【新人エンジニア向け】Vue.jsテスト完全入門ガイド

はじめに

Vue.jsでアプリケーション開発を始めた新人エンジニアの皆さん、テストコードの書き方で悩んでいませんか？この記事では、Vue.jsテストの基礎から実践まで、分かりやすく解説します。

1. Vue.jsテストの基礎知識

なぜテストが必要なのか？

- コードの品質向上
- バグの早期発見
- リファクタリングの安全性確保
- チーム開発での信頼性向上

主要なテストツール

- **Jest**: JavaScriptテストフレームワーク
- **Vitest**: Vite環境に最適化されたテストツール
- **Vue Test Utils**: Vue.js専用のテストライブラリ

2. プロジェクト環境別のテスト実行方法

Vue CLIプロジェクトの場合

```bash
# 全てのテストを実行
npm run test:unit

# 特定のファイルをテスト
npm run test:unit -- MyComponent.test.js

# ウォッチモードで実行
npm run test:unit -- --watch
```

Vite + Vitestプロジェクトの場合

```bash
# 全てのテストを実行
npm run test

# 特定のファイルをテスト
npm run test MyComponent.test.js

# ウォッチモードで実行
npm run test -- --watch
```

Jestを直接使用する場合

```bash
# 特定のファイルをテスト
npx jest MyComponent.test.js

# パターンマッチングでテスト
npx jest --testNamePattern="ボタンクリック"
```

3. テストファイルの命名規則と配置

推奨される命名パターン

```
src/
├── components/
│   ├── MyComponent.vue
│   └── __tests__/
│       └── MyComponent.test.js
```

または

```
src/
├── components/
│   ├── MyComponent.vue
│   └── MyComponent.spec.js
```

命名規則

- `コンポーネント名.test.js`
- `コンポーネント名.spec.js`
- `__tests__/コンポーネント名.js`

4. 基本的なテスト構文

describe() - テストスイート

```javascript
describe('MyComponent', () => {
  // 関連するテストケースをまとめる
})
```

test() / it() - テストケース

```javascript
test('ボタンをクリックしたときにカウンターが増加する', () => {
  // テストの内容
})

// it()も同じ意味で使える
it('should increment counter when button is clicked', () => {
  // テストの内容
})
```

expect() - アサーション（期待値の検証）

```javascript
expect(実際の値).toBe(期待する値)          // 完全一致
expect(実際の値).toEqual(期待する値)       // オブジェクトの内容比較
expect(wrapper.exists()).toBe(true)       // 存在確認
expect(wrapper.text()).toContain('Hello')  // 文字列を含むかチェック
```

5. Vue Test Utilsの基本的な使い方

コンポーネントのマウント

```javascript
import { mount } from '@vue/test-utils'
import MyComponent from '@/components/MyComponent.vue'

test('コンポーネントが正しくレンダリングされる', () => {
  const wrapper = mount(MyComponent)
  expect(wrapper.exists()).toBe(true)
})
```

プロップスを渡すテスト

```javascript
test('プロップスが正しく表示される', () => {
  const wrapper = mount(MyComponent, {
    props: {
      title: 'テストタイトル',
      count: 5
    }
  })

  expect(wrapper.text()).toContain('テストタイトル')
})
```

要素の検索とイベントのテスト

```javascript
test('ボタンクリックでカウンターが増加する', async () => {
  const wrapper = mount(Counter)

  // ボタンを見つける
  const button = wrapper.find('button')

  // クリックイベントを発火
  await button.trigger('click')

  // 結果を確認
  expect(wrapper.text()).toContain('1')
})
```

6. よく使うラッパーメソッド

```javascript
// 要素の検索
wrapper.find('button')                    // CSSセレクターで検索
wrapper.findAll('.item')                  // 複数要素を検索
wrapper.findComponent(ChildComponent)     // 子コンポーネントを検索

// イベントの発火
wrapper.trigger('click')                  // イベント発火
wrapper.find('input').setValue('text')    // 入力値の設定

// 内容の取得
wrapper.text()                            // テキスト内容取得
wrapper.html()                            // HTML取得
wrapper.vm                                // Vueインスタンスへのアクセス
```

7. 非同期処理のテスト

async/awaitを使用

```javascript
test('非同期処理のテスト', async () => {
  const wrapper = mount(AsyncComponent)

  // 非同期操作を実行
  await wrapper.find('button').trigger('click')

  // Vue.jsの更新を待つ
  await wrapper.vm.$nextTick()

  // 結果を確認
  expect(wrapper.find('.result').text()).toBe('完了')
})
```

8. 実践的なテスト例

TodoListコンポーネントのテスト

```javascript
import { mount } from '@vue/test-utils'
import TodoList from '@/components/TodoList.vue'

describe('TodoList', () => {
  test('新しいTodoを追加できる', async () => {
    const wrapper = mount(TodoList)

    // 入力欄にテキストを入力
    await wrapper.find('input').setValue('新しいタスク')

    // 追加ボタンをクリック
    await wrapper.find('button').trigger('click')

    // Todoが追加されたことを確認
    expect(wrapper.text()).toContain('新しいタスク')
  })

  test('Todoを完了状態にできる', async () => {
    const wrapper = mount(TodoList, {
      data() {
        return {
          todos: [{ id: 1, text: 'テストタスク', completed: false }]
        }
      }
    })

    // チェックボックスをクリック
    await wrapper.find('input[type="checkbox"]').setChecked(true)

    // 完了状態になったことを確認
    expect(wrapper.vm.todos[0].completed).toBe(true)
  })
})
```

9. モック（Mock）の使い方

関数のモック

```javascript
import { vi } from 'vitest' // Vitestの場合
// import { jest } from '@jest/globals' // Jestの場合

test('API呼び出しのモック', async () => {
  // モック関数を作成
  const mockApi = vi.fn().mockResolvedValue({ data: 'テストデータ' })

  const wrapper = mount(ApiComponent, {
    global: {
      provide: {
        api: mockApi
      }
    }
  })

  await wrapper.find('button').trigger('click')

  // モック関数が呼ばれたことを確認
  expect(mockApi).toHaveBeenCalled()
})
```

10. 新人エンジニアがつまずきやすいポイント

1. 非同期処理の待機忘れ

```javascript
// ❌ 悪い例
test('bad example', () => {
  const wrapper = mount(Component)
  wrapper.find('button').trigger('click') // awaitがない
  expect(wrapper.text()).toContain('Updated') // まだ更新されていない
})

// ✅ 良い例
test('good example', async () => {
  const wrapper = mount(Component)
  await wrapper.find('button').trigger('click') // awaitを付ける
  expect(wrapper.text()).toContain('Updated')
})
```

2. セレクターの間違い

```javascript
// ❌ 存在しない要素を検索
wrapper.find('.non-existent-class')

// ✅ 事前に要素の存在確認
const button = wrapper.find('button')
expect(button.exists()).toBe(true)
```

3. プロップスの型違い

```javascript
// ❌ 型が違う
mount(Component, {
  props: {
    count: '5' // 数値を文字列で渡している
  }
})

// ✅ 正しい型
mount(Component, {
  props: {
    count: 5 // 数値で渡す
  }
})
```

11. テスト実行時のTips

package.jsonの設定例

```json
{
  "scripts": {
    "test": "vitest",
    "test:unit": "jest",
    "test:watch": "vitest --watch",
    "test:coverage": "vitest --coverage"
  }
}
```

カバレッジの確認

```bash
# テストカバレッジを確認
npm run test:coverage
```

12. まとめ

Vue.jsのテストは最初は難しく感じるかもしれませんが、基本的な書き方を覚えれば徐々に慣れてきます。重要なポイントは：

1. **小さなテストから始める** - 簡単なコンポーネントのレンダリングテストから
2. **実際のユーザー操作を模倣する** - クリック、入力、フォーム送信など
3. **非同期処理を忘れずに待つ** - async/awaitを適切に使用
4. **テストの目的を明確にする** - 何をテストしたいのかを明確に

継続的にテストを書くことで、より安全で保守性の高いコードが書けるようになります。まずは簡単なコンポーネントから始めて、徐々に複雑なテストにチャレンジしてみましょう！