Covariant Script 编程语言

标准文档(201201)

目录

3. API

3.1. 标准库

1. 类型

1.1. 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

1.2. 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）

1.3. 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

注意，**小心环形引用行为**，这将造成不可预估的内存泄漏

1.4. char（字符）类型

char 的字面量是由单引号括起的单个字符，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）

1.5. string（字符串）类型

`string` 的字面量是由双引号括起的任意个数字符，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`(字符串长度-1)`，小于零时即为倒序访问。多段字符串可以使用加运算符连接

字符串后可跟字面量名，如`"0xff"hex`，称为字符串字面量。用法详见注册字面量

1.6. 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`

1.7. 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

1.8. 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）

1.9. 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。也可以使用点运算符访问散列表中的字符串键对应的值，如与散列表扩展函数冲突则被忽略，若键不存在将触发运行时错误

例如，`map[“hello”]` 可使用 `map.hello` 代替，但 `map.hello` 不会自动新建映射

2. 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

2.1. 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

2.2. 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

@charset: *字符集*

指定程序的编码，可以是：

编码	字符集
ASCII 纯文本	ascii
UTF-8 简体中文	utf-8
GBK 简体中文	gbk

当选择 Unicode 中文编码(UTF-8/GBK)时，**可以使用简体中文作为标识符**

2.3. 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)

import	as	package	namespace	using	struct	class	extends
引入包		包定义	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

2.4. 模块

`import` *`Package 名, Package 名...`*

引入一个或多个 Package

`import` *`Package 名[(.名称空间名)...]`* `as` *`Package 别名`*

引入一个 Package 或 Package 内的名称空间，并设置别名

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

2.5. 变量

2.5.1. 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...*

[慎用] 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

请**尽量谨慎使用常量**，复杂常量的行为取决于实现

2.5.2. 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

2.5.3. 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

2.6. 表达式

2.6.1. 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

2.6.2. 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

2.6.3. 二元运算符

2.6.3.1. 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*			

2.6.3.2. 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

2.6.3.3. 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看作以一个整体进行求值

2.6.4. 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

2.6.5. 结构化绑定

(*表达式 1*, *表达式 2*...) = *表达式 3*

结构化绑定(Structured Binding)指的是将数组中的值按位置绑定至小括号括起的逗号表达式列表中，如：`(a, b) = {1, 2}`

我们将结构化绑定中等号左侧的小括号括起的逗号表达式列表称为结构化绑定列表

在结构化绑定中要求表达式 3(等号右侧)的计算结果必须为数组

结构化绑定也可以嵌套，但要求嵌套的结构化绑定中对应位置的数组必须也是嵌套的，如：`(a, (b, c)) = {1, {2, 3}}`

结构化绑定也可以用于变量和常量的声明，但结构化绑定列表中的元素必须为变量名或嵌套的结构化绑定列表，如：

```
var (a, (b, c)) = test(....)

constant (d, (e, f)) = {4, {5, 6}}
```

如果是常量的声明，则被绑定的数组必须为常量

2.7. 作用域和名称空间

2.7.1. 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

2.7.2. 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

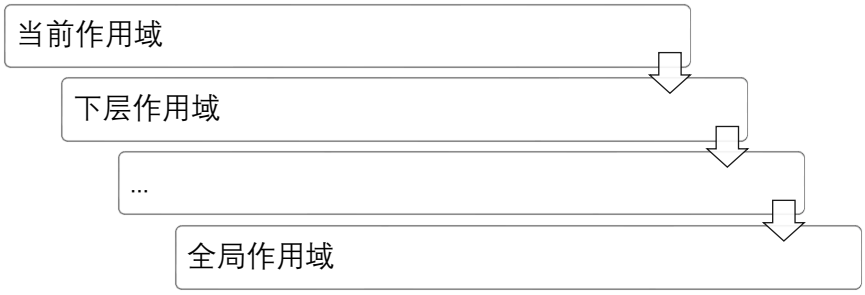
名称空间名`.`*变量名*

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()`

除此之外，支持扩展的类型将自动调用无参数的成员访问函数

也就是说：`string.size(str)` 等价于 `str.size`

2.8. 语句

2.8.1. 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

2.8.2. 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

注意，请勿在 foreach 中修改被遍历的容器，否则将导致未定义行为

2.8.3. 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

2.9. 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
```

函数在被调用时，将传入参数的引用，返回时也将返回引用

2.9.1. 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

编译器会为 Lambda 表达式插入一个隐式的 `self` 参数用于表示自身

2.9.2. 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展开

2.10. 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

2.11. 结构和类

2.11.1. 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

2.11.2. 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`

2.11.3. 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

```
function to_string()
    语句块
    return new string
end
```

字符串转换函数，结构被转换成字符串时调用，必须返回字符串

3. 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。函数功能若标记为*，则说明这是一个成员访问函数

3.1. 标准库

3.1.1. 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
var move(var)	将变量标记为右值
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)
```

3.1.2. 异常(Exception)

代码	功能
`string what([exception])`	*获取异常详情

3.1.3. 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)
`char_buff`	字符缓冲区类型(3.1.3.5)
`[istream] ifstream(string path)`	新建一个输入文件流(`openmode.in`)
`[ostream] ofstream(string path)`	新建一个输出文件流(`openmode.out`)
`[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)

3.1.3.1. 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

3.1.3.2. 打开方式名称空间

代码	功能
`in`	为读打开(输入流)
`bin_in`	为读打开(输入流，二进制)
`out`	为写打开(清空内容，输出流)
`bin_out`	为写打开(清空内容，输出流，二进制)
`app`	为写打开(追加内容，输出流)
`bin_app`	为写打开(追加内容，输出流，二进制)

3.1.3.3. 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

3.1.3.4. 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)

3.1.3.5. 字符缓冲区类型

代码	功能
`[istream] get_istream([char_buff])`	转换至输入流
`[ostream] get_ostream([char_buff])`	转换至输出流
`string get_string([char_buff])`	将实际缓冲区转换为字符串

3.1.4. 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
`boolean is_platform_windows()`	判定是否为 Windows 平台
`boolean is_platform_linux()`	判定是否为 Linux 平台
`boolean is_platform_darwin()`	判断是否为 macOS 平台
`boolean is_platform_unix()`	判断是否为 Unix 兼容平台

3.1.4.1. 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

3.1.4.2. 文件名称空间

代码	功能
`boolean copy(string path, string path)`	复制文件
`boolean remove(string path)`	删除文件
`boolean exists(string path)`	判断文件是否存在
`boolean rename(string path, string path)`	重命名/移动文件
`boolean is_file(string path)`	判定一个路径是否为文件
`boolean is_directory(string path)`	判定一个路径是否为目录
`boolean can_read(string path)`	判定一个路径是否可读
`boolean can_write(string path)`	判定一个路径是否可写
`boolean can_execute(string path)`	判定一个文件是否为可执行文件
`boolean mkdir(string path)`	创建一个目录，成功返回真
`boolean mkdir_p(string path)`	递归创建一个目录，成功返回真
`boolean chmod(string path, string mod)`	更改路径权限
`boolean chmod_r(string path, string mod)`	递归更改路径权限

注意，更改路径权限这里，权限使用特定格式的字符串指定，格式如下:

字符串可以分为三组字符，每一组有三个，每个字符都代表不同的权限，分别为读取(r)、写入(w)和执行(x)，例如“rwxr-xr--”

a) 第一组字符(1-3)表示文件所有者的权限，这里表示所有者拥有读取(r)、写入(w)和执行(x)的权限

b) 第二组字符(4-6)表示文件所属用户组的权限，这里表示该组拥有读取(r)和执行(x)的权限，但没有写入权限

c) 第三组字符(7-9)表示所有其他用户的权限，这里表示其他用户只能读取(r)文件

3.1.4.3. 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

3.1.4.3.1. 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

3.1.4.3.2. 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

3.1.5. 运行时(Runtime)

代码	功能
time_type	时间类型名称空间(3.1.5.1)
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
[time_type] local_time()	获取当地时间和日期
[time_type] utc_time()	获取 UTC 时间和日期
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] import([context], string folder_path, string name)	从 folder_path 动态加载一个扩展，规则与 import 语句相同
[namespace] source_import([context], string path)	动态加载一个扩展，需指定详细路径和文件名
number argument_count([function] func)	获取一个函数的参数数量
void add_literal([context], string literal, [function] func)	注册字面量处理函数
string get_current_dir()	获取当前执行路径

代码	功能
`var wait_for(number time, [function] func, array args)`	使用传入的参数执行函数，等待指定时间(毫秒)后若函数仍未完成则抛出异常
`var wait_until(number time, [function] func, array args)`	使用传入的参数执行函数，直到指定时间(毫秒)后若函数仍未完成则抛出异常

3.1.5.1. 时间类型名称空间

代码	功能
`number sec([time_type])`	*分后之秒，范围是[0, 60]
`number min([time_type])`	*时后之分，范围是[0, 59]
`number hour([time_type])`	*自午夜起之时，范围是[0, 23]
`number wday([time_type])`	*自星期日起之日，范围是[0, 6]
`number mday([time_type])`	*月之日，范围是[1, 31]
`number yday([time_type])`	*自 1 月 1 日起之日，范围是[0, 365]
`number mon([time_type])`	*自一月起之月，范围是[0, 11]
`number year([time_type])`	*自 1900 起之年
`Boolean is_dst([time_type])`	*是否为夏令时

3.1.6. 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

3.1.6.1. 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)

3.1.7. 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

3.1.8. 字符串(String)

代码	功能
`void assign(string, number index,char ch)`	在字符串 index 处给字符赋值
`string append(string, var)`	在尾部追加内容
`string insert(string, number index, var)`	在指定位置处插入内容
`string erase(string, number begin, number end)`	将范围内的字符删除
`string replace(string, number begin, number count, var)`	将从指定位置开始的指定个数字符替换
`string substr(string, number begin, number count)`	从指定位置截取指定长度的子字符串
`number find(string, string target, number begin)`	从指定位置开始从左向右查找一段字符串
`number rfind(string, string target, number begin)`	从指定位置开始从右向左查找一段字符串

代码	功能
`string cut(string, number count)`	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

3.1.9. 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
`var pop_front(array)`	删除第一个元素并返回
`void push_back(array, var)`	将元素添加到容器末尾
`var pop_back(array)`	删除最后一个元素并返回
`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

3.1.9.1. 数组迭代器名称空间

代码	功能
`[iterator] next([iterator])`	向前移动迭代器，返回移动前的迭代器
`[iterator] next_n([iterator])`	同 next，向前移动 n 个单位
`[iterator] prev([iterator])`	向后移动迭代器
`[iterator] prev_n([iterator])`	同 prev，向后移动 n 个单位
`var data([iterator])`	*访问迭代器指向的元素

3.1.10. 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
var pop_front(list)	删除第一个元素并返回
void push_back(list, var)	将元素添加到容器末尾
var pop_back(list)	删除最后一个元素并返回
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

3.1.10.1. 线性表迭代器名称空间

代码	功能
[iterator] next([iterator])	向前移动迭代器
[iterator] prev([iterator])	向后移动迭代器
var data([iterator])	*访问迭代器指向的元素

3.1.11. 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

3.1.12. 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

3.2. 扩展库

3.2.1. 图形库字体扩展(ImGUI Font)

代码	功能
source_han_sans	思源黑体字体(imgui font)

3.2.2. 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)

3.2.2.1. 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

3.2.2.2. 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 拥有 x、y 两个成员变量
[vec4] vec4(number a, number b, number c, number d)	创建四维向量 拥有 x、y、z、w 四个成员变量
number get_time()	获取帧时间
number get_framerate()	获取帧率

3.2.2.3. 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

3.2.2.4. 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

3.2.2.5. 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

代码	功能
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

3.2.2.6. 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

3.2.2.7. 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

代码	功能
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

3.2.2.8. 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

3.2.2.9. 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()

3.2.2.10. 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

代码	功能
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

3.2.2.11. 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

3.2.2.12. 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

代码	功能
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

3.2.2.13. 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

3.2.2.14. 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

3.2.2.15. 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)

代码	功能
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

3.2.2.16. 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

3.2.2.17. 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
```

3.2.2.18. 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

3.2.2.19. 方位名称空间

代码	功能
left	左
right	右
up	上
down	下

3.2.2.20. 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

3.2.2.21. 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

3.2.2.22. 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

3.2.3. 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(需提前调用 fit_drawable)
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

3.2.3.1. 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

3.2.3.2. 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

代码	功能
void set_back_color([pixel], [color])	设置像素的背景色

3.2.3.3. 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

3.2.4. 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

3.2.4.1. 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])`	检查匹配是否为空(空为失败)
`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

3.2.5. 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)
`json`	Json 解码译码器名称空间(3.2.4.4)

3.2.5.1. 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

3.2.5.2. 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=

3.2.5.3. 译码器函数

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

3.2.5.4. Json 编码译码器名称空间

代码	功能
`value`	Json 值名称空间(3.2.5.4.1)
`[json] from_string(string)`	从字符串新建 Json 值
`[json] from_stream([istream])`	从流新建 Json 值
`[json] from_var(var)`	从 CovScript 变量新建 Json 值
`string to_string([json])`	将 Json 值转换为字符串
`var to_var([json])`	将 Json 值转换为 CovScript 变量

Covariant Script 类型与 `Json` 类型之间的对应关系如下:

Covariant Script 类型	`Json` 类型
`null`	`null`
`number`	`real`
`string`	`string`
`boolean`	`boolean`
`array`	`array`
`hash_map`	`object`

3.2.5.4.1. Json 值名称空间

代码	功能
`[json] make_null()`	创建空值
`[json] make_array()`	创建数组
`[json] make_object()`	创建对象
`[json] make_int(number val)`	创建整数
`[json] make_uint(number val)`	创建无符号整数
`[json] make_real(number val)`	创建浮点数
`[json] make_string(string str)`	创建字符串
`[json] make_boolean(boolean val)`	创建布尔值
`number as_int([json])`	获取整数
`number as_uint([json])`	获取无符号整数
`number as_real([json])`	获取浮点数
`string as_string([json])`	获取字符串
`boolean as_boolean([json])`	获取布尔值
`boolean is_int([json])`	判断是否为整数
`boolean is_uint([json])`	判断是否为无符号整数
`boolean is_real([json])`	判断是否为浮点数
`boolean is_null([json])`	判断是否为空值
`boolean is_array([json])`	判断是否为数组
`boolean is_object([json])`	判断是否为对象
`boolean is_number([json])`	判断是否为数字

代码	功能
boolean is_string([json])	判断是否为字符串
boolean is_boolean([json])	判断是否为布尔值
number arr_size([json])	获取数组大小
boolean arr_empty([json])	判断数组是否为空
void arr_clear([json])	清空数组
void arr_resize([json])	调整数组大小，多余填充空值
[json] arr_append([json] this, [json] value)	在尾部添加新值
[json] arr_get([json], number)	获取指定下标的值
void arr_set([json], number idx, [json] value)	设置指定下标的值
[json] get_member([json] this, string key)	获取对象成员
[json] set_member([json] this, string key, [json] value)	设置对象成员
boolean has_member([json] this, string key)	判断是否存在成员
array get_member_names([json])	获取成员名称列表
void to_stream([json] this, [ostream] os)	将 Json 值输出至流

注意，这里的数组操作以及成员操作，仅值为数组或成员时可用

3.2.6. 流式 API 扩展(Stream)

代码	功能
`[stream] of(list)`	从 list 构建一个有限流
`[stream] iterate(head, function iterator)`	对流中的当前元素反复应用 iterator 函数构造无限流
`[stream] repeat(elem)`	构造一个元素全为 elem 的无限流
`void for_each([stream], function action)`	对流中每一个元素执行一次 action。当流为无限流时，该函数不返回
`[stream] peek([stream], function action)`	与 for_each 功能相同,但 peek 不会终止流。当流为无限流时，该函数不返回
`number count([stream])`	获取当前流中可操作元素的数目，当流为无限流时，该函数不返回
`[stream] skip([stream], number)`	跳过前 n 个元素
`[stream] filter([stream], function predicate)`	过滤流中的元素。只保留满足 predicate 的元素
`[stream] map([stream], function mapper)`	对流中的元素执行 mapper,并将结果创建一个新流
`[stream] reduce([stream], identity, function accumulator)`	对流中的元素执行 accumulator,并将结果返回。当流为无限流时，该函数不返回
`boolean any_match([stream], function predicate)`	只要流中有元素符合 predicate,就返回 true。当流为无限流时，该函数可能不返回
`boolean all_match([stream], function predicate)`	只有流中所有元素都符合 predicate,才返回 true。当流为无限流时，该函数可能不返回
`boolean none_match([stream], function predicate)`	只有流中所有元素都不符合 predicate,才返回 true。当流为无限流时，该函数可能不返回
`var find_any([stream])`	从流中任意选取一个元素返回（即第一个元素）（已废弃的 API）
`var find_first([stream])`	获取流中第一个元素
`[stream] take_while(function predicate)`	将流最前方满足 predicate 的所有元素取出构造有限流。当 predicate 第一次返回 false 时，该函数返回新流
`[stream] take(n)`	将流前方 n 个元素取出构造有限流
`list to_list([stream])`	将当前流中可操作元素作为一个 list 返回，当流为无限流时，该函数不返回

3.2.7. 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

3.2.7.1. 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

3.2.8. 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

3.2.8.1. 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

3.2.8.1.1. TCP 套接字类型扩展

代码	功能
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

3.2.8.1.2. 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

3.2.8.2. 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 地址的指定端口上建立端点
[udp::endpoint] endpoint_broadcast(number port)	在 IPv4 广播地址的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

3.2.8.2.1. 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
void set_opt_reuse_address(boolean)	设置地址重用选项
void set_opt_broadcast(boolean)	设置广播选项
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

3.2.8.2.2. 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

3.2.9. 进程扩展(Process)

代码	功能
builder	进程构建器类型
builder_type	构建器名称空间
process_type	进程名称空间
[process] exec(string command, array arguments)	使用默认设置启动一个进程

3.2.9.1. 构建器名称空间

代码	功能
[builder] cmd([builder], string command)	设置启动命令
[builder] arg([builder], array arguments)	设置启动参数
[builder] dir([builder], string directory)	设置启动路径
[builder] env([builder], string key, string value)	设置环境变量
[builder] merge_output ([builder], bool value)	合并输出流和错误流
[process] start([builder])	启动进程，返回进程对象

3.2.9.2. 进程名称空间

代码	功能
[ostream] in([process])	获取标准输入流
[istream] out([process])	获取标准输出流
[istream] err([process])	获取标准错误流
number wait([process])	等待退出并获取退出码
bool has_exited([process])	判定进程是否已经推出
void kill([process], bool force)	杀死进程，参数指定是否强制退出

3.2.10. 压缩文件扩展(Zip)

本扩展支持 Zip 格式的压缩文件，详见：

https://en.wikipedia.org/wiki/ZIP_(file_format)

代码	功能
openmode	打开方式名称空间(3.2.10.1)
entry_type	入口名称空间(3.2.10.2)
zip_type	压缩文件名称空间(3.2.10.3)
bool extract(string zip_path, string target_path)	解压一个 Zip 文件到目标目录，返回是否解压成功
[zip] open(string path, [openmode] mode)	按 mode 参数指定的方式打开一个 Zip 文件，返回压缩文件类型

3.2.10.1. 打开方式名称空间

代码	功能
read	读取
write	新建
append	追加

3.2.10.2. 入口名称空间

代码	功能
string name([entry])	获取入口名称
boolean is_dir([entry])	返回是否为目录
number size([entry])	当入口为文件时，返回文件大小
string crc32([entry])	当入口为文件时，返回 CRC32 校验码

3.2.10.3. 压缩文件名称空间

代码	功能
bool is_open([zip])	返回文件是否打开
array get_entries([zip])	打开方式为读时，返回文件入口数组
boolean read_entry_stream([zip], string path, [ostream])	读取文件到输出流，返回是否成功
boolean write_entry_stream ([zip], string path, [istream])	从输入流读取内容并写入到文件，返回是否成功
boolean entry_add(string path, string target)	添加文件到压缩文件中，返回是否成功
boolean entry_extract(string path, string target)	从压缩文件中解压文件，返回是否成功
boolean entry_delete(string)	删除压缩文件中的入口

3.2.11. cURL 网络扩展(cURL)

代码	功能
ssl_level	SSL 选项名称空间(3.2.11.1)
session	cURL 会话名称空间(3.2.11.2)
[session] make_session_is([istream])	创建 cURL 会话并绑定输入流
[session] make_session_os([ostream])	创建 cURL 会话并绑定输出流
[session] make_session_ios ([istream], [ostream])	创建 cURL 会话并绑定输入和输出流

3.2.11.1. SSL 选项名称空间

代码	功能
none	不使用 SSL
try_use	尝试使用 SSL
control	在控制连接中使用 SSL
all	在控制和数据连接中使用 SSL

3.2.11.2. cURL 会话名称空间

代码	功能
`void set_url([session], string)`	设置会话 URL
`void set_tcp_keep_alive ([session], number probe)`	设置 TCP Keep Alive 心跳包时间
`void set_http_post([session], boolean use_post)`	设置是否使用 HTTP POST 请求
`void use_ssl([session], [ssl_level])`	设置 SSL 选项
`void set_ssl_verify_host ([session], bool verify)`	设置是否验证服务器证书
`void set_ssl_verify_peer ([session], bool verify)`	设置是否验证对等证书
`void set_ssl_cert([session], string path)`	设置 SSL 证书
`void set_ssl_key([session], string path)`	设置 SSL 密钥
`void set_ssl_passwd([session], string passwd)`	设置 SSL 证书密码
`void set_connect_timeout ([session], number time)`	设置连接超时时间（秒）
`void set_connect_timeout_ms ([session], number time)`	设置连接超时时间（毫秒）
`void set_accept_timeout_ms ([session], number time)`	设置接受超时时间（毫秒）
`void set_transmit_timeout ([session], number time)`	设置传输超时时间（秒）
`void set_transmit_timeout_ms ([session], number time)`	设置传输超时时间（毫秒）
`boolean perform([session])`	执行请求，成功时返回真

3.2.12. wiringPi 扩展(wiringPi)

注意：该扩展仅适用于 Raspberry Pi 以及其他兼容 wiringPi GPIO 库的设备

代码	功能
`input`	GPIO 输入模式
`output`	GPIO 输出模式
`pwm_output`	GPIO PWM 输出模式
`pud_off`	无上拉或下拉电阻
`pud_down`	下拉电阻
`pud_up`	上拉电阻
`high`	高电平
`low`	低电平

代码	功能
`void setup()`	加载 wiringPi 功能
`void pin_mode(number pin, [mode])`	设置 GPIO 针脚模式
`void pud_cntl(number pin, [mode])`	设置 GPIO PUD 模式
`void digital_write(number pin, [value])`	设置针脚为高电平或低电平
`void pwm_write(number pin, number value)`	写入 PWM 方波，频率范围取决于设备
`[value] digital_read(number pin)`	通过针脚读取，返回高电平或低电平
`number analog_read(number pin)`	读取模拟量（需要附加设备）
`void analog_write(number pin, number value)`	写入模拟量（需要附加设备）
`number time()`	返回微秒计时结果
`void delay(number)`	暂停一段时间，单位微秒
`[serial] serial_open(string device, number baud)`	打开一个串口设备
`void serial_close([serial])`	关闭一个串口设备
`void serial_putchar([serial], char)`	在串口中写入字符
`void serial_print([serial], string)`	在串口中写入字符串
`number serial_data_avail([serial])`	获取串口中可用的字符数量
`number serial_getchar([serial])`	从串口中获取一个字符的 ASCII 码
`void serial_flush([serial])`	刷新串口缓冲区

***注意，所有解释均以 wiringPi 官方为准**

wiringPi Reference: http://wiringpi.com/reference/

GPIO 针脚编号：https://projects.drogon.net/raspberry-pi/wiringpi/pins/

4. 解释器

4.1. 命令行参数

4.1.1. 解释器

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	导出抽象语法树
--dependency	-r	导出模块依赖
--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，若不设置日志输出路径，将直接输出至标准输出流

4.1.2. 交互式解释器

注意，从 3.3 版本开始，交互式解释器与解释器已经合二为一

cs [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
--no-optimize	-o	禁用优化器
--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

4.1.3. 调试器

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

4.1.3.1. 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

4.2. 解释器扩展

4.2.1. 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

4.2.1.1. 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

4.2.1.2. 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

4.2.1.3. 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

4.2.1.3.1. 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

4.2.1.3.2. 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

`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

`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

4.2.1.3.3. 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**

CNI_VISITOR(*函数名*)

向当前名称空间添加一个 CNI 成员访问函数

CNI_VISITOR_V(*函数名*，*函数*)

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

CNI 成员访问函数是一种特殊的函数，要求仅接受一个参数。设该参数的完全退化类型为 T，当该函数被添加到 T 类型的 CNI 类型扩展名称空间中时，能够被自动调用

一般要求函数必须是 C++类型成员变量的访问器，且函数不可抛出异常、不可阻塞进程、不可执行过长时间、不能有调用副作用

CNI_CLASS_MEMBER(*类型*，*成员*)

向当前名称空间添加一个 CNI 类型成员

CNI_CLASS_MEMBER_CONST(*类型*，*成员*)

向当前名称空间添加一个 CNI 常量类型成员

CNI 类型成员将提供两个方法(get 和 set)分别用于访问和设置类型成员的值，但 CNI 常量类型成员将不可使用 set 方法

4.2.1.3.4. 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_NAMESPACE_ALIAS(名称空间名, 名称空间别名)
```

声明一个名称空间别名

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**

4.2.1.3.5. 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

版本信息

Covariant Script 标准	201201
文档修订版本	210202