

Covariant Script编程语言

标准文档(190501)

# 目录

1. [**类型**](#_类型)
   1. [数值类型(Number)](#_number（数值）类型)
   2. [逻辑类型(Boolean)](#_boolean（逻辑）类型)
   3. [指针类型(Pointer)](#_pointer（指针）类型)
   4. [字符类型(Char)](#_char（字符）类型)
   5. [字符串类型(String)](#_string（字符串）类型)
   6. [数组类型(Array)](#_array（数组）类型)
   7. [线性表类型(List)](#_list（线性表）类型)
   8. [映射类型(Pair)](#_pair（映射）类型)
   9. [散列表类型(Hash Map)](#_hash_map（散列表）类型)
2. [**语法**](#_语法)
   1. [语句(Statements)](#_语句_1)
   2. [预处理(Preprocessor)](#_预处理)
   3. [关键字(Keywords)](#_关键字)
   4. [模块(Modules)](#_模块)
   5. [变量(Variables)](#_变量)
      1. [定义](#_定义)
      2. [在栈上新建对象](#_在栈上新建对象)
      3. [在堆上新建对象](#_在堆上新建对象)
   6. [表达式(Expression)](#_表达式)
      1. [定义](#_定义_1)
      2. [一元运算符](#_一元运算符)
      3. [二元运算符](#_二元运算符)
         1. [右结合运算符](#_右结合运算符)
         2. [左结合运算符](#_左结合运算符)
         3. [特殊运算符](#_特殊运算符)
      4. [条件表达式](#_条件表达式)
   7. [作用域和名称空间(Domain&Namespace)](#_作用域和名称空间)
      1. [定义](#_定义)
      2. [名称查找](#_名称查找)
   8. [语句(Statements)](#_语句)
      1. [分支语句](#_分支语句)
      2. [循环语句](#_循环语句)
      3. [控制语句](#_控制语句)
   9. [函数(Function)](#_函数)
      1. [Lambda表达式](#_Lambda表达式)
      2. [可变参数](#_可变参数(Variable_Arguments))
   10. [异常(Exception)](#_异常)
   11. [结构和类(Struct&Class)](#_结构和类)
       1. [定义](#_定义_2)
       2. [派生与继承](#_派生和继承)
       3. [控制结构的行为](#_控制结构的行为)
3. [**API**](#_API)
   1. [**标准库**](#_标准库)
      1. [全局(Global)](#_全局(Global))
      2. [异常(Exception)](#_异常(Exception))
      3. [输入输出流(Iostream)](#_输入输出流(Iostream))
         1. [寻位方向(Seekdir)](#_寻位方向名称空间)
         2. [打开方式(Openmode)](#_打开方式名称空间)
         3. [输入流(Istream)](#_输入流名称空间)
         4. [输出流(Ostream)](#_输出流名称空间)
      4. [系统(System)](#_系统(System))
         1. [控制台(Console)](#_控制台名称空间)
         2. [文件(File)](#_文件名称空间)
         3. [路径(Path)](#_路径名称空间)
            1. [路径类型(Type)](#_路径类型名称空间)
            2. [路径信息(Info)](#_路径信息名称空间)
      5. [运行时(Runtime)](#_运行时(Runtime))
      6. [数学(Math)](#_数学(Math))
         1. [常量(Constants)](#_数学常量名称空间)
      7. [字符(Char)](#_字符(Char))
      8. [字符串(String)](#_字符串(String))
      9. [数组(Array)](#_数组(Array))
         1. [数组迭代器(Iterator)](#_数组迭代器名称空间)
      10. [线性表(List)](#_线性表(List))
          1. [线性表迭代器(Iterator)](#_线性表迭代器名称空间)
      11. [映射(Pair)](#_映射(Pair))
      12. [散列表(Hash Map)](#_散列表(Hash_Map))
   2. [**扩展库**](#_扩展库)
      1. [图形库扩展(ImGUI)](#_图形库扩展(ImGUI))
         1. [OPEN GL函数](#_OPEN_GL函数)
         2. [应用程序](#_应用程序)
         3. [样式](#_样式)和字体
         4. [窗口](#_窗口)
         5. [布局](#_布局)
         6. [标识符](#_标识符)
         7. [控件](#_控件)
         8. [提示信息](#_提示信息)
         9. [树形结构](#_树形结构)
         10. [菜单](#_菜单)
         11. [标签](#_标签)
         12. [表格](#_表格)
         13. [焦点](#_焦点)
         14. [工具](#_工具)
         15. [输入](#_输入)
         16. [画板](#_画板)
         17. [应用程序(Application)](#_应用程序名称空间)
         18. [位图(Bmp Image)](#_位图名称空间)
         19. [方位(Dirs)](#_方位名称空间)
         20. [窗口参数(Flags)](#_窗口参数名称空间)
         21. [标签参数(Flags)](#_标签参数名称空间)
         22. [特殊键(Keys)](#_特殊键名称空间)
      2. [字符图形库扩展(Darwin)](#_字符图形库扩展(Darwin))
         1. [Darwin图形(Ui)](#_Darwin图形名称空间)
         2. [Darwin核心(Core)](#_Darwin核心名称空间)
         3. [Darwin画布(Drawable)](#_Darwin画布名称空间)
      3. [正则表达式扩展(Regex)](#_正则表达式扩展(Regex))
         1. [正则匹配结果(Result)](#_正则匹配结果名称空间)
      4. [Base64编解码扩展(Codec)](#_Base64编解码扩展(Codec))
         1. [Base32编码译码器(Base32)](#_Base32编码译码器名称空间)
         2. [Base64编码译码器(Base64)](#_Base64编码译码器名称空间)
         3. [译码器函数](#_译码器函数)
      5. [流式API扩展(Streams)](#_流式API扩展(Streams))
      6. [数据库扩展(SQLite)](#_数据库扩展(SQLite))
         1. [SQLite语句(Statement)](#_SQLite_语句名称空间)
      7. [套接字扩展(Network)](#_套接字扩展(Network))
         1. [TCP套接字(Tcp)](#_TCP_套接字名称空间)
            1. [TCP套接字类型](#_TCP_套接字类型扩展)
            2. [TCP端点类型](#_TCP_端点类型扩展)
         2. [UDP套接字(Udp)](#_UDP_套接字名称空间)
            1. [UDP套接字类型](#_UDP_套接字类型扩展)
            2. [UDP端点类型](#_UDP_端点类型扩展)
4. [**解释器**](#_解释器)
   1. [**命令行参数**](#_命令行参数)
      1. [解释器(Cs)](#_解释器(Cs))
      2. [交互式解释器(Cs REPL)](#_交互式解释器(Cs_REPL))
      3. [调试器(Cs Debugger)](#_调试器(Cs_Debugger))
         1. [调试器命令](#_调试器命令)
   2. [**解释器扩展**](#_解释器扩展)
      1. [基于CNI标准扩展的Covariant Script解释器扩展](#_基于CNI标准扩展的Covariant_Script解释器扩展)
         1. [头文件](#_头文件)
         2. [标准类型转换规则](#_标准类型转换规则)
         3. [CNI组成宏](#_CNI组成宏)
            1. [CNI根名称空间](#_CNI根名称空间)
            2. [声明CNI函数](#_声明CNI函数)
            3. [声明CNI变量](#_声明CNI变量)
            4. [声明CNI名称空间](#_声明CNI名称空间)
            5. [CNI类型扩展](#_CNI类型扩展)

# 类型

## number（数值）类型

number 的字面量由 0~9 十个数和小数点组成，如 12，3.14 等

number 表示的数仅限于全体实数

number 的初始值为0

## boolean（逻辑）类型

boolean 的字面量只有两个，分别是 true（真）和 false（假）

boolean 的初始值为 true（真）

## pointer（指针）类型

pointer 没有字面量

pointer 指向存储在堆空间中的对象，可使用gcnew运算符新建一个堆上对象

pointer 的初始值为 null（空指针），null（空指针）指向一个无意义的对象，不允许解引用一个空指针

不再使用的对象将由 GC（垃圾回收器）自动回收

可使用解引用运算符访问指针指向的对象，若指针指向的是结构体实例，可使用箭头运算符访问结构体的成员

## char（字符）类型

char 的字面量是由单引号括起的单个ASCII字符，如'A'，'C'等

特殊符号需使用转义序列来表示：

|  |  |
| --- | --- |
| 转义序列 | 符号 |
| \a | 响铃 |
| \b | 退格 |
| \f | 换页 |
| \n | 换行 |
| \r | 回车 |
| \t | 水平制表 |
| \v | 垂直制表 |
| \\ | 反斜杠 |
| \' | 单引号 |
| \" | 双引号 |
| \0 | 空字符 |

char 的初始值为 ‘\0’（空字符）

## string（字符串）类型

string 的字面量是由双引号括起的任意个数 ASCII或UTF-8字符，如"Hello"

string 的初始值为""（空串）

可使用下标运算符访问字符串中的字符，下标不能超出范围(0~字符串长度-1)

多段字符串可以使用加运算符连接

## array（数组）类型

array 的字面量为大括号扩起的以逗号分隔的任意个数元素，如 {1,2,3}

array 是一种元素均匀分布的顺序容器，初始值为{}（空数组）

可使用下标运算符访问数组中的元素，下标小于零时即为倒序访问

Covariant Script 编程语言的数组是变长数组(VLA)，若下标超出范围数组将自动增长，增长的部分自动填0

## list（线性表）类型

list 没有字面量

list 是一种元素不均匀分布的顺序容器

list 的初始值为空表

## pair（映射）类型

pair 的字面量为冒号对应的一个键值对，如2:3

pair 的初始值为0:0（空映射）

## hash\_map（散列表）类型

hash\_map 没有字面量

hash\_map 是一种元素均匀分布的无序容器

hash\_map 要求其存储的映射的键的类型必须支持生成哈希值

hash\_map 的初始值为空表

可使用下标运算符访问散列表中的键对应的值，若键不存在，将自动建立键与0组成的映射

# 语法

## 说明

本文档格式为：正文内容 是代码或正文，*斜体内容* 是说明，**加粗内容** 是解释

## 语句

*语句*

**以换行符结束的一行代码称为语句**

*语句1* ; *语句2* ; *语句3*

**任意语句都可以将分号作为终结符**

单独一行语句也可以使用分号作为结尾

## 预处理

*语句* # *注释*

**任意#之后的内容均视为注释**

@begin

*语句*

@end

**在 @begin 和 @end 之间的代码将视为一行语句**

也就是说， @begin 和 @end 之间的所有换行符都将会被忽略

## 关键字

Covariant Script的关键字分为两种：

一种为**强制型关键字**，即编译器遇到这个词即视为使用这个语法标识符；

一种为**标识型关键字**，即仅在符合语法时编译器才会将其视为语法标识符

强制型关键字表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 关键字 | and | or | not | typeid | new |
| 含义 | 与运算符 | 或运算符 | 非运算符 | 类型信息 | 新建栈对象 |
| null | local | global | true | false | gcnew |
| 空指针 | 本地作用域 | 全局作用域 | 逻辑真 | 逻辑假 | 新建堆对象 |

标识型关键字表(190501)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 关键字 | import | package | namespace | using | struct | class | extends |
| 含义 | 引入包 | 包 | 声明名称空间 | 引入名称空间 | 声明结构体 | | 继承结构体 |
| block | var | constant | if | else | switch | case | default |
| 声明语句块 | 声明变量 | 声明常量 | 分支语句 | | | | |
| end | while | loop | until | for | foreach | in | do |
| 结束语句块 | 循环语句 | | | | | | |
| break | continue | function | override | return | try | catch | throw |
| 跳出循环 | 进入下一轮循环 | 声明函数 | 覆写函数 | 返回语句 | 异常处理 | | |

**注意，在190501前的标准中，标识性关键字也属于强制型关键字的一部分**

## 模块

import *Package名, Package名...*

**引入一个或多个Package**

引入的Package可以是 \*.csp 文件(CovScript包)或者是 \*.cse 文件(CovScript扩展)

当两者同时存在时会优先引入 \*.csp 文件(CovScript包)

package *Package名*

**声明一个Package**

原则上包名应和文件名相同

## 变量

### 定义

var *变量名 = 表达式, 变量名 = 表达式...*

**定义一个或多个变量，初始值为表达式的值**

constant *常量名 = 表达式, 常量名 = 表达式...*

**定义一个或多个常量，其值为表达式的值**

常量和变量的处理方式完全不同，常量类似于 C++中的 constexpr，其实际上**仅存在于编译期**。使用常量将有利于提高性能，编译器在某些情况下也会自动进行常量折叠，但显式声明将帮助编译器更进一步优化您的程序

### 在栈上新建对象

new *类型*

**在栈上新建一个指定类型的对象**

此对象将会遵从RAII原则自动回收

### 在堆上新建对象

gcnew *类型*

**在堆上新建一个指定类型的对象，并返回指向这个对象的指针**

此对象将会由垃圾回收器自动回收

## 表达式

### 定义

表达式由操作数和运算符组成

*操作数 运算符 操作数*

一般有左右两个操作数的运算符是二元运算符

只有一个操作数的运算符是一元运算符

二元运算符有结合律，左结合是从右向左运算，右结合是从左向右运算

所有的运算符都有优先级，优先级越高越先计算

**注意，对于递增、递减、赋值等会修改操作数的运算符，要求表达式的计算结果可被修改，不能是表达式计算中产生的中间临时量或常量**

### 一元运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 优先级 | 功能 |
| - *表达式* | 10 | 对数值进行数学取反运算 |
| \* *表达式* | 11 | 对指针进行解引用 |
| typeid *表达式* | 14 | 获取表达式的运行时类型信息 |
| new *表达式* | 14 | 新建表达式表示的类型的对象 |
| gcnew *表达式* | 14 | 新建表达式表示的类型的内存区块 |
| ! *表达式* | 8 | 对表达式进行非运算 |
| ++ *表达式* | 13 | 对表达式进行递增运算 |
| *表达式* ++ | 13 | 对表达式进行递增运算并保留表达式原先的值 |
| -- *表达式* | 13 | 对表达式进行递减运算 |
| *表达式* -- | 13 | 对表达式进行递减运算并保留表达式原先的值 |

### 二元运算符

#### 右结合运算符

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 优先级 | 结合律 | 功能 |
| *表达式* + *表达式* | 10 | 右 | 对数值进行数学加法运算或拼接字符串 |
| *表达式* - *表达式* | 10 | 右 | 对数值进行数学减法运算 |
| *表达式* \* *表达式* | 11 | 右 | 对数值进行数学乘法运算 |
| *表达式* / *表达式* | 11 | 右 | 对数值进行数学除法运算 |
| *表达式* % *表达式* | 12 | 右 | 对数值进行数学取余运算 |
| *表达式* ^ *表达式* | 12 | 右 | 对数值进行数学幂运算 |
| *表达式* . *表达式* | 15 | 右 | 对各种对象进行访问 |
| *表达式* -> *表达式* | 15 | 右 | 对指针指向的各种对象进行访问 |
| *表达式* < *表达式* | 9 | 右 | 比较左侧数值是否小于右侧 |
| *表达式* > *表达式* | 9 | 右 | 比较左侧数值是否大于右侧 |
| *表达式* <= *表达式* | 9 | 右 | 比较左侧数值是否小于或等于右侧 |
| *表达式* >= *表达式* | 9 | 右 | 比较左侧数值是否大于或等于右侧 |
| *表达式* == *表达式* | 9 | 右 | 比较两侧表达式的值是否相等 |
| *表达式* != *表达式* | 9 | 右 | 比较两侧表达式的值是否不相等 |
| *表达式* && *表达式* | 7 | 右 | 对两侧表达式进行与运算，若左侧为假则停止求值 |
| *表达式* and*表达式* |
| *表达式* || *表达式* | 6 | 右 | 对两侧表达式进行或运算，若左侧为真则停止求值 |
| *表达式* or *表达式* |

#### 左结合运算符

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 优先级 | 结合律 | 功能 |
| *表达式* = *表达式* | 1 | 左 | 将右侧表达式的值赋予左边 |
| *表达式* += *表达式* | 1 | 左 | 对数值进行数学加法运算或对字符串进行拼接后再将其值赋予左边 |
| *表达式* -= *表达式* | 1 | 左 | 对数值进行数学减法运算后再赋予左边 |
| *表达式* \*= *表达式* | 1 | 左 | 对数值进行数学乘法运算后再赋予左边 |
| *表达式* /= *表达式* | 1 | 左 | 对数值进行数学除法运算后再赋予左边 |
| *表达式* %= *表达式* | 1 | 左 | 对数值进行数学取余运算后再赋予左边 |
| *表达式* ^= *表达式* | 1 | 左 | 对数值进行数学幂运算后再赋予左边 |

#### 

#### 特殊运算符

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 优先级 | 结合律 | 功能 |
| *表达式*, *表达式* | 0 | 右 | 联接多个表达式并依次运算 |
| *表达式* : *表达式* | 4 | 右 | 建立左侧表达式的值到右侧的映射 |
| *表达式*(*参数列表*) | 15 | 右 | 调用表达式表示的函数 |
| *表达式*[*表达式*] | 15 | 右 | 访问数组、哈希表或是字符串中的元素 |
| *表达式*... | 20 | 无 | 若表达式的值为数组，则将其展开 |
| (*表达式*) | 无 | 无 | 创建子表达式 |

逗号表达式将遵循从左到右的规则，其值为最后一个表达式的值

数组在访问时，若下标越界会自动增长，增长的部分填0；若下标为负则访问下标为长度-下标绝对值的元素，哈希表在访问时若映射不存在则将自动建立到0的映射

展开表达式的使用范围**仅限**函数调用时的参数列表和数组字面量的声明中

小括号扩起的子表达式将无视内部的优先级而将其看作以一个整体进行求值

### 条件表达式

*逻辑表达式* ? *表达式1* : *表达式2*

逻辑表达式的值为真时整个表达式的值为表达式1

逻辑表达式的值为假时整个表达式的值为表达式2

## 作用域和名称空间

### 定义

block

*语句块*

end

**定义一个临时作用域**

临时作用域中的变量会在离开作用域后销毁

namespace *名称空间名*

*语句块*

end

**定义一个名称空间**

名称空间中只允许引入其他名称空间，变量定义，函数定义，类型定义以及名称空间定义

using *名称空间名*

引入一个名称空间，这将会在当前作用域中建立引入的名称空间中所有变量的引用

### 名称查找

*变量名*

**从最上层作用域开始向下查找变量**

local.*变量名*

**查找当前作用域中的变量**

global.*变量名*

**查找全局作用域中的变量**

*名称空间名*.*变量名*

**查找名称空间中的变量**

*变量名*.*变量名*

**查找结构体或扩展中的变量**

作用域结构以及变量查找方式如图所示

注意，对于最后一种访问方法,仅变量类型为结构或支持扩展的类型时可用，如访问的是扩展或结构中的函数，将会把点运算符左边的变量作为函数的第一个参数传入。

也就是说: char.isspace(ch) 等价于 ch.isspace()

## 语句

### 分支语句

if *逻辑表达式*

*语句块*

end

逻辑表达式的值为真则执行语句块

if *逻辑表达式*

*语句块1*

else

*语句块2*

end

逻辑表达式的值为真则执行语句块1

逻辑表达式的值为假则执行语句块2

switch *表达式*

case *常量标签*

*语句块*

end

default

*语句块*

end

end

执行与表达式的值相等的常量标签对应的case中的语句块

当无匹配的常量标签时会执行default中的语句块，如default未找到则跳出

常量标签的类型必须支持生成哈希值

### 循环语句

while *逻辑表达式*

*语句块*

end

当逻辑表达式的值为真时循环执行语句块

loop

*语句块*

until *逻辑表达式*

直到逻辑表达式的值为真时跳出循环

loop

*语句块*

end

循环执行语句块直到用户手动跳出

for *变量名* = *表达式*, *逻辑表达式*, *后处理表达式*

*语句块*

end

定义一个变量，当逻辑表达式为真时循环执行语句块，在每个循环的最后执行后处理表达式

for *变量名* = *表达式*, *逻辑表达式*, *后处理表达式* do *表达式*

定义一个变量，当逻辑表达式为真时循环执行表达式，在每个循环的最后执行后处理表达式

foreach *变量名* in *表达式*

*语句块*

end

表达式的值必须是一个支持 foreach 遍历的容器

定义一个变量正序遍历容器，循环执行语句块

foreach *变量名* in *表达式1* do *表达式2*

表达式1的值必须是一个支持 foreach 遍历的容器

定义一个变量正序遍历容器，循环执行表达式2

### 控制语句

break

**跳出循环**

continue

**进入下一轮循环**

return

**结束函数并返回 null**

return 表达式

**结束函数并返回表达式的值**

## 函数

function *函数名*(*参数列表（可选）*)

*语句块*

end

**定义一个函数**

参数列表中的参数只能指定名称，参数名不可重复，各参数之间以逗号分隔，如：

function test(a0, a1, a2)

end

### Lambda表达式

[](*参数列表（可选）*) -> *表达式*

**定义一个Lambda表达式**

参数列表中的参数只能指定名称，参数名不可重复，各参数之间以逗号分隔

Lambda 表达式是一种匿名函数，调用 Lambda 表达式将计算表达式的值并返回

### 可变参数

在声明函数或Lambda表达式时，可在参数列表中声明**可变参数列表**：

function (...*参数名*)

*语句块*

end

或

[](...*参数名*) -> *表达式*

若声明了可变参数列表，则参数列表中不允许有其他任何形式的参数，这种函数我们称之为**可变参数函数**

可变参数函数在被调用时参数数量**无上限**

可变参数函数被调用时，可变参数列表会被以数组的形式呈现，若需要二次转发可使用[展开运算符](#_特殊运算符)展开

## 异常

throw 异常

**抛出一个异常**

try

*语句块*

catch *异常名*

*语句块*

end

测试代码是否会抛出异常，如抛出则抓取异常

## 结构和类

### 定义

struct *结构名*

*结构体*

end

结构定义后结构名就可以作为类型名使用

结构体中只允许变量定义和函数定义

结构体中的变量或函数称为结构体的成员

编译器会为成员函数插入一个隐式的this参数

this指的是调用成员函数的结构实例本身

this只在成员函数中可用

可用class代替struct关键字编写程序，两者无实质区别

### 派生和继承

struct *结构名* extends *父类结构名*

*结构体*

end

派生结构将引入父类结构的所有成员并自动插入一个名为parent的成员

parent成员是结构实例本身的父类实例

如果派生类想要重新实现父类函数，可使用override关键字覆写

function *函数名*(*参数列表（可选）*) override

*语句块*

end

### 控制结构的行为

function initialize()

*语句块*

end

**构造函数，结构被构建时调用，返回值无意义**

function duplicate(orig)

*语句块*

end

**复制函数，结构被复制时调用，参数为被复制的实例，返回值无意义**

function equal(orig)

*语句块*

return true

end

**比较函数，结构被比较时调用，参数为等号右边的实例，必须返回true（代表相等）或false（代表不相等）**

function finalize()

*语句块*

end

**析构函数，结构被回收时调用，返回值无意义**

# API

## 说明

函数在本文档中表示为 *返回类型* *函数名*(*参数列表（可选）*)

返回类型标注为void的将返回null

参数列表中的参数表示为 *参数类型* *参*数名（可选）

参数类型和返回类型如用中括号([ ])标记，则表示这个类型不是Covariant Script内建类型

## 标准库

### 全局(Global)

|  |  |
| --- | --- |
| 代码 | 功能 |
| context | 运行环境 |
| char | 字符类型 |
| number | 数值类型 |
| boolean | 逻辑类型 |
| pointer | 指针类型 |
| string | 字符串类型 |
| list | 链表类型 |
| array | 数组类型 |
| pair | 映射类型 |
| hash\_map | 哈希表类型 |
| exception | 异常名称空间 |
| iostream | 输入输出流名称空间 |
| system | 系统名称空间 |
| runtime | 运行时名称空间 |
| math | 数学名称空间 |
| [range] range(number stop) | 生成一个步长为1的一个区间[0, stop) |
| [range] range(number start, number stop[, number step=1]) | 生成一个步长为step(默认为1)的区间[start, stop) |
| number to\_integer(var) | 将一个变量转换为整数 |
| string to\_string(var) | 将一个变量转换为字符串 |
| string type(var) | 获取一个变量的类型名称 |
| var clone(var) | 复制一个变量并返回 |
| void swap(var,var) | 交换两个变量的值 |

注意，range函数生成的区间仅可配合foreach用于遍历，如：

foreach num in range(10) do system.out.println(num)

### 异常(Exception)

|  |  |
| --- | --- |
| 代码 | 功能 |
| string what([exception]) | 获取异常详情 |

### 输入输出流(Iostream)

|  |  |
| --- | --- |
| 代码 | 功能 |
| seekdir | 寻位方向名称空间(3.1.3.1) |
| openmode | 打开方式名称空间(3.1.3.2) |
| istream | 输入流名称空间(3.1.3.3) |
| ostream | 输出流名称空间(3.1.3.4) |
| [istream/ostream] fstream(string path,[openmode] mode) | 新建一个文件流,具体类型取决于打开方式 |
| void setprecision(number) | 设置输出精度(to\_string 的精度) |

#### 寻位方向名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| start | 流的开始 |
| finish | 流的结尾 |
| present | 当前位置 |

#### 打开方式名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| in | 为读打开(输入流) |
| out | 为写打开(清空内容,输出流) |
| app | 为写打开(追加内容,输出流) |

#### 输入流名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| char get([istream]) | 读取字符 |
| char peek([istream]) | 读取下一个字符而不删除 |
| void unget([istream]) | 放回字符 |
| string getline([istream]) | 读取一行 |
| number tell([istream]) | 获取流位置指示器 |
| void seek([istream],number pos) | 设置流位置 |
| void seek\_from([istream],[seekdir],number offset) | 设置相对寻位方向的流位置 |
| boolean good([istream]) | 检查是否有错误 |
| boolean eof([istream]) | 检查是否到达文件结尾 |
| var input([istream]) | 从流中获取输入(格式化) |
| void ignore([istream]) | 忽略流中下一行前所有内容 |

#### 输出流名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| void put([ostream],char) | 插入字符 |
| number tell([ostream]) | 获取流位置指示器 |
| void seek([ostream],number pos) | 设置流位置 |
| void seek\_from([ostream],[seekdir],number offset) | 设置相对寻位方向的流位置 |
| void flush([ostream]) | 与底层存储设备同步 |
| boolean good([ostream]) | 检查是否有错误 |
| void print([ostream],var) | 向流中输出内容,仅可输出支持 to\_string 的类型(不换行) |
| void println([ostream],var) | 向流中输出内容,仅可输出支持 to\_string 的类型(换行) |

### 系统(System)

|  |  |
| --- | --- |
| 代码 | 功能 |
| console | 控制台名称空间(3.1.4.1) |
| file | 文件名称空间(3.1.4.2) |
| path | 路径名称空间(3.1.4.3) |
| in | 标准输入流 |
| out | 标准输出流 |
| number run(string) | 在系统环境中运行一条指令,返回错误码 |
| string getenv(string) | 获取环境变量的值并返回 |
| void exit(number) | 清理资源并退出 |

#### 控制台名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| number terminal\_width() | 获取控制台宽度 |
| number terminal\_height() | 获取控制台高度 |
| void gotoxy(number x, number y) | 移动光标 |
| void echo(boolean) | 设置光标可见性 |
| void clrscr() | 清屏 |
| char getch() | 获取键盘输入 |
| bool kbhit() | 判断是否有键盘输入 |

#### 文件名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| boolean copy(string,string) | 复制文件 |
| boolean remove(string) | 删除文件 |
| boolean exists(string) | 判断文件是否存在 |
| boolean rename(string,string) | 重命名/移动文件 |

#### 路径名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| type | 路径类型名称空间(3.1.4.3.1) |
| info | 路径信息名称空间(3.1.4.3.2) |
| separator | 路径分隔符 |
| delimiter | 路径定界符 |
| array scan(string) | 扫描路径 |

##### 路径类型名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| unknown | 未知 |
| fifo | 管道 |
| sock | 套接字 |
| chr | 字符设备 |
| dir | 文件夹 |
| blk | 块设备 |
| reg | 常规 |
| lnk | 链接 |

##### 路径信息名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| string name([path\_info]) | 获取路径名 |
| [path\_type] type([path\_info]) | 获取路径类型 |

### 运行时(Runtime)

|  |  |
| --- | --- |
| 代码 | 功能 |
| std\_version | 标准版本号(Number) |
| void info() | 解释器版本信息 |
| string get\_import\_path() | 获取引入目录 |
| number time() | 获取计时器的读数,单位毫秒 |
| void delay(number) | 使程序暂停一段时间,单位毫秒 |
| [exception] exception(string) | 新建运行时异常 |
| [hash\_value] hash(var) | 计算一个变量的哈希值 |
| array cmd\_args([context]) | 获取运行参数 |
| [expression] build([context],string) | 构建一个可用于计算的表达式 |
| var solve([context],[expression]) | 计算一个表达式 |
| [namespace] advance\_import([context],string path,string name) | 动态加载一个扩展，规则与import语句相同 |

### 数学(Math)

|  |  |
| --- | --- |
| 代码 | 功能 |
| constants | 常量名称空间(3.1.6.1) |
| number abs(number) | 绝对值 |
| number ln(number) | 以 e 为底的对数 |
| number log10(number) | 以 10 为底的对数 |
| number log(number a,number b) | 以 a 为底 b 的对数 |
| number sin(number) | 正弦 |
| number cos(number) | 余弦 |
| number tan(number) | 正切 |
| number asin(number) | 反正弦 |
| number acos(number) | 反余弦 |
| number atan(number) | 反正切 |
| number sqrt(number) | 开方 |
| number root(number a,number b) | a 的 b 次方根 |
| number pow(number a,number b) | a 的 b 次方 |
| number min(number a,number b) | a 和 b 的最小值 |
| number max(number a,number b) | a 和 b 的最大值 |
| number rand(number,number) | 获取区间内的伪随机数 |
| number randint(number,number) | 获取区间内的伪随机整数 |

#### 数学常量名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| max | 数值类型最大值(Number) |
| min | 数值类型最小值(Number) |
| inf | 数值类型正无穷(Number) |
| nan | 数值类型无意义(Number) |
| pi | 圆周率(Number) |
| e | 自然底数(Number) |

### 字符(Char)

|  |  |
| --- | --- |
| 代码 | 功能 |
| boolean isalnum(char) | 检查字符是否是字母或数字 |
| boolean isalpha(char) | 检查字符是否是字母 |
| boolean islower(char) | 检查字符是否是小写字母 |
| boolean isupper(char) | 检查字符是否是大写字母 |
| boolean isdigit(char) | 检查字符是否是数字 |
| boolean iscntrl(char) | 检查字符是否是控制字符 |
| boolean isgraph(char) | 检查字符是否是图形字符 |
| boolean isspace(char) | 检查字符是否是空白字符 |
| boolean isblank(char) | 检查字符是否是空格或 tab |
| boolean isprint(char) | 检查字符是否是打印字符 |
| boolean ispunct(char) | 检查字符是否是标点符号 |
| char tolower(char) | 将字符转换为小写 |
| char toupper(char) | 将字符转换为大写 |
| char from\_ascii(number) | 将 ascii 码转换为字符 |

### 字符串(String)

|  |  |
| --- | --- |
| 代码 | 功能 |
| string append(string,var) | 在尾部追加内容 |
| string insert(string,number,var) | 在指定位置处插入内容 |
| string erase(string,number,number) | 将范围内的字符删除 |
| string replace(string,number,number,var) | 将从指定位置开始的指定个数字符替换 |
| string substr(string,number,number) | 从指定位置截取指定长度的子字符串 |
| number find(string,string,number) | 从指定位置开始从左向右查找一段字符串 |
| number rfind(string,string,number) | 从指定位置开始从右向左查找一段字符串 |
| string cut(string,number) | 从尾部删除指定长度的字符串 |
| boolean empty(string) | 检查字符串是否为空 |
| void clear(string) | 清空 |
| number size(string) | 获取字符个数 |
| string tolower(string) | 将字符串转换为小写 |
| string toupper(string) | 将字符串转换为大写 |
| number to\_number(string) | 将字符串转换为数值 |
| array split(string,array) | 使用指定的字符集合分割字符串 |

### 数组(Array)

|  |  |
| --- | --- |
| 代码 | 功能 |
| iterator | 数组迭代器名称空间(3.1.9.1) |
| var at(array,number) | 访问指定的元素,同时进行越界检查 |
| var front(array) | 访问第一个元素 |
| var back(array) | 访问最后一个元素 |
| [iterator] begin(array) | 获取指向容器第一个元素的迭代器 |
| [iterator] end(array) | 获取指向容器尾端的迭代器 |
| boolean empty(array) | 检查容器是否为空 |
| number size(array) | 获取容纳的元素数 |
| void clear(array) | 删除全部内容 |
| [iterator] insert(array,[iterator],var) | 插入元素, 插入到迭代器指向的元素之前,返回指向插入的元素的迭代器 |
| [iterator] erase(array,[iterator]) | 删除元素,返回指向要删除的元素的下一个元素的迭代器 |
| void push\_front(array,var) | 在容器的开始处插入新元素 |
| void pop\_front(array) | 删除第一个元素 |
| void push\_back(array,var) | 将元素添加到容器末尾 |
| void pop\_back(array) | 删除最后一个元素 |
| hash\_map to\_hash\_map(array) | 将数组转换为散列表,要求数组中元素必须都是映射 |
| list to\_list(array) | 将数组转换为链表 |

#### 数组迭代器名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| [iterator] forward([iterator]) | 向前移动迭代器 |
| [iterator] backward([iterator]) | 向后移动迭代器 |
| var data([iterator]) | 访问迭代器指向的元素 |

### 线性表(List)

|  |  |
| --- | --- |
| 代码 | 功能 |
| iterator | 线性表迭代器名称空间(3.1.10.1) |
| var front(list) | 访问第一个元素 |
| var back(list) | 访问最后一个元素 |
| [iterator] begin(list) | 获取指向容器第一个元素的迭代器 |
| [iterator] end(list) | 获取指向容器尾端的迭代器 |
| boolean empty(list) | 检查容器是否为空 |
| number size(list) | 获取容纳的元素数 |
| void clear(list) | 删除全部内容 |
| [iterator] insert(list,[iterator],var) | 插入元素, 插入到迭代器指向的元素之前,返回指向插入的元素的迭代器 |
| [iterator] erase(list,[iterator]) | 删除元素,返回指向要删除的元素的下一个元素的迭代器 |
| void push\_front(list,var) | 在容器的开始处插入新元素 |
| void pop\_front(list) | 删除第一个元素 |
| void push\_back(list,var) | 将元素添加到容器末尾 |
| void pop\_back(list) | 删除最后一个元素 |
| void remove(list,var) | 删除所有与指定变量相等的元素 |
| void reverse(list) | 将该线性表的所有元素的顺序反转 |
| void unique(list) | 删除连续的重复元素 |

#### 线性表迭代器名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| [iterator] forward([iterator]) | 向前移动迭代器 |
| [iterator] backward([iterator]) | 向后移动迭代器 |
| var data([iterator]) | 访问迭代器指向的元素 |

### 映射(Pair)

|  |  |
| --- | --- |
| 代码 | 功能 |
| var first(pair) | 获取第一个元素 |
| var second(pair) | 获取第二个元素 |

### 散列表(Hash Map)

|  |  |
| --- | --- |
| 代码 | 功能 |
| boolean empty(hash\_map) | 检查容器是否为空 |
| number size(hash\_map) | 获取容纳的元素数 |
| void clear(hash\_map) | 删除全部内容 |
| void insert(hash\_map,var,var) | 插入一对映射 |
| void erase(hash\_map,var) | 删除键对应的映射 |
| var at(hash\_map,var) | 访问指定的元素,同时进行越界检查 |
| boolean exist(hash\_map,var) | 查找是否存在映射 |

## 扩展库

### 图形库字体扩展(ImGUI Font)

|  |  |
| --- | --- |
| 代码 | 功能 |
| source\_han\_sans | 思源黑体字体(imgui font) |

### 图形库扩展(ImGUI)

|  |  |
| --- | --- |
| 代码 | 功能 |
| application | 应用程序名称空间(3.2.1.17) |
| image\_type | 图像名称空间(3.2.1.18) |
| dirs | 方位名称空间(3.2.1.19) |
| flags | 窗口/标签参数名称空间(3.2.1.20/21) |
| keys | 特殊键名称空间(3.2.1.22) |

#### OPEN GL函数

|  |  |
| --- | --- |
| 代码 | 功能 |
| number get\_monitor\_count() | 获取屏幕数量 |
| number get\_monitor\_width(number monitor\_id) | 获取指定屏幕的宽度 |
| number get\_monitor\_height(number monitor\_id) | 获取指定屏幕的高度 |

#### 应用程序

|  |  |
| --- | --- |
| 代码 | 功能 |
| [application] fullscreen\_application(number monitor\_id,string title) | 创建全屏应用 |
| [application] window\_application(number width,number height,string title) | 创建窗口应用 |
| [image] make\_image(unsigned char\* data, number width, number height) | 使用原始BGR数据创建图像  每行像素应按4字节对齐  保证会由C++中的delete自动释放 |
| [image] load\_bmp\_image(string path) | 从文件加载 24bit 位图 |
| [vec2] vec2(number a,number b) | 创建二维向量 |
| [vec4] vec4(number a,number b,number c,number d) | 创建四维向量 |
| number get\_framerate() | 获取帧率 |

#### 样式和字体

|  |  |
| --- | --- |
| 代码 | 功能 |
| [font] add\_font(string path,number size) | 增加字体 |
| [font] add\_font\_chinese(string path, number size) | 增加中文字体 |
| [font] add\_font\_default(number size) | 增加默认字体 |
| [font] add\_font\_extend([imgui\_font] data, number size) | 增加扩展中的字体 |
| [font] add\_font\_extend\_cn([imgui\_font] data, number size) | 增加扩展中的中文字体 |
| void push\_font([font]) | 使字体生效 |
| void pop\_font() | 删除当前字体 |
| [font] get\_font() | 获取当前字体 |
| number get\_font\_size() | 获取当前字体大小 |
| void set\_font\_scale(number scale) | 设置字体缩放比例 |
| void style\_color\_classic() | 切换到经典主题 |
| void style\_color\_light() | 切换到亮色主题 |
| void style\_color\_dark() | 切换到暗色主题 |

#### 窗口

|  |  |
| --- | --- |
| 代码 | 功能 |
| void set\_next\_window\_pos([vec2] pos) | 设置下一个窗口位置 |
| void set\_window\_pos([vec2] pos) | 设置当前窗口位置 |
| number get\_window\_pos\_x() | 获取当前窗口位置x坐标 |
| number get\_window\_pos\_y() | 获取当前窗口位置y坐标 |
| void set\_next\_window\_size([vec2] size) | 设置下一个窗口大小 |
| void set\_window\_size([vec2] size) | 设置当前窗口大小 |
| void set\_next\_window\_collapsed(boolean collapsed) | 设置下一个窗口展开 |
| void set\_window\_collapsed(boolean collapsed) | 设置当前窗口展开 |
| void set\_next\_window\_focus() | 设置下一个窗口为焦点 |
| void set\_window\_focus() | 设置当前窗口为焦点 |
| void set\_window\_font\_scale(number scale) | 设置当前窗口字体缩放比例 |
| number get\_window\_width() | 获取当前窗口宽度 |
| number get\_window\_height() | 获取当前窗口高度 |
| void show\_demo\_window(boolean is\_open) | 打开示例窗口 |
| void show\_about\_window(boolean is\_open) | 打开关于窗口 |
| void show\_metrics\_window(boolean is\_open) | 打开指标窗口 |
| void show\_style\_editor() | 打开主题编辑器 |
| boolean show\_style\_selector(string label) | 主题选择器控件 |
| void show\_font\_selector(string label) | 字体选择器控件 |
| void show\_user\_guide() | 显示用户操作指引 |
| void begin\_window(string str,boolean is\_open,array flags) | 开始新窗口布局 |
| void end\_window() | 结束窗口布局 |
| void begin\_child(string str) | 开始新子滚动区域 |
| void end\_child() | 结束子滚动区域 |

#### 布局

|  |  |
| --- | --- |
| 代码 | 功能 |
| void begin\_group() | 开始新组 |
| void end\_group() | 结束组 |
| void separator() | 横向分割线 |
| void same\_line() | 设置下一个控件为同一行 |
| void spacing() | 插入空格 |
| void indent() | 缩进 |
| void unindent() | 反缩进 |

#### 标识符

标识符一般用于右键菜单等，默认情况下标识符就是控件名，但有些控件无 ID 时就要特别标示

一般控件名可使用 Text##ID 的形式指定控件的标识符

|  |  |
| --- | --- |
| 代码 | 功能 |
| void push\_id(string id) | 创建新标识符 |
| void pop\_id() | 结束标识符 |

#### 控件

按钮类控件会在被按下时返回真

输入框需要指定字符缓冲区的大小

|  |  |
| --- | --- |
| 代码 | 功能 |
| void text(string str) | 文本控件 |
| void text\_colored([vec4] color, string str) | 带颜色的文本控件 |
| void text\_disabled(string str) | 禁用的文本控件 |
| void text\_wrappered(string str) | 自动折行文本控件 |
| void label\_text(string label, string str) | 标签文本控件 |
| void bullet\_text(string str) | 圆圈文本控件 |
| boolean button(string str) | 按钮 |
| boolean small\_button(string str) | 小按钮 |
| boolean arrow\_button(string str,[dir] dir) | 箭头按钮 |
| void image([image] img,[vec2] size) | 图片 |
| boolean image\_button([image] img,[vec2] size) | 图片按钮 |
| void check\_box(string str,boolean val) | 多选框 |
| void radio\_button(string str,number v,number v\_button) | 单选框 |
| void plot\_lines(string label, string text, array data) | 折线图 |
| void plot\_histogram(string label, string text, array data) | 直方图 |
| void progress\_bar(number fraction,string overlay) | 进度条，进度的范围是0~1 |
| void bullet() | 圆圈提示控件，会自动插入same\_line() |
| void combo\_box(string str,number current,array items) | 下拉框 |
| void drag\_float(const string label, number n) | 拖动条 |
| void slider\_float(string str,number n,number min,number max) | 滑动块 |
| void input\_text(string str,string text,number buff\_size) | 输入框 |
| void input\_text\_hint(string str, string hint, string text, number buff\_size) | 带有提示的输入框 |
| void input\_text\_multiline(string str,string text,number buff\_size) | 多行输入框 |
| void color\_edit3(string str,[vec4] color) | 三色色彩编辑器 |
| void color\_edit4(string str,[vec4] color) | 四色色彩编辑器 |
| void selectable(string str,boolean selected) | 可选控件 |
| void list\_box(string str,number current,array items) | 列表控件 |

#### 提示信息

|  |  |
| --- | --- |
| 代码 | 功能 |
| void set\_tooltip(string str) | 设置提示信息 |
| void begin\_tooltip() | 开始提示信息布局 |
| void end\_tooltip() | 结束提示信息布局 |

#### 树形结构

|  |  |
| --- | --- |
| 代码 | 功能 |
| boolean tree\_node(string label) | 创建新的树节点 |
| void tree\_pop() | 结束树节点 |

注意，只有 tree\_node()返回真时才需要调用 tree\_pop()

#### 菜单

|  |  |
| --- | --- |
| 代码 | 功能 |
| boolean begin\_main\_menu\_bar() | 开始主菜单布局 |
| void end\_main\_menu\_bar() | 结束主菜单布局 |
| boolean begin\_menu\_bar() | 开始窗口菜单布局 |
| void end\_menu\_bar() | 结束窗口菜单布局 |
| boolean begin\_menu(string str,boolean enabled) | 开始菜单项 |
| void end\_menu() | 结束菜单项 |
| boolean menu\_item(string str,string shortcut,boolean enabled) | 菜单项目 |
| void open\_popup(string id) | 显示弹出 |
| boolean begin\_popup(string id) | 开始弹出菜单布局 |
| boolean begin\_popup\_item(string id) | 开始控件弹出菜单布局 |
| boolean begin\_popup\_window() | 开始窗口弹出菜单布局 |
| boolean begin\_popup\_background() | 开始背景弹出菜单布局 |
| boolean begin\_popup\_modal(string title, boolean is\_open, array flags\_arr) | 开始弹出窗口布局 |
| void end\_popup() | 结束弹出布局 |
| void close\_current\_popup() | 关闭当前弹出 |

注意，只有菜单成功打开才需要调用结束函数

#### 标签

|  |  |
| --- | --- |
| 代码 | 功能 |
| boolean begin\_tab\_bar(string id) | 开始标签栏布局 |
| void end\_tab\_bar() | 结束标签栏布局 |
| boolean begin\_tab\_item(string id, boolean is\_open, array flags) | 开始标签页布局 |
| void end\_tab\_item() | 结束标签页布局 |
| void set\_tab\_item\_closed(string id) | 关闭特定标签页 |

注意，只有标签栏或标签页成功打开才需要调用结束函数

#### 表格

|  |  |
| --- | --- |
| 代码 | 功能 |
| void columns(number count, string id, boolean border) | 插入表格 |
| void next\_column() | 进入下一个表格区域 |
| number get\_column\_index() | 获取当前列索引 |
| number get\_column\_width(number index) | 获取指定列索引处的宽度 |
| void set\_column\_width(number index, number width) | 设置指定列索引处的宽度 |
| number get\_column\_offset(number index) | 获取指定列索引处的x偏移量 |
| void set\_column\_offset(number index, number offset) | 设置指定列索引处的x偏移量 |
| number get\_cloumns\_count() | 获取列数量 |

#### 焦点

|  |  |
| --- | --- |
| 代码 | 功能 |
| void set\_scroll\_here() | 将滚动条滚动到当前位置 |
| void set\_keyboard\_focus\_here() | 设置上一个控件为键盘焦点 |

#### 工具

|  |  |
| --- | --- |
| 代码 | 功能 |
| boolean is\_item\_hovered() | 判断控件是否被鼠标悬停 |
| boolean is\_item\_active() | 判断控件是否激活 |
| boolean is\_item\_focused() | 判断控件是否在焦点 |
| boolean is\_item\_clicked(number button) | 判断控件是否被点击(0=左键, 1=右键, 2=中键) |
| boolean is\_item\_visible() | 判断控件是否可见 |
| boolean is\_any\_item\_hovered() | 判断是否有任何控件被鼠标悬停 |
| boolean is\_any\_item\_active() | 判断是否有任何控件激活 |
| boolean is\_any\_item\_focused() | 判断是否有任何控件在焦点 |
| string get\_clipboard\_text() | 获取剪贴板文字 |
| void set\_clipboard\_text(string str) | 设置剪贴板文字 |

#### 输入

|  |  |
| --- | --- |
| 代码 | 功能 |
| number get\_key\_index([keys] key) | 获取特殊键序号 |
| boolean is\_key\_down(number key) | 判断键是否激发 |
| boolean is\_key\_pressed(number key) | 判断键是否按下 |
| boolean is\_key\_released(number key) | 判断键是否松开 |
| boolean is\_mouse\_down(number button) | 判断鼠标按键是否按下(0=左键, 1=右键, 2=中键) |
| boolean is\_any\_mouse\_down() | 判断是否任何鼠标按键被按下 |
| boolean is\_mouse\_clicked(number button) | 判断鼠标按键是否单击(0=左键, 1=右键, 2=中键) |
| boolean is\_mouse\_double\_clicked(number button) | 判断鼠标按键是否双击(0=左键, 1=右键, 2=中键) |
| boolean is\_mouse\_released(number button) | 判断鼠标按键是否释放(0=左键, 1=右键, 2=中键) |
| boolean is\_mouse\_dragging(number button) | 判断鼠标是否拖动(0=左键, 1=右键, 2=中键) |
| number get\_mouse\_pos\_x() | 获取鼠标位置x坐标 |
| number get\_mouse\_pos\_y() | 获取鼠标位置y坐标 |
| number get\_mouse\_drag\_delta\_x() | 获取鼠标拖动x坐标变化值 |
| number get\_mouse\_drag\_delta\_y() | 获取鼠标拖动y坐标变化值 |

#### 画板

|  |  |
| --- | --- |
| 代码 | 功能 |
| void add\_line([vec2] a,[vec2] b,[vec4] color,number thickness) | 画线 |
| void add\_rect([vec2] a,[vec2] b,[vec4] color,number rounding,number thickness) | 绘制矩形线框 |
| void add\_rect\_filled([vec2] a,[vec2] b,[vec4] color,number rounding) | 填充矩形 |
| void add\_quad([vec2] a,[vec2] b,[vec2] c,[vec2] d,[vec4] color,number thickness) | 绘制四边形线框 |
| void add\_quad\_filled([vec2] a,[vec2] b,[vec2] c,[vec2] d,[vec4] color) | 填充四边形 |
| void add\_triangle([vec2] a,[vec2] b,[vec2] c,[vec4] color,number thickness) | 绘制三角形线框 |
| void add\_triangle\_filled([vec2] a,[vec2] b,[vec2] c,[vec4] color) | 填充三角形 |
| void add\_circle([vec2] centre,number radius,[vec4] color,number seg,number thickness) | 画圆 |
| void add\_circle\_filled([vec2] centre,number radius,[vec4] color,number seg) | 填充圆 |
| void add\_text([font] font, number size, [vec2] pos,[vec4] color,string text) | 绘制文字 |
| void add\_image([image] image,[vec2] a,[vec2] b) | 绘制图片 |

#### 应用程序名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| number get\_window\_width([application] app) | 获取主窗口宽度 |
| number get\_window\_height([application] app) | 获取主窗口高度 |
| void set\_window\_size([application] app,number width,number height) | 设置主窗口大小 |
| void set\_window\_title([application] app,string str) | 设置主窗口标题 |
| void set\_bg\_color([application] app,[vec4] color) | 设置主窗口背景色 |
| boolean is\_closed([application] app) | 判断主窗口是否已经关闭 |
| void prepare([application] app) | 准备帧 |
| void render([application] app) | 渲染帧 |

一般情况下一个程序只允许有一个应用程序实例，多个实例的行为未定义

ImGui 要求在渲染前必须准备帧，渲染帧将会呈现当前帧到屏幕上

典型的主循环结构如下

while !app.is\_closed()

app.prepare()

*循环体*

app.render()

end

#### 图像名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| number get\_width([image] image) | 获取图片宽度 |
| number get\_height([image] image) | 获取图片高度 |

#### 方位名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| left | 左 |
| right | 右 |
| up | 上 |
| down | 下 |

#### 窗口参数名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| no\_title\_bar | 无标题栏 |
| no\_resize | 不允许调整大小 |
| no\_move | 不允许移动位置 |
| no\_scroll\_bar | 无滚动条 |
| no\_collapse | 不允许折叠 |
| always\_auto\_resize | 自动调整大小 |
| no\_saved\_settings | 不保存设置 |
| menu\_bar | 开启菜单栏 |
| horizontal\_scroll\_bar | 开启横向滚动条 |

#### 标签参数名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| unsaved\_document | 设置为未保存标签 |
| set\_selected | 选中标签 |

#### 特殊键名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| tab | 制表键 |
| left | 方向左键 |
| right | 方向右键 |
| up | 方向上键 |
| down | 方向下键 |
| page\_up | 向上翻页键 |
| page\_down | 向下翻页键 |
| home | 主页键 |
| end\_key | 结束键 |
| insert | 插入键 |
| delete | 删除键 |
| backspace | 退格键 |
| space | 空格键 |
| enter | 回车键 |
| escape | 回退键 |
| ctrl\_a | Ctrl+A |
| ctrl\_c | Ctrl+C |
| ctrl\_v | Ctrl+V |
| ctrl\_x | Ctrl+X |
| ctrl\_y | Ctrl+Y |
| ctrl\_z | Ctrl+Z |

### 字符图形库扩展(Darwin)

|  |  |
| --- | --- |
| 代码 | 功能 |
| ui | Darwin图形名称空间(3.2.2.1) |
| core | Darwin核心名称空间(3.2.2.2) |
| drawable | Darwin画布名称空间(3.2.2.3) |
| black | 黑色 |
| white | 白色 |
| red | 红色 |
| green | 绿色 |
| blue | 蓝色 |
| pink | 粉色 |
| yellow | 黄色 |
| cyan | 青色 |
| [pixel] pixel(char,[color] front,[color] back) | 创建一个像素 |
| [drawable] picture(number width,number height) | 创建一幅图片(Drawable) |
| void load() | 加载 Darwin 功能 |
| void exit() | 退出 Darwin 功能 |
| boolean is\_kb\_hit() | 判断是否有按键按下 |
| char get\_kb\_hit() | 获取按下的按键 |
| void fit\_drawable() | 使画布适合当前屏幕大小 |
| [drawable] get\_drawable() | 获取画布 |
| void update\_drawable() | 将画布中的内容更新至屏幕上 |
| void set\_frame\_limit(number fps) | 设置帧率 |
| void set\_draw\_line\_precision(number) | 设置画线精度 |

#### Darwin图形名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| void message\_box(string title,string message,string button) | 弹出一个消息对话框 |
| var input\_box(string title,string message,string default,boolean format) | 弹出一个输入对话框 |

#### Darwin核心名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| char get\_char([pixel]) | 获取像素的字符 |
| void set\_char([pixel],char) | 设置像素的字符 |
| [color] get\_front\_color([pixel]) | 获取像素的前景色 |
| void set\_front\_color([pixel],[color]) | 设置像素的前景色 |
| [color] get\_back\_color([pixel]) | 获取像素的背景色 |
| void set\_back\_color([pixel],[color]) | 设置像素的背景色 |

#### Darwin画布名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| void load\_from\_file([drawable],string path) | 从指定路径加载图片(Darwin CDPF 图片文件) |
| void save\_to\_file([drawable],string path) | 将图片保存至指定路径(Darwin CDPF 图片文件) |
| void clear([drawable]) | 清空画布 |
| void fill([drawable],[pixel]) | 填充画布 |
| void resize([drawable],number width,number height) | 重新设置画布大小 |
| number get\_width([drawable]) | 获取画布宽度 |
| number get\_height([drawable]) | 获取画布高度 |
| [pixel] get\_pixel([drawable],number x,number y) | 获取画布上的点 |
| void draw\_pixel([drawable],number x,number y,[pixel]) | 在画布上画点 |
| void draw\_line([drawable],number x1,number y1,number x2,number y2,[pixel]) | 在画布上画线 |
| void draw\_rect([drawable],number x,number y,number width,number height,[pixel]) | 在画布上绘制线框 |
| void fill\_rect([drawable],number x,number y,number width,number height,[pixel]) | 在画布上填充矩形 |
| void draw\_triangle([drawable],number x1,number y1,number x2,number y2,number x3,number y3,[pixel]) | 在画布上绘制三角形 |
| void fill\_triangle([drawable],number x1,number y1,number x2,number y2,number x3,number y3,[pixel]) | 在画布上填充三角形 |
| void draw\_string([drawable],number x,number y,string,[pixel]) | 在画布上绘制文字 |
| void draw\_picture([drawable],number x,number y,[drawable]) | 将一幅图片绘制到画布上 |

### 正则表达式扩展(Regex)

|  |  |
| --- | --- |
| 代码 | 功能 |
| result | 正则匹配结果名称空间(3.2.3.1) |
| [regex] build(string) | 构建一个正则表达式 |
| [result] match([regex],string) | 匹配正则表达式到整个字符序列 |
| [result] search([regex],string) | 匹配正则表达式到字符序列的任何部分 |
| string replace([regex],string str,string fmt) | 以格式化的替换文本来替换正则表达式匹配的出现位置 |

#### 正则匹配结果名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| boolean ready([result]) | 检查结果是否合法 |
| boolean empty([result]) | 检查匹配是否成功 |
| number size([result]) | 获取完全建立的结果状态中的匹配数 |
| number length([result],number) | 获取特定子匹配的长度 |
| number position([result],number) | 获取特定子匹配首字符的位置 |
| string str([result],number) | 获取特定子匹配的字符序列 |
| string prefix([result]) | 获取目标序列起始和完整匹配起始之间的子序列 |
| string suffix([result]) | 获取完整匹配结尾和目标序列结尾之间的子序列 |

### Base64编解码扩展(Codec)

|  |  |
| --- | --- |
| 代码 | 功能 |
| base32 | Base32编码译码器名称空间(3.2.4.1) |
| base64 | Base64编码译码器名称空间(3.2.4.2) |

#### Base32编码译码器名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| standard | 标准编码译码器(RFC4648) |
| rfc4648 | RFC4648编码译码器 |
| crockford | Crockford编码译码器 |
| hex | 二进制编码译码器 |

#### Base64编码译码器名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| standard | 标准编码译码器(RFC4648) |
| rfc4648 | RFC4648编码译码器 |
| url | 为链接优化的RFC4648编码译码器 |
| url\_unpadded | 为链接优化的RFC4648编码译码器，无对齐符号= |

#### 译码器函数

|  |  |
| --- | --- |
| 代码 | 功能 |
| string encode([codec], string) | 编码 |
| string decode([codec], string) | 译码 |

### 流式API扩展(Streams)

|  |  |
| --- | --- |
| 代码 | 功能 |
| [streams] of(list) | 从 list 构建一个 stream 对象 |
| void for\_each([streams],function action) | 对流中每一个元素执行一次 action |
| [streams] peek([streams],function action) | 与 for\_each 功能相同,但 peek 不会终止流 |
| number count([streams]) | 获取当前流中可操作元素的数目 |
| [streams] skip([streams],number) | 跳过前 n 个元素 |
| [streams] reverse([streams]) | 将流中的元素顺序颠倒 |
| [streams] filter([streams],function predicate) | 用当前流中的所有符合条件的元素创建一个新流 |
| [streams] map([streams],function mapper) | 对流中的元素执行 mapper,并将结果创建一个新流 |
| [streams] reduce([streams],identity,function accumulator) | 对流中的元素执行 accumulator,并将结果返回 |
| [streams] limit([streams],number) | 设置流中可操作元素的数目 |
| boolean any\_match([streams],function predicate) | 只要流中有元素符合 predicate,就返回 true |
| boolean all\_match([streams],function predicate) | 只有流中所有元素都符合 predicate,才返回 true |
| boolean none\_match([streams],function predicate) | 只有流中所有元素都不符合 predicate,才返回 true |
| var find\_any([streams]) | 从流中任意选取一个元素返回 |
| var find\_first([streams]) | 获取流中第一个元素 |
| list to\_list([streams]) | 将当前流中可操作元素作为一个 list 返回 |

### 数据库扩展(SQLite)

|  |  |
| --- | --- |
| 代码 | 功能 |
| statement | SQLite 语句名称空间(3.2.6.1) |
| integer | SQLite 整数类型 |
| real | SQLite 浮点类型 |
| text | SQLite 文本类型 |
| [sqlite] open(string path) | 打开或创建一个 SQLite 数据库 |
| [statement] prepare([sqlite] database,string sql) | 准备一个 SQLite 语句 |

#### SQLite 语句名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| boolean done([statement]) | 判断是否执行完毕 |
| void reset([statement]) | 重置语句 |
| void exec([statement]) | 执行语句 |
| number column\_count([statement]) | 获取记录数量 |
| [type] column\_type([statement],number index) | 获取记录类型 |
| string column\_name([statement],number index) | 获取记录名称 |
| string column\_decltype([statement],number index) | 推导记录类型 |
| number column\_integer([statement],number index) | 获取整数记录 |
| number column\_real([statement],number index) | 获取浮点记录 |
| string column\_text([statement],number index) | 获取文本记录 |
| number bind\_param\_count([statement]) | 绑定参数数量 |
| void bind\_integer([statement],number index,number data) | 绑定整数参数 |
| void bind\_real([statement],number index,number data) | 绑定浮点参数 |
| void bind\_text([statement],number index,string data) | 绑定文本参数 |
| void clear\_bindings([statement]) | 清除所有绑定 |

### 套接字扩展(Network)

|  |  |
| --- | --- |
| 代码 | 功能 |
| tcp | TCP 套接字名称空间(3.2.7.1) |
| udp | UDP 套接字名称空间(3.2.7.2) |
| string host\_name() | 获取主机名 |

#### TCP 套接字名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| socket | TCP 套接字类型 |
| [tcp::acceptor] acceptor([tcp::endpoint]) | 在端点上建立接收器 |
| [tcp::endpoint] endpoint(string address,number port) | 在指定 IP 地址和端口上建立端点 |
| [tcp::endpoint] endpoint\_v4(number port) | 在任意 IPv4 地址上和指定端口上建立端点 |
| [tcp::endpoint] endpoint\_v6(number port) | 在任意 IPv6 地址上和指定端口上建立端点 |
| [tcp::endpoint] resolve(string host,string service) | 解析主机的服务并建立端点 |

##### TCP 套接字类型扩展

|  |  |
| --- | --- |
| 代码 | 功能 |
| void set\_timeout([tcp::socket], number time) | 设置超时时间，单位为毫秒 |
| void connect([tcp::socket],[tcp::endpoint]) | 连接到端点 |
| void accept([tcp::socket],[tcp::acceptor]) | 接受一个 TCP 请求 |
| void close([tcp::socket]) | 关闭套接字 |
| boolean is\_open([tcp::socket]) | 查询套接字是否打开 |
| string receive([tcp::socket],number buffer\_size) | 接收一些数据,最大长度为buffer\_size |
| void send([tcp::socket],string buffer) | 发送一些数据 |
| [tcp::endpoint] remote\_endpoint([tcp::socket]) | 获取远程端点 |

##### TCP 端点类型扩展

|  |  |
| --- | --- |
| 代码 | 功能 |
| string address([tcp::endpoint]) | 获取端点指向的地址 |
| number port([tcp::endpoint]) | 获取端点指向的端口 |

#### UDP 套接字名称空间

|  |  |
| --- | --- |
| 代码 | 功能 |
| socket | UDP 套接字类型 |
| [udp::endpoint] endpoint(string address,number port) | 在指定 IP 地址和端口上建立端点 |
| [udp::endpoint] endpoint\_v4(number port) | 在任意 IPv4 地址上和指定端口上建立端点 |
| [udp::endpoint] endpoint\_v6(number port) | 在任意 IPv6 地址上和指定端口上建立端点 |
| [udp::endpoint] resolve(string host,string service) | 解析主机的服务并建立端点 |

##### UDP 套接字类型扩展

|  |  |
| --- | --- |
| 代码 | 功能 |
| void open\_v4([udp::socket]) | 打开 IPv4 协议套接字 |
| void open\_v6([udp::socket]) | 打开 IPv6 协议套接字 |
| void bind([udp::socket],[udp::endpoint]) | 将套接字绑定到端点上 |
| void close([udp::socket]) | 关闭套接字 |
| boolean is\_open([udp::socket]) | 查询套接字是否打开 |
| string receive\_from([udp::socket],number buffer\_size,[udp::endpoint]) | 从远程端点处接收一些数据,最大长度为 buffer\_size |
| void send\_to([udp::socket],string buffer,[udp::endpoint]) | 发送一些数据到远程端点 |

##### UDP 端点类型扩展

|  |  |
| --- | --- |
| 代码 | 功能 |
| string address([udp::endpoint]) | 获取端点指向的地址 |
| number port([udp::endpoint]) | 获取端点指向的端口 |

注意，TCP 套接字和 UDP 套接字的端点不能通用

# 解释器

## 命令行参数

### 解释器(Cs)

cs [选项...] <文件> [参数...]

|  |  |  |
| --- | --- | --- |
| 选项 | 助记符 | 功能 |
| --compile-only | -c | 仅编译 |
| --no-optimize | -o | 禁用优化器 |
| --help | -h | 显示帮助信息 |
| --version | -v | 显示版本信息 |
| --wait-before-exit | -w | 等待进程退出 |
| --dump-ast | -d | 导出高级语法树 |
| --log-path <PATH> | -l <PATH> | 设置日志和语法树导出路径 |
| --import-path <PATH> | -i <PATH> | 设置引入查找路径 |

注意，若不设置日志和导出AST路径，这两者将直接输出至标准输出流

### 交互式解释器(Cs REPL)

cs\_repl [选项...]

|  |  |  |
| --- | --- | --- |
| 选项 | 助记符 | 功能 |
| --help | -h | 显示帮助信息 |
| --version | -v | 显示版本信息 |
| --silent | -s | 关闭命令提示符 |
| --wait-before-exit | -w | 等待进程退出 |
| --args <...> | -a <...> | 设置程序参数 |
| --log-path <PATH> | -l <PATH> | 设置日志路径 |
| --import-path <PATH> | -i <PATH> | 设置引入查找路径 |

注意：

1. 在选项--args或其助记符-a之后设置的每一项都将被视为参数
2. 若不设置日志路径，将直接输出至标准输出流

### 调试器(Cs Debugger)

cs\_dbg [选项...] <文件>

|  |  |  |
| --- | --- | --- |
| 选项 | 助记符 | 功能 |
| --help | -h | 显示帮助信息 |
| --version | -v | 显示版本信息 |
| --wait-before-exit | -w | 等待进程退出 |
| --log-path <PATH> | -l <PATH> | 设置日志路径 |
| --import-path <PATH> | -i <PATH> | 设置引入查找路径 |

注意，若不设置日志路径，将直接输出至标准输出流

#### 调试器命令

|  |  |  |
| --- | --- | --- |
| 完整命令 | 短命令 | 功能 |
| quit | q | 退出调试器 |
| help | h | 显示帮助信息 |
| next | n | 执行下一个语句 |
| step | s | 执行下一个语句并进入函数 |
| continue | c | 继续运行程序直到下一个断点被击中 |
| backtrace | bt | 显示函数调用栈 |
| break [*行号*|*函数名*] | b | 设置断点 |
| lsbreak | lb | 显示已有断点 |
| rmbreak [*断点号*] | rb | 移除断点 |
| print [*表达式*] | p | 对表达式求值并显示 |
| optimizer [on|off] | o | 关闭或打开优化器，默认为开 |
| run <*运行参数*…> | r | 运行程序 |

## 解释器扩展

### 基于CNI标准扩展的Covariant Script解释器扩展

C/C++ Native Interface Standard Extension，即CNI标准扩展，是Covariant Script解释器扩展的一部分，旨在降低中低复杂度的Covariant Script语言扩展的编写难度

#### 头文件

<covscript/dll.hpp>

**Covariant Script扩展标准头文件，定义了必须的扩展入口函数**

<covscript/cni.hpp>

**Covariant Script CNI标准扩展头文件，定义了所有的标准类型转换规则和CNI组成宏**

#### 标准类型转换规则

CNI标准类型转换规则规定了C/C++中类型到Covariant Script类型的自动转换规则

|  |  |
| --- | --- |
| C/C++类型 | Covariant Script类型 |
| bool | cs::boolean |
| signed short | cs::number |
| unsigned short |
| signed int |
| unsigned int |
| signed long |
| unsigned long |
| signed long long |
| unsigned long long |
| float |
| double |
| long double |
| signed char | cs::char |
| unsigned char |
| const char\* | cs::string |
| std::string |

自动转换指的是当CNI调用时若遇到上表中的C/C++类型，可接受对应的Covariant Script类型作为兼容类型并自动进行转换，当函数返回时将直接按照表将相应的C/C++类型转换为Covariant Script类型

在引入头文件之前定义CNI\_DISABLE\_STD\_CONVERSION宏可以禁用标准类型转换规则

#### CNI组成宏

C/C++ Native Interface Composer Macros，即CNI组成宏，是CNI标准扩展的一部分，旨在用简单易用的宏定义代替繁杂的CNI声明

##### CNI根名称空间

CNI\_ROOT\_NAMESPACE {

// TODO

}

**声明一个CNI根名称空间**

CNI根名称空间是所有CNI组成宏的实现基础，除特殊说明外所有CNI组成宏必须写在CNI根名称空间内

CNI根名称空间在每个扩展中只允许存在一个，其对应着C++名称空间名cni\_root\_namespace

CNI根名称空间会自动完成Covariant Script扩展标准头文件中定义的扩展入口函数并添加CNI组成宏必需组件

##### 声明CNI函数

CNI(*函数名*)

**向当前名称空间添加一个CNI函数**

CNI\_V(*函数名*, *函数*)

**向当前名称空间添加一个CNI函数，指定函数实现，可以是Lambda表达式**

CNI\_CONST(*函数名*)

**向当前名称空间添加一个CNI常量函数**

CNI\_CONST\_V(*函数名*, *函数*)

**向当前名称空间添加一个CNI常量函数，指定函数实现，可以是Lambda表达式**

这里的CNI常量函数指的是简单的、可以被编译期调用的函数，如加减法等

一般要求函数不可有阻塞的操作，不可执行过长时间

只有常量函数会被编译器尝试进行常量折叠优化

##### 声明CNI变量

CNI\_VALUE(*变量名*, *值*)

**向当前名称空间添加一个CNI变量，类型遵从**[**标准类型转换规则**](#_标准类型转换规则)

CNI\_VALUE\_V(*变量名*, *类型*, *值*)

**向当前名称空间添加一个CNI变量，指定其类型**

CNI\_VALUE\_CONST(*变量名*, *值*)

**向当前名称空间添加一个CNI常量，类型遵从**[**标准类型转换规则**](#_标准类型转换规则)

CNI\_VALUE\_CONST\_V(*变量名*, *类型*, *值*)

**向当前名称空间添加一个CNI常量，指定其类型**

##### 声明CNI名称空间

CNI\_NAMESPACE(*名称空间名*) {

// TODO

}

**声明一个CNI名称空间**

CNI名称空间可以嵌套定义，其对应的C++名称空间名即为宏中填写的参数

CNI名称空间会自动完成在上层名称空间中的注册工作并添加CNI组成宏必需组件

无论是在C++层面或是Covariant Script层面，在CNI名称空间内定义的CNI变量和CNI函数均属于这个名称空间，如：

CNI\_ROOT\_NAMESPACE {

CNI\_NAMESPACE(test) {

// TODO

}

}

那么常规C++代码在访问test的时候可以这样写：cni\_root\_namespace::test

常规Covariant Script代码在访问test的时候可以这样写：*包名*.test

但CNI变量和CNI函数属于特殊表示，**C++代码无法直接访问**

##### CNI类型扩展

CNI\_TYPE\_EXT(*类型名*, *类型*, *构造方法*) {

// TODO

}

**声明一个CNI类型扩展名称空间，名称空间名与类型名相同**

CNI\_TYPE\_EXT\_V(名称空间名, 类型, 类型名, 构造方法) {

// TODO

}

**声明一个CNI类型扩展名称空间，独立声明名称空间名和类型名**

CNI类型扩展名称空间是一种特殊的名称空间，不仅是一个CNI名称空间，同时也会在父名称空间中声明一个Covariant Script类型，而这个类型的构造函数就会遵从构造方法参数中填写的简短的代码，一般情况下只需调用目标类型的构造函数即可

CNI类型扩展名称空间声明后，可以使用另外一个特殊的宏启用CNI类型扩展

CNI\_ENABLE\_TYPE\_EXT(*名称空间*, *类型*)

**在一个名称空间上启用类型扩展支持**

CNI\_ENABLE\_TYPE\_EXT\_V(*名称空间*, *类型*, *类型名*)

**在一个名称空间上启用类型扩展支持，独立声明类型名**

这两个宏非常特殊，**必须在CNI根名称空间后编写**

名称空间无需手动加上cni\_root\_namespace，但若存在多级名称空间则必须写全

可以在普通的名称空间上开启类型扩展

CNI开启类型扩展后，会将第一个参数视为this，但传入的是this指向的对象

除特殊要求外，一般无需专为CNI写成员函数的转发，只需传入指向成员函数地址的**原始函数指针**(不能是抹除了类型信息的void\*等)，CNI即可自动侦测并进行调用

# 版权所有 © 2019 智锐科创

**此文档基于智锐科创通用公共许可证分发，使用此文档即代表你同意并遵守智锐科创通用公共许可证的所有条款，保留所有权利**

## 著作权信息

编著 李登淳

审校 史为成 李登 李志成

## **版本信息**

Covariant Script标准 190501

文档修订版本 190506

## 版权信息

Copyright © 2019 Michael Lee(李登淳)

Licensed under the Covariant Innovation General Public License,

Version 1.0 (the "License");

you may not use this file except in compliance with the License.

You may obtain a copy of the License at

<https://covariant.cn/licenses/LICENSE-1.0>

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.