在一些传统编程语言中，一个表达式由一个或多个操作数（operand）通过零个或多个操作符（operator）组合而成，表达式总是隐含着一个计算过程，因此每个表达式都会有一个计算结果，对于只有操作数而没有操作符的表达式，其计算结果就是操作数自身，对于包含操作符的表达式，计算结果是对操作数执行操作符定义的计算而得到的值。在这种定义下的表达式也被称为算术运算表达式。操作符优先级请参见[操作符](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/Appendix/operator.html)章节。

在仓颉编程语言中，简化并延伸了表达式的传统定义——凡是可求值的语言元素都是表达式。因此，仓颉不仅有传统的算术运算表达式，还有条件表达式、循环表达式和 try 表达式等，它们都可以被求值，并作为值去使用，如作为变量定义的初值和函数实参等。此外，因为仓颉是强类型的编程语言，所以仓颉表达式不仅可求值，还有确定的类型。

仓颉编程语言的各种表达式将在后续章节中逐一介绍，本节介绍最常用的条件表达式、循环表达式以及部分控制转移表达式（break、continue）。

任何一段程序的执行流程，只会涉及三种基本结构——顺序结构、分支结构和循环结构。实际上，分支结构和循环结构，是由某些指令控制当前顺序执行流产生跳转而得到的，它们让程序能够表达更复杂的逻辑，在仓颉中，这种用来控制执行流的语言元素就是条件表达式和循环表达式。

在仓颉编程语言中，条件表达式分为 if 表达式和 if-let 表达式两种，它们的值与类型需要根据使用场景来确定。循环表达式有四种：for-in 表达式、while 表达式、do-while 表达式和 while-let 表达式，它们的类型都是 Unit、值为 ()。其中 if-let 表达式和 while-let 表达式都与模式匹配相关，请参见[if-let 表达式](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/enum_and_pattern_match/if_let.html)和[while-let 表达式](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/enum_and_pattern_match/while_let.html)章节，本节只介绍以上提及的其他几种表达式。

在仓颉程序中，由一对大括号“{}”包围起来的一组表达式，被称为“代码块”，它将作为程序的一个顺序执行流，其中的表达式将按编码顺序依次执行。如果代码块中有至少一个表达式，规定此代码块的值与类型等于其中最后一个表达式的值与类型，如果代码块中没有表达式，规定这种空代码块的类型为 Unit、值为 ()。

**注意：**

代码块本身不是一个表达式，不能被单独使用，它将依附于函数、条件表达式和循环表达式等执行和求值。

## [if 表达式](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_programming_concepts/expression.html" \l "if-%E8%A1%A8%E8%BE%BE%E5%BC%8F)

if 表达式的基本形式为：

if (条件) {

分支 1

} else {

分支 2

}

其中“条件”是布尔类型表达式，“分支 1”和“分支 2”是两个代码块。if 表达式将按如下规则执行：

1. 计算“条件”表达式，如果值为 true 则转到第 2 步，值为 false 则转到第 3 步。
2. 执行“分支 1”，转到第 4 步。
3. 执行“分支 2”，转到第 4 步。
4. 继续执行 if 表达式后面的代码。

在一些场景中，可能只关注条件成立时该做些什么，所以 else 和对应的代码块是允许省略的。

如下程序演示了 if 表达式的基本用法：

import std.random.\*

main() {

let number: Int8 = Random().nextInt8()

println(number)

if (number % 2 == 0) {

println("偶数")

} else {

println("奇数")

}

}

在这段程序中，使用仓颉标准库的 random 包生成了一个随机整数，然后使用 if 表达式判断这个整数是否能被 2 整除，并在不同的条件分支中打印“偶数”或“奇数”。

仓颉编程语言是强类型的，if 表达式的条件只能是布尔类型，不能使用整数或浮点数等类型，和 C 语言等不同，仓颉不以条件取值是否为 0 作为分支选择依据，例如以下程序将编译报错：

main() {

let number = 1

if (number) { // Error, mismatched types

println("非零数")

}

}

在许多场景中，当一个条件不成立时，可能还要判断另一个或多个条件、再执行对应的动作，仓颉允许在 else 之后跟随新的 if 表达式，由此支持多级条件判断和分支执行，例如：

import std.random.\*

main() {

let speed = Random().nextFloat64() \* 20.0

println("${speed} km/s")

if (speed > 16.7) {

println("第三宇宙速度，鹊桥相会")

} else if (speed > 11.2) {

println("第二宇宙速度，嫦娥奔月")

} else if (speed > 7.9) {

println("第一宇宙速度，腾云驾雾")

} else {

println("脚踏实地，仰望星空")

}

}

if 表达式的值与类型，需要根据使用形式与场景来确定：

当含 else 分支的 if 表达式被求值时，需要根据求值上下文确定 if 表达式的类型：

* + 如果上下文明确要求值类型为 T，则 if 表达式各分支代码块的类型必须是 T 的子类型，这时 if 表达式的类型被确定为 T，如果不满足子类型约束，编译会报错。
  + 如果上下文没有明确的类型要求，则 if 表达式的类型是其各分支代码块类型的最小公共父类型，如果最小公共父类型不存在，编译会报错。

如果编译通过，则 if 表达式的值就是所执行分支代码块的值。

如果含 else 分支的 if 表达式没有被求值，在这种场景里，开发者一般只想在不同分支里做不同操作，不会关注各分支最后一个表达式的值与类型，为了不让上述类型检查规则影响这一思维习惯，仓颉规定这种场景下的 if 表达式类型为 Unit、值为 ()，且各分支不参与上述类型检查。

对于不含 else 分支的 if 表达式，由于 if 分支也可能不被执行，所以规定这类 if 表达式的类型为 Unit、值为 ()。

例如，以下程序基于 if 表达式求值，模拟一次简单的模数转换过程：

main() {

let zero: Int8 = 0

let one: Int8 = 1

let voltage = 5.0

let bit = if (voltage < 2.5) {

zero

} else {

one

}

}

在以上程序中，if 表达式作为变量定义的初值使用，由于变量 bit 没有被标注类型、需要从初值中推导，所以 if 表达式的类型取为两个分支代码块类型的最小公共父类型，根据前文对“代码块”的介绍，可知两个分支代码块类型都是 Int8，所以 if 表达式的类型被确定为 Int8，其值为所执行分支即 else 分支代码块的值，所以变量 bit 的类型为 Int8、值为 1。

## [while 表达式](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_programming_concepts/expression.html" \l "while-%E8%A1%A8%E8%BE%BE%E5%BC%8F)

while 表达式的基本形式为：

while (条件) {

循环体

}

其中“条件”是布尔类型表达式，“循环体”是一个代码块。while 表达式将按如下规则执行：

1. 计算“条件”表达式，如果值为 true 则转第 2 步，值为 false 转第 3 步。
2. 执行“循环体”，转第 1 步。
3. 结束循环，继续执行 while 表达式后面的代码。

例如，以下程序使用 while 表达式，基于二分法，近似计算数字 2 的平方根：

main() {

var root = 0.0

var min = 1.0

var max = 2.0

var error = 1.0

let tolerance = 0.1 \*\* 10

while (error \*\* 2 > tolerance) {

root = (min + max) / 2.0

error = root \*\* 2 - 2.0

if (error > 0.0) {

max = root

} else {

min = root

}

}

println("2 的平方根约等于：${root}")

}

运行以上程序，将输出：

2 的平方根约等于：1.414215

## [do-while 表达式](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_programming_concepts/expression.html" \l "do-while-%E8%A1%A8%E8%BE%BE%E5%BC%8F)

do-while 表达式的基本形式为：

do {

循环体

} while (条件)

其中“条件”是布尔类型表达式，“循环体”是一个代码块。do-while 表达式将按如下规则执行：

1. 执行“循环体”，转第 2 步。
2. 计算“条件”表达式，如果值为 true 则转第 1 步，值为 false 转第 3 步。
3. 结束循环，继续执行 do-while 表达式后面的代码。

例如，以下程序使用 do-while 表达式，基于蒙特卡洛算法，近似计算圆周率的值：

import std.random.\*

main() {

let random = Random()

var totalPoints = 0

var hitPoints = 0

do {

// 在 ((0, 0), (1, 1)) 这个正方形中随机取点

let x = random.nextFloat64()

let y = random.nextFloat64()

// 判断是否落在正方形内接圆里

if ((x - 0.5) \*\* 2 + (y - 0.5) \*\* 2 < 0.25) {

hitPoints++

}

totalPoints++

} while (totalPoints < 1000000)

let pi = 4.0 \* Float64(hitPoints) / Float64(totalPoints)

println("圆周率近似值为：${pi}")

}

运行以上程序，将输出：

圆周率近似值为：3.141872

**说明：**

由于算法涉及随机数，所以每次运行程序输出的数值可能都不同，但都会约等于 3.14。

## [for-in 表达式](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_programming_concepts/expression.html" \l "for-in-%E8%A1%A8%E8%BE%BE%E5%BC%8F)

for-in 表达式可以遍历那些扩展了迭代器接口 Iterable<T> 的类型实例。for-in 表达式的基本形式为：

for (迭代变量 in 序列) {

循环体

}

其中“循环体”是一个代码块。“迭代变量”是单个标识符或由多个标识符构成的元组，用于绑定每轮遍历中由迭代器指向的数据，可以作为“循环体”中的局部变量使用。“序列”是一个表达式，它只会被计算一次，遍历是针对此表达式的值进行的，其类型必须扩展了迭代器接口 Iterable<T>。for-in 表达式将按如下规则执行：

1. 计算“序列”表达式，将其值作为遍历对象，并初始化遍历对象的迭代器。
2. 更新迭代器，如果迭代器终止，转第 4 步，否则转第 3 步。
3. 将当前迭代器指向的数据与“迭代变量”绑定，并执行“循环体”，转第 2 步。
4. 结束循环，继续执行 for-in 表达式后面的代码。

**说明：**

仓颉内置的区间和数组等类型已经扩展了 Iterable<T> 接口。

例如，以下程序使用 for-in 表达式，遍历中国地支字符构成的数组 noumenonArray，输出农历 2024 年各月的干支纪法：

main() {

let metaArray = [r'甲', r'乙', r'丙', r'丁', r'戊',

r'己', r'庚', r'辛', r'壬', r'癸']

let noumenonArray = [r'寅', r'卯', r'辰', r'巳', r'午', r'未',

r'申', r'酉', r'戌', r'亥', r'子', r'丑']

let year = 2024

// 年份对应的天干索引

let metaOfYear = ((year % 10) + 10 - 4) % 10

// 此年首月对应的天干索引

var index = (2 \* metaOfYear + 3) % 10 - 1

println("农历 2024 年各月干支：")

for (noumenon in noumenonArray) {

print("${metaArray[index]}${noumenon} ")

index = (index + 1) % 10

}

}

其中 r 开头的字符表示[字符类型字面量](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_data_type/characters.html" \l "%E5%AD%97%E7%AC%A6%E7%B1%BB%E5%9E%8B%E5%AD%97%E9%9D%A2%E9%87%8F)。运行以上程序，将输出：

农历 2024 年各月干支：

丙寅 丁卯 戊辰 己巳 庚午 辛未 壬申 癸酉 甲戌 乙亥 丙子 丁丑

### [遍历区间](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_programming_concepts/expression.html" \l "%E9%81%8D%E5%8E%86%E5%8C%BA%E9%97%B4)

for-in 表达式可以遍历区间类型实例，例如：

main() {

var sum = 0

for (i in 1..=100) {

sum += i

}

println(sum)

}

运行以上程序，将输出：

5050

关于区间类型的详细内容，请参阅基本数据类型[区间类型](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_data_type/range.html)章节。

### [遍历元组构成的序列](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_programming_concepts/expression.html" \l "%E9%81%8D%E5%8E%86%E5%85%83%E7%BB%84%E6%9E%84%E6%88%90%E7%9A%84%E5%BA%8F%E5%88%97)

如果一个序列的元素是元组类型，则使用 for-in 表达式遍历时，“迭代变量”可以写成元组形式，以此实现对序列元素的解构，例如：

main() {

let array = [(1, 2), (3, 4), (5, 6)]

for ((x, y) in array) {

println("${x}, ${y}")

}

}

运行以上程序，将输出：

1, 2

3, 4

5, 6

### [迭代变量不可修改](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_programming_concepts/expression.html" \l "%E8%BF%AD%E4%BB%A3%E5%8F%98%E9%87%8F%E4%B8%8D%E5%8F%AF%E4%BF%AE%E6%94%B9)

在 for-in 表达式的循环体中，不能修改迭代变量，例如以下程序在编译时会报错：

main() {

for (i in 0..5) {

i = i \* 10 // Error, cannot assign to value which is an initialized 'let' constant

println(i)

}

}

### [使用通配符 \_ 代替迭代变量](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_programming_concepts/expression.html" \l "%E4%BD%BF%E7%94%A8%E9%80%9A%E9%85%8D%E7%AC%A6-_-%E4%BB%A3%E6%9B%BF%E8%BF%AD%E4%BB%A3%E5%8F%98%E9%87%8F)

在一些应用场景中，只需要循环执行某些操作，但并不使用迭代变量，这时可以使用通配符 \_ 代替迭代变量，例如：

main() {

var number = 2

for (\_ in 0..5) {

number \*= number

}

println(number)

}

运行以上程序，将输出：

4294967296

**注意：**

在这种场景下，如果使用普通的标识符定义迭代变量，编译会输出“unused variable”告警，使用通配符 \_ 则可以避免这一告警。

### [where 条件](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_programming_concepts/expression.html" \l "where-%E6%9D%A1%E4%BB%B6)

在部分循环遍历场景中，对于特定取值的迭代变量，可能需要直接跳过、进入下一轮循环，虽然可以使用 if 表达式和 continue 表达式在循环体中实现这一逻辑，但仓颉为此提供了更便捷的表达方式——可以在所遍历的“序列”之后用 where 关键字引导一个布尔表达式，这样在每次将进入循环体执行前，会先计算此表达式，如果值为 true 则执行循环体，反之直接进入下一轮循环。例如：

main() {

for (i in 0..8 where i % 2 == 1) { // i 为奇数才会执行循环体

println(i)

}

}

运行以上程序，将输出：

1

3

5

7

## [break 与 continue 表达式](https://docs.cangjie-lang.cn/docs/0.53.18/user_manual/source_zh_cn/basic_programming_concepts/expression.html" \l "break-%E4%B8%8E-continue-%E8%A1%A8%E8%BE%BE%E5%BC%8F)

在循环结构的程序中，有时需要根据特定条件提前结束循环或跳过本轮循环，为此仓颉引入了 break 与 continue 表达式，它们可以出现在循环表达式的循环体中，break 用于终止当前循环表达式的执行、转去执行循环表达式之后的代码，continue 用于提前结束本轮循环、进入下一轮循环。break 与 continue 表达式的类型都是 Nothing。

例如，以下程序使用 for-in 表达式和 break 表达式，在给定的整数数组中，找到第一个能被 5 整除的数字：

main() {

let numbers = [12, 18, 25, 36, 49, 55]

for (number in numbers) {

if (number % 5 == 0) {

println(number)

break

}

}

}

当 for-in 迭代至 numbers 数组的第三个数 25 时，由于 25 可以被 5 整除，所以将执行 if 分支中的 println 和 break，break 将终止 for-in 循环，numbers中的后续数字不会被遍历到，因此运行以上程序，将输出：

25

以下程序使用 for-in 表达式和 continue 表达式，将给定整数数组中的奇数打印出来：

main() {

let numbers = [12, 18, 25, 36, 49, 55]

for (number in numbers) {

if (number % 2 == 0) {

continue

}

println(number)

}

}

在循环迭代中，当 number 是偶数时，continue 将被执行，这会提前结束本轮循环、进入下一轮循环，println 不会被执行，因此运行以上程序，将输出：

25

49

55