

Truffle Framework

hexlant.

조성은

Truffle Framework

- 이더리움 스마트 컨트랙트를 보다 쉽게 개발, 테스트, 배포할 수 있는 프레임워크
 - * EVM을 사용하는 블록체인 ex) Quorum, Hyperledger Fabric's Evm
- Solidity, Javascript를 사용하여 테스트 케이스 작성 가능
- Mainnet, Testnet, Privatenet 모두 배포 가능

Github : <https://github.com/trufflesuite/truffle>

WebSite : <https://www.trufflesuite.com/>

Docs : <https://www.trufflesuite.com/docs/truffle/overview>

Truffle Framework 개발 환경 셋팅 1

- Node.js : <https://nodejs.org/en/>

```
$ node -v      // node version 확인
```

```
$ npm -v       // npm version 확인
```

- VSCode : <https://code.visualstudio.com/>

Truffle Framework 개발 환경 셋팅 2

- Truffle : <https://www.npmjs.com/package/truffle>

```
$ npm install -g truffle
```

- Ganache-cli : <https://www.npmjs.com/package/ganache-cli>

```
$ npm install -g ganache-cli
```

Truffle 프로젝트 디렉토리 생성 및 초기화

프로젝트 디렉토리 생성 및 이동

```
$ mkdir project  
$ cd project
```

트러플 초기화 명령어

```
$ truffle init
```

Truffle 프로젝트 디렉토리 구조

```
project $ tree
```

```
.
├── contracts
│   └── Migrations.sol
├── migrations
│   └── 1_initial_migration.js
├── test
└── truffle-config.js
```

contracts : 컨트랙트(*.sol) 파일을 담는 폴더

contracts/Migrations.sol : 컨트랙트 업그레이드를 관리하는
도우미(helper) 컨트랙트

migrations : contracts 폴더에 정의된 컨트랙트를 배포하는 배포
스크립트를 담는 폴더

migrations/1_initial_migration.js : contracts 폴더 안에 있는
Migrations.sol 파일을 배포하는
배포 스크립트

test : 테스트 스크립트 파일을 담는 폴더

truffle-config.js : 트러플 구성 파일
(네트워크 정보, 컴파일 버전, 최적화, 라이브러리
등)

Ganache-cli

- 트러플 이더리움 개발 툴 중 하나
- cli & gui
- Ethereum RPC client for testing and development

Ganache 실행 명령어

```
$ ganache-cli // running 127.0.0.1:8545
```

컨트랙트 작성 - contracts/mytoken.sol

mytoken contract

https://github.com/eun0o/eth_token_contract/blob/master/mytoken.sol

- 컨트랙트 함수 설명

name	: 토큰의 이름 반환
symbol	: 토큰의 심볼 반환
decimals	: 토큰의 데시멀 반환
totalSupply	: 토큰의 총 발행량 반환
balanceOf(owner)	: 'owner'의 토큰 잔액 반환
transfer(to, amount)	: 'to'에게 'amount' 만큼의 토큰 전송

트러플 구성 - truffle-config.js

```
module.exports = {  
  networks: {  
    development: {  
      host: "127.0.0.1",  
      port: 8545,  
      network_id: "*"  
    }  
  }  
  compilers: {  
    solc: {  
      version: "0.5.0"  
      settings: {  
        optimizer: {  
          enabled: false,  
          runs: 200  
        }  
      }  
    }  
  }  
};
```

networks: 네트워크 정보 등록
development: 네트워크 이름 정의
host: 호스트 ip
port: 포트
network_id: 네트워크 ID

compilers: 컴파일러 정보 등록
solc: 컴파일러 관련 정의
version: 컴파일 버전
settings:
optimizer: 최적화
enabled: 최적화 활성화 유무
run: 200

컨트랙트 컴파일

이더리움 컨트랙트는 **Solidity**로 작성된 고급 언어입니다.
컨트랙트가 **EVM(Ethereum Virtual Machine)** 위에서 실행되기 위해서는 **bytecode**로 컴파일이 되어야합니다.

Compile 과정은 **EVM**에서의 실행을 위한 **Solidity** 코드의 변환 과정입니다.

컨트랙트 컴파일 명령어

```
$ truffle compile
```

컨트랙트 테스트 필요성

- 스마트 컨트랙트 자체는 한번 배포되면 수정이 불가능하다.
- 추후 버그가 발견되도 수정이 불가능한 상황이 된다.
- 토큰과 같이 자산을 다루는 컨트랙트의 경우, 개발자의 코드 실수는 막대한 자금의 피해를 초래한다.
- **The DAO 사건** : 탈 중앙화된 방식으로 클라우드 세일을 통해 자금을 모집하는 컨트랙트에서 코드 결함이 발견되고 **750억** 가량의 자산이 해킹된 사건
- **SMT, BEC 토큰의 무한 생성 (Overflow)** : 토큰의 수량이 무한대의 수로 생성되버린 사건

<https://etherscan.io/token/0x55f93985431fc9304077687a35a1ba103dc1e081#balances>

트러플 테스트 코드 작성법 예시

```
const contractName = artifacts.require("mytoken");

contract("My Test" async function([유저1, 유저2, 유저3]) {

  beforeEach(async function () {
    this.contract = await contractName.new();
  })

  describe("시나리오 1", function () {
    it("유저1이 유저2에게 토큰 100개 전송하면 유저2의 토큰 잔액이 100개가 되는가?", async function () {
      await this.contract.transfer(유저2, 100, { from: 유저1 })
      var 토큰잔액_유저2 = await this.contract.balanceOf(유저2)
      assert.equal(await this.contract.balanceOf(유저2), 토큰잔액_유저2, "토큰 잔액 불 일치")
    })

    it("유저1이 유저2에게 마이너스의 토큰을 전송하면 전송이 실패하는가?", async function () {
      await expectRevert.unspecified(this.contract.transfer(유저2, -1, { from: 유저1 }))
    })
  })
})
```

테스트 코드 Hooks

before() : 테스트 전체가 진행되기 이전에 1회 실행

beforeEach() : 개별 테스트가 진행되기 이전에 지속적으로 실행

describe() : 관련 테스트를 그룹핑

it() : 개별 테스트

afterEach() : 개별 테스트가 진행된 이후에 지속적으로 실행

after() : 테스트 전체가 종료된 이후 1회 실행

```
contract("contract 입니다.", async function () {  
  before(async function () {  
    console.log("before 입니다.")  
  })  
  
  beforeEach(async function () {  
    console.log("beforeEach 입니다.")  
  })  
  
  afterEach(async function () {  
    console.log("afterEach 입니다.")  
  })  
  
  after(async function () {  
    console.log("after 입니다.")  
  })  
  
  describe("1번 describe 입니다.", function () {  
    it("", async function () {  
      console.log("1-1 it 입니다.")  
    })  
    it("", async function () {  
      console.log("1-2 it 입니다.")  
    })  
  })  
  
  describe("2번 describe 입니다.", function () {  
    it("", async function () {  
      console.log("2-1 it 입니다.")  
    })  
    it("", async function () {  
      console.log("2-2 it 입니다.")  
    })  
  })  
})
```

```
Contract: contract 입니다 .  
before 입니다 .  
  1번 describe 입니다 .  
beforeEach 입니다 .  
1-1 it 입니다 .  
✓  
afterEach 입니다 .  
beforeEach 입니다 .  
1-2 it 입니다 .  
✓  
afterEach 입니다 .  
  2번 describe 입니다 .  
beforeEach 입니다 .  
2-1 it 입니다 .  
✓  
afterEach 입니다 .  
beforeEach 입니다 .  
2-2 it 입니다 .  
✓  
afterEach 입니다 .  
after 입니다 .
```

4 passing (48ms)

테스트 코드 작성 - test/test.js (함수 호출)

```
const contractName = artifacts.require('mytoken');

contract('mytoken contract test', async function(성은, 철수, 영화, 민수) {

    contract = await contractName.new();

    // 토큰 이름을 확인하는 함수 호출
    var token_name = await contract.name();
    console.log(token_name);

    // 토큰 심볼을 확인하는 함수 호출
    var token_symbol = await contract.symbol();
    console.log(token_symbol);

    // 토큰의 총 발행량을 확인하는 함수 호출
    var token_totalSupply = await contract.totalSupply();
    console.log(token_totalSupply);

    // 토큰 잔액을 확인하는 함수 호출
    var 성은_balance = await contract.balanceOf(성은);
    console.log(성은_balance);

    // 토큰 전송 함수 호출
    await contract.transfer(철수, 100, { from : 성은 });

    // 토큰 잔액을 확인하는 함수 호출
    var 철수_balance = await contract.balanceOf(철수);
    console.log(철수_balance);

})
```

테스트 종류 - Positive & Negative

Positive test

요구 사항에 따라 유효한 데이터 입력하여 프로그램이 예상대로 동작하는지 테스트
ex) 토큰을 양수 개 전송, 토큰을 존재하는 주소로 전송

```
assert.equal(프로그램 실제 기능 실행, 예상 결과 값, 실패 시 출력 문자열);
```

Negative test

요구 사항을 따르지 않고 원치 않은 데이터 입력 및 동작을 시행하여 프로그램이 예상대로 동작하는지 테스트
ex) 토큰을 음수 개 전송, 토큰을 존재하지 않는 주소(0)로 전송

```
await expectRevert.unspecified( 실패할 프로그램 기능 실행 );
```

expectRevert를 사용하기 위해서는 추가적인 라이브러리가 필요하다. (npm init -> npm install --save openzeppelin-test-helpers chai)

테스트 코드 작성 - test/test.js

```
const contractName = artifacts.require('mytoken');
const { expectRevert } = require('openzeppelin-test-helpers')

contract('mytoken contract test', async function(선은, 철수, 영희, 민수) {

  beforeEach(async function () {
    this.contract = await contractName.new();
  })

  describe("시나리오 1 - 토큰의 기본 정보 테스트", function () {
    it("배포된 컨트랙트의 토큰 이름이 지정한 이름과 동일한가?", async function () {
      // 테스트 작성
    })

    it("배포된 컨트랙트의 토큰 심볼이 지정한 심볼과 동일한가?", async function () {
      // 테스트 작성
    })

    it("배포된 컨트랙트의 토큰 초기 발행량이 지정한 발행량과 일치하는가?", async function () {
      // 테스트 작성
    })
  })

  describe("시나리오 2 - 토큰 전송 테스트", function () {
    it("영은이 철수에게 토큰을 100개 전송하면 철수의 토큰 잔액이 100개가 되는가?", async function () {
      // 테스트 작성
    })

    it("영은이 철수에게 토큰을 100개를 전송하고 또 다시 100개를 전송하면 철수의 토큰 잔액이 200개가 되는가?", async function () {
      // 테스트 작성
    })
  })
})
```


커버리지(Coverage)

Coverage는 소프트웨어의 테스트를 논할 때, 진행된 테스트가 얼마나 프로젝트의 기능을 커버하는 지, 테스트가 충분히 진행되었는 지를 나타내는 지표 중 하나

- 코드의 실행량
- 코드가 진행되는 가지 수 (내부 조건 if..)
- 기능(함수) 실행 빈도

Solidity Coverage 설치 및 실행

```
$ npm install --save solidity-coverage // 설치
```

```
$ ./node_modules/.bin/solidity-coverage // 실행
```

커버리지(Coverage) 결과

터미널 출력 결과

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/ mytoken.sol	43.86	25	54.55	43.86	... 142,143,144
All files	43.86	25	54.55	43.86	

Html 결과 파일 - coverage/contracts/index.html, mytoken.sol.html

all files contracts/

43.86% Statements 25/57 25% Branches 6/24 54.55% Functions 12/22 43.86% Lines 25/57

File	Statements	Branches	Functions	Lines
mytoken.sol	43.86% 25/57	25% 6/24	54.55% 12/22	43.86% 25/57

```
contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) internal _balances;

    mapping (address => mapping (address => uint256)) internal _allowed;

    uint256 private _totalSupply;

    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }

    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowed[owner][spender];
    }

    function transfer(address to, uint256 value) public returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }

    function approve(address spender, uint256 value) public returns (bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender] = value;
        emit Approval(msg.sender, spender, value);
        return true;
    }
}
```

배포 스크립트 작성 - migrations/2_deploy_contract.js

```
const mytoken = artifacts.require("mytoken");

module.exports = function(deployer) {
  deployer.deploy(mytoken);
};
```

컨트랙트 배포 명령어

```
$ truffle migrate
```

truffle migrate 결과 값

2_deploy_contract.js

=====

Deploying 'mytoken'

> transaction hash: 0x083701364893e2a3386666d393690a443ff9666706d72dfabd92a682e5fdec13 // 해당 배포 트랜잭션 해쉬

> Blocks: 0 Seconds: 0

> contract address: 0x1E5ca95ae08483454817Abb87C2a408a5C52BF0D // 배포된 컨트랙트의 주소

> block number: 3

> block timestamp: 1565070743

> account: 0xb01523475d6Ce3FCd22baD4545741dd003e5d532 // 컨트랙트를 배포한 account = ganache-cli의 0번째 account

> balance: 99.96441964

> gas used: 1475602

> gas price: 20 gwei

> value sent: 0 ETH

> total cost: 0.02951204 ETH // 배포 시 사용된 가스 비용

> Saving migration to chain.

> Saving artifacts

> Total cost: 0.02951204 ETH

감사합니다.