        SpringMVC 中的Interceptor 拦截器也是相当重要和相当有用的，它的主要作用是拦截用户的请求并进行相应的处理。比如通过它来进行权限验证，或者是来判断用户是否登陆，或者是像12306 那样子判断当前时间是否是购票时间。

# 一、定义Interceptor实现类

   SpringMVC 中的Interceptor 拦截请求是通过HandlerInterceptor 来实现的。在SpringMVC 中定义一个Interceptor 非常简单，主要有两种方式，第一种方式是要定义的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 ；第二种方式是实现Spring的WebRequestInterceptor接口，或者是继承实现了WebRequestInterceptor的类。

## （一）实现HandlerInterceptor接口

   HandlerInterceptor 接口中定义了三个方法，我们就是通过这三个方法来对用户的请求进行拦截处理的。

   （1 ）preHandle (HttpServletRequest request, HttpServletResponse response, Object handle) 方法，顾名思义，该方法将在请求处理之前进行调用。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。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。

   （2 ）postHandle (HttpServletRequest request, HttpServletResponse response, Object handle, ModelAndView modelAndView) 方法，由preHandle 方法的解释我们知道这个方法包括后面要说到的afterCompletion 方法都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。postHandle 方法，顾名思义就是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，所以我们可以在这个方法中对Controller 处理之后的ModelAndView 对象进行操作。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，这和Struts2 里面的Interceptor 的执行过程有点类型。Struts2 里面的Interceptor 的执行过程也是链式的，只是在Struts2 里面需要手动调用ActionInvocation 的invoke 方法来触发对下一个Interceptor 或者是Action 的调用，然后每一个Interceptor 中在invoke 方法调用之前的内容都是按照声明顺序执行的，而invoke 方法之后的内容就是反向的。

   （3 ）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。顾名思义，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。

下面是一个简单的代码说明：

**Java代码  [IMG_256](http://elim.iteye.com/blog/javascript:void())**

1. **import** javax.servlet.http.HttpServletRequest;
2. **import** javax.servlet.http.HttpServletResponse;
4. **import** org.springframework.web.servlet.HandlerInterceptor;
5. **import** org.springframework.web.servlet.ModelAndView;
7. **public** **class** SpringMVCInterceptor **implements** HandlerInterceptor {

10. /\*\*
11. \* preHandle方法是进行处理器拦截用的，顾名思义，该方法将在Controller处理之前进行调用，SpringMVC中的Interceptor拦截器是链式的，可以同时存在
12. \* 多个Interceptor，然后SpringMVC会根据声明的前后顺序一个接一个的执行，而且所有的Interceptor中的preHandle方法都会在
13. \* Controller方法调用之前调用。SpringMVC的这种Interceptor链式结构也是可以进行中断的，这种中断方式是令preHandle的返
14. \* 回值为false，当preHandle的返回值为false的时候整个请求就结束了。
15. \*/
16. @Override
17. **public** **boolean** preHandle(HttpServletRequest request,
18. HttpServletResponse response, Object handler) **throws** Exception {
19. // TODO Auto-generated method stub
20. **return** **false**;
21. }
23. /\*\*
24. \* 这个方法只会在当前这个Interceptor的preHandle方法返回值为true的时候才会执行。postHandle是进行处理器拦截用的，它的执行时间是在处理器进行处理之
25. \* 后，也就是在Controller的方法调用之后执行，但是它会在DispatcherServlet进行视图的渲染之前执行，也就是说在这个方法中你可以对ModelAndView进行操
26. \* 作。这个方法的链式结构跟正常访问的方向是相反的，也就是说先声明的Interceptor拦截器该方法反而会后调用，这跟Struts2里面的拦截器的执行过程有点像，
27. \* 只是Struts2里面的intercept方法中要手动的调用ActionInvocation的invoke方法，Struts2中调用ActionInvocation的invoke方法就是调用下一个Interceptor
28. \* 或者是调用action，然后要在Interceptor之前调用的内容都写在调用invoke之前，要在Interceptor之后调用的内容都写在调用invoke方法之后。
29. \*/
30. @Override
31. **public** **void** postHandle(HttpServletRequest request,
32. HttpServletResponse response, Object handler,
33. ModelAndView modelAndView) **throws** Exception {
34. // TODO Auto-generated method stub
36. }
38. /\*\*
39. \* 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。该方法将在整个请求完成之后，也就是DispatcherServlet渲染了视图执行，
40. \* 这个方法的主要作用是用于清理资源的，当然这个方法也只能在当前这个Interceptor的preHandle方法的返回值为true时才会执行。
41. \*/
42. @Override
43. **public** **void** afterCompletion(HttpServletRequest request,
44. HttpServletResponse response, Object handler, Exception ex)
45. **throws** Exception {
46. // TODO Auto-generated method stub
48. }
50. }

## （二）实现WebRequestInterceptor 接口

  WebRequestInterceptor 中也定义了三个方法，我们也是通过这三个方法来实现拦截的。这三个方法都传递了同一个参数WebRequest ，那么这个WebRequest 是什么呢？这个WebRequest 是Spring 定义的一个接口，它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。

   （1 ）preHandle(WebRequest request) 方法。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。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，所以我们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。这里可以说说这个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。

   （2 ）postHandle(WebRequest request, ModelMap model) 方法。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。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，我们可以通过改变它的属性来改变返回的Model 模型。

   （3 ）afterCompletion(WebRequest request, Exception ex) 方法。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。所以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 。

   下面是一个简单的代码说明：

**Java代码  [IMG_257](http://elim.iteye.com/blog/javascript:void())**

1. **import** org.springframework.ui.ModelMap;
2. **import** org.springframework.web.context.request.WebRequest;
3. **import** org.springframework.web.context.request.WebRequestInterceptor;
5. **public** **class** AllInterceptor **implements** WebRequestInterceptor {
7. /\*\*
8. \* 在请求处理之前执行，该方法主要是用于准备资源数据的，然后可以把它们当做请求属性放到WebRequest中
9. \*/
10. @Override
11. **public** **void** preHandle(WebRequest request) **throws** Exception {
12. // TODO Auto-generated method stub
13. System.out.println("AllInterceptor...............................");
14. request.setAttribute("request", "request", WebRequest.SCOPE\_REQUEST);//这个是放到request范围内的，所以只能在当前请求中的request中获取到
15. request.setAttribute("session", "session", WebRequest.SCOPE\_SESSION);//这个是放到session范围内的，如果环境允许的话它只能在局部的隔离的会话中访问，否则就是在普通的当前会话中可以访问
16. request.setAttribute("globalSession", "globalSession", WebRequest.SCOPE\_GLOBAL\_SESSION);//如果环境允许的话，它能在全局共享的会话中访问，否则就是在普通的当前会话中访问
17. }
19. /\*\*
20. \* 该方法将在Controller执行之后，返回视图之前执行，ModelMap表示请求Controller处理之后返回的Model对象，所以可以在
21. \* 这个方法中修改ModelMap的属性，从而达到改变返回的模型的效果。
22. \*/
23. @Override
24. **public** **void** postHandle(WebRequest request, ModelMap map) **throws** Exception {
25. // TODO Auto-generated method stub
26. **for** (String key:map.keySet())
27. System.out.println(key + "-------------------------");;
28. map.put("name3", "value3");
29. map.put("name1", "name1");
30. }
32. /\*\*
33. \* 该方法将在整个请求完成之后，也就是说在视图渲染之后进行调用，主要用于进行一些资源的释放
34. \*/
35. @Override
36. **public** **void** afterCompletion(WebRequest request, Exception exception)
37. **throws** Exception {
38. // TODO Auto-generated method stub
39. System.out.println(exception + "-=-=--=--=-=-=-=-=-=-=-=-==-=--=-=-=-=");
40. }
42. }

# 二、把定义的拦截器类加到SpringMVC的拦截体系中

## 1.在SpringMVC的配置文件中加上支持MVC的schema

**Xml代码  [IMG_258](http://elim.iteye.com/blog/javascript:void())**

1. xmlns:mvc="http://www.springframework.org/schema/mvc"
2. xsi:schemaLocation=" http://www.springframework.org/schema/mvc
3. http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd"

         下面是我的声明示例：

**Xml代码  [IMG_259](http://elim.iteye.com/blog/javascript:void())**

1. **<beans** xmlns="http://www.springframework.org/schema/beans"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
3. xmlns:mvc="http://www.springframework.org/schema/mvc"
4. xsi:schemaLocation="http://www.springframework.org/schema/beans
5. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
6. http://www.springframework.org/schema/context
7. http://www.springframework.org/schema/context/spring-context-3.0.xsd
8. http://www.springframework.org/schema/mvc
9. http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd"**>**

         这样在SpringMVC的配置文件中就可以使用mvc标签了，mvc标签中有一个mvc:interceptors是用于声明SpringMVC的拦截器的。

## （二）使用mvc:interceptors标签来声明需要加入到SpringMVC拦截器链中的拦截器

**Xml代码  [IMG_260](http://elim.iteye.com/blog/javascript:void())**

1. **<mvc:interceptors>**
2. <!-- 使用bean定义一个Interceptor，直接定义在mvc:interceptors根下面的Interceptor将拦截所有的请求 -->
3. **<bean** class="com.host.app.web.interceptor.AllInterceptor"**/>**
4. **<mvc:interceptor>**
5. **<mvc:mapping** path="/test/number.do"**/>**
6. <!-- 定义在mvc:interceptor下面的表示是对特定的请求才进行拦截的 -->
7. **<bean** class="com.host.app.web.interceptor.LoginInterceptor"**/>**
8. **</mvc:interceptor>**
9. **</mvc:interceptors>**

          由上面的示例可以看出可以利用mvc:interceptors标签声明一系列的拦截器，然后它们就可以形成一个拦截器链，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。

          在mvc:interceptors标签下声明interceptor主要有两种方式：

                    （1）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。

                    （2）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。

          经过上述两步之后，定义的拦截器就会发生作用对特定的请求进行拦截了。