

EasyIOS

朱潮

Published
with GitBook



Table of Contents

1. [Introduction](#)
2. [Home](#)
 - i. [Scene](#)
 - ii. [SceneModel](#)
 - i. [Action](#)
 - ii. [Request](#)
 - iii. [Model](#)
 - iv. [ReactiveCocoa](#)
 - v. [AutoLayoutCell](#)
 - vi. [Pulltorefresh](#)
 - vii. [AutoLayout-UIScrollView](#)
 - viii. [URLManager](#)
 - ix. [Others](#)
 - i. [EasyKit](#)
 - ii. [IconFont](#)
 - iii. [RegexCategories](#)
 - iv. [TMCache](#)
 - v. [GCDObjC](#)
 - vi. [FLKAutoLayout](#)
 - vii. [UICKeyChainStore](#)
 - viii. [Crontab-IOS](#)
3. [FAQ](#)



EasyIOS - Happy New Year 2015

pod v2.2.2 platform ios license MIT QQ群 340906744

EasyIOS is a new generation of development framework based on `Model-View-ViewModel` which makes faster and easier app development, Build your app by geek's way.

- [EasyIOS官方论坛-IOsx](#)
- [EasyIOS小组](#)
- [EasyIOS在线文档](#)
- EasyIOS官方qq群 :[340906744](#) 欢迎大家加入讨论
- [EasyRSS](#):基于EasyIOS 2.2.2 的开源项目,EasyIOS的官方Demo
- [RACSwift for EasyIOS](#):EasyIOS swift 版本Demo
- [json转Model工具 ModelCoder](#)
- [基于EasyIOS的上线应用](#)

EasyIOS 以提升开发效率为宗旨

- 代码分离 - `Model-View-ViewModel` - 分离ViewController中的大量逻辑代码, 解决ViewController承担了过多角色而造成的代码质量低下。增加视图与模型的绑定特性。
- 自动持久化 - `Model to Db` - 我再也不想思考如何实现持久化了。在我的想法里, 将模型对象直接扔到一个bucket里, 然后它就能自动的对数据进行存储、缓存、合并以及唯一化。我应当关注于描述对象间的属性和联系, 以及我希望它们分组的方式。其他的实现细节都应该是不可见的。
- 自动RESTful API - `Json to Model` - 一旦我给程序发出指令, 将一个API响应对应到一个数据对象, 网络和JSON转换应该被自动完成。我只想关注如何将JSON中那些项目展示给用户。
- 有表现力的触发器和响应 - `ReactiveCocoa` - 我想用源于响应意图 (Intent) 的语法来描述事件的响应和触发器, 我不关心它们间的连接是如何实现的, 并且这些连接也不应该在重构时出错。
- 简洁明了的网络请求 - `Action and Request` - 对于简单的GET、POST请求, 可以进行对象化操作, 我只想告诉程序, 链接在哪里, 有哪些参数, 接下来就自动拉取到想要的数 据, 顺便帮我把缓存也做齐了, 也是极好的。
- 便捷的UI布局 - `FLKAutolayout` - 更加便捷的进行autolayout布局, 不管你使用springs & struts或者AutoLayout, 每种方法都需要你明确相关视图如何排列。你需要花大量的时间编写和修正这些排列, 特别是现在有这么 多设备需要适配 的情况下。没有什么是自动写好的, UI布局依赖于对细节的不断调

整。推荐开发期间Debug工具FLEX, pod 'FLEX', '~> 1.1.1' 需要手动集成, 发布release版本时请删除。

- 友好的线程控制 - GCDObjC -
- 便捷的正则匹配
- 富文本的Label
- and so on.....

The MVVM(Model-View-ViewModel)

全新基于 MVVM(Model-View-ViewModel) 编程模式架构, 开启EasyIOS开发函数式编程新篇章。

EasyIOS 2.0类似 AngularJs , 最为核心的是: MVVM、ORM、模块化、自动化双向数据绑定、等等

喜欢 swift 的同学, 同样有 swift 的2.0 demo [RACSwift for EasyIOS](#), 供大家学习。

关于有疑问什么是MVVM, 以及为什么IOS开发需要MVVM思想编程的, 请看文章[用Model-View-ViewModel构建iOS App](#)有详细介绍。

EasyIOS 2.0是基于MVVM编程思想进行构建的, 封装了Scene,SceneModel,Model, Action四种模型来对IOS进行开发, 4种模型的定义解决了IOS开发中ViewController承担了过多角色而造成的代码质量低下, 使得结构思路更加清晰。

- 1.其中 Scene 就是 ViewController 的子类, 仅仅负责界面的展示逻辑
- 2. Model 数据模型, 父类实现了ORM, 可以实现json、object、sqlite三者之间的一键转换,
- 3. SceneModel 视图-数据模型, 主要负责 视图与模型的绑定工作, 其中binding的工作交给了 ReactiveCocoa 。
- 4. SceneModel 包含 Action 成员, Action 类主要负责网络数据的请求,数据缓存, 数据解析工作

如果你有看Github的Trending Objective-C榜单, 那你肯定是见过 ReactiveCocoa 了。如果你在微博上关注唐巧、onevc等国内开发者。那也应该听说过 ReactiveCocoa 了。 ReactiveCocoa 简称RAC, 就是基于响应式编程思想的Objective-C实践, 它是Github的一个开源项目, 你可以在[这里](#)找到它。

二次封装 AFNetworking , 集成到Action, 增加了网络缓存功能, 轻松控制是否启用缓存。

采用 ReactiveCocoa 框架, 实现响应式编程, 减少代码复杂度。

Model 类整合 JsonModel 的类库和 MojoDataBase 类库

整合了很多开源的优秀代码

常用类库:

Action 负责网络数据请求

Model 负责数据存储

SceneModel 负责 Scene 与 Model 的绑定, 调用action进行数据请求

Scene 一个视图相当于 UIViewController ,提供了快速集成网络请求和下拉刷新上拉加载的方法。

SceneTableView 一个 TableView , 配合 Scene 提供了集成下拉刷新上拉加载的方法

SceneCollectionView 一个 UICollectionView，配合 Scene 提供了集成下拉刷新上拉加载的方法

- [git on oschina](#)
- [git on github](#)

How To Install

- Import from CocoaPods
- Add below to Podfile and run pod install

```
platform :ios, "6.0"
pod 'EasyIOS', '~> 2.2.2'
pod 'EasyIOS-Extention', '~> 1.2'
```

- If you use swift , please click [here](#)

2.2.2 版本更新

- 指定RAC最高版本号为2.4.4
- Action类POST方法增加缓存功能
- Request类增加设置http头信息

2.2版本更新

- 修改Action类中的配置方式：由原来的宏调用改为类方法配置
- 针对IOS8优化
- 为UIScrollView增加下拉放大效果
- 新增 EZNavigationController 类，解决ios7中快速push容易crash的问题
- 重写下拉刷新，与上拉加载，完全解耦，支持自定义UI。
- 新增AutoLayoutCell自适应高度的cell，以及UIScrollView的AutoLayout实现自适应contentSize
- 智能键盘，防止键盘遮盖输入框
- 新增强大的XAspect,完美实现代码解耦，再一次为UIViewController减负
- 兼容IOS8的 UIAlertViewController 与IOS7的 UIActionSheet 、 UIAlertView
- Action类新增Download下载方法
- EasyIOS-Extention 分离可选的第三方类库
- 移除了 RTLable , EzUILabel
- 修复部分bug
-

2.1版本更新

- 多谢各位小伙伴们支持以及不断的提出Issues，清晰的指出了EasyIOS的优化项,本次更新主要针对网络访问Action类
- 2.1版本移除了 MKNetworkKit，基于现有的api重新封装了 AFNetworking，并且加入了缓存控制。
- 如果利用 Action 类来发起请求的小伙伴可以体验到无痛升级的快感。。
- 同时移除 UIImageView+MKNetworkKit 替换为大家熟悉的 SDWebImage，解决图片闪烁问题。
- 修复部分循环引用的bug

2.0.3版本更新

- 紧急修复2.0.2 model层 初始化数据为nil的bug
- 移除 `EUIGestureRecognizer`
- 增加 `SHGestureRecognizerBlocks`

2.0.2版本更新

- 新增gcd封装 `GCDObjC` ,告别CocoaTouch原生难记的gcd调用方法
- 新增正则表达式封装 `RegexCategories` ,可以轻松的开始码正则表达式了
- 新增缓存处理封装 `TMCache` ,方便手动操作数据缓存(MK的自动缓存没有采用TMCache)
- Model层升级：[MojoDatabase+JsonModel](#)新增自动检测、自动创建数据表，新增查询方法，order by、group by、limit等等方法
- 新增功能[教程在Wiki Paper](#)

2.0.1版本更新

- 修改pod依赖
- [FontIcon](#)剥离项目，单独维护
- Model层修改：移除Jastor，添加JsonModel[MojoDatabase+JsonModel](#)剥离项目，单独维护。
- 增加懒人程序代码生成工具[ModelCoder](#)
- 移除部分非必要类库，代码整合
- 本着引导大家编程更easy的原则,增加 `Easykit` 、 `FLKAutoLayout` 类

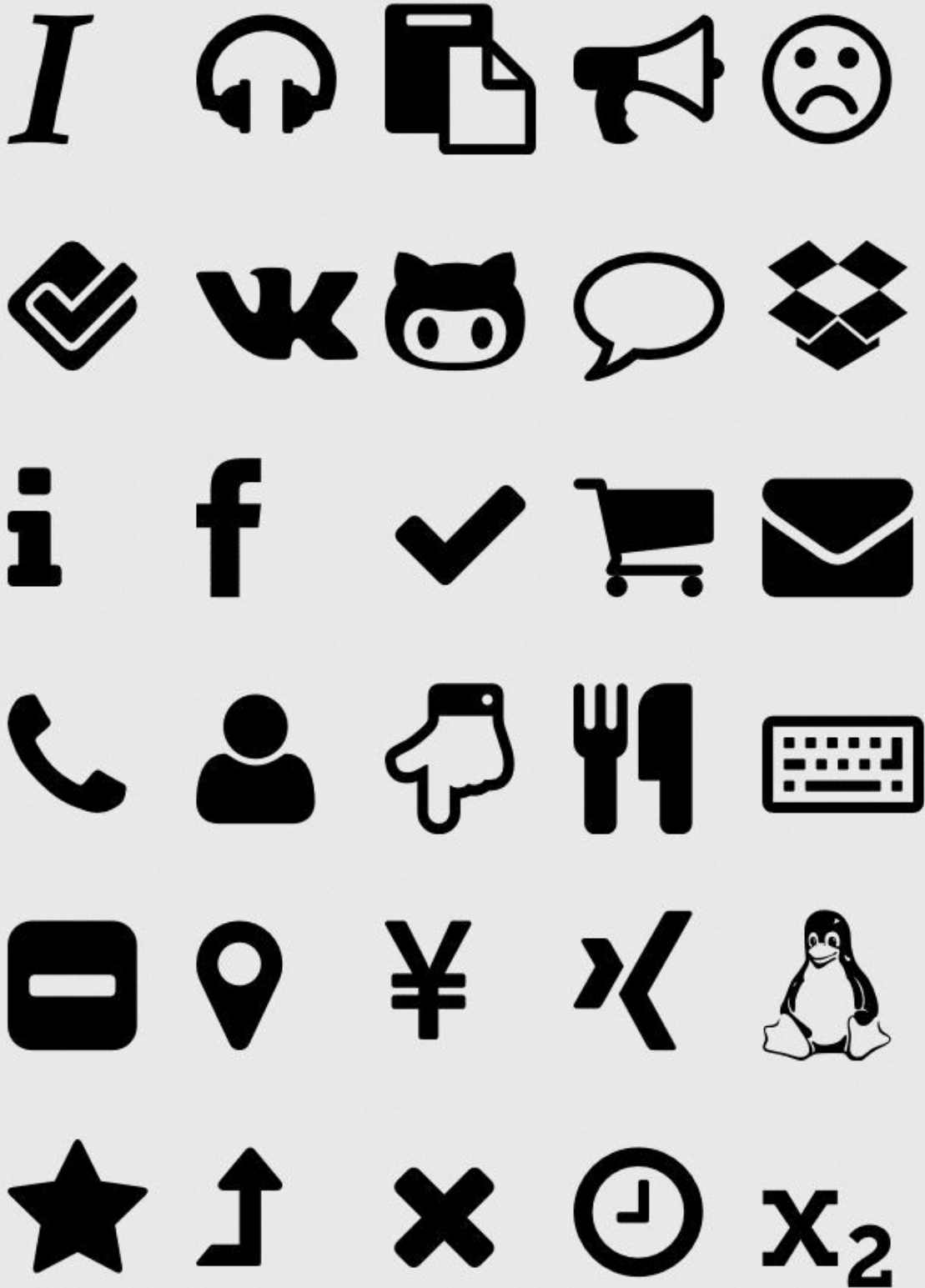
2.0版本更新

- 架构修改，基于MVVM架构
- 把 `SceneModel` 从 `Scene` 中剥离出来，并且加入响应式编程框架 `ReactiveCocoa`
- `ReactiveCocoa` 中文使用说明教程 [ReactiveCocoa2实战](#)
- `ReactiveCocoa` 在github上有开源项目[ReactiveCocoa2](#)

1.0.3版本更新

- 再也不用担心奇葩的图文混排了
- 新增字体图片支持 资源里的demo 就是一个基于 `swift` 和 `easyios` 的字体图片演示,可以用来作为图片字典查阅
- 可扩展的字体库，字需要添加 `ttf` 和 `json` 文件就可以轻松扩展特殊字体
- 目前支持4种图片字体 `FontAwesome` 、 `Zocial-Regular` 、 `Ionicons` 、 `Foundation`
- [FontAwesome 4.1](#) 字体库, 包含 439 个图标
- [Foundation icons](#) 字体库, 包含283 个图标
- [Zocial Contains](#) 字体库, 包含99 个图标
- [ionicons 1.5.2](#) 字体库, 包含601 个图标,大部分是 IOS7 style

FontAwesome



1.0.2版本更新

- fix一些头文件的引用关系，增加了swift头文件支持。
- 用swift的同学，要设置 Objective-C Bridging Header 为 `${PODS_ROOT}/Headers/EasyIOS/swift-bridge.h`
- 1.0.2版本发布到了CocoaPods

1.0.1版本更新

- 1.增加了ORM支持，从此可以实现json、object、sqlite三者之间的一键转换,可以节省很多代码,是不是很酷。

model类整合了Jastor的类库和MojoDataBase类库

- 2.修改了Action类中的post的参数，增加了files参数，

因此，action.POST_MSG的时候现在至少要3个参数哦

- 3.借鉴了beeframework的消息通知机制。。默默给郭大点个赞。。
- 4.修复了一个下拉刷新的bug
- 5.提供了一键打开百度地图、苹果地图、google地图、高德地图发起调用的接口，再也不用担心看地图文档了

Welcome to the EasyIOS wiki!

EasyIOS 教程入口

Coding with Swift

- [RACSwift](#)

基于EasyIOS的上线应用

- [脉冲书志](#)
- [兼职达人](#)

Other Link

- [花瓣网李忠：ReactiveCocoa是Cocoa的未来](#)
- [ReactiveCocoa2实战](#)
- [EasyIOS 2.0 发布，基于 MVVM 的 IOS 开发框架](#)
- [EasyIOS用Model-View-ViewModel构建iOS App](#)
- [iOS Framework: Introducing MKNetworkKit \(MKNetworkKit介绍，入门，翻译\)](#)
- [Swift语言主流学习资源](#)

Best Wishes

如果你觉得EasyIOS好，并且在某个项目里成功地使用了它，请分享您的app到这里，供大家参考学习，一起见证EasyIOS的成长。Thanks!

Tool Collection

- [SwipeView](#)

Horizontal, paged scrolling views based on UIScrollView, with convenient functionality in UITableView-style.

- [StyledPageControl](#)

Customizable PageControl for iOS.

- [synx](#)

A command-line tool that reorganizes your Xcode project folder to match your Xcode groups

Scene Class Reference

Scene 继承UIViewController，EasyIOS里面仅仅负责视图的展示工作。

===

Tasks

parentScene `property`

- – showBarButton:title:fontColor:
- – showBarButton:imageName:
- – showBarButton:button:
- – leftButtonTouch
- – rightButtonTouch
- – setTitleText:
- – setTitleText:hidden:

===

Properties

parentScene

```
@property (nonatomic, retain) Scene *parentScene
```

===

Instance Methods

leftButtonTouch

```
- (void)leftButtonTouch
```

左按钮点击事件。

rightButtonTouch

```
- (void)rightButtonTouch
```

右按钮点击事件。

setTitleText:

```
- (void)setTitleText:(NSString *)str
```

//添加到导航条标题

showBarButton:button:

```
- (void)showBarButton:(EzNavigationBar)position button:(UIButton *)button
```

设置导航条左右按钮，使用方法：

```
[self showBarButton:NAV_RIGHT button:button]; //把某按钮设置为导航右按钮
```

showBarButton:imageName:

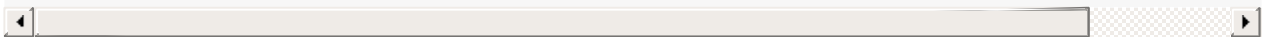
```
- (void)showBarButton:(EzNavigationBar)position imageName:(NSString *)imageName
```

设置导航条左右按钮，使用方法：

```
[self showBarButton:NAV_RIGHT imageName:@"image"]; //把某图片设置为导航右按钮
```

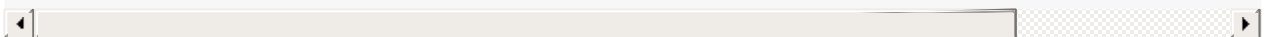
showBarButton:title:fontColor:

```
- (void)showBarButton:(EzNavigationBar)position title:(NSString *)name fontColor:(UIColor
```



设置导航条左右按钮，使用方法：

```
[self showBarButton:NAV_LEFT title:@"left" fontColor:[UIColor blackColor]]; //设置左按钮为1
```



SceneModel Class Reference

SceneModel 负责数据的绑定，控制视图更新，网络请求的发起，以及Model的生成工作

===

Tasks

action `property`

- +SceneModel
- – handleActionMsg:
- – SEND_ACTION:
- – SEND_CACHE_ACTION:
- – SEND_NO_CACHE_ACTION:
- – initModel

===

Properties

action

```
@property (nonatomic, strong) Action *action
```

===

Class Methods

SceneModel

```
+ (id)SceneModel
```

Instance Methods

SEND_ACTION:

```
- (void)SEND_ACTION:(NSDictionary *)dict
```

SEND_CACHE_ACTION:

```
- (void)SEND_CACHE_ACTION:(NSDictionary *)dict
```

SEND_NO_CACHE_ACTION:

```
- (void)SEND_NO_CACHE_ACTION:(NSDictionary *)dict
```

handleActionMsg:

```
- (void)handleActionMsg:(ActionData *)msg
```

initModel

```
- (id)initModel
```

Action Class Reference

Action 负责网络请求的发起，设置请求的成功、失败和错误的状态及对应的回调方法

初始化：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [Action actionConfigHost:@"maichong.08dream.com" client:@"easyIOS" codeKey:@"code" ri
}
```

===

Tasks

action property

- +Action:
- – initWithCache:
- – success:
- – error:
- – failed:
- – Send:

===

Properties

action

```
@property(nonatomic, strong)id<ActionDelegate> aDelegaete;
```

===

Class Methods

SceneModel

```
+ (id)Action;
```

生成一个实例的Action对象

Instance Methods

initWithCache:

```
- (id)initWithCache;
```

初始化对象，并且默认使用缓存

success:

```
- (void)success:(Request *)msg;
```

设置请求的状态为成功，并且触发对应的回调

error:

```
- (void)error:(Request *)msg;
```

设置请求的状态为失败，并且触发对应的回调

failed:

```
- (void)failed:(Request *)msg;
```

设置请求的状态为失败，并且触发对应的回调

Send

```
-(MKNetworkOperation*)Send:(Request *) msg;
```

发送请求，会根据msg中配置的方式（GET或POST）和参数发起相应的请求操作，请求被加入请求队列中。

Request

Request为初始化网络请求数据的类

GET

如果你想发这样一个GET请求 `http://test-leway.zjseek.com.cn:8000/api/goods/goodsList?type=1&pageSize=10&page=1&categoryId=-1&areaName=浙江`

你可以这样定义一个GoodsListRequest

```
#import "Request.h"
@interface GoodsListRequest : Request
@property(nonatomic,retain) NSString *type;
@property(nonatomic,retain) NSNumber *page;
@property(nonatomic,retain) NSNumber *pageSize;
@property(nonatomic,retain) NSString *categoryId;
@property(nonatomic,retain) NSString *areaName;
@end

#import "GoodsListRequest.h"
@implementation GoodsListRequest
/**
 * 初始化 GoodsListRequest 的参数
 */
-(void)loadRequest{
    [super loadRequest];
    //self.HOST = @"test-leway.zjseek.com.cn:8000"; 不设置host 就采用宏定义的全局host
    self.PATH = @"/api/goods/goodsList";
    self.METHOD = @"GET";
    self.type = @"1";
    self.categoryId = @"-1";
    self.areaName = @"浙江";
    self.pageSize = @10;
}
@end
```

loadRequest方法里面初始化HOST, PATH, METHOD以及各种变量。EasyIOS会自动进行拼接为`http://test-lewa

POST

如果你想POST文件

你可以这样定义一个ImagePostRequest

```
#import "Request.h"
@interface ImagePostRequest : Request
@property(nonatomic,retain) NSString *name;
@end

#import "ImagePostRequest.h"
@implementation ImagePostRequest
```



```

/**
 * 初始化 GoodsListRequest 的参数
 */
-(void)loadRequest{
    [super loadRequest];
    //self.HOST = @"test-leway.zjseek.com.cn:8000"; 不设置host 就采用宏定义的全局host
    self.PATH = @"/api/goods/goodsList";
    self.METHOD = @"POST"; //设置为post
    self.name = @"图片";
    NSString *localPath1 = [NSString stringWithFormat:@"%s/%s", $.documentPath, @"face1"];
    NSString *localPath2 = [NSString stringWithFormat:@"%s/%s", $.documentPath, @"face2"];
    self.requestFiles = @{@"image1":[NSURL URLWithStringPath:localPath1],
                          @"image2":[NSURL URLWithStringPath:localPath2]};
}
@end

```

requestFiles方法里面返回一个文件字典。EasyIOS会自动进行识别上传

激活请求

如果你想快速的激活一个Request

```
self.sceneModel.request.requestNeedActive = YES;
```

一个在SceneModel中使用的例子

```

-(void)loadSceneModel{
    [super loadSceneModel];
    [self.action useCache];
    self.userData = nil;

    @weakify(self);
    _request = [UserRequest RequestWithBlock:^(
        @strongify(self);
        if(self.request.userId.isEmpty){
            [self SEND_IQ_ACTION:self.request];
        }
    )];

    [[RACObserve(self.request, state) //监控 网络请求的状态
     filter:^(NSNumber *state) {
         @strongify(self);
         return self.request.succeed;
     }]
     subscribeNext:^(NSNumber *state) {
         @strongify(self);
         NSError *error;
         self.userData = [[UserEntity alloc] initWithDictionary:[self.request.output obje
     ]];
    }];
}

```

一、Model的对象映射操作

Model作为数据核心，功能不强大是不行的！

Examples

- a.你有可能有这样的 JSON:

```
{
    "name": "Foo",
    "age": 13
}
```

- b.或者还有这样的模型类:

```
#import "Friend.h"
@interface Friend : Model
@property (nonatomic, retain) NSString *name;
@property (nonatomic, retain) NSNumber *age;
@end
```

- c.利用Model的对象映射操作，就可以实现一键的转换：

```
// Some other code
NSString *json = /* json form net */;
Friend *friend = [[Friend alloc] initWithString:json error:&err];

// Log
friend.name // => Foo
friend.age // => 13
```

还有更酷的！

- a.你有可能还有这样的 JSON:

```
{
    "name": "Foo",
    "category": {
        "name": "Bar Category"
    }
}
```

- b.或者还有这样的模型类:

```
#import "FriendCategory.h"
@interface FriendCategory : Model
@property (nonatomic, retain) NSString *name;
@end
```

- c.利用Model你可以这样：

```
#import "Friend.h"
#import "FriendCategory.h"

@interface Friend : Model
@property (nonatomic, retain) NSString *name;
@property (nonatomic, retain) FriendCategory *category;
@end
```

- d.同样可以一键转换

```
// Code
NSString *json = /* json form net */;
Friend *friend = [[Friend alloc] initWithString:json error:&err];

// Log
friend.name // => Foo
friend.category // => <FriendCategory>
friend.category.name // => Bar Category
```

遇上Array怎么办？

- a.你有可能还有这样的 JSON:

```
// JSON
{
    "name": "Foo",
    "categories": [
        { "name": "Bar Category 1" },
        { "name": "Bar Category 2" },
        { "name": "Bar Category 3" }
    ]
}
```

- b.利用Model你可以这样：

```
#import "Friend.h"
#import "FriendCategory.h"

@protocol FriendCategory
@end

@interface Friend : Model
@property (nonatomic, copy) NSString *name;
@property (nonatomic, retain) NSArray<FriendCategory> *categories;
@end
```

- c.同样可以一键转换

```
// Code
NSString *json = /* json form net */;
```

```

Friend *friend = [[Friend alloc] initWithString:json error:&err];

// Log
friend.name // => Foo
friend.categories // => <NSArray>
[friend.categories count] // => 3
[friend.categories objectAtIndex:1] // => <ProductCategory>
[[friend.categories objectAtIndex:1] name] // => Bar Category 2

```

然后你就可以乱来了。。。。

```

// JSON
{
  "name": "1",
  "children": [
    { "name": "1.1" },
    { "name": "1.2",
      children: [
        { "name": "1.2.1",
          children: [
            { "name": "1.2.1.1" },
            { "name": "1.2.1.2" },
          ]
        },
        { "name": "1.2.2" },
      ]
    },
    { "name": "1.3" }
  ]
}

```

还没完呢！！！！

- a.擦，这个json不是驼峰！强迫症！服务器没节操！

```

{
  "first_name": "John",
  "last_name": "Doe"
}

```

- b.你可以这样

```

// Person.h
@interface Person : Model
@property (nonatomic, copy) NSString *firstName;
@property (nonatomic, copy) NSString *lastName;
@end

// Person.m
@implementation Person

+(JSONKeyMapper*)keyMapper
{
    return [[JSONKeyMapper alloc] initWithDictionary:@{

```

```

        @"user.first_name": @"firstName",
        @"user.last_name": @"lastName"
    }];
}

@end

```

- c.根本停不下来

```

// Code
NSDictionary *dictionary = /* parse the JSON response to a dictionary */;
Person *person = [[Person alloc] initWithDictionary:dictionary];

// Log
person.firstName // => John
person.lastName // => Doe

```

服务器有时候给了null

- a.服务器有时候给了null

```

{
    "id": "123",
    "name": null,
    "price": 12.95
}

```

或者干脆不给了。

```

{
    "id": "123",
    "price": 12.95
}

```

- b.你可以这样设置Optional

```

@interface ProductModel : JSONModel
@property (assign, nonatomic) int id;
@property (strong, nonatomic) NSString<Optional>* name;
@property (assign, nonatomic) float price;
@property (strong, nonatomic) NSNumber<Optional>* uuid;
@end

@implementation ProductModel
@end

```

- c.全部设为Optional

```

@implementation ProductModel
+(BOOL)propertyIsOptional:(NSString*)propertyName
{

```

```
    return YES;
}
@end
```

服务器有时候给了用不到的

- a.服务器有时候给多了。

```
{
  "id": "123",
  "a": "123",
  "b": "123",
  "c": "123"
}
```

- b.你可以这样设置Ignore,来减少客户端工作量

```
@interface ProductModel : JSONModel
@property (assign, nonatomic) int id;
@property (strong, nonatomic) NSString<Ignore>* a;
@property (strong, nonatomic) NSString<Ignore>* b;
@property (strong, nonatomic) NSString<Ignore>* c;
@end

@implementation ProductModel
@end
```

如果你更懒一点.h都不想写。。。。

- 这里有个[懒人工具ModelCoder](#)

Using the built-in thin HTTP client

```
//add extra headers
[[JSONHTTPClient requestHeaders] setValue:@"MySecret" forKey:@"AuthorizationToken"];

//make post, get requests
[JSONHTTPClient postJSONFromURLWithString:@"http://mydomain.com/api"
                 params:@{@"postParam1":@"value1"}
                 completion:^(id json, JSONModelError *err) {

    //check err, process json ...

}];
```

Export model to NSDictionary or to JSON text

```
ProductModel* pm = [[ProductModel alloc] initWithString:jsonString error:nil];
pm.name = @"Changed Name";
```

```
//convert to dictionary
NSDictionary* dict = [pm toDictionary];

//convert to text
NSString* string = [pm toJSONString];
```

二、Model的DataDase数据库操作

1.创建一个数据库并连接

- (1).在应用的 AppDelegate.h 文件声明一个数据库实例：

```
@class MojoDatabase;

@interface AppDelegate : NSObject <UIApplicationDelegate> {
    MojoDatabase *database;
}

@property (nonatomic, retain) MojoDatabase *database;

@end
```

- (2).在代理方法 didFinishLaunchingWithOptions 里初始化这个数据库：

```
#import "AppDatabase.h"

@implementation AppDelegate

-(BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Setup Connection to the database
    self.database = [[AppDatabase alloc] initWithMigrations];
}

@end
```

这样就会初始化（如果不存在就会自动创建）一个数据库，如果你想自定义数据库的名字可以在`AppDatabase.m`文件

2.创建一个数据模型

- (1).每一个数据模型都需要在数据库里面有对应的数据表，数据表当你需要用到数据库的时候model会自动创建
- (2).你还需要有一个和数据表名称一致的数据模型Model的子类Friend：

```
#import "Model.h"

@interface Friend : Model
@property (nonatomic, retain) NSString *name;
@property (nonatomic, retain) NSNumber *age;
@end
```

- (3).然后你就可以像这样轻松的插入一条数据到Friend数据表了：

```
Friend *newFriend = [[Friend alloc] init];
newFriend.name = @"Norman Rockwell";
newFriend.age = [NSNumber numberWithInt:55];
[newFriend save]; // this will write out the new friend record to the database
```

- (4).从Friend表中查询数据，同样也很Easy的啦：

```
NSArray *records = [self findByColumn:@"name" value:@"Norman Rockwell"];
if ( [records count] ) {
    Friend *normanRockwell = [records objectAtIndex:0];
}
```

备注：findByColumn 总是返回一个NSArray对象。

3.提供的api列表（顾名思义，就不一一说明了）

```
+(void)setDatabase:(MojoDatabase *)newDatabase;
+(MojoDatabase *)database;
-(void)resetAll;
-(MojoModel*)table:(NSString *)table;
-(MojoModel*)field:(id)field;
-(MojoModel*)limit:(NSUInteger)start size:(NSUInteger)size;
-(MojoModel*)order:(NSString *)order;
-(MojoModel*)group:(NSString *)group;
-(MojoModel*)where:(NSDictionary *)map;
-(NSArray *)select;
-(NSUInteger)getCount;

-(void)update:(NSDictionary *)data;
-(void)beforeUpdate:(NSDictionary *)data;
-(void)afterUpdate:(NSDictionary *)data;

-(void)save;
-(void)beforeSave;
-(void)afterSave;

-(void)deleteSelf;
-(void)beforeDeleteSelf;
-(void)afterDeleteSelf;

-(void)delete;
-(void)beforeDelete;
-(void)afterDelete;
```

```
+(void)afterFind:(NSArray **)results;
+(void)beforeFindSql:(NSString **)sql parameters:(NSArray **)parameters;
+(NSArray *)findWithSql:(NSString *)sql withParameters:(NSArray *)parameters;
+(NSArray *)findWithSqlWithParameters:(NSString *)sql, ...;
+(NSArray *)findWithSql:(NSString *)sql;
+(NSArray *)findByColumn:(NSString *)column value:(id)value;
+(NSArray *)findByColumn:(NSString *)column unsignedIntegerValue:(NSUInteger)value;
```



```
+(NSArray *)findByColumn:(NSString *)column integerValue:(NSInteger)value;  
+(NSArray *)findByColumn:(NSString *)column doubleValue:(double)value;  
+(id)find:(NSUInteger)primaryKey;  
+(NSArray *)findAll;  
+(void)deleteAll;  
-(BOOL)isTableExist;  
-(void)createTable;
```

为什么RAC更加适合编写Cocoa App？说这个之前，我们先来看下Web前端编程，因为有些相似之处。目前很火的AngularJS有一个很重要的特性：数据与视图绑定。就是当数据变化时，视图不需要额外的处理，便可正确地呈现最新的数据。而这也是RAC的亮点之一。RAC与Cocoa的编程模式，有点像AngularJS和jQuery。所以要了解RAC，需要先在观念上做调整。

- 1.RAC进行数据绑定KVO的例子：

```
// When self.username changes, logs the new name to the console.
//
// RACObserve(self, username) creates a new RACSignal that sends the current
// value of self.username, then the new value whenever it changes.
// -subscribeNext: will execute the block whenever the signal sends a value.
[RACObserve(self, username) subscribeNext:^(NSString *newName) {
    NSLog(@"%@", newName);
}];
```

当每次self.username改变，都会自动执行subscribeNext，打出内容到控制台

- 2.RAC不仅仅是kvo，它还可以对信号进行过滤，筛选正确的信号，例如：

```
// Only logs names that starts with "j".
//
// -filter returns a new RACSignal that only sends a new value when its block
// returns YES.
[[RACObserve(self, username)
    filter:^(NSString *newName) {
        return [newName hasPrefix:@"j"];
    }]
    subscribeNext:^(NSString *newName) {
        NSLog(@"%@", newName);
    }];
```

筛选前缀是j的username，打出到控制台

Swift 与 Objective-C 中的使用区别：

设置请看 [RACSwift](#)

RAC In Objective-C：

```
RAC(self.collectionView,page) = RACObserve(self.collectionView,page);
```

In Swift：

```
RAC(self.collectionView,"page") <= RACObserve(self.collectionView,"page")
RACObserve(self.collectionView,"page") => RAC(self.collectionView,"page")
```

RACObserve In Objective-C：

```

[[[RACObserve(self.collectionView, page)
  map:^(NSString*(NSNumber* newPage) {
    return @"123";
  }]
  filter:^(BOOL(NSString* newPage) {
    return false;
  }]
  subscribeNext:^(NSString* text) {
    NSLog(@"%@",text);
  }]];

```

In Swift :

```

RACObserve(self.collectionView, "page")
.mapAs{
    (newpage:NSNumber) -> NSString in
    return "123"
}
.filterAs{
    (newpage:NSString) -> Bool in
    return false
}
.subscribeNextAs{
    (text:String) -> () in
    println(text)
}

```

AutoLayoutCell

我们在实际开发中肯定遇到过自适应Cell的需求

在iOS6或者更早以前

在iOS6或者更早以前，大都是通过程序计算出cell得高度，在进行渲染。可是实际开发中使用 UITableView，我们发现，UITableView 是先请求Cell的高度再去渲染视图，这就需要在请求高度的时候模拟渲染获得高度，或者根据图片大小或者字符串长度计算出Cell的最小高度。这样就导致了计算不准确等问题。

在iOS8

iOS8中, UITableView 有了self sizing cells 这样一个功能来解决动态Cell的问题。

为了兼容iOS7，并且更简单的实现动态Cell，EasyIOS封装了基于Autolayout的动态Cell布局的功能。

下面是一个例子：

- 首先我们需要声明一个ALTableView

```
self.tableView = [[ALTableView alloc] init];
self.tableView.separatorStyle = UITableViewCellSeparatorStyleNone;
self.tableView.delegate = self;
self.tableView.dataSource = self;
self.tableView.cellFactory.delegate = self;
self.tableView.estimatedRowHeight = 250.0f;
self.tableView.rowHeight = UITableViewAutomaticDimension;
```

estimatedRowHeight 如果不设置tableView会去调用 heightForRowAtIndexPath ,所以为了减轻渲染的负担，我们在这里提供一个大致值。(应该也是苹果团队的一种补救措施)。

- 实现 UITableViewDataSource

```
- (CGFloat)tableView:(ALTableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {
    return [tableView.cellFactory cellHeightForIdentifier:@"RssCell" forIndexPath:indexPath];
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return self.rssSceneModel.dataArray.count;
}

- (UITableViewCell *)tableView:(ALTableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    return [tableView.cellFactory cellWithIdentifier:@"RssCell" forIndexPath:indexPath];
}
```

- 实现 ALTableViewCellFactoryDelegate 代理

```
- (void)tableView:(UITableView *)tableView configureCell:(RssCell *)cell forIndexPath:indexPath {
    cell.delegate = self;
    [cell reloadRss:[self.rssSceneModel.dataArray objectAtIndex:indexPath.row]]
}
```

```
        with:self.rssSceneModel.rssList.feed];  
    }  
}
```

这里仅仅是导入需要渲染的数据

- 声明一个 `ALBaseCell` 的子类

```
@interface RssCell : ALBaseCell  
@end
```

- `ALBaseCell` 子类的初始化入口

```
-(void)commonInit{  
    [super commonInit];  
    //在这里添加需要的控件到self.contentView  
    [self.contentView addSubview:_feedTitle];  
    [self loadAutolayout];  
}  
-(void)loadAutolayout{  
    //这里使用AutoLayout对Cell进行布局  
}
```

当然上面这些，都可以在xib中完成:)，特别要注意最底部的控件 要设置他与bottom的关系，否则 `ContentView`无法知道他的最小高度比如

```
[_lastView alignBottom:@"-5" toView:self.contentView];
```

- OK,现在就可以实现动态的Cell了，具体实现可以参见[EasyRSS](#)这个项目

SVPullToRefresh + SVInfiniteScrolling

1.addPullToRefreshWithActionHandler

```
[self.tableView addPullToRefreshWithActionHandler:^(  
    self.sceneModel.request.page = @1;  
    self.sceneModel.active = YES;  
});
```

2.addInfiniteScrollingWithActionHandler

```
[self.tableView addInfiniteScrollingWithActionHandler:^(  
    self.sceneModel.request.page = [self.sceneModel.request.page increase:@1];  
    self.sceneModel.active = YES;  
});
```

3.触发刷新

```
[self.tableView triggerPullToRefresh];  
[self.tableView triggerInfiniteScrolling];
```

4.停止所有刷新动作

```
[tableView endAllRefreshing];
```

5.自动分页（需要Pagination）类

```
Pagination *pagination = [Pagination Model];  
pagination.page = @2;  
pagination.pageSize = @10;  
pagination.total = @19;  
pagination.isEnd = @1;  
  
self.tableview.pagination = pagination;  
  
self.dataArray = [pagination  
    success:self.dataArray  
    newArray:self.sceneModel.list];  
  
[self.tableView endAllRefreshing];
```

文档正在完善。。。

文档正在完善。。。。

EasyKit Class Reference

EasyKit 里面有很多实用便捷的小功能。☺ 统一用\$符号调用

- (NSString *)homePath;//应用程序目录的路径，在该目录下有三个文件夹：Documents、Library、temp以及一个.app包！该目录下就是应用程序的沙盒，应用程序只能访问该目录下的文件夹！！
 - (NSString *)desktopPath;//数据所存桌面的绝对路径
 - (NSString *)documentPath;// 文档目录，需要ITUNES同步备份的数据存这里
 - (NSString *)libPrePath; // 配置目录，配置文件存这里
 - (NSString *)libCachePath; // 缓存目录，系统永远不会删除这里的文件，ITUNES会删除
 - (NSString *)appPath; // .app 程序相对目录，不能存任何东西
 - (NSString *)tmpPath; // 缓存目录，APP退出后，系统可能会删除这里的内容
 - (NSString *)resourcePath; // .app 程序绝对目录，不能存任何东西
 - (BOOL)touchPath:(NSString *)path;//创建路径
 - (BOOL)touchFile:(NSString *)file;//创建文件
-
- (RACSignal*) rac_didNetworkChanges （rac监控网络状态的改变。）

使用示例：

```
[[ $ rac_didNetworkChanges
  subscribeNext:^(Reachability* reach) {
    if([reach isReachableViaWiFi]){
      EZLog(@"正在使用Wi-Fi网络");
    }else if ([reach isReachableViaWWAN]){
      EZLog(@"正在使用移动数据网络");
    }else{
      EZLog(@"网络连接已断开");
    }
  }
  ]];
```

- 生成一个图形按钮方法

```
NSString* icon = [UIFont icon:@"fa_angle_down" fromFont:fontAwesome]; // 获取字符  
UIButton *button = [UIFont buttonWithIcon:icon fontName:fontAwesome size:15.0f co
```



字体名称 `fa_angle_down` 可以在demo字典中查找到：目前提供了4种图片字体下面是他们的宏定义：

```
#define fontAwesome @"FontAwesome"  
#define zocialRegularWebfont @"Zocial-Regular"  
#define ionIcons @"Ionicons"  
#define foundationIcons @"fontcustom"
```

Objective-C RegEx Categories

Overview

This project is a collection of objective-c categories for `NSRegularExpression` and `NSString` that make usage easier and more concise. For example:

```
//Using NSRegularExpression
NSString* string = @"I have 2 dogs.";
NSRegularExpression *regex = [NSRegularExpression regular ExpressionWithPattern:@"\\d+" o
NSTextCheckingResult *match = [regex firstMatchInString:string options:0 range:NSMakeRangerang
BOOL isMatch = match != nil;

// Using this library
BOOL isMatch = [@"I have 2 dogs." isMatch:RX(@"\\d+")];
```



Getting Started

This library has no dependencies and works for iOS4+ and OSX v10.7+.

To install it, just copy these two files into your project:

- RegExCategories.h
- RegExCategories.m

You may want to add it to your Prefix.pch so that it is available across your code base.

```
#ifdef __OBJC__
/* ...other references... */
#import "RegExCategories.h"
#endif
```

You also need to have [ARC](#) enabled on your XCode project. If you don't then add the `-fobjc-arc` flag on `Objective-C-Regex-Categories.m` under Targets > Build Phases > Compile Sources ([more info here](#)).

Examples

Support

If you need help, [submit an issue](#), [send a pull request](#), or tweet at me [@BendyTree](#).

Licensing

[MIT License](#) - do whatever you want, just (1) provide attribution and (2) don't hold me liable.

Travis-CI

This repository includes unit tests written in the [XCTest](#) framework. These test are automatically verified using [Travis-CI](#). Here is the current status:

build failing

TMCache

Fast parallel object cache for iOS and OS X.

[TMCache](#) is a key/value store designed for persisting temporary objects that are expensive to reproduce, such as downloaded data or the results of slow processing. It is comprised of two self-similar stores, one in memory ([TMMemoryCache](#)) and one on disk ([TMDiskCache](#)), all backed by GCD and safe to access from multiple threads simultaneously. On iOS, `TMMemoryCache` will clear itself when the app receives a memory warning or goes into the background. Objects stored in `TMDiskCache` remain until you trim the cache yourself, either manually or by setting a byte or age limit.

`TMCache` and `TMDiskCache` accept any object conforming to [NSCoding](#). Put things in like this:

```
UIImage *img = [[UIImage alloc] initWithData:data scale:[UIScreen mainScreen] scale]];
[[TMCache sharedCache] setObject:img forKey:@"image" block:nil]; // returns immediately
```

Get them back out like this:

```
[[TMCache sharedCache] objectForKey:@"image"
    block:^(TMCache *cache, NSString *key, id object) {
        UIImage *image = (UIImage *)object;
        NSLog(@"image scale: %f", image.scale);
    }];
```

`TMMemoryCache` allows for concurrent reads and serialized writes, while `TMDiskCache` serializes disk access across all instances in the app to increase performance and prevent file contention. `TMCache` coordinates them so that objects added to memory are available immediately to other threads while being written to disk safely in the background. Both caches are public properties of `TMCache`, so it's easy to manipulate one or the other separately if necessary.

Collections work too. Thanks to the magic of `NSKeyedArchiver`, objects repeated in a collection only occupy the space of one on disk:

```
NSArray *images = @[ image, image, image ];
[[TMCache sharedCache] setObject:images forKey:@"images"];
NSLog(@"3 for the price of 1: %d", [[[TMCache sharedCache] diskCache] byteCount]);
```

Installation

Manually

[Download the latest tag](#) and drag the `TMCache` folder into your Xcode project.

Install the docs by double clicking the `.docset` file under `docs/`, or view them online at cocoadocs.org

Git Submodule

```
git submodule add https://github.com/tumblr/TMCache.git
git submodule update --init
```

CocoaPods

Add [TMCache](#) to your `Podfile` and run `pod install`.

Build Status

build **passing**

Requirements

TMCache requires iOS 5.0 or OS X 10.7 and greater.

Contact

[Justin Ouellette](#)

License

Copyright 2013 Tumblr, Inc.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. [See the License](#) for the specific language governing permissions and limitations under the License.

GCDObjC

GCDObjC is an Objective-C wrapper for the most commonly used features of Grand Central Dispatch. It has four main aims:

- Organize the flat C API into appropriate classes.
- Use intention-revealing names to distinguish between synchronous and asynchronous functions.
- Use more convenient arguments such as `NSTimeIntervals`.
- Add convenience methods.

Usage

GCDObjC requires ARC and iOS 6.0. (Prior to 6.0, dispatch objects were not considered Objective-C objects, and therefore required manual memory management.)

GCDObjC.h is the only header file that needs to be imported.

For usage examples, see [GCDObjCTests.m](#).

GCDQueue

Queues are implemented in the **GCDQueue** class.

- convenience accessors for global queues

```
+ (GCDQueue *)mainQueue;  
+ (GCDQueue *)globalQueue;  
+ (GCDQueue *)highPriorityGlobalQueue;  
+ (GCDQueue *)lowPriorityGlobalQueue;  
+ (GCDQueue *)backgroundPriorityGlobalQueue;
```

- creating serial and concurrent queues

```
- (instancetype)initSerial;  
- (instancetype)initConcurrent;
```

- queueing blocks for asynchronous execution

```
- (void)queueBlock:(dispatch_block_t)block;  
- (void)queueBlock:(dispatch_block_t)block afterDelay:(double)seconds;  
- (void)queueBlock:(dispatch_block_t)block inGroup:(GCDGroup *)group;
```

- queueing blocks for synchronous execution

```
- (void)queueAndWaitBlock:(dispatch_block_t)block;  
- (void)queueAndWaitBlock:(void (^)(size_t))block iterationCount:(size_t)count;
```

- queueing barrier blocks for synchronous or asynchronous execution

```
- (void)queueBarrierBlock:(dispatch_block_t)block;  
- (void)queueAndAwaitBarrierBlock:(dispatch_block_t)block;
```

- queueing notify blocks on groups

```
- (void)queueNotifyBlock:(dispatch_block_t)block inGroup:(GCDGroup *)group;
```

- suspending and resuming a queue

```
- (void)suspend;  
- (void)resume;
```

GCDSemaphore

Semaphores are implemented in the **GCDSemaphore** class.

- creating semaphores

```
- (instancetype)init;  
- (instancetype)initWithValue:(long)value;
```

- signaling and waiting on a semaphore

```
- (BOOL)signal;  
- (void)wait;  
- (BOOL)wait:(double)seconds;
```

GCDGroup

Groups are implemented in the **GCDGroup** class.

- creating groups

```
- (instancetype)init;
```

- entering and leaving a group

```
- (void)enter;  
- (void)leave;
```

- waiting on completion of a group

- (void)wait;
- (BOOL)wait:(double)seconds;

Macros

Two macros are provided for wrapping **dispatch_once()** calls.

- executing a block only once: **GCDExecOnce(block)**

```
for (int i = 0; i < 10; ++i) {  
    GCDExecOnce(^{ NSLog(@"This will only be logged once."); });  
}
```

- creating a singleton instance of a class: **GCDSharedInstance(block)**

```
+ (instancetype)sharedInstance {  
    GCDSharedInstance(^{ return [[self class] new]; });  
}
```

The block supplied to **GCDSharedInstance()** must return an instance of the desired class.

FLKAutoLayout

FLKAutoLayout is a category on `UIView` which makes it easy to setup layout constraints in code.

FLKAutoLayout creates simple constraints with a readable syntax and provides many convenience methods to setup more complex constraints between multiple views at once. It automatically adds the constraints to the nearest common superview of the views involved and sets the `translatesAutoresizingMaskIntoConstraints` property on those views to `NO`.

FLKAutoLayout provides methods on `UIView` *instances* for simple layout constraints like width and height or constraining an edge of one view to another. Furthermore it provides methods on the `UIView` *class* for more complex layout constraints where more than two views are involved.

For some examples of how to setup constraints please check the [example project](#).

Example

Let's assume we have a bunch of labels and an equal amount of textFields and we want to align them nicely in a grid like manner:

```
// align the first label with its superview
[labels[0] alignTop:@"20" leading:@"20" toView:labels[0].superview];
// give it a minimum width of 200 and a maximum width of 300
[labels[0] constrainWidth:@">=200,<=300"];
// now constrain all labels to this size
[UIView alignLeadingAndTrailingEdgesOfViews:labels];
// space the labels out vertically with 10 points in between
[UIView spaceOutViewsVertically:labels predicate:@"10"];

// now let's take care of the text fields.
// the first one has a fixed space of 20 to its label
[textFields[0] constrainLeadingSpaceToView:labels[0] predicate:@"20"];
// constrain the right edge to its superview with 20 points padding
[textFields[0] alignTrailingEdgeWithView:textFields[0].superview predicate:@"20"];
// constrain all other text fields to the same width
[UIView alignLeadingAndTrailingEdgesOfViews:textFields];
// and finally let's align the baseline of each label with the baseline of each text field
[UIView alignAttribute:NSLayoutAttributeBaseline ofViews:labels toViews:textFields predic
```

FLKAutoLayout instance methods

FLKAutoLayout extends `UIView` instances with methods to setup simple constraints in a readable form.

Aligning edges of one view to another:

```
// constrain the leading edge of the view to the leading edge of another
[view alignLeadingEdgeWithView:otherView predicate:nil];

// same as before but use a 20 point offset
[view alignLeadingEdgeWithView:otherView predicate:@"20"];
```

```
// same as before but give this constraint a priority of 750
[view alignLeadingEdgeWithView:otherView predicate:@"20@750"];

// aligning some other edge types
[view alignTopEdgeWithView:otherView predicate:nil];
[view alignBottomEdgeWithView:otherView predicate:nil];
[view alignTrailingEdgeWithView:otherView predicate:nil];
[view alignBaselineWithView:otherView predicate:nil];
[view alignCenterXWithView:otherView predicate:nil];
[view alignCenterYWithView:otherView predicate:nil];

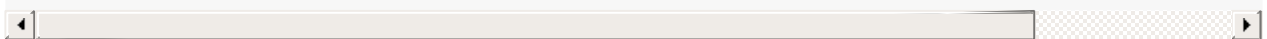
// centering two views
[view alignCenterWithView:otherView];
```

Constraining width & height:

```
[view constrainWidth:@"400"];
[view constrainHeight:@"300"];
// or combined:
[view constrainWidth:@"400" height:@"300"];
// or relative to another view
[view constrainWidthToView:otherView predicate:@"*.3"]; // 30% of otherView's width
[view constrainHeightToView:otherView predicate:@">*.5"]; // 50% of otherView's width
```

Spacing out two views:

```
// creating a >=20 points space between the top edge of one view to the bottom edge of th
[view constrainTopSpaceToView:otherView predicate:@">=20"];
// creating a >=20 points space between the leading edge of one view to the trailing edge
[view constrainLeadingSpaceToView:otherView predicate:@">=20"];
```



If you need more control over which layout attribute of one view should be constrained to which layout attribute of another view, you can use a generic helper method:

```
[view alignAttribute:NSLayoutAttributeCenterX to Attribute:NSLayoutAttributeTrailing ofVi
```



FLKAutoLayout class methods

For laying out more than two views at once FLKAutoLayout provides extends UIView with some class methods.

Align multiple views at once:

```
// align all views in the views array along their leading edge
[UIView alignLeadingEdgesOfViews:views];
// align all views in the views array along their bottom edge
[UIView alignBottomEdgesOfViews:views];
// see UIView+FLKAutoLayout.h for more...
```

Constrain width and height of multiple views:

```
// constrain all views to the same height
[UIView equalHeightForViews:views];
// constrain all views to the same width
[UIView equalWidthForViews:views];
```

Spacing out multiple views:

```
// space out views horizontally with 20 points in between
[UIView spaceOutViewsHorizontally:views predicate:@"20"];
// space out views vertically with no space in between
[UIView spaceOutViewsVertically:views predicate:nil];

// Distribute views according to their horizontal center
[UIView distributeCenterXOfViews:views inView:containerView];
// Distribute views according to their vertical center
[UIView distributeCenterYOfViews:views inView:containerView];
```

Please note that distributing views at their centers will line up the center of the first view at the edge of the container view.

The predicate argument

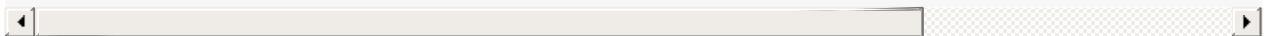
Many of the methods take a predicate string which resembles the syntax Apple uses in their visual format language, extended by the possibility to also specify a multiplier. A nil predicate is the same as @"0".

[== | >= | <=] [*multiplier] [constant] [@priority], ...

For example:

```
// greater than or equal to 300points, small then 500 points
[view constrainWidth:@">=300,<=500"];
// equal to 300 points
[view constrainWidth:@"300"];
// greater than or equal to 300 points with priority of 250
[view constrainWidth:@">=300@250"];

// greater than or equal to 1/2 of the otherView width
[view constrainWidthToView:otherView predicate:@">=*.5"];
// greater than or equal to 1/2 of the otherView width, smaller than or equal to 600 poin
[view constrainWidthToView:otherView predicate:@">=*.5,<=600@100"];
```



Creator

Florian Kugler (@floriankugler).

License

FLKAutoLayout is available under the MIT license. See the LICENSE file for more info.

UICKeyChainStore license MIT

pod v2.0.4 platform ios | osx build passing analytics GA

UICKeyChainStore is a simple wrapper for Keychain on iOS and OS X. Makes using Keychain APIs as easy asNSUserDefaults.

Installation

CocoaPods

```
pod 'UICKeyChainStore'
```

Manual Install

1. Add `Security.framework` to your target.
2. Copy files in Lib (`UICKeyChainStore.h` and `UICKeyChainStore.m`) to your project.

Usage

Using convenient class methods

Add items using default service name (=bundle identifier).

```
[UICKeyChainStore setString:@"kishikawakatsumi" forKey:@"username"];
[UICKeyChainStore setString:@"password1234" forKey:@"password"];

//=> ["username" = "kishikawakatsumi", "password" = "password1234"]
```

Or specify the service name.

```
[UICKeyChainStore setString:@"kishikawakatsumi" forKey:@"username" service:@"com.kishikawakatsumi"];
[UICKeyChainStore setString:@"password1234" forKey:@"password" service:@"com.kishikawakatsumi"];
```

Remove items.

```
[UICKeyChainStore removeItemForKey:@"username" service:@"com.kishikawakatsumi"];
[UICKeyChainStore removeItemForKey:@"password" service:@"com.kishikawakatsumi"];
```

=====


Using store object, easier to edit multiple items

Instantiate store object with default service name.

```
UICKeyChainStore *store = [UICKeyChainStore keyChainStore];
```

Or specify the service name.

```
UICKeyChainStore *store = [UICKeyChainStore keyChainStoreWithService:@"com.kishikawakatsu
```



Add items and save.

```
[store setString:@"kishikawakatsumi@mac.com" forKey:@"username"];  
[store setString:@"password1234" forKey:@"password"];  
  
[store synchronize]; // Write to keychain.
```


Remove items.

```
[store removeItemForKey:@"username"];  
[store removeItemForKey:@"password"];  
  
[store synchronize]; // Write to keychain.
```

=====

Object Subscribing

```
UICKeyChainStore *store = [UICKeyChainStore keyChainStoreWithService:@"com.kishikawakatsu
```

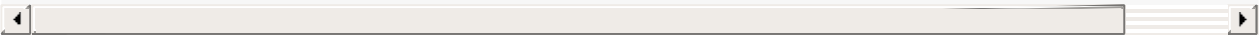


```
store[@"username"] = @"kishikawakatsumi@mac.com";  
store[@"password"] = @"password1234";
```

```
[store synchronize];
```

Debug print

```
UICKeyChainStore *store = [UICKeyChainStore keyChainStoreWithService:@"com.kishikawakatsu  
NSLog(@"%@", store); // Print all keys and values for the service.
```



Easy as that. (See [UICKeyChainStore.h](#) for all of the methods.)

License

UICKeyChainStore is available under the [MIT license](#). See the LICENSE file for more info.

Crontab-IOS

A light-weight library to execute Objective-C codes only once in app life with a crontab like way. Execute code/blocks only for the first time the app runs, for example.

Installation

We recommend you to install this project using CocoaPods:

Installation with CocoaPods

CocoaPods is a dependency manager for Objective-C, which automates and simplifies the process of using 3rd-party libraries like [EasyIOS](#) in your projects.

Podfile

```
platform :ios, '5.0'
pod "Crontab-IOS"
```

Usage

cronRule : times day

```
NSString * cronRule = @"1 1";
[CronTabCenter addCronJob:cronRule forBlock:^(
    NSLog(@"1 1 executed");
)];
```

- The cronRule is kind of NSString class , and separated By a Space.
- The first number represents the times of the app launch today.
- The second number represents the day of the app launch.
- `@"1 1"` means the first day's first launch, just like for the first time to launch. Use this option, for example, to run a welcome dialog or create a initial database.
- `@"1 *"` means the every one day's first launch. Use this option, for example, to ask to buy the PRO version every day.
- `@"1 */7"` means the every seven day's first launch. Use this option, for example, to ask to rate your app every 7 days.

cronRule : totalTimes

```
NSString * cronRule = @"1";
[CronTabCenter addCronJob:cronRule forBlock:^(
    NSLog(@"1 executed");
)];
```

- The cronRule is kind of NSString class , with only one number.
- The cronRule number represents the total times of the app launch.
- `@"1"` means the first day's first launch, just like the `@"1 1"` .
- `@"*"` means the every launch, just like the `@"* *"` .

Example

This is just a example to show the usage of crontab-ios

```
[CronTabCenter addCronJob:@"1 1" forBlock:^(
    NSLog(@"1 1 executed");
)];

[CronTabCenter addCronJob:@"2 1" forBlock:^(
    NSLog(@"2 1 executed");
)];

[CronTabCenter addCronJob:@"1 *" forBlock:^(
    NSLog(@"1 * executed");
)];

[CronTabCenter addCronJob:@"2 *" forBlock:^(
    NSLog(@"2 * executed");
} elseBlock:^(
    NSLog(@"2 * else executed");
)];

[CronTabCenter addCronJob:@"3" forBlock:^(
    NSLog(@"3 executed");
)];
```

- When we launch the app for the first time, it will log the result

```
1 1 executed
1 * executed
2 * else executed
```

- And when we launch the app for the second time,it will log the result

```
2 1 executed
2 * executed
```

- And when we launch the app for the third time,it will log the result

```
2 * else executed  
3 executed
```

- And when we launch the app for the forth time,it will log the result

```
2 * else executed
```

- Scenemode是怎么出发request 调用action的呢?demo里是scene观察tableview的page

scenemodel有一个成员属性叫action, scenemodel会把你的request提交给action, 观察page是因为有个分

- 触发的时机呢

想要发起网络请求就[self SEND_NO_CACHE_ACTION:_goodsListRequest];

- 如果viewDidLoad里我不初始化tableview的page 是不会请求网络数据

网络请求和page没关系。。。看你哪个页面要不要分页, 如果不用, 就没有page啊。如果有分页, 那么我只要把

- scene里声明了成名变量scenemodel 然后我在viewDidLoad里调用 [scenemodel SEND_NO_CACHE_ACTION] 这样可行?

这样是可以的。。。不过就违背了mvvm思想, 而且你在viewDidLoad里面调用, 要是你别的地方还想要调用, 就又

- 但是我要按mvvm的思想要做的话我是不是要在scenemodel里声明个变量然后我在viewDidLoad里改变scenemodel的变量来触发scenemodel调用 SEND_NO_CACHE_ACTION

是的

- 但这个变量我好像没啥意义 只是为了触发才这么做?

是的!, ReactiveViewModel, 也是这样的, 它定义了一个叫active属性, 只要active是yes就干活, 这个变
如果你实在是没有这个状态变量, 就仿照ReactiveViewModel, 定义一个active属性, 每次修改成YES, 执行

- [ReactiveViewModel项目地址](#)

- RAC(self.homeSceneModel.request,page) = RACObserve(self.tableView,page); 这个大概是啥意思 没看懂

观察 self.tableView,page 如果有修改, 就同步到 self.homeSceneModel.request,page