import UIKit

import CoreData

import BUAdSDK

import Alamofire

@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate,WXApiDelegate,BUSplashAdDelegate {

var window: UIWindow?

let manager = NetworkReachabilityManager(host: "http://www.baidu.com")

func application(\_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

// Override point for customization after application launch.

// 如果当前在登陆状态，使用refresh\_token刷新本地存的access\_token

if WMUserModel.share.isLogin() == true{

WMNetTools.share.refreshToken {

}

}else{

}

// 默认目标步数10000步

if (UserDefaults.standard.value(forKey: "TargetStepCount") != nil) {

}else{

UserDefaults.standard.set(10000, forKey: "TargetStepCount")

UserDefaults.standard.synchronize()

}

ZKProgressHUD.setAutoDismissDelay(1)// 全局HUD设置自动隐藏延时秒数，默认值：2

ZKProgressHUD.setAnimationStyle(.system)

ZKProgressHUD.setMaskStyle(.hide)

IQKeyboardManager.shared.enable = true

IQKeyboardManager.shared.enableAutoToolbar = false// 不显示工具条

IQKeyboardManager.shared.shouldResignOnTouchOutside = true// 点空白处收回键盘

self.window?.rootViewController = WMMainTabBarController()

self.window?.makeKeyAndVisible()

// 向微信注册

WXApi.registerApp(WXAPPID, universalLink: WXUniversalLink)

// 穿山甲广告

self.setBUAd()

// 网络监听

self.monitorNetWorkStatus()

return true

}

// MARK: - 穿山甲、开屏广告

func setBUAd() -> Void {

BUAdSDKManager.setAppID("5099782")

BUAdSDKManager.setIsPaidApp(false)

// BUAdSDKManager.setLoglevel(.debug) // 打印日志

// 开屏广告

let splashView = BUSplashAdView.init(slotID: "887369141", frame: UIScreen.main.bounds)

splashView.delegate = self

let keyWindow = self.window

splashView.loadAdData()

keyWindow?.rootViewController?.view.addSubview(splashView)

splashView.rootViewController = keyWindow?.rootViewController

}

// MARK: - 监听网络状态

func monitorNetWorkStatus() -> Void {

manager?.startListening(onUpdatePerforming: { (status) in

print("Network Status Changed: \(status)")

switch status{

case .unknown:

// 未知网络

break

case .notReachable:

// 无法连接网络

break

case .reachable(.cellular):

// 蜂窝移动网络

NotificationCenter.default.post(name: NSNotification.Name("NetworkStatusChangedNotiName"), object: nil)

case .reachable(.ethernetOrWiFi):

// wifi

NotificationCenter.default.post(name: NSNotification.Name("NetworkStatusChangedNotiName"), object: nil)

}

})

}

// MARK: - BUSplashAdDelegate

func splashAdDidClose(\_ splashAd: BUSplashAdView) {

splashAd.removeFromSuperview()

}

func splashAdDidLoad(\_ splashAd: BUSplashAdView) {

}

func splashAd(\_ splashAd: BUSplashAdView, didFailWithError error: Error?) {

splashAd.removeFromSuperview()

}

/\*

// MARK: UISceneSession Lifecycle

func application(\_ application: UIApplication, configurationForConnecting connectingSceneSession: UISceneSession, options: UIScene.ConnectionOptions) -> UISceneConfiguration {

// Called when a new scene session is being created.

// Use this method to select a configuration to create the new scene with.

return UISceneConfiguration(name: "Default Configuration", sessionRole: connectingSceneSession.role)

}

func application(\_ application: UIApplication, didDiscardSceneSessions sceneSessions: Set<UISceneSession>) {

// Called when the user discards a scene session.

// If any sessions were discarded while the application was not running, this will be called shortly after application:didFinishLaunchingWithOptions.

// Use this method to release any resources that were specific to the discarded scenes, as they will not return.

}

\*/

// MARK: - Core Data stack

lazy var persistentContainer: NSPersistentContainer = {

/\*

The persistent container for the application. This implementation

creates and returns a container, having loaded the store for the

application to it. This property is optional since there are legitimate

error conditions that could cause the creation of the store to fail.

\*/

let container = NSPersistentContainer(name: "WalkMoney")

container.loadPersistentStores(completionHandler: { (storeDescription, error) in

if let error = error as NSError? {

// Replace this implementation with code to handle the error appropriately.

// fatalError() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.

/\*

Typical reasons for an error here include:

\* The parent directory does not exist, cannot be created, or disallows writing.

\* The persistent store is not accessible, due to permissions or data protection when the device is locked.

\* The device is out of space.

\* The store could not be migrated to the current model version.

Check the error message to determine what the actual problem was.

\*/

fatalError("Unresolved error \(error), \(error.userInfo)")

}

})

return container

}()

// MARK: - Core Data Saving support

func saveContext () {

let context = persistentContainer.viewContext

if context.hasChanges {

do {

try context.save()

} catch {

// Replace this implementation with code to handle the error appropriately.

// fatalError() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.

let nserror = error as NSError

fatalError("Unresolved error \(nserror), \(nserror.userInfo)")

}

}

}

// WXApi

func application(\_ application: UIApplication, handleOpen url: URL) -> Bool {

return WXApi.handleOpen(url, delegate: self)

}

func application(\_ application: UIApplication, open url: URL, sourceApplication: String?, annotation: Any) -> Bool {

return WXApi.handleOpen(url, delegate: self)

}

func application(\_ application: UIApplication, continue userActivity: NSUserActivity, restorationHandler: @escaping ([UIUserActivityRestoring]?) -> Void) -> Bool {

return WXApi.handleOpenUniversalLink(userActivity, delegate: self)

}

// MARK: - WXApiDelegate

func onReq(\_ req: BaseReq) {

print("wx--> \(req)")

}

func onResp(\_ resp: BaseResp) {

if resp.errCode == 0 {

let \_resp = resp as! SendAuthResp

if let code = \_resp.code {

//通知传递返回的code

NotificationCenter.default.post(name: NSNotification.Name("WXAuthNotifyKey"), object: nil, userInfo: ["code":code])

}

}else{

print(resp.errStr)

}

}

class func sendAuthRequest(){

if WXApi.isWXAppInstalled() {

let req = SendAuthReq.init()

req.scope = "snsapi\_userinfo"

req.state = Date().milliStamp

WXApi.send(req) { (iserror) in

}

}else{

ZKProgressHUD.showMessage("请先安装微信!")

}

}

}

import UIKit

class WMMainTabBarController: UITabBarController {

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

let walkVC = WMWalkViewController()

let nav1 = WMBaseNavigationController.init(rootViewController: walkVC)

nav1.tabBarItem = UITabBarItem.init(title: "走路", image: UIImage.init(named: "walk\_no"), selectedImage: UIImage.init(named: "walk\_yes"))

let makeMoneyVC = WMMakeMoneyViewController()

let nav2 = WMBaseNavigationController.init(rootViewController: makeMoneyVC)

nav2.tabBarItem = UITabBarItem.init(title: "赚钱", image: UIImage.init(named: "makemoney\_no"), selectedImage: UIImage.init(named: "makemoney\_yes"))

let meVC = WMMeViewController()

let nav3 = WMBaseNavigationController.init(rootViewController: meVC)

nav3.tabBarItem = UITabBarItem.init(title: "我的", image: UIImage.init(named: "me\_no"), selectedImage: UIImage.init(named: "me\_yes"))

self.viewControllers = [nav1, nav2, nav3]

if #available(iOS 13.0, \*) {

let appearance = tabBar.standardAppearance.copy()

appearance.backgroundImage = UIImage.Create(size: CGSize(width: 1, height: 1), color: .white)

appearance.shadowImage = UIImage.Create(size: CGSize(width: 1, height: 1), color: UIColor.hexString("#000000", a: 0))

tabBar.standardAppearance = appearance

} else {

// Fallback on earlier versions

tabBar.backgroundImage = UIImage.Create(size: CGSize(width: 1, height: 1), color: .white)

tabBar.shadowImage = UIImage.Create(size: CGSize(width: 1, height: 1), color: UIColor.hexString("#000000", a: 0))

}

let topBorder = CALayer()

let borderHeight : CGFloat = 0.33

topBorder.borderWidth = borderHeight

topBorder.borderColor = UIColor.hexString("#000000", a: 0.1).cgColor

topBorder.frame = CGRect(x: 0, y: 0, width: SCREEN\_WIDTH, height: borderHeight)

self.tabBar.layer.addSublayer(topBorder)

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

import UIKit

// 引入三方头文件

@\_exported import ZKProgressHUD

@\_exported import Alamofire

@\_exported import SwiftyJSON

@\_exported import IQKeyboardManagerSwift

@\_exported import CommonCrypto

@\_exported import KakaJSON

@\_exported import MJRefresh

@\_exported import Kingfisher

@\_exported import AdSupport

@\_exported import SwiftDate

@\_exported import SnapKit

class WMTools: UIView {

/\*

// Only override draw() if you perform custom drawing.

// An empty implementation adversely affects performance during animation.

override func draw(\_ rect: CGRect) {

// Drawing code

}

\*/

}

// 屏幕 宽度、高度

let SCREEN\_WIDTH = UIScreen.main.bounds.size.width

let SCREEN\_HEIGHT = UIScreen.main.bounds.size.height

// 判读是否为iPhone X及以上

let is\_iPhoneX = UIApplication.shared.statusBarFrame.size.height > 20 ? true : false

// 状态栏高度

let statusBarHeight = is\_iPhoneX ? 44 : 20

// 状态栏+导航栏高度

let statusBarAndNavigationBarHeight = is\_iPhoneX ? 88 : 64

// tabbar高度

let tabbarHeight = is\_iPhoneX ? 49+34 : 49

// 底部安全区域

let safeAreaBottom = is\_iPhoneX ? 34 : 0

let WXAPPID = "wxc41c8fee5f1d3979"

let WXUniversalLink = "https://www.52yqs.com/"

let AppStoreAPPID = "382201985"// 假数据

let WebUrl\_Privacy = "https://www.jd.com" // 隐私协议 // 假数据

let WebUrl\_Terms = "https://www.taobao.com" // 服务条款 // 假数据

let WebUrl\_TurnTable = "https://www.gmgmapps.com/walk/turntable.html" // 大转盘

let WebUrl\_Prize = "https://www.gmgmapps.com/walk/prize.html" // 摇奖机

let WebUrl\_InviteFriend = "https://www.gmgmapps.com/walk/invite\_friends.html" // 邀请好友

// 常用扩展------

// color

extension UIColor {

static func hexString(\_ hex:String, a: Float) -> UIColor {

var cString:String = hex.trimmingCharacters(in: .whitespacesAndNewlines).uppercased()

if (cString.hasPrefix("#")) {

cString.remove(at: cString.startIndex)

}

if ((cString.count) != 6) {

return UIColor.gray

}

var rgbValue:UInt32 = 0

Scanner(string: cString).scanHexInt32(&rgbValue)

return UIColor(

red: CGFloat((rgbValue & 0xFF0000) >> 16) / 255.0,

green: CGFloat((rgbValue & 0x00FF00) >> 8) / 255.0,

blue: CGFloat(rgbValue & 0x0000FF) / 255.0,

alpha: CGFloat(a)

)

}

// UIColor.hexString("#2D73F1")

static func fromRGB(\_ rgbValue: UInt, a: Float) -> UIColor {

return UIColor(

red: CGFloat((rgbValue & 0xFF0000) >> 16) / 255.0,

green: CGFloat((rgbValue & 0x00FF00) >> 8) / 255.0,

blue: CGFloat(rgbValue & 0x0000FF) / 255.0,

alpha: CGFloat(a)

)

}

// UIColor.fromRGB(0x209624)

}

// Frame

extension UIView {

var width: CGFloat {

get { return self.frame.size.width }

set {

var frame = self.frame

frame.size.width = newValue

self.frame = frame

}

}

var height: CGFloat {

get { return self.frame.size.height }

set {

var frame = self.frame

frame.size.height = newValue

self.frame = frame

}

}

var size: CGSize {

get { return self.frame.size }

set {

var frame = self.frame

frame.size = newValue

self.frame = frame

}

}

var origin: CGPoint {

get { return self.frame.origin }

set {

var frame = self.frame

frame.origin = newValue

self.frame = frame

}

}

var x: CGFloat {

get { return self.frame.origin.x }

set {

var frame = self.frame

frame.origin.x = newValue

self.frame = frame

}

}

var y: CGFloat {

get { return self.frame.origin.y }

set {

var frame = self.frame

frame.origin.y = newValue

self.frame = frame

}

}

var centerX: CGFloat {

get { return self.center.x }

set {

self.center = CGPoint(x: newValue, y: self.center.y)

}

}

var centerY: CGFloat {

get { return self.center.y }

set {

self.center = CGPoint(x: self.center.x, y: newValue)

}

}

var top : CGFloat {

get { return self.frame.origin.y }

set {

var frame = self.frame

frame.origin.y = newValue

self.frame = frame

}

}

var bottom : CGFloat {

get { return frame.origin.y + frame.size.height }

set {

var frame = self.frame

frame.origin.y = newValue - self.frame.size.height

self.frame = frame

}

}

var right : CGFloat {

get { return self.frame.origin.x + self.frame.size.width }

set {

var frame = self.frame

frame.origin.x = newValue - self.frame.size.width

self.frame = frame

}

}

var left : CGFloat {

get { return self.frame.origin.x }

set {

var frame = self.frame

frame.origin.x = newValue

self.frame = frame

}

}

}

// 设置圆角、边框(方便在Storyboard中设置Layer层，圆角、边框、边框宽度和颜色等)

extension UIView {

@IBInspectable var cornerRadius: CGFloat {

get {

return layer.cornerRadius

}

set {

layer.cornerRadius = newValue

layer.masksToBounds = newValue > 0

}

}

@IBInspectable var borderColor: UIColor {

get {

return UIColor.init(cgColor: layer.borderColor ??

UIColor.white.cgColor)

}

set {

layer.borderColor = newValue.cgColor

}

}

@IBInspectable var borderWidth: CGFloat {

get {

return layer.borderWidth

}

set {

layer.borderWidth = newValue

}

}

}

// 绘制渐变

class CGView: UIView {

var colors = [CGFloat]()

var startPoint = CGPoint()

var endPoint = CGPoint()

override init(frame: CGRect) {

super.init(frame: frame)

// 设置背景色为透明，否则是黑色背景

self.backgroundColor = UIColor.clear

}

required init?(coder: NSCoder) {

fatalError("init(coder:) has not been implemented")

}

override func draw(\_ rece: CGRect){

super.draw(rece)

// 获取绘图上下文

guard let context = UIGraphicsGetCurrentContext() else {

return

}

// 使用rgb颜色空间

let colorSpace = CGColorSpaceCreateDeviceRGB()

// 颜色数组(这里使用两组颜色作为渐变)

let compoents:[CGFloat] = colors

// 每组颜色所在位置(范围0~1)

let locations:[CGFloat] = [0, 1]

// 生成渐变色(count参数表示渐变个数)

let gradient = CGGradient(colorSpace: colorSpace, colorComponents: compoents, locations: locations, count: locations.count)!

// 渐变开始位置

let start = startPoint

// 渐变结束为止

let end = endPoint

// 绘制渐变

context.drawLinearGradient(gradient, start: start, end: end, options: .drawsBeforeStartLocation)

}

}

extension Date {

/// 获取当前 毫秒级 时间戳 - 13位

var milliStamp : String {

let timeInterval: TimeInterval = self.timeIntervalSince1970

let millisecond = CLongLong(round(timeInterval\*1000))

return "\(millisecond)"

}

}

extension String{

//随机一个10到18位的字符串

static let random\_str\_characters = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"

static func randomStr() -> String{

let len = arc4random() % 8 + 10

var ranStr = ""

for \_ in 0..<len {

let index = Int(arc4random\_uniform(UInt32(random\_str\_characters.count)))

ranStr.append(random\_str\_characters[random\_str\_characters.index(random\_str\_characters.startIndex, offsetBy: index)])

}

return ranStr

}

//sha1编码

func sha1() -> String {

let data = self.data(using: String.Encoding.utf8)!

var digest = [UInt8](repeating: 0, count: Int(CC\_SHA1\_DIGEST\_LENGTH))

CC\_SHA1([UInt8](data), CC\_LONG(data.count), &digest)

// let resultBytes = Data(bytes: digest, count: Int(CC\_SHA1\_DIGEST\_LENGTH))

// let resultStr = resultBytes.base64EncodedString()

// return resultStr

//无需base64输出,装换为16进制字符串输出

let output = NSMutableString(capacity: Int(CC\_SHA1\_DIGEST\_LENGTH))

for byte in digest {

output.appendFormat("%02x", byte)

}

return output as String

}

}

// MD5

public func md5(strs:String) ->String!{

let str = strs.cString(using: String.Encoding.utf8)

let strLen = CUnsignedInt(strs.lengthOfBytes(using: String.Encoding.utf8))

let digestLen = Int(CC\_MD5\_DIGEST\_LENGTH)

let result = UnsafeMutablePointer<CUnsignedChar>.allocate(capacity: digestLen)

CC\_MD5(str!, strLen, result)

let hash = NSMutableString()

for i in 0 ..< digestLen {

hash.appendFormat("%02x", result[i])

}

return String(format: hash as String)

}

/// JSONString转换为字典

/// - Parameter jsonString: <#jsonString description#>

/// - Returns: <#return value description#>

public func getDictionaryFromJSONString(jsonString:String) ->NSDictionary{

let jsonData:Data = jsonString.data(using: .utf8)!

let dict = try? JSONSerialization.jsonObject(with: jsonData, options: .mutableContainers)

if dict != nil {

return dict as! NSDictionary

}

return NSDictionary()

}

/\*

字典转换为JSONString

- parameter dictionary: 字典参数

- returns: JSONString

\*/

func getJSONStringFromDictionary(dictionary:NSDictionary) -> String {

if (!JSONSerialization.isValidJSONObject(dictionary)) {

print("无法解析出JSONString")

return ""

}

let data : NSData! = try? JSONSerialization.data(withJSONObject: dictionary, options: []) as NSData?

let JSONString = NSString(data:data as Data,encoding: String.Encoding.utf8.rawValue)

return JSONString! as String

}

//MARK: - UIDevice延展

public extension UIDevice {

static let modelName: String = {

var systemInfo = utsname()

uname(&systemInfo)

let machineMirror = Mirror(reflecting: systemInfo.machine)

let identifier = machineMirror.children.reduce("") { identifier, element in

guard let value = element.value as? Int8, value != 0 else { return identifier }

return identifier + String(UnicodeScalar(UInt8(value)))

}

func mapToDevice(identifier: String) -> String { // swiftlint:disable:this cyclomatic\_complexity

#if os(iOS)

switch identifier {

case "iPod5,1": return "iPod touch (5th generation)"

case "iPod7,1": return "iPod touch (6th generation)"

case "iPod9,1": return "iPod touch (7th generation)"

case "iPhone3,1", "iPhone3,2", "iPhone3,3": return "iPhone 4"

case "iPhone4,1": return "iPhone 4s"

case "iPhone5,1", "iPhone5,2": return "iPhone 5"

case "iPhone5,3", "iPhone5,4": return "iPhone 5c"

case "iPhone6,1", "iPhone6,2": return "iPhone 5s"

case "iPhone7,2": return "iPhone 6"

case "iPhone7,1": return "iPhone 6 Plus"

case "iPhone8,1": return "iPhone 6s"

case "iPhone8,2": return "iPhone 6s Plus"

case "iPhone9,1", "iPhone9,3": return "iPhone 7"

case "iPhone9,2", "iPhone9,4": return "iPhone 7 Plus"

case "iPhone8,4": return "iPhone SE"

case "iPhone10,1", "iPhone10,4": return "iPhone 8"

case "iPhone10,2", "iPhone10,5": return "iPhone 8 Plus"

case "iPhone10,3", "iPhone10,6": return "iPhone X"

case "iPhone11,2": return "iPhone XS"

case "iPhone11,4", "iPhone11,6": return "iPhone XS Max"

case "iPhone11,8": return "iPhone XR"

case "iPhone12,1": return "iPhone 11"

case "iPhone12,3": return "iPhone 11 Pro"

case "iPhone12,5": return "iPhone 11 Pro Max"

case "iPad2,1", "iPad2,2", "iPad2,3", "iPad2,4":return "iPad 2"

case "iPad3,1", "iPad3,2", "iPad3,3": return "iPad (3rd generation)"

case "iPad3,4", "iPad3,5", "iPad3,6": return "iPad (4th generation)"

case "iPad6,11", "iPad6,12": return "iPad (5th generation)"

case "iPad7,5", "iPad7,6": return "iPad (6th generation)"

case "iPad7,11", "iPad7,12": return "iPad (7th generation)"

case "iPad4,1", "iPad4,2", "iPad4,3": return "iPad Air"

case "iPad5,3", "iPad5,4": return "iPad Air 2"

case "iPad11,4", "iPad11,5": return "iPad Air (3rd generation)"

case "iPad2,5", "iPad2,6", "iPad2,7": return "iPad mini"

case "iPad4,4", "iPad4,5", "iPad4,6": return "iPad mini 2"

case "iPad4,7", "iPad4,8", "iPad4,9": return "iPad mini 3"

case "iPad5,1", "iPad5,2": return "iPad mini 4"

case "iPad11,1", "iPad11,2": return "iPad mini (5th generation)"

case "iPad6,3", "iPad6,4": return "iPad Pro (9.7-inch)"

case "iPad6,7", "iPad6,8": return "iPad Pro (12.9-inch)"

case "iPad7,1", "iPad7,2": return "iPad Pro (12.9-inch) (2nd generation)"

case "iPad7,3", "iPad7,4": return "iPad Pro (10.5-inch)"

case "iPad8,1", "iPad8,2", "iPad8,3", "iPad8,4":return "iPad Pro (11-inch)"

case "iPad8,5", "iPad8,6", "iPad8,7", "iPad8,8":return "iPad Pro (12.9-inch) (3rd generation)"

case "AppleTV5,3": return "Apple TV"

case "AppleTV6,2": return "Apple TV 4K"

case "AudioAccessory1,1": return "HomePod"

case "i386", "x86\_64": return "Simulator \(mapToDevice(identifier: ProcessInfo().environment["SIMULATOR\_MODEL\_IDENTIFIER"] ?? "iOS"))"

default: return identifier

}

#elseif os(tvOS)

switch identifier {

case "AppleTV5,3": return "Apple TV 4"

case "AppleTV6,2": return "Apple TV 4K"

case "i386", "x86\_64": return "Simulator \(mapToDevice(identifier: ProcessInfo().environment["SIMULATOR\_MODEL\_IDENTIFIER"] ?? "tvOS"))"

default: return identifier

}

#endif

}

return mapToDevice(identifier: identifier)

}()

}

/// 总的能量 = (体重÷2000 )\*总步数, 默认体重60kg

/// - Parameter stepsCount: 步数

/// - Returns: 能量 kcal

func KcalWithStepsCount(stepsCount:Int) -> Double {

return Double(60/2000.0) \* Double(stepsCount)

}

/// 步行时间 = 距离(km) / 速度(默认4.6km/h)

/// - Parameter distance: 单位：千米

/// - Returns: 单位：时

func stepsTimeWithDistance(distance:Double) -> String {

let hour = Double(distance/4.6) // 单位: 时

let minute = Int(hour \* 60) // 单位: 分

// 转换成 x:x' （时分）

if (minute/60) > 0 {

return "\(minute/60):\(minute%60)’"

}else{

return "\(minute%60)’"

}

}

// 每天签到奖励的金币数

func coinWithSignInDay(nextDay:Int) -> Int {

switch nextDay {

case 1:

return 78

case 2:

return 88

case 3:

return 98

case 4:

return 108

case 5:

return 118

case 6:

return 128

case 7:

return 188

default:

return 0

}

}

/// GCD实现定时器

///

/// - Parameters:

/// - timeInterval: 间隔时间

/// - handler: 事件

/// - needRepeat: 是否重复

func dispatchTimer(timeInterval: Double, handler: @escaping (DispatchSourceTimer?) -> Void, needRepeat: Bool) {

let timer = DispatchSource.makeTimerSource(flags: [], queue: DispatchQueue.main)

timer.schedule(deadline: .now(), repeating: timeInterval)

timer.setEventHandler {

DispatchQueue.main.async {

if needRepeat {

handler(timer)

} else {

timer.cancel()

handler(nil)

}

}

}

timer.resume()

}

import UIKit

typealias WMResponseSuccess = (\_ response: String) -> Void

typealias WMResponseFail = (\_ error: AnyObject) -> Void

let MainUrl = "https://www.52yqs.com/magicad/"

//let MainUrl = "http://172.168.1.27:51169/"

let SECRET = "DSoierSAweiodf124saSdg56"

class WMNetTools: NSObject {

static let share = WMNetTools()

private override init() {

super.init()

}

/// post网络请求

/// - Parameters:

/// - url: url地址

/// - params: 参数

/// - isToken: 是否需要token

/// - success: 请求成功

/// - failure: 请求失败

/// - Returns: void

public func postDataRequest(url: String,

params: [String: Any],

isToken: Bool,

success: @escaping WMResponseSuccess,

failure: @escaping WMResponseFail) -> Void {

let urlStr = MainUrl + url

var request = URLRequest(url: URL.init(string: urlStr)!)

request.httpMethod = HTTPMethod.post.rawValue

request.setValue("application/json", forHTTPHeaderField: "Content-Type")

let jsonstr = getJSONStringFromDictionary(dictionary: params as NSDictionary)

let newjsonstr = jsonstr.replacingOccurrences(of: "\\", with: "")

let jsonData = newjsonstr.data(using: .utf8, allowLossyConversion: false)

request.httpBody = jsonData

let ts = Date().milliStamp

let nonce = String.randomStr()

let str = (md5(strs: newjsonstr) + ts + nonce + SECRET).sha1()

request.headers.add(name: "X-TimeStamp", value: ts)

request.headers.add(name: "X-Nonce", value: nonce)

request.headers.add(name: "X-Signature", value: str)

if isToken == true {

let userModel = WMUserModel.share.readUserModel()

let tk = "Bearer " + userModel.access\_token

request.headers.add(name: "Authorization", value: tk)

}else{

}

ZKProgressHUD.show()

AF.request(request).responseString{ (response) in

ZKProgressHUD.dismiss()

switch response.result{

case .success(let json):

print("newnew111--> \(url)--> \(json)")

let dic = getDictionaryFromJSONString(jsonString: json)

let status = dic["status"] as? String

if status == "ok" {

success(json)

}else{

let errmsg = dic["errmsg"] as? String

// 判断是否token超时失效

if isToken && (errmsg?.contains("expired"))!{

WMNetTools.share.refreshToken {// 刷新token

success("expired")

}

}

// else if (errmsg == "Illegal Access"){// 非法访问

// // 退出登陆，重新登录

// // 退出登陆，清除本地userModel数据

// WMUserModel.share.clearUserModel()

//

// }

else{

ZKProgressHUD.showMessage(errmsg)

}

}

break

case .failure(let error):

failure(error as AnyObject)

ZKProgressHUD.showMessage("请检查网络状态")

break

}

}

}

// 刷新token

public func refreshToken(block: @escaping() -> ()){

let model = WMUserModel.share.readUserModel()

let dic = ["refresh\_token":model.refresh\_token, "uuid":model.uuid]

WMNetTools.share.postDataRequest(url: "token/refresh/", params: dic, isToken: false, success: { (response) in

let dic = getDictionaryFromJSONString(jsonString: response)

model.access\_token = dic["access\_token"] as! String

WMUserModel.share.updateUserModel(model: model)

// 首次访问激活

WMNetTools.share.configGet {

block()

}

}) { (error) in

print("\(error)")

}

}

// 首次访问激活

public func configGet(completion: @escaping() -> ()){

let model = WMUserModel.share.readUserModel()

let dic = ["uuid":model.uuid]

WMNetTools.share.postDataRequest(url: "config/get/", params: dic, isToken: true, success: { (response) in

completion()

//通知传递 token刷新成功, configGet配置接口请求成功

NotificationCenter.default.post(name: NSNotification.Name("IsRefreshTokenAndConfigGetNotiName"), object: nil)

}) { (error) in

print("\(error)")

}

}

// 获取服务器当前时间

public func getCurrentDate(block: @escaping(Int) -> ()){

AF.request("https://www.52yqs.com/magicad/util/now/").responseString { (response) in

print("server time--> \(response)")

switch response.result{

case .success(let json):

let dic = getDictionaryFromJSONString(jsonString: json)

let ts = dic["ts"] as? Int

block(ts!)

break

case .failure( \_):

block(0)

ZKProgressHUD.showMessage("请检查网络状态")

break

}

}

}

// 查明某事件总数

public func loadEventsActivateCount(name:String,success: @escaping(String) -> ()) -> Void {

let model = WMUserModel.share.readUserModel()

let dic = ["name":name, "uuid":model.uuid]

WMNetTools.share.postDataRequest(url: "events/activate/count/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

success("expired")

}else{

let dic = getDictionaryFromJSONString(jsonString: response)

success(dic["result"] as! String)

}

}) { (error) in

print("\(error)")

}

}

// 查明某事件连续天数

public func loadEventsContinueCount(name:String,success: @escaping(String) -> ()) -> Void {

let model = WMUserModel.share.readUserModel()

let dic = ["name":name, "uuid":model.uuid]

WMNetTools.share.postDataRequest(url: "events/continue/count/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

success("expired")

}else{

let dic = getDictionaryFromJSONString(jsonString: response)

success(String(dic["result"] as! Int))

}

}) { (error) in

print("\(error)")

}

}

}

import UIKit

@\_exported import CoreMotion

class WMStepTools: NSObject {

var pedometer = CMPedometer()

static let share = WMStepTools()

private override init() {

super.init()

}

public func getStepCount(block: @escaping (Dictionary<String,Any>)->()){

pedometer = CMPedometer.init()

WMNetTools.share.getCurrentDate { (timeStamp) in

if timeStamp == 0{

block([:] as Dictionary<String, Any>)

}else{

//--------------------------获取步数截止时间----------------------------------

//指定一个Region

let rome = Region(calendar: Calendars.gregorian, zone: Zones.asiaShanghai,

locale: Locales.chinese)

// 从时间间隔创建, 现在时间(获取步数截止时间)

let nowDate = DateInRegion(seconds: Double(timeStamp/1000), region: rome)

//--------------------------获取步数开始时间----------------------------------

let date = DateInRegion(components: {

$0.year = nowDate.dateComponents.year

$0.month = nowDate.dateComponents.month

$0.day = nowDate.dateComponents.day

$0.hour = 0

$0.minute = 0

$0.second = 0

})

// 凌晨的格林威治时间（8小时前）

let startDate = date!.date - 8.hours

// 凌晨的格林威治时间戳

let timeInterval : TimeInterval = startDate.timeIntervalSince1970

let startTimeStamp = Int(timeInterval) \* 1000

// 本地有存储上次步数获取时间 && 在同一天 && 本地有存储上次上传步数，需要减去上次的步数

var todayLastSteps = 0 // 今日上次上传的总步数，用于add增加步数减法

if case Optional<Any>.none = KeychainManager.keyChainReadData(identifier: "UpLoadStepTime"){

}else{

let saveTimeStamp = KeychainManager.keyChainReadData(identifier: "UpLoadStepTime") as! Int

print("0点--> \(startTimeStamp) 保存--> \(saveTimeStamp) 现在--> \(timeStamp)")

//是当天 当天0点的时间戳 < 存储的时间戳 < 现在的时间戳

if (saveTimeStamp > startTimeStamp) && (saveTimeStamp < timeStamp){

if case Optional<Any>.none = KeychainManager.keyChainReadData(identifier: "TodayLastStepsCount"){

}else{

todayLastSteps = KeychainManager.keyChainReadData(identifier: "TodayLastStepsCount") as! Int

}

}else{

}

}

// 判断记步功能

if CMPedometer.isStepCountingAvailable(){

print("获取步数时间范围--> 开始--> \(startDate) 结束--> \(nowDate.date)")

self.pedometer.queryPedometerData(from: startDate, to: nowDate.date) { (pedometerData, error) in

if (error != nil) {

print("stepError--> \(error ?? "" as! Error)")

DispatchQueue.main.async{

ZKProgressHUD.showMessage("请打开健康权限")

}

}else{

print("ooooo--> 步数--> \(pedometerData?.numberOfSteps)")

print("ooooo--> 距离--> \(pedometerData?.distance)")

var addstep\_count = pedometerData?.numberOfSteps.intValue

if todayLastSteps > 0 {// 如果今天有缓存的数据,则减去

addstep\_count = (pedometerData?.numberOfSteps.intValue)! - todayLastSteps

}

let dic = ["step\_count":pedometerData?.numberOfSteps as Any, "distance":pedometerData?.distance as Any, "addstep\_count":addstep\_count as Any] as Dictionary

block(dic as Dictionary<String, Any>)

}

}

}else{

DispatchQueue.main.async{

ZKProgressHUD.showMessage("设备不支持步数计数功能")

}

}

}

}

}

}

import UIKit

import EventKit

class WMEventTools: NSObject {

lazy var eventStore = EKEventStore()

static let share = WMEventTools()

private override init() {

super.init()

}

/// 系统日历添加提醒事件

/// - Parameter eventTitle: event 名字

/// - Parameter eventNotes: event 信息

/// - Parameter eventRepeatTime: event 重复提醒时间 Date 类型 精确到分钟

/// - Parameter eventEndTime: event 结束时间 Date 类型

/// - Parameter eventRepeatDate: event 重复日 例: 0, 1, 2 0 为周日 以此类推

/// - Parameter eventAlarmTime: event 提醒时间 单位: min 为空 则不提醒

/// - Parameter eventIdentifier: event Block 系统分配事件 identifier

/// - Parameter eventAddFailure: event Block 添加失败 一般会是 权限问题

public func addACalendarTime(\_ eventTitle: String,

eventNotes: String,

eventRepeatTime: Date,

eventEndTime: Date,

eventRepeatDate: String?,

eventAlarmTime: Int?,

eventIdentifier: @escaping ((\_ identifier: String) -> Void),

eventAddFailure: @escaping (() -> Void)) {

let event = EKEvent(eventStore: self.eventStore)

event.title = eventTitle

event.notes = eventNotes

event.startDate = eventRepeatTime

event.endDate = eventEndTime

event.isAllDay = false

if let time = eventAlarmTime {

// 添加闹钟结合（0）若为正数则是开始后多少秒。

event.addAlarm(EKAlarm(relativeOffset: TimeInterval(0 \* time)))

}

event.calendar = self.eventStore.defaultCalendarForNewEvents

var weekArr: [EKRecurrenceDayOfWeek] = []

if let wt = eventRepeatDate {

let wArr = wt.components(separatedBy: ",")

for index in 0..<wArr.count {

let weekInt: Int = Int(wArr[index]) ?? -1

if weekInt > -1 {

let daysOfWeek = EKRecurrenceDayOfWeek(EKWeekday.init(rawValue: weekInt + 1)!)

weekArr.append(daysOfWeek)

}

}

}

let rule = EKRecurrenceRule(recurrenceWith: .weekly,

interval: 1,

daysOfTheWeek: weekArr,

daysOfTheMonth: nil,

monthsOfTheYear: nil,

weeksOfTheYear: nil,

daysOfTheYear: nil,

setPositions: nil,

end: nil)// (按次数提醒:EKRecurrenceEnd(count: 100) ,(按结束时间提醒:EKRecurrenceEnd(end: eventEndTime))

event.addRecurrenceRule(rule)

do {

try self.eventStore.save(event, span: .thisEvent)

eventIdentifier(event.eventIdentifier)

print("添加成功")

}catch {

print("添加失败")

}

}

/// 查询日历中的事件

/// - Parameter completion: 两个日期中间日历事件数组

/// - Returns: void

public func checkEvent(completion: @escaping(Array<Any>) -> ()) -> Void {

// 起始日期

let startDate = Date()

// 结束日期

let endDate = Date() + 1.days

let predicate = self.eventStore.predicateForEvents(withStart: startDate, end: endDate, calendars: [self.eventStore.defaultCalendarForNewEvents!])

let eventArray = self.eventStore.events(matching: predicate)

completion(eventArray)

}

}

import UIKit

class KeychainManager: NSObject {

// TODO: 创建查询条件

class func createQuaryMutableDictionary(identifier:String)->NSMutableDictionary{

// 创建一个条件字典

let keychainQuaryMutableDictionary = NSMutableDictionary.init(capacity: 0)

// 设置条件存储的类型

keychainQuaryMutableDictionary.setValue(kSecClassGenericPassword, forKey: kSecClass as String)

// 设置存储数据的标记

keychainQuaryMutableDictionary.setValue(identifier, forKey: kSecAttrService as String)

keychainQuaryMutableDictionary.setValue(identifier, forKey: kSecAttrAccount as String)

// 设置数据访问属性

keychainQuaryMutableDictionary.setValue(kSecAttrAccessibleAfterFirstUnlock, forKey: kSecAttrAccessible as String)

// 返回创建条件字典

return keychainQuaryMutableDictionary

}

// TODO: 存储数据

class func keyChainSaveData(data:Any ,withIdentifier identifier:String)->Bool {

// 获取存储数据的条件

let keyChainSaveMutableDictionary = self.createQuaryMutableDictionary(identifier: identifier)

// 删除旧的存储数据

SecItemDelete(keyChainSaveMutableDictionary)

// 设置数据

keyChainSaveMutableDictionary.setValue(NSKeyedArchiver.archivedData(withRootObject: data), forKey: kSecValueData as String)

// 进行存储数据

let saveState = SecItemAdd(keyChainSaveMutableDictionary, nil)

if saveState == noErr {

return true

}

return false

}

// TODO: 更新数据

class func keyChainUpdata(data:Any ,withIdentifier identifier:String)->Bool {

// 获取更新的条件

let keyChainUpdataMutableDictionary = self.createQuaryMutableDictionary(identifier: identifier)

// 创建数据存储字典

let updataMutableDictionary = NSMutableDictionary.init(capacity: 0)

// 设置数据

updataMutableDictionary.setValue(NSKeyedArchiver.archivedData(withRootObject: data), forKey: kSecValueData as String)

// 更新数据

let updataStatus = SecItemUpdate(keyChainUpdataMutableDictionary, updataMutableDictionary)

if updataStatus == noErr {

return true

}

return false

}

// TODO: 获取数据

class func keyChainReadData(identifier:String)-> Any {

var idObject:Any?

// 获取查询条件

let keyChainReadmutableDictionary = self.createQuaryMutableDictionary(identifier: identifier)

// 提供查询数据的两个必要参数

keyChainReadmutableDictionary.setValue(kCFBooleanTrue, forKey: kSecReturnData as String)

keyChainReadmutableDictionary.setValue(kSecMatchLimitOne, forKey: kSecMatchLimit as String)

// 创建获取数据的引用

var queryResult: AnyObject?

// 通过查询是否存储在数据

let readStatus = withUnsafeMutablePointer(to: &queryResult) { SecItemCopyMatching(keyChainReadmutableDictionary, UnsafeMutablePointer($0))}

if readStatus == errSecSuccess {

if let data = queryResult as! NSData? {

idObject = NSKeyedUnarchiver.unarchiveObject(with: data as Data) as Any

}

}

return idObject as Any

}

// TODO: 删除数据

class func keyChianDelete(identifier:String)->Void{

// 获取删除的条件

let keyChainDeleteMutableDictionary = self.createQuaryMutableDictionary(identifier: identifier)

// 删除数据

SecItemDelete(keyChainDeleteMutableDictionary)

}

}

import UIKit

import WebKit

class LBWebViewController: UIViewController, WKNavigationDelegate, WKScriptMessageHandler {

var url : String = ""

var isShowNavigation = true

var backButtonTitle = ""

var webView : WKWebView?

lazy var myProgressView : UIProgressView = {

let ProgressView = UIProgressView.init(frame: CGRect(x: 0, y: 0, width: SCREEN\_WIDTH, height: 1))

ProgressView.tintColor = UIColor.blue

ProgressView.trackTintColor = UIColor.white

return ProgressView

}()

lazy var backButton: UIButton = {

let button = UIButton.init(frame: CGRect(x: 0, y: statusBarHeight, width: 100, height: 44))

button.backgroundColor = UIColor.clear

button.setImage(UIImage.init(named: "back\_white"), for: .normal)

button.setTitle(self.backButtonTitle, for: .normal)

button.titleLabel?.font = UIFont.systemFont(ofSize: 16)

button.imageEdgeInsets = UIEdgeInsets.init(top: 0.0, left: 0.0, bottom: 0.0, right: 30.0)

button.titleEdgeInsets = UIEdgeInsets.init(top: 0.0, left: 0.0, bottom: 0.0, right: 30.0)

button.addTarget(self, action: #selector(backButtonAction), for: .touchUpInside)

return button

}()

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

self.navigationController?.navigationBar.isTranslucent = false

if self.isShowNavigation {

// Do noting

}else{

self.navigationController?.setNavigationBarHidden(true, animated: animated)

}

}

override func viewWillDisappear(\_ animated: Bool) {

super.viewWillDisappear(animated)

self.navigationController?.navigationBar.isTranslucent = true

if self.isShowNavigation {

// Do noting

}else{

self.navigationController?.setNavigationBarHidden(false, animated: animated)

}

}

override func viewDidLoad() {

super.viewDidLoad()

let preferences = WKPreferences()

let configuration = WKWebViewConfiguration()

configuration.preferences = preferences

configuration.userContentController = WKUserContentController()

// 注册js函数

configuration.userContentController.add(self, name: "jstoios")

self.webView = WKWebView.init(frame: CGRect(x: 0, y: 0, width: 0, height: 0), configuration: configuration)

self.webView?.navigationDelegate = self

if #available(iOS 11.0, \*) {

self.webView?.scrollView.contentInsetAdjustmentBehavior = .never

} else {

// Fallback on earlier versions

}

self.view.addSubview(self.webView!)

self.webView!.snp.makeConstraints { (make) in

make.left.equalToSuperview().offset(0)

make.right.equalToSuperview().offset(0)

make.top.equalToSuperview().offset(0)

make.bottom.equalToSuperview().offset(0)

}

let request = URLRequest.init(url: URL.init(string: self.url)!)

self.webView!.load(request)

self.webView!.scrollView.bounces = false

// 进度条

self.view.addSubview(self.myProgressView)

self.webView!.addObserver(self, forKeyPath: "estimatedProgress", options: NSKeyValueObservingOptions.new, context: nil)

// BackButton

if self.isShowNavigation {

// Do noting

}else{

self.webView!.addSubview(self.backButton)

}

////

let btn = UIButton.init(frame: CGRect(x: 200, y: 200, width: 200, height: 50))

btn.setTitle("iosto——js", for: .normal)

btn.backgroundColor = UIColor.blue

btn.addTarget(self, action: #selector(buttonAction), for: .touchUpInside)

self.webView?.addSubview(btn)

}

deinit {

self.webView!.removeObserver(self, forKeyPath: "estimatedProgress")

}

override func observeValue(forKeyPath keyPath: String?, of object: Any?, change: [NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?) {

if keyPath == "estimatedProgress" {

self.myProgressView.progress = Float(webView!.estimatedProgress)

if (self.myProgressView.progress >= 1.0) {

let deadline = DispatchTime.now() + 0.3

DispatchQueue.global().asyncAfter(deadline: deadline) {

DispatchQueue.main.async {

self.myProgressView.progress = 0;

}

}

}

}else{

}

}

@objc func backButtonAction() -> Void {

self.navigationController?.popViewController(animated: false)

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

func userContentController(\_ userContentController: WKUserContentController, didReceive message: WKScriptMessage) {

print("-----------------" + message.name)

if message.name == "jstoios" {

print("fffmessageBody: --> \(message.body)")

}else{

// Do noting

}

}

@objc func buttonAction() -> Void {

webView?.evaluateJavaScript("iostojs('test345')", completionHandler: { (response, error) in

print("fffmessage: \(response) error--> \(error.debugDescription)")

})

}

}

import KakaJSON

class WMUserModel: NSObject,Convertible,NSCoding {

var status : String = ""

var nickname : String = ""

var url : String = ""

var uuid : String = ""

var access\_token : String = ""

var access\_token\_expires\_at : String = ""

var refresh\_token : String = ""

var refresh\_token\_expires\_at : String = ""

var wx\_bind : String = ""

var my\_refer\_code : String = ""

// 单利初始化

static let share : WMUserModel = {

let instance = WMUserModel()

return instance

}()

required internal override init() {

super.init()

}

// 更新本地的用户数据

public func updateUserModel(model: WMUserModel) -> Void {

let userDefault = UserDefaults.standard

//实例对象转换成Data

let modelData = NSKeyedArchiver.archivedData(withRootObject: model)

//存储Data对象

userDefault.set(modelData, forKey: "userModel")

//将存储的偏好存储同步到ROM中

userDefault.synchronize()

}

// 清除本地的用户数据

public func clearUserModel() -> Void {

let userDefault = UserDefaults.standard

userDefault.removeObject(forKey: "userModel")

//将存储的偏好存储同步到ROM中

userDefault.synchronize()

}

// 查询本地的用户数据

public func readUserModel() -> WMUserModel {

let userDefault = UserDefaults.standard

let modelData = userDefault.data(forKey: "userModel")

var model = WMUserModel.share

if modelData != nil {

model = NSKeyedUnarchiver.unarchiveObject(with: modelData!) as! WMUserModel

}

return model

}

public func isLogin() -> Bool{

let userDefault = UserDefaults.standard

let modelData = userDefault.data(forKey: "userModel")

var model = WMUserModel.share

if modelData != nil {

model = NSKeyedUnarchiver.unarchiveObject(with: modelData!) as! WMUserModel

}

if (model.access\_token as NSString).length > 0 {

return true

}else{

return false

}

}

//从object解析回来

required init(coder decoder: NSCoder) {

self.status = decoder.decodeObject(forKey: "status") as? String ?? ""

self.nickname = decoder.decodeObject(forKey: "nickname") as? String ?? ""

self.url = decoder.decodeObject(forKey: "url") as? String ?? ""

self.uuid = decoder.decodeObject(forKey: "uuid") as? String ?? ""

self.access\_token = decoder.decodeObject(forKey: "access\_token") as? String ?? ""

self.access\_token\_expires\_at = decoder.decodeObject(forKey: "access\_token\_expires\_at") as? String ?? ""

self.refresh\_token = decoder.decodeObject(forKey: "refresh\_token") as? String ?? ""

self.refresh\_token\_expires\_at = decoder.decodeObject(forKey: "refresh\_token\_expires\_at") as? String ?? ""

self.wx\_bind = decoder.decodeObject(forKey: "wx\_bind") as? String ?? ""

self.my\_refer\_code = decoder.decodeObject(forKey: "my\_refer\_code") as? String ?? ""

}

//编码成object

func encode(with coder: NSCoder) {

coder.encode(status, forKey:"status")

coder.encode(nickname, forKey:"nickname")

coder.encode(url, forKey:"url")

coder.encode(uuid, forKey:"uuid")

coder.encode(access\_token, forKey:"access\_token")

coder.encode(access\_token\_expires\_at, forKey:"access\_token\_expires\_at")

coder.encode(refresh\_token, forKey:"refresh\_token")

coder.encode(refresh\_token\_expires\_at, forKey:"refresh\_token\_expires\_at")

coder.encode(wx\_bind, forKey:"wx\_bind")

coder.encode(my\_refer\_code, forKey:"my\_refer\_code")

}

}

import UIKit

import AuthenticationServices

// 登陆成功回调

typealias siginSuccessBlock = () -> Void

class WMLoginViewController: WMBaseViewController,UITextViewDelegate {

@IBOutlet weak var textView: UITextView!

var siginBlock : siginSuccessBlock?

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

self.view.backgroundColor = UIColor.white

// 同意协议

let str1 = "注册即代表同意“走路有钱”"

let str2 = "服务条款"

let str3 = "和"

let str4 = "隐私政策"

let str:NSString = (str1 + str2 + str3 + str4) as NSString

let range1 = str.range(of: str2)

let range2 = str.range(of: str4)

let parStyle = NSMutableParagraphStyle()

parStyle.lineSpacing = 0.5

let att = NSMutableAttributedString(string: str as String, attributes: [NSAttributedString.Key.font:UIFont.systemFont(ofSize: 12),NSAttributedString.Key.foregroundColor:UIColor.black,NSAttributedString.Key.paragraphStyle:parStyle])

let valueStr1 = "fwtk://\(str2)".addingPercentEncoding(withAllowedCharacters: CharacterSet.urlFragmentAllowed)

let valueStr2 = "yszc://\(str4)".addingPercentEncoding(withAllowedCharacters: CharacterSet.urlFragmentAllowed)

att.addAttributes([NSAttributedString.Key.link : valueStr1 as Any,], range: range1)

att.addAttributes([NSAttributedString.Key.link : valueStr2 as Any], range: range2)

textView.linkTextAttributes = [NSAttributedString.Key.foregroundColor:UIColor.hexString("#0096FF", a: 1.0)]

textView.attributedText = att

//监听接收消息，获取传过来的code，触发微信登陆方法

NotificationCenter.default.addObserver(self, selector: #selector(WXLogin(noti:)), name: NSNotification.Name(rawValue: "WXAuthNotifyKey"), object: nil)

}

// MARK: - UITextViewDelegate

func textView(\_ textView: UITextView, shouldInteractWith URL: URL, in characterRange: NSRange, interaction: UITextItemInteraction) -> Bool {

if URL.scheme == "fwtk" {

let vc = LBWebViewController.init()

vc.url = WebUrl\_Terms

vc.title = "服务条款"

let nav = UINavigationController.init(rootViewController: vc)

self.present(nav, animated: true) {

}

return false

}else if URL.scheme == "yszc"{

let vc = LBWebViewController.init()

vc.url = WebUrl\_Privacy

vc.title = "隐私政策"

let nav = UINavigationController.init(rootViewController: vc)

self.present(nav, animated: true) {

}

return false

}

return true

}

// MARK: - Action

@IBAction func wechatButtonAction(\_ sender: UIButton) {

print("微信登陆")

AppDelegate.sendAuthRequest()

}

@IBAction func signInWithAppleButtonAction(\_ sender: UIButton) {

print("苹果登陆")

if #available(iOS 13.0, \*) {

let requests = [ASAuthorizationAppleIDProvider().createRequest()]

let authorizationController = ASAuthorizationController(authorizationRequests: requests)

authorizationController.delegate = self

authorizationController.presentationContextProvider = self

authorizationController.performRequests()

} else {

// Fallback on earlier versions

}

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

deinit {

NotificationCenter.default.removeObserver(self)

}

@objc private func WXLogin(noti:Notification){

let dic = noti.userInfo

let code = dic!["code"]

let idfa = ASIdentifierManager.shared().advertisingIdentifier.uuidString

let iosVersion = UIDevice.current.systemVersion

let modelName = UIDevice.modelName //获取设备的型号 例如：iPhone

let dic2 = ["code":code,"idfa":idfa,"iosversion":iosVersion,"device":modelName]

// 微信登陆

WMNetTools.share.postDataRequest(url: "wx/login/", params: dic2 as [String : Any], isToken: false, success: { (response) in

// model储存用户信息

let dic = getDictionaryFromJSONString(jsonString: response)

let userModel = dic.kj.model(WMUserModel.self)

WMUserModel.share.updateUserModel(model: userModel!)

self.dismiss(animated: true) {

}

// 请求事件注册

WMNetTools.share.configGet {

// 登陆成功回调

if self.siginBlock != nil {

self.siginBlock?()

}

}

}) { (error) in

print("error:\(error)")

}

}

}

//扩展WMLoginViewController

extension WMLoginViewController: ASAuthorizationControllerDelegate {

//授权成功回调

@available(iOS 13.0, \*)

func authorizationController(controller: ASAuthorizationController, didCompleteWithAuthorization authorization: ASAuthorization) {

switch authorization.credential {

case let appleIDCredential as ASAuthorizationAppleIDCredential:

let identityToken = appleIDCredential.identityToken

print("授权成功：identityToken = \(identityToken!)")

let idfa = ASIdentifierManager.shared().advertisingIdentifier.uuidString

let iosVersion = UIDevice.current.systemVersion

let dic = ["token":String(data: identityToken!, encoding: .utf8),"idfa":idfa,"iosversion":iosVersion,"email":appleIDCredential.email]

// 苹果登录

WMNetTools.share.postDataRequest(url: "apple/login/", params: dic as [String : Any], isToken: false, success: { (response) in

// model储存用户信息

let dic = getDictionaryFromJSONString(jsonString: response)

let userModel = dic.kj.model(WMUserModel.self)

WMUserModel.share.updateUserModel(model: userModel!)

self.dismiss(animated: true) {

}

// 请求事件注册

WMNetTools.share.configGet {

// 登陆成功回调

if self.siginBlock != nil {

self.siginBlock?()

}

}

}) { (error) in

}

default:

break

}

}

//授权失败回调

@available(iOS 13.0, \*)

func authorizationController(controller: ASAuthorizationController, didCompleteWithError error: Error) {

//处理错误

print("授权失败：error = \(error)")

}

}

extension WMLoginViewController: ASAuthorizationControllerPresentationContextProviding {

//提供弹窗根视图

@available(iOS 13.0, \*)

func presentationAnchor(for controller: ASAuthorizationController) -> ASPresentationAnchor {

return self.view.window!

}

}

import Foundation

fileprivate struct WMEndDate {

var day: Int?

var hour: Int?

var minute: Int?

var second: Int?

}

final class WMCountDown {

/// 要计算的时间

fileprivate var currentDate: WMEndDate?

/// 定时器

fileprivate var timer: DispatchSourceTimer?

/// 开始时间

fileprivate var startDateString: String? = nil

/// 结束时间

fileprivate var endDateString: String? = nil

/// 回调 计算好的 天、时、分、秒

var countDown: ((\_ day: String, \_ hour: String, \_ minute: String, \_ second: String) -> Void)?

/// 开始启动定时器，需传入结束时间

func start(with startDate: String?, end endDate: String) { startDateString = startDate ; endDateString = endDate ; createTimer() }

/// 启用定时器

func resume() { if endDateString == nil { return } ; createTimer() }

/// 暂停定时器

func suspend() { stop() }

/// 停止定时器

func stop() { guard let timer = timer else { return } ; timer.cancel() }

/// 创建定时器

fileprivate func createTimer() {

guard let endDateString = endDateString else { print("没有设置开始时间") ; return }

guard let countDown = countDown else { print("没有设置回调函数") ; return }

// 将 endDateString 转化成时间戳

let dateFormatter = DateFormatter()

dateFormatter.dateFormat = "yyyy-MM-dd HH:mm:ss"

let startDate: Date

if let startDateString = startDateString { startDate = dateFormatter.date(from: startDateString)! }

else { startDate = Date() }

let endDate = dateFormatter.date(from: endDateString)

var timeCount = endDate!.timeIntervalSince(startDate)

// 当时间小于当前时间

if timeCount <= 0 { countDown("00", "00", "00", "00") ; return }

// 设置计算时间的格式

let calendar = Calendar.current

let unit: Set<Calendar.Component> = [.day, .hour, .minute, .second]

// let component = calendar.dateComponents(unit, from: Date.init(), to: endDate!)

let component = calendar.dateComponents(unit, from: startDate, to: endDate!)

currentDate = WMEndDate(day: component.day, hour: component.hour, minute: component.minute, second: component.second)

// 设置定时器

timer = DispatchSource.makeTimerSource(flags: [], queue: DispatchQueue.global())

timer!.schedule(wallDeadline: DispatchWallTime.now(), repeating: DispatchTimeInterval.seconds(1))

// 定时器的回调事件

timer!.setEventHandler { [unowned self] in self.calculateCurrentDate() ; timeCount -= 1 }

timer!.resume()

}

/// 定时器计时事件

fileprivate func calculateCurrentDate() {

// 解包

guard let countDown = countDown else { return }

// 在创建定时器之前已经判断过timeCount <= 0, 保证了timeCount > 0, 此时这里不用判断

// 计算显示时间

currentDate!.second! -= 1

if currentDate!.second! == -1 { currentDate!.second! = 59 ; currentDate!.minute! -= 1

if currentDate!.minute! == -1 { currentDate!.minute! = 59 ; currentDate!.hour! -= 1

if currentDate!.hour! == -1 { currentDate!.hour! = 23 ; currentDate!.day! -= 1 }

}

}

if currentDate!.second! == 0

&& currentDate!.minute! == 0

&& currentDate!.hour! == 0

&& currentDate!.day! == 0 {

timer!.cancel()

DispatchQueue.main.async(execute: { countDown("0", "00", "00", "00") })

return

}

// 主线程更新返回数据

DispatchQueue.main.async(execute: { [unowned self ] in

countDown(String(format: "%02d", self.currentDate!.day!),

String(format: "%02d", self.currentDate!.hour!),

String(format: "%02d", self.currentDate!.minute!),

String(format: "%02d", self.currentDate!.second!))

})

}

deinit { timer?.cancel() }

}

import UIKit

class LBRoundProgressView: UIView {

var startAngle : CGFloat = -225 // 开始的角度

var endAngle : CGFloat = 45 // 结束的角度

var progess: CGFloat = 0.0 // 环形进度

var label: UILabel? // 中心文本显示

var lineWidth: CGFloat = 0.0 // 环形的宽

private var foreLayer: CAShapeLayer? // 进度条的layer层（可做私有属性）

override init(frame: CGRect) {

super.init(frame: frame)

}

// 覆写父类构造器后这个方法是必须实现的

required init?(coder aDecoder: NSCoder) {

fatalError("init(coder:) has not been implemented")

}

// 遍历构造器传入frame，以及进度条宽度

convenience init(frame: CGRect, lineWidth: CGFloat) {

self.init(frame: frame)

self.lineWidth = lineWidth

seup(rect: frame) // 绘制自定义视图的函数

}

func seup(rect: CGRect) -> Void {

// 背景圆环（灰色背景）

let shapeLayer: CAShapeLayer = CAShapeLayer.init()

// 设置frame

shapeLayer.frame = CGRect.init(x: 0, y: 0, width: rect.size.width, height: rect.size.height)

shapeLayer.lineWidth = self.lineWidth

shapeLayer.fillColor = UIColor.clear.cgColor

shapeLayer.strokeColor = UIColor.hexString("#81BFFF", a: 0.2).cgColor

shapeLayer.lineCap = .round // 设置画笔

let center: CGPoint = CGPoint.init(x: rect.size.width/2, y: rect.size.height/2)

// 画出曲线（贝塞尔曲线）

// let bezierPath: UIBezierPath = UIBezierPath.init(arcCenter: center, radius: (rect.size.width - self.lineWidth)/2, startAngle: CGFloat(-0.5 \* Double.pi), endAngle: CGFloat(1.5 \* Double.pi), clockwise: true)// 整圆

let bezierPath: UIBezierPath = UIBezierPath.init(arcCenter: center, radius: (rect.size.width - self.lineWidth)/2, startAngle: CGFloat(Double.pi)\*startAngle/180.0, endAngle: CGFloat(Double.pi)\*endAngle/180.0, clockwise: true)

shapeLayer.path = bezierPath.cgPath // 将曲线添加到layer层

self.layer.addSublayer(shapeLayer) // 添加蒙版

// 渐变色 加蒙版 显示蒙版区域

let gradientLayer: CAGradientLayer = CAGradientLayer.init()

gradientLayer.frame = self.bounds

gradientLayer.colors = NSArray.init(array: [UIColor.hexString("#198AFF", a: 1.0).cgColor, UIColor.hexString("#81BFF0", a: 1.0).cgColor]) as? [Any]

gradientLayer.startPoint = CGPoint.init(x: 0, y: 1)

gradientLayer.endPoint = CGPoint.init(x: 1, y: 1)

self.layer.addSublayer(gradientLayer) // 将渐变色添加带layer的子视图

self.foreLayer = CAShapeLayer.init()

self.foreLayer?.frame = self.bounds

self.foreLayer?.fillColor = UIColor.clear.cgColor

self.foreLayer?.lineWidth = self.lineWidth

self.foreLayer?.strokeColor = UIColor.red.cgColor

self.foreLayer?.strokeEnd = 0

/\* The cap style used when stroking the path. Options are `butt', `round'

\* and `square'. Defaults to `butt'. \*/

self.foreLayer?.lineCap = .round // 设置画笔

self.foreLayer?.path = bezierPath.cgPath

shapeLayer.addSublayer(self.foreLayer!)

// 修改渐变layer层的遮罩, 关键代码

gradientLayer.mask = self.foreLayer

// 中间imageView

let centerImageView = UIImageView.init(image: UIImage.init(named: "progress"))

centerImageView.frame = CGRect(x: (self.width-162)/2, y: (self.height-139)/2-10, width: 162, height: 139)

self.addSubview(centerImageView)

let personImageView = UIImageView.init(image: UIImage.init(named: "person"))

personImageView.frame = CGRect(x: (self.width-24)/2, y: (self.height-48)/2-30, width: 24, height: 48)

self.addSubview(personImageView)

self.label = UILabel.init(frame:CGRect(x: (self.width-100)/2, y: personImageView.bottom+8, width: 100, height: 48))

self.label?.text = "0"

self.addSubview(self.label!)

self.label?.font = UIFont(name:"DINCondensed-Bold", size:48)

self.label?.textColor = UIColor.hexString("#198AFF", a: 1.0)

self.label?.textAlignment = NSTextAlignment.center

}

func setProgress(stepCount: Int) -> Void {

let step = UserDefaults.standard.value(forKey: "TargetStepCount") as! Double

progess = CGFloat(Double(stepCount)/step) // 设置当前属性的值

self.label?.text = "\(stepCount)"

self.foreLayer?.strokeEnd = progess // 视图改变的关键代码

}

/\*

// Only override draw() if you perform custom drawing.

// An empty implementation adversely affects performance during animation.

override func draw(\_ rect: CGRect) {

// Drawing code

}

\*/

}

import UIKit

class WMWalkRankTableViewCell: UITableViewCell {

@IBOutlet weak var avaImageView: UIImageView!

@IBOutlet weak var nameLabel: UILabel!

@IBOutlet weak var stepCountLabel: UILabel!

@IBOutlet weak var stepChangeLabel: UILabel!

@IBOutlet weak var rankLabel: UILabel!

@IBOutlet weak var rankLabelCenterY: NSLayoutConstraint!

var dic : Dictionary<String, Any> = [:]{

didSet{

self.nameLabel.text = dic["nickname"] as? String

self.stepCountLabel.text = "\(dic["steps"]!)步"

if dic["url"] is NSNull {

if dic["me"] as? Bool == true {

let userModel = WMUserModel.share.readUserModel()

self.avaImageView.kf.setImage(with: URL.init(string: userModel.url), placeholder: UIImage.init(named: "ava\_default"))

}else{

self.avaImageView.image = UIImage.init(named: "ava\_default")

}

}else{

self.avaImageView.kf.setImage(with: URL.init(string: (dic["url"] as? String)!), placeholder: UIImage.init(named: "ava\_default"))

}

self.rankLabel.text = "\(dic["rank"]!)"

}

}

override func awakeFromNib() {

super.awakeFromNib()

// Initialization code

}

override func setSelected(\_ selected: Bool, animated: Bool) {

super.setSelected(selected, animated: animated)

// Configure the view for the selected state

}

}

import UIKit

class WMChartsView: UIView {

@IBOutlet weak var HLineSpacing0: NSLayoutConstraint!

@IBOutlet weak var HLineSpacing1: NSLayoutConstraint!

@IBOutlet weak var HLineSpacing2: NSLayoutConstraint!

@IBOutlet weak var HLineSpacing3: NSLayoutConstraint!

@IBOutlet weak var HLineSpacing4: NSLayoutConstraint!

@IBOutlet weak var HLineView3: UIView!

@IBOutlet weak var HLineView4: UIView!

@IBOutlet weak var HLineView5: UIView!

@IBOutlet weak var HLabel3: UILabel!

@IBOutlet weak var HLabel4: UILabel!

@IBOutlet weak var HLabel5: UILabel!

@IBOutlet weak var VView0Height: NSLayoutConstraint!

@IBOutlet weak var VView1Height: NSLayoutConstraint!

@IBOutlet weak var VView2Height: NSLayoutConstraint!

@IBOutlet weak var VView3Height: NSLayoutConstraint!

@IBOutlet weak var VView4Height: NSLayoutConstraint!

@IBOutlet weak var VView5Height: NSLayoutConstraint!

@IBOutlet weak var VView6Height: NSLayoutConstraint!

@IBOutlet weak var VView7Height: NSLayoutConstraint!

@IBOutlet weak var VView8Height: NSLayoutConstraint!

@IBOutlet weak var VView9Height: NSLayoutConstraint!

@IBOutlet weak var dateLabel0: UILabel!

@IBOutlet weak var dateLabel1: UILabel!

@IBOutlet weak var dateLabel2: UILabel!

@IBOutlet weak var dateLabel3: UILabel!

@IBOutlet weak var dateLabel4: UILabel!

@IBOutlet weak var dateLabel5: UILabel!

@IBOutlet weak var dateLabel6: UILabel!

@IBOutlet weak var dateLabel7: UILabel!

@IBOutlet weak var dateLabel8: UILabel!

@IBOutlet weak var dateLabel9: UILabel!

@IBOutlet weak var dottedLineBottomHeight: NSLayoutConstraint!

var HLineCount : Int?{// 找出最大值(2...5几根横线?),默认间距2

didSet{

self.HLineSpacing0.constant = CGFloat(80/HLineCount!)

self.HLineSpacing1.constant = CGFloat(80/HLineCount!)

self.HLineSpacing2.constant = CGFloat(80/HLineCount!)

self.HLineSpacing3.constant = CGFloat(80/HLineCount!)

self.HLineSpacing4.constant = CGFloat(80/HLineCount!)

switch HLineCount {

case 2:

self.HLineView3.isHidden = true

self.HLineView4.isHidden = true

self.HLineView5.isHidden = true

self.HLabel3.isHidden = true

self.HLabel4.isHidden = true

self.HLabel5.isHidden = true

case 3:

self.HLineView4.isHidden = true

self.HLineView5.isHidden = true

self.HLabel4.isHidden = true

self.HLabel5.isHidden = true

case 4:

self.HLineView5.isHidden = true

self.HLabel5.isHidden = true

default:

break

}

// 根据横线数量，设置各个VView高度

let VViewHeightArray = [VView0Height,VView1Height,VView2Height,VView3Height,VView4Height,VView5Height,VView6Height,VView7Height,VView8Height,VView9Height]

for i in 0..<array!.count {

let dic = array![i] as! Dictionary<String, Any>

// 步数

let step = dic.values.first as! Int

// 高度/80 == step/(HLineCount\*10000)

VViewHeightArray[i]?.constant = CGFloat(Double(step)/(Double(HLineCount!\*10000))\*80)

}

// 根据横线数量，设置目标线高度

let targetStep = UserDefaults.standard.value(forKey: "TargetStepCount") as! Double

// 目标线高度 高度/80 = targetStep/(HLineCount\*10000),底部距离24

self.dottedLineBottomHeight.constant = CGFloat(targetStep/(Double(HLineCount!\*10000))\*80 + 24)

}

}

var array : Array<Any>?{

didSet{

// 日期

let dateLabelArray = [dateLabel0,dateLabel1,dateLabel2,dateLabel3,dateLabel4,dateLabel5,dateLabel6,dateLabel7,dateLabel8,dateLabel9]

var steps = Array<Int>.init()

for i in 0..<array!.count {

let dic = array![i] as! Dictionary<String, Any>

// 日期

let date = dic.keys.first?.toDate("yyyy-MM-dd")

dateLabelArray[i]?.text = i==0 ? "\(date?.month ?? 0)月\(date?.day ?? 0)" : "\(date?.day ?? 0)"

// 步数

let step = dic.values.first as! Int

steps.append(step)

}

// 找出最大步数值和目标线判断，取最大值

let targetStep = UserDefaults.standard.value(forKey: "TargetStepCount") as! Int

let max = steps.max()! > targetStep ? steps.max()! : targetStep

// 找出最大值，设置横线数量

if max > 40000 {

HLineCount = 5

}else if max > 30000{

HLineCount = 4

}else if max > 20000{

HLineCount = 3

}else{

HLineCount = 2

}

}

}

// Only override draw() if you perform custom drawing.

// An empty implementation adversely affects performance during animation.

override func draw(\_ rect: CGRect) {

// Drawing code

}

}

import UIKit

@IBDesignable

class WMDottedLine: UIView {

// line

@IBInspectable var lineWidth: CGFloat = 1

@IBInspectable var lineColor: UIColor = UIColor.black

// padding

@IBInspectable var paddingLeft: CGFloat = 0

@IBInspectable var paddingRight: CGFloat = 0

@IBInspectable var paddingTop: CGFloat = 0

@IBInspectable var paddingBottom: CGFloat = 0

// direct

@IBInspectable var isHorizontal: Bool = true

// dash

@IBInspectable var isDash: Bool = false

@IBInspectable var dashPointWidth: CGFloat = 3.0

@IBInspectable var dashSpace: CGFloat = 1.0

// Only override draw() if you perform custom drawing.

// An empty implementation adversely affects performance during animation.

override func draw(\_ rect: CGRect) {

// Drawing code

super.draw(rect)

// Drawing code

// 获取上下文对象

let context = UIGraphicsGetCurrentContext()

var bx: CGFloat = 0, by: CGFloat = 0, ex: CGFloat = 0, ey: CGFloat = 0;

if isHorizontal {

bx = paddingLeft

by = CGFloat(Int(rect.size.height)/2)

ex = rect.size.width - paddingRight

ey = by

} else {

bx = CGFloat(Int(rect.size.width)/2)

by = paddingTop

ex = bx

ey = rect.size.height - paddingBottom

}

// 画中间虚线

let path = CGMutablePath()

let begin = CGPoint(x: bx, y: by),

end = CGPoint(x: ex, y: ey)

path.move(to: begin)

path.addLine(to: end)

// 2、 添加路径到图形上下文

context!.addPath(path)

// 3、 设置状态

context!.setLineWidth(lineWidth / UIScreen.main.scale)

context!.setStrokeColor(lineColor.cgColor)

if isDash {

context!.setLineDash(phase: 0, lengths: [dashPointWidth, dashSpace])

}

// 4、 绘制图像到指定图形上下文

context!.drawPath(using: .fillStroke)

}

}

import UIKit

class WMWalkViewController: WMBaseViewController,UITableViewDelegate,UITableViewDataSource {

@IBOutlet weak var tableView: UITableView!

@IBOutlet var todayDataCell: UITableViewCell!

@IBOutlet var headerView: UIView!

@IBOutlet var sortNoDataCell: UITableViewCell!

var roundView = LBRoundProgressView()

// 气泡

@IBOutlet weak var pointView1: UIView!

@IBOutlet weak var pointView2: UIView!

@IBOutlet weak var pointView3: UIView!

@IBOutlet weak var pointView4: UIView!

@IBOutlet weak var pointCoinLabel1: UILabel!

@IBOutlet weak var pointNameLabel1: UILabel!

@IBOutlet weak var pointCoinLabel2: UILabel!

@IBOutlet weak var pointNameLabel2: UILabel!

@IBOutlet weak var pointCoinLabel3: UILabel!

@IBOutlet weak var pointNameLabel3: UILabel!

@IBOutlet weak var pointCoinLabel4: UILabel!

@IBOutlet weak var pointNameLabel4: UILabel!

// 累计天数

@IBOutlet weak var allSignInDayLabel: UILabel!// 签到

@IBOutlet weak var morningDayLabel: UILabel!// 早起

@IBOutlet weak var eightWaterDayLabel: UILabel!// 八杯水

@IBOutlet weak var tenThousandStepsDayLabel: UILabel!// 一万步

var pointArray : Array<UIView> = []

var pointCoinLabelArray : Array<UILabel> = []

var pointNameLabelArray : Array<UILabel> = []

var pointDataArray : Array<Dictionary<String, Any>> = []// 展示气泡数据

var sortArray : Array<Dictionary<String, Any>> = [] // 排序数组

@IBOutlet weak var distanceLabel: UILabel!// 步行里程

@IBOutlet weak var kcalLabel: UILabel!// 消耗能量

@IBOutlet weak var stepTimeLabel: UILabel!// 步行时间

var isRefreshTokenAndConfigGetStatus : Bool = false

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

self.navigationController?.setNavigationBarHidden(true, animated: animated)

// 获取设备步数并且登陆状态下上传至服务器

self.setStepCount()

// 登陆状态查询气泡等事件信息

if WMUserModel.share.isLogin() == true && self.isRefreshTokenAndConfigGetStatus == true{

self.loadEventsQuery()

self.loadUsersSortData()

//喝水 早起 签到 一万步，累计总数

self.loadEventsAllCount(name: "喝水")

self.loadEventsAllCount(name: "早起")

self.loadEventsAllCount(name: "签到")

self.loadEventsAllCount(name: "每天一万步")

}else{

// 未登录排名去掉

self.sortArray.removeAll()

self.tableView.reloadData()

// 气泡去掉

self.pointDataArray.removeAll()

}

}

override func viewWillDisappear(\_ animated: Bool) {

super.viewWillDisappear(animated)

self.navigationController?.setNavigationBarHidden(false, animated: animated)

}

override func viewDidLoad() {

super.viewDidLoad()

self.title = "走路"

//监听通知 token刷新成功, configGet配置接口请求成功

NotificationCenter.default.addObserver(self, selector: #selector(LoadData(noti:)), name: NSNotification.Name(rawValue: "IsRefreshTokenAndConfigGetNotiName"), object: nil)

// Do any additional setup after loading the view.

headerView.frame = CGRect(x: 0, y: 0, width: SCREEN\_WIDTH, height: 373)

self.tableView.tableHeaderView = headerView

// 让tableview顶部从状态栏上开始算Y轴坐标

if #available(iOS 11.0, \*) {

self.tableView.contentInsetAdjustmentBehavior = UIScrollView.ContentInsetAdjustmentBehavior.never// 让ScrollView 不自动计算的方法

} else {

// Fallback on earlier versions

automaticallyAdjustsScrollViewInsets = false

}

// 绘制渐变

let cgView = CGView(frame: CGRect(x: 0, y: 0, width: headerView.width, height: headerView.height))

let compoents:[CGFloat] = [129/255, 191/255, 240/255, 0.1,

255/255, 250/255, 240/255, 0]

cgView.colors = compoents

cgView.startPoint = CGPoint(x: self.headerView.bounds.minX, y: self.headerView.bounds.minY)

cgView.endPoint = CGPoint(x: self.headerView.bounds.minX, y: self.headerView.bounds.maxY)

headerView.insertSubview(cgView, at: 0)

// 注册cell

self.tableView.register(UINib.init(nibName: "WMWalkRankTableViewCell", bundle: nil), forCellReuseIdentifier: "WMWalkRankTableViewCell")

// 气泡动画和点击事件

pointArray = [pointView1!, pointView2!, pointView3!, pointView4!]

for i in 0..<pointArray.count {

// 添加动画

self.upDownAnimate(view: pointArray[i])

// 添加手势点击事件

let pointTap = UITapGestureRecognizer.init(target: self, action: #selector(self.PointViewAction(\_:)))

pointArray[i].addGestureRecognizer(pointTap)

}

pointCoinLabelArray = [pointCoinLabel1!,pointCoinLabel2!,pointCoinLabel3!,pointCoinLabel4!]

pointNameLabelArray = [pointNameLabel1!,pointNameLabel2!,pointNameLabel3!,pointNameLabel4!]

// 中间进度条

roundView = LBRoundProgressView.init(frame: CGRect.init(x: (SCREEN\_WIDTH - 200)/2, y: 85, width: 200, height: 200), lineWidth: 16)

roundView.backgroundColor = UIColor.clear

headerView.addSubview(roundView)

// 添加手势点击事件

let roundViewTap = UITapGestureRecognizer.init(target: self, action: #selector(self.stepRecordButtonAction(\_:)))

roundView.addGestureRecognizer(roundViewTap)

}

@objc private func LoadData(noti:Notification){

self.isRefreshTokenAndConfigGetStatus = true

self.loadEventsQuery()

self.loadUsersSortData()

//喝水 早起 签到 一万步，累计总数

self.loadEventsAllCount(name: "喝水")

self.loadEventsAllCount(name: "早起")

self.loadEventsAllCount(name: "签到")

self.loadEventsAllCount(name: "每天一万步")

// 请求新人红包接口

self.loadNewUserRedEnvelopeData()

}

@objc func NetworkStatusChanged(noti:Notification) -> Void {

// 获取设备步数并且登陆状态下上传至服务器

self.setStepCount()

// 如果当前在登陆状态，使用refresh\_token刷新本地存的access\_token

if WMUserModel.share.isLogin() == true{

WMNetTools.share.refreshToken {

}

}else{

// Do noting

}

}

deinit {

NotificationCenter.default.removeObserver(self)

}

// MARK: - UITableViewDelegate,UITableViewDataSource

func numberOfSections(in tableView: UITableView) -> Int{

return 2

}

func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int{

if section == 0 {

return 1

}else{

// 登陆状态查询

if WMUserModel.share.isLogin() == true{

return self.sortArray.count

}else{

return 1

}

}

}

func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat{

if indexPath.section == 0 {

return 200

}else{

// 登陆状态查询

if WMUserModel.share.isLogin() == true{

return 48

}else{

return SCREEN\_WIDTH

}

}

}

func tableView(\_ tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat{

return 33.9

}

func tableView(\_ tableView: UITableView, heightForFooterInSection section: Int) -> CGFloat{

return 0.01

}

func tableView(\_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {

if section == 0 {

return "今日数据"

}else{

return "今日排名"

}

}

func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell{

if indexPath.section == 0 {

return self.todayDataCell

}else{

// 登陆状态查询

if WMUserModel.share.isLogin() == true{

let cell = tableView.dequeueReusableCell(withIdentifier: "WMWalkRankTableViewCell", for: indexPath) as! WMWalkRankTableViewCell

// if self.sortArray.count > 0 {

let dic = self.sortArray[indexPath.row]

cell.dic = dic

if (indexPath.row == 0){

if (dic["rank"] as! Int != 1) {

cell.stepChangeLabel.isHidden = false

cell.rankLabelCenterY.constant = 7

cell.rankLabel.textColor = UIColor.hexString("#1989FF", a: 1)

let selfSteps = dic["steps"] as! Int

let agoSteps = dic["ago\_steps"] as! Int

cell.stepChangeLabel.text = "+\(agoSteps - selfSteps)步排名提升"

}else{ // 自己就是第一名，不用显示+？步排名提升

cell.stepChangeLabel.isHidden = true

cell.rankLabelCenterY.constant = 0

cell.rankLabel.textColor = UIColor.hexString("#1989FF", a: 1)

}

}else{

cell.stepChangeLabel.isHidden = true

cell.rankLabelCenterY.constant = 0

cell.rankLabel.textColor = UIColor.hexString("#000000", a: 0.7)

}

// }

return cell

}else{

return self.sortNoDataCell

}

}

}

func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath){

print(indexPath)

}

func tableView(\_ tableView: UITableView, willDisplayHeaderView view: UIView, forSection section: Int) {

if view.isKind(of: UITableViewHeaderFooterView.self) {

view.tintColor = UIColor.clear

let header = view as! UITableViewHeaderFooterView

header.textLabel?.textAlignment = .left

header.textLabel?.textColor = UIColor.hexString("#1989FF", a: 1.0)

}

}

// 禁止header悬浮

func scrollViewDidScroll(\_ scrollView: UIScrollView) {

let sectionHeaderHeight: CGFloat = 33.9

if scrollView.contentOffset.y <= sectionHeaderHeight && scrollView.contentOffset.y >= 0 {

scrollView.contentInset = UIEdgeInsets(top: -scrollView.contentOffset.y, left: 0, bottom: 0, right: 0)

} else if scrollView.contentOffset.y >= sectionHeaderHeight {

scrollView.contentInset = UIEdgeInsets(top: -sectionHeaderHeight, left: 0, bottom: 0, right: 0)

}

}

// 上下浮动动画

func upDownAnimate(view: UIView) -> Void {

UIView.animate(withDuration: 1) {

view.frame = CGRect(x: view.x, y: view.y+10, width: view.width, height: view.height)

}

UIView.animate(withDuration: 1, animations: {

}) { (finished) in

}

UIView.animate(withDuration: 1, delay: 1, options: [UIView.AnimationOptions.allowUserInteraction , UIView.AnimationOptions.curveEaseInOut], animations: {

view.frame = CGRect(x: view.x, y: view.y-10, width: view.width, height: view.height)

}) { (finished) in

self.upDownAnimate(view: view)

}

}

// MARK: - Action

@objc func PointViewAction(\_ sender: UITapGestureRecognizer) {

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

print("点击了第\(String(describing: sender.view?.tag))个气泡")

// 领取金币弹窗

if (self.pointDataArray[sender.view!.tag]["name"]! as! String) == "签到" {

// 先查询当前连续签到日期,根据日期返回奖励金币

self.loadSignInCount(idStr: self.pointDataArray[sender.view!.tag]["id"]! as! String)

}else{

let vc = WMReceiveCoinViewController()

vc.coins = self.pointDataArray[sender.view!.tag]["coins"] as! Int

vc.modalPresentationStyle = .overFullScreen

vc.AdBlock = {

// 激活事件

self.loadEventsActivate(idStr: self.pointDataArray[sender.view!.tag]["id"]! as! String, coin: 0)

}

self.present(vc, animated: false) {

}

}

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

// 查询连续签到

func loadSignInCount(idStr:String) -> Void {

WMNetTools.share.loadEventsContinueCount(name: "签到") { (continueDay) in

if continueDay == "expired"{// token 失效

self.loadSignInCount(idStr: idStr)

}else{

// 计算签到奖励的金币

let coin = coinWithSignInDay(nextDay: ((Int(continueDay)!%7==0) ? Int(continueDay)! : Int(continueDay)!%7) + 1)

let vc = WMReceiveCoinViewController()

vc.coins = coin

vc.AdBlock = {

// 激活事件

self.loadEventsActivate(idStr: idStr, coin: coin)

}

vc.modalPresentationStyle = .overFullScreen

self.present(vc, animated: false) {

}

}

}

}

/// 激活事件

/// - Parameters:

/// - idStr: 事件id

/// - coin: 签到奖励的金币数量

/// - Returns: void

func loadEventsActivate(idStr:String,coin:Int) -> Void {

let model = WMUserModel.share.readUserModel()

var dic = ["uuid":model.uuid,"id":idStr] as [String : Any]

if coin > 0 {

dic.updateValue(coin, forKey: "force\_add\_coins")

}else{

// Do noting

}

WMNetTools.share.postDataRequest(url: "events/activate/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadEventsActivate(idStr: idStr, coin: coin)

}else{

// ZKProgressHUD.showMessage("金币领取成功")

// 刷新气泡

self.loadEventsQuery()

}

}) { (error) in

print("\(error)")

}

}

@IBAction func todayDataWaterButtonAction(\_ sender: UIButton) {

print("喝水")

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

let vc = WMItemDetailViewController()

vc.viewType = .water

vc.modalPresentationStyle = .overFullScreen

self.present(vc, animated: false) {

}

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

@IBAction func todayDataGetupButtonAction(\_ sender: UIButton) {

print("早起")

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

let vc = WMItemDetailViewController()

vc.viewType = .getup

vc.modalPresentationStyle = .overFullScreen

self.present(vc, animated: false) {

}

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

@IBAction func todayDataSignInButtonAction(\_ sender: UIButton) {

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

print("签到")

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

@IBAction func todayDataWalkButtonAction(\_ sender: Any) {

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

print("一万步")

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

@IBAction func receiveButtonAction(\_ sender: UIButton) {

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

print("领百元现金")

let vc = LBWebViewController.init()

vc.url = WebUrl\_InviteFriend

vc.title = "邀请好友"

vc.isShowNavigation = false

vc.backButtonTitle = "走路"

self.navigationController?.pushViewController(vc, animated: false)

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

@IBAction func stepRecordButtonAction(\_ sender: Any) {

print("步数记录")

let vc = WMStepRecordViewController()

self.navigationController?.pushViewController(vc, animated: true)

}

@IBAction func noDataButtonAction(\_ sender: Any) {

// 跳转登陆界面

self.pushLoginVC()

}

// MARK: - LoadData

func setStepCount() -> Void {

WMStepTools.share.getStepCount { (dic) in

print("设备获取步数--> \(dic) count--> \(dic.count)")

if dic.count > 0{

// 获取总步数赋值UI

DispatchQueue.main.async {

self.distanceLabel.text = NSString.init(format: "%.2f", (dic["distance"] as! Double)/1000) as String

self.roundView.setProgress(stepCount: dic["step\_count"] as! Int)

self.kcalLabel.text = String.init(format: "%.2f", KcalWithStepsCount(stepsCount: (dic["step\_count"] as! Int)))

self.stepTimeLabel.text = stepsTimeWithDistance(distance: (dic["distance"] as! Double)/1000)

}

// 登陆状态下，获取add步数上传服务器

if WMUserModel.share.isLogin() == true {

// 增长步数大于0上传服务器

if (dic["addstep\_count"] as! Int) > 0 {

WMNetTools.share.getCurrentDate { (timeStamp) in

if timeStamp == 0{

// Do noting

}else{

//指定一个Region

let rome = Region(calendar: Calendars.gregorian, zone: Zones.asiaShanghai,

locale: Locales.chinese)

// 从时间间隔创建

let nowDate = DateInRegion(seconds: Double(timeStamp/1000), region: rome)

let model = WMUserModel.share.readUserModel()

let dic2 = ["uuid":model.uuid, "steps":dic["addstep\_count"] as! Int, "year":"\(nowDate.dateComponents.year!)", "month":"\(nowDate.dateComponents.month!)", "day":"\(nowDate.dateComponents.day!)"] as [String : Any]

print("增加步数接口参数--> \(dic2)")

// 上传步数

self.uploadStepsAdd(dic: dic2, timeStamp: timeStamp, todayAllSteps: dic["step\_count"] as! Int)

}

}

}else{

// Do noting

}

}else{

// Do noting

}

}else{

// 网络状态不好,添加通知

//监听通知 网络状态变化

NotificationCenter.default.removeObserver(self, name: NSNotification.Name(rawValue: "NetworkStatusChangedNotiName"), object: nil)

NotificationCenter.default.addObserver(self, selector: #selector(self.NetworkStatusChanged(noti:)), name: NSNotification.Name(rawValue: "NetworkStatusChangedNotiName"), object: nil)

}

}

}

func uploadStepsAdd(dic:Dictionary<String, Any>, timeStamp:Int, todayAllSteps:Int) -> Void {

WMNetTools.share.postDataRequest(url: "steps/add/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.uploadStepsAdd(dic: dic, timeStamp: timeStamp, todayAllSteps: todayAllSteps)

}else{

// 上传成功步数本地存储上传步数、上传时间

// 存储数据

let saveBool = KeychainManager.keyChainSaveData(data: todayAllSteps as Any, withIdentifier: "TodayLastStepsCount")

if saveBool {

print("keychain存储步数成功")

}else{

print("keychain存储步数失败")

}

let saveBool2 = KeychainManager.keyChainSaveData(data: timeStamp as Any, withIdentifier: "UpLoadStepTime")

if saveBool2 {

print("keychain存储时间戳成功")

}else{

print("keychain存储时间戳失败")

}

}

}) { (error) in

print("\(error)")

}

}

func loadEventsQuery() -> Void {

let model = WMUserModel.share.readUserModel()

let dic = ["uuid":model.uuid] as [String : Any]// 有效事件

// 查询当前有效事件信息

WMNetTools.share.postDataRequest(url: "events/query/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadEventsQuery()

}else{

self.pointDataArray.removeAll()

var pointAllDataArray = Array<Dictionary<String, Any>>()

let dic = getDictionaryFromJSONString(jsonString: response)

let eventsArray = dic["events"] as! Array<Dictionary<String, Any>>

for dic:Dictionary<String, Any> in eventsArray {

let show\_placeArray = dic["show\_place"] as! Array<String>

if show\_placeArray.contains("bubble") {// 气泡类型

pointAllDataArray.append(dic)

}else{

// Do noting

}

}

if pointAllDataArray.count > 0 {

// 先计算接口返回的气泡数量是否有4个

if self.pointArray.count > pointAllDataArray.count {

// 根据返回的气泡个数取值，隐藏多余的pointView

self.pointDataArray = pointAllDataArray

}else{

// 取前四个

self.pointDataArray = Array(pointAllDataArray[0...3])

}

// 隐藏多余的气泡

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

if i < self.pointDataArray.count {

self.pointArray[i].isHidden = false

}else{

self.pointArray[i].isHidden = true

}

}

// 分配气泡数据

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

let dic = self.pointDataArray[i] as Dictionary<String, Any>

self.pointCoinLabelArray[i].text = String(dic["coins"] as! Int)

self.pointNameLabelArray[i].text = dic["name"] as? String

}

}else{

// 隐藏所有气泡

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

self.pointArray[i].isHidden = true

}

}

}

}) { (error) in

print("\(error)")

}

}

func loadNewUserRedEnvelopeData() -> Void {

let model = WMUserModel.share.readUserModel()

let dic = ["uuid":model.uuid] as [String : Any]// 有效事件

// 查询当前有效事件信息

WMNetTools.share.postDataRequest(url: "events/query/", params: dic, isToken: true, success: { (response) in

if response == "expired" {// token 失效

self.loadNewUserRedEnvelopeData()

}else{

let dic = getDictionaryFromJSONString(jsonString: response)

let eventsArray = dic["events"] as! Array<Dictionary<String, Any>>

for dic:Dictionary<String, Any> in eventsArray {

let show\_placeArray = dic["show\_place"] as! Array<String>

if show\_placeArray.contains("firstopen"){// 新人红包

// 展开新人红包

let redEnvelope = Bundle.main.loadNibNamed("WMNewUserRedEnvelopeView", owner: self, options: nil)?.first as! WMNewUserRedEnvelopeView

redEnvelope.backgroundColor = UIColor.black.withAlphaComponent(0.5)

redEnvelope.frame = self.view.bounds

redEnvelope.dic = dic

self.tabBarController?.view.insertSubview(redEnvelope, at: (self.tabBarController?.view.subviews.count)!-1)

}else{

// Do noting

}

}

}

}) { (error) in

print("\(error)")

}

}

// 排名

func loadUsersSortData() -> Void {

let model = WMUserModel.share.readUserModel()

let dic = ["uuid":model.uuid] as [String : Any]//page:默认多少名

// 查询当前有效事件信息

WMNetTools.share.postDataRequest(url: "users/sort/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadUsersSortData()

}else{

self.sortArray.removeAll()

let dic = getDictionaryFromJSONString(jsonString: response)

let array = dic["result"] as! Array<Dictionary<String, Any>>

if array.count > 0 {

// self.sortArray = array

/\*

// 找出自己

// 查询数组里自己的数据是否包含排名 // 比如参数名pos

// 查询数组里数否包含上一名的数据 // 比如参数名higher\_steps

if 没有排名{

按照数组顺序排列名次

根据下标，将自己挪到数组第一位,记录自己的排名名次

根据数组顺序展示其余人名次

}else{

将自己挪到数组第一位,根据排名展示自己的排名名次

根据数组顺序展示其余人名次

}

\*/

var outArray = Array<Dictionary<String, Any>>.init()

for i in 0..<array.count {

var dic2 = array[i] as Dictionary<String, Any>

if (dic2.keys.contains("me")) && (dic2["me"] as! Bool == true) {

// 是否包含排名参数pos

if (dic2.keys.contains("pos")) {// 使用排名

dic2.updateValue(dic2["pos"] as! Int, forKey: "rank")

}else{// 使用下标

dic2.updateValue(i+1, forKey: "rank")

}

// 是否包含上一名用户步数参数higher\_steps

if (dic2.keys.contains("higher\_steps")) {// 使用参数

dic2.updateValue(dic2["higher\_steps"] as! Int, forKey: "ago\_steps")

}else{// 使用下标查找上一位用户的步数

if i == 0 {//第一名不用找上一名用户

}else{

let dic3 = array[i-1] as Dictionary<String, Any>

dic2.updateValue(dic3["steps"] as! Int, forKey: "ago\_steps")

}

}

outArray.insert(dic2, at: outArray.startIndex)

}else{

dic2.updateValue(i+1, forKey: "rank")

outArray.append(dic2)

}

}

self.sortArray = outArray

self.tableView.reloadData()

}else{

// 理论接口不会返回小于0的数据，因为有自己的数据

}

}

}) { (error) in

print("\(error)")

}

}

func loadEventsAllCount(name:String) -> Void {

WMNetTools.share.loadEventsActivateCount(name: name) { (allDay) in

if allDay == "expired"{// token 失效

self.loadEventsAllCount(name: name)

}else{

switch name {

case "喝水":

self.eightWaterDayLabel.text = allDay

case "早起":

self.morningDayLabel.text = allDay

case "签到":

self.allSignInDayLabel.text = allDay

case "每天一万步":

self.tenThousandStepsDayLabel.text = allDay

default:

break

}

}

}

}

func pushLoginVC() -> Void {

let vc = WMLoginViewController()

vc.siginBlock = {

// 登陆成功回调

// 上传步数

self.setStepCount()

}

self.present(vc, animated: true) {

}

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

import UIKit

class WMStepRecordViewController: WMBaseViewController {

var roundView = LBRoundProgressView()

@IBOutlet weak var alreadyExchangeStepCountLabel: UILabel!

@IBOutlet weak var scrollView: UIScrollView!

@IBOutlet weak var contentView: UIView!

@IBOutlet weak var distanceLabel: UILabel! // 步行里程

@IBOutlet weak var stepTimeLabel: UILabel! // 步行时间

@IBOutlet weak var kcalLabel: UILabel!// 消耗能量

@IBOutlet weak var targetPercentLabel: UILabel!

@IBOutlet weak var kcalDetailButton: UIButton!

var userSteps : Int = 0 // 今日步数

var alreadySteps : Int = 0 // 已兑换步数

var chartsView = WMChartsView() // 柱状图

var chartsArray : Array<Dictionary<String, Any>> = [] // 柱状图数组

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

WMStepTools.share.getStepCount { (dic) in

DispatchQueue.main.async {

if dic.count > 0{

self.distanceLabel.text = NSString.init(format: "%.2f", (dic["distance"] as! Double)/1000) as String

self.kcalLabel.text = String.init(format: "%.2f", KcalWithStepsCount(stepsCount: (dic["step\_count"] as! Int)))

self.stepTimeLabel.text = stepsTimeWithDistance(distance: (dic["distance"] as! Double)/1000)

self.userSteps = dic["step\_count"] as! Int

self.roundView.setProgress(stepCount: dic["step\_count"] as! Int)

self.alreadyExchangeStepCountLabel.text = "已兑换 -/\(dic["step\_count"] as! Int)"

let step = UserDefaults.standard.value(forKey: "TargetStepCount") as! Double

let percent = CGFloat(Double(dic["step\_count"] as! Int)/step) // 设置当前属性的值

self.targetPercentLabel.text = String.init(format: "目标完成%.0f%%", percent\*100)

/\* 食物计算逻辑

3个苹果 = 1个鸡腿

3个鸡腿 = 1个巧克力

苹果57kcal 鸡腿177kcal 巧克力586kcal

< 57 0， 0

> 57 && <177 单位苹果

> 177 && <586 单位鸡腿

> 586 单位巧克力

5.3大卡=1克碳排放

\*/

var kcal = 0.0

kcal = KcalWithStepsCount(stepsCount: (dic["step\_count"] as! Int))

var foodText = ""

if kcal < 57{

foodText = "消耗热量=0, 减少了0克碳排放"

}else if kcal > 57 && kcal < 177{

foodText = "消耗热量=\(Int(kcal/57))个🍎, 减少了\(Int(kcal/5.3))克碳排放"

}else if kcal > 177 && kcal < 586{

foodText = "消耗热量=\(Int(kcal/177))个🍗, 减少了\(Int(kcal/5.3))克碳排放"

}else{

foodText = "消耗热量=\(Int(kcal/586))个🍫, 减少了\(Int(kcal/5.3))克碳排放"

}

self.kcalDetailButton.setTitle(foodText, for: .normal)

// 请求已兑换步数

// 判断登陆状态下请求数据

if WMUserModel.share.isLogin() == true{

self.loadStepsConsumedData()

}else{

}

// 设置柱状图信息

self.setChartsView()

}else{

// 网络状态不好,没有网

}

}

}

}

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

self.title = "步数记录"

self.view.backgroundColor = UIColor.white

// 中间进度条

roundView = LBRoundProgressView.init(frame: CGRect.init(x: (SCREEN\_WIDTH - 200)/2, y: 25, width: 200, height: 200), lineWidth: 16)

roundView.backgroundColor = UIColor.clear

self.contentView.addSubview(roundView)

// 柱状图

chartsView = Bundle.main.loadNibNamed("WMChartsView", owner: self, options: nil)?.first as! WMChartsView

chartsView.frame = CGRect(x: 16, y: 368, width: self.contentView.width-32, height: 150)

chartsView.layer.masksToBounds = true

chartsView.layer.cornerRadius = 8

// 渐变

let cgView = CGView(frame: CGRect(x: 0, y: 0, width: SCREEN\_WIDTH-32, height: 150))

let compoents:[CGFloat] = [54/255, 190/255, 255/255, 1,

25/255, 138/255, 255/255, 1]

cgView.colors = compoents

cgView.startPoint = CGPoint(x: cgView.bounds.minX, y: cgView.bounds.minY)

cgView.endPoint = CGPoint(x: cgView.bounds.minX, y: cgView.bounds.maxY)

chartsView.insertSubview(cgView, at: 0)

self.contentView.addSubview(chartsView)

}

// 设置柱状图信息

func setChartsView() -> Void {

// 柱状图数组初始化及请求信息

WMNetTools.share.getCurrentDate { (timeStamp) in

if timeStamp == 0{

// Do noting

}else{

for i in 0..<10 {

//指定一个Region

let rome = Region(calendar: Calendars.gregorian, zone: Zones.asiaShanghai,

locale: Locales.chinese)

// 从时间间隔创建

let date = DateInRegion(seconds: Double((timeStamp - i\*3600\*24\*1000)/1000), region: rome)

// 现在时间(获取步数截止时间)

let date2 = DateInRegion(components: {

$0.year = date.dateComponents.year

$0.month = date.dateComponents.month

$0.day = date.dateComponents.day

})

let formatter = DateFormatter()

formatter.dateFormat = "yyyy-MM-dd"

let dateStr = formatter.string(from: date2!.date)

var dic = Dictionary<String, Any>.init()

if i == 0 {// 今天的取本地的数据

dic = [dateStr : self.userSteps] as Dictionary<String, Any>

}else{// 其余初始化为0

dic = [dateStr : 0] as Dictionary<String, Any>

}

self.chartsArray.insert(dic, at: 0)

}

// 如果登陆状态下

// 判断登陆状态下请求数据

if WMUserModel.share.isLogin() == true{

self.loadStepsQueryData()

}else{

self.chartsView.array = self.chartsArray

}

}

}

}

// MARK: - LoadData

// 已兑换步数

func loadStepsConsumedData() -> Void {

let userModel = WMUserModel.share.readUserModel()

let dic = ["uuid":userModel.uuid]

WMNetTools.share.postDataRequest(url: "steps/consumed/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadStepsConsumedData()

}else{

let dic2 = getDictionaryFromJSONString(jsonString: response)

self.alreadySteps = Int((dic2["consumed\_steps"] as! String))!

self.alreadyExchangeStepCountLabel.text = "已兑换 \(dic2["consumed\_steps"]!)/\(self.userSteps)"

}

}) { (error) in

print("\(error)")

}

}

// 连续天数步数信息

func loadStepsQueryData() -> Void {

let userModel = WMUserModel.share.readUserModel()

let dic = ["uuid":userModel.uuid,"days":10] as [String : Any]

WMNetTools.share.postDataRequest(url: "steps/query/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadStepsQueryData()

}else{

let dic = getDictionaryFromJSONString(jsonString: response)

let array = dic["result"] as! Array<Dictionary<String, Any>>

if array.count > 0 {

for indexDic:Dictionary in array {

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

let dic2 = self.chartsArray[i]

if (indexDic["date"] as! String) == dic2.keys.first {

// 替换

let dic3 = [(indexDic["date"] as! String) : (indexDic["steps"] as! Int)] as Dictionary<String, Any>

self.chartsArray[i] = dic3

}

}

}

}else{

}

// 更新柱状图

self.chartsView.array = self.chartsArray

}

}) { (error) in

print("\(error)")

}

}

// MARK: - Action

@IBAction func exchangeCoinButtonAction(\_ sender: Any) {

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

print("兑换金币")

// 可兑换步数 = 总步数 - 已兑换步数

let canExchangeSteps = self.userSteps - self.alreadySteps

if canExchangeSteps < 1000 {

ZKProgressHUD.showMessage("步数不够，一次最少需要兑换1000步")

return

}

// let userModel = WMUserModel.share.readUserModel()

// // 一次最少兑换1000步，最多兑换9990步,(中间取1000的整倍数)

// let dic = ["uuid":userModel.uuid, "steps":canExchangeSteps>9990 ? 9990 : (canExchangeSteps - canExchangeSteps%1000)] as [String : Any]

// WMNetTools.share.postDataRequest(url: "steps/exchange\_for\_coins/", params: dic, isToken: true, success: { (response) in

//

// if response == "expired"{// token 失效

// self.exchangeCoinButtonAction(sender)

// }else{

// ZKProgressHUD.showMessage("金币兑换成功")

// self.loadStepsConsumedData()

// }

//

// }) { (error) in

// print("\(error)")

//

// }

// 要兑换的步数

let steps = canExchangeSteps>9990 ? 9990 : (canExchangeSteps - canExchangeSteps%1000)

// 领取金币弹窗

let vc = WMReceiveCoinViewController()

vc.coins = steps/10

vc.modalPresentationStyle = .overFullScreen

vc.AdBlock = {

// 兑换步数

self.loadStepsExchangeForCoins(steps: steps)

}

self.present(vc, animated: false) {

}

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

// 兑换步数

func loadStepsExchangeForCoins(steps:Int) -> Void {

let userModel = WMUserModel.share.readUserModel()

// 一次最少兑换1000步，最多兑换9990步,(中间取1000的整倍数)

let dic = ["uuid":userModel.uuid, "steps":steps] as [String : Any]

WMNetTools.share.postDataRequest(url: "steps/exchange\_for\_coins/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadStepsExchangeForCoins(steps: steps)

}else{

// ZKProgressHUD.showMessage("金币兑换成功")

self.loadStepsConsumedData()

}

}) { (error) in

print("\(error)")

}

}

func pushLoginVC() -> Void {

let vc = WMLoginViewController()

vc.siginBlock = {

// 登陆成功回调

// 已兑换接口

self.loadStepsConsumedData()

// 连续天数步数信息接口

self.loadStepsQueryData()

}

self.present(vc, animated: true) {

}

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

enum ItemType {

case water // 喝水

case getup // 早起

}

import UIKit

import EventKit

class WMItemDetailViewController: WMBaseViewController {

@IBOutlet weak var bottomView: UIView!

@IBOutlet weak var titleLabel: UILabel!

@IBOutlet weak var continueDayLabel: UILabel! // 最长持续天数

@IBOutlet weak var addupLabel: UILabel! // 已完成天数

@IBOutlet weak var startDateLabel: UILabel!

@IBOutlet weak var submitButton: UIButton!

@IBOutlet weak var submitButtonLabel: UILabel!

@IBOutlet weak var countDownTimeLabel: UILabel!

var eventDic : Dictionary<String, Any> = [:] // 事件数据

var viewType : ItemType?

/// 懒加载 倒计时

lazy var countdownTimer: WMCountDown = {

let timer = WMCountDown()

// 此闭包可以在本类任意方法中写

timer.countDown = { [weak self] (d, h, m, s) in

// self?.dayLabel.text = "\(d)天:"

// self?.hourLabel.text = "\(h)时:"

// self?.minuteLabel.text = "\(m)分:"

// self?.secondLabel.text = "\(s)秒"

self?.countDownTimeLabel.text = "习惯将于 \(d):\(h):\(m):\(s) 后刷新"

}

return timer

}()

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

countdownTimer.resume() // 恢复倒计时

}

override func viewWillDisappear(\_ animated: Bool) {

super.viewWillDisappear(animated)

countdownTimer.suspend() // 停止倒计时

}

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

self.view.backgroundColor = UIColor.black.withAlphaComponent(0.5)

// 导两个圆角

let rect = CGRect(x: bottomView.bounds.origin.x, y: bottomView.bounds.origin.y, width: SCREEN\_WIDTH, height: bottomView.bounds.size.height)

let path = UIBezierPath.init(roundedRect: rect, byRoundingCorners: [.topLeft, .topRight] , cornerRadii: CGSize(width: 16, height: 16))

let layer = CAShapeLayer.init()

layer.frame = rect

layer.path = path.cgPath

bottomView.layer.mask = layer

var detailText = ""

// 赋值

switch self.viewType {

case .water:

self.titleLabel.text = "第?杯水：清理肠胃"

detailText = "早上起床时身体正处于缺水状态，喝一杯温开水，有利于清理肠胃，这时也正是肾脏解毒的时间段，有利于排除身体毒素，促进身体恢复正常的新陈代谢。"

self.submitButtonLabel.text = "喝水"

self.loadEventsDetails(name: "喝水")

case .getup:

self.titleLabel.text = "早起"

detailText = "早睡早起可增强免疫力，有益抗击感冒等病侵入。 不仅如此，早睡早起还可以大大降低心脏病、肾病、高血压、糖尿病和中风等慢性病风险。"

self.submitButtonLabel.text = "早起"

self.loadEventsDetails(name: "早起")

default:

break

}

// 文字说明

let label = UILabel()

label.frame = CGRect(x: 22, y: 57, width: SCREEN\_WIDTH - 44, height: 60)

label.numberOfLines = 0

let attrString = NSMutableAttributedString(string: detailText)

let attr: [NSAttributedString.Key : Any] = [.font: UIFont.systemFont(ofSize: 14),.foregroundColor: UIColor(red: 0, green: 0, blue: 0, alpha: 0.4)]

attrString.addAttributes(attr, range: NSRange(location: 0, length: attrString.length))

label.attributedText = attrString

self.bottomView.addSubview(label)

// 总数

WMNetTools.share.loadEventsActivateCount(name: self.submitButtonLabel.text!) { (allDay) in

self.addupLabel.text = allDay

}

// 连续天数

WMNetTools.share.loadEventsContinueCount(name: self.submitButtonLabel.text!) { (day) in

self.continueDayLabel.text = day

}

// 开始倒计时

// 可以传递开始时间参数，用于计算倒计时时间差，不传默认从系统当前时间开始计算时间差

countdownTimer.start(with: "1991-11-12 14:00:00", end: "1992-11-12 18:00:00")

// countdownTimer.start(with: nil, end: "2020-10-10 19:00:00")

}

deinit {

// 可写可不写

countdownTimer.stop()

}

// MARK: - LoadData

// 查询事件

func loadEventsDetails(name:String) -> Void {

let model = WMUserModel.share.readUserModel()

let dic = ["uuid":model.uuid, "query":["name":name]] as [String : Any]// 全部事件

WMNetTools.share.postDataRequest(url: "events/query/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadEventsDetails(name: name)

}else{

let dic = getDictionaryFromJSONString(jsonString: response)

let array = dic["events"] as! Array<Dictionary<String, Any>>

if array.count > 0 {

self.eventDic = array.first! as Dictionary<String, Any>

if (array.first!["activate"] as! Bool) == true{// 已激活

self.submitButton.setImage(UIImage.init(named: "btn\_bg\_no"), for: .normal)

self.submitButton.isUserInteractionEnabled = false

}else{// 未激活

self.submitButton.setImage(UIImage.init(named: "btn\_bg"), for: .normal)

self.submitButton.isUserInteractionEnabled = true

}

}else{

}

}

}) { (error) in

print("\(error)")

}

}

// 激活事件

func loadActivateEvents(id:String) -> Void {

let model = WMUserModel.share.readUserModel()

let dic = ["uuid":model.uuid,"id":id] as [String : Any]

WMNetTools.share.postDataRequest(url: "events/activate/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadActivateEvents(id: id)

}else{

// ZKProgressHUD.showMessage("金币领取成功")

// 刷新查询事件接口

self.loadEventsDetails(name: self.eventDic["name"] as! String)

}

}) { (error) in

print("\(error)")

}

}

// MARK: - Action

@IBAction func closeButtonAction(\_ sender: Any) {

self.dismiss(animated: false) {

}

}

@IBAction func itemButtonAction(\_ sender: Any) {

if self.eventDic.count > 0 {

// 激活事件

// 领取金币弹窗

let vc = WMReceiveCoinViewController()

vc.coins = self.eventDic["coins"] as! Int

vc.modalPresentationStyle = .overFullScreen

vc.AdBlock = {

// 激活事件

self.loadActivateEvents(id: self.eventDic["id"] as! String)

}

self.present(vc, animated: false) {

}

}else{

ZKProgressHUD.showMessage("早起事件name没有查询到数据")

}

}

@IBAction func riliButtonAction(\_ sender: Any) {

print("日历")

let eventStore = EKEventStore.init()

eventStore.requestAccess(to: EKEntityType.event, completion: {(isAllow, error) in

DispatchQueue.main.async {

if (error != nil) {

print("添加日历事件错误...")

}else if (isAllow == false) {

// 被用户拒绝

ZKProgressHUD.showMessage("请前往设置开启日历权限")

}else{

// 添加日历事件

switch self.viewType {

case .water:

WMEventTools.share.checkEvent { (array) in

if array.count > 0{

var isHave : Bool = false

for i in 0..<array.count {

let event = array[i] as! EKEvent

if event.title.contains("杯水") {

isHave = true

}else{

// Do noting

}

}

if isHave == true {

ZKProgressHUD.showMessage("日历中已有喝水事件")

}else{

// 添加事件

self.addRiliEventWater()

}

}else{

// 添加事件

self.addRiliEventWater()

}

}

case .getup:

WMEventTools.share.checkEvent { (array) in

if array.count > 0{

var isHave : Bool = false

for i in 0..<array.count {

let event = array[i] as! EKEvent

if event.title.contains("早起") {

isHave = true

}else{

// Do noting

}

}

if isHave == true {

ZKProgressHUD.showMessage("日历中已有早起事件")

}else{

// 添加事件

self.addRiliEventGetup()

}

}else{

// 添加事件

self.addRiliEventGetup()

}

}

default:

break

}

}

}

})

}

func addRiliEventWater() -> Void {

// 固定时间触发：7，9，11.30，13.30，15.30，17.30，19，21（结束时间为下一杯水的开始时间）

//指定一个Region

let rome = Region(calendar: Calendars.gregorian, zone: Zones.asiaShanghai,

locale: Locales.chinese)

let array = [

["startTime":["hour":7,"minute":0],"endTime":["hour":9,"minute":0]], ["startTime":["hour":9,"minute":0],"endTime":["hour":11,"minute":30]], ["startTime":["hour":11,"minute":30],"endTime":["hour":13,"minute":30]], ["startTime":["hour":13,"minute":30],"endTime":["hour":15,"minute":30]], ["startTime":["hour":15,"minute":30],"endTime":["hour":17,"minute":30]], ["startTime":["hour":17,"minute":30],"endTime":["hour":19,"minute":0]], ["startTime":["hour":19,"minute":0],"endTime":["hour":21,"minute":0]],

["startTime":["hour":21,"minute":0],"endTime":["hour":24,"minute":0]]

]

for i in 0..<array.count {

let dic = array[i]

let startDate = DateInRegion(year: Date().year, month: Date().month, day: Date().day, hour: dic["startTime"]!["hour"]!, minute: dic["startTime"]!["minute"]!, second: 0, nanosecond: 0, region: rome)

let endDate = DateInRegion(year: Date().year, month: Date().month, day: Date().day, hour: dic["endTime"]!["hour"]!, minute: dic["endTime"]!["minute"]!, second: 0, nanosecond: 0, region: rome)

WMEventTools.share.addACalendarTime("喝第\(i+1)杯水", eventNotes: "喝第\(i+1)杯水日历事件", eventRepeatTime: startDate.date, eventEndTime: endDate.date, eventRepeatDate: "0,1,2,3,4,5,6", eventAlarmTime: 1, eventIdentifier: { (identitier) in

ZKProgressHUD.showMessage("新增喝水日历事件成功")

}) {

print("添加日历事件错误--> 喝水")

}

}

}

func addRiliEventGetup() -> Void {

// 固定时间段：5:00-7:30

//指定一个Region

let rome = Region(calendar: Calendars.gregorian, zone: Zones.asiaShanghai,

locale: Locales.chinese)

let startDate = DateInRegion(year: Date().year, month: Date().month, day: Date().day, hour: 5, minute: 0, second: 0, nanosecond: 0, region: rome)

let endDate = DateInRegion(year: Date().year, month: Date().month, day: Date().day, hour: 7, minute: 30, second: 0, nanosecond: 0, region: rome)

WMEventTools.share.addACalendarTime("早起", eventNotes: "早起日历事件", eventRepeatTime: startDate.date, eventEndTime: endDate.date, eventRepeatDate: "0,1,2,3,4,5,6", eventAlarmTime: 1, eventIdentifier: { (identitier) in

ZKProgressHUD.showMessage("新增早起日历事件成功")

}) {

print("添加日历事件错误--> 早起")

}

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

import UIKit

class WMNewUserRedEnvelopeView: UIView {

/\*

// Only override draw() if you perform custom drawing.

// An empty implementation adversely affects performance during animation.

override func draw(\_ rect: CGRect) {

// Drawing code

}

\*/

@IBOutlet weak var closeBackgroundButton: UIButton!

@IBOutlet weak var closeLabel1: UILabel!

@IBOutlet weak var closeLabel2: UILabel!

@IBOutlet weak var closeCloseButton: UIButton!

@IBOutlet weak var openBackgroundLabel: UIButton!

@IBOutlet weak var openCoinsLabel: UILabel!

var dic : Dictionary<String, Any> = [:]

// MARK: - Action

@IBAction func closeButtonAction(\_ sender: Any) {

self.removeFromSuperview()

}

@IBAction func openButtonAction(\_ sender: Any) {

print("打开红包")

// 激活事件

self.loadEventsActivate(idStr: self.dic["id"]! as! String)

}

func loadEventsActivate(idStr:String) -> Void {

// 激活事件

let model = WMUserModel.share.readUserModel()

let dic = ["uuid":model.uuid,"id":idStr] as [String : Any]

WMNetTools.share.postDataRequest(url: "events/activate/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadEventsActivate(idStr: idStr)

}else{

self.closeBackgroundButton.isHidden = true

self.closeLabel1.isHidden = true

self.closeLabel2.isHidden = true

self.closeCloseButton.isHidden = true

self.openBackgroundLabel.isHidden = false

self.openCoinsLabel.isHidden = false

self.openCoinsLabel.text = String(self.dic["coins"] as! Int)

}

}) { (error) in

print("\(error)")

}

}

}

import UIKit

import BUAdSDK

typealias AdSuccessBlock = () -> Void

class WMReceiveCoinViewController: WMBaseViewController,BUNativeExpressFullscreenVideoAdDelegate {

var AdBlock : AdSuccessBlock?

@IBOutlet weak var coinsLabel: UILabel!

@IBOutlet weak var firstCloseButton: UIButton!

@IBOutlet weak var firstReceiveButton: UIButton!

@IBOutlet weak var secondButton: UIButton!

@IBOutlet weak var secondCoinsLabel: UILabel!

var coins : Int = 0

var videoAd = BUNativeExpressFullscreenVideoAd()

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

self.view.backgroundColor = UIColor.black.withAlphaComponent(0.5)

self.coinsLabel.text = "+\(coins)"

self.secondCoinsLabel.text = "+\(coins)"

// 穿山甲广告

videoAd = BUNativeExpressFullscreenVideoAd.init(slotID: "945448371")

videoAd.delegate = self

videoAd.loadData()

}

// MARK: - Action

@IBAction func closeButtonAction(\_ sender: Any) {

self.dismiss(animated: false) {

}

}

@IBAction func receiveButtonAction(\_ sender: Any) {

videoAd.show(fromRootViewController: self)

}

// MARK: - BUNativeExpressFullscreenVideoAdDelegate

func nativeExpressFullscreenVideoAdDidClose(\_ fullscreenVideoAd: BUNativeExpressFullscreenVideoAd) {

self.coinsLabel.isHidden = true

self.firstCloseButton.isHidden = true

self.firstReceiveButton.isHidden = true

self.secondButton.isHidden = false

self.secondCoinsLabel.isHidden = false

// 请求激活事件接口

if (self.AdBlock != nil) {

self.AdBlock!()

}

}

func nativeExpressFullscreenVideoAdViewRenderFail(\_ rewardedVideoAd: BUNativeExpressFullscreenVideoAd, error: Error?) {

}

func nativeExpressFullscreenVideoAdViewRenderSuccess(\_ rewardedVideoAd: BUNativeExpressFullscreenVideoAd) {

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

import UIKit

typealias receiveButtonActionBlock = (\_ dic:Dictionary<String, Any>) -> Void

class WMMakeMoneyTableViewCell: UITableViewCell {

var receiveButtonBlock : receiveButtonActionBlock?

@IBOutlet weak var nameLabel: UILabel!

@IBOutlet weak var introductionLabel: UILabel!

@IBOutlet weak var receiveButton: UIButton!

@IBOutlet weak var coinImageView: UIImageView!

@IBOutlet weak var coinNumberLabel: UILabel!

var dic : Dictionary<String, Any> = [:]{

didSet{

self.nameLabel.text = dic["name"] as? String

self.introductionLabel.text = dic["description"] as? String

if (dic["status"] as! String) == "valid"{

self.receiveButton.isHidden = false

self.coinImageView.isHidden = true

self.coinNumberLabel.isHidden = true

}else{

self.receiveButton.isHidden = true

self.coinImageView.isHidden = false

self.coinNumberLabel.isHidden = false

self.coinNumberLabel.text = String(dic["coins"] as! Int)

}

}

}

override func awakeFromNib() {

super.awakeFromNib()

// Initialization code

}

override func setSelected(\_ selected: Bool, animated: Bool) {

super.setSelected(selected, animated: animated)

// Configure the view for the selected state

}

// MARK: - Action

@IBAction func receiveButtonAction(\_ sender: UIButton) {

if self.receiveButtonBlock != nil {

self.receiveButtonBlock!(self.dic)

}

}

}

import UIKit

class WMMakeMoneyViewController: WMBaseViewController,UITableViewDelegate,UITableViewDataSource {

@IBOutlet weak var tableView: UITableView!

@IBOutlet var signInCell: UITableViewCell!

@IBOutlet var lotteryCell: UITableViewCell!

// 签到UI

@IBOutlet weak var signInImageView0: UIImageView!

@IBOutlet weak var signInImageView1: UIImageView!

@IBOutlet weak var signInImageView2: UIImageView!

@IBOutlet weak var signInImageView3: UIImageView!

@IBOutlet weak var signInImageView4: UIImageView!

@IBOutlet weak var signInImageView5: UIImageView!

@IBOutlet weak var signInImageView6: UIImageView!

@IBOutlet weak var signInCountLabel: UILabel!

@IBOutlet weak var signInButton: UIButton!

var signInID = "" // 签到的ID

var signInCoins : Int = 0 // 签到的金币

var allEventsDataArray : Array<Dictionary<String, Any>> = []

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

self.navigationController?.setNavigationBarHidden(true, animated: animated)

// 登陆状态

if WMUserModel.share.isLogin() == true{

//获取连续签到状态

self.loadSignInCount()

// 查询所有事件

self.loadAllEventsQuery()

// 查询今天是否已签到

self.loadSignInEventsStatus()

}else{

// // 所有事件数据清除

// self.allEventsDataArray.removeAll()

// self.tableView.reloadData()

}

}

override func viewWillDisappear(\_ animated: Bool) {

super.viewWillDisappear(animated)

self.navigationController?.setNavigationBarHidden(false, animated: animated)

}

override func viewDidLoad() {

super.viewDidLoad()

self.title = "赚钱"

// 让tableview顶部从状态栏上开始算Y轴坐标

if #available(iOS 11.0, \*) {

self.tableView.contentInsetAdjustmentBehavior = UIScrollView.ContentInsetAdjustmentBehavior.never// 让ScrollView 不自动计算的方法

} else {

// Fallback on earlier versions

automaticallyAdjustsScrollViewInsets = false

}

// Do any additional setup after loading the view.

// 注册cell

self.tableView.register(UINib.init(nibName: "WMMakeMoneyTableViewCell", bundle: nil), forCellReuseIdentifier: "WMMakeMoneyTableViewCell")

}

// MARK: - UITableViewDelegate,UITableViewDataSource

func numberOfSections(in tableView: UITableView) -> Int{

return 3

}

func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int{

if section == 0 || section == 1 {

return 1

}else{

return self.allEventsDataArray.count

}

}

func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat{

if indexPath.section == 0 {

return 268

}else if indexPath.section == 1 {

return 120

}else{

return 56

}

}

func tableView(\_ tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat{

if section == 2{

return 33.9

}else{

return 0.01

}

}

func tableView(\_ tableView: UITableView, heightForFooterInSection section: Int) -> CGFloat{

return 0.01

}

func tableView(\_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {

if section == 2 {

return "健康好习惯"

}else{

return ""

}

}

func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell{

if indexPath.section == 0 {

return self.signInCell

}else if indexPath.section == 1 {

return self.lotteryCell

}else {

let cell = tableView.dequeueReusableCell(withIdentifier: "WMMakeMoneyTableViewCell", for: indexPath) as! WMMakeMoneyTableViewCell

cell.dic = self.allEventsDataArray[indexPath.row]

cell.receiveButtonBlock = { dic in

print(dic)

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

print("领取cell")

// 领取金币弹窗

let vc = WMReceiveCoinViewController()

if (dic["name"] as! String) == "签到" {

vc.coins = self.signInCoins

vc.AdBlock = {

// 激活事件

self.loadActivateEvents(id: dic["id"] as! String, coin: self.signInCoins)

}

}else{

vc.coins = dic["coins"] as! Int

vc.AdBlock = {

// 激活事件

self.loadActivateEvents(id: dic["id"] as! String, coin: 0)

}

}

vc.modalPresentationStyle = .overFullScreen

self.present(vc, animated: false) {

}

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

return cell

}

}

func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath){

print(indexPath)

}

func tableView(\_ tableView: UITableView, willDisplayHeaderView view: UIView, forSection section: Int) {

if view.isKind(of: UITableViewHeaderFooterView.self) {

view.tintColor = UIColor.clear

let header = view as! UITableViewHeaderFooterView

header.textLabel?.textAlignment = .left

header.textLabel?.textColor = UIColor.hexString("#1989FF", a: 1.0)

}

}

// 禁止header悬浮

func scrollViewDidScroll(\_ scrollView: UIScrollView) {

let sectionHeaderHeight: CGFloat = 33.9

if scrollView.contentOffset.y <= sectionHeaderHeight && scrollView.contentOffset.y >= 0 {

scrollView.contentInset = UIEdgeInsets(top: -scrollView.contentOffset.y, left: 0, bottom: 0, right: 0)

} else if scrollView.contentOffset.y >= sectionHeaderHeight {

scrollView.contentInset = UIEdgeInsets(top: -sectionHeaderHeight, left: 0, bottom: 0, right: 0)

}

}

// MARK: - Action

@IBAction func signInButtonAction(\_ sender: Any) {

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

print("签到")

// 领取金币弹窗

let vc = WMReceiveCoinViewController()

vc.coins = self.signInCoins

vc.modalPresentationStyle = .overFullScreen

vc.AdBlock = {

// 激活事件

self.loadActivateEvents(id: self.signInID, coin: self.signInCoins)

}

self.present(vc, animated: false) {

}

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

@IBAction func turntableButtonAction(\_ sender: Any) {

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

print("大转盘")

let vc = LBWebViewController.init()

vc.url = WebUrl\_TurnTable

vc.title = "幸运大转盘"

vc.isShowNavigation = false

vc.backButtonTitle = "赚钱"

self.navigationController?.pushViewController(vc, animated: false)

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

@IBAction func lotteryButtonAction(\_ sender: Any) {

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

print("摇奖")

let vc = LBWebViewController.init()

vc.url = WebUrl\_Prize

vc.title = "摇奖机"

vc.isShowNavigation = false

vc.backButtonTitle = "赚钱"

self.navigationController?.pushViewController(vc, animated: false)

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

// 每天签到奖励的金币数

// func coinWithSignInDay(nextDay:Int) -> Int {

//

// switch nextDay {

// case 1:

// return 78

// case 2:

// return 88

// case 3:

// return 98

// case 4:

// return 108

// case 5:

// return 118

// case 6:

// return 128

// case 7:

// return 188

// default:

// return 0

// }

//

// }

// MARK: - LoadData

// 查询连续签到

func loadSignInCount() -> Void {

WMNetTools.share.loadEventsContinueCount(name: "签到") { (continueDay) in

if continueDay == "expired"{// token 失效

self.loadSignInCount()

}else{

self.signInCountLabel.text = continueDay

let signInImageViewArray = [self.signInImageView0, self.signInImageView1, self.signInImageView2, self.signInImageView3, self.signInImageView4, self.signInImageView5, self.signInImageView6]

for i in 0..<signInImageViewArray.count{

if i < ((Int(continueDay)!%7==0) ? Int(continueDay)! : Int(continueDay)!%7){// 最多连续7天,第八天成为1

signInImageViewArray[i]?.image = UIImage.init(named: "coin\_yes")

}else{

signInImageViewArray[i]?.image = UIImage.init(named: "coin\_no")

}

}

// 计算签到奖励的金币

self.signInCoins = coinWithSignInDay(nextDay: ((Int(continueDay)!%7==0) ? Int(continueDay)! : Int(continueDay)!%7) + 1)

}

}

}

// 查询所有事件

func loadAllEventsQuery() -> Void {

let model = WMUserModel.share.readUserModel()

let dic = ["uuid":model.uuid, "query":[:]] as [String : Any]// 全部事件

WMNetTools.share.postDataRequest(url: "events/query/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadAllEventsQuery()

}else{

self.allEventsDataArray.removeAll()

let dic = getDictionaryFromJSONString(jsonString: response)

// self.allEventsDataArray = dic["events"] as! Array<Dictionary<String, Any>>

let array = dic["events"] as! Array<Dictionary<String, Any>>

for dic2 in array {

if (dic2["activate"] as! Bool) == false{

self.allEventsDataArray.append(dic2)

}else{

}

}

self.tableView.reloadData()

}

}) { (error) in

print("\(error)")

}

}

/// 激活事件

/// - Parameters:

/// - id: 事件id

/// - coin: 签到奖励的金币数量

/// - Returns: void

func loadActivateEvents(id:String, coin:Int) -> Void {

let model = WMUserModel.share.readUserModel()

var dic = ["uuid":model.uuid,"id":id] as [String : Any]

if coin > 0 {

dic.updateValue(coin, forKey: "force\_add\_coins")

}else{

// Do noting

}

WMNetTools.share.postDataRequest(url: "events/activate/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadActivateEvents(id: id, coin: coin)

}else{

// ZKProgressHUD.showMessage("金币领取成功")

// 刷新列表

self.loadAllEventsQuery()

// 刷新签到接口

self.loadSignInEventsStatus()

//获取连续签到状态

self.loadSignInCount()

}

}) { (error) in

print("\(error)")

}

}

// 查询今天是否已签到

func loadSignInEventsStatus() -> Void {

let model = WMUserModel.share.readUserModel()

let dic = ["uuid":model.uuid, "query":["name":"签到"]] as [String : Any]// 全部事件

WMNetTools.share.postDataRequest(url: "events/query/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadSignInEventsStatus()

}else{

let dic = getDictionaryFromJSONString(jsonString: response)

let array = dic["events"] as! Array<Dictionary<String, Any>>

if (array.first!["activate"] as! Bool) == true{// 已签到

self.signInButton.setTitle("已签到", for: .normal)

self.signInButton.backgroundColor = UIColor.lightGray

self.signInButton.isUserInteractionEnabled = false

}else{// 未签到

self.signInButton.setTitle("签到", for: .normal)

self.signInButton.backgroundColor = UIColor.hexString("#198AFF", a: 1.0)

self.signInButton.isUserInteractionEnabled = true

}

self.signInID = (array.first!["id"] as! String)

}

}) { (error) in

print("\(error)")

}

}

func pushLoginVC() -> Void {

let vc = WMLoginViewController()

vc.siginBlock = {

// 登陆成功回调

//获取连续签到状态

self.loadSignInCount()

// 查询所有事件

self.loadAllEventsQuery()

// 查询今天是否已签到

self.loadSignInEventsStatus()

}

self.present(vc, animated: true) {

}

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

import UIKit

class WMCoinDetailTableViewCell: UITableViewCell {

@IBOutlet weak var timeLabel: UILabel!

@IBOutlet weak var coinsLabel: UILabel!

@IBOutlet weak var eventsLabel: UILabel!

var dic : Dictionary<String, Any> = [:] {

didSet{

// 取出时间戳，转成时分秒

//指定一个Region

let rome = Region(calendar: Calendars.gregorian, zone: Zones.asiaShanghai,

locale: Locales.chinese)

// 从时间间隔创建

let nowDate = DateInRegion(seconds: Double((dic["date"] as! Int)/1000), region: rome)

// 单数前面加0

var hour = String(nowDate.dateComponents.hour!)

if hour.count < 2 {

hour.insert("0", at: hour.startIndex)

}

var minute = String(nowDate.dateComponents.minute!)

if minute.count < 2 {

minute.insert("0", at: minute.startIndex)

}

var second = String(nowDate.dateComponents.second!)

if second.count < 2 {

second.insert("0", at: second.startIndex)

}

self.timeLabel.text = "\(hour):\(minute):\(second)"

// 事件名称

var string = dic["type"] as? String

switch string {

case "exchange\_for\_coins":

string = "步数兑换"

case "exchange\_for\_payment":

string = "提现"

default:

break

}

self.eventsLabel.text = string

// 金币

var str = String(dic["coins"] as! Int)

if str.contains("-") {

}else{

str = "+\(str)"// 增加前面多个+号

}

self.coinsLabel.text = "\(str)金币"

}

}

override func awakeFromNib() {

super.awakeFromNib()

// Initialization code

}

override func setSelected(\_ selected: Bool, animated: Bool) {

super.setSelected(selected, animated: animated)

// Configure the view for the selected state

}

}

import UIKit

class WMCashCollectionViewCell: UICollectionViewCell {

@IBOutlet weak var RMBLabel: UILabel!

@IBOutlet weak var coinLabel: UILabel!

@IBOutlet weak var selectedImageView: UIImageView!

lazy var cgView : UIView = {

// 绘制渐变

let view = CGView(frame: CGRect(x: 0, y: 0, width: (SCREEN\_WIDTH-32-32-20)/3, height: 56))

let compoents:[CGFloat] = [251/255, 251/255, 251/255, 1,

245/255, 245/255, 245/255, 1]

view.colors = compoents

view.startPoint = CGPoint(x: self.bounds.minX, y: self.bounds.maxY)

view.endPoint = CGPoint(x: self.bounds.maxX, y: self.bounds.maxY)

return view

}()

var dic : Dictionary<String, Any>?{

didSet{

let select = dic!["select"] as! Bool

self.contentView.insertSubview(cgView, at: 0)

if select {

self.layer.borderWidth = 2

self.layer.borderColor = UIColor.hexString("#198AFF", a: 1.0).cgColor

self.selectedImageView.isHidden = false

self.cgView.isHidden = true

}else{

self.layer.borderWidth = 0

self.layer.borderColor = UIColor.clear.cgColor

self.selectedImageView.isHidden = true

self.cgView.isHidden = false

}

self.RMBLabel.text = (dic!["RMB"] as! String)

self.coinLabel.text = "\((dic!["coin"] as! String))金币"

}

}

override func awakeFromNib() {

super.awakeFromNib()

// Initialization code

}

}

import UIKit

import McPicker

class WMMeViewController: WMBaseViewController {

@IBOutlet weak var gradientView: UIView!

@IBOutlet weak var nickNameLabel: UILabel!

@IBOutlet weak var avaImageView: UIImageView!

@IBOutlet weak var stepCountLabel: UILabel!

@IBOutlet weak var myReferCodeLabel: UILabel!

@IBOutlet weak var unusedCoinLabel: UILabel!

@IBOutlet weak var canCashLabel: UILabel!

@IBOutlet weak var view3: UIView!

@IBOutlet weak var view1: UIView!

var stepArray : Array<String> = []

let targetStepsArray = ["10000","20000","30000","40000","50000"]

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

self.navigationController?.setNavigationBarHidden(true, animated: animated)

self.avaImageViewState()

// 判断登陆状态

if WMUserModel.share.isLogin() == true {

self.setUI()

}else{

}

}

override func viewWillDisappear(\_ animated: Bool) {

super.viewWillDisappear(animated)

self.navigationController?.setNavigationBarHidden(false, animated: animated)

}

override var preferredStatusBarStyle: UIStatusBarStyle{

return .lightContent

}

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

// 目标步数

for i in 3000...50000 {

if i%1000 == 0{

stepArray.append("\(i)")

}else{

}

}

self.title = "我的"

// 添加手势点击事件

let avaImageViewTap = UITapGestureRecognizer.init(target: self, action: #selector(self.avaImageViewAction(\_:)))

self.avaImageView.addGestureRecognizer(avaImageViewTap)

// 绘制渐变

self.gradientView.addSubview(self.drawGradientView())

// 设置目标步数

let step = UserDefaults.standard.value(forKey: "TargetStepCount") as! Int

self.stepCountLabel.text = "\(step)步"

// 分割线

let lineView = UIView.init()

lineView.backgroundColor = UIColor.hexString("#000000", a: 0.1)

lineView.frame = CGRect(x: (SCREEN\_WIDTH - 32)/2, y: 14, width: 0.33, height: 120 - 28)

self.view1.addSubview(lineView)

// 分割线

let lineView3 = UIView.init()

lineView3.backgroundColor = UIColor.hexString("#000000", a: 0.1)

lineView3.frame = CGRect(x: 52, y: 56, width: SCREEN\_WIDTH - 32 - 52 - 14, height: 0.33)

self.view3.addSubview(lineView3)

// 复制邀请码点击事件

let referCodeLabelTap = UITapGestureRecognizer.init(target: self, action: #selector(self.myReferCodeLabelAction(\_:)))

self.myReferCodeLabel.addGestureRecognizer(referCodeLabelTap)

}

func drawGradientView() -> UIView {

let cgView = CGView(frame: CGRect(x: 0, y: 0, width: self.gradientView.width, height: self.gradientView.height))

let compoents:[CGFloat] = [25/255, 138/255, 255/255, 1,

25/255, 138/255, 255/255, 0]

cgView.colors = compoents

cgView.startPoint = CGPoint(x: self.gradientView.bounds.minX, y: self.gradientView.bounds.minY)

cgView.endPoint = CGPoint(x: self.gradientView.bounds.minX, y: self.gradientView.bounds.maxY)

return cgView

}

// MARK: LoadData

func loadUserInfoData() -> Void {

let userModel = WMUserModel.share.readUserModel()

let dic = ["uuid":userModel.uuid]

WMNetTools.share.postDataRequest(url: "user/info/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadUserInfoData()

}else{

let dic = getDictionaryFromJSONString(jsonString: response)

self.unusedCoinLabel.text = String.init(format: "%@", dic["unused\_coins"] as! NSNumber)

let cashStr = NSString.init(format: "%f", (dic["unused\_coins"] as! NSNumber).doubleValue/10000) as String

let location : Int = cashStr.distance(from: cashStr.startIndex, to: cashStr.range(of: ".")!.upperBound)

self.canCashLabel.text = String(cashStr.prefix(location + 2))

let model = WMUserModel.share.readUserModel()

model.wx\_bind = "\(dic["wx\_bind"] ?? "")"

WMUserModel.share.updateUserModel(model: model)

}

}) { (error) in

print("\(error)")

}

}

// MARK: Action

@IBAction func inviteCodeButtonAction(\_ sender: UIButton) {

print("填写邀请码")

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

// 跳转填写邀请码界面

let vc = WMInputInviteCodeViewController()

self.navigationController?.pushViewController(vc, animated: true)

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

@IBAction func coinButtonAction(\_ sender: UIButton) {

print("金币")

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

// 跳转金币界面

let vc = WMCoinViewController()

self.navigationController?.pushViewController(vc, animated: true)

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

@IBAction func cashButtonAction(\_ sender: Any) {

print("提现")

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

let model = WMUserModel.share.readUserModel()

if (model.wx\_bind == "1") {// 已绑定微信

// 跳转提现

let vc = WMCashViewController()

vc.coins = self.unusedCoinLabel.text!

self.navigationController?.pushViewController(vc, animated: true)

}else{// 未绑定微信

// 提示绑定微信

let alert = UIAlertController.init(title: "提现到微信钱包", message: "需要绑定微信才能提现哦！", preferredStyle: .alert)

let alertAction = UIAlertAction.init(title: "取消", style: .cancel) { (action) in

print("取消")

}

alertAction.setValue(UIColor.hexString("#8E8E93", a: 1.0), forKey: "titleTextColor")

alert.addAction(alertAction)

alert.addAction(UIAlertAction.init(title: "授权", style: .default, handler: { (action) in

print("授权")

NotificationCenter.default.removeObserver(self, name: NSNotification.Name(rawValue: "WXAuthNotifyKey"), object: nil)

// 绑定微信

AppDelegate.sendAuthRequest()

//监听接收消息，获取传过来的code，触发微信登陆方法

NotificationCenter.default.addObserver(self, selector: #selector(self.WXBind(noti:)), name: NSNotification.Name(rawValue: "WXAuthNotifyKey"), object: nil)

}))

self.present(alert, animated: true) {

}

}

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

@IBAction func inviteButtonAction(\_ sender: UIButton) {

print("邀请好友")

// 判断登陆状态

if WMUserModel.share.isLogin() == true{

// 跳转邀请好友h5页面

ZKProgressHUD.showMessage("邀请好友H5")

}else{

// 跳转登陆界面

self.pushLoginVC()

}

}

@IBAction func aimButtonAction(\_ sender: UIButton) {

print("目标")

let array = [stepArray]

let picker = McPicker.init(data: array)

picker.toolbarDoneButtonColor = UIColor.black

picker.toolbarCancelButtonColor = UIColor.hexString("#000000", a: 0.6)

let customLabel = UILabel()

customLabel.textAlignment = .center

customLabel.textColor = UIColor.hexString("#000000", a: 0.9)

customLabel.font = UIFont.systemFont(ofSize: 20)

picker.label = customLabel // Set your custom label

let fixedSpace = McPickerBarButtonItem.fixedSpace(width: 20.0)

let flexibleSpace = McPickerBarButtonItem.flexibleSpace()

let doneItem = McPickerBarButtonItem.done(mcPicker: picker, title: "确定", barButtonSystemItem: .done)

let cancelItem = McPickerBarButtonItem.cancel(mcPicker: picker, title: "取消", barButtonSystemItem: .cancel)

picker.setToolbarItems(items: [fixedSpace, cancelItem, flexibleSpace, doneItem, fixedSpace])

picker.show { (string) in

let number = string[0]!

self.stepCountLabel.text = "\(number)步"

// 设置目标步数

UserDefaults.standard.set(Int(number), forKey: "TargetStepCount")

UserDefaults.standard.synchronize()

}

}

@IBAction func editButtonAction(\_ sender: UIButton) {

print("设置")

let vc = WMEditViewController()

vc.outBlock = {

// 退出登录回调

self.setUI()

}

self.navigationController?.pushViewController(vc, animated: true)

}

@objc func avaImageViewAction(\_ sender: UITapGestureRecognizer) {

print("头像")

self.pushLoginVC()

}

@objc func myReferCodeLabelAction(\_ sender: UITapGestureRecognizer) {

print("复制邀请码")

// 判断登陆状态

if WMUserModel.share.isLogin() == true {

let str = self.myReferCodeLabel.text

let range: Range = str!.range(of: " ")!

let location: Int = str!.distance(from: str!.startIndex, to: range.upperBound)

let subStr = str!.suffix(str!.count - location)

UIPasteboard.general.string = String(subStr)

ZKProgressHUD.showMessage("邀请码已复制")

}else{

// Do noting

}

}

func pushLoginVC() -> Void {

let vc = WMLoginViewController()

vc.siginBlock = {

// 登陆成功回调

self.setUI()

self.avaImageViewState()

}

self.present(vc, animated: true) {

}

}

func setUI() -> Void {

// 修改用户昵称、头像

let userModel = WMUserModel.share.readUserModel()

self.avaImageView.kf.setImage(with: URL.init(string: userModel.url), placeholder: UIImage.init(named: "ava\_default"))

self.nickNameLabel.text = (userModel.nickname as NSString).length>0 ? userModel.nickname : "哄哄的网友名"

// 判断登陆状态

if WMUserModel.share.isLogin() == true {

self.myReferCodeLabel.text = "复制专属邀请码: \(userModel.my\_refer\_code)"

// 请求个人数据

self.loadUserInfoData()

}else{

self.myReferCodeLabel.text = "复制专属邀请码: \*\*\*"

}

}

func avaImageViewState() -> Void {

// 判断登陆状态

if WMUserModel.share.isLogin() == true {

self.avaImageView.isUserInteractionEnabled = false

}else{

self.avaImageView.isUserInteractionEnabled = true

}

}

deinit {

NotificationCenter.default.removeObserver(self)

}

@objc private func WXBind(noti:Notification){

let dic = noti.userInfo

let code = dic!["code"]

let idfa = ASIdentifierManager.shared().advertisingIdentifier.uuidString

let iosVersion = UIDevice.current.systemVersion

let userModel = WMUserModel.share.readUserModel()

let dic2 = ["code":code,"uuid":userModel.uuid,"idfa":idfa,"iosversion":iosVersion]

// 绑定微信

WMNetTools.share.postDataRequest(url: "wx/bind/", params: dic2 as [String : Any], isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.WXBind(noti: noti)

}else{

// model储存用户信息

let dic = getDictionaryFromJSONString(jsonString: response)

let userModel = dic.kj.model(WMUserModel.self)

WMUserModel.share.updateUserModel(model: userModel!)

// 设置头像、昵称

self.setUI()

ZKProgressHUD.showMessage("绑定微信成功")

}

}) { (error) in

print("error:\(error)")

}

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

import UIKit

// 登陆成功回调

typealias signOutBlock = () -> Void

class WMEditViewController: WMBaseViewController,UITableViewDelegate,UITableViewDataSource {

@IBOutlet weak var tableView: UITableView!

var dataArray : Array<Array<String>> = []

var outBlock : signOutBlock?

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

self.title = "设置"

// 判断登陆状态

if WMUserModel.share.isLogin() == true {

dataArray = [["分享","评分"],["服务协议","隐私政策","版本号"],["退出登录"]]

}else{

dataArray = [["分享","评分"],["服务协议","隐私政策","版本号"]]

}

}

// MARK: - UITableViewDelegate,UITableViewDataSource

func numberOfSections(in tableView: UITableView) -> Int{

return dataArray.count

}

func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int{

return dataArray[section].count

}

func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat{

return 44

}

func tableView(\_ tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat{

return 9.9

}

func tableView(\_ tableView: UITableView, heightForFooterInSection section: Int) -> CGFloat{

return 0.01

}

func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell{

let idStr = "WMEditViewControllerCell"

var cell = tableView.dequeueReusableCell(withIdentifier: idStr)

if cell == nil {

if indexPath.section == dataArray.count - 1 {

cell = UITableViewCell.init(style:UITableViewCell.CellStyle.default , reuseIdentifier: idStr)

}else{

cell = UITableViewCell.init(style:UITableViewCell.CellStyle.value1 , reuseIdentifier: idStr)

}

}

cell?.textLabel?.text = dataArray[indexPath.section][indexPath.row]

cell?.selectionStyle = .none

switch cell?.textLabel?.text {

case "版本号":

cell?.accessoryType = .none

cell?.detailTextLabel?.text = "1.0"

case "退出登录":

cell?.accessoryType = .none

cell?.textLabel?.textAlignment = .center

cell?.textLabel?.textColor = UIColor.hexString("#000000", a: 0.7)

default:

cell?.accessoryType = .disclosureIndicator

}

// 最后一行不用加分割线

if (indexPath.row+1) == tableView.numberOfRows(inSection: indexPath.section) {

// do noting

}else{

// 分割线

let lineView = UIView.init(frame: CGRect(x: 15, y: 44 - 0.33, width: SCREEN\_WIDTH - 15, height: 0.33))

lineView.backgroundColor = UIColor.hexString("#000000", a: 0.1)

cell?.contentView.addSubview(lineView)

}

return cell!

}

func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath){

switch dataArray[indexPath.section][indexPath.row] {

case "分享":

// 调用系统分享

// 分享内容

let shareTitle = "走路有钱"

let shareImage = UIImage.init(named: "login\_logo")

let shareUrl = URL.init(string: "https://apps.apple.com/us/app/id\(AppStoreAPPID)")

let activityItemsArray : Array = [shareTitle, shareImage!, shareUrl!] as [Any]

// 调用分享

let activityVC = UIActivityViewController.init(activityItems: activityItemsArray, applicationActivities: nil)

activityVC.isModalInPopover = true

// iOS8.0 之后用此方法回调

activityVC.completionWithItemsHandler = { activityType, completed, items, error in

if completed {

print("分享成功")

}else{

print("分享失败")

}

}

self.present(activityVC, animated: true) {

}

case "评分":

let urlStr = "itms-apps://itunes.apple.com/WebObjects/MZStore.woa/wa/viewContentsUserReviews?type=Purple+Software&id=\(AppStoreAPPID)&pageNumber=0&sortOrdering=2&mt=8"

UIApplication.shared.open(URL.init(string: urlStr)!) { (success) in

}

case "服务协议":

let vc = LBWebViewController.init()

vc.url = WebUrl\_Terms

vc.title = "服务条款"

let nav = UINavigationController.init(rootViewController: vc)

self.present(nav, animated: true) {

}

case "隐私政策":

let vc = LBWebViewController.init()

vc.url = WebUrl\_Privacy

vc.title = "隐私政策"

let nav = UINavigationController.init(rootViewController: vc)

self.present(nav, animated: true) {

}

case "退出登录":

self.loadLogout()

default:

print("...")

}

}

func tableView(\_ tableView: UITableView, willDisplayHeaderView view: UIView, forSection section: Int) {

if view.isKind(of: UITableViewHeaderFooterView.self) {

view.tintColor = UIColor.clear

}

}

func loadLogout() -> Void {

let userModel = WMUserModel.share.readUserModel()

let dic = ["uuid":userModel.uuid]

WMNetTools.share.postDataRequest(url: "wx/logout/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadLogout()

}else{

// 退出登陆成功，清除本地userModel数据

WMUserModel.share.clearUserModel()

self.navigationController?.popViewController(animated: true)

// 退出登陆成功回调

if self.outBlock != nil{

self.outBlock?()

}

}

}) { (error) in

print("\(error)")

}

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

import UIKit

class WMInputInviteCodeViewController: WMBaseViewController {

@IBOutlet weak var textField: UITextField!

override func viewDidLoad() {

super.viewDidLoad()

self.view.backgroundColor = UIColor.white

// Do any additional setup after loading the view.

self.title = "填写邀请码"

}

// MARK: - Action

@IBAction func sureButtonAction(\_ sender: Any) {

print("提交验证")

if self.textField.text!.count <= 0 {

ZKProgressHUD.showMessage("验证码不能为空")

return

}

let model = WMUserModel.share.readUserModel()

let dic = ["uuid":model.uuid, "refer\_code":self.textField.text!] as [String : Any]

// 提交邀请码

WMNetTools.share.postDataRequest(url: "events/refer/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.sureButtonAction(sender)

}else{

}

}) { (error) in

print("\(error)")

}

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

import UIKit

import SwiftDate

class WMCoinViewController: WMBaseViewController,UITableViewDelegate,UITableViewDataSource {

@IBOutlet weak var tableView: UITableView!

@IBOutlet var currentCoinCell: UITableViewCell!

@IBOutlet weak var currentCoinsLabel: UILabel!

@IBOutlet weak var todayCoinsLabel: UILabel!

@IBOutlet weak var allIncomeLabel: UILabel!

@IBOutlet weak var canCashLabel: UILabel!

@IBOutlet weak var coinRMBRatioLabel: UILabel!

var dataArray : Array<Dictionary<String, Any>> = []// 缓存请求下来的数据，上拉加载更多时使用

var coinDetailDataArray : Array<Dictionary<String, Any>> = []

// 顶部刷新

let header = MJRefreshNormalHeader()

// 底部刷新

let footer = MJRefreshAutoNormalFooter()

var coinDetailLastDate = "" // 金币明细上拉加载传上一页最后一个的时间戳

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

self.loadUserInfoData()

self.loadTodayCoinsData()

}

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

self.title = "金币"

// 注册cell

self.tableView.register(UINib.init(nibName: "WMCoinDetailTableViewCell", bundle: nil), forCellReuseIdentifier: "WMCoinDetailTableViewCell")

// 下拉刷新

header.setRefreshingTarget(self, refreshingAction: #selector(WMCoinViewController.headerRefresh))

// 上拉刷新

footer.setRefreshingTarget(self, refreshingAction: #selector(WMCoinViewController.footerRefersh))

// footer.setTitle("", for: .idle)//设置闲置状态下不显示“点击或上拉加载更多”

self.tableView.mj\_header = header

self.tableView.mj\_footer = footer

header.beginRefreshing()

}

// MARK: - UITableViewDelegate,UITableViewDataSource

func numberOfSections(in tableView: UITableView) -> Int{

return self.coinDetailDataArray.count + 1

}

func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int{

if section == 0 {

return 1

}else{

let dic = self.coinDetailDataArray[section-1] as Dictionary<String, Any>

let array = dic.values.first as? Array<Any>

return array!.count

}

}

func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat{

if indexPath.section == 0 {

return 190

}else{

return 32

}

}

func tableView(\_ tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat{

if section == 0 {

return 0.01

}else{

return 32

}

}

func tableView(\_ tableView: UITableView, heightForFooterInSection section: Int) -> CGFloat{

return 0.01

}

func tableView(\_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {

if section == 0 {

return ""

}else{

let dic = self.coinDetailDataArray[section-1] as Dictionary<String, Any>

return dic.keys.first

}

}

func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell{

if indexPath.section == 0{

return self.currentCoinCell

}else{

let cell = tableView.dequeueReusableCell(withIdentifier: "WMCoinDetailTableViewCell", for: indexPath) as! WMCoinDetailTableViewCell

let dic = self.coinDetailDataArray[indexPath.section-1] as Dictionary<String, Any>

let array = dic.values.first as? Array<Any>

cell.dic = array![indexPath.row] as! Dictionary<String, Any>

return cell

}

}

func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath){

print(indexPath)

}

func tableView(\_ tableView: UITableView, willDisplayHeaderView view: UIView, forSection section: Int) {

if view.isKind(of: UITableViewHeaderFooterView.self) {

view.tintColor = UIColor.clear

let header = view as! UITableViewHeaderFooterView

header.textLabel?.textAlignment = .left

header.textLabel?.font = UIFont.systemFont(ofSize: 18)

header.textLabel?.textColor = UIColor.hexString("#000000", a: 0.7)

}

}

// 禁止header悬浮

func scrollViewDidScroll(\_ scrollView: UIScrollView) {

let sectionHeaderHeight: CGFloat = 32

if scrollView.contentOffset.y <= sectionHeaderHeight && scrollView.contentOffset.y >= 0 {

scrollView.contentInset = UIEdgeInsets(top: -scrollView.contentOffset.y, left: 0, bottom: 0, right: 0)

} else if scrollView.contentOffset.y >= sectionHeaderHeight {

scrollView.contentInset = UIEdgeInsets(top: -sectionHeaderHeight, left: 0, bottom: 0, right: 0)

}

}

// MARK: - Action

@IBAction func cashButtonAction(\_ sender: UIButton) {

print("提现")

// 跳转提现

let vc = WMCashViewController()

vc.coins = self.currentCoinsLabel.text!

self.navigationController?.pushViewController(vc, animated: true)

}

// MARK: LoadData

func loadUserInfoData() -> Void {

let userModel = WMUserModel.share.readUserModel()

let dic = ["uuid":userModel.uuid]

WMNetTools.share.postDataRequest(url: "user/info/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadUserInfoData()

}else{

let dic = getDictionaryFromJSONString(jsonString: response)

self.currentCoinsLabel.text = String.init(format: "%@", dic["unused\_coins"] as! NSNumber)

let cashStr = NSString.init(format: "%f", (dic["unused\_coins"] as! NSNumber).doubleValue/10000) as String

let location : Int = cashStr.distance(from: cashStr.startIndex, to: cashStr.range(of: ".")!.upperBound)

self.canCashLabel.text = String(cashStr.prefix(location + 2))

self.allIncomeLabel.text = String.init(format: "%@", dic["total\_payments"] as! NSNumber)

self.coinRMBRatioLabel.text = "当前金币 (\((dic["coin\_rmb\_ratio"] as! NSNumber).intValue \* 100)金币=1元)" // coin\_rmb\_ratio 单位:分

}

}) { (error) in

print("\(error)")

}

}

// 今日金币

func loadTodayCoinsData() -> Void {

let userModel = WMUserModel.share.readUserModel()

let dic = ["uuid":userModel.uuid]

WMNetTools.share.postDataRequest(url: "coins/today/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadTodayCoinsData()

}else{

let dic = getDictionaryFromJSONString(jsonString: response)

self.todayCoinsLabel.text = (dic["coins"] as! String)

}

}) { (error) in

print("\(error)")

}

}

// 金币明细

func loadTransactionsQueryData(date:String) -> Void {

let userModel = WMUserModel.share.readUserModel()

var dic = ["uuid":userModel.uuid] as Dictionary<String, Any>

if date.count > 0{

dic.updateValue(Int64(date) ?? 0, forKey: "date")

}else{

}

WMNetTools.share.postDataRequest(url: "transactions/query/", params: dic, isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.loadTransactionsQueryData(date: date)

}else{

self.tableView.mj\_header?.endRefreshing()

self.tableView.mj\_footer?.endRefreshing()

let dic = getDictionaryFromJSONString(jsonString: response)

guard let array = dic["results"] as? Array<Dictionary<String, Any>> else{

return

}

/\* 整理后台返回的数据，根据年月日分类

[

[date: [[:],[:],[:]]],

[date: [[:],[:],[:]]]

]

\*/

for dic:Dictionary<String, Any> in array {

// 记录最后一个时间戳

self.coinDetailLastDate = String(dic["date"] as! Int)

// 取出时间戳，转成年月日

//指定一个Region

let rome = Region(calendar: Calendars.gregorian, zone: Zones.asiaShanghai,

locale: Locales.chinese)

// 从时间间隔创建

let nowDate = DateInRegion(seconds: Double((dic["date"] as! Int)/1000), region: rome)

let dateStr = "\(nowDate.dateComponents.year!)-\(nowDate.dateComponents.month!)-\(nowDate.dateComponents.day!)"

if self.dataArray.count > 0 {

// 找出所有的dic.key

var allKeysArray = Array<String>.init()

for dic3:Dictionary<String, Any> in self.dataArray {

allKeysArray.append(dic3.keys.first!)

}

// 判断所有的dic.key是否包含dateStr

if allKeysArray.contains(dateStr) {

// 如果包含，则找出key下标，添加进key对应的数组

let index = allKeysArray.firstIndex(of: dateStr)

let dic5 = self.dataArray[index!]

var array2 = dic5["\(dateStr)"] as? Array<Any>

array2?.append(dic)

self.dataArray[index!] = [dateStr:array2!]

}else{

// 直接添加进去

let dic4 = [dateStr:([dic] as Array<Any>)] as Dictionary

self.dataArray.append(dic4)

}

}else{

// 直接添加进去

let dic2 = [dateStr:([dic] as Array<Any>)] as Dictionary

self.dataArray.append(dic2)

}

}

self.coinDetailDataArray = self.dataArray

self.tableView.reloadData()

// 不超过10条，取消下拉刷新

if array.count < 10 {

self.tableView.mj\_footer?.isHidden = true

}else{

self.tableView.mj\_footer?.isHidden = false

}

// self.tableView.mj\_header?.isHidden = true

}

}) { (error) in

print("\(error)")

}

}

// 顶部刷新

@objc func headerRefresh() -> Void {

print("上拉刷新")

self.coinDetailDataArray.removeAll()

self.dataArray.removeAll()

self.loadTransactionsQueryData(date: "")

}

// 底部刷新

@objc func footerRefersh() -> Void {

print("下拉刷新")

// 传最后一个的时间戳

self.loadTransactionsQueryData(date: self.coinDetailLastDate)

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

import UIKit

typealias cashBlock = (\_ currentCoins:String) -> Void

class WMCashViewController: WMBaseViewController,UICollectionViewDelegate,UICollectionViewDataSource {

var cashCoinBlock : cashBlock?

@IBOutlet weak var collectionView: UICollectionView!

@IBOutlet weak var coinsLabel: UILabel!

@IBOutlet weak var avaImageView: UIImageView!

@IBOutlet weak var nikeNameLabel: UILabel!

var coins : String = ""

var dataArray : Array<Dictionary<String, Any>> = [["RMB":"2元","coin":"20000","select":true],

["RMB":"10元","coin":"100000","select":false],

["RMB":"30元","coin":"300000","select":false],

["RMB":"50元","coin":"500000","select":false],

["RMB":"100元","coin":"1000000","select":false]]

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

self.coinsLabel.text = coins

// 修改用户昵称、头像

let userModel = WMUserModel.share.readUserModel()

self.avaImageView.kf.setImage(with: URL.init(string: userModel.url), placeholder: UIImage.init(named: "ava\_default"))

self.nikeNameLabel.text = (userModel.nickname as NSString).length>0 ? userModel.nickname : "哄哄的网友名"

self.title = "提现"

let layout = UICollectionViewFlowLayout.init()

// 行间距、列间距

layout.minimumLineSpacing = 15

layout.minimumInteritemSpacing = 10

// item大小

layout.itemSize = CGSize(width: (SCREEN\_WIDTH-32-32-20)/3, height: 56)

// 内边距

layout.sectionInset = UIEdgeInsets(top: 0, left: 0, bottom: 0, right: 0)

self.collectionView.collectionViewLayout = layout

// 注册cell

self.collectionView.register(UINib.init(nibName: "WMCashCollectionViewCell", bundle: nil), forCellWithReuseIdentifier: "WMCashCollectionViewCell")

}

// MARK: - UICollectionViewDelegate,UICollectionViewDataSource

func numberOfSections(in collectionView: UICollectionView) -> Int{

return 1

}

func collectionView(\_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int{

return self.dataArray.count

}

func collectionView(\_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell{

let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "WMCashCollectionViewCell", for: indexPath) as! WMCashCollectionViewCell

cell.dic = self.dataArray[indexPath.row]

return cell

}

func collectionView(\_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath){

print(indexPath)

self.dataArray = self.dataArray.compactMap { dic -> Dictionary<String, Any> in

var dic2 = dic

dic2["select"] = false

return dic2

}

self.dataArray[indexPath.row]["select"] = true

self.collectionView.reloadData()

}

// MARK: - Action

@IBAction func inviteFriendButtonAction(\_ sender: UIButton) {

print("邀请好友")

ZKProgressHUD.showMessage("邀请好友H5")

}

@IBAction func cashButtonAction(\_ sender: UIButton) {

print("立即提现")

var coins = 0

for dic in self.dataArray {

if (dic["select"] as! Bool) == true {

coins = Int(dic["coin"] as! String)!

}

}

if Int(self.coins)! < coins {

ZKProgressHUD.showMessage("当前金币不足")

return

}

let userModel = WMUserModel.share.readUserModel()

let dic2 = ["uuid":userModel.uuid,"coins":coins] as [String : Any]

WMNetTools.share.postDataRequest(url: "payments/apply/", params: dic2 as [String : Any], isToken: true, success: { (response) in

if response == "expired"{// token 失效

self.cashButtonAction(sender)

}else{

let dic = getDictionaryFromJSONString(jsonString: response)

self.coinsLabel.text = dic["unused\_coins"] as? String

ZKProgressHUD.showMessage("提现成功")

}

}) { (error) in

print("error:\(error)")

}

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

import UIKit

class WMBaseViewController: UIViewController {

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

self.view.backgroundColor = UIColor.hexString("#f8f8f8", a: 1.0)

}

override var preferredStatusBarStyle: UIStatusBarStyle{

if #available(iOS 13.0, \*) {

return .darkContent

} else {

// Fallback on earlier versions

return .default

}

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

import UIKit

class WMBaseNavigationController: UINavigationController {

override func viewDidLoad() {

super.viewDidLoad()

// Do any additional setup after loading the view.

//通过设置shadowImage为 navigationBar设置分割线颜色

// self.navigationBar.setBackgroundImage(UIImage.init(), for: .any, barMetrics: .default)

// self.navigationBar.shadowImage = UIImage.init()

//

// let line = UIView.init(frame: CGRect(x: 0, y: self.navigationBar.height - 0.33, width: SCREEN\_WIDTH, height: 0.33))

// line.backgroundColor = UIColor.hexString("#000000", a: 0.1)

// self.navigationBar.addSubview(line)

navigationBar.setBackgroundImage(UIImage.Create(size: CGSize(width: 1, height: 1), color: .white), for: .default)

navigationBar.shadowImage = UIImage()

let line = UIView.init(frame: CGRect(x: 0, y: self.navigationBar.height - 0.33, width: SCREEN\_WIDTH, height: 0.33))

line.backgroundColor = UIColor.hexString("#000000", a: 0.1)

navigationBar.addSubview(line)

}

override func pushViewController(\_ viewController: UIViewController, animated: Bool) {

if self.children.count > 0 {

viewController.hidesBottomBarWhenPushed = true

}

super.pushViewController(viewController, animated: animated)

}

override var preferredStatusBarStyle: UIStatusBarStyle{

let topVC = self.topViewController

return topVC!.preferredStatusBarStyle

}

/\*

// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

// Get the new view controller using segue.destination.

// Pass the selected object to the new view controller.

}

\*/

}

extension UIImage{

class func Create(size:CGSize,color : UIColor,alpha:CGFloat? = 1) -> UIImage{

UIGraphicsBeginImageContext(size)

color.setFill()

let bounds = CGRect.init(x: 0, y: 0, width: size.width, height: size.height)

UIRectFill(bounds)

UIImage().draw(in: bounds, blendMode: CGBlendMode.destinationIn, alpha: alpha ?? 1)

let tintedImage = UIGraphicsGetImageFromCurrentImageContext()

UIGraphicsEndImageContext()

return tintedImage!

}

}