ajax

JSON

1.1概念
数据格式，实质是一个字符串
1.2注意
1）属性名必须使用双引号
2）当属性值是字符串形式时，必须使用双引号
3）JSON文件中不允许出现注释
4）JSON数据不支持undefined和函数

方法
1）JSON.parse(JSON的字符串) -- 将JSON数据格式转换为对象
2）JSON.stringify(object) -- 将对象转换为JSON字符串

```
例子
{
"title":"苹果手机",
"list":[
{
"id":1,
"name":"苹果13",
"price":"5799元"
},
{
"id":2,
"name":"苹果13 pro",
"price":"7799元"
},
]
}
```

概念

异步的js和xml
2.2作用
客户端给服务端发送http请求
2.3特点
1）异步请求
同步请求--第一个请求没有完成，第二个请求不可以发送
异步请求--第一个请求没有完成，不影响后面的请求
2）局部刷新
form发送请求--刷新整个页面
ajax发送请求--局部刷新--登录，注册页面，快递单号查询，搜索等
2.4优点
局部刷新，给客户端节省宽高，提高用户的体验度
2.5缺点
依赖于js，浏览器不支持js，ajax就无法运行，ajax应用程序中安全性较低
2.6原理
1）前端创建ajax对象
2）让ajax对象去请求服务器
3）服务器接收到前端的请求，处理请求
4）服务器把处理的数据返回给ajax对象
5）客户端监听ajax对象拿到的数据

Ajax请求步骤

1）创建xhr对象 -- 秘书
let xhr = new XMLHttpRequest();
2）准备发送请求 -- 让秘书找小王
xhr.open('请求方式','url',true);
3）发送请求 -- 秘书去找小王
xhr.send();
4）监听函数 -- 监督秘书的工作
5）状态执行完毕获取数据 -- 秘书或功找到小王
xhr.onreadystatechange=function(){
//后台成功的响应注满的数据
if(xhr.readyState===4 && status === 200){
//成功获取后台的数据
xhr.response //返回后台的数据
}
}

```
例子
<script>
//ajax使用步骤
//1.创建xhr对象 -- 秘书
let xhr = new XMLHttpRequest();
console.log(xhr);

//2.准备发送请求 -- 让秘书找一下小王
xhr.open('get', '/06ajax.php', true);

//3.发送请求 -- 秘书去找小王
xhr.send();

//4.监听函数 -- 监督秘书的工作
//只要状态发生改变，就会触发事件，事件会触发多次
xhr.onreadystatechange = function () {
//5.状态执行完毕才可以获取数据 -- 秘书正确的找到了小王
if (xhr.readyState === 4 && xhr.status === 200)
//成功获取后台的数据
console.log(xhr.response);
}
}
</script>
```

Ajax对象和服务器交互时产生的状态值
0 -- 还没有发送请求
1 -- 开始发送请求
2 -- 服务器接收到请求
3 -- 服务器处理请求
4 -- 服务器响应请求

http协议状态码
200 -- 成功
404 -- 资源不存在
5XX -- 后台的错误

jquery封装的Ajax
2.1语法
$.ajax({
type:'get|post',//请求方式
url:'路径',//请求的地址
data:{//传参 -- 没有参数可以省略
后台参数:前端参数值,
后台参数:前端参数值,
后台参数:前端参数值,
},
dataType:'返回的数据类型',//可以省略
async:'是否异步',//默认true--异步--可以省略
success:function(res){//形参--用来接收后台成功时返回的数据

},
error:function(err){//形参--用来接收后台失败时返回的数据--可以省略

},
beforeSend:function(){//请求成功之前执行的函数--可以省略

},
complete:function(){//请求完成时执行的函数，不管成功还是失败--可以省略

})

```
/*
登录:
用户名和密码正确：返回登录成功
用户名和密码不正确：返回用户名或密码错误
*/

//1.绑定点击事件
$('#login').click(function () {

//3.获取前端参数值
let user = $('#user').val();
let pwd = $('#pwd').val();


//2.发送ajax请求
$.ajax({
type: 'post',
url: '/php/02login.php',
data: {
//后台参数:前端参数值
// user:user,
// pwd:pwd
user,
pwd
},
success(res) {//res--形参--接收后台成功时返回的数据
// console.log(res);
//4.判断后台的返回值
if (res === '1') {//成功
$('.msg').html('登录成功').css('color', 'green');
} else {//失败
$('.msg').html('用户名或密码错误').css('color', 'red');
}
}
})
})
```

跨域请求
只有浏览器有跨域服务器没有

什么是跨域
从当前的域下去访问其他域下的资源，浏览器的同源策略出于安全考虑，默认是不允许访问其他网站的脚本

2.什么是同源策略
同源策略(Same origin policy)是一种约定，它是浏览器最核心也最基本的安全功能。如果缺少了同源策略，则浏览器的正常功能可能都会受到影响。可以说Web是构建在同源策略基础之上的，浏览器只是针对同源策略的一种实现。

所谓的同源，指的**协议、域名、端口相同。**浏览器处于安全方面的考虑，只允许本域名下的接口交互，不同源的客户端脚本，在没有明确授权的情况下，不能读写对方的资源。

如何解决跨域
1)后台代理（后端服务器）
2) jsonp(只支持get请求)
3) cors资源共享(最自)
4) proxy反向代理(框架)
5)iframe(早期写法)

后台代理
5.后台代理解决跨域5.1原因
跨域:浏览器的同源策略--安全考虑
思路:让后端代码去访问服务器资源，服务器把数据返回给后台代码后，前端再访问服务器资源。

jsonp
6.1jsonp实现原理
动态创建script标签，利用script标签里面的src属性可以访问浏览器上的任意资源，不产生跨域。
让src去访问要请求的资源，服务器接受请求，并返回数据，**前端创建一个函数接受后台返回的数据(不是json格式)。**

注意:jsonp只支持get请求

```
<script>
//jsonp解决跨域
//5.创建script 2.cc请求数据 3.前端用函数接收返回的数据

//1.绑定点击事件
document.getElementById('search').onclick = function () {
//2. 获取当前值
let text = document.getElementById('text').value;

//3.网络请求
//3.1创建script
let script = document.createElement('script');
//3.2cc请求数据
//wd -- 后台定义接收前端参数的变量
//cb=callback -- cb:后台定义用来接收前端函数的变量，callback：前端接收数据的变量名
script.src = 'http://suggestion.baidu.com/su?cb=callback&wd=' + text;
//3.3添加到body中
document.body.appendChild(script);

}

//3.4前端定义接收数据的函数
function callback(res){
console.log(res);
}

</script>
```

6.2.2**jquery封装的跨域请求：（掌握）**

```
//请法
$.ajax({
type:"get",//jsonp跨域只有get请求
url:"接口",
dataType:'jsonp',//必须写 jsonp
data:{ wd:'前端内容'},
jsonp:'',//后台定义的接受的函数名字 比如，百度 cb 一般是callback可以省略
jsonpCallback:"",//可以省略 前端接受的函数名 不写默认生成一个jquery+数字
success:function(){//成功 数据在这里
}
})
```

jquery封装

cookie

1）特点
①存储大小有限制 -- 4kb
②需要网络 -- 每次都要发送http请求
③没有兼容性问题

2）原理
客户端访问服务器的时候，服务器会自动生成一个cookie，响应给客户端是会传送自动生成的cookie，客户端会把cookie保存起来，等以后再次访问服务器时，会携带cookie去访问服务器。

3）百度登录例子的过程
①第一次登录百度，访问服务器，服务器生成一个cookie
②服务器响应给客户端，把cookie传递给客户端了，用户需要输入用户名和密码
③以后再登录百度，客户端携带着cookie访问服务器
④服务器响应客户端，用户直接登录百度页面，不是示登录按钮

4）语法
①后台存储，前端获取 -- document.cookie
②前端存储，前端获取
前端存储：document.cookie= "name=value;expires=结束时间;path=/";
有效时间--只支持GMT(格林尼治时间)和UTC(国际统一时间)
path--/ --表示根目录

前端获取：document.cookie

5）注意
当name相同时，后面的数据会完全覆盖前面的数据

```
//后台存储，前端获取 -- document.cookie
//前端存储，前端获取
//前端存储 -- document.cookie = "name=value"; -- 多个键值对分号隔开
//前端获取 -- document.cookie

//存储 -- 一次只能存储一条数据
// document.cookie = "user=admin";
// document.cookie = "pwd=666";
// document.cookie = "sex=男";

//有效时间 -- 只支持GMT(格林尼治时间),UTC(国际统一时间)
//路径 -- /--整个域名下所有文件--跟目录 /code--code文件
//安全设置 -- secure=true/false -- 设置请求和响应的过程数据加密，客户端本地保存的cookie没有加密，需要自己设置加密

//处理时间
//1.获取当前时间 2.转换为格林尼治时间
//new Date()--默认返回字符串
let now = new Date(new Date().getTime() + 1 * 1000 * 60);
let endTime = now.toUTCString();
document.cookie = "user=王一博;expires=" + endTime + ";path=/";
document.cookie = "pwd=666;expires=" + endTime + ";path=/";

console.log(document.cookie);//user=王一博; pwd=666
```

```
//封装设置cookie的方法
//document.cookie = "name=value;expires=结束时间;path=/";
/**
* name -- 键名
* value -- 键值
* endTime -- 结束时间 --分钟 ，5--表示5分钟后失效
* */
function setCookie(name, value, endTime) {
//2.拼value加密
value = encodeURiComponent(value);

//3.处理结束时间
let now = new Date(new Date().getTime() + endTime * 60 * 1000);
let endTime1 = now.toUTCString();

//1.设置cookie
document.cookie = `${name}=${value};expires=${endTime1};path=/`;

}

//设置cookie
setCookie('user', '甚样千玺', 10);//10分钟后失效
// setCookie('username', '迪丽热巴', 5);
// setCookie('user', '王一博=hello', 20);//cookie存储数据是name不可以重复，如果重复后面的数据会覆盖前面的数据
setCookie('pwd', '666', 1);

//获取cookie
// let cookie = document.cookie;
//
console.log(cookie);//username=%E8%8F%AA%E4%B8%BD%E7%83%AD%E5%B7%8A; user=%E7%8E%8B%E4%B8%80%E5%BD%9Ahello;

//封装获取cookie的方法
//限造，传入name，返回对应的value
function getCookie(key) {
//1.获取所有cookie
let cookie = document.cookie + ';';
// console.log(cookie);
//username=%E8%8F%AA%E4%B8%BD%E7%83%AD%E5%B7%8A; user=%E7%8E%8B%E4%B8%80%E5%BD%9Ahello;

//2.拿到需要的value
//限造，字符串截取 -- str.slice(开始位置的下标,结束位置的下标)--含头不含尾
//开始位置的下标 -- key后面第一个等号的下一位
//结束位置的下标 -- key后面第一个分号 -- 最后一个键值对后面没有分号，补一个

//2.1查找key,获取key的下标
let index = cookie.indexOf(key);
// console.log(index);
let startIndex = index + key.length + 1;
// console.log(startIndex);

//2.2获取结束位置的下标
let endIndex = cookie.indexOf(';', index);

//2.3截取字符串 -- 返回对应的value并解码
return decodeURiComponent(cookie.slice(startIndex, endIndex));
}
console.log(getCookie('pwd'));//666
console.log(getCookie('user'));//甚样千玺
```

localStorage -- 本地永久存储

1）特点
①存储在客户端的浏览器上
②大小有限制--5M
③存储的数据都是字符串形式
④可以直接从本地获取数据，不需要像cookie一样每次都请求服务器
⑤ie低版本不支持
⑥永久的本地存储，只要不主动删除数据，数据就会永远存在

2）语法
设置数据 -- localStorage.setItem('key','value') -- 复杂数据类型使用JSON.stringify转换为JSON格式的字符串
获取数据 -- localStorage.getItem('key') -- 复杂数据类型使用JSON.parse转换为对象
删除数据 -- localStorage.removeItem('key')
清空数据 -- localStorage.clear()

```
//H5新增本地存储
//localStorage -- 永久存储
//存储数据
localStorage.setItem('user', '古力娜扎');
localStorage.setItem('pwd', '666');
let obj = { name: '王昭君', age: 18 };

//参数需要一个字符串
//把对象转换为字符串，使用JSON.stringify()把对象转换为字符串
localStorage.setItem('boyFriend', JSON.stringify(obj));

//获取数据
console.log(localStorage.getItem('user'));
//获取数据时返回json格式的字符串，通过JSON.parse()转换为对象
console.log(JSON.parse(localStorage.getItem('boyFriend')));

//删除数据
localStorage.removeItem('pwd');

//清空数据
localStorage.clear();

console.log('----------------------------');
//sessionStorage()--临时存储
//存储数据
sessionStorage.setItem('user', '迪丽热巴');
sessionStorage.setItem('boyFriend', JSON.stringify({ name: '肖战' }));

//获取数据
console.log(sessionStorage.getItem('user'));
console.log(JSON.parse(sessionStorage.getItem('boyFriend')));

//删除数据
sessionStorage.removeItem('user');

//清空数据
sessionStorage.clear();
```

sessionStorage -- 本地临时存储

1）特点
①存储在客户端的浏览器上
②大小有限制--5M
③存储的数据都是字符串形式
④可以直接从本地获取数据，不需要像cookie一样每次都请求服务器
⑤ie低版本不支持
⑥本地临时存储，只要关闭浏览器，数据就消失

2）语法
设置数据 -- sessionStorage.setItem('key','value') -- 复杂数据类型使用JSON.stringify转换为JSON格式的字符串
获取数据 -- sessionStorage.getItem('key') -- 复杂数据类型使用JSON.parse转换为对象
删除数据 -- sessionStorage.removeItem('key')
清空数据 -- sessionStorage.clear()

1.4区别
1）存储大小，cookie--4kb，storage--5M
2）有效时间，cookie可以设置有效时间，localStorage--永久有效，sessionStorage--关闭浏览器失效
3）兼容性，cookie没有兼容性问题，storage在ie以下不支持
4）网络，cookie每次都请求服务器，storage本地有存储可以直接使用
5）方法，cookie方法单一，storage有五个方法