如何撰寫具彈性的測試程式

Summer@WebConf 2024

Summer

Summer。桑莫。夏天　cythilya.tw

📖 前端測試指南：策略與實踐
打造高速網站，從網站指標開始！全方位提升
使用者體驗與流量的關鍵

cythilya.dev　cythilya　cythilya@gmail.com


UI 測試中

撰寫適應變更、有彈性的測試程式

設計保護力佳的測試架構

撰寫適應變更、有彈性的測試程式

測試為何失敗

UI 一更新測試就失敗，該怎麼辦？

選取元素的方式

太過鬆散、嚴謹或意義不明

太過鬆散

```
const Hello = () => {
  return (
    <div>
      這是盒子
      <div>測試文字</div>
    </div>
  );
};
```

```
it('should render the correct
content', () => {
  const { getByText } = render(<Hello
/>);

  expect(
    getByText('測試文字').textContent
  ).toBe('測試文字');
});
```

改為 toHaveTextContent 更具可讀性

需求變更，改起來很麻煩 :(

```
it('should render the correct content', () => {
  const { getByText } = render(<Hello />);

  expect(getByText('測試文字').textContent).toBe('測試文字');
});
```

無法辨識唯一性

```
const Hello = () => {
  return (
    <div>
      這是盒子
      <div>這是測試文字</div>
      <div>這是測試文字</div>
    </div>
  );
};
```

太過嚴謹

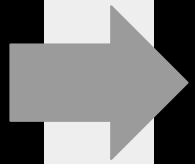
XPath：/div/div

```
const Hello = () => {
  return (
    <div>
      這是盒子
      <div>測試文字</div>
    </div>
  );
};
```

更新標籤

```
const Hello = () => {
  return (
    <div>
      這是盒子
      <div>測試文字</div>
    </div>
  );
};
```

XPath：/div/div



```
const Hello = () => {
  return (
    <div>
      這是盒子
      <p>測試文字</p>
    </div>
  );
};
```

XPath：/div/p

意義不夠明確

樣式與測試混用

```
const Hello = () => {
  return (
      <div className="text">
        測試文字
      </div>
  );
};
```

`data-* attribute`

專注與彈性

```
const Hello = () => {
  return (
    <div data-test-id="text">
        測試文字
    </div>
  );
};
```

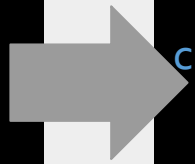
`data-test-id`

```
it('should render the correct content 2', () => {
  const { getByTestId } = render(<Hello />);

  expect(getByTestId('text')).toHaveTextContent('測試文字');
});
```

包含過多實作細節

過多模擬

```
jest.mock('./Hello, () => (
  <div data-test-id="hello">Hello
World!</div>
));

const Hello = () => {
  return (
    <div data-test-id="hello">
      Hello World!
    </div>
  );
};
```



```
jest.mock('./Hello, () => (
  <div data-test-id="hello">Hello
World!</div>
));

const Hello = () => {
  return (
    <div data-test-id="hello">
      Hi!
    </div>
  );
};
```

快照

記錄實作細節

```
exports[`ImageList should render correct
image items when have data 1`] = `
Array [
  <div
    data-test-id="image-item"
  >
    <img src="https://bit.ly/4228IT0" />
    <p data-test-id="image-item-title">
      打造高速網站，從網站指標開始！
      全方位提升使用者體驗與流量的關鍵
    </p>
  </div>,
  <div
    data-test-id="image-item"
  >
    <img src="https://bit.ly/4b5nLPO"/>
    <p data-test-id="image-item-title">
      島波海道。單車跳島之旅
    </p>
  </div>,
]
```

比對細節

```
<div data-test-id="item-quantity">
  2
</div>
```

```
expect(getByTestId('item-quantity'))
  .toHaveTextContent('2')
```

```
expect(getByTestId('item-quantity'))
  .toBe(DOM 結構)
```

拆分邏輯、狀態與顯示

`<Calculator>`


```jsx
const Calculator = () => {
  const [num1, setNum1] = useState('');
  const [num2, setNum2] = useState('');
  const [operator, setOperator] = useState('+');
  const [result, setResult] = useState('');

  const handleChange = (e) => // 略...

  const calculateResult = () => {
    switch (operator) {
      case '+':
        setResult(parsedNum1 + parsedNum2);
        break;
        // 略...
    }
  };

  return (
    <div>
      <input type="number" value={num1} onChange={handleNum1Change}
      />
      <select value={operator} onChange={handleOperatorChange} >
        <option value="+">+</option>
          // 略...
      </select>
      <input type="number" value={num2} onChange={handleNum2Change} />
      <button onClick={calculateResult}>Calculate</button>
      <div>Result: {result}</div>
    </div>
  );
};
```

`<Calculator>`

將運算放在 utils

```
const calculate = (num1, num2, operator) =>
{
 switch (operator) {
   case '+':
     return num1 + num2;
   case '-':
     return num1 - num2;
   // 略...
 }
};
```

寫測試

將資料運算放在 utils

優點

- 對重構專案的壓力最小、最容易實作
- 保護效果顯著
- AI 快速實作，節省成本

```
describe('calculate function', () => {
 test('should get 8 when add 5 and 3', ()
=> {
   expect(calculate(5, 3, '+')).toBe(8);
 });
});
```

`<Calculator>`

將商業邏輯與資料狀態
封裝在 custom hook

```
const useCalculator = () => {
  const [num1, setNum1] = useState(0);
  const [num2, setNum2] = useState(0);
  // 略...

  const handleNum1Change = e =>
    setNum1(parseFloat(e.target.value) || 0);
  // 略...

  return {
    calculateResult, handleNum1Change, ...
  };
};
```

寫測試 - 將商業邏輯與資料狀態封裝在 custom hook

```
test('should get 10 when add 7 and 3', () => {
   const { result } = renderHook(() => useCalculator());
   act(() => {
     result.current.handleNum1Change({ target: { value: '7' } });
     result.current.handleNum2Change({ target: { value: '3' } });
     result.current.handleOperatorChange({ target: { value: '+' } });
     result.current.calculateResult();
   });
   expect(result.current.result).toBe(10);
 });
```

`<Calculator>`

將畫面顯示放在元件

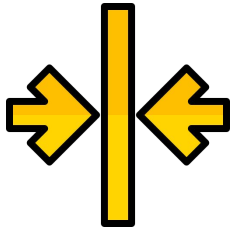
```
const Calculator = () => {
 const { handleNum1Change, ...} = useCalculator();

 return (
   <>
     <input
       data-test-id='number1'
       type='number'
       value={num1}
       onChange={handleNum1Change}
     />
     <select
       data-test-id='operator'
       value={operator}
       onChange={handleOperatorChange}
     >
       <option value='+'>+</option>
     </select>
     // 略...
```

寫測試 - 將畫面顯示放在元件

```
test('updates result correctly for subtraction', () => {
   render(<Calculator />);

   const number1Input = screen.getByTestId('number1');
   // 略...

   fireEvent.change(number1Input, { target: { value: '20' } });
   fireEvent.change(number2Input, { target: { value: '7' } });
   fireEvent.change(operatorSelect, { target: { value: '-' } });
   fireEvent.click(calculateButton);
   expect(result).toBe('13');
 });
```

總結 - 撰寫適應變更、有彈性的測試程式

data-*
attribute`

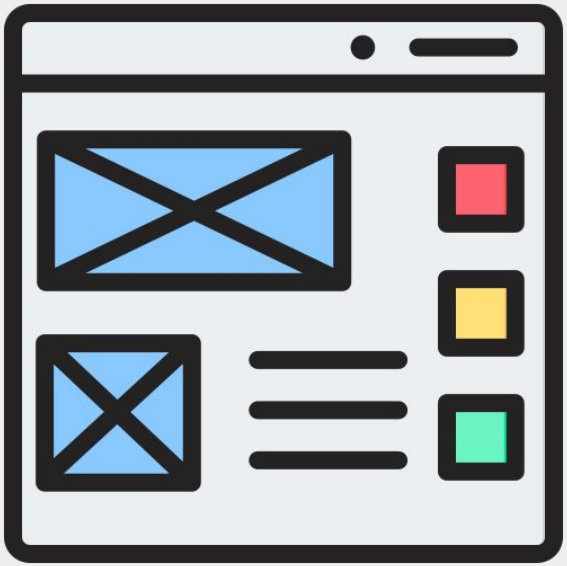


避免過多
實作細節

拆分邏輯、狀態
與顯示

前端測試指南：策略與實踐

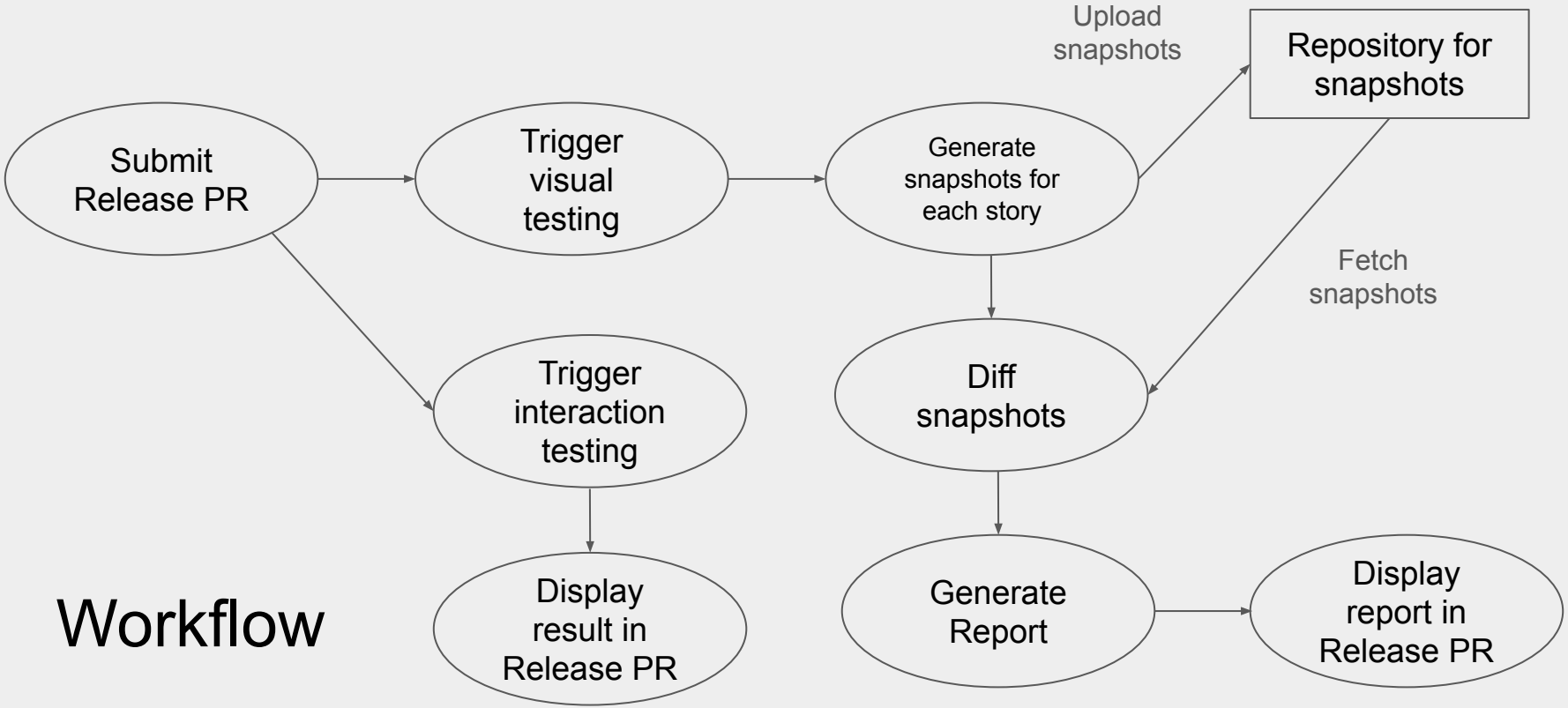
設計保護力佳的測試架構



Visual testing

Interaction testing

Workflow

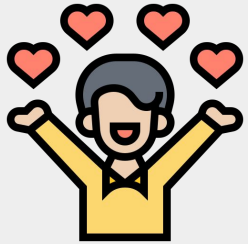

Submit Release PR
Trigger visual testing
Generate snapshots for each story
Upload snapshots
Repository for snapshots
Fetch snapshots
Trigger interaction testing
Diff snapshots
Display result in Release PR
Generate Report
Display report in Release PR

總結 - 設計保護力佳的測試架構

視覺

行為

成本



突破框架，流程整合

特別感謝

Max Shen • Taylor Wu

Thank you!