

Benny Huo

学海无涯，其乐无穷



闲话 Swift 协程 (2) : 将回调改写成 async 函数

📅 2021-10-13 | 📅 2022-12-31 | 👁 2142

📖 3k | ⌚ 5 分钟

最理想的情况下，系统、第三方框架当中使用回调的 API 都最好在一夜之间改成 async 函数，显然这不太现实。

- [闲话 Swift 协程 \(0\) : 前言](#)
- [闲话 Swift 协程 \(1\) : Swift 协程长什么样?](#)
- [闲话 Swift 协程 \(2\) : 将回调改写成 async 函数](#)
- [闲话 Swift 协程 \(3\) : 在程序当中调用异步函数](#)
- [闲话 Swift 协程 \(4\) : TaskGroup 与结构化并发](#)
- [闲话 Swift 协程 \(5\) : Task 的取消](#)
- [闲话 Swift 协程 \(6\) : Actor 和属性隔离](#)
- [闲话 Swift 协程 \(7\) : GlobalActor 和异步函数的调度](#)
- [闲话 Swift 协程 \(8\) : TaskLocal](#)
- [闲话 Swift 协程 \(9\) : 异步函数与其他语言的互调用](#)

我们前面已经简单介绍了 Swift 的协程，可以确认的一点是，如果你只是看了上一篇文章，那么你一定还是不会用这一个特性。你一定还有一些疑问：

- 异步函数是谁提供的？
- 我可以自己定义吗？
- 我该怎么正确地定义一个异步函数？

异步函数谁都可以提供，不然它的应用范围就会大大受限制，因此我们既可以有机会使用到系统或者第三方框架提供的异步函数，也自然有机会自己去定义。那关键的问题就是如何定义异步函数了。

我们先随便定义一个函数：

```
1 func hello() -> Int{
2     1
3 }
```

这个函数返回了一个整数 1。接下来我们把它改造成异步函数，只需要加上 `async` 关键字：

```
1 func hello() async -> Int{
2     1
3 }
```

那么，它现在真的是异步的吗？当然不是，它只是长得像罢了。

`async` 关键字并不会真正带来异步，那么异步的能力是谁提供的？这时候我们就要想想，过去我们见到的异步函数都是什么样的：

```
1 func helloAsync(onComplete: @escaping (Int) -> Void) {
2     DispatchQueue.global().async {
3         onComplete(Int(arc4random()))
4     }
5 }
```

这是一个很简单的例子，我们在 `helloAsync` 当中通过 `DispatchQueue` 将代码逻辑调度到 `global()` 上，使得回调 `onComplete` 的调用脱离了 `helloAsync` 的调用栈。调用这个函数的样子就像这样：

```
1 helloAsync { result in
2     print("Got result from callback: \(result)")
3 }
```

这么看来，我们在异步函数当中都应该有这么个切换调用栈的过程，并且有个类似于回调的东西将结果能传递出去。那在 Swift 协程当中，谁来扮演这个角色呢？

这里就要稍微提一下 Swift 协程的设计原理了。它采用了一种叫做 Continuation Passing Style 的设计思路（熟悉 Kotlin 的朋友是不是觉得非常熟悉？），而这个所谓的 Continuation 就充当了回调的作用。我们把 Swift 标准库当中提供的 Continuation 的定义给出来，大家简单了解一下它的形式即可：

```

1  @frozen public struct UnsafeContinuation<T, E> where E : Error {
2
3      public func resume(returning value: T) where E == Never
4
5      public func resume(returning value: T)
6
7      public func resume(throwing error: E)
8  }

```

注意到它实际上有两种类型的函数，一种是 returning，一种是 throwing。也就是说，对于任何一段代码逻辑，其执行的结果都无非返回结果和抛出异常两种。Continuation 其实就是描述协程当中异步代码在挂起点的状态，而当程序需要恢复执行时，调用它对应的 resume 函数即可。

好了，现在我们知道了有了 Continuation 这个东西了，相当于我们已经知道对于 Swift 的 async 函数而言，我们可以通过 Continuation 来传递异步结果。那么下一个问题就是如何获取这个 Continuation 的实例呢？Swift 标准库提供了相应的函数来做到这一点：

```

1  public func withCheckedContinuation<T>(<
2      function: String = #function,
3      _ body: (CheckedContinuation<T, Never>) -> Void
4  ) async -> T
5
6  public func withCheckedThrowingContinuation<T>(<
7      function: String = #function,
8      _ body: (CheckedContinuation<T, Error>) -> Void
9  ) async throws -> T

```

如果我们的异步函数不会抛出异常，那就用 withCheckedContinuation 来获取 Continuation；如果会抛出异常，那就用 withCheckedThrowingContinuation。这么看来，改造前面的回调的方法就显而易见了：

```

1  func helloAsync() async -> Int {
2      await withCheckedContinuation { continuation in
3          DispatchQueue.global().async {
4              continuation.resume(returning: Int(arc4random()))
5          }
6      }
7  }

```

如果需要抛出异常，那么：



```
1 func helloAsyncThrows() async throws -> Int {
2     try await withCheckedThrowingContinuation { continuation in
3         DispatchQueue.global().async {
4             do {
5                 let result = try doSomethingThrows() // 可能抛异常
6                 continuation.resume(returning: result)
7             } catch {
8                 continuation.resume(throwing: error)
9             }
10        }
11    }
12 }
```

注意 Swift 要求对于标记为 throws 的函数需要使用 try 关键字来调用。

好了，现在我们已经学会如何将异步回调转成异步函数了，距离最终的目标又近了一步。下一篇文章当中我们将介绍如何从程序入口调用异步函数，试着把程序跑起来。

关于作者

霍丙乾 bennyhuo，Kotlin 布道师，Google 认证 Kotlin 开发专家（Kotlin GDE）；《深入理解 Kotlin 协程》作者（机械工业出版社，2020.6）；前腾讯高级工程师，现就职于猿辅导

- GitHub: <https://github.com/bennyhuo>
- 博客: <https://www.bennyhuo.com>
- bilibili: [bennyhuo不是算命的](#)
- 微信公众号: bennyhuo

相关推荐

- [闲话 Swift 协程 \(0\) : 前言](#)
- [闲话 Swift 协程 \(1\) : Swift 协程长什么样?](#)
- [闲话 Swift 协程 \(3\) : 在程序当中调用异步函数](#)
- [闲话 Swift 协程 \(4\) : TaskGroup 与结构化并发](#)



- 闲话 Swift 协程 (5) : Task 的取消

coroutines

swift


async await

< 闲话 Swift 协程 (1) : Swift 协程长什么样?

2021 总结 – bennyhuo >

0 条评论

未登录用户



说点什么




支持 Markdown 语法

使用 GitHub 登录

预览

来做第一个留言的人吧!

京ICP备16022265号-3

© 2018 — 2022  Benny Huo |  478k |  14:29

由 [Hexo](#) & [NexT.Pisces](#) 强力驱动