

关于原则：最重要的目标：每个元素都能够准确清晰的表达出它的含义。做出 API 设计、声明后要检查在上下文中是否足够清晰明白。

一、命名

#协议

(1) 协议描述的是“做的事情”，命名为名词

```
1 protocol TableViewSectionProvider {
2     func rowHeight(at row: Int) -> CGFloat
3     var numberOfRows: Int { get }
4     /* ... */
5 }
```

(2) 协议描述的是“能力”，需添加后缀able或 ing。（如 Equatable、ProgressReporting）

```
1 protocol Loggable {
2     func logCurrentState()
3     /* ... */
4 }
```

```
1 protocol Equatable {
2     func ==(lhs: Self, rhs: Self) -> bool {
3         /* ... */
4     }
5 }
```

(3) 如果已经定义类，需要给类定义相关协议，则添加Protocol后缀

```
1 protocol InputTextViewProtocol {
2     func sendTrackingEvent()
3     func inputText() -> String
4     /* ... */
5 }
```

#实现protocol:

如果确定protocol的实现不会被重写，建议用extension将protocol实现分离

推荐:

```
1 class MyViewController: UIViewController {
2     // class stuff here
3 }
4
5 // MARK: - UITableViewDataSource
6
7 extension MyViewController: UITableViewDataSource {
8     // table view data source methods
9 }
10
11 // MARK: - UIScrollViewDelegate
12
13 extension MyViewController: UIScrollViewDelegate {
14     // scroll view delegate methods
15 }
```

不推荐:

```
1 class MyViewController: UIViewController, UITableViewDataSource, UIScrollViewDelegate {
```

```

2 // all methods
3 }

```

2、Bool类型命名：用is最为前缀

```

1 var isString: Bool = true

```

3、枚举定义尽量简写，不要包括类型前缀

```

1 public enum UITableViewRowAnimation : Int {
2     case fade
3     case right // slide in from right (or out to right)
4     case left
5     case top
6     case bottom
7     case none // available in iOS 3.0
8     case middle // available in iOS 3.2. attempts to keep cell centered in the space it will/did occupy
9     case automatic // available in iOS 5.0. chooses an appropriate animation style for you
10 }
11
12 enum ZHCodeStandardEnum: Int {
13     case code = 0 // code无特殊情况 字母小写
14     case code1 = 1 // code1
15     case code2 = 2 // code2
16 }

```

4、swift建议不要使用前缀

```

1 推荐
2 HomeController
3 Bundle
4
5 不推荐
6 NEHomeController
7 NSBundle

```

5、减少不必要的简写

```

1 推荐
2 let viewFrame = view.frame
3 let textField = ...
4 let table = ...
5 let controller = ...
6 let button = ...
7 let label = ...
8
9 不推荐
10 let r = view.frame
11 let tf = ...
12 let tb = ...
13 let vc = ...
14 let btn = ...
15 let lbl = ...

```

6、变量命名应该能推断出该变量类型，如果不能推断，则需要以变量类型结尾

```

1 推荐
2 class TestClass: class {
3     // UIKit的子类，后缀最好加上类型信息
4     let coverImageView: UIImageView
5
6     @IBOutlet weak var usernameTextField: UITextField!
7
8     // 作为属性名的firstName，明显是字符串类型，所以不用在命名里不用包含String
9     let firstName: String
10
11     // UIViewContrller以ViewController结尾
12     let fromViewController: UIViewController
13 }
14
15 不推荐
16 class TestClass: class {
17     // image不是UIImageView类型
18     let coverImage: UIImageView
19
20     // or cover不能表明其是UIImageView类型
21     var cover: UIImageView
22
23     // String后缀多余
24     let firstNameString: String
25
26     // UIViewContrller不要缩写
27     let fromVC: UIViewController
28 }

```

7、省略所有的冗余的参数标签

```

1 func min(_ number1: Int, _ number2: Int) {
2     /* ... */
3 }
4
5 min(1, 2)

```

8、进行安全值类型转换的构造方法可以省略参数标签，非安全类型转换则需要添加参数标签以表示类型转换方法

```

1 extension UInt32 {
2     /// 安全值类型转换，16位转32位，可省略参数标签
3     init(_ value: Int16)
4
5     /// 非安全类型转换，64位转32位，不可省略参数标签
6     /// 截断显示
7     init(truncating source: UInt64)
8
9     /// 非安全类型转换，64位转32位，不可省略参数标签
10    /// 显示最接近的近似值
11    init(saturating valueToApproximate: UInt64)
12 }

```

9、当第一个参数构成整个语句的介词时（如，at, by, for, in, to, with 等），为第一个参数添加介词参数标签

```

1 推荐

```

```

2 // 添加介词标签havingLength
3 func removeBoxes(havingLength length: int) {
4     /* ... */
5 }
6
7 x.removeBoxes(havingLength: 12)

```

#例外情况是，当后面所有参数构成独立短语时，则介词提前：

```

1 推荐
2 // 介词To提前
3 a.moveTo(x: b, y: c)
4
5 // 介词From提前
6 a.fadeFrom(red: b, green: c, blue: d)
7
8
9 不推荐
10 a.move(toX: b, y: c)
11 a.fade(fromRed: b, green: c, blue: d)

```

10、当第一个参数构成整个语句一部分时，省略第一个参数标签，否则需要添加第一个参数标签。其余情况下，给除第一个参数外的参数都添加标签

```

1 // 参数构成语句一部分，省略第一个参数标签
2 x.addSubview(y)
3
4 // 参数不构成语句一部分，不省略第一个参数标签
5 view.dismiss(animated: false)

```

#不要使用冗余的单词，特别是与参数及参数标签重复

```

1 推荐
2 func remove(_ member: Element) -> Element?
3
4
5 不推荐
6 func removeElement(_ member: Element) -> Element?

```

#命名出现缩写词，缩写词要么全部大写，要么全部小写，以首字母大小写为准

```

1 推荐
2 let urlRouterString = "https://xxxxx"
3 let htmlString = "xxxx"
4 class HTMLModel {
5     /* ... */
6 }
7 struct URLRouter {
8     /* ... */
9 }
10
11
12 不推荐
13 let uRLRouterString = "https://xxxxx"
14 let hTMLString = "xxxx"

```

```

15 class HtmlModel {
16     /* ... */
17 }
18 struct UrlRouter {
19     /* ... */
20 }

```

参数名要准确的表达出参数类型:

```

1 // 推荐
2 class ConnectionTableViewCell: UITableViewCell {
3     let personImageView: UIImageView
4     let animationDuration: NSTimeInterval
5     // 作为属性名的firstName, 很明显是字符串类型, 所以不用在命名里不用包含String
6     let firstName: String
7     // 虽然不推荐, 这里用 Controller 代替 ViewController 也可以。
8     let popupController: UIViewController
9     let popupViewController: UIViewController
10    // 如果需要使用UIViewController的子类
11    // 如TableViewController, CollectionViewController, SplitViewController, 等, 需要在命名里标名类型。
12    let popupTableViewController: UITableViewController
13    // 当使用outlets时, 确保命名中标注类型。
14    @IBOutlet weak var submitButton: UIButton!
15    @IBOutlet weak var emailTextField: UITextField!
16    @IBOutlet weak var nameLabel: UILabel!
17 }
18
19
20 // 不推荐
21 class ConnectionTableViewCell: UITableViewCell {
22    // 这个不是 UIImage, 不应该以Image 为结尾命名。
23    // 建议使用 personImageView
24    let personImage: UIImageView
25    // 这个不是String, 应该命名为 titleLabel
26    let text: UILabel
27    // animation 不能清晰表达出时间间隔
28    // 建议使用 animationDuration 或 animationTimeInterval
29    let animation: NSTimeInterval
30    // transition 不能清晰表达出是String
31    // 建议使用 transitionText 或 transitionString
32    let transition: String
33    // 这个是ViewController, 不是View
34    let popupView: UIViewController
35    // 由于不建议使用缩写, 这里建议使用 ViewController替换 VC
36    let popupVC: UIViewController
37    // 技术上讲这个变量是 UIViewController, 但应该表达出这个变量是TableViewController
38    let popupViewController: UITableViewController
39    // 为了保持一致性, 建议把类型放到变量的结尾, 而不是开始, 如submitButton
40    @IBOutlet weak var btnSubmit: UIButton!
41    @IBOutlet weak var buttonSubmit: UIButton!
42    // 在使用outlets 时, 变量名内应包含类型名。
43    // 这里建议使用 firstNameLabel
44    @IBOutlet weak var firstName: UILabel!
45 }

```

11、数组和字典变量定义，定义时需要标明泛型类型，并使用更简洁的语法

```
1 推荐
2  var names: [String] = []
3  var lookup: [String: Int] = [:]
4
5  不推荐
6  var names = [String]()
7  var names: Array<String> = [String]() // 不够简洁
8
9  var lookup = [String: Int]()
10 var lookup: Dictionary<String, Int> = [String: Int]() // 不够简洁
```

12、常量定义，建议尽可能定义在Type类型里面，避免污染全局命名空间

```
1 推荐
2  class TestTableViewCell: UITableViewCell {
3      static let kCellHeight = 80.0
4
5      /* ... */
6  }
7  // uses
8  let cellHeight = TestTableViewCell.kCellHeight
9
10
11 不推荐
12 let kCellHeight = 80.0
13 class TestTableViewCell: UITableViewCell {
14     /* ... */
15 }
16 // uses
17 let cellHeight = kCellHeight
```

14、当方法最后一个参数是Closure类型，调用时建议使用尾随闭包语法

```
1 推荐
2  UIView.animate(withDuration: 1.0) {
3      self.myView.alpha = 0
4  }
5
6  不推荐
7  UIView.animate(withDuration: 1.0, animations: {
8      self.myView.alpha = 0
9  })
```

15、一般情况下，在逗号后面加一个空格

```
1 推荐
2  let testArray = [1, 2, 3, 4, 5]
3
4  不推荐
5  let testArray = [1,2,3,4,5]
```

16、基类属性及方法

```
1  // 私有属性及方法使用前缀 _
```

```

2 // let
3 private let _testA: String = "testA"
4 public let testB: Int = 2019
5
6 // var
7 private var _testC: String = "testC"
8 public var testD: String = "testD"
9
10 // 方法
11 private func _testFunction() {
12     /* ... */
13 }
14 public func b_testFunction() {
15     /* ... */
16 }
17 public func base_testFunction() {
18     /* ... */
19 }

```

二、注释

#尽可能使用Xcode注释快捷键 (⌘⌥/) [command + option + /]

```

1 推荐
2 /// <#Description#>
3 ///
4 /// - Parameter testString: <#testString description#>
5 /// - Returns: <#return value description#>
6 func testFunction(testString: String?) -> String? {
7     /* ... */
8 }
9
10
11 不推荐
12 // Comment
13 func testFunction(testString: String?) -> String? {
14     /* ... */
15 }

```

#使用// MARK: -, 按功能和协议/代理分组

```

1 /// MARK:顺序没有强制要求, 但System API & Public API一般分别放在第一块和第二块。
2
3 // MARK: - Public
4
5 // MARK: - Request
6
7 // MARK: - Action
8
9 // MARK: - Private
10
11 // MARK: - xxxDelegate
12
13

```

MARK: 用于方法或函数的注释

TODO: 表示这里的代码还需继续处理

FIXME: 表示这里修改了代码

```
1 // MARK: - 网络判断
2 extension IBCBaseViewController{
3     /// 判断当前ViewController是否显示
4     ///
5     /// - Returns: 是否显示
6     public func checkIsShow() -> Bool{
7         // FIXME: 修改bug
8         if self.isViewLoaded && self.view.window != nil{
9             //当前viewController在显示
10            return true
11        }
12        return false
13    }
14
15    /// 网络状态更改
16    @objc public func networkStatusChange(){
17        if checkIsShow(){
18            shouldUpdateDatas()
19        }else{
20            // TODO: 未在当前页显示处理
21        }
22    }
23
24    /// 需要更新数据
25    @objc open func shouldUpdateDatas(){
26        // TODO: 基础处理
27    }
28 }
```

#swift 注释是支持 mark down 语法的

```
1 /**
2  ## 功能列表
3  这个类提供一下很赞的功能，如下：
4  - 功能 1
5  - 功能 2
6  - 功能 3
7  ## 例子
8  这是一个代码块使用四个空格作为缩进的例子。
9      let myAwesomeThing = MyAwesomeClass()
10      myAwesomeThing.makeMoney()
11  ## 警告
12  使用的时候总注意以下几点
13  1. 第一点
14  2. 第二点
15  3. 第三点
16  */
17 class MyAwesomeClass {
18     /* ... */
19 }
```

12、对外接口不兼容时，使用@available(iOS x.0, *)标明接口适配起始系统版本号


```

1  @available(iOS x.0, *)
2  class myClass {
3
4  }
5
6  @available(iOS x.0, *)
7  func myFunction() {
8
9  }

```

三、关于闭包

在Closures中使用self时避免循环引用

推荐

```

1  resource.request().onComplete { [weak self] response in
2      guard let strongSelf = self else {
3          return
4      }
5      let model = strongSelf.updateModel(response)
6      strongSelf.updateUI(model)
7  }

```

不推荐

```

1  // 不推荐使用unowned
2  // might crash if self is released before response returns
3  resource.request().onComplete { [unowned self] response in
4      let model = self.updateModel(response)
5      self.updateUI(model)
6  }

```

不推荐

```

1  // deallocate could happen between updating the model and updating UI
2  resource.request().onComplete { [weak self] response in
3      let model = self?.updateModel(response)
4      self?.updateUI(model)
5  }

```

Golden Path, 最短路径原则

推荐

```

1  func login(with username: String?, password: String?) throws -> LoginError {
2      guard let username = context.username else {
3          throw .noUsername
4      }
5      guard let password = password else {
6          throw .noPassword
7      }
8
9      /* login code */
10 }

```

不推荐

```
1 func login(with username: String?, password: String?) throws -> LoginError {
2     if let username = username {
3         if let password = inputData.password {
4             /* login code */
5         } else {
6             throw .noPassword
7         }
8     } else {
9         throw .noUsername
10    }
11 }
```

单例

```
1 class TestManager {
2     static let shared = TestManager()
3
4     /* ... */
5 }
```

四、缩进和换行

1 使用四个空格进行缩进。

2 每行最多160个字符，这样可以避免一行过长。(Xcode->Preferences->Text Editing->Page guide at column: 设置成160即可)

3 确保每个文件结尾都有空白行。

4 确保每行都不以空白字符作为结尾 (Xcode->Preferences->Text Editing->Automatically trim trailing whitespace + Including whitespace-only lines).

5 左大括号不用另起一行。

遵守Xcode内置的缩进格式(如果已经遵守, 按下CTRL-I 组合键文件格式没有变化)。当声明的一个函数需要跨多行时, 推荐使用Xcode默认的格式

```
1 // Xcode针对跨多行函数声明缩进
2 func myFunctionWithManyParameters(parameterOne: String,
3                                     parameterTwo: String,
4                                     parameterThree: String) {
5     // Xcode会自动缩进
6     print("\(parameterOne) \(parameterTwo) \(parameterThree)")
7 }
8
9 // Xcode针对多行 if 语句的缩进
10 if myFirstVariable > (mySecondVariable + myThirdVariable)
11     && myFourthVariable == .SomeEnumValue {
12     // Xcode会自动缩进
13     print("Hello, World!")
14 }
```

当调用的函数有多个参数时, 每一个参数另起一行, 并比函数名多一个缩进。

```
1 someFunctionWithManyArguments(
2     firstArgument: "Hello, I am a string",
3     secondArgument: resultFromSomeFunction(),
4     thirdArgument: someOtherLocalVariable)
```

当遇到需要处理的数组或字典内容较多需要多行显示时, 需把 [和] 类似于方法体里的括号, 方法体里的闭包也要做类似处理。

```

1  someFunctionWithABunchOfArguments(
2      someStringArgument: "hello I am a string",
3      someArrayArgument: [
4          "dadada daaaa daaaa dadada daaaa daaaa dadada daaaa daaaa",
5          "string one is crazy - what is it thinking?"
6      ],
7      someDictionaryArgument: [
8          "dictionary key 1": "some value 1, but also some more text here",
9          "dictionary key 2": "some value 2"
10     ],
11     someClosure: { parameter1 in
12         print(parameter1)
13     })

```

应尽量避免出现多行断言，可使用本地变量或其他策略。

```

1  // 推荐
2  let firstCondition = x == firstReallyReallyLongPredicateFunction()
3  let secondCondition = y == secondReallyReallyLongPredicateFunction()
4  let thirdCondition = z == thirdReallyReallyLongPredicateFunction()
5  if firstCondition && secondCondition && thirdCondition {
6      // 你要干什么
7  }
8
9
10 // 不推荐
11 if x == firstReallyReallyLongPredicateFunction()
12    && y == secondReallyReallyLongPredicateFunction()
13    && z == thirdReallyReallyLongPredicateFunction() {
14     // 你要干什么
15 }

```

当在写一个变量类型，一个字典里的主键，一个函数的参数，遵从一个协议，或一个父类，不用在分号前添加空格。

```

1  // 指定类型
2  let pirateViewController: PirateViewController
3  // 字典语法(注意这里是向左对齐而不是分号对齐)
4  let ninjaDictionary: [String: AnyObject] = [
5      "fightLikeDairyFarmer": false,
6      "disgusting": true
7  ]
8  // 声明函数
9  func myFunction<t, u: someprotocol where t.relatedtype == u>(firstArgument: U, secondArgument: T) {
10     /* ... */
11 }
12 // 调用函数
13 someFunction(someArgument: "Kitten")
14 // 父类
15 class PirateViewController: UIViewController {
16     /* ... */
17 }
18 // 协议
19 extension PirateViewController: UITableViewDataSource {
20     /* ... */
21 }

```

五、操作符

二元运算符(+, ==, 或->)的前后都需要添加空格，左小括号后面和右小括号前面不需要空格。

```
1 let myValue = 20 + (30 / 2) * 3
2 if 1 + 1 == 3 {
3     fatalError("The universe is broken.")
4 }
5 func pancake() -> Pancake {
6     /* ... */
7 }
```

六、其他

1 多使用let，少使用var

2 少用!去强制解包

3 可选类型拆包取值时，使用if let判断

4 不要使用 as! 或 try!

5 数组访问尽可能使用 .first 或 .last, 推荐使用 for item in items 而不是 for i in 0..

6 如果变量能够推断出类型，则不建议声明变量时指明类型

7 如果变量能够推断出类型，则不建议声明变量时指明类型

8 switch case选项不需要使用break关键词

9 访问控制

10 对于私有访问，如果在文件内不能被修改，则标记为private；如果在文件内可修改，则标记为fileprivate

11 对于公有访问，如果不希望在外面继承或者override，则标记为public，否则标明为open

12 访问控制权限关键字应该写在最前面，除了@IBOutlet、IBAction、@discardableResult、static 等关键字在最前面

13 如调用者可以不使用方法的返回值，则需要使用@discardableResult标明

14 使用==和!=判断内容上是否一致

15 使用===和!==判断class类型对象是否同一个引用，而不是用 ==和!=

16 Runtime兼容

17 Swift语言本身对Runtime并不支持，需要在属性或者方法前添加dynamic修饰符才能获取动态型，继承自NSObject的类其继承的父类的方法也具有动态型，子类的属性和方法也需要加dynamic才能获取动态性

七、Objective-C兼容

1 Swift接口不对Objective-C兼容，在编译器或者Coding中就会出现错误

2 暴露给Objective-C的任何接口，需要添加@objc关键字，如果定义的类继承自NSObject则不需要添加

3 如果方法参数或者返回值为空，则需要标明为可选类型

八、2019.10.8补充

关于空白：

- 用 tab，而非 空格
- 文件结束时留一空行
- 用足够的空行把代码分割成合理的块
- 不要在一行结尾留下空白
- 千万别在空行留下缩进
- 一块代码中间不要留空白

关于返回：

```
1 if n.isNumber {
```

```
2      /* ... */
3  } else {
4      return
5  }
6
7  用这个:
8  guard n.isNumber else {
9      return
10 }
11 /* ... */
```

关于解包:

```
1  推荐:
2  if let foo = foo {
3      // Use unwrapped `foo` value in here
4  } else {
5      // If appropriate, handle the case where the optional is nil
6  }
7  或者使用可选链, 比如:
8  // Call the function if `foo` is not nil. If `foo` is nil, ignore we ever tried to make the call
9  foo?.callSomethingIfFooIsNotNil()
10
11 不推荐:
12  foo!.callSomethingIfFooIsNotNil()
```

关于函数返回值:

```
1  推荐:
2  func testStringFunction() -> String
3
4  func testFunction()
5
6  不推荐:
7  func testFunction() -> Void
```