**SpringMVC框架技术**

**安阳师范学院 软件学院**

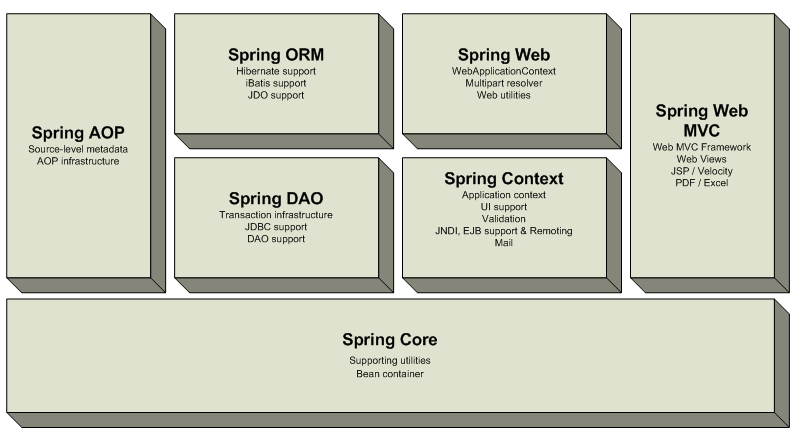
主讲：郭磊

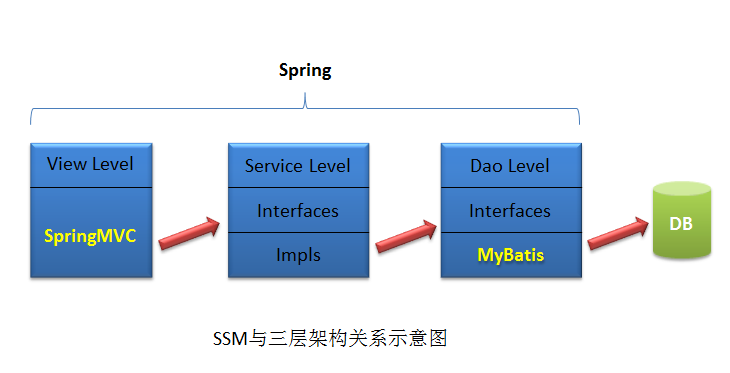
2017年5月

# SpringMVC概述

## SpringMVC简介

SpringMVC，也称为Spring Web MVC，是Spring框架的一部分，其是从Spring3.0开始增加的新的功能。SpirngMVC是一个视图层框架。



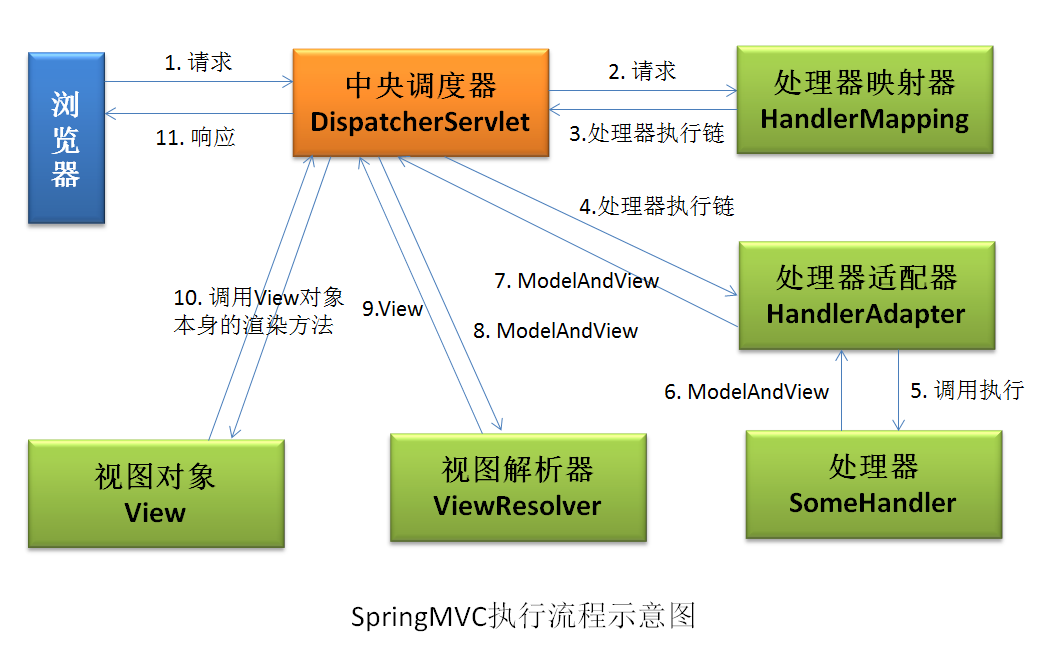


## 第一个SpringMVC程序

功能需求：浏览器提交一个请求，服务端接收到这个请求后，跳转到一个新的页面，并显示欢迎信息。

## SpringMVC执行流程

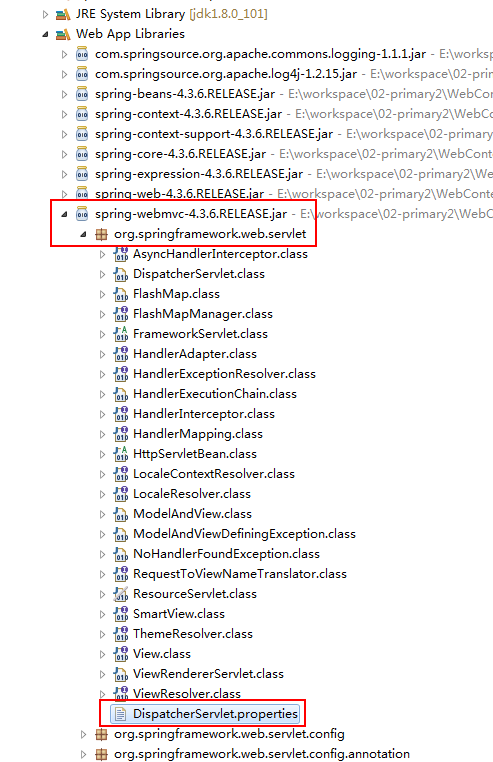
### 执行流程简图



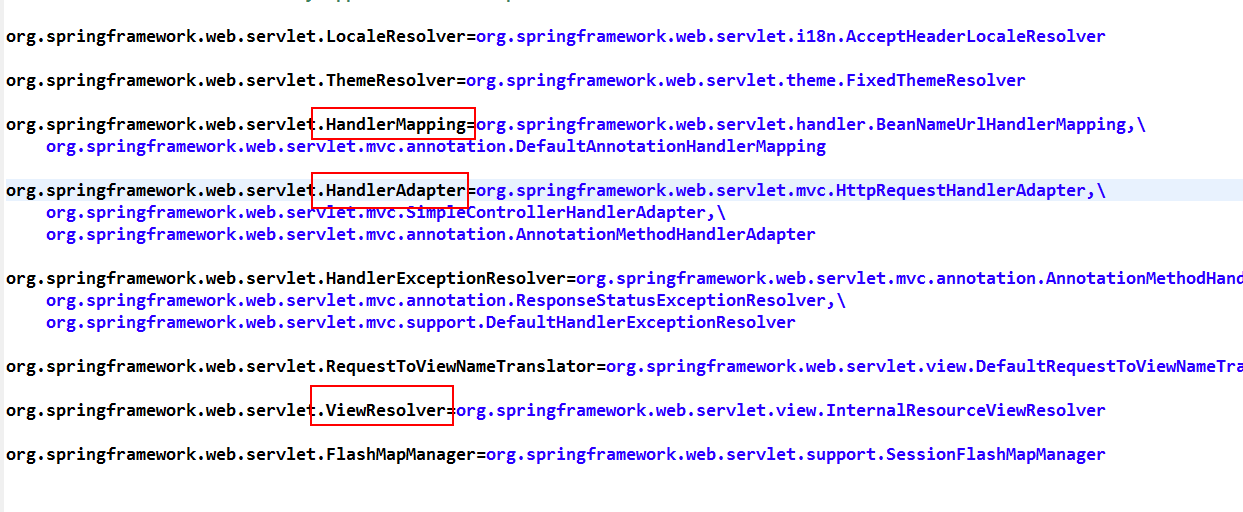
### API简介

### 中央调度器默认属性文件

位置：



内容：



## UrlPattern取值

## 请求路径问题

### 请求路径的构成

客户端提交请求路径的目的是为访问服务端的指定资源，所以请求路径由两部分构成：资源路径与资源名称。即

请求路径 = 资源路径 + 资源名称

资源路径与资源名称的分水岭为：请求路径中的最后一个斜杠。斜杠前的部分称为资源路径，斜杠后的部分称为资源名称。例如：

* 请求路径： <http://localhost:8888/01-primary/some.do>
* 资源路径： http://localhost:8888/01-primary
* 资源名称： some.do

### 请求路径的分类

根据是否可以唯一的定位一个资源标准，可以将路径划分为两类：绝对路径与相对路径。

* 绝对路径：可以唯一的定位一个资源。在Web应用中，一般情况下，以请求协议开头的路径为绝对路径。绝对路径以URL形式表示。
* 相对路径：仅仅依靠此路径，无法唯一的定位一个资源。但若将其再与一个参照路径相结合后，就可以转换为一个可以定位资源的绝对路径，这样的路径，称为相对路径。在Web应用中，一般情况下，不以请求协议开头的路径为相对路径。相对路径以URI形式表示。
* 转换关系： 绝对路径 = 参照路径 + 相对路径

### 相对路径的分类

以相对路径是否以斜杠开头划分，可以将相对路径划分为两类：以斜杠开头的相对路径，与不以斜杠开头的相对路径。

对于以斜杠开头的相对路径，再根据路径出现的位置的不同，可以划分为两类：前台路径与后台路径。

* 前台路径：出现在HTML文件，及JSP文件的静态部分中的以斜杠开头的相对路径，称为前台路径。例如，出现在<img/>的src中的路径、<a/>的href中的路径、<form/>的action中的路径、<script/>的src。。。前台路径的解析者是浏览器。
* 后台路径：出现在Java代码、JSP文件的动态部分、及XML、Properties等配置文件中的以斜杠开头的相对路径，称为后台路径。后台路径的解析者是服务器。

注意，不以斜杠开头的相对路径，其路径解析者，根据路径出现的位置不同，分别可以是浏览器或服务器。

### 转换规则

不同的路径解析者，对同一个相对路径的解析结果是不同的。当然，所谓解析结果，指的是将相对路径转换为的最终的绝对路径。由于绝对路径 = 参照路径 + 相对路径，所以这里的解析规则，就是指为这个相对路径匹配上一个怎样的对照路径。或者说，我们要学习的重点是，浏览器、服务器对相对路径所匹配的参照路径是谁。

* 前台路径：其参照路径为当前Web服务器的根。
* 后台路径：其参照路径为当前Web应用的根。
* 不以斜杠开头的相对路径：其参照路径为当前请求路径的资源路径。

例如，

请求路径： http://localhost:8888/01-primary/xxx/ooo/jjj/some.do

* 当前Web服务器的根： http://localhost:8888
* 当前Web应用的根： http://localhost:8888/01-primary
* 资源路径： http://localhost:8888/01-primary/xxx/ooo/jjj

### 绝对路径分类

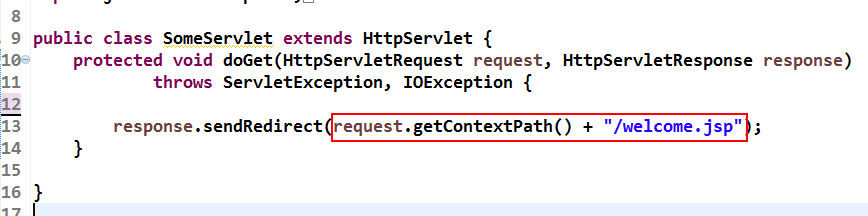
根据路径的作用的不同，可以分为两类：资源定义路径，与资源请求路径。

资源定义路径：用于表示资源在服务端的位置。换句话说，就是客户端若要访问服务端的某个资源，必须要提交的绝对路径。

资源请求路径：客户端所发出的对指定资源的请求路径。

### 特例

Response的sendRedirect()方法中的以斜杠开头的相对路径，按照前面的定义，其为后台路径。后台路径的参照路径应为当前WEB应用的根，但实际却不是，而是当前WEB服务器的根。



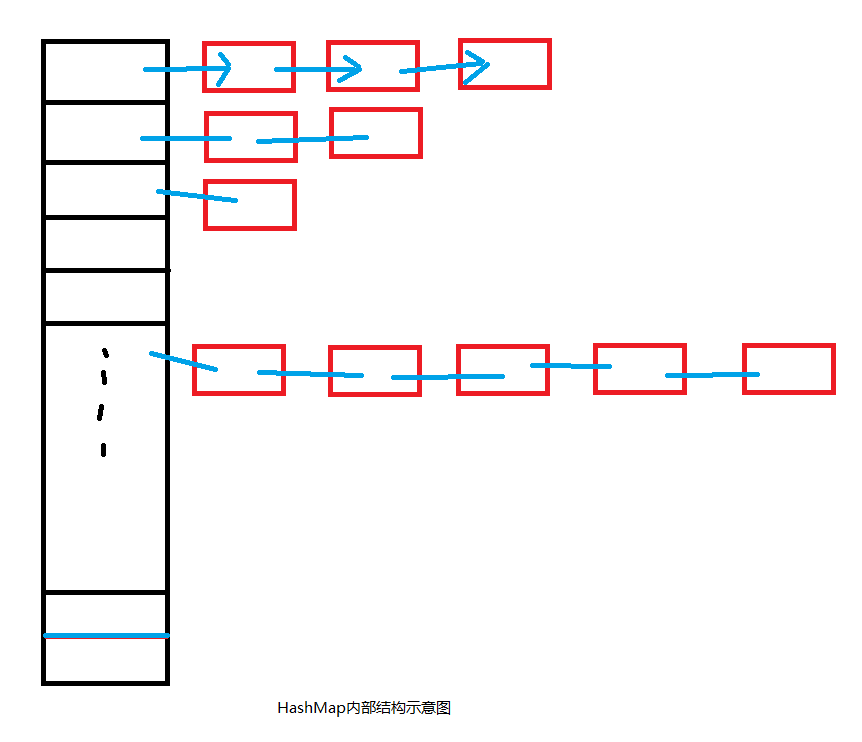
# SpringMVC配置式开发

## 处理器映射器

## 处理器适配器

## 处理器

## ModelAndView



## 视图解析器

# SpringMVC注解式开发

## 第一个注解式开发程序

Step1：导入aop的Jar包

Step2：处理器类不再需要实现Controller接口（或继承相关类），而是需在在处理器类上添加@Controller注解

Step3：在处理器方法上添加@RequestMapping注解

Step4：在SpringMVC配置文件中引入context约束

Step5：在SpringMVC配置文件中注册组件扫描器

## 请求映射规则的定义

### 多个URI对应同一个处理器方法

### 命名空间的定义

### 通配符的应用

### 请求方式的定义

### 请求参数的定义

## 处理器方法的参数

处理器方法中的参数类型很多，这些参数值都是由系统自动赋值,程序员只需要在处理器方法中直接使用即可。常用的参数类型有六种：

* HttpServletRequest
* HttpServletResponse
* HttpSession
* Model
* Map<String, Object>
* 请求中所携带的参数

## 处理器方法的返回值

### ModelAndView

若处理器对请求处理过后，不仅要进行跳转，而且在跳转过程中，还要传递数据，此时使用ModelAndView较为方便。

### String

#### 返回物理视图名

#### 返回逻辑视图名

#### 返回逻辑视图名（BeanName）

### void

#### 需要跳转

#### 不需跳转

##### JSON

JSON，JavaScript Object Notaion，JS 对象标记。JSON对象使用{ }包裹，内部数据使用键值对的形式表示，键值对以冒号分隔。一个JSON对象中可以包含多个键值对，这多个键值对间使用逗号分隔。例如，{ name:”张三”, age:23}，为了增加可读性，一般分行来写：

{

name:”张三”,

age:23

}

也可以定义JSON数组，即以JSON对象为元素的数组。数组使用[ ] 包裹数组元素，使用逗号分隔数组元素。例如：[ {.., ..}, {.., ..}, {.., ..} ]

也可以先定义一个空的数组，逐个向JSON数组中添加元素。向JSON数组中添加元素的方法使用push()。

##### HttpMessageConverter

Http消息转换器，是适配器接口，这个适配器接口的实现类，即各种适配器对象是在MVC注解驱动被加载时，由系统自动创建，其会创建七个适配器。其中就包含JSON与Java对象间转换的适配器。但是，这个适配器类，并不是SpringMVC框架自带的类型，而是由第三方提供。

JSON与Java对象间转换的工具类很多，例如，JSON-LIB，GSON，JACKSON等。我们这里使用JACKSON。所以需要导入JACKSON的Jar包。

##### 编写步骤

Step1：在项目的WebContent中导入jQuery的库文件

Step2：在index页面中导入jQuery的库文件

<script type="text/javascript" src="js/jquery-1.8.3.js"></script>

Step3：导入JACKSON 的Jar包

Step4：在SpringMVC配置文件中注册MVC注解驱动

Step5：在用于接收AJAX请求所携带参数的处理器方法参数前添加@RequestBody

### Object

返回Object时需要注意的问题：

* 处理器方法返回的Object对象，是作为数据出现的，而不是作为视图出现的。
* 返回Object数据的应用场景是，服务端向AJAX请求作为响应。
* 将Object数据传递给页面，需要HttpMessageConverter将其转换为JSON，而这个转换适配器类对象是由JACKSON充当。所以需要导入JACKSON的Jar包；需要注册MVC注解驱动。
* 转换为JSON的处理器方法返回对象，是存放在响应体中的，所以需要处理器告知系统，该返回值要存放到响应体中。怎样告知？在处理器方法前添加@ResponseBody注解。

#### 数值型对象

#### String对象

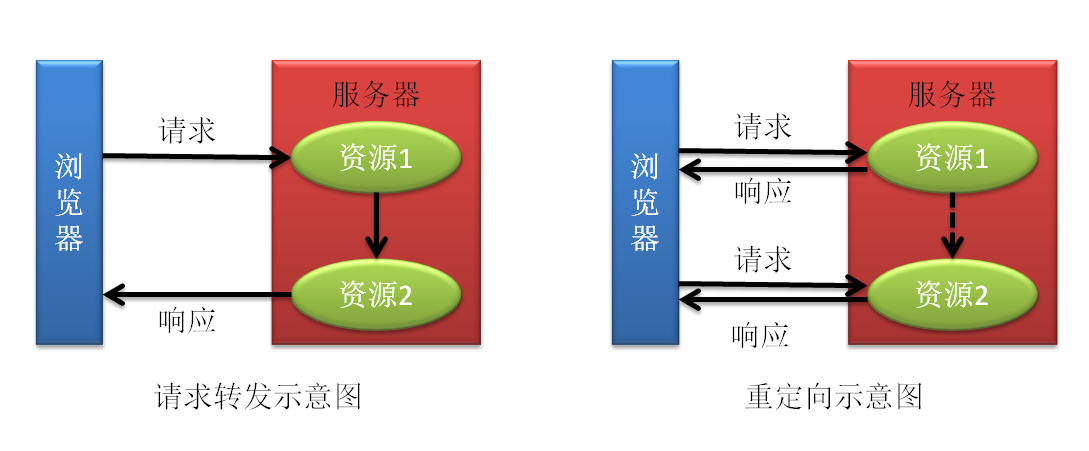
#### 自定义类型对象

#### Map

#### List

# SpringMVC核心技术

## 请求转发与重定向



请求转发与重定向的区别：

请求转发：

1.浏览器发出一次请求（用户提交的请求），获取到一次响应

2.浏览器地址栏未发生变化，仍然是第一次发出的请求

3.也称为服务器内跳转

4.资源2中可以获取到用户提交请求中的数据

5.可以将请求转发到WEB-INF目录中的资源

6.只能将请求转到当前WEB应用的内部资源

重定向：

1.浏览器发出二次请求（一次为用户请求，一次为浏览器自动提交请求），获取到二次响应

2.浏览器地址栏发生改变，变为了第二次请求的地址

3.也称为服务器外跳转

4.资源2中是不能获取到用户提交请求中的数据的，但可以获取到第二次由浏览器自动发出的请求中所携带的数据

5.不能将请求重定向到WEB-INF目录中的资源

6.可以将请求重定向到当前WEB应用的外部资源

根据处理器方法返回值类型的不同，其具体实现请求转发与重定向的方式不同。但总体差不多：需要请求转发，则在视图名称前添加“forward:”，不添加默认也是请求转发；需要重定向，则在视图名称前添加“redirect:”。但需要注意一点，一旦添加了forward：或redirect:，则内部资源视图解析器InternalResourceViewResolver的前辍与后辍将不起作用。

### 返回ModelAndView

#### 请求转发到页面

#### 请求转发到处理器

#### 重定向到页面

#### 重定向到处理器

### 返回String

#### 请求转发到页面

#### 请求转发到处理器

#### 重定向到页面

#### 重定向到处理器

### 返回void

#### 请求转发到页面

#### 请求转发到处理器

#### 重定向到页面

#### 重定向到处理器

## 异常处理

### 使用SimpleMappingExceptionResolver

### 使用自定义异常解析器

### 使用@ExceptionHandler

## 类型转换器

## 初始化绑定器

## 数据验证

数据验证是用户输入的数据值进行的业务逻辑有效性验证，而非数据类型的验证。数据类型的验证是否成功，是由类型转换器完成的。

数据验证分为前端验证与服务端验证两种。前端验证一般通过AJAX完成，而服务端验证则是通过代码完成的验证。

SpringMVC支持JSR（Java Specification Requests，Java 规范提案）303规范，Bean Validation提案，对象验证提案。该规范的实现者很多，但较著名的是Hibernate Validator。Hibernate Validator与Hibernate ORM都是Hibernate的产品，是一种并列的关系，这点可以从Hibernate的官网首页看到（hibernate.org）。

需求：表单中姓名、成绩、手机号：

要求：

姓名：非空、长度为6-12个字符

成绩：百分制

手机：非空，且符合手机号格式

## 文件上传

文件上传要求表单满足三个条件:

* 提交方式为：POST
* 添加一个属性enctype，其值为multipart/form-data
* 至少具有一个file表单元素

文件上传一般都要使用apache提供的文件上传工具类，所以我们要导入两个Jar包：

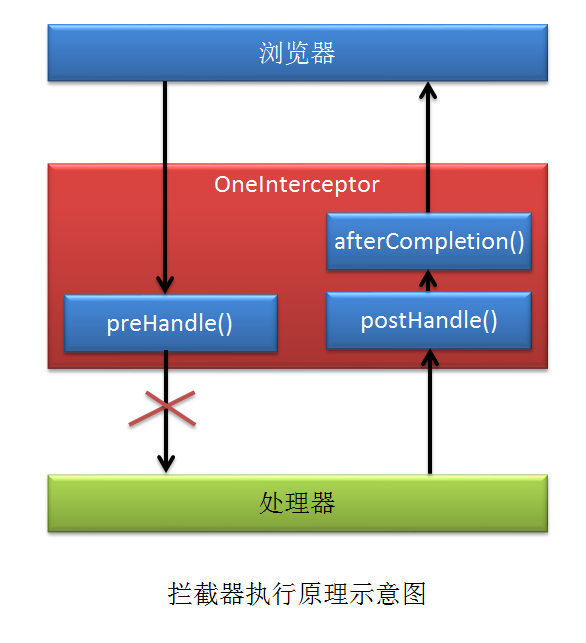
* 文件上传核心Jar包
* 文件上传所依赖的IO Jar包

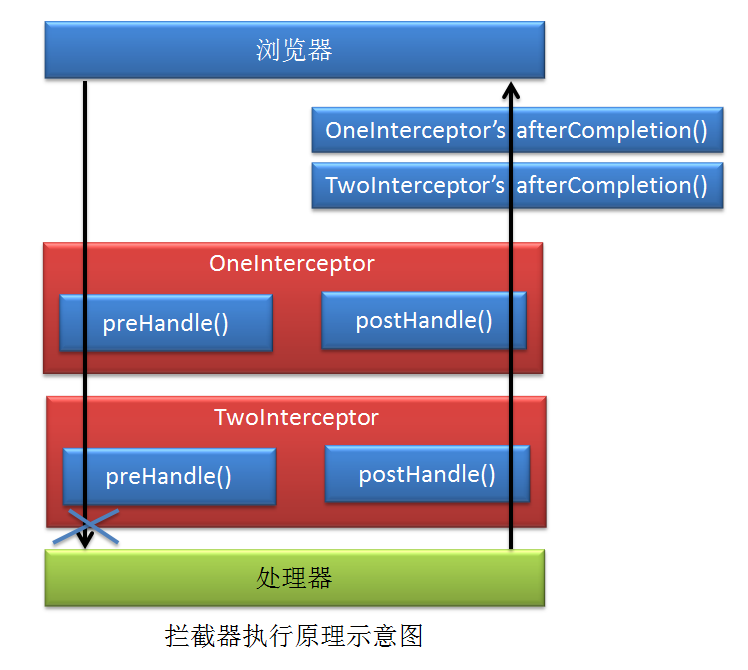
## 拦截器

拦截器，顾名思义，用于拦截请求。其可以完成两件工作：

* 处理器执行之前的预处理操作
* 处理器执行之后的后处理操作

其作用类似于过滤器。不同的是，过滤器是Servlet API，即是任何Web程序都可以使用的API。但拦截器是SpringMVC的API，必须在使用SpringMVC框架时才能使用该API。





## 源码阅读

### “三器”的初始化

### “三器”对象的创建

### 视图对象的渲染

# SSM整合技术

## 搭建SSM整合环境

### Jar包的导入

#### Spring的Jar包

#### AspectJ的核心Jar包

#### AOP联盟的Jar包

#### MyBatis的Jar包

#### MyBatis与Spring整合的Jar包

#### MySql的驱动Jar包

#### C3P0的Jar包

### web.xml的配置

#### 配置内容

##### <context-param/>指定Spring配置文件的位置及名称

##### ServletContext的监听器ContextLoaderListener的注册

##### 注册解决中文乱码问题的Filter

##### 注册SpringMVC的中央调度器

#### 源码分析

##### Spring容器的创建

通过对ContextLoaderListener的源码分析可知，在应用被启动时会自动创建ServletContext对象，而该对象的创建会触发其监听器ContextLoaderListener的执行。监听器的执行会创建Spring容器，并且该容器被存放到了全局域ServletContext中。可以通过WebApplicationContextUtils工具类从全局域获取到Spring容器对象。

##### SpringMVC容器的创建

对DispatcherServlet的源码分析可知，在应用被启动时会创建DispatcherServlet对象，执行其从父类继承来的init()方法，而该方法最终创建了SpringMVC容器。

##### 两个容器的关系

通过对FrameworkServlet类的initWebApplicationContext()方法的分析可知，SpringMVC容器将Spring容器指定了自己的父容器，即将这两个容器进行了合并，最终变为了一个容器。

通过以上源码分析可知，应用中可以不生成Spring容器，但必须要生成SpringMVC容器。即web.xml中可以不进行ContextLoaderListener的配置，但必须要配置DispatcherServlet。

## 全配置式开发

## 全注解式开发