Spring MVC

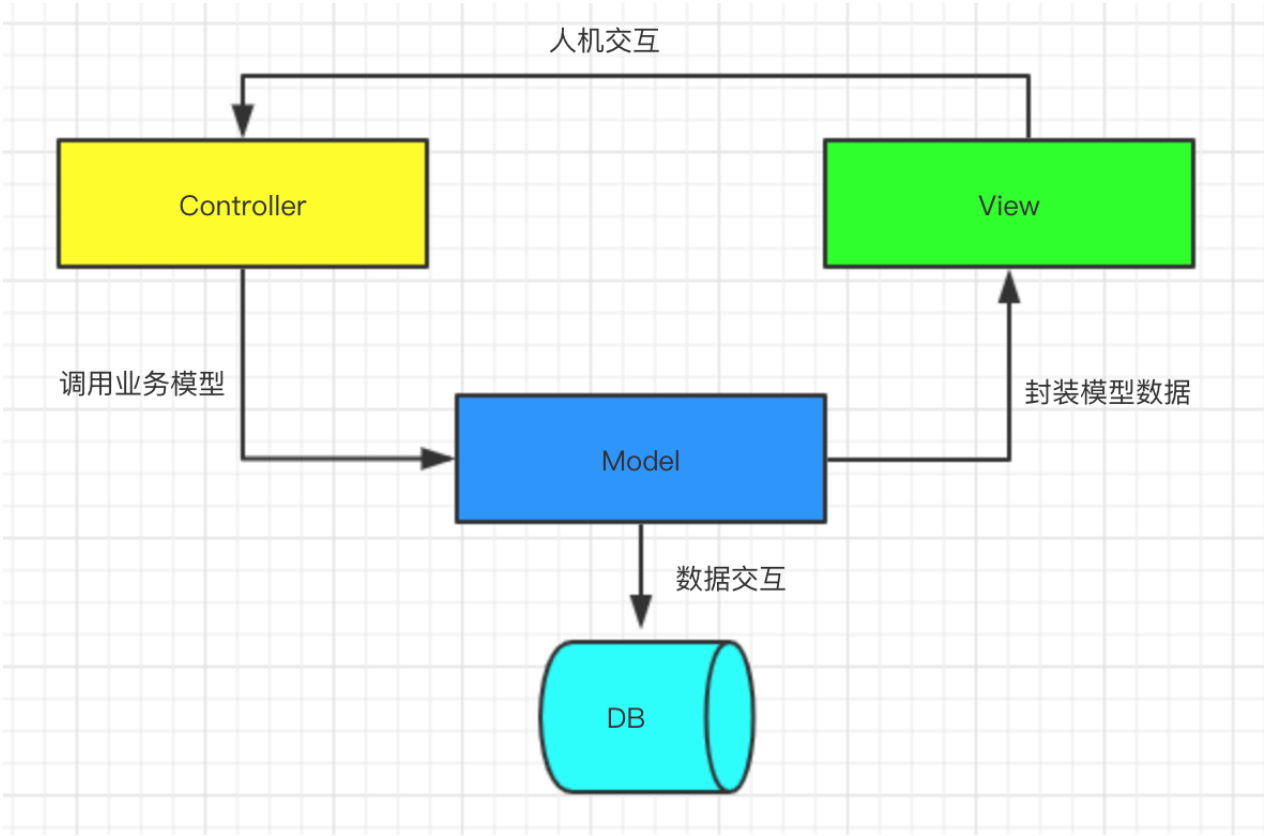
Spring MVC 是 Spring Framework 提供的 Web 组件，全称是 Spring Web MVC，是目前主流的实现 MVC 设计模式的框架，提供前端路由映射、视图解析等功能。

Java Web 开发者必须要掌握的技术框架。

Spring MVC 功能

MVC：Controller（控制层）、Model（模型层）、View（视图层）

流程：Controller 接收客户端请求，调用相关业务层组件产出 Model，或业务数据并返回给 Controller，Controller 再结合 View 完成业务数据的视图层渲染，并将结果响应给客户端，如下图所示。



Spring MVC 对这套 MVC 流程进行封装，帮助开发者屏蔽底层代码，并且开放出相关接口供开发者调用，让 MVC 开发变得更加简单方便。

Spring MVC 实现原理

核心组件

- DispatcherServlet：前置控制器，负责调度其他组件的执行，可以降低不同组件之间的耦合性，是整个 Spring MVC 的核心模块。
- Handler：处理器，完成具体的业务逻辑，相当于 Servlet。
- HandlerMapping：DispatcherServlet 是通过 HandlerMapping 将请求映射到不同的 Handler。
- HandlerInterceptor：处理器拦截器，是一个接口，如果我们需要进行一些拦截处理，可以通过实

现该接口完成。

- HandlerExecutionChain：处理器执行链，包括两部分内容：Handler 和 HandlerInterceptor（系统会有一个默认的 HandlerInterceptor，如果需要额外拦截处理，可以添加拦截器进行设置）。
- HandlerAdapter：处理器适配器，Handler 执行业务方法之前，需要进行一系列的操作包括表单的数据验证、数据类型的转换、将表单数据封装到 POJO 等，这一些列操作都是由 HandlerAdapter 完成，DispatcherServlet 通过 HandlerAdapter 执行不同的 Handler。
- ModelAndView：封装了模型数据和视图信息，作为 Handler 的处理结果，返回给 DispatcherServlet。
- ViewResolver：视图解析器，DispatcherServlet 通过它将逻辑视图解析为物理视图，最终将渲染的结果响应给客户端。

工作流程

1、客户端请求被 DispatcherServlet 接收。

2、根据 HandlerMapping 映射到 Handler。

3、生成 Handler 和 HandlerInterceptor。

4、Handler 和 HandlerInterceptor 以 HandlerExecutionChain 的形式一并返回给 DispatcherServlet。

5、DispatcherServlet 通过 HandlerAdpater 调用 Handler 的方法完成业务逻辑处理。

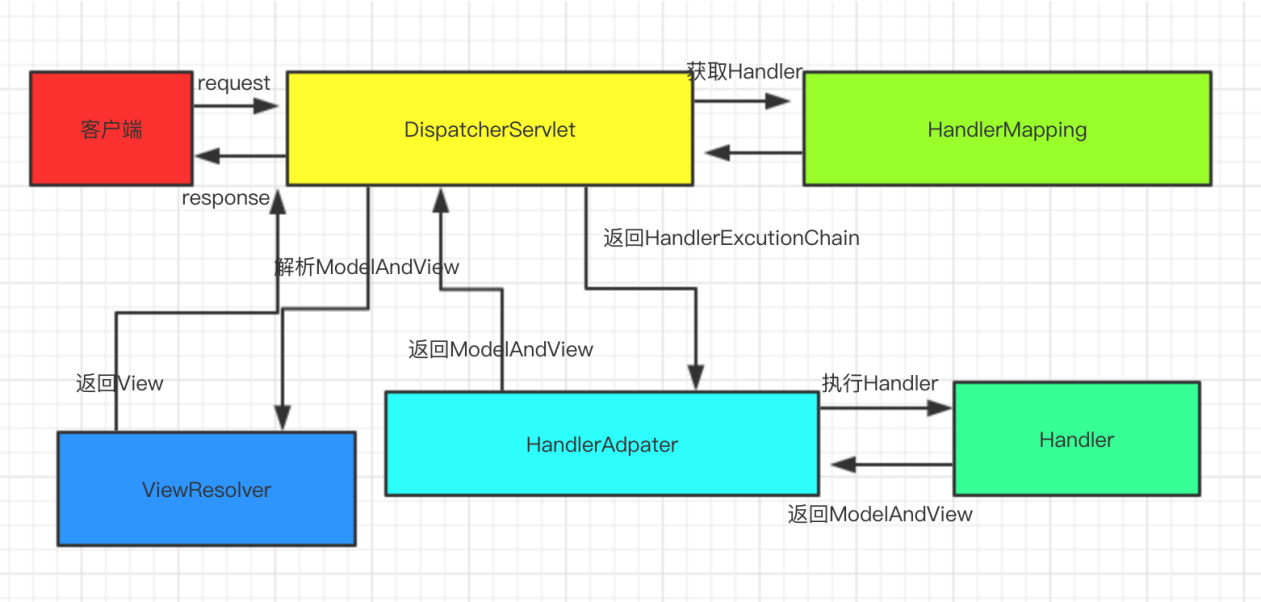
6、返回一个 ModelAndView 对象给 DispatcherServlet。

7、DispatcherServlet 将获取的 ModelAndView 对象传给 ViewResolver 视图解析器，将逻辑视图解析成物理视图。

8、ViewResolver 返回一个 View 给 DispatcherServlet。

9、DispatcherServlet 根据 View 进行视图渲染（将模型数据填充到视图中）。

10、DispatcherServlet 将渲染之后的视图响应给客户端。



Spring MVC 具体使用

1、创建 Maven 工程，pom.xml。

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.2.3.RELEASE</version>
</dependency>
```

2、在 web.xml 中配置 Spring MVC 的 DispatcherServlet。

```
<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

3、springmvc.xml

```
<?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/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       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
       http://www.springframework.org/schema/mvc
       http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <!-- 配置自动扫描 -->
    <context:component-scan base-package="com.southwind"></context:component-
scan>

    <!-- 视图解析器 -->
```

```
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!-- 前缀 -->
        <property name="prefix" value="/"></property>
        <!-- 后缀 -->
        <property name="suffix" value=".jsp"></property>
    </bean>

</beans>
```

4、创建 Handler

```
package com.southwind.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloHandler {

    @RequestMapping("/index")
    public String index(){
        System.out.println("接收到了请求");
        //返回逻辑视图
        return "index";
    }

}
```

流程梳理

1、DispatcherServlet 接收到 URL 请求 index，结合 @RequestMapping("/index") 注解将该请求交给 index 业务方法进行处理。

2、执行 index 业务方法，控制台打印日志，并且返回 "index" 字符串（逻辑视图）。

3、结合 springmvc.xml 中的视图解析器配置，找到目标资源：/index.jsp，即根目录下的 index.jsp 文件，将该 JSP 资源返回给客户端完成响应。

Spring MVC 环境搭建成功。

Spring MVC 常用注解

@RequestMapping

Spring MVC 通过 @RequestMapping 注解将 URL 请求与业务方法进行映射，在控制器的类定义处以及方法定义处都可以添加 @RequestMapping ，在类定义处添加相当于多了一层访问路径。

```
package com.southwind.controller;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/hello")
public class HelloHandler {

    @RequestMapping("/index")
    public String index(){
        System.out.println("接收到了请求");
        //返回逻辑视图
        return "index";
    }

}
```

http://localhost:8080/hello/index

@RequestMapping 常用参数

- value：指定 URL 请求的实际地址，是 @RequestMapping 的默认值

```
@RequestMapping("/index")
public String index(){
    System.out.println("接收到了请求");
    //返回逻辑视图
    return "index";
}
```

等同于

```
@RequestMapping(value="/index")
public String index(){
    System.out.println("接收到了请求");
    //返回逻辑视图
    return "index";
}
```

- method：指定请求的 method 类型，包括 GET、POST、PUT、DELETE 等。

```
@RequestMapping(value = "/index",method = RequestMethod.POST)
public String index(){
    System.out.println("接收到了请求");
    //返回逻辑视图
    return "index";
}
```

上述代码表示只有 POST 请求可以访问该方法，若使用其他请求访问，直接抛出异常，比如 Get。

- params：指定 request 请求中必须包含的参数值，若不包含，无法调用该方法。

```
@RequestMapping(value = "/index",method = RequestMethod.POST,params =
{"id=1","name=tom"})
public String index(){
    System.out.println("接收到了请求");
    //返回逻辑视图
    return "index";
}
```

上述代码表示 request 请求中必须包含 name 和 id 两个参数，并且 id 的值必须为 1，name 的值必须为 tom，才可调用，否则抛出 400 异常。

参数绑定

params 是对 URL 请求参数进行限制，不满足条件的 URL 无法访问该方法，需要在业务方法中获取 URL 的参数值。

1、在业务方法定义时声明参数列表。

2、给参数列表添加 @RequestParam 注解进行绑定。

```
@RequestMapping(value = "/index",method = RequestMethod.POST)
public String index(@RequestParam("num") Integer id,@RequestParam("str")
String name){
    System.out.println("接收到了请求,参数是：id="+id+",name="+name);
    //返回逻辑视图
    return "index";
}
```

Spring MVC 可以自动完成数据类型转换，该工作是由 HandlerAdapter 来完成的。

Spring MVC 也支持 RESTful 风格的 URL 参数获取

传统的 URL：localhost:8080/hello/index?id=1&name=tom

RESTful URL：localhost:8080/hello/index/1/tom

```
@RequestMapping("/restful/{id}/{name}")
public String restful(@PathVariable("id") Integer id,@PathVariable("name")
String name){
    System.out.println(id+"-"+name);
    return "index";
}
```

将参数列表的注解改为 @PathVariable("id") 即可。

映射 Cookie

```
@RequestMapping("/cookie")
public String getCookie(@CookieValue("JSESSIONID") String sessionId){
    System.out.println(sessionId);
    return "index";
}
```

使用 POJO 绑定参数

Spring MVC 会根据请求参数名和 POJO 属性名进行匹配，自动为该对象填充属性值，并且支持属性级联。

如果出现中文乱码，可以通过配置过滤器来解决，在 web.xml 中添加配置即可。

```
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Address

```java
package com.southwind.entity;

import lombok.Data;

@Data
public class Address {
    private Integer code;
    private String value;
}
```

User

```java
package com.southwind.entity;

import lombok.Data;

@Data
public class User {
    private Integer id;
    private String name;
    private Address address;
}
```

addUser.jsp

```jsp
<%--
  Created by IntelliJ IDEA.
  User: southwind
  Date: 2020-02-07
  Time: 16:19
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form action="/hello/add" method="post">
        <table>
            <tr>
                <td>编号：</td>
                <td>
                    <input type="text" name="id"/>
                </td>
            </tr>
            <tr>
                <td>姓名：</td>
```

```
                <td>
                    <input type="text" name="name"/>
                </td>
            </tr>
            <tr>
                <td>地址编号：</td>
                <td>
                    <input type="text" name="address.code"/>
                </td>
            </tr>
            <tr>
                <td>地址信息：</td>
                <td>
                    <input type="text" name="address.value"/>
                </td>
            </tr>
            <tr>
                <td>
                    <input type="submit" value="提交"/>
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

Handler

```
@RequestMapping(value = "/add",method = RequestMethod.POST)
public String add(User user){
  System.out.println(user);
  return "index";
}
```

主体对象可以没有无参构造函数，但是级联对象必须有无参构造函数。