interface 关键字：定义了一个接口。其他合同可以实现这个接口。

智能合约之间通过 interface 定义的方法互相调用

一、简单的 interface 和其实现。

下面的合约实现了 interfaceContract 的接口。

```
pragma solidity ^0.4.16;

interface interfaceContract {
    function receiveApproval(address _from, uint256 _value, address _token, bytes _extraData);
}

contract InterfaceImplContract is interfaceContract {
    event Receive(address from, uint256 value, address token, bytes extraData);
    function receiveApproval(address _from, uint256 _value, address _token, bytes _extraData) {
        Receive(_from,_value,_token,_extraData);
    }
}
```

这里部署合约时，只需部署 InterfaceImplContract 即可，不用管接口。接口只是为了声明。如下：

二、调用接口。

下面的合约，想与实现了 interfaceContract 接口的合约进行交互。

```
pragma solidity ^0.4.16;

interface interfaceContract {
    function receiveApproval(address _from, uint256 _value, address _token, bytes _extraData);
}

contract RemoteContract {
    function func(address _addr, uint _value) {
```

```
        //注意这里的_addr 参数，需要填写 tokenRecipient 合约的地址。这里加载已经存在的智能合约。如何合约不存在会报错回滚。
        interfaceContract _interfaceContract = interfaceContract(_addr);
        _interfaceContract.receiveApproval(msg.sender, _value, address(this), "这是一些信息");
    }
}
```

- 注意这里调用 **func(address _addr, uint _value)**方法时，传递的参数_addr，在下面的代码中，用来加载合约 **interfaceContract _interfaceContract = interfaceContract(_addr);** 所以 addr 必须传递合约地址。并且这个合约地址是 **interfaceContract** 的实现类的合约地址。也就是第一步创建的 InterfaceImplContract 合约的地址。
- 如果传递的**_addr** 参数错误，调用失败。它将回滚所有已执行的功能。也就是这个方法会回滚。
- 这里部署时，只需部署 RemoteContract 即可。不用管接口。接口只是为了声明。

三、实际应用。

在官方代币合约中。就有实际的引用。如下：

```
pragma solidity ^0.4.16;

interface tokenRecipient { function receiveApproval(address _from, uint256 _value, address _token, bytes _extraData) public; }

contract TokenERC20 {
    // 代币名
    string public name;
   // 代币 logo
    string public symbol;
    // 代币小数位数
    uint8 public decimals = 18;
    // 18 位小数是强烈建议的默认值，避免改变它
    uint256 public totalSupply;

    // 这将创建一个包含所有余额的数组
    mapping (address => uint256) public balanceOf;
    //某人授权给某人，使用自己的多少代币.
    //比如：当前调用者 msg.sender，可以授权给很多地址他代币的使用权限。
    mapping (address => mapping (address => uint256)) public allowance;

    // 代币转移日志
    event Transfer(address indexed from, address indexed to, uint256 value);

    // 销毁代币日志
    event Burn(address indexed from, uint256 value);

    /**
     * 构造方法
```

```
29.         */
30.     function TokenERC20(uint256 initialSupply,string tokenName,string tokenSymbol) public {
31.         //10 ** uint256(decimals)是 10^18 次方。
32.         totalSupply = initialSupply * 10 ** uint256(decimals);  //初始化代币总数  使用 decimal，两个星号**表示次方。
33.         balanceOf[msg.sender] = totalSupply;                // 给创建者所有的初始化代币
34.         name = tokenName;                                   // 设置代币显示的名字
35.         symbol = tokenSymbol;                               //设置代币显示的符合，代币 logo
36.     }
37. 
38.     /**
39.      * 代币转移, internal 只能合约内部调用
40.      */
41.     function _transfer(address _from, address _to, uint _value) internal {
42.         // 防止传输到 0x0 地址。 使用 burn（）来代替
43.         require(_to != 0x0);
44.         // 检查发送方有足够代币
45.         require(balanceOf[_from] >= _value);
46.         // 防止溢出，超过 uint 的数据范围，会变为负数
47.         require(balanceOf[_to] + _value > balanceOf[_to]);
48.         // 保存以备将来的断言
49.         uint previousBalances = balanceOf[_from] + balanceOf[_to];
50.         // 发送方减掉代币
51.         balanceOf[_from] -= _value;
52.         // 接收方增加代币
53.         balanceOf[_to] += _value;
54.         Transfer(_from, _to, _value);//event 日志
55.         // 断言用于使用静态分析来查找代码中的 bug。 他们永远不会失败
56.         assert(balanceOf[_from] + balanceOf[_to] == previousBalances);
57.     }
58. 
59.     /**
60.      * Transfer tokens
61.      *
62.      * @param _to 收币方地址
63.      * @param _value 转移多少代币
64.      */
65.     function transfer(address _to, uint256 _value) public {
66.         _transfer(msg.sender, _to, _value);
67.     }
68. 
69.     /**
70.      * 从其他地址转移代币，需要其他地址授权给调用的人。
71.      *
72.      */
73.     function transferFrom(address _from, address _to, uint256 _value) public returns (bool success) {
```

```
74.             require(_value <= allowance[_from][msg.sender]);        // 检查授权额度
75.             allowance[_from][msg.sender] -= _value;
76.             _transfer(_from, _to, _value);
77.             return true;
78.         }
79.
80.         /**
81.          * 允许其他人，花费我的代币。
82.          *
83.          *授权给_spender 地址，_value 个代币
84.          *
85.          * @param _spender 授权给_spender 地址
86.          * @param _value 代币数
87.          */
88.         function approve(address _spender, uint256 _value) public returns (bool success) {
89.             allowance[msg.sender][_spender] = _value;
90.             return true;
91.         }
92.
93.         /**
94.          * Set allowance for other address and notify
95.          *
96.          * 允许 `_spender` 代表你花费不大于`_value` 个代币, and then ping the contract about it
97.          *
98.          * @param _spender 授权给哪个地址
99.          * @param _value 授权最大代币数量
100.                 * @param _extraData 一些额外的信息发送到批准的合同
101.                 */
102.             function approveAndCall(address _spender, uint256 _value, bytes _extraData)public returns (bool success)
    {
103.                 tokenRecipient spender = tokenRecipient(_spender);
104.                 if (approve(_spender, _value)) {
105.                     spender.receiveApproval(msg.sender, _value, this, _extraData);
106.                     return true;
107.                 }
108.             }
109.
110.             /**
111.              * 销毁代币
112.              *
113.              * 从系统中不可逆地删除`_value'个代币
114.              */
115.             function burn(uint256 _value) public returns (bool success) {
116.                 require(balanceOf[msg.sender] >= _value);     //检查调用者金额大于销毁数量
117.                 balanceOf[msg.sender] -= _value;                  // 调用者代币减掉
118.                 totalSupply -= _value;                                // 总供应量减掉
```

```
119.            Burn(msg.sender, _value);                   // event 日志
120.            return true;
121.        }
122.
123.        /**
124.         * 从其他账户销毁代币
125.         *
126.         *
127.         * @param _from 从哪个账户销毁代币
128.         * @param _value 销毁的代币数
129.         */
130.        function burnFrom(address _from, uint256 _value) public returns (bool success) {
131.            require(balanceOf[_from] >= _value);          // Check if the targeted balance is enough
132.            require(_value <= allowance[_from][msg.sender]);       // Check allowance
133.            balanceOf[_from] -= _value;                           // Subtract from the targeted balance
134.            allowance[_from][msg.sender] -= _value;               // Subtract from the sender's allowance
135.            totalSupply -= _value;                                 // 减掉总供应量
136.            Burn(_from, _value);                                  // event 日志
137.            return true;
138.        }
139.    }
```

这里的 **approveAndCall** 方法，就调用了接口。把自己的代币授权给某个合约接口。并且调用合约的后续处理方法。执行通知。