# Solidity基础语法（上）

## 环境准备

- 添加测试网络：https://chainlist.org/chain/97
- faucet：https://testnet.bnbchain.org/faucet-smart
- 浏览器：https://testnet.bscscan.com/
- remix：https://remix.ethereum.org/#optimize=false&version=soljson-v0.8.18+commit.87f61d96.js

## 两种账户

- EOA：Externally Owned Account，与一个私钥一一对应，例如小狐狸里面的account1就是EOA
- CA：Contract Account，合约账户，没有私钥与之对应，我们部署的合约就是一个CA，它也可以持有资金。

## 第一个dapp

1. 写状态变量（上链）是一笔交易（tx），需要矿工打包，所以需要花费资金；
2. 读取状态变量，是从账本中获取数据，不是一笔交易，所以免费。（必须加上view）

```
// 指定编译器版本，版本标识符
pragma solidity ^0.8.13;

// 关键字 contract 跟java的class一样  智能合约是Inbox
contract Inbox{

    // 状态变量，存在链上
  string public message;

    // 构造函数
  constructor(string memory initMessage) {
        // 本地变量
        string memory tmp = initMessage;
        message = tmp;
  }

    // 写操作，需要支付手续费
    function setMessage(string memory _newMessage) public {
        message = _newMessage;
    }

    // 读操作，不需要支付手续费
    function getMessage() public view returns(string memory) {
        return message;
    }
```

```
}
```

## 基础数据类型

- int（有符号整型，有正有负）int默认为int256
- uint（无符号整型，无负数）uint默认为uint256
- 以8位为区间，支持int8，int16，int24 至 int256（uint同理）
- bool类型：true，false
- 定长字节：bytes1~bytes32
- 地址：address（20个字节，40个16进制字符，共160位），如：0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Primitives {
    bool public flag = 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

    // 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
}
```

# 变量variables

- **状态变量（state）**
  - 定义在合约内，函数外
  - 存储在链上
- **本地变量（local）**
  - 定义在函数内
  - 不会存储在链上
- **全局变量（global）**
  - 与当前合约无关，描述整个区块链的信息（时间、块高等）

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Variables {
    // State variables are stored on the blockchain.
    string public msg = "Hello";
    uint public age = 26;

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

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

```
}
```

描述区块链信息的全局变量，常用如下：

| 函数 | 含义 | 备注 |
| --- | --- | --- |
| blockhash(uint blockNumber) | (byte32)哈希值 | |
| block.coinbase | (address) 当前块矿工的地址 | |
| block.difficulty | (uint)当前块的难度 | |
| block.gaslimit | (uint)当前块的gaslimit | |
| block.number | (uint)当前区块的块号 | |
| block.timestamp | (uint)当前块的时间戳 | 常用 |
| gasleft() | (uint)当前还剩的gas | |
| tx.origin | (address)交易的原始发送者的地址，只能是EOA | 常用 |
| msg.sender | (address)当前调用发起人的地址（可能是合约CA，也可能是EOA） | 常用 |
| msg.sig | (bytes4)调用数据的前四个字节（函数标识符） | 常用 |
| msg.value | (uint)这个消息所附带的货币量，单位为wei | 常用 |
| msg.data | (bytes)完整的调用数据（calldata) | 常用 |
| tx.gasprice | (uint) 交易的gas价格 | |

# 常量constant

1. 常量与变量相对，需要硬编码在合约中，合约部署之后，无法改变。
2. 常量更加节约gas，一般用大写来代表常量。
3. 高阶用法：clone合约时，如果合约内有初始值，必须使用constant，否则clone的新合约初始值为空值。

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Constants {
    // coding convention to uppercase constant variables
    address public constant MY_ADDRESS = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
    uint public constant MY_UINT = 123;
}
```

# 不可变量immutable

1. 与常量类似，但是不必硬编码，可以在构造函数时传值，部署后无法改变。
2. immutable仅支持值类型（如：int，address，bytes8），不支持非值类型（如：string，bytes）

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Immutable {
    // coding convention to uppercase constant variables
    address public immutable MY_ADDRESS;
    uint public immutable MY_UINT;
    bytes1 public immutable MY_BYTES1 = 0xff;
    // string public immutable greetings = "hello";  // error

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

# ether和wei

- 常用单位为：wei，gwei，ether
- 不含任何后缀的默认单位是 wei
- 1 gwei = 10^9 wei

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

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;
}
```

# msg三人组

当用户发起一笔交易时，相当于向合约发送一个消息(msg)，这笔交易可能会涉及到三个重要的全局变量，具体如下：

1. **msg.sender**：表示这笔交易的调用者是谁（地址），同一个交易，不同的用户调用，msg.sender不同；
2. **msg.value**：表示调用这笔交易时，携带的ether数量，这些以太坊由msg.sender支付，转入到当前合约（wei单位整数）；
    1. 注意：一个函数（或地址）如果想接收ether，需要将其修饰为：**payable**。
3. **msg.data**：表示调用这笔交易的信息，由函数签名和函数参数（16进制字符串），组成代理模式时常用msg.data（后续讲解）。

# msg.sender

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract MsgSender {
    address public owner;
    uint256 public value;
    address public caller;

    constructor() {
        //在部署合约的时候，设置一个全局唯一的合约所有者，后面可以使用权限控制
        owner = msg.sender;
    }

    //1. 对与合约而言，msg.sender是一个可以改变的值，并不一定是合约的创造者
    //2. 任何人调用了合约的方法，那么这笔交易中的from就是当前合约中的msg.sender
    function setValue(uint256 input) public {
        value = input;
        caller = msg.sender;
    }
}
```

## msg.value

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract MsgValue {

    // uint256 public money;
    mapping(address=> uint256) public personToMoney;

    // 函数里面使用了msg.value，那么函数要修饰为payable
    function play() public payable {

        // 如果转账不是100wei，那么参与失败
        // 否则成功，并且添加到维护的mapping中
        require(msg.value == 100, "should equal to 100!");
        personToMoney[msg.sender] = msg.value;
    }

    // 查询当前合约的余额
    function getBalance() public view returns(uint256) {
        return address(this).balance;
    }
}
```

## msg.data

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract MsgData {
    event Data(bytes data, bytes4 sig);

    // input0: addr: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
    // input1: amt : 1
    function transfer(address addr, uint256 amt) public {
        bytes memory data = msg.data;

        // msg.sig 表示当前方法函数签名（4字节）
        // msg.sig 等价于 this.transfer.selector
        emit Data(data, msg.sig);
    }

    //output:
    // - data:
0xa9059cbb0000000000000000000000005b38da6a701c568545dcfcb03fcb875f56beddc40000000000000000000000000000000000000000000000000000000000000001
    // - sig: 0xa9059cbb
```

```
    // 对data进行分析：
    // 0xa9059cbb //前四字节
    // 0000000000000000000000005b38da6a701c568545dcfcb03fcb875f56beddc4 //第一个参数占位符（32字节）
    // 0000000000000000000000000000000000000000000000000000000000000001 //第二个参数占位符（32字节）
}
```

# payable

1. 一个函数（或地址）如果想接收ether，需要将其修饰为：**payable**。
2. address常用方法：
    1. balance(): 查询当前地址的ether余额
    2. transfer(uint): 合约向当前地址转指定数量的ether，如果失败会回滚
    3. send(uint): 合约向当前地址转指定数量的ether，如果失败会返回false，不回滚（不建议使用send）

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Payable {
    // 1. Payable address can receive Ether
    address payable public owner;

    // 2. Payable constructor can receive Ether
    constructor() payable {
        owner = payable(msg.sender);
    }

    // 3. Function to deposit Ether into this contract.
    function deposit() public payable {}

    // 4. Call this function along with some Ether.
    // The function will throw an error since this function is not payable.
    function notPayable() public {}

    // 5. Function to withdraw all Ether from this contract.
    function withdraw() public {
        uint amount = address(this).balance;
        owner.transfer(amount);
    }

    // 6. Function to transfer Ether from this contract to address from input
    function transfer(address payable _to, uint _amount) public {
        _to.transfer(_amount);
    }
}
```

# gas相关

**gas描述执行一笔交易时需要花费多少ether！（1 ether = 10^18wei)**

交易手续费 = **gas_used * gas_price**，其中：

1. gas：是数量单位，uint
2. gas_used：表示一笔交易实际消耗的gas数量
3. gas_price：每个gas的价格，单位是wei或gwei
4. gas limit：表示你允许这一笔交易消耗的gas上限，用户自己设置（防止因为bug导致的损失）
    1. 如果gas_used小于gas_limit，剩余gas会返回给用户，这个值不再合约层面设置，在交易层面设置（如metamask）
    2. 如果gas_used大于gas_limit，交易失败，资金不退回
5. block gas limit：表示一个区块能够允许的最大gas数量，由区块链网络设置

# view和pure

view和pure用于修饰Getter函数（只读取数据的函数），其中：

1. **view**：表示函数中不会修改状态变量，只是读取；
2. **pure**：表示函数中不会使用状态变量，既不修改也不读取。

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract ViewAndPure {
    uint public x = 1;

    // Promise not to modify the state.
    function addToX(uint y) public view returns (uint) {
        return x + y;
    }

    // Promise not to modify or read from the state.
    function add(uint i, uint j) public pure returns (uint) {
        return i + j;
    }
}
```

# bytes和string

byteN、bytes、string直接的关系

**bytes:**

- bytes是动态数组，相当于byte数组（如：byte[10])
- 支持push方法添加
- 可以与string相互转换

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract  Bytes {
    bytes public name;

    //1. 获取字节长度
    function getLen() public view returns(uint256) {
        return name.length;
    }

    //2. 可以不分空间，直接进行字符串赋值，会自动分配空间
    function setValue(bytes memory input) public {
        name = input;
    }

    //3. 支持push操作，在bytes最后面追加元素
    function pushData() public {
        name.push("h");
    }
}
```

**string:**

- string 动态尺寸的UTF-8编码字符串，是特殊的可变字节数组
- string **不支持下标索引、不支持length、push方法**
- string **可以修改(需通过bytes转换)**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract  String {
    string public name = "lily";

    function setName() public {
        bytes(name)[0] = "L";
    }

    function getLength() public view returns(uint256) {
        return bytes(name).length;
    }
}
```

# struct

- 自定义结构类型，将不同的数据类型组合到一个结构中，目前支持参数传递结构体。
- 枚举和结构体都可以定义在另外一个文件中，进行import后使用

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Todos {
    struct Todo {
        string text;
        bool completed;
    }

    // An array of 'Todo' structs
    Todo[] public todos;

    // [["hello", true], ["world", false]]
    function createByElement(Todo[] memory _todos) public {
        for (uint i = 0; i < _todos.length; i++) {
            todos.push(_todos[i]);
        }
    }

    function create(string memory _text) public {
        // 3 ways to initialize a struct
        // - calling it like a function
        todos.push(Todo(_text, false));

        // key value mapping
        todos.push(Todo({text: _text, completed: false}));

        // initialize an empty struct and then update it
        Todo memory todo;
        todo.text = _text;
        // todo.completed initialized to false

        todos.push(todo);
    }

    // Solidity automatically created a getter for 'todos' so
    // you don't actually need this function.
    function get(uint _index) public view returns (string memory text, bool completed)
{
        Todo storage todo = todos[_index];
        return (todo.text, todo.completed);
    }

    // update text
    function update(uint _index, string memory _text) public {
        Todo storage todo = todos[_index];
        todo.text = _text;
    }
```

```
    // update completed
    function toggleCompleted(uint _index) public {
        Todo storage todo = todos[_index];
        todo.completed = !todo.completed;
    }
}
```

# mapping

- 定义：mapping(keyType => valueType) myMapping
- key可以是任意类型，value可以是任意类型（value也可以是mapping或者数组）
- mapping不支持迭代器
- 不需要实例化等，定义后直接可以使用

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Mapping {
    // Mapping from address to uint
    mapping(address => uint) public myMap;

    function get(address _addr) public view returns (uint) {
        // Mapping always returns a value.
        // If the value was never set, it will return the default value.
        return myMap[_addr];
    }

    function set(address _addr, uint _i) public {
        // Update the value at this address
        myMap[_addr] = _i;
    }

    function remove(address _addr) public {
        // Reset the value to the default value.
        delete myMap[_addr];
    }
}

contract NestedMapping {
    // Nested mapping (mapping from address to another mapping)
    mapping(address => mapping(uint => bool)) public nested;

    function get(address _addr1, uint _i) public view returns (bool) {
        // You can get values from a nested mapping
        // even when it is not initialized
        return nested[_addr1][_i];
    }
```

```
    function set(
        address _addr1,
        uint _i,
        bool _boo
    ) public {
        nested[_addr1][_i] = _boo;
    }

    function remove(address _addr1, uint _i) public {
        delete nested[_addr1][_i];
    }
}
```

# 修饰器modifier

修饰器用于修饰函数，在函数执行前或执行后进行调用，经常用于：

1. 权限控制
2. 参数校验
3. 防止重入攻击等

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract FunctionModifier {
    // We will use these variables to demonstrate how to use modifiers.
    address public owner;
    uint public x = 10;
    bool public locked;

    constructor() {
        // Set the transaction sender as the owner of the contract.
        owner = msg.sender;
    }

    // 1. Modifier to check that the caller is the owner of the contract.
    modifier onlyOwner() {
        require(msg.sender == owner, "Not owner");
        // Underscore is a special character only used inside
        // a function modifier and it tells Solidity to
        // execute the rest of the code.
        _;
    }

    // 2. Modifiers can take inputs. This modifier checks that the
    // address passed in is not the zero address.
    modifier validAddress(address _addr) {
        require(_addr != address(0), "Not valid address");
```

```
        _;
    }

    function changeOwner(address _newOwner) public onlyOwner validAddress(_newOwner) {
        owner = _newOwner;
    }

    // Modifiers can be called before and / or after a function.
    // This modifier prevents a function from being called while
    // it is still executing.
    modifier noReentrancy() {
        require(!locked, "No reentrancy");

        locked = true;
        _;
        locked = false;
    }

    function decrement(uint i) public noReentrancy {
        x -= i;

        if (i > 1) {
            decrement(i - 1);
        }
    }
}
```

# 事件Event

事件是区块链上的日志，每当用户发起操作的时候，可以发送相应的事件，常用于：

1. 监听用户对合约的调用
2. 便宜的存储（用合约存储更加昂贵）

通过链下程序（如：subgraph）对合约进行事件监听，可以对Event进行搜集整理，从而做好数据统计，常用方式：

1. 合约触发后发送事件
2. subgraph对合约事件进行监听，计算（如：统计用户数量）
3. 前端程序直接访问subgraph的服务，获得统计数据（这避免了在合约层面统计数据的费用，并且获取速度更快）

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Event {
    // Event declaration
    // Up to 3 parameters can be indexed.
    // Indexed parameters helps you filter the logs by the indexed parameter
    event Log(address indexed sender, string message); // 修饰为indexed
```

```
    event AnotherLog(); // 无参数的事件
    event TestAnonymous(address indexed sender, uint256 num) anonymous; // 匿名事件

    function test() public {
        emit Log(msg.sender, "Hello World!");
        emit Log(msg.sender, "Hello EVM!");
        emit AnotherLog();
    }
}
```

# 可见性visibility

合约的方法和状态变量需要使用关键字进行修饰，从而决定其是否可以被其他合约调用，修饰符包括：

- public：所有的合约和外部账户（EOA）都可以调用；
- private：只允许合约内部调用；
- internal：仅允许合约内部以及子合约中调用；
- external：仅允许外部地址(EOA或CA)调用，合约内部及子合约都不能调用；(早期版本可以使用this调用external方法)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Base {
    // Private function can only be called
    // - inside this contract
    // Contracts that inherit this contract cannot call this function.
    function privateFunc() private pure returns (string memory) {
        return "private function called";
    }

    function testPrivateFunc() public pure returns (string memory) {
        return privateFunc();
    }

    // Internal function can be called
    // - inside this contract
    // - inside contracts that inherit this contract
    function internalFunc() internal pure returns (string memory) {
        return "internal function called";
    }

    function testInternalFunc() public pure virtual returns (string memory) {
        return internalFunc();
    }

    // Public functions can be called
    // - inside this contract
    // - inside contracts that inherit this contract
```

```
    // - by other contracts and accounts
    function publicFunc() public pure returns (string memory) {
        return "public function called";
    }

    // External functions can only be called
    // - by other contracts and accounts
    function externalFunc() external pure returns (string memory) {
        return "external function called";
    }

    // This function will not compile since we're trying to call
    // an external function here.
    // function testExternalFunc() public pure returns (string memory) {
    //     return externalFunc();
    // }

    // State variables
    string private privateVar = "my private variable";
    string internal internalVar = "my internal variable";
    string public publicVar = "my public variable";
    // State variables cannot be external so this code won't compile.
    // string external externalVar = "my external variable";
}

contract Child is Base {
    // Inherited contracts do not have access to private functions
    // and state variables.
    // function testPrivateFunc() public pure returns (string memory) {
    //     return privateFunc();
    // }

    // Internal function call be called inside child contracts.
    function testInternalFunc() public pure override returns (string memory) {
        return internalFunc();
    }
}
```

# ERC20（标准Token）

1. EIP: Ethereum Improvement Propose
2. 任何遵从EIP-20协议（ERC20标准）的Contract都属于ERC20 Token

## 标准接口

```
// 6 REQUIRED FUNCTIONS
```

```
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
function transfer(address _to, uint256 _value) public returns (bool success)
function transferFrom(address _from, address _to, uint256 _value) public returns (bool
success)
function approve(address _spender, uint256 _value) public returns (bool success)
function allowance(address _owner, address _spender) public view returns (uint256
remaining)

// 2 REQUIRED EVENTS
event Transfer(address indexed _from, address indexed _to, uint256 _value)
event Approval(address indexed _owner, address indexed _spender, uint256 _value)

// 3. OPTIONAL FUNCTIONS
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
```

# 发行Token

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

// https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v3.0.0/contracts/token/ERC20/IERC20.sol
interface IERC20 {
    // 6 REQUIRED FUNCTIONS
    // 总发行量
    function totalSupply() external view returns (uint);  // -> 总发行量 100000000 *
10**decimals

    // 标准decimal： 18
    // USDT: 6
    // WBTC: 8
    function balanceOf(address account) external view returns (uint); // 指定账户的余额

    // 币的持有人直接调用，进行转账
    function transfer(address recipient, uint amount) external returns (bool);

    // 最常用的！！
    // 1. 我这个owner对合约进行approve，此时approve内部会修改allowance变量
    // 2. 合约内部调用transferFrom来支配owner的token
    function transferFrom( // spender就是这个合约
        address sender, // owner
        address recipient,  // 转给谁
        uint amount // 金额
    ) external returns (bool);
```

```
    // owner: 币的持有人
    // spender: 是指定帮助花费的代理人（被授权的人）
    function allowance(address owner, address spender) external view returns (uint); //
授权的额度

    // decimals view，这是一个public 的变量，自动提供了一个读取的方法 // 返回精度
    // 持有人对spender进行授权，在approve内部，会调用msg.sender来知道owner是谁
    function approve(address spender, uint amount) external returns (bool);

    // 2 REQUIRED EVENTS
    // 事件
    event Transfer(address indexed from, address indexed to, uint value);
    event Approval(address indexed owner, address indexed spender, uint value);

    // 3. OPTIONAL FUNCTIONS
    function name() public view returns (string)
    function symbol() public view returns (string)
    function decimals() public view returns (uint8)
}
```

以下是ERC20的案例：

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

import "./IERC20.sol";

contract ERC20 is IERC20 {
    uint public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;
    string public name = "Solidity by Example";
    string public symbol = "SOLBYEX";
    uint8 public decimals = 18;

    function transfer(address recipient, uint amount) external returns (bool) {
        balanceOf[msg.sender] -= amount;
        balanceOf[recipient] += amount;
        emit Transfer(msg.sender, recipient, amount);
        return true;
    }

    function approve(address spender, uint amount) external returns (bool) {
        allowance[msg.sender][spender] = amount;
        emit Approval(msg.sender, spender, amount);
        return true;
    }
```

```
    function transferFrom(
        address sender,
        address recipient,
        uint amount
    ) external returns (bool) {
        allowance[sender][msg.sender] -= amount;
        balanceOf[sender] -= amount;
        balanceOf[recipient] += amount;
        emit Transfer(sender, recipient, amount);
        return true;
    }

    function mint(uint amount) external {
        balanceOf[msg.sender] += amount;
        totalSupply += amount;
        emit Transfer(address(0), msg.sender, amount);
    }

    function burn(uint amount) external {
        balanceOf[msg.sender] -= amount;
        totalSupply -= amount;
        emit Transfer(msg.sender, address(0), amount);
    }
}
```

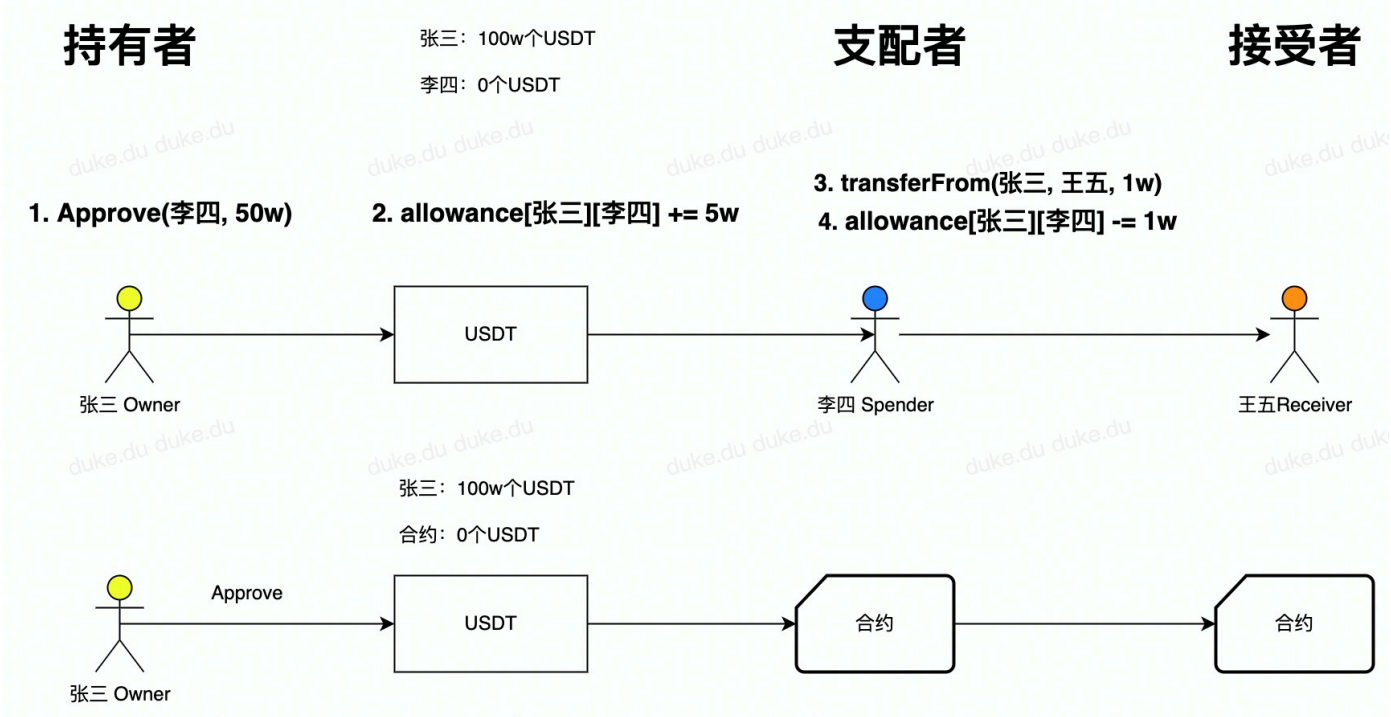
可以使用openzeppelin库进行创建自己的token：

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

// import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v4.0.0/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";


contract MyToken is ERC20 {
    constructor(string memory name, string memory symbol) ERC20(name, symbol) {
        // Mint 100 tokens to msg.sender
        // Similar to how
        // 1 dollar = 100 cents
        // 1 token = 1 * (10 ** decimals)
        _mint(msg.sender, 100 * 10**uint(decimals()));
    }

    // 默认是18，可以进行override
    function decimals() public view override returns (uint8) {
        return 6;
    }
}
```


持有者
张三：100w个USDT
李四：0个USDT
支配者
接受者
1. Approve(李四, 50w)
2. allowance[张三][李四] += 5w
3. transferFrom(张三, 王五, 1w)
4. allowance[张三][李四] -= 1w
USDT
张三 Owner
李四 Spender
王五Receiver
张三：100w个USDT
合约：0个USDT
Approve
USDT
合约
合约
张三 Owner