wlang 语法介绍

梧桐链数字科技研究院

目录

简单介绍

wvm是梧桐链上相对于运行于docker中的go语言智能合约的另一种合约语言，运行于梧桐链自主研发的虚拟机，语法上类似java，方便上手，同时提供了许多便捷的内置函数。本文档旨在帮助梧桐链智能合约开发者快速入门，并且对wvm有一个简单的认识。

基础语法

下面看一个简单的wvm程序示例，它将打印字符串Hello world!

```
//合约示例
contract TestExample {
    public string test(){
        print("Hello world!")
        return ""
    }
}
```

运行流程

- 打开任意代码编辑器，输入以上代码
- 使用wvm install将代码安装到我们预制的测试环境
- 使用wvm invoke test执行test方法
- 控制台输出执行结果 Hello world!

基本语法

- 与java不同的是，wvm是由contract取代了class为开头，同时不需要';'为一行代码的结尾，直接使用回车换行作为一行代码的结束
- 合约名建议开头大写
- 整个合约没有主入口，任何方法都可以通过外部调用独立执行
- wvm是大小写敏感的

wvm标识符

- 所有的标识符都应以数字(0-9)字母(a-zA-Z)下划线(_)组成,同时开头只能是字母或者下划线
- 关键字不能作为标识符
- 合法标识符举例：a_1、_b2
- 非法标识符举例：2a、-b

wvm注释和空行

- wvm中使用'//'来进行注释，暂时不支持多行注释
- wvm在编译时自动会跳过空行和空格

基本类型

内置数据类型

整型：int、int8、int16、int32、int64、uint、uint8、uint16、uint32、uint64

字符型：string、byte

浮点型：float、double

布尔型：bool

自定义结构体类型

结构体：struct

变量类型

基本变量类型

在wvm中，所有变量在使用前必须声明。声明格式如下：

```
type identifier [ = value]
```

格式说明:type为wvm数据类型。identifier是变量名。

以下列举出一些变量的声明实例

```
byte z = 22          // 声明并初始化 z
string s = "wvm" // 声明并初始化字符串 s
double pi = 3.14159 // 声明了双精度浮点型变量 pi
```

结构体变量类型

在wvm中提供了用户自定义结构体类型，使用方法如下：

```
struct identifier_struct {
    type identifier
    [type identifier ...]
}
identifier_struct identifier
```

格式说明:struct是定义结构体的关键字，identifier_struct是自定义的结构体类型名称，声明结构体变量时就可以直接拿identifier_struct来声明变量。

以下列举出一个结构体变量的声明实例

```
struct student {
    int id
    string name
}
student s
```

数组变量类型

在wvm中也提供了数组类型，使用方法如下：

```
type [] identifier [= {value,value}]
type [num] identifier
```

格式说明:数组创建方式和java类似，但是wvm中的数组不需要实例化，声明即是实例化。目前仅支持二维数组

以下列举出一个数组变量的声明实例

```
int []b = {100,200}
int [2]b
```

map变量类型

在wvm中除了数组以外，也提供了map，使用方法如下：

```
map<type,type> identifier
identifier = {value:value}
[identifier = {value:value}...]
```

格式说明:数组创建方式和java类似，但是wvm中的数组不需要实例化，声明即是实例化。目前仅支持一维map

以下列举出一个map变量的实例

```
map<string,int> a
int b
a = {"5":5}
a = {"5":4}
b = 5+a<"5">
```

函数

wvm写的智能合约不需要主入口，可以在外部直接调用对应的函数。各个函数互相独立，在同一个合约内可以互相调用，函数外部可以定义全局变量，合约内所有函数都可以共同使用。函数内部定义的变量不互通。函数名不可重复，使用方法如下：

```
public type func_identifier([[type arg][,type arg]...]){
    [statement]...
    return type
}
```

格式说明:函数的创建方式和java类似，函数的执行语句由两个大括号包起来，每个函数都必须有return关键字

运算符

wvm作为一门计算机语言，当然也提供了运算符来操作变量，运算符一共分为以下几种（设表中的变量a为10，b为15）：

算术运算符

运算符	描述	例子
+	加法，相加运算符两侧的值。	a+b得到的结果为25。
-	减法，左操作数减去右操作数。	a-b得到的结果为-5。
*	乘法，相乘运算符两侧的值。	a*b得到的结果为150。
/	除法，左操作数除以右操作数。	a/b得到的结果为0.4。
%	取余，左操作数除以右操作数后的余数。	a%b得到的结果为10。
++	自增，操作数的值加1。	a++得到的结果为11。
--	自减，操作数的值减1。	a--得到的结果为9。

关系运算符

运算符	描述	例子
==	检查如果两个操作数的值是否相等，如果相等则条件为真。	(a == b)为假。
!=	检查如果两个操作数的值是否相等，如果值不相等则条件为真。	(a != b)为真。
>	检查左操作数的值是否大于右操作数的值，如果是那么条件为真。	(a > b)为假。
<	检查左操作数的值是否小于右操作数的值，如果是那么条件为真。	(a < b)为真。

运算符	描述	例子
>=	检查左操作数的值是否大于或等于右操作数的值，如果是那么条件为真。	(a >= b)为假。
<=	检查左操作数的值是否小于或等于右操作数的值，如果是那么条件为真。	(a <= b)为真。

逻辑运算符

操作符	描述	例子
&&	称为逻辑与运算符。当且仅当两个操作数都为真，条件才为真。	(a && b)为假。
||	称为逻辑或操作符。如果任何两个操作数任何一个为真，条件为真。	(a || b)为真。
!	称为逻辑非运算符。用来反转操作数的逻辑状态。如果条件为true，则逻辑非运算符将得到false。	!(a && b)为真。

赋值运算符

操作符	描述	例子
=	简单的赋值运算符，将右操作数的值赋给左侧操作数	c = a + b将把a + b得到的值赋给c

流程控制

wvm是一个图灵完备的语言，所以它也有循环和条件控制。

循环语句

while循环

while是最基本的循环，它的结构为：

```
while( 布尔表达式 ) {
  //循环内容
}
```

格式说明:只要布尔表达式为true，循环就会一直执行下去

for循环

for 循环，使一些循环结构变得更加简单。for循环执行的次数是在执行前就确定的。语法格式如下：

```
for(初始化; 布尔表达式; 更新) {
    //代码语句
}
```

格式说明:最先执行初始化步骤。可以声明一种类型，但可初始化一个或多个循环控制变量，也可以是空语句。然后，检测布尔表达式的值。如果为 true，循环体被执行。如果为false，循环终止，开始执行循环体后面的语句。执行一次循环后，更新循环控制变量再次检测布尔表达式。循环执行上面的过程。

条件语句 - if...else

if 语句的语法如下：

```
if(布尔表达式 1){
   //如果布尔表达式 1的值为true执行代码
}else if(布尔表达式 2){
   //如果布尔表达式 2的值为true执行代码
}else if(布尔表达式 3){
   //如果布尔表达式 3的值为true执行代码
}else {
   //如果以上布尔表达式都不为true执行代码
}
```

格式说明:f 语句至多有 1 个 else 语句，else 语句在所有的 else if 语句之后。if 语句可以有若干个 else if 语句，它们必须在 else 语句之前。一旦其中一个 else if 语句检测为 true，其他的 else if 以及 else 语句都将跳过执行。还有就是因为换行就是代码结束，所以'{'必须跟在关键字之后。

内置函数

```
json_to_obj<type>(string)
```
json字符串转对象,type表明对象类型。

```
db_set(key string,value obj)
```
数据库写入key-value值。

```
db_get<type>( key string)
```
通过key值从数据库获取value值。 type表明value值的类型。

```
db_exist(key string)
```
检查key是否存在数据库中。

```
verify(cryptType string,pub string,msg string,sign string)
return bool
```
sm2验签 依次输入 签名算法类型 公钥 原文 签名。返回验签成功与否。其中公钥需要使用pem格式，而签名使用十六进制格式编码

```
db_search(prefix_key string,filter funciton,res
map<string,string>)
```
db_search是一个获取批量数据到res这个map里面的内置函数，prefix_key是你想要获取的数据key的前缀，filter是一个过滤函数，在里面可以设定过滤条件，筛选出真正需要的数据。

```
add(arrary,value)
```
在数组末尾增加值

```
len(arrary)
```
获取数组长度

```
hash(string)return string
```
使用sm3的方式计算字符串的hash值

base64ToString(string)return string
将base64字符串解码

stringToBase64(string)return string
将字符串进行base64编码

getTxTime()return int64
获取当前交易的时间戳

getTxID()return string
获取当前交易ID

getTxSender()return string
获取当前交易的发送者

range(map,k,v) return bool
遍历map字典，将值写入到传入的参数k,v中，返回值bool若为false则是map遍历结束

invoke(string,string,string)
调用另一个合约（合约地址，合约方法，合约参数的数组字符串）

event(string,string)
发送合约事件到链上（事件主题，事件内容）

print(obj)
打印对象内容。

parseBool(s1 string)return bool
返回字符串s1表示的bool值。它接受1、0、t、f、T、F、true、false、True、False、TRUE、FALSE。

parseInt(s1 string,base int,bitSize int)return inbt64
返回字符串s1表示的整数值，接受正负号。
base指定进制（2到36），如果base为0，则会从字符串前置判断，"0x"是16进制，"0"是8进制，否则是10进制；
bitSize指定结果必须能无溢出赋值的整数类型，0、8、16、32、64 分别代表 int、int8、int16、int32、int64；

parseUint(String,bitSize int)return uint64
parseUint类似parseInt但不接受正负号，用于无符号整型。

parseFloat(String)return float
解析一个表示浮点数的字符串并返回其值。

parseDouble(String)return double
解析一个表示双精度浮点数的字符串并返回其值。

atoi(string)return int
atoi是parseInt(s, 10, 0)的简写。

itoa(int)return string
itoa是formatInt(i, 10) 的简写。

formatBool(b bool)return string
根据b的值返回"true"或"false"。

formatInt(i int64,int)return string
返回i的base进制的字符串表示。base 必须在2到36之间，结果中会使用小写字母'a'到'z'表示大于10的数字。

formatUint(u uint64,base int)return string
返回u的base进制的字符串表示。base 必须在2到36之间，结果中会使用小写字母'a'到'z'表示大于10的数字。

formatFloat(float,fmt byte,prec int)return string
函数将浮点数表示为字符串并返回。
fmt表示格式：'f'（-ddd.dddd）、'b'（-ddddp±ddd，指数为二进制）、'e'（-d.dddde±dd，十进制指数）、'E'（-d.ddddE±dd，十进制指数）、'g'（指数很大时用'e'格式，否则'f'格式）、'G'（指数很大时用'E'格式，否则'f'格式）。
prec控制精度（排除指数部分）：对'f'、'e'、'E'，它表示小数点后的数字个数；对'g'、'G'，它控制总的数字个数。如果prec 为-1，则代表使用最少数量的、但又必需的数字来表示f。

formatDouble(double,fmt byte,prec int)return string
函数将双精度浮点数表示为字符串并返回。
fmt表示格式：'f'（-ddd.dddd）、'b'（-ddddp±ddd，指数为二进制）、'e'（-d.dddde±dd，十进制指数）、'E'（-d.ddddE±dd，十进制指数）、'g'（指数很大时用'e'格式，否则'f'格式）、'G'（指数很大时用'E'格式，否则'f'格式）。prec控制精度（排除指数部分）：对'f'、'e'、'E'，它表示小数点后的数字个数；对'g'、'G'，它控制总的数字个数。
如果prec 为-1，则代表使用最少数量的、但又必需的数字来表示f。

quote(string)return string
返回字符串s的双引号字面值表示，控制字符、不可打印字符会进行转义。（如\t，\n，\xFF，\u0100）

toLowerCase(string)return string
返回将所有字母都转为对应的小写版本的拷贝。

toUpperCase(string)return string
返回将所有字母都转为对应的大写版本的拷贝。

trim(s string)return string
返回将s前后端所有空白（unicode.IsSpace指定）都去掉的字符串。

removeStr(string,cutstring string)return string
返回将s中cutstring子串都删除的新字符串。

replace(s string, old string, new string)return string
返回将s中old子串都替换为new的新字符串，替换所有old子串。

split(s string ,sep string)return string[]
用去掉s中出现的sep的方式进行分割，会分割到结尾，并返回生成的所有片段组成的数组。如果sep为空字符，Split会将s切分成每一个unicode码值一个字符串。

contains(s string,substr string)return bool
判断字符串s是否包含子串substr。

indexOf(s string,sep string)return int
子串sep在字符串s中第一次出现的位置，不存在则返回-1。

lastindexOf(s string,sep string) return int
子串sep在字符串s中最后一次出现的位置，不存在则返回-1。

title(s string)return string

返回s中每个单词的首字母都改为标题格式的字符串拷贝。

substring(s1 string,begin int,end int)return string
截取s1从begin位开始到end位结束，返回截取后的字符串

startsWith(s1 string,prefix string) return bool
判断s1是否以prefix字符串开头。若以prefix开头返回true 否则返回false

endsWith(s1 string,suffix string) return bool
判断s1是否以suffix字符串结尾。若以suffix结尾返回true 否则返回false

compareTo(s1 string,s2 string) return int
按照字典表比较s1与s2，返回 -1 、0、1。表明s1 小于、等于、大于 s2

countMatches(s1 string, s2 string)return int

统计s1中具有多少个字符串s2

reverseString(string) return string

反转字符串，输入字符串，输出字符串的反转值

合约示例

数组示例

```
//合约示例
contract TestExample {
    public string init(){
        return "success"
    }
    int[2][2] a
    public int test(string account,int amount){
        int k = 1
        int []b = {100,200}
```

```
        for(int i=0; i<2; i++){
            for(int j=0; j<2; j++){
                a[i][j] = k
                k=k+1
            }
        }
        a[1]=b
        a[1][0]= a[1][0] + a[0][0]
        if (check(a[1][0],3)){
            return a[1][0]
        }else{
            return check(a[1][0],3)
        }

    }

    public bool check(int a,int b){
        if (a>b){
            return true
        }else{
            return false
        }
    }
}
```

map示例

```
//合约示例
contract TestExample {
    public string init(){
        return "success"
    }
    public string test(string account,int amount){
        map<string,int> a
        int b
        a = {"5",5}
        a = {"5",4}
        b = 5+a<"5">
        return b
    }
}
```

结构体示例

```
//合约示例
contract TestExample {
    struct aa {
        int a
        int b
        int d
    }
    public string init(){
        return "success"
    }
    public int test(string account,int amount){
        int a
        aa b
        b.a= 4
        b.d= 7
        int k = b.a
        print(k)
        return b
    }
}
```