# 可选项

@M了个J

https://github.com/CoderMJLee

http://cnblogs.com/mjios

码拉松



小码哥教育 SEEMYGO

实力IT教育 www.520it.com

# 可选项（Optional）

- 可选项，一般也叫可选类型，它允许将值设置为nil
- 在类型名称后面加个问号 ？ 来定义一个可选项

```
var name: String? = "Jack"
name = nil
```

```
var age: Int? // 默认就是nil
age = 10
age = nil
```

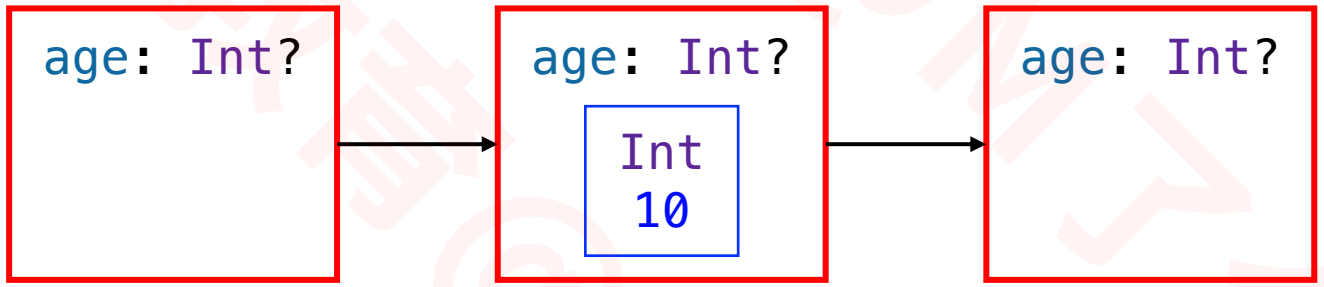
```
var array = [1, 15, 40, 29]
func get(_ index: Int) -> Int? {
    if index < 0 || index >= array.count {
        return nil
    }
    return array[index]
}
```

```
print(get(1)) // Optional(15)
print(get(-1)) // nil
print(get(4)) // nil
```

# 强制解包（Forced Unwrapping）

- 可选项是对其他类型的一层包装，可以将它理解为一个盒子
  - 如果为nil，那么它是个空盒子
  - 如果不为nil，那么盒子里装的是：被包装类型的数据

```
var age: Int? // 默认就是nil
age = 10
age = nil
```




- 如果要从可选项中取出被包装的数据（将盒子里装的东西取出来），需要使用感叹号 ！ 进行强制解包

```
var age: Int? = 10
let ageInt: Int = age!
ageInt += 10
```

- 如果对值为nil的可选项（空盒子）进行强制解包，将会产生运行时错误

```
var age: Int?
age!
```

```
Fatal error: Unexpectedly found nil while unwrapping an Optional value
```

小码哥教育 SEEMYGO

# 判断可选项是否包含值

```
let number = Int("123")
if number != nil {
    print("字符串转换整数成功：\(number!)")
} else {
    print("字符串转换整数失败")
}
// 字符串转换整数成功：123
```

# 可选项绑定（Optional Binding）

- 可以使用可选项绑定来判断可选项是否包含值
- 如果包含就自动解包，把值赋给一个临时的常量(let)或者变量(var)，并返回true，否则返回false

```
if let number = Int("123") {
    print("字符串转换整数成功：\(number)")
    // number是强制解包之后的Int值
    // number作用域仅限于这个大括号
} else {
    print("字符串转换整数失败")
}
// 字符串转换整数成功：123
```

```
enum Season : Int {
    case spring = 1, summer, autumn, winter
}
if let season = Season(rawValue: 6) {
    switch season {
    case .spring:
        print("the season is spring")
    default:
        print("the season is other")
    }
} else {
    print("no such season")
}
// no such season
```

小码哥教育 SEEMYGO

# 等价写法

```swift
if let first = Int("4") {
    if let second = Int("42") {
        if first < second && second < 100 {
            print("\(first) < \(second) < 100")
        }
    }
}
// 4 < 42 < 100
```

```swift
if let first = Int("4"),
    let second = Int("42"),
    first < second && second < 100 {
    print("\(second) < \(second) < 100")
}
// 4 < 42 < 100
```

# while循环中使用可选项绑定

```
// 遍历数组，将遇到的正数都加起来，如果遇到负数或者非数字，停止遍历
var strs = ["10", "20", "abc", "-20", "30"]
```

```
var index = 0
var sum = 0
while let num = Int(strs[index]), num > 0 {
    sum += num
    index += 1
}
print(sum)
```

# 空合并运算符 ??（Nil-Coalescing Operator）

```
public func ?? <T>(optional: T?, defaultValue: @autoclosure () throws -> T?) rethrows -> T?
```

```
public func ?? <T>(optional: T?, defaultValue: @autoclosure () throws -> T) rethrows -> T
```

- a ?? b
  - a 是可选项
  - b 是可选项 或者 不是可选项
  - b 跟 a 的存储类型必须相同
  - 如果 a 不为nil，就返回 a
  - 如果 a 为nil，就返回 b
  - 如果 b 不是可选项，返回 a 时会自动解包

# 空合并运算符 ??（Nil-Coalescing Operator）

```swift
let a: Int? = 1
let b: Int? = 2
let c = a ?? b // c是Int? , Optional(1)
```

```swift
let a: Int? = nil
let b: Int? = 2
let c = a ?? b // c是Int? , Optional(2)
```

```swift
let a: Int? = nil
let b: Int? = nil
let c = a ?? b // c是Int? , nil
```

```swift
let a: Int? = 1
let b: Int = 2
let c = a ?? b // c是Int , 1
```

```swift
let a: Int? = nil
let b: Int = 2
let c = a ?? b // c是Int , 2
```

```swift
let a: Int? = nil
let b: Int = 2
// 如果不使用??运算符
let c: Int
if let tmp = a {
    c = tmp
} else {
    c = b
}
```

小码哥教育 SEEMYGO

# 多个 ?? 一起使用

```
let a: Int? = 1
let b: Int? = 2
let c = a ?? b ?? 3 // c是Int , 1
```

```
let a: Int? = nil
let b: Int? = 2
let c = a ?? b ?? 3 // c是Int , 2
```

```
let a: Int? = nil
let b: Int? = nil
let c = a ?? b ?? 3 // c是Int , 3
```

# ??跟if let配合使用

```
let a: Int? = nil
let b: Int? = 2
if let c = a ?? b {
    print(c)
}
// 类似于if a != nil || b != nil
```

```
if let c = a, let d = b {
    print(c)
    print(d)
}
// 类似于if a != nil && b != nil
```

小码哥教育 SEEMYGO

# if语句实现登陆

```swift
func login(_ info: [String : String]) {
    let username: String
    if let tmp = info["username"] {
        username = tmp
    } else {
        print("请输入用户名")
        return
    }
    let password: String
    if let tmp = info["password"] {
        password = tmp
    } else {
        print("请输入密码")
        return
    }
    // if username ....
    // if password ....
    print("用户名：\(username)", "密码：\(password)", "登陆ing")
}
```

```swift
login(["username" : "jack", "password" : "123456"]) // 用户名：jack 密码：123456 登陆ing
login(["password" : "123456"]) // 请输入密码
login(["username" : "jack"]) // 请输入用户名
```

# guard语句

```
guard 条件 else {
    // do something....
    退出当前作用域
    // return、break、continue、throw error
}
```

- 当guard语句的条件为false时，就会执行大括号里面的代码
- 当guard语句的条件为true时，就会跳过guard语句
- guard语句特别适合用来“提前退出”

- 当使用guard语句进行可选项绑定时，绑定的常量(let)、变量(var)也能在外层作用域中使用

```
func login(_ info: [String : String]) {
    guard let username = info["username"] else {
        print("请输入用户名")
        return
    }
    guard let password = info["password"] else {
        print("请输入密码")
        return
    }
    // if username ....
    // if password ....
    print("用户名：\(username)", "密码：\(password)", "登陆ing")
}
```

# 隐式解包（Implicitly Unwrapped Optional）

- 在某些情况下，可选项一旦被设定值之后，就会一直拥有值
- 在这种情况下，可以去掉检查，也不必每次访问的时候都进行解包，因为它能确定每次访问的时候都有值
- 可以在类型后面加个感叹号！定义一个隐式解包的可选项

```
let num1: Int! = 10
let num2: Int = num1
if num1 != nil {
    print(num1 + 6) // 16
}
if let num3 = num1 {
    print(num3)
}
```

```
let num1: Int! = nil
// Fatal error: Unexpectedly found nil while implicitly unwrapping an Optional value
let num2: Int = num1
```

# 字符串插值

- 可选项在字符串插值或者直接打印时，编译器会发出警告

```
var age: Int? = 10
print("My age is \(age)")
```

- 至少有3种方法消除警告

```
print("My age is \(age!)")
// My age is 10
```

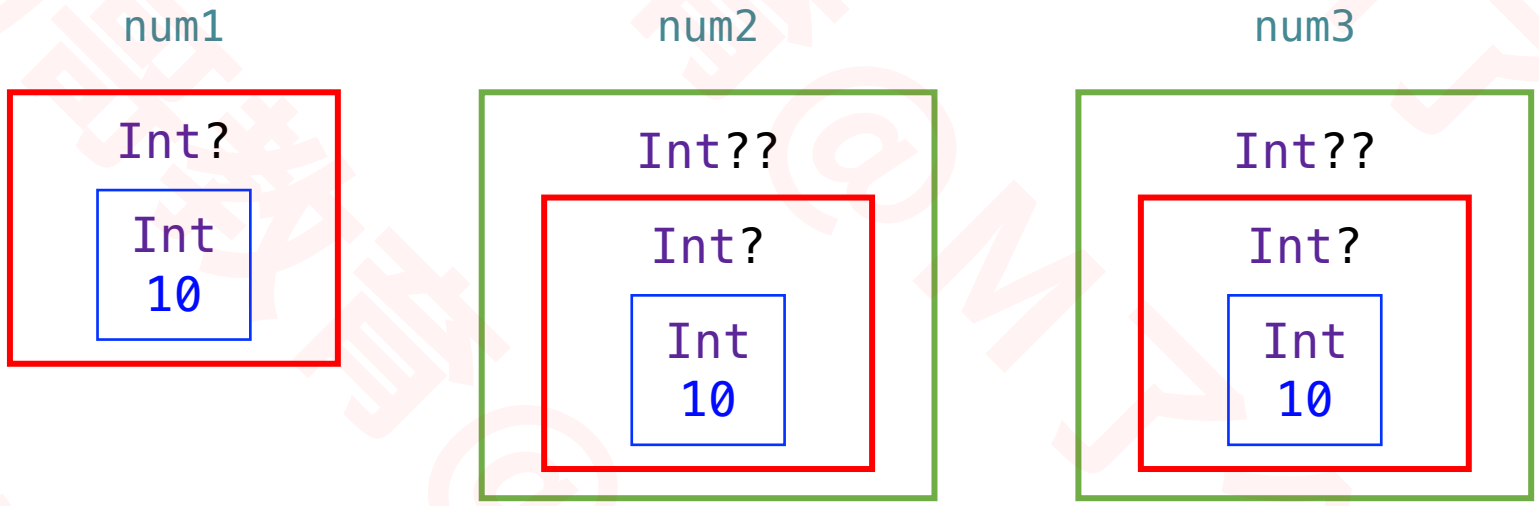
```
print("My age is \(String(describing: age))")
// My age is Optional(10)
```

```
print("My age is \(age ?? 0)")
// My age is 10
```

# 多重可选项

```
var num1: Int? = 10
var num2: Int?? = num1
var num3: Int?? = 10
```

```
print(num2 == num3) // true
```




- 可以使用lldb指令 **frame variable -R** 或者 **fr v -R** 查看区别

小码哥教育 SEEMYGO

# 多重可选项

```
var num1: Int? = nil
var num2: Int?? = num1
var num3: Int?? = nil
```

```
print(num2 == num3) // false
```

```
(num2 ?? 1) ?? 2 // 2
(num3 ?? 1) ?? 2 // 1
```