安装与配置

笔记本：	work in eisoo		
创建时间：	2019/8/13 15:38	**更新时间：**	2019/9/10 14:20
作者：	cathy.cai@eisoo.com		
URL：	https://zhuanlan.zhihu.com/p/28162082		

jest 入门

jest 是 Facebook 出品的一个测试框架。相比其他测试框架，其一大的特点就是内置常用的测试工具，比如自带断言，测试覆盖率工具等。

安装

jest 可以通过 yarn 或者 npm 进行安装：

```
yarn add --dev jest
```

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

其中使用`--dev --save-dev`参数，表明依赖作为`devDependencies`，即只安装在开发环境下，不在生产环境中安装。

在项目的`package.json`中添加如下配置，即可在项目中使用`yarn test`或`npm run test`运行测试。

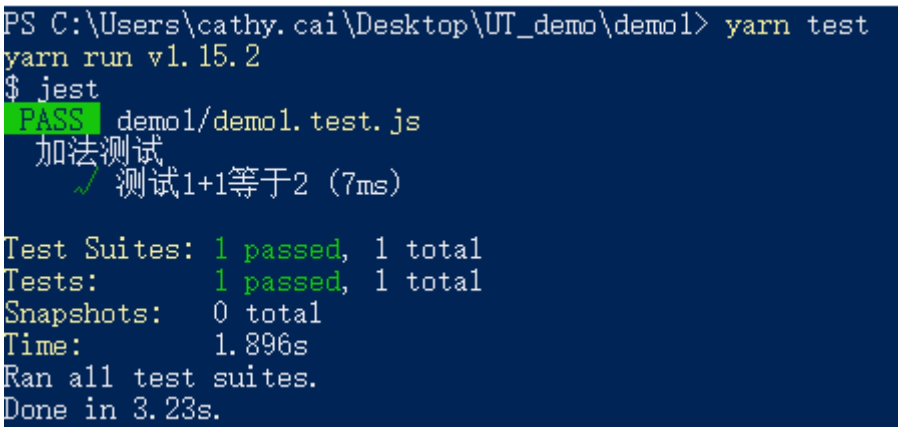
```
{
    "scripts": {
        "test": "jest"
    }
}
```

也可以通过`yarn global add jest`或`npm install jest --global`全局安装`jest`，安装成功后，可以通过命令行直接运行`jest`。

基本使用

```
describe('加法测试',()=>{
    it('测试1+1等于2', () => {
        expect(1 + 1).toBe(2);
    });
})
```

上述代码，就是一个基本的测试脚本，他能够独立运行。执行`yarn test`，得到如下测试结果：



describe 和 it

describe

describe(name, fn) 块称为**测试套件**（test suite）,表示一组相关测试。它有两个参数 name 和 fn ，name 是测试套件标题，string 类型。fn 是测试套件要执行的函数。
比如有好几个加法测试，就可以放在一个 describe 中进行描述。

```
describe('加法测试',()=>{
    it('测试1+1等于2', () => {
        expect(1 + 1).toBe(2);
    });

     it('测试2+2等于4', () => {
        expect(2 + 2).toBe(4);
    });
})
```

其输出结果如下：

```
PS C:\Users\cathy.cai\Desktop\UT_demo\demo2> jest demo2.test.js
 PASS  ./demo2.test.js
  加法测试
    √ 测试1+1等于2 (5ms)
    √ 测试2+2等于4

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.777s
Ran all test suites matching /demo2.test.js/i.
```

it

it(name, fn, timeout) 块称为**测试用例**（test case），表示一个单独的测试，是测试的最小单位，也可以使用别名test(name,fn,timeout) 。它有三个参数，name 和 fn 同 describe 一样，表示测试用例的名称和实际执行的函数，timeout 是一个可选参数，代表该条测试用例的测试超时时间，默认超时时间为5s。
下面是一个timeout 参数使用示例：

```
describe('timeout 参数测试', () => {
    it('测试 不超时（测试用例执行成功），指定的超时时间大于代码执行时间',
(done) => {
```

```
        setTimeout(() => {
            done()
        }, 500);
    }, 600);

    it('测试 超时（测试用例执行失败），指定的超时时间小于代码执行时间',
(done) => {
        setTimeout(() => {
            done()
        }, 500);
    }, 400);
})
```

当代码的执行时间小于设定的超时时间时，测试用例执行成功，当代码的执行时间大于设定的超时时间时，测试用例执行失败。运行结果如下所示：

.each

当有多个测试数据，需要重复的执行同一测试套件时，可以使用`describe.each` 一次传入数据。它有两种使用方式：
一种是`describe.each(table)(name, fn, timeout)` ，其中：

- `table` 是测试数据数组，该数组将作为参数传递给`fn` 。
- `name` 表示测试套件的标题，可以通过 `printf` 格式注入来生成唯一的测试标题，其规则如下：
  - %p - pretty-format.
  - %s- String.
  - %d- Number.
  - %i - Integer.
  - %f - Floating point value.
  - %j - JSON.

    - %o - Object.
    - %# - Index of the test case.
    - %% - single percent sign ('%'). This does not consume an argument.
- `fn` 是实际执行的测试函数，可以接收 `table` 中传递过来的测试数据。
- `timeout` 是可选参数，表示该测试套件超时时间。

以下是一个`describe.each` 的使用示例：

```
describe.each([
    [1, 1, 2],
    [1, 2, 3],
    [2, 1, 3]
])('测试%i+%i=%i', (a, b, expected) => {
        it(`returns ${expected}`, () => {
            expect(a + b).toBe(expected);
        });
    },
);
```

执行测试用例后得到如下结果。总共有3组测试数据，通过一个测试套件得到了3组运行结果。

```
PS C:\Users\cathy.cai\Desktop\UT_demo\demo4> jest demo4.test.js
 PASS  ./demo4.test.js
  测试1+1=2
    √ returns 2 (3ms)
  测试1+2=3
    √ returns 3 (1ms)
  测试2+1=3
    √ returns 3

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.061s
Ran all test suites matching /demo4.test.js/i.
```

`describe.each` 还有另外一种调用方式：

```
describe.each`table`(name, fn, timeout)
```

- `table` 使用一种标记模板文字，第一行用 `|` 分隔变量名，后面的一行或者多行是使用 `${value}` 语法为模板表达式提供的数据。
- `name` 表示测试套件的标题，使用 `$variable` 将测试数据从标记模板中注入套件标题。
- `fn` 是实际执行的测试函数，可以接收 `table` 中传递过来的测试数据。
- `timeout` 是可选参数，表示该测试套件超时时间。

其使用方法如下所示：

```
describe.each `
    a | b | expected
    ${1} | ${1} | ${2}
    ${1} | ${2} | ${3}
    ${2} | ${1} | ${3}
`('测试$a+$b=$expected', ({a,b,expected}) => {
    it(`returns ${expected}`, () => {
        expect(a + b).toBe(expected);
    });
});
```

运行结果如下所示：

```
PS C:\Users\cathy.cai\Desktop\UT_demo\demo5> jest demo5.test.js
 PASS  ./demo5.test.js
  测试1+1=2
    √ returns 2 (3ms)
  测试1+2=3
    √ returns 3
  测试2+1=3
    √ returns 3 (1ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.2s
Ran all test suites matching /demo5.test.js/i.
```

`it` 也有 `.each` 方法 `test.each(table)(name, fn, timeout)` ，使用方法和`describe` 类似。

.only

`describe.only(name,fn,timeout)` 、`it.only(name, fn, timeout)` 指定在测试中唯一需要运行的测试套件或测试用例。

```
describe('验证 .only 运行唯一的一条测试用例', () => {
    it('没有only，不运行的测试用例1', () => {
        expect(1 + 1).toBe(2);
    });
    it('没有only，不运行的测试用例2', () => {
        expect(1 + 1).toBe(2);
    });
    it.only('包含only,唯一运行的测试用例', () => {
        expect(1 + 1).toBe(2);
    });
})
```

执行测试发现，只有带有only 标志的测试用例被运行，其他两条测试用例被跳过。

```
PS C:\Users\cathy.cai\Desktop\UT_demo\demo6> jest demo6.test.js
 PASS  ./demo6.test.js
  验证 .only 运行唯一的一条测试用例
    √ 包含only,唯一运行的测试用例 (4ms)
    ○ skipped 没有only，不运行的测试用例1
    ○ skipped 没有only，不运行的测试用例2

Test Suites: 1 passed, 1 total
Tests:       2 skipped, 1 passed, 3 total
Snapshots:   0 total
Time:        2.768s
Ran all test suites matching /demo6.test.js/i.
```

当调试大型测试文件时，可以用于修复损坏的测试文件，调试完成后，应该将only标志删除。

.skip

describe.skip(name, fn) ，it.skip(name, fn) 在测试中跳过某一测试套件或者测试用例。

```
describe('验证 .skip 跳过一条测试用例', () => {
    it('没有skip，运行的测试用例1', () => {
        expect(1 + 1).toBe(2);
    });
    it.skip('有skip，跳过的测试用例2', () => {
        expect(1 + 1).toBe(2);
    });
    it('没有skip，运行的测试用例3', () => {
        expect(1 + 1).toBe(2);
    });
})
```

执行测试发现，带有skip 标志的测试用例被跳过，其他两条测试用例正常执行。

```
PS C:\Users\cathy.cai\Desktop\UT_demo\demo7> jest demo7.test.js
 PASS  ./demo7.test.js
  验证 .skip 跳过一条测试用例
    √ 没有skip，运行的测试用例1 (4ms)
    √ 没有skip，运行的测试用例3
    ○ skipped 有skip，跳过的测试用例2

Test Suites: 1 passed, 1 total
Tests:       1 skipped, 2 passed, 3 total
Snapshots:   0 total
Time:        2.278s
Ran all test suites matching /demo7.test.js/i.
```

当维护大型测试代码库时，由于某些原因需要中断测试但又不想删除此测试代码时，可以使用skip 代替注释跳过测试。

it.todo

`it.todo(name)` 当你有计划编写的测试用例时可以放到 `todo` 中。

```
describe('todo 用法示例', () => {
    it('已经完成的测试用例', () => {
        expect(1 + 1).toBe(2);
    });
    it.todo('待编写的测试用例');
})
```

在执行测试时，会突出显示，方便了解还有多少测试用例需要编写。

```
PS C:\Users\cathy.cai\Desktop\UT_demo\demo8> jest demo8.test.js
 PASS  ./demo8.test.js
  todo 用法示例
    √ 已经完成的测试用例 (4ms)
    todo 待编写的测试用例

Test Suites: 1 passed, 1 total
Tests:       1 todo, 1 passed, 2 total
Snapshots:   0 total
Time:        1.959s, estimated 2s
Ran all test suites matching /demo8.test.js/i.
```

断言库

在编写测试用例时，需要检测执行结果与预期结果是否一致。断言就是提供这种功能，让使用者方便的验证不同的事情。

expect

每次要测试一个值时，都会使用 `expect(value)` 函数。其参数 `value` 应该是代码产生的值。

toBe

`toBe(value)` 用于比较原始值或者对象实例的引用标识。

```
const person = {
    name: 'Bob',
```

```
    age: 12,
};

describe('toBe 判断对象实例的引用值相等', () => {
    it('person 的name属性值为 Bob', () => {
        expect(person.name).toBe('Bob');
    });
    it('person 的age 属性值为 12', () => {
        expect(person.age).toBe(12);
    });
});
```

```
PS C:\Users\cathy.cai\Desktop\UT_demo\demo9> jest demo9.test.js
 PASS  ./demo9.test.js
  toBe 判断对象实例的引用值相等
    √ person 的name属性值为 Bob (4ms)
    √ person 的age 属性值为 12 (1ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.873s
Ran all test suites matching /demo9.test.js/i.
```

toEqual

`toEqual(value)` 以递归的方式比较对象实例的所有属性，也称为深比较。

```
const person1 = {
    name: 'Bob',
    age: 12,
};
const person2 = {
    name: 'Bob',
    age: 12,
};
describe('toBe 和 toEqual 比较', () => {
    it('toBe 无法比较两个对象是否相等', () => {
        expect(person1).toBe(person2);
    });
    it('toEqual 可以比较两个对象是否相等', () => {
        expect(person1).toEqual(person2);
    });
});
```

```
PS C:\Users\cathy.cai\Desktop\UT_demo\demo10> jest demo10.test.js
 FAIL  ./demo10.test.js
  toBe 和 toEqual 比较
    × toBe 无法比较两个对象是否相等 (9ms)
    √ toEqual 可以比较两个对象是否相等 (1ms)

  ● toBe 和 toEqual 比较 › toBe 无法比较两个对象是否相等

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

    If it should pass with deep equality, replace "toBe" with "toStrictEqual"

    Expected: {"age": 12, "name": "Bob"}
    Received: serializes to the same string

      11 | describe('toBe 和 toEqual 比较', () => {
      12 |     it('toBe 无法比较两个对象是否相等', () => {
    > 13 |         expect(person1).toBe(person2);
         |
      14 |     });
      15 |
      16 |     it('toEqual 可以比较两个对象是否相等', () => {

      at Object.toBe (demo10/demo10.test.js:13:25)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 passed, 2 total
Snapshots:   0 total
Time:        2.13s
Ran all test suites matching /demo10.test.js/i.
```

not

not() 表示某一事物的反面。

```
describe('not 表示事物的反面，当测试中遇到不好测试的其正面成立的例子时，可以测试其反面不成立', () => {
    it('测试用例 1+1 等于 3 不成立 ', () => {
        expect(1 + 1).toBe(3);
    });
    it('测试用例 1+1 不等于 3 成立', () => {
        expect(1 + 1).not.toBe(3);
    });
});
```

toContain

toContain(item) 检查项目是否包含在数组中或者一个字符串是否包含在另外一个字符串中。

```
describe('toContain', () => {
    it('检查项目是否包含在数组中', () => {
        expect([1, 2, 3, 4]).toContain(1);
    });
    it('检查字符串是否包含在另一字符串中', () => {
        expect('can1').toContain('can');
    });
});
```

```
PS C:\Users\cathy.cai\Desktop\UT_demo\demo12> jest demo12.test.js
 PASS  ./demo12.test.js
  toContain
    √ 检查项目是否包含在数组中 (4ms)
    √ 检查字符串是否包含在另一字符串中 (1ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.311s
Ran all test suites matching /demo12.test.js/i.
```

`jest` 断言库还为大家提供了许多丰富的API，此处只是简单介绍了几种常用的方法，更多方法将在后续逐步进行介绍或者自行参考官方文档jest 断言库。

练习

1. 安装 `jest`
2. 编写一个测试用例，该测试用例需要包含以下功能：

- 验证乘法功能
- 包含3组测试数据
- 选择合适的断言