Shenzhen Shilian Information Technology Co., Ltd.

深圳市识链信息科技有限公司

智能合约开发入门第一步：正确认识并理解合约代码的含义

智能合约开发第
一步，在于要认识与理解合约代
码的含义。接下来
让我们先看一下最基本的例子。现在就算你都不理解也不要紧，后面我们会有更深入的讲解。

1、存储合约示例

把一个数据保存到链上

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity>=0.4.16<0.9.0;


contractSimpleStorage{

uintstoredData;


functionset(uintx)public{

storedData=x;

}


functionget()public view returns(uint){

returnstoredData;
```

}

}

第一行是说明源代码是根据GPL 3.0版本授权的。默认情况下，在发布源代码时加入机器可读许可证说明是很重要的，

下一行是告诉编译器源代码所适用的Solidity版本为>=0.4.16 及 pragma 是告知编译器如何处理源代码的通用指令（例如，pragma once）。

Solidity中智能合约的含义就是一组
代码（它的功能)和数据（它的状态
）的集合，并且它
们是位于以太坊区块链的一个特定地
址上的。uintstoredData;
这一行代码声明了一个名为``storedData
``的状态变量，其类型为uint(256位无符号整数）。 你也可以认为它是数据库里的一个插槽，并且可以通过调用管理数据库代码的函数进行查询和更改。在这个例子中，上述的合约定义了``set``和``get`` 函数，可以用来修改或检索变量的值。

要访问当前合约的成
员（如：状态变量），通常不需要像添加this.
这样的前缀，你只需要通过名字就可以直接访问它。 与其他一些语言不同的是，省略它不仅仅是一个风格问题，因为它是一种完全不同的访问成员的方式，这一块后面会详细介绍。

该合约能完成的事情并不多（由于以太坊构建的基础架构的原因）：它能允许任何人在
合约中存
储一个单独的数字
，并且这个数字可以被世界上任何人
访问，且
没有可行的办
法阻止你发布这个数字。当
然，任何人都可以再次调用set
，传入不同的值，覆盖你的数字，但是这个数字仍会被存储在区块链的历史记录中。随后，我们会看到怎样施加访问限制，以确保只有你才能改变这个数字。

2、货币合约（Subcurrency）示例

下面的合约实现了一个最简单的
加密货币。这里，币确实
可以无中生有地产生，但是只有创建合约的人才能做到（实现一个不同的发行计划也不难）。而且，任何人都可以给其他人转币，不需要注册用户名和密码 —— 所需要的只是以太坊密钥对。

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity^0.8.4;


contractCoin{

// 关键字“public”让这些变量可以从外部读取

addresspublic minter;

mapping(address=>uint)publicbalances;


// 轻客户端可以通过事件针对变化作出高效的反应

eventSent(addressfrom,addressto,uintamount);


// 这是构造函数，只有当合约创建时运行

constructor(){

 minter =msg.sender;

}


functionmint(addressreceiver,uintamount)public{

require(msg.sender== minter);

 balances[receiver]+=amount;
```

```
}

// Errors allow you to provide information about
// why an operation failed. They are returned
// to the caller of the function.
 error InsufficientBalance(uintrequested,uintavailable);


functionsend(addressreceiver,uintamount)public{
if(amount > balances[msg.sender])
 revert InsufficientBalance({
 requested: amount,
 available: balances[msg.sender]
});

 balances[msg.sender]-= amount;
 balances[receiver]+=amount;
 emit Sent(msg.sender, receiver, amount);
}
}
```

这个合约引入了一些新的概念，让我们逐一解读。

```
addresspublicminter;
```

这一行声明了一个可以被公开访问的address类型的状态变量。address
类型是一个160位的值，且不允许任何算数操作。这种类型适合存储合约地址或外部人员的密钥对。关键字public
自动生成一个函数，允许你在这个合约之外访问这个状态变量的当前值。如果没有这个关键字，其他的合约没有办法访问这个变量。由编译器生成的函数的代码大致如下所示（暂时忽略 external 和 view）：

functionminter()external view returns(address){return minter;}

当然，加一个和上面完全一样的函数是行不通的，因为我们会有同名的一个函数和一个变量，这里，主要是希望你能明白——编译器已经帮你实现了。

下一行，mapping(address=>uint)publicbalances;
也创建一个公共状态变量，但它是一个更复杂的数据类型。
该类型将address映射为无符号整数。 Mappings
可以看作是一个哈希表
它会执行虚拟初始化，以使所有可能存在的键都映射到一个字节表示为全零的值。但是，这种类比并不太恰当，因为它既不能获得映射的所有键的列表，也不能获得所有值的列表。 因此，要么记住你添加到mapping中的数据（使用列表或更高级的数据类型会更好），要么在不需要键列表或值列表的上下文中使用它，就如本例。 而由public关键字创建的getter函数getter function则是更复杂一些的情况，它大致如下所示：

functionbalances(addressaccount)external view returns(uint){

return balances[account];

}

正如你所看到的，你可以通过该函数轻松地查询到账户的余额。

eventSent(addressfrom,addressto,uintamount);
这行声明了一个所谓的“事件
（event）”，它会在send
函数的最后一行被发出。用户界面（当然也包括服务器应用程序）可以监听区块链上正在发送的事件，而不会花费太多成本。一旦它被发出，监听该事件的listener都将收到通知。而所有的事件都包含了from，to和amount
三个参数，可方便追踪交易。
为了监听这个事件，你可以使用如下JavaScript代码（假设 Coin

是已经通过web3.js 创建好的合约对象）：

```
Coin.Sent().watch({},'',function(error,result){

if(!error){

console.log("Coin transfer: "+result.args.amount+

" coins were sent from "+result.args.from+

" to "+result.args.to+".");

console.log("Balances now:\n"+

"Sender: "+Coin.balances.call(result.args.from)+

"Receiver: "+Coin.balances.call(result.args.to));

}

})
```

这里请注意自动生成的balances函数是如何从用户界面调用的。

特殊函数constructor
是仅在创建合约期间运行的构造函数，不能在创建之后调用。 在 Coin
合约中，构造函数永久存储创建合约的人的地址:msg(类似的还有tx和block)
是一个特殊的全局变量，
参考特殊变量和函数，这些变量允许我们访问区块链的属性。
msg.sender始终记录当前（外部）函数调用是来自于哪一个地址。

最后，真正被
用户或其他合约所调用的，以完成本合约功能的方法是mint和send。

mint
函数用来新发行一定数量的币到一个地
址。require用来检查某些条件，如果不满足这些条件就会回推所有的状态变化。
在这个例子中,require(msg.sender==minter);
确保只有合约的创建者可以调用mint。

一般来说，创建者可以随心所欲地铸造代币，但在某些时候，这将导致一种叫做“溢出” 的现象。

请注意，由于默认的算术检查模式，如果表达式
balances[receiver]+=amount;溢出交易将被还原。
即当任意精度算术中的balances[receiver]+amount大于uint(2**256-1
)。同样在在函数send中的balances[receiver]+=amount;
这对语句来说也是如此。

Errors用来向调用者描述错误信息。Error与revert
语句一起使用。revert
语句无条件地中止执行并回退所
有的变化，类似于require
函数，它也同样允许你提供一个错误的名称和额外的数据，这些额外数据将提供给调用者(并最终提供给前端应用程序或区块资源管理器），这样就可以更容易地调试或应对失败。

任何人（已经拥有一些代币）都可以使用send
函数来向其他人
发送代币。如果发送者没有足够
的代币可以发送，if条件为真revert将触发失败，并通过
InsufficientBalance向发送者提供错误细节。

如果mint被合约创建者外的其他人调用则什么也不会发生。
另一方面，send函数可被任何人用于向他人发送币
(当然，前提是发送者拥
有这些币)。记住，如果你使用合约发送币给一个
地址，当你在区块链浏览器上查看该地址时是看不到任何相关信息的。因为，实际上你发送币和更改余额的信息仅仅存储在
特定合约的数据存储器中。通过使用事件，你可以非常简单地为你的新币创建一个“区块链浏览器”来追踪交易和余额。

国内领先的Web3.0倡导者