JSP页面的转发和重定向

Spring MVC默认以转发的形式响应JSP，也可以自主进行设置。

重定向：

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

转发：

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

等于

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

Spring MVC数据绑定

什么是数据绑定？在后台业务方法中，直接获取前端HTTP请求中参数的形式，将HTTP请求中的参数之间绑定到业务方法参数列表的方式就是数据绑定。

底层原理：HTTP请求传输的参数都是String类型，但是Handler的业务方法中的参数都是开发者指定的数据类型，如int、Object等，所以需要处理参数的类型转换。具体的转换工作不需要开发者去完成，Spring MVC的HandlerAdapter组件会在执行Handler业务方法之前，完成参数的绑定。

基本数据类型

以int为例，后台需要int类型的参数，直接在业务方法定义处添加int类型的形参即可，HTTP请求参数名称要与形参保持一致。

@ResponseBody 注解直接将数据以字符串的形式响应给客户端。

```
@RequestMapping("/baseType")
@ResponseBody
public String baseType(@RequestParam("id") int num){
    System.out.println(num+1);
    return "num:"+num;
}
```

包装类

```
@RequestMapping("/packageType")
@ResponseBody
public String packageType(@RequestParam(value = "num",required =
false,defaultValue = "1") Integer num){
    return "num:"+num;
}
```

@RequestParam:

- value = "num"：将HTTP请求中名为num的参数与形参进行映射。
- required = false：num参数为非必填项，可以省略。
- defaultValue = "1"：若HTTP请求中没有num参数，默认值为1.

数组

```
@RequestMapping("/arrayType")
@ResponseBody
public String arrayType(String[] name){
    StringBuffer stringBuffer = new StringBuffer();
    for(String item:name){
        stringBuffer.append(item).append(" ");
    }
    return "name:"+stringBuffer.toString();
}
```

List

Spring MVC不支持List类型的直接转换，需要包装成Object。

List的自定义包装类：

```
public class UserList {
    private List<User> users;

    public List<User> getUsers() {
        return users;
    }

    public void setUsers(List<User> users) {
        this.users = users;
    }
}
```

业务方法：

```
@RequestMapping("/listType")
@ResponseBody
public String listType(UserList userList){
    StringBuffer stringBuffer = new StringBuffer();
    for(User user:userList.getUsers()){
        stringBuffer.append(user).append(" ");
    }
    return "用户："+stringBuffer.toString();
}
```

创建addList.jsp，同时添加3个用户信息，input的name指向自定义包装类UserList的users属性，及联到name和age，同时以下标区分集合中不同的对象。

```
<form action="/data/listType" method="post">
    用户1ID：<input type="text" name="users[0].id"/><br/>
    用户1姓名：<input type="text" name="users[0].name"/><br/>
    用户2ID：<input type="text" name="users[1].id"/><br/>
    用户2姓名：<input type="text" name="users[1].name"/><br/>
    用户3ID：<input type="text" name="users[2].id"/><br/>
    用户3姓名：<input type="text" name="users[2].name"/>
    <input type="submit" value="提交"/>
</form>
```

@ResponseBody出现中文乱码的解决方法，在springmvc.xml中配置消息转换器。

```
<mvc:annotation-driven>
    <!-- 消息转换 -->
    <mvc:message-converters>
        <bean
class="org.springframework.http.converter.StringHttpMessageConverter">
            <property name="supportedMediaTypes"
value="text/html;charset=UTF-8"></property>
        </bean>
    </mvc:message-converters>
</mvc:annotation-driven>
```

Map

自定义包装类：

```
public class UserMap {
    private Map<String,User> users;

    public Map<String, User> getUsers() {
        return users;
    }

    public void setUsers(Map<String, User> users) {
        this.users = users;
    }
}
```

业务方法：

```
@RequestMapping("/mapType")
@ResponseBody
public String mapType(UserMap userMap){
    StringBuffer stringBuffer = new StringBuffer();
    for (String key:userMap.getUsers().keySet()){
        User user = userMap.getUsers().get(key);
        stringBuffer.append(user).append(" ");
    }
    return "用户:"+stringBuffer.toString();
}
```

addMap.jsp

```
<form action="/data/mapType" method="post">
    用户1ID：<input type="text" name="users['a'].id"/><br/>
    用户1姓名：<input type="text" name="users['a'].name"/><br/>
    用户2ID：<input type="text" name="users['b'].id"/><br/>
    用户2姓名：<input type="text" name="users['b'].name"/><br/>
    用户3ID：<input type="text" name="users['c'].id"/><br/>
    用户3姓名：<input type="text" name="users['c'].name"/><br/>
    <input type="submit" value="提交"/>
</form>
```

Set

需要封装自定义包装类，将Set集合作为属性，不同的是，使用Set集合，需要在包装类构造函数中，为Set集合添加初始化对象。

```
public class UserSet {
    private Set<User> userSet = new HashSet<>();

    public UserSet(){
        userSet.add(new User());
        userSet.add(new User());
        userSet.add(new User());
    }
}
```

业务方法：

```
@RequestMapping("/setType")
@ResponseBody
public String setType(UserSet userSet){
    StringBuffer stringBuffer = new StringBuffer();
    for (User user:userSet.getUserSet()){
        stringBuffer.append(user).append(" ");
    }
    return "用户:"+stringBuffer.toString();
}
```

addSet.jsp

```
<form action="/data/setType" method="post">
    用户1ID：<input type="text" name="users[0].id"/><br/>
    用户1姓名：<input type="text" name="users[0].name"/><br/>
    用户2ID：<input type="text" name="users[1].id"/><br/>
    用户2姓名：<input type="text" name="users[1].name"/><br/>
    用户3ID：<input type="text" name="users[2].id"/><br/>
    用户3姓名：<input type="text" name="users[2].name"/><br/>
    <input type="submit" value="提交"/>
</form>
```

JSON

JSP:Ajax请求后台业务方法，并将json格式的参数传给后台。

```
<script type="text/javascript">
    var user = {
        "id":1,
        "name":"张三"
    };
$.ajax({
    url:"/data/jsonType",
    data:JSON.stringify(user),
    type:"post",
    contentType:"application/json;charset=UTF-8",
    dataType:"JSON",
    success:function (data) {
        alert(data.name+"---"+data.id);
    }
});
</script>
```

Spring MVC会拦截静态资源的请求，需要在web.xml中进行配置。

```
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.js</url-pattern>
</servlet-mapping>
```

业务方法：

```
@RequestMapping("/jsonType")
@ResponseBody
public User jsonType(@RequestBody User user){
    user.setId(user.getId()+10);
    return user;
}
```

@RequestBody：读取HTTP请求参数，通过Spring MVC提供的HttpMessageConverter接口将读取的参数转为json，xml格式的数据，绑定到业务方法的形参中。

@ResponseBody：将业务方法返回的对象，通过HttpMessageConverter接口转为指定格式的数据，json、xml等，响应给客户端。

fastjson来完成转换工作，fastjson的优势在于如果属性为空就不会将其转化为json，数据会简洁很多。

如何使用fastjson

- pom.xml中引入fastjson的依赖。

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.18</version>
</dependency>
```

- springmvc.xml中配置fastjson。

```
<mvc:annotation-driven>
    <!-- 消息转换 -->
    <mvc:message-converters>
        <bean
class="org.springframework.http.converter.StringHttpMessageConverter">
            <property name="supportedMediaTypes"
value="text/html;charset=UTF-8"></property>
        </bean>
        <!-- fastjson -->
        <bean
class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter4">
</bean>
    </mvc:message-converters>
</mvc:annotation-driven>
```