Handle Errors With Solidity Require, Assert and Revert Functions

Reading time
3 min

Published
Dec 8, 2017

Updated
Oct 3, 2019

Solidity has functions designed for reverting state changes to prevent possible issues. This tutorial reviews Solidity require, assert, and revert functions. We also determine the situations that generate exceptions of assert or require types.

For instance, Solidity assert-type of exception generates when you access an array at an index that is negative or too big. Another example is that the exception of the require type triggers when a contract with new keyword does not end successfully.

Solidity require: Main Tips

- Solidity **deals with errors** by reverting modifications to state with exceptions.
- Solidity `assert` and `require` are **convenience** functions that check invariants, conditions, and throw exceptions.
- `revert` triggers exceptions and can have an **error string**.

Error Handling Explained

Solidity manages issues by using **state-reverting** exceptions. These exceptions revert all modifications done to the state in the current call, including all sub-calls. Additionally, it shows an error to the caller.

However, exceptions bubble up when Solidity triggers exceptions in a sub-call. This rule does not include `send`, `call`, `delegatecall` and `staticcall`. Instead of bubbling up, they return `false`.

> **Remember:** Solidity **does not allow** catching exceptions.

assert and require

Solidity `assert` and `require` are convenience functions that check for conditions. In cases when conditions are not met, they throw exceptions.

> **Note:** you should apply **assert** only for internal errors or for analyzing invariants.

If code is well-written, it will not result in a failing `assert` statement. Such issues signal bugs in contracts. You can find tools, helping you test contracts and find potential problems.

These are the cases when Solidity creates **assert-type** of exceptions:

- When you invoke Solidity `assert` with an argument, showing `false`.
- When you invoke a zero-initialized variable of an internal function type.
- When you convert a large or negative value to `enum`.
- When you **divide** or **modulo** by zero.
- When you access an array in an index that is **too big** or **negative**.

The `require` Solidity function **guarantees** validity of conditions that cannot be detected before execution. It checks **inputs**, **contract state variables** and **return values** from calls to external contracts.

In the following cases, Solidity **triggers** a require-type of exception:

- When you call `require` with arguments that **result** in `false`.

- When a function called through a **message** does not end properly.
- When you create a contract with `new` keyword and the process **does not end** properly.
- When you target a codeless contract with an **external function**.
- When your contract gets Ether through a **public getter** function.
- When `.transfer()` ends in **failure**.

The example below reveals the way to make Solidity `require` look through **input conditions** and make `assert` check for **internal errors**.

Example

Copy

```
pragma solidity ⩾0.5.0 <0.7.0;

contract Sharer {
    function sendHalf(address payable addr) public payable returns
(uint balance) {
        require(msg.value % 2 == 0, "Even value required.");
        uint balanceBeforeTransfer = address(this).balance;
        addr.transfer(msg.value / 2);
        // Since transfer throws an exception on failure and
        // cannot call back here, there should be no way for us to
        // still have half of the money.
        assert(address(this).balance == balanceBeforeTransfer -
msg.value / 2);
        return address(this).balance;
    }
}
```

> **Note:** Solidity assert exceptions use **all of the gas** provided to the call. Since Metropolis (the third stage of Ethereum launch process), **require** does not use gas.

revert

You can also use Solidity `revert` function to generate **exceptions** to display an error and **redo** the current call. The function accepts a **string message**, consisting of information about the issue delivered to the caller.

The example below illustrates the way Solidity `revert` is used **together** with an **error string** and the Solidity `require` function:

Example

Copy

```
pragma solidity ^0.5.0 <0.7.0;

contract VendingMachine {
    function buy(uint amount) public payable {
        if (amount > msg.value / 2 ether)
            revert("Not enough Ether provided.");
        // Alternative way to do it:
        require(
            amount <= msg.value / 2 ether,
            "Not enough Ether provided."
        );
        // Perform the purchase.
    }
}
```

The specified string is **ABI-encoded**. In the example above, `revert` delivers the following content as error:

Example

Copy

```
0x08c379a0
// Function selector for Error(string)
0x0000000000000000000000000000000000000000000000000000000000000020
// Data offset
0x000000000000000000000000000000000000000000000000000000000000001a
// String length
0x4e6f7420656e6f756768204574686572207072f76696465642e000000000000
// String data
```

Solidity require: Summary

- To handle errors, Solidity **undoes changes** that might have caused issues.
- `assert` checks for **internal** errors. `require` analyzes conditions.
- Exceptions that Solidity `revert` generates can contain **error strings**.

← Previous Topic