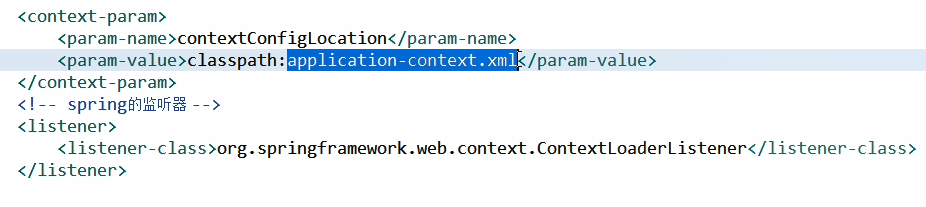
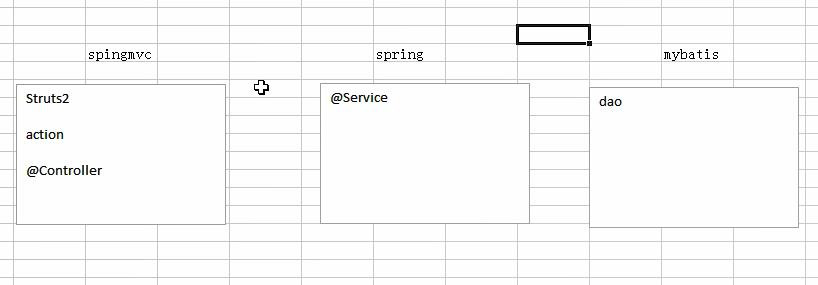
# Day-01

## Spring容器监听器

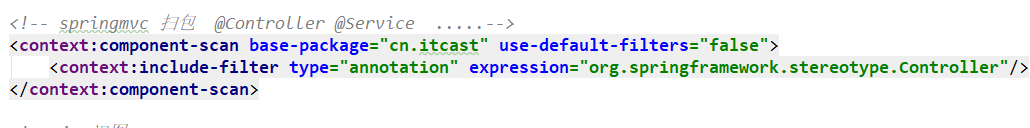


服务启动时加载spring配置文件

## 扫描包实例化类

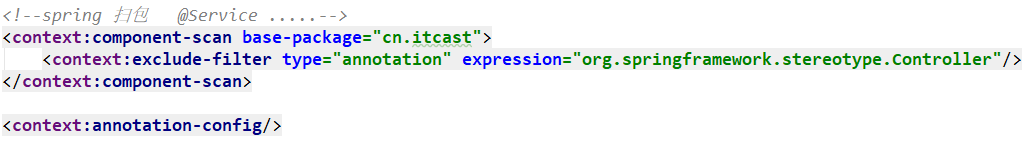


### springMvc负责扫描Controller



### spring 负责扫描 Service

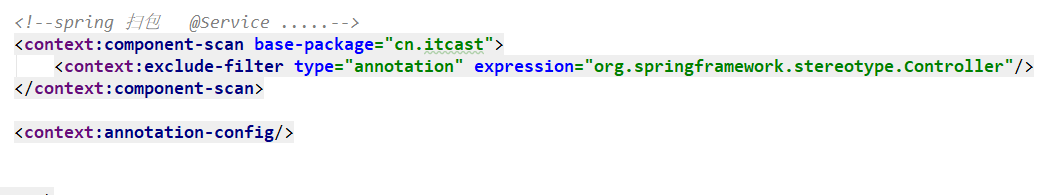
base-package没有具体指定到service，是因为service位置可能不确定，但是基础包是确定的。扫描基础包后，排除扫描controller



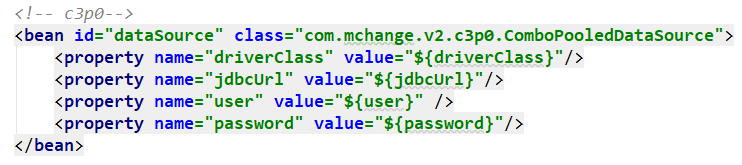
### mybatis 负责扫描 dao

## 将spring配置按照功能拆分

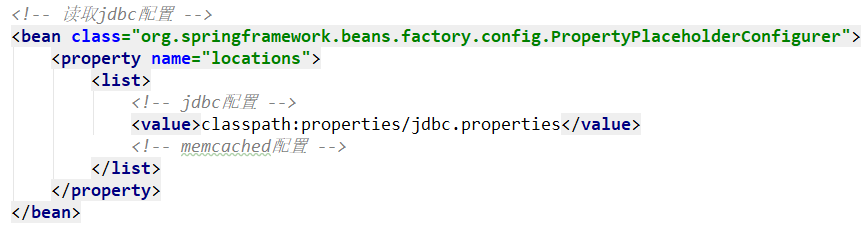
### 注解式扫描包，实例化类



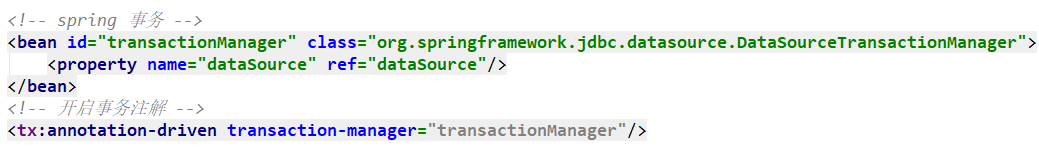
### C3p0数据源



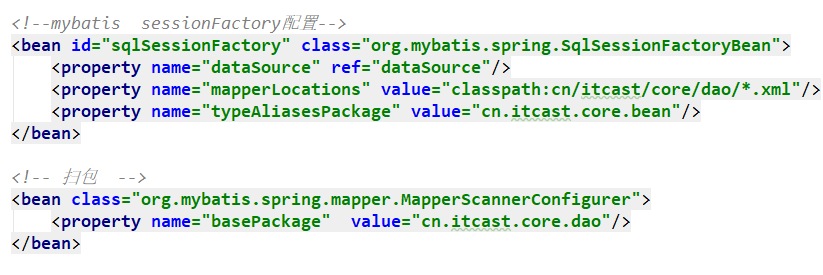
### 读取properties配置文件



### 配置事物

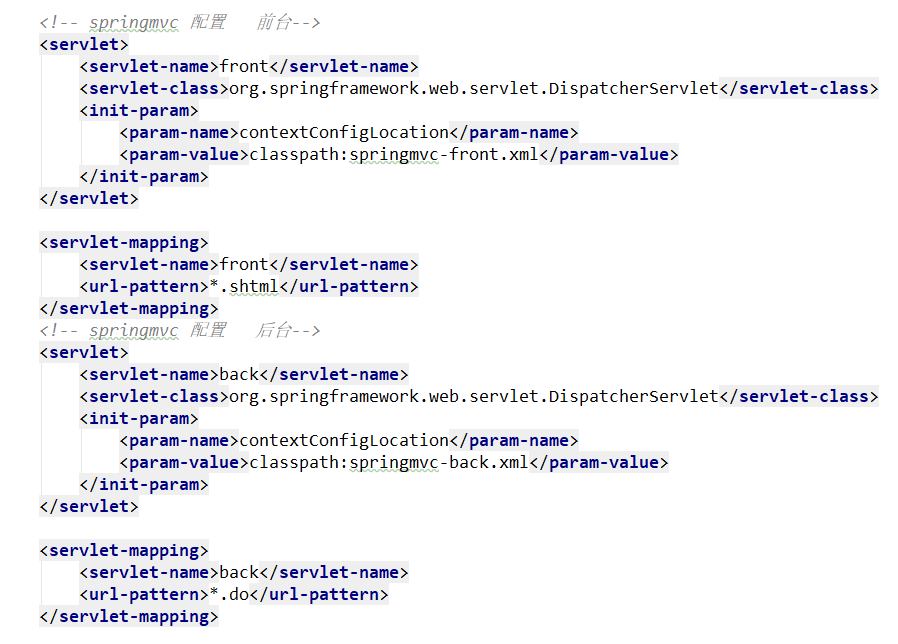


### Mybatis配置



## Springmvc配置

分为前台和后台



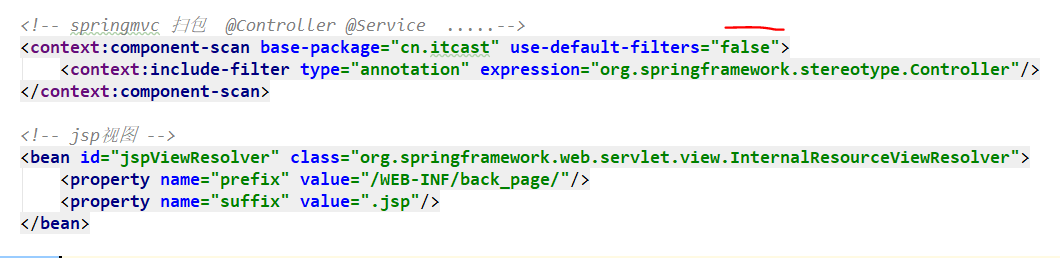
配置springmvc只扫描controller层

**use-default-filters="false"** 为不使用默认过滤器

当不配置时为使用默认过滤器，默认过滤器为先后扫描controller,service,comment 即扫描完controller后扫描service， 这样的话会与spring的扫描内容重复，springmvc是spring的子类，

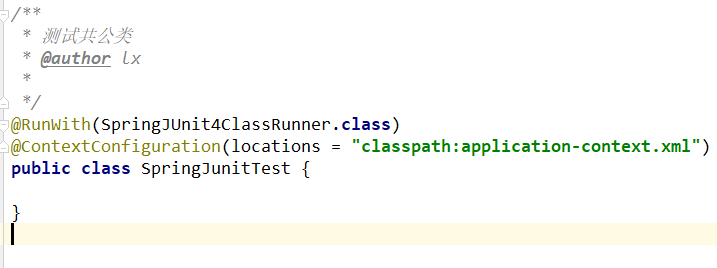
所以，为避免重复扫描，在springmvc扫描controller时，不使用默认的过滤器

则可以将**use-default-filters="false"设**置，扫描完controller后，不继续扫描

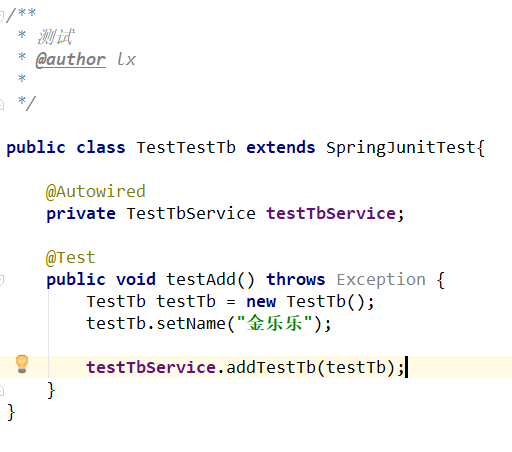


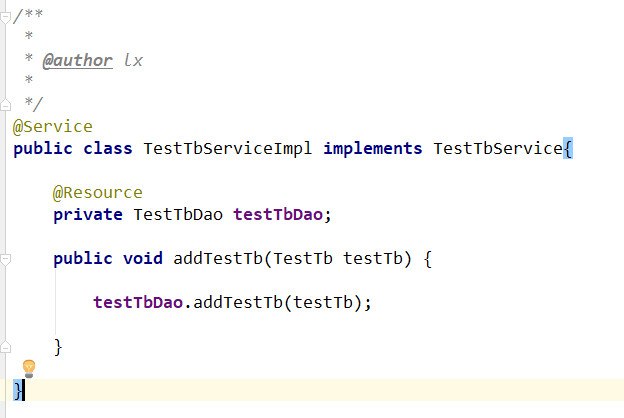
## 通用测试

## 通用测试类



## 继承通用测试类



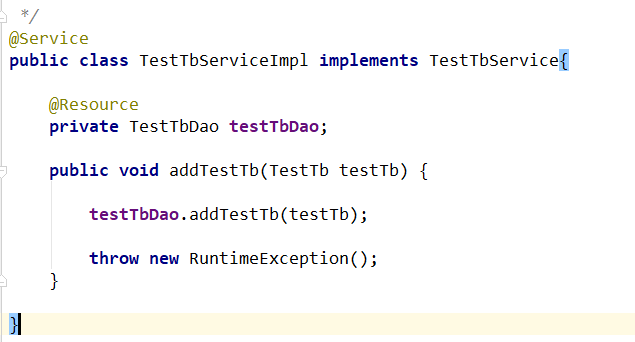


在未开启事务情况下，数据将正常提交到数据库

## 测试事物

### 未开启事务

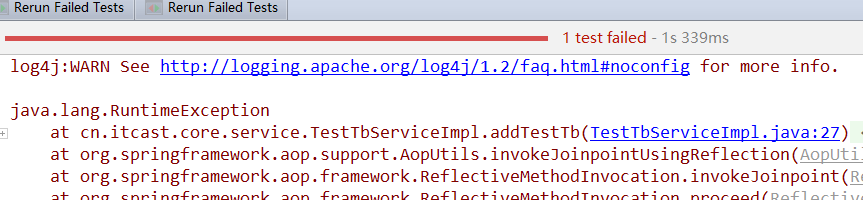
未开启事务，在抛出异常情况下，数据会提交到数据库，这是错误的



此种情况下，虽然抛出异常，但是数据会继续保存，不应该出现

### 开启事务

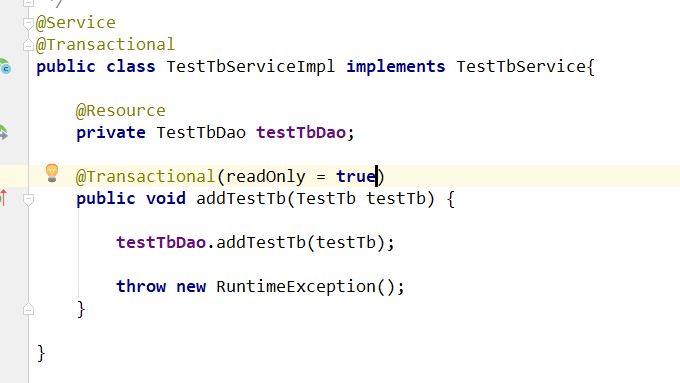




开启事务后，在抛出异常情况下，数据没有被保存，是正确的

### 关闭事务

如果在类上开启事务，方法中是查询语句，就是不需要事务，可以关闭事务

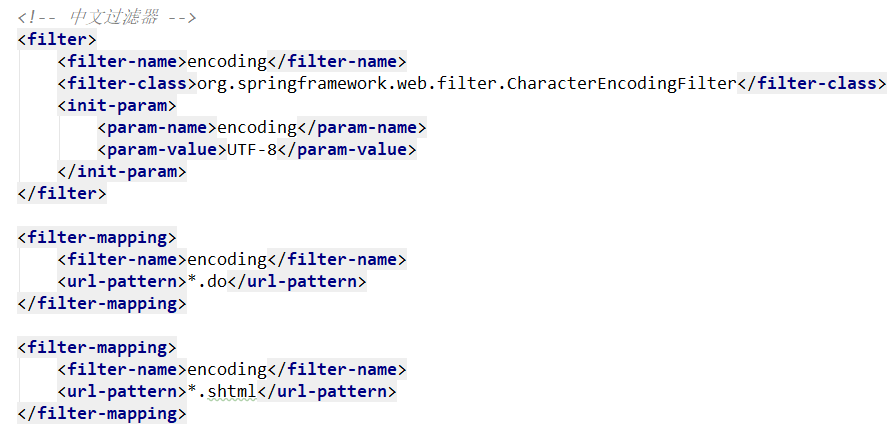


例如：关闭某个方法上的事务，多用在查询语句中

# Day-02

## 乱码问题

### 解决提交中文乱码

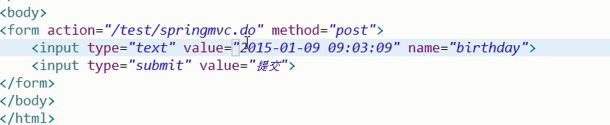


### 解决 get 提交中文乱码

在tomcat 中 server.xml 的 8080端口下添加



## 日期配置





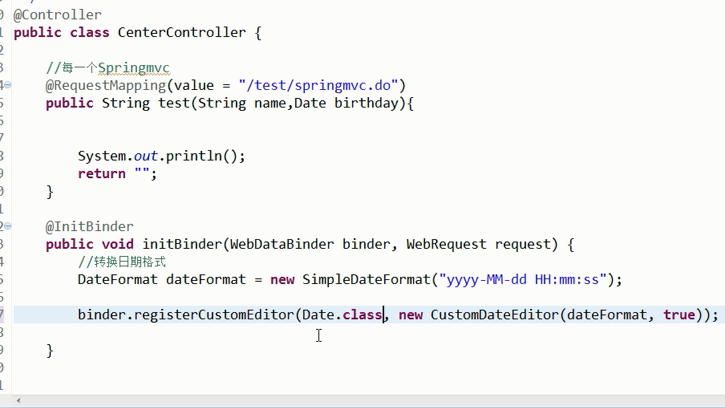
出现格式转化异常

### 局部配置

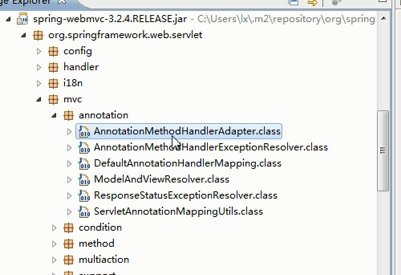
在需要做日期转换的controller中实现

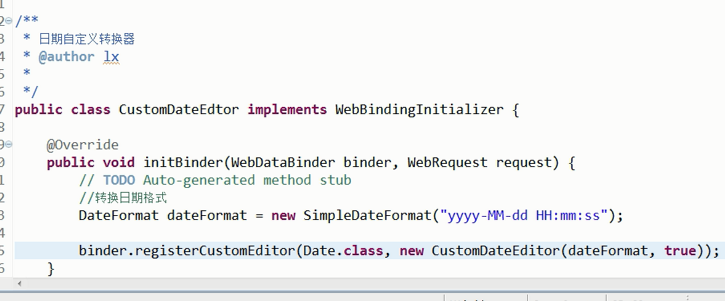


注解方式也可以不需要实现类



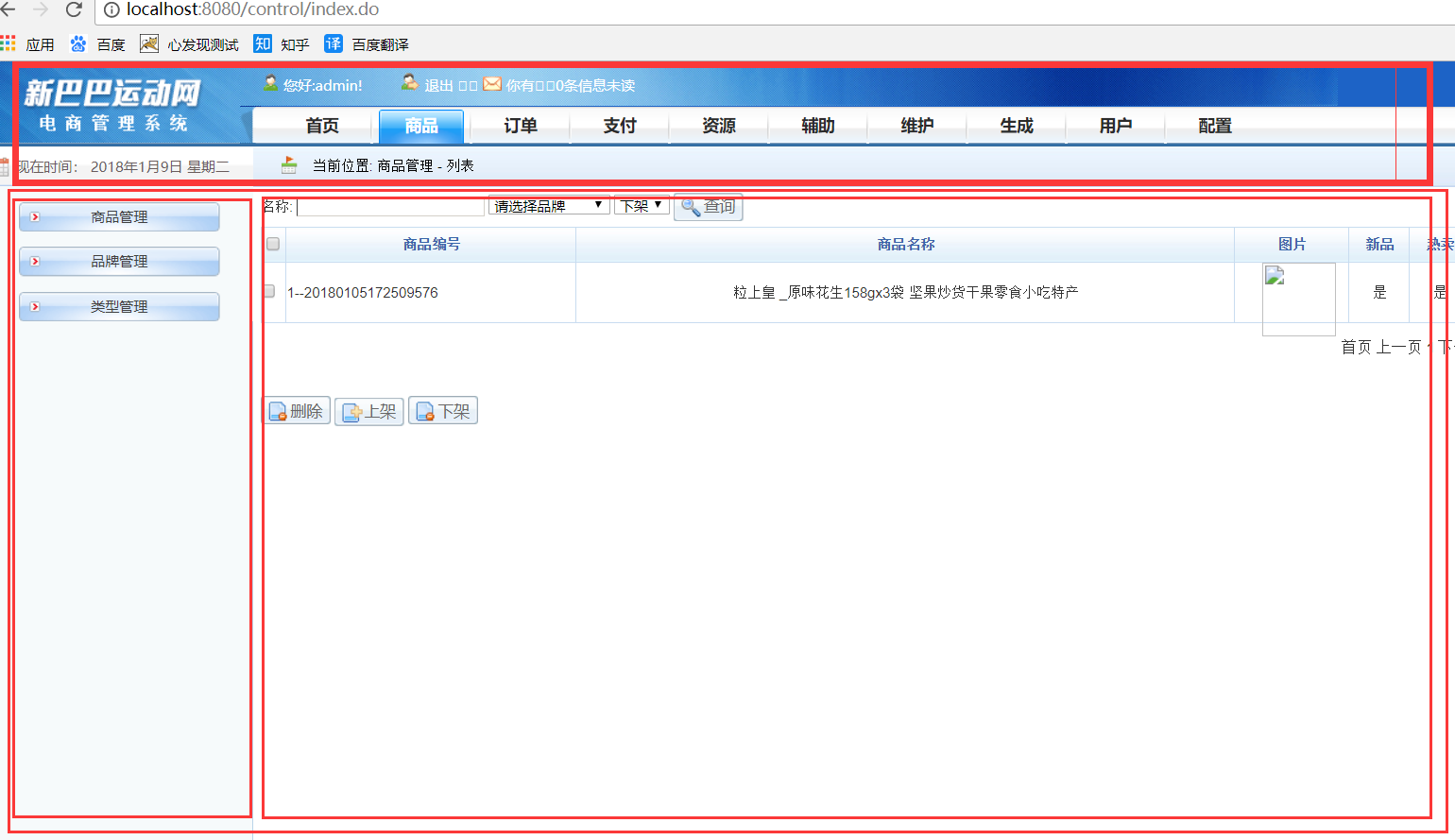
### 全局配置





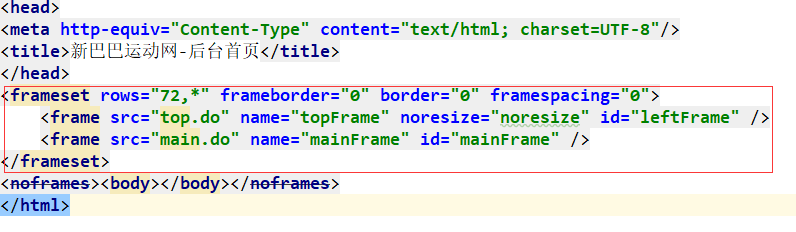


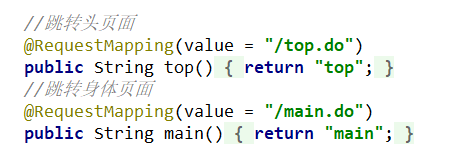
## 后台页面布局



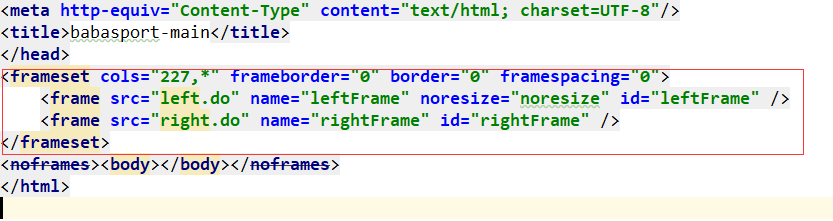
页面主要分为 头和身体 , 身体部分包括 左侧和右侧

### 1, 页面划分为头和身体



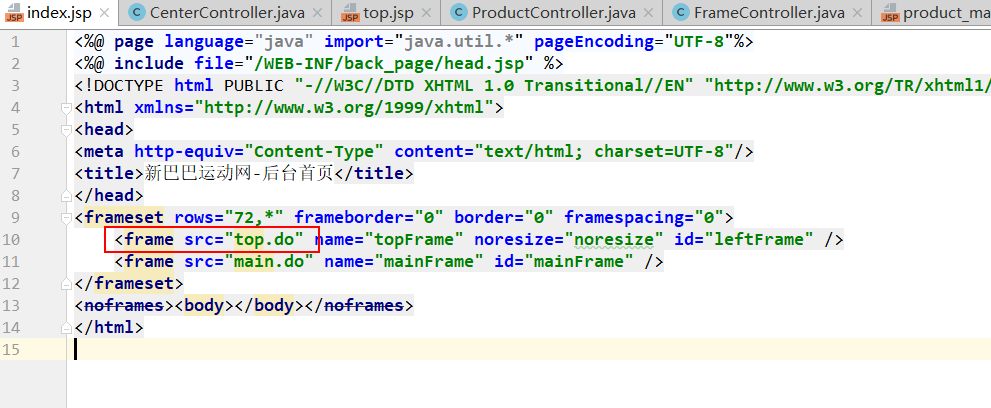


### 2, 身体分为左侧和右侧



## 相对路径和绝对路径





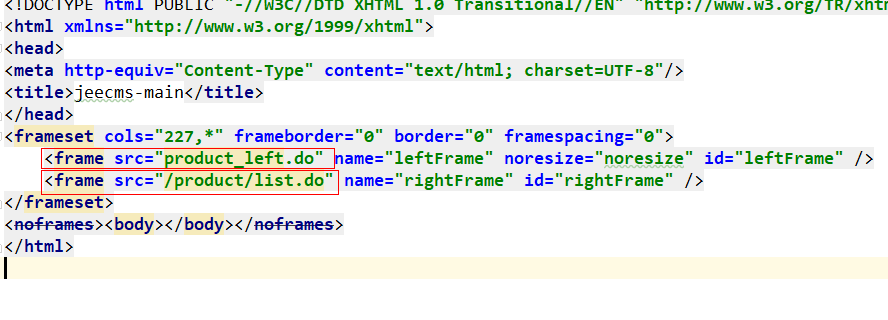
该页面内有两个跳转链接, 这两个路径均为相对路径

相对于 <http://localhost:8080/control/index.do>

<**frame src="main.do" name="mainFrame" id="mainFrame"** />

即为访问: <http://localhost:8080/control/main.do>

### 二者区别



以 / 开头为绝对路径, 否则为相对路径, / 代表项目根目录

## 分页

### 封装数据

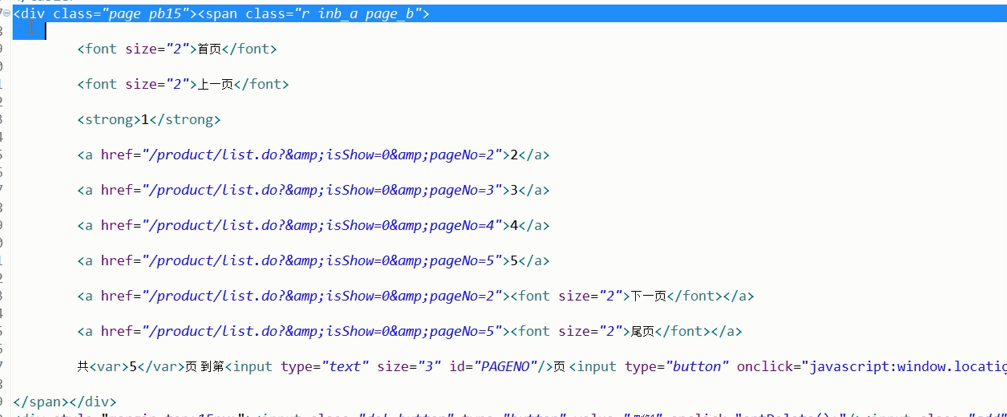


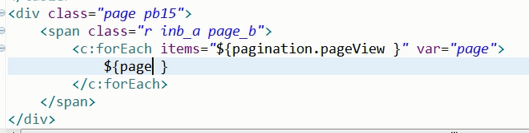


### 参数准备



### 页面部分





### 参数非空校验

分页查询参数为空导致查询异常, 需要对参数进行非空校验



## 列表页面加载

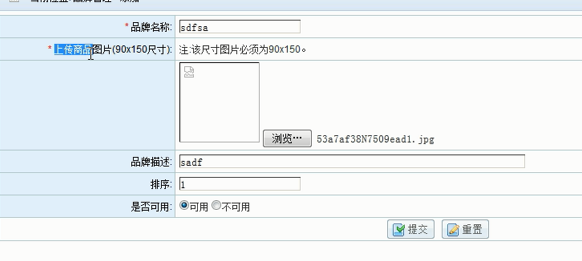
### 方法命名规则

所有的查找都用 get开头 ,禁止 find,select



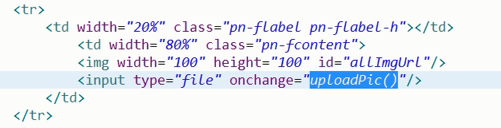
## 图片上传

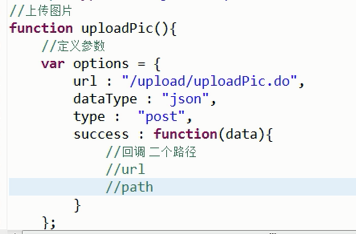
### 流程



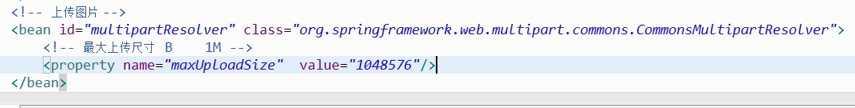


### 页面





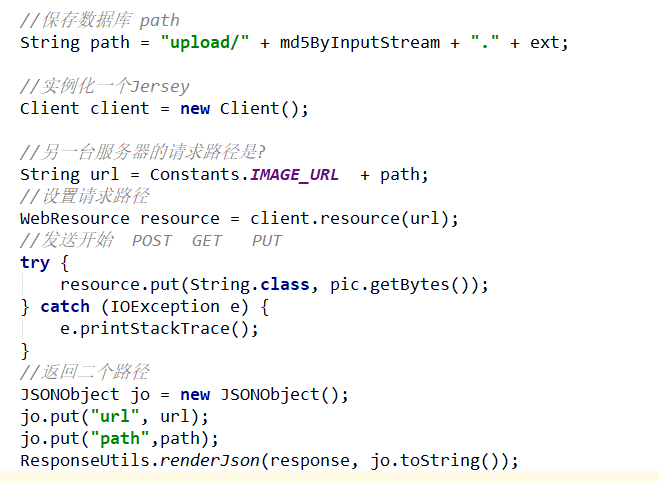
### 配置springmvc支持上传图片



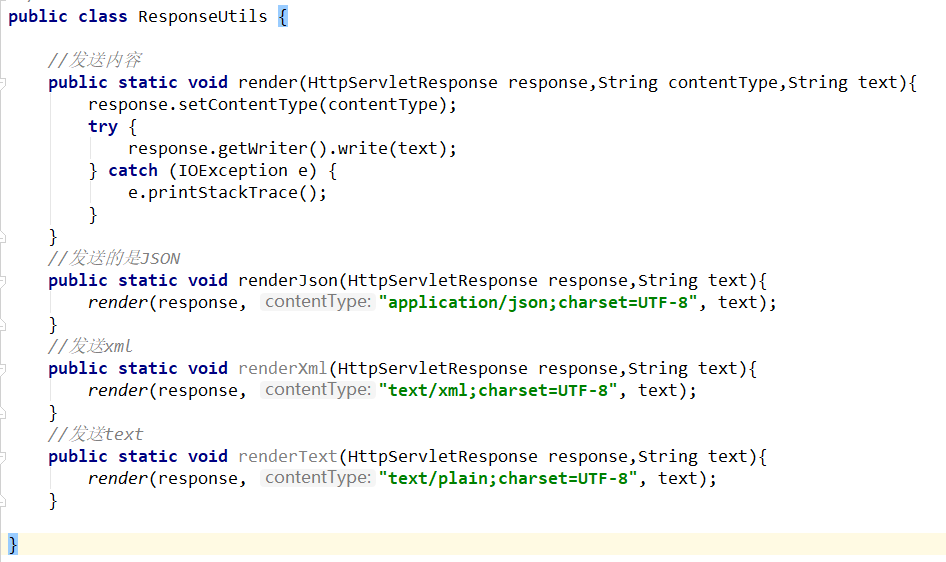
### 后台接收



使用Jersey 上传图片到图片服务器

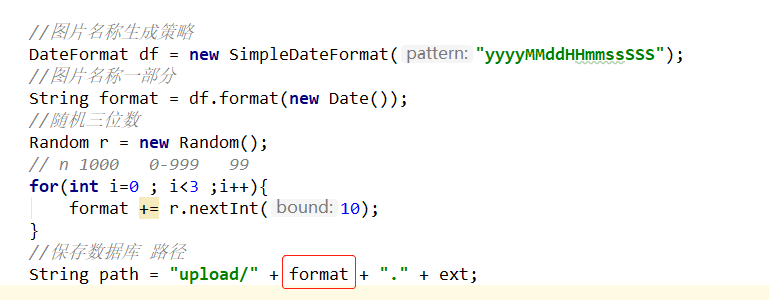


### 异步返回各种格式



### 图片名称生成策略

#### 方式1

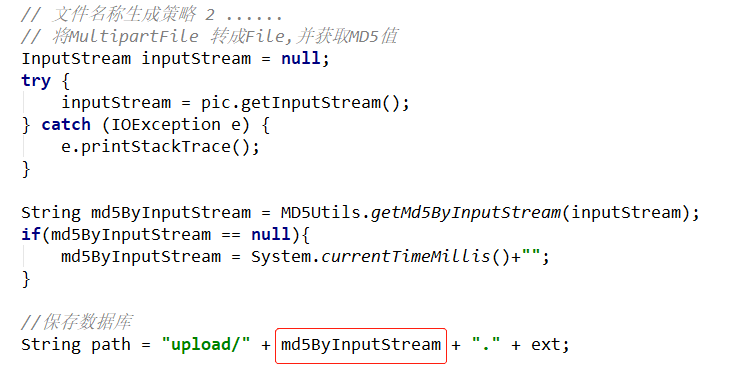


当前时间戳+随机三位数

缺点:

上传同一张图片,会重复上传, 占用系统空间

#### 方式2



获取文件的md5值,所为文件名称

优点:

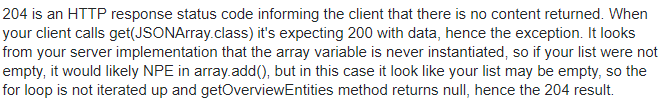
同一份文件的md5值是固定的,所以同一张图片不会重复上传

### jersey上传重复图片遇到的问题

错误信息:

com.sun.jersey.api.client.UniformInterfaceException: PUT http://47.94.108.128:8088/img\_service/upload/6406B05C7D04B47D86455D838EEC3B8E.jpg returned a response status of 204 No Content

报错信息解释:



解决方案 :

捕获异常后即可

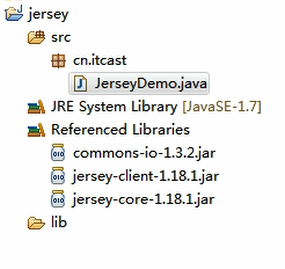
**catch** (UniformInterfaceException e){  
 e.printStackTrace();  
}

图片在远端服务器已经存在,jersey不在重新上传,所以没有返回content, return204状态码, 对业务来说并没有什么问题.

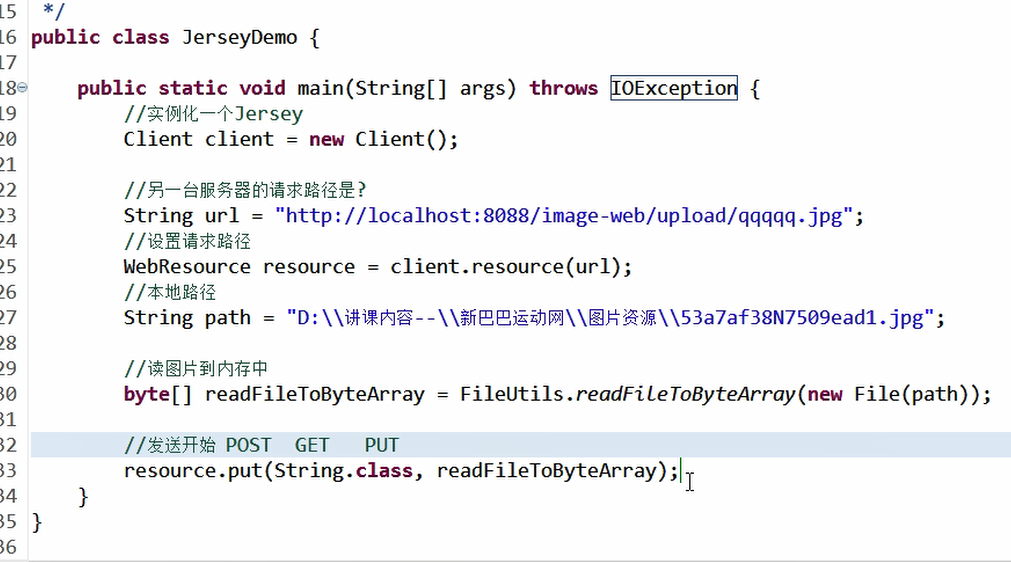
这个时候捕获异常, 让程序继续执行, 也可以不打印异常信息

## 图片服务器

### 项目结构

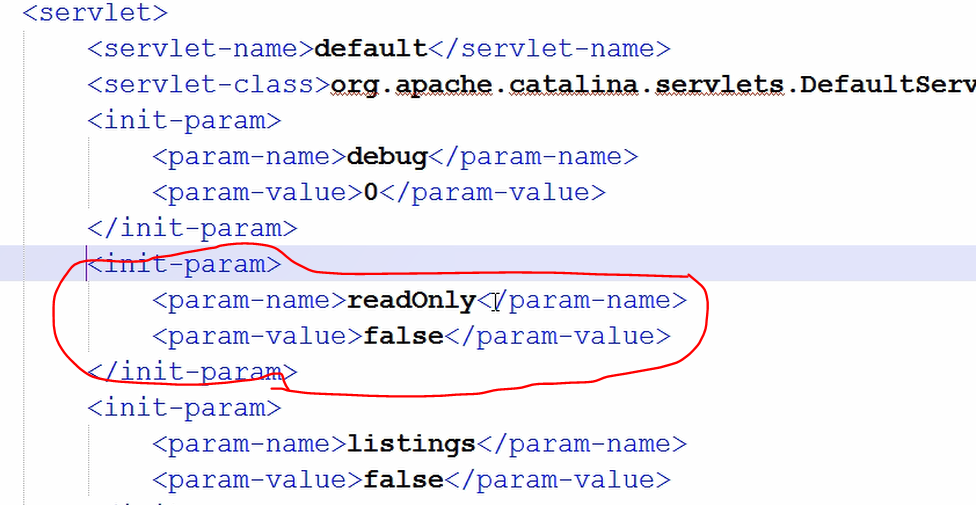


### 代码



### 配置tomcat可读写

将图片上传到图片服务器,需要设置该图片服务器可读写



注意: readonly为全小写, 否则图片无法上传

# day03

# day05

## 四种常见缓存技术

### 一级缓存

sqlSession级别的缓存, 作用范围是局部的, 发生在dao中

### 二级缓存

sqlSessionFactory级别的缓存, 作用范围是全项目的, 是全局中

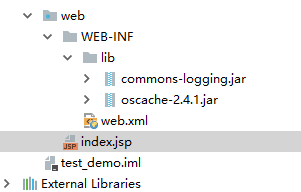
### 分布式缓存

作用范围是多个项目之间

### 页面缓存

缓存页面, 不频繁访问数据库,所以性能最好

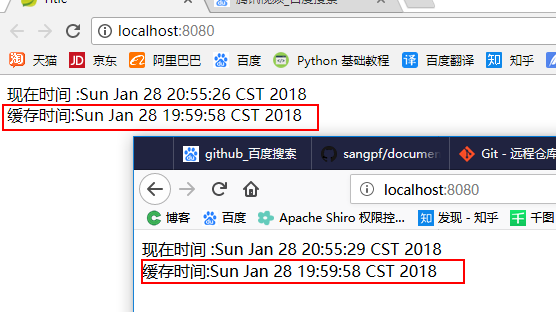
#### 创建web项目，导入jar包



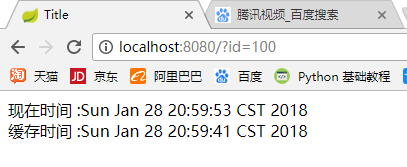
#### 引入标签库，使用标签缓存



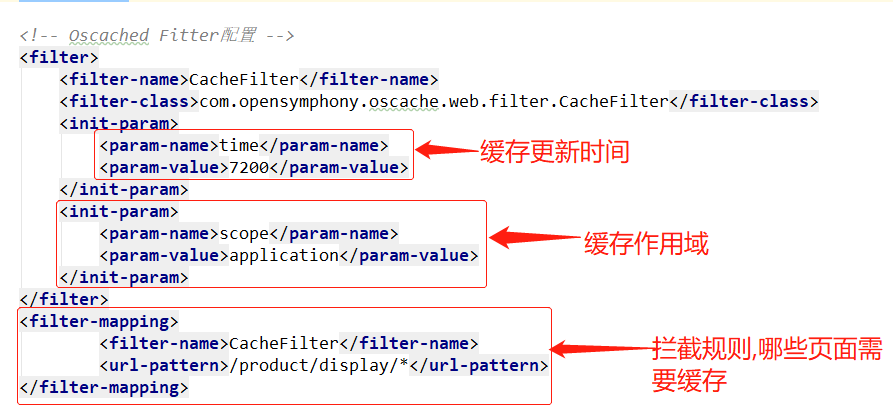
#### 效果测试



修改url地址，缓存时间更新。因为缓存的是url地址，只要url地址不变，更改用户后访问到的缓存页面是相同的



### 将页面缓存加入到项目中



经过以上配置,已经把缓存加入到项目中了

## 修改库存



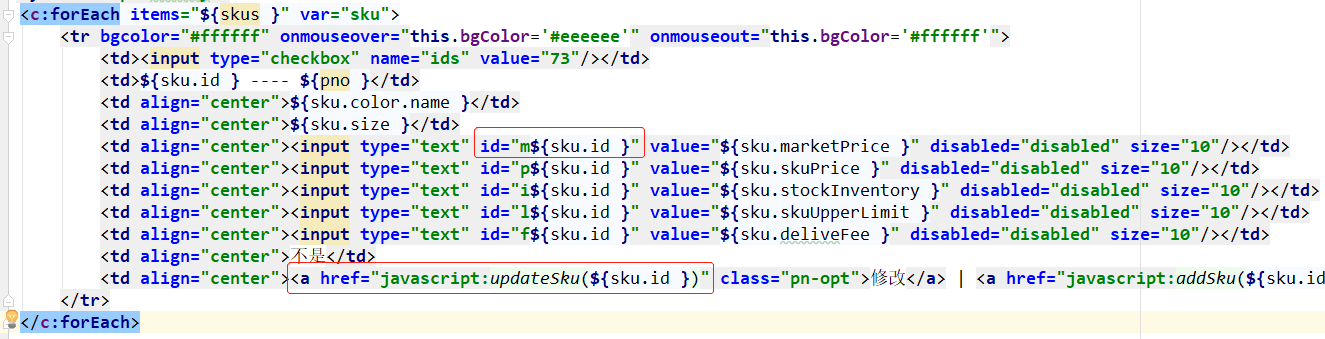
在商品列表中点击库存修改,

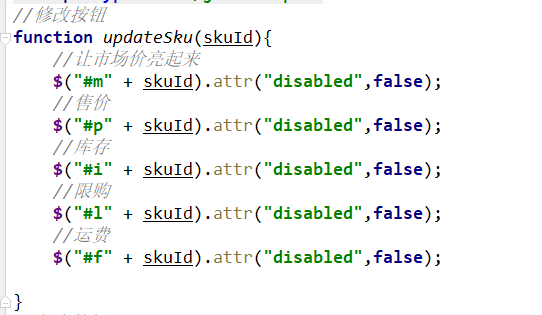


将商品下对应所有的最小销售单元 sku列表加载展示到页面上



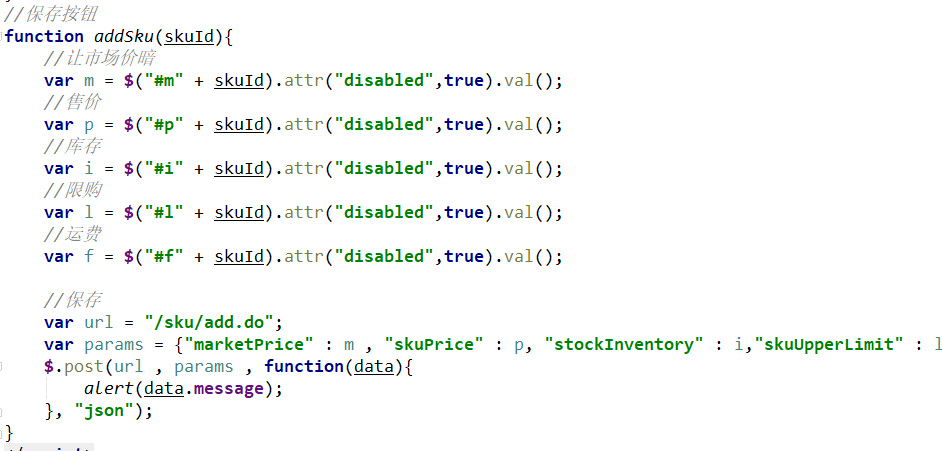
当点击修改按钮后, 输入框中字段变成可编辑



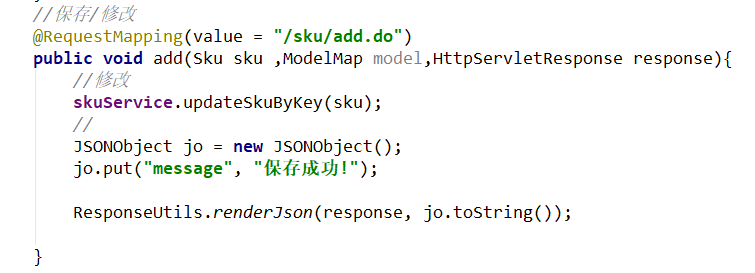


至此,点击修改按钮已经可以,然后就是保存





后端接收



## 商品详情页

### <a>标签介绍



Href=”javascript:void(0)” 代表不使用href标签



Target=”\_blank” 为href标签要跳转到的页面, blank为跳转到空页面, 如果不使用href, 则target也要删除

onclick="*window*.open('/html/product/${entry.id}.html')" 打开新的标签页面

### 商品图片放大镜

<a id="showImg" class="cloud-zoom" href="${product.img.allUrl }">

<img alt="" src="${product.img.allUrl }">

</a>

<img>标签中为显示的小图片

<a>标签中为放大镜图片

### 商品详情加载



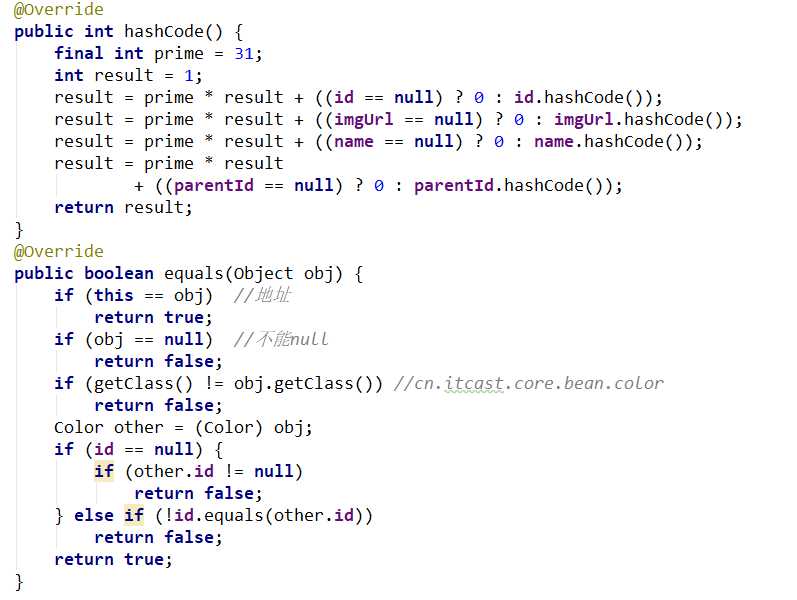
后端接口



需要加载数据, 商品对象, 最小销售单元sku集合, 颜色color集合

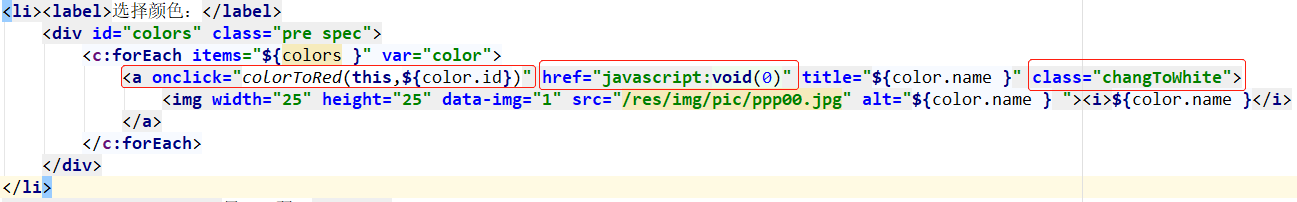
一个商品对应多个最小销售单元, 在加载一个商品的详情时,需要加载出该商品所有的最小销售单元

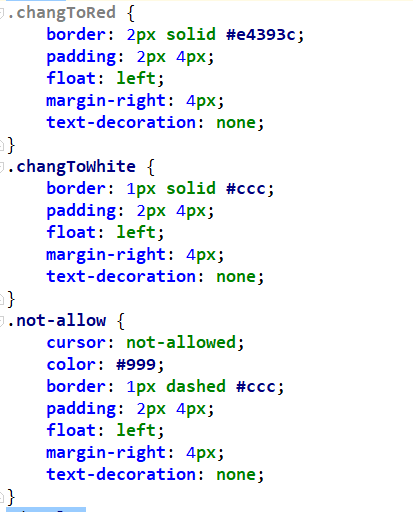
注意: 在使用list的contains方法时,需要对color对象复写hashcode和equals方法

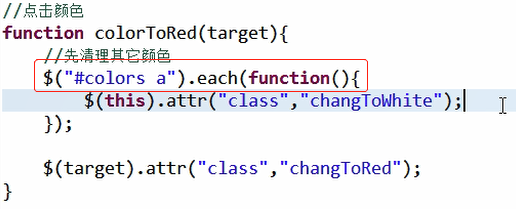


### 实现点击变红









取 id 为 colors下的所有的<a>标签,然后遍历

### 选择颜色,显示有库存的尺码



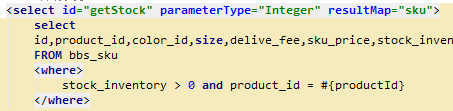
例如,点击西瓜红, 只显示有库存的尺码

思路1: 库存在sku中存储,当点击颜色,可以知道colorId ,通过colorId可查询sku, 进而判断库存, 如果每次点击颜色通过ajax请求到后台,则需要查询数据库, 这样浪费系统的性能

思路2: 如果能在加载商品详情的时候就查询出有库存的sku集合,点击颜色的时候判断sku中的库存,则可以节省性能

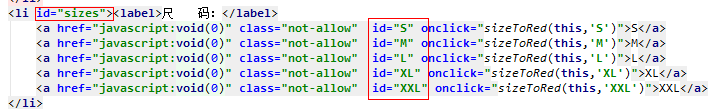
加载商品详情时,准备加载sku集合数据





页面



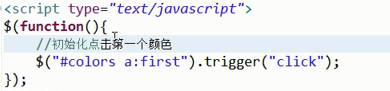


### 当选中一个颜色后,默认选中第一个尺码



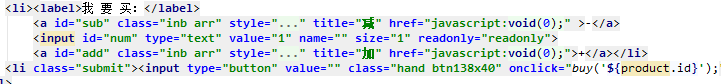
使用flag作为标记

### 在页面刷新的时候,默认选中第一个颜色



### sku联动

### 选择件数，添加到购物车

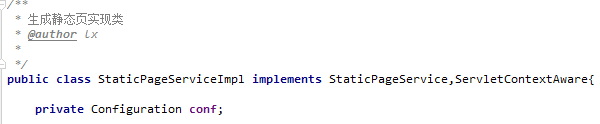




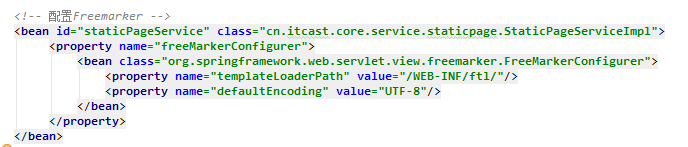
# day06

## 使用freemarker页面静态化

定义实现类

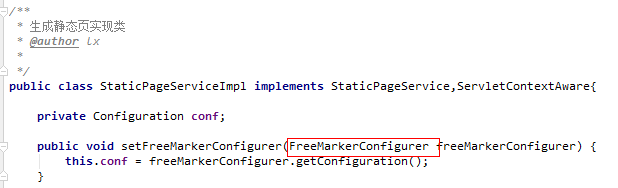


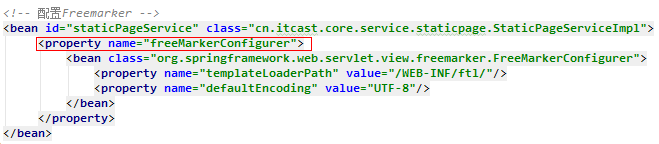
实例化类，可以使用@service , 和另一种方式在<xml>配置文件中配置



两种方式作用相同

### 注入FreeMarkerConfigurer





注入方式

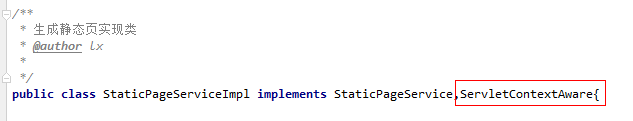
1，注解方式扫描包实例化类中使用@Autowired注入

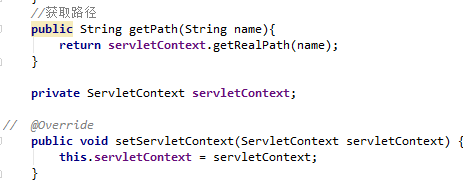
2，配置式使用set方法加上 **property 属性，实现注入**

### 封装输出流



### 动态获取输出流路径





### 上架功能使用freemarker静态化

### 修改模板为html

html页面的编码



## Session供应类

### 定义session接口

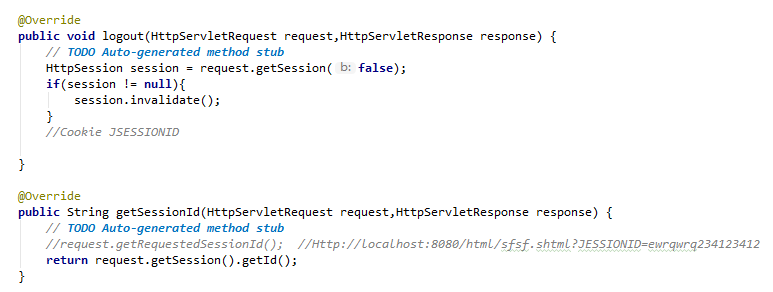


### 本地session实现类



request.getSession(); 默认为 true , 如果有则返回,如果没有则创建

当设置为false, 如果没有则不创建,返回null



退出登录, 获取session

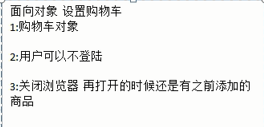
## 验证码



# day07

## 购物车分析

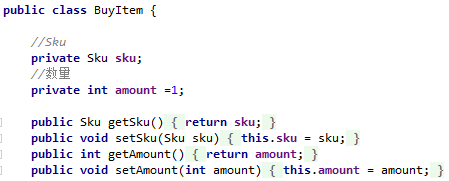




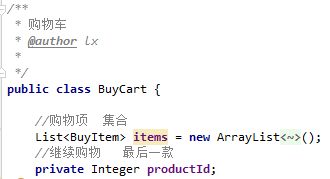
购物车就是购物项的集合, 购物项中包括最小销售单元sku, 和数量

## 实现

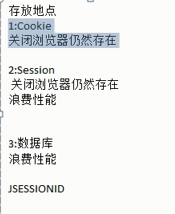
### 购物项



### 购物车



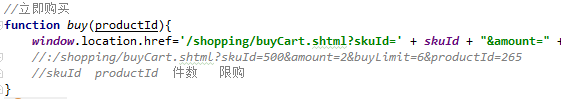
### 购物车对象存放位置



所以最终选择存放到cookie中

## 立即购买

参数准备

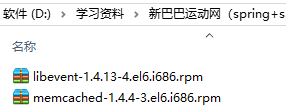




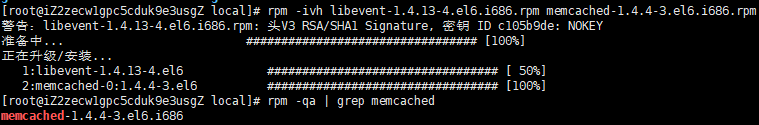
# day10

## 搭建缓存服务器

### 安装包



### 安装

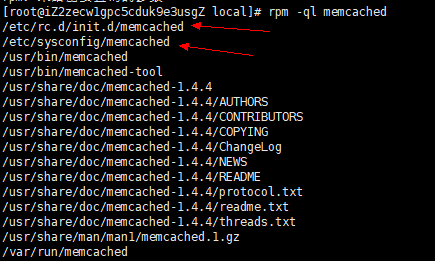


### 查看安装目录

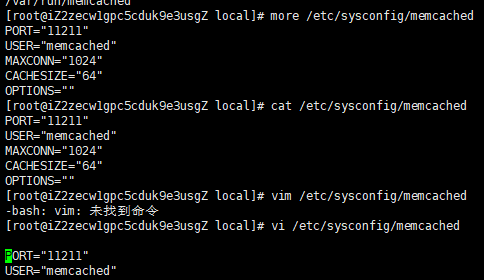
箭头所示为 ：

启动命令， 当有如同所示文件，可以使用 service memcached start 启动服务

配置文件



### 查看配置文件的几种方式



### 启动缓存服务



## 接口调用memcached服务

### 添加memcached依赖

不知道什么原因，在maven的远程仓库没有找到memcached依赖，但是下载到了memcached的jar包，将该jar包安装到自己本地maven仓库，并不复杂

1：下载你要使用版本的memcached。[下载memcached client](https://github.com/gwhalin/Memcached-Java-Client/downloads)

2：将里面的   java\_memcached-release\_2.6.6.jar 这个文件 放在d盘（你看哪边爽 放在哪边）

3：cmd 进入命令；

**[java]** [view plain](http://blog.csdn.net/u010800970/article/details/52162295) [copy](http://blog.csdn.net/u010800970/article/details/52162295)

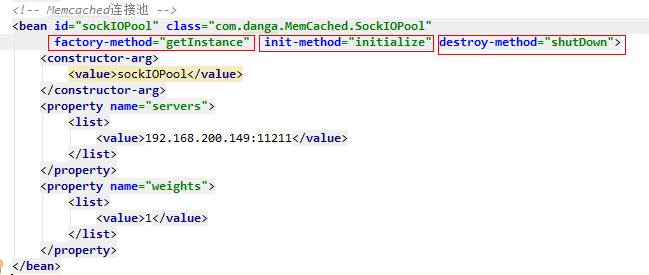
D:\>mvn install:install-file -Dfile=d:/java\_memcached-release\_2.6.6.jar -DgroupId=com.danga -DartifactId=memcached -Dversion=2.5.3 -Dpackaging=jar -DgeneratePom=**true**

然后在你的maven仓库 就看到com包下面多了一个dange文件夹 打开看看  有 ok 增加成功

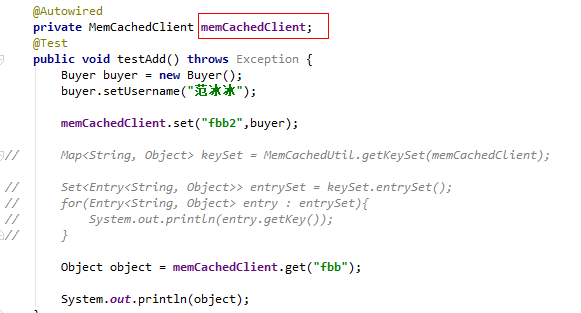
### memcached配置

连接池中配置整个生命周期，实例化，初始化，销毁

配置服务地址，相对应的是该服务的权重



### 测试



### 将memcached放入业务中

配置切面,环绕通知



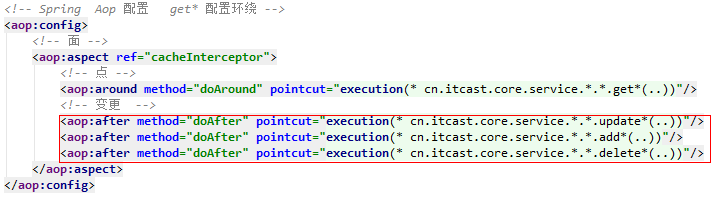
环绕方法



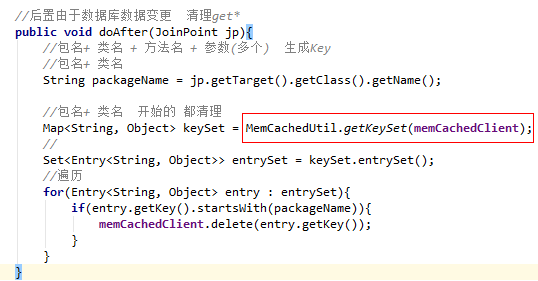
### 数据同步

执行add,update,delete修改数据库,后如何与缓存同步

配置后置通知



数据同步, 情理缓存中数据, MemCachedUtil为封装好的jar包,没有引入



## 共享session

# maven国内镜像源

|  |
| --- |
| 最近maven工程里需要导入大量jar包，无奈国外自动下载实在太慢，OSCHINA还不能用，所以笔者找到一个较好的阿里云开源镜像，通过把~./m2下的settings.xml里放入如下的内容：  <?xml version="1.0" encoding="UTF-8"?>  <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"           xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">     <mirrors>         <!-- 阿里云仓库 -->         <mirror>             <id>alimaven</id>             <mirrorOf>central</mirrorOf>             <name>aliyun maven</name>             <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>         </mirror>         <!-- 中央仓库1 -->         <mirror>             <id>repo1</id>             <mirrorOf>central</mirrorOf>             <name>Human Readable Name for this Mirror.</name>             <url>http://repo1.maven.org/maven2/</url>         </mirror>              <!-- 中央仓库2 -->         <mirror>             <id>repo2</id>             <mirrorOf>central</mirrorOf>             <name>Human Readable Name for this Mirror.</name>             <url>http://repo2.maven.org/maven2/</url>         </mirror>     </mirrors>   </settings> |