# Jfish技术指南

# 简介

jfish是基于spring轻量级容器的一个mvc web项目模板。

mvc框架使用的是spring mvc，view层可以使用jsp或freemarker。

持久层使用的hibernate和spring jdbc。

项目使用maven构建和管理依赖。

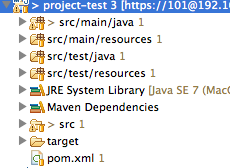
# 开发环境要求

如果使用eclipse作为开发环境，请安装maven插件。

如果需要，请安装freemarker的ide插件。

# 项目结构

整个项目目录机构如下：



主代码和资源目录

/src/main目录包含了项目的主要代码和资源文件。

项目的java代码都放在/src/mian/java下，

资源文件，比如xml，properties文件放在/src/main/resources。

测试代码和资源目录

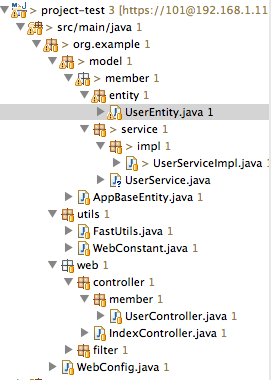
/src/test目录主要包含了项目的测试代码和资源文件。

测试的java代码主要放在/src/test/java下，

测试的资源文件主要放在/src/test/resources下。

## 主代码目录/src/main/java

/src/main/java目录结构：



包的划分按照先项目架构结构和模块混合的方式。

如上图，首先按照项目结构划分为model、web、utils等包。

### Model包：

model包，是广义上的model，是放实体和业务代码（dao、service）的地方，在model包下再按功能划分模块，比如上图的member模块。

在模块member下面的entity子包放置实体类，service包放置业务接口类，service下面的impl包放置业务接口的实现类。

### Web包（Controller）：

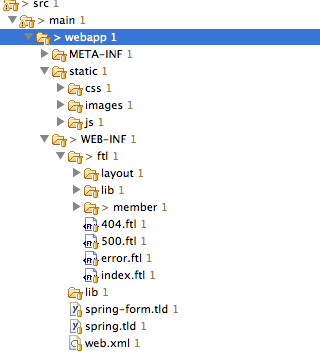
主要包含和web层相关的代码，和model包一样，可按模块功能划分包。

这个包里面最主要的是controller包，放置spring controller的相关代码。

### View目录

/src/main目录下面还有一个webapp目录，这是作为mvc中的view层代码存放目录，因为view层使用的是freemarker技术，所以这个一般就是web资源文件和ftl文件的存放位置。

View层目录结构如下图：



项目默认已经有了基本的文件，比如404和500错误文件。

这个目录下也是按功能划分子目录，比如member目录，放置就是member模块的ftl文件。

static目录

放置css、js、图片之类的静态文件。

WEB-INF下面的flt目录

放置view层freemarker文件的目录。

这个目录下的layout目录，是放置布局文件的地方，项目默认会有一个application.ftl布局文件，里面定义了界面的整体布局，和那些地方是可以被子页面覆盖替换的。

Lib目录下一般放置编写view层ftl文件过程中的自定义宏。

### Utils包：

主要放置相关的实用类工具类等各种helper类型代码。

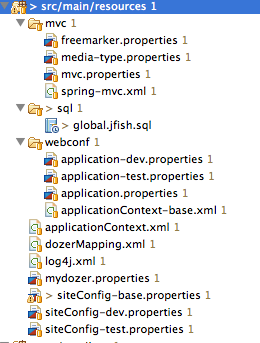
默认项目已经有两个类：

FastUtils：里面包含有一些处理xml、json和bean复制的相关方法。

WebConstant：这是一个空类，用于集中放置项目中用到的各种全局常量。

顶层还有一个WebConfig类，通过里面的getBaseSiteConfig方法，可以获取/src/main/resources/sitecofig-base.properties文件配置的各种参数。

## 主资源目录/src/main/resources



项目默认的资源目录主要分为顶层、mvc、sql、webconf三个目录。

### 顶层目录：

mydozer.properties和dozerMapping.xml：

开源库dozer的配置文件。

Log4j.xml：

日志配置文件。

applicationContext.xml：

spring容器的配置文件。

siteConfig-\*.properties：

以siteConfig开头的几个文件是web应用的各种全局配置文件。其中的后缀表示不同环境的配置，项目会根据siteConfig-base.properties文件里配置的app.environment加载不同的配置文件，比如app.environment=test，将会加载siteConfig-base.properties文件后，加载siteConfig-test.property文件里的配置，如果配置名称相同，后者的配置将会覆盖前者的配置。

### Mvc目录

这里的几个文件都是spring mvc的一些基本配置支持，基本上不用怎么修改。

其中freemarker.properties是配置view层的freemarker的，具体配置参考freemarker的相关配置。

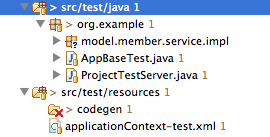
### Webconf目录

这个目录下的配置文件是spring容器的基本配置文件。

Application-\*.properties主要是有关数据源连接信息的配置。

ApplicationContext-base.xml主要是各种环境下（比如dev、test、product）数据源的配置。

## 测试代码和资源目录



这是放置测试代码和资源文件的目录，项目默认已经有了一些文件，其中比较重要的是AppBaseTes类，这个是项目的测试基类，一般的单元测试可以继承这个类，这个基类会启动spring容器和相关的数据库资源。所以，如果测试不需要用到这些，就没必要继承这个类。

另一个ProjectTestServer类是项目开发的启动类。

一般来说，在开发期间，不需要部署到tomcat服务器，因为项目已经内嵌了一个tomcat，启动这个类就会启动整个项目，然后进行开发和测试。

如果ide启动失败，有可能是ide启动jvm的默认内存太小，建议配置更大的内存，比如：

-Xms512m

-Xmx1024m

# controller

## 约定

控制器主要工作是接受请求，调用service的业务方法，根据需要返回到相应页面，这里不应该包含任何业务逻辑代码。

一个controller的类最好要以Controller为后缀，各action方法命名能简单表示这个action所要做的事。

## rest风格的路由规则

在controller层，建议一般的crud（增查改删）的url映射和action方法映射规则遵循rest风格的url映射规则，即url+http method对应一个controller方法的方式，比如member包下的UserController映射规则：

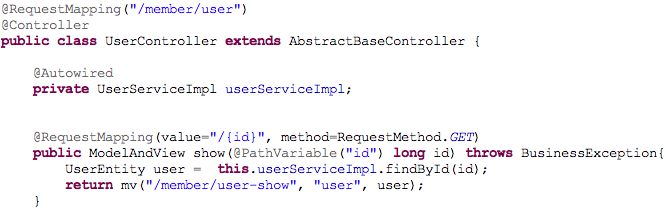
|  |  |  |
| --- | --- | --- |
| url | http method | controller方法 |
| /member/user | get | index |
| /member/user/new | get | \_new |
| /member/user/{id} | get | show |
| /member/user/{id}/edit | get | edit |
| /member/user | post | create |
| /member/user/{id} | put | Update |
| /member/user | delete | deleteBatch |
| /member/user/{id} | delete | delete |

其它类型的url路由最好也遵循类似的规则，比如一些获取的操作，应该使用get方法，而一些创建、更新或者删除的动作，则对应使用post、update或者delete方法，是一个良好的习惯。

## 如何使一个类成为controller

这部分其实是spring mvc方面的知识。

要使一个类成为controller，要在类名上面使用@Controller注解，并使用@RequestMapping 注解映射通过浏览器可以访问的url，如下：



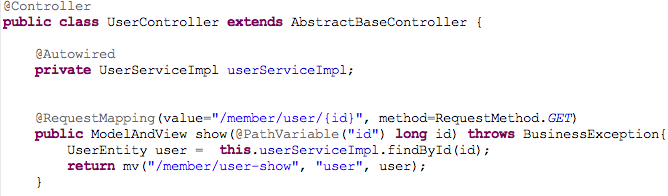
（一般controller最好继承项目提供的AbstractBaseController ，里面有提供一些实用的方法，当然这不是必须的。）

这就是你要让一个类成为controller的所有工作了。Spring容器会在启动的时候扫描这些标注了注解的类作为web controller。

Spring会把类上面的请求映射和方法上的映射合并起来成为一个完整的请求路径，比如要访问上面的show方法，可以通过在浏览器输入以下地址访问：

<http://localhost:8080/projecttest/member/user/1>

实际上，类上面的@RequestMapping注解不是必须的，完全可以把完整的路径映射都放在action方法上，比如：



在方法里，通过mv方法返回了一个ModelAndView对象，mv方法的第一个参数是view页面的路径，余下的参数通过key1, value1, key2, value2成对出现，实际会被转成键值对添加到model里（这里不是上文说的model包，而是侠义的spring mvc中的model对象），以便让view层的ftl页面通过freemarker的表达式语法访问数据的值。

Mv方法只是父类提供的便捷方法，实际上你完全可以自己构造一个ModelAndView对象，或者字符串，更多的使用方法参考spring mvc文档。

# 实体映射

这部分参见hibernate文档的内容。

# service

service是实现业务逻辑的代码，一般来说，一个完整的操作封装成一个service方法。所以，事务的边界也是service方法开始和结束的。

按照流行的做法，一个entity对应着一个service，而按照标准面对接口编程做法，个service接口和一个service实现类。但实践下来，你会发现web应用中一般的crud，这种做法极其铺张浪费，会导致项目中一大堆空类和无用或者模糊不清的接口类，而所得到好处不大。所以在实际开发中，可以不用很死板地规定一个实体对应着一个service接口，接口再对应着一个实现类。

实际开发中，service的粒度可以大些，可以管理多个实体的逻辑操作，也没必要必须实现接口，是否需要接口按需要去做。

## 约定

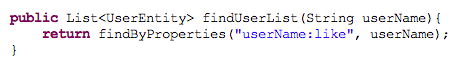
service类的命名，接口的话一般以Service为后缀，实现类以ServcieImpl为后缀。

## 注解service类

service类一般以spring的@Service注解标注，如下：

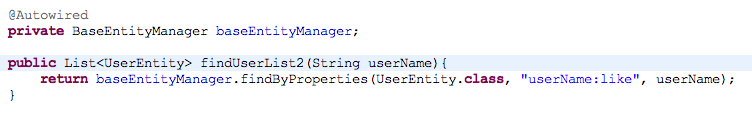


一般的service可以直接继承泛型的抽象类HibernateCrudServiceImpl，以获得一些便捷的增删改查方法，比如通过名称查询用户表，可以这样写：



findByProperties方法就是继承自父类的。

但集成不是必须的，如果没有继续JFishCrudServiceImpl，则上面的查询可以这样写：



一样很简单，区别只在如果没有继承，要通过手动注入BaseEntityManager对象后，才能调用findByProperties方法，而且要多传入一个参数，即实体类型。

实际上，HibernateCrudServiceImpl的便利方法都来自BaseEntityManager接口。

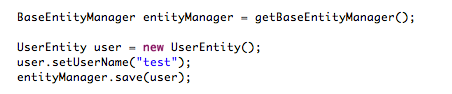
## Dao或BaseEntityManager

按照以前流行的做法，通常dao层也和service层一样，一个实体对应一个dao，dao有接口有实现类……但这真的没有必要，够用的话，一个通用的dao或者说实体管理器就够了。

在这个项目，这个通用的dao或者说实体管理就是BaseEntityManager接口。

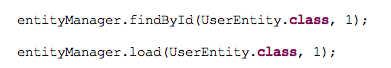
BaseEntityManager的接口定义了一系列增删改查的方法，使用方法如下：

### 保存或者更新实体：



实际上，save方法会据跟实体的id是否有值去判断该调用插入还是更新操作。

### 通过主键查找实体：



这两个方法的区别在，load方法如果找不到会抛出JFishEntityNotFoundException，而findById返回null值。

### 删除实体



前者通过id删除，后者通过实体对象删除。

### 简易的属性查询接口

#### 一般的条件查询



条件以键值对的形式出现，键就是实体的属性名称。

意思就是查找t\_user表里user \_name=’test’ and age=25 的用户。

注意，默认的条件操作符是：=

如果想userName用like，age大于等于25岁的条件查询，可以这样：



属性和条件操作符通过冒号分割。

#### 动态条件查询

动态条件查询怎么办？比如userName变量为null值时，忽略这个条件。

一般的方法无非就是加个if条件判断，但在这里，你什么都不用做，BaseEntityManager将会忽略条件值为null的条件，从而节省你大量啰嗦的if判断。

BaseEntityManager已经内置了分页查询，如果想查询一个分页对象，如下：



findPage方法基本上和findByProperties方法没啥不同，只是要传入一个page对象。

如果只是想查询符合条件的单挑记录，则使用findUnique接口：



如果想count符合条件的记录，则使用countRecord接口:



#### 属性查询支持的操作符

属性的条件查询支持以下操作符：

|  |  |  |
| --- | --- | --- |
| 操作 | 符号 | 说明 |
| 等于 | = | (…“userName”, “test”) |
| 大于 | > | (…“age:>”, 23) |
| 大于或等于 | >= |  |
| 小于 | < |  |
| 小于或等于 | <= |  |
| 不等于 | != , <> |  |
| 是否为null | is null | (…“userName:is null”, true) |
| 模糊查询 | like, not like |  |
| in查询 | in, not in |  |
| 日期查询 | date in | date in 操作符一般需要两个参数，表示一个日期区间，如果只传入一个Date对象，则会自动把这个Date对象解释成date对象所代表的那天的开始和结束时间。如：  (…”birthDay:date in”, new Object[]{date1, date2}) |

属性的条件查询支持数组或list参数，大多数操作符都会对数组和list参数生成or条件。

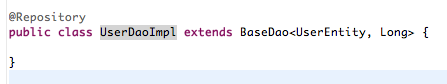
### 复杂的查询

复杂的查询一般指不能通过简易查询接口查询的查询。

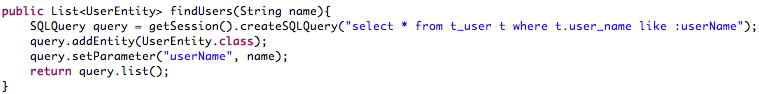
复杂的查询建议建立Dao对象进行封装，然后在service里通过注入dao对象来调用。

#### Dao

为了方便，可以直接继承泛型类BaseDao，当然也可以不继承，直接注入BaseEntityManager或者SessionFactory，但别忘了加@Repository注解，如：



在dao里，你可以使用hibernate的原生api来进行数据访问的逻辑，你可以通过hibernate的session api创建一个SQLQuery查询，比如：



当然，不止SQLQuery，有了原生session对象，你就可以调用各种原生的hibernate查询api。

# view层

view层就是界面，负责生成最终的用户界面，一般用模板技术实现，只负责显示逻辑。

模板可以选用jsp或者freemarker。

## jsp

主要使用jsp el、jstl、spring tag和一些自定义的标签为主。除了某些例外，jsp页面一般禁止直接用java编写。

### 标签

#### Layout标签

Layout标签是用来实现网站整体风格布局的标签。

标签有三个：

<layout:extends>

<layout:defined>

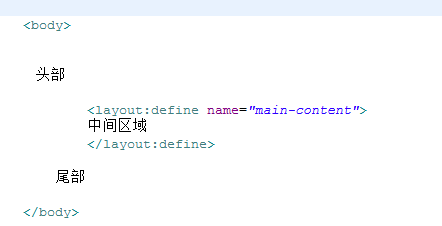
<layout:override>

一个网站的整体风格是固定，也就是说，无论任何页面，某些区域总是一样的，比如头部和尾部，不同的只是少数区域，比如中间部分，标题部分。

通过laytou标签，可以把相对固定的部分写在一起，然后把变化的区域定义起来，再通过子页面覆写。

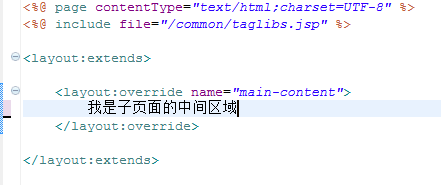
在项目的/views/layout/目录下，有一个application.jsp文件，这个jsp文件就是布局文件，里面定义了一个网站的整体风格，而变化的区域通过<layout:defined>定义，可以把这个定义看成一个变量的定义，而把application.jsp当作是一个父页面，如：

application.jsp



然后可以在子页面通过<layout:override>覆写中间区域的内容，如：

child.jsp



渲染child.jsp页面时，最终把两个页面的内容合并在一起：

头部

我是子页面的中间区域

尾部

父页面通过<layout:defined>的部分最终被子页面的通过<layout:override>覆写了。

## freemarker

freemarker是一个简单而功能齐备的模板引擎，是一个可以生成各种文本输出的生成工具，也就是说它不是web专用，但完全可以用来取代jsp作为web应用的模板语言。

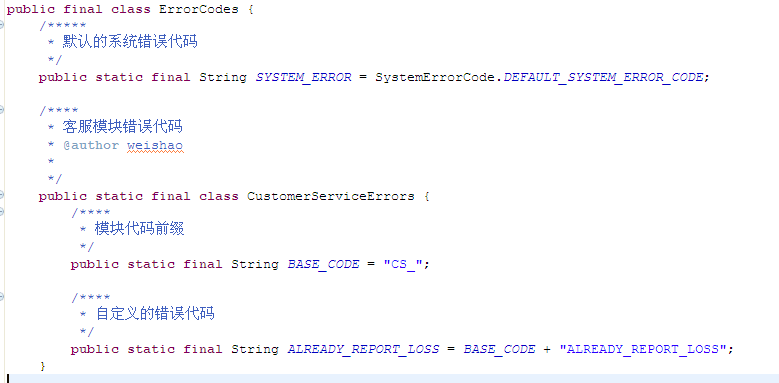
参见freemarker官方文档。

# 异常处理

所有service中的异常处理均抛出JFishBusinessException或JFishServiceException及其子类异常。

JFishBusinessException和JFishServiceException的区别是，前者是受检异常，excepton的子类后者是非受检异常，runtimeException的子类。在service里抛出前者，事务不会回滚，抛出后者，事务回自动回滚。一般抛出后者即可。

系统所有的业务异常错误都应该定义ErrorCodes类，比如客服模块的错误，可以定义如下：



不同模块有一个固定的前缀以示区分，比如客服就是CS\_，自定义的错误代码就是模块代码加上自定义的代码，所以上面的ALREADY\_REPORT\_LOSS常量的完整错误代码就是：CS\_ALREADY\_REPORT\_LOSS

代码的错误消息定义在

/main/resources/message/ExceptionMessages.properties

文件里，如：



Service抛出异常时只需要用错误代码创建一个JFishServiceException即可。如：



抛出的业务异常将会统一处理，并且在错误页面显示在

ExceptionMessages.properties 里配置的错误信息。