【新人エンジニア向け】Vue.jsテスト実行結果と正常・異常パターン完全解説

はじめに

テストコードを書いたけれど、実行結果をどう読めばいいかわからない。正常に通るテストと失敗するテストの違いがわからない。そんな新人エンジニアの疑問を解決します！

この記事では、実際のテストコードを1行ずつ解説し、想定される結果パターンを詳しく説明します。

1. テスト対象のVueコンポーネント

まず、テスト対象となるシンプルなカウンターコンポーネントを見てみましょう。

vue

```vue
<!-- Counter.vue -->
<template>
  <div class="counter">
    <h1>{{ title }}</h1>
    <p class="count">Count: {{ count }}</p>
    <button @click="increment" class="increment-btn">+</button>
    <button @click="decrement" class="decrement-btn">-</button>
    <button @click="reset" class="reset-btn">Reset</button>
  </div>
</template>

<script>
export default {
  name: 'Counter',
  props: {
    title: {
      type: String,
      default: 'カウンター'
    },
    initialValue: {
      type: Number,
      default: 0
    }
  },
  data() {
    return {
      count: this.initialValue
    }
  },
  methods: {
    increment() {
      this.count++
    },
    decrement() {
      this.count--
    },
    reset() {
      this.count = this.initialValue
    }
  }
}
</script>
```

2. 正常パターンのテストコード（1行ずつ解説）

テストケース1: 基本的なレンダリングテスト

```javascript
// 必要なライブラリをインポート
import { mount } from '@vue/test-utils'  // Vue Test Utilsからmount関数をインポート
import Counter from '@/components/Counter.vue'  // テスト対象のコンポーネントをインポート

// テストスイートを定義（関連するテストをまとめる）
describe('Counter コンポーネント', () => {

  // 個別のテストケースを定義
  test('デフォルトの状態で正しくレンダリングされる', () => {

    // コンポーネントをDOMにマウント（実際にブラウザ上に表示するような状態にする）
    const wrapper = mount(Counter)

    // コンポーネントが存在することを確認
    expect(wrapper.exists()).toBe(true)  // 結果: ✅ PASS - wrapperが存在するため

    // デフォルトタイトルが表示されていることを確認
    expect(wrapper.find('h1').text()).toBe('カウンター')  // 結果: ✅ PASS - h1にデフォルトタイト

    // 初期カウント値が0であることを確認
    expect(wrapper.find('.count').text()).toBe('Count: 0')  // 結果: ✅ PASS - 初期値は0

    // 各ボタンが存在することを確認
    expect(wrapper.find('.increment-btn').exists()).toBe(true)  // 結果: ✅ PASS - +ボタンが存在
    expect(wrapper.find('.decrement-btn').exists()).toBe(true)  // 結果: ✅ PASS - -ボタンが存在
    expect(wrapper.find('.reset-btn').exists()).toBe(true)      // 結果: ✅ PASS - Resetボタンが
  })
})
```

期待される実行結果：

```
✓ Counter コンポーネント › デフォルトの状態で正しくレンダリングされる (15ms)
```

テストケース2: プロップス渡しのテスト

```javascript
test('プロップスが正しく渡される', () => {

  // カスタムプロップスを指定してコンポーネントをマウント
  const wrapper = mount(Counter, {
    props: {
      title: 'マイカウンター',  // カスタムタイトルを設定
      initialValue: 5          // 初期値を5に設定
    }
  })

  // カスタムタイトルが表示されることを確認
  expect(wrapper.find('h1').text()).toBe('マイカウンター')  // 結果: ✅ PASS - プロップスのタイト

  // 初期値が正しく設定されることを確認
  expect(wrapper.find('.count').text()).toBe('Count: 5')  // 結果: ✅ PASS - プロップスの初期値

  // dataの値も正しく設定されていることを確認
  expect(wrapper.vm.count).toBe(5)  // 結果: ✅ PASS - Vueインスタンスのdataが正しく初期化される
})
```

期待される実行結果：

```
✓ Counter コンポーネント › プロップスが正しく渡される (8ms)
```

テストケース3: インクリメント機能のテスト

```javascript
test('インクリメントボタンクリックでカウントが増える', async () => {

  // コンポーネントをマウント
  const wrapper = mount(Counter)

  // 初期状態の確認
  expect(wrapper.vm.count).toBe(0)  // 結果: ✅ PASS - 初期値は0

  // インクリメントボタンを取得
  const incrementButton = wrapper.find('.increment-btn')

  // ボタンが存在することを確認
  expect(incrementButton.exists()).toBe(true)  // 結果: ✅ PASS - ボタンが存在

  // ボタンをクリック（非同期処理のためawaitが必要）
  await incrementButton.trigger('click')

  // カウントが1増えていることを確認
  expect(wrapper.vm.count).toBe(1)  // 結果: ✅ PASS - count値が1になる

  // 画面表示も更新されていることを確認
  expect(wrapper.find('.count').text()).toBe('Count: 1')  // 結果: ✅ PASS - 表示も更新される

  // もう一度クリック
  await incrementButton.trigger('click')

  // カウントがさらに増えていることを確認
  expect(wrapper.vm.count).toBe(2)  // 結果: ✅ PASS - count値が2になる
})
```

期待される実行結果：

```
✓ Counter コンポーネント › インクリメントボタンクリックでカウントが増える (12ms)
```

3. 異常パターンのテストコード（失敗例と対処法）

失敗例1: セレクターの間違い

```javascript
test('❌ 失敗例: 間違ったセレクター', () => {
  // コンポーネントをマウント
  const wrapper = mount(Counter)  // 正常

  // 存在しないクラス名でボタンを探す
  const button = wrapper.find('.wrong-class-name')  // 存在しないクラス

  // 存在しない要素をクリックしようとする
  expect(button.exists()).toBe(true)  // 結果: ❌ FAIL - 要素が存在しないため
})
```

実際のエラー結果：

```
✗ Counter コンポーネント › ❌ 失敗例: 間違ったセレクター (5ms)

  expect(received).toBe(expected) // Object.is equality

  Expected: true
  Received: false
```

修正版：

```javascript
test('✅ 修正版: 正しいセレクター', () => {
  const wrapper = mount(Counter)

  // 正しいクラス名を使用
  const button = wrapper.find('.increment-btn')  // 存在するクラス名

  // 要素の存在確認
  expect(button.exists()).toBe(true)  // 結果: ✅ PASS - 要素が存在する
})
```

失敗例2: 非同期処理のawait忘れ

```javascript
test('❌ 失敗例: awaitを忘れた場合', () => {
  const wrapper = mount(Counter)

  // awaitを付けずにクリック
  wrapper.find('.increment-btn').trigger('click')  // await忘れ

  // すぐに結果を確認してしまう
  expect(wrapper.vm.count).toBe(1)  // 結果: ❌ FAIL - まだ更新されていない
})
```

実際のエラー結果：

```
✗ Counter コンポーネント › ❌ 失敗例: awaitを忘れた場合 (3ms)

  expect(received).toBe(expected) // Object.is equality

  Expected: 1
  Received: 0
```

修正版：

```javascript
test('✅ 修正版: 正しいawait使用', async () => {
  const wrapper = mount(Counter)

  // awaitを付けてクリック
  await wrapper.find('.increment-btn').trigger('click')  // 正しくawait

  // 更新後の結果を確認
  expect(wrapper.vm.count).toBe(1)  // 結果: ✅ PASS - 正しく更新される
})
```

失敗例3: 型の不一致

```javascript
test('❌ 失敗例: 型の不一致', () => {

  const wrapper = mount(Counter, {
    props: {
      initialValue: '5'  // 文字列として渡してしまう
    }
  })

  // 数値として比較してしまう
  expect(wrapper.vm.count).toBe(5)  // 結果: ❌ FAIL - '5'（文字列）と5（数値）は異なる
})
```

実際のエラー結果：

```
✗ Counter コンポーネント › ❌ 失敗例: 型の不一致 (4ms)

  expect(received).toBe(expected) // Object.is equality

  Expected: 5
  Received: "5"
```

修正版：

```javascript
test('✅ 修正版: 正しい型で渡す', () => {

  const wrapper = mount(Counter, {
    props: {
      initialValue: 5  // 数値として正しく渡す
    }
  })

  expect(wrapper.vm.count).toBe(5)  // 結果: ✅ PASS - 型が一致
})
```

4. 複雑なテストケース（正常・異常両方のパターン）

複数操作のテスト

```javascript
test('複数の操作を組み合わせたテスト', async () => {

  // 初期値5でコンポーネントをマウント
  const wrapper = mount(Counter, {
    props: { initialValue: 5 }
  })

  // 初期状態確認
  expect(wrapper.vm.count).toBe(5)  // 結果: ✅ PASS

  // インクリメント操作 × 3回
  for (let i = 0; i < 3; i++) {
    await wrapper.find('.increment-btn').trigger('click')
  }

  // カウントが8になっていることを確認
  expect(wrapper.vm.count).toBe(8)  // 結果: ✅ PASS (5 + 3 = 8)

  // デクリメント操作 × 2回
  for (let i = 0; i < 2; i++) {
    await wrapper.find('.decrement-btn').trigger('click')
  }

  // カウントが6になっていることを確認
  expect(wrapper.vm.count).toBe(6)  // 結果: ✅ PASS (8 - 2 = 6)

  // リセット操作
  await wrapper.find('.reset-btn').trigger('click')

  // 初期値に戻っていることを確認
  expect(wrapper.vm.count).toBe(5)  // 結果: ✅ PASS (初期値に戻る)

  // 画面表示も正しく更新されていることを確認
  expect(wrapper.find('.count').text()).toBe('Count: 5')  // 結果: ✅ PASS
})
```

5. よくあるエラーパターンと対処法

パターン1: テストタイムアウト

```javascript
// ❌ 問題のあるコード
test('タイムアウトが発生するテスト', async () => {
  const wrapper = mount(Counter)

  // 無限ループや重い処理があるボタン
  await wrapper.find('.heavy-processing-btn').trigger('click')

  // この行に到達する前にタイムアウト
  expect(wrapper.vm.someValue).toBe('expected')
})
```

エラー結果：

```
X Counter コンポーネント › タイムアウトが発生するテスト (10000ms)
  thrown: "Exceeded timeout of 10000 ms for a test."
```

対処法：

```javascript
// ✅ 修正版
test('タイムアウト対策版', async () => {
  const wrapper = mount(Counter)

  // タイムアウト時間を延長
}, 15000)  // 15秒のタイムアウト

// または、モックを使用して重い処理を回避
```

パターン2: DOM更新の待機不足

```javascript
// ❌ 問題のあるコード
test('DOM更新待機不足', async () => {
  const wrapper = mount(Counter)

  await wrapper.find('.increment-btn').trigger('click')

  // Vue.jsの更新サイクルを待たずに確認
  expect(wrapper.find('.count').text()).toBe('Count: 1')  // 失敗の可能性
})
```

対処法：

javascript

```javascript
// ✅ 修正版
test('DOM更新待機あり', async () => {
  const wrapper = mount(Counter)

  await wrapper.find('.increment-btn').trigger('click')

  // Vue.jsの更新を明示的に待つ
  await wrapper.vm.$nextTick()

  expect(wrapper.find('.count').text()).toBe('Count: 1')  // 結果: ✅ PASS
})
```

6. テスト実行結果の読み方

成功時の出力例

bash

```
 ✓ src/components/__tests__/Counter.test.js (4)
   ✓ Counter コンポーネント (4)
     ✓ デフォルトの状態で正しくレンダリングされる
     ✓ プロップスが正しく渡される
     ✓ インクリメントボタンクリックでカウントが増える
     ✓ 複数の操作を組み合わせたテスト

 Test Files  1 passed (1)
      Tests  4 passed (4)
   Start at  10:30:25
   Duration  285ms
```

失敗時の出力例

bash

```
 ✗ src/components/__tests__/Counter.test.js (1)
   ✗ Counter コンポーネント (1)
     ✗ セレクターが間違っているテスト
       AssertionError: expected false to be true

       at Object.<anonymous> (src/components/__tests__/Counter.test.js:15:32)

 Test Files  1 failed (1)
      Tests  1 failed (1)
   Start at  10:30:25
   Duration  156ms
```

7. デバッグのコツ

console.logを使ったデバッグ

javascript

```
test('デバッグ用のテスト', async () => {
  const wrapper = mount(Counter)

  // 現在の状態を確認
  console.log('初期状態:', wrapper.vm.count)  // コンソールに出力

  await wrapper.find('.increment-btn').trigger('click')

  // 更新後の状態を確認
  console.log('クリック後:', wrapper.vm.count)
  console.log('HTML:', wrapper.html())  // HTML全体を出力

  expect(wrapper.vm.count).toBe(1)
})
```

wrapper.vm を使った内部状態の確認

```javascript
test('内部状態の詳細確認', () => {
  const wrapper = mount(Counter, {
    props: { initialValue: 10 }
  })

  // Vueインスタンスの全プロパティを確認
  console.log('Vue instance:', wrapper.vm)
  console.log('Props:', wrapper.vm.$props)
  console.log('Data:', wrapper.vm.$data)
})
```

8. まとめ

テスト成功のポイント

1. **正しいセレクターを使用** - 存在する要素を正確に指定
2. **非同期処理にはawaitを付ける** - DOM更新やイベント処理の完了を待つ
3. **型を正しく渡す** - プロップスやデータの型に注意
4. **段階的に確認** - 一度に多くをテストせず、段階的に確認

よくある失敗の原因

1. **セレクターのタイプミス** - クラス名やID名の間違い
2. **非同期処理の理解不足** - awaitの付け忘れ
3. **型の不一致** - 文字列と数値の混同
4. **DOM更新タイミング** - Vue.jsの更新サイクルの理解不足

テストコードは最初は難しく感じますが、1行ずつ理解していけば必ずマスターできます。エラーが出ても慌てず、エラーメッセージをよく読んで対処していきましょう！