苹果在发布了Xcode 6 Bate 4后为Swift添加了新的特性--访问控制（Access Control），并且更新了The Swift Programming Language文档，我抽空把这篇文档翻译了一下，下面让我们来详细了解一下Access Control。

**本页内容包括：**

**1.模块和源文件**

**2.访问级别**

**（1）访问级别的使用原则**

**（2）默认访问级别**

**（3）单目标应用程序的访问级别**

**（4）Framework的访问级别**

**3.访问控制语法**

**4.自定义类型**

**（1）元组类型**

**（2）函数类型**

**（3）枚举类型**

**（4）原始值和关联值**

**（5）嵌套类型**

**5.子类**

**6.常量、变量、属性、下标**

**（1）Getter和Setter**

**7.初始化**

**（1）默认初始化方法**

**（2）结构体的默认成员初始化方法**

**8.协议**

**（1）协议继承**

**（2）协议一致性**

**9.扩展**

**（1）协议的扩展**

**10.泛型**

**11.类型别名**

访问控制可以限定你在源文件或模块中访问代码的级别，也就是说可以控制哪些代码你可以访问，哪些代码你不能访问。这个特性可以让我们隐藏功能实现的一些细节，并且可以明确的指定我们提供给其他人的接口中哪些部分是他们可以使用的，哪些是他们看不到的。

你可以明确的给类、结构体、枚举、设置访问级别，也可以给属性、函数、初始化方法、基本类型、下标索引等设置访问级别。协议也可以被限定在一定的范围内使用，包括协议里的全局常量、变量和函数。

在提供了不同访问级别的同时，Swift 并没有规定我们要在任何时候都要在代码中明确指定访问级别。其实，如果我们作为独立开发者在开发我们自己的 app，而不是在开发一些 Framework 的时候，我们完全可以不用明确的指定代码的访问级别。

注意：为方便起见，在代码中可以设置访问级别的它们（属性、基本类型、函数等）在下面的章节中我们称之为“实体”。

**模块和源文件**

Swift 中的访问控制模型基于模块和源文件这两个概念。

模块指的是 Framework 或 App bundle 。在 Swift 中，可以用 import 关键字引入自己的工程。

在 Swift 中，Framework 或 App bundle 被作为模块处理。如果你是为了实现某个通用的功能，或者是为了封装一些常用方法而将代码打包成 Framework，这个 Framework  在 Swift 中就被称为模块。不论它被引入到某个 App 工程或者其他的Framework，它里面的一切（属性、函数等）都属于这个模块。

源文件指的是 Swift 中的Swift File，就是编写 Swift 代码的文件，它通常属于一个模块。通常一个源文件包含一个类，在类中又包含函数、属性等类型。

**访问级别**

Swift 提供了三种不同的访问级别。这些访问级别相对于源文件中定义的实体，同时也相对于这些源文件所属的模块。

**1.Public：**可以访问自己模块或应用中源文件里的任何实体，别人也可以访问引入该模块中源文件里的所有实体。通常情况下，某个接口或 Framework 是可以被任何人使用时，你可以将其设置为 public 级别。

**2.Internal：**可以访问自己模块或应用中源文件里的任何实体，但是别人不能访问该模块中源文件里的实体。通常情况下，某个接口或 Framework 作为内部结构使用时，你可以将其设置为 internal 级别。

**3.Private：**只能在当前源文件中使用的实体，称为私有实体。使用 private 级别，可以用作隐藏某些功能的实现细节。

Public 为最高级访问级别，Private 为最低级访问级别。

**访问级别的使用原则**

在 Swift 中，访问级别有如下使用原则：访问级别统一性。 比如说：

1.一个 public 访问级别的变量，不能将它的类型定义为 internal 和 private 的类型。因为变量可以被任何人访问，但是定义它的类型不可以，所以这样就会出现错误。

2.函数的访问级别不能高于它的参数、返回类型的访问级别。因为如果函数定义为 public 而参数或者返回类型定义为 internal 或 private，就会出现函数可以被任何人访问，但是它的参数和返回类型不可以，同样会出现错误。

**默认访问级别**

代码中的所有实体，如果你不明确的定义其访问级别，那么它们默认为 internal 级别。在大多数情况下，我们不需要明确的设置实体的访问级别，因为我们大多数时候都是在开发一个 App bundle。

**单目标应用程序的访问级别**

当你编写一个单目标应用程序时，该应用的所有功能都是为该应用服务，不需要提供给其他应用或者模块使用，所以我们不需要明确设置访问级别，使用默认的访问级别 internal 即可。但是如果你愿意，你也可以使用 private 级别，用于隐藏一些功能的实现细节。

**Framework的访问级别**

当你开发 Framework 时，就需要把一些实体定义为 public 级别，以便其他人导入该 Framework 后可以正常使用其功能。这些被你定义为 public 的实体，就是这个 Framework 的API。

**注意：**Framework 的内部实现细节依然可以使用默认的 internal 级别，或者也可以定义为 private 级别。只有你想将它作为 API 的实体，才将其定义为 public 级别。

**访问控制语法**

通过修饰符 public、internal、private 来声明实体的访问级别：

1. **public** **class** SomePublicClass {}
2. internal **class** SomeInternalClass {}
3. **private** **class** SomePrivateClass {}
5. **public** **var** somePublicVariable = 0
6. internal let someInternalConstant = 0
7. **private** func somePrivateFunction() {}

除非有特殊的说明，否则实体都使用默认的访问级别 internal，可以查阅 [Default Access Levels](https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/AccessControl.html#//apple_ref/doc/uid/TP40014097-CH41-XID_34) 这一节。这意味着 SomeInternalClass 和 someInternalConstant 不用明确的使用修饰符声明访问级别，但是他们任然拥有隐式的访问级别 internal：

1. **class** SomeInternalClass {}              // 隐式访问级别 internal
2. **var** someInternalConstant = 0            // 隐式访问级别 internal

**自定义类型**

如果你想为一个自定义类型指定一个明确的访问级别，那么你要明确一点。那就是你要确保新类型的访问级别和它实际的作用域相匹配。比如说，如果某个类里的属性、函数、返回值它们的作用域仅在当前的源文件中，那么你就可以将这个类声明为 private 类，而不需要声明为 public 或者 internal 类。

类的访问级别也可以影响到类成员（属性、函数、初始化方法等）的默认访问级别。如果你将类声明为 private 类，那么该类的所有成员的默认访问级别也会成为 private 。如果你将类声明为 public 或者 internal 类（或者不明确的指定访问级别，而使用默认的 internal 访问级别），那么该类的所有成员的访问级别是 internal 。

**注意：**上面提到，一个 public 类的所有成员的访问级别默认为 internal 级别，而不是 public 级别。如果你想将某个成员声明为 public 级别，那么你必须使用修饰符明确的声明该成员。这样做的好处是，在你定义公共接口API的时候，可以明确的选择哪些属性或方法是需要公开的，哪些是内部使用的，可以避免将内部使用的属性方法公开成公共API的错误。

1. **public** **class** SomePublicClass {          // 显示的 public 类
2. **public** **var** somePublicProperty = 0    // 显示的 public 类成员
3. **var** someInternalProperty = 0         // 隐式的 internal 类成员
4. **private** func somePrivateMethod() {}  // 显示的 private 类成员
5. }
7. **class** SomeInternalClass {               // 隐式的 internal 类
8. **var** someInternalProperty = 0         // 隐式的 internal 类成员
9. **private** func somePrivateMethod() {}  // 显示的 private 类成员
10. }
12. **private** **class** SomePrivateClass {        // 显示的 private 类
13. **var** somePrivateProperty = 0          // 隐式的 private 类成员
14. func somePrivateMethod() {}          // 隐式的 private 类成员
15. }

**元组类型**

元组的访问级别使用是所有类型的访问级别使用中最为严谨的。比如说，如果你构建一个包含两种不同类型元素的元组，其中一个元素类型的访问级别为 internal，另一个为 private 级别，那么这个元组的访问级别为private 。也就是说元组的访问级别遵循它里面元组中最低级的访问级别。

**注意：**元组不同于类、结构体、枚举、函数那样有单独的定义。元组的访问级别是在它被使用时自动推导出的，而不是明确的声明。

**函数类型**

函数的访问级别需要根据该函数的参数类型访问级别、返回类型访问级别得出。如果根据参数类型和返回类型得出的函数访问级别不符合上下文，那么就需要明确的声明该函数的访问级别。

下面的例子中定义了一个全局函数名为 someFunction ，并且没有明确的声明其访问级别。你也许会认为该函数应该拥有默认的访问级别 internal，但事实并非如此。事实上，如果按下面这种写法，编译器是无法编译通过的：

1. func someFunction() -> (SomeInternalClass, SomePrivateClass) {
2. // function implementation goes here
3. }

我们可以看到，这个函数的返回类型是一个元组，该元组中包含两个自定义的类（可查阅[Custom Types](https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/AccessControl.html#//apple_ref/doc/uid/TP40014097-CH41-XID_39)）。其中一个类的访问级别是 internal，另一个的访问级别是 private，所以根据元组访问级别的原则，该元组的访问级别是 private（元组的访问级别遵循它里面元组中最低级的访问级别）。

因为该函数返回类型的访问级别是private，所以你必须使用private修饰符，明确地申请该函数：

1. **private** func someFunction() -> (SomeInternalClass, SomePrivateClass) {
2. // function implementation goes here
3. }

将该函数声明为public或internal，或者使用默认的访问级别internal都是错误的，因为如果把该函数当做 public 或 internal 级别来使用的话，是无法得到private级别的返回值的。

**枚举类型**

枚举中成员的访问级别继承自该枚举，你不能为枚举中的成员指定访问级别。

比如下面的例子，枚举 CompassPoint 被明确的声明为 public 级别，那么它的成员 North，South，East，West 的访问级别同样也是 public：

1. **public** **enum** CompassPoint {
2. **case** North
3. **case** South
4. **case** East
5. **case** West
6. }

**原始值和关联值**

用于枚举定义中的任何原始值，或关联的值类型必须有一个访问级别，至少要高于枚举的访问级别。比如说，你不能在一个 internal 访问级别的枚举中定义 private 级别的原始值类型。

**嵌套类型**

如果在private级别的类型中定义嵌套类型，那么该嵌套类型就自动拥有 private 访问级别。如果在 public 或者 internal 级别的类型中定义嵌套类型，那么该嵌套类型自动拥有 internal 访问级别。如果想让嵌套类型拥有 public 访问级别，那么需要对该嵌套类型进行明确的访问级别声明。

**子类**

子类的访问级别不得高于父类的访问级别。比如说，父类的访问级别是 internal ，子类的访问级别就不能声明为 public 。

此外，在满足子类不高于父类访问级别以及遵循各访问级别作用域（即模块或源文件）的前提下，你可以重写任意类成员（方法、属性、初始化方法、下标索引等）。

如果我们无法直接访问某个类中的属性或函数等，那么可以继承该类，从而可以更容易的访问到该类的类成员。下面的例子中，类A的访问级别是 public ，它包含一个函数 someMethod，访问级别为private。类B继承类A，并且访问级别声明为 internal ，但是在类B中重写了类A中访问级别为 private 的方法 someMethod，并重新声明为 internal 级别。通过这种方式，我们就可以访问到某类中 private 级别的类成员，并且可以重新声明其访问级别，以便其他人使用：

1. **public** **class** A {
2. **private** func someMethod() {}
3. }
5. internal **class** B: A {
6. override internal func someMethod() {}
7. }

只要满足子类不高于父类访问级别以及遵循各访问级别作用域的前提下（即 private 的作用域在同一个源文件中，internal 的作用域在同一个模块下），我们甚至可以在子类中，用子类成员访问父类成员，哪怕父类成员的访问级别比子类成员的要低：

1. **public** **class** A {
2. **private** func someMethod() {}
3. }
5. internal **class** B: A {
6. override internal func someMethod() {
7. **super**.someMethod()
8. }
9. }

因为父类A和子类B定义在同一个源文件中，所以在类B中可以在重写的 someMethod 方法中调用 super.someMethod() 。

**常量、变量、属性、下标**

常量、变量、属性不能拥有比它们的类型更高的访问级别。比如说，你定义一个 public 级别的属性，但是它的类型是 private 级别的，这是编译器不允许的。同样，下标也不能拥有比索引类型或返回类型更高的访问级别。

如果常量、变量、属性、下标索引的定义类型是 private 级别的，那么它们必须要明确的声明访问级别为 private：

1. **private** **var** privateInstance = SomePrivateClass()

**Getter和Setter**

常量、变量、属性、下标索引的 Getters 和 Setters 的访问级别继承自它们所属成员的访问级别。

Setter 的访问级别可以低于对应的 Getter 的访问级别，这样就可以控制变量、属性或下标索引的读写权限。在var或subscript定义作用域之前，你可以通过private(set)或internal(set)先为它门的写权限声明一个较低的访问级别。

**注意：**这个规定适用于用作存储的属性或用作计算的属性。即使你不明确的声明存储属性的 Getter 和 Setter，Swift 也会隐式的为其创建 Getter 和 Setter，用于对该属性进行读取操作。使用 private(set) 和internal(set) 可以改变 Swift 隐式创建的 Setter 的访问级别。在计算属性中也是同样的。

下面的例子中定义了一个结构体名为 TrackedString，它记录了value属性被修改的次数：

1. struct TrackedString {
2. **private**(set) **var** numberOfEdits = 0
3. **var** value: String = "" {
4. didSet {
5. numberOfEdits++
6. }
7. }
8. }

TrackedString 结构体定义了一个用于存储的属性名为 value，类型为 String，并将初始化值设为""（即一个空字符串）。该结构体同时也定义了另一个用于存储的属性名为 numberOfEdits ，类型为 Int，它用于记录属性 value 被修改的次数。这个功能的实现通过属性 value 的 didSet 方法实现，每当给value赋新值时就会调用 didSet 方法，给 numberOfEdits 加一。

结构体 TrackedString 和它的属性 value 均没有明确的声明访问级别，所以它们都拥有默认的访问级别 internal。但是该结构体的 numberOfEdits 属性使用 private(set) 修饰符进行声明，这意味着numberOfEdits 属性只能在定义该结构体的源文件中赋值。numberOfEdits 属性的 Getter 依然是默认的访问级别 internal，但是 Setter 的访问级别是 private ，这表示该属性只有在当前的源文件中是可读可写的，在当前源文件所属的模块中它只是一个可读的属性。

如果你实例化 TrackedString 结构体，并且多次对value属性的值进行修改，你就会看到 numberOfEdits 的值会随着修改次数更改：

1. **var** stringToEdit = TrackedString()
2. stringToEdit.value = "This string will be tracked."
3. stringToEdit.value += " This edit will increment numberOfEdits."
4. stringToEdit.value += " So will this one."
5. println("The number of edits is \(stringToEdit.numberOfEdits)")
6. // prints "The number of edits is 3"

虽然你可以在其他的源文件中实例化该结构体并且获取到numberOfEdits属性的值，但是你不能对其进行赋值。这样就能很好的告诉使用者，你只管使用，而不需要知道其实现细节。

**初始化**

我们可以给自定义的初始化方法指定访问级别，但是必须要低于或等于它所属类的访问级别。但如果该初始化方法是必须要使用的话，那它的访问级别就必须和所属类的访问级别相同（参见[Required Initializers](https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Initialization.html#//apple_ref/doc/uid/TP40014097-CH18-XID_336)）。

如同函数或方法参数，初始化方法参数的访问级别也不能低于初始化方法的访问级别。

**默认初始化方法**

Swift 为结构体、类都提供了一个默认的无参初始化方法，用于给它们的所有属性提供赋值操作，但不会给出具体值。默认初始化方法可以参阅 [Default Initializers](https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Initialization.html#//apple_ref/doc/uid/TP40014097-CH18-XID_318) 。默认初始化方法的访问级别与所属类型的访问级别相同。

**注意：**如果一个类型被声明为public级别，那么默认的初始化方法的访问级别为 internal 。如果你想让无参的初始化方法在其他模块中可以被使用，那么你必须提供一个具有 public 访问级别的无参初始化方法。

**结构体的默认成员初始化方法**

如果结构体中的任一存储属性的访问级别为 private ，那么它的默认成员初始化方法访问级别就是 private 。尽管如此，结构体的初始化方法的访问级别依然是 internal 。

如果你想在其他模块中使用该结构体的默认成员初始化方法，那么你需要提供一个访问级别为 public 的默认成员初始化方法。

**协议**

如果你想为一个协议明确的声明访问级别，那么有一点需要注意，就是你要确保该协议只在你声明的访问级别作用域中使用。

协议中的每一个必须要实现的函数都具有和该协议相同的访问级别。这样才能确保该协议的使用者可以实现它所提供的函数。

**注意：**如果你定义了一个 public 访问级别的协议，那么实现该协议提供的必要函数也会是public的访问级别。这一点不同于其他类型，比如 public 访问级别的其他类型，他们成员的访问级别为 internal 。

**协议继承**

如果定义了一个新的协议，并且该协议继承了一个已知的协议，那么新协议拥有的访问级别最高也只和被继承协议的访问级别相同。比如说，你不能定义一个public的协议而去继承一个internal的协议。

**协议一致性**

类可以采用比自身访问级别低的协议。比如说，你可以定义一个 public 级别的类，可以让它在其他模块中使用，同时它也可以采用一个 internal 级别的协议，并且只能在定义了该协议的模块中使用。

采用了协议的类的访问级别遵循它本身和采用协议中最低的访问级别。也就是说如果一个类是 public 级别，采用的协议是 internal 级别，那个采用了这个协议后，该类的访问级别也是 internal 。

如果你采用了协议，那么实现了协议必须的方法后，该方法的访问级别遵循协议的访问级别。比如说，一个 public 级别的类，采用了 internal 级别的协议，那么该类实现协议的方法至少也得是 internal 。

**注意：**在 Swift 中和 Objective-C 中一样，协议的一致性保证了一个类不可能在同一个程序中用不同的方法采用同一个协议。

**扩展**

你可以在条件允许的情况下对类、结构体、枚举进行扩展。扩展成员应该具有和原始类成员一致的访问级别。比如你扩展了一个公共类型，那么你新加的成员应该具有和原始成员一样的默认的 internal 访问级别。

或者，你可以明确声明扩展的访问级别（比如使用 private extension）给该扩展内所有成员指定一个新的默认访问级别。这个新的默认访问级别仍然可以被单独成员所指定的访问级别所覆盖。

**协议的扩展**

如果一个扩展采用了某个协议，那么你就不能对该扩展使用访问级别修饰符来声明了。该扩展中实现协议的方法都会遵循该协议的访问级别。

**泛型**

泛型类型或泛型函数的访问级别遵循泛型类型、函数本身、泛型类型参数三者中访问级别最低的级别。

**类型别名**

任何被你定义的类型别名都会被视作为不同的类型，这些类型用于访问控制。一个类型别名的访问级别可以低于或等于这个类型的访问级别。比如说，一个private级别的类型别名可以设定给一个public、internal、private的类型，但是一个public级别的类型别名只能设定给一个public级别的类型，不能设定给internal或private的类类型。

**注意：**这条规则也适用于为满足协议一致性而给相关类型命名别名。