# 处理器方法的返回值类型

## 使用注解@Controller的处理器的处理器方法返回值一共有四种：

第一种：ModelAndView
第二种：String
第三种：无返回值
第四种：返回自定义类型对象

## 3、处理器方法返回void（了解）

不能表示数据，也不能表示视图。
但是可以：
　　在处理Ajax请求时可以使用void返回。通过HttpServletResponse对象来输出数据，来响应Ajax请求。Ajax请求服务器返回的就是数据，和视图无关

## 4、处理器方法返回值为Object对象

　　处理器方法返回Object对象，可以是Integer，String，自定义对象，Map，List等。返回的对象并不是作为逻辑视图出现的，而是作为数据直接在页面显示的。
　　返回对象需要使用@ResponseBody注解，表示是将转换后的Json数据放入到响应体中。
--步骤：
　　--现在前端都是Ajax请求，用的就是jSON格式数据。返回的对象怎么处理，如下所示：
1、要将对象转换成json数据，需要用到json工具库，所以要加入json工具库的依赖，而SpringMVC默认使用的工具库是Jackson
2、框架自动帮我们进行对象到json的转换，需要我们在springMVC配置文件中加入<mvc:annotation-driven>标签，叫做注解驱动。这样就不需要我们手动的在控制器方法中将对象转换成json数据了。
　　就是这步：json = om.writeValueAsString(java对象);
3、数据是输出到页面上的，所以我们在控制器方法上添加注解@ResponseBody，作用就相当于在控制器中执行下面代码：

```
response.setContentType("application/json;charset=UTF-8");//告诉浏览器传的是UTF-8编码的json数据
PrintWriter out = response.getWriter();
out.println(json);
```

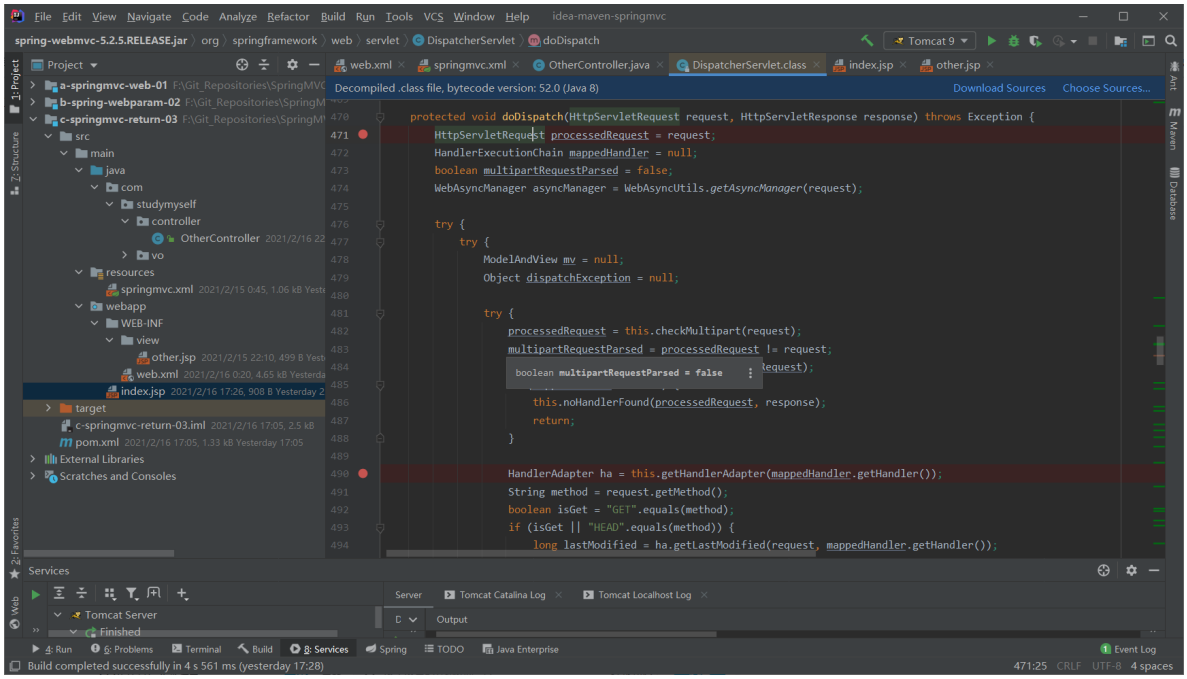
--框架原理：
SpringMVC处理器方法返回java对象，转为json输出到浏览器，响应Ajax请求的原理：

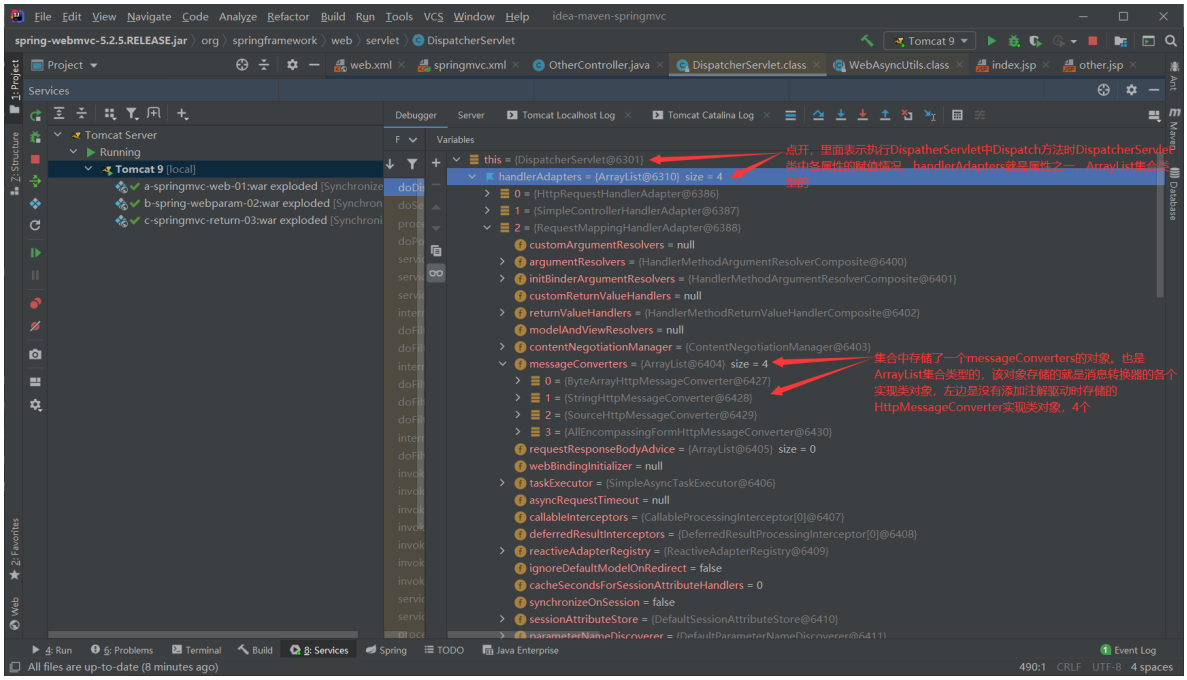
1、-- <mvc:annotation-driven>标签:注解驱动
　　注解驱动的功能：完成java对象到json、xml、text、二进制数据格式的转换
　　<mvc:annotation-driven>在加入到SpringMVC配置文件中后，会自动创建HttpMessageConverter接口的七个实现类对象。
　　HttpMessageConverter接口：叫做--消息转换器
--接口功能：
　　里面定义了java对象转为json，xml等数据格式的方法。该接口有很多的实现类，每个实现类实现的是某种数据格式转换。经常用到的有：
　　StringHttpMessageConverter（处理字符串数据框架用到的）和
　　MappingJackson2HttpMessageConverter（这个是将Java对象转为json格式的实现类，用到的工具库是Jackson）

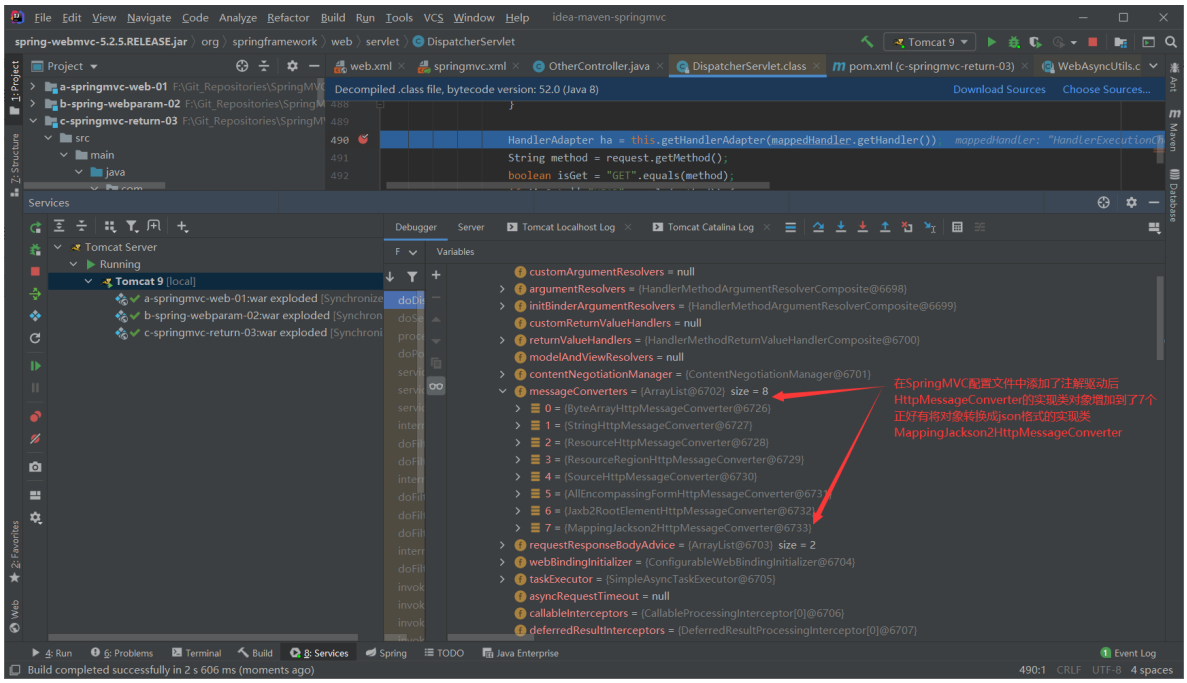
```
//HttpMessageConverter接口源码：
public interface HttpMessageConverter<T> {
    //该方法和下面的read方法是一组，代表法请求时用到的操作方法，现在不需要
    boolean canRead(Class<?> var1, @Nullable MediaType var2);

    //该方法和下面的write方法是一组，代表的是控制器类中输出结果到浏览器用到的方法
    boolean canWrite(Class<?> var1, @Nullable MediaType var2);

    List<MediaType> getSupportedMediaTypes();//该方法不需要关心

    T read(Class<? extends T> var1, HttpInputMessage var2) throws IOException,
HttpMessageNotReadableException;

    void write(T var1, @Nullable MediaType var2, HttpOutputMessage var3) throws
IOException, HttpMessageNotWritableException;
}
//1）、canWrite方法功能是检查处理器方法返回的java对象能不能var2表示的数据格式
    var2是MediaType类型的，该类中定义了很多中的数据格式，json、xml等等。当我们的处理方法如
下：

    @RequestMapping(value = "/returnString-view1.do",method =
RequestMethod.POST)
    public String  doReturnView1(String name,int age,HttpServletRequest request)
{
        Student stu = new Student(name,age);
        return "stu";
    此时canWrite方法就会检查stu对象能否转换成var2代表的数据格式，先检查的是能否转为json格
式，一旦有个格式能够转换，canWrite方法就停止检查，返回true，否者返回false。
//2）、当canWrite方法返回true，write方法执行，就是把处理器方法返回的对象，调用Jackson中的
ObjectMapper类中的方法转为json字符串
```

```
2、--@ResponseBody注解：放在返回对象的控制器方法上
框架底层执行如下：
    response.setContentType("application/json;charset=UTF-8");//告诉浏览器传的是
UTF-8编码的json数据
    PrintWriter out = response.getWriter();
    out.println(json);
```

```
我们在DispatcherServlet类中的doDispatch方法中添加断点Debug项目：找到框架运行项目时中央调度
器DispatcherServlet中存储消息转换器的属性（ArrayList集合类型）
```


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
idea-maven-springmvc
spring-webmvc-5.2.5.RELEASE.jar org springframework web servlet DispatcherServlet doDispatch
Tomcat 9
Project
web.xml springmvc.xml OtherController.java DispatcherServlet.class index.jsp other.jsp
Decompiled .class file, bytecode version: 52.0 (Java 8)
Download Sources Choose Sources...
protected void doDispatch(HttpServletRequest request, HttpServletResponse response) throws Exception {
HttpServletRequest processedRequest = request;
HandlerExecutionChain mappedHandler = null;
boolean multipartRequestParsed = false;
WebAsyncManager asyncManager = WebAsyncUtils.getAsyncManager(request);
try {
try {
ModelAndView mv = null;
Object dispatchException = null;
try {
processedRequest = this.checkMultipart(request);
multipartRequestParsed = processedRequest != request;
boolean multipartRequestParsed = false
this.noHandlerFound(processedRequest, response);
return;
}
HandlerAdapter ha = this.getHandlerAdapter(mappedHandler.getHandler());
String method = request.getMethod();
boolean isGet = "GET".equals(method);
if (isGet || "HEAD".equals(method)) {
long lastModified = ha.getLastModified(request, mappedHandler.getHandler());
Services
Server Tomcat Catalina Log Tomcat Localhost Log
Tomcat Server
Finished
Output
4: Run 6: Problems Terminal Build 8: Services Spring TODO Java Enterprise
Event Log
Build completed successfully in 4 s 561 ms (yesterday 17:28)
471:25 CRLF UTF-8 4 spaces


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
idea-maven-springmvc
spring-webmvc-5.2.5.RELEASE.jar org springframework web servlet DispatcherServlet
Tomcat 9
web.xml springmvc.xml OtherController.java DispatcherServlet.class WebAsyncUtils.class index.jsp other.jsp
Services
Debugger Server Tomcat Localhost Log Tomcat Catalina Log
Variables
Tomcat Server
Running
Tomcat 9 [local]
a-springmvc-web-01:war exploded
b-spring-webparam-02:war exploded
c-springmvc-return-03:war exploded
this = {DispatcherServlet@6301}
handlerAdapters = {ArrayList@6310} size = 4
0 = {HttpRequestHandlerAdapter@6386}
1 = {SimpleControllerHandlerAdapter@6387}
2 = {RequestMappingHandlerAdapter@6388}
customArgumentResolvers = null
argumentResolvers = {HandlerMethodArgumentResolverComposite@6400}
initBinderArgumentResolvers = {HandlerMethodArgumentResolverComposite@6401}
customReturnValueHandlers = null
returnValueHandlers = {HandlerMethodReturnValueHandlerComposite@6402}
modelAndViewResolvers = null
contentNegotiationManager = {ContentNegotiationManager@6403}
messageConverters = {ArrayList@6404} size = 4
0 = {ByteArrayHttpMessageConverter@6427}
1 = {StringHttpMessageConverter@6428}
2 = {SourceHttpMessageConverter@6429}
3 = {AllEncompassingFormHttpMessageConverter@6430}
requestResponseBodyAdvice = {ArrayList@6405} size = 0
webBindingInitializer = null
taskExecutor = {SimpleAsyncTaskExecutor@6406}
asyncRequestTimeout = null
callableInterceptors = {CallableProcessingInterceptor[0]@6407}
deferredResultInterceptors = {DeferredResultProcessingInterceptor[0]@6408}
reactiveAdapterRegistry = {ReactiveAdapterRegistry@6409}
ignoreDefaultModelOnRedirect = false
cacheSecondsForSessionAttributeHandlers = 0
synchronizeOnSession = false
sessionAttributeStore = {DefaultSessionAttributeStore@6410}
点开，里面表示执行DispatherServlet中Dispatch方法时DispatcherServlet类中各属性的赋值情况，handlerAdapters就是属性之一，ArrayList集合类型的
集合中存储了一个messageConverters的对象。也是ArrayList集合类型的，该对象存储的就是消息转换器的各个实现类对象，左边显没有添加注解驱动时存储的HttpMessageConverter实现类对象，4个
4: Run 6: Problems Terminal Build 8: Services Spring TODO Java Enterprise
Event Log
All files are up-to-date (8 minutes ago)
490:1 CRLF UTF-8 4 spaces


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
idea-maven-springmvc
spring-webmvc-5.2.5.RELEASE.jar org springframework web servlet DispatcherServlet
Tomcat 9
web.xml springmvc.xml OtherController.java DispatcherServlet.class pom.xml (c-springmvc-return-03) WebAsyncUtils.c
Decompiled .class file, bytecode version: 52.0 (Java 8)
Download Sources Choose Sources...
HandlerAdapter ha = this.getHandlerAdapter(mappedHandler.getHandler())
String method = request.getMethod();
boolean isGet = "GET".equals(method);
Services
Debugger Server Tomcat Localhost Log Tomcat Catalina Log
Variables
Tomcat Server
Running
Tomcat 9 [local]
a-springmvc-web-01:war exploded
b-spring-webparam-02:war exploded
c-springmvc-return-03:war exploded
customArgumentResolvers = null
argumentResolvers = {HandlerMethodArgumentResolverComposite@6698}
initBinderArgumentResolvers = {HandlerMethodArgumentResolverComposite@6699}
customReturnValueHandlers = null
returnValueHandlers = {HandlerMethodReturnValueHandlerComposite@6700}
modelAndViewResolvers = null
contentNegotiationManager = {ContentNegotiationManager@6701}
messageConverters = {ArrayList@6702} size = 8
0 = {ByteArrayHttpMessageConverter@6726}
1 = {StringHttpMessageConverter@6727}
2 = {ResourceHttpMessageConverter@6728}
3 = {ResourceRegionHttpMessageConverter@6729}
4 = {SourceHttpMessageConverter@6730}
5 = {AllEncompassingFormHttpMessageConverter@6731}
6 = {Jaxb2RootElementHttpMessageConverter@6732}
7 = {MappingJackson2HttpMessageConverter@6733}
requestResponseBodyAdvice = {ArrayList@6703} size = 2
webBindingInitializer = {ConfigurableWebBindingInitializer@6704}
taskExecutor = {SimpleAsyncTaskExecutor@6705}
asyncRequestTimeout = null
callableInterceptors = {CallableProcessingInterceptor[0]@6706}
deferredResultInterceptors = {DeferredResultProcessingInterceptor[0]@6707}
在SpringMVC配置文件中添加了注解驱动后HttpMessageConverter的实现类对象增加到了7个正好有将对象转换成json格式的实现类MappingJackson2HttpMessageConverter
4: Run 6: Problems Terminal Build 8: Services Spring TODO Java Enterprise
Event Log
Build completed successfully in 2 s 606 ms (moments ago)
490:1 CRLF UTF-8 4 spaces

```
结论：
    --当SpringMVC配置文件中没有添加注解驱动时，框架实现消息转换器的类的对象如下：
org.springframework.http.converter.ByteArrayHttpMessageConverter@14e28ebc,
org.springframework.http.converter.StringHttpMessageConverter@7aed0194,
org.springframework.http.converter.xml.SourceHttpMessageConverter@5264b5d8,
org.springframework.http.converter.support.AllEncompassingFormHttpMessageConvert
er@1357adea
    --当添加了注解驱动后（前提的添加了Jackson依赖），实现的消息转换器类增加到了8个，我们用到的
就是最后一个：
org.springframework.http.converter.ByteArrayHttpMessageConverter@6533222f,
org.springframework.http.converter.StringHttpMessageConverter@7f6e4bfd,
org.springframework.http.converter.ResourceHttpMessageConverter@680416cb,
org.springframework.http.converter.ResourceRegionHttpMessageConverter@4ace467b,
org.springframework.http.converter.xml.SourceHttpMessageConverter@36902757,
org.springframework.http.converter.support.AllEncompassingFormHttpMessageConvert
er@3baca3db,
org.springframework.http.converter.xml.Jaxb2RootElementHttpMessageConverter@73d6
482b,
org.springframework.http.converter.json.MappingJackson2HttpMessageConverter@4b93
25b4
```

## Jackson依赖：

```
    <!--添加Jackson依赖，SpringMVC需要使用其进行json格式的转换-->
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-core</artifactId>
      <version>2.10.2</version>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.10.2</version>
    </dependency>
```

## 5、在项目03中新建一个MyController类编写一个方法进行测试

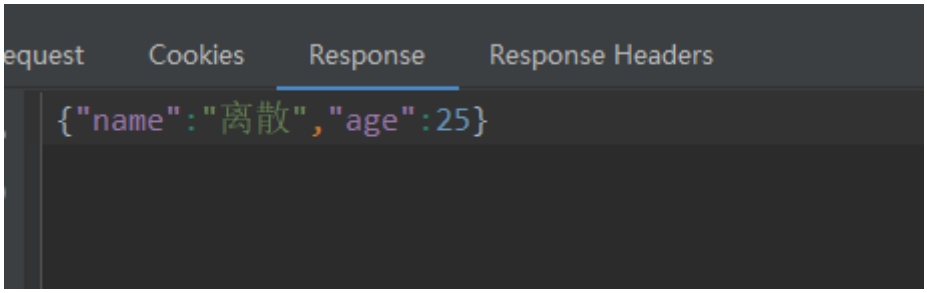
```
package com.studymyself.controller;

import com.studymyself.vo.Student;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;


@Controller
@RequestMapping("/test")
public class MyController {

 /**
     * 处理方法返回一个Student对象，通过框架转为json，响应Ajax请求
     * @ResponseBody：
     *       作用：把处理器方法的返回对象转为的json字符串，通过HttpServletResponse
     *             输出给浏览器
```

```
     *       位置：写在方法的上面，与其他注解无先后顺序关系
     *
     *  返回对象时框架的处理流程：
     *       1、框架用返回的Student对象，调用框架中ArrayList<HttpMessageConverter>中的
每个实现类的canWrite
     *       方法检测哪个实现类能够处理Student类型的数据-
MappingJackson2HttpMessageConverter
     *       2、然后框架调用该MappingJackson2HttpMessageConverter的write方法，
     *       把返回的Student对象中各个属性的信息转为json字符串，用的是Jackson的
ObjectMapper类
     *       中的方法，其中会设置 content=application/json;charset=utf-8
     *       3、最后框架调用@ResponseBody把2的结果数据输出到浏览器，请求完成
     * @return
     */

    @RequestMapping(value = "/returnJson.do",method = RequestMethod.GET)
    @ResponseBody
    public Student  doStudentJson() {
        Student student = new Student("离散",25);
        return student;

    }

}
```

index.jsp如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Welcome Page</title>
</head>
<body>
    <P>处理器方法返回String，表示视图名称</P>
    <form action="test/returnString-view1.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>处理器方法返回String，表示视图完整路径</P>
    <form action="test/returnString-view2.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>处理器方法java对象，输出对象的json字符串到浏览器</P>
    <form action="test/returnJson.do" method="get">
        <input type="submit" value="提交参数">
    </form>
</body>
</html>
```

结果数据页面如下：




## 6、在MyController类中新建一个方法，返回值为List，代码如下

```
    /**
     *  处理器方法返回值是List集合
     *  @ResponseBody：
     *      作用：把处理器方法的返回对象转为的json字符串，通过HttpServletResponse
     *           输出给浏览器
     *     位置：写在方法的上面，与其他注解无先后顺序关系
     *
     * 返回对象时框架的处理流程：
     *       1、框架用返回的List<Student>对象，调用框架中
ArrayList<HttpMessageConverter>中的每个实现类的canWrite
     *          方法检测哪个实现类能够处理Student类型的数据-
MappingJackson2HttpMessageConverter
     *       2、然后框架调用该MappingJackson2HttpMessageConverter的write方法，
     *          把返回的Student对象中各个属性的信息转为json字符串的ArrayJson数组，用的是
Jackson的ObjectMapper类
     *          中的方法，其中会设置 content=application/json;charset=utf-8
     *       3、最后框架调用@ResponseBody把2的结果数据输出到浏览器，请求完成
     * @return
     */
    @RequestMapping(value = "/returnListJson.do",method = RequestMethod.GET)
    @ResponseBody
    public List<Student> doStudentListJson() {

        Student student1 = new Student("离散",25);
        Student student2 = new Student("李四",99);

        List<Student> studentList = new ArrayList<>();
        studentList.add(student1);
        studentList.add(student2);

        return studentList;

    }
```

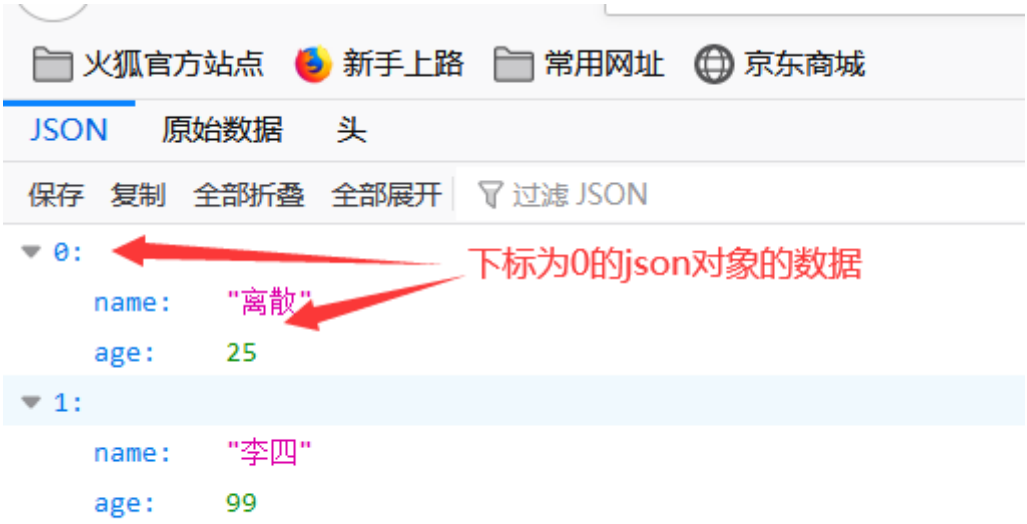
index.jsp中添加超链接

```
<br>
    <P>处理器方法返回List</P>
    <form action="test/returnListJson.do" method="get">
        <input type="submit" value="提交参数">
    </form>
```

启动Tomcat服务器，到欢迎页面点击执行doStudentListJson的链接，或者直接输入地址

http://localhost:8080/c_springmvc_return_03/test/returnListJson.do

结果图如下


可以看到输出的是一个数组，框架把List集合转换成了Json数组，所以我们可以看到有下标。而且我们添加对象到集合中的顺序和json数组中输出的顺序是一致的

还有一点是，虽然处理器方法可以返回map集合，但是需要key，又不能排序，很麻烦，不常用。随意推荐使用List集合来存储对个对象进行返回

## 7、在MyController类中新建一个方法，是返回值是String类型的对象，如下：

```
  /**
     * 处理器方法返回的是String，这里的String表示数据，不是逻辑视图
     * 区分返回的是String是数据还是视图，就看方法上面有没有@ResponseBody注解
     * 有该注解，框架底层把返回的字符串输出到浏览器上，没有就会调用跳转资源的方法
     *
     *  默认使用 text/plain;charset=ISO-8859-1 作为contentType
     *  这样返回的数据会出现中文乱码，需要设置@ResponseMapping注解中的produces属性，
     *  指定新的contentType,此时注解中的value就不能省略了
     * @return
     */
    @RequestMapping(value = "/returnStringData.do",produces =
"text/plain;charset=UTF-8")
    @ResponseBody
    public String doStringData(){
        return "SpringMVC的控制器方法返回String对象：表示数据";
    }
```

测试的时候，由于该方法的路径映射值没有在后面添加.do，导致在输入URL：

http://localhost:8080/c_springmvc_return_03/test/returnStringData

的时候，这样的请求并没有交给中央调度器，因为我们声明的中央调度器是处理.do结尾的请求。