项目测试

前言

本文主要讲述了前端测试在React（create-react-app）项目中的应用，本文将会包含以下内容：

- 单元测试（Jest）
- 端到端测试（Cypress）

测试分类

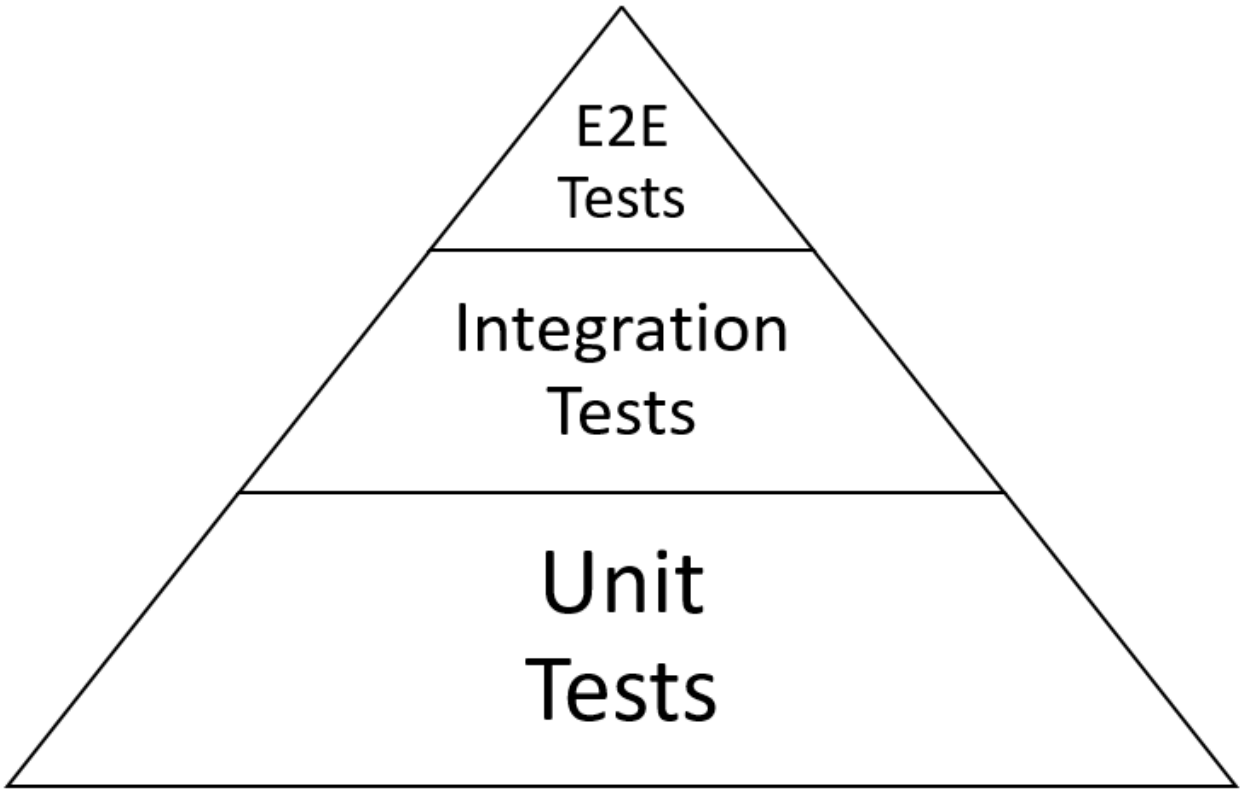
常见的开发流程里，都有测试人员，他们不管内部实现机制，只看最外层的输入输出，这种我们称为**黑盒测试**。比如你写一个加法的页面，会设计N个用例，测试加法的正确性，这种测试我们称之为**E2E**测试。

还有一种测试叫做**白盒测试**，我们针对一些内部核心实现逻辑编写测试代码，称之为**单元测试**。 更负责一些的我们称之为**集成测试**，就是集合多个测试过的单元一起测试。

编写测试的好处

- 提供描述组件行为的文档
- 节省手动测试的时间
- 减少研发新特性时产生的 bug 改进设计
- 促进重构

> 自动化测试使得大团队中的开发者可以维护复杂的基础代码。让你改代码不再小心翼翼



单元测试

准备工作

在React中，推荐用Mocha或者Jest，演示代码使用Jest，它们语法基本一致 。

> **Jest** 是一个 JavaScript 测试运行器。它允许你使用 `jsdom` 操作 DOM 。尽管 jsdom 只是对浏览器工作表现的一个近似模拟，对测试 React 组件来说它通常也已经够用了。Jest 有着十分优秀的迭代速度，同时还提供了若干强大的功能，比如它可以模拟 modules 和 timers 让你更精细的控制代码如何执行

> Jest 与 React 项目广泛兼容，支持诸如模拟 模块、计时器 和 `jsdom` 等特性。**如果你使用 Create React App，Jest 已经能够开箱即用且包含许多实用的默认配置。**

由于create-react-app脚手架已经集成Jest。所以，直接开干。

```
$ create-react-app hello-demo
```

编写单元测试

单元测试(unit testing)，是指对软件中的最小可测试单元进行检查和验证。

- 新建src/pages/demo1.test.js， *.test.js 是命名规范

```
function add(num1, num2) {
  return num1 + num2;
}
```

```
// 测试套件 test suite
describe("函数测试", () => {
  // 测试用例 test case
  it("测试add函数", () => {
    // 断言 assert
    expect(add(1, 3)).toBe(4);
    expect(add(1, 3)).toBe(5);
    expect(add(-2, 3)).toBe(1);
  });
});
```

Note

Jest 将使用以下任何流行的命名约定来查找测试文件：

- `__tests__` 文件夹中带有 `.js` 后缀的文件。
- 带有 `.test.js` 后缀的文件。
- 带有 `.spec.js` 后缀的文件。

> 我们建议将测试文件（或 `__tests__` 文件夹）放在他们正在测试的代码旁边

执行单元测试

- 执行: `yarn test`

断言API简介

- `describe`: 定义一个测试套件
- `it`: 定义一个测试用例
- `expect` : 断言的判断条件

> 这里面仅演示了toBe，更多断言API

测试React组件

React 官方提供了用于单元测试的实用工具库 @testing-library/react（CRA项目已集成该库，所以不用重新添加）。

创建一个React组件

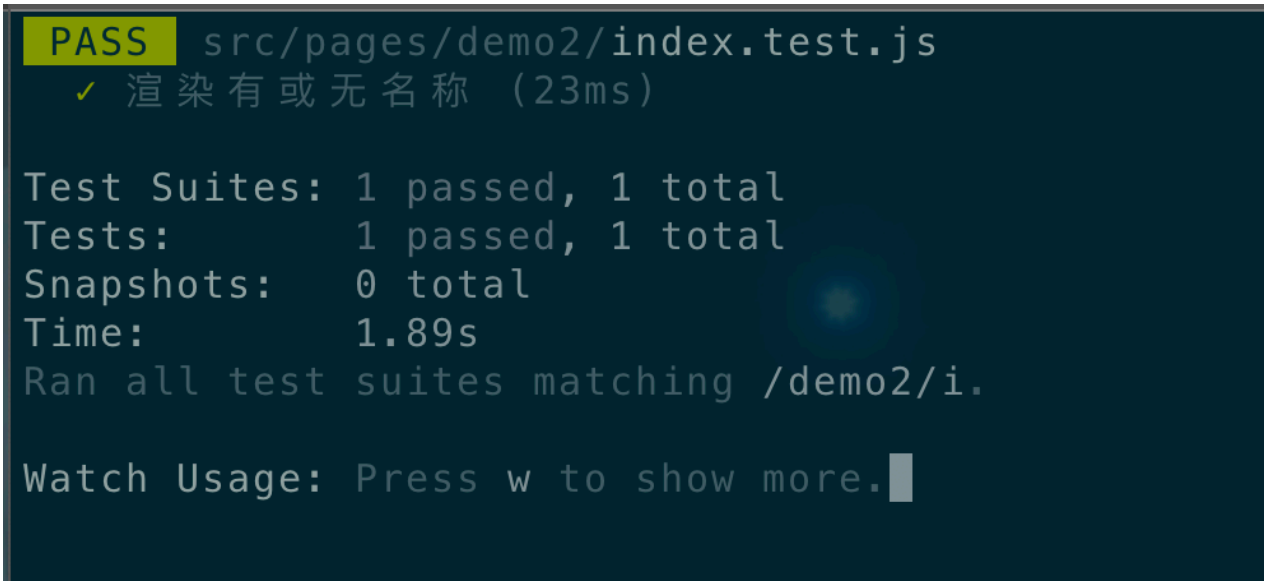
```
import React from "react";
```

```
export function add(params) {
  return params;
}

const JestPage = props => {
  if (props.name) {
    return <h1>你好，{props.name}！</h1>;
  } else {
    return <span>嘿，陌生人</span>;
  }
};

export default JestPage;
```

测试该组件

```
import React from "react";
import { render, unmountComponentAtNode } from "react-dom";
import { act } from "react-dom/test-utils";

import Hello from "./index";

let container = null;
beforeEach(() => {
  // 创建一个 DOM 元素作为渲染目标
  container = document.createElement("div");
  document.body.appendChild(container);
});

afterEach(() => {
  // 退出时进行清理
  unmountComponentAtNode(container);
  container.remove();
  container = null;
});

it("渲染有或无名称", () => {
  act(() => {
    render(<Hello />, container);
  });
  expect(container.textContent).toBe("嘿，陌生人");

  act(() => {
    render(<Hello name="Jenny" />, container);
  });
  expect(container.textContent).toBe("你好，Jenny！");

  act(() => {
```

```
    render(<Hello name="Margaret" />, container);
  });
  expect(container.textContent).toBe("你好，Margaret！");
});
```



测试覆盖率

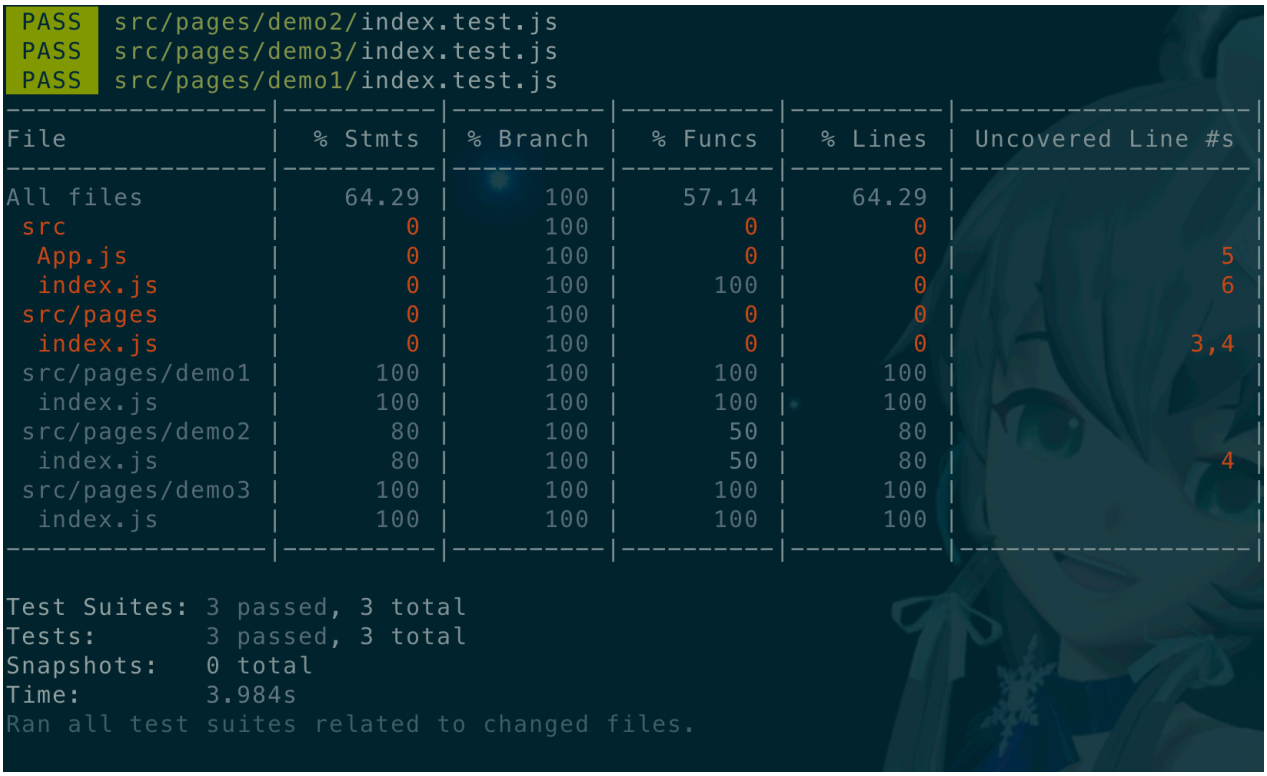
jest自带覆盖率，如果用的mocha，需要使用istanbul来统计覆盖率

- 方法一、package.json里修改jest配置 （需要eject）

```
"collectCoverage": true,
"collectCoverageFrom": [
  "src/**/*.{js,jsx,ts,tsx}",
  "!src/**/*.d.ts"
],
```

> - 方法二、若采用独立配置，则修改jest.config.js:
>
> ```
> module.exports = {
>  "collectCoverage": true,
>  "collectCoverageFrom": ["src/**/*.{js,vue}"]
> }
> ```

- 方法三、在此执行 `yarn test -- --coverage`



> %stmts是语句覆盖率(statement coverage):是不是每个语句都执行了?
>
> %Branch分支覆盖率(branch coverage):是不是每个if代码块都执行了?
>
> %Funcs函数覆盖率(function coverage):是不是每个函数都调用了?
>
> %Lines行覆盖率(line coverage):是不是每一行都执行了?

> create-react-app 测试
>
> react 测试

E2E测试

借用浏览器的能力，站在用户测试人员的角度，输入框，点击按钮等，完全模拟用户，这个和具体的框架关系不大，完全模拟浏览器行为。

在React中，推荐用puppeteer或者cypress做端到端测试，它们语法基本一致 。本文主要介绍 cypress

> create-react-app中使用cypress

准备工作

- 安装

```
$ create-react-app my-new-app
$ npm i -D @cypress/instrument-cra
$ npm i @cypress/code-coverage nyc istanbul-lib-coverage cypress
```

- 修改`package.json`

```
{
  "start-dev": "react-scripts -r @cypress/instrument-cra start",
  "cypress": "cypress open",
}
```

- 修改`cypress/support/index.js`文件

```
import '@cypress/code-coverage/support'
```

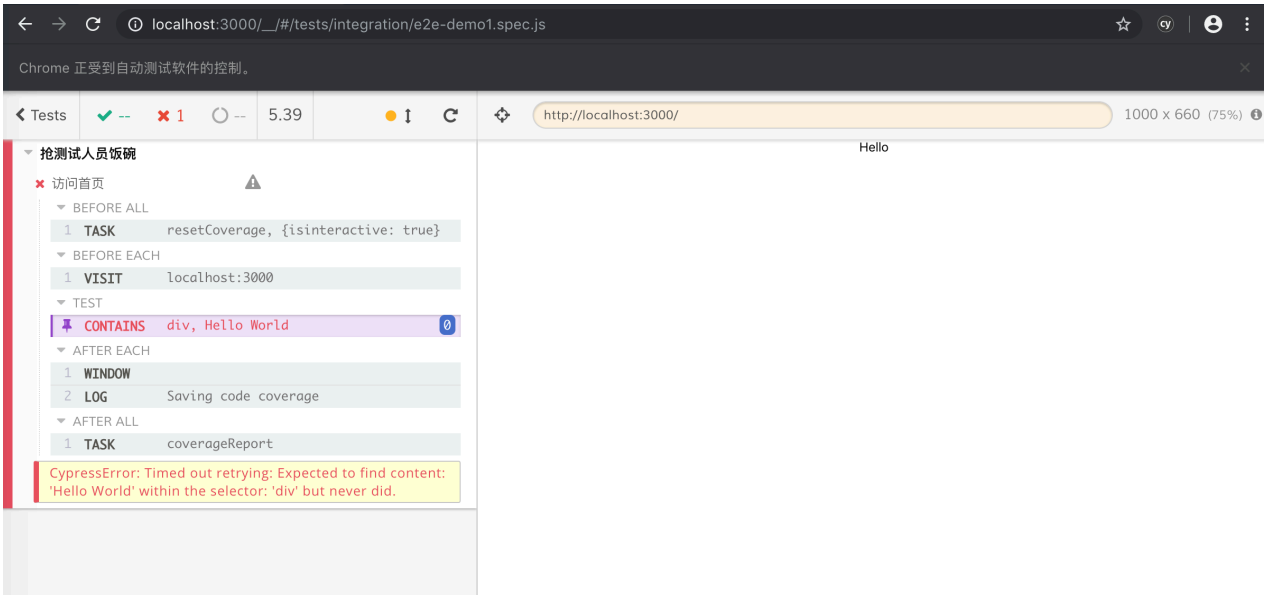
- 修改`cypress/plugins/index.js`文件

```
module.exports = (on, config) => {
  on('task', require('@cypress/code-coverage/task'))
}
```

编写端到端测试

新建`/cypress/integration/e2e-demo1.spec.js`文件

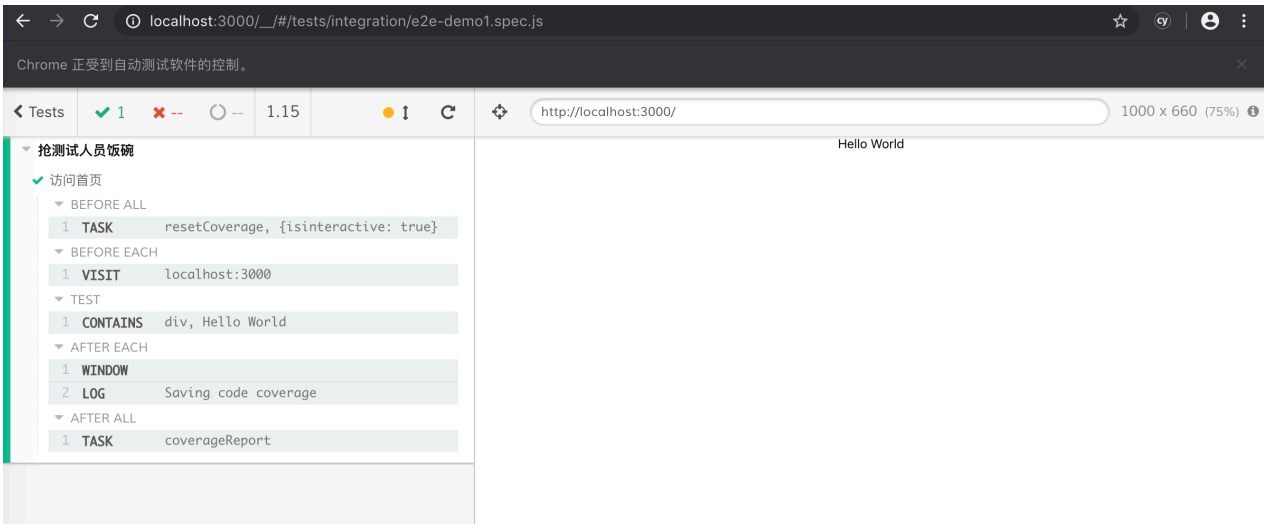
```
// https://docs.cypress.io/api/introduction/api.html

describe("My First Test", () => {
  beforeEach(() => {
    cy.visit("localhost:3000");
  });
  it("Visits the app root url", () => {
    cy.contains("div", "Hello World");
  });
});
```



修改`App.js`

```
import React from "react";
import "./App.css";

function App() {
  return (
    <div className="App">
      Hello World
    </div>
  );
}

export default App;
```

~ 测试通过



E2E测试登录流程

- 新建`login.js`

```
import React, { Component } from "react";

class login extends Component {
  constructor(props) {
    super(props);
    this.state = {
      userName: "",
      password: "",
      message: "未登录"
    };
  }

  login = () => {
```

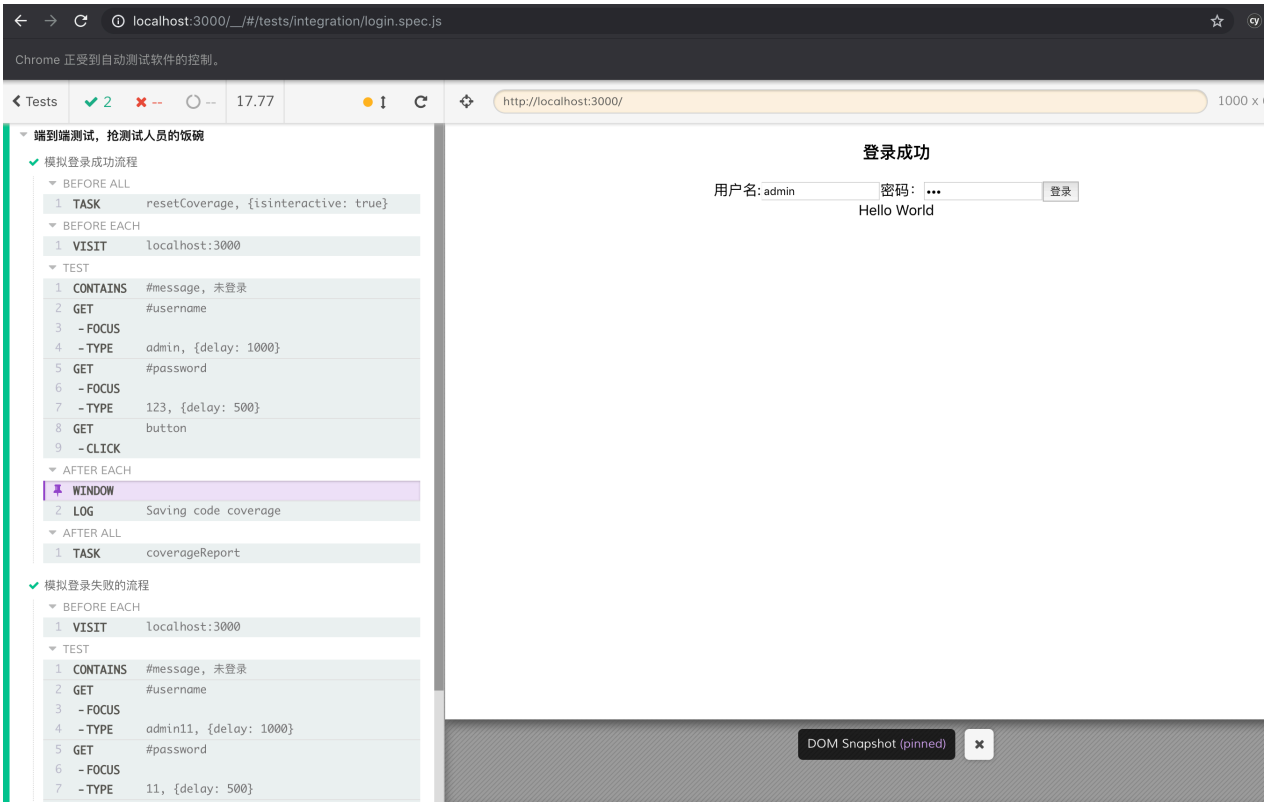
```
    const { userName, password } = this.state;
    if (userName === "admin" && password === "123") {
      this.setState({ message: "登录成功" });
    } else {
      this.setState({ message: "登录失败" });
    }
  };

  render() {
    const { message } = this.state;
    return (
      <div>
        <h3 id="message">{message}</h3>用户名:
        <input
          id="username"
          type="text"
          onChange={e => this.setState({ userName: e.target.value })}
        />
        密码:
        <input
          type="password"
          id="password"
          onChange={e => this.setState({ password: e.target.value })}
        />
        <button onClick={this.login}>登录</button>
      </div>
    );
  }
}

export default login;
```

- 新建`login.spec.js`

```
describe("端到端测试，抢测试人员的饭碗", () => {
  beforeEach(() => {
    cy.visit("localhost:3000");
  });
  it("模拟登录成功流程", () => {
    cy.contains("#message", "未登录");

    const text = "admin";
    cy.get("#username")
      .focus()
      .type(text, { delay: 1000 });

    cy.get("#password")
      .focus()
```

```
      .type(123, { delay: 500 });

    cy.get("button").click();
  });

  it("模拟登录失败的流程", () => {
    cy.contains("#message", "未登录");

    const text = "admin11";
    cy.get("#username")
      .focus()
      .type(text, { delay: 1000 });

    cy.get("#password")
      .focus()
      .type(11, { delay: 500 });

    cy.get("button").click();
  });

  // it("测试用例失败的方法", () => {
  //   cy.visit("/");
  //   cy.contains("h1", "用户名");
  // });
});
```

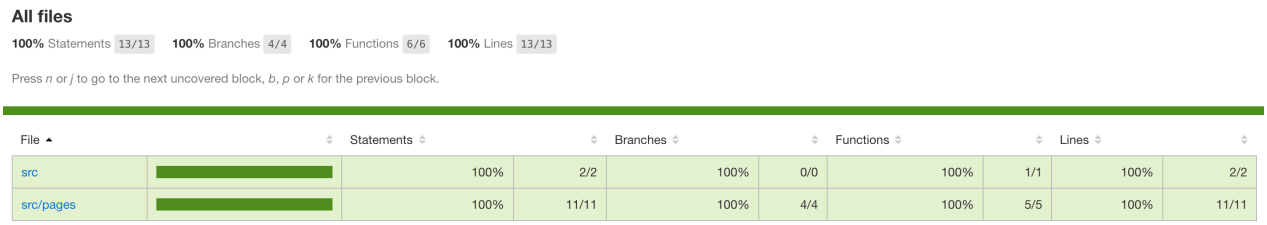
- 测试结果



测试覆盖率

测试完成之后，会自动生成测试报告。执行以下命令在浏览器中打开测试报告

```
$ open coverage/lcov-report/index.html
```



All files

100% Statements `13/13` **100%** Branches `4/4` **100%** Functions `6/6` **100%** Lines `13/13`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
src		100%	2/2	100%	0/0	100%	1/1	100%	2/2
src/pages		100%	11/11	100%	4/4	100%	5/5	100%	11/11