솔리디티 개발 실전

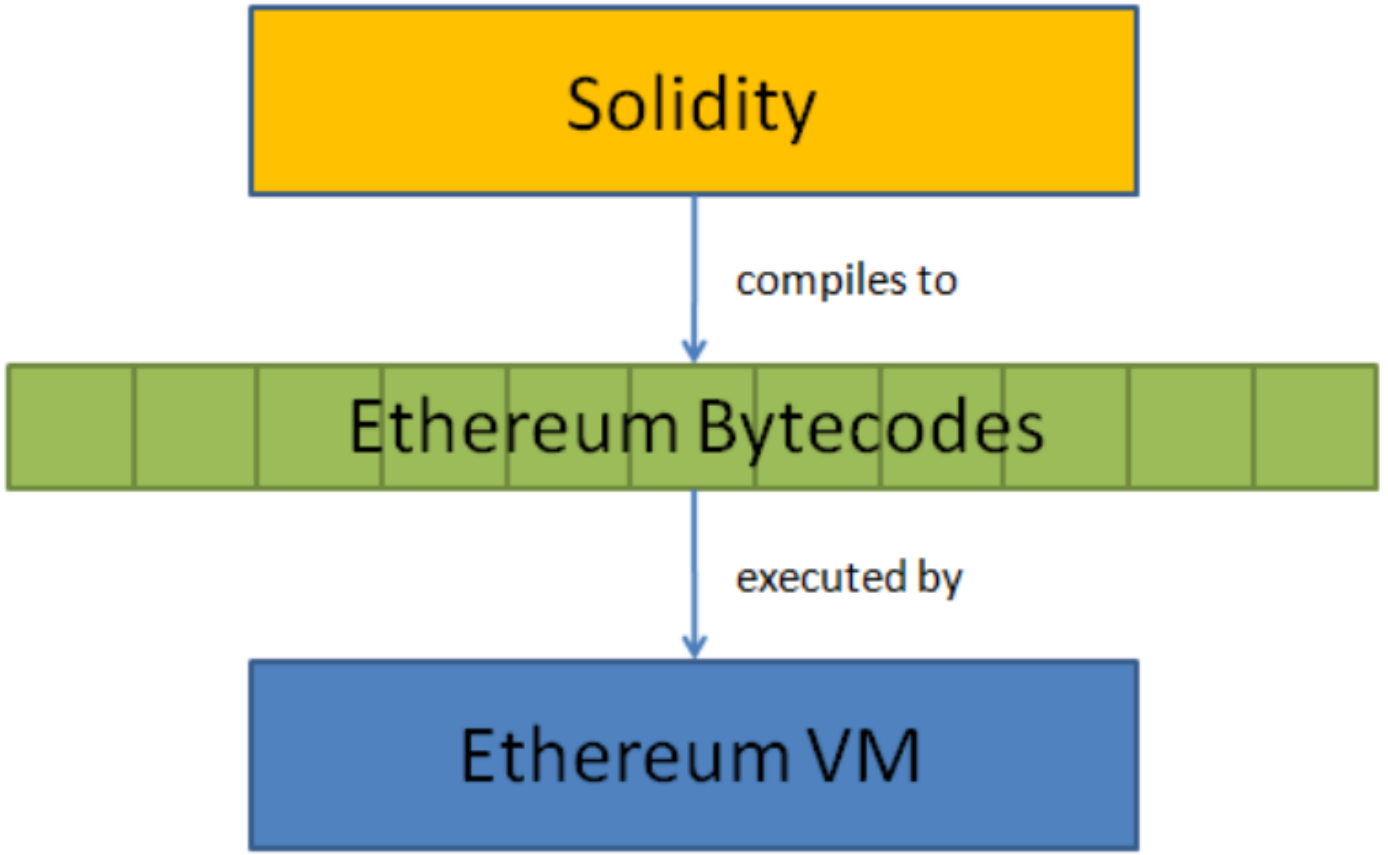
dApp 에 필요한 솔리디티 규칙을 익히고, 토큰을 발행 및 일번 서버에 연동해 보자.

목차

1. 솔리디티 언어 이해
2. ERC20과 ERC721(NFC)
3. 스마트컨트랙트 개발과정
4. 이더리움 dApp, 문서 증명 개발

솔리디티 작동 과정 - 솔리디티 언어 이해

- 솔리디티로 짠 코드는 컴파일러에 의해 기계어로 만들어짐
- 이것을 Ethereum Bytecode 라 함
- 이 바이트코드를 EVM 이 실행
- EVM은 하나의 기계, Machine



EVM 동작 순서 - 솔리디티 언어 이해

EVM이 동작하는 순서는 다음과 같습니다.

1. 트랜잭션이 올바른 형식인지 확인
2. 트랜잭션 수수료 계산(Gas limit x Gas Price)
3. Gas 지불 초기화. 이 시점부터 트랜잭션에서 처리된 바이트만큼 특정 양의 가스를 차감
4. 트랜잭션 금액을 수신 계정으로 보냄 (Smart Contract도 이 단계에서 실행됨)
5. 송신 계정에 트랜잭션을 완료할 수 있을 만큼 Gas Price가 충분하지 않으면 트랜잭션의 모든 변경 사항이 되돌려짐. 그러나, 트랙잰션 수수료는 채굴자에게 지불되고 환불되지 않음.
6. 5번의 경우와 다른 이유로 트랜잭션이 실패한 경우, 송신 계정에 Gas price를 환불하고, 채굴자가 사용된 Gas와 관련된 비용은 채굴자에게 전달됨

솔리디티 언어 이해

- 솔리디티 데이터와 타입
- 솔리디티는 다른 언어처럼 ‘값 ' 과 ‘참조 ' 타입 있음

(1) 정수 타입

(1) 부호 있는 정수 : int8, int16, int24 ---int256
(2) 부호 없는 정수 : uint8, unint16, ---uint256
(3) 비교 연산자 : <=, <, ==, !=, >=, >
(4) 비트 연산자 : &(and), |(or), ^(XOR), ~(NOT)
(5) 산술 연산자 : +, -, *, /, %, **(지수연산),<<(왼쪽 시프트), <<

솔리디티 언어 이해

2)논리 타입

NOT: !

AND: &&

OR: ||

EQUAL: ==

NOT EQUAL: !=

3)주소 타입

주소타입(address)은 20바이트임

비교 연산자 사용 가능

초깃값은 0x0

멤버함수:

- balance : 주소의 이더를 wei로 리턴
- transfer : 이더 송금
- send : 이더 송금, 못하면 false 리턴
- call : 이더 송금, 가스량 조절 가능
- delegatecall : 다른 계약 호출 메소드

솔리디티 언어 이해

4) 바이트 배열 타입

bytes1, bytes2 … bytes32 선언가능

bytes : 참조 타입으로 동적 크기 바이트 배열

string : UTF-8 인코딩 문자열

5) 열거 타입

enum은 리턴값으로 uint8 타입 전달

솔리디티 언어 이해

6) 함수 타입

함수 제한자

- modifier: 이 키워드로 다양한 조건을 부여할 수 있음
- constant: 상수를 선언하는 키워드
- view: 상태를 변경하지 않도록 선언, 변경 실행하면 경고 출력
- pure: 함수 상태 변경 제한, 함수 안 상태변수를 참조하지 않는다는 선언임

접근 제한자

- external: 외부 계약 및 거래에서 호출
- public: 계약 내부 또는 메시지로 외부에서 호출
- internal: 계약 내부 또는 계약을 상속받는 계약에서 호출
- private: 계약 내부에서만 호출

솔리디티 언어 이해

7) 데이터 위치 타입
memory : 일반 연산을 위한 데이터 저장, 메모리
storage : 2차 저장장치, IPFS, SWARM
  - 함수 파라미터 리턴값은 memory를 기본 데이터 위치로함
  - 지역변수, 상태변수는 storage를 기본 데이터 위치로 함
  - 상태변수는 자바의 멤버변수와 같음.
8) 배열 리터럴 타입
 고정 배열 선언 : uint[3] result=[1,2,3];
 동적 배열 선언 : uint[ ] dynaminArr;

솔리디티 언어 이해

9) 구조체

구조체를 정의한 컨트랙트 안에서만 사용

값을 변수에 복사하지 않고 참조함

```
struct User {
        address account;
        string name;
}
```

솔리디티 언어 이해

10) 삭제 연산자

해당 타입에 초깃값을 대입

동적 배열은 배열 요소 크기를 0으로 설정

구조체는 모든 필드를 재설정

```
delete unusedVar;
```

솔리디티 언어 이해

11) 매핑 타입

- mapping(키 => 값) 형태로 정의
- 키로 값을 참조할 수 있는 집합 형태
- 키로 구조체를 제외한 대부분 타입 가능, mapping, 동적 배열.
- 값으로는 모든 타임을 가능, mapping 포함
- 해시 테이블 형태로 어떤 타입이든 사용할 수 있도록 키 초기화

솔리디티 언어 이해

12) 타입 캐스팅

컴파일러는 자동으로 타입 캐스팅

```
int8 y = -3;
uint8 x = uint(y);
```

13) 통화 단위 예약어

wei, szabo, finney, ether

14) 시간단위 예약어: second, minute, hour, day, week, year

전역 변수 – 솔리디티 이해

block.blockhash(uint blocknumber) return (bytes32)

- 지정한 블록의 해시값

block.gaslimit(uint) : 현재 블록 번호

block.timestamp(uint) : 현재 블록의 타임스탬프

msg.data(bytes) : 완전한 호출 데이터

msg.gas(uint) : 남은 가스량

msg.sender(address) : 송금자 주소, 함수를 호출한 계정의 주소

msg.sig(bytes4) : calldata 의 첫 4바이트를 리턴, 함수 식별자사용

tx.gasprice(uint) : 해당 거래의 가스 가격

this : 현재 계약의 어드레스

솔리디티 실습1 – 솔리디티 이해

```
1  pragma solidity ^0.4.19;
2
3  contract TimeUintSample{
4      uint public startTime;
5
6      function start() public {
7          startTime = now;
8          startTime = block.timestamp;
9      }
10
11     function minutesAfter(uint min) public constant returns (bool){
12         if(startTime == 0)
13             return false;
14         return ((now - startTime) / 1 minutes >= min);
15     }
16
17     function getSeconds() public constant returns (uint) {
18         require(startTime > 0);
19         return (now - startTime);
20     }
21 }
```

예외 처리 함수 – 솔리디티 이해

- assert(bool condition) :
  - 내부 에러를 나타낼 때 사용, 조건 만족 안하면 에러 발생
- require(bool condition) :
  - 조건이 만족하지 않으면 에러 발생.
  - 에러 발생 시 해당 처리까지만 가소를 소비하고 잔액 반환
- revert() :
  - 에러 발생하면 상태를 거래 처리 전으로 되돌림.

암호화 함수 – 솔리디티 이해

- keccak256(...) returns (bytes32)
  - Ethereum-SHA-3(KECCACK-256) 해시를 계산
- sha256(...) returns (bytes32)
  - SHA-256 해시를 계산
- sha3(...) returns (bytes32)
  - Keccack256 별칭

계약 함수 – 솔리디티 이해

- this(address)
  - 현재 계약을 뜻한다
  - 계약 주소를 명시해서 바꿀 수 있다
- selfdestruct(address recipient)
  - 현재 계약을 파기해 지정한 주소로 금액을 보낸다.
  - 문제가 발생하여 계약 배포를 중지할 때 사용한다
- suicide(address recipient)
  - Selfdestruct 별칭

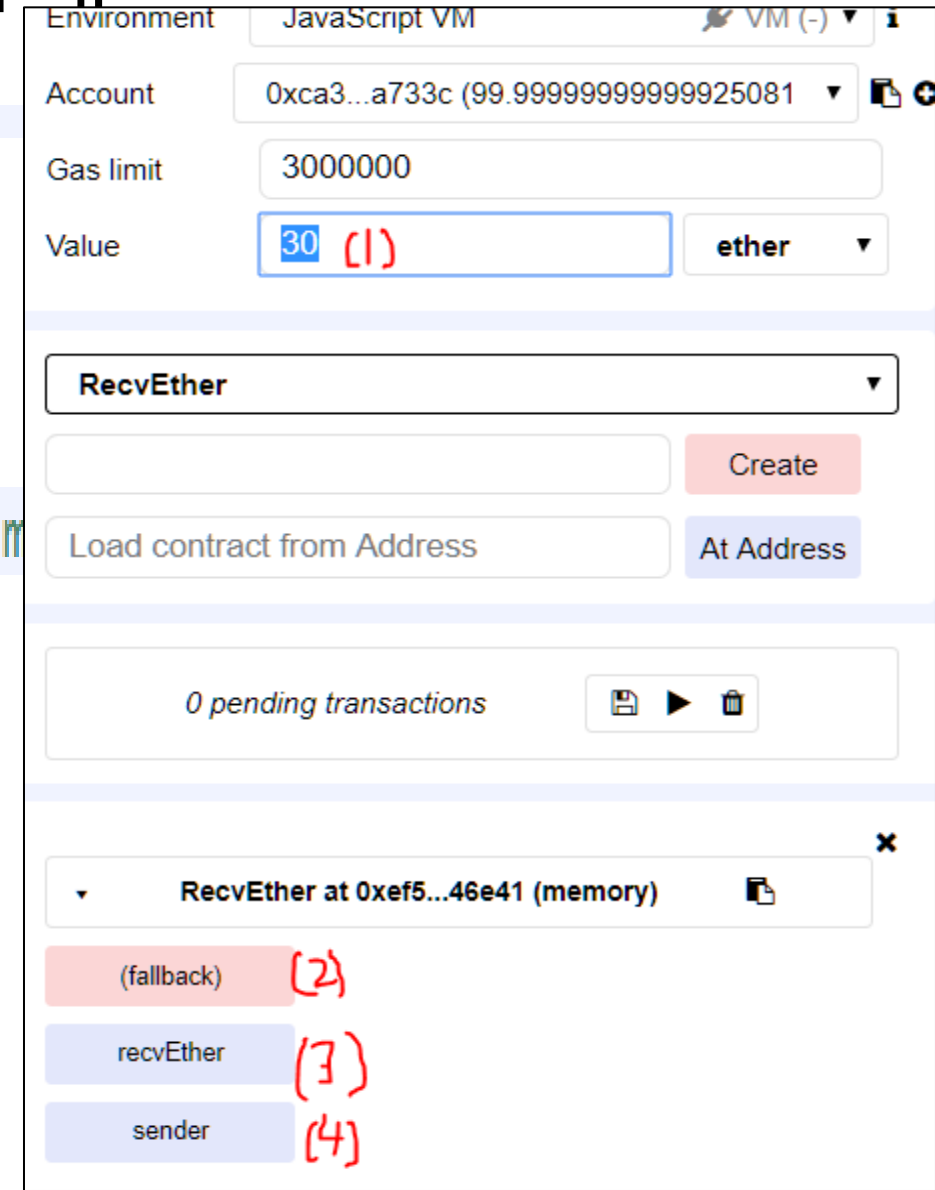
```
pragma solidity ^0.4.19;

contract SelfDestructSample{
    address public owner = msg.sender;

    function() payable{

    }

    function close(){
        require(msg.sender == owner);

        selfdestruct(owner); // == suicide(owner);
    }

    function Balance() constant returns (uint) {
        return this.balance;
    }
}
```

15) 입력과 출력 파라미터 – 솔리디티 이해

- 입력 파라미터는 변수와 같은 방식으로 선언
- returns 키워드 다음에 출력 파라미터 선언
- 출력 파라미터의 이름은 생략가능
- 출력 값은 하나, 여러 값을 리턴 가능
- 외부 함수 호출로 다른 계약의 함수를 호출할 때, 이더를 전송하는 경우 payable 키워드를 지정해야 함.

솔리디티 실습2 – 솔리디티 이해

```
1  pragma solidity ^0.4.19;
2
3  contract RecvEther{
4      address public sender;     // first value set
5      uint public recvEther;     // after call payable, m
6
7      function() public payable{
8          sender = msg.sender;
9          recvEther += msg.value;
10     }
11 }
```



16) 계약 – 솔리디티 이해

- 생성자는 선택 사양
- 이름없는 함수(fallback function)
  - 이름, 파라미터, 리턴값이 없는 함수
  - 거래나 메시지에서 지정한 함수가 계약 내에 없는 경우
  - 이더를 송금하는 경우, payable 키워드와 함께 사용한다

```
function() public {
    revert();
}
```

- event : 거래 로그를 출력, event CustomEvent(uint type);
- 상속 : is 키워드를 사용해 상속한다, contract A is B

솔리디티 실습3-1 – 토큰 컨트랙트

- 토큰 컨트랙트 만들때 필요한 기본 요소로
- 토큰의 이름, 단위, 총 발행량, 잔고 관리, 송금기능을 기본적으로 가지고 그 외의 기능도 가질 수 있음

솔리디티 실습3-2 – 토큰 컨트랙트

```
pragma solidity ^0.4.19;
contract CrocusCoin{
    // 1. 상태변수
    string public name; // 토큰 이름
    string public symbol; // 토큰 단위
    uint8 public decimals; // 소수점 이하 자릿수
    uint public totalSupply; // 토큰 총 발행량

    mapping(address = > uint) public balanceOf; // 각 주소의 잔고

    // 2. 이벤트
    event Transfer(address indexed from, address indexed to, uint value);

    // 3. 생성자
    function CrocusCoin(uint _supply, string _name, string _symbol, uint8 _decimals) {
        balanceOf[msg.sender] = _supply;
        name = _name;
        symbol = _symbol;
        decimals = _decimals;
        totalSupply = _supply;
    }

    // 4. 송금 기능
    function transfer(address _to, uint _value) {
        require(balanceOf[msg.sender] >= _value);
        require(balanceOf[_to] + _value >= balanceOf[_to]);

        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;

        Transfer(msg.sender, _to, _value);
    }
}
```

솔리디티 실습3-3 – 토큰 컨트랙트

1. 상태변수

상태변수는 위와같이 선언할 수 있다. 이때 mapping을 통해 해당 address가 가지는 토큰 잔고를 보관하자.

2. 이벤트

이벤트는 컨트랙트가 블록체인 상에서 사용자 단의 앱에서 액션이 발생할 경우 귀를 기울이고 있는 상황을 의미한다.

이렇게 이벤트를 설정하면 특정 이벤트가 발생하면 블록체인 상에서 행동을 취하게 된다.

이러한 이벤트는 계약 중 발생한 처리를 추적할 수 있게 도와주는 로그를 출력해준다.

3. 생성자

생성자에서 balanceOf[msg.sender] = _supply를 통해 모든 토큰을 최초 계약을 생성한 사람의 주소로 보내주게 된다 즉, 어떤 컨트랙트를 개발한 창시자에게 모든 토큰이 가있고 그로부터 시작된다는 의미이다.

4. 송금 기능

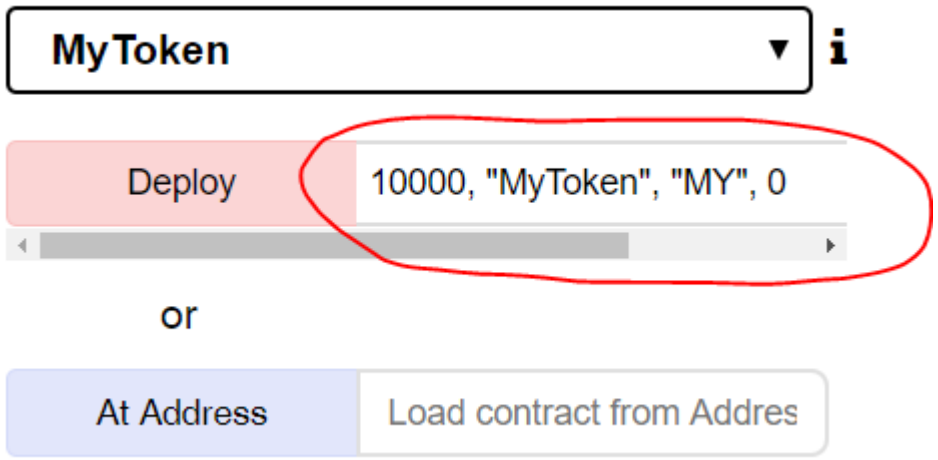
송금 기능에서 require를 통해 두가지를 검사한다.

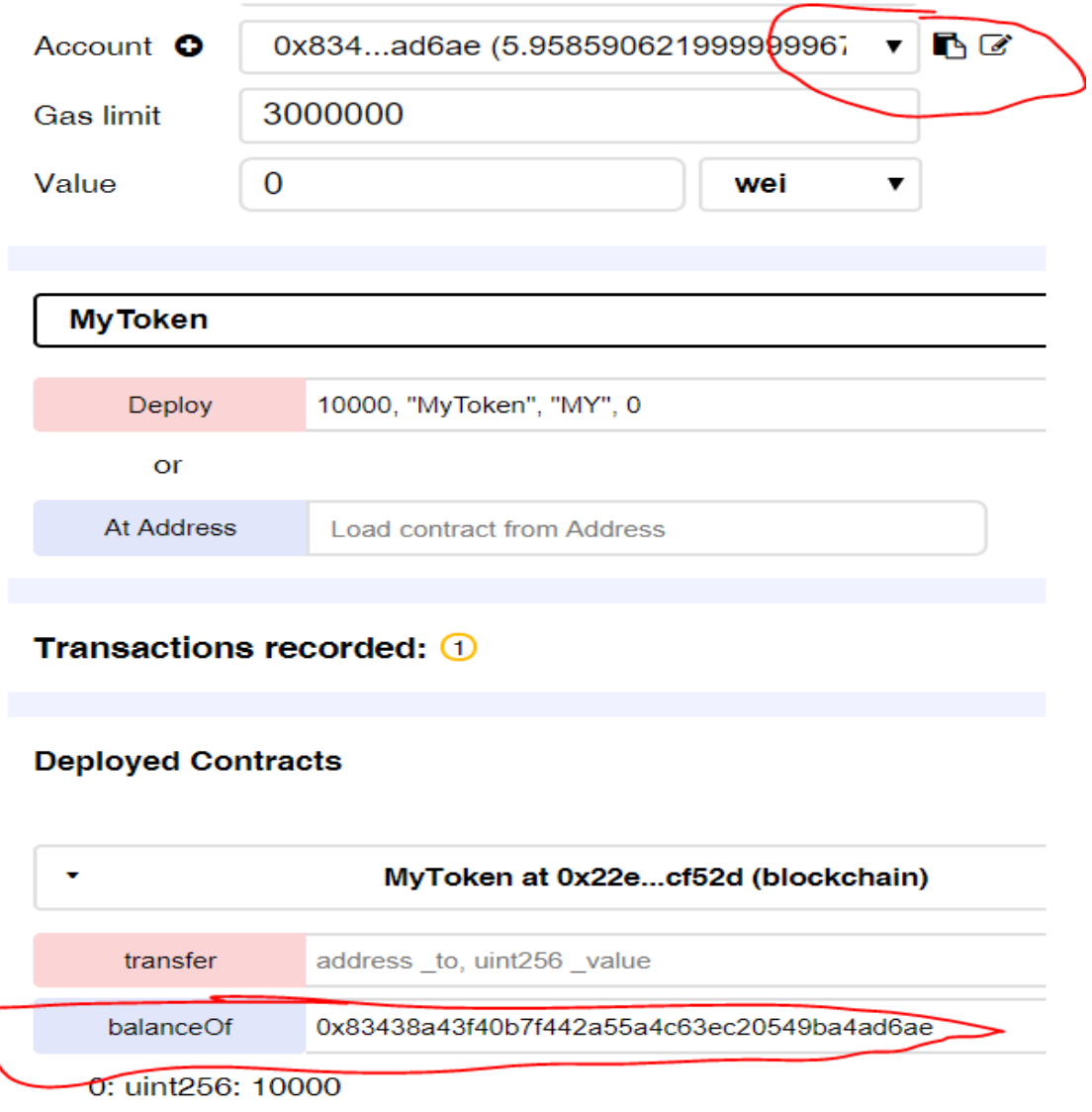
1. 현재 트랜잭션을 보내려는 유저의 보유 토큰이 자신의 자신이 보내려 하는 토큰보다 크거나 같은지 체크한다.
2. 받는 사람의 토큰 보유량 + 받으려는 토큰이 받는 사람의 토큰 보유량 보다 크거나 같은지 체크한다.(오버 플로우 체크)

그리고 마지막에 이벤트를 호출하며 로그를 남기게 된다.

솔리디티 실습3-4 – 토큰 컨트랙트

5. 실행 과정

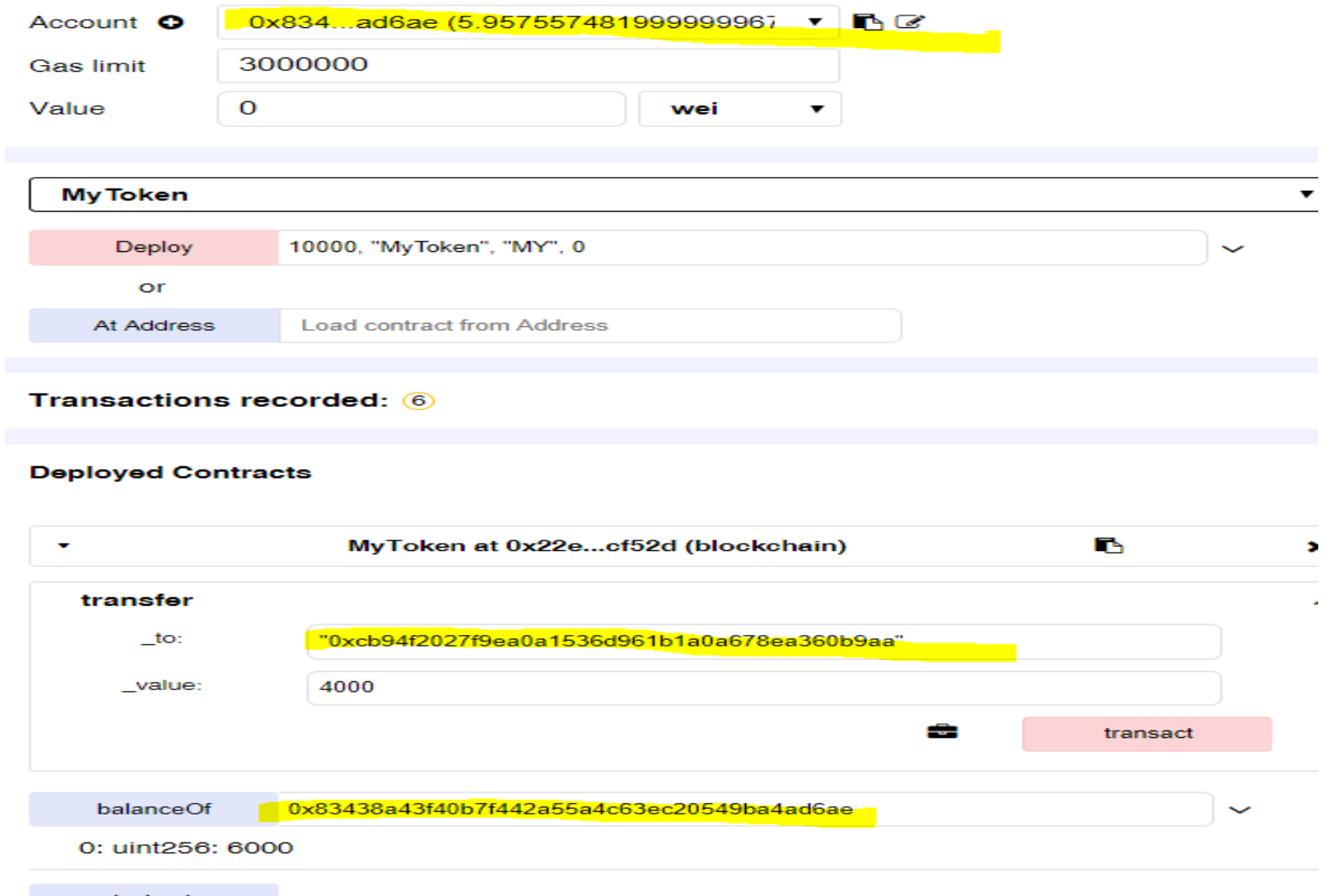




소유자 주소를 복사하여 여서 잔액을 체크해 본다

코인 컨트랙트를 Deploy하여 배포한다
총 발행량은 10000개, MyToken 단위·
MY로 하고 소숫점은 0자리로 한다.

솔리디티 실습3-5 – 토큰 컨트랙트



1) 메타마스크에서 계정을 하나 더 만듦
2) 새로운 주소를 transfer 에 전송하려는 토큰액과 같이 입력
3) 소유자 계정으로 실행해야 하니, 메타마스크에서 토큰 컨트랙트 계정으로 변경
4) Transfer 함수 클릭
5) blanaceOf 함수 클릭해서 줄어든 토큰 확인

ERC20과 ERC721(NFT)

- ERC20 은 토큰 컨트랙트 인터페이스를 표준화한 것
- 따라서 같은 규격으로 만들어진 토큰들과 거래할 수 있음
- ERC의 20은 RFC의 20번째 규격임을 뜻한다.
- https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

- ERC721 은 NFT(Non-fungible-Tokens)을 표준화한 것
- NTF 는 게임의 아이템이나 디지털 티켓, 증서처럼 고유한 것
- https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md

스마트컨트랙트 개발과정 정리

스마트 컨트랙트를 만드는 순서는 다음과 같습니다.

1. 스마트 컨트랙트 코딩
   - 구현하고자 하는 내용을 솔리디티나 다른 언어로 코딩합니다.
2. 구현한 소스 코드를 컴파일
   -컴파일 결과 EVM 바이트 코드가 생성됩니다.
3. 스마트 컨트랙트 배포
   -스마트 컨트랙트를 배포한다는 것은 컴파일된 EVM 코드를 하나의 트랜잭션 처럼 블록에 추가시켜 블락체인에 등록시키는 작업.
   -소스 컴파일 -> EVM 바이트 코드 -> 구체적인 작업은 ABI 취득 -> ABI로부터 컨트랙트 객체 생성 -> 트랜잭션 생성하여 블록에 추가
   -마이너가 해당 블록을 채굴하게 되면 블락체인에 포함됨

이더리움 dApp, 문서 증명 개발

전체과정

1. 환경설정
   - Geth 로컬 구축
   - Nodejs 설치
   - 소스 다운로드
2. 스마트컨트랙트 작성 및 배포
3. 웹서버 구축
4. 웹클라이언트 구축

환경구축 - Geth 로컬 가동

- C:₩users₩계정 폴더로 이동
- 시행:
  1. genesis.json 파일을 c:₩users₩user₩geth 폴더 밑으로 이동
  2. geth --datadir docu5 init genesis.json console
  3. geth --networkid 1234 --datadir docu5 --rpc --rpcaddr "0.0.0.0" --rpcport 8545 --rpccorsdomain "*" --rpcapi "admin, db, eth, debug, miner, net, shh, txpool, personal, web3" console
  4. 명령 프롬프트 추가로 띄움
  5. > geth attach rpc:http://localhost:8545 // 원격 연결 가능여부 확인
  6. > personal.newAccount("1234") // 계정 생성
  7. > personal.unlockAccount(eth.accounts[0],"1234")
  8. > miner.start()
  9. > eth.mining
  10. > eth.getBalance(eth.accounts[0]) // 채굴해서 코인생겼는지 확인

환경구축 – Nodejs 설치

- https://nodejs.org/ko/

- 10.16.0 안전한 버전 설치

환경구축 – 소스 다운로드

- https://github.com/ganadara135/dAppDocu
- Download 버튼 클릭
- 원하는 위치에 복사
- 원하는 에디터로 오픈(Visual Code)

스마트컨트랙트 작성 및 배포 1/2

```
1   pragma solidity ^0.4.0;
2
3   contract Proof
4   {
5       struct FileDetails
6       {
7           uint timestamp;
8           string owner;
9       }
10      mapping (string => FileDetails) files;
11      event logFileAddedStatus(bool status, uint timestamp, string owner, string fileHash);
12
13      //this is used to store the owner of file at the block
14      function set(string owner, string fileHash) public
15      {
16          if(files[fileHash].timestamp == 0)
17          {
18              files[fileHash] = FileDetails(block.timestamp, owner);
19              logFileAddedStatus(true, block.timestamp, owner, fileHash);
20          }
21          else
22          {
23              logFileAddedStatus(false, block.timestamp, owner, fileHash);
24          }
25      }
26      //this is used to get file information
27      function get(string fileHash) public returns (uint timestamp, string owner)
28      {
29          return (files[fileHash].timestamp, files[fileHash].owner);
30      }
31  }
```

스마트컨트랙트 작성 및 배포 2/2

- 리믹스에 소스 작성
- 배포는 geth RPC 에 연동
- 계정 설정시 코인베이스 설정(이더 채굴한 계정)

웹 서버 구축

- Package.json 위치서 아래 명령어 시행
  - Npm install
  - 패키지 다 설치될때까지 대기
- App.js 파일을 열고, 아래 위치의 어드레스를 Geth 코인베이스로 수정
  - var proof = proofContract.at("0x83438A43F40b7f442a55a4C63EC20549ba4AD6ae");
- 웹서버 가동
  - node app.js

웹클라이언트 구축

- 웹브라우저 창에서
  - http://localhost:8080 입력

- 원하는 파일 선택
- 소유자명 입력
- Send 보냄

- 원하는 파일 선택
- getInfo 버튼 누름