什么是 MVC 模式

    MVC 是一种设计模式.

MVC 的原理图如下：

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。

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>

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开处理请求，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

如下图所示：

上图有个小问题：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。

流程说明：

（1）客户端（浏览器）发送请求，直接请求到 DispatcherServlet。

（2）DispatcherServlet 根据请求信息调用 HandlerMapping，解析请求对应的 Handler。

（3）解析到对应的 Handler（也就是我们平常说的 Controller 控制器）后，开始由 HandlerAdapter 适配器处理。

（4）HandlerAdapter 会根据 Handler 来调用真正的处理器开处理请求，并处理相应的业务逻辑。

（5）处理器处理完业务后，会返回一个 ModelAndView 对象，Model 是返回的数据对象，View 是个逻辑上的 View。

（6）ViewResolver 会根据逻辑 View 查找实际的 View。

（7）DispaterServlet 把返回的 Model 传给 View（视图渲染）。

（8）把 View 返回给请求者（浏览器）

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。

DispatcherServlet详细解析

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：

    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。

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

    ViewResolver：根据实际配置解析实际的View类型。

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

    MultipartResolver：解析多部分请求，以支持从HTML表单上传文件。-

    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：

    SimpleUrlHandlerMapping类通过配置文件把URL映射到Controller类。

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

HandlerAdapter

HandlerAdapter接口-处理请求映射

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

    SimpleMappingExceptionResolver通过配置文件进行异常处理。

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

ViewResolver

ViewResolver接口解析View视图。

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

总结

    SpringMVC是一种基于Java，实现了Web MVC设计模式，请求驱动类型的轻量级Web框架，框架的目的就是帮助我们简化开发。前端控制器是DispatcherServlet；应用控制器其实拆为处理映射器(Handler Mapping)进行处理器管理和视图解析器(View Resolver)进行视图管理；页面控制器/动作/处理器为Controller接口（仅包含ModelAndView handleRequest(request, response) 方法）的实现（也可以是任何的POJO类）；支持本地化（Locale）解析、主题（Theme）解析及文件上传等；提供了非常灵活的数据验证、格式化和数据绑定机制；提供了强大的约定大于配置（惯例优先原则）的契约式编程支持。

————————————————

版权声明：本文为CSDN博主「chorsion」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/bug\_remover/article/details/86680265