TIMSS业务开发编码手册

V1.0

­

广东粤电信息科技有限公司

目录

[**1.** **后端编码目录结构** 3](#_Toc514314499)

[1.1 façade模块的用途 3](#_Toc514314500)

[1.2 JAVA代码目录结构 5](#_Toc514314501)

[1.3 各个JAVA目录的用途： 6](#_Toc514314502)

[1.4 resources资源目录 7](#_Toc514314503)

[1.5 Test测试目录 7](#_Toc514314504)

[**2.** **前端编码目录结构** 8](#_Toc514314505)

[2.1 前端目录 8](#_Toc514314506)

[2.2 js目录 8](#_Toc514314507)

[2.3 page目录 9](#_Toc514314508)

[2.4 站点自定义 10](#_Toc514314509)

[2.5 编码命名 10](#_Toc514314510)

[**3.** **后端框架及编码** 11](#_Toc514314511)

[3.1 类命名方式 11](#_Toc514314512)

[3.2 后端框架 11](#_Toc514314513)

[3.3 Bean 12](#_Toc514314514)

[3.4 Dao 13](#_Toc514314515)

[3.5 Service 14](#_Toc514314516)

[3.6 站点自定义实现类的实例化 15](#_Toc514314517)

[3.7 Controller 16](#_Toc514314518)

[**4.** **前端框架及编码** 17](#_Toc514314519)

[4.1 概述 17](#_Toc514314520)

[4.2 列表页 18](#_Toc514314521)

[4.3 详情页 21](#_Toc514314522)

[5. 通过配置渲染前端框架 23](#_Toc514314523)

[5.1 列表页 23](#_Toc514314524)

[5.2 详情页 24](#_Toc514314525)

1. **后端编码目录结构**

## façade模块的用途

业务开发将接口、实体类、视图vo放到facade中，便于供其他模块调用，且避免集成时循环依赖。

集成循环依赖是指：比如自己模块内定义了一个接口A，接口内定义了一个去调其他模块的接口B的属性，那么在初始化A时，依赖B已经初始化完成，如果这时候B的初始化，也依赖一个接口A的属性才能完成，这就是循环依赖了，大家都等对方给自己初始化。解决这个问题的措施就是把这些接口都放进一个模块facade内去定义，这样他们的实现类初始化时，依赖的接口对象已经存在了。并且由于业务模块都会依赖facade，因此facade模块要最先加载。

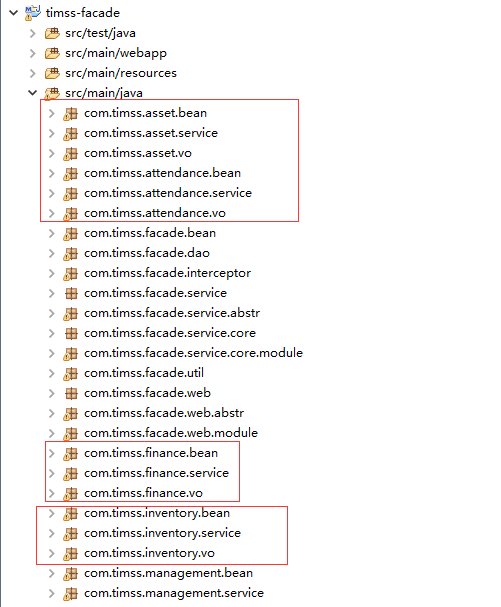
facade模块如下图1.1-1所示：

图1.1-1

service在业务模块内部中用来存放接口。

## JAVA代码目录结构

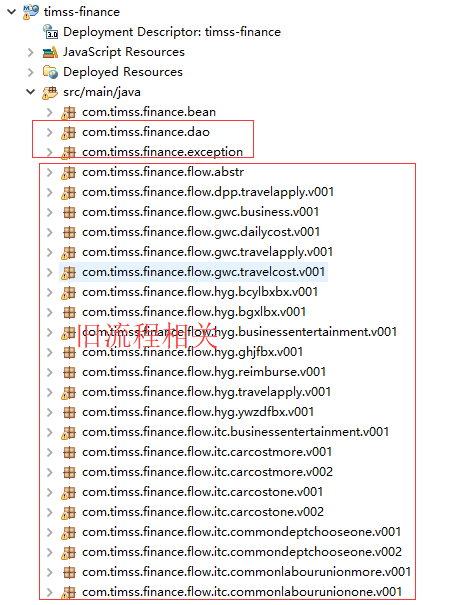


图1.2-1

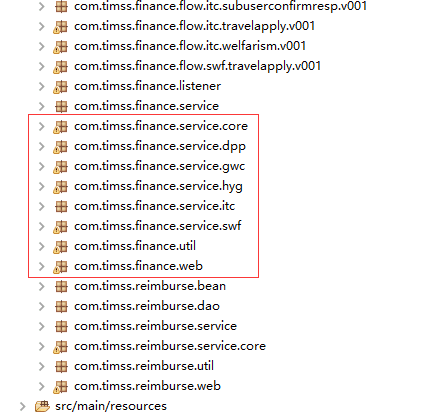


图1.2-2

如下图1.2-1和图1.2-2所示，标准规定业务模块内部只需要dao，exception、service.XXX、util、web这些目录，但是如果有其他特殊需求，例如listener、schedule也可以添加。

另外，若上图中出现的bean、service不需要公开给其他模块调用，且没有依赖问题时，可以放自己模块里而不是facade里。

## 各个JAVA目录的用途：

一般必须的目录有：

bean实体类

dao数据库操作接口

service业务接口

service.core通用业务接口的实现类

service.dpp/gwc等站点，用于站点自定义的接口的实现类

web放controller的用于视图层

其他常见的还有：

exception错误类

util模块自己用的工具类

vo已经不建议使用，现在都是使用标准一套接口，用一个业务bean配置就行。这个涉及现在框架对bean的定义，bean做为一个完整的数据载体，包括了一个业务实例内所有的数据，用于从前端传递给后端，后端处理后再查到最新的数据返回给前端展示。所以为了统一接口和操作，我们抛弃了vo，使用大而全的bean，简化接口和对象管理。所以在重构时需要把vo合并到业务的bean中（一个功能只有一个bean）。

## resources资源目录

一般存放dao接口对应的实现的sql，还有自己模块定义的配置项以及exception对应的错误信息处理。如下图1.4-1所示：

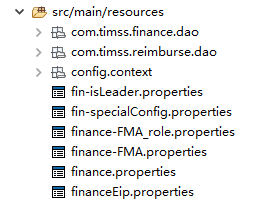


图1.4-1

## Test测试目录

对应目录代码的单元测试，一般建议要写，尽量覆盖所有逻辑。如下图1.5-1所示：

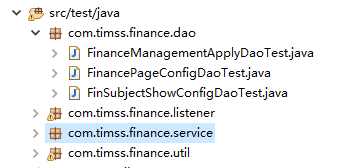


图1.5-1

1. **前端编码目录结构**

## 前端目录

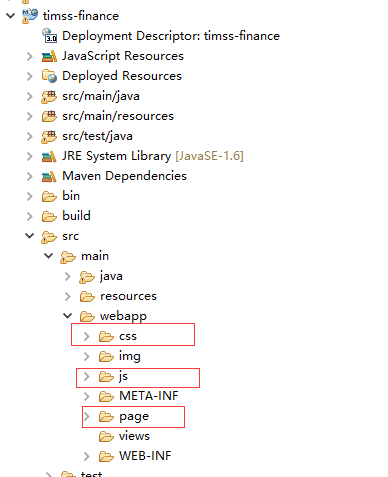
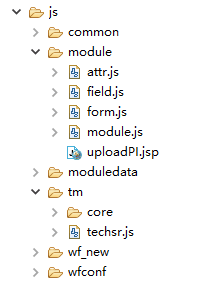


图2.1-1

开发人员一般只用到css，js，page三个目录，其中css存放css文件，js存放js文件，page存放jsp文件。

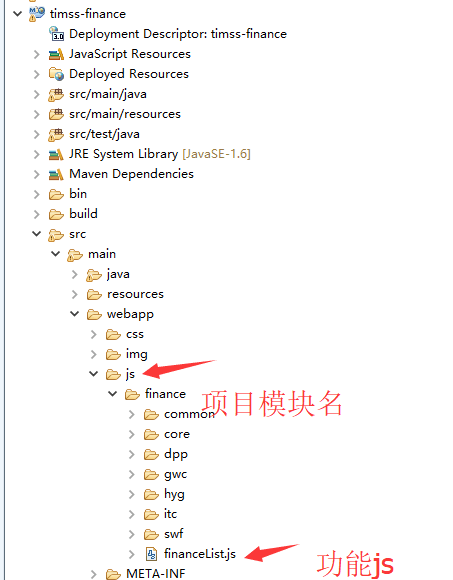
## js目录

js目录下的命名规范为:js-模块编码-功能.js或者js-通用模块编码-功能.js  
例如：js-tm-techsr.js或者js-module-module.js。如下图2.2-1所示：



下图2.2-1

其中tm为项目模块编码，对应finance。techsr为功能名。如下图2.2-2所示：



下图2.2-2

## page目录

page目录下的命名规范为：page-模块编码-站点-功能编码-jsp

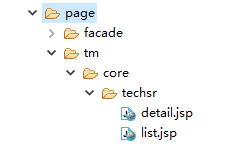
例如：page-tm-core-techsr-list.jsp，如下图2.3-1所示  


图2.3-1

其中tm为模块编码，core为通用站点，techsr为功能编码。

## 站点自定义

我们的系统有区分多站点，例如A电厂和B电厂，它们某个功能基本是一样的，但是有些字段不同，或者操作有不同，兼容这种情况，我们称之为站点自定义。

我们的框架已经能兼容前端jsp和后端service的站点自定义了，就是根据登录人属于哪个站点，自动调用对应站点的jsp或者service，如果不存在自定义的文件，则调用core的。

所以对于前端的站点个性化，我们在jsp中处理这种个性化的需求,而不是在js中，js我们定义为多站点共用的前端代码，所以目录结构中js不区分站点，jsp区分。当然，如果某个站点的个性化需求需要写很长的前端代码，我们把他放到js中，这个是不禁止的。

## 编码命名

公司为每个模块定义了编码，模块编码由项目经理决定，站点编码也是固定了的，功能编码开发人员决定。功能编码的命名要求用驼峰，非拼音或拼音缩写，并且要求站点内唯一。

1. **后端框架及编码**

## 类命名方式

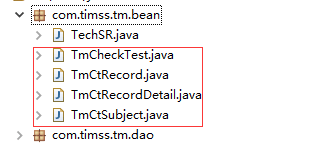


图3.1-1

类命名是首字母大写，对象是驼峰，一般要求简短但能区分，图3.1-1红框是反例，我们能从包路径知道是tm项目，类前缀就不加tm了，其他类的命名

也是如此。

## 后端框架

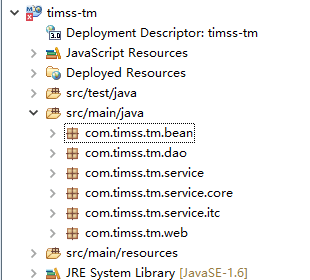


图3.1-2

公司业务总体上分为两类，一类为有审批流程的业务，一类为只有数据的增删查改，没有流程的业务

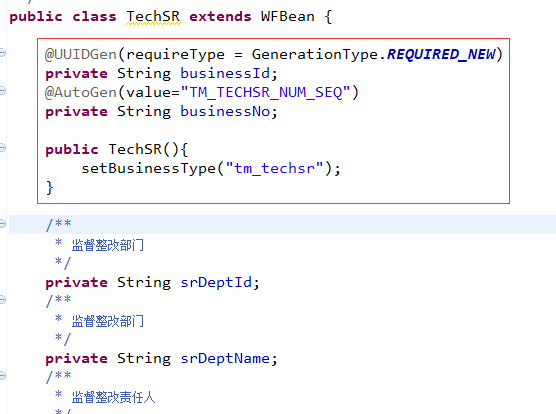
后端框架定义了一套基类，包括bean、dao、service（service的实现）、controller，以及基类通用的操作，包括无流程的增删查改，有流程的暂存、提交、审批、退回、作废等。

业务类继承对应的基类，如果通用操作无法满足自己的需求，那就重写对应的接口方法，无特殊需求的话，后台基本都是定义一个类去继承基类就搞定了。

## Bean

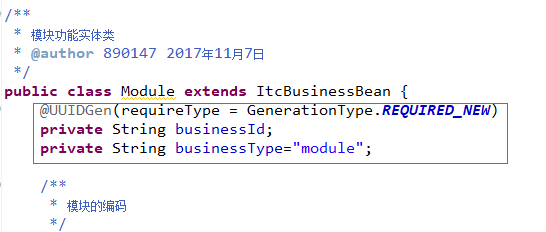
无流程就继承ItcBusinessBean，有流程继承WFBean，这两个基类都封装了通用的属性，业务bean添加自己的属性，然后要重写以下属性: businessId, businessNo, businessType。

### 有工作流的Bean



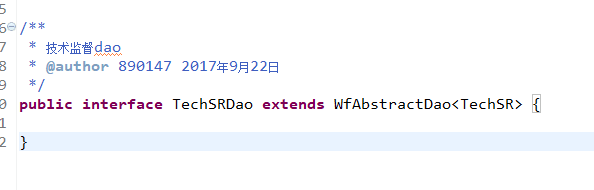
* 加注解的businessId是定义了一种自动生成id的方法加注解的；
* businessNo是定义了一种自动生成业务流水号（申请单号）的方法；
* 构造函数是写死了模块功能编码，将一个业务Bean绑定到对应的业务上

### 无工作流的Bean

同3.3.1，其中直接给businessType赋值和前面的构造函数中赋值，效果是一样的

## Dao

Dao层有流程的继承WfAbstractDao，无流程的继承AbstractDao，如果有自己的dao接口就在这里添加，如果基类的够用就不用添加了.



然后要添加Dao接口的sql，如图3.4-1及3.4-2所示：

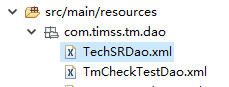


图3.4-1

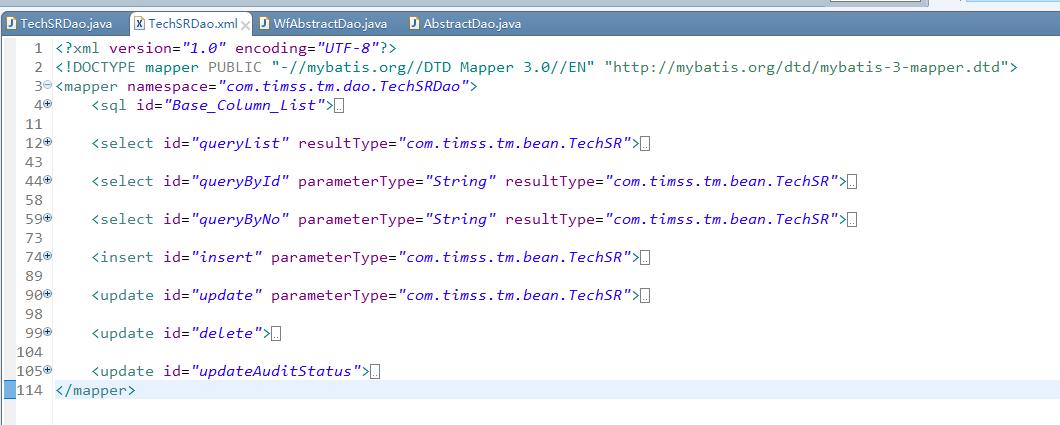
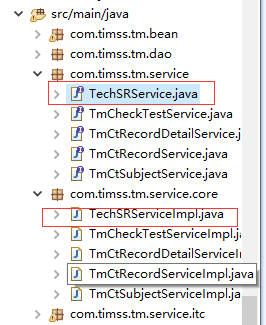


图3.4-2

一般有流程的Dao要写这几个sql，可以参考上图的写法。

## Service



有流程的继承WfAbstractService，无流程的继承AbstractService，如果有需要添加的service接口就在这里添加，如果基类够用就不用添加了

service接口的实现类如下图3.5-1所示：

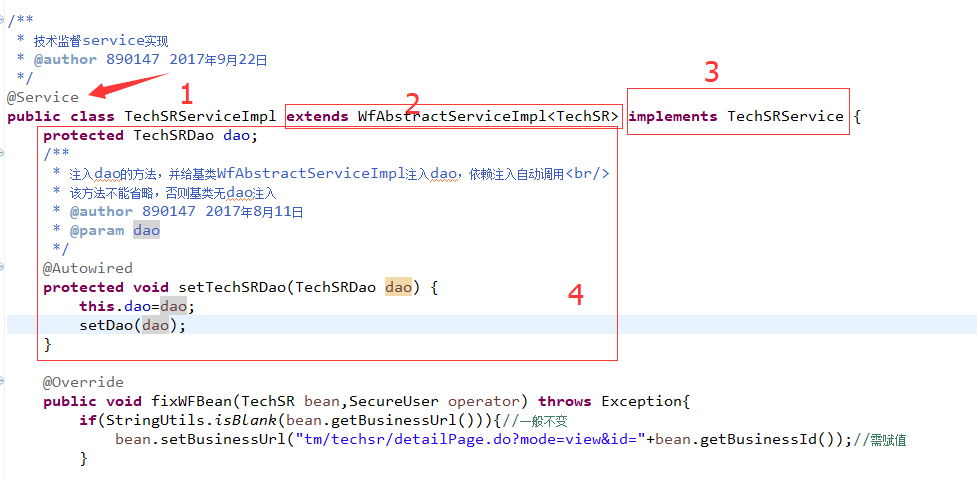


图3.5-1

1. 要加注解@service否则不能被实例化，类命名为接口类+Impl
2. 继承自WfAbstractService的默认实现类WfAbstractServiceImpl
3. 实现业务接口
4. 这样写注入业务的dao
5. 下面的方法，有自己的业务需求默认实现类不满足的，可以重写对应方法，具体要重写哪个，等看后台框架的详细设计

## 站点自定义实现类的实例化

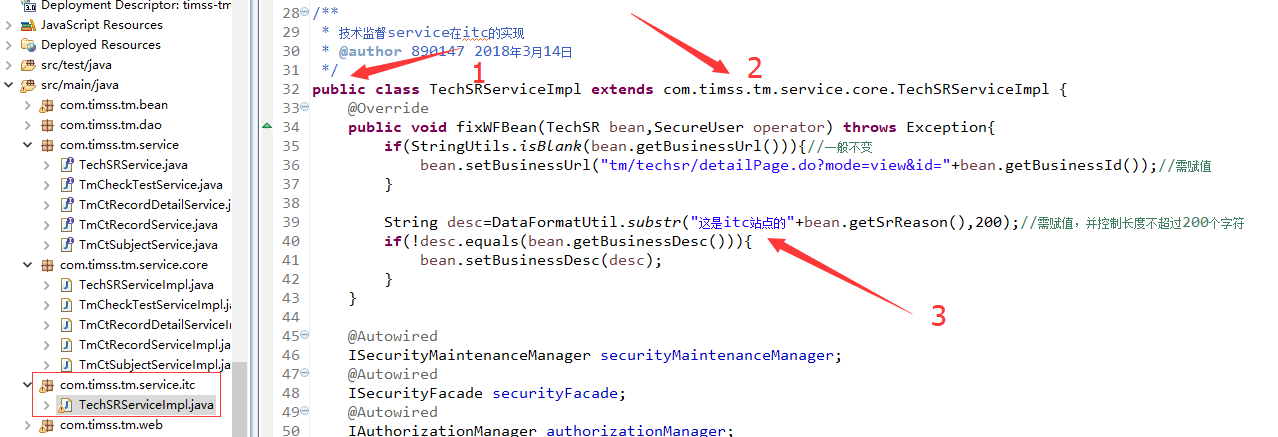
站点个性化的重写如下图3.5-2所示：

图3.5-2

用于某个站点的个性化需求和通用实现类有少许不同的时候

1、不用加service注解，同样的文件名

2、直接继承自core里的实现类

3、重写需要修改的方法

框架是根据文件名，去找指定路径的指定文件名的类再来实例化，所以文件的路径名和文件名不要弄错,如下图3.6-3所示：

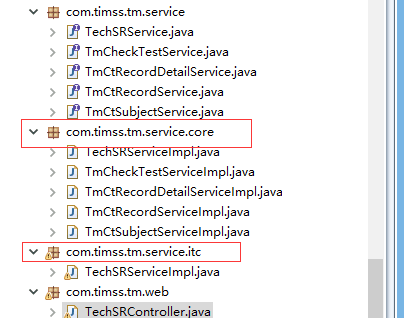
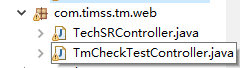


图3.6-3

## Controller

如下图3.6-1所示：



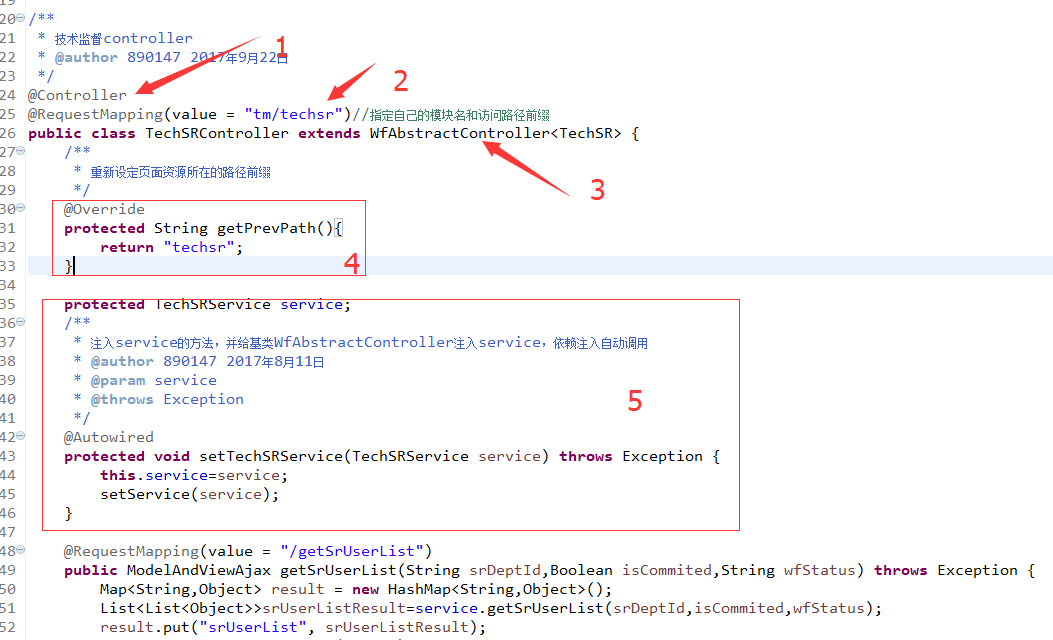


图3.6-1

1、加注解@Controller

2、加注解@RequestMapping(value = "tm/techsr")指定这个controller的映射路径，要两级，模块编码/功能编码

3、继承自基类WfAbstractController

4、指定页面资源的路径，如下图3.6-2所示:

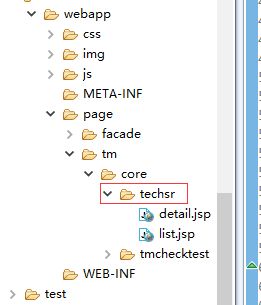


图3.6-2

5、用于注入业务service

6、同样重写需要修改的方法，或者映射的路径@RequestMapping(value = "/XXX")

7、返回的数据，json用ModelAndViewAjax，页面用ModelAndViewPage，写法参考基类

1. **前端框架及编码**

## 概述

一般一个功能，需要两个页面，列表页面和详情页面，list展示数据的列表，用户双击list中的一条数据，跳到detail页面展示数据的详情。目录如下图4.1-1所示

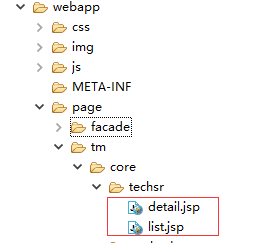


图4.1-1

具体设计思路：框架定义了一堆配置项和默认的操作方法，开发者通过修改配置项渲染页面和功能，通过修改默认的操作方法，实现自己业务需要的操作。

## 列表页

### 代码结构

如下图4.3.1-1所示

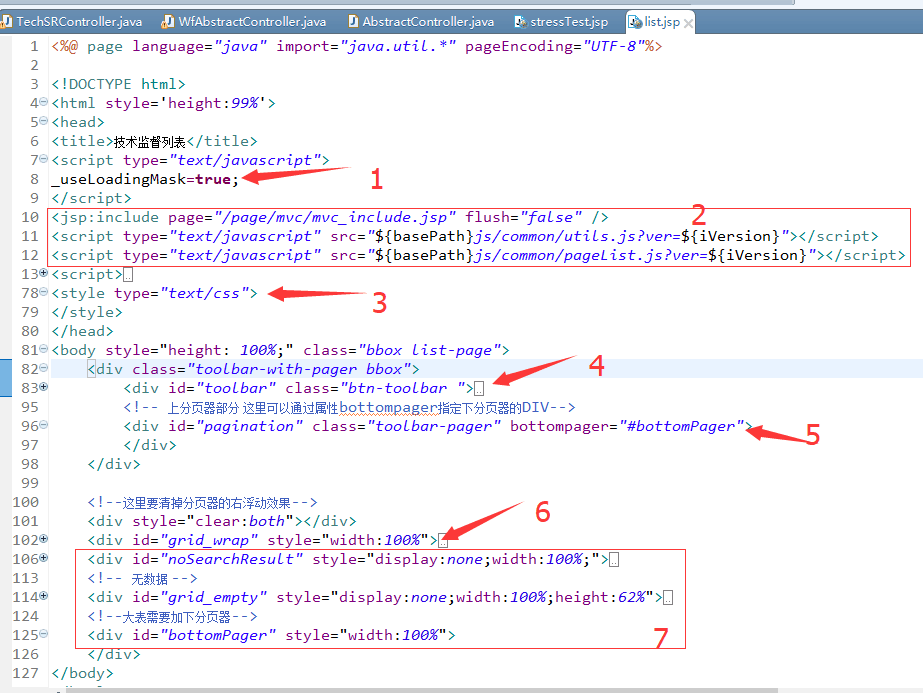


图4.3.1-1

1、使用遮罩层在数据未加载完成时

2、引入框架js

3、页面js，自定义各种操作的地方，也是框架的入口

4、按钮栏

5、分页器栏

6、页面无数据的提示和下分页器

一般我们需要修改3和4，其他基本不用改。有时6里面的提示信息也需要修改。

### 页面js

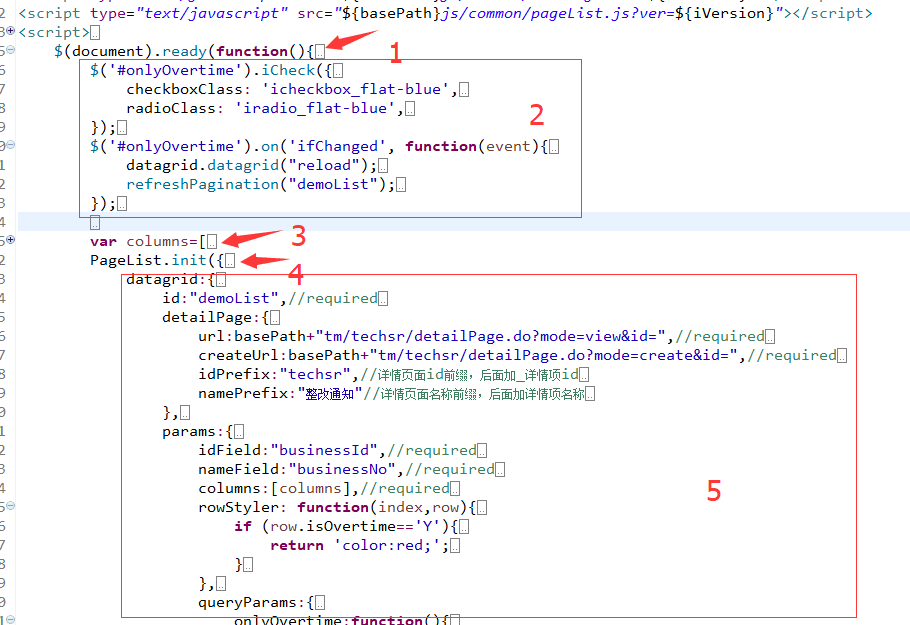


图4.3.2-1

1、页面渲染结束执行用户js的入口

2、各种自定义操作和对框架默认方法的重写

3、列表定义的列

4、框架初始化入口

1. 框架的配置项修改
   * 1. **按钮栏**

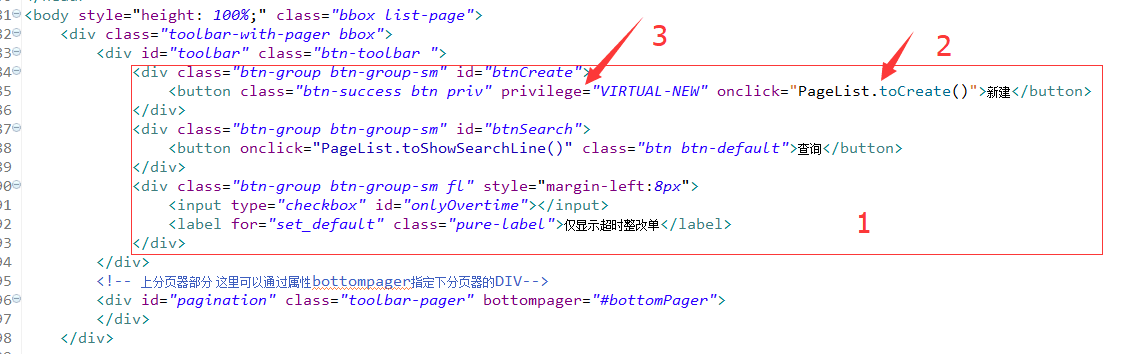


图4.3.3-1

1、自定义按钮和控件

2、按钮点击事件使用框架提供的方法

1. 按钮权限控制部分，wiki上有写，暂时不关注

## 详情页

### 代码结构

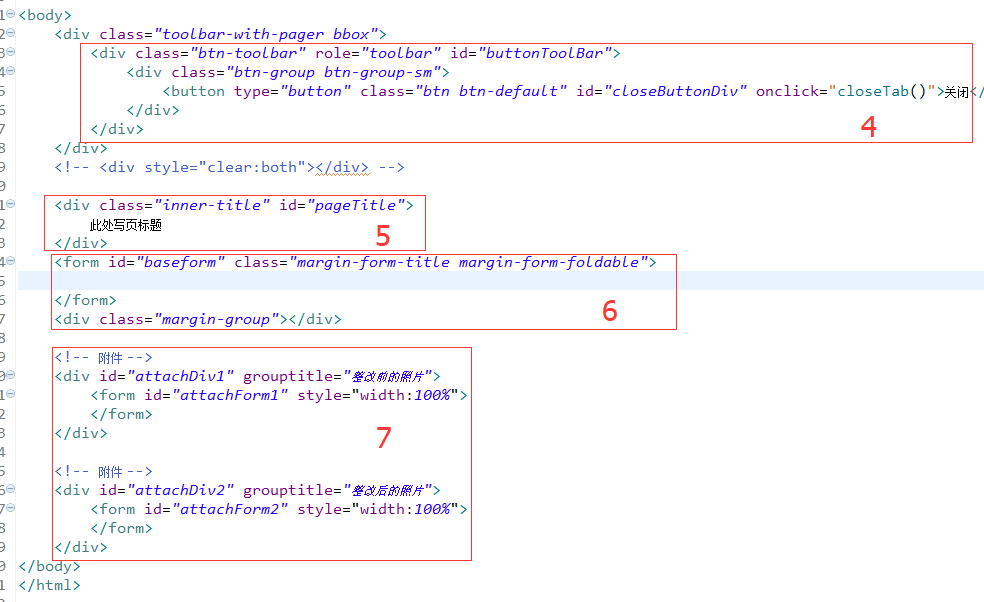
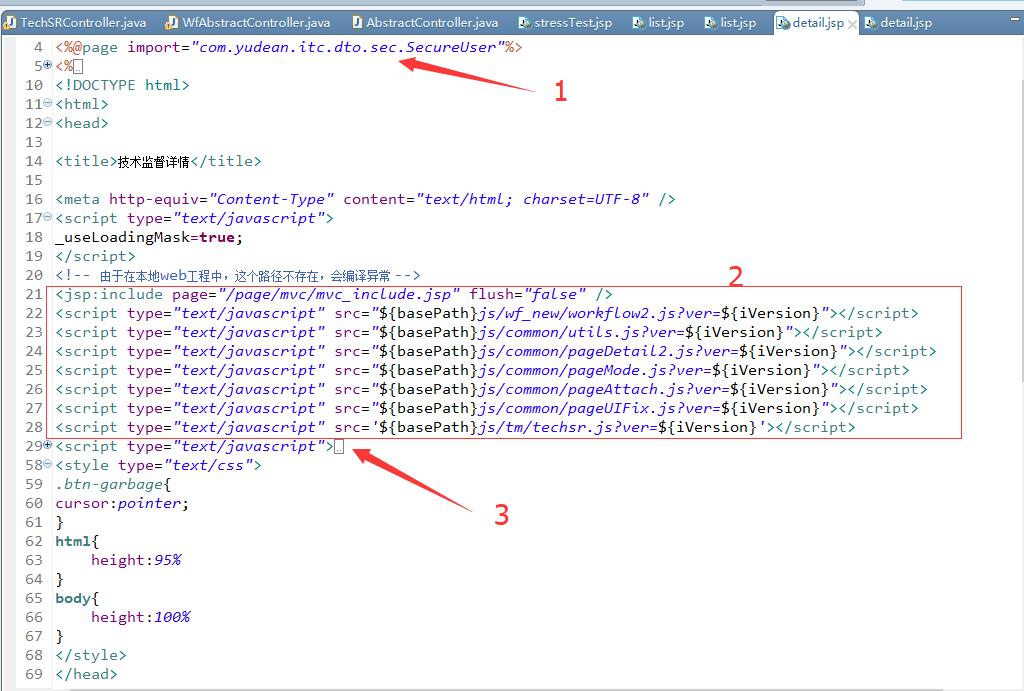


图4.3.4-1

1、获取系统参数，照抄

2、引入js，根据详情页面需要使用的功能，引入对应的组件js

3、页面js和框架入口

4、按钮区，目前有工作流的功能，按钮由配置生成，所以这里放了关闭按钮

5、页面标题

6、主表单

7、附件或者其他子表单、子列表等，根据自己的业务需求对应添加

除了2、3、7，其他也基本不用改

### 页面js

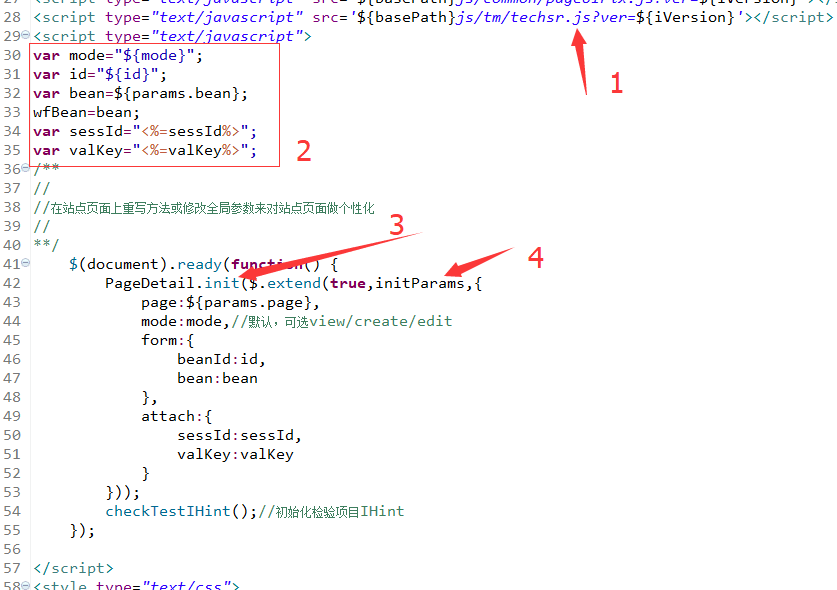


图4.4.2-1

1. 把各种自定义操作和对框架默认方法的重写、配置项的修改，都放js

中了，方便复用

2、页面定义的变量，不放js里，是因为需要从jsp中取值

3、框架初始化的入口

4、1的js修改的配置项

### 前端框架的使用

框架有哪些方法可以重写，有哪些配置项可以用，直接查看框架对应的js文件，配置项都写在头部的objs里了，下面的所有方法，都是可以重写的！所以很灵活很自由也很省心啦，详细说明也要看前端框架的设计文档了

# 通过配置渲染前端框架

配置项渲染可以让前端写起来更省心，即PageModule组件，根据用户的配置，自动渲染页面。具体用户如何配置，参考文档“新工作流与快速表单使用手册”。

如下图图5.1-1所示：

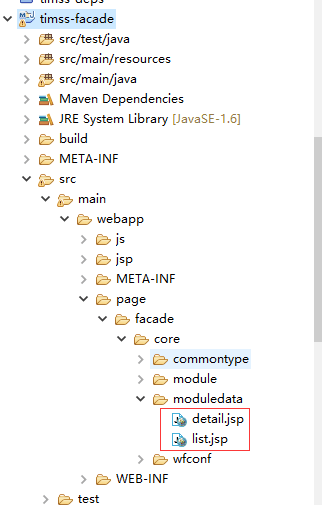


图5.1-1

## 列表页

如下图5.2-1所示

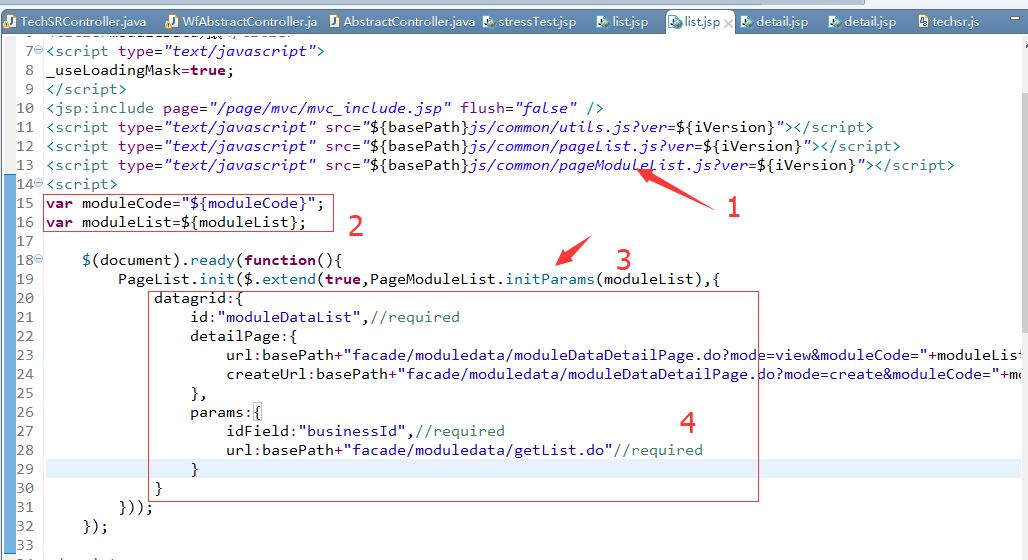


图5.2-1

1、引用组件PageModuleList

2、提供给组件，告知功能编码和配置的参数

3、组件根据配置，渲染出框架需要的初始化参数

4、对组件渲染的参数还需要修正

5、其他和普通列表页是一样的

## 详情页

如下图5.3-1所示：

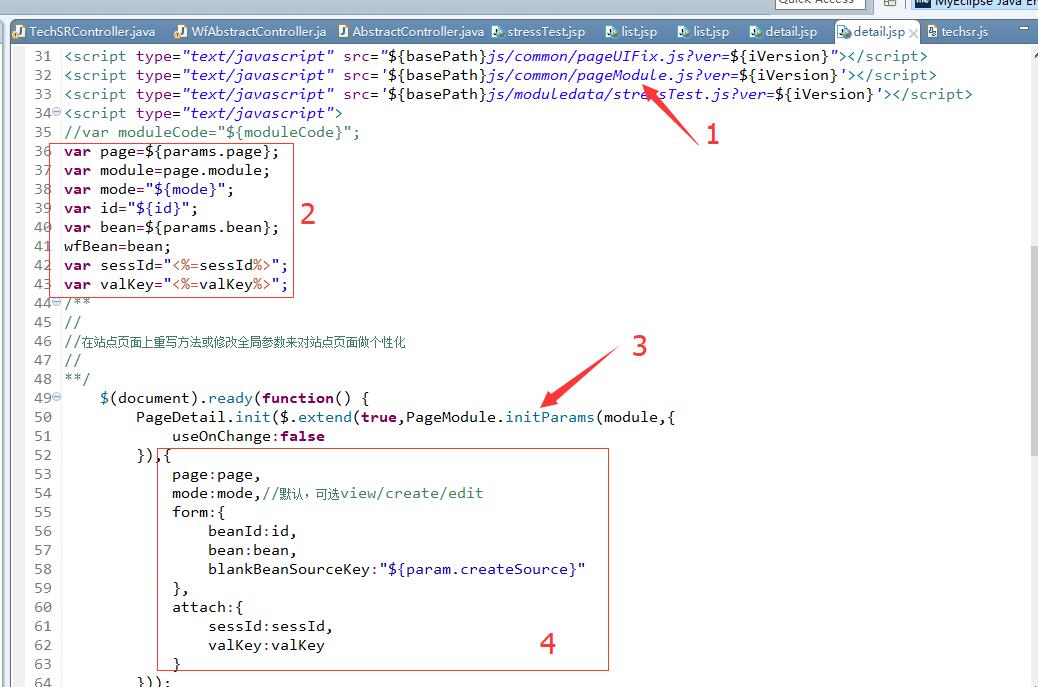


图5.3-1

1、引入组件PageModule

2、参数和配置，照抄好了

3、组件根据配置渲染初始化参数

4、修正参数