Generated on 2025-05-28

ParaN3xus's Blog 2023

Archive of Blog posts in 2023

目录

2023-11-28

利用 Cloudflare Worker 实现的 Telegram Bot API 代理

如何使用 Cloudflare Worker 搭建 Telegram Bot API 代理, 解决国内无法直接访问的问题

本代码已修复 “Bad Request: message text is empty” 异常.

1.1 缘由

想用服务器搭一个青龙面板薅羊毛, 但是服务器在国内, Telegram 消息通知没法走官方 API, 所以只能自建代理.

但是从网上的很多教程里面抄的代码在发送消息的时候都出现的上述异常, 似乎是因为 Post 请求的 data 没有一并转发(?), 最奇怪的是这些教程还都很新, 距今都不到一年.

最后也没找到不存在这个异常的代码, 所以决定自己写一个, 顺便理一下整个过程写一篇教程.

1.2 准备工作

- Cloudflare 账号
- 自己的域名

1.3 教程

1.3.1 创建和配置 Cloudflare Worker

登录 Cloudflare 后, 面板左侧的侧边栏中可以找到 “Workers & Pages”.

进入后, 点击右侧的 “Create application”, 然后点击 “Create Worker”.

Worker 的名字可以随便填写, 然后点击 “Deploy”.

点击 “Edit code” 进入编辑器, 全选粘贴以下代码

```
javascript
var URL_PATH_REGEX = /^\/bot(?<bot_token>[^/]+)\/(?<api_method>[a-z]+)/i;

var src_default = {
  async fetch(request) {
    const { pathname: path, search } = new URL(request.url);

    let matchResult;
    let apiUrl;

    try {
      matchResult = path.match(URL_PATH_REGEX);
    } catch (e) {
      return new Response("Invalid URL", {
        status: 400,
        headers: { "content-type": "text/plain" }
      });
    }
```

```
    if (matchResult && matchResult.groups) {
      const { bot_token, api_method } = matchResult.groups;
      apiUrl = "https://api.telegram.org/bot" + bot_token + "/" + api_method +
search;
    } else {
      return new Response("Invalid URL", {
        status: 400,
        headers: { "content-type": "text/plain" }
      });
    }

    if (request.method === "GET") {
      const response = await fetch(apiUrl, {
        method: "GET",
        headers: request.headers
      });
      const responseBody = await response.text();

      return new Response(responseBody, {
        status: response.status,
        statusText: response.statusText,
        headers: response.headers
      });
    } else if (request.method === "POST") {
      const response = await fetch(apiUrl, {
        method: request.method,
        headers: request.headers,
        body: request.body
      });
      const responseBody = await response.text();

      return new Response(responseBody, {
        status: response.status,
        statusText: response.statusText,
        headers: response.headers
      });
    } else {
      return new Response("Unsupported Request Method", {
        status: 400,
        headers: { "content-type": "text/plain" }
      });
    }
  }
};
export {
  src_default as default
};
```

保存后点击右上角 “Save and deploy” 并确认.

1.3.2 添加 DNS 记录

由于 Cloudflare Workers 的默认域名已经被墙, 所以我们需要使用自己的域名, 并通过 Workers Route 调用刚刚创建的 Worker.

回到面板首页, 点击左侧侧边栏的 “Websites”.

如果你的域名已经在使用 Cloudflare, 那么直接点击你的域名, 否则先使用“Add a site”把域名添加到 Cloudflare 再进入.

点击左侧侧边栏 “DNS”, 点击 “Add record”, Type 选择 “A”,Name 可填写 “tgproxy”, IPv4 地址可以随意填写, 但是必须合法, Proxy status 要开启.

点击“Save”保存记录.

1.3.3 创建 Workers Route

点击左侧侧边栏的 “Worker Routes”.

点击右侧的 “Add route”, 其中“Route”填写 `DNS 记录中的 Name.你的域名/*` ,“Worker” 选择你刚刚创建的 Worker.

1.3.4 测试

可以使用以下的代码简单测试

```
import requests
import json

TG_BOT_TOKEN = ''
TG_USER_ID = ''
TG_API_HOST = 'DNS记录中的Name.你的域名'

try:
    headers = {'Content-Type': 'application/json'}
    data = {'chat_id': TG_USER_ID, 'text': 'test'}
    json_data = json.dumps(data)

    response = requests.post(
        'https://' + TG_API_HOST + '/bot' + TG_BOT_TOKEN + '/sendMessage',
        headers=headers,
        data=json_data
    )
    print(response.text)
except Exception as e:
    print(e)
```

1.4 故障排除

如果运行测试中的代码后没有收到消息, 可以执行以下流程:

1.4.1 检查参数填写有无错误

将原代码 `TG_API_HOST` 行及其以下内容替换为:

```
TG_API_HOST = 'api.telegram.org'

proxies = {
    "http": "http://127.0.0.1:7890",
    "https": "http://127.0.0.1:7890"
}

try:
    headers = {'Content-Type': 'application/json'}
    data = {'chat_id': TG_USER_ID, 'text': 'test'}
```

```
    json_data = json.dumps(data)

    response = requests.post(
        'http://' + TG_API_HOST + '/bot' + TG_BOT_TOKEN + '/sendMessage',
        headers=headers,
        data=json_data
    , proxies=proxies)
    print(response.text)
except Exception as e:
    print(e)
```

其中 `proxies` 的内容要更改为本机可用的代理.

若本测试不通过, 则参数出错.

1.4.2 检查 Worker Route 有无错误

将上一步故障排除中的代码的 `TG_API_HOST` 更改为使用的 Worker 的页面的 Preview 后的网址

若本测试仍不通过, 则 DNS 记录或 Worker Route 配置出错.

1.4.3 检查 Cloudflare Worker 有无错误

若上述两项检查都通过, 则为 Cloudflare Worker 配置出错.

1.5 总结

`index.js` 代码已经开源在 Github.

虽然但是, 我不是很懂 JavaScript, 所以这个代码写得比较简陋(比起网上我之前找到的其他实现), 不过总算是能正常运行了.

我还是很疑惑为什么这么蠢的问题找不到现存的解决方案, 这篇 Blog 也算是做了一点小小的贡献吧.

2023-10-28

一行命令提升 WSL 使用体验

在 WSL 中创建一个 alias, 直接使用 Windows 文件资源管理器打开目录或文件.

简而言之, 运行:

```
echo "alias open=\"explorer.exe\"" >> ~/.bashrc && source ~/.bashrc
```

现在你可以:

- 运行 `open path` 使用 Windows 文件资源管理器直接打开某个目录(可使用 `.`, `~` 等)
- 运行 `open filename` 使用 Windows 默认打开方式(若未指定则要求你选择)打开某个文件. (比如我想用 Typora 编辑 WSL 里面的 markdown 文件)

我感觉很爽, 但是似乎没有看到太多人这么做, 所以分享出来(虽然没什么技术含量).

2023-10-24

giffgaff 申请踩坑

申请 giffgaff SIM 卡的经验分享

今年早些时候在群友那里了解到 giffgaff, 本着“反正不花钱, 不要白不要”的原则一口气申请了三张, 结果一张都没到.

后来本人因各种原因暂时到了南方某沿海省, 想着可能经济发达的地方可能平邮会更容易到, 再加上沿海可能辗转更少, 所以又申请了两张, 这次全到了.

但是过程并不是一帆风顺, 还是有一些网上其他教程没有提到的小坑, 给了我水这篇 Blog 的机会.

3.1 giffgaff 是什么?

这里引用另一篇教程的介绍, 因为我也不太懂.

> Giffgaff 是一家总部位于英国的行动电话公司. 作为一家行动虚拟网路电信业者, Giffgaff 使用 O2 的网路, 是 O2 的全资公司, 成立于 2009 年 11 月 25 日. Giffgaff 与传统的行动电话电信业者不同, 区别在于其使用者也可以参与公司的部份运营, 如销售, 客服, 行销环节等. 作为参与运营的回报, 使用者从公司的“回赠”(Payback)系统中获得补偿.

从结果来看, 你得到的是可在中国激活, 零月租保号的一张实体英国电话卡.

3.2 成果展示

图 1　最终收到的 giffgaff SIM 卡

3.3 申请过程

过程非常简单和人性化, 只要在 giffgaff 官网点击 Order your free SIM (也可以直接: 传送门), 然后跟着引导走就行了.

注意进官网不要挂英国梯子, 不然没有选择送货国家的位置.

既然是要零月租保号, 那当然是什么套餐都不选, 这个应该都能懂.

下面重点介绍最关键的填写地址, 这直接决定你能不能收到卡.

3.3.1 坑点 1: 地址填写

在我收到 giffgaff 后, 我发现信息栏只有 9 行字, 大体布局如下:

```
1 名 (First name) 姓 (Last name)
2
3 地址行1和2共用这四行
4
5
6 邮编 (Postcode/Zip) 市 (Town/City)
7 省 (County/Province/State (optional))
8 THE PEOPLES REPUBLIC OF CHINA(Country)
9 一传类似于某种ID的东西
```

名和姓没必要写真名. 省市邮编这些都好说, 地址行一二才是关键.

地址行一二只能填英文字母, 数字, 空格, 符号. 本人实测, 一行字最多 26 个字符, 而且你还要考虑为了保持单词不断行做的提前换行.

那如果我写太多了会怎么样呢? 答案是: 你的地址会只剩下首尾, 中间的连省略号都没有. 我猜测这就是我第一次申请没有成功送到的原因.

我建议的是, 地址行 2 只填你的电话号, 因为他是先把两行直接拼接再打印的, 所以第二行的内容一定会印上去. 这个电话号等于是一个保底的内容, 如果邮局的人有责任心的话肯定会注意到然后打给你, 到时候你就可以再完善信息了.

至于地址行 1, 就不要填写已经存在的国家, 省, 市等等信息, 只需要填市再往下的内容即可. 因为这个信息只是对市级的邮局有用, 而他们一般对下一级的区划比较熟悉, 所以像是“区”, “小区”这种就可以不要填了, 尽量保证信息能够完整显示.

此外, 本着“反正不花钱, 不要白不要”的原则, 结合目前国内平邮丢件率极高的现状, 我建议一次申请两张起步. 毕竟 giffgaff 给我的邮件里也说:

> Can’t find your SIM? No worries. Simply head to giffgaff.com to order a new SIM today and we’ll pop another one in the post.

3.4 寄送过程

从官网申请到拿到实物一共花了 38 天, 其中从市级到达我手里就花了一星期左右. 平邮是这样的, 慢一点很正常, 所以不要急, 反正你也没花钱.

3.5 激活过程

激活过程也比较人性化, 跟着引导来就可以了, 但是要注意我们的目标是零月费保号, 所以:

3.5.1 坑点 2: 选套餐

一定要用网页版激活! 选套餐的时候一定要划到最下面选 pay as you go (除非你很有钱, 能付得起一个月 50CNY).

如果你不慎真选错了, 那也没关系, 只要关掉自动续费下一个月就会变成无套餐, 然后就等同于 pay as you go 了.

充钱的话需要用外币卡, 我这边实测 dupay 的预付卡可以用, 所以应该是没有太多限制的.

似乎 Paypal 也可以充值, 但是我没试过, 所以不清楚是否有限制.

3.6 实际使用

待更新, 因为我就是那个充了五十块钱的冤种, 现在没钱冲话费了, 不敢用怕给我停机了.

保号的话只需要 180 天余额变动一次就行了, 简直无门槛.

2023-09-16

C++ 标识符中的 Unicode 字符

C++ 中标识符的命名规则, 以及 Unicode 字符在其中的应用.

这其实是我的课程作业, 但是因为确实解决了我的一大疑惑和内容比较充实, 所以发到这里.

在 C++ References 的 Language 栏目下的 Identifiers 栏目中, 有关于 C++ 中标识符命名规则的说明[1]:

- 首字符应为:
  - 大小写字母
  - 下划线
  - 带有 `XID_Start` 属性的任意 Unicode 字符
- 其他字符应为:
  - 数字
  - 大小写字母
  - 下划线
  - 带有 `XID_Continue` 属性的任意 Unicode 字符

原网页在两个规则的最后一条末尾给出了一个链接, 指向 2023 年 9 月 1 日发布的 Unicode 15.1.0 标准的第 31 号附件: UNICODE IDENTIFIERS AND SYNTAX. 这个附件的表 2[2]指出了上述规则中提到的 `XID_` 属性的说明和覆盖范围:

- `XID_Start`: 根据 NFKC 修改[3]从 `ID_Start` 派生
- `XID_Continue`: 根据 NFKC 修改从 `ID_Continue` 派生

这里 `XID_` 中的 `X` 大概表示 `Extended`, 实际意义就是其从 `ID_` 中经过 `NFKC`(Normalization Form Compatibility Composition)派生. 在表格中, 同样给出了 `ID_` 属性的说明和覆盖范围:

- `ID_Start`
  - 大小写字母
  - titlecase letters (似乎与希腊语有关, 这里因为和“大写字母”翻译重合不再翻译)
  - modifier letters (同上, 但似乎不是希腊语)
  - other letters (同上)
  - letter numbers (同上, 类似于各种语言的罗马数字)
  - 除 `Pattern_Syntax` 和 `Pattern_White_Spacex` 的 `Other_ID_Start`
- `ID_Continue`
  - 符合 `ID_Start` 属性的字符
  - 非间距标记
  - 间距组合标记
  - 数字
  - 连接符标点
  - 除 `Pattern_Syntax` 和 `Pattern_White_Space`[4]的 `Other_ID_Continue`

[1] Identifiers - cppreference.com
[2] UAX =31: Unicode Identifiers and Syntax
[3] UAX =31: Unicode Identifiers and Syntax
[4] UAX =31: Unicode Identifiers and Syntax

其中仍然存在的一些不明属性因过于无聊和没有必要不再展开.

其中, 平常提到的希腊字母在小写字母的范畴内, 大部分中文汉字和其他国家的文字在 other letter 范畴内. 因此实际上你可以用中文、希腊字母做标识符名字的一部分.

但是, 在实际测试中, gcc 10.2.1 能够正确识别由 Emoji 表情符号组成的标识符, 而 Emoji 表情所属的 other symbol 并不在上述属性的范围内. 说明编译器开发者对待标准的态度也不是那么一板一眼.

需要注意的是, 以上所说的内容只适用于 C++11 及之后的标准. 实际上, C++ 的 Unicode 字符支持首次出现是在 N3337 草案[5]中, 这是 C++11 发布后的第一个标准草案, 对 C++11 标准做了一些重要的修改, 应用于 C++11 标准.

[5] 14882: Contents (timsong-cpp.github.io)

2023-09-10

GitHub 学生包申请踩坑

申请 GitHub 学生包的经验分享

心血来潮(早有预谋)想要申请 GitHub 学生包, 跟随网上已有的教程走了一遭发现还是有一些没法解决的问题, 经过一些摸索后决定写这篇踩坑.

5.1 要求 completed your GitHub user profile

5.1.1 问题

拒绝原因表现为:

> You are unlikely to be verified until you have completed your GitHub user profile with your full name exactly as it appears in your academic affiliation document plus a short bio. Please do not use a variation of your name or a nickname. Once you have updated your profile information log out and log back into GitHub before re-applying.
>
> Have you completed your GitHub user profile with all your relevant information, such as your full name as it appears in your image and a short bio?

即使你按照其要求, 将自己的用户名改为学生卡的信息也不能成功.

5.1.2 解决方案

在 community 中找到了相关的问题, 并找到了用户 snowball-rain 分享的解决方法, 原链接: [‘The GitHub Student Developer Pack’ was not verified in spite of my student ID is valid · community · Discussion =53463](https://github.com/orgs/community/discussions/53463)

具体的方法如下:

1. 修改 Github user profile
   - Name (注意不是登录使用的 Username) 改为学生卡(其它证明材料)的名字, 使用英文(拼音).
   - Bio 改为 xxx of XXX University, 这里要填写自己和自己学校名字的英文.
2. 让需要的信息的英文出现在学生卡(其它证明材料)的照片上: 拍下来后自己在图片上加上翻译
   - 学校名
   - Name, 需要和 profile 填写的一致
   - Valid Date, 比如 vaild until 09/2023, 如果中文版没有这个信息, 那么需要把发卡日期和学制等信息一并翻译(一般都有吧).
3. 不要上传图片, 要拍. 电脑打开图片, 然后手机对着电脑上修改过的图片拍.
4. 等, 我等了好多天.

2023-09-03

C++ 学习笔记: 输入输出

C++ 的输入输出方法和重定向到文件的技巧

6.1 输入

6.1.1 `scanf()`

可读取固定格式的输入.

用法:

```
scanf(format, contents...)
```

其中, `format` 可填写的占位符有:

- `%d`: `int` (十进制)
- `%u`: `uint` (十进制)
- `%o`: `int` (八进制)
- `%x`/`%X`: `int` (十六进制)
- `%i`: `int` (十六、十、八进制)
- `%f`: `float`/`double`
- `%e`: `float`/`double` (以科学计数法形式表示)
- `%g`: `float`/`double` (科学计数法和正常形式表示皆可)
- `%s`: `string`
- `%c`: `char`
- `%p`: `void*`

6.1.2 `getchar()`

可以读取一个字符, 返回值即为读取的一个字符, 当输入结束时, 会返回一个特殊的常量 `EOF`(End of File).

6.1.3 `fgets()`

可读取一行.

用法:

```
fgets(chararr, sizeof(chararr), stream)
```

当使用控制台输入时, `stream` 可填写 `stdin`.

6.1.4 `getline()`

可读取一行.

用法:

```
getline(cin, str)
```

6.1.5 cin

cin 为标准输入流, 可使用以下方法读入空格、回车、占位符分割的数据:

```
cin >> var1 >> var2 ...;
```

6.2 输出

6.2.1 printf()

与 scanf() 类似.

6.2.2 putchar()

输出一个字符.

6.2.3 cout

cout 为标准输出流, 可使用以下方法输出无分割的数据:

```
cout << val1 << val2 ...;
```

6.3 重定向到文件

6.3.1 freopen()

用法:

```
freopen(filename, mode, stream);
```

其中 mode 可以填写下列选项, 或它们的组合:

- "r": 读
- "w": 写
- "a": 追加

此外, stream 可以填写类型为 FILE * 的任意变量, 如果要将控制台的输入输出重定向到文件, 只需要对输入输出文件分别填写 stdin 和 stdout.

2023-06-14

新的开始

希望少犯之前犯过的错,

希望少犯之前犯过的错