第5章 类和对象

本章导读

在仓颉编程语言中，类是面向对象程序设计的核心构造，它将数据（属性）与行为（方法）封装成一个完整的逻辑单元。通过类的抽象能力，开发者可以准确模拟现实世界的实体概念，有效提升代码的模块化程度和重用性。仓颉语言为类提供了丰富的特性支持：多种类型的构造函数满足对象初始化需求，属性访问机制在保障数据封装安全性的同时提供了便捷的成员访问方式，灵活的访问控制修饰符精确管理类成员的可见范围。作为引用类型，类在赋值和参数传递时操作的是对象引用而非副本，这一特性使其特别适合构建具有共享状态和复杂生命周期管理的应用程序。类的引用语义为开发大型软件系统提供了坚实的基础架构支持，是现代程序设计不可或缺的重要组成部分。

本章要点

- 类与结构体的区别：理解类作为引用类型与结构体值类型的本质差异。
- 类定义语法：掌握类的基本定义方法和组成部分。
- 数据成员：成员变量与属性的定义与使用。
- 构造函数：普通构造函数和主构造函数的定义与调用。
- 成员函数：示例函数、静态函数和抽象函数的不同特性。
- 终结器：了解对象销毁时的清理机制。
- 访问运算符：成员访问操作符的使用。
- 访问修饰符：public、private、protected 等访问权限控制。
- 访问属性：getter 和 setter 方法的定义与使用。
- 对象创建：示例化对象的各种方式。
- 对象数组：创建和管理对象数组。
- 对象赋值：理解对象赋值的引用语义。
- 类作用域：类内部的作用域规则。

- 自引用指针：this 关键字的使用场景和方法。

5.1 类

在第 1 章的学习中，我们已经学习了面向对象的基础知识，了解了面向对象的基本概念和特征——封装、继承和多态，类是实现这些特征的基础，是对现实世界中客观事物的抽象，是从属性和操作两方面对数据的描述，是将不同类型的数据和与这些数据相关的操作封装在一起的集合体。数据可以描述类的属性，用数据成员表示，操作可以描述类的服务，用成员函数表示。从数据处理的角度来说，和结构体一样，仓颉语言中的类也是一种用户定义的数据类型，都是将数据的静态属性和操作的行为属性进行了封装，不同的是，结构体是表示值类型的数据类型，而类则是引用类型的数据类型。

5.1.1 从结构体到类

类是面向对象编程中的经典概念，仓颉支持使用类来实现面向对象编程。在第 4 章的结构体部分，已经了解了结构体的基本结构如下：

```
struct  结构体名称{
    let 变量名 1:变量类型 1
    let 变量名 2:变量类型 2
    ……
    构造函数
    成员函数
    }
```

例如，一个点的结构体及结构体类型变量定义可以用以下程序实现。

【例 5-1】 点的结构体及结构体类型变量定义。

```
package demo
struct Point {
      let x: Int64                        // 数据成员 x
      let y: Int64                        // 数据成员 y
    public init() {                       // 不带参数的构造函数
        x=50
        y=100  }
    public init(a:Int64,b:Int64) {    //带参数的构造函数
    x=a
    y=b
    }
}
main(): Int64 {
      var a=Point()                       //结构体变量
      var b=Point(5,10)                   //构造函数对结构体变量进行初始化
      println(" 点 a 的坐标为:(${a.x},${a.y})")
      println(" 点 b 的坐标为:(${b.x},${b.y})")
    return 0
}
```

编译执行上述代码，输出结果为：

```
点 a 的坐标为:(50,100)
点 b 的坐标为:(5,10)
```

仓颉的结构体包括了数据成员以及为这些成员进行初始化的构造函数，在使用时需要创建结构体变量，并利用构造函数对结构体变量进行初始化。仓颉的结构体可以认为是数据和行为的一种简单封装，在第 1 章提到面向对象的特征时，除了封装之外，还有继承和多态，这两项特性用结构体是没法实现的，必须使用面向对象的类机制来建立类的内部数据的管理和控制方式，才能实现不同类的数据与数据的行为之间的继承和多态关系。

5.1.2 类的定义

仓颉的类与其他的面向对象的编程语言一样，其构成一般分为类修饰符、class 关键字、类名和类体等四部分。其中，类修饰符说明这个类的可访问或可继承属性，如 public 修饰符说明该类是包外可见，sealed 和 abstract 表示是抽象类，open 表示该类是可被续承的，类名是用于标识和区分不同类的标识符，是类的唯一名称。类体内包括数据成员（成员变量）和成员函数。其组成结构为：

```
类修饰符 class  类名称{
    访问修饰符 let 变量名 1:变量类型 1
    访问修饰符 let 变量名 2:变量类型 2
    ……
    访问修饰符 构造函数
    访问修饰符 成员函数
    ……

}
```

简单地说，仓颉的 class 类的定义是以关键字 class 开头，后跟类的名字，接着是定义在一对花括号中的类体。类体中可以定义一系列的成员变量、成员属性、静态初始化器、构造函数、成员函数和操作符函数等。要系统了解仓颉语言中类的组成结构，可参考图 5-1 所示的仓颉编程语言类组成结构示意图。

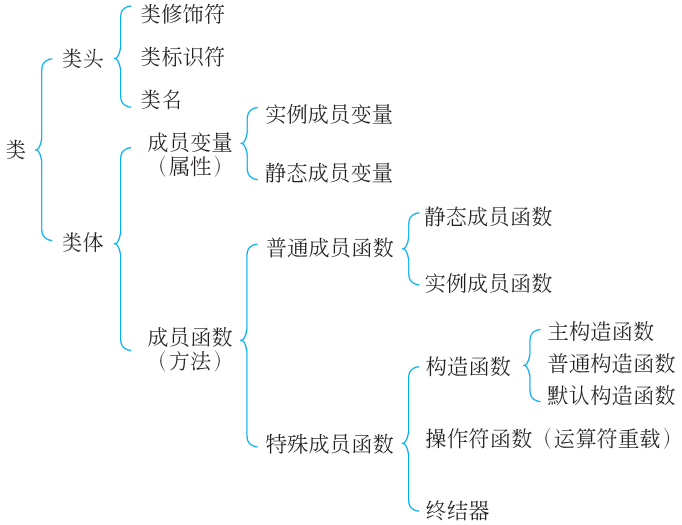

图 5-1 仓颉编程语言类组成结构示意图

仓颉语言支持中文变量命名，当然也支持中文类名。例如，在解决两个汉字部件组成的汉字结构分析问题时，如果用传统的英文类名和变量名的方式写出来是以下形式：

```
class Hanzi {
    let radical: String                 // 部件 1
    let component: String               // 部件 2
    public init(radical: String, component: String) {
        this.radical =radical
        this.component =component
    }
    public func compose(): String {
        return radical +component
    }
}
main() {
    let h =Hanzi("讠", "青")            // 比如“清”字的组合
    let fullChar =h.compose()
    println("组合后的汉字是:${fullChar}")       //
}
```

在进行汉字结构分析中，很多汉字是由两个或两个以上的独体字组合而成的合体字，古人把左右结构的合体字的左方称为“偏”，右方称为“旁”，部首是表义的偏旁，现在的仓颉编程语言可以用“汉字”定义一个汉字类，使得在分析汉字的组成结构时一目了然。例 5-2 是上述程序的“汉字”编程方式。

【例 5-2】 利用偏旁和部首组字的类。

```
package demo
class 汉字 {
    let 偏旁: String                    //汉字偏旁数据成员
    let 部首: String                    //汉字部首数据成员
    public init(radical: String, component: String) {     //构造函数
        this.偏旁 =radical
        this.部首 =component
    }
    public func compose(): String {    // 成员函数
        return 偏旁 +部首
    }
}
main() {
    let 清 =汉字("讠", "青")           // 用汉字类示例化一个”清”字对象
    let fullChar =清.compose()          //”清”字的组成
    println("\"清\"由\"${清.偏旁}\"与\"${清.部首}\"两部分组成:")
    println( "${清.偏旁}+${清.部首}=${fullChar}=清")
}
```

编译执行上述代码，输出结果为：

```
"清"由"讠"与"青"两部分组成:
讠+青=讠青=清
```

例 5-2 中定义了名为汉字(Hanzi)的 class 类型，它有两个 String 类型的成员变量。偏旁(radical)和部首(component)，一个包含有两个 String 类型参数的构造函数 init()，实现对成员变量偏旁和部首的初始化，以及一个成员函数 compose()，实现返回偏旁和部首的组合。

注意：

(1) class 只能定义在源文件最上层，类的成员变量和类的数据成员是类的重要数据成员。

(2) 类修饰符是说明该类的使用和访问属性的，通常有访问修饰符和非访问修饰符，访问修饰符主要用来进行包和模块管理，非访问修饰符所包含的功能较多，将在相关章节中做

详细的介绍。

（3）非访问修饰符如表 5-1 所示。

表 5-1　非访问修饰符的名称及功能

编　号	修　饰　符	功 能 描 述
1	open	表示该示例成员可被子类覆盖，或者该类能被子类继承
2	sealed	表示该 class 或 interface 只能在当前包内被继承或实现
3	override	表示覆盖父类的成员
4	redef	表示重新定义父类的静态成员
5	static	表示该成员是静态成员，静态成员不能通过示例对象访问
6	abstract	表示该 class 是抽象类
7	foreign	表示该成员是外部成员
8	unsafe	表示与 C 语言进行互操作的上下文
9	sealed	表示该 class 或 interface 只能在当前包内被继承或实现
10	mut	表示该成员是具有可变语义的

注：表中这些修饰符的具体使用将在相关章节进行介绍。

5.2　类的数据成员

类的数据成员包括成员变量和成员函数，成员变量分为实例成员变量和静态成员变量，成员函数可分为构造函数、成员函数和终结器函数。

5.2.1　成员变量

1. 实例成员变量

实例成员变量是在类中定义的特殊变量，和普通定义的变量相比，实例成员变量定义时可以设置初值，也可以不设置初值，但必须标注类型，且实例成员变量只能通过对象来访问。例如 Hanzi 类中表示汉字两个部件的 radical 和 component 就是两个示例成员变量：

```
let radical: String                    // 部件 1
let component: String                  // 部件 2
```

radical 和 component 这两个变量都没有初始化。

如果需要初始化，Hanzi 类可以写成如下形式：

```
class Hanzi {
    let radical: String ="讠"          // 部件 1
    let component: String ="王"        // 部件 2
}
main() {
    let h =Hanzi()
    println("两个部件输出是:${h.radical}和${h.component}")
}
```

上述程序中分别用"讠"和"王"对两个部件进行了初始化。两个部件输出时只能通过对

象 h 的访问(即 h.radical 和 h.component)来实现。

2. 静态成员变量

静态成员变量使用 static 修饰符修饰,必须有初值,与实例成员不同的是,静态成员变量不能使用对象访问,只能通过类名访问。例如,可以在上述偏旁和部首组字的类中定义一个可以表示组成汉字数量的静态成员变量 num。

【例 5-3】 设计偏旁和部首组字的类,用静态成员变量表示组成汉字的数量。

```
class Hanzi {
    let radical: String
    let component: String
    static var num=0
        public Hanzi(radical: String, component: String) {
        this.radical =radical
        this.component =component
        num++
    }
    public func compose(): String {
        return radical +component
    }
}
main() {
    let h1 =Hanzi("讠", "青")
    let fullChar1 =h1.compose()
    println("第${Hanzi.num}个汉字的组合是:${fullChar1}")
    let h2 =Hanzi("讠", "王")
    let fullChar2 =h2.compose()
    println("第${Hanzi.num}个汉字的组合是:${fullChar2}")
}
```

编译执行上述代码,输出结果为:

```
第 1 个汉字的组合是:讠青
第 2 个汉字的组合是:讠王
```

与实例成员变量相比,静态成员变量声明时必须有初始值,如上述示例中的 num,声明时用 static var num=0 来表示,访问时使用类名访问 Hanzi.num,不能用对象名访问。

3. 类的静态初始化器

仓颉的类支持定义静态初始化器,并在静态初始化器中通过赋值表达式来对静态成员变量进行初始化。静态初始化器以关键字组合 static init 开头,后跟无参参数列表和函数体,且不能被访问修饰符修饰。函数体中必须完成对所有未初始化的静态成员变量的初始化,否则编译报错。例如例 5-2,也可用以下程序表示。

【例 5-4】 类的静态初始化器的使用。

```
class Hanzi {
    let radical: String
    let component: String
    static var num:Int64
    static init(){
        num=0
    }
      public init(radical: String, component: String) {
        this.radical =radical
```

```
        this.component =component
        num++
    }
    public func compose(): String {
        return radical +component
    }
}
其中:static init(){
        num=0
    }
```

就是一个静态初始化器,它的功能是实现对静态成员 num 的初始化。

5.2.2 成员函数

1. 成员函数的基本概念

成员函数(也称为成员方法)是定义在类中的函数,用于实现类的行为操作。与成员变量(数据成员)不同,成员函数定义了对象可以执行的操作,通过函数封装了类的行为逻辑,实现了面向对象程序设计中的行为抽象和封装。

仓颉编程语言中的成员函数可以定义在结构体、枚举、类以及接口中,在类中定义的函数称为类的成员函数。成员函数可以访问类中的所有成员(包括私有成员),并通过函数参数和返回值与外部进行交互。

普通成员函数和特殊成员函数,其中普通成员函数包括静态成员函数和实例成员函数,特殊成员函数包括构造函数、运算符重载和终结器等,有关成员函数的内容将在 5.3 节和 5.4 节详细介绍,本节重点介绍仓颉编程语言的属性。

2. 成员属性

仓颉编程语言中的属性是用 get 和 set 封装的对成员变量进行数据访问和控制的函数,使用它便于管理数据的读取和修改,这些属性提供了简洁的方式来实现封装性、访问控制、数据绑定等功能。属性可以定义在结构体、枚举和类中,当然也可以定义在第 6 章讲到的接口(interface)中,在类中定义的这种属性也叫成员属性。

先看以下示例:

```
class Counter {
    private var count:Int64=0
    public mut prop value: Int64 {
        get() {
            println("Getting count: ${count}")
            count
        }
        set(newValue) {
            println("Setting count: ${newValue}")
            count =newValue
        }
    }
```

为了在类中隐藏对私有成员 count 的访问,做到更强的安全性,设计了一个成员属性控制的变量 value,通过 value 来控制对 count 的访问。例如,在主程序中执行以下两行指令:

```
var counter =Counter()
println("Initial count: ${counter.value}")
```

可以得到访问 count 的结果是：

```
Getting count: 0
Initial count: 0
```

外部接口中 counter.value 并没有直接访问 count，但可以得到间接访问的结果。如果需要修改 count 属性值，只需要设计对 value 的操作即可。例如，在类中增加一个 increment 和 decrement 的控制：

```
    public func increment() {
        value =value +1
    }
    public func decrement() {
        value =value -1
    }
}
```

这个类就变成了一个双向计数器类，即可以根据程序设计的需要来进行增量或减量的操作。例如，在主程序中增加一行访问 increment 的指令：

```
counter.increment()
```

可以得到访问 count 的结果是：

```
Getting count: 0
Setting count: 1
```

若删除 increment 指令，增加一行访问 decrement 的指令：

```
counter.decrement()
```

可以得到访问 count 的结果是：

```
Getting count: 0
Setting count: -1
```

这种通过属性 value，让外部对 Counter 隐藏的成员变量 count 完全不感知的情况下，却可以做到同样的访问和修改操作，实现了有效的、更加安全的封装性。

成员属性的设置要注意以下几个问题。

(1) 用 prop 关键字来声明属性，带有 get 和可选的 set。

(2) 在 prop 前面可以声明需要的修饰符，如 open、override、redef 等。

(3) mut 修饰符用于声明可以修改的属性。无 mut 修饰符的属性：只能读取，无法修改，类似于 let 变量。带 mut 修饰符的属性：可以读取和修改，类似于 var 变量。

成员属性可以分为示例成员属性和静态成员属性，例如：

```
package demo
class A {
    public prop x: Int64 {              //示例成员属性
        get() {
            123
        }
    }
    public static prop y: Int64 {       //静态成员属性
        get() {
            321
        }
    }
}
```

```
main() {
    var a =A()
    println(a.x)                    // 123
    println(A.y)                    // 321
}
```

和 5.2.1 节的实例成员变量、静态成员变量一样，实例成员变量的属性用对象访问，而静态成员变量的属性的访问只能使用类名。

5.3 类的构造函数

构造函数也称构造器，是负责对象的创建的特殊成员函数。它的作用是创建对象时对成员变量进行初始化，在 5.2 节中已经用到了 init 关键字所引导的给成员变量赋值的构造函数。本节将详细介绍它的使用方法及应注意的几个问题。

和 struct 一样，仓颉类也支持定义普通构造函数和主构造函数。普通构造函数是以关键字 init 引导的构造函数，主构造函数是以类名引导的构造函数，不论使用哪一种构造函数，都必须完成类中所有未初始化示例成员变量的初始化。

以一个简单的学生成绩管理为例，学生的基本信息包括学号、姓名、年龄和学生成绩，可以利用 5.2 节介绍的方法，使用成员属性来隐藏学生成绩，例 5-5 是这个类的基本框架。

【例 5-5】 学生成绩管理类的基本框架。

```
package demo
class Student {
    let studentId: String=""
    let name: String=""
    var age: Int=0
    private var internalScore: Float64 =0.0
    public mut prop score: Float64 {
        get() {
            return internalScore
        }
        set(value) {
            if (value >=0.0 && value <=100.0) {
                internalScore =value
            }
        }
    }
    public prop grade: String {
        get() {
            if (this.score >=90.0) {return "A"}
            if (this.score >=80.0) {return "B"}
            if (this.score >=70.0) {return "C"}
            if (this.score >=60.0) {return "D"}
            return "还没有录入成绩！"
        }
    }
}
main() {
    let stdent =Student()               // 示例化 stdent 对象
    println(stdent.grade)               //输出学生成绩等级
}
```

上述程序定义了一个 Student 类，该类除了 4 个示例成员变量之外，还设置了 2 个成员属性 score 和 grade，都可以通过 score 对类内成员 internalScore 进行访问。运行以上程序，程序结果是：

```
还没有录入成绩!
```

基于 Student 类的成绩输入就是要用具体的学生信息示例化类中的抽象信息，这就需要在类中增加构造函数，可以使用普通构造函数，也可以使用主构造函数。

5.3.1 普通构造函数

普通构造函数是以关键字 init 开头的函数，以例 5-5 中的 Student 类为例，可以增加一个普通构造函数：

```
init(studentId: String,name: String,age: Int,internalScore: Float64){
      this.studentId=studentId
      this.name=name
      this.age=age
      this.internalScore=internalScore
  }
```

注意，构造函数的参数类型和数量一定要覆盖实例成员变量的所有成员。

在主程序中示例化 stdent 对象，可以修改主程序中的指令为如下格式：

```
let stdent =Student("250101","李晓泓",25,90.6)
    println(stdent.grade)
```

重新编译执行程序，程序输出结果为：

```
A
```

一个类中可以定义多个普通构造函数，但它们必须构成函数重载。例 5-6 是普通构造函数重载示例。

【例 5-6】 普通构造函数重载示例。

```
package demo
class Hanzi {
    let radical: String
    let component: String
    static var num=0
    public init(){                          //普通构造函数
        this.radical="氵"
        this.component="永"
        num++
      }
    public init(radical: String){           //普通构造函数重载
        this.radical=radical
        this.component=radical
        num++
      }
    public init(radical: String, component: String) {     //普通构造函数重载
        this.radical =radical
        this.component =component
        num++
      }
    public func compose(): String {
```