

Claude Code 画像生成 接続キット

APIキーを使わずに、Claude Codeから画像を作るための手順書です。4つの経路それぞれの「実際に打つもの」と、詰まったときの見分け方をまとめています。

2通りの使い方があります。

- **読む:** 上から順に読んで、自分の環境に合う経路を1つ選びます
- **読ませる:** このファイルをプロジェクトに置き、Claude Codeに「接続キットを読んで、②の手順で画像生成の環境を作って」と頼みます

0. まず1つ選ぶ

経路	必要なもの	向く場面	速さ
① Codex CLI	Codex CLI とChatGPT のログインだけ	月に数枚。すぐ始めたい	中
② CDP直結	専用プロファイルの Chrome + リモートデバッグ	何十枚も作る。止めて 再開したい	速い
③ Claude in Chrome	Chrome拡張機能だけ	端末の設定を触れない	遅い
④ Playwright	Node と Playwright	Claude Codeの外で自 動化したい	速い

どれもChatGPTの月額の中で動きます。APIキーは1つも要りません。ただし①だけは、ChatGPTの利用枠を通常の会話より3～5倍速く消費します（OpenAIの公式ヘルプに明記）。

1. ① Codex CLI

用意する

```
npm install -g @openai/codex
codex login          # ブラウザが開くのでChatGPTのアカウントでログイン
codex login status   # "Logged in using ChatGPT" と出れば完了
```

画像を作る

Codexにタスクとして頼みます。Codexが内部の画像生成ツールを呼び、OAuthのまま生成につながります。

```
codex exec --skip-git-repo-check "次のプロンプトで画像を1枚生成して、カレントディレクトリに out.png として保存してください。プロンプト: <ここに英語または日本語のプロンプト>"
```

生成物はCodexの生成画像フォルダ（`~/codex/generated_images/`）にも残るので、必要ならそこからコピーします。

落とし穴

- OAuthのトークンで画像生成のREST APIを直接叩くと401です。あれはセッショントークンで、`sk-`から始まるAPIキーではありません。必ず `codex exec` を通します
- Windowsではコマンドライン全体が8,191文字を超えると、0秒・終了コード1・出力なしで即死します。長いプロンプトはファイルに書いて標準入力から渡してください（`codex exec ... - <prompt.txt`）
- 枠の減りが速いので、何十枚も作る用途には向きません

2. ② CDP直結（一番安定します）

Chromeを専用のプロファイルでリモートデバッグ起動し、そこにNodeから直接つなぐ方式です。

2-1. 専用プロファイルでChromeを起動する

PowerShellでの例です。フォルダ名に空白を入れないでください（理由は5章）。

```
Start-Process "C:\Program Files\Google\Chrome\Application\chrome.exe" -  
    ArgumentList `  
    '--remote-debugging-port=9333', '--remote-allow-origins=*', `  
    '--user-data-dir="C:\Users\<ユーザー名  
        >\AppData\Local\Google\Chrome\DebugChatGPT"', `  
    '--disable-backgrounding-occluded-windows', '--disable-renderer-backgrounding',  
    `  
    '--disable-background-timer-throttling', `  
    '--no-first-run', '--no-default-browser-check', 'https://chatgpt.com/'
```

macOS / Linux なら次の形になります。

```
"/Applications/Google Chrome.app/Contents/MacOS/Google Chrome" \  
--remote-debugging-port=9333 --remote-allow-origins='*' \  
--user-data-dir="$HOME/chrome-debug-chatgpt" \  
--disable-backgrounding-occluded-windows --disable-renderer-backgrounding \  
--disable-background-timer-throttling \  
--no-first-run https://chatgpt.com/
```

フラグの意味

フラグ	何のため
<code>--remote-debugging-port</code>	ここにつながます。普段使いのChromeとは別の番号にしてください
<code>--remote-allow-origins=*</code>	これが無いと接続が拒否されます
<code>--user-data-dir</code>	専用プロファイル。普段使いのプロファイルを指定するとデバッグ接続が拒否されます
<code>--disable-backgrounding-occluded-windows</code> <code>--disable-renderer-backgrounding</code> <code>--disable-background-timer-throttling</code>	この3つが無いと、最小化した瞬間に生成が激遅になります

起動したら、そのChromeでChatGPTにログインします（Plus / Team / Enterprise など、画像生成が使えるプラン）。ログイン状態はこのプロファイルにだけ残ります。

2-2. 生きているか確認する

```
curl -s -o /dev/null -w "%{http_code}\n" http://127.0.0.1:9333/json/version
```

200 が返れば生きています。何も返らなければ、起動に失敗しています（5章の1番を見てください）。

2-3. 自分専用のタブを1枚作って固定する

```
curl -s "http://127.0.0.1:9333/json/new?https://chatgpt.com/"
```

返ってきたJSONの `id` を控えます。以降の操作は必ずこのidのタブに対して行います。1枚しか作らないときも省略しないでください（理由は5章の3番）。

2-4. 生成する

Claude Codeに次のように頼みます。

ポート9333のChromeの、タブID <控えたid> を使ってください。
そのタブでChatGPTに次のプロンプトを送り、返ってきた画像を <保存先> に保存してください。
1枚ごとに新しいチャットに移ってから送ってください。
プロンプト: <ここに>

複数枚を作るときは、先に「プロンプトと保存先ファイル名の組」の一覧を作ります。

```
[  
  { "prompt": "...", "out": "C:/work/images/img_01.png" },  
  { "prompt": "...", "out": "C:/work/images/img_02.png" }  
]
```

そのうえで、Claude Codeにこう頼みます。

この一覧のうち、out のファイルがまだ存在しないものだけを作ってください。
1枚ごとに新規チャットへ移り、1枚保存するたびにファイルができたことを確認してから次へ進んでください。
途中で失敗したら、そこで止めずに次へ進み、最後に失敗した分の一覧を出してください。

「まだ存在しないものだけ」が、このキットでいちばん効くところです。途中で止まっても、もう一度同じことを頼めば続きから走るので、最初からやり直す必要はありません。

2-5. 守ること

- 1枚ごとに新しいチャットへ移る。同じチャットで4枚以上続けると、送信ボタンが押せなくなります
- 同時に走らせるのは合計2本まで。3本以上にすると「リクエストが多すぎます」でアカウントごと一時的に止まります
- 作業が終わったら専用Chromeを閉じる（7章）

3. ③ Claude in Chrome

1. Chromeに Claude in Chrome の拡張機能を入れます
2. 拡張機能の設定で、ChatGPTのドメインに対する操作を許可します
3. Claude Codeから「ChatGPTを開いて、このプロンプトを送って、出てきた画像を保存して」と頼みます

リモートデバッグポートも専用プロファイルも要りません。そのぶん②より遅いので、1日に数枚までの使い方に向きます。

4. ④ Playwright

②と同じことを、Claude Codeを介さずスクリプト単体でやります。

```
npm i -D playwright
npx playwright install chromium
```

ログイン状態は、一度手でログインしたときのものをファイルに保存して使い回す形です。

```
npx playwright open --save-storage=auth.json https://chatgpt.com/
```

以降、スクリプト側で `browser.newContext({ storageState: 'auth.json' })` として起動すれば、ログイン済みの状態で画面を操作できます。

画面の構造が変わると動かなくなります。直せる前提で使ってください。

5. つまづき集（4つとも、エラーが出ません）

1. プロファイル名に空白を入れると、別のChromeを掴む

`--user-data-dir` の値に空白があると、そこで指定が2つに割れます。 `Debug ChatGPT` なら `Debug` まだがプロファイル、余った `ChatGPT` はURLとして別ウィンドウに開きます。しかもその `Debug` を他の何かが使っていると、こちらのChromeは既存のプロセスに引き渡して自分は終了します。

症状: 起動したのにポートが応答しない。 **対処:** フォルダ名に空白を入れない。PowerShellでは値を二重引用符で囲む。

確認: 起動中のChromeのコマンドラインに、意図したプロファイル名がそのまま出ているかを見ます。

```
Get-CimInstance Win32_Process -Filter "Name='chrome.exe'" |  
  Where-Object { $_.CommandLine -match 'remote-debugging-port' } |  
  Select-Object -ExpandProperty CommandLine
```

2. 専用Chromeが落ちても、終了コードは0で返る

落ちた状態で生成を回すと、1枚ずつ0秒で空振りして、正常終了として返ってきます。ログには成功が並び、フォルダには1枚も増えていません。

対処: 生成に入る前に毎回 2-2 のポート確認をします。落ちていたら立て直し、タブIDも取り直します（前のタブは死んでいます）。

「終了コード0」と「成功した枚数」を完了の根拠にしないでください。根拠は、フォルダにできたファイルです。

3. タブを固定しないと、別の作業の画像が入ってくる

生成の処理は、既定では「最初に見つかったChatGPTのタブ」を掴みます。同じChromeを別のセッションも触っていると、両方が同じタブを奪い合い、相手の画像が自分の保存先に落ちます。

対処その1: 2-3 で自分専用のタブを作り、そのIDを固定します。 **対処その2:** 出来上がりを1枚ずつ目で見ます。並べて見るのが速いので、全部を1枚のシートにまとめ、各セルにファイル名・寸法・縦横比を焼き込みます。指定していない比率が混ざっていれば、その時点で浮きます。

ハッシュの重複チェックでは見つかりません。別の画像なので、ハッシュは当然違うからです。

4. 最小化すると激遅になる

最小化した瞬間、Chromeが「見えていない」と判断して描画を節電に落とします。2-1 の3つのフラグで止まります。

このフラグは起動時にしか効きません。すでに動いているChromeに後から足せないで、立て直してください。

6. 指定と違う絵が出るとき

タブの奪い合い（5章の3番）でなければ、共通のスタイル指定と、その1枚の内容が矛盾しているのが原因です。

全部に「薄暗い室内の照明」と付けたまま「夕暮れの商店街」を頼むと、指示の弱いほうが負けて室内の絵が出ます。共通指定を使うときは、屋外・俯瞰・テクスチャなど「その照明では成立しない絵」を先に洗い出し、その1枚だけ文を差し替えます。

これは機械では検出できません。出来上がりを、見出しやキャプションの文言と突き合わせて読んでください。

7. 終わったら閉じる

リモートデバッグのポートは、その端末の中からは触れる状態です。作業が終わったら専用Chromeを落とします。

```
Get-CimInstance Win32_Process -Filter "Name='chrome.exe'" |  
  Where-Object { $_.CommandLine -match 'DebugChatGPT' } |  
  ForEach-Object { Stop-Process -Id $_.ProcessId -Force }
```

ログインごと消したいときは、専用プロファイルのフォルダを削除します。

8. チェックリスト

- ☐ 専用プロファイルのフォルダ名に空白が入っていない
 - ☐ ポート番号を普段使いと分けた
 - ☐ 3つのバックグラウンド抑止フラグを付けた
 - ☐ `/json/version` が200を返す
 - ☐ 自分専用のタブを作ってIDを控えた
 - ☐ 1枚ごとに新規チャットへ移る指示を入れた
 - ☐ 同時に走らせるのは2本まで
 - ☐ 出来上がりを1枚ずつ目を見た（寸法と縦横比を焼き込んだシートで）
 - ☐ 終わったら専用Chromeを閉じた
-

株式会社スターストリーム・スタジオ