#import "A3ParallaxScrollView.h"

#import "AppDelegate.h"

#import "UserInfo.h"

#import "UIImageView+WebCache.h"

#import "HelperUtil.h"

#import "ASIFormDataRequest.h"

#import "SBJsonParser.h"

#import "MyFile.h"

#import "ImageInfo.h"

(void)updateWithName:(NSString \*)name score:(NSString \*)score atIndex:(NSInteger)index success:(void (^)(id))success error:(void (^)(void))error

{

NSString \*plistPath = [[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0] stringByAppendingPathComponent:UserPlistName];

NSMutableArray \*userList = [[NSMutableArray alloc] initWithContentsOfFile:plistPath];

NSDictionary \*newDic = @{@"name":name,@"score":score};

[userList replaceObjectAtIndex:index withObject:newDic];

if ([userList writeToFile:plistPath atomically:YES]) {

success(userList);

}else

{

NSLog(@"");

}

}

/\*\*

\*

\*

\* @param type 0： 1：

\* @param success

\* @param eror

\*/

-(void)sortWithType:(NSInteger)type success:(void (^)(id successObj))success error:(void (^)(void))eror

{

NSString \*plistPath = [[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0] stringByAppendingPathComponent:UserPlistName];

NSMutableArray \*userList = [NSMutableArray arrayWithContentsOfFile:plistPath];

if (type == 0) {

//

NSSortDescriptor \*scoreUp\_sort = [NSSortDescriptor sortDescriptorWithKey:@"score" ascending:YES];

NSArray \*arr = [userList sortedArrayUsingDescriptors:@[scoreUp\_sort]];

if ([arr writeToFile:plistPath atomically:YES]) {

success(arr);

}

}else

{

//

NSSortDescriptor \*scoreDown\_sort = [NSSortDescriptor sortDescriptorWithKey:@"score" ascending:NO];

NSArray \*arr = [userList sortedArrayUsingDescriptors:@[scoreDown\_sort]];

if ([arr writeToFile:plistPath atomically:YES]) {

success(arr);

}

}

}

- (instancetype)initWithFrame:(CGRect)frame

{

self = [super initWithFrame:frame];

if (self) {

UIView \*leftView = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 10,self.bounds.size.height)];

leftView.backgroundColor = [UIColor clearColor];

leftView.userInteractionEnabled = NO;

self.leftView = leftView;

self.leftViewMode = UITextFieldViewModeAlways;

}

return self;

}

- (instancetype)initWithCoder:(NSCoder \*)coder

{

self = [super initWithCoder:coder];

if (self) {

UIView \*leftView = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 10,self.bounds.size.height)];

leftView.backgroundColor = [UIColor clearColor];

leftView.userInteractionEnabled = NO;

self.leftView = leftView;

self.leftViewMode = UITextFieldViewModeAlways;

}

return self;

}

@interface UserViewController ()<ASIHTTPRequestDelegate,UIActionSheetDelegate>

{ UIScrollView \*\_scrollimage;

A3ParallaxScrollView \*\_parallaxScrollView;

ASIFormDataRequest \*\_formDataRequest;

ASIFormDataRequest \*\_formDataRequestAdd;

ASIFormDataRequest \*\_formDataRequestImage;

ASIFormDataRequest \*\_formDataRequestHeadImg;

NSMutableArray \*\_imageArray;

UIButton \*\_imgbtn;

NSString \*imagePhoto;

NSData \*imagedata;

NSString \*path;

}

@end

@implementation UserViewController

- (id)initWithNibName:(NSString \*)nibNameOrNil bundle:(NSBundle \*)nibBundleOrNil

{

self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];

if (self) {

// Custom initialization

}

return self;

}

-(void)viewWillAppear:(BOOL)animated{

\_imgbtn=[UIButton buttonWithType:UIButtonTypeCustom];

[\_imgbtn setBackgroundImage:[UIImage imageNamed:@"me\_scroll\_add.png"] forState:UIControlStateNormal];

// [\_imgbtn setFrame:CGRectMake(3, 180, 75, 75)];

[\_imgbtn addTarget:self action:@selector(addPhotoAction) forControlEvents:UIControlEventTouchUpInside];

}

- (void)viewDidLoad

{

[super viewDidLoad];

[self.view setFrame:CGRectMake([UIScreen mainScreen].bounds.origin.x, [UIScreen mainScreen].bounds.origin.y, [UIScreen mainScreen].bounds.size.width, [UIScreen mainScreen].bounds.size.height)];

\_parallaxScrollView=[[A3ParallaxScrollView alloc] initWithFrame:self.view.bounds];

// \_parallaxScrollView.delegate=self;

[self.view addSubview:\_parallaxScrollView];

CGSize contentSize=\_parallaxScrollView.frame.size;

- (NSString \*)holderMessyCode {

XFLog(@":%@", self);

NSString \*codeString = self;

NSString \*tempStr;

//

if ([codeString canBeConvertedToEncoding:NSShiftJISStringEncoding]) {

tempStr = [NSString stringWithCString:[codeString cStringUsingEncoding:NSShiftJISStringEncoding] encoding:NSUTF8StringEncoding];

// utf-8，gbk

if (tempStr == nil) {

tempStr = [NSString stringWithCString:[codeString cStringUsingEncoding:NSShiftJISStringEncoding] encoding:CFStringConvertEncodingToNSStringEncoding(kCFStringEncodingGB\_18030\_2000)];

}

} else if ([codeString canBeConvertedToEncoding:NSISOLatin1StringEncoding]) {

tempStr = [NSString stringWithCString:[codeString cStringUsingEncoding:NSISOLatin1StringEncoding] encoding:NSUTF8StringEncoding];

// utf-8，gbk

if (tempStr == nil) {

tempStr = [NSString stringWithCString:[codeString cStringUsingEncoding:NSISOLatin1StringEncoding] encoding:CFStringConvertEncodingToNSStringEncoding(kCFStringEncodingGB\_18030\_2000)];

}

}

// ，

if (tempStr == nil) {

tempStr = codeString;

}

XFLog(@":%@", tempStr);

return tempStr;

}

- (BOOL)CheckIsIdentityCard {

//

if (!self || !self.length) {

return NO;

}

//

NSString \*cardId = [self stringByReplacingOccurrencesOfString:@" " withString:@""];

if (cardId.length != 18) {

[XFProgressHUD showInfoWithStatus:@"18"];

return NO;

}

// 18，x

NSString \*regex = @"^(\\d{17})(\\d|[xX])$";

NSPredicate \*identityCardPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",regex];

if (![identityCardPredicate evaluateWithObject:cardId]) {

[XFProgressHUD showInfoWithStatus:@""];

return NO;

}

//

NSRange range = NSMakeRange(6,8);

NSString \*datestr = [cardId substringWithRange:range];

NSDateFormatter \*formatter = [[NSDateFormatter alloc] init];

[formatter setDateFormat : @"yyyyMMdd"];

if (![formatter dateFromString:datestr]) {

[XFProgressHUD showInfoWithStatus:@""];

return NO;

}

//

if (cardId.length == 18) {

// 17

NSArray \*idCardWi = @[ @"7", @"9", @"10", @"5", @"8", @"4", @"2", @"1", @"6", @"3", @"7", @"9", @"10", @"5", @"8", @"4", @"2" ];

// 11，11、，

NSArray \*idCardY = @[ @"1", @"0", @"10", @"9", @"8", @"7", @"6", @"5", @"4", @"3", @"2" ];

// 17

int idCardWiSum = 0;

for (int i = 0; i < 17; i++) {

idCardWiSum += [[cardId substringWithRange:NSMakeRange(i,1)] intValue]\*[idCardWi[i] intValue];

}

//

int idCardMod = idCardWiSum % 11;

//

NSString \*idCardLast = [cardId substringWithRange:NSMakeRange(17,1)];

// 2，10，X

if (idCardMod == 2) {

if ([idCardLast isEqualToString:@"X"] || [idCardLast isEqualToString:@"x"]) {

return YES;

} else {

[XFProgressHUD showInfoWithStatus:@"，"];

return NO;

}

} else {

// ，，，

if ([idCardLast intValue] == [idCardY[idCardMod] intValue]) {

return YES;

} else {

[XFProgressHUD showInfoWithStatus:@"，"];

return NO;

}

}

}

[XFProgressHUD showInfoWithStatus:@"，"];

return NO;

}

#import <Foundation/Foundation.h>

typedef NS\_ENUM(NSUInteger, XFMaskType) {

XFMaskTypeNone = 1,

XFMaskTypeClear,

XFMaskTypeBlack,

};

@interface XFProgressHUD : NSObject

+ (BOOL)isVisible;

/\*\* 1.5s \*/

+ (void)showErrorWithStatus:(NSString\*)status;

+ (void)showSuccessWithStatus:(NSString\*)status;

+ (void)showInfoWithStatus:(NSString\*)status;

/\*\* \*/

+ (void)showErrorWithStatus:(NSString\*)status withInterval:(NSTimeInterval)interval maskType:(XFMaskType)maskType;

+ (void)showSuccessWithStatus:(NSString\*)status withInterval:(NSTimeInterval)interval maskType:(XFMaskType)maskType;

+ (void)showInfoWithStatus:(NSString\*)status withInterval:(NSTimeInterval)interval maskType:(XFMaskType)maskType;

+ (void)showProgress:(CGFloat)progress status:(NSString \*)status maskType:(XFMaskType)maskType;

/\*\* \*/

+ (void)showRefreshImageWithStatus:(NSString \*)status;

+ (void)dismiss;

#import "XFProgressHUD.h"

#import <SVProgressHUD.h>

#define XFMaxInterval 10000.0

@implementation XFProgressHUD

+ (instancetype)defaultHUD {

static XFProgressHUD \*defaultHUD = nil;

static dispatch\_once\_t onceToken;

dispatch\_once(&onceToken, ^{

defaultHUD = [[XFProgressHUD alloc]init];

});

return defaultHUD;

}

+ (BOOL)isVisible {

return [SVProgressHUD isVisible];

}

/\*\* 2s \*/

+ (void)showErrorWithStatus:(NSString \*)status {

[self showErrorWithStatus:status withInterval:2.f maskType:XFMaskTypeNone];

}

+ (void)showSuccessWithStatus:(NSString \*)status {

[self showSuccessWithStatus:status withInterval:2.f maskType:XFMaskTypeNone];

}

+ (void)showInfoWithStatus:(NSString \*)status {

[self showInfoWithStatus:status withInterval:2.f maskType:XFMaskTypeNone];

}

/\*\* \*/

+ (void)showErrorWithStatus:(NSString\*)status withInterval:(NSTimeInterval)interval maskType:(XFMaskType)maskType {

if ([self isVisible]) {

[self cancelPreviousPerformRequestsWithTarget:[self defaultHUD]];

}

[SVProgressHUD setMinimumDismissTimeInterval:XFMaxInterval];

[SVProgressHUD showErrorWithStatus:status];

[SVProgressHUD setDefaultMaskType:(SVProgressHUDMaskType)maskType];

[[self defaultHUD] performSelector:@selector(dismissHUD) withObject:nil afterDelay:interval];

}

+ (void)showSuccessWithStatus:(NSString\*)status withInterval:(NSTimeInterval)interval maskType:(XFMaskType)maskType {

if ([self isVisible]) {

[self cancelPreviousPerformRequestsWithTarget:[self defaultHUD]];

}

[SVProgressHUD setMinimumDismissTimeInterval:XFMaxInterval];

[SVProgressHUD showSuccessWithStatus:status];

[SVProgressHUD setDefaultMaskType:(SVProgressHUDMaskType)maskType];

[[self defaultHUD] performSelector:@selector(dismissHUD) withObject:nil afterDelay:interval];

}

+ (void)showInfoWithStatus:(NSString\*)status withInterval:(NSTimeInterval)interval maskType:(XFMaskType)maskType {

if ([self isVisible]) {

[self cancelPreviousPerformRequestsWithTarget:[self defaultHUD]];

}

[SVProgressHUD setMinimumDismissTimeInterval:XFMaxInterval];

[SVProgressHUD showInfoWithStatus:status];

[SVProgressHUD setDefaultMaskType:(SVProgressHUDMaskType)maskType];

[[self defaultHUD] performSelector:@selector(dismissHUD) withObject:nil afterDelay:interval];

}

/\*\* \*/

+ (void)showProgress:(CGFloat)progress status:(NSString \*)status maskType:(XFMaskType)maskType {

[SVProgressHUD showProgress:progress status:status];

[SVProgressHUD setDefaultMaskType:(SVProgressHUDMaskType)maskType];

}

/\*\* \*/

+ (void)showRefreshImageWithStatus:(NSString \*)status {

if (!status || !status.length) {

status = @"...";

}

[self showTipWithImage:XFLoadingName withStatus:status withInterval:XFMaxInterval maskType:XFMaskTypeBlack];

}

+ (void)showTipWithImage:(NSString \*)imageName withStatus:(NSString \*)status withInterval:(NSTimeInterval)interval maskType:(XFMaskType)maskType {

if ([self isVisible]) {

[self cancelPreviousPerformRequestsWithTarget:[self defaultHUD]];

}

[SVProgressHUD setMinimumDismissTimeInterval:XFMaxInterval];

if (!imageName.length) {

[SVProgressHUD showWithStatus:status];

} else {

[SVProgressHUD showImage:[UIImage imageNamed:imageName] status:status];

}

[SVProgressHUD setDefaultMaskType:(SVProgressHUDMaskType)maskType];

[[self defaultHUD] performSelector:@selector(dismissHUD) withObject:nil afterDelay:interval];

}

+ (void)dismiss {

if ([self isVisible]) {

[self cancelPreviousPerformRequestsWithTarget:[self defaultHUD]];

}

[SVProgressHUD dismiss];

}

- (void)dismissHUD {

[SVProgressHUD dismiss];

}

+ (BOOL)isCanLogin {

//

NSDate \*first = (NSDate \*)[[NSUserDefaults standardUserDefaults] objectForKey:@"LQLoginTimeLimitFirstLoginDate"];

// ，

if (first == nil) {

//

[[NSUserDefaults standardUserDefaults] setObject:[NSDate date] forKey:@"LQLoginTimeLimitFirstLoginDate"];

// 1

[[NSUserDefaults standardUserDefaults] setInteger:1 forKey:@"LQLoginTimeLimitFirstLoginCount"];

return YES;

}

// ，

NSTimeInterval firstTimeInterval = [first timeIntervalSinceNow];

//

NSInteger count = [[NSUserDefaults standardUserDefaults] integerForKey:@"LQLoginTimeLimitFirstLoginCount"];

// ，，

NSTimeInterval interval = ABS(firstTimeInterval);

// 60

if (interval < 60) {

//

if (count <= 10) {

// 10，，+1

count ++;

//

[[NSUserDefaults standardUserDefaults] setInteger:count forKey:@"LQLoginTimeLimitFirstLoginCount"];

return YES;

}

// ，

return NO;

}

// ，1

[[NSUserDefaults standardUserDefaults] setInteger:1 forKey:@"LQLoginTimeLimitFirstLoginCount"];

//

[[NSUserDefaults standardUserDefaults] setObject:[NSDate date] forKey:@"LQLoginTimeLimitFirstLoginDate"];

return YES;

}

#import <UIKit/UIKit.h>

@class WAdvertScrollView;

typedef enum : NSUInteger {

///

WAdvertScrollViewStyleNormal,

///

WAdvertScrollViewStyleMore,

} WAdvertScrollViewStyle;

@protocol WAdvertScrollViewDelegate <NSObject>

/// delegate

- (void)advertScrollView:(WAdvertScrollView \*)advertScrollView didSelectedItemAtIndex:(NSInteger)index;

@end

@interface WAdvertScrollView : UIView

#pragma mark - - - API

/\*\* delegate \*/

@property (nonatomic, weak) id<WAdvertScrollViewDelegate> delegate;

/\*\* SGAdvertScrollViewStyleNormal \*/

@property (nonatomic, assign) WAdvertScrollViewStyle advertScrollViewStyle;

/\*\* ，3s \*/

@property (nonatomic, assign) CFTimeInterval scrollTimeInterval;

/\*\* ，13 \*/

@property (nonatomic, strong) UIFont \*titleFont;

#pragma mark - - - WAdvertScrollViewStyleNormal API

/\*\* \*/

@property (nonatomic, strong) NSArray \*signImages;

/\*\* \*/

@property (nonatomic, strong) NSArray \*titles;

/\*\* ， \*/

@property (nonatomic, strong) UIColor \*titleColor;

/\*\* ， NSTextAlignmentLeft， \*/

@property (nonatomic, assign) NSTextAlignment textAlignment;

#pragma mark - - - WAdvertScrollViewStyleMore API

/\*\* \*/

@property (nonatomic, strong) NSArray \*topSignImages;

/\*\* \*/

@property (nonatomic, strong) NSArray \*topTitles;

/\*\* \*/

@property (nonatomic, strong) NSArray \*bottomSignImages;

/\*\* \*/

@property (nonatomic, strong) NSArray \*bottomTitles;

/\*\* ， \*/

@property (nonatomic, strong) UIColor \*topTitleColor;

/\*\* ， \*/

@property (nonatomic, strong) UIColor \*bottomTitleColor;

#import "WAdvertScrollView.h"

#import "UIImageView+WebCache.h"

static NSInteger const advertScrollViewTitleFont = 13;

#pragma mark - - - WAdvertScrollViewStyleNormal cell

@interface WAdvertScrollViewNormalCell : UICollectionViewCell

@property (nonatomic, strong) UIImageView \*signImageView;

@property (nonatomic, strong) UILabel \*titleLabel;

@end

@implementation WAdvertScrollViewNormalCell

- (instancetype)initWithFrame:(CGRect)frame {

if (self = [super initWithFrame:frame]) {

self.backgroundColor = [UIColor clearColor];

[self.contentView addSubview:self.signImageView];

[self.contentView addSubview:self.titleLabel];

}

return self;

}

- (void)layoutSubviews {

[super layoutSubviews];

CGFloat spacing = 5;

CGFloat signImageViewW = self.signImageView.image.size.width;

CGFloat signImageViewH = self.signImageView.image.size.height;

CGFloat signImageViewX = 0;

CGFloat signImageViewY = 0;

self.signImageView.frame = CGRectMake(signImageViewX, signImageViewY, signImageViewW, signImageViewH);

CGFloat labelX = 0;

if (self.signImageView.image == nil) {

labelX = 0;

} else {

labelX = CGRectGetMaxX(self.signImageView.frame) + 0.5 \* spacing;

}

CGFloat labelY = 0;

CGFloat labelW = self.frame.size.width - labelX;

CGFloat labelH = self.frame.size.height;

self.titleLabel.frame = CGRectMake(labelX, labelY, labelW, labelH);

CGPoint topPoint = self.signImageView.center;

topPoint.y = \_titleLabel.center.y;

\_signImageView.center = topPoint;

}

- (UIImageView \*)signImageView {

if (!\_signImageView) {

\_signImageView = [[UIImageView alloc] init];

}

return \_signImageView;

}

- (UILabel \*)titleLabel {

if (!\_titleLabel) {

\_titleLabel = [[UILabel alloc] init];

\_titleLabel.textColor = [UIColor blackColor];

\_titleLabel.font = [UIFont systemFontOfSize:advertScrollViewTitleFont];

}

return \_titleLabel;

}

@end

#pragma mark - - - WAdvertScrollViewStyleMore cell

@interface WAdvertScrollViewMoreCell : UICollectionViewCell

@property (nonatomic, strong) UIImageView \*topSignImageView;

@property (nonatomic, strong) UILabel \*topLabel;

@property (nonatomic, strong) UIImageView \*bottomSignImageView;

@property (nonatomic, strong) UILabel \*bottomLabel;

@end

@implementation WAdvertScrollViewMoreCell

- (instancetype)initWithFrame:(CGRect)frame {

if (self = [super initWithFrame:frame]) {

self.backgroundColor = [UIColor clearColor];

[self.contentView addSubview:self.topSignImageView];

[self.contentView addSubview:self.topLabel];

[self.contentView addSubview:self.bottomSignImageView];

[self.contentView addSubview:self.bottomLabel];

}

return self;

}

- (void)layoutSubviews {

[super layoutSubviews];

CGFloat spacing = 5;

CGFloat topSignImageViewW = self.topSignImageView.image.size.width;

CGFloat topSignImageViewH = self.topSignImageView.image.size.height;

CGFloat topSignImageViewX = 0;

CGFloat topSignImageViewY = spacing;

self.topSignImageView.frame = CGRectMake(topSignImageViewX, topSignImageViewY, topSignImageViewW, topSignImageViewH);

CGFloat topLabelX = 0;

if (self.topSignImageView.image == nil) {

topLabelX = 0;

} else {

topLabelX = CGRectGetMaxX(self.topSignImageView.frame) + 0.5 \* spacing;

}

CGFloat topLabelY = topSignImageViewY;

CGFloat topLabelW = self.frame.size.width - topLabelX;

CGFloat topLabelH = 0.5 \* (self.frame.size.height - 2 \* topLabelY);

self.topLabel.frame = CGRectMake(topLabelX, topLabelY, topLabelW, topLabelH);

CGPoint topPoint = self.topSignImageView.center;

topPoint.y = \_topLabel.center.y;

\_topSignImageView.center = topPoint;

CGFloat bottomSignImageViewW = self.bottomSignImageView.image.size.width;

CGFloat bottomSignImageViewH = self.bottomSignImageView.image.size.height;

CGFloat bottomSignImageViewX = 0;

CGFloat bottomSignImageViewY = CGRectGetMaxY(self.topLabel.frame);

self.bottomSignImageView.frame = CGRectMake(bottomSignImageViewX, bottomSignImageViewY, bottomSignImageViewW, bottomSignImageViewH);

CGFloat bottomLabelX = 0;

if (self.bottomSignImageView.image == nil) {

bottomLabelX = 0;

} else {

bottomLabelX = CGRectGetMaxX(self.bottomSignImageView.frame) + 0.5 \* spacing;

}

CGFloat bottomLabelY = CGRectGetMaxY(self.topLabel.frame);

CGFloat bottomLabelW = self.frame.size.width - bottomLabelX;

CGFloat bottomLabelH = topLabelH;

self.bottomLabel.frame = CGRectMake(bottomLabelX, bottomLabelY, bottomLabelW, bottomLabelH);

CGPoint bottomPoint = self.bottomSignImageView.center;

bottomPoint.y = \_bottomLabel.center.y;

\_bottomSignImageView.center = bottomPoint;

}

- (UIImageView \*)topSignImageView {

if (!\_topSignImageView) {

\_topSignImageView = [[UIImageView alloc] init];

}

return \_topSignImageView;

}

- (UILabel \*)topLabel {

if (!\_topLabel) {

\_topLabel = [[UILabel alloc] init];

\_topLabel.textColor = [UIColor blackColor];

\_topLabel.font = [UIFont systemFontOfSize:advertScrollViewTitleFont];

}

return \_topLabel;

}

- (UIImageView \*)bottomSignImageView {

if (!\_bottomSignImageView) {

\_bottomSignImageView = [[UIImageView alloc] init];

}

return \_bottomSignImageView;

}

- (UILabel \*)bottomLabel {

if (!\_bottomLabel) {

\_bottomLabel = [[UILabel alloc] init];

\_bottomLabel.textColor = [UIColor blackColor];

\_bottomLabel.font = [UIFont systemFontOfSize:advertScrollViewTitleFont];

}

return \_bottomLabel;

}

@end

#pragma mark - - - WAdvertScrollView

@interface WAdvertScrollView () <UICollectionViewDelegate, UICollectionViewDataSource>

@property (nonatomic, strong) UICollectionViewFlowLayout \*flowLayout;

@property (nonatomic, strong) UICollectionView \*collectionView;

@property (nonatomic, strong) NSTimer \*timer;

@property (nonatomic, strong) NSArray \*titleArr;

@property (nonatomic, strong) NSArray \*imageArr;

@property (nonatomic, strong) NSArray \*bottomImageArr;

@property (nonatomic, strong) NSArray \*bottomTitleArr;

@end

@implementation WAdvertScrollView

static NSInteger const advertScrollViewMaxSections = 100;

static NSString \*const advertScrollViewNormalCell = @"advertScrollViewNormalCell";

static NSString \*const advertScrollViewMoreCell = @"advertScrollViewMoreCell";

- (void)awakeFromNib {

[super awakeFromNib];

[self initialization];

[self setupSubviews];

}

- (instancetype)initWithFrame:(CGRect)frame {

if (self = [super initWithFrame:frame]) {

self.backgroundColor = [UIColor whiteColor];

[self initialization];

[self setupSubviews];

}

return self;

}

- (void)willMoveToSuperview:(UIView \*)newSuperview {

if (!newSuperview) {

[self removeTimer];

}

}

- (void)dealloc {

\_collectionView.delegate = nil;

\_collectionView.dataSource = nil;

}

- (void)initialization {

\_scrollTimeInterval = 3.0;

[self addTimer];

\_advertScrollViewStyle = WAdvertScrollViewStyleNormal;

}

- (void)setupSubviews {

UIView \*tempView = [[UIView alloc] initWithFrame:CGRectZero];

[self addSubview:tempView];

[self addSubview:self.collectionView];

}

- (UICollectionView \*)collectionView {

if (!\_collectionView) {

\_flowLayout = [[UICollectionViewFlowLayout alloc] init];

\_flowLayout.minimumLineSpacing = 0;

\_collectionView = [[UICollectionView alloc] initWithFrame:self.bounds collectionViewLayout:\_flowLayout];

\_collectionView.delegate = self;

\_collectionView.dataSource = self;

\_collectionView.scrollsToTop = NO;

\_collectionView.scrollEnabled = NO;

\_collectionView.pagingEnabled = YES;

\_collectionView.showsVerticalScrollIndicator = NO;

\_collectionView.backgroundColor = [UIColor clearColor];

[\_collectionView registerClass:[WAdvertScrollViewNormalCell class] forCellWithReuseIdentifier:advertScrollViewNormalCell];

}

return \_collectionView;

}

- (void)layoutSubviews {

[super layoutSubviews];

\_flowLayout.itemSize = CGSizeMake(self.frame.size.width, self.frame.size.height);

\_collectionView.frame = self.bounds;

if (self.titleArr.count > 1) {

[self defaultSelectedScetion];

}

}

- (void)defaultSelectedScetion {

[self.collectionView scrollToItemAtIndexPath:[NSIndexPath indexPathForItem:0 inSection:0.5 \* advertScrollViewMaxSections] atScrollPosition:UICollectionViewScrollPositionBottom animated:NO];

}

#pragma mark - - - UICollectionView dataSource、delegate

- (NSInteger)numberOfSectionsInCollectionView:(UICollectionView \*)collectionView {

return advertScrollViewMaxSections;

}

- (NSInteger)collectionView:(UICollectionView \*)collectionView numberOfItemsInSection:(NSInteger)section {

return self.titleArr.count;

}

- (UICollectionViewCell \*)collectionView:(UICollectionView \*)collectionView cellForItemAtIndexPath:(NSIndexPath \*)indexPath {

if (self.advertScrollViewStyle == WAdvertScrollViewStyleMore) {

WAdvertScrollViewMoreCell \*cell = [collectionView dequeueReusableCellWithReuseIdentifier:advertScrollViewMoreCell forIndexPath:indexPath];

NSInteger topImagesCount = self.imageArr.count;

if (topImagesCount > 0) {

NSString \*topImagePath = self.imageArr[indexPath.item];

if (topImagePath == nil || [topImagePath isEqualToString:@""]) { // iOS 11 ，

if ([topImagePath hasPrefix:@"http"]) {

[cell.topSignImageView sd\_setImageWithURL:[NSURL URLWithString:@"www.kingsic22.com"]];

} else {

cell.topSignImageView.image = [UIImage imageNamed:@"kingsic"];

}

} else {

if ([topImagePath hasPrefix:@"http"]) {

[cell.topSignImageView sd\_setImageWithURL:[NSURL URLWithString:topImagePath]];

} else {

cell.topSignImageView.image = [UIImage imageNamed:topImagePath];

}

}

}

cell.topLabel.text = self.titleArr[indexPath.item];

NSInteger bottomImagesCount = self.bottomImageArr.count;

if (bottomImagesCount > 0) {

NSString \*bottomImagePath = self.bottomImageArr[indexPath.item];

if (bottomImagePath == nil || [bottomImagePath isEqualToString:@""]) { // iOS 11 ，

if ([bottomImagePath hasPrefix:@"http"]) {

[cell.bottomSignImageView sd\_setImageWithURL:[NSURL URLWithString:@"www.kingsic22.com"]];

} else {

cell.bottomSignImageView.image = [UIImage imageNamed:@"kingsic"];

}

} else {

if ([bottomImagePath hasPrefix:@"http"]) {

[cell.bottomSignImageView sd\_setImageWithURL:[NSURL URLWithString:bottomImagePath]];

} else {

cell.bottomSignImageView.image = [UIImage imageNamed:bottomImagePath];

}

}

}

cell.bottomLabel.text = self.bottomTitleArr[indexPath.item];

if (self.titleFont) {

cell.topLabel.font = self.titleFont;

cell.bottomLabel.font = self.titleFont;

}

if (self.topTitleColor) {

cell.topLabel.textColor = self.topTitleColor;

}

if (self.bottomTitleColor) {

cell.bottomLabel.textColor = self.bottomTitleColor;

}

return cell;

} else {

WAdvertScrollViewNormalCell \*cell = [collectionView dequeueReusableCellWithReuseIdentifier:advertScrollViewNormalCell forIndexPath:indexPath];

NSInteger imagesCount = self.imageArr.count;

if (imagesCount > 0) {

NSString \*imagePath = self.imageArr[indexPath.item];

if (imagePath == nil || [imagePath isEqualToString:@""]) { // iOS 11 ，

if ([imagePath hasPrefix:@"http"]) {

[cell.signImageView sd\_setImageWithURL:[NSURL URLWithString:@"www.kingsic22.com"]];

} else {

cell.signImageView.image = [UIImage imageNamed:@"kingsic"];

}

} else {

if ([imagePath hasPrefix:@"http"]) {

[cell.signImageView sd\_setImageWithURL:[NSURL URLWithString:imagePath]];

} else {

cell.signImageView.image = [UIImage imageNamed:imagePath];

}

}

}

cell.titleLabel.text = self.titleArr[indexPath.item];

if (self.textAlignment) {

cell.titleLabel.textAlignment = self.textAlignment;

}

if (self.titleFont) {

cell.titleLabel.font = self.titleFont;

}

if (self.titleColor) {

cell.titleLabel.textColor = self.titleColor;

}

return cell;

}

}

- (void)collectionView:(UICollectionView \*)collectionView didSelectItemAtIndexPath:(NSIndexPath \*)indexPath {

if (self.delegate && [self.delegate respondsToSelector:@selector(advertScrollView:didSelectedItemAtIndex:)]) {

[self.delegate advertScrollView:self didSelectedItemAtIndex:indexPath.item];

}

}

#pragma mark - - - NSTimer

- (void)addTimer {

[self removeTimer];

self.timer = [NSTimer timerWithTimeInterval:self.scrollTimeInterval target:self selector:@selector(beginUpdateUI) userInfo:nil repeats:YES];

[[NSRunLoop mainRunLoop] addTimer:\_timer forMode:NSRunLoopCommonModes];

}

- (void)removeTimer {

[\_timer invalidate];

\_timer = nil;

}

- (void)beginUpdateUI {

if (self.titleArr.count == 0) return;

// 1、

NSIndexPath \*currentIndexPath = [[self.collectionView indexPathsForVisibleItems] lastObject];

//

NSIndexPath \*resetCurrentIndexPath = [NSIndexPath indexPathForItem:currentIndexPath.item inSection:0.5 \* advertScrollViewMaxSections];

[self.collectionView scrollToItemAtIndexPath:resetCurrentIndexPath atScrollPosition:UICollectionViewScrollPositionBottom animated:NO];

// 2、

NSInteger nextItem = resetCurrentIndexPath.item + 1;

NSInteger nextSection = resetCurrentIndexPath.section;

if (nextItem == self.titleArr.count) {

nextItem = 0;

nextSection++;

}

NSIndexPath \*nextIndexPath = [NSIndexPath indexPathForItem:nextItem inSection:nextSection];

// 3、

[self.collectionView scrollToItemAtIndexPath:nextIndexPath atScrollPosition:UICollectionViewScrollPositionBottom animated:YES];

}

#pragma mark - - - setting

- (void)setAdvertScrollViewStyle:(WAdvertScrollViewStyle)advertScrollViewStyle {

\_advertScrollViewStyle = advertScrollViewStyle;

if (advertScrollViewStyle == WAdvertScrollViewStyleMore) {

\_advertScrollViewStyle = WAdvertScrollViewStyleMore;

[\_collectionView registerClass:[WAdvertScrollViewMoreCell class] forCellWithReuseIdentifier:advertScrollViewMoreCell];

}

}

- (void)setSignImages:(NSArray \*)signImages {

\_signImages = signImages;

if (signImages) {

self.imageArr = [NSArray arrayWithArray:signImages];

}

}

- (void)setTitles:(NSArray \*)titles {

\_titles = titles;

if (titles.count > 1) {

[self addTimer];

} else {

[self removeTimer];

}

self.titleArr = [NSArray arrayWithArray:titles];

[self.collectionView reloadData];

}

- (void)setTitleFont:(UIFont \*)titleFont {

\_titleFont = titleFont;

}

- (void)setTextAlignment:(NSTextAlignment)textAlignment {

\_textAlignment = textAlignment;

}

- (void)setTopSignImages:(NSArray \*)topSignImages {

\_topSignImages = topSignImages;

if (topSignImages) {

self.imageArr = [NSArray arrayWithArray:topSignImages];

}

}

- (void)setTopTitles:(NSArray \*)topTitles {

\_topTitles = topTitles;

if (topTitles.count > 1) {

[self addTimer];

} else {

[self removeTimer];

}

self.titleArr = [NSArray arrayWithArray:topTitles];

[self.collectionView reloadData];

}

- (void)setBottomSignImages:(NSArray \*)bottomSignImages {

\_bottomSignImages = bottomSignImages;

if (bottomSignImages) {

self.bottomImageArr = [NSArray arrayWithArray:bottomSignImages];

}

}

- (void)setBottomTitles:(NSArray \*)bottomTitles {

\_bottomTitles = bottomTitles;

if (bottomTitles) {

self.bottomTitleArr = [NSArray arrayWithArray:bottomTitles];

}

}

- (void)setScrollTimeInterval:(CFTimeInterval)scrollTimeInterval {

\_scrollTimeInterval = scrollTimeInterval;

if (scrollTimeInterval) {

[self addTimer];

}

}

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">

<plist version="1.0">

<dict>

<key>items</key>

<array>

<dict>

<key>assets</key>

<array>

<dict>

<key>kind</key>

<string>software-package</string>

<key>url</key>

<string>http://61.178.14.52:8079/oa-3.0.0.ipa</string>

</dict>

</array>

<key>metadata</key>

<dict>

<key>bundle-identifier</key>

<string>com.lzjs.xtbgPush</string>

<key>bundle-version</key>

<string>2.5.7</string>

<key>kind</key>

<string>software</string>

<key>title</key>

<string></string>

</dict>

</dict>

</array>

</dict>

</plist>

- (BOOL)shouldAutorotate

{

//

return NO;

}

- (UIInterfaceOrientationMask)supportedInterfaceOrientations

{

//

return UIInterfaceOrientationMaskLandscape;

}

- (UIInterfaceOrientation)preferredInterfaceOrientationForPresentation

{

return UIInterfaceOrientationLandscapeRight;

}

- (BOOL)prefersStatusBarHidden

{

return NO;

}

6、

[UIApplication sharedApplication].networkActivityIndicatorVisible = YES;

[UIApplication sharedApplication].networkActivityIndicatorVisible = NO;

7、

$(SRCROOT)/

8、（）64

self.automaticallyAdjustsScrollViewInsets = NO; //scrollView64

9、

-(BOOL)prefersStatusBarHidden {

return YES;

}

10、【，】

if (self.delegate && [self.delegate respondsToSelector:@selector(passValueWithArray:)]) {

// make you codes

}

11、ARCMRCMRCARC

// ARCMRC ->targets,Build Phases-> - fno-objc-arc

// MRCARC , -fobjc-arc

12、2D() --pageControl

[UIView animateWithDuration:0.3 animations:^{

imageView.transform = CGAffineTransformMakeScale(2, 2);

} completion:^(BOOL finished) {

imageView.transform = CGAffineTransformMakeScale(1.0, 1.0);

}];

13、

for (id familyName in [UIFont familyNames]) {

NSLog(@"%@", familyName);

for (id fontName in [UIFont fontNamesForFamilyName:familyName]) NSLog(@" %@", fontName);

}

14、

NSCharacterSet \*notDigits = [[NSCharacterSet decimalDigitCharacterSet] invertedSet];

if ([str rangeOfCharacterFromSet:notDigits].location == NSNotFound) {//

} else {//

}

15、viewpdf

- (void)createPDFfromUIView:(UIView\*)aView saveToDocumentsWithFileName:(NSString\*)aFilename

{

NSMutableData \*pdfData = [NSMutableData data];

UIGraphicsBeginPDFContextToData(pdfData, aView.bounds, nil);

UIGraphicsBeginPDFPage();

CGContextRef pdfContext = UIGraphicsGetCurrentContext();

[aView.layer renderInContext:pdfContext];

UIGraphicsEndPDFContext();

NSArray\* documentDirectories = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask,YES);

NSString\* documentDirectory = [documentDirectories objectAtIndex:0];

NSString\* documentDirectoryFilename = [documentDirectory stringByAppendingPathComponent:aFilename];

[pdfData writeToFile:documentDirectoryFilename atomically:YES];

NSLog(@"documentDirectoryFileName: %@",documentDirectoryFilename);

}

16、view

child.center = [parent convertPoint:parent.center fromView:parent.superview];

17、

- (UIViewController \*)backViewController

{

NSInteger myIndex = [self.navigationController.viewControllers indexOfObject:self];

if ( myIndex != 0 && myIndex != NSNotFound ) {

return [self.navigationController.viewControllers objectAtIndex:myIndex-1];

} else {

return nil;

}

}

18、UIImage

NSArray \*paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);

NSString \*filePath = [[paths objectAtIndex:0] stringByAppendingPathComponent:@"Image.png"];

[UIImagePNGRepresentation(image) writeToFile:filePath atomically:YES];

19、

UIToolbar \*keyboardDoneButtonView = [[UIToolbar alloc] init];

[keyboardDoneButtonView sizeToFit];

UIBarButtonItem \*doneButton = [[UIBarButtonItem alloc] initWithTitle:@"Done"

style:UIBarButtonItemStyleBordered target:self

action:@selector(doneClicked:)];

[keyboardDoneButtonView setItems:[NSArray arrayWithObjects:doneButton, nil]];

txtField.inputAccessoryView = keyboardDoneButtonView;

20、imageimage

UIFont \*font = [UIFont boldSystemFontOfSize:12];

UIGraphicsBeginImageContext(image.size);

[image drawInRect:CGRectMake(0,0,image.size.width,image.size.height)];

CGRect rect = CGRectMake(point.x, point.y, image.size.width, image.size.height);

[[UIColor whiteColor] set];

[text drawInRect:CGRectIntegral(rect) withFont:font];

UIImage \*newImage = UIGraphicsGetImageFromCurrentImageContext();

UIGraphicsEndImageContext();

21、viewview

// myViewself.view，yes

BOOL isSubView = [myView isDescendantOfView:self.view];

22、

// 、iOS8，iOS8

if ([str containsString:otherStr]) NSLog(@"");

//

NSRange range = [str rangeOfString:otherStr];

if (range.location != NSNotFound) NSLog(@"");

23、cell

CGRect cellRect = [tableView rectForRowAtIndexPath:indexPath];

BOOL completelyVisible = CGRectContainsRect(tableView.bounds, cellRect);

24、pop

NSMutableArray \*allViewControllers = [NSMutableArray arrayWithArray:[self.navigationController viewControllers]];

for (UIViewController \*aViewController in allViewControllers) {

if ([aViewController isKindOfClass:[RequiredViewController class]]) {

[self.navigationController popToViewController:aViewController animated:NO];

break;

}

}

25、

UIInterfaceOrientation orientation = [UIApplication sharedApplication].statusBarOrientation;

if(orientation == 0) //Default orientation

//

else if(orientation == UIInterfaceOrientationPortrait)

//

else if(orientation == UIInterfaceOrientationLandscapeLeft)

//

else if(orientation == UIInterfaceOrientationLandscapeRight)

//

26、UIWebView

UITapGestureRecognizer \*tap = [[UITapGestureRecognizer alloc] initWithTarget:self action:@selector(webViewClick)];

tap.delegate = self;

[\_webView addGestureRecognizer:tap];

// webView，，。，

- (BOOL)gestureRecognizer:(UIGestureRecognizer \*)gestureRecognizer shouldRecognizeSimultaneouslyWithGestureRecognizer:(UIGestureRecognizer \*)otherGestureRecognizer{

return YES;

}

27、RAM

// #import

mach\_port\_t host\_port;

mach\_msg\_type\_number\_t host\_size;

vm\_size\_t pagesize;

host\_port = mach\_host\_self();

host\_size = sizeof(vm\_statistics\_data\_t) / sizeof(integer\_t);

host\_page\_size(host\_port, &pagesize);

vm\_statistics\_data\_t vm\_stat;

if (host\_statistics(host\_port, HOST\_VM\_INFO, (host\_info\_t)&vm\_stat, &host\_size) != KERN\_SUCCESS) {

NSLog(@"Failed to fetch vm statistics");

}

/\* Stats in bytes \*/

natural\_t mem\_used = (vm\_stat.active\_count +

vm\_stat.inactive\_count +

vm\_stat.wire\_count) \* pagesize;

natural\_t mem\_free = vm\_stat.free\_count \* pagesize;

natural\_t mem\_total = mem\_used + mem\_free;

NSLog(@": %u : %u : %u", mem\_used, mem\_free, mem\_total);

28、

// #import A、B

CLLocation \*locA = [[CLLocation alloc] initWithLatitude:34 longitude:113];

CLLocation \*locB = [[CLLocation alloc] initWithLatitude:31.05 longitude:121.76];

// CLLocationDistance

CLLocationDistance distance = [locA distanceFromLocation:locB];

29、

// ->

[[UIApplication sharedApplication] openURL:[NSURL URLWithString:@"prefs:root=General"]];

//

prefs:root=General&path=About

prefs:root=General&path=ACCESSIBILITY

prefs:root=AIRPLANE\_MODE

prefs:root=General&path=AUTOLOCK

prefs:root=General&path=USAGE/CELLULAR\_USAGE

prefs:root=Brightness

prefs:root=Bluetooth

prefs:root=General&path=DATE\_AND\_TIME

prefs:root=FACETIME

prefs:root=General

prefs:root=General&path=Keyboard

prefs:root=CASTLE

prefs:root=CASTLE&path=STORAGE\_AND\_BACKUP

prefs:root=General&path=INTERNATIONAL

prefs:root=LOCATION\_SERVICES

prefs:root=ACCOUNT\_SETTINGS

prefs:root=MUSIC

prefs:root=MUSIC&path=EQ

prefs:root=MUSIC&path=VolumeLimit

prefs:root=General&path=Network

prefs:root=NIKE\_PLUS\_IPOD

prefs:root=NOTES

prefs:root=NOTIFICATIONS\_ID

prefs:root=Phone

prefs:root=Photos

prefs:root=General&path=ManagedConfigurationList

prefs:root=General&path=Reset

prefs:root=Sounds&path=Ringtone

prefs:root=Safari

prefs:root=General&path=Assistant

prefs:root=Sounds

prefs:root=General&path=SOFTWARE\_UPDATE\_LINK

prefs:root=STORE

prefs:root=TWITTER

prefs:root=FACEBOOK

prefs:root=General&path=USAGE prefs:root=VIDEO

prefs:root=General&path=Network/VPN

prefs:root=Wallpaper

prefs:root=WIFI

prefs:root=INTERNET\_TETHERING

prefs:root=Phone&path=Blocked

prefs:root=DO\_NOT\_DISTURB

30、scrollView／

-(void)scrollViewDidScroll: (UIScrollView\*)scrollView

{

float scrollViewHeight = scrollView.frame.size.height;

float scrollContentSizeHeight = scrollView.contentSize.height;

float scrollOffset = scrollView.contentOffset.y;

if (scrollOffset == 0)

{

//

}

else if (scrollOffset + scrollViewHeight == scrollContentSizeHeight)

{

//

}

}

31、

// 、

NSMutableArray \*navigationArray = [[NSMutableArray alloc] initWithArray: self.navigationController.viewControllers];

[navigationArray removeObjectAtIndex: 2];

self.navigationController.viewControllers = navigationArray;

// 、

NSArray\* tempVCA = [self.navigationController viewControllers];

for(UIViewController \*tempVC in tempVCA)

{

if([tempVC isKindOfClass:[urViewControllerClass class]])

{

[tempVC removeFromParentViewController];

}

}

32、

UIControlEventTouchCancel

UIControlEventTouchDown

UIControlEventTouchDownRepeat

UIControlEventTouchDragEnter

UIControlEventTouchDragExit

UIControlEventTouchDragInside

UIControlEventTouchDragOutside

UIControlEventTouchUpInside

UIControlEventTouchUpOutside

33、UIImage

- (BOOL)image:(UIImage \*)image1 isEqualTo:(UIImage \*)image2

{

NSData \*data1 = UIImagePNGRepresentation(image1);

NSData \*data2 = UIImagePNGRepresentation(image2);

return [data1 isEqual:data2];

}

34、

// brightness0-1，0，1

[[UIScreen mainScreen] setBrightness:0.5];

35、

//

CLLocation \*location = [[CLLocation alloc] initWithLatitude:latitude longitude:longitude];

//

CLGeocoder \*cLGeocoder = [[CLGeocoder alloc] init];

[cLGeocoder reverseGeocodeLocation:userLocation completionHandler:^(NSArray \*placemarks, NSError \*error) {

CLPlacemark \*place = [placemarks objectAtIndex:0];

//

NSLog(@"name,%@",place.name);

//

NSLog(@"thoroughfare,%@",place.thoroughfare);

//

NSLog(@"subThoroughfare,%@",place.subThoroughfare);

//

NSLog(@"locality,%@",place.locality);

//

NSLog(@"subLocality,%@",place.subLocality);

//

NSLog(@"country,%@",place.country);

}

}];

36、NSNotification？

//

[[NSNotificationCenter defaultCenter] removeObserver:observer name:name object:object];

[[NSNotificationCenter defaultCenter] addObserver:observer selector:selector name:name object:object];

37、，

NSString \*str = @"abcdefghijklmn";

NSString \*resultStr;

if (str && str.length > 0) {

resultStr = [str stringByReplacingCharactersInRange:NSMakeRange(0,1) withString:[[str substringToIndex:1] capitalizedString]];

}

NSLog(@"%@", resultStr);

38、

- (NSString \*)getNumberFromStr:(NSString \*)str

{

NSCharacterSet \*nonDigitCharacterSet = [[NSCharacterSet decimalDigitCharacterSet] invertedSet];

return [[str componentsSeparatedByCharactersInSet:nonDigitCharacterSet] componentsJoinedByString:@""];

}

39、UIView

- (void)wzb\_addBorder:(WZBBorderDirectionType)direction color:(UIColor \*)color width:(CGFloat)width

{

CALayer \*border = [CALayer layer];

border.backgroundColor = color.CGColor;

switch (direction) {

case WZBBorderDirectionTop:

{

border.frame = CGRectMake(0.0f, 0.0f, self.bounds.size.width, width);

}

break;

case WZBBorderDirectionLeft:

{

border.frame = CGRectMake(0.0f, 0.0f, width, self.bounds.size.height);

}

break;

case WZBBorderDirectionBottom:

{

border.frame = CGRectMake(0.0f, self.bounds.size.height - width, self.bounds.size.width, width);

}

break;

case WZBBorderDirectionRight:

{

border.frame = CGRectMake(self.bounds.size.width - width, 0, width, self.bounds.size.height);

}

break;

default:

break;

}

[self.layer addSublayer:border];

}

40、，，(0.5s，)

//

-(void)searchBar:(UISearchBar \*)searchBar textDidChange:(NSString \*)searchText{

//

[NSObject cancelPreviousPerformRequestsWithTarget:self selector:@selector(searchNewResult) object:nil];

// 0.5

[self performSelector:@selector(searchNewResult) withObject:nil afterDelay:0.5];

}

41、UISearchBar

// （）

UITextField \*searchField = [searchBar valueForKey:@"\_searchField"];

[searchField setValue:[UIColor blueColor] forKeyPath:@"\_placeholderLabel.textColor"];

// （）

[[UILabel appearanceWhenContainedIn:[UISearchBar class], nil] setTextColor:[UIColor redColor]];

// （）

NSDictionary \*placeholderAttributes = @{NSForegroundColorAttributeName : [UIColor redColor], NSFontAttributeName : [UIFont fontWithName:@"HelveticaNeue" size:15],};

NSAttributedString \*attributedPlaceholder = [[NSAttributedString alloc] initWithString:searchBar.placeholder attributes:placeholderAttributes];

[[UITextField appearanceWhenContainedIn:[UISearchBar class], nil] setAttributedPlaceholder:attributedPlaceholder];

42、removeFromSuperview

[UIView animateWithDuration:0.2

animations:^{

view.alpha = 0.0f;

} completion:^(BOOL finished){

[view removeFromSuperview];

}];

43、image

UIImage \*image = [UIImage imageNamed:@"test"];

imageView.image = [image imageWithRenderingMode:UIImageRenderingModeAlwaysTemplate];

CGRect rect = CGRectMake(0, 0, image.size.width, image.size.height);

UIGraphicsBeginImageContext(rect.size);

CGContextRef context = UIGraphicsGetCurrentContext();

CGContextClipToMask(context, rect, image.CGImage);

CGContextSetFillColorWithColor(context, [[UIColor redColor] CGColor]);

CGContextFillRect(context, rect);

UIImage \*img = UIGraphicsGetImageFromCurrentImageContext();

UIGraphicsEndImageContext();

UIImage \*flippedImage = [UIImage imageWithCGImage:img.CGImage scale:1.0 orientation: UIImageOrientationDownMirrored];

imageView.image = flippedImage;

44、runtime

- (NSArray \*)allPropertyNames:(Class)aClass

{

unsigned count;

objc\_property\_t \*properties = class\_copyPropertyList(aClass, &count);

NSMutableArray \*rv = [NSMutableArray array];

unsigned i;

for (i = 0; i < count; i++)

{

objc\_property\_t property = properties[i];

NSString \*name = [NSString stringWithUTF8String:property\_getName(property)];

[rv addObject:name];

}

free(properties);

return rv;

}

45、pushmodal()

- (void)push

{

TestViewController \*vc = [[TestViewController alloc] init];

vc.view.backgroundColor = [UIColor redColor];

CATransition\* transition = [CATransition animation];

transition.duration = 0.4f;

transition.type = kCATransitionMoveIn;

transition.subtype = kCATransitionFromTop;

[self.navigationController.view.layer addAnimation:transition forKey:kCATransition];

[self.navigationController pushViewController:vc animated:NO];

}

- (void)pop

{

CATransition\* transition = [CATransition animation];

transition.duration = 0.4f;

transition.type = kCATransitionReveal;

transition.subtype = kCATransitionFromBottom;

[self.navigationController.view.layer addAnimation:transition forKey:kCATransition];

[self.navigationController popViewControllerAnimated:NO];

}

#import "UIImage+BHBEffects.h"

#import <Accelerate/Accelerate.h>

#import <float.h>

@implementation UIImage (BHBEffects)

- (UIImage \*)bhb\_applyBlurWithRadius:(CGFloat)blurRadius tintColor:(UIColor \*)tintColor saturationDeltaFactor:(CGFloat)saturationDeltaFactor maskImage:(UIImage \*)maskImage

{

if (self.size.width < 1 || self.size.height < 1) {

return nil;

}

if (!self.CGImage) {

return nil;

}

if (maskImage && !maskImage.CGImage) {

return nil;

}

CGRect imageRect = { CGPointZero, self.size };

UIImage \*effectImage = self;

BOOL hasBlur = blurRadius > \_\_FLT\_EPSILON\_\_;

BOOL hasSaturationChange = fabs(saturationDeltaFactor - 1.) > \_\_FLT\_EPSILON\_\_;

if (hasBlur || hasSaturationChange) {

UIGraphicsBeginImageContextWithOptions(self.size, NO, [[UIScreen mainScreen] scale]);

CGContextRef effectInContext = UIGraphicsGetCurrentContext();

CGContextScaleCTM(effectInContext, 1.0, -1.0);

CGContextTranslateCTM(effectInContext, 0, -self.size.height);

CGContextDrawImage(effectInContext, imageRect, self.CGImage);

vImage\_Buffer effectInBuffer;

effectInBuffer.data = CGBitmapContextGetData(effectInContext);

effectInBuffer.width = CGBitmapContextGetWidth(effectInContext);

effectInBuffer.height = CGBitmapContextGetHeight(effectInContext);

effectInBuffer.rowBytes = CGBitmapContextGetBytesPerRow(effectInContext);

UIGraphicsBeginImageContextWithOptions(self.size, NO, [[UIScreen mainScreen] scale]);

CGContextRef effectOutContext = UIGraphicsGetCurrentContext();

vImage\_Buffer effectOutBuffer;

effectOutBuffer.data = CGBitmapContextGetData(effectOutContext);

effectOutBuffer.width = CGBitmapContextGetWidth(effectOutContext);

effectOutBuffer.height = CGBitmapContextGetHeight(effectOutContext);

effectOutBuffer.rowBytes = CGBitmapContextGetBytesPerRow(effectOutContext);

if (hasBlur) {

CGFloat inputRadius = blurRadius \* [[UIScreen mainScreen] scale];

unsigned int radius = floor(inputRadius \* 3. \* sqrt(2 \* M\_PI) / 4 + 0.5);

if (radius % 2 != 1) {

radius += 1;

}

vImageBoxConvolve\_ARGB8888(&effectInBuffer, &effectOutBuffer, NULL, 0, 0, radius, radius, 0, kvImageEdgeExtend);

vImageBoxConvolve\_ARGB8888(&effectOutBuffer, &effectInBuffer, NULL, 0, 0, radius, radius, 0, kvImageEdgeExtend);

vImageBoxConvolve\_ARGB8888(&effectInBuffer, &effectOutBuffer, NULL, 0, 0, radius, radius, 0, kvImageEdgeExtend);

}

BOOL effectImageBuffersAreSwapped = NO;

if (hasSaturationChange) {

CGFloat s = saturationDeltaFactor;

CGFloat floatingPointSaturationMatrix[] = {

0.0722 + 0.9278 \* s, 0.0722 - 0.0722 \* s, 0.0722 - 0.0722 \* s, 0,

0.7152 - 0.7152 \* s, 0.7152 + 0.2848 \* s, 0.7152 - 0.7152 \* s, 0,

0.2126 - 0.2126 \* s, 0.2126 - 0.2126 \* s, 0.2126 + 0.7873 \* s, 0,

0, 0, 0, 1,

};

const int32\_t divisor = 256;

NSUInteger matrixSize = sizeof(floatingPointSaturationMatrix)/sizeof(floatingPointSaturationMatrix[0]);

int16\_t saturationMatrix[matrixSize];

for (NSUInteger i = 0; i < matrixSize; ++i) {

saturationMatrix[i] = (int16\_t)roundf(floatingPointSaturationMatrix[i] \* divisor);

}

if (hasBlur) {

vImageMatrixMultiply\_ARGB8888(&effectOutBuffer, &effectInBuffer, saturationMatrix, divisor, NULL, NULL, kvImageNoFlags);

effectImageBuffersAreSwapped = YES;

}

else {

vImageMatrixMultiply\_ARGB8888(&effectInBuffer, &effectOutBuffer, saturationMatrix, divisor, NULL, NULL, kvImageNoFlags);

}

}

if (!effectImageBuffersAreSwapped)

effectImage = UIGraphicsGetImageFromCurrentImageContext();

UIGraphicsEndImageContext();

if (effectImageBuffersAreSwapped)

effectImage = UIGraphicsGetImageFromCurrentImageContext();

UIGraphicsEndImageContext();

}

UIGraphicsBeginImageContextWithOptions(self.size, NO, [[UIScreen mainScreen] scale]);

CGContextRef outputContext = UIGraphicsGetCurrentContext();

CGContextScaleCTM(outputContext, 1.0, -1.0);

CGContextTranslateCTM(outputContext, 0, -self.size.height);

CGContextDrawImage(outputContext, imageRect, self.CGImage);

if (hasBlur) {

CGContextSaveGState(outputContext);

if (maskImage) {

CGContextClipToMask(outputContext, imageRect, maskImage.CGImage);

}

CGContextDrawImage(outputContext, imageRect, effectImage.CGImage);

CGContextRestoreGState(outputContext);

}

if (tintColor) {

CGContextSaveGState(outputContext);

CGContextSetFillColorWithColor(outputContext, tintColor.CGColor);

CGContextFillRect(outputContext, imageRect);

CGContextRestoreGState(outputContext);

}

UIImage \*outputImage = UIGraphicsGetImageFromCurrentImageContext();

UIGraphicsEndImageContext();

return outputImage;

}

(void)changeCity {

CityGroupTableViewController \*cityCtrl = [[CityGroupTableViewController alloc]init];

UINavigationController \*nav = [[UINavigationController alloc] initWithRootViewController:cityCtrl];

[self presentViewController:nav animated:YES completion:nil];

\_\_weak typeof(self) weakSelf = self;

[cityCtrl setBlock:^void (NSString \*cityName){

[weakSelf.weatherV.cityBt setTitle:cityName forState:UIControlStateNormal];

[self sendRequestToServer:cityName]

}];

}

- (void)createBackgroundView {

self.backgroudView = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"bg\_normal.jpg"]];

\_backgroudView.frame = self.view.bounds;

[self.view addSubview:self.backgroudView];

}

- (void)getLoactionAndSendRequest

{

\_\_weak typeof(self) weakSelf = self;

[WSLocation getUserLocation:^(double lat, double lon,NSString \*cityName) {

NSLog(@"cityName = %@",cityName);

CLLocation \*location = [[CLLocation alloc]initWithLatitude:lat longitude:lon];

weakSelf.userLocation = location;

[weakSelf sendRequestToServer:cityName];

}];

}

- (void)sendRequestToServer:(NSString \*)cityName {

\_manager = [AFHTTPSessionManager manager];

NSString \*url = [NSString stringWithFormat:@"https:api.thinkpage.cn/v3/weather/daily.json?key=osoydf7ademn8ybv&location=%@&language=zh-Hans&start=0&days=3",cityName];

url = [url stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding];

[\_manager GET:url parameters:nil progress:nil success:^(NSURLSessionDataTask \* \_Nonnull task, id \_Nullable responseObject) {

NSLog(@"response=%@",responseObject);

NSArray \*resultArray = responseObject[@"results"];

for (NSDictionary \*dic in resultArray) {

WeatherModel \*model = [[WeatherModel alloc]init];

model.cityName = dic[@"location"][@"name"];

model.todayDic = (NSDictionary \*)[dic[@"daily"] objectAtIndex:0];

model.tomorrowDic = (NSDictionary \*)[dic[@"daily"] objectAtIndex:1];

model.afterTomorrowDic = (NSDictionary \*)[dic[@"daily"] objectAtIndex:2];

self.weatherV.model = model;

[self addAnimationWithType:[dic[@"daily"] objectAtIndex:0][@"code\_day"]];

}

} failure:^(NSURLSessionDataTask \* \_Nullable task, NSError \* \_Nonnull error) {

}];

}

- (void)changeImageAnimated:(UIImage \*)image {

CATransition \*transition = [CATransition animation];

transition.duration = 1;

transition.timingFunction = [CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionEaseInEaseOut];

transition.type = kCATransitionFade;

[self.backgroudView.layer addAnimation:transition forKey:@"a"];

[self.backgroudView setImage:image];

}

- (void)removeAnimationView {

[self.birdImage removeFromSuperview];

[self.birdRefImage removeFromSuperview];

[self.cloudImageViewF removeFromSuperview];

[self.cloudImageViewS removeFromSuperview];

[self.sunImage removeFromSuperview];

[self.sunshineImage removeFromSuperview];

[self.sunCloudImage removeFromSuperview];

[self.rainCloudImage removeFromSuperview];

for (NSInteger i = 0; i < \_jsonArray.count; i++) {

UIImageView \*rainLineView = (UIImageView \*)[self.view viewWithTag:100+i];

[rainLineView removeFromSuperview];

}

}

- (void)addAnimationWithType:(NSString \*)weatherType{

[self removeAnimationView];

NSInteger type = [weatherType integerValue];

if (type >= 0 && type < 4) {

[self changeImageAnimated:[UIImage imageNamed:@"bg\_sunny\_day.jpg"]];

[self sun];/

}

else if (type >= 4 && type < 10) {

[self changeImageAnimated:[UIImage imageNamed:@"bg\_normal.jpg"]];

[self wind];

}

else if (type >= 10 && type < 20) {

[self changeImageAnimated:[UIImage imageNamed:@"bg\_rain\_day.jpg"]];

[self rain];

}

else if (type >= 20 && type < 26) {

[self changeImageAnimated:[UIImage imageNamed:@"bg\_snow\_night.jpg"]];

}

else if (type >= 26 && type < 30) {

[self changeImageAnimated:[UIImage imageNamed:@"bg\_sunny\_day.jpg"]];

}

else if (type >= 30 && type < 32)

[self changeImageAnimated:[UIImage imageNamed:@"bg\_haze.jpg"]];

}

else if (type >= 32 && type < 37) { /

[self changeImageAnimated:[UIImage imageNamed:@"bg\_sunny\_day.jpg"]];

}

else if (type == 37) {

[self changeImageAnimated:[UIImage imageNamed:@"bg\_fog\_day.jpg"]];

}

else if (type == 38) {

[self changeImageAnimated:[UIImage imageNamed:@"bg\_sunny\_day.jpg"]];

}

else if (type == 99) {

}

[self.view bringSubviewToFront:self.weatherV];

}

- (void)sun {

\_sunImage = [[UIImageView alloc]initWithImage:[UIImage imageNamed:@"ele\_sunnySun"]];

CGRect frameSun = \_sunImage.frame;

frameSun.size = CGSizeMake(200, 200\*579/612.0);

\_sunImage.frame = frameSun;

\_sunImage.center = CGPointMake(kScreenHeight \* 0.1, kScreenHeight \* 0.1);

[self.view addSubview:\_sunImage];

[\_sunImage.layer addAnimation:[self sunshineAnimationWithDuration:40] forKey:nil];

\_sunshineImage = [[UIImageView alloc]initWithImage:[UIImage imageNamed:@"ele\_sunnySunshine"]];

CGRect \_sunImageFrame = \_sunshineImage.frame;

\_sunImageFrame.size = CGSizeMake(400, 400);

\_sunshineImage.frame = \_sunImageFrame;

\_sunshineImage.center = CGPointMake(kScreenHeight \* 0.1, kScreenHeight \* 0.1);

[self.view addSubview:\_sunshineImage];

[\_sunshineImage.layer addAnimation:[self sunshineAnimationWithDuration:40] forKey:nil];

\_sunCloudImage = [[UIImageView alloc]initWithImage:[UIImage imageNamed:@"ele\_sunnyCloud2"]];

CGRect frame = \_sunCloudImage.frame;

frame.size = CGSizeMake(kScreenHeight \*0.7, kScreenWidth\*0.5);

\_sunCloudImage.frame = frame;

\_sunCloudImage.center = CGPointMake(kScreenWidth \* 0.25, kScreenHeight\*0.5);

[\_sunCloudImage.layer addAnimation:[self birdFlyAnimationWithToValue:@(kScreenWidth+30) duration:50] forKey:nil];

[self.view addSubview:\_sunCloudImage];

}

- (void)wind {

\_birdImage = [[UIImageView alloc]initWithFrame:CGRectMake(-30, kScreenHeight \* 0.2, 70, 50)];

[\_birdImage setAnimationImages:self.imageArr];

\_birdImage.animationRepeatCount = 0;

\_birdImage.animationDuration = 1;

[\_birdImage startAnimating];

[self.view addSubview:\_birdImage];

[\_birdImage.layer addAnimation:[self birdFlyAnimationWithToValue:@(kScreenWidth+30) duration:10 ] forKey:nil];

\_birdRefImage = [[UIImageView alloc]initWithFrame:CGRectMake(-30, kScreenHeight \* 0.8, 70, 50)];

[self.backgroudView addSubview:self.birdRefImage];

[\_birdRefImage setAnimationImages:self.imageArr];

\_birdRefImage.animationRepeatCount = 0;

\_birdRefImage.animationDuration = 1;

\_birdRefImage.alpha = 0.4;

[\_birdRefImage startAnimating];

[\_birdRefImage.layer addAnimation:[self birdFlyAnimationWithToValue:@(kScreenWidth+30) duration:10] forKey:nil];

\_cloudImageViewF = [[UIImageView alloc]initWithImage:[UIImage imageNamed:@"ele\_sunnyCloud2"]];

CGRect frame = \_cloudImageViewF.frame;

frame.size = CGSizeMake(kScreenHeight \*0.7, kScreenWidth\*0.5);

\_cloudImageViewF.frame = frame;

\_cloudImageViewF.center = CGPointMake(kScreenWidth \* 0.25, kScreenHeight\*0.7);

[\_cloudImageViewF.layer addAnimation:[self birdFlyAnimationWithToValue:@(kScreenWidth+30) duration:70] forKey:nil];

[self.view addSubview:\_cloudImageViewF];

\_cloudImageViewS = [[UIImageView alloc]initWithImage:[UIImage imageNamed:@"ele\_sunnyCloud1"]];

\_cloudImageViewS.frame = self.cloudImageViewF.frame;

\_cloudImageViewS.center = CGPointMake(kScreenWidth \* 0.05, kScreenHeight\*0.7);

[\_cloudImageViewS.layer addAnimation:[self birdFlyAnimationWithToValue:@(kScreenWidth+30) duration:70] forKey:nil];

[self.view addSubview:\_cloudImageViewS];

}

- (void)rain {

NSString \*path = [[NSBundle mainBundle] pathForResource:@"rainData.json" ofType:nil];

NSData \*data = [NSData dataWithContentsOfFile:path];

NSDictionary \*dict = [NSJSONSerialization JSONObjectWithData:data options:NSJSONReadingMutableContainers error:nil];

\_jsonArray = dict[@"weather"][@"image"];

for (NSInteger i = 0; i < \_jsonArray.count; i++) {

NSDictionary \*dic = [\_jsonArray objectAtIndex:i];

UIImageView \*rainLineView = [[UIImageView alloc]initWithImage:[UIImage imageNamed:dic[@"-imageName"]]];

rainLineView.tag = 100+i;

NSArray \*sizeArr = [dic[@"-size"] componentsSeparatedByString:@","];

NSArray \*originArr = [dic[@"-origin"] componentsSeparatedByString:@","];

rainLineView.frame = CGRectMake([originArr[0] integerValue]\*widthPix , [originArr[1] integerValue], [sizeArr[0] integerValue], [sizeArr[1] integerValue]);

[self.view addSubview:rainLineView];

[rainLineView.layer addAnimation:[self rainAnimationWithDuration:2+i%5] forKey:nil];

[rainLineView.layer addAnimation:[self rainAlphaWithDuration:2+i%5] forKey:nil];

}

\_rainCloudImage = [[UIImageView alloc]initWithImage:[UIImage imageNamed:@"night\_rain\_cloud"]];

CGRect frame = \_rainCloudImage.frame;

frame.size = CGSizeMake(768/371.0\* kScreenWidth\*0.5, kScreenWidth\*0.5);

\_rainCloudImage.frame = frame;

\_rainCloudImage.center = CGPointMake(kScreenWidth \* 0.25, kScreenHeight\*0.1);

[\_rainCloudImage.layer addAnimation:[self birdFlyAnimationWithToValue:@(kScreenWidth+30) duration:50] forKey:nil];

[self.view addSubview:\_rainCloudImage];

}

- (CABasicAnimation \*)birdFlyAnimationWithToValue:(NSNumber \*)toValue duration:(NSInteger)duration{

CABasicAnimation \*animation = [CABasicAnimation animationWithKeyPath:@"transform.translation.x"];

animation.toValue = toValue;

animation.duration = duration;

animation.removedOnCompletion = NO;

animation.repeatCount = MAXFLOAT;

animation.fillMode = kCAFillModeForwards;

return animation;

}

- (CABasicAnimation \*)sunshineAnimationWithDuration:(NSInteger)duration{

CABasicAnimation\* rotationAnimation = [CABasicAnimation animationWithKeyPath:@"transform.rotation.z"];

rotationAnimation.toValue = [NSNumber numberWithFloat: M\_PI \* 2.0 ];

[CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionEaseInEaseOut];

rotationAnimation.duration = duration;

rotationAnimation.repeatCount = MAXFLOAT;/

rotationAnimation.cumulative = NO;

rotationAnimation.removedOnCompletion = NO;

rotationAnimation.fillMode = kCAFillModeForwards;

return rotationAnimation;

}

- (CABasicAnimation \*)rainAnimationWithDuration:(NSInteger)duration{

CABasicAnimation\* caBaseTransform = [CABasicAnimation animation];

caBaseTransform.duration = duration;

caBaseTransform.keyPath = @"transform";

caBaseTransform.repeatCount = MAXFLOAT;

caBaseTransform.removedOnCompletion = NO;

caBaseTransform.fillMode = kCAFillModeForwards;

caBaseTransform.fromValue = [NSValue valueWithCATransform3D:CATransform3DMakeTranslation(-170, -620, 0)];

caBaseTransform.toValue = [NSValue valueWithCATransform3D:CATransform3DMakeTranslation(kScreenHeight/2.0\*34/124.0, kScreenHeight/2, 0)];

return caBaseTransform;

}

- (CABasicAnimation \*)rainAlphaWithDuration:(NSInteger)duration {

CABasicAnimation \*showViewAnn = [CABasicAnimation animationWithKeyPath:@"opacity"];

showViewAnn.fromValue = [NSNumber numberWithFloat:1.0];

showViewAnn.toValue = [NSNumber numberWithFloat:0.1];

showViewAnn.duration = duration;

showViewAnn.repeatCount = MAXFLOAT;

showViewAnn.fillMode = kCAFillModeForwards;

showViewAnn.timingFunction = [CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionEaseInEaseOut];

showViewAnn.removedOnCompletion = NO;

return showViewAnn;

}

-(NSMutableArray \*)imageArr {

if (!\_imageArr) {

\_imageArr = [NSMutableArray array];

for (int i = 1; i < 9; i++) {

NSString \*fileName = [NSString stringWithFormat:@"ele\_sunnyBird%d.png",i];

NSString \*path = [[NSBundle mainBundle] pathForResource:fileName ofType:nil];

UIImage \*image = [UIImage imageWithContentsOfFile:path];

[\_imageArr addObject:image];

}

}

return \_imageArr;

}

#pragma mark Will Show ViewController

- (void)navigationController:(UINavigationController \*)navigationController willShowViewController:(UIViewController \*)viewController animated:(BOOL)animated {

BOOL needHideNaivgaionBar = NO;

if ([viewController isKindOfClass: [self class]]) {

needHideNaivgaionBar = YES;

}

[self.navigationController setNavigationBarHidden:needHideNaivgaionBar animated: animated];

}

#import "CityGroupTableViewController.h"

@interface CityGroupTableViewController ()

@property (nonatomic, strong) NSArray \*cityGroupArray;

@end

@implementation CityGroupTableViewController

- (NSArray \*)cityGroupArray {

if (!\_cityGroupArray) {

NSString \*plistPath = [[NSBundle mainBundle] pathForResource:@"cityGroups.plist" ofType:nil];

NSArray \*cityGroupArray = [NSArray arrayWithContentsOfFile:plistPath];

NSMutableArray \*mutableArray = [NSMutableArray array];

for (NSDictionary \*dic in cityGroupArray) {

CityGroup \*cityGroup = [CityGroup new];

[cityGroup setValuesForKeysWithDictionary:dic];

[mutableArray addObject:cityGroup];

}

\_cityGroupArray = mutableArray;

}

return \_cityGroupArray;

}

- (void)viewDidLoad {

[super viewDidLoad];

self.navigationItem.title = @"

UIBarButtonItem \*backItem = [[UIBarButtonItem alloc] initWithTitle:@" " style:UIBarButtonItemStyleDone target:self action:@selector(clickBackItem)];

self.navigationItem.leftBarButtonItem = backItem;

}

- (void)clickBackItem {

[self dismissViewControllerAnimated:YES completion:nil];

}

- (NSInteger)numberOfSectionsInTableView:(UITableView \*)tableView {

return self.cityGroupArray.count;

}

- (NSInteger)tableView:(UITableView \*)tableView numberOfRowsInSection:(NSInteger)section {

CityGroup \*cityGroup = self.cityGroupArray[section];

return cityGroup.cities.count;

}

- (UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath {

UITableViewCell \*cell = [tableView dequeueReusableCellWithIdentifier:@"cell"];

if (!cell) {

cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:@"cell"];

}

CityGroup \*cityGroup = self.cityGroupArray[indexPath.section];

cell.textLabel.text = cityGroup.cities[indexPath.row];

return cell;

}

- (NSString \*)tableView:(UITableView \*)tableView titleForHeaderInSection:(NSInteger)section {

CityGroup \*cityGroup = self.cityGroupArray[section];

return cityGroup.title;

}

- (nullable NSArray<NSString \*> \*)sectionIndexTitlesForTableView:(UITableView \*)tableView {

NSMutableArray \*titleMutablArray = [NSMutableArray array];

for (TRCityGroup \*cityGroup in self.cityGroupArray) {

[titleMutablArray addObject:cityGroup.title];

}

return [titleMutablArray copy];

return [self.cityGroupArray valueForKeyPath:@"title"];

}

- (void)tableView:(UITableView \*)tableView didSelectRowAtIndexPath:(NSIndexPath \*)indexPath

{

CityGroup \*cityGroup = self.cityGroupArray[indexPath.section];

NSString \*cityName = cityGroup.cities[indexPath.row];

if (\_block != nil) {

\_block(cityName);

}

[self dismissViewControllerAnimated:YES completion:nil];

[self dismissViewControllerAnimated:YES completion:^{

[self showViewController:[UIApplication sharedApplication] sender:nil];

}];

}

@end

#import "WSLocation.h"

#import <UIKit/UIKit.h>

@interface WSLocation ()<CLLocationManagerDelegate>

@property (nonatomic, strong) CLLocationManager \*manager;

@property (nonatomic, strong) saveLocationBlock savelocationBlock;

@property (nonatomic, strong) NSString \*cityName;

@end

@implementation WSLocation

+ (id)sharedLoactionManager{

static WSLocation \*locationManager = nil;

if (!locationManager) {

locationManager = [[WSLocation alloc]init];

}

return locationManager;

}

- (instancetype)init

{

if (self = [super init]) {

self.manager = [CLLocationManager new];

if ([[UIDevice currentDevice].systemVersion floatValue] >= 8.0) {

[self.manager requestWhenInUseAuthorization];

}

self.manager.delegate = self;

}

return self;

}

+ (void)getUserLocation:(saveLocationBlock)locationBlock

{

WSLocation \*locationMangaer = [WSLocation sharedLoactionManager];

return[locationMangaer getUserLoaction:locationBlock];

}

- (void)getUserLoaction:(saveLocationBlock)locationBlock

{

if (![CLLocationManager locationServicesEnabled]) {

return;

}

\_savelocationBlock = [locationBlock copy];

self.manager.distanceFilter = 100;

[self.manager startUpdatingLocation];

}

#pragma mark -

#pragma mark - CLLocatoinManagerDelegate

- (void)locationManager:(CLLocationManager \*)manager didUpdateLocations:(NSArray<CLLocation \*> \*)locations

{

CLLocation \*location = [locations lastObject];

CLGeocoder \*geocoder = [[CLGeocoder alloc]init];

[geocoder reverseGeocodeLocation:location completionHandler:^(NSArray<CLPlacemark \*> \* \_Nullable placemarks, NSError \* \_Nullable error) {

if (!error) {

NSString \*cityName = placemarks.lastObject.addressDictionary[@"City"];

NSString \*str = [cityName substringToIndex:cityName.length -1];

\_savelocationBlock(location.coordinate.latitude,location.coordinate.longitude,str);

}

}];

}

@end

#import "WeatherModel.h"

@implementation WeatherModel

@end

#import "WeatherView.h"

@implementation WeatherView

#define kScreenWidth [UIScreen mainScreen].bounds.size.width

#define kScreenHeight [UIScreen mainScreen].bounds.size.height

#define widthPix kScreenWidth/320

#define heightPix kScreenHeight/568

- (instancetype)initWithFrame:(CGRect)frame {

self = [super initWithFrame:frame];

if (self) {

self.userInteractionEnabled = YES;

[self creatUI];

}

return self;

}

- (void)creatUI {

\_cityBt = [UIButton buttonWithType:UIButtonTypeCustom];

\_cityBt.frame = CGRectMake(80, 20, kScreenWidth-160, 44);

\_cityBt.tag = 10;

\_cityBt.titleLabel.textAlignment = NSTextAlignmentCenter;

\_cityBt.titleLabel.font = [UIFont boldSystemFontOfSize:30];

[self addSubview:\_cityBt];

UIImageView \*weatherImg = [[UIImageView alloc]initWithFrame:CGRectMake(kScreenWidth/2-50\*widthPix, CGRectGetMaxY(\_cityBt.frame)+30, 100\*widthPix, 100\*heightPix)]

weatherImg.tag = 11;

[self addSubview:weatherImg];

UILabel \*tempLabel = [[UILabel alloc]initWithFrame:CGRectMake(50, CGRectGetMaxY(weatherImg.frame), (kScreenWidth-100), 40\*heightPix)];

tempLabel.font = [UIFont boldSystemFontOfSize:30\*heightPix];

tempLabel.textAlignment = NSTextAlignmentCenter;

tempLabel.textColor = [UIColor whiteColor];

tempLabel.tag = 12;

[self addSubview:tempLabel];

for (NSInteger i = 0; i < 2; i++) {

UILabel \*dayLabel = [[UILabel alloc]initWithFrame:CGRectMake((kScreenWidth/2-100\*widthPix)/2+kScreenWidth/2\*i, CGRectGetMaxY(tempLabel.frame)+50, 100\*widthPix, 30\*heightPix)];

dayLabel.font = [UIFont boldSystemFontOfSize:15\*heightPix];

dayLabel.textColor = [UIColor whiteColor];

dayLabel.textAlignment = NSTextAlignmentCenter;

dayLabel.tag = 20+i;

[self addSubview:dayLabel];

UIImageView \*smallWeatherImg = [[UIImageView alloc]initWithFrame:CGRectMake((kScreenWidth/2-70\*widthPix)/2+kScreenWidth/2\*i, CGRectGetMaxY(dayLabel.frame), 70\*widthPix, 70\*heightPix)];

smallWeatherImg.tag = 30+i;

[self addSubview:smallWeatherImg];

UILabel \*samllTempLabel = [[UILabel alloc]initWithFrame:CGRectMake((kScreenWidth/2-100\*widthPix)/2+kScreenWidth/2\*i, CGRectGetMaxY(smallWeatherImg.frame), 100\*widthPix, 30\*heightPix)];

samllTempLabel.font = [UIFont boldSystemFontOfSize:15\*heightPix];

samllTempLabel.textColor = [UIColor whiteColor];

samllTempLabel.tag = 40+i;

samllTempLabel.textAlignment = NSTextAlignmentCenter;

[self addSubview:samllTempLabel];

UILabel \*windLabel = [[UILabel alloc]initWithFrame:CGRectMake((kScreenWidth/2-100\*widthPix)/2+kScreenWidth/2\*i, CGRectGetMaxY(samllTempLabel.frame), 100\*widthPix, 30\*heightPix)];

windLabel.font = [UIFont boldSystemFontOfSize:15\*heightPix];

windLabel.textColor = [UIColor whiteColor];

windLabel.tag = 50+i;

windLabel.textAlignment = NSTextAlignmentCenter;

[self addSubview:windLabel];

}

}

-(void)setModel:(WeatherModel \*)model {

\_model = model;

UIButton \*cityBt = (UIButton \*)[self viewWithTag:10];

UIImageView \*weatherImg = (UIImageView \*)[self viewWithTag:11];

UILabel \*tempLabel = (UILabel \*)[self viewWithTag:12];

UILabel \*dayLabel1 = (UILabel \*)[self viewWithTag:20];

UILabel \*dayLabel2 = (UILabel \*)[self viewWithTag:21];

UIImageView \*smallWeatherImg1 = (UIImageView \*)[self viewWithTag:30];

UIImageView \*smallWeatherImg2 = (UIImageView \*)[self viewWithTag:31];

UILabel \*samllTempLabel1 = (UILabel \*)[self viewWithTag:40];

UILabel \*samllTempLabel2 = (UILabel \*)[self viewWithTag:41];

UILabel \*windLabel1 = (UILabel \*)[self viewWithTag:50];

UILabel \*windLabel2 = (UILabel \*)[self viewWithTag:51];

[cityBt setTitle:model.cityName forState:UIControlStateNormal];

weatherImg.image = [UIImage imageNamed:[model.todayDic objectForKey:@"code\_day"]];

[self changeImageAnimatedWithView:weatherImg AndImage: [UIImage imageNamed:[model.todayDic objectForKey:@"code\_day"]]];

tempLabel.text = [NSString stringWithFormat:@"%@℃/%@℃",[model.todayDic objectForKey:@"high"],[model.todayDic objectForKey:@"low"]];

dayLabel1.text =[model.tomorrowDic objectForKey:@"date"];

smallWeatherImg1.image = [UIImage imageNamed:[model.tomorrowDic objectForKey:@"code\_day"]];

[self changeImageAnimatedWithView:smallWeatherImg1 AndImage: [UIImage imageNamed:[model.tomorrowDic objectForKey:@"code\_day"]]];

samllTempLabel1.text = [NSString stringWithFormat:@"%@℃/%@℃",[model.tomorrowDic objectForKey:@"high"],[model.tomorrowDic objectForKey:@"low"]];

windLabel1.text = [NSString stringWithFormat:@"%@:%@",[model.tomorrowDic objectForKey:@"wind\_direction" ],[model.tomorrowDic objectForKey:@"wind\_speed"]];

dayLabel2.text =[model.afterTomorrowDic objectForKey:@"date"];

smallWeatherImg2.image = [UIImage imageNamed:[model.afterTomorrowDic objectForKey:@"code\_day"]];

[self changeImageAnimatedWithView:smallWeatherImg2 AndImage: [UIImage imageNamed:[model.afterTomorrowDic objectForKey:@"code\_day"]]];

samllTempLabel2.text = [NSString stringWithFormat:@"%@℃/%@℃",[model.afterTomorrowDic objectForKey:@"high"],[model.afterTomorrowDic objectForKey:@"low"]];

windLabel2.text = [NSString stringWithFormat:@"%@:%@",[model.afterTomorrowDic objectForKey:@"wind\_direction" ],[model.afterTomorrowDic objectForKey:@"wind\_speed"]];

}

- (void)changeImageAnimatedWithView:(UIImageView \*)imageV AndImage:(UIImage \*)image {

CATransition \*transition = [CATransition animation];

transition.duration = 1;

transition.timingFunction = [CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionEaseInEaseOut];

transition.type = kCATransitionFade;

[imageV.layer addAnimation:transition forKey:@"a"];

[imageV setImage:image];

}

import UIKit

import Mattress

@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate {

var window: UIWindow?

func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {

let kB = 1024

let MB = 1024 \* kB

let GB = 1024 \* MB

let isOfflineHandler: (() -> Bool) = {

/\*

We are returning true here for demo purposes only.

You should use Reachability or another method for determining whether the user is

offline and return the appropriate value

\*/

return true

}

let urlCache = Mattress.URLCache(memoryCapacity: 20 \* MB, diskCapacity: 20 \* MB, diskPath: nil,

mattressDiskCapacity: 1 \* GB, mattressDiskPath: nil, mattressSearchPathDirectory: .DocumentDirectory,

isOfflineHandler: isOfflineHandler)

NSURLCache.setSharedURLCache(urlCache)

return true

}

}

{

"images" : [

{

"idiom" : "iphone",

"size" : "29x29",

"scale" : "2x"

},

{

"idiom" : "iphone",

"size" : "29x29",

"scale" : "3x"

},

{

"idiom" : "iphone",

"size" : "40x40",

"scale" : "2x"

},

{

"idiom" : "iphone",

"size" : "40x40",

"scale" : "3x"

},

{

"idiom" : "iphone",

"size" : "60x60",

"scale" : "2x"

},

{

"idiom" : "iphone",

"size" : "60x60",

"scale" : "3x"

},

{

"idiom" : "ipad",

"size" : "29x29",

"scale" : "1x"

},

{

"idiom" : "ipad",

"size" : "29x29",

"scale" : "2x"

},

{

"idiom" : "ipad",

"size" : "40x40",

"scale" : "1x"

},

{

"idiom" : "ipad",

"size" : "40x40",

"scale" : "2x"

},

{

"idiom" : "ipad",

"size" : "76x76",

"scale" : "1x"

},

{

"idiom" : "ipad",

"size" : "76x76",

"scale" : "2x"

}

],

"info" : {

"version" : 1,

"author" : "xcode"

}

}

import UIKit

import Mattress

class ViewController: UIViewController {

@IBOutlet var webView: UIWebView!

let urlToCache = NSURL(string: "https:www.google.com")

@IBAction func cachePage() {

NSLog("Caching page")

if let

cache = NSURLCache.sharedURLCache() as? Mattress.URLCache,

urlToCache = urlToCache

{

cache.diskCacheURL(urlToCache, loadedHandler: { (webView) -> (Bool) in

let state = webView.stringByEvaluatingJavaScriptFromString("document.readyState")

if state == "complete" {

Loading is done once we've returned true

return true

}

return false

}, completeHandler: { () -> Void in

NSLog("Finished caching")

}, failureHandler: { (error) -> Void in

NSLog("Error caching: %@", error)

})

}

}

@IBAction func loadPage() {

if let urlToCache = urlToCache {

let request = NSURLRequest(URL: urlToCache)

webView.loadRequest(request)

}

}

}

@end

MODULEMAP\_FILE[sdk=iphoneos\*] = $(SRCROOT)/CommonCrypto/iphoneos/module.modulemap

MODULEMAP\_FILE[sdk=iphonesimulator\*] = $(SRCROOT)/CommonCrypto/iphonesimulator/module.modulemap

MODULEMAP\_FILE[sdk=macosx\*] = $(SRCROOT)/CommonCrypto/macosx/module.modulemap

module CommonCrypto [system] {

header "/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/include/CommonCrypto/CommonCrypto.h"

export \*

}

import XCTest

class DiskCacheTests: XCTestCase {

override func setUp() {

Ensure plist on disk is reset

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 0)

if let path = diskCache.diskPathForPropertyList()?.path {

try! NSFileManager.defaultManager().removeItemAtPath(path)

}

}

func testDiskPathForRequestIsDeterministic() {

let url = NSURL(string: "foo:bar")!

let request1 = NSURLRequest(URL: url)

let request2 = NSURLRequest(URL: url)

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 1024)

let path = diskCache.diskPathForRequest(request1)

XCTAssertNotNil(path, "Path for request was nil")

XCTAssert(path == diskCache.diskPathForRequest(request2), "Requests for the same url did not match")

}

func testDiskPathsForDifferentRequestsAreNotEqual() {

let url1 = NSURL(string: "foo:bar")!

let url2 = NSURL(string: "foo:baz")!

let request1 = NSURLRequest(URL: url1)

let request2 = NSURLRequest(URL: url2)

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 1024)

let path1 = diskCache.diskPathForRequest(request1)

let path2 = diskCache.diskPathForRequest(request2)

XCTAssert(path1 != path2, "Paths should not be matching")

}

func testStoreCachedResponseReturnsTrue() {

let url = NSURL(string: "foo:bar")!

let request = NSURLRequest(URL: url)

let cachedResponse = cachedResponseWithDataString("hello, world", request: request, userInfo: ["foo" : "bar"])

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 1024 \* 1024)

let success = diskCache.storeCachedResponse(cachedResponse, forRequest: request)

XCTAssert(success, "Did not save the cached response to disk")

}

func testCachedResponseCanBeArchivedAndUnarchivedWithoutDataLoss() {

Saw some old reports of keyedArchiver not working well with NSCachedURLResponse

so this is just here to make sure things are working on Apple's end

let url = NSURL(string: "foo:bar")!

let request = NSURLRequest(URL: url)

let cachedResponse = cachedResponseWithDataString("hello, world", request: request, userInfo: ["foo" : "bar"])

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 1024 \* 1024)

diskCache.storeCachedResponse(cachedResponse, forRequest: request)

let restored = diskCache.cachedResponseForRequest(request)

if let restored = restored {

assertCachedResponsesAreEqual(response1: restored, response2: cachedResponse)

} else {

XCTFail("Did not get back a cached response from diskCache")

}

}

func testCacheReturnsCorrectResponseForRequest() {

let url1 = NSURL(string: "foo:bar")!

let request1 = NSURLRequest(URL: url1)

let cachedResponse1 = cachedResponseWithDataString("hello, world", request: request1, userInfo: ["foo" : "bar"])

let url2 = NSURL(string: "foo:baz")!

let request2 = NSURLRequest(URL: url2)

let cachedResponse2 = cachedResponseWithDataString("goodbye, cruel world", request: request2, userInfo: ["baz" : "qux"])

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 1024 \* 1024)

let success1 = diskCache.storeCachedResponse(cachedResponse1, forRequest: request1)

let success2 = diskCache.storeCachedResponse(cachedResponse2, forRequest: request2)

XCTAssert(success1 && success2, "The responses did not save properly")

let restored1 = diskCache.cachedResponseForRequest(request1)

if let restored = restored1 {

assertCachedResponsesAreEqual(response1: restored, response2: cachedResponse1)

} else {

XCTFail("Did not get back a cached response from diskCache")

}

let restored2 = diskCache.cachedResponseForRequest(request2)

if let restored = restored2 {

assertCachedResponsesAreEqual(response1: restored, response2: cachedResponse2)

} else {

XCTFail("Did not get back a cached response from diskCache")

}

}

func testStoredRequestIncrementsDiskCacheSizeByFilesize() {

let url = NSURL(string: "foo:bar")!

let request = NSURLRequest(URL: url)

let cachedResponse = cachedResponseWithDataString("hello, world", request: request, userInfo: ["foo" : "bar"])

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 1024 \* 1024)

XCTAssert(diskCache.currentSize == 0, "Current size should start zeroed out")

diskCache.storeCachedResponse(cachedResponse, forRequest: request)

if let path = diskCache.diskPathForRequest(request)?.path {

let attributes = try! NSFileManager.defaultManager().attributesOfItemAtPath(path)

if let fileSize = attributes[NSFileSize] as? NSNumber {

let size = fileSize.integerValue

XCTAssert(diskCache.currentSize == size, "Disk cache size was not incremented by the correct amount")

} else {

XCTFail("Could not get fileSize from attribute")

}

} else {

XCTFail("Did not get a valid path for request")

}

}

func testStoringARequestIncreasesTheRequestCachesSize() {

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 1024)

let url = NSURL(string: "foo:bar")!

let request = NSURLRequest(URL: url)

let cachedResponse = cachedResponseWithDataString("hello, world", request: request, userInfo: nil)

XCTAssert(diskCache.requestCaches.count == 0, "Should not start with any request caches")

diskCache.storeCachedResponse(cachedResponse, forRequest: request)

XCTAssert(diskCache.requestCaches.count == 1, "requestCaches should be 1")

}

func testFilesAreRemovedInChronOrderWhenCacheExceedsMaxSize() {

let cacheSize = 1024 \* 1024 1MB so dataSize dwarfs the size of encoding the object itself

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: cacheSize)

let dataSize = cacheSize/3 + 1

let url1 = NSURL(string: "foo:bar")!

let request1 = NSURLRequest(URL: url1)

let cachedResponse1 = cachedResponseWithDataOfSize(dataSize, request: request1, userInfo: nil)

let url2 = NSURL(string: "bar:baz")!

let request2 = NSURLRequest(URL: url2)

let cachedResponse2 = cachedResponseWithDataOfSize(dataSize, request: request2, userInfo: nil)

let url3 = NSURL(string: "baz:qux")!

let request3 = NSURLRequest(URL: url3)

let cachedResponse3 = cachedResponseWithDataOfSize(dataSize, request: request2, userInfo: nil)

diskCache.storeCachedResponse(cachedResponse1, forRequest: request1)

diskCache.storeCachedResponse(cachedResponse2, forRequest: request2)

diskCache.storeCachedResponse(cachedResponse3, forRequest: request3) This should cause response1 to be removed

let requestCaches = [diskCache.hashForURLString(url2.absoluteString)!, diskCache.hashForURLString(url3.absoluteString)!]

XCTAssert(diskCache.requestCaches == requestCaches, "Request caches did not match expectations")

}

func testPlistIsUpdatedAfterStoringARequest() {

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 1024)

let url = NSURL(string: "foo:bar")!

let request = NSURLRequest(URL: url)

let cachedResponse = cachedResponseWithDataString("hello, world", request: request, userInfo: nil)

diskCache.storeCachedResponse(cachedResponse, forRequest: request)

let data = NSKeyedArchiver.archivedDataWithRootObject(cachedResponse)

let expectedSize = data.length

let expectedRequestCaches = diskCache.requestCaches

if let plistPath = diskCache.diskPathForPropertyList()?.path {

if NSFileManager.defaultManager().fileExistsAtPath(plistPath) {

if let dict = NSDictionary(contentsOfFile: plistPath) {

if let currentSize = dict.valueForKey(DiskCache.DictionaryKeys.maxCacheSize) as? Int {

XCTAssert(currentSize == expectedSize, "Current size did not match expected value")

} else {

XCTFail("Plist did not have currentSize property")

}

if let requestCaches = dict.valueForKey(DiskCache.DictionaryKeys.requestsFilenameArray) as? [String] {

XCTAssert(requestCaches == expectedRequestCaches, "Request caches did not match expected value")

} else {

XCTFail("Plist did not have requestCaches property")

}

}

} else {

XCTFail("Could not find plist")

}

} else {

XCTFail("Could not get plist path")

}

}

func testDiskCacheRestoresPropertiesFromPlist() {

var expectedRequestCaches: [String] = []

var expectedSize = 0

autoreleasepool { [unowned self] in

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 1024)

let url = NSURL(string: "foo:bar")!

let request = NSURLRequest(URL: url)

let cachedResponse = self.cachedResponseWithDataString("hello, world", request: request, userInfo: nil)

diskCache.storeCachedResponse(cachedResponse, forRequest: request)

expectedRequestCaches = diskCache.requestCaches

expectedSize = diskCache.currentSize

}

let newDiskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 1024)

XCTAssert(newDiskCache.currentSize == expectedSize, "Size property did not match expectations")

XCTAssert(newDiskCache.requestCaches == expectedRequestCaches, "RequestCaches did not match expectations")

}

func testRequestCacheIsRemovedFromDiskAfterTrim() {

let cacheSize = 1024 \* 1024 1MB so dataSize dwarfs the size of encoding the object itself

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: cacheSize)

let dataSize = cacheSize/3 + 1

let url1 = NSURL(string: "foo:bar")!

let request1 = NSURLRequest(URL: url1)

let cachedResponse1 = cachedResponseWithDataOfSize(dataSize, request: request1, userInfo: nil)

let pathForResponse = (diskCache.diskPathForRequest(request1)?.path)!

let url2 = NSURL(string: "bar:baz")!

let request2 = NSURLRequest(URL: url2)

let cachedResponse2 = cachedResponseWithDataOfSize(dataSize, request: request2, userInfo: nil)

let url3 = NSURL(string: "baz:qux")!

let request3 = NSURLRequest(URL: url3)

let cachedResponse3 = cachedResponseWithDataOfSize(dataSize, request: request2, userInfo: nil)

diskCache.storeCachedResponse(cachedResponse1, forRequest: request1)

diskCache.storeCachedResponse(cachedResponse2, forRequest: request2)

var isFileOnDisk = NSFileManager.defaultManager().fileExistsAtPath(pathForResponse)

XCTAssert(isFileOnDisk, "File should be on disk")

diskCache.storeCachedResponse(cachedResponse3, forRequest: request3) This should cause response1 to be removed

isFileOnDisk = NSFileManager.defaultManager().fileExistsAtPath(pathForResponse)

XCTAssertFalse(isFileOnDisk, "File should no longer be on disk")

}

func testiOS7CanSaveCachedResponse() {

let cacheSize = 1024 \* 1024 1MB so dataSize dwarfs the size of encoding the object itself

let dataSize = cacheSize/3 + 1

let diskCache = DiskCacheiOS7(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: cacheSize)

let url = NSURL(string: "foo:bar")!

let request = NSURLRequest(URL: url)

let userInfo = ["foo" : "bar"]

let cachedResponse = cachedResponseWithDataOfSize(dataSize, request: request, userInfo: userInfo)

diskCache.storeCachedResponse(cachedResponse, forRequest: request)

let basePath = (diskCache.diskPathForRequest(request)?.path)!

let responsePath = diskCache.hashForResponseFromHash(basePath)

let dataPath = diskCache.hashForDataFromHash(basePath)

let userInfoPath = diskCache.hashForUserInfoFromHash(basePath)

XCTAssert(NSFileManager.defaultManager().fileExistsAtPath(responsePath), "Response file should be on disk")

XCTAssert(NSFileManager.defaultManager().fileExistsAtPath(dataPath), "Data file should be on disk")

XCTAssert(NSFileManager.defaultManager().fileExistsAtPath(userInfoPath), "User Info file should be on disk")

}

func testiOS7CanRestoreCachedResponse() {

let cacheSize = 1024 \* 1024 1MB so dataSize dwarfs the size of encoding the object itself

let dataSize = cacheSize/3 + 1

let diskCache = DiskCacheiOS7(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: cacheSize)

let url = NSURL(string: "foo:bar")!

let request = NSURLRequest(URL: url)

let userInfo = ["foo" : "bar"]

let cachedResponse = cachedResponseWithDataOfSize(dataSize, request: request, userInfo: userInfo)

diskCache.storeCachedResponse(cachedResponse, forRequest: request)

if let response = diskCache.cachedResponseForRequest(request) {

assertCachedResponsesAreEqual(response1: response, response2: cachedResponse)

} else {

XCTFail("Could not retrieve cached response")

}

}

func testClearCacheRemovesAnyExistingRequests() {

let url = NSURL(string: "foo:bar")!

let request = NSURLRequest(URL: url)

let userInfo = ["foo" : "bar"]

let dataSize = 1

let cachedResponse = cachedResponseWithDataOfSize(dataSize, request: request, userInfo: userInfo)

let diskCache = DiskCache(path: "test", searchPathDirectory: .DocumentDirectory, maxCacheSize: 1024\*1024)

diskCache.storeCachedResponse(cachedResponse, forRequest: request)

diskCache.clearCache()

XCTAssertFalse(diskCache.hasCacheForRequest(request))

}

Mark: - Test Helpers

func assertCachedResponsesAreEqual(response1 response1 : NSCachedURLResponse, response2: NSCachedURLResponse) {

XCTAssert(response1.data == response2.data, "Data did not match")

XCTAssert(response1.response.URL == response2.response.URL, "Response did not match")

if response1.userInfo != nil && response2.userInfo != nil {

XCTAssert(response1.userInfo!.description == response2.userInfo!.description, "userInfo didn't match")

} else if !(response1.userInfo == nil && response2.userInfo == nil) {

XCTFail("userInfo did not match")

}

}

func cachedResponseWithDataString(dataString: String, request: NSURLRequest, userInfo: [NSObject : AnyObject]?) -> NSCachedURLResponse {

let data = dataString.dataUsingEncoding(NSUTF8StringEncoding, allowLossyConversion: false)!

let response = NSURLResponse(URL: request.URL!, MIMEType: "text/html", expectedContentLength: data.length, textEncodingName: nil)

let cachedResponse = NSCachedURLResponse(response: response, data: data, userInfo: userInfo, storagePolicy: .Allowed)

return cachedResponse

}

func cachedResponseWithDataOfSize(dataSize: Int, request: NSURLRequest, userInfo: [NSObject : AnyObject]?) -> NSCachedURLResponse {

var bytes: [UInt32] = Array(count: dataSize, repeatedValue: 1)

let data = NSData(bytes: &bytes, length: dataSize)

let response = NSURLResponse(URL: request.URL!, MIMEType: "text/html", expectedContentLength: data.length, textEncodingName: nil)

let cachedResponse = NSCachedURLResponse(response: response, data: data, userInfo: userInfo, storagePolicy: .Allowed)

return cachedResponse

}

}

class DiskCacheiOS7: DiskCache {

override var isAtLeastiOS8: Bool {

return false

}

}

import Foundation

import UIKit

/\*\*

DiskCache is a NSURLCache replacement that will store

and retrieve NSCachedURLResponses to disk.

\*/

class DiskCache {

/\*\*

Keys used to store properties in the plist.

\*/

struct DictionaryKeys {

static let maxCacheSize = "maxCacheSize"

static let requestsFilenameArray = "requestsFilenameArray"

}

MARK: - Properties

var isAtLeastiOS8: Bool {

struct Static {

static var onceToken : dispatch\_once\_t = 0

static var value: Bool = false

}

dispatch\_once(&Static.onceToken) {

Static.value = (UIDevice.currentDevice().systemVersion as NSString).doubleValue >= 8.0

}

return Static.value

}

/ Filesystem path where the cache is stored

private let path: String

/ Search path for the disk cache location

private let searchPathDirectory: NSSearchPathDirectory

/ Size limit for the disk cache

private let maxCacheSize: Int

/ Provides locking for multi-threading sensitive operations

private let lockObject = NSObject()

/ Current disk cache size

var currentSize = 0

/ File paths for requests cached on disk

var requestCaches: [String] = []

Mark: - Instance methods

/\*\*

Initializes a new DiskCache

:param: path The path of the location on disk that should be used

to store requests. This MUST be unique for each DiskCache instance.

Otherwise, you will have a hard time debugging crashes.

:param: searchPathDirectory The NSSearchPathDirectory that will be

used to find the location at which to store requests.

:param: maxCacheSize The size limit of this diskCache. When the size

of the requests exceeds this amount, older requests will be removed.

No requests that are larger than this size will even attempt to be

stored.

\*/

init(path: String?, searchPathDirectory: NSSearchPathDirectory, maxCacheSize: Int) {

self.path = path ?? "mattress/"

self.searchPathDirectory = searchPathDirectory

self.maxCacheSize = maxCacheSize

loadPropertiesFromDisk()

}

/\*\*

Loads appropriate properties from the plist to restore

this cache from disk.

\*/

private func loadPropertiesFromDisk() {

synchronized(lockObject) { () -> Void in

if let plistPath = self.diskPathForPropertyList()?.path {

if !NSFileManager.defaultManager().fileExistsAtPath(plistPath) {

self.persistPropertiesToDisk()

} else {

if let dict = NSDictionary(contentsOfFile: plistPath) {

if let currentSize = dict.valueForKey(DictionaryKeys.maxCacheSize) as? Int {

self.currentSize = currentSize

}

if let requestCaches = dict.valueForKey(DictionaryKeys.requestsFilenameArray) as? [String] {

self.requestCaches = requestCaches

}

}

}

}

}

}

/\*\*

Saves appropriate properties to a plist to save

this cache to disk.

\*/

private func persistPropertiesToDisk() {

synchronized(lockObject) { () -> Void in

if let plistPath = self.diskPathForPropertyList()?.path {

let dict = self.propertiesDictionary()

(dict as NSDictionary).writeToFile(plistPath, atomically: true)

}

return

}

}

func clearCache() {

if let path = diskPath()?.path {

do {

try NSFileManager.defaultManager().removeItemAtPath(path)

requestCaches = []

currentSize = 0

} catch {

NSLog("Error clearing cache")

}

} else {

NSLog("Error clearing cache")

}

}

/\*\*

Keeps removing the oldest request until our

currentSize is not greater than the maxCacheSize.

Clears the cache on any failures.

\*/

private func trimCacheIfNeeded() {

while currentSize > maxCacheSize && !requestCaches.isEmpty {

let lastCurrentSize = currentSize

let fileName = requestCaches.first

if let

fileName = fileName,

path = diskPathForRequestCacheNamed(fileName)?.path

{

var attributes: [String : AnyObject]?

do {

try attributes = NSFileManager.defaultManager().attributesOfItemAtPath(path)

try NSFileManager.defaultManager().removeItemAtPath(path)

} catch {

NSLog("Error getting attributes of or deleting item at path \(path)")

attributes = nil

clearCache()

return

}

if let

attributes = attributes,

fileSize = attributes[NSFileSize] as? NSNumber

{

let size = fileSize.integerValue

currentSize -= size

}

if let index = requestCaches.indexOf(fileName) {

requestCaches.removeAtIndex(index)

}

} else {

NSLog("Error getting filename or path")

clearCache()

return

}

if currentSize == lastCurrentSize {

NSLog("Error: current cache size did not decrement")

clearCache()

return

}

}

}

/\*\*

Creates a dictionary that will be used to store

this diskCache's properties to disk.

:returns: A dictionary of the cache's properties

\*/

private func propertiesDictionary() -> [String: AnyObject] {

var dict = [String: AnyObject]()

dict[DictionaryKeys.maxCacheSize] = currentSize

dict[DictionaryKeys.requestsFilenameArray] = requestCaches

return dict

}

/\*\*

Functions much like NSURLCache's similarly named method, storing a

response and request to disk only.

:param: cachedResponse an NSCachedURLResponse to persist to disk.

:param: forRequest an NSURLRequest to associate the cachedResponse with.

:returns: A Bool representing whether or not we successfully

stored the response to disk.

\*/

func storeCachedResponse(cachedResponse: NSCachedURLResponse, forRequest request: NSURLRequest) -> Bool {

var success = false

synchronized(lockObject) { () -> Void in

if let hash = self.hashForRequest(request) {

if self.isAtLeastiOS8 {

success = self.saveObject(cachedResponse, withHash: hash)

} else {

success = self.storeCachedResponsePieces(cachedResponse, withHash: hash)

}

}

}

return success

}

/\*\*

Stores components of the NSCachedURLResponse to disk individually to

work around iOS 7 not properly storing the response to disk with its

data and userInfo.

NOTE: Storage policy is not stored because it is irrelevant to Mattress

cached responses.

:param: cachedResponse an NSCachedURLResponse to persist to disk.

:param: hash The hash associated with the NSCachedURLResponse.

:returns: A Bool representing whether or not we successfully

stored the response to disk.

\*/

private func storeCachedResponsePieces(cachedResponse: NSCachedURLResponse, withHash hash: String) -> Bool {

var success = true

synchronized(lockObject) { () -> Void in

let responseHash = self.hashForResponseFromHash(hash)

success = success && self.saveObject(cachedResponse.response, withHash: responseHash)

let dataHash = self.hashForDataFromHash(hash)

success = success && self.saveObject(cachedResponse.data, withHash: dataHash)

if let userInfo = cachedResponse.userInfo {

if !userInfo.isEmpty {

let userInfoHash = self.hashForUserInfoFromHash(hash)

success = success && self.saveObject(userInfo, withHash: userInfoHash)

}

}

}

return success

}

/\*\*

Saves an archived object's data to disk with the hash it should be

associated with. This will only store the request if it can fit in

our max cache size, and will empty out older cached items if it

needs to to make room.

:param: object The NSCoding compliant object to save.

:param: hash The hash associated with that object.

:returns: A Bool indicating that the saves were successful.

\*/

private func saveObject(object: NSCoding, withHash hash: String) -> Bool {

var success = false

synchronized(lockObject) { () -> Void in

if let path = self.diskPathForRequestCacheNamed(hash)?.path {

let data = NSKeyedArchiver.archivedDataWithRootObject(object)

if data.length < self.maxCacheSize {

self.currentSize += data.length

var index = -1

for i in 0..<self.requestCaches.count {

if self.requestCaches[i] == hash {

index = i

break

}

}

if index != -1 {

self.requestCaches.removeAtIndex(index)

}

self.requestCaches.append(hash)

self.trimCacheIfNeeded()

self.persistPropertiesToDisk()

success = true

do {

try data.writeToFile(path, options: [])

} catch {

success = false

NSLog("Error writing request to disk: \(error)")

}

}

}

}

return success

}

/\*\*

Functions much like NSURLCache's method of the same signature.

An NSCachedURLResponse associated with the specified NSURLRequest

will be returned.

:param: request The request.

:returns: The cached response.

\*/

func cachedResponseForRequest(request: NSURLRequest) -> NSCachedURLResponse? {

var response: NSCachedURLResponse?

synchronized(lockObject) { () -> Void in

if let path = self.diskPathForRequest(request)?.path {

if self.isAtLeastiOS8 {

response = NSKeyedUnarchiver.unarchiveObjectWithFile(path) as? NSCachedURLResponse

} else {

response = self.cachedResponseFromPiecesForRequest(request)

}

}

}

return response

}

/\*\*

This will simply check if a response exists in the cache for the

specified request.

\*/

internal func hasCachedResponseForRequest(request: NSURLRequest) -> Bool {

if let path = self.diskPathForRequest(request)?.path {

return NSFileManager.defaultManager().fileExistsAtPath(path)

}

return false

}

/\*\*

Will create the cachedResponse from its response, data and

userInfo. This is only used to workaround the bug in iOS 7

preventing us from just saving the cachedResponse itself.

:param: request The request.

:returns: The cached response.

\*/

private func cachedResponseFromPiecesForRequest(request: NSURLRequest) -> NSCachedURLResponse? {

var cachedResponse: NSCachedURLResponse? = nil

synchronized(lockObject) { () -> Void in

var response: NSURLResponse? = nil

var data: NSData? = nil

var userInfo: [NSObject : AnyObject]? = nil

if let basePath = self.diskPathForRequest(request)?.path {

let responsePath = self.hashForResponseFromHash(basePath)

response = NSKeyedUnarchiver.unarchiveObjectWithFile(responsePath) as? NSURLResponse

let dataPath = self.hashForDataFromHash(basePath)

data = NSKeyedUnarchiver.unarchiveObjectWithFile(dataPath) as? NSData

let userInfoPath = self.hashForUserInfoFromHash(basePath)

userInfo = NSKeyedUnarchiver.unarchiveObjectWithFile(userInfoPath) as? [NSObject : AnyObject]

}

if let

response = response,

data = data

{

cachedResponse = NSCachedURLResponse(response: response, data: data, userInfo: userInfo, storagePolicy: .Allowed)

}

}

return cachedResponse

}

/\*\*

hasCacheForRequest: returns a Bool indicating whether

this diskCache has a cachedResponse associated with the

specified NSURLRequest.

:param: The request.

:returns: A boolean indicating whether the cache has a

response cached for the given request.

\*/

func hasCacheForRequest(request: NSURLRequest) -> Bool {

if let hash = hashForRequest(request) {

for requestHash in requestCaches {

if hash == requestHash {

return true

}

}

}

return false

}

/\*\*

Returns the path where we should store our plist.

:returns: The file path URL.

\*/

func diskPathForPropertyList() -> NSURL? {

var url: NSURL?

let filename = "diskCacheInfo.plist"

if let baseURL = diskPath() {

url = NSURL(string: filename, relativeToURL: baseURL)

}

return url

}

/\*\*

Returns the path where we should store a cache

with the specified filename.

:params: name The filename of the cached request.

:returns: The file path URL.

\*/

private func diskPathForRequestCacheNamed(name: String) -> NSURL? {

var url: NSURL?

if let baseURL = diskPath() {

url = NSURL(string: name, relativeToURL: baseURL)

}

return url

}

/\*\*

Returns the path where a response should be stored

for a given NSURLRequest.

:params: request The request.

:returns: The file path URL.

\*/

func diskPathForRequest(request: NSURLRequest) -> NSURL? {

var url: NSURL?

if let

hash = hashForRequest(request),

baseURL = diskPath()

{

url = NSURL(string: hash, relativeToURL: baseURL)

}

return url

}

}

}}