外部 API 接入 · 2026-11-08

> **默认场景**：H5 嵌在第三方宿主（App WebView / 小游戏平台）里，页面域名不归你，`connect-src 'self'` 锁死，`fetch`、`WebSocket`、`EventSource` 全部不可用。 全文按这个前提写。如果你有自己的页面域名，第 9 节有更省事的路。 更新：2026-11-08

---

0. 先盘点：你手里到底有什么

宿主内嵌场景里，四样东西的归属决定了你能走哪条路：

资源	通常归谁	你能不能改	影响
页面所在的源	宿主（如 `app.builda.game`）	✗	决定能不能用同源反代
页面的 CSP	宿主	✗	决定 `fetch` / `script` / `iframe` 哪条通道活着
一个自己的域名 / Worker	你	✓	决定你能不能有个"接应点"
服务端出口 IP	你的云厂商	✓	决定会不会被上游按 IP 限流

一句话结论：

> 页面源不归你 → **同源反代直接出局** → 只剩下"让浏览器用别的方式发请求"，也就是 `<script src>`（JSONP）。

0.1 决策树

```
页面源是你的吗?
├─ 是 ──▶ 同源反代（第 9 节），最稳，首选
└─ 否
   ├─ script-src 放行你的域名吗?
   │  ├─ 是 ──▶ JSONP + 长轮询（第 3–5 节）   ← 宿主内嵌的唯一现实解
   │  └─ 否
   │     ├─ frame-src 放行吗?  ──▶ iframe + postMessage 桥（第 10 节）
   │     └─ 都锁 ──▶ 找宿主加白名单，没有技术手段能绕
   └─ 你的 Worker/服务器能出网吗?
```

```
├─ 能 ──▶ 用它做接应点
└─ 被上游按 IP 限 ──▶ 再加一级出口（第 6 节）
```

0.2 先跑这三行

在宿主页面的控制台里：

```
location.origin                                          // 页面在哪个源（不是你的就认命）
document.querySelector('meta[http-equiv="Content-Security-Policy"]')?.content
```

再看响应头（浏览器 Network → 文档请求 → Response Headers）：

```
content-security-policy: default-src 'self'; script-src 'self' https://cdn.xxx.com; c
```

把 `connect-src` 和 `script-src` 抄下来，后面每一步都要对照它。

0.3 域名到底怎么来的（别被"二级域名"绕晕）

Cloudflare 的「添加站点」只接受**根域**，例如 `mydomain.com`。`relay.mydomain.com`、`docs.mydomain.com`、`bridge.mydomain.com` **都不是**单独添加的站点，它们只是根域 zone 里的**一条 DNS 记录**。

你想要	在 Cloudflare 怎么来	是不是"添加站点"
`mydomain.com`	Add a site → 改 NS → 等生效	✓ 是，且只需一次
`relay.mydomain.com`	根域 → DNS → 加 CNAME（指向 Pages）或 A（指向 VPS）	✗ 否
`docs.mydomain.com`（Pages）	Pages 项目 → Custom domains 填子域，再建对应 CNAME	✗ 否
Worker 路由 `relay.mydomain.com/*`	根域 zone → Workers Routes 绑定	✗ 否

所以：

- 有根域 `mydomain.com` → 想开几个子域就开几个，不用再"添加站点"
- 没有根域 → 直接用平台给的 `项目名.pages.dev` / `项目名.workers.dev`，一样能跑，只是域名不好看
- 想让 `relay.别人的域名.com` 生效而根域不在你账号里 → 需要对方在 DNS 里给你加 CNAME，你这边做不了

后文所有 `mydomain.com` 都是指**你自己的根域**，`relay.` / `bridge.` / `docs.` 是加在它下面的子域。

---

1. 判断被哪道门拦住

1.1 四种"连不上"

类型	现象	控制台报错	请求发出了吗
A 跨域	curl 正常，网页报错	`blocked by CORS policy: No 'Access-Control-Allow-Origin'`	✓ 发了，响应被扣
B CSP	Network 里根本没有这条请求	`violates ... "connect-src 'self'"`	✗ 没发
C 出口被封	服务端日志有请求，返回 429/403	`HTTP 429`	✓ 发了，对方不认
D 网络不可达	一直转圈或立刻失败	`ERR_NAME_NOT_RESOLVED` / `ERR_CERT_*`	✗ 或半路断

宿主内嵌场景，**九成是 B**。因为 CSP 是宿主统一下发的，所有嵌进去的页面都吃同一套策略。

1.2 三十秒确认

```
# 服务端视角通不通
curl -i -X POST https://api.example.com/v1/x \
  -H "content-type: application/json" -d '{"q":"test"}'
```

curl	浏览器	结论
200	报 CORS	A
200	请求没发出	**B**
429	同样 429	C
超时	同样超时	D

1.3 CSP 指令对照

指令	管什么	被锁后
`connect-src`	`fetch`、`XHR`、`WebSocket`、`EventSource`、`sendBeacon`	**所有"取数据"的方式全死**
`script-src`	`<script src>`、动态插入的 script、`eval`	JSONP 也死
`img-src`	`<img>`、CSS 背景图	图片裂
`frame-src`	`<iframe>`	嵌不了外部页面
`form-action`	表单提交	表单跳转被拦
`default-src`	以上没写时的兜底	全锁

核心事实：`connect-src` 只锁"连接"，不锁"加载资源"。所以 `fetch` 死了，`<script src>` 往往还活着——这是宿主内嵌场景唯一可用的出口。

---

2. 唯一通道：`<script src>`

2.1 为什么它能出去

```
<!-- 受 connect-src 管，锁了就死 -->
<script>fetch('https://relay.mydomain.com/q')</script>
<script>new WebSocket('wss://relay.mydomain.com')</script>
<script>new EventSource('https://relay.mydomain.com/sse')</script>

<!-- 受 script-src 管，connect-src 管不着 -->
<script src="https://relay.mydomain.com/q?cb=cb1"></script>
```

浏览器把 `src` 指向的地址当成**脚本**去加载并执行。只要 `script-src` 放行了这个域名，这段代码就会跑。

而 JS 能执行任意语句——只要服务端把数据包成 `cb1({...})`，执行时就把数据交给了页面。

这不是漏洞，是 JSONP。 它之所以在宿主内嵌场景里还活着，是因为 CSP 的 `connect-src` 从来没打算管脚本加载。

2.2 能力边界（先认清，再动手）

能力	能不能	说明
发 GET	✓	参数全塞 URL
发 POST	✗	script 标签只支持 GET
自定义请求头	✗	带不了 `Authorization`
读到 HTTP 状态码	✗	401/429 都表现为"脚本没执行"
读到响应头	✗	同上
流式读取	✗	脚本一次性执行完
带 Cookie	!	受 `SameSite` 限制，跨站基本带不上

所以 JSONP 拿到的永远是"要么成功执行，要么没执行"，所有业务错误必须由**服务端包进 payload**，HTTP 状态永远返 200。

2.3 什么时候连这条路也没了

控制台出现这句，就是 `script-src` 也锁了：

```
Refused to load the script 'https://relay.mydomain.com/q' because it violates
the following Content Security Policy directive: "script-src 'self'"
```

此时只剩三条路：同源反代（前提你有页面源）、iframe 桥（前提 `frame-src` 放行）、让宿主加白名单。前两条在 9、10 节。

---

3. JSONP 服务端：你自己的接应点

宿主场景里，接应点通常是一个 Cloudflare Worker（免费、自带 HTTPS）。想用自己的域名，就在根域下开一个子域（如 `relay.mydomain.com`）指过来；没有根域就用 `xxx.workers.dev`，不影响功能。

3.1 请求流程

```
宿主页面
  | <script src="https://relay.mydomain.com/q?q=你好&cb=__jp1">
  ▼
```

```
Worker（你的接应点，持上游 Key）
  | 1. 生成 sid，把任务丢给 Durable Object 后台跑
  | 2. 立刻返回 __jp1({"ok":true,"sid":"a1b2c3"})
  ▼
宿主页面拿到 sid，开始轮询
  | <script src="https://relay.mydomain.com/poll?sid=a1b2c3&from=0&cb=__jp2">
  ▼
Worker → DO 取增量 → __jp2({"d":"增量文本","done":0})
```

为什么先返回 sid 而不是直接返回答案：script 加载有超时（浏览器和宿主都可能掐），模型生成要几十秒，一次请求等不起。所以拆成"发起"和"取结果"两段。

3.2 Worker 入口（完整）

```
// src/index.js
const CB_RE = /^[A-Za-z_$][A-Za-z0-9_$.]{0,63}$/;

function js(cb, obj) {
  return new Response(cb + '(' + JSON.stringify(obj) + ')', {
    headers: {
      'content-type': 'application/javascript; charset=utf-8',
      'cache-control': 'no-store',
      'x-content-type-options': 'nosniff',
    },
  });
}

export default {
  async fetch(req, env) {
    const u = new URL(req.url);
    const cb = u.searchParams.get('cb') || '';

    // 没 cb 就当普通 JSON 接口（方便 curl 调试）
    if (cb && !CB_RE.test(cb)) {
      return new Response('bad callback', { status: 400 });
    }

    // 只放行这两条路径，别做成任意代理
    if (u.pathname !== '/q' && u.pathname !== '/poll') {
      return new Response('not found', { status: 404 });
    }
```

```
    if (u.pathname === '/q') {
      const q = (u.searchParams.get('q') || '').slice(0, 4000);
      if (!q) return cb ? js(cb, { ok: false, err: 'empty' }) : Response.json({ ok: f

      const sid = crypto.randomUUID().replace(/-/g, '').slice(0, 12);
      const stub = env.SESSION.get(env.SESSION.idFromName(sid));

      const t = new URL('https://do/start');
      t.searchParams.set('q', q);
      t.searchParams.set('think', u.searchParams.get('think') || '');
      await stub.fetch(t.toString());     // 内部 waitUntil，立即返回

      return cb ? js(cb, { ok: true, sid }) : Response.json({ ok: true, sid });
    }

    // /poll
    const sid = u.searchParams.get('sid') || '';
    if (!/^[a-z0-9]{6,16}$/.test(sid)) {
      return cb ? js(cb, { ok: false, err: 'bad-sid' }) : Response.json({ ok: false,
    }

    const stub = env.SESSION.get(env.SESSION.idFromName(sid));
    const t = new URL('https://do/poll');
    t.searchParams.set('from', u.searchParams.get('from') || '0');
    t.searchParams.set('rfrom', u.searchParams.get('rfrom') || '0');

    const r = await stub.fetch(t.toString());
    const data = await r.text();

    return cb
      ? new Response(cb + '(' + data + ')', {
          headers: {
            'content-type': 'application/javascript; charset=utf-8',
            'cache-control': 'no-store',
            'x-content-type-options': 'nosniff',
          },
        })
      : new Response(data, { headers: { 'content-type': 'application/json' } });
  },
};
```

3.3 会话状态放 Durable Object，不要放 KV

这是 Cloudflare 上最容易踩的坑：

存储	问题
KV	**最终一致**，刚写进去的数据，另一个边缘节点可能读到旧值 → 轮询永远拿不到最新增量
Durable Object	单实例串行，读写强一致，适合"一个会话一个对象"

```
// src/session.js
export class Session {
  constructor(state, env) {
    this.state = state;
    this.env = env;
    this.text = '';
    this.reason = '';
    this.done = 0;
    this.err = '';
  }

  async fetch(req) {
    const u = new URL(req.url);
    const now = Date.now();

    if (u.pathname === '/start') {
      if (this.text || this.done) return Response.json({ ok: true });   // 幂等
      const q = u.searchParams.get('q') || '';
      const think = u.searchParams.get('think') === '1';

      // waitUntil：后台跑，不阻塞这次 fetch
      this.state.waitUntil(this.run(q, think));
      return Response.json({ ok: true });
    }

    // /poll
    const from = Math.max(0, Number(u.searchParams.get('from')) || 0);
    const rfrom = Math.max(0, Number(u.searchParams.get('rfrom')) || 0);

    // 十分钟没人取，标记过期
    if (!this.done && now - (this.at || now) > 600000) {
      this.err = this.err || 'expired';
      this.done = 1;
    }
    this.at = now;

    return Response.json({
```

```
    d: this.text.slice(from),
    rd: this.reason.slice(rfrom),
    done: this.done,
    err: this.err,
  });
}

async run(q, think) {
  const ctl = new AbortController();

  try {
    const up = await fetch('https://api.example.com/v1/chat', {
      method: 'POST',
      headers: {
        'content-type': 'application/json',
        authorization: 'Bearer ' + this.env.UP_KEY,
      },
      body: JSON.stringify({ q, think, stream: true }),
      signal: ctl.signal,
    });

    if (!up.ok) throw new Error('HTTP ' + up.status);

    const reader = up.body.getReader();
    const dec = new TextDecoder();
    let buf = '';

    for (;;) {
      // 20 秒没有任何新数据 → 判死，否则永远转圈
      let idle = null;
      const guard = new Promise((_, rej) => {
        idle = setTimeout(() => rej(new Error('idle-timeout')), 20000);
      });

      let it;
      try {
        it = await Promise.race([reader.read(), guard]);
      } finally {
        clearTimeout(idle);
      }

      if (it.done) break;
      buf += dec.decode(it.value, { stream: true });

      let cut;
```

```
        while ((cut = buf.indexOf('\n')) >= 0) {
          const line = buf.slice(0, cut).trim();
          buf = buf.slice(cut + 1);
          if (!line.startsWith('data:')) continue;

          const payload = line.slice(5).trim();
          if (payload === '[DONE]') { this.done = 1; return; }

          let j;
          try { j = JSON.parse(payload); } catch { continue; }
          const d = j.choices && j.choices[0] && j.choices[0].delta;
          if (!d) continue;
          if (d.reasoning_content) this.reason += d.reasoning_content;
          if (d.content) this.text += d.content;
        }
      }
      this.done = 1;
    } catch (e) {
      this.err = String((e && e.message) || e);
      this.done = 1;
    }
  }
}
```

`wrangler.toml`：

```
name = "my-relay"
main = "src/index.js"
compatibility_date = "2026-11-01"

[[durable_objects.bindings]]
name = "SESSION"
class_name = "Session"

[[migrations]]
tag = "v1"
new_sqlite_classes = ["Session"]

[[routes]]
pattern = "relay.mydomain.com/*"
zone_name = "mydomain.com"      # 根域（已接入 Cloudflare 的那个），不是子域
```

> `zone_name` 填的是根域；`pattern` 里的 `relay.` 是根域下自动生效的子域，不需要单独"添加站点"。

```
wrangler secret put UP_KEY
wrangler deploy
```

3.4 没有 Cloudflare 时：Node 版

```
import express from 'express';
const app = express();
const SESS = new Map();   // sid → { text, reason, done, err, at }
const CB_RE = /^[A-Za-z_$][A-Za-z0-9_$.]{0,63}$/;

function reply(req, res, payload) {
  const cb = String(req.query.cb || '');
  res.set('Cache-Control', 'no-store');
  res.set('X-Content-Type-Options', 'nosniff');
  if (!cb) return res.json(payload);
  if (!CB_RE.test(cb)) return res.status(400).type('text/plain').send('bad callback')
  res.type('application/javascript').send(`${cb}(${JSON.stringify(payload)})`);
}

app.get('/q', (req, res) => {
  const sid = Math.random().toString(36).slice(2, 12);
  const st = { text: '', reason: '', done: 0, err: '', at: Date.now() };
  SESS.set(sid, st);

  runUpstream(String(req.query.q || '').slice(0, 4000), st);   // 后台跑
  reply(req, res, { ok: true, sid });
});

app.get('/poll', (req, res) => {
  const st = SESS.get(String(req.query.sid));
  if (!st) return reply(req, res, { ok: false, err: 'no-session' });

  const from = Number(req.query.from) || 0;
  const rfrom = Number(req.query.rfrom) || 0;
  st.at = Date.now();

  reply(req, res, {
```

```
    d: st.text.slice(from),
    rd: st.reason.slice(rfrom),
    done: st.done,
    err: st.err,
  });
});

async function runUpstream(q, st) {
  try {
    const up = await fetch('https://api.example.com/v1/chat', {
      method: 'POST',
      headers: {
        'content-type': 'application/json',
        authorization: 'Bearer ' + process.env.UP_KEY,
      },
      body: JSON.stringify({ q, stream: true }),
      signal: AbortSignal.timeout(120000),
    });
    if (!up.ok) throw new Error('HTTP ' + up.status);

    const reader = up.body.getReader();
    const dec = new TextDecoder();
    let buf = '';

    for (;;) {
      const { done, value } = await reader.read();
      if (done) break;
      buf += dec.decode(value, { stream: true });

      let cut;
      while ((cut = buf.indexOf('\n')) >= 0) {
        const line = buf.slice(0, cut).trim();
        buf = buf.slice(cut + 1);
        if (!line.startsWith('data:')) continue;

        const payload = line.slice(5).trim();
        if (payload === '[DONE]') { st.done = 1; return; }

        let j;
        try { j = JSON.parse(payload); } catch { continue; }
        const d = j.choices && j.choices[0] && j.choices[0].delta;
        if (!d) continue;
        if (d.reasoning_content) st.reason += d.reasoning_content;
        if (d.content) st.text += d.content;
      }
```

```
    }
    st.done = 1;
  } catch (e) {
    st.err = String((e && e.message) || e);
    st.done = 1;
  }
}

// 十分钟清理
setInterval(() => {
  const now = Date.now();
  for (const [k, v] of SESS) if (now - v.at > 600000) SESS.delete(k);
}, 60000);

app.listen(8787);
```

> 生产环境把 `Map` 换成 Redis，并把 `runUpstream` 拆成独立 worker 进程——Node 单进程里跑长任务，并发一高就互相拖。

3.5 服务端四个必做项

项	原因
callback 名白名单校验	不校验就是反射型 XSS：`cb=alert(1)//` 直接在你页面执行
`Cache-Control: no-store`	否则旧 WebView 会缓存 GET，第二次拿到旧答案
`X-Content-Type-Options: nosniff` + 正确 `content-type`	少了会被当文本拒绝执行
HTTP 永远 200	浏览器读不到状态码，错误只能进 payload

---

4. JSONP 客户端：动态插 script

4.1 完整封装

```
let seq = 0;

function jsonp(url, timeout = 30000) {
```

```
  return new Promise((resolve, reject) => {
    const name = '__jp' + Date.now().toString(36) + (seq++);
    const script = document.createElement('script');
    let settled = false;

    const cleanup = () => {
      try { delete window[name]; } catch (e) { window[name] = undefined; }
      if (script.parentNode) script.parentNode.removeChild(script);
      clearTimeout(timer);
    };

    window[name] = (data) => {
      if (settled) return;
      settled = true;
      resolve(data);
      cleanup();
    };

    const timer = setTimeout(() => {
      if (settled) return;
      settled = true;
      reject(new Error('jsonp-timeout'));
      cleanup();
    }, timeout);

    script.onerror = () => {
      if (settled) return;
      settled = true;
      reject(new Error('jsonp-network'));
      cleanup();
    };

    script.src = url + (url.indexOf('?') >= 0 ? '&' : '?') + 'cb=' + name;
    script.async = true;
    document.head.appendChild(script);
  });
}
```

四个细节都是踩过坑才有的：

细节	不做会怎样
`settled` 标志	超时后又执行回调 → 重复 resolve / 内存泄漏
`cleanup()` 删全局函数	每问一次泄一个 `window.__jp*`，长会话必崩

细节	不做会怎样
`script.onerror`	域名解析失败、被 CSP 拦，只会静默卡到超时
服务端 `no-store`	GET 被缓存，第二次拿到旧答案

4.2 调用

```
const r = await jsonp('/q?q=' + encodeURIComponent('你好'));
if (!r.ok) throw new Error(r.err);
const sid = r.sid;

// 然后轮询（第 5 节）
```

4.3 URL 长度是硬约束

浏览器/WebView	上限
IE / 老 Android WebView	2083 字符
Chrome / Safari 现代版	8 KB 以上，但不保证
经过 CDN / 反代	常被压到 4–8 KB

中文经 `encodeURIComponent` 后每个字 9 字节（`%E4%BD%A0`），200 字问题就 1.8 KB。

做法：客户端先截到 2000 字符；更长的内容（文章、代码）需要"先上传换 token"——但上传本身要 POST，JSONP 做不了，只能由宿主提供上传接口，或把长文本切段分多次发。

---

5. 长轮询：把"流式"拆成"多次取增量"

5.1 状态机

```
① GET /q?q=...&cb=        →  {ok:true, sid}        立即返回，不等结果
② GET /poll?sid&from=0    →  {d:"增量", rd:"", done:0}
③ 重复 ②，from 累加
④ done:1 或 err 非空 → 结束
```

`from` / `rfrom` 是**字符偏移**，服务端每次只回"你还没看过的那一段"。这样即使某次 poll 丢了，下一次也能自动补上，天然支持断线续传。

5.2 客户端实现

```
const sleep = (ms) => new Promise(r => setTimeout(r, ms));

async function ask(question, onDelta, onReason, signal) {
  const start = await jsonp('/q?q=' + encodeURIComponent(question), 20000);
  if (!start.ok) throw new Error(start.err || 'start-failed');

  const sid = start.sid;
  let pos = 0, rpos = 0, idle = 0;
  const deadline = Date.now() + 180000;

  while (Date.now() < deadline) {
    if (signal && signal.aborted) throw new Error('aborted');

    const t0 = Date.now();
    const r = await jsonp(`/poll?sid=${sid}&from=${pos}&rfrom=${rpos}`, 15000);

    if (r.err) throw new Error(r.err);

    if (r.d) { onDelta(r.d); pos += r.d.length; idle = 0; }
    if (r.rd) { onReason(r.rd); rpos += r.rd.length; idle = 0; }
    if (r.done) return;

    // 没新内容 → 指数退避；有内容 → 保持 300ms 节奏
    if (!r.d && !r.rd) {
      idle++;
      await sleep(Math.min(3000, 300 * Math.pow(1.6, Math.min(idle, 6))));
    } else if (Date.now() - t0 < 300) {
      await sleep(300);
    }
  }
  throw new Error('timeout');
}
```

5.3 三个必须处理的细节

细节	不处理会怎样
退避	空轮询 300ms 一次，10 个用户就把 Worker 打满
`visibilitychange` 暂停	用户切后台，页面还在每秒发请求，白烧流量和额度
总时长上限	服务端挂了不返回 `done`，客户端永远轮询

暂停与恢复：

```
let paused = false;
document.addEventListener('visibilitychange', () => {
  paused = document.hidden;
});
// 轮询循环里：while (paused) await sleep(500);
```

5.4 体感优化

轮询天生比流式"顿"。两个办法：

1. **首包优先**：服务端在收到上游第一个 token 时就把 `done:0` 的增量返回，别攒够一句再给
2. **前端补间**：拿到一段增量后按 30ms/字 逐字渲染，视觉上和流式几乎无差别

代价是首字仍然比直连慢 200–600 ms（两次 RTT + 轮询间隔），这是宿主内嵌场景绕不开的成本。

---

6. 出口 IP 被限：再加一跳

6.1 现象

```
本机 curl 上游：200
Worker 调上游：429 / 403，稳定复现
换一把 Key：还是 429
```

6.2 原因

Worker 的出站请求来自 Cloudflare **共享 IP 池**。同一个 IP 上有成千上万个项目在打同一个目标，对方按 IP 限流，你被连坐。

换 Key 没用——限的是 IP，不是账号。

6.3 做法：不同 IP 的二级出口

```
宿主页面 ──script──▶ Worker ──▶ VPS / 云函数（独立 IP）──▶ 上游
```

VPS 上用 Caddy 做一层，必须带 token，否则就是公开代理：

```
relay2.mydomain.com {
    @bad not header X-Relay-Token "你的随机长串"
    respond @bad 403

    reverse_proxy https://api.example.com {
        header_up Host api.example.com
        flush_interval -1
    }
}
```

Worker 只改上游地址：

```
const up = await fetch('https://relay2.mydomain.com/v1/chat', {
  method: 'POST',
  headers: {
    'content-type': 'application/json',
    'x-relay-token': this.env.RELAY_TOKEN,
  },
  body,
});
```

6.4 出口选择

出口	IP 干净度	成本	适合
Cloudflare Worker	差（共享）	免费	一般接口
轻量云 VPS	好（独享）	￥24/月起	被限流时的第一选择

出口	IP 干净度	成本	适合
云函数（Lambda / SCF）	中（池化）	按量	已有云账号
代理池	差且不稳	—	不建议生产

> 换出口只解决"IP 被限"，解决不了"账号被限"。换 IP 后仍 429，看 `Retry-After` 和配额说明。

---

7. 鉴权：JSONP 接口是公开的 GET

JSONP 有个天然劣势：**它就是一个 GET URL**。任何人拿到这个 URL，浏览器地址栏一敲就能用，不需要你的页面。

所以中转必须自己建门槛，否则等于把你的 Key 挂在公网。

7.1 三层

层	做法	挡住什么
来源校验	检查 `Referer` / `Origin` 是否属于宿主域名	别的网站盗用（可伪造，但抬高成本）
应用 token	URL 里带 `t=<随机串>`，服务端比对	随手扫接口的脚本
登录态	URL 带一次性 `tk`，服务端校验后即焚	长期盗用

`Referer` 校验示例：

```
const REF_OK = ['app.builda.game', 'game.builda.game'];
const ref = req.headers.get('referer') || '';
let host = '';
try { host = ref ? new URL(ref).host : ''; } catch (e) {}

if (!REF_OK.includes(host)) {
  return cb ? js(cb, { ok: false, err: 'forbidden' }) : new Response('forbidden', { s
}
```

> `Referer` 可以被伪造，也能被隐私设置抹掉。它只能当"第一道筛子"，不能当唯一鉴权。

7.2 一次性令牌（推荐）

宿主后端签发，页面拿到后塞进 URL：

```
// 签发（你的服务端）
const tk = crypto.randomUUID();
await KV.put('tk:' + tk, uid, { expirationTtl: 300 });   // 5 分钟有效

// 校验（Worker）
const uid = await env.KV.get('tk:' + tk);
if (!uid) return forbidden();
await env.KV.delete('tk:' + tk);   // 用一次就废
```

这样即使 URL 被贴到群里，五分钟后也失效，而且只能用一次。

7.3 限流（令牌桶）

```
const BUCKET = new Map();   // key → { tokens, at }
const RATE = 20 / 60000;    // 每毫秒补多少（20 次/分钟）
const CAP = 20;

function allow(key) {
  const now = Date.now();
  let b = BUCKET.get(key);
  if (!b) b = { tokens: CAP, at: now };

  b.tokens = Math.min(CAP, b.tokens + (now - b.at) * RATE);
  b.at = now;

  if (b.tokens < 1) { BUCKET.set(key, b); return false; }
  b.tokens -= 1;
  BUCKET.set(key, b);
  return true;
}
```

7.4 参数白名单

JSONP 的参数全在 URL 上，更要防：

```
const ALLOW_MODEL = new Set(['model-a', 'model-b']);
if (body.model && !ALLOW_MODEL.has(body.model)) return bad('model');

// URL 里能传的只有这些，其余一律忽略
const q = (u.searchParams.get('q') || '').slice(0, 2000);
```

别把 `q` 原样拼进 system prompt——用户可以通过 URL 注入指令。要放在 `user` 角色里，并对长度、字符做限制。

---

8. 稳定性：三层超时 + 看门狗

层	值	覆盖
连接超时	12 s	建连、等响应头
首字节超时	15–20 s	上游"已接收没开始生成"
空闲超时	20 s	流式中途卡住
总时长	120–180 s	兜底

8.1 `AbortSignal.timeout` 不够用

它只管到"拿到响应头"。上游返回 200 之后挂住不吐字，`reader.read()` 会一直悬着：

```
// ▫ 只防了第一段
const r = await fetch(url, { signal: AbortSignal.timeout(12000) });

// ▫ 读流再加一道
const it = await Promise.race([
  reader.read(),
  new Promise((_, rej) => setTimeout(() => rej(new Error('idle')), 20000)),
]);
```

没有这一道，那次请求既不记成功也不记失败——监控全绿，用户永远转圈。

8.2 重试

状态	重试	原因
401 / 402 / 403 / 429	✓ 换 Key / 换出口	通道不可用
500 / 502 / 503 / 超时	✓ 退避重试	上游抖动
400 / 404 / 413 / 422	✗	参数错，重试一百次也一样

退避：`min(8s, 500ms × 2^n) + 随机 0–300ms`，最多 2 次。

8.3 幂等

重试前问一句：重复执行有副作用吗？

- 查询：随便重试
- 扣费 / 下单：必须带幂等键
- 生成：重试 = 多花一次钱，业务层去重

---

9. 如果你有页面域名：同源反代（更优）

宿主场景用不上，但值得知道差多少。

9.1 判定

```
location.origin   // "https://mygame.com"
```

`location.origin` 就是唯一可用的"同源域名"，不用另买。前提是**这个域名的服务端归你**。

```
curl -i https://mygame.com/api/__probe
# 返回你自己的东西 → 能挂；返回平台 404 页 → 不能
```

9.2 为什么同源就没问题

检查	请求 `https://api.example.com`	请求 `/api/...`
同源策略	跨源，需对方给 `Access-Control-Allow-Origin`	同源，**不触发检查**
`connect-src 'self'`	不在 `'self'`，**请求不发出**	命中 `'self'`，放行

而真正访问外部的是服务器——服务器之间没有"源"的概念，也不执行 CSP。

9.3 Nginx

```
server {
    listen 443 ssl http2;
    server_name mygame.com;

    root /var/www/game;
    index index.html;

    location /api/ {
        proxy_pass https://api.example.com/;

        proxy_ssl_server_name on;
        proxy_set_header Host api.example.com;
        proxy_set_header Authorization "Bearer sk-你的密钥";

        # 流式三件套
        proxy_buffering off;
        proxy_cache off;
        proxy_http_version 1.1;
        proxy_set_header Connection "";

        proxy_read_timeout 300s;
        proxy_send_timeout 300s;
    }
}
```

两个高频错：`proxy_pass` 结尾少了 `/`（路径不剥前缀，404）、没设 `Host`（上游按域名路由，回 404）。

9.4 Caddy

```
mygame.com {
    root * /var/www/game
    file_server

    handle_path /api/* {
        reverse_proxy https://api.example.com {
            header_up Host api.example.com
            header_up Authorization "Bearer sk-你的密钥"
            flush_interval -1
        }
    }
}
```

9.5 两条路的真实差距

	JSONP + 轮询	同源反代
首字延迟	+200–600 ms	几乎无
请求次数	1 次发起 + N 次轮询	1 次
能否流式	✗	✓
能否 POST	✗	✓
URL 长度限制	2 KB	无
服务端复杂度	会话状态 + 轮询	一段配置
调试	只能看服务端日志	正常看 Network

能拿到域名就选反代，这不是风格问题，是上面七项全面占优。

---

10. 连 `script-src` 都锁了：iframe 桥

只剩 `frame-src` 允许你的域名时可用。

```
宿主页面（connect-src / script-src 都锁）
   | postMessage
   ▼
你的桥页面 https://bridge.mydomain.com（CSP 你自己定，宽松）
   | fetch
```

```
    ▼
外部 API
```

父页面：

```
const waits = new Map();
let seq = 0;

addEventListener('message', (e) => {
  if (e.origin !== 'https://bridge.mydomain.com') return;   // 必须校验
  const { id, ok, data, err } = e.data || {};
  const w = waits.get(id);
  if (!w) return;
  waits.delete(id);
  ok ? w.resolve(data) : w.reject(new Error(err));
});

function viaBridge(payload) {
  return new Promise((resolve, reject) => {
    const id = ++seq;
    waits.set(id, { resolve, reject });
    frame.contentWindow.postMessage({ id, payload }, 'https://bridge.mydomain.com');
    setTimeout(() => { if (waits.delete(id)) reject(new Error('timeout')); }, 30000);
  });
}
```

桥页面：

```
<script>
const ALLOW = ['https://app.builda.game'];
addEventListener('message', async (e) => {
  if (!ALLOW.includes(e.origin)) return;
  const { id, payload } = e.data;
  try {
    const r = await fetch('https://api.example.com/v1/chat', {
      method: 'POST',
      headers: { 'content-type': 'application/json' },
      body: JSON.stringify(payload),
    });
    parent.postMessage({ id, ok: r.ok, data: await r.text() }, e.origin);
  } catch (err) {
    parent.postMessage({ id, ok: false, err: String(err) }, e.origin);
```

```
  }
});
</script>
```

两个前提：桥页面不能有 `X-Frame-Options: DENY`；`postMessage` 必须校验 `origin`。

缺点：多一层、延迟 +30–100 ms、调试更难。能 JSONP 就别用这个。

---

11. 排错对照表

报错	根因	处理
`blocked by CORS policy`	无 `Access-Control-Allow-Origin`	同源反代（有域名时）或 Worker 加头
`violates "connect-src 'self'"`	CSP 禁外连	JSONP（第 3 节）
`Refused to load the script ... "script-src"`	script 也锁	同源反代 / iframe 桥 / 找宿主
`Refused to display ... in a frame`	`X-Frame-Options`	桥页面去掉该头
`net::ERR_NAME_NOT_RESOLVED`	DNS	换域名、确认解析
`net::ERR_CERT_COMMON_NAME_INVALID`	证书不匹配	用正规证书
Mixed Content	HTTPS 页面调 HTTP	全站 HTTPS
`HTTP 429`，换 Key 无效	出口 IP 被限	二级出口（第 6 节）
流式"憋一大段"	反代开了缓冲	`proxy_buffering off` / `flush_interval -1`
流式不结束	无空闲超时	20 s 看门狗
JSONP `Unexpected token`	没包 callback	响应必须是 `cb({...})`
JSONP 拿到旧数据	GET 被缓存	`Cache-Control: no-store`
JSONP 回调不执行，也无报错	`content-type` 不对	`application/javascript` + `nosniff`
轮询一直空	会话存 KV，一致性延迟	换 Durable Object
请求体 413	超过限制	调 `client_max_body_size` 或分片

---

12. 上线清单

通道

- [ ] `location.origin` 确认页面源归属，选对了路
- [ ] `connect-src` / `script-src` 都抄下来核对过
- [ ] JSONP 接口用 curl 能返回 `cb({...})`
- [ ] 宿主真机上跑通一次完整问答（不是只在本机浏览器）
- [ ] 出口 IP 压测过，没触发 429

安全

- [ ] callback 名做了白名单校验
- [ ] 中转有 token / 一次性令牌，不是裸 GET
- [ ] `Referer` 白名单 + 一次性令牌双保险
- [ ] 参数白名单：`q` 长度、模型名、`max_tokens` 都有上限
- [ ] Key 只在服务端，`grep -r "sk-"` 无命中
- [ ] 日志不打印完整 Key（只留前 6 位）

稳定

- [ ] 三层超时都配了，尤其是空闲看门狗
- [ ] 轮询有退避，切后台会暂停
- [ ] 401/429/5xx 会换通道，400/422 不重试
- [ ] 有 `/health` 和一次真实调用的冒烟测试
- [ ] 会话有 TTL，内存不会涨爆

---

13. 一页速查

```
# 我在哪个源
location.origin

# 宿主到底锁了什么
curl -sI https://app.builda.game | grep -i content-security-policy
```

```
# 我的接应点通不通
curl -s "https://relay.mydomain.com/q?q=hi&cb=t"   # 应返回 t({...})
```

```
页面源归你          → 同源反代（最优）
页面源不归你
  ├ script-src 放行 → JSONP + 长轮询（宿主场景唯一解）
  ├ 只放 frame-src  → iframe 桥
  └ 都锁            → 找宿主加白名单
出口 IP 被限        → 加一跳 VPS / 云函数
```

关键数字：

项	值
连接超时	12 s
空闲超时	20 s
JSONP 超时	30 s
轮询间隔	300 ms 起，退避到 3 s
轮询总时长	180 s
URL 上限	2 KB
请求体上限	64 KB
一次性令牌 TTL	300 s

页面由 Markdown 生成，单文件、无外部依赖，可直接托管在 Cloudflare Pages / Workers。