Jest 快速参考

测试、断言、模拟、异步与快照

配置

安装

```
npm install --save-dev jest
# package.json: "scripts": { "test": "jest" }
npx jest              # run all tests
npx jest --watch      # re-run on changes
```

文件命名

`*.test.js`	测试文件（默认匹配模式）
`*.spec.js`	备用测试文件模式
`__tests__/`	测试目录（自动发现）

运行指定测试

```
npx jest path/to/file.test.js
npx jest --testNamePattern="adds"
npx jest --verbose     # detailed output
```

基础测试

测试结构

```
describe("Calculator", () => {
  test("adds 1 + 2 to equal 3", () => {
    expect(add(1, 2)).toBe(3);
  });
});
```

test 与 it

```
test("works correctly", () => { /* ... */ });
it("should work correctly", () => { /* ... */ });
// Both are identical; "it" reads like English
```

跳过与聚焦

`test.skip()`	跳过此测试
`test.only()`	仅运行此测试
`describe.skip()`	跳过整个测试套件
`describe.only()`	仅运行此测试套件

断言（Matchers）

相等性

`.toBe(val)`	严格相等（`===`）
`.toEqual(val)`	深度相等（对象/数组）
`.toStrictEqual(val)`	深度 + 类型 + undefined 属性
`.not.toBe(val)`	取反任意断言

真值判断

`.toBeTruthy()`	值为真
`.toBeFalsy()`	值为假
`.toBeNull()`	严格为 **null**
`.toBeUndefined()`	严格为 **undefined**
`.toBeDefined()`	不为 **undefined**

数值

`.toBeGreaterThan(n)`	大于 n
`.toBeLessThanOrEqual(n)`	小于等于
`.toBeCloseTo(0.3, 5)`	浮点数比较（5 位精度）

字符串、数组、对象

`.toMatch(/regex/)`	字符串匹配正则
`.toContain(item)`	数组/可迭代对象包含元素
`.toHaveLength(n)`	数组或字符串的长度
`.toHaveProperty(key, val)`	对象包含指定属性
`.toMatchObject(obj)`	对象包含指定子集

异步测试

async / await

```
test("fetches data", async () => {
  const data = await fetchData();
  expect(data).toEqual({ id: 1 });
});
```

Promise

```
test("resolves to data", () => {
  return expect(fetchData())
    .resolves.toEqual({ id: 1 });
});
```

拒绝（Rejection）

```
test("rejects with error", async () => {
  await expect(fetchBad())
    .rejects.toThrow("Not Found");
});
```

异常

```
test("throws on invalid input", () => {
  expect(() => validate(null)).toThrow();
  expect(() => validate(null)).toThrow("invalid");
});
```

模拟（Mocking）

模拟函数

```
const fn = jest.fn();
fn("hello");
expect(fn).toHaveBeenCalledWith("hello");
expect(fn).toHaveBeenCalledTimes(1);
```

模拟返回值

```
const fn = jest.fn()
  .mockReturnValue(42)
  .mockReturnValueOnce(99);
fn(); // 99 (first call)
fn(); // 42 (subsequent)
```

模拟模块

```
jest.mock("./api");
const { fetchUser } = require("./api");
fetchUser.mockResolvedValue({ name: "Alice" });
```

模拟断言

`.toHaveBeenCalled()`	至少被调用一次
`.toHaveBeenCalledTimes(n)`	恰好被调用 n 次
`.toHaveBeenCalledWith(args)`	以指定参数被调用
`.toHaveBeenLastCalledWith(args)`	最后一次调用的参数

间谍（Spies）

监听方法

```
const spy = jest.spyOn(Math, "random")
  .mockReturnValue(0.5);
expect(Math.random()).toBe(0.5);
spy.mockRestore();  // restore original
```

监听对象方法

```
const obj = { greet: (n) => `Hi ${n}` };
const spy = jest.spyOn(obj, "greet");
obj.greet("Alice");
expect(spy).toHaveBeenCalledWith("Alice");
```

快照（Snapshots）

快照测试

```
test("renders correctly", () => {
  const tree = renderer.create(<App />).toJSON();
  expect(tree).toMatchSnapshot();
});
```

内联快照

```
test("formats name", () => {
  expect(formatName("alice"))
    .toMatchInlineSnapshot(`"Alice"`);
});
```

更新快照

```
npx jest --updateSnapshot   # update all
npx jest --updateSnapshot --testNamePattern="renders"
```

Setup & Teardown

生命周期钩子

```
beforeAll(() => { /* once before all tests */ });
afterAll(() => { /* once after all tests */ });
beforeEach(() => { /* before each test */ });
afterEach(() => { /* after each test */ });
```

作用域

```
describe("Database", () => {
  beforeEach(() => db.connect());
  afterEach(() => db.disconnect());
  test("reads data", () => { /* ... */ });
});
```

describe 内的钩子仅作用于该块

配置选项

jest.config.js

```
module.exports = {
  testEnvironment: "node",
  coverageThreshold: {
    global: { branches: 80, lines: 80 }
  },
};
```

常用选项

`testEnvironment`	**"node"** 或 **"jsdom"**（DOM 环境）
`roots`	搜索测试文件的目录
`collectCoverage`	启用覆盖率报告
`coverageDirectory`	覆盖率报告输出目录
`moduleNameMapper`	路径别名（如 **@/** 前缀）
`transform`	文件转换（Babel、TS 等）
`setupFilesAfterFramework`	每个测试套件前执行的初始化

Jest 快速参考

覆盖率

```
npx jest --coverage
npx jest --collectCoverageFrom="src/**/*.js"
```

常见模式

测试 API 调用

```
jest.mock("./api");
test("loads users", async () => {
  api.getUsers.mockResolvedValue([{id: 1}]);
  const users = await loadUsers();
  expect(users).toHaveLength(1);
});
```

定时器模拟

```
jest.useFakeTimers();
test("delays execution", () => {
  const cb = jest.fn();
  setTimeout(cb, 1000);
  jest.advanceTimersByTime(1000);
  expect(cb).toHaveBeenCalled();
});
```

参数化测试

```
test.each([
  [1, 1, 2],
  [2, 3, 5],
])("add(%i, %i) = %i", (a, b, expected) => {
  expect(add(a, b)).toBe(expected);
});
```

自定义断言

```
expect.extend({
  toBeWithinRange(received, floor, ceil) {
    const pass = received >= floor
      && received <= ceil;
    return { pass, message: () =>
      `expected ${received} in [${floor},${ceil}]` };
}});
```