

文档

析构过程

4.2

▾

Swift 编程语言

/

析构过程

这是一篇社区协同翻译的文章，你可以点击右边区块信息里的『改进』按钮向译者提交改进建议。

析构器 会在类的实例销毁之前被立即调用。使用 `deinit` 关键字来标示析构器，类似于使用 `init` 关键字标示构造器。析构器仅适用于 class 类型。

析构器是如何运作的

当不再需要某一个实例时，Swift 会自动销毁该实例，以释放资源。Swift 通过 *自动引用计数*（ARC）来管理实例内存，就如 [自动引用计数](#) 中所述。通常在实例释放时，你无需行手动清理。但是，当你使用自己的资源时，可能需要自己执行一些额外的清理。例如，如果你创建了一个自定义类以打开文件并向其写入一些数据，则可能需要在销毁类实例之前关闭该文件。

在类的定义中, 类最多只能有一个析构器。析构器不接受任何参数，并且没有括号：

```
deinit {  
    //执行析构器  
}
```

在实例销毁之前，会自动调用析构器。你不能自己调用析构器。父类的析构器由其子类继承，父类析构器会在子类析构器实现的末尾自动调用。即使子类不提供自己的析构器，父类析构器也会被调用。

因为实例在调用析构器之后才会被释放，所以析构器可以访问调用它的实例的所有属性，并可以根据这些属性修改其行为（例如查找需要关闭的文件的名称）。

析构器实践

这是一个析构器的实践。这个例子为一个简单的游戏定义了两个新类型，`Bank` 和

Infinity

翻译进度

3

分块数量

3

参与人数



jihongboo 翻译于 5个月

0

重译

由 Aufree 审阅



`Player`。 `Bank` 类管理一种虚拟货币，且流通的货币永远不会超过10,000枚。在游戏中有且只有一个 `Bank`，因此 `Bank` 使用类来实现，并且该类含有类型属性以及用于存储和管理其当前状态的类型方法：

```
class Bank {
    static var coinsInBank = 10_000

    static func distribute(coins numberOfCoinsRequested: Int) -> Int {
        let numberOfCoinsToVend = min(numberOfCoinsRequested, coinsInBank)
        coinsInBank -= numberOfCoinsToVend
        return numberOfCoinsToVend
    }

    static func receive(coins: Int) {
        coinsInBank += coins
    }
}
```

`Bank` 使用 `coinsInBank` 属性跟踪它当前持有的硬币的数量。它还提供了两个方法--- `distribute(coins:)` 和 `receive(coins:)` ---来处理硬币的分配和收集。

`distribute(coins:)` 方法在分配硬币之前先检查银行中是否有足够的硬币。如果没有足够的硬币，`Bank` 返回的数字将会小于请求的数字（如果银行中没有硬币则返回零）。该方法返回一个整数值，表示实际可提供的硬币数。

`receive(coins:)` 方法只是将得到的硬币数量添加回银行的总硬币数中。

`Player` 类描述了游戏中的玩家。每个玩家的钱包在任何时候都有一定数量的硬币。这个数量是由玩家的 `coinsInPurse` 属性表示：

```
class Player {
    var coinsInPurse: Int

    init(coins: Int) {
        coinsInPurse = Bank.distribute(coins: coins)
    }

    func win(coins: Int) {
        coinsInPurse += Bank.distribute(coins: coins)
    }

    deinit {
        Bank.receive(coins: coinsInPurse)
    }
}
```

每个 `Player` 实例在初始化期间，会得到来自银行的指定数量硬币。但是如果银行没有足够的硬币，那么 `Player` 实例只能拿到少于指定数量的硬币。



jihongboo 翻译于 5个月

👍 0

重译

由 Aufree 审阅



jihongboo 翻译于 5个月

👍 0

重译

由 RecherJ 审阅

`Player` 类定义了一个 `win(coins:)` 方法，它从银行中取回一定数量的硬币并将它们添加到玩家的钱包中。`Player` 类还实现了一个析构器，它会在 `Player` 实例被释放前调用。在这个例子中，析构器只是将所有玩家的硬币归还给银行：

```
var playerOne: Player? = Player(coins: 100)
print("A new player has joined the game with \$(playerOne!.coinsInPurse) coins"
// 打印 「一个新玩家加入了游戏并获得了100个硬币」
print("There are now \$(Bank.coinsInBank) coins left in the bank")
// 打印 「银行现在剩余9900个硬币」
```

如果剩余硬币充足，创建一个新 `Player` 实例同时就需要立即分配100个硬币。这个 `Player` 实例存储在一个名为 `playerOne` 的可选值变量中。这里使用可选变量，因为玩家可以随时离开游戏。可选项方式可让你跟踪游戏中当前玩家是否存在。

因为 `playerOne` 是一个可选值，所以当我们尝试访问它的 `coinsInPurse` 属性以打印其默认硬币数，或调用它的 `win(coins:)` 方法时，我们需要加上一个感叹号（`!`）来解包：

```
playerOne!.win(coins: 2_000)
print("PlayerOne won 2000 coins & now has \$(playerOne!.coinsInPurse) coins")
// 打印 「PlayerOne 赢了2000个硬币，现在有2100个硬币」
print("The bank now only has \$(Bank.coinsInBank) coins left")
// 打印 「银行现在只剩下7900个硬币」
```

现在，玩家赢得了 2,000 个硬币。玩家的钱包现在包含 2,100 个硬币，银行只剩下 7,900 个硬币。

```
playerOne = nil
print("PlayerOne has left the game")
// 打印 「PlayerOne 离开游戏」
print("The bank now has \$(Bank.coinsInBank) coins")
// 打印 「银行现在有10000个硬币」
```

玩家现在已离开了游戏。通过将可选的 `playerOne` 变量置为 `nil` 来表示「没有 `Player` 实例。」此时，`playerOne` 变量对 `Player` 实例的引用被销毁。没有其他属性或变量仍然引用着 `Player` 实例，因此它就会被销毁以释放其内存。在销毁前，`Player` 自动调用了它的析构器被，并将它的硬币送回银行。

接




我们的翻译工作遵照 [CC 协议](#)，如果我们的工作有侵犯到您的权益，请及时联系我们。

← 上一篇

下一篇 →

 点赞

 参与译者：3

更多职位





 讨论数量：0

 发起讨论

☐ 只看当前版本讨论

暂无话题~

兄弟社区

-  Laravel China
-  PythonCaff.com
-  GolangCaff.com
-  VuejsCaff.com

资源推荐

资源推荐

其他信息

-  软件外包
-  商务合作
-  联系站长

