码农求职小助手：SpringMVC高频面试题

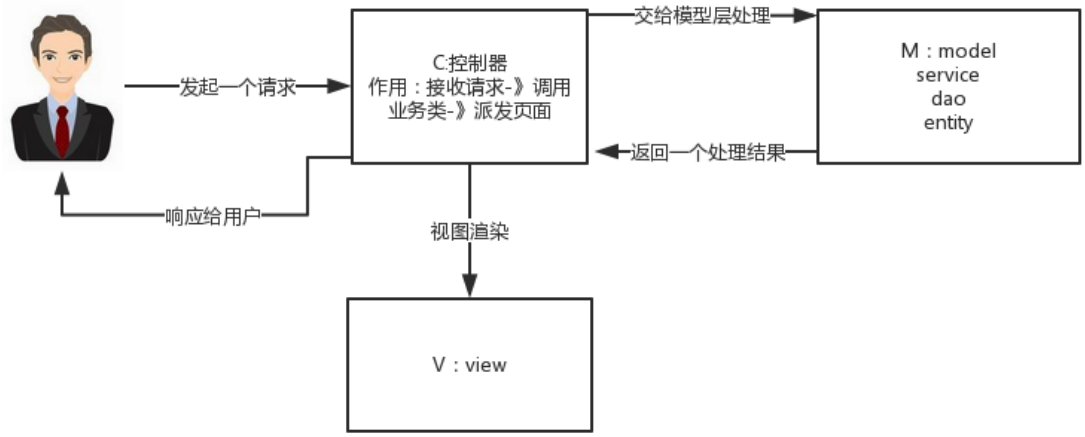
笔记本： 12-Spring-MVC
创建时间： 2019/9/7 21:52 **更新时间：** 2019/9/7 21:53
作者： pc941206@163.com

更多资料请关注微信公众号：**码农求职小助手**

一、什么是 MVC 模式

MVC 是一种设计模式。

MVC 的原理图如下：



1. SpringMVC 简单介绍

SpringMVC 框架是以请求为驱动，围绕 Servlet 设计，将请求发给控制器，然后通过模型对象，分派器来展示请求结果视图。其中**核心类是 DispatcherServlet**，它是一个 Servlet，顶层是实现的 Servlet 接口。

2. SpringMVC 使用

需要在 web.xml 中配置 DispatcherServlet 。并且需要配置 Spring 监听器 ContextLoaderListener。

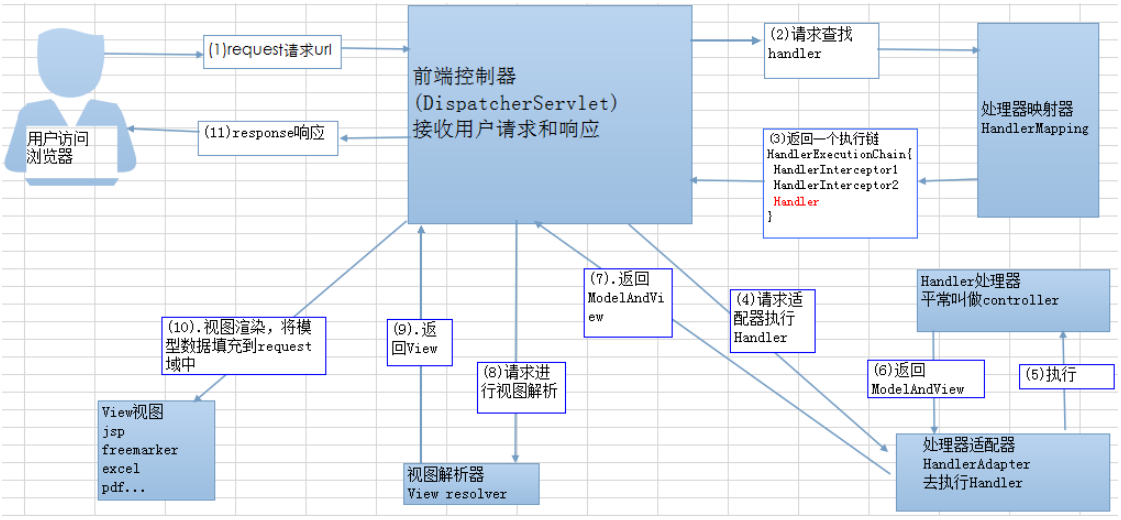
```
<listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>
<servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <!-- 如果不设置init-param标签，则必须在/WEB-INF/下创建xxx-servlet.xml
文件，其中xxx是servlet-name中配置的名称。 -->
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>classpath:spring/springmvc-servlet.xml</param-
value>
        </init-param>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>/</url-pattern>
</servlet-mapping>
```

二、SpringMVC 工作原理

简单来说：

客户端发送请求-> 前端控制器 DispatcherServlet 接受客户端请求 -> 找到处理器映射 HandlerMapping 解析请求对应的 Handler -> HandlerAdapter 会根据 Handler 来调用真正的处理器来处理请求，并处理相应的业务逻辑 -> 处理器返回一个模型视图 ModelAndView -> 视图解析器进行解析 -> 返回一个视图对象 -> 前端控制器 DispatcherServlet 渲染数据（Moder）-> 将得到视图对象返回给用户。

- Spring MVC 运行流程图：



上图的一个笔误的小问题：Spring MVC 的入口函数也就是前端控制器 DispatcherServlet 的作用是接收请求，响应结果。

- **SpringMVC 运行流程描述：**

1、用户向服务器发送请求，请求被 Spring 前端控制Servelt DispatcherServlet 捕获；

2、DispatcherServlet 对请求 URL 进行解析，得到请求资源标识符（URI）。然后根据该 URI，调用 HandlerMapping 获得该 Handler 配置的所有相关的对象（包括 Handler 对象以及 Handler 对象对应的拦截器），最后以 HandlerExecutionChain 对象的形式返回；

3、DispatcherServlet 根据获得的 Handler，选择一个合适的HandlerAdapter；（附注：如果成功获得 HandlerAdapter 后，此时将开始执行拦截器的 preHandler(...)方法）

4、提取 Request 中的模型数据，填充 Handler 入参，开始执行Handler（Controller)。在填充 Handler 的入参过程中，根据你的配置，Spring 将帮你做一些额外的工作：

- HttpMessageConveter： 将请求消息（如 Json、xm l等数据）转换成一个对象，将对象转换为指定的响应信息；
- 数据转换：对请求消息进行数据转换。如 String 转换成Integer、Double等；
- 数据格式化：对请求消息进行数据格式化。 如将字符串转换成格式化数字或格式化日期等；
- 数据验证： 验证数据的有效性（长度、格式等），验证结果存储到 BindingResult 或 Error 中;

5、Handler 执行完成后，向 DispatcherServlet 返回一个 ModelAndView 对象；

6、根据返回的 ModelAndView，选择一个适合的 ViewResolver（必须是已经注册到 Spring 容器中的 ViewResolver)返回给DispatcherServlet；

7、ViewResolver 结合 Model 和 View，来渲染视图；

8、将渲染结果返回给客户端。

三、SpringMVC 重要组件说明

1、**前端控制器 DispatcherServlet**（不需要工程师开发），由框架提供（重要）

作用：**Spring MVC 的入口函数。接收请求，响应结果，相当于转发器，中央处理器。**有了 DispatcherServlet 减少了其它组件之间的耦合度。用户请求到达前端控制器，它就相当于 mvc 模式中的 c，**DispatcherServlet 是整个流程控制的中心，由它调用其它组件处理用户的请求，DispatcherServlet 的存在降低了组件之间的耦合性。**

2、**处理器映射器 HandlerMapping**（不需要工程师开发），由框架提供

作用：根据请求的 url 查找 Handler。HandlerMapping 负责根据用户请求找到 Handler 即处理器（Controller），SpringMVC 提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

3、**处理器适配器 HandlerAdapter**

作用：按照特定规则（HandlerAdapter 要求的规则）去执行 Handler。 通过 HandlerAdapter 对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

4、**处理器 Handler**（需要工程师开发）

注意：编写 Handler 时按照 HandlerAdapter 的要求去做，这样适配器才可以去正确执行 Handler。 Handler 是继 DispatcherServlet 前端控制器的后端控制器，在 DispatcherServlet 的控制下 Handler 对具体的用户请求进行处理。 由于 Handler 涉及到具体的用户业务请求，所以一般情况需要工程师根据业务需求开发 Handler。

5、**视图解析器 View resolver**（不需要工程师开发），由框架提供

作用：**进行视图解析，根据逻辑视图名解析成真正的视图（view）。**View Resolver 负责将处理结果生成 View 视图，View Resolver 首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成 View 视图对象，最后对 View 进行渲染将处理结果通过页面展示给用户。 SpringMVC 框架提供了很多的 View 视图类型，包括：jstlView、freemarkerView、pdfView 等。 一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由工程师根据业务需求开发具体的页面。

6、**视图 View**（需要工程师开发）

View 是一个接口，实现类支持不同的 View 类型（jsp、freemarker、pdf...）

注意：处理器 Handler（也就是我们平常说的 Controller 控制器）以及视图层 View 都是需要我们自己手动开发的。其他的一些组件比如：前端控制器 DispatcherServlet、处理器映射器 HandlerMapping、处理器适配器 HandlerAdapter 等等都是框架提供给我们的，不需要自己手动开发。

四、DispatcherServlet 的源码分析

- **首先看下源码**：

```
package org.springframework.web.servlet;

@SuppressWarnings("serial")
public class DispatcherServlet extends FrameworkServlet {

	public static final String MULTIPART_RESOLVER_BEAN_NAME =
"multipartResolver";
	public static final String LOCALE_RESOLVER_BEAN_NAME =
"localeResolver";
	public static final String THEME_RESOLVER_BEAN_NAME =
"themeResolver";
	public static final String HANDLER_MAPPING_BEAN_NAME =
"handlerMapping";
	public static final String HANDLER_ADAPTER_BEAN_NAME =
"handlerAdapter";
	public static final String HANDLER_EXCEPTION_RESOLVER_BEAN_NAME =
"handlerExceptionResolver";
	public static final String REQUEST_TO_VIEW_NAME_TRANSLATOR_BEAN_NAME
= "viewNameTranslator";
	public static final String VIEW_RESOLVER_BEAN_NAME = "viewResolver";
	public static final String FLASH_MAP_MANAGER_BEAN_NAME =
"flashMapManager";
	public static final String WEB_APPLICATION_CONTEXT_ATTRIBUTE =
DispatcherServlet.class.getName() + ".CONTEXT";
	public static final String LOCALE_RESOLVER_ATTRIBUTE =
DispatcherServlet.class.getName() + ".LOCALE_RESOLVER";
	public static final String THEME_RESOLVER_ATTRIBUTE =
DispatcherServlet.class.getName() + ".THEME_RESOLVER";
	public static final String THEME_SOURCE_ATTRIBUTE =
DispatcherServlet.class.getName() + ".THEME_SOURCE";
	public static final String INPUT_FLASH_MAP_ATTRIBUTE =
DispatcherServlet.class.getName() + ".INPUT_FLASH_MAP";
	public static final String OUTPUT_FLASH_MAP_ATTRIBUTE =
DispatcherServlet.class.getName() + ".OUTPUT_FLASH_MAP";
	public static final String FLASH_MAP_MANAGER_ATTRIBUTE =
DispatcherServlet.class.getName() + ".FLASH_MAP_MANAGER";
	public static final String EXCEPTION_ATTRIBUTE =
DispatcherServlet.class.getName() + ".EXCEPTION";
	public static final String PAGE_NOT_FOUND_LOG_CATEGORY =
"org.springframework.web.servlet.PageNotFound";
	private static final String DEFAULT_STRATEGIES_PATH =
"DispatcherServlet.properties";
	protected static final Log pageNotFoundLogger =
LogFactory.getLog(PAGE_NOT_FOUND_LOG_CATEGORY);
```

```
        private static final Properties defaultStrategies;
        static {
                try {
                        ClassPathResource resource = new
ClassPathResource(DEFAULT_STRATEGIES_PATH, DispatcherServlet.class);
                        defaultStrategies =
PropertiesLoaderUtils.loadProperties(resource);
                }
                catch (IOException ex) {
                        throw new IllegalStateException("Could not load
'DispatcherServlet.properties': " + ex.getMessage());
                }
        }

        /** Detect all HandlerMappings or just expect "handlerMapping" bean?
*/
        private boolean detectAllHandlerMappings = true;

        /** Detect all HandlerAdapters or just expect "handlerAdapter" bean?
*/
        private boolean detectAllHandlerAdapters = true;

        /** Detect all HandlerExceptionResolvers or just expect
"handlerExceptionResolver" bean? */
        private boolean detectAllHandlerExceptionResolvers = true;

        /** Detect all ViewResolvers or just expect "viewResolver" bean? */
        private boolean detectAllViewResolvers = true;

        /** Throw a NoHandlerFoundException if no Handler was found to
process this request? **/
        private boolean throwExceptionIfNoHandlerFound = false;

        /** Perform cleanup of request attributes after include request? */
        private boolean cleanupAfterInclude = true;

        /** MultipartResolver used by this servlet */
        private MultipartResolver multipartResolver;

        /** LocaleResolver used by this servlet */
        private LocaleResolver localeResolver;

        /** ThemeResolver used by this servlet */
        private ThemeResolver themeResolver;
```

```
	/** List of HandlerMappings used by this servlet */
	private List<HandlerMapping> handlerMappings;

	/** List of HandlerAdapters used by this servlet */
	private List<HandlerAdapter> handlerAdapters;

	/** List of HandlerExceptionResolvers used by this servlet */
	private List<HandlerExceptionResolver> handlerExceptionResolvers;

	/** RequestToViewNameTranslator used by this servlet */
	private RequestToViewNameTranslator viewNameTranslator;

	private FlashMapManager flashMapManager;

	/** List of ViewResolvers used by this servlet */
	private List<ViewResolver> viewResolvers;

	public DispatcherServlet() {
		super();
	}

	public DispatcherServlet(WebApplicationContext webApplicationContext)
{
		super(webApplicationContext);
	}
	@Override
	protected void onRefresh(ApplicationContext context) {
		initStrategies(context);
	}

	protected void initStrategies(ApplicationContext context) {
		initMultipartResolver(context);
		initLocaleResolver(context);
		initThemeResolver(context);
		initHandlerMappings(context);
		initHandlerAdapters(context);
		initHandlerExceptionResolvers(context);
		initRequestToViewNameTranslator(context);
		initViewResolvers(context);
		initFlashMapManager(context);
	}
}
```

- **DispatcherServlet 类中的属性 beans：**

1、HandlerMapping：用于 handlers 映射请求和一系列的对于拦截器的前处理和后处理，大部分用@Controller 注解；

2、HandlerAdapter：帮助 DispatcherServlet 处理映射请求程序的适配器，而不用考虑实际调用的是哪个处理程序；

3、ViewResolver：根据实际配置解析实际的 View 类型；

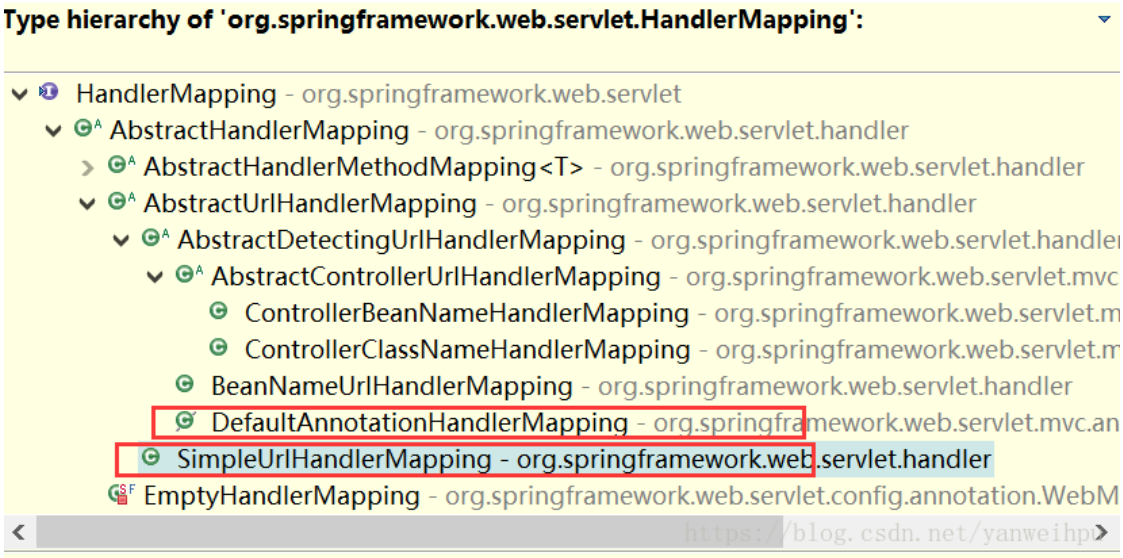
4、ThemeResolver：解决 Web 应用程序可以使用的主题，例如：提供个性化布局；

5、MultipartResolver：解析多部分请求，以支持从 HTML 表单上传文件；

6、FlashMapManager：存储并检索可用于将一个请求属性传递到另一个请求的 input 和 output 的FlashMap，通常用于重定向。 ‘

在 Web MVC 框架中，每个 DispatcherServlet 都拥自己的 WebApplicationContext，它继承了 ApplicationContext。WebApplicationContext 包含了其上下文和 Servlet 实例之间共享的所有的基础框架 beans。

- **HandlerMapping**

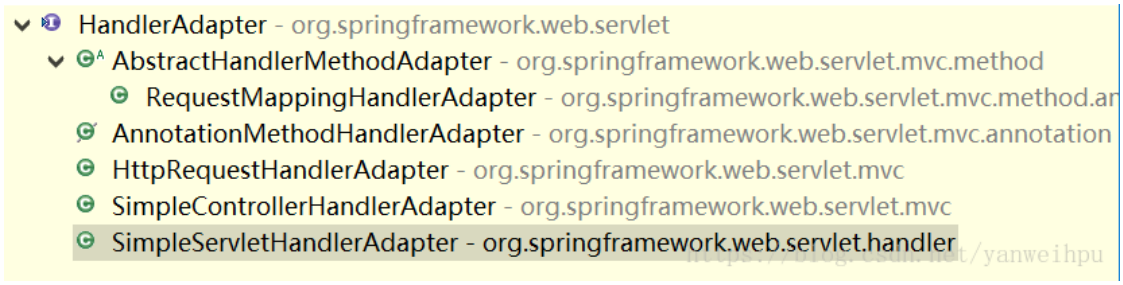


HandlerMapping 接口处理请求的映射，HandlerMapping 接口的实现类：

1、SimpleUrlHandlerMapping 类通过配置文件把 URL 映射到 Controller 类；

2、DefaultAnnotationHandlerMapping 类通过注解把 URL 映射到 Controller 类。

- **HandlerAdapter**

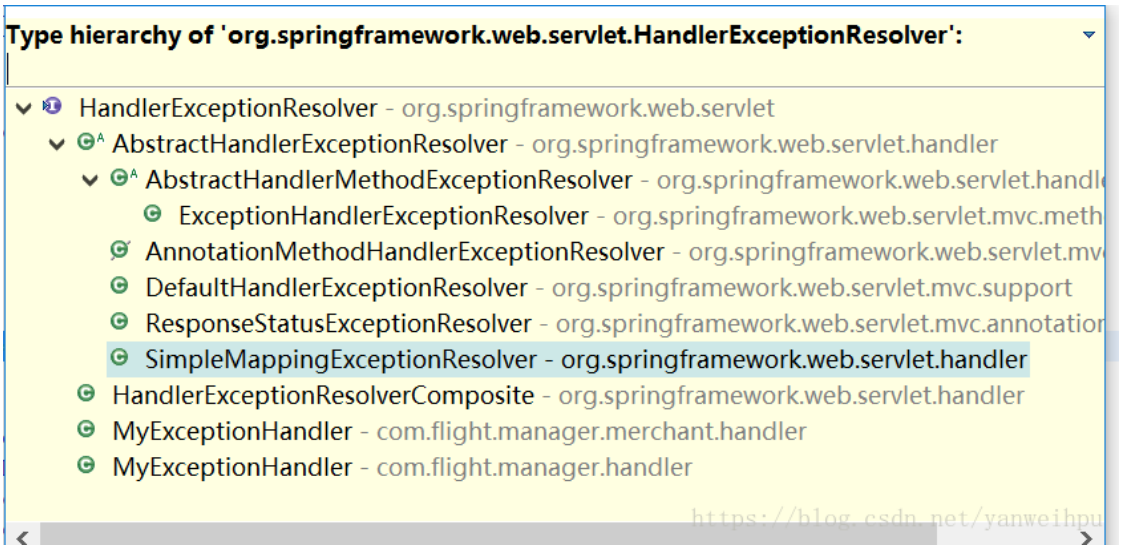


HandlerAdapter接口：处理请求映射

SimpleServletHandlerAdapter：通过配置文件，把请求 URL 映射到 Controller 类的方法上；

AnnotationMethodHandlerAdapter：通过注解，把请求 URL 映射到 Controller 类的方法上。

- **HandlerExceptionResolver**

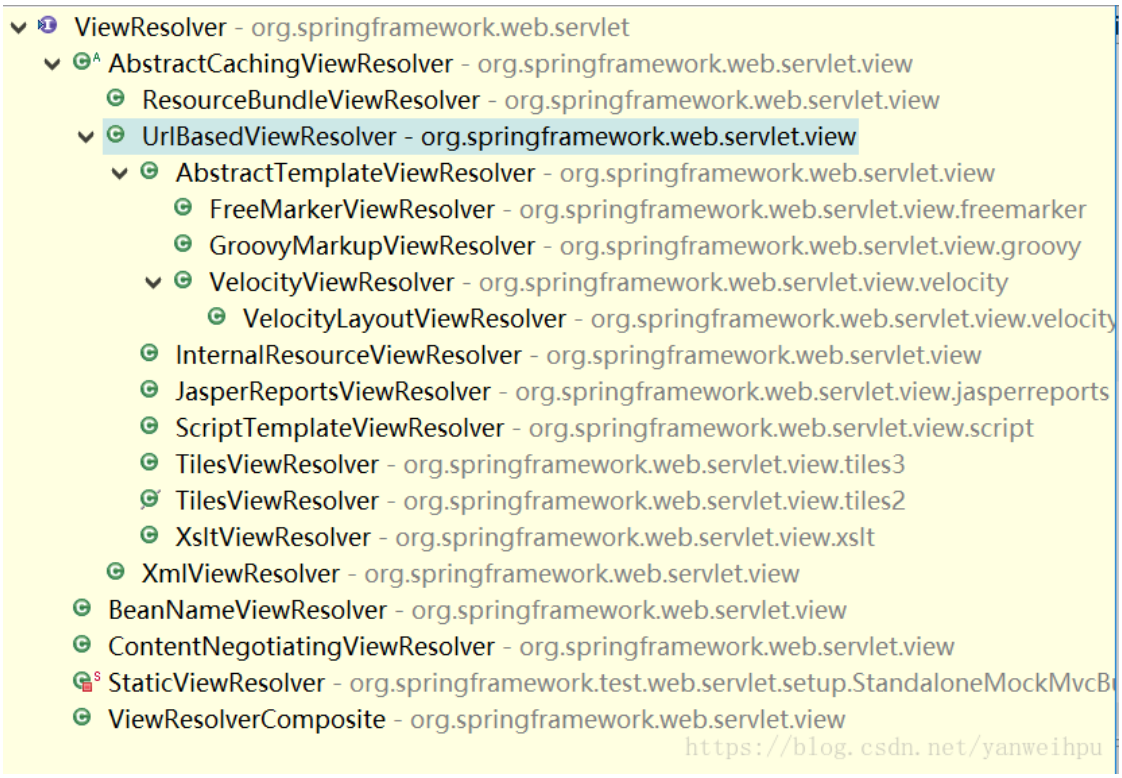


HandlerExceptionResolver接口：异常处理接口。

1、SimpleMappingExceptionResolver：通过配置文件进行异常处理；

2、AnnotationMethodHandlerExceptionResolver：通过注解进行异常处理。

- **ViewResolver**



ViewResolver 接口解析 View 视图。

1、UrlBasedViewResolver：通过配置文件，把一个视图名交给到一个 View 来处理。

五、Spring MVC 的注解

1、@RequestMapping 的作用是什么?

RequestMapping 是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。RequestMapping 注解有六个属性，下面我们把它分成三类进行说明。

- **value， method：**

value：指定请求的实际地址，指定的地址可以是 URI Template 模式（后面将会说明）；

method：指定请求的method类型， GET、POST、PUT、DELETE等；

- **consumes，produces：**

consumes：指定处理请求的提交内容类型（Content-Type），例如 application/json, text/html；

produces：指定返回的内容类型，仅当 request 请求头中的（Accept）类型中包含该指定类型才返回；

- **params，header：**

sparams： 指定 request 中必须包含某些参数值是，才让该方法处理。

headers：指定 request 中必须包含某些指定的 header 值，才能让该方法处理请求。

六、其他面试题

1、SpringMVC 的优点?

1、可以支持各种视图技术，而不仅仅局限于 JSP；

2、与 Spring 框架集成（如 IOC 容器、AOP 等）；

3、清晰的角色分配：前端控制器（dispatcherServlet），请求到处理器映射（handlerMapping)，处理器适配器（HandlerAdapter)，视图解析器（ViewResolver）；

4、支持各种请求资源的映射策略。

2、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、Struts2 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 request 域传输到页面。Jsp 视图解析器默认使用 jstl。

3、SpringMVC 怎么样设定重定向和转发的?

1、转发：在返回值前面加 "forward:"，譬如："forward:user.do?name=method4"

2、重定向：在返回值前面加 "redirect:"，譬如："redirect:http://www.baidu.com"

4、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、在接收 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。

5、如何解决 POST 请求中文乱码问题，GET 的又如何处理呢?

1、解决 post 请求乱码问题：在 web.xml 中配置一个 CharacterEncodingFilter 过滤器，设置成 utf-8；

2、get 请求中文参数出现乱码解决方法有两个：

（1）修改 tomcat 配置文件添加编码与工程编码一致，如下：

```
<ConnectorURIEncoding="utf-
8"  connectionTimeout="20000"  port="8080"  protocol="HTTP/1.1"  redirectPort="8
```

（2）对参数进行重新编码：

```
String  userName
=  new  String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")
```

ISO8859-1 是 tomcat 默认编码，需要将 tomcat 编码后的内容按 utf-8 编码。

6、SpringMVC 的控制器是不是单例模式，如果是,有什么问题,怎么解决?

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，解决方案是在控制器里面不能写字段。

7、SpringMVC 常用的注解有哪些?

1、**@RequestMapping**：用于处理请求 url 映射的注解，可用于类或方法上。用于类上，则表示类中的所有响应请求的方法都是以该地址作为父路径；

2、**@RequestBody**：注解实现接收 http 请求的 json 数据，将 json 转换为 java 对象；

3、**@ResponseBody**：注解实现将 conreoller 方法返回对象转化为 json 对象响应给客户。

8、如果在拦截请求中，我想拦截 get 方式提交的方法，怎么配置?

可以在 @RequestMapping 注解里面加上 method=RequestMethod.GET。

9、怎样在方法里面得到 Request 或者 Session?

直接在方法的形参中声明 request，SpringMVC 就自动把 request 对象传入。

10、如果前台有很多个参数传入，并且这些参数都是一个对象的，那么怎么样快速得到这个对象?

直接在方法中声明这个对象，SpringMVC 就自动会把属性赋值到这个对象里面。

11、SpringMVC 用什么对象从后台向前台传递数据的?

通过 **ModelMap** 对象，可以在这个对象里面调用 put 方法，把对象加到里面，前台就可以通过 el 表达式拿到。

12、SpringMVC 里面拦截器是怎么写的?

有两种写法,一种是实现 HandlerInterceptor 接口；另外一种是继承适配器类，接着在接口方法当中，实现处理逻辑，然后在 SpringMVC 的配置文件中配置拦截器即可。

13、注解原理

注解本质是一个继承了 Annotation 的特殊接口，其具体实现类是 Java 运行时生成的动态代理类。我们通过反射获取注解时，返回的是 Java 运行时生成的动态代理对象。通过代理对象调用自定义注解的方法，会最终调用 AnnotationInvocationHandler 的 invoke 方法。该方法会从 memberValues 这个 Map 中索引出对应的值。而 memberValues 的来源是 Java 常量池。