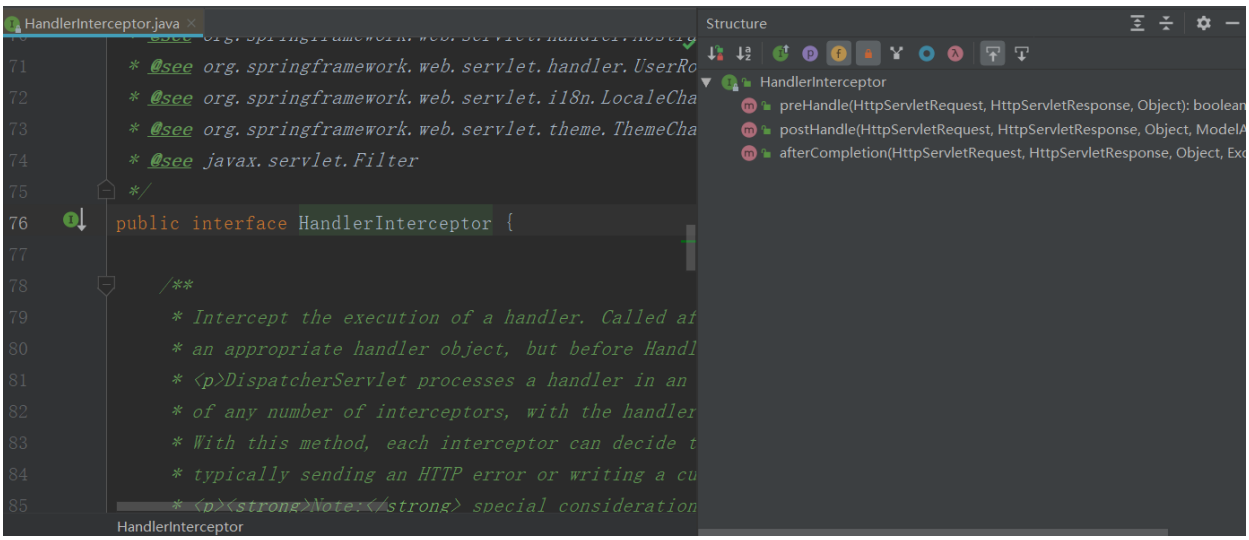
07-SpringMVC基于注解使用：拦截器

1、Springmvc拦截器

拦截器采用AOP的设计思想， 它跟过滤器类似， 用来拦截处理方法在之前和之后执行一些跟主业务没有关系的一些公共功能：

比如：可以实现:权限控制、日志、异常记录、记录方法执行时间.....

SpringMVC提供了拦截器机制，允许运行目标方法之前进行一些拦截工作或者目标方法运行之后进行一下其他相关的处理。自定义的拦截器必须实现**HandlerInterceptor**接口。



拦截器一个有3个回调方法，而一般的过滤器Filter才两个:

preHandle：预处理回调方法，实现处理器的预处理（如登录检查），第三个参数为响应的处理器返回值：true表示继续流程（如调用下一个拦截器或处理器）；false表示流程中断（如登录检查失败），不会继续调用其他的拦截器或处理器，此时我们需要通过response来产生响应；

postHandle：后处理回调方法，实现处理器的后处理（但在渲染视图之前），此时我们可以通过modelAndView（模型和视图对象）对模型数据进行处理或对视图进行处理，modelAndView也可能为null。

afterCompletion：整个请求处理完毕回调方法，即在视图渲染完毕时回调，如性能监控中我们可以在此记录结束时间并输出消耗时间，还可以进行一些资源清理，类似于try-catch-finally中的finally，但仅调用处理器执行链中preHandle返回true的拦截器才会执行

1、自定义拦截器

MyFirstInterceptor.java

```
package cn.tulingxueyuan.interceptor;


import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyFirstInterceptor implements HandlerInterceptor {
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Ob
        System.out.println(this.getClass().getName()+"------->preHandle");
        return true;
    }

    public void postHandle(HttpServletRequest request, HttpServletResponse response, Obje
        System.out.println(this.getClass().getName()+"------->postHandle");
    }

    public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
        System.out.println(this.getClass().getName()+"------->afterCompletion");
    }
}
```

TestInterceptorController.java

```
package cn.tulingxueyuan.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class TestInterceptorController {
```

```
8
9     @RequestMapping("test01")
10    public String test01(){
11        System.out.println("test01");
12        return "success";
13    }
14 }
15 springmvc.xml
16 <?xml version="1.0" encoding="UTF-8"?>
17 <beans xmlns="http://www.springframework.org/schema/beans"
18        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
19        xmlns:context="http://www.springframework.org/schema/context"
20        xmlns:mvc="http://www.springframework.org/schema/mvc"
21
22        xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springfr
23    <mvc:default-servlet-handler></mvc:default-servlet-handler>
24    <mvc:annotation-driven></mvc:annotation-driven>
25    <context:component-scan base-package="cn.tulingxueyuan"></context:component-scan>
26
27    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
28        <property name="prefix" value="/WEB-INF/page/"></property>
29        <property name="suffix" value=".jsp"></property>
30    </bean>
31    <mvc:interceptors>
32        <bean class="cn.tulingxueyuan.interceptor.MyFirstInterceptor"></bean>
33    </mvc:interceptors>
34 </beans>
35 success.jsp
36 <%@ page contentType="text/html;charset=UTF-8" language="java" %>
37 <html>
38 <head>
39    <title>Title</title>
40 </head>
41 <body>
42 <% System.out.println("success.jsp");%>
43 success
44 </body>
45 </html>
```

通过运行结果能够发现拦截器的执行顺序如下：

可以看到先执行拦截器的preHandle方法----》执行目标方法----》执行拦截器的postHandle方法----》执行页面跳转----》执行拦截器的afterCompletion方法

在配置拦截器的时候有两个需要注意的点：

1、如果prehandle方法返回值 为false，那么意味着不放行，那么就会造成后续的所有操作都中断

2、如果执行到方法中出现异常，那么后续流程不会处理但是afterCompletion方法会执行

2、定义多个拦截器

再添加另外一个拦截器

MySecondInterceptor.java

```
package cn.tulingxueyuan.interceptor;


import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MySecondInterceptor implements HandlerInterceptor {
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Obj
        System.out.println(this.getClass().getName()+"------->preHandle");
        return true;
    }

    public void postHandle(HttpServletRequest request, HttpServletResponse response, Obje
        System.out.println(this.getClass().getName()+"------->postHandle");
    }

    public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
        System.out.println(this.getClass().getName()+"------->afterCompletion");
    }
}
```

看到如下执行顺序：

调整两个拦截器的配置顺序：

```
com.mashibing.interceptor.MySecondInterceptor------->preHandle
com.mashibing.interceptor.MyFirstInterceptor------->preHandle
test01
com.mashibing.interceptor.MyFirstInterceptor------->postHandle
com.mashibing.interceptor.MySecondInterceptor------->postHandle
success.jsp
com.mashibing.interceptor.MyFirstInterceptor------->afterCompletion
com.mashibing.interceptor.MySecondInterceptor------->afterCompletion
```

大家可以看到对应的效果，谁先执行取决于配置的顺序。

拦截器的preHandle是按照顺序执行的

拦截器的postHandle是按照逆序执行的

拦截器的afterCompletion是按照逆序执行的

如果执行的时候核心的业务代码出问题了，那么已经通过的拦截器的afterCompletion会接着执行。

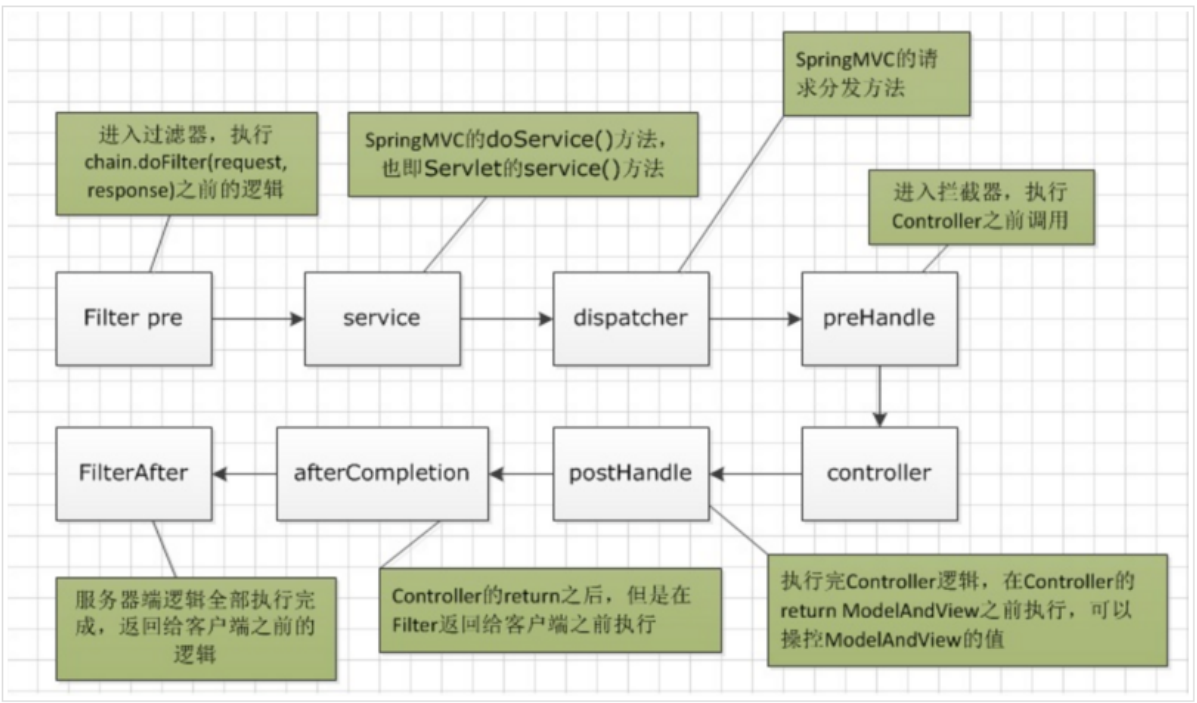
2、拦截器跟过滤器的区别

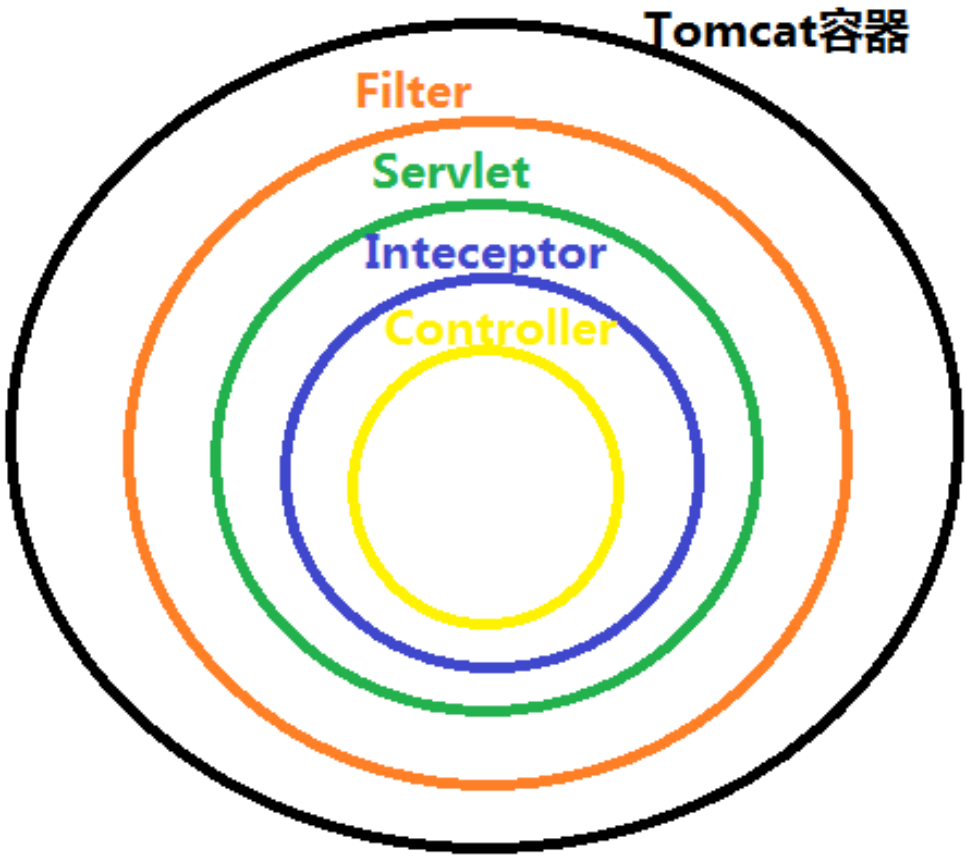
1、过滤器是基于函数回调的，而拦截器是基于java反射的

2、过滤器依赖于servlet容器，而拦截器不依赖与Servlet容器，拦截器依赖SpringMVC

3、过滤器几乎对所有的请求都可以起作用，而拦截器只能对SpringMVC请求起作用

4、拦截器可以访问处理方法的上下文，而过滤器不可以






```
====================过滤器前====================
---------方法后执行，在渲染之前--------------preHandle
请求方法执行中...
---------方法后执行，在渲染之前--------------postHandle
---------在视图渲染之后--------------afterCompletion
====================过滤器后====================
```

3、使用拦截器实现登录权限拦截

·1、拦截器

```
@Override
    public boolean preHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler) throws Exception {
        System.out.println("-------拦截----------");
        if(request.getSession().getAttribute("user")==null){
            response.sendRedirect(request.getContextPath()+"/login");
            return false;
        }
        return true;
```

```
    }
```

2、拦截映射和排除拦截

```
<mvc:interceptor>
<mvc:mapping path="/**"/>
<mvc:exclude-mapping path="/login"/>
<bean class="com.ns.interceptors.CheckLoginInterceptor"></bean>
</mvc:interceptor>
```