Mock

笔记本：	work in eisoo		
创建时间：	2019/8/19 9:48	**更新时间：**	2019/9/10 13:17
作者：	cathy.cai@eisoo.com		
URL：	https://jestjs.io/docs/zh-Hans/mock-functions		

Mock

在项目中，一个模块的方法内常常会去调用另外一个模块的方法。在单元测试中，我们可能并不关心调用方法的内部实现，只想知道它是否被调用或者能够返回相应的值即可。此时，使用Mock函数是十分有必要。

Mock 的简单使用

假如我们要测试函数`forEach` 的内部实现，这个函数为传入的数组中的每个元素调用一次回调函数：

```
export function forEach(items, callback) {
    for (let index = 0; index < items.length; index++) {
        callback(items[index]);
```

```
    }
}
```

为了测试此函数，我们可以使用mock,，然后检测mock 的状态来确保回调函数如期调用：

```
import {forEach} from './demo1'

it('模拟callback函数', () => {
    const mockCallback = jest.fn(x => 42 + x);
    forEach([0, 1], mockCallback);

    console.log(mockCallback.mock.calls)
    // [[0],[1]]

    expect(mockCallback.mock.calls.length).toBe(2);
    expect(mockCallback.mock.calls[0][0]).toBe(0);
    expect(mockCallback.mock.calls[1][0]).toBe(1);

    console.log(mockCallback.mock.results)
    // [ { type: 'return', value: 42 }, { type: 'return', value: 43 } ]
    expect(mockCallback.mock.results[0].value).toBe(42);
})
```

执行得到如下测试结果：

```
PASS demo1/demo1.test.js
  √ 模拟callback函数 (12ms)

  console.log demo1/demo1.test.js:9
    [ [ 0 ], [ 1 ] ]

  console.log demo1/demo1.test.js:15
    [ { type: 'return', value: 42 }, { type: 'return', value: 43 } ]

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.788s, estimated 2s
Ran all test suites.
Done in 2.75s.
```

Mock 的创建

jest.fn()

jest.fn(implementation) 是创建 mock 函数最简单的方式，它返回一个新的，未使用的 mock 函数。可以传入 implementation 来模拟函数内部实现，如果没有定义 implementation 则返回
undefined。

```
it('jest.fn', () => {
    const mockFn = jest.fn();
    mockFn();  // undefined;
    expect(mockFn).toHaveBeenCalled();

    // With a mock implementation:
    const returnsTrue = jest.fn(() => true);
    console.log(returnsTrue()); // true;
})
```

jest.mock()

jest.mock(moduleName, factory, options) 可以用于模拟一个模块。

假设我们有一个从API中获取用户的类。 该类使用axios调用API然后返回包含所有用户的data属性：

```
import axios from 'axios';

class Users {
    static all() {
        return axios.get('/users.json').then(resp => resp.data);
    }
}
export default Users;
```

为了在不实际访问API的情况下测试此方法，我们可以使用jest.mock(...)函数模拟axios模块。

一旦我们模拟了axios模块，我们就可以为.get提供一个mockResolvedValue，它返回我们希望测试要断言的数据。

```
import axios from 'axios';
import Users from './demo6';

jest.mock('axios');
```

```
test('should fetch users', async () => {
    const users = [{name: 'Bob'}];
    const resp = {data: users};

    axios.get.mockResolvedValue(resp);

    expect(await Users.all()).toEqual(users)
});
```

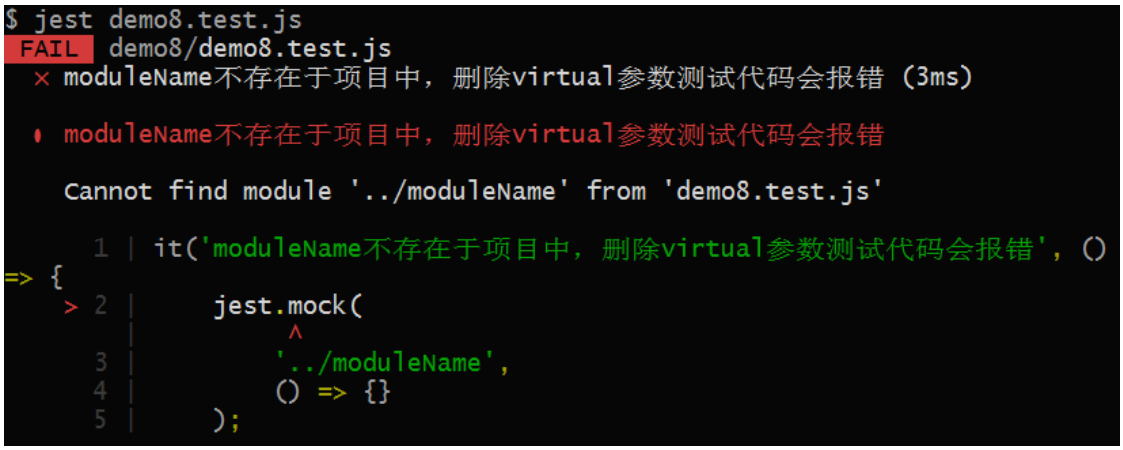
第二个参数可用于指定运行的函数，而不是使用Jest的自动模拟功能：

```
import Users from './demo7';

const users = [{name: 'Bob'}];

jest.mock('axios', () => {
    return {
        get: jest.fn().mockResolvedValue({
            data: [{
                name: 'Bob'
            }]
        })
    }
});

test('should fetch users', async () => {
    expect(await Users.all()).toEqual(users)
});
```

第三个参数可用于创建虚拟模拟 - 在系统中任何位置都不存在的模块的模拟。

```
it('moduleName不存在于项目中，删除virtual参数测试代码会报错', () => {
    jest.mock(
        '../moduleName',
        () => {}, {
            virtual: true
        },
    );
})
```

上述代码，添加virtual则测试通过，删除virtual后，提示moduleName找不到：



jest.spyOn()

jest.spyOn(object, methodName)可以创建一个类似于 jest.fn 的mock 函数，能够用于指定的methodName的模拟。

```
export const video = {
    play() {
        return true;
    },
};
```

```
import {video} from './demo9'

test('plays video', () => {
    const spy = jest.spyOn(video, 'play');
    const isPlaying = video.play();

    expect(spy).toHaveBeenCalled();
    expect(isPlaying).toBe(true);
});
```

Mock 的常用方法

.mock 属性

所有的 mock 函数都有 .mock 属性，它保存了关于此函数如何被调用，调用时的返回值等信息。

除开上述 mock.calls 以数组的方式记录调用参数信息，mock.results 以数组的方式记录调用结果信息，还有mock.instance 以数组的方式记录从该模拟函数实例化的所有对象实例。

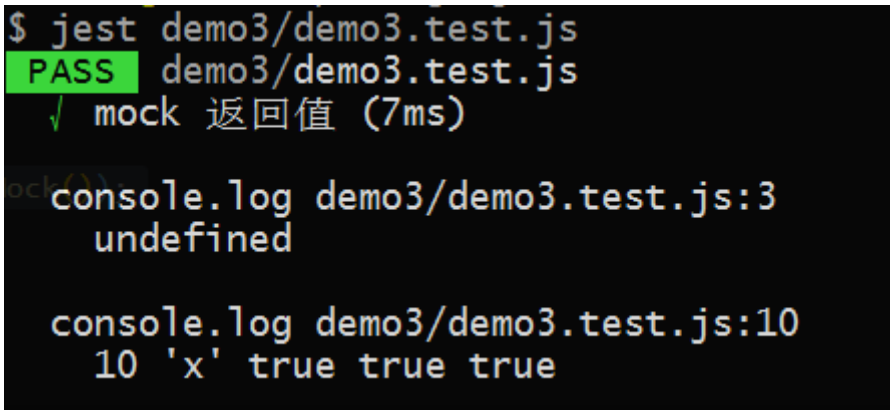
```
it('mock instance', () => {
    const myMock = jest.fn();

    const a = new myMock();
    const b = {};
    const bound = myMock.bind(b);
    bound();

    console.log(myMock.mock.instances);
    // > [ <a>, <b> ]

    expect(myMock.mock.instances[0]).toEqual(a);
    expect(myMock.mock.instances[1]).toEqual(b);
})
```

Mock 返回值

Mock 函数也可以指定返回值。

同步返回值

```
it('mock 返回值', () => {
    const myMock = jest.fn();
    console.log(myMock());
    // > undefined

    myMock.mockReturnValue(true)
        .mockReturnValueOnce(10)
        .mockReturnValueOnce('x')
    console.log(myMock(), myMock(), myMock(), myMock(), myMock());
    // > 10, 'x', true, true, true
})
```



上述代码中，.mockReturnValue(value) 调用 mock 函数的返回值。.mockReturnValueOnce(value) ，调用一次 mock 函数的返回值，可以链接，以便对 mock 函数的连续调用返回不同的值。

异步返回值

jest 还提供 .mockResolvedValue(value) .mockResolvedValueOnce(value) .mockRejectedValue(value) .mockRejectedValueOnce(value) 方法，用于在异步测试中返回值。

```
it('mock 异步 resolve', async () => {
const asyncMock = jest
    .fn()
    .mockResolvedValue('default')
    .mockResolvedValueOnce('first call')
    .mockResolvedValueOnce('second call');

    await asyncMock(); // first call
    await asyncMock(); // second call
    await asyncMock(); // default
})
```

```
it('mock 异步 reject', async () => {
    const asyncMock = jest
    .fn()
    .mockRejectedValue('Error Message')

    try {
        await asyncMock()
    } catch (error) {
        console.log(error) // "Error Message"
    };
})
```

Mock Implementations

还有一个比较常用的方法 .mockImplementation(fn) 。该方法接受一个 fn 参数，该函数被用作 mock 的实现。

```
it('mockImplementation', () => {
    const mockFn = jest.fn().mockImplementation(scalar => 42 + scalar);

    console.log(mockFn(0)); // 42
    console.log(mockFn(1)); // 43
})
```

Mock 清理

jest 提供了三种方式清理 mock：

- mockClear() ：重置存储在 mockFn.mock.calls 和 mockFn.mock.instances 数组中的所有信息。可用于清理不同断言之间 mock 的使用数据。
- mockReset()：重置存储在 mock 中的所有信息。可用于将 mock 完全重置回初始状态。
- mockRestore()：删除 mock 并恢复初始实现。可用于在某些测试用例中使用 mock 函数，并在其他测试用例中恢复原始实现。

练习

1. 练习使用Mock，并实现如下功能：

- 实现一个类，包含两个方法，一个
- 源代码包含一个异步 `Promise` ：当执行成功时，返回用户信息对象，如`userid` ，`tokenid` 等。当执行失败时，返回错误对象，如错误码，错误详情描述等
- 验证成功和失败情况返回的结果是否和预期相等
- 测试代码至少包含两种异步测试方式