day72-sol-errorhandle

태그	
날짜	@2023년 1월 10일

0.8.7 이후에

aseert는 오직 내부적 에러체크 용도로 사용하라고 권고

assert에러를 발생시키면 Panic(uint256) 나누기 a/b = 0

에러타입의 에러를 발생시키면 try/catch문 으로 잡을 수 있다.

try/catch 특징

1. try/catch문 안에서, aseert/revert/require를 통해 에러가 난다면 catch는 에러를 잡지를 못하고, 개발자가 의도한지 알고 정상적으로 프로그램이 끝낸다.

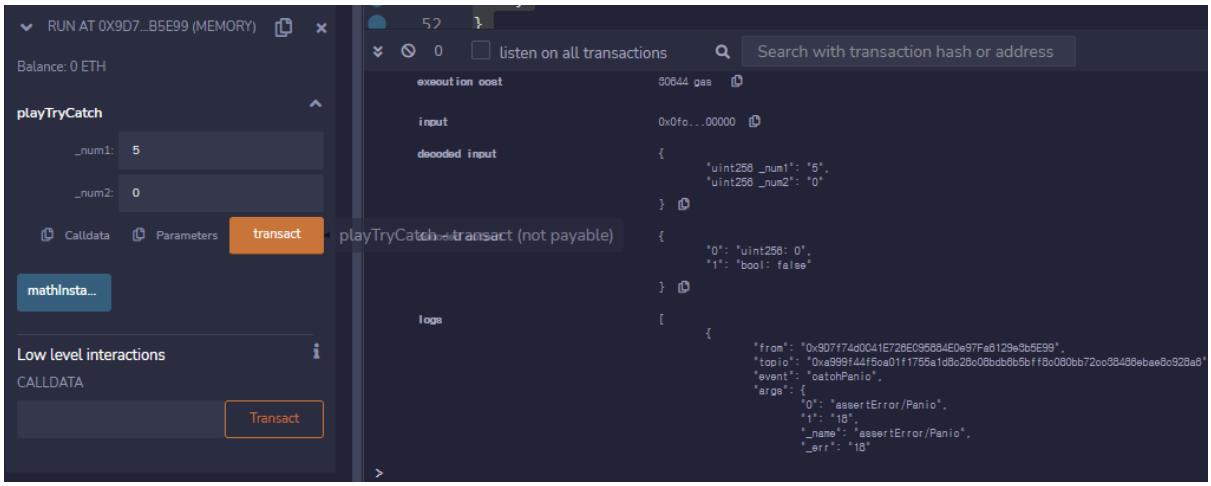
0.6 버전 이후

try/catch 왜 써야하는가?
기존의 에러 핸들러 assert/revert/require는 에러를 발생시키고 프로그램을 끝냄.
그러나, try/catch로 에러가 났어도, 프로그램을 종료시키지 않고 어떠한 대처를 하게 만들수 있다.

3. 어디서 쓰는가?
외부 스마트 컨트랙을 함수를 부를때 : 다른 스마트 컨트랙을 인스턴스화 하여서, try/catch문이 있는 스마트 컨트랙의 함수를 불러와서 사용.
외부 스마트 컨트랙을 생성 할 때 : 다른 스마트컨트랙을 인스턴스화 생성 할 때 씀
스마트컨트랙 내에서 함수를 부를때: this를 통해 try/catch를 씀

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.0  <0.9.0;

// try/catch 특징

// 1. try/catch문 안에서, aseert/revert/require를 통해 에러가 난다면 catch는 에러를 잡지를 못하고, 개발자가 의도한지 알고 정상적으로 프로그램이 끝낸다.

// 0.6 버전 이후

// try/catch 왜 써야하는가?
// 기존의 에러 핸들러 assert/revert/require는 에러를 발생시키고 프로그램을 끝냄.
// 그러나, try/catch로 에러가 났어도, 프로그램을 종료시키지 않고 어떠한 대처를 하게 만들수 있다.

// 3. 어디서 쓰는가?
// 외부 스마트 컨트랙을 함수를 부를때 : 다른 스마트 컨트랙을 인스턴스화 하여서, try/catch문이 있는 스마트 컨트랙의 함수를 불러와서 사용.
// 외부 스마트 컨트랙을 생성 할 때 : 다른 스마트컨트랙을 인스턴스화 생성 할 때 씀
// 스마트컨트랙 내에서 함수를 부를때: this를 통해  try/catch를 씀

contract math{
    //나누는 함수
    function division(uint256 _num1, uint256 _num2) public pure returns (uint256){
        //10보다 작은수만 통과
        require(_num1 < 10,"num1 should not be more than 10");
        return _num1/_num2;
    }
}

contract run{
    //이벤트
    event catchErr(string _name);
    event catchPanic(string _name, uint256 _err);
    event catchLowLevelErr(string _name, bytes _err);

    //인스턴스
    math public mathInstance = new math();
```

```
    function playTryCatch(uint256 _num1, uint256 _num2) public returns(uint256, bool){

        try mathInstance.division(_num1, _num2) returns(uint256 value){
            return(value,true);
        }catch Error(string memory _err){
            emit catchErr("revert/require");
            return(0, false);
        }catch Panic(uint256 _errorCode){
            emit catchPanic("assertError/Panic", _errorCode);
            return(0, false);
        }catch(bytes memory _errorCode){
            emit catchLowLevelErr("LowlevelError", _errorCode);
            return(0,false);
        }
    }
}
```



try / catch

외부 스마트 컨트렉트를 생성할 때

내부 스마트 컨트렉트의 함수를 부를 때

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.0  <0.9.0;

contract character{
    string private name;
    uint256 private power;

    constructor(string memory _name, uint256 _power){
        name = _name;
        power = _power;
    }
}

contract main{

    event catchOnly(string _name, string _err);

    function playTryCatch(string memory _name, uint256 _power) public returns(bool successFail){

        try new character(_name,_power){
            return (true);
        }
        catch{
            emit catchOnly("catch","Error!!");
            return (false);
        }
    }
}
```

```
contract main2{
    event catchOnly(string _name, string _err);

    function simple() public pure returns(uint256){
        return 5;
    }

    function playTryCatch() public returns(uint256, bool){
        try this.simple() returns(uint256 _value){
            return(_value, true);
        }
        catch{
            emit catchOnly("catch", "Error");
            return(0,false);
        }
    }
}
```

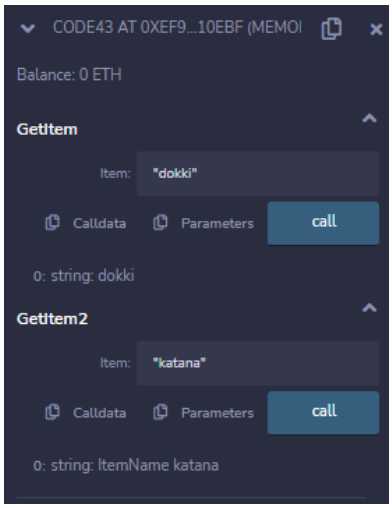
리턴 변수 명시

함수 returns(uint256 value)

함수 returns(uint256)

return value;

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.0  <0.9.0;

contract code43{

    function GetItem(string memory Item) public pure returns (string memory){
        return Item;
    }

    function GetItem2(string memory Item) public pure returns (string memory ItemName){
        string memory ItemName;
        ItemName = Item;
        return ItemName;
    }
}
```



모디파이어

담배 require(_age > 19, “you are under 18 years old”);

술

require(_age>19,”you are under 18 years old”);

만약 함수가 여러개면 매번 require 체크하기 너무 힘들다.

modifier Adults{

revert(”you are not allowed to pay for the cigarette”);

_;

}

SPDX
SPDX 라이센스는 0.68 이후 부터, 솔리디티 프로그램 맨위에 명시를 요구 하고 있습니다.솔리디티 문서에 의하면, 저희는 라이센서를 명시해줌으로써 스마트컨트랙에 대한 신뢰감을 높일수도 있고, 스마트 컨트랙 소스코드가 워낙 오픈되어 있으니, 저작권과 같은 관련된 문제를 해소하기위해 명시를 한다고 설명이 되어 있습니다.

주석
주석이라는것은 프로그램의 함수, 변수 등에 설명을 덧 붙여주는 것입니다.물론 코드를 짤때, 아무러 부연 설명없이 한번에 누구나 이해하기 쉬운게 최고이겠으나, 그러기 쉽지 않기에, 저희는 주석을 사용합니다.
주석에는 두가지 사용방법이 있습니다.

1. 블록단위
/*
주석 내용
*/
2. 행단위
// 주석 내용

Payable

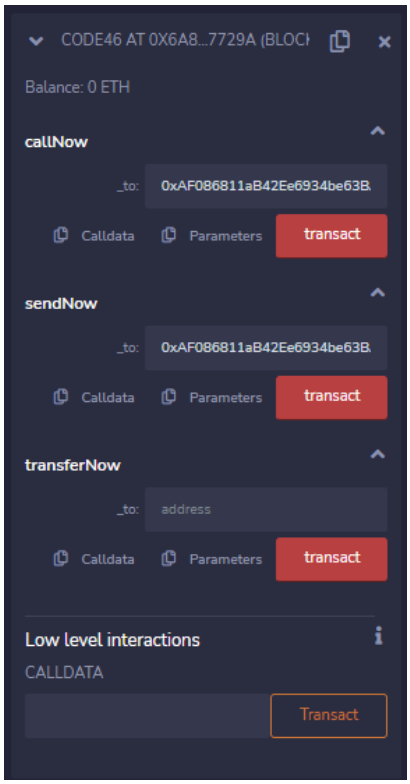
Payable은 코인과 상호작용(송금)시 필요한 키워드라고 생각하시면 간단합니다.
즉, send, trnafer, call을 이용하여, 이더를 보낼때 Payable이라는 키워드가 필요 합니다.
이 Payable은 주로 함수,주소,생성자에 붙여서 사용된답니다.
payable은 send, trnafer, call 예제에서 같이 보도록 하겠습니다.

msg.value
msg.value는 송금보낸 코인의 값 입니다
.msg.value 역시 payable 과 함께 send, transfer, call 예제에서 같이 보도록 하겠습니다.

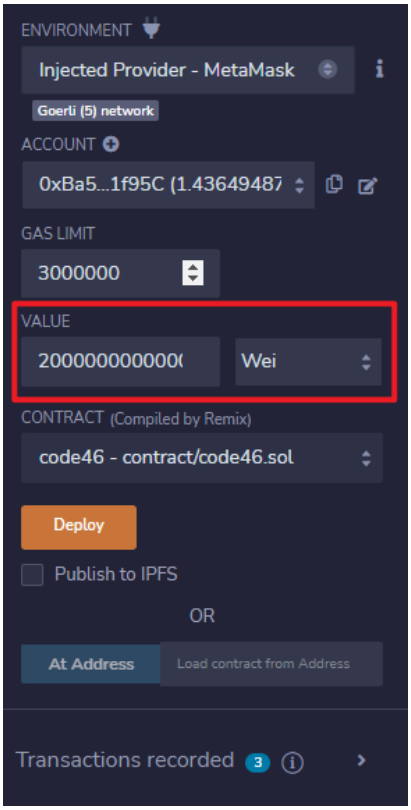
이더를 보내는 3가지 방법
이더를 보내는 3가지 방법은 send, transfer, call 이 있습니다.
이 세가지는 각각의 특징이 있습니다.
1.send : 2300 gas를 소비, 성공여부를 true 또는 false로 리턴한다

2.transfer : 2300 gas를 소비, 실패시 에러를 발생
3.call : 가변적인 gas 소비 (gas값 지정 가능), 성공여부를 true 또는 false로 리턴
재진입(reentrancy) 공격 위험성 있음, 2019년 12월 이후 call 사용을 추천.

```
└// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.0  <0.9.0;

contract code46{
    event howMuch(uint256 _value);

    function sendNow(address payable _to) public payable{
        bool sent = _to.send(msg.value); // return true or false
        require(sent,"Failed to send ether");
        emit howMuch(msg.value);
    }

    function transferNow(address payable _to) public payable{
        _to.transfer(msg.value);
        emit howMuch(msg.value);
    }

    function callNow(address payable _to) public payable{
        //~0.7vesion
        // (bool sent, ) = _to.call.gas(1000).value(msg.value)("");
        // require(sent,"Failed to send ether");


        (bool sent, ) = _to.call{value : msg.value, gas:10000}("");
        require(sent,"Failed to send ether");
        emit howMuch(msg.value);
    }
}
```



_to: 받는 주소



※ 주의 wei라서 가치가 달라서 유의해서 넣을것

10의 18 = 1

---

balance

어떤 주소의 이더의 잔액을 볼 수 있다.

주소.balance

msg.sender

msg.sender는 스마트컨트렉트랑 상호작용하는 주체라고 볼 수 있다.

msg.sender는 스마트컨트렉트의 현재 주소를 가지고 있는 주체

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.0  <0.9.0;

// msg.sender L 보내는 사람 주소
// msg.balance : 계정잔고
// to : 받을 사람 주소
// to.balance : 받는사람 잔액


contract MobileBanking{
    event SendInfo(address _msgSender, uint256 _currentValue);
    event MyCurrentValue(address _msgSender, uint256 _value)
    event CurrentValueOfSomeone(address _msgSender, address _to, uint256 _value);

    function sendEther(address payable _to) public payable{
        require(msg.sender.balance >= msg.value ,"Your balance is not enough");
        _to.transfer(msg.value);
```

```
        emit SendInfo(msg.sender,(msg.sender).balance);
    }

    function  checkValueNow() public{
        emit MyCurrentValue(msg.sender, msg.sender.balance);
    }

    function checkUserMoney(address _to) public{
        emit CurrentValueOfSomeone(msg.sender, _to, _to.balance);
    }
}
```