もうデバッグで悩まない！

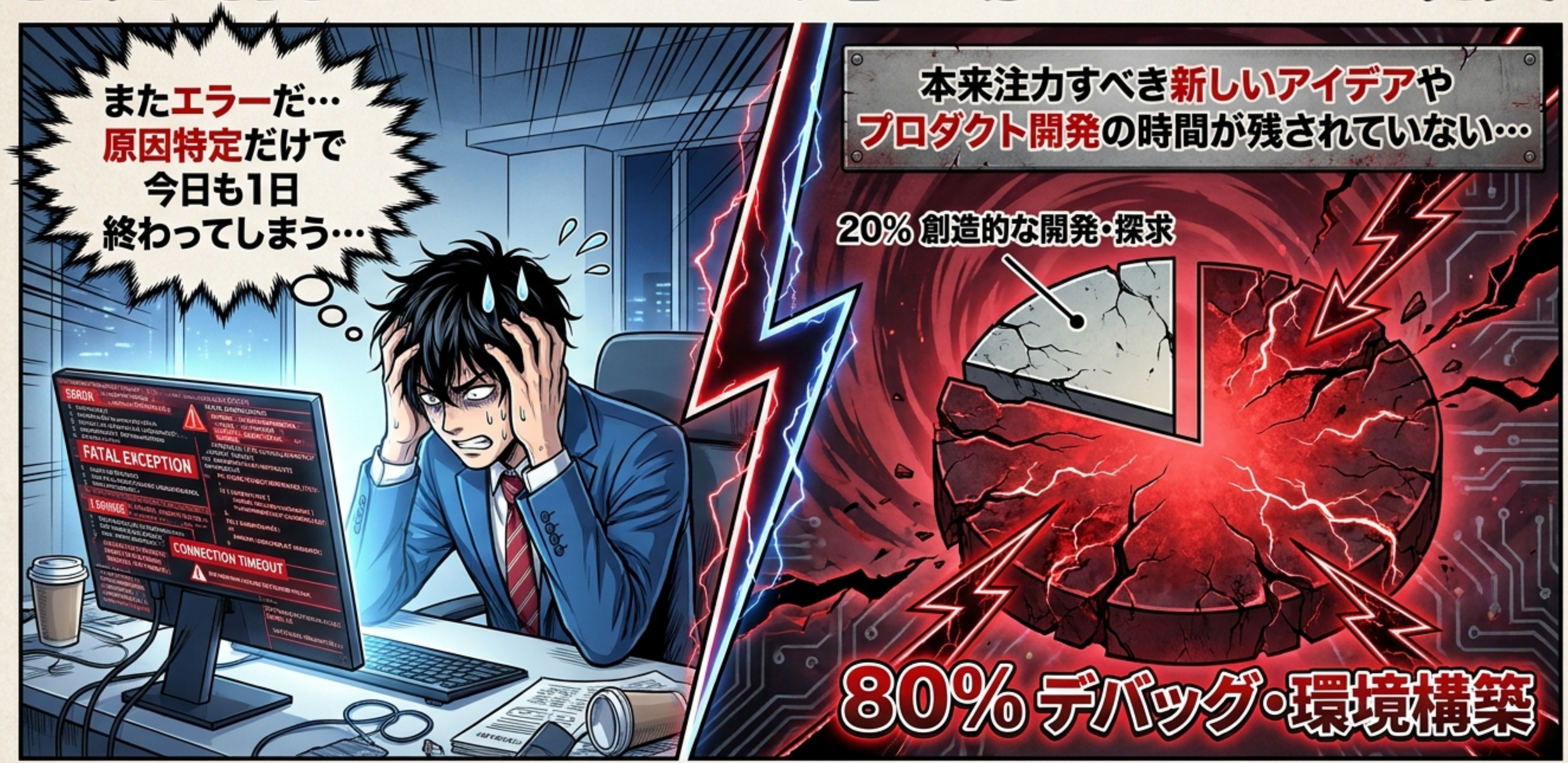
何時間もエラーとにらめっこするのは、今日で終わりにしよう。

リアイニ絵画に

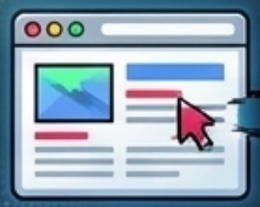
Claudeにバグ調査とコード修正を任せる完全ステップアップ手順書



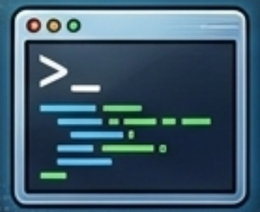
開発時間の80%が「バグ」に奪われている現実



デバッグを丸投げする3つのステップアップ



1. フロントエンドの異変: 画面のエラーや表示崩れを丸投げ！即時デバッグ



2. 手ごわいバックエンド: Claude Codeの「エフォートレベル」を使いこなす



3. 見えないロジック: 監視UIと反復ループ（Closed Loop）による自律改善

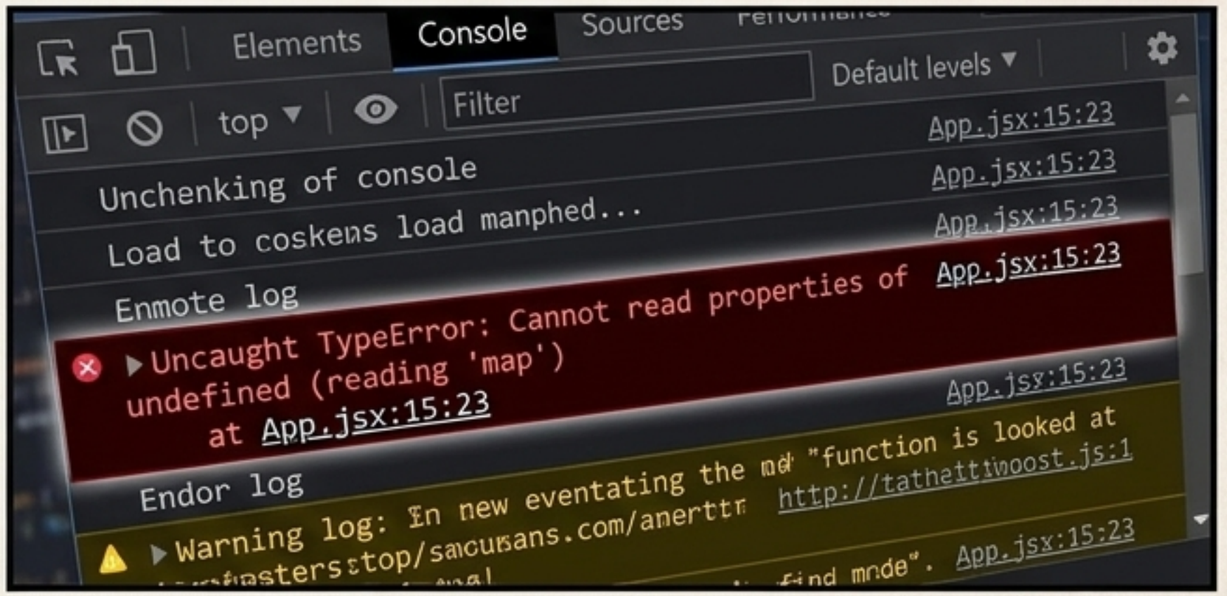


画面が真っ白に!?エラー文はそのまま「丸投げ」せよ









Claudeも陥る罠:「見えないコード」には事前制約を

エラーが出ないのに画面に図形が表示されない原因の多くは、
論理エラーではなく「スタイリングの罠」。

透明なSVG:
デフォルトが
fill:none;
stroke:none;



非表示CSS:
visibility:hidden;
が残存



これらを防ぐ最大のテクニックは、プロンプトに『事前制約』を仕込むことだ。

ID消失:
最適化で重要IDが消える
cleanupIds: false

【注意点と事前制約】
SVG要素はデフォルトの塗りを適用し、cleanupIds: falseを設定してください。

複数ファイルにまたがる手ごわいバグには「Claude Code」



思考力とリソースを制御する「エフォートレベル」

より多くのファイルを
読み込む
（Deep Context）

何度もツールを
呼び出す
（Tool Usage）



自らテストを実行し、
二重チェックを行う
（Self-Check）



複雑なバグには
`/effort high`だ。
自分でテストを回し、
パスするまで粘り強く
修正してくれる。

完了後は`/usage`コマンドで消費リソースを確認し、`/exit`で終了。

モデル知能とエフォートの相関マトリクス

Max

エフォートレベル
(Effort)

Low

空回り
一生懸命考えるが、知識不足で
解決できずリソースを浪費。

最適解
少ないリソースで瞬時に
正確な解決策を提示。

Haiku

モデルの知能 (Knowledge)

Sonnet / Opus



『エフォート（努力）』と『モデル（知能）』を取り違えるな。
解決できない時は、モデルを上位に変更し、必ず `/effort auto` でリセットしろ。

見えないバグは「監視UI」を作らせて可視化する



AIが自律的に精度を高める反復ループ（Closed Loop）

1. Execute (実行)：
スクリプトを走らせる

2. Measure (測定)：
エラー率や気泡の
混入度合いを自ら評価

3. Adjust (調整)：
閾値や感度パラメータ
を自動で微調整

4. Repeat (反復)：
最適化されるまで
ループを回す

AI

Prompt

問題が検出された場合はパラメータを自動調整し、精度が最適化されるまでClosed Loopで反復してください。



最大のコツ：魔法の箱ではなく「優秀な同僚」として扱う

魔法の箱（NG）



- 一発で完璧なコードを出力させようとする
- 1回エラーが出ただけで『使えない』と切り捨てる

優秀な同僚（OK）



- 試行→測定→修正の『超高速な反復プロセス』を前提とする
- 『ここは動いたけど、ここの表示が崩れているよ』と対話でフィードバックする

AIの本質的な強みは、人間とは比較にならない超高速の試行錯誤（Iterative Process）にある。

今日、このプロンプトで「デバッグからの解放」を体験しよう

過去に保存したままのエラーでも構いません。
今日1つだけエラーを選び、以下のプロンプトをそのまま貼り付けてみてください。

このエラーが発生して画面が正しく
動作せず困っています。コード全体の
ロジックを追跡して原因を特定し、
修正したコードを作成してください。

【エラー内容】：（ここにエラーを貼り付け）



圧倒的なデバッグスピードを、
今すぐあなたの手元に。