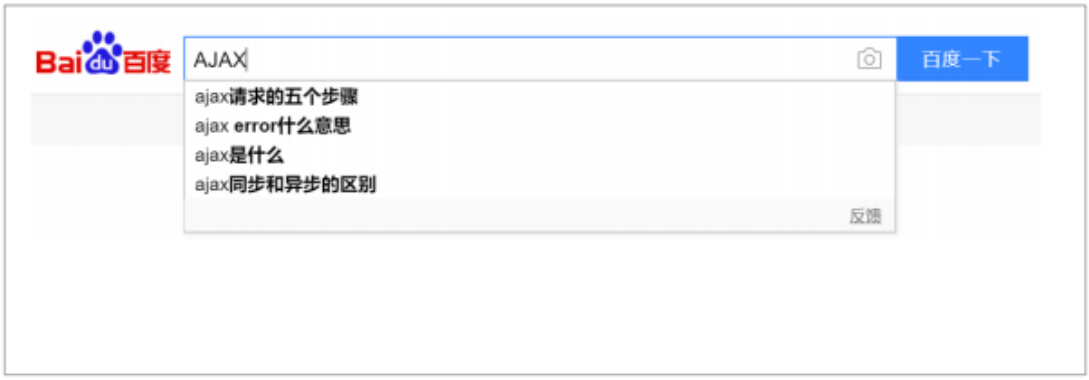
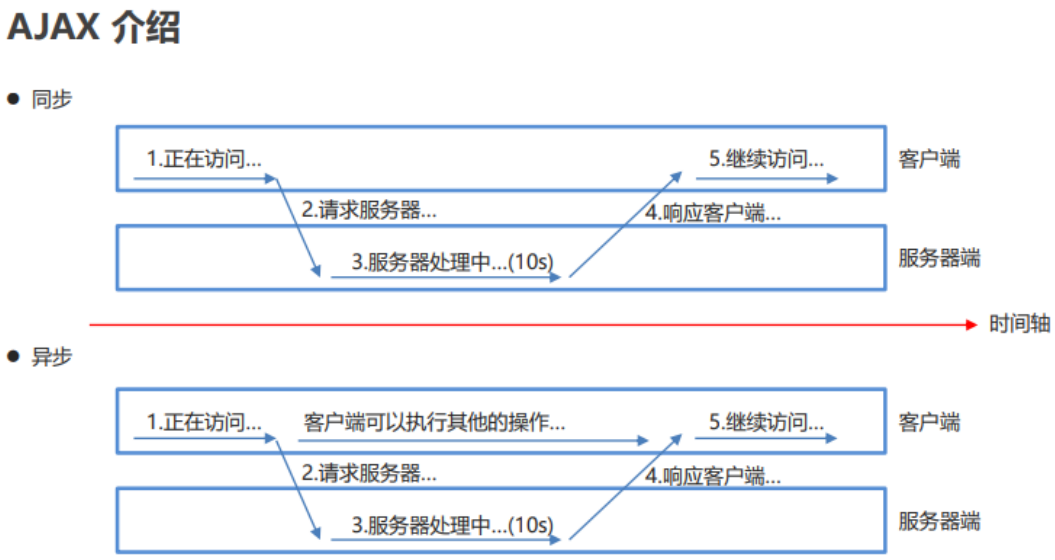
1、Ajax快速入门

1.1、AJAX介绍

- AJAX(Asynchronous JavaScript And XML)：异步的 JavaScript 和 XML。
- 本身不是一种新技术，而是多个技术综合。用于快速创建动态网页的技术。
- 一般的网页如果需要更新内容，必需重新加载个页面。
- 而 AJAX 通过浏览器与服务器进行少量数据交换，就可以使网页实现异步更新。也就是在不重新加载整个页 面的情况下，对网页的部分内容进行**局部更新**。





1.2、原生JS实现AJAX

- 代码实现

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
```

```java
@WebServlet("/userServlet")
public class UserServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //设置请求和响应的乱码
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html;charset=UTF-8");

        //1.获取请求参数
        String username = req.getParameter("username");

        //模拟服务器处理请求需要5秒钟
        /*try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }*/

        //2.判断姓名是否已注册
        if("zhangsan".equals(username)) {
            resp.getWriter().write("<font color='red'>用户名已注册</font>");
        }else {
            resp.getWriter().write("<font color='green'>用户名可用</font>");
        }
    }

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

html代码

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>用户注册</title>
</head>
<body>
    <form autocomplete="off">
        姓名：<input type="text" id="username">
        <span id="uSpan"></span>
        <br>
        密码：<input type="password" id="password">
        <br>
        <input type="submit" value="注册">
    </form>
</body>
<script>
    //1.为姓名绑定失去焦点事件
    document.getElementById("username").onblur = function() {
        //2.创建XMLHttpRequest核心对象
        let xmlHttp = new XMLHttpRequest();
```

```
        //3.打开链接
        let username = document.getElementById("username").value;
        xmlHttp.open("GET","userServlet?username="+username,true);
        //xmlHttp.open("GET","userServlet?username="+username,false);

        //4.发送请求
        xmlHttp.send();

        //5.处理响应
        xmlHttp.onreadystatechange = function() {
            //判断请求和响应是否成功
            if(xmlHttp.readyState == 4 && xmlHttp.status == 200) {
                //将响应的数据显示到span标签
                document.getElementById("uSpan").innerHTML =
xmlHttp.responseText;
            }
        }
    }
</script>
</html>
```

1.3、原生JS实现AJAX详解

- **核心对象：XMLHttpRequest**

  用于在后台与服务器交换数据。可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

- **打开链接：open(method,url,async)**

  method：请求的类型 GET 或 POST。

  url：请求资源的路径。

  async：true(异步) 或 false(同步)。

- **发送请求：send(String params)**

  params：请求的参数(POST 专用)。

- **处理响应：onreadystatechange**

  readyState：0-请求未初始化，1-服务器连接已建立，2-请求已接收，3-请求处理中，4-请求已完成，且响应已就绪。

  status：200-响应已全部 OK。

- **获得响应数据形式**

  responseText：获得字符串形式的响应数据。

  responseXML：获得 XML 形式的响应数据。

1.4、JQuery的GET方式实现AJAX

- **核心语法：**$.get(url,[data],[callback],[type]);
  - url：请求的资源路径。
  - data：发送给服务器端的请求参数，格式可以是key=value，也可以是 js 对象。
  - callback：当请求成功后的回调函数，可以在函数中编写我们的逻辑代码。
  - type：预期的返回数据的类型，取值可以是 xml, html, js, json, text等。
- **代码实现**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>用户注册</title>
</head>
<body>
    <form autocomplete="off">
        姓名：<input type="text" id="username">
        <span id="uSpan"></span>
        <br>
        密码：<input type="password" id="password">
        <br>
        <input type="submit" value="注册">
    </form>
</body>
<script src="js/jquery-3.3.1.min.js"></script>
<script>
    //1.为用户名绑定失去焦点事件
    $("#username").blur(function () {
        let username = $("#username").val();
        //2.jQuery的GET方式实现AJAX
        $.get(
            //请求的资源路径
            "userServlet",
            //请求参数
            "username=" + username,
            //回调函数
            function (data) {
                //将响应的数据显示到span标签
                $("#uSpan").html(data);
            },
            //响应数据形式
            "text"
        );
    });
</script>
</html>
```

1.5、JQuery的POST方式实现AJAX

- **核心语法：**$.post(url,[data],[callback],[type]);
  - url：请求的资源路径。
  - data：发送给服务器端的请求参数，格式可以是key=value，也可以是 js 对象。
  - callback：当请求成功后的回调函数，可以在函数中编写我们的逻辑代码。
  - type：预期的返回数据的类型，取值可以是 xml, html, js, json, text等。
- **代码实现**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>用户注册</title>
</head>
<body>
```

```
    <form autocomplete="off">
        姓名：<input type="text" id="username">
        <span id="uSpan"></span>
        <br>
        密码：<input type="password" id="password">
        <br>
        <input type="submit" value="注册">
    </form>
</body>
<script src="js/jquery-3.3.1.min.js"></script>
<script>
    //1.为用户名绑定失去焦点事件
    $("#username").blur(function () {
        let username = $("#username").val();
        //2.jQuery的POST方式实现AJAX
        $.post(
            //请求的资源路径
            "userServlet",
            //请求参数
            "username=" + username,
            //回调函数
            function (data) {
                //将响应的数据显示到span标签
                $("#uSpan").html(data);
            },
            //响应数据形式
            "text"
        );
    });
</script>
</html>
```

1.6、JQuery的通用方式实现AJAX

- **核心语法：**$.ajax({name:value,name:value,...});
  - url：请求的资源路径。
  - async：是否异步请求，true-是，false-否 (默认是 true)。
  - data：发送到服务器的数据，可以是键值对形式，也可以是 js 对象形式。
  - type：请求方式，POST 或 GET (默认是 GET)。
  - dataType：预期的返回数据的类型，取值可以是 xml, html, js, json, text等。
  - success：请求成功时调用的回调函数。
  - error：请求失败时调用的回调函数。
- **代码实现**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>用户注册</title>
</head>
<body>
    <form autocomplete="off">
        姓名：<input type="text" id="username">
        <span id="uSpan"></span>
        <br>
```

```
        密码：<input type="password" id="password">
        <br>
        <input type="submit" value="注册">
    </form>
</body>
<script src="js/jquery-3.3.1.min.js"></script>
<script>
    //1.为用户名绑定失去焦点事件
    $("#username").blur(function () {
        let username = $("#username").val();
        //2.jQuery的通用方式实现AJAX
        $.ajax({
            //请求资源路径
            url:"userServletxxx",
            //是否异步
            async:true,
            //请求参数
            data:"username="+username,
            //请求方式
            type:"POST",
            //数据形式
            dataType:"text",
            //请求成功后调用的回调函数
            success:function (data) {
                //将响应的数据显示到span标签
                $("#uSpan").html(data);
            },
            //请求失败后调用的回调函数
            error:function () {
                alert("操作失败...");
            }
        });
    });
</script>
</html>
```

1.7、小结

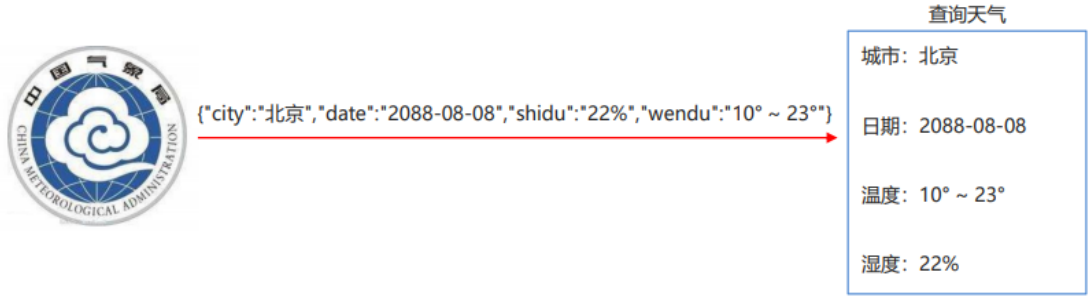
- **AJAX(Asynchronous JavaScript And XML)：**异步的 JavaScript 和 XML。
- **通过浏览器与服务器进行少量数据交换，就可以使网页实现异步更新。也就是在不重新加载整个页面的情况下，对网页的部 分内容进行局部更新。**
- **同步和异步**
  - 同步：服务器端在处理过程中，无法进行其他操作。
  - 异步：服务器端在处理过程中，可以进行其他操作。
- **GET 方式实现：**$.get();
- **POST 方式实现：**$.post();
  - url：请求的资源路径。
  - data：发送给服务器端的请求参数，格式可以是key=value，也可以是 js 对象。
  - callback：当请求成功后的回调函数，可以在函数中编写我们的逻辑代码。
  - type：预期的返回数据的类型，取值可以是 xml, html, js, json, text等。
- **通用方式实现：**$.ajax();
  - url：请求的资源路径。
  - async：是否异步请求，true-是，false-否 (默认是 true)。

- data：发送到服务器的数据，可以是键值对形式，也可以是 js 对象形式。
- type：请求方式，POST 或 GET (默认是 GET)。
- dataType：预期的返回数据的类型，取值可以是 xml, html, js, json, text等。
- success：请求成功时调用的回调函数。
- error：请求失败时调用的回调函数。

2、JSON的处理

2.1、JSON回顾

- JSON(JavaScript Object Notation)：是一种轻量级的数据交换格式。
- 它是基于 ECMAScript 规范的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。
- 简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。易于人阅读和编写，同时也易于计算机解析和生成，并有效的 提升网络传输效率。



- **创建格式**

类型	语法	说明
对象类型	{name:value,name:value,...}	name是字符串类型，value可以是任意类型
数组/集合类型	[{name:value,...},{name:value,...}]	
混合类型	{name: [{name:value,...},{name:value,...}] }	

- **常用方法**

成员方法	说明
stringify(对象)	将指定对象转换为json格式字符串
parse(字符串)	将指定json格式字符串解析成对象

2.2、JSON转换工具的介绍

- 我们除了可以在 JavaScript 中来使用 JSON 以外，在 JAVA 中同样也可以使用 JSON。
- JSON 的转换工具是通过 JAVA 封装好的一些 JAR 工具包。
- 可以将 JAVA 对象或集合转换成 JSON 格式的字符串，也可以将 JSON 格式的字符串转成 JAVA 对象。
- Jackson：开源免费的 JSON 转换工具，SpringMVC 转换默认使用 Jackson。
    - 1. 导入 jar 包。

2. 创建核心对象。
3. 调用方法完成转换。

- **常用类**

类名	说明
ObjectMapper	是Jackson工具包的核心类, 它提供一些方法来实现 JSON 字符串和对象之间的转换
TypeReference	对集合泛型的反序列化，使用TypeReference可以明确的指定反序列化的对象类型

- **ObjectMapper常用方法**

方法名	说明
String writeValueAsString(Object obj)	将Java对象转换成JSON字符串
<T> T readValue(String json, Class<T> valueType)	将JSON字符串转换成Java对象
<T> T readValue(String json, TypeReference valueTypeRef)	将JSON字符串转换成Java对象

2.3、JSON转换练习

1. **对象转 JSON, JSON 转对象。**

```
/*
    JSON转换工具的使用
 */
public class ObjectMapperTest {
    private ObjectMapper mapper = new ObjectMapper();
    /*
        1.User对象转json, json转User对象
          json字符串 = {"name":"张三","age":23}
          user对象 = User{name='张三', age=23}
     */
    @Test
    public void test01() throws Exception{
        //User对象转json
        User user = new User("张三",23);
        String json = mapper.writeValueAsString(user);
        System.out.println("json字符串: " + json);

        //json转User对象
        User user2 = mapper.readValue(json, User.class);
        System.out.println("java对象: " + user2);
    }
}
```

2. **Map转 JSON, JSON 转 Map。**

```
/*
  2.map<String,String>转json, json转map<String,String>
  json字符串 = {"姓名":"张三","性别":"男"}
  map对象 = {姓名=张三, 性别=男}
*/
@Test
public void test02() throws Exception{
    //map<String,String>转json
    HashMap<String,String> map = new HashMap<>();
```

```
    map.put("姓名","张三");
    map.put("性别","男");
    String json = mapper.writeValueAsString(map);
    System.out.println("json字符串：" + json);

    //json转map<String,String>
    HashMap<String,String> map2 = mapper.readValue(json, HashMap.class);
    System.out.println("java对象：" + map2);
}
```

3. **Map转 JSON, JSON 转 Map。**

```
/*
  3.map<String,User>转json, json转map<String,User>
    json字符串 = {"黑马一班":{"name":"张三","age":23},"黑马二班":{"name":"李四","age":24}}
    map对象 = {黑马一班=User{name='张三', age=23}, 黑马二班=User{name='李四', age=24}}
 */
@Test
public void test03() throws Exception{
    //map<String,User>转json
    HashMap<String,User> map = new HashMap<>();
    map.put("黑马一班",new User("张三",23));
    map.put("黑马二班",new User("李四",24));
    String json = mapper.writeValueAsString(map);
    System.out.println("json字符串：" + json);

    //json转map<String,User>
    HashMap<String,User> map2 = mapper.readValue(json,new TypeReference<HashMap<String,User>>(){});
    System.out.println("java对象：" + map2);
}
```

4. **List转 JSON, JSON 转 List。**

```
/*
  4.List<String>转json, json转 List<String>
    json字符串 = ["张三","李四"]
    list对象 = [张三, 李四]
*/
@Test
public void test04() throws Exception{
    //List<String>转json
    ArrayList<String> list = new ArrayList<>();
    list.add("张三");
    list.add("李四");
    String json = mapper.writeValueAsString(list);
    System.out.println("json字符串：" + json);

    //json转 List<String>
    ArrayList<String> list2 = mapper.readValue(json,ArrayList.class);
    System.out.println("java对象：" + list2);
}
```

5. **List转 JSON, JSON 转 List。**

```
/*
  5.List<User>转json, json转List<User>
    json字符串 = [{"name":"张三","age":23},{"name":"李四","age":24}]
    list对象 = [User{name='张三', age=23}, User{name='李四', age=24}]
     */
@Test
public void test05() throws Exception{
    //List<User>转json
    ArrayList<User> list = new ArrayList<>();
    list.add(new User("张三",23));
    list.add(new User("李四",24));
    String json = mapper.writeValueAsString(list);
    System.out.println("json字符串：" + json);

    //json转List<User>
    ArrayList<User> list2 = mapper.readValue(json,new
TypeReference<ArrayList<User>>(){});
    System.out.println("java对象：" + list2);
}
```

2.4、小结

- **Jackson**：开源免费的 JSON 转换工具，SpringMVC 转换默认使用 Jackson。
- **可以将 JAVA 对象或集合转换成 JSON 格式的字符串，也可以将 JSON 格式的字符串转成 JAVA 对象。**
- **常用类**

方法名	说明
String writeValueAsString(Object obj)	将Java对象转换成JSON字符串
<T> T readValue(String json, Class<T> valueType)	将JSON字符串转换成Java对象
<T> T readValue(String json, TypeReference valueTypeRef)	将JSON字符串转换成Java对象

3、综合案例 搜索联想

- **案例效果和环境介绍**

- **案例分析**
  - **页面**

        1. 为用户名输入框绑定鼠标点击事件。
        2. 获取输入的用户名数据。
        3. 判断用户名是否为空。
        4. 如果为空，则将联想提示框隐藏。
        5. 如果不为空，则发送 AJAX 请求，并将响应的数据显示到联想框。
    - **控制层**
        1. 获取请求参数。
        2. 调用业务层的模糊查询方法。
        3. 将返回的数据转成 JSON，并响应给客户端。
- **代码实现**
    - **html页面**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>用户搜索</title>
    <style type="text/css">
        .content {
            width: 643px;
            margin: 100px auto;
            text-align: center;
        }

        input[type='text'] {
            width: 530px;
            height: 40px;
            font-size: 14px;
        }

        input[type='button'] {
            width: 100px;
            height: 46px;
            background: #38f;
            border: 0;
            color: #fff;
            font-size: 15px
        }

        .show {
            position: absolute;
            width: 535px;
            height: 100px;
            border: 1px solid #999;
            border-top: 0;
            display: none;
        }
    </style>
</head>
<body>
<form autocomplete="off">
    <div class="content">
        <img src="img/logo.jpg">
        <br/><br/>
        <input type="text" id="username">
```

```
        <input type="button" value="搜索一下">
        <!--用于显示联想的数据-->
        <div id="show" class="show"></div>
    </div>
</form>
</body>
<script src="js/jquery-3.3.1.min.js"></script>
<script>
    //1.为用户名输入框绑定鼠标点击事件
    $("#username").mousedown(function () {
        //2.获取输入的用户名
        let username = $("#username").val();

        //3.判断用户名是否为空
        if(username == null || username == "") {
            //4.如果为空，将联想框隐藏
            $("#show").hide();
            return;
        }

        //5.如果不为空，发送AJAX请求。并将数据显示到联想框
        $.ajax({
            //请求的资源路径
            url:"userServlet",
            //请求参数
            data:{"username":username},
            //请求方式
            type:"POST",
            //响应数据形式
            dataType:"json",
            //请求成功后的回调函数
            success:function (data) {
                //将返回的数据显示到show的div
                let names = "";
                for(let i = 0; i < data.length; i++) {
                    names += "<div>"+data[i].name+"</div>";
                }
                $("#show").html(names);
                $("#show").show();
            }
        });
    });


</script>
</html>
```

- **控制层-Java代码**

```
@WebServlet("/userServlet")
public class UserServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        //设置请求和响应的编码
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html;charset=UTF-8");
```

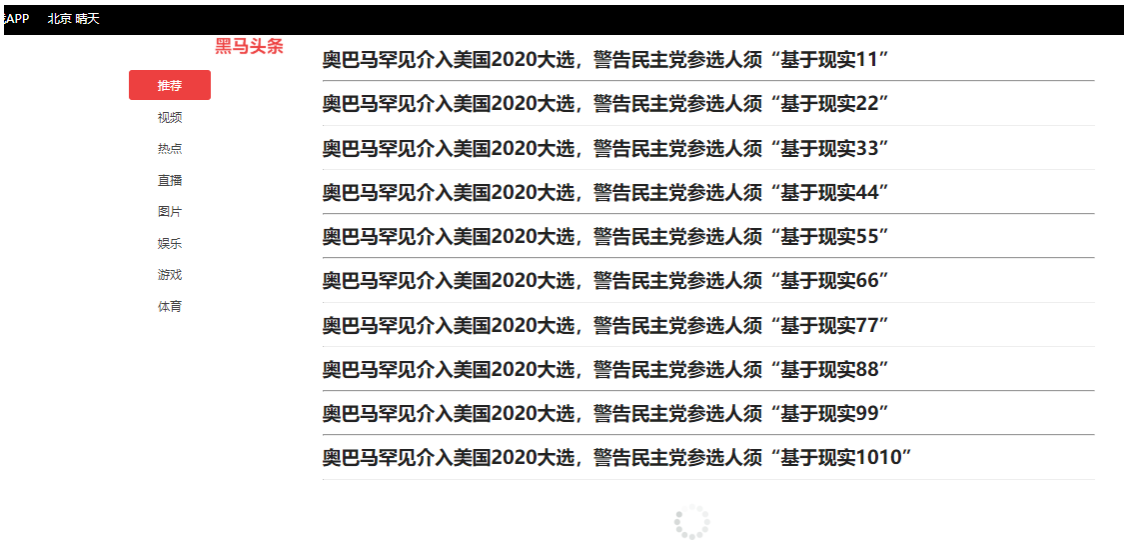
```
        //1.获取请求参数
        String username = req.getParameter("username");

        //2.调用业务层的模糊查询方法得到数据
        UserService service = new UserServiceImpl();
        List<User> users = service.selectLike(username);

        //3.将数据转成JSON，响应到客户端
        ObjectMapper mapper = new ObjectMapper();
        String json = mapper.writeValueAsString(users);
        resp.getWriter().write(json);
    }

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

4、综合案例 分页

瀑布流无限加载数据分页

4.1、案例效果和环境准备

- **案例效果**



- **环境准备**
  - 1.导入“案例二的sql语句.sql”文件（已在当天的**SQL语句**中提供）
  - 2.导入“分页案例原始环境”中的ajax03项目（已在当天的**资料**中提供）

4.2、案例的分析

- **如何确定当前显示的数据已经浏览完毕?**
  - 公式：(滚动条距底部的距离 + 滚动条上下滚动的距离 + 当前窗口的高度) >= 当前文档的高度
  - 当前文档高度：存储10条数据，100px。
  - 滚动条距底部的距离：1px。

- 当前窗口的高度：80px。
- 滚动条上下滚动的距离：>=19px。

- **前置知识**

功能	说明
$(function(){})	页面加载事件
$(window)	获取当前窗口对象
scroll()	鼠标滚动事件
$(window).height()	当前窗口的高度
$(window).scrollTop()	滚动条上下滚动的距离
$(document).height()	当前文档的高度

4.3、案例的实现

- **实现思路**
  - **页面**
    1. 定义发送请求标记。
    2. 定义当前页码和每页显示的条数。
    3. 定义滚动条距底部的距离。
    4. 设置页面加载事件。
    5. 为当前窗口绑定滚动条滚动事件。
    6. 获取必要信息(当前窗口的高度,滚动条上下滚动的距离,当前文档的高度)。
    7. 计算当前展示数据是否浏览完毕。
    8. 判断请求标记是否为 true。
    9. 将请求标记置为 false，当前异步操作完成前，不能重新发起请求。
    10. 根据当前页和每页显示的条数来请求查询分页数据。
    11. 当前页码+1。
  - **服务器**
    1. 获取请求参数(当前页,每页显示的条数)。
    2. 根据当前页码和每页显示的条数，调用业务层的方法，得到分页 Page 对象。
    3. 将得到的数据转为 json。
    4. 将数据响应给客户端。

- **代码实现**
  - **html页面**

```
<script>
    //1.定义发送请求标记
    let send = true;

    //2.定义当前页码和每页显示的条数
    let start = 1;
    let pageSize = 10;

    //3.定义滚动条距底部的距离
    let bottom = 1;

    //4.设置页面加载事件
    $(function () {
```

```
        //5.为当前窗口绑定滚动条滚动事件
        $(window).scroll(function () {
            //6.获取必要信息，用于计算当前展示数据是否浏览完毕
            //当前窗口的高度
            let windowHeight = $(window).height();

            //滚动条从上到下滚动距离
            let scrollTop = $(window).scrollTop();

            //当前文档的高度
            let docHeight = $(document).height();

            //7.计算当前展示数据是否浏览完毕
            //当 滚动条距底部的距离 + 当前滚动条滚动的距离 + 当前窗口的高度 >= 当前文档的高度
            if((bottom + scrollTop + windowHeight) >= docHeight) {
                //8.判断请求标记是否为true
                if(send == true) {
                    //9.将请求标记置为false，当前异步操作完成前，不能重新发起请求。
                    send = false;
                    //10.根据当前页和每页显示的条数来 请求查询分页数据
                    queryByPage(start,pageSize);
                    //11.当前页码+1
                    start++;
                }
            }
        });
    });

    //定义查询分页数据的函数
    function queryByPage(start,pageSize){
        //加载动图显示
        $(".loading").show();
        //发起AJAX请求
        $.ajax({
            //请求的资源路径
            url:"newsServlet",
            //请求的参数
            data:{"start":start,"pageSize":pageSize},
            //请求的方式
            type:"POST",
            //响应数据形式
            dataType:"json",
            //请求成功后的回调函数
            success:function (data) {
                if(data.length == 0) {
                    $(".loading").hide();
                    $("#no").html("我也是有底线的...");
                    return;
                }
                //加载动图隐藏
                $(".loading").hide();
                //将数据显示
                let titles = "";
                for(let i = 0; i < data.length; i++) {
                    titles += "<li>\n" +
                        "                <div class=\"title-box\">\n" +
```

```
                        "                        <a href=\"#\"
class=\"link\">\n" +
                                                        data[i].title +
                        "                            <hr>\n" +
                        "                        </a>\n" +
                        "                    </div>\n" +
                        "                </li>";
                }

                //显示到页面
                $(".news_list").append(titles);
                //将请求标记设置为true
                send = true;
            }
        });
    }
</script>
```

- **java代码**

```
import com.fasterxml.jackson.databind.ObjectMapper;
import com.github.pagehelper.Page;
import com.itheima.service.NewsService;
import com.itheima.service.impl.NewsServiceImpl;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
@WebServlet("/newsServlet")
public class NewsServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        //设置请求和响应的编码
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html;charset=UTF-8");

        //1.获取请求参数
        String start = req.getParameter("start");
        String pageSize = req.getParameter("pageSize");

        //2.根据当前页码和每页显示的条数来调用业务层的查询方法，得到分页Page对象
        NewsService service = new NewsServiceImpl();
        Page page = service.pageQuery(Integer.parseInt(start),
Integer.parseInt(pageSize));

        //3.将得到的数据转为JSON
        String json = new ObjectMapper().writeValueAsString(page);

        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

```
        //4.将数据响应给客户端
        resp.getWriter().write(json);
    }

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

4.4、点击按钮分页

- **案例效果和环境准备**
  - **按钮效果**

  - **环境准备**
    1. 使用已完善的ajax3项目
    2. 复制index.html更名为index2.html，并去掉原js中的方法
- **案例的分析**
  - **页面**

    1.引入分页插件的样式文件和 js 文件。

    2. 定义当前页码和每页显示的条数。
    3. 调用查询数据的函数。
    4. 定义请求查询分页数据的函数，发起 AJAX 异步请求。
    5. 为分页按钮区域设置页数参数(总页数和当前页)。
    6. 为分页按钮绑定单击事件,完成上一页下一页查询功能。
  - **服务器**
    1. 获取请求参数。
    2. 根据当前页码和每页显示的条数，调用业务层的方法，得到分页 Page 对象
    3. 封装 PageInfo 对象。
    4. 将得到的数据转为 json。
    5. 将数据响应给客户端。
- **案例的实现**
  - **html页面**

```
<script>
    //1.定义当前页码和每页显示的条数
    let start = 1;
    let pageSize = 10;

    //2.调用查询数据的方法
    queryByPage(start,pageSize);

    //3.定义请求查询分页数据的函数，发起AJAX异步请求，将数据显示到页面
    function queryByPage(start,pageSize) {
        $.ajax({
            //请求的资源路径
            url:"newsServlet2",
```

```
            //请求的参数
            data:{"start":start,"pageSize":pageSize},
            //请求的方式
            type:"POST",
            //响应数据形式
            dataType:"json",
            //请求成功后的回调函数
            success:function (pageInfo) {
                //将数据显示到页面
                let titles = "";
                for(let i = 0; i < pageInfo.list.length; i++) {
                    titles += "<li>\n" +
                        "                <div class=\"title-box\">\n" +
                        "                    <a href=\"#\"
class=\"link\">\n" +
                                                pageInfo.list[i].title
+
                        "                     <hr>\n" +
                        "                </a>\n" +
                        "            </div>\n" +
                        "        </li>";
                }
                $(".news_list").html(titles);

                //4.为分页按钮区域设置页数参数（总页数和当前页）
                $("#light-pagination").pagination({
                    pages:pageInfo.pages,
                    currentPage:pageInfo.pageNum
                });

                //5.为分页按钮绑定单击事件,完成上一页下一页查询功能
                $("#light-pagination .page-link").click(function () {
                    //获取点击按钮的文本内容
                    let page = $(this).html();
                    //如果点击的是Prev，调用查询方法，查询当前页的上一页数据
                    if(page == "Prev") {
                        queryByPage(pageInfo.pageNum - 1,pageSize);
                    }else if (page == "Next") {
                        //如果点击的是Next，调用查询方法，查询当前页的下一页数据
                        queryByPage(pageInfo.pageNum + 1,pageSize);
                    } else {
                        //调用查询方法，查询当前页的数据
                        queryByPage(page,pageSize);
                    }
                });
            }
        });
    }
</script>
```

- **Java代码**

```
import com.fasterxml.jackson.databind.ObjectMapper;
import com.github.pagehelper.Page;
import com.github.pagehelper.PageInfo;
import com.itheima.bean.News;
import com.itheima.service.NewsService;
```

```
import com.itheima.service.impl.NewsServiceImpl;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;

@WebServlet("/newsServlet2")
public class NewsServlet2 extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        //设置请求和响应的编码
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html;charset=UTF-8");

        //1.获取请求参数
        String start = req.getParameter("start");
        String pageSize = req.getParameter("pageSize");

        //2.根据当前页码和每页显示的条数来调用业务层的查询方法，得到分页Page对象
        NewsService service = new NewsServiceImpl();
        Page page = service.pageQuery(Integer.parseInt(start),
Integer.parseInt(pageSize));

        //3.封装PageInfo对象
        PageInfo<List<News>> info = new PageInfo<>(page);

        //4.将得到的数据转为JSON
        String json = new ObjectMapper().writeValueAsString(info);

        //5.将数据响应给客户端
        resp.getWriter().write(json);

    }

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