Технология блокчейн
#7

Introduction to Smart Contracts - Введение в смарт-контракты

Обзор

- Что такое смарт контракт?
- Введение в Solidity

Blockchain
Center

Что такое смарт контракт?

Смарт-контракт - это самоисполняющийся контракт, в котором условия соглашения между покупателем и продавцом записаны в строках кода и хранятся в сети блокчейн.

SMART CONTRACT

PARTIES SMART CONTRACT EXECUTION

Blockchain Center

Традиционный договор - это юридическое соглашение между сторонами, определяющее условия сделки или отношений. В нем излагаются права и обязанности каждой стороны и могут содержаться положения, определяющие, что должно произойти, если одна из сторон не выполнит свои обязательства.

Смарт-контракты похожи на традиционные контракты в плане изложения условий соглашения, но они написаны в коде и хранятся в блокчейне. Это облегчает их применение и исполнение, поскольку условия автоматически выполняются кодом.

Смарт-контракты автоматизируют процессы и снижают необходимость в промежуточной и ручной обработке. Они часто используются для облегчения, проверки и обеспечения соблюдения переговоров или исполнения контракта.

Blockchain Center

Скорость, эффективность
и точность

Безопасность

Доверие и прозрачность

Экономия времени

Blockchain
Center

Идентифицировать соглашение

Несколько сторон определяют возможности сотрудничества и желаемые результаты.

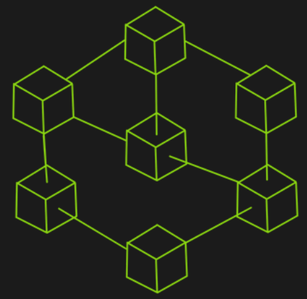
Установить условия

Смарт-контракты выполняются автоматически при выполнении о пределенных условий.

Код

Компьютерная программа написана.

Блокчейн



Шифрование обеспечивает безопасную передачу сообщений между сторонами.

Выполнение

Код выполняется, а результаты запоминаются.

Обновления сети

Все узлы сети обновляют свой ledger.

Blockchain Center

Виды смарт-контрактов

- Степень анонимности. По степени анонимности смарт-контракты бывают конфиденциальными, частично открытыми и полностью открытыми.

- Механизм инициирования. По механизму инициирования смарт-контракты бывают автоматизированные и не автоматизированные

Blockchain Center

Виды смарт-контрактов

Степень автоматизации смарт-контракта.

- Полностью автоматизированные - бумажный носитель не требуется.
- Частично автоматизированные - необходима копия смарт-контракта на бумажном носителе
- Автоматизированные преимущественно в хранилище.

Языки смарт-контрактов на разных блокчейнах

- Ethereum/EVM - Solidity, Vyper
- Aptos и Sui - Move
- Solana, Polkadot, Near - Rust

Введение в Solidity

Примитивные типы данных

```
// Значения по умолчанию
bool public defaultBoo; // false
uint public defaultUint; // 0
int public defaultInt; // 0
address public defaultAddr; //
0x0000000000000000000000
0000000000000000
```

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Primitives {
    bool public boo = true;

    /*
    uint stands for unsigned integer, meaning non negative integers
    different sizes are available
        uint8   ranges from 0 to 2 ** 8 - 1
        uint16  ranges from 0 to 2 ** 16 - 1
        ...
        uint256 ranges from 0 to 2 ** 256 - 1
    */
    uint8 public u8 = 1;
    uint public u256 = 456;
    uint public u = 123; // uint is an alias for uint256

    /*
    Negative numbers are allowed for int types.
    Like uint, different ranges are available from int8 to int256

    int256 ranges from -2 ** 255 to 2 ** 255 - 1
    int128 ranges from -2 ** 127 to 2 ** 127 - 1
    */
    int8 public i8 = -1;
    int public i256 = 456;
    int public i = -123; // int is same as int256
```

Примитивные типы данных 2

```
// minimum and maximum of int
int public minInt = type(int).min;
int public maxInt = type(int).max;

address public addr = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c;

/*
In Solidity, the data type byte represent a sequence of bytes.
Solidity presents two type of bytes types :

 - fixed-sized byte arrays
 - dynamically-sized byte arrays.

 The term bytes in Solidity represents a dynamic array of bytes.
 It’s a shorthand for byte[] .
*/
bytes1 a = 0xb5; //  [10110101]
bytes1 b = 0x56; //  [01010110]

// Default values
// Unassigned variables have a default value
bool public defaultBoo; // false
uint public defaultUint; // 0
int public defaultInt; // 0
address public defaultAddr; // 0x0000000000000000000000000000000000000000
```

Переменные

В Solidity существует 3 типа переменных:

локальные
объявлены внутри функции
не хранятся в блокчейне

состояния
объявленные вне функции
хранится в блокчейне

глобальные (предоставляет информацию о блокчейне)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Variables {
    // State variables are stored on the blockchain.
    string public text = "Hello";
    uint public num = 123;

    function doSomething() public {
        // Local variables are not saved to the blockchain.
        uint i = 456;

        // Here are some global variables
        uint timestamp = block.timestamp; // Current block timestamp
        address sender = msg.sender; // address of the caller
    }
}
```

Константы

Константы - это переменные, которые не могут быть изменены.

Их значение жестко закодировано, и использование констант позволяет экономить газ.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Constants {
    // coding convention to uppercase constant variables
    address public constant MY_ADDRESS = 0x777788889999AaAAbBbbCcccddDdeeeEfFFfCcCc;
    uint public constant MY_UINT = 123;
}
```

Неизменяемые

Неизменяемые переменные **подобны константам.** Значения неизменяемых переменных могут быть заданы в конструкторе, но **не могут быть изменены впоследствии**.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Immutable {
    // coding convention to uppercase constant variables
    address public immutable MY_ADDRESS;
    uint public immutable MY_UINT;

    constructor(uint _myUint) {
        MY_ADDRESS = msg.sender;
        MY_UINT = _myUint;
    }
}
```

Переменные состояния

Чтобы **записать** или **обновить** переменную состояния, необходимо **отправить транзакцию.**
С другой стороны, вы можете **читать** переменные **состояния бесплатно**, без платы за транзакцию.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract SimpleStorage {
    // State variable to store a number
    uint public num;

    // You need to send a transaction to write to a state variable.
    function set(uint _num) public {
        num = _num;
    }

    // You can read from a state variable without sending a transaction.
    function get() public view returns (uint) {
        return num;
    }
}
```

Ether и wei

Транзакции оплачиваются эфиром (ether).

Подобно тому, как один доллар равен 100 центам, один ether равен 10**18 wei.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract EtherUnits {
    uint public oneWei = 1 wei;
    // 1 wei is equal to 1
    bool public isOneWei = 1 wei == 1;

    uint public oneEther = 1 ether;
    // 1 ether is equal to 10^18 wei
    bool public isOneEther = 1 ether == 1e18;
}
```

if / else

```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract IfElse {
    function foo(uint x) public pure returns (uint) {
        if (x < 10) {
            return 0;
        } else if (x < 20) {
            return 1;
        } else {
            return 2;
        }
    }

    function ternary(uint _x) public pure returns (uint) {
        // if (_x < 10) {
        //     return 1;
        // }
        // return 2;

        // shorthand way to write if / else statement
        // the "?" operator is called the ternary operator
        return _x < 10 ? 1 : 2;
    }
}
```

Циклы For и While

Solidity поддерживает циклы for, while и do while.

Не пишите беспредельные циклы, так как это может привести к ограничению газа, что вызовет сбой транзакции.

По этой причине циклы while и do while используются редко.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Loop {
    function loop() public {
        // for loop
        for (uint i = 0; i < 10; i++) {
            if (i == 3) {
                // Skip to next iteration with continue
                continue;
            }
            if (i == 5) {
                // Exit loop with break
                break;
            }
        }

        // while loop
        uint j;
        while (j < 10) {
            j++;
        }
    }
}
```

Blockchain Center

Спасибо за внимание