TiCheck 编码规范

ID: TC-DDP

同济大学苹果俱乐部

TiCheck 项目团队

2014年4月1日

TiCheck 编码规范

ID: TC-DDP

修订历史记录

编写日期	SEPG	版本	说明	作者	评审时间	参与人员	批准日期	确认人员
2014.4.1	TC	0.1	初稿	杨明	2014.4.2	李博一，黄泽彪，邱峰，刘大畅，李韧	2014.4.2	杨明
2014.4.9	TC	0.2	修改稿	杨明	2014.4.10	李博一，黄泽彪，邱峰，刘大畅，李韧	2014.4.10	杨明
2014.4.14	TC	1.0	第一版发布	杨明	2014.4.14	李博一，黄泽彪，邱峰，刘大畅，李韧	2014.4.14	杨明

目录

TiCheck 编码规范 **2**

目录 **2**

1. 引言 **4**

2. 项目级 **5**

2.1. 文件位置放置 5

2.1.1. 案例 5

2.1.2. 参考 7

2.2. 如何组织类 7

3. Objective-C类级 **9**

3.1. 命名属性、常量、变量和方法。 9

3.1.1. 案例 9

3.1.2. 命名准则 9

3.1.3. 参考 9

3.2. 属性、变量和方法成员的访问权限 9

3.2.1. 参考 9

3.3. 集合类型 10

3.3.1. 参考 10
3.4. 使用mark 10
4. Storyboard级 11
4.1. Storyboard实践 11
4.1.1. 适配3.5英寸 11
4.1.2. 参考 11
5. 应用编码规范到实际开发中 12
5.1. 代码审查 12
5.2. 代码重构 12
5.3. 如何定义完成 12
5.3.1. 完成包括一下几点 12
5.3.2. 参考 13
6. 通用级 14
6.1. 不允许注释代码 14
6.2. 标记未完成部分代码 14
7. 词表 15

1. 引言

此文档包含了在编码规范会议上讨论的针对Objective-C编程语言的编码规范。这些规范并不是一成不变的，反之我们允许进一步的讨论以及更改。将这些编码规范用于我们每日的工作当中可以使得我们可以更好的理解其他人的代码。同时这也属于我们进行代码审查的重要依据。

2. 项目级

2.1. 文件位置放置

- 我们采用**MVC（Model-View-Controller）**界面设计模式[R1]，文件分类主要按照文件类型，所属层级以及功能模块分类。
- 对于**View**我们采用Storyboard进行布局，部分为代码实现或使用Nib进行布局的，我们将尽可能一步步转为Storyboard实现。Storyboard置于PresentationLayer中。
- 所有**Model**我们放在LogicLayer文件夹下的Entity中，不进一步进行分级。
- **Controller**文件放在PresentationLayer下，根据功能模块划分不同的子文件夹。
- **LogicLayer**中除了放置Model之外，主要为存放所有业务逻辑，其中OTA文件夹中存放携程API的实现部分，ServerCommunication部分完成与后台服务器的数据交互，ConfigurationHelper管理项目的配置的单例，包括数据的配置等，APIResourceHelper为管理所有静态资源文件（主要为静态XML）的单例。
- **Category**为下放置类的Category文件，用于扩展已存在的类功能。
- **Resources**文件夹下放置所有非图片的资源，例如字体，证书，静态XML资源文件等。
- 所有图片文件放置于**Images.xcassets**中，按不同界面划分不同子文件夹。同时图片资源上我们只支持Retina。
- 所有单元测试部分放置于**TiCheckTests**中。
- 所有第三库的采用**CocoaPods**管理。

2.1.1. 案例

TiCheck (Project)

TiCheck

Category

UIImage+ImageResize.h/m

LogicLayer

APIResourceHelper.h/m

ConfigurationHelper.h/m

Entity

Flight.h/m

OTA

OTARequest

OTAFlightSearch.h/m

OTAResponse

PresentationLayer

Main.storyboard

PersonalCenter

PersonalCenterViewController.h/m

BookListViewController.h/m

TicketInfo

Resources

Images.xcassets

TiCheckTests

Pods (Project)

Pods

AFNetworking

MBProgressHUD

2.1.2. 参考

- [R1] https://developer.apple.com/library/ios/featuredarticles/viewcontrollerpgforiphoneos/Introduction/Introduction.html

2.2. 如何组织类

- Model类[E1]

[E1] Model.h/m

Macro

Const

Properties

Constructor

Public Methods

Private/Helper Methods

Nested Types

- Controller类[E2]

[E2] Controller.h/m

extern

Const

Properties

Static Fields

InstanceFields

dependencies

other field

Life Cycle Related

Event Handler

Selector

Delegate Methods

Private/Helper Methods

Navigation

---

3. Objective-C类级

3.1. 命名属性、常量、变量和方法。

- 基于小驼峰命名法，即第一个单词小写字母开头，其他单词首字母大写。
- 类名、协议名遵从大驼峰命名法，所有单词首字母大写。
- 常量，包括宏定义，采用小写“k”作为前缀，即名称遵循大驼峰命名法。
- 方法名和方法参数采用小驼峰命名法，并且方法名不允许get前缀，对于参数过多的，采用每个参数各占一行，参考[E1]。

3.1.1. 案例

[E1]

```
+ (NSString *)soap11ResponseWithURL:(NSString *)webURL
                  flightRequestType:(FlightRequestType)requestType
                       xmlNameSpace:(NSString *)nameSpace
                    webServiceName:(NSString *)serviceName
                    xmlRequestBody:(NSString *)requestBody;
```

3.1.2. 命名准则

- 保证清晰描述，对于有歧义的必须有注释。
- 保证命名含义与功能的一致。
- 进一步阅读Apple Coding Guidelines[R1]。

3.1.3. 参考

- [R1] https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CodingGuidelines/CodingGuidelines.html

3.2. 属性、变量和方法成员的访问权限

- 在.h中声明的属性，变量和方法即意味着public。
- 在.m中在interface extension中声明即意味这private，参考[R1]。
- 保证对属性、变量和方法使用最具有限制性的访问权限。

3.2.1. 参考

- [R1] http://www.tutorialspoint.com/objective_c/objective_c_extensions.htm

3.3. 集合类型

- 根据数据属性进行正确集合选择，对于数组我们使用Array，键值对采用Dictionary，对于无序集合采用Set。
- 对于只读集合，即采用NSArray，NSDictionary和NSSet。
- 对于可写的集合，要采用NSMutableArray，NSMutableDictionary和NSMutableSet。
- 对于进一步更具体的情况，可以采用更加具体的集合类型以提高性能，相关阅读[R1]。

3.3.1. 参考

- [R1] http://www.objc.io/issue-7/collections.html

3.4. 使用mark

- 对于Objective-C代码中，使用#pragma mark -进行适当的划分。

4. Storyboard级

4.1. Storyboard实践

- 在ViewController中放置新的ViewController采用Container的方式实现。
- 对于界面跳转，使用Segue。
- 对于自定义的UITableViewCell或UICollectionViewCell创建相应的类，不要只是单独使用Prototype Cell。
- 使用UIStoryboard注意参考中的几点[R1]

4.1.1. 适配3.5英寸

- 对于同一个View，尽可能使用同一种方式进行不同尺寸的适配，即Auto Layout或Code Behind。推荐使用Auto Layout，更好的明确View和Controller的职责。
- 尽可能对每个View设定Auto Layout，Auto Layout指南见参考[R2]。

4.1.2. 参考

- [R1] http://robsprogramknowledge.blogspot.de/2012/01/uistoryboard-best-practices.html
- [R2] https://developer.apple.com/library/ios/documentation/userexperience/conceptual/AutolayoutPG/Introduction/Introduction.html

5. 应用编码规范到实际开发中

为了将编码规范用在我们每日的开发任务当中，我们必须采用以下模式进行开发。针对每一个功能模块我们会创建新的分支，我们的目标是将分支最终并入到主干中，同时我们需要保证开发速度与开发质量。在大多数情况下，这两个目标总有冲突，在此情况下，Reviewer以及Developer需要进行讨论并找到一个合理折中的方式去解决问题。

5.1. 代码审查

- 在每一个分支需要合并到主干时，必须进行至少一次的代码审查。Reviewer找到其中不符合我们代码规范或难以理解的部分添加相应的TODO注释。如有必要，需进行多次迭代审查。
- Developer在提交代码给Reviewer之前自己进行一次审查，并进行必要的重构，这会减少迭代审查的次数。
- 对于小的问题，Reviewer应该当即立刻解决并告知Developer。
- Reviewer必须进行彻底的排查。

5.2. 代码重构

- 代码重构同样必须参照代码规范进行，同时重构结束之后仍然需要由Reviewer进行审查。
- 对于不紧急的重构，Developer需要进行标记，标记方式在第六部分提到。
- 重构要点之一为将大的功能模块分成小部分，这样就可以分开进行审查和测试。

5.3. 如何定义完成

5.3.1. 完成包括一下几点

1. 实现了需要的功能；
2. 代码进行过了审查；
3. 代码进行过了重构；
4. 开发团队对功能进行了测试；
5. 此功能部分被合并到了主干；
6. 客户对功能进行了测试；
7. 在客户的环境下，成功部署此功能；
8. 客户确认功能实现。

5.3.2. 参考

- [R1] http://www.scrumalliance.org/community/articles/2008/september/what-is-definition-of-done-%28dod%29

6. 通用级

6.1. 不允许注释代码

- 描述和解释性注释允许出现，但注释代码只能降低代码可读性。

6.2. 标记未完成部分代码

- TODO !!!: 标记重要或紧急的任务部分。
- TODO ???: 标记难以理解的代码部分。
- TODO FIXME: 标记BUG的部分。

7. 词表

代码规范：代码规范为针对某一种编程语言以及建议的编程风格、实践的一系列准则，一般来说包括文件组织，缩进，注释，命名规范，编程实践，编程原则以及架构实践等，这些规范用于保证软件结构的质量。

MVC：MVC模式（Model-View-Controller）是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）。

CocoaPods：CocoaPods是一个用于Objective-C编程语言的应用级运行时依赖管理工具，它提供了标准的管理第三方库的格式。

驼峰式大小写：驼峰式大小写（Camel-Case，Camel Case，camel case），计算机程序编写时的一套命名规则（惯例）。当变量名和函数名称是由二个或多个单字链接在一起，而构成的唯一识别字时，利用“驼峰式大小写”来表示，可以增加变量和函数的可读性。

代码审查：代码审查是指对计算机源代码系统化地审查，常用软件同行评审的方式进行，其目的是在找出及修正在软件开发初期未发现的错误，提升软件质量及开发者的技术。代码审查常以不同的形式进行，例如结对编程、非正式的看过整个代码，或是正式的软件检查。