henu_Newxc03
作者相关精选

浏览器同源策略与如何解决跨域问题总结

浏览器同源策略与如何解决跨域问题总结

发布于 2022-05-05 18:20:33 1.5K 0 举报

文章被收录于专栏： Newxc03的前端之路

什么是同源策略

跨域问题实际就是浏览器的同源策略造成的。

> 同源策略限制了从同一个源加载的文档或脚本如何与另一个源的资源进行交互。这是浏览器的一个用于隔离潜在恶意文件的重要安全机制。同源指的是： 协议 、 端口号 、 域名 必须一致。

下表给出了与 URL http://store.company.com/dir/page.html 的源进行对比的示例:

URL	是否跨域	原因
http://store.company.com/dir/page.html	同源	完全相同
http://store.company.com/dir/inner/another.html	同源	只有路径不同
https://store.company.com/secure.html	跨域	协议不同
http://store.company.com:81/dir/etc.html	跨域	端口不同（http:// 默认端口是80）
http://news.company.com/dir/other.html	跨域	主机不同

在这里插入图片描述

同源策略：protocol(协议)、domain(域名)、port(端口)三者必须一致

同源策略主要限制了三个方面：

当前域下的 js 脚本不能够访问其他域下的 cookie、localStorage 和 indexDB

当前域下的 js 脚本不能够操作访问操作其他域下的 DOM。

当前域下ajax无法发送跨域请求

同源政策的目的主要是为了保证用户的信息安全，它只是对 `js 脚本` 的一种限制，并不是对浏览器的限制，对于一般的 `img` 、或者 `script` 脚本请求都不会有跨域的限制，这是因为这些操作都不会通过响应结果来进行可能出现安全问题的操作。

如何解决跨域问题

(1) CORS

下面是MDN对于CORS的定义：

> 跨域资源共享(CORS) 是一种机制，它使用额外的 HTTP 头来告诉浏览器 让运行在一个 origin(domain)上的Web应用被准许访问来自不同源 服务器 上的指定的资源。当一个资源从与该资源本身所在的服务器不同的域、协议或端口请求一个资源时，资源会发起一个跨域HTTP 请求。

CORS需要 浏览器 和 服务器 同时支持，整个CORS过程都是浏览器完成的， 无需用户参与 。因此实现CORS的**关键就是服务器，只要服务器实现了CORS请求，就可以跨源通信了。**

浏览器将CORS分为**简单请求**和**非简单请求**:

简单请求不会触发CORS预检请求。若该请求满足以下两个条件，就可以看作是简单请求：

1）请求方法是以下三种方法之一：

HEAD

目录

henu_Newxc03
作者相关精选

浏览器同源策略与如何解决跨域问题总结

2）HTTP的头信息不超出以下几种字段：

Accept

Accept-Language

Content-Language

Last-Event-ID

Content-Type：只限于三个值application/x-www-form-urlencoded、multipart/form-data、text/plain

若不满足以上条件，就属于非简单请求了

简单请求过程：

对于简单请求，浏览器会直接发出CORS请求，它会在请求的头信息中增加一个Orign字段，该字段用来说明本次请求来自哪个源（协议+端口+域名），服务器会根据这个值来决定是否同意这次请求。如果Orign指定的域名在许可范围之内，服务器返回的响应就会多出以下信息头：

```
Access-Control-Allow-Origin: http://api.bob.com // 和Orign一直
Access-Control-Allow-Credentials: true // 表示是否允许发送Cookie
Access-Control-Expose-Headers: FooBar // 指定返回其他字段的值
Content-Type: text/html; charset=utf-8 // 表示文档类型
```

如果Orign指定的域名不在许可范围之内，服务器会返回一个正常的HTTP回应，浏览器发现没有上面的Access-Control-Allow-Origin头部信息，就知道出错了。这个错误无法通过状态码识别，因为返回的状态码可能是200。

在简单请求中，在服务器内，至少需要设置字段： `Access-Control-Allow-Origin`

非简单请求过程：

非简单请求是对服务器有特殊要求的请求，比如请求方法为DELETE或者PUT等。非简单请求的CORS请求会在正式通信之前进行一次HTTP查询请求，称为 预检请求 。

浏览器会询问服务器，当前所在的网页是否在服务器允许访问的范围内，以及可以使用哪些HTTP请求方式和头信息字段，只有得到肯定的回复，才会进行正式的HTTP请求，否则就会报错。

预检请求使用的请求方法是 `OPTIONS` ，表示这个请求是来询问的。他的头信息中的关键字段是 `Orign` ，表示请求来自哪个源。除此之外，头信息中还包括两个字段：

Access-Control-Request-Method：该字段是必须的，用来列出浏览器的CORS请求会用到哪些HTTP方法。

Access-Control-Request-Headers： 该字段是一个逗号分隔的字符串，指定浏览器CORS请求会额外发送的头信息字段。

服务器在收到浏览器的预检请求之后，会根据头信息的三个字段来进行判断，如果返回的头信息在中有 `Access-Control-Allow-Origin` 这个字段就是允许跨域请求，如果没有，就是不同意这个预检请求，就会报错。

服务器回应的CORS的字段如下：

```
Access-Control-Allow-Origin: http://api.bob.com // 允许跨域的源地址
Access-Control-Allow-Methods: GET, POST, PUT // 服务器支持的所有跨域请求的方法
Access-Control-Allow-Headers: X-Custom-Header // 服务器支持的所有头信息字段
Access-Control-Allow-Credentials: true // 表示是否允许发送Cookie
Access-Control-Max-Age: 1728000 // 用来指定本次预检请求的有效期，单位为秒
```

只要服务器通过了预检请求，在以后每次的CORS请求都会自带一个 `Origin` 头信息字段。服务器的回应，也都会有一个 `Access-Control-Allow-Origin` 头信息字段。

在非简单请求中，至少需要设置以下字段：

复制

```
'Access-Control-Allow-Origin'
'Access-Control-Allow-Methods'
'Access-Control-Allow-Headers'
```

减少OPTIONS请求次数：

OPTIONS请求次数过多就会损耗页面加载的性能，降低用户体验度。所以尽量要减少OPTIONS请求次数，可以后端在请求的返回头部添加： `Access-Control-Max-Age: number` 。它表示预检请求的返回结果可以被缓存多久，单位是秒。该字段**只对完全一样的URL的缓存设置生效，所以设置了缓存时间，在这个时间范围内，再次发送请求就不需要进行预检请求了。**

CORS中Cookie相关问题：

在CORS请求中，如果想要传递Cookie，就要满足以下三个条件：

在请求中设置 `withCredentials`

默认情况下在跨域请求，浏览器是不带 cookie 的。但是我们可以通过设置 `withCredentials` 来进行传递 cookie

```
// 原生 xml 的设置方式
var xhr = new XMLHttpRequest();
xhr.withCredentials = true;
```

目录

henu_Newxc03
作者相关精选

浏览器同源策略与如何解决跨域问题总结

`Access-Control-Allow-Credentials` **设置为 true**

`Access-Control-Allow-Origin` **设置为false**

(2) JSONP

jsonp的原理就是利用 `<script>` 标签没有跨域限制，通过 `<script>` 标签src属性，发送带有callback参数的GET请求，服务端将接口返回数据拼凑到callback函数中，返回给浏览器，浏览器解析执行，从而前端拿到callback函数返回的数据。

1）原生JS实现

```
<script>
  let script = document.createElement("script");
  script.type = "text/javascript";
  //传参一个回调函数名给后端，方便后端返回时执行这个在前端定义的回调函数
  script.src =
    "https://www.domain2.com:8080/login?user=admin&callback=handleCallback";
  document.head.appendChild(script);
  //回调执行函数
  function handleCallback(res) {
    alert(JSON.stringify(res));
  }
</script>
```

服务端返回如下(返回时即执行全局函数):

```
handleCallback({"success":true,"user":"admin"})
```

2）Vue axios实现

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

后端nodejs代码

```
var querystring = require('querystring');
var http = require('http');
var server = http.createServer();
server.on('request', function(req, res) {
var params = querystring.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...');
```

(3) postMessage跨域

postMessage是 HTML5 XMLHttpRequest Level 2中的 API ，且是为数不多可以跨域操作的window属性之一，它可用于解决以下方面的问题：

页面和其打开的新窗口的数据传递

多窗口之间消息传递

页面与嵌套的iframe消息传递

上面三个场景的跨域数据传递

用法：postMessage(data,origin)方法接受两个参数：

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"/"。

1） a.html:(domain1.com/a.html)

henu_Newxc03
作者相关精选

浏览器同源策略与如何解决跨域问题总结

```
style="display:none;"></iframe>
<script>
var iframe = document.getElementById('iframe');
iframe.onload = function() {
var data = {
name: 'aym'
};
// 向domain2传送跨域数据
iframe.contentWindow.postMessage(JSON.stringify(data),
'http://www.domain2.com');
};
// 接受domain2返回数据
window.addEventListener('message', function(e) {
alert('data from domain2 ---> ' + e.data);
}, false);
</script>
```

2） b.html：(domain2.com/b.html)

```
<script>
// 接收domain1的数据
window.addEventListener('message', function(e) {
alert('data from domain1 ---> ' + e.data);
var data = JSON.parse(e.data);
if (data) {
data.number = 16;
// 处理后再发回domain1
window.parent.postMessage(JSON.stringify(data),
'http://www.domain1.com');
}
}, false);
</script>
```

(4) nginx代理跨域

nginx代理跨域，实质和CORS跨域原理一样，通过配置文件设置请求响应头 Access-Control-Allow-Origin… 等字段

1）nginx配置解决iconfont跨域

浏览器跨域访问js、css、img等常规静态资源被同源策略许可，但iconfont字体文件(eot|otf|ttf|woff|svg)例外，此时可在nginx的静态资源服务器中加入以下配置。

```
location / {
add_header Access-Control-Allow-Origin *;
}
```

2）nginx反向代理接口跨域

跨域问题：同源策略仅是针对浏览器的安全策略。服务器端调用HTTP接口只是使用HTTP协议，不需要同源策略，也就不存在跨域问题。

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，实现跨域访问。

```
#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
    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;
    }
}
```

(5) nodejs 中间件代理跨域

henu_Newxc03
作者相关精选

浏览器同源策略与如何解决跨域问题总结

1）非vue框架的跨域

使用node + express + http-proxy-middleware搭建一个proxy服务器。

前端代码

```
var xhr = new XMLHttpRequest();
// 前端开关：浏览器是否读写cookie
xhr.withCredentials = true;
// 访问http-proxy-middleware代理服务器
xhr.open('get', 'http://www.domain1.com:3000/login?user=admin', true);
xhr.send();
```

中间件服务器代码

```
var express = require('express');
var proxy = require('http-proxy-middleware');
var app = express();
app.use('/', proxy({
// 代理跨域目标接口
target: 'http://www.domain2.com:8080',
changeOrigin: true,
// 修改响应头信息，实现跨域并允许带cookie
onProxyRes: function(proxyRes, req, res) {
res.header('Access-Control-Allow-Origin', 'http://www.domain1.com');
res.header('Access-Control-Allow-Credentials', 'true');
},
// 修改响应信息中的cookie域名
cookieDomainRewrite: 'www.domain1.com' // 可以为false，表示不修改
}));
app.listen(3000);
console.log('Proxy server is listen at port 3000...');
```

2）vue框架的跨域

node + vue + webpack + webpack-dev-server搭建的项目，跨域请求接口，直接修改webpack.config.js配置。开发环境下，vue渲染服务和接口代理服务都是webpack-dev-server同一个，所以页面与代理接口之间不再跨域。

webpack.config.js部分配置

```
module.exports = {
    entry: {},
    module: {},
    ...
    devServer: {
        historyApiFallback: true,
        proxy: [{
        context: '/login',
        target: 'http://www.domain2.com:8080', // 代理跨域目标接口
        changeOrigin: true,
        secure: false, // 当代理某些https服务报错时用
        cookieDomainRewrite: 'www.domain1.com' // 可以为false，表示不修改
        }],
        noInfo: true
    }
}
```


access http 网络安全 json nginx

评论

登录 后参与评论