Proxy를 사용한 Upgradeable Contract

- **각 컨트랙트의 역할**
  - ERC1967Proxy

    Upgrade가 가능한 스토리지를 할당 받아 데이터 충돌 없이 Upgrade가 가능한 프록시를 구현해 주는 Contract입니다.

    왕태현 작성
  - ERC1967Upgrade

    추상 컨트랙트로 getter 함수를 제공하며, EIP1967슬롯과 관련된 업데이트 함수 event를 emit 한다.

    우영하 작성
  - IBeacon

    동일한 논리 계약 주소를 가리키는 여러 프록시가 있는 경우 논리 계약을 할 때 마다 모든 프록시를 업데이트 해야합니다. 프록시 주소를 통합적으로 관리하기 위해서 나온 것이 Beacon Contract입니다. 계약 주소를 통합적으로 관리하기 때문에 관리가 용이하며 업데이트 시 모든 프록시를 업데이트하는 것이 아니라 Beacon Contract만 업데이트하면 되기 때문에 가스비도 줄일 수 있습니다.

    왕태현 작성
  - Address

    컨트랙트 계정을 호출하거나 컨트랙트 진위여부를 판단하는 등 계정타입과 관련된 Contract

    우영하 작성
  - StorageSlot

    컨트렉트 내에 데이터를 저장하는 공간으로 업그레이드 가능한 계약을 처리할 때 스토리지 충돌를 피하기 위해 자주 사용됩니다.

    왕태현 작성
- **implemetated code 첨부 후 설명**
  - ERC1967Proxy

```
/* Functions */
constructor(_logic, _data)     //업그레이드 가능한 프록시를 초기화
_implementation()              //현재 구현 주소
_getImplementation()           //현재 구현 주소 반환
_upgradeTo(newImplementation)  //upgraded이벤트를 발생
*(Upgraded이벤트는 아래 Events 항목에)
_upgradeToAndCall(newImplementation, data, forceCall)
//추가 설정 호출로 구현 업그레이드를 수행해줌
_upgradeToAndCallSecure(newImplementation, data, forceCall)
//UUPS프록시에 대한 보안 검사 및 추가 설정 호출로 구현업그레이드를 수행
_getAdmin()
//현재 관리자를 반환해줌
_changeAdmin(newAdmin)
//프록시의 관리자 반환
_getBeacon()
//현재 비콘 반환
```

```
_upgradeBeaconToAndCall(newBeacon, data, forceCall)
//추가 설정 호출로 비콘 업그레이드를 수행(비콘의 주소를 업그레이드하지만 비콘에 포함된
구현은 업그레이드x)
_delegate(implementation)
//호출을 위임하는 함수(외부 호출자에게 직접 반환)
_fallback()
//반환한 주소로 현재 호출 위임(내부 호출 사이트로 돌아가지 않고 외부 호출자에게 직접 반환)
fallback()
//반환한 주소로 호출을 위임하는 대체 함수
receive()
//반환된 주소로 호출을 위임하는 대체 함수(call 데이터가 비어있으면 실행)
_beforeFallback()
//구현으로 fallback하기 전 호출되는 함수
//_fallback 호출의 일부로,  fallback 또는 receive기능의 일부로 발생할 수 있음
/* Events */
Upgraded(implementation)
//구현이 업그레이드될 때 발생
AdminChanged(previousAdmin, newAdmin)
//관리자 계정이 변경되면 발생
BeaconUpgraded(beacon)
//비콘이 업그레이드될 때 발생
```

도형우 작성

- ERC1967Upgrade

```
/* Functions */
_getImplementation()
// 현재 구현체 주소를 반환한다.
_upgradeTo(newImplementation)
// 구현체 주소를 바꾼다. (upgarded 이벤트를 발생시킨다.)
_upgradeToAndCall(newImplementation, data, forceCall)
// 구현체 주소를 바꾸고 실행한다. (upgarded 이벤트를 발생시킨다.)
_upgradeToAndCallSecure(newImplementation, data, forceCall)
// uups 프록시에 의해 security체크를 진행하고,
// 구현체 주소를 바꾼 후 실행한다. (upgarded 이벤트를 발생시킨다.)
_getAdmin()
// 현재 admin(EOA)계정을 반환한다.
_changeAdmin(newAdmin)
// admin(EOA)계정을 변환한다.
_getBeacon()
// 현재 beacon CA를 반환한다.
_upgradeBeaconToAndCall(newBeacon, data, forceCall)
// beacon CA에 포함되어있는 구현체들은 그대로 두고,
// beacon CA 주소만을 변환하고 Call한다.

/* Events */
Upgraded(implementation)
// 구현체가 업그레이드 될 때 발생하는 이벤트
AdminChanged(previousAdmin, newAdmin)
// Admin계정이 바뀌었을 때 발생하는 이벤트
BeaconUpgraded(beacon)
// Beacon계정이 바뀌었을 때 발생하는 이벤트
```

도형우 작성

- IBeacon

```
/* Functions */
implementation()
// Delegate Call을 사용할 주소를 반환하며 BeaconProxy는 사용할 수 있는 주소인지 확인합니다.
```

왕태현 작성

- Address

```
/* Functions */
isContract(account)
// 입력받은 account가 CA계정인지 확인하고 bool형태로 반환한다.
// iscontract함수는 매개변수로 들어온 account값이 eoa계정이 아닌,
// 1. 외부 소유 계정 주소
// 2. construct 계정 주소
// 3. contract 가 생성될 계정주소
// 4. 파기된 contract 계정주소
// 에도 false 값을 반환할 수 있기 때문에 iscontract함수를 전적으로 의지하는 것은
// 안전하지 않다.
// flash loan 공격에 취약할 수 있다.
sendValue(recipient, amount)
// 기존에 존재하던 transfer 함수를 대체하는 함수이다. recipient에게 amount를 보낼 때 사용된다.
// 모든 권한이 recipient에게 이전되기 때문에, Reentrancy 취약점 이슈가 생길 수 있다.
functionCall(target, data)
// low level call을 수행한다. 일반 call은 안전하지 않기 때문에, functioncall()
// 함수를 이용하는 것을 권장한다.
// revert된 이유를 abi.decode를 사용하여 readable한 자료로 바꿀 수 있다.
functionCall(target, data, errorMessage)
// functioncall(target, data)와 유사하지만 errorMessage를 함께 반환한다.
functionCallWithValue(target, data, value)
// functionCall(target, data)와 유사하지만 target에게 wei값을 함께 반환한다.
// 호출자 contract는 value값 이상의 ETH를 가지고 있어야 한다.
// 호출되는 함수는 payable해야 한다.
functionCallWithValue(target, data, value, errorMessage)
// functionCallWithValue()함수와 같지만 errorMessage를 함께 반환한다.
functionStaticCall(target, data)
// functioncall과 같지만 정적call을 수행한다.
functionStaticCall(target, data, errorMessage)
// functionStaticcall과 같지만 정적 call을수행하며 errorMessage를 함께 반환한다.
functionDelegateCall(target, data)
// functionCall과 같지만 delegatecall을 수행한다.
functionDelegateCall(target, data, errorMessage)
// functionDelegateCall과 같지만 errorMessage를 함께 반환한다.
verifyCallResult(success, returndata, errorMessage)
// lowlevel call이 성공했을 때 verify를 수행하는 함수이다.
// revert됐을때는 verify를 수행하지않으며, revert된 이유를 제공한다.
```

우영하 작성

- StorageSlot

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;


library StorageSlotUpgradeable {
    struct AddressSlot {
        address value;
    }
// address 형태로 저장되어있는 변수들을 관리하는 struct

    struct BooleanSlot {
        bool value;
    }
// Boolean 형태로 저장되어있는 변수들을 관리하는 struct
    struct Bytes32Slot {
        bytes32 value;
```

```
    }
// Bytes32 형태로 저장되어있는 변수들을 관리하는 struct
    struct Uint256Slot {
        uint256 value;
    }
// Uint256 형태로 저장되어있는 변수들을 관리하는 struct
    function getAddressSlot(bytes32 slot) internal pure returns (AddressSlot storage r) {
        assembly {
            r.slot := slot
        }
    }
// AddressSlot형태로 저장된 값들을 struct형태로 반환한다.
// assembly{}안에 있는 코드가 의미하는바를 잘 모르겠습니다.
    function getBooleanSlot(bytes32 slot) internal pure returns (BooleanSlot storage r) {
        assembly {
            r.slot := slot
        }
    }
// assembly{}안에 있는 코드가 의미하는바를 잘 모르겠습니다.
    function getBytes32Slot(bytes32 slot) internal pure returns (Bytes32Slot storage r) {
        assembly {
            r.slot := slot
        }
    }
// assembly{}안에 있는 코드가 의미하는바를 잘 모르겠습니다.
    function getUint256Slot(bytes32 slot) internal pure returns (Uint256Slot storage r) {
        assembly {
            r.slot := slot
        }
    }
// assembly{}안에 있는 코드가 의미하는바를 잘 모르겠습니다.
}
```

우영하 작성

- Proxy의 정의와 사용하는 이유 그리고 각 개발자들이 위 방식으로 erc20을 배포한 이유 등
  - Proxy

[그림 1. Proxy Contract 작동 방식]

그림 1. 은 Proxy Contract의 작동 방식을 보여줍니다. Logic Contract의 Method를 이용하고 싶은 Client는 Logic Contract를 직접 호출하는 것이 아니라, Proxy Contract를 호출하고 있는 모습을 보입니다. 이처럼 Proxy는 호출자가 컨트랙트를 직접 부르는게 아니라 Proxy라고 하는 대리자를 중간에 두고 간접 호출하는 방식의 디자인 패턴입니다. 실제 컨트랙트는 프록시 컨트랙트를 통해서 참조될 뿐입니다. 예로 JAVA 진영의 스프링 프레임워크에서는 데이터를 서버로 보내기전에 캐싱하거나 접근제어 목적으로 쓰입니다. 대표적으로 스프링 AOP의 컨셉이 있습니다. 그러나 이더리움 네트워크의 스마트 컨트랙트는 네트워크에 한 번 배포되면 끝이죠. 수정할 수 없는 상태가 됩니다. 그렇기 때문에 인간의 실수로 탄생한 잘못된 코드 혹은 미래에 발생할 수 있는 문제점을 해결하기 위해 이미 배포된 컨트랙트를 대신해 업그레이드하거나 보안 문제를 해결하는 방법으로 Proxy를 사용합니다.

이러한 Proxy 패턴이 적용된 컨트랙트를 ‘Proxy Contract’ 혹은 ‘Upgradable Smart Contract’ 라고 합니다. 그리고 컨트랙트를 Upgradable하게 만들어 줄 수 있는 세가지 컨셉이 있습니다.

- Data와 Logic을 각각 다른 컨트랙트로 분리 및 관리
- fallback function
- delegate call

Data와 Logic을 다른 컨트랙트로 분리하고 관리하는 것은 ERC20 토큰을 예를 들어 보겠습니다. ERC20 토큰에서 Data는 이름, 심볼, 발행량, 보유량 등이 있습니다. Logic은 전송, 승인 등과 같이 토큰 전송에 관여하는 기능이 있습니다. 이들을 분리하여 관리하면 컨트랙트에 문제가 발생했을 때 오직 Logic만을 수정하여 새로 배포하는 방식으로 문제를 해결할 수 있습니다.

fallback function은 호출된 메소드의 이름이 컨트랙트 내에 존재하지 않을 때 fallback으로 정의한 메소드가 대신 호출되는 기능입니다. 이름, 식별자 없이 정의되며 명시적 호출을 할 수 없습니다. 이는 fallback function이 어떠한 인자도, 반환값도 갖지 않음을 의미합니다. 만약 fallback function이 ETH를 받아야 한다면 payable 키워드를 정의해 줄 수 있습니다. (fallback function에는 gas를 보낼 수 없습니다. 대신 EVM에서 2,300의 gas를 제공해주고 이를 초과할 시 Error를 반환합니다.)

```
contract MyFallback {
  function () payable {
    // fallback function은 이런식으로 작성합니다.
  }
}
```

delegate call은 특정 컨트랙트가 다른 컨트랙트의 메소드를 호출하는데 도움을 주는 저수준 함수입니다. Proxy Contract가 이 delegate call 기능으로 Logic Contract의 메소드를 대신 호출해주게 됩니다. 이 때, 중요한 것은 delegate call을 사용한 컨트랙트가 컨트랙트의 컨텍스트(Context)를 유지한다는 특징이 있습니다. 아래 코드를 통해 이해를 돕겠습니다.

```
contract StandardToken is ERC20 {
  using SafeMath for uint256;
  mapping(address => uint256) balances;
  uint256 totalSupply_;
  mapping (address => mapping (address => uint256)) internal allowed;

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

  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to == address(0));
    require(_value <= balances[msg.sender]);
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value);
    return true;
  }

  function balanceOf(address _owner) public view returns (uint256 balance) {
    return balances[_owner];
  }
```

```
}

contract MyToken is StandardToken {
  string public name;
  string public symbol;
  uint8 public decimals;

  constructor(string _name, string _symbol, uint8 _decimals, uint256 _initial_supply) public {
    name = _name;
    symbol = _symbol;
    decimals = _decimals;

    totalSupply_ = _initial_supply;
    balances[msg.sender] = _initial_supply;
  }
}

contract UpgradableToken is MyToken, Ownable {
  StandardToken public functionBase;

  constructor()
    MyToken("Upgradable Token", "UGT", 18, 10e28) public
  {
    functionBase = new StandardToken();
  }

  function setFunctionBase(address _base) onlyOwner public {
    require(_base != address(0) && functionBase != _base);
    functionBase = StandardToken(_base);
  }

  function transfer(address _to, uint256 _value) public returns (bool) {
    require(address(functionBase).delegatecall(0xa9059cbb, _to, _value));
    return true;
  }
}
```

위 코드의 컨트랙트는 ERC20 기반의 ‘StandardToken’, StandardToken의 Data를 분리/관리 해주는 ‘MyToken’, 그리고 MyToken을 업그레이드 가능하게 만들어주는 ‘UpgradableToken’ 으로 구성됩니다.

이 때, UpgradableToken의 transfer에서 delegate call로 StandardToken의 transfer를 호출하고 있습니다. 위에서 delegate call은 컨트랙트를 호출한 쪽에서 컨텍스트를 유지한다고 했습니다. 즉, StandardToken의 transfer를 호출한 쪽인 UpgradableToken에서 배포/변경될 데이터와 결과 값을 갖고 있고, StandardToken은 자신의 transfer 기능을 제공하지만 실제로 StandardToken의 Data에는 아무런 영향이 가지 않습니다.

이러한 구성과 특징으로 StandardToken 컨트랙트에 문제가 발생했을 때, 문제를 해결한 StandardToken을 재배포하기만 하면 되며, UpgradableToken, MyToken은 수정없이 그대로 사용할 수 있게 됩니다.

김한결 작성

- ERC-20으로 배포하는 이유

  ERC-20은 이더리움 블록체인 네트워크에서 정한 표준 사양입니다.

ERC-20토큰은 이더리움과 교환이 가능하며 이더리움 지갑으로 전송이 가능합니다.이더리움과의 호환성을 위해 모든 요구사항을 충족시키는 표준이 ERC-20입니다.

이더리움은 자신의 생태계를 활용하여 다른 탈중앙화된 애플리케이션이 작동할 수 있게 만들어진 플랫폼 네트워크로, 가장 큰 특징은 Dapp이 있고 스마트 컨트랙트를 이용하고 쉽고 빠르게 토큰 발행이 가능하고,개발자들이 개발하는 다른 Dapp또한 각각 자신들만의 고유한 토큰을 가지고 있으며 이더리움과의 호환성 덕분에 이더리움을 개발한 다른 Dapp에서도 활용이 가능합니다

결론: ERC-20으로 배포해야 이더리움 생태계를 활용 가능하고, 쉬운 소통이 가능합니다.

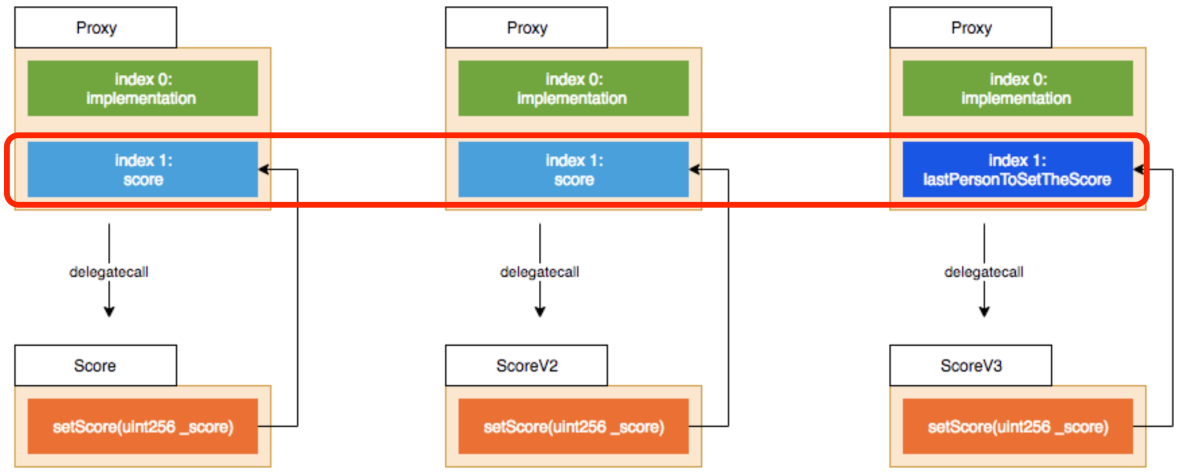
도형우 작성

질문해도 될까요????

1. assembly{}안에 있는 코드가 의미하는바를 잘 모르겠습니다.

```
// AddressSlot형태로 저장된 값들을 struct형태로 반환한다.
// assembly{}안에 있는 코드가 의미하는바를 잘 모르겠습니다.
    function getBooleanSlot(bytes32 slot) internal pure returns (BooleanSlot storage r) {
        assembly {
            r.slot := slot
        }
    }
```

2. StorageSlot에 index 값은 어떤 기준으로 값이 부여하는지 궁금합니다.



3. Beacon은 다중 프록시를 관리하기 위해서 사용되는 Contract인데 다중 프록시가 발생되는 시나리오에 대해서 알고 싶습니다.