天外天移动 iOS 组 Swift 语言规范

移动端 App 由于执行环境和系统平台的千差万别，特别是 iOS 平台需要对提交的应用进行审核的限制，导致应用的应急维护成本相较 Web 端高了许多，所以对 iOS 开发、测试人员的要求也就更高。在我们 App 的长期开发迭代维护过程中，踩过很多坑，也积累了很多的经验，因此总结为此规范，可以指导写出更好的代码来保障 App 的性能和稳定性。

本规范针对 Swift 语言，包括代码规范、命名规范、工程规范、最佳实践、Demo 示例等几个章节。

本规范一共有三个约束等级，根据约束力强弱，规约依次分为强制、推荐、参考三大类:

- 【强制】 必须遵守。是不得不遵守的约定，违反本约定或将会引起严重的后果。
- 【推荐】 尽量遵守。长期遵守这样的规定，有助于系统稳定性和合作效率的提升。
- 【参考】 充分理解。技术意识的引导，是个人学习、团队沟通、项目合作的方向。

一. 代码规范:

1. 符号规范:

- 【推荐】每条语句后面不要加分号。
- 【推荐】逗号前面不要空格，后面需要一个空格。

```
let array = [1, 2, 3, 4]
```

- 【推荐】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。

正例:

```
let number = 1 * 2
```

```
if number == 1 { ... }
```

```
func sum(of a: Int, and b: Int) -> Int {
    return a + b
}
sum(of: 1, and: 2)
```

【推荐】条件两侧不要加括号。

正例：

```
if name == "Hello" {
    print("World")
}
```

反例：

```
if (name == "Hello") {
    print("World")
}
```

-

【推荐】左大括号不用换行，大括号里的语句不要写在一行

正例：

```
if number == 1 {
    return true
}
```

```
// 闭包除外
 array.filter({ return $1 == 0 })
```

反例：

```
if number == 1 { return true }
```

-

【推荐】左大括号前和右大括号后各留一个空格。

正例：

```
if score < 60 {
    print("不及格")
} else if score < 90 {
    print("及格")
} else {
```

```
6       print("优秀")
7   }
```

- 【推荐】当指定一个类型时，把冒号前不留空格，冒号后留一个空格。

正例：

```
1 class Car: Vehicle { ... }
```

```
1 let wheels: [Wheel]
```

```
1 func drive(to: Location) { ... }
```

反例：

```
1 class Car:Vehicle { ... }
```

```
1 class Car :Vehicle { ... }
```

```
1 class Car : Vehicle { ... }
```

- 【参考】字典中冒号的写法不唯一，选取自己喜欢的方式

```
1 let dict: [String : String] = ["playground" : "noun", "swift" : "adject
```

```
1 let dict: [Country: City] = ["China": "Beijing", "Japan": "Tokyo"]
```

- 【参考】尽量使用类型语法糖(Gramma Sugar)。

正例：

```
1  let array: [Int] = []
2  let dictionary: [String: Int] = [:]
```

反例：

```
let array: Array<Int> = Array<Int>()
let dictionary: Dictionary<String, Int> = Dictionary<String, Int>()
```

2. 留白：

- 【推荐】不要用空格，用 Tab 替代
- 【推荐】文件结束时留一空行
- 【推荐】用空行把代码分割成合理的块
- 【推荐】不要在一行结尾留下空白
- 【推荐】不要在空行留下缩进

二. 命名规范

1. 类、结构体、枚举、协议命名规范

- 【强制】类、结构体、枚举、协议的命名使用帕斯卡命名法（大驼峰法），即每个单词的首字母大写，中间没有分割符。

正例：

```
class SearchViewController: UIViewController { ... }
struct SessionManager { ... }
enum UITableViewStyle { ... }
```

反例：

```
class searchViewController: UIViewController { ... }
struct session_manager { ... }
enum uitableviewstyle { ... }
```

- 【推荐】在Swift中不用像 Objective-C 一样添加类前缀（但可以加上子模块名）

正例：

```
// Bicycle: 自行车模块里的 Controller
class BicycleSearchController: UIViewController { ... }
struct BYXSessionManager { ... }
enum JHXTableViewStyle { ... }
```

反例：

```
class HCPageController: UIViewController { ... }
struct BYXSessionManager { ... }
enum JHXTableViewStyle { ... }
```

- 【推荐】命名应该具有描述性和清晰性。

正例：

```
class RoundAnimatingButton: UIButton { ... }
```

反例：

```
class CustomButton: UIButton { ... }
```

- 【推荐】根据苹果接口设计指导文档, 如果协议描述的是协议做的事应该命名为名词(如 Collection)，如果描述的是行为，需添加后缀 able 或 ing (如 Equatable 和 ProgressReporting)。 如果上述两者都不能满足需求，可以添加 Protocol 作为后缀，例子见下面。

正例：

```
// 这个协议描述的是协议能做的事，应该命名为名词。
 protocol TableViewSectionDataSource {
     func rowHeight(atRow row: Int) -> CGFloat
     var numberOfRows: Int { get }
 }
 // 这个协议表达的是行为，以able最为后缀
 protocol Loggable {
     func logCurrentState()
 }
 // 因为已经定义类InputTextView，如果依然需要定义相关协议，可以添加Protocol作为
 protocol InputTextViewProtocol {
     func sendTrackingEvent()
     func inputText() -> String
 }
```

2. 方法命名规范

- 【推荐】保证短的函数定义在同一行中，并且包含左大括号。

正例：

```
func reticulateSplines(_ spline: [Double]) -> Bool { ... }
```

- 【推荐】在一个长的函数定义时，在适当的地方进行换行，同时在下一行中添加一个额外的缩进。

正例：

```
func reticulateSplines(_ spline: [Double], factor: Double,
    constant: Int, comment: String) -> Bool { ... }
```

- 【推荐】如果闭包表达式参数在参数列表中的最后一个时，使用结尾闭包表达式。如果结尾有超过一个的闭包，不要使用结尾闭包。

正例：

```
UIView.animate(withDuration: 0.5) {
    fooView.alpha = 0
}
UIView.animate(withDuration: 0.5, animations: {
    fooView.alpha = 0
}, completion: { finished in
    fooView.removeFromSuperView()
})
```

反例：

```
UIView.animate(withDuration: 0.5, animations: {
    fooView.alpha = 0
})
UIView.animate(withDuration: 0.5, animations: {
    fooView.alpha = 0
}) { f in
    fooView.removeFromSuperView()
}
```

-

【推荐】当单个闭包表达式上下文清晰时，使用隐式的返回值。

```
// 最长方式
 items.sort(by: { (a: Int, b: Int) -> Bool in
     return a > b
 })
 // 最短方式
 items.sort { $0 > $1 }
```

3. 变量命名规范

- 【推荐】使用前缀k + 大骆驼命名法为所有非单例的静态常量命名。 > 正例： > > swift
class SomeClass {   static let kSomeConstantHeight: CGFloat = 80.0
    static let kDeleteButtonColor = UIColor.red   // 单例
    static let shared = SomeClass() }

-

【强制】如果原有命名不能明显表明类型，则命名内要包括类型信息。

正例：

```
class ConnectionTableViewCell: UITableViewCell {
     let personImageView: UIImageView
     let animationDuration: TimeInterval
     // 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Stri
     let firstName: String
     // 这里用 Controller 代替 ViewController 也可以。
     let popupController: UIViewController
     let popupViewController: UIViewController
     // 如果需要使用UIViewController的子类，如TableViewController，Collect
     let popupTableViewController: UITableViewController
     // 当使用 outlets 时，确保命名中标注类型。
     @IBOutlet weak var submitButton: UIButton!
     @IBOutlet weak var emailTextField: UITextField!
     @IBOutlet weak var nameLabel: UILabel!
 }
```

反例：

```
class ConnectionTableViewCell: UITableViewCell {
```

```
    // 这个不是 UIImage，不应该以Image 为结尾命名。
    // 建议使用 personImageView
    let personImage: UIImageView
    // 这个不是String，应该命名为 textLabel
    let text: UILabel
    // animation 不能清晰表达出时间间隔
    // 建议使用 animationDuration 或 animationTimeInterval
    let animation: NSTimeInterval
    // transition 不能清晰表达出是String
    // 建议使用 transitionText 或 transitionString
    let transition: String
    // 这个是ViewController，不是View
    let popupView: UIViewController
    // 由于不建议使用缩写，这里建议使用 ViewController替换 VC
    let popupVC: UIViewController
    // 技术上讲这个变量是 UIViewController，但应该表达出这个变量是TableView(
    let popupViewController: UITableViewController
    // 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
    @IBOutlet weak var btnSubmit: UIButton!
    @IBOutlet weak var buttonSubmit: UIButton!
    // 在使用outlets 时，变量名内应包含类型名。
    // 这里建议使用 firstNameLabel
    @IBOutlet weak var firstName: UILabel!
}
```

4. 其他命名规范

- 【推荐】缩略词在命名种一般是全部大写，例外的情况是以缩略词开头并首字母需要小写时，缩略词全部小写。

正例：

```
let htmlContent = "<p>Hello!</p>"
func getNBAPlayers -> [String] { ... }
let queryURL = "https://open.twtstudio.com/query"
class HTMLLoader { ... }"
struct SolaURLManager { ... }"
```

反例：

```
let HTMLContent = "<p>Hello!</p>"
func getNbaPlayers -> [String] { ... }
```

```
let queryUrl = "https://open.twtstudio.com/query"
class HtmlLoader { ... }"
struct SolaUrlManager { ... }"
```