Introduction to blockchain and Ethereum smart contracts

Building a fully-transparent donation smart contract

Written by Peter, Ho Man Fai

Microsoft
The Hong Kong Polytechnic University 香港理工大學
Department of Computing 電子計算學系

Learning outcomes

After you complete this lab, you will be able to:

1. Retrieve block and transaction information by using global objects.
2. Implement a payable function.
3. Implement a fallback function.
4. Understand the cryptocurrency units in Solidity programming.
5. Make use of function modifiers to alter a function's behavior.
6. Make use of events to push notifications to different network nodes.
7. Write code to destroy a published smart contract.
8. Write code to transfer Ethers from a smart contract to a personal account.

Global objects

These special variables always exist in the global namespace and are mainly used to provide information about the blockchain.

block.number	uint	Latest block number
block.timestamp	uint	Latest block timestamp as seconds since unix epoch
block.coinbase	address	Latest block miner's address
msg.data	bytes	The low-level data associated with the message call
msg.sender	address	The address of the message call sender
msg.value	uint	Number of wei sent with the message call
tx.origin	address	The address of the original transaction sender
now	uint	Alias for block.timestamp
this	address	Address of the current smart contract
this.balance	uint	The current Ether balance of the smart contract

Important note:

In smart contract programming, it is possible to have one contract calling another contract, and **the values of msg.sender and tx.origin are different when it involves two or more contracts calling each other**.

To illustrate the differences, let's look at an example. Assuming we have a user calling contract A (C_A) using his personal account (P). Then, C_A calls another contract, B (C_B). Eventually, C_B calls another contract, C_C. In summary, the call relationship is P → C_A → C_B → C_C. During the runtime, the value of msg.sender in C_C is the address of C_B (i.e. the address who makes the call), and the value of tx.origin in C_C is the address of P (the address of the function call initiator).

We are still learning how to program a single contract without connecting with other contracts. Thus, you may use msg.sender when you want to find out who sends the transaction.

Payable function

Besides passing values to a function, you can also send some amount of Ether when you call a function. The usage of sending Ether to a function really depends on your smart contract's usage. For example, you might want to implement a function to be paid by the caller before taking any actions, similar to a vending machine which will not give you a drink before you insert coins.

By default, a function is not payable. That is, you cannot send Ether to the function and it will reject any Ethers received. To make your function payable, you declare the function with the "payable" keyword.

```
pragma solidity ^0.4.11;

contract MemberRegistrationSmartContract {
    function checkDetails() {
        // Non-payable function, cannot receive Ethers
    }

    function register() payable {
        // Payable function, able to receive Ethers when called
        msg.value; // Get how many wei received from the caller
    }
}
```

Ether units

There is global object "msg.value" that returns the number of wei sent with the message. However, what is wei?

In Solidity, every Ether amount received will be converted to another unit called wei. Wei is the base and minimum unit of Ether coins (just like we have cents in dollars). The following shows the relationship and conversion values between Ether and wei. In short, 1 ether equals to 1 x 10^{18} wei.

Unit	Wei Value	Wei
wei	1 wei	1
babbage	1e3 wei	1,000
lovelace	1e6 wei	1,000,000
shannon	1e9 wei	1,000,000,000
szabo	1e12 wei	1,000,000,000,000
finney	1e15 wei	1,000,000,000,000,000
ether	1e18 wei	1,000,000,000,000,000,000

Therefore, if the msg.value is 15,000,000,000,000, and you want to know how many Ethers received, you have to divide that number by 10^{18}.

Ether unit literals

To make programming easier, you can use the following literal numbers to mean the amount of Ether in your smart contract.

```
12 wei
53 finney
20 szabo
1 ether

// Equivalent to myFunc(1000000000000000)
myFunc(1 finney)

// True
2 ether == 2000 finney

// Evaluates to true if msg.value is equal to
// 5000000000000000000
msg.value == 5 ether
```

Function modifier

Modifiers can be used to change the behavior of functions. You can use function modifiers to check whether a condition is met or do some actions before a function gets executed.

```
pragma solidity ^0.4.0;

contract TestingSmartContract {
    bool isEnabled = false;

    modifier mustEnabled {  // No brackets if no parameters
        if( ! isEnabled) {
            throw;
        }

        // The _ operator means to execute the function
        // code in which the modifier is attached to.
        _;
    }
```

```
    modifier mustHaveEther(uint amount) {
        if(msg.value < amount) {
            throw;
        }
        _;
    }

    function myFuncA() isEnabled {
        // Some code......
    }

    // Two modifiers, the isEnabled modifier gets executed first,
    // and when it runs the _ operator, it jumps to the
    // mustHaveEther modifier. Then, when the mustHaveEther
    // modifier runs the _ operator, it jumps back to the myFuncB
    // code and executes. If one of the modifier does not execute
    // the _ operator, the code inside myFuncB will not get
    // executed at all.
    function myFuncB() payable isEnabled mustHaveEther(1 ether) {
        // Some code
    }
}
```

Event

Imagine you have a several lines of code that change a critical value in a smart contract, and you want to inform everyone on the blockchain network when it changes. Besides sending emails or sending WhatsApp messages, how can you do that? That answer is: event.

Events allow programmers to push notification to all Ethereum network nodes in a smart contract.

Firstly, you define the event signature with a name and possible parameters, similar to how you define a function. Secondly, you call the event by passing values to those parameters.

```
pragma solidity ^0.4.0;

contract ClientReceipt {
    event Deposit(string message, bytes32 depositID, uint amount);

    function deposit(bytes32 _id) payable {
        // Some code here......

        // Publish a new event to every network node
        Deposit("Someone has just deposited!", _id, msg.value);
    }
}
```

Important note:
Event is very useful if you want to return values from a non-constant function, which cannot use the return keyword to send back the values.

Destroy a published smart contract

Until now, you can only create contracts but not removing them. What if you think you have too many contracts and you just want to delete some of them?

There is a special global function in Solidity called selfdestruct() that allows you to disable a smart contract in the blockchain network.

The selfdestruct() function receives one address parameter. The address parameter is used for Ether refund (i.e. all the Ethers in the smart contract will be transfer to that address).

There is not condition checking mechanism built inside the selfdestruct() function, meaning everyone is eligible to call this function if you programmed it in your smart contract. Therefore, to make sure only the contract maker is given a permission to disable the contract, we have to save the address of the contract maker, and then check if the function caller is the contract maker before we disable the contract.

```
pragma solidity ^0.4.11;

contract TestingSmartContract {
    address contractMaker;

    // Save the contract maker's address in the constructor
    function TestingSmartContract() payable {
        contractMaker = msg.sender;  // Use the global object
    }


    function destroyContract() {
        // Make sure only the contract maker can disable the
        // contract
        if(msg.sender != contractMaker) {
            throw;
        }
        selfdestruct(contractMaker);  // Pass address for refund
    }
}
```

Important note:

Even though a contract gets destroyed, all the records about the contract remains inside the blockchain and nothing is deleted. The selfdestruct() function just DISABLE (technically, not delete or remove) the contract. Thus, the immutability of blockchain still holds.

Transfer Ether from a smart contract to an account

We have mentioned that we can implement payable functions to receive Ethers from users' personal account, and it is actually possible to do the reverse.

To send back Ethers from smart contract to an account, you have to know the account's address and call the transfer() function.

```
address receiverA = 0x887795EC5A6C0c3403abD788910a3840562eE441;
address receiverB = msg.sender;

// Not to be confused: receiverA is the Ether receiver,
// not the one who transfer Ether to the smart contract
receiverA.transfer(5 ether);
receiverB.transfer(200 finnney);
```

Example: a donation smart contract

The following Solidity program is a donation campaign smart contract. It provides functions for different donors to donate Ether, and allows the contract creator to claim back all donations.

The highlighted code are the contents that you should learn in this tutorial.

```
pragma solidity ^0.4.11;

contract DonationCampaignContract {
    address contractCreator;
    mapping(address => uint) donorList;

    event NewDonation(address donor, uint amount);

    modifier minimumDonation(uint threshold) {
        if(msg.value < threshold) {
            throw;
        }
        _;
    }

    function DonationCampaignContract() {
        contractCreator = msg.sender;
    }

    function donate() payable minimumDonation(10 ether) {
        // Check if this guy has ever donated
        uint currentDonation = 0;
        if(donorList[msg.sender] != 0) {
            currentDonation = donorList[msg.sender];
        }

        // Save the donation record into a mapping table
        donorList[msg.sender] = currentDonation + msg.value;

        // Fire event
        NewDonation(msg.sender, msg.value);
    }
```

Event definition

Function modifier

Record who creates this contract when the smart contract is constructed

You have to use payable function to receive Ether

Any donation that is less than 10 ethers will be rejected

Push event notification to other network nodes with 2 values

```
    function claimDonation() {
        uint currentBalance = this.balance;
        contractCreator.transfer(currentBalance);
    }

    function destroyContract() {
        if(msg.sender != contractCreator) {
            throw;
        }
        selfdestruct(contractCreator);
    }
}
```

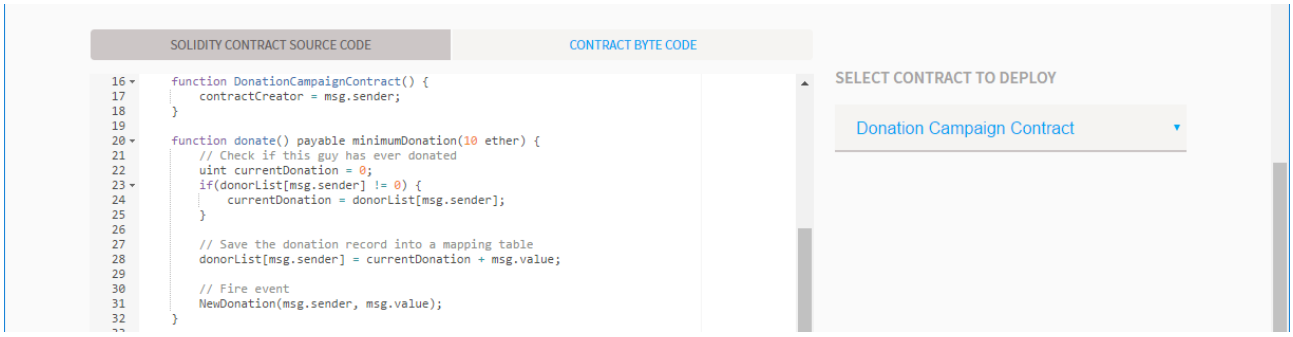
The current Ether balance of the smart contract

Transfer an amount of Ether from the smart contract to the contract creator

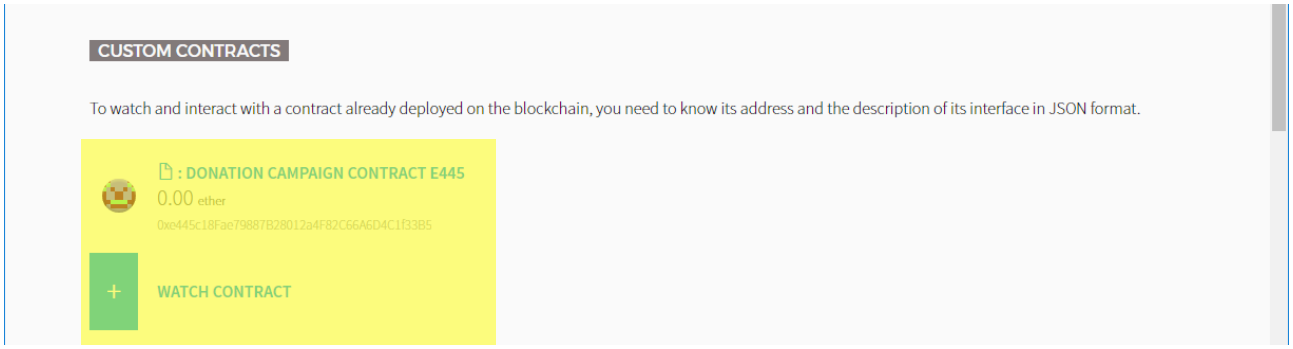
Disable the smart contract and refund all Ethers to the contract creator

Testing the contract

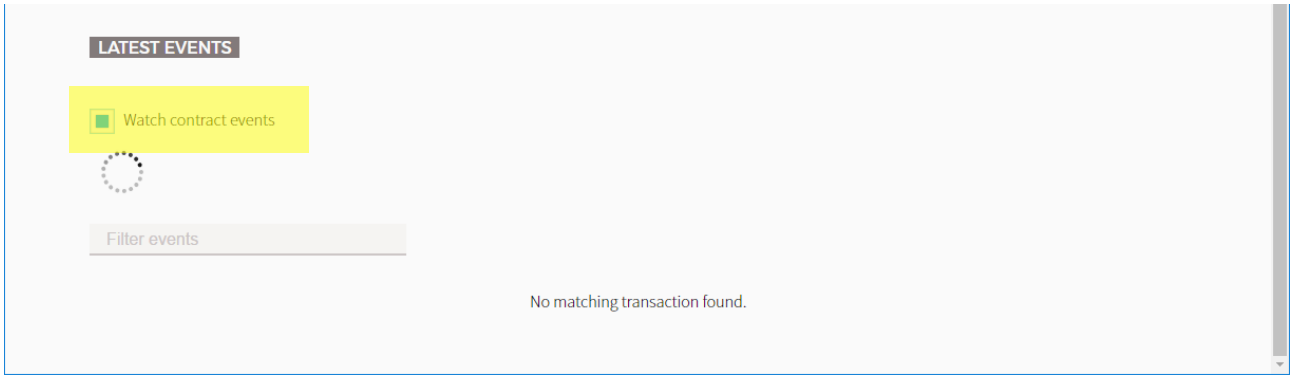
1. Deploy the smart contract above to your blockchain network.



2. Add the deployed smart contract to your second computer's Ethereum Wallet watch contract list.



3. Inside the smart contract page → Check the "Watch contract events" box.



4. Try to donate 50 Ethers by making a transaction → Wait for the next block being mined → You should see an event in the "Latest events" section.

5. Try to donate 5 Ethers → When you click the execute button, you shall see a red line that states your transaction will fail. It happens because our contract has programmed to reject donation that is less than 10 Ethers.

6. Switch to your second computer → try to destroy the smart contract by sending a new transaction → You will get another failure message. It is because we have programmed to just allow the contract creator (which is the account in your first computer) to delete the contract.

7. Now, switch back to your first computer → try to destroy the smart contract → You should see no failure messages.

Exercises

Modify the donation smart contract given in the example section based on the following requirements.

1. Create a constant function called “checkContractBalance” that returns the current balance (in Wei unit) of the smart contract.
2. Create a constant function called “checkSelfDonationAmount” that returns the amount of Wei that the caller has donated.
3. Create a new event called “ClaimDonation”. Whenever the contract creator claims the donation, this new event should be sent. You should include the claimed amount of Wei (an unsigned integer) in the event.
4. Create a new modifier called “onlyCreator”. It checks if the transaction caller (i.e. msg.sender) is equal to the contract creator. If yes, then proceed (i.e. call the _ operator), otherwise, throw an error.
5. Delete the condition check in destoryContract() function, then apply the onlyCreator modifier.
6. Create another modifier called “onlyHasEther”. It checks if the current smart contract Ether balance is greater than 0. If yes, then proceed, otherwise, throw an error.
7. Apply both “onlyCreator” and “onlyHasEther” modifiers to the claimDonation() function.
8. Change the minimum donation amount from 10 ethers to 0.5 ethers.

References

1. Solidity Official Documentation Website
   https://solidity.readthedocs.io/en/develop/
2. Learn Solidity in Y Minutes
   https://learnxinyminutes.com/docs/solidity/