最近从小密圈看到了一个很有意思的漏洞timeline，该漏洞在国外著名漏洞赏金平台hackerone上被披露，hackerone给予了漏洞作者2.5w美刀的奖励，tql。
相关介绍在博客：

https://blog.teddykatz.com/2019/11/05/github-oauth-bypass.html?from=groupmessage&isappinstalled=0

大致意思是：

Github在处理OAuth的流程时，会显示一个提示框，询问用户是否同意赋予某个App以权限，当用户点击确认后，会发送一个POST请求到当前地址。

其处理逻辑如下：

```
if request.get?

  显示询问框

else

  用户点击确认，这里就赋予App权限

end
```

现在大部分的Web框架有默认CSRF防御，Github也不例外。默认情况下，GET以外的所有请求都会检查CSRF Token。所以，上述代码理论上是很安全的，因为如果要进入else，必须传入正确的CSRF Token。

而HEAD请求是个例外，通常来说HEAD是和GET的逻辑是完全相同的，只不过HEAD不会返回body，所以HEAD请求也不会进行CSRF检查。这个下面有图文并茂的介绍。

攻击者诱导用户发送一个HEAD请求到上面的逻辑，框架不会检查CSRF Token，然后request.get又是false（因为不是GET请求），最后进入else，成功窃取用户权限。

其实看到这里，我们基本都明白了，这是攻击者通过滥用HEAD请求的方式达到了一些非法目的。

而本文并非讲述这些细节，主要是通过这个漏洞引发了一些对http协议的思考，从本文的标题也可以看的出是讲浏览器同源策略及跨域相关问题的。

其实http是一个相对松散的文本协议。在浏览器同源策略及跨域问题上，我们可能都知道，怎样才算跨域?

三要素：

协议，域名，端口 任意一个不同都算跨域，这些知识都可以搜索到，你也都知道，没有任何问题。

我们还知道，如果是跨域，那么浏览器会首先预发送options请求，当response返回：

```
header("Access-Control-Allow-Origin: * or 你的地址");
```

你才可以组织GET或者POST请求发给服务端，服务端完成后续响应。

分割线，以上作为有一些工作经验的develop人员肯定是知道的！

现在继续一些问题：

1，如果是第一次options请求，服务端会返回内容吗?

2，如果服务端不设置以下这个response头，客户端就无法请求到内容了吗?

```
header("Access-Control-Allow-Origin: * or 你的地址");
```

下面我们开始探索，我们是追求真理的极客，又不是普通的develop人员啊，这些问题肯定得探索清楚的啊！

首先，第一个问题，如果跨域了，浏览器发起的options请求，服务端会返回内容吗?

答案：会！

你可能会说，为什么我在浏览器的network监控里没有看到options的任何内容返回？我告诉你，这是浏览器给你看到的假象，其实是发生了内容返回。

现在在/tmp目录用 php -S 0:9999 来监听本地的9999端口，对应目录下存在文件index.php

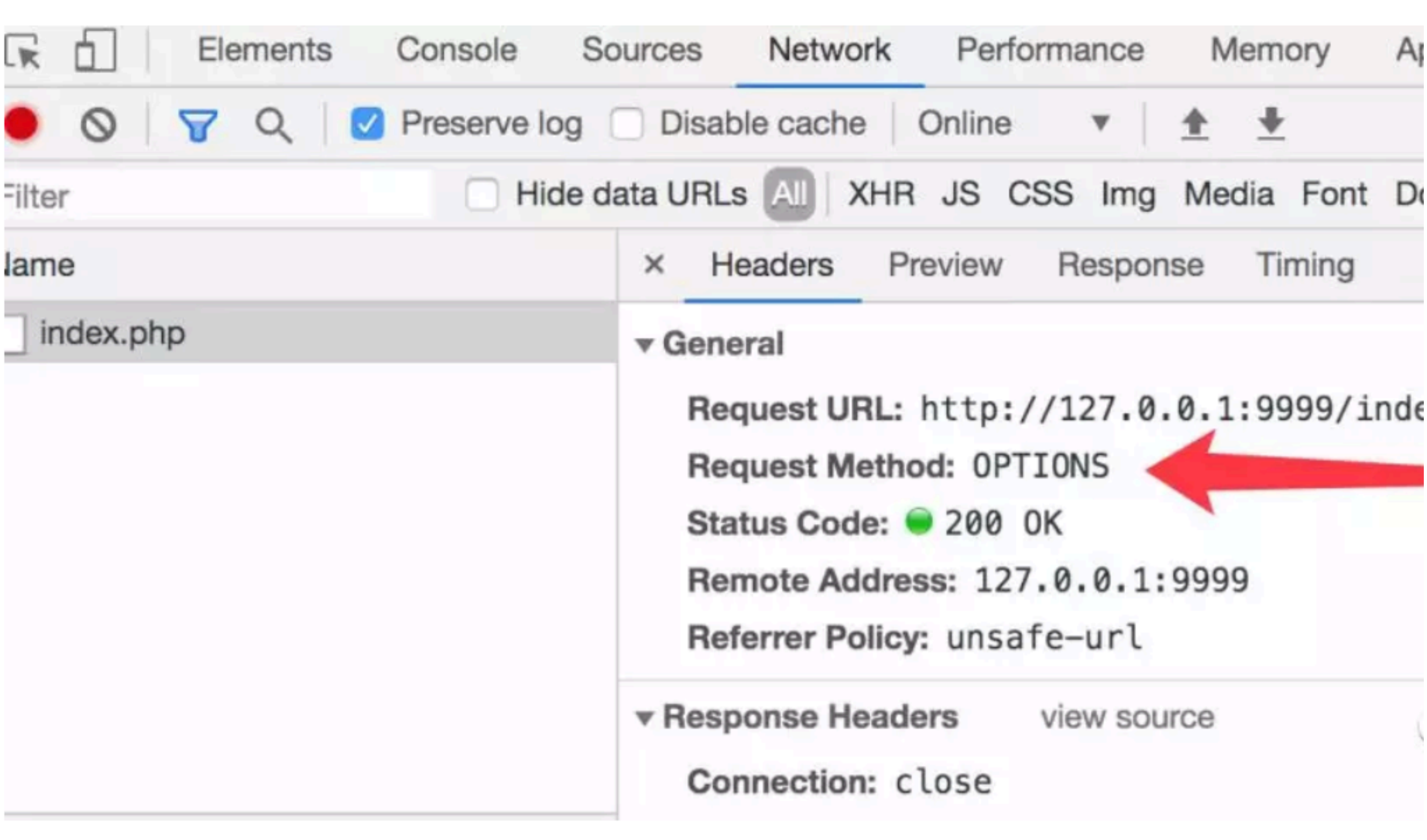
```
➜ /tmp cat index.php

<?php

echo "hello";
```

```
curl http://127.0.0.1:9999/index.php   返回 hello 没有问题
```

使用浏览器打开https://www.baidu.com 进入console界面，执行：

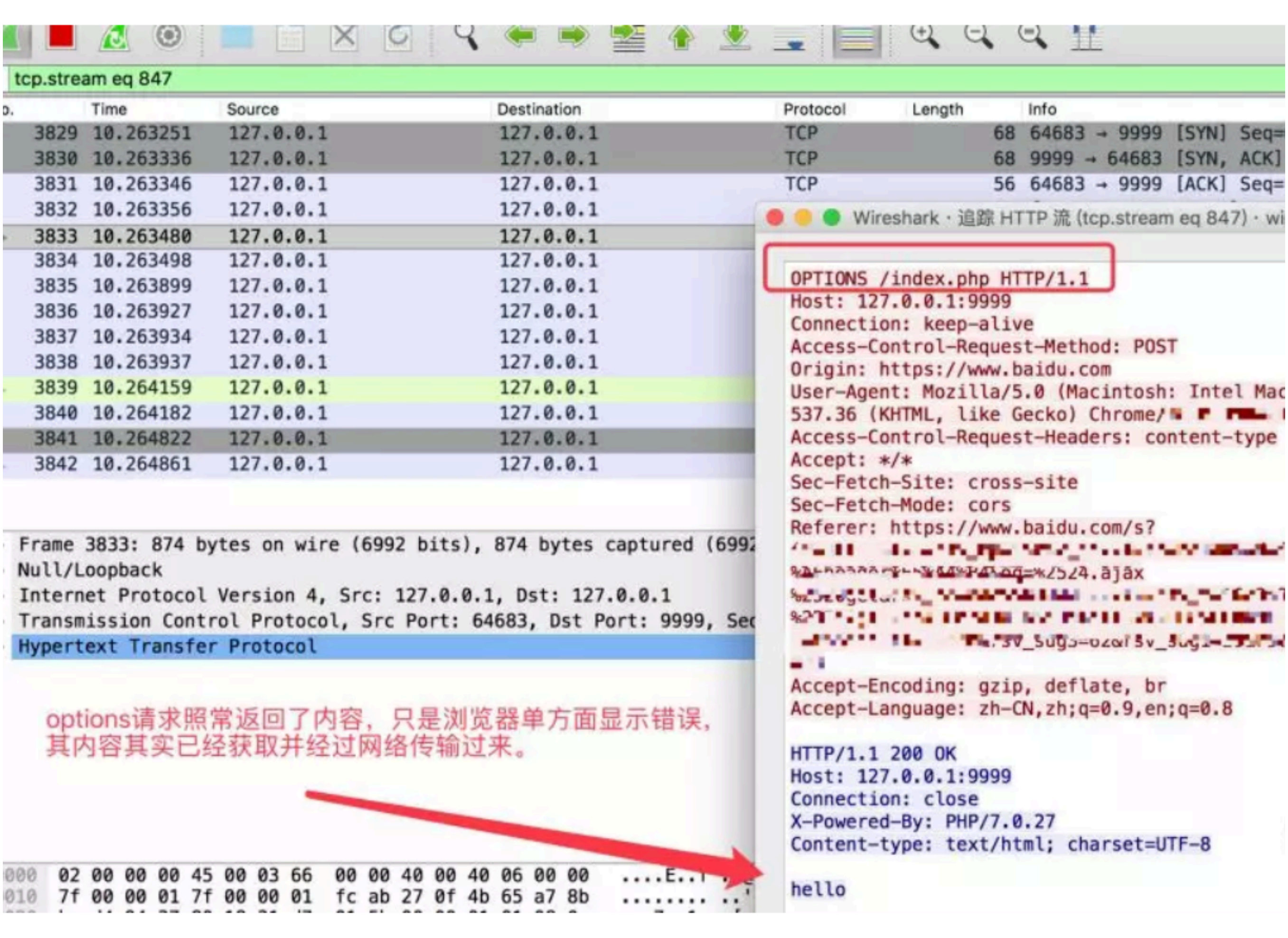
```
fetch("http://127.0.0.1:9999/index.php", {

  method: 'GET',

  credentials: 'include',

}).

catch(err = >{

  throw err;

}).then(() = >{});
```

```
> fetch(
        "http://127.0.0.1:9999/index.php",
        {
          method: 'GET',
          credentials: 'include',
        }
      ).catch(err => {
        throw err;
      }).then(() => {
      });
< ▸Promise {<pending>}
✖ Access to fetch at 'http://127.0.0.1:9999/index.php' from origin 'https://www.baidu.com' has been blocked by CORS policy: No 'A
  on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource wi
✖ ▸Uncaught (in promise) TypeError: Failed to fetch
```



Request URL: http://127.0.0.1:9999/inde
Request Method: OPTIONS
Status Code: 200 OK
Remote Address: 127.0.0.1:9999
Referrer Policy: unsafe-url
Connection: close

可以看到此时浏览器console出了一个红色警告，network确实是options请求，我们抓包看一看：



```
o.   Time       Source     Destination  Protocol  Length  Info
3829 10.263251  127.0.0.1  127.0.0.1    TCP       68  64683 → 9999 [SYN] Seq=
3830 10.263336  127.0.0.1  127.0.0.1    TCP       68  9999 → 64683 [SYN, ACK]
3831 10.263346  127.0.0.1  127.0.0.1    TCP       56  64683 → 9999 [ACK] Seq=
3832 10.263356  127.0.0.1  127.0.0.1
3833 10.263480  127.0.0.1  127.0.0.1
3834 10.263498  127.0.0.1  127.0.0.1
3835 10.263899  127.0.0.1  127.0.0.1
3836 10.263927  127.0.0.1  127.0.0.1
3837 10.263934  127.0.0.1  127.0.0.1
3838 10.263937  127.0.0.1  127.0.0.1
3839 10.264159  127.0.0.1  127.0.0.1
3840 10.264182  127.0.0.1  127.0.0.1
3841 10.264822  127.0.0.1  127.0.0.1
3842 10.264861  127.0.0.1  127.0.0.1

Frame 3833: 874 bytes on wire (6992 bits), 874 bytes captured (6992
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 64683, Dst Port: 9999, Sec
Hypertext Transfer Protocol

000  02 00 00 00 45 00 03 66  00 00 40 00 40 06 00 00   ....E..
010  7f 00 00 01 7f 00 00 01  fc ab 27 0f 4b 65 a7 8b   ........
```

```
OPTIONS /index.php HTTP/1.1
Host: 127.0.0.1:9999
Connection: keep-alive
Access-Control-Request-Method: POST
Origin: https://www.baidu.com
User-Agent: Mozilla/5.0 (Macintosh: Intel Mac
537.36 (KHTML, like Gecko) Chrome/
Access-Control-Request-Headers: content-type
Accept: */*
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: cors
Referer: https://www.baidu.com/s?
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8

HTTP/1.1 200 OK
Host: 127.0.0.1:9999
Connection: close
X-Powered-By: PHP/7.0.27
Content-type: text/html; charset=UTF-8

hello
```

options请求照常返回了内容，只是浏览器单方面显示错误，其内容其实已经获取并经过网络传输过来。

hello字符串成功返回，只是浏览器没有看到返回，很明显，内容经过了网络传输。所以第一个问题得证，客户端尽管发送options，服务端其实是给你返回了的啊，虽然在浏览器network的response中没有看到内容，但是别信浏览器的，服务端内容其实是经过了网络传输并发送到了浏览器，只是浏览器这家伙不展示而已！

这里如果服务端响应了：

```
header("Access-Control-Allow-Origin: * ")
```

那么客户端再次请求一次，服务端在把hello 字符串再次返回一次！这一次浏览器会直接显示在network的response中。但是这里，你每次请求，服务端其实都是返回了真实数据的！不显示这个内容是浏览器的行为，红色警告也是浏览器单方面的行为！

我们再看第2个问题，如果服务端不设置以下这个response头

```
header("Access-Control-Allow-Origin: * ")
```

浏览器客户端就无法在network的response中取到内容了吗？或者说就无法在network的response中显示内容了吗?

答案：不是！

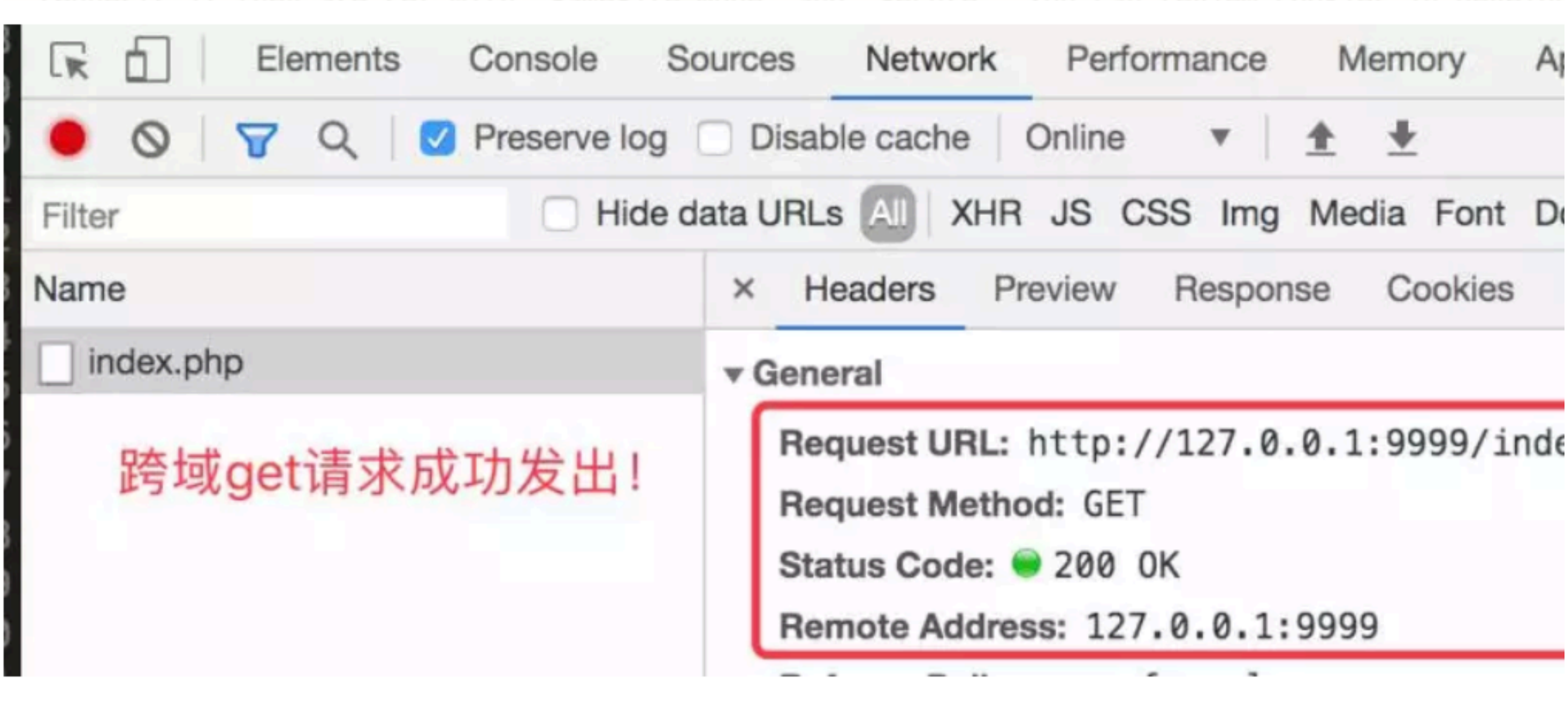
服务端不给你干的事儿，浏览器客户端你可以自己干！我们在百度的域名下浏览器console界面执行如下：

```
fetch("http://127.0.0.1:9999/index.php", {

  method: 'GET',

  credentials: 'include',

  mode: 'no-cors',

}).

catch(err = >{

  throw err;

}).then(() = >{});
```

注意这一次我加了一个参数：

```
mode: 'no-cors'
```

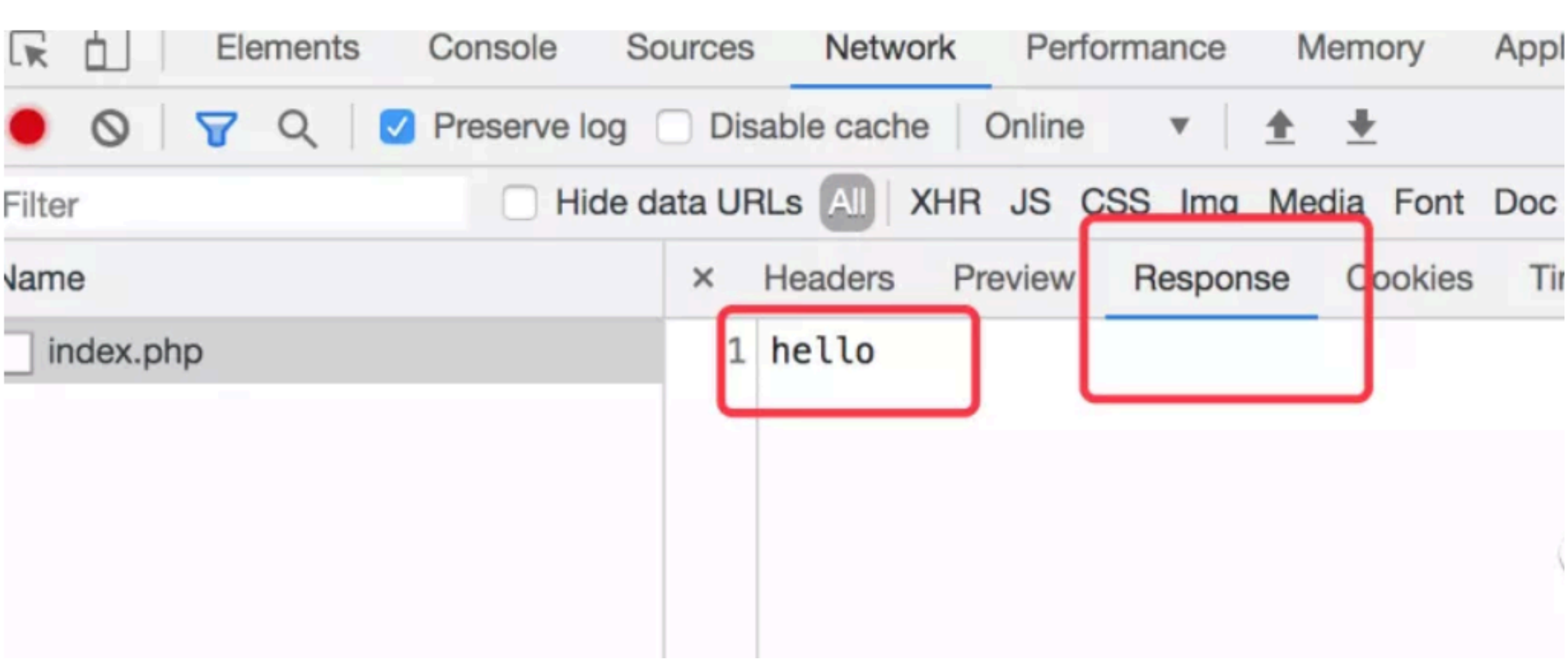
```
> fetch(
        "http://127.0.0.1:9999/index.php",
        {
          method: 'GET',
          credentials: 'include',
          mode: 'no-cors',
        }
      ).catch(err => {
        throw err;
      }).then(() => {
      });
< ▸Promise {<pending>}
⚠ A cookie associated with a cross-site resource at http://127.0.0.1/ was set without the `Same
```



跨域get请求成功发出！

Request URL: http://127.0.0.1:9999/inde
Request Method: GET
Status Code: 200 OK
Remote Address: 127.0.0.1:9999

跨域成功得到返回：



```
1 hello
```

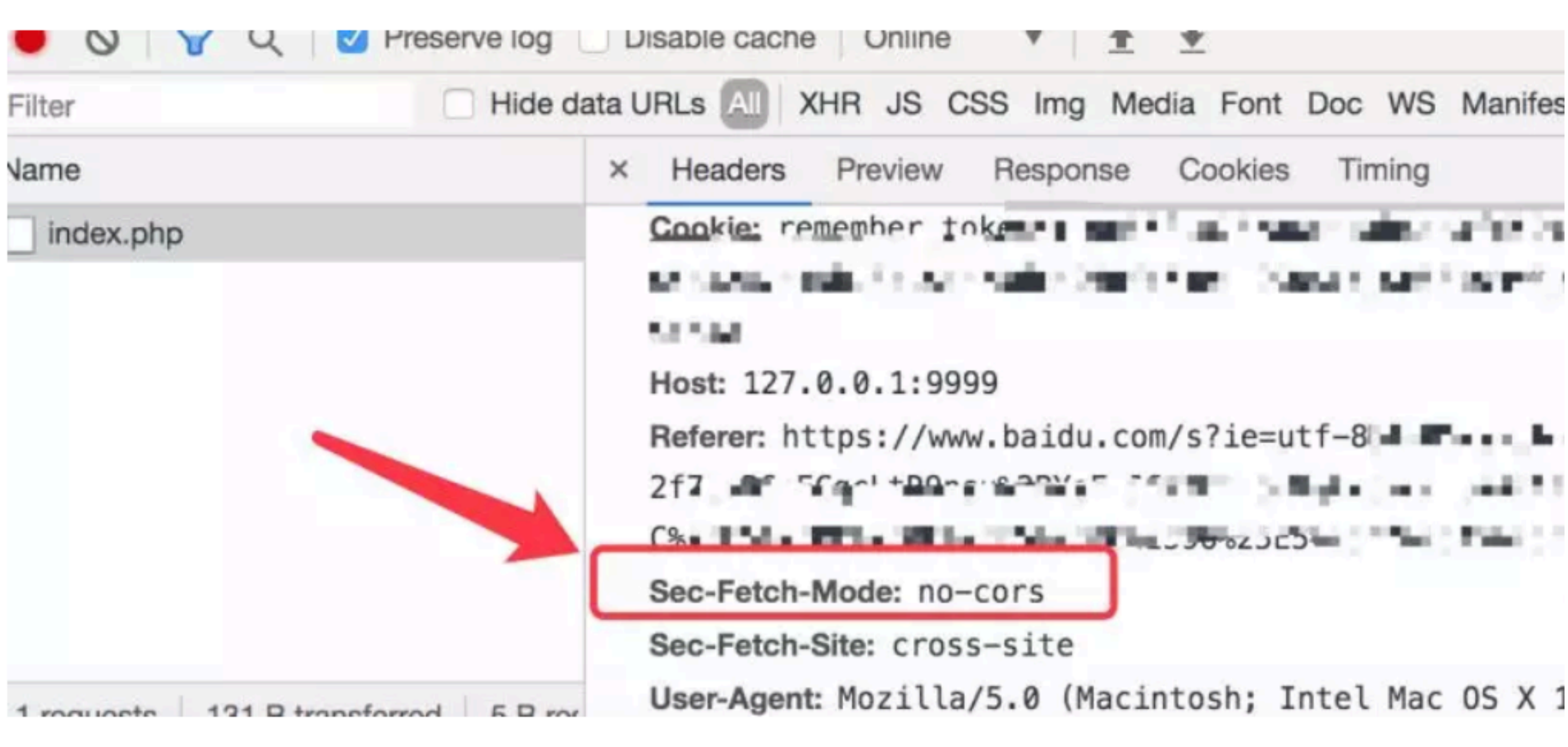
这里可以看到，即使服务端没有设置以下这个允许的头部

```
header("Access-Control-Allow-Origin: * ")
```

但是浏览器照样跨域请求成功了，服务端没有做任何更改，仅仅客户端增加了以下这个header就可以了，mode的默认值是cors，这是设置no-cors

```
mode: 'no-cors'
```

所以，读到这里，第二个问题得证，你不需要服务端在nginx层面或者程序里做：header("Access-Control-Allow-Origin: * ")的返回，客户端是有办法读取到跨域资源的，办法就是自己在fetch的时候增加：mode: 'no-cors' 的属性即可。其实就是给request的header头增加Sec-Fetch-Mode:no-cors属性即可，具体可以让前端在请求前自己加上这个header，服务端别操心这些事情了。



Cookie: remember_tok
Host: 127.0.0.1:9999
Referer: https://www.baidu.com/s?ie=utf-8
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: cross-site
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 1

最后，在说下之前说的那个HEAD请求，这个不管你设置mode等于什么，HEAD请求不会返回body体，这是协议规定，这也是大家都知道的事情！

```
fetch("http://127.0.0.1:9999/index.php", {

  method: 'HEAD',

  credentials: 'include',

  mode: 'no-cors',

}).

catch(err = >{

  throw err;

}).then(() = >{});
```

最后简单总结下：

1，跨域发起的options请求其实服务端的内容是返回来了的，只是浏览器这家伙不展示，具体从所抓的数据包中得证。

2，服务端即使不配合前端设置header("Access-Control-Allow-Origin: * ")，前端可以直接设置Sec-Fetch-Mode:no-cors属性来实现跨域

3，HEAD请求不返回body体的事情大家都知道，这也是为什么一些心跳检测喜欢用HEAD请求的原因，没有body体，数据也少传输点，随便一想也知道自然更高效！