（1）常见跨域场景

URL	说明	是否允许通信
http://www.domain.com/a.js http://www.domain.com/b.js http://www.domain.com/lab/c.js	同一域名，不同文件或路径	允许
http://www.domain.com:8000/a.js http://www.domain.com/b.js	同一域名，不同端口	不允许
http://www.domain.com/a.js https://www.domain.com/b.js	同一域名，不同协议	不允许
http://www.domain.com/a.js http://192.168.4.12/b.js	域名和域名对应相同ip	不允许
http://www.domain.com/a.js http://x.domain.com/b.js http://domain.com/c.js	主域相同，子域不同	不允许
http://www.domain1.com/a.js http://www.domain2.com/b.js	不同域名	不允许

（2）跨域解决方案

1. 通过jsonp跨域
2. document.domain + iframe跨域
3. location.hash + iframe
4. window.name + iframe跨域
5. postMessage跨域
6. 跨域资源共享（CORS）
7. nginx代理跨域
8. nodejs中间件代理跨域
9. WebSocket协议跨域

（3）通过jsonp跨域

通常为了减轻web服务器的负载，我们把js、css，img等静态资源分离到另一台独立域名的服务器上，在html页面中再通过相应的标签从不同域名下加载静态资源，而被浏览器允许，基于此原理，我们可以通过动态创建script，再请求一个带参网址实现跨域通信。

（a）原生实现

```
//动态创建script标签
var script = document.createElement('script');
script.type = 'text/javascript';
// 传参并指定回调执行函数为onBack
script.src = 'http://www.domain2.com:8080/login?user=admin&callback=onBack';
document.head.appendChild(script);
// 回调执行函数
function onBack(res) {
    alert(JSON.stringify(res));
}
```

服务端返回如下（返回时即执行全局函数）：

```
onBack({"status": true, "user": "admin"})
```

（b）jquery ajax:

```
$.ajax({
    url: 'http://www.domain2.com:8080/login',
    type: 'get',
    dataType: 'jsonp',  // 请求方式为jsonp
    jsonpCallback: "onBack",    // 自定义回调函数名
    data: {}
});
```

（c）vue.js

```
this.$http.jsonp('http://www.domain2.com:8080/login', {
    params: {},
    jsonp: 'onBack'
}).then((res) => {
    console.log(res);
})
```

后端node.js代码示例：

```
var querystring = require('querystring');
var http = require('http');
var server = http.createServer();

server.on('request', function(req, res) {
    var params = qs.parse(req.url.split('?')[1]);
    var fn = params.callback;

    // jsonp返回设置
    res.writeHead(200, { 'Content-Type': 'text/javascript' });
    res.write(fn + '(' + JSON.stringify(params) + ')');
```

```
    res.end();
});

server.listen('8080');
console.log('Server is running at port 8080...');
```

jsonp缺点：只能实现 get 一种请求

（4）跨域资源共享（CORS）

普通跨域请求：服务端只设置Access-Control-Allow-Origin即可，前端无须设置，若要带cookie请求：前后端都需要设置。

CORS是一个W3C标准，全称是"跨域资源共享"（Cross-origin resource sharing）。它允许浏览器向跨源(协议 + 域名 + 端口)服务器，发出XMLHttpRequest请求，从而克服了AJAX只能同源使用的限制。（IE10+）

支持CORS请求的浏览器一旦发现ajax请求跨域，会对请求做一些特殊处理，对于已经实现CORS接口的服务端，接受请求，并做出回应。

有一种情况比较特殊，如果我们发送的跨域请求为“非简单请求”，浏览器会在发出此请求之前首先发送一个请求类型为OPTIONS的“预检请求”，验证请求源是否为服务端允许源，这些对于开发这来说是感觉不到的，由浏览器代理。

总而言之，客户端不需要对跨域请求做任何特殊处理。

因此，实现CORS通信的关键是服务器。只要服务器实现了CORS接口，就可以跨源通信。

- 原生ajax

```
// 前端设置是否带cookie
xhr.withCredentials = true;
```

示例代码：

```
var xhr = new XMLHttpRequest(); // IE8/9需用window.XDomainRequest兼容

// 前端设置是否带cookie
xhr.withCredentials = true;

xhr.open('post', 'http://www.domain2.com:8080/login', true);
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
xhr.send('user=admin');

xhr.onreadystatechange = function() {
    if (xhr.readyState == 4 && xhr.status == 200) {
        alert(xhr.responseText);
    }
};
```

- jQuery ajax

```
$.ajax({
    ...
   xhrFields: {
       withCredentials: true    // 前端设置是否带cookie
   },
   crossDomain: true,   // 会让请求头中包含跨域的额外信息，但不会含cookie
    ...
});
```

- vue框架

在vue-resource封装的ajax组件中加入以下代码：

```
Vue.http.options.credentials = true
```

2、 服务端设置：

若后端设置成功，前端浏览器控制台则不会出现跨域报错信息，反之，说明没设成功。

1.) Java后台：

```
/*
 * 导入包：import javax.servlet.http.HttpServletResponse;
 * 接口参数中定义：HttpServletResponse response
 */
// 允许跨域访问的域名：若有端口需写全（协议+域名+端口），若没有端口末尾不用加'/'
response.setHeader("Access-Control-Allow-Origin", "http://www.domain1.com");
// 允许前端带认证cookie：启用此项后，上面的域名不能为'*'，必须指定具体的域名，否则浏览器会提示
response.setHeader("Access-Control-Allow-Credentials", "true");
```

2.) Nodejs后台示例：

```
var http = require('http');
var server = http.createServer();
var qs = require('querystring');

server.on('request', function(req, res) {
    var postData = '';
    // 数据块接收中
    req.addListener('data', function(chunk) {
        postData += chunk;
    });
    // 数据接收完毕
    req.addListener('end', function() {
        postData = qs.parse(postData);
        // 跨域后台设置
        res.writeHead(200, {
            'Access-Control-Allow-Credentials': 'true',     // 后端允许发送Cookie
            'Access-Control-Allow-Origin': 'http://www.domain1.com',    // 允许访问的域（协议+域名
+端口）
```

```
            /*
             * 此处设置的cookie还是domain2的而非domain1，因为后端也不能跨域写cookie(nginx反向代理可以实现)，
             * 但只要domain2中写入一次cookie认证，后面的跨域接口都能从domain2中获取cookie，从而实现所有的接口都能跨域访问
             */
            'Set-Cookie': 'l=a123456;Path=/;Domain=www.domain2.com;HttpOnly'  // HttpOnly的作用是让js无法读取cookie
        });
        res.write(JSON.stringify(postData));
        res.end();
    });
});
server.listen('8080');
console.log('Server is running at port 8080...');
```

（5）nginx代理跨域

跨域原理： 同源策略是浏览器的安全策略，不是HTTP协议的一部分。服务器端调用HTTP接口只是使用HTTP协议，不会执行JS脚本，不需要同源策略，也就不存在跨越问题。

实现思路：通过nginx配置一个代理服务器（域名与domain1相同，端口不同）做跳板机，反向代理访问domain2接口，并且可以顺便修改cookie中domain信息，方便当前域cookie写入，实现跨域登录。

```
#proxy服务器
server {
    listen       81;
    server_name  www.domain1.com;
    location / {
        proxy_pass   http://www.domain2.com:8080;  #反向代理
        proxy_cookie_domain www.domain2.com www.domain1.com; #修改cookie里域名
        index  index.html index.htm;
        # 当用webpack-dev-server等中间件代理接口访问nignx时，此时无浏览器参与，故没有同源限制，下面的跨域配置可不启用
        add_header Access-Control-Allow-Origin http://www.domain1.com;  #当前端只跨域不带cookie时，可为*
        add_header Access-Control-Allow-Credentials true;
    }
    location /qt {
        alias  E:/dist;
        index  index.html;
        try_files $uri $uri/ /index.html;
    }
    location /case {
        proxy_pass   http://192.168.35.29:8700;
    }
    location /cmt/ope {
        alias   E:/WS/cmt-operation;
        index welcome.html;
    }
    location /cmt/service {
        proxy_pass   http://192.168.9.24:8900/service;
```

```
    }
    location /cmt/service/operation/upload {
        proxy_pass   http://172.16.11.119/upload;
    }
}
```

```
var xhr = new XMLHttpRequest();
// 前端开关：浏览器是否读写cookie
xhr.withCredentials = true;
// 访问nginx中的代理服务器
xhr.open('get', 'http://www.domain1.com:81/?user=admin', true);
xhr.send();
```