智能合约部署

部署方面，学习汇报主要是以代码的形式，展示相关合约代码（现阶段对代码理解比较片面，下个月重点学习这一方向）。加深下对这些代码的印象，至于深刻去理解每一句的话还是十一月下点功夫。

**1.解压代码：**

# tar -xf /home/centos/YYXcoin.tar.gz

# ls

flowrevenue nodepledge teamlock

**2.修改配置文件：**

# vim teamlock/truffle-config.js

**编译流量挖矿只能合约代码：**

# cd flowrevenue/

# truffle compile

Compiling your contracts...

===========================

> Compiling ./contracts/Migrations.sol

> Compiling ./contracts/TrafficProfitLockUp.sol

> Compiling openzeppelin-solidity/contracts/math/SafeMath.sol

> Artifacts written to /data/workspace/YYXACoin/flowrevenue/build/contracts

> Compiled successfully using:

- solc: 0.5.1+commit.c8a2cb62.Emscripten.clang

**编译节点质押智能合约代码：**

# cd ../nodepledge/

# truffle compile

Compiling your contracts...

===========================

> Compiling ./contracts/Migrations.sol

> Compiling ./contracts/NodePledgeLockUp.sol

> Compiling openzeppelin-solidity/contracts/math/SafeMath.sol

> Artifacts written to /data/workspace/YYXACoin/nodepledge/build/contracts

> Compiled successfully using:

- solc: 0.5.1+commit.c8a2cb62.Emscripten.clang

**编译团队锁仓智能合约代码：**

# cd ../teamlock/

# truffle compile

Compiling your contracts...

===========================

> Compiling ./contracts/Migrations.sol

> Compiling ./contracts/TeamFundLockUp.sol

> Compiling openzeppelin-solidity/contracts/math/SafeMath.sol

> Artifacts written to /data/workspace/YYXACoin/teamlock/build/contracts

> Compiled successfully using:

- solc: 0.5.1+commit.c8a2cb62.Emscripten.clang

**编译代币智能合约代码：**

# cd /data/workspace/SDVN/

# truffle compile

C：智能合约测试

在查找和测试相关的文档的时候，主要是ethers.js + Waffle和Web3.js + Truffle这两种方式，现在还只了解了ethers.js，剩下那个下个月继续补充。这里引入到一个新的工具：Hardhat。在开发，部署和测试环境的时候，虽然两者都可以达到这样的目的，但感觉Hardhat更高级一些（不过只要能精通一个那就是很牛逼的了）。

这是百度的一些测试内容：

**状态检查。**例如合约部署后检查初始状态是否正确，函数调用后检查状态是否改变。一般状态检查为读取view函数。

**事件触发**。基本上，合约中的关键操作都应该触发事件进行相应追踪。在单元测试中了可以测试事件是否触发，抛出的参数是否正确。

**交易重置**。在测试一些非预期条件时，交易应当重置并给出相应的原因。使用单元测试可以检测是否重置及错误原因是否相同。

**函数计算**。例如要计算不同条件下某函数的返回值（例如奖励值），我们需要循环调用 某个函数并输入不同的参数，看是否结果相符。

**完全功能测试**。例如我们合约中涉及到了区块高度或者 区块时间，比如质押一年后才能提取。此时我们一般需要加速区块时间或者区块高度来进行测试。幸运的是，hardhat提供了接口可以方便的进行此项测试。

**测试覆盖率**。包含代码覆盖率，函数覆盖率和分支覆盖率。一般情况下，应该追求 100%完全覆盖。比如你写了一个modifier，但是忘记加到函数上去了，而单元测试也漏掉了，此时代码覆盖就会显示该代码未测试，这样可以发现一些简单的BUG。特殊情况下或者确定有代码不会执行的情况下，不追求100%覆盖率。

在学习测试代码的时候参考了网上的案例

const { expect, util, assert } = require("chai");

const { ethers } = require("hardhat");

describe("Greeter", function () {

let greeter;

let owner, user1, users;

beforeEach(async () => {

[owner, user1, ...users] = await ethers.getSigners();

const Greeter = await ethers.getContractFactory("Greeter");

greeter = await Greeter.deploy("Hello, world!");

await greeter.deployed();

});

describe("State check test", function () {

it("Should return the new greeting once it's changed", async function () {

expect(await greeter.greet()).to.equal("Hello, world!");

const setGreetingTx = await greeter.setGreeting("Hola, mundo!");

// wait until the transaction is mined

await setGreetingTx.wait();

expect(await greeter.greet()).to.equal("Hola, mundo!");

});

});

describe("Console test", function () {

it("Console.log should be successful", async function () {

expect(await greeter.testConsole()).to.be.equal(true);

});

});

describe("Event test", function () {

it("owner emit test", async () => {

await expect(greeter.eventTest())

.to.be.emit(greeter, "CallerEmit")

.withArgs(owner.address, 500);

});

it("user1 emit test", async () => {

await expect(greeter.connect(user1).eventTest())

.to.be.emit(greeter, "CallerEmit")

.withArgs(user1.address, 500);

});

it("Get emit params test", async () => {

const tx = await greeter.connect(users[0]).eventTest();

await tx.wait();

const receipt = await ethers.provider.getTransactionReceipt(tx.hash);

const hash = ethers.utils.solidityKeccak256(

["string"],

["CallerEmit(address,uint256)"]

);

const infos = receipt.logs[0];

assert.equal(infos.topics[0], hash);

const sender = ethers.utils.getAddress(

"0x" + infos.topics[1].substring(26)

);

assert.equal(sender, users[0].address);

const value = ethers.BigNumber.from(infos.data);

expect(value).to.be.equal(500);

});

});

});

个人总结这个代码是分为三个目的的

**1.正常测试，检查事件是否触发，参数是否正确。**

**2.切换合约调用者为user1。**

**3.这里是解析事件来获取事件参数，此场景应用于某些事件参数无法提前获取等，比如一个伪随机数。**

接下来还学习了重置测试和区块测试

A:重置测试

测试条件不满足时的交易重置需要在合约中添加如下代码：

function revertTest(uint a, uint b) public {

require(a > 10, "a <= 10");

if(b > 10) {

revert("b > 10 ");

}else {

revert();

}

}

**当报错结果显示最后一个revert缺少提示字符串时在测试文件中添加如下describe：**

**describe("Revert test", function () {**

**it("a < 10 should be failed", async () => {**

**await expect(greeter.revertTest(5, 5)).to.be.revertedWith("a <= 10");**

**});**

**it("b > 10 should be failed", async () => {**

**await expect(greeter.revertTest(15, 55)).to.be.revertedWith("b > 10");**

**});**

**it("b < 10 should be failed", async () => {**

**await expect(greeter.revertTest(15, 5)).to.be.reverted;**

**});**

**});**

B:区块测试

当合约中的内容涉及到区块时，一般需要进行相应区块高度或者区块时间的条件测试。先在测试合约中添加如下内容：

function blockNumberTest() public {

require(block.number >= 10000,"not matched block number");

console.log("block number: %d", block.number);

}

function blockTimeTest() public {

require(block.timestamp >= 1750631915,"not matched block time");

console.log("block timestamp: %d", block.timestamp);

}

**编译时会提示上面两个函数为view函数，但是如果我们把它标记为view函数，那么测试时便不会mine一个新区块。为了模拟真实场景，我们不把它标记为view函数，从而在调用时产生一个新的区块。**

**然后在测试文件中增加如下describe：**

describe("Block test", () => {

let block;

let timestamp;

// 用来去除16进制的左边自动补零

function convertNum(num) {

let big = ethers.BigNumber.from("" + num)

let str = big.toHexString()

let index = 0

for(let i=2;i<str.length;i++) {

if(str[i] !== "0") {

index = i;

break;

}

}

if(index === 0) {

return str;

}else {

return str.substring(0,2) + str.substring(index)

}

}

beforeEach(async () => {

block = await ethers.provider.getBlockNumber();

timestamp = (await ethers.provider.getBlock()).timestamp;

});

// 注意，这里hardhat network 默认是一秒一个区块

it("Call before timestamp 1651631915 should be failed", async () => {

assert.ok(timestamp < 1651631915);

await expect(greeter.blockTimeTest()).to.be.revertedWith(

"not matched block time"

);

});

it("Call at timestamp 1651631915 should be successfult", async () => {

await ethers.provider.send("evm\_mine", [1651631915 - 1]);

await greeter.blockTimeTest();

});

it("Call before block 10000 should be failed", async () => {

assert.ok(block < 10000);

await expect(greeter.blockNumberTest()).to.be.revertedWith(

"not matched block number"

);

});

it("Call at block 10000 should be successful", async () => {

let value = 10000 - block - 1;

//快速推进到100000区块前一个

await ethers.provider.send("hardhat\_mine", [convertNum(value]);

await greeter.blockNumberTest();

});

});