solidity-기본-6

태그	
날짜	@2023년 6월 21일

try/catch

```
기존의 에러 핸들러 assert/revert/require는 에러는 발생시키고 프로그램을 끝낸다.
그러나, try/catch로 에러가 났어도 프로그램을 종료시키지 않고 어떠한 대처를 하게 만들 수 있다.

try/catch 특징
1. try/catch문 안에서 assert/revert/require을 통해 에러가 난다면 catch는 에러를 잡지 못하고,
   개발자가 의도한지 알고 정상적으로 프로그램을 끝낸다.

try/catch문 밖에서 assert/revert/require을 통해 에러가 난다면 catch는 에러를 잡고,
에러를 핸들할 수 있다.

2. 3가지 catch
catch Error(string memory reason) {...} : revert나 require을 통해 생성된 에러용도
catch panic(uint errorCode) {...} : assert 를 통해 생성된 에러가 날 때 이 catch에 잡힌다.

3. 어디서 쓰는가>
- 외부 스마트 컨트랙트 함수를 부를 때 : 다른 스마트 컨트랙트를 인스턴스 화 하여, try/catch문이
                                        있는 스마트 컨트랙의 함수를 불러와서 사용
- 외부 스마트컨트랙트를 생성할 때 : 다른 스마트컨트랙트를 인스턴스화 생성할 때 씀
- 스마트 컨트랙트 내에서 함수를 부를때 : this를 통해 try/catch를 씀
```

```
외부 스마트 컨트랙트 함수를 부를 때
contract math{
    function division(uint256 _num1, uint256 _num2) public pure returns (uint256){
        require(_num1<10, "num1 should not be more than 10");
        return _num1/_num2;
    }
}

contract runner{
    event catchErr(string _name, string _err);
    event catchPanic(string _name, uint256 _err);
    event catchLowLevelErr(string _name, bytes _err);
    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", _err);
              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);
          }
      }
}
```

```
외부 스마트 컨트랙트를 생성할 때

contract character{
    string private name;
    uint256 private power;

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

contract runner{
    event catchOnly(string _name, string _err);

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

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

```
스마트컨트랙트 내에서 함수를 부를 때
contract character{
```

```
    string private name;
    uint256 private power;

    constructor(string memory _name, uint256 _power){
      name = _name;
      power = _power;
    }
}
contract  runner2{

    event catchOnly(string _name, string _err);

    function simple() public returns (uint256) {
        return 4;
    }

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