第6章 面向对象

- 1 创建合约
- 2 面向对象
- 3 继承
- 4 覆盖
- 5 抽象合约
- 6 接口合约
- 7 库合约

目录

CONTENTS

CONTENTS

章节

第一章 创建合约

腾讯云

创建合约

- 创建（部署）合约有两种方式，前面课程中使用的在WeBASE-Front的IDE中点击“部署”按钮直接创建合约，如果想要在A合约中动态创建合约，可以使用 “new” 关键字创建

创建合约

- 通过“部署”按钮创建合约时，默认会执行合约中的构造函数，构造函数常用于全局变量的初始化，构造函数不是必须存在的，可以省略
- 构造函数与普通函数不同点
  - 构造函数不使用function关键字声明，而是使用constructor关键字声明
  - 构造函数没有返回值

创建合约

- 构造函数合约示例

Test.sol 保存 编译

```
pragma solidity 0.6.10;


contract Test {
    uint age;
    string name;

    // 初始化age和name全局变量
    constructor(uint _age, string memory _name) public {
        age = _age;
        name = _name;
    }
}
```

创建合约

- 当构造函数存在参数时，部署合约时需要传递对应的参数

new关键字

- 创建并编写Test2合约，在Test合约中引入Test2合约，并使用new关键字创建Test2合约

Test2.sol

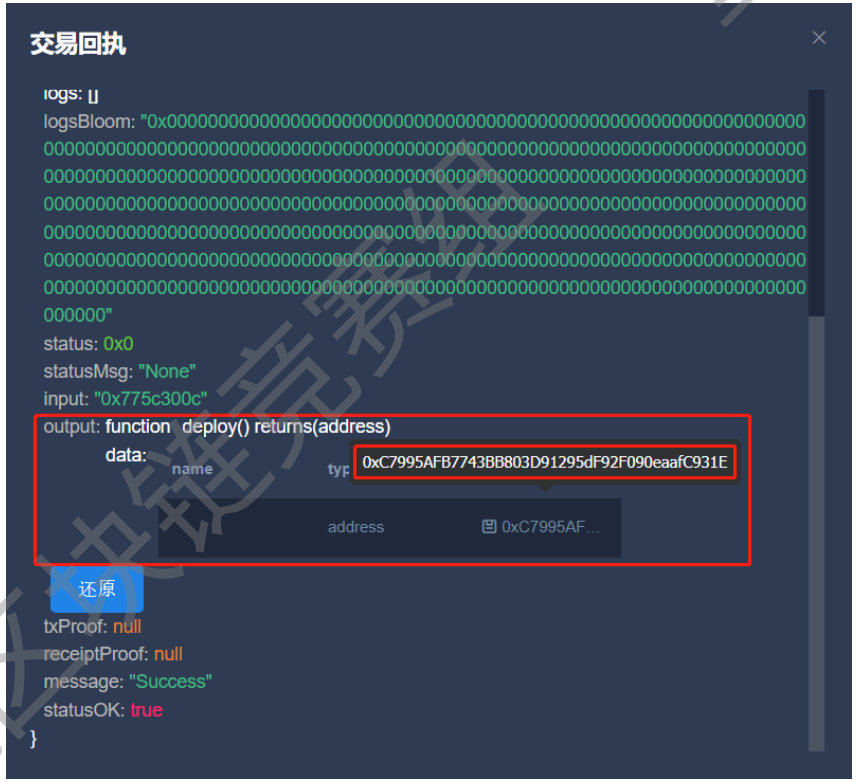
```
pragma solidity 0.6.10;


contract Test2 {
    uint x;

    // 设置x值
    constructor(uint _x) public {
        x = _x;
    }
    // 获取x值
    function getX() public returns(uint) {
        return x;
    }

}
```

Test.sol 保存 编译

```
pragma solidity 0.6.10;
import "./Test2.sol";


contract Test {
    address t2Address;

    function deploy() public returns(address) {
        // 使用new关键字部署Test2合约，传入初始化参数10
        Test2 t2 = new Test2(10);
        // 使用address(t2)获取部署的Test2合约地址
        t2Address = address(t2);
        return t2Address;
    }
    // 通过t2Address地址调用Test2合约的getX函数
    function get() public returns(uint) {
        return Test2(t2Address).getX();
    }
}
```

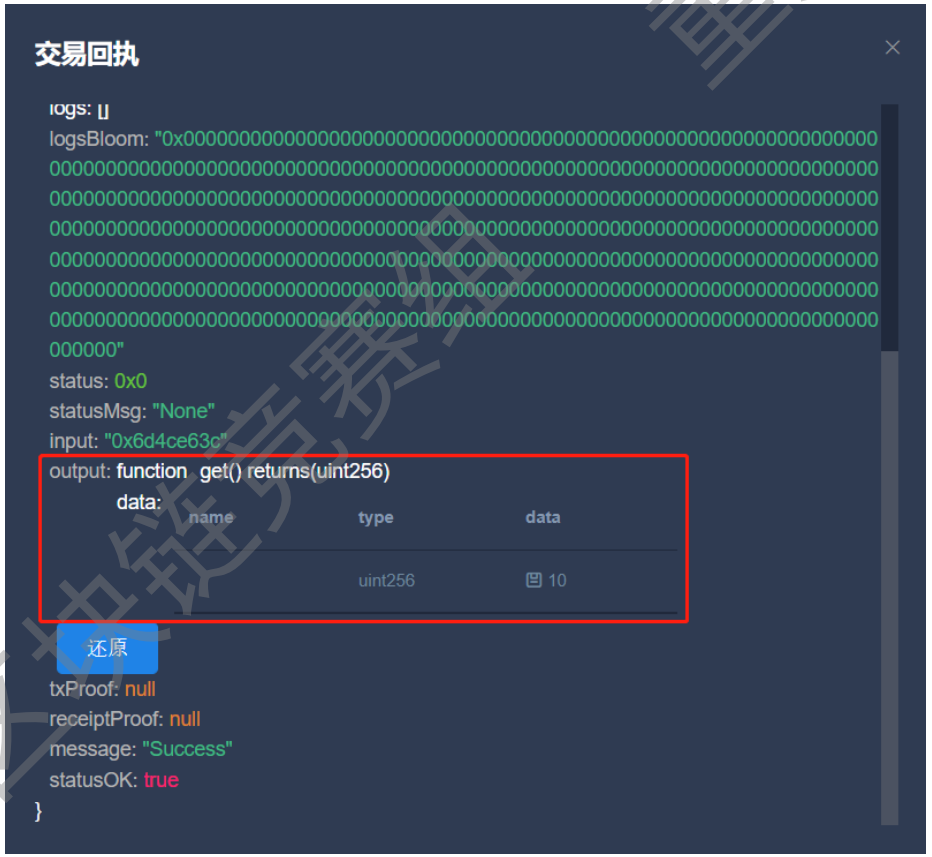
new关键字

- 编译并部署Test合约，调用deploy函数，记录下返回的合约地址



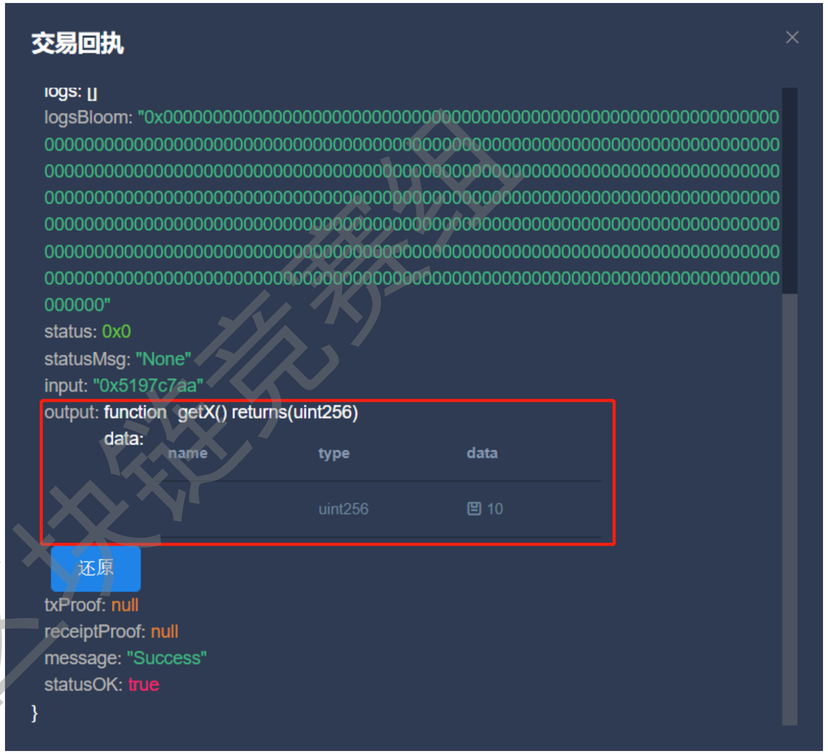
new关键字

- 调用Test合约中的getX函数，获取Test2合约中x的值



new关键字

- 切换到Test2合约，直接点击“合约调用”按钮，输入记录下的合约地址，调用getX函数获取x变量值



腾讯云

CONTENTS

章节

第二章 面向对象

2.1 面向对象

腾讯云

面向对象

- 面向对象编程，是一种通过对象的方式，把现实世界映射到计算机模型的一种编程方法。
- 现实世界中，我们定义了“猫”这种抽象概念，而具体的猫则是“花花”“小咪”“大咪”等一个个具体的猫。所以，“猫”可以定义为一个合约（contract），而具体的猫则是实例（instance）

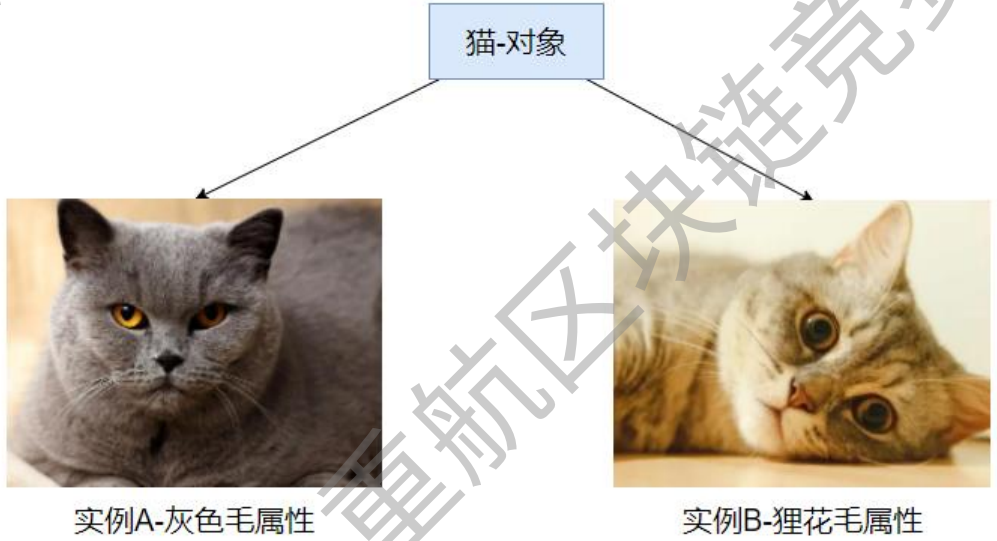
面向对象

- 现实世界与合约模型对应关系

现实世界	合约模型	合约代码
猫	合约/contract	contract Cat {}
花花	合约实例/0xaa...bb	Cat huahua = new Cat()
小咪	合约实例/0xaa...cc	Cat xiaomi = new Cat()
大咪	合约实例/0xaa...dd	Cat dami = new Cat()

面向对象

- 只要理解了对象和实例的概念，就明白了什么是面向对象编程
  - 对象（contract），是一种模板，它定义了如何创建实例，实例能实现什么样的功能
  - 实例（instance），是根据对象创建的实例，可以创建多个实例，每个实例类型相同，但各自属性可能不相同



面向对象

- 面向对象示例，创建一个“猫”合约，可以根据这个模板创建多个“猫”合约实例，不同的猫具有不同的属性，但它们有统一的行为，“跑”和“叫”

Cat.sol

保存 编译 部

```
pragma solidity 0.6.10;

// 定义一个“猫”合约对象
contract Cat {
    // 猫的属性-名字
    string name;
    // 猫的属性-年龄
    uint age;

    constructor(string memory _name, uint _age) public {
        name = _name;
        age = _age;
    }
    // 获取猫名字
    function getName() public returns(string memory) {
        return name;
    }
    // 获取猫年龄
    function getAge() public returns(uint) {
        return age;
    }
    // 猫的行为-跑
    function run() public returns(string memory) {
        return "cat run";
    }
    // 猫的行为-叫
    function cry() public returns(string memory) {
        return "miao";
    }

}
```

面向对象

- 部署两个合约，部署合约时传入不同的属性，部署成功的合约即为两个“猫”合约实例

CONTENTS

章节

第三章 继承

继承

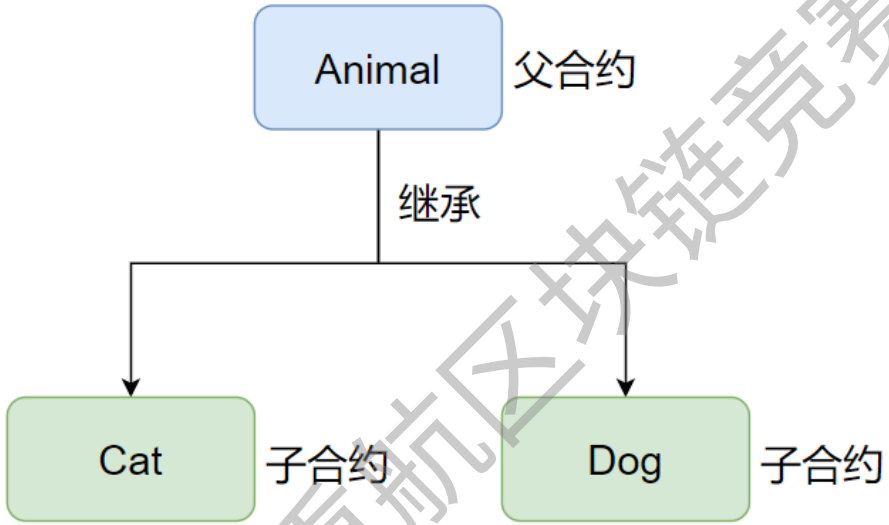
- 在前面章节，我们定义了Cat对象，假设我们需要再次定义一个Dog合约对象，可以观察到在Dog合约中，包含了在Cat合约中重复出现过的属性和函数
- 那么能不能不编写重复的代码呢？此时继承就派上了用场

Dog.sol 保存 编译

```
pragma solidity 0.6.10;

// 定义一个“狗”合约对象
contract Dog {
    // 狗的属性-名字
    string name;
    // 狗的属性-年龄
    uint age;

    constructor(string memory _name, uint _age) public {
        name = _name;
        age = _age;
    }
    // 获取狗名字
    function getName() public returns(string memory) {
        return name;
    }
    // 获取狗年龄
    function getAge() public returns(uint) {
        return age;
    }
    // 狗的行为-跑
    function run() public returns(string memory) {
        return "dog run";
    }
    // 狗的行为-叫
    function cry() public returns(string memory) {
        return "wang";
    }

}
```

继承

- “猫”和“狗”都属于动物，再编写一个“动物”合约，让“猫”和“狗”合约继承它，被继承的合约称为父合约（基合约），继承的合约称为子合约（派生合约）。继承可以将父合约中的变量、函数、修饰器、事件继承到子合约中



继承

- 定义一个“Animal”合约，实现与“Cat”“Dog”合约中重复的部分

Animal.sol 保存 编译

```
pragma solidity 0.6.10;


contract Animal {
    string name;
    uint age;

    constructor(string memory _name, uint _age) public {
        name = _name;
        age = _age;
    }

    function getName() public returns(string memory) {
        return name;
    }

    function getAge() public returns(uint) {
        return age;
    }
}
```

继承

- 修改Cat合约，导入Animal合约并使用is关键字继承Animal合约，此时Cat合约就拥有了Animal合约中的属性和方法（有external、private限定符的除外）

Cat.sol　保存　编译　部署　合约调用

```
pragma solidity 0.6.10;
import "./Animal.sol";

// Cat通过is继承Animal合约
contract Cat is Animal {
    // 调用Animal合约中的constructor构造函数
    constructor(string memory _name, uint _age) Animal(_name, _age) public {
    }
    // 猫的行为-跑
    function run() public returns(string memory) {
        return "cat run";
    }
    // 猫的行为-叫
    function cry() public returns(string memory) {
        return "miao";
    }

}
```

继承

- 同样地，修改Dog合约，使其继承Animal合约

Dog.sol　保存　编译　部署　合约调用

```
pragma solidity 0.6.10;
import "./Animal.sol";


// Cat通过is继承Animal合约
contract Dog is Animal {
    // 调用Animal合约中的constructor构造函数
    constructor(string memory _name, uint _age) Animal(_name, _age) public {
    }
    // 狗的行为-跑
    function run() public returns(string memory) {
        return "dog run";
    }
    // 狗的行为-叫
    function cry() public returns(string memory) {
        return "wang";
    }

}
```

继承

- 在子合约中可以执行父合约中的方法，支持使用super.function()或者合约名.function()两种方法调用

Dog.sol　保存　编译　部署　合约调用

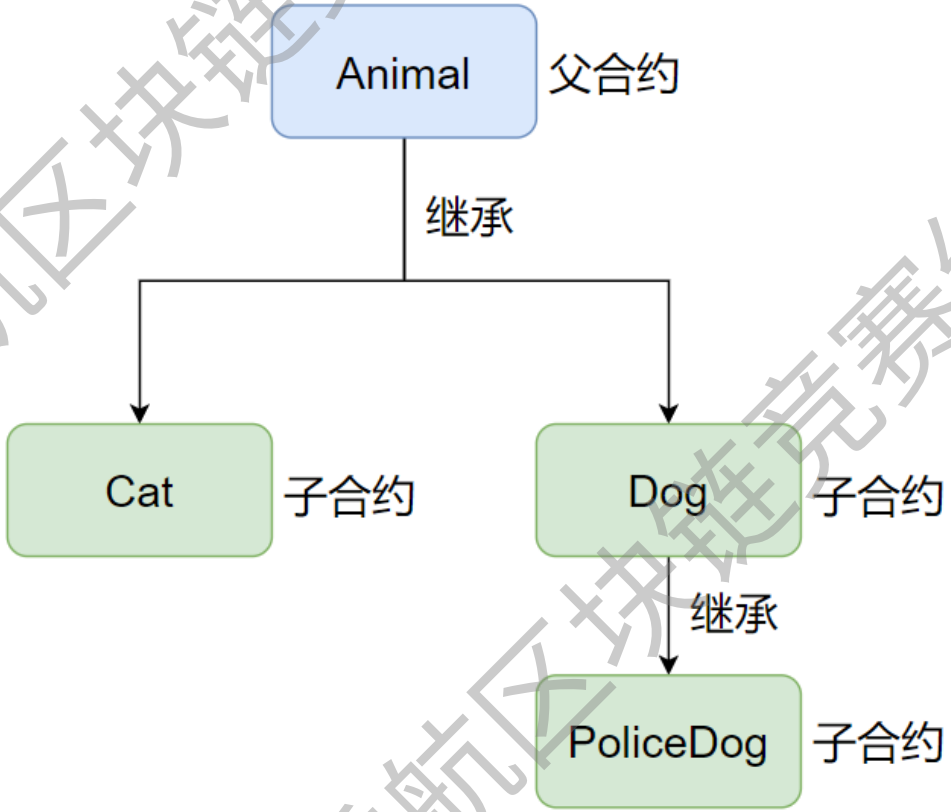
```
pragma solidity 0.6.10;
import "./Animal.sol";

// Cat通过is继承Animal合约
contract Dog is Animal {
    // 调用Animal合约中的constructor构造函数
    constructor(string memory _name, uint _age) Animal(_name, _age) public {
    }
    // 狗的行为-跑
    function run() public virtual returns(string memory) {
        return "dog run";
    }
    // 狗的行为-叫
    function cry() public returns(string memory) {
        return "wang";
    }
    // 使用super调用父合约中的getName函数
    function dogName() public returns(string memory) {
        return super.getName();
    }
    // 通过指定合约名调用父合约中的getAge函数
    function dogAge() public returns(uint) {
        return Animal.getAge();
    }

}
```

继承

- 部署Cat合约，传入“大咪”和“5”两个参数，部署完成后点击“合约调用”，在Cat合约中可以访问父合约Animal中的函数

多级继承

- 子合约仍然可以作为父合约被另一个合约继承，称为多级继承



多级继承

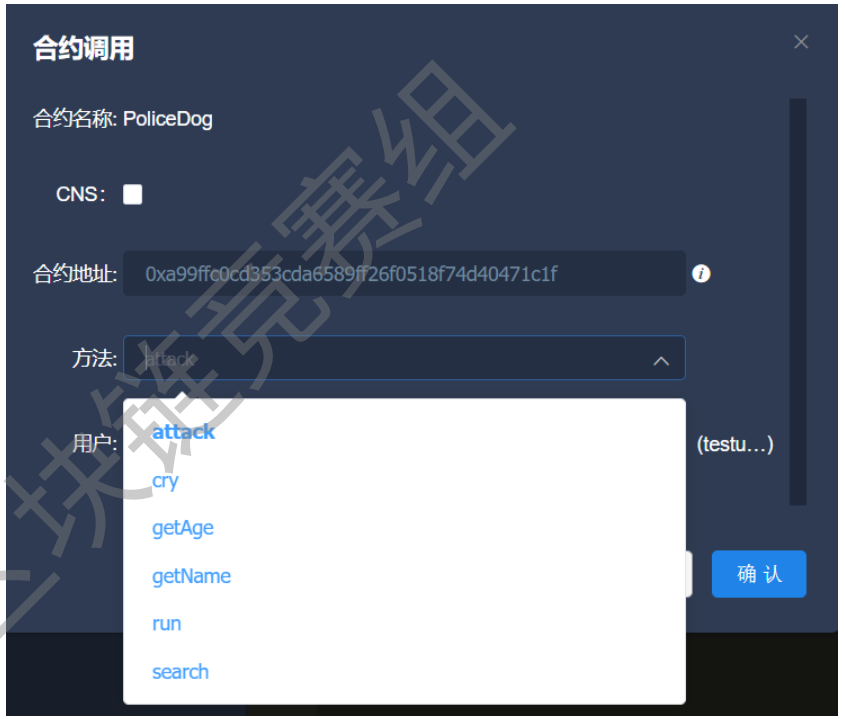
- 定义一个PoliceDog警犬合约，使其继承Dog合约，具有独立的“search”和“attack”函数

PoliceDog.sol　保存　编译　部署　合约调用

```
pragma solidity 0.6.10;
import "./Dog.sol";

// 警犬合约继承狗合约
contract PoliceDog is Dog {

    // 调用Dog合约中的constructor构造函数
    constructor(string memory _name, uint _age) Dog(_name, _age) public {
    }

    // 警犬搜索毒品
    function search() public returns(string memory) {
        return "police dogs search for drugs";
    }

    // 警犬攻击坏人
    function attack() public returns(string memory) {
        return "police dogs attack bad guys";
    }
}
```

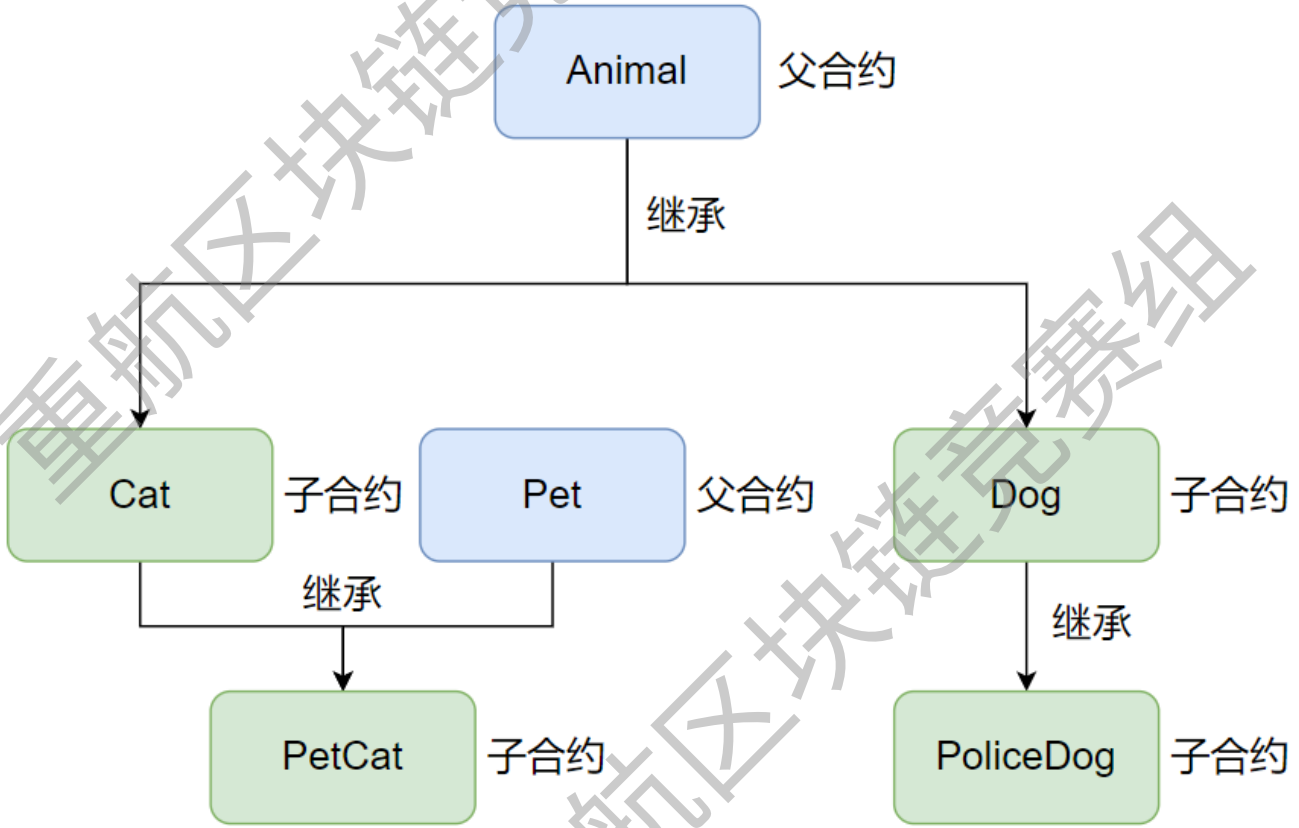
多级继承

- 部署PoliceDog合约，传入“大黄”和“3”两个参数，部署完成后点击“合约调用”，在PoliceDog中可以访问父合约的父合约中的函数



多重继承

- 子合约可以同时继承两个父合约，称为多重继承



多重继承

- 定义一个Pet宠物合约，具有独立的“play”和“accompany”函数

Pet.sol 保存 编译

```
pragma solidity 0.6.10;

contract Pet {

    // 和人玩耍
    function play() public returns(string memory) {
        return "play with people";
    }

    // 陪伴人
    function accompany() public returns(string memory) {
        return "accompany with people";
    }
}
```

多重继承

- 定义一个PetCat宠物猫合约，使其继承Cat和Pet合约

PetCat.sol　保存　编译　部署　合约调用

```
pragma solidity 0.6.10;
import "./Cat";
import "./Pet";

// 宠物猫合约继承宠物合约和猫合约
contract PetCat is Cat, Pet {

    // 调用Cat合约中的constructor构造函数
    constructor(string memory _name, uint _age) Cat(_name, _age) public {
    }

}
```

多重继承

- 部署PetCat合约，传入“小咪”和“3”两个参数，部署完成后点击“合约调用”，在PetCat中可以访问多重继承合约中的函数

CONTENTS

章节

第四章 覆盖

4.1 覆盖

腾讯云

覆盖

- 覆盖（override），即在子合约中将父合约中的实现覆盖，以达到重新实现功能的目的。
  - 修饰符（modifier）、函数（function）支持覆盖
  - 父合约中允许重写的函数要添加virtual关键字标识
  - 子合约中重写的函数要添加override关键字标识

覆盖

- 编写Test和Test2合约，使Test2合约覆盖Test合约中的修饰符和函数

Test.sol 保存 编译 部署

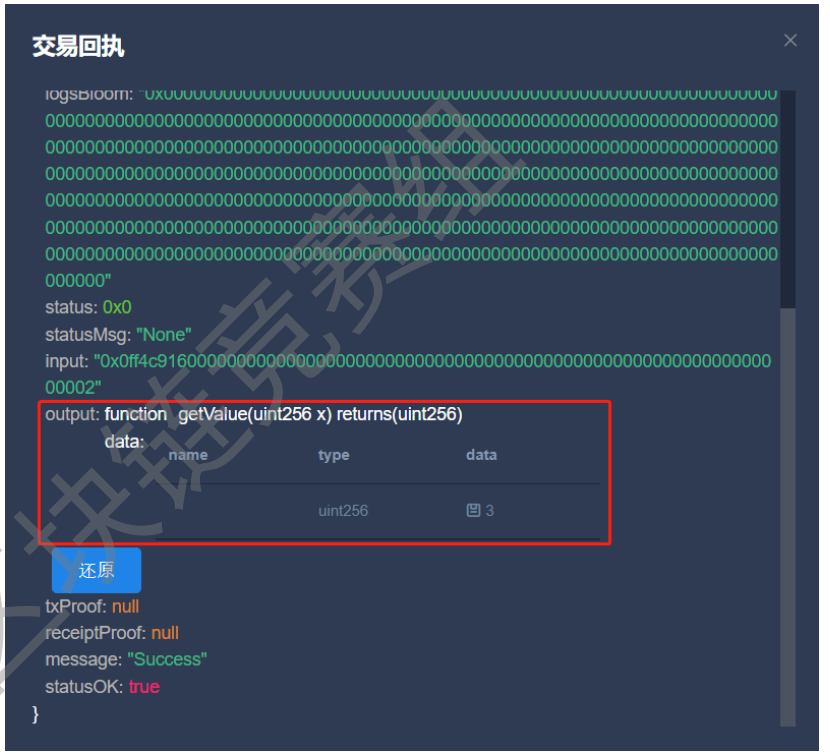
```
pragma solidity 0.6.10;


contract Test {
    // 检查value必须大于等于5
    modifier checkValue(uint x) virtual {
        require(x >= 5);
        _;
    }
    // 获取输入的value值
    function getValue(uint x) public virtual returns(uint) {
        return x;
    }

}
```

Test2.sol 保存 编译 部署

```
pragma solidity 0.6.10;
import "./Test.sol";


contract Test2 is Test {
    // 检查value必须大于等于2
    modifier checkValue(uint x) override {
        require(x >= 2);
        _;
    }
    // 获取输入的value+1值
    function getValue(uint x) public override returns(uint) {
        return x + 1;
    }

}
```

覆盖

- 部署Test2合约，调用getValue方法，传入x=2，合约执行成功，修饰符和函数都被覆盖



覆盖

- 当子合约为多重继承的合约时，要使用override关键字指定所有继承的合约，子合约同时继承Test和Test2合约

Test.sol

保存 编译 部署

```
pragma solidity 0.6.10;


contract Test {
    // 检查value必须大于等于5
    modifier checkValue(uint x) virtual {
        require(x >= 5);
        _;
    }
    // 获取输入的value值
    function getValue(uint x) public virtual returns(uint) {
        return x;
    }
}
```

Test2.sol

保存 编译 部署

```
pragma solidity 0.6.10;


contract Test2 {
    // 检查value必须大于等于2
    modifier checkValue(uint x) virtual {
        require(x >= 2);
        _;
    }
    // 获取输入的value+1值
    function getValue(uint x) public virtual returns(uint) {
        return x + 1;
    }
}
```

覆盖

- 在Test3合约中，使用override(Test,Test2)覆盖Test和Test2合约的实现

Test3.sol　保存　编译　部署　合约调用

```
pragma solidity 0.6.10;
import "./Test.sol";
import "./Test2.sol";


// Test继承Test和Test2合约
contract Test3 is Test, Test2 {
    // 检查value必须大于等于1
    modifier checkValue(uint x) override(Test, Test2) {
        require(x >= 1);
        _;
    }
    // 获取输入的value+1值
    function getValue(uint x) public override(Test, Test2) returns(uint) {
        return x + 2;
    }

}
```

作业

1.创建合约有哪两种方式？

2.在A合约中如何继承B合约？

3.继承有哪几种方式？

4.在子合约中覆盖父合约中的函数时，需要使用哪几种关键字？

5.在子合约中调用父合约方法，有哪两种方式？

谢谢