访问控制

在本页

*访问控制*限制从其他源文件和模块中的代码访问部分代码。此功能使您可以隐藏代码的实现细节，并指定可以访问和使用该代码的首选接口。

您可以为单个类型（类，结构和枚举）以及属于这些类型的属性，方法，初始化程序和下标分配特定的访问级别。协议可以限制在某个上下文中，全局常量，变量和函数也可以。

除了提供各种级别的访问控制之外，Swift通过为典型场景提供默认访问级别，减少了指定显式访问控制级别的需求。事实上，如果您正在编写单目标应用程序，则可能根本不需要指定明确的访问控制级别。

> 注意
>
> 为简洁起见，您的代码的各个方面（可以具有应用于它们的访问控制（属性，类型，函数等））在下面的部分中称为“实体”。

模块和源文件

Swift的访问控制模型基于模块和源文件的概念。

一个*模块*是一个代码分发单元 - 一个框架或应用程序，它是作为一个单元构建和交付的，并且可以由另一个模块通过Swift的`import`关键字导入。

Xcode中的每个构建目标（如应用程序包或框架）都被视为Swift中的独立模块。如果将应用程序代码的各个方面作为独立框架组合在一起（也许是将这些代码封装并重复使用到多个应用程序中），那么您在该框架中定义的所有内容都将在应用程序中导入和使用时成为单独模块的一部分，或者当它在另一个框架内使用时。

甲*源文件*是一个模块内的单个夫特源代码文件（实际上，一个应用程序或框架内的一个单独的文件）。尽管在单独的源文件中定义单个类型是很常见的，但单个源文件可以包含多种类型，函数等的定义。

访问级别

Swift为代码中的实体提供了五种不同的*访问级别*。这些访问级别与实体定义的源文件相关，也与源文件所属的模块有关。

- *开放访问*和*公共访问*使实体可以在来自其定义模块的任何源文件中使用，并且也可以在来自导入定义模块的另一个模块的源文件中使用。在为框架指定公共接口时，通常使用开放或公共访问。下面介绍开放和公共访问之间的区别。
- *内部访问*使实体可以在其定义模块的任何源文件中使用，但不能在该模块之外的任何源文件中使用。定义应用程序或框架的内部结构时，通常使用内部访问。
- *文件私有访问*限制了实体对其自己定义的源文件的使用。在整个文件中使用这些详细信息时，使用文件私有访问来隐藏特定功能部分的实现细节。
- *私人访问*限制了实体对封闭声明的使用，并限制了该声明在同一文件中的扩展。当这些细节仅在单个声明中使用时，使用私有访问来隐藏特定功能的实现细节。

开放访问是最高（限制最少）的访问级别，私人访问是最低（限制最多）的访问级别。

开放访问只适用于类和类成员，它与公共访问的区别如下：

- 具有公共访问权限的类别或任何更具限制性的访问级别只能在其定义的模块中进行分类。
- 具有公共访问权限或具有更多限制性访问级别的类成员只能由其定义的模块内的子类覆盖。
- 打开的类可以在定义它们的模块内以及在任何导入定义模块的模块中进行子类化。
- 开放类成员可以被它们定义的模块中的子类覆盖，也可以在导入定义模块的任何模块中覆盖。

将一个类标记为open明确指出你已经考虑了来自其他模块的代码作为超类的影响，并且已经相应地设计了你的类的代码。

访问级别的指导原则

Swift中的访问级别遵循总体指导原则：*没有实体可以用另一个具有较低（更具限制性）访问级别的实体来定义。*

例如：

- 公共变量不能被定义为具有内部，文件私有或私有类型，因为该类型在使用公共变量的任何地方都可能不可用。
- 函数不能具有比其参数类型和返回类型更高的访问级别，因为该函数可用于其构成类型对于周围代码不可用的情况。

下面详细介绍这一指导原则对语言不同方面的具体含义。

默认访问级别

如果您没有自己指定明确的访问级别，那么代码中的所有实体（具有一些特定的例外，如本章后面所述）都具有默认的内部访问级别。因此，在许多情况下，您不需要在代码中指定明确的访问级别。

单目标应用程序的访问级别

当您编写简单的单目标应用程序时，应用程序中的代码通常是自包含在应用程序中，并且不需要在应用程序模块之外提供。内部的默认访问级别已经符合此要求。因此，您不需要指定自定义访问级别。但是，您可能希望将代码的某些部分标记为私有文件或私有文件，以便将应用程序模块中的其他代码的实现细节隐藏起来。

框架访问级别

在开发框架时，将面向公众的界面标记为open或public，以便可以被其他模块（例如导入框架的应用程序）查看和访问。这个面向公众的接口是框架的应用程序编程接口（或API）。

> 注意
>
> 你的框架的任何内部实现细节仍然可以使用默认的内部访问级别，或者如果你想隐藏框架内部代码的其他部分，可以将其标记为私有或私有文件。只有当您希望它成为框架API的一部分时，您才需要将实体标记为公开或公开。

单元测试目标的访问级别

当您使用单元测试目标编写应用程序时，应用程序中的代码需要提供给该模块才能进行测试。默认情况下，只有标记为open或public的实体可以被其他模块访问。但是，如果您标记具有该`@testable`属性的产品模块的导入声明并且启用了测试编译该产品模块，则单元测试目标可以访问任何内部实体。

访问控制语法

通过放置的一个定义实体的访问级别`open`，`public`，`internal`，`fileprivate`，或`private`实体的导入前修饰语：

```
public class SomePublicClass {}
internal class SomeInternalClass {}
fileprivate class SomeFilePrivateClass {}
private class SomePrivateClass {}

public var somePublicVariable = 0
internal let someInternalConstant = 0
fileprivate func someFilePrivateFunction() {}
private func somePrivateFunction() {}
```

除非另有说明，否则默认访问级别是内部的，如默认访问级别中所述。这意味着`SomeInternalClass`并且`someInternalConstant`可以在没有显式访问级别修饰符的情况下编写，并且仍然具有内部访问级别：

```
class SomeInternalClass {}             // implicitly internal
```

```
let someInternalConstant = 0            // implicitly internal
```

自定义类型

如果要为自定义类型指定显式访问级别，请在定义类型的位置执行此操作。然后可以在访问级别允许的地方使用新类型。例如，如果您定义了一个文件私有类，则该类只能用作定义文件私有类的源文件中的属性类型，或者用作函数参数或返回类型。

类型的访问控制级别还会影响该类型*成员*（其属性，方法，初始化程序和下标）的默认访问级别。如果您将类型的访问级别定义为私有或私有文件，则其成员的默认访问级别也将为私有或私有文件。如果将类型的访问级别定义为内部或公共级别（或者使用内部的默认访问级别而不明确指定访问级别），则类型成员的默认访问级别将为内部级别。

> 重要
>
> 公共类型默认为拥有内部成员，而不是公共成员。如果你想让一个类型成员公开，你必须明确地标记它。此要求确保面向公众的API是您选择发布的内容，并避免错误地将类型的内部工作作为公共API呈现。

```
public class SomePublicClass {                  // explicitly public class
    public var somePublicProperty = 0            // explicitly public class member
    var someInternalProperty = 0                 // implicitly internal class
  member
    fileprivate func someFilePrivateMethod() {}  // explicitly file-private class
  member
    private func somePrivateMethod() {}          // explicitly private class member
}

class SomeInternalClass {                       // implicitly internal class
    var someInternalProperty = 0                 // implicitly internal class
  member
    fileprivate func someFilePrivateMethod() {}  // explicitly file-private class
  member
    private func somePrivateMethod() {}          // explicitly private class member
}

fileprivate class SomeFilePrivateClass {        // explicitly file-private class
    func someFilePrivateMethod() {}              // implicitly file-private class
  member
    private func somePrivateMethod() {}          // explicitly private class member
}

private class SomePrivateClass {                // explicitly private class
    func somePrivateMethod() {}                  // implicitly private class member
}
```

元组类型

元组类型的访问级别是该元组中使用的所有类型的限制最多的访问级别。例如，如果您从两种不同的类型组成一个元组，其中一个具有内部访问权限，另一个具有私有访问权限，则该复合元组类型的访问级别将是私有的。

> 注意
>
> 元组类型没有像类，结构，枚举和函数那样的独立定义。元组类型的访问级别在使用元组类型时自动推导出来，并且不能明确指定。

函数类型

函数类型的访问级别计算为函数参数类型和返回类型的限制最多的访问级别。如果函数的计算访问级别与上下文默认值不匹配，则必须明确指定访问级别作为函数定义的一部分。

下面的例子定义了一个全局函数`someFunction()`，没有为函数本身提供一个特定的访问级修饰符。您可能会希望此功能具有“内

```
func someFunction() -> (SomeInternalClass, SomePrivateClass) {
    // function implementation goes here
}
```

该函数的返回类型是一个元组类型，由上面自定义类型中定义的两个自定义类组成。其中一个类被定义为内部类，另一个类被定义为私有类。因此，复合元组类型的整体访问级别是私有的（元组构成类型的最小访问级别）。

因为函数的返回类型是私有的，所以必须使用`private`修饰符标记函数的整体访问级别，以使函数声明有效：

```
private func someFunction() -> (SomeInternalClass, SomePrivateClass) {
    // function implementation goes here
}
```

`someFunction()`使用`public`或`internal`修饰符 的定义标记或使用内部的默认设置是无效的，因为函数的公共或内部用户可能没有适当的访问函数返回类型中使用的私有类的权限。

枚举类型

枚举的各种情况自动获得与它们所属的枚举相同的访问级别。您无法为单个枚举案例指定不同的访问级别。

在下面的例子中，`CompassPoint`枚举具有明确的公共访问级别。枚举的情况下`north`，`south`，`east`，和`west`因此也有公共的访问级别：

```
public enum CompassPoint {
    case north
    case south
    case east
    case west
}
```

原始价值和相关价值

枚举定义中用于任何原始值或关联值的类型必须具有至少与枚举的访问级别一样高的访问级别。例如，您不能将私有类型用作具有内部访问级别的枚举的原始值类型。

嵌套类型

在私有类型中定义的嵌套类型具有私有的自动访问级别。在文件 - 私有类型中定义的嵌套类型具有文件私有的自动访问级别。在公共类型或内部类型中定义的嵌套类型具有内部自动访问级别。如果您希望公共类型中的嵌套类型公开可用，那么您必须显式声明嵌套类型为`public`。

子类

您可以继承任何可在当前访问上下文中访问的类。子类的访问级别不能超过其超类 - 例如，不能写出内部超类的公共子类。

另外，您可以重写在某个访问上下文中可见的任何类成员（方法，属性，初始值设定项或下标）。

覆盖可以使继承的类成员比其超类版更易于访问。在下面的例子中，class `A`是一个带有文件私有方法的公共类`someMethod()`。类`B`是`A`“内部”访问级别降低的子类。尽管如此，类`B`提供了`someMethod()`“内部”访问级别的覆盖，该级别*高于*以下原始实现`someMethod()`：

```
public class A {
    fileprivate func someMethod() {}
}
```

```
4
5   internal class B: A {
6       override internal func someMethod() {}
7   }
```

对于子类成员来说，只要对超类成员的调用发生在允许的访问级别上下文中（即，在与源类型相同的源文件中），就可以调用具有比子类成员更低的访问权限的超类成员，文件私有成员调用的超类，或与内部成员调用的超类相同的模块中）：

```
1   public class A {
2       fileprivate func someMethod() {}
3   }
4
5   internal class B: A {
6       override internal func someMethod() {
7           super.someMethod()
8       }
9   }
```

因为超类A和子类B是在同一个源文件中定义的，所以它对于B实现someMethod()调用是有效的super.someMethod()。

常量，变量，属性和下标

一个常量，变量或属性不能比它的类型更公开。例如，编写具有私有类型的公共属性是无效的。同样，下标不能比其索引类型或返回类型更公开。

如果常量，变量，属性或下标使用私有类型，则常量，变量，属性或下标也必须标记为private：

```
private var privateInstance = SomePrivateClass()
```

吸气剂和固化剂

常量，变量，属性和下标的访问器和设置器会自动获得与它们所属的常量，变量，属性或下标相同的访问级别。

您可以为setter设置比其相应getter *更低的*访问级别，以限制该变量，属性或下标的读写范围。通过编写分配较低访问级别fileprivate(set)，private(set)或internal(set)前var或subscript引导。

> 注意
>
> 此规则适用于存储的属性以及计算的属性。即使你没有为存储属性写一个明确的getter和setter，Swift仍然会为你提供一个隐含的getter和setter来提供对存储属性的后备存储的访问。使用fileprivate(set)，private(set)和internal(set)更改此合成设置器的访问级别，与计算属性中的显式设置器完全相同。

下面的例子定义了一个叫做的结构TrackedString，它跟踪一个字符串属性被修改的次数：

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

该TrackedString结构定义了一个名为的存储字符串属性value，初始值为""（空字符串）。该结构还定义了一个存储的整数属性numberOfEdits，用于跟踪value修改的次数。这个修改跟踪与执行didSet的财产观察者value属性，增加numberOfEdits每次的时间value属性被设置为一个新值。

在TrackedString结构和value性能不提供明确的访问级别的修正，所以他们都收到内部默认的访问级别。但是，该numberOfEdits属性的访问级别用一个private(set)修饰符标记，以指示该属性的getter仍具有内部的默认访问级别，但该属性仅可在TrackedString结构中的代码内部进行设置。这使得TrackedString可以在numberOfEdits[illegible]

如果您创建TrackedString实例并修改其字符串值几次，您可以看到numberOfEdits属性值更新以匹配修改的数量：

```
var stringToEdit = TrackedString()
stringToEdit.value = "This string will be tracked."
stringToEdit.value += " This edit will increment numberOfEdits."
stringToEdit.value += " So will this one."
print("The number of edits is \(stringToEdit.numberOfEdits)")
// Prints "The number of edits is 3"
```

尽管您可以numberOfEdits从另一个源文件中查询该属性的当前值，但不能从其他源文件*修改*该属性。此限制保护TrackedString编辑跟踪功能的实现细节，同时仍然提供对该功能方面的方便访问。

请注意，如果需要，您可以为getter和setter分配明确的访问级别。下面的例子显示了一个TrackedString结构版本，其中结构被定义为明确的公开访问级别。因此，结构的成员（包括numberOfEdits属性）默认具有内部访问级别。您可以将结构的numberOfEdits属性getter公众，它的属性setter私人，通过合并public和private(set)访问级别修饰符：

```
public struct TrackedString {
    public private(set) var numberOfEdits = 0
    public var value: String = "" {
        didSet {
            numberOfEdits += 1
        }
    }
    public init() {}
}
```

初始化器

可以为自定义初始化程序分配一个小于或等于它们初始化的类型的访问级别。唯一的例外是所需的初始化器（如必需的初始化器中定义的）。必需的初始化程序必须具有与其所属的类相同的访问级别。

与函数和方法参数一样，初始值设定项参数的类型不能比初始值设定项自身的访问级别更私有。

默认初始化程序

如在描述的默认初始值设定，自动斯威夫特提供了一个*默认初始值*而无需任何结构或基类，对于其所有属性提供缺省值，并且不提供至少一个初始值设定本身的任何参数。

默认初始化程序与它初始化的类型具有相同的访问级别，除非该类型被定义为public。对于被定义为的类型，public默认初始化程序被认为是内部的。如果您希望公共类型在使用另一个模块时使用无参数初始值设定项进行初始化，则您必须明确提供一个公共无参数初始值设定项，作为该类型定义的一部分。

结构类型的默认成员初始化器

如果结构的任何存储属性都是私有的，则结构类型的默认成员初始值设定项会被视为私有。同样，如果任何结构的存储属性都是文件私有的，则初始化程序是文件私有的。否则，初始化程序具有内部访问级别。

与上面的默认初始化程序一样，如果您希望在另一个模块中使用公共结构类型时可以使用成员初始化程序进行初始化，您必须自己提供一个公共成员初始化程序，作为类型定义的一部分。

协议

如果要为协议类型指定明确的访问级别，请在定义协议的位置执行此操作。这使您可以创建只能在特定访问上下文中采用的协议。

协议定义中每个需求的访问级别会自动设置为与协议相同的访问级别。您不能将协议要求设置为与其支持的协议不同的访问级别。这确保了所有协议的要求都可以在任何采用该协议的类型上看到。

> 注意
>
> 如果您定义了一个公共协议，那么协议的要求在这些要求实施时需要公共访问级别。这种行为不同于其他类型，其中公共类型定义意味着类型成员的内部访问级别。

协议继承

如果您定义了一个从现有协议继承的新协议，那么新协议最多可以拥有与其从中继承的协议相同的访问级别。例如，您不能编写从内部协议继承的公共协议。

协议一致性

一种类型可以符合比类型本身具有更低访问级别的协议。例如，您可以定义可用于其他模块的公共类型，但其内部协议的一致性只能在内部协议的定义模块中使用。

类型符合特定协议的上下文是类型访问级别和协议访问级别的最小值。如果一个类型是公共的，但它符合的协议是内部的，那么类型与协议的一致性也是内部的。

当您编写或扩展某个类型以符合协议时，您必须确保每个协议要求的类型实现至少具有与该协议类型一致的访问级别。例如，如果公共类型符合内部协议，则每个协议要求的类型实现必须至少为“内部”。

> 注意
>
> 在Swift中，和Objective-C一样，协议一致性是全局的 - 在同一个程序中，类型不可能以两种不同的方式符合协议。

扩展

您可以在类，结构或枚举可用的任何访问上下文中扩展类，结构或枚举。在扩展中添加的任何类型成员与在扩展的原始类型中声明的类型成员具有相同的默认访问级别。如果您扩展公共或内部类型，则添加的任何新类型成员都具有内部的默认访问级别。如果扩展文件私有类型，则添加的任何新类型成员都具有私有文件的默认访问级别。如果您扩展私有类型，则您添加的任何新类型成员都具有私有默认访问级别。

或者，您可以使用明确的访问级别修饰符（例如，`private extension`）为扩展名标记扩展名，以便为扩展名中定义的所有成员设置新的默认访问级别。这个新的默认值仍然可以在个别类型成员的扩展名中被覆盖。

如果您使用该扩展来添加协议一致性，则不能为扩展提供明确的访问级别修饰符。相反，协议自己的访问级别用于为扩展中的每个协议需求实现提供默认访问级别。

私人会员在扩展

与它们扩展的类，结构或枚举位于同一个文件中的扩展的行为就好像扩展中的代码已被写入原始类型声明的一部分一样。因此，您可以：

- 在原始声明中声明私有成员，并从同一文件的扩展中访问该成员。
- 在一个扩展中声明私有成员，并从同一文件中的另一个扩展访问该成员。
- 在扩展中声明一个私有成员，并从同一个文件的原始声明中访问该成员。

这种行为意味着您可以使用扩展以相同的方式组织代码，无论您的类型是否具有私有实体。例如，给定以下简单的协议：

```
protocol SomeProtocol {
    func doSomething()
}
```

您可以使用扩展来添加协议一致性，如下所示：

```
struct SomeStruct {
    private var privateVariable = 12
}

extension SomeStruct: SomeProtocol {
    func doSomething() {
        print(privateVariable)
    }
}
```

泛型

泛型类型或泛型函数的访问级别是泛型类型或函数本身的访问级别以及其类型参数的任何类型限制的访问级别的最小值。

类型别名

出于访问控制的目的，您定义的任何类型别名都被视为不同的类型。类型别名可以具有小于或等于其别名类型的访问级别的访问级别。例如，私有类型别名可以使用私有类型，文件私有类型，内部类型，公共类型或开放类型，但公用类型别名不能使用内部类型，文件私有类型或私有类型。

> 注意
>
> 此规则也适用于用于满足协议一致性的关联类型的类型别名。