

## 前言

因为国内网络环境的原因，执行 `npm i` 安装依赖时，肯定会遇到 安装过慢 或 安装失败 的情况。有经验的同学通常会在 Node 安装完毕顺便把 Npm镜像 设置为国内的淘宝镜像。

```
npm config set registry https://registry.npmirror.com/
```

这样就能应付很多 `npm i` 的安装情况。当然这只是解决很多 安装过慢 或 安装失败 的情况，随着项目的深入开发，肯定还会遇到一些更奇葩的情况。本章将带领你**填埋 Npm镜像那些险象环生的坑**，通过多方面探讨 Npm镜像 问题，全方位解决 安装过慢 或 安装失败，使 `npm i` 安装依赖时更顺畅。

## 背景：如何优雅切换Npm镜像

若开源一个 Npm模块，在开发时使用淘宝镜像，但发布时必须使用原镜像。在着手解决上述问题前，先推荐一个**镜像管理工具**。

- 原镜像：<https://registry.npmjs.org/>
- 淘宝镜像：<https://registry.npmirror.com/>

## 管理镜像

主角就是 `nrm`，它是一个可随时随地自由切换 Npm镜像 的管理工具。打开 CMD工具，执行 `npm i -g nrm` 安装 `nrm`，再执行 `nrm -V`，输出版本表示安装成功。

有了它，上述何时使用何种镜像的问题就迎刃而解了。只需掌握以下命令就能操作 `nrm`。

命令	功能
<code>nrm add &lt;name&gt; &lt;url&gt;</code>	新增镜像
<code>nrm del &lt;name&gt;</code>	删除镜像
<code>nrm test &lt;name&gt;</code>	测试镜像
<code>nrm use &lt;name&gt;</code>	切换镜像
<code>nrm current</code>	查看镜像
<code>nrm ls</code>	查看镜像列表

熟悉命令后来一波操作，原镜像与淘宝镜像随意切换。当然记性好也无需该工具。

```
root@JowayYoungMac:/Users/yangzw# nrm ls

npm ----- https://registry.npmjs.org/
yarn ----- https://registry.yarnpkg.com/
tencent ----- https://mirrors.cloud.tencent.com/npm/
cnpm ----- https://r.cnpmjs.org/
taobao ----- https://registry.npmmirror.com/
npmMirror ---- https://skimdb.npmjs.com/registry/

root@JowayYoungMac:/Users/yangzw# nrm test taobao

taobao --- 321ms

root@JowayYoungMac:/Users/yangzw# nrm use taobao

Registry has been set to: https://registry.npm.taobao.org/
```

## 遇坑填坑

有了 `nrm` 切换到淘宝镜像，安装速度明显加快，但遇到安装的 `Npm` 模块 依赖了 `C++` 模块 那就坑爹了。在安装时会隐式安装 `node-gyp`，`node-gyp` 可编译这些依赖 `C++` 模块 的 `Npm` 模块。

那问题来了，`node-gyp` 在首次编译时会依赖 `Node` 源码，所以又悄悄去下载 `Node`。虽然在安装依赖时已切换到淘宝镜像，但一点卵用都有。这样又因为国内网络环境的原因，再次遇到 安装过慢 或 安装失败 的情况。感觉是死循环啊！

还好 `npm config` 提供一个参数 `disturl`，它可设置 **Node** 镜像地址，当然还是将其指向国内的淘宝镜像。这样又能愉快地安装这些依赖 **C++** 模块的 **Npm** 模块了。

```
npm config set disturl https://npm.taobao.org/mirrors/node/
```

问题一步一步地解决，接着又出现另一个问题。平常都会使用 `node-sass` 作为开发依赖，但 `node-sass` 的安装一直都是一个让人头疼的问题。

安装 `node-sass` 时，在 `install` 阶段 会从 **Github Releases** 中下载一个叫 `binding.node` 的文件，而 **Github Releases** 中的文件都托管在 `s3.amazonaws.com` 中，该网址被Q了，所以又安装不了。

然而办法总比困难多，从 `node-sass` 的官方文档中可找到一个叫 `sass_binary_site` 的参数，它可设置 **Sass** 镜像地址，毫无疑问还是将其指向国内的淘宝镜像。这样又能愉快地安装 `node-sass` 了。

```
npm config set sass_binary_site https://npm.taobao.org/mirrors/node-sass/
```

其实还有好几个类似的模块，为了方便使用，我还是把它们源码中的镜像参数地址扒出来，统一设置方便安装。它们分别是 `electron`、`phantom`、`puppeteer`、`python`、`sass`、`sentry`、`sharp` 和 `sqlite`。

## 配置镜像地址

`npm config set <name> <url>`，赶紧**一键复制，永久使用**。特别注意，别漏了最后的 `/`。

```
npm config set electron_mirror https://npm.taobao.org/mirrors/electron/
npm config set phantomjs_cdnurl https://npm.taobao.org/mirrors/phantomjs/
npm config set puppeteer_download_host https://npm.taobao.org/mirrors/
npm config set python_mirror https://npm.taobao.org/mirrors/python/
npm config set sass_binary_site https://npm.taobao.org/mirrors/node-sass/
npm config set sentrycli_cdnurl https://npm.taobao.org/mirrors/sentry-cli/
npm config set sharp_binary_host https://npm.taobao.org/mirrors/sharp/
npm config set sharp_dist_base_url https://npm.taobao.org/mirrors/sharp-libvips/
npm config set sharp_libvips_binary_host https://npm.taobao.org/mirrors/sharp-libvips/
npm config set sqlite3_binary_site https://npm.taobao.org/mirrors/sqlite3/
```

有了这波操作，执行 `npm i` 安装上述 `Npm模块` 就能享受国内速度了。若有条件，建议把这些镜像文件搬到自己或公司的服务器中，将镜像地址指向自有的服务器。在公司内网搭建一个这样的镜像服务器，一直安装一直爽。

```
npm config set electron_mirror https://mirror.yangzw.vip/electron/
```

## 分析：通过node-sass探究镜像配置

### 源码

以经常卡住的 `node-sass` 为例，以下是坑爹货 `node-sass/lib/extensions.js` 的[源码部分](#)，可看出它会默认走 `Github Releases` 的托管地址，上述也分析过原因，在此就不重复了。

源码部分随着版本更新可能会有所变化，但大概的逻辑差不多一样。

txt

```
function getBinaryUrl() {
  const site = getArgument("--sass-binary-site")
    || process.env.SASS_BINARY_SITE
    || process.env.npm_config_sass_binary_site
    || (pkg.nodeSassConfig && pkg.nodeSassConfig.binarySite)
    || "https://github.com/sass/node-sass/releases/download";
  const result = [site, "v" + pkg.version, getBinaryName()].join("/");
  return result;
}
```

js

而其他 `Npm模块` 也有类似的代码，例如 `puppeteer` 安装 `Chromium` 的[源码部分](#)，有兴趣的同学去扒下源码，如出一辙。

源码部分随着版本更新可能会有所变化，但大概的逻辑差不多一样。

txt

```
async function download() {
  await compileTypeScriptIfRequired();
  const downloadHost =
    process.env.PUPPETEER_DOWNLOAD_HOST
    || process.env.npm_config_puppeteer_download_host
    || process.env.npm_package_config_puppeteer_download_host;
  const puppeteer = require("./index");
```

js

```
const product =
  process.env.PUPPETEER_PRODUCT
  || process.env.npm_config_puppeteer_product
  || process.env.npm_package_config_puppeteer_product
  || "chrome";
const browserFetcher = puppeteer.createBrowserFetcher({
  product,
  host: downloadHost,
});
const revision = await getRevision();
await fetchBinary(revision);
// 还有很多
}
```

## 原因

因为 `node-sass` 是经常使用的开发依赖，也是安装时间较长与最常见到报错的 `Npm` 模块，在此就花点篇幅分析与解决可能会遇到的问题。

`node-sass` 安装失败的原因其实并不止上述提到的情况，可从安装时分析并获取突破口解决问题。分析 `npm i node-sass` 输出信息可得到以下过程。

- 检测项目 `node_modules` 文件夹的 `node-sass` 是否存在且当前安装版本是否一样
  - **Yes**: 跳过，完成安装过程
  - **No**: 进入下一步
- 从 `Npm` 公有仓库 中下载 `node-sass`
- 检测 全局缓存 或 项目缓存 中是否存在 `binding.node` 文件
  - **Yes**: 跳过，完成安装过程
  - **No**: 进入下一步
- 从 `Github Releases` 中下载 `binding.node` 文件并将其缓存到全局
  - **Success**: 将版本信息写入 `package-lock.json`
  - **Error**: 进入下一步
- 尝试本地编译出 `binding.node` 文件
  - **Success**: 将版本信息写入 `package-lock.json`
  - **Error**: 输出错误信息

不难看出，`node-sass` 依赖了一个二进制文件 `binding.node`，不仅需从 `Npm` 公有仓库 中下载本体还需从 `Github Releases` 中下载 `binding.node` 文件。

从实际情况来看，`node-sass` 出现 安装过慢 或 安装失败 的情况可能存在以下原因。

## Npm镜像托管在国外服务器

执行以下命令解决。

```
nrm use taobao
```

安装时悄悄下载 `node-gyp`

执行以下命令解决。

```
npm config set disturl https://npm.taobao.org/mirrors/node/
```

`binding.node` 文件托管在国外服务器

执行以下命令解决。

```
npm config set sass_binary_site https://npm.taobao.org/mirrors/node-sass/
```

`node-sass`版本 与 `Node`版本 不兼容

`node-sass`版本 兼容性好差，必须与 `Node`版本 对应使用才行，可查看[node-sass-version-association](#)，复用官方文档的版本对照表。

NodeJS	Supported node-sass version	Node Module
Node17	7.0+	102
Node16	6.0+	93
Node15	5.0+,<7.0	88
Node14	4.14+	83
Node13	4.13+,<5.0	79
Node12	4.12+	72
Node11	4.10+,<5.0	67
Node10	4.9+,<6.0	64
Node8	4.5.3+,<5.0	57
Node<8	<5.0	<57

执行 `npm i` 安装依赖前请确保当前的 `node-sass`版本 与 `Node`版本 已兼容。

全局缓存中的 `binding.node`版本 与 `Node`版本 不兼容

若 本地环境 使用 `Nvm` 管理 `Node`版本 且已切换了 `Node`版本 , 在安装时可能会出现 `Windows/OS X/Linux 64-bit with Node.js 16.x` 这样的提示, 这种情况也是我经常遇到的。

我电脑安装了30多个Node版本且经常来回切换。

txt

从上述表格可知 `node-sass`版本 与 `Node`版本 是关联的, 修改 `Node`版本 后在全球缓存中匹配不到对应 `binding.node` 文件而导致安装失败。根据错误提示, 清理 `Npm`缓存 再重新安装。

```
npm cache clean -f
npm rebuild node-sass
```

所以没事就别来回切换 **Node版本**，像我装这么多 **Node版本** 也是迫不得已，老项目太多了😅。

## 安装失败后重新安装

有可能无权限卸载已安装的内容，导致重新安装时可能产生某些问题，建议将 **node\_modules** 文件夹全部卸载并重新安装。

在 **MacOS系统** 中卸载 **node\_modules** 文件夹较快，但在 **Windows系统** 中卸载 **node\_modules** 文件夹较慢，推荐使用[rimraf](#)卸载 **node\_modules** 文件夹，它是一个 **Node** 的 **rm -rf** 工具。

```
npm i -g rimraf
```

在 **package.json** 中指定 **scripts**，让 **rimraf** 常驻，三大操作系统通用。

```
{
  "scripts": {
    "reset": "rimraf node_modules package-lock.json yarn.lock && npm i"
  }
}
```

json

有 **安装失败** 与 **重新安装** 相关操作，执行 **npm run reset**。

## 方案：填埋Npm镜像那些险象环生的坑

若看得有点乱，那直接贴出命令执行顺序，建议前端小白在 **Node** 安装完毕立即处理这些镜像问题，防止后续产生不必要的麻烦。

```
# 查看Node版本与Npm版本，确认已安装Node环境
node -v
npm -v

# 全局安装nrm并设置Npm镜像为淘宝镜像
npm i -g nrm
```



```
nrm use taobao
```

```
# 设置依赖在安装时内部模块下载的Node镜像为淘宝镜像
```

```
npm config set disturl https://registry.npmirror.com/node/
```

```
# 设置常见Npm模块的淘宝镜像
```

```
npm config set electron_mirror https://npm.taobao.org/mirrors/electron/
```

```
npm config set phantomjs_cdnurl https://npm.taobao.org/mirrors/phantomjs/
```

```
npm config set puppeteer_download_host https://npm.taobao.org/mirrors/
```

```
npm config set python_mirror https://npm.taobao.org/mirrors/python/
```

```
npm config set sass_binary_site https://npm.taobao.org/mirrors/node-sass/
```

```
npm config set sentrycli_cdnurl https://npm.taobao.org/mirrors/sentry-cli/
```

```
npm config set sharp_binary_host https://npm.taobao.org/mirrors/sharp/
```

```
npm config set sharp_dist_base_url https://npm.taobao.org/mirrors/sharp-libvips/
```

```
npm config set sharp_libvips_binary_host https://npm.taobao.org/mirrors/sharp-libvips/
```

```
npm config set sqlite3_binary_site https://npm.taobao.org/mirrors/sqlite3/
```

针对 `node-sass` 的情况。

```
# 全局安装rimraf
```

```
npm i -g rimraf
```

```
# 安装前请确保node-sass版本与当前Node版本已兼容
```

```
# 安装失败
```

```
npm cache clean -f
```

```
npm rebuild node-sass
```

```
# 或 npm run reinstall
```

在 `package.json` 中指定 `scripts` 。

```
{  
  "scripts": {  
    "reset": "rimraf node_modules package-lock.json yarn.lock && npm i"  
  }  
}
```

## 总结

上述关于镜像的所有操作都是在 **本地环境** 中部署，还需将其在 **服务器环境** 中部署一次。

镜像问题的坑确实很多，归根到底还是网络环境导致的。当然这些问题也阻碍不了乐于探索的你，办法总比困难多，坚持下去始终能找到解决方案。

我总结出一个解决镜像问题的好方式，遇到一些上述未提到的 **Npm模块**，可尝试通过以下步骤解决问题。

- 执行 `npm i` 前设置淘宝镜像，保证安装依赖时都走国内网络
- 安装不成功时，肯定是在安装时该模块内部又去下载了其他国外服务器的文件
- 在 **Github** 中克隆一份该模块的源码并分析下载逻辑，搜索包括 `base`、`binary`、`cdn`、`config`、`dist`、`download`、`host`、`mirror`、`npm`、`site`、`url` 等相关关键词(通常**mirror**的匹配度最高)
- 在搜索结果中查找形态像镜像地址的代码块，再分析该代码块的功能并提取最终的镜像地址，例如 `node-sass` 的 `sass_binary_site`
- 去淘宝镜像官网、百度、谷歌等网站查找所需镜像地址，若实在找不到就规范上网把国外服务器的镜像文件拉下来搬到自己或公司的服务器中
- 设置模块依赖的镜像地址：`npm config set <name> <url>`
- 重新执行 `npm i` 安装依赖

本章内容到此为止，希望能对你有所启发，欢迎你把自己的学习心得打到评论区！

☑ 示例项目：[fe-engineering](#)

☑ 正式项目：[bruce](#)

## 留言

输入评论 (Enter换行, Ctrl + Enter发送)

发表评论

## 全部评论 (9)



ougege\_

前端开发 @ 浙江云针 1月前

常见模块的镜像地址可以用npmrc 文件维护

👍 点赞    💬 回复



qingkooo

前端开发 1月前

其实科学下就可以避免技术的复杂性了，对吗

点赞 回复



恰柠檬男孩

辣鸡前端工程师 1月前

现在不需要node-sass了吧？

点赞 1

JowayYoung (作者) 1月前

你有考虑老项目吗，去npm看看node-sass与sass的下载量，对比下

点赞 回复



NewName

前端CV 2月前

很有用，总结大师

点赞 回复



用户3104234440...

4月前

服务器怎么执行镜像操作，服务器啥也没有

点赞 1

JowayYoung (作者) 4月前

可以详细说下什么情况吗，你的描述我听得不是很懂

点赞 回复

Ramirez

Web前端复制粘贴工程... 4月前

还是用dart-sass吧

点赞 1

JowayYoung (作者) 4月前

老项目无法迁移dart-sass还不是得处理

点赞 回复

