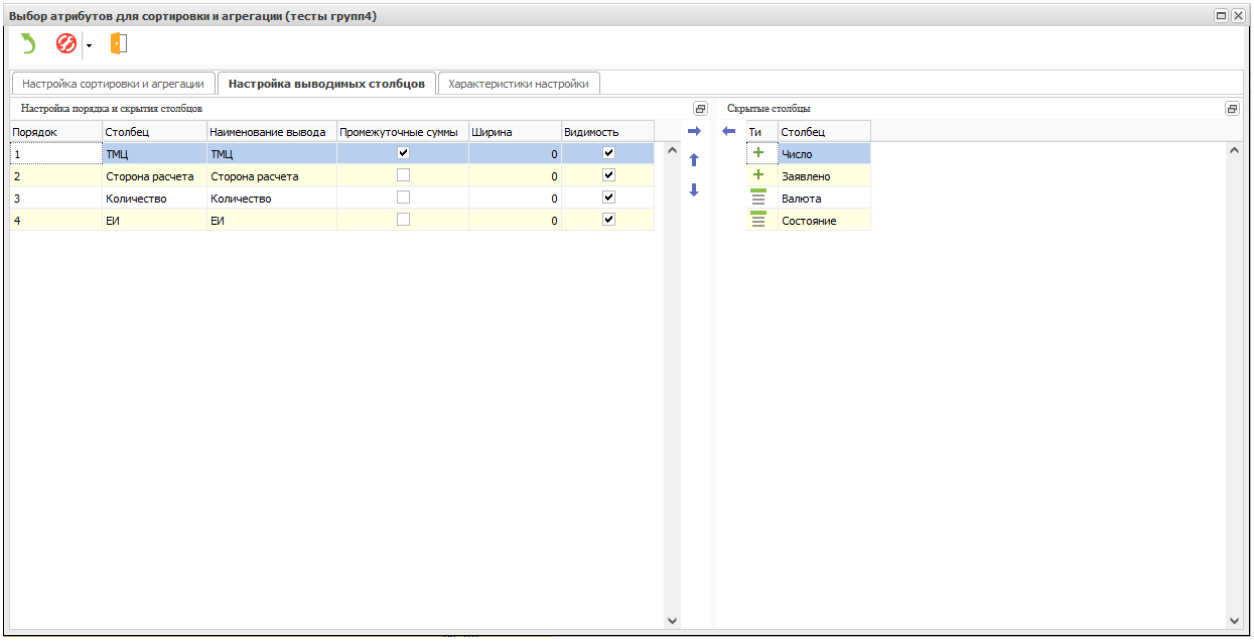
Настройка происходит через выборку `ru.bitec.app.btk.sel.mda.Btk_MdaViewEditorAvi#attrEditor`. Выборка ожидает в качестве параметра `mdaModel` в формате JSON и опционально прошлый `mdaView` для формирования нового `mdaView`. Возвращает JSON отредактированного `mdaView`.

Для работы с созданием/сохранением/загрузки настроек выборка ожидает названия выборки и репрезентации и опционально id прошлой настройки.



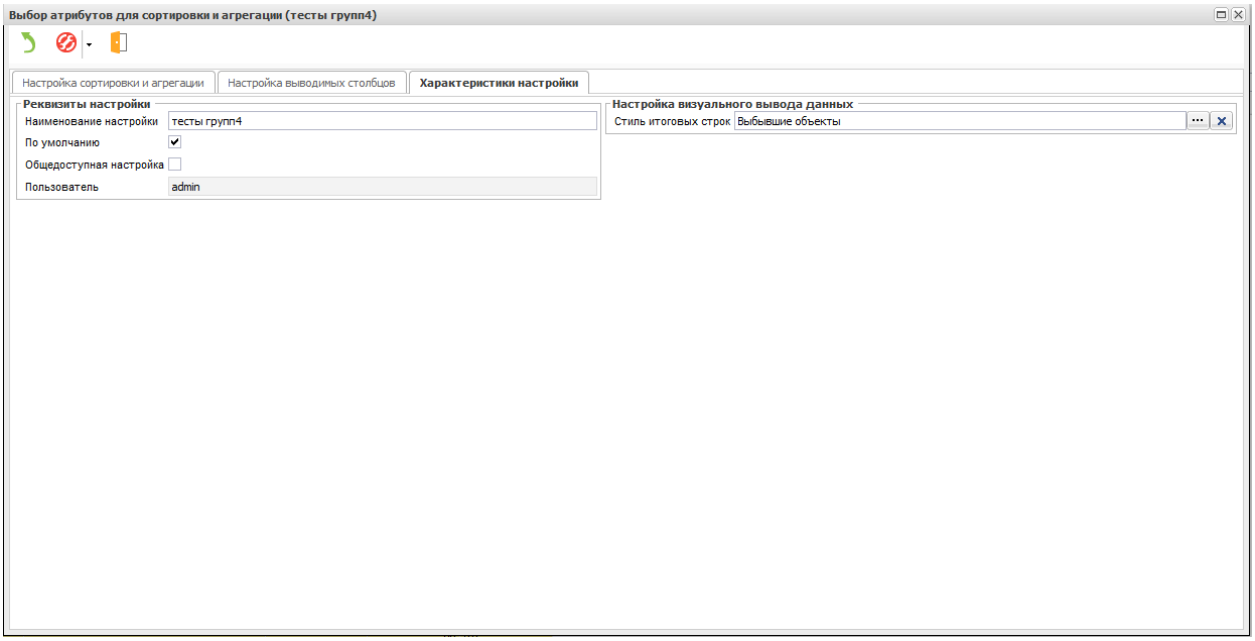
На вкладке `Настройка сортировки и агрегации` можно настроить

- Порядок и тип сортировки
- Вывод промежуточных итогов
- Тип агрегация значений показателей
- Стили подытогов



На вкладке `Настройка выводимых столбцов` можно настроить

- Порядок вывода колонок на экран
- Пользовательское переопределение наименования колонки
- Размеры, видимость колонок



На вкладке `Характеристики настройки` можно настроить

- Изменение параметров настройки
- Стили итогов

Глава 14

Аудит открытия форм и выполнения операций

Позволяет вести запись расширенного аудита действий пользователя при открытии форм и выполнении операций в выборке. Запись производится в автономной транзакции. Если выборка, на которой ведется аудит, будет открыта как детальный фрейм, то в аудит будут записаны все ее мастер-выборки.

14.1 Подключение настроек аудита

- В приложении `Настройка системы` перейти в `Обозреватель проектов`
- Для выбранной выборки перейдите на закладку `Настройка`

Для настройки аудита (класс Btk_SelSetting) можно выбрать следующие типы ведения аудита

- Не вести аудит (по умолчанию)
- Аудит только успешного открытия выборки/выполнения операции
- Аудит с сохраннением ошибок

14.2 Отчет аудита выборки

- Просмотр аудита от определенного объекта. В карточке объекта под операцией `Информация`, выбрать `Расширенный аудит по объекту`.
- Аудит по всем выборкам. В приложении `Настройка системы -Аудит - Аудит открытия форм и выполнения операций`.

При включенном чек-боксе `Только главные выборки` на панели фильтров в верхнем фрейме будут выведены только главные выборки, но при этом записи от детальных выборок можно всегда посмотреть на закладке `Дерево элементов формы`.

Часть IV

Инструменты

Глава 15

Конфигуратор

Системное приложение «Конфигуратор» предназначено для автоматизации и ускорению выполнения рутинных задач по разработке проекта:

- Создание нового проекта
- Создание нового модуля
- Работа с классами
  - Создание
  - Редактирование
- Работа с выборками
  - Создание
  - Редактирование
- Управление разметкой выборок
  - Редактор карточки
  - Редактор фреймов.
- Управление подсистемой распространения изменений.
- Управление технической документацией сущностей и методов проекта.

Конфигуратор предназначен для работы на локальном компьютере разработчика.

Конфигуратор взаимодействует с IDE Intellij Idea через специальный плагин.

Конфигуратор поддерживает систему контроля версий SVN.

15.1 Запуск

Для запуска конфигуратора выполните следующие действия:

1. Запустите сервер Global
   Выполните скрипт `C:\Global3se\start.bat` или запустите отладку в IDE Intellij Idea
2. Откройте в браузере адрес `http://localhost:8080/`
3. Введите логин, пароль, базу данных
4. Разверните настройки подключения
5. Установите флажок «Конфигуратор»
6. Войдите в систему

15.2 Первичная настройка

При первом запуске конфигуратор предлагает выбрать режим работы по умолчанию: локально или через систему контроля версий. При работе через систему контроля версий необходимо далее указать:

- Логин svn
- Пароль svn
- Url svn

Настройки сохраняются в поддиректорий конфигурации сервера Global: `C:\Global3se\application\config\cfg`

- `vcs.dat` – Настройка SVN
- `project.dat` – Настройка проектов

При удалении этих файлов мастер настройки автоматически запустится заново.

15.3 Создание нового проекта

Запустить мастер создания проектов: `Проекты > Новый проект`

15.3.1 Общие настройки

- *Системное имя проекта*
  Внутреннее имя проекта, обычно совпадает с наименованием директории проекта (пример: `intern199`)
- *Локальный путь до проекта*
  Корневая директория, в которой будет создан проект пример: `C:\SVN\ASSource\database\pgIntern\...`

15.3.2 Настройки для локального режима

В локальном режиме работы требуется указать

- *параметры соединения с БД*
- *шаблон проекта *
  - или путь к архиву с шаблоном проекта
  - или каталог с существующим проектом
  -

15.3.3 Настройки для работы через svn

Для режима работы через систему контроля версий предусмотрена расширенная конфигурация при которой надо задать параметры:

- *Путь в репозитории* Каталог в SVN, в который будет сохранен проект
  пример: `ASSource/database/pgIntern/...`

15.3.4 Настройки Sbt-plugin

Sbt-plugin является расширением к системе сборки sbt (scala build tool). При настройке рекомендуется не использовать пункт «trunk-версия», а оставить предложенную версию, которая является максимальной на момент создания проекта.

15.3.5 Настройки Базы данных

- *Url*
  Указывается тип драйвера и параметры подключения
  пример: `jdbc:postgresql://v39:5432/intern199`
- *Пользователь*
  Имя пользователя для подключения к БД
- *Пароль* Пароль для подключения к БД

Примечание: Параметры подключения к БД можно узнать у куратора проекта или системного администратора

15.3.6 Подключаемые модули

В списке подключаемых модулей, будут подобраны основные модули, для создания проекта. Их можно удалить или добавить новые. После выполнить операцию «Подобрать зависимости».

15.3.7 Завершение создание проекта.

1. По завершению конфигурации выполните операцию «Создать проект»
   При этом:
   - В случае работы с svn, в репозитории системы контроля версий будет создана структура каталогов проекта (каталог хранения проекта приложения, каталог хранения проектных модулей)
   - Отобразится диалог конфигурации сервера для запуска проекта
2. Завершите конфигурацию сервера
3. Импортируйте проект в Intellij Idea

15.4 Модули

Интерфейс для работы с модулями проекта расположен на панели инструментов главного меню.

Позволяет выполнять настройку модулей, подключать существующие модули, создавать новые.

15.5 Создание модуля

Операция позволяет создать новый модуль и подключить его к проекту.

1. На панели инструментов выполните операцию `Подключить модули > Создать новый модуль`
2. В открывшемся мастере введите:
   1. *Системное имя*
      Имя модуля в нижнем регистре
   2. *Наименование модуля*
   3. *Проектный модуль* (только для режима SVN)
      Если установить этот признак, то модуль будет создан внутри каталога проекта в подкаталоге module, и не будет доступен к подключению в других проекта. Если признак не установить, то модуль будет создан в основном каталоге модулей (`ASSource/module`) и будет доступен к подключению в других проектах.
3. Подключите модули, от которых будет зависеть создаваемый модуль

15.6 Обозреватель проекта

Отображает дерево сущностей проекта с возможностью их редактирования и просмотра документации. Сущности в обозревателе проекта:

- Классы
- Выборки
- Прикладная бизнес-логика (Api)
- Пакеты прикладной бизнес-логики (Pkg)

Для вызова используется операция панели инструментов главного меню «Обозреватель проекта»

15.6.1 Дерево сущностей

В дереве отображаются сущности, сгруппированные по каталогам. Для классов отображаются в качестве потомков их базовая выборка и Api

Дерево отображается в двух режимах, которые переключаются флагами на панели фильтров:

- *Логическая структура* \ Каталоги для сущностей определяются так: если в метаданных класса, выборки или пакета указан тег `logicFolder`, то каталогом этой сущности будет указанное значение, иначе физический каталог. Пример значения: `logicFolder = "audit.test"`
- *Физическая структура*
  Каталоги для сущностей определяются их расположением внутри проекта. Например для scala-пакета `ru.bitec.app.btk.audit.test` каталогом будет `audit.test`

Основная закладка для всех сущностей — это «Описание», которое формируется на основе документации, которая хранится в исходном коде. Подробнее см. главу «Документирование»

15.6.2 Создание новых сущностей

Реализовано создание основных сущностей:

- *Класс*
  Создает odm-файл
- *Выборка*
  Создает avi и avm файлы
- *Пакет*
  Создает Pkg и pkg.xml файлы
- *Описание каталога* Создает package-info.java в указанном каталоге.

15.6.3 Тегирование

Позволяет для сущностей повесить разнообразные теги. Которые будут отображены в дереве обозревателя.

Настройка доступных тегов осуществляется в гл. меню `Настройка > Теги обозревателя проекта`

Для установки тегов сущности используется операция `Дополнительно > Редактировать теги`

15.6.4 Документирование

Для сущностей на закладке «Описание» собирается их документация, хранящаяся в исходном коде.

Документация пишется в mark down.

Формирование описания по типам сущностей:

Класс

Документация для класса хранится в odm файле. В теге `documentation`, который доступен для:

- класса
- Атрибута

Выборка

Документация для выборки хранится в Avm файле. В тег `documentation`, который доступен для:

- Выборки
- Отображения
- Атрибута

Api

Документация хранится в scala doc в файле Api-класса

Pkg

Документация хранится в:

- теге `documentation` файла pkg.xml
- scala doc Pkg-класса
-

Каталог

Документация хранится в package-info.java, расположенном в каталоге на диске в scala-ветке. Документация в нем так же пишется в mark down.

Модуль

Документация хранится в `package-info.java`, расположенном корневом каталоге scala-ветки

15.6.5 Интеграция с IntelliJ IDEA

Интеграция осуществляется через плагин для IDEA

Позволяет открывать файлы в IDEA, которые соответствуют записям в дереве обозревателя. Для некоторых сущностей есть возможность открывать дополнительные файлы. Если не открыт текущий проект в IDEA, то будет выдано сообщение, что проект не найден.

При открытии проекта в IDEA запускается json-rpc сервер, который получает порт для прослушивания, и сохраняет в структуре проекта файл `.idea/ideasocket/<номер порта>.lock`, при этом блокируя файл. Конфигуратор при отправлении команды на открытие файла сканирует эту директорию на наличие заблокированных файлов, и получает порт, в который требуется отправлять данные.

Для открытия файла в IDE используется операция «Открыть в IDE»

Открываемые файлы по типам сущностей:

- *Класс*
  Открывает odm-файл
- *Выборка*
  Открывает Avi-файл.
  Дополнительная подоперация позволяет открыть avm
- *Api*
  Открывает Api-файл
- *Pkg*
  Открывает Pkg.scala
  Дополнительная подоперация позволяет открыть pkg.xml

15.7 Разработка конфигуратора

Код конфигуратора содержится в отдельном модуле. Для разработки этого модуля необходимо его настроить в режим разработки.

Конфигурация модуля происходит в файле `global3.config.xml` через тэги:

- `database`
  Конфигурация базы данных с именем «{DB_ALIAS}.CFG», где DB_ALIAS имя базы к которой идет подключение.
- `sbt` Настройка модуля содержащего исходный код конфигуратора.

15.7.1 Конфигурация базы данных

Если конфигурация отсутствует она создаются автоматически по алгоритму:

1. Скопировать конфигурацию «{DB_ALIAS}» в «{DB_ALIAS}.CFG»
2. Заменить имя sbt на значения `cfg`

```
<metaManager sbtName="cfg"/>
```

15.7.2 Конфигурация SBT

Если конфигурация sbt отсутствует, формируется конфигурация по умолчанию.

Шаблон конфигурации:

```
<sbt name="cfg"
     jarFolder="%GLOBAL3_HOME%\server\configurator\applib\\"
     binaryFolder="%GLOBAL3_HOME%\server\configurator\appbin\\"
     sourceMode="Jar"
     lazyLoad="true">
</sbt>
```

где:

- GLOBAL3_HOME
  Переменная окружения указывающая на расположения дистрибутива.
  Данная переменная задается автоматически при старте сервера

Конфигурация sbt для разработчика

Для разработки необходимо переопределить конфигурацию sbt по шаблону:

```
<sbt name="cfg"
     source="{PATH_TO_CFG}\configurator\application\"
     sourceMode="Dev"
     lazyLoad="true">
</sbt>
```

Алгоритм формирования для режима системной разработки

Для перехода в режим разработки:

- Задайте переменную окружения G3_ISDEV = 1
- Задайте переменную окружения G3_SHARE = «путь к jar архивам конфигуратора»

Шаблон генерации конфигурации в режиме системной разработки:

```
<sbt name="cfg"
    jarFolder="%G3_SHARE%\cfgapplib"
    binaryFolder="%G3_SHARE%\cfgappbin"
    sourceMode="Jar"
    lazyLoad="true">
</sbt>
```

Глава 16

Работа в IntelliJ IDEA

16.1 Подключение XSD-схем к редактору кода

Для проверки синтаксиса и работы подсказчика при редактировании xml-файлов (avm.xml, odm.xml, global3.config.xml), необходимо подключить к редактору xsd-схемы. Схемы хранятся в библиотеке `C:\programs\gsf-cli\workspace\dists\pgdev\Global3se\server\lib\engine\engine-1.0.jar`

xmlns	xsd -схема
http://www.global-system.ru/xsd/global3-module-1.0	C:\programs\gsf-cli\workspace\dists\pgdev\Global3se\server\lib\engine\engine-1.0.jar!\schemas\global3-module-1.0.xsd
http://www.global-system.ru/xsd/global3-view-1.0	C:\programs\gsf-cli\workspace\dists\pgdev\Global3se\server\lib\engine\engine-1.0.jar!\schemas\global3-view-1.0.xsd
http://www.global-system.ru/xsd/global3-view-template-1.0	C:\programs\gsf-cli\workspace\dists\pgdev\Global3se\server\lib\engine\engine-1.0.jar!\schemas\global3-view-template-1.0.xsd
http://www.global-system.ru/xsd/global3.config.1.0	C:\programs\gsf-cli\workspace\dists\pgdev\Global3se\server\lib\engine\engine-1.0.jar!\schemas\global3.config.1.0.xsd

Подключить схемы можно в настройках IDE IntelliJ Idea:
`File > Settings > Languages & Frameworks > Schemas and DTDs`

16.2 Создание сущностей без конфигуратора

Допускается создание сущностей фреймворка в ручном режиме прямо в IDE Intellij Idea

16.2.1 Создание Odm файла класса

Создание класса начинается с создания odm файла (объектно-документного представления сущности):

`application/{модуль}/src/main/resources/ru/bitec/app/{модуль}/{имя_сущности}.odm.xml`

Для создания можно воспользоваться файловым шаблоном из контекстного меню: `New > odm AllTypes`

В качестве примера, откройте файл: `../bs/src/main/resources/ru/bitec/app/bs/Bs_Contras.odm.xml`

16.2.2 Использование генератора кода

Для формирования базового кода для работы с сущностями предназначена утилита `ru.bitec.app.gtk.meta.SourceGenerator`. Генератор подключен к IDE Intellij Idea как внешняя утилита (External Tool). Доступ к генератору кода осуществляется через контекстное меню в обозревателе проекта или заголовка закладки: `External Tools > Generate sources`

Примечание: Для настройки утилиты см. главу «Начало работы».

Запуск утилиты возможен от любого файла, относящегося к сущности:

- `*.orm.xml`
- `*.odm.xml`
- `*.avm.xml`
- `*Avi.scala`
- `*Dvi.scala`
- `*Api.scala`
- `*Dpi.scala`

При работе утилиты создаются следующие файлы:

- В каталоге `../main/java/ru/bitec/app/{модуль}/`
  - `{имя_сущности}.java`
    Pojo-класс сущности
- В каталоге `../main/scala/ru/bitec/app/{модуль}/`
  - `{имя_сущности}Dpi.scala`
    Не изменяемая бизнес логика контроллера
  - `{имя_сущности}Api.scala` Изменяемая бизнес логика контроллера

    Внимание: Создается только если нет Api файла

  - `{имя_сущности}Dvi.scala`
    Не изменяемая бизнес логика представления (View). Шаблон
  - `{имя_сущности}Avi.scala`
    Изменяемая бизнес логика представления (View).

    Внимание: Создается только если нет Avi файла

- В каталоге `../main/resources/ru/bitec/app/{модуль}/`
  - `{имя_сущности}.orm.xml`
    Метаданные сущности для Orm
  - `{имя_сущности}.avm.xml` Xml-разметка представления

    Внимание: Создается только если нет Avm файла

При запуске формирования кода для сущности, у которой есть коллекции, так же будет пересозданы коды для коллекций.

Для массовой перегенерации файлов, можно вызвать перегенерацию для каталога, содержащего `odm.xml` – файлы. В этом случае будут пересозданы исходные коды для всех `odm.xml` в каталоге.

При вызове перегенерации от корневого каталога проекта, будут пересозданы исходные коды для всех сущностей.

16.2.3 Тонкая настройка Orm

Для тонкой настройки формируемого `orm.xml` файла допускается создавать файл-шаблон с именем `{имя_класса}.erm.xml` по формату, идентичному `{имя_класса}.orm.xml` файлу, и включать в него элементы и свойства, необходимые для тонкой настройки. При формировании `orm.xml`, будет прочитан erm.xml, и всё его содержимое будет добавлено в сформированный файл `orm.xml`.

16.2.4 Обновление схемы БД

Для корректной работы системы схема БД должна быть согласована с кодом.

В процессе разработки модуля, или при подключении нового модуля к проекту требуется обновлять схему БД специальной утилитой.

Так же, как и генератор кода, утилита обновления схемы БД может быть запущена от любой сущности системы. Таблицы создаются на основе метаданных, объявленных в файлах `*.odm.xml`. Создание таблиц с использованием EclipseLink отключено. Утилита генерации схемы запускается из контекстного меню: `External Tools > Generate Tables`

Примечание: При обновлении бинарного кода в продакшн-режиме согласование схемы запускается автоматически после обновления jar файлов проекта.

Примечание: Для настройки утилиты см. главу «Начало работы».

16.2.5 Создание главной выборки приложения

Для создания нового прикладного приложения, создайте в своём модуле Avi-класс унаследованный от `ProjectApplicationAvi`.

Если требуется создать приложение с глобальными фильтрами, то Avi-класс должен быть унаследован от `Bs_ApplicationAvi`. Для создания файла `avm.xml` используйте шаблон «avm Application» в меню создания файлов IntelliJ Idea.

Avi-класс, должен быть зарегистрирован в файле META-INF/applications.xml своего модуля.

Глава 17

Отладка приложений

17.1 Логирование на сторону клиента

Логирование позволяет отслеживать процесс выполнения запросов и подстановку параметров, последовательность вызова операций и открытия форм.

Для просмотра логов:

1. Зайдите в инструмент разработчика
   В Google Chrome и Mozilla Firefox открывается по `Ctrl+Shift+i`
2. Перейдите на закладку «Console»

Для изменения настроек логирования:

1. Нажмите на кнопку настроек
   Кнопка с изображением шестеренки находится в правом верхнем углу экрана.

   Примечание: Отображение кнопки может быть выключено в настройках сервера.

Настройки логирования позволяют выбрать тип информации, отображаемой в логе(по умолчанию включены все):

- Операции
- SQL
- Скрипт

Для логирования можно задать уровень (предполагается, что нижестоящие уровни выводят также данные по уровням, находящимся выше, за исключением OFF):

Наименование	Описание
TRACE	
DEBUG	
INFO	
WARN	
ERROR	
OFF	Отключение логирования

17.2 Клиентское окно отладчика

17.2.1 Дерево выборок

Окно отладчика вызывается при нажатии сочетания клавиш `Ctrl+Shift+Alt+d` или `Ctrl+Shift+Alt+w`.

В левой части расположена Иерархическая структура открытых выборок (включая выборки выпадающих списков). Со столбцами:

- Системное имя отображения
- Каноническое имя класса которому принадлежит выборка
  Для выборок без отображается имя без окончания «Avi»

В правой части отображаются закладки с данными по выбранной в левой части выборке:

- Атрибуты
- Параметры фильтра
- Параметры выборки
- Операции
- Отладочный макрос

17.2.2 Описание закладок

Атрибуты

На данной закладке отображаются данные по атрибутам из основного запроса выборки для того элемента, на котором находится фокус ввода. Данные не доступны для редактирования.

Атрибуты:

- *Системное имя*
  Системное имя атрибута
- *Значение*
  Значение атрибута

- *Только чтение*
  Признак того, что атрибут доступен только для чтения
- *Видимость*
  Флаг видимости атрибута
- *Порядковый №*
- *Ширина*
- *Тип данных*
- *Наименование*
  Наименование атрибута

Параметры фильтра

На данной закладке отображаются данные по параметрам фильтра, если фильтр доступен в выбранном отображении. Данные не доступны для редактирования.

Атрибуты:

- *Системное имя*
  Системное имя атрибута фильтра
- *Значение*
  Значение атрибута фильтра
- *Только чтение*
  Признак того, что атрибут фильтра доступен только для чтения
- *Видимость*
  Флаг видимости атрибута фильтра
- *Тип данных*
- *Наименование*
  Наименование атрибута фильтра

Параметры выборки

Закладка хранит список переменных, созданных в выборке, и параметров, переданных в выборку.

- *Системное имя*
  Системное имя параметра/переменной
- *Значение*
  Значение параметра/переменной
- *Тип данных*
  Тип данных параметра/переменной

Операции

На данной закладке отображаются операции выборки (отображения), которые определены непосредственно в ней, либо унаследованы.

- *Системное имя*
  Системное имя операции
- *Активность*
  Флаг активности операции
- *Видимость на ТБ*
  Флаг видимости операции на панели инструментов
- *Порядковый №*
  Порядковый № операции на панели инструментов
- *№ иконки*
  Номер картинки из коллекции
- *Наименование*
  Наименование операции

Отладочный макрос

На данной закладке можно выполнять следующие действия:

- `open`
  Открытие выборки, пример выражения:

```
ru.bitec.app.bs.Bs_PrjVer#List
```

- `openModal`
  Открытие выборки в модальном окне в режиме просмотра, пример выражения:

```
ru.bitec.app.bs.Bs_PrjVer#List
```

- `doLookUp`
  Открытие выборки в модальном окне в режиме выбора, пример выражения:

```
ru.bitec.app.bs.Bs_PrjVer#List
```

- `call`
  Вызов библиотечной операции, пример выражения:

```
ru.bitec.app.btk.Btk_GroupLib#editGroups
```

Для каждого действия можно задавать список параметров для передачи в выборку/библиотечный метод

`Ctrl+F9` - горячая клавиша для запуск макроса. Запуск осуществляется от выборки, на которой находится фокус ввода.

17.3 Отладка сервера в среде IDE

1. Запустите сервер приложения в режиме отладки \
   Для настройки запуска сервера приложений из среды, смотрите раздел «Настройку рабочего места» в данной документации
2. Установите точку останова в интересующем месте.
3. Откройте приложение в браузере.
4. Выберите необходимое приложение.
5. Выполните действие, которое вызовет отлаживаемый код.

17.3.1 Запуск jexl скрипта

Позволяет запускать на отладку автономную бизнес логику. Запуск jexl скрипта доступен в приложения «Настройка системы» в меню `Сервисы > Библиотека JEXL`

17.3.2 Определение причины SQL-вызова из EclipseLink

Для определения причины SQL-вызова из EclipseLink, поставьте точку остановки в методе

```
org.eclipse.persistence.queries.ObjectLevelReadQuery#executeInUnitOfWork:1219
```

Через стек вызова можно проследить, что является причиной вызова.

17.3.3 Определение списка загружаемых расширений (Xxx_YyExt)

Для определения списка загружаемых расширений, поставьте точку остановки в методе:

```
ru.bitec.app.gtk.eclipse.api.ExtensionPoint#onCreate : 26
```

17.3.4 Отслеживания изменения атрибута класса.

Для того чтоб поймать изменение любого атрибута объекта в коде поставьте точку остановки в методе (сеттер атрибутов)

```
ru.bitec.app.gtk.eclipse.rdb.EntityAro#_set : 280
```

По желанию можно добавить на точку условия.

Пример: `key == 635651L && propertyName.equals("bBaseMsrItem")`

- `635651L` - id объекта(Long)
- `bBaseMsrItem` - название атрибута

17.4 Мониторинг производительности

Для мониторинга производительности:

- jmc.exe
- jvisualvm.exe
- pg_stat_statements

17.4.1 Jmc

Входит в поставку JDK, имеет ограничения для коммерческого использования. Удобна тем что позволяет видеть снимки стека со строчками кода.

Позволяет искать узкие места в сервере приложения.

17.4.2 Jvisualvm

Входит в поставку JDK. Позволяет искать узкие места в сервере приложения.

Снятие снимка

1. Запустите jvisualvm
2. Откройте процесс для мониторинга \ Для этого в окне «Application», сделайте двойной клик по нужному приложению.
3. Начните запись снимка
   Для этого на закладке «Sampler» в открытом процессе, нажмите кнопку «CPU»
4. При необходимости, в открытом процессе, запустите задачу, которую вам нужно профилировать.
5. Остановите запись снимка
6. Перейдите к снимку для анализа
   Для этого на закладке «Sampler/CPU Sampler» нажмите кнопку «Snapshot»

Примечание: Профилирования идет по java функциям.

Получение полного списка функций для класса

Scala компилятор может генерировать служебные функции, которых не видно в исходном коде, однако знание их имени, может повысить локализацию узких мест.

Для получения соответствия анонимной функции и строчки кода в классе:

1. Запустите scala console
   1. Откройте Run/Debug configuration
      Пункт «Edit configuration» в списке с лева от кнопки «запустить» в меню
   2. Нажмите добавить конфигурацию
   3. Выберете «scala console»

4. Укажите модуль, для класса которого вы хотите посмотреть список служебных функций
5. Запустите консоль
6. Выполните команду `:javap -l ru.bitec.app.Модуль.Класс`

17.4.3 JProfiler

Jprofiler - это комплексный профилировщик Java. Интуитивный пользовательский интерфейс Jprofiler поможет устранить узкие места производительности, точно определить утечки памяти и понять проблемы многопоточности.

Дистрибутив: «ftp://ftp/pub/%23Distrib/IntelliJ%20IDEA/JProfiler/»

Установщик сам интегрируется с IntelliJ IDEA

17.4.4 pg_stat_statements

Позволяет вести статистику запросов в базе.

Подробнее смотрите документацию postgresql

17.5 Мониторинг оперативной памяти

Для мониторинга оперативной памяти в продакшене можно использовать

```
jcmd pid GC.heap_dump dumpFile
```

Для анализа файла, рекомендуется использовать:

- Eclipse Memory Analyzer

17.6 Отладка в закрытых средах

17.6.1 Отладка в Visual Studio Code

Редактор VSCode с установленными расширениями является более легковесным средством отладки java-приложений по сравнению с ItelliJ IDEA и более удобным по сравнению с JDB.

Расширение vscode-gtk-debug применяется, в случае необходимости отладить приложение на сервере заказчика и при невозможности предоставить доступ к исходным кодам на рабочем месте.

Внимание: Не рекомендуется применять jvm отладчик на боевой базе, если есть такая возможность желательно запустить копию экземпляра сервера и отлаживать уже его. Отладчик может оказать негативное влияние на производительность сервера приложения.

В случае необходимости отлаживаться на боевом сервере обязательно задавайте в фильтрах поток отладки, в противном случае, возможно срабатывание точек прерывания в пользовательских сеансах, что внесет путаницу в отладку и приведет к зависаниям на стороне пользователей.

Настройка рабочего места

1. Установите VSCode
2. Установите расширение (Extension) для отладки
   Дистрибутив расширения размещён в подкаталоге, «plugins\vscode
   vscode-gtk-debug-x.x.x.vsix»
3. Запустите VSCode
4. На левой панели управления нажмите кнопку Extensions
5. Нажмите кнопку с тремя точками на заголовке открывшейся панели
   В открывшемся меню выберите «Install from VSIX»
6. Найдите и выберите файл дистрибутива расширения `vscode-gtk-debug-x.x.x.vsix`
7. В списке установленных расширений появится «Global for Postgres Debugger»

Настройка рабочего каталога

Для начала работы с VSCode необходимо открыть рабочий каталог.

1. Создайте пустой каталог в произвольном месте.
2. Перейдите в VSCode
3. Откройте созданный каталог
   Выберете пункт меню `File > Open Folder...` и укажите созданный каталог.
4. Переключитесь на закладку отладки.
   Для этого нажмите кнопку «Run and Debug» в левой панели
5. Создайте файл `launch.json`
   1. Кликните на ссылку «create a launch.json file».
   2. В открывшемся выпадающем списке выберите тип «Gtk»
      В результате, в каталоге будет создан `.\vscode\launch.json`, содержащий конфигурацию подключения к отлаживаемому серверу Global, по умолчанию.
6. Измените значения параметров в соответствии с настройками вашего сервера приложений.

Параметры конфигурации для отладки:

- `hostname`
  Имя или IP отлаживаемого сервера
- `port`
  Порт, открытый отлаживаемым сервером для подключения java-отладчика. Значение задаётся параметром при старте java-процесса сервера Global.

```
java -Xdebug -Xrunjdwp:server=y,transport=dt_socket,address=4000,suspend=n
```

- `threadPattern`
  RegExp-выражение фильтрующее имена потоков, доступных для срабатывания точек останова. По умолчанию, доступны все потоки пользовательских сессий.

> **Внимание:** Из конфигурации подставляются экранированные слэши. Выражение должно выглядеть: `ESession_\\d*`

- `httpPort`
  Порт на котором запущен сервер Global. Обычно: 80 или 8080.
- `httpUser`, `httpPassword`
  Имя пользователя и пароль, которые будут использоваться для авторизации Rest-запросов к серверу Global, при получении декомпилированных текстов прикладных Java и Scala-классов.

Подключение отладчика к процессу сервера

Для подключения java-отладчика, процесс сервера должен быть запущен с соответствующими параметрами:

```
java -Xdebug -Xrunjdwp:server=y,transport=dt_socket,address=4000,suspend=n
```

1. Переключитесь в режим отладки нажатием кнопки «Run and Debug»
2. Запустите отладку

В случае успешного подключения к отлаживаемому процессу появится панель с командами отладчика а так же строка состояния изменит цвет. Смотрите Документацию по отладке VSCode

Получение текста прикладного класса

Для установки точки остановки «Breakpoint», необходимо открыть в редакторе декомпилированный текст отлаживаемого класса. Получение текста класса не зависит от подключения отладчика к процессу сервера. Главное, что бы в конфигурации отладки были корректно указаны параметры подключения.

1. Нажмите комбинацию клавиш `Shift+Ctrl+P`.
2. В открывшемся поле ввода наберите `Find Classes`
3. Выберите найденную команду «Find Classes. . . »
4. В открывшемся поле ввода введите имя искомого класса и нажмите Enter
   Буде выполнен Rest-запрос к серверу приложений, для получения списка имён классов, содержащих введённое имя
5. Выберите искомое имя класса из отображённого списка
   Если был найдено только одно имя, будет открыт декомпилированный текст класса без промежуточного выбора.

Декомпилированный текст класса содержит расширение `*.gdc` (Global Decompiled Class). Являются виртуальным и не сохраняются на диске. Файл состоит из набора строк на которые можно ставить точки сохранения. Каждая строка содержит имена методов, вызовы которых происходят на этой строке исходного кода.

Задание точки остановки

Для установки точки, щёлкните мышкой слева от номера требуемой строчки.

В списке «BREAKPOINS» появится информация об установленной точке. Точки могут устанавливаться до или после подключения отладчика к процессу сервера.

Срабатывание точки остановки

При срабатывании точки остановки в одном из потоков, имя которого соответствует паттерну, произойдёт его остановка. В соответствующих фреймах будут отображены стек вызова и значения локальных переменных. Кликами по стеку, можно перемещаться между файлами.

Перемещение между классами

Если установить курсор на имя класса или вызываемого метода и нажать `F12` или `Ctrl + Click` будет выполнена попытка перехода к файлу, где объявлен класс.

Если не удастся однозначно определить класс по имени, будет открыт диалог выбора.

Замена декомпилированного текста на исходный код

Скопируйте исходный код класса в буфер обмена и выполните вставку в соответствующий gdc-файл через контекстное меню

После вставки исходного кода возможно переключаться между исходным кодом и декомпилированным текстом кнопкой расположенной в заголовке файла.

Так же в заголовке файла находятся кнопки которые переключают режимы подсветки синтаксиса для соответствующего языка.

17.6.2 Отладка приложений в консоли jdb

Используется если ошибка не повторяется в окружении с настроенной IDE и невозможна удаленная отладка.

> **Внимание:** Штатный консольный отладчик не позволяет фильтровать поток для точки останова, так что полноценно работать на production сервере, с таким отладчиком нельзя

Внимания примеры приведены для операционной системы windows для Linux смотрите официальную документацию по jdb.

1. Убедитесь что процесс запущен с опциями позволяющими отладку:

```
-agentlib:jdwp=transport=dt_shmem,address=jdbconn,server=y,suspend=n
```

2. Запустите jdb

```
${jdk_home}\bin\jdb -attach jdbconn
```

3. Наберите help, для просмотра списка допустимых команд

Смотрите так документацию: `ru.bitec.app.gtk.debug.Jdb`

Список самых полезных команд

- `stop`
  Установка точки останова
- `where`
  Просмотр стэка потока
- `locals`
  Печатает переменые в текущем фрейме
- `dump this`
  Печатает текущий класс
- `next`
  Выполняет строчку кода
- `cont`
  Продолжает выполнение

Глава 18

Тестирование

18.1 Unit-тестирование

При необходимости проверки работоспособности отдельных частей исходного кода имеется возможность реализации unit-тестирования на основе ScalaTest.

Для создания набора тестов необходимо создать класс, который будет наследоваться от LangFunSuite, ApiTest или других классов, в которые подмешаны трейты из ScalaTest.

Совет: Для дополнительной информации смотрите библиотеку unit тестирования: scalatest

18.1.1 Создание класса с тестами

1. Перейдите в окно проекта
2. Выберете целевой модуль
3. Перейдите в папку с исходными кодами
   `[module_name]/src/test/scala`

   Совет: Создать недостающую папку можно из контекстного меню в idea `New > Directory`

4. Создайте пакет `ru.bitec.app.[module_name]`
5. Создайте тестовый класс

```
class Lesson1Test extends LangFunSuite{
  test("HelloWorld"){
      println("hello world")
  }
}
```

Совет: Запустить тест можно из контекстного меню, для этого:

1. Переведите курсор на декларацию функции или класса, если нужно запустить все объявленные тесты класса
2. В контекстном меню выполните операцию „Debug“ для запуска в режиме отладки или „Run“ для простого запуска

Подробнее смотрите выполнение тестов в руководстве idea.

Примечание: Существуют два специализированных базовых класса для тестовых случаев:

- `LangFunSuite`
  Используется для тестов которые не нуждаются в контроллерах бизнес логики.

  Данные тесты не могут использовать `Api`, `Pkg`, и не имеют подключения к базе данных по умолчанию.
- `ApiTest`
  В данном тексте доступен контекст автономной бизнес логики, тестовые случаи могут использовать `Api`,`Pkg` объекты. При этом запуск теста становится медленней из за необходимости инициализировать контекст.

18.1.2 Базовые Assert методы

Трейт Assertions содержит основные методы для проверки предположений.

1. assert(condition: Boolean)

Этот метод проверяет условие. Если переданное условие возвращает true, то метод завершается нормально, иначе выбрасывает ошибку TestFailedException.

Пример:

```
  test("создание объекта класса Btk_SomeClass") {
    try {
      val rop = Btk_SomeClassApi().insert()
      assert(Btk_SomeClassExtApi().findSomeClassExt(rop).isDefined)
    } finally {
      session.rollback()
    }
  }
```

Данный тест проверяет, что при создании объекта Btk_SomeClass для него создается расширение Btk_SomeClassExt. Если расширение не было создано, то тест не будет пройден. В консоль будет выведен текст ошибки

```
Btk_SomeClassExtApi.apply().findSomeClassExt(rop).isDefined was false
```

В assert можно передать дополнительный комментарий, который будет добавляться к тексту ошибки:

```
  test("создание объекта класса Btk_SomeClass") {
    try {
      val rop = Btk_SomeClassApi().insert()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      assert(rop.get(_.dBeginDate).isNotNull, "При создании не установилась дата начала.
↪")
      assert(Btk_SomeClassExtApi().findSomeClassExt(rop).isDefined, "При создании не␣
↪зарегистрировалось расширение.")
    } finally {
      session.rollback()
    }
  }
```

1. assertResult(expected: Any)(actual: Any)

Сравнивает ожидаемое ожидаемое значение с переданным

```
  test("parseGtkSessionClientFullName") {
    val x = WorkSessionClientHelper.parseGtkSessionClientFullName("DESkTOP-23565:8080
↪#E1@123-45")
    assert(x.isDefined)
    assertResult("DESkTOP-23565:8080")(x.get._1)
    assertResult("E1")(x.get._2)
    assertResult("123-45")(x.get._3)
  }
```

1. assertThrows[T <: AnyRef](f: => Any)

Используется,когда нужно удостоверится, что при определенных методах тестируемый код выдаст ошибку.

```
  test("data.oaObjQty.isEmpty") {
    val data = new Bs_DistrData()
    assertThrows[AppException] {
      Bs_DistributionPkg().distribQty(data)
    }
  }
```

1. assume(condition: Boolean)

Метод аналогичен методу assert, но в случае, когда не выполняется условие, не завершается ошибкой, а отменяет тест. Так же может содержать пояснительный комментарий.

```
assume(database.isAvailable, "База данных не доступна.")
assume(database.getAllUsers.count === 9)
```

1. intercept[T <: AnyRef](f: => Any)

Метод аналогичен методу assertThrows, но позволяет получить ошибку ожидаемого типа для более детальной проверки.

```
        val vEx = intercept[AppException] {
          _api.validateOnBeforeManualInsert(ropDuplicate)
        }
        assert(vEx.getMessage.startsWith("Ошибка. Уже существует запись с классом типа␣
↪объектов:"))
```

18.1.3 Matchers

Трейт Matchers позволяет использовать в тестах комбинаторы вроде shouldBe вместо обычного assert, улучшая читаемость кода.

По умолчанию к ApiTest и LangFunSuite Matchers не подключен, поэтому чтобы использовать его методы, добавьте его в объявлении класса:

```
class Bs_AccApiTest extends ApiTest with Matchers {
  test("set level on insert") {
    val rop = api.insert()
    rop.get(_.nLevel) should ===(1.nn)
  }
}
```

Для дополнительной информации смотрите в руководстве пользователя: matchers

Некоторые примеры:

1. Проверка размера и длины

Для проверки размера и длины используйте:

- <проверяемый объект> should have length (<ожидаемое значение>)
- <проверяемый объект> should have size (<ожидаемое значение>)

Пример:

```
  test("проверка размера массива") {
    val result = getArgs()
    result should have length (3)
  }
```

1. Проверка строк

- на наличие подстроки

```
    string should startWith ("Hello")
    string should endWith ("world")
    string should include ("seven")
```

- с использованием регулярных выражений

```
    string should startWith regex ("Hel*o")
    string should endWith regex ("wo.ld")
    string should include regex ("wo.ld")
```

- проверка всей строки на соответствие регулярному выражению

```
    string should fullyMatch regex ("""(-)?(\d+)(\.\d*)?""")
```

1. Сравнения больше/меньше/равно Можно сравнивать объекты любого типа, кроме тех, которые могут быть преобразованы в Ordered[T]. Для сравнения списков, используйте greater than, less than, greater than or equal, less than or equal to a value of type T

```
    one should be < (7)
    one should be > (0)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    one should be <= (7)
    one should be >= (0)
```

- с использованием регулярных выражений

```
    string should startWith regex ("Hel*o")
    string should endWith regex ("wo.ld")
    string should include regex ("wo.ld")
```

- проверка всей строки на соответствие регулярному выражению

```
    string should fullyMatch regex ("""(-)?(\d+)(\.\d*)?""")
```

18.2 Массовое тестирование модулей

18.2.1 Из консоли sbt

- Для запуска тестов по имени/модулю воспользуйтесь командой testOnly

```
testOnly {путь до класса}.{имя класса}
```

Например:

```
testOnly ru.bitec.app.pro.diagram.Pro_DiagramApiTest
```

Символ * заменяет любое количество символов в наименовании теста, таким образом можно запустить выполнение тестов не по полному имени, а по части наименования. Например, запустить тесты по модулю можно так:

```
testOnly ru.bitec.app.{ваш модуль}.*
```

Например:

```
 testOnly ru.bitec.app.tax.*
```

- Для запуска тестов всего проекта воспользуйтесь командой test

```
 test
```

Примечание: Не рекомендуется использовать, запускайте этот сценарий только на тех проектах, где точно известно, что все тестовые классы не делают нежелательных изменений в файлах или в БД.

- Для запуска только тех тестов, которые связаны с последними изменениями в коде, используйте команду testQuick, которую можно вызывать с такой же фильтрацией, как и testOnly.

18.2.2 Запуск сценариев тестирования с использованием JMeter

- Для запуска скачайте и установите JMeter
- Скачайте пример, запускающего тестирование, и откройте его в `JMeter`:

  Обратитесь к дистрибьютору документации за файлом `files/unitTest.jmx`
- Данный проект содержит три примера сценария:
  - TestByModule - тестирование по модулю
  - TestByClass - тестирование по классу
  - TestAll - запуск всех тестов проекта
- Выберете нужный тред и сделайте его активным (операция контекстного меню Enable. Disable - сделает выбранный элемент неактивным).
- Развернув выбранный тред, отредактируйте параметры компонента run test:
  - При необходимости замените выбранную рабочую директорию, на ту, где лежит исполняемый файл sbt нужного вам проекта.
  - В параметрах замените исполняемую команду (Примеры см. выше), исправьте модуль или имя файла.
- Запустите тестирование через операцию Start на тулбаре. Выполнение тестирования может занять продолжительное время. Дождитесь окончания.
- Результаты выполнения будут отображены в View Results Tree. Для более детального отчета можно посмотреть логи на вкладке «Response data».

18.2.3 Выборочное исключение тестов из массового тестирования

Иногда возникает необходимость временно отключить тест, для этого нужно вызов метода test заменить на ignore. В этом случае при тестировании в результат будет записано, что этот тест проигнорирован.

18.3 Jexl-тесты

В меню `Настройки и сервисы` - `Сервисы Jexl` - `Jexl-тесты` можно писать и выполнять jexl-тесты.

18.3.1 Создание

При создании jexl-теста ему необходимо задать план, который будет состоять из разных шагов, выполняемых друг за другом. Для каждого шага необходимо задать наименование и текст скрипта, который должен выполниться. Порядковый номер шага проставится автоматически.

На закладке «Глобальные переменные» можно задать переменные, к которым можно обращаться в тесте через методы `tst.setGlobalVar` и `tst.getGlobalVar`. Для их создания необходимо указать системное имя переменной, которая должна быть уникальна для теста, и ее значение в виде jexl-скрипта, который будет выполняться перед стартом теста.

Доступные методы

tst.setVar

Записывает в контекст выполнения jexl-теста переданное значение. Информация об изменении переменной записывается в лог выполнения теста с типом `SETVAR`

Пример:

```
tst.setVar("param_name", param_value); //param_name - имя параметра, param_value -␣
↪значение параметра
```

tst.getVar

Достает из контекста значение параметра по его имени.

Пример:

```
var p = tst.getVar("param_name"); //param_name - имя параметра
```

tst.setGlobalVar

Записывает в глобальные переменные выполнения jexl-теста переданное значение. Доступ к этим переменным есть на любом шаге теста, без необходимости получать нужный контекст. Информация об изменении переменной записывается в лог выполнения теста с типом `SETVAR`

Пример:

```
tst.setGlobalVar("param_name", param_value); //param_name - имя параметра, param_value -␣
↪значение параметра
```

tst.getGlobalVar

Получает значение глобальной переменной по ее имени.

Пример:

```
var p = tst.getGlobalVar("param_name"); //param_name - имя параметра
```

tst.findParentContext

Возвращает контекст предыдущего шага выполнения теста. Из него можно получить параметры, записанные в контекст на предудущем шаге.

Пример:

```
var parent = tst.findParentContext();
var parentParam = parent.getVar("param_name");
```

tst.info

Записывает в лог выполнения теста переданный текст с типом `INFO`.

Пример:

```
tst.info("Текст с информацией");
```

tst.warning

Записывает в лог выполнения теста переданный текст с типом `WARNING`.

Пример:

```
tst.warning("Текст с предупреждением");
```

tst.error

Записывает в лог выполнения теста переданный текст с типом `ERROR`.

Пример:

```
tst.error("Текст с ошибкой");
```

tst.raise

Выбрасывает ошибку с переданным текстом, а также записывает ее в лог выполнения с типом `ERROR`.

Пример:

```
tst.raise("Текст с ошибкой");
```

tst.assertTrue

Проверят, что переданное условие истинно. Если оно ложно, то прекращает выполнение теста и записывает в лог выполнения ошибку. Если выполняется группа тестов, и проверка не пройдена то переходит к следующему тесту.

Пример:

```
tst.assertTrue(1 == 2); //запишет в лог "assertTrue не прошло проверку"

tst.assertTrue(1 == 2, "Число 1 не равно числу 2"); //запишет в лог переданный текст
↪"Число 1 не равно числу 2"
```

tst.shouldBeTrue

Проверят, что переданное условие истинно. Если оно ложно, то записывает в лог выполнения ошибку.

Пример:

```
tst.shouldBeTrue(1 == 2); //запишет в лог "shouldBeTrue не прошло проверку"

tst.shouldBeTrue(1 == 2, "Число 1 не равно числу 2"); //запишет в лог переданный текст
↪"Число 1 не равно числу 2"
```

tst.sqlRows

Возвращает количество строк, полученных в результате выполнения переданного запроса

Пример:

```
var count = tst.sqlRows(`select t.id from RplTst_AllDbTypes t where t.idObjectType =␣
↪120`);
```

18.3.2 Выполнение

Выполненение теста осуществляется в помощью операции `Выполнить тест` в карточке теста или списке тестов. Для выполнения тестов целой группы необходимо в списке тестов выполнить операцию `Выполнение всех тестов текущей группы`.

Выполнение с помощью Rest сервиса

Выполнение тестов также возможно с помощью Rest сервиса.

Путь для выполнения одиночного теста: http://{имя_сервера}/app/sys/rest/ss/pkg/Bts_TestPkg/runSingleTest

Путь для выполнения группы тестов: http://{имя_сервера}/app/sys/rest/ss/pkg/Bts_TestPkg/runTestForGroup

Тело обоих запросов является json-объектом, который состоит из ключа `name` и его значения, в виде системного имени теста или группы, соответственно.

Запрос выполнения одиночного теста возвращает json-объект, который содержит в себе следующие параметры: имя теста `name`, наличие ошибок `hasErrors`, наличие предупреждений `hasWarnings`, количество ошибок `countErrors` и количество предупреждений `countWarnings`.

Пример для одиночного теста:

```
//пример обращения
curl -H Database:pgDev -u admin:admin -d {\"name\":\"PRS_PurchReqTest\"} -H "Content-
↪Type: application/json" -X POST http://localhost:8080/app/sys/rest/ss/pkg/Bts_TestPkg/
↪runSingleTest

//пример результата
{"hasErrors":true,"hasWarnings":false,"countErrors":1,"countWarnings":0,"name":"Тест␣
↪заявок на закупку услуг"}
```

Запрос выполнения группы тестов возвращает json-объект, который содержит в себе следующие параметры: наличие ошибок `hasErrors`, наличие предупреждений `hasWarnings`, количество ошибок `countErrors`, количество предупреждений `countWarnings` и результаты выполнения всех тестов `tests` в виде массива json-объектов таких же, как при выполнении одиночного теста.

```
//пример обращения
curl -H Database:pgDev -u admin:admin -d {\"name\":\"Test1\"} -H "Content-Type:␣
↪application/json" -X POST http://localhost:8080/app/sys/rest/ss/pkg/Bts_TestPkg/
↪runTestForGroup

//пример результата
{"hasErrors":true,"hasWarnings":false,"countErrors":2,"countWarnings":0,"tests":[{
↪"hasErrors":false,"hasWarnings":false,"countErrors":0,"countWarnings":0,"name":"test_ok
↪"},{"hasErrors":true,"hasWarnings":false,"countErrors":1,"countWarnings":0,"name":
↪"test333"},{"hasErrors":true,"hasWarnings":false,"countErrors":1,"countWarnings":0,
↪"name":"test22"}]}
```

Часть V

Отчеты

Глава 19

Отчеты

Отчеты системы GlobalFramework лежат в базе данных, что позволяет легко перекрывать их на проектах. Для управления отчетами необходимо зайти в приложение Global `Настройка системы`

19.1 Типы шаблонов печатных форм

Определяют формат, в котором задаются шаблоны печатных форм.

- jasper
  Шаблон представляет из себя zip архив содержащий набор xml файлов, сформированных в *jasper studio*. Jasper studio это специализированное средство построения печатных форм.
- xlsx
  Шаблон представляет из себя файл в формате xls со спец тэгами
- docx
  Шаблон представляет из себя файл в формате docx со спец тэгами

19.2 Печатная форма

Печатная форма (сокр. `ПФ`) определяет правила построения отчета, а также доступность этого отчета для приложения, *типа объекта* и пользователя. При выполнении печатной формы формируется отчет в виде файла с заданным форматов.
Печатную форму можно подключить к выборке на проекте без внесения изменений в программный код.

19.2.1 Версии печатной формы

Версия печатной формы позволяет:

- Безопасно вернутся к предыдущей реализации
- Запускать реализацию в зависимости от периода

Версия печатной формы содержит:

- Тип шаблона печатной формы
- Бинарный файл для построителя отчетов
- Дату версии
- Описание

Доступные форматы

В случае если тип шаблона печатной формы поддерживает конвертацию в другие форматы можно указать перечень доступных форматов. В случае если указано несколько доступных форматов, то при печати печатной формы пользователю будет задан вопрос в каком формате построить отчет.

19.2.2 Подписи для печатных бланков

В случае если отчет должен быть подписан после печати. В запрос печатной формы необходимо добавить атрибуты необходимые для формирования блока подписания.

```
with signs as(select * from jsonb_array_elements( $P{SIGNDATA_DZ}::jsonb ->'data'))
select t.*
, (select s.value ->>'sFIO' from signs s WHERE  s.value->>'idBlankSignTypeMC' =
↪'matching1') as sMatching1
, (select s.value ->>'sPosition' from signs s WHERE  s.value->>'idBlankSignTypeMC' =
↪'matching1') as sMatching1Pos
```

На печатный бланк необходимо вывести блок с подписями.

Блок с подписями формируется перед печатью и передается в отчет в формате json, в параметр SIGNDATA_DZ пример:

```
{
        "data": [
                {
                        "idBlankSignTypeMC": "HeadOfDepartment",
                        "bReadOnly": 0,
                        "idBlankSignTypeHL": "Руководитель подразделения",
                        "nOrder": 1,
                        "dDate": "10.12.2020",
                        "idBlankSignType": 95401,
                        "sPosition": "Начальник отдела",
                        "sFIO": "Дьяченко Евгений Владимирович"
                }
        ]
}
```

Где:

- idBlankSignType
  идентификатор тип подписи
- idBlankSignTypeMC
  код типа подписи
- nOrder
  порядковый номер подписи
- sPosition
  Должность
- sFIO
  ФИО
- dDate \ дата

Блок формируется на основе настроек данной закладки.
Настройки могут быть переопределены\уточнены на конкретном объекте в закладке `Подписи` (`Bs_OBjectSign`)

19.2.3 Аудит построения отчета

Содержит информацию о построении отчетов пользователями.

19.2.4 Параметры отчета

Параметры отчета передаются в запрос, формирующий данные отчета.

Пользовательские параметры

Настраиваются на печатной форме. Перед выполнением отчета пользователь может ввести данные в эти параметры.

Служебные параметры

Формируются автоматически

- `SIGNDATA_DZ` - Блок подписи
- `IDUSER` - Пользователь
- `IDSRCOBJECT` - Объект источник
  Идентификатор объекта от которого выполняется печатная форма.
- `IDSRCCLASS` - Класс источник
  Идентификатор класса объекта от которого выполняется печатная форма

19.3 Вызов печатных форм

Печатные формы могут вызываться:

- из операции выборки
- *из запроса к сервису печати*
- *из rest сервиса*
- От произвольного объекта системы
- От интерфейса свободные отчеты

19.3.1 Вызов печатной формы из операции выборки

Для вызова отчета из операции выборки используйте функцию `ru.bitec.app.gtk.gl.Reports#createReportEx`:

```
  /**
    * Выполняет построение отчёта по системному имени отчёта.
    *
    * @param reportName        Имя отчёта
    * @param reportVersionDate Дата
    * @param postBuildAction   Действие, которое необходимо произвести после заполнения␣
↪отчёта
    * @param propertyMap       Карта входящих параметров
    */
  @throws[ApplicationException]
  def createReportEx(reportName: String, reportVersionDate: Date, postBuildAction:␣
↪PostBuildAction, propertyMap: Map[String, Any]): Unit
```

Данная функция может быть вызвана только в контексте интерактивной бизнес логики(интерфейса пользователя).

Пример:

```
reports.createReportEx("Mct_OrderSheetMaterials2", null, PostBuildAction.print,
        Map[String, Any]("IDSRCOBJECT" -> getVar("id").asNLong,
          "GIDSRCOBJECT" -> getVar("gid").asNString,
          "SDESIGNATION" -> getVar("sCode").asNString)
      )
```

19.3.2 Формирования файла с отчетом

Для формирования файла с отчетом используйте функцию `ru.bitec.app.rpt.Rpt_Pkg#getReportStreamEx`:

```
  /**
    * Выполняет построение отчёта по системному имени отчёта.
    * Если для версии отчета указано несколько доступных для печати форматов,
    * то будет выдана ошибка построения.
    * Требуется явно указать формат построения, указав параметр [[Rpt_Pkg.
↪ParamFormatType]]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    *
    * @param reportName         Имя отчёта
    * @param reportVersionDate Дата
    * @param propertyMap        Карта входящих параметров
    * @return InputStream, содержащий результат построения отчёта. ByteArrayInputStream␣
↪не требует закрытия.
    */
  def getReportStreamEx(reportName: String, reportVersionDate: Date, propertyMap:␣
↪Map[String, Any]): Option[InputStream]
```

Данная функция может быть вызвана в контексте автономной логики(rest сервиса)

Внимание: В случае если получен поток, его необходимо обязательно закрыть.

При формировании файла с отчётом в потоке переменные выборки не доступны.

19.3.3 Вызов печатной формы от произвольного объекта

При открытии карточки любого объекта, выводятся стандартные операции печати по которым можно выполнять вызов печатных форм.

Для добавления печатной формы к списку печати для типа объекта:

1. Откройте приложение `Настройка системы`
2. Откройте типы объектов
   Выполните пункт меню `Сущности > Типы объектов > Типы объектов`
3. Перейдите на вкладку `Печатные формы`
4. Добавьте необходимые печатные формы

19.3.4 Вызов печатной формы в свободных отчетах

Свободные отчеты позволяют настроить для пользователя и приложения перечень отчетов, которые можно построить без привязки к каким-либо типам объектов.

Для вызова интерфейса построения свободных отчетов:

1. Откройте приложение, в котором есть пункт меню `Отчеты`
2. Откройте свободные отчеты
   Выполните пункт меню `Отчеты > Свободные отчеты`
3. Выберите нужный отчет
4. Заполните параметры
5. Напечатайте отчет
   Для этого выполните операцию `Печать`

Для того чтобы ПФ могла быть вызвана из свободных отчетов:

1. Откройте приложение `Настройки системы`
2. Откройте печатные формы
   Пункт меню `Отчеты > Печатные формы`

3. Выберете требуемую печатную форму
4. Включите признак `Свободный отчет`
5. Настройте параметры отчета
6. Укажите приложение для печати
7. Укажите требуемые роли

Примечание: Если роли не указаны, печатная форма будет доступна всем пользователям

19.4 Создание печатной формы

1. Откройте приложение `Настройки системы`
2. Откройте печатные формы
   Пункт меню `Отчеты > Печатные формы`
3. Выполните операцию `Создать`
4. Заполните `Системное имя`, `Наименование`, `Модуль`
5. Создайте версию печатной формы
6. Выберете тип шаблона
7. Загрузите файл шаблона выбранного типа
   Для этого выполните операцию `Загрузить файл в базу`

19.5 Настройки вставки изображений в печатную форму типа docx

Данная настройка производится в коллекции к печатной форме. Все поля данной коллекции обязательны для заполнения.

- `Активность`
  Данное поле отвечает будет ли вставленные изображения в печатную форму
- `Шаблон(Тег)`
  Определяет тег по которому будет вставлено изображение. Тег должен содержаться в печатной форме иначе изображение не будет вставлено. Существует два вида тегов:
  - `Общий`
    Общий тег не имеет динамической части и изображение будет вставлено при нахождении данного тега в документе.
    * Пример:
      · В поле: SomeTag
      · В документе: [SomeTag]
  - `Подпись`
    Тег для подписи состоит из двух частей: общая часть для всех тегов, которые задаётся в данном поле, и имя пользователя подпись которого необходимо вставить в данный документ
    * Пример:

        - В поле: SomeSignTag
        - В документе(с динамической частью): [SomeSignTagAdmin]

- `Печатная форма изображения`
  В поле указывается печатная форма с типом jasper и форматом png. В зависимости от типа изображения данный печатной форме будут переданы следующие аргументы:
  - `Подпись`
    - «PSDATE» - дата подписания документа
    - «PSSERNUMBER» - № сертификата подписи
    - «PSDBEGIN» - дата начала действия подписи
    - «PSDENG» - дата окончания действия подписи
    - «PSFIO» - ФИО сотрудника подписавшего документ
    - «IDDOC» - id документа
    - «IDDOCVER» - id версии документа(если документ версионный то подписывается именно версия а не сам документ)
    - «JOBJ» - json объект подписи документа
  - `Общий`
    - «IDDOC» - id документа
    - «IDDOCVER» - id версии документа(если документ версионный то подписывается именно версия а не сам документ)
- `Масштаб` и `Разрешение`
  Размер изображения высчитывается по формуле: Масштаб *= Measures.DOTS_PER_INCH / Разрешение
- `Тип`
  В поле указывается тип изображения:
  - `Общий`
    Изображение вставляется при печати в любом случае
  - `Подпись`
    Изображение вставляется если на документе есть хотя бы одна подпись

19.6 Вставка изображений в печатную форму docx

Для вставки изображений в печатную форму типа docx должны быть выполнены следующие условия:

1. Печатная форма должна содержать теги вида [SomeTag]
2. Тип шаблона печатной формы - docx
3. Формат печатной формы - pdf
4. В коллекции к печатной форме(Настройки вставки изображений в печатную форму) настроены необходимые изображения для вставки

Если все требования выполнены корректно, то при печати такой печатной формы будет получен pdf файл с изображениями на месте тегов.

Глава 20

Jasper Reports

Java библиотека для построения печатных отчетов. Позволяет строить:

- Готовые к печати PDF-файлы в интерактивном динамическом HTML с навигацией внутри или за пределами отчета
- Высококачественные документы PowerPoint, RTF, Word
- Электронные таблицы или необработанные CSV, JSON или XML

Примечание: Данный раздел не является полным руководством к средству построения отчётов JasperReport. Документ содержит описания базовых принципов создания шаблонов и способов взаимодействия инфраструктуры Global 3 SE с построителем отчётов JasperReports.

20.1 Дистрибутив

Скачать дистрибутив можно

- с ftp по адресу `ftp://ftp.bitec.ru/pub/#Distrib/JasperStudio`

Для разработки шаблонов отчётов доступны две IDE:

- Jaspersoft® Studio
- iReport Designer.

Примечание: Данный раздел касается только Jaspersoft Studio.

20.2 Jaspersoft Studio

Программное обеспечение, которое позволяет создавать и редактировать шаблоны JasperReports. С помощью Jaspersoft Studio возможно:

- Разрабатывать и запускать шаблоны отчетов
- Создавать запросы к отчетам
- Писать сложные выражения
- Компоновать визуальные компоненты

Внимание: Перечень шрифтов в дизайнере Jaspersoft Studio может отличатся от перечня шрифтов на сервере приложения. Используйте стандартные шрифты, к примеру `Arial`

20.3 Обучающее видео

Виде выложено на Ftp по адресу `ftp://ftp.bitec.ru/pub/#Global/Video/Обучение/Создание ПФ отчетов в Jasper`

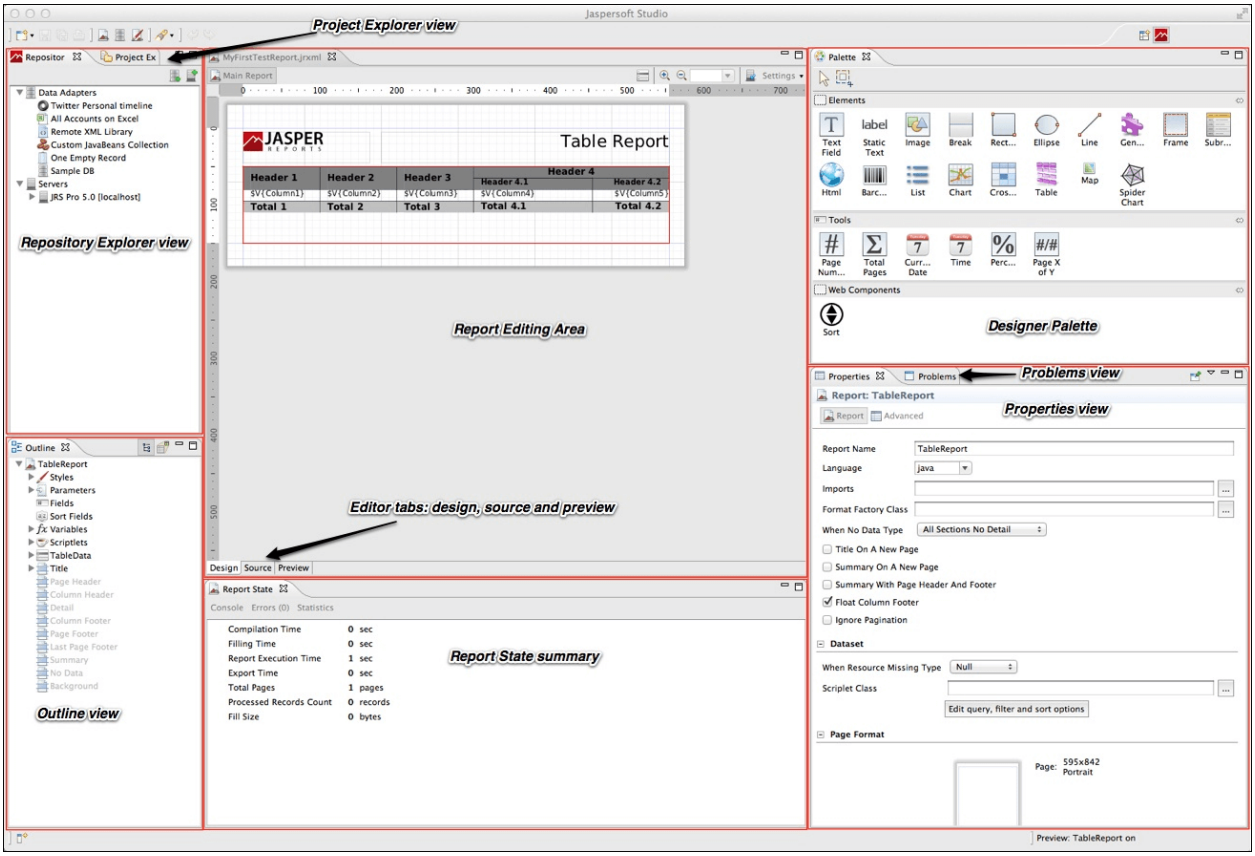
20.4 Документация JasperReports

- Ресурсы с jaspersoft
- jaspersoft Wiki
- Руководство пользователя по адресу `ftp://ftp.bitec.ru/pub/#Distrib/JasperStudio`

Для того чтобы открыть примеры Jaspersoft Studio:

1. Выполните `File > New > Project > JasperReports Samples`.
   В выбранном рабочем каталоге будет создано множество примеров отчётов.

20.5 Пользовательский интерфейс



20.5.1 Repository

Содержит перечень подключений к разным базам данных. Подключения к базе данных необходимо настраивать для того чтобы была возможность тестировать отчеты из Jaspersoft Studio.

20.5.2 Outline

Отображает структуру открытого отчета.

20.5.3 Properties

Отображает свойства элемента выбранного в `outline`.

20.5.4 Editor

Отображает открытый отчет. Содержит следующие закладки:

- Design
- Source
- Preview

20.6 Колонки

Страницы отчета содержат банды(группы визуальных элементов), независимые от данных (например, заголовок или нижние колонтитулы страницы), и банды, которые печатаются только при наличии одной или нескольких записей из данных для печати. Банды с данными можно разделить на вертикальные столбцы, чтобы максимально использовать доступное пространство на странице. Количество колонок можно задать в диалоге разметки страницы(`properties\reports\page format` для `outline\{Report}`)

20.7 Bands

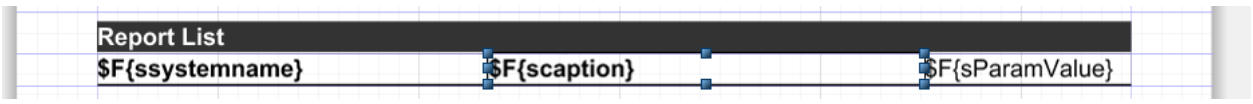
Отчет делится на банды, в которые добавляются визуальные элементы. Список основных предопределенных бандов:

- `Title`
  Это первый видимый банд. Он создается только один раз и может быть распечатан на отдельной странице
- `Page Header`
  Позволяет определить заголовок страницы
- `Column Header`
  Печатается перед началом отрисовки колонки. Количество колонок на странице можно задать в
- `Column Footer`
  Печатается в конце каждой колонки
- `Page Footer`
  Появляется в конце каждой страницы
- `Summary`
  Содержит итоги по отчету

20.7.1 Detail band

Печатается для каждой записи данных отчета. Детальных бандов может быть несколько, поэтому по умолчанию создается детальный банд с именем „detail 1“

В детальной части можно расположить «Text Field» и задать для каждого выражения получения данных.

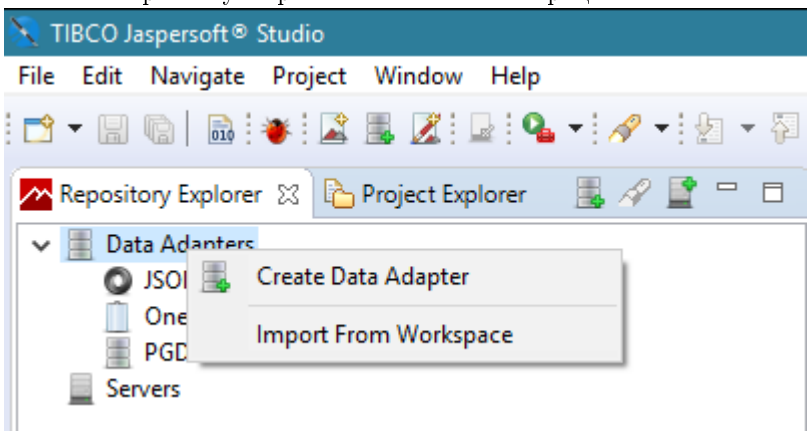


20.8 Настройка среды Jaspersoft Studio

Запустите среду разработки отчётов Jaspersort Studio

20.8.1 Создание подключения к ДБ

1. В окне «Repository Explorer» выполните операцию «Create Data Adapter»



2. В открывшемся диалоге выберите «Data base JDBC connectivity»
3. Укажите реквизиты подключения к базе Postgres
   - `name=PGDEV`
   - `JDBC driver=org.postgresql.Driver`
   - `JDBC url=jdbc:postgresql://{host}/{db}`
   - `Username ={username}`

Примечание: Подключение к БД необходимо для того чтобы тестировать отчет из студии

20.9 Проект отчета

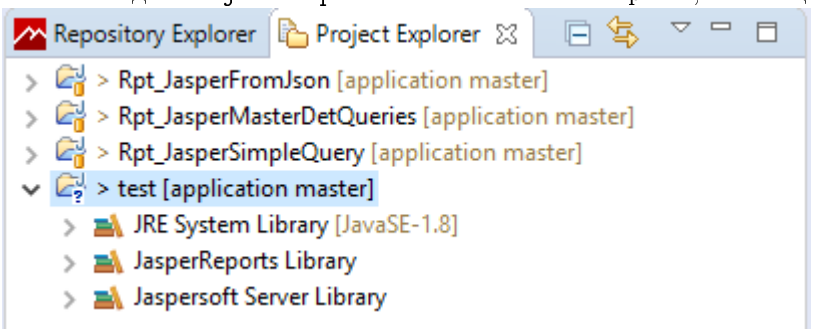
Проект отчета создается на группу печатных форм, на усмотрение аналитика.

Совет: Печатные формы проще создавать и редактировать в одном проекте если они не содержат скрипелтов.

Для создания проекта отчёта:

1. Выполните `File > New > Project ...`
2. Выберите `Jaspersoft Studio / JasperReports Project`
3. Задайте имя отчёта в следующем диалоге.

По завершению на закладке `Project Explorer` появится новый проект, пока ещё не содержащий шаблонов отчёта.



20.10 Создание простой печатной формы

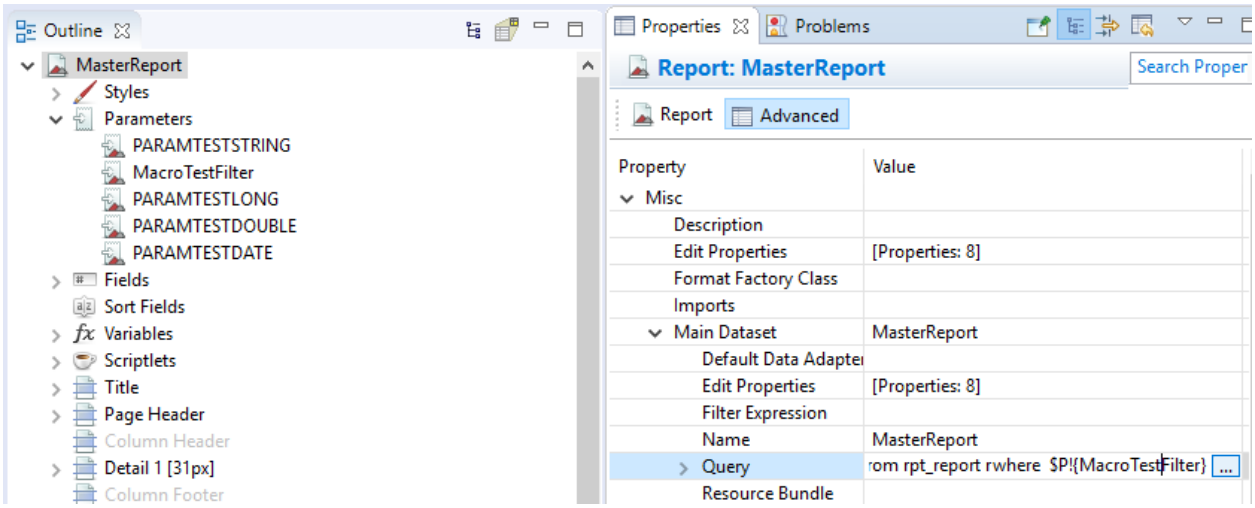
Простейший отчёт, выводящий список на основе SQL-запроса.

1. В окне `Project Explorer` выберете требуемы проект
2. Создайте каталог для печатной формы
   В контекстном меню выполните пункт `New > folder`
3. Создайте отчет
   В контекстном меню выполните пункт `New > Jasper Report`
4. Задайте имя шаблона отчета `main.jrxml`
   Файл с этим именем будет искаться при построении отчёта из Global 3 SE.
5. Нажмите кнопку `next`
6. Выберете адаптер базы данных
7. Нажмите кнопку `next`
8. Введите запрос
9. Нажмите кнопку`next`
10. Добавьте требуемые поля в соответствующие банды
11. Нажмите кнопку `finish`
12. Проверьте корректность отчета на закладке `Preview` в окне `Editor`
13. *Опубликуйте отчет в БД*

Редактирование запроса с данными:

1. Откройте отчет
2. В окне outlet выберете корень
3. Нажмите кнопку `Edit query, filter and sort options`
   Кнопка находится в окне Properties по адресу `Report \ DataSet`

Так же запрос можно посмотреть в окне Properties на закладке`Advanced`



20.11 Двухуровневый отчёт (мастер-деталь)

Документация в wiki

1. Создайте файл мастер-шаблона `main.jrxml`(имя обязательно)
   Аналогично одноуровнему шаблону.
2. Создайте файл детального-шаблона `subreport.jrxml`
   В данном шаблоне будут отображать детальные данные. Имя может быть любым.
3. Добавьте в мастер `Subreport`
   Откройте мастер шаблон и в окне outlet в секцию `Detail 1` добавьте элемент `Subreport`, в свойствах которого укажите
   - Имя файла детального шаблона
     Expression=`"subreport.jrxml"`
   - Подключение к БД
     Connection Expression = `$P{REPORT_CONNECTION}`
4. Свяжите запросы двух шаблонов параметрами.
   1. Создайте в детальном шаблоне параметр `ID_MASTER`
   2. Параметризуйте детальный запрос
      Обращение к параметру в детальном запросе `$P{ID_MASTER}`
   3. Свяжите параметр детального шаблона с мастером \
      1. Зайдите в раздел «Subreport» свойств мастер-шаблона
      2. Вызовите диалог редактирования свойства «Edit Parameters»
      3. Задайте связь
         Для этого создайте параметр `ID_MASTER` с выражением `$F{id}`

Совет: Для вывода актуальных данных в предпросмотре мастер отчета,
после редактирования детального отчета, следует вызвать перекомпиляцию.

20.12 Отчет книга (Report Book)

Отчет книга – это .jrxml, который объединяет несколько отчетов в единый объект.

Документация в wiki

1. Создайте новый отчет-книгу
    1. Кликните правой кнопкой мыши в `Project Explore` и выполните пункт `New > Jasper Report`
    2. В новом мастере создания отчета выберет `Report Books \ Empty Book`
    3. Укажите имя `main.jrxml` (имя обязательно)
2. Создайте отчет, который будет отображаться в книге.
   Отчет может быть как одноуровневым, так и двухуровневым.
3. В отчете-мастере вызовите контекстное меню и выполните пункт «Add new book part»
4. Добавьте отчет, созданный в пункте 2
5. Свяжите отчеты между собой Связь запросов двух отчетов параметрами происходит аналогично двухуровневому отчету

Внимание: Отображение номеров страниц в отчетах, отображаемых в книге осуществляется через переменную `$V{MASTER_CURRENT_PAGE}`, а общее количество страниц – через `$V{MASTER_TOTAL_PAGES}`. В настройках текстового поля должно быть указано `evaluationTime="Master"`

20.13 Параметризация шаблонов

Документация в wiki

Параметры отчета находятся в окне `Outline` по адресу `{Report} \ parameters` Параметры отчёта используются для следующих целей:

- В SQL-запросах
  Для ограничения данных по переданным из вне параметрам
- Связи между частями шаблонов
- Для связи между движком отчёта и внешним окружением.

Параметры могут быть любого java-типа.

Например:

- `REPORT_CONNECTION` тип `java.sql.Connection`
- `REPORT_PARAMETERS_MAP` тип `Map<String,Object>`

Шаблон отчёта содержит множество служебных параметров. В дереве элементов шаблона они отображаются серыми. Их невозможно удалить, или изменить значение из дизайнера. Их значения могут быть переданы извне или изменены в коде.

Параметр имеет значение по умолчанию. Его удобно использовать при разработке и тестировании отчёта, заменяя передачу значения параметра извне.

Внимание: При задании строкового значения по умолчанию, его необходимо брать в двойные кавычки.

Флаг `Is For Prompting` означает, что при тестовом построении отчёта из среды Jaspersoft Studio будет запрошено значение параметра.

Внимание: Создавайте параметры с именами в верхнем регистре. Это связано с тем, что имена параметров, переданных извне, приводятся к верхнему регистру автоматом.

20.13.1 Передача параметров в отчёт

В отчёт можно передать значение параметра с любым типом данных, в том числе служебных параметров. Так же, через параметры можно передать значения макросов фильтра, сортировки или весь запрос целиком.

```
@Oper(
    caption = "Печать",
    description = "Печать",
    imageIndex =17,
    order = 100
)
def printReport(): Unit = {
    createReportEx(
        getVar("sSystemName").asString(),
        null,
        PostBuildAction.print,
        Map[String, Any](
            "ParamTestLong" -> 100500.nl,
            "ParamTestDouble" -> NNumber.fromAny(100500.100500D),
            "ParamTestString" -> "Тестовая строка",
            "ParamTestDate" -> NDate.now(),
            "MacroTestFilter" -> "(1 = 1)"
        )
    )
}
```

Карта значений всех переданных извне параметров доступна в служебном параметре `REPORT_PARAMETERS_MAP`.

Значение можно получить выражением:

```
$P{REPORT_PARAMETERS_MAP}.get("ИМЯ_ПАРАМЕТРА")
```

Если в шаблоне создать параметр `MY_PARAMETER`, имя которого в верхнем регистре равно имени переданного в отчёт параметра, значение `MY_PARAMETER` будет проинициализировано переданным значением. Это позволяет использовать переданные значения параметров в SQL-запросах.

Пример использования, в отчёте: `Rpt_JasperSimpleQuery`

20.13.2 Использование параметров в SQL-запросах

Подстановка значения

Подстановка значения параметра в запрос выполняется выражением $P{ИМЯ _ПАРАМЕТРА}. Следите за регистром имён параметров, в случае несовпадения будет ошибка компиляции.

```
select o.*, $P{MY_ORDER_ID} as idMy
  from orders o
 where order_id = $P{MY_ORDER_ID}
```

В данном случае будет выполнен sql запрос:

```
select o.*, ? as idMy from orders o where order_id = ?
```

Значение параметра отчёта будет передано как значение sql-параметра.

Подстановка выражения (макроса)

Для подстановки в sql-запрос выражения (или полного текста sql-запроса) необходимо использовать синтаксис `$P!{ИМЯ _ПАРАМЕТРА}`.

```
select * from orders order by $P!{ИМЯ _ПАРАМЕТРА}
select * from orders where $P!{ИМЯ _ПАРАМЕТРА}
$P!{MY_QUERY}
```

20.13.3 Использование параметров, содержащих список значений

В отчет можно передавать массив значений в одном параметре. Для этого в `REPORT_PARAMETERS_MAP` в значении параметра нужно передать массив `Array[java тип]()`.

Например:

```
"TEST" -> Array[Long](23553,23552)
```

Настройка параметра с множеством в Jasper Studio

В Jasper Studio нужно выбрать тип для параметра, которому будет передаваться массив значений:

1. Перейдите к параметрам отчета
2. Откройте свойства параметра
3. Задайте «Class» значением `ArrayList`
4. Задайте «Nested type name»
   К примеру `Long` или `String`

Использование параметров с null

Для использования в SQL параметров которые могут быть принимать значение null необходимо использовать синтаксис `$X{ИМЯ_ФУНКЦИИ, ИМЯ_КОЛОНКИ, ИМЯ_ПАРАМЕТРА}`.

Где `ИМЯ_ФУНКЦИИ`

- `EQUAL`
- `NOTEQUAL`
- `LESS`
- `GREATER`
- и др.

Для запроса

```
select * from orders where $X{EQUAL, num_column, num_param}
```

Будут сгенерирован текст:

```
select *  from orders where num_column = 1
```

Или если параметр null

```
select *  from orders where num_column IS null
```

20.13.4 Управляющие параметры

На процесс построения и экспорта отчёта могут влиять:

- Служебные параметры отчёта
- Свойства контекста построения отчёта

Служебные параметры отчёта.

Имя	Возможные значения	Описание
REPORT_LOCALE	Locale.forLanguageTag(«ru-RU»)	Устанавливает локаль, используемую при построении отчёта

Свойства контекста построения отчёта.

Имя свойства контекста построения отчета начинается с `net.sf.jasperreports` Полный список можно найти в документации JasperReports

Значения по умолчанию определены в ресурсах JasperReports:

```
jasperreports-x.x.x.jar\default.jasperreports.properties
```

Значения могут быть переданы через параметры метода `createReportEx()` или определены с свойствах шаблона («Properties \ Advanced \ Misc \ Edit Properties»)

Дополнительные управляющие параметры

Параметры, передающиеся в отчет стандартной операцией печати:

- `idSrcObject`
  id объекта от которого создан отчет
- `idSrcClass`
  id класса объекта от которого создан отчет
- `idUser`
  id пользователя
- `signData_dz`
  json, подписи печатных бланков для отчетов
  Пример значения:

```
[
    {
        "sFIO": "Калинин В.М.",
        "sPosition": "Транспортировщик",
        "idEmployee": 25712,
        "idDepartment": 101672,
        "sBasisDocument": "  №036/426-06-2020-ОЛПС           от 01.01.2020",
        "idBlankSignType": 95351,
        "gidBasisDocument": "139800/5433"
    }
]
```

20.14 Экспорт

По умолчанию, отчёт экспортируется в pdf. Формат, в который будет выполнен экспорт, определяется параметром `FILENAME`.

На экспорт влияют *параметры контекста построения*, имена которых начинаются с `net.sf.jasperreports.export`.

Описание параметров:

- `FILENAME`
  Определяет имя и формат файла с результатом построения отчёта. Отчёт может быть экспортирован в следующие форматы: pdf, xls, docx, xlsx, pptx, xml, json, html, ods, odt, rtf, txt, jpg, jpeg, gif, png.
  По умолчанию, экспортируется в pdf. Возможные значения с AS 1.18: \

```
{name} | {name.} | {name.ext} | {.ext}
```

  Возможные значения до AS 1.17: \

```
{name.ext} | {.ext} | {ext}
```

20.14.1 Растровые изображения

Дополнительные параметры контекста, отсутствующие в списке стандартных свойств контекста JasperReports.

- `net.sf.jasperreports.export.graphics2d.page.index` тип `Integer`
  Номер экспортируемой страницы.
- `net.sf.jasperreports.export.graphics2d.zoom.ratio` тип Float
  Коэффициент увеличения. По умолчанию, страница формата A4 имеет размеры 842x595px, 72dpi. При экспорте в растровое изображение с такими параметрами, читаемость текста крайне низка. Параметр позволяет пропорционально увеличить размеры изображения без потери качества.
- `net.sf.jasperreports.export.graphics2d.offset.x` тип `Integer`
  Сдвиг по оси X
- `net.sf.jasperreports.export.graphics2d.offset.y` тип `Integer`
  Сдвиг по оси Y

GIF и PNG с прозрачностью

Для экспорта страницы отчёта в gif или png с прозрачным фоном необходимо отключить заливку фона белым цветом. Параметр:

- `net.sf.jasperreports.export.graphics2d.white.page.background` тип `boolean`
  Управляет заливкой фона страниц отчёта белым цветом

20.15 Построение отчёта на основе JSON-данных

Документация в wiki

Для построения отчёта на основе JSON-данных:

1. Создайте xxx.json файл содержащий данные в json формате
   Файл должен находится в каталоге с шаблоном. Пример содержимого:

```
{"Northwind": {
"Customers": [
    {
    "Phone": "030-0074321",         // nonstandard unquoted field name
    "PostalCode": 12209,         // nonstandard single-quoted field name
    "ContactName": "Maria Anders",         // standard double-quoted field name
    "Fax": "030-0076545",
    "Address": "Obere Str. 57",
    "CustomerID": "ALFKI",
    "CompanyName": "Alfreds Futterkiste",
    "Country": "Germany",
    "City": "Berlin",
    "ContactTitle": "Sales Representative"
    }
]
}}
```

2. В «Repository Explorer» создайте JSON-источник данных
3. Перейдите в свойства «Main DataSet \ Query» шаблона

4. Выберете созданный источник JSON-данных
5. Установите Language = JSON.
6. Заполните выражения запроса
   Пример:

   ```
   Northwind.Customers(Country == $P{Country})
   ```

   Более подробную информацию по синтаксису смотрите в вики.
   Для автоматического формирования запроса кликните по требуемому узлу JSON дерева(слева от выражения запроса)
7. Нажать «Read Fields» для создания полей на основе JSON-данных

Теперь к полям источника данных можно обращаться как к полям SQL-запроса `$F{ПОЛЕ}`.

Для передачи данных в функцию `createReportEx()` используйте параметр `JSON_INPUT_STREAM` с типом `java.io.InputStream` содержащим JSON-данные.

пример:

```
val json = """{json-данные}"""
createReportEx(
    getVar("sSystemName").asString,
    null,
    PostBuildAction.print,
    Map[String, Any](
        "JSON_INPUT_STREAM" ->
            IOUtils.toInputStream(json, "UTF-8")
    )
)
```

Также доступны параметры:

- `JSON_LOCALE`
- `JSON_TIME_ZONE`
- `net.sf.jasperreports.json.date.pattern`
- `net.sf.jasperreports.json.number.pattern`

20.16 Скриптлет (Scriptlet)

Скриптлет - это Java-класс, позволяющий кастомизировать обработку событий построения отчета а так же добавить вспомогательные функции

Дополнительная информация:

- all-you-want-know-about-scriptlets
- hello-world-sample-example
- sample reference

Скриптлет наследуется от одного из классов:

- `net.sf.jasperreports.engine. JRAbstractScriptlet`
- `net.sf.jasperreports.engine.JRDefaultScriptlet`

Экземпляр данного класса создаётся при заполнении отчёта, и его методы вызываются на различных этапах в качестве обработчиков. Так же скриптлет может содержать пользовательские методы, которые можно вызвать через выражения «Expression» полей, переменных и т.п.

20.16.1 Создание скриптлета

Для создания скриптлетов необходимо иметь Java-проект, к которому подключена библиотека:

```
"net.sf.jasperreports" % "jasperreports" % "6.5.1"
```

Перечень действий:

1. Создайте Java-класс \

```
public class JRMyExampleScriptlet extends
     net.sf.jasperreports.engine.JRDefaultScriptlet {

}
```

2. Переопределите необходимые методы-обработчики событий
3. Если требуется добавьте необходимые функции

Для создания и компиляции класса скриптлета, можно использовать проект `Application`. Для этого необходимо подключить к модулю, в котором будет создан класс скритплета, библиотеку:

```
settings(
    libraryDependencies ++= Seq(
        "net.sf.jasperreports" % "jasperreports" % "6.5.1" %
        "compile"
        excludeAll(*ExclusionRule*("com.lowagie"),*ExclusionRule*("org.olap4j"))
    )
)
```

При этом, если построение отчёта будет запущено из приложения отлаживаемого в IDEA, будет возможна отладка кода скриптлета.

20.16.2 Подключение скритплета к шаблону

Видеоурок

1. Соберите Jar, содержащий данный класс
2. Скопируйте полученный Jar в каталог проекта версии отчёта
   Так же допустимо не создавая Jar, скопировать структуру каталогов с class-файлом скриптлета в каталог проекта отчёта
3. Подключите скриплет к дизайнеру JasperReport Studio
   1. Кликните правой кнопкой мыши на каталоге проекта и выберите «Buld path\ Configure Build Path»
   2. Если нужно подключите jar
      Для этого выполните «Add JARs» и выберете jar-файл из каталога проекта версии отчёта
   3. Если нужно подключите каталог
      Для этого выполните «Add Class Folder. . . » и выберете каталог

4. Создайте новый скриплет в отчете
   Новый скриплет создается из списка скриплетов отчета в окне `outline` по адресу `{Reports} \ Scriptlets`

5. В свойствах скриплета укажите класс созданного скриплета

Теперь при выполнении компиляции и просмотра отчёта в JasperReports Studio будут вызываться методы созданного скриптлета.

20.16.3 Пример скриплета для реализации суммы прописью

Rpt_JRScriptletNumberToStrSample

Сумма прописью реализована в библиотеке numbertostr: «ftp://ftp.bitec.ru/pub/%23Global/utils/NumberToStr/numberto
all-1.0-SNAPSHOT.jar»

Эта библиотека поставляется с сервером global

Для работы в Jasper Studio необходимо подключить эту библиотеку к проекту отчета.

скриплет – обертка

Создание скриплета:

1. создайте в проекте файл скриплета Файл создается в проекте отчета в каталоге `src`

```
import net.sf.jasperreports.engine.JRDefaultScriptlet;
import net.sf.jasperreports.engine.JRScriptletException;
import ru.bitec.numbertostr.MoneyToStr;
import ru.bitec.numbertostr.settings.Currency;
import ru.bitec.numbertostr.settings.PennyShowType;
import ru.bitec.numbertostr.settings.Language;

import java.math.BigDecimal;

public class JRMoneyToStrScriptlet extends JRDefaultScriptlet {

    public String moneyToStrRu(BigDecimal npSum) throws
    JRScriptletException{
        return MoneyToStr.spellout(
            npSum,
            Currency.RUR,
            Language.RUS,
            PennyShowType.TEXT
        );
    }

    public String moneyToStrByCurrencyAndLanguageAndPennyShowType(
        BigDecimal
        npSum,
        String currency,
        String language,
        String pennyShowType
    ) throws JRScriptletException {
        return MoneyToStr.spellout(
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            npSum,
            Currency.valueOf(currency),
            Language.valueOf(language),
            PennyShowType.valueOf(pennyShowType)
        );
    }
}
```

2. Добавляем новый скриплет в отчет \
    1. Зайдите в свойства отчета
       Для этого кликните правой кнопкой мыши по каталогу отчета
    2. Перейдите в раздел «Java build path»
    3. Перейдите на закладку «source»
    4. Добавьте каталог
       Нажмите «Add folder» и укажите каталок `src`
3. В свойствах сервлета укажите
    - Name = `JRMoneyToStrScriptlet`
    - Class = `JRMoneyToStrScriptlet`

После этого функции скриплета становятся доступными в expression editor в разделе Parameters

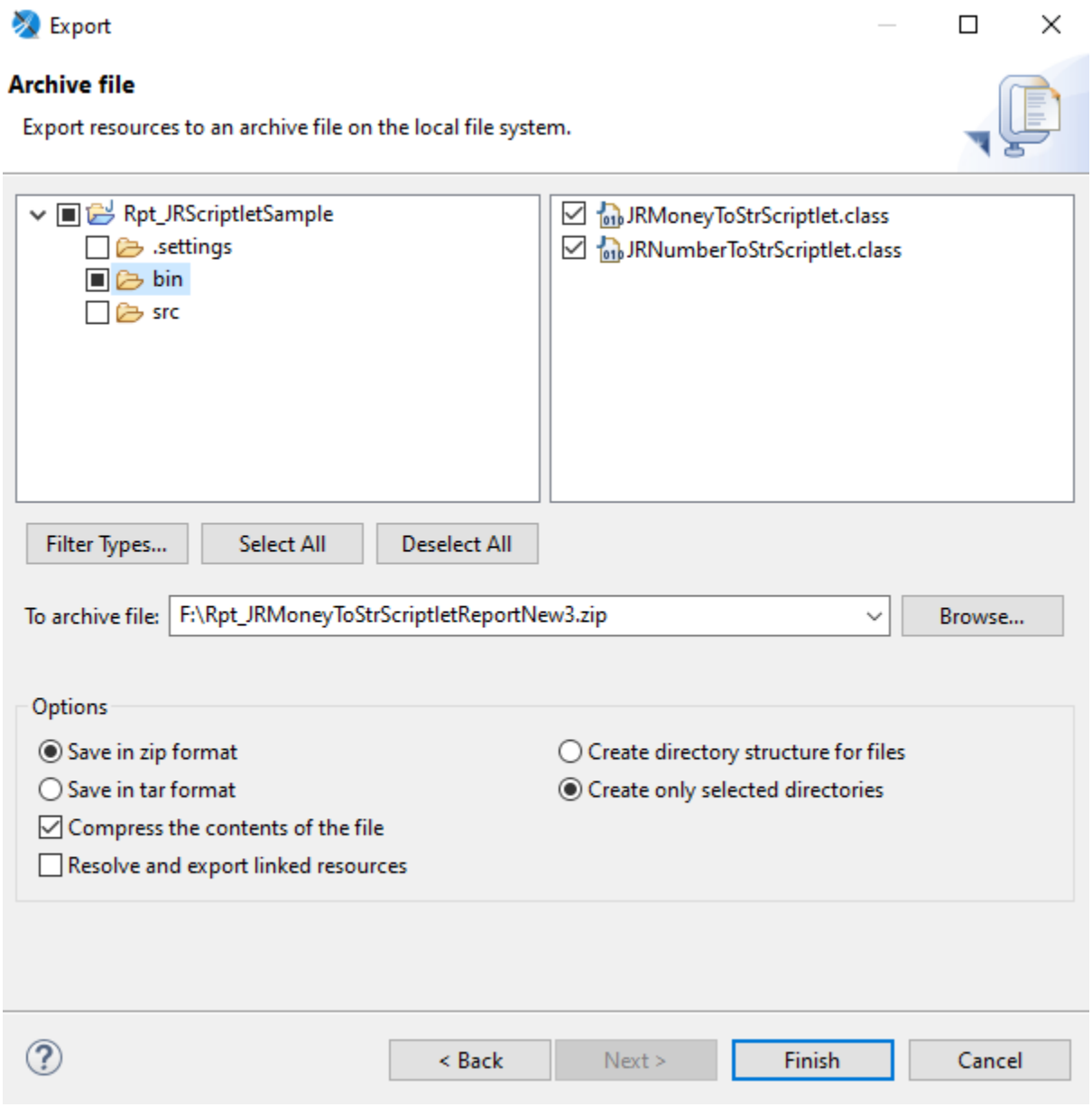
Пример выражения:

```
$P{JRMoneyToStrScriptlet_SCRIPTLET}.moneyToStrRu($F{nSum})
```

После завершения редактирования отчета необходимо собрать проекта.

При экспорте кроме `*.jrxml` файлов надо экспортировать `*.class` файлы скриптлетов.

Внимание: Именно файлы, а не папку bin, чтобы они оказались в корне архива, иначе jasper их не найдет при печати из глобал



20.17 Публикация отчётов в БД

Для обеспечения доступа сервера Global 3 к шаблону отчёта, zip-архив с шаблоном необходимо загрузить в базу данных через приложение «Настройка системы», пункт меню `Отчёты > Печатные формы > Версии печатной формы > Загрузить в базу`.

Шаблон JasperReports-отчёта может состоять из нескольких файлов (мастер-деталь, несколько страниц) и содержать различные ресурсы (картинки, скриптлеты). Поэтому, для сохранения в базу, шаблон приходится архивировать в zip-архив. `Файл main.jrxml` должен находиться в корне архива, иначе он не будет найден. Остальные файлы могут быть во вложенных каталогах, главное, что бы в шаблоне были указаны верные относительные ссылки.

Архив не должен содержать бинарных `*.jasper` файлов.

При построении, zip-архив будет разархивирован во временный каталог.

20.17.1 Ручная загрузка отчетов

1. В `Project Explorer` выберите папку содержащую отчеты печатной формы
2. В контекстном меню выполните `Export`
3. Выберите тип `Archive file`
4. Выберете фалы для экспорта

> **Внимание:** Допустимы только следующие расширения:
>
> - `jrxml`
> - `class`
> - `jar`

5. Укажите опцию `save in zip archive`
6. Укажите файл куда сохранить шаблон
7. Откройте приложение `Настройки системы`
8. Откройте печатные формы
   Пункт меню `Отчеты > Печатные формы`
9. Создайте версию печатной формы
10. Загрузите шаблон печатной формы

> **Внимание:** Убедитесь что файлы экспортируются в корень архива. Для этого при экспорте должна быть установлена опция `Create only selected directories`

20.17.2 Загрузка отчета на dataInstall

Загрузку отчета в базу можно добавить в dataInstall получив blob отчета с помощью метода `Rpt_ReportVersionApi().compress`.

Пример:

```
val blobData = Rpt_ReportVersionApi().compress(
    List((
        "subreport1.jrxml",
        getClass.getResourceAsStream("/reports/Mes_PlanReport/subreport1.jrxml")
    ),(
        "subreport2.jrxml",
        getClass.getResourceAsStream("/reports/Mes_PlanReport/subreport2.jrxml")
    ),(
        "order.jrxml",
        getClass.getResourceAsStream("/reports/Mes_PlanReport/order.jrxml")
    ),(
        "main.jrxml",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        getClass.getResourceAsStream("/reports/Mes_PlanReport/main.jrxml"))
    )
)
```

Полностью пример можно посмотреть в `ru.bitec.app.mes.Mes_DataInstallPkg#reportInstall`.

20.18 Отчет с содержанием

Документация на sourceforge

Возьмем Report Book, в котором 2 отчета:

- `TableOfContents`
- `Content`.

Для создания содержания с гиперссылками в начале документа

1. Укажите свойства отчета TableOfConents
   - если отчет стоит перед `Content` то `EvaluationTime` = `Report`
   - `net.sf.jasperreports.bookmarks.data.source.parameter` = `REPORT_DATA_SOURCE`
2. При необходимости установить галочку «Use Cache» Галочка находится на `Pages`
3. Настройте поле для `Anchor` в отчете `Content`
   1. Перейдите на поле которое хотите видеть в содержании
   2. Задайте свойства на закладке `Hyperlink`
      - «Anchor name expression» = `$F{sposition}` Можно указывать имя в виде `"Позиция: " + $F{sposition}`, но нужно учитывать, что если тут будет число – нужно привести его к `.toString`.
      - «Bookmark level» = `1`
4. Создайте поля в Dataset'е отчета TableOfContents
   - `level` тип `java.lang.Integer`
   - `pageIndex` тип `java.lang.Integer`
   - `label` тип `java.lang.String`
5. Для отображения имени гиперссылки
   1. Создайте или перейдите на соответствующий TextField в TableOfContents
   2. Задайте значением `$F{label}`
   3. Свяжите поле с уровнем anchor
      Укажите свойство «Appearance \ print when \ print when expression» = `$F{level}==1`
6. Для отображения номера страницы
   1. Создайте или перейдите на соответствующий TextField в TableOfContents
   2. Задайте значение `$V{PAGE_NUMBER} + $F{pageIndex} + 1`
   3. Задайте свойство «Evaluation Time» = `Auto`

4. Свяжите поле с уровнем anchor
   Укажите свойство «Appearance \ print when \ print when expression» = `$F{level}==1`

7. При необходимости можно сделать поля с именем анкора и номером страницы гиперссылками

   1. Откройте свойства поля
   2. Перейдите на закладку Hyperlink
   3. В разделе HyperLink задайте

      - «Link Target» = `Self`
      - «Link Type» = `LocalAnchor`
      - «Hyperlink Anchor Expression» = `$F{lable}`

Глава 21

Шаблоны отчетов

21.1 Принцип разметки

Шаблонов документов имеют одинаковую разметку, что позволяет, научившись размечать шаблоны одного типа, создавать отчёты других типов.

Примечание: Раздел не относится к специализированным отчетам, таким как Jasper Reports.

Рассмотрим основные положения.

Основным элементом разметки шаблона является

- `Тэг`
  Текст окружённый спецсимволами, управляющий формированием отчёта.

Все тэги имеют следующий вид: `[#{значение}#]`

Простейшим случаем является Тэг, возвращающий значение атрибута строки: `[#{имя атрибута}#]`.

С помощью таких тэгов обозначаются места, которые должны быть заполнены данными. Если в данных не будет найден атрибут или параметр с именем `{имя атрибута}`, текст будет заменён на пустоту.

Для печати в отчете набора данных предназначены переменные части, позволяющие выполнять запросы и выводить в отчёт данные с изменяющимся количеством строк.

Переменной частью называется область шаблона, ограниченная открывающим и закрывающим тэгами. Переменная часть будет выведена в отчёт столько раз, сколько записей находится в выборке переменной части.

Форматы тегов для печати данных из запроса:

- открывающий тэг

```
[#@{произвольное системное имя}={SQL запрос}#]
```

- закрывающий тэг

```
[#/{произвольное системное имя}#]
```

Переменная часть обязательно должна заканчиваться закрывающим тэгом, системное имя которого совпадает с системным именем открывающего тэга. Все системные имена открывающих тэгов начинаются с символа &, а все закрывающие тэги начинаются с символа /.

При выводе данных в отчет из запроса открывающий тэг переменной части состоит из произвольного уникального в рамках шаблона имени и SQL запроса. В этом случае для данной переменной части также будет создан набор данных с запросом указанным в тэге.

Переменные части могут быть вложенными друг в друга, образовывая связку мастер-деталь.

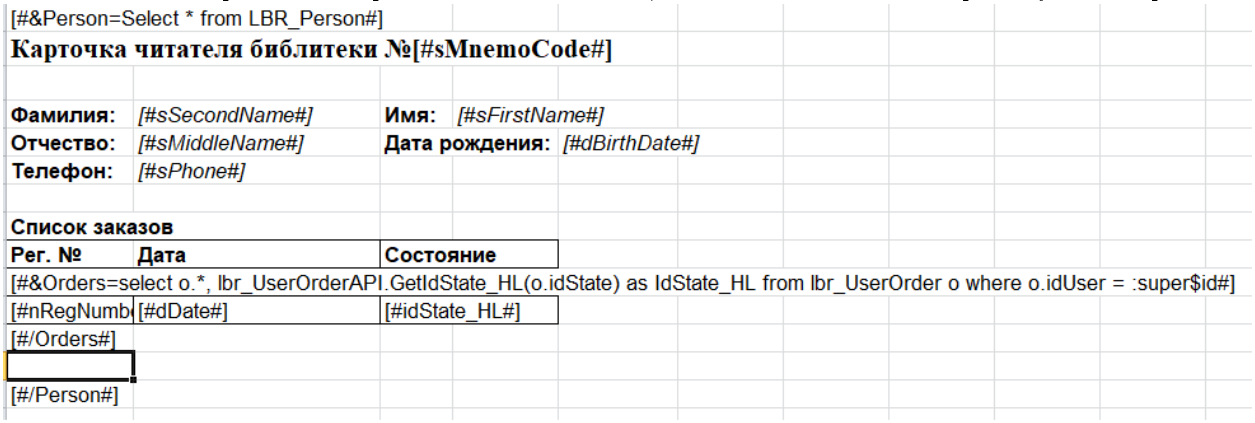
Если из вложенного уровня (уровень «деталь») необходимо получить значение атрибута из запроса верхнего уровня (уровень «мастер»), то следует использовать маркер следующего вида: [#super$sField#] где «sField» - имя атрибута из Sql-запроса верхнего уровня. При этом стоит отметить, что маркер [#sField#] тоже сработает, если атрибут «sField» присутствует только в запросе верхнего уровня. Если же он встречается в запросе нижнего уровня, то в печатной форме будет выведено значение из него.

Совет:

1. Располагайте открывающие и закрывающие тэги на отдельных строчках.
2. Выносите запросы переменных частей в выборки, если их длина больше 1000 символов.

21.2 Шаблон xlxs

Шаблоном xlxs является документ в формате `xlxs`, размеченный согласно общим правилам разметки шаблонов, описанным в предыдущем разделе.



Обратитесь к дистрибьютору документации за файлом `report_template/ReportExample.xlsx`

21.3 Шаблон docx

Шаблоном docx является документ в формате `docx`, размеченный согласно общим правилам разметки шаблонов. Если создать отчёт с шаблоном docx, скопировать в него разметку из отчёта xlsx, отчёт результат построения будет таким же, как для XLS. Отличия будут только в форматировании, данные будут теми же.

21.4 Шаблон TXT

Шаблон на основе текстового файла был реализован для быстрой печати на матричных принтерах, поскольку печать текста происходит на много быстрее, чем печать графики или векторных шрифтов. Пример:

```
[#&Person=Select * from LBR_Person#]
Карточка читателя библиотеки №[#sMnemoCode#]
Фамилия: [#sSecondName#] Имя: [#sFirstName#]
Отчество: [#sMiddleName#] Дата рождения: [#dBirthDate#]
Телефон: [#sPhone#]
Список заказов
Рег. №        Дата        Состояние
[#&Orders=select o.*,
lbr_UserOrderAPI.GetIdState_HL(o.idState) as IdState_HL
from lbr_UserOrder o
where o.idUser = :super$id#]
[#nRegNumber#][#dDate#][#idState_HL#]
[#/Orders#]
[#/Person#]
```

Внимание: Открывающий тег с запросом должен располагаться в одной строке!

Часть VI

Организация разработки

Глава 22

Основные понятия

Данная глава описывает основные понятия необходимые для понимания организации разработки

22.1 Решение

Дистрибутив решения это набор скомпилированных библиотек (`jar` файлов) устанавливаемых в сервер приложения. Решение предоставляет весь перечень прикладной бизнес логики необходимой для работы сервера приложения.

22.1.1 Системные требования

- Установленный сервер приложения GlobalFramework
- Доступ к базе данных PostgreSql

22.2 Дистрибутив сервера приложения

Дистрибутив сервера приложений представляет из себя архив содержащий набор файлов необходимый и достаточный для запуска `решений`.

22.2.1 Системные требования

- Java на базе OpenJDK 1.8

22.3 Модуль

Неделимый набор файлов для сборки решения. Набор модулей определяет доступную функциональность развернутую в сервере приложения.

Модуль содержит:

- `Классы`
  Описание атрибутов таблицы, и бизнес логика обработки строк.

  Подробнее смотри:

  - *Классы*
  - *Сервисы класса*
  - *Тип объекта*

- `Выборки`
  Выборка определяет правило получения, отображение данных и обеспечивает взаимодействие с пользователем.

  Подробнее смотри:

  - *Выборка*

- Поставочные данные
  Данные и скрипты для установки в БД при первом запуске решения.

- Скрипты миграции
  Скрипты комплексного обновления схемы базы данных и модуля при первом запуске новой версии решения

22.3.1 Зависимости

Модуль может содержать зависимости от другим модулей:

- Статическая зависимость
  При статической зависимости в программе дочернего модуля можно использовать весь код родительского модуля, как если бы этот код был в самом дочернем модуле. При этом невозможен параллельный выпуск родительского и дочернего модуля. Для исключения ошибок бинарной несовместимости необходимо выпустить сначала родительский модуль и на основе него дочерний. Либо оба модуля должны выпускаться одновременно в одной сборке.

- Динамическая зависимость
  При этом доступ к дочернему модулю возможен только через jexl скрипты. Jexl скрипты ограничивают возможности совместного использования кода, часто неявно требуя специализированных фасадов для взаимодействия. Однако это позволяет выпускать релизы модулей независимо друг от друга.

22.3.2 Проектный модуль

Модуль содержащий в себе конфигурацию решения, данный модуль содержит:

- Обработку прикладных событий
  Прикладные модули могут предоставлять набор событий на которые можно подписаться для расширения или склейки поставочной бизнес логики
- Конфигурацию проектных переопределений
  Можно переопределить контроллеры классов и выборок
- Проектную бизнес-логику
  Если нет специальной необходимости выделять отдельный модуль для бизнес-логики, ее можно расположить непосредственно в проектном модуле.

22.4 Репозиторий библиотек

Репозиторий библиотек хранит повторно используемые артефакты. Для корректной сборки решения необходимо предоставить доступ к следующим репозиториям:

- Репозиторий библиотек
- Репозиторий комплектов сборки
  Используется в случае если *проект* собирается на основе комплекта сборки

Примечание: Используются разные репозитории, так как это облегчает сброс локального кэша, что может быть полезно при использовании `snapshot` версий.

22.5 Комплект сборки

Набор одновременно выпускаемых артефактов, достаточный для сборки проекта. Базовый комплект сборки содержит:

- Сервер приложений
- Плагин сборки
  Используется для расширения стандартной функциональности sbt
- Базовые модули

Модули публикуются группами в комплекте сборки. Что существенно снижает нагрузку на программистов при разработки модуля. Так как позволяет не отслеживать бинарную совместимость. Это позволяет существенно увеличить допустимые изменения в рамках одной версии релиза или `snapshot` версии.

22.6 База данных

Для хранения транзакционных данных используется база данных postgreSQL

22.6.1 Системные требования

- postgreSQL 12 и выше
- *Расширения БД*

22.6.2 Конфигурация решения

Позволяет настраивать поведение решения без привлечения программистов.
В основном содержит:

- *Настройки типов*
- *Печатные формы*
- Настройки электронных подписей для пользователей
- Права доступа
- Универсальные характеристики
- Универсальные справочники

22.6.3 Создание базы данных для разработки

Программист может в любой момент создать себе локальную копию базы данных(*Начало работы*).

При необходимости распространять конфигурацию по локальным базам разработки, возможно:

- Использование полных релизов конфигурации
- Клонирование базы разработки с настроенной конфигурацией решения
  В простейшем случае достаточно снятия и установки дампа схемы `public`

22.6.4 Создание тестовой базы данных

Тестовая база может быть создана от рабочей базы. В простейшем случае достаточно снятия и установки дампа схемы `public`.

22.6.5 Шаблон ландшафта

- `common_dev`
  Центральный проект разработки. На нем разрабатываются базовые версии модулей. И выпускаются релизы коробочных решений.
- `project_dev`
  Проектная база разработки, осуществляется детальная настройка модулей и интеграций по ТЗ заказчика. Производится тестирование.
- `project_test`
  Тестирование решения на мощностях заказчика.

- `project_prod`
  Рабочий контур заказчика

22.7 Файловое хранилище

Используется для хранения прикрепленных файлов отдельно от БД. Это позволяет существенно снизить нагрузку на базу данных.

Доступ к файловым хранилищам настраивается в приложении: `Настройка системы \ Сущности > Файловые хранилища`

Глава 23

Проект

Директория на диске, хранящая файлы для сборки решения.

23.1 Структура проекта

- `module1`
  Прикладной модуль
- `module2`
- `moduleN`
- `projectModule`
  Проектный модуль
- `project`
  Мета информация по сборке
- `build`
  Каталог находится вне системы контроля версий, и содержит данные специфичные для локальной сборки.
  - lib
    Локальные jar библиотеки для запуска сервера, и утилиты прикладной разработки
  - config.yml
    Создается при необходимости локального переопределения конфигурации проекта
- `build.sbt`
  Декларация сборки для sbt
- `project.yaml`
  Конфигурация проекта

23.2 Конфигурация проекта

Конфигурация проекта описывает правила сборки прикладного проекта.

Конфигурация проекта располагается в проекте по адресу `application/project.yaml`

Конфигурация перечисляет перечень модулей используемых в сборке решения, а так же их источник. В случае необходимости источник проекта можно переключить, получая таким образом возможность отлаживать баг фиксы с текущего проекта.

Внимание: После отладки баг фикса в комплекте сборки необходимо выпустить пул реквест в ветку разработки модуля и дождаться нового релиза. В случае необходимости выпустить решение не дожидаясь релиза комплекта сборки. Необходимо переключится на режим сборки по исходному коду для всех дочерних модулей.

23.2.1 Источники модуля

23.2.2 local

Код модуля находится в системе контроля версий исходного проекта. Подключается по исходному коду.

23.2.3 git

Код модуля находится в отдельном `git` репозитории. Подключается по исходному коду.

Совет: Для массовой синхронизации модулей используйте утилиту `gsf-cli`

23.2.4 base

Модуль выпускается в базовом комплекте сборки. Подключается в виде jar артефакта.

23.2.5 buildKit

Модуль выпускается в прикладном комплекте сборки. Подключается в виде jar артефакта.

23.2.6 Пример конфигурации

```
#Комплект сборки, куда будет происходить публикация прикладных модулей
buildKit:
  name: ru.bitec.app
  version: 1.0.1

#Комплект сборки на основе которого работает проект
#Базовый комплект сборки обычно содержит сервер приложений и базовые модули
baseBuildKit:
  name: ru.bitec.base
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  version: 1.0.1

#Источник сервера приложений
#Используется утилитой gs3-cli для подготовки проекта к работе
applicationServer:
  source: "base://"

#Источник плагина для сборки
sbtPlugin:
  source: "base://"

modules:
  #Подключение модуля через jar артефакт из maven репозитория
  - gtkjpa:
      source: "base://"
  - gtk:
      source: "base://"
  - btk:
      source: "base://"
  #Подключение модуля через исходный код из удаленного репозитория
  - bts:
      source: "git://{host}/bts.git"
      branch: main
      isPublish: true
```

23.3 Конфигурация репозиториев

Для того чтобы обеспечить возможность собирать проект в разных локальных сетях. Настройка репозиториев вынесена в отдельный файл и может быть переопределена.

По умолчанию настройка репозиториев берется из файла: `application/project/repositories/default.yaml`

Пример настройки:

```
repositories:
  - name: maven-global
    url: "https://repo.global-system.ru/artifactory/libs-release"

publishRepository:
  name: app-global
  url: "https://repo.global-system.ru/artifactory/build-kit"
  #Запрашивать данные авторизации из менеджера учетных данных для данного репозитория
  isLoginRequired: true
```

23.3.1 Работа в изолированной сети

В случаи если требуется настроить проект таким образом чтобы он обращался только к репозиториям локальной сети.

Необходимо:

1. Настроить глобальную конфигурацию репозиториев `sbt`
   Для этого добавьте локальные репозитории в файл `~/.sbt/repositories`

```
[repositories]
  maven-proxy: http://host/repository/maven/
  buildkit: http://host/repository/buildkit/
```

2. Включите изоляцию репозиториев
   Это позволить обращаться только к локальным репозиториям, игнорирую репозитории вшитые в программный код. Для этого в файле проекта `application/.sbtopts` добавьте флаг

```
-Dsbt.override.build.repos=true
```

> **Примечание:** Включение данного флага не перекрывает настройки авторизации

Смотрите: Прокси репозитории

23.4 Интеграция в vcs

Исходный код проекта может располагаться в произвольной системе контроля версий. Модуль может быть расположен как в репозитории самого проекта так и в удаленном репозитории. В момент инициализации проекта код из удаленного репозитория добавляется в каталог основного проекта.

> **Совет:** Модули из удаленного репозитория следует добавить в игнор лист репозитория проекта.

23.4.1 Бранч-стратегии

После инициализации проекта можно пользоваться штатными средствами бранчивания используемой `vcs`.

> **Предупреждение:** В случае наличия изменений нарушающих обратную совместимость схемы базы данных желательно внести эти изменения в мастер ветку, для минимизации конфликтов в схеме базы дынных между разработчиками. Если изменения слишком обширны, стоит использовать локальную базу разработки для их тестирования.

23.5 Публикация решения

Публикация решения происходит средствами CI, например jenkins.

Команда sbt для публикации решения в локальный каталог:

```
sbt publishSolution
```

23.6 Публикация комплекта сборки

Публикация комплекта сборки происходит средствами CI

Команда sbt для публикации решения в репозиторий:

```
sbt publish
```

Глава 24

Релизы

Релиз представляет из себя снимок файлов определенной версии.

24.1 Решение

Релиз решения содержит перечень модулей и их версии.

24.2 Конфигурация

Релиз конфигурации содержит данные и скрипты обновления `конфигурации решения`. База выпуска релизов выбирается по согласованию с заказчиком.

24.3 Сервер приложения

Содержит в себе ядровой функционал необходимый для запуска решения. Использует семантическое версионирование.

24.4 Комплект сборки

Содержит скомпилированные прикладные модули. Версия формируется по следующему правилу:

- `MAJOR`
  Модули должны содержать одну мажор версию.
- `MINOR`
  Перещелкивается при любом изменении минорной версии модуля в составе комплекта.

- PATCH
  Перещелкивается при любом изменении патч версии модуля в составе комплекта.

24.5 Snapshot версия

Используется для распространения артефактов из ночной сборки или внеплановой сборки. Артефакты snapshot версии могут повторно закачиваться в репозиторий.

Для обновления Snapshot версий разработчику необходимо выполнить следующие команды sbt

```
sbt clean
sbt update
sbt updateClassifiers
sbt publishDevDependencies
```

24.6 Релиз прикладного модуля

Исходный код, согласованный на момент выпуска релиза, а также скрипты необходимые для установки обновления модуля в базе даннхы.

24.6.1 Версия модуля

Версия модуля задается в формате: МажорнаяВерсия.МинорнаяВерсия.Релиз.Билд

24.6.2 Типы релиза

- Мажорный – Крупные изменения, которые могут ломать обратную совместимость. Так же в этом типе релиза происходит очистка миграционных скриптов предыдущей версии. Устанавливается только на предыдущую мажорную версию.
- Минорный – Крупные изменения, которые могут ломать обратную совместимость.
- Релиз – изменения, требующие выполнения sql-скриптов миграции.
- Билд – сборка, не требующая выполнения sql-скриптов миграции. Разрешается выпускать, если нет невыпущенных sql-скриптов миграции.

Состав релиза

Релиз состоит из:

- Исходного кода
- Первичных данных, указанных в `odm.xml` и `pkg.xml`
- Первичных и миграционных данных, указанных в пакетах миграции (`<Имя модуля>_MigratePkg`)
- Sql-скриптов миграции

Схема базы данных модуля

Схема базы данных, прикладного модуля создается в базе данных при установке модуля.

Объекты схемы, которые были сформированы ранее, и были удалены из исходного кода приложения, не удаляются из БД автоматически. Этим занимается администратор используя инструмент «Корзина удаленных объектов схемы».

Доменная схема

Доменная схема, представляет из себя таблицы, индексы и последовательности создаваемые по описанию классов в `odm`-файлах.

Расширенная схема

Расширенная схема, может быть описана в `odm`-, `pkg`-файлах, в тэге `dbSchema`.

Поставочные данные модуля

Поставочные данные прикладного модуля устанавливаются в базу данных при установки модуля.

Миграционные данные

Миграционные данные делятся на 2 типа:

- `Sql-скрипты` - используются для работы с изменением схемы напрямую в БД, без `eclipse-link`. Хранятся в структуре модуля в ветке `resource/migration/dbSchema`
- `Миграционные задачи` – скрипты, которые написаны с использованием `Api` и `Pkg`. Хранятся в миграционных пакетах модуля.

24.6.3 Миграционные пакеты

Общие сведения

В каждом модуле возможно создание миграционного пакета, который должен иметь имя `<Имя модуля>_MigratePkg`, и находится в рутовом `scala`-каталоге модуля. Например, полное имя пакета модуля `btk` будет иметь вид: `ru.bitec.app.btk.Btk_MigratePkg`

Этот пакет используется для написания кода по установке первичных данных или выполнения миграции данных, которое можно выполнить с помощью `scala`-кода.

Миграционные пакеты требуется наследовать от `MigratePkg`, переопределяя в них методы `onUpMigrate` и `onDownMigrate`, в этих методах требуется создавать соответствующие задачи миграции через методы `upTask` и `downTask`. Вызов миграционных пакетов осуществляется после выполнения команды установки первичных данных (`init data`).

Порядок вызова

Вызов миграционных пакетов осуществляется в 2 прохода:

- первый раз осуществляется вызов метода `upMigrate`, при обходе модулей от низкоуровниевых (например, `gtk`), к высокоуровниевым (например, `stk`). В этом методе можно писать логику по созданию новых данных
- при втором проходе вызывается метод `downMigrate`, при проходе модулей от высокоуровниевых к низкоуровниевым. В этом методе можно удалять данные.

Миграционные задачи

Задачи имеют имя и версию. Задача выполняется только один раз, для того чтобы выполнить задачу повторно, требуется изменить ее версию или имя. Задачи делятся на 2 вида:

- `upTask` - используется в `upMigrate`
- `downTask` – используется в `downMigrate`

UpTask

Задачи метода `upMigrate`. Позволяют указывать зависимости от других задач, от скриптов `dbData` из `odm.xml` и `pkg.xml`. Могут содержать секцию, которая будет выполнена, если задача уже выполнялась на целевой базе. А также могут быть анонимными и не содержать тело.

Простая задача

Содержит только тело. Пример объявления задачи:

```
upTask("taskWOElse", 1){
  //Код задачи
}
```

Задача с секцией orElse

Если задача выполняет установку каких-то данных, например, типа объекта, то можно указать ей секцию `orElse`, которая будет вызвана, если тело задачи уже было выполнено, например, в этой секции можно вызвать метод поиска типа объекта по его системному имени. Пример объявления такой задачи:

```
upTask("installType", 1){
  Btk_AuditActionApi().register("someName", "someCaption", "SomeDesc")
}.orElse{
  Btk_AuditActionApi().findByMnemoCode("someName")
}
```

Анонимная задача

Задача без тела, выполняется для группировки зависимостей. Пример объявления:

```
upTask("dependencyGroup")
  .dependsOnDbData("Wf_Doc", "dataInstall", 25)
  .dependsOn(installStkTypeTask())
```

Зависимости

Для каждой задачи можно указать зависимость как от другой задачи, так и от данных, устанавливаемых секцией `dbData`.

Пример объявления зависимостей.

```
val someOtherTask = upTask("someOtherTask", 1){
  //Код задачи
}

upTask("taskWOElse", 1){
  //Код задачи
}
  .dependsOnDbData("Wf_Doc", "dataInstall", 25) //зависимость от dbData
  .dependsOn(someOtherTask) // зависимость от другой задачи
```

Зависимости между модулями.

Для того, чтобы сделать зависимость задачи одного модуля (`module1`) от задачи другого (`module2`) требуется:

1. В `Module1_MigratePkg` задачу вынести в отдельный метод и вызывать его из `upMigrate`.

   ```
   def someTask(): UpTask[NLong] = {
     upTask("someTask", 1){
       1.nl
     }
   }

   override def onUpMigrate(): Unit = {
     someTask()
   }
   ```

2. В `Module2_MigratePkg` для задачи указать зависимость от задачи из `module1`.

   ```
   override def onUpMigrate(): Unit = {
     val someTask = Module1_MigratePkg().someTask()
     upTask("someDepTask", 1){
       //Получаем значение, которое вернула задача, от которой зависим
       val someTaskValue = someTask.get()
     }
       .dependsOn(someTask)
   }
   ```

DownTask

Задачи метода `downMigrate`. Содержат только тело задачи. Пример объявления задачи:

```
downTask("Task_Down", 1) {
  //код задачи
}
```

24.6.4 Sql-скрипты миграции

Общие сведения

Используются для внесения изменений в схему БД, до инициализации `eclipse-link`. Позволяют гарантированно выполняться в схеме, идентичной той, что была на момент выпуска релиза.

Описание файла скриптов

Миграционные скрипты размещаются внутри файла `resource/migration/trunk.script.jexl`. А при выпуске релиза копируются в подкаталог `resource/migration/dbSchema`.

Скрипты пишутся в формате `jexl`-скрипта, и могут быть выполнены до обновления схемы или после.

Исполняются внутри специального `jexl`-контекста, который обладает функциями:

- `before` – скрипт будет выполнен до обновления схемы
- `after` – скрипт будет выполнен после обновления схемы.

Если при выполнении скрипта миграции будет получена ошибка, то процесс установки релиза и обновления базы будет прерван. Для того, чтобы выполнить какой-либо скрипт с игнорированием ошибок нужно добавить к нему директиву `force()`.

Пример файла скриптов

```
//выполнение скрипта после обновления схемы
after("scriptName1", `
  delete from btk_class where 1=2 asd
`);

//выполнение скрипта перед обновлением схемы
before("scriptName2", `
  delete from btk_class where 1=2
`);

//выполнение скрипта с игнорированием ошибок.
//Если в таком скрипте произойдет ошибка, то обновление не будет остановлено

after("scriptName3", `
  delete from btk_class where 1=2
`).force();
```

24.7 Инструкции

24.7.1 Создание задачи миграции

1. Зайдите в IDE
2. В нужном модуле в `scala`-ветки найдите или создайте пакет с именем `<Имя модуля>_MigratePkg`
3. Создайте задачу миграции, используя методы `upTask` или `downTask`

24.7.2 Создание sql-скрипта миграции

1. Зайдите в IDE
2. В нужном модуле в ветке `resource/migration` найдите или создайте файл `trunk.script.jexl`
3. Добавьте в него скрипт используя методе `before` или `after`

24.7.3 Выпуск новой версии

Выпуск билда

1. Запустите конфигуратор для базы разработки
2. Зайдите в модули
3. Выполните операцию `Выпустить билд`

Выпуск релиза

1. Отправьте изменения в систему контроля версий
2. Запустите конфигуратор для базы разработки
3. Зайдите в модули
4. Выполните операцию `Выпустить релиз`

Выпуск минорной версии

1. Отправьте изменения в систему контроля версий
2. Запустите конфигуратор для базы разработки
3. Зайдите в модули
4. Выполните операцию `Выпустить минорную версию`

Выпуск мажорной версии

1. Отправьте изменения в систему контроля версий
2. Запустите конфигуратор для базы разработки
3. Зайдите в модули
4. Выполните операцию `Выпустить мажорную версию`

24.7.4 Установка релиза

1. Сервер приложений Global должен быть запущен.
2. Подключитесь к серверу SSH-клиентом. И выполните скрипт.

```
attach db {dbAlias} as sys
upgrade
detach
exit
```

`dbAlias` – алиас базы данных. Если не указан, будет использоваться значение по умолчанию из `global3.config.xml`

Обновление будет выполняться под суперпользователем `SYS`

Алгоритм работы

1. Shh-команда `attach db as sys` переключает консоль в режим работы с обновлениями.
2. Команда `upgrade` запустит процесс обновления, состоящий из следующих действий:
    1. Формирование очереди выполнения `sql`-скриптов миграции.
    2. Последовательное обновление схемы с промежуточными выполнениями скриптов
    3. Установка первичных данных (`dbData`)
    4. Выполнение задач миграции
3. Отключение от базы
4. Выход из SSH

Сброс состояния обновления

Если требуется откатить информацию об установленных первичных данных или миграционных задачах, то используется `ssh`-команда `reset upgrade`

Если в нее не передан идентификатор обновления, то будет сброшено последнее выполненное обновление.

```
attach db {dbAlias} as sys
reset upgrade
detach
exit
```

или

```
attach db {dbAlias} as sys
reset upgrade 10202
detach
exit
```

Глава 25

Работа с Git

Git — это система контроля версий. Она хранит текущую версию кода, а также все изменения, которые вносили разработчики.

25.1 Основные понятия

25.1.1 Сущности

Репозиторий: Это место, где хранится проект. Код прикладных модулей хранится в удалённом репозитории на GitLab. У каждого разработчика на ПК есть локальный репозиторий - копия удалённого. Разработчик вносит изменения в код на своём ПК и, выполнив задачу, отправляет изменения на удалённый репозиторий. Остальные программисты периодически обновляют свои локальные репозитории, чтобы иметь последнюю версию кода. Это можно сделать с помощью команды pull или с помощью `gsf-cli`.

Коммит (commit): Это сохранение изменений в репозитории. Коммит - это конкретное осмысленное изменение: добавлен класс, исправлена ошибка, переделана функция и т.д. Git позволяет восстановить исходный код проекта на момент любого коммита. О них можно думать как о «точках сохранения» исходного кода.

Ветка (branch): Ветка представляет собой цепь коммитов. От основной ветки можно отводить новые, создавая отдельные линии разработки. Ветки позволяют работать над новыми функциями и исправлениями, не затрагивая основную версию проекта. Когда новый функционал готов в отдельной ветке, можно применить проделанные изменения к основной ветке. Эта операция называется *merge*.

Запрос на слияние (merge request): Когда в отельной ветке готов новый функционал, он не включается в основную ветку автоматически. Вместо этого автор изменений создаёт запрос на слияние, который должен быть одобрен для того, чтобы ветки можно было объединить. Этот процесс проверки кода называется **код-ревью**.

Тег (tag): Тег используется для маркировки определённых точек в истории коммитов. Обычно теги применяются для обозначения релизов (например, 1.0.161). Все версии модулей компилируются и хранятся в облаке - такие размещённые в облаке сборки называются **артефактами**. В итоге для того,

чтобы поменять версию модуля, которую импортирует проект, достаточно поменять номер версии в конфиге.

Ишью (issue): предложение на внутреннем форуме проекта в гитлабе. Там можно указывать на ошибки, предлагать и обсуждать идеи. В нашей компании ошибки указывают не там - вместо этого создают ДП в системе Support, не имеющей отношения к GitLab.

25.1.2 Роли

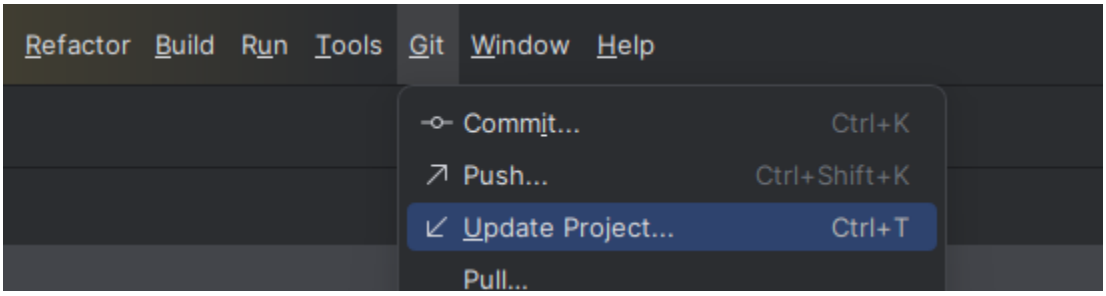
У аккаунтов в GitLab могут быть разные роли в разных репозиториях. У разработчика один аккаунт на весь GitLab, при этом он может быть мейнтейнером в одном модуле и рядовым разработчиком - в другом.

Девелопер (developer): программист, работающий над проектом. Может создавать новые коммиты, ветки и merge request'ы.

Мейнтейнер (maintainer): Мейнтейнер отвечает за управление репозиторием. Он проводит код-ревью, принимает или отклоняет запросы на слияние.

25.2 Основные команды Git

В IDEA во вкладке Git или VCS есть три основные команды - Update (она же Pull), Commit и Push



25.2.1 Обновление проекта

Используется для того, чтобы обновить локальный репозиторий последними изменениями. Применяются коммиты, созданные другими разработчиками в той же ветке.

25.2.2 Коммит

Разработчик может редактировать код у себя на ПК как угодно - изменения не будут зафиксированы, пока он это не сделает с помощью команды commit. Её смысл - сохранить рабочую версию кода, в которой реализован новый функционал или исправлена ошибка. Когда разработчик доволен написанным кодом, он нажимает commit и отмечает изменённые файлы. Очень важно дать коммиту понятное имя, потому что с ним будут иметь дело другие программисты.

Названия коммитов

Общий формат коммита - `Label(Scopes): Subject [#issue Issue]`

`Label` - это одно из:

- `feat`, для добавлений нового функционала
- `fix`, для исправлений ошибок
- `hotfix` для быстрого исправления недавней ошибки
- `test` для новых тестов
- `refactor` для улучшений кода без изменений функциональности
- `docs` для правок документации
- `style` для форматирования, приведения к CodeStyle
- `chore` для случаев, когда не подходят остальные ярлыки.

`Scopes` - модул(и), в которые вносят изменения.

`Subject` - короткое описание изменений. Оно начинается с заглавной буквы и и совершенного глагола неопределённой формы - Изменить, Исправить, Реализовать.

`Issue` - номер задачи, при наличии. Обычно это номер ДП.

Пример названия коммита:

```
fix(rpl): Убрать поле "Источник" у входящих сообщений интеграции с превышенным числом
ошибок #issue 182831T
```

Новосозданный коммит существует только в локальном репозитории. Чтобы отправить изменения на GitLab, используется команда Push.

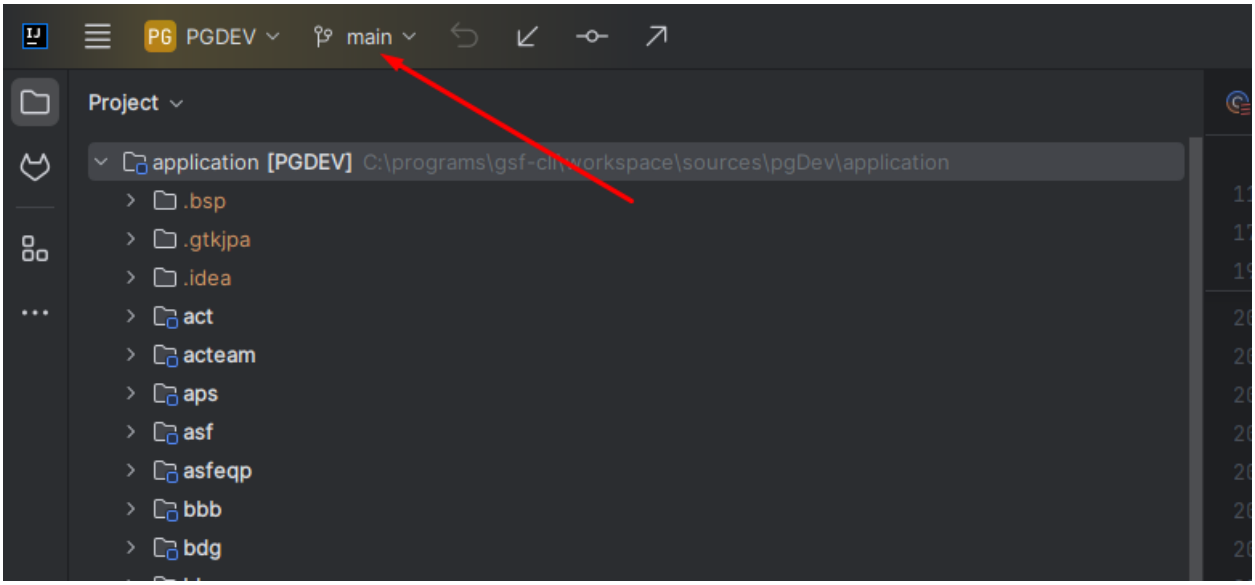
25.2.3 Push

Эта команда отправляет коммиты на удалённый репозиторий.

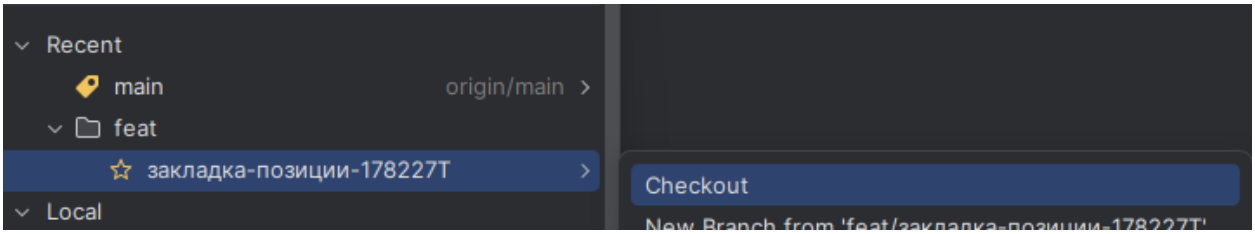
25.3 Ветки

Для каждой новой функции (feature) приложения создают отдельную ветку изменений, чтобы функции можно было разрабатывать независимо друг от друга.

В IDEA меню управления ветками расположено справа от выбора проекта:



Переключаться на другую ветку можно с помощью операции checkout. При этом код приложения меняется на версию в ветке. Если в ветке, с которой переключаются, были незакоммиченные изменения, они пропадут; нужно их закоммитить или как-то ещё сохранить.



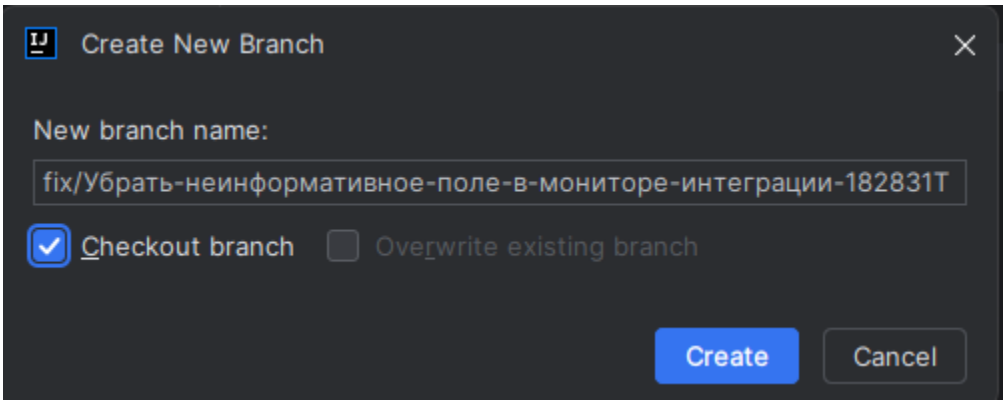
25.3.1 Создание веток

Новую ветку можно создать в меню управления ветками, с помощью New Branch. Обычно новая ветка должна отходить от основной ветки - main, поэтому перед созданием новой нужно переключиться на main.

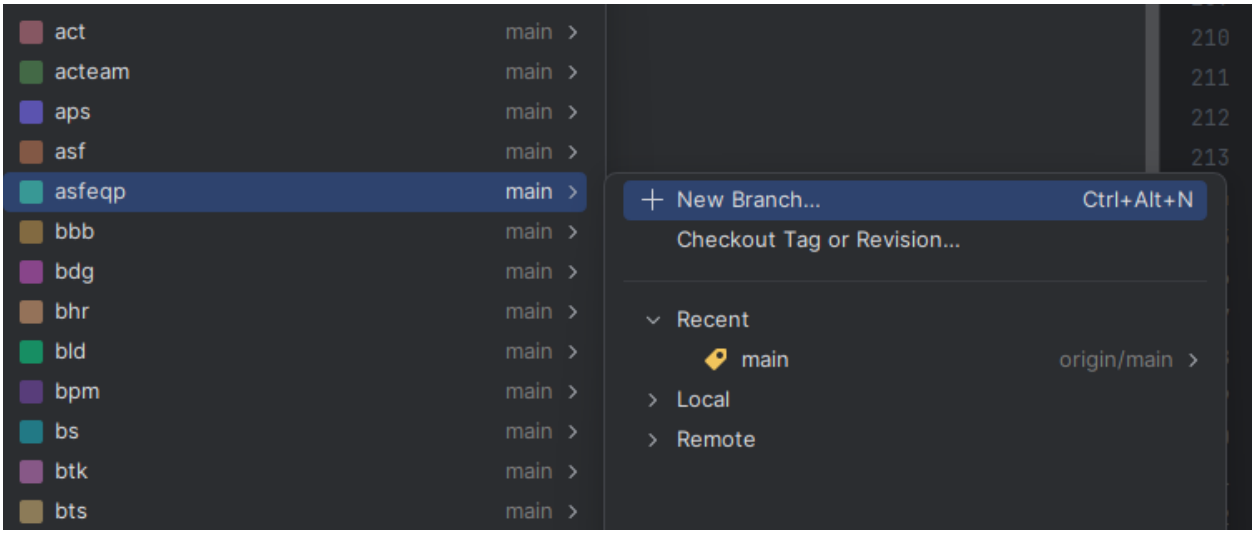
25.3.2 Названия веток

Общая форма названия ветки - `Label/Scope [Issue]`.

`Label`, `Scope` и `Issue` - как в названиях коммитов. Например,



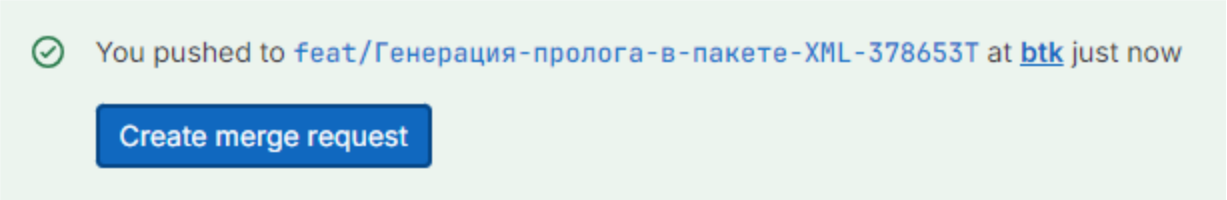
Каждый модуль - это отдельный git репозиторий, ветку нужно создавать именно в том модуле, который будут править:



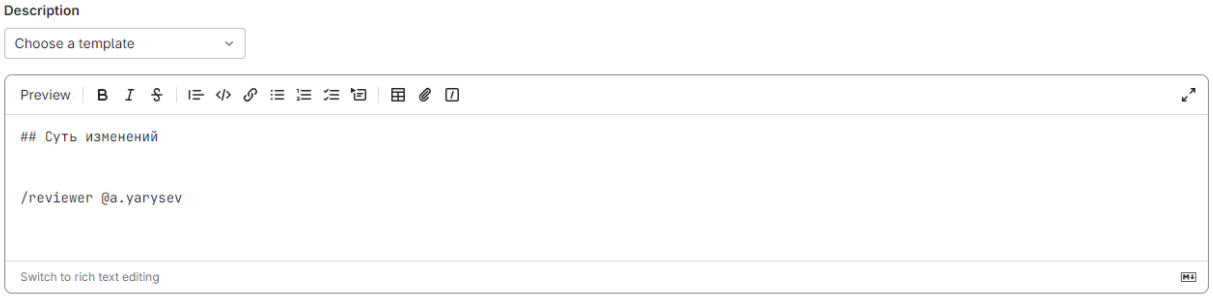
25.4 Merge request

Когда функция готова в своей отдельной ветке удалённого репозитория, создают запрос на слияние веток – merge request. Ответственный за модуль разработчик с ролью мейнтейнера изучает предложенные изменения, комментирует их и указывает на ошибки и недоработки. Создатель реквеста продолжает делать коммиты в ту же ветку, пока проблемы не будут решены. После этого происходит слияние веток (*merge*), и изменения попадают в главную ветку main.

Merge request'ы можно делать из IDEA, но удобнее через GitLab. После пуша в новую ветку GitLab автоматически предлагает создать merge request на главной странице модуля:



Можно указать ревьюера - пользователя, без чьего подтверждения нельзя будет сделать merge. Иногда он проставляется автоматически, как /reviewer на скрине. Могут быть настроены несколько шаблонов (template) merge request'ов.



В общем случае назначать ревьюера необязательно - мейнтейнер и так видит все мёрдж реквесты в модуле.

Ревьюер либо одобрит (Approve) merge request, либо укажет на ошибки, которые нужно исправить. Нужно исправлять ошибки, коммитить и пушить в ту же ветку, пока реквест не получит approval. После этого можно в меню реквеста жать merge.

25.5 Порядок выполнения ДП

- Создать новую ветку под ДП или перейти в существующую
- Написать код
- commit с указанием сути задачи и номером ДП
- push
- merge request

Часть VII

Дополнительно

Глава 26

Локализация приложений

Локализация приложений осуществляется средствами Java Resource Bundle и специальными тэгами, внедряемыми в код и метаданные приложений.

26.1 Словари

Для хранения строк на разных языках используются словари. Словарём является пакет ресурсов (Resource Bundle), содержащий properties-файлы со строками `{ключ}={значение}`:

- `{имя_словаря}.properties`
  Словарь по умолчанию
- `{имя_словаря}_{язык}.properties`
  Словарь для указанного языка.
- `{имя_словаря}_{язык}_{локаль}.properties`
  Словарь для указанного языка с уточнением локали.

26.1.1 Создание словаря

Новый словарь создается через контекстное меню от каталога с ресурсами.

1. В idea перейдите в каталог в котором хотите создать словарь
2. Вызовите на каталоге контекстное меню
3. Выполните пункт `New > Resource Bundle`

Имя словаря должно совпадать с именем сущности или пакета.

В создавшиеся файлы, добавьте локализованные строки в формате `{ключ}={значение}`

```
Caption=Валюта
ID=Идентификатор
MultiLineName=Многострочный \
текст
```

Обратите внимание, что при создании многострочной ресурсной строки, каждую строку необходимо заканчивать символом \.

26.2 Использование локализованных строк

Локализация строк доступна в:

- Scala-коде `*Dvi/Avi` – классов
- Scala-коде `*Dpi/Api` – классов
- Scala-коде `*_Pkg` – классов
- Аннотациях `@Oper` , `@MainSelection`
- `*.avm.xml` – файлах.
  Для свойств, хранящих «наименование»

Для получения значения локализованной строки из словаря, имя которого совпадает с текущей сущностью, необходимо указать ключ локализованной строки, обрамлённый тэгами: `[#Ключ_строки]`

Если необходимо получить значение строки из произвольного словаря, необходимо указать полный путь к словарю: `[#ru/bitec/app/{модуль}/Словарь.Ключ_строки]`

26.2.1 Локализованные строки в Scala-коде

Для подстановки локализованного значения строки в Scala-код используется макрос `ls"#Ключ_строки"` или `ls"""#Ключ_строки"""`.

```
def foo(): Unit = {
    showMessage(ls"[#MyLocalizedMessage]")
}
```

26.2.2 Локализованные строки в аннотациях

```
@MainSelection(caption = "[#Gs3_StockMainMenu.caption]")
object Gs3_StockMainMenuAvi extends ProjectApplicationAvi {}
```

26.2.3 Локализованные строки в *.avm.xml

```
<representation caption="[#caption]">
        <attributes>
                <attr name="sSystemName"
              caption="[#sSystemName]"
                          isVisible="true"/>
    <attributes/>
<representation/>
```

Глава 27

Логирование

При необходимости логирования в базу данных, необходимо обеспечить:

- независимость от основной ветки изменения данных
- высокую отказоустойчивость
- низкую нагрузку на сервер/базу данных

Для реализации такого логирования существует специальный инструмент – логирующая транзакция.

Данный инструмент позволяющий выделить запись логов в отдельную сессию. Для уменьшения количества обращений к базе производите сброс данных из логирующей транзакции в базу не чаще, чем раз в 1000 update/insert.

Пример:

```
val logTransaction = new LogTransaction() {
  override protected def run(): Unit = {
    Rpl_IntImportLogApi().setsError(Rpl_IntImportLogApi().load(idp),value)
    commitByInterval()
  }
}

logTransaction.execute()
```

Более подробно примеры использования можно посмотреть в модуле интеграции Rpl `ru.bitec.app.rpl.channel.intg.in.Rpl_IntImportLogApi` – методы с постфиксом `LT`.

При выполнении `.execute()` сервер производит попытку найти логирующую сессию, если она уже была инициирована. Если не была – выделяется новая. Внутри блока `LogTransaction() implicit session` является логирующая сессия и все действия производимые с использованием `logSession` относятся к ней.

В `LogSession` происходит `commit`:

- При освобождении `LogSession`
  `LogSession` освобождается на `endWork`.

Смотрите также методы основной сессии:

- `beginWork`
- `endWork`

Глава 28

Проектные расширения

28.1 Точка расширения

Аббревиатуры:

- `Ept` - Точка расширения
- `Ext` - Расширения

Точка расширения (ExtensionPoint) – специфический пакет, методы которого объявлены таким образом, что к ним можно подключить дополнительный исполняемый код из других пакетов (расширений/Extension). Код расширений будет выполнен при вызове метода точки расширения.

```
class Btk_ExampleEpt extends ExtensionPoint {
  val doSomeThing = declFunc("doSomeThing") {
    (sf: SuperFunc[Int], a: Int, b: Int) =>
      val superVal = sf().getOrElse(0)
      a + b + superVal
  }
}
object Btk_ExampleEpt extends PkgFactory[Btk_ExampleEpt]
```

Пример представляет объявление «Точки расширения» с методом `doSomeThing`

Правила наименования для точек расширения, которые используются для дополнения функционала сеттеров, вставки. удаления – `Xxx_Ept`, где `Xxx` – имя модуля, для которого точка расширения создается (например `Bs_Ept`, `Btk_Ept`).

Расширение (Extension) – специфический пакет, методы которого подключаются к методам точки расширения, и выполняются при вызове соответствующего метода точки расширения.

К методу точки расширения может быть подключено произвольное число методов расширений.

При первом обращении к классу точки расширения, производится поиск и сканирование файлов `META-INF/extensions.xml`. Расширения, у которых свойство `targetEpt` совпадает с именем точки расширения, подключаются к точке расширения.

```
class Gs3_Btk_ExampleExt extends Extension {

subFunc("doSomeThing") {
  (sf: SuperFunc[Int], a: Int, b: Int) =>
  val superVal = sf(a - 2, b).getOrElse(0)
  superVal + 2
}
}
```

Пример представляет объявление метода расширения. На вход метод получает ссылку на метод расширения, находящийся ниже в очереди вызовов.

Внимание: Если не вызвать `sf()`, методы расширений, находящиеся ниже в очереди не будут вызваны.

Правила наименования для расширений, которые используются для дополнения функционала других модулей для сеттеров, вставки. удаления – `Xxx_YyyExt`, где `Xxx` – имя модуля расширения, `Yyy` – имя модуля, в котором находится точка расширения (например `Mct_BsExt`).

28.1.1 Конфигурирование расширений

Конфигурация расширений находится в файле `META-INF/extensions.xml`.

Структура фала:

```
<?xml version="1.0" encoding="UTF-8"?>
<exts>
        <ext class="ru.bitec.app.btk.ext.Btk_ExceptionHandlerExt"
             targetEpt="ExceptionHandlerEpt"/>
</exts>
```

28.1.2 Порядок вызовов

По умолчанию, метод точки расширения получает порядковый номер «0», а все методы расширений – номер «10». В результате, в первую очередь буду вызваны все методы расширений, в произвольном порядке, в конце будет вызван код точки расширения.

Для изменения порядка вызовов необходимо задать порядковые номера методам расширений.

```
subFunc("doSomeThing", 1) {
  (sf: SuperFunc[Int], a: Int, b: Int) =>

}
```

28.2 Проектное перекрытие кода Api, Avi, Lib, Pkg

Для перекрытия кода Api, Avi, Lib, Pkg необходимо:

1. Создать свой класс-наследник от перекрываемого класса.
2. Зарегистрировать перекрытие класса в файле `META-INF/overrides.xml`

28.2.1 Пример перекрытия Avi-класса

Код исходного Avi-класса `Gs3_RootTestAvi.scala`

```
object Gs3_RootTestAvi extends Gs3_RootTestAvi

@AvmFile(name = "Gs3_RootTest.avm.xml")
class Gs3_RootTestAvi extends Gs3_RootTestDvi {
  override def list(): List = {
    new List {
    override def meta = this
    }
  }
}
```

Код перекрытия Avi-класса `Gs3_RootTestOverrideAvi.scala`

```
object Gs3_RootTestOverrideAvi extends Gs3_RootTestOverrideAvi

@AvmFile(name = "Gs3_RootTestOverride.avm.xml")
class Gs3_RootTestOverrideAvi extends Gs3_RootTestAvi {
  override def list(): List = {
    new List {
    override def meta = this
    }
  }
  trait List extends super.List {

  }
}
```

Аннотация `@AvmFile` может быть не указана, в этом случае будет использоваться аннотация из класса-предка.

Текст META-INF/overrides.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<overrides>
  <replace-with
      class="ru.bitec.app.gs3.test.Gs3_RootTestOverrideAvi">
    <when-type-is class="ru.bitec.app.gs3.test.Gs3_RootTestAvi"/>
  </replace-with>
</overrides>
```

28.2.2 Пример перекрытия Api

Код перекрытия

```
class Gs3_RootTestOverrideApi extends Gs3_RootTestApi {

}

object Gs3_RootTestOverrideApi extends ApiFactory[
  java.lang.Long,
  Gs3_RootTestAro,
  Gs3_RootTestOverrideApi
]
```

Текст META-INF/overrides.xml

```
<replace-with
    class="ru.bitec.app.gs3.test.collections.Gs3_RootTestOverrideApi">
  <when-type-is
  class="ru.bitec.app.gs3.test.collections.Gs3_RootTestApi"/>
</replace-with>
```

28.2.3 Пример перекрытия Pkg

Код перекрытия

```
class Btk_OverridePkg extends Btk_Pkg {

}

object Btk_OverridePkg extends PkgFactory[Btk_OverridePkg]
```

Текст `META-INF/overrides.xml`

```
<replace-with class="ru.bitec.app.gs3.Btk_OverridePkg">
  <when-type-is class="ru.bitec.app.btk.Btk_Pkg"/>
</replace-with>
```

28.2.4 Пример перекрытия Lib

Код перекрытия

```
class Btk_OverrideLib extends Btk_Lib {

}
object Btk_OverrideLib extends AviLibFactory[Btk_Lib]
```

Текст `META-INF/overrides.xml`

```
<replace-with class="ru.bitec.app.gs3.Btk_OverrideLib">
  <when-type-is class="ru.bitec.app.btk.Btk_Lib"/>
</replace-with>
```

28.2.5 Запрет перекрытия

Для запрета проектного перекрытия кода Avi-класса, необходимо отметить класс аннотацией @Final.

```
@Final
class Gs3_RootTestAvi extends Gs3_RootTestDvi {}
```

Для запрета проектного перекрытия кода Api,Lib,Pkg классов, необходимо отметить класс аннотацией `@Final` или ключевым словом `final`.

```
@Final
class Gs3_RootTestApi extends Gs3_RootTestDpi[
  Gs3_RootTestAro,
  Gs3_RootTestApi,
  Gs3_RootTestAta
] {}
```

или

```
final class Gs3_RootTestApi extends
Gs3_RootTestDpi[
  Gs3_RootTestAro,
  Gs3_RootTestApi,
  Gs3_RootTestAta
] {}
```

28.3 Jexl расширения методов

С помощью расширений методов имеется возможность доработки и модификации стандартных серверных методов и операций пользовательских интерфейсов из приложения `Настройки системы`. Расширение представляет из себя jexl-скрипт, вызываемый перед или после выполнения стандартного метода/операции.

28.3.1 Расширения Api-методов

Добавление точек расширения в `Api` предусмотрено только в следующие методы:

- `Сеттеры атрибутов` – `Dpi`-сеттеры всех атрибутов выбранного класса;
- `AfterEdit` – `Dpi`-метод, вызываемый на окончание редактирования объекта класса, например при редактировании строки и перемещении курсора на другую строку;
- `BeforeEdit` – `Dpi`-метод, вызываемый в начале редактирования объекта класса;
- `FlushObject` – метод записи (сохранения) объекта в базу данных;
- `Delete` – `Dpi`-метод удаления объекта класса из базы данных;
- `Insert` – `Dpi`-метод создания нового объекта класса.

Примечание: Все расширения вызываются из „Dpi“. Т.е. срабатывание перед сеттером - это срабатывание перед вызовом `Dpi`-сеттера.

Примечание: Для методов «FlushObject» и «BeforeEdit» можно добавлять только точки расширения «До».

Выполнение этих точек будет производиться в контексте сессии, без использования интерактивной бизнес логики (т.е. нельзя использовать методы открытия всплывающих окон и диалогов, открытия каких либо интерфейсов системы).

Для подключения точек расширения Api-методов перейдите: `Настройки системы - Обозреватель проекта`. Выбрав Api нужного класса, перейдите на закладку `Точки расширения`. На закладке `Расширения до` или `Расширения после` создайти запись нового расширения и пропишите ему исполняемый скрипт.

Пример скрипта с использованием вызова процедур из Bts:

```
Bts_ProcedureApi.execByMnemoCode("ProcedureCode", {"argName1": value1, "argName2":␣
↪value2, ...})
```

Пример выброса ошибки, если у объекта не заполнен атрибут `sDescription`:

```
var aro = rop.copyAro();
if (aro.sDescription() == null){
  raise("Не заполнено описание!")
}
```

28.3.2 Расширения Avi-методов

Добавление точек расширения в `Avi` предусмотрено в сеттеры выборки, в ее операции или на события выборки, такие как beforeEdit, afterEdit и т.д.

Выполнение этих точек будет производиться в контексте выборки.

Для подключения точек расширения Avi-методов перейдите: `Настройки системы - Обозреватель проекта`. Выбрав нужную выборку, перейдите на закладку `Редактор операций`. На закладке `Расширения до` или `Расширения после` создайте запись нового расширения и пропишите ему исполняемый скрипт.

28.3.3 Пользовательские события

Реализована возможность для `Api` добавлять собственные события, а так же для существующих стандартных событий добавлять новые типы срабатывания.

Добавление нового события

Новое событие будет отображено в списке событий `Api`. Оно создается без типов срабатываний, типы срабатывание привязываются к событию отдельным методом.

Пример добавления нового события:

```
Btk_MethodExtensionApi().registerClassEvent(
  idvClass,
  "SomeUserEventName",
  "Новое пользовательское событие"
)
```

Пример удаления пользовательского события:

```
Btk_MethodExtensionApi().unregisterClassEvent(
  idvClass,
  "SomeUserEventName"
)
```

Добавление типа срабатывания к событию

Стандартные события (сеттеры, вставка и тд.) имеют по умолчанию 2 типа срабатывания: `До` и `После`, а пользовательские события создаются без типа срабатывания. Реализована возможность добавления нового типа срабатывания к событию. Каждый тип срабатывания будет отображаться отдельной закладкой у события.

Создание нового типа срабатывания:

```
Btk_MethodExtensionTypeApi().register("SomeUserEventType", "На пересчет денормализации")
```

Создание связи между событием и типом срабатывания:

```
Btk_MethodExtensionApi().registerEventTypeToClassEvent(
  idvClass,
  "SomeUserEventName",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  Btk_MethodExtensionTypeApi().findByMnemoCode("SomeUserEventType")
)
```

Удаление связи между событием и типом срабатывания:

```
Btk_MethodExtensionApi().unregisterEventTypeFromClassEvent(
  idvClass,
  "SomeUserEventName",
  Btk_MethodExtensionTypeApi().findByMnemoCode("SomeUserEventType")
)
```

Вызов пользовательского события или пользовательского типа срабатывания из кода приложения

Разработчик в коде сам размещает вызов срабатывания события в нужном месте.

Пример вызова:

```
Btk_MethodExtensionApi().runMethodExtensionByClass(idvClass, "SomeUserEventType",
↪"SomeUserEventName", "valueStr" -> "abc", "valueNum" -> 1.nn)
```

Переданные в метод значения параметров будут доступны в `Jexl-скриптах` этого события.

Пример обращения к переменной из `Jexl-скрипта`:

```
println("Значение переменной valueStr: " + valueStr + " Значение переменной valueNum: "␣
↪+ valueNum.toString());
```

Глава 29

Интеграция с сервером

29.1 Код инициализации модуля

На событие (пере-)загрузки прикладного кода, возможно выполнить свой код. Для этого, в каталоге со scala-кодом модуля, необходимо создать файл с именем `{модуль}_ModuleInit.scala`, содержащий

```
object {модуль}_ModuleInit {}
```

Пример:

```
Gs3_ModuleInit.scala
object Gs3_ModuleInit {
// Размещённый тут код будет выполнен при запуске загрузчика классов
  SBT
}
```

29.2 Обработка системных событий сервера

В процессе работы, сервер генерирует ряд системных событий, который можно обработать в прикладном коде. Для этого, необходимо подписаться на события объекта `ServerEventSource` в код инициализации модуля.

```
object Gs3_ModuleInit {
  private val _logger: Logger =
  Logger.Factory.get("ru.bitec.app.gs3.Gs3_ModuleInit")
  ServerEventSource.subscribeOnInstanceStart(
    (conAccessor) =>_logger.info("OnInstanceStart")
  )
  ServerEventSource.subscribeOnConnect(
    (conAccessor, btkUserInfo) =>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    _logger.info(s"OnConnect: ${btkUserInfo.name}")
  )
}
```

Перечень событий:

- `OnInstanceStart`
  Событие возникает после подключения пула sql-соединений сервера приложений к базе данных
- `OnInstanceStop`
  Событие возникает перед отключением пула sql-соединений сервера приложений к базе данных
- `OnConnect`
  Событие возникает в процессе запуска пользовательской сессиии, после успешной авторизации пользователя в системе. Для запрета подключения пользователя к системе, необходимо выбросить ошибку.
- `OnSessionStart`
  Событие возникает после начала сеанса работы, сеанс работы начинается после авторизации, перед открытием выборки выбора приложения
- `OnSessionStop`
  Событие возникает перед остановкой сеанса работы

Пример смотри: `ru.bitec.app.btk.Btk_ModuleInit`

Глава 30

Параллельные вычисления

Параллельные вычисления позволяют кратно ускорить расчет большого объема данных, за счет одновременного выполнения заданий по расчету.

Общий алгоритм параллельных вычислений в основной сессии:

1. Создать пул потоков
   Основной характеристикой пула потоков является кол-во возможных одновременно исполняемых задач.
2. Запросить данные для расчетов
3. Разбить данные для расчета на пачки для параллельного вычисления
   По каждой пачке необходимо сформировать задание на расчет
4. Направить задания для расчета в пул потоков
   При этом задания на расчет выполняются параллельно в пуле потоков

30.1 Планирование допустимого количества параллельных задач

1. Пусть S - количество доступных активных сессий базы данных
   S можно приблизительно рассчитать как `кол-во ядер доступных postgresql*2`
2. Пусть C - количество ядер доступных серверу приложения
3. Рассчитать количество потоков как:

```
Наименьшее(C*2,S*2)
```

Примечание: Данный алгоритм дает приблизительную оценку, так как оптимальное кол-во потоков зависит от соотношения нагрузки на диски и процессоры, а так же от административных квот на оборудование

30.2 Инструменты для параллельных вычислений

- `ru.bitec.app.gtk.eclipse.parallel.Parallel.withPool`
  Метод получения пула для параллельных вычислений.

  Доступен в контексте прикладной сессии(`api`,`pkg`,`avi`).

- `ASQL"""select..."""`.`withTempFileAs`
  Запрос с сохранением результата в файл. Данный запрос позволяет минимизировать потребление памяти и сессий базы данных во время долгих вычислений.

  Доступен в контексте прикладной сессии.

- `dialogs.withInfoForm`
  Отображает сообщение с индикацией расчета.

  Доступен в отображения

- `dialogs.showInfoForm`
  Обновляет сообщение с индикацией расчета.

  Доступен в отображении

Совет: Дополнительную информацию смотрите в документации методов.

30.3 Шаблон параллельных вычислений

```
dialogs.withInfoForm("Подготовка данных для расчета") {
    //Выполнить параллельные вычисления в 16 потоках
    Parallel.withPool(16) { pool =>
        //Размер пачки
        val batchSize = 500
        //Пачка для обработки
        val batch = ArrayBuffer.empty[NLong]
        //Количество обработанных записей
        var executedSize = 0

        //Отправка пачки на выполнение
        def submitBatch(): Unit= {
            val curSize = batch.size
            pool.submit(batch.toSet) { implicit session =>
                ids =>
                //ВНИМАНИЕ:
                //В процедуре вычисления не доступны данные из основной сессии
                //Поэтому работа может идти только с данными переданными
                //В процедуру submit, и полученными в текущие замыкание
                //в данном примере с именем ids
                //Передать можно только данные которые можно безопасно сериализовать
                //Попытка сослаться на другие данные приведет к ошибке в моменте
↪выполнения.

                //Массовая загрузка rop
                for (r <- Btk_QueryPkg().largeInQuery(SomeApi(), "id", ids)) {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                    //Выполнение вычислений
                }
            }.onSuccess { r =>
                //Обработка результатов выполнения идет в основном потоке
                //поэтому здесь доступно использование любых переменных

                //Расчет общего количества посчитанных данных
                executedSize = executedSize + curSize
                //Обновление диалога прогресса
                dialogs.showInfoForm(s"Выполнение расчетов ${executedSize}")
            }
            batch.clear()
        }

        //В данном примере запрос с большими данными для вычисления сохраняется в файл
        //Это позволяет сократить затраты по оперативной памяти и используемым
        //сессиям базы данных в момент расчет
        //Возможно работа с несколькими полями for(id~name<-...withTempFileAs(nLong("id
↪")~nStr("id")) )
        //Для этого необходимо сделать import anorm._
        for (id <-
                ASQL"""
                SELECT t.id  FROM SomeTable t
                """.withTempFileAs(nLong("id"))
        ) {
            //Формирование пачек на параллельное выполнение
            batch += id
            if (batch.size >= batchSize) {
                submitBatch()
            }
        }
        //Вычисление оставшихся данных
        if (batch.nonEmpty) {
            submitBatch()
        }
    }
}
```

Глава 31

Асинхронное обновление данных в связанных классах

Асинхронное обновление данных в связанных классах позволяет на событие в одном документе обновить атрибуты в другом, который может быть заблокирован в данное время.

Обновление происходит каждые пять минут.

> **Внимание:** На выполнение операции обновления даётся десять попыток, то есть в течении пятидесяти минут атрибуты обновятся или нет, если документ всё ещё заблокирован.

Что бы добавить обновить данные в связанном классе, необходимо зарегистрировать её в очереди:

```
Btk_SyncDataQueueApi().register(
  sOperCode = "UpdateDocVerByDoc",
  sJexl =
    """var ropDocVer = Wf_DocVerApi.load(123L);
      |Wf_DocVerApi.setsCaption(ropDocVer, "new Caption");
      |""".stripMargin,
  gidSrc = parent_Rop.gid,
  gidPurpose = purpose_Rop.gid,
  isOnlyLastExecute = false,
  isExecuteAnotherBySrcOnError = true
)
```

Очередь группируется по „Коду операции“ и „Глобальному идентификатору объекта источника“ и образует подгруппу выполняемых скриптов. В свою очередь подгруппа сортируется по дате создания и внутри по идентификатору записи очереди.

Флаг `isOnlyLastExecute` означает, что в подгруппе будет выполнен только последний скрипт, остальные будут помечены как не обязательные. Если признак выставлен только на одной записи в подгруппе, он повлияет на всю группу.

Флаг `isExecuteAnotherBySrcOnError` означает, что если текущий скрипт был выполнен с ошибкой, то последующие скрипты всё равно будут выполняться. Если признак не стоит, то последующие скрипты не будут выполнены, но у них будет увеличен счётчик пропуска.

Журнал очереди обновления данных можно посмотреть в приложении «Настройка системы» -> «Настройки и сервисы» -> «Очередь синхронизации данных классов». В этом же отображении можно внепланово синхронизировать данные.

Все исполненные операции автоматически очищаются через 180 дней после создания. А все не исполненные через 365 дней.

Глава 32

Средства мониторинга работы системы.

Мониторинг системы реализован в двух независимых формах:

1. Средства мониторинга встроенные в сервер приложений
   Реализованы по средствам обычных пользовательских интерфейсов системы, и расположены в меню инструментов.
2. Внешние средства мониторинга
   Реализованы на основе протокола передачи телеметрии `OpenTelemetry`. Метрики передаются в инструмент визуализации `Grafana`

32.1 Мониторинг в приложениях GlobalERP

Для открытия внутренних средств мониторинга необходимо выполнить операцию главного меню `Сервис` - `Инструменты` - `Монитор сессий сервера приложений`

32.1.1 Монитор сессий сервера приложений

Пользовательский интерфейс, который содержит в себе набор инструментов, для просмотра состояния сервера приложений и базы данных.

Пользовательские сеансы

Закладка `Пользовательские сеансы` отображает список всех сессий сервера приложений. Если сервер приложений работает в режиме кластера, то будут отображены сессии других узлов.

Перечень основных столбцов:

- `Системное имя пользователя` и `Пользователь` - информация о пользователе
- `Ip-адрес клиента` - ip-адрес, с которого подключено клиентское приложение
- `Системное имя приложения` и `Приложение` - информация о запущенном приложении `Global`

- Системное имя активной формы и Активная форма - информация о текущей активной форме
- Действие - информация о выполняемых действиях сессии.
- Кластерный узел - Имя узла, к которому подключен клиент.
- Имя базы данных - имя БД, выбранное в окне логина при авторизации.
- блок квот - отображает включены лы квоты на сессию, и текущие использованные ресурсы.

Перечень закладок

- Сессии базы данных - отображает активные подключения к БД, выбранной пользовательской сессии. Позволяет просмотреть выполняемый запрос, и увидеть блокирующую сессию, если она есть
- Расшифровка - отображает информацию об открытых формах пользовательской сессии
- Стек сеанса - позволяет получить актуальный стек выполняемых действий пользовательской сессии.

Активные подключения к базе данных

Закладка Активные подключения к базе данных позволяет отобразить активные в данный момент запросы к БД, а так же информацию о пользовательских сессиях, выполняющих запросы.

Перечень основных столбцов:

- Системное имя пользователя и Пользователь - информация о пользователе
- Ip-адрес клиента - ip-адрес, с которого подключено клиентское приложение
- Системное имя приложения и Приложение - информация о запущенном приложении Global
- Системное имя активной формы и Активная форма - информация о текущей активной форме
- Действие - информация о выполняемых действиях сессии.
- Кластерный узел - Имя узла, к которому подключен клиент.
- Имя базы данных - имя БД, выбранное в окне логина при авторизации.

32.1.2 Инструмент анализа Базы данных

Расположен так же на форме Монитор сессий сервера приложений на закладке Анализ базы данных. Позволяет увидеть состояние сессий БД, блокировки и выполняемые запросы.

Анализ размера таблиц

Инструмент, который позволяет увидеть текущий размер таблиц БД. Операция История позволяет отследить динамику увеличения таблиц

Работа автовакуума

Интерфейс, который отображает информацию о работе автоматической очистке таблиц.

Автовакуум - специальный процесс БД `postgresql`, который занимается очисткой устаревших данных в таблицах. Архитектура `postgresql` каждое изменение в таблице регистрирует новой записью, и требуется удалять устаревшие записи, которые не являются актуальными.

Документация postgresql vacuum

На этой закладке, основное внимание требуется уделять таблицам с выключенным автовакуумом. При нормальной работе системы, все таблицы должны автоматически очищаться.

Доступные операции:

- `Запуск vacuum full`
  Команда на полную очистку таблицы с опцией `full`.

  Выбирает режим «полной» очистки, который может освободить больше пространства, но выполняется гораздо дольше и запрашивает исключительную блокировку таблицы. Этот режим также требует дополнительное место на диске, так как он записывает новую копию таблицы и не освобождает старую до завершения операции. Обычно это следует использовать, только когда требуется высвободить значительный объём пространства, выделенного таблице.

- `Запуск vacuum analyze`
  Обновляет статистику, которую использует планировщик для выбора наиболее эффективного способа выполнения запроса.

Операции следует запускать, только если автовакуум был отключен, или таблица часто изменяется, и автоматическая очистка не справляется.

Анализ активных запросов

Инструмент, который позволяет отобразить текущие активные запросы к БД, и информацию о вызвавших их пользовательских сессиях.

Перечень основных столбцов:

- `pid` - pid процесса БД, по которому можно идентифицировать сессию БД.
- `Время выполнения` - время выполнения запроса в минутах.
- `Состояние` - состояние сессии БД
  Основные состояния:
  - `idle` - сессия БД находится в режиме ожидания. Т.е. подключение к БД есть, но в данный момент никаких запросов не выполняется. Стандартное ожидание для неактивных подключений, которые находятся в `пуле подключений сервера приложений`
  - `active` - сессия БД в данный момент выполняет запрос.
  - `idle in transaction` - сессия БД находится в режиме ожидания, но есть открытая транзакция. Опасное состояние сессии, т.к. длинные открытые транзакции отрицательно влияют на производительность `postgresql`. Наличие такой сессии сигнализирует о проблеме.
- `Системное имя пользователя` и `Пользователь` - информация о пользователе
- `Ip-адрес клиента` - ip-адрес, с которого подключено клиентское приложение
- `Системное имя приложения` и `Приложение` - информация о запущенном приложении `Global`
- `Форма` - информация о текущей активной форме

- `Кластерный узел` - Имя узла, к которому подключен клиент.
- `Действие` - информация о выполняемых действиях сессии. Эта колонка будет заполнена, если:
  - сессия принадлежит выполняемому фоновому заданию, в том числе интеграция
  - выполняется построение печатной формы

Операция `История` позволяет просмотреть исторические данные активных запросов, чья длительность превышала 15 минут.

Блокировки

Отображает дерево блокировок сессий БД.

Блокировка записи — метод предотвращения одновременного доступа к данным в базе данных, чтобы предотвратить противоречивые результаты. Чаще всего возникают, если несколько сессий БД пытаются изменить одну и туже запись в таблице.

Корневыми записями этого дерева будут сессии, вызывающие какие-либо блокировки других сессий. Записи второго и следующих уровней показывают сессии, которые ожидают освобождения блокировок.

Таким образом можно проследить цепочки сессий, ожидающих освобождения блокировок.

Операция `История` позволяет просмотреть исторические данные по блокировкам БД.

Журнал алертов

Специальный журнал, в который периодически пишется информация о выявленных проблемах.

Журнал заполняется по средствам фоновых заданий:

1. `Сохранить Долгий запрос` - `Btk_SlowQueryHistoryUpdate`
2. `Сохранить размер таблиц на дату` - `Btk_TableSizeHistoryUpdate`

События записываемые в журнал:

1. Запрос БД выполняется более 15 минут
2. Резкое увеличение размеров таблиц
3. На таблице выключен автовакуум
4. Транзакция в БД выполняется более 15 минут
5. Фоновое задание с ночным расписанием выполняется в дневное время

32.2 Мониторинг телеметрии в Grafana

32.2.1 Общая информация о внешних средствах мониторинга и телеметрии

Внешний мониторинг организован следующим образом:

1. Сервер приложений, используя стандарт `OpenTelemetry` отправляет метрики, трассировки и логи во внешний сервис (коллектор).
2. Коллектор перенаправляет метрики, трассировки и логи в инструмент визуализации данных `Grafana`
3. В `Grafana` реализованы дашборды визуализации метрик и трассировки.

Конфигурирование сервера приложений

В сервере приложений в каталоге с конфигурациями (../application/config) располагаются конфигурационные файлы телеметрии:

- `otel-globalserver.config.yaml`
  Основной конфигурационный файл OpenTelemetry SDK. Управляет активностью телеметрии и настройками экспорта.
- `otel-sdk.config.yaml`
  Дополнительный конфигурационный файл. Управляет настройками системной телеметрии, специфичной для сервера приложений.

Документация телеметрии сервера приложений

Основные моменты:

1. Для включения телеметрии сервера приложений необходимо настроить файл `otel-globalserver.config.yaml`
   - включить телеметрию общим флагом (`disabled: false`)
   - настроить `endpoint`-ы для коллекторов метрик, логов и трассировки
2. Изменения конфигурационных файлов применятся только после перезагрузки экземпляра сервера приложений.

Grafana как средство визуализации

Grafana — это платформа с открытым исходным кодом для визуализации, мониторинга и анализа данных.

Позволяет отображать данные в графическом виде из разных источников данных.

В нашем мониторинге используются следующие источники данных:

- `Prometheus` - база данных метрик
- `Loki` - база данных логов
- `Tempo` - система хранения трассировок

Основы стандарта OpenTelemetry

OpenTelemetry - стандарт сбора и передачи телеметрических данных. Стандарт описывает набор практик и инструментов, определяющих, какие сигналы может генерировать приложение.

Три основных типа сигналов:

- `метрики`
- `трассировка`
- `логи`

Каждый сигнал может обладать рядом дополнительных свойств, которые могут характеризовать событие, с которым связан сигнал. Например:

- `Имя пользователя`
- `Кластерный узел`
- и тд.

Метрики

`Метрики` - количественные данные о работе системы. Например:

- Время выполнение операции
- Количество успешно выполненных операций
- Количество ошибок при выполнении операций.
- Время выполнения запроса к БД
- и тд.

Метрики делятся на несколько типов:

- `Counter (увеличивающийся счетчик)`
  Представляет из себя метрику, которая увеличивается с течением времени. Например, количество запросов к БД.
- `UpDownCounter (увеличивающийся и уменьшающийся счетчик)`
  Представляет из себя метрику, которая может как увеличиваться так и уменьшаться с течением времени. Например, количество подключений к БД
- `Gauge (измеритель, спидометр)`
  Представляет из себя метрику, которая используется для измерения чего-либо в конкретный момент времени. Например, время выполнения запроса к БД
- `Histogram (гистограмма)`
  Позволяет накапливать исторические данные на клиентской стороне, и затем отправлять их сигналом во внешний мир.

Трассировка

В стандарте `OpenTelemetry` реализована через объекты с типом `span`.

`Span` - сигнал, который имеет начало и конец, а так же может иметь родительский `span`. С помощью этих сигналов можно построить дерево трассировки, вкладывая `span`-ы друг в друга, и отмечая начало и конец события.

Пример дерева `span-ов`:

```
+ Выполнение операции
  |_+ Выполнение запроса к серверу приложений
    |_+ Выполнение запроса к БД
    |_+ Обработка результатов запроса к БД
```

Логи

Сигнал, который из себя представляет логовую запись, сделанную системой.

32.2.2 Типы и виды метрик сервера приложений

Сервер приложений посылает различные виды сигналов. Часть из них отправляется сервером приложений, другая же часть прикладным решением.

Системные метрики

Метрики сервера приложений, настраиваемые через `Администратор метрик`:

- `Время реакции на действие пользователя`
  - `rpc.server.duration.nanos` - длительность обработки `RPC` сервером. Число наносекунд.
  - `rpc.client.duration.millis` - длительность выполнения `RPC` с точки зрения клиента, равная длительность передачи по сети + длительность обработки `RPC` сервером. Число миллисекунд

Полный перечень метрик сервера приложений

Прикладные метрики и трассировки

Метрики и трассировки, отправляемые из кода прикладного решения.

Метрики пользовательских операций

Специальные метрики, которые отслеживают выполнение пользователем операций в пользовательском интерфейсе. На каждое выполнение операции осуществляется отправка различных метрик:

- `btk_item_telemetry.user_operation_exec_time` - время выполнения операции
- `btk_item_telemetry.user_operation_calls_total` - количество вызовов операции
- `btk_item_telemetry.user_operation_calls_successful` - количество успешных вызовов операции
- `btk_item_telemetry.user_operation_calls_failed` - количество неуспешных вызовов операции

Атрибуты метрик:

- `solution` - Имя решения из конфигурации сервера приложений
- `userName` - Имя пользователя
- `session.sid` - Sid сессии сервера приложений
- `work.session.sid` - Sid рабочего сеанса сервера приложений
- `cluster.node` - имя кластерного узла
- `selection.name` - имя выборки
- `representation.name` - имя отображения
- `selection.caption` - наименование выборки
- `oper.name` - имя операции
- `oper.caption` - наименование операции
- `form.name` - имя главной выборки формы
- `form.caption` - наименование главной выборки формы

- `form.representation.name` - имя отображения главной выборки формы

Трассировки пользовательских операций

- `btk_item_telemetry.user_operation_trace` - трассировка выполнения операции.

  Пишется при выполнении пользователем какой-либо операции или сеттера. На каждое пользовательское действие создается `span`, если это действие вызвало другие операции или события, они будут добавлены в дерево трассировки.

- `btk_item_telemetry.open_selection_trace` - трассировка открытия выборки.

  Пишется, если на выборку целиком включена трассировка или для пользователя включена вся телеметрия. Представляет из себя дерево `span`-ов, в котором будут отображены все операции, события и сеттеры, выполняемые при открытии выборки.

  Если выборка имеет дочерние выборки, то трассировка их открытия, так же будет добавлена в дерево `span-ов.

Атрибуты трейсов:

- `solution` - Имя решения из конфигурации сервера приложений
- `userName` - Имя пользователя
- `session.sid` - Sid сессии сервера приложений
- `work.session.sid` - Sid рабочего сеанса сервера приложений
- `cluster.node` - имя кластерного узла
- `selection.name` - имя выборки
- `representation.name` - имя отображения
- `selection.caption` - наименование выборки
- `oper.name` - имя операции
- `oper.caption` - наименование операции
- `form.name` - имя главной выборки формы
- `form.caption` - наименование главной выборки формы
- `form.representation.name` - имя отображения главной выборки формы

Метрики и трассировки, объявленные в прикладных модулях

Разработчики в прикладном коде могут объявлять собственные метрики и трассировки для реализации телеметрии в каких-либо высоко-нагруженных местах.

Пример работы с объектной метрикой

1. Объявление метрики (вызов требуется добавить в dataInstall)

```
//Объявление метрики с именем Some_Metric_Name, которая относится к Адм. объекту␣
↪Some_ObjectName
Btk_AcObjectMetricApi().register(
   Btk_AcObjectApi().findByMnemoCode("Some_ObjectName"),
   "Some_Metric_Name",
   "Пример метрики прикладного кода",
   "Демонстрация использования прикладной метрики "
)
```

2. Использование метрики в коде

```
//получение счетчика
Btk_TelemetryPkg().getAcObjectLongCounter("Some_Metric_Name", "Some_Metric_Name").
↪foreach{counter =>
   //увеличение счетчика, если эта метрика включена в администраторе метрик
   counter.add(42)
}
```

Пример работы с объектной трассировкой

1. Объявление трассировки

```
//Объявление трассировки с именем Some_Trace_Name, которая относится к Адм. объекту␣
↪Some_ObjectName
Btk_AcObjectTraceApi().register(
   Btk_AcObjectApi().findByMnemoCode("Some_ObjectName"),
   "Some_Trace_Name",
   "Пример метрики прикладного кода",
   "Демонстрация использования прикладной метрики "
)
```

2. Использование трассировки

```
//получение счетчика
Btk_TelemetryPkg().getAcObjectTrace("Some_Trace_Name", "Some_Metric_Name").foreach
↪{spanBuilder =>
  //выполнение действий с созданием span-а. Перед началом действия создается span,␣
↪по окончанию действия - span закрывается.
  spanBuilder.forSpan{span =>
    try {
      //код выполнения
      Thread.sleep(5000)
    } finally {
      //закрытие span-а
      span.end()
    }
  }
}
```

Ядровые метрики приложения

- `app.transaction.active.count` - Кол-во открытых транзакций БД
- `app.jexl.count` - Кол-во выполненных jexl скриптов
- `app.rest.count` - Кол-во полученных Rest-запросов
- `app.error.count` - Кол-во ошибок (бизнес-логика)
- `app.setting.change.count` - Число измененных системных настроек

Метрики использованных ячеек памяти

- `app.work_session.ui_cell.count` - Кол-во ui-ячеек
- `app.work_session.ui_row.count` - Кол-во ui-строк
- `app.work_session.read_cell.count` - Кол-во загруженных ячеек
- `app.work_session.insert_cell.count` - Кол-во созданных ячеек
- `app.work_session.update_cell.count` - Кол-во измененных ячеек

Атрибуты метрик:

- `solution` - Имя решения из конфигурации сервера приложений
- `user` - Имя пользователя
- `session_id` - Sid сессии сервера приложений
- `session_kind` - Тип сессии сервера приложений

Метрики фоновых заданий

- `app.job.active.count` - Количество активных джобов
- `app.job.run.failed` - Количество ошибок джобов

Включение и фильтрация прикладных метрик осуществляется через `Администратор метрик`

32.2.3 Администратор метрик

Инструмент, который позволяет управлять активностью и фильтрацией прикладных и серверных метрик.

Расположен `Приложение Администратор - Настройки - Телеметрия`.

В интерфейсе слева располагается список с возможными настройкам, справа - отображается выборка этих настроек.

Управление прикладными метриками

Адм. объекты

Эта настойка позволяет включить метрики и трассировку для операций пользовательского интерфейса. Представляет из себя дерево, где корневым узлом является `администрируемый объект`, на втором уровне расположены выборки, входящие в этот объект.

Варианты включения:

- Конкретная операция
- Выборка целиком
- Бизнес объект целиком
- для всех пользователей
- для конкретных пользователей

В детализации расположены закладки:

- `Настройка телеметрии выборки` - управляет включением телеметрии на выборку целиком.
  - `Все пользователи` - при установке признака для всех пользователей, выполняющих операции этой выборки, будет отправлять телеметрия
  - `Включена трассировка` - признак означает, что для всех операций выборки будет формироваться трассировка
  - `Включены метрики` - признак означает, что для всех операций выборки будут формироваться метрики
- `Настройка телеметрии элементарных привилегий` - управляет включением телеметрии на конкретные операции выборки. Отображает список элементарных привилегий выборки, и позволят включать телеметрию точечно. Детализация `Настройка телеметрии объекта` функционально повторяет закладку `Настройка телеметрии выборки`, но ее настройки относятся к конкретным операциям.

Пользователи

Настройка, которая позволяет включить все прикладные метрики и трассировки для конкретного пользователя. Если пользователь добавлен в этот список, и включен признак `Включена вся телеметрия`, то будет оправляться телеметрия по всем операциям и объектным метрикам и трассировкам.

Объектные метрики и трассировки

Инструмент, предназначенный для объявления метрик и трассировок прикладного кода. Позволяет разработчикам объявить свои собственные метрики и трассировки, и использовать как телеметрию для особо-нагруженных или проблемных участков кода приложения.

Основной принцип работы:

1. Разработчик регистрирует метрику или трассировку в специализированном справочнике, присвоив им уникальное системное имя.
2. В прикладном коде в необходимых местах отправляет значения метрик или трассировки, используя методы пакета `ru.bitec.app.btk.telemetry.Btk_TelemetryPkg`

3. В `администраторе метрик` администратор включает или выключает активность объектных метрик и трассировок.

Управление системными метриками

Управляется через закладку `Серверная телеметрия`. Позволяет настроить необходимость отправки метрик сервера приложений для всех пользователей и для конкретных.

Управляет метриками `Время реакции на действие пользователя` сервера приложений.

32.2.4 Grafana

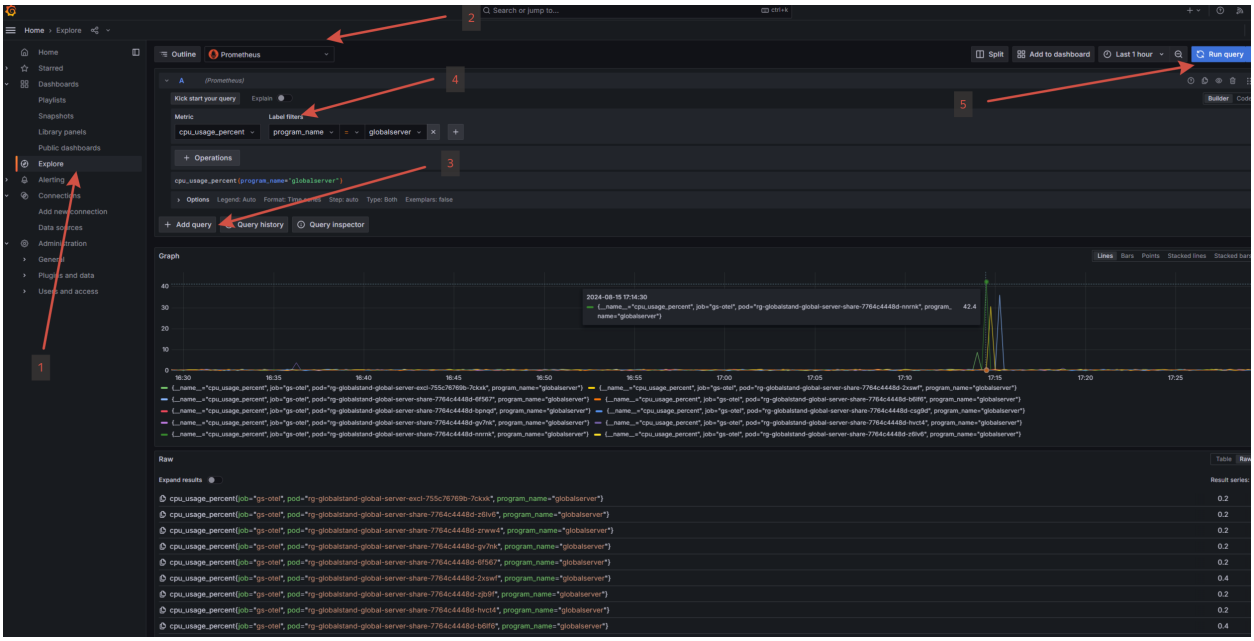
Графана поставляется как отдельный сервис. Если система развернута в виде кластера `Kubernetes`, то по умолчанию для подключения к веб-интерфейсу используется 3000 порт основного адреса. Т.е. если для входа в систему используется адрес `http://192.168.29.17`, то для подключения к веб-интерфейсу графаны используется адрес `http://192.168.29.17:3000/`. Это может быть изменено, и конфигурируется системным администратором.

Для проектов, где система развернута не через кластер `Kubernetes`, адрес подключения определяется системным администратором.

Запрос метрик

Для просмотра логов выполните следующие действия:

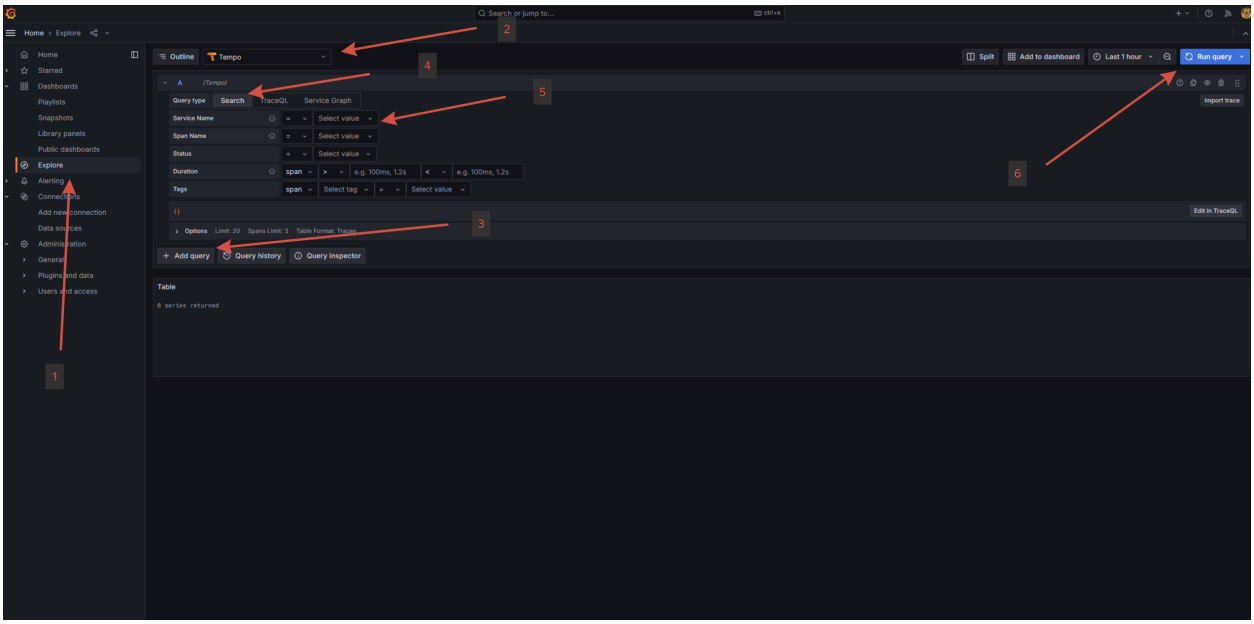
1. В меню выберите `Explore`
2. Выберите источник данных `Prometheus`
3. Добавьте новый запрос, если требуется
4. Укажите один из фильтров:
   - `metric` - имя конкретной метрики
   - `label` - дополнительные атрибуты метрики:
     - `program_name` - имя сервиса, поставщика метрик. (globalserver, globalscheduller и тд.)
     - `podname` - имя кластерного узла
     - и тд.
5. Выполните запрос, нажав `Run query`



Запрос трассировки

Для просмотра трассировки выполните следующие действия:

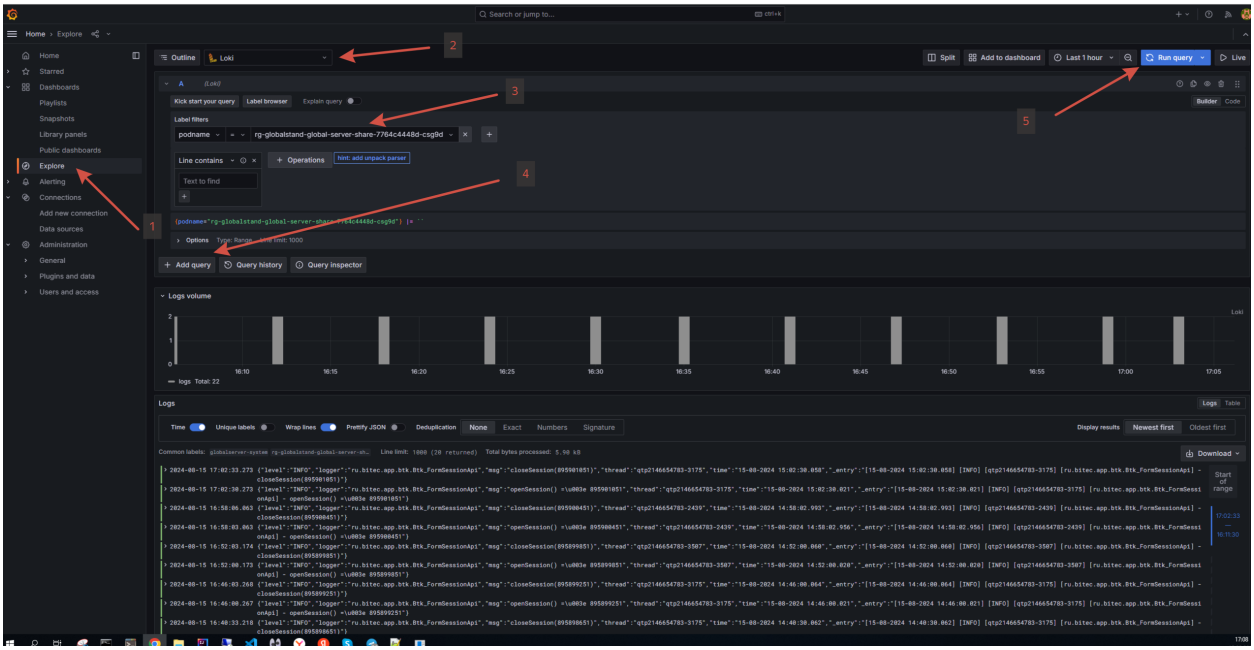
1. В меню выберите `Explore`
2. Выберите источник данных `Tempo`
3. Добавьте новый запрос, если требуется
4. Укажите `QueryType` = `Search`
5. Укажите один из фильтров:
   - `Service Name` - имя сервиса, поставщика метрик. (globalserver, globalscheduller и тд.)
   - `Span Name` - имя трассировки
6. Выполните запрос, нажав `Run query`



Просмотр логов

Для просмотра логов выполните следующие действия:

1. В меню выберите `Explore`
2. Выберите источник данных `Loki`
3. Добавьте новый запрос, если требуется
4. Укажите один из фильтров:
   - `filename` - имя файла логов
   - `log` - имя сервиса, чьи логи требуется просмотреть (globalserver, globalscheduller и тд.)
   - `podname` - имя кластерного узла
5. Выполните запрос, нажав `Run query`

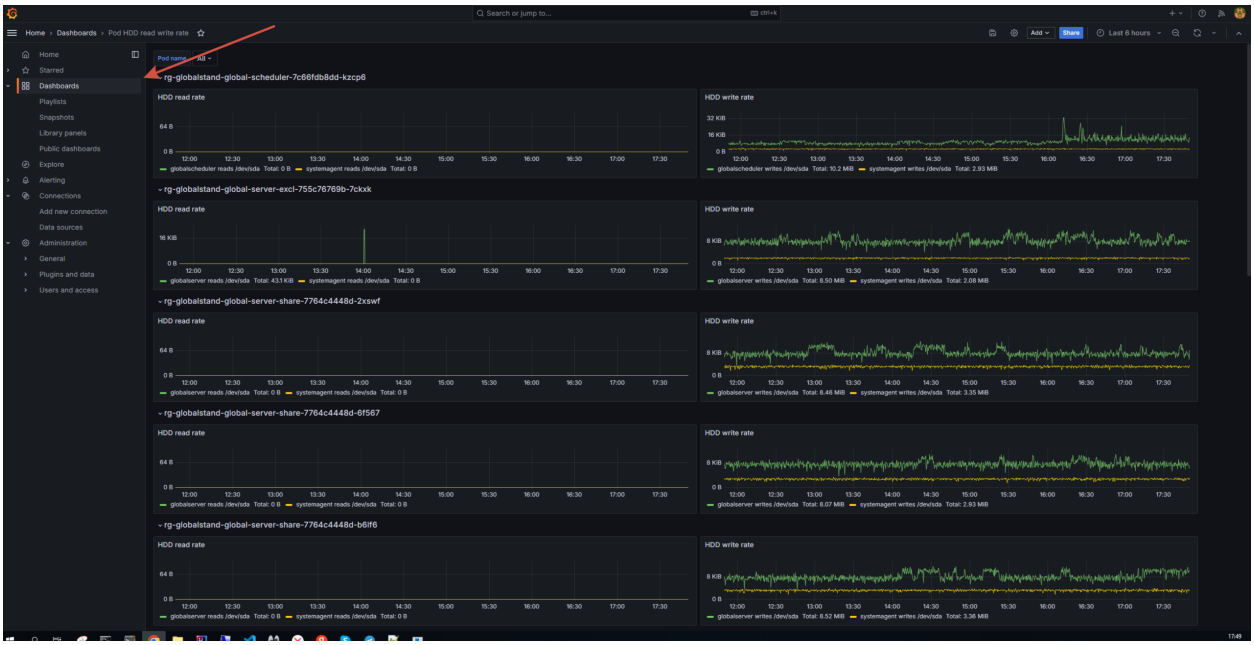


Дашборды

Графана поставляется с преднастроенными дашбордами, которые позволяют отобразить ту или иную информацию.

Для просмотра логов выполните следующие действия:

1. В меню выберите `Dashboards`
2. В появившемся списке выберите нужный дашборд.



Глава 33

Сервисы сервера приложений

Сервисы реализованы в опциональном модуле sys, который входит в дистрибутив Global Server Scala Edition. Так же модуль может быть подключен в конфигурационном файле сервера `global3.config.xml`:

```
<sbt name="sys"
      lazyLoad="false"
      source="C:\global3\sysappbin\"
      sourceMode="Jar"
      binaryFolder="C:\global3\sysappbin\"
/>
```

Ленивая загрузка должна быть отключена, для того чтобы сервисы могли всегда принимать запросы.

Более подробную информацию смотрите в руководстве по администрированию сервера.

33.1 SSH консоль сервера

33.1.1 Введение

Global 3 SE Server включает в себя SSH (Secure Shell) сервер, к которому возможно подключиться с помощью любого SSH-терминала. С помощью команд командной строки возможно:

- Смотреть статистику, логи, и управлять клиентскими сессиями
- Перезагружать код прикладных приложений
- Перезапускать Web-приложения
- Выполнять SQL и Jexl-скрипты

33.1.2 Конфигурирование

Конфигурационный файл global3.config.xml может содержать секцию `<ssh/>`:

```
<ssh defaultDb="PostgreSql" port="22"/>
```

- `defaultDb` - Алиас базы данных по умолчанию
  Данное значение используется в качестве значения параметра по умолчанию для ssh-команды `login`. По имени базы будет определены имя и пароль пользователя, для подключения к базе.
- `port` – Порт ssh-сервера
  По умолчанию: 22.
  Данное значение может быть jvm опцией

  ```
  -Dglobal.ssh.port=xxxx
  ```

  или параметрами командной строки при запуске Global3 сервера

  ```
  -global.ssh.port xxxx
  ```

Учётные данные пользователей, имеющих права на ssh-сединения, указываются в секции `<security/>` конфигурационного файл global3.config.xml:

```
<security>
  <users>
    <user name="admin" password="admin" roles="ssh"/>
  </users>
</security>
```

- `user` – Пользователь
- `password` | `encryptedPassword` – Пароль
  Пароль может быть указан как в явном, так и в зашифрованном виде.
- `roles` – Роли
  Список ролей, через запятую, доступных пользователю.

Шифрование пароля

Для шифрования пароля необходимо запустить Global 3 Server с парамерами

```
Start.bat -encryptPassword password -masterPassword masterpassword
```

Где:

- `-encryptPassword`
  Пароль который необходимо зашифровать
- `-masterPassword`
  Ключ шифрования, если не указан используется секретный ключ по умолчанию.

Результатом выполнения будет вывод в консоль строки, полученной в результате шифрования пароля переданного параметром `-encryptPassword`.

33.1.3 Подключение

Подключиться к SSH-серверу возможно любым SSH-терминалом. Рекомендуемым клиентом для подключения является PuTTY.

Документация по putty

Основные параметры

Для подключения необходимо указать:

- *Host Name*
  host или user@host
- *Port*
  22 или пользовательский
- *Connection type*
  SSH

Что бы не вводить значения полей при повторном запуске PuTTY, можно сохранить параметры подключения по умолчанию, нажав кнопку «Save».

Пример запуска из командной строки:

```
"C:\Program Files\PuTTY\putty.exe" -ssh localhost -P 22 -l admin -pw admin
```

Логирование

Для сохранения всего текстового вывода в файл, на вкладке Logging диалога подключения необходимо указать:

- *Session Logging* = `All session output`
- *Log file name* = `полный путь к файлу`

Внимание: В адресах пути:

- Символ \ необходимо экранировать удвоением
  Пример: \\
- Символ / экранировать не надо

Подключение с использованием SSH-RSA ключа

При подключении к SSH из `*.bat` иил `*.sh` можно воспользоваться RSA-ключами вместо пароля.

```
"C:\Program Files\PuTTY\putty.exe" -ssh localhost -P 22 -l admin -i ssh_admin_private.ppk
```

Необходимые файлы размещены в подкаталоге `.\ssh`.

Ключи для подключения к пользователю `admin` выдаются по запросу.

Запуск терминала

При первом подключении к серверу будет выдано сообщение с запрашивающие разрешение на подключение к серверу, нажмите: Да.

33.1.4 Команды

Для получения актуальной справки по доступным командам, выполните команду `help`.

Список команд:

- `alter server mode service|normal`
  Переключает сервер в сервисный режим и обратно. В сервисном режиме возможно подключение пользовательских сессий только от имени системного пользователя сервера приложений, указанного в конфигурационном файле.

```
<systemUsers>
        <user name="system" password="system"/>
</systemUsers>
```

- `attach session {sid}`
  Подключение к существующей пользовательской сессии.
- `attach db {dbAlias} [as sys]`
  Подключение к базе данных для выполнения сервисных операций и/или Jexl-скриптов.
- `clear`
  Очищает экран терминала.
- `clear persistence cache [{dbAlias}]`
  Очищает Shared-кэш объектов.
- `compare applib {path}`
  Сравнивает jar-файлы в каталоге (или zip-архиве) `{path}` с jar-файлами в каталоге `SBT.jarFolder`, соответствующему базе данных, к которой выполнено подключение командой `attach db`.

  Выполняется сравнение версий модулей.
  Выполняется сравнение *.odm.xml файлов на предмет существования новых атрибутов
- `copy applib [force] {path}`
  Выполняет копирование jar-файлов из каталога `{path}` в каталог `SBT.jarFolder`, соответствующий базе данных, к которой выполнено подключение командой `attach db`. При указании ключевого слова `force` перед копированием не выполняется сравнения `*.odm.xml` файлов. Сравнение версий модулей выполняется в любом случаи.
- `Init schema`
  Выполняет создание/обновление объектов схемы БД
- `execute {expression}`
  Выполняет однострочное Jexl-выражение
- `exit`
  Закрывает SSH-подключение
- `jexl [{file}]`
  Переключает терминал в режим ввода и выполнения

33.1.5 Выполнение SQL

Для выполнения sql-скрипта:

1. Подключитесь к сессии командами `login`, `attach` или `set sid`.
2. Перейдите в режим ввода скрипта командой `sql`.
3. Введите текст скрипта.
4. Выполните скрипт
   Для этого введите символ `/` с новой строки.
5. Выйдите из режима ввода скрипта
   Для этого введите символ `/` с новой строки.

Пример:

```
login admin/admin@postgres
sql
INSERT INTO gs3_roottest (
    id,
    idClass
  ) VALUES (
  (select
    nextval('GS3_ROOTTEST_SEQ'))
    , 12351
  ) ON CONFLICT DO NOTHING;
/
/
```

Выполнение SQL скрипта из файла

Для выполнения sql-скрипта из файла:

1. Подключитесь к сессии.
2. Выполните команду `sql {file}`.
   Где `{file}` – путь к файлу к файлу на сервере.

> **Внимание:** Файл должен находится на локальном диске сервера.

Пример:

```
login admin/admin@postgres
sql D:\\svn\\depot\\ASSource\\sysapplication\\ssh\\src\\test\\java\\ru\\bitec\\app\\ssh\\
↪shh_sql_exams.txt
```

Содержимое файла:

```
INSERT INTO gs3_roottest (id, idClass) VALUES ((select nextval('GS3_ROOTTEST_SEQ')),␣
↪12351) ON CONFLICT DO NOTHING;
INSERT INTO gs3_roottest (id, idClass) VALUES ((select nextval('GS3_ROOTTEST_SEQ')),␣
↪12351) ON CONFLICT DO NOTHING;
/
```

(continues on next page)

(продолжение с предыдущей страницы)

```
INSERT INTO gs3_roottest (id, idClass) VALUES ((select nextval('GS3_ROOTTEST_SEQ')),␣
↪12351) ON CONFLICT DO NOTHING;
INSERT INTO gs3_roottest (id, idClass) VALUES ((select nextval('GS3_ROOTTEST_SEQ')),␣
↪12351) ON CONFLICT DO NOTHING;
/
/
```

33.1.6 Jexl скрипты

Выполнение Jexl скрипта

Для выполнения Jexl-скрипта необходимо:

1. Подключитесь к сессии
2. Переключитесь в режим ввода скрипта командой `jexl`.
3. Введите текст скрипта.
4. Выполните скрипт
   Для этого введите символ / с новой строки.
5. Выйдите из режима ввода
   Для этого введите символ / с новой строки.

Пример 1:

```
login admin/admin@postgres
jexl
var name = Btk_ClassApi.getCanonicalClassName("Btk_Object");
Btk_ClassApi.getApiByCanonicalClassName(name);
/
/
```

Пример 2 (для dataInstall):

```
login admin/admin@postgres
jexl
Bbb_DBTypeApi.dataInstall();
Btk_Pkg.commit();
/
/
```

Пример 3:

```
login admin/admin@postgres
jexl
Btk_Pkg.setRWSharedUOWEditType();
Prs_EntTransApi.dataInstall();
Btk_Pkg.commit();
/
/
```

Выполнение Jexl скрипта из файла

Для выполнения Jexl-скрипта из файла необходимо:

1. Подключитесь к сессии
2. Выполнить команду jexl `{file}`
   Где `{file}` – путь к файлу к файлу на сервере.

> **Внимание:** Файл должен находится на локальном диске сервера.

Пример:

```
login
jexl D:\\svn\\depot\\ASSource\\sysapplication\\ssh\\src\\test\\java\\ru\\bitec\\app\\ssh\
↪\shh_jexl_exams.txt
```

Содержимое файла:

```
var name = Btk_ClassApi.getCanonicalClassName("Btk_Object");
Btk_ClassApi.getApiByCanonicalClassName(name);
/
/
```

Контекст выполнения Jexl скрипта

Возможны следующие контексты выполнения Jexl-скрипта.

Пользовательская сессия

```
>login user/password@alias
>jexl
jexl>
/
/
>
```

В данном контексте доступны все Api-классы, присутствующие в SBT, соответсвующего пользовательской сессии.

База данных

```
>attach db alias
>jexl
jexldb>
/
/
>
```

Доступны методы управления инстансом базы данных:

- `initschema()`
  Выполняет инициализацию/обновление схемы БД в соответствии с текущей прикладной кодовой базой.

Системный контекст

```
>attach db alias as sys
>jexl
jexlsys>
/
/
>
```

Данный контекст является административным и предназначен для управления сервером приложений. Доступны следующие методы:

- `upgrade({release_path})`
  Выполняет инициализацию/обновление схемы БД в соответствии с текущей прикладной кодовой базой.
  Где:
  `{release_path}` – путь к каталогу или zip-архиву с релизом прикладных модулей

33.1.7 Выполнение командного файла ssh

Для выполнения командного файла из командной строки можно выполнить:

```
"C:\Program Files\PuTTY\putty.exe" -ssh localhost -P 22 -l admin -pw admin -m shh_logger_
↪test_script.txt
```

Внимание: В данном примере файл shh_logger_test_script.txt размещён на диске клиентской машины, он считывается в момент подключения к SSH-серверу. Пути к файлам в скрипте, являются локальными для сервера.

Содержимое файла:

```
login
log-info app el

execute Btk_ClassApi.getCanonicalClassName("Btk_Object")

jexl
var name = Btk_ClassApi.getCanonicalClassName("Btk_Object");
Btk_ClassApi.getApiByCanonicalClassName(name);
/
/

jexl D:/svn/depot/ASSource/sysapplication/ssh/src/test/java/ru/bitec/app/ssh/shh_jexl_
↪exams.txt
jexl D:\\svn\\depot\\ASSource\\sysapplication\\ssh\\src\\test\\java\\ru\\bitec\\app\\ssh\
↪\shh_jexl_exams.txt
```

(continues on next page)

(продолжение с предыдущей страницы)

```
sql
INSERT INTO gs3_roottest (id, idClass) VALUES ((select nextval('GS3_ROOTTEST_SEQ')),␣
↪12351) ON CONFLICT DO NOTHING;
/
/

sql D:\\svn\\depot\\ASSource\\sysapplication\\ssh\\src\\test\\java\\ru\\bitec\\app\\ssh\\
↪shh_sql_exams.txt

log-off all
logout
```

Логирование

При выполнении скрипта, по умолчанию, в ssh-консоль выводится результат выполнения команды или сообщение об ошибке со стеком вызова.

Для вывода в ssh-консоль дополнительных логов, необходимо их включить. В консоль можно вывести логи следующих типов:

- `oper`
  Логи из классов с namespace `ru.bitec.engine.model.operation`.
- `sql`
  Логи sql-вызовов с уровня jdbc-соединения с базой.
- `script`
  Логи из скриптового языка в режиме совместимости с Global 1. При работе со Scala не имеют смысла.
- `app`
  Логи из классов прикладной логики с namespace `ru.bitec.app.*`.
  Пример отправки сообщения:

  ```
  Logger.Factory.get(Xxx_XxxApi.class).info("Текст сообщения")
  ```

- `el`
  Логи из инфраструктуры EclipseLink. В основном, это sql-вызовы, в более компактном виде, чем логи jdbc.
- `all`
  Все выше перечисленные типы логов.

Для переключения уровней логирования используются команды:

- `log-off`
- `log-error`
- `log-warn`
- `log-info`
- `log-debug`
- `log-trace`

Команды указаны в порядке уменьшения уровня логов.

В результате выполнения этой команды:

```
log-info app el
```

в ssh-лог будут попадать сообщения типов `app` и `el` для уровней логирования: `error`, `warn`, `info`.

Для отключения вывода сообщений в ssh-лог необходимо вызвать команду:

```
log-off all
```

Перенаправление ssh-лога Putty в файл

При выполнении скриптов из командной строки, часто необходимо перенаправить вывод в файл. Для ssh-консоли Putty, это выполняется передачей параметра `-sessionlog {имя_файла.txt}`

```
"C:\Program Files\PuTTY\putty.exe" -ssh localhost -P 22 -l admin -pw admin -m shh_logger_
↪test_script.txt -sessionlog sessionlog.txt
```

33.1.8 FAQ

Зависает ssh-терминал на этапе подключения к серверу.

При запуске Putty из скрипта при первом подключении к серверу выдается сообщение PuTTY Security Alert, что может приводить к зависанию скрипта.

Необходимо дополнительно передать подтверждение при запуске:

```
echo yes | %plink% -ssh localhost -P 2222 -l admin -i ssh_admin_private.ppk
```

Примечание: Plink также входит в состав дистрибутива PuTTY.

33.2 WebSocket консоль сервера

33.2.1 Введение

Global 3 SE Server предоставляет консоль управления, доступную через WebSocket соединение.

С помощью команд передаваемых через WebSocket возможно:

- Перезагружать код прикладных приложений
- Выполнять Jexl-скрипты

33.2.2 Алгоритм работы с консолью

- Открытие WebSocket-соединения
- Отправка команды аутентификации пользователя в системе
- Отправка исполняемых команд
- Закрытие WebSocket-соединения

33.2.3 Открытие WebSocket соединения

Для окрытия WebSocket соединения с консолью сервера, необходимо выпонить http-запрос по адресу: `ws[s]://{server[:port]}/app/sys/ws/console`.

33.2.4 Формат команды

Командой является строка в формате:

`{command}[\n{arguments}]`

где:

`{command}` - строка команды. Может состоять из одного или нескольких слов.
`\n` - символ новой строки #10. Является разделителем команды и её аргументов. Не обязателен, если у команды нет аргуметтов.
`{arguments}` - строка аргументов команды. Может содержать любые символы, включая перносы строк.

33.2.5 Формат результата выполнения команды

Результатом выполения команды является строка в формате JSON:

```
{
  "success":true,
  "data": null,
  "exception":null,
  "exceptionStack":null
}
```

где:

`success` - Флаг успешности выполнения команды: `true|false`
`data` - Результат выполненения команды: `null|"string"|JSON`
`exception` - Сообщение возникшего исключения: `null|"string"`
`exceptionStack` - Стек возникшего исключения: `null|"string"`

33.2.6 Список команд

- `login\n`
  `{user}/{password}@{database}`

  Выполняет аутентификацию пользователя `user` в базе данных `database` и запускает рабочий сеанс пользователя.

  Использование команды `login`, в качестве способа уатентификации, было выбрано по причине невозможности использования http-заголовков некоторыми WebSocket-клиентами. Например: JMeter WebSocket Load Testing Sampler.

- `logout`

  Закрывает рабочий сеанс пользователя.

- `reload sbt`

  Выполняет перезагрузку прикладного кода текущего решения.

- `reload sbt force`

  Выполняет перезагрузку инфраструктуры EclipseLink, прикладного кода и общих библиотек текущего решения.

- `jexl\n`
  `// Произвольный текст Jexl-скрипта`
  `return 1 + 2;`

  Любое значение, возвращённое из Jexl-скрипта, будет преобразовано в строку.

33.2.7 Java пример взаимодействия с консолью

Библиотеки, используемые в примере клиента:

- «org.asynchttpclient» % «async-http-client» % «2.12.3»
- «com.fasterxml.jackson.core» % «jackson-databind» % «2.8.9»

```
package ru.bitec.app.examples.ws.console;

import org.asynchttpclient.AsyncHttpClient;
import org.asynchttpclient.Dsl;
import org.asynchttpclient.ws.WebSocket;
import org.asynchttpclient.ws.WebSocketListener;
import org.asynchttpclient.ws.WebSocketUpgradeHandler;

import com.fasterxml.jackson.databind.ObjectMapper;

import javax.websocket.CloseReason;
import java.io.Serializable;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.TimeUnit;

public class WsConsoleClientTest {

  public static void main(String[] args) throws Exception {
      try (WsConsoleClient client = WsConsoleClient.open("ws://localhost:8080/app/sys/ws/
↪console")) {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        client.login("admin", "admin", "pgtest");
        System.out.println("1 + 2 = " + client.evaluateJexl("return 1 + 2;"));
        client.reloadSbt(false);
        client.logout();
    }
}

public class WsConsoleClient implements AutoCloseable {

  public static WsConsoleClient open(String url) throws Exception {
      return open(url, 3000);
  }

  public static WsConsoleClient open(String url, int timeout) throws Exception {
      return new WsConsoleClient().connect(url, timeout);
  }

  private final ObjectMapper objectMapper_ = new ObjectMapper();
  private int commandTimeout_ = 60000;
  private AsyncHttpClient asyncHttpClient_;
  private WebSocket webSocketClient_;
  private volatile CompletableFuture<String> webSocketResultFuture_;
  private final WebSocketUpgradeHandler wsHandler = new WebSocketUpgradeHandler.
↪Builder().addWebSocketListener(new WebSocketListener() {
      @Override
      public void onOpen(WebSocket websocket) {
          // WebSocket connection opened
      }

      @Override
      public void onClose(WebSocket websocket, int code, String reason) {
          if (code != CloseReason.CloseCodes.NORMAL_CLOSURE.getCode()) {
              webSocketResultFuture_.completeExceptionally(new Exception(reason));
          }
      }

      @Override
      public void onError(Throwable t) {
          webSocketResultFuture_.completeExceptionally(t);
      }

      @Override
      public void onTextFrame(String payload, boolean finalFragment, int rsv) {
          webSocketResultFuture_.complete(payload);
      }
  }).build();

  public WsConsoleClient connect(String url, int timeout) throws Exception {
      AsyncHttpClient syncHttpClient = Dsl.asyncHttpClient();
      try {
          webSocketClient_ = syncHttpClient
                  .prepareGet(url)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                .setRequestTimeout(timeout)
                .execute(wsHandler)
                .get();
        } catch (Exception e) {
            syncHttpClient.close();
            throw e;
        }
        asyncHttpClient_ = syncHttpClient;
        return this;
    }

    public void close() throws Exception {
        if (asyncHttpClient_ != null) {
            try {
                if (webSocketClient_ != null && webSocketClient_.isOpen()) {
                    try {
                        webSocketClient_.sendCloseFrame().get();
                    } finally {
                        webSocketClient_ = null;
                    }
                }
            } finally {
                asyncHttpClient_.close();
            }
        }
    }

    public int getCommandTimeout() {
        return commandTimeout_;
    }

    public void setCommandTimeout(int commandTimeout) {
        this.commandTimeout_ = commandTimeout;
    }

    public void login(String user, String password, String database) throws Exception {
        sendCommand("login", String.format("%s/%s@%s", user, password, database));
    }

    public void logout() throws Exception {
        sendCommand("logout");
    }

    public String evaluateJexl(String script) throws Exception {
        return sendCommand("jexl", script);
    }

    public void reloadSbt(boolean force) throws Exception {
        sendCommand("reload sbt" + (force ? " force" : ""));
    }

    String sendCommand(String command) throws Exception {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        return sendCommand(command, null);
    }

    String sendCommand(String command, String args) throws Exception {
        validateConnection();
        if (command == null || command.trim().isEmpty()) {
            throw new Exception("Command can not be null or empty.");
        }
        webSocketResultFuture_ = new CompletableFuture<>();
        try {
            String payload = command + "\n" + (args != null ? args : "");
            webSocketClient_.sendTextFrame(payload).get();
        } catch (Exception e) {
            webSocketResultFuture_.completeExceptionally(e);
        }
        return parseResponse(webSocketResultFuture_.get(commandTimeout_, TimeUnit.
↪MILLISECONDS));
    }

    private String parseResponse(String response) throws Exception {
        ConsoleResponse consoleResponse = objectMapper_.readValue(response,␣
↪ConsoleResponse.class);
        if (consoleResponse.success) {
            return consoleResponse.data;
        } else {
            throw new Exception(consoleResponse.exception);
        }
    }

    private void validateConnection() throws Exception {
        if (webSocketClient_ == null) {
            throw new Exception("WebSocket connection is not connected.");
        }
        if (!webSocketClient_.isOpen()) {
            throw new Exception("WebSocket connection is closed");
        }
        CompletableFuture<String> webSocketResultFuture = webSocketResultFuture_;
        if (webSocketResultFuture != null && !webSocketResultFuture.isDone()) {
            throw new Exception("Prior console command is not completed. Wait for the␣
↪result of the previous command.");
        }
    }

    private final static class ConsoleResponse implements Serializable {

        private static final long serialVersionUID = -5577579081118070434L;
        /**
         * Флаг успешного выполнения запроса. false, если при обработке запроса возникло␣
↪исключение.
         */
        private boolean success = false;
        /**
```

(continues on next page)

(продолжение с предыдущей страницы)

```
         * Строковые данные
         */
        private String data;
        /**
         * Текст возникшего исключения
         */
        private String exception;
        /**
         * Стек возникшего исключения
         */
        private String exceptionStack;

        public boolean getSuccess() {
            return success;
        }

        public void setSuccess(boolean success) {
            this.success = success;
        }

        public String getData() {
            return data;
        }

        public void setData(String data) {
            this.data = data;
        }

        public String getException() {
            return exception;
        }

        public void setException(String exception) {
            this.exception = exception;
        }

        public String getExceptionStack() {
            return exceptionStack;
        }

        public void setExceptionStack(String exceptionStack) {
            this.exceptionStack = exceptionStack;
        }
    }
  }
}
```

33.3 Jexl через SOAP XML

33.3.1 Введение

В Global 3 SE Server реализован SOAP XML сервис, позволяющий выполнять Jexl-скрипт в контексте пользовательской сессии.

33.3.2 Аутентификация

Читайте в разделе Аутентификация в REST/SOAP-сервисах.

33.3.3 Схемы

WSDL описание сервиса доступно по адресу:

```
http://{server:port}/app/sys/soap/sys-service-1.0.0?wsdl
```

XSD схемы POJO запроса и ответа доступны по адресу:

```
http://{server:port}/app/sys/soap/sys-service-1.0.0?xsd=1
```

33.3.4 Передача бинарных данных с использованием MTOM

Сервис поддерживает оптимизированную передачу бинарных данных в сервис и обратно, через поля attachment объектов `JexlSoapRequest` и `JexlSoapResponse`.

33.3.5 Доступ к данным SOAP-запроса из прикладного кода

В прикладном коде Api-/Pkg-классов, выполняющемся в результате вызова Jexl-скрипта из SOAP-сервиса, доступен объект `SoapContext`. `SoapContext` предоставляет доступ к данным SOAP-запроса и SOAP-ответа. Для получения контекста, выполните:

```
val soapContextOpt = SoapContext()
```

Методы доступные в `SoapContext`:

- `hasInputAttachment: Boolean`
  Указывает на наличие у SOAP-запроса прикреплённых с использованием MTOM бинарных данных
- `getInputData: String`
  Возвращает строковые данные SOAP-запроса.
- `forInputStream(foo: InputStream => Unit): Unit`
  Предоставляет доступ к прикреплённым бинарным данным
- `setOutputData(data: String): Unit` \ Устанавливает строковые данные в SOAP-ответ.
- `forOutputStream(foo: OutputStream => Unit): Unit`
  Предоставляет доступ к выходному потоку бинарных данных передаваемых средствами MTOM

33.3.6 Методы обработки данных SOAP-запроса из Jexl-скрипта

В пакете `Gtk_SoapPkg` хранятся методы, использующие `SoapContext`:

- `setOutputData(value: AnyRef): Unit`
  Установливает вывод результата выполнения soap-запроса.
- `inputStreamToTemp: File`
  Создает временный файл с данными из прикреплённых данных soap-запроса
- `attachFile(file: File): Unit`
  Принимает файл, закачаивает его данные в выходной поток бинарных данных soap-запроса

  Пример ипользования из Jexl-скрипта:

```
var f = new("java.io.File", 'C:\users\userName\data.txt');
Gtk_SoapPkg.setOutputData('Данные в прикрепленном файле');
Gtk_SoapPkg.attachFile(f);
```

Данный скрипт, отправленный через SOAP-запрос, в ответ запишет

- в data - строку „Данные в прикрепленном файле“
- в attachment - закодированный base64 файл

33.4 REST сервис с обработкой http-запроса в прикладном пакете

33.4.1 Введение

В Global 3 SE Server реализован REST-сервис, позволяющий выполнять обработку HTTP-запроса в прикладном пакете, в контексте прикладной сессии.

Возможны 2 режима обработки запросов:

- Exclusive Session - с сохранением состояния между запросами. Rest- и Gtk-сессии создаются при первом обращении клинета к сервису и закрываются по таймауту или при явном указании на необходимость закрытия сессий по завершению запроса.
- Shared Session - без сохранения состояния между запросами. Каждый запрос обрабатывается в новых Rest- и Gtk-сессиях. Доступно с AS 1.14 RC7.

Пакет должен быть унаследован от одного из трейтов `RestPkg`, `RestESPkg`, `RestSSPkg`.

```
class Xxx_XxxPkg extends Pkg with RestPkg {
}
object Xxx_XxxPkg extends PkgFactory[Xxx_XxxPkg]{
}
```

33.4.2 Аутентификация

Читайте в разделе Аутентификация в REST/SOAP-сервисах.

33.4.3 Методы `RestPkg` / `RestESPkg` / `RestSSPkg`

- `get(relativePath: String): AnyRef`
  Вызывается при поступлении GET запроса.
  Допускается возврат: `null`, `None`, `String`, `ResponseBuilder`.
- `post(relativePath: String): AnyRef`
  Вызывается при поступлении POST запроса.
  Допускается возврат: `null`, `None`, `String`, `ResponseBuilder`.
- `onActivate()`
  Вызывается при подключении новой gtk-сессии
- `onDeactivate()`
  Вызывается при закрытии gtk-сессии
- `beforeReload(keyBundle: KeyBundle)`
  Вызывается перед перезагрузкой SBT
- `afterReload(keyBundle: KeyBundle)`
  Вызывается после перезагрузки SBT и пересоздании gtk-сессии
- `isSupportsSharedSession(): Boolean`
  Метод указывает, что пакет может обрабатывать запросы в режиме разделяемой GTK-сессии.
- `isSupportsExclusiveSession(): Boolean`
  Метод указывает, что пакет может обрабатывать запросы в режиме эксклюзивной GTK-сессии.

Допустимые результаты методов get() и post()

От типа возвращённого значения будет зависеть способ формирования ответа на Http-запрос.

- `null`,`None`
  Ответ будет сформирован через `RestfulContext().get.responseBuilder`
- `String`
  В ответ будет записано возвращённая строка
- `ResponseBuilder`
  Ответ будет сформирован через возвращенный `ResponseBuilder`. Он может быть равен `RestfulContext().get.responseBuilder`, либо сформирован произвольным образом.
- `Response`
  Ответом будет возвращенное значение.

33.4.4 Адреса

Доступно с AS 1.14 RC7:

- `http://{server:port}/app/sys/rest/es/pkg/{Xxx_XxxxPkg}/{relativePath}`

Доступно с AS 1.14 RC7:

- `http://{server:port}/app/sys/rest/ss/pkg/{Xxx_XxxxPkg}/{relativePath}`

где:

- `{Xxx_XxxxPkg}`
  Имя прикладного пакета, унаследованного от RestfulPkg
- `{relativePath}`
  Произвольный путь, который будет передан в методы get/post прикладного пакета
- `http://{server:port}`
  Адрес подключения к серверу
- `sys`
  Системный прикладной модуль, в будущем будет app/sys
- `rest`
  Шлюз для rest запросов
- `es` / `ss`
  Группировка по времени жизни gtk-сессии (Exclusive Session) / (Shared Session)
- `pkg`
  Узел для доступа к пакетам.

33.4.5 Рабочее пространство

`Workspace` - Рабочее пространство.

Используется для возможности параллельной работы нескольких gtk сессий в рамках одного пользователя в es режиме.

Для предотвращения неконтролируемого разрастания сессий, количество сессий на пользователя в эксклюзивном режиме ограничено. На одного пользователя и один workspace может существовать только одна сессия.

Рабочее пространство задается в http заголовке:

- `Workspace`
  Имя рабочего пространства пользователя.

33.4.6 Exclusive Session

Принцип работы сервиса

1. Получение HTTP-запроса (GET или POST)
2. Проверка авторизационных данных пользователя.
3. Получение, захват существующего рабочего сеанса или создание нового.
4. Вызов метода прикладного пакета `get(...)` или `post(...)`, соответственно
   В данном методе производится формирование тела http-ответа

5. Отправка ответа

Жизненный цикл http-сессий

При обращении к rest - сервису.

1. Если cookie `JSESSIONID` не задан, создается новая http сессия
2. Иначе, используется сессия с переданным идентификатором
   Если переданный id не корректный создается новая http сессия
3. Обработка запроса
4. Возврат результата
   При этом cookie `JSESSIONID` будет содержать идентификатор http сессии

Время жизни http-сессии 15 минут. При отсутствии обращений к http-сессии в течении этого интервала, сессия уничтожается, и установленные в её атрибуты значения становятся не доступны.

Жизненный цикл gtk-сессий

Gtk-сессия существует в разрезе 4-х параметров:

1. Алиас база данных
   Получается из http-заголовка `Database`. Если заголовок не передан, используется алиас из конфигурационного файла сервера.
2. Имя пользователя
   Получается из авторизационных данных. Передаётся в http-заголовке `Authorization`
3. Имя прикладного пакета
   Получается из строки адреса
4. Рабочее пространство
   Получается из http-заголовка `Workspace`. Если не задано, используется `Default`

Одной http-сессии (одному `JSESSIONID`) может соответствовать только одна gtk-сессия. При обращении к другому пакету из одной http-сессии, предыдущая gtk-сессия будет деактивирована и закрыта.

Таймаут gtk-сессии по умолчанию – 15 минут. Не используемая gtk-сессия будет закрыта через 15 минут. Изменить таймаут или закрыть сессию по завершению обработки запроса возможно через установку свойств `RestEXContext()`.

Создание новой сессии

Происходит в случае, если:

- в запрос не передан куки http сессии
- или gtk сессия не существует в уникальном разрезе
  - Имя пользователя
  - База данных
  - Workspace
  - Пакет

При этом:

- Создается gtk сессия
- Вызываются методы прикладного пакета
  - `onActivate`
  - в соответствии с типом http запроса вызывается один из методов обработки запроса
    * `get`
    * `post`
- В ответ добавляется кука http сессии

Работа в текущий сессии

Происходит в случае, если:

- в запрос передан куки http сессии
- и gtk сессия существует в уникальном разрезе, и ее куки совпадает с переданным запросом

Таймаут сессии

Происходит в случае, если:

- в запрос передан куки не существующий http сессии
- gtk сессии не существует

При этом

- Происходит создание новой сессии и вызов метода обработки (смотри создание)
- Возврат новой куки

Конфликт сессии

Происходит в случае, если:

- в запрос передан куки http сессии
- gtk сессия существует в уникальном разрезе, и ее текущий куки не совпадает с куки http сессии

При этом

- Генерируется ошибка захвата сессии

Захват сессии

Происходит при условии:

- в запрос не передан куки http сессии
- или передан устаревший куки, который совпадает с куки в gtk сессии
- gtk сессия существует в уникальном разрезе

При этом:

- Вызываются методы прикладного пакета

  - onDeactivate
  - onActivate
- в соответствии с типом http запроса вызывается один из методов обработки
- Возвращается куки новой сессии

Доступ к данным REST-запроса из прикладного кода

В прикладном коде Pkg-класса, выполняемом при REST-запросе, доступен объект `RestESContext`, предоставляющий доступ к данным REST-запроса и REST-ответа. Для получения контекста, выполните:

```
val restContextOpt = RestESContext()
```

Методы контекста:

- `sessionTimeout: Long`
  Время жизни gtk-сессии после последнего запроса
- `request: HttpServletRequest`
  Объект-запрос. Предоставляет доступ к параметрам, кукам и т.д.
- `responseBuilder: Response.ResponseBuilder`
  Билдер ответа.
- `markSessionClose(): Unit`
  Устанавливает флаг закрытия gtk-сессии по завершении обработки запроса
- `isSessionClose: Boolean`
  Возвращает значение флага закрытия gtk-сессии

33.4.7 Обработка ошибок

Коды ошибок

- 500 – server error
  Прикладное исключение
- 409 - conflict
  Конфликт сессий

Формат ошибок

При возникновении ошибки, по умолчанию, ответ будет возвращён в формате xml.

Для изменения формата ответа, передайте http-параметр `format`.

Доступные значения:

- `xml`
- `json`

Пример:

```
http://localhost:8080/app/sys/rest/es/pkg/Gs3_RestfulTestPkg/anypath?format=json
```

Xml ответ с ошибкой

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>\
<response >
  <status>%s</status>
  <error>
    <type>%s</type>
    <message>%s</message>
    <stacktrace><![CDATA[%s]]></stacktrace>
  </error>
</response>
```

Json ответ с ошибкой

```
{
"response": {
    "status" : 0,
    "data":{},
    "error": {
        "type": "" ,
        "message": "",
        "stacktrace": ""
    }
}}
```

33.4.8 Пример обращения к сервису

Java

```
package ru.bitec.app.sys.rest;

import org.junit.Test;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Invocation;
import javax.ws.rs.core.Cookie;
import javax.ws.rs.core.Response;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class PkgRestServiceTest0 {

    private static String rootAddress = "http://localhost:8080/sys/rest/es";
    private static Client client = ClientBuilder.newClient();
    private Path testPath = Paths.get(System.getProperty("user.dir"), "src\\test\\java\\
↪ru\\bitec\\app\\sys\\rest");
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    private Path cookiePath = testPath.resolve("JSESSIONID.cookie");


    @Test
    public void get_2_Cookie_test() throws Exception {
        String JSESSION_Cookie = load_JSESSIONID_Cookie();
        Invocation.Builder builder = client.target(rootAddress + "/pkg/Gs3_
↪RestfulTestPkg/anypath").request()
                //.header("Authorization", "Basic " + Base64.encodeToString("admin:admin
↪".getBytes("UTF-8"), false))
                .header("Authorization", "Basic " + java.util.Base64.getEncoder().
↪encodeToString("admin:admin".getBytes("UTF-8")))
                .cookie(Cookie.valueOf(JSESSION_Cookie));
        Response response = builder.get();
        try {
            save_JSESSIONID_Cookie(response);
            String s = response.readEntity(String.class);
            System.out.println("response: " + s);
        } finally {
            response.close();
        }
    }

    private void save_JSESSIONID_Cookie(Response response) throws IOException {
        if (response.getCookies().containsKey("JSESSIONID")) {
            String cookie = response.getCookies().get("JSESSIONID").toString();
            Files.write(cookiePath, cookie.getBytes(Charset.defaultCharset()));
        }
    }

    private String load_JSESSIONID_Cookie() throws IOException {
        if (Files.exists(cookiePath))
            return new String(Files.readAllBytes(cookiePath), Charset.defaultCharset());
        else
            return null;
    }

}
```

33.5 REST-сервис для взаимодействия с пользовательскими сессиями

Сервис доступен только в режиме разработки. В проектных решениях не доступен.

Функциональность сервиса:

1. Получение списка сессий, удовлетворяющих условиям поиска.
2. Выполнение Jexl-скриптов в контексте гл.выборки приложения.

33.5.1 Получение списка сессий

При отправке HTTP GET на адрес сервиса, будет возвращён JSON со списком сессий, удовлетворяющих условиям поиска. Http-запрос может содержать следующие параметры:

- `sbt`
  Фильтр по имени SBT
- `clientId`
  Фильтр по идентификатору клиента (этот идентификатор присваивается каждому экземпляру браузера и содержится в куках)
- `user`
  Фильтр по имени пользователя
- `app`
  Фильтр по имени главной выборки приложения (поиск осуществляется по вхождению переданного значения в полное имя гл.выбокрки)

Пример HTTP GET:

```
http://localhost:8080/app/sys/rest/sessions?sbt=test&user=admin&clientId=123456&app=Xxx_
↪xxxxxx
```

Пример ответа:

```
{"sessions":[{
  "sid": "E1",
  "id": "02b5f602-ac2c-4259-9452-18840a1cd124",
  "user": "admin",
  "clientId": "B54E2C1F-04E5-4BB2-9372-A9AB8E9C6676",
  "database": "PGTEST",
  "app": "gtk-ru.bitec.app.gs3.Gs3_TestXmlApplication"
}]}
```

33.5.2 Выполнение Jexl в контексте главной выборки приложения

Простой HTTP POST

При отправке HTTP POST на адрес сервиса, в контексте гл.выборки приложения сессии, с переданным идентификатором, будет выполнен Jexl-скрипт, переданный в теле Http POST.

Адреса сервисов:

- `http://{server:port}/app/sys/rest/sessions/{id}/jexl/mainsel`
  Где `{id}` - Идентификатор сессии. Значение `{id}` можно получить из результата http get к сервису.
- `http://{server:port}/app/sys/rest/sessions/new/jexl/mainsel?appname={имя_гл_выборки}`
  При этом будет запущена новая сессия. Для корректного запуска новой сессии и открытия приложения, необходимо передать имя главной выборки через http-параметр `appname`

Пример HTTP POST:

```
http://localhost:8080/app/sys/rest/sessions/949d77e9-80bf-4b93-bd3b-b424f8887783/jexl/
↪mainsel

http://localhost:8080/app/sys/rest/sessions/new/jexl/mainsel?appname=Gs3_
↪TestXmlApplication
```

(continues on next page)

(продолжение с предыдущей страницы)

В ответе вернётся JSON с результатом выполнения:

```
{
  "sessionId":"14849330-3196-4b33-80c5-caa370186720",
  "result":"[результат выполнения Jexl]"
}
```

Пример на Java

```
@Test
    public void post_test_1() throws Exception {
        Map<String, Object> rootMap = (Map<String, Object>) Json.
↪parse(doTestGET(rootAddress + "?sbt=test"));
        List<Map<String, Object>> sessions = (List<Map<String, Object>>) rootMap.get(
↪"sessions");
        if (sessions.isEmpty())
            throw new Exception("No one ESession opened.");
        Map<String, Object> session = sessions.get(0);
        String id = (String) session.get("id");
                String rootAddress = "http://localhost:8080/app/sys/rest/sessions";
        doTestPOST(rootAddress + "/" + id + "/jexl/mainsel", "Some Jexl Script");
    }

    public String doTestGET(String uri, String database) throws Exception {
        System.out.println("HTTP GET: " + uri);
        Invocation.Builder builder = client.target(uri).request()
                //.header("Authorization", "Basic " + Base64.encodeToString("admin:admin
↪".getBytes("UTF-8"), false))
                .header("Authorization", "Basic " + java.util.Base64.getEncoder().
↪encodeToString("admin:admin".getBytes("UTF-8")));
        if (database != null) {
            builder.header("Database", database);
        }
        String result = "";
        Response response = builder.get();
        try {
            System.out.println("response status = " + response.getStatus());
            System.out.println("response MediaType = " + response.getMediaType());
            if (response.getStatus() == 200) {
                ByteArrayOutputStream out = new ByteArrayOutputStream();
                try (InputStream inputStream = response.readEntity(InputStream.class)) {
                    StreamHelper.copyStream(inputStream, out);
                }
                byte[] bytes = out.toByteArray();
                System.out.println("response: " + bytes.length + " bytes");
                result = new String(bytes);
                //System.out.println("filename: " + response.getHeaderString("Content-
↪Disposition"));
            } else {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                result = response.readEntity(String.class);
            }
        } finally {
            response.close();
        }
        System.out.println(result);
        return result;
    }

    public String doTestPOST(String uri, String database, String body) throws Exception {
        System.out.println("HTTP POST: " + uri);
        Invocation.Builder builder = client.target(uri).request()
                //.header("Authorization", "Basic " + Base64.encodeToString("admin:admin
↪".getBytes("UTF-8"), false))
                .header("Authorization", "Basic " + java.util.Base64.getEncoder().
↪encodeToString("admin:admin".getBytes("UTF-8")));
        if (database != null) {
            builder.header("Database", database);
        }
        String result = "";
        Entity entity = Entity.entity(body, MediaType.TEXT_PLAIN_TYPE);
        Response response = builder.post(entity);
        try {
            System.out.println("response status = " + response.getStatus());
            System.out.println("response MediaType = " + response.getMediaType());
            if (response.getStatus() == 200) {
                ByteArrayOutputStream out = new ByteArrayOutputStream();
                try (InputStream inputStream = response.readEntity(InputStream.class)) {
                    StreamHelper.copyStream(inputStream, out);
                }
                byte[] bytes = out.toByteArray();
                System.out.println("response: " + bytes.length + " bytes");
                result = new String(bytes);
                //System.out.println("filename: " + response.getHeaderString("Content-
↪Disposition"));
            } else {
                result = response.readEntity(String.class);
            }
        } finally {
            response.close();
        }
        System.out.println(result);
        return result;
    }
```

Form HTTP POST

Альтернативным вариантом передачи Jexl скрипта, является отправка на адрес сервиса http формы, поле которой содержит Jexl-скрипт. Адреса сервисов:

- `http://{server:port}/app/sys/rest/form/sessions/{id}/jexl/mainsel`
- `http://{server:port}/app/sys/rest/form/sessions/new/jexl/mainsel?appname={имя_гл_выборки}`

В ответе вернётся команда перенаправления на страницу входа с идентификатором сессии в качестве http-параметра.

Пример HTML-страницы.

```
<html>
   <body onload="onLoadFunc()">
    <script>
  function onLoadFunc() {
   const form = document.createElement('form');
   form.method = 'POST';
   form.id = 'MultilinePostForm'
   form.action = 'http://localhost:8080/app/sys/rest/form/sessions/new/jexl/mainsel?
↪appname=Gs3_TestXmlApplication';
   const hiddenField = document.createElement('input');
   hiddenField.type = 'hidden';
   hiddenField.name = 'script';
   hiddenField.value = `Btk_ClassAvi.list().newForm().open();
   Gs3_RootTestAvi.list().newForm().open();`;
   form.appendChild(hiddenField);
   document.body.appendChild(form);
   form.submit();
  }
    </script>
   </body>
</html>
```

33.6 Сервис отчётов

Сервис предназначен для формирования печатных документов, средствами сервера Global 3, по запросу из внешних систем.

Для получения файла со сформированным отчётом, необходимо отправить HTTP GET по адресу сервиса. Адреса сервисов:

- `http://{server:port}/app/sys/rest/report/{name}`
- `http://{server:port}/app/sys/rest/report/{name}/{date}`

где:

- `{name}`
  Системное имя печатной формы

- `{date}`
  Дата версии отчёта

Пример:

```
http://localhost:8080/app/sys/rest/report/Rpt_JasperSimpleQuery/15.02.2019
```

33.6.1 Аутентификация

Читайте в разделе Аутентификация в REST/SOAP-сервисах.

33.6.2 Параметризация отчёта

Для передачи параметров в формируемый отчёт, передавайте значения через http-параметры. Служебные символы и пробелы должны бить заменены Escape-символами.

Пример:

```
http://localhost:8080/app/sys/rest/report/Rpt_JasperSimpleQuery?param1=value1&
↪param2=value2
```

33.7 Аутентификация в REST/SOAP сервисах

Сервисы используют HTTP-аутентификацию двух типов: Basic и Bearer.

33.7.1 Basic - с использованием логина, пароля и имени базы

При Basic-аутентификации, клиент должен передать в запросе:

- Имя пользователя
- Пароль
- Имя базы

Имя пользователя и пароль передаются через HTTP-заголовок:

- `Authorization`
  Значение: `Basic {Base64Cred}`
  где:
  `{Base64Cred}` – строка `user:password`, кодированная в Base64

Имя базы может быть передано через (в порядке приоритета):

- Сегмент строки адреса. Пример: `http://server/{DATABASE}/`
- HTTP-загловок `Database` со значением `{DATABASE}`.
- HTTP-параметр `Database` со значением `{DATABASE}`. Пример: `http://server/?Database={DATABASE}`

где:
`{DATABASE}` - имя базы данных.

Если имя базы не передано ни одним из описанных способов, будет произведена попытка аутентификации с использованием имени базы по-умолчанию. База по умолчанию определяется из конфигурационного файла `global3.config.xml` (в порядке приоритета).

- Значение атрибута `<databases defaultDb="{DATABASE}"/>`
- Значение атрибута `<database alias="{DATABASE}"/>` первой базы в списке конфигураций `<databases/>`

33.7.2 Bearer - с использованием токена аутентификации

При Bearer-аутентификации, клиент должен передать в запросе:

- Токен аутентификации

```
До версии AS 1.20 rc 15 включительно, доступны только токены сформрованные сервером␣
↪приложений на основе: имени базы, имени пользователя и пароля.
```

- Имя базы

```
До версии AS 1.20 rc 15 включительно, указания имени базы при Bearer-аутентификации␣
↪не требуется.
```

Запрос должен содержать HTTP-заголовок:

- `Authorization`
  Значение: `Bearer {Token}`
  Где:
  `{Token}` – ключ авторизации, присвоенный пользователю. Можно получить в прикладном коде от сущности

```
session.user.token: String
```

Имя базы может быть передано через (в порядке приоритета):

- Сегмент строки адреса. Пример: `http://server/{DATABASE}/`
- HTTP-загловок `Database` со значением `{DATABASE}`.
- HTTP-параметр `Database` со значением `{DATABASE}`. Пример: `http://server/?Database={DATABASE}`

где:
`{DATABASE}` - имя базы данных.

Если имя базы не передано ни одним из описанных способов, будет произведена попытка аутентификации с использованием имени базы по-умолчанию. База по умолчанию определяется из конфигурационного файла `global3.config.xml` (в порядке приоритета).

- Значение атрибута `<databases defaultDb="{DATABASE}"/>`
- Значение атрибута `<database alias="{DATABASE}"/>` первой базы в списке конфигураций `<databases/>`

33.7.3 Типы токенов аутентификации

- `ast` - *Application Server Token*
- `gjwt` - *Gtk Json Web Token*

Application Server Token

Токен формируется сервером приложений после входа в приложение Глобал с использованием имени пользователя и пароля. Токен возвращется клиенту в Cookie access_token, привязанной к подмножеству адресов `http[s]://server/{DATABASE}/`. Эта Cookie автоматически присоединяется ко всем запросам по адресам `http[s]://server/{DATABASE}/`, выполняемым из браузера, аутентифицированного в системе Глобал.

Токен устаревает после перезапуска сервера приложений.

Gtk Json Web Token

Токен, по стандарту JWT, формируется в модуле GTK (или на внешнем серсисе) и валидируется в модуле GTK.

GJWT разделяются на подтипы

- `UserHash` - *Долгоживущий токен пользователя*
- `UserCrt` - *Подписанный пользователем токен*
- `ProxyCrt` - *Подписанный прокси-пользователем токен*

Долгоживущий токен пользователя

Формируется администратором системы Глобал и сообщается пользователю любым удобным способом, исключающим утечку токена.
Токен, сопоставленный с учётной записью пользователя, хранится в БД решения.
Срок годности - определяется администратором.

Тело JWT (payload) должно содержать следующие поля:

- `typ` - `"UserHash"`
- `sub` - имя пользователя, под которым необходимо аутентифицироваться
- `exp` - дата устаревания

При проверке валидности токена, проверяется существование токена в списке сопоставленных с учётной записью токенов и дата устаревания.

Подписанный пользователем токен

Формируется пользователем и подписывается закрытым RSA-ключом пользователя.
Открытый RSA-ключ, сопоставленный с учётной записью пользователя, хранится в БД решения.
Срок годности - определяется пользователем.

Тело JWT (payload) должно содержать следующие поля:

- `typ` - `"UserCrt"`
- `sub` - имя пользователя, под которым необходимо аутентифицироваться
- `cid` - идентификатор открытого ключа в системе Глобал, сопоставленного с пользователем
- `exp` - дата устаревания

При проверке валидности токена, на основе полей `sub` и `cid` из БД решения получается открытый RSA-ключ и проверяется валидность подписи JWT.

Подписанный прокси-пользователем токен

```
Прокси-аутентификация - это аутентификация под именем пользователя ``№2`` от имени␣
↪пользователя ``№1``,
именуемого ``прокси-пользователем``.

Данный вид аутентификации может быть использован внешними планировщиками задач, которые␣
↪необходимо выполнять под разными пользователями.
При этом, планировщику нет необходимости хранить и передавать секретные учётные данные␣
↪пользователей.
```

Формируется прокси-пользователем и подписывается закрытым RSA-ключом прокси-пользователя.
Открытый RSA-ключ, сопоставленный с учётной записью прокси-пользователя, хранится в БД решения.
Срок годности - определяется прокси-пользователем.

Тело JWT (payload) должно содержать следующие поля:

- `typ` - `"ProxyCrt"`
- `sub` - имя пользователя, под которым необходимо аутентифицироваться
- `psub` - имя прокси-пользователя, которому принадлежитам закрытый и открыты ключи
- `cid` - идентификатор открытого ключа в системе Глобал, сопоставленного с прокси-пользователем
- `exp` - дата устаревания

При проверке валидности токена, на основе полей `psub` и `cid`, из БД решения получается открытый RSA-ключ прокси-пользователя и проверяется валидность подписи JWT.
Если токен валиден и пользователь `psub` имеет права на выполнение кода от имени других пользователей, запрос аутентифицируется под именем пользователя `sub`.

Пример запроса с аутентификацией по токену

В примере используются библиотеки:

- «io.jsonwebtoken» % «jjwt-api» % «0.10.8»,
- «io.jsonwebtoken» % «jjwt-impl» % «0.10.8»,
- «io.jsonwebtoken» % «jjwt-jackson» % «0.10.8»,
- «org.apache.httpcomponents» % «httpclient» % «4.5.8»

```
import io.jsonwebtoken.Jwts;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.junit.Assert;
import org.junit.Test;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.KeyPair;
import java.security.PrivateKey;
import java.util.Calendar;
import java.util.Date;
import java.util.stream.Collectors;

public class AuthProviderTest {

    private PrivateKey getPrivateKey(String proxyUser) throws NoSuchAlgorithmException {
        // Формируется рандомный закрытый ключ.
        // В рабочем коде необходимо использовать реальный закрытый ключ, принадлежащий␣
↪пользователю.
        return KeyPairGenerator.getInstance("RSA").generateKeyPair().getPrivate();
    }

    @Test
    public void authenticate_with_gjwt_proxy_cert() throws Exception {
        Calendar calendar = Calendar.getInstance();
        calendar.add(Calendar.DAY_OF_MONTH, 7);
        String proxyUser = "proxy_user";
        String gjwt = "gjwt_" + Jwts.builder()
                .setAudience("GS")                  // Кому предназначен токен.␣
↪Необязательный параметр.
                .setIssuer("Scheduler")             // Кем сформирован токен.␣
↪Необязательный параметр.
                .setSubject("real_user")            // Имя пользователя, под которым␣
↪необходимо аутентифицироваться.
                .claim("typ", "ProxyCrt")           // Подтип токена. В данном случае,␣
↪токен для прокси-аутентификации.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                .claim("psub", proxyUser)           // Имя прокси-пользователя.
                .claim("cid", "123456789")          // Идентификатор открытого RSA-ключа␣
↪прокси-пользователя в системе Глобал.
                .setExpiration(calendar.getTime())  // Дата устаревания токена.
                .signWith(getPrivateKey(proxyUser)) // Закрытый ключ прокси-пользователя,
↪ которым будет подписан токен.
                .compact();

        HttpGet httpGet = new HttpGet("http://localhost:8080/PGTEST/app/sys/rest/ss/pkg/
↪Gs3_RestfulTestPkg/anypath");
        httpGet.addHeader(HttpHeaders.AUTHORIZATION, "Bearer " + gjwt);
        try (CloseableHttpClient client = HttpClients.createDefault()) {
            try (CloseableHttpResponse response = client.execute(httpGet)) {
                try (InputStream in = response.getEntity().getContent()) {
                    String s = new BufferedReader(new InputStreamReader(in,␣
↪StandardCharsets.UTF_8))
                            .lines()
                            .collect(Collectors.joining(System.lineSeparator()));
                    System.out.println(s);
                }
            }
        }
    }
}
```

33.8 Администрирование Rest-сервисов

Класс Btk_AcPackage содержит информацию об администрируемых пакетах:

- Имя пакета
- id модуля, которому он принадлежит
- является ли он rest-пакетом (`bIsRest`)
- администрируются ли серверные полномочия (`bControlServerPriv`)

Пакет считается Rest-пакетом, если он наследуется от трейта RestPkg, в том числе от его потомков. Rest-пакеты регистрируются при обновлении администрируемых объектов по модулю. Если при обновлении окажется, что пакет больше не наследуется от RestPkg, то признак `bIsRest` в таблице будет снят.

Внимание: После изменения статуса администрируемости пакета, необходимо сбросить shared-кэш по классу Btk_AcPackage.

33.8.1 Выдача прав на вызов Rest-пакетов

В карточке роли на закладке «Права на Rest-пакеты» отображен список пакетов из класса Btk_AcPackage и имеет ли роль доступ к ним. Информация по доступу для роли хранится в таблице Btk_AcRolePackagePriv. Для выбора\снятия доступа роли к пакету используйте чекбокс Имеет доступ. Во время индексации прав пользователей по роли для всех Rest-пакетов, у которых стоит галка «Контролировать серверные полномочия», будут выданы объектные привилегии пользователям, которые имеют данный профиль. Привилегии регистрируются на адм. объект Btk_AcPackage.

Примечание: Супер-пользователь имеет полный доступ

33.9 Администрирование SOAP-вызовов

Администрирование SOAP-вызовов включается в Администратор \ Настройки > Настройки администрирования соответствующим чекбоксом. Для выдачи права роли необходимо дать доступ к объектной привилегии UseSoap адм. объекта Btk_ManagementPkg.

Примечание: Супер-пользователь имеет полный доступ

Часть VIII

Приложение

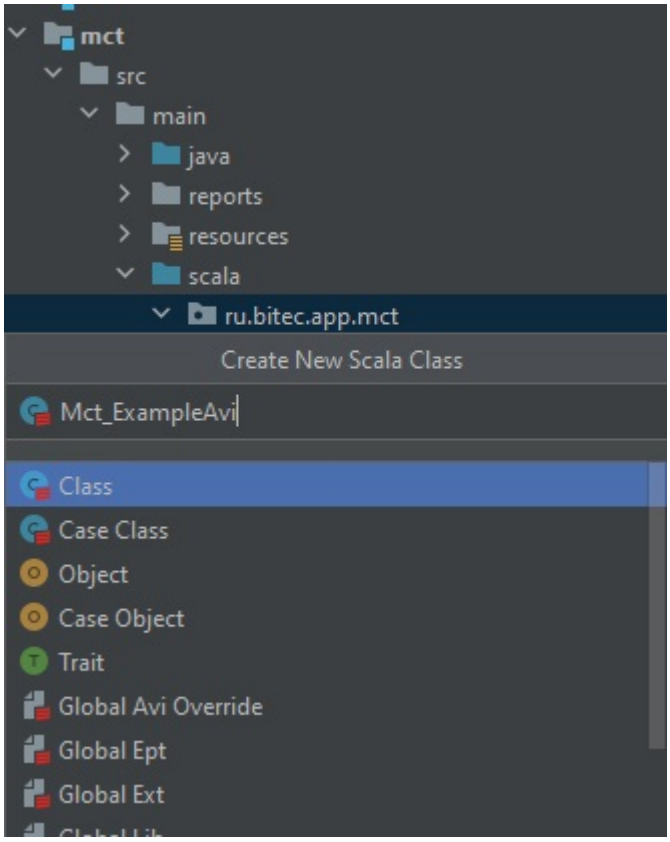
Глава 34

Практические советы

34.1 Создание выборки без класса

34.1.1 Первичное создание файла выборки

1. В нужной дирректории проекта (`имя_модуля/src/main/scala/ru/bitec/app/имя_модуля/...`) создайте Scala класс, задайте ему имя в формате `<имя модуля>_<имя выборки>Avi`, например `Mct_ExampleAvi`.



- Класс должен наследоваться от `EntityAvi`.
- В файле должен присутствовать объект-компаньон, с его помощью в дальнейшем будут создаваться объекты целевых выборок в системе.
- У класса должна присутствовать аннотация avm-файла, где указывается файл с разметкой выборок. *Как создать avm-файл*.

Общий вид файла должен быть следующим:

```
package ...

import ...

object Mct_ExampleAvi extends Mct_ExampleAvi

@AvmFile(name = "Mct_Example.avm.xml")
class Mct_ExampleAvi extends EntityAvi{
    Код выбороки
}
```

34.1.2 Создание трейта и методы получения данных, выводимых трейтом

1. В классе создается трейт, который в дальнейшем будет использован для открытия экземпляра выборки.

   Этот трейт должен наследоваться от одного или более базовых трейтов из `EntityAvi`, или от созданных вами, например:

```
trait List extends Default with super.List

trait MyList extends List
```

   Для каждого трейта должна быть создана функция-компаньон, причем если имя вашего трейта совпадает с именем базового трейта из `BaseAvi`(наследуется в `EntityAvi`) - `Card`, `List`, `Lookup`, то функцию компаньон надо переопределять, а если имя иное, то функцию нужно создать:

```
override def list(): List = {
    new List {
        override def meta = this
    }
}

def myList(): MyList = {
    new MyList {
        override def meta = this
    }
}
```

   - Имя функции-компаньона начинается с прописной буквы, имя трейта с заглавной.

2. Если трейт создается для отображения данных из базы, тогда необходимо определить метод получения данных для выборки. Для этого существует два подхода:

Реляционный запрос:

1. Базово определяется в методе `selectStatement`.
2. Тело метода является обычным sql запросом, помещенным в scala-строку.

```
override protected def selectStatement: String = {
    s"""
        |select
        |  st.id
        |  ,st.sCaption
        |  ,st.sCode
        |  ,ot.sDescription as otherDescription
        |from someTable st
        |join otherTable ot on st.someParam = ot.otherParam
        |""".stripMargin
}
```

3. Если итоговый набор данных требуется отфильтровать при помощи данных, *переданных извне* (обычно из фильтров и параметров выборок), и запрос имеет линейную структуру(нет вложенных запросов, временных таблиц и т.д., в которых требуется фильтрация *по внешним данным*), тогда:

- Переопределяем метод `onRefresh` следующим образом:

```
override protected def onRefresh: Recs = {
    prepareSelectStatement(s"&DefFltReferenceMacro and someStatement and $
↪{someScalaCode}")
}
```

- `DefFltReferenceMacro` - стандартное *имя макроса соответствующего трейта из разметки avm-файла*. Вы можете задать любое другое удобное вам имя макроса.
- `someStatement` - дополнительные условия в формате sql.
- `someScalaCode` - любой Scala-код, результатом работы которого является *булевое значение* или некие данные, которые подставляются в логическое выражение sql с одной из сторон:

```
s"....
... t.idState = ${getSelfVar("idState").asNLong}
...."
```

Объектный запрос:

1. Базово определяется в методе `onRefresh`.

Примечание: Метод onRefresh возвращает как строку в виде sql-запроса, так и набор Scala-объектов, определяющих набор данных выборки

---

2. В теле метода создается некоторая коллекция однотипных объектов:

```
override protected def onRefresh: Recs = {
    //***_SomeClassApi().load(id)
    //***_SomeClassApi().byParent(parentRop[или parentId])[.toList[.filter[Not][.
↪orderBy]]]
    //someTXIndex().byKey(TXIndexKey)[.toList[.filter[Not][.orderBy]]]
    new OQuery(***_SomeClassAta.Type) {
        where(t.someClassAttr === someScalaVAlue [and[or] ....])
    }
}
```

- При помощи метода load - если требуется получить единственный объект (обычно используется в карточных выборках).
- При помощи метода byParent - если класс является коллекцией, с передачей в него id или rop родительского объекта.
- При помощи транзакционного индекса по переданному ключу - если класс не является коллекцией, или если нужно получить элементы коллекции не в разрезе предка, а в разрезе иного(-ых) атрибута(-ов) класса.
- При помощи запроса OQuery.

Последние три подхода выбирайте на ваше усмотрение, в зависмиости от обстоятельств. Полученный в итоге набор rop класса можно финально отфильтровать при помощи методов filter[Not], отсортировать при помощи orderBy.

В отличие от реляционного запроса, предыдущие четыре подхода формируют набор данных в формате списка rop определенного класса и содержат только данные этого класса. Если требуется добавить данные, не принадлежащие классу, то:

1. Если нужно получить данные без обработки методами api, тогда можете пеоерпределить метод `onRefreshExt`:

```
override protected def onRefreshExt: String = {
    s"""
    with t as ( select
        :idParent as idParent
        :gid/*@NString*/ as gid
        .....
    )
    SELECT
        t1.sHeadline as idParentHL
        ,.....
    FROM t
    join parent_table t1 on t.idParent = t1.id....
    """
}
```

   - Во временной таблице t указываем, какие данные забираем из основного набора данных. Этот запрос выполняется для каждой отдельной rop-ы. Далее при помощи обычного sql запроса вытягиваем дополнительные данные.
   - Для gid атрибутов указываем аннотацию `/*@NString*/`, что бы парсер корректно передал данные с типом NGid.

2. Если нужно получить данные обработанные методами api:

   1. В текущем трейте или его предке создается case class с атрибутами, которые вам требуется добавить на выборки:

```
case class AdditionalInfo(
        var someIdParameter: NLong,
        var someIdParameterHL: NString,
        var someStrParameter: NString,
        var someDateParameter: NDate,
        var someBoolParameter: NNumber,
        var someNumParameter: NNumber
        )
```

      - Обычно класс называется AdditionalInfo, но вы можете выбрать и иное имя.
      - `someIdParameter` - ссылочный атрибут, добавляется для полноты данных выборки, *обычно является скрытым* (смотри пояснение свойства `editorType`, настройки `changeableAttr`), пользователю не виден.
      - `someIdParameterHL`[или `MC`] - заголовок ссылочного атрибута (`MC` для кодов/мнемокодов), данные ссылочного атрибута, отображаемые пользователю.
      - `someStrParameter` - иные строковые данные.
      - `someDateParameter` - дата.
      - `someBoolParameter` - булевые данные, обычно на выборку передаются в виде данных в формате NNumber: 0 - ложное значение, остальные - истина. Используются в атрибутах с редактором *check* (смотри пояснение свойства `editorType`).

- someNumParameter - иные числовые данные.

2. После создается метод getAdditionalInfo(rop):

```
def getAdditionalInfo(rop: ***_SomeClassApi#ApiRop): AdditionalInfo = {
    //Получение и обработка данных
    AdditionalInfo(
        someIdParameter = ...
        someIdParameterHL = ...
        ......................
        someNumParameter = ...
    )
}
```

3. Дополнительные данные добавляются в кортеж к изначальным данным выборки:

```
override protected def onRefresh: Recs = {
    val rop = ***_SomeClassApi().load(id)
    (rop, getAdditionalInfo(rop))
}
или
override protected def onRefresh: Recs = {
    ***_SomeClassApi().byParent(parentRop)//Или любые другие из трех␣
↪перечисленных методов получения набора данных класса
        .map(rop => (rop, getAdditionalInfo(rop)))
}
```

34.1.3 Создание трейта для получения данных от пользователя

Иногда требуется создать выборку, единственной целью которой является получение данных от пользователя. В таком случае:

1. Метод onRefresh выглядит следующим образом:

```
override protected def onRefresh: Recs = {
    "select 1 as id"
}
```

   Тк у выборки всегда должен присутствовать атрибут id для корректной работы

2. У выборки в avm-разметке *создаем фильтры*, которые будет заполнять пользователи.

После введения данных пользователем существует несколько вариантов их обработки:

- Пользователь закрывает выборку

  1. Если введенные пользователем данные не требуется возвращать в метод, вызвавший открытие текущей формы, то на закрытие выборки по согласию пользователя срабатывает метод closeFormOk(), соответственно в его переопределении в вашем трейте можно обработать введенные данные.
  2. Если данные требуется вернуть в метод, вызвавший открытие текущей формы, то на закрытие выборки переопределяем метод beforeCloseForm:

```
override def beforeCloseForm(event: BeforeCloseFormEvent): Unit = {
    event.lookupResultObject = someObjectWithUserData
    super.beforeCloseForm(event)
}
```

  - Где `someObjectWithUserData` - некая структура, хранящая пользовательские данные. Может быть List-ом, кортежем, case class-ом и т.д. на ваш выбор.
  - Получить такие данные можно в методе, вызвавшем открытие выборки, определив тип возвращаемых данных:

```
val data = Mct_ExampleAvi.myList().newForm().params(Map(...)).
↪results(List(attributesName)).openLookup()
val res = data.getResultObject.asInstanceOf[someObjectWithUserData]
```

Примечание: Здесь `someObjectWithUserData` - либо тот же самый тип, если есть возможность типизировать возвращаемое значение тем же классом, либо новая структура, схожая по строению, если, например, в `lookupResultObject` сохранен объект case class-а, недоступного в месте вызова выборки, тогда достаточно типизировать этот объект кортежем(или, например, листом, если в `lookupResultObject` записан лист), у которого вложенные типы следуют в том же порядке, что и у case class-а:

saved case class example(NLong, Boolean, List[NString]) -> Tuple3(NLong, Boolean, List[NString])

saved List[(case class example, NLong, ...)] -> List[(Tuple3(NLong, Boolean, List[NString]), NLong, ...)]

- Пользователь вызывает кастомную операцию на текущей выборке:
  - В этом случае требуется создать данную операцию, если ее еще нет, в текущем трейте или его предках, либо же переопределить такую операцию и забирать введенные пользователем данные напрямую из выборки при помощи методов `getVar("...")`/`getSelfVar("...")` и обрабатывать любым удобным вам способом.

34.1.4 Комбинированный подход

- Вы можете использовать комбинацию предыдущих пунктов в любом удобном вам виде - *выводить* какие-либо требуемые вам данные и *получать* от пользователя введенные им данные.

34.2 Создание avm-файла для выборки без класса

После того, как вами была *определена* логика работы вашей выборки, определены данные, с которыми выборка будет работать, требуется произвести разметку отображения.

34.2.1 Первичное создание avm-файла

- После того, как вы добавили Avi-файл выборки в определенное место проекта: имя_модуля/src/main/scala/ru/bitec/app/имя_модуля/...

  Файл с разметкой необходимо добавить в каталог имеющий идентичный путь:

  имя_модуля/src/main/resources/ru/bitec/app/имя_модуля/...

  Внимание:
  Структуры каталогов, после ../ru/bitec/app/имя_модуля/ должны полностью совпадать

- Имя avm-файла обычно выбирают такое, же как и у файла выборки. Вместо суффикса Avi используем суффикс .avm, например Mct_Example.avm.xml
- После создания файла добавляем туда начальный код:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<view xmlns="http://www.global-system.ru/xsd/global3-view-1.0" name="Mct_Example">

</view>
```

  Свойство name тега *view* - имя выборки без суффиксов.

34.2.2 Создание разметки для соответствующего отображения

После создания трейта в Avi-файле выборки создаем тег representation с таким же именем:

```
trait MyList extends List
```

```
<representation name="representationName" caption="representationCaption">
    <filter name="globalFilterName">
        <macros name="macrosName">
            <condition id="condName" logicalOperator="logicalOperator" attr="condAttr"␣
↪operator="condOperator" isExpression="true\false" expression="someSqlExpression">
                <filterAttr attribute-type="fltAttrType" name="fltName" editorType=
↪"fltEditorType" order="fltAttrOrder" caption="fltCaption" defaultValue="defVal"␣
↪isLastInLine="true\false">
                    <editor>
                        <...>
                    </editor>
                </filterAttr>
            </condition>
        </macros>
        <layout>
            <hBox>
                <vBox>
                    <attr name="fltName"/>
                    <vGroup>
                        <hBox>
                            <attr name="fltName"/>
                            <vBox>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                <attr name="fltName"/>
                                <attr name="fltName"/>
                            </vBox>
                        </hBox>
                    </vGroup>
                </vBox>
            </hBox>
        </layout>
    </filter>
    <layout>
        <composerType>
            <frame filter.isVisible="true\false" toolBar.isVisible="true\false" caption=
↪"caption" description="description">
                <frameType>
                    <layout>
                        <...>
                    </layout>
                </frameType>
            </frame>
            <composerSettings/>
        </composerType>
    </layout>
    <bandGroups>
        <bandGroup name="bandGroupName" caption="Группа"/>
    </bandGroups>
    <attributes>
        <attr name="attrName" order="attrOrder" caption="attrCaption" editorType=
↪"attrEditorType" isVisible="true\false" bandGroup="attrBandGroupName" isReadOnly="true\
↪false" isRequired="true\false" isLastInLine="true\false"/>
    </attributes>
    <operations>
        <oper name="operName" order="operOrder" caption="operCaption" isActive="true\
↪false"/>
    </operations>
</representation>
```

Разберем код разметки:

Основной тег representation

- Содержит два основных свойства:

`name`

name = «representationName» - где representationName - имя отображения, такое же как у трейта созданного вами ранее, например *MyList*

`caption`

caption = «representationCaption» - где representationCaption описание отображения, которое увидит пользователь в шапке отображения.

- Внутри тега `representation` содержатся несколько тегов -
  - Тег *filter* .
  - Тег *layout* .

Тег filter

- Содержит одно основное свойство `name` = «globalFilterName» - Системное имя фильтра.
- Внутри тега `filter` присутствуют два тега разметки -
  - Тег *macros*
  - Тег *layout*

Тег macros

Обязательный

- Содержит одно свойство: `name` = «macrosName» , например `DefFltReferenceMacro`, нужен системе, для дальнейшей подстановки данных из фильтра при обработке *selectStatement-запроса*
- Внутри можно определить несколько тегов `condition`

Тег condition

- Содержит несколько свойств:
  1. `id` - системное имя тега, используется в обработке метаданных выборки. Обязателен для заполнения. Должен быть уникальным в разрезе макроса.

     Обязателен

  2. `logicalOperator` - логический оператор(`and`\`or`). С помощью указанного оператора текущий `condition` будет добавлен к остальным условиям в `selectStatement-запроса`.

     Обязателен

  3. `isExpression` = «true\false» - свойство тега, определяющее каким образом будет обработан фильтр.

     Обязателен

     Так же от его значения зависит обязательность указания дальнейших свойств фильтра -
     - Если true:

* `expression` - условие. **Становится обязательным**.

  По сути представляет из себя фрагмент sql условия, по которому происходит фильтрация:

```
(:filter$flt_bShowNotUsed = 1 or
    (coalesce(t.bnotactive,0) = 0
        and (t.dexpirydate is null or t.dexpirydate > current_date)
    )
)
```

```
1 = :filter$flt_bShowNotUsed
```

  Внимание: Условие обязано быть заключено в скобки, если оно состоит из более чем одного логического элемента, иначе фильтрация итогового набора данных будет некорректна!

– Если false:

  Тогда заполняем следующие, **ставшие обязательными**, атрибуты -

  1. `attr` = «condAttr» - атрибут, по которому происходит фильтрация.

     Если в *selectStatement-запросе* присутствуют несколько таблиц, тогда в качестве condAttr можно указать любой атрибут этих таблиц, используя соответствующий псевдоним. Например:

```
attr="ot.someAttribute"
```

  2. `operator` = «condOperator» - оператор условия стандартного фильтра. Такой как:

```
operator`="equals"/"not equals"/"less"/"greater"/"greater or equals"/
↪"less or equals"/"in"/"between"/"like"/"is null"
```

     По этому значению соответствующим образом будет проведено сравнение `attr` и значения, переданного из фильтра.

- Внутри тега может содержаться несколько тегов `filterAttr`

Тег filterAttr

- Содержит несколько свойств:

  1. `name` - системное имя фильтра. Позволяет получить значение фильтра как в scala-коде выборки, так и внутри sql-запросов, вызываемых из scala-кода выборок. Например:

```
...
val bVal = getVar("flt_bShowNotUsed").asNNumber.toBoolean
...
```

```
s"""
    select
        ...
    from ...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    where ...
    logicalOperator (:filter$flt_bShowNotUsed = 1 or
        (coalesce(t.bnotactive,0) = 0
            and (t.dexpirydate is null or t.dexpirydate > current_date)
        )
    )
"""
```

2. `attribute-type` - тип атрибута стандартного фильтра:
   - `Varchar`
   - `Number`
   - `Long`
   - `Date`
3. `editorType` - определяет тип редактора, который будет отображаться при редактировании ячейки списка, дерева или отображаться в карточке:
   - `edit` - Редактор в строке
   - `editButton` - Редактор в строке с кнопкой
   - `currency` - Денежный редактор
   - `lookup` - Выпадающий список по запросу
   - `combo` - Фиксированный выпадающий список
   - `check` - Чекбокс
   - `datePick` - Редактор даты
   - `dateTimePick` - Редактор даты и времени
   - `timePick` - Редактор времени
   - `memo` - Memo - поле
   - `icon` - Изображение
   - `calendar` - Календарь
   - `buttonsEdit` - Редактор в строке с произвольными кнопками
   - `imageCollection` - Список картинок
   - `tagLookup` - Выпадающий список с мультивыбором
   - `editPassword` - Редактор пароля в строке
   - `button` - Кнопка
   - Если `editorType` = buttonsEdit, тогда внутри этого тега надо указать еще два тега `editor` и `ref`:

```
<editor>
    <buttonsEdit canEditText="true\false" changeableAttr="someAttr">
        <buttons lookup="true\false" openCard="true\false" reset="true\false
↪"/>
    </buttonsEdit>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
</editor>
<ref class="someClass"/>
```

Где:

* `canEditText` - по умолчанию, в поля ввода типа «Редактор скнопкой» и «Редактор с произвольными кнопками» вводить текст запрещено. Установив данное свойство в True, возможно разрешить ввод текста в поле ввода.
* `changeableAttr` - свойство содержит имя атрибута, в который будет записан идентификатор ссылочного объекта, при редактировании ссылочного поля. Каждому ссылочному атрибуту класса соответсвуют 2 атрибута выборки: значимый и отображаемый. Первый содержит идентификатор, а второй заголовок ссылочного объекта, который понятен пользователю. Значимый атрибут обычно является скрытым, а пользователь редактирует отображемый атрибут, при этом необходимо, что бы редактор отображаемого атрибута знал имя значимого атрибута.
* `class` - свойство содержит системное имя класса, на который ссылается данный атрибут. Это свойсво используется фильтром, для открытия формы выбора значения ссылочного атрибута, при формировании условия фильтрации.

4. `order` - порядковый номер. С его помощью можно настроить положение фильтров относительно друг друга.
5. `isLastInLine` - свойство, отвечающее за перенос последующих фильтров на новую строку.
6. `caption` - видимое имя фильтра.
7. `defaultValue` - умолчательное значение фильтра.

Тег filter layout

Необязательный тег. Служит для более детальной настройки расположения фильтров на выборке.

- На первом уровне, внутри тега `layout` должны располагаться следующие теги:
  - `vSection` - секция фильтров
  - `vGroup` - группа фильтров
  - `vBox` - вертикальный контейнер
  - `hBox` - горизонтальный контейнер
- На втором и последующих уровнях вложенности могут быть расположены те же теги, а так же тег `attr`, у которого есть два базовых свойства:
  - `name` - системное имя *фильтра*.
  - `caption` - отображаемое имя фильтра, перекрывает видимое имя фильтра, указанное в теге *filterAttr*.
- Комбинируя данные теги, вы можете гибко настроить фильтры выборки.

Тег representation layout

- Содержит внутри себя один тег:
  - `simpleComposer` - отображается только эта выборка, без подчиненных.
  - `tabComposer` - подчиненные выборки будут отображаться закладками.

    Вложенный тег:

```
<tabItems isVisible="false">
    <tabItem selection="gtk-Mct_ExampleAvi" representation="MyList" caption=
↪"Имя закладки, которое увидит пользователь"/>
    <tabItem.../>
    ...
</tabItems>
```

  - `dynamicComposer` - подчиненные выборки будут отображаться как отдельная часть формы.

    Вложенный тег:

```
<dynamicItems masterAlign="top/bottom/left/right/client">
    <dynamicItem selection="gtk-Mct_ExampleAvi" representation="MyList" align=".
↪."/>
</dynamicItems>
```

    * `masterAlign` - выравнивание главного фрейма. `client` - фрейм занимает всё доступное пространство компоновщика, не занятое фреймами с остальными типами выравнивания.
    * `align` - выравнивание детальной выборки.
  - `tabDynamicComposer` - сочетает в себе функционал tabComposer и dynamicComposer.
  - `tabDynDetComposer` - подчиненная выборка будет определятся автоматически, в зависимости от параметров этой выборки. Так же включает в себя функционал tabComposer.
  - Внутри любого из перечисленных тегов находится один обязательный тег:
    * `frame`

      Основные свойства:

      · `filter.isVisible` - будет ли видна панель фильтров на выборке.

      · `toolBar.isVisible` - будет ли виден тулбар с операциями на выборке.

      · `caption` - наименование фрейма. Отображается в первой строке заголовка фрейма или в заголовке формы, если фрейм является главным на форме

      · `description` - описание. Отображается второй строкой на расширенном заголовке фрейма.

      В этом теге, в свою очередь, устанавливает тег с указанием типа фрейма:

      · `grid` - данные отображаются списком.

      · `tree` - данные отображаются в виде дерева.

      · `card` - данные отображаются карточкой.

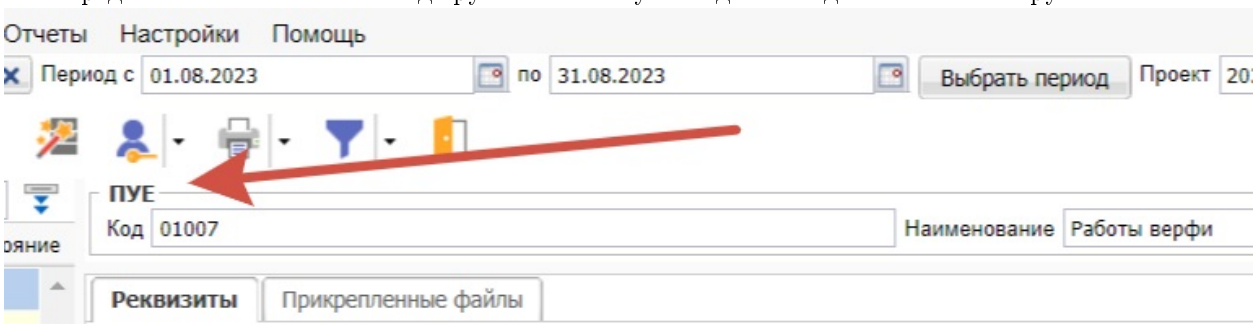
        В случае выбора карточного типа, внутри текущего тега можно установить тег `layout`, полностью аналогичный тегу *layout* фильтра.

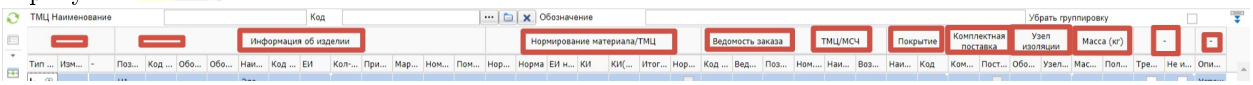
Тег representation bandGroups

Нужен для определения банд-групп. Содержит в себе набор тегов `bandGroup`:

```
<bandGroup name="bandName" caption="Имя банд-группы"/>
```

В дальнейшем определенные в этом теге банд-группы используются для объединения в этих группах атрибутов.





Тег representation attributes

Содержит в себе набор тегов `attr` со следующими свойствами:

- `name` - системное имя атрибута.
- `order` - свойство управляет порядковым номером колонки в списке/дереве и порядковым номером поля ввода в карточке.
- `caption` - Наименование атрибута.
- `editorType` - аналогичен *editorType* фильтра.
- `bandGroup` - свойство определяет группу объединения, в которую входит атрибут.
- `isVisible` - свойство видимости атрибута.
- `isReadOnly` - свойство редактируемости атрибута.
- `isRequired` - свойство обязательности заполнения атрибута.
- `isLastInLine` - аналогичен *isLastInLine* фильтра.

Тег representation operations

Нужен для первичной настройки операций, отображаемых в виде кнопок на тулбаре и в контекстном меню. В основном используется для того, что бы скрыть или отключить нужную операцию.

Содержит в себе набор тегов `oper` со следующими базовыми свойствами:

- `name` - системное имя операции. Имя должно совпадать с именем scala-метода (без учёта регистра) в Rep-классе, соответствующем отображению выборки.
- `order` - порядковый номер операции. Свойство управляет положением операции на панели управления, в меню, и прочих контролах управления. По умолчанию равно 0. Если значения совпадают, порядок операций будет соответствовать порядку в XML.
- `caption` - наименование операции. Отображается в меню и подменю.

- `isActive` - флаг управляет активностью операции. Если флаг снят, операция не загружается в клиент и, соответственно, не отображается на панели управления, в меню и т.д.

34.3 Проектное переопределение

34.3.1 Предисловие

Стоит отдельно указать, что помимо проектного переопределения существует так же опция наследования базовых файлов в проектных модулях. Для того что бы выбрать нужный подход, следует понимать что требуется сделать:

- Переопределить базовую логику так, что бы изменилось поведение этой логики в любом месте системы.

  Например, если вы хотите во всей системе изменить выборку Bs_GoodsAvi#Card, добавить туда проектные атрибуты, изменить разметку выборки итд, и при этом вы хотите видеть эти изменения абсолютно во всех местах где вызывается эта выборка(аналогично с переопределением api и иных файлов).

- Дополнить базовую логику работы и использовать её локально, в определенных ситуациях.

  Например, если вы хотите дополнить вкладку Bs_GoodsAvi#Card проектными атрибутами, изменить её разметку итд, при этом измененную вкладку вы хотите видеть в каком то конкретном отображении, а в остальной системе оставить вид этой выборки базовым(аналогично с переопределением api и иных файлов).

В зависимости от этого и выбирается сценарий дополнения базовой логики:

- *Наследование базовой логики*
- *Проектное переопределение*

34.4 Наследование базовой логики

Обычное наследование функционала. Реализуется с помощью ключевого слова `extends`. Определимся только с установленными практиками наименования и расположения наследующих файлов. В остальном работают обычные правила наследования. Рассмотрим пример наследования от Rpl_QueryAvi, и создание нового трейта с измененной логикой.

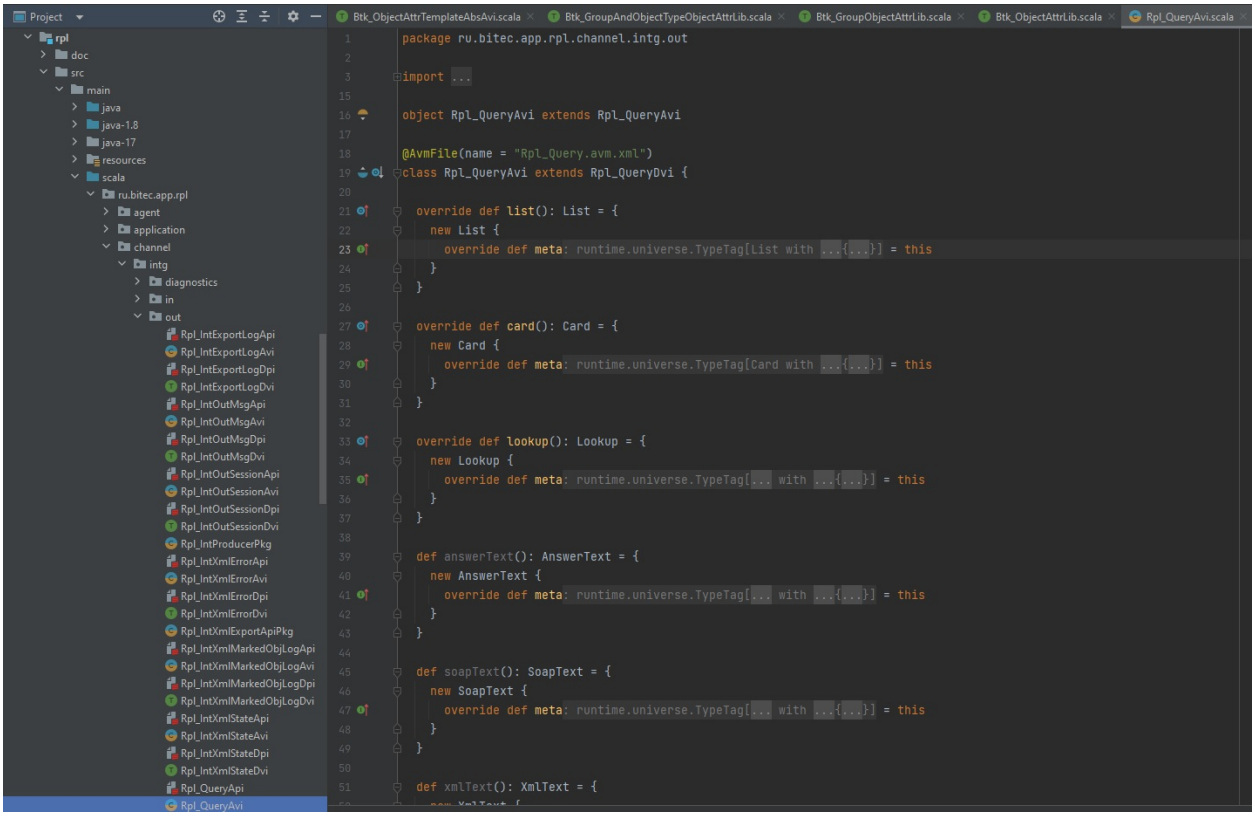
34.4.1 Расположение наследующего файла

Наследующий файл в вашем проекте должен располагаться в соответствующем модуле для проектных переопределений. Так же желательно повторять структуру директорий базового модуля. Так, если `Rpl_QueryAvi` расположен по следующему пути:

```
ru.bitec.app.rpl.channel.intg.out.Rpl_QueryAvi,
```

то и наследующий его файл рекомендуется располагать в

```
ru.bitec.app.<Имя проектного модуля переопределений>.channel.intg.out.<Имя наследующего␣
↪файла>
```



34.4.2 Наименование наследующего файла

При наследовании базового файла в ваш имя наследующего файла рекомендуется формировать следующим образом:

<Имя проектного модуля переопределений>_<Имя базового класса БЕЗ имени базового модуля> Так например, для `Rpl_QueryAvi` имя наследующего файла будет выглядеть так - `<Имя проектного модуля переопределений>_QueryAvi`.

Пример наследующего файла

```
package ru.bitec.app.<Имя проектного модуля переопределений>.channel.intg.out

import ru.bitec.app.rpl.channel.intg.out.Rpl_QueryAvi
import ru.bitec.gtk.core.AvmFile

@AvmFile(name = "Файл разметки.avm.xml")
class <Имя проектного модуля переопределений>_QueryAvi extends Rpl_QueryAvi{

    override def card(): Card = {
        new Card {
            override def meta = this
        }
    }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        trait Card extends Default with super.Card {
            ...
        }

    }
```

В случае наследования вы получаете возможность дополнить базовую логику, при этом не изменив поведение соответствующих выборок ранее созданных и выведенных в вашей системе.

34.5 Проектные переопределения

Рассмотрим проектное переопределение на примере класса. Проектное переопределение *выборок без класса*, pkg и lib файлов происходит аналогично.

34.5.1 Проектное переопределение файлов окружения класса

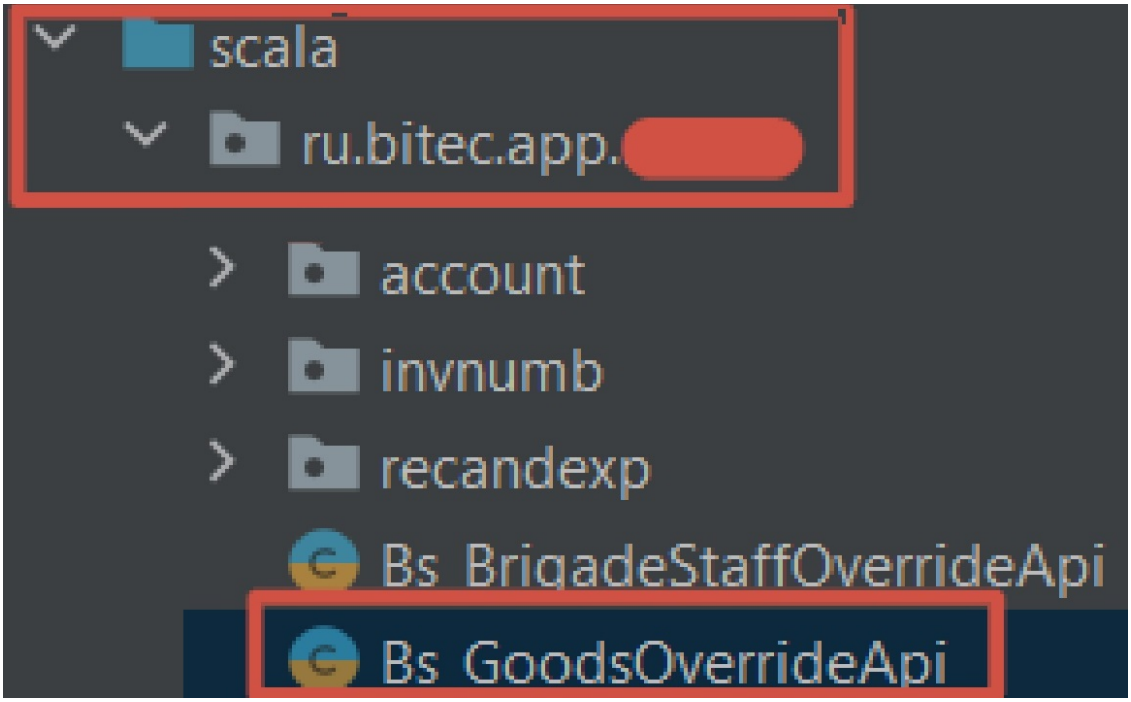
- Часто возникает необходимость переопределить или дополнить функционал определенного класса, в таком случае переопределяются нужные элементы *окружения класса*. Переопределяются обычно Avi, Api и avm.xml файлы окружения класса.

Переопределение Api

1. Внутри проектного модуля `.../src/main/scala/ru.bitec.app.проектный_модуль.<Если файл вложен в поддиректории, то нужно повторить их структуру>` создайте api-файл.
    - Имя файла задавайте по следующему шаблону <Имя Класса С ИМЕНЕМ БАЗОВОГО МОДУЛЯ ЭТОГО ДЛЯ КЛАССА>OverrideApi.

Примечание: Обычно имена классов и файлов получаются с использованием имени модуля и имени класса `***_ИмяКласса`, однако для переопределяемых файлов не рекомендуется использовать такой подход. Лучше модифицировать имя базового файла. Например: `Bs_GoodsApi` -> `Bs_GoodsOverrideApi`.

Внимание: ВАЖНО! При составлении имени файла НЕ используйте имя текущего модуля! В имя файла добавляется только суффикс `Override`!!!



2. Содержание файла должно быть следующим:

```
package ru.bitec.app.<имя проектного модуля>.bs

import ...

class Bs_GoodsOverrideApi extends Bs_GoodsApi {
    Код класса
}

object Bs_GoodsOverrideApi extends ApiFactory[
    java.lang.Long,
    Bs_GoodsAro,
    Bs_GoodsOverrideApi
]
```

3. Внутри тела класса можете переопределять существующие методы и добавлять новые.

Переопределение Avi

1. В том же пакете что и *api*: `.../src/main/scala/ru.bitec.app.проектный_модуль/bs.<Если файл вложен в поддиректории, то нужно повторить их структуру>`, создайте Avi-файл:

   - Имя файла получаем аналогично имени api-файла.

     Например: `Bs_GoodsAvi -> Bs_GoodsOverrideAvi`.

```
package ru.bitec.app.<имя проектного модуля>.bs

import ...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
object Bs_GoodsOverrideAvi extends Bs_GoodsOverrideAvi

@AvmFile(name = "Bs_GoodsOverride.avm.xml")
class Bs_GoodsOverrideAvi extends Bs_GoodsAvi {
    Код выборок
}
```

> **Внимание:** ВАЖНО! При составлении имени файла НЕ используйте имя текущего модуля! В имя файла добавляется только суффикс `Override`!!!

- Общая структура файла аналогично обычному Avi-файлу.
- Объект-компаньон должен иметь имя как у переопределяемого файла.
- Аннотация `@AvmFile(name = "Bs_GoodsOverride.avm.xml")` не обязательна. Если ее не указывать, то будет взята настройка базового файла.

2. Внутри тела выборки можете переопределять существующие трейты и их методы, а так же создавать новые трейты и методы(в т.ч. внутри переопределенных трейтов).

Переопределение Avm

1. Рекомендуется создавать avm-файл в дирректории `.../src/main/resources/ru.bitec.app.проектный_модуль.<Если файл вложен в поддиректории, то нужно повторить их структуру>`, и уже в нем создавайте файл с разметкой.

- Таким образом структуры каталогов в разных ветках будут идентичны.

  Например:

  `.../src/main/`**`scala`**`/ru.bitec.app.проектный_модуль/bs` для проектной логики и

  `.../src/main/`**`resources`**`/ru.bitec.app.проектный_модуль/bs` для проектной разметки.

1. Базовая структура полностью аналогична обычным amv-файлам.

   - В теге `view` указываем имя класса и дополнительное свойство `th:extends`:

```
<view xmlns="http://www.global-system.ru/xsd/global3-view-1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.global-system.ru/xsd/global3-view-1.0"
xmlns:th="http://www.global-system.ru/xsd/global3-view-template-1.0"
name="Bs_GoodsOverride" th:extends="Bs_Goods.avm.xml">
```

   - `name` - имя переопределенного файла, получаем аналогично прошлым пунктам.

     > **Внимание:** ВАЖНО! При составлении имени файла НЕ используйте имя текущего модуля! В имя файла добавляется только суффикс `Override`!!!

   - `th:extends` - базовый avm-файл, нужен для того, что бы в переопределенном файле указывать только проектную разметку. Остальные настройки будут вытягиваться из базового файла.

2. Если вы переопределили Avi-файл и создали там новый трейт, или переопределили существующий трейт Avi-файла, изменив возвращаемые данные(добавили новые атрибуты выборки, переименовали старые итд) тогда в avm нужно создать тег `representation` с именем, соответствующим имени этого трейта, и настроить разметку.

3. Так же, если вам требуется проектно переопределить только разметку, нужно помимо этого переопределить Avi-файл и указать переопределенный файл разметки в аннотации.

> **Внимание:** Включение переопределенных файлов разметки отличается от включения переопределенной логики `Api` и `Avi` ***!!!*** Что бы подключить переопределенную разметку, нужно в соответствующем Avi-файле для класса использовать аннотацию *@AvmFile(name = «Bs_GoodsOverride.avm.xml»)*

Включение проектной логики

- После того, как вы создали необходимые вам переопределения проектной логики, требуется связать переопределяемый и переопределяющий файлы.
- Для этого в проектном модуле, в каталоге `/src/main/resources/META-INF` откройте или создайте файл `overrides.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<overrides>
    <replace-with class="ru.bitec.app.<Имя проектного модуля>.<Имя модуля, которому␣
↪принадлежит переопределяемый файл>.<Имя переопределенного файла>">
        <when-type-is class="ru.bitec.app.<Имя модуля, которому принадлежит␣
↪переопределяемый файл>.<Имя переопределяемого файла>"/>
    </replace-with>
</overrides>
```

  - В теге `replace-with` укажите свойство `class` - относительный путь до переопределенного файла начиная с каталога `ru.bitec.app`.
  - В теге `when-type-is`, вложенном в предыдущий тег, укажите относительный путь до базового файла, так же начиная с каталога `ru.bitec.app`.

- Так например, если вы переопределили *Api* и *Avi*:

```
<?xml version="1.0" encoding="UTF-8"?>
<overrides>
    ....
    <replace-with class="ru.bitec.app.<ProjectModule>.bs.Bs_GoodsOverrideAvi">
        <when-type-is class="ru.bitec.app.gds.Bs_GoodsAvi"/>
    </replace-with>
    <replace-with class="ru.bitec.app.<ProjectModule>.bs.Bs_GoodsOverrideApi">
        <when-type-is class="ru.bitec.app.gds.Bs_GoodsApi"/>
    </replace-with>
    ....
</overrides>
```

Примечание: Так же стоит заметить, что если вы переопределили какой-либо api/avi-файл, и в нем **переопределили** метод(ы) базового api/avi, тогда в другой части кода вы можете вызывать эти методы используя имя базового файла, ваша проектная логика будет подхватываться системой и

вызываться от имени базового файла. Однако, если вы добавили новый метод, который отсутствует в базовом файле, вызывать его придется, используя имя переопределенного файла.

34.6 Работа с данными, хранящимися в jsonb контейнере

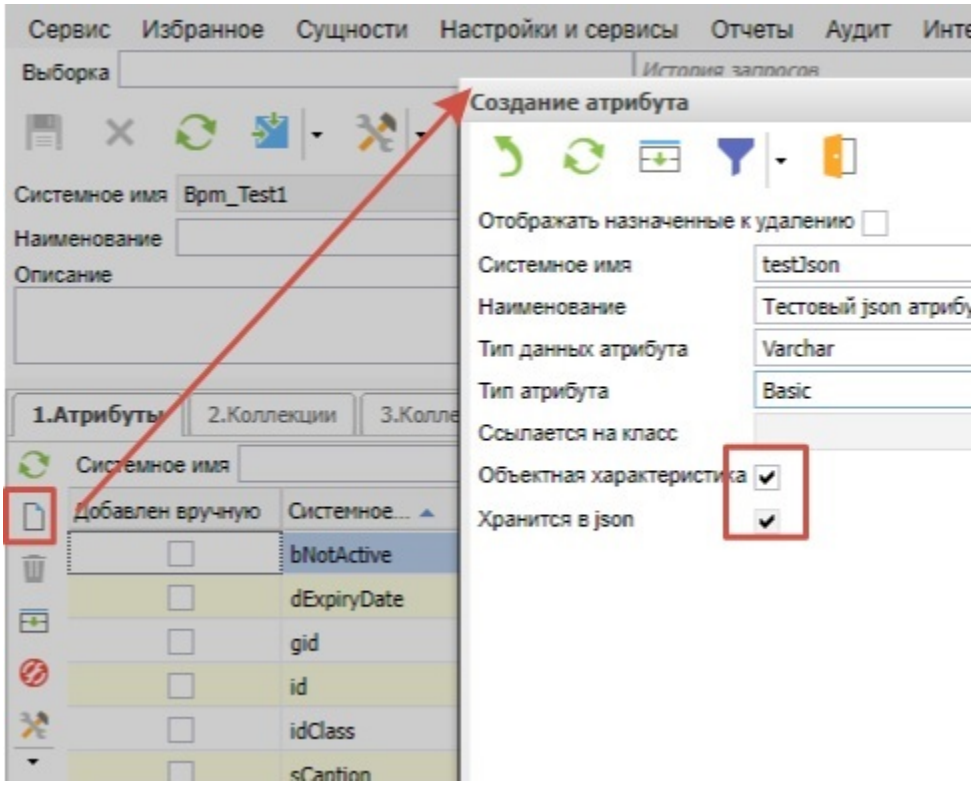
34.7 Атрибуты, хранящиеся в jsonb контейнере

34.7.1 Предисловие

В нашей системе есть два типа атрибутов:

- Базовые, определяемые в odm-файле
- Json-хранимые - определяемые как через интерфейс приложения(в том числе универсальные характеристики), так и через исходный код

В самом простом случае при создании json-атрибута в интерфейсе приложения он будет отмечен как объектная характеристика, хранящаяся в json.



Редактировать такие атрибуты можно на вкладке объектных характеристик соответствующего класса. Обработка этих атрибутов уже реализована средствами фреймворка.

Внимание: В базе для хранения данных используется контейнер типа jsonb

34.8 Получение и установка атрибутов, хранимых в json

34.8.1 Получение и обработка данных методами api текущего класса для объектных характеристик

Примечание: Все json-хранимые объектные характеристики и универсальные характеристики хранятся в контейнере с именем `jObjAttrs_dz`. И все перечисленные далее методы чтения и записи данных для атрибутов-характеристик работают именно с этим контейнером.

34.8.2 Для атрибута, хранимого в json и настроенного как объектная характеристика в `JObjectAttrApi.scala`, реализован набор методов для записи и чтения данных из контейнера:

Запись по id

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#setObjAttrValue(rop: ApiRop,␣
↪idpAttr: NLong, pValue: Any)
```

- Метод установки значения json атрибута по его id
- Вызвать этот метод можно от апи вашего класса
- `rop` - rop-а, чей атрибут вы хотите изменить
- `idpAttr` - id атрибута
- `pValue` - значение, которое хотите установить атрибуту

Метод проверяет, что переданное значение совпадает с типом данных, настроенным на атрибуте.

Запись по имени

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#setObjAttrValue(rop: ApiRop,␣
↪spAttrName: NString, pValue: Any)
```

- Метод установки значения json атрибута по его системному имени
- Вызвать этот метод можно от апи вашего класса
- `rop` - rop-а, чей атрибут вы хотите изменить
- `spAttrName` - системное имя атрибута
- `pValue` - значение, которое хотите установить атрибуту

Метод проверяет наличие настроенного атрибута в классе Btk_Attribute, если атрибут не найден, то кидает ошибку.

После успешного прохождения проверки вызывается метод *def setObjAttrValue(rop: ApiRop, idpAttr: NLong, pValue: Any)*.

Внимание: При передаче значения для устанавливаемого атрибута необходимо передавать данные, приведенные к соответствующему типу!

Чтение по id

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#getObjAttrValue(rop: ApiRop,␣
↪idpAttr: NLong)
```

- Метод получения значения json атрибута по его id
- Так же, как и предыдущие, проверяет наличие атрибута в Btk_Attribute
- Вызвать этот метод можно от апи вашего класса
- `rop` - rop-а, чей атрибут вы хотите получить
- `idpAttr` - id атрибута

Чтение по системному имени

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#getObjAttrValue(rop: ApiRop,␣
↪spAttrName: NString)
```

- Метод получения значения json атрибута по его системному имени
- Так же, как и предыдущие, проверяет наличие атрибута в Btk_Attribute
- Вызвать этот метод можно от апи вашего класса
- `rop` - rop-а, чей атрибут вы хотите получить
- `idpAttr` - id атрибута

Метод проверяет наличие настроенного атрибута в классе Btk_Attribute, если атрибут не найден, то кидает ошибку.

После успешного прохождения проверки вызывается метод *def getObjAttrValue(rop: ApiRop, idpAttr: NLong)*.

34.9 Универсальные характеристики(UC)

Универсальные характеристики являются отдельным сервисом, позволяющим подключить к объектам класса дополнительные атрибуты и не требующим компиляции исходного кода.

Универсальные характеристики хранятся в справочнике Btk_UniversalCharacteristic.

При подключении универсальной характеристики в качестве атрибута к целевому классу итоговые данные по этому атрибуту-характеристике записываются в контейнер `jObjAttrs_dz`, туда же, где хранятся значения json объектных характеристик.

34.9.1 Получение и обработка данных методами api текущего класса для универсальных характеристик

Запись UC по id

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#setUniCharValue(rop: ApiRop,␣
↪idpUniChar: NLong, pValue: Any)
```

- Метод установки значения UC json атрибута по его id
- Вызвать этот метод можно от апи вашего класса
- `rop` - rop-а, чей атрибут вы хотите изменить
- `idpUniChar` - id UC атрибута
- `pValue` - значение, которое хотите установить атрибуту. Возможна множественная установка значений

Метод проверяет, что переданное значение совпадает с типом данных, настроенным на атрибуте.

Запись по имени

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#setUniCharValue(rop: ApiRop, ␣
↪spUniCharName: NString, pValue: Any)
```

Аналогичен предыдущему. Предварительно ищет id универсальной характеристики по системному имени. После вызывает метод вызывается метод *def setUniCharValue(rop: ApiRop, idpUniChar: NLong, pValue: Any)*. Возможна множественная установка значений.

Чтение UC по id

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#getUniCharValue(rop: ApiRop,␣
↪idpUniChar: NLong)
```

- Метод получения значения json атрибута по его id
- Так же как и предыдущие проверяет наличие атрибута в Btk_Attribute
- Вызвать этот метод можно от апи вашего класса
- `rop` - rop-а, чей атрибут вы хотите получить
- `idpUniChar` - id UC атрибута

Чтение UC по системному имени

```
ru.bitec.app.btk.class_.service.objectAttr.JObjectAttrApi#getUniCharValue(rop: ApiRop,␣
↪spUniCharName: NString)
```

Проверяет наличие универсальной характеристики по системному имени. После вызывает метод *def getUniCharValue(rop: ApiRop, idpUniChar: NLong)*.

34.10 Чтение и запись иных данных в json контейнер

Если вам нужно хранить иные структуры данных, не подходящие под прошлые пункты, тогда:

- Для получения доступа к данным в контейнере используйте метод подключения json документа `ru.bitec.app.gtk.eclipse.json.JEmbeddedDoc#parseProperty` с последующей конвертацией к JObject и метод `ru.bitec.app.gtk.eclipse.json.JEObject#set`
- Для добавления или перезаписи значения по ключу:

```
JEmbeddedDoc.parseProperty(rop,"Имя json контейнера").asJObject.set(key: NString,␣
↪value: NString)
```

> **Внимание:** Если попытаться напрямую установть значение в атрибут, можно затереть данные из контейнера, которые там присутствовали ранее!

- Для получения данных из контейнера по ключу используйте один из типизированных геттеров `ru.bitec.app.gtk.eclipse.json.JEObject`:

```
JEmbeddedDoc.parseProperty(rop,"Имя json контейнера").asJObject.getNNumber(key:␣
↪NString)
//def getNString(key: NString)
//def getBoolean(key: NString)
//def getNLong(key: NString)
//def getNDate(key: NString)
//def getNGid(key: NString)
//def getHashMap(key: NString)
```

Или общий сеттер:

```
getJValue(key: NString)
```

С последующей обработкой полученных данных.

34.11 Примечания по работе с jsonb контейнерами

- Все строковые значения необходимо заворачивать в двойные кавычки `"text"`
- Если вы пытаетесь поставить в нестроковое поле пустое значение, то используйте значение „null“
- Оператор `->` возвращает элемент типа `jsonb`. Оператор `->>` возвращает элемент типа `text`. Для промежуточных манипуляций с объектами контейнера используйте `->`, а для получения конечного значения `->>`
- При добавлении значений при помощи `||` или `jsonb_set()` к уже существующему набору(например `jtypesizeattrs`) пар ключ-значение необходимо оборачивать его в `coalesce(jtypesizeattrs, '{}':: jsonb)`, так как если `jtypesizeattrs` до этого был null, то прибавляя у нему любое значение, мы получим null
- При добавлении значений при помощи `||` или `jsonb_set()` к уже существующему массиву(например `jSign_dz`) необходимо оборачивать его в `coalesce(jSign_dz, '[]':: jsonb)` так как если `jSign_dz` до этого был null, то, прибавляя к нему любое значение, мы получим null

34.12 Работа с контейнером jObjAttrs_dz

В контейнере jObjAttrs_dz данные хранятся в виде мапы, соответственно сразу можно использовать метод ->> для получения конечных данных из этой мапы.

Например:

- Получение json-атрибута из контейнера:

```
select
    w.jObjAttrs_dz ->> 'idJsonAttr' as idJsonAttr,
    cast(w.jObjAttrs_dz ->> 'bJsonAttr' as numeric(1)) as bJsonAttr,
    cast(w.jObjAttrs_dz ->> 'nJsonAttr' as numeric(38, 18)) as nJsonAttr,
    w.jObjAttrs_dz ->> 'sJsonAttr' as sJsonAttr,
    cast(w.jObjAttrs_dz ->> 'dJsonAttr' as date) as dJsonAttr,
    cast(w.jObjAttrs_dz ->> 'dJsonAttr' as timestamp) as dJsonAttrDateTime
from (
    select '{"idJsonAttr": 123456, "bJsonAttr": 0, "nJsonAttr": 0.01, "sJsonAttr":
↪"Some test string", "dJsonAttr": "01.01.2101 10:10:10"}' :: jsonb as jObjAttrs_
↪dz) w
return: |idJsonAttr|bJsonAttr|           nJsonAttr|          sJsonAttr|   dJsonAttr|␣
↪       dJsonAttrDateTime|
        |    123456|        0|0.010000000000000000|"Some test string"|"2101-01-01"|
↪"2101-01-01 10:10:10.000"|
```

34.13 Работа с обобщенными json контейнерами в Postgres

Например:

- Получение поля из массива

```
select (w.jsign_dz -> 1) -> 'sFIO'  -- 1 - индекс массива. Отсчет ведётся с 0. sFIO␣
↪- имя искомого поля
from (select
        '[
            {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":␣
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
            {"sFIO": "Линчук Владимир Владимирович", "dDate": "04.12.2019",
↪"sPosition": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType
↪": 95402}
        ]' :: jsonb as jsign_dz) w

return: "Линчук Владимир Владимирович"
```

- Апдейтим поле в массиве записей. В этом случае используем метод jsonb_set. Аргументы - первоначальное значение, путь к изменяемому значению, устанавливаемое значение

```
update stk_warrantin w
set jsign_dz = jsonb_set(coalesce(w.jSign_dz, '[]':: jsonb),'{1,sFIO}', '"Иванов␣
↪Ива Иванович"') -- {1,sFIO} - путь к интересуемому полю 1 - индекс в массиве.␣
↪sFIO - поле в массиве.
where id = someId
```

```
select jsonb_set(coalesce(w.jSign_dz, '[]':: jsonb),'{1,sFIO}', '"Иванов Ива␣
↪Иванович"')
from (select
        '[
            {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":␣
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
            {"sFIO": "Линчук Владимир Владимирович", "dDate": "04.12.2019",
↪"sPosition": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType
↪": 95402}
        ]' :: jsonb as jsign_dz) w

return: [
            {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":␣
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
            {"sFIO": "Иванов Ива Иванович", "dDate": "04.12.2019", "sPosition":␣
↪null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType": 95402}
        ]
```

- Удаляем запись из результирующего массива (пару ключ - значение из объекта)

```
select w.jsign_dz - 1
from (select
        '[
            {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":␣
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
            {"sFIO": "Линчук Владимир Владимирович", "dDate": "04.12.2019",
↪"sPosition": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType
↪": 95402}
        ]' :: jsonb as jsign_dz) w

return: [
            {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":␣
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401}
        ]
```

- Удаляем поле из мапы

```
select (w.jsign_dz -> 0) - 'sFIO'
from (select
        '[
            {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":␣
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
            {"sFIO": "Линчук Владимир Владимирович", "dDate": "04.12.2019",
↪"sPosition": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType
↪": 95402}
        ]' :: jsonb as jsign_dz) w

return: {"sPosition": "Нач. отдела", "idEmployee": null, "idDepartment": 102696,
↪"sBasisDocument": null, "idBlankSignType": 95401}
```

- Добавляем запись в массив. В этом случае работаем с объединением двух массивов. Необходимо задать строку с новым объектом массива и добавить её с помощью операции || к старому значению

```
select coalesce(w.jSign_dz, '[]':: jsonb)  ||
        '[
            {"sFIO": "Петров Е.В.", "sPosition": "тест", "idEmployee": null,
↪"idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401}
        ]' :: jsonb
from (select
        '[
            {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee":␣
↪null, "idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
            {"sFIO": "Линчук Владимир Владимирович", "dDate": "04.12.2019",
↪"sPosition": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType
↪": 95402}
        ]' :: jsonb as jsign_dz) w

return:
    [
        {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee": null,
↪"idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
        {"sFIO": "Линчук Владимир Владимирович", "dDate": "04.12.2019", "sPosition
↪": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType": 95402},
        {"sFIO": "Петров Е.В.", "sPosition": "тест", "idEmployee": null,
↪"idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401}
    ]
```

- Можно полностью перезаписать контейнер jSign_dz. Можно сформировать строку при помощи обычных строковых методов и полностью перезаписать атрибут jSign_dz, используя приведения строки к типу jsonb и соблюдая синтаксис jsonb

```
update stk_warrantin w
set jsign_dz = ('[{"sFIO": "' || 'Тест' || '", "sPosition": "тест", "idEmployee":␣
↪null, "idDepartment": ' || '1' || ', "sBasisDocument": null, "idBlankSignType":␣
↪95401}]') :: jsonb
where id = someId
```

- Разворачиваем массив подписей в записи

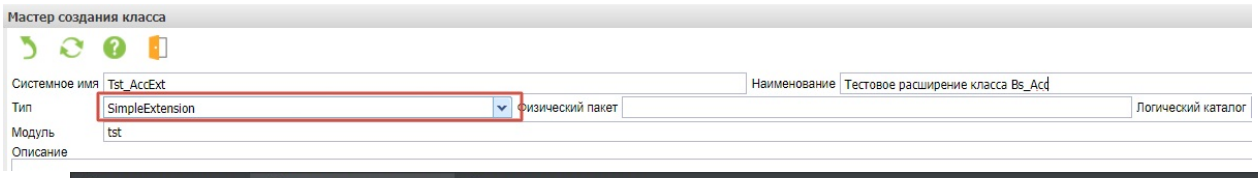
```
select value ->> 'sFIO' as sFIO
from jsonb_array_elements(
    '[
        {"sFIO": "Степанова Е.В.", "sPosition": "Нач. отдела", "idEmployee": null,
↪"idDepartment": 102696, "sBasisDocument": null, "idBlankSignType": 95401},
        {"sFIO": "Линчук Владимир Владимирович", "dDate": "04.12.2019", "sPosition
↪": null, "idEmployee": 21035, "sBasisDocument": null, "idBlankSignType": 95402}
    ]' :: jsonb) v

return:
    1: `Степанова Е.В.`
    2: `Линчук Владимир Владимирович`
```

34.14 Классы-расширения. Simple Extensions

34.14.1 Создание класса-расширения

Класс-расширение создается полностью аналогично любому другому классу. При создании необходимо указать супертип `simpleExtension`.



Либо

```
Bs_Acc.odm.xml | Tst_AccExt.odm.xml | global3-class-1.0.xsd
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <class xmlns="http://www.global-system.ru/xsd/global3-class-1.0" name="Tst_AccExt"
3         caption="Тестовое расширение класса Bs_Acc" supertype="simpleExtension"
4         cardEditor.representation="Card" listEditor.representation="List"
5         viewOptions.openCardType="mdi">
6      <attributes>
7          <attr name="gidRef" attribute-type="Varchar" order="-1" type="basic" isVisible="false"/>
8          <attr name="idClass" attribute-type="Long" caption="idClass" order="-2" type="basic" isVisible="false"/>
9      </attributes>
10 </class>
11
```

Внимание: Если к классу-расширению предполагается добавлять коллекции, то он не может иметь супертип simpleExtention! В таком случае можно установить ему тип Journal или иной, наиболее подходящий в контексте задачи!

Внимание: Как и у миксинов, у классов-расширений не должно быть собственных id и gid, они ссылаются на gid мастер-объекта при помощи поля gidRef!

Внимание: В отличие от миксинов, при создании расширения, в мастер-классе не делается никаких записей, напрямую связанных с классом-расширением. В первую очередь потому, что большая часть расширений обычно создается в модулях, прямой доступ к которым из мастер-объекта невозможен!

34.14.2 Создание и удаление объекта класса-расширения

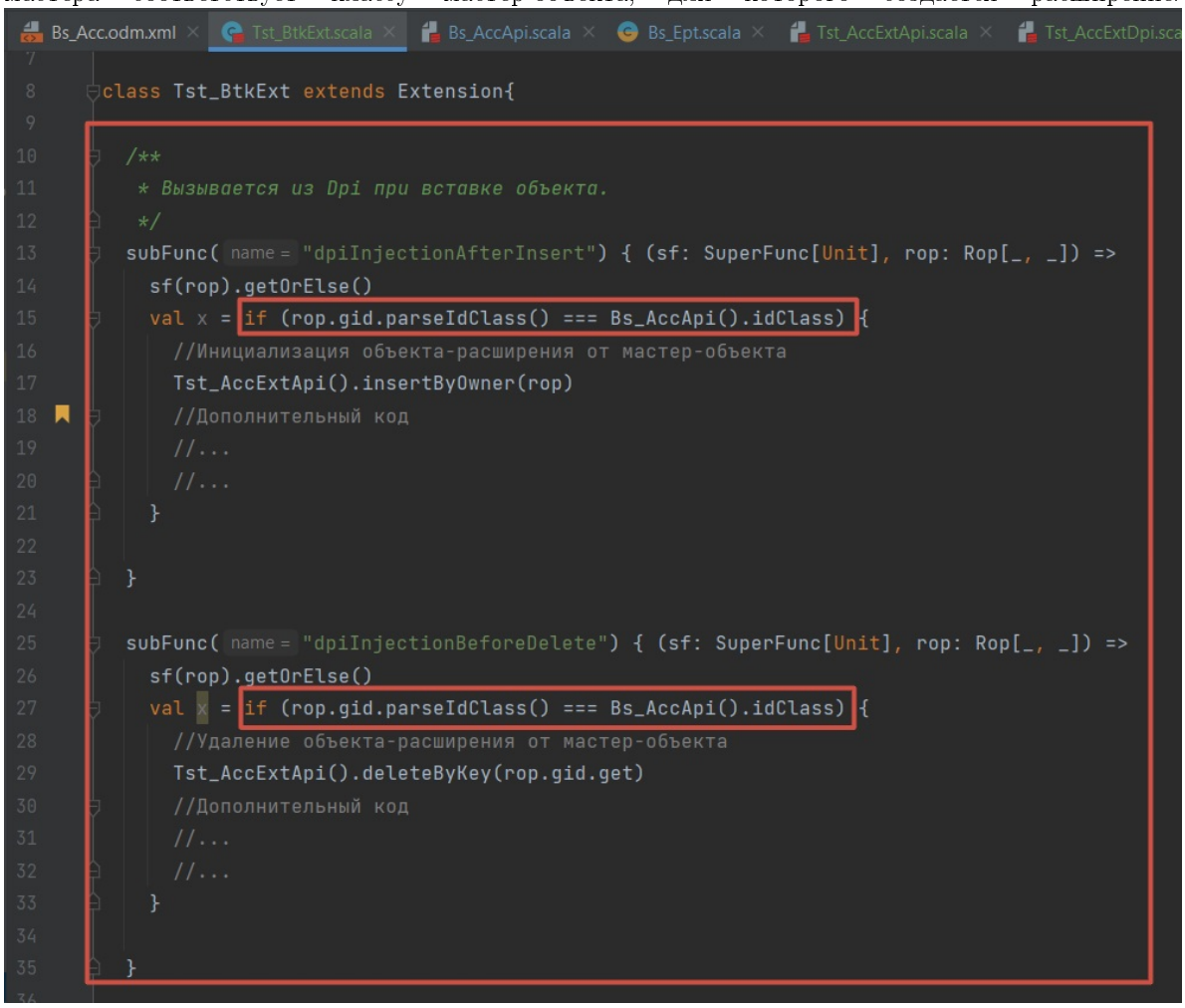
- Объект-расширение тесно связан с мастер-объектом и его жизненным циклом. Жизненный цикл объекта расширения не может выходить за пределы такового у его мастер-объекта, но может быть меньше. Для создания и удаления объекта-расширения требуется создавать *точки расширения* и добавлять их вызов в нужный момент жизненного цикла мастер-объекта.
- Объект-расширение может быть создан не раньше создания его мастер-объекта. Самый ранний этап жизненного цикла мастера, в который может быть создано расширение - момент создания мастера. Для этого можно использовать уже существующую точку расширения `Btk_Ept#dpiInjectionAfterInsert`. Код этой точки расширения будет вызван сразу же после

создания мастер-объекта. Вызов этой точки расширения добавлен во все dpi по умолчанию, поэтому требуется создавать только вторую часть точкирасширения - файл `Ext.scala`.

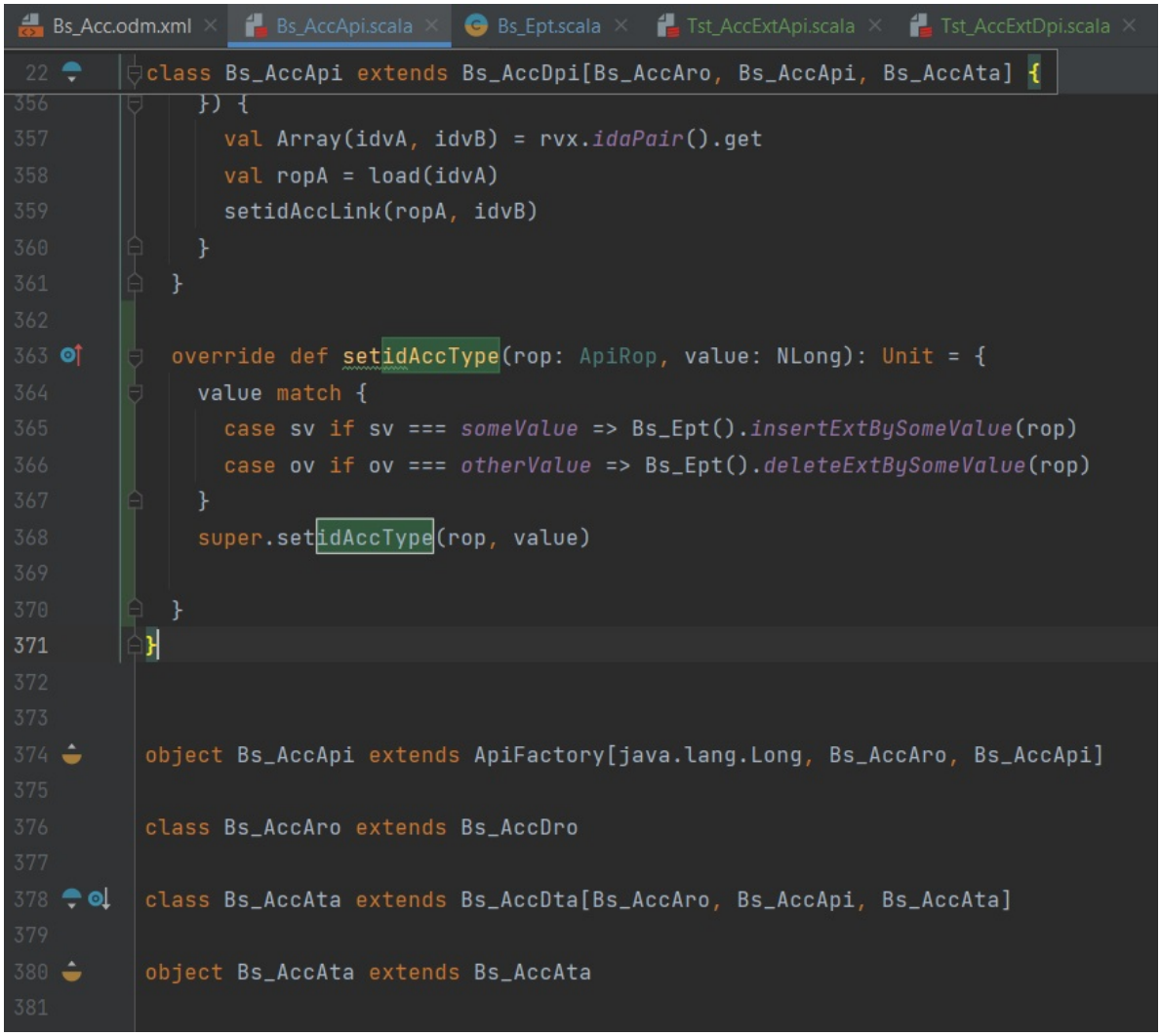
- Аналогично производится удаление объекта `Btk_Ept#dpiInjectionBeforeDelete` - эта точка расширения вызывается перед удалением мастер-объекта. Её вызов так же доавлен во все dpi по умолчанию.

Внимание: Необходимо учитывать, что эти точки расширения вызываются при создании объектов ЛЮБЫХ классов! Поэтому во второй части точки расширения `dpiInjectionAfterInsert` или `dpiInjectionBeforeDelete` *СТРОГО НЕОБХОДИМО* проверять к какому классу принадлежит объект мастер-класса, который обрабатывается точкой расширения!

- Инициализация/удаление объекта-расширения происходит только в случае, если класс мастера соответствует классу мастер-объекта, для которого создается расширение.



- Если вам необходимо укоротить жизненный цикл объекта-расширения, например если такой объект может существовать только пока мастер-объекту присвоен определенный тип объекта, тогда необходимо создать свою точку расширения для мастер-объекта, либо воспользоваться уже существующей точкой, предназначенной для создания объектов-расширений, однако с существующими точками стоит быть предельно внимательными и убедиться, что такая точка подходит для вашей задачи!



Вызов такой точки расширения происходит из api-сеттера соответствующего атрибута *МАСТЕР-ОБЪЕКТА*, на который завязан жизненный цикл расширения. Вторая часть точки расширения объявляется соответствующим образом из парного для этой *точки* файла.

- В случае использования специализированной точки расширения для создания и удаления файлов допускается отсутствие проверки на совпадение классов родительских объектов, в отличие от использования общих точек `Btk_Ept#dpiInjectionAfterInsert` и `Btk_Ept#dpiInjectionBeforeDelete`.

Примечание: В SimpleExtentionApi определены базовые методы для работы с объектами-расширениями и доступные для работы из вашего класса-расширения по умолчанию. Для вывода данных объекта-расширения совместно с мастер-объектом рекомендуется создавать отдельную выборку в классе-расширении, в которую выводить все требуемые атрибуты(так же, при необходимости, на эту выборку можно добавлять атрибуты мастер-объекта), после чего прикреплять эту выборку к выборке мастер-объекта, используя настройку на типе объекта, регистрируя из модуля в котором создано расширение, закладку на тип(ы) мастер-объекта, либо используя схожий подход для возможности динамической настройки детализации выборок мастер-объекта без использования настроек на типе объекта(например при отсутствии такого атрибута у мастера).

34.15 Запуск отладки/теста

1. Запустите `idea`
   Если вы запускаете idea от ярлыка gsf-cli, idea запустится с открытым проектом. Иначе может потребоваться открыть проект вручную.
2. При необходимости обновите sbt(отметка №2)
   Обновление требуется, в случае если
   - Произошли изменения в структуре проекта
     Файлы `*.sbt` , `project.yaml`
   - Обновились внешние библиотеки
3. - Для того, чтобы запустить отладку:
     Для этого выполните пункт меню `Run > Debug 'Project_Name'`
   - Для того, чтобы запустить тест:
     Для этого нажмите на зелёную кнопку play напротив строку с обьявлением теста в коде, далее `Debug 'Test_Name'`

34.16 Как вносить изменения

Данный совет напоминает какие действие надо совершать в среде разработки Intelliji Idea при изменении тех или иных элементов в проекте решения.

34.16.1 Изменение разметки класса

1. Перейдите в проект Intelliji Idea
2. Создайте или отредактируйте разметку класса
   Файл с расширением `.odm.xml`
3. Запустите генерацию кода
   Выберете в проекте файл с измененной разметкой и выполните пункт меню `Tools > External Tools > Generate Sources`

   Генерация кода файлы окружения класса(класс для хранение данных в кэше, бизнес логика по умолчанию).
4. Запустите сборку проекта
   Пункт меню `Build > Build Project`
5. Обновите метаданные в базе данных
   Выберете в проекте файл с измененной разметкой и выполните пункт меню `Tools > External Tools > Generate Tables`
6. Перейдите в приложение для которого внесли изменения
   По умолчанию для локального хоста приложение запускается по адресу `http://localhost:8080/`
7. Обновите код и библиотеки orm
   Пункт меню `Сервис > Управление решением > Обновить код, Библиотеки Orm`

34.16.2 Изменение разметки выборки или исходного кода

1. Перейдите в проект Intelliji Idea
2. При необходимости отредактируйте разметку выборки
   Файл с расширением `.avm.xml`
3. При необходимости измените программный код
   Файлы с расширением `.scala`, `.java`
4. Запустите сборку проекта
   Пункт меню `Build > Build Project`
5. Перейдите в приложение для которого внесли изменения
   По умолчанию для локального хоста приложение запускается по адресу `http://localhost:8080/`
6. Обновите код
   Пункт меню `Сервис > Управление решением > Обновить код`

> **Внимание:** Убедитесь в приложении сброшен флаг `Использовать кэш метаданных выборок`, пункт меню `Сервис > Управление решением > Использовать кэш метаданных выборок`

34.17 Как обновить внешние зависимости

Данный совет предполагает что для работы с проектом вы используете gsf-cli

1. Закройте idea

   > **Внимание:** Должны быть закрыты все экземпляры Intellij Idea

2. Запустите ярлык `refresh.cmd` Ярлык находится в каталоге `[programs]\gsf-cli\workspace\links\[project_name]\refresh.cmd`, где

   - `programs`
     Каталог куда установлен `gsf-cli`
   - `project_name`
     Имя проекта для обновления

3. Откройте idea
   Для этого запустите ярлык `[programs]\gsf-cli\workspace\links\[project_name]\start_idea.cmd`
4. Обновите sbt
   Смотрите документацию idea
5. Соберите проект
   Для этого выполните пункт меню `Build > Build Project`

34.18 Как переопределить методы API

1. Перейдите на необходимый класс API в среде разработки
2. Внутри класса пропишите `override def`
3. Из выпадающего списка выберите необходимый метод для переопределения.

34.19 Как переопределить методы AVI

1. Перейдите на необходимую выборку AVI в среде разработки
2. Перейдите на трейт отображения
3. Внутри трейта пропишите `override def`
4. Из выпадающего списка выберите необходимый метод для переопределения.

34.20 Как переопределить сеттеры API

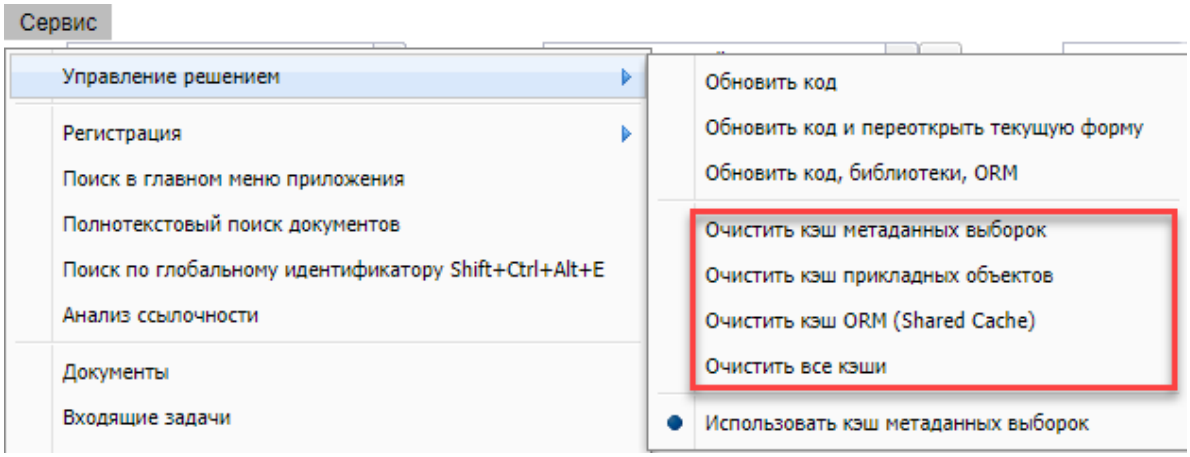
1. Перейдите на необходимый класс API в среде разработки
2. Внутри класса пропишите `override def set`
3. Из выпадающего списка выберите необходимый сеттер для переопределения и нажмите ввод.

34.21 Как переопределить сеттеры AVI

1. Перейдите на необходимую выборку AVI в среде разработки
2. Перейдите на трейт отображения
3. Внутри трейта пропишите `override def set`
4. Из выпадающего списка выберите необходимый сеттер для переопределения.

34.22 Как сбросить кэш

1. В любом приложении при наличии прав в пункте меню `Сервисс -> Управление решением` будет блок операций по сбросу кэша



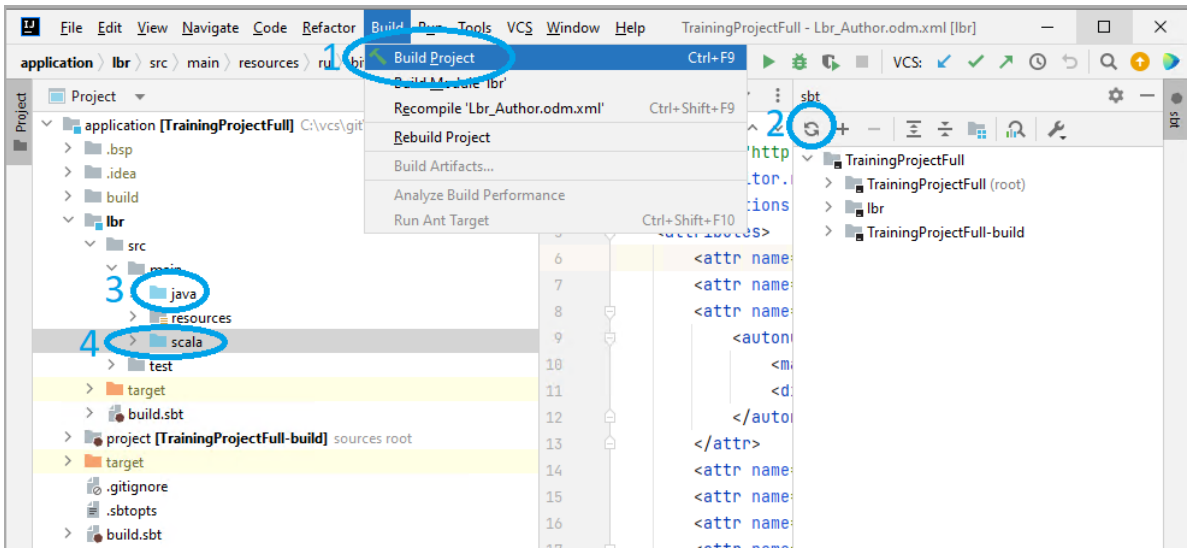
2. Для сброса кэша метаданных выборок необходимо снять признак `Использовать кэш метаданных выборок`

34.23 Как собрать проект

1. Запустите `idea`
   Если вы запускаете idea от ярлыка gsf-cli, idea запустится с открытым проектом. Иначе может потребоваться открыть проект вручную.

Совет: Более подробно по работе с проектом смотрите документацию idea

---



2. При необходимости обновите sbt(отметка №2)
   Обновление требуется, в случае если

   - Произошли изменения в структуре проекта
     Файлы `*.sbt` , `project.yaml`
   - Обновились внешние библиотеки

> **Внимание:** Папки исходного кода должны быть отмечены цветом, смотрите отметки №3 и №4.

3. Соберите проект
   Для этого выполните пункт меню `Build > Build Project`(отметка №1)

34.24 Как создать класс

1. В директории с ресурсами (`Наименование модуля`/src/main/resources/ru/bitec/app) создать файл odm(Object domain model/Доменная модель сущности) с наименованием `Наименование модуля`/`Наименование модуля_Наименование класса`.odm.xml
2. В odm определить необходимые атрибуты, подключить коллекции и миксины.
3. Сгенерировать код по созданному odm файлу
   1. Кликнуть правой кнопкой мыши по файлу
   2. В подменю External Tools выбрать Generate Sources
4. Собрать проект
5. Сгенерировать таблицы по odm файлу
   1. Кликнуть правой кнопкой мыши по файлу
   2. В подменю External Tools выбрать Generate Tables

34.25 Как создать коллекцию

1. В директории с ресурсами (`Наименование модуля`/src/main/resources/ru/bitec/app) создать файл odm(Object domain model/Доменная модель сущности) с наименованием `Наименование модуля`/`Наименование модуля_Наименование коллекции`.odm.xml
2. Указать тип класса: `supertype="collection"`
3. В odm определить ссылочный атрибут на класс родитель у данного атрибута указать `genListCollectionRep="true"`
4. В odm определить необходимые атрибуты
5. Подключить коллекцию к классу родителю
   - Для этого в odm класса родителя необходимо указать

```
<collections>
   <collection name="Наименование коллекции" cascadeOnDelete="true" ref.attr=
↪"Наименование ссылочного атрибута на класс родитель в коллекции"/>
</collections>
```

6. Сгенерировать код по odm класса родителя
   1. Кликнуть правой кнопкой мыши по файлу
   2. В подменю External Tools выбрать Generate Sources
7. Собрать проект

8. Сгенерировать таблицы по odm файлу коллекции
    1. Кликнуть правой кнопкой мыши по файлу
    2. В подменю External Tools выбрать Generate Tables

34.26 Как создать новое отображение

1. В Avi файле в классе объявите новый `trait`, унаследовав его от нужных отображений
2. В Avi файле в классе добавьте метод для отображения - например

```
def list(): List = {
    new List {
    override def meta = this
    }
}
```

   В случае, если в переопределяемом отображении уже есть такой метод, нужно добавить `override`

3. В avm файле добавьте разметку для нового отображения

34.27 Как создать точку расширения

Модуль в котором необходима точка расширения - `Модуль1`. Модуль расширения - `Модуль2`.

1. В модуле в котором необходима точка расширения создаем класс с объектом компаньоном `Модуль1_Ept.scala`.
2. Класс унаследовать от класса `ExtensionPoint`.
3. Объект компаньон унаследовать от PkgFactory[`Модуль1`_Ept].
4. Создать метод точки расширения:

```
  val extentionMethodExamlpe = declFunc("extentionMethodExamlpe") {
    (sf: SuperFunc[NString], arg1: NString, arg2: NString) =>
      sf().getOrElse(None.ns)
  }
```

1. В модуле расширении создать класс `Модуль2_Модуль1Ext.scala`.
2. Унаследовать класс от класса `Extension`.
3. Создать метод расширения:

```
  subFunc("extentionMethodExamlpe") { (sf: SuperFunc[NString], arg1: NString, arg2:␣
↪NString) =>
    ...
  }
```

1. В файле `src/main/resources/META-INF/extensions.xml` модуля расширения зарегистрировать новую точку расширения:

```
<exts>
    <ext class="ru.bitec.app.Модуль2.Модуль2_Модуль1Ext" targetEpt="`Модуль1`_Ept"/>
</exts>
```

34.28 Найти и открыть класс из настройки системы

1. Откройте приложение «Настройка системы»
2. Выберите пункт меню «Сущности» -> «Классы»
3. В фильтре введите системное имя необходимого класса
4. На панели операций нажмите на операцию «Редактировать объекты класса в списке».

34.29 Настройка автонумерации

1. Откройте `odm` класса;
2. Укажите для нужного атрибута свойство `type` = `autonum`;
3. Создайте внутри тега `attr` вложенный тег `autonum`. Укажите в нем следующие свойства:
   - `id` - уникальный в рамках класса идентификатор автонумерации.
   - `isHoleFill` - заполнение пропусков (опционально). Если в настройках автонумерации указан разрез, то автоматически включено. Чтобы выключить нужно прописать `isHoleFill` = `false`.
4. Внутри тега `autonum` создайте вложенные теги:
   - `mask` - маска автонумерации. Маска должна собой представлять jexl-скрипт, с экранированием спец. символов. Для каждого автонумерующегося атрибута может быть настроена своя маска автонумерации. В маске доступен для использования параметр counter, который будет заменен на значение счетчика.
   - `dimension` - разрез автонумерации:
     - Установите у тега следующие свойства:
       * `name` - название разреза. Задается всегда, даже если используется автонумерация без разреза;
       * `attr` - атрибут, в рамках которого рассматривается разрез. Разрезы указываются из атрибутов класса. Используется только для автонумерации с разрезом.
       * `order` - порядковый номер. Если его не задать, то разрезы будут обрабатываться в алфавитном порядке. Используется для автонумерации с разрезом по нескольким атрибутам.
     - Дополнительные возможности:
       * При необходимости можно добавить более одного тега `dimension`. В этом случае разрез будет реализован по нескольким атрибутам.
       * К значениям атрибутов разреза возможно применение выражений, они указываются во вложенном теге `expression` для `dimension`. На каждый разрез может быть не больше одного выражения. Значение, указанное в элементе `expression`, будет

передано в обработчик Jexl-скриптов, параметр, заменяемый на значение атрибута разреза, называется `dimValue`.

Пример автонумерации без разреза:

```
<attr name="sNoDep"
    caption="Автонумерация без разреза"
    attribute-type="Varchar"
    type="autoNum"
    isHeadLine="true">
    <autonum id="1">
        <mask>lpad(counter,3,"0")</mask>
        <dimension name="dim1"/>
    </autonum>
</attr>
```

Пример указания нескольких разрезов:

```
<attr name="sTwoDepExpr"
        caption="Автонумерация с 2 разрезами"
        attribute-type="Varchar"
        type="autoNum">
    <autonum id="3">
        <mask>lpad(counter,3,"0")</mask>
        <dimension name="nNumber"
                attr="nNumber"
                order="10" />
        <dimension name="dDate_dim"
                attr="dDate"
                order="20">
            <expression>truncDate(dimValue)</expression>
        </dimension>
    </autonum>
</attr>
```

5. Сформируйте окружение класса:
   - Сформируйте исходный код для класса (`generate sources`);
   - Выполните сборку проекта (`build`);
   - Сформируйте таблицы класса (`generate tables`)

34.30 Настройка группировки класса

1. Откройте `odm` класса
2. У тега `class` укажите свойства:
   1. `group.type`
      - `single`
        Единичная группировка, объект может входить только в одну группу
      - `multi`
        Множественная группировка, объект может входить в несколько групп

2. `group.root`
   Системное имя корневой группы класса
3. `objectAttrCardType`
   - `group`
     Настройка объектных характеристик будет регулироваться группами
   - `groupAndObjectType`
     Настройка объектных характеристик будет регулироваться группами и дополнятся настройками с типа объекта
4. `group.isPanelVisible`
   Опционально. Если указать `true`, то панель групп будет отображаться при открытии списка объектов класса.

Пример тега `class`:

```
<class name="Some_Class" group.root="Profiles" group.type="single"␣
↪objectAttrCardType="group" group.isPanelVisible="true">
```

3. Под тегом `class` создайте тег `collections`, если он не создан
4. Под тегом `collections` создайте тег `var-collection`
5. Укажите свойства:
   - `name` = `Btk_ObjectGroup`
   - `ref.attr` = `gidSrcObject`

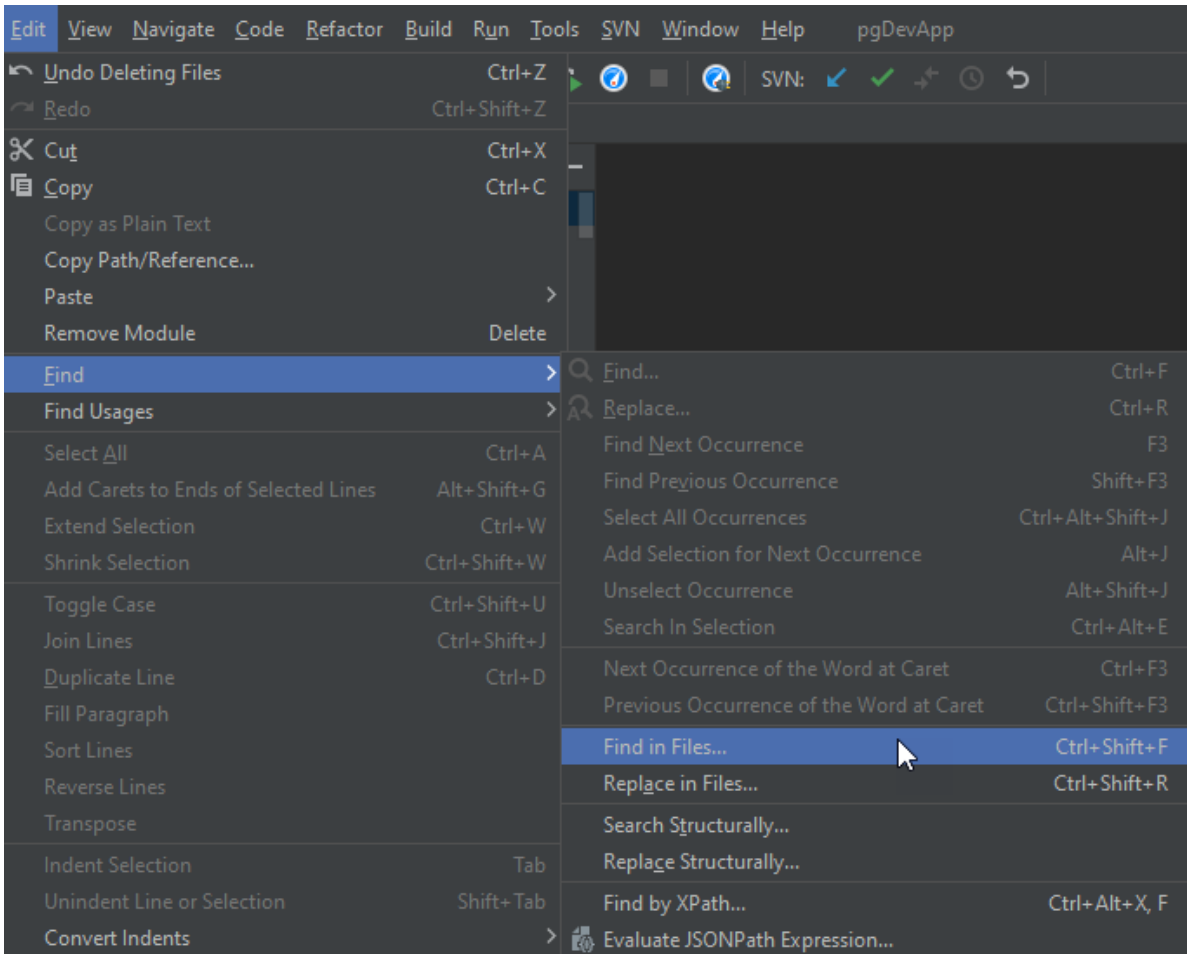
Пример тега `collections`:

```
<collections>
     <var-collection name="Btk_ObjectGroup" ref.attr="gidSrcObject"/>
 </collections>
```

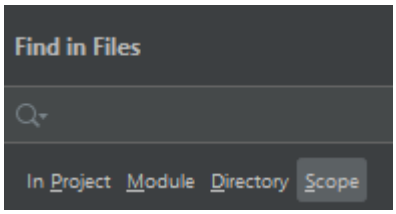
6. Сформируйте исходный код для класса (`generate sources`)
7. Сформируйте таблицы класса (`generate tables`)

34.31 Поиск по коду в Idea

1. Для поиска по коду Вызовите пункт меню `Edit` -> `Find` -> `Find in files` (горячая клавиша `Ctrl` + `Shift` + `F`)



2. Для поиска по всем файлам (в т.ч. и исходникам бандлов) выберите категорию `Scope`



3. Результат поиска можно открыть на отдельной закладке по кнопке `Open in Find Window` в нижнем-правом углу окна поиска

34.32 Создание логического атрибута класса

1. Откройте `odm` класса
2. Создайте тег `attributes`, если он не создан
3. Под тегом `attributes` укажите новый тег `attr`
4. Укажите свойства:
   - `name` - Имя атрибута
   - `caption` - Наименование атрибута
   - `attribute-type` = `Number`
   - `editorType` = `check`
5. Под тегом `attr` добавьте новый тег `booleanColumn`

Пример:

```
<attr name="bBool" caption="Логический" attribute-type="Number" editorType="check">
    <booleanColumn/>
</attr>
```

Глава 35

Практики разработки

35.1 Полезные практики от опытных разработчиков

35.1.1 Переопределение checkWorkAbility()

В `checkWorkAbility` нельзя использовать:

- `thisRop()`, заместо этого лучше использовать `getVar/getSelfVar`
- обходчики, особенно с инвалидацией в БД (запросом на согласование данных в БД) (например, `refreshByParent()`).

Нужно понимать, что `checkWorkAbility` вызывается часто, поэтому его оптимизация сильно сказывается на времени работы бизнес-логики.

35.1.2 session.flush() заместо session.commit()

`session.commit()` лучше избегать и использовать `session.flush()`.

`commit()` выгружает данные из кэша в БД.

`flush()` применяет изменения сессии к БД.

В случае использования `flush()` данные ещё не в БД и есть возможность откатить изменения, например, в случае ошибки.

`commit()` вызывает `afterEdit()` с проверкой обязательных полей в отличии от `flush()`.

35.1.3 Избегать использование [Option].get и [Collection].head

Не использовать для:

- Option `.get`, если внутри был `None`, то выпадет исключение.

  Использовать `.getOrElse(<значение или исключение, в случае None>)`.

- коллекций .head без проверки на пустоту коллекции.

  лучше использовать `.headOption`, что вернёт результат в конструкции `Option`, который дальше распаковать через `.getOrElse()`.

35.1.4 Миксин Btk_Object

миксин Btk_Object хранит все записи классов супер-типа документ и справочник.

В него нельзя вносить изменения! Это системный миксин.

35.2 Практика Avi

35.2.1 Про `selectStatement` и `onRefreshExt`

Изначально результат выборки формируется операцией `onRefresh` или `onRefreshItem` одним из двух способов: реляционный или объектный запрос.

Реляционный запрос

В этом случае в теле `onRefresh` вызывается метод `selectStatement` или `prepareSelectStatement` (вызывает тот же `selectStatement`, но с добавлением фильтров, описанных в `avm`). Иначе говоря, результатом операции `onRefresh` является текстовое значение, внутри которого реляционный запрос.

Примечание: onRefreshExt в случае формирования выборки реляционным запросом не вызывается!

Так формируется по умолчанию отображение `List` (кроме коллекций). Не учитываются значения хранящиеся в кэше, учитываются значения только из БД.

Пример `selectStatement` для отображения `List`:

```
override protected def selectStatement: String = {
  s"""SELECT
    t.id
    ,t.idClass
    ,t.idState
    ,t1.sHeadLine_dz as idStateHL
    ,t.idStateMC
    ,t.idObjectType
    ,t2.sHeadLine_dz as idObjectTypeHL
    ,t.idDepOwner
    ,t3.sHeadLine_dz as idDepOwnerHL
    ,t.gidSrc
    ,t.idResourceHolder
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    ,t4.sHeadLine_dz as idResourceHolderHL
    ,t.sRegNum
    ,t.dReg
    ,t.idPeriod
    ,t5.sHeadLine_dz as idPeriodHL
    ,t.sDescription
    ,t.gid
    ,t.sRegNum_dz
    ,t.sRegNumBMs_dz
    ,t.sRegNumidVer_dz
    FROM Oil_DemandMov t
      LEFT JOIN  Btk_ClassState t1 on t.idState = t1.id
      LEFT JOIN  Btk_ObjectType t2 on t.idObjectType = t2.id
      LEFT JOIN  Bs_DepOwner t3 on t.idDepOwner = t3.id
      LEFT JOIN  Bs_Contras t4 on t.idResourceHolder = t4.id
      LEFT JOIN  Bs_Period t5 on t.idPeriod = t5.id
  """
}
```

Коммит в БД при выполнении реляционного запроса и @FlushBefore

Важно понимать, что при реляционном запросе сервер приложения делает коммит в БД (особенность работы сервера приложения). Поэтому на выборках, где вводятся значения в поля и поддерживается возможность пользователю отменить изменения, нельзя использовать формирование выборки на реляционном запросе. Иначе данные запишутся в БД без разрешения пользователя (без нажатия пользователем на операцию сохранения (дискетка)).

Однако, иногда есть необходимость в выполнении выборки на реляционном запросе в отображении с редактированием полей. Например, выборка выпадающего списка `Lookup`.

В таком случае перед `onRefresh` необходимо добавить аннотоцию `@FlushBefore(mode = FlushBeforeMode.Disabled)`:

```
trait Lookup extends Default with super.Lookup {

  @FlushBefore(mode = FlushBeforeMode.Disabled)
  override protected def onRefresh: Recs = {
    s"""SELECT
        t.id
        ,t.sHeadLine_dz as sHeadLine
        ,t.sMnemoCode_dz as sMnemoCode
        ,coalesce(t.sMnemoCode_dz, '') || ' ' || coalesce(t.sHeadLine_dz, '') as␣
↪sMnemoCodeHeadLine
      from AsfEqp_IntoOperExt t
      order by upper(t.sHeadLine_dz)
    """
  }
}
```

Иначе при открытии выпадающего списка данные будут записываться в БД.

35.2.2 Объектный запрос

В этом случае onRefresh формируется из экземпляров объектов SRop (инструменты refreshByParent, load, TxIndex, OQuery) или case class'ов.

В этом случае учитываются значения, хранящиеся в кэше.

Примеры:

По умолчанию отображение List у коллекций формируется с помощью refreshByParent:

```
trait List_idPlanFactoryShip extends Default with super.List_Master {

  override protected def onRefresh: Recs = {
    Oil_PlanFactShipSegrGroupApi().refreshByParent(getIdMaster)
  }
}
```

По умолчанию отображение Card у коллекций формируется с помощью load:

```
trait Card extends Default with super.Card {

  override protected def onRefreshItem: Recs = {
    Oil_PlanFactShipSegrGroupApi().load(getVar(getPKFieldName).asNLong)
  }
}
```

onRefreshExt

В случае объектного запроса сервер сам вызывает метод onRefreshExt, в который можно дописать получение нехранимых полей на синтаксисе SQL, НЕ учитывая значения из кэша:

```
override protected def onRefreshExt: String = {
  s"""with t as ( select
   :id as id
  ,:idPlanFactoryShip as idPlanFactoryShip
  ,:idSegregationGroup as idSegregationGroup
)
SELECT
   t.id
  ,t1.sHeadLine_dz as idPlanFactoryShipHL
  ,t2.sHeadLine_dz as idSegregationGroupHL
  ,t2.sMnemoCode_dz as idSegregationGroupMC
FROM t
  LEFT JOIN  Oil_PlanFactoryShip t1 on t.idPlanFactoryShip = t1.id
  LEFT JOIN  Oil_SegregationGroup t2 on t.idSegregationGroup = t2.id
"""
}
```

Примечание: Также существует другой способ реализовать нехранимые поля (через case class AdditionalInfo), который будет описан дальше. Данный способ формирует значения с учётом кэша.

35.2.3 Добавление нехранимых полей

Здесь будет рассмотрено 2 случая, как реализовать нехранимые поля в выборке, реализованные реляционным запросом и объектным.

Для формирования нехранимого поля нужно:

- получить поле в запросе под заданным псевдонимом
- описать поле в разметке `avm` под тем же псевдонимом

Формировать реляционно или объектно

Если нехранимый атрибут вычисляется динамически от изменений на выборке, это значит, что работа идёт с значениями из кэша, т.е. с значениями, которые ещё не закоммичены в БД. Это означает, что нехранимое поле реализовывать нужно на объектном запросе.

Например: в карточке документа нужно посчитать сумму по всем записям в коллекции. Если поле суммы реализовать реляционно, то оно будет формироваться только из согласованных в БД значений. Это значит, для того чтобы учитывались новые записи в коллекции, их нужно закоммитить (сохранить) в БД. Если же поле реализовать объектно, то в нём будут учитываться данные из кэша, т.е. те самые созданные новые записи в коллекции, которые ещё не закоммичены в БД.

Нехранимые поля в реляционном запросе

Примечание: Не учитывает значения из кэша, которые не закоммичены в БД.

Чтобы добавить не хранимый атрибут реляционно, нужно иметь значение атрибута в выборке запроса `selectStatement`.

Примером может служить результат кодогенератора ссылочного поля для отображения `List`, где описывается получение значения хэдлайна в `Dvi` и описание атрибута в `dvm`.

Dvi:

```
override protected def selectStatement: String = {
  s"""SELECT
t.id
,t.idClass
,t.idObjectType
,t1.sHeadLine_dz as idObjectTypeHL   --нехранимое поле
,t.dReg
,t.sRegNum
,t.idState
,t2.sHeadLine_dz as idStateHL    --нехранимое поле
FROM Rzd_Train t
  LEFT JOIN  Btk_ObjectType t1 on t.idObjectType = t1.id
  LEFT JOIN  Btk_ClassState t2 on t.idState = t2.id
"""
```

dvm:

```
<representation name="Default">
    <attributes>
        <attr name="idObjectType" caption="Тип документа" isVisible="false" editorType=
↪"edit" order="10" isRequired="true"/>
        <attr name="idObjectTypeHL" caption="Тип документа" order="10.2" isRequired="true
↪">
            <editor>
                <buttonsEdit canEditText="true" changeableAttr="idObjectType">
                    <buttons lookup="true" openCard="true" reset="true"/>
                </buttonsEdit>
            </editor>
            <ref class="Btk_ObjectType"/>
        </attr>
        <attr name="idState" caption="Состояние" isVisible="false" editorType="edit"␣
↪order="40" isRequired="true"/>
        <attr name="idStateHL" caption="Состояние" order="40.2" isRequired="true">
            <editor>
                <lookup changeableAttr="idState" isLookupLazyLoad="true" lookupKeyAttr=
↪"id" lookupListAttr="sHeadLine" lookupQuery="gtk-ru.bitec.app.btk.Btk_ClassStateAvi
↪#Lookup_Class"/>
            </editor>
            <ref class="Btk_ClassState"/>
            <grid columnWidth="12"/>
        </attr>
    </attributes>
</representation>
```

Примечание: Напомню, что Dvi и dvm - это результат работы кодогенератора по данным описанным в разметке odm. Разработчик работает в соответствующих файлах Avi и avm, переопределяя содержимое Dvi и dvm.

ВНИМАНИЕ! Изменять Dvi и dvm нельзя.

А точнее бесполезно, потому что изменения будут перетёрты при следующем запуске кодогенератора.

Нехранимые поля в объектном запросе

Примечание: Учитывает значения из кэша, которые не закоммичены в БД.

onRefreshExt

Данный метод вызывается, если результатом `onRefresh` является набор объектов, а не текст, представляющий реляционный запрос.

С помощью `onRefreshExt` в отображении `Card` получены хэдлайны в `Dvi`:

```
    override protected def onRefreshExt: String = {
      s"""with t as ( select
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      :id as id
     ,:idObjectType as idObjectType
     ,:idState as idState
     ,:gidSrc/*@NString*/ as gidSrc
   )
   SELECT
      t.id
     ,t1.sHeadLine_dz as idObjectTypeHL
     ,t2.sHeadLine_dz as idStateHL
     ,t3.sHeadLine as gidSrcHL
   FROM t
     LEFT JOIN  Btk_ObjectType t1 on t.idObjectType = t1.id
     LEFT JOIN  Btk_ClassState t2 on t.idState = t2.id
     LEFT JOIN  Btk_Object t3 on t.gidSrc = t3.gidRef
   """
   }
```

Здесь используется синтаксис postgresql.

Применяется конструкция with (ссылка).

Через `":"` подставляются текущие значения атрибутов с выборки (в том числе из кэша). Так, например, `":id as id"` означает, что с выборки будет получено текущее значение атрибута `id` и подставлено в таблицу `t` под псевдонимом поля `id`. Теперь обращение в основном запросе `t.id` будет возвращать текущее значеине `id`.

Примечание: Под фразой «текущее значение» понимается значение на момент обновления выборки и выполнения метода onRefreshExt.

Примечание: Описание в dvm не меняется в зависимости от метода получения значения нехранимого поля (реляционного или объектного) и останется таким же, как в случае формирования нехранимого поля в selectStatement.

---

case class AdditionalInfo

Нехранимое поле можно сформировать, переопределив `onRefresh` и `onRefreshItem`.

В этом случае объект представляют как кортеж (`SRop`, экземпляр `case class'a`). Тем самым объект имеет поля записи, описанных в БД (хранимые поля), и поля `case class'a` (нехранимые).

В этом случае принято называть `case class AdditionalInfo`, а заполнение описывать в методе `getAdditionalInfo`.

Нехранимое поле в отображении `Card`:

```
case class AdditionalInfo(
                          var nQtyLoadRecievActs: NNumber,
                          var nQtyLoadTransferCertificate: NNumber,
                          var nQtyLoadTransferCertificateAccounted: NNumber,
                          var nQtyRemains: NNumber,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                            var nQtyReserv: NNumber
                          )

protected def getAdditionalInfo(rop: Oil_ExternalMovApi#ApiRop): AdditionalInfo = {
  AdditionalInfo(
    nQtyLoadRecievActs = None.nn
    , nQtyLoadTransferCertificate = None.nn
    , nQtyLoadTransferCertificateAccounted = None.nn
    , nQtyRemains = None.nn
    , nQtyReserv = None.nn
  )
}

override protected def onRefresh: Recs = {
  val rop = thisApi().load(getVar(CardRep.IdItemSharp).asNLong)
  (rop, getAdditionalInfo(rop))
}

override protected def onRefreshItem: Recs = {
  val rop = thisApi().load(getVar(getPKFieldName).asNLong)
  (rop, getAdditionalInfo(rop))
}
```

Примечание: Переменные внутри case class необходимо указывать var для Avi

Нехранимое поле в отображении `List_idBrigade` коллекции:

```
trait List_idBrigade extends Default with super.List_idBrigade {

  case class AdditionalInfo(var sEmpId: NString = None.ns,
                            var sPosition: NString = None.ns)

  def getAdditionalInfo(rop: Bs_BrigadeStaffApi#ApiRop): AdditionalInfo = {
    if (rop.get(_.idEmployee).isNotNull) {
      val avEmploee = Bs_EmployeeApi().load(rop.get(_.idEmployee)).copyAro()
      AdditionalInfo(
        sEmpId = avEmploee.sEmpId,
        sPosition = avEmploee.sPosition)
    } else AdditionalInfo()
  }

  override protected def onRefresh: Recs = {
    Bs_BrigadeStaffApi().refreshByParent(getVarWithDep("super$id").asNLong)
      .map(rop => {
        (rop, getAdditionalInfo(rop))
      })
  }
}
```

Такой способ самый универсальный.

Например, с помощью `onRefreshExt` не получится посчитать сумму по всем записям коллекции с

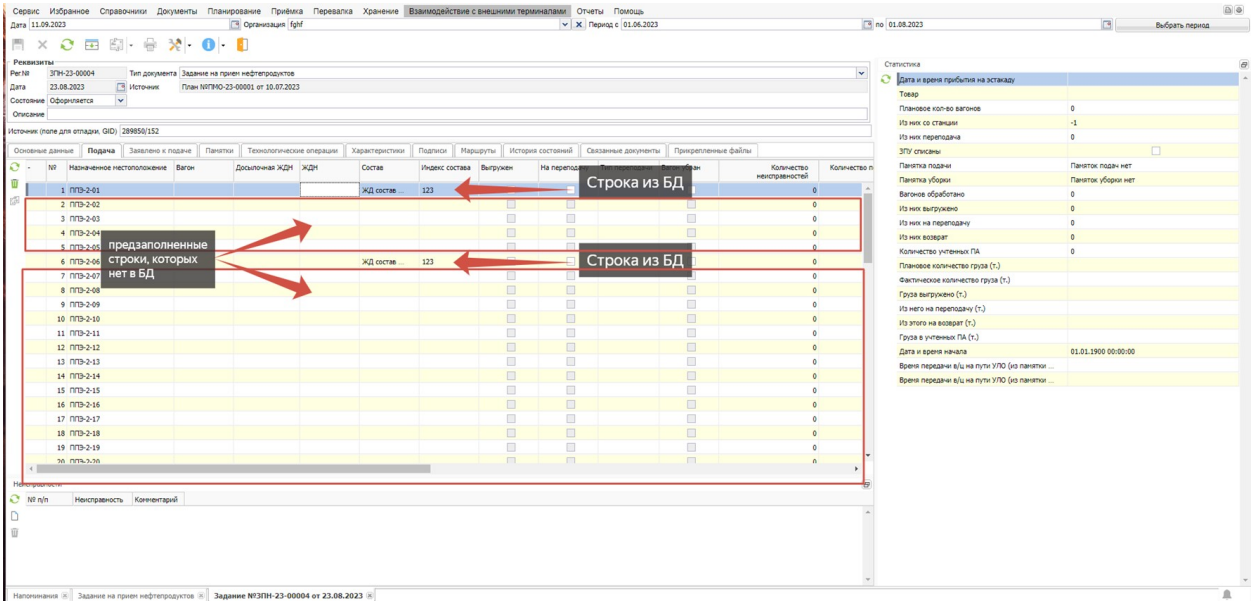
учётом кэша (для этого необходимо присоединить таблицу коллекции по условию `idparent = t.id`, что обеспечит учёт только тех записей и их значений, которые закоммичены в БД, т.е. без учёта кэша).

В рассматриваемой реализации, можно получить все записи коллекции с учётом кэша с помощью метода `refreshByParent(rop.get(_.id))` и далее в обходчике сложить необходимые значения полей.

35.2.4 Нехранимые строки

Здесь будет описан пример реализации пустых предзаполненных строк в списке, как частный случай, в коллекции. Такие строки не хранятся в БД, а существуют визуально в интерфейсе (иными словами нехранимые строки). Запись в БД создаётся при вводе пользователем какого-либо поля.

Пример в проекте pgDev: `ru.bitec.app.oil.Oil_RecievTaskDetAvi.List_idRecievTaskByFlyOverWay`.




```
case class Row
```

По сколку поля строки редактируемые, то не нужно использовать реляционный запрос для формирования выборки. Выборка будет формироваться экземплярами case class'а. Таким образом onRefresh вернёт список экземпляров case class'а.

В примере в коллекции «Подача» имеет столько строк, сколько вагонов указано в записи «Путь эстакады», на который ссылается мастер-документ.

Case class должен иметь такие же поля, как поля класса (таблицы), описанные в odm и нехранимые, формируемые не реляционно.

При добавлении нового хранимого поля, необходимо это поле добавить в case class.

```
/**
 * Представление строки, поля соответствуют хранимым полям строки
 *
 * Полем id выступает idFlyOverPos, т.к. для работы некоторых инструментов (например,␣
↪onRefreshExt) необходимо
 * наличие уникального индификатора
```

(continues on next page)

(продолжение с предыдущей страницы)

```
 */
case class Row(
                var id: NLong //Oil_RecievTaskDet.idFlyOverPos
                , var gid: NGid = None.ng
                , var idRecievTask: NLong
                , var nRow: NNumber
                , var idWagon: NLong = None.nl
                , var idRailwayInvoice: NLong = None.nl
                , var bUnloaded: NNumber = None.nn
                , var bReFeed: NNumber = None.nn
                , var idReFeedType: NLong = None.nl
                , var bWagonRemoved: NNumber = None.nn
                , var bRequiredMeasure: NNumber = None.nn
                , var nQtyMeasure: NNumber = None.nn
                , var idStorageTank: NLong = None.nl
                , var sCertificateList: NString = None.ns
                , var idRecievAct: NLong = None.nl
                , var bRepeatedFeed: NNumber = None.nn
                , var bCommAct: NNumber = None.nn
                , var bRecievActCreated: NNumber = None.nn
                , var idFlyOverPos: NLong = None.nl
                , var idLockDeviceOut: NLong = None.nl
                , var idRecievTaskDet: NLong = None.nl //Oil_RecievTaskDet.id
                , var idTrain: NLong = None.nl
                , var bRecievByMeasure: NNumber = None.nn
              )
```

`onRefresh`

`ru.bitec.app.oil.Oil_RecievTaskDetAvi.List_idRecievTaskByFlyOverWay#onRefresh`.

Описание алгоритма:

- Получаем все записи данной коллекции по мастеру (т.е. те, что хранятся в БД или в кэше).

```
RecievTaskDetApi().refreshByParent(getIdMaster)
```

- Получаем все вагоны выбранной у мастера «Пути эстакады».

```
val ropParent = Oil_RecievTaskApi().load(selection.master.getSelfVar("id").asNLong)
Oil_FlyOverPosApi().txidFlyOverWay.refreshByKey(ropParent.get(_.idFlyOverWay))
```

Примечание: В примере в коллекции «Подача» имеет столько строк, сколько вагонов указано в записи «Путь эстакады», на который ссылается мастер-документ.

- Реализуем обходчик по каждому вагону и заполняем экземпляры case class'а
  - Если на данный вагон есть запись из таблицы БД, то заполняем данными из БД
  - Иначе устанавливаем предзаполненные значения необходимых полей или `None`

```
ropaFOP.map(ropFOP => {
  //поиск ропы в БД
  val ropOpt = ropa.find(_.get(_.idFlyOverPos) === ropFOP.get(_.id))
  getRowByRop(ropOpt, ropFOP, nvRow)
})


protected def getRowByRop(ropOpt: Option[SRop[_ <: JLong, _ <: Oil_
↪RecievTaskDetAro]]
                          , ropFOP: Oil_FlyOverPosApi#ApiRop
                          , npRow: NNumber = None.nn
                         ): Row = {
  Row(
idPlacement = ropOpt.map(_.get(_.idPlacement)).nl
  .nvl(thisApi().getidPlacementByFlyOverPos(ropFOP.get(_.id)))
...)
}
```

При добавлении нового хранимого атрибута, необходимо в этом методе заполнения описать правило заполнения нового поля.

На данном этапе предположим, что `id` заполняется от `ropOpt` (реальной записи), в случае отсутствия `None.nl`. Т.е. пустая строка имеет `id = None.nl`, что дальше будет использоваться, как признак пустой строки.

Примечание: Внимание: такая реализация не совсем корректна. В таком случае не будет работать ряд системных операций, таких, как onRefreshExt, которые требуют уникального значения id. Но для понимания будет рассмотрена такая реализация, а вопрос уникальности id будет рассмотрен далее.

`Row - case class`, представление строки, поля соответствуют хранимым полям записи.

- Возвращаем полученный список экземпляров case class'а.

`insert` при вводе значения в нехранимую строку

Необходимо переопределить операцию `beforeEdit()`.

```
override def beforeEdit(): Unit = {
                  if (getSelfVar("id").isNull) {
                            regRow()
                  }
}
```

`beforeEdit()` вызывается при каждой попытке редактировать поле. Здесь необходимо инициализировать, что заполнение введётся в пустой нехранимой строке или в хранимой. Это можно сделать, проверив заполнен ли `id` (на данном этапе предполагается, что у пустой строки `id = None.nl`).

Если заполнение ведётся в пустой строке, то необходимо создать запись с переносом всех предустановленных значений и введённое пользователем значение применить к только что созданной записи.

Переносить предустановленные значения можно двумя способами:

- Прописать, какой сеттер вызывать и какое значение из case class'а подставлять на каждое поле в отдельности. Такая реализация усложняет поддержание кода. При добавлении нового хранимого атрибута, в случае если оно имеет предустановленное значение, нужно добавлять код в этот фрагмент.
- Реализовать обходчик по всем имеющимся полям case class'а, отсечь поля, которые не имеют сеттеров (`gid`, например) и значение которых `null`, по оставшимся полям вызывать сеттеры и проставлять соответствующие значения:

```
/**
 * insertByParent(ropParent) + сеттеры заполненных полей case class'а Row
 *
 * @param ropParent
 * @param row
 * @return
 */
def insertByParentAndRow(ropParent: Oil_RecievTaskApi#ApiRop, row: Row): ApiRop = {
  insertByParent(ropParent) :/ { rop =>
    //setter'ы для переноса умолчательных значений из Row
    row.getClass.getDeclaredFields.map(_.getName.ns).zip(row.productIterator.to)
      .filterNot(field => saFieldsNotSetter.contains(field._1) || field._2.
↪asInstanceOf[Nullable[_ <: Any, _]].isNull)
      .foreach(field => {
          setAttrValue(rop, field._1, field._2)
      })
    rop
  }
}
```

Чтобы введенное пользователем значение относилось к только что созданной записи, необходимо принудительно после регистрации задать `id` на выборке:

```
val rop = thisApi().registerByRow(thisRow())
setVar("id", rop.get(_.id))
```

`afterEdit()`

Также необходимо добавить проверку, является ли строка пустой, в `afterEdit()`.

Иначе `load()` внутри `afterEdit()` будет вызван по некорректного id пустой строки и будет вызвана ошибка.

Поддержание уникального id в нехранимых полях.

Без уникального `id` ряд системных операций не будет работать или будет работать некорректно, в том числе для работы `onRefreshExt`.

Необходимо понять, что на выборке является уникальным, в случае примера это ссылка соответствующая вагону: `idFlyOverPos` .

Теперь параметр `id` на выборке будет иметь не значение `null` в случае пустой строки и значение записи коллекции из БД, а значение `idFlyOverPos`.

А сам `id` записи коллекции будет храниться в поле case class'а `idRecievTaskDet` Это нужно учесть в:

- Структуре case class'а
- В переносе значений из записи из БД в case class для формирования `onRefresh`
- В условии по признаку пустая ли строка (теперь строка пустая, если `idRecievTaskDet === None.nl`):

```
override def beforeEdit(): Unit = {
  if (getSelfVar("idRecievTaskDet").isNull) {
    regRow()
  }
}
```

- При сеттере в поле пустой строки в методе переноса умолчательных значений:

```
/**
* insertByParent(ropParent) + сеттеры заполненных полей case class'а Row
*
* @param ropParent
* @param row
* @return
*/
def insertByParentAndRow(ropParent: Oil_RecievTaskApi#ApiRop, row: Row): ApiRop = {
  insertByParent(ropParent) :/ { rop =>
    //setter'ы для переноса умолчательных значений из Row
    row.getClass.getDeclaredFields.map(_.getName.ns).zip(row.productIterator.
↪to(scala.collection.immutable.IndexedSeq))
      .filterNot(field => saFieldsNotSetter.contains(field._1) || field._2.
↪asInstanceOf[Nullable[_ <: Any, _]].isNull)
      .foreach(field => {
        if (field._1 == sid.ns) {
          setAttrValue(rop, sidFlyOverPos, field._2)
        } else {
          setAttrValue(rop, field._1, field._2)
        }
      })
    rop
  }
}
```

- При сеттере в поле пустой строки после создания записи проставить параметр выборки `idRecievTaskDet`, заместо `id`, в `id` только что созданной записи:

```
val rop = thisApi().registerByRow(thisRow())
setVar("idRecievTaskDet", rop.get(_.id))
```

- При взятии параметра у дочерних выборок по `super$id` -> `super$idRecievTaskDet`
- В `CWA` управление свойством `isEnabled` у операций:

```
selection.opers().setEnabled("Delete",selection.canDelete && selection.getSelfVar(
↪"idRecievTaskDet").notNull())
```

- Удаление:

```
thisApi().delete(thisApi().load(getSelfVar("idRecievTaskDet").asNLong))
```

- thisRop()

```
thisApi().load(getSelfVar("idRecievTaskDet").asJLong)
```

- onInvalidateItem():

```
override protected def onInvalidateItem(): Unit = {
  if (!getSelfVar(thisApi().sidRecievTaskDet).isNull && thisRop() != null) {
    session.invalidateObject(thisRop())
  }
}
```

- учесть в onRefreshExt() в тексте запроса

35.2.5 Динамическое присоединение столбцов

Если такое отображение только для чтения и результат не использует значения, которые могут быть в кэше, тогда можно реализовать на реляционном запросе, иначе необходимо использовать инструменты DynMetaBuilder и DynRecBuilder.

Реляционная реализация

В этом случае необходимо собрать текст запроса selectStatement.

Ниже пример формирования одного из столбца:

```
s"""
,(select string_agg(cast(t.id as varchar), ', ')
  from Oil_Task t
  where t.idResource  = ${rv.idResource()}
    and (t.dBegin < $sMainFromTab.dEndTime and t.dExec > $sMainFromTab.dBegTime)
    and t.idObjectType = :${Oil_TaskMonitorPkg().sfltidObjectType}
    and t.idStateMC >= 200
    and t.idDepOwner = :super$$idGlobalDepOwner
  group by t.idResource
  ) as \"$sNameFieldForGST[${rv.id()}]\"
"""
```

Пример текста после подстановки значения:

```
(select string_agg(cast(t.id as varchar), ', ')
  from Oil_Task t
  where t.idResource  = 115
    and (t.dBegin < '2023-08-25 12:00:00' and t.dExec > '2023-08-25 14:00:00')
    and t.idObjectType = 225
    and t.idStateMC >= 200
    and t.idDepOwner = 336
  group by t.idResource
  ) as "idFlyOverWay[13]"
```

Мы получим столбец с названием idFlyOverWay[13], результатом будет значение подзапроса.

DynMetaBuilder и DynRecBuilder

Пример: `ru.bitec.app.oil.Oil_TaskMonitorAvi.List_CoreResource`.

Структура формирования:

```
Recs(<объекты, представляющие строку (список rop или case class)>)
        .extend(<DynMetaBuilder>.build())
      .foreach((row, builder) =>  (row, <DynRecBuilder>.build))
```

- Пример использования `Recs`:

  ```
  Recs(SomeApi().refreshByParent(ropMaster))
  ```

  ,где SomeApi() - какая-либо Api класса.

- Пример использования `.extend()`

  Здесь формируются методанные о добавляемых столбцах: название столбца, тип данных, caption.

  ```
  val dynMetaBuilder = DynMetaBuilder()
  dynMetaBuilder.add("idFlyOverWay[1]", classOf[String], "ПЭ-1")
  dynMetaBuilder.add("idFlyOverWay[20]", classOf[String], "ПЭ-2")
  dynMetaBuilder.add("idFlyOverWay[360]", classOf[String], "ПЭ-3")

  Recs(SomeApi().refreshByParent(ropMaster))
  .extend(dynMetaBuilder.build())
  ```

  ,где SomeApi() - какая-либо Api класса.

- Пример использования .foreach()

  Стоит понимать, что это не привычный обходчик по коллекции, который ничего не возвращает, а отдельный метод для динамического формирования столбцов, который должен ВЕРНУТЬ результат выборки!

  В данном методе столбец получает построчно значения.

  ```
  Recs(SomeApi().refreshByParent(ropMaster))
    .extend(dynMetaBuilder.build())
    .foreach((row, builder) =>  {
      builder.set(("idFlyOverWay[1]", row.get(_.sDiscription)))
      builder.set(("idFlyOverWay[20]", "hello, world"))
      builder.set(("idFlyOverWay[20]", None.ns))

      //метод должен вернуть кортеж
      (row, builder.build)
  })
  ```

  ,где SomeApi() - какая-либо Api класса.

  Тут заданы различные значения для столбцов. На каждую строку `row` будет формироваться 3 значения для 3ёх столбцов. Столбец `idFlyOverWay[1]` будет в каждой строке разный, остальные же иметь одинаковое значение.

Описать столбец в разметке `avm` можно по имени поля без квадратных скобок.

35.2.6 Динамическое изменение свойств avm

Для этого необходимо использовать следующий метод:

`ru.bitec.app.gtk.gl.Rep#setMetaProp`

Описание

Пример использования:

```
setMetaProp(attr.sSystemName.get //для какого поля меняется свойство
//какое свойство
//в данном случае свойство задает системное имя атрибута в выборке,
//которое хранит настройки типа редактора
, s"View.Representation.Attributes.Attribute.Editor.editorTypeAttr"меняется
, thisPkg.getSEditorAttrName(attr) //значения свойства
)
```

35.2.7 OnrefreshExt и значение даты с выборки

В `onRefreshExt` нет приведения `/*NDate*/`, нужно писать через `/*NString*/`.

В `onRefreshExt` в конструкции with объявляются поля, которые нужно получить с выборки (из кэша). Иногда требуется указать тип данных. Тип данных указывается в `/*...*/`, но парсер на знает `NDate`, поэтому необходимо указать, как `NString` и далее использовать в запросе `cast(... as timestamp)` или `as date`.

```
override protected def onRefreshExt: String = {
  s"""with t as ( select
       :id as id
      ,:idState as idState
      ,:gidSrc/*@NString*/ as gidSrc
      ,:idObjectType as idObjectType
      ,:idDepOwner as idDepOwner
      ,:idDischargePlace as idDischargePlace
      ,:idSegregationGroup as idSegregationGroup
      ,:idService as idService
      ,:idFlyOverWay as idFlyOverWay
      ,:idStateMC as idStateMC
      ,:dPlanEnd /*@NString*/ as dPlanEnd
      ,:dPlanBegin /*@NString*/ as dPlanBegin
    )
    SELECT
       t.id
      ,to_char(cast(t.dPlanEnd as timestamp) - cast(t.dPlanBegin as timestamp), 'dd д.␣
↪hh24 ч.') as nPlanBusy
      ,t1.sHeadLine_dz as idStateHL
      ,t2.sHeadLine_dz as idObjectTypeHL
      ,t3.sHeadLine_dz as idDepOwnerHL
      ,t4.sHeadLine_dz as idDischargePlaceHL
      ,t5.sMnemoCode_dz as idSegregationGroupHL
      ,t6.sHeadLine_dz as idServiceHL
      ,t7.sHeadLine as gidSrcHL
      ,t8.sHeadLine_dz as idFlyOverWayHL
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    FROM t
      LEFT JOIN  Btk_ClassState t1 on t.idState = t1.id
      LEFT JOIN  Btk_ObjectType t2 on t.idObjectType = t2.id
      LEFT JOIN  Bs_DepOwner t3 on t.idDepOwner = t3.id
      LEFT JOIN  Bs_Placement t4 on t.idDischargePlace = t4.id
      LEFT JOIN  Oil_SegregationGroup t5 on t.idSegregationGroup = t5.id
      LEFT JOIN  Gds_Service t6 on t.idService = t6.id
      LEFT JOIN  Btk_Object t7 on t.gidSrc = t7.gidRef
      LEFT JOIN  Oil_FlyOverWay t8 on t.idFlyOverWay = t8.id
  """
}
```

35.2.8 Поиск отображения на выборке

`selection.form.findSelection(...)`

Ссылка

Полезный метод, который позволяет найти отображение по системному имени среди видимых в сессии.

Пример использования:

```
val sel = selection.form.findSelection(Bdg_ForecastAvi.card())
```

35.2.9 Пользовательская блокировка

Пользовательская блокировка включается при взаимодействии пользователя с интерфейсом в методе `Dvi#beforeEdit`, который является результатом кодогенератора.

Принудительно можно включать блокировку с помощью вызова метода:

- `Btk_FormSessionApi().lockObject(gid)`
- `Btk_FormSessionApi().lockObjectMulti(gida)`

Больше про пользовательскую блокировку можно узнать здесь.

35.2.10 Объект класса в процессе создания и другие состояния объекта rop

Иногда, есть необходимость в Avi знать, что запись находится в процессе создания, т.е. не имеет реализации, как запись в таблице БД.

Это можно узнать по объекту записи `rop`:

- используйте метод Avi `thisRop()` чтобы получить объект `rop`, чья карточка открыта или на котором стоит фокус, если отображение `list`.
- если `rop.ropMode == InsertRopMode`, то объект находится в процессе создания, т.е. есть в кэше приложения, но не имеет реализации в таблице БД.

Где поле объекта `rop.ropMode` хранит в себе информацию состояния объекта. Есть и иные состояния объекта:

- ReadRopMode
- UpdateRopMode

- DeleteRopMode
- InsertRopMode

Пример кода из проекта:

```
    //Документ должен быть закоммичен, чтобы были пройдены все соответсвующие проверки,
    // т.к. по закрытию card_ReadFromFile будет flush()
    if (rop.ropMode == InsertRopMode) {
      throw AppException("Для вызова операции необходимо, чтобы документ был заполенен и
↪сохранён.")
    }
```

35.2.11 Как узнать, что выборка является главным меню или главной выборкой формы

В системе есть 3 типа форм:

- главная
- модальная
- MDI (отображается в качестве закладки на главной форме)

У каждой формы есть главная выборка. Флаг `selection.isMainOnForm` и указывает, что выборка - главная на форме.

Условие `application.mainSelection == selection` определит, является ли выборка главным меню.

35.3 Практика SQL

35.3.1 Функции getmnemocode, getheadline и getattribute

В SQL-запросе заместо получения хэдлайна, мнемокода или иного атрибута через join или подзапрос можно использовать функции `getmnemocode`, `getheadline` и `getattribute`.

```
--getmnemocode
getmnemocode(idpclass bigint, idpobj bigint)

getmnemocode(gid text)

--getheadline
getheadline(idpclass bigint, idpobj bigint)

getheadline(gid text)

--getattribute
getattribute(idpclass bigint, idpobj bigint, spattr text)

getattribute(pgid text, pattr text)
```

Данные процедуры можно использовать, если возвращается немногочисленные объекты, иначе на БД будет большая нагрузка.

Внимание: Нельзя использовать при формировании отчётов или отображения List.

Можно использовать для:

- получения данных через onRefreshExt для отображение Card и ему подобных
- печатных форм
- для разового анализа данных через приложение управления БД (напримеир, dbeaver)

Примеры использования:

```
select    t.sRegNum
        , t.dReg
        , t.idService
        --для id нужно принудительно указать класс
        , getmnemocode(51001,t.idService) idServiceMC
        , getheadline(51001,t.idService) idServiceHL
        --гид можно сразу передать на вход
        , t.gidSrc
        , getmnemocode(t.gidSrc) as gidSrcMC
        , getheadline(t.gidSrc) as gidSrcHL
        --getattribute
        , getattribute('2101', t.idResourceholder, 'gid') gidResourceHolder
        , getheadline(getattribute('2101', t.idResourceholder, 'gid')) as␣
↪idResourceHolderHL
from oil_tankerpos t
```

В разработке расширение методов:

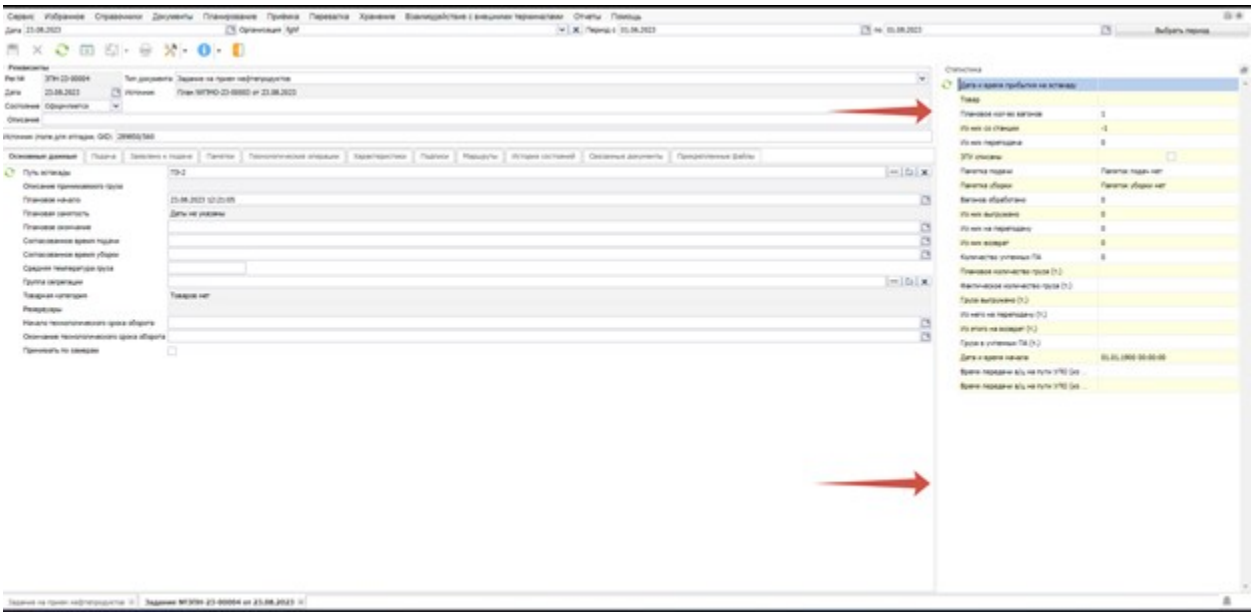
```
--getmnemocode
getmnemocode(sNameTb text, idpobj bigint)

--getheadline
getheadline(sNameTb text, idpobj bigint)
```

35.4 Практика avm, примеры интерфейсов

В данном разделе будет расписана некоторая практика, связанная с разметкой в avm. Будут рассмотрены различные реализации интерфейсов.

35.4.1 Присоединение нескольких отображений (одновременно с закладками и без)



Для этого в разметке нужно выбрать tabDynamicComposer, который позволяет присоединять, как с помощью dynamicItems, так и с помощью tabItems.

Примечание: Напомню, что:

- dynamicItems - обычное присоединение отображения с возможность выбора выравнивания;
- tabItems - присоединение отображения, как вкладка.

```
<representation name="Card" editMode="edit" stdFilter.isAvailable="false">
    <layout>
        <tabDynamicComposer>
            <frame filter.isVisible="false">
                <card/>
            </frame>
            <dynamicItems>
                <dynamicItem selection="gtk-ru.bitec.app.oilOil_RecievTaskAvi
↪"representation="List_Statistics" align="right"/>
            </dynamicItems>
            <tabItems isVisible="true"
                selection="gtk-ru.bitec.app.btk.Btk_ObjectTypeTabAvi"
                representation="List_Tab"
                selection.selectionAttr="SSELECTIONNAME"
                selection.representationAttr="SREPRESENTATIONNAME"
                selection.captionAttr="SCAPTION"
                selection.imageIndexAttr="NIMAGE"
                selection.paramsAttr="JSONPARAMS"/>
        </tabDynamicComposer>
    </layout>
</representation>
```

Примечание: В примере разметки в tabItems присоединены вкладки, настроенные на типе объекта. Это часто встречающаяся практика, данный пример полезно понимать.

35.4.2 Как убрать кнопку сворачивания отображения в модальное окно

Чтобы убрать кнопку сворачивания отображения, присоединенного dynamicItems, необходимо выключить header у frame:

```
<representation name="Card_Body" editMode="edit" stdFilter.isAvailable="false">
    <layout>
        <simpleComposer>
            <frame header.isVisible="false">
                <card/>
            </frame>
        </simpleComposer>
    </layout>
</representation>
```

Таким способом будет убран заголовок с кнопкой.

Заголовок можно вернуть с помощью конструкции `vGroup`.

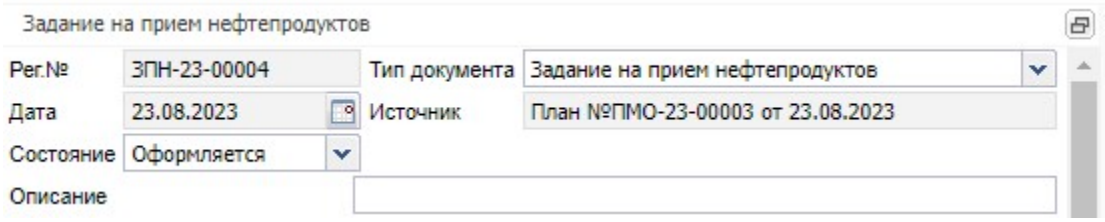
```
<representation name="Card_Body" editMode="edit" stdFilter.isAvailable="false">
    <layout>
        <tabComposer>
            <frame filter.isVisible="false" toolBar.isVisible="false" header.isVisible=
↪"false">
                <card>
                    <layout>
                        <vBox>
                            <vGroup caption="Задание на приём нефтепродуктов">
                                <hBox>
                                    <vBox>
                                        <attr name="sRegNum"/>
                                        <attr name="dReg"/>
                                        <attr name="idStateHL"/>
                                    </vBox>
                                    <vBox>
                                        <attr name="idObjectTypeHL"/>
                                        <attr name="gidSrcHL"/>
                                        <attr name="gidSrc"/>
                                    </vBox>
                                </hBox>
                                <attr name="sDescription"/>
                            </vGroup>
                        </vBox>
                    </layout>
                </card>
            </frame>
            <tabItems isVisible="true"
                      selection="gtk-ru.bitec.app.btk.Btk_ObjectTypeTabAvi"
```

(continues on next page)

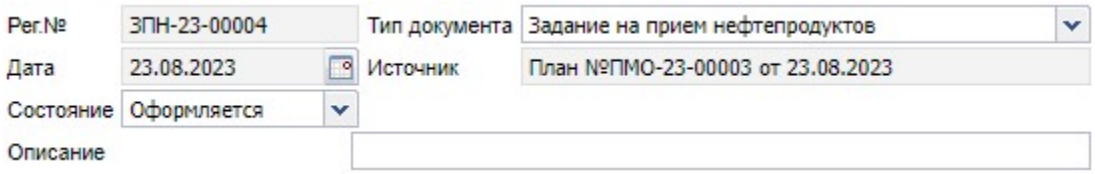
(продолжение с предыдущей страницы)

```
                    representation="List_Tab"
                    selection.selectionAttr="SSELECTIONNAME"
                    selection.representationAttr="SREPRESENTATIONNAME"
                    selection.captionAttr="SCAPTION"
                    selection.imageIndexAttr="NIMAGE"
                    selection.paramsAttr="JSONPARAMS"/>
        </tabComposer>
    </layout>
</representation>
```

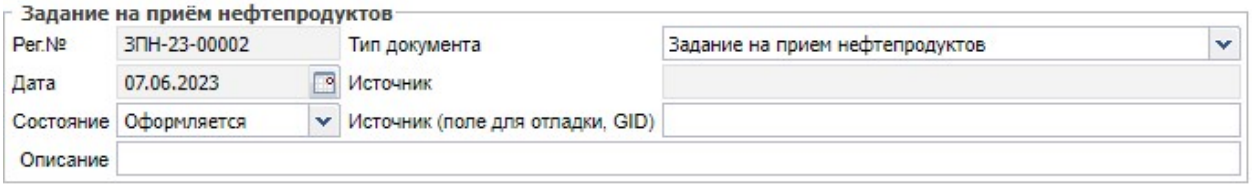
Отображение с кнопкой сворачивания:



Отображение с выключенным header:



Отображение с vGroup:



35.4.3 Динамическая смена отображения

Эта тема отвечает на вопрос, как при изменении значения поля менять присоединяемое отображение.

Примечание: Так работает tabItems на типе объекта: при смене типа объекта набор вкладок в tabItems меняется на те, которые настроены на выбранном типе объекта.

Смена отображений при изменении значения фильтра

Пример реализации:

`ru.bitec.app.oil.Oil_TaskMonitor.avm.xml` - разметка отображения `Card`,

`ru.bitec.app.oil.Oil_TaskMonitorAvi.Card` - результат выборки отображения `Card`.

avm:

```
<representation name="Card">
    <filter name="Oil_TaskMonitorFilter">
        <macros name="DefFltReferenceMacro">
            <condition logicalOperator="and" id="attrs" isExpression="false">
                <filterAttr name="flt_idObjectType" attribute-type="Long" caption="Тип␣
↪задания"
                            isLastInLine="false" order="10" editorType="edit" isVisible=
↪"false"/>
                <filterAttr name="flt_idObjectTypeHL" attribute-type="Varchar" caption=
↪"Тип задания"
                            isLastInLine="false" order="10.2" editorType="lookup">
                    <editor>
                        <lookup lookupQuery="gtk-Btk_ObjectTypeAvi#MainLookup"␣
↪lookupKeyAttr="id"
                                lookupListAttr="sHeadLine" changeableAttr="flt_
↪idObjectType"
                                isLookupLazyLoad="true"/>
                    </editor>
                </filterAttr>
                <filterAttr name="flt_dBegin" attribute-type="Date" caption="Дата, время␣
↪c" isLastInLine="false"
                            order="20" editorType="dateTimePick">
                    <card controlWidth="60" isControlWidthFixed="true"/>
                </filterAttr>
                <filterAttr name="flt_dEnd" attribute-type="Date" caption="по"␣
↪isLastInLine="true"
                            order="30" editorType="dateTimePick">
                    <card controlWidth="60" isControlWidthFixed="true"/>
                </filterAttr>
            </condition>
        </macros>
    </filter>
    <layout>
        <tabDynDetComposer>
            <frame filter.isVisible="true">
                <card/>
            </frame>
            <tabItems selection="gtk-Oil_TaskMonitorAvi" representation="List_RecievTask
↪" isVisible="false"/>
            <dynDetail masterAlign="top" detailAlign="client" selectionAttr=
↪"sSelectionAttr"
                       representationAttr="sRepresentationAttr" paramsAttr="jsonParams"␣
↪isVisible="true"/>
        </tabDynDetComposer>
    </layout>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    <attributes>
        <attr name="sSelectionAttr" isVisible="false"/>
        <attr name="sRepresentationAttr" isVisible="false"/>
        <attr name="sCaptionAttr" isVisible="false"/>
        <attr name="nImageIndexAttr" isVisible="false"/>
        <attr name="jsonParams" isVisible="false"/>
    </attributes>
</representation>
```

Это отображение Card.

В тэге `filter` определены фильтры. Панель фильтров включена свойством тэга `<frame filter. isVisible="true">`.

Присоединение отображений выполнено инструментом `tabDynDetComposer`, который позволяет присоединять как `tabItems` (закладками), так и `dynDetail` (аналогия `dynamicItem`). Закладка `tabItems` присоединена, как пустышка, чтобы заработал `tabDynDetComposer` (проблема была актуальна на 05.07.2023, обещали исправить).

В закладке `dynDetail` определены свойства, значения которых являются атрибутами отображения (см. перечень атрибутов отображения). Тем самым отображение `Card`, как результат выборки должно иметь настройку для `dynDetail`.

Для формирования такой выборки необходимо рассмотреть Avi:

```
trait List_CommonResource extends Default with List_CoreResource {
  override protected val sNameFieldForGST = "Brigade"
  override protected val sNameTableForGST = "Bs_Brigade"
}

trait Card extends Default with super.Card {
  //Типы объекта класса Oil_Task и соответствующие отображения к ним
  lazy val mapRepByOT: Map[NString, NString] =
    Map("Oil_TaskReceive".ns -> "List_RecievTask".ns
      , "Oil_TaskShip".ns -> "List_ShipTask".ns
      , "Oil_TaskCommon".ns -> "List_CommonTask".ns
    )

  @FlushBefore(mode = FlushBeforeMode.Disabled)
  override protected def onRefresh: Recs = {
    val idvOT = getVar("flt_idObjectType").asNLong
    if (idvOT.isNotNull) {
      val sRep = mapRepByOT.getOrElse(Btk_ObjectTypeApi().getMnemoCode(idvOT), throw␣
↪AppException("Не найдено отображение."))
      s"""
        select
            'gtk-Oil_TaskMonitorAvi' as sSelectionAttr
            ,'$sRep' as sRepresentationAttr
            ,'${Btk_ObjectTypeApi().getShortCaption(idvOT)}' as sCaptionAttr
            ,cast(null as numeric) as nImageIndexAttr
            ,'{}' as jsonParams
        """
    } else {
      s"""
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        select cast(null as int8) as id
          ,cast(null as numeric) as nOrder
          ,cast(null as varchar) as sCaptionAttr
          ,cast(null as numeric) as nImageIndexAttr
          ,cast(null as varchar) as sRepresentationAttr
          ,cast(null as varchar) as sSelectionAttr
          ,'{}' as jsonParams
        where false
        """
    }
  }

  override protected def selectStatement: JClob = s""
}
```

У отображения есть фильтр по типу объекта `flt_idObjectType` с редактором `lookup`. При выборе различных типов объекта к `Card` должны присоединяться различные отображения (их соответствие описано в Map `mapRepByOT`).

Алгоритм метода `onRefresh`:

- Проверка, не пустое ли значение фильтра типа объекта

```
val idvOT = getVar("flt_idObjectType").asNLong
if (idvOT.isNotNull)
```

- Получение соответствующего отображения для выбранного типа объекта из `mapRepByOT`:

```
val sRep = mapRepByOT.getOrElse(Btk_ObjectTypeApi().getMnemoCode(idvOT), throw␣
↪AppException("..."))
```

- Формирование выборки:

`sSelectionAttr` - выборка, отображение которого присоединяется (в рассматриваемом случае все отображения принадлежат одной и той же выборке);

`sRepresentationAttr` - соответствующее отображение

`sCaptionAttr` - описание отображения

`nImageIndexAttr` - картинка отображения

`jsonParams` - json параметры

Смена отображений при изменении значения поля отображения

Если нужно менять отображения в зависимости от значения полей отображения `Card`, то нужно сделать промежуточное отображение с механикой, описанной выше (например, `List_Tab`), которое присоединить к `Card`.

35.4.4 Фиксация ширины столбца/поля и задание значения ширины

В карточке:

Необходимо для атрибута задать свойства в тэге `card`:

```
<attr name="sRegNum" caption="Per.№" editorType="edit" order="10" isReadOnly="true">
    <card controlWidth="30" isControlWidthFixed="true"/>
</attr>
```

В списке:

Необходимо для атрибута задать свойства в тэге `grid`:

```
<attr name="sRegNum" caption="Per.№" editorType="edit" order="10">
    <grid controlWidth="30" isControlWidthFixed="true"/>
</attr>
```

`controlWidth` - ширина поля

`isControlWidthFixed` - true - ширина будет зафиксирована, false - доступна для редактирования в интерфейсе.

35.4.5 Убрать описание у поля

Чтобы убрать заголовок у поля, писать `<arrt caption="">` неправильно.

Необходимо задать свойство `labelPosition`:

```
<attr name="idTankerHL" caption="Танкер" order="120.1">
    <editor labelPosition="none"/>
</attr>
```

Помимо значения `"none"` свойство `labelPosition` имееть значения:

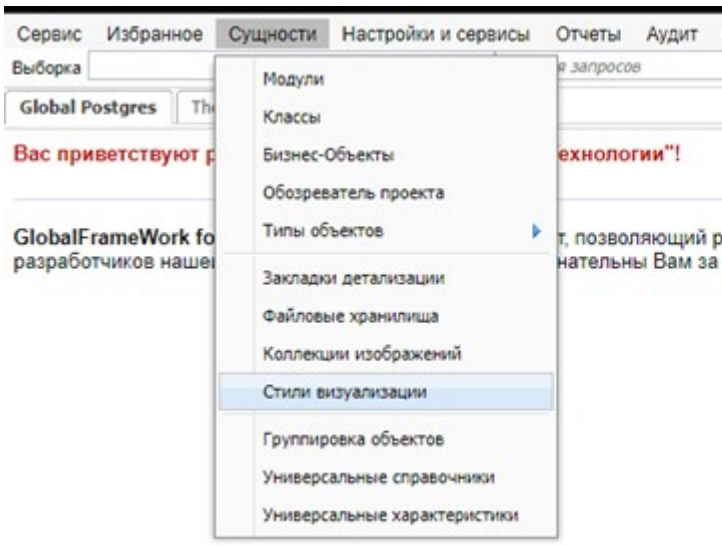
- `"right"` - справа
- `"left"` - слева
- `"above"` - сверху
- `"under"` - снизу

35.4.6 Как в списке покрасить ячейку

Для этого необходимо иметь дополнительный атрибут, в который будет записываться системное имя стиля. Обычно такой атрибут называют `sStyle`.

Далее необходимо указать, что столбец такого-то атрибута имеет стиль, описанный в атрибуте `sStyle`:

```
<attr name="nRating" order="20" caption="Разряд">
    <style attr="sStyle"/>
</attr>
```



Стили можно посмотреть и создать в «Настройках систем»:

35.5 Практика odm

В данном разделе будет расписана некоторая практика, связанная с разметкой в odm.

35.5.1 Свойство not null для атрибута

В разметке odm нет настройки для того, чтобы указать атрибуту свойство not null.

Примечание: Свойство isRequired=»true» влияет на результат работы кодогенератора:

- реализуется настройка в avm для визуального обозначения поля обязательным
- в методе Dpi.validateRequired() реализуется проверка на заполненность обязательных полей.

Но генератор таблиц не смотрит на данную настройку и рассматриваемое поле в таблице не будет иметь признак not null.

Для этого необходимо реализовать sql-скрипт для задания данной настройки для поля:

```
<dbSchema>
    <scripts>
        <script name="isNotNull_sKeyResDepOwnOTArchive" version="1">
            <install>
                ALTER TABLE public.oil_task ALTER COLUMN skeyresdepownotarchive SET NOT␣
↪NULL;
            </install>
        </script>
    </scripts>
</dbSchema>
```

35.6 Практика код

35.6.1 Вычисление суммы без использования буфера

Пример, где подсчитывается сумма по записям коллекции Oil_PlanFactoryShipPos:

```
//получаем все записи коллекции по родителю (объект rop)
Oil_PlanFactoryShipPosApi().byParent(rop)
//получаем только значение поля nQtyLoad
//в случае незаполненности поля необходимо иметь значение 0, иначе в дальнейшем сумма с␣
↪null = null
.map(_.get(_.nQtyLoad).nvl(0.nn))
//сложение, результат которого будет обернут безопасной конструкцией Option
.reduceOption(_ + _)
//распаковка Option
//если внутри был null или в коллекции не было записей, то вернётся указанное значение 0.
↪nn
.getOrElse(0.nn)
```

35.6.2 Группировка объектов с использованием null-типов

При группировке объектов коллекций, в которых используется null-типы нужно учитывать, что метод `groupBy` считает хэши группируемых объектов, поэтому перед группировкой, нужно убедиться, что параметр, используемый для группировки, не равен null.

Пример, с NPE:

```
List(
  asd(123.nl, "asd".ns),
  asd(None.nl, "asd".ns), // У NLong с underlying = null хэш не посчитается
).groupBy(_.id).foreach { case (_, name) =>
  println(name)
}
```

Перед использованием `groupBy` необходимо отфильтровать значения или использовать метод `nvl`.

35.6.3 Сравнение диапазона дат

Допустим документы имеют поля dBeginDoc и dEndDoc.

Фильтр имеет 2 поля dBeginFilter и dEndFilter.

Чтобы найти все документы, которые началом или окончанием входят в диапазон, указанный в фильтре, нужно использовать следующее условие:

```
dBeginDoc <= dEndFilter && dEndDoc >= dBeginFilter
```

35.6.4 .distinct или .toSet для scala-коллекций и особенности применения

Если необходимо в scala-коллекции держать только уникальные объекты, можно использовать методы:

- `.distinct`
- `.toSet`

Перед использованием метода `.distinct` scala-коллекцию необходимо подготовить: убрать значения null. Иначе в ходе выполнения программы выпадет ошибка `java.lang.NullPointerException`.

Пример использования и демонстрация поведения методов:

```
test("distinctOrToSet") {

  val data: Seq[NString] = Seq("h".ns, "i".ns, "i".ns, None.ns, None.ns)

  println("Результат работы .distinct с null внутри scala-коллекции:")
  try {
  println(data.distinct)
  } catch {
    case e: Throwable => println("Ошибка:\n" + "java.lang.NullPointerException")
  } finally {

    println("\nРезультат работы .filter(_.isNotNull).distinct:")
    println(data.filter(_.isNotNull).distinct)

    try {
      println("\nРезультат работы .toSet с null внутри scala-коллекции:")
      println(data.toSet)
    } catch {
      case e: Throwable => println("Ошибка:\n" + "java.lang.NullPointerException\n")
    }
  }
}
```

Результат:

Примечание: Результат работы .distinct с null внутри scala-коллекции: Ошибка: java.lang.NullPointerException

Результат работы .filter(_.isNotNull).distinct: List(h, i)

Результат работы .toSet с null внутри scala-коллекции: Set(h, i, Null)

35.6.5 `immutable.Map.builder` вместо `mutable.Map`

Если по результату сформированной Map она больше не изменяется, то для формирования лучше использовать конструктор `immutable.Map.builder`, чем `mutable.Map`.

Пример:

```
val map = Map.newBuilder[NString, NString]
map ++= Map("1" -> "11")
```

(continues on next page)

(продолжение с предыдущей страницы)

```
map += "2" -> "22"
map.result() //Map("1" -> "11", "2" -> "22")
```

35.6.6 Признак наличия модуля на проекте

```
val isInstallProModule = session.sbtClassLoader.getModuleMap.containsKey("pro")
```

Вернёт true, если такой модуль есть в проекте, иначе false.

35.6.7 Когда использовать ASQL/ ASelect/ OQuery/ TxIndex/ refreshByParent и byParent у коллекций

Про данные инструменты можно почитать *здесь*.

ASQL

Выполнение реляционного запроса на чтение.

Внимание: Вызывает транзакцию в БД, не учитывая значения в кэше.

Удобен для получения значения по одному столбцу результата запроса.

```
    val idaWagonByTrain = ASQL"""
      select string_agg(cast(rwt.idWagon as varchar), ', ')
      from Rzd_TrainWagon rwt
      where rwt.idTrain = $idpTask
    """
    //берется 1ый столбец результата запроса, как NString
    //если результат запроса вернул несколько строк, то будет взята первая в конструкции␣
↪Option,
    // но предполагается, что результат вернёт максимум 1 строку
    .as(nStr(1).singleOpt)
    //если Option пустой, т.е. в результате запроса не было строк, то вернётся None.ns
    .getOrElse(None.ns)
```

`$idpTask` это подстановка значения из переменной scala `idpTask` в запрос связанной переменной (binding).

Внимание: Нельзя использовать внутри цикла. Это приведёт к многочилсенным транзакциям в БД.

Вместо использования `ASQL` внутри цикла с множественными транзакциями в БД нужно перед циклом одной транзакцией сформировать массив данных, который дальше будет использоваться внутри цикла.

ATSQL

Выполнение реляционного запроса с изменением данных или блокировками.

Используется редко, в основном в ядровых процедурах, потому что минует серверную логику, записи в системные миксины, ведение аудитов и другое.

ASelect

Выполнение реляционного запроса на чтение/запись.

> **Внимание:** Вызывает транзакцию в БД, не учитывая значения в кэше.

Используется в основном для чтения, потому что при изменении данных минует серверную логику, записи в системные миксины, ведение аудитов и другое.

Удобен для получения значений по нескольким столбцам результата запроса.

```
val mapPerson: Map[NLong, NString] = new ASelect {
  val sCode = asNString("sCode")
  val id = asNLong("id")
  SQL"""
    select p.id
          ,p.sCode
    from Bs_Person p
    where idObjectType = $idvObjectType
  """
}.map { rv => //rv представляет собой одну строку результата запроса
  rv.id() -> rv.sCode()
}.toMap
```

`$idvObjectType` это подстановка значения из переменной scala `idvObjectType` в запрос связанной переменной (binding).

> **Внимание:** Нельзя использовать внутри цикла. Это приведёт к многочилсенным транзакциям в БД.

Вместо использования `ASelect` внутри цикла с множественными транзакциями в БД нужно перед циклом одной транзакцией сформировать массив данных, который дальше будет использоваться внутри цикла.

Когда использовать ASQL, а когда ASelect

ASQL удобен для получения значения по одному столбцу результата запроса.

ASelect удобен для получения значений по нескольким столбцам результата запроса.

Подстановка связанных переменных (binding)

Актуально для инструментов ASQL, ATSQL, ASelect.

Примечание: Использование ASQL с подстановкой связанных переменных является полезной практикой, потому что запрос остаётся неизменным, меняется лишь значение параметра. Тем самым запрос не воспринимается системой, как новый, и будет записан в системную таблицу запросов единожды, что не приводит к распуханию БД.

Инструмент `ASQL"""<текст запроса>"""` подставляет бинды с учётом типа данных переменной.

- Если бы была подстановка `NString`, то `ASQL` сам обернул бы значение в одинарные кавычки, т.е. нет необходимости их указывать вручную.
- Если `NString` подставляется в текст `s"""<текст запроса>"""`, то одинарные кавычки необходимо указывать вручную.

Если текст запроса для `ASQL"""<текст запроса>"""` собирается динамически средствами scala, в том числе название таблицы для select формируется переменной, то его необходимо подставлять через `#$bind`, чтобы ASQL не обернул подставляемое значение в одинарные кавычки.

```
val sNameTb = {
  if (a = 1) "Bs_Goods".ns
  else "Bs_Person".ns
}

val ida: List[NLong] = ASQL"""
  select t.id
  from #$sNameTb
  where idObjectType = $idvObjectType
""".as(nLong(1).*)
```

35.7 Практики при разработке документов

35.7.1 Odm документа

- Для документов создаем отдельную директорию (пакет), чтобы все коллекции были тоже в ней
- У полей с датами в названии системного имени НЕ пишем слово Date
  пример:
  dReg
  dDoc
  dExec \
- Для поля с датой регистрации, ставим значение по умолчанию текущую дату `defaultValue="sysdate"`
- Для полей с номерами, датой рег., состоянием и номером состояния устанавливаем `isCopyInCopyObject="false"`, чтобы значения не копировались при копировании
- Для полей с номером, датой и типом объекта устанавливаем свойство `isHeadLine="true"`, так как они в большинстве случаем участвуют в вычислении заголовка
- Добавляем поле с номером состояния, не видимое

```
        <attr name="idStateMC" attribute-type="Number" caption="Номер состояния" order=
↪"80" type="basic"
              isVisible="false" isCopyInCopyObject="false" isStateMC="true">
            <mnemoCodeColumn/>
        </attr>
```

- Большинство полей можно взять из шаблона `bs\src\main\resources\META-INF\attribute-template.xml`
  Доп. поля связанные со складом и тмц `stk\src\main\resources\META-INF\attribute-template.xml`
- Для денежных полей с суммами и ценами устанавливаем сразу денежный редактор `editorType="currency"`
- Для булевых полей ставим значение по умолчанию 0 `defaultValue="0"` и тип редактора галку `editorType="check"`
- Добавляем скрипты регистрации состояний, закладок и типов документа (вместо Demo_Doc напишите свой класс)

```
    <dbData>
        <script name="regObjectType" version="1">
            <depends>
                <dep on="Demo_Doc.regTab"/>
                <dep on="Demo_Doc.regState"/>
            </depends>
            <install>Demo_DocApi.regObjectType(false);</install>
        </script>
        <script name="regState" version="1">
            <install>Demo_DocApi.regState();</install>
        </script>
        <script name="regTab" version="1">
            <install>Demo_DocApi.regTab();</install>
        </script>
    </dbData>
```

- Если нужны прикрепленные файлы устанавливаем свойство в шапку `attachType="simple"` или `attachType="versioned"` для версионного режима
- Если нужна настройка по счетам учета, то добавляем в коллекции настройку `<var-collection name="Bs_AccObjSetting" ref.attr="gidSrc" cascadeOnDelete="true"/>`

35.7.2 Odm коллекции документа

- Наименование коллекций, как правило, содержит имя документа (может быть и сокращенное) и суффикс `Det` или какой то другой, например `Demo_DocDet`
- Порядковый номер для позиций следует наименовать как `nRow - № п/п` и делать его числовым, наименование `nOrder` путает с `Bs_Order`. Его не следует делать автонумерующимся, так как нумерация срабатывает на сохранение. Нумерацию можно сделать руками в `Api.insertByParent` (ниже будет пример)

35.7.3 Api документа:

- Вместо `ListBuffer` следует использовать `ArrayBuffer`, так как он меньше занимает места в памяти
- Переопределяем calcHeadLine, для вычисление заголовка с учетом типа документа, номера и даты

```
  override def calcHeadLine(idp: NLong): NString = {
    var svHeadLine = NString()
    if (idp.isNotNull) {
      load(idp) :> { aro =>
        if (aro != null) {
          svHeadLine = s"${Btk_ObjectTypeApi().getShortCaption(aro.idObjectType).nvl(Btk_
↪ClassApi().getHeadLine(aro.idClass))} №" ++ aro.sNumDoc.nvl("") ++ " от " ++ aro.dDoc.
↪toString("dd.MM.yyyy").nvl("")
        }
      }
    }
    svHeadLine
  }
```

- Переопределяем insert для установки типа объекта по умолчанию и других атрибутов (например валюта и тип курса) при создании

```
  override def insert(): ApiRop = {
    val rop = super.insert()
    //установка типа объекта по умолчанию
    setidObjectType(rop, Btk_ObjectTypeApi().getDefaultObjType(idClass))

    //установки даты документа без времени
    //setdDoc(rop, NDate.now().truncate())

    //установка нац. валюты Cur_Currency
    //setidCur(rop, Cur_CurrencySettingsApi().getidCurrencyMain)
    //установка типа курса Cur_CurrencyRateType
    //setidCurRateType(rop, Cur_CurrencySettingsApi().getidCurrencyRateType)

    //установка типа налогообложения Tax_TaxType
    //setidTaxType(rop, Tax_TaxTypeApi().getDefaults)
    //установка ставки налога по умолчанию Tax_TaxRate
    //setidVATRate(rop, Tax_TaxRateApi().getDefaults)

    //установка текущего пользователя Btk_User
    //setidUser(rop, Btk_UserApi().getCurrentUserID)
    //установка текущего физ. лица Bs_Person
    //setidPerson(rop, Bs_PersonApi().getCurrentPersonID)
    //установка текущего сотрудника Bs_Employee
    //setidEmployee(rop, Bs_EmployeeApi().getidCurrentEmployee)

    rop
  }
```

- Переопределяем delete, чтобы нельзя было удалить документ, если он находится не в состоянии `Оформляется`

```
  override def delete(rop: ApiRop): Unit = {
    if (rop.get(_.idStateMC).isDistinct(100.nn))
      throw AppException("Ошибка. Удаление документа возможно только в состоянии \
↪"Оформляется\".")
    //Удаление из журнала с признаками проведения в учетах
    //Bs_FlagAccKindBookDocApi().delByObj(rop.gid)
    //Удаление из журнала связанных документов
    //Bts_DocLinkApi().delByDoc(rop.gid)
    super.delete(rop)
  }
```

- Если есть настройка по счетам учета, то на установку типа объекта и организации прописываем установку счетов учета для документа

```
  override def setidDepOwner(rop: ApiRop, value: NLong): Unit = {
    super.setidDepOwner(rop, value)
    Bs_AccObjSettingApi().registerAccObjSetByObjectForDoc(
      gidpObject = rop.gid
    )
    //если есть КПП организации, то устанавливаем его
    //setidSelfKpp(rop, Bs_DepOwnerApi().getIdDefKpp(value))
  }

  override def setidObjectType(rop: ApiRop, value: NLong): Unit = {
    super.setidObjectType(rop, value)
    Bs_AccObjSettingApi().registerAccObjSetByObjectForDoc(
      gidpObject = rop.gid
    )
  }
```

- Для setidState в прикладной логике надо использовать номера состояний, а на их id. Пример действий на перевод состояний:

```
  //проверки для документа при выполнении
  private def validateOnAccept(rop: ApiRop): Unit = {

  }

  //проверки для документа при откате
  private def validateOnDecline(rop: ApiRop): Unit = {
    //проверка на наличие проведения в бух., нал., упр. учетах
    //Bs_FlagAccKindBookDocApi().validateObj(rop.gid)
  }

  //действия на выполнен
  private def accept(rop: ApiRop): Unit = {
    //если не заполнена дата исполнения, то заполняем ее текущей датой
    //if (rop.get(_.dExec).isNull)
    //  setdExec(rop, NDate.now())
  }

  //действия на откат из выполнен
  private def decline(rop: ApiRop): Unit = {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  }

  override def setidState(rop: ApiRop, value: NLong): Unit = {
    //номера состояний
    //из какого
    val nvStateFrom = rop.get(_.idStateMC) //если нет поля можно использовать Btk_
↪ClassStateApi().getOrder(rop.get(_.idState))
    //в какое
    val nvStateTo = Btk_ClassStateApi().getOrder(value)

    //переход в выполнен (вверх по состоянию)
    if (nvStateTo >= 300.nn && nvStateFrom < 300.nn) {
      //проверки для документа при выполнении
      validateOnAccept(rop)
      //действия на выполнен
      accept(rop)
    } else if (nvStateTo < 300.nn && nvStateFrom >= 300.nn) {
      //проверки для документа при откате
      validateOnDecline(rop)
      //действия на откат из выполнен
      decline(rop)
    }

    super.setidState(rop, value)
    //добавление в очередь отложенного проведения документов
    //Bs_DocTransQueueApi().addToQueue(rop.gid, rop.get(_.idObjectType), nvStateFrom,␣
↪nvStateTo)
  }
```

- Различные проверки должны выполняться, как правило, в начале перевода состояния, чтобы было как можно меньше логики выполнено, до получения ошибки
- Если у документа есть коллекции, а у коллекций есть еще коллекции, то на перевод состояния сначала можно выполнить загрузку в кэш

```
  /**
    * прогрузка в кэш самого документа с коллекциями
    *
    * @param idap
    */
  private def loadDocWithCollections(idp:NLong): Unit = {
    new OQuery(entityAta.Type) {
      where(t.id === idp)
      batchIn(Demo_DocDetAta, Demo_DocDetItemAta, Stk_OperationAta)
    }.foreach { rop => }
  }
```

- Создаем метод `regState` для регистрация состояний (здесь просто написан пример)

```
  def regState(): Unit = {
    session.commit()
    Btk_Pkg().setRWSharedUOWEditType()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  //первоначальное состояние
  Btk_ClassStateApi().register(
    idpMasterClass = idClass,
    spSystemName = "Create",
    spCaption = "Оформляется",
    bpStartState = 1.nn,
    npOrer = 100.nn
  )
  session.flush()
  Btk_ClassStateApi().register(
    idpMasterClass = idvClass,
    spSystemName = "Annulled",
    spCaption = "Аннулирован",
    bpStartState = 0.nn,
    npOrer = 50.nn)
  Btk_ClassStateApi().register(
    idpMasterClass = idClass,
    spSystemName = "Agreed",
    spCaption = "Согласуется",
    bpStartState = 0.nn,
    npOrer = 200.nn)
  Btk_ClassStateApi().register(
    idpMasterClass = idClass,
    spSystemName = "Executed",
    spCaption = "Выполнен",
    bpStartState = 0.nn,
    npOrer = 300.nn
  )
  session.commit()
}
```

- Создаем метод `regTab` для регистрация закладок (здесь просто написаны примеры)

```
//регистрация закладок
def regTab(): Unit = {
  session.commit()
  Btk_Pkg().setRWSharedUOWEditType()

  Btk_TabApi().register(
    spCaption = "Позиции",
    spSel = "gtk-Demo_DocDetAvi",
    spRep = "List_idDoc",
    idprefClass = idClass
  )

  Btk_TabApi().register(
    spCaption = "Прикрепленные файлы"
    , spSel = "gtk-Btk_AttachItemAvi"
    , spRep = "List_SimpleAttach"
    , idprefClass = idClass
  )
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    Btk_TabApi().register(
      spCaption = "Объектные характеристики",
      spSel = "gtk-Demo_DocAvi",
      spRep = "Card_ObjectAttr",
      idprefClass = idClass
    )

    Btk_TabApi().register(
      spCaption = "Связанные документы"
      , spSel = "gtk-Bts_DocLinkAvi"
      , spRep = "Tree"
      , idprefClass = idClass
    )

    Btk_TabApi().register(
      spCaption = "Проводки",
      spSel = "gtk-Act_TransAvi",
      spRep = "List_gidDoc",
      idprefClass = idClass
    )

    Btk_TabApi().register(
      spCaption = "Настройка счетов учета",
      spSel = "gtk-Bs_AccObjSettingAvi",
      spRep = "List_gidSrc",
      idprefClass = idClass
    )
    session.commit()
  }
```

- Создаем метод `regObjectType` для регистрация типов объекта (здесь просто написан пример)

```
  //регистрация типов
  def regObjectType(bpNeedRefresh: Boolean = false): Unit = {
    session.commit()
    Btk_Pkg().setRWSharedUOWEditType()
    //параметр bpNeedRefresh = false нужен, чтобы не затирать данные на проектных базах,␣
↪если такого типа объекта нет, то он будет создан
    //если хотите все перезаписать, то выполните в Jexl скрипте Demo_DocApi.
↪regObjectType(true);
    if (bpNeedRefresh || Btk_ObjectTypeApi().findByMnemoCodeAndClass("Demo_DocObjType",␣
↪idClass).isNull) {
      //регистрируем тип
            val idvOT = Btk_ObjectTypeApi().register(spCode = "Demo_DocObjType"
        , spCaption = "Тип А"
        , spShortCaption = "Документ А"
        , idpRefClass = idClass
        , bpIsDefault = 1.nn)

            //ищем закладку и привязываем к типу
      var idvTab = Btk_TabApi().findByMnemoCode("Demo_DocAvi.List_idDoc")
      if (idvTab.isNotNull)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        Btk_ObjectTypeTabApi().registerTab(
          idpObjectType = idvObjectType
          , idpTab = idvTab
          , spCaption = "Позиции"
          , npOrder = 10.nn
        )

      idvTab = Btk_TabApi().findByMnemoCode("Demo_DocAvi.Card_ObjectAttr")
      if (idvTab.isNotNull)
        Btk_ObjectTypeTabApi().registerTab(
          idpObjectType = idvObjectType
          , idpTab = idvTab
          , spCaption = "Характеристики"
          , npOrder = 20.nn
        )

            //регистрируем переходы состояний для типа
      Btk_StateChangeApi().registerForObjectType(idvOT, List(
        "Create" -> "Agreed",
        "Agreed" -> "Executed",
        "Agreed" -> "Create",
        "Executed" -> "Create",
        "Executed" -> "Agreed"
      ))
    }
    session.commit()
  }
```

35.7.4 Api коллекции

- Переопределяем insertByParent для установки порядкового номера при создании

```
  override def insertByParent(ropParent: AnyRop): ApiRop = {
    val nvRow = byParent(ropParent).map(_.get(_.nRow).nvl(0.nn)).reduceOption(_ max _).
↪getOrElse(0.nn) + 1.nn
    super.insertByParent(ropParent) :/ { rop =>
      setnRow(rop, nvRow)
      rop
    }
  }
```

35.7.5 Avi документа:

для отображения List

- Переопределяем `onRefreshItem`, чтобы выполнялся запрос из `selectStatement`

```
override protected def onRefreshItem: Recs = {
  prepareSelectStatement(s"t.$getPKFieldName = :$getPKFieldName")
}
```

- Переопределять `onRefreshExt` в списках не нужно, так как весь запрос должен быть в `selectStatement`
- Переопределяем CWA, чтобы дописать блокировку кнопки удаления, если документ находится не в состоянии оформляется

```
override def checkWorkability(): Unit = {
  super.checkWorkability()
  val bvRO = getSelfVar("idStateMC").asNNumber.isDistinct(100.nn)
  opers().setEnabled("Delete", selection.canDelete && !getVar("ID").isNull && !bvRO)
}
```

- Если нужно передать параметры в карточку при создании из списка, надо переопределить метод `insert_Params`, и затем обработать данный параметр в отображении для карточки в методе `onInsertItem`.
  Пример передачи организации из фильтра списка.

```
override def insert_Params(): Map[String, Any] = {
  Map("idDepOwner#" -> getVar("flt_idDepOwner").asNLong)
}
```

- Если нужно сделать фильтр по состоянию или типу, надо добавить на `beforeFirstOpen` переменные со значением id нашего класса, чтобы работали стандартные вып. списки для типа и состояния

```
override protected def beforeFirstOpen(): Unit = {
  super.beforeFirstOpen()
  addVar("idStateClass#", thisApi().idClass, FieldType.ftInteger) // для фильтра по␣
↪состоянию
  addVar("idObjTypeClass", thisApi().idClass, FieldType.ftInteger) // для фильтра по␣
↪типу
}
```

- Для установки значений по умолчанию при открытии для полей в фильтре с датами и организации, значения которых будут браться из глобального фильтра, надо использовать значения по умолчанию через Avm, а не через `beforeFirstOpen`

для отображения Card - шапки

- Переопределяем `onInsertItem` для установки организации из глобального фильтра при создании

```
  override protected def onInsertItem(): Unit = {
    super.onInsertItem()
    //если был передана организация через параметр
    val idvDepOwner = getSelfVar("idDepOwner#").asNLong.nvl(getVar("super
↪$idGlobalDepOwner").asNLong)
    if (idvDepOwner.isNotNull) thisApi().setidDepOwner(thisRop(), idvDepOwner)
  }
```

- Если есть закладки или детальные формы, то переопределяем refresh, чтобы при нажатии обновить обновлялись детали тоже

```
  override def refresh(): Unit = {
    super.refresh()
    selection.refreshDetails()
  }
```

- Переопределяем `saveForm`, чтобы прописать обновление карточки, если есть автонумерующиеся атрибуты

```
  @Oper(refreshAfter = true)
  override def saveForm(): Unit = super.saveForm()
  //или так
  override def saveForm(): Unit = {
    super.saveForm()
    selection.refreshItem()
  }
```

- Если нужна операция с молоточком и ключом, то устанавливаем для нее активность

```
  @Oper(active = true)
  override def extraOperations() = super.extraOperations()
```

- Переопределяем `setidState`, чтобы дописать различные действия

```
  override def setidState(event: SetterEvent): Unit = {
    super.setidState(event)

    //завершение транзакции
    session.commit()
    //снятие блокировки
    Btk_FormSessionApi().closeLockUnit()

    //обновление карточки
    selection.refreshItem()
    //вызов checkWorkability в карточке
    selection.checkWorkability()
    //вызов checkWorkability в деталях
    selection.cwaDetails()
    //если нужно обновление деталей вместо cwaDetails используем selection.
↪refreshDetails
  }
```

- Переопределяем CWA, для блокировки полей и операций в зависимости от состояния

```
//для управлением редактируемостью атрибутов
private def setAttrReadOnly(): Unit = {
  val avAttrsNotApply = Set("DREG", "SREGNUM", "IDSTATE", "IDSTATEHL")
  val bvStNotForm = thisRop().get(_.idStateMC).isDistinct(100.nn)
  attrs().filterNot(f => avAttrsNotApply.contains(f.name.toUpperCase))
    .foreach(_.isReadOnly = bvStNotForm)
}

//для управлением видимостью атрибутов
private def setAttrVisible(): Unit = {
  //(A.idObjectType + "HL").isVisible = false
}

//для управлением активность операций
private def setEnabledOpers(): Unit = {
  val bvStNotForm = thisRop().get(_.idStateMC).isDistinct(100.nn)
  val bvStAcc = thisRop().get(_.idStateMC) >= 300.nn
  opers("fillDoc").isEnabled = !bvStNotForm
  opers("clonePrint").isEnabled = bvStAcc
}

override def checkWorkability(): Unit = {
  super.checkWorkability()
  //для управлением редактируемостью атрибутов
  setAttrReadOnly
  //для управлением видимостью атрибутов
  setAttrVisible
  //для управлением активность операций
  setEnabledOpers
}
```

- реляционные запросы в onRefresh в карточках не следует использовать, так как если будет написан запрос, то перед выполнением произойдет сразу СОХРАНЕНИЕ. Все доп. поля пишем либо через onRefreshExt, additionalInfo или case class.
- если переопределяете LookUp или пишите свои вып. списки, не забывайте дописывать `@FlushBefore(mode = FlushBeforeMode.Disabled)`, так как это тоже приводит к коммиту при вызове запроса

```
import ru.bitec.app.gtk.gl.{FlushBefore, FlushBeforeMode}

@FlushBefore(mode = FlushBeforeMode.Disabled)
override protected def onRefresh: Recs
```

35.7.6 Avi коллекции

- Переопределяем onRefresh, чтобы добавить сортировку по №п/п

```
override def onRefresh: Recs = {
  thisApi().refreshByParent(getIdMaster).toList.sortBy(_.get(_.nRow))
}
```

- реляционные запросы в onRefresh в коллекциях не следует использовать, так как если будет написан запрос, то перед выполнением произойдет сразу СОХРАНЕНИЕ. Все доп. поля пишем либо через onRefreshExt, additionalInfo или объектный `List[case class]`.

- Переопределяем CWA, для блокировки полей и операций, в зависимости от состояния

```
//для управлением редактируемостью атрибутов
private def setAttrReadOnly(): Unit = {
  val bvStNotForm = getVar("super$idStateMC").asNNumber.isDistinct(100.nn)
  attrs().foreach(_.isReadOnly = bvStNotForm)
}

//для управлением видимостью атрибутов
private def setAttrVisible(): Unit = {
  //(A.idGds + "HL").isVisible = false
}

//для управлением активность операций
private def setEnabledOpers(): Unit = {
  val bvStNotForm = getVar("super$idStateMC").asNNumber.isDistinct(100.nn)
  opers("insert").isEnabled = getIdMaster.isNotNull && !bvStNotForm
  opers("delete").isEnabled = !bvStNotForm && !A.id.isNull
  opers("copyObject").isEnabled = !bvStNotForm && !A.id.isNull
}

override def checkWorkability(): Unit = {
  super.checkWorkability()
  //для управлением редактируемостью атрибутов
  setAttrReadOnly
  //для управлением видимостью атрибутов
  setAttrVisible
  //для управлением активность операций
  setEnabledOpers
}
```

35.7.7 Avm документа

- Для атрибута тип документа в отображении Default устанавливаем тип редактора - выпадающий список, чтобы отображались типы только нашего класса

```
          <attr name="idObjectTypeHL" caption="Тип" order="50.2" editorType="lookup"␣
↪isLastInLine="false" isVisible="true">
              <editor>
                  <lookup lookupQuery="gtk-Btk_ObjectTypeAvi#MainLookup"␣
↪isLookupLazyLoad="true"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                            changeableAttr="idObjectType" lookupKeyAttr="id"␣
↪lookupListAttr="sHeadLine"
                            isResetButtonVisible="true"/>
                </editor>
            </attr>
```

- Для атрибута организация в отображении Default устанавливаем тип редактора - выпадающий список

```
            <attr name="idDepOwnerHL" caption="Организация" order="70.2" isRequired="true
↪">
                <editor>
                    <lookup lookupQuery="gtk-Bs_DepOwnerAvi#Lookup"
                            changeableAttr="idDepOwner" isLookupLazyLoad="false"
                            lookupKeyAttr="id" lookupListAttr="sHeadLine"␣
↪isResetButtonVisible="false"/>
                </editor>
            </attr>
```

- Устанавливаем закладки от типа документа, по умолчанию видимые в карточке и не видимые в списке (`isVisible="false"`)

```
<tabItems isVisible="true" selection="gtk-Btk_ObjectTypeTabAvi"
                          representation="List_Tab"
                          selection.selectionAttr="SSELECTIONNAME"
                          selection.representationAttr="SREPRESENTATIONNAME"
                          selection.captionAttr="SCAPTION"
                          selection.imageIndexAttr="NIMAGE"
                          selection.paramsAttr="JSONPARAMS"
                />
```

- Если есть поля в фильтре, значения которых должны браться из глобального фильтра (например по датам и организации), то прописываем значения по умолчанию через `defaultValue`

```
                <!--Период с-->
                <condition id="filterFrom" logicalOperator="and" isExpression="true"
                           expression=":flt_dFrom &lt;= t.dPeriod">
                    <filterAttr attribute-type="Date" name="flt_dFrom" editorType=
↪"datePick" order="10.0"
                                caption="Период с" isLastInLine="false" defaultValue=
↪"super$DGLOBALBEGINDATE">
                        <card controlWidth="24" isControlWidthFixed="true"/>
                    </filterAttr>
                </condition>
                <!--Период по-->
                <condition id="filterTo" logicalOperator="and" isExpression="true"
                           expression=":flt_dTo &gt;= t.dPeriod">
                    <filterAttr attribute-type="Date" name="flt_dTo" editorType="datePick
↪" order="20.0" caption="по"
                                isLastInLine="false" defaultValue="super$DGLOBALENDDATE">
                        <card controlWidth="20" isControlWidthFixed="true"/>
                    </filterAttr>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                </condition>
                <!--План счетов-->
                <condition logicalOperator="and" id="filterIdAdjustMethod" isExpression=
↪"true"
                           expression="t.idAdjustMethod = :flt_idAdjustMethod">
                    <filterAttr name="flt_idAdjustMethod" attribute-type="Long"␣
↪isVisible="false" order="30"
                                defaultValue="super$idGlobalAdjustMethod"/>
                    <filterAttr name="flt_idAdjustMethodHL" attribute-type="Varchar"␣
↪caption="План счетов" order="30.1"
                                editorType="lookup" isLastInLine="false">
                        <editor>
                            <lookup lookupQuery="gtk-Bs_AdjustMethodAvi#Lookup"
                                    lookupKeyAttr="id" lookupListAttr="sHeadLine"␣
↪changeableAttr="flt_idAdjustMethod"
                                    isLookupLazyLoad="false" isResetButtonVisible="true"/
↪>
                        </editor>
                    </filterAttr>
                </condition>
                <!--Организация-->
                <condition logicalOperator="and" id="filterIdDepOwner" isExpression="true
↪"
                           expression="t.idDepOwner = :flt_idDepOwner">
                    <filterAttr name="flt_idDepOwner" attribute-type="Long" caption=
↪"Организация" order="50"
                                defaultValue="super$idGlobalDepOwner" isVisible="false"/>
                    <filterAttr name="flt_idDepOwnerHL" attribute-type="Varchar" caption=
↪"Организация" order="50.2"
                                editorType="lookup" isLastInLine="false">
                        <editor>
                            <lookup lookupQuery="gtk-Bs_DepOwnerAvi#Lookup"
                                    changeableAttr="flt_idDepOwner" isLookupLazyLoad=
↪"false"
                                    lookupKeyAttr="id" lookupListAttr="sHeadLine"␣
↪isResetButtonVisible="true"/>
                        </editor>
                    </filterAttr>
                </condition>
```