Solidity智能合约开发

从入门到精通

第5.1课

基础项目：简单代币合约

掌握ERC20代币标准

第一个完整区块链项目

ERC20代币标准

什么是ERC20, 为什么需要代币标准, 6个核心函数, 2个必需事件

实现核心功能

状态变量设计, transfer – 直接转账, approve – 授权机制, transferFrom – 授权转账

铸造和销毁功能

mint – 增加代币供应, burn – 减少代币供应, 权限控制 (onlyOwner)

部署和测试

Remix IDE部署流程, 多账户交互测试, 功能完整性验证

安全与优化

常见安全问题, OpenZeppelin库, Gas优化技巧, 最佳实践建议

ERC20 – 以太坊代币标准

什么是ERC20?

ERC20 = Ethereum Request for Comment 20

2015年由Fabian Vogelsteller提出

定义了可替代代币(Fungible Token)的统一接口

数据统计

超过**50万个**ERC20代币

日交易量超过**百亿美元**

99%的代币项目使用ERC20

知名ERC20代币

代币	符号	用途
Tether	USDT	稳定币
USD Coin	USDC	稳定币
Dai	DAI	去中心化稳定币
Chainlink	LINK	预言机代币
Uniswap	UNI	治理代币

为什么需要标准?

没有标准的问题:

钱包无法统一支持所有代币

交易所需要为每个代币定制代码

DeFi协议无法通用处理代币

开发者需要学习不同的接口

有了ERC20标准:

一套代码支持所有ERC20代币

钱包、交易所、DApp无缝集成

统一的用户体验

降低开发和集成成本

ERC20的6个核心函数

查询函数 (View)

```
function totalSupply() publicview returns (uint256) {
```

返回：代币总发行量
示例：1,000,000 tokens

```
function balanceOf(addressowner) public view returns (uint256) {
```

参数：账户地址
返回：uint256余额
示例：balanceOf(Alice) → 100

```
function allowance(address owner, address spender) public view returns (uint256) {
```

转账函数 (State-Changing)

```
function transfer(address to, uint256 amount) publicreturns (bool) {
```

参数：接收地址、数量
返回：bool成功标志
示例：transfer(Bob, 100)

```
function approve(addressspender, uint256 amount) public returns (bool) {
```

参数：被授权人、数量
返回：bool成功标志
示例：approve(Uniswap, 1000)

```
function transferFrom(addressfrom, address to, uint256amount) public returns (bool) {
```

事件

```
event Transfer(
address indexed from,
address indexed to,
uint256 value
);
```

记录所有转账
链下查询历史
钱包更新余额

```
event Approval(
address indexed owner,
address indexed spender,
uint256 value
);
```

记录授权操作
DApp监听授权
审计追踪

授权机制 – ERC20的核心设计

场景：Alice想在Uniswap上用USDT买ETH

1 授权 (Approve)

Alice → 调用:

```
usdt.approve(Uniswap, 1000)
```

USDT合约

```
allowance[Alice][Uniswap] = 1000
```

```
emit Approval(Alice, Uniswap, 1000)
```

2 使用授权 (TransferFrom)

Uniswap → 调用:

```
usdt.transferFrom(Alice, Pool, 500)
```

USDT合约

```
检查: allowance[Alice][Uniswap] >= 500
```

```
执行: Alice余额 –500
Pool余额 +500
allowance[Alice][Uniswap] –500
```

转账成功

Alice余额: 500 – 500 = 0

Pool余额: 0 + 500 = 500

授权额度: 1000 – 500 = 500

为什么需要授权机制?

- 智能合约无法主动获取用户代币
- transfer只能自己转给别人
- 合约不能直接从你账户取代币
- 解决方案：授权机制
- 用户主动授权合约，合约代表用户操作

安全提示

- 不要授权过大的额度
- approve(contract, type(uint256).max) ← 危险!
- 按需授权
- approve(contract, 实际需要的数量)
- 使用后撤销授权

代币合约结构

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract MyToken {
    // ============ 代币基本信息 ============
    string public name;         // 代币名称: "My Token"
    string public symbol;       // 代币符号: "MTK"
    uint8 public decimals;      // 小数位数: 18
    uint256 public totalSupply; // 总供应量: 1000 * 10^18

    // ============ 状态变量 ============
    // 记录每个地址的余额
    mapping(address => uint256) public balanceOf;

    // 记录授权关系: owner → spender → amount
    mapping(address => mapping(address => uint256)) public allowance;

    // ============ 事件 ============
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    // ============ 构造函数 ============
    constructor(
        string memory _name,
        string memory _symbol,
        uint8 _decimals,
        uint256 _initialSupply
    ) {
        // 初始化代币参数
        name = _name;
        symbol = _symbol;
        decimals = _decimals;

        // 分配初始供应量
        totalSupply = _initialSupply * (10**uint256(decimals));
```

许可证和版本

SPDX: MIT开源许可
pragma: Solidity 0.8.19

代币信息

name: 人类可读的名称
symbol: 代币符号(2–5字符)
decimals: 通常为18
totalSupply: 总发行量

状态变量

balanceOf: 余额映射
allowance: 双重映射存储授权

事件定义

Transfer: 记录转账历史
Approval: 记录授权操作

构造函数

实现transfer函数 – 直接转账

```
function transfer(address to, uint256 amount) public returns (bool)
{
    // 步骤1：检查接收地址
    require(to != address(0), "Cannot transfer to zero address");

    // 步骤2：检查余额
    require(balanceOf[msg.sender] >= amount, "Insufficient
balance");

    // 步骤3：更新余额
    balanceOf[msg.sender] -= amount;
    balanceOf[to] += amount;

    // 步骤4：触发事件
    emit Transfer(msg.sender, to, amount);

    // 步骤5：返回true
    return true;
}
```

步骤1：检查接收地址

防止发送到零地址 (address(0) = 0x000...000)
零地址是"黑洞"，发送到此地址的代币永远丢失

步骤2：检查余额

确保发送者余额 >= 转账数量
防止透支，保证交易有效

步骤3：更新余额

先减后加（CEI模式）
不会整数溢出（0.8.0+自动检查）
原子操作，确保数据一致性

步骤4：触发事件

记录转账历史
钱包监听并更新显示
区块链浏览器展示

approve – 授权函数

```
// approve - 授权函数
function approve(address spender, uint256 amount) public returns (bool) {
    // 检查被授权人地址
    require(spender != address(0), "Cannot approve zero address");

    // 设置授权额度
    allowance[msg.sender][spender] = amount;

    // 触发授权事件
    emit Approval(msg.sender, spender, amount);

    return true;
}
```

参数： spender(被授权人地址), amount(授权数量)

特点： 设置授权额度，不转移代币

注意： 新的授权会覆盖旧的授权

transferFrom – 授权转账

```
// transferFrom - 授权转账
function transferFrom(address from, address to, uint256 amount) public returns (bool) {
    // 检查地址有效性
    require(from != address(0), "From zero");
    require(to != address(0), "To zero");

    // 检查余额
    require(balanceOf[from] >= amount, "Insufficient balance");

    // 检查授权额度
    require(allowance[from][msg.sender] >= amount, "Insufficient allowance");

    // 更新余额
    balanceOf[from] -= amount;
    balanceOf[to] += amount;

    // 减少授权额度
    allowance[from][msg.sender] -= amount;

    // 触发转账事件
    emit Transfer(from, to, amount);

    return true;
}
```

参数： from(所有者), to(接收者), amount(数量)

特点： 使用授权额度进行转账

关键点： msg.sender必须被from授权

approve设置授权额度　transferFrom使用授权额度　共同构成ERC20核心授权机制

扩展功能 – Mint & Burn

Mint（铸造）– 增加供应

使用场景：

- 稳定币（USDC）
- 游戏代币
- 激励代币
- 流动性挖矿

特点：

- 增加totalSupply
- 凭空创造代币
- 通常需要权限控制
- from = address(0)

Burn（销毁）– 减少供应

使用场景：

- 稳定币（USDC）
- 通缩模型
- 购买服务
- 回购销毁

特点：

- 减少totalSupply
- 代币永久消失
- 任何人都可以销毁自己的代币
- to = address(0)

对比表：

特性	Mint	Burn
作用	增加供应	减少供应
totalSupply	增加	减少

课程知识点总结与作业布置

课程知识点总结

ERC20标准
了解ERC20代币标准的定义、历史和重要性，掌握其6个核心函数和2个必需事件

核心功能实现
实现transfer、approve和transferFrom函数，理解授权机制的工作原理

扩展功能
添加mint和burn函数，实现代币的铸造和销毁功能

安全实践
避免整数溢出/下溢、重入攻击、approve竞态条件等常见安全问题

作业布置

创建自己的代币
使用Solidity编写一个完整的ERC20代币合约，包含名称、符号、小数位数等基本信息

必做

实现批量转账
添加一个函数，允许用户一次将代币转账给多个地址

必做

添加暂停功能
实现一个只有所有者可以调用的功能，用于暂停/恢复合约中的转账操作

必做

使用OpenZeppelin重写
使用OpenZeppelin库重新实现一个更安全的ERC20代币合约

选做

部署到测试网
将你的代币合约部署到Rinkeby或Goerli测试网，并展示交互过程

选做