Isolation Frameworks

The Art of Unit Testing, 3e

Summer

2024/05/02

Agenda

- 測試框架的類型：Loosely Typed vs Strongly Typed
- 模組模擬 (Modular Faking)
- 函式模擬 (Function Faking)
- 介面模擬 (Interface Faking)
- 動態模擬行為 (Dynamic Stubbing Behavior)
- 寫測試，到底要用手刻？還是框架？

Isolation Frameworks

Stubs

Mocks

Utilities

Jest

Loosely Typed

vs

```
substitute.js
```

Strongly Typed

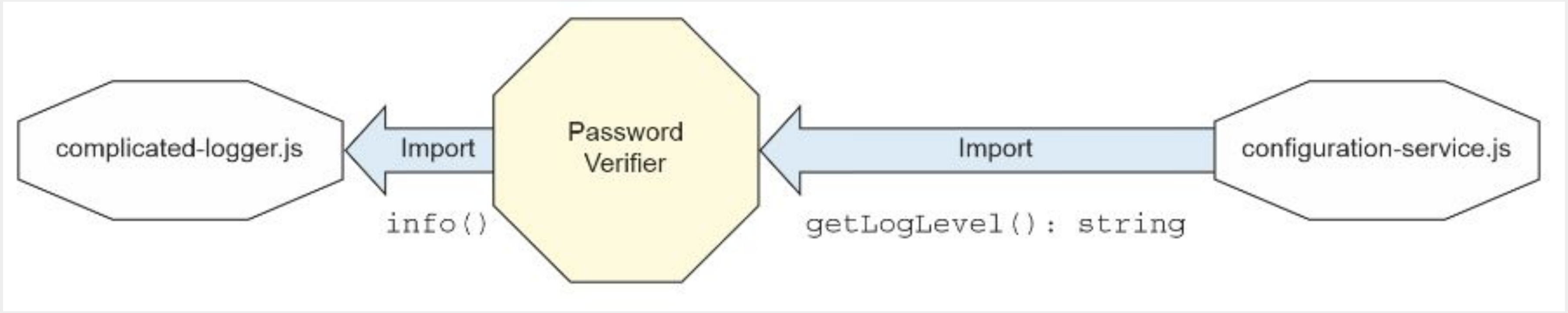
Fake

Module

Function

Object-oriented

模組模擬 (Modular Faking)



```
import { verifyPassword } from './password-verifier';
import { getLogLevel } from './configuration-service';
import { debug, info } from './logger';

jest.mock('./logger');
jest.mock('./configuration-service');

const { stringMatching } = expect;

describe('password verifier', () => {
  afterEach(() => {
    jest.resetAllMocks();
  });
```

```
 it('should call info with PASSED when no rules', () => {
    getLogLevel.mockReturnValue('info');

    verifyPassword('anything', []);

    expect(info).toHaveBeenCalledWith(stringMatching(/PASS/));
  });

  it('should call debug with PASSED when no rules', () => {
    getLogLevel.mockReturnValue('debug');

    verifyPassword('anything', []);

    expect(debug).toHaveBeenCalledWith(stringMatching(/PASS/));
  });
});
```

```
describe('password verifier', () => {
  it('should call info with PASSED when no rules 1', () => {
    getLogLevel.mockReturnValue('info');

    verifyPassword('anything', []);

    expect(info).toHaveBeenCalledWith(stringMatching(/PASS/));

    expect(info).toHaveBeenCalledTimes(1);
  });

  it('should call info with PASSED when no rules 2', () => {
    getLogLevel.mockReturnValue('info');

    verifyPassword('anything', []);

    expect(info).toHaveBeenCalledWith(stringMatching(/PASS/));

    // Received number of calls: 2
    expect(info).toHaveBeenCalledTimes(1);
  });
});
```

```
describe('password verifier', () => {
  afterEach(() => jest.resetAllMocks());

  it('should call info with PASSED when no rules 1', () => {
    getLogLevel.mockReturnValue('info');

    verifyPassword('anything', []);

    expect(info).toHaveBeenCalledWith(stringMatching(/PASS/));

    expect(info).toHaveBeenCalledTimes(1);
  });

  it('should call info with PASSED when no rules 2', () => {
    getLogLevel.mockReturnValue('info');

    verifyPassword('anything', []);

    expect(info).toHaveBeenCalledWith(stringMatching(/PASS/));

    expect(info).toHaveBeenCalledTimes(1);
  });
});
```

重置模擬

```
// 真實的實作細節
const HelloWorld = () => {
  return <div data-test-id="hello-world">Hello World</div>;
};

// 模擬
jest.mock('./HelloWorld', () =>
  <div data-test-id="hello-world">Hi</div>
);

// 在測試程式中會這樣用 ...
expect(getByTestId('hello-world')).toHaveTextContent('Hi');
```

函式模擬 (Function Faking)

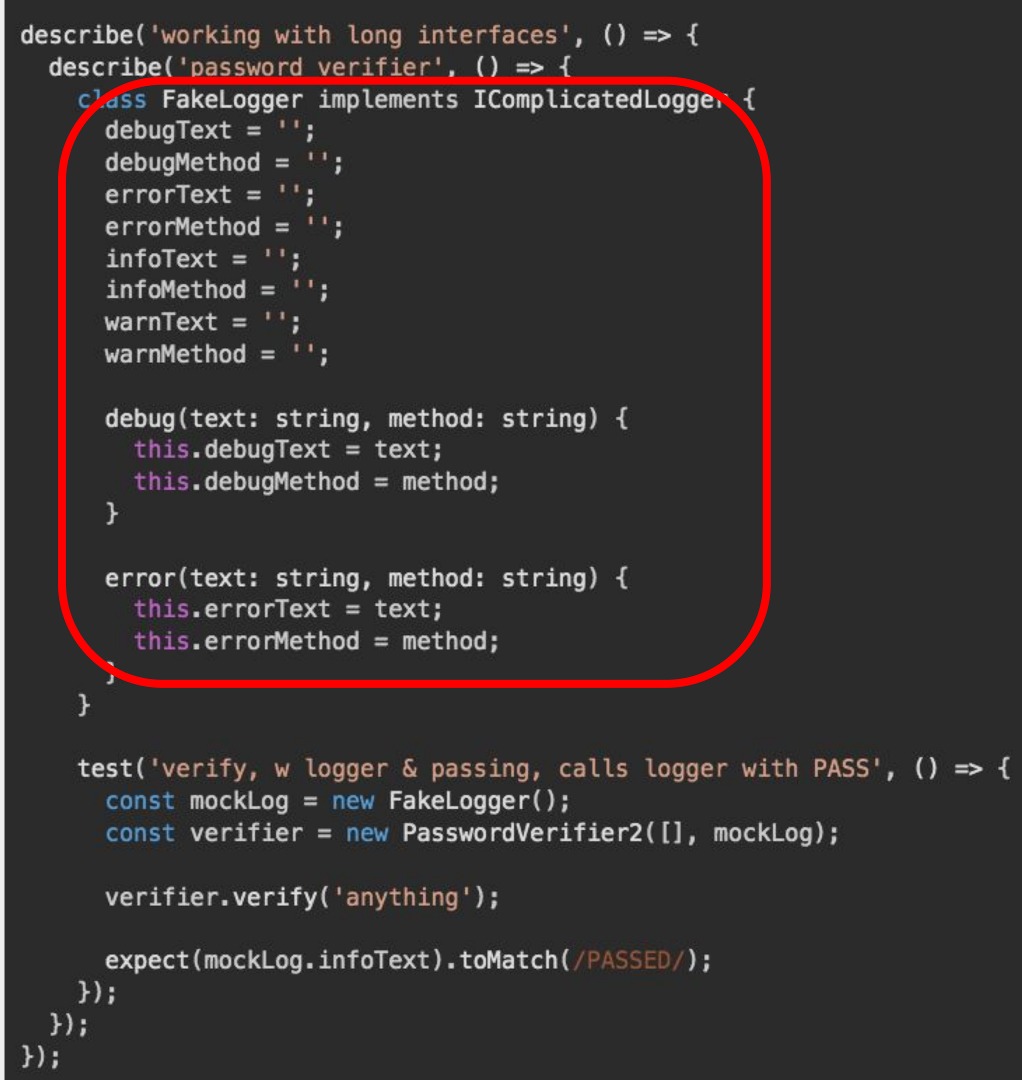
```
test('given logger and passing scenario', () => {
  let logged = '';
  const mockLog = { info: (text) => (logged = text) };
  const verify = makeVerifier([], mockLog);

  verify('any input');

  expect(logged).toMatch(/PASSED/);
});
```

將監控的機制交給 Jest

```
test('given logger and passing scenario', () => {
  const mockLog = { info: jest.fn() };
  const verify = makeVerifier([], mockLog);

  verify('any input');

  expect(mockLog.info).toHaveBeenCalledWith(stringMatching(/PASS/));
});
```

介面模擬 (Interface Faking)

```
interface IComplicatedLogger {
  info(text: string, method: string);
  debug(text: string, method: string);
  warn(text: string, method: string);
  error(text: string, method: string);
}
```



```
describe('working with long interfaces', () => {
  describe('password verifier', () => {
    class FakeLogger implements IComplicatedLogger {
      debugText = '';
      debugMethod = '';
      errorText = '';
      errorMethod = '';
      infoText = '';
      infoMethod = '';
      warnText = '';
      warnMethod = '';

      debug(text: string, method: string) {
        this.debugText = text;
        this.debugMethod = method;
      }

      error(text: string, method: string) {
        this.errorText = text;
        this.errorMethod = method;
      }
    }

    test('verify, w logger & passing, calls logger with PASS', () => {
      const mockLog = new FakeLogger();
      const verifier = new PasswordVerifier2([], mockLog);

      verifier.verify('anything');

      expect(mockLog.infoText).toMatch(/PASSED/);
    });
  });
});
```

```
describe('working with long interfaces', () => {
  describe('password verifier', () => {
    test('verify, w logger & passing, calls logger with PASS', () => {
      const mockLog: IComplicatedLogger = {
        info: jest.fn(),
        warn: jest.fn(),
        debug: jest.fn(),
        error: jest.fn(),
      };

      const verifier = new PasswordVerifier2([], mockLog);
      verifier.verify('anything');

      expect(mockLog.info).toHaveBeenCalledWith(stringMatching(/PASS/));
    });
  });
});
```

Refactored by `jest.fn`

Refactored by substitute.js

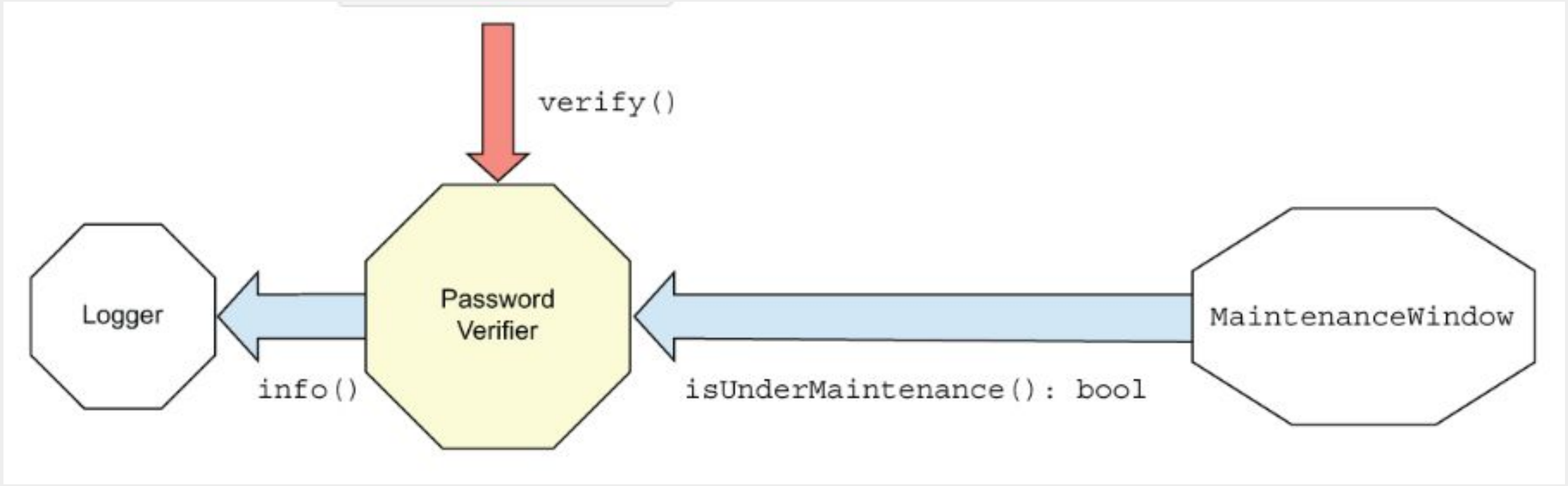
```
describe('working with long interfaces', () => {
  describe('password verifier', () => {
    test('verify, w logger & passing, calls logger w PASS', () => {
      const mockLog = Substitute.for<IComplicatedLogger>();

      const verifier = new PasswordVerifier2([], mockLog);
      verifier.verify('anything');

      mockLog.received().info(
        Arg.is((x) => x.includes('PASSED')),
        'verify'
      );
    });
  });
});
```

動態模擬行為

Dynamic Stubbing Behavior

- 模擬回傳值
  - mockReturnValue
  - mockReturnValueOnce
- 模擬拋出例外或內涵其他操作
  - mockImplementation
  - mockImplementationOnce

動態模擬行為


verify()
Logger
Password Verifier
MaintenanceWindow
info()
isUnderMaintenance(): bool

```
describe('working with substitute', () => {
 test('verify, with logger, calls logger', () => {
    const stubMaintWindow: MaintenanceWindow = {
      isUnderMaintenance: jest
        .fn()
        .mockImplementationOnce(() => true)
        .mockImplementationOnce(() => false),
    };

    const mockLog = Substitute.for<IComplicatedLogger>();

    const verifier = new PasswordVerifier3([], mockLog, stubMaintWindow);

    verifier.verify('anything');

    mockLog.received().info(
      Arg.is((s) => s.includes('Maintenance')),
      'verify'
    );
  });
});
```

```
interface MaintenanceWindow {
  isUnderMaintenance(): boolean;
}
```

```
describe('working with substitute part 2', () => {
  test('verify, during maintanance, calls logger', () => {
    const stubMaintWindow = Substitute.for<MaintenanceWindow>();
    stubMaintWindow.isUnderMaintenance().returns(true);
    const mockLog = Substitute.for<IComplicatedLogger>();
    const verifier = makeVerifierWithNoRules(mockLog, stubMaintWindow);

    verifier.verify('anything');

    mockLog.received().info('Under Maintenance', 'verify');
  });

  test('verify, outside maintanance, calls logger', () => {
    const stubMaintWindow = Substitute.for<MaintenanceWindow>();
    stubMaintWindow.isUnderMaintenance().returns(false);
    const mockLog = Substitute.for<IComplicatedLogger>();
    const verifier = makeVerifierWithNoRules(mockLog, stubMaintWindow);

    verifier.verify('anything');

    mockLog.received().info('PASSED', 'verify');
  });
});
```

```
interface MaintenanceWindow {
  isUnderMaintenance(): boolean;
}
```

寫測試...

到底要用手刻？還是用框架？

總結

- stub 沒有使用限制，但不要拿來驗證。
- 少用 mock，一個 test case 最多一個 mock。
- 依照專案狀況和使用的開發技術來選擇測試框架。

Thank you!