SpringMV C框架第一天

第一章：三层架构和MVC

1. 三层架构

1. 咱们开发服务器端程序，一般都基于两种形式，一种C/S架构程序，一种B/S架构程序
2. 使用Java语言基本上都是开发B/S架构的程序，B/S架构又分成了三层架构
3. 三层架构
    1. 表现层：WEB层，用来和客户端进行数据交互的。表现层一般会采用MVC的设计模型
    2. 业务层：处理公司具体的业务逻辑的
    3. 持久层：用来操作数据库的

2. MVC模型

1. MVC全名是Model View Controller 模型视图控制器，每个部分各司其职。
2. Model：数据模型，JavaBean的类，用来进行数据封装。
3. View：指JSP、HTML用来展示数据给用户
4. Controller：用来接收用户的请求，整个流程的控制器。用来进行数据校验等。

第二章：SpringMV C的入门案例

1. SpringMV C的概述（查看大纲文档）

1. SpringMVC的概述
    1. 是一种基于Java实现的MVC设计模型的请求驱动类型的轻量级WEB框架。
    2. Spring MVC属于SpringFrameWork的后续产品，已经融合在Spring Web Flow里面。Spring 框架提供了构建 Web 应用程序的全功能 MVC 模块。
    3. 使用 Spring 可插入的 MVC 架构，从而在使用Spring进行WEB开发时，可以选择使用Spring的SpringMVC框架或集成其他MVC开发框架，如Struts1(现在一般不用)，Struts2等。
2. SpringMVC在三层架构中的位置
    1. 表现层框架
3. SpringMVC的优势
4. SpringMVC和Struts2框架的对比

2. SpringMV C的入门程序

1. 创建WEB工程，引入开发的jar包
    1. 具体的坐标如下

```
    <!-- 版本锁定 -->
    <properties>
        <spring.version>5.0.2.RELEASE</spring.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
```

2. 配置核心的控制器（配置DispatcherServlet）

1. 在web.xml配置文件中核心控制器DispatcherServlet

```
    <!-- SpringMVC的核心控制器 -->
    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
        <!-- 配置Servlet的初始化参数，读取springmvc的配置文件，创建spring容器 -->

        <init-param>
```

```
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springmvc.xml</param-value>
        </init-param>
        <!-- 配置servlet启动时加载对象 -->
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

servlet配置/表示拦截所有资源除了jsp

3. 编写springmvc.xml的配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 配置spring创建容器时要扫描的包 -->
    <context:component-scan base-package="com.itheima"></context:component-scan>

    <!-- 配置视图解析器 -->
    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>

    <!-- 配置spring开启注解mvc的支持
    <mvc:annotation-driven></mvc:annotation-driven>-->
</beans>
```

使用<mvc:annotation-driven> 自动加载 RequestMappingHandlerMapping （处理映射器）和 RequestMappingHandlerAdapter （处理适配器），可用在SpringMVC.xml配置文件中使用<mvc:annotation-driven>替代注解处理器和适配器的配置。

4. 编写index.jsp和HelloController控制器类

1. index.jsp

```
<body>

    <h3>入门案例</h3>

    <a href="${ pageContext.request.contextPath }/hello">入门案例</a>

</body>
```

2. HelloController

```
package cn.itcast.controller;

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

/**
 * 控制器
 * @author rt
 */
@Controller
public class HelloController {

    /**
     * 接收请求
     * @return
     */
    @RequestMapping(path="/hello")
    public String sayHello() {
        System.out.println("Hello SpringMVC!!");
        return "success";
    }

}
```

5. 在WEB-INF目录下创建pages文件夹，编写success.jsp的成功页面

```
<body>

    <h3>入门成功！！</h3>

</body>
```

6. 启动Tomcat服务器，进行测试

3. 入门案例的执行过程分析

1. 入门案例的执行流程

1. 当启动Tomcat服务器的时候，因为配置了load-on-startup标签，所以会创建DispatcherServlet对象，就会加载springmvc.xml配置文件
2. 开启了注解扫描，那么HelloController对象就会被创建
3. 从index.jsp发送请求，请求会先到达DispatcherServlet核心控制器，根据配置@RequestMapping注解找到执行的具体方法
4. 根据执行方法的返回值，再根据配置的视图解析器，去指定的目录下查找指定名称的JSP文件
5. Tomcat服务器渲染页面，做出响应

2. SpringMVC官方提供图形

3. 入门案例中的组件分析

1. 前端控制器（DispatcherServlet）
2. 处理器映射器（HandlerMapping）
3. 处理器（Handler）
4. 处理器适配器（HandlAdapter）
5. 视图解析器（View Resolver）
6. 视图（View）

4. RequestMapping 注解

1. RequestMapping注解的作用是建立请求URL和处理方法之间的对应关系
2. RequestMapping注解可以作用在方法和类上
    1. 作用在类上：第一级的访问目录
    2. 作用在方法上：第二级的访问目录
    3. 细节：路径可以不编写 / 表示应用的根目录开始
    4. 细节：${ pageContext.request.contextPath }也可以省略不写，但是路径上不能写 /
3. RequestMapping的属性
    1. path　　指定请求路径的url
    2. value　　value属性和path属性是一样的
    3. mthod　　指定该方法的请求方式
    4. params　指定限制请求参数的条件
    5. headers　发送的请求中必须包含的请求头

第三章：请求参数的绑定

1. 请求参数的绑定说明
    1. 绑定机制
        1. 表单提交的数据都是k=v格式的　username=haha&password=123
        2. SpringMVC的参数绑定过程是把表单提交的请求参数，作为控制器中方法的参数进行绑定的
        3. 要求：提交表单的name和参数的名称是相同的
    2. 支持的数据类型
        1. 基本数据类型和字符串类型
        2. 实体类型（JavaBean）
        3. 集合数据类型（List、map集合等）
2. 基本数据类型和字符串类型
    1. 提交表单的name和参数的名称是相同的
    2. 区分大小写
3. 实体类型（JavaBean）
    1. 提交表单的name和JavaBean中的属性名称需要一致
    2. 如果一个JavaBean类中包含其他的引用类型，那么表单的name属性需要编写成：对象.属性 例如：address.name
4. 给集合属性数据封装

封装数组：

jsp:
```
<form action="param/checkbox" method="post">
    爱好：<br/>
    <input type="checkbox" name="hobby" value="1">看电影
    <input type="checkbox" name="hobby" value="2">玩游戏
    <input type="checkbox" name="hobby" value="3">听音乐
    <input type="submit" value="提交">
</form>
```

controll:
```
@RequestMapping("/checkbox")
public String testCheckbox(Integer[] hobby){
    System.out.println(Arrays.toString(hobby));
    return "success";
}
```

集合： 集合属性名[索引].集合中元素的属性名

map： map属性名['key'].map集合中元素的属性名

1. JSP页面编写方式：list[0].属性

5. 请求参数中文乱码的解决
    1. 在web.xml中配置Spring提供的过滤器类

```
    <!-- 配置过滤器，解决中文乱码的问题 -->
    <filter>
        <filter-name>characterEncodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-
class>
        <!-- 指定字符集 -->
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>characterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

6. 自定义类型转换器
    1. 表单提交的任何数据类型全部都是字符串类型，但是后台定义Integer类型，数据也可以封装上，说明Spring框架内部会默认进行数据类型转换。
    2. 如果想自定义数据类型转换，可以实现Converter的接口
        1. 自定义类型转换器

```
package cn.itcast.utils;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

import org.springframework.core.convert.converter.Converter;

/**
 * 把字符串转换成日期的转换器
 * @author rt
 */                                                    转换后的值
                                                  要转换的的类型
public class StringToDateConverter implements Converter<String, Date>{

    /**
     * 进行类型转换的方法
     */
    public Date convert(String source) {
        // 判断
        if(source == null) {
            throw new RuntimeException("参数不能为空");

        }
```

```
        try {
            DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
            // 解析字符串
            Date date = df.parse(source);
            return date;
        } catch (Exception e) {
            throw new RuntimeException("类型转换错误");
        }
    }

}
```

2. 注册自定义类型转换器，在springmvc.xml配置文件中编写配置

```
    <!-- 注册自定义类型转换器 -->
    <bean id="conversionService"
class="org.springframework.context.support.ConversionServiceFactoryBean">
        <property name="converters">
            <set>
                <bean class="cn.itcast.utils.StringToDateConverter"/>
            </set>
        </property>
    </bean>

    <!-- 开启Spring对MVC注解的支持 -->
    <mvc:annotation-driven conversion-service="conversionService"/>
```

会覆盖原先的String转date方式，其他类型转换不受影响

7. 在控制器中使用原生的ServletAPI对象
    1. 只需要在控制器的方法参数定义HttpServletRequest和HttpServletResponse对象

第四章：常用的注解

1. RequestParam注解
    1. 作用：把请求中的指定名称的参数传递给控制器中的形参赋值
    2. 属性
        1. value：请求参数中的名称
        2. required：请求参数中是否必须提供此参数，默认值是true，必须提供
    3. 代码如下

```
/**
 * 接收请求
 * @return
 */
@RequestMapping(path="/hello")
public String sayHello(@RequestParam(value="username",required=false)String name) {
    System.out.println("aaaa");
    System.out.println(name);
    return "success";
}
```

2. RequestBody注解 用于接受ajax异步请求中的json数据
    1. 作用：用于获取请求体的内容（注意：get方法不可以） 只支持post请求方式
    2. 属性
        1. required：是否必须有请求体，默认值是true
    3. 代码如下

```
/**
 * 接收请求
 * @return
 */
@RequestMapping(path="/hello")
public String sayHello(@RequestBody String body) {
    System.out.println("aaaa");
    System.out.println(body);
    return "success";
}
```

3. PathVariable注解
    1. 作用：拥有绑定url中的占位符的。例如：url中有/delete/{id}，{id}就是占位符
    2. 属性
        1. value：指定url中的占位符名称

页面：/delete/10

后台：

@RequestMapping(/delete/{id})

public String testDelete(@PathVariable String id){}

    3. Restful风格的URL
        1. 请求路径一样，可以根据不同的请求方式去执行后台的不同方法
        2. restful风格的URL优点
            1. 结构清晰
            2. 符合标准
            3. 易于理解
            4. 扩展方便
    4. 代码如下

```
    <a href="user/hello/1">入门案例</a>

    /**
     * 接收请求
     * @return
     */
    @RequestMapping(path="/hello/{id}")
    public String sayHello(@PathVariable(value="id") String id) {
        System.out.println(id);
        return "success";
    }
```

4. RequestHeader注解
    1. 作用：获取指定请求头的值
    2. 属性
        1. value：请求头的名称
    3. 代码如下

```
@RequestMapping(path="/hello")
    public String sayHello(@RequestHeader(value="Accept") String header) {
        System.out.println(header);
        return "success";
    }
```

5. CookieValue注解
    1. 作用：用于获取指定cookie的名称的值
    2. 属性
        1. value：cookie的名称
    3. 代码

```
    @RequestMapping(path="/hello")
    public String sayHello(@CookieValue(value="JSESSIONID") String cookieValue) {
        System.out.println(cookieValue);
        return "success";
    }
```

6. ModelAttribute注解
    1. 作用　　用于给数据添加默认值，可以储存数据
        1. 出现在方法上：表示当前方法会在控制器方法执行前线执行。
        2. 出现在参数上：获取指定的数据给参数赋值。
    2. 应用场景
        1. 当提交表单数据不是完整的实体数据时，保证没有提交的字段使用数据库原来的数据。
    3. 具体的代码

1. 修饰的方法有返回值

```
/**
 * 作用在方法，先执行
 * @param name
 * @return
 */
@ModelAttribute
public User showUser(String name) {
    System.out.println("showUser执行了...");
    // 模拟从数据库中查询对象
    User user = new User();
    user.setName("哈哈");
    user.setPassword("123");
    user.setMoney(100d);
    return user;
}

/**
 * 修改用户的方法
 * @param cookieValue
 * @return
 */
@RequestMapping(path="/updateUser")
public String updateUser(User user) {
    System.out.println(user);
    return "success";
}
```

2. 修饰的方法没有返回值

```
/**
 * 作用在方法，先执行
 * @param name
 * @return
 */
@ModelAttribute
public void showUser(String name,Map<String, User> map) {
    System.out.println("showUser执行了...");
    // 模拟从数据库中查询对象
    User user = new User();
    user.setName("哈哈");
    user.setPassword("123");
    user.setMoney(100d);
    map.put("abc", user);
}

/**
 * 修改用户的方法
 * @param cookieValue
 * @return
 */
```

```
    @RequestMapping(path="/updateUser")
    public String updateUser(@ModelAttribute(value="abc") User user) {
        System.out.println(user);
        return "success";
    }
```

4. SessionAttributes注解
    1. 作用：用于多次执行控制器方法间的参数共享
    2. 属性
        1. value：指定存入属性的名称　2.type：用于指定存入的数据类型
    3. 代码如下

```
@Controller
@RequestMapping(path="/user")
@SessionAttributes(value= {"username","password","age"},types=
{String.class,Integer.class})        // 把数据存入到session域对象中
public class HelloController {

    /**
     * 向session中存入值
     * @return
     */
    @RequestMapping(path="/save")
    public String save(Model model) {
        System.out.println("向session域中保存数据");
        model.addAttribute("username", "root");
        model.addAttribute("password", "123");
        model.addAttribute("age", 20);
        return "success";
    }

    /**
     * 从session中获取值
     * @return
     */
    @RequestMapping(path="/find")
    public String find(ModelMap modelMap) {
        String username = (String) modelMap.get("username");
        String password = (String) modelMap.get("password");
        Integer age = (Integer) modelMap.get("age");
        System.out.println(username + " : "+password +" : "+age);
        return "success";
    }

    /**
     * 清除值
     * @return
     */
    @RequestMapping(path="/delete")

    public String delete(SessionStatus status) {
```

```
        status.setComplete();
        return "success";
    }

}
```

课程总结

1. SpringMVC的概述
2. 入门
    1. 创建工程，导入坐标
    2. 在web.xml中配置前端控制器（启动服务器，加载springmvc.xml配置文件）
    3. 编写springmvc.xml配置文件
    4. 编写index.jsp的页面，发送请求
    5. 编写Controller类，编写方法（@RequestMapping(path="/hello")），处理请求
    6. 编写配置文件（开启注解扫描），配置视图解析器
    7. 执行的流程
    8. @RequestMapping注解
        1. path
        2. value
        3. method
        4. ....
3. 参数绑定
    1. 参数绑定必须会
    2. 解决中文乱码，配置过滤器
    3. 自定义数据类型转换器