

Block chain

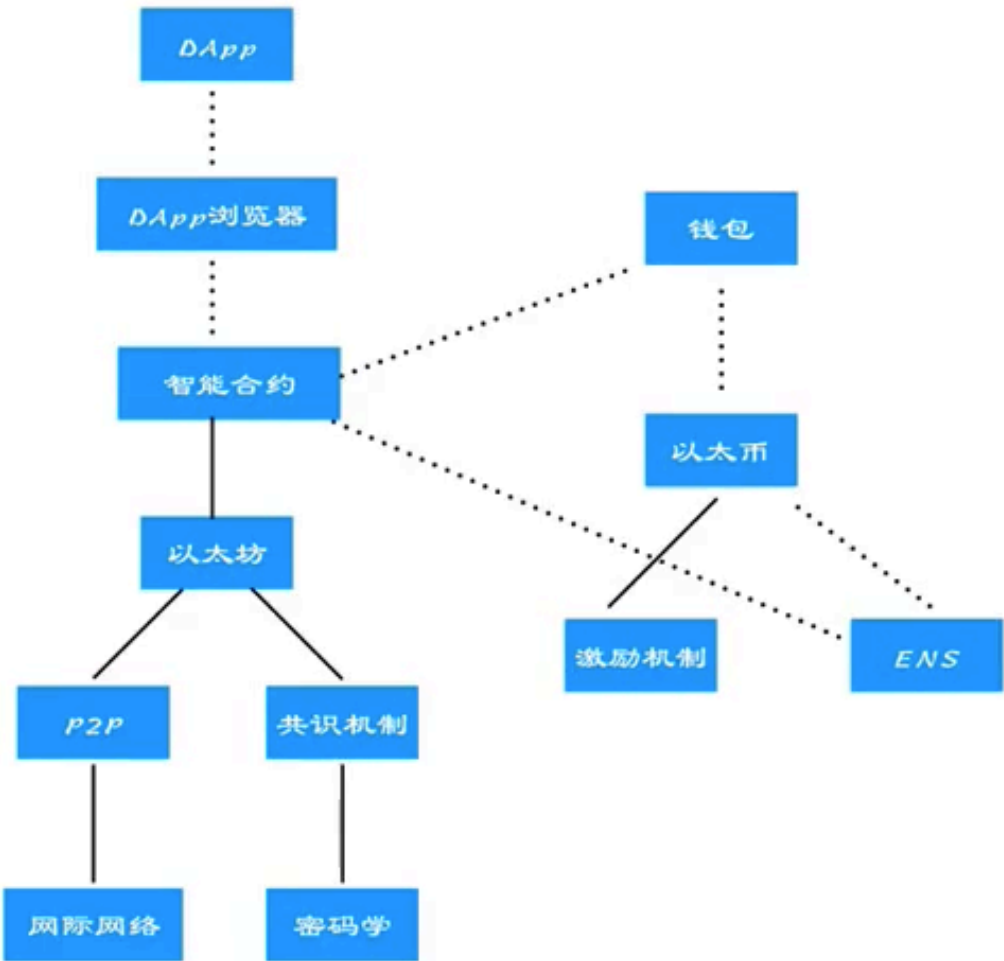
- 区块链
 - 区块头
 - 区块链网络需要的构造数据
 - 区块体
 - 存放有价值的数据

区块链基础

区块链的基础技术架构



以太坊生态



以太币（Ether）的实际用途

在电脑上执行一般的代码，写出来可能会有循环不断执行（死循环）的情况，搬到区块链上亦然。那么以太坊怎么解决这个问题呢？以太坊开发者想到的办法是让执行的代码变得有价。EVM里支持的所有的指令（OPCODE）都有明码标价，执行智能合约需要消耗与执行的指令数量相当的以太币。在智能合约的术语中，这些拿来消耗的以太币被称为**gas**（燃料）。部署合约到区块链上时，需要附加一定数量的燃料。当燃料消耗完而程序还没执行完，就会出现**Out of Gas**（燃料耗尽）错误。智能合约透过这样的方式来避免死循环等情况。

智能合约能做什么事？

创造新的加密货币

智能合约最基本的应用，就是做出新的加密货币！只要遵循一定的规范（ERC20），部署一份智能合约到以太坊区块链上，任何人都可以很容易地创造出自己的加密货币。

不用像比特币的山寨币一样需要自行征募矿工。你的加密货币会以“以太坊的一个智能合约”的形式存在于以太坊的区块链中。现有支持以太币的钱包与交易所，也可以快速地接入你的加密货币。只要你发行的加密货币与代币背后所代表的服务具有交易的价值，代币也可能变货币。

募资

此外，在募资上，智能合约也有无与伦比的优势。新创公司ICO可透过以太坊的智能合约，在收到资助者的以太币时，自动发出等值的加密货币。

由于这些加密货币可交易，如果公司的服务有价值，早期投资人在投资的前期，就可以透过交易加密货币来调整持有量。

普通的投资人也第一次有机会直接支持自己感兴趣的创业者，并可能从而获得丰厚（或归零）的回报。

程序能做的事

当你需要公开，可被信任的纪录时，都可以透过智能合约达成。

以太坊的缺点

目前以太坊区块链的速度和电脑执行速度无法相比，不适合快速交易，或是需要储存较大数据的情境。近期的plasma提案有望解决快速交易的问题。

因为缺乏即时调控区块大小的手段，在一些很热门的交易时段（如某些热门的ICO开放认购时），整个网络的交易延迟会变地很长。

智能合约一经部署就永远存在，除非拥有者启动智能合约中的自毁（selfdestruct）功能。如何升级合约并保存其中的参数与代币，是一个值得探讨的课题。



以太坊网络节点上并不适合储存较大的档案。目前有Swarm与IPFS等分布式档案储存方式可供选择。

其他参考资料

智能合约（区块链程序）

- 区块链中的应用层

智能合约是什么？

在区块链上运行的程序，通常称为智能合约（Smart Contract）。所以通常会把写区块链程序改称写智能合约。虽然比特币（Bitcoin）上也能写智能合约，但是比特币所支持的语法仅与交易有关，能做的事情比较有限。因此目前提到写智能合约，通常指的是支持执行图灵完备程序的以太坊（Ethereum）区块链。

智能合約可以做什麼？

目前最常见的智能合约是各种加密货币合约📄，开发者可以很容易地透过部署一个智能合约，来提供运行于以太坊上的新加密货币。如果这份智能合约相容于ERC20标准1，开发者不需要重新开发从挖矿到交易的整个代币生态系，你的新加密货币就可以直接使用支持以太坊的电子钱包💰来收送，大大降低了建立新加密货币的门槛。

智能合约也可以用来运作各种公开公正的自动服务机构（DAO，权力下放自治组织）🏛️。透过分散在全球各节点上运作的智能合约，所有运作与决策都是公开透明的，降低了交易的不确定性（不确定性）。

智能合约和一般程序程序的差异

以太坊智能合约确实有些和一般程序不同的特性，以下整理了四个不同点。

一、整合金流容易

一般的应用程序要整合金流是件非常不容易的事情而智能合约极容易整合金流系统（使用以太币或自行建立的新代币合约）。

二、部署时与后续写入时需费用

一般的应用程序需要提供网址让使用者下载，一般的网页应用程序也需要运行在伺服器上，开发者需要维持伺服器的运作以提供服务，这需要持续地花费（就算是免费的伺服器或网页空间，也是厂商自行吸收了费用），程序开始运作后，除了维持费用外不需额外的花费。

智能合约在部署时需要一笔费用，这笔费用将分给参与交易验证（挖矿）的人。而在合约部署成功后，合约会作为不可更改的区块链的一部分，分散地储存在全球各地以太坊的节点上。也因此，智能合约在部署后，并不需定期提供维持费用，同时查询已写入区块链的静态资料时也不需费用。只有在每次透过智能合约写入或读取计算结果时，需要提供一小笔交易费用。

三、储存资料的成本更高

一般的应用程序将资料储存在本机或伺服器上，需要资料时再从本机或伺服器上读取，而智能合约将资料储存在区块链上，储存资料所需的时间与成本相对昂贵。

四、部署后无法更改

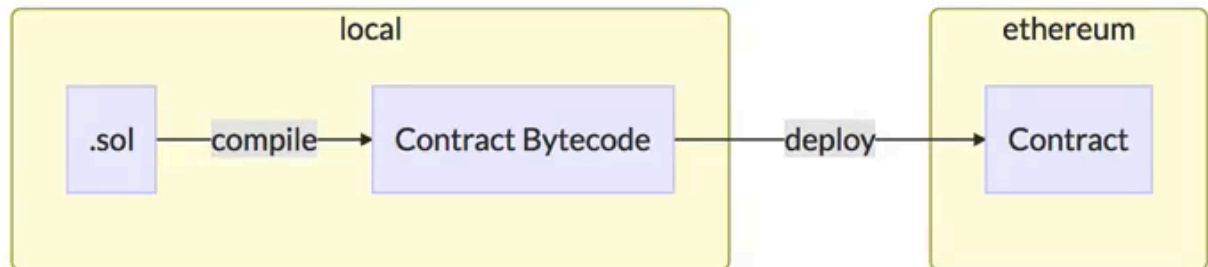
一般的应用程序改版时可透过安装新版程序，网页应用程序也可透过部署新版程序达成，而智能合约一旦部署到区块链上后，就无法更改这个智能合约。当然聪明的开发者透过加入额外的智能合约，也已有办法绕过智能合约部署后无法再更改的限制。

如何编写智能合约？

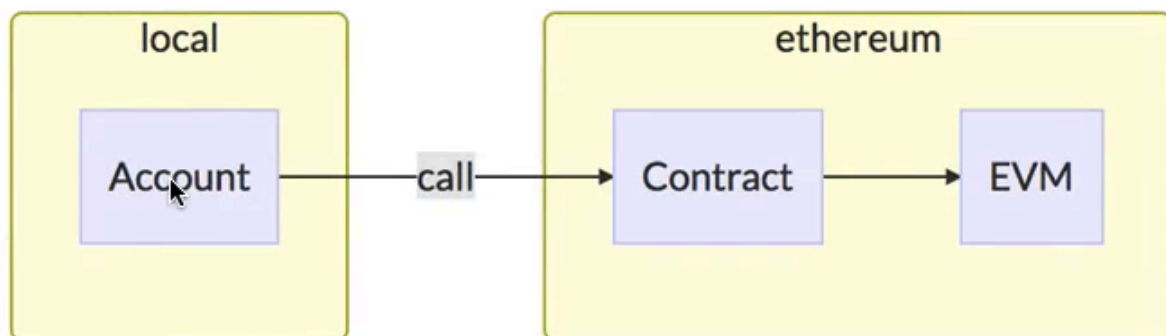
Ethereum 上的智能合约需要使用 solidity 语言来编写。之前还有其他能用来编写智能合约的语言如 Serpent（类的Python），LLL（类的Fortran），但目前看到所有公开的智能合约都是使用 solidity 编写。官方宣传上说 solidity 是一种类似的 JavaScript 的语言，而且围绕着 JavaScript 的各种开发工具链都是使用属于使用Javascript生态系的NPM来提供的。

将智能合约部署到区块链的流程

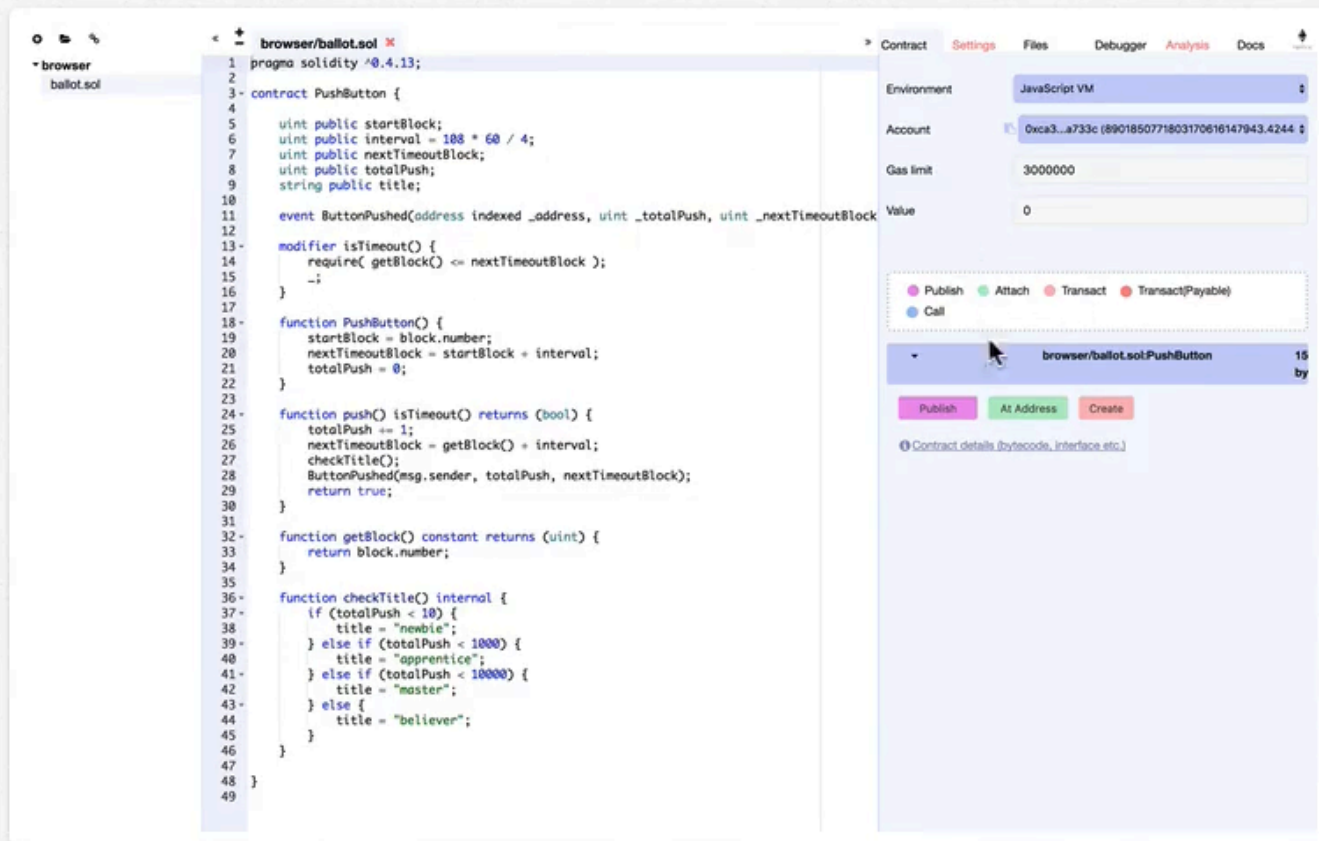
写好 solidity 代码 (.sol) 后，需要先将程序代码编译（编译）成EVM（Ethereum Virtual Machine）能读懂的二进制度 Contract ByteCode，才能部署到Ethereum的区块链上执行。部署到区块链上的合约会有一个和钱包地址（地址）一样格式的合约地址（Contract Address）。



部署后智能合约可自动执行。后续呼叫智能合约的时候，使用者可以使用部署合约的钱包地址（所有者帐户），或依据编写的智能合约条件，让其他钱包地址也能呼叫这个智能合约。呼叫智能合约，其实就是向这个合约地址发起交易，只是交易的不只是代币，而可以是智能合约提供的呼叫方法。



智能合约范例



编辑器 atom

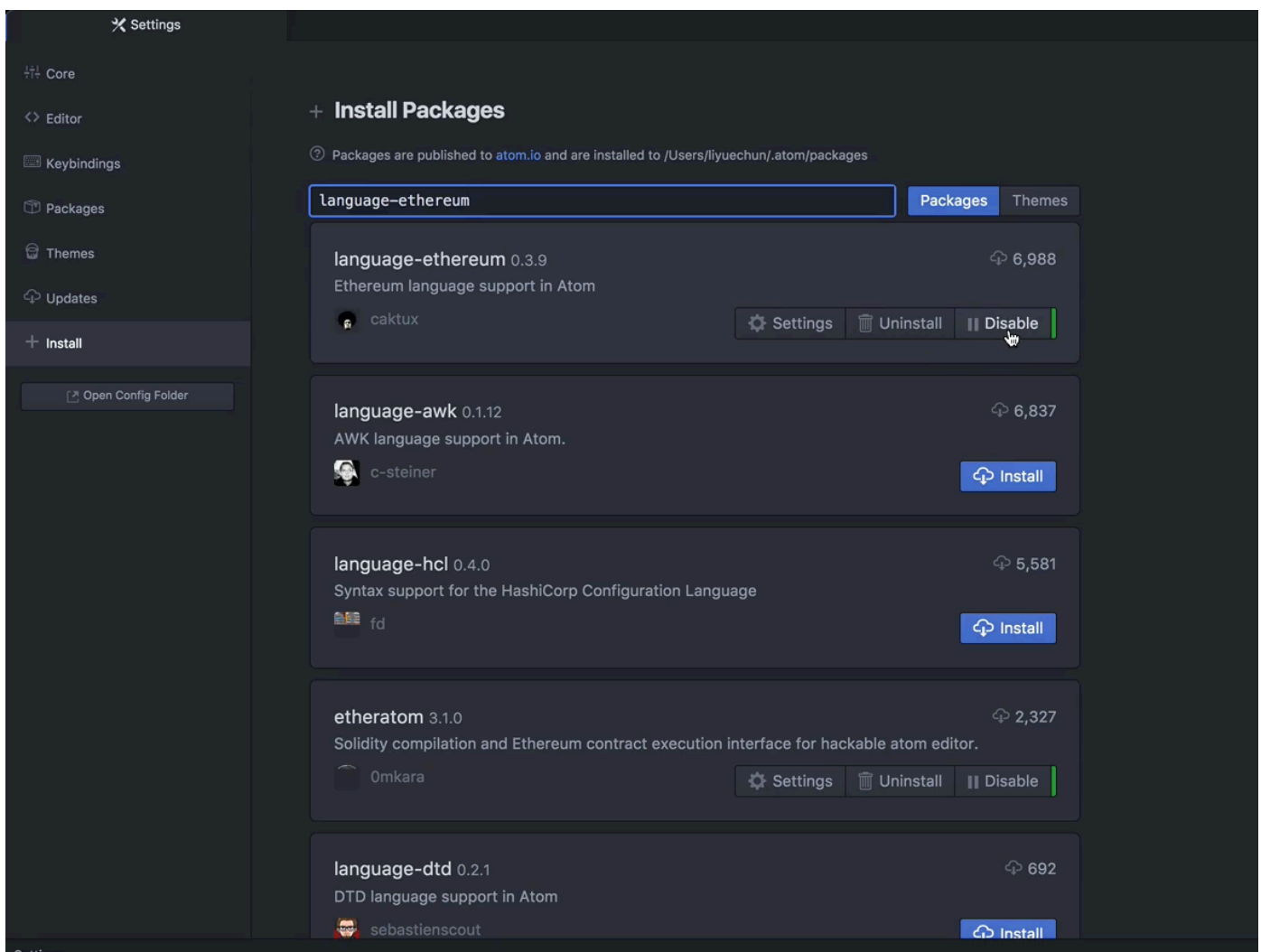
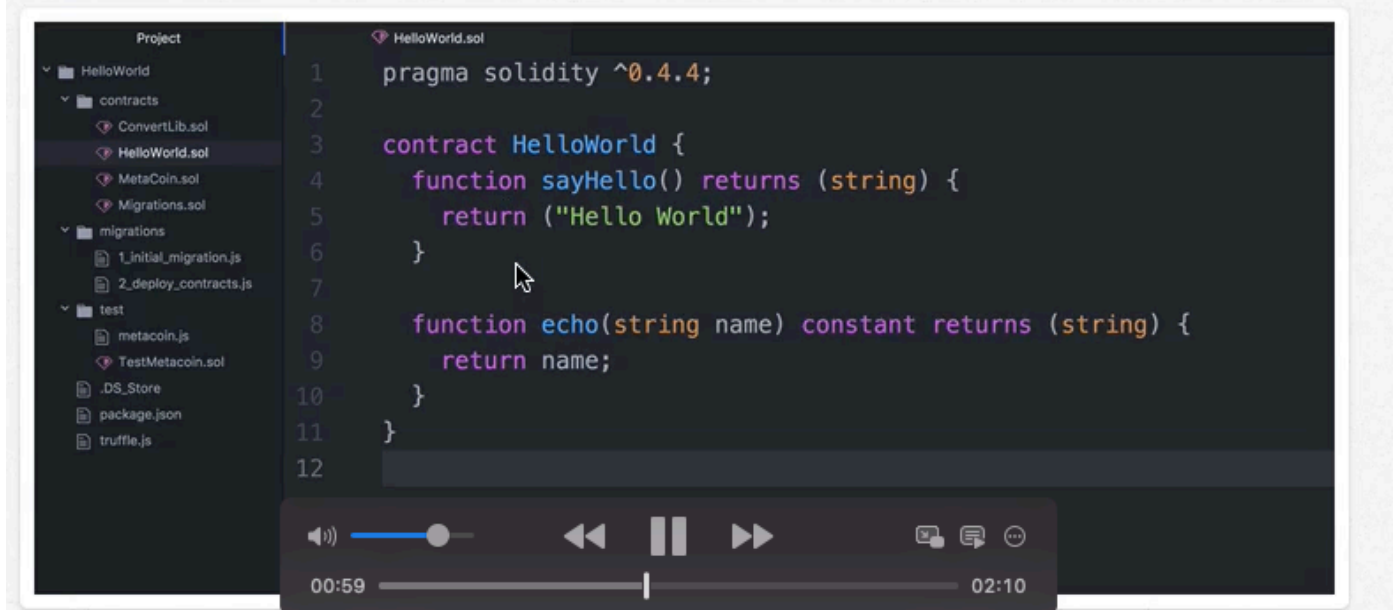
编辑器选择

理论上讲任何编辑器都可以编写 Solidity 合约代码，比如：WebStorm，VSCode，Sublime，等等。我选择的是Atom，没有任何理由，因为Atom轻量并且界面漂亮。

- 移步<https://atom.io/>地址，下载安装Atom。
- `autocomplete-solidity` 代码自动补齐



- `linter-solium`、`linter-solidity` 代码错误检查
- `language-ethereum` 支持 Solidity 代码高亮以及 Solidity 代码片段



面向对象

```
// 定义属性
int a;
int b;
int c;

// 实例化得到counter对象
Counter counter = new Counter;
```

solidity

- <http://remix.hubwiz.com/#optimize=false&version=soljson-v0.5.1+commit.c8a2cb62.js> 写完的代码都在此处执行测试

合约

- 类: 合约
 - set: set + 属性名 修改属性的方法
 - get: 属性名作为方法名

```
pragma solidity ^0.4.4;

/*
pragma: 声明版本
solidity: 开发语言
0.4.4: 当前合约的版本, 0.4代表主版本, .4代表修复bug的升级版本
^: 代表向上兼容, 0.4.4 ~ 0.4.9 可以对我们当前的代码进行编译
*/

// contract Person 类
contract Person {
    uint _height; // 身高
    uint _age; // 年龄
    address _owner; // 合约持有者

    // 方法名和合约名相同时就属于构造函数, 在创建对象时, 构造函数会先调用执行
    function Person(){
        _height = 180;
        _age = 29;
        _owner = msg.sender;
    }

    // 返回合约地址
    function owner() constant returns(address){
```

```

    return _owner;
}

// set方法, 修改height属性
function setHeight(uint height){
    _height = height;
}

// get方法, 读取height信息 (不需要gas费)。constant 代表只读
function height() constant returns (uint){ // returns 返回值
    return _height;
}

function setAge(uint age){
    _age = age;
}

function age() constant returns (uint){
    return _age;
}

// kill方法, 对象销毁
function kill() constant{
    // selfdestruct(msg.sender); // selfdestruct(合约地址) 销毁当前合约地址
    if (_owner == msg.sender){
        selfdestruct(_owner);
    }
}
}

```

solidity中合约属性的访问权限

属性的访问权限

- public
- internal
- private

说明: 属性默认类型为internal, internal和private类型的属性不能被外部访问, 当属性类型为public时, 会生成一个属性名相同并返回当前属性的get函数。

```
uint public _money
```

```
pragma solidity ^0.4.4;
```

```
/*
```

```
pragma: 声明版本
```

solidity: 开发语言

0.4.4: 当前合约的版本, 0.4代表主版本, .4代表修复bug的升级版本

^: 代表向上兼容, 0.4.4 ~ 0.4.9 可以对我们当前的代码进行编译

```
*/
```

```
/*
```

solidity属性的访问权限

- public 公开
- internal
- private

```
*/
```

// 定义一个Person合约

```
contract Person{
```

// internal 合约属性的默认访问权限

```
uint internal _age;
```

```
uint _weight;
```

```
uint private _height;
```

```
uint public _money;
```

```
function _money() constant returns (uint){
```

// 当调用_money时返回290, 相当于将_money的值覆盖为290;

```
return 290;
```

```
}
```

```
}
```

方法的访问权限

The screenshot shows the Remix IDE interface. The main editor displays a Solidity contract named `Person` with the following code:

```
pragma solidity ^0.4.4;

/*
pragma: 声明版本
solidity: 开发语言
0.4.4: 当前合约的版本, 0.4代表主版本, .4代表修复bug的升级版本
^: 代表向上兼容, 0.4.4 ~ 0.4.9 可以对我们当前的代码进行编译
*/

// 方法的访问权限
contract Person{
    function age() constant returns (uint) {
        return 35;
    }

    function weight() constant returns (uint){
        return 120;
    }

    function height() constant returns (uint){
        return 190;
    }

    function money() constant returns (uint){
        return 1000000;
    }
}
```

The right sidebar shows the deployment details for the `Person` contract. The node environment is set to `JavaScript虚拟机`. The current account is `0xca3...a733c (99.999999999999999999 wei)`. The gas limit is `3000000` and the transaction amount is `0`. The contract is deployed to the `Person` address. The deployment details table shows the following values:

Variable	Value
age	0: uint256: 35
height	0: uint256: 190
money	0: uint256: 1000000
weight	0: uint256: 120

```
pragma solidity ^0.4.4;
```

```
/*
```

pragma: 声明版本

solidity: 开发语言

0.4.4: 当前合约的版本, 0.4代表主版本, .4代表修复bug的升级版本
^: 代表向上兼容, 0.4.4 ~ 0.4.9 可以对我们当前的代码进行编译
*/

// 方法的访问权限

```
contract Person{  
    function age() constant returns (uint) {  
        return 35;  
    }  
  
    function weight() constant returns (uint){  
        return 120;  
    }  
  
    function height() constant returns (uint){  
        return 190;  
    }  
  
    function money() constant returns (uint){  
        return 1000000;  
    }  
}
```

The screenshot shows the Remix IDE interface. The main editor displays the Solidity code for the 'Person' contract, which includes comments in Chinese explaining the versioning and access control. The right-hand sidebar contains several sections: '节点环境' (Node Environment) set to 'JavaScript虚拟机'; '当前账号' (Current Account) showing '0xca3...a733c'; 'Gas上限' (Gas Limit) set to '3000000'; '交易金额' (Transaction Amount) set to '0'; a dropdown menu for the contract name 'Person'; a '部署' (Deploy) button; a section for '已记录的交易' (Recorded Transactions) showing 2 transactions; a section for '已部署的合约' (Deployed Contracts) showing the 'Person' contract at address '0xbbf...732db'; and a '变量监视器' (Variable Inspector) showing the values of 'age' (35) and 'weight' (120) for the 'Person' contract instance.

- 方法默认权限为public

```
pragma solidity ^0.4.4;  
  
/*  
pragma: 声明版本  
solidity: 开发语言  
0.4.4: 当前合约的版本, 0.4代表主版本, .4代表修复bug的升级版本  
^: 代表向上兼容, 0.4.4 ~ 0.4.9 可以对我们当前的代码进行编译  
*/
```



```

/*
    solidity属性的访问权限
    - public 公开
    - internal
    - private
*/

// 定义一个Person合约
contract Person{
    // internal 合约属性的默认访问权限
    uint _weight;
    uint private _height;
    uint internal _age;
    uint public _money;

    // 方法不声明权限，则默认为public
    function test1() constant returns(uint){
        return _weight;
    }

    function test2() constant public returns(uint){
        return _height;
    }

    function test3() constant internal returns(uint){
        return _age;
    }

    function test4() constant private returns(uint){
        return _money;
    }

    // 通过 this. 指针的方式访问，只能访问public类型的属性与方法
    function testInternal1() constant returns(uint){
        return this.test1();
    }

    function testInternal2() constant returns(uint){
        return this.test2();
    }
}

```

属性与方法合约内部的访问权限总结 (***)

- 属性默认权限为internal，只有public类型的属性外部才能访问。internal和private类型的属性只能在合约内部访问；
- 方法的权限默认是public类型，public类型的方法可供外部访问；而internal和private类型的方法不能够通过指针进行访问，在内部通过this访问也会报错，在合约内部访问，可以直接访问方法。
- **注意:** 不管属性还是方法，只有是public类型时，才能通过合约地址进行访问，合约内部的this就是当前合约地址。在合约内部若要访问internal与private类型的属性或方法，直接访问即可，不要通过this去访问，否则会报错！

合约单继承与多继承

25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64

// 方法不声明权限，则默认为public
function test1() constant returns(uint){
 return _weight;
}

function test2() constant public returns(uint){
 return _height;
}

function test3() constant internal returns(uint){
 return _age;
}

function test4() constant private returns(uint){
 return _money;
}

// 通过 this_ 指针的方式访问，只能访问public类型的属性与方法
function testInternal1() constant returns(uint){
 return this.test1();
}

function testInternal2() constant returns(uint){
 return this.test2();
}
}

contract Dog is Animal{
 // function testHeight() constant returns(uint){
 // return _weight // 因为_weight 是internal 属性
 // }
 // function testHeight() constant returns(uint){
 // return _height // 因为_height 是private 属性
 // }
 function testMoney() constant returns(uint){
 return _money;
 }
}

节点环境JavaScript虚拟机 VM (-) i

当前账号 0xca3...a733c (99.9999999999999985280) ▼

Gas上限3000000

交易金额0 wei ▼

Animal

部署或者合约地址 载入部署在这个地址的合约

已记录的交易: 0

已部署的合约目前没有可以交互的合约实例。

0

121 Remix交易、脚本

搜索交易

- 合约继承属性与方法的权限访问
 - 子合约只能继承父合约的 public 与 internal类型的属性，以及只能继承public类型的方法…

```
pragma solidity ^0.4.4;
```

/*

pragma: 声明版本

solidity: 开发语言

0.4.4：当前合约的版本，0.4代表主版本，.4代表修复bug的升级版本

^: 代表向上兼容, 0.4.4 ~ 0.4.9 可以对我们当前的代码进行编译

 $\ast/$

/*

solidity属性的访问权限

- public 公开
- internal
- private

 $\ast/$

```
// 定义一个Person合约
```

```
contract Animal{
```

```

// internal 合约属性的默认访问权限
uint _weight;
uint private _height;
uint internal _age;
uint public _money;

// 方法不声明权限, 则默认为public
function test1() constant returns(uint){
    return _weight;
}

function test2() constant public returns(uint){
    return _height;
}

function test3() constant internal returns(uint){
    return _age;
}

function test4() constant private returns(uint){
    return _money;
}

// 通过 this. 指针的方式访问, 只能访问public类型的属性与方法
function testInternal1() constant returns(uint){
    return this.test1();
}

function testInternal2() constant returns(uint){
    return this.test2();
}

}

contract Animal2{
    uint sex;
    // 构造函数
    function Animal2(){
        sex = 1;
    }
}

contract Dog is Animal{ // 单继承
    function testWeight() constant returns(uint){
        return _weight;
    }
    // function testHeight() constant returns(uint){
    //     return _height // 因为_height 是private 属性

```

```

    // }

    function testMoney() constant returns(uint){
        return _money;
    }
}

contract Dog2 is Animal, Animal2{ // 多继承
    function testWeight() constant returns(uint){
        return _weight;
    }
    // function testHeight() constant returns(uint){
    //     return _height // 因为_height 是private 属性
    // }
    function testMoney() constant returns(uint){
        return _money;
    }
}
}

```

引用传递

browser/ballot_test.sol

```

1 pragma solidity ^0.4.4;
2
3 /*
4  * pragma: 声明版本
5  * solidity: 开发语言
6  * 0.4.4: 当前合约的版本, 0.4代表主版本, .4代表修复bug的升级版本
7  * ^: 代表向上兼容, 0.4.4 ~ 0.4.9 可以对我们当前的代码进行编译
8  */
9
10 /*
11  * solidity属性的访问权限
12  * - public 公开
13  * - internal
14  * - private
15  */
16
17 // 定义一个Person合约
18 contract Person{
19
20     string _name;
21
22     function Person(string name){
23         _name = name;
24     }
25
26     function f(){
27         modify(_name);
28     }
29
30     // function modify(string name){ 值传递
31     //     name = "Tank";
32     // }
33
34     function modify(string storage name) internal { // storage 设置 name 为引用传递, 传递的是指针
35         bytes(name)[0] = "T"; // 修改一个字符
36     }
37
38     function name() constant returns (string){
39         return _name;
40     }
41 }

```

编译 运行 分析 测试 调试器 设置

节点环境 JavaScript虚拟机 VM (i)

当前账号 0xca3...a733c (99.99999999999973685 wei)

Gas上限 3000000

交易金额 0 wei

Person (i)

部署 string name

或者

合约地址 载入部署在这个地址的合约

已记录的交易: 2

已部署的合约

Person at 0x692...77b3a (memory)

f

name

0: string: Tankjam

值传递