由于`Solidity`是一个`静态类型`的语言，所以编译时需明确`指定变量的类型`（包括`本地变量`或`状态变量`），`Solidity`编程语言提供了一些基本类型(elementary types)可以用来组合成复杂类型。

值类型(Value Type)

先来看看有哪些类型属于值类型

- `布尔(bool)`
- `整型(int/uint)`
- `地址(Address)`
- `固定长度字节数组(fixed byte arrays)`
- `枚举类型(Enums)`
- `函数(Function Types)`

有其他语言开发经验的童鞋都知道，值类型传值时，会临时拷贝一份内容出来，而不是拷贝指针，当你修改新的变量时，不会影响原来的变量的值。

例如：

```
int a = 100;  // a == 100
int b = a;   // b == 100,a == 100
b = 300;    // b == 300,a == 100
```

由上面的数据看，执行`b = a`时，会将`a`的值临时拷贝一份传给`b`，所以当你修改`b`时，其实与`a`没任何关系。

一、布尔

`bool`：只有俩种取值`true`和`false`。

支持的运算符：

- `!` 逻辑非
- `&&` 逻辑与
- `||` 逻辑或
- `==` 等于
- `!=` 不等于

```
bool a = true;
bool b = !a;

// a == b -> false
// a != b -> true
// a || b -> true
// a && b -> false
```

二、整型

`int/uint`：各种大小的有符号和无符号整数。 关键字`uint8`到`uint256`的步幅是8（无符号的8到256位）和`int8`到`int256`。 `uint`和`int`分别是`uint256`和`int256`的别名。

有符号整型（int）可以表示任何规定范围内的整数。

无符号整型只能表示非负数（**0及正数**）。

如果用二进制表示：

- **uint8**: **0b**`00000000` ~ **0b**`11111111`，每一位都存储值，范围为**0 ~ 255**
- **int8**：**0b**`11111111` ~ **ob**`01111111`，最左一位表示符号，`1`表示`负`，`0`表示`正`，范围为**-127 ~ 127**

支持的运算符

- 比较：`<=`，`<`，`==`，`!=`，`>=`，`>`，返回值为`bool`类型。
- 位运算符：`&`，`|`，（`^`异或），（`~`非）。
- 数学运算：`+`，`-`，一元运算`+`，`*`，`/`，（`%`求余），（`**`次方），（`<<`左移），（`>>`右移）。

`**`为次方运算，如`uint c = 3**4; //c=81`;

三、地址

address：地址，一般用作保存账户地址，Ethereum地址的大小是一个20字节的值，不同账户之间的交易实际上就是地址之间的交易。

地址类型也有成员，并作为所有合约的基础。

- `balance`：查询地址的余额。

```
pragma solidity ^0.4.24;

contract AddressBalance{
    function getBalance(address addr) constant returns (uint){
        return addr.balance;
    }
}
```

- `transfer`：从合约发起方向某个地址转入以太币(单位是wei)。

```
pragma solidity ^0.4.24;

contract TransferDemo{
    address account;

    function TransferDemo(address addr) public {
        account = addr;
    }

    // 从合约发起方向 account 地址转入 msg.value 个以太币，单位是 wei
    function deposit() payable{
        account.transfer(msg.value);
        // account.send(msg.value);
    }

    // 读取 account 地址的余额
    function getAccountBalance() constant returns (uint) {
        return account.balance;
    }

    // 读取合约发起方的余额
    function getOwnerBalance() constant returns (uint) {
        address Owner = msg.sender;
        return Owner.balance;
    }
}
```

- `send`

send是低级对等的转账。 如果执行失败，当前合同将不会以异常方式停止，`send`不会抛出异常，但发送将返回false。

```
pragma solidity ^0.4.24;

contract TransferDemo{
    address account;

    function TransferDemo(address addr) public {
        account = addr;
    }

    // 从合约发起方向 account 地址转入 msg.value 个以太币，单位是 wei
    function deposit() payable{
        // account.transfer(msg.value);
        account.send(msg.value);
    }

    // 读取 account 地址的余额
    function getAccountBalance() constant returns (uint) {
        return account.balance;
    }

    // 读取合约发起方的余额
    function getOwnerBalance() constant returns (uint) {
        address Owner = msg.sender;
        return Owner.balance;
    }
}
```

- `call`, `callcode` and `delegatecall`

  此外，为了与不遵守ABI的合同进行接口，提供了任意数量的任意类型参数的函数调用。 这些参数被填充到32字节并连接。 一个例外是第一个参数被编码到正好四个字节的情况。 在这种情况下，这里没有填充以允许使用功能签名。

  三个功能`call`, `callcode`和`delegatecall`都是非常低级的功能，只能作为最后的手段，因为它们打破了Solidity的类型安全性。不鼓励使用`callcode`，将来会被删除。

- this 代表当前合约

  原因是对于合约来说，地址代表的就是合约本身，合约对象默认继承自地址对象，所以内部有地址的属性。

```
pragma solidity ^0.4.24;

contract addressBalance{

    function getContractAddrees() constant returns (address){
        return this;
```

```
    }

    function getBalance() constant returns (uint){
        return this.balance;
    }

    //向当前合约存款
    function testSend() payable returns (bool){
        address contractAddress = this;
        //msg.value 全局变量，调用合约的发起方转发的货币量，以wei为单位。
        //send() 执行的结果
        return contractAddress.send(msg.value);
    }

    function getAccountBalance() constant returns (uint) {
        return msg.sender.balance;
    }
}
```

四、固定长度字节数组

`bytes1`，`bytes2`，`bytes3`，…，`bytes32`。`byte`的别名就是`bytes1`。

- `bytes1`只能存储`一个`字节，也就是二进制`8位`的内容。
- `bytes2`只能存储`两个`字节，也就是二进制`16位`的内容。
- `bytes3`只能存储`三个`字节，也就是二进制`24位`的内容。
- ……
- `bytes32`能存储`三十二个`字节，也就是二进制`32 * 8 = 256`位的内容。

```
pragma solidity ^0.4.24;

contract Demo {

    byte public a = 0xe5;
    bytes1 public b = 0xe5;
    bytes2 public c = 0xe5ad;
    bytes3 public d = 0xe5ad94;
    bytes4 public e = 0xe5ad94e5;
    bytes5 public f = 0xe5ad94e5a3;
    bytes6 public g = 0xe5ad94e5a3b9;
}
```

操作运算符

- 比较运算符：`<=`，`<`，`==`，`!=`，`>=`，`>`

- 位操作符：`&`，`|`，`^(异或)`，`~（取反）`，`<<（左移）`，`>>（右移）`
- 索引访问：如果`x`是一个`bytesI`,那么可以通过`x[k](0 < k < I)`获取对应索引的字节，**PS:** x[k]是只读，不可写。

成员函数

- `.length` 返回字节的个数。（只读）
- 长度不可变
- 内部字节不可修改

```
pragma solidity ^0.4.4;

contract Demo {

    bytes6 public name = 0x6b6f6e677969;

    function nameByteLength() constant returns (uint) {
        // 报错
        // 长度不可变
        // name.length = 2;
        // 内部字节不可修改
        // name[0] = 0x1b;
        return name.length;
    }

}
```

`bytesI(1 <= I <= 32)`可以声明固定字节大小的字节数组变量，一旦声明，内部的字节和字节数组长度不可修改，当然可以通过索引读取(只读)对应索引的字节，或者通过`length`读取字节数组的字节数。

五、枚举类型

`ActionChoices`就是一个自定义的整型，当枚举数不够多时，它默认的类型为`uint8`，当枚举数足够多时，它会自动变成`uint16`，下面的`GoLeft == 0`,`GoRight == 1`,`GoStraight == 2`,`SitStill == 3`。在`setGoStraight`方法中，我们传入的参数的值可以是`0 - 3`当传入的值超出这个范围时，就会中断报错。

```
pragma solidity ^0.4.24;

contract Demo {
    enum ActionChoices {
        ActionChoicesGoLeft,
        ActionChoicesGoRight,
        ActionChoicesGoStraight,
        ActionChoicesSitStill
```

```
    }
    ActionChoices _choice;
    ActionChoices defaultChoice = ActionChoices.ActionChoicesGoStraight;

    function setActionChoice(ActionChoices choice) public {
        _choice = choice;
    }

    function getChoice() constant public returns (ActionChoices) {
        return _choice;
    }

    function getDefaultChoice() constant public returns (uint) {
        return uint(defaultChoice);
    }
}
```

六、函数

```
function 函数名(参数)  public|private|internal|external  pure|view|constant|payable
无返回值|returns (返回值类型)
```