测试源代码

笔记本：	work in eisoo		
创建时间：	2019/8/15 9:40	**更新时间：**	2019/8/16 15:32
作者：	cathy.cai@eisoo.com		
URL：	https://jestjs.io/docs/zh-Hans/asynchronous		

测试源代码

- 测试源代码
  - 一个简单的测试用例
  - 测试ES6
  - 异步测试
    - 回调
    - Promises
    - .resolves / .rejects
    - Async/Await
  - 练习

一个简单的测试用例

下面是一个加法模块的代码 `sum.js` ：

```
function sum(a, b) {
    return a + b;
}
module.exports = sum;
```

要测试这个加法模块是否正确，就需要测试脚本。通常测试脚本的命名测试的源码脚本同名，但后缀为`.test.js` 。比如，`sum.js` 的测试脚本名称就是`sum.test.js` ：

```
const sum = require('./sum');

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

执行测试脚本，得到如下测试结果：

```
PS C:\Users\cathy.cai\Desktop\UT_demo2> yarn test
yarn run v1.15.2
$ jest
 PASS  demo1/sum.test.js
  √ adds 1 + 2 to equal 3 (4ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        5.763s
Ran all test suites.
Done in 8.86s.
```

测试ES6

如果源代码和测试代码使用 ES6 风格，那么运行测试之前需要进行转码。 jest 支持使用 babel 转码，首先需要在项目中安装下面的依赖：

```
yarn add --dev babel-jest @babel/core @babel/preset-env
```

在项目的根目录中创建 babel.config.js 文件，用于配置与你当前 Node 版本兼容的 Babel ：

```
module.exports = {
    presets: [
        [
            '@babel/preset-env',
            {
                targets: {
                    node: 'current',
                },
            },
        ],
    ],
};
```

Babel 的配置取决于具体的项目使用场景 。可以查阅 Babel官方文档来获取更多详细的信息。

编写一个 ES6 风格的源代码：

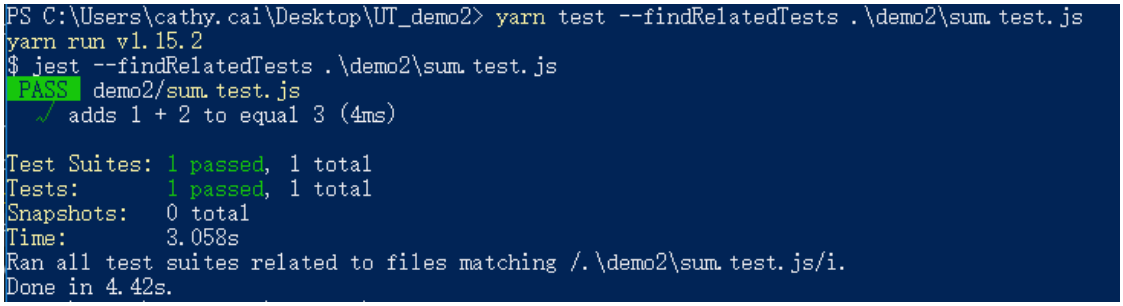
```
export const sum = (a, b) => {
    return a + b;
```

```
}
```

测试代码如下所示：

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

配置好 babel 后，运行测试，得到如下结果：



异步测试

在 JavaScript 中执行异步代码是很常见的。当你有以异步方式运行的代码时，Jest 需要知道它正在测试的代码何时完成，然后才能继续进行另一个测试。官方提供了一下几种方法处理异步测试。

回调

假如有一个 fetchData(callback) 函数，如下：

```
export const fetchData = (name, callback) => {
    setTimeout(() => {
        callback(`Hello ${name}`)
    }, 1000)
}
```

默认情况下，一旦达到运行上下文底部，jest 测试立即结束，而不会等待异步代码运行结束。因此按照下面方式书写的测试代码将无法按照预期工作。

```
import { fetchData } from './demo3'
// Don't do this!
it('本来应该失败的测试用例(断言Hello world===MyWorld) 执行成功了', () => {
```

```
    fetchData('world', (result) => {
        expect(result).toBe('MyWorld')
    })
});
```

上述预期失败的测试用例却得到了如下成功的测试结果：

```
PS C:\Users\cathy.cai\Desktop\UT_demo2> yarn test --findRelatedTests .\demo3\demo3.test.js
yarn run v1.15.2
$ jest --findRelatedTests .\demo3\demo3.test.js
 PASS  demo3/demo3.test.js
  √ 本来应该失败的测试用例(断言Hello world===MyWorld) 执行成功了 (1ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.927s, estimated 2s
Ran all test suites related to files matching /.\demo3\demo3.test.js/i.
Done in 3.02s.
```

jest 提供回调的方法解决上述问题。使用参数done 代替空的参数传入测试，jest 会等待 done() 函数执行结束后，才结束测试。

```
import {fetchData} from './demo4'
it('预期失败的测试用例(断言Hello world===MyWorld) 按照预期结果执行失败',
(done) => {
    fetchData('world', (result) => {
        expect(result).toBe('MyWorld')
        done()
    })
});
```

```
PS C:\Users\cathy.cai\Desktop\UT_demo2> yarn test --findRelatedTests .\demo4\demo4.test.js
yarn run v1.15.2
$ jest --findRelatedTests .\demo4\demo4.test.js
 FAIL  demo4/demo4.test.js
  × 预期失败的测试用例(断言Hello world===MyWorld) 按照预期结果执行失败 (1022ms)

  ● 预期失败的测试用例(断言Hello world===MyWorld) 按照预期结果执行失败

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

    Expected: "MyWorld"
    Received: "Hello world"

       5 | it('预期失败的测试用例(断言Hello world===MyWorld) 按照预期结果执行失败', (done) => {
       6 |     fetchData('world', (result) => {
    >  7 |         expect(result).toBe('MyWorld')
         |                        ^
       8 |         done()
       9 |     })
      10 | });

      at toBe (demo4/demo4.test.js:7:24)
      at callback (demo4/demo4.js:3:9)
```

需要注意的是，如果传入参数 done ，必须调用 done() 函数，没有调用，测试始终会等待 done() 的调用，直到超时失败。

Promises

如果在代码中使用 Promises ，可以用更简单的方式处理异步测试。只需要在测试中返回一个 promise ，jest 将等待该 promise 执行结果。

```
export const fetchData = (data) => {
    return new Promise((resolve, reject) => {
        if (data === 'peanut butter') {
            resolve(data)
        } else {
            reject('error')
        }
    })
}
```

```
import { fetchData } from './demo5'

it('peomise get data = peanut butter', () => {
    expect.assertions(1) // 验证在测试期间是否调用了一定数量的断言
    return fetchData('peanut butter').then((data) => {
        expect(data).toBe('peanut butter');
    })
});

it('promise catch error', () => {
    expect.assertions(1)
    return fetchData().catch((e) => {
        expect(e).toMatch('error')
    });
});
```

测试等待异步执行完成/失败后结束，得到预期测试结果：

```
PS C:\Users\cathy.cai\Desktop\UT_demo2> yarn test --findRelatedTests .\demo5\demo5.test.js
yarn run v1.15.2
$ jest --findRelatedTests .\demo5\demo5.test.js
 PASS  demo5/demo5.test.js
   √ peomise get data = peanut butter (4ms)
   √ promise catch error (1ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.598s
Ran all test suites related to files matching /.\demo5\demo5.test.js/i.
Done in 3.85s.
```

.resolves / .rejects

也可以使用 expect 中的 .resolves / .rejects 匹配器，jest 测试将等待 promise 执行结果。

```
import {fetchData} from './demo6'

it('异步函数，使用resolves', () => {
    expect.assertions(1) // 验证在测试期间是否调用了一定数量的断言
    return expect(fetchData('peanut butter')).resolves.toBe('peanut butter')
});
it('异步函数，使用rejects', () => {
    expect.assertions(1)
    return expect(fetchData()).rejects.toMatch('error');
});
```

测试等待异步执行完成/失败后结束，得到预期测试结果：

```
PS C:\Users\cathy.cai\Desktop\UT_demo2> yarn test --findRelatedTests .\demo6\demo6.test.js
yarn run v1.15.2
$ jest --findRelatedTests .\demo6\demo6.test.js
 PASS  demo6/demo6.test.js
  √ 异步函数，使用resolves (3ms)
  √ 异步函数，使用rejects (1ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.377s
Ran all test suites related to files matching /.\demo6\demo6.test.js/i.
Done in 3.85s.
```

Async/Await

Jest 也支持 async /await 语法的编写测试用例，无需多余的操作，只要在 await 后进行断言即可，和同步测试的写法一致。

```
import {fetchData} from './demo7'

it('异步函数，使用async/await 验证正常结果', async () => {
    expect.assertions(1)
    const data = await fetchData('peanut butter')
    expect(data).toBe('peanut butter')
});
it('异步函数，使用async/await 测试捕获异常', async () => {
    expect.assertions(1)
    try {
        await fetchData()
    } catch (error) {
        expect(error).toMatch('error');
```

```
    }
});
```

测试等待异步执行完成/失败后结束，得到预期测试结果：

```
PS C:\Users\cathy.cai\Desktop\UT_demo2> yarn test --findRelatedTests .\demo7\demo7.test.js
yarn run v1.15.2
$ jest --findRelatedTests .\demo7\demo7.test.js
 PASS  demo7/demo7.test.js
  √ 异步函数，使用async/await 验证正常结果 (4ms)
  √ 异步函数，使用async/await 测试捕获异常 (1ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.22s
Ran all test suites related to files matching /.\demo7\demo7.test.js/i.
Done in 4.10s.
```

练习

1. 配置支持 ES6 风格的测试环境
2. 实现一个异步测试，要求：

- 使用 ES6 风格编写代码
- 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