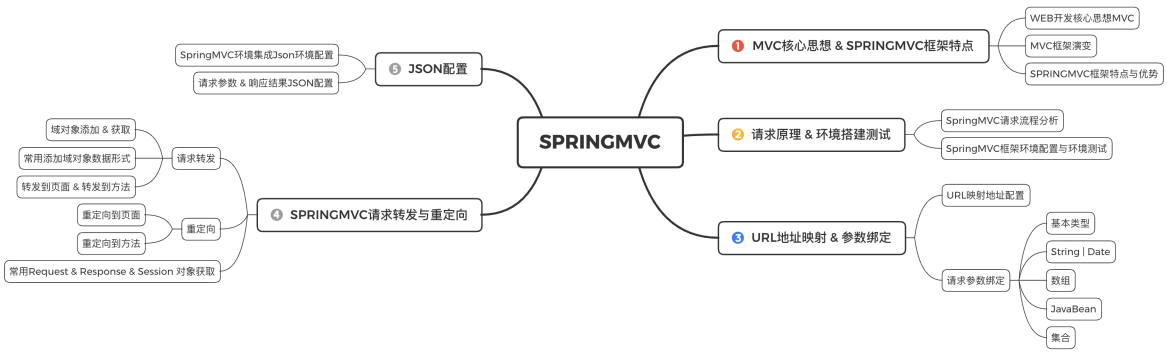
SpringMVC - 第一天

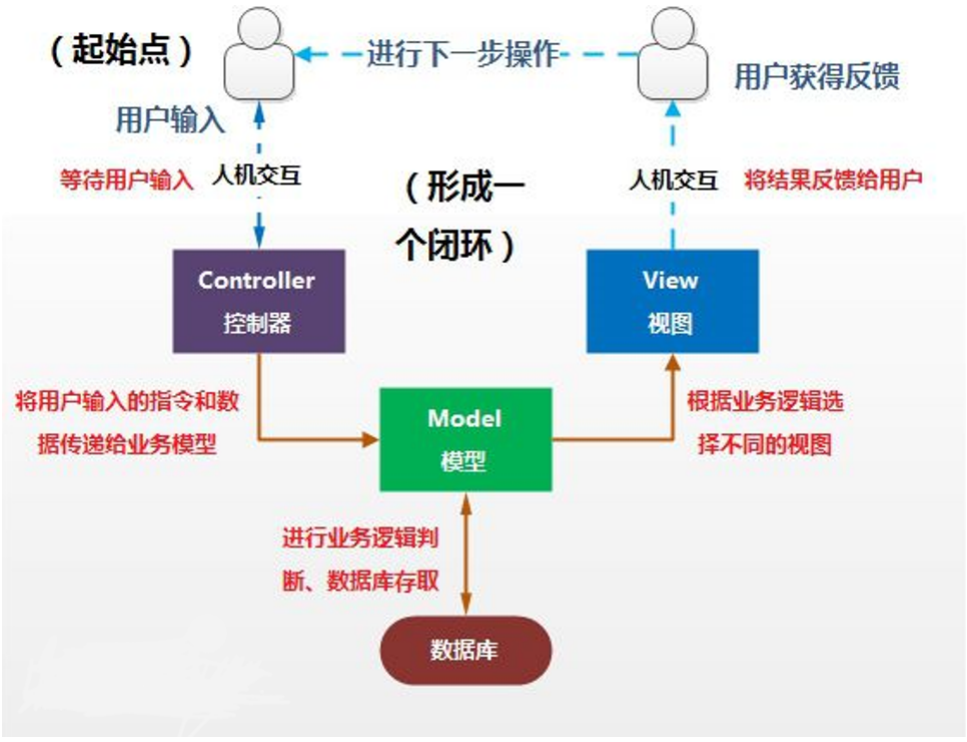
1. 学习目标



2. MVC 思想

2.1. 什么叫MVC?

模型-视图-控制器（MVC）是一个众所周知的以设计界面应用程序为基础的设计思想。它主要通过分离模型、视图及控制器在应用程序中的角色将业务逻辑从界面中解耦。通常，模型负责封装应用程序数据在视图层展示。视图仅仅只是展示这些数据，不包含任何业务逻辑。控制器负责接收来自用户的请求，并调用后台服务（service或者dao）来处理业务逻辑。处理后，后台业务层可能会返回了一些数据在视图层展示。控制器收集这些数据及准备模型在视图层展示。MVC模式的核心思想是将业务逻辑从界面中分离出来，允许它们单独改变而不会相互影响。



2.2. 常见MVC框架运行性能比较

Jsp+servlet > struts1 > spring mvc > struts2+freemarker > struts2,ognl,值栈。

开发效率上，基本正好相反。值得强调的是，spring mvc开发效率和struts2不相上下，但从目前来看，spring mvc 的流行度已远远超过struts2。

3. SpringMVC 框架概念与特点

3.1. Spring MVC是什么?

Spring MVC是Spring家族中的一个web成员，它是一种基于Java的实现了Web MVC设计思想的请求驱动类型的轻量级Web框架，即使用了MVC架构模式的思想，将web层进行职责解耦，基于请求驱动指的就是使用请求-响应模型，框架的目的就是帮助我们简化开发，Spring MVC也是要简化我们日常Web开发的。

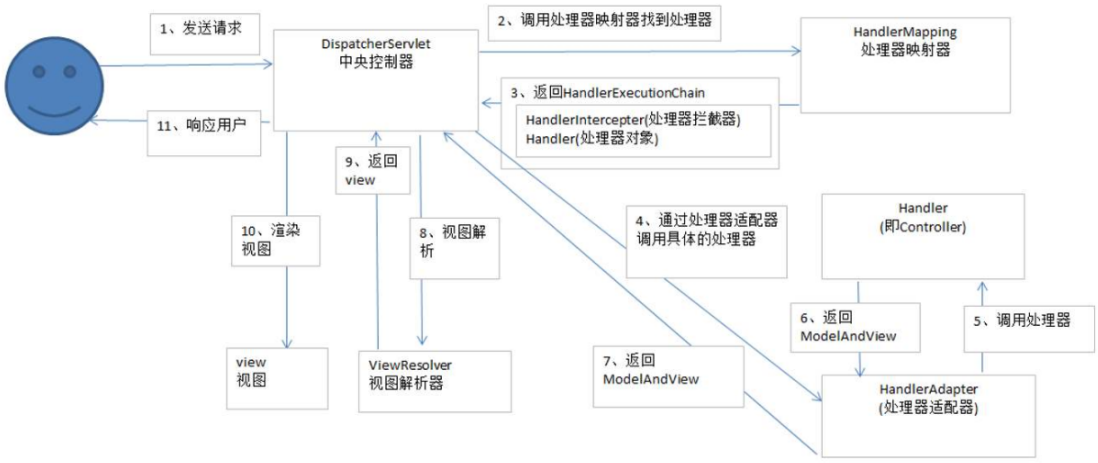
Spring MVC是服务到工作者思想的实现。中央控制器是DispatcherServlet；应用控制器拆为处理器映射器(Handler Mapping)进行处理器管理和视图解析器(View Resolver)进行视图管理；支持本地化/国际化（Locale）解析及文件上传等；提供了非常灵活的数据验证、格式化和数据绑定机制；提供了强大的约定大于配置（惯例优先原则）的契约式编程支持。

3.2. Spring MVC能帮我们做什么?

1. 让我们能非常简单的设计出干净的Web层；
2. 进行更简洁的Web层的开发；
3. 天生与Spring框架集成（如IOC容器、AOP等）；
4. 提供强大的约定大于配置的契约式编程支持；
5. 能简单的进行Web层的单元测试；
6. 支持灵活的URL到页面控制器的映射；
7. 非常容易与其他视图技术集成，如jsp、Velocity、FreeMarker等等，因为模型数据不放在特定的API里，而是放在一个Model里（Map数据结构实现，因此很容易被其他框架使用）；
8. 非常灵活的数据验证、格式化和数据绑定机制，能使用任何对象进行数据绑定，不必实现特定框架的API；
9. 支持灵活的本地化等解析；
10. 更加简单的异常处理；
11. 对静态资源的支持；
12. 支持Restful风格。

4. SpringMVC 请求流程

4.1. Spring MVC 请求处理流程分析



Spring MVC框架也是一个基于请求驱动的Web框架，并且使用了前端控制器模式（是用来提供一个集中的请求处理机制，所有的请求都将由一个单一的处理程序处理来进行设计，再根据请求映射规则分发给相应的页面控制器（动作/处理器）进行处理。首先让我们整体看一下Spring MVC处理请求的流程：

1. 首先用户发送请求,请求被SpringMvc前端控制器（DispatherServlet）捕获；
2. 前端控制器(DispatherServlet)对请求URL解析获取请求URI，根据URI，调用HandlerMapping；
3. 前端控制器(DispatherServlet)获得返回的HandlerExecutionChain（包括Handler对象以及Handler对象对应的拦截器）；
4. DispatcherServlet 根据获得的 HandlerExecutionChain，选择一个合适的HandlerAdapter。（附注：如果成功获得HandlerAdapter后，此时将开始执行拦截器的preHandler(...)方法）；
5. HandlerAdapter根据请求的Handler适配并执行对应的Handler；HandlerAdapter(提取Request中的模型数据，填充Handler入参，开始执行Handler（Controller)。 在填充Handler的入参过程中，根据配置，Spring将做一些额外的工作：

   HttpMessageConveter： 将请求消息（如Json、xml等数据）转换成一个对象，将对象转换为指定的响应信息。

   数据转换：对请求消息进行数据转换。如String转换成Integer、Double等数据格式化：

   数据格式化。 如将字符串转换成格式化数字或格式化日期等

   数据验证： 验证数据的有效性（长度、格式等），验证结果存储到BindingResult或Error中）
6. Handler执行完毕，返回一个ModelAndView(即模型和视图)给HandlerAdaptor
7. HandlerAdaptor适配器将执行结果ModelAndView返回给前端控制器。
8. 前端控制器接收到ModelAndView后，请求对应的视图解析器。
9. 视图解析器解析ModelAndView后返回对应View;
10. 渲染视图并返回渲染后的视图给前端控制器。

11、最终前端控制器将渲染后的页面响应给用户或客户端

4.2. Spring MVC 优势

1. 清晰的角色划分：前端控制器（DispatcherServlet）、请求到处理器映射（HandlerMapping）、处理器适配器（HandlerAdapter）、视图解析器（ViewResolver）、处理器或页面控制器（Controller）、验证器（ Validator）、命令对象（Command 请求参数绑定到的对象就叫命令对象）、表单对象（Form Object 提供给表单展示和提交到的对象就叫表单对象）。
2. 分工明确，而且扩展点相当灵活，可以很容易扩展，虽然几乎不需要；

3. 和Spring 其他框架无缝集成，是其它Web框架所不具备的；
4. 可适配，通过HandlerAdapter可以支持任意的类作为处理器；
5. 可定制性，HandlerMapping、ViewResolver等能够非常简单的定制；
6. 功能强大的数据验证、格式化、绑定机制；
7. 利用Spring提供的Mock对象能够非常简单的进行Web层单元测试；
8. 本地化、主题的解析的支持，使我们更容易进行国际化和主题的切换。
9. 强大的JSP标签库，使JSP编写更容易。

还有比如RESTful（一种软件架构风格，设计风格而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件，目前了解即可）风格的支持、简单的文件上传、约定大于配置的契约式编程支持、基于注解的零配置支持等等。

5. Spring MVC 环境搭建

5.1. 开发环境

Idea + Maven + Jdk1.8 + Jetty

5.2. 新建 Maven webApp

Idea 下创建 springmvc01 工程

5.3. pom.xml 坐标添加

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>
  <dependencies>
    <!-- spring web -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>5.2.4.RELEASE</version>
    </dependency>
    <!-- spring mvc -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.2.4.RELEASE</version>
    </dependency>
    <!-- web servlet -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.0.1</version>
      <scope>provided</scope>
    </dependency>

  </dependencies>
<build>
  <plugins>
    <!-- 编译环境插件 -->
```

```
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <source>11</source>
        <target>11</target>
        <encoding>UTF-8</encoding>
      </configuration>
    </plugin>
    <!-- jetty插件 -->
      <plugin>
        <groupId>org.eclipse.jetty</groupId>
        <artifactId>jetty-maven-plugin</artifactId>
        <version>9.4.27.v20200227</version>
        <configuration>
          <scanIntervalSeconds>10</scanIntervalSeconds>
          <!-- 设置端口 -->
          <httpConnector>
            <port>8080</port>
          </httpConnector>
          <!-- 设置项目路径 -->
          <webAppConfig>
            <contextPath>/springmvc01</contextPath>
          </webAppConfig>
        </configuration>
      </plugin>
  </plugins>
</build>
```

5.4. 配置 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="3.0"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

  <!-- 编码过滤 utf-8 -->
  <filter>
    <description>char encoding filter</description>
    <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>

  <!-- servlet请求分发器 -->
  <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:spring.xml</param-value>
    </init-param>
    <!-- 表示启动容器时初始化该Servlet -->
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springMvc</servlet-name>
    <!-- 这是拦截请求，  "/"代表拦截所有请求，"*.do"拦截所有.do请求 -->
    <!-- <url-pattern>/</url-pattern> -->
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

要想启动我们的 SpringMVC 环境，目前对于 mvc 框架的配置还未进行。以上在 web.xml 中引用了 **spring.xml** 文件。

5.4.1. spring.xml 配置

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        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">

    <!-- 开启扫描器 -->
    <context:component-scan base-package="com.xxxx.springmvc.controller"/>

    <!-- 使用默认的 Servlet 来响应静态文件 -->
    <mvc:default-servlet-handler/>

    <!-- 开启注解驱动-->
    <mvc:annotation-driven/>

    <!-- 配置视图解析器 -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver"
          id="internalResourceViewResolver">
        <!-- 前缀：在WEB-INF目录下的jsp目录下 -->
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <!-- 后缀：以.jsp结尾的资源 -->
        <property name="suffix" value=".jsp"/>
    </bean>

</beans>
```

5.4.2. 页面控制器的编写

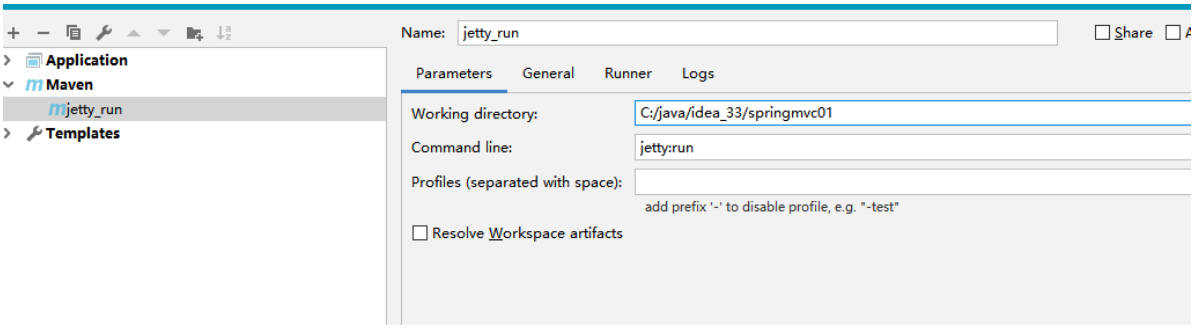
```
@Controller
public class HelloController {

    /**
     * 请求映射地址  /hello.do
     * @return
     */
    @RequestMapping("/hello")
    public ModelAndView hello(){
        ModelAndView mv=new ModelAndView();
        mv.addObject("hello", "hello spring mvc");
        mv.setViewName("hello");
        return mv;
    }
}
```

5.4.3. 添加视图页面

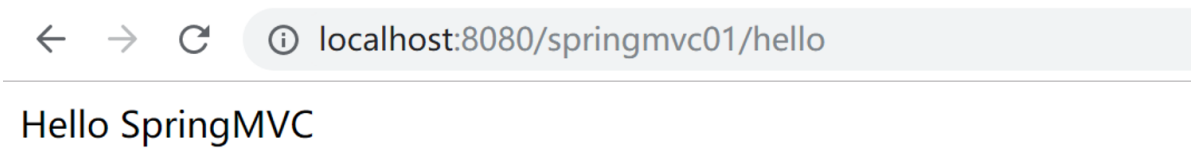
在 WEB-INF 下新建 jsp 文件夹， 并在文件夹下新建 hello.jsp

```
<%@page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+pa
th+"/";
%>
<!DOCTYPE HTML>
<html>
  <head>
    <base href="<%=basePath %>">
    <title>My JSP 'hello.jsp' starting page</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
  </head>
  <body>
    <!-- el表达式接收参数值 -->
    ${hello}
  </body>
</html>
```

5.4.4. 启动 jetty 服务器



访问地址 http://localhost:8080/springmvc01/hello.do



Hello SpringMVC

至此，springmvc 环境搭建完毕

6. URL 地址映射配置

6.1. @RequestMapping

通过注解 @RequestMapping 将请求地址与方法进行绑定，可以在类级别和方法级别声明。类级别的注解负责将一个特定的请求路径映射到一个控制器上，将 url 和类绑定；通过方法级别的注解可以细化映射，能够将一个特定的请求路径映射到某个具体的方法上，将 url 和类的方法绑定。

6.1.1. 映射单个 URL

@RequestMapping("") 或 @RequestMapping(value="")

```
/**
  * @RequestMapping 声明在方法上面，映射单个 URL
  *      访问地址：（如果有类路径需要写在方法路径前面）
  *          http://ip:port/springmvc01/test01
  * @return
  */
@RequestMapping("/test01")
// @RequestMapping(value = "/test01")
public ModelAndView test01(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("hello","test01");
    modelAndView.setViewName("hello");
    return modelAndView;
}

/**
  * 路径开头是否加 斜杠"/" 均可
  *      @RequestMapping("/请求路径") 与 @RequestMapping("请求路径")均可
  *      建议加上，如：@RequestMapping("/test02")
  *      访问地址：（如果有类路径需要写在方法路径前面）
  *          http://ip:port/springmvc01/test02
```

```
  * @return
  */
@RequestMapping("test02")
public ModelAndView test02(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("hello","test02");
    modelAndView.setViewName("hello");
    return modelAndView;
}
```

6.1.2. 映射多个 URL

@RequestMapping({"",""}) 或 @RequestMapping(value={"",""})

```
/**
  * @RequestMapping 声明在方法上面，映射多个 URL
  *     支持一个方法绑定多个 url 的操作
  *     访问地址：（如果有类路径需要写在方法路径前面）
  *         http://ip:port/springmvc01/test03_01
  *         http://ip:port/springmvc01/test03_02
  * @return
  */
@RequestMapping({"/test03_01","/test03_02"})
// @RequestMapping(value = {"/test03_01","/test03_02"})
public ModelAndView test03(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("hello","test03");
    modelAndView.setViewName("hello");
    return modelAndView;
}
```

6.1.3. 映射 URL 在控制器上

用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。

```
@Controller
@RequestMapping("/url")
public class UrlController {
    /**
     * @RequestMapping 声明在类上面，类中的的方法都是以该地址作为父路径
     *      声明级别：
     *          类级别 + 方法级别 （/类路径/方法路径）
     *      访问地址：
     *          http://ip:port/springmvc01/url/test04
     * @return
     */
    @RequestMapping("/test04")
    public ModelAndView test04(){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("hello","test04");
        modelAndView.setViewName("hello");
        return modelAndView;
    }
}
```

6.1.4. 设置 URL 映射的请求方式

默认没有设置请求方式，在HTTP 请求中最常用的请求方法是 GET、POST，还有其他的一些方法，如：DELETE、PUT、HEAD 等。

可以通过 method 属性设置支持的请求方式，如 method=RequestMethod.POST；如设置多种请求方式，以大括号包围，逗号隔开即可。

```
/**
  * 设置请求方式
  *  通过 method 属性设置方法支持的请求方式，默认 GET请求和 POST等请求都支持。
  *  设置了请求方式，则只能按照指定的请求方式请求。
  *      访问地址：（只能使用POST请求访问）
  *           http://ip:port/springmvc01/url/test05
  * @return
  */
@RequestMapping(value = "/test05",method = RequestMethod.POST)
public ModelAndView test05(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("hello","test05");
    modelAndView.setViewName("hello");
    return modelAndView;
}
```

6.1.5. 通过参数名称映射 URL

```
/**
  * 通过参数名称访问
  *   通过参数的形式访问
  *   访问地址：
  *      http://ip:port/springmvc01/url?test06
  * @return
  */
@RequestMapping(params = "test06")
public ModelAndView test06(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("hello","test06");
    modelAndView.setViewName("hello");
    return modelAndView;
}
```

7. 参数绑定

客户端请求的参数到控制器功能处理方法上的参数的绑定，对于参数绑定非常灵活。

7.1. 基本数据类型

```
/**
  * 基本类型数据绑定
  *      参数值必须存在。如果没有指定参数值，也没有设置参数默认值，则会报500异常。
  * @param age
  * @param money
  */
@RequestMapping("data01")
```

```
public void data01(int age, double money){
    System.out.println("age:" + age + ", money:" + money);
}

/**
  * 基本类型数据绑定
  *       通过注解 @RequestParam 标记一个形参为请求参数。（注解声明在形参的前面）
  *       可以通过注解的属性设置相关内容
  *           设置参数的默认值 defaultValue
  * @param age
  * @param money
  */
@RequestMapping("data02")
public void data02(@RequestParam(defaultValue = "18") int age,
@RequestParam(defaultValue = "10.0") double money){
    System.out.println("age:" + age + ", money:" + money);
}

/**
  * 基本类型数据绑定
  *       通过注解 @RequestParam 标记一个形参为请求参数。（注解声明在形参的前面）
  *       可以通过注解的属性设置相关内容
  *           设置参数的参数名（别名） name
  * @param age
  * @param money
  */
@RequestMapping("data03")
public void data03(@RequestParam(defaultValue = "18", name = "userAge") int age,
            @RequestParam(defaultValue = "10.0", name = "userMoney") double
money){
    System.out.println("age:" + age + ", money:" + money);
}
```

7.2. 包装类型

```
/**
  * 包装类型数据绑定 （如果数据是基本类型，建议使用包装类型）
  *       客户端请求参数名与方法形参名保持一致，默认参数值为null
  *       可以通过 @RequestParam 的name属性设置参数的别名，defaultValue属性设置参数默认值
  * @param age
  * @param money
  */
@RequestMapping("data05")
public void data05(Integer age, Double money){
    System.out.println("age:" + age + ", money:" + money);
}
```

7.3. 字符串类型

```
/**
  * 字符串数据绑定
  *       客户端请求参数名与方法形参名保持一致，默认参数值为null
  *       可以通过 @RequestParam 的name属性设置参数的别名，defaultValue属性设置参数默认值
  * @param userName
  * @param userPwd
  */
@RequestMapping("data04")
public void data04(String userName, String userPwd){
    System.out.println("userName:" + userName + ", userPwd:" + userPwd);
}
```

7.4. 数组类型

```
/**
  * 数组类型数据绑定
  *       客户端传参形式：ids=1&ids=2&ids=3
  * @param ids
  */
@RequestMapping("/data06")
public void data06(String[] ids){
    for(String id : ids){
        System.out.println(id + "---");
    }
}
```

7.5. JavaBean 类型

```
/**
  * JavaBean 数据绑定
  *       客户端请求的参数名与JavaBean对象的属性字段名保持一致
  * @param user
  */
@RequestMapping("/data07")
public void data07(User user) {
    System.out.println(user);
}
```

User.java

```
package com.xxxx.springmvc.po;

public class User {

    private int id;
    private String userName;
    private String userPwd;
```

```
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getUserPwd() {
        return userPwd;
    }
    public void setUserPwd(String userPwd) {
        this.userPwd = userPwd;
    }
    @Override
    public String toString() {
        return "User [id=" + id + ", userName=" + userName + ", userPwd="
                + userPwd + "]";
    }

}
```

7.6. List 类型

此时 User 实体需要定义对应 list 属性。（对于集合的参数绑定，一般需要使用 JavaBean 对象进行包装）

```
public class User {

    private int id;
    private String userName;
    private String userPwd;

    private List<Phone> phones = new ArrayList<Phone>();


    public List<Phone> getPhones() {
        return phones;
    }
    public void setPhones(List<Phone> phones) {
        this.phones = phones;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUserName() {
        return userName;
```

```
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getUserPwd() {
        return userPwd;
    }
    public void setUserPwd(String userPwd) {
        this.userPwd = userPwd;
    }
    @Override
    public String toString() {
        return "User [id=" + id + ", userName=" + userName + ", userPwd="
                + userPwd + ", phones=" + phones + "]";
    }

}
```

Phone实体

```
public class Phone {

    private String num;

    public String getNum() {
        return num;
    }

    public void setNum(String num) {
        this.num = num;
    }

    @Override
    public String toString() {
        return "Phone [num=" + num + "]";
    }

}
```

Jsp 页面定义

```
<form action="data08" method="post">
    <input name="phones[0].num" value="123456" />
    <input name="phones[1].num" value="4576" />
    <button type="submit"> 提交</button>
</form>
```

Controller 方法

```
@RequestMapping("/data08")
public void data08(User user){
    System.out.println(user);
}
```

7.7. Set 类型

Set 和 List 类似，也需要绑定在对象上，而不能直接写在 Controller 方法的参数中。但是，绑定Set数据时，必须先在Set对象中add相应的数量的模型对象。

```
public class User {
    private int id;
    private String userName;
    private String userPwd;

    private Set<Phone> phones = new HashSet<Phone>();



    public User() {
        phones.add(new Phone());
        phones.add(new Phone());
        phones.add(new Phone());
    }
    /*public List<Phone> getPhones() {
        return phones;
    }
    public void setPhones(List<Phone> phones) {
        this.phones = phones;
    }*/
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getUserPwd() {
        return userPwd;
    }
    public void setUserPwd(String userPwd) {
        this.userPwd = userPwd;
    }
    public Set<Phone> getPhones() {
        return phones;
    }
    public void setPhones(Set<Phone> phones) {
        this.phones = phones;
    }
}
```

Controller 方法:

```
@RequestMapping("/data09")
public void data09(User user){
    System.out.println(user);
}
```

表单页面

```
<form action="data09" method="post">
   <input name="phones[0].num" value="123456" />
   <input name="phones[1].num" value="4576" />
   <input name="phones[2].num" value="4576" />
   <button type="submit"> 提交</button>
</form>
```

7.8. Map 类型

Map最为灵活，它也需要绑定在对象上，而不能直接写在Controller方法的参数中。

```
public class User {
    private int id;
    private String userName;
    private String userPwd;

    private Set<Phone> phones=new HashSet<Phone>();

    private Map<String, Phone> map=new HashMap<String, Phone>();

    // private List<Phone> phones=new ArrayList<Phone>();


    public User() {
        phones.add(new Phone());
        phones.add(new Phone());
        phones.add(new Phone());
    }
    /*public List<Phone> getPhones() {
        return phones;
    }
    public void setPhones(List<Phone> phones) {
        this.phones = phones;
    }*/
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
```

```
    }
    public String getUserPwd() {
        return userPwd;
    }
    public void setUserPwd(String userPwd) {
        this.userPwd = userPwd;
    }
    public Set<Phone> getPhones() {
        return phones;
    }
    public void setPhones(Set<Phone> phones) {
        this.phones = phones;
    }
    public Map<String, Phone> getMap() {
        return map;
    }
    public void setMap(Map<String, Phone> map) {
        this.map = map;
    }
}
```

Controller 方法

```
@RequestMapping("/data10")
public void data10(User user){
    Set<Entry<String, Phone>>  set = user.getMap().entrySet();
    for(Entry<String, Phone> entry:set){
        System.out.println(entry.getKey()+"--"+entry.getValue().getNum());

    }
}
```

表单页面

```
<form action="data10" method="post">
   <input name="map['1'].num" value="123456" />
   <input name="map['2'].num" value="4576" />
   <input name="map['3'].num" value="4576" />
   <button type="submit"> 提交</button>
</form>
```

8. 请求转发与重定向

SpringMVC 默认采用服务器内部转发的形式展示页面信息。同样也支持重定向页面。

8.1. 重定向

重定向是发一个302的状态码给浏览器，浏览器自己去请求跳转的网页。地址栏会发生改变。

重定向以 redirect: 开头

```
/**
  * 重定向到JSP页面
  * @return
```

```
  */
@RequestMapping(value="/view01")
public String view01(){
    return "redirect:view.jsp";
}

/**
  * 重定向到JSP页面
  *      传递参数
  * @return
  */
@RequestMapping(value="/view02")
public String view02(){
    return "redirect:view.jsp?uname=zhangsan&upwd=123456";
}

/**
  * 重定向到JSP页面
  *      传递参数 （传递中文参数会出现乱码）
  * @return
  */
@RequestMapping(value="/view03")
public String view03(){
    return "redirect:view.jsp?uname=张三&upwd=123456";
}

/**
  *  重定向到JSP页面
  *      传递参数 （通过 RedirectAttributes 对象设置重定向参数，避免中文乱码问题）
  * @param redirectAttributes
  * @return
  */
@RequestMapping(value="/view04")
public String view04(RedirectAttributes redirectAttributes){
    redirectAttributes.addAttribute("uname","张三");
    redirectAttributes.addAttribute("upwd","123456");
    return "redirect:view.jsp";
}


/**
  * 重定向到JSP页面
  *      返回 ModelAndView 对象
  * @param modelAndView
  * @return
  */
@RequestMapping(value="/view06")
public ModelAndView view06(ModelAndView modelAndView){
    modelAndView.addObject("uname","李四");
    modelAndView.addObject("upwd","123321");
    modelAndView.setViewName("redirect:view.jsp");
    return modelAndView;
}

/**
  * 重定向到Controller
  *      返回 ModelAndView 对象
  * @param modelAndView
```

```
  * @return
  */
@RequestMapping(value="/view07")
public ModelAndView view07(ModelAndView modelAndView){
    modelAndView.addObject("uname","admin");
    modelAndView.setViewName("redirect:test01");
    return modelAndView;
}
```

页面中获取参数值

```
${param.参数名}
```

8.2. 请求转发

请求转发，直接调用跳转的页面，让它返回。对于浏览器来说，它无法感觉服务器有没有forward。地址栏不发生改变。可以获取请求域中的数据。

请求转发以 forward: 开头

```
/**
  * 请求转发到JSP页面
  */
@RequestMapping("/view08")
public String view08(){
    return "forward:view.jsp";
}

/**
  * 请求转发到JSP页面
  *       设置参数
  */
@RequestMapping("/view09")
public String view09(){
    return "forward:view.jsp?uname=张三&upwd=123456";
}

/**
  * 请求转发到JSP页面
  *       设置请求域
  */
@RequestMapping("/view10")
public String view10(Model model){
    model.addAttribute("uname","张三");
    return "forward:view.jsp";
}

/**
  * 请求转发到JSP页面 （默认）
  *       默认会去指定目录下找JSP页面 （配置文件中设置的）
  */
@RequestMapping("/view11")
public String view11(){
```

```
    return "/../../view";
}

/**
  * 请求转发到 Controller
  * @return
  */
@RequestMapping("/view12")
public ModelAndView view12(ModelAndView modelAndView){
    modelAndView.setViewName("forward:test01");
    return  modelAndView;
}

/**
  * 请求转发到 Controller
  *      传递参数
  * @return
  */
@RequestMapping("/view13")
public ModelAndView view13(ModelAndView modelAndView){
    modelAndView.setViewName("forward:test01?uname=admin");
    return  modelAndView;
}
```

页面中获取数据

```
获取传递的参数：${param.参数名}
获取请求域的数据：${请求域中设置的名称}
```

9. JSON 数据开发

9.1. 基本概念

Json 在企业开发中已经作为通用的接口参数类型，在页面（客户端）解析很方便。SpringMVC 对于 json 提供了良好的支持，这里需要修改相关配置，添加 json 数据支持功能

9.1.1. @ResponseBody

该注解用于将 Controller 的方法返回的对象，通过适当的 HttpMessageConverter 转换为指定格式后，写入到 Response 对象的 body 数据区。

返回的数据不是 html 标签的页面，而是其他某种格式的数据时（如 json、xml 等）使用（通常用于 ajax 请求）。

9.1.2. @RequestBody

该注解用于读取 Request 请求的 body 部分数据，使用系统默认配置的 HttpMessageConverter 进行解析，然后把相应的数据绑定到要返回的对象上 ，再把 HttpMessageConverter 返回的对象数据绑定到 controller 中方法的参数上。

9.2. 使用配置

9.2.1. 添加 json相关坐标

pom.xml

```
<!-- 添加json 依赖jar包 -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.10.0</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.10.0</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.10.0</version>
</dependency>
```

9.2.2. 修改配置文件

servlet-context.xml

```
<!-- mvc 请求映射 处理器与适配器配置 -->
<mvc:annotation-driven>
    <mvc:message-converters>
        <bean
class="org.springframework.http.converter.StringHttpMessageConverter" />
        <bean
class="org.springframework.http.converter.json.MappingJackson2HttpMessageConvert
er" />
    </mvc:message-converters>
</mvc:annotation-driven>
```

9.2.3. 注解使用

9.2.3.1. @ResponseBody

```
@Controller
@RequestMapping("/user")
public class UserController {

    /**
     * @ResponseBody 返回的是JOSN格式的数据，返回JavaBean对象
     *       注解设置在方法体上
     * @return
     */
    @RequestMapping("queryUser01")
    @ResponseBody
    public User queryUser01(){
        User user = new User();
        user.setId(1);
```

```
        user.setUserName("zhangsan");
        user.setUserPwd("123456");
        // 返回的是user对象
        return  user;
    }

    /**
     * @ResponseBody 返回的是JOSN格式的数据，返回JavaBean对象
     *      注解设置在方法返回对象前，修饰符之后
     * @return
     */
    @RequestMapping("queryUser02")
    public  @ResponseBody User queryUser02(){
        User user = new User();
        user.setId(2);
        user.setUserName("lisi");
        user.setUserPwd("123321");
        // 返回的是user对象
        return  user;
    }

    /**
     * @ResponseBody 返回的是JOSN格式的数据，返回集合
     * @return
     */
    @RequestMapping("/queryUser03")
    @ResponseBody
    public List<User> queryUser03(){

        List<User> list = new ArrayList<>();

        User user01 = new User();
        user01.setId(1);
        user01.setUserName("zhangsan");
        user01.setUserPwd("123456");

        User user02 = new User();
        user02.setId(2);
        user02.setUserName("lisi");
        user02.setUserPwd("123321");

        list.add(user01);
        list.add(user02);

        // 返回的是user集合
        return  list;
    }
}
```

9.2.3.2. @RequestBody

@RequestBody 注解常用来处理 content-type 不是默认的 application/x-www-form-urlcoded 类型的内容，比如说：application/json 或者是application/xml 等。一般情况下来说常用其来处理 application/json 类型。@RequestBody接受的是一个 json 格式的字符串，一定是一个字符串。

通过 @RequestBody 可以将请求体中的 JSON 字符串绑定到相应的 bean 上，当然，也可以将其分别绑定到对应的字符串上。

```java
/**
  *  @RequestBody 规定请求的参数是JOSN格式的字符串
  *      注解设置在形参前面
  * @param user
  * @return
  */
@RequestMapping("/getUser")
@ResponseBody
public User getUser(@RequestBody User user){
    System.out.println(user);
    return user;
}
```

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>JSON处理</title>
    <%-- 引入Jquery的核心JS文件 --%>
    <script type="text/javascript" src="js/jquery-3.4.1.js"></script>
</head>
<body>

    <input type="button" value="JSON数据测试" onclick="test()"/>

    <script type="text/javascript">

        /**
         * 请求传递JSON格式的数据
         * 返回JSON格式的数据
         */
        function test(){
            $.ajax({
                // 请求方式 Get|Post
                type: "post",
                // 请求路径
                url: "user/getUser",
                // 预期服务器返回的额数据类型
                dataType: "json",
                // 设置服务器请求类型的数据类型为JSON格式
                contentType: "application/json;charset=utf-8",
                // 传递给服务器的参数
                data:‘{"userName":"admin","userPwd":"123456"}’,
                // 回调函数，接收服务器返回的响应的结果 （函数中的形参用来接收服务器返回的数据）
                success:function(data){
                    console.log(data);
                }
            })
        }
    </script>
</body>
</html>
```