(合约继承的重要特点：

a.派生合约可以访问父合约的所有非私有成员，包括内部方法和状态变量。但是不允许使用this。

b.如果函数输出参数保持不变，则允许函数重写。如果输出参数不同，编译将失败。

c.可以使用super关键字或父合约名称调用父合约的函数。

)

**1. 继承的本质**

继承的实现方案是代码拷贝，所以合约继承后，部署到网络时，将变成一个合约。

代码将从父类拷贝到子类中。

**2. 继承的定义**

继承使用 is，一个合约可以继承多个合约，用逗号分开。

如果继承的合约之间也有父子关系，那么合约要按照先父后子的顺序排序。

比如：

contract A {}

contract B is A {} // B继承了A

contract C is A, B {} // 先A，再B

contract C is B, A {} // 错误

**3. 继承的可见性**

子合约不能访问父合约的 private私有成员;

子合约可以访问父合约所有的非私有成员（包括 internal的函数和状态变量）。

**4. 父合约传参**

继承的子合约，必须提供父合约构造方法需要的所有参数。

有两种方式实现，如下：

contract Liu{

uint x;

function Liu(uint \_x) {

x = \_x;

}

}

// 方式一，在定义继承的时候传参数

contract DerivedA is Liu(7){}

// 方式二，在合约构造函数中传参数

contract DerivedB is Liu{ // 仅指定继承关系而不提供参数，请不要加括号

function DerivedB(uint \_y) Liu(\_y) {} // 0.4.x之前旧版本的构造函数写法

constructor(uint \_y) Liu(\_y) {} // 0.5.x之后新版本的构造函数写法

}

**5. 重写**

在子类中允许重写函数，但不允许重写返回参数类型。

如下面的代码：

pragma solidity ^0.8.16;

contract Liu{

function data() public pure returns(uint){

return 1;

}

}

contract InheritOverride is Liu{

function data(uint) public pure {}

function data() public pure returns(uint){}

// function data() public pure returns(string){} // 报错

}

上面代码中的 function data() returns(string){} 将导致报错：

// 报错信息为：

TypeError: Overriding function return types differ.

因为不能修改返回类型。

**6. 最远继承原则**

在继承链中，继承实现是代码复制。实际执行时，依据的是最远继承原则（most derived）。

示例：

pragma solidity ^0.8.16;

contract Liu1 {

function data() public view returns(uint){

return 1;

}

}

contract Liu2 {

function data() public view returns(uint){

return 2;

}

}

contract Derived1 is Liu1,Liu2 {

function call() public view returns(uint) {

return data(); // 返回 2

}

}

contract Derived2 is Liu2,Liu1 {

function call() public view returns(uint) {

return data(); // 返回 1

}

}

**7. super关键字**

有些时候，我们希望继承链条上每一个函数都能被调用；这个时候，需要用到super关键字。

示例：

pragma solidity ^0.8.16;

contract owned {

address owner;

constructor() public {

owner = msg.sender;

}

}

contract simple is owned {

function kill() public {

if (msg.sender == owner) selfdestruct(owner);

}

}

contract Liu1 is simple {

function kill() public {

simple.kill();

}

}

contract Liu2 is simple {

function kill() public {

simple.kill();

}

}

contract Final is Liu1, Liu2 {}

在 Final 中调用 kill() ，将仅仅触发 Liu2.kill() 被调用，因为它是最远继承合约，从而跳过Liu1.kill()。

如果我们想也触发 Liu1.kill()，解决方案是使用 super关键字：

pragma solidity ^0.8.16;

contract owned {

address owner;

constructor() public {

owner = msg.sender;

}

}

contract simple is owned {

function kill() public {

if(msg.sender == owner) selfdestruct(owner);

}

}

contract Liu1 is simple {

function kill() public {

super.kill();

}

}

contract Liu2 is simple {

function kill() public {

super.kill();

}

}

contract Final is Liu1, Liu2 {}

**8.接口**

接口与抽象合约类似，与之不同的是，接口内没有任何函数是已实现的，同时还有如下限制：{

不能继承其它合约或接口 / 不能定义构造器 / 不能定义变量 / 不能定义结构体 / 不能定义枚举类

}

接口基本上限制为合约ABI定义可以表示的内容，ABI和接口定义之间的转换应该是可能的，不会有任何信息丢失。

接口用自己的关键词表示：

interface Token {

function transfer(address recipient, uint amount);

}

就像继承其他合约一样，合约可以继承接口。

接口中的函数都会隐式的标记为 virtual ，意味着他们会被重写。

但是不表示重写（overriding）函数可以再次重写，仅仅当重写的函数标记为 virtual 才可以再次重写。