

Swift Testing による エラーのテスト



宇佐見公輔

2024-10-24

株式会社ゆめみ

自己紹介

- 宇佐見公輔
 - ▶ 株式会社ゆめみ iOS テックリード
- 最近のアウトプット
 - ▶ Swift Testing を活用する (Mobile Act OSAKA 14)
 - ▶ XCTest から Swift Testing へ (関モバ A #5)
 - ▶ 3次元回転とクォータニオン (iOSDC Japan 2024)

Swift Testing の技術同人誌

- Swift Testing の技術同人誌を制作中
- 11 月 2 日（土）からの技術書典 17 に出展予定
 - （今回はオンラインのみの出展）

Swift Testing とは

Swift Testing とは

Swift 用の単体テストフレームワーク

- Swift 公式の GitHub リポジトリで公開されている
 - <https://github.com/swiftlang/swift-testing>
- Xcode 16 に統合された
- XCTest との関係
 - 従来の XCTest に比べて、Swift の機能をより活用
 - XCTest と同じプロジェクトで混在可能

Swift Testing の構成要素

- テスト関数
 - `@Test` 属性
- 期待値の確認
 - `#expect` マクロ
- テストスイート
 - `@Suite` 属性
- トレイト
 - `TestTrait` / `SuiteTrait`

テスト関数

テスト関数定義：XCTest

```
import XCTest

class FoodTruckTests: XCTestCase {
    func testEngineWorks() {
        // ...
    }
}
```

- XCTestCase のサブクラス内で定義する必要がある
- メソッド名を test 始まりで命名する必要がある

テスト関数定義：Swift Testing

```
import Testing

struct FoodTruckTests {
    @Test
    func engineWorks() {
        // ...
    }
}
```

- テスト関数はどこで定義してもよい
- メソッドに `@Test` 属性をつければテスト関数になる

期待値の確認


期待値の確認

```
// XCTest
func testEngineWorks() throws {
    XCTAssertNotNil(engine.parts.first)
    XCTAssertGreaterThan(engine.batteryLevel, 0)
    XCTAssertTrue(engine.isRunning)
}

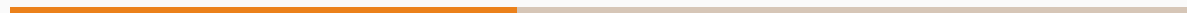
// Swift Testing
@Test func engineWorks() throws {
    try #require(engine.parts.first != nil)
    #expect(engine.batteryLevel > 0)
    #expect(engine.isRunning)
}
```

Xcode 上の表示

#expect マクロはテスト失敗時の結果をきれいに表示する

```
@Test func videoMetadata() {  
9     let expectedMetadata = Metadata(duration: .seconds(90))  
10    #expect(video.metadata == expectedMetadata)  Expectation failed: (video.metadata → duration: 0.0...  
  
    Results  
    ▾ video.metadata:duration: 0.0 seconds, resolution: 3840.0 × 2160.0  
      Metadata  
      > duration: 0.0 seconds  
        Duration  
      > resolution: 3840.0 × 2160.0  
        Resolution  
  
    ▾ expectedMetadata:duration: 90.0 seconds, resolution: 3840.0 × 2160.0  
      Metadata  
      > duration: 90.0 seconds  
        Duration  
      > resolution: 3840.0 × 2160.0  
        Resolution  
  
11 }  
}
```

テストスイート



テストスイート

```
struct FoodTruckTests {  
    @Test  
    func engineWorks() {  
        // ...  
    }  
}
```

- テスト関数を含む型が、自動的にテストスイートになる
- 後述するトレイトを使う場合は `@Suite` 属性を指定する

Swift の機能の活用

```
final class FoodTruckTests {  
    init() async throws {  
        // ...  
    }  
  
    deinit {  
        // ...  
    }  
}
```

- 専用の `setUp` の代わりに、通常の `init` が使える
- `actor` や `@MainActor` などにも使える

トレイト



テスト関数やテストスイートの振る舞いを指定する

```
@Suite(.timeLimit(.minutes(1)))  
struct FoodTruckTests {  
    @Test  
    func engineWorks() {  
        // ...  
    }  
}
```

- テストスイートのタイムアウト時間を指定する例
- ちなみに `TimeLimitTrait.Duration` は `.minutes` 指定のみ

用意されているトレイト

- `.enabled / .disabled`
- `.timeLimit`
- `.serialized`
- `.tags`
- `.bug`
- `.isRecursive`

Swift Testing の機能

Swift Testing の機能

Swift Testing ならではの機能

- エラーのテスト
- パラメトライズテスト
- テストの並列実行

今回は、エラーのテストについて紹介

エラーのテスト

エラーのテスト：XCTest

- 素朴に `do ~ catch` でテストを書いた場合

```
func testExample() throws {  
    let myModel = MyModel()  
    do {  
        try myModel.doSomething()  
        XCTFail("Expect to throw error")  
    } catch {  
        XCTAssertEqual(error as? MyError, .someError)  
    }  
}
```

エラーのテスト：XCTest

- XCTAssertThrowsError を使うとより安全に書ける

```
func testExample2() throws {  
    let myModel = MyModel()  
    XCTAssertThrowsError(  
        try myModel.doSomething()  
    ) { error in  
        XCTAssertEqual(error as? MyError, .someError)  
    }  
}
```

エラーのテスト : Swift Testing

- #expect マクロで書ける

```
@Test func example() throws {  
    let myModel = MyModel()  
    #expect(throws: MyError.someError) {  
        try myModel.doSomething()  
    }  
}
```


エラーのテスト : Swift Testing

- 特定の型のエラーであるかを確認

```
#expect(throws: MyError.self) {  
    ...  
}
```

- 特定のエラーインスタンスと一致するかを確認
 - ▶ この場合、エラー型が `Equatable` であることが必要

```
#expect(throws: MyError.someError) {  
    ...  
}
```

エラーのテストのカスタマイズ

- エラー判定をカスタマイズしたい場合
 - ▶ エラーが `Equatable` でない場合などに有益

```
@Test func example2() throws {  
    let myModel = MyModel()  
    #expect {  
        try myModel.doSomething()  
    } throws: { error in  
        return error as? MyError == .someError  
    }  
}
```

エラーが発生しないことのテスト

- 単にコードをテスト関数の中に書けばよい

```
@Test func example3() throws {  
    let myModel = MyModel()  
    try myModel.doSomething2()  
}
```

- #expect(throws: Never.self) という書き方も可能
 - ▶ エラーが throw されても処理を続行したい場合

まとめ



- Swift Testing の構成要素
 - テスト関数、期待値の確認、テストスイート、トレイト
- Swift Testing の機能として、エラーのテストを紹介
 - 型やインスタンスの一致を確認
 - 確認処理のカスタマイズ