目录

1.认识Ajax

- Ajax：Asynchronous JavaScript and XML (**异步**JavaScript和XML)
  - 节省用户操作、时间、提高用户体验，减少数据请求 传输获取数据
  - 异步的javascript 和 xml(数据传输格式)

- `xml` 和 `json`
  - `xml` 数据传输格式　(大型门户网站 新浪、网易、凤凰网)
    - 优点：(标签名自定义)
      1. 种类丰富
      2. 传输量非常大
    - 缺点：
      1. 解析麻烦
      2. 不太适合轻量级数据
  - `json` 数据传输格式(字符串)　95%移动端应用
    - 优点：
      1. 轻量级数据
      2. 解析比较轻松
    - 缺点：
      1. 数据种类少
      2. 传输数据量比较小
- JS提供的两个方法可以解析 `json`
  - `JSON.stringify()` ： 数据结构 => json字符串
  - `JSON.parse()` ： json格式字符串 => 数据结构

2.同步和异步

- 任何一个程序都是由很多个小程序组成的。
- **同步和异步**用来**描述一个程序运行状态**
  - 同步： 阻塞，当前程序运行，必须**等前一个程序运行完毕**以后，才能运行
  - 异步： 非阻塞，当前程序运行，和前面程序的运行没有任何关系。

- 同步和异步的例子：
  - 同步任务：指在主线程上排队执行的任务，只有前一个任务执行完毕，才能继续执行下一个任务

```
for(var i = 0; i < 20000; i++){
  console.log(i);
}
console.log(2);
// 这里要等for循环执行完毕，才能执行后面的代码
```

  - 异步任务：委托给宿主环境执行(JS宿主是浏览器)
    - 定时器 `setTimeout` `setInterval`
    - `ajax` 的异步请求
    - es6的 `promise`

3.ajax语法

- `ajax` 默认异步请求

```
//1.创建ajax对象
var xhr = new XMLHttpRequest();

// 2.调用open
xhr.open("get", "1.txt", true);
  // 第一个参数：请求方式 get post(不止这两种)
  // 第二个参数：完整的url/相对路径
  // 第三个参数：是否异步 true异步（默认），false同步

// 3.调用send    发送请求
xhr.send();

// 4.等待数据响应
xhr.onreadystatechange = function(){
  if(xhr.readyState == 4){
    if(xhr.status == 200){
        alert(xhr.responseText)
    }else{
        alert('请求出错' + xhr.status);
    }
  }
}
```

4.ajax对象

- 获取ajax对象有兼容问题
  - `XMLHttpRequest();` IE8以下不兼容
  - `ActiveXObject('Microsoft.XMLHTTP')` IE8以下声明ajax的方法

```
var xhr = null;
if(window.XMLHttpRequest){
  xhr = new XMLHttpRequest();
}else{
  xhr = new ActiveXObject('Microsoft.XMLHTTP');
}
```

5.try_catch

```
try{
  // 尝试执行的代码;
}catch(error){
  // error 错误对象，try括号中代码执行的异常信息;
  // 补救代码;
}
```

- 先去执行 `try` 中的代码
- 如果 `try` 中的代码执行正常， `catch` 中的代码就不执行了
- 如果 `try` 中的代码执行异常，直接执行 `catch` 中的代码进行补救
  - `error` 存储异常信息
- `try_catch_finally` 中的 `throw`
  - throw 手动抛出异常 (制造出一个异常) `throw new Error("这是演习，不要紧张");`

```
try{
  alert("异常信息之前");
  throw new Error("这是演习，不要紧张");
  alert("异常信息之后");
}catch(error){
  alert("补救代码,Error"+error); //此时的error为 这是演习
}finally{
  // 无论出不出错都会执行的代码
}
```

6.onreadystatechange 事件

- `xhr.readystatechange` 事件在 `xhr.readyState` 发生变化的时候触发
- `xhr.readyState` 的值
  - `0` ： 调用 `open` 方法之前
  - `1` ： 调用 `open` 方法之后-调用 `send` 方法发送请求
  - `2` ： `send` 方法完成,已经接受到所有的响应内容
  - `3` ： 正在解析下载到的数据
  - `4` ： 解析完成
- `xhr.status` 状态码
  - `200` ： 交易成功
  - `400` ： 错误请求 如：语法错误
  - `404` ： 没有发现文件、查询或 `url`

    - 状态码为200时，说明下载是成功的，否则就弹出`Error + xhr.status`；

7.php中的 `$_GET` 和 `$_POST`

- `$_GET` ： 存放通过get提交 提交的所有数据

```
$age = $_GET['age'];
```

- `$_POST`： 存储通过post发送过来的数组

```
$age = $_POST["age"];
```

8.form表单的get请求/post请求

- form表单的两个属性
    - `action` ： 点击`submit`(提交按钮)之后跳转到的`url`
    - `method` ： 表单的提交数据方式
        - `get`(默认)
            - 提交方式：直接将数据拼接在`url`的后面进行提交，通过`?`进行拼接,查询字符串
              `http://localhost/14/6.get.php?username=aaa&age=18&password=wangwang`
                - 好处：简单
                - 缺点：1.不安全； 2.地址栏最大`2kb` ； 3.没法实现上传
        - `post`
            - 要在form表单添加一个属性 `enctype="application/x-www-form-urlencoded"`
                - `enctype` 提交数据的格式
            - 提交方式： `post`提交通过浏览器内部进行提交。
            - 好处：1.安全； 2.大小理论上没有上限； 3.可以上传
            - 缺点：比get复杂

9.ajax get和post

- `get`

```
xhr.open("get", "6.get.php?username=xxx&age=19&password=wang", true);
```

- `post`

```
xhr.open("post", "6.post.php", true);
// 设置post的请求格式
xhr.setRequestHeader("content-type","application/x-www-form-urlencoded");
xhr.send("username=xxx&age=19&password=wang");
```

    - 必须在`send`方法之前，设置`post`的请求格式
    - `post` 需要通过`send`方法进行数据提交

```
xhr.open("get", "6.get.php?username=xxx&age=19", true);  // get要加?
xhr.send("username=xxx&age=19");   // post不要加?
```

10.ajax函数的封装

```
function $ajax({method = "get",url,data}){
    // 1. 创建ajax对象
    var xhr = null;
    try{
      xhr = new XMLHttpRequest();
    }catch(error){
        xhr = new ActiveXObject('Microsoft.XMLHTTP');
    }
    // 判断数据是否存在
    if(data){ data = queryString(data); }
    // 2.调用open方法  判断是否为get方法
    if(method.toLowerCase() == "get" && data){ url += "?" + data; }
    xhr.open(method, url, true);
    // 3.调用send方法 发送请求
    if(method.toLowerCase() == "get"){
        xhr.send();
    }else if(method.toLowerCase() == "post"){
        //必须在send方法之前设置请求的格式
        xhr.setRequestHeader("content-type","application/x-www-form-urlencoded");
        xhr.send(data);
    }else{
        alert("Error:请检查提交数据方式");
        console.log("Error:请检查提交数据方式");
        return;
    }
    // 4.等待数据响应
    xhr.onreadystatechange = function(){
        if(xhr.readyState == 4){
            if(xhr.status == 200){
                alert(xhr.responseText);
            }else{
                alert("Error:" + xhr.status);
            }
        }
    }
}
// 将对象拼接成字符串
function queryString(obj){
    var str = "";
    for(var attr in obj){
        str += attr + "=" + obj[attr] + "&";
    }
    return str.substr(0,str.length - 1);
}
```

11.ajax_函数封装_回调函数

- 利用回调函数 + 大括号解构

```
$ajax({method, url, data, success, msg})
```

- `method`：请求方式
- `url`：请求地址
- `data`：是否有数据
- `success`：数据下载成功执行的函数
- `error`：数据下载失败执行的函数

12.JSON对象

- `JSON.stringify()`：数据结构 => json格式字符串
- `JSON.parse()`：json格式字符串 => 数据结构

- `JSON`支持的数据类型
  - **4种基本类型**：`String` `Numbers` `Booleans` `Null`
  - **2种结构类型**
    - 对象 `{name:"zhangsan",age:18}`
      - 键为`String`，值为`Json`的任意数据类型之一。对象的键值对是无序的。
    - 数组 `[1,2,3]` 顺序排列的零个或多个`Json`数据类型
- 数组 转 `json` 字符串

```
var arr = [100, true, "hello"];
JSON.stringify(arr)  /// '[100, true, "hello"]'
```

- 对象 转 `json` 字符串

```
var obj = {
  username: "钢铁侠",
  age: 18
}
JSON.stringify(obj) // '{"username":"钢铁侠","age":18}'
```

13.案例-php中的json

- `php`中的`json`
  - 编码: `json_encode()` 将数据结构 转成 json字符串
  - 解码: `json_decode()` 将json字符串 转成 数据结构
    - 后端通过`echo`输出转成`json`字符串的数据，前端通过`ajax`获取数据
- 前后端交互流程
  1. 通过`ajax`下载数据
  2. 分析数据，转成对应数据结构
  3. 处理数据

14.案例-新闻列表

1. 通过节点插入
2. 先清空数据 `oUl.innerHTML = "";`

3. 下载到数据之后，通过 `JSON.pares(arr)` 转成数组
4. 通过`for`循环,添加和插入节点，向`ul`添加数据
5. 新闻列表_拼接字符串
6. 不用清空数据
7. 创建一个空ECMA6字符串 ``
8. 通过循环，每次给字符串添加数据,组成一个完整的字符串。

```
str += `<li>
            <a href="#+${i}"> ${arr[i].title} </a>
            <span>[ ${arr[i].date} ]
        </li>`
```

9. 通过 `oUl.innerHTML` 自动解析标签的功能，添加数据。