有哪些你了解的后端实现跨域的方式?

对于跨域，相信同学们都有所了解。前端的跨域的若干中方式，大家也都知道，什么 JSONP，iframe+domain 等等。但是我们今天的主题，不是前端跨域，而是**后端跨域**。

对于前端同学来说，一旦提及到跨域，就会想到同源策略，那我们就来回顾跨域和同源策略。

一、什么是跨域请求

首先，我们要了解什么是跨域请求。简单来说，当一台服务器资源从另一台服务器（不同的域名或者端口）请求一个资源或者接口，就会发起一个跨域 HTTP 请求。

举个简单的例子，从 http://demo-a.com/index.html,发送一个 Ajax 请求，请求地址是 http://demo-b.com/下面的一个接口，这就是发起了一个跨域请求。

在不做任何处理的情况下，这个跨域请求是无法被成功请求的，因为浏览器基于同源策略会对跨域请求做一定的限制。那什么又是同源策略呢？

二、同源策略

首先大家要知道同源策略发生的场景——浏览器中，什么意思呢？如果不是浏览器的话，就不会受到同源策略的影响。也就是说，两个服务器直接进行跨域请求是可以进行数据请求的。这也就为我们接下来的后端跨域埋下一下小伏笔。

同源策略的目的是什么呢？同源策略限制了从同一个源加载的文档或者脚本如何与来自另一个源的资源进行交互。这是一个用于隔离潜在恶意文件的重要安全机制。

那什么又是同源？

同源需要同时满足三个条件：

1. 请求的协议相同（例如同为 http 协议）
2. 请求的域名相同（例如同为 http://demo.com）
3. 请求的端口相同（同为 80 端口）

第 2 点需要注意的是，必须是域名完全相同，比如说 a.example.com 和 b.example.com 这两个域名，虽然它们的顶级域名和二级域名（均为 example.com）都相同，但是三级域名（a 和 b）不相同，所以也不能算作域名相同。

如果不同时满足这上面三个条件，那就不符合浏览器的同源策略。

需要注意的是，不是所有的交互都会被同源策略拦截下来，下面两种交互就不会触发同源策略：

- 跨域写操作（Cross-origin writes），例如超链接、重定向以及表单的提交操作，特定少数的 HTTP 请求需要添加预检请求（preflight）；
- 跨域资源嵌入（Cross-origin embedding）：
  - <script> 标签嵌入的跨域脚本；
  - <link> 标签嵌入的 CSS 文件；
  - <img> 标签嵌入图片；
  - <video> 和 <audio> 标签嵌入多媒体资源；
  - <object>, <embed>, <applet> 的插件；
  - @font-face 引入的字体，一些浏览器允许跨域字体（cross-origin fonts），一些需要同源字体（same-origin fonts）；
  - <frame> 和 <iframe> 载入的任何资源，站点可以使用 X-Frame-Options 消息头来组织这种形式的跨域交互。

接下来我们利用 Node 服务开启两个服务器，产生跨域问题，在之后我们看看是如何解决的。上代码：

server1 是文件服务器

server1.js

```
const app = express();
app.use(express.static('page'))
app.listen(8080);
```

server2 是数据服务器

server2.js

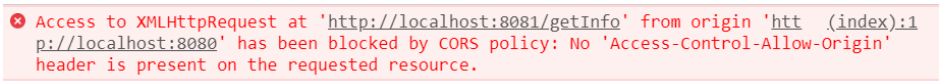
```
const express = require('express');
const app = express();

app.get('/getInfo', (req, res) => {
    res.end('你好')
})
app.listen(8081);
```

当我们在 server1 下的 page 下的 index.html 发送 Ajax 请求：

page/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
</head>
<body>
    我是 server1
    <script>
       axios.get('http://localhost:8081/getInfo')
       .then(data => console.log(data))
       .catch(err => console.log(err))
       // 这里的 axios 是一个用于发送 Ajax 的工具
    </script>
</body>
</html>
```


当我们在 http://localhost:8080/index.html 中访问 http://localhost:8081/getInfo 这个接口时，浏览器会报错，报错信息如下：



这样的问题，我们看一下就知道，这次请求不满足同源策略，所以驳回了。这个驳回，其实 server2 能接收到了 index.html 发送的请求，再返回结果时，被浏览器忽略了。所以我们看不到返回结果，并且给我们一个原因。

接下来我们从服务器的角度来解决以上的跨域问题。

三、从服务器端解决跨域问题

从服务器角度解决跨域问题，主要的方式有两种，一种是服务器代理，一种是 CORS。

1.服务器代理

你想访问其他域名下的资源，但是浏览器不允许。一旦访问会触发同源策略，然后 GG。

既然浏览器不允许，我们就不再浏览器层面发送跨域请求，那还需要数据，咋办？在服务器端进行跨域请求

之前我们说过，同源策略只存在浏览器中，服务器中可没有同源策略一说，这样的话，我们可以让服务器替我们发送一个请求，请求其他服务器下面的数据。然后我们的页面访问当前服务器下的接口，就没有问题了。

没错，就是这样。接下来代码改写：

server1.js

```
const express = require('express');
const app = express();
const Axios = require('axios');

app.use(express.static('page'))
app.get('/getInfo', (req, res) => {
    Axios.get('http://localhost:8081/getInfo').then(data => {
        res.end(data.data)
    }).catch(err => console.log(err))
})

app.listen(8080);
```

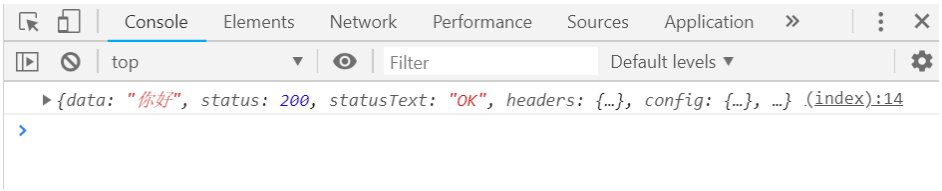
在 server1.js 服务器中发送一个请求，请求服务器 2 中的数据，由于不存在同源策略，是可以拿到的，在服务器 1 中把拿到的数据，以自身的接口暴露出去。让当前页面访问，就可以把数据拿到了。

page/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
</head>
<body>
```

```
    我是 server1
    <script>
        axios.get('http://localhost:8080/getInfo')
        .then(data => console.log(data))
        .catch(err => console.log(err))
        // 这里的 axios 是一个用于发送 Ajax 的工具
    </script>
</body>
</html>
```

请求结果：



说明我们这么做是可以的。这种方式被称为代理

接下来说说 CORS

2. CORS

CORS（Cross-origin resource sharing），跨域资源共享。

CORS 其实是浏览器制定的一个规范，它的实现则主要在服务端，它通过一些 HTTP Header 来限制可以访问的域，例如页面 A 需要访问 B 服务器上的数据，如果 B 服务器上声明了允许 A 的域名访问，那么从 A 到 B 的跨域请求就可以完成。

对于那些会对服务器数据产生副作用的 HTTP 请求，浏览器会使用 OPTIONS 方法发起一个预检请求（preflight request），从而可以获知服务器端是否允许该跨域请求，服务器端确认允许后，才会发起实际的请求。在预检请求的返回中，服务器端也可以告知客户端是否需要身份认证信息。

我们只需要设置响应头，即可进行跨域请求。

- Access-Control-Request-Method：用于预检请求，作用是将实际请求所使用 HTTP 方法告诉服务器；
- Access-Control-Request-Headers：用于预检请求，作用是将实际请求所使用的 Header 字段告诉服务器；

HTTP 响应 Headers：

- Access-Control-Allow-Origin：指定了允许访问该资源的外域 URI；

在代码中实现

server1.js

```
const express = require('express');
const app = express();


app.get('/getInfo', (req, res) => {

    res.setHeader('Access-Control-Allow-Origin', '*');
    res.setHeader('Access-Control-Allow-Methods', 'GET');
    res.setHeader('Access-Control-Allow-Header','x-request-with,content-type')
    res.end('你好')
})
app.listen(8081);
```

这样修改完之后，我们在服务器 1 下正常发送 AJax 请求，请求 http://localhost:8081/getInfo，这样就可以正常进行跨域了。