GsfWebExtensionsGuide

Выпуск 0.0.29

"Бизнес Технологии" ООО

апр. 02, 2025

Содержание

1 Предисловие

Данное руководство описывает создание веб расширений.

Веб расширение позволяет отображать данные фрейма или компоновать выборки произвольным образом.

Руководство предназначено для прикладных разработчиков системных модулей.

2 Введение

2.1 Веб расширение

Отвечает за отображение данных выборки в браузере и взаимодействие с пользователем. Расширение состоит из:

- Бэкэнд компонента
  Обрабатывает бизнес локику компонента на стороне сервера.
- Франтед компонента
  Обрабатывает бизнес логику компонента в браузере.

Типы расширений

Контрол

Расширение отображающее данные фрейма

Для реализации смотри: `btk/src/web/gs-web-dtk/src/common/control.ts`

Компоновщик

Расширение компонующий данные выборки, управляет координатами и размерами фрейма, а так же дочерних выборок.

Для реализации смотри: `btk/src/web/gs-web-dtk/src/common/composer.ts:8`

2.2 Структура веб расширения

Веб расширение состоит из:

- *Клиентской части*
  Ресурсов доступных из браузера
  - Обязательно:
    * main.js
      javascript bundle содержащий в себе es module расширения для клиента. Данный бандл создается с помощью nodejs модуля.
  - Дополнительно:
    * Изображения

    * Html страницы
    * Файлы доступные по url из браузера

- *Серверной части*
  Бизнес логики на сервере
  - Выборка(AVI)
    Содержит данные и пользовательские операции
  - Компонент расширения (ABI – application backend interface)
    Содержит правила отображения и обновления данных выборки

2.3 Nodejs проект

Nodejs проект представляет из себя набор файлов редактируемый vscode с помощью которого создается javascript bundle es модуля который загружается в браузера для выполнннения бизнес логики веб расширения на стороне клиента

Входной точкой для сборки модуля является файл `btk/src/web/ext/packages/{extension_name}/src/main.ts` Данный файл экспортирует контекст модуля:

```
export default new ModuleContext()
```

Контекст модуля

Содержит функции:

- Создание контрола
- Создания компоновщика

Для реализации смотри: `btk/src/web/gs-web-dtk/src/common/module.ts`

2.4 Создания веб расширения контрола

1. Создайте проект `nodejs`
   Проект должен быть создан в папке `btk/src/web/ext/packages`.

   Подробности смотрите в `btk/src/web/ext/doc/CONTRIBUTING.md`
2. Создайте модуль в `nodejs` проекте
   Модуль представляет из себя входной файл для компеляции `packages/dashboard/src\main.ts` который реализовывает контекст модуля
3. Создайте контрол на фратэнд
   Пример: `btk/src/web/ext/packages/dashboard/src/control.ts`
4. Реализуйте фабрику контрола в контексте модуля
5. Создайте бэкэнд контрола
   Пример: `ru.bitec.app.btk.dashboard.Btk_DashboardControlAbi`
6. Создайте трейт расширения для выборки реализующий типизированное взаимодействие между abi и avi `ru.bitec.app.btk.dashboard.Btk_DashboardAbsAvi`
7. Объявите расширение в файле `btk/src/main/resources/META-INF/web-extensions.xml`
   Пример:

```
<ext name="DashboardControl" backendClass="ru.bitec.app.btk.dashboard.Btk_
↪DashboardControlAbi" frontendUri="btk/dashboard/web/main.js"/>
<ext name="DashboardComposer" backendClass="ru.bitec.app.btk.dashboard.Btk_
↪DashboardComposerAbi" frontendUri="btk/dashboard/web/main.js"/>
```

2.5 Подключение веб расширения к выборке

1. Создайте выборку
2. Унаследуйте выборку от трейта расширения
3. В отображения выборки добавьте в фрейм наименование контрола

Пример:

```
<simpleComposer>
    <frame filter.isVisible="false">
        <extControl name="btk/Dashboard"/>
     </frame>
</simpleComposer>
```

3 Взаимодействие с сервером

3.1 Модель данных

Позволяет передавать состояние для отображения с сервера на клиенте.

Модель данных интегрировано с событиями react таким образом, react компоненты могут подписываться на изменения модели данных.

К примеру модель данных для дашборда:

- Таблица
  - Колонки
    - Колонка
      - Атрибуты колонки
        - Ширина
  - Строки
  - Ячейки
    - Ячейка
      - Атрибуты ячейки
        - Видимость
        - Текст для отображения
        - Количество строк
        - Количество колонок

События модели на который можно подписаться на стороне клиента:

- На изменения свойств
- На изменение дочерних элементов
- На перестроение узла
  При перестроение узла на клиенте, событие приходит только в узел перестроения, дочернии элементы должны быть обновлены в результате перестроения узла и события по ним не приходят. Это позволяет ускорить массовое перестроение элементов без дос атаки событиями.

Действия с моделью на стороне сервера:

- Создать узел модели
- Изменить свойство узла модели
- Сменить ветку узла модели
  При этом на стороне клиента произойдет событие изменения дочерних элементов для родительского узла.
  Данная операция позволяет гарантировать согласованное изменение данных на клиенте, так как для данного узла будет создана отдельная копия данных для отображения(при этом по сети будут переданы только изменения).

Местоположение данных

На сервере

Расположение данных на сервере позволяет упростить написание компонентов так как механизм репликации изменений автоматизирует процесс синхронизации состояний на клиенте и на сервере. Однако подобный механизм применим только в случаи если компонент на клиенте поддерживает частичную загрузку данных.

На клиенте

Используется в случаи если компонент на клиенте не может быть интегрирован в реплицируемую модель данных. В таком случаи модель данных содержит только версию для отображения, по которой компонент на модели понимает о необходимости запросить данные в случаи если произошли изменения.

При таком взаимодействии данные на клиент приходят только по запросу клиента:

- на пользовательские действия
- на обработку изменений модели

3.2 Датасет

Позволяет хранить изменения модели данных и отправлять их на сервер. Датасет интегрирован в модель данных и на прямую используется только при отправки изменений на сервер. Пример отправки смотрите в разделе Данные ввода.

3.3 Данные ввода

Позволяет передавать изменения с клиента на сервер.

Данные ввода попадают `Abi.onInputData`

Данные для ввода синхронизируются с серверной частью перед выполнением любой синхронной операции.

Синхронными операциями являются:

- Выполнение операции выборки
- Синхронный запрос

Для синхронизации данных ввода, с сервером используется понятие маркер ввода.

3.4 Маркер ввода

Маркер ввода позволяет накапливать изменения до момента синхронной операции, перед которой все изменения будут отправлены в бэкэнд. Таким образом гарантируется синхронизация данных без необходимости немедленной отправки данных.

Маркер ввода по своей сути повторяет концепцию фокуса ввода, однако есть ряд отличий:

- Маркер ввода поддерживает асинхронные методы для операций.
- Маркер ввода, учитывается на уровне всего компонента
- Маркер ввода запрашивается перед началом редактирования, в то время как фокус ввода меняется при любой навигации

Запрос маркера ввода

Перед начало изменения данных ввода, контрол обязан запросить маркер ввода. При этом:

1. Если маркер ввода находится в другом контроле
    1. Происходит запрос маркера ввода с другого контрола
    2. Если другой контрол отменяет запрос, происходит возврат ошибки
       При этом инициатор запроса должен отменить текущую операцию
    3. Если другой контрол готов вернуть маркер ввода
        1. Формируются данные ввода для синхронизации с серверной частью для другого контрола
        2. Данные ввода синхронизируются с сервером
        3. Новый маркер ввода возвращается в запросивший контрол.

Пример установки значения в контексте маркера ввода:

```
<input type="checkbox"
       checked={cell.getAttrValue("isSelected")}
       onChange={function(event){
          controller.withInputMarker(function(){
            cell.setAttrValue("isSelected",!event.target.checked)
          }).catch(function(){
            console.log("cancel input")
```

(continues on next page)

(продолжение с предыдущей страницы)

```
          })
        }}
/>
```

Запрос возврата маркера ввода

Если другому контролу нужно начать операцию редактирования данных ввода, и при этом маркер ввода находится в текущем контроле. У текущего контрола происходит событие на возврат маркера ввода.

При этом:

1. Если текущий контрол не может завершить формирование данных ввода
   К примеру дата введена не полностью, и данные ввода находятся в недостоверном состоянии
   1. Текущий контрол отменяет запрос
2. Иначе
   1. происходит формирование данных для ввода
   2. Сформированные данные отправляются на сервер

Пример:

```
this.controller.onCompleteInputMarker = (inputMarker)=>{
  const cds =  this.controller.localContext["cellsDataSet"] as NDataSet.NDataSet
  const curChNumber = cds.getCurChNumber()
  this.controller.coreController.postInputData(inputMarker,{
      data: cds.buildChanges()
  }).finally(()=>{
      cds.clear(curChNumber)
  })
}
```

3.5 Rpc

Позволяет вызывать запрос в ABI.

Запросы попадают в `ru.bitec.app.gs3.test.extctrl.Gs3_ExtControlExampleCbi#onRequest`

Синхронные вызовы

Передает запрос в CBI гарантируя синхронизацию данных ввода перед началом выполнения запроса.

Перед началом выполнения синхронной операции, открытый маркер ввода закрывается.

Асинхронные вызовы

Передает запрос в CBI, без синхронизации данных ввода

3.6 Согласованность изменений

В случаи если необходимо обеспечить согласованность изменений на сервере. То есть при ошибки одной асинхронной операции связанные асинхронные операции тоже должны быть отменены. Применяется механизм оптимистической блокировки. В каждый запрос на сервер может быть добавлен атрибут `consignmentVersion` если он задан запрос выполнится только в том случаи если на сервере этот атрибут не был изменен. Данный атрибут изменяется при любой ошибки запроса. Таким образом исключается возможность несогласованных изменений.

4 Декларативный подход

Для программирования компонентов в декларативном подходе используется библиотека react.

4.1 Модель данных

Для облегчения взаимодействие модель данных может быть декорированная, основой декорации служит класс `gs-web-dtk/gsreact/model.ts/NDataModelAbst`.

Декоратор узла модели данных позволяющий подписываться на изменения в декларации отображения компонента. Модель данных может отображать данные через привязанный датасет.

Хуки

Для того чтобы подписаться на перерисовку интерфейса при изменении узла модели существуют хуки:

- `useTree`
  Хук для привязки к изменениям атрибутов и дочерних элементам.
- `useTreeAndChContext`
  Хук для привязки к изменениям модели и dataset.

Привязка dataset

Привязка dataset позволяет модифицировать данные модели через механизм dataset.

Для привязки модели к dataset необходимо переопределить в модели узла функцию `getDataSetName`

Примеры

Пример модели:

```
export class Cols extends rm.NDataModelAbst{
    getColsBySpan(startIndex:number,colspan:number): Col[]{
        var cols = []
        var cs = colspan
        if (cs==undefined){
            cs = 1
        }
        for(let i = 0;i<cs;i++){
            cols.push(this.getChildAs(Col,startIndex+i))
        }
        return cols
    }
}

export class Col extends rm.NDataModelAbst{
    getDataSetName(): string {
        return "cols"
    }
    getWidth():number{
       return this.getAttrValue("width")
    }
    setWidth(value:number){
        this.setAttrValue("width",value)
     }
}
```

Пример отображения по модели данных:

```
  /**
   * Декларация псевдо строки где будут расположены ячейки с линейками для столбцов.
   */
  export const RowTag = React.memo((props:{cols:Cols}) =>{
    const cols = props.cols.useTree()
    //console.log(`RowTag, cols:${cols}`)
    var rullersContent = []
    rullersContent.push(<td key="dummy" className={styleStore.getStyle("td")}></td>)
    for(let curColIndex=0;curColIndex<cols.getChildrenCount();curColIndex++){
      const col = cols.getChildAs(Col,curColIndex)
      rullersContent.push(
        <td key={curColIndex} className={styleStore.getStyle("td")}>
          <RulerTag col={col}/>
        </td>)
    }
    return <tr key="col-rulers">{rullersContent}</tr>
  })
```

4.2 Подключение компонента к Dom

Для подключения компонента необходимо переопределить функции компонента `attach` и `detach`. Пример:

```
export class SomeControl extends Control.ControlAbst{
    root : ReactDOM.Root

    attach(parent: HTMLElement) {
        this.root = ReactDOM.createRoot(parent);
        this.root.render(tagBuilder(this.controller));
    }

    detach() {
        this.root.unmount()
    }

}
```

5 Императивный подход

> **Внимание:** Раздел находится в разработке

5.1 Модель данных

При императивном подходе для обработки изменений модели необходимо переопределить функцию контроллера `onModelChanges`. Функция принимает на вход корневой узел модели.

У узла модели доступны следующие методы.

- `hasChanges`
  Узел или его потомки имеют изменения
- `isAttsChanged`
  Атрибуты узла изменились
- `isChildrenChanged`
  Дочернии элементы узла изменились
- `isNew`
  Узел полностью перестроин
- `getChangedChilren` \ Возвращает перечень дочерних узлов по которым есть изменения

Методы модели позволяют рекурсивно обойти изменения и обработать их, или полностью перестроить модель в случаи необходимости.

6 Рабочее пространство frontend

Клиентский код пишется в редакторе `vscode` в рабочем пространстве. Рабочее пространство позволяет одновременно работать с несколькими nodejs проектами.

Рабочее пространство веб расширений располагается по адресу: `btk/src/web/ext/packages.code-workspace`

6.1 Nodejs проект

Франтент код компилируется с помощью технологии nodejs.

Шаблон проекта:

- `node_modules`
  Модули необходимые для разработки и компиляции проекта. Данные модули закачиваются при инициализации проекта с помощью менеджера пакетов `npm`
- `src`
  Исходный код
- `package.json`
  Содержит перечень зависимостей необходимых для компиляции и выполнения проекта, а так же необходимые скрипты для разработки.
- `rollup.mjs`
  Содержит правила сборки проекта в rollup.
- `tsconfig.json`
  Содержит настройки языка typescript для проекта.

6.2 gs-web-dtk

Nodejs проект предоставляющий общий код в веб расширения располагается по пути `btk/src/web/ext/gs-web-dtk`

6.3 Веб расширение

nodejs проект реализующий веб расширение, располагается по пути `btk/src/web/ext/packages/[web_extension_name]`

Для обеспечения удобной отладки и модификации кода, модуль `gs-web-dtk` подключается по исходному коду. Это приводит к следующим особенностям в конфигурации веб расширения:

- корнем сборки является каталог рабочего пространства
- необходимо жестко определять корневой репозиторий `node_modules`
- так как `@rollup/plugin-node-resolve` смотрит так же репозитории модуля `gs-web-dtk`, необходимо для общих модулей указывать правило разрешения конфликта.

Примечание: В случаи если `gs-web-dtk` не будет иметь собственного стандартного репозитория модулей, подсказчик vscode не сможет корректно осуществлять проверку синтаксиса.

Пример конфигурации `plugin-node-resolve`:

```
    nodeResolve(
      {browser: true,
       dedupe: ['react', 'react-dom'],
       rootDir: process.cwd()
      }
    )
```

Пример конфигурации `tsconfig.json` для веб расширения:

```
{
    "compilerOptions": {
      "rootDir": "../../",
      "sourceMap": true,
      "module": "ES6",
      "jsx": "react",
      "moduleResolution": "node",
      "esModuleInterop": true,
      "baseUrl": "../../",
      "paths": {
        "gs-web-dtk": ["gs-web-dtk/src/index.ts"],
        "*" : ["packages/dashboard/node_modules/*"]
      },
      "lib": ["es2018","DOM"]
    }
}
```

6.4 Подготовка к работе

1. установите nodejs
   https://nodejs.org/en/
2. Откройте `workspace` vscode
3. Включите закладку npn scripts
   Перечень доступных закладок находится под 3 точками в заголовке `Explorer`
4. Подготовьте к работе `gw-web-dtk`
   Выполните `npm script: gs-web-dtk > package.json > npm_install`
5. Подготовьте пакет к работе
   Выполните `npm script: npm_install`

6.5 Редактирование исходного кода

1. Отредактируйте файлы
2. Нажмите сохранить
3. Выполните npm script: `build` \

   Внимание: Скрипт должен быть выполнент для текущего веб расширения

4. Дождитесь компиляция бандла

Примечание: Чтобы исключить необходимость пересборки проекта в `Intellij idea` при компиляции в vscode в `rollup` проект добавлен следующий хук:

```
    copy({
      targets: [
        { src: '../../../../main/resources/ru/bitec/app/btk/dashboard/web', dest: '../../
↪../../../target/scala-2.12/classes/ru/bitec/app/btk/dashboard' },
      ],
      hook:'writeBundle'
    })
```

Данный хук копирует скомпилированный код в собранные файлы проекта.

6.6 Отладка проекта

Отладка в chrome

1. Обновите страницу браузера чтобы исключить устаревшие копии исходного кода
2. Откройте инструмент разработчика и перейдите на вкладку «Sources»
3. Откройте требуемый файл
   Файлы находится в попке {host}/webserver/sid/{guid}/siv/web
4. Установите точку остановки

Отладка в vscode

1. Запустите браузер в режиме отладки
   Для этого выполните команду `chrome.exe --remote-debugging-port=9229`
2. В разделе отладке запустите цель `Attache chrome`
3. Откройте требуемый файл
4. Установите точку остановки

Настройка отладки в vscode

Для настройки отладки в новом проекте создайте файл `.vscode/launch.json` с следующим содержимым:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Attach Chrome against localhost",
            "type": "chrome",
            "request": "attach",
            "port": 9229,
            "sourceMapPathOverrides": {
                "../../../../../../../../web/ext/*": "${workspaceFolder}/../../*"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            }
        }
    ]
}
```

Совет: Для диагностики отладки в `debug console` воспользуйтесь командой:

- `.scripts`
  Данная команда помогает корректно настроить `sourceMapPathOverrides` для работы с точками прерывания.
