Версия: 29.7

Подготовка и очистка

Часто при написании тестов вам нужно проделать некоторую работу до того, как запустится тест, и некоторую работу по его завершению. Jest предоставляет вспомогательные функции для этих целей.

Повторяющаяся настройка<a>

Если у вас есть какая-то работа, которую вам нужно повторно выполнять для множества тестов, то вы можете использовать хуки `beforeEach` и `afterEach`.

К примеру, допустим, что несколько тестов взаимодействуют с базой городов. У вас есть метод `initializeCityDatabase()`, который должен быть вызван перед каждым тестом, а также метод `clearCityDatabase()`, который должен быть вызван после каждого из них. Это можно сделать следующим образом:

```
beforeEach(() => {
  initializeCityDatabase();
});

afterEach(() => {
  clearCityDatabase();
});

test('city database has Vienna', () => {
  expect(isCity('Vienna')).toBeTruthy();
});

test('city database has San Juan', () => {
  expect(isCity('San Juan')).toBeTruthy();
});
```

`beforeEach` и `afterEach` могут работать с асинхронным кодом также, как это делают асинхронные тесты - они могут либо принимать функцию `done` в качестве параметра, либо возвращать promise. К примеру, если `initializeCityDatabase()` возвращает promise, который вызывает resolve, когда база данных инициализирована, нам бы хотелось вернуть этот promise:

```
beforeEach(() => {
  return initializeCityDatabase();
});
```

Единовременная настройка

В некоторых случаях, подготовительные работы нужны единожды в начале файла. Особенно это касается случаев, когда подготовительный код исполняется асинхронно, и вы не можете просто заинлайнить его. Jest предоставляет хуки `beforeAll` и `afterAll` для таких случаев.

Например, если бы обе функции `initializeCityDatabase()` и `clearCityDatabase()` возвращали promises, а база данных могла быть повторно использована между тестами, мы могли бы изменить наш тестовый код:

```
beforeAll(() => {
  return initializeCityDatabase();
});

afterAll(() => {
  return clearCityDatabase();
});

test('city database has Vienna', () => {
  expect(isCity('Vienna')).toBeTruthy();
});

test('city database has San Juan', () => {
  expect(isCity('San Juan')).toBeTruthy();
});
```

Определение контекста

Перехваты верхнего уровня `before*` и `after*` применяются к каждому тесту в файле. Перехваты, объявленные внутри `describe` блока, применяются только к тестам внутри этого `describe` блока.

К примеру, допустим у нас есть не только база городов, но и база продовольствия. Мы могли бы организовать различную подготовку к разным тестам:

```
// Применяется ко всем тестам в этом файле
beforeEach(() => {
  return initializeCityDatabase();
});

test('city database has Vienna', () => {
  expect(isCity('Vienna')).toBeTruthy();
});

test('city database has San Juan', () => {
  expect(isCity('San Juan')).toBeTruthy();
});
```

```
describe('matching cities to foods', () => {
  // Применяется только к тестам в этом describe блоке
  beforeEach(() => {
    return initializeFoodDatabase();
  });

  test('Vienna <3 veal', () => {
    expect(isValidCityFoodPair('Vienna', 'Wiener Schnitzel')).toBe(true);
  });

  test('San Juan <3 plantains', () => {
    expect(isValidCityFoodPair('San Juan', 'Mofongo')).toBe(true);
  });
});
```

Обратите внимание, что `beforeEach`, находящийся уровнем выше, выполнится до `beforeEach`, находящегося внутри `describe` блока. Пример ниже иллюстрирует последовательность выполнения всех блоков (хуков).

```
beforeAll(() => console.log('1 - beforeAll'));
afterAll(() => console.log('1 - afterAll'));
beforeEach(() => console.log('1 - beforeEach'));
afterEach(() => console.log('1 - afterEach'));
test('', () => console.log('1 - test'));
describe('Scoped / Nested block', () => {
  beforeAll(() => console.log('2 - beforeAll'));
  afterAll(() => console.log('2 - afterAll'));
  beforeEach(() => console.log('2 - beforeEach'));
  afterEach(() => console.log('2 - afterEach'));
  test('', () => console.log('2 - test'));
});

// 1 - beforeAll
// 1 - beforeEach
// 1 - test
// 1 - afterEach
// 2 - beforeAll
// 1 - beforeEach
// 2 - beforeEach
// 2 - test
// 2 - afterEach
// 1 - afterEach
// 2 - afterAll
// 1 - afterAll
```

Порядок выполнения

Jest выполняет все обработчики `describe` внутри одного файла *до того*, как будет запущен какой-либо тест. Это еще одна причина, чтобы проводить подготовительные и завершающие работы внутри обработчиков `before*` и `after*`, вместо того, чтобы описывать их внутри блоков `describe`. Как только завершатся все `describe` блоки, по умолчанию Jest запустит все тесты последовательно в том порядке, в котором они были обнаружены на этапе сбора, ожидая, пока каждый из них завершится и будет убран, прежде чем двигаться дальше.

Рассмотрим следующий пример тестового файла и результат его выполнения:

```
describe('describe outer', () => {
  console.log('describe outer-a');

  describe('describe inner 1', () => {
    console.log('describe inner 1');

    test('test 1', () => console.log('test 1'));
  });

  console.log('describe outer-b');

  test('test 2', () => console.log('test 2'));

  describe('describe inner 2', () => {
    console.log('describe inner 2');

    test('test 3', () => console.log('test 3'));
  });

  console.log('describe outer-c');
});

// describe outer-a
// describe inner 1
// describe outer-b
// describe inner 2
// describe outer-c
// test 1
// test 2
// test 3
```

Jest вызывает хуки `before*` и `after*` в порядке их объявления, точно так же, как блоки `describe` и `test`. Обратите внимание, что сначала вызываются хуки `after*` области видимости. Например, вот как вы можете настроить и отключить ресурсы, которые зависят друг от друга:

```
beforeEach(() => console.log('connection setup'));
beforeEach(() => console.log('database setup'));

afterEach(() => console.log('database teardown'));
```

```
afterEach(() => console.log('connection teardown'));

test('test 1', () => console.log('test 1'));

describe('extra', () => {
  beforeEach(() => console.log('extra database setup'));
  afterEach(() => console.log('extra database teardown'));

  test('test 2', () => console.log('test 2'));
});

// connection setup
// database setup
// test 1
// database teardown
// connection teardown

// connection setup
// database setup
// extra database setup
// test 2
// extra database teardown
// database teardown
// connection teardown
```

ПРИМЕЧАНИЕ

Если вы используете `jasmine2` тестовый раннер, примите во внимание, что он вызывает `after*` перехваты в порядке, обратном порядку объявления. Чтобы получить идентичный результат, приведенный выше пример следует изменить следующим образом:

```
  beforeEach(() => console.log('connection setup'));
+ afterEach(() => console.log('connection teardown'));

  beforeEach(() => console.log('database setup'));
+ afterEach(() => console.log('database teardown'));

- afterEach(() => console.log('database teardown'));
- afterEach(() => console.log('connection teardown'));

  // ...
```

Общие рекомендации

Если тест падает, в первую очередь нужно проверить, что он падает, будучи запущенным в оди
В Jest это легко сделать: временно поменяйте команду `test` на `test.only`:

```
test.only('this will be the only test that runs', () => {
  expect(true).toBe(false);
});

test('this test will not run', () => {
  expect('A').toBe('A');
});
```

Если у вас есть тест, который часто падает при выполнении внутри набора тестов, но не падает будучи запущенным в одиночку, значит, что-то из другого теста мешает текущему. Часто это легко исправить, очищая общее состояние внутри функции `beforeEach`. Если нет уверенности, нужно ли очищать общее для тестов состояние, можно воспользоваться `beforeEach` для записи логов выполнения.