軟體測試

以 JavaScript 與 Vue 3 為例

大綱

- 以人工測試
- 用程式測試
  - 自己寫測試
  - 用套件測試
- TDD(Test-Driven Development) 範例

人工測試

加總函數

```
function sum(a, b) {
    return a + b;
}
```

```
sum(1, 2) // 3
sum(10, 4) // 14
sum(3, 6) // 9
```

用程式測試

測試的種類

- 單元測試
- 整合測試
- 端對端測試，(End to end，簡稱 e2e 測試)

自己寫測試

加總函數

```
function sum(a, b) {
    return a + b;
}
```

測試程式

```
console.log(sum(1, 2) == 3) // true
console.log(sum(10, 4) == 14) // true
console.log(sum(3, 6) == 9) // true
```

稍微包装一下

```
function sum(a, b) {
    return a + b;
}
```

```
function testSum() {
    if (sum(1, 2) != 3) return false
    if (sum(10, 4) != 14) return false
    if (sum(3, 6) != 9) return false
    return true
}

if (testSum()) {
    console.log('pass')
} else {
    console.log('error')
}
```

用套件測試

初始化專案

```
npm init -y
```

安裝測試套件 jest

```
npm install --save-dev jest
```

修改 package.json

```
// ...
"scripts": {
    "test": "jest"
},
// ...
```

跑測試

```
npm run test
```

測試並顯示覆蓋率

```
npm run test --coverage
```

加總函數 (sum.js)

```
function sum(a, b) {
    return a + b;
}
```

測試程式 (sum.test.js)

```
test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

測試覆蓋率

```
PASS ./sum.test.js
  √ adds 1 + 2 to equal 3 (2 ms)

----------|---------|----------|---------|---------|-------------------
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------|---------|----------|---------|---------|-------------------
All files |     100 |      100 |     100 |     100 |
 sum.js   |     100 |      100 |     100 |     100 |
----------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.385 s, estimated 1 s
Ran all test suites matching /sum/i.
Done in 0.97s.
```

- Stmts: 陳述式(表達式)的覆蓋率 (例如 a = 5; b = 10;)
- Branch: 分支的覆蓋率 (例如 if-else)
- Funcs: 函數的覆蓋率
- Lines: 行數的覆蓋率

測試覆蓋率 - 範例

```
PASS ./coverage.test.js
  √ test branch 1 (1 ms)
  √ test func a

-----------|---------|----------|---------|---------|-------------------
File       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------|---------|----------|---------|---------|-------------------
All files  |      60 |       50 |      50 |   66.66 |
 branch.js |      75 |       50 |     100 |      75 | 3
 funcs.js  |   66.66 |      100 |      50 |   66.66 | 6
 stmts.js  |   33.33 |      100 |       0 |      50 | 2
-----------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.404 s, estimated 1 s
Ran all test suites.
Done in 1.11s.
```

Stmts

```
function stmts(x, y) {
    const sum = x + y; return x + y;
}

module.exports = stmts
```

測試程式

```
無
```

```
PASS ./coverage.test.js
  √ test branch 1 (1 ms)
  √ test func a

-----------|---------|----------|---------|---------|-------------------
File       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------|---------|----------|---------|---------|-------------------
All files  |      60 |       50 |      50 |   66.66 |
 branch.js |      75 |       50 |     100 |      75 | 3
 funcs.js  |   66.66 |      100 |      50 |   66.66 | 6
 stmts.js  |   33.33 |      100 |       0 |      50 | 2
-----------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.404 s, estimated 1 s
Ran all test suites.
Done in 1.11s.
```

Branch

```
function branch(n) {
    if (n % 2 === 0) {
        return 'even'
    } else {
        return 'odd'
    }
}

module.exports = branch
```

測試程式

```
test('test branch 1', () => {
    expect(branch(1)).toBe('odd')
})
```

```
PASS ./coverage.test.js
  √ test branch 1 (1 ms)
  √ test func a

-----------|---------|----------|---------|---------|-------------------
File       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------|---------|----------|---------|---------|-------------------
All files  |      60 |       50 |      50 |   66.66 |
 branch.js |      75 |       50 |     100 |      75 | 3
 funcs.js  |   66.66 |      100 |      50 |   66.66 | 6
 stmts.js  |   33.33 |      100 |       0 |      50 | 2
-----------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.404 s, estimated 1 s
Ran all test suites.
Done in 1.11s.
```

Funcs

```
function a() {
    return 'a'
}

function b() {
    return 'b'
}

module.exports = {
    a, b
}
```

測試程式

```
test('test func a', () => {
    expect(funcs.a()).toBe('a')
})
```

測試驅動開發

TDD(Test-Driven Development)

方法

- 先寫測試在開發
  - 寫測試
  - 得到紅燈(測試沒過)
  - 實作讓測試通過的程式
  - 得到綠燈(測試過了)
  - 重構程式 (包含測試)

帶來的好處

- 測試積少成多，覆蓋率高，最後對系統信心度越高
- 一開始就可以從使用者的觀點切入去寫測試
- 強迫降低程式耦合度(不然測試不好寫)
- 不怕改A壞B(如果你有測到)

範例 1

找零錢問題

需求

- 預設面額為 [1, 5, 10, 50]，也可以自己設定，且面額一定要有 1 元
- 可以給他一個金額，幫我找出最小數量的硬幣

※ 為了精簡一些，假設丟進去的面額都排序過。

測試預設面額

```
const CoinChanger = require('./coin-changer');

test('new CoinChanger default coins = [1, 5, 10, 50].', () => {
    const coinChanger = new CoinChanger();
    expect(coinChanger.coins).toEqual([1, 5, 10, 50]);
})
```

寫剛好的產品程式(實現預設面額)

```
class CoinChanger {
    constructor() {
        this.coins = [1, 5, 10, 50]
    }
}

module.exports = CoinChanger
```

測試自訂面額

```
test('new CoinChanger withs coins = [1, 5, 7, 10].', () => {
    const coinChanger = new CoinChanger([1, 5, 7, 10]);
    expect(coinChanger.coins).toEqual([1, 5, 7, 10]);
})
```

實現自訂面額

```
class CoinChanger {
    constructor(coins = [1, 5, 10, 50]) {
        this.coins = coins
    }
}

module.exports = CoinChanger
```

測試當不存在面額 1 的情況

```
test('when new CoinChanger without 1 coin, throw error.', () => {
    expect(() => {
        new CoinChanger([5, 10, 50]);
    }).toThrow();
})
```

實現

```
class CoinChanger {
    constructor(coins = [1, 5, 10, 50]) {
        this.coins = coins
        if (!this.coins.includes(1)) {
            throw new Error('coins must include 1');
        }
    }
}

module.exports = CoinChanger
```

測試用預設面額換 30 元

```
test('default coins change coins with 30.', () => {
    const coinChanger = new CoinChanger();
    expect(coinChanger.change(30)).toEqual([10, 10, 10]);
})
```

實現

```
    change(amount) {
        const result = [];
        let rest = amount;
        for (let i = this.coins.length - 1; i >= 0; i--) {
            const coin = this.coins[i];
            while (rest >= coin) {
                result.push(coin);
                rest -= coin;
            }
        }
        return result;
    }
```

測試以自訂面額換 24 元

```
test('coins = [1, 5, 7, 10] change coins with 24.', () => {
    const coinChanger = new CoinChanger([1, 5, 7, 10]);
    expect(coinChanger.change(24)).toEqual([10, 7, 7]);
})
```

實現

```
    change(amount) {
        const dp = Array(amount + 1).fill(0).map((_, i) => i)
        for (let coin of this.coins) {
            for (let i = 1; i <= amount; i++) {
                if (i >= coin) {
                    dp[i] = Math.min(dp[i], dp[i - coin] + 1)
                }
            }
        }

        const result = []
        while (amount > 0) {
            for (let i = this.coins.length - 1; i >= 0; i--) {
                if (dp[amount] === dp[amount - this.coins[i]] + 1) {
                    result.push(this.coins[i])
                    amount -= this.coins[i]
                    break
                }
            }
        }
        return result
    }
```

測試覆蓋率

```
PASS ./coin-changer.test.js
  √ new CoinChanger default coins = [1, 5, 10, 50]. (2 ms)
  √ new CoinChanger withs coins = [1, 5, 7, 10].
  √ when new CoinChanger without 1 coin, throw error. (6 ms)
  √ default coins change coins with 30.
  √ coins = [1, 5, 7, 10] change coins with 24.

-----------------|---------|----------|---------|---------|-------------------
File             | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------|---------|----------|---------|---------|-------------------
All files        |     100 |      100 |     100 |     100 |
 coin-changer.js |     100 |      100 |     100 |     100 |
-----------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        0.276 s, estimated 1 s
```

範例 2

Vue Counter

需求

- 定義一個計數器的元件(Component)
- 預設值為 0
- 最小值為 0
- 當點擊遞增按鈕則 +1
- 當點擊遞減按鈕則 -1

設置環境

開新專案

```
npm init vue@latest
```

- 建立元件 /src/components/TheCounter.vue
- 建立測試 /src/components/__tests__/TheCounter.spec.js

執行測試

```
npm run test:unit
```

測試並顯示覆蓋率

```
npm run test:unit --coverage
```

測試

```
import { describe, it, expect } from "vitest";

import { mount } from "@vue/test-utils";
import Counter from "../TheCounter.vue";

describe("Counter", () => {
  it("default render count is 0.", () => {
    const wrapper = mount(Counter);
    expect(wrapper.find("span").text()).toBe("0");
  });
});
```

實現

```
<template>
  <span>0</span>
</template>
```

測試

```
  it("click increment button count is 1.", async () => {
    const wrapper = mount(Counter);
    await wrapper.find("#increment").trigger("click");
    expect(wrapper.find("span").text()).toBe("1");
  });
```

實現

```
<script setup>
  import { ref } from 'vue';

  const count = ref(0);
  const increment = () => {
    count.value = 1
  };
</script>

<template>
  <button id="increment" @click="increment">increment</button>
  <span>{{ count }}</span>
</template>
```

這時發現 increment 是寫死的，把他抽象化

```
  const increment = () => {
    count.value = count.value + 1
  };
```

測試

```
  it("click decrement button count is -1, but min count is 0.", async () => {
    const wrapper = mount(Counter);
    await wrapper.find("#decrement").trigger("click");
    expect(wrapper.find("span").text()).toBe("0");
  });
```

實現

```
<script setup>
  //   略...

  const decrement = () => {
    count.value = 0
  };
</script>

<template>
    <!-- 略... -->
  <button id="decrement" @click="decrement">decrement</button>
</template>
```

測試

```
it("click increment button twice and then decrement button count is 1.", async () =>
  const wrapper = mount(Counter);
  await wrapper.find("#increment").trigger("click");
  await wrapper.find("#increment").trigger("click");
  await wrapper.find("#decrement").trigger("click");
  expect(wrapper.find("span").text()).toBe("1");
});
```

實現

```
const decrement = () => {
  count.value = count.value - 1
};
```

改成這樣會發生問題

通過: click increment button twice and then decrement button count is 1.

失敗: click decrement button count is -1, but min count is 0.

所以要再修改成以下

```
const decrement = () => {
  count.value = Math.max(0, count.value - 1)
};
```

完整程式碼

TheCounter

```
<script setup>
  import { ref } from 'vue';

  const count = ref(0);
  const increment = () => {
    count.value = count.value + 1
  };

  const decrement = () => {
    count.value = Math.max(0, count.value - 1)
  };
</script>

<template>
  <button id="increment" @click="increment">increment</button>
  <span>{{ count }}</span>
  <button id="decrement" @click="decrement">decrement</button>
</template>
```

測試程式

```
import { describe, it, expect } from "vitest";

import { mount } from "@vue/test-utils";
import Counter from "../TheCounter.vue";

describe("Counter", () => {
  it("default render count is 0.", () => {
    const wrapper = mount(Counter);
    expect(wrapper.find("span").text()).toBe("0");
  });

  it("click increment button count is 1.", async () => {
    const wrapper = mount(Counter);
    await wrapper.find("#increment").trigger("click");
    expect(wrapper.find("span").text()).toBe("1");
  });

  it("click decrement button count is -1, but min count is 0.", async () => {
    const wrapper = mount(Counter);
    await wrapper.find("#decrement").trigger("click");
    expect(wrapper.find("span").text()).toBe("0");
  });

  it("click increment button twice and then decrement button count is 1.", async () =>
    const wrapper = mount(Counter);
    await wrapper.find("#increment").trigger("click");
    await wrapper.find("#increment").trigger("click");
    await wrapper.find("#decrement").trigger("click");
    expect(wrapper.find("span").text()).toBe("1");
  });
});
```

使用方式，直接到 App.vue

```vue
<script setup>
  import TheCounter from "./components/TheCounter.vue";
</script>

<template>
  <TheCounter />
</template>
```

測試都通過了，至少會照著我們寫的 test cases 運作。

範例 3

Router

mock範例需跑在4.0.多版，4.1版目前有bug

需求

- 頁面
  - 按下 #about-link 可以到 /about 頁面
- 組件
  - 按下按鈕可以回到上一頁

測試

```
import { mount } from "@vue/test-utils";
import { describe, expect, test, vi } from "vitest";
import App from "@/App.vue";
import router from "..";

describe("test router", () => {
  test("click about link, go to /about page.", async () => {
    const wrapper = mount(App, {
      global: {
        plugins: [router],
      },
    });

    const push = vi.spyOn(router, "push");

    const link = wrapper.get("#about-link");
    await link.trigger("click");

    expect(push).toBeCalledWith("/about");
  });
});
```

實現

```
<RouterLink id="about-link" to="/about">About</RouterLink>
```

或

```
<button id="about-link" @click="$router.push('/about')">about</button>
```

又或者

```
<script setup>
  import { useRouter } from "vue-router";

  const router = useRouter();
  const goAboutPage = () => router.push("/about");
</script>

<template>
  <button id="about-link" @click="goAboutPage">about</button>
</template>
```

測試

```
import { mount } from "@vue/test-utils";
import { describe, expect, test, vi } from "vitest";
import { useRouter } from "vue-router";
import TheBackButton from "../TheBackButton.vue";

vi.mock("vue-router", () => ({
  useRouter: vi.fn(),
}));

describe("test TheBackButton", () => {
  test("when click button should router back.", async () => {
    const back = vi.fn();
    useRouter.mockImplementationOnce(() => ({
      back,
    }));
    const wrapper = mount(TheBackButton);
    const button = wrapper.find("button");
    await button.trigger("click");
    expect(back).toHaveBeenCalled();
  });
});
```

實現

```
<script setup>
import { useRouter } from "vue-router";

const router = useRouter();
const back = () => {
  router.back();
};
</script>
<template>
  <button @click="back"></button>
</template>
```

不能使用 $router

```
<template>
  <button @click="$router.back()"></button>
</template>
```

除非改成使用真實 router

```
  test("when click button should router back.", async () => {
    const wrapper = mount(TheBackButton, {
      global: {
        plugins: [router],
      },
    });

    const back = vi.spyOn(router, "back");
    const button = wrapper.find("button");
    await button.trigger("click");
    expect(back).toHaveBeenCalled();
  });
```

範例 4

Pinia

需求

- 當 window 被點擊，呼叫 store 中的 increment 方法
- 當 unmount 應該刪除 window 點擊事件

測試

```
import { mount } from "@vue/test-utils";
import { describe, expect, test, vi } from "vitest";
import { useCounterStore } from "./stores/counter";
import App from "@/App.vue";
import router from "./router";
import { createPinia, setActivePinia } from "pinia";

describe("test App", () => {
  test("when click window, call counterStore increment.", () => {
    const pinia = createPinia();
    setActivePinia(pinia);

    const counterStore = useCounterStore();
    const increment = vi.spyOn(counterStore, "increment");

    mount(App, {
      global: {
        plugins: [router, pinia],
      },
    });

    window.dispatchEvent(new Event("click"));
    expect(increment).toHaveBeenCalled();
  });
});
```

實現

```
const { increment } = useCounterStore();
onMounted(() => window.addEventListener("click", increment));
```

測試

```
test("when unmounted should remove window click increment event.", () => {
  const pinia = createPinia();
  setActivePinia(pinia);

  const counterStore = useCounterStore();
  const increment = vi.spyOn(counterStore, "increment");

  const wrapper = mount(App, {
    global: {
      plugins: [router, pinia],
    },
  });

  wrapper.unmount();
  window.dispatchEvent(new Event("click"));
  expect(increment).not.toHaveBeenCalled();
});
```

實現

```
onUnmounted(() => window.removeEventListener("click", increment));
```

THANK YOU!