Solidity智能合约开发

从入门到精通

Parent

```
// 合约继承示例
contract Child is Parent {
  function foo() public override { }
}
```

Child

第6.1课

合约继承

代码复用　面向对象编程　模块化设计

讲师：[layer]

今天的学习内容

为什么需要继承
代码复用、维护与升级

单继承和多重继承
基础语法与C3线性化

super关键字
调用父合约函数

构造函数继承
执行顺序与参数传递

函数重写
virtual与override关键字

抽象合约和接口
定义规范与标准

为什么需要继承?

没有继承的问题

代码重复冗余
相同功能在多个合约中重复实现

维护困难
修改功能需要在多处进行

容易出错
重复代码容易产生不一致

无法统一升级
难以对多个合约进行一致性更新

VS

使用继承的解决方案

代码复用
功能在父合约中定义，子合约自动获得

易于维护
修改父合约中的功能，所有子合约自动受益

统一升级
通过更新父合约实现统一升级

模块化设计
将功能拆分为独立合约，提高代码组织性

单继承和多重继承基础

单继承语法

```
// 单继承语法
contract Child is Parent {
  // 子合约代码
}
```

关键字说明

- is：表示继承关系
- Child：子合约
- Parent：父合约

访问权限控制

- public：子合约可以访问
- internal：子合约可以访问
- private：子合约无法访问

多重继承与C3线性化

```
// 多重继承语法
contract Child is Parent1, Parent2 {
  // 同时继承多个父合约
}
```

继承顺序

- 从左到右：Parent1优先于Parent2
- 使用C3线性化算法确定顺序

C3线性化示例

继承顺序： Child → Parent1 → Parent2 → GrandParent

super关键字与构造函数继承

super关键字

作用

- 调用父合约的函数
- 即使子合约重写了该函数

```
// 语法示例
function foo() public override returns (string memory) {
  // 调用父合约的foo()
  string memory parentResult = super.foo();
  return string.concat("Child -> ", parentResult);
}
```

使用场景

- 扩展父合约功能
- 在重写函数中调用父函数
- 多重继承中的函数链

构造函数继承

执行顺序

- 父合约构造函数先执行
- 按照继承顺序从左到右
- 最后执行子合约构造函数

```
// 参数传递方式1：继承声明中
contract Child is Parent(100) {
  // Parent的构造函数接收100
  constructor() {
    // 子合约构造函数
  }
}
```

```
// 参数传递方式2：子构造函数中
contract Child is Parent {
  constructor(uint256 _value) Parent(_value) {
    // 通过Parent(_value)传递参数
  }
}
```

函数重写 – virtual和override

基本示例

```
contract Parent {
  // virtual：表示可以被重写
  function foo() public virtual returns ("Parent") {
  }
}

contract Child is Parent {
  // override：表示重写父合约函数
  function foo() public override returns ("Child") {
  }
}
```

关键字说明

- **virtual**：在父合约中使用，表示函数可以被重写
- **override**：在子合约中使用，表示重写父合约函数
- 两者必须**配对使用**才能成功重写函数

多重继承中的规则

```
contract A {
  function foo() public virtual returns ("A") {
  }
}

contract B {
  function foo() public virtual returns ("B") {
  }
}

contract C is A, B {
  // 必须明确指定重写哪些父合约
  function foo() public override(A, B) returns ("C") {
  }
}
```

错误示例：

```
contract C is A, B {
   function foo() public override returns ("C") {
   }
}
```

重要提示

构造函数继承 – 执行顺序

构造函数执行顺序

执行规则

- 父合约构造函数先执行
- 按照继承顺序从左到右
- 最后执行子合约构造函数

执行顺序示例

```
// 父合约构造函数
contract A {
  constructor() {
1.  // 第一个执行
  }
}

// 父合约构造函数
contract B {
  constructor() {
2.  // 第二个执行
  }
}

// 子合约构造函数
contract C is A, B {
  constructor() A() B() {
3.  // 最后执行
  }
```

构造函数参数传递

方式1：继承声明中传递

```
contract Parent {
  uint256 public value;
  constructor(uint256 _value) {
    value = _value;
  }
}

contract Child is Parent(100) {
  // Parent的构造函数接收100
  constructor() {
    // 子合约构造函数
  }
}
```

✔ 适合固定值　✔ 代码简洁

方式2：子构造函数中传递

```
contractParent {
```

抽象合约与接口

抽象合约

关键字

abstract – 声明抽象合约

特点

- 可包含未实现的函数
- 不能直接部署
- 必须被继承
- 可以有状态变量和构造函数
- 可以有部分实现

使用场景

- 定义基础规范
- 强制子合约实现特定功能
- 部分实现的基础合约

接口

关键字

interface – 声明接口

特点

- 所有函数必须是 external
- 不能有状态变量
- 不能有构造函数
- 不能有实现（函数体为空）
- 可以声明事件

使用场景

- 定义标准接口
- 合约间交互
- 强制实现规范

对比总结

实际应用 – OpenZeppelin库

OpenZeppelin代币合约示例

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

// 导入OpenZeppelin合约
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/security/Pausable.sol";

// 继承多个OpenZeppelin合约
contract MyToken is ERC20, Ownable, Pausable {
    constructor() ERC20("My Token", "MTK") {
        _mint(msg.sender, 1000 * 10** decimals());
    }

    // 重写transfer函数，添加暂停功能
    function transfer(address to, uint256 amount)
        public
        override
        whenNotPaused
        returns (bool)
    {
        return super.transfer(to, amount);
    }

    // onlyOwner修饰符确保只有所有者可以调用
    function pause() public onlyOwner {
        _pause();
    }
```

OpenZeppelin优势

- 经过安全审计
- 广泛使用和测试
- 标准化实现
- 持续更新维护

常用合约

- ERC20
- Ownable
- Pausable
- ReentrancyGuard

继承最佳实践与常见错误

最佳实践

使用继承复用代码
避免代码重复，提高开发效率

使用OpenZeppelin库
经过安全审计的标准化实现

明确继承顺序
从左到右，深度优先

正确使用virtual和override
父合约：virtual，子合约：override

使用抽象合约定义规范
强制子合约实现特定功能

使用接口定义标准
合约间交互的规范定义

常见错误

忘记标记virtual
父合约函数未标记virtual导致无法重写

忘记标记override
子合约函数未标记override导致编译错误

继承顺序错误
导致函数调用链异常

构造函数参数传递错误
参数类型或数量不匹配

多重继承中override不明确
未明确指定重写哪些父合约

课程总结与实践作业

核心知识点回顾

- **代码复用：**继承允许子合约获得父合约的功能，减少代码重复
- **单继承与多重继承：**子合约可以继承多个父合约，遵循C3线性化算法
- **super关键字：**用于调用父合约的函数，即使子合约重写了该函数
- **构造函数继承：**父合约构造函数先执行，按照继承顺序从左到右
- **函数重写：**使用virtual和override关键字控制函数的可重写性
- **抽象合约与接口：**定义规范和标准，接口用于合约间交互

实践作业

必做题目

1. 创建多级权限管理系统，实现onlyOwner修饰符和角色管理
2. 实现ERC20接口，创建自己的代币合约，包含名称、符号、总供应量等

选做题目

1. 使用OpenZeppelin库实现更安全、更完善的合约
2. 创建可升级的合约系统，了解代理模式和升级陷阱