为什么要重视编程思想

2022-07-24

1

前两天遇到一个小问题，Solidity 写的智能合约超过 24 KB，不能部署到以太坊主网上，因为 EVM 对合约的代码小小有限制。于是考虑怎么减小合约的大小，当时对合约大小的概念都是模糊不清的。

其中注意到一个地方，合约是可以引入其他合约、调用其他合约方法的，只需要把部署后的合约地址作为参数传到合约里：

```
contract Demo {}
contract Main {
    Demo demo;
    constructor(Demo _demo) public {
        demo = _demo;
    }
}
```

合约大小包括引入的合约吗？EVM 在执行合约的时候，会不会先把其他合约的代码也加载进来，然后一起运行？代码大小的计算要包括所有合约？那可就麻烦了。

后来注意到，可以使用接口替代合约：

```
contract IDemo {}
contract Demo is IDemo {}
contract Main {
    IDemo demo;
    contract(IDemo _demo) public {
        demo = _demo;
    }
}
```

接口的代码量一定是少于具体实现的，因为接口不包含方法体，把引入的合约全部替换成接口，合约不就小多了？

当然，在这里纠结的不是 Solidity 合约怎么写或者合约代码大小怎么计算的问题，后来搞清楚了。比较在意的是，那个时候突然有点恍惚，用接口和直接用合约，有什么区别？

之前给合约定义接口是为了提供一个对外方法的描述，这里才意识到接口可以替代合约本身，直接用来定义变量，并且使用接口定义的变量，去调用合约里面的方法。但为什么可以呢，它不就只是一个接口吗？

2

如果你刚学习过 Java，或者使用 Java 作为工作语言，一定会有哑然失笑的感觉，这个问题太幼稚了，这不就是多态吗？

上第一节 Java 的课程，老师就告诉我们，面向对象有三大特性，封装、继承、多态，这句话时至今日我都能想起来，这是多么基础的概念，结果在工作多年后的今天，我竟然在实际工作上因为如此简单的问题犯了难，一时没反应过来，用接口作为类型的写法是什么意思。这太荒唐了。可能也是因为很久没写 Java，现在一直在用 Golang。

不得不说 Java 是面向对象编程语言的标杆，Solidity 虽然是一种看似新的用于智能合约的脚本语言，揉杂了多种语言的特性，但基本的编程思想还是基于面向对象的。合约就是类，部署一个合约就是实例化了一个对象，合约地址就是对象的内存地址，合约调用就是对象的方法调用……

只要是支持面向对象的编程语言，就包含有面向对象的特性，就可以使用面向对象的写法，就离不开最基本的像多态一样的特性。从面向对象的角度去理解，Solidity 有什么难的呢？无非不就是换了一些表面上的形式，编程思路甚至可以一模一样，此外再添上一些区块链特有的概念，像转账、块高度之类，就没了。

从编程语言的角度看，Solidity 和 Java 那样成熟的语言自然没法比，面向对象的特性是残缺的，modifier、require 之类的写法看似好用却增加了很多理解成本，而且代码结构也变得不是太统一。EVM 怎么能和 JVM 相提并论呢？但作为一种轻量级的脚本语言，Solidity 又要使用静态类型那样冗余的写法。

当然要注意，编程思想是先于编程语言的，我仍然会认为形式上的编程语言不值得学习，但是不否认从学习编程语言的角度入手去学习编程思想。比如多态这个概念，含义是使用统一的符号去代表不同的类型，包括三种类型的解释，一是支持多种类型的参数，对应 Java 里方法的重载，二也是支持多种类型的参数，对应 Java 里的范型，三是子类型，也就是把接口作为类型，对应上面提到的场景中的多态的含义。

面向对象是一种编程思想，包含很多计算机科学的概念，而 Java 是一种完全的面向对象的编程语言，不但涵盖众多有用的特性，而且实现的完整漂亮，如果你学习了 Java，自然也就知道面向对象是怎么回事了，受用无尽。从这个角度看，和 Java 相比，Golang 有什么值得学习的地方吗？是 struct 的写法还是 * 号的用法？可能 Golang 更像是一种快餐式的语言吧，可以很方便地 go func()。不过要是为了学习，就不是太推荐了。

花了几分钟看 Java 文档的目录，倒是能很快想起来那些内容，毕竟实在是太基础了。也是要告诫自己，别忘了代码怎么写。