TEST

Solidity智能合约开发

第11.2课：Hardhat 3 单元测试基础与技巧

第三阶段　开发框架与实战　测试驱动开发

区块链导师【Layer】

为什么需要单元测试? Why Unit Testing?

没有测试的问题Problems

- **手动测试繁琐**
  Manual testing is tedious
- **容易遗漏边界情况**
  Easy to miss edge cases
- **重构时缺乏信心**
  Lack confidence when refactoring
- **生产环境容易出错**
  Production errors
- **团队协作困难**
  Team collaboration issues
- **无法保证代码质量**
  No quality assurance

单元测试的优势Advantages

- **自动化验证功能**
  Automated validation
- **快速反馈问题**
  Quick feedback
- **文档化合约行为**
  Documentation of behavior
- **提升代码质量**
  Improved code quality
- **支持持续集成**
  CI/CD support
- **重构更安全**
  Safer refactoring

Hardhat 3 测试框架概览 Testing Framework Overview

Mocha
测试运行器

- **测试运行器**
  Test runner
- **异步测试支持**
  Async test support
- **钩子函数**
  Hook functions
- **灵活配置**
  Flexible configuration
- **丰富报告**
  Rich reporting
- **社区成熟**
  Mature community

Chai
断言库

- **断言库**
  Assertion library
- **expect/should/assert**
  Multiple styles
- **链式语法**
  Chainable syntax
- **可读性强**
  Readable assertions
- **BDD/TDD风格**
  BDD/TDD style
- **插件丰富**
  Rich plugins

Hardhat 插件
区块链特定功能

- **hardhat-chai-matchers**
  区块链断言
- **.to.emit()**
  事件断言
- **.to.changeEtherBalance()**
  余额断言
- **.to.be.revertedWith()**
  回退断言
- **hardhat-network-helpers**
  网络助手
- **快照恢复**
  Snapshot recovery

Hardhat 3 测试框架 = Mocha + Chai + 区块链特定功能

测试文件结构Test File Structure

文件组织File Organization

- **test/ 目录存放测试文件**
  Directory for test files
- **命名：ContractName.test.ts**
  Naming: ContractName.test.ts

结构层次Structure Hierarchy

- **describe() – 测试套件**
  Test suite
- **describe() – 子套件**
  Sub suite
- **it() – 测试用例**
  Test case

最佳实践Best Practices

- **按功能分组测试**
  Group tests by functionality
- **使用描述性名称**
  Use descriptive names

测试文件示例Example

```
//导入语句
import { expect } from "chai";
import { network } from "hardhat";
import { loadFixture } from "@nomicfoundation/hardhat-network-helpers";

//连接网络
const { ethers } = await network.connect();

//测试套件
describe("Counter", function () {
 //子套件
 describe("Deployment", function () {
  //测试用例
  it("Should deploy with initial value 0", async function () {
   const counter = await ethers.deployContract("Counter");
   await expect(counter.x()).to.equal(0);
  });
 });

 //子套件
 describe("Increment", function () {
  //测试用例
  it("Should increment the counter", async function () {
   const counter = await ethers.deployContract("Counter");
   await counter.inc();
   await expect(counter.x()).to.equal(1);
  });
 });
});
```

Mocha 测试结构详解Mocha Test Structure

Mocha 钩子函数Hook Functions

describe
测试套件容器

it
测试用例

beforeEach/afterEach
每个测试前/后执行

before/after
所有测试前/后执行一次

```
describe("Counter", function() {
  before(() => { /* 一次 */ });
  beforeEach(() => { /* 每个 */ });
  it("Test 1", () => { /* 测试 */ });
  it("Test 2", () => { /* 测试 */ });
  afterEach(() => { /* 每个 */ });
  after(() => { /* 一次 */ });
});
```

执行流程Execution Flow

before()
所有测试前执行一次

beforeEach()
每个测试前执行

it("Test 1")
执行第一个测试用例

afterEach()
每个测试后执行

after()
所有测试后执行一次

Chai 断言语法基础 Chai Assertion Syntax

基础断言 Basic Assertions

相等性断言
Equality assertions

```JS
expect(value).to.equal(42);
expect(value).to.not.equal(0);
```

数值比较
Number comparisons

```JS
expect(balance).to.be.above(100);
expect(balance).to.be.below(1000);
expect(balance).to.be.at.least(100);
```

链式断言 Chained Syntax

链式语法示例
Chained syntax examples

```JS
expect(balance)
  to.be.a("bigint")
  and.to.be.above(0)
  and.to.be.below(1000000);
```

组合断言
Combined assertions

```JS
expect(accounts)
  to.be.an("array")
  with.length(3)
  .that.includes(owner.address);
```

链式语法提高可读性，组合多个断言为一行

Hardhat 3 特定断言 Hardhat-Specific Assertions

事件断言

验证事件是否触发及参数是否正确

```
await expect(counter.inc()).
to.emit(counter, "Increment").
withArgs(1n);
```

验证事件触发与参数

余额变化断言

验证以太坊或代币余额变化

```
await expect(
owner.sendTransaction({
to: addr1.address,
value: amount
})
).to.changeEtherBalances(
[owner, addr1],
[-amount, amount]
);
```

验证转账金额与手续费

回退断言

验证交易是否正确回退

```
await expect(counter.incBy(0)).
to.be.revertedWith(
"incBy: increment should be positive"
);
```

验证错误消息与回退

使用场景与最佳实践

- 事件断言：验证状态变化通知，确保事件参数正确
- 余额断言：验证转账金额，确保资金流向正确
- 回退断言：验证错误处理，确保安全机制生效
- 注意：消息必须完全匹配，事件名称区分大小写

合约部署和函数测试

部署测试策略Deployment Testing

- **验证合约地址有效**
  Verify contract address is valid
- **检查初始状态**
  Check initial state
- **确认合约字节码**
  Verify contract bytecode

```
import { expect } from "chai";
import { network } from "hardhat";
import { loadFixture } from "@nomicfoundation/hardhat-network-helpers";
const { ethers } = await network.connect();

describe("Count er Deployment", function() {
  it("Should deploy with initial value 0", async function() {
    const counter = await ethers.deployContract("Counter");
    const address = await counter.getAddress();

    expect(address).to.be.a("string");
    expect(address).to.have.length(42);
    expect(await counter.x()).to.equal(0);

    // 验证字节码
    const bytecode = await hre.artifacts.readArtifact("Counter");
    expect(bytecode.bytecode.length).to.be.greaterThan(0);
  });
});
```

边界测试策略 Boundary Testing Strategies

零值测试

测试输入值为零的情况

```
// 转账零金额
await expect(token.transfer(recipient, 0))
  .to.emit(token, "Transfer")
  .withArgs(owner, recipient, 0);
```

最大值测试

测试最大允许值的情况

```
// 测试最大整数值
const maxUint = ethers.MaxUint256;
await expect(token.transfer(recipient, maxUint))
  .to.emit(token, "Transfer")
  .withArgs(owner, recipient, maxUint);
```

溢出测试

测试算术溢出临界情况

```
// 测试溢出控制
it("should prevent overflow", async function() {
  const maxUint = ethers.MaxUint256;
  await expect(token.safeAdd(maxUint, 1))
    .to.be.revertedWith("Overflow");
});
```

事件触发测试 Event Emission Testing

事件测试示例 Examples

```
// 单个事件测试
it( "Should emit Increment event" , async function () {
  const counter =  await ethers.deployContract( "Counter" );
  await expect(counter.inc())
          .to.emit(counter,"Increment" )
                1n );                    .withArgs(
                  });
```

```
// 多个事件测试（同一测试中）
it( "Should emit multiple events" , async function () {
  const token =  await ethers.deployContract( "Token" );

  // 在一个交易中发出多个事件
  await expect(token.transfer(owner, recipient, amount))
          .to.emit(token,"Transfer"  )
                                    .withArgs(owner, recipient, amount)
                                    .and
                        "Approval"  ) .to.emit(token,
                                    .withArgs(owner, spender, amount);
                                  });
```

```
// 判断事件是否未触发
it( "Should not emit event" , async function () {
  const token =  await ethers.deployContract( "Token" );

  await expect(token.transfer(owner, owner, amount))
          .to.not.emit(token, "Transfer"  );  // 转账给自己不会发出Transfer事件
                                                          });
```

索引参数与filters Indexing & Filters

索引参数

indexed参数会进行哈希索引，便于后续查询

```
event Transfer(address indexed from, address indexed to, uint256 amount);
```

filters使用方法

查询已索引的参数值可提高效率

```
// 基于索引参数的查询
const filter = token.filters.Transfer(null, recipient);  // 任何接收者为指定账户
const events = await token.queryFilter(filter, blockFrom, blockTo);
```

最佳实践

- 为常用查询参数添加indexed标记
- 索引每个参数约花费28900gas
- 避免同时索引多个不相关参数

错误和回退测试Error & Revert Testing

验证智能合约中的各种错误处理机制，确保安全性和预期行为

require 回退

测试带错误消息的回退

```
await expect(counter.incBy(0))
  .to.be.revertedWith("incBy: increment should be positive");
```

自定义错误

Solidity 0.8.4+ 的自定义错误

```
await expect(contract.withdraw(1000))
  .to.be.revertedWithCustomError(contract, "InsufficientBalance");
```

Panic 错误

算术溢出、数组越界等

```
await expect(contract.divide(10, 0))
  .to.be.revertedWithPanic(0x12); // Division by zero
```

无原因回退

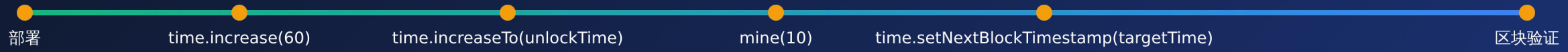
revert() 无参数或 require(false)

```
await expect(contract.fail())
  .to.be.reverted;
```

测试技巧

- 错误消息必须精确匹配
- 测试正常和异常路径
- 验证安全机制
- 使用描述性测试名称
- 使用 await 等待断言
- 确保消息完全匹配

时间旅行和区块操作 Time Travel & Block Operations



时间操作 Time Operations

time.increase(seconds)

增加指定秒数

```
// 快进 60 秒
await time.increase(60n);
```

time.increaseTo(timestamp)

跳转到特定时间戳

```
// 直接跳到解锁时间
await time.increaseTo(unlockTime);
```

time.latest()

获取当前时间戳

```
// 获取当前时间
const now = await time.latest();
```

区块操作 Block Operations

mine(blocks)

挖指定数量区块

```
// 挖 10 个区块
await mine(10);
```

time.setNextBlockTimestamp(timestamp)

设置下一个区块的时间戳

```
// 设置下一区块时间
await time.setNextBlockTimestamp(targetTime);
```

测试覆盖率Test Coverage

覆盖率配置Configuration

```ts
// hardhat.config.ts
import { HardhatUserConfig } from "hardhat/config";
import "@nomicfoundation/hardhat-toolbox";
import "solidity-coverage";

const config: HardhatUserConfig = {
  solidity: "0.8.24",
  networks: {
    hardhat: {
      type: "edr-simulated",
      chainId: 31337,
    },
  },
  //覆盖率配置
  coverage: {
    enabled: true,
    exclude: [
      "test/",
      "node_modules/",
      "coverage/",
    ],
  },
};
```

运行命令Commands

```
npx hardhat coverage
; 运行覆盖率测试

npx hardhat coverage --report html
生成 HTML 报告

npx hardhat coverage --threshold 80
查看覆盖率阈值
```

报告解读Report

File	Stmts	Branch	Funcs	Lines
Counter.sol	100%	100%	100%	100%
Token.sol	95.2%	87.5%	90%	95%
Lock.sol	80.0%	75.0%	85%	80.0%
All files	**91.7%**	**87.5%**	**91.7%**	**91.7%**

覆盖率指标与目标Metrics & Targets

覆盖率指标

- **Statements:** 语句覆盖率
- **Branch:** 分支覆盖率

覆盖率目标

- **关键合约:** 100%
- **一般合约:** >80%

Gas 报告 Gas Reporting

Gas 报告配置 Configuration

```
// hardhat.config.ts
gasReporter: {
  enabled: true,
  currency: "USD",
  gasPrice: 50, // gwei
  coinmarketcap: process.env.COINMARKETCAP_API_KEY
}
```

环境变量设置 Environment Variables

```
# .env 文件
REPORT_GAS = true
COINMARKETCAP_API_KEY = your_api_key
```

运行命令 Commands

```
# 生成 Gas 报告
REPORT_GAS = true npx hardhat test

# 生成详细报告
REPORT_GAS = true npx hardhat test --verbose
```

使用 --verbose 查看详细 Gas 报告

Gas 报告示例 Report Example

Contract	Method	Min	Max	Avg
Counter	inc()	21,063	21,063	21,063
Token	transfer	52,000	65,000	58,500

报告解读 Report Interpretation

- Min: 最小 Gas 消耗
- Max: 最大 Gas 消耗
- Avg: 平均 Gas 消耗

优化建议 Optimization Tips

- 对比不同实现方式
- 识别高 Gas 函数
- 优化存储操作和减少外部调用

快照和恢复 Snapshots & Fixtures

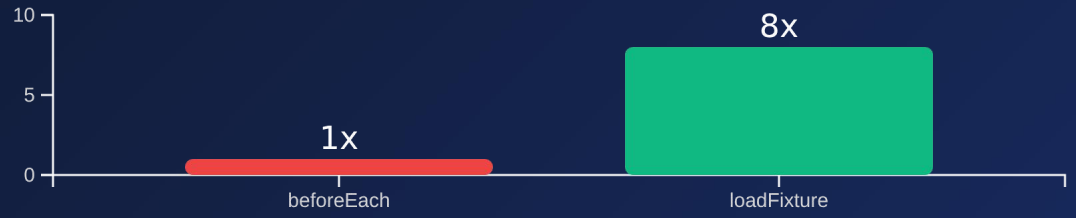
使用 **loadFixture** 可以创建测试快照并恢复状态，提升测试性能。

使用场景

测试隔离
每个测试独立运行，不受其他测试影响

性能优化
避免重复部署，大幅提升测试速度

性能对比



快照示例

```
//复杂 Fixture 示例
async function deployTokenFixture() {
  const [owner, addr1, addr2] = await ethers.getSigners();

  const token = await ethers.deployContract("Token", [
    "MyToken",
    "MTK",
    ethers.parseEther("1000000")
  ]);

  //执行一些初始操作
  await token.transfer(addr1.address, ethers.parseEther("1000"));
  await token.transfer(addr2.address, ethers.parseEther("2000"));

  return { token, owner, addr1, addr2 };
}

//测试套件
describe("Token with Fixture", function () {
  //测试用例：余额测试
  it("Should have initial balances", async function () {
    const { token, addr1, addr2 } = await loadFixture(deployTokenFixture);

    await expect(token.balanceOf(addr1.address))
      .to.equal(ethers.parseEther("1000"));
    await expect(token.balanceOf(addr2.address))
      .to.equal(ethers.parseEther("2000"));
  });
});
```

最佳实践

- 使用 loadFixture 替代 beforeEach

测试最佳实践 Testing Best Practices

1. 描述性命名
使用清晰的测试名称，描述测试场景和预期结果

2. 测试组织
按功能分组测试，使用嵌套 describe，保持结构清晰

3. 测试独立性
每个测试独立运行，使用 loadFixture 隔离状态

4. 全面覆盖
测试正常流程、异常情况和边界条件

5. 错误处理
验证错误消息，测试所有回退路径，使用自定义错误

6. 性能优化
使用快照恢复，避免重复部署，并行运行测试

7. 代码复用
提取公共逻辑到 Fixture，使用辅助函数

8. 文档化
添加注释说明复杂逻辑，使用描述性变量名

9. CI/CD 集成
自动化测试运行，覆盖率检查，Gas 报告生成

10. 持续改进
定期审查测试，重构冗余测试，优化测试性能

提示：遵循这些最佳实践可以显著提高测试质量和开发效率

常见错误和解决方案Common Errors & Solutions

常见错误Common Errors

忘记 await

Error: Expected transaction to be reverted

```
await expect(contract.fail()).to.be.reverted;
```

事件名称错误

Error: Event "Increment" not found

检查事件名称大小写

参数类型不匹配

Error: Expected "bigint", got "number"

使用 1n 而不是 1

错误消息不匹配

Error: Expected revert reason mismatch

确保消息完全匹配

解决方案Solutions

使用 await

```
// 使用 await 等待异步断言
await expect(contract.fail())
.to.be.reverted;
```

检查事件定义

```
// 检查事件名称大小写
event Increment(uint256 by);
使用 counter.filters.Increment()
```

使用 bigint

```
// 使用 1n 而不是 1
expect(value).to.equal(1n);
或使用 ethers.parseEther()
```

精确匹配消息

```
// 确保消息完全匹配
.to.be.revertedWith("exact message");
或使用 .to.be.reverted
```

运行测试命令 Test Commands

基本命令

npx hardhat test
运行所有测试

npx hardhat test test/Token.test.ts
运行特定测试文件

npx hardhat test --grep "Deployment"
运行匹配的测试

覆盖率命令

npx hardhat coverage
生成覆盖率报告

npx hardhat coverage --report html
HTML 报告

npx hardhat coverage --threshold 80
设置覆盖率阈值

Gas 报告命令

REPORT_GAS=true npx hardhat test
生成 Gas 报告

REPORT_GAS=true npx hardhat test > gas-report.txt
保存到文件

测试调试技巧 Test Debugging Tips

有效的调试技巧可以快速定位和修复测试中的问题

使用 console.log

打印变量值和中间状态，帮助定位问题

```
console.log("Value:", await counter.x()); console.log("Address:", await counter.getAddress());
```

使用 debugger

设置断点，逐行执行代码，检查状态变化

```
it("Should debug", async function() { const counter = await ethers.deployContract("Counter"); debugger; // 断点 await counter.inc(); });
```

运行: node --inspect-brk node_modules/.bin/hardhat test

测试单个用例

使用 it.only 只运行特定测试，聚焦问题

```
it.only("Should test this", async function() { // 只运行这个测试 });
```

查看交易详情

检查 gas 消耗和事件，分析异常原因

```
const tx = await counter.inc(); const receipt = await tx.wait(); console.log("Gas used:", receipt.gasUsed); console.log("Events:", receipt.logs);
```

Hardhat Console

交互式调试环境

打印完整对象

使用 JSON.stringify

事件日志

使用 queryFilter 检查事件

课程总结 Course Summary

核心知识点 Core Knowledge

测试框架
Mocha: 测试运行器, Chai: 断言库
Hardhat 插件: 区块链特定功能

测试结构
describe/it 组织测试, beforeEach/afterEach 钩子
loadFixture 快照恢复

断言方法
基础断言: equal, above, true
Hardhat 断言: emit, changeBalance

高级特性
事件测试: .to.emit(), 时间操作: time.increase()
区块操作: mine(), 快照恢复: loadFixture()

进阶学习 Next Steps

集成测试
测试多个合约之间的交互

端到端测试
测试完整用户流程和场景

测试策略设计
制定全面的测试计划和策略

CI/CD 集成
将测试集成到持续集成流程

持续改进测试实践，不断提升代码质量 — Continuous improvement