

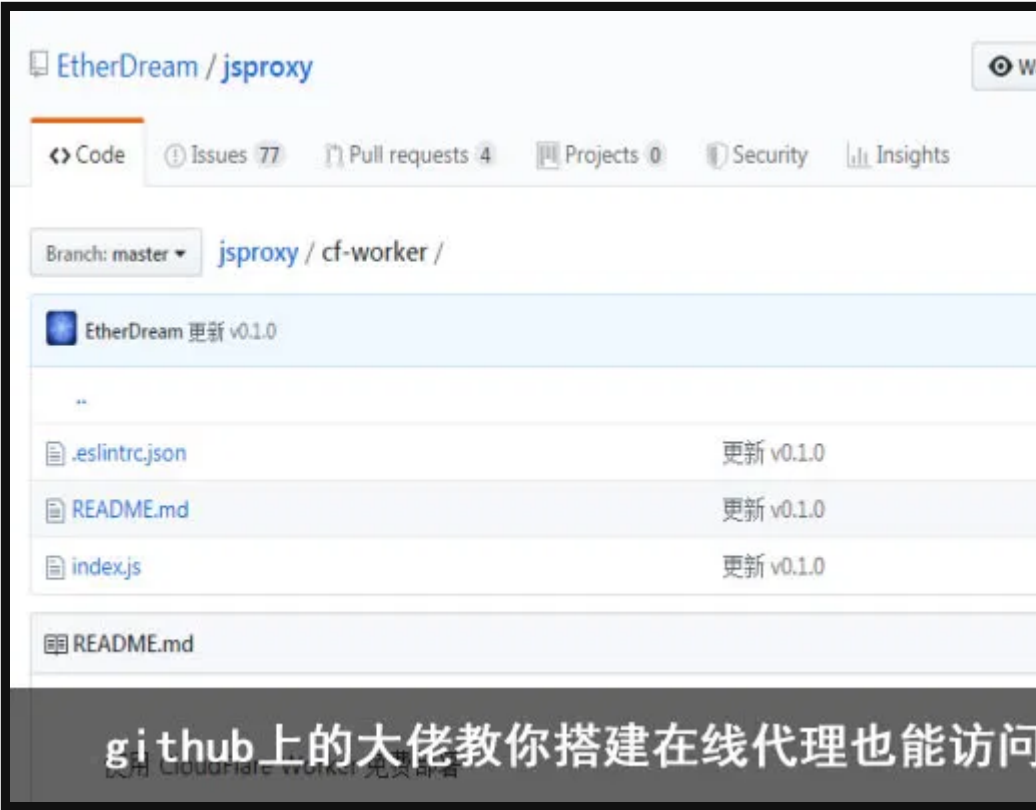
🕒 2019-09-24 11:00:50

💬 发表评论

✖

摘要

站长之前介绍过一种搭建代理方法，具体可以参见下方的前情提要，而这次介绍的是利用一个github项目来搭建的真实有效，so easy！不太懂的可以参看下面的Youtube指导视频，于是有了今天这篇文章.....



站长寄语

站长之前介绍过一种搭建代理方法，具体可以参见下方的前情提要，而这次介绍的是利用一个github项目来搭建，站长亲自测试，真实有效，so easy！不太懂的可以参看下面的Youtube指导视频，于是有了今天这篇文章.....

前情提要

- 《利用github上分享的代码搭建自己的在线代理访问谷歌分分钟》

搭建视频



项目介绍

github项目地址：<https://github.com/EtherDream/jsproxy/tree/master/cf-worker>^ψ。

CloudFlare Worker 是 CloudFlare 的边缘计算服务。开发者可通过 JavaScript 对 CDN 进行编程，无需自己的服务器参与。

实现方法

- 打开：<https://workers.cloudflare.com>，有账号登陆，没有账号的注册后再登陆；



You write code. We handle the rest.

Deploy serverless code to data centers across 194 cities in 90 countries to give it exceptional performance and reliability.

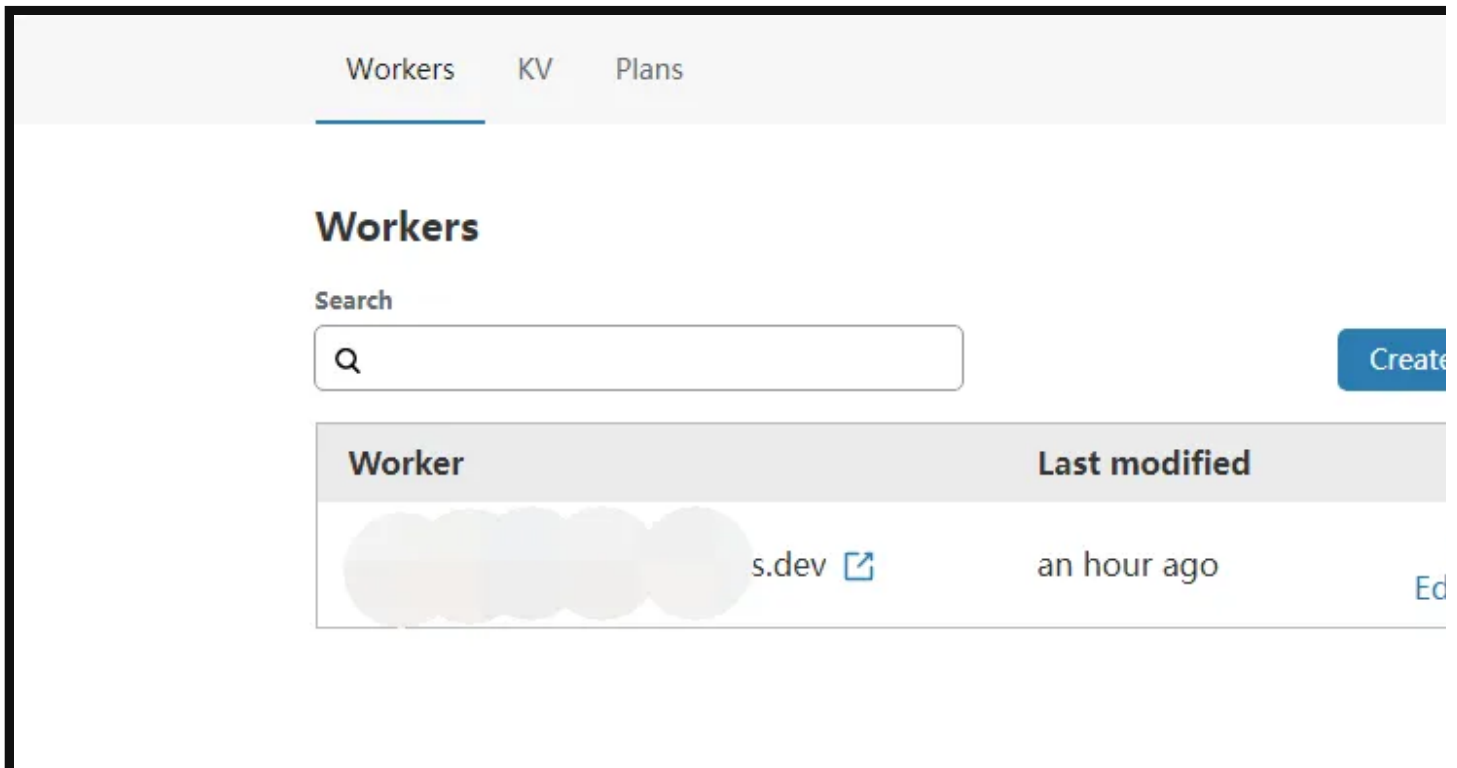
[Start building](#)[Read docs](#)

- Deploy to all of our data centers in **~15 seconds**
- Your code runs within **milliseconds** from your users worldwide
- Cold starts up to **50× faster** than other platforms

```
# Install
~/ $ npm i
~/ $ wrangle

# Create an
~/ $ wrangle
~/ $ cd hel
~/hello $ w
~/hello $ w
Published h
```

- 注册，登陆，**Start building**，取一个子域名，**Create a Worker**。



- 清空左侧代码区，复制下方代码到左侧代码框，save and display。如果正常，右侧就会显示页面

```
1. 'use strict'
```

```
2.
3.  /**
4.   * static files (404.html, sw.js, conf.js)
5.   */
6.  const ASSET_URL = 'https://etherdream.github.io/jsproxy'
7.
8.  const JS_VER = 10
9.  const MAX_RETRY = 1
10.
11.  /** @type {RequestInit} */
12.  const PREFLIGHT_INIT = {
13.    status: 204,
14.    headers: new Headers({
15.      'access-control-allow-origin': '*',
16.      'access-control-allow-methods': 'GET, POST, PUT, PATCH, TRACE, DELETE, HEAD, OPTIONS',
17.      'access-control-max-age': '1728000',
18.    }),
19.  }
20.
21.  /**
22.   * @param {any} body
23.   * @param {number} status
24.   * @param {Object<string, string>} headers
25.   */
26.  function makeRes(body, status = 200, headers = {}) {
27.    headers['--ver'] = JS_VER
28.    headers['access-control-allow-origin'] = '*'
29.    return new Response(body, {status, headers})
30.  }
31.
32.
33.  /**
34.   * @param {string} urlStr
35.   */
36.  function newUrl(urlStr) {
37.    try {
38.      return new URL(urlStr)
39.    } catch (err) {
40.      return null
41.    }
42.  }
43.
44.
45.  addEventListener('fetch', e => {
46.    const ret = fetchHandler(e)
47.    .catch(err => makeRes('cfworker error:\n' + err.stack, 502))
48.    e.respondWith(ret)
49.  })
50.
51.
52.  /**
53.   * @param {FetchEvent} e
54.   */
55.  async function fetchHandler(e) {
56.    const req = e.request
57.    const urlStr = req.url
58.    const urlObj = new URL(urlStr)
59.    const path = urlObj.href.substr(urlObj.origin.length)
```

```
60.
61.   if (urlObj.protocol === 'http:') {
62.       urlObj.protocol = 'https:'
63.       return makeRes(' ', 301, {
64.           'strict-transport-security': 'max-age=99999999; includeSubDomains; preload',
65.           'location': urlObj.href,
66.       })
67.   }
68.
69.   if (path.startsWith('/http/')) {
70.       return httpHandler(req, path.substr(6))
71.   }
72.
73.   switch (path) {
74.       case '/http':
75.           return makeRes('请更新 cfworker 到最新版本!')
76.       case '/ws':
77.           return makeRes('not support', 400)
78.       case '/works':
79.           return makeRes('it works')
80.       default:
81.           // static files
82.           return fetch(ASSET_URL + path)
83.   }
84. }
85.
86.
87. /**
88.  * @param {Request} req
89.  * @param {string} pathname
90.  */
91. function httpHandler(req, pathname) {
92.     const reqHdrRaw = req.headers
93.     if (reqHdrRaw.has('x-jsproxy')) {
94.         return Response.error()
95.     }
96.
97.     // preflight
98.     if (req.method === 'OPTIONS' &&
99.         reqHdrRaw.has('access-control-request-headers'))
100.    ) {
101.        return new Response(null, PREFLIGHT_INIT)
102.    }
103.
104.    let acehOld = false
105.    let rawSvr = ''
106.    let rawLen = ''
107.    let rawEtag = ''
108.
109.    const reqHdrNew = new Headers(reqHdrRaw)
110.    reqHdrNew.set('x-jsproxy', '1')
111.
112.    // 此处逻辑和 http-dec-req-hdr.lua 大致相同
113.    // https://github.com/EtherDream/jsproxy/blob/master/lua/http-dec-req-hdr.lua
114.    const refer = reqHdrNew.get('referer')
115.    const query = refer.substr(refer.indexOf('?') + 1)
116.    if (!query) {
117.        return makeRes('missing params', 403)
```

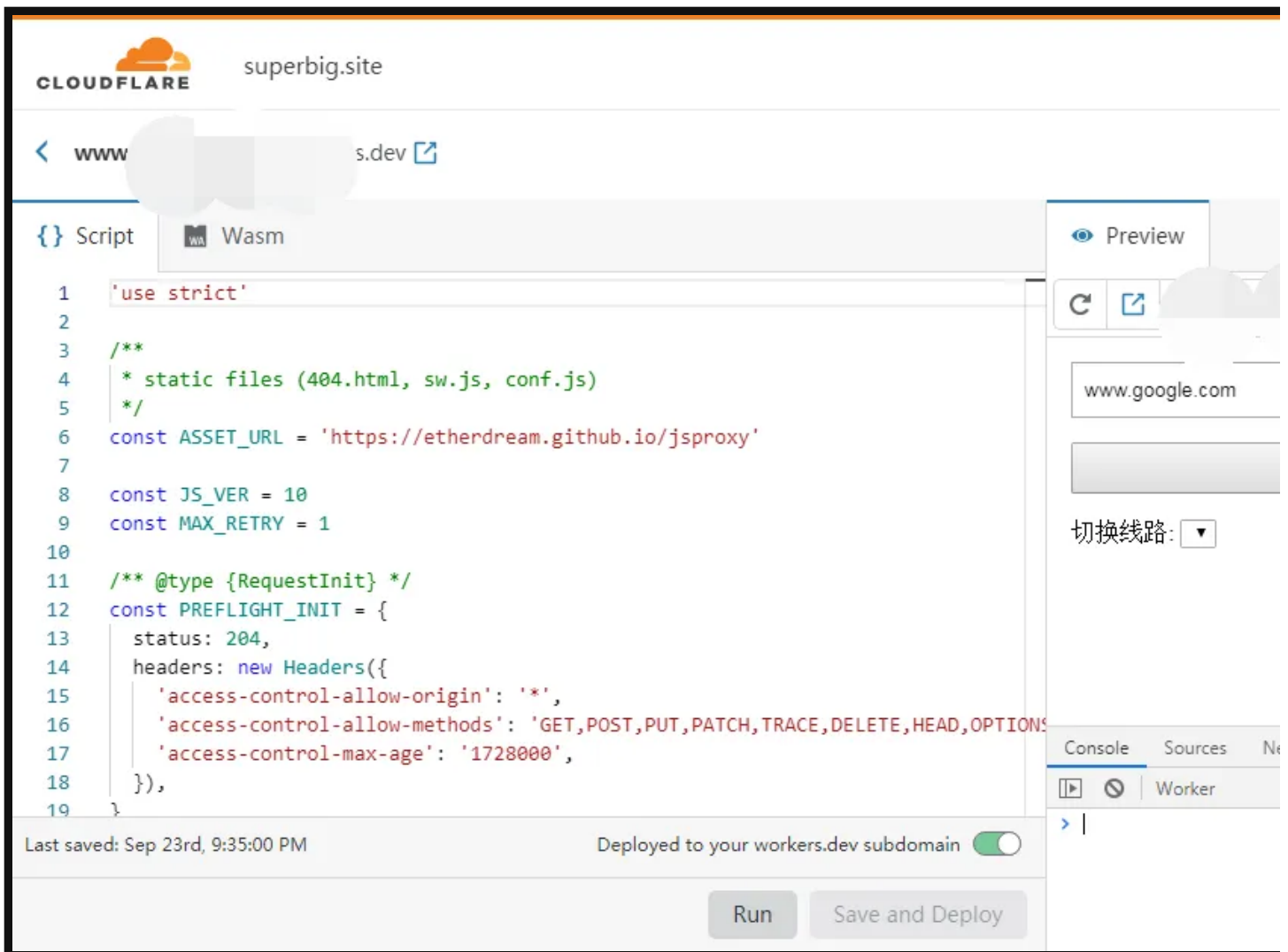
```

118.     }
119.     const param = new URLSearchParams(query)
120.
121.     for (const [k, v] of Object.entries(param)) {
122.         if (k.substr(0, 2) === '--') {
123.             // 系统信息
124.             switch (k.substr(2)) {
125.                 case 'aceh':
126.                     acehOld = true
127.                     break
128.                 case 'raw-info':
129.                     [rawSvr, rawLen, rawEtag] = v.split('|')
130.                     break
131.             }
132.         } else {
133.             // 还原 HTTP 请求头
134.             if (v) {
135.                 reqHdrNew.set(k, v)
136.             } else {
137.                 reqHdrNew.delete(k)
138.             }
139.         }
140.     }
141.     if (!param.has('referer')) {
142.         reqHdrNew.delete('referer')
143.     }
144.
145.     // cfworker 会把路径中的 `//` 合并成 `/`
146.     const urlStr = pathname.replace(/^((https?):\/+/, '$1:///')
147.     const urlObj = new URL(urlStr)
148.     if (!urlObj) {
149.         return makeRes('invalid proxy url: ' + urlStr, 403)
150.     }
151.
152.     /** @type {RequestInit} */
153.     const reqInit = {
154.         method: req.method,
155.         headers: reqHdrNew,
156.         redirect: 'manual',
157.     }
158.     if (req.method === 'POST') {
159.         reqInit.body = req.body
160.     }
161.     return proxy(urlObj, reqInit, acehOld, rawLen, 0)
162. }
163.
164.
165. /**
166.  *
167.  * @param {URL} urlObj
168.  * @param {RequestInit} reqInit
169.  * @param {number} retryTimes
170.  */
171. async function proxy(urlObj, reqInit, acehOld, rawLen, retryTimes) {
172.     const res = await fetch(urlObj.href, reqInit)
173.     const resHdrOld = res.headers
174.     const resHdrNew = new Headers(resHdrOld)
175.

```

```
176. let expose = '*'
177.
178. for (const [k, v] of resHdrOld.entries()) {
179.   if (k === 'access-control-allow-origin' ||
180.       k === 'access-control-expose-headers' ||
181.       k === 'location' ||
182.       k === 'set-cookie'
183.   ) {
184.     const x = '--' + k
185.     resHdrNew.set(x, v)
186.     if (acehOld) {
187.       expose = expose + ',' + x
188.     }
189.     resHdrNew.delete(k)
190.   }
191.   else if (acehOld &&
192.       k !== 'cache-control' &&
193.       k !== 'content-language' &&
194.       k !== 'content-type' &&
195.       k !== 'expires' &&
196.       k !== 'last-modified' &&
197.       k !== 'pragma'
198.   ) {
199.     expose = expose + ',' + k
200.   }
201. }
202.
203. if (acehOld) {
204.   expose = expose + ',--s'
205.   resHdrNew.set('--t', '1')
206. }
207.
208. // verify
209. if (rawLen) {
210.   const newLen = resHdrOld.get('content-length') || ''
211.   const badLen = (rawLen !== newLen)
212.
213.   if (badLen) {
214.     if (retryTimes < MAX_RETRY) {
215.       urlObj = await parseYtVideoRedir(urlObj, newLen, res)
216.       if (urlObj) {
217.         return proxy(urlObj, reqInit, acehOld, rawLen, retryTimes + 1)
218.       }
219.     }
220.     return makeRes(res.body, 400, {
221.       '--error': `bad len: ${newLen}, except: ${rawLen}`,
222.       'access-control-expose-headers': '--error',
223.     })
224.   }
225.
226.   if (retryTimes > 1) {
227.     resHdrNew.set('--retry', retryTimes)
228.   }
229. }
230.
231. let status = res.status
232.
233. resHdrNew.set('access-control-expose-headers', expose)
```

```
234.     resHdrNew.set('access-control-allow-origin', '*')
235.     resHdrNew.set('--s', status)
236.     resHdrNew.set('--ver', JS_VER)
237.
238.     resHdrNew.delete('content-security-policy')
239.     resHdrNew.delete('content-security-policy-report-only')
240.     resHdrNew.delete('clear-site-data')
241.
242.     if (status === 301 ||
243.         status === 302 ||
244.         status === 303 ||
245.         status === 307 ||
246.         status === 308
247.     ) {
248.         status = status + 10
249.     }
250.
251.     return new Response(res.body, {
252.         status,
253.         headers: resHdrNew,
254.     })
255. }
256.
257.
258. /**
259.  * @param {URL} urlObj
260.  */
261. function isYtUrl(urlObj) {
262.     return (
263.         urlObj.host.endsWith('.googlevideo.com') &&
264.         urlObj.pathname.startsWith('/videoplayback')
265.     )
266. }
267.
268. /**
269.  * @param {URL} urlObj
270.  * @param {number} newLen
271.  * @param {Response} res
272.  */
273. async function parseYtVideoRedir(urlObj, newLen, res) {
274.     if (newLen > 2000) {
275.         return null
276.     }
277.     if (!isYtUrl(urlObj)) {
278.         return null
279.     }
280.     try {
281.         const data = await res.text()
282.         urlObj = new URL(data)
283.     } catch (err) {
284.         return null
285.     }
286.     if (!isYtUrl(urlObj)) {
287.         return null
288.     }
289.     return urlObj
290. }
```

The screenshot shows the Cloudflare Workers dashboard for a worker named 'superbig.site'. The 'Script' tab is active, displaying a JavaScript file named 'Wasm'. The script is a proxy that intercepts requests and forwards them to 'https://etherdream.github.io/jsproxy'. It includes a preflight request to set CORS headers. The interface includes a 'Preview' button, a 'Console' tab, and a 'Worker' tab. The 'Console' tab shows the output of the script, and the 'Worker' tab shows the worker's status. The 'Preview' button is highlighted, and the 'Console' tab is selected. The 'Worker' tab shows the worker's status as 'Running'.

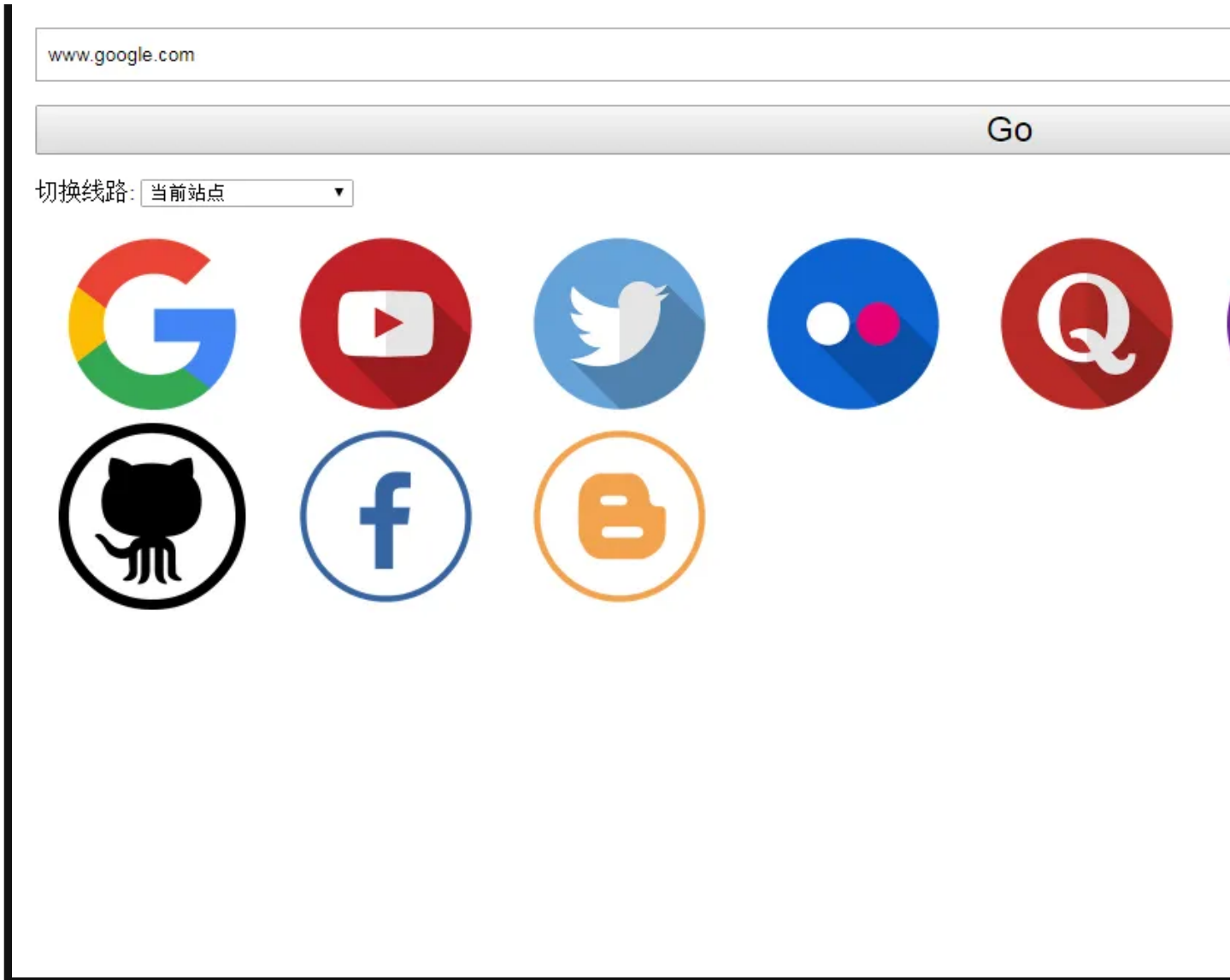
```
1 'use strict'
2
3 /**
4  * static files (404.html, sw.js, conf.js)
5  */
6 const ASSET_URL = 'https://etherdream.github.io/jsproxy'
7
8 const JS_VER = 10
9 const MAX_RETRY = 1
10
11 /** @type {RequestInit} */
12 const PREFLIGHT_INIT = {
13   status: 204,
14   headers: new Headers({
15     'access-control-allow-origin': '*',
16     'access-control-allow-methods': 'GET,POST,PUT,PATCH,TRACE,DELETE,HEAD,OPTIONS',
17     'access-control-max-age': '1728000',
18   }),
19 }
```

Last saved: Sep 23rd, 9:35:00 PM

Deployed to your workers.dev subdomain ☒

Run Save and Deploy

- 之后把右侧的网址复制浏览器，收藏下来，以后就可以使用了。免费版每天有10万个调用限制，



温馨提示

如果客观喜欢看该篇文章，那么下面的文章，或多或少你会喜欢，推荐浏览一下：

- 《 免费用runkit搭建一个谷歌镜像 》
- 《 Psiphon当然也能访问谷歌等知名网站，就是速度一般 》
- 《 简单几步，优雅地用上谷歌搜索 》
- 《 发现一个网站，可以轻松访问谷歌等知名网站 》
- 《 利用hosts+在线代理，轻松访问谷歌等知名网站 》

更多精彩内容，可以点击页面右上方的搜索框，搜索关键词，探索未知的世界，祝你好运！

相关文章

- **电脑端如何利用heroku免费搭建访问谷歌等知名的proxy方法，每月750小时**

站长之间推荐了一种利用heroku的特点以ss的形式可以访问谷歌等知名网址，具体可以参看实例，每月750小时》，反响还不错，这次又有一个方法可以达到访问谷歌等知名网站的目的，：

■ 电脑端如何利用heroku免费搭建访问谷歌等知名的另一方法，每月750小时

站长之间推荐了一种利用heroku的特点以ss的形式可以访问谷歌等知名网址，具体可以参看实例，每月750小时》，反响还不错，这次又有一个方法可以达到访问谷歌等知名网站的目的，：

■ 利用github上分享的代码搭建自己的谷歌镜像分分钟

站长之前在这篇文章《简单几步，优雅地用上谷歌搜索》，介绍的网址就是利用该项目来搭建站长亲自测试，真实有效，so easy！于是有了今天这篇文章.....

■ github上发现一款名为aproxy的项目可以访问谷歌等知名网站

今天逛github不仅发现了gridea博客制作客户端，还发现一款名为aproxy的应用可以访问github，胡须有更多好用的项目被发现，被世人所了解，于是就有了今天这篇文章.....

■ 如何用订阅实现科学的访问国际互联网比如谷歌之类的

之前介绍的各种能访问国际互联网的应用想必大家都会用了，这次咱们来点进阶的，利用订阅呢，于是就有了今天这篇文章.....

分享至：



我的微信

微信扫一扫加好友

