## 工作原理

1:发起请求到前端控制器(DispatcherServlet)

2:前端控制器请求处理器映射器HanderMapping查找Handler

根据xml配置 注解查找

3:处理器映射器HanderMapping向前端控制器返回Handler

4:前端控制器调用处理器适配器执行Handler

5:处理器适配器执行Handler

6:Handler执行完向适配器返回MAV

7:处理器适配器向前端控制器返回MAV

ModelAndView是springmvc一个底层对象 包括model view

8:前端控制器请求视图解析器去进行视图解析

根据逻辑视图名解析成真正的视图(jsp)

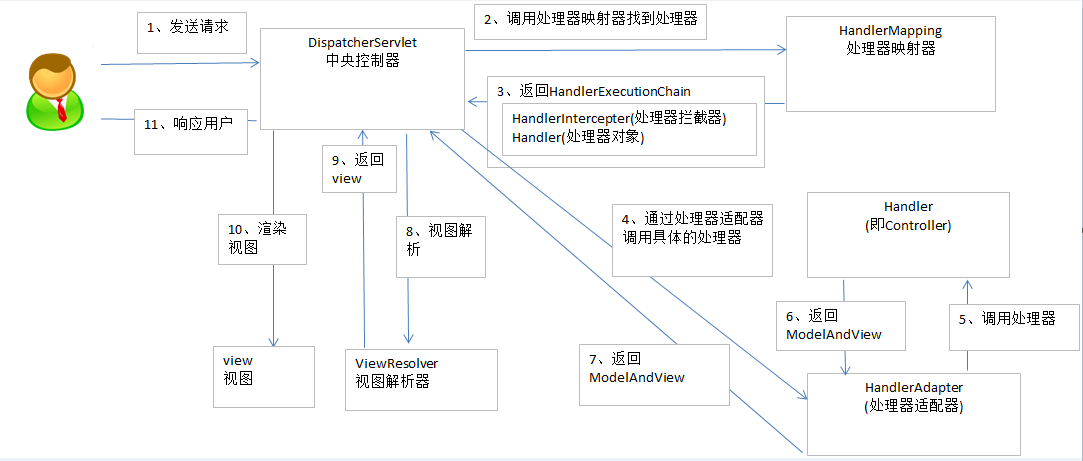
9:视图解析器想前端控制器返回View

10:前端控制器进行视图渲染

试图渲染就是把模型数据填充到request域

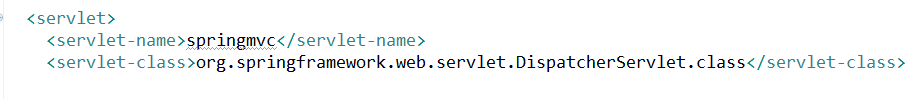
11：前端控制器向用户响应结果



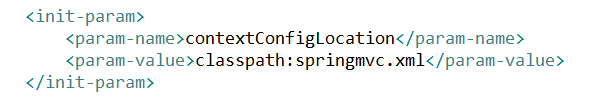


配置前端控制器：

在web.xml配置文件中对前端控制器配置

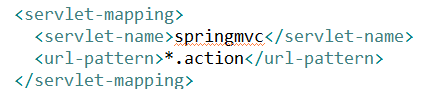


配置springmvc加载的配置文件(配置处理器映射器 适配器) 拥有默认配置





配置servlet-mapping



url三种写法

.action / /\*

第二种所有地址都由DispatchServlet解析 需要配置不解析静态文件

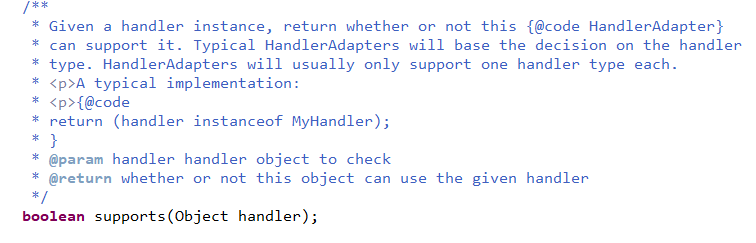
第三种 解析jsp 会报错 无法找到handler

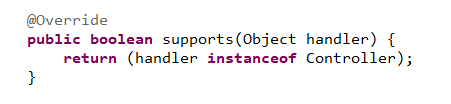
## Springmvc配置文件

需要配置处理器映射器 处理器适配器 视图解析器

处理器适配器

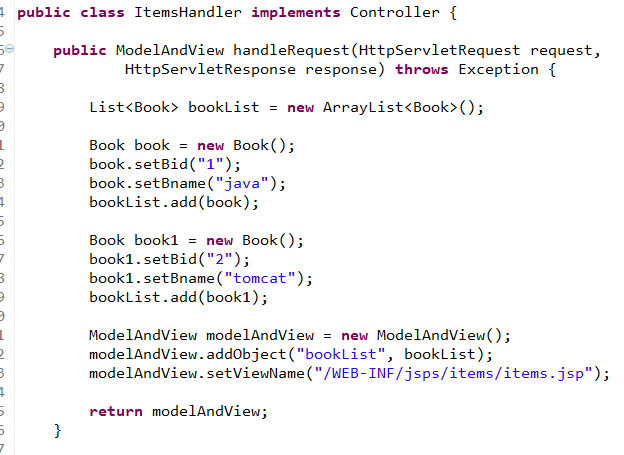




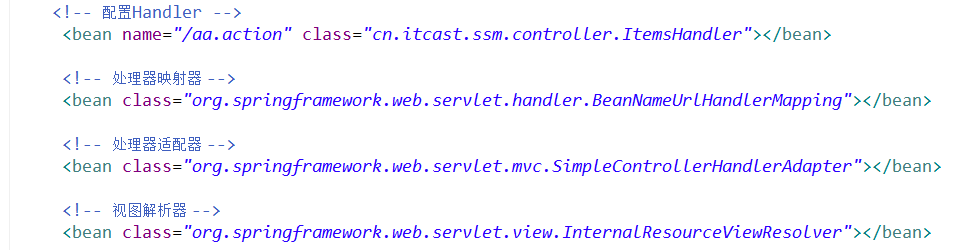


根据HandlerAdapter接口定义的方法此适配器能够执行时间Controller接口的handler

### 编写Handler



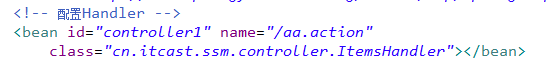
### 控制器映射器编写

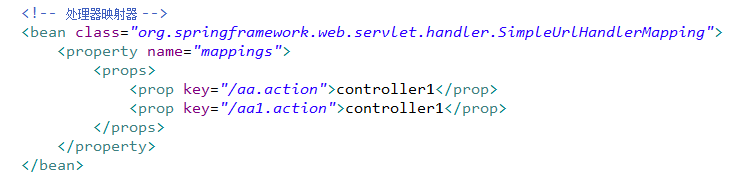


### 非注解的映射器和适配器

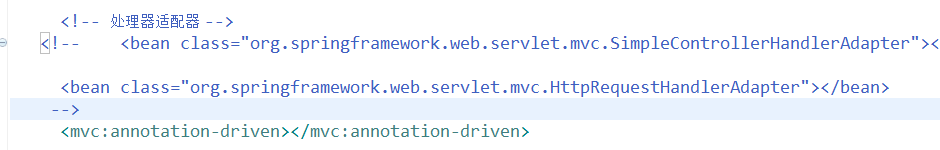
多个映射器可以共存

多个不同适配器需要在编写Handler实现不同的接口

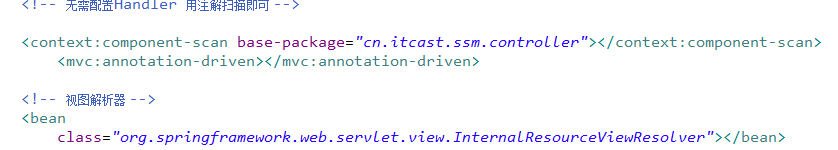


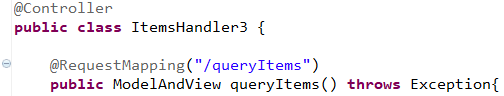


mvc注解驱动



代替适配器和映射器配置 并进行参数绑定

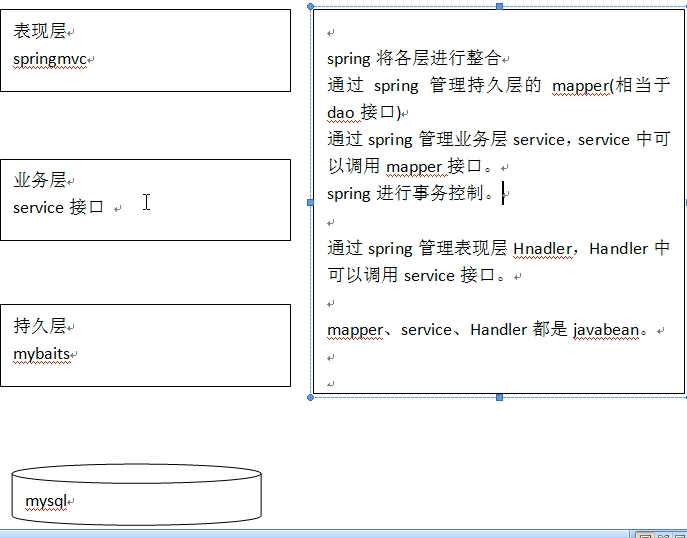




对标记有Controller类中标记有RequestMapping的方法进行映射

### 视图解析器

### Mvc整合mybatis



### 整合详细步骤-dao

#### mybatis主配置文件

1:mapper.xml和mapper.java同名且同一目录

2:配置pojo别名

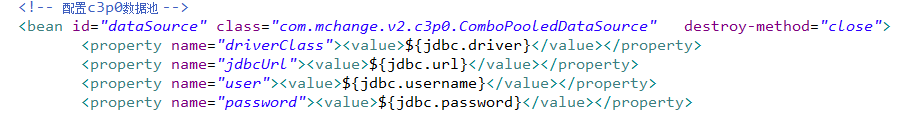


#### Mybatis和spring整合配置文件

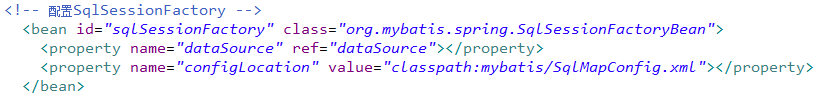


1:配置c3p0数据源

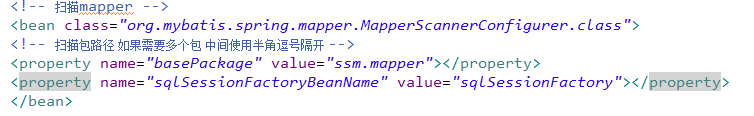




2:配置sqlSessionFactory



3:mapper扫描器(多个包中间使用逗号隔开)



4:逆向工程生成mapper pojo(关联查询需要手动编写mapper)

### 整合service

创建applicationContext.xml-service文件

1：配置service

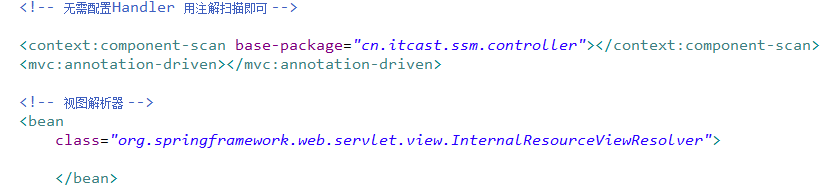
### 事务控制

applicationContext-transaction.xml

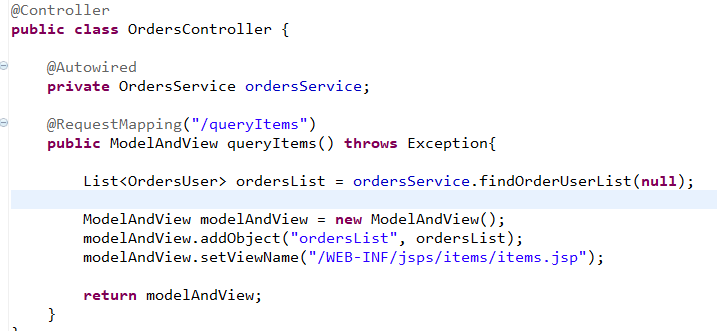


### 整合springmvc

创建springmvc.xml 管理处理器映射器 适配器 视图解析器



编写Controller



### 加载spring容器（加载上文配置的mapper service等）



使用通配符进行加载

在web.xml中添加监听器 加载spring容器

### 实现修改图书功能(开发mapper service controller)

###### mapper开发

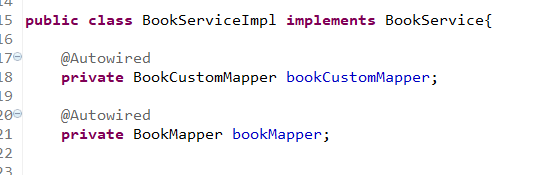
1:根据bid查询图书信息

2:将图书信息更新到orderitem表

功能分析:

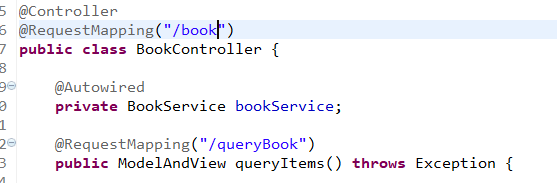
需要book的查询方法和更新方法 逆向工程已经实现

BookService实现类加入逆向工程BookMapper

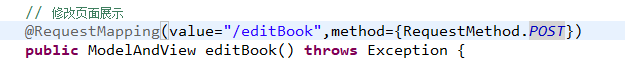


##### RequestMapping注解

1:分类controller的url



2:限制Http请求方法

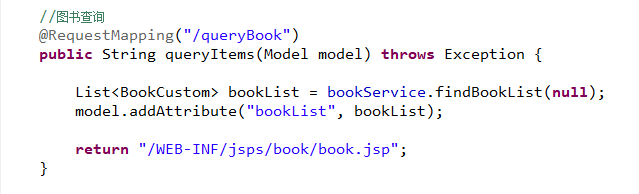


##### Controller内方法的返回值

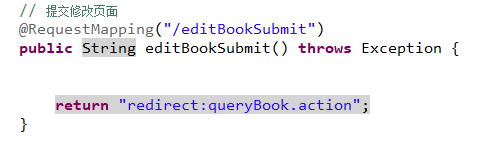
1:ModelAndView 编辑mav并且返回

2:String 返回逻辑视图名

真正视图(jsp路径) = 前缀+逻辑视图名+后缀



完成重定向



完成转发 request存活



3:void 可以定义request设置响应头

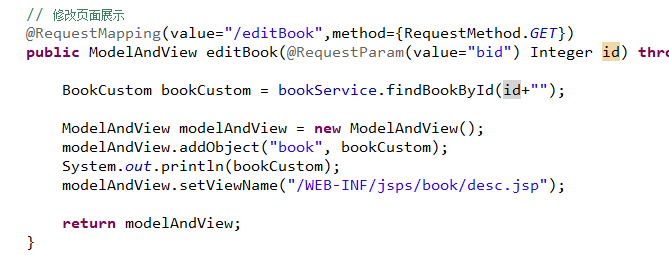
##### 参数绑定

mvc中 接受页面提交的数据是根据自定义方法的形参获取的

**默认支持的类型(HttpServletRequest/response/Session)**

###### 使用@RequestParam注解简单类型绑定

value指定页面传递的参数 修饰变量指定参数的副本



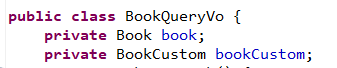
###### POJO绑定

乱码问题：过滤器解决(post) get方式再议

对象属性要求和页面input的name同名

### 绑定包装类型的pojo

图书pojo



Controller



Jsp



保证页面表单的name对应pojo的成员的属性名

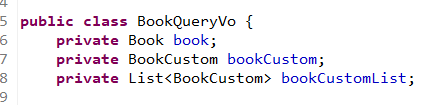
不仅可以做查询 还可以做修改 如果user和book的name同时存在表单 建议使用pojo方式传值

### 绑定数组(批量删除)

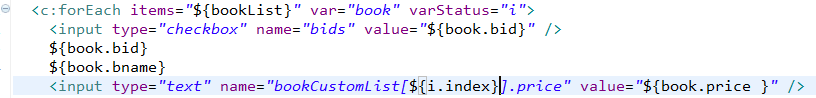
表单name对应参数名即可

### 绑定List

包装pojo内部定义List属性



Jsp



这样可以获取到list属性 **map同理**

### 商品修改信息校验

实际中 经常会在前端校验

服务端：控制层controller校验请求参数合法性

业务层：service层主要校验业务参数 仅限于service接口中使用的参数 实际较多

持久层：dao一般不进行校验

Spring使用validation校验框架

加入jar包

Springmvc.xml文件配置校验器（注入到处理器适配器）

创建报错信息配置文件 编码必须默认



**图书名长度为例**

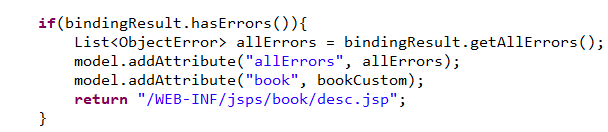
校验的信息对应的属性名增加注释



参数列表前加注释 后加结果集



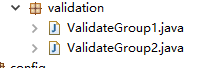
结果集处理即可



### 分组校验

实际开发对于属性校验可能会有不同情况

定义校验接口 无需编写 只用来分辨



编辑属性



Crotroller形参编辑

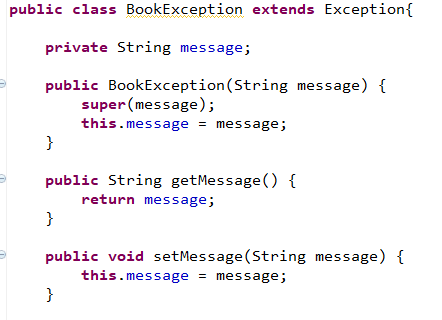


### 异常处理

预期异常和运行时异常

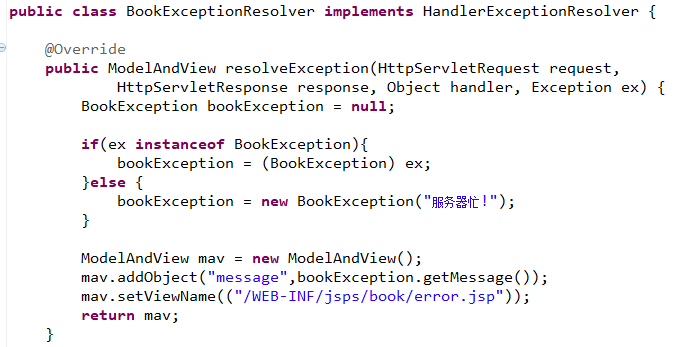
mvc提供统一的异常处理器(一个系统只有一个)

自定义异常类



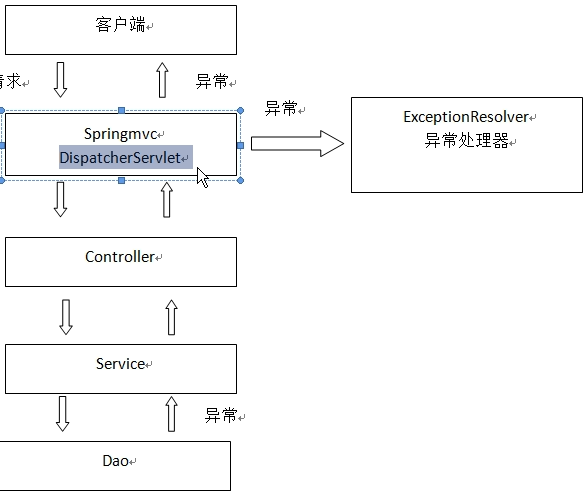
异常处理器解析是否是自定义异常 是取出信息 并显示

如果是运行异常 构造一个异常类 友好处理



mvc.xml中进行配置

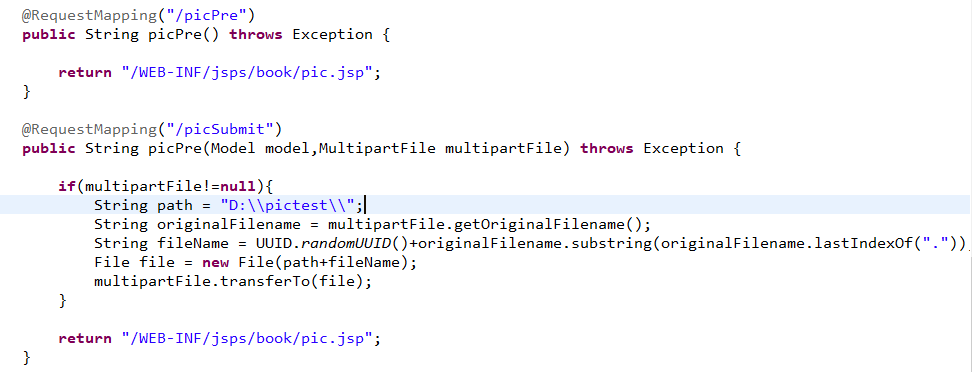




### 图片上传下载

Tomcat设置虚拟目录 编辑server.xml

实际开发 会有专门的图片服务器 图片分层管理 提高io

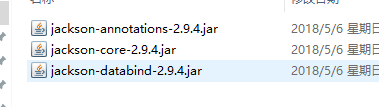


### JSON交互

@RequestBody将json转成java对象处理

@ResponseBody将java对象转成json输出

Jar包

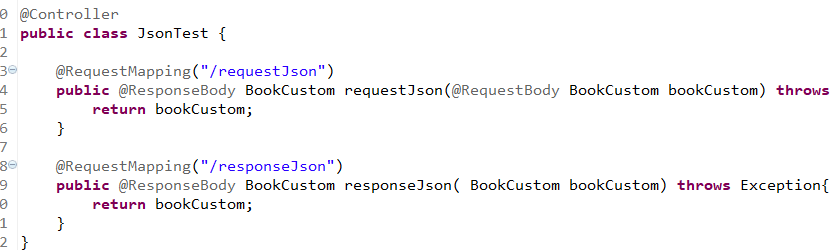


在适配器中配置转换器

实际开发 情况分两种：输入输出都是json 输入时key-value 输出是json

测试使用ajax





### 拦截器

Springmvc拦截器针对HandlerMapping进行拦截 经过Mapping成功才会进行拦截工作

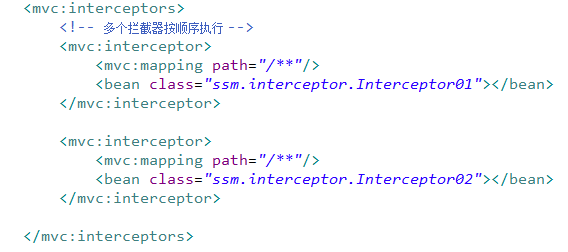
所以拦截器可以对单个HandlerMapping拦截 也可以对多个Mapping进行拦截

Springmvc可以配置类似全局的拦截器 框架会将拦截器注入到每个映射器中



实现HandlerInterceptor接口

Springmvc中配置拦截器



可以用来设计日志处理拦截器功能 放于拦截器中首位

拦截器接口

