16. 项目测试

单元测试

1. 概述：单元测试是指开发人员针对某个**模块、函数或组件**编写的测试代码。与之相对的，系统测试是指测试人员针对整个系统的功能流程编写的测试代码。
2. 单元测试工具：jest
   a. 安装 `npm install --save-dev jest`
   b. 配置 npm scripts

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

   c. 编写单元测试：对于文件 xxx.ts，其对应的单元测试文件应命名为 xxx.test.ts
      i. 测试用例：使用 `test` 函数（或 `it`）可以创建一个测试用例，其接收两个参数 name、fn，前者表示测试名称，后者表示包含断言的函数。
      ii. 断言：使用 `expect(计算得到的数据).toXxx(期望得到的数据)` 可以创建一个断言，表示计算结果应该与预期相匹配，如果断言全部通过，则该测试用例 PASS，否则 FAIL。
      iii. 为了让 React 组件可以在测试用例中单独渲染，需要使用
         - Step1. 安装相关库

```
npm install --save-dev jest babel-jest @babel/preset-env
@babel/preset-react @babel/preset-typescript @testing-
library/react @testing-library/jest-dom @types/jest ts-jest jest-
environment-jsdom
```

         - Step2. 在项目根目录创建 `.json` 文件来配置 Babel

```
{
  "presets": [
    "@babel/preset-env",
```

```
    "@babel/preset-react",
    "@babel/preset-typescript"
  ]
}
```

- Step3. 在项目根目录创建 `jest.config.ts` 文件，配置 Jest

```ts
// jest.config.ts
import type { Config } from 'jest';

const config: Config = {
  testEnvironment: 'jest-environment-jsdom',
  transform: {
    '^.+\\.tsx?$': 'babel-jest',
  },
  moduleNameMapper: {
    '\\.(css|less|scss|sass)$': 'identity-obj-proxy',
  },
  setupFilesAfterEnv: ['<rootDir>/setupTests.ts'],
  testMatch: ['<rootDir>/src/**/*.{test,spec}.{ts,tsx}'],
};

export default config;
```

- Step4. 在项目根目录创建 `setupTests.ts` 文件，用于全局引入 `@testing-library/jest-dom`

```ts
// setupTests.ts
import '@testing-library/jest-dom';
```

d. 通过 npm scripts 执行单元测试

3. 测试文件的位置
   a. 选择1：与 src 同级的 __test__ 目录下
   b. 选择2：和源码文件放在一起，使用 .test.ts 的后缀命名
4. 单元测试 - 自动化：即在 husky 的 pre-commit 文件中添加一行内容 `npm run test`

可视化测试

1. 采用工具 StoryBook

a. 安装 `npx storybook@latest init`

b. 运行 `npm run storybook`