Engineer

# Playwright

テストでブラウザーを操作するツール

## どのようなものか

Playwrightは、Microsoftによるウェブブラウザー自動操作用のオープンソースのフレームワークです。Chromium、Firefox、WebKitを、一つのプログラミングインターフェースで扱います。その上に構築したテストランナーがPlaywright Testで、npmでは `@playwright/test` として公開されています。大半の人がPlaywrightという名前で指すのは、このランナーです。同じプロジェクトは、テスト以外のスクリプト用のライブラリー、コマンドラインツール、AIエージェント向けのMCPサーバー、Visual Studio Codeの拡張機能も、すべてApache License 2.0で提供しています。

JavaScriptとTypeScript以外でも、同じブラウザー操作が使えます。Pythonではプロジェクトがpytestプラグインを勧め、JavaではJUnitなど任意のテストフレームワークを使えます。.NET向けには、MSTest、NUnit、xUnit用の基底クラスがあります。

`npx playwright install` は、三つのエンジンについて専用のビルドをダウンロードします。リリースごとにエンジンの版が進むため、アップグレード後には、コマンドを再実行する必要があるかもしれません。すでにマシンへ入っているGoogle ChromeやMicrosoft Edgeも操作できます。既定では、ウィンドウを表示しないヘッドレスモードでテストします。インストールガイドの例は、設定したブラウザープロジェクトを使い、三つのエンジンで並行してテストを実行します。これらのプロジェクトがない場合、テストランナーの既定ブラウザーはChromiumです。各テストには、新しいブラウザープロファイルに相当する、新しいブラウザーコンテキストを用意します。そのため、あるテストのCookieやストレージは、次のテストへ引き継がれません。

テストは、人が行う手順のように読めます。

```
import { test, expect } from "@playwright/test";

test("signing in opens the inbox", async ({ page }) => {
    await page.goto("https://example.com/sign-in");
    await page.getByLabel("Email").fill("reader@example.com");
    await page.getByRole("button", { name: "Sign in" }).click();
    await expect(page.getByRole("heading", { name: "Inbox" })).toBeVisible();
});
```

中心となる仕組みは二つあります。Playwrightは操作前に、その操作が可能か検査し、すべての条件を満たすまで待ちます。クリックなら、ロケーターが指す要素は、厳密に一つでなければなりません。その要素は、見えていて、安定している必要があります。安定とは、連続する少なくとも二つのアニメーションフレームで、境界ボックスが変わらないことです。さらに、別の要素の下に隠れずクリックを受け取れ、有効な状態である必要があります。制限時間内に条件を満たさなければ、操作は `TimeoutError` で終わります。 `await expect(locator)` と書くアサーションも、条件を満たすか、既定で五秒のアサーションの待機時間を超えるまで、要素を繰り返し取得します。

概念を判断に生かす

「Playwrightとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 3社の間で受け渡す代わりに、1つのチームがアイデアのスケッチから、お客様がログインして使える製品まで担当します。

**サービスを見る** お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 GIS地図アプリの開発を主導し売上を10倍に — 10x 地図製品の開発を主導して実現した売上成長

**GIS地図アプリの開発を主導し売上を10倍に** お問い合わせ →

## なぜ重要なのか

Playwrightを使うときは、どの動作を実ブラウザーで検査すべきか、そしてビルド失敗の報告を信頼できるように、どう検査を作るかを決めます。ページが組み上がってからしか現れない欠陥があります。本来見せるはずのボタンを隠すダイアログ、ある画面幅で入力欄を隠すレイアウト、あるエンジンでは動き、別のエンジンでは例外を出すスクリプトです。コードの一部分を単独で動かすテストには、これらは見えません。

テストとページの処理が競合すると、ブラウザーテストの結果は不安定になります。ページはデータを取得し、アニメーションを動かし、再描画します。該当行が実行された途端にクリックするテストは、時には早過ぎます。従来は固定時間だけ待つ方法が一般的でしたが、遅いマシンでは短過ぎ、速いマシンでは時間を

無駄にします。自動待機と再試行するアサーションは、その待ち時間を、テストが実際に必要とする条件へ置き換えます。アサーションのガイドは、ウェブページは非同期に情報を表示するので、一度しか検査しないアサーションは、不安定なテストにつながると明記しています。

次に決めるのは、テストが要素を探す方法です。ロケーターのガイドは、まずユーザーに分かる属性を使い、特にロールによる検索を勧めます。アクセシブルな名前を指定した `getByRole` は、「ユーザーと支援技術がページをどう認識するか」に最も近いからです。これには、副次的な利点もあります。「Sign in」という名前のボタンをクリックするテストは、そのボタンが名前を失うと失敗します。スクリーンリーダーのユーザーも、同じ欠陥に直面します。一方で、ロールによる検索は「アクセシビリティー監査や適合性テストを置き換えない」と、ガイドは説明しています。テストIDは代替策です。ガイドはこれを最も変更に強い方法としつつ、「ユーザー向けではない」とも述べています。すべてをIDで見つけるテストでは、訪問者の体験について確認できることが少なくなります。DOMは頻繁に変わるため、CSSやXPathのセレクターは推奨されていません。

代償は時間です。ブラウザーを起動してページを読み込むため、ブラウザーテストは、プロジェクト内で最も遅い種類のテストになります。ビルドサーバーで失敗したときは、トレースビューアーから調べられます。文書にある設定例では、失敗したテストの最初の再試行でトレースを記録し、再試行回数をCIでは二回、ローカルでは零回に設定しています。テストランナーの組み込みの既定値では、トレースの記録も再試行も行いません。トレースでは、各操作を順にたどり、操作の前後のページ、ネットワーク要求、コンソールを確認できます。

## 実際の活用例

あるプラットフォームでは、業務ロジックをデータベース内に置いており、四つの層にまたがるテスト一式を構築しました。記録された課題は、「すべてを外側からブラウザー経由で検査するのではなく」、各層の動作を、実際にその動作が存在する場所で検査することでした。Go APIには981個のテスト関数があります。データベースには494個の振る舞いを検証するテストファイルがあり、ストアド関数を、その上の三層のアプリケーション経由ではなく、データベース内で検査します。フロントエンドには、VitestとPlaywrightを合わせて543件のテスト仕様があり、その中に48個のエンドツーエンド用ファイルと30件のブラウザーテスト仕様を含みます。このページの件数は、二つのツールを合わせたものです。全体を612行のワークフローで10ジョブとして実行します。ページは代償も明記しています。このテスト一式は遅く、変更のたびに負担がかかり、この規模では、テストそのものにも保守が必要です。

このサイトのブラウザー検査は、別のライブラリーPuppeteerで動かしており、クレジットページにも同じ利点と負担が示されています。検査するのは、実測コントラスト、フォーカス順、操作対象のサイズ、表示領域からのはみ出し、印刷時の改ページなど、バリデーターでは確認できないことです。品質ゲートは、表示領域からのはみ出し以外を、毎回のプッシュ前にすべて実行します。はみ出し検査は、ピクセル単位の視覚的回帰検査と同様、人が必要に応じて実行します。

## 問題が起きるところ

Playwrightの問題の多くは、名前から期待するほどのことを、テストが検査していないために起こります。

**条件の代わりに、固定時間だけ待ちます。** 二秒待ってからページを読んだり、一度だけ値を読んで一般的なアサーションで比較したりします。混雑したビルドサーバーでは、二秒では足りないことがあります。`toHaveText` のようなウェブ向けのアサーションで、訪問者に見えるはずの状態を検査し、再試行に任せます。

**マークアップで要素を探します。** 三重に入れ子になった `div` や、装飾用クラスをたどるセレクターは、ページが動いていても、次のデザイン変更で壊れます。ロールとアクセシブルな名前、またはラベルやテキストで要素を探します。ユーザーが認識できるものでは何も識別できないときにだけ、テストIDを追加します。

**ロールによる検索の成功を、アクセシビリティー検査と見なします。** `getByRole` が示すのは、指定したロールと名前を持つ要素が一つあることです。コントラスト、フォーカス順、ページの残りの部分については何も分かりません。そのためには、Playwright自身のガイドが `@axe-core/playwright` 経由で行うように、axeなどのルールエンジンをテストから実行します。検査対象は「呼び出したときの、ページの現在の状態」なので、先にメニューやダイアログを開きます。

**クリックを強制します。** `force` は、操作可能性の検査のうち、必須でないものを無効にします。クリックなら、対象が本当にイベントを受け取るかを確認しなくなります。この指定が必要なテストは、ボタンを覆うバナーなど、訪問者も出会う問題を回避していることがよくあります。元のエラーを発見事項として扱い、訪問者と同じ方法でバナーを閉じるか、重なりを修正します。

**不安定なテストを、成功するまで再試行します。** 再試行を有効にした場合、初回に失敗して再試行で成功したテストは、失敗ではなく不安定と分類されます。毎週のように不安定なら、ページの処理と競合しているか、本物の間欠的な欠陥

を検出しています。最初の再試行で記録したトレースから調べ始めます。

**すべてをブラウザー経由で検査します。** 価格や権限の規則を、三つの画面をクリックしてたどって検査すると、遅くなり、規則と関係ない理由で失敗します。規則は、それが実装されている場所で検査します。ブラウザーテストは、操作の流れ、レイアウト、エンジン間の違いなど、ブラウザーでしか分からないことに使います。

## 参考資料

- インストール — Playwright Testの導入、サンプルテストの実行、HTMLレポート、UI Modeです。
- ロケーター — ロールやラベルからテストIDまで、要素を探すときの推奨順序です。
- 自動待機 — 各操作前の操作可能性の検査と、`force` で無効になるものです。
- アサーション — 再試行するものとしないもの、既定の待機時間、失敗してもテストを続けるソフトアサーション、ポーリングです。
- トレースビューアー — 再試行時のトレース記録と、失敗したテストを操作ごとにたどる方法です。
- ブラウザー — エンジンのインストール、製品版ChromeとEdgeの実行、二つのヘッドレスモードです。



用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

**料金を見る** お問い合わせ →

# 用語を実務に結び付ける

「Playwrightとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/playwright/