SpringMVC拦截器

1、拦截器是属于SpringMVC框架中的一种，需要实现HandlerInterceptor接口
2、拦截器和过滤器类似，但功能的侧重点方向不同。
    过滤器是用来过滤请求参数的、设置编码字符集等工作的
    拦截器是拦截用户请求，对用户的请求做判断处理的
3、拦截器是一个全局作用的，可以对多个Controller做拦截。
    一个项目中可以实现0个或多个拦截器，他们一起拦截用户的请求
    常用于：用户登录处理、权限检查、记入日志等

拦截器使用步骤：
1、定义一个普通类实现HandlerInterceptor接口
2、在SpringMVC配置文件中声明所定义的拦截器对象

拦截器的执行时间点：
    1）、可以在请求处理之前，即在执行Controller对象的方法之前执行拦截器，达到拦截请求
    2）、可以在控制器方法执行之后整个请求为完成之前执行拦截器方法拦截请求
    3）、可以在请求处理完成后执行拦截器，但是这样一般都是做释放资源等的收尾功能

创建项目例子进行拦截器的使用说明

步骤：
1、创建maven项目
2、添加相应的依赖
3、创建Controller类
4、创建一个普通类，作为拦截器使用：
    1）、实现HandlerInterceptor接口
    2）、实现接口中的三个方法
5、创建各种情况所需要转发到的jsp页面
6、创建SpringMVC的配置文件：
    1）、声明组件扫描器，扫描@Controller注解
    2）、声明拦截器，同时指定拦截的请求URL地址
    3）、声明视图解析器

项目模块创建在idea-maven-springmvc工程中，项目名f-springmvc-interceptor-06

1、pom文件中的内容如下

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.studymyself</groupId>
  <artifactId>f-springmvc-interceptor-06</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
```

```
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>

    <!--    servlet依赖-->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
      <scope>provided</scope>
    </dependency>

    <!--    SpringMVC依赖-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.2.5.RELEASE</version>
    </dependency>

  </dependencies>

  <build>

  </build>
</project>
```

2、创建Controller类

```
package com.studymyself.controller;

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

@Controller
@RequestMapping("/myInterceptor")
public class MyController {


    @RequestMapping("/some.do")
    public ModelAndView doSome(String name,Integer age){

        ModelAndView mv = new ModelAndView();
        mv.addObject("myName",name);
        mv.addObject("myAge",age);
```

```
        mv.setViewName("show");

        return mv;

    }

}
```

3、创建一个普通类，作为拦截器使用：

1）、实现HandlerInterceptor接口
2）、实现接口中的三个方法

HandlerInterceptor接口内容如下：

```
public interface HandlerInterceptor {
    default boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
        return true;
    }

    default void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler, @Nullable ModelAndView modelAndView) throws Exception
{
    }

    default void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, @Nullable Exception ex) throws Exception {
    }
}
/*
    该接口中对应有三个方法，在Spring框架的5.0版本后，接口中的方法前面都有一个default修饰，表示实现这个接口，你需要使用哪个方法就实现哪个方法，其他非法不实现也没问题。因为我们最常用的是上面的preHandler方法，所以一般我们就实现这个方法就行了
*/
```

```
package com.studymyself.handler;

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

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.Date;

//拦截器类：拦截用户的请求
public class MyInterceptor implements HandlerInterceptor {

    long begin;

    /**
     *  preHandler叫做预处理方法：
     *      非常重要：是整个项目的入口、门户。
     *      当preHandle返回true，请求可以被处理；反之请求被拦截
```

```
     *  参数：
     *      Object handler：是被拦截的控制器对象，就是Controller对象，在配置文件中声明
     *                      拦截器的步骤中，框架会根据指定的信息自动给该参数赋值
     *  返回值boolean：
     *      true：说明请求通过了拦截器的验证，可以执行处理器方法
     *      输出如下：
     *      拦截器MyInterceptor的preHandle方法执行
     *      ==执行doSome方法==
     *      拦截器MyInterceptor的postHandle方法执行
     *      拦截器MyInterceptor的afterCompletion方法执行
     *
     *      false：说明请求无法通过了拦截器的验证，请求到达拦截器就截止了，请求没有被处理
     *      输出如下：
     *      拦截器MyInterceptor的preHandle方法执行
     *
     *  特点：
     *      1、在控制器方法执行之前执行
     *      用户的请求首先到达这里
     *      2、在该方法中我们可以获取请求的信息，然后验证是否符合要求。如
     *      验证用户是否是登入状态发送该请求，验证用户是否有该权限访问这个资源链接
     *      验证失败，返回false，即可截断请求，请求不能被处理
     *      验证成功，返回true，放行请求，执行对应的控制器方法
     */
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {

        begin = System.currentTimeMillis();
        System.out.println("拦截器MyInterceptor的preHandle方法执行");
        boolean re = true;
        //在这里获取请求信息，计算业务逻辑，根据计算结果，返回true或是false
        if (!re){

 request.getRequestDispatcher("/view/tip.jsp").forward(request,response);
            return re;
        }
        return re;
    }

    /**
     *  postHandler方法:处理器方法执行完毕后，请求还未完成前执行
     *  参数：
     *      Object handler：是被拦截的控制器对象，就是Controller对象，在配置文件中声明
     *                      拦截器的步骤中，框架会根据指定的信息自动给该参数赋值
     *      ModelAndView mv：框架将处理器方法执行的返回值赋值给该参数mv
     *  特点：
     *      1、理器方法执行完毕后，请求还未完成前执行该方法
     *      2、在该方法中获取处理器方法的ModelAndView返回值mv，可以修改
     *      mv中的数据和视图，影响最后的执行结果
     *      作用：对原来执行结果进行二次修正
     */
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("拦截器MyInterceptor的postHandle方法执行");

        //通过获取处理器方法中的返回值ModelAndView，通过
        //判断返回的数据视图是否符合预期结果，不符合就调整，或者是给
```

```
        //处理器方法的执行结果进行增添
        if (modelAndView!=null){
            //对返回值中的数据进行添加
            modelAndView.addObject("myDate",new Date());
            //不修改视图
        }
    }

    /**
     *  afterCompletion方法：在请求处理完成后执行
     *  参数：
     *      Object handler：被拦截的控制器对象，就是Controller对象
     *      Exception ex：处理器方法发生的异常
     *  特点：
     *      1、在请求被处理完成后执行该方法。
     *      框架中是规定在视图处理完成后，即底层对视图执行转发forward方法后认为请求处理完成
     *      2、一般做资源回收工作，程序请求过程中创建了一些对象，可以在该方法中进行回收，释放内存
     */
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {
        System.out.println("拦截器MyInterceptor的afterCompletion方法执行");
        long end = System.currentTimeMillis();
        //这里计算从preHandle方法开始到请求处理完成的时间
        System.out.println("计算preHandle方法开始执行到请求处理完成时间："+(end-begin));
    }
}
```

4、创建各种情况所需要转发到的jsp页面

首页、欢迎页面index.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>欢迎页面</title>
</head>
<body>
<P>提交参数给Controller，接收请求参数</P>
<form action="myInterceptor/some.do" method="post">
    姓名:<input type="text" name="name"><br>
    年龄:<input type="text" name="age"><br>
    <input type="submit" value="提交参数">
</form>
</body>
</html>
```

未拦截的结果展示页面show.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>正确结果展示</title>
</head>
<body>
    show.jsp结果展示页面<br>
    myName数据：${myName}<br>
    myAge数据：${myAge}<br>
    拦截器postHandle方法中添加的myDate数据：${myDate}
</body>
</html>
```

被拦截后的提示页面tip.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>提示</title>
</head>
<body>
    tip.jsp提示展示页面<br>
    该用户无法访问该请求，该请求被拦截！！！
</body>
</html>
```

5、创建SpringMVC的配置文件：

1）、声明组件扫描器，扫描@Controller注解
2）、声明拦截器，同时指定拦截的请求URL地址
3）、声明视图解析器

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/mvc
https://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!--声明组件扫描器-->
    <context:component-scan base-package="com.studymyself.controller"/>

    <!--声明配置视图解析器,框架提供的一个类-->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!--前缀属性：视图文件的路径，前后要加/，前一个/表示项目根路径:
http://localhost:8080/a_springmvc_web_01-->
        <property name="prefix" value="/view/"/>
```

```
        <!--后缀属性：视图文件的类型-->
        <property name="suffix" value=".jsp"/>
    </bean>

<!--声明拦截器：可以有多个，也可以没有，所以下面的第一个拦截器标签是复数
        先声明的拦截器的preHandle方法先声明，后两个方法怎么执行看笔记
        框架中用ArrayList集合保存声明的拦截器，先声明的先放，所以
        先取出来，所以有先后执行那个拦截器的preHandle方法的顺序-->
    <mvc:interceptors>
        <!--声明第一个拦截器-->
        <mvc:interceptor>
            <!--下面指定拦截的请求URL地址：
                path：填的是URL地址，可以使用**通配符
                  **：表示任意的字符，文件或者多级目录和目录中的文件
                  下面以/开头，表示的是该项目的根路径下查找
                  http://localhost:8080/myapp/myInterceptor/some.do
                  其中myapp目录就是根目录-->
            <mvc:mapping path="/myInterceptor/**"/>
            <!--声明拦截器对象-->
            <bean class="com.studymyself.handler.MyInterceptor"/>
        </mvc:interceptor>
    </mvc:interceptors>

</beans>
```

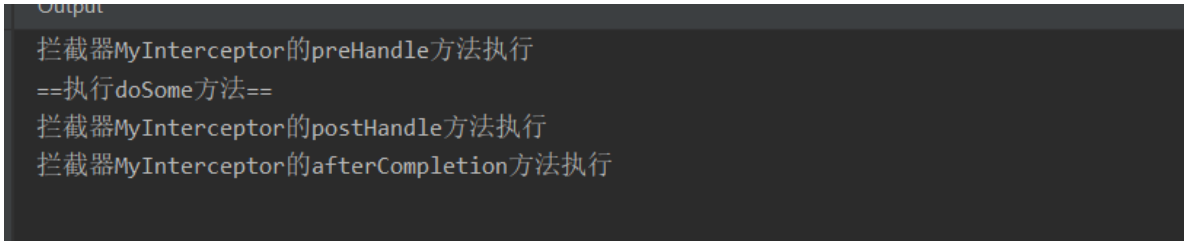
6、配置web.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <servlet>
        <servlet-name>springMVC</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springmvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springMVC</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>

    <!--声明注册过滤器，解决POST请求中文乱码-->
    <filter>
        <filter-name>characterEncodingFilter</filter-name>
        <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <!--CharacterEncodingFilter中有三个属性：
            private String encoding;
            private boolean forceRequestEncoding;
            private boolean forceResponseEncoding;
```

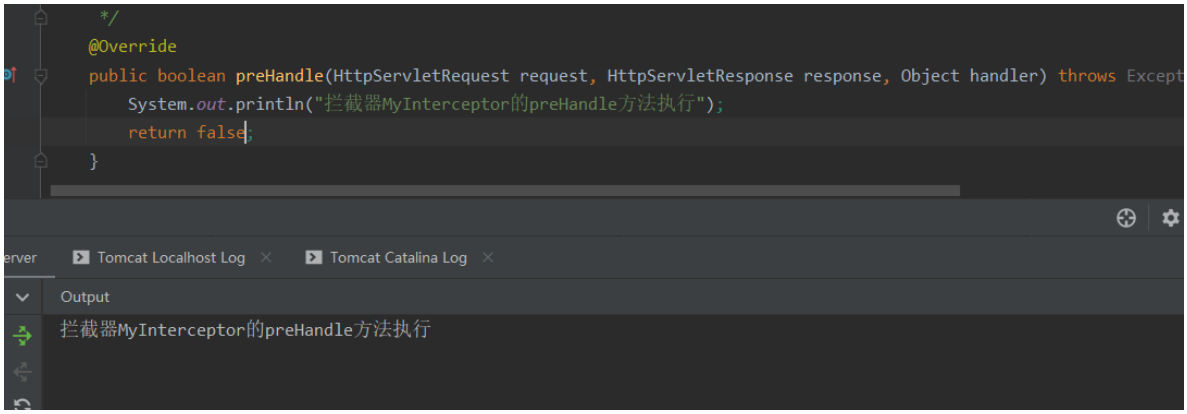
```
        -->
        <!--该属性设置项目中使用的字符编码方式-->
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <!--该属性为true时强制请求对象（HttpServletRequest）使用encoding属性的值的编码-
->
        <init-param>
            <param-name>forceRequestEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
        <!--该属性为true时强制响应对象（HttpServletResponse）使用encoding属性的值的编
码-->
        <init-param>
            <param-name>forceResponseEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>characterEncodingFilter</filter-name>
        <!--将这个过滤器的url-pattern设置为/*
            就是在启动项目后，所有的请求发送后无论是什么请求路径都会
            执行这个过滤器对象，每一次的请求生成的请求对象request和响应对象response
            的编码方式都设置为encoding的值的编码，然后在执行其他对应的Servlet，这样我们
            就不必每一个处理器方法都设置编码方式了。
            -->
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

7、测试结果

测试preHandle方法返回true时的结果



测试preHandle方法返回false时的结果，可以看到只要该方法返回false，后面所有的包括处理器方法都不再执行

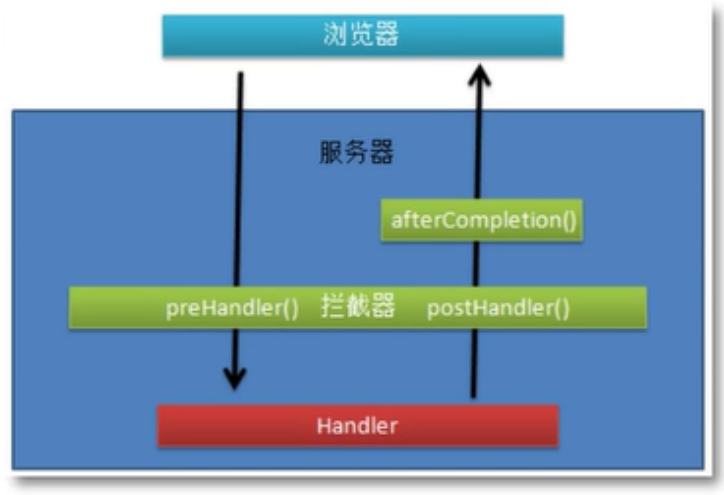


测试postHandle方法，给返回值的数据中添加数据

show.jsp结果展示页面
myName数据：gaoerfu
myAge数据：151
拦截器postHandle方法中添加的myDate数据：Mon Feb 22 20:32:24 CST 2021

测试afterCompletion方法，计算时间，可以发现第一次由于需要完成请求的初始化工作（包括对象的创建等），耗时很长，之后的请求耗时变短

```
拦截器MyInterceptor的preHandle方法执行
==执行doSome方法==
拦截器MyInterceptor的postHandle方法执行
拦截器MyInterceptor的afterCompletion方法执行
计算preHandle方法开始执行到请求处理完成时间：891
拦截器MyInterceptor的preHandle方法执行
==执行doSome方法==
拦截器MyInterceptor的postHandle方法执行
拦截器MyInterceptor的afterCompletion方法执行
计算preHandle方法开始执行到请求处理完成时间：3
拦截器MyInterceptor的preHandle方法执行
==执行doSome方法==
拦截器MyInterceptor的postHandle方法执行
拦截器MyInterceptor的afterCompletion方法执行
计算preHandle方法开始执行到请求处理完成时间：2
```



8、创建多个拦截器，新建一个MyInterceptor2实现HandlerInterceptor接口，如下

```
package com.studymyself.handler;

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

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

//第二个拦截器
public class MyInterceptor2 implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
        System.out.println("222拦截器MyInterceptor的preHandle方法执行222");
        return false;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("222拦截器MyInterceptor的postHandle方法执行222");
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) throws Exception {
        System.out.println("222拦截器MyInterceptor的afterCompletion方法执行222");
    }
}
```

在SpringMVC配置文件中声明第二个拦截器，跟拦截器1拦截同样的请求

```
   <mvc:interceptors>
        <!--声明第一个拦截器-->
        <mvc:interceptor>
            <!--下面指定拦截的请求URL地址：
                path：填的是URL地址，可以使用**通配符
                  **：表示任意的字符，文件或者多级目录和目录中的文件
                  下面以/开头，表示的是该项目的根路径下查找
                  http://localhost:8080/myapp/myInterceptor/some.do
                  其中myapp目录就是根目录-->
            <mvc:mapping path="/myInterceptor/**"/>
            <!--声明拦截器对象-->
            <bean class="com.studymyself.handler.MyInterceptor"/>
        </mvc:interceptor>

        <!--声明第二个拦截器-->
        <mvc:interceptor>
            <mvc:mapping path="/myInterceptor/**"/>
            <!--声明拦截器对象-->
            <bean class="com.studymyself.handler.MyInterceptor2"/>
        </mvc:interceptor>
    </mvc:interceptors>
```

部署项目进行测试

通过以上我们可以知道，拦截器：

可以看做多个Controller类公用的功能，该功能是验证请求是否符合标准的，访问所有控制器中的方法的请求的验证集中到拦截器中，统一进行处理，使用的就是AOP的思想

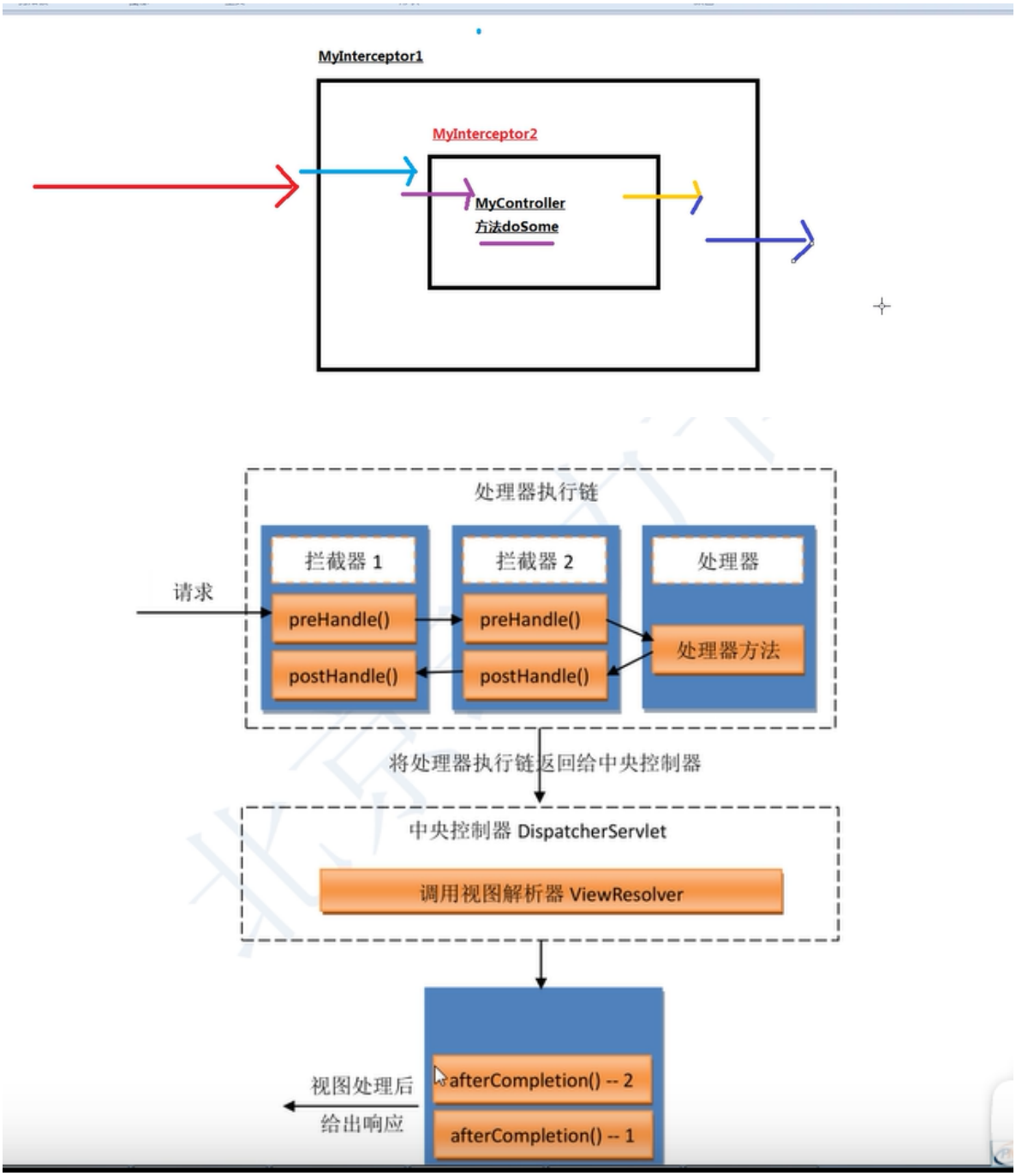
进行多个拦截器时：

1、第一个拦截器是preHandle返回true，第二个拦截器preHandle返回true，执行的结果为：

```
111拦截器MyInterceptor的preHandle方法执行111
222拦截器MyInterceptor的preHandle方法执行222
==执行doSome方法==
222拦截器MyInterceptor的postHandle方法执行222
111拦截器MyInterceptor的postHandle方法执行111
222拦截器MyInterceptor的afterCompletion方法执行222
111拦截器MyInterceptor的afterCompletion方法执行111
```

从中我们看到，在执行处理器方法之前，是拦截器1的preHandle先执行，再到拦截器2的preHandle；

但是处理器方法执行后，后面两个方法的执行顺序，却是拦截器2的先执行，具体原理如下图：



可以看到，拦截器对请求的访问资源是包围式的，所以有上面的执行顺序

2、第一个拦截器是preHandle返回true，第二个拦截器preHandle返回false，执行的结果为：

111拦截器MyInterceptor的preHandle方法执行111

222拦截器MyInterceptor的preHandle方法执行222

111拦截器MyInterceptor的afterCompletion方法执行111

出现这样的执行顺序原因是，afterCompletion方法的执行与否取决于自身拦截器的preHandle方法返回值，返回true，afterCompletion方法一定执行，返回false不执行。而postHandle方法执行第一要素是preHandle返回真，第二要素是处理器方法执行后返回一个值

9、拦截器和过滤器的区别

1、过滤器是servlet规范中的对象，而拦截器是Spring框架中的对象

所以声明过滤器在web.xml中声明，在spring配置文件中声明拦截器

2、过滤器实现的是Filter接口，拦截器实现handlerInterceptor接口

3、过滤器是用来设置request、response的参数、属性的，侧重对数据的过滤

拦截器是用来验证请求的不符合要求的请求会被截断

4、过滤器会在拦截器之前执行

5、过滤器是服务器创建的对象，而拦截器是SpringMVC容器创建的对象

6、过滤器只有一个时间点，拦截器有三个时间点

7、过滤器可以处理jsp、js、html等

拦截器侧重拦截对Controller的对象，请求没有给中央调度器DispatcherServlet接收，请求就不会被拦截器拦截。因为拦截器最后还是把返回的数据交给中央调度器