ERC20 토큰 발행

ERC20 Token Issuance

ERC20 : 이더리움 토큰 표준

```
1  contract ERC20Interface {
2       function totalSupply() public view returns (uint);
3       function balanceOf(address tokenOwner) public view returns (uint balance);
4       function allowance(address tokenOwner, address spender) public view returns (uint
   remaining);
5       function transfer(address to, uint tokens) public returns (bool success);
6       function approve(address spender, uint tokens) public returns (bool success);
7       function transferFrom(address from, address to, uint tokens) public returns (bool
   success);
8
9       event Transfer(address indexed from, address indexed to, uint tokens);
10      event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
11  }
```

ERC20 토큰 발행

```
pragma solidity ^0.4.24;

contract DwidderToken {
    address public owner;
    string public name = “Dwidder”;
    string public symbol = “DWD”;
    uint8 public decimals = 8;
    uint256 public totalSupply = 0;

    mapping(address => uint256) balances;
    mapping(address => mapping(address => uint256)) internal allowed;
}
```

ERC20 토큰 발행

```
event Transfer(address indexed from, address indexed to, uint256 value);
event Approval(address indexed owner, address indexed spender, uint256 value);
event Mint(address indexed to, uint256 amount);

construtor() public {
    owner = msg.sender;
}
```

ERC20 토큰 발행

```
function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[msg.sender]);

    // TODO: sender의 잔액을 감소시킨다.
    // TODO: _to의 잔액을 증가시킨다.
    // TODO: Transfer 이벤트를 발생시킨다.
    return true;
}

function balanceOf(address _owner) public view returns (uint256 balance) {
    // TODO: _owner의 잔액을 반환한다.
}
```

ERC20 토큰 발행

```
1  function transferFrom(address _from, address _to, uint256 _value) public returns
   (bool) {
2      require(_to != address(0));
3      require(_value <= balances[_from]);
4      require(_value <= allowed[_from][msg.sender]);
5
6      // TODO: _from의 잔액을 감소시킨다.
7      // TODO: _to의 잔액을 증가시킨다.
8      // TODO: _from으로부터 msg.sender로의 allowance를 _value 만큼 감소시킨다.
9      // Transfer 이벤트를 발생시킨다.
10     return true;
11 }
12
```

ERC20 토큰 발행

```
function approve(address _spender, uint256 _value) public returns (bool) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    // TODO: msg.sender로부터 _spender로의 allowance를 _value로 설정한다.
    // TODO: Approval 이벤트를 발생시킨다.
    return true;
}

function allowance(address _owner, address _spender) public view returns (uint256) {
    // TODO: _owner로부터 _spender로의 allowance를 반환한다.
}

```

ERC20 토큰 발행

```
function mint(address _to, uint256 _amount) public returns (bool) {
    require(msg.sender == owner);

    // TODO: 총 공급량에 발행량을 추가한다.
    // TODO: _to의 잔액을 증가시킨다.
    // TODO: Mint 이벤트를 발생시킨다.
    // TODO: Transfer 이벤트를 발생시킨다.
    return true;
}
```

변경자(MODIFIER)

```
1  contract MyContract {
2      address public owner;
3      uint public data;
4
5      modifier onlyOwner {
6          require(msg.sender == owner);
7          _;
8      }
9
10     function updateData(uint _data) onlyOwner {
11         data = _data;
12     }
   }
```

require 뒤의 조건문이 false이면 함수 수행이 중단된다.

QUIZ: 변경자가 적용된 DWIDDERTOKEN

01

OWNABLE 계약

Ownable 계약은 owner라는 상태 변수를 가지고 있고 생성자에서 msg.sender의 값으로 초기화된다.

02

ONLYOWNER 변경자

Ownable 계약은 owner만 실행을 허용하는 onlyOwner 변경자를 갖는다.

03

DWIDDERTOKEN 계약

DwidderToken 계약은 Ownable을 상속하여 onlyOwner를 도입한다.

ZEPPELIN-SOLIDITY

https://github.com/OpenZeppelin/openzeppelin-solidity

01

오픈 소스 라이브러리

Github에 공개되어 있고 누구나 개발에 참여 가능.

02

COMMUNITY AUDIT

취약점이 발견되면 이슈를 등록하고 이를 수정하는 방식으로 보안성 제공.

03

표준 토큰 계약 제공

ERC20, ERC721, ERC827 등 이더리움 기반의 표준 토큰 계약을 제공.

04

표준 크라우드 세일 계약 제공

토큰 판매를 위한 표준 크라우드 세일 계약 제공.

STANDARDTOKEN.SOL

https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/StandardToken.sol

```
1  import “zeppelin-solidity/contracts/token/ERC20/
   StandardToken.sol";
2
3  contract MyToken is StandardToken {
4      string public name = ‘Bitcoin';
5      string public symbol = ‘BTC';
6      uint8 public decimals = 8;
7
8      constructor() public {
9          totalSupply_ = 21000000;
10         balances[msg.sender] = totalSupply_;
11     }
12 }
```

ERC20의 기능을 모두 구현해놓은 표준 토큰.

StandardToken을 상속해서 MyToken 정의.

라이브러리(LIBRARY)

라이브러리는 자주 사용되는 함수를 모아놓은 것으로써, using 키워드로 특정 자료형에 대해 라이브러리를 적용할 수 있다.

```
1  pragma solidity ^0.4.24;
2
3  import "./ERC20Basic.sol";
4  import "../../math/SafeMath.sol";
5
6  contract BasicToken is ERC20Basic {
7    using SafeMath for uint256;
8  …
9      balances[msg.sender] = balances[msg.sender].sub(_value);
10 …
11 }
12
```

형변환(type casting)

실습: BURNABLE, MINTABLE, PAUSABLE 토큰 구현

01

BURNABLETOKEN

토큰 소각 기능 (zeppelin-solidity/contracts/token/ERC20/Burnable.sol)

02

MINTABLETOKEN

토큰 추가 발행 기능 (zeppelin-solidity/contracts/token/ERC20/MintableToken.sol)

03

PAUSABLETOKEN

사용 중지 기능 (zeppelin-solidity/contracts/token/ERC20/PausableToken.sol)

04

다중 상속으로 구현

위 세가지 계약을 다중 상속해서 구현.

계약 계정 참조 1

```
1  import “./MyToken.sol”;
2
3  contract MyContract {
4
5      MyToken internal token;
6
7      function MyContract(address _token) {
8          token = MyToken(_token);
9      }
10 }
11
12
```

계약 계정 참조 2

```
1  import “./MyToken.sol”;
2
3  contract MyContract {
4
5      MyToken internal token;
6
7      function MyContract(MyToken _token) {
8          token = _token;
9      }
10 }
11
12
```

QUIZ : 유연성 토큰 (FLEXIBLE TOKEN)

기기

01

ERC20 토큰

유연성 토큰은 하나의 ERC20 토큰이므로 해당 기능을 제공해야 한다.

02

토큰 위임

유연성 토큰은 그 자체로써 기능이 있는 것이 아니라 위임된 토큰의 ERC20 함수를 대리 호출해줄 뿐이다.

03

위임 토큰 지정

유연성 토큰 생성 시 하나의 위임 토큰이 지정되어 있어야 하고, 추후에 소유자(owner)만 위임 토큰을 변경할 수 있다.

CROWDSALE.SOL

https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/crowdsale/Crowdsale.sol

```
1  import “github.com/OpenZeppelin/zeppelin-solidity/contracts/
   token/ERC20/StandardToken.sol";
2
3  contract MyToken is StandardToken {
4      string public name = ‘Bitcoin';
5      string public symbol = ‘BTC';
6      uint8 public decimals = 8;
7
8      constructor() public {
9          totalSupply_ = 21000000;
10         balances[msg.sender] = totalSupply_;
11     }
12 }
```

ERC20의 기능을 모두 구현해놓은 표준 토큰.

StandardToken을 상속해서 MyToken 정의.

실습: 크라우드 세일 배포 및 테스트

01

CROWDSALE

크라우드 세일 계약을 배포 (교환 비율, 지갑, 토큰 주소를 설정)

02

토큰 구입

트랜잭션 발생을 통해 Fallback 함수가 호출되도록 한다.

03

ETH 입금 확인

지갑 계정에 ETH가 잘 입금되었는지 확인한다.

04

토큰 분배 확인

토큰이 잘 구매자 계정으로 잘 분배되었는지 확인한다.

실습: 커스텀 크라우드 세일 구현

01

ALLOWANCECROWDSALE

ERC20의 allow 기능을 활용해 토큰을 공급(zeppelin-solidity/contracts/crowdsale/emission/AllowanceCrowdsale.sol)

02

CAPPEDCROWDSALE

최대 모금액(cap) 설정 가능 (zeppelin-solidity/contracts/crowdsale/validation/CappedCrowdsale.sol)

03

TIMEDCROWDSALE

시작/종료 시각 설정 (zeppelin-solidity/contracts/crowdsale/validation/TimedCrowdsale.sol)

04

POSTDELIVERYCROWDSALE

세일이 종료될 때까지 토큰을 출금할 수 없음 (zeppelin-solidity/contracts/crowdsale/distribution/PostDeliveryCrowdsale.sol)

Q&A

무엇이든 물어보세요!

Contact us:

서울시 강남구 논현로75길 5-2 4층

02-6732-2000

support@dnext.co

Follow us on:

facebook.com/dnextco

@dnextco