# 1. JSON

## 1.1. JSON 数据交互

Spring 提供了 `HttpMessageConverter<T>` 接口，用于将请求信息中的数据转换为一个类型为 T 的对象，并将该对象绑定到请求处理方法的参数中。

Spring为 `HttpMessageConverter<T>` 接口提供了许多实现类,用于实现对不同类型的数据进行信息转换。其中 `MappingJackson2HttpMessageConverter` 是 SpringMVC 默认处理 JSON 格式请求响应的实现类,需要 Jackson 开源包的支持。

JSON是特定格式的字符串，是前后端数据交互时使用的一种字符串格式，是目前最流行的一种字符串格式

JSON使用"{}"表示对象，使用"[]"表示数组，所有对象的属性使用双引号包裹，常见格式:
JSON对象1: `{"name":"zhang","age":20,"addr":"青岛"}`
JSON对象2: `{"name":"zhang","age":20,"info":{"certi":37028312341234,"no":111111}}`
JSON数组: `[{"name":"zhang","age":20,"addr":"青岛"},{"name":"li","age":18,"addr":"济南"}]`

## 1.2 前后端使用JSON字符串交互，传统方式

前台发ajax空请求-->后台处理请求并返回对象-->前台解析收到的对象

传统方式：使用HttpServletResponse把JSON字符串写到客户端

注意JavaScript中的2个方法:
JSON.stringify()将JavaScript对象转换为JSON字符串
JSON.parse()　将JSON字符串转为JavaScript对象

1）编写页面，发送ajax请求，并显示收到的用户信息的email

```
<script src="js/jquery-3.2.1.js"></script>
<script type="text/javascript">
$(function() {
   $("#click").click(function(){
    $.ajax({
      url:"f1",
      type:"post",
      success:function(data){alert(JSON.parse(data).email);}
    });
   });
});
</script>

<button id="click">click</button>
```

2）处理器中编写请求处理方法，使用HttpServletResponse把JSON字符串写到客户端

```
@RequestMapping("/f1")
public void f1(HttpServletResponse res) throws IOException {
  String str1 = "{\"name\":\"zhang\",\"email\":\"123@163.com\"}"  ;
  System.out.println(str1);
  res.getWriter().write(str1);
}
```

---

## 1.3 前后端使用JSON字符串交互，`@ResponseBody`注解

前台发ajax空请求-->后台处理请求并返回对象-->前台解析收到的对象

@ResponseBody方式：使用@ResponseBody注解返回值
@ResponseBody注解可以放到返回值前面
@ResponseBody注解也可以放到@RequestMapping上面

1）编写页面，发送ajax请求，并显示收到的用户信息的email

```
<script src="js/jquery-3.2.1.js"></script>
<script type="text/javascript">
$(function() {
   $("#click").click(function(){
    $.ajax({
      url:"f2",
      type:"post",
      success:function(data){alert(JSON.parse(data).email);}
    });
   });
});
</script>
<button id="click">click</button>
```

2）处理器中编写请求处理方法，返回JSON字符串到客户端

```
@RequestMapping("/f2")
public @ResponseBody String f2() {
  String str1 = "{\"name\":\"zhang\",\"email\":\"456@163.com\"}"  ;
  System.out.println(str1);
  return str1;
}
```

---

## 1.4 前后端使用JSON字符串交互, 3方jar包jackson

前台发ajax空请求-->后台处理请求并返回对象-->前台解析收到的对象

1）pom.xml中导入JSON依赖的第三方jar包

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.3</version>
</dependency>
```

2）SpringMVC配置文件中，配置mvc:annotation-driven

```
<mvc:annotation-driven></mvc:annotation-driven>
```

3）编写页面，发送ajax请求，并显示收到的用户信息的email

```
<script src="js/jquery-3.2.1.js"></script>
<script type="text/javascript">
$(function() {
   $("#click").click(function(){
    $.ajax({
      url:"f3",
      type:"post",
      success:function(data){alert(data.email);}
    });
   });
});
</script>
<button id="click">click</button>
```

4）处理器中编写请求处理方法，返回User对象到客户端（jackson会把User对象转为JSON字符串）

```
@RequestMapping("/f3")
public @ResponseBody User f3() {
  User u = new User();
  u.setId(1);
  u.setName("admin");
  u.setPwd("123");
  u.setEmail("123456@qq.com");
  u.setRole("管理员");
  return u;
}
```

---

27、前台发送JSON对象4

1）pom.xml中导入JSON依赖的第三方jar包

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.3</version>
</dependency>
```

2）SpringMVC配置文件中，配置mvc:annotation-driven

```
<mvc:annotation-driven></mvc:annotation-driven>
```

3）编写页面，发送ajax请求，并显示收到的用户信息的email

4）处理器中编写请求处理方法，使用@RequestBody注解参数User可以获取前台传来的name、pwd

```
@RequestMapping("/f4")
public @ResponseBody User f4(@RequestBody User u) {
  u.setId(1);
  u.setEmail("654321@qq.com");
  u.setRole("管理员");
  System.out.println(u);
  return u;
}
```

---

# 2. RESTful

## 2.1. 添加支持

SpringMVC 默认接收不了 PUT 方式的请求，需要在 `web.xml` 配置 `HttpPutFormContentFilter` 过滤器

```
<filter>
    <filter-name>hiddenHttpMethodFilter</filter-name>
    <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>hiddenHttpMethodFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

## 2.2. 添删改查

传统 URL

- 添：`http://localhost:8080/addUser` post
- 删：`http://localhost:8080/deleteUser` get
- 改：`http://localhost:8080/ModifyUser` post
- 查：`http://localhost:8080/queryUser?id=1` get

REST 风格的URL

- 添：`http://localhost:8080/user` post
- 删：`http://localhost:8080/user` put
- 改：`http://localhost:8080/user/1` delete
- 查：`http://localhost:8080/user` get

#### 传递参数

- `RequestMapping` 中使用 `value = "/fun/{param}"`
- 方法中定义参数 `@PathVariable` 获取