**主要功能：**

**前端数据传给请求处理方法**

**注解使用**

**拦截器**

**Rest风格**

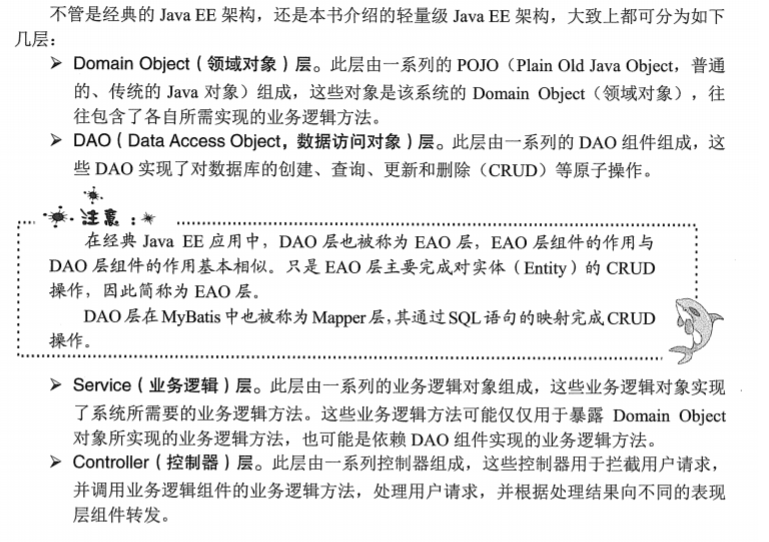
数据校验 //验证客户输入的数据是否合法，比如客户登录时，用户名不能为空

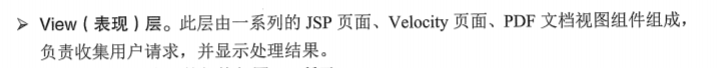
数据转换 //将一种类型转为另外一种类型(如将日期转字符串)

异常处理

上传文件

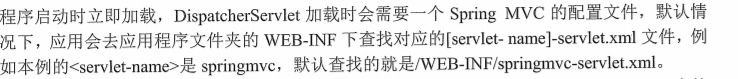
**架构**





**Dispatcher配置方式**









EL表达式

Jstl（jsp标准标签库）

**请求处理方法参数有多种**



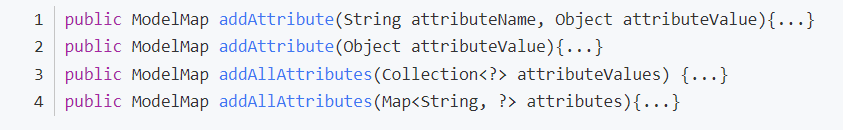
**前端数据传给请求处理方法数据**

包装类（参数对应属性自动set进去），list和map都可以

**三种model区别**

Model和ModelMap用于传递数据

页面上通过el表达式语言$attributeName获取数据



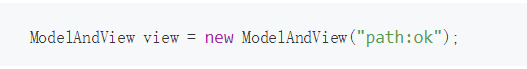
可以通过控制器方法的字符串返回值来设置跳转url地址别名或者物理跳转地址

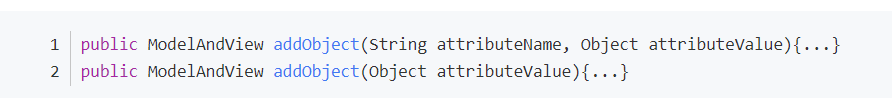


在默认情况下，ModelMap中的属性作用域是request级别，也就是说，当本次请求结束后，ModelMap 中的属性将销毁。如果要在多个请求中共享ModelMap中的属性，必须将其属性转存到session中

ModelAndView用于传递数据和设置转向地址







也可以通过el表达式语言$attributeName获取数据

**字符过滤器**

<filter>

<filter-name>encoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encoding</filter-name>

<servlet-name>/\*</servlet-name>

</filter-mapping>

setEncoding()相当于：ServletRequest.setCharacterEncoding(java.lang.String)

setForceEncoding()的作用是：

强制ServletResponse的编码格式和ServletRequest的编码格式一样。

Request是默认设置为true，response默认是false，要true才设置为utf-8

**过滤器**

* 依赖于servlet容器。在实现上，基于函数回调
* 它可以对几乎所有请求进行过滤，随你的web应用启动而启动的，只初始化一次，以后就可以拦截相关请求，只有当你的web应用停止或重新部署的时候才销毁

在过滤器中修改HttpServletRequest的一些参数

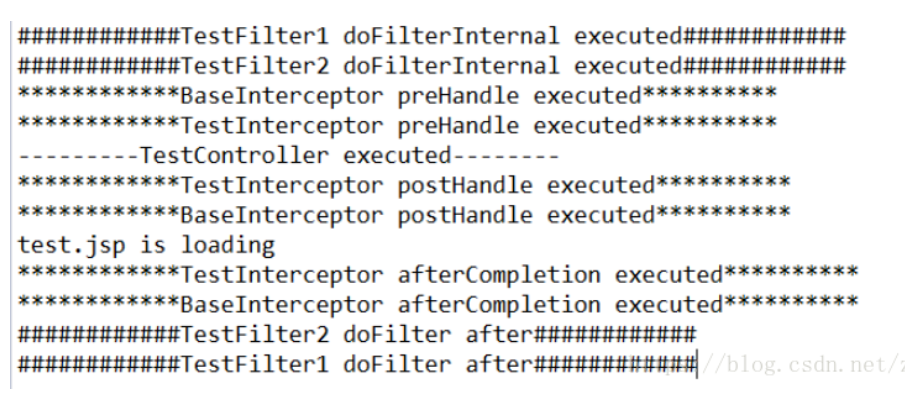
如：过滤低俗文字、危险字符, 修改字符编码（CharacterEncodingFilter）

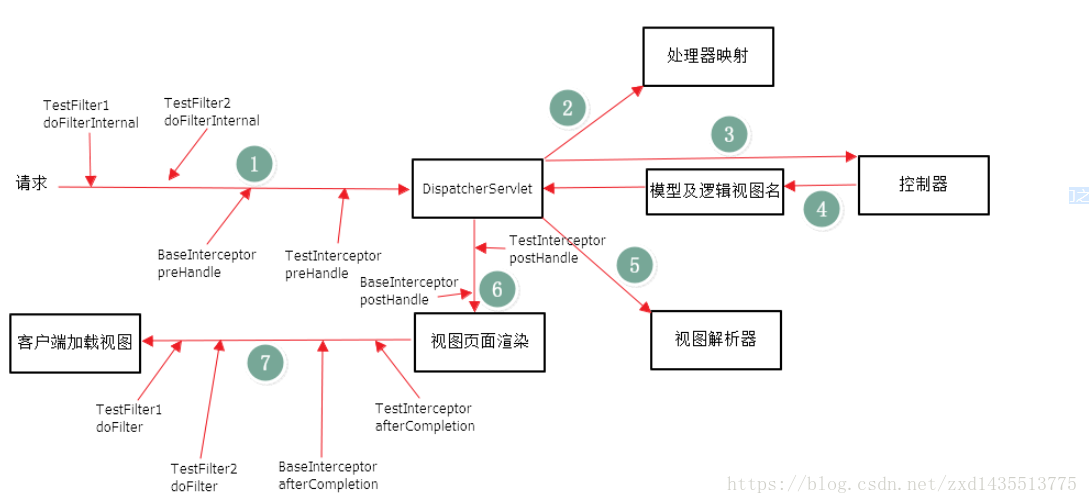
**拦截器**

* 基于Java的反射机制，依赖于web框架
* 一个拦截器实例在一个controller生命周期之内可以多次调用。但是缺点是只能对controller请求进行拦截

就是在service或者一个方法前，调用一个方法，或者在方法后，调用一个方法

*如：检测session登陆状态或访问权限*





拦截器具体实现

* 实现HandlerInterceptor接口
* 继承提供了实现HandlerInterceptor接口的抽象类HandlerInterceptorAdapter

preHandle()： 这个方法在业务处理器处理请求之前被调用

在这个方法中进行一些前置初始化操作或者是对当前请求的一个预处理，进行一些判断来决定请求是否要继续进行下去

postHandle()：这个方法在当前请求进行处理之后，也就是Controller 方法调用之后执行，但是它会在DispatcherServlet 进行视图渲染之前被调用

在这个方法中对Controller 处理之后的ModelAndView 对象进行操作(比如添加数据)

afterCompletion()：该方法将在整个请求结束之后，也就是在DispatcherServlet 渲染了对应的视图之后执行。

这个方法的主要作用是用于进行资源清理工作的

<!-- 对静态资源文件的访问-->

<!-- <mvc:resources mapping="/images/\*\*" location="/images/"/>

<mvc:resources mapping="/css/\*\*" location="/css/" />

<mvc:resources mapping="/js/\*\*" location="/js/" />

<mvc:resources mapping="/favicon.ico" location="favicon.ico" /> -->

/\*\*的意思是所有文件夹及里面的子文件夹

/\*是所有文件夹，不含子文件夹

/是web项目的根目录

<mvc:default-servlet-handler/>

<!-- 不拦截静态文件 -->

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>/js/\*</url-pattern>

<url-pattern>/css/\*</url-pattern>

<url-pattern>/images/\*</url-pattern>

<url-pattern>/fonts/\*</url-pattern>

</servlet-mapping>

**校验：**

前端：脚本语言

后端：控制层（请求参数） 业务层（业务参数即自己方法参数） dao层不需要

**Rest风格：**

特点：不出现动词，全小写,单词之间用\_连接，一个url就是一个资源



* GET 用来获取资源
* POST 用来新建资源（也可以用于更新资源）
* PUT 用来更新资源
* DELETE 用来删除资源