Algorithms in Swift 3

选择排序（Selection sort）
(https://www.boxueio.com/series/algorithms-in-swift3/ebook/86)

BST I - 初始化和插入
(https://www.boxueio.com/series/algorithms-in-swift3/ebook/87)

使用SPM构建开发环境

Back to series (/series/algorithms-in-swift3)

字号
字号
默认主题
金色主题
暗色主题

在开始后面的算法例子之前，为了模拟算法的实现和应用之间的关系，我们先来介绍下如何通过Swift Package Manager (https://github.com/apple/swift-package-manager)（以下简称SPM）来构建一个简单的开发环境。

创建一个Product

假设我们要创建一个最基本的平衡二叉树，可以执行下面这些命令：

```
mkdir BST
cd BST
swift package init --type=library
```

这样，SPM就会为我们创建下面的目录结构：

```
➜  BST tree
.
├── Package.swift
├── Sources
│   └── BST.swift
└── Tests
    ├── BSTTests
    │   └── BSTTests.swift
    └── LinuxMain.swift

3 directories, 4 files
```

在这个目录结构里，我们可以了解以下内容：

- Swift通过*Modules*来管理代码，默认情况下，所有在*Sources*目录下的文件都在同一个module中（稍后我们也会看到多个module的情况）；
- 所有*Sources*目录中的代码和根目录的*Package.swift*文件形成了一个*Package*；
- 在一个*Package*里，我们可以定义一个或多个*Target*；
- *Target*可以是我们在一开始定义的library，它可以被其他的Swift module使用；也可以是一个executable，稍后，我们会看到它的用法；

创建第一个module

在*Sources*根目录中，所有源代码默认都是在同一个module中的。我们先在 `BST.swift` 中编写一些示例代码：

```
open class BST  {
    public init() {
        print("New BST initialized.")
    }
}

extension BST: CustomStringConvertible {
    public var description: String {
        return "BST"
    }
}
```

它们当然还不是 `BST` 的正式实现，我们在这里只是为了演示module的用法。然后，我们在 `Tests/BSTTests/BSTTests.swift` 里，添加一个演示用的测试用例：

```
class BSTTests: XCTestCase {
    func testExample() {
        XCTAssertEqual(BST().description, "BST")
    }
}
```

因为我们实现的 `CustomStringConvertible` 只是简单返回了字符串"BST"，因此上面的比较应该是相等的。

完成之后，在项目根目录执行： `swift build` ，我们就生成了一个Swift module：

```
➜  BST swift build
Compile Swift Module 'BST' (1 sources)
```

执行 `swift test` ，SPM就会帮我们完成之前定义的测试用例：

```
➜  BST swift test
Test Suite 'All tests' started at 2016-10-01 22:25:00.668
Test Suite 'BSTPackageTests.xctest' started at 2016-10-01 22:25:00.669
Test Suite 'BSTTests' started at 2016-10-01 22:25:00.669
Test Case '-[BSTTests.BSTTests testExample]' started.
New BST initialized.
Test Case '-[BSTTests.BSTTests testExample]' passed (0.001 seconds).
Test Suite 'BSTTests' passed at 2016-10-01 22:25:00.670.
        Executed 1 test, with 0 failures (0 unexpected) in 0.001 (0.001) seconds
Test Suite 'BSTPackageTests.xctest' passed at 2016-10-01 22:25:00.670.
        Executed 1 test, with 0 failures (0 unexpected) in 0.001 (0.001) seconds
Test Suite 'All tests' passed at 2016-10-01 22:25:00.670.
        Executed 1 test, with 0 failures (0 unexpected) in 0.001 (0.002) seconds
```

从上面的结果可以看到，所有测试都通过了。不过，我们创建library，最终还是为了提供给应用程序使用的。因此，接下来，我们就来了解如何给package添加一个可执行程序，我们把它定义在一个新的module中。

创建多个module

默认情况下，*Sources*目录中所有代码都是在同一个module中的。因此，要创建多个module，我们要在*Sources*目录中创建多个子目录，像这样：

- `BST`： 表示BST module，并且把之前创建的 `BST.swift` 移动到这里；
- `Application`： 表示我们要新添加的应用程序；在其中，添加一个 `main.swift` ；这是每一个应用程序都必须定义的文件；

最终，我们的目录看起来是这样的：

```
➜  BST tree
.
├── Package.swift
├── Sources
│   ├── Application
│   │   └── main.swift
│   └── BST
│       └── BST.swift
└── Tests
    ├── BSTTests
    │   └── BSTTests.swift
    └── LinuxMain.swift

5 directories, 5 files
```

然后，在 `main.swift` 中，添加下面的代码：

```
import BST

let bst = BST()
print(bst)
```

重新执行 `swift build` ，我们会得到下面的错误：

```
➜  BST swift build
Compile Swift Module 'Application' (1 sources)
Compile Swift Module 'BST' (1 sources)
Linking ./.build/debug/Application
Undefined symbols for architecture x86_64:
  "__TFC3BST3BSTCfT_S0_", referenced from:
      _main in main.swift.o
  "__TMaC3BST3BST", referenced from:
      _main in main.swift.o
```

显然，尽管我们使用了 `import BST` ，SPM在生成Application module的时候，并不知道它和BST之间存在依赖关系。为了解决这个问题，我们需要在Package.swift中，添加必要的依赖关系：

```
let package = Package(
    name: "BST",
    targets: [
        Target(name: "Application", dependencies: ["BST"])
    ]
)
```

这样，我们就创建了一个叫做Application的target，它依赖我们之前创建的BST module。完成后，重新执行 `swift build`，就可以看到成功了：

```
➜  BST swift build
Compile Swift Module 'Application' (1 sources)
Linking ./.build/debug/Application
```

编译好的两个swift module在 `./build/debug` 目录中，我们直接执行 `Application` 就可以看到结果了：

```
➜  BST .build/debug/Application
New BST initialized.
BST
```

What's next?

这就是我们这一节的内容，接下来，在各种算法实现里，我们都会采用类似的方式，用library实现算法，用executable编写演示用法。当然，我们还可以在Tests中编写测试用例。在下一节中，我们就来基于SPM实现在这一节中提到的平衡二叉树。

◀ 选择排序（Selection sort）
(https://www.boxueio.com/series/algorithms-in-swift3/ebook/86)

BST I - 初始化和插入 ▶
(https://www.boxueio.com/series/algorithms-in-swift3/ebook/87)

泊 .I HD

职场漂泊的你，每天多学一点。

从开发、测试到运维，让技术不再成为你成长的绊脚石。我们用打磨产品的精神去传播知识，把最新的移动开发技术，通过简单的图表， 清晰的视频，简明的文字和切实可行的例子一一向你呈现。让学习不仅是一种需求，也是一种享受。

泊学动态

一个工作十年PM终创业的故事（二） (https://www.boxueio.com/after-the-full-upgrade-to-swift3)
Mar 4, 2017

人生中第一次创业的"10有" (https://www.boxueio.com/founder-chat)
Jan 9, 2016

猎云网采访报道泊学 (http://www.lieyunwang.com/archives/144329)
Dec 31, 2015

What most schools do not teach (https://www.boxueio.com/what-most-schools-do-not-teach)
Dec 21, 2015

一个工作十年PM终创业的故事（一） (https://www.boxueio.com/founder-story)
May 8, 2015

泊学相关

关于泊学 ›

加入泊学 ›

泊学用户隐私以及服务条款 (HTTPS://WWW.BOXUEIO.COM/TERMS-OF-SERVICE)

版权声明 (HTTPS://WWW.BOXUEIO.COM/COPYRIGHT-STATEMENT)