# 拦截器的简介

## 概念

1. Spring里的拦截器是**动态拦截Action调用的对象(部分web请求)**，它提供了一种机制可以使开发者在一个Action执行的前后执行一段代码，也可以在一个Action执行前阻止其执行，同时也提供了一种可以提取Action中可重用部分代码的方式。
2. 在AOP中，拦截器用于在某个方法或者字段被访问之前，进行拦截然后再之前或者之后加入某些操作。
3. 一般应用在，拦截未登录用户和审计日志。

## 实现原理

1. 大部分时候，拦截器方法都是通过代理的方式来调用的，Java的反射机制（动态代理）。
2. Struts2的拦截器实现相对简单。当请求到达Struts2的ServletDispatcher时，Struts2会查找配置文件，并根据配置实例化相对的拦截器对象，然后串成一个列表（List），最后一个一个的调用列表中的拦截器。
3. Struts2的拦截器是可插拔的，**拦截器是AOP的一个实现**。Struts2拦截器栈就是将拦截器按一定的顺序连接成一条链。在访问被拦截的方法或者字段时，Struts2拦截器链中的拦截器就会按照之前定义的顺序进行调用。

## 自定义拦截器步骤

1. 自定义一个实现了Interceptor接口的类。
2. 在配置文件中注册定义的拦截器。
3. 在需要使用Action中引用上述定义的拦截器，为了方便也可以将拦截器定义为默认的拦截器，这样在不加特殊说明的情况下，所有的Action都被这个拦截器拦截。

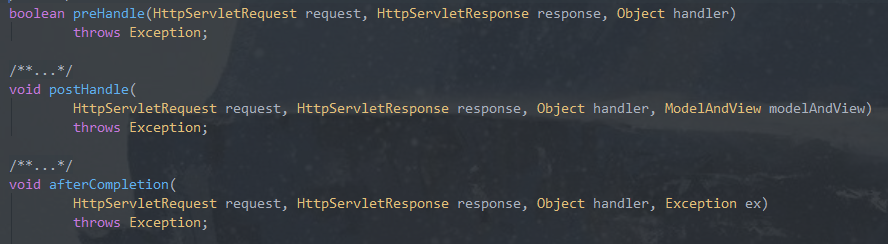
# 拦截器的实现

在SpringMVC 中定义一个Interceptor 非常简单，主要有两种方式:

1. 要定义的Interceptor类要实现了Spring 的**HandlerInterceptor** 接口，或者是这个类继承实现了HandlerInterceptor 接口的类，比如Spring 已经提供的实现了HandlerInterceptor 接口的抽象类HandlerInterceptorAdapter 。
2. 实现Spring的**WebRequestIntercepto**接口，或者是继承实现了WebRequestInterceptor的类。
3. HandlerInterceptor接口位于spring-mvcweb包里，是springMVC特定的接口。

而WebRequestIntercepto接口位于spring-web包里，是通用的web接口，会比H接口包装得更上层一点。

## HandlerInterceptor接口



### preHandle方法

1. preHandle (HttpServletRequest request, HttpServletResponse response, Object handle) 方法，该方法将在请求处理之前进行调用。
2. SpringMVC 中的Interceptor 是链式的调用的，在一个应用中或者说是在一个请求中可以同时存在多个Interceptor。每个Interceptor 的调用会依据它的声明顺序依次执行，而且最先执行的都是Interceptor 中的preHandle方法，所以可以在这个方法中进行**一些前置初始化操作或者是对当前请求的一个预处理**，也可以在这个方法中**进行一些判断来决定请求是否要继续进行下去**。
3. 该方法的返回值是布尔值Boolean类型的，当它返回为false 时，表示请求结束，后续的Interceptor和Controller都不会再执行；当返回值为true时就会继续调用下一个Interceptor的preHandle方法，如果已经是最后一个Interceptor 的时候就会是调用当前请求的Controller方法。

### 1.2 postHandle方法

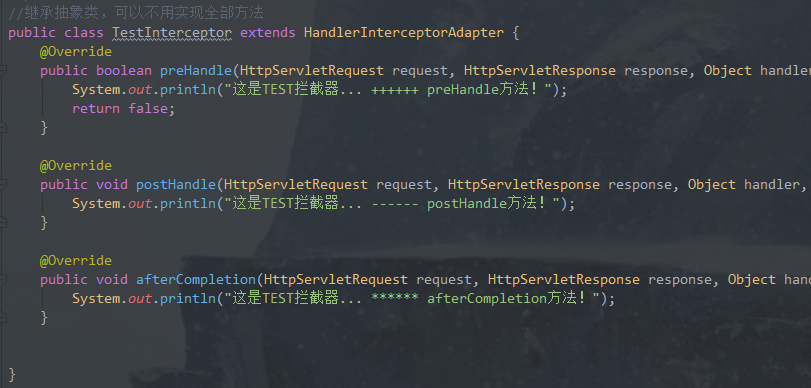
1. postHandle (HttpServletRequest request, HttpServletResponse response, Object handle, ModelAndView modelAndView)方法，**在当前请求进行处理之后，也就是Controller 方法调用之后执行，但是它会在DispatcherServlet 进行视图返回渲染之前被调用**，所以可以在这个方法中对Controller处理之后的ModelAndView对象进行操作。
2. 只能是在当前所属的Interceptor 的preHandle 方法的返回值为true 时才能被调用。
3. postHandle 方法被调用的方向跟preHandle 是相反的，也就是说先声明的Interceptor 的postHandle 方法反而会后执行

### 1.3 afterCompletion方法

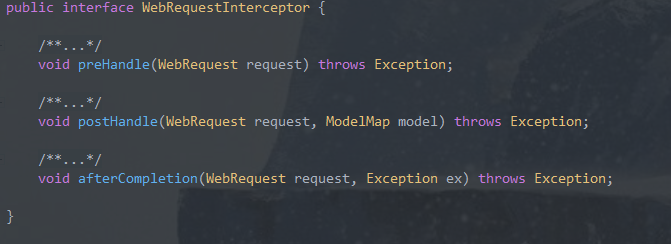
1. afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handle, Exception ex) 方法，该方法也是需要当前对应的Interceptor 的preHandle 方法的返回值为true 时才会执行。
2. 该方法将在整个请求结束之后，也就是在DispatcherServlet 渲染了对应的视图之后执行。这个方法的主要作用是用于进行资源清理工作的。

### 1.4 继承HandlerInterceptorAdapter类

特点：可以不用实现HandlerInterceptor接口的全部方法。



## WebRequestInterceptor 接口



1. 这三个方法都传递了同一个参数WebRequest ，这个WebRequest是Spring定义的一个接口，里面的方法定义都基本跟HttpServletRequest一样，在WebRequestInterceptor 中对WebRequest 进行的所有操作都将同步到HttpServletRequest 中，然后在当前请求中一直传递。

### preHandle(WebRequest request)方法

1. 该方法将在请求处理之前进行调用，也就是说会在Controller方法调用之前被调用。
2. 这个方法跟HandlerInterceptor 中的preHandle 是不同的，主要区别在于该方法的返回值是void ，也就是没有返回值，所以**一般主要用它来进行资源的准备工作**，比如我们在使用Hibernate 的时候可以在这个方法中准备一个Hibernate 的Session 对象，然后利用WebRequest 的setAttribute(name, value, scope) 把它放到WebRequest 的属性中。
3. 这个setAttribute 方法的第三个参数scope ，该参数是一个Integer 类型的。在WebRequest 的父层接口RequestAttributes 中对它定义了三个常量：

**SCOPE\_REQUEST** ：它的值是0 ，代表只有在request 中可以访问。

**SCOPE\_SESSION** ：它的值是1 ，如果环境允许的话它代表的是一个局部的隔离的session，否则就代表普通的session，并且在该session范围内可以访问。

**SCOPE\_GLOBAL\_SESSION** ：它的值是2 ，如果环境允许的话，它代表的是一个全局共享的session，否则就代表普通的session，并且在该session 范围内可以访问。

### postHandle(WebRequest request, ModelMap model)方法

1. 该方法将在请求处理之后，也就是**在Controller 方法调用之后被调用，但是会在视图返回被渲染之前被调用**，所以可以在这个方法里面通过改变数据模型ModelMap 来改变数据的展示。
2. 该方法有两个参数，WebRequest 对象是用于传递整个请求数据的，比如在preHandle 中准备的数据都可以通过WebRequest 来传递和访问；ModelMap 就是Controller 处理之后返回的Model 对象，可以通过改变它的属性来改变返回的Model 模型。

### afterCompletion(WebRequest request, Exception ex)方法

1. 该方法会在整个请求处理完成，也就是在视图返回并被渲染之后执行。
2. 在该方法中可以进行资源的释放操作,而WebRequest参数就可以把我们在preHandle 中准备的资源传递到这里进行释放。Exception 参数表示的是当前请求的异常对象，如果在Controller中抛出的异常已经被Spring 的异常处理器给处理了的话，那么这个异常对象就是是null。

# 配置拦截器到SpringMVC

## XML配置文件



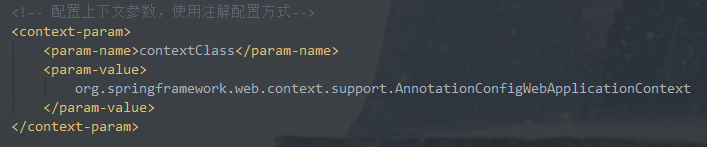
1. 拦截器的执行顺序是按声明的先后顺序执行的，先声明的拦截器中的preHandle方法会先执行，然而它的postHandle方法和afterCompletion方法却会后执行。
2. 直接定义一个Interceptor实现类的bean对象。使用这种方式声明的Interceptor拦截器将会对所有的请求进行拦截。
3. 使用mvc:interceptor标签进行声明。使用这种方式进行声明的Interceptor可以通过mvc:mapping子标签来定义需要进行拦截的请求路径。
4. 可以使用一个星号表示任意字符，使用两个星号表示任意层次的任意字符。

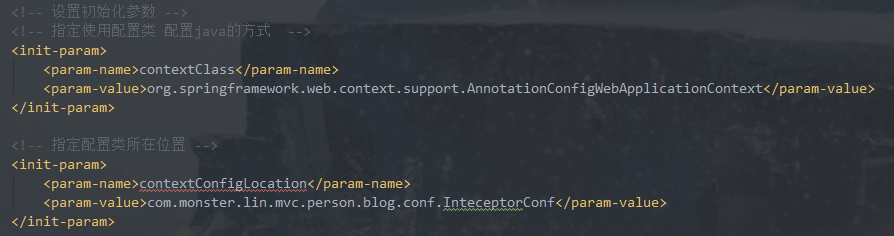


## WebMvcConfigurerAdapter配置类(spring-webmvc)

不起作用，以后再说

(web.xml)





(Java配置类)



# springMVC自带的拦截器

暂时不做了解。

# 拦截器和过滤器的区别

## Filter过滤器

### 介绍

1. Filter可以认为是Servlet的一种“加强版”，它主要用于对用户请求进行预处理，也可以对HttpServletResponse进行后处理，是个典型的处理链。
2. Filter也可以对用户请求生成响应，这一点与Servlet相同，但实际上很少会使用Filter向用户请求生成响应。
3. 使用Filter完整的流程是：Filter对用户请求进行预处理，接着将请求交给Servlet进行预处理并生成响应，最后Filter再对服务器响应进行后处理。
4. **Filter初始化是随着容器初始化的，原理是基于函数调用，init()和destroy()只会发生在容器初始化和销毁的情况下，只有doFilter()方法可以对每个请求作出处理。**

### 1.2 作用

1. 在**HttpServletRequest**到达Servlet之前，拦截客户的HttpServletRequest。
2. 根据需要检查HttpServletRequest，也可以修改HttpServletRequest头和数据。
3. 在**HttpServletResponse**到达客户端之前，拦截HttpServletResponse。
4. 根据需要检查HttpServletResponse，也可以修改HttpServletResponse头和数据。

### Filter分类

1. 用户授权的Filter：Filter负责检查用户请求，根据请求过滤用户非法请求。
2. 日志Filter：详细记录某些特殊的用户请求。
3. 负责解码的Filter:包括对非标准编码的请求解码。
4. 能改变XML内容的XSLT Filter等。
5. Filter可以负责拦截多个请求或响应；一个请求或响应也可以被多个Filter拦截。

### 创建步骤

1. 创建Filter处理类
2. 在web.xml配置Filter类

创建Filter必须实现javax.servlet.Filter接口，在该接口中定义了如下三个方法。

1. void init(FilterConfig config):用于完成Filter的初始化。
2. void destory():用于Filter销毁前，完成某些资源的回收。
3. void doFilter(ServletRequest request,ServletResponse response,FilterChain chain):实现过滤功能，该方法就是对每个请求及响应增加的额外处理。该方法可以实现对用户请求进行预处理(ServletRequest request)，也可实现对服务器响应进行后处理(ServletResponse response)—它们的分界线为是否调用了chain.doFilter(),执行该方法之前，即对用户请求进行预处理；执行该方法之后，即对服务器响应进行后处理。

## 拦截器

### 介绍

1. 拦截器，在**AOP**(Aspect-Oriented Programming)中用于在某个方法或字段被访问之前，进行拦截，然后在之前或之后加入某些操作。拦截是AOP的一种实现策略。**拦截器是动态拦截Action调用的对象。**
2. 拦截器将Action共用的行为独立出来，在Action执行前后执行。这也就是我们所说的AOP，它是分散关注的编程方法，它将通用需求功能从不相关类之中分离出来；同时，能够共享一个行为，一旦行为发生变化，不必修改很多类，只要修改这个行为就可以。
3. 当你提交对Action(默认是.action结尾的url)的请求时，ServletDispatcher会根据你的请求，去调度并执行相应的Action。在Action执行之前，调用被Interceptor截取，Interceptor在Action执行前后执行。
4. **拦截器三个方法，是对于每个请求都会执行一遍的，因为拦截器的原理是基于AOP，针对每个方法进行截获处理。这是和Filter的最大不同。**

### 创建Interceptor

创建Interceptor必须实现org.springframework.web.servlet.HandlerInterceptor接口，该接口定义了如下三个方法。

1. boolean preHandle():截获请求，进行预处理，返回true才能继续执行后续。
2. void postHandle()：在action方法处理完成，还没返回modelAndView时，进行处理。
3. void afterCompletion：在渲染了视图后执行，主要进行资源的清理关闭工作。

## 区别

### Filter和Interceptor的主要区别

1. Filter是基于函数回调的，而Interceptor则是基于Java反射的。
2. **Filter依赖于Servlet容器，而Interceptor不依赖于Servlet容器。**
3. Filter对几乎所有的请求起作用，而Interceptor只能对action请求起作用。
4. Interceptor可以访问Action的上下文，值栈里的对象，而Filter不能。
5. **在action的生命周期里，Interceptor可以被多次调用，而Filter只能在容器初始化时调用一次**

### 执行顺序

过滤前-拦截前-action执行-拦截后-过滤后