Playwright 技術仕様書

1. はじめに

Playwrightは、Microsoftが開発したモダンなWebアプリケーション向けの信頼性の高いエンドツーエンドテストフレームワークです。クロスブラウザ、クロスプラットフォーム、クロス言語に対応しており、Webテスト自動化を効率的に行うための強力な機能を提供します。

2. 主な機能と特徴

2.1. クロスブラウザ対応

Playwrightは、Chromium、WebKit、Firefoxを含むすべてのモダンなレンダリングエンジンをサポートしています。これにより、異なるブラウザ環境でのテストを容易に行うことができます。

2.2. クロスプラットフォーム対応

Windows、Linux、macOSといった主要なオペレーティングシステム上でテストを実行できます。ローカル環境だけでなく、CI/CD環境でもヘッドレスモードまたはヘッドモードで実行可能です。

2.3. クロス言語対応

Playwright APIは、TypeScript、JavaScript、Python、.NET、Javaなど、複数のプログラミング言語で利用できます。

2.4. モバイルWebテスト

Google Chrome for AndroidおよびMobile Safariのネイティブモバイルエミュレーションをサポートしています。デスクトップとクラウドの両方で同じレンダリングエンジンが動作します。

2.5. 堅牢性と安定性

2.5.1. 自動待機 (Auto-wait)

Playwrightは、アクションを実行する前に要素が操作可能になるまで自動的に待機します。これにより、テストの不安定性の主な原因である人工的なタイムアウトの必要性を排除します。

2.5.2. Webファーストアサーション (Web-first assertions)

Playwrightのアサーションは、動的なWebのために特別に作成されており、必要な条件が満たされるまで自動的にリトライされます。

2.5.3. トレース (Tracing)

テストの再試行戦略を設定し、実行トレース、ビデオ、スクリーンショットをキャプチャすることで、テストの不安定性を排除します。

2.6. 制限のないテストシナリオ

Playwrightは、モダンなブラウザのアーキテクチャに準拠しており、テストをプロセス外で実行します。これにより、一般的なプロセス内テストランナーの制限から解放されます。

2.6.1. 複数タブ、複数オリジン、複数ユーザー

複数のタブ、複数のオリジン、複数のユーザーにまたがるテストシナリオをサポートします。異なるユーザーに対して異なるコンテキストを持つシナリオを作成し、それらを単一のテストでサーバーに対して実行できます。

2.6.2. 信頼性の高いイベント (Trusted events)

要素のホバー、動的なコントロールとのインタラクション、信頼性の高いイベントの生成が可能です。Playwrightは、実際のユーザーと区別できない実際のブラウザ入力パイプラインを使用します。

2.6.3. フレームとShadow DOM

PlaywrightのセレクターはShadow DOMを透過し、フレームにシームレスに入ることができます。

2.7. 完全な分離と高速な実行

2.7.1. ブラウザコンテキスト (Browser contexts)

Playwrightは、各テストに対してブラウザコンテキストを作成します。ブラウザコンテキストは、新しいブラウザプロファイルと同等であり、オーバーヘッドなしで完全なテスト分離を実現します。新しいブラウザコンテキストの作成は、わずか数ミリ秒で完了します。

2.7.2. ログイン状態の保存

コンテキストの認証状態を保存し、すべてのテストで再利用できます。これにより、各テストでの繰り返しのログイン操作を回避しつつ、独立したテストの完全な分離を実現します。

3. 強力なツール群

3.1. Codegen

ユーザーのアクションを記録することでテストを生成します。生成されたテストは、任意の言語で保存できます。

3.2. Playwrightインスペクター (Playwright inspector)

ページの検査、セレクターの生成、テスト実行のステップ実行、クリックポイントの確認、実行ログの探索が可能です。

3.3. トレースビューアー (Trace Viewer)

テストの失敗を調査するためのすべての情報をキャプチャします。Playwrightトレースには、テスト実行のスクリーンキャスト、ライブDOMスナップショット、アクションエクスプローラー、テストソースなどが含まれます。

4. アーキテクチャ

Playwrightは、クライアントとサーバー間のすべてのリクエストを単一のWebSocket接続で通信します。Chrome DevTools Protocol (CDP) を使用してChromiumと通信し、

WebKitとFirefoxに対しては「ネイティブアダプテーション」戦略を採用し、詳細なカスタマイズを施しています。

5. ベストプラクティス

（このセクションは、Playwrightの公式ドキュメントやコミュニティの情報をさらに調査して追記します。）

6. API仕様

（このセクションは、Playwrightの公式APIドキュメントを参考に追記します。）

5.1. ロケーターの使用

Playwrightの組み込みロケーターを使用し、要素の自動待機と再試行可能性を活用します。ユーザーが目にする属性や明示的な契約を優先し、DOM構造の変更に強いロケーターを使用します。

5.2. Webファーストアサーション

`toBeVisible()`のようなWebファーストアサーションを使用し、期待される条件が満たされるまでPlaywrightが自動的に待機するようにします。これにより、テストの不安定性を排除します。

5.3. デバッグの設定

VS Code拡張機能やPlaywrightインスペクターを使用して、テストのデバッグを効率的に行います。トレースビューアーを活用し、テストの失敗原因を詳細に調査します。

5.4. テストの分離

各テストは完全に独立しており、独自のローカルストレージ、セッションストレージ、データ、クッキーなどを持つようにします。これにより、再現性が向上し、デバッグが容易になり、連鎖的なテストの失敗を防ぎます。

5.5. 外部依存関係の回避

制御できない外部サイトやサードパーティのサーバーへのリンクはテストしないようにします。代わりに、Playwright Network APIを使用して、必要なレスポンスを保証します。

5.6. CIでのテスト実行

CI/CD環境でテストを実行し、自動化されたテストプロセスを構築します。

5.7. 並列実行とシャーディング

テストの実行時間を短縮するために、並列実行とシャーディングを活用します。

6. API仕様

Playwrightは、ブラウザ操作を自動化するための豊富なAPIを提供します。主要なAPIクラスと機能は以下の通りです。

6.1. Playwrightモジュール

`playwright` モジュールは、ブラウザインスタンスを起動または接続するためのメソッドを提供します。`chromium`, `firefox`, `webkit` といったブラウザタイプオブジェクトを通じて、それぞれのブラウザを操作できます。

- `playwright.chromium`: Chromiumブラウザを起動または接続するためのオブジェクト。
- `playwright.firefox`: Firefoxブラウザを起動または接続するためのオブジェクト。
- `playwright.webkit`: WebKitブラウザを起動または接続するためのオブジェクト。
- `playwright.devices`: `browser.newContext()` または `browser.newPage()` で使用するデバイスの辞書を返します。
- `playwright.errors`: Playwrightメソッドがリクエストを完了できない場合にスローされる特定のエラークラスを提供します（例: `TimeoutError`）。
- `playwright.request`: Web APIテストに使用できるAPIを公開します。

- `playwright.selectors`: カスタムセレクターエンジンをインストールするために使用できます。

6.2. Browserクラス

ブラウザインスタンスを表します。新しいページやコンテキストの作成、ブラウザの終了などの操作を提供します。

- `browser.newPage()`: 新しい`Page`オブジェクトを作成します。
- `browser.newContext()`: 新しい`BrowserContext`オブジェクトを作成します。
- `browser.close()`: ブラウザを閉じます。

6.3. Pageクラス

単一のタブまたはウィンドウを表します。ページのナビゲーション、要素の操作、スクリプトの実行など、Webページとのインタラクションのための主要なAPIを提供します。

- `page.goto(url)`: 指定されたURLに移動します。
- `page.click(selector)`: 指定されたセレクターに一致する要素をクリックします。
- `page.fill(selector, value)`: 指定されたセレクターに一致する入力フィールドに値を入力します。
- `page.screenshot(options)`: ページのスクリーンショットを撮ります。
- `page.waitForSelector(selector)`: 指定されたセレクターに一致する要素が表示されるまで待機します。

6.4. Locatorクラス

要素を見つけるための強力なメカニズムを提供します。自動待機と再試行可能性を備えています。

- `page.locator(selector)`: 指定されたセレクターに一致する要素のロケーターを作成します。
- `locator.click()`: ロケーターに一致する要素をクリックします。
- `locator.fill(value)`: ロケーターに一致する入力フィールドに値を入力します。

6.5. Request/Responseクラス

ネットワークリクエストとレスポンスをインターセプト、変更、または監視するためのAPIを提供します。

- `page.route(url, handler)`: 指定されたURLパターンに一致するリクエストをインターセプトします。
- `request.continue()`: リクエストの処理を続行します。
- `request.abort()`: リクエストを中止します。
- `response.json()`: レスポンスボディをJSONとして解析します。

6.6. その他の主要なクラス

- `BrowserContext`: ブラウザのセッションを表し、クッキー、ローカルストレージ、セッションストレージなどを分離します。
- `ElementHandle`: DOM要素への参照を表します。
- `Frame`: iframe内のコンテンツを操作するためのAPIを提供します。
- `Keyboard`: キーボード操作をシミュレートします。
- `Mouse`: マウス操作をシミュレートします。
- `Tracing`: テスト実行のトレースを収集し、デバッグに役立てます。