Документация разработчика

Содержание

[1. Структура пакетов 3](#_Toc383612112)

[2. Правила наименований 3](#_Toc383612113)

[3. Описание разработки типового модуля 4](#_Toc383612114)

[3.1 Клиентская часть 4](#_Toc383612115)

[3.1.1 Графический интерфейс 4](#_Toc383612116)

[3.1.1.1 Точка входа 4](#_Toc383612117)

[3.1.1.2 Локализация модуля 5](#_Toc383612118)

[3.1.1.3 Работа с изображениями 6](#_Toc383612119)

[3.1.1.4 Работа с конфигурацией и разграничениями 7](#_Toc383612120)

[3.1.1.5 Загрузка данных с сервера (Loaders) 8](#_Toc383612121)

[3.1.1.6 Компонент Grid 9](#_Toc383612122)

[3.1.2 Домены модуля 10](#_Toc383612123)

[3.1.3 Интерфейсы удаленных сервисов 11](#_Toc383612124)

[3.2 Серверная часть 11](#_Toc383612125)

[3.2.1 Слой сервисов 12](#_Toc383612126)

[3.2.1.1 GWT сервисы. 12](#_Toc383612127)

[3.2.1.2 Бизнесс сервисы 12](#_Toc383612128)

[3.2.2 Слой обращения к данным (DAO) 13](#_Toc383612129)

[3.2.2.1 JDBC 13](#_Toc383612130)

[3.2.2.2 Модификация данных 17](#_Toc383612131)

[3.2.3 Локализация серверной части 18](#_Toc383612132)

[3.2.4 Контроллер 18](#_Toc383612133)

[3.2.5 Конфигурация модуля 18](#_Toc383612134)

[3.2.5.1 Предварительная настройка конфигурации 18](#_Toc383612135)

[3.2.5.2 Права доступа 19](#_Toc383612136)

[4. Установка модуля 20](#_Toc383612137)

[4.1 Назначение адреса модулю 20](#_Toc383612138)

[4.2 Настройка прав доступа 20](#_Toc383612139)

[4.3 Настройка в точке доступа GWT 20](#_Toc383612140)

[4.4 Добавление ссылки в главном меню 20](#_Toc383612141)

# Структура пакетов

|  |  |
| --- | --- |
| **Пакет** | **Описание** |
| **pst**.arm | Главный пакет |
| **pst.arm.client** | **Классы клиентской стороны** |
| .common | Общие классы для всего приложения |
| .common.ui | Общие элементы GUI |
| .common.lang | Общие языковые константы, сообщения |
| .common.domain | Общие домены |
| .common.exception | Общие классы исключений |
| .common.images | Общие иконки, изображения |
| .common.service.remote | Интерфейсывызоваудаленныхметодовсервера (GWTService.java, GWTServiceAsync.java) |
| .modules | Пакет размещения пакетов модулей |
| .modules.<modulename> | Пакет модуля |
| .modules.<modulename>.portlet | Портлеты модуля |
| .modules.<modulename>.lang | Языковеконстанты, сообщения, локали  (Constants.java, Constants\_en.properties, Constants\_ru\_RU.properties, Messages.java, Messages\*.properties) |
| .modules.<modulename>.service.remote | Интерфейсывызоваудаленныхметодовсервера (GWTService.java, GWTServiceAsync.java) |
| .modules.<modulename>.domain | Классы доменов модуля |
| .modules.<modulename>.exception | Классы исключений модуля |
| .modules.<modulename>.images | Интерфейс для вставки изображения |
| .modules.<modulename>.<submodule> | Пакет подмодуля, может содержать такие же пакеты как и главный модуль |
| utils | Вспомогательные утилиты (работа с датами) |
| **pst.arm.server** | **Классы серверной стороны** |
| .utils | Вспомогательные утилиты (транслитерация, запись числа строкой и т.п.) |
| .common | Общие классы для всего приложения |
| .common.domain | Общие домены |
| .common.dao | Интерфейсы слоя доступа к данным |
| .common.dao.jdbc | Реализация доступа к данным через JDBC |
| .common.web | Классы контроллеров ядра приложения |
| .modules | Пакет размещения пакетов модулей |
| .modules.<modulename> | Пакет модуля |
| .modules.<modulename>.dao | Интерфейсы слоя доступа к данным |
| .modules.<modulename>.dao.jdbc | Реализация доступа к данным через JDBC |
| .modules.<modulename>.dao.jdbc.mapper | Классы мапперов |
| .modules.<modulename>.service | Интерфейсы сервисов |
| .modules.<modulename>.service.impl | Реализация интерфейса |
| .modules.<modulename>.service.gwt | Реализация серверного обработчика вызова gwt сервиса клиентом |
| .modules.<modulename>.web | Классы контроллеров модуля |

# Правила наименований

|  |  |
| --- | --- |
| Что | **Как** |
| Клиентская сторона | |
| Точка входа, главная станица модуля | <Module>Page |
| Класс доступа к константам/сообщениям | <Module>Constants/Messages.java |
| Файл свойств, содержащий значения констант на английском | <Module>Constants/Messages\_en.properties |
| Файл свойств, содержащий значения констант на русском | <Module>Constants/Messages\_ru\_RU.properties |
| Интерфейс GWT сервиса | GWT**<**ServiceName**>**Service |
| Асинхронный интерфейс GWT сервиса | GWT<ServiceName>ServiceAsync |
| Серверная сторона | |
| Реализация GWT сервиса | GWT**<**ServiceName**>**ServiceImpl |
| Интерфейс бизнесс-сервиса | **<**ServiceName**>**Service |
| Реализация бизнесс-сервиса | **<**ServiceName**>**ServiceImpl |
| Интерфейс доступа к данным | **<**DAOName**>**DAO |
| JDBC-реализация доступа к данным | **<**DAOName**>**DAOJdbcImpl |
|  |  |
|  |  |

# Описание разработки типового модуля

Типовой модуль состоит у нас из клиентской и серверной части. Под клиентской частью подразумевается та часть которая работает на компьютере пользователя в браузере, а под серверной все скрипты выполняющиеся на удаленном сервере.

Рассмотрим более подробно каждую из частей.

## Клиентская часть

Клиентская часть в общем случае состоит из классов графического интерфейса, классов основных сущностей (доменов) с которыми работает модуль, и естественно интерфейсов сервисов для удаленных вызовов процедур на серверной стороне.

Все файлы, относящиеся к модулю должны находится в пакете

Package pst**.**arm**.**client**.**modules**.<**module**>;**

### Графический интерфейс

#### Точка входа

Точкой входа в модуль является класс <Module>Page.

package pst**.**arm**.**client**.**modules**.<**module**>;**

**import** com**.**extjs**.**gxt**.**ui**.**client**.**widget**.**LayoutContainer**;**

**import** pst**.**arm**.**client**.**modules**.**BasePageNew**;**

publicclass**<**Module**>**Page **extends** BasePageNew **{**

@Override

protected LayoutContainer getContentPanel**(){**

**returnnew<**Module**>**Panel**();**

**}**

**}**

Для отображения модуля на экране мы должны переопределить метод getContentPanel**().** В результате возвращаемый LayoutContainer отобразится в центре экрана под главным меню приложения.

Далее по своим нуждам создаем начинку нашего модуля различными панелями, гридами и другими элементамиGUI.

#### Локализация модуля

Детальное описание процесса локализации GWT приложения находится по адресу:<http://projectpossibility.org/projects/word_prediction/gwt-linux-1.4.60/doc/html/com.google.gwt.doc.DeveloperGuide.Internationalization.html>

**Вынесение значений в отдельные файлы**

Все строковые локализуемые константы и сообщения должны выносится в подпакетlang.

Онвобщемслучаесодержитфайлы:

|  |  |
| --- | --- |
| <Module>Constants.java | интерфейс, содержат методы для обращения из кода программы |
| <Module>Constants\_en.properties | значения констант на английском |
| <Module>Constants\_ru\_RU.properties | значения констант на русском |
| <Module>Messages.java | интерфейс, содержат методы для обращения из кода программы |
| <Module>Messages\_en.properties | значения сообщений на английском |
| <Module>Messages\_ru\_RU.properties | значения сообщений на русском |

**Форматирование интерфейсов и свойств**

* Наименование:

Методы наименуются в соответствии с верблюжьей нотацией (первая буква строчная)

Значения наименуются через точку строчными буквами (каждому горбу верблюда предшествует точка)

Наименования методов и значений должны совпадать! Однако, наименование метода не должно содержать имени модуля!

* Построение имен:

methods - <имя\_константы>()

properties - <имя\_модуля>.<имя\_константы\_через\_точку\_маленькими\_буквами>

* Сокращения

|  |  |
| --- | --- |
| **Полностью** | **Сокращение** |
| Button | btn |
| CheckBox | cb |
| ComboBox | cmb |
| Column | col |
| Edit | edit |
| FieldSet | fst |
| Form | frm |
| Grid | grid |
| Image | img |
| Label | lbl |
| List | lst |
| ListBox | lb |
| MessageBox | mb |
| NumberField | nf |
| PageControl | pg |
| Panel | pnl |
| PopupMenu | pmn |
| PopupMenuItem | pmni |
| RadioButton | rb |
| RadioGroup | rg |
| StatusBar | stb |
| Table | tbl |
| TabPanel | tpnl |
| TextArea | ta |
| ToggleButton | tbtn |
| ToolBar | tlb |
| ToolTip | tlt |
| Tree | tree |
| Window | win |
| Fund | fund |
| Inventory | inv |
| Unit | unit |
| Document | doc |
| Архивный шифр | num |
| Литера | l |
| Номер | num |
| Том Часть | vol |

* Примеры:

Constants.java

package pst**.**arm**.**client**.**pages**.<**module**>.**lang**;**

publicinterface Constants **extends** com**.**google**.**gwt**.**i18n**.**client**.**Constants **{**

@Key**(**"<module.heading"**)**

String heading**();**

**}**

Constants.properties

safetyfund.heading=Заголовок

**Обращение к константам и сообщениям из кода**

* Необходимо вначале сгенерировать объект используя метод GWT**.**create.
* Далее обратиться к нужному методу объекта для получения значения (constants**.**heading**()**).

package pst**.**arm**.**client**.**pages**.<**module**>;**

**import** com**.**extjs**.**gxt**.**ui**.**client**.**widget**.**LayoutContainer**;**

publicclass**<**Module**>**Panel **extends** LayoutContainer **{**

private**<**Module**>**Constants constants **=(<**Module**>**Constants**)**  
 GWT**.**create**(<**Module**>**Constants**.**class**);**

private**<**Module**>**Messages messages **=(<**Module**>**Messages**)**  
 GWT**.**create**(<**Module**>**Messages**.**class**);**

private ContentPanel panel**;**

public**<**Module**>**Panel**(){**

panel **=new** ContentPanel**(new** FitLayout**());**

//установка заголовка панели из констант

panel**.**setHeading**(**constants**.**heading**());**

//формирование сообщения о количестве сохраняемых записей

recordsToSave **=**100**;**

totalRecords **=**600**;**

MessageBox box **=** MessageBox**.**wait**(**constants**.<**module**>**BoxTitle**(),** messages**.<**module**>**MsgText**(**recordsToSave**,** totalRecords**),**""**);**

**}**

**}**

#### Работа с изображениями

* Создаем интерфейс <Module>Images в пакете  
  pst**.**arm**.**client**.**modules**.<**module>.images
* Записываем файл изображения в ресурсы по пути  
  /src/resources/pst/arm/client/modules/<module>/images/<файл изображения>

Расположение файла в ресурсах должно совпадать с расположением файла интерфейса.

**Пример**

Файл /src/java/pst/arm/client/modules/<module>/images/<Module>Images.java

package pst**.**arm**.**client**.**pages**.<**module**>.**images**;**

**import** com**.**google**.**gwt**.**user**.**client**.**ui**.**AbstractImagePrototype**;**

**import** com**.**google**.**gwt**.**user**.**client**.**ui**.**ImageBundle**;**

publicinterface**<**Module**>**Images **extends** ImageBundle **{**

AbstractImagePrototype icon**();**

**}**

Листингдиректории/src/resources/pst/arm/client/modules/<module>/images

/src/resources/pst/arm/client/modules/<module>/images

--..

--.

--icon.png

**Обращение к изображениям в коде**

package pst**.**arm**.**client**.**pages**.<**module**>;**

**import** com**.**extjs**.**gxt**.**ui**.**client**.**widget**.**LayoutContainer**;**

publicclass**<**Module**>**Panel **extends** LayoutContainer **{**

private**<**Module**>**Images images **=(<**Module**>**Images**)** GWT**.**create**(<**Module**>**Images**.**class**);**

private ContentPanel panel**;**

public**<**Module**>**Panel**(){**

panel **=new** ContentPanel**(new** FitLayout**());**

//установка заголовка панели из констант

panel**.**setIcon**(**images**.**icon**());**

**}**

#### Работа с конфигурацией и разграничениями

Конфигурация и разграничения предназначены для настройки интерфейса АРМ'а.

Как устанавливаются значения конфигурации описано в разделе [настройка конфигурации](#_Настройка_конфигурации).

Обращение к конфигурации из кода осуществляется с помощью класса ConfigurationManager расположенного в пакете pst.arm.client.common.

// обращение к глобальным настройкам (getGlobalProperty - возвращает String):

String userName **=** Integer**.**valueOf**(**ConfigurationManager**.**getGlobalProperty**(**"currentUser"**));**

// пример использования настроек для модуля.

// getProperty - возвращает String

// т.к. множество настроек представляют из себя булевую переменую,

// то был добавлен метод getPropertyAsBoolean возвращающий Boolean

**if** **(**ConfigurationManager**.**getPropertyAsBoolean**(**"searchByAdditionalParam.enabled"**))** **{**

CheckBox cbParam1 **=** **new** CheckBox**();**

cbParam1**.**setEnabled**(**ConfigurationManager**.**getPropertyAsBoolean**(**"param1.enabled"**));**

cbParam1**.**setValue**(**ConfigurationManager**.**getPropertyAsBoolean**(**"param1.marked"**));**

lc**.**add**(**cbParam1**);**

CheckBox cbParam2 **=** **new** CheckBox**();**

cbParam2**.**setVisible**(**ConfigurationManager**.**getPropertyAsBoolean**(**"param2.visible"**));**

lc**.**add**(**cbParam2**);**

TextField**<**String**>** tfText **=** **new** TextField**<**String**>();**

tfText**.**setValue**(**ConfigurationManager**.**getProperty**(**"param3.defaultValue"**));**

lc**.**add**(**tfText**);**

**}**

Проверка на доступ к модулю так же осуществляется через ConfigurationManager

**if** **(**ConfigurationManager**.**isModuleAvailable**(**"adminfield"**)){**

**...**

**}**

При частом использовании какого либо свойства или группы свойств, в ConfigurationManager можно добавить соответствующий метод. Пример:

publicstatic Boolean hasFacilitiesEditor**(){**

**return** hasFacilities**(**"edit"**)**

**||** hasFacilities**(**"editAll"**)**

**||** hasFacilities**(**"approve"**);**

**}**

#### Загрузка данных с сервера (Loaders)

Преобразованием данных, получаемых с сервера, на клиенте занимаются объекты следующих классов

* RpcProxy
* BeanModelReader

Для загрузки необходимых данных в грид используется объект класса ListLoader.

Пример создания грида с удаленной загрузкой данных.

publicclass MyDialog **extends** Window **{**

public MyDialog() **{**

// Указываем тип данных получаемых с сервера списком - [Domain]

RpcProxy**<**List**<[**Domain**]>>** proxy **=new** RpcProxy**<**List**<[**Domain**]>>(){**

// Операция, загружающаяданные

@Override

**public void** load(Object loadConfig, AsyncCallback<List<Role>> callback) {

service.getRoles(callback);

}

**}**

// Осуществляетчтениеобъектов BeanModel

BeanModelReader reader **=new** BeanModelReader**();**

// Обеспечиваетзагрузкуданныхвобъект store

ListLoader**<**ListLoadResult**<**ModelData**>>** loader **=new** BaseListLoader**<**ListLoadResult**<**ModelData**>>(**proxy**,** reader**);**

// Хранилищеданныхдляобъекта grid

ListStore**<**BeanModel**>** store **=new** ListStore**<**BeanModel**>(**loader**);**

List**<**ColumnConfig**>** columns **=new** ArrayList**<**ColumnConfig**>();**

columns**.**add**(new** ColumnConfig**(**"id"**,**"Номер"100**));**// #NLS

columns**.**add**(new** ColumnConfig**(**"text"**,**"Текст"**,**300**));**// #NLS

ColumnModel cm **=new** ColumnModel**(**columns**);**

Grid**<**BeanModel**>** rolesGrid **=new** Grid**<**BeanModel**>(**store**,** cm**);**

**}**

**…{**

// Инициация загрузки данных

**loader.load();**

**}**

**}**

#### Компонент Grid

Если в разрабатываемом модуле необходимо реализовать табличное представление данных, то можно воспользоваться базовыми классами классы **BaseGridSimple.**

Данные классы реализуют возможности:

* Табличное представление данных с постраничным выводом
* Панель инструментов для таблицы

Для использования функциональности компонента в своем модуле необходимо:

1. Создать класс наследник **BaseGridSimple**
2. Переопределить функцию асинхронного получения данных
3. Переопределить функцию инициализации столбцов в таблице
4. При необходимости переопределить функцию первоначальной настройки (панели инструментов, опций загрузки и т.д.)

##### Инициализация грида

Инициализация грида включает в себя инициализацию колонок и первоначальную настройку грида.

**Инициализация колонок** заключается в заполнении массива gridColumnsConfig (List<ColumnConfig>) с использованием объектов типа ColumnConfig (входит в состав библиотеки GXT)либо типа GridColumnConfig (по сути является оберткой для ColumnConfig и содержит некоторое кол-во конструкторов для удобства) в методе createColumnsConfig()

##### Визуализация данных (GridCellRenderer)

Для отображения данных в гриде, использую определенную логику, можно использовать рендеры. Один рендер может быть использован для одной колонки.

Например, если мы хотим отобразить в ячейке таблицы значок картинки, в случае наличия изображения у объекта:

1. Необходимо создать класс, реализующий функцию render интерфейса GridCellRenderer

protectedclass HasImageRenderer **implements** GridCellRenderer**<**BeanModel**>{**

@Override

public Object render**(**BeanModel model**,** String property**,** ColumnData config**,**int rowIndex**,**int colIndex**,** ListStore store**,** Grid grid**){**

ArchiveStoreObject obj **=** model**.**getBean**();**

String div**;**

**if(**obj**.**hasImages**()){**

div **=**"<div class='has-image' title='"**+** asconstants**.**fundHasImagesYes**()+**"("**+** obj**.**getImageCount**()+**")'></div>"**;**

**}else{**

div **=**"<div style='width:20px;' title='"**+** asconstants**.**hasImagesNo**()+**"'></div>"**;**

**}**

**return** div**;**

**}**

**}**

1. Добавить рендер нужной колонке грида:

ColumnConfig config = **new** ColumnConfig();

config.setRenderer(**new** HasImageRenderer());

##### Редактирование данных RowEditor.

Для построчного редактирования данных в Grid используется плагин RowEditor. Для того чтобы столбец стал редактируемым необходимо назначить CellEditor конфигурации столбца. Остальные данные будут браться из модели и отображаться как есть. Если представление отличается от модели то нужно создать текстовое поле и задать ему PropertyEditor.

Пример:

...

RowEditor<BeanModel> columnEditor = new RowEditor<BeanModel>();

columnEditor.setClicksToEdit(ClicksToEdit.TWO);

TextField tfColNameCell = new TextField();

TextField tfColTypeCell = new TextField();

tfColTypeCell.setPropertyEditor(new PropertyEditor() {

@Override

public String getStringValue(Object value) {

return colTypeToStr((DictionaryColumn.DictionatyColumnType) value);

}

@Override

public Object convertStringValue(String value) {

return strToColType(value);

}

});

tfColTypeCell.setEnabled(false);

...

ColumnConfig config = new ColumnConfig("name", constants.colName(), 200);

...

config.setEditor(new CellEditor(tfColNameCell));

columns.add(config);

...

config.setEditor(new CellEditor(tfColTypeCell));

columns.add(config);

...

gridColumns.addPlugin(columnEditor);

...

### Домены модуля

**Доменные объекты** — это объекты в объектно-ориентированных компьютерных программах, выражающие сущности из модели предметной области, относящейся к программе, и реализующие бизнес-логику программы.

* **Размещение доменов**

Домены размещаются в пакете domain модуля.

* **Наименование**

Наименование класса домена должно явно отражать его назначение в предметной области и бизнес-логике программы.

* **Использование доменов в data-backed компонентах (DataProxies, DataReaders,. ListLoadResults, ComboBox, ColumnModel и т.д.)**

Для использования наших доменов в таких компонентах требует, чтобы домен реализовывал интерфейс ModelData.

Из трех различных способов достижения этого:

1. Расширение доменом класса BaseModel,

2. Имплементация интерфейса BeanModelTag,

3. Создание интерфейса BeanModelMarker, используя аннотация @BEAN,

нужно использовать **3-й способ,** т.к. он позволяет избежать непосредственной модификации класса домена и предоставляет самую прозрачную логику приложения.

Пример.

Домен

package pst**.**arm**.**client**.**modules**.<**module**>.**domain**;**

public class**<**Domain**>{**

**...**

**}**

Интерфейс BeanModelMarker

package pst**.**arm**.**client**.**modules**.<**module**>.**domain**;**

**import** com**.**extjs**.**gxt**.**ui**.**client**.**data**.**BeanModelMarker**;**

**import** com**.**extjs**.**gxt**.**ui**.**client**.**data**.**BeanModelMarker**.**BEAN**;**

@BEAN**(**pst**.**arm**.**client**.**modules**.<**module**>.**domain**.<**Domain**>.**class**)**

publicinterface**<**Domain**>**BeanModel **extends** BeanModelMarker **{}**

### Интерфейсы удаленных сервисов

Взаимодействие клиента с сервером организуется через сервисы.

Для осуществления данного взаимодействия на клиенте необходимо следующее:

1. Создать интерфейс GWT<ServiceName>Service

package pst**.**arm**.**client**.<**modules**>.<**module**>.**service**.**remote**;**

**import** com**.**google**.**gwt**.**user**.**client**.**rpc**.**RemoteService**;**

**import** com**.**google**.**gwt**.**user**.**client**.**rpc**.**RemoteServiceRelativePath**;**

@RemoteServiceRelativePath**(**"service/<ServiceName>Service"**)**

publicinterface GWT**<S**erviceName**>**Service **extends** RemoteService **{**

/\* здесь размещаются объявления удаленных методов \*/

public String getString**(**Object param**);**

**}**

1. Создать асинхронный интерфейс GWT<ServiceName>ServiceAsync

package pst**.**arm**.**client**.<**modules**>.<**module**>.**service**.**remote**;**

**import** com**.**google**.**gwt**.**user**.**client**.**rpc**.**AsyncCallback**;**

publicinterface GWT**<**ServiceName**>**ServiceAsync**{**

publicvoid getString**(**Object param**,** AsyncCallback**<**String**>** asyncCallback**);**

**}**

## Серверная часть

Серверная часть модуля в базовом случае состоит из слоя сервисов и слоя доступа к данным.

Все классы серверной части модуля должны располагаться в пакете

pst.arm.server.modules.<modulename>

### Слой сервисов

Слой сервисов в свою очередь разделяется на два слоя.

#### GWT сервисы.

Реализуют интерфейс GWT сервиса объявленного на клиенте. Предназначены для приема удаленного вызова, предварительной обработки входящих параметров, вызова сервисов следующего уровня и отправки ответа клиенту.

Реализация GWT сервисов размещается в пакете <modulename>.service.gwt

Пример.

package pst**.**arm**.**server**.<**modules**>.<**module**>.**service**.**gwt**;**

**import** pst**.**arm**.**client**.<**modules**>.<**module**>.**service**.**remote**.**GWT**<**ServiceName**>**Service**;**

**import** pst**.**arm**.**server**.**web**.**controllers**.**GWTController**;**

@Service("GWT<ServiceName>Service"))

publicclass GWT**<**ServiceName**>**ServiceImpl **extends** GWTController **implements** GWT**<**ServiceName**>**Service **{**

@Autowired

private **<**ServiceName**>**Service **<**serviceName**>**Service**;**

/\* здесь размещается реализация удаленных методов \*/

@Override

public String getString**(**Object param**){**

/\* предварительная обработка параметров \*/

//вызов бизнесс-сервиса

**return<**serviceName**>**Service**.**getString**(**param**);**

**}**

**}**

Имя GWT сервиса указанное в аннотации @Service необходимо указать в файле конфигурации   
dispatcher-servlet.xml

<prop key="/pst.arm.Interactive/service/<servicename>">GWT<ServiceName>Service</prop>

#### Бизнесс сервисы

Сервисы, реализующие основную бизнесс-логику приложения.

Интерфейсы располагаются в пакете <modulename>.service

Реализация в пакете<modulename>.service.impl

Интерфейс бизнесс-сервиса

Package pst**.**arm**.**server**.<**modules**>.<**module**>.**service**;**

publicinterface**<**ServiceName**>**Service**{**

public String getString**(**Object param**)**;

**}**

Реализация бизнесс-сервиса

Package pst**.**arm**.**server**.<**modules**>.<**module**>.**service.impl**;**

**import** pst**.**arm**.**server**.<**modules**>.<**module**>.**service**.<**ServiceName**>**Service**;**

@Service

publicclass**<**ServiceName**>**ServiceImpl **implements<**ServiceName**>**Service**{**

@Autowired

**<**DAOName**>**DAO**<d**aoName**>**DAO**;**

@Override

public String getString**(**Object param**){**

//обращениекслоюDAO

**return<d**aoName**>**DAO**.**getString**(**param**);**

**}**

**}**

### Слой обращения к данным (DAO)

Интерфейсы и классы, расположенные в слое DAO, должны строго соответствовать названию слоя и выполнять функции по извлечению и модификации данных из различных источников. В частном и наиболее частом случае этим источником служит БД.

Интерфейсы размещаются в пакете <modulename>.dao.

#### JDBC

##### Размещение

Реализация доступа к БД через JDBC в пакете<modulename>.dao.jdbc.

Пример.

ИнтерфейсDAO

Package pst**.**arm**.**server**.<**modules**>.<**module**>.**dao**;**

publicinterface**<**DAOName**>**DAO**{**

publicObject getData**(**Object condition**)**;

**}**

Реализация бизнесс-сервиса

package pst**.**arm**.**server**.<**modules**>.<**module**>.**dao.jdbc**;**

**import** pst**.**arm**.**server**.<**modules**>.<**module**>.**dao**.<**DAOName**>**DAO**;**

**import** org.springframework.stereotype.Repository;

**import** org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.beans.factory.annotation.Qualifier;

**import** javax.sql.DataSource;

@Repository

public class**<**DAOName**>**ServiceDAOJdbcImpl **implements<**DAOName**>**DAO**{**

private NamedParameterJdbcTemplate jdbcTemplate;

@Autowired

@Qualifier("dsArm")

public void setDataSource(DataSource dataSource){

this.jdbcTemplate = new NamedParameterJdbcTemplate(dataSource);

}

@Override

public Object getData**(**Object condition**){**

//формирование запроса и выборка данных из БД

**return** result**;**

**}**

**}**

##### @Repository указывает что класс является бином и используется для доступа к данным.

@Autowired указывает что связь разрешается автоматически.

@Qualifier("dsArm") указывает что используется DataSource с параметром qualifier dsArm в нашем случае такой подход позволяет различать DataSource для арма и хранилища данных Pmasc (у хринилища Pmasc квалификатор dsPmasc).

Для генерации запросов используется NamedParameterJdbcTemplate, что позволяет именовать параметры.

Пример 1:

...

@Override

public List<Domain> getData**(<**Domain> domain**){**

String query = "SELECT SOME\_COLUMN1, SOME\_COLUMN1 "

+ "FROM SOME\_TABLE "

+ "WHERE DOMAIN\_ID = :domainId";

SqlParameterSource source =

**new** BeanPropertySqlParameterSource(domain);

**return** jdbcTemplate.query(query, source, new <Domain>Mapper());

**}**

...

В таком случае имена параметров должны соответствовать соглашению об именах для бинов.

Такой способ не будет работать если в качестве параметров приходит болеее одного бина. В таком случае можно поступить следующим образом.

Премер 2:

...

@Override

public List<Domain> getData**(<**Domain1> domain1, <Domain2> domain2, Long param**){**

String query = "SELECT SOME\_COLUMN1, SOME\_COLUMN1 "

+ "FROM SOME\_TABLE1, SOME\_TABLE2 "

+ "WHERE SOME\_TABLE1.ID = :id "

+ "AND SOME\_TABLE2.PARAM = :param "

+ "AND SOME\_TABLE2.NAME = :name";

HashMap params = new HashMap();

params.put("id", domain1.getId());

params.put("param", param);

params.put("name", domain2.getName());

SqlParameterSource source = new MapSqlParameterSource(params);

**return** jdbcTemplate.query(query, source, new <Domain>Mapper());

**}**

...

В таком случае можно передавать запросу параметры с разных доменов.

##### Операции с БД

Операции с БД можно разделить на следующие подвиды:

* Выборка данных
* Вставка данных
* Обновление данных

Для взаимодействия с БД необходимо сгенерировать SQLзапрос и выполнить его. В АРМ существует несколько подходов для генерации запроса и его выполнения

1. Генерация параметризированного запроса
2. Непосредственная генерация запроса
3. **Параметризированный запрос** – запрос, конкретные параметры в который подставляются на этапе выполнения.

Пример:

@Override

public Long insertUser**(**User u**){**

**...**

// Генерация SQL запроса, вместо конкретных значений выставлены знаки '?'

String sql **=**" INSERT INTO USERS ( "

**+**"USER\_LOGIN, PASSWORD, ARCHIVE\_ID, ENABLED,"

**+**"FIRST\_NAME, LAST\_NAME, PATRONYMIC, "

**+**"ADDRESS, PHONE, POSITION, LOCATION, DESCRIPTION ) VALUES ("

**+**":userName, :password, :archiveId, :enabled,"

**+**":firstName, :lastName, :patronymic,"

**+**" :address, :phone, :position, :location, :description)"**;**

SqlParameterSource source = new BeanPropertySqlParameterSource(u);

KeyHolder keyHolder = new GeneratedKeyHolder();

// Выполнение SQL запроса с подстановкой значений, заменяющих знаки '?'

jdbcTemplate**.**update**(**sql, source, keyHolder, new String[]{"USER\_ID"}**);**

**return** keyHolder.getKey().longValue();

**}**

**Подходит для** – генерации небольшого запроса с постоянным количеством параметров

**Операции** – операции вставки, обновления данных, простые операции выборки (например, получение записи по её id)

**Плюсы подхода** – автоматически экранирует спецсимволы, самостоятельно следит за их типами.

**Минусы подхода** – не удобен для сложно-составных запросов

**PS:** Для получения идентификатора вставленного значения нужно использовать KeyHolder. Параметр new String[]{"USER\_ID"} указывает на то что поле USER\_ID нужно вернуть после выполнения запроса.

1. **Непосредственная генерация SQL запроса**

В данном случае уже на этапе генерации запроса все параметры выставлены, выполняется именно строка запроса. Параметры

Пример:

@Override

public List**<**Fund**>** getFunds**(**ArchiveStoreSearchCondition condition**){**

String where **=** condition2Sql **(**condition**);** // получениеусловийпоиска

String order **=** condition2Order**(**condition**);** // получениеусловийпоиска

String offset **=** condition2Offset**(**condition**);**

String fields **=** condition2Fields**(**condition**);**

String markup **=** condition2Markup**(**condition**);**

String sql **=**"SELECT "**+fields +** ","**+ markup**

**+**"FROM ( SELECT ROWNUM AS R, X.\* FROM ( "

**+**"SELECT "**+fields**

**+**"FROM FUND F "

**+**" "**+where**

**+**" "**+order**

**+**" ) X "") "

**+**" "**+offset**

**return** jdbcTemplate**.**query**(**sql**, null, new** FundMapper**(**condition**));**

**}**

private String condition2Sql **(**ArchiveStoreSearchCondition condition**){**

String ret **=**""**;**

**if(**condition**.**hasId**()==false){**

**return** ret**;**

**}**

**switch(**condition**.**getSearchDirection**()){**

**case** eFund**:**

ret **+=**" FUND\_ID"**;**

**break;**

**case** eInventory**:**

ret **+=**" INVENTORY\_ID"**;**

**break;**

**case** eUnit**:**

ret **+=**" UNIT\_ID"**;**

**break;**

**case** eDocument**:**

ret **+=**" DOCUMENT\_ID"**;**

**break;**

**default:**

**thrownew** AssertionError**();**

**}**

ret **+=**" = "**+** condition**.**getRecId**();**

**return**ret**;**

**}**

**Подходит для** – генерации сложных запросов с переменным количеством параметров

**Операции** – сложные операции выборки данных с большим количеством параметров

**Плюсы подхода** – получение наглядного кода запроса, который можно выполнить через сторонние утилиты

**Минусы подхода** – необходимо самостоятельно заботится об экранировании, о типе данных (строковые параметры дополнять символами «'»)

Отдельного внимания в данном подходе заслуживает класс с параметрами поиска SearchCondition (либо унаследованный от этого класса). Использование этого класса обеспечивает более универсальный подход для передачи разнотипных параметров поиска.

##### Mappers

При выборки записей из БД, как правило, необходимо отобразить полученные абстрактные данные на поля объекта домена. Реализация мапперов в пакете<modulename>.dao.jdbc.mappers

Пример

privatestaticclass UserMapper **implements** ParameterizedRowMapper**<**User**>{**

@Override

public User mapRow**(**ResultSet rs**,**int rowNum**)throws** SQLException **{**

User u **=new** User**();**

u**.**setId**(**rs**.**getLong**(**"USER\_ID"**));**

u**.**setUserName**(**rs**.**getString**(**"USER\_LOGIN"**));**

u**.**setDateCreated**(**getDateFromRs**(**rs**,**"DATE\_CREATED"**));**

u**.**setEnabled**(**rs**.**getInt**(**"ENABLED"**));**

**return** u**;**

**}**

**}**

При написании маппера необходимо учитывать:

* Особенности получения данных типов Integer, Long

При получении данных этого типа необходимо учитывать то, что если в базе NULL, то функция rs.getInt(…) - rs.getLong(…) вернет число 0, вместо ожидаемого NULL. Для предотвращения путаницы рекомендовано использовать

…

Integer integer = rs.getInt("COLUMN\_NAME");

**if**(rs.wasNull()){

integer = **null**;

}

…

#### Модификация данных

Существует 2 подхода для модификации данных: с использованием и без использования транзакций. Самый простой вариант (**без транзакций**) может быть использован, если необходимо изменить/удалить одну или несколько записей, используя одно SQL-выражение.

Пример:

publicvoid deleteById**(**final Long id**){**

getJdbcTemplate**().**update**(**"DELETE FROM TABLE WHERE ID="**+**id**+**""**);**

**}**

**Транзакции** можно использовать следующим образом:

1. Реализовать set метод для установки менеджера транзакций в файле реализации слоя DAO

private TransactionTemplate txTemplate**;**

publicvoid setTransactionManager**(**PlatformTransactionManager txManager**){**

**this.**txTemplate **=new** TransactionTemplate**(**txManager**);**

**this.**txTemplate**.**setPropagationBehavior**(**DefaultTransactionDefinition**.**PROPAGATION\_REQUIRED**);**

**}**

1. В файле applicationContext-jdbc.xml задать бин менеджер транзакций и задать ссылку на него для слоя DAO

<bean id="transactionManagerArm" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSourceArm"/>

</bean>

<beanid="userDAO" class="pst.arm.server.common.dao.jdbc.UserDAOJdbcImpl">

<propertyname="dataSource"ref="dataSourceArm"/>

<propertyname="transactionManager"ref="transactionManagerArm"/>

</bean>

1. Выполнить несколько SQL-выражений в одной транзакции, в случае возникновения исключений – выполнить откат транзакции.

@Override public Card insertCard**(**final Card card**){**

**return** txTemplate**.**execute**(new** TransactionCallback**<**Card**>(){**

@Override public Card doInTransaction**(**TransactionStatus ts**){**

Card ret **=null;**

**try{**

getJdbcTemplate**().**update**(**"SQL 1"**);**

getJdbcTemplate**().**update**(**"SQL 2"**);**

getJdbcTemplate**().**update**(**"SQL 3"**);**

ret **=** getCard**();**

**}catch(**Exception e**){**

ts**.**setRollbackOnly**();**

**}**

**return** ret**;**

**}**

**});**

**}**

### Локализация серверной части

### Контроллер

Класс контроллер требуется для того, чтобы модуль можно было непосредственно вызвать через обращение к нему по url, а также для предварительной конфигурации модуля на серверной стороне.

URL-адрес нового модуля задается в аннотации @RequestMapping класса контроллера.

Package pst**.**arm**.**server**.**modules**.**module**.**web**;**

**import** pst**.**arm**.**server**.**common**.**web**.**ModuleController**;**

@Controller

@RequestMapping**(**"/secure/modulename.htm"**)**

publicclass ModuleNameController **extends** ModuleController **{**

private String moduleName **=**"modulename"**;**

@Override

protected String getModuleName**(){**

**return** moduleName**;**

**}**

@Override

protected Map**<**String**,** String**>** getModuleConfig**(){**

**returnnew** HashMap**<**String**,** String**>();**

**}**

**}**

### Конфигурация модуля

Конфигурация представляется собой набор пар значений вида ключ = значение и выполнен как объект типа Map**<**String**,**String**>**.

Предварительная настройка конфигурации осуществляется в файлах properties. Окончательная настройка конфигурации для отдельно взятого модуля осуществляется в [контролере](#_Контроллер) отвечающего за этот модуль в методе getModuleConfig**()**, отталкиваясь от прав доступа предоставленных для текущего пользователя. Полученные значения перекрывают настройки из файлов properties и являются окончательными и предназначены для использования на клиентской части.

#### Предварительная настройка конфигурации

Предустановленные значения параметров конфигурации (значения по умолчанию, состояния чекбоксов, радиобаттонов и прочее) описаны в файлах properties и расположены в директории /src/resources/configs/ и имеют следующее наименование: base.properties и <archiveId>.properties, где archiveId – код архива (0 – АК, 1 – ЦГА, 2 ЦГИА, 5 - ЦГАЛИ). Настройки из properties подтягиваются в следующем порядке

1. base.properties
2. <archiveId>.properties

Второй пункт перекрывает настройки первого.

Код архива определяется по пользователю в момент его входа в систему.

Файлы properties содержат глобальные настройки и настройки для каждого модуля и описывается следующим образом:

1. Глобальные настройки

config.global.<property>=value

1. Настройки модуля

config.modules.<Module>.<property>=value

где <property> - параметр/настройка, value – его значение.

Пример файла properties:

#Common

config.global.multiArchiveSearch**=**false

config.global.availableArchives**=**

#Modules

#[main]

config.modules.main.portlets**=**changelog

#[funds]

config.modules.funds.controls.conditionTab.visible=false

#searchMode.select = all | any | exact

config.modules.funds.controls.conditionTab.searchMode.select=all

config.modules.funds.controls.conditionTab.byRequisites.marked=true

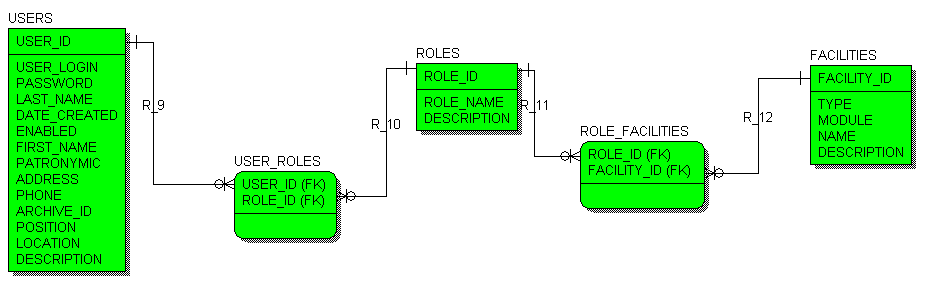
config.modules.funds.controls.conditionTab.byNsa.marked=true

config.modules.funds.controls.conditionTab.fullText.marked=true

config.modules.funds.controls.conditionTab.byEntry.visible=false

#### Права доступа

Разграничения прав (facilities / доступ к модулям и функционалу модулей) описывается в БД с помощью ролей назначаемых пользователям.



В таблицу FACILITIES заносятся записи двух типов ‘module’ и ‘module\_operation’. Пример:

**INSERTINTO** FACILITIES**(**"TYPE"**,**"MODULE"**,**"NAME"**,**"DESCRIPTION"**)VALUES('module',**'main'**,** 'access'**,**'Главнаястраница'**);**

**INSERTINTO** FACILITIES**(**"TYPE"**,**"MODULE"**,**"NAME"**,**"DESCRIPTION"**)VALUES('module\_operation',**'cardsearch'**,**'edit'**,** 'Составлениеиредактированиесвоихкарточек'**);**

Тип module говорит о доступе к модулю. Тип module\_operation – о доступе к функционалу в указанном модуле (в поле MODULE).

Права пользователя составляются из ролей. Каждая роль имеет набор facilities. Пользователь имеет суммарный набор facilities из всех назначенных ему ролей.

# Установка модуля

## Назначение адреса модулю

URL-адрес нового модуля задается в аннотации @RequestMapping класса контроллера модуля в пакете webсерверной стороны.

## Настройка прав доступа

* Добавление значения в список прав доступа в БД, как описано в [Права доступа](#_Права_доступа)
* Добавление записи для ограничения доступа к url-адресу модуля в конфигурационном файле SpringSecurity «/web/WEB-INF/ applicationContext-acegi-security.xml»,по следующему примеру:

<beanid="filterInvocationInterceptor"class="org.springframework.security.intercept.web.FilterSecurityInterceptor">

...

<propertyname="objectDefinitionSource">

<value>

CONVERT\_URL\_TO\_LOWERCASE\_BEFORE\_COMPARISON

\A/service/.\*\Z=IS\_AUTHENTICATED\_REMEMBERED

\A/secure/main.htm\Z=IS\_AUTHENTICATED\_REMEMBERED

...

**\A/secure/modulename.htm\Z=modulename**

<!-- need to put login service under diff path -->

\A/login.htm\Z=IS\_AUTHENTICATED\_ANONYMOUSLY

</value>

</property>

</bean>

## Настройка в точке доступа GWT

Класс Interactiveв пакете pst.arm.client.

Добавить строки в метод определения модуля для отображения.

private BasePageNew createLoadableWidget**(**String widget**){**

...

**if(**widget**.**equals**(**"main"**)){**

loadableWidget **= new** MainPageNew**();**

**}elseif(**widget**.**equals**(**"modulename"**)){**

loadableWidget **= new** ModuleNamePage**();**

**}**

...

**}**

## Добавление ссылки в главном меню