利用CloudFlare Worker 免费部署 JSProxy 服务

2019-11-22

JSProxy 一个基于浏览器端 JS 实现的在线代理，这里不多介绍！

本文主要介绍一下利用 `CloudFlare Worker` 来搭建一个JSProxy服务。

`CloudFlare Worker` 是 CloudFlare 的边缘计算服务。开发者可通过 JavaScript 对 CDN 进行编程，从而能灵活处理 HTTP 请求。这使得很多任务可在 CDN 上完成，无需自己的服务器参与。

CFW免费服务，支持每天10 万次免费请求！基本也够用了！

项目介绍

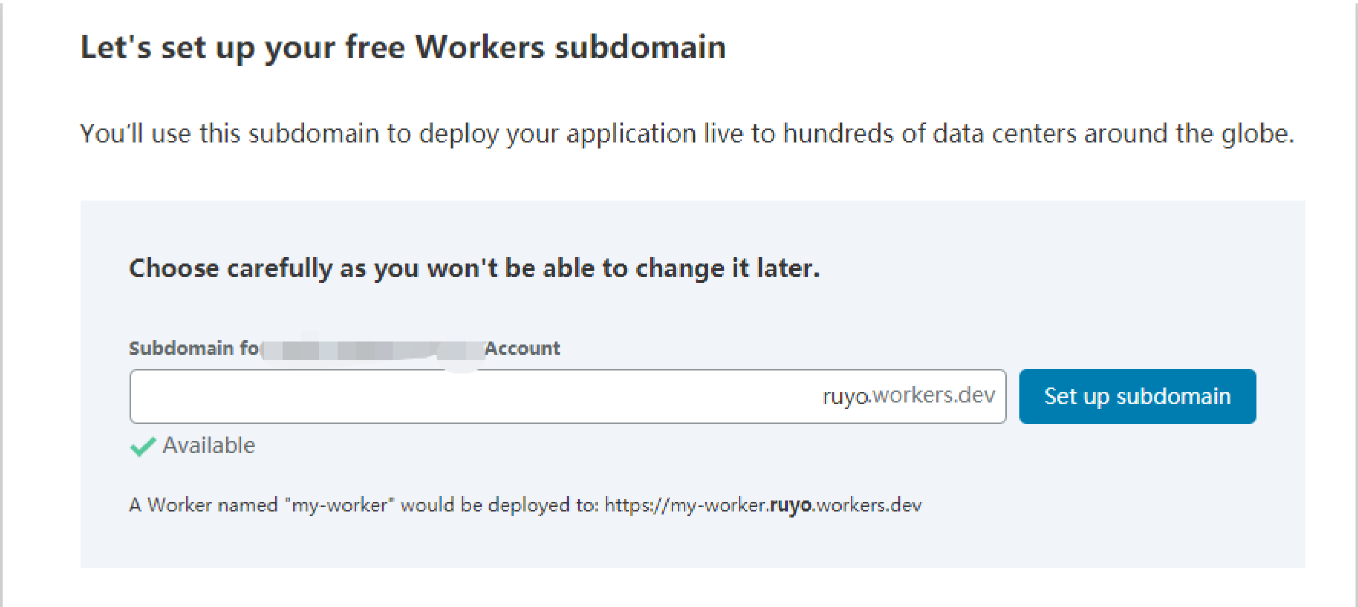
项目地址：https://github.com/EtherDream/jsproxy

准备工作

Cloudflare 账号一个

使用教程

1）打开 https://workers.cloudflare.com，登陆上你的 Cloudflare 账号激活 Workers 服务


Let's set up your free Workers subdomain

You'll use this subdomain to deploy your application live to hundreds of data centers around the globe.

Choose carefully as you won't be able to change it later.

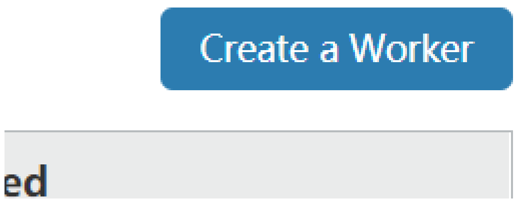
Subdomain fo[redacted]Account

ruyo.workers.dev

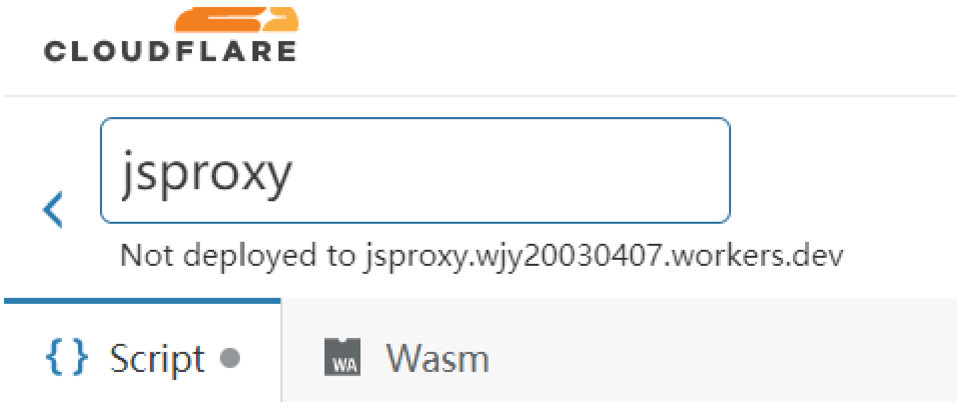
✓ Available

A Worker named "my-worker" would be deployed to: https://my-worker.ruyo.workers.dev

然后创建一个 Workers【Create a Worker】



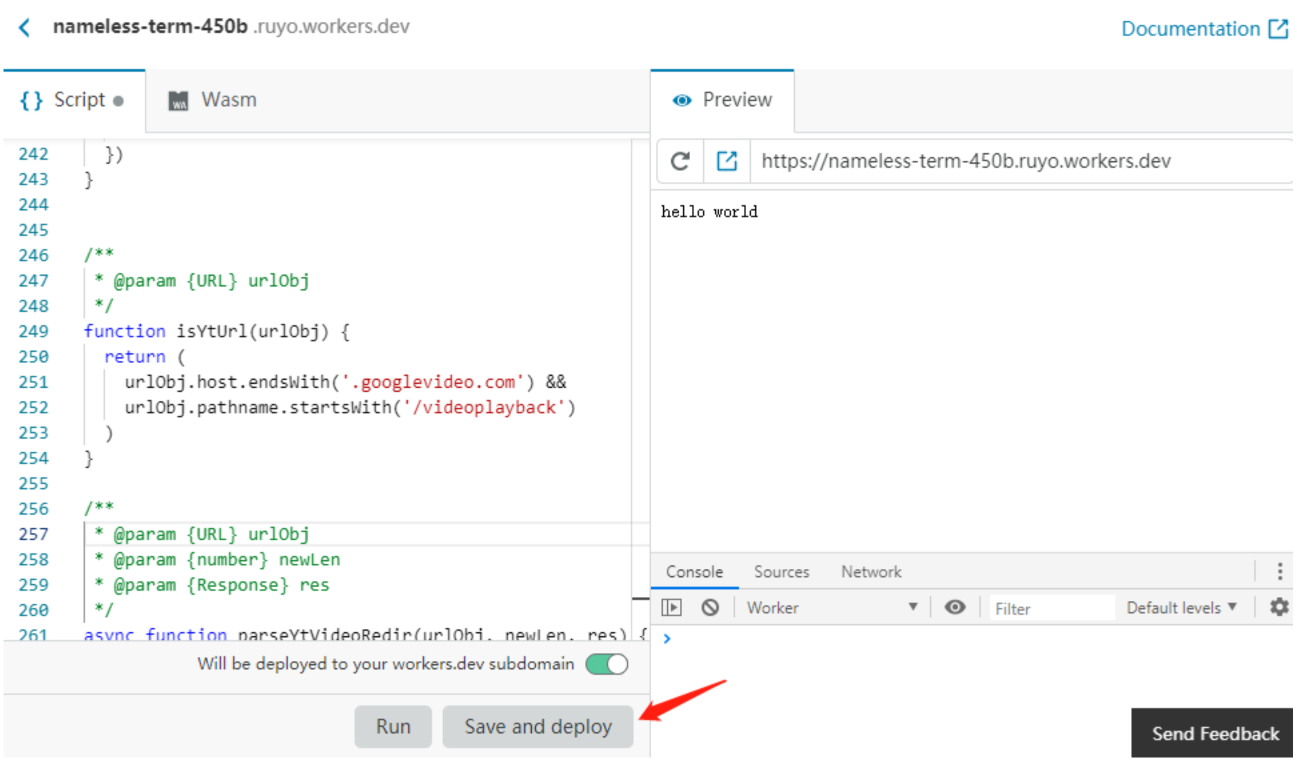
2）修改一下子域名，创建出来的域名格式 自定义的内容.Cloudflare用户名.workers.dev



Not deployed to jsproxy.wjy20030407.workers.dev

3）复制 https://raw.githubusercontent.com/EtherDream/jsproxy/master/cf-worker/index.js 的内容到左侧代码（Script）区域

文章最下方有代码备份！！


nameless-term-450b.ruyo.workers.dev
Documentation
{ } Script ●
Wasm
Preview
https://nameless-term-450b.ruyo.workers.dev
Console
Sources
Network
Worker
Filter
Default levels
Will be deployed to your workers.dev subdomain
Run
Save and deploy
Send Feedback

```
  })
}


/**
 * @param {URL} urlObj
 */
function isYtUrl(urlObj) {
  return (
    urlObj.host.endsWith('.googlevideo.com') &&
    urlObj.pathname.startsWith('/videoplayback')
  )
}

/**
 * @param {URL} urlObj
 * @param {number} newLen
 * @param {Response} res
 */
async function parseYtVideoRedir(urlObj, newLen, res) {
```

```
hello world
```

4）先点击【Run】右侧看执行效果，再点击 【Save and deploy】 部署代码

5）届时你可以访问你的站点https://xxx.子域名.workers.dev查看效果

进入站点后将线路选择切换为当前站点即可使用

特别提示：浏览网站的时候，有时候会提示加载不安全脚本，点击允许即可！

代码备份

```
'use strict'

/**
 * static files (404.html, sw.js, conf.js)
 */
const ASSET_URL = 'https://etherdream.github.io/jsproxy'

const JS_VER = 10
const MAX_RETRY = 1

/** @type {RequestInit} */
const PREFLIGHT_INIT = {
  status: 204,
  headers: new Headers({
    'access-control-allow-origin': '*',
    'access-control-allow-methods': 'GET,POST,PUT,PATCH,TRACE,DEL
    'access-control-max-age': '1728000',
  }),
}

/**
 * @param {any} body
 * @param {number} status
 * @param {Object<string, string>} headers
 */
function makeRes(body, status = 200, headers = {}) {
  headers['--ver'] = JS_VER
  headers['access-control-allow-origin'] = '*'
  return new Response(body, {status, headers})
}


/**
 * @param {string} urlStr
 */
function newUrl(urlStr) {
  try {
    return new URL(urlStr)
  } catch (err) {
    return null
  }
}


addEventListener('fetch', e => {
  const ret = fetchHandler(e)
    .catch(err => makeRes('cfworker error:\n' + err.stack, 502))
  e.respondWith(ret)
})
```

```
/**
 * @param {FetchEvent} e
 */
async function fetchHandler(e) {
  const req = e.request
  const urlStr = req.url
  const urlObj = new URL(urlStr)
  const path = urlObj.href.substr(urlObj.origin.length)

  if (urlObj.protocol === 'http:') {
    urlObj.protocol = 'https:'
    return makeRes('', 301, {
      'strict-transport-security': 'max-age=99999999; includeSubD
      'location': urlObj.href,
    })
  }

  if (path.startsWith('/http/')) {
    return httpHandler(req, path.substr(6))
  }

  switch (path) {
  case '/http':
    return makeRes('请更新 cfworker 到最新版本!')
  case '/ws':
    return makeRes('not support', 400)
  case '/works':
    return makeRes('it works')
  default:
    // static files
    return fetch(ASSET_URL + path)
  }
}


/**
 * @param {Request} req
 * @param {string} pathname
 */
function httpHandler(req, pathname) {
  const reqHdrRaw = req.headers
  if (reqHdrRaw.has('x-jsproxy')) {
    return Response.error()
  }

  // preflight
  if (req.method === 'OPTIONS' &&
      reqHdrRaw.has('access-control-request-headers')
  ) {
    return new Response(null, PREFLIGHT_INIT)
  }

  let acehOld = false
  let rawSvr = ''
  let rawLen = ''
  let rawEtag = ''

  const reqHdrNew = new Headers(reqHdrRaw)
  reqHdrNew.set('x-jsproxy', '1')

  // 此处逻辑和 http-dec-req-hdr.lua 大致相同
  // https://github.com/EtherDream/jsproxy/blob/master/lua/http-d
  const refer = reqHdrNew.get('referer')
  const query = refer.substr(refer.indexOf('?') + 1)
  if (!query) {
    return makeRes('missing params', 403)
  }
  const param = new URLSearchParams(query)

  for (const [k, v] of Object.entries(param)) {
    if (k.substr(0, 2) === '--') {
      // 系统信息
      switch (k.substr(2)) {
      case 'aceh':
        acehOld = true
        break
      case 'raw-info':
        [rawSvr, rawLen, rawEtag] = v.split('|')
        break
      }
    } else {
      // 还原 HTTP 请求头
      if (v) {
        reqHdrNew.set(k, v)
      } else {
        reqHdrNew.delete(k)
      }
    }
  }
  if (!param.has('referer')) {
    reqHdrNew.delete('referer')
  }

  // cfworker 会把路径中的 `//` 合并成 `/`
  const urlStr = pathname.replace(/^(https?):\/+/, '$1://')
  const urlObj = newUrl(urlStr)
  if (!urlObj) {
    return makeRes('invalid proxy url: ' + urlStr, 403)
  }

  /** @type {RequestInit} */
  const reqInit = {
    method: req.method,
    headers: reqHdrNew,
    redirect: 'manual',
  }
  if (req.method === 'POST') {
    reqInit.body = req.body
  }
  return proxy(urlObj, reqInit, acehOld, rawLen, 0)
}


/**
```

```
 *
 * @param {URL} urlObj
 * @param {RequestInit} reqInit
 * @param {number} retryTimes
 */
async function proxy(urlObj, reqInit, acehOld, rawLen, retryTimes
  const res = await fetch(urlObj.href, reqInit)
  const resHdrOld = res.headers
  const resHdrNew = new Headers(resHdrOld)

  let expose = '*'

  for (const [k, v] of resHdrOld.entries()) {
    if (k === 'access-control-allow-origin' ||
        k === 'access-control-expose-headers' ||
        k === 'location' ||
        k === 'set-cookie'
    ) {
      const x = '--' + k
      resHdrNew.set(x, v)
      if (acehOld) {
        expose = expose + ',' + x
      }
      resHdrNew.delete(k)
    }
    else if (acehOld &&
      k !== 'cache-control' &&
      k !== 'content-language' &&
      k !== 'content-type' &&
      k !== 'expires' &&
      k !== 'last-modified' &&
      k !== 'pragma'
    ) {
      expose = expose + ',' + k
    }
  }

  if (acehOld) {
    expose = expose + ',--s'
    resHdrNew.set('--t', '1')
  }

  // verify
  if (rawLen) {
    const newLen = resHdrOld.get('content-length') || ''
    const badLen = (rawLen !== newLen)

    if (badLen) {
      if (retryTimes < MAX_RETRY) {
        urlObj = await parseYtVideoRedir(urlObj, newLen, res)
        if (urlObj) {
          return proxy(urlObj, reqInit, acehOld, rawLen, retryTim
        }
      }
      return makeRes(res.body, 400, {
        '--error': `bad len: ${newLen}, except: ${rawLen}`,
        'access-control-expose-headers': '--error',
      })
    }

    if (retryTimes > 1) {
      resHdrNew.set('--retry', retryTimes)
    }
  }

  let status = res.status

  resHdrNew.set('access-control-expose-headers', expose)
  resHdrNew.set('access-control-allow-origin', '*')
  resHdrNew.set('--s', status)
  resHdrNew.set('--ver', JS_VER)

  resHdrNew.delete('content-security-policy')
  resHdrNew.delete('content-security-policy-report-only')
  resHdrNew.delete('clear-site-data')
  if (status === 301 ||
      status === 302 ||
      status === 303 ||
      status === 307 ||
      status === 308
  ) {
    status = status + 10
  }

  return new Response(res.body, {
    status,
    headers: resHdrNew,
  })
}


/**
 * @param {URL} urlObj
 */
function isYtUrl(urlObj) {
  return (
    urlObj.host.endsWith('.googlevideo.com') &&
    urlObj.pathname.startsWith('/videoplayback')
  )
}

/**
 * @param {URL} urlObj
 * @param {number} newLen
 * @param {Response} res
 */
async function parseYtVideoRedir(urlObj, newLen, res) {
  if (newLen > 2000) {
    return null
  }
  if (!isYtUrl(urlObj)) {
    return null
  }
  try {
    const data = await res.text()
    urlObj = new URL(data)
  } catch (err) {
    return null
  }
  if (!isYtUrl(urlObj)) {
    return null
  }
  return urlObj
}
```

PyOne一键安装脚本 for CentOS 7/Debian 8+/Ubuntu 16+
注册激活Navicat全系产品的开源代码分享

评论基础模式加载失败，是否 重载 或 尝试完整 Disqus 模式？