1. SpringMVC 的入门案例

1.1 创建一个maven工程并导入依赖

maven项目pom配置:

```
<dependencies>

    <dependency>
        <!-- Junit测试 -->
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>

    <!-- jstl -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

    <!-- 添加Spring包 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>4.3.7.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.7.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
        <version>4.3.7.RELEASE</version>
```

```
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>4.3.7.RELEASE</version>
    </dependency>

    <!-- 为了方便进行单元测试，添加spring-test包 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>4.3.7.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aspects</artifactId>
        <version>4.3.7.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-expression</artifactId>
        <version>4.3.7.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>4.3.7.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.8.5</version>
    </dependency>

</dependencies>
```

1.2 配置核心控制器（前端控制器）

配置在 **web.xml** 中配置 **DispatcherServlet** 核心控制器(Servlet)

负责接受用户的请求，程序的入口。

```
<!--配置SpringMvc的核心控制器-->
<servlet>
```

```
    <servlet-name>DispatcherServlet</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</s
ervlet-class>
    <!-- 配置DispatcherServlet的初始化参数：设置SpringMVC配置文
件的路径和文件名称 -->
    <init-param>
        <!--配置的这个参数是DispatcherServlet类中的一个属性，可
以去源码中看一下-->
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:springmvc.xml</param-value>
    </init-param>
    <!-- 配置 servlet 的对象的创建时间点：应用加载时创建。取值只能
是非 0 正整数，表示启动顺序 -->
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <!--表示所有的请求都会纳入到SrpingMVC中，便于ResutFul风格
        但是会导致静态资源文件被拦截。
    -->
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

1.3 创建 spring mvc 的配置文件

springmvc.xml文件：

```
<?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/conte
xt"

xsi:schemaLocation="http://www.springframework.org/schema/
beans http://www.springframework.org/schema/beans/spring-
beans.xsd http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-
context.xsd">

    <!--扫描控制层的包，创建由@Controller注解注释的类的Bean实例--
>
    <context:component-scan base-
package="com.bruceliu.controller"/>

    <!--视图解析器-->
```

```
    <bean
class="org.springframework.web.servlet.view.InternalResour
ceViewResolver">
        <!--配置视图的前缀
            在Tomcat目录结构中WEB-INF是受保护的目录，存放在该目录
中的文件，通过浏览器地址不可以直接访问，访问不到
        -->
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <!--配置视图的后缀-->
        <property name="suffix" value=".jsp"/>
    </bean>

</beans>
```

1.4 编写控制器并使用注解配置

```
//声明Bean对象，为一个控制器组件
@Controller
public class HelloWorld {

    /**
     * 映射请求的名称：用于客户端请求；类似Struts2中action映射配置
的action名称
     * 1. 使用 @RequestMapping 注解来映射请求的 URL
     * 2. 返回值会通过视图解析器解析为实际的物理视图，对于
InternalResourceViewResolver 视图解析器，
     * 会做如下的解析:
     *                  通过 prefix + returnVal + suffix 这
样的方式得到实际的物理视图，然后做转发操作.
     *                  /WEB-INF/jsp/success.jsp
     */
    @RequestMapping("/helloworld")
    public String hellworld() {
        System.out.println("helloworld");
        return "success";  //默认是请求转发：   结果如何跳转呢？
需要配置映射解析器    /WEB-INF/jsp/success.jsp
    }
}
```

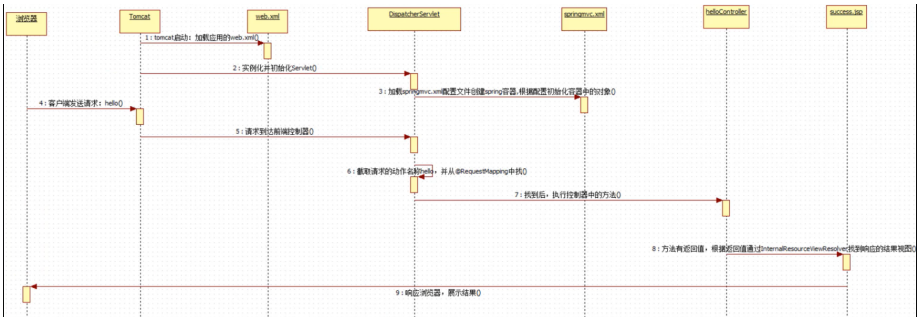
@Controller 来源于@Component，标示为控制层，用于加在类上。
@RequestMapping("/helloworld") 该方法对应的uri; 控制器类的方法返回字符串类型非常常见,返回字符串,代表根据返回的字符串找到对应的视图! 根据springmvc配置文件中视图解析器(InternalResourceViewResolver) 配置的视图文件的前缀和后缀! helloworld()方法返回 "helloworld" 会找到 WEB-INF/jsp/success.jsp文件!

1.5 测试Spring MVC

发布项目,通过浏览器,访问 当前项目对应地址+ /helloworld即可 http://localhost:8080/springmvc-demo1/helloworld

2. 入门案例的执行过程及原理分析

2.1 案例的执行过程



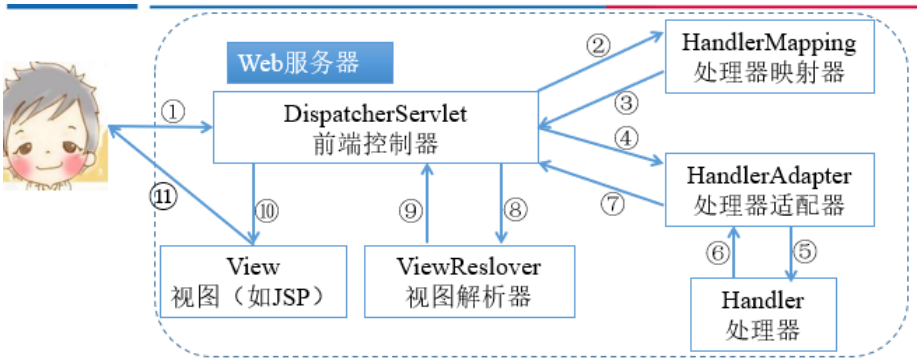
1、服务器启动，应用被加载。读取到 web.xml 中的配置创建 spring 容器并且初始化容器中的对象。从入门案例中可以看到的是：HelloController 和 InternalResourceViewResolver，但是远不 止这些。

2、浏览器发送请求，被DispatherServlet 捕获，该 Servlet 并不处理请求，而是把请求转发出去。转发 的路径是根据请求 URL，匹配@RequestMapping 中的内容。 3、匹配到了后，执行对应方法。该方法有一个返回值。

4、根据方法的返回值，借助InternalResourceViewResolver 找到对应的结果视图。

5、渲染结果视图，响应浏览器。

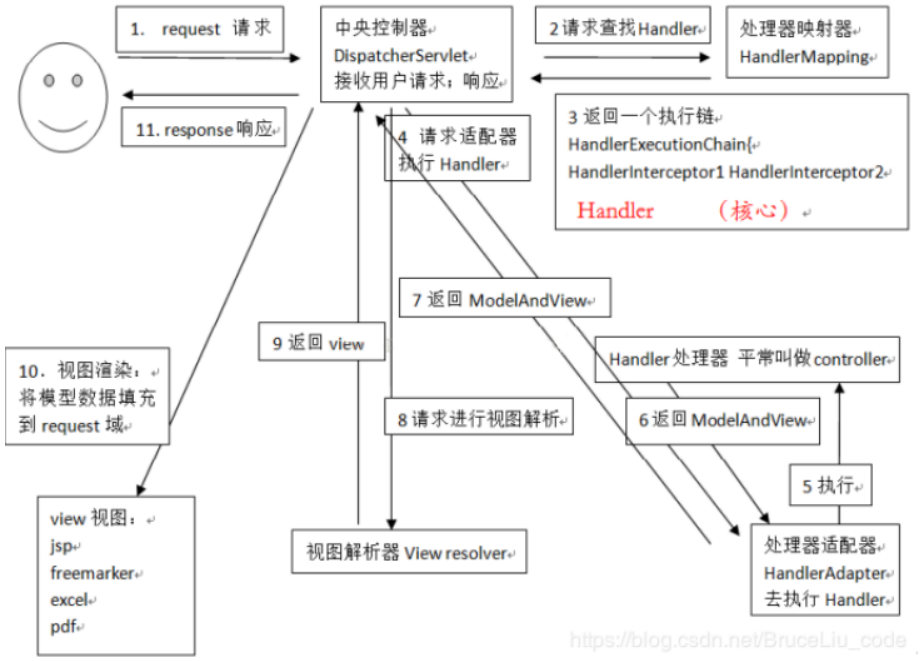
2.2 Spring MVC的工作流程（教材）



1. 用户通过浏览器向服务器发送请求，请求会被Spring MVC的前端控制器DispatcherServlet所拦截;
2. DispatcherServlet拦截到请求后，会调用HandlerMapping处理器映射器;
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会通过返回信息选择合适的HandlerAdapter（处理器适配器）;

5. HandlerAdapter会调用并执行Handler（处理器），这里的处理器指的就是程序中编写的Controller类，也被称之为后端控制器;
6. Controller执行完成后，会返回一个ModelAndView对象，该对象中会包含视图名或包含模型和视图名;
7. HandlerAdapter将ModelAndView对象返回给DispatcherServlet;
8. DispatcherServlet会根据ModelAndView对象选择一个合适的ViewReslover（视图解析器）;
9. ViewReslover解析后，会向DispatcherServlet中返回具体的View（视图）;
10. DispatcherServlet对View进行渲染（即将模型数据填充至视图中）;
11. 视图渲染结果会返回给客户端浏览器显示。

2.3 SpringMVC 的请求响应流程(面试题)



第一步：用户发送请求到前端控制器（DispatcherServlet）。

第二步：前端控制器请求 HandlerMapping 查找 Handler(Controller)，可以根据 xml 配置或者注解进行查找。

第三步：处理器映射器 HandlerMapping 向前端控制器返回 Handler(Controller)

第四步：前端控制器调用处理器适配器(HandlerAdapter)去执行 Handler(Controller)

第五步：处理器适配器执行 Handler

第六步：Handler 执行完成后给适配器返回ModelAndView

第七步：处理器适配器向前端控制器返回 ModelAndView ModelAndView是SpringMVC 框架的一个底层对象，包括 Model(模型数据) 和 View(视图)

第八步：前端控制器请求试图解析器去进行视图解析 根据逻辑视图名来解析真正的视图。

第九步：试图解析器向前端控制器返回 view

第十步：前端控制器进行视图渲染 就是将模型数据（在 ModelAndView 对象中）填充到 request 域

第十一步：前端控制器向用户响应结果

2.4 SpringMVC的核心组件

DispatcherServlet：前端控制器（不需要程序员开发,需要WEB.xml配置） 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。是Spring MVC 的入口。

HandlerMapping：处理器映射器（不需要程序员开发） HandlerMapping 负责根据用户请求找到 Handler 即处理器，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。

Handler(Controller)：处理器（需要程序员开发） 它就是我们开发中要编写的具体业务控制器。由 DispatcherServlet 把用户请求转发到 Handler。由 Handler 对具体的用户请求进行处理。

HandlAdapter：处理器适配器（不需要程序员开发） 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。按照特定规则（HandlerAdapter要求的规则）去执行 Handler（Controller）

View Resolver：视图解析器（不需要程序员开发，只需要配置） 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。

View：视图 SpringMVC 框架提供了很多的 View 视图类型的支持，包括：jstlView、freemarkerView、pdfView等。我们最常用的视图就是 jsp。

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，需要由程序员根据业务需求开发具体的页面。