**智能合约访问权限控制（入门篇）Ownable.sol**

*本文参考 [OpenZeppelin文档](https://docs.openzeppelin.com/contracts/4.x/api/access) 并结合自己的理解。*

**智能合约访问权限非常重要**

由于所有项目的智能合约代码都是开源的(不开源没人敢用)，这使得合约逻辑中的漏洞更容易被人利用。而多数合约的漏洞都是由于访问权限(Access Control)的设置出了问题。

本文笔者将介绍一种最简单的权限控制手段，引入 Ownable.sol 合约。

**快速构建**

在你的合约中直接导入 OpenZepplin 官方提供的库即可。

|  |
| --- |
| Solidity import "@openzeppelin/contracts/access/Ownable.sol"; |

**Ownable.sol**

最简单的模型就是分为**两类身份**，普通人和所有者(Owner)。

* 普通人即除所有者外所有账户，他们没有特权，调用的方法任何人都能调用。
* 所有者(Owner)拥有一定的特权，可以调用普通人无法调用的方法。

我们先来看下源码，然后逐行分析。

[Ownable.sol](https://github.com/Blockchain-Engineer-Learning/Contract-Interpretation/blob/main/Access-Control/Ownable/Ownable.sol)

|  |
| --- |
| Solidity // SPDX-License-Identifier: MIT pragma solidity ^0.8.0;  abstract contract Ownable {  address private \_owner;   event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);   constructor() {  \_transferOwnership(msg.sender);  }   function owner() public view virtual returns (address) {  return \_owner;  }   modifier onlyOwner() {  require(owner() == msg.sender, "Ownable: caller is not the owner");  \_;  }   function renounceOwnership() public virtual onlyOwner {  \_transferOwnership(address(0));  }   function transferOwnership(address newOwner) public virtual onlyOwner {  require(newOwner != address(0), "Ownable: new owner is the zero address");  \_transferOwnership(newOwner);  }   function \_transferOwnership(address newOwner) internal virtual {  address oldOwner = \_owner;  \_owner = newOwner;  emit OwnershipTransferred(oldOwner, newOwner);  } } |

**分析**

- 合约中仅定义了一个状态变量 `\_owner` 用来存放所有者(Owner)地址。

- 仅有一个事件 `OwnershipTransferred` ，在所有者身份发生转变时触发。

- 构造函数在合约部署时将部署者地址设为默认所有者(Owner)。

- 修饰器 `onlyOwner` 用来检查消息调用者是否为所有者(Owner)，给需要的函数加上访问权限控制。

- `renounceOwnership` 函数是放弃所有者(Owner)权限，代码表现为移权给零地址，该操作将使所有带有 `onlyOwner` 修饰器的函数作废(无法被调用)。

- `transferOwnership` 函数通过调用内部的 `\_transferOwnership` 函数来改变所有者地址。

**小结**

Ownable.sol 合约标注为抽象(abstract)类型，通常以继承的方式来使用。

函数都为 virtual 修饰，可以在继承的子合约中重写。