

错误处理

@M了个J

<https://github.com/CoderMJLee>

<http://cnblogs.com/mjios>

码拉松



实力IT教育 www.520it.com

错误类型

- 开发过程常见的错误
 - 语法错误（编译报错）
 - 逻辑错误
 - 运行时错误（可能会导致闪退，一般也叫做异常）
 -

- Swift中可以通过Error协议自定义运行时的错误信息

```
enum SomeError : Error {  
    case illegalArg(String)  
    case outOfBounds(Int, Int)  
    case outOfMemory  
}
```

- 函数内部通过throw抛出自定义Error，可能会抛出Error的函数必须加上throws声明

```
func divide(_ num1: Int, _ num2: Int) throws -> Int {  
    if num2 == 0 {  
        throw SomeError.illegalArg("0不能作为除数")  
    }  
    return num1 / num2  
}
```

- 需要使用try调用可能会抛出Error的函数

```
var result = try divide(20, 10)
```

■ 可以使用do-catch捕捉Error

```
func test() {  
    print("1")  
    do {  
        print("2")  
        print(try divide(20, 0))  
        print("3")  
    } catch let SomeError.illegalArg(msg) {  
        print("参数异常:", msg)  
    } catch let SomeError.outOfBounds(size, index) {  
        print("下标越界:", "size=\(size)", "index=\(index)")  
    } catch SomeError.outOfMemory {  
        print("内存溢出")  
    } catch {  
        print("其他错误")  
    }  
    print("4")  
}
```

```
test()  
// 1  
// 2  
// 参数异常: 0不能作为除数  
// 4
```

■ 抛出Error后，try下一句直到作用域结束的代码都将停止运行

处理Error

■ 处理Error的2种方式

① 通过do-catch捕捉Error

② 不捕捉Error，在当前函数增加throws声明，Error将自动抛给上层函数

✓ 如果最顶层函数（main函数）依然没有捕捉Error，那么程序将终止

```
func test() throws {  
    print("1")  
    print(try divide(20, 0))  
    print("2")  
}  
try test()  
// 1  
// Fatal error: Error raised at top level
```

```
do {  
    print(try divide(20, 0))  
} catch is SomeError {  
    print("SomeError")  
}
```

```
func test() throws {  
    print("1")  
    do {  
        print("2")  
        print(try divide(20, 0))  
        print("3")  
    } catch let error as SomeError {  
        print(error)  
    }  
    print("4")  
}  
try test()  
// 1  
// 2  
// illegalArg("0不能作为除数")  
// 4
```

try?、try!

- 可以使用try?、try!调用可能会抛出Error的函数，这样就不用去处理Error

```
func test() {  
    print("1")  
    var result1 = try? divide(20, 10) // Optional(2), Int?  
    var result2 = try? divide(20, 0) // nil  
    var result3 = try! divide(20, 10) // 2, Int  
    print("2")  
}  
test()
```

- a、b是等价的

```
var a = try? divide(20, 0)  
  
var b: Int?  
do {  
    b = try divide(20, 0)  
} catch { }
```

rethrows

■ **rethrows**表明：函数本身不会抛出错误，但调用闭包参数抛出错误，那么它会将错误向上抛

```
func exec(_ fn: (Int, Int) throws -> Int, _ num1: Int, _ num2: Int) rethrows {  
    print(try fn(num1, num2))  
}  
// Fatal error: Error raised at top level  
try exec(divide, 20, 0)
```

defer

■ **defer**语句：用来定义以任何方式（抛错误、**return**等）离开代码块前必须要执行的代码

□ **defer**语句将延迟至当前作用域结束之前执行

```
func open(_ filename: String) -> Int {  
    print("open")  
    return 0  
}  
  
func close(_ file: Int) {  
    print("close")  
}
```

■ **defer**语句的执行顺序与定义顺序相反

```
func fn1() { print("fn1") }  
func fn2() { print("fn2") }  
func test() {  
    defer { fn1() }  
    defer { fn2() }  
}  
  
test()  
// fn2  
// fn1
```

```
func processFile(_ filename: String) throws {  
    let file = open(filename)  
    defer {  
        close(file)  
    }  
    // 使用file  
    // ....  
    try divide(20, 0)  
  
    // close将会在这里调用  
}  
  
try processFile("test.txt")  
// open  
// close  
// Fatal error: Error raised at top level
```