Controller对象中的方法（即处理器方法或控制器方法）中可以有参数，包含下面四类：

1）HttpServletRequest

2）HttpServletResponse

3）HttpSession

4）请求中携带的参数

这些参数会在系统调用时自动赋值，其中前三类我们可以定义在处理器方法的参数位置，就像之前自己创建Servlet实现类时重写的doGet或doPost方法的参数一样，然后在方法体中使用这些参数类中的取值方法获取参数值。如下面所示

```
@RequestMapping(value = "/test")
public class OtherController {


    @RequestMapping(value = "/otherOne.do",method = RequestMethod.POST)
    public ModelAndView doOtherOne(HttpServletRequest request,
                                   HttpServletResponse response,
                                   HttpSession session) {

        String param = request.getParameter("name");
        System.out.println(param);//zhangshan
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("msg", param);

        modelAndView.setViewName("other");

        return modelAndView;

    }

}
```

如上面所示，我们发送这样的一个请求：

http://localhost:8080//b_spring_webparam_02/test/otherOne.do?name=zhangshan

这样我们就可以通过每个参数类中的获取方法获取到请求给这些参数的值了，这里只演示HttpServletRequest类的参数，其他两类获取是大同小异的。

第四类参数（请求中携带的参数）我们要深入探讨

1、逐个接收参数

我们新建一个项目模块b-spring-webparam-02，将01项目中的都复制到02项目中，具体要修改的如下步骤所示

1）、修改欢迎页面，如下

```
<%--
  Created by IntelliJ IDEA.
  User: 钟荣杰
  Date: 2021/2/14
  Time: 19:23
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Welcome Page</title>
</head>
<body>
    <P>提交参数给Controller，接收请求参数</P>
    <form action="test/otherOne.do" method="post">
        姓名:<input type="text" name="name">
        年老:<input type="text" name="age">
        <input type="submit" value="提交参数">
    </form>
</body>
</html>
```

2）、OtherController类如下：

```
package com.studymyself.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@Controller
@RequestMapping(value = "/test")
public class OtherController {

    /**
     * 逐个接收参数：
     *       要求：处理器（控制器）方法的形参名和请求中的参数名一致
     *             这样同名的请求参数赋值给同名的形式参数
     *
     *     这种方式接收参数，其底层还是调用request对象中的方法实现的：
     *     1、底层使用request对象接收请求参数，获取处理器方法的形参名作为key
     *      String strName = request.getParameter("name");
     *      String strAge = request.getParameter("age");
     *
     *     2、当SpringMVC框架通过执行DispatcherServlet对象的方法来调用OtherController
```

```
     *      类的doOtherOne方法时按名称对应，把接收的参数赋值给方法的形参：
     *      doOtherOne(strName,Integer.valueOf(strAge));
     *
     *      上面request对象接收的参数都是字符串类型的
     *      所以框架给控制器方法中的形参赋值时会根据形参的类型将字符串自动转换为：
     *      int long float double等类型
     */

    @RequestMapping(value = "/otherOne.do",method = RequestMethod.POST)
    public ModelAndView doOtherOne(String name,int age) {
        //当age的输入参数是null或者非数字类型字符串，底层自动转换时会出现错误：405
        //我们数字类型的形参一般定义为Integer类型

        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("myName",name);
        modelAndView.addObject("myAge",age);
        modelAndView.setViewName("other");

        return modelAndView;

    }

}
```

3）、结果页面如下：other.jsp

```
<%--
  Created by IntelliJ IDEA.
  User: 钟荣杰
  Date: 2021/2/15
  Time: 0:34
  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>
<h3>/WEB-INF/view/other.jsp从request作用域中获取数据</h3>
<!--把结果取出来，通过EL表达式${key值}来取值-->
<h3>name数据：${myName}</h3>
<h3>age数据：${myAge}</h3>
</body>
</html>
```

启动Tomcat服务器输入网址进行测试，在测试的时候跳转到other.jsp页面的时候，由于没有修改自动生成的web.xml文件，导致使用的是最低版本的web.xml，使得jsp中的<%@ page isELIgnored=`"true"`%>这个设置默认为true，jsp忽略EL表达式，在输出的时候直接输出${myName}这个字符串，而不是表达式的value值。

4）、我们把欢迎页面的超链接的请求方式设置为POST和doOtherOne方法的请求方式一致，我们获取name参数的时候得到的中文值是乱码，当我们设置为GET的时候并没有乱码，这是我们之前学习的参数传递乱码现象。主要是由于Tomcat并不知道请求所传递的数据是什么编码格式，只是按其默认的解码方式接收解码，所以中文出现乱码，我们可以在Controller类的处理器方法获取参数之前使用request.setCharacterEncoding("UTF-8");语句告诉Tomcat服务器请求中的数据编码是UTF-8就可以了。但是处理器方法过多时，一个一个设置太麻烦，所以我们需要在web.xml配置注册过滤器来全局完成该操作。

5）、使用过滤器。过滤器类可以自己定义，也可以使用框架提供的CharacterEncodingFilter

框架提供的这个过滤器是位于spring-web这个依赖包装中的。我们要在web.xml文件中注册声明过滤器，如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <!--注册SpringMVC的核心对象DispatcherServlet
      这样web容器启动后就会创建这里声明的所有Servlet对象，这里是创建DispatcherServlet对象实例
      原因：
          DispatcherServlet对象在被服务器创建的过程中，同时创建SpringMVC容器对象，spring容器对象的创建
          会读取SpringMVC的配置文件，同时就把文件中声明的bean全部创建好了，到时用户发送请求就可以直接调用
          容器中的对象就好了。就跟前面的监听器一样。
      原理：
      servlet初始化执行init()方法。DispatcherServlet类中的init方法有如下内容{
          //创建web项目的Spring容器
          WebApplicationContext wac = new
ClassPathXmlApplicationContext("springmvc.xml");
          //把容器对象放到ServletContext对象中
          getServletContext().setAttribute(key,wac);
      }
      -->
    <servlet>
        <servlet-name>springMVC</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

        <!--跟Spring中的监听器一样，在创建中央调度器对象的时候为我们创建SpringMVC
            容器对象时读取的spring配置文件默认读取路径是/WEB-INF/springMVC-
springmvc.xml
            所以我们要修改读取路径,记得配置文件中这些标签的添加顺序，不然会报错-->
```

```
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:springmvc.xml</param-value>
        </init-param>

        <!--我们希望在Tomcat启动后创建Servlet对象
            使用load-on-startup标签：表示Tomcat服务器启动后创建Servlet的对象的顺序
            其中的值是整数，单位越小越快创建这个声明的Servlet对象，大于等于0的整数
        -->
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springMVC</servlet-name>
        <!--在使用框架时，url-pattern标签中可以使用两种值

            1、使用扩展名方式，语法*.xxx，其中xxx是自定义的扩展名。常用
*.do,*.action,*.mvc等
            意思是以.do结尾的请求路径的请求都交给名称为<servlet-
name>springMVC</servlet-name>
            这个servlet来处理,如
            http://localhost:8080/JD/some.do
            http://localhost:8080/JD/other.do

            2、第二种就是使用斜杠"/"
            -->
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>

    <!--声明注册过滤器，解决POST请求中文乱码-->
    <filter>
        <filter-name>characterEncodingFilter</filter-name>
        <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <!--CharacterEncodingFilter中有三个属性：
            private String encoding;
            private boolean forceRequestEncoding;//创建该过滤器时这两个布尔属性赋值为
false
            private boolean forceResponseEncoding;
        -->
        <!--该属性设置项目中使用的字符编码方式-->
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <!--该属性为true时强制请求对象（HttpServletRequest）使用encoding属性的值的编码-
->
        <init-param>
            <param-name>forceRequestEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
        <!--该属性为true时强制响应对象（HttpServletResponse）使用encoding属性的值的编
码-->
        <init-param>
            <param-name>forceResponseEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
```

```
        <filter-name>characterEncodingFilter</filter-name>
        <!--将这个过滤器的url-pattern设置为/*
            就是在启动项目后，所有的请求发送后无论是什么请求路径都会
            执行这个过滤器对象，每一次的请求生成的请求对象request和响应对象response
            的编码方式都设置为encoding的值的编码，然后在执行其他对应的Servlet，这样我们
            就不必每一个处理器方法都设置编码方式了。
            -->
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

过滤器CharacterEncodingFilter的父父类是Filter类，主要执行方法的源码如下

```
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
response, FilterChain filterChain) throws ServletException, IOException {
        String encoding = this.getEncoding();
        if (encoding != null) {
            if (this.isForceRequestEncoding() || request.getCharacterEncoding()
== null) {
                request.setCharacterEncoding(encoding);
            }

            if (this.isForceResponseEncoding()) {
                response.setCharacterEncoding(encoding);
            }
        }
```

6）、当我们的形参名和请求的参数名不一致时，使用@RequestParam注解进行形参名和请求参数名的映射

在index.jsp中新加一个超链接和表单，如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Welcome Page</title>
</head>
<body>
    <P>提交参数给Controller，接收请求参数</P>
    <form action="test/otherOne.do" method="post">
        姓名:<input type="text" name="name"><br>
        年龄:<input type="text" name="age"><br>
        <input type="submit" value="提交参数">
    </form>
    <br>
     <P>请求参数名和方法形参名不一样</P>
    <form action="/test/othertwo.do" method="post">
        姓名:<input type="text" name="myname"><br>
        年龄:<input type="text" name="myage"><br>
        <input type="submit" value="提交参数">
    </form>
</body>
</html>
```

我们先在Controller类中新建一个doOtherTwo方法，如下：

```
    /**
     * 请求中参数名和处理器方法中的形参名不一致
     * @RequestParam：解决请求中参数名和形参名不一样的问题
     *  属性：value：请求参数的名字
     *        required：是一个布尔类型，默认是true
     *                  true就表示请求中必须包含value的参数的值
     *                  就是相当于你直接访问
http://localhost:8080//b_spring_webparam_02/test/othertwo.do
     *                  而注解@RequestParam中value的参数没有值，就会报错误400，如下面所
示：
     *                  设置为required=false，就不需要参数myname和参数myage一定要有值
     *  位置：在处理器方法的形参定义的前面添加
     */
    @RequestMapping(value = "/othertwo.do",method = RequestMethod.POST)
    public ModelAndView doOtherTwo(@RequestParam(value = "myname",required =
false) String name,
                                   @RequestParam(value = "myage",required =
false) int age) {

        System.out.println("name:"+name+"\nage:"+age);

        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("myName",name);
        modelAndView.addObject("myAge",age);
        modelAndView.setViewName("other");

        return modelAndView;

    }
```

2、使用对象接收参数

1）、创建一个vo包，在其中创建一个普通类Student，其中有两个私有属性name跟age，如下：

```
package com.studymyself.vo;

/**
 * 保存参数的一个普通类
 */
public class Student {

    //要求是属性名和参数名一致
    private String name;
    private Integer age;

    //无参构造中输出一句话，验证框架是否调用了无参构造创建Student对象
    public Student() {
        System.out.println("====Student无参构造执行====");
    }

    public String getName() {
        return name;
```

```
    }

    //在set方法中输出一句话，验证框架是否调用了set方法给该对象赋值
    public void setName(String name) {
        System.out.println("setName方法执行");
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        System.out.println("setAge方法执行");
        this.age = age;
    }

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

2）、在OtherController类中添加一个处理器方法doOtherThree，如下：

```
/**
     * 处理器方法的形参是Java对象，该对象的属性名和请求参数名一样
     * 框架会使用反射机制自动帮我们创建该Java对象，给属性赋值。
     * 请求参数是name的值，框架会调用setName方法给Java对象的name属性赋值，set注入
     */
    @RequestMapping(value = "/otherthree.do",method = RequestMethod.POST)
    public ModelAndView doOtherThree(Student student) {

        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("myName",student.getName());
        modelAndView.addObject("myAge",student.getAge());
        modelAndView.setViewName("other");

        return modelAndView;

    }
```

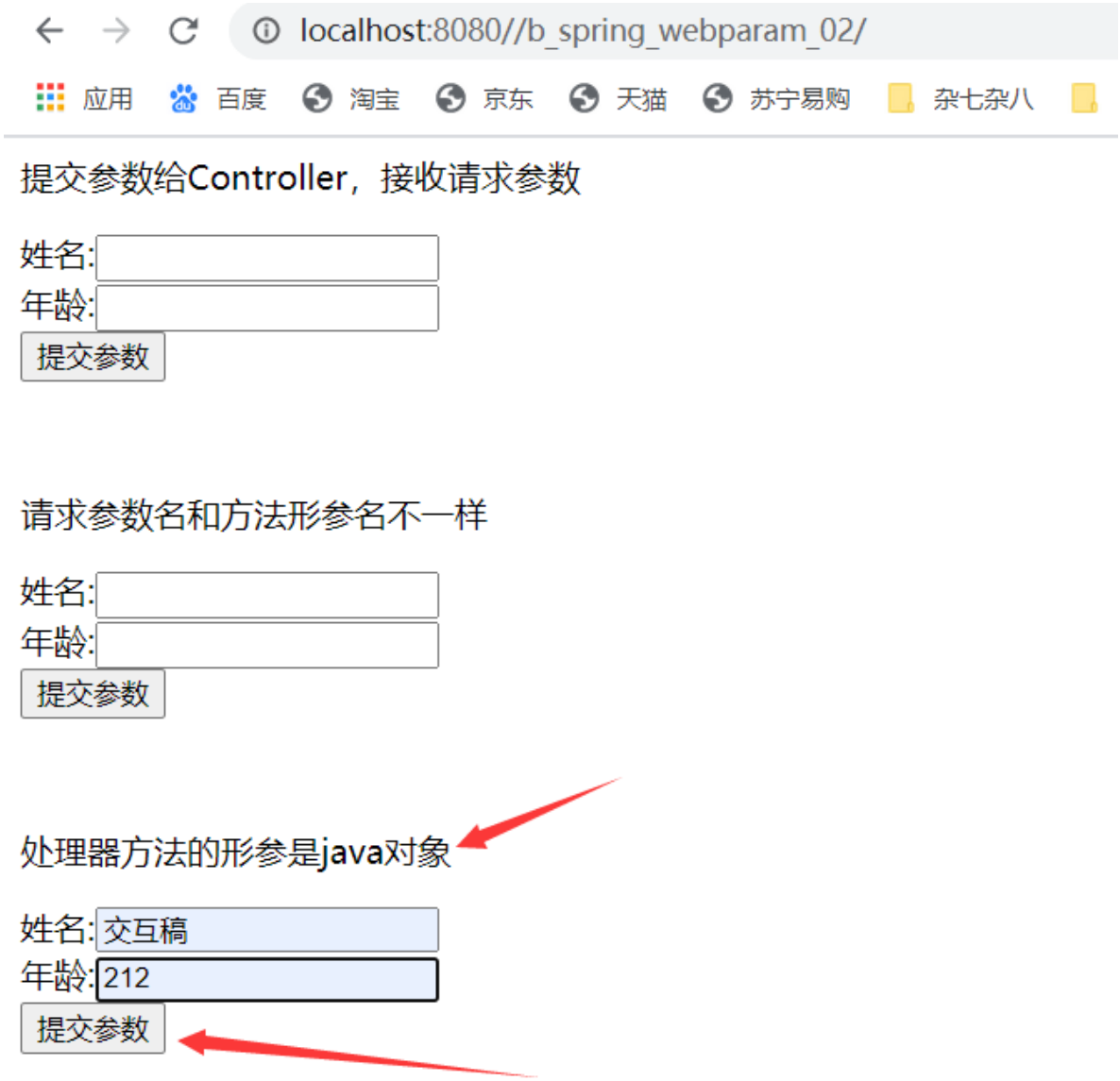
3）、在index.jsp页面中添加

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Welcome Page</title>
</head>
<body>
    <P>提交参数给Controller，接收请求参数</P>
    <form action="test/otherOne.do" method="post">
```

```
        姓名:<input type="text" name="name"><br>
        年龄:<input type="text" name="age"><br>
        <input type="submit" value="提交参数">
    </form>
    <br>
    <P>请求参数名和方法形参名不一样</P>
    <form action="test/othertwo.do" method="post">
        姓名:<input type="text" name="myname"><br>
        年龄:<input type="text" name="myage"><br>
        <input type="submit" value="提交参数">
    </form>
    <br>
    <P>处理器方法的形参是java对象</P>
    <form action="test/otherthree.do" method="post">
        姓名:<input type="text" name="name"><br>
        年龄:<input type="text" name="age"><br>
        <input type="submit" value="提交参数">
    </form>
</body>
</html>
```

启动服务器，进行测试，发现index.jsp中的参数名和java对象的属性名不一致，没能成功接收参数，修改过来再进行测试

结果如图：



成功获取参数到Student对象

框架自动创建Student对象，调用set方法把获取的参数按照名字进行属性赋值，使用的是反射机制。

处理器方法值也可以定义多个对象进行参数的接收，非常方便