Ajax 技术实战

主要内容

Ajax 简介

Ajax 的使用

JSON 详解

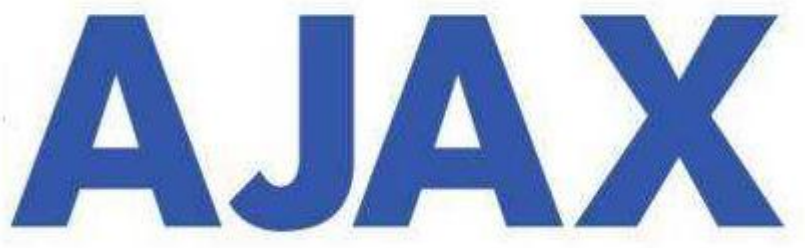
Jquery 的 Ajax 使用

Ajax 实战案例

学习目标

知识点	要求
Ajax 简介	了解
Ajax 的使用	掌握
JSON 详解	掌握
Jquery 的 Ajax 使用	掌握
Ajax 实战案例	掌握

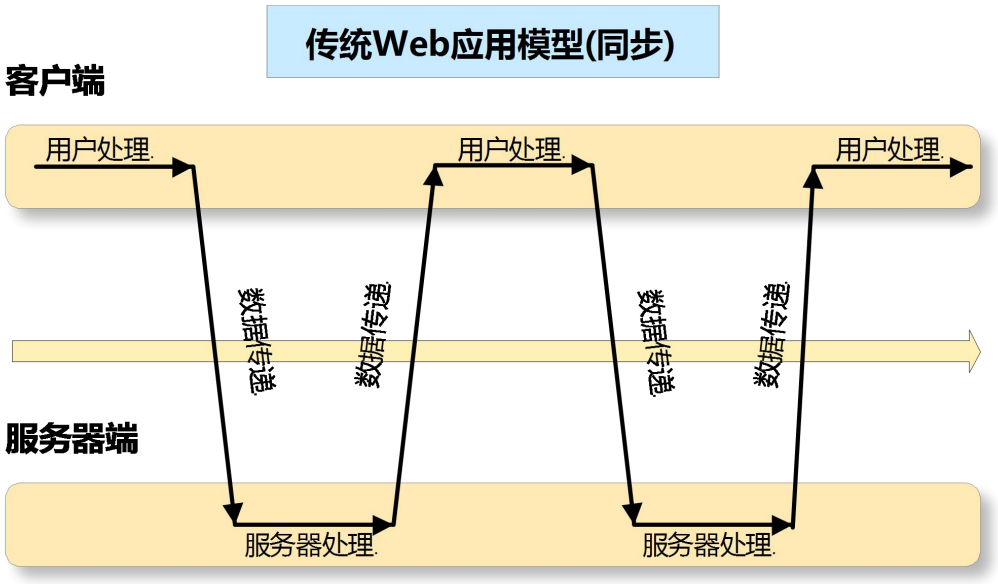
一、 Ajax 简介

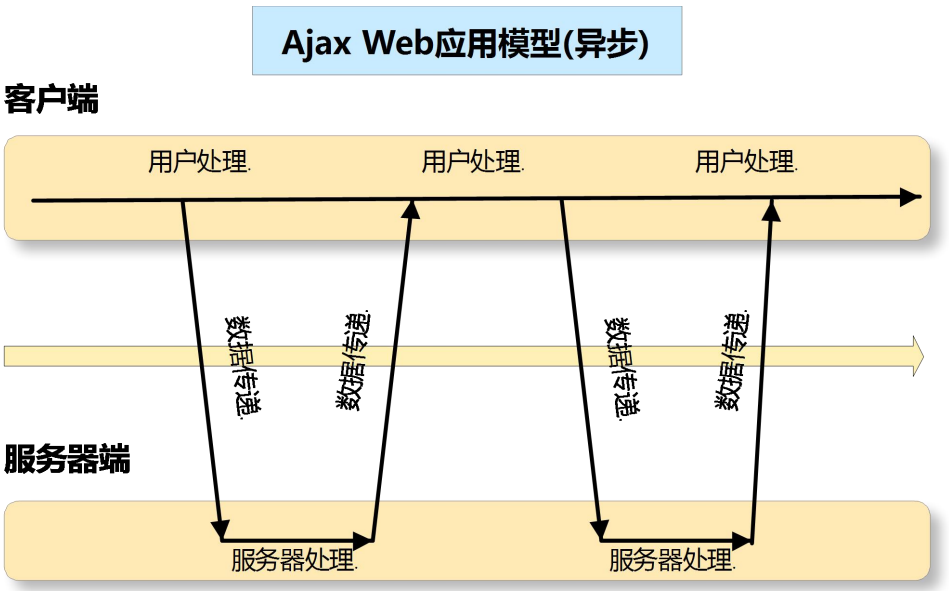


Ajax 即“Asynchronous Javascript And XML”（异步 JavaScript 和 XML），是指一种创建交互式、快速动态应用的网页开发技术，无需重新加载整个网页的情况下，能够更新页面局

部数据的技术。

通过在后台与服务器进行少量数据交换，Ajax 可以使页面实现异步更新。这意味着可以在不重新加载整个页面的情况下，对页面的某部分进行更新。





二、 Ajax 的使用

1 XMLHttpRequest 对象

XMLHttpRequest 是浏览器接口对象，该对象的 API 可被 JavaScript、VBScript 以及其它 web 浏览器内嵌的脚本语言调用，通过 HTTP 协议在浏览器和 web 服务器之间收发 XML 或其它数据。XMLHttpRequest 可以与服务器实现异步交互，而无需让整个页面刷新，因此成为 Ajax 编程的核心对象。

2 Ajax 的使用步骤

1. 创建 XMLHttpRequest 对象

```
var xhr = new XMLHttpRequest();
```

2. 给定请求方式以及请求地址

```
xhr.open("get","http://www.example.com");
```

3. 发送请求

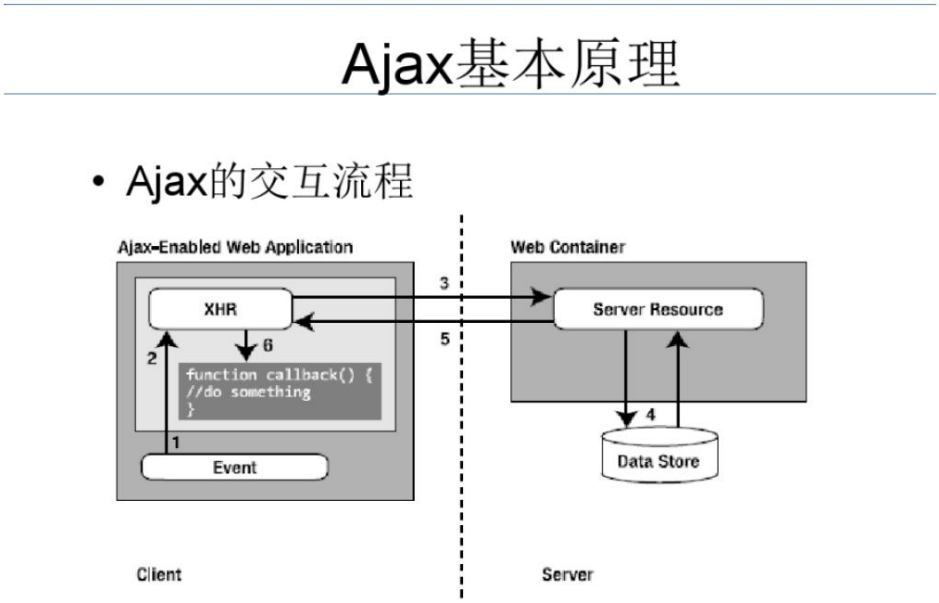
```
xhr.send();
```

4. 获取服务器端给客户端的响应数据

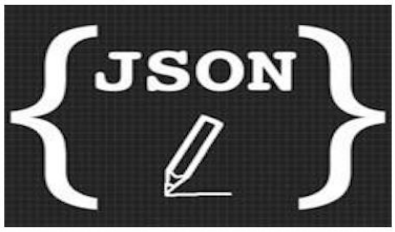
```
xhr.onreadystatechange = function(){
    //0:open()没有被调用
    //1:open()正在被调用
    //2:send()正在被调用
    //3:服务端正在返回结果
    //4:请求结束，并且服务端已经结束发送数据到客户端
    if(xhr.readyState == 4 && xhr.status == 200){
        document.getElementById("span").innerHTML=xhr.responseText;
            alert(xhr.responseText);
    }
```

```
}
```

3 Ajax 的运行原理



三、 JSON 详解



1 JSON 简介

JSON(JavaScript Object Notation) 是一种基于字符串的轻量级的数据交换格式。易于人阅读和编写，同时也易于机器解析和生成。JSON 是 JavaScript 数据类型的子集。

2 为什么要使用 JSON

在 JSON 未出现之前在 Ajax 中对于数据传递方式，会使用 XML 作为主要数据格式来传输数据。直到 JSON 出现后逐渐放弃使用 XML 作为数据传输格式。JSON 比 XML 更小、更

快，更易解析。

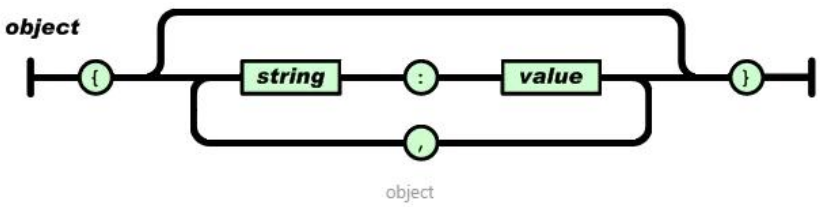
3 JSON 格式的特征

3.1 JSON 的语法规则

JSON 是按照特定的语法规则所生成的字符串结构。

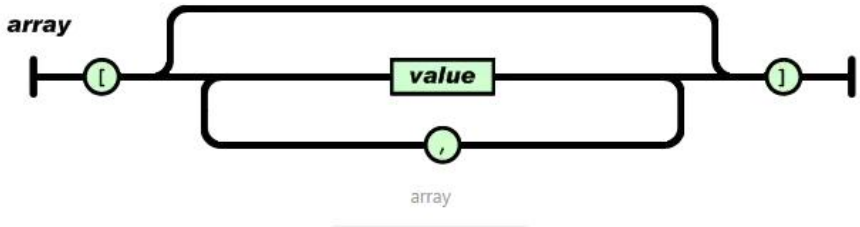
- 大括号表示 JSON 的字符串对象。{ }
- 属性和值用冒号分割。{"属性":"value"}
- 属性和属性之间用逗号分割。{"属性":"value","属性":"value",...}
- 中括号表示数组。[{"属性":"value"...},{"属性":"value"...}]

JSON 字符串对象：



{"userid":1,"username":"admin","sex":"male"}

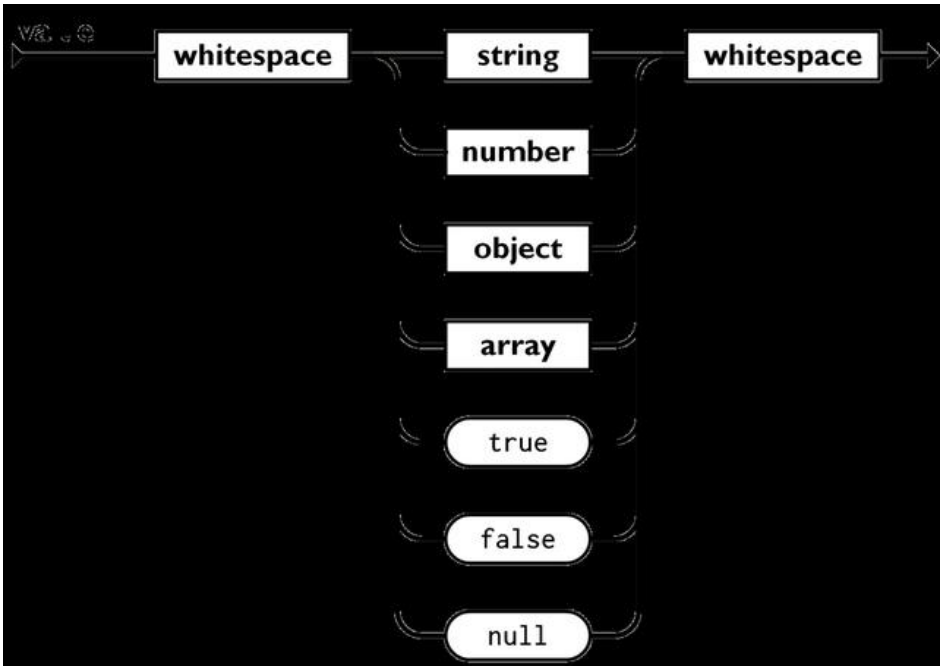
数组：



array

[{"userid":1,"username":"admin"},{"userid":2,"username":"oldlu"}]

3.2 JOSN 的 6 种数据类型



- string：字符串，必须要用双引号引起来。
- number：数值，与 JavaScript 的 number 一致，
- object：JavaScript 的对象形式，{ key:value }表示方式，可嵌套。
- array：数组，JavaScript 的 Array 表示方式[value]，可嵌套。
- true/false：布尔类型，JavaScript 的 boolean 类型。
- null：空值，JavaScript 的 null。

4 Jackson 的使用

在 JDK 中并没有内置操作 JSON 格式数据的 API，因此使用处理 JSON 格式的数据需要借助第三方类库。

几个常用的 JSON 解析类库：

Gson: 谷歌开发的 JSON 库，功能十分全面。

FastJson: 阿里巴巴开发的 JSON 库，性能十分优秀。

Jackson: 社区十分活跃且更新速度很快。被称为“最好的 Json 解析器”

4.1 Jackson 简介

Jackson 是一种解析 JSON 格式数据的 API，也是最流行，速度最快的 JSON API。在 SpringMVC 中默认使用 Jackson API 处理 JSON 格式的数据。

Jackson 下载地址：

https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations
https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core
https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind

4.2 在响应中通过 JSON 格式传递数据

在响应中使用 Jackson 处理 JSON 格式数据的步骤：

- 添加 jackson-annotations.jar、jackson-core.jar、jackson-databind.jar
- 通过 jackson API 将 Java 对象转换 JSON 格式
- 修改响应头，响应类型为 application/json
- 将结果基于字符输出流推回客户端浏览器
- 在页面的中通过 JavaScript 的 JSON.parse()函数将 JSON 格式的数据转换为 JavaScript

对象

4.2.1 通过 JSON 格式在响应中传递单个对象

需求：

定义一个 Users 类，包含 userid、username 属性。

实例化一个 Users 对象，通过 JSON 格式将 Users 对象响应到客户端浏览器。

将 Users 对象中的数据插入到页面中。

4.2.2 通过 JSON 格式在响应中传递多个对象

需求：

定义一个 Users 类，包含 userid、username 属性。

实例化多个 Users 对象，通过 JSON 格式将 Users 对象响应到客户端浏览器。

将 Users 对象中的数据插入到页面中。

4.2.3 在 JSON 中通过 Map 传递数据

在 JSON 格式中可以直接使用 Map 作为传递数据的模型。因为 Map 结构本身就是 key 与 value 的结构与 JSON 格式对象模型完全匹配，所以我们可以直接将一个 Map 对象转换为 JSON 格式的字符串对象。这对于我们来说是一件非常方便的事情，如果我们返回的数据并没有对应的模型来存放数据，那么我们可以通过 Map 来解决。

4.3 在请求中通过 JSON 格式传递数据

我们除了可以在响应中通过 JSON 格式来传递数据以外，在请求中也可以使用 JSON 格式传递数据。如果在请求中使用 JSON 格式传递数据，那么提交方式需要使用 POST 方式，

通过 JavaScript 中的 JSON.stringify()函数将 JavaScript 对象转换为 JSON 格式数据。通过 send 方法将参数传递到 Servlet 中，在 Servlet 中通过字符输入流获取 JSON 格式数据。

在请求中使用 Jackson 处理 JSON 格式数据的步骤：

- 添加 jackson-annotations.jar、jackson-core.jar、jackson-databind.jar
- 在页面的 JavaScript 中通过 JSON.stringify()函数将 JavaScript 对象转换为 JSON 格式的数据
- 将请求方式修改为 POST 方式
- 通过 send()函数将 JSON 格式的数据提交到服务端。
- 在 Servlet 中通过字符输入流读取请求体中 JSON 格式的数据
- 通过 jackson API 将获取到的 JSON 格式的数据转换为 Java 对象

4.4 Jackson 的常用注解

4.4.1 @JsonProperty

此注解用于属性上，作用是把该属性的名称序列化为另外一个名称，如把 username 属性序列化为 name，@JsonProperty("name")。

4.4.2 @JsonIgnore

此注解用于属性或者方法上（一般都是定义在属性上），用来完全忽略被注解的字段和方法对应的属性，返回的 json 数据即不包含该属性。

4.4.3 @JsonFormat

此注解用于属性或者方法上（一般都是定义在属性上），可以方便的把 Date 类型属性

的值直接转化为我们想要的样式。如：@JsonFormat(pattern="yyyy-MM-dd hh:mm:ss")

4.5 Jackson 工具类的使用

```
/**
 * JSON 转换工具类
 */
public class JsonUtils {

    // 定义 jackson 对象
    private static final ObjectMapper MAPPER = new ObjectMapper();

    /**
     * 将对象转换成 json 字符串。
     * <p>Title: pojoToJson</p>
     * <p>Description: </p>
     * @param data
     * @return
     */
    public static String objectToJson(Object data) {
        try {
            String string = MAPPER.writeValueAsString(data);
            return string;
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 将 json 结果集转化为对象

     * @param jsonData json 数据

     * @param beanType 对象中的 object 类型

     * @return
     */
    public static <T> T jsonToPojo(String jsonData, Class<T> beanType)
```

```
{
        try {
            T t = MAPPER.readValue(jsonData, beanType);
            return t;
        } catch (Exception e) {
           e.printStackTrace();
        }
        return null;
    }
    /**
     * 将 json 数据转换成 pojo 对象 list
     * <p>Title: jsonToList</p>
     * <p>Description: </p>
     * @param jsonData
     * @param beanType
     * @return
     */
    public static <T>List<T> jsonToList(String jsonData, Class<T>
beanType) {
       JavaType javaType =
MAPPER.getTypeFactory().constructParametricType(List.class,
beanType);
       try {
          List<T> list = MAPPER.readValue(jsonData, javaType);
          return list;
       } catch (Exception e) {
          e.printStackTrace();
       }
       return null;
    }
}
```

四、 Jquery 的 Ajax 使用

在 JQuery 中提供了对 Ajax 的封装，让我们在使用 Ajax 技术时变得更加容易。在 JQuery 中提供了很多的基于 Ajax 发送异步请求的方法，如：$.ajax()、$.get()、$.post()、$.getJSON()。

1 $.ajax()的使用

1.1语法结构

$.ajax({name:value,name:value.....})

名称	值/描述
async	布尔值，表示请求是否异步处理。默认是 true。
beforeSend(*xhr*)	发送请求前运行的函数。
cache	布尔值，表示浏览器是否缓存被请求页面。默认是 true。
complete(*xhr,status*)	请求完成时运行的函数（在请求成功或失败之后均调用，即在 success 和 error 函数之后）。
contentType	发送数据到服务器时所使用的内容类型。默认是："application/x-www-form-urlencoded"。
context	为所有 AJAX 相关的回调函数规定 "this" 值。
data	规定要发送到服务器的数据。
dataFilter(*data,type*)	用于处理 XMLHttpRequest 原始响应数据的函数。
dataType	预期的服务器响应的数据类型。
error(*xhr,status,error*)	如果请求失败要运行的函数。
global	布尔值，规定是否为请求触发全局 AJAX 事件处理程序。默认是 true。
ifModified	布尔值，规定是否仅在最后一次请求以来响应发生改变时才请求成功。默认是 false。
jsonp	在一个 jsonp 中重写回调函数的字符串。
jsonpCallback	在一个 jsonp 中规定回调函数的名称。
password	规定在 HTTP 访问认证请求中使用的密码。
processData	布尔值，规定通过请求发送的数据是否转换为查询字符串。默认是 true。
scriptCharset	规定请求的字符集。
success(*result,status,xhr*)	当请求成功时运行的函数。
timeout	设置本地的请求超时时间（以毫秒计）。
traditional	布尔值，规定是否使用参数序列化的传统样式。
type	规定请求的类型（GET 或 POST）。
url	规定发送请求的 URL。默认是当前页面。
username	规定在 HTTP 访问认证请求中使用的用户名。
xhr	用于创建 XMLHttpRequest 对象的函数。

1.2 $.ajax()在异步请求中提交数据

在$.ajax()方法中通过 data 属性来存放提交的数据，支持 JSON 格式的数据。

1.2.1 提交普通格式数据

在 data 属性中我们可以通过两种方式来指定需要提交的数据。一种是通过 name=value&name=value 的结构。另一种是通过 JavaScript 对象来指定提交数据。无论使用哪种方式在 Servlet 中都是通过 request.getParameter 方法根据 name 获取 value 的。

1.2.1.1 通过标准格式指定提交数据

data:”name=value&name=value.....”

在 Servlet 中通过 request.getParameter 来获取提交的数据。

1.2.1.2 通过 JavaScript 对象指定提交数据

```
data:{
    userid:100,
    username:”oldlu”
}
```

在 Servlet 中通过 request.getParameter 来获取提交的数据。

1.2.2 提交 JSON 格式数据

在$.ajax()中提交 JSON 格式的数据需要使用 post 方式提交，通过 JSON.stringify()函数将 JavaScript 对象转换成 JSON 格式的字符串。在 Servlet 中通过字符输入获取提交的 JSON 格式的数据。

```
data:JSON.stringify({name:value,name:value.....})
```

在 Servlet 中通过 req.getReader().readLine()来获取提交的数据。

1.3 $.ajax()处理响应中的 JSON 格式数据

$.ajax()方法会根据 dataType 属性中的值自动对响应的数据做类型处理。如果响应的是一个 JSON 格式的数据，那么 dataType 的值为“JSON”,在回调函数中我们得到的直接就是

JSON 字符串转换完的 JavaScript 对象。不需要在使用 JSON.parse()做格式的转换处理。

2 $.get()的使用

$.get()方法是$.ajax()方法基于 get 方式发送异步请求的简化版。

2.1语法结构

$.get(url,function(result))
$.get(url,data,function(result))

2.2通过标准格式指定提交数据

$.get(url,”name=value&name=value”,function(result))

2.3通过 JavaScript 对象指定提交数据

$.get(url,{userid:1,username:”oldlu”,……},function(result))

3 $.post()的使用

$.post()方法是$.ajax()方法基于 post 方式发送异步请求的简化版。

3.1语法结构

$.post(url,function(result))
$.post(url,data,function(result))

3.2通过标准格式指定提交数据

$.post(url,”name=value&name=value”,function(result))

3.3通过 JavaScript 对象指定提交数据

$.post(url,{userid:1,username:”oldlu”,……},function(result))

4 $.getJSON()的使用

$.getJSON()方法是$.ajax()方法基于 get 方式发送异步请求，并将响应结果中 JSON 格式的字符串对象自动转换为 JavaScript 对象。在使用该方法时要求返回的数据必须是 JSON 格式类型。$.getJSON()方法和 resp.setContentType(“application/json”)是一起使用的。

4.1语法结构

$.getJSON(url,function(result))
$.getJSON(url,data,function(result))

4.2通过标准格式指定提交数据

$.getJSON(url,”name=value&name=value”,function(result))

要求返回的数据格式必须是 JSON 格式。

4.3通过 JavaScript 对象指定提交数据

$.getJSON(url,{userid:1,username:”oldlu”,……},function(result))

要求返回的数据格式必须是 JSON 格式。

5 serialize()方法的使用

将 form 表单中的数据自动拼接成 name=value&name=value 结构。

5.1语法结构

var param = $(“form”).serialize();

param 的值为：name=value&name=value

五、 Ajax 实战案例

需求:

创建 User 类，包含 uesrid、username、usersex、userbirth 属性。

在用户管理页面中通过 Ajax 技术完成对用户数据载入、添加用户、更新用户、删除用户操作。

1 搭建环境

1.1创建 User 类

```
public class User {
    private int userid;
    private String username;
    private String usersex;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private Date userbirth;

    public int getUserid() {
        return userid;
    }

    public void setUserid(int userid) {
        this.userid = userid;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getUsersex() {
        return usersex;
    }

    public void setUsersex(String usersex) {
        this.usersex = usersex;
    }

    public Date getUserbirth() {
        return userbirth;
    }
```

```
    public void setUserbirth(Date userbirth) {
        this.userbirth = userbirth;
    }
}
```

1.2在页面中创建表格

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>用户管理</title>
    <script src="js/jquery.js"></script>
    <script>
    </script>
</head>
<body>

<table align="center" width="60%" border="1">
    <tr>
        <td>ID：</td>
        <td><input type="text" name="userid" id="userid"/></td>
        <td>姓名：</td>
        <td><input type="text" name="username" id="username"/></td>
    </tr>
    <tr>
        <td>性别：</td>
        <td><input type="text" name="usersex" id="usersex"/></td>
        <td>生日：</td>
        <td><input type="text" name="userbirth"
id="userbirth"/></td>
    </tr>
    <tr align="center">
        <td colspan="4">
            <input type="button"  value="添加用户" id="add" />
```

```
            <input type="button"  value="更新用户" id="update"/>
        </td>
    </tr>
</table>
<hr/>
<table align="center" width="60%" bgcolor="" border="1"
id="myTable">
    <thead>
    <tr align="center">
        <td>ID</td>
        <td>姓名</td>
        <td>性别</td>
        <td>生日</td>
        <td>操作</td>
    </tr>
    </thead>
    <tbody id="tBody"></tbody>
</table>
</body>
</html>
```

2 加载用户数据

2.1通过 Ajax 完成页面数据初始化

```
<script>
    $(function () {
        //获取页面初始化数据
        getData();
    });
    //获取页面初始化数据
    function getData(){
        $.getJSON("user.do",{flag:"getData"},function (result) {
            init(result);
        });
```

```
    }

    //遍历数据生成数据
    function init(obj){
        var str ="";
        $.each(obj,function(){
            str+= "<tr align='center'><td
id='"+this.userid+"'>"+this.userid
+"</td><td>"+this.username+"</td><td>"+this.usersex+"</td><td>"+t
his.userbirth+"</td><td><a href='#' onclick='preUpdateUser()'>选择
更新</a><a href='#' onclick='deleteUser("+this.userid+")'>删除
</a></td></tr>"
        });
        $("#tBody").prepend(str);
    }
</script>
```

2.2创建 Servlet 处理页面数据初始化请求

```
/**
 * 用户管理 Servlet
 */
@WebServlet("/user.do")
public class UserServlet extends HttpServlet {

    //生成模拟数据初始化
    @Override
    public void init() throws ServletException {
        User user = new User();
        user.setUserid(1);
        user.setUsername("Oldlu");
        user.setUsersex("male");
        user.setUserbirth(new Date());
        User user2 = new User();
        user2.setUserid(2);
        user2.setUsername("Kevin");
        user2.setUsersex("male");
        user2.setUserbirth(new Date());
        List<User> list = new ArrayList<>();
        list.add(user);
```

```
        list.add(user2);
        ServletContext servletContext = this.getServletContext();
        servletContext.setAttribute("list",list);
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        this.doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
            String flag = req.getParameter("flag");
            if("getData".equals(flag)){
                this.getData(req,resp);
            }
    }


    //获取页面初始化数据
    private void getData(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        List<User> list = (List<User>)
this.getServletContext().getAttribute("list");
        String s = JsonUtils.objectToJson(list);
        resp.setContentType("application/json");
        PrintWriter pw = resp.getWriter();
        pw.print(s);
        pw.flush();
        pw.close();
    }
}
```

3 添加用户操作

3.1通过 Ajax 完成添加用户

```
<script>
    $(function () {

        //获取页面初始化数据
```

```
        getData();

        //添加按钮绑定点击事件

        $("#add").click(function(){
            addOrUpdateUser("addUser");
        });
    });

    //获取页面初始化数据

    function getData(){
        $.getJSON("user.do",{flag:"getData"},function (result) {
            init(result);
        });
    }

    //遍历数据生成数据

    function init(obj){
        var str ="";
        $.each(obj,function(){
            str+= "<tr align='center'><td
id='"+this.userid+"'>"+this.userid
+"</td><td>"+this.username+"</td><td>"+this.usersex+"</td><td>"+t

his.userbirth+"</td><td><a href='#' onclick='preUpdateUser()'>选择

更新</a>   <a href='#'

onclick='deleteUser("+this.userid+")'>删除</a></td></tr>"

        });
        $("#tBody").prepend(str);
    }

    //用户添加或者用户更新

    function addOrUpdateUser(flag){

        //从页面中获取数据

        var userid = $("#userid").val();
        var username = $("#username").val();
        var usersex = $("#usersex").val();
        var userbirth = $("#userbirth").val();

        var data = {
            userid:userid,
```

```
            username:username,
            usersex:usersex,
            userbirth:userbirth,
            flag:flag
        }
        location.reload();
        $.get("user.do",data,function(resuslt){
            alert(resuslt);
        });

    }
</script>
```

3.2在 Servlet 中处理添加用户请求

Servlet 代码

```
/**
 * 用户管理 Servlet
 */
@WebServlet("/user.do")
public class UserServlet extends HttpServlet {

    //生成模拟数据初始化
    @Override
    public void init() throws ServletException {
        User user = new User();
        user.setUserid(1);
        user.setUsername("Oldlu");
        user.setUsersex("male");
        user.setUserbirth(new Date());
        User user2 = new User();
        user2.setUserid(2);
        user2.setUsername("Kevin");
        user2.setUsersex("male");
        user2.setUserbirth(new Date());
        List<User> list = new ArrayList<>();
        list.add(user);
        list.add(user2);
        ServletContext servletContext = this.getServletContext();
        servletContext.setAttribute("list",list);
```

```
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        this.doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
            String flag = req.getParameter("flag");
            if("getData".equals(flag)){
                this.getData(req,resp);
            }else if("addUser".equals(flag)){
                this.addUser(req,resp);
            }
    }


    //获取页面初始化数据
    private void getData(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        List<User> list = (List<User>)
this.getServletContext().getAttribute("list");
        String s = JsonUtils.objectToJson(list);
        resp.setContentType("application/json");
        PrintWriter pw = resp.getWriter();
        pw.print(s);
        pw.flush();
        pw.close();
    }

    //处理添加用户请求
    private void addUser(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
            User user = this.createUser(req);
            ServletContext servletContext = this.getServletContext();
            List<User> list = (List<User>)
servletContext.getAttribute("list");
            list.add(user);
            resp.setContentType("text/plain;charset=utf-8");
            PrintWriter pw = resp.getWriter();
```

```
            pw.print("添加成功");
            pw.flush();
            pw.close();
    }

    //获取请求数据
    private User createUser(HttpServletRequest req){
        String userid = req.getParameter("userid");
        String username = req.getParameter("username");
        String usersex = req.getParameter("usersex");
        String userbirth  = req.getParameter("userbirth");

        User user  = new User();
        user.setUserid(Integer.parseInt(userid));
        user.setUsername(username);
        user.setUsersex(usersex);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        try {
            Date parse = sdf.parse(userbirth);
            user.setUserbirth(parse);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return user;
    }
}
```

解决时区问题

```
@JsonFormat(pattern = "yyyy-MM-dd",timezone = "GMT+8")
```

4 更新用户操作

4.1通过 Ajax 完成预更新用户选择

```
<script>
    $(function () {
        //获取页面初始化数据
        getData();
        //添加按钮绑定点击事件
        $("#add").click(function(){
```

```
            addOrUpdateUser("addUser");
        });
    });

    //获取页面初始化数据

    function getData(){
        $.getJSON("user.do",{flag:"getData"},function (result) {
            init(result);
        });
    }

    //遍历数据生成数据

    function init(obj){
        var str ="";
        $.each(obj,function(){
            str+= "<tr align='center'><td
id='"+this.userid+"'>"+this.userid
+"</td><td>"+this.username+"</td><td>"+this.usersex+"</td><td>"+t
his.userbirth+"</td><td><a href='#'

onclick='preUpdateUser("+this.userid+")'>选择更新

</a>   <a href='#'

onclick='deleteUser("+this.userid+")'>删除</a></td></tr>"

        });
        $("#tBody").prepend(str);
    }

    //用户添加或者用户更新

    function addOrUpdateUser(flag){

        //从页面中获取数据

        var userid = $("#userid").val();
        var username = $("#username").val();
        var usersex = $("#usersex").val();
        var userbirth = $("#userbirth").val();


        var data = {
            userid:userid,
            username:username,
            usersex:usersex,
            userbirth:userbirth,
            flag:flag
```

```
        }
        $.get("user.do",data,function(result){
            alert(result);
        });
        location.reload();
    }

    //更新之前的数据选择
    function preUpdateUser(userid){
        var arr = new Array();

$("#"+userid).closest("tr").children().each(function(index,ele){
            if(index <=3){
              arr[index]= ele.innerText
            }
        });
        $("#userid").val(arr[0]);
        $("#username").val(arr[1]);
        $("#usersex").val(arr[2]);
        $("#userbirth").val(arr[3]);
        $("#userid").attr("readonly",true);
    }
</script>
```

4.2通过 Ajax 完成更新用户

```
<script>
    $(function () {

        //获取页面初始化数据
        getData();
        //添加按钮绑定点击事件
        $("#add").click(function(){
            addOrUpdateUser("addUser");
        });
        //更新按钮绑定点击事件
        $("#update").click(function(){
            addOrUpdateUser("updateUser");
        });
    });
```

```
    //获取页面初始化数据
    function getData(){
        $.getJSON("user.do",{flag:"getData"},function (result) {
            init(result);
        });
    }

    //遍历数据生成数据
    function init(obj){
        var str ="";
        $.each(obj,function(){
            str+= "<tr align='center'><td
id='"+this.userid+"'>"+this.userid
+"</td><td>"+this.username+"</td><td>"+this.usersex+"</td><td>"+t
his.userbirth+"</td><td><a href='#'
onclick='preUpdateUser("+this.userid+")'>选择更新
</a>   <a href='#'
onclick='deleteUser("+this.userid+")'>删除</a></td></tr>"
        });
        $("#tBody").prepend(str);
    }

    //用户添加或者用户更新
    function addOrUpdateUser(flag){
        //从页面中获取数据
        var userid = $("#userid").val();
        var username = $("#username").val();
        var usersex = $("#usersex").val();
        var userbirth = $("#userbirth").val();

        var data = {
            userid:userid,
            username:username,
            usersex:usersex,
            userbirth:userbirth,
            flag:flag
        }
        $.get("user.do",data,function(result){
            alert(result);
```

```
        });
        location.reload();
    }

    //更新之前的数据选择

    function preUpdateUser(userid){
        var arr = new Array();

$("#"+userid).closest("tr").children().each(function(index,ele){
            if(index <=3){
              arr[index]= ele.innerText
            }
        });
        $("#userid").val(arr[0]);
        $("#username").val(arr[1]);
        $("#usersex").val(arr[2]);
        $("#userbirth").val(arr[3]);
        $("#userid").attr("readonly",true);
    }
</script>
```

4.3在 Servlet 中处理更新用户请求

```
/**
 * 用户管理 Servlet
 */
@WebServlet("/user.do")
public class UserServlet extends HttpServlet {

    //生成模拟数据初始化

    @Override
    public void init() throws ServletException {
        User user = new User();
        user.setUserid(1);
        user.setUsername("Oldlu");
        user.setUsersex("male");
        user.setUserbirth(new Date());
        User user2 = new User();
        user2.setUserid(2);
        user2.setUsername("Kevin");
        user2.setUsersex("male");
        user2.setUserbirth(new Date());
```

```
        List<User> list = new ArrayList<>();
        list.add(user);
        list.add(user2);
        ServletContext servletContext = this.getServletContext();
        servletContext.setAttribute("list",list);
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        this.doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
            String flag = req.getParameter("flag");
            if("getData".equals(flag)){
                this.getData(req,resp);
            }else if("addUser".equals(flag)){
                this.addUser(req,resp);
            }else if("updateUser".equals(flag)){
                this.updateUser(req,resp);
            }
    }


    //获取页面初始化数据
    private void getData(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        List<User> list = (List<User>)
this.getServletContext().getAttribute("list");
        String s = JsonUtils.objectToJson(list);
        resp.setContentType("application/json");
        PrintWriter pw = resp.getWriter();
        pw.print(s);
        pw.flush();
        pw.close();
    }

    //处理添加用户请求
    private void addUser(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
```

```
            User user = this.createUser(req);
            ServletContext servletContext = this.getServletContext();
            List<User> list = (List<User>)
servletContext.getAttribute("list");
            list.add(user);
            resp.setContentType("text/plain;charset=utf-8");
            PrintWriter pw = resp.getWriter();

            pw.print("添加成功");

            pw.flush();
            pw.close();
    }

    //获取请求数据

    private User createUser(HttpServletRequest req){
        String userid = req.getParameter("userid");
        String username = req.getParameter("username");
        String usersex = req.getParameter("usersex");
        String userbirth  = req.getParameter("userbirth");

        User user  = new User();
        user.setUserid(Integer.parseInt(userid));
        user.setUsername(username);
        user.setUsersex(usersex);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        try {
            Date parse = sdf.parse(userbirth);
            user.setUserbirth(parse);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return user;
    }


    //处理更新用户请求

    private void updateUser(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException{
        User user = this.createUser(req);
        ServletContext servletContext = this.getServletContext();
        List<User> list = (List<User>)
servletContext.getAttribute("list");
        User u = null;
```

```
        for(User temp : list){
            if(temp.getUserid() == user.getUserid()){
                u = temp;
                break;
            }
        }
        if(u != null){
            list.remove(u);
        }
        list.add(user);
        resp.setContentType("text/plain;charset=utf-8");
        PrintWriter pw = resp.getWriter();

        pw.print("更新成功");

        pw.flush();
        pw.close();
    }
}
```

5 删除用户操作

5.1通过 Ajax 完成删除用户

```
<script>
    $(function () {

        //获取页面初始化数据

        getData();

        //添加按钮绑定点击事件

        $("#add").click(function(){
            addOrUpdateUser("addUser");
        });

        //更新按钮绑定点击事件

        $("#update").click(function(){
            addOrUpdateUser("updateUser");
        });
    });

    //获取页面初始化数据

    function getData(){
        $.getJSON("user.do",{flag:"getData"},function (result) {
```

```
            init(result);
        });
    }

    //遍历数据生成数据
    function init(obj){
        var str ="";
        $.each(obj,function(){
            str+= "<tr align='center'><td
id='"+this.userid+"'>"+this.userid
+"</td><td>"+this.username+"</td><td>"+this.usersex+"</td><td>"+t
his.userbirth+"</td><td><a href='#'
onclick='preUpdateUser("+this.userid+")'>选择更新
</a>   <a href='#'
onclick='deleteUser("+this.userid+")'>删除</a></td></tr>"
        });
        $("#tBody").prepend(str);
    }

    //用户添加或者用户更新
    function addOrUpdateUser(flag){
        //从页面中获取数据
        var userid = $("#userid").val();
        var username = $("#username").val();
        var usersex = $("#usersex").val();
        var userbirth = $("#userbirth").val();

        var data = {
            userid:userid,
            username:username,
            usersex:usersex,
            userbirth:userbirth,
            flag:flag
        }
        $.get("user.do",data,function(result){
            alert(result);
        });
        location.reload();
    }
```

```
    //更新之前的数据选择
    function preUpdateUser(userid){
        var arr = new Array();

$("#"+userid).closest("tr").children().each(function(index,ele){
            if(index <=3){
              arr[index]= ele.innerText
            }
        });
        $("#userid").val(arr[0]);
        $("#username").val(arr[1]);
        $("#usersex").val(arr[2]);
        $("#userbirth").val(arr[3]);
        $("#userid").attr("readonly",true);
    }

    //删除用户
    function deleteUser(userid){
        $("#"+userid).closest("tr").remove();
        $.get("user.do",{userid:userid},function(result){
            alert(result);
        })
    }
</script>
```

5.2在 Servlet 处理删除用户请求

```
/**
 * 用户管理 Servlet
 */
@WebServlet("/user.do")
public class UserServlet extends HttpServlet {

    //生成模拟数据初始化
    @Override
    public void init() throws ServletException {
        User user = new User();
        user.setUserid(1);
        user.setUsername("Oldlu");
        user.setUsersex("male");
```

```
        user.setUserbirth(new Date());
        User user2 = new User();
        user2.setUserid(2);
        user2.setUsername("Kevin");
        user2.setUsersex("male");
        user2.setUserbirth(new Date());
        List<User> list = new ArrayList<>();
        list.add(user);
        list.add(user2);
        ServletContext servletContext = this.getServletContext();
        servletContext.setAttribute("list",list);
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        this.doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
            String flag = req.getParameter("flag");
            if("getData".equals(flag)){
                this.getData(req,resp);
            }else if("addUser".equals(flag)){
                this.addUser(req,resp);
            }else if("updateUser".equals(flag)){
                this.updateUser(req,resp);
            }else{
                this.deleteUser(req,resp);
            }
    }


    //获取页面初始化数据
    private void getData(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        List<User> list = (List<User>)
this.getServletContext().getAttribute("list");
        String s = JsonUtils.objectToJson(list);
        resp.setContentType("application/json");
        PrintWriter pw = resp.getWriter();
```

```
        pw.print(s);
        pw.flush();
        pw.close();
    }

    //处理添加用户请求
    private void addUser(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
            User user = this.createUser(req);
            ServletContext servletContext = this.getServletContext();
            List<User> list = (List<User>)
servletContext.getAttribute("list");
            list.add(user);
            resp.setContentType("text/plain;charset=utf-8");
            PrintWriter pw = resp.getWriter();

            pw.print("添加成功");

            pw.flush();
            pw.close();
    }

    //获取请求数据
    private User createUser(HttpServletRequest req){
        String userid = req.getParameter("userid");
        String username = req.getParameter("username");
        String usersex = req.getParameter("usersex");
        String userbirth  = req.getParameter("userbirth");

        User user  = new User();
        user.setUserid(Integer.parseInt(userid));
        user.setUsername(username);
        user.setUsersex(usersex);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        try {
            Date parse = sdf.parse(userbirth);
            user.setUserbirth(parse);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return user;
    }
```

```
    //处理更新用户请求
    private void updateUser(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException{
        User user = this.createUser(req);
        ServletContext servletContext = this.getServletContext();
        List<User> list = (List<User>)
servletContext.getAttribute("list");
        User u = null;
        for(User temp : list){
            if(temp.getUserid() == user.getUserid()){
                u = temp;
                break;
            }
        }
        if(u != null){
            list.remove(u);
        }
        list.add(user);
        resp.setContentType("text/plain;charset=utf-8");
        PrintWriter pw = resp.getWriter();

        pw.print("更新成功");

        pw.flush();
        pw.close();
    }


    //处理删除用户请求
    private void deleteUser(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException{
        ServletContext servletContext = this.getServletContext();
        List<User> list = (List<User>)
servletContext.getAttribute("list");
        String userid = req.getParameter("userid");
        User user = null;
        for(User temp:list){
            if((temp.getUserid()+"").equals(userid)){
                user = temp;
                break;
            }
        }
        if(user != null){
```

```
            list.remove(user);
        }
        resp.setContentType("text/plain;charset=utf-8");
        PrintWriter pw = resp.getWriter();

        pw.print("删除成功");

        pw.flush();
        pw.close();
    }
}
```