1. 枚举与结构体

枚举简单的说也是一种数据类型，只不过是这种数据类型只包含自定义的特定数据，它是一组有共同特性的数据的集合。枚举的功能为：

它声明在类中，可以通过实例化类来访问它的值。

枚举也可以定义构造函数来提供一个初始成员值；可以在原始的实现基础上扩展他们的功能

可以遵守协议来提供标准的功能

6.1枚举类型的创建与应用

swift中使用enum关键词来创建枚举并且把他们的整个定义放在一对大括号内：

enum 枚举名{

}

例如我们定义表示星期的枚举：

enum week{

case Sunday

case Monday

case TUESDAY

case WEDNESDAY

case THURSDAY

case FRIDAY

case Saturday}

var weekDay = week.THURSDAY

weekDay = .THURSDAY

switch weekDay{

case .Sunday: print("星期天")

case .Monday: print("星期一")

case .TUESDAY: print("星期二")

case .WEDNESDAY: print("星期三")

case .THURSDAY: print("星期四")

case .FRIDAY: print("星期五")

case .Saturday: print("星期六")

default: print("没有匹配值")}

以上程序执行输出结果为：

星期四

在使用时，枚举和其他类型一样，开发者可以在声明变量时将变量时将变量的类型指定为某个枚举类型，也可以通过变量初始化来使编译器自动推断出变量的类型。枚举中定义的枚举在使用时，开发和可以使用点语法来获取，示例如下：

还有另一种创建枚举类型的方式，示例如下：

enum alphabet{

case a,b,c,d

}

var apb:alphabet

apb = alphabet.a

实际上，如果一个变量的类型已经确认为某个枚举类型，那么开发者再进行变量赋值的时候，是可以将枚举类型省略掉的，直接使用点语法获取枚举值即可，示例如下：

//对apb进行修改

apb = .b

6.2枚举的原始值与相关值

枚举的原始值特性可以将枚举值与另一种数据类型进行绑定，相关值则可以为枚举值关联一些其他数据。通过相关值，开发者可以实现复杂的枚举类型。

6.2.1枚举的原始值

上节中创建的枚举其实并没有声明一个原始值类型，Swift语言中的枚举支持开发者声明一个原始类型，并将某个已经存在的类型的值与枚举值进行绑定，枚举指定原始值类型的语法与继承的语法有些类似，示例如下：

//为枚举类型指定一个原始值类型

enum original:Character{

case a = ‘a’

case b = ’b’

case c = ‘c’

case d = ‘d’

}

通过枚举类型中的rawValue属性来获取枚举的原始值，示例如下：

//创建枚举变量

var char = original.a

//获取char枚举变量的原始值“a”

var rawValue = char.rawValue

6.2.2枚举的相关值

在Swift语言中，枚举的相关值是在枚举中声明的枚举可以是不同的数据类型，且值不是事先设定的。相关值是当你在创建一个基于枚举成员的新常量或变量时才会被设置，并且每次当你这么做得时候，它的值可以时不同的。

以下实例中我们定义一个名为Student的枚举类型，它可以是Name的一个字符串(String),或者是Mark的一个相关值(Int,Int,Int)。示例如下：

enum Student{

case Name(String)

case Mark(Int,Int,Int)

}

var stuDetails = Student.Name(“HDU”)

var studMarks = Student.Mark(10,10,10)

switch studmarks{

case .Name(let studName):

print(“学生的名字:\(studName)”)

case .Mark(let Mark1,let Mark2,let Mark3):

print(“学生的成绩:\(Mark1),\(Mark2),\(Mark3)”)

}

以上程序执行输出结果为：

学生的成绩是：10，10，10

6.3Swift结构体

Swift结构体是构建代码所用的一种通用且灵活的构造体。

我们可以为结构体定义属性(常量、变量)和添加方法，从而扩展结构体的功能。

与C语言不同的是：

结构体不需要包含实现文件和接口。

结构体允许我们创建一个单一文件，且系统会自动生成面向其他代码的外部接口。

结构体总是通过被复制的方式在代码中传递，因此它的值是不可修改的。

语法

我们通过关键字struct来定义结构体：

struct nameStruct{

Definition 1

Definition 2

....

Definition N

}

### **实例**

我们定义一个名为 MarkStruct 的结构体 ，结构体的属性为学生三个科目的分数，数据类型为 Int：

struct MarkStruct{

var mark1: Int

var mark2: Int

var mark3: Int}

我们可以通过结构体名来访问结构体成员。

结构体实例化使用 **let** 关键字：

import Cocoa

struct studentMarks {

var mark1 = 100

var mark2 = 78

var mark3 = 98}let marks = studentMarks()print("Mark1 是 \(marks.mark1)")print("Mark2 是 \(marks.mark2)")print("Mark3 是 \(marks.mark3)")

以上程序执行输出结果为：

Mark1 是 100Mark2 是 78Mark3 是 98

实例中，我们通过结构体名 'studentMarks' 访问学生的成绩。结构体成员初始化为mark1, mark2, mark3，数据类型为整型。

然后我们通过使用 **let** 关键字将结构体 studentMarks() 实例化并传递给 marks。

最后我们就通过 **.** 号来访问结构体成员的值。

以下实例化通过结构体实例化时传值并克隆一个结构体：

import Cocoa

struct MarksStruct {

var mark: Int

init(mark: Int) {

self.mark = mark

}}var aStruct = MarksStruct(mark: 98)var bStruct = aStruct // aStruct 和 bStruct 是使用相同值的结构体！

bStruct.mark = 97print(aStruct.mark) // 98print(bStruct.mark) // 97

以上程序执行输出结果为：

9897

## 结构体应用

在你的代码中，你可以使用结构体来定义你的自定义数据类型。

结构体实例总是通过值传递来定义你的自定义数据类型。

按照通用的准则，当符合一条或多条以下条件时，请考虑构建结构体：

* 结构体的主要目的是用来封装少量相关简单数据值。
* 有理由预计一个结构体实例在赋值或传递时，封装的数据将会被拷贝而不是被引用。
* 任何在结构体中储存的值类型属性，也将会被拷贝，而不是被引用。
* 结构体不需要去继承另一个已存在类型的属性或者行为。

举例来说，以下情境中适合使用结构体：

* 几何形状的大小，封装一个width属性和height属性，两者均为Double类型。
* 一定范围内的路径，封装一个start属性和length属性，两者均为Int类型。
* 三维坐标系内一点，封装x，y和z属性，三者均为Double类型。

结构体实例是通过值传递而不是通过引用传递

import Cocoa

struct markStruct{

var mark1: Int

var mark2: Int

var mark3: Int

init(mark1: Int, mark2: Int, mark3: Int){

self.mark1 = mark1

self.mark2 = mark2

self.mark3 = mark3

}}

print("优异成绩:")var marks = markStruct(mark1: 98, mark2: 96, mark3:100)print(marks.mark1)print(marks.mark2)print(marks.mark3)

print("糟糕成绩:")var fail = markStruct(mark1: 34, mark2: 42, mark3: 13)print(fail.mark1)print(fail.mark2)print(fail.mark3)

以上程序执行输出结果为：

优异成绩:9896100糟糕成绩:344213

以上实例中我们定义了结构体 markStruct，三个成员属性：mark1, mark2 和 mark3。结构体内使用成员属性使用 self 关键字。

从实例中我们可以很好的理解到结构体实例是通过值传递的。