翻译：Hawstein 校对：numbbbbb

特性

本页内容包括：

特性提供了关于声明和类型的更多信息。在Swift中有两类特性，用于修饰声明的以及用于修饰类型的。例如，`required`特性，当应用于一个类的指定或便利初始化器声明时，表明它的每个子类都必须实现那个初始化器。再比如`noreturn`特性，当应用于函数或方法类型时，表明该函数或方法不会返回到它的调用者。

通过以下方式指定一个特性：符号`@`后面跟特性名，如果包含参数，则把参数带上：

```
@attribute name
@attribute name(attribute arguments)
```

有些声明特性通过接收参数来指定特性的更多信息以及它是如何修饰一个特定的声明的。这些特性的参数写在小括号内，它们的格式由它们所属的特性来定义。

声明特性

声明特性只能应用于声明。然而，你也可以将`noreturn`特性应用于函数或方法类型。

`assignment`

该特性用于修饰重载了复合赋值运算符的函数。重载了复合赋值运算符的函数必需将它们的初始输入参数标记为`inout`。如何使用`assignment`特

性的一个例子，请见：复合赋值运算符。

`class_protocol`

该特性用于修饰一个协议表明该协议只能被类类型采用[待改：adopted]。

如果你用`objc`特性修饰一个协议，`class_protocol`特性就会隐式地应用到该协议，因此无需显式地用`class_protocol`特性标记该协议。

`exported`

该特性用于修饰导入声明，以此来导出已导入的模块，子模块，或当前模块的声明。如果另一个模块导入了当前模块，那么那个模块可以访问当前模块的导出项。

`final`

该特性用于修饰一个类或类中的属性，方法，以及下标成员。如果用它修饰一个类，那么这个类则不能被继承。如果用它修饰类中的属性，方法或下标，则表示在子类中，它们不能被重写。

`lazy`

该特性用于修饰类或结构体中的存储型变量属性，表示该属性的初始值最多只被计算和存储一次，且发生在第一次访问它时。如何使用`lazy`特性的一个例子，请见：惰性存储型属性。

`noreturn`

该特性用于修饰函数或方法声明，表明该函数或方法的对应类型，`T`，是`@noreturn T`。你可以用这个特性修饰函数或方法的类型，这样一来，函数或方法就不会返回到它的调用者中去。

对于一个没有用`noreturn`特性标记的函数或方法，你可以将它重写(override)为用该特性标记的。相反，对于一个已经用`noreturn`特性标记的函数或方法，你则不可以将它重写为没使用该特性标记的。相同的规则试用于当你在一个comforming类型中实现一个协议方法时。

`NSCopying`

该特性用于修饰一个类的存储型变量属性。该特性将使属性的setter与属性值的一个副本合成，由`copyWithZone`方法返回，而不是属性本身的值。该属性的类型必需遵循`NSCopying`协议。

`NSCopying`特性的行为与Objective-C中的`copy`特性相似。

`NSManaged`

该特性用于修饰`NSManagedObject`子类中的存储型变量属性，表明属性的存储和实现由Core Data在运行时基于相关实体描述动态提供。

`objc`

该特性用于修饰任意可以在Objective-C中表示的声明，比如，非嵌套类，协议，类和协议中的属性和方法（包含getter和setter），初始化器，析构器，以下下标。`objc`特性告诉编译器该声明可以在Objective-C代码中使用。

如果你将`objc`特性应用于一个类或协议，它也会隐式地应用于那个类或协议的成员。对于标记了`objc`特性的类，编译器会隐式地为它的子类添加`objc`特性。标记了`objc`特性的协议不能继承自没有标记`objc`的协议。

`objc`特性有一个可选的参数，由标记符组成。当你想把`objc`所修饰的实体以一个不同的名字暴露给Objective-C，你就可以使用这个特性参数。你可以使用这个参数来命名类，协议，方法，getters，setters，以及初始化器。下面的例子把`ExampleClass`中`enabled`属性的getter暴露给Objective-C，名字是`isEnabled`，而不是它原来的属性名。

```
@objc
class ExampleClass {
    var enabled: Bool {
    @objc(isEnabled) get {
        // Return the appropriate value
    }
    }
}
```

`optional`

用该特性修饰协议的属性，方法或下标成员，表示实现这些成员并不需要一致性类型（conforming type）。

你只能用`optional`特性修饰那些标记了`objc`特性的协议。因此，只有类类型可以adopt和comform to那些包含可选成员需求的协议。更多关于如何使用`optional`特性以及如何访问可选协议成员的指导，例如，当你不确定一个conforming类型是否实现了它们，请见：可选协议需求。

`required`

用该特性修饰一个类的指定或便利初始化器，表示该类的所有子类都必需实现该初始化器。

加了该特性的指定初始化器必需显式地实现，而便利初始化器既可显式地实现，也可以在子类实现了超类所有指定初始化器后继承而来（或者当子类使用便利初始化器重写了指定初始化器）。

Interface Builder使用的声明特性

Interface Builder特性是Interface Builder用来与Xcode同步的声明特性。Swift提供了以下的Interface Builder特性：`IBAction`，`IBDesignable`，`IBInspectable`，以及`IBOutlet`。这些特性与Objective-C中对应的特性在概念上是相同的。

`IBOutlet`和`IBInspectable`用于修饰一个类的属性声明；`IBAction`特性用于修饰一个类的方法声明；`IBDesignable`用于修饰类的声明。

类型特性

类型特性只能用于修饰类型。然而，你也可以用`noreturn`特性去修饰函数或方法声明。

`auto_closure`

这个特性通过自动地将表达式封闭到一个无参数闭包中来延迟表达式的求值。使用该特性修饰无参的函数或方法类型，返回表达式的类型。一个如

何使用`auto_closure`特性的例子，见函数类型

`noreturn`

该特性用于修饰函数或方法的类型，表明该函数或方法不会返回到它的调用者中去。你也可以用它标记函数或方法的声明，表示函数或方法的相应类型，`T`，是`@noreturn T`。

特性的语法 attribute -> @ attribute-name attribute-argument-clauseopt attribute-name -> identifier attribute-argument-clause -> (balanced-tokensopt) attributes -> attribute attributesopt balanced-tokens -> balanced-token balanced-tokensopt balanced-token -> (balanced-tokensopt) balanced-token -> [balanced-tokensopt] balanced-token -> { balanced-tokensopt } balanced-token -> 任意标识符，关键字，字面量，或运算符 balanced-token -> 任意标点符号，除了(,), [,], {, 或 }