

# 60分で学ぶE2Eテスト（実践編）

---

# 今日お伝えしたいこと

---

- テスト自動化そのものは難しくありません
- 自動化の最大の目的は、自動化を通じて **あいまいな部分をなくす** ことです
  - あいまいな部分が残っていると、不安定で読みにくいコードになります
- **ユーザー目線で書いた** E2Eテストコードは開発を助ける資産になります

**あいまいさを排除した、ユーザー目線のテストコードの書き方** をお伝えします

# 今日お話しできないこと

---

- 自動化の技術選定をどのように行うか
- 自動化やプログラミングに必要な基礎知識の説明
  - JavaScriptの文法
  - コマンドラインの使い方

スライドのおわりにおまけを付けておきます  
(予稿集のおわりにもあります)

# テストに使うツール

---

## Cypress

デベロッパーフレンドリーなE2Eテストツール

- NodeJSで動作する（=JavaScriptで記述する）
- Chrome/Firefoxに対応
- テストコードの作成やデバッグを楽にする機能がいろいろある

# 準備

---

## NodeJSのインストール

Macの場合は以下

```
$ brew install node
```

# 準備

---

## インストール

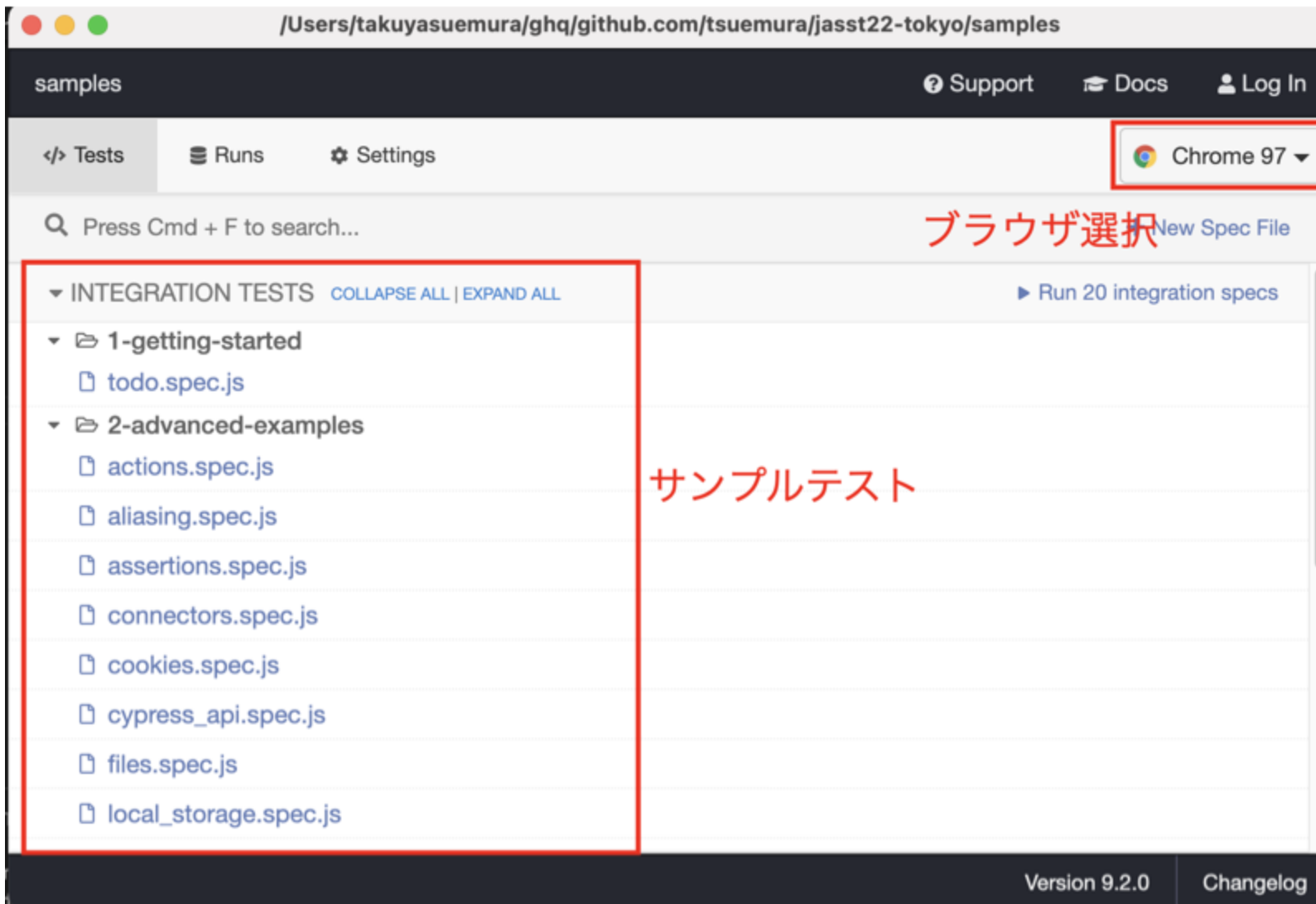
```
$ mkdir jasst22tokyo  
$ cd jasst22tokyo  
$ npm install cypress @testing-library/cypress
```

# 起動

---

```
$ npx cypress open
```

初回起動時に設定ファイルとサンプルのテストコードが生成されます



ブラウザ選択

サンプルテスト



< Tests

✓ 120

✗ --

○ --

74.45

● ↑ ↺

All Specs

▼ with a checked task

✓ can filter for uncompleted tasks

▼ BEFORE EACH (1)

1 visit https://example.cypress.io/todo

▼ BEFORE EACH (2)

1 contains Pay electric bill

2 -parent

3 -find input[type=checkbox]

4 -check

▼ TEST BODY

1 contains Active

2 -click

(new url) https://example.cypress.io/todo#...

3 get .todo-list li

4 -assert expected <li> to have a length of 1

5 -first

6 -assert expected <li> to have text Walk the dog

7 contains Pay electric bill

8 -assert expected :cy-contains('Pay electric bill'), [type='submit'][value~='Pay electric bill'] not to exist in the DOM

✓ can filter for completed tasks

https://example.cypress.io/commands/window 1000 x 660 (75%)

cypress.io

Commands ▾

Utilities

Cypress API

GitHub

Window

Examples of referencing window and other properties on window in Cypress, for a full reference of commands, go to [docs.cypress.io](https://docs.cypress.io)

cy.window()

To get the global window object, use the `cy.window()` command.

```
cy.window().should('have.property', 'top')
```

cy.document()

To get the document object, use the `cy.document()` command.

```
cy.document().should('have.property', 'charset').and('eq', 'UTF-8')
```

cy.title()

テストを実行すると実行結果が細かく表示されます

# 早速書いていきます

---

- **非会員で予約**
- 会員登録→予約→ログアウト
- プレミアム会員でログイン→予約→ログアウト
- 一般会員でログイン→予約→ログアウト
- 一般会員の画面にプレミアム会員限定プランが表示されないこと
- 非会員の画面に一般・プレミアム会員限定プランが表示されないこと

# テスト設計はこんな感じでした

---

- テスト対象のサイトにアクセス
- 宿泊プランを選択
- 宿泊予約
- 予約内容の確認

# テストコードを書いてみよう

---

smoke\_test.js を作成

```
describe('スモークテスト', () => {  
  it('非会員で予約', () => {  
    // ここにテストコードを書いていきます  
  })  
})
```

# テストコードを書いてみよう

---

## テスト対象のサイトにアクセス

```
describe('スモークテスト', () => {  
  it('非会員で予約', () => {  
  
    // テスト対象のサイトにアクセス  
    cy.visit("https://hotel.testplanisphere.dev/ja/index.html");  
  
  })  
})
```

- コマンドは（一部の例外を除き） `cy` から始まる
- `cy.visit()` は指定したURLに移動するコマンド

# テストコードを書いてみよう

---

## 宿泊予約サイトにアクセス

- `宿泊予約` をクリック

Cypressでは `contain()` を使って特定の文字を含む要素を指定できる

```
cy.contain('宿泊予約').click()
```

# テストコードを書いてみよう

---

## 宿泊予約サイトにアクセス

```
describe('スモークテスト', () => {  
  it('非会員で予約', () => {  
  
    // テスト対象のサイトにアクセス  
    cy.visit("https://hotel.testplanisphere.dev/ja/index.html");  
  
    cy.contain('宿泊予約').click()  
  })  
})
```

# 自動化は難しくない

---

テスト手順をそのまま1:1対応でプログラミングすれば、それがテストコード

"https://hotel.testplanisphere.dev/ja/index.html" にアクセスする

↓

```
cy.visit("https://hotel.testplanisphere.dev/ja/index.html");
```

"宿泊予約" をクリックする

↓

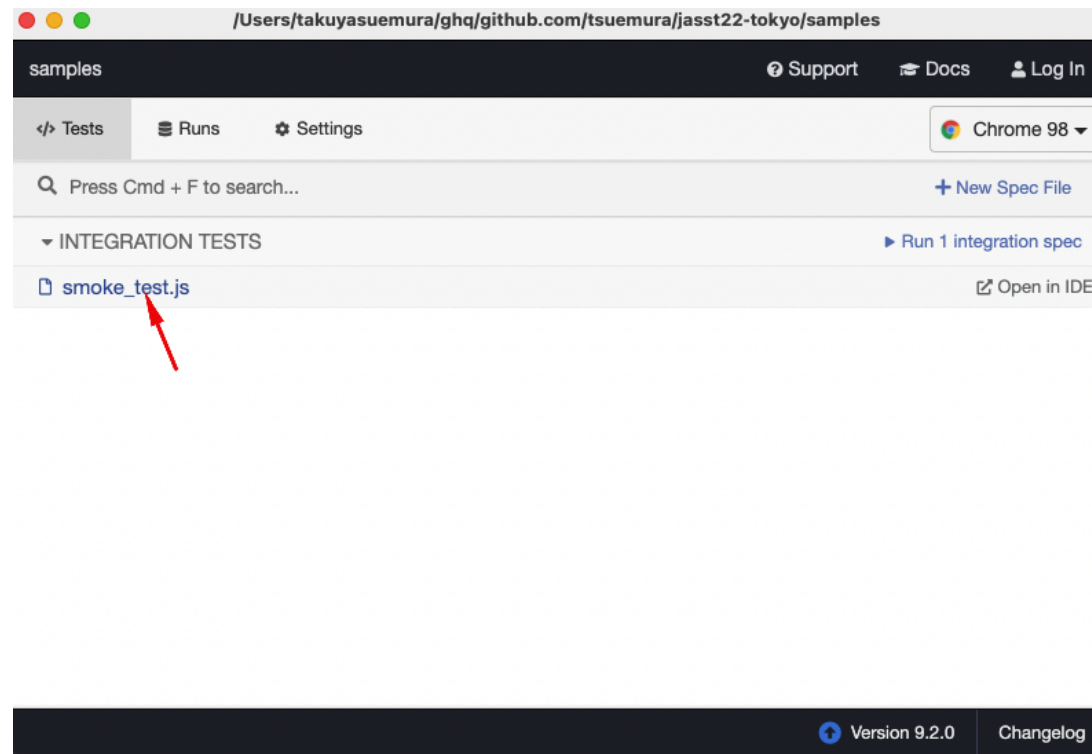
```
cy.contains('宿泊予約').click()
```



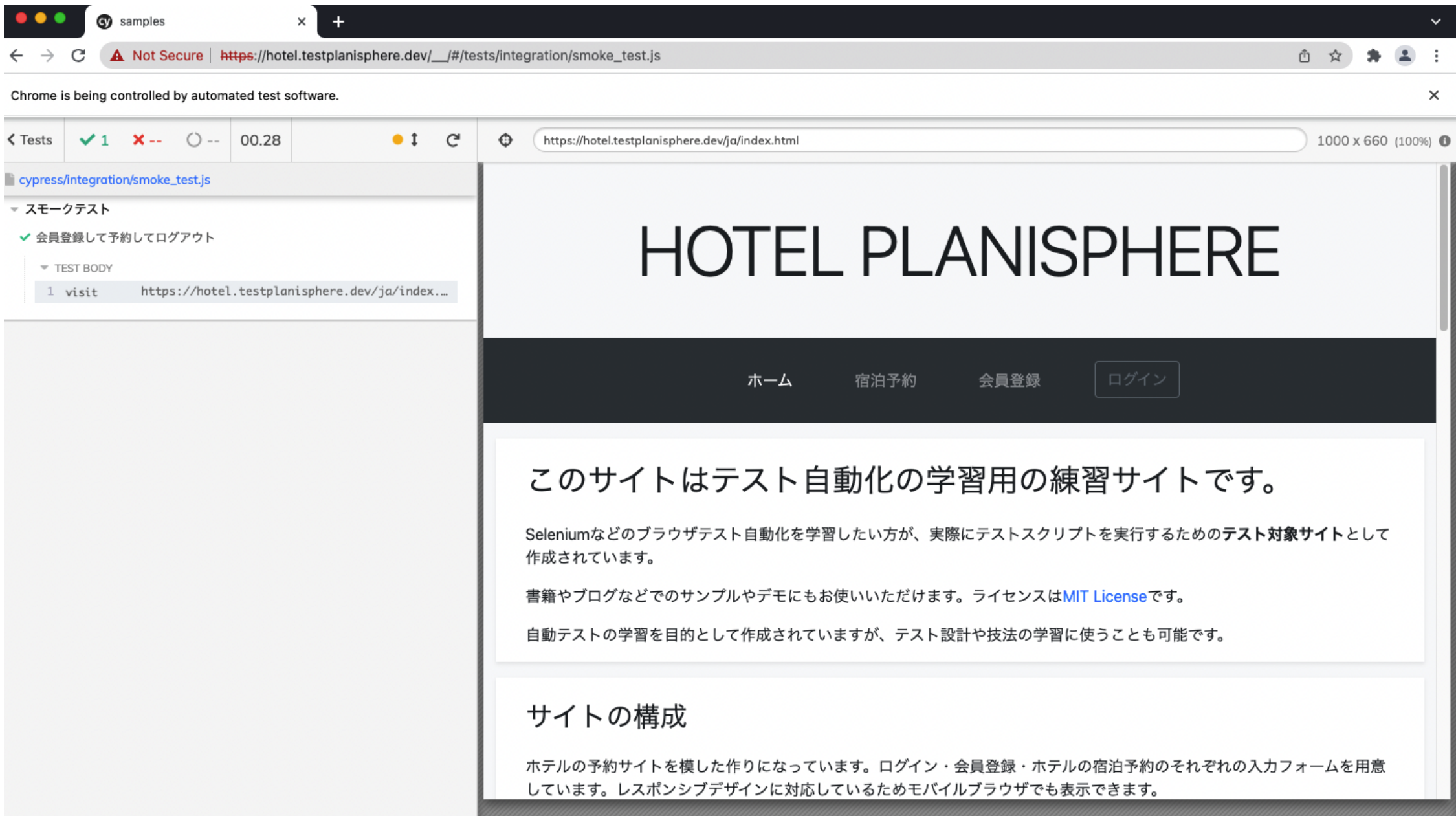
# 実際に動かしてみよう

コマンドラインから以下を実行する

```
$ npx cypress open
```



smoke\_test.jsをクリック



ブラウザが開いて、URLに遷移できた

# 宿泊プランの選択

## HOTEL PLANISPHERE

[ホーム](#)[宿泊予約](#)[会員登録](#)[ログイン](#)

### 宿泊プラン一覧

★おすすめプラン★

お得な特典付きプラン

大人1名7,000円  
1名様から  
スタンダードツイン

[このプランで予約](#)

本日より

素泊まり

大人1名5,500円  
1名様から  
シングル

[このプランで予約](#)

出張ビジネスプラン

大人1名7,500円  
1名様から  
シングル

[このプランで予約](#)

エステ・マッサージプラン

大人1名9,000円  
1名様から  
部屋指定なし

[このプランで予約](#)

貸し切り露天風呂プラン

大人1名9,000円  
1名様から  
部屋指定なし

[このプランで予約](#)

カップル限定プラン

大人1名8,000円  
2名様から  
プレミアムツイン

[このプランで予約](#)

テーマパーク優待プラン

大人1名10,000円  
1名様から  
部屋指定なし

[このプランで予約](#)

複数の宿泊プランから  
「素泊まり」を選択したい

# 込み入った処理を表現する

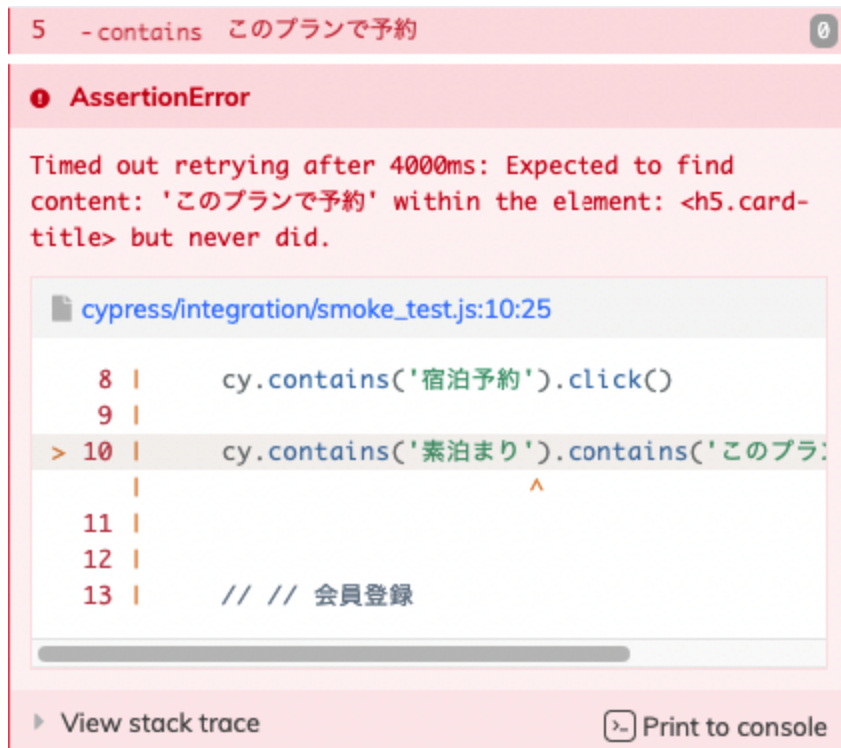
---

素泊まり を含む 宿泊プラン の このプランを選択 をクリックする

```
cy.contains('素泊まり').contains('このプランで予約').click()
```

このコードで動くかな.....? 🤔

# 目当ての要素が見つからない



```
5 - contains このプランで予約 0

❌ AssertionError

Timed out retrying after 4000ms: Expected to find
content: 'このプランで予約' within the element: <h5.card-
title> but never did.

cypress/integration/smoke_test.js:10:25

   8 |     cy.contains('宿泊予約').click()
   9 |
>  10 |     cy.contains('素泊まり').contains('このプランで予約')
      |                                     ^
   11 |
   12 |
   13 |     // // 会員登録

View stack trace Print to console
```

`cy.contains('素泊まり')` が `h5` 要素にマッチしてしまったのが原因

# 探索の範囲を絞り込む

---



## やりたいこと

**素泊まり** というテキストを含む  
**カード** の取得

## 実際

**素泊まり** というテキストを含む  
**見出し** が取得された

# 探索の範囲を絞り込む



```
cy.contains('div.card-body', '素泊まり')  
  .contains('このプランで予約').click()
```

h5 ではなく

card-body というclassを持つ

div 要素を取得するようになった

# 現在のテストコード

---

```
describe('スモークテスト', () => {  
  it('非会員で予約', () => {  
  
    // テスト対象のサイトにアクセス  
    cy.visit("https://hotel.testplanisphere.dev/ja/index.html");  
  
    cy.contains('宿泊予約').click()  
  
    cy.contains('div.card-body', '素泊まり')  
      .contains('このプランで予約').click()  
  })  
})
```



# 考えてみよう

---

## このコードは読みやすい？

```
cy.contains('div.card-body', '素泊まり')  
  .contains('このプランで予約').click()
```

- `div.card-body` なんて、元のテスト設計にあったっけ？
- `div.card-body` がどのUIに対応してるか、後で思い出せる？

## よくない臭いがするぞ！

---

テスト設計に **出てこない言葉** がテストコードに出てきたら、  
テストコードからその箇所を **分離** すべきかも

# カスタムコマンドを追加する

---

`cypress/support/commands.js` に以下を追加する

```
Cypress.Commands.add("getCardByText", (text) => {  
  const selector = 'div.card-body'  
  cy.contains(selector, text)  
});
```

こう書けるようになった

```
// before  
cy.contains('div.card-body', '素泊まり')  
  .contains('このプランで予約').click()  
  
// after  
cy.getCardByText('素泊まり').contains('このプランで予約').click()
```

## さらに別の問題

---

**このプランで予約** は新しいウィンドウを開くが  
Cypressは **複数ウィンドウのテストに対応していない**

# 新しいウィンドウを開かないようにする

---

```
cy.getCardByText('素泊まり')  
  .contains('このプランで予約')  
  .invoke('removeAttr', 'target')
```

リンクから「新しいウィンドウを開く」ための指定 `target="_blank"` を除く

参考: <https://testersdock.com/cypress-new-window/>

# 新たなカスタムコマンドを定義しよう

## 予約プランを開く カスタムコマンドを定義する

```
Cypress.Commands.add("openReservationPlan", (planName) => {  
  const buttonText = "このプランで予約"  
  cy  
    .getCardByText(planName)  
    .contains(buttonText)  
    .invoke("removeAttr", "target")  
    .click()  
})
```

## テストコードはこう書ける

```
// before  
cy.getCardByText('素泊まり').contains('このプランで予約').click()  
  
// after  
cy.openReservationPlan('素泊まり')
```

# なんかめんどくさいね？

---

E2Eテストを書くこと自体は簡単ですが

- ツールの技術的制約の回避
- テストしづらいコンポーネントの操作

などはやっぱりめんどくさい（そしてどうしようもない）

# なんでわざわざ Custom Command とか使うの？

---

テストスクリプトから **ユーザー操作と無関係な部分** を切り離す

- 可読性の向上
- ポータビリティの向上

めんどくさい部分はどうしても出てくるので  
そこを上手く隠せると読みやすいコードになる



# 続けて書いていきましょう

---

- テスト対象のサイトにアクセス
- 宿泊プランを選択
- **宿泊予約**
- **予約内容の確認**

# その前に

---

フォームに対する操作の場合、 `contains` では上手く動かない場合がある

- 普通の入力フォームであればOK
- セレクトボックスやチェックボックスはNG

`contains` で取得できる要素は厳密には `label` 要素なので上手く取得できないかも

# 宿泊予約

---

```
cy.getByLabel('宿泊日').type('2022-02-12')
cy.getByLabel('宿泊数').type('7')
cy.getByLabel('人数').type('1')
cy.getByLabel('朝食バイキング').check()
cy.getByLabel('氏名').type('ジャスト 太郎')
cy.getByLabel('確認のご連絡').select('希望しない')
cy.contains('予約内容を確認する').click()
```

# 上手く行かなかった

宿泊日 必須

2022/02/1220220212✓

**2022年 2月**

日	月	火	水	木	金	土
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28					

閉じる

泊

人

- 元々入力されているテキストに追記してしまった
- カレンダーウィジェットが表示されたまま

# 対処

---

```
// 「宿泊日」フィールドに入っている値を一度全て消す  
cy.getByLabel('宿泊日').type('{selectall}{backspace}')
```

```
// 入力の後に ESC キーを押下してカレンダーウィジェットを消す  
cy.getByLabel('宿泊日').type('2022/02/12{esc}')
```

# これもカスタムコマンドにしてしまえ

---

値を一度削除してから入力する `fill` メソッドを定義する

```
Cypress.Commands.add("fill", { prevSubject: 'element' }, (subject, text) => {  
  subject.type("{selectall}{backspace}");  
  subject(text)  
})
```

テストコードはこうなる

```
cy.getByLabel('宿泊日').fill('2022/02/21{esc}')
```

# 予約内容の確認

---

TBD

# いかがでしたか

---

- Cypressは拡張性が高く、テストコードをきれいに記述するのに十分な機能を備えています
- 反面、複数ウィンドウを利用するサイトのテストなど、対応していないサイトのテストにはコツが要ります
- まずは触ってみて、自分のプロジェクトに適用可能か確かめてみましょう



# おさらい: わかりやすいテストコードを書くコツ

---

## ユーザー目線の表記を心がける

サイトの内部構造を使わず、表示されたテキストで選択する

## あいまいな部分を減らす

「xxの中のyy」というように指定して、要素探索の範囲を絞り込む

## 「何をテストしているのか」と「どうテストするのか」を分ける

カスタムコマンドを上手く使って、  
処理やUIコンポーネントに名前を付ける

おまけ

# 利用する技術の選定

---

いくつか軸があります

# 利用する技術の選定

---

クロスブラウザ・クロスデバイスはどのぐらい必要？

やるやら	フレームワーク
いない	Cypress
いる	PlayWright
超いる	Selenium

## 参考: クロスブラウザ・クロスデバイスについて

- 「いろんなブラウザでテスト出来たほうがいいじゃん」と思ってしまうのが人の常
- 実際は自動テストを使おうが使わまいがクロスブラウザ対応はしんどい
  - IE対応が辛いのはテストが辛いからじゃない、IEが辛いからだ
  - **おまえは本当にIEについて良く理解しながらIEに対応しているのか？**
- クロスブラウザテストできる ≠ クロスブラウザテストが **期待通り** 動く
  - 自動化レイヤーのバグを踏むことも多い
- nice to have でクロスブラウザに手を出さないこと
  - お兄さんとの約束だよ

# 利用する技術の選定

---

## テストコードを読むのは開発者だけか？

- 開発者だけ → 普通のテストツールで良い
- チームみんなで読む → BDD系のツールを使う

## 参考: BDD (Behavior Driven Development)

- 自然言語でプロダクトの **振る舞い** を記述するためのツール
  - 独自の **自然言語っぽい** 記述を使うもの
  - Markdown を使うもの
- チーム全体でプロダクトの振る舞いを考えて、それを自動化するためのもの
- ノーコードのツールとは違う
  - プログラミング不要でテスト自動化できるツールは志向していない
- 今日はこの話はしません
  - が、ここはここで **かなりアツい** 領域.....
  - この話がしたいひとは後で声かけてください

今回は

**クロスブラウザは割り切る** し、

**テストコードは開発者がメンテする** ということで、

**Cypress** にします