Spring MVC

Spring MVC是什么?

Spring MVC是Spring框架中的一个分支，是目前最好的实现MVC设计模式的框架，以Spring框架的IoC容器为基础，并利用容器的特性来简化它的配置，Spring MVC相当于Spring的一个子模块，可以很好地和Spring框架进行整合开发，是Java Web开发者必须要掌握的框架。

Spring MVC能干什么?

实现了MVC设计模式，该设计模式是一种常用的软件架构方式：以Controller（控制层）、Model（模型层）、View（视图层）三个模块分离的形式来组织代码。

MVC流程：控制层接收到客户端请求，调用模型层生成业务数据，传递给视图层，将最终的业务数据和视图响应给客户端做展示。

Spring MVC就是对这套流程的封装，屏蔽了很多底层代码，开放出接口，让开发者可以更加轻松快捷地完成基于MVC设计模式的Web开发。

Spring MVC实现原理

核心组件

- DispatcherServlet：前端控制器，是整个流程控制的核心，控制其他组件的执行，统一调度，降低组件之间的耦合性，相当于总指挥。
- Handler：处理器，完成具体业务逻辑，相当于Servlet。
- HandlerMapping：处理器映射，DispatcherServlet接收到请求之后，通过HandlerMapping将不同的请求分发给不同的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，可以来实现这个接口。
- 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，如果需要额外设置拦截处理，可以添加拦截器设置）。
- HandlerAdapter：处理器适配器，Handler执行业务方法之前，需要进行一些操作，包括表单数据的验证，数据类型的转换，将表单数据封装到JavaBean等，这一系列的操作，都是由HandlerAdapter来完成，DispatcherServlet通过HandlerAdapter执行不同的Handler。
- ModelAndView：装载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，最终将渲染结果响应给客户端。

以上就是Spring MVC的核心组件，这些组件之间是如何进行交互的?

Spring MVC实现流程

- 客户端请求被DisptacherServlet接收。
- 根据HandlerMapping映射到Handler。
- 生成Handler和HandlerInterceptor。

- Handler和HandlerInterceptor以HandlerExecutionChain的形式一并返回给DispatcherServlet。
- DispatcherServlet通过HandlerAdapter调用Handler的方法完成业务逻辑处理。
- 返回一个ModelAndView对象给DispatcherServlet。
- 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View。
- ViewResolver返回一个View给DispatcherServlet。
- DispatcherServlet根据View进行视图渲染（将模型数据填充到视图中）。
- DispatcherServlet将渲染后的视图响应给客户端。

如何使用?

Spring MVC底层原理虽然很复杂，但是使用起来非常简单，很多组件都由框架提供，作为开发者我们直接使用即可，并不需要自己手动编写代码，真正需要开发者进行编写的组件只有两个：

- Handler，处理业务逻辑。
- View，JSP做展示。

（1）搭建环境。

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.3.1.RELEASE</version>
</dependency>
```

（2）web.xml中配置Spring MVC的DispatcherServlet。

```
<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>
```

（3）创建springmvc.xml配置文件：

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

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

自动扫描的类结合注解交给IoC容器来管理，视图解析器是Spring MVC底层的类，开发者只需要进行配置即可完成JSP页面的跳转，配置也很好理解，一个规则：目标资源路径 = 前缀+返回值+后缀。

比如，Handler返回index，配置文件的前缀是/，后缀是.jsp，代入上述公式：目标资源路径 = /index.jsp。

handler

```
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("执行了index业务方法");
        return "index";
    }

}
```

直接在业务方法定义处添加注解@RequestMapping("/index")，将url请求index直接于index业务方法进行映射，使用起来非常方便。

- 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。
- 执行index业务方法，控制台打印日志，并返回“index”字符串。
- 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，将该JSP资源返回给客户端完成响应。

Spring MVC注解

Spring MVC框架提供了功能强大的注解，大大简化了代码开发的同时也提升了程序的可扩展性。

@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添加@RequestMapping注解，在类定义处添加@RequestMapping注解，相当于多了一层访问路径。

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

    @RequestMapping("/index")
    public String index(){
        System.out.println("执行了index业务方法");
        return "index";
    }

}
```

参数

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

```
@RequestMapping(value = "/index")
public String index(){
    System.out.println("执行了index业务方法");
    return "index";
}
```

等于

```
@RequestMapping("/index")
public String index(){
    System.out.println("执行了index业务方法");
    return "index";
}
```

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

```
@RequestMapping(value = "/index",method = RequestMethod.POST)
public String index(){
    System.out.println("执行了index业务方法");
    return "index";
}
```

如果不设置method参数，无论是GET还是POST都可以来访问该业务方法，如果设置了method，那么对请求类型就有了限制。

- params：指定request中必须包含某些参数，否则无法访问业务方法。

```
@RequestMapping(value = "/index",params = {"name","id=1"})
public String index(){
    System.out.println("index...");
    return "index";
}
```

表示url请求必须包含name参数和id参数，同时id参数的值必须为1。

- 参数绑定

params是对url请求的参数进行限制，不满足条件的url无法访问业务方法，这个特性并不是我们开发中常用的，我们需要用到的是在业务方法中获取url请求中的参数。

（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。

```
@RequestMapping(value = "/index",params = {"name","id=1"})
public String index(@RequestParam("name") String name, @RequestParam("id")
int id){
    System.out.println(name);
    System.out.println(id);
    return "index";
}
```

将url请求的参数name和id分别赋给形参name和id，同时完成了数据类型的转换，url参数都是String类型的，根据形参的数据类型，将id转换为int类型，所有的工作都是由HandlerAdapter来完成。

Spring MVC同时支持RESTful风格的url

RESTful是一种互联网架构方式，简单将就是四种不同的请求分别表示四种不同的操作。

- get：获取
- post：添加
- put：修改
- delete：删除

传统的请求：http://localhost:8080/index?name=zhangsan&id=1

RESTful请求：http://localhost:8080/rest/zhangsan/1

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

映射Cookie

Spring MVC通过映射可以直接在业务方法中获取Cookie的值。

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

使用JavaBean绑定参数

Spring MVC会根据请求参数名和Java Bean属性名进行自动匹配，自动为该对象填充属性值，并且支持及联属性。

（1）创建实体类Address、User并进行及联设置

```
package com.southwind.entity;

public class Address {
    private int id;
    private String name;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    @Override
    public String toString() {
        return "Address{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }
}
```

```
package com.southwind.entity;

public class User {
    private int id;
    private String name;
    private Address address;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", address=" + address +
```

```
                '}';
    }
}
```

(2)创建addUser.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form action="/test/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：</td>
                <td>
                    <input type="text" name="address.name"/>
                </td>
            </tr>
            <tr>
                <td>
                    <input type="submit" value="提交"/>
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

(3)业务方法

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

(4)处理中文乱码

```
<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>
```