第 3 章

编 程 基 础

第 2 章成功搭建好了仓颉语言的开发环境。本章介绍在仓颉语言中的基础程序结构、仓颉语言的内置保留关键字、标识符、变量和常量的定义和使用，以及仓颉语言代码编写规范。

3.1 程序结构

仓颉语言代码保存在一个以.cj 为后缀的源程序文件中。

程序的基础结构如下：

```
//全局变量
let x: Int64 = 1
let y: Int64 = 2

//全局函数
func f1() {}
func f2() {}

//自定义类型,class 类型 A
class A {}

//自定义类型,结构体类型 B
structB {}

//程序执行入口
main() {
   ...
   //code blocks
   ...
}
```

在使用仓颉语言编写的代码中，必须包含一个入口程序，即 main( )函数，main 函数前面并没有使用 func 函数定义关键字，所以该函数的作用只是作为程序的主入口。

一个标准的仓颉语言项目的目录结构如下：

```
├── src                    \
│   ├── aaaa               |
│   │   └── xxxxxxx.cj     |
│   ├── bbbb               |
│   │   └── xxxxxxx.cj      >源码文件: *.cj
│   ├── cccc               |
│   │   └── xxxxxxx.cj     |
│   ├── xxxx.cj            /
├── test
│    ├── HLT
│    │   └── testcase0001.cj \
│    ├── LLT                 |
│    │   └── testcase0001.cj  >LLT：自测用例；UT：单元测试用例；HLT：测试级别用例
│    └── UT                  |
│        └── testcase0001.cj /
├── module.json     ----------->用于 CPM 构建
├── doc
└── README.md
```

在仓颉语言中，代码是以包为单位进行组织的，一个目录即是一个包，目录可以相互嵌套，多个包组成一个模块，一个仓颉项目可以包含多个模块，但是必须有一个主模块。

3.2　关键字

首先来了解一下在仓颉语言中的一些被语言内部使用的关键字。需要注意的是，关键字是不能作为标识符使用的特殊字符串，仓颉语言的关键字见表 3-1。

表 3-1　在仓颉语言中有 71 个系统保留字

abstract	as	break	match	mut	override
Bool	case	continue	open	operator	prop
catch	class	else	package	private	quote
Char	do	false	protected	public	return
enum	extend	foreign	struct	redef	super
finally	for	Float16	spawn	static	throw
from	func	if	synchronized	this	type
Float32	Float64	init	true	try	UInt8
import	in	Int8	This	unsafe	UInt64
interface	is	Int64	UInt16	UInt32	var
Int16	Int32	macro	UIntNative	Unit	main
IntNative	let	Nothing	where	while	

3.3 标识符

在介绍如何定义和使用变量之前，有必要先介绍仓颉语言中的标识符，因为变量名必须是一个合法的标识符。

标识符分为普通标识符和原始标识符（Raw Identifier）。

3.3.1 普通标识符

普通标识符是除了关键字（参见表 3-1 关键字）以外，由字母开头，后接任意长度的字母、数字或下画线组合而成的连续字符。

合法的普通标识符如下：

```
xyz
x1y2z3
x_y_z
x1_y2_z3
```

非法的普通标识符如下：

```
_xyz                //不能以'_'开头
1xyz                //不能以数字开头
if                  //错误：if 是保留关键字
while               //错误：while 是保留关键字
```

3.3.2 原始标识符

原始标识符是在普通标识符的外面加上一对反引号（`），或者反引号内使用关键字。原始标识符主要用在需要将仓颉关键字作为标识符使用的场景。

合法的原始标识符如下：

```
`xyz`
`x1y2z3`
`x_y_z`
`x1_y2_z3`
`if`
`while`
```

非法的原始标识符如下：

```
`_xyz`              //不能以 '_'开头
`1xyz`              //不能以数字开头
```

3.4　注释

程序的注释是解释性语句，可以在仓颉语言代码中添加注释，这将提高源代码的可读性。所有的编程语言都允许某种形式的注释。

仓颉语言支持单行注释和多行注释。注释中的所有字符会被仓颉语言编译器忽略。

3.4.1　单行注释

单行注释以“//”开始，直到行末为止，示例代码如下：

```
var foo = 100                //右边的单行注释
var bar = 200

main() {
    //上方的单行注释
    println("hello cangjie")
    var num : Int64 = 10    //右边的单行注释
    //上方的单行注释
    return 0
}
```

对于单行注释的要求如下：

（1）代码上方的注释与被注释的代码行间无空行，保持与代码一样的缩进。

（2）代码右边的注释与代码之间至少留有 1 个空格。代码右边的注释无须插入空格使其强行对齐。

3.4.2　多行注释

多行注释以“/*”开始，以“*/”终止。例如，给一个仓颉代码文件添加注释，就可以使用多行注释：

```
/*
 * Copyright (c) Cangjie Library Team 2022-2022. All rights reserved.
 * Description:
 */
```

给一个类添加注释，一般也使用多行注释：

```
**
 * The class is ByteBuffer inherited from Collection<UInt8>
 * @author Leo
 * @since 0.29.3
 */
public class ByteBuffer <: Collection<UInt8>{
```

```
    ...
    //code blocks
    ...
}
```

在“ /*”和“ */”注释内部，“//”字符没有特殊的含义。在“//”注释内，“/*”和“ */”字符也没有特殊的含义，因此，可以在一种注释内嵌套另一种注释，示例如下：

```
/* 用于输出 Hello World 的注释
 println("Hello World")         //输出 Hello World
*/
```

3.5 变量和常量

仓颉语言是强类型语言，变量必须有类型，同时变量仅可以存储特定类型的数据。每个变量都代表一块内存，变量名是给某块内存起的一个别名，内存中存的值就是给变量赋的值。变量名在程序编译期间会直接转换为内存地址。

在仓颉语言中的变量被分为两类：不可变变量和可变变量，其中不可变变量是指初始化后值不可变的变量，可变变量是指可以重复赋值的变量；不可变变量和可变变量分别使用 let 和 var 定义。

3.5.1 定义变量

仓颉语言虽然是静态语言，但是在代码语法风格上并不强制要求在定义一个变量时指定变量的数据类型，编译器可以根据变量的值来推断它的数据类型。变量声明有显式声明、隐式声明。

显式定义变量的格式如图 3-1 所示。

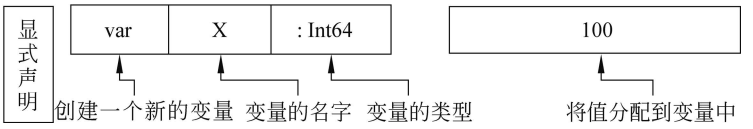

图 3-1 显式定义一个变量

隐式声明在定义变量时如果指定了初始值，则可以省略类型的定义，仓颉语言可以自己由数据推导出类型，如图 3-2 所示。

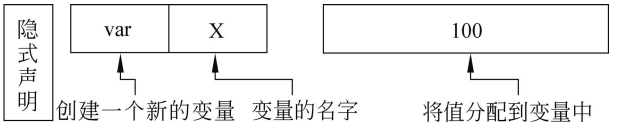

图 3-2 隐式定义一个变量

3.5.2　定义常量

常量是固定值，在程序执行期间不会改变，常量可以是任何的基本数据类型，例如整数常量、浮点常量、字符常量，或字符串字面值，也有枚举常量。

在仓颉语言中，定义常量的格式和定义变量的格式一样，只不过声明常量时使用 let 关键字，如图 3-3 所示。

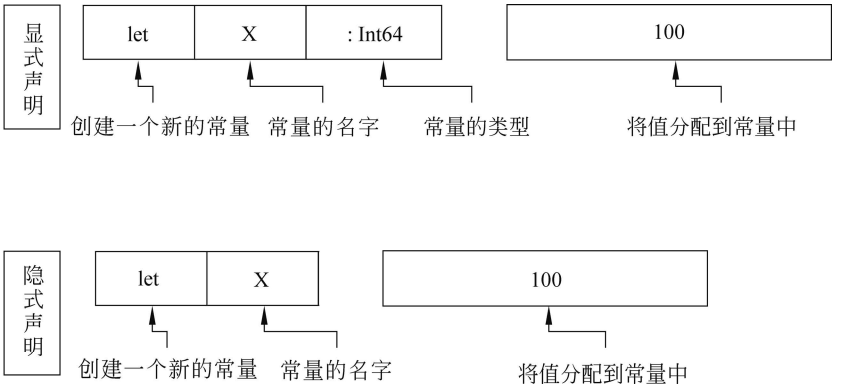

图 3-3　定义一个变量

说明：对于 let 的全局变量和 static let 变量的名称，建议采用全大写。用于不可变场景的 let 变量，其命名可以用下画线分隔的全大写单词来命名以强调是常量，如 let MAXUSERNUM = 200。

3.5.3　作用域

在下面的例子中，定义了两种不同作用域的变量和常量，g1 和 g2 是全局变量，PI 为全局常量，*x* 和 *y* 定义在 main 函数内部，是局部变量，代码如下：

```
//全局变量
var g1:Int64 = 2;
var g2:Int64 = 3;

//全局常量
let PI:Float64 = 3.14

func main() {
        //定义两个局部变量 x 和 y
    var x : Int64 = 100;
    var y : Int64 = 200;
    println(g2)             //2
    println(x)              //100
    println(PI)
```

```
}
```

如果尝试修改不可变变量 *x* 的值，将编译报错，如图 3-4 所示，代码如下：

```
func main() {
    let PI:Float64 = 3.14;
    PI = 3.145; //不可以重新向一个常量赋值
    println(PI)
}
```

```
02.cj:7:5: error: cannot assign to value
 which is an initialized 'let' constant
    7 |     PI = 3.145;
      |     ^
1 error generated, 1 error printed.
```

图 3-4　重新向一个常量赋值，编译期间报错

当初始值的类型明确时，编译器可以根据初始值推断变量的类型，这时可以省略变量的类型标注。

对上面的例子进行修改，使其可以正常编译和运行，输出结果相同，代码如下：

```
//全局变量
var g1 = 2;
var g2 = 3;
//全局常量
let PI = 3.14
func main() {
    //定义两个局部变量 x 和 y
    var x = 100;
    var y = 200;
    println(x)          //输出 3.14
}
```

3.5.4　初始化

定义全局变量和静态成员变量（定义在类和结构体中）时必须初始化，例如，在下面的例子中，定义全局变量 *b* 和静态变量 *d* 时会因未初始化而报错，代码如下：

```
let a: Int64 = 10
let b: Int64
//Error: 全局常量 b 必须初始化
record R {
static let c: Int64 = 100
static let d: Int64
//Error: 静态变量 d 必须初始化
```

```
}
```

对于局部变量，允许定义时不初始化（此时必须标注类型），但必须确保初始化一定要在该变量第 1 次被读取之前完成，代码如下：

```
func main() {
  let x: Int64
  x = 1
  print("x = ${x}\n")
  var y: Float64
  y = 1.1
  print("y = ${y}\n")
  y = 2.2
  print("y = ${y}")
}
```

在仓颉语言中要求每个变量在使用前必须进行初始化，否则编译时会报错。例如，下例中在 *y* 的初始化表达式中访问 *x* 时，*x* 并未初始化，代码如下：

```
func main() {
  let x: Int64
  var y = x + 1          //Error: 变量 x 尚未初始化
  print("y = ${y}")
}
```

3.6　代码编写规范

代码编写规范一般包含标识符的命名、注释及排版。一致的编码习惯与风格会使代码更容易阅读、理解，更容易维护。在仓颉语言中的各种类型的名字均使用统一的命名风格，具体见表 3-2。

表 3-2　在仓颉语言中的各种类型的名字均使用统一的命名风格

类　别	命 名 风 格	形　式
包名和文件名	unix_like：单词全小写，用下画线分隔	如 aaa_bbb

接口，类，结构体，枚举和类型别名	大驼峰：首字母大写，单词连在一起，不同单词间通过单词首字母大写分开，可包含数字	如 AaaBbb
		//my_class.cj
		public class MyClass {
		//CODE
		}

变量，函数，函数参数	小驼峰：首字母小写，单词连在一起，不同单词间通过单词首字母大写分开。例外：测试函数可有下画线，循环变量 try-catch 中的异常变量允许单个小写字母	如 aaaBbb

续表

类　别	命 名 风 格	形　式
全局变量, static 成员变量	建议全大写，用下画线分隔	如 let MAX_USER_NUM = 200
泛型类型变量	单个大写字母，或单个大写字母加数字，或单个大写字母接下画线、大写字母和数字的组合，例如 E, T, T2, E_IN, E_OUT,T_CONS	如 A

3.7 本章小结

本章详细介绍了仓颉语言的基础程序结构、内置保留关键字、标识符、变量和常量的定义和使用，以及仓颉语言代码编写规范。