SpringMVC小知识点：<https://blog.csdn.net/a745233700/article/details/80963758>

Spring和SpringMVC相关：<https://www.cnblogs.com/angelye/p/7506566.html>

SpringMVC过滤器：<https://blog.csdn.net/tobeabetterone/article/details/79310250>

SpringMvc拦截器：<https://www.cnblogs.com/jianjianyang/p/5009457.html>

# SpringMVC

## 什么是Spring MVC ？简单介绍下你对springMVC的理解?

Spring MVC是一个基于MVC架构的用来简化web应用程序开发的应用开发框架，它是Spring的一个模块,无需中间整合层来整合 ，它和Struts2一样都属于表现层的框架。在web模型中，MVC是一种很流行的框架，通过把Model，View，Controller分离，把较为复杂的web应用分成逻辑清晰的几部分，简化开发，减少出错，方便组内开发人员之间的配合。

## springMVC和struts2的区别有哪些?

（1）springmvc的入口是一个servlet即前端控制器（DispatchServlet），而struts2入口是一个filter过虑器（StrutsPrepareAndExecuteFilter）。

（2）springmvc是基于方法开发(一个url对应一个方法)，请求参数传递到方法的形参，可以设计为单例或多例(建议单例)，struts2是基于类开发，传递参数是通过类的属性，只能设计为多例。

（3）Struts采用值栈存储请求和响应的数据，通过OGNL存取数据，springmvc通过参数解析器是将request请求内容解析，并给方法形参赋值，将数据和视图封装成ModelAndView对象，最后又将ModelAndView中的模型数据通过reques域传输到页面。Jsp视图解析器默认使用jstl。

## SpringMVC的流程？

（1）用户发送请求至前端控制器DispatcherServlet；

（2） DispatcherServlet收到请求后，调用HandlerMapping处理器映射器，请求获取Handle；

（3）处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet；

（4）DispatcherServlet通过HandlerAdapter处理器适配器调用处理器；

（5）执行处理器(Handler，也叫后端控制器)；

（6）Handler执行完成返回ModelAndView；

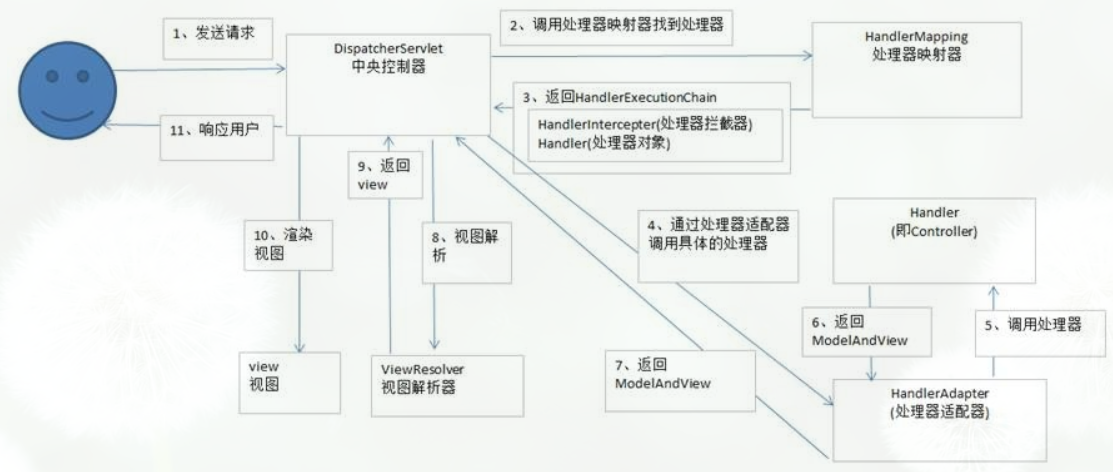
（7）HandlerAdapter将Handler执行结果ModelAndView返回给DispatcherServlet；

（8）DispatcherServlet将ModelAndView传给ViewResolver视图解析器进行解析；

（9）ViewResolver解析后返回具体View；

（10）DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）

（11）DispatcherServlet响应用户。



## Springmvc的优点:

（1）它是基于组件技术的。全部的应用对象,无论控制器和视图,还是业务对象之类的都是 java组件.并且和Spring提供的其他基础结构紧密集成.

（2）不依赖于Servlet API(目标虽是如此,但是在实现的时候确实是依赖于Servlet的)

（3）可以任意使用各种视图技术,而不仅仅局限于JSP

（4） 支持各种请求资源的映射策略

（5）它应是易于扩展的

## Spring MVC的主要组键？

（1）前端控制器 DispatcherServlet（不需要程序员开发）

作用：接收请求、响应结果 相当于转发器，有了DispatcherServlet 就减少了其它组件之间的耦合度。

（2）处理器映射器HandlerMapping（不需要程序员开发）

作用：根据请求的URL来查找Handler

（3）处理器适配器HandlerAdapter

注意：在编写Handler的时候要按照HandlerAdapter要求的规则去编写，这样适配器HandlerAdapter才可以正确的去执行Handler。

（4）处理器Handler（需要程序员开发）

（5）视图解析器 ViewResolver（不需要程序员开发）

作用：进行视图的解析 根据视图逻辑名解析成真正的视图（view）

（6）视图View（需要程序员开发jsp）

View是一个接口， 它的实现类支持不同的视图类型（jsp，freemarker，pdf等等）

## SpringMVC怎么样设定重定向和转发的？

（1）在返回值前面加"forward:"就可以让结果转发,譬如"forward:user.do?name=method4"

（2）在返回值前面加"redirect:"就可以让返回值重定向,譬如"redirect:http://www.baidu.com"

## SpringMvc怎么和AJAX相互调用的？

通过Jackson框架就可以把Java里面的对象直接转化成Js可以识别的Json对象。具体步骤如下 ：

（1）加入Jackson.jar

（2）在配置文件中配置json的映射

（3）在接受Ajax方法里面可以直接返回Object,List等,但方法前面要加上@ResponseBody注解。

## Spring MVC的异常处理 ？

答：可以将异常抛给Spring框架，由Spring框架来处理；我们只需要配置简单的异常处理器，在异常处理器中添视图页面即可。

## SpringMvc的控制器是不是单例模式,有什么问题,怎么解决？

答：是单例模式,所以在多线程访问的时候有线程安全问题,不要用同步,会影响性能的,解决方案是在控制器里面不能写字段。

## 拦截器简介

Spring web MVC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。

## 1.2 常见应用场景

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，只要是多个处理器都需要的即可使用拦截器实现。

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

…………本质也是AOP（面向切面编程），也就是说符合横切关注点的所有功能都可以放入拦截器实现。

## 回调方法

**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（如我们上一章的Controller实现）；

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的拦截器的afterCompletion。

## 拦截器适配器

有时候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，不管你需不需要，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。

handlerInterceptor接口实现

DispatcherServlet处理流程：DispatcherServlet处理请求时会构造一个Excecution Chain，即(可能多个)拦截器和真正处理请求的Handler即Interceptor是链式调用的。

**preHandle:** 在执行Handler之前进行，即Controller方法调用之前执行，主要进行初始化操作。

**postHandle:** 在执行Handler之后进行，即Controller 方法调用之后执行，主要对ModelAndView对象进行操作。

**afterCompletion:** 在整个请求结束之后，即渲染对应的视图之后执行, 主要进行资源清理工作。

## WebRequestInterceptor接口

WebRequestInterceptor和HandlerInterceptor一样，也有三个方法，preHandle，postHandle，afterCompletion，不同的是preHandle没有返回值,而且WebRequestInterceptor的三个方法的参数都是WebRequest,WebRequest和HttpServletRequest用法基本一样。

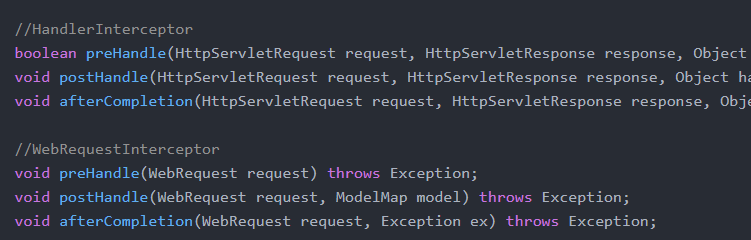
**SCOPE\_REQUEST ：**它的值是0 ，代表只有在request 中可以访问。

**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。

## HandlerInterceptor与WebRequestInterceptor的异同

两个接口都可用于Contrller层请求拦截，接口中定义的方法作用也是一样的。



WebRequestInterceptor间接实现了HandlerInterceptor，只是他们之间使用WebRequestHandlerInterceptorAdapter适配器类联系。

## 不同点

1.WebRequestInterceptor的入参WebRequest是包装了HttpServletRequest 和HttpServletResponse的，通过WebRequest获取Request中的信息更简便。

2.WebRequestInterceptor的preHandle是没有返回值的，说明该方法中的逻辑并不影响后续的方法执行，所以这个接口实现就是为了获取Request中的信息，或者预设一些参数供后续流程使用。

3.HandlerInterceptor的功能更强大也更基础，可以在preHandle方法中就直接拒绝请求进入controller方法。

## 使用场景

这个在上条已经说了，如果想更方便获取HttpServletRequest的信息就使用WebRequestInterceptor，当然这些HandlerInterceptor都能做，只不过要多写点代码

## 如何配置

配置类继承WebMvcConfigurationSupport或WebMvcConfigurerAdapter类，重写addInterceptors，InterceptorRegistry实例就可以直接添加。

顺便说下继承WebMvcConfigurationSupport或WebMvcConfigurerAdapter的区别，继承WebMvcConfigurationSupport不需要声明@EnableWebMvc注解，继承WebMvcConfigurerAdapter需要

Spring 过滤器Filter

## 和拦截器的区别

过滤器和拦截器最大的区别在于：过滤器是servlet的规范规定的，只能用于过滤请求，而interceptor是Spring里面基于切面编程的一种实现。

过滤器作用于请求到达servlet之前，在spring中也就是在dispacherServlet之前。而拦截器最早只能作用于请求到达servlet之后。

基于这个特点，过滤器经常被用于处理请求的编码格式，session，日志等。

业务逻辑大多实现在拦截器中。

## 通常实现拦截器的方式：

实现javax.servlet.Filter接口，Filter是servlet包的定义的接口，现有的这种实现方法有CompositeFilter

继承抽象类GenericFilterBean 这个类实现了Filter接口。现有的实现有DelegatingFilterProxy

继承抽象类OncePerRequestFilter 这个类是GenericFilterBean 的子类。现有的实现类有：CharacterEncodingFilter 、HiddenHttpMethodFilter 、HttpPutFormContentFilter、RequestContextFilter和ShallowEtagHeaderFilter

继承抽象类AbstractRequestLoggingFilter，该类为OncePerRequestFilter的直接子类，这一类过滤器包括CommonsRequestLoggingFilter、Log4jNestedDiagnosticContextFilter和ServletContextRequestLoggingFilter。

## GenericFilterBean

从类结构图可以看到，这个类是Spring中直接实现了servlet的Filter的抽象类。所有的Filter都是基于这个类来实现的。

实现的接口有：

Filter：过滤器的实现

BeanNameAware：setBeanName方法，便于Bean管理器生成Bean

EnvironmentAware：用于指明Bean的运行环境

EnvironmentCapable：用于获取bean的运行环境

ServletContextAware：实现该接口的setServletContextAware方法，指明ServletContext

InitializingBean：实现该接口的afterPropertiesSet方法，在Bean初始化时设置属性

DisposableBean ：实现该接口的destroy方法，用于回收资源。

先后执行的方法为：init-doFilter-destroy。

filter可以在request执行之前进行处理，也可以在request执行之后进行处理。只需要在调用filterChain的doFilter方法之前和之后加入自己的处理逻辑就可以了。

没有实现Filter接口的doFilter方法，是在子类OncePerRequestFilter 和DelegatingFilterProxy中实现的。接下来看它的两个子类。

## OncePerRequestFilter

**OncePerRequestFilter** 依然是一个抽象类。实现了doFilter方法。但是真正要实现的逻辑在**doFilterInternal（）** 方法中，这个方法由子类自己实现。所以如果要通过实现**OncePerRequestFilter** 类来实现过滤器，需要实现doFilterInternal() 方法

## DelegatingFilterProxy

这个类是具体的实现类，不是抽象的。

因为Filter是servlet容器规范的，会在servlet启动的时候就初始化，这个时候Spring的context还没有被初始化。当我们需要在Filter里面初始化一些spring的bean或者spring的特性时默认的Filter实现就不行。

比如我们如果要在Filter里注入一个用于鉴权的Service，因为在Filter初始化时spring容器还没有启动，因此这样不能实现我们的需求，这种情况下DelegatingFilterProxy可以实现我们的需求，它可以作为Filter在servlet容器中被启动，在使用的时候再去spring容器中获取被代理的对象。这样被代理对象的初始化就可以和普通spring的bean一样使用一些复杂的依赖注入等。

从类名可以看出，是一个委派过滤器代理。它作为一个过滤器没有实现任何逻辑功能。只是执行它所代理的类的方法。