浏览器有同源策略：

当你访问一个网站时虽然请求还是会发送出去但是不让你的JS拿到响应内容也就形成了不能跨过这个域去访问其它域的功能

这里的域指的是协议，端口号和主机都需要一致的URL



换句话说这两个URL必须同源才允许交互

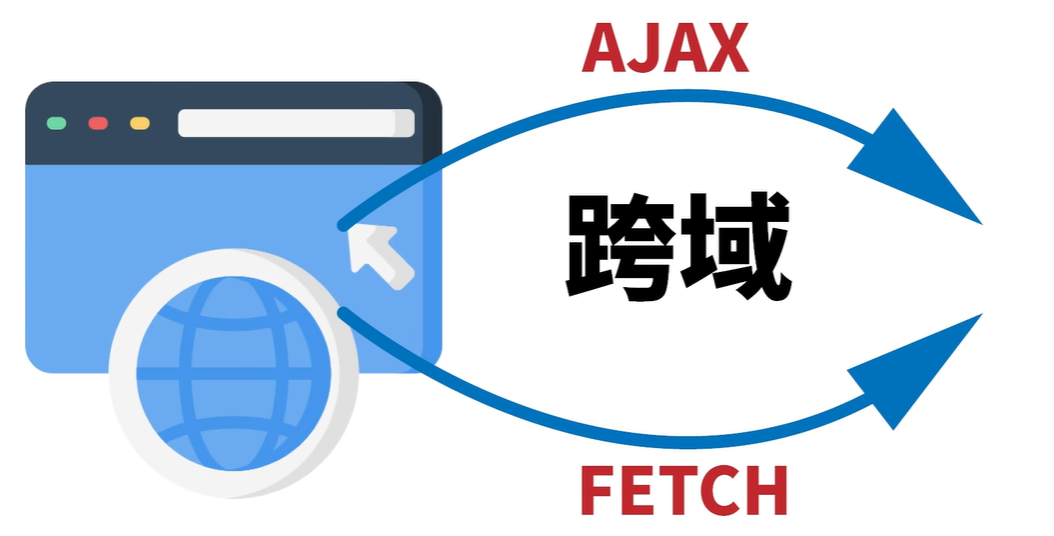


但是他们都是隶属于bilibili.com这个域名的

B站现在需要让这两个源的资源进行交互

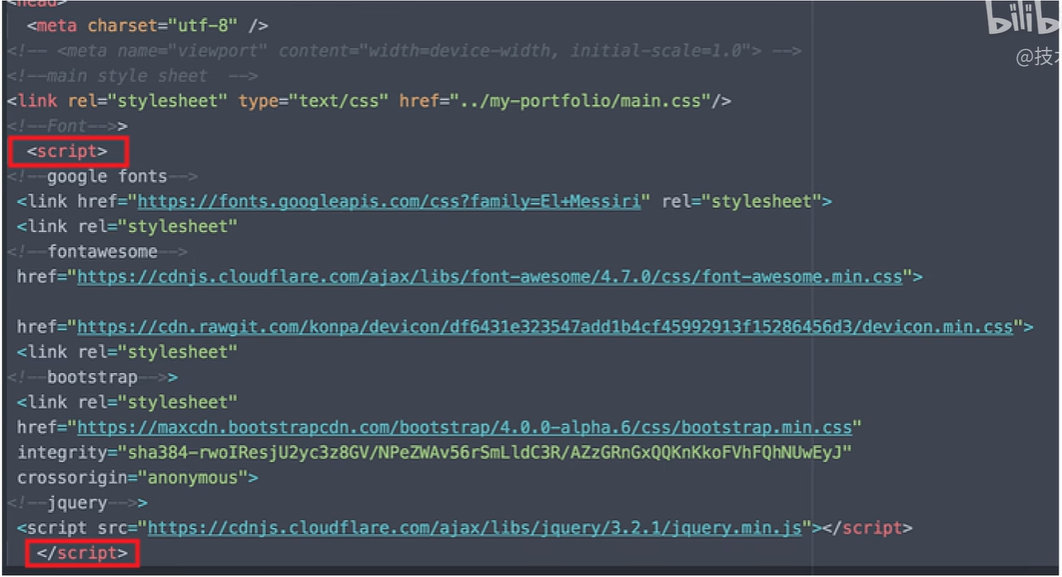
这就需要用到跨域的技术了

我们从另一个角度来看跨域这个问题



其实就是在用Ajax进行请求的时候存在访问问题，当然Fetch也是这个问题

为什么在script标签引用了别的源又不会出现问题呢



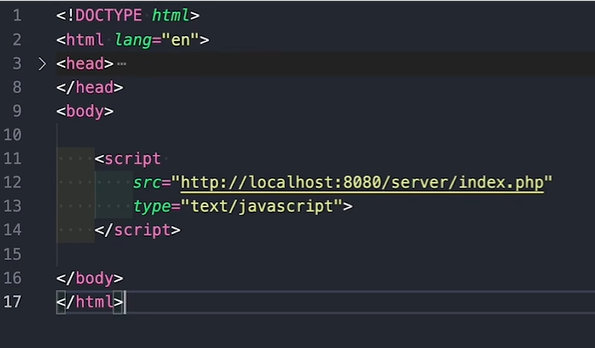
这是因为当时在设计这个script标签的时候就允许在别的源请求脚本

所以就有很聪明的开发人员利用这个“漏洞”来进行跨域

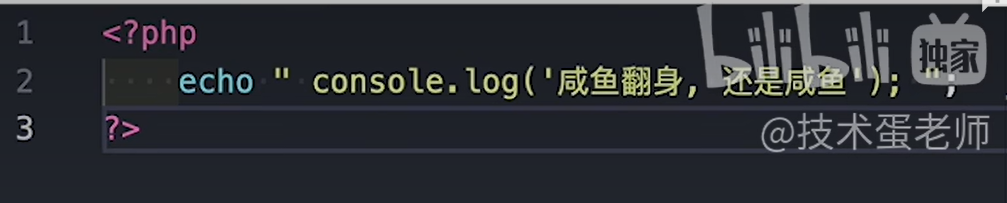
这个方法就叫做JSONP

先不解释这个名字，来看看一个交互过程

我们用HTML和JS来模拟客户端



我们用PHP来模拟服务端

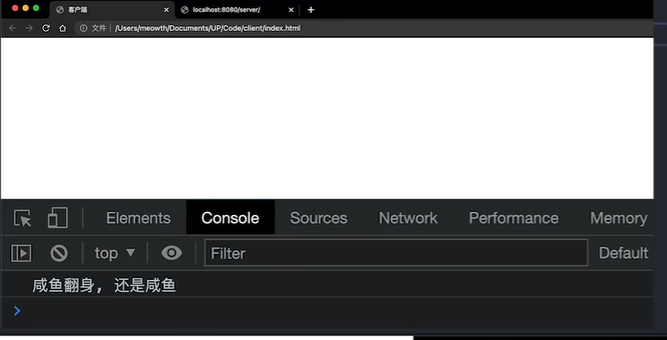


大家可以看到控制台和刚刚php文件在浏览器里的内容是不一样的

php中输出：



html控制台中输出：

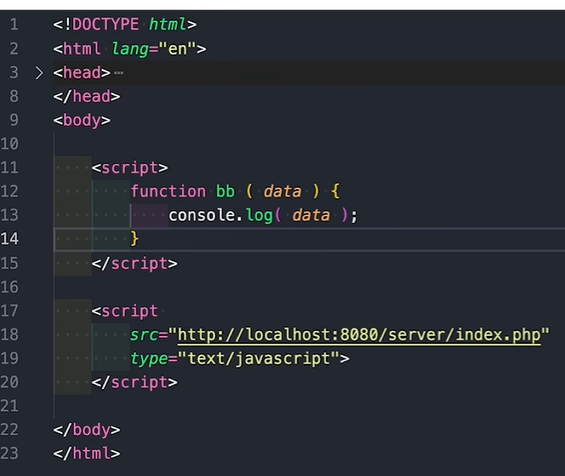


这是因为HTML5里的script标签默认的type属性是text/javascript

正是因为定义了这个类型请求的内容会被浏览器执行在这里就是被浏览器以JS代码来进行执行所以就会出现这种结果

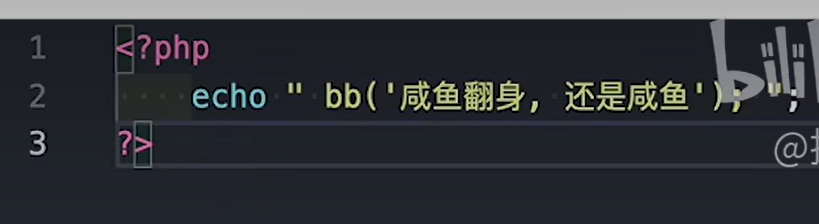
这就为跨域提供了可能性

这里修改一下代码并且在里面定义一个函数bb然后在控制台输出里面的参数



但是在客户端这里没有明文地显示调用这个bb函数

而是在服务端这边把echo里面的字符串改为JS代码



我们先来快速复习一下URL小知识

很多同学都知道URL基本是由协议，域名，端口和路径组成



但想必很多同学有见过URL上有问好”？"

从?开始表示客户端提供给服务器的额外参数

而这些参数是键值对，用&来进行分割的

服务器看到这样的请求后

而且已经提前设置好代码的话

就会使用这些参数来执行额外的操作

我们就可以利用这点来获取服务器的数据了

客户端和服务端常见通用的格式就是JSON格式了

因此我们可以把数据都设置为JSON格式

但是因为我们利用script的方式获取数据script会被执行因此我们一般会给数据外包一个函数

然后再包一层作为传输用的JSON格式当然真实数据也是可以为JSON数据

也就是说我们在客户端提前设置好一个函数这个函数专门处理跨域获取服务器JSON格式的数据

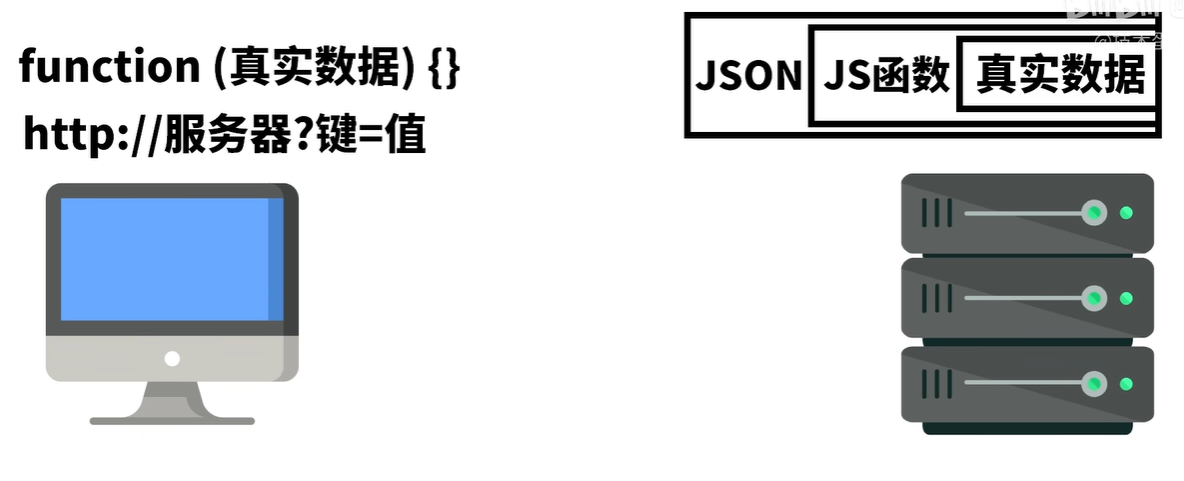
然后在URL上提供给服务器额外参数服务器看到这些额外参数就会执行相应的代码

然后把数据外包一个JS函数再以JSON数据的形式回传给客户端

客户端收到后，调用提前设置好的函数执行服务器传过来的数据也就是执行函数，并且获取数据

简单来说就是不直接获取服务器的数据而是给数据包裹JS函数，变成不一样的JSON数据

去到客户端那边就能够顺理成章了因为一般都是用来JS函数包裹JSON数据



所以就称为JSON with Padding，JSONP

Padding就是裹了一层的意思

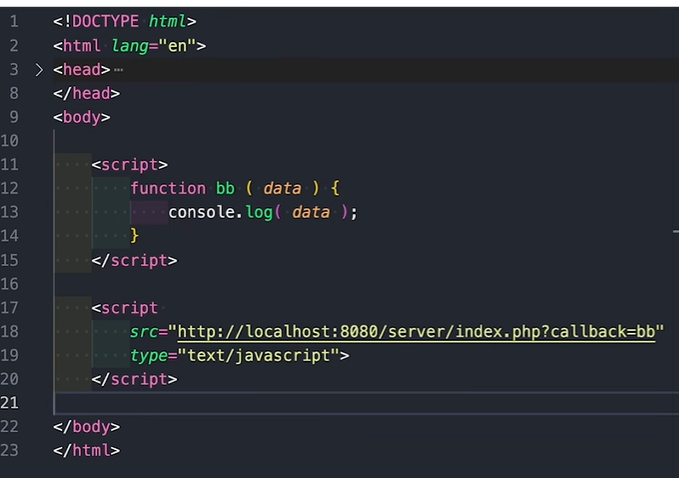
现在我们可以回到代码这边了

按照刚刚的步骤客户端提前设置函数bb

参数data就是要处理的服务器JSON格式的数据了

接着修改一下URL地址

添加?callback=bb



callback是客户端告诉服务端需要进行调用的

因为等会要回传过来，所以一般叫callback

服务器要回传的是JSON格式

因此先设置HTTP头部－内容类型

然后我们用$\_GET来收集客户端发来的值

在这里也就是收集了键值对callback对应的值bb

然后保存在一个叫callbackGET的变量里面

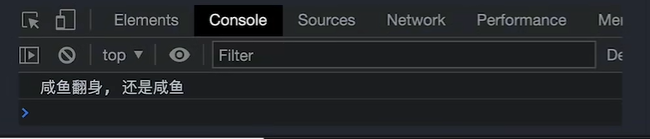
接着创建data变量，并且传入字符串

最后用echo返回callbackGET变量，并且和data变量拼接



实际返回的数据就是bb('咸鱼翻身，还是咸鱼')

我们来看一下客户端这边就能查看到数据顺利地过来了



这就是JSONP的核心原理了

当然你也可以用jQuery来实现JSONP

JSONP有一个比较大的缺点就是只支持GET，不支持POST

因为JSONP缺点比较多

因此就诞生了CORS这种方法

当浏览器在进行跨域请求的时候会在请求里添加头部origin

表明自己的协议，主机和端口号

当服务器接收到请求并且看到这个origin头部时

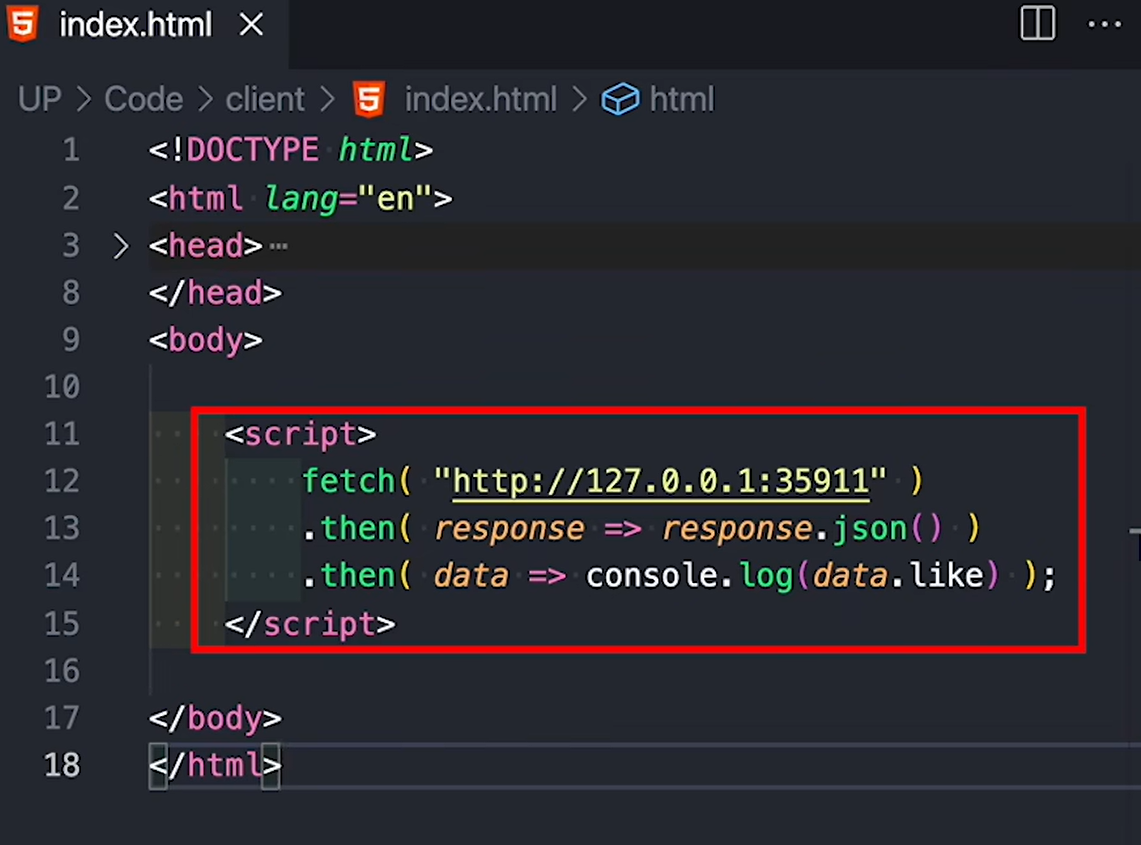
如果需要允许能够访问

就得添加头部Access-Control-Allow-Origin到响应里面



浏览器看到服务器传回来的这个头部就知道能不能进行跨域请求了

我们来看看这里的代码

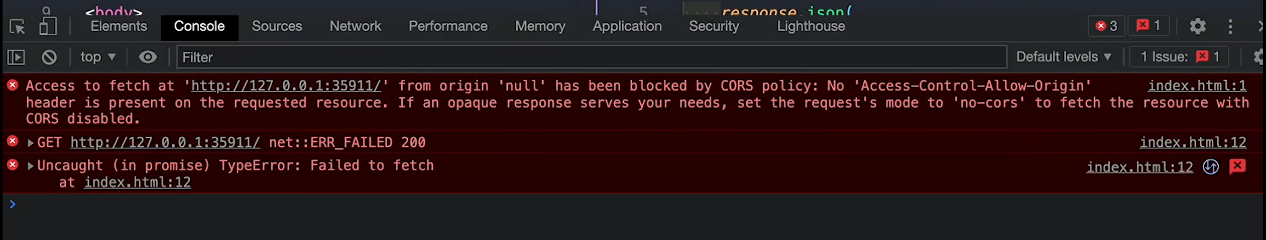


客户端这边尝试用fetch获取不同域的资源

服务端设置了返回json数据



但是客户端打开浏览器就发现错误了



错误提示的主要意思是客户端在访问服务器资源的时候被跨域资源共享政策给阻止了

原因是没有在响应里发现头部Access-Control-Allow-Origin

我们看看Network里面也确实没有发现这个头部

所以我们得先在服务器这边提前进行设置

一般服务端语言都有自己的第三方库来处理这个新的头部

这边演示的服务端语言是NodeJS

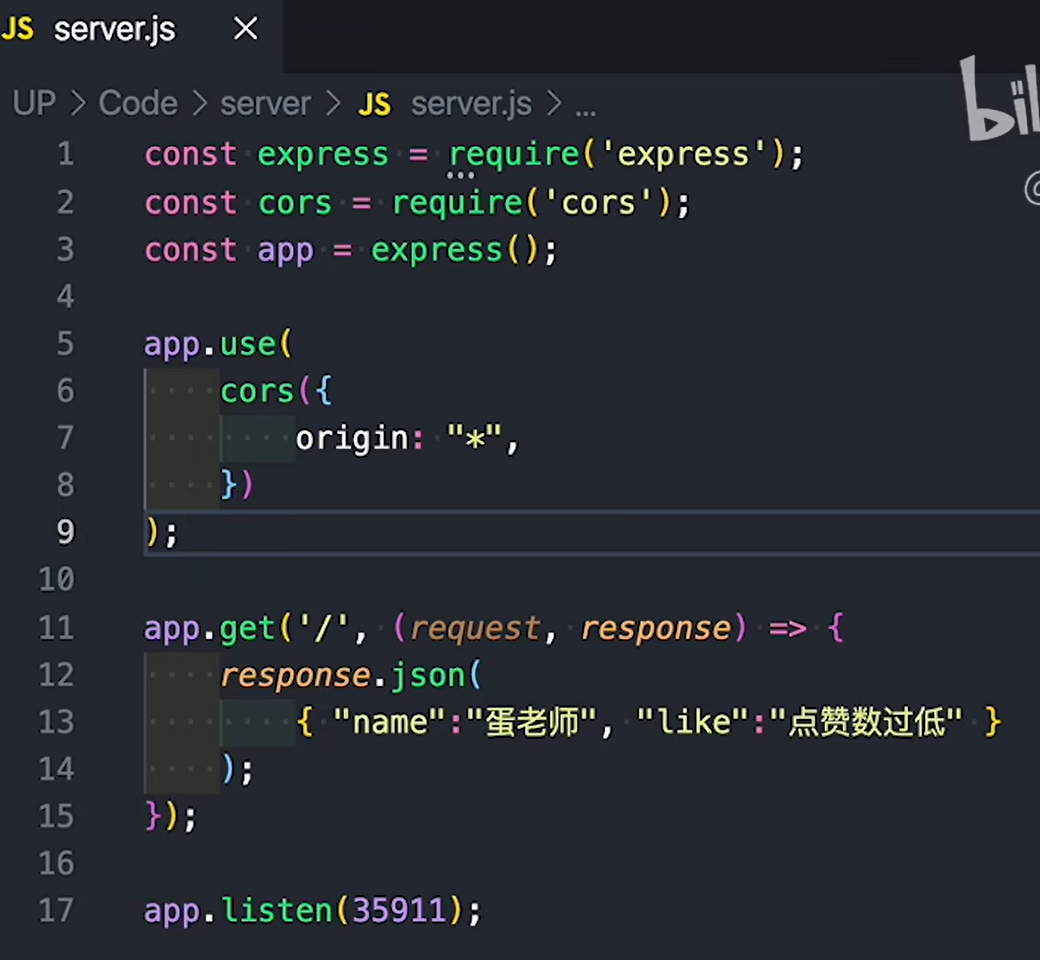
也已经提前用npm install cors配置好直接可以用了

接着我们引入这个cors然后使用这个cors并且设置origin

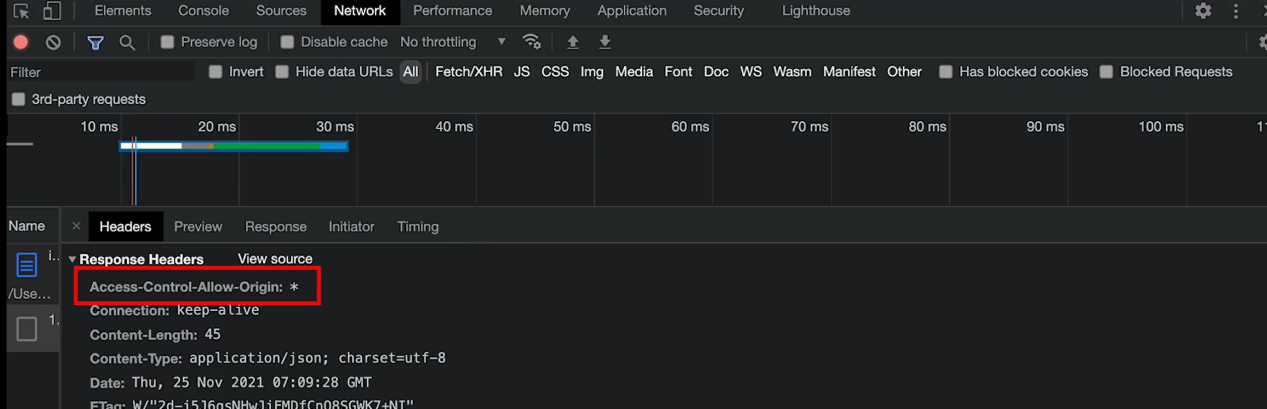
也就是允许什么源过来拿资源

协议号，主机和端口号都要写上

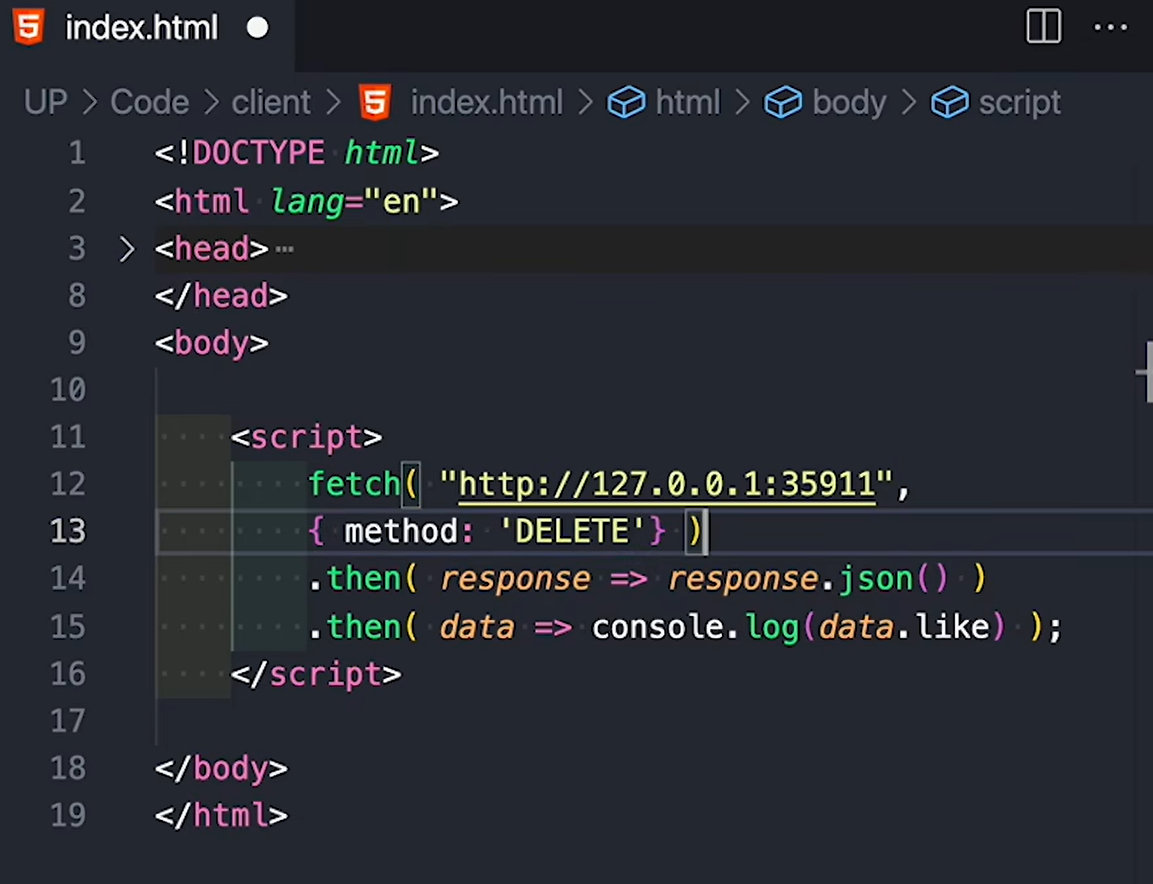
也可以用\*号来表示允许所有源



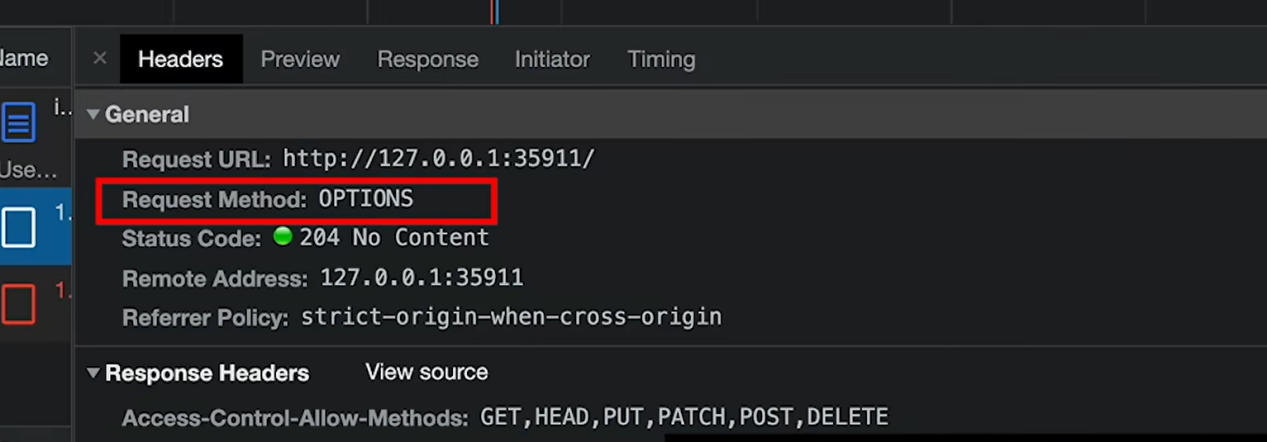
保存一下，在客户端那边有结果了



如果有人恶意地添加DELETE方法在Fetch里面我们服务器该提前进行什么样的设置呢？



先来看看Network这边显示的请求方法是OPTIONS不是DELETE



为什么会这样呢还好浏览器防了一手当用户想修改服务器数据的时候

例如使用PUT，PATCH或者DELETE方法的时候

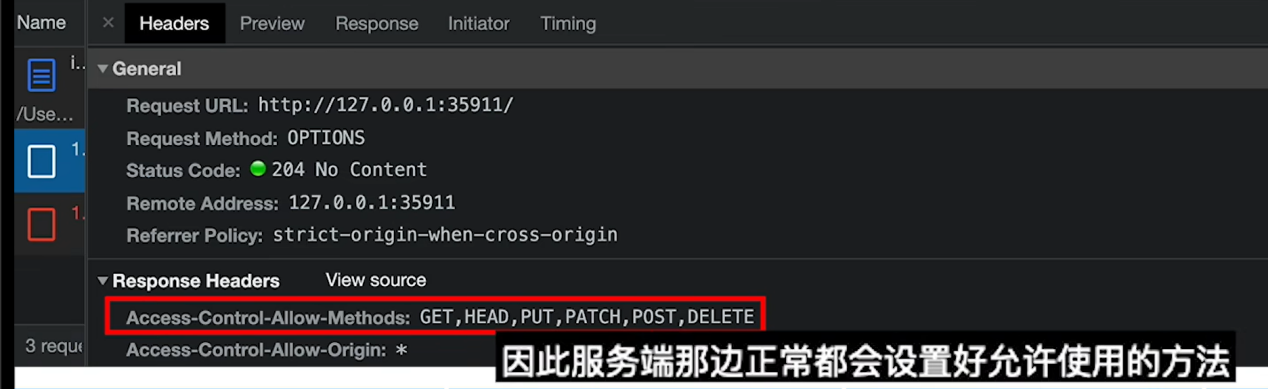
浏览器会自动发出一个预检请求

预检请求就是查着服务器是否支持当前跨域请求

因此预检请求没有具体哪个方法，而是用OPTIONS

我们也同时看到服务器那边显示了可以使用的方法知道允许以后才去实际请求

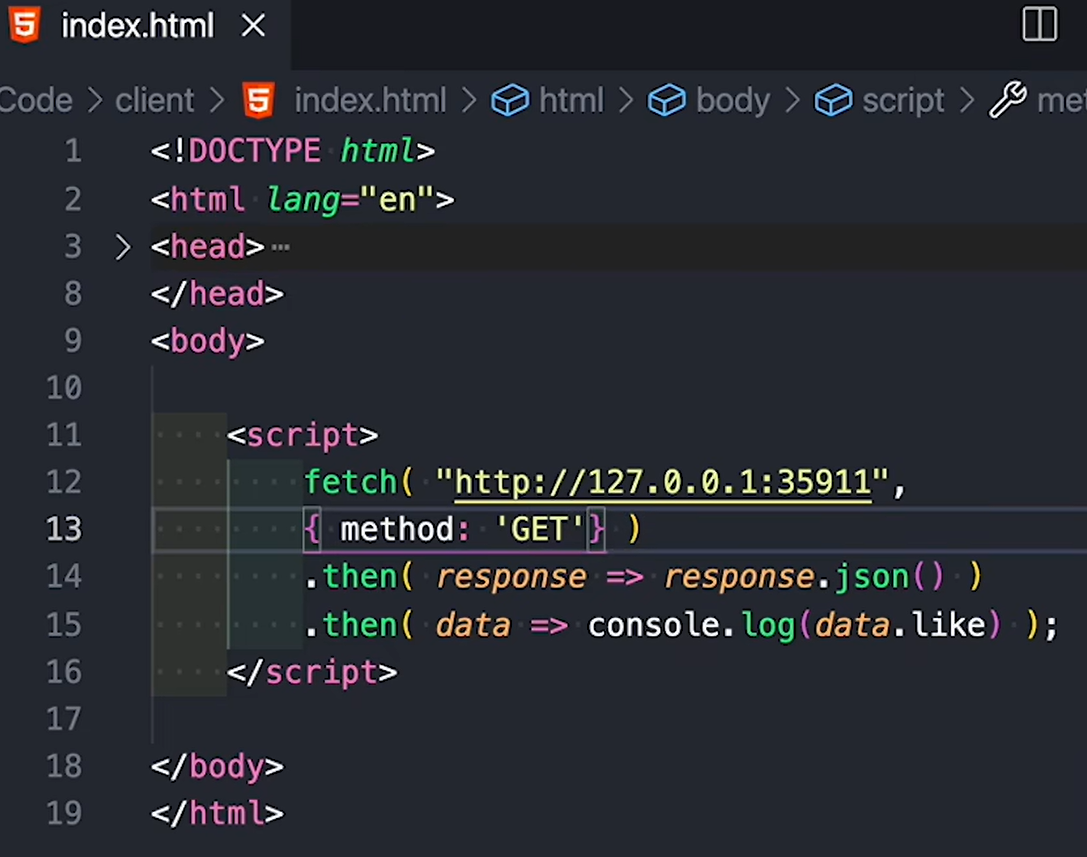
因此服务端那边正常都会设置好允许使用的方法



最常用的就是GET和POST两个方法了



这个时候客户端以正确的方法访问就不会出现错误了

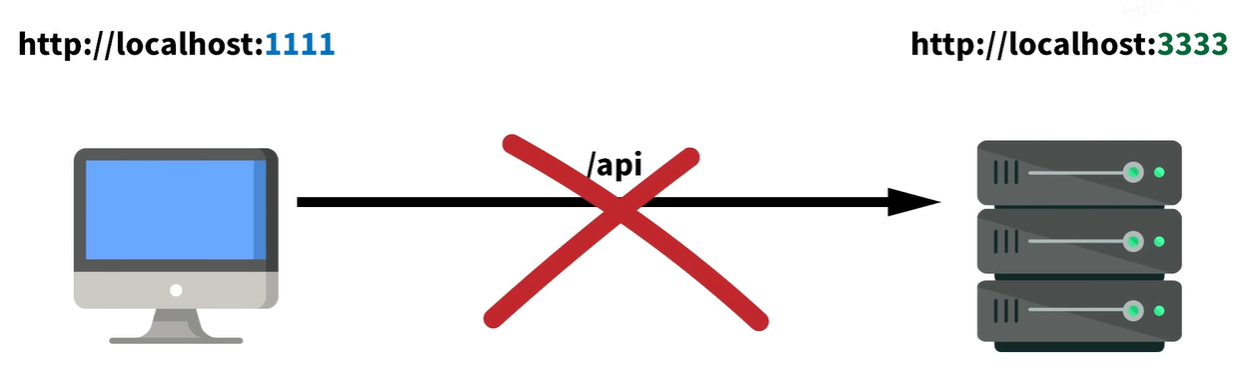


CORS的核心简单来说就是设置头部Access-Control-Allow-Origin

方法比JSONP简单多了，不过要注意兼容问题

还有一种常见解决跨域的方法就是设置反向代理

现在假设客户端这边端口是1111服务端那边端口是3333客户端的页面请求服务端/api的资源很明显会有跨域的问题

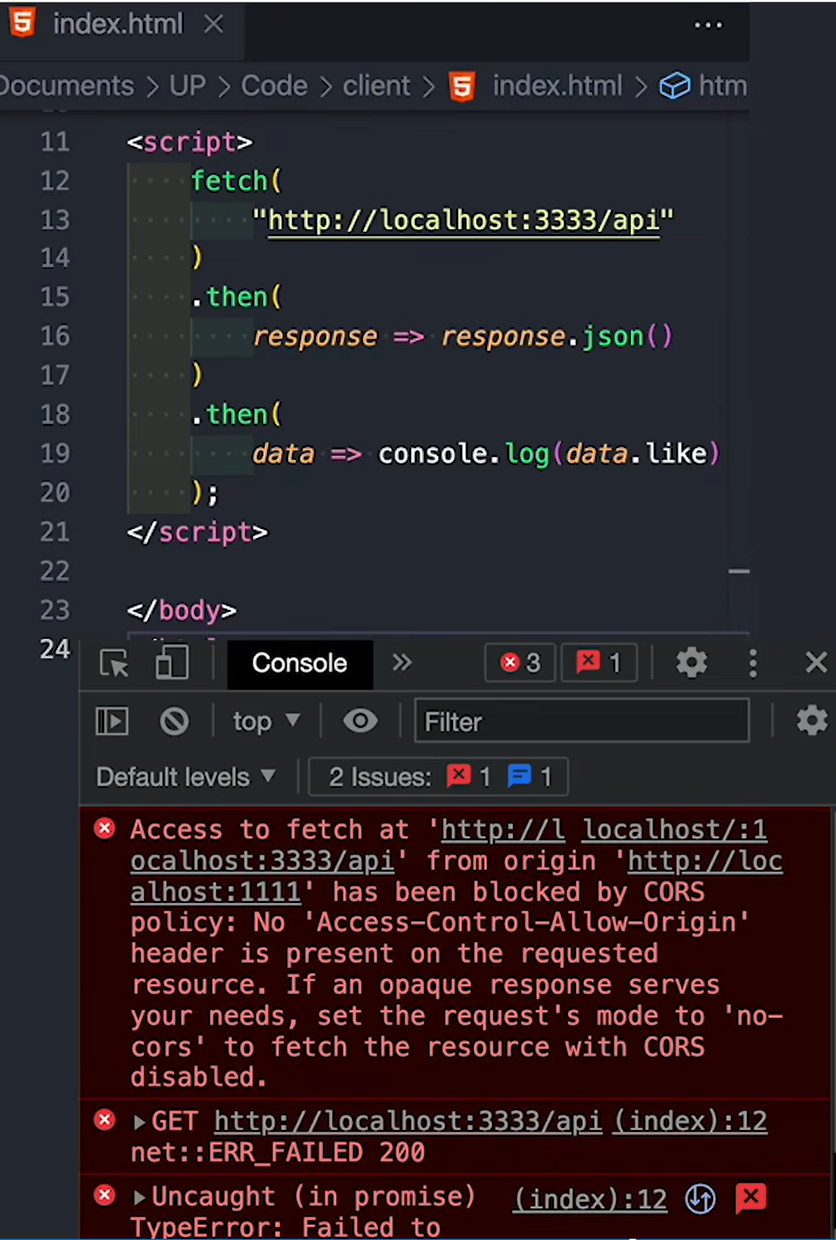


如果服务器那边配置了反向代理就可以让前端页面请求反向代理反向代理再把请求转给服务器因为反向代理和服务端这边都属于服务器之间的沟通不像前端有浏览器的同源策略限制因此就解决了跨域的问题



我们来看看代码

客户端这边我设置了获取服务端/api的资源可以看着到前端浏览器显示了跨域问题



现在我们就可以配置一下反向代理这里用nginx来演示

修改一下nginx.conf配置文件这里把nginx监听的端口设置为2222然后再设置location来识别/api地址的时候转发到服务器的地址最后加上Access-Control-Allow-Origin头部这个头部和前面说的是一致的可以指定源，这里用\*表示所有当然有些同学会说我不加这个头部也可以跨域但我还是建议你加上



现在我启动一下nginx服务





前端不直接访问服务器而是通过代理访问修改后当然也重启了一下服务

现在我们来看看前端浏览器这边就没有报错了并且把结果正确地显示出来了

