使用func和closure加工数据

什么时候需要把参数自动转化为closure? (https://www.boxueio.com/series/functions-and-closure/ebook/158)

返回视频 (/series/functions-and-closure)

Closure什么时候需要escaping?

Back to series (/series/functions-and-closure)

字号
字号
默认主题
金色主题
暗色主题

如果你还记得我们在讨论delegate还是callback的时候，当我们用一个 `class` 类型来实现callback时，要在实现callback的closure中，使用capture list：

```
fav.buttonPressed = { [weak counter] index in
    counter?.buttonPressed(at: index)
}
```

这是为了避免在view controller和view之间，意外造成循环引用。但是，如果你记忆力更好一些，就会发现，我们之前在讲到集合类型的时候，给那些 `map` ， `filter` 以及 `reduce` 传递的closure参数，并没有使用什么capture list：

```
let numbers = [1, 2, 3]

numbers.map { number in
    number * 2
}
```

究竟考虑是否应该使用capture list的标准是什么呢?

从是否需要capture list说起

一个比较简单的准则，就是看我们使用的closure是否会独立存活于它的执行环境之外。如果是，那么你通常要考虑正确使用capture list；如果不是，那你完全可以忽略这个东西。

在我们的第一个例子里，由于 `fav` 已经对 `counter` 有了一个strong reference，而 `counter` 是完全独立存活于 `fav` 之外的，因此，为了避免让 `counter` 又意外引用回 `fav` ，我们使用了capture list。

而在 `map` 里，closure只是完全存活在 `map` 的执行过程内的，它没有任何机会在 `map` 的作用域外被保存和使用，因此，我们不用在这种closure中，考虑capture list的事情。

在Swift里，这两种closure分别叫做escaping closure和non escaping closure。对于后者而言，使用它们相对是安全的，我们无需过多关心循环引用的问题。**当把一个closure用作函数参数时，默认都是non escaping属性的**，也就是说，它们只负责执行逻辑，但不会被外界保存和使用。

> 这点和Swift 2是完全相反的，在Swift 2里，closure默认都是escaping属性的，大家可以查看SE-0103 Make non-escaping closures the default (https://github.com/apple/swift-evolution/blob/master/proposals/0103-make-noescape-default.md)了解更多细节。

一旦closure有机会逃逸到函数作用域外部，我们不仅要在定义它的时候考虑使用capture list。还必须在声明这种参数的时候，使用 `@escaping` 来修饰它。例如，我们之前在模拟OC运行时特性时，定义的 `makeDescriptor` 方法：

```
func makeDescriptor<Key, Value>(
    key: @escaping (Key) -> Value,
    _ isAscending: @escaping (Value, Value) -> Bool
) -> SortDescriptor<Key> {

        return { isAscending(key($0), key($1)) }
}
```

在它的实现里，由于我们返回的 `SortDescriptor<T>` 一旦被其他变量保存，它的两个参数 `key` 和 `isAscending` 就完全脱离 `makeDescriptor` 的作用域了，因此，我们必须使用 `@escaping` 来修饰它们。

Closure默认不为non escaping的情况

刚才我们提到过，作为函数参数的closure默认是non escaping属性的。但在下面这两种情况里，closure默认是escaping属性的：

首先，所有自定义类型的closure属性，默认是escaping的。例如：我们之前已经使用过的 `fav.buttonPressed`，想想这也自然，毕竟我们把closure单独保存了起来；

其次，如果closure被封装在一个optional里，它默认是escaping的，来看下面这个例子：

```
func calc(_ n: Int, by: ((Int) -> Int)?) -> Int {
    guard let by = by else { return n }

    return by(n)
}
```

其实这也合情合理，如果参数 by 需要一个optional类型，表示这个closure应该是从外部传递进来的，而不像是 map 那种定义在参数原地的（否则我们也就不用optional类型了）。既然是来自外部的closure，它就已经生于函数作用域之外了，当然也就是escaping属性的。

总之，之所以要使用 `@escaping` 来修饰closure参数，是为了时刻提醒你，不要让它们成为脱缰的野马。把一个外部的closure传递给 `@escaping` 参数的时候，你也要时刻记着："喔，我可能会创建循环引用，要认真考虑是否需要在closure内使用capture list来避免这个问题。"

◀ 什么时候需要把参数自动转化为closure?
(https://www.boxueio.com/series/functions-and-closure/ebook/158)

返回视频 ▶
(/series/functions-and-closure)