Servlet过滤器与SpringMVC拦截器之间的关系与区别

**①拦截器是基于Java的反射机制的，而过滤器是基于函数回调。**

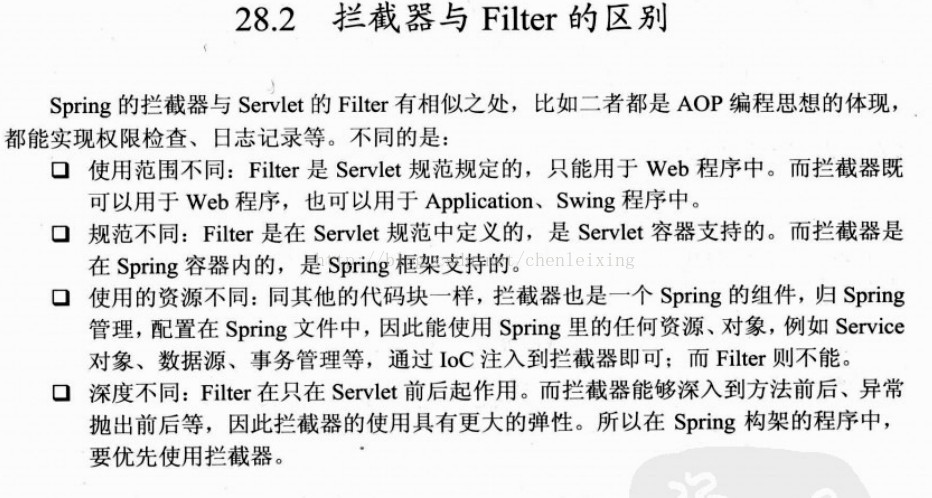
**②拦截器不依赖与servlet容器，过滤器依赖与servlet容器，即只能在servlet容器中执行。**

**③拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用。**

**④拦截器可以访问action上下文、值栈里的对象，而过滤器不能访问。**

**⑤在action的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次。**

**⑥拦截器可以获取IOC容器中的各个bean，而过滤器就不行，这点很重要，在拦截器里注入一个service，可以调用业务逻辑。**



写了点[**测试**](http://lib.csdn.net/base/softwaretest)代码，顺便整理一下思路，搞清楚这几者之间的顺序：

　　1.过滤器是JavaEE标准，采用函数回调的方式进行。是在请求进入容器之后，还未进入Servlet之前进行预处理，并且在请求结束返回给前端这之间进行后期处理。

1. **@Override**
2. **public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {**
3. **System.out.println("before...");**
4. **chain.doFilter(request, response);**
5. **System.out.println("after...");**
6. **}**

chain.doFilter(request, response);这个方法的调用作为分水岭。事实上调用Servlet的doService()方法是在chain.doFilter(request, response);这个方法中进行的。

　　2.拦截器是被包裹在过滤器之中的。

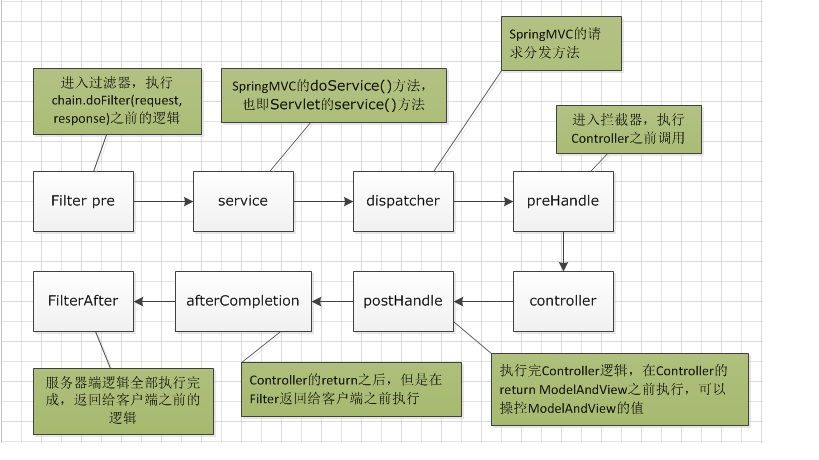
1. **public** **class** SpringMVCInterceptor **implements** HandlerInterceptor {
2. /\*\*
3. \* preHandle方法是进行处理器拦截用的，顾名思义，该方法将在Controller处理之前进行调用，SpringMVC中的Interceptor拦截器是链式的，可以同时存在
4. \* 多个Interceptor，然后SpringMVC会根据声明的前后顺序一个接一个的执行，而且所有的Interceptor中的preHandle方法都会在
5. \* Controller方法调用之前调用。SpringMVC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
6. \* 回值为false，当preHandle的返回值为false的时候整个请求就结束了。
7. \*/
8. @Override
9. **public** boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {
10. System.out.println("preHandle");
11. return true;
12. }
13. /\*\*
14. \* 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
15. \* 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
16. \* 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，
17. \* 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
18. \* 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。
19. \*/
20. @Override
21. **public** void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) **throws** Exception {
22. System.out.println("postHandle");
23. }
24. /\*\*
25. \* 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，
26. \* 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。
27. \*/
28. @Override
29. **public** void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) **throws** Exception {
30. System.out.println("afterCompletion");
31. }
32. }

　　 a.preHandle()这个方法是在过滤器的chain.doFilter(request, response)方法的前一步执行，也就是在 [System.out.println("before...")][chain.doFilter(request, response)]之间执行。

　　b.postHandle()方法之后，在return ModelAndView之前进行，可以操控Controller的ModelAndView内容。

c.afterCompletion()方法是在过滤器返回给前端前一步执行，也就是在[chain.doFilter(request, response)][System.out.println("after...")]之间执行。

　　3.SpringMVC的机制是由同一个Servlet来分发请求给不同的Controller，其实这一步是在Servlet的service()方法中执行的。所以过滤器、拦截器、service()方法，dispatc()方法的执行顺序应该是这样的，大致画了个图：其实非常好测试，自己写一个过滤器，一个拦截器，然后在这些方法中都加个断点，一路F8下去就得出了结论。

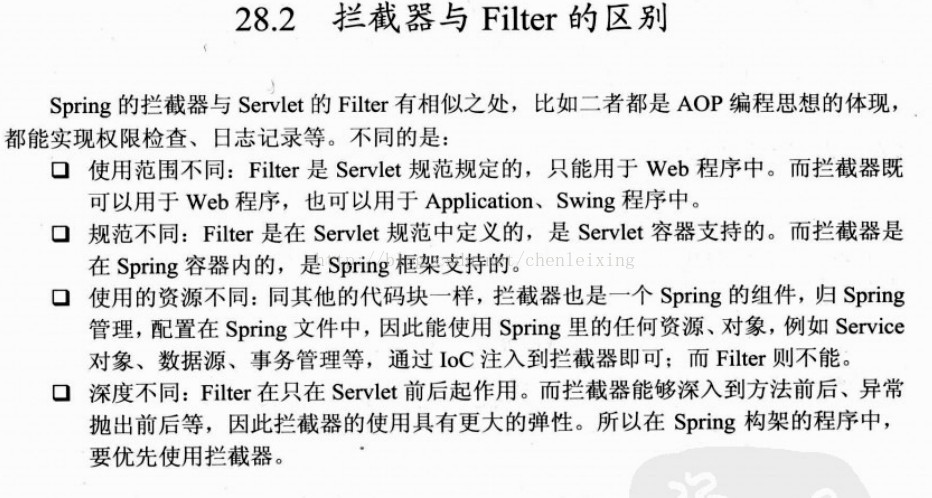


**总结：**拦截器功在对请求权限鉴定方面确实很有用处，在我所参与的这个项目之中，第三方的远程调用每个请求都需要参与鉴定，所以这样做非常方便，而且他是很独立的逻辑，这样做让业务逻辑代码很干净。和框架的其他功能一样，原理很简单，使用起来也很简单，大致看了下SpringMVC这一部分的源码，其实还是比较容易理解的。

　　我们项目中仅仅用到了preHandle这个方法，而未用其他的，框架提供了一个已经实现了拦截器接口的适配器类HandlerInterceptorAdapter，继承这个类然后重写一下需要用到的方法就行了，可以少几行代码，这种方式Java中很多地方都有体现。

以上部分是摘自[神一样存在的博客](http://www.cnblogs.com/dreamroute/p/4198087.html?utm_source=tuicool)，参考了一下这个帖子：http://haohaoxuexi.iteye.com/blog/1750680

**大家还可以参考一下这个电子书的截图：**



其他网友理解：

Filter基于回调函数，我们需要实现的filter接口中doFilter方法就是回调函数，而interceptor则基于java本身的反射机制,这是两者最本质的区别。

Filter是依赖于servlet容器的，即只能在servlet容器中执行，很显然没有servlet容器就无法来回调doFilter方法。而interceptor与servlet容器无关。

Filter的过滤范围比Interceptor大,Filter除了过滤请求外通过通配符可以保护页面，图片，文件等等，而Interceptor只能过滤请求。

Filter的过滤例外一般是在加载的时候在init方法声明,而Interceptor可以通过在xml声明是guest请求还是user请求来辨别是否过滤

简单的总结理解：

1：Filter 对在web.xml中该filter配置的路径进行单向拦截， request匹配Filter的路径时做一次拦截！ 这个底层实现没想过， 嘿嘿！

2：interceptor的拦截处理其实就是代理机制！

Struts2官方文档那个图画的相当明了了。既然是代理，那么是双向拦截，拦截器在action的方法之前执行一次， 之后执行一次！

根据那个图，你可以再对应的去看下StrutsPrepareAndExecuteFilter的源代码，前面一段代码产生Action的环境，判断mapping之类的， 可以跟进ExecuteOperations execute这个成员变量，下面如果前面一系列判断都满足的话， 会调用execute.executeAction， 这个里面的实现就是产生action的代理， 然后通过配置信息， 去获得这个action身上配置了哪些拦截器，然后执行拦截器的代码，拦截器中用ActionInvocation去通知下一个拦截器执行， 等到所有的拦截器都拦完了以后，执行action， 随后拦截器又会反向执行一遍（官方确实是这个意思， LZ可以自己写两个拦截器测试下看是不是这么会是）

所以对于Interceptor是不对request进行处理的， 它只是对action起作用。Struts的那个核心过滤器拦截了action的请求之后产生action的代理后进行处理的！

我只能理解这么多， 呵呵！ LZ如果有好的见解记得分享下！

**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  [收藏代码](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  [收藏代码](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  [收藏代码](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。