SpringMVC拦截器（重点）

一、基本介绍

1、概念

SpringMVC的处理器拦截器,类似于Servlet开发中的过滤器Filter，用于对处理器进行预处理和后处理。

拦截器链：就是将拦截器按一定的顺序联结成一条链。在访问被拦截的方法或字段时，拦截器链中的拦截器就会按其之前定义的顺序被调用。

2、过滤器和拦截器的区别

（1）过滤器：

依赖于servlet容器，任何 java web 工程都可以使用；
在实现上基于函数回调，可以对几乎所有请求进行过滤
缺点：是一个过滤器实例只能在容器初始化时调用一次。
使用过滤器的目的是用来做一些过滤操作，比如：在过滤器中修改字符编码；在过滤器中修改HttpServletRequest的一些参数，包括：过滤低俗文字、危险字符等。

（2）拦截器：

只适用于springmvc框架，在实现上基于Java的反射机制，属于面向切面编程（AOP）的一种运用。
由于拦截器是基于web框架的调用，因此可以使用Spring的依赖注入（DI）进行一些业务操作，同时一个拦截器实例在一个controller生命周期之内可以多次调用。
只会拦截访问的控制器controller方法，如果访问的是静态资源是不会进行拦截的，例如 jsp，html,css,image 或者 js 。（但可以通过springmvc配置解决）

3、应用场景

1）日志记录：记录请求信息的日志，以便进行信息监控、信息统计、计算PV（Page View）等。

2）权限检查：如登录检测，进入处理器检测是否登录，如果没有直接返回到登录页面；

3）性能监控：有时候系统在某段时间莫名其妙的慢，可以通过拦截器在进入处理器之前记录开始时间，在处理完后记录结束时间，从而得到该请求的处理时间（如果有反向代理，如apache可以自动记录）；

4）通用行为：读取cookie得到用户信息并将用户对象放入请求，从而方便后续流程使用，还有如提取Locale、Theme信息等，只要是多个Controller中的处理方法都需要的，我们就可以使用拦截器实现。

5）OpenSessionInView：如Hibernate，在进入处理器打开Session，在完成后关闭Session。

二、详细属性

1、实现方式

第一种方式是要定义的（Interceptor）拦截类要实现了Spring的 HandlerInterceptor 接口

```
public class MyInterceptor1 implements HandlerInterceptor {
```

第二种方式是继承实现了HandlerInterceptor接口的类，比如Spring已经提供的实现了HandlerInterceptor接口的抽象 HandlerInterceptorAdapter

```
public class HandlerInterceptor1 extends HandlerInterceptorAdapter
```

2、HandlerInterceptor 接口的3个回调方法

```
public interface HandlerInterceptor {
   //处理前
      boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)  throws Exception;
    //处理后
      void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView)  throws Exception;
    //视图页面渲染后
      void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)  throws Exception;
}
```

1）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；

2）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。

3）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afterCompletion。

3、拦截器适配器：HandlerInterceptorAdapter

有时我们可能只需要实现三个回调方法中的某一个，如果实现HandlerInterceptor接口的话，三个方法必须实现，此时spring提供了一个HandlerInterceptorAdapter适配器（一种适配器设计模式的实现），允许我们只实现需要的回调方法。

三、springmvc配置拦截器（重点）

```
<!-- 配置拦截器 -->
<mvc:interceptors>
    <!-- 配置一个全局拦截器，拦截所有请求 -->
    <bean class="interceptor.TestInterceptor" />

     <!-- 配置第一个拦截器 -->
    <mvc:interceptor>

        <!-- 配置拦截器作用的方法的路径 -->
        <mvc:mapping path="/**" />

        <!-- 配置不需要拦截作用的路径 -->
```

```
        <mvc:exclude-mapping path="" />

        <!-- 定义第一个拦截器<mvc:interceptor>元素中，表示匹配指定路径的请求才进行拦截 -->
        <bean class="interceptor.Interceptor1" />
    </mvc:interceptor>

    <!-- 配置第二个拦截器 -->
    <mvc:interceptor>

        <!-- 配置拦截器作用的路径 -->
        <mvc:mapping path="/gotoTest" />

        <!-- 定义在<mvc: interceptor>元素中，表示匹配指定路径的请求才进行拦截 -->
        <bean class="interceptor.Interceptor2" />
    </mvc:interceptor>

</mvc:interceptors>
```

注意点：

1、path:

/　　：表示拦截所有路径下的所有文件夹及里面的子文件夹**

/*　　：表示拦截所有文件夹，不含子文件夹

/gotoTest　　：表示拦截所有以“/gotoTest”结尾的路径

2、配置顺序

<mvc：interceptor> 元素的子元素必须按照 <mvc：mapping.../>、<mvc:exclude-mapping.../>、<bean.../> 的顺序配置