# 第九章 异常处理与类型转换

在软件开发过程中，通常希望程序在运行过程中，出现了未知异常，还能够继续的运行下去，所以引入了异常处理。并且在开发中还经常需要利用到类型的强制转换。

## 9.1异常处理

程序中经常会出现一些语法错误和逻辑错误。这些错误能够被编译器检查出来，且程序终止运行。下面来介绍两种异常处理方式。

### 9.1.1异常的处理

对于没有异常捕获的程序中，是如何来修改和处理程序中的错误？例如：

var x:Int = 6

var y:Int = 0

var z = x/y

print(z)

程序必然运行不通过，如果要增加程序的健壮性，可以这样修改程序，实例如下：

var x:Int = 6

var y:Int = 0

var z:Int?

if y==0{

print(“除数不能为零”)

}else{

z = x/y

}

print(z)

通过以上的两个例子，可以看出对于程序运行中出现的异常，程序只有终止运行，让开发者自己手动的去修改程序。下面来介绍一种，在程序运行中出现未知异常以后，通过捕获异常，使程序继续的运行下去。

### 9.1.2异常的捕获

如果对于程序中出现的异常不处理，程序会终止运行。这种情况非常的不利于开发者编写程序，Swift中为开发者提供了异常的处理方法，即使用do-catch结构来捕获异常。

利用do-catch异常结构处理方法，需要把有可能抛出异常的代码语句放入到do结构块当中，如果此语句出现了异常，会判断所属异常类型，执行相匹配异常代码。此处需要注意的是，在Swift语言中，所有的错误和异常都有Error协议来指定的，也就是说开发者必须要自己编写自定义的枚举类型。示例如下：

enum TestError:Error{

case HighError

case MidError

case lowError

}

func TestFunc(param:bool)throws->Void{

if param{

print(“ok”)

}else{

throw TestError.lowError

}

}

do{

TestFunc(param:false)

}catch TestError.lowError{

print(“this is lowError”)

}

do语句块中存放了一个能抛出异常的函数体，当函数抛出异常时，会与catch的异常名进行比对，如果对比成功，即捕获到异常，程序就会运行相对应异常下的语句。

## 9.2类型转换

对于Swift中的类型转换，一般分为两步，第一步是相对其进行类型检查，判断某个实例时候属于具体类型。第二步就是对其类型的实例进行类型的转换。

### 9.2.1类型检查

Swift语言中要判断某个实例是否属于某个具体类型，可以使用is关键字，组成一个判断语句来返回一个布尔值，可以根据布尔值的真假来判断是否同属一个类型。实例如下:

var str = “HDU”

if str is String{

print(“str 的类型是String”)

}

类型检查主要是用在有继承关系的类上，即子类实例进行父类类型检查时为真，父类实例进行子类类型的检查时为假。

### 9.2.2类型转换

Swift语言中的类型转换是使用as关键字。同类型检查相似，类型转换也是主要用在有继承关系的类上，即一个父类类型的集合可以接受子类类型的实例，在父类类型也可以直接转换成子类类型。实例如下：

class MyClass:BaseClass{

var name:String?

}

class MySubClassOne:MyClass{

var count:Int?

}

class MySubClassTwo:MyClass{

var isBiger:Bool?

}

var obj1 = MyClass()

obj1.name=”HD”

var obj2=MySubClassOne()

obj2.count =100

var obj3 =MySubClassTwo()

obj3.isBiger=true

var array:[MyClass] =[obj1,obj2,obj3]

for var in 0..<array.count{

var obj = array[i]

if obj is MySubClassOne{

print((obj as! MySubClassOne).count!)

continue

}

if obj is MySubClassTwo{

print((obj as! MySubClassTwo).isBiger!)

continue

}

if obj is MySubClassOne{

print(obj.name!)

}

}

注：在使用类型转换时，需要使用as?和as!方式。as?是一种比较安全的转换方式，其会将类型转换后的结果映射为Optional值，如果类型转换成功，则值为原实例，如果类型转换失败，则会放回Optional值nil.而as!是一种强制转换方式，其默认此次转换一定成功，如果转换失败，则会产生运行时错误，程序会崩溃。开发者在使用as!进行类型转换时，必须保证实例的真实类型和要转化的类型一致。

本章小节

本章中介绍了异常以及异常的处理，在Swift中所需要抛出的异常类型，需要开发者自己来定义相对应的异常，这样在编写过程中可以高度自定义所要抛出的异常。类型转换主要是指子类与父类之间的转换，在转换过程中必须要对类之间的继承关系要比较清楚。本章的内容是难点也是重点，在实际的开发过程中应用的非常频繁，希望读者认真掌握。