**跨域的概念**

浏览器的同源策略，出于防范跨站脚本的攻击，禁止客户端脚本（如 JavaScript）对不同域的服务进行跨站调用。

一般的，只要网站的 协议名protocol、 主机host、 端口号port 这三个中的任意一个不同，网站间的数据请求与传输便构成了跨域调用。

同源策略会限制以下几种行为：

1. Cookie, LocalStorage和IndexDB无法获取
2. DOM 和 JS 对象无法获取
3. AJAX请求无法发送

常见的跨域类型包括以下一些：

| **URL** | **说明** | **是否允许通信** |
| --- | --- | --- |
| <http://www.example.com/a.js> <https://www.example.com/b.js> | 不同协议 | 不允许 |
| <http://www.example1.com/a.js> <http://www.example2.com/b.js> | 不同域名 | 不允许 |
| <http://www.example.com/a.js> <http://192.168.1.1/b.js> | 域名与域名所对应的IP | 不允许 |
| <http://www.example.com/a.js> <http://x.example.com/b.js> <http://example.com/c.js> | 主域相同，子域不同 | 不允许 |
| <http://www.example.com:8080/a.js> <http://www.example.com/b.js> | 同个域名，不同端口 | 不允许 |
| <http://www.example.com/a.js> <http://www.example.com/b.js> <http://www.example.com/lab/c.js> | 同域名同端口，不同文件目录 | 允许 |

**跨域问题解决方案**

1. jsonp
2. Ajax
3. CORS
4. document.domain + iframe
5. window.postMessage()
6. window.name + iframe
7. nginx 代理

**1.jsonp跨域**

原理：动态生成一个script标签，插入head中，浏览器会执行script标签中的代码，但只能实现get请求

具体原生实现

<script>

var script = document.createElement('script');

script.type = 'text/javascript';

script.src = 'http://www.example.com?name=michael&callback=onCallback';

document.head.appendChild(script);

function onCallback(res) {

console.log(JSON.stringify(res));

// 在这里处理数据

}

</script>

// 服务端返回，返回后执行onCallback函数

onCallback({resultCode: 0, data: {}});

**2.Ajax请求**

$.ajax({

url: 'http://www.example.com',

type: 'get',

dataType: 'jsonp',

jsonpCallback: 'onCallback',

data: {}

})

**3.CROS方式**

Cross-Origin Reasource Sharing 跨域资源共享可以避开浏览器的同源策略，但CROS不仅仅支持GET请求，还支持其他请求

**方式：**

在服务器的返回信息里对请求头进行设置：

1. Access-Control-Allow-Origin => \*
2. Access-Control-Allow-Headers => X-Requested-With
3. Access-Control-Allow-Methods => PUT, POST, GET, DELETE, OPTIONS

**4.Document.domain + iframe**

**条件：**

页面 <http://www.example.com/a.html>

页面中有一个iframe <http://iframe.com/b.html>

**方式：**

将两个页面的document.domain 设置成相同的域名，就可以在页面中拿到iframe中的数据

**限制：**

只能把document.domain设置成自身或更高一级的父域

**实现：**

//a.html

document.domain = 'example.com';

var iframe = document.createElement('iframe');

iframe.src = 'http://iframe.com/b.html';

iframe.style.display = 'none';

document.body.appendChild(iframe);

iframe.onload = function() {

var doc = iframe.contentDocument || iframe.contentWindow.document;

console.log(doc)

}

//b.html

document.domain = 'example.com';

**5.window.postMessage()**

HTML5的新特性，允许来自不同源的脚本采用异步方式进行通信，实现跨域传递消息

使用方法：

postMessage(data, origin)

data: html5规范支持任意基本类型或可复制的对象，但部分浏览器只支持字符串，所以传参时最好用JSON.stringify()序列化

origin: 协议+主机+端口号，也可以设置为"\*"，表示可以传递给任意窗口，如果要指定和当前窗口同源的话设置为"/"

**实现**

// http://www.example.com/a.html

<iframe src="http://www.example2.com/b.html" style="display: none;" id="ifr"></iframe>

<script>

var iframe = document.getElementById('ifr');

iframe.onload = function() {

var data = {};

iframe.contentWindow.postMessage(JSON.stringify(data), 'http://www.example2.com');

};

// 接受传输的数据

window.addEventListener('message', function(e) {

console.log(e.data);

}, false);

</script>

// http://www.example2.com/b.html

<script>

window.addEventListener('message', function(e) {

console.log(e.data);

})

</script>

**6.window.name + iframe**

**原理：window.name 在不同的页面加载后依然存在，最大为2M**