

第76章：NSUserDefaults

第76.1节：设置值

要在NSUserDefaults中设置值，可以使用以下函数：

Swift 3之前版本

```
setBool(_:forKey:)  
setFloat(_:forKey:)  
setInteger(_:forKey:)  
setObject(_:forKey:)  
setDouble(_:forKey:)  
setURL(_:forKey:)
```

Swift 3

在Swift 3中，函数名称改为set，后面不再跟类型。

```
set(_:forKey:)
```

Objective-C

```
- (void)setBool:(BOOL)value forKey:(nonnull NSString *)defaultName;  
- (void)setFloat:(float)value forKey:(nonnull NSString *)defaultName;  
- (void)setInteger:(NSInteger)value forKey:(nonnull NSString *)defaultName;  
- (void)setObject:(nullable id)value forKey:(nonnull NSString *)defaultName;  
- (void)setDouble:(double)value forKey:(nonnull NSString *)defaultName;  
- (void)setURL:(nullable NSURL *)value forKey:(nonnull NSString *)defaultName;
```

示例用法如下：

Swift 3之前版本

```
NSUserDefaults.standardUserDefaults.setObject("Netherlands", forKey: "HomeCountry")
```

Swift 3

```
UserDefault.standard.set("Netherlands", forKey: "HomeCountry")
```

Objective-C

```
[[NSUserDefaults standardUserDefaults] setObject:@"Netherlands" forKey:@"HomeCountry"];
```

自定义对象

要将自定义对象保存到 `NSUserDefaults`，您需要让您的 CustomClass 遵循 `NSCoding` 协议。您需要实现以下方法：Swift

```
public func encodeWithCoder(aCoder: NSCoder) {  
    aCoder.encodeObject(name, forKey: "name")  
    aCoder.encodeObject(unitId, forKey: "unitId")  
}  
  
required public init(coder aDecoder: NSCoder) {  
    super.init()  
    name = aDecoder.decodeObjectForKey("name") as? String  
    unitId = aDecoder.decodeIntegerForKey("unitId") as? NSInteger  
}
```

Chapter 76: UserDefaults

Section 76.1: Setting values

To set a value in `NSUserDefaults`, you can use the following functions:

Swift < 3

```
setBool(_:forKey:)  
setFloat(_:forKey:)  
setInteger(_:forKey:)  
setObject(_:forKey:)  
setDouble(_:forKey:)  
setURL(_:forKey:)
```

Swift 3

In Swift 3 the names of function is changed to `set` insted of `set` folloed by the type.

```
set(_:forKey:)
```

Objective-C

```
- (void)setBool:(BOOL)value forKey:(nonnull NSString *)defaultName;  
- (void)setFloat:(float)value forKey:(nonnull NSString *)defaultName;  
- (void)setInteger:(NSInteger)value forKey:(nonnull NSString *)defaultName;  
- (void)setObject:(nullable id)value forKey:(nonnull NSString *)defaultName;  
- (void)setDouble:(double)value forKey:(nonnull NSString *)defaultName;  
- (void)setURL:(nullable NSURL *)value forKey:(nonnull NSString *)defaultName;
```

Example usage would be:

Swift < 3

```
NSUserDefaults.standardUserDefaults.setObject("Netherlands", forKey: "HomeCountry")
```

Swift 3

```
UserDefault.standard.set("Netherlands", forKey: "HomeCountry")
```

Objective-C

```
[[NSUserDefaults standardUserDefaults] setObject:@"Netherlands" forKey:@"HomeCountry"];
```

Custom objects

To save custom objects into the `NSUserDefaults` you need to make your CustomClass confirm to protocol of `NSCoding`. You need to implement the following methods: **Swift**

```
public func encodeWithCoder(aCoder: NSCoder) {  
    aCoder.encodeObject(name, forKey: "name")  
    aCoder.encodeObject(unitId, forKey: "unitId")  
}  
  
required public init(coder aDecoder: NSCoder) {  
    super.init()  
    name = aDecoder.decodeObjectForKey("name") as? String  
    unitId = aDecoder.decodeIntegerForKey("unitId") as? NSInteger  
}
```

Objective-C

```
- (id)initWithCoder:(NSCoder *)coder {
    self = [super init];
    if (self) {
        name = [coder decodeObjectForKey:@"name"];
        unitId = [coder decodeIntegerForKey:@"unitId"];
    }
    return self;
}

- (void)encodeWithCoder:(NSCoder *)coder {
    [coder encodeObject:name forKey:@"name"];
    [coder encodeInteger:unitId forKey:@"unitId"];
}
```

Objective-C

```
- (id)initWithCoder:(NSCoder *)coder {
    self = [super init];
    if (self) {
        name = [coder decodeObjectForKey:@"name"];
        unitId = [coder decodeIntegerForKey:@"unitId"];
    }
    return self;
}

- (void)encodeWithCoder:(NSCoder *)coder {
    [coder encodeObject:name forKey:@"name"];
    [coder encodeInteger:unitId forKey:@"unitId"];
}
```

第76.2节：Swift 3中UserDefaults的使用

每个应用程序都需要在UserDefaults中存储用户会话或用户相关的详细信息。因此，我们将整个逻辑封装在一个类中，以更好地管理UserDefaults。

Swift 3

```
import Foundation

public struct Session {

    fileprivate static let defaults = UserDefaults.standard

    enum userValues: String {
        case auth_token
        case email
        case fname
        case mobile
        case title
        case userId
        case userType
        case OTP
        case isApproved
    }

    //MARK: - 获取用户详细信息
    static func getUserSessionDetails() -> [String: AnyObject]? {
        let dictionary = defaults.object(forKey: "LoginSession") as? [String: AnyObject]
        return dictionary
    }

    //MARK: - 保存设备令牌
    static func saveDeviceToken(_ token: String) {
        guard (gettingDeviceToken() ?? "").isEmpty else {
            return
        }
        defaults.removeObject(forKey: "deviceToken")
        defaults.set(token, forKey: "deviceToken")
        defaults.synchronize()
    }

    //MARK: - 获取令牌
    static func gettingDeviceToken() -> String? {
        let token = defaults.object(forKey: "deviceToken") as? String
    }
}
```

Section 76.2: UserDefaults uses in Swift 3

Every application needed to store User Session or User related details inside application in UserDefaults. So we made whole logic inside a Class for managing UserDefaults better way.

Swift 3

```
import Foundation

public struct Session {

    fileprivate static let defaults = UserDefaults.standard

    enum userValues: String {
        case auth_token
        case email
        case fname
        case mobile
        case title
        case userId
        case userType
        case OTP
        case isApproved
    }

    //MARK: - Getting here User Details
    static func getUserSessionDetails() -> [String: AnyObject]? {
        let dictionary = defaults.object(forKey: "LoginSession") as? [String: AnyObject]
        return dictionary
    }

    //MARK: - Saving Device Token
    static func saveDeviceToken(_ token: String) {
        guard (gettingDeviceToken() ?? "").isEmpty else {
            return
        }
        defaults.removeObject(forKey: "deviceToken")
        defaults.set(token, forKey: "deviceToken")
        defaults.synchronize()
    }

    //MARK: - Getting Token here
    static func gettingDeviceToken() -> String? {
        let token = defaults.object(forKey: "deviceToken") as? String
    }
}
```

```

        if token == nil{
            return ""
        }else{ return token}
    }

//MARK: - 设置用户详情
static func setUserSessionDetails(_ dic :[String : AnyObject]){
    defaults.removeObject(forKey: "LoginSession")
    defaults.set(dic, forKey: "LoginSession")
    defaults.synchronize()
}

//MARK: - 移除所有默认值
static func userSessionLogout(){
    //设置活动
defaults.removeObject(forKey: "LoginSession")
    defaults.synchronize()
}

//MARK: - 在此处从会话获取值
static func getUserValues(value: userValues) -> String? {
    let dic = getUserSessionDetails() ?? [:]
guard let value = dic[value.rawValue] else{
    return ""
}
    return value as? String
}
}

```

UserDefaults 类的使用

```

//保存用户详情
Session.setUserSessionDetails(json ?? [:])

//获取用户详情
let userId = Session.getUserValues(value: .userId) ?? ""

```

第76.3节：使用管理器保存和读取数据

虽然你可以在任何地方使用`NSUserDefaults`方法，但有时定义一个管理器来为你保存和读取`NSUserDefaults`会更好，然后使用该管理器来读取或写入数据。

假设我们想将用户的分数保存到`NSUserDefaults`。我们可以创建一个如下的类，它至少有两个方法：`setHighScore`和高分。无论何时想访问高分，都创建该类的一个实例。

Swift

```

public class ScoreManager: NSObject {

    let highScoreDefaultKey = "HighScoreDefaultKey"

    var highScore = {
        set {
            // 此方法包含保存最高分的实现
            // 你可以使用 NSUserDefaults 或其他数据存储方式，如 CoreData 或
            // SQLite 等。
            NSUserDefaults.standardUserDefaults().setInteger(newValue, forKey: highScoreDefaultKey)
        }
    }
}

```

```

        if token == nil{
            return ""
        }else{ return token}
    }

//MARK: - Setting here User Details
static func setUserSessionDetails(_ dic :[String : AnyObject]){
    defaults.removeObject(forKey: "LoginSession")
    defaults.set(dic, forKey: "LoginSession")
    defaults.synchronize()
}

//MARK: Removing here all Default Values
static func userSessionLogout(){
    //Set Activity
defaults.removeObject(forKey: "LoginSession")
    defaults.synchronize()
}

//MARK: - Get value from session here
static func getUserValues(value: userValues) -> String? {
    let dic = getUserSessionDetails() ?? [:]
guard let value = dic[value.rawValue] else{
    return ""
}
    return value as? String
}
}

```

Use of UserDefaults Class

```

//Saving user Details
Session.setUserSessionDetails(json ?? [:])

//Retrieving user Details
let userId = Session.getUserValues(value: .userId) ?? ""

```

Section 76.3: Use Managers to Save and Read Data

While you can use the `NSUserDefaults` methods anywhere, it can sometimes be better to define a manager that saves and reads from `NSUserDefaults` for you and then use that manager for reading or writing your data.

Suppose that we want to save a user's score into `NSUserDefaults`. We can create a class like the one below that has at two methods: `setHighScore` and `highScore`. Anywhere you want to access the high scores, create an instance of this class.

Swift

```

public class ScoreManager: NSObject {

    let highScoreDefaultKey = "HighScoreDefaultKey"

    var highScore = {
        set {
            // This method includes your implementation for saving the high score
            // You can use NSUserDefaults or any other data store like CoreData or
            // SQLite etc.
            NSUserDefaults.standardUserDefaults().setInteger(newValue, forKey: highScoreDefaultKey)
        }
    }
}

```

```

        NSUserDefaults.standardUserDefaults().synchronize()
    }
    get {
        // 此方法包含读取最高分的实现

        let score = UserDefaults.standardUserDefaults().objectForKey(highScoreDefaultKey)

        if (score != nil) {
            return score.integerValue;
        } else {
            //没有可用的高分，所以返回 -1
            return -1;
        }
    }
}

```

Objective-C

```

#import "ScoreManager.h"

#define HIGHSCORE_KEY @"highScore"

@implementation ScoreManager

- (void)setHighScore:(NSUInteger) highScore {
    // 此方法包含保存高分的实现
    // 你可以使用 UserDefaults 或其他数据存储方式，如 CoreData 或
    // SQLite 等。
    [[NSUserDefaults standardUserDefaults] setInteger:highScore forKey:HIGHSCORE_KEY];
    [[NSUserDefaults standardUserDefaults] synchronize];
}

- (NSUInteger)highScore
{
    // 此方法包含读取最高分的实现

    NSNumber *highScore = [[NSUserDefaults standardUserDefaults] objectForKey:HIGHSCORE_KEY];
    if (highScore) {
        return highScore.integerValue;
    }else
    {
        //没有可用的高分，因此返回-1
        return -1;
    }
}

@end

```

优点如下：

1. 你的读写过程的实现只在一个地方，可以随时更改（例如从NSUserDefaults切换到Core Data），而不必担心更改所有使用高分的地方。
2. 只需调用一个方法即可访问或写入分数。

```

        NSUserDefaults.standardUserDefaults().synchronize()
    }
    get {
        //This method includes your implementation for reading the high score

        let score = UserDefaults.standardUserDefaults().objectForKey(highScoreDefaultKey)

        if (score != nil) {
            return score.integerValue;
        } else {
            //No high score available, so return -1
            return -1;
        }
    }
}

```

Objective-C

```

#import "ScoreManager.h"

#define HIGHSCORE_KEY @"highScore"

@implementation ScoreManager

- (void)setHighScore:(NSUInteger) highScore {
    // This method includes your implementation for saving the high score
    // You can use UserDefaults or any other data store like CoreData or
    // SQLite etc.
    [[NSUserDefaults standardUserDefaults] setInteger:highScore forKey:HIGHSCORE_KEY];
    [[NSUserDefaults standardUserDefaults] synchronize];
}

- (NSUInteger)highScore
{
    //This method includes your implementation for reading the high score

    NSNumber *highScore = [[NSUserDefaults standardUserDefaults] objectForKey:HIGHSCORE_KEY];
    if (highScore) {
        return highScore.integerValue;
    }else
    {
        //No high score available, so return -1
        return -1;
    }
}

@end

```

The advantages are that:

1. The implementation of your read and write process is only in one place and you can change it (for example switch from `NSUserDefaults` to Core Data) whenever you want and not worry about changing all places that you are working with the high score.
2. Simply call only one method when you want to access to score or write it.

3. 当你发现错误或类似问题时，调试也很简单。

注意

如果你担心同步问题，最好使用一个单例类来管理同步。

3. Simply debug it when you see a bug or something like this.

Note

If you are worried about synchronization, it is better to use a singleton class that manages the synchronization.

第76.4节：保存数值

NSUserDefaults 由系统定期写入磁盘，但有时你希望立即保存更改，例如当应用程序进入后台状态时。这可以通过调用 synchronize 来完成。

Swift

```
NSUserDefaults.standardUserDefaults().synchronize()
```

Objective-C

```
[[NSUserDefaults standardUserDefaults] synchronize];
```

Section 76.4: Saving Values

NSUserDefaults are written to disk periodically by the system, but there are times when you want your changes saved immediately, such as when the app transitions into background state. This is done by calling synchronize.

Swift

```
NSUserDefaults.standardUserDefaults().synchronize()
```

Objective-C

```
[[NSUserDefaults standardUserDefaults] synchronize];
```

第76.5节：清除NSUserDefaults

Swift

```
let bundleIdentifier = NSBundle.mainBundle().bundleIdentifier()  
  
NSUserDefaults.standardUserDefaults().removePersistentDomainForName(bundleIdentifier)
```

Objective-C

```
NSString *bundleIdentifier = [[NSBundle mainBundle] bundleIdentifier];  
  
[[NSUserDefaults standardUserDefaults] removePersistentDomainForName: bundleIdentifier];
```

Section 76.5: Clearing NSUserDefaults

Swift

```
let bundleIdentifier = NSBundle.mainBundle().bundleIdentifier()  
  
NSUserDefaults.standardUserDefaults().removePersistentDomainForName(bundleIdentifier)
```

Objective-C

```
NSString *bundleIdentifier = [[NSBundle mainBundle] bundleIdentifier];  
  
[[NSUserDefaults standardUserDefaults] removePersistentDomainForName: bundleIdentifier];
```

第76.6节：获取默认值

要在NSUserDefaults中获取值，可以使用以下函数：

Swift

```
arrayForKey(_:)  
boolForKey(_:)  
dataForKey(_:)  
dictionaryForKey(_:)  
floatForKey(_:)  
integerForKey(_:)  
objectForKey(_:)  
stringArrayForKey(_:)  
stringForKey(_:)  
doubleForKey(_:)  
URLForKey(_:)
```

Objective-C

```
-(nullable NSArray *)arrayForKey:(nonnull NSString *)defaultName;  
-(BOOL)boolForKey:(nonnull NSString *)defaultName;  
-(nullable NSData *)dataForKey:(nonnull NSString *)defaultName;  
-(nullable NSDictionary<NSString *, id>)dictionaryForKey:(nonnull NSString *)defaultName;
```

Section 76.6: Getting Default Values

To get a value in NSUserDefaults you can use the following functions:

Swift

```
arrayForKey(_:)  
boolForKey(_:)  
dataForKey(_:)  
dictionaryForKey(_:)  
floatForKey(_:)  
integerForKey(_:)  
objectForKey(_:)  
stringArrayForKey(_:)  
stringForKey(_:)  
doubleForKey(_:)  
URLForKey(_:)
```

Objective-C

```
-(nullable NSArray *)arrayForKey:(nonnull NSString *)defaultName;  
-(BOOL)boolForKey:(nonnull NSString *)defaultName;  
-(nullable NSData *)dataForKey:(nonnull NSString *)defaultName;  
-(nullable NSDictionary<NSString *, id>)dictionaryForKey:(nonnull NSString *)defaultName;
```

```
-(float)floatForKey:(nonnull NSString *)defaultValue;
-(NSInteger)integerForKey:(nonnull NSString *)defaultValue;
-(nullable id)objectForKey:(nonnull NSString *)key;
-(nullable NSArray<NSString *> *)stringArrayForKey:(nonnull NSString *)defaultValue;
-(nullable NSString *)stringForKey:(nonnull NSString *)defaultValue;
-(double)doubleForKey:(nonnull NSString *)defaultValue;
-(nullable NSURL *)URLForKey:(nonnull NSString *)defaultValue;
```

示例用法如下：

Swift

```
let homeCountry = UserDefaults.standard.string(forKey: "HomeCountry")
```

Objective-C

```
NSString *homeCountry = [[NSUserDefaults standardUserDefaults] stringForKey:@"HomeCountry"];
```

```
-(float)floatForKey:(nonnull NSString *)defaultValue;
-(NSInteger)integerForKey:(nonnull NSString *)defaultValue;
-(nullable id)objectForKey:(nonnull NSString *)key;
-(nullable NSArray<NSString *> *)stringArrayForKey:(nonnull NSString *)defaultValue;
-(nullable NSString *)stringForKey:(nonnull NSString *)defaultValue;
-(double)doubleForKey:(nonnull NSString *)defaultValue;
-(nullable NSURL *)URLForKey:(nonnull NSString *)defaultValue;
```

Example usage would be:

Swift

```
let homeCountry = UserDefaults.standard.string(forKey: "HomeCountry")
```

Objective-C

```
NSString *homeCountry = [[NSUserDefaults standardUserDefaults] stringForKey:@"HomeCountry"];
```

第77章：NSHTTPCookieStorage

第77.1节：从NSUserDefaults存储和读取Cookie

```
import Foundation

class CookiesSingleton {

    static let instance : CookiesSingleton = CookiesSingleton()
    static var enableDebug = true

    func loadCookies() {
        if let cookiesDetails = UserDefaults.standardUserDefaults().objectForKey("customeWebsite") {
            for (keys,_) in cookiesDetails as! NSDictionary{
                if let cookieDict = UserDefaults.standardUserDefaults().objectForKey(keys as!
字符串){
                    if let cookie = NSHTTPCookie(properties:cookieDict as! [String:AnyObject]) {
                        NSHTTPCookieStorage.sharedHTTPCookieStorage().setCookie(cookie)
                        if(CookiesSingleton.enableDebug){
                            print("Each Cookies",cookieDict)
                        }
                    }
                }
            }
        }
    }

    func removeCookies(){
        NSURLCache.sharedURLCache().removeAllCachedResponses()
        NSURLCache.sharedURLCache().diskCapacity = 0
        NSURLCache.sharedURLCache().memoryCapacity = 0

        let storage : NSHTTPCookieStorage = NSHTTPCookieStorage.sharedHTTPCookieStorage()
        for cookie in storage.cookies! {
            storage.deleteCookie(cookie as NSHTTPCookie)
        }

        UserDefaults.standardUserDefaults().setValue("", forKey: "customeWebsite")
        UserDefaults.standardUserDefaults().synchronize()

        if(CookiesSingleton.enableDebug){
            print("Cookies Removed")
        }
    }

    func saveCookies() {

        let cookieArray = NSMutableArray()
        let savedC = NSHTTPCookieStorage.sharedHTTPCookieStorage().cookies

        let allCookiesDic:NSMutableDictionary = NSMutableDictionary()

        for c : NSHTTPCookie in savedC! {

            let cookieProps = NSMutableDictionary()
            cookieArray.addObject(c.name)
            cookieProps.setValue(c.name, forKey: NSHTTPCookieName)
            cookieProps.setValue(c.value, forKey: NSHTTPCookieValue)
            cookieProps.setValue(c.domain, forKey: NSHTTPCookieDomain)
        }
    }
}
```

Chapter 77: NSHTTPCookieStorage

Section 77.1: Store and read the cookies from UserDefaults

```
import Foundation

class CookiesSingleton {

    static let instance : CookiesSingleton = CookiesSingleton()
    static var enableDebug = true

    func loadCookies() {
        if let cookiesDetails = UserDefaults.standardUserDefaults().objectForKey("customeWebsite") {
            for (keys,_) in cookiesDetails as! NSDictionary{
                if let cookieDict = UserDefaults.standardUserDefaults().objectForKey(keys as!
String){
                    if let cookie = NSHTTPCookie(properties:cookieDict as! [String:AnyObject]) {
                        NSHTTPCookieStorage.sharedHTTPCookieStorage().setCookie(cookie)
                        if(CookiesSingleton.enableDebug){
                            print("Each Cookies",cookieDict)
                        }
                    }
                }
            }
        }
    }

    func removeCookies(){
        NSURLCache.sharedURLCache().removeAllCachedResponses()
        NSURLCache.sharedURLCache().diskCapacity = 0
        NSURLCache.sharedURLCache().memoryCapacity = 0

        let storage : NSHTTPCookieStorage = NSHTTPCookieStorage.sharedHTTPCookieStorage()
        for cookie in storage.cookies! {
            storage.deleteCookie(cookie as NSHTTPCookie)
        }

        UserDefaults.standardUserDefaults().setValue("", forKey: "customeWebsite")
        UserDefaults.standardUserDefaults().synchronize()

        if(CookiesSingleton.enableDebug){
            print("Cookies Removed")
        }
    }

    func saveCookies() {

        let cookieArray = NSMutableArray()
        let savedC = NSHTTPCookieStorage.sharedHTTPCookieStorage().cookies

        let allCookiesDic:NSMutableDictionary = NSMutableDictionary()

        for c : NSHTTPCookie in savedC! {

            let cookieProps = NSMutableDictionary()
            cookieArray.addObject(c.name)
            cookieProps.setValue(c.name, forKey: NSHTTPCookieName)
            cookieProps.setValue(c.value, forKey: NSHTTPCookieValue)
            cookieProps.setValue(c.domain, forKey: NSHTTPCookieDomain)
        }
    }
}
```

```
cookieProps.setValue(c.path, forKey: NSHTTPCookiePath)
    cookieProps.setValue(c.version, forKey: NSHTTPCookieVersion)
    cookieProps.setValue(NSDate().dateByAddingTimeInterval(2629743), forKey:
NSHTTPCookieExpires)

allCookiesDic.setValue(cookieProps, forKey: c.name)

}

NSUserDefaults.standardUserDefaults().setValue(allCookiesDic, forKey: "customewebsite")
NSUserDefaults.standardUserDefaults().synchronize()

if(CookiesSingleton.enableDebug){
    print("Cookies Saved")
}
}
```

```
cookieProps.setValue(c.path, forKey: NSHTTPCookiePath)
    cookieProps.setValue(c.version, forKey: NSHTTPCookieVersion)
    cookieProps.setValue(NSDate().dateByAddingTimeInterval(2629743), forKey:
NSHTTPCookieExpires)

allCookiesDic.setValue(cookieProps, forKey: c.name)

}

NSUserDefaults.standardUserDefaults().setValue(allCookiesDic, forKey: "customewebsite")
NSUserDefaults.standardUserDefaults().synchronize()

if(CookiesSingleton.enableDebug){
    print("Cookies Saved")
}
}
```

第78章：NSURLConnection

第78.1节：代理方法

```
//符合NSURLConnectionDelegate协议。  
  
@interface ViewController : UIViewController<NSURLConnectionDelegate>  
{  
    NSMutableData *_responseData;  
}  
  
// NSURLConnection 协议方法的实现。  
  
#pragma mark NSURLConnection 代理方法  
  
- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse *)response {  
    // 已收到响应，这里初始化你创建的实例变量  
    // 以便在 didReceiveData 方法中追加数据  
    // 此外，每次发生重定向时都会调用此方法，因此重新初始化  
    // 也起到清空的作用  
    _responseData = [[NSMutableData alloc] init];  
}  
  
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data {  
    // 将新数据追加到你声明的实例变量中  
    [_responseData appendData:data];  
}  
  
- (NSCachedURLResponse *)connection:(NSURLConnection *)connection  
willCacheResponse:(NSCachedURLResponse*)cachedResponse {  
    // 返回 nil 表示不需要为此连接存储缓存响应  
    return nil;  
}  
  
- (void)connection:(NSURLConnection *)connection {  
    // 请求已完成，数据已接收  
    // 你现在可以解析实例变量中的内容了  
}  
  
- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error {  
    // 请求因某种原因失败了！  
    // 检查 error 变量  
}
```

第78.2节：同步请求

```
NSURLRequest * urlRequest = [NSURLRequest requestWithURL:[NSURL  
URLWithString:@"http://google.com"]];  
NSURLResponse * response = nil;  
NSError * error = nil;  
NSData * data = [NSURLConnection sendSynchronousRequest:urlRequest  
returningResponse:&response  
error:&error];  
  
if (error == nil)  
{  
    // 在这里解析数据
```

Chapter 78: NSURLConnection

Section 78.1: Delegate methods

```
//conform the NSURLConnectionDelegate protocol.  
  
@interface ViewController : UIViewController<NSURLConnectionDelegate>  
{  
    NSMutableData *_responseData;  
}  
  
//Implementation of the NSURLConnection protocol methods.  
  
#pragma mark NSURLConnection Delegate Methods  
  
- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse *)response {  
    // A response has been received, this is where we initialize the instance var you created  
    // so that we can append data to it in the didReceiveData method  
    // Furthermore, this method is called each time there is a redirect so reinitializing it  
    // also serves to clear it  
    _responseData = [[NSMutableData alloc] init];  
}  
  
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data {  
    // Append the new data to the instance variable you declared  
    [_responseData appendData:data];  
}  
  
- (NSCachedURLResponse *)connection:(NSURLConnection *)connection  
willCacheResponse:(NSCachedURLResponse*)cachedResponse {  
    // Return nil to indicate not necessary to store a cached response for this connection  
    return nil;  
}  
  
- (void)connectionDidFinishLoading:(NSURLConnection *)connection {  
    // The request is complete and data has been received  
    // You can parse the stuff in your instance variable now  
}  
  
- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error {  
    // The request has failed for some reason!  
    // Check the error var  
}
```

Section 78.2: Synchronous Request

```
NSURLRequest * urlRequest = [NSURLRequest requestWithURL:[NSURL  
URLWithString:@"http://google.com"]];  
NSURLResponse * response = nil;  
NSError * error = nil;  
NSData * data = [NSURLConnection sendSynchronousRequest:urlRequest  
returningResponse:&response  
error:&error];  
  
if (error == nil)  
{  
    // Parse data here
```

}

第78.3节：异步请求

```
// 创建请求实例。  
NSURLRequest *request = [NSURLRequest requestWithURL:[NSURL URLWithString:@"http://google.com"]];  
  
// 创建URL连接并发送请求  
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

}

Section 78.3: Asynchronous request

```
// Create the request instance.  
NSURLRequest *request = [NSURLRequest requestWithURL:[NSURL URLWithString:@"http://google.com"]];  
  
// Create url connection and fire request  
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

第79章：NSURL

第79.1节：如何从NSURL字符串获取最后的字符串组件

```
NSURL *url = [NSURL URLWithString:@"http://www.example.com/images/apple-tree.jpg"];
NSString *fileName = [url lastPathComponent];
// fileName = "apple-tree.jpg"
```

第79.2节：如何从Swift中的URL（NSURL）获取最后的字符串组件

Swift 2.3

```
let url = NSURL(string: "http://google.com/lastPath")
let lastPath = url?.lastPathComponent
```

Swift 3.0

```
let url = URL(string: "http://google.com/lastPath")
let lastPath = url?.lastPathComponent
```

Chapter 79: NSURL

Section 79.1: How to get last string component from NSURL String

```
NSURL *url = [NSURL URLWithString:@"http://www.example.com/images/apple-tree.jpg"];
NSString *fileName = [url lastPathComponent];
// fileName = "apple-tree.jpg"
```

Section 79.2: How to get last string component from URL (NSURL) in Swift

Swift 2.3

```
let url = NSURL(string: "http://google.com/lastPath")
let lastPath = url?.lastPathComponent
```

Swift 3.0

```
let url = URL(string: "http://google.com/lastPath")
let lastPath = url?.lastPathComponent
```

第80章：NSData

第80.1节：将NSData转换为十六进制字符串

NSData 可以表示为十六进制字符串，类似于其 `description` 方法输出的内容。

Swift

```
extension NSData {  
  
    func hexString() -> String {  
        return UnsafeBufferPointer<UInt8>(start: UnsafePointer<UInt8>(bytes), count: length)  
            .reduce("") { $0 + String(format: "%02x", $1) }  
    }  
  
}
```

Objective-C

```
@implementation NSData (HexRepresentation)  
  
- (NSString *)hexString {  
    const unsigned char *bytes = (const unsigned char *)self.bytes;  
    NSMutableString *hex = [NSMutableString new];  
    for (NSInteger i = 0; i < self.length; i++) {  
        [hex appendFormat:@"%@", bytes[i]];  
    }  
    return [hex copy];  
}  
  
@end
```

第80.2节：创建NSData对象

使用文件

Swift

```
let data = NSData(contentsOfFile: filePath) //假设filePath是有效路径
```

Objective-C

```
NSData *data = [NSData dataWithContentsOfFile:filePath]; //假设filePath是有效路径
```

使用字符串对象

Swift

```
let data = (string as NSString).dataUsingEncoding(NSUTF8StringEncoding) //假设 string 是一个  
String 对象
```

Objective-C

```
NSData *data = [string dataUsingEncoding:NSUTF8StringEncoding]; //假设 string 是一个 String  
对象
```

第 80.3 节：将 NSData 转换为其他类型

转换为字符串

Swift

```
let string = String(NSString(data: data, encoding: NSUTF8StringEncoding)) //假设 data 是一个
```

Chapter 80: NSData

Section 80.1: Converting NSData to HEX string

NSData can be represented as hexadecimal string, similar to what it outputs in its `description` method.

Swift

```
extension NSData {  
  
    func hexString() -> String {  
        return UnsafeBufferPointer<UInt8>(start: UnsafePointer<UInt8>(bytes), count: length)  
            .reduce("") { $0 + String(format: "%02x", $1) }  
    }  
  
}
```

Objective-C

```
@implementation NSData (HexRepresentation)  
  
- (NSString *)hexString {  
    const unsigned char *bytes = (const unsigned char *)self.bytes;  
    NSMutableString *hex = [NSMutableString new];  
    for (NSInteger i = 0; i < self.length; i++) {  
        [hex appendFormat:@"%@", bytes[i]];  
    }  
    return [hex copy];  
}  
  
@end
```

Section 80.2: Creating NSData objects

Using a file

Swift

```
let data = NSData(contentsOfFile: filePath) //assuming filePath is a valid path
```

Objective-C

```
NSData *data = [NSData dataWithContentsOfFile:filePath]; //assuming filePath is a valid path
```

Using a String object

Swift

```
let data = (string as NSString).dataUsingEncoding(NSUTF8StringEncoding) //assuming string is a  
String object
```

Objective-C

```
NSData *data = [string dataUsingEncoding:NSUTF8StringEncoding]; //assuming string is a String  
object
```

Section 80.3: Converting NSData to other types

To String

Swift

```
let string = String(NSString(data: data, encoding: NSUTF8StringEncoding)) //assuming data is a
```

Objective-C

```
NSString *string = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding]; //假设  
data 是一个有效的 NSData 对象  
[string release];
```

转换为数组

Swift

```
let array = data.bytes as! NSMutableArray //假设 data 是一个有效的 NSData 对象
```

Objective-C

```
NSMutableArray *array = (NSMutableArray *)[data bytes]; //假设 data 是一个有效的 NSData 对象
```

转换为字节数组

Swift

```
let bytesArray = data.bytes as! UInt8 //假设 data 是一个有效的 NSData 对象
```

Objective-C

```
UInt8 *bytesArray = (UInt8 *)data.bytes; //假设 data 是一个有效的 NSData 对象
```

Objective-C

```
NSString *string = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding]; //assuming  
data is a valid NSData object  
[string release];
```

To Array

Swift

```
let array = data.bytes as! NSMutableArray //assuming data is a valid NSData object
```

Objective-C

```
NSMutableArray *array = (NSMutableArray *)[data bytes]; //assuming data is a valid NSData object
```

To Bytes Array

Swift

```
let bytesArray = data.bytes as! UInt8 //assuming data is a valid NSData object
```

Objective-C

```
UInt8 *bytesArray = (UInt8 *)data.bytes; //assuming data is a valid NSData object
```

第81章：NSInvocation

第81.1节：NSInvocation Objective-C

参考这篇 [原文](#) 由 [e.James](#)

根据 [苹果的 NSInvocation 类参考文档](#)：

NSInvocation 是一个 Objective-C 消息的静态表现，也就是说，它是一个被转化为对象的动作。

更详细一点：

消息的概念是Objective-C哲学的核心。每当你调用某个对象的方法或访问其变量时，你就是在向它发送一条消息。 NSInvocation 在你想在不同时间点向对象发送消息，或多次发送同一条消息时非常有用。 NSInvocation 允许你描述你将要发送的消息，然后稍后调用它（实际上发送给目标对象）。

例如，假设你想向数组添加一个字符串。你通常会发送addObject:消息，方式如下：

```
[myArray addObject:myString];
```

现在，假设你想使用NSInvocation在其他时间点发送这条消息：

首先，你需要为NSMutableArray的addObject:选择器准备一个NSInvocation对象：

```
NSMethodSignature * mySignature = [NSMutableArray  
instanceMethodSignatureForSelector:@selector(addObject:)];  
NSInvocation * myInvocation = [NSInvocation  
invocationWithMethodSignature:mySignature];
```

接下来，你需要指定要向哪个对象发送消息：

```
[myInvocation setTarget:myArray];
```

指定你希望发送给该对象的消息：

```
[myInvocation setSelector:@selector(addObject:)];
```

并填写该方法的所有参数：

```
[myInvocation setArgument:&myString atIndex:2];
```

请注意，对象参数必须以指针方式传递。感谢[Ryan McCuaig](#)指出这一点，[更多详情请参见Apple的文档](#)。

此时，myInvocation 是一个完整的对象，描述了可以发送的消息。要实际发送该消息，你需要调用：

```
[myInvocation invoke];
```

这最后一步将导致消息被发送，实质上执行了[myArray addObject:myString];。

Chapter 81: NSInvocation

Section 81.1: NSInvocation Objective-C

Refer to this [original Post](#) by [e.James](#)

According to [Apple's NSInvocation class reference](#):

An [NSInvocation](#) is an Objective-C message rendered static, that is, it is an action turned into an object.

And, in a *little* more detail:

The concept of messages is central to the objective-c philosophy. Any time you call a method, or access a variable of some object, you are sending it a message. [NSInvocation](#) comes in handy when you want to send a message to an object at a different point in time, or send the same message several times. [NSInvocation](#) allows you to *describe* the message you are going to send, and then *invoke* it (actually send it to the target object) later on.

For example, let's say you want to add a string to an array. You would normally send the addObject: message as follows:

```
[myArray addObject:myString];
```

Now, let's say you want to use [NSInvocation](#) to send this message at some other point in time:

First, you would prepare an [NSInvocation](#) object for use with [NSMutableArray's addObject:](#) selector:

```
NSMethodSignature * mySignature = [NSMutableArray  
instanceMethodSignatureForSelector:@selector(addObject:)];  
NSInvocation * myInvocation = [NSInvocation  
invocationWithMethodSignature:mySignature];
```

Next, you would specify which object to send the message to:

```
[myInvocation setTarget:myArray];
```

Specify the message you wish to send to that object:

```
[myInvocation setSelector:@selector(addObject:)];
```

And fill in any arguments for that method:

```
[myInvocation setArgument:&myString atIndex:2];
```

Note that object arguments must be passed by pointer. Thank you to [Ryan McCuaig](#) for pointing that out, and please see [Apple's documentation](#) for more details.

At this point, myInvocation is a complete object, describing a message that can be sent. To actually send the message, you would call:

```
[myInvocation invoke];
```

This final step will cause the message to be sent, essentially executing [myArray addObject:myString];.

可以把它想象成发送电子邮件。你打开一封新邮件（`NSInvocation`对象），填写你想发送人的地址（对象），为收件人输入消息（指定selector和参数），然后点击“发送”（调用`invoke`）。

更多信息请参见使用`NSInvocation`。

`NSUndoManager`使用`NSInvocation`对象来实现撤销命令。基本上，你所做的是创建一个`NSInvocation`对象，表示：“嘿，如果你想撤销我刚才做的操作，就向该对象发送这条消息，并带上这些参数”。你将`NSInvocation`对象交给`NSUndoManager`，它会将该对象添加到可撤销操作的数组中。如果用户调用“撤销”，`NSUndoManager`只需查找数组中最近的操作，并调用存储的`NSInvocation`对象来执行必要的操作。

有关详细信息，请参阅注册撤销操作。

Think of it like sending an email. You open up a new email (`NSInvocation` object), fill in the address of the person (object) who you want to send it to, type in a message for the recipient (specify a selector and arguments), and then click "send" (call `invoke`).

See [Using `NSInvocation`](#) for more information.

`NSUndoManager` uses `NSInvocation` objects so that it can *reverse* commands. Essentially, what you are doing is creating an `NSInvocation` object to say: "Hey, if you want to undo what I just did, send this message to that object, with these arguments". You give the `NSInvocation` object to the `NSUndoManager`, and it adds that object to an array of undoable actions. If the user calls "Undo", `NSUndoManager` simply looks up the most recent action in the array, and invokes the stored `NSInvocation` object to perform the necessary action.

See [Registering Undo Operations](#) for more details.

第82章：NSUserActivity

NSUserActivity对象可用于协调应用程序中的重要事件与系统的交互。它是不同运行iOS和macOS设备之间Handoff功能的基础。此外，它还可用于改进公共索引，并增强或创建应用程序的Spotlight搜索结果。从iOS 10开始，它还可用于协调您的应用程序与Siri之间通过SiriKit的交互。

第82.1节：创建NSUserActivity

要创建一个NSUserActivity对象，您的应用必须在其Info.plist文件中声明支持的活动类型。

支持的活动由您的应用定义，且应当唯一。活动使用反向域名风格的命名方案定义（例如“com.companyName.productName.activityName”）。以下是Info.plist中的一个示例条目：

键	值
NSUserActivityTypes [数组]	
- 项目0	com.companyName.productName.activityName01
- 项目1	com.companyName.productName.activityName02

一旦定义了所有支持的活动类型，您就可以开始在应用程序的

代码中访问和使用它们。

要创建一个NSUserActivity对象，您必须执行以下操作

```
// 初始化活动对象并从应用程序的
NSUserActivity *currentActivity = [[NSUserActivity alloc]
initWithActivityType:@"com.companyName.productName.activityName01"];

// 设置活动的标题。
// 该标题可能会显示给用户，因此请确保其已本地化且易于理解
currentActivity.title = @"当前活动";

// 配置其他属性，如将包含在活动中的userInfo
currentActivity.userInfo = @{@"informationKey" : @"value"};

// 配置活动，以便系统知道可以对其执行的操作
// 重要的是，您只对应用程序支持的任务设置YES
// 在此示例中，我们仅启用该活动以用于Handoff
[currentActivity setEligibleForHandoff:YES];
[currentActivity setEligibleForSearch:NO]; // 默认为NO
[currentActivity setEligibleForPublicIndexing:NO]; // 默认为NO

// 将此活动设置为当前用户活动
// 在同一时间内，设备上只能有一个活动处于当前状态。调用此方法会使任何其他当前活动失效。
[currentActivity becomeCurrent];
```

之后，上述活动应该可以用于接力（Handoff）（尽管还需要更多工作来正确处理“接力”）。

Chapter 82: NSUserActivity

An NSUserActivity object can be used to coordinate significant events in an app with the system. It is the basis for [Handoff](#) between different devices running iOS and macOS. Additionally, it may also be used to improve public-indexing and augment or create Spotlight Search results for an app. As of iOS 10, it may also be used to coordinate interactions between your app and Siri using SiriKit.

Section 82.1: Creating a NSUserActivity

To create a [NSUserActivity](#) object, your app must declare the types of activities it supports in its [Info.plist](#) file. Supported activities are defined by your application and should be unique. An activity is defined using a reverse-domain style naming scheme (i.e. "com.companyName.productName.activityName"). Here is what an entry in your Info.plist might look like:

Key	Value
NSUserActivityTypes [Array]	
- item0	com.companyName.productName.activityName01
- item1	com.companyName.productName.activityName02

Once you have defined all supported activity types, you may begin to access and use them in your application's code.

To create a [NSUserActivity](#) object you must do the following

```
// Initialize the activity object and set its type from one of the ones specified in your app's
plist
NSUserActivity *currentActivity = [[NSUserActivity alloc]
initWithActivityType:@"com.companyName.productName.activityName01"];

// Set the title of the activity.
// This title may be displayed to the user, so make sure it is localized and human-readable
currentActivity.title = @"Current Activity";

// Configure additional properties like userInfo which will be included in the activity
currentActivity.userInfo = @{@"informationKey" : @"value"};

// Configure the activity so the system knows what may be done with it
// It is important that you only set YES to tasks that your application supports
// In this example, we will only enable the activity for use with Handoff
[currentActivity setEligibleForHandoff:YES];
[currentActivity setEligibleForSearch:NO]; // Defaults to NO
[currentActivity setEligibleForPublicIndexing:NO]; // Defaults to NO

// Set this activity as the current user activity
// Only one activity may be current at a time on a device. Calling this method invalidates any
other current activities.
[currentActivity becomeCurrent];
```

After this, the activity above should be available for Handoff (although more work is required to properly handle the "Handoff").

第83章：NSPredicate

第83.1节：使用NSPredicate进行表单验证

```
NSString *emailRegex = @"[A-Z0-9a-z][A-Z0-9a-zA-Z]{0,64}+[A-Z0-9a-zA-Z]+@[A-Z0-9a-zA-Z]+([A-Za-z0-9.-]{0,64})+([A-Z0-9a-zA-Z])+\\.[A-Za-z]{2,4}"; NSString *firstNameRegex = @"[0-9A-Za-z\"'-]{2,32}$"; NSString *lastNameRegex = @"[0-9A-Za-z\"'-]{2,32}$"; NSString *mobileNumberRegEx = @"^+[0-9]{10}$"; NSString *zipcodeRegEx = @"^+[0-9]{5}$"; NSString *SSNRegEx = @"^\\d{3}-?\\d{2}-?\\d{4}$"; NSString *addressRegEx = @"^+[A-Za-z0-9]{2,32}$"; NSString *cityRegEx = @"^+[A-Za-z0-9]{2,25}$"; NSString *PINRegEx = @"^+[0-9]{4}$"; NSString *driversLiscRegEx = @"^+[0-9a-zA-Z]{5,20}$";  
  
-(BOOL)validateEmail {  
    //当电子邮件地址以"."、"-"、"_"开头时，电子邮件地址字段应显示错误。  
    NSPredicate *emailPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", emailRegex];  
  
    return ([emailPredicate evaluateWithObject:self.text] && self.text.length <= 64 && ([self.text rangeOfString:@".."].location == NSNotFound));  
}  
  
-(BOOL)validateFirstName {  
    NSPredicate *firstNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", firstNameRegex];  
    return [firstNamePredicate evaluateWithObject:self.text];  
}  
  
-(BOOL)validateLastName {  
    NSPredicate *lastNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", lastNameRegex];  
    return [lastNamePredicate evaluateWithObject:self.text];  
}  
  
-(BOOL)validateAlphaNumericMin2Max32 {  
    NSPredicate *firstNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", firstNameRegex];  
    return [firstNamePredicate evaluateWithObject:self.text];  
}  
  
-(BOOL)validateMobileNumber {  
    NSString *strippedMobileNumber = [[[self.text stringByReplacingOccurrencesOfString:@"(" withString:@"") stringByReplacingOccurrencesOfString:@")" withString:@""] stringByReplacingOccurrencesOfString:@"-" withString:@""] stringByReplacingOccurrencesOfString:@" " withString:@""];  
  
    NSPredicate *mobileNumberPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", mobileNumberRegEx];  
  
    return [mobileNumberPredicate evaluateWithObject:strippedMobileNumber];  
}  
  
-(BOOL)validateZipcode {  
    NSPredicate *zipcodePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", zipcodeRegEx];  
  
    return [zipcodePredicate evaluateWithObject:self.text];  
}
```

Chapter 83: NSPredicate

Section 83.1: Form validation using NSPredicate

```
NSString *emailRegex = @"[A-Z0-9a-zA-Z][A-Z0-9a-zA-Z]{0,64}+[A-Z0-9a-zA-Z]+@[A-Z0-9a-zA-Z]+([A-Za-z0-9.-]{0,64})+([A-Z0-9a-zA-Z])+\\.[A-Za-z]{2,4}"; NSString *firstNameRegex = @"[0-9A-Za-z\"'-]{2,32}$"; NSString *lastNameRegex = @"[0-9A-Za-z\"'-]{2,32}$"; NSString *mobileNumberRegEx = @"^+[0-9]{10}$"; NSString *zipcodeRegEx = @"^+[0-9]{5}$"; NSString *SSNRegEx = @"^\\d{3}-?\\d{2}-?\\d{4}$"; NSString *addressRegEx = @"^+[A-Za-z0-9]{2,32}$"; NSString *cityRegEx = @"^+[A-Za-z0-9]{2,25}$"; NSString *PINRegEx = @"^+[0-9]{4}$"; NSString *driversLiscRegEx = @"^+[0-9a-zA-Z]{5,20}$";  
  
-(BOOL)validateEmail {  
    //Email address field should give an error when the email address begins with ".", "-", "_".  
    NSPredicate *emailPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", emailRegex];  
  
    return ([emailPredicate evaluateWithObject:self.text] && self.text.length <= 64 && ([self.text rangeOfString:@".."].location == NSNotFound));  
}  
  
-(BOOL)validateFirstName {  
    NSPredicate *firstNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", firstNameRegex];  
    return [firstNamePredicate evaluateWithObject:self.text];  
}  
  
-(BOOL)validateLastName {  
    NSPredicate *lastNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", lastNameRegex];  
    return [lastNamePredicate evaluateWithObject:self.text];  
}  
  
-(BOOL)validateAlphaNumericMin2Max32 {  
    NSPredicate *firstNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", firstNameRegex];  
    return [firstNamePredicate evaluateWithObject:self.text];  
}  
  
-(BOOL)validateMobileNumber {  
    NSString *strippedMobileNumber = [[[self.text stringByReplacingOccurrencesOfString:@"(" withString:@"") stringByReplacingOccurrencesOfString:@")" withString:@""] stringByReplacingOccurrencesOfString:@"-" withString:@""] stringByReplacingOccurrencesOfString:@" " withString:@""];  
  
    NSPredicate *mobileNumberPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", mobileNumberRegEx];  
  
    return [mobileNumberPredicate evaluateWithObject:strippedMobileNumber];  
}  
  
-(BOOL)validateZipcode {  
    NSPredicate *zipcodePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", zipcodeRegEx];  
  
    return [zipcodePredicate evaluateWithObject:self.text];  
}
```

```

}

- (BOOL)validateSSN {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", SSNRegEx];
    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateAddress {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", addressRegEx];
    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateCity {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", cityRegEx];
    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validatePIN {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", PINRegEx];
    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateDriversLiscNumber {
    if([self.text length] > 20) {
        return NO;
    }
    NSPredicate *driversLiscPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", driversLiscRegEx];
    return [driversLiscPredicate evaluateWithObject:self.text];
}

```

第83.2节：使用

`predicateWithBlock` 创建 `NSPredicate`

Objective-C

```

NSPredicate *predicate = [NSPredicate predicateWithBlock:^BOOL(id item,
                                                       NSDictionary *bindings) {
    return [item isKindOfClass:[UILabel class]];
}];

```

Swift

```

let predicate = NSPredicate { (item, bindings) -> Bool in
    return item.isKindOfClass(UILabel.self)
}

```

在此示例中，谓词将匹配属于 `UILabel` 类的项目。

第83.3节：使用

`predicateWithFormat` 创建 `NSPredicate`

Objective-C

```

NSPredicate *predicate = [NSPredicate predicateWithFormat: @"self[SIZE] = %d", 5];

```

Swift

```

}

- (BOOL)validateSSN {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", SSNRegEx];
    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateAddress {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", addressRegEx];
    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateCity {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", cityRegEx];
    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validatePIN {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", PINRegEx];
    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateDriversLiscNumber {
    if([self.text length] > 20) {
        return NO;
    }
    NSPredicate *driversLiscPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", driversLiscRegEx];
    return [driversLiscPredicate evaluateWithObject:self.text];
}

```

Section 83.2: Creating an NSPredicate Using `predicateWithBlock`

Objective-C

```

NSPredicate *predicate = [NSPredicate predicateWithBlock:^BOOL(id item,
                                                       NSDictionary *bindings) {
    return [item isKindOfClass:[UILabel class]];
}];

```

Swift

```

let predicate = NSPredicate { (item, bindings) -> Bool in
    return item.isKindOfClass(UILabel.self)
}

```

In this example, the predicate will match items that are of the class `UILabel`.

Section 83.3: Creating an NSPredicate Using `predicateWithFormat`

Objective-C

```

NSPredicate *predicate = [NSPredicate predicateWithFormat: @"self[SIZE] = %d", 5];

```

Swift

```
let predicate = NSPredicate(format: "self[SIZE] >= %d", 5)
```

在此示例中，谓词将匹配长度至少为5的数组项目。

第83.4节：使用替换变量创建 NSPredicate

NSPredicate 可以使用替换变量以允许动态绑定值。

Objective-C

```
NSPredicate *template = [NSPredicate predicateWithFormat: @"self BEGINSWITH $letter"];
NSDictionary *variables = @{@"letter": @"r"};
NSPredicate *beginsWithR = [template predicateWithSubstitutionVariables: variables];
```

Swift

```
let template = NSPredicate(format: "self BEGINSWITH $letter")
let variables = ["letter": "r"]
let beginsWithR = template.predicateWithSubstitutionVariables(variables)
```

模板谓词不会被 `predicateWithSubstitutionVariables` 修改。相反，会创建一个副本，该副本接收替换变量。

第83.5节：带有`AND`、`OR`和`NOT`条件的NSPredicate

使用 `NSCompoundPredicate` 类可以使条件谓词更简洁且更安全，该类为给定的谓词提供基本的布尔运算符。

Objective-C

AND - 条件

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
NSPredicate *combinedPredicate = [NSCompoundPredicate andPredicateWithSubpredicates:
@+[predicate,anotherPredicate]];
```

OR - 条件

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
NSPredicate *combinedPredicate = [NSCompoundPredicate orPredicateWithSubpredicates:
@+[predicate,anotherPredicate]];
```

非 - 条件

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
NSPredicate *combinedPredicate = [NSCompoundPredicate notPredicateWithSubpredicate:
@+[predicate,anotherPredicate]];
```

第83.6节：使用NSPredicate过滤数组

Objective-C

```
NSArray *heroes = @[@"tracer", @"bastion", @"reaper", @"junkrat", @"roadhog"];
NSPredicate *template = [NSPredicate predicateWithFormat:@"self BEGINSWITH $letter"];
```

```
let predicate = NSPredicate(format: "self[SIZE] >= %d", 5)
```

In this example, the predicate will match items that are arrays with length of at least 5.

Section 83.4: Creating an NSPredicate with Substitution Variables

An `NSPredicate` can use substitution variables to allow values to be bound on the fly.

Objective-C

```
NSPredicate *template = [NSPredicate predicateWithFormat: @"self BEGINSWITH $letter"];
NSDictionary *variables = @{@"letter": @"r"};
NSPredicate *beginsWithR = [template predicateWithSubstitutionVariables: variables];
```

Swift

```
let template = NSPredicate(format: "self BEGINSWITH $letter")
let variables = ["letter": "r"]
let beginsWithR = template.predicateWithSubstitutionVariables(variables)
```

The template predicate is not modified by `predicateWithSubstitutionVariables`. Instead, a copy is created, and that copy receives the substitution variables.

Section 83.5: NSPredicate with `AND`、`OR` and `NOT` condition

Conditional predicate will be cleaner and safer by using the `NSCompoundPredicate` class which provides basic boolean operators for the given predicates.

Objective-C

AND - Condition

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
NSPredicate *combinedPredicate = [NSCompoundPredicate andPredicateWithSubpredicates:
@+[predicate,anotherPredicate]];
```

OR - Condition

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
NSPredicate *combinedPredicate = [NSCompoundPredicate orPredicateWithSubpredicates:
@+[predicate,anotherPredicate]];
```

NOT - Condition

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
NSPredicate *combinedPredicate = [NSCompoundPredicate notPredicateWithSubpredicate:
@+[predicate,anotherPredicate]];
```

Section 83.6: Using NSPredicate to Filter an Array

Objective-C

```
NSArray *heroes = @[@"tracer", @"bastion", @"reaper", @"junkrat", @"roadhog"];
NSPredicate *template = [NSPredicate predicateWithFormat:@"self BEGINSWITH $letter"];
```

```

NSDictionary *beginsWithRVariables = @{@"letter": @"r"};
NSPredicate *beginsWithR = [template predicateWithSubstitutionVariables: beginsWithRVariables];

NSArray *beginsWithRHeroes = [heroes filteredArrayUsingPredicate: beginsWithR];
// ["reaper", "roadhog"]

NSDictionary *beginsWithTVariables = @{@"letter": @"t"};
NSPredicate *beginsWithT = [template predicateWithSubstitutionVariables: beginsWithTVariables];

NSArray *beginsWithTHeroes = [heroes filteredArrayUsingPredicate: beginsWithT];
// ["tracer"]

```

Swift

```

let heroes = ["tracer", "bastion", "reaper", "junkrat", "roadhog"]

let template = NSPredicate(format: "self BEGINSWITH %@", $letter)

let beginsWithRVariables = ["letter": "r"]
let beginsWithR = template.predicateWithSubstitutionVariables(beginsWithRVariables)

let beginsWithRHeroes = heroes.filter { beginsWithR.evaluateWithObject($0) }
// ["reaper", "roadhog"]

let beginsWithTVariables = ["letter": "t"]
let beginsWithT = template.predicateWithSubstitutionVariables(beginsWithTVariables)

let beginsWithTHeroes = heroes.filter { beginsWithT.evaluateWithObject($0) }
// ["tracer"]

```

```

NSDictionary *beginsWithRVariables = @{@"letter": @"r"};
NSPredicate *beginsWithR = [template predicateWithSubstitutionVariables: beginsWithRVariables];

NSArray *beginsWithRHeroes = [heroes filteredArrayUsingPredicate: beginsWithR];
// ["reaper", "roadhog"]

NSDictionary *beginsWithTVariables = @{@"letter": @"t"};
NSPredicate *beginsWithT = [template predicateWithSubstitutionVariables: beginsWithTVariables];

NSArray *beginsWithTHeroes = [heroes filteredArrayUsingPredicate: beginsWithT];
// ["tracer"]

```

Swift

```

let heroes = ["tracer", "bastion", "reaper", "junkrat", "roadhog"]

let template = NSPredicate(format: "self BEGINSWITH %@", $letter)

let beginsWithRVariables = ["letter": "r"]
let beginsWithR = template.predicateWithSubstitutionVariables(beginsWithRVariables)

let beginsWithRHeroes = heroes.filter { beginsWithR.evaluateWithObject($0) }
// ["reaper", "roadhog"]

let beginsWithTVariables = ["letter": "t"]
let beginsWithT = template.predicateWithSubstitutionVariables(beginsWithTVariables)

let beginsWithTHeroes = heroes.filter { beginsWithT.evaluateWithObject($0) }
// ["tracer"]

```

第84章：NSBundle

第84.1节：通过路径获取Bundle

1. 使用路径定位Cocoa Bundle

要使用Cocoa获取特定路径下的bundle，调用NSBundle的bundleWithPath:类方法

```
NSBundle *myBundle;
// 获取一个可加载bundle的引用
myBundle = [NSBundle bundleWithPath:@"/Library/MyBundle.bundle"];
```

2. 使用路径定位Cocoa Foundation Bundle

要使用Core Foundation在特定路径获取捆绑包，调用CFBundleCreate函数，且必须使用CFURLRef类型。

```
CFURLRef bundleURL;
CFBundleRef myBundle;
// 从捆绑包路径的CFString表示创建一个CFURLRef。
bundleURL = CFURLCreateWithFileSystemPath(kCFAllocatorDefault,
CFSTR("/Library/MyBundle.bundle"), kCFURLPOSIXPathStyle, true);
// 使用URLRef创建捆绑包实例。
myBundle = CFBundleCreate(kCFAllocatorDefault, bundleURL);
// 现在可以释放URL了。
CFRelease(bundleURL);
// 使用捆绑包...
// 使用完成后释放捆绑包。
CFRelease(myBundle);
```

第 84.2 节：获取主捆绑包

1. 使用 Cocoa 获取主包的引用。

要获取Cocoa应用程序中的主包，调用NSBundle类的mainBundle类方法。

```
NSBundle *mainBundle;
// 获取应用程序的主包；
mainBundle = [NSBundle mainBundle];
```

2. 使用Core Foundation获取主包的引用。

使用CFBundleGetMainBundle函数来检索基于C的应用程序的主包。

```
CFBundleRef mainBundle;
// 获取应用程序的主包
mainBundle = CFBundleGetMainBundle();
```

Chapter 84: NSBundle

Section 84.1: Getting Bundle by Path

1. Locating a Cocoa bundle using its path

To obtain the bundle at a specific path using Cocoa, call the **bundleWithPath:** class method of the **NSBundle**

```
NSBundle *myBundle;
// obtain a reference to a loadable bundle
myBundle = [NSBundle bundleWithPath:@"/Library/MyBundle.bundle"];
```

2. Locating a Cocoa Foundation bundle using its Path

To obtain the bundle at a specific path using Core Foundation, call the **CFBundleCreate** function and must use **CFURLRef** type.

```
CFURLRef bundleURL;
CFBundleRef myBundle;
// Make a CFURLRef from the CFString representation of the bundle's path.
bundleURL = CFURLCreateWithFileSystemPath(kCFAllocatorDefault,
CFSTR("/Library/MyBundle.bundle"), kCFURLPOSIXPathStyle, true);
// Make a bundle instance using the URLRef.
myBundle = CFBundleCreate(kCFAllocatorDefault, bundleURL);
// You can release the URL now.
CFRelease(bundleURL);
// Use the bundle ...
// Release the bundle when done.
CFRelease(myBundle);
```

Section 84.2: Getting the Main Bundle

1. Getting a reference to the main bundle using Cocoa.

To get the main bundle in Cocoa application, call the **mainBundle** class method of the **NSBundle** class.

```
NSBundle *mainBundle;
// Get the main bundle for the app;
mainBundle = [NSBundle mainBundle];
```

2. Getting a reference to the main bundle using Core Foundation.

Use the **CFBundleGetMainBundle** function to retrieve the main bundle for your C-based application.

```
CFBundleRef mainBundle;
// Get the main bundle for the app
mainBundle = CFBundleGetMainBundle();
```

第85章：CAAnimation

第85.1节：将视图从一个位置动画到另一个位置

Objective-C

```
CABasicAnimation *animation = [CABasicAnimation animationWithKeyPath:@"position.x"];
animation.fromValue = @0;
animation.toValue = @320;
animation.duration = 1;

[_label.layer addAnimation:animation forKey:@"basic"];
```

Swift

```
let animation = CABasicAnimation(keyPath: "position.x")
animation.起始值 = NSNumber(value: 0.0)
animation.结束值 = NSNumber(value: 320.0)

_label.图层.添加动画(animation, forKey: "basic")
```

视图将在水平方向上从0移动到320。如果你想让视图垂直移动，只需将keyPath替换为：

```
"position.y"
```

第85.2节：动画视图 - 投掷

OBJECTIVE-C

```
CATransition* transition = [CATransition animation];
transition.开始进度 = 0;
transition.结束进度 = 1.0;
transition.类型 = @"flip";
transition.子类型 = @"fromLeft";
transition.持续时间 = 0.8;
transition.repeatCount = 5;
[_label.layer 添加动画:transition forKey:@"transition"];
```

SWIFT

```
var transition = CATransition()
transition.startProgress = 0
transition.endProgress = 1.0
transition.type = "flip"
transition.subtype = "fromLeft"
transition.duration = 0.8
transition.repeatCount = 5
label.layer.addAnimation(transition, forKey: "transition")
```

第85.3节：推入视图动画

Objective C

```
CATransition *animation = [CATransition animation];
[animation setSubtype:kCATransitionFromRight];//kCATransitionFromLeft
[animation setDuration:0.5];
[animation setType:kCATransitionPush];
[animation setTimingFunction:[CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut]];
[[yourView layer] addAnimation:animation forKey:@"SwitchToView1"];
```

Chapter 85: CAAnimation

Section 85.1: Animate a view from one position to another

Objective-C

```
CABasicAnimation *animation = [CABasicAnimation animationWithKeyPath:@"position.x"];
animation.fromValue = @0;
animation.toValue = @320;
animation.duration = 1;
```

```
[_label.layer addAnimation:animation forKey:@"basic"];
```

Swift

```
let animation = CABasicAnimation(keyPath: "position.x")
animation.fromValue = NSNumber(value: 0.0)
animation.toValue = NSNumber(value: 320.0)

_label.layer.addAnimation(animation, forKey: "basic")
```

The view will move from 0 to 320 horizontally. if you want to Move view to Vertically just replace keypath like this:

```
"position.y"
```

Section 85.2: Animate View - Toss

OBJECTIVE-C

```
CATransition* transition = [CATransition animation];
transition.startProgress = 0;
transition.endProgress = 1.0;
transition.type = @"flip";
transition.subtype = @"fromLeft";
transition.duration = 0.8;
transition.repeatCount = 5;
[_label.layer addAnimation:transition forKey:@"transition"];
```

SWIFT

```
var transition = CATransition()
transition.startProgress = 0
transition.endProgress = 1.0
transition.type = "flip"
transition.subtype = "fromLeft"
transition.duration = 0.8
transition.repeatCount = 5
label.layer.addAnimation(transition, forKey: "transition")
```

Section 85.3: Push View Animation

Objective C

```
CATransition *animation = [CATransition animation];
[animation setSubtype:kCATransitionFromRight];//kCATransitionFromLeft
[animation setDuration:0.5];
[animation setType:kCATransitionPush];
[animation setTimingFunction:[CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut]];
[[yourView layer] addAnimation:animation forKey:@"SwitchToView1"];
```

Swift

```
let animation = CATransition()
animation.subtype = kCATransitionFromRight//kCATransitionFromLeft
animation.duration = 0.5
animation.type = kCATransitionPush
animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionEaseInEaseOut)
yourView.layer.addAnimation(animation, forKey: "SwitchToView1")
```

第85.4节：旋转视图

```
CGRect boundingRect = CGRectMake(-150, -150, 300, 300);

CAKeyframeAnimation *orbit = [CAKeyframeAnimation animation];
orbit.keyPath = @"position";
orbit.path = CFAutorelease(CGPathCreateWithEllipseInRect(boundingRect, NULL));
orbit.duration = 4;
orbit.additive = YES;
orbit.repeatCount = HUGE_VALF;
orbit.calculationMode = kCAAnimationPaced;
orbit.rotationMode = kCAAnimationRotateAuto;

[_label.layer addAnimation:orbit forKey:@"orbit"];
```

第85.5节：摇动视图

Objective-C

```
CAKeyframeAnimation *animation = [CAKeyframeAnimation animationWithKeyPath:@"position.x"];
animation.values = @[@0, @10, @-10, @10, @0];
animation.keyTimes = @[@0, @(1 / 6.0), @(3 / 6.0), @(5 / 6.0), @1];
animation.duration = 0.4;
animation.additive = YES;
[_label.layer addAnimation:animation forKey:@"shake"];
```

Swift 3

```
let animation = CAKeyframeAnimation(keyPath: "position.x")
animation.values = [ 0, 10, -10, 10, 0 ]
    animation.keyTimes = [ 0, NSNumber(value: (1 / 6.0)), NSNumber(value: (3 / 6.0)), NSNumber(value:
(5 / 6.0)), 1 ]
animation.duration = 0.4
animation.isAdditive = true
label.layer.add(animation, forKey: "shake")
```

Swift

```
let animation = CATransition()
animation.subtype = kCATransitionFromRight//kCATransitionFromLeft
animation.duration = 0.5
animation.type = kCATransitionPush
animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionEaseInEaseOut)
yourView.layer.addAnimation(animation, forKey: "SwitchToView1")
```

Section 85.4: Revolve View

```
CGRect boundingRect = CGRectMake(-150, -150, 300, 300);

CAKeyframeAnimation *orbit = [CAKeyframeAnimation animation];
orbit.keyPath = @"position";
orbit.path = CFAutorelease(CGPathCreateWithEllipseInRect(boundingRect, NULL));
orbit.duration = 4;
orbit.additive = YES;
orbit.repeatCount = HUGE_VALF;
orbit.calculationMode = kCAAnimationPaced;
orbit.rotationMode = kCAAnimationRotateAuto;

[_label.layer addAnimation:orbit forKey:@"orbit"];
```

Section 85.5: Shake View

Objective-C

```
CAKeyframeAnimation *animation = [CAKeyframeAnimation animationWithKeyPath:@"position.x"];
animation.values = @[@0, @10, @-10, @10, @0];
animation.keyTimes = @[@0, @(1 / 6.0), @(3 / 6.0), @(5 / 6.0), @1];
animation.duration = 0.4;
animation.additive = YES;
[_label.layer addAnimation:animation forKey:@"shake"];
```

Swift 3

```
let animation = CAKeyframeAnimation(keyPath: "position.x")
animation.values = [ 0, 10, -10, 10, 0 ]
    animation.keyTimes = [ 0, NSNumber(value: (1 / 6.0)), NSNumber(value: (3 / 6.0)), NSNumber(value:
(5 / 6.0)), 1 ]
animation.duration = 0.4
animation.isAdditive = true
label.layer.add(animation, forKey: "shake")
```

第86章：并发

调度块中的代码将在哪个队列中运行。队列 (queue) 类似于（但不完全等同于）线程；不同队列中的代码可以并行运行。使用 `dispatch_get_main_queue` 可以获取主线程的队列。要创建一个新队列（这也会创建一个新线程），使用 `dispatch_queue_create("QUEUE_NAME", DISPATCH_QUEUE_CONCURRENT)`。第一个参数是队列的名称，如果在块仍在运行时暂停调试器，该名称会显示在调试器中。第二个参数除非你想在多个 `dispatch_async` 或 `dispatch_sync` 调用中使用同一个队列，否则无关紧要。它描述了当另一个块被放入同一队列时会发生什么；`DISPATCH_QUEUE_CONCURRENT` 会使两个块同时运行，而 `DISPATCH_QUEUE_SERIAL` 会让第二个块等待第一个块完成后再运行。

此块中的代码将在队列 `queue` 中运行；将你想在独立队列上运行的代码放在这里。一个有用的提示：如果你在 Xcode 中编写此代码，并且块参数周围有蓝色轮廓，双击该参数，Xcode 会自动生成一个空块（这适用于任何函数或方法中的所有块参数）

相关主题：Grand Central Dispatch (GCD)

第86.1节：调度组 - 等待其他线程完成

```
dispatch_group_t preapreWaitingGroup = dispatch_group_create();

dispatch_group_enter(preapreWaitingGroup);
[self doAsynchronousTaskWithComplete:^(id someResults, NSError *error) {
    // 通知该任务已完成。
    dispatch_group_leave(preapreWaitingGroup);
}]

dispatch_group_enter(preapreWaitingGroup);
[self 执行其他异步任务并完成:^(id someResults, NSError *error) {
    dispatch_group_leave(preapreWaitingGroup);
}]

dispatch_group_notify(preapreWaitingGroup, dispatch_get_main_queue(), ^{
    // 当上述所有线程完成并调用 dispatch_group_leave 后，此代码块将被执行
    NSLog(@"准备完成。我准备好了");
});
```

更新 1. Swift 3 版本。

```
let prepareGroup = DispatchGroup()
prepareGroup.enter()
doAsynchronousTaskWithComplete() { (someResults, error) in
    // 通知该任务已完成。
    prepareGroup.leave()
}

prepareGroup.enter()
doOtherAsynchronousTaskWithComplete() { (someResults, error) in
    // 通知该任务已完成。
    prepareGroup.leave()
}

prepareGroup.notify(queue: DispatchQueue.main) {
    // 当上述所有线程完成后，此代码块将被执行并调用 dispatch_group_leave
    print("准备完成。我准备好了")
}
```

Chapter 86: Concurrency

The queue that the code in the dispatch block will run in. A queue is like (but not exactly the same as) a thread; code in different queues can run in parallel. Use `dispatch_get_main_queue` to get the queue for the main thread. To create a new queue, which in turn creates a new thread, use `dispatch_queue_create("QUEUE_NAME", DISPATCH_QUEUE_CONCURRENT)`. The first parameter is the name of the queue, which is displayed in the debugger if you pause while the block is still running. The second parameter doesn't matter unless you want to use the same queue for multiple `dispatch_async` or `dispatch_sync` calls. It describes what happens when another block is put on the same queue; `DISPATCH_QUEUE_CONCURRENT` will cause both blocks to run at the same time, while `DISPATCH_QUEUE_SERIAL` will make the second block wait for the first block to finish.

Code in this block will run in the queue `queue`; put code you want to run on the separate queue here. A helpful tip: If you're writing this in Xcode and the block argument has the blue outline around it, double click on the argument and Xcode will automatically make an empty block (this applies to all block arguments in any function or method).

Related topic: Grand Central Dispatch

Section 86.1: Dispatch group - waiting for other threads completed

```
dispatch_group_t preapreWaitingGroup = dispatch_group_create();

dispatch_group_enter(preapreWaitingGroup);
[self doAsynchronousTaskWithComplete:^(id someResults, NSError *error) {
    // Notify that this task has been completed.
    dispatch_group_leave(preapreWaitingGroup);
}]

dispatch_group_enter(preapreWaitingGroup);
[self doOtherAsynchronousTaskWithComplete:^(id someResults, NSError *error) {
    dispatch_group_leave(preapreWaitingGroup);
}]

dispatch_group_notify(preapreWaitingGroup, dispatch_get_main_queue(), ^{
    // This block will be executed once all above threads completed and call dispatch_group_leave
    NSLog(@"Prepare completed. I'm readyyyy");
});
```

Update 1. Swift 3 version.

```
let prepareGroup = DispatchGroup()
prepareGroup.enter()
doAsynchronousTaskWithComplete() { (someResults, error) in
    // Notify that this task has been completed.
    prepareGroup.leave()
}

prepareGroup.enter()
doOtherAsynchronousTaskWithComplete() { (someResults, error) in
    // Notify that this task has been completed.
    prepareGroup.leave()
}

prepareGroup.notify(queue: DispatchQueue.main) {
    // This block will be executed once all above threads completed and call dispatch_group_leave
    print("Prepare completed. I'm readyyyy")
}
```

第86.2节：在主线程上执行

在异步执行任务时，通常需要确保某段代码在主线程上运行。例如，你可能想异步调用REST API，但将结果放入屏幕上的UILabel中。在更新UILabel之前，必须确保代码在主线程上运行：

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    //执行耗时任务
    //...
    
    //现在在更新UI之前，确保回到主线程
    dispatch_async(dispatch_get_main_queue(), ^{
        label.text = //....
    });
}
```

每当你更新屏幕上的视图时，务必确保在主线程上执行，否则可能会出现未定义的行为。

第86.3节：并发运行代码——在运行其他代码时运行代码

假设你想在执行某个操作时（在本例中是记录“Foo”），同时做另一件事（记录“Bar”）。通常，如果不使用并发，其中一个操作会被完全执行，另一个操作只有在前一个完全结束后才会运行。但使用并发，你可以让两个操作同时运行：

```
dispatch_async(dispatch_queue_create("Foo", DISPATCH_QUEUE_CONCURRENT), ^{
    for (int i = 0; i < 100; i++) {
        NSLog(@"Foo");
        usleep(100000);
    }
});

for (int i = 0; i < 100; i++) {
    NSLog(@"Bar");
    usleep(50000);
}
```

这将打印“Foo”100次，每次打印后暂停100毫秒，但所有操作都将在一个单独的线程上进行。虽然正在打印Foo，“Bar”也会以50毫秒的间隔同时打印。理想情况下，你应该看到“Foo”和“Bar”混合在一起的输出

Section 86.2: Executing on the main thread

When performing tasks asynchronously there typically becomes a need to ensure a piece of code is run on the main thread. For example you may want to hit a REST API asynchronously, but put the result in a UILabel on the screen. Before updating the UILabel you must ensure that your code is run on the main thread:

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    //Perform expensive tasks
    //...
    
    //Now before updating the UI, ensure we are back on the main thread
    dispatch_async(dispatch_get_main_queue(), ^{
        label.text = //....
    });
}
```

Whenever you update views on the screen, always ensure you are doing so on the main thread, otherwise undefined behavior could occur.

Section 86.3: Running code concurrently -- Running code while running other code

Say you want to perform in action (in this case, logging "Foo"), while doing something else (logging "Bar"). Normally, if you don't use concurrency, one of these actions is going to be fully executed, and the other run will only after it's completely finished. But with concurrency, you can make both actions run at the same time:

```
dispatch_async(dispatch_queue_create("Foo", DISPATCH_QUEUE_CONCURRENT), ^{
    for (int i = 0; i < 100; i++) {
        NSLog(@"Foo");
        usleep(100000);
    }
});

for (int i = 0; i < 100; i++) {
    NSLog(@"Bar");
    usleep(50000);
}
```

This will log "Foo" 100 times, pausing for 100ms each time it logs, but it will do all this on a separate thread. While Foo is being logged, "Bar" will also be logged in 50ms intervals, at the same time. You should ideally see an output with "Foo"s and "Bars" mixed together

第87章：CAGradientLayer

参数

	详细信息
颜色	一个包含CGColorRef对象的数组，定义每个渐变停靠点的颜色。可动画。
位置	一个可选的包含NSNumber对象的数组，定义每个渐变停靠点的位置。可动画。
终点	在图层坐标空间中绘制渐变的终点。可动画。
起点	在图层坐标空间中绘制渐变的起点。可动画。
类型	图层绘制的渐变样式。默认为kCAGradientLayerAxial。

第87.1节：创建CAGradientLayer

```
// 用于承载CAGradientLayer的视图。  
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))  
  
// 初始化渐变图层。  
let gradientLayer: CAGradientLayer = CAGradientLayer()  
  
// 设置渐变图层的尺寸。  
gradientLayer.frame = view.bounds  
  
// 渐变顶部的颜色。  
let topColor: CGColor = UIColor.red.cgColor  
  
// 渐变底部的颜色。  
let bottomColor: CGColor = UIColor.yellow.cgColor  
  
// 设置颜色数组。  
gradientLayer.colors = [topColor, bottomColor]  
  
// 设置颜色位置。  
gradientLayer.locations = [0.0, 1.0]  
  
// 将渐变图层插入视图的图层层级中。  
view.layer.insertSublayer(gradientLayer, at: 0)
```

结果：



Chapter 87: CAGradientLayer

Parameter

	Details
color	An array of <code>CGColorRef</code> objects defining the color of each gradient stop. Animatable.
locations	An optional array of <code>NSNumber</code> objects defining the location of each gradient stop. Animatable.
endPoint	The end point of the gradient when drawn in the layer's coordinate space. Animatable.
startPoint	The start point of the gradient when drawn in the layer's coordinate space. Animatable.
type	Style of gradient drawn by the layer. Defaults to <code>kCAGradientLayerAxial</code> .

Section 87.1: Creating a CAGradientLayer

```
// View to hold the CAGradientLayer.  
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))  
  
// Initialize gradient layer.  
let gradientLayer: CAGradientLayer = CAGradientLayer()  
  
// Set frame of gradient layer.  
gradientLayer.frame = view.bounds  
  
// Color at the top of the gradient.  
let topColor: CGColor = UIColor.red.cgColor  
  
// Color at the bottom of the gradient.  
let bottomColor: CGColor = UIColor.yellow.cgColor  
  
// Set colors.  
gradientLayer.colors = [topColor, bottomColor]  
  
// Set locations of the colors.  
gradientLayer.locations = [0.0, 1.0]  
  
// Insert gradient layer into view's layer hierarchy.  
view.layer.insertSublayer(gradientLayer, at: 0)
```

Result:



第87.2节：在CAGradientLayer中实现颜色变化动画

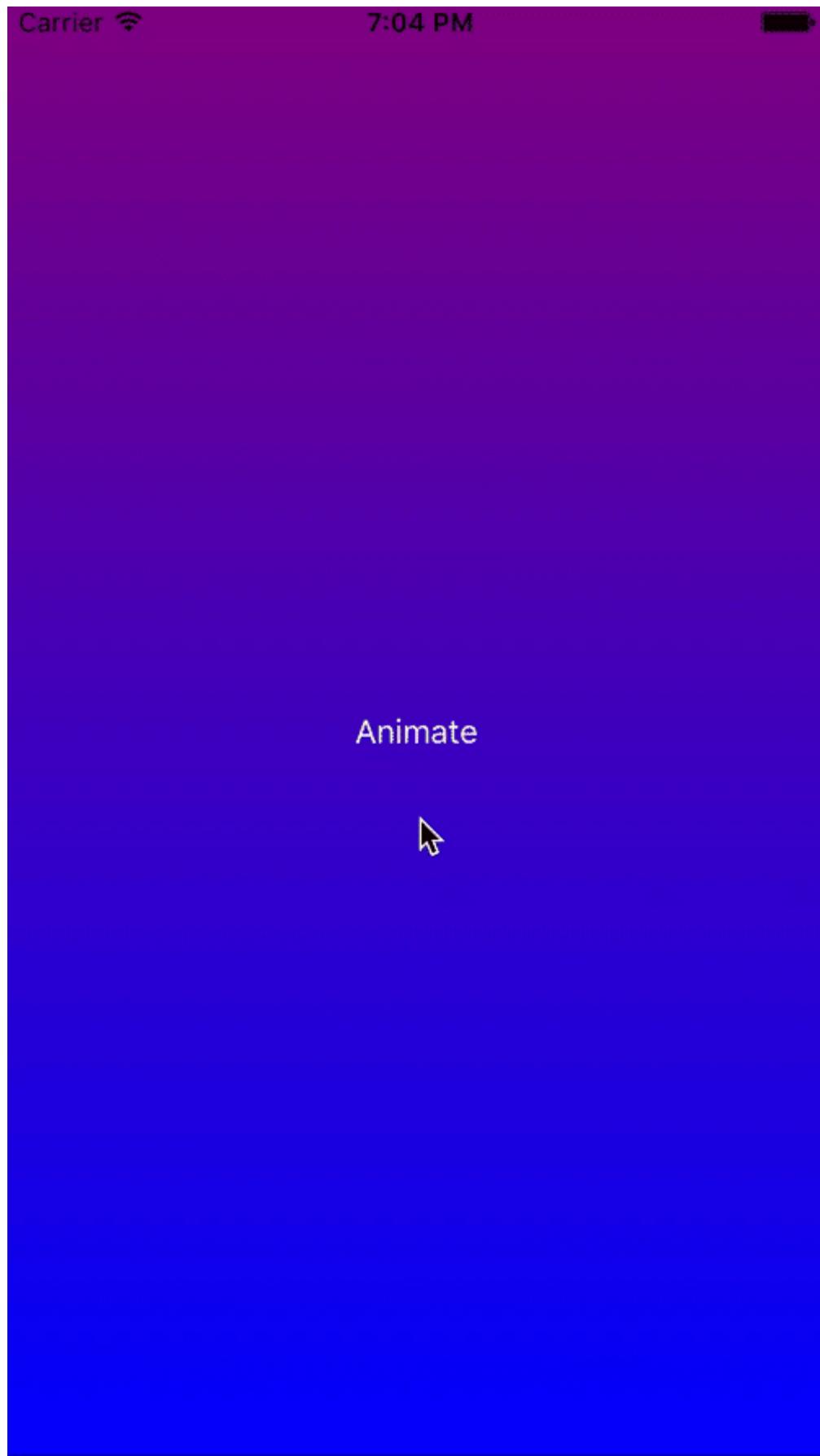
```
// 获取渐变的当前颜色。  
let oldColors = self.gradientLayer.colors  
  
// 定义渐变的新颜色。  
let newColors = [UIColor.red.cgColor, UIColor.yellow.cgColor]  
  
// 设置渐变的新颜色。  
self.gradientLayer.colors = newColors  
  
// 初始化用于更改渐变颜色的新动画。  
let animation: CABasicAnimation = CABasicAnimation(keyPath: "colors")  
  
// 设置当前颜色值。  
animation.fromValue = oldColors  
  
// 设置新颜色值。  
animation.toValue = newColors  
  
// 设置动画持续时间。  
animation.duration = 0.3  
  
// 设置动画完成后移除。  
animation.isRemovedOnCompletion = true  
  
// 设置接收器在动画完成后保持可见的最终状态。  
animation.fillMode = kCAFillModeForwards  
  
// 设置线性节奏，使动画在其持续时间内均匀发生。  
animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionLinear)  
  
// 设置动画的代理。  
animation.delegate = self  
  
// 添加动画。  
self.gradientLayer.addAnimation(animation, forKey: "animateGradientColorChange")
```

结果：

Section 87.2: Animating a color change in CAGradientLayer

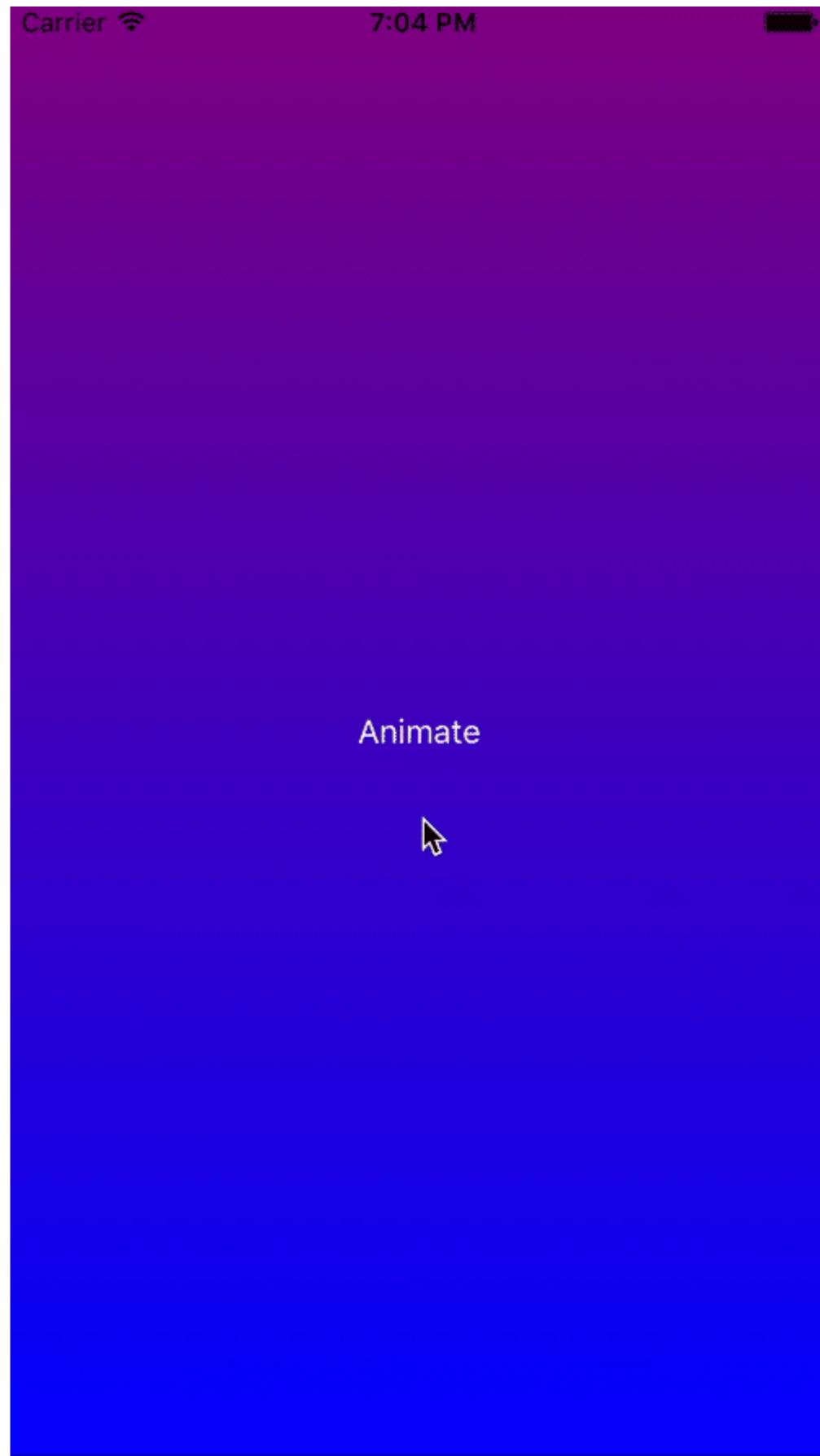
```
// Get the current colors of the gradient.  
let oldColors = self.gradientLayer.colors  
  
// Define the new colors for the gradient.  
let newColors = [UIColor.red.cgColor, UIColor.yellow.cgColor]  
  
// Set the new colors of the gradient.  
self.gradientLayer.colors = newColors  
  
// Initialize new animation for changing the colors of the gradient.  
let animation: CABasicAnimation = CABasicAnimation(keyPath: "colors")  
  
// Set current color value.  
animation.fromValue = oldColors  
  
// Set new color value.  
animation.toValue = newColors  
  
// Set duration of animation.  
animation.duration = 0.3  
  
// Set animation to remove once its completed.  
animation.isRemovedOnCompletion = true  
  
// Set receiver to remain visible in its final state when the animation is completed.  
animation.fillMode = kCAFillModeForwards  
  
// Set linear pacing, which causes an animation to occur evenly over its duration.  
animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionLinear)  
  
// Set delegate of animation.  
animation.delegate = self  
  
// Add the animation.  
self.gradientLayer.addAnimation(animation, forKey: "animateGradientColorChange")
```

Result :



第87.3节：创建水平CAGradientLayer

```
// 用于承载CAGradientLayer的视图。  
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))  
  
// 初始化渐变图层。  
// Initialize gradient layer.
```



Section 87.3: Creating a horizontal CAGradientLayer

```
// View to hold the CAGradientLayer.  
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))  
  
// Initialize gradient layer.
```

```

let gradientLayer: CAGradientLayer = CAGradientLayer()

// 设置渐变图层的尺寸。
gradientLayer.frame = view.bounds

// 渐变顶部的颜色。
let topColor: CGColor = UIColor.redColor().CGColor

// 渐变底部的颜色。
let bottomColor: CGColor = UIColor.yellowColor().CGColor

// 设置颜色。
gradientLayer.colors = [topColor, bottomColor]

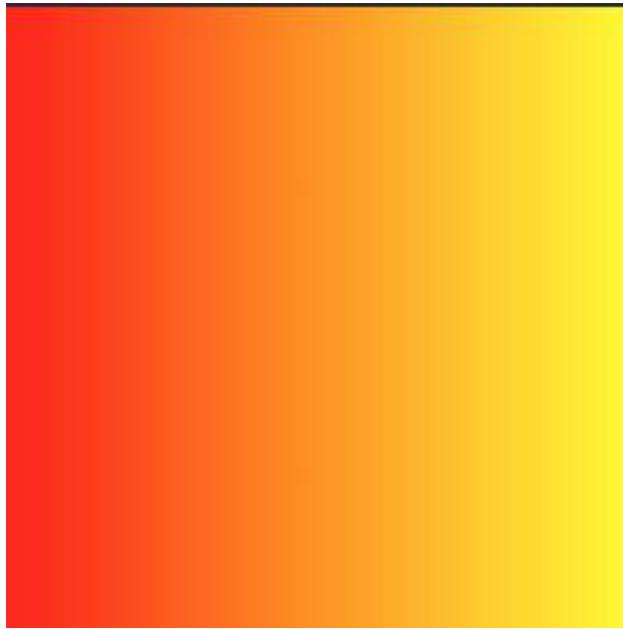
// 设置颜色数组。
gradientLayer.startPoint = CGPointMake(x: 0.0, y: 0.5)

// 设置起点。
gradientLayer.endPoint = CGPointMake(x: 1.0, y: 0.5)

// 设置终点。
view.layer.insertSublayer(gradientLayer, atIndex: 0)

```

结果：



第87.4节：创建一个带有多种颜色的水平CAGradientLayer

```

// 用于承载CAGradientLayer的视图。
let view: UIView = UIView(frame: CGRectMake(x: 0, y: 0, width: 320, height: 320))

// 初始化渐变图层。
let gradientLayer: CAGradientLayer = CAGradientLayer()

// 设置渐变图层的尺寸。
gradientLayer.frame = view.bounds

// 渐变顶部的颜色。
let topColor: CGColor = UIColor.greenColor().CGColor

// 渐变中间的颜色。

```

```

let gradientLayer: CAGradientLayer = CAGradientLayer()

// Set frame of gradient layer.
gradientLayer.frame = view.bounds

// Color at the top of the gradient.
let topColor: CGColor = UIColor.redColor().CGColor

// Color at the bottom of the gradient.
let bottomColor: CGColor = UIColor.yellowColor().CGColor

// Set colors.
gradientLayer.colors = [topColor, bottomColor]

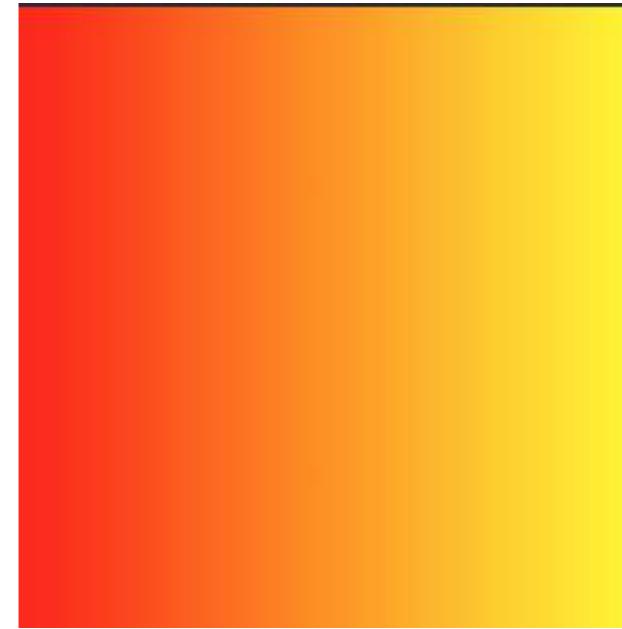
// Set start point.
gradientLayer.startPoint = CGPointMake(x: 0.0, y: 0.5)

// Set end point.
gradientLayer.endPoint = CGPointMake(x: 1.0, y: 0.5)

// Insert gradient layer into view's layer heirarchy.
view.layer.insertSublayer(gradientLayer, atIndex: 0)

```

Result:



Section 87.4: Creating a horizontal CAGradientLayer with multiple colors

```

// View to hold the CAGradientLayer.
let view: UIView = UIView(frame: CGRectMake(x: 0, y: 0, width: 320, height: 320))

// Initialize gradient layer.
let gradientLayer: CAGradientLayer = CAGradientLayer()

// Set frame of gradient layer.
gradientLayer.frame = view.bounds

// Color at the top of the gradient.
let topColor: CGColor = UIColor.greenColor().CGColor

// Color at the middle of the gradient.

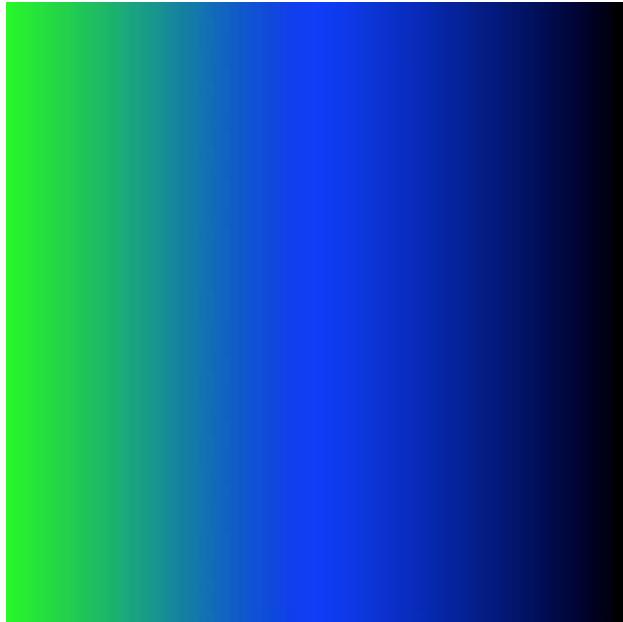
```

```

let middleColor: CGColor = UIColor.blueColor().CGColor
// 渐变底部的颜色。
let bottomColor: CGColor = UIColor.blackColor().CGColor
// 设置颜色。
gradientLayer.colors = [topColor, middleColor, bottomColor]
// 设置起点。
gradientLayer.startPoint = CGPointMake(x: 0.0, y: 0.5)
// 设置终点。
gradientLayer.endPoint = CGPointMake(x: 1.0, y: 0.5)
// 设置终点。
view.layer.insertSublayer(gradientLayer, atIndex: 0)

```

结果：



第87.5节：创建具有多种颜色的CGGradientLayer

```

// 用于承载CAGradientLayer的视图。
let view: UIView = UIView(frame: CGRectMake(x: 0, y: 0, width: 320, height: 320))

// 初始化渐变图层。
let gradientLayer: CAGradientLayer = CAGradientLayer()

// 设置渐变图层的尺寸。
gradientLayer.frame = view.bounds

// 渐变顶部的颜色。
let topColor: CGColor = UIColor.blue.cgColor

// 渐变的顶部颜色。
let middleColor: CGColor = UIColor.yellow.cgColor

// 渐变的中间颜色。
let bottomColor: CGColor = UIColor.green.cgColor

// 设置颜色。
gradientLayer.colors = [topColor, middleColor, bottomColor]

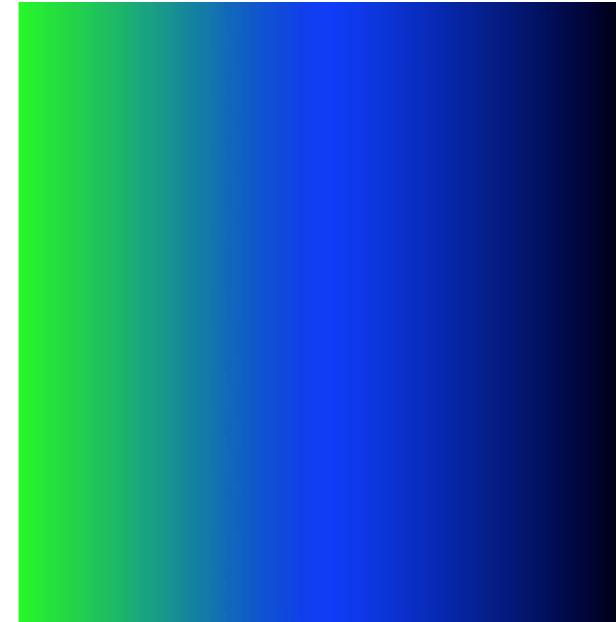
```

```

let middleColor: CGColor = UIColor.blueColor().CGColor
// Color at the bottom of the gradient.
let bottomColor: CGColor = UIColor.blackColor().CGColor
// Set colors.
gradientLayer.colors = [topColor, middleColor, bottomColor]
// Set start point.
gradientLayer.startPoint = CGPointMake(x: 0.0, y: 0.5)
// Set end point.
gradientLayer.endPoint = CGPointMake(x: 1.0, y: 0.5)
// Insert gradient layer into view's layer hierarchy.
view.layer.insertSublayer(gradientLayer, atIndex: 0)

```

Result :



Section 87.5: Creating a CGGradientLayer with multiple colors

```

// View to hold the CAGradientLayer.
let view: UIView = UIView(frame: CGRectMake(x: 0, y: 0, width: 320, height: 320))

// Initialize gradient layer.
let gradientLayer: CAGradientLayer = CAGradientLayer()

// Set frame of gradient layer.
gradientLayer.frame = view.bounds

// Color at the top of the gradient.
let topColor: CGColor = UIColor.blue.cgColor

// Color at the middle of the gradient.
let middleColor: CGColor = UIColor.yellow.cgColor

// Color at the bottom of the gradient.
let bottomColor: CGColor = UIColor.green.cgColor

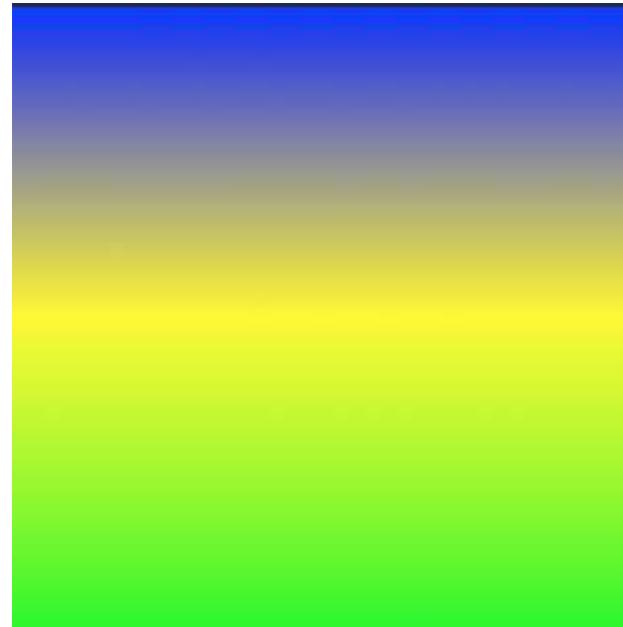
// Set colors.
gradientLayer.colors = [topColor, middleColor, bottomColor]

```

```
// 设置颜色的位置。  
gradientLayer.locations = [0.0, 0.5, 1.0]
```

```
// 将渐变图层插入视图的图层层级中。  
view.layer.insertSublayer(gradientLayer, at: 0)
```

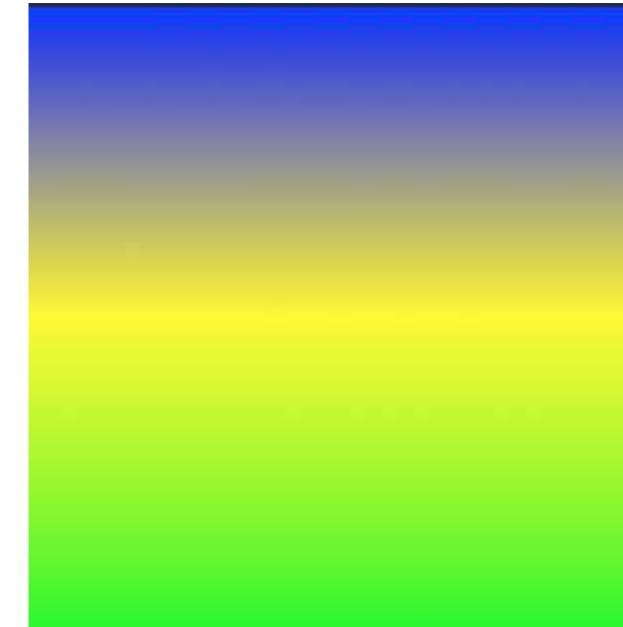
结果：



```
// Set locations of the colors.  
gradientLayer.locations = [0.0, 0.5, 1.0]
```

```
// Insert gradient layer into view's layer heirarchy.  
view.layer.insertSublayer(gradientLayer, at: 0)
```

Result:



第88章：Safari服务

第88.1节：使用SafariViewController打开URL

别忘了先导入必要的框架。

```
import SafariServices  
//Objective-C  
@import SafariServices;
```

实例化一个SafariViewController对象。

```
let safariVC = SFSafariViewController(URL: URL(string: "your_url")!)  
//Objective-C  
@import SafariServices;  
NSURL *URL = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.google.com"]];  
SFSafariViewController *sfvc = [[SFSafariViewController alloc] initWithURL:URL];
```

你也可以选择在加载完成后让SafariViewController进入阅读模式（如果可能）。

```
let safariVC = SFSafariViewController(URL: URL(string: "your_url")!, entersReaderIfAvailable: true)  
//Objective-C  
NSURL *URL = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.google.com"]];  
SFSafariViewController *sfvc = [[SFSafariViewController alloc] initWithURL:URL  
entersReaderIfAvailable:YES];
```

展示视图控制器。

```
present(safariVC, animated: true, completion: nil)  
//Objective-C  
[self presentViewController:sfvc animated:YES completion:nil];
```

第88.2节：实现SFSafariViewControllerDelegate

你应该实现SFSafariViewControllerDelegate，这样当用户点击SafariViewController上的完成按钮时，你的类会收到通知，并且你可以关闭它。

首先声明你的类实现该协议。

```
class MyClass: SFSafariViewControllerDelegate {  
}
```

实现代理方法以接收关闭通知。

```
func safariViewControllerDidFinish(controller: SFSafariViewController) {  
    // 完成时关闭SafariViewController  
    controller.dismissViewControllerAnimated(true, completion: nil)  
}
```

别忘了将你的类设置为SafariViewController的代理。

```
let safariVC = SFSafariViewController(URL: yourURL)  
safariVC.delegate = self
```

Chapter 88: Safari Services

Section 88.1: Open a URL with SafariViewController

Don't forget to import the necessary framework first.

```
import SafariServices  
//Objective-C  
@import SafariServices;
```

Instantiate a SafariViewController instance.

```
let safariVC = SFSafariViewController(URL: URL(string: "your_url")!)  
//Objective-C  
@import SafariServices;  
NSURL *URL = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.google.com"]];  
SFSafariViewController *sfvc = [[SFSafariViewController alloc] initWithURL:URL];
```

Optionally you can also tell SafariViewController to enter reading mode if possible once it's done loading.

```
let safariVC = SFSafariViewController(URL: URL(string: "your_url")!, entersReaderIfAvailable: true)  
//Objective-C  
NSURL *URL = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.google.com"]];  
SFSafariViewController *sfvc = [[SFSafariViewController alloc] initWithURL:URL  
entersReaderIfAvailable:YES];
```

Present the view controller.

```
present(safariVC, animated: true, completion: nil)  
//Objective-C  
[self presentViewController:sfvc animated:YES completion:nil];
```

Section 88.2: Implement SFSafariViewControllerDelegate

You should implement SFSafariViewControllerDelegate so that your class is notified when the user hits the Done button on the SafariViewController and you can dismiss it as well.

First declare your class to implement the protocol.

```
class MyClass: SFSafariViewControllerDelegate {  
}
```

Implement the delegate method to be notified on dismissal.

```
func safariViewControllerDidFinish(controller: SFSafariViewController) {  
    // Dismiss the SafariViewController when done  
    controller.dismissViewControllerAnimated(true, completion: nil)  
}
```

Don't forget to set your class as the SafariViewController's delegate.

```
let safariVC = SFSafariViewController(URL: yourURL)  
safariVC.delegate = self
```

您可以实现的其他代理方法有：

```
// 当初始 URL 加载完成时调用。  
safariViewController(_ controller: SFSafariViewController, didCompleteInitialLoad  
didLoadSuccessfully: Bool) { }  
  
// 当用户点击操作按钮时调用。  
safariViewController(_ controller: SFSafariViewController, activityItemsFor URL: URL, title:  
String?) -> [UIActivity] { }
```

第 88.3 节：向 Safari 阅读列表添加项目

您可以通过调用 SSReadingList 单例的 addItem 方法，将项目添加到用户的 Safari 阅读列表中。

```
let readingList = SSReadingList.default()  
readingList?.addItem(with: yourURL, title: "optional title", previewText: "optional preview text")
```

默认的阅读列表可以是nil，如果不允许访问阅读列表。

此外，你可以通过调用supportsURL来检查阅读列表是否支持某个URL。

```
SSReadingList.default().supportsURL(URL(string: "https://example.com")!)
```

这将返回true或false，表示Safari阅读列表是否支持给定的URL。例如，可以用来判断是否显示添加URL到阅读列表的按钮。

Additional delegate methods you can implement are:

```
// Called when the initial URL load is complete.  
safariViewController(_ controller: SFSafariViewController, didCompleteInitialLoad  
didLoadSuccessfully: Bool) { }  
  
// Called when the user taps an Action button.  
safariViewController(_ controller: SFSafariViewController, activityItemsFor URL: URL, title:  
String?) -> [UIActivity] { }
```

Section 88.3: Add Items to Safari Reading List

You can add items to a user's Reading List in Safari by calling the addItem method on the SSReadingList singleton.

```
let readingList = SSReadingList.default()  
readingList?.addItem(with: yourURL, title: "optional title", previewText: "optional preview text")
```

The default Reading List can be nil if access to the Reading List is not permitted.

Additionally you can check if the Reading List supports a URL by calling supportsURL.

```
SSReadingList.default().supportsURL(URL(string: "https://example.com")!)
```

This will return either true or false indicating if the given URL is supported by Safari Reading List. Use this for example to determine whether to show a button to add a URL to the Reading List.

第89章：CALayer

第89.1节：向CALayer添加变换（平移、旋转、缩放）

基础知识

你可以对图层进行多种不同的变换，但基本的有

- 平移（移动）
- 缩放
- 旋转



要对CALayer进行变换，您需要将图层的transform属性设置为CATransform3D类型。例如，要平移一个图层，可以这样写：

```
myLayer.transform = CATransform3DMakeTranslation(20, 30, 0)
```

单词Make用于创建初始变换的名称中：CATransform3D**Make**Translation。后续应用的变换则省略了Make。例如，下面是先旋转再平移的操作：

```
let rotation = CATransform3DMakeRotation(CGFloat(30.0 * M_PI / 180.0), 20, 20, 0)
myLayer.transform = CATransform3DTranslate(rotation, 20, 30, 0)
```

现在我们已经了解了如何创建变换的基础，接下来看看每种变换的示例。不过，首先我会展示如何搭建项目，以防你也想试一试。

设置

在以下示例中，我创建了一个单视图应用，并在故事板中添加了一个背景为浅蓝色的UIView。我用以下代码将视图连接到视图控制器：

```
import UIKit

class ViewController: UIViewController {

    var myLayer = CATextLayer()
    @IBOutlet weak var myView: UIView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // 设置子图层
        addSubLayer()
    }
}
```

Chapter 89: CALayer

Section 89.1: Adding Transforms to a CALayer (translate, rotate, scale)

Basics

There are a number of different transforms you can do on a layer, but the basic ones are

- translate (move)
- scale
- rotate



To do transforms on a CALayer, you set the layer's transform property to a CATransform3D type. For example, to translate a layer, you would do something like this:

```
myLayer.transform = CATransform3DMakeTranslation(20, 30, 0)
```

The word Make is used in the name for creating the initial transform: CATransform3D**Make**Translation. Subsequent transforms that are applied omit the Make. See, for example, this rotation followed by a translation:

```
let rotation = CATransform3DMakeRotation(CGFloat(30.0 * M_PI / 180.0), 20, 20, 0)
myLayer.transform = CATransform3DTranslate(rotation, 20, 30, 0)
```

Now that we have the basis of how to make a transform, let's look at some examples of how to do each one. First, though, I'll show how I set up the project in case you want to play around with it, too.

Setup

For the following examples I set up a Single View Application and added a `UIView` with a light blue background to the storyboard. I hooked up the view to the view controller with the following code:

```
import UIKit

class ViewController: UIViewController {

    var myLayer = CATextLayer()
    @IBOutlet weak var myView: UIView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // setup the sublayer
        addSubLayer()
    }
}
```

```

    // 执行变换
    transformExample()

}

func addSubLayer() {
myLayer.frame = CGRect(x: 0, y: 0, width: 100, height: 40)
    myLayer.backgroundColor = UIColor.blueColor().CGColor
myLayer.string = "Hello"
    myView.layer.addSublayer(myLayer)
}

//***** 用下面的示例替换此函数 *****
func transformExample() {

    // 在这里添加变换代码 ...
}

}

```

有许多不同种类的CALayer，但我选择使用CATextLayer，这样变换在视觉上会更清晰。

翻译

平移变换会移动图层。基本语法是

```
CATransform3DMakeTranslation(tx: CGFloat, ty: CGFloat, tz: CGFloat)
```

其中 tx 是 x 坐标的变换，ty 是 y 坐标的变换，tz 是 z 坐标的变换。

示例



在 iOS 中，坐标系的原点位于左上角，因此如果我们想将图层向右移动 90 点并向右下移动 50 点，可以这样做：

```
myLayer.transform = CATransform3DMakeTranslation(90, 50, 0)
```

注意事项

- 请记住，你可以将此代码粘贴到上面项目代码中的 transformExample() 方法中。
- 由于这里我们只处理二维，tz 设置为 0。
- 上图中的红线从原始位置的中心延伸到新位置的中心。这是因为变换是相对于锚点进行的，而锚点默认位于图层的中心。

```

    // do the transform
    transformExample()

}

func addSubLayer() {
myLayer.frame = CGRect(x: 0, y: 0, width: 100, height: 40)
    myLayer.backgroundColor = UIColor.blueColor().CGColor
myLayer.string = "Hello"
    myView.layer.addSublayer(myLayer)
}

//***** Replace this function with the examples below *****
func transformExample() {

    // add transform code here ...
}

}

```

[There are many different kinds of CALayer](#), but I chose to use CATextLayer so that the transforms will be more clear visually.

Translate

The translation transform moves the layer. The basic syntax is

```
CATransform3DMakeTranslation(tx: CGFloat, ty: CGFloat, tz: CGFloat)
```

where tx is the change in the x coordinates, ty is the change in y, and tz is the change in z.

Example



In iOS the origin of the coordinate system is in the top left, so if we wanted to move the layer 90 points to the right and 50 points down, we would do the following:

```
myLayer.transform = CATransform3DMakeTranslation(90, 50, 0)
```

Notes

- Remember that you can paste this into the transformExample() method in the project code above.
- Since we are just going to deal with two dimensions here, tz is set to 0.
- The red line in the image above goes from the center of the original location to the center of the new location. That's because transforms are done in relation to the anchor point and the anchor point by default is in the center of the layer.

缩放

缩放变换会拉伸或压缩图层。基本语法是

```
CATransform3DMakeScale(sx: CGFloat, sy: CGFloat, sz: CGFloat)
```

其中 sx、sy 和 sz 分别是对 x、y 和 z 坐标进行缩放（乘法）的数值。

示例



如果我们想将宽度缩小一半，高度放大三倍，可以这样写

```
myLayer.transform = CATransform3DMakeScale(0.5, 3.0, 1.0)
```

注意事项

- 由于我们只在二维空间中操作，所以对 z 坐标乘以 1.0 以保持其不变。
- 上图中的红点表示锚点。注意缩放是相对于锚点进行的，也就是说，所有内容都是向锚点方向拉伸或远离锚点拉伸。

旋转

旋转变换围绕锚点（默认情况下为图层中心）旋转图层。基本语法是

```
CATransform3DMakeRotation(angle: CGFloat, x: CGFloat, y: CGFloat, z: CGFloat)
```

其中angle是图层应旋转的弧度角，x、y和z是旋转的轴。将某个轴设置为0则取消该轴的旋转。

示例

Scale

The scale transform stretches or squishes the layer. The basic syntax is

```
CATransform3DMakeScale(sx: CGFloat, sy: CGFloat, sz: CGFloat)
```

where sx, sy, and sz are the numbers by which to scale (multiply) the x, y, and z coordinates respectively.

Example



If we wanted to half the width and triple the height, we would do the following

```
myLayer.transform = CATransform3DMakeScale(0.5, 3.0, 1.0)
```

Notes

- Since we are only working in two dimensions, we just multiply the z coordinates by 1.0 to leave them unaffected.
- The red dot in the image above represents the anchor point. Notice how the scaling is done in relation to the anchor point. That is, everything is either stretched toward or away from the anchor point.

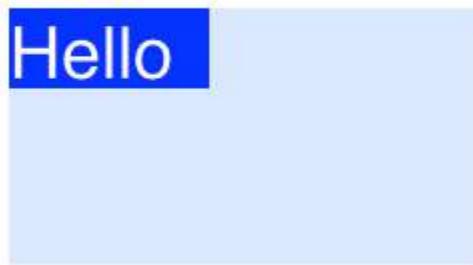
Rotate

The rotation transform rotates the layer around the anchor point (the center of the layer by default). The basic syntax is

```
CATransform3DMakeRotation(angle: CGFloat, x: CGFloat, y: CGFloat, z: CGFloat)
```

where angle is the angle in radians that the layer should be rotated and x, y, and z are the axes about which to rotate. Setting an axis to 0 cancels a rotation around that particular axis.

Example



如果我们想顺时针旋转图层30度，可以这样做：

```
let degrees = 30.0  
let radians = CGFloat(degrees * M_PI / 180)  
myLayer.transform = CATransform3DMakeRotation(radians, 0.0, 0.0, 1.0)
```

注意事项

- 由于我们在二维空间中工作，只想让xy平面绕z轴旋转。因此我们将 x和 y设置为0.0，将 z设置为1.0。
- 这使图层顺时针旋转。我们也可以通过将 z设置为-1.0来实现逆时针旋转。
- 红点显示了锚点的位置。旋转是围绕锚点进行的。

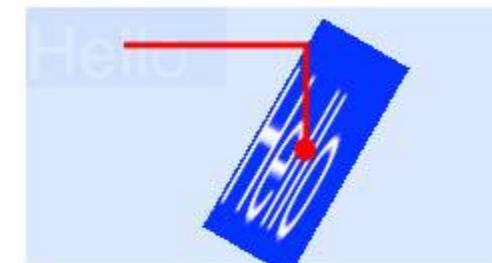
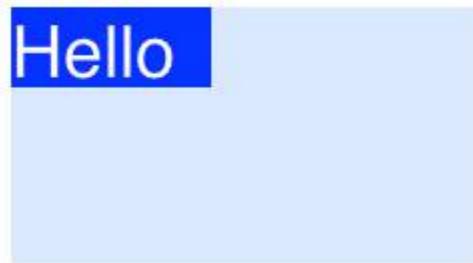
多重变换

为了组合多个变换，我们可以像这样使用连接（concatenation）

```
CATransform3DConcat(a: CATransform3D, b: CATransform3D)
```

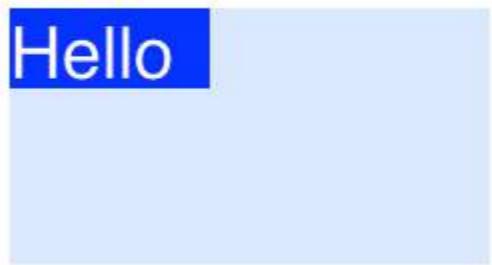
但是，我们将依次执行每个变换。第一个变换将在其名称中使用Make。后续的变换将不使用Make，但它们会将前一个变换作为参数。

示例



这次我们将之前的三个变换全部组合起来。

```
let degrees = 30.0  
let radians = CGFloat(degrees * M_PI / 180)  
  
// 平移  
var transform = CATransform3DMakeTranslation(90, 50, 0)  
  
// 旋转  
transform = CATransform3DRotate(transform, radians, 0.0, 0.0, 1.0)
```



If we wanted to rotate a layer clockwise 30 degrees, we would do the following:

```
let degrees = 30.0  
let radians = CGFloat(degrees * M_PI / 180)  
myLayer.transform = CATransform3DMakeRotation(radians, 0.0, 0.0, 1.0)
```

Notes

- Since we are working in two dimensions, we only want the xy plane to be rotated around the z axis. Thus we set x and y to `0.0` and set z to `1.0`.
- This rotated the layer in a clockwise direction. We could have rotated counterclockwise by setting z to `-1.0`.
- The red dot shows where the anchor point is. The rotation is done around the anchor point.

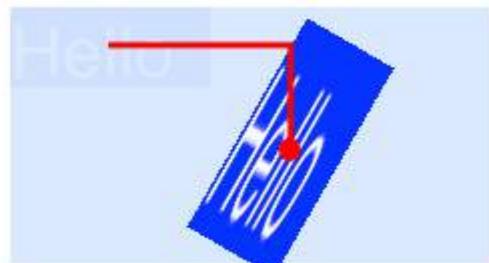
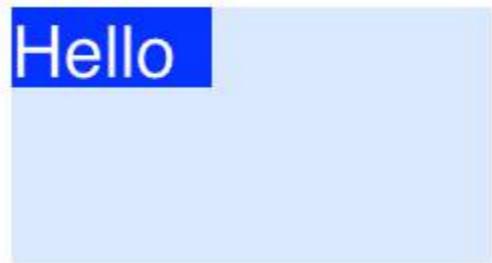
Multiple transforms

In order to combine multiple transforms we could use concatenation like this

```
CATransform3DConcat(a: CATransform3D, b: CATransform3D)
```

However, we will just do one after another. The first transform will use the Make in its name. The following transforms will not use Make, but they will take the previous transform as a parameter.

Example



This time we combine all three of the previous transforms.

```
let degrees = 30.0  
let radians = CGFloat(degrees * M_PI / 180)  
  
// translate  
var transform = CATransform3DMakeTranslation(90, 50, 0)  
  
// rotate  
transform = CATransform3DRotate(transform, radians, 0.0, 0.0, 1.0)
```

```
// 缩放  
transform = CATransform3DScale(transform, 0.5, 3.0, 1.0)
```

```
// 应用变换  
myLayer.transform = transform
```

注意事项

- 变换的顺序很重要。
- 所有操作都是相对于锚点（红点）进行的。

关于锚点和位置的说明

我们在上面进行了所有变换而没有改变锚点。不过有时确实需要改变锚点，比如你想绕中心点以外的某个点旋转。然而，这可能有点棘手。

锚点和位置都在同一个地方。锚点以图层的坐标系单位表示（默认是0.5, 0.5），位置以父图层的坐标系表示。它们可以这样设置

```
myLayer.anchorPoint = CGPointMake(x: 0.0, y: 1.0)  
myLayer.position = CGPointMake(x: 50, y: 50)
```

如果只设置锚点而不改变位置，帧会发生变化，使得位置处于正确的位置。更准确地说，帧是基于新的锚点和旧的位置重新计算的。这通常会产生意想不到的结果。以下两篇文章对此有很好的讨论。

- [关于anchorPoint](#)
- [平移 旋转 平移？](#)

另见

- [边框、圆角和阴影在CALayer上的应用](#)
- [使用带有贝塞尔路径的边框为图层](#)

此示例最初来自这个[Stack Overflow示例](#)。

第89.2节：阴影

您可以在每个图层上使用5个属性来配置阴影：

- shadowOffset - 此属性用于将阴影向左/右或向上/下移动

```
self.layer.shadowOffset = CGSizeMake(-1, -1); // 向左上移动1像素
```

```
self.layer.shadowOffset = CGSizeMake(1, 1); // 向右下移动1像素
```

- shadowColor - 此属性设置阴影的颜色

```
self.layer.shadowColor = [UIColor blackColor].CGColor;
```

- shadowOpacity - 阴影的不透明度，范围从0到1

```
self.layer.shadowOpacity = 0.2;
```

- shadowRadius - 模糊半径（相当于Sketch或Photoshop中的模糊属性）

```
// scale  
transform = CATransform3DScale(transform, 0.5, 3.0, 1.0)
```

```
// apply the transforms  
myLayer.transform = transform
```

Notes

- The order that the transforms are done in matters.
- Everything was done in relation to the anchor point (red dot).

A Note about Anchor Point and Position

We did all our transforms above without changing the anchor point. Sometimes it is necessary to change it, though, like if you want to rotate around some other point besides the center. However, this can be a little tricky.

The anchor point and position are both at the same place. The anchor point is expressed as a unit of the layer's coordinate system (default is 0.5, 0.5) and the position is expressed in the superlayer's coordinate system. They can be set like this

```
myLayer.anchorPoint = CGPointMake(x: 0.0, y: 1.0)  
myLayer.position = CGPointMake(x: 50, y: 50)
```

If you only set the anchor point without changing the position, then the frame changes so that the position will be in the right spot. Or more precisely, the frame is recalculated based on the new anchor point and old position. This usually gives unexpected results. The following two articles have an excellent discussion of this.

- [About the anchorPoint](#)
- [Translate rotate translate?](#)

See also

- [Border, rounded corners, and shadow on a CALayer](#)
- [Using a border with a Bezier path for a layer](#)

This example originally comes from [this Stack Overflow example](#).

Section 89.2: Shadows

You can use 5 properties on each layer to configure your shadows:

- shadowOffset - this property moves your shadow left/right or up/down

```
self.layer.shadowOffset = CGSizeMake(-1, -1); // 1px left and up
```

```
self.layer.shadowOffset = CGSizeMake(1, 1); // 1px down and right
```

- shadowColor - this sets the color of your shadow

```
self.layer.shadowColor = [UIColor blackColor].CGColor;
```

- shadowOpacity - this is the opacity of the shadow, from 0 to 1

```
self.layer.shadowOpacity = 0.2;
```

- shadowRadius - this is the blur radius (equivalent of the blur property in Sketch or Photoshop)

```
self.layer.shadowRadius = 6;
```

- shadowPath - 这是一个对性能非常重要的属性，当未设置时，iOS 会基于视图的 alpha 通道来绘制阴影，对于带有 alpha 的复杂 PNG 来说，这可能会导致性能开销较大。该属性允许你强制指定阴影的形状，从而提升性能。

Objective-C

```
self.layer.shadowPath = [UIBezierPath bezierPathWithOvalInRect:CGRectMake(0,0,100,100)]; // 这会生成一个圆形阴影
```

Swift 3

```
self.layer.shadowPath = UIBezierPath(ovalIn: CGRect(x: 0, y: 0, width: 100, height: 100)).cgPath
```

第89.3节：带自定义图像的发射器视图

例如，我们将创建一个包含发射器图层并能动画粒子的视图。

```
import QuartzCore

class ConfettiView: UIView {
    // 主发射器图层
    var emitter: CAEmitterLayer!

    // 发射颜色数组
    var colors: [UIColor]!

    // 出现强度
    var intensity: Float!

    private var active :Bool!

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }

    override init(frame: CGRect) {
        super.init(frame: frame)
        setup()
    }

    func setup() {
        // 初始化
        colors = [UIColor.redColor(),
                  UIColor.greenColor(),
                  UIColor.blueColor()]
    }

    intensity = 0.2

    active = false
}

func startConfetti() {
    emitter = CAEmitterLayer()

    发射器.发射器位置 = CGPoint(x: frame.尺寸.宽度 / 2.0, y: -20)
    发射器.发射器形状 = kCAEmitterLayerLine
```

```
self.layer.shadowRadius = 6;
```

- shadowPath - this is an important property for performance, when unset iOS bases the shadow on the alpha channel of the view, which can be performance intensive with a complex PNG with alpha. This property lets you force a shape for your shadow and be more performant because of it.

Objective-C

```
self.layer.shadowPath = [UIBezierPath bezierPathWithOvalInRect:CGRectMake(0,0,100,100)]; //this does a circular shadow
```

Swift 3

```
self.layer.shadowPath = UIBezierPath(ovalIn: CGRect(x: 0, y: 0, width: 100, height: 100)).cgPath
```

Section 89.3: Emitter View with custom image

For example we will create view that contains emitter layer and animates particles.

```
import QuartzCore

class ConfettiView: UIView {
    // main emitter layer
    var emitter: CAEmitterLayer!

    // array of color to emit
    var colors: [UIColor]!

    // intensity of appearance
    var intensity: Float!

    private var active :Bool!

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setup()
    }

    override init(frame: CGRect) {
        super.init(frame: frame)
        setup()
    }

    func setup() {
        // initialization
        colors = [UIColor.redColor(),
                  UIColor.greenColor(),
                  UIColor.blueColor()]
    }

    intensity = 0.2

    active = false
}

func startConfetti() {
    emitter = CAEmitterLayer()

    emitter.emitterPosition = CGPoint(x: frame.size.width / 2.0, y: -20)
    emitter.emitterShape = kCAEmitterLayerLine
```

```
发射器.发射器大小 = CGSize(宽度: frame.size.width, 高度: 1)
```

```
var 细胞 = [CAEmitterCell]()
for 颜色 in 颜色数组 {
    细胞添加(confettiWithColor(颜色))
}

发射器.发射器细胞 = 细胞
图层.添加子图层(发射器)
激活 = true
}

func 停止彩带() {
    发射器?.出生率 = 0
    激活 = false
}

func 彩带颜色(颜色: UIColor) -> CAEmitterCell {
    let 彩带 = CAEmitterCell()

    彩带.出生率 = 10.0 * 强度
    彩带.生命周期 = 180.0 * 强度
    彩带.生命周期范围 = 0
    彩带.颜色 = 颜色.CGColor
    彩带.速度 = CGFloat(350.0 * 强度)
    彩带.速度范围 = CGFloat(40.0 * 强度)
    彩带.发射经度 = CGFloat(M_PI)
    confetti.emissionRange = CGFloat(M_PI_4)
    confetti.spin = CGFloat(3.5 * 强度)
    confetti.spinRange = CGFloat(4.0 * 强度)

    // 警告：图层可以将此属性设置为 CGImageRef，以显示图像作为其
    // 内容。
    confetti.contents = UIImage(named: "confetti")?.CGImage
    return confetti
}

internal func isActive() -> Bool {
    return self.active
}
}
```

您需要添加“confetti”图像或使用confetti.contentsRect定义矩形

第89.4节：圆角

```
layer.masksToBounds = true;
layer.cornerRadius = 8;
```

第89.5节：使用 CAEmitterLayer 创建粒子

CAEmitterLayer类为 Core Animation 提供了粒子发射系统。粒子由CAEmitterCell的实例定义。

粒子绘制在图层的背景色和边框之上。

```
var emitter = CAEmitterLayer()
发射器.emitterPosition = CGPoint(x: frame.size.width / 2.0, y: -20)
```

```
emitter.emitterSize = CGSize(width: frame.size.width, height: 1)
```

```
var cells = [CAEmitterCell]()
for color in colors {
    cells.append(confettiWithColor(color))
}

emitter.emitterCells = cells
layer.addSublayer(emitter)
active = true
}

func stopConfetti() {
    emitter?.birthRate = 0
    active = false
}

func confettiWithColor(color: UIColor) -> CAEmitterCell {
    let confetti = CAEmitterCell()

    confetti.birthRate = 10.0 * intensity
    confetti.lifetime = 180.0 * intensity
    confetti.lifetimeRange = 0
    confetti.color = color.CGColor
    confetti.velocity = CGFloat(350.0 * intensity)
    confetti.velocityRange = CGFloat(40.0 * intensity)
    confetti.emissionLongitude = CGFloat(M_PI)
    confetti.emissionRange = CGFloat(M_PI_4)
    confetti.spin = CGFloat(3.5 * intensity)
    confetti.spinRange = CGFloat(4.0 * intensity)

    // WARNING: A layer can set this property to a CGImageRef to display the image as its
    // contents.
    confetti.contents = UIImage(named: "confetti")?.CGImage
    return confetti
}

internal func isActive() -> Bool {
    return self.active
}
}
```

You need to add "confetti" image or define rect with **confetti.contentsRect**

Section 89.4: Rounded corners

```
layer.masksToBounds = true;
layer.cornerRadius = 8;
```

Section 89.5: Creating particles with CAEmitterLayer

The **CAEmitterLayer** class provides a particle emitter system for Core Animation. The particles are defined by instances of **CAEmitterCell**.

The particles are drawn above the layer's background color and border.

```
var emitter = CAEmitterLayer()
emitter.emitterPosition = CGPoint(x: frame.size.width / 2.0, y: -20)
```

```
发射器。emitterShape = kCAEmitterLayerLine  
发射器。emitterSize = CGSize(width: frame.size.width, height: 1)
```

```
    发射器。emitterCells = cells  
图层。addSublayer(emitter)
```

第89.6节：如何向CALayer添加UIImage

你可以通过使用视图的layer的contents属性，简单地向视图添加图像：

```
myView.layer.contents = UIImage(named: "star")?.CGImage
```

- 注意，UIImage需要转换为CGImage。

如果你希望将图像添加到它自己的图层中，可以这样做：

```
let myLayer = CALayer()  
let myImage = UIImage(named: "star")?.CGImage  
myLayer.frame = myView.bounds  
myLayer.contents = myImage  
myView.layer.addSublayer(myLayer)
```

修改外观

上述代码生成了如下视图。浅蓝色是UIView，深蓝色的星星是UIImage。



如你所见，它看起来有像素化现象。这是因为UIImage比UIView小，所以它被缩放以填充视图，这是默认行为，除非你另有指定。

下面的示例展示了图层的contentsGravity属性的不同变化。代码如下：

```
myView.layer.contents = UIImage(named: "star")?.CGImage  
myView.layer.contentsGravity = kCAGravityTop  
myView.layer.geometryFlipped = true
```

在iOS中，如果你使用顶部或底部对齐，可能需要将geometryFlipped属性设置为true，否则它的表现会与你预期相反。
(只有重力方向垂直翻转，内容渲染不会翻转。如果内容被翻转导致问题，请参见[this Stack Overflow answer](#).)

下面每个contentsGravity设置都有两个UIView示例，一个视图比UIImage大，另一个比UIImage小。这样你可以看到缩放和重力的效果。

kCAGravityResize

这是默认值。

```
emitter.emitterShape = kCAEmitterLayerLine  
emitter.emitterSize = CGSize(width: frame.size.width, height: 1)
```

```
emitter.emitterCells = cells  
layer.addSublayer(emitter)
```

Section 89.6: How to add a UIImage to a CALayer

You can add an image to a view's layer simply by using its contents property:

```
myView.layer.contents = UIImage(named: "star")?.CGImage
```

- Note that the `UIImage` needs to be converted to a `CGImage`.

If you wish to add the image in its own layer, you can do it like this:

```
let myLayer = CALayer()  
let myImage = UIImage(named: "star")?.CGImage  
myLayer.frame = myView.bounds  
myLayer.contents = myImage  
myView.layer.addSublayer(myLayer)
```

Modifying the appearance

The above code produces a view like this. The light blue is the `UIView` and the dark blue star is the `UIImage`.



As you can see, though, it looks pixelated. This is because the `UIImage` is smaller than the `UIView` so it is being scaled to fill the view, which is the default if you don't specify anything else.

The examples below show variations on the layer's contentsGravity property. The code looks like this:

```
myView.layer.contents = UIImage(named: "star")?.CGImage  
myView.layer.contentsGravity = kCAGravityTop  
myView.layer.geometryFlipped = true
```

In iOS, you may want to set the `geometryFlipped` property to `true` if you are doing anything with top or bottom gravity, otherwise it will be the opposite of what you expect. (Only the gravity is flipped vertically, not the content rendering. If you are having trouble with the content being flipped, see [this Stack Overflow answer](#).)

There are two `UIView` examples below for every contentsGravity setting, one view is larger than the `UIImage` and the other is smaller. This way you can see the effects of the scaling and gravity.

kCAGravityResize

This is the default.



kCAGravityResizeAspect



kCAGravityResizeAspectFill



kCAGravityCenter



kCAGravityTop



kCAGravityResizeAspect



kCAGravityResizeAspectFill



kCAGravityCenter



kCAGravityTop



kCAGravityBottom



kCAGravityBottom



kCAGravityLeft



kCAGravityRight



kCAGravityTopLeft



kCAGravityTopRight



kCAGravityBottomLeft



kCAGravityBottomRight



kCAGravityTopRight



kCAGravityBottomLeft



kCAGravityBottomRight



相关

- [视图的内容模式属性](#)
- [在drawRect中使用CGContextDrawImage绘制UIImage](#)
- [CALayer教程：入门](#)

注意事项

- 此示例最初来自这个Stack Overflow答案。

第89.7节：禁用动画

CALayer属性动画默认启用。当不需要时，可以按如下方式禁用。

Swift

```
CATransaction.begin()  
CATransaction.setDisableActions(true)  
  
// 更改你不想动画的图层属性
```

Related

- [Content mode property of a view](#)
- [Drawing a UIImage in drawRect with CGContextDrawImage](#)
- [CALayer Tutorial: Getting Started](#)

Notes

- This example comes originally from [this Stack Overflow answer](#).

Section 89.7: Disable Animations

CALayer property animations are enabled by default. When this is undesirable, they can be disabled as follows.

Swift

```
CATransaction.begin()  
CATransaction.setDisableActions(true)  
  
// change layer properties that you don't want to animate
```

CATransaction.commit()

Objective-C

```
[CATransaction begin];
[CATransaction setDisableActions:YES];

// 更改你不想动画的图层属性

[CATransaction commit];
```

CATransaction.commit()

Objective-C

```
[CATransaction begin];
[CATransaction setDisableActions:YES];

// change layer properties that you don't want to animate

[CATransaction commit];
```

第90章：iOS - 使用Robbie Hanson框架实现XMPP

第90.1节：iOS XMPP Robbie Hanson示例与Openfire

SRXMPPDemo

在此下载示例及所有类 - <https://github.com/SahebRoy92/SRXMPPDemo>

这是一个使用Objective C实现的XMPP演示，包含了各种简单和复杂的功能。所有XMPP的功能均通过“in band”的xmpp函数完成。该项目包含的部分功能有——

SRXMPP - 一个几乎包含一对聊天应用所需所有功能的包装单例类。

- 一对聊天
- 聊天（文本消息）的核心数据实现，因此可以保存之前的消息和离线消息。
- 通过Robbie Hanson自有框架提供的XML和核心数据实现vCard（用户个人资料信息，包括自己和他人）。
- 好友状态的可用性（在线/离线/正在输入）

操作步骤

如果你想将此项目作为参考，可以执行以下操作——

1. 在一台在线服务器上安装Openfire - 租用服务器，安装Openfire。

2. 想在自己的电脑上无忧尝试 - 你需要下载、安装并设置3个东西才能开始

a. Java -

- 下载并安装适用于Mac的Java。

b. XAMPP -

- 安装XAMPP相对简单。
- 安装后，只需启动XAMPP并启动数据库(SQL)和Apache服务器。

Chapter 90: iOS - Implementation of XMPP with Robbie Hanson framework

Section 90.1: iOS XMPP Robbie Hanson Example with Openfire

SRXMPPDemo

Download the example and all the classes here - <https://github.com/SahebRoy92/SRXMPPDemo>

A demo on XMPP in Objective C, with various simple and complex features implemented in it. All the features of XMPP is done by "in band" xmpp functions. Few features this project contains are --

SRXMPP - A wrapper Singleton class that almost has all features needed for one-to-one chat application.

- one to one chat
- Core data implementation of chat (text message) thus having saving of previous messages, offline messages.
- implementation of vCard(profile information of user, own and others too) from XML and Core Data provided by Robbie Hanson's own framework.
- availability of friends status (online/offline/typing)

Steps to follow

You want to use this project as a reference then you can do the following--

1. **Installed Openfire in a live server** - Rent a server, install openfire.

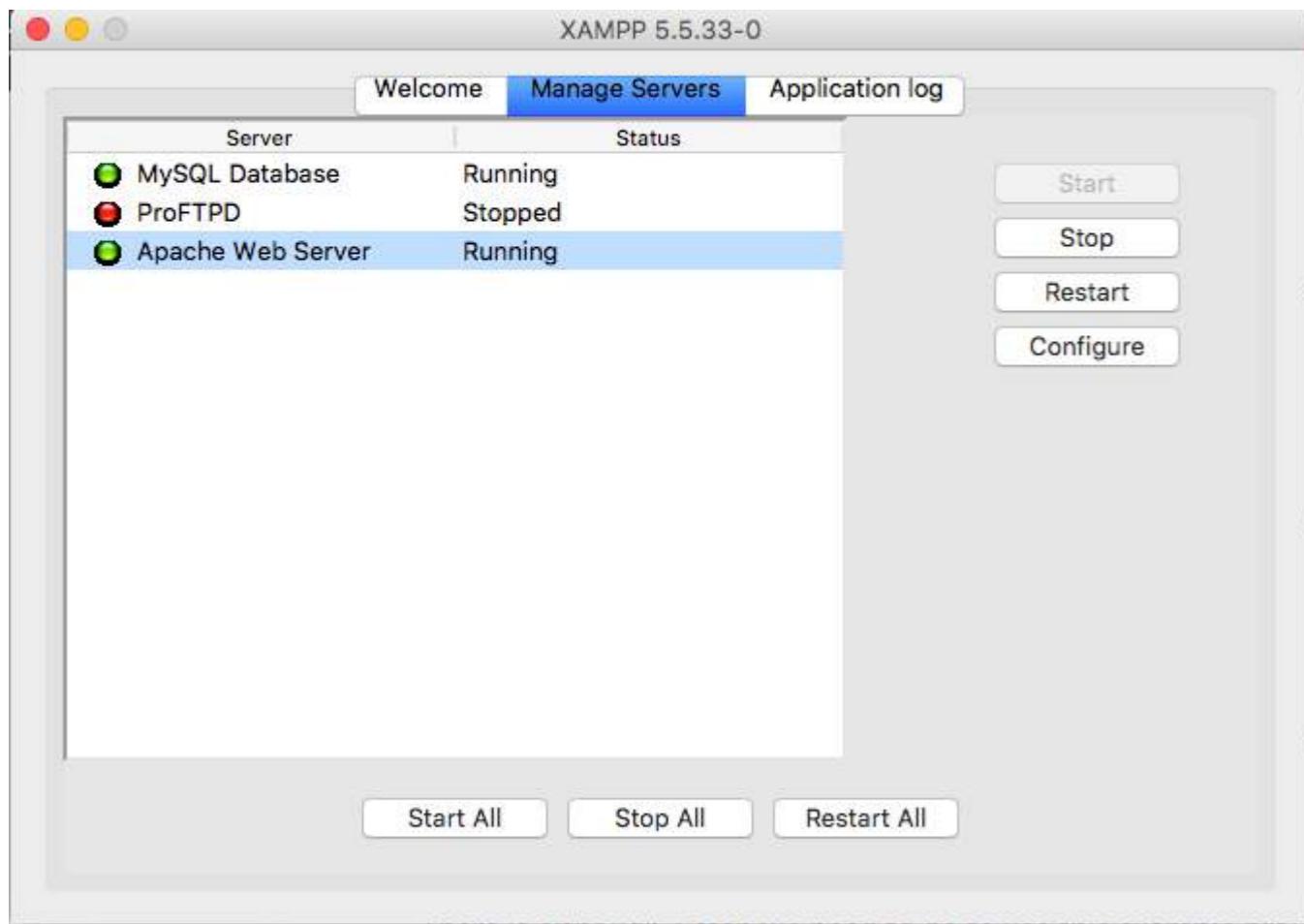
2. **Want to try it out without a hassle in your own computer** - You need to download, install and setup 3 things to start

a. Java -

- Download and install Java for Mac.

b. XAMPP -

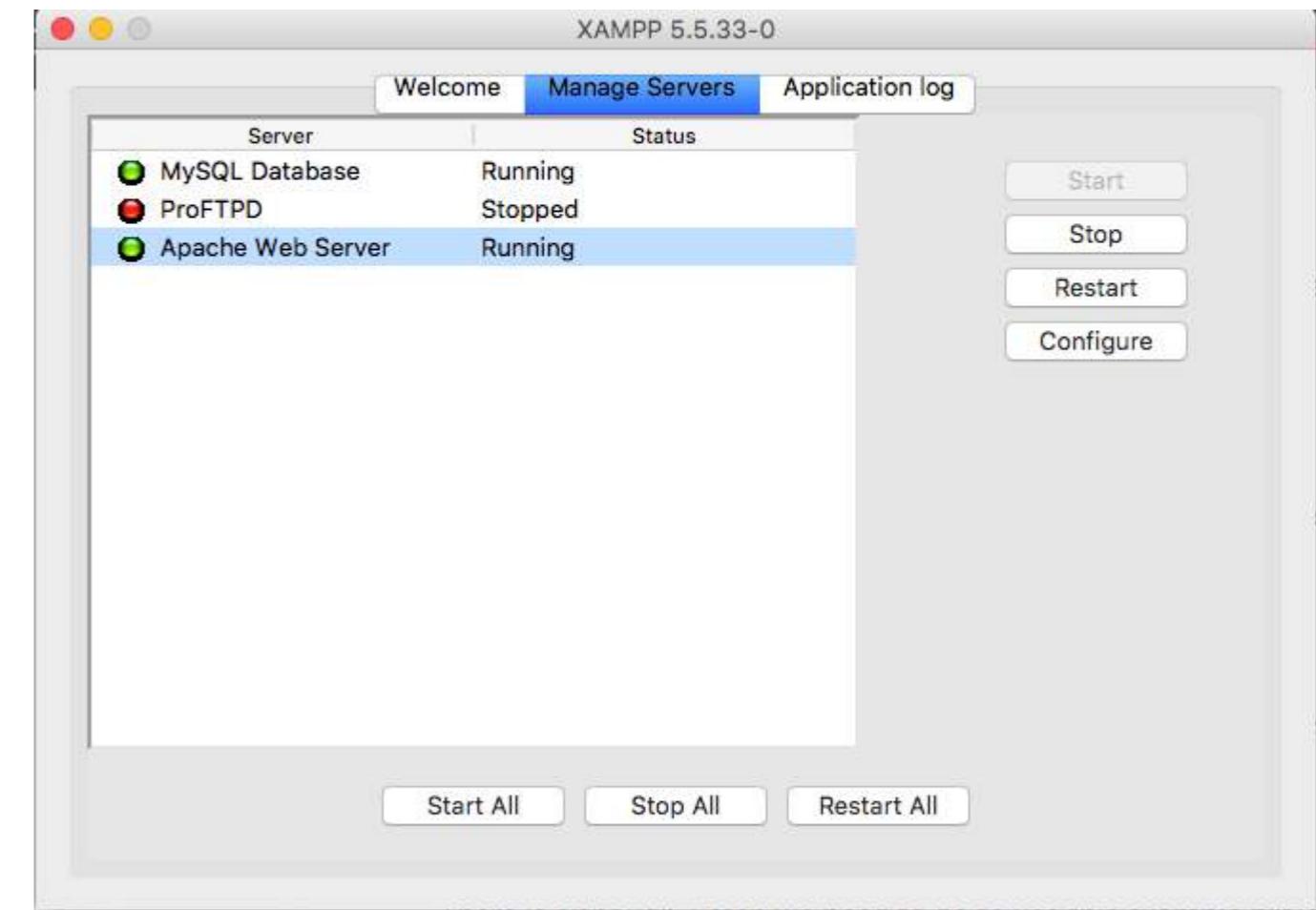
- Install XAMPP is relatively easy.
- After installation just start the XAMPP and start **Database(SQL)** and **Apache Server**.



- 然后打开浏览器，粘贴此URL<http://localhost/phpmyadmin/>
- 。从左侧面板创建一个新的数据库。
- 给数据库命名，名称可以随意，但请记住这个名字，假设我们命名为ChatDB

c. Openfire -

- 安装Openfire并运行应用程序，然后“启动Openfire”



- Then open browser and paste this URL <http://localhost/phpmyadmin/>
- . Create a new DB from the left hand side panel.
- Name the DB anything but remember this name, suppose we name it ChatDB**

c. Openfire -

- Install Openfire and run the application and "Start Openfire"



- 打开浏览器并粘贴此网址 -
[\[http://localhost:9090/setup/index.jsp\]\(http://localhost:9090/setup/index.jsp\)](http://localhost:9090/setup/index.jsp)

- 进行正常安装
 - 选择语言 >
 - 服务器设置，保持原样，直接继续 >
 - 数据库设置，保持为“标准数据库连接”如所选 >
 - 数据库设置 - 标准连接”。现在记住你设置的数据库名称是**ChatDB**。
 - 选择数据库驱动预设为*“MySQL”。JDBC驱动类保持不变。现在在数据库URL中你可以看到括号中提到的主机名和数据库名。只需将主机名改为“localhost”，数据库名改为“ChatDB”，或者你之前在设置XAMPP时设置的其他数据库名。用户名和密码保持为空。填写如图所示的详细信息

Database Settings - Standard Connection

Specify a JDBC driver and connection properties to connect to your database. If you need more information about this process please see the database documentation distributed with Openfire.

Note: Database scripts for most popular databases are included in the server distribution at [Openfire_HOME]/resources/database.

- Open Browser and Paste this URL -
[\[http://localhost:9090/setup/index.jsp\]\(http://localhost:9090/setup/index.jsp\)](http://localhost:9090/setup/index.jsp)
- Do normal setup
 - Select Language >
 - Server settings, leave as it is, just do continue >
 - Database Settings, leave as it is as "Standard Database Connection as selected" >
 - Database Settings - Standard Connection". Now remember the name of the DB you set was **ChatDB**.
 - Select Database Driver Presets as *"MySQL". Leave JDBC Driver Class as it is. Now in the Database URL you can see, brackets mentioning hostname and Database Name. Just change Hostname to "**localhost**", and database name to "**ChatDB**", or any other name of DB you have set earlier, while setting up XAMPP. Leave the Username and password as blank. Fill up details like the image here

Database Settings - Standard Connection

Specify a JDBC driver and connection properties to connect to your database. If you need more information about this process please see the database documentation distributed with Openfire.

Note: Database scripts for most popular databases are included in the server distribution at [Openfire_HOME]/resources/database.

- 接下来通过设置用户名和密码并确认完成安装。就这样，你完成了 Openfire 的设置。

现在到了你需要在代码中修改一个小细节的部分。

#Important 我们需要进入类 **SRXMPP.m**，找到 NSString 外部变量 **SRXMPP_Hostname** (在顶部) 并将其值覆盖为

- 安装了 OpenFire 的服务器的 IP，或者
- 如果你是在本地安装的，请将该值覆盖为“localhost”。

就是这样，你已经准备好使用这个示例项目，开始编码并将其打造为自己的更好项目了。

这个入门包将帮助你更好地理解 XMPP 结构，并掌握 XMPP 协议。

你可以在本网站找到其他 XMPP 协议 -

[\[https://xmpp.org/rfcs/rfc3920.html\]\(https://xmpp.org/rfcs/rfc3920.html\)](https://xmpp.org/rfcs/rfc3920.html)

开发工作仍在进行中，部分内容我希望以后能加入

1. 群聊
2. 图片发送支持

总而言之，这个示例项目连同单例几乎包含了一对一聊天应用所需的所有功能。

- Next complete setup by giving a username and password and reconfirming it. That's it your done Setting up Openfire.

Now the part comes when you have to change a tiny detail in the code.

#Important We need to go to the class - **SRXMPP.m**, locate the NSString extern **SRXMPP_Hostname** (in the top) and overwrite the value of it to the

- IP of the server where OpenFire is installed , **OR**
- if you have installed it locally, overwrite the value to - "**localhost**".

That's it, you are ready to use this example project and start coding and making it into a better project of your own.

This starter pack will help you in understanding XMPP structure better as well as getting a grasp into XMPP protocols.

You can find other XMPP protocols here in this site -

[\[https://xmpp.org/rfcs/rfc3920.html\]\(https://xmpp.org/rfcs/rfc3920.html\)](https://xmpp.org/rfcs/rfc3920.html)

Development is still left and parts where I hope to include them later on

1. Group Chat
2. Image sending support

In short this example project along with the singleton has almost all features that are needed for a One-to-One chat application to have.

第91章：Swift 与 Objective-C 互操作性

第91.1节：在 Swift 中使用 Objective-C 类

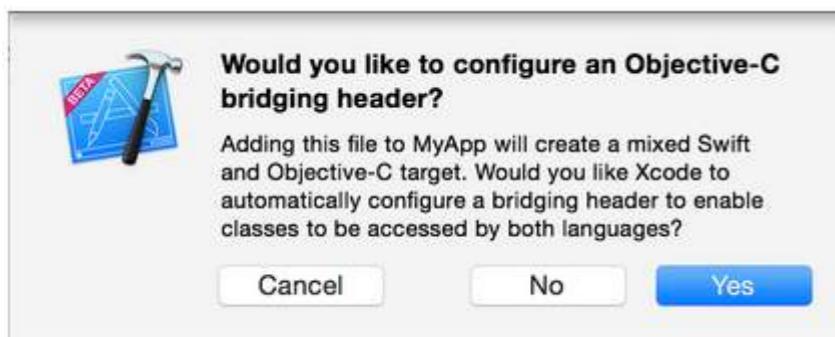
如果你有一个现有的类想要使用，执行**步骤2**然后跳到**步骤5**。（在某些情况下，我不得不在较旧的 ObjC 文件中显式添加`#import <Foundation/Foundation.h>`）

步骤1：添加Objective-C实现文件 -- .m

向你的类中添加一个.m文件，并命名为CustomObject.m

步骤2：添加桥接头文件

添加你的.m文件时，系统很可能会弹出如下提示：



点击YES！

如果没有看到提示，或者不小心删除了桥接头文件，请在项目中添加一个新的.h文件，命名为<#YourProjectName#>-Bridging-Header.h

在某些情况下，尤其是使用ObjC框架时，你不会显式添加Objective-C类，Xcode无法找到链接器。此时，创建上述命名的.h文件，确保在目标的项目设置中链接其路径，如下所示：

Chapter 91: Swift and Objective-C interoperability

Section 91.1: Using Objective-C Classes in Swift

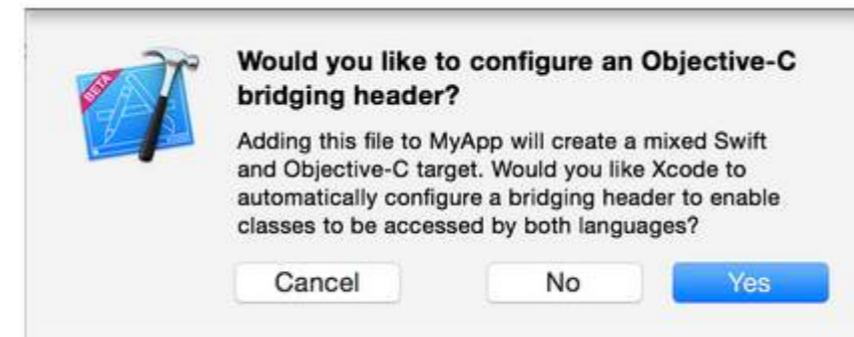
If you have an existing class that you'd like to use, perform **Step 2** and then skip to **Step 5**. (For some cases, I had to add an explicit `#import <Foundation/Foundation.h>` to an older ObjC File)

Step 1: Add Objective-C Implementation -- .m

Add a .m file to your class, and name it CustomObject.m

Step 2: Add Bridging Header

When adding your .m file, you'll likely be hit with a prompt that looks like this:



Click YES！

If you did not see the prompt, or accidentally deleted your bridging header, add a new .h file to your project and name it <#YourProjectName#>-Bridging-Header.h

In some situations, particularly when working with ObjC frameworks, you don't add an Objective-C class explicitly and Xcode can't find the linker. In this case, create your .h file named as mentioned above, then make sure you link its path in your target's project settings like so:

```

// AppDelegate.swift
// SkipParse
// Created by Logan Wright on 7/20/14.
// Copyright (c) 2014 Logan Wright. All rights reserved.

import UIKit
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(application: UIApplication!, didFinishLaunchingWithOptions launchOptions: NSDictionary!) -> Bool {
        // Override point for customization after application launch.
        return true
    }

    func applicationWillResignActive(application: UIApplication!) {
        // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
        // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game.
    }

    func applicationDidEnterBackground(application: UIApplication!) {
        // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
        // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
    }

    func applicationWillEnterForeground(application: UIApplication!) {
        // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.
    }

    func applicationDidBecomeActive(application: UIApplication!) {
        // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
    }

    func applicationWillTerminate(application: UIApplication!) {
        // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
    }
}

```

注意

最佳实践是使用\$(SRCROOT)宏来链接项目，这样如果你移动项目，或者与他人通过远程仓库协作，项目仍然能正常工作。\$(SRCROOT)可以理解为包含你的.xcodeproj文件的目录。它可能看起来像这样：

\$(SRCROOT)/Folder/Folder/<#YourProjectName#>-Bridging-Header.h

步骤3：添加Objective-C头文件 -- .h

添加另一个.h文件并命名为CustomObject.h

步骤4：构建你的Objective-C类

在CustomObject.h中

```

#import <Foundation/Foundation.h>

@interface CustomObject : NSObject

@property (strong, nonatomic) id someProperty;

- (void) someMethod;

@end

```

在CustomObject.m中

```

// AppDelegate.swift
// SkipParse
// Created by Logan Wright on 7/20/14.
// Copyright (c) 2014 Logan Wright. All rights reserved.

import UIKit
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(application: UIApplication!, didFinishLaunchingWithOptions launchOptions: NSDictionary!) -> Bool {
        // Override point for customization after application launch.
        return true
    }

    func applicationWillResignActive(application: UIApplication!) {
        // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
        // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game.
    }

    func applicationDidEnterBackground(application: UIApplication!) {
        // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
        // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
    }

    func applicationWillEnterForeground(application: UIApplication!) {
        // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.
    }

    func applicationDidBecomeActive(application: UIApplication!) {
        // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
    }

    func applicationWillTerminate(application: UIApplication!) {
        // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
    }
}

```

Note

It's best practice to link your project using the \$(SRCROOT) macro so that if you move your project, or work on it with others using a remote repo, it will still work. \$(SRCROOT) can be thought of as the directory that contains your.xcodeproj file. It might look like this:

\$(SRCROOT)/Folder/Folder/<#YourProjectName#>-Bridging-Header.h

Step 3: Add Objective-C Header -- .h

Add another .h file and name it CustomObject.h

Step 4: Build your Objective-C Class

In CustomObject.h

```

#import <Foundation/Foundation.h>

@interface CustomObject : NSObject

@property (strong, nonatomic) id someProperty;

- (void) someMethod;

@end

```

In CustomObject.m

```
#import "CustomObject.h"

@implementation CustomObject

- (void) someMethod {
    NSLog(@"SomeMethod Ran");
}

@end
```

步骤5：将类添加到Bridging-Header

在YourProject-Bridging-Header.h中：

```
#import "CustomObject.h"
```

步骤6：使用你的对象

在SomeSwiftFile.swift中：

```
var instanceOfCustomObject: CustomObject = CustomObject()
instanceOfCustomObject.someProperty = "Hello World"
println(instanceOfCustomObject.someProperty)
instanceOfCustomObject.someMethod()
```

无需显式导入，这就是桥接头文件的作用。

第91.2节：在Objective-C中使用Swift类

步骤1：创建新的Swift类

向你的项目中添加一个.swift文件，命名为MySwiftObject.swift

在MySwiftObject.swift中：

```
import Foundation

class MySwiftObject : NSObject {

    var someProperty: AnyObject = "Some Initializer Val"

    init() {}

    func someFunction(someArg:AnyObject) -> String {
        var returnVal = "你发送给我的 \(someArg)"
        return returnVal
    }
}
```

步骤 2：将 Swift 文件导入到 ObjC 类中

在 SomeRandomClass.m 中：

```
#import "<#YourProjectName#>-Swift.h"
```

文件：`<#YourProjectName#>-Swift.h` 应该已经在你的项目中自动创建，即使你看不到它。

```
#import "CustomObject.h"

@implementation CustomObject

- (void) someMethod {
    NSLog(@"SomeMethod Ran");
}

@end
```

Step 5: Add Class to Bridging-Header

In YourProject-Bridging-Header.h:

```
#import "CustomObject.h"
```

Step 6: Use your Object

In SomeSwiftFile.swift:

```
var instanceOfCustomObject: CustomObject = CustomObject()
instanceOfCustomObject.someProperty = "Hello World"
println(instanceOfCustomObject.someProperty)
instanceOfCustomObject.someMethod()
```

No need to import explicitly, that's what the bridging header is for.

Section 91.2: Using Swift Classes in Objective-C

Step 1: Create New Swift Class

Add a .swift file to your project, and name it MySwiftObject.swift

In MySwiftObject.swift:

```
import Foundation

class MySwiftObject : NSObject {

    var someProperty: AnyObject = "Some Initializer Val"

    init() {}

    func someFunction(someArg:AnyObject) -> String {
        var returnVal = "You sent me \(someArg)"
        return returnVal
    }
}
```

Step 2: Import Swift Files to ObjC Class

In SomeRandomClass.m:

```
#import "<#YourProjectName#>-Swift.h"
```

The file:`<#YourProjectName#>-Swift.h` should already be created automatically in your project, even if you can not

步骤 3：使用你的类

```
MySwiftObject * myOb = [MySwiftObject new];
NSLog(@"MyOb.someProperty: %@", myOb.someProperty);
myOb.someProperty = @"Hello World";
NSLog(@"MyOb.someProperty: %@", myOb.someProperty);
NSString * retString = [myOb someFunction:@"Arg"];
NSLog(@"RetString: %@", retString);
```

注意：

1. 代码补全的表现没有达到我期望的准确度。在我的系统上，使用“cmd + r”快速构建似乎有助于 Swift 找到一些 Objc 代码，反之亦然。似乎有助于 Swift 找到一些 Objc 代码，反之亦然。

2. 如果你向旧项目中添加了.swift文件并出现错误：dyld: Library not loaded:
@rpath/libswift_stdlib_core.dylib，尝试完全重新启动 Xcode。

3. 虽然最初可以通过使用@objc前缀在 Objective-C 中使用纯 Swift 类，但在 Swift 2.0 之后，这已不再可能。请参阅编辑历史以获取原始解释。如果未来 Swift 版本重新启用此功能，答案将相应更新。

see it.

Step 3: Use your class

```
MySwiftObject * myOb = [MySwiftObject new];
NSLog(@"MyOb.someProperty: %@", myOb.someProperty);
myOb.someProperty = @"Hello World";
NSLog(@"MyOb.someProperty: %@", myOb.someProperty);
NSString * retString = [myOb someFunction:@"Arg"];
NSLog(@"RetString: %@", retString);
```

Note:

1. CodeCompletion wasn't behaving as accurately as I'd like it to. On my system, running a quick build w/ "cmd + r" seemed to help Swift find some of the Objc code and vice versa.

2. If you add .swift file to an older project and get error: dyld: Library not loaded:
@rpath/libswift_stdlib_core.dylib, try completely [restarting Xcode](#).

3. While it was originally possible to use pure Swift classes in Objective-C by using the @objc prefix, after Swift 2.0, this is no longer possible. See edit history for original explanation. If this functionality is reenabled in future Swift versions, the answer will be updated accordingly.

第92章：自定义字体

第92.1节：嵌入自定义字体

自定义字体支持

想要使用自定义字体的应用程序现在可以将这些字体包含在其应用程序包中，并通过在Info.plist文件中包含UIAppFonts键来向系统注册这些字体。该键的值是一个字符串数组，用于标识应用程序包中的字体文件。当系统检测到该键时，会加载指定的字体并使其可供应用程序使用。

字体设置在Info.plist后，您可以像使用其他字体一样在IB中或通过编程方式使用自定义字体。

1. 将字体拖放到Xcode的Supporting Files文件夹中。别忘了在“Add to targets”部分勾选您的应用程序。从此刻起，您可以在IB中使用该字体，并从字体面板中选择它。



2. 要使该字体在设备上可用，打开Info.plist并添加“Fonts provided by application”键（UIAppFonts）。将字体名称作为Item 0键的值添加。注意：字体名称可能与您的字体文件名不同。



3. 使用以下代码片段获取自定义添加的字体名称

[Swift 3]

```
for family in UIFont.familyNames {  
    print("\(family)")  
  
    for name in UIFont.fontNames(forFamilyName: family) {  
        print("    \(name)")  
    }  
}
```

[Objective - C]

```
for (NSString *familyName in [UIFont familyNames]) {  
    NSLog(@"Family name: %@", familyName);  
    for (NSString *fontName in [UIFont fontNamesForFamilyName:familyName]) {  
        NSLog(@"--Font name: %@", fontName);  
    }  
}
```

第92.2节：在

Storyboard中将自定义字体应用于控件

以下示例展示了如何将自定义字体应用于导航栏，并包含了针对Xcode中一些奇怪行为的修复。也可以将自定义字体应用于任何其他UI控件，如UILabel，

Chapter 92: Custom fonts

Section 92.1: Embedding custom fonts

Custom Font Support

Applications that want to use custom fonts can now include those fonts in their application bundle and register those fonts with the system by including the UIAppFonts key in their Info.plist file. The value of this key is an array of strings identifying the font files in the application's bundle. When the system sees the key, it loads the specified fonts and makes them available to the application.

Once the fonts have been set in the `Info.plist`, you can use your custom fonts as any other font in IB or programmatically.

1. Drag and drop your font to Xcode Supporting Files folder. Don't forget to mark your app at "Add to targets" section. From this moment you can use this font in IB and choose it from font pallet.



2. To make this font available on the device, open `Info.plist` and add `Fonts provided by application` key (UIAppFonts). Add font name as the value to the Item 0 key. Note: Font name can vary from your font file name.



3. Get the custom added font name using below snippet

[Swift 3]

```
for family in UIFont.familyNames {  
    print("\(family)")  
  
    for name in UIFont.fontNames(forFamilyName: family) {  
        print("    \(name)")  
    }  
}
```

[Objective - C]

```
for (NSString *familyName in [UIFont familyNames]) {  
    NSLog(@"Family name: %@", familyName);  
    for (NSString *fontName in [UIFont fontNamesForFamilyName:familyName]) {  
        NSLog(@"--Font name: %@", fontName);  
    }  
}
```

Section 92.2: Applying custom fonts to controls within a Storyboard

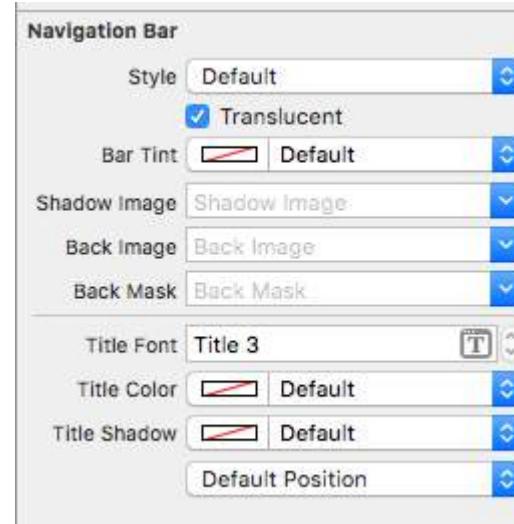
The following example shows how to apply custom fonts to a Navigation Bar and includes fixes for some quirky behaviors found in Xcode. One also may apply the custom fonts to **any other UIControls** such as **UILabels**,

UIButton等，方法是在将自定义字体添加到项目后，通过属性检查器进行设置。请注意文末附近的外部链接，包含可用示例和视频。

1. 在导航控制器中选择您的导航栏



2. 在属性检查器中更改标题字体



(在 Xcode 识别新字体之前，您可能需要切换导航栏的栏色)

注意事项（警告）

已验证此方法在 Xcode 7.1.1 及以上版本有效。（请参见下面的示例）

1. 您确实需要切换导航栏的栏色，字体才会生效（这似乎是 Xcode 的一个漏洞；您可以切换回默认，字体就会生效）
2. 如果选择系统字体 ~ 请确保字号不是 0.0 （否则新字体将被忽略）



3. 当视图层级中只有一个导航栏时，这似乎能正常工作。看来同一堆栈中的次级导航栏会被忽略。（注意，如果显示主导航控制器的导航栏，所有其他自定义导航栏设置都会被忽略）

注意事项（二）

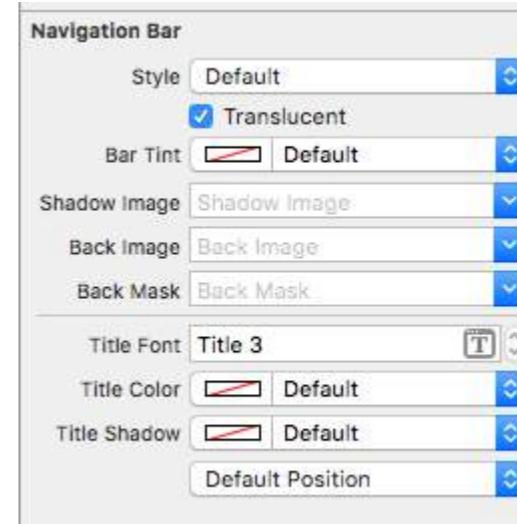
其中一些内容有重复，说明它们非常值得注意。

UIButtons, and more by using the attributes inspector after the custom font is added to the project. Please note the external links to working samples and videos near the bottom.

1. Select Your Navigation Bar within your Navigation Controller



2. Change the Title Font in the Attributes Inspector

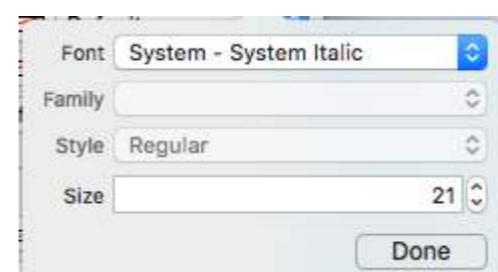


(You will likely need to toggle the Bar Tint for the Navigation Bar before Xcode picks up the new font)

Notes (Caveats)

Verified that this does work on Xcode 7.1.1+. (See the Samples below)

1. You do need to toggle the nav bar tint before the font takes effect (seems like a bug in Xcode; you can switch it back to default and font will stick)
2. If you choose a system font ~ Be sure to make sure the size is not 0.0 (Otherwise the new font will be ignored)



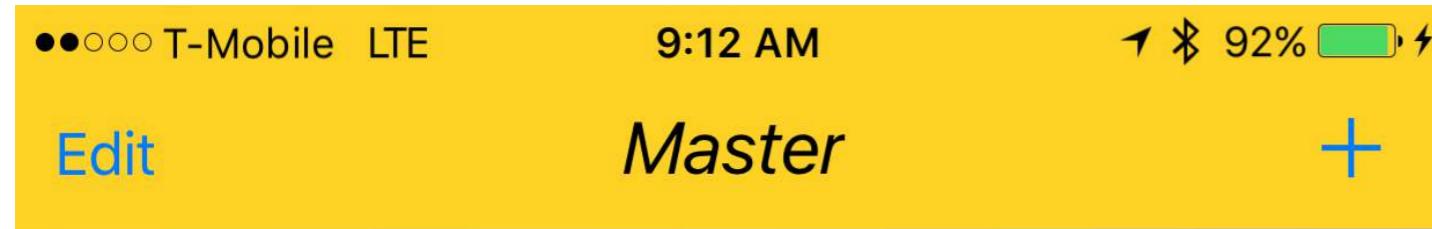
3. Seems like this works with no problem when only one NavBar is in the view hierarchy. It appears that secondary NavBars in the same stack are ignored. (Note that if you show the master navigation controller's navBar all the other custom navBar settings are ignored).

Gotchas (deux)

Some of these are repeated which means they are very likely worth noting.

- 有时故事板的 XML 会损坏。这需要您以源代码模式查看故事板结构（右键点击故事板文件 > 以...方式打开）
- 在某些情况下，与用户定义的运行时属性关联的 `navigationItem` 标签被设置为 `view` 标签的子节点，而不是 `view controller` 标签的子节点。如果是这样，请将其从标签之间移除以确保正常运行。
- 切换导航栏颜色以确保使用自定义字体。
- 验证字体的大小参数，除非使用动态字体样式
- 视图层级会覆盖设置。看起来每个堆栈只能使用一种字体。

结果



示例

- [展示多种字体的高级项目视频](#)
- [简单源码下载](#)
- [高级项目下载 ~ 展示多种导航栏字体和自定义字体解决方法](#)
- [展示多种字体和自定义字体的视频](#)

处理自定义字体

注意 ~ 可以在 [Code With Chris](#) 网站找到一份很好的检查清单，并且可以查看示例下载项目。

如果你有自己的字体并想在故事板中使用，那么在以下SO 问题中有一套不错的答案。其中一个答案列出了这些步骤。

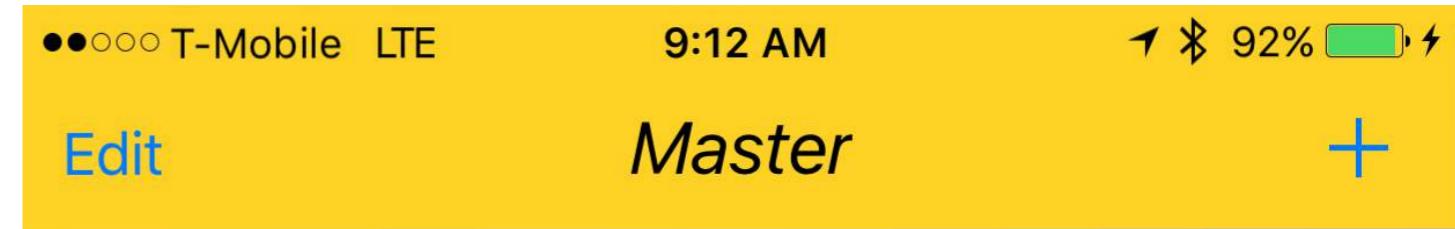
- 获取您的自定义字体文件 (.ttf, .ttc)
- 将字体文件导入到您的Xcode项目中
- 在app-info.plist中，添加一个名为“Fonts provided by application”的键。它是数组类型，将所有字体文件名（包括文件扩展名）添加到数组中。
- 在Storyboard中，选中导航栏，进入属性检查器，点击字体选择区域右侧的图标按钮。在弹出面板中，选择字体为自定义，并选择您嵌入字体的字体族名称。

自定义字体解决方法

所以Xcode看起来可以自然地处理UINavigationItem上的自定义字体，但该功能实际上没有正确更新（所选字体被忽略）。

- Sometimes the storyboard xml gets corrupt. This requires you to review the structure in Storyboard as Source Code mode (right click the storyboard file > Open As ...)
- In some cases the `navigationItem` tag associated with user defined runtime attribute was set as an xml child of the `view` tag instead of the `view controller` tag. If so remove it from between the tags for proper operation.
- Toggle the NavBar Tint to ensure the custom font is used.
- Verify the size parameter of the font unless using a dynamic font style
- View hierarchy will override the settings. It appears that one font per stack is possible.

Result



Samples

- [Video Showing Multiple Fonts In Advanced Project](#)
- [Simple Source Download](#)
- [Advanced Project Download ~ Shows Multiple NavBar Fonts & Custom Font Workaround](#)
- [Video Showing Multiple Fonts & Custom Fonts](#)

Handling Custom Fonts

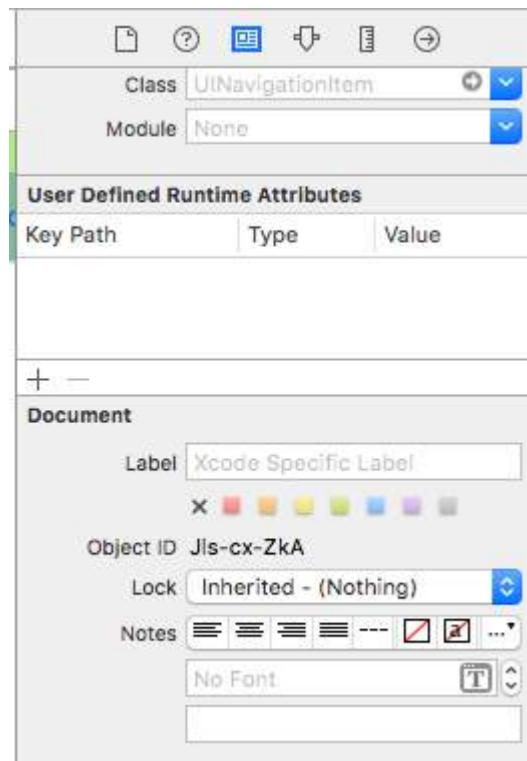
Note ~ A [nice checklist](#) can be found from the [Code With Chris](#) website and you can see the sample download project.

If you have your own font and want to use that in your storyboard, then there is a decent set of answers on the following [SO Question](#). One answer identifies these steps.

- Get your custom font file(.ttf,.ttc)
- Import the font files to your Xcode project
- In the app-info.plist, add a key named Fonts provided by application. It's an array type, add all your font file names to the array, note: including the file extension.
- In the storyboard, on the NavigationBar go to the Attribute Inspector, click the right icon button of the Font select area. In the popup panel, choose Font to Custom, and choose the Family of your embedded font name.

Custom Font Workaround

So Xcode naturally looks like it can handle custom fonts on UINavigationItem but that feature is just not updating properly (The font selected is ignored).



解决方法如下：

一种方法是结合Storyboard和添加一行代码来修复：首先向视图控制器添加一个UIView（UIButton、UILabel或其他UIView子类）（不是导航项.....Xcode目前不允许这样做）。添加控件后，您可以在Storyboard中修改字体，并将其作为Outlet引用添加到您的视图控制器。只需将该视图赋值给UINavigationItem.titleView。如果需要，也可以在代码中设置文本名称。已报告的Bug (23600285)。

```
@IBOutlet var customFontTitleView: UIButton!

//过了一段时间....
self.navigationItem.titleView = customFontTitleView
```

注意 - 这个示例来源于我在SO (Stack Overflow) 上发布的一个回答 ([here](#))。

第92.3节：使用Storyboard的自定义字体

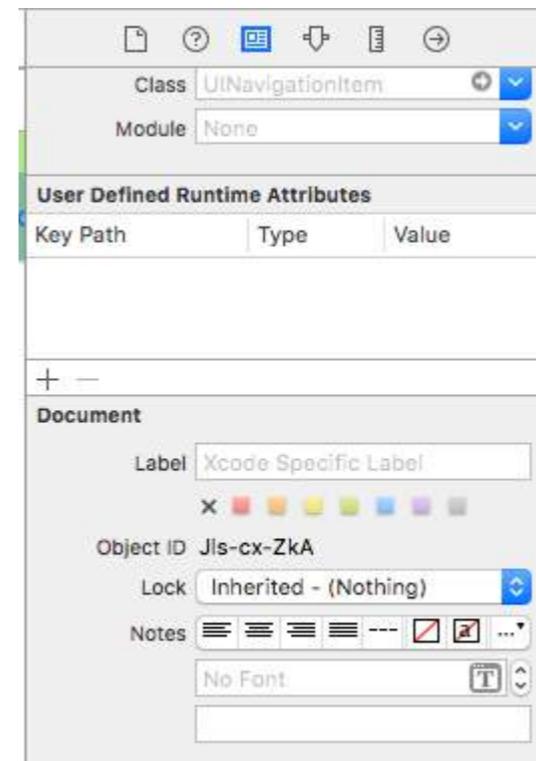
Storyboard中UI组件的自定义字体可以通过[用户定义运行时属性](#)和[Storyboard中的类别](#)轻松实现。

其优点包括，

- 无需为UI元素定义outlet
- 无需通过代码为元素设置字体。

操作步骤

1. 字体文件：将字体文件 (.ttf) 添加到应用程序包中，并在Info.plist中添加字体条目在应用程序提供的字体下，如本自定义字体文档所示。
2. 定义类别：添加一个类似UIKit+IBExtensions的文件，并为UI元素添加类别，如



To workaround this:

One way is to fix using the storyboard and adding a line of code: First add a UIView (UIButton, UILabel, or some other UIView subclass) to the View Controller (Not the Navigation Item...Xcode is not currently allowing one to do that). After you add the control you can modify the font in the storyboard and add a reference as an outlet to your View Controller. Just assign that view to the UINavigationItem.titleView. You could also set the text name in code if necessary. Reported Bug (23600285).

```
@IBOutlet var customFontTitleView: UIButton!

//Sometime later...
self.navigationItem.titleView = customFontTitleView
```

Note - This example is derived from an answer I posted on SO ([here](#)).

Section 92.3: Custom Fonts with Storyboard

Custom Fonts for UI components from storyboard can be easily achieved with [User Defined Runtime Attributes](#) in storyboard and [Categories](#).

The advantages are like,

- No need to define outlets for the ui element
- No need to set font for elements programmatically.

Steps to follow

1. **Font File:** Add the Font file (.ttf) to the application bundle and add the entry for the font in Info.plist under **Font provided by application** as in this documentation of custom fonts.
2. **Define Categories:** Add a file like **UIKit+IBExtensions** and add the categories for UI elements like

UILabel、UIButton 等你想设置自定义字体的控件。所有类别都会有一个自定义属性，比如fontName。这个属性稍后将在故事板中用于设置自定义字体（如步骤4所示）。

UIKit+IBExtensions.h

```
#import <UIKit/UIKit.h>

// UILabel 的类别扩展
@interface UILabel (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end

// UITextField 的类别扩展
@interface UITextField (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end

// UIButton 的类别扩展
@interface UIButton (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end
```

3. Getter 和 Setter：为每个类别定义 fontName 属性的 getter 和 setter 已添加。

UIKit+IBExtensions.m

```
#import "UIKit+IBExtensions.h"

@implementation UILabel (IBExtensions)

- (NSString *)fontName {
    return self.font.fontName;
}

- (void)setFontName:(NSString *)fontName {
    self.font = [UIFont fontWithName:fontName size:self.font.pointSize];
}
@end

@implementation UITextField (IBExtensions)

- (NSString *)fontName {
    return self.font.fontName;
}

- (void)setFontName:(NSString *)fontName {
    self.font = [UIFont fontWithName:fontName size:self.font.pointSize];
}
@end

@implementation UIButton (IBExtensions)

- (NSString *)fontName {
    return self.titleLabel.font.fontName;
}
```

UILabel, UIButton etc. for which you want to set custom font. All the categories will be having a custom property say **fontName**. This will be using from the storyboard later for setting custom font (as in step 4).

UIKit+IBExtensions.h

```
#import <UIKit/UIKit.h>

// Category extension for UILabel
@interface UILabel (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end

// Category extension for UITextField
@interface UITextField (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end

// Category extension for UIButton
@interface UIButton (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end
```

3. **Getters and Setters:** Define getters and setters for the fontName property towards each category added.

UIKit+IBExtensions.m

```
#import "UIKit+IBExtensions.h"

@implementation UILabel (IBExtensions)

- (NSString *)fontName {
    return self.font.fontName;
}

- (void)setFontName:(NSString *)fontName {
    self.font = [UIFont fontWithName:fontName size:self.font.pointSize];
}
@end

@implementation UITextField (IBExtensions)

- (NSString *)fontName {
    return self.font.fontName;
}

- (void)setFontName:(NSString *)fontName {
    self.font = [UIFont fontWithName:fontName size:self.font.pointSize];
}
@end

@implementation UIButton (IBExtensions)

- (NSString *)fontName {
    return self.titleLabel.font.fontName;
}
```

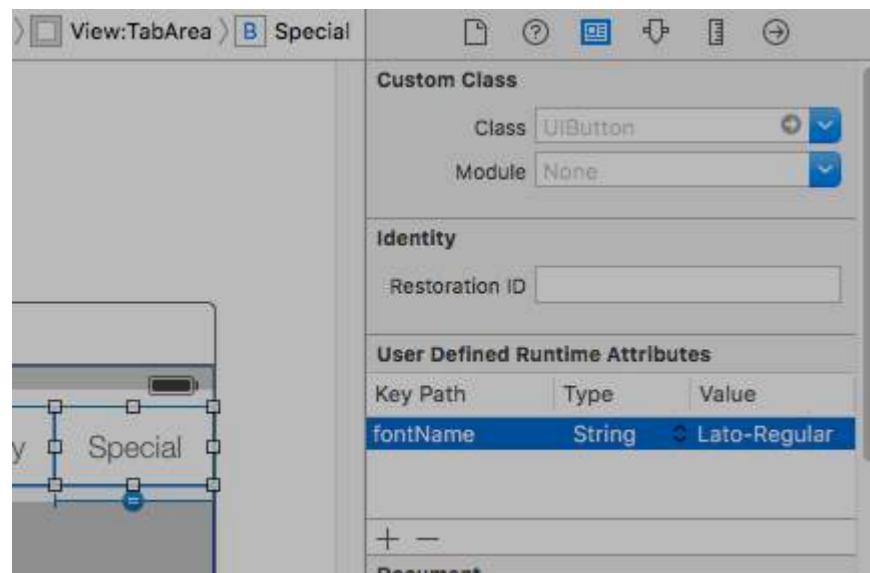
}

```
- (void)setFontName:(NSString *)fontName{
    self.titleLabel.font = [UIFont fontWithName:fontName size:self.titleLabel.font.pointSize];
}
@end
```

}

```
- (void)setFontName:(NSString *)fontName{
    self.titleLabel.font = [UIFont fontWithName:fontName size:self.titleLabel.font.pointSize];
}
@end
```

4. 在Storyboard中设置字体：在用户定义的运行时属性中添加一项，键路径为fontName，值为您的自定义字体名称，类型为字符串，如图所示。
keyPath 和您的 Custom Font's Name 作为值，类型为字符串，如图所示。

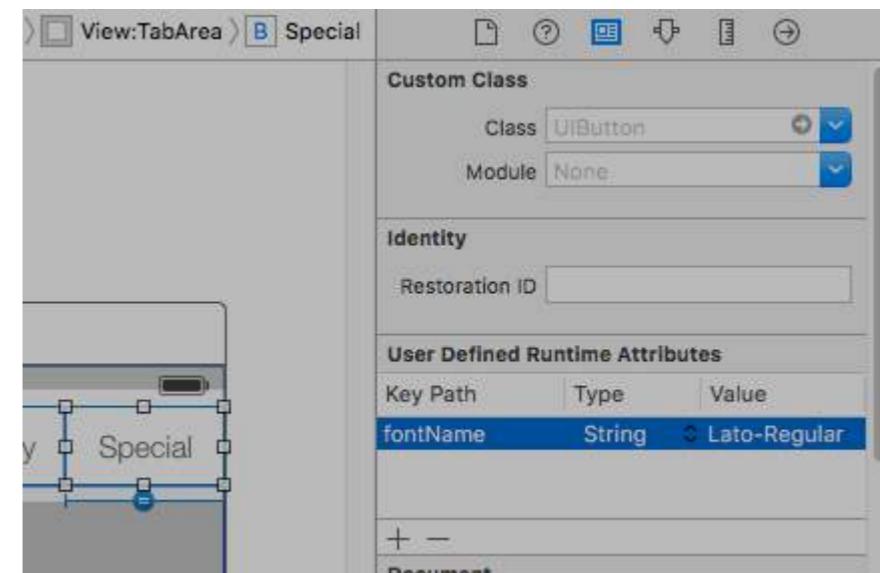


这将在运行应用时设置您的自定义字体。

注意：

- Lato-Regular 是我使用的自定义字体。
- 在捆绑包中添加的 .ttf 文件中使用的相同名称应在故事板中使用，且不带扩展名。
- 字体大小将与 UI 元素属性检查器中定义的大小相同。

4. **Setting font in storyboard:** Add an entry in User Defined Runtime Attributes with **fontName** as keyPath and your **Custom Font's Name** as value with type as String as shown.



This will set your custom font while running the app.

Notes:

- Lato-Regular is the custom font I have used.
- Same name in the **.ttf** file added in bundle should be used without extension in storyboard.
- Font size will be same as it is defined in the UI element's attribute inspector.

第93章：AVSpeechSynthesizer

参数	详细信息
扬声器	AVSpeechSynthesizer 对象
语音	AVSpeechUtterance 对象

第93.1节：创建基本的文本转语音

使用AVSpeechSynthesizer的 speakUtterance: 方法将文本转换为语音。你需要传入一个 AVSpeechUtterance对象给该方法，该对象包含你想要朗读的文本。

Objective C

```
AVSpeechSynthesizer *speaker = [[AVSpeechSynthesizer alloc] init];
AVSpeechUtterance *speech    = [AVSpeechUtterance speechUtteranceWithString:@"Hello World"];
[speaker speakUtterance:speech];
```

Swift

```
let speaker = AVSpeechSynthesizer()
let speech = AVSpeechUtterance(string: "Hello World")
speaker.speakUtterance(speech)
```

Chapter 93: AVSpeechSynthesizer

Parameter	Details
speaker	AVSpeechSynthesizer object
speech	AVSpeechUtterance object

Section 93.1: Creating a basic text to speech

Use the speakUtterance: method of AVSpeechSynthesizer to convert text to speech. You need to pass an AVSpeechUtterance object to this method, which contains the text that you want to be spoken.

Objective C

```
AVSpeechSynthesizer *speaker = [[AVSpeechSynthesizer alloc] init];
AVSpeechUtterance *speech    = [AVSpeechUtterance speechUtteranceWithString:@"Hello World"];
[speaker speakUtterance:speech];
```

Swift

```
let speaker = AVSpeechSynthesizer()
let speech = AVSpeechUtterance(string: "Hello World")
speaker.speakUtterance(speech)
```

第94章：本地化

本地化是iOS提供的功能，可以将你的应用翻译成多种语言。对于本地化，国际化是必要的。国际化是使iOS应用能够适应不同文化、语言和地区的过程。

第94.1节：iOS中的本地化

为每种语言创建一个单独的Localizable.strings文件。右侧内容会因语言不同而不同。可以将其视为键值对：

```
"str" = "str-language";
```

在 Objective-C 中访问字符串：

```
//尝试为本地化字符串提供描述，以便在需要时能够创建适当的文档  
NSString *str = NSLocalizedString(@"string", @"字符串的描述");
```

在 Swift 中访问字符串：

```
let str = NSLocalizedString("string", comment: "语言");
```

Chapter 94: Localization

Localization is feature provided by iOS which translates your app into multiple language. For **Localisation**, **Internationalization** is necessary. **Internationalization** is process of making iOS app able to adapt different culture, language and regions.

Section 94.1: Localization in iOS

Create an individual Localizable.strings file for each language. The right side would be different for each language. Think of it as a key-value pair:

```
"str" = "str-language";
```

Access str in Objective-C:

```
//Try to provide description on the localized string to be able to create a proper documentation if  
needed  
NSString *str = NSLocalizedString(@"string", @"description of the string");
```

Access str in Swift:

```
let str = NSLocalizedString("string", comment: "language");
```

第95章：Alamofire

参数	详细信息
方法	.OPTIONS, .GET, .HEAD, .POST, .PUT, .PATCH, .DELETE, .TRACE, .CONNECT
URL字符串	URLStringConvertible
parameters	[String: AnyObject]?
encoding	ParameterEncoding
headers	[String: String]?

第95.1节：手动验证

```
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])
    .validate(statusCode: 200..<300)
    .validate(contentType: ["application/json"])
    .response { response in
        print(response)
    }
```

第95.2节：自动验证

```
Alamofire.request("https://httpbin.org/get").validate().responseJSON { response in
    switch response.result {
    case .success:
        print("验证成功")
    case .failure(let error):
        print(error)
    }
}
```

第95.3节：链式响应处理器

```
Alamofire.request(.GET, "https://httpbin.org/get")
    .validate()
    .responseString { response in
        print("响应字符串: \(response.result.value)")
    }
    .responseJSON { response in
        print("响应JSON: \(response.result.value)")
    }
```

第95.4节：发起请求

```
import Alamofire

Alamofire.request(.GET, "https://httpbin.org/get")
```

第95.5节：响应处理

```
Alamofire.request(.GET, "https://httpbin.org/get", 参数: ["foo": "bar"])
    .responseJSON { response in
        print(response.request) // 原始URL请求
        print(response.response) // URL响应
        print(response.data) // 服务器数据
        print(response.result) // 响应序列化结果
    }
```

Chapter 95: Alamofire

Parameter	Details
Method	.OPTIONS, .GET, .HEAD, .POST, .PUT, .PATCH, .DELETE, .TRACE, .CONNECT
URLString	URLStringConvertible
parameters	[String: AnyObject]?
encoding	ParameterEncoding
headers	[String: String]?

Section 95.1: Manual Validation

```
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])
    .validate(statusCode: 200..<300)
    .validate(contentType: ["application/json"])
    .response { response in
        print(response)
    }
```

Section 95.2: Automatic Validation

```
Alamofire.request("https://httpbin.org/get").validate().responseJSON { response in
    switch response.result {
    case .success:
        print("Validation Successful")
    case .failure(let error):
        print(error)
    }
}
```

Section 95.3: Chained Response Handlers

```
Alamofire.request(.GET, "https://httpbin.org/get")
    .validate()
    .responseString { response in
        print("Response String: \(response.result.value)")
    }
    .responseJSON { response in
        print("Response JSON: \(response.result.value)")
    }
```

Section 95.4: Making a Request

```
import Alamofire

Alamofire.request(.GET, "https://httpbin.org/get")
```

Section 95.5: Response Handling

```
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])
    .responseJSON { response in
        print(response.request) // original URL request
        print(response.response) // URL response
        print(response.data) // server data
        print(response.result) // result of response serialization
    }
```

```
if let JSON = response.result.value {  
    print("JSON: \(JSON)")  
}  
}
```

第95.6节：响应处理器

```
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])  
.validate()  
.response { request, response, data, error in  
    print(request)  
    print(response)  
    print(data)  
    print(error)  
}
```

```
if let JSON = response.result.value {  
    print("JSON: \(JSON)")  
}  
}
```

Section 95.6: Response Handler

```
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])  
.validate()  
.response { request, response, data, error in  
    print(request)  
    print(response)  
    print(data)  
    print(error)  
}
```

第96章：iBeacon

参数	详细信息
管理器	CLLocationManager 引用
区域	CLRegion 可以是圆形区域（地理围栏或信标区域）
信标	包含所有检测到信标的 CLBeacon 数组

第96.1节：iBeacon 基本操作

1. 设置监测信标

```
func initiateRegion(ref:BeaconHandler){  
    let uuid: NSUUID = NSUUID(UUIDString: "<UUID>")  
    let beacon = CLBeaconRegion(proximityUUID: uuid, identifier: "")  
    locationManager?.requestAlwaysAuthorization() //CLLocationManager obj.  
beacon?.notifyOnEntry = true  
beacon?.notifyOnExit = true  
beacon?.notifyEntryStateOnDisplay = true  
locationManager?.startMonitoringForRegion(beacon!)  
locationManager?.delegate = self;  
// 检查此设备是否支持信标监测  
if (!CLLocationManager.isMonitoringAvailableForClass(CLBeaconRegion)) {  
    print("error")  
}  
locationManager!.startRangingBeaconsInRegion(self.beacon!)  
}
```

2. 位置管理器进入和退出区域

```
func locationManager(manager: CLLocationManager, didEnterRegion region: CLRegion) {  
    if(region.isKindOfClass(CLBeaconRegion)) {  
locationManager!.startRangingBeaconsInRegion(self.beacon!)  
    }  
}  
  
func locationManager(manager: CLLocationManager, didExitRegion region: CLRegion) {  
    if(region.isKindOfClass(CLBeaconRegion)) {  
locationManager!.stopRangingBeaconsInRegion(self.beacon!)  
    }  
}
```

3. 位置管理器信标测距

```
func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion region: CLBeaconRegion) {  
    print(beacons.first.major)  
}
```

第96.2节：测距iBeacon

首先，您必须请求定位服务的授权

```
let locationManager = CLLocationManager()  
locationManager.delegate = self  
locationManager.requestWhenInUseAuthorization()  
// 或 locationManager.requestAlwaysAuthorization()
```

Chapter 96: iBeacon

Parameters	Details
manager	CLLocationManager reference
region	CLRegion could be circular region (geofence or beacon region)
beacons	Array of CLBeacon contains all ranged beacons

Section 96.1: iBeacon Basic Operation

1. Setup monitoring beacons

```
func initiateRegion(ref:BeaconHandler){  
    let uuid: NSUUID = NSUUID(UUIDString: "<UUID>")  
    let beacon = CLBeaconRegion(proximityUUID: uuid, identifier: "")  
    locationManager?.requestAlwaysAuthorization() //CLLocationManager obj.  
beacon?.notifyOnEntry = true  
beacon?.notifyOnExit = true  
beacon?.notifyEntryStateOnDisplay = true  
locationManager?.startMonitoringForRegion(beacon!)  
locationManager?.delegate = self;  
// Check if beacon monitoring is available for this device  
if (!CLLocationManager.isMonitoringAvailableForClass(CLBeaconRegion)) {  
    print("error")  
}  
locationManager!.startRangingBeaconsInRegion(self.beacon!)  
}
```

2. Location manager enter and exit region

```
func locationManager(manager: CLLocationManager, didEnterRegion region: CLRegion) {  
    if(region.isKindOfClass(CLBeaconRegion)) {  
        locationManager!.startRangingBeaconsInRegion(self.beacon!)  
    }  
}  
  
func locationManager(manager: CLLocationManager, didExitRegion region: CLRegion) {  
    if(region.isKindOfClass(CLBeaconRegion)) {  
        locationManager!.stopRangingBeaconsInRegion(self.beacon!)  
    }  
}
```

3. Location manager range beacon

```
func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion region: CLBeaconRegion) {  
    print(beacons.first.major)  
}
```

Section 96.2: Ranging iBeacons

First, you have to request authorization of location services

```
let locationManager = CLLocationManager()  
locationManager.delegate = self  
locationManager.requestWhenInUseAuthorization()  
// OR locationManager.requestAlwaysAuthorization()
```

然后，您可以在 `didRangeBeacons` 中获取所有iBeacon的信息

```
func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion region: CLBeaconRegion) {
    for beacon in beacons {
        print(beacon.major)
        print(beacon.minor)
    }
}
```

第96.3节：扫描特定的Beacon

```
beacon = CLBeaconRegion(proximityUUID: <#NSUUID#>, major: <#CLBeaconMajorValue#>, identifier: <#String#>) // 监听所有具有指定UUID和主值的信标
beacon = CLBeaconRegion(proximityUUID: <##NSUUID#>, major: <##CLBeaconMajorValue#>, minor: <##CLBeaconMinorValue#>, identifier: <##String#>) // 监听所有具有指定UUID和主值及次值的信标
```

Then you can get all iBeacons' information inside `didRangeBeacons`

```
func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion region: CLBeaconRegion) {
    for beacon in beacons {
        print(beacon.major)
        print(beacon.minor)
    }
}
```

Section 96.3: Scanning specific Beacons

```
beacon = CLBeaconRegion(proximityUUID: <#NSUUID#>, major: <#CLBeaconMajorValue#>, identifier: <#String#>) // listening to all beacons with given UUID and major value
beacon = CLBeaconRegion(proximityUUID: <##NSUUID#>, major: <##CLBeaconMajorValue#>, minor: <##CLBeaconMinorValue#>, identifier: <##String#>) // listening to all beacons with given UUID and major and minor value
```

第97章：CLLocation

第97.1节：使用距离过滤器

示例：

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
locationManager.delegate = self;
locationManager.desiredAccuracy = kCLLocationAccuracyBest;
locationManager.distanceFilter = 5;
```

例如：在上述示例代码中，位置变化小于5米时不会发送回调，而是被忽略。

第97.2节：使用CLLocationManager获取用户位置

1 - 在项目中包含 CoreLocation.framework；操作方法是点击：

根目录 -> 构建阶段 -> 连接二进制与库

点击 (+) 按钮，查找 CoreLocation.framework 并点击添加。

2- 修改 info.plist 文件以请求使用用户位置的权限，方法是以源代码形式打开它。在标签下添加以下任一键值对，以请求应用程序使用时访问用户位置的权限：

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>请求权限时显示的消息</string>
```

3- 在将使用 CoreLocation 的 ViewController 中导入 CoreLocation。

```
import CoreLocation
```

4- 确保你的 ViewController 遵循 CLLocationManagerDelegate 协议

```
class ViewController: UIViewController, CLLocationManagerDelegate {}
```

完成以上步骤后，我们可以创建一个 CLLocationManager 对象作为实例变量，并在 ViewController 中使用它。

```
var manager: CLLocationManager!
```

我们这里不使用 'let'，因为我们将修改管理器以指定其代理、更新事件前的最小距离以及其精度

```
//初始化管理器
manager = CLLocationManager()

//指定代理
manager.delegate = self

//设置手机在触发更新事件前需要移动的最小距离（例如：100米）
manager.distanceFilter = 100

//根据您的使用情况设置精度为以下任意值
```

Chapter 97: CLLocation

Section 97.1: Distance Filter using

Example :

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
locationManager.delegate = self;
locationManager.desiredAccuracy = kCLLocationAccuracyBest;
locationManager.distanceFilter = 5;
```

E.g. In the above example code above, location changes of less than 5 metres won't be sent to the callback, but instead be ignored.

Section 97.2: Get User Location Using CLLocationManager

1 - Include the CoreLocation.framework in your project; this is accomplished by clicking on:

root directory -> build phases -> Link Binary With Libraries

Click on the (+) button, look for CoreLocation.framework and click add.

2- Modify the info.plist file to ask for permission to use user location by opening it as source code. Add either of the following key:value pair under the tag to ask for usage of user's location while the application is in use:

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>message to display when asking for permission</string>
```

3- import CoreLocation to the ViewController that will be using it.

```
import CoreLocation
```

4- Make sure your ViewController conforms to the CLLocationManagerDelegate protocol

```
class ViewController: UIViewController, CLLocationManagerDelegate {}
```

After these steps, we can create a CLLocationManager object as instance variable and use it in the ViewController.

```
var manager: CLLocationManager!
```

We do not use 'let' here because we will modify the manager to specify its delegate, minimum distance before update event, and its accuracy

```
//initialize the manager
manager = CLLocationManager()

//specify delegate
manager.delegate = self

//set the minimum distance the phone needs to move before an update event is triggered (for example: 100 meters)
manager.distanceFilter = 100

//set Accuracy to any of the following depending on your use case
```

```

//let kCLLocationAccuracyBestForNavigation: CLLocationAccuracy
//let kCLLocationAccuracyBest: CLLocationAccuracy
//let kCLLocationAccuracyNearestTenMeters: CLLocationAccuracy
//let kCLLocationAccuracyHundredMeters: CLLocationAccuracy
//let kCLLocationAccuracyKilometer: CLLocationAccuracy
//let kCLLocationAccuracyThreeKilometers: CLLocationAccuracy

manager.desiredAccuracy = kCLLocationAccuracyBest

//请求用户授权
manager.requestWhenInUseAuthorization()

//开始收集位置信息
if #available(iOS 9.0, *) {

manager.requestLocation()

} else {

manager.startUpdatingLocation()

}

```

现在，为了获取位置更新，我们可以实现下面的函数，该函数会在达到 distanceFilter 时被调用。

```
func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {}
```

locations 参数是一个 CLLocation 对象数组，表示设备的实际位置。通过这些对象，可以访问以下属性：coordinate（坐标）、altitude（海拔）、floor（楼层）、horizontalAccuracy（水平精度）、verticalAccuracy（垂直精度）、timestamp（时间戳）、description（描述）、course（航向）、speed（速度），以及一个函数 distance(from:)，用于测量两个位置之间的距离。

注意：请求位置权限时，有两种不同类型的授权。

“使用时”授权仅允许应用在使用中或前台时接收您的位置。

“始终”授权则赋予应用后台权限，这可能会导致在应用关闭时电池寿命减少。

需要根据情况调整 Plist 文件。

```

//let kCLLocationAccuracyBestForNavigation: CLLocationAccuracy
//let kCLLocationAccuracyBest: CLLocationAccuracy
//let kCLLocationAccuracyNearestTenMeters: CLLocationAccuracy
//let kCLLocationAccuracyHundredMeters: CLLocationAccuracy
//let kCLLocationAccuracyKilometer: CLLocationAccuracy
//let kCLLocationAccuracyThreeKilometers: CLLocationAccuracy

manager.desiredAccuracy = kCLLocationAccuracyBest

//ask the user for permission
manager.requestWhenInUseAuthorization()

//Start collecting location information
if #available(iOS 9.0, *) {

    manager.requestLocation()

} else {

    manager.startUpdatingLocation()

}

```

Now to get access to the location updates, we can implement the function below which is called overtime the distanceFilter is reached.

```
func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {}
```

The locations parameter is an array of CLLocation objects that represent the actual location of the device. From these objects, one can get access to the following attributes: coordinate, altitude, floor, horizontalAccuracy, verticalAccuracy, timestamp, description, course, speed, and a function `distance(from:)` that measures the distance between two locations.

Note: While requesting permission for location, there are two different types of authorization.

“When In Use” authorization only gives the app permission to receive your location when the app is in use or foreground.

“Always” authorization, gives the app background permissions which may lead to decrease battery life in case your app is closed.

Plist file should be adjusted as necessary.

第98章：检查iOS版本

第98.1节：iOS 8及更高版本

Swift 3：

```
let minimumVersion = OperatingSystemVersion(majorVersion: 8, minorVersion: 1, patchVersion: 2)
if ProcessInfo().isOperatingSystemAtLeast(minimumVersion) {
    //当前版本 >= (8.1.2)
} else {
    //当前版本 < (8.1.2)
}
```

第98.2节：Swift 2.0及更高版本

```
if #available(iOS 9, *) {
    // iOS 9
} else {
    // iOS 8或更早版本
}
```

第98.3节：版本比较

```
let minimumVersionString = "3.1.3"
let versionComparison = UIDevice.current.systemVersion.compare(minimumVersionString, options: .numeric)
switch versionComparison {
    case .orderedSame, .orderedDescending:
        //当前版本 >= (3.1.3)
        break
    case .orderedAscending:
        //当前版本 < (3.1.3)
        fallthrough
    default:
        break;
}
```

Objective-C

```
NSString *version = @"3.1.3";
NSString *currentVersion = @"3.1.1";
NSComparisonResult result = [currentVersion compare:version options:NSNumericSearch];
switch(result){
    case: NSOrderedAscending:
        // 小于当前版本
        break;
    case: NSOrderedDescending:
    case: NSOrderedSame:
        // 等于或大于当前版本
        break;
}
```

第98.4节：设备iOS版本

这将返回当前系统版本。

Objective-C

Chapter 98: Checking iOS version

Section 98.1: iOS 8 and later

Swift 3:

```
let minimumVersion = OperatingSystemVersion(majorVersion: 8, minorVersion: 1, patchVersion: 2)
if ProcessInfo().isOperatingSystemAtLeast(minimumVersion) {
    //current version is >= (8.1.2)
} else {
    //current version is < (8.1.2)
}
```

Section 98.2: Swift 2.0 and later

```
if #available(iOS 9, *) {
    // iOS 9
} else {
    // iOS 8 or earlier
}
```

Section 98.3: Compare versions

```
let minimumVersionString = "3.1.3"
let versionComparison = UIDevice.current.systemVersion.compare(minimumVersionString, options: .numeric)
switch versionComparison {
    case .orderedSame, .orderedDescending:
        //current version is >= (3.1.3)
        break
    case .orderedAscending:
        //current version is < (3.1.3)
        fallthrough
    default:
        break;
}
```

Objective-C

```
NSString *version = @"3.1.3";
NSString *currentVersion = @"3.1.1";
NSComparisonResult result = [currentVersion compare:version options:NSNumericSearch];
switch(result){
    case: NSOrderedAscending:
        //less than the current version
        break;
    case: NSOrderedDescending:
    case: NSOrderedSame:
        // equal or greater than the current version
        break;
}
```

Section 98.4: Device iOS Version

This will give current system version.

Objective-C

```
NSString *version = [[UIDevice currentDevice] systemVersion]
```

Swift

```
let version = UIDevice.currentDevice().systemVersion
```

Swift 3

```
let version = UIDevice.current.systemVersion
```

```
NSString *version = [[UIDevice currentDevice] systemVersion]
```

Swift

```
let version = UIDevice.currentDevice().systemVersion
```

Swift 3

```
let version = UIDevice.current.systemVersion
```

第99章：通用链接

第99.1节：设置iOS应用程序（启用通用链接）

应用程序端的设置需要两项内容：

1. 配置应用程序的权限，并通过开启项目中的关联域功能来启用通用链接能力。
2. 在你的AppDelegate中处理传入的链接。

1. 配置应用程序的权限，并启用通用链接。

配置应用程序权限的第一步是为您的应用程序标识符（App ID）启用该权限。请在 Apple 开发者会员中心进行操作。点击“证书、标识符与描述文件”，然后选择“标识符”。选择您的应用程序标识符（如有需要，请先创建），点击“编辑”，并启用“关联域名”权限。

ID: com.Universal-Links		
Application Services:		
Service	Development	Distribution
App Group	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Associated Domains	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
Data Protection	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Game Center	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
HealthKit	<input type="radio"/> Disabled	<input type="radio"/> Disabled
HomeKit	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Wireless Accessory Configuration	<input type="radio"/> Disabled	<input type="radio"/> Disabled
iCloud	<input type="radio"/> Disabled	<input type="radio"/> Disabled
In-App Purchase	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
Inter-App Audio	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Apple Pay	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Wallet	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Push Notifications	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Personal VPN	<input type="radio"/> Disabled	<input type="radio"/> Disabled

Chapter 99: Universal Links

Section 99.1: Setup iOS Application (Enabling Universal Links)

The setup on the app side requires two things:

1. Configuring the app's entitlement, and enabling the universal links by turning on the Associated Domains capability in the project.
2. Handling Incoming Links in your AppDelegate.

1. Configuring the app's entitlement, and enabling universal links.

The first step in configuring your app's entitlements is to enable it for your App ID. Do this in the Apple Developer Member Center. Click on Certificates, Identifiers & Profiles and then Identifiers. Select your App ID (create it first if required), click Edit and enable the Associated Domains entitlement.

ID: com.Universal-Links		
Application Services:		
Service	Development	Distribution
App Group	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Associated Domains	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
Data Protection	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Game Center	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
HealthKit	<input type="radio"/> Disabled	<input type="radio"/> Disabled
HomeKit	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Wireless Accessory Configuration	<input type="radio"/> Disabled	<input type="radio"/> Disabled
iCloud	<input type="radio"/> Disabled	<input type="radio"/> Disabled
In-App Purchase	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
Inter-App Audio	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Apple Pay	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Wallet	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Push Notifications	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Personal VPN	<input type="radio"/> Disabled	<input type="radio"/> Disabled

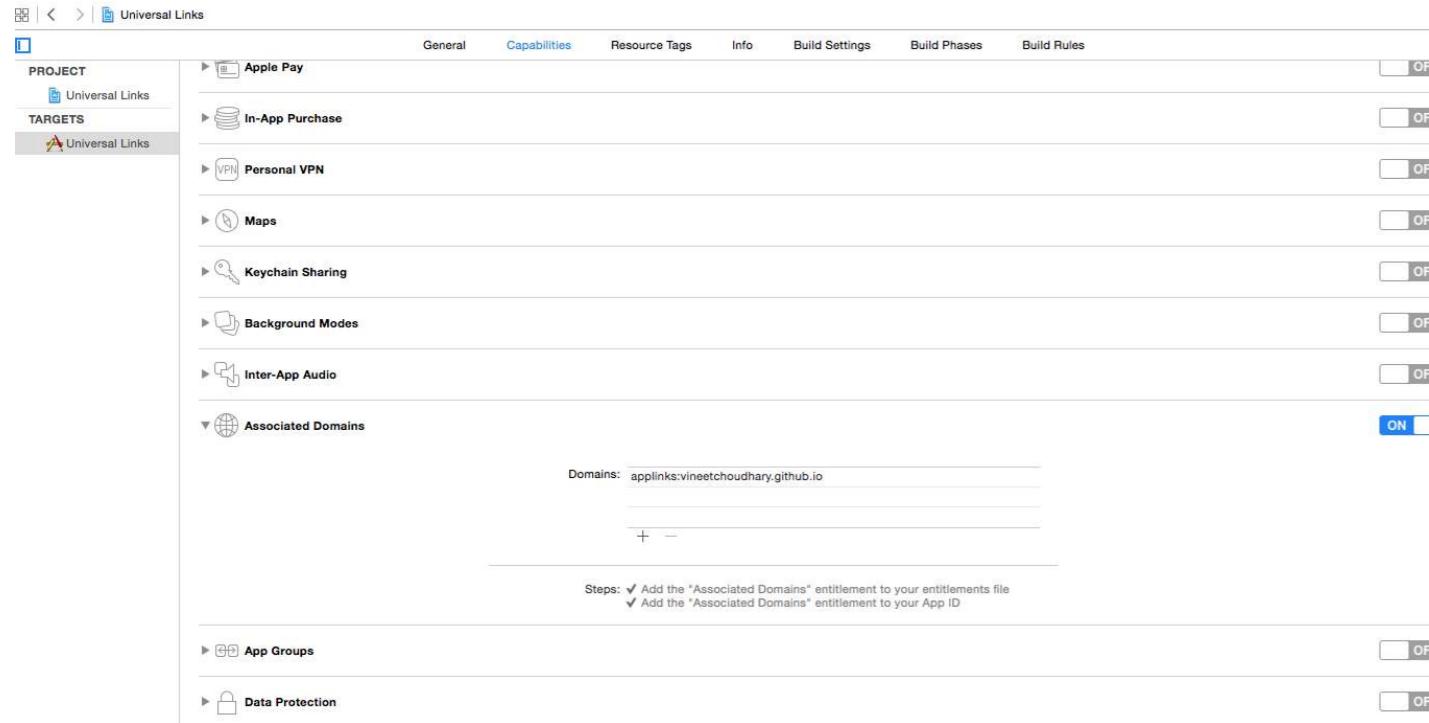
接下来，通过点击相应的 App ID 获取 App ID 的前缀和后缀。

App ID 的前缀和后缀应与 apple-app-site-association 文件中的一致。

接下来，在Xcode中，选择您的应用的目标，点击功能（Capabilities），并将关联域（Associated Domains）切换为开启。为您的应用支持的每个域添加一条条目，前缀为app links:

例如`applinks:YourCustomDomainName.com`

对于示例应用，看起来像这样：



注意：确保您选择了相同的团队，并输入了与会员中心注册的 App ID 相同的 Bundle ID。还要确保 Xcode 包含了权限文件，方法是选中该文件，在文件检查器（File Inspector）中确认已勾选您的目标。

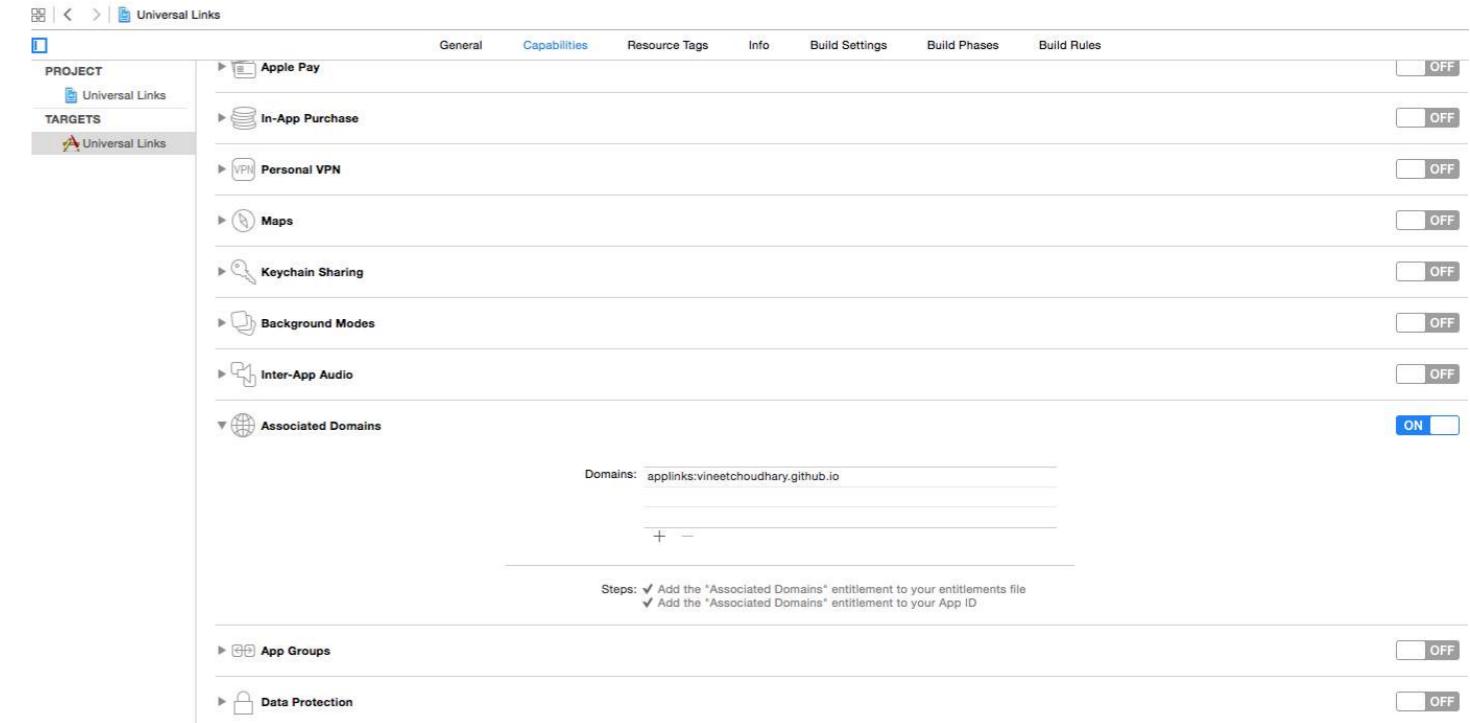
Next, get the App ID prefix and suffix by clicking on the respective App ID.

The App ID prefix and suffix should match the one in the apple-app-site-association file.

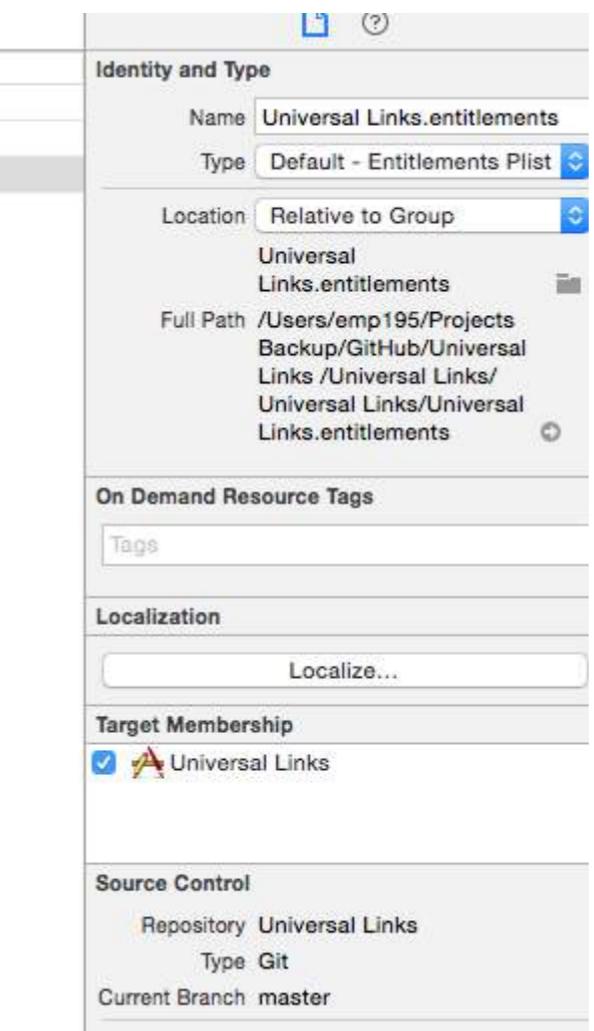
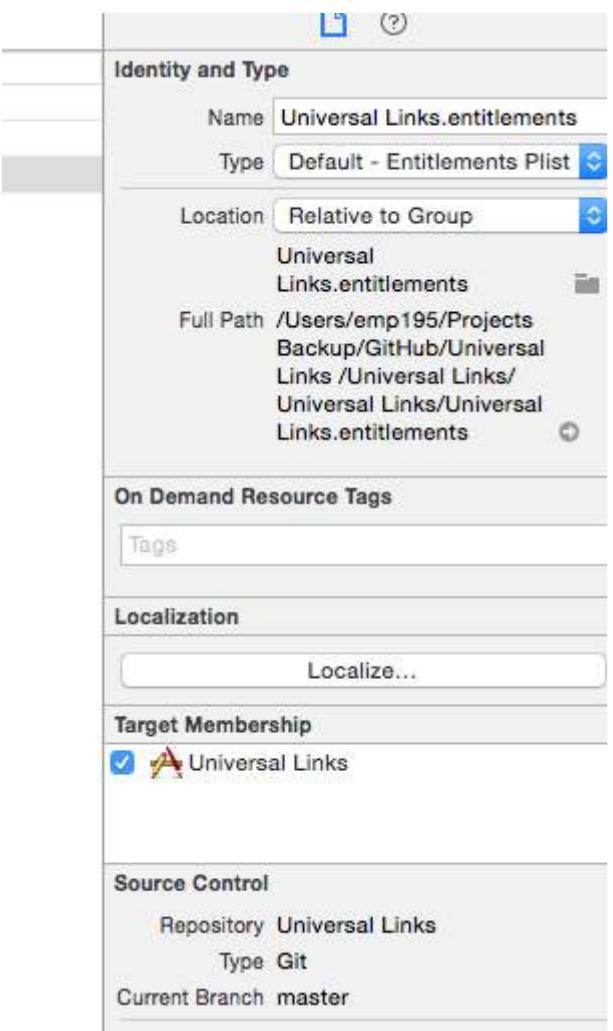
Next in Xcode, select your App's target, click Capabilities and toggle Associated Domains to On. Add an entry for each domain that your app supports, prefixed with **app links**:

For example `applinks:YourCustomDomainName.com`

Which looks like this for the sample app:



Note: Ensure you have selected the same team and entered the same Bundle ID as the registered App ID on the Member Center. Also ensure that the entitlements file is included by Xcode by selecting the file and in the File Inspector, ensure that your target is checked.



2. 在您的 AppDelegate 中处理传入链接

所有从 Safari 重定向到应用的通用链接都会通过应用的 AppDelegate 类中的以下方法。您需要解析此 URL 以确定应用中的正确操作。

[UIApplicationDelegate application: continueUserActivity: restorationHandler:]

Objective-C

```
- (BOOL)application:(UIApplication *)application continueUserActivity:(NSUserActivity *)userActivity
restorationHandler:(void (^)(NSArray * _Nullable))restorationHandler{
    //检查活动是否来自网页重定向到应用。
    if ([userActivity.activityType isEqualToString: NSUserActivityTypeBrowsingWeb]) {
        //从UserActivity对象获取URL。
        NSURL *url = userActivity.webpageURL;
        UIStoryboard *storyBoard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
        UINavigationController *navigationController = (UINavigationController
*)_window.rootViewController;
        if ([url.pathComponents containsObject:@"home"]){
            [navigationController pushViewController:[storyBoard
instantiateViewControllerWithIdentifier:@"HomeScreenId"] animated:YES];
        }else if ([url.pathComponents containsObject:@"about"]){
            [navigationController pushViewController:[storyBoard
instantiateViewControllerWithIdentifier:@"AboutScreenId"] animated:YES];
        }
    }
    return YES;
}
```

Swift :

```
func application(application: UIApplication, continueUserActivity userActivity: NSUserActivity,
```

2. Handling Incoming Links in your AppDelegate

All redirects from Safari to the app for universal links go via the below method in the Application's AppDelegate class. You parse this URL to determine the right action in the app.

[UIApplicationDelegate application: continueUserActivity: restorationHandler:]

Objective-C

```
- (BOOL)application:(UIApplication *)application continueUserActivity:(NSUserActivity *)userActivity
restorationHandler:(void (^)(NSArray * _Nullable))restorationHandler{
    //Checking whether the activity was from a web page redirect to the app.
    if ([userActivity.activityType isEqualToString: NSUserActivityTypeBrowsingWeb]) {
        //Getting the URL from the UserActivity Object.
        NSURL *url = userActivity.webpageURL;
        UIStoryboard *storyBoard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
        UINavigationController *navigationController = (UINavigationController
*)_window.rootViewController;
        if ([url.pathComponents containsObject:@"home"]){
            [navigationController pushViewController:[storyBoard
instantiateViewControllerWithIdentifier:@"HomeScreenId"] animated:YES];
        }else if ([url.pathComponents containsObject:@"about"]){
            [navigationController pushViewController:[storyBoard
instantiateViewControllerWithIdentifier:@"AboutScreenId"] animated:YES];
        }
    }
    return YES;
}
```

Swift :

```
func application(application: UIApplication, continueUserActivity userActivity: NSUserActivity,
```

```

restorationHandler: ([AnyObject]?) -> Void) -> Bool {
    if userActivity.activityType == NSUserActivityTypeBrowsingWeb {
        let url = userActivity.webpageURL!
        //handle url
    }
    返回 true
}

```

iOS 应用代码

应用代码可以在 master 分支 [here](#) 找到。

第 99.2 节：支持多个域名

应用中支持的每个域名都需要提供其自己的 apple-app-site-association 文件。如果每个域名提供的内容不同，那么文件内容也会相应更改以支持各自的路径。否则，可以使用相同的文件，但该文件需要在每个支持的域名上都能访问。

第 99.3 节：为 App-Site-Association 文件签名

注意: 如果您的服务器使用HTTPS提供内容，可以跳过此部分，直接进入应用程序设置指南。

如果您的应用程序针对的是iOS 9，且服务器使用HTTPS提供内容，则无需对文件进行签名。如果不是（例如在支持iOS 8的Handoff时），则必须使用受认可的证书颁发机构颁发的SSL证书对其进行签名。

注意: 这不是Apple提供用于提交应用到App Store的证书。它应由第三方提供，建议使用与您的HTTPS服务器相同的证书（尽管这不是必需的）。

要签名文件，首先创建并保存一个简单的.txt版本。接着，在终端运行以下命令：

```
cat <unsigned_file>.txt | openssl smime -sign -inkey example.com.key -signer example.com.pem -certfile intermediate.pem -noattr -nodetach -outform DER > apple-app-site-association
```

这将在当前目录输出签名后的文件。example.com.key、example.com.pem和intermediate.pem是由您的证书颁发机构提供的文件。

注意: 如果文件未签名，其Content-Type应为application/json。否则，应为application/pkcs7-mime。

使用Apple应用搜索验证工具验证您的服务器

测试您的网页以支持iOS 9搜索API。输入URL，Applebot将爬取您的网页并展示如何优化以获得最佳效果 <https://search.developer.apple.com/appsearch-validation-tool/>

第99.4节：设置服务器

您需要有一台在线运行的服务器。为了安全地将您的 iOS 应用与服务器关联，苹果要求您提供一个配置文件，称为apple-app-site-association。该文件是一个JSON文件，描述了域名和支持的路由。

apple-app-site-association 文件需要通过HTTPS访问，且不能有任何重定向，路径为<https://domain/apple-app-site-association>。

```

restorationHandler: ([AnyObject]?) -> Void) -> Bool {
    if userActivity.activityType == NSUserActivityTypeBrowsingWeb {
        let url = userActivity.webpageURL!
        //handle url
    }
    return true
}

```

iOS Application Code

The app code can be found master branch [here](#).

Section 99.2: Supporting Multiple Domains

Each domain supported in the app needs to make available its own apple-app-site-association file. If the content served by each domain is different, then the contents of the file will also change to support the respective paths. Otherwise, the same file can be used, but it needs to be accessible at every supported domain.

Section 99.3: Signing the App-Site-Association File

Note: You can skip this part if your server uses HTTPS to serve content and jump to Application Setup guide.

If your app targets iOS 9 and your server uses HTTPS to serve content, you don't need to sign the file. If not (e.g. when supporting Handoff on iOS 8), it has to be signed using a SSL certificate from a recognized certificate authority.

Note: This is not the certificate provided by Apple to submit your app to the App Store. It should be provided by a third-party, and it's recommended to use the same certificate you use for your HTTPS server (although it's not required).

To sign the file, first create and save a simple .txt version of it. Next, in the terminal, run the following command:

```
cat <unsigned_file>.txt | openssl smime -sign -inkey example.com.key -signer example.com.pem -certfile intermediate.pem -noattr -nodetach -outform DER > apple-app-site-association
```

This will output the signed file in the current directory. The example.com.key, example.com.pem, and intermediate.pem are the files that would made available to you by your Certifying Authority.

Note: If the file is unsigned, it should have a Content-Type of application/json. Otherwise, it should be application/pkcs7-mime.

Validate your Server with Apple App search validation tool

Test your webpage for iOS 9 Search APIs. Enter a URL and Applebot will crawl your webpage and show how you can optimize for best results <https://search.developer.apple.com/appsearch-validation-tool/>

Section 99.4: Setup Server

You need to having a server running online. To securely associate your iOS app with a server, Apple requires that you make available a configuration file, called apple-app-site-association. This is a JSON file which describes the domain and supported routes.

The apple-app-site-association file needs to be accessible via HTTPS, without any redirects, at <https://domain/apple-app-site-association>.

该文件内容如下：

```
{  
  "applinks": {  
    "apps": [ ],  
    "details": [  
      {  
        "appID": "{app_prefix}.{app_identifier}",  
        "paths": [ "/path/to/content", "/path/to/other/*", "NOT /path/to/exclude" ]  
      },  
      {  
        "appID": "TeamID.BundleID2",  
        "paths": [ "*" ]  
      }  
    ]  
  }  
}
```

注意 - 不要在apple-app-site-association文件名后追加.json。

键如下：

apps：其值应为空数组，且必须存在。这是苹果的要求。

details：是一个字典数组，每个字典对应网站支持的一个iOS应用。每个字典包含关于该应用、团队和捆绑ID的信息。

定义路径有三种方式：

静态：整个支持的路径是硬编码的，用于识别特定链接，例如 /static/terms

通配符：可以使用 * 来匹配动态路径，例如 /books/* 可以匹配任何作者's 页面的路径。?

在特定路径组件内，例如 books/1? 可用于匹配任何ID以1开头的书籍。

排除：在路径前加上 NOT 表示排除该路径的匹配。

路径在数组中出现的顺序很重要。索引越靠前优先级越高。一旦路径匹配，评估停止，其他路径被忽略。每个路径区分大小写。

#网站代码

网站代码可以在 gh-pages 分支的

<https://github.com/vineetchoudhary/iOS-Universal-Links/tree/gh-pages> 找到

The file looks like this:

```
{  
  "applinks": {  
    "apps": [ ],  
    "details": [  
      {  
        "appID": "{app_prefix}.{app_identifier}",  
        "paths": [ "/path/to/content", "/path/to/other/*", "NOT /path/to/exclude" ]  
      },  
      {  
        "appID": "TeamID.BundleID2",  
        "paths": [ "*" ]  
      }  
    ]  
  }  
}
```

NOTE - Don't append .json to the apple-app-site-association filename.

The keys are as follows:

apps: Should have an empty array as its value, and it must be present. This is how Apple wants it.

details: Is an array of dictionaries, one for each iOS app supported by the website. Each dictionary contains information about the app, the team and bundle IDs.

There are 3 ways to define paths:

Static: The entire supported path is hardcoded to identify a specific link, e.g. /static/terms

Wildcards: A * can be used to match dynamic paths, e.g. /books/* can matches the path to any author's page. ? inside specific path components, e.g. books/1? can be used to match any books whose ID starts with 1.

Exclusions: Prepending a path with NOT excludes that path from being matched.

The order in which the paths are mentioned in the array is important. Earlier indices have higher priority. Once a path matches, the evaluation stops, and other paths ignored. Each path is case-sensitive.

#Website Code

The website code can be found gh-pages branch on

<https://github.com/vineetchoudhary/iOS-Universal-Links/tree/gh-pages>

第100章：iOS中的PDF创建

第100.1节：创建PDF

```
UIGraphicsBeginPDFContextToFile(fileName, CGRectMakeZero, nil);
UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 612, 792), nil);
[self drawText];
UIGraphicsEndPDFContext();
```

fileName 是您将要追加或附加的文档文件

```
NSString* temporaryFile = @"firstIOS.PDF";
NSArray *arrayPaths =
NSSearchPathForDirectoriesInDomains(
    NSDocumentDirectory,
    NSUserDomainMask,
    YES);

NSString *path = [arrayPaths objectAtIndex:0];
NSString* fileName = [path stringByAppendingPathComponent:fileName];
```

其中 drawText 是

```
(void)drawText
{
    NSString* textToDraw = @"Lorem Ipsum 是印刷和排版行业的虚拟文本。自1500年代以来，Lorem Ipsum一直是该
行业的标准虚拟文本，当时一位未知的印刷工人拿起一排字母并将其打乱，制作了一本字体样本书。";

    CFStringRef stringRef = (_bridge CFStringRef)textToDraw;

    CFAAttributedStringRef currentText = CFAAttributedStringCreate(NULL, stringRef, NULL);

    CTFramesetterRef framesetter = CTFramesetterCreateWithAttributedString(currentText);

    CGRect frameRect = CGRectMake(0, 0, 300, 100);

    CGMutablePathRef framePath = CGPathCreateMutable();

    CGPathAddRect(framePath, NULL, frameRect);

    CFRange currentRange = CFRangeMake(0, 0);

    CTFrameRef frameRef = CTFramesetterCreateFrame(framesetter, currentRange, framePath, NULL);
    CGPathRelease(framePath);

    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    CGContextSetTextMatrix(currentContext, CGAffineTransformIdentity);

    CGContextTranslateCTM(currentContext, 0, 450);
    CGContextScaleCTM(currentContext, 2, -2);
```

Chapter 100: PDF Creation in iOS

Section 100.1: Create PDF

```
UIGraphicsBeginPDFContextToFile(fileName, CGRectMakeZero, nil);
UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 612, 792), nil);
[self drawText];
UIGraphicsEndPDFContext();
```

fileName is the document file where You are going to append or attach

```
NSString* temporaryFile = @"firstIOS.PDF";
NSArray *arrayPaths =
NSSearchPathForDirectoriesInDomains(
    NSDocumentDirectory,
    NSUserDomainMask,
    YES);

NSString *path = [arrayPaths objectAtIndex:0];
NSString* fileName = [path stringByAppendingPathComponent:fileName];
```

Where drawText is

```
(void)drawText
{
    NSString* textToDraw = @"Lorem Ipsum is simply dummy text of the printing and typesetting
industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an
unknown printer took a galley of type and scrambled it to make a type specimen book.';

    CFStringRef stringRef = (_bridge CFStringRef)textToDraw;

    CFAAttributedStringRef currentText = CFAAttributedStringCreate(NULL, stringRef, NULL);

    CTFramesetterRef framesetter = CTFramesetterCreateWithAttributedString(currentText);

    CGRect frameRect = CGRectMake(0, 0, 300, 100);

    CGMutablePathRef framePath = CGPathCreateMutable();

    CGPathAddRect(framePath, NULL, frameRect);

    CFRange currentRange = CFRangeMake(0, 0);

    CTFrameRef frameRef = CTFramesetterCreateFrame(framesetter, currentRange, framePath, NULL);
    CGPathRelease(framePath);

    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    CGContextSetTextMatrix(currentContext, CGAffineTransformIdentity);

    CGContextTranslateCTM(currentContext, 0, 450);
    CGContextScaleCTM(currentContext, 2, -2);
```

```

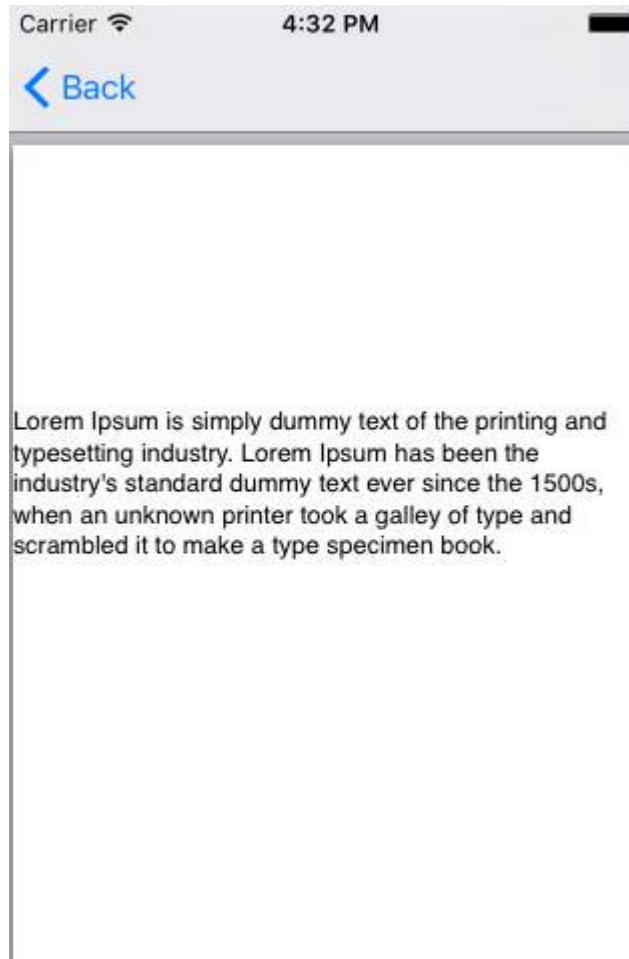
CTFrameDraw(frameRef, currentContext);

CFRelease(frameRef);

CFRelease(stringRef);

CFRelease(framesetter);
}

```



第100.2节：显示PDF

```

NSString* fileName = @"firstIOS.PDF";

NSArray *arrayPaths =
NSSearchPathForDirectoriesInDomains(
    NSDocumentDirectory,
    NSUserDomainMask,
    YES);

NSString *path = [arrayPaths objectAtIndex:0];

NSString* pdfFileName = [path stringByAppendingPathComponent:fileName];

UIWebView* webView = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)];

NSURL *url = [NSURL fileURLWithPath:pdfFileName];

NSURLRequest *request = [NSURLRequest requestWithURL:url];

```

```

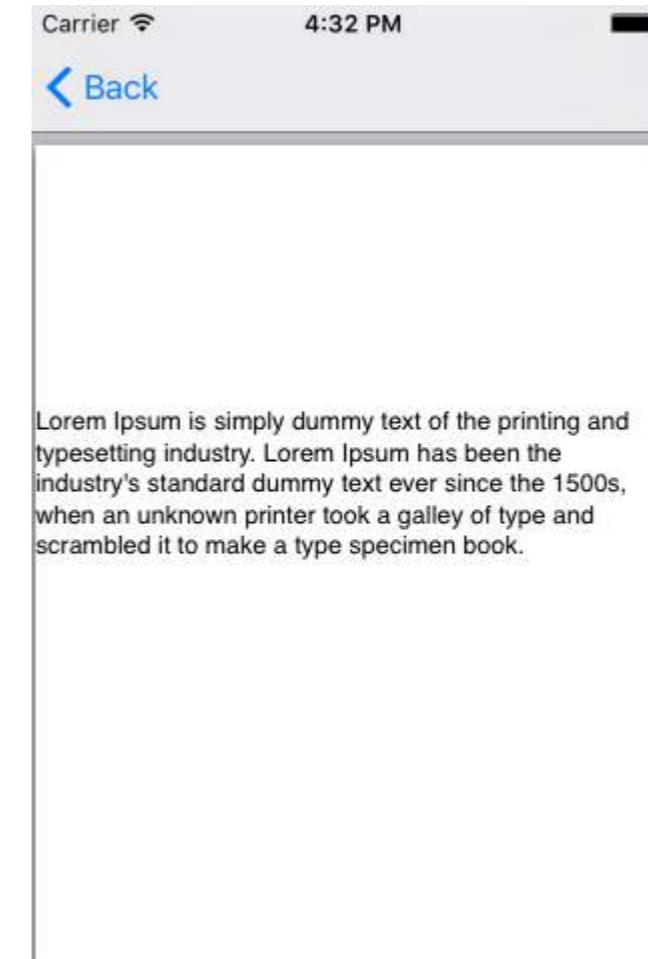
CTFrameDraw(frameRef, currentContext);

CFRelease(frameRef);

CFRelease(stringRef);

CFRelease(framesetter);
}

```



Section 100.2: Show PDF

```

NSString* fileName = @"firstIOS.PDF";

NSArray *arrayPaths =
NSSearchPathForDirectoriesInDomains(
    NSDocumentDirectory,
    NSUserDomainMask,
    YES);

NSString *path = [arrayPaths objectAtIndex:0];

NSString* pdfFileName = [path stringByAppendingPathComponent:fileName];

UIWebView* webView = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)];

NSURL *url = [NSURL fileURLWithPath:pdfFileName];

NSURLRequest *request = [NSURLRequest requestWithURL:url];

```

```
[webView setScalesPageToFit:YES];  
[webView loadRequest:request];  
[self.view addSubview:webView];
```

第100.3节：多页PDF

```
UIGraphicsBeginPDFContextToFile(fileName, CGRectMakeZero, nil);  
  
UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);  
  
UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);  
  
UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);  
  
UIGraphicsEndPDFContext();
```

第100.4节：从UIWebview中加载的任何Microsoft文档创建PDF

```
#define kPaperSizeA4 CGSizeMake(595.2, 841.8)
```

首先实现UIPrintPageRenderer协议

```
@interface UIPrintPageRenderer (PDF)  
- (NSData*) printToPDF;  
@end  
  
@implementation UIPrintPageRenderer (PDF)  
- (NSData*) printToPDF  
{  
    NSMutableData *pdfData = [NSMutableData data];  
    UIGraphicsBeginPDFContextToData( pdfData, self.paperRect, nil );  
    [self prepareForDrawingPages: NSMakeRange(0, self.numberOfPages)];  
    CGRect bounds = UIGraphicsGetPDFContextBounds();  
    for ( int i = 0 ; i < self.numberOfPages ; i++ )  
    {  
        UIGraphicsBeginPDFPage();  
        [self drawPageAtIndex: i inRect: bounds];  
    }  
    UIGraphicsEndPDFContext();  
    return pdfData;  
}  
@end
```

然后，在UIWebView文档加载完成后调用以下方法

```
-(void)createPDF:(UIWebView *)webView {  
  
UIPrintPageRenderer *render = [[UIPrintPageRenderer alloc] init];  
[render addPrintFormatter:webView.viewPrintFormatter startingAtPageAtIndex:0];  
  
float padding = 10.0f;  
CGRect paperRect = CGRectMake(0, 0, kPaperSizeA4.width, kPaperSizeA4.height);
```

```
[webView setScalesPageToFit:YES];  
[webView loadRequest:request];  
[self.view addSubview:webView];
```

Section 100.3: Multiple page PDF

```
UIGraphicsBeginPDFContextToFile(fileName, CGRectMakeZero, nil);  
  
UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);  
  
UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);  
  
UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);  
  
UIGraphicsEndPDFContext();
```

Section 100.4: Create PDF from any Microsoft Document loaded in UIWebView

```
#define kPaperSizeA4 CGSizeMake(595.2, 841.8)
```

First of all implement UIPrintPageRenderer protocol

```
@interface UIPrintPageRenderer (PDF)  
- (NSData*) printToPDF;  
@end  
  
@implementation UIPrintPageRenderer (PDF)  
- (NSData*) printToPDF  
{  
    NSMutableData *pdfData = [NSMutableData data];  
    UIGraphicsBeginPDFContextToData( pdfData, self.paperRect, nil );  
    [self prepareForDrawingPages: NSMakeRange(0, self.numberOfPages)];  
    CGRect bounds = UIGraphicsGetPDFContextBounds();  
    for ( int i = 0 ; i < self.numberOfPages ; i++ )  
    {  
        UIGraphicsBeginPDFPage();  
        [self drawPageAtIndex: i inRect: bounds];  
    }  
    UIGraphicsEndPDFContext();  
    return pdfData;  
}
```

Then, call below method after document finished loading in UIWebView

```
-(void)createPDF:(UIWebView *)webView {  
  
UIPrintPageRenderer *render = [[UIPrintPageRenderer alloc] init];  
[render addPrintFormatter:webView.viewPrintFormatter startingAtPageAtIndex:0];  
  
float padding = 10.0f;  
CGRect paperRect = CGRectMake(0, 0, kPaperSizeA4.width, kPaperSizeA4.height);
```

```
CGRect printableRect = CGRectMake(padding, padding, kPageSizeA4.width-(padding * 2),  
kPageSizeA4.height-(padding * 2));  
  
[render setValue:[NSValue valueWithCGRect:paperRect] forKey:@"paperRect"];  
[render setValue:[NSValue valueWithCGRect:printableRect] forKey:@"printableRect"];  
  
NSData *pdfData = [render printToPDF];  
  
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{  
  
    if (pdfData) {  
        [pdfData writeToFile:directoryPath atomically: YES];  
    }  
    else  
    {  
        NSLog(@"PDF couldnot be created");  
    }  
});}
```

```
CGRect printableRect = CGRectMake(padding, padding, kPageSizeA4.width-(padding * 2),  
kPageSizeA4.height-(padding * 2));  
  
[render setValue:[NSValue valueWithCGRect:paperRect] forKey:@"paperRect"];  
[render setValue:[NSValue valueWithCGRect:printableRect] forKey:@"printableRect"];  
  
NSData *pdfData = [render printToPDF];  
  
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{  
  
    if (pdfData) {  
        [pdfData writeToFile:directoryPath atomically: YES];  
    }  
    else  
    {  
        NSLog(@"PDF couldnot be created");  
    }  
});}
```

第101章：应用内购买

第101.1节：Swift 2中的单一应用内购买

在iTunesConnect中创建应用内购买后：

在你想要购买的视图控制器中

```
import StoreKit
```

并添加相关的代理

```
class ViewController: UIViewController, SKProductsRequestDelegate, SKPaymentTransactionObserver {
```

声明一个包含来自iTunesConnect的产品ID的变量

```
var product_id: NSString?
```

重写函数 viewDidLoad()

```
product_id = "YOUR_PRODUCT_ID"  
super.viewDidLoad()  
SKPaymentQueue.defaultQueue().addTransactionObserver(self)
```

//检查产品是否已购买
if (NSUserDefaults.standardUserDefaults().boolForKey("purchased")){

```
// 隐藏广告  
adView.hidden = true  
  
}  
}
```

将按钮连接到购买应用内购买功能的函数

```
@IBAction func 解锁操作(sender: AnyObject) {  
  
    打印("即将获取产品...")  
  
    // 可以进行支付  
    如果 (SKPaymentQueue.canMakePayments())  
    {  
        让 productID: NSSet = NSSet(对象: self.product_id!);  
        让 productsRequest: SKProductsRequest = SKProductsRequest(productIdentifiers: productID as!  
Set<NSString>);  
        productsRequest.代理 = self;  
        productsRequest.开始();  
        打印("正在获取产品");  
    } 否则{  
        打印("无法进行购买");  
    }  
}
```

Chapter 101: In-App Purchase

Section 101.1: Single IAP in Swift 2

After creating an IAP in iTunesConnect:

In the view controller that you want to buy in

```
import StoreKit
```

and add the relevant delegates

```
class ViewController: UIViewController, SKProductsRequestDelegate, SKPaymentTransactionObserver {
```

declare a variable with the product id from iTunesConnect

```
var product_id: NSString?
```

```
override func viewDidLoad() {
```

```
product_id = "YOUR_PRODUCT_ID"  
super.viewDidLoad()  
SKPaymentQueue.defaultQueue().addTransactionObserver(self)
```

//Check if product is purchased
if (NSUserDefaults.standardUserDefaults().boolForKey("purchased")){

```
// Hide ads  
adView.hidden = true  
  
}  
else {  
    打印("Should show ads...")  
}  
}
```

wire a button to a function to purchase the IAP

```
@IBAction func unlockAction(sender: AnyObject) {  
  
    打印("About to fetch the product...")  
  
    // Can make payments  
    if (SKPaymentQueue.canMakePayments())  
    {  
        let productID: NSSet = NSSet(object: self.product_id!);  
        let productsRequest: SKProductsRequest = SKProductsRequest(productIdentifiers: productID as!  
Set<NSString>);  
        productsRequest.delegate = self;  
        productsRequest.start();  
        打印("Fetching Products");  
    } else{  
        打印("Can't make purchases");  
    }  
}
```

```
}
```

以下是一些辅助方法

```
func buyProduct(product: SKProduct){
    println("向苹果发送支付请求");
    let payment = SKPayment(product: product)
    SKPaymentQueue.defaultQueue().addPayment(payment);
}
```

必须声明的代理方法

```
func productsRequest (request: SKProductsRequest, didReceiveResponse response: SKProductsResponse)
```

```
let count : Int = response.products.count
if (count>0) {
    var validProduct: SKProduct = response.products[0] as SKProduct
    if (validProduct.productIdentifier == self.product_id) {
        print(validProduct.localizedTitle)
        print(validProduct.localizedDescription)
        print(validProduct.price)
    }
    buyProduct(validProduct);
} else {
    print("nothing")
}
}
```

```
func request(request: SKRequest!, didFailWithError error: NSError!) {
    print("获取产品信息出错");
}
```

```
func paymentQueue(_ queue: SKPaymentQueue,
updatedTransactions transactions: [SKPaymentTransaction])
{
    print("收到来自苹果的支付交易响应");

    for transaction:AnyObject in transactions {
        if let trans:SKPaymentTransaction = transaction as? SKPaymentTransaction{
            switch trans.transactionState {
            case .Purchased:
                print("产品已购买");
            SKPaymentQueue.defaultQueue().finishTransaction(transaction as!
SKPaymentTransaction)
                // 处理购买
                NSUserDefaults.standardUserDefaults().setBool(true , forKey: "purchased")
                adView.hidden = true
                break;
            case .Failed:
                print("购买失败");
            SKPaymentQueue.defaultQueue().finishTransaction(transaction as!
SKPaymentTransaction)
                break;
            }
        }
    }
}
```

```
}
```

And here are some helper methods

```
func buyProduct(product: SKProduct){
    println("Sending the Payment Request to Apple");
    let payment = SKPayment(product: product)
    SKPaymentQueue.defaultQueue().addPayment(payment);
}
```

the delegate methods that must be declared

```
func productsRequest (request: SKProductsRequest, didReceiveResponse response: SKProductsResponse)
```

```
let count : Int = response.products.count
if (count>0) {
    var validProduct: SKProduct = response.products[0] as SKProduct
    if (validProduct.productIdentifier == self.product_id) {
        print(validProduct.localizedTitle)
        print(validProduct.localizedDescription)
        print(validProduct.price)
        buyProduct(validProduct);
    } else {
        print(validProduct.productIdentifier)
    }
} else {
    print("nothing")
}
}
```

```
func request(request: SKRequest!, didFailWithError error: NSError!) {
    print("Error Fetching product information");
}
```

```
func paymentQueue(_ queue: SKPaymentQueue,
updatedTransactions transactions: [SKPaymentTransaction])
{
    print("Received Payment Transaction Response from Apple");

    for transaction:AnyObject in transactions {
        if let trans:SKPaymentTransaction = transaction as? SKPaymentTransaction{
            switch trans.transactionState {
            case .Purchased:
                print("Product Purchased");
                SKPaymentQueue.defaultQueue().finishTransaction(transaction as!
SKPaymentTransaction)
                    // Handle the purchase
                    NSUserDefaults.standardUserDefaults().setBool(true , forKey: "purchased")
                    adView.hidden = true
                    break;
            case .Failed:
                print("Purchased Failed");
                SKPaymentQueue.defaultQueue().finishTransaction(transaction as!
SKPaymentTransaction)
                    break;
            }
        }
    }
}
```

```

        case .Restored:
            print("已购买");
            SKPaymentQueue.defaultQueue().restoreCompletedTransactions()

            // 处理购买
            UserDefaults.standard.setBool(true, forKey: "purchased")
            adView.hidden = true
            break;
        default:
            break;
    }
}

}

```

然后是恢复非消耗型应用内购买的代码

```

if (SKPaymentQueue.canMakePayments()) {
    SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
}

```

第101.2节：购买/订阅应用内购买用户的最基本步骤

假设你知道productID：

首先

```
import StoreKit
```

然后在你的代码中

```

let productID: Set = ["premium"]
let request = SKProductsRequest(productIdentifiers: productID)
request.delegate = self
request.start()

```

以及在SKProductsRequestDelegate中：

```

func productsRequest(request: SKProductsRequest, didReceiveResponse response: SKProductsResponse) {
    if response.products.count > 0 {
        let product = response.products[0]
        let payment = SKPayment(product: product)
        SKPaymentQueue.defaultQueue().addPayment(payment)
    }
}

```

第101.3节：在iTunesConnect中设置

在iTunesConnect中，选择您想要添加内购项目（IAP）的应用程序。

点击功能，您将看到如下界面：

```

        case .Restored:
            print("Already Purchased");
            SKPaymentQueue.defaultQueue().restoreCompletedTransactions()

            // Handle the purchase
            UserDefaults.standard.setBool(true, forKey: "purchased")
            adView.hidden = true
            break;
        default:
            break;
    }
}

}

```

And then the code to restore a non-consumable in app purchase

```

if (SKPaymentQueue.canMakePayments()) {
    SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
}

```

Section 101.2: Most basic steps for purchasing/subscribing a user to an IAP

Assuming you know the productID:

First

```
import StoreKit
```

Then in your code

```

let productID: Set = ["premium"]
let request = SKProductsRequest(productIdentifiers: productID)
request.delegate = self
request.start()

```

and in the SKProductsRequestDelegate:

```

func productsRequest(request: SKProductsRequest, didReceiveResponse response: SKProductsResponse) {
    if response.products.count > 0 {
        let product = response.products[0]
        let payment = SKPayment(product: product)
        SKPaymentQueue.defaultQueue().addPayment(payment)
    }
}

```

Section 101.3: Set Up in iTunesConnect

In [iTunesConnect](#), select the app which you want to add an IAP to.

Click on features and you will see this:

In-App Purchases (0)

[View Shared Secret](#)

Click + to add an In-App Purchase.

点击加号。然后您需要选择想要制作的内购项目类型。

接着，您需要填写内购项目的所有信息。

In-App Purchase Summary

Enter a reference name and a product ID for this In-App Purchase.

Reference Name:

Product ID:

Pricing and Availability

Enter the pricing and availability details for this In-App Purchase below.

Cleared for Sale Yes No

Price Tier
[View Pricing Matrix](#)

如果遇到任何问题，您可以参考内购项目设置指南。[_____](#)

In-App Purchases (0)

[View Shared Secret](#)

Click + to add an In-App Purchase.

Click the plus. You will then need to select which type of IAP you want to make.

Then you will need to fill out all of the information for your IAP.

In-App Purchase Summary

Enter a reference name and a product ID for this In-App Purchase.

Reference Name:

Product ID:

Pricing and Availability

Enter the pricing and availability details for this In-App Purchase below.

Cleared for Sale Yes No

Price Tier
[View Pricing Matrix](#)

If you have any trouble you can consult the [IAP Set Up Guide](#).

第102章：CGContext参考

第102.1节：绘制线条

```
CGContextRef context = UIGraphicsGetCurrentContext();

CGContextSetLineWidth(context, 5.0);
CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();
CGContextMoveToPoint(context, 200, 400);
CGContextAddLineToPoint(context, 100, 100);
CGContextStrokePath(context);
CGColorSpaceRelease(colorspace);
```



第102.2节：绘制文本

绘制需要在构建阶段添加Core Text框架

```
[NSString* textToDraw = @"Welcome to the world Of IOS";

CFStringRef stringRef = (_bridge CFStringRef)textToDraw;

CFAAttributedStringRef currentText = CFAAttributedStringCreate(NULL, stringRef, NULL);
CTFramesetterRef framesetter = CTFramesetterCreateWithAttributedString(currentText);
CGRect frameRect = CGRectMake(0, 0, 300, 100);
CGMutablePathRef framePath = CGPathCreateMutable();
CGPathAddRect(framePath, NULL, frameRect);

CFRange currentRange = CFRangeMake(0, 0);
CTFrameRef frameRef = CTFramesetterCreateFrame(framesetter, currentRange, framePath, NULL);
CGPathRelease(framePath);
CGContextRef currentContext = UIGraphicsGetCurrentContext();

CGContextSetTextMatrix(currentContext, CGAffineTransformIdentity);
CGContextTranslateCTM(currentContext, 200, 300);
CGContextScaleCTM(currentContext, 2, -2);
CTFrameDraw(frameRef, currentContext);

CFRelease(frameRef);
CFRelease(stringRef);
CFRelease(framesetter);
```

Chapter 102: CGContext Reference

Section 102.1: Draw line

```
CGContextRef context = UIGraphicsGetCurrentContext();

CGContextSetLineWidth(context, 5.0);
CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();
CGContextMoveToPoint(context, 200, 400);
CGContextAddLineToPoint(context, 100, 100);
CGContextStrokePath(context);
CGColorSpaceRelease(colorspace);
```



Section 102.2: Draw Text

Draw To requires **Core Text framework** to be added in the Build Phase

```
[NSString* textToDraw = @"Welcome to the world Of IOS";

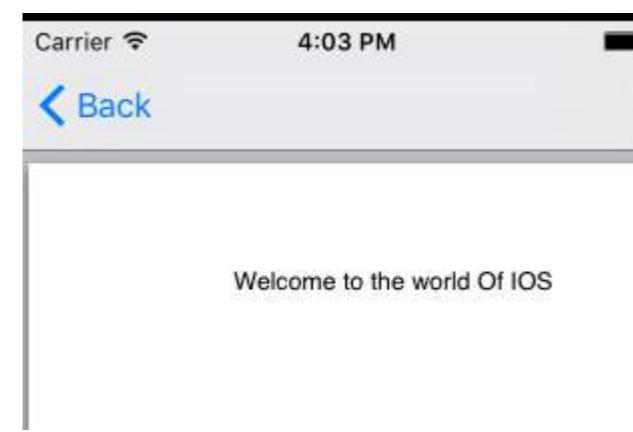
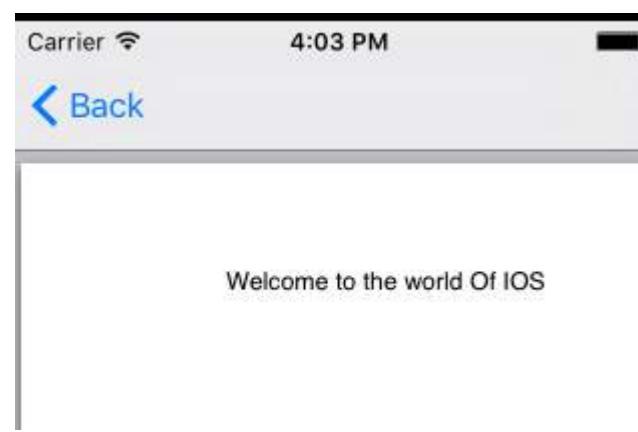
CFStringRef stringRef = (_bridge CFStringRef)textToDraw;

CFAAttributedStringRef currentText = CFAAttributedStringCreate(NULL, stringRef, NULL);
CTFramesetterRef framesetter = CTFramesetterCreateWithAttributedString(currentText);
CGRect frameRect = CGRectMake(0, 0, 300, 100);
CGMutablePathRef framePath = CGPathCreateMutable();
CGPathAddRect(framePath, NULL, frameRect);

CFRange currentRange = CFRangeMake(0, 0);
CTFrameRef frameRef = CTFramesetterCreateFrame(framesetter, currentRange, framePath, NULL);
CGPathRelease(framePath);
CGContextRef currentContext = UIGraphicsGetCurrentContext();

CGContextSetTextMatrix(currentContext, CGAffineTransformIdentity);
CGContextTranslateCTM(currentContext, 200, 300);
CGContextScaleCTM(currentContext, 2, -2);
CTFrameDraw(frameRef, currentContext);

CFRelease(frameRef);
CFRelease(stringRef);
CFRelease(framesetter);
```



第103章：核心定位

第103.1节：请求使用定位服务的权限

使用以下方法检查应用的授权状态：

```
//Swift  
let status: CLAuthorizationStatus = CLLocationManager.authorizationStatus()  
  
//Objective-C  
CLAuthorizationStatus status = [CLLocationManager authorizationStatus];
```

将状态与以下常量进行比较：

```
//Swift  
switch 状态 {  
case .未确定:  
    // 执行操作  
case .始终授权:  
    // 执行操作  
case .使用时授权:  
    // 执行操作  
case .受限:  
    // 执行操作  
case .拒绝:  
    // 执行操作  
}  
  
//Objective-C  
switch (状态) {  
    case kCLAuthorizationStatusNotDetermined:  
        // 用户尚未选择是否允许您的应用使用定位服务。  
        break;  
  
    case kCLAuthorizationStatusAuthorizedAlways:  
        // 用户已允许您的应用始终使用定位服务，即使应用在后台。  
        break;  
  
    case kCLAuthorizationStatusAuthorizedWhenInUse:  
        // 用户仅允许您的应用在前台使用定位服务。  
        break;  
  
    case kCLAuthorizationStatusRestricted:  
        // 用户无法选择是否允许您的应用使用定位服务，这可能是由于家长控制等原因。  
        break;  
  
    case kCLAuthorizationStatusDenied:  
        // 用户已选择不允许您的应用使用定位服务。  
        break;
```

Chapter 103: Core Location

Section 103.1: Request Permission to Use Location Services

Check the app's authorization status with:

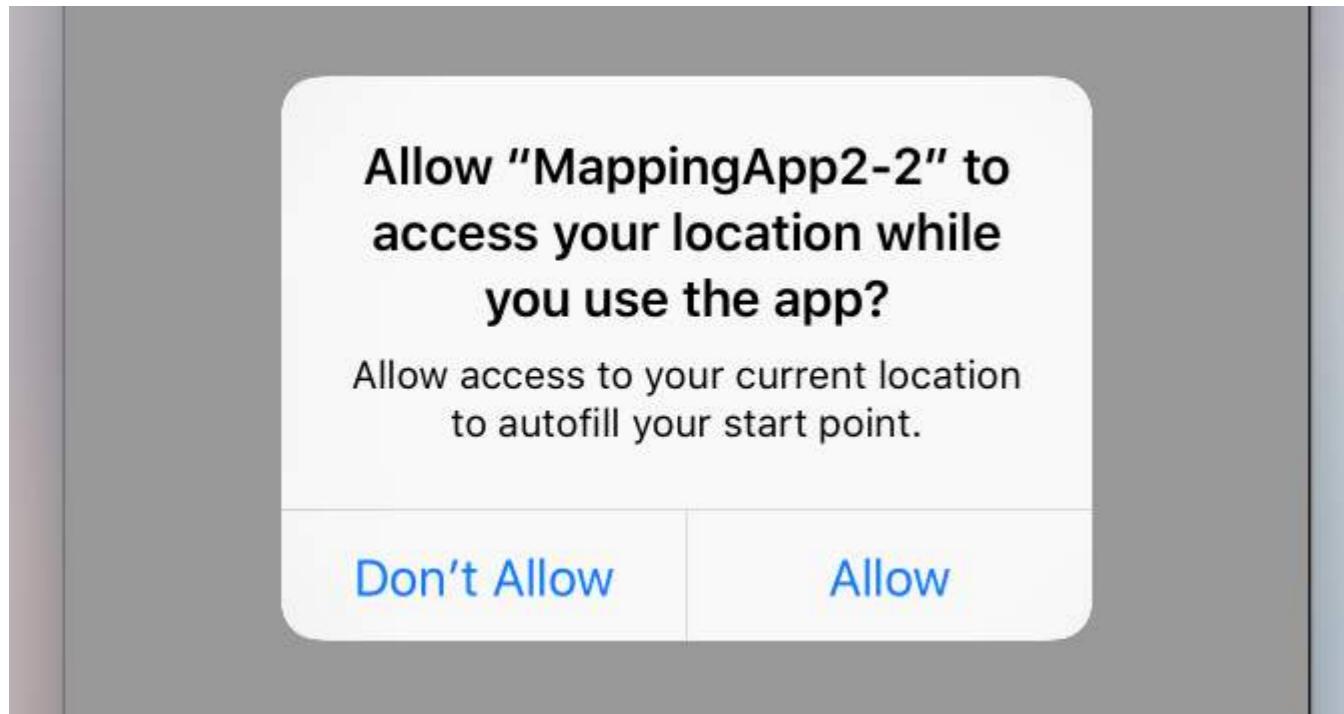
```
//Swift  
let status: CLAuthorizationStatus = CLLocationManager.authorizationStatus()  
  
//Objective-C  
CLAuthorizationStatus status = [CLLocationManager authorizationStatus];
```

Test the status against the follow constants:

```
//Swift  
switch status {  
case .NotDetermined:  
    // Do stuff  
case .AuthorizedAlways:  
    // Do stuff  
case .AuthorizedWhenInUse:  
    // Do stuff  
case .Restricted:  
    // Do stuff  
case .Denied:  
    // Do stuff  
}  
  
//Objective-C  
switch (status) {  
    case kCLAuthorizationStatusNotDetermined:  
        // The user hasn't yet chosen whether your app can use location services or not.  
        break;  
  
    case kCLAuthorizationStatusAuthorizedAlways:  
        // The user has let your app use location services all the time, even if the app is in the background.  
        break;  
  
    case kCLAuthorizationStatusAuthorizedWhenInUse:  
        // The user has let your app use location services only when the app is in the foreground.  
        break;  
  
    case kCLAuthorizationStatusRestricted:  
        // The user can't choose whether or not your app can use location services or not, this could be due to parental controls for example.  
        break;  
  
    case kCLAuthorizationStatusDenied:  
        // The user has chosen to not let your app use location services.
```

```
break;  
  
default:  
    break;  
}
```

在应用使用时获取定位服务权限



最简单的方法是将定位管理器初始化为根视图控制器的属性，并将权限请求放在其viewDidLoad中。

这会弹出请求权限的警告控制器：

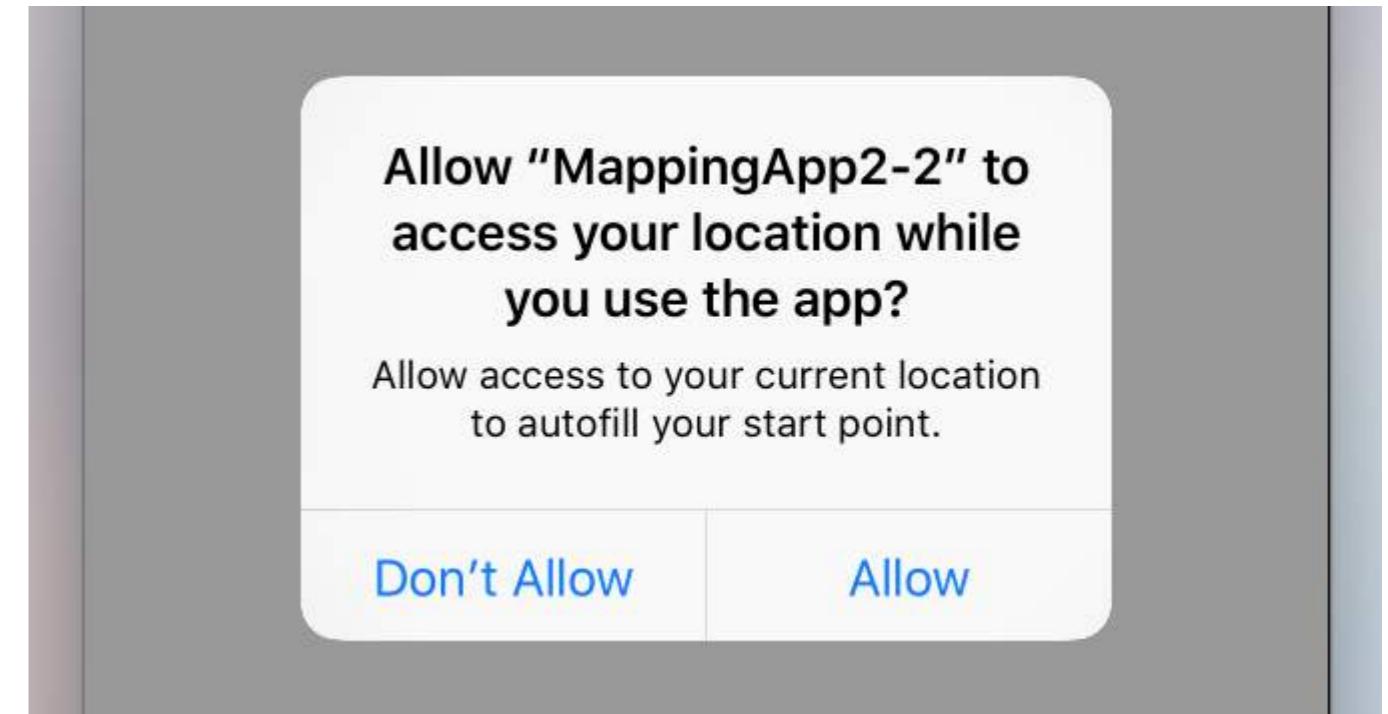
```
//Swift  
let locationManager = CLLocationManager()  
locationManager.requestWhenInUseAuthorization()
```

```
//Objective-C  
CLLocationManager *locationManager = [[CLLocationManager alloc] init];  
[locationManager requestWhenInUseAuthorization];
```

在你的Info.plist中添加NSLocationWhenInUseUsageDescription键。该值将用于警告控制器的message标签。

```
break;  
  
default:  
    break;  
}
```

Getting Location Service Permission While App is in Use



Simplest method is to initialize the location manager as a property of your root view controller and place the permission request in its viewDidLoad.

This brings up the alert controller that asks for permission:

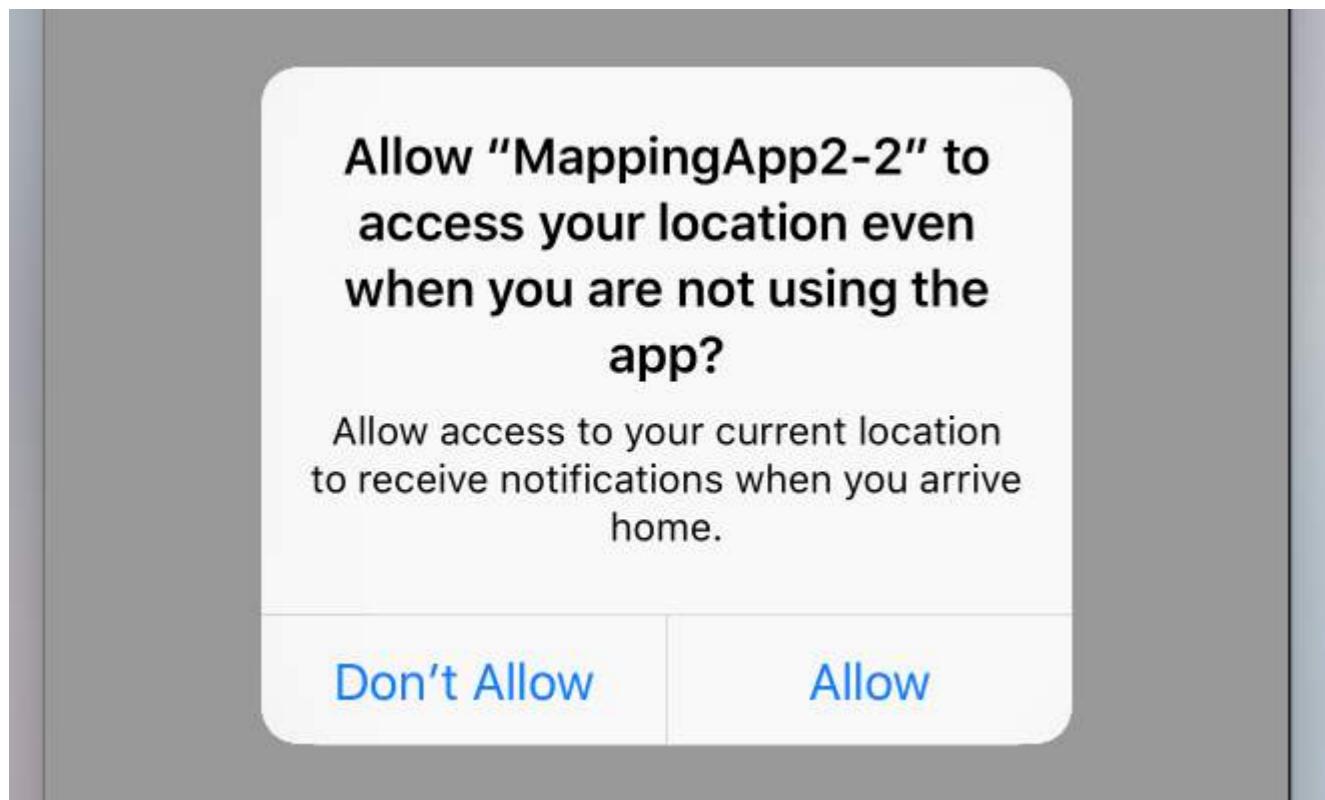
```
//Swift  
let locationManager = CLLocationManager()  
locationManager.requestWhenInUseAuthorization()
```

```
//Objective-C  
CLLocationManager *locationManager = [[CLLocationManager alloc] init];  
[locationManager requestWhenInUseAuthorization];
```

Add the **NSLocationWhenInUseUsageDescription** key to your *Info.plist*. The value will be used in the alert controller's message label.

MappingApp2-2 > MappingApp2-2 > Info.plist > No Selection			
M	Key	Type	Value
	Information Property List	Dictionary	(16 items)
A	NSLocationAlwaysUsageDescription	String	Allow access to your current location to receive notifications when you arrive home.
A	NSLocationWhenInUseUsageDescription	String	Allow access to your current location to autofill your start point.
A	Localization native development region	String	en
A	Executable file	String	\$(EXECUTABLE_NAME)
A	Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
A	InfoDictionary version	String	6.0
A	Bundle name	String	\$(PRODUCT_NAME)
A	Bundle OS Type code	String	APPL
A	Bundle versions string, short	String	1.0
A	Bundle creator OS Type code	String	????
A	Bundle version	String	1
A	Application requires iPhone environment	Boolean	YES
A	Launch screen interface file base name	String	LaunchScreen
A	Main storyboard file base name	String	Main
A	Required device capabilities	Array	(1 item)
A	Supported interface orientations	Array	(3 items)

获取始终允许使用定位服务的权限



如果想请求即使应用未激活时也能使用定位服务的权限，请使用以下调用：

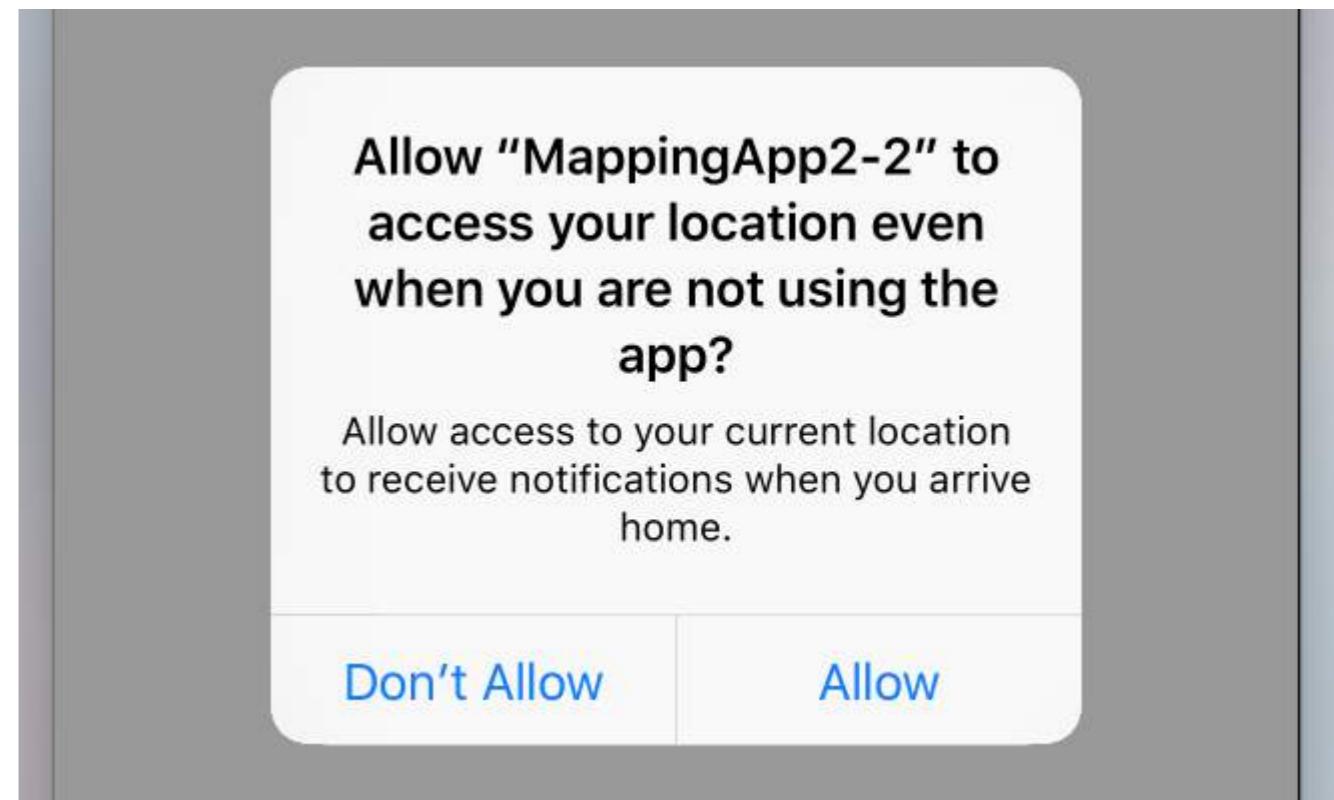
```
//Swift
locationManager.requestAlwaysAuthorization()

//Objective-C
[locationManager requestAlwaysAuthorization];
```

然后在你的Info.plist中添加NSLocationAlwaysUsageDescription键。同样，该值将用于警告控制器的message标签。

MappingApp2-2 > MappingApp2-2 > Info.plist > No Selection			
M	Key	Type	Value
	Information Property List	Dictionary	(16 items)
A	NSLocationAlwaysUsageDescription	String	Allow access to your current location to receive notifications when you arrive home.
A	NSLocationWhenInUseUsageDescription	String	Allow access to your current location to autofill your start point.
A	Localization native development region	String	en
A	Executable file	String	\$(EXECUTABLE_NAME)
A	Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
A	InfoDictionary version	String	6.0
A	Bundle name	String	\$(PRODUCT_NAME)
A	Bundle OS Type code	String	APPL
A	Bundle versions string, short	String	1.0
A	Bundle creator OS Type code	String	????
A	Bundle version	String	1
A	Application requires iPhone environment	Boolean	YES
A	Launch screen interface file base name	String	LaunchScreen
A	Main storyboard file base name	String	Main
A	Required device capabilities	Array	(1 item)
A	Supported interface orientations	Array	(3 items)

Getting Location Service Permission Always



To ask for permission to use location services even when the app is not active, use the following call instead:

```
//Swift
locationManager.requestAlwaysAuthorization()

//Objective-C
[locationManager requestAlwaysAuthorization];
```

Then add the **NSLocationAlwaysUsageDescription** key to your *Info.plist*. Again, the value will be used in the alert controller's message label.

MappingApp2-2 > MappingApp2-2 > Info.plist > No Selection		
	Key	Type
	NSLocationAlwaysUsageDescription	String
	NSLocationWhenInUseUsageDescription	String
	Localization native development region	String
	Executable file	String
	Bundle identifier	String
	InfoDictionary version	String
	Bundle name	String
	Bundle OS Type code	String
	Bundle versions string, short	String
	Bundle creator OS Type code	String
	Bundle version	String
	Application requires iPhone environment	Boolean
	Launch screen interface file base name	String
	Main storyboard file base name	String
	Required device capabilities	Array
	Supported Interface orientations	Array

第103.2节：使用GPX文件添加自定义位置

要检查定位服务，我们需要真实设备，但出于测试目的，我们也可以使用模拟器并通过以下步骤添加我们自己的位置：

- 将新的GPX文件添加到您的项目中。
- 在GPX文件中添加航点，如下所示

```
<?xml version="1.0"?>
<gpx version="1.1" creator="Xcode">
<!--
提供一个或多个包含纬度/经度对的航点。如果只提供一个航点，Xcode将模拟该特定位置。如果提供多个航点，Xcode将模
拟访问每个航点的路线。
-->
<wpt lat="52.599878" lon="4.702029">
    <name>位置名称（例如佛罗里达）</name>
</wpt>
```

- 然后进入产品-->方案-->编辑方案，在运行（RUN）中将默认位置设置为您的GPX文件名。

第103.3节：链接CoreLocation框架

MappingApp2-2 > MappingApp2-2 > Info.plist > No Selection		
	Key	Type
	NSLocationAlwaysUsageDescription	String
	NSLocationWhenInUseUsageDescription	String
	Localization native development region	String
	Executable file	String
	Bundle identifier	String
	InfoDictionary version	String
	Bundle name	String
	Bundle OS Type code	String
	Bundle versions string, short	String
	Bundle creator OS Type code	String
	Bundle version	String
	Application requires iPhone environment	Boolean
	Launch screen interface file base name	String
	Main storyboard file base name	String
	Required device capabilities	Array
	Supported Interface orientations	Array

Section 103.2: Add own custom location using GPX file

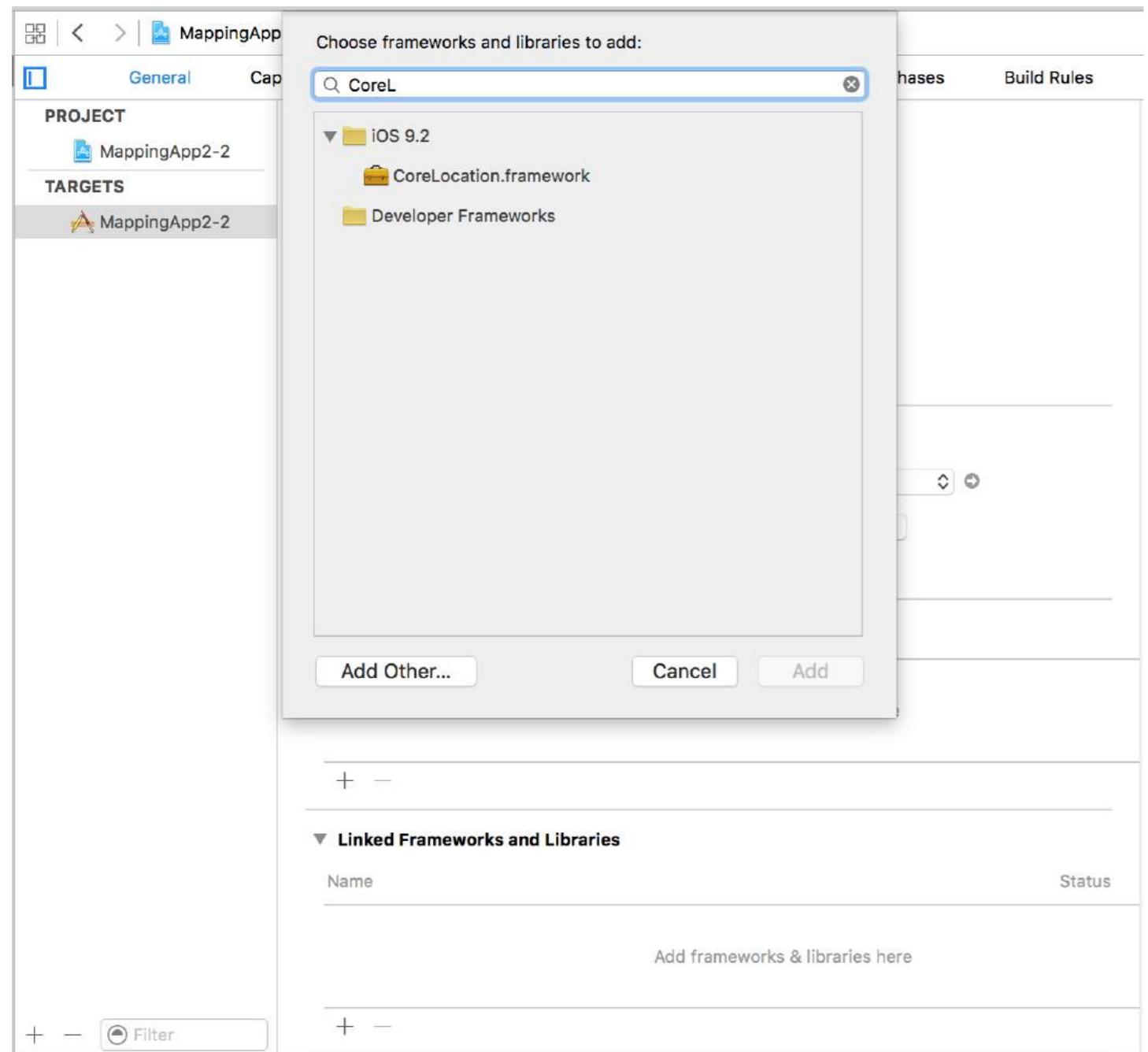
To check for location services we need real device but for testing purpose we can also use simulator and add our own location by following below steps:

- add new GPX file into your project.
- in GPX file add waypoints like

```
<?xml version="1.0"?>
<gpx version="1.1" creator="Xcode">
<!--
Provide one or more waypoints containing a latitude/longitude pair. If you provide one
waypoint, Xcode will simulate that specific location. If you provide multiple waypoints,
Xcode will simulate a route visiting each waypoint.
-->
<wpt lat="52.599878" lon="4.702029">
    <name>location name (eg. Florida)</name>
</wpt>
```

- then go to product-->Scheme-->Edit Scheme and into RUN set default location as your GPX file name.

Section 103.3: Link CoreLocation Framework



在使用 CoreLocation 功能的类中导入 CoreLocation 模块。

```
//Swift
import CoreLocation

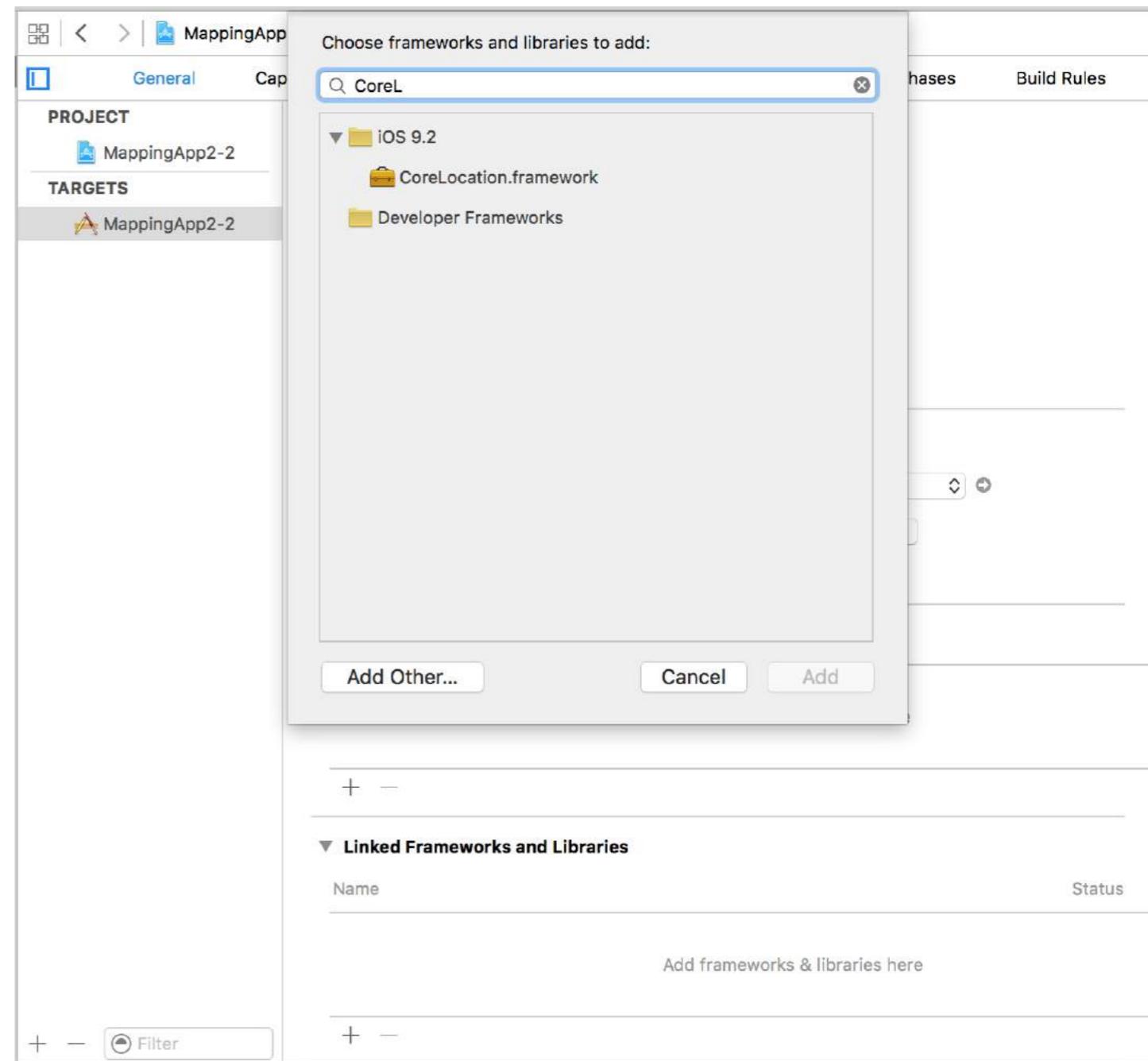
//Objective-C
#import <CoreLocation/CoreLocation.h>
```

第 103.4 节：后台位置服务

要在应用程序处于后台时使用标准位置服务，首先需要在目标设置的功能 (Capabilities) 选项卡中开启后台模式 (Background Modes) 并选择位置更新 (Location updates)。

或者，直接将其添加到 Info.plist 中。

```
<key>NSLocationAlwaysUsageDescription</key>
<string>我想在后台获取您的位置信息</string>
```



Import the CoreLocation module in your classes that use CoreLocation functionality.

```
//Swift
import CoreLocation

//Objective-C
#import <CoreLocation/CoreLocation.h>
```

Section 103.4: Location Services in the Background

To use standard location services while the application is in the background you need first turn on Background Modes in the Capabilities tab of the Target settings, and select Location updates.

Or, add it directly to the Info.plist.

```
<key>NSLocationAlwaysUsageDescription</key>
<string>I want to get your location Information in background</string>
```

```
<key>UIBackgroundModes</key>
<array>
    位置
```

然后你需要设置 CLLocationManager

Objective C

```
//位置管理器必须有强引用。
_locationManager = [[CLLocationManager alloc] init];
_locationManager.delegate = self;

//请求始终授权 (iOS8及以上)
if ([_locationManager respondsToSelector:@selector(requestAlwaysAuthorization)]) {
    [_locationManager requestAlwaysAuthorization];
}

//允许后台定位更新 (iOS9及以上)
if ([_locationManager respondsToSelector:@selector(allowsBackgroundLocationUpdates)]) {
    _locationManager.allowsBackgroundLocationUpdates = YES;
}

[_locationManager startUpdatingLocation];
```

Swift

```
self.locationManager.delegate = self

if #available (iOS 8.0,*) {
    self.locationManager.requestAlwaysAuthorization()
}

if #available (iOS 9.0,*) {
    self.locationManager.allowsBackgroundLocationUpdates = true
}

self.locationManager.startUpdatingLocation()
```

```
<key>UIBackgroundModes</key>
<array>
    <string>location</string>
</array>
```

Then you need to setup the CLLocationManager

Objective C

```
//The Location Manager must have a strong reference to it.
_locationManager = [[CLLocationManager alloc] init];
_locationManager.delegate = self;

//Request Always authorization (iOS8+)
if ([_locationManager respondsToSelector:@selector(requestAlwaysAuthorization)]) {
    [_locationManager requestAlwaysAuthorization];
}

//Allow location updates in the background (iOS9+)
if ([_locationManager respondsToSelector:@selector(allowsBackgroundLocationUpdates)]) {
    _locationManager.allowsBackgroundLocationUpdates = YES;
}

[_locationManager startUpdatingLocation];
```

Swift

```
self.locationManager.delegate = self

if #available (iOS 8.0,*) {
    self.locationManager.requestAlwaysAuthorization()
}

if #available (iOS 9.0,*) {
    self.locationManager.allowsBackgroundLocationUpdates = true
}

self.locationManager.startUpdatingLocation()
```

第104章：FacebookSDK

第104.1节：创建你自己的自定义“使用Facebook登录”按钮

有时我们想设计自己的“使用Facebook登录”按钮的用户界面，而不是使用FacebookSDK自带的原始按钮。

1. 在你的故事板中，拖拽UIButton并按你想要的方式设置它。
2. 按住Ctrl键，将按钮拖拽到视图控制器，作为IBAction连接。
3. 在**IBAction方法内部**，你需要模拟点击实际的Facebook按钮，方法如下：

Swift:

```
let loginButton = FBSDKLoginButton()  
loginButton.delegate = self  
// 你的自定义权限数组  
loginButton.readPermissions =  
[  
    "public_profile",  
    "email",  
    "user_about_me",  
    "user_photos"  
]  
// 隐藏按钮  
loginButton.hidden = true  
self.view.addSubview(loginButton)  
// 模拟点击实际的 Facebook SDK 按钮  
loginButton.sendActionsForControlEvents(UIControlEvents.TouchUpInside)
```

Objective-C:

```
FBSDKLoginButton *FBBButton = [FBSDKLoginButton new];  
  
// 你的自定义权限数组  
FBBButton.readPermissions = @[@"public_profile",  
                                @"email",  
                                @"user_about_me",  
                                @"user_photos"];  
FBBButton.loginBehavior = FBSDKLoginBehaviorNative;  
[FBBButton setDelegate:self];  
[FBBButton setHidden:true];  
[loginButton addSubview:FBBButton];  
  
[FBBButton sendActionsForControlEvents:UIControlEventTouchUpInside];
```

你完成了。

第104.2节：FacebookSDK集成

步骤1：安装SDK

你可以手动安装SDK，或者通过CocoaPods安装。强烈推荐后者。

将以下内容写入Podfile：

Chapter 104: FacebookSDK

Section 104.1: Creating your own custom "Sign In With Facebook" button

Sometimes we want to design our own UI for "Sign In With Facebook" button instead of the original button that comes with FacebookSDK.

1. In your storyboard, drag your UIButton and set it however you want it to be.
2. Ctrl + drag your button to your view controller as IBAction.
3. **Inside** the IBAction method you will have simulate a tap on the actual Facebook button as follow:

Swift:

```
let loginButton = FBSDKLoginButton()  
loginButton.delegate = self  
// Your Custom Permissions Array  
loginButton.readPermissions =  
[  
    "public_profile",  
    "email",  
    "user_about_me",  
    "user_photos"  
]  
// Hiding the button  
loginButton.hidden = true  
self.view.addSubview(loginButton)  
// Simulating a tap for the actual Facebook SDK button  
loginButton.sendActionsForControlEvents(UIControlEvents.TouchUpInside)
```

Objective-C:

```
FBSDKLoginButton *FBBButton = [FBSDKLoginButton new];  
  
// Your Custom Permissions Array  
FBBButton.readPermissions = @[@"public_profile",  
                                @"email",  
                                @"user_about_me",  
                                @"user_photos"];  
FBBButton.loginBehavior = FBSDKLoginBehaviorNative;  
[FBBButton setDelegate:self];  
[FBBButton setHidden:true];  
[loginButton addSubview:FBBButton];  
  
[FBBButton sendActionsForControlEvents:UIControlEventTouchUpInside];
```

You're done.

Section 104.2: FacebookSDK Integration

Step 1: Install the SDK

You can install the SDK [manually](#) or via CocoaPods. The latter option is highly recommended.

Put these lines in Podfile:

```
target 'MyApp' do
  use_frameworks!

  pod 'FBSDKCoreKit'
    pod 'FBSDKLoginKit'
    pod 'FBSDKShareKit'
end
```

在终端运行 `pod install`，然后打开 `.xcworkspace` 而不是 `.xcodeproj`。

`FBSDKLoginKit` 和 `FBSDKShareKit` 是可选的。你可以选择是否需要它们。

步骤2：在Facebook上创建应用

前往 [Quick Starts - Facebook for Developers](#) 创建一个应用。

创建应用后，Facebook会要求你下载SDK。如果你已经通过

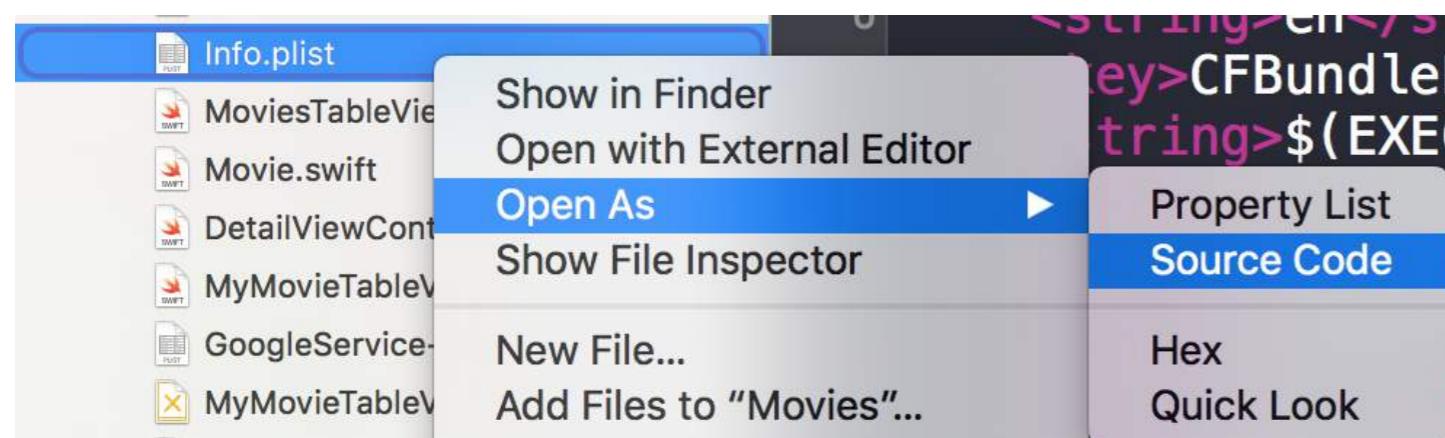
CocoaPods安装了SDK，可以跳过这一步。

步骤3：编辑 `.plist`

a. 为了让你的应用能够与Facebook“通信”，你需要在 `.plist` 文件中添加一些设置。

Facebook会在Quick Starts页面提供定制的代码片段。

b. 编辑您的`.plist`文件作为源代码。



c. 将您自定义的代码片段粘贴到源代码中。注意！该代码片段必须是 `<dict>` 标签的直接子元素。您的源代码应类似于：

```
<plist version="1.0">
<dict>
  // ...
//一些默认设置
  // ...
<key>CFBundleURLTypes</key>
<array>
  <dict>
    <key>CFBundleURLSchemes</key>
    <array>
      <string>fb{FBAppId}</string>
    </array>
  </dict>
</array>

<key>FacebookAppID</key>
<string>{FBAppId}</string>
```

```
target 'MyApp' do
  use_frameworks!

  pod 'FBSDKCoreKit'
    pod 'FBSDKLoginKit'
    pod 'FBSDKShareKit'
end
```

Run `pod install` in the terminal and open `.xcworkspace` instead of `.xcodeproj` afterwards.

`FBSDKLoginKit` and `FBSDKShareKit` are optional. You may or may not need them.

Step 2: Create an app on Facebook

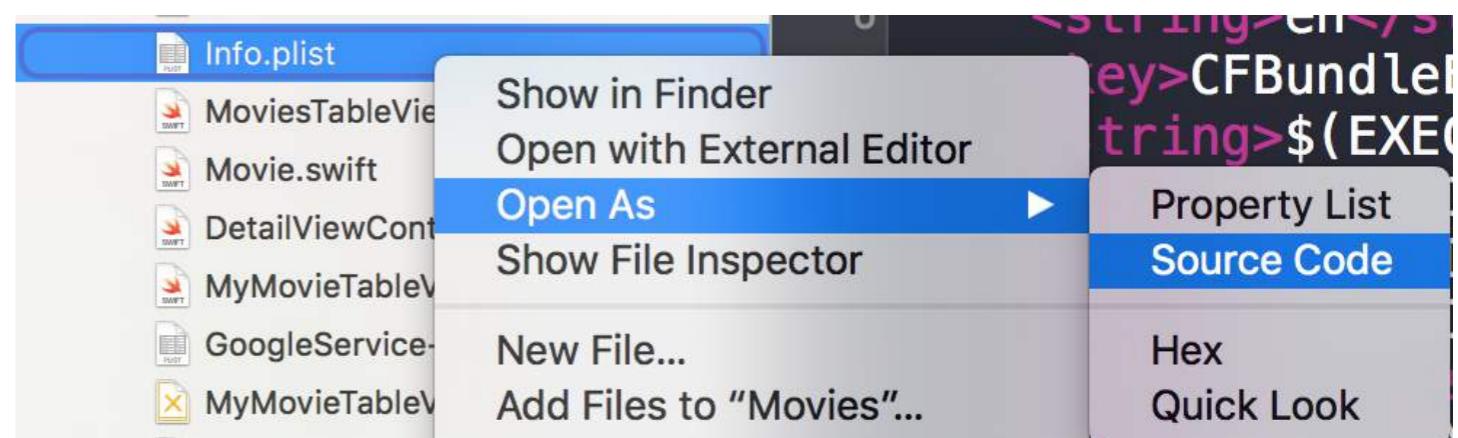
Go to [Quick Starts - Facebook for Developers](#) to create an app.

Facebook will ask you to download the SDK after creating the app. You can skip this part if you installed the SDK via CocoaPods already.

Step 3: Edit `.plist`

a. To make your app able to "communicate" with Facebook, you need to put some settings in your `.plist` file. Facebook will give you the customized snippet on the Quick Starts page.

b. Edit your `.plist` file as source code.



c. Paste your customized snippet in the source code. **Be careful!** The snippet must be exactly the child of the `<dict>` tag. Your source code should be something like:

```
<plist version="1.0">
<dict>
  // ...
//some default settings
  // ...
<key>CFBundleURLTypes</key>
<array>
  <dict>
    <key>CFBundleURLSchemes</key>
    <array>
      <string>fb{FBAppId}</string>
    </array>
  </dict>
</array>

<key>FacebookAppID</key>
<string>{FBAppId}</string>
```

```

<key>FacebookDisplayName</key>
<string>{FBAppName}</string>
<key>LSApplicationQueriesSchemes</key>
<array>
    fbapi
    fb-messenger-api
    fbauth2
    fbshareextension

<key>NSAppTransportSecurity</key>
<dict>
    <key>NSEExceptionDomains</key>
    <dict>
        <key>facebook.com</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSEExceptionRequiresForwardSecrecy</key>
            <false/>

        <key>fbcdn.net</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSEExceptionRequiresForwardSecrecy</key>
            <false/>

        <key>akamaihd.net</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSEExceptionRequiresForwardSecrecy</key>
            <false/>
        </dict>
    </dict>
</plist>

```

如果你将代码片段粘贴到错误的位置，会遇到问题。

步骤4：在快速入门页面告诉Facebook你的应用包标识符。

=> [如何获取包标识符](#)

步骤5：编辑你的AppDelegate.swift

a.

```
import FBSDKCoreKit
```

b.

```

func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
    FBSDKApplicationDelegate.sharedInstance().application(application,
    didFinishLaunchingWithOptions: launchOptions)
    返回 true
}

func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?,

```

```

<key>FacebookDisplayName</key>
<string>{FBAppName}</string>
<key>LSApplicationQueriesSchemes</key>
<array>
    fbapi
    fb-messenger-api
    fbauth2
    fbshareextension
</array>
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSEExceptionDomains</key>
    <dict>
        <key>facebook.com</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSEExceptionRequiresForwardSecrecy</key>
            <false/>
        </dict>
        <key>fbcdn.net</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSEExceptionRequiresForwardSecrecy</key>
            <false/>
        </dict>
        <key>akamaihd.net</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSEExceptionRequiresForwardSecrecy</key>
            <false/>
        </dict>
    </dict>
</dict>
</plist>

```

If you paste the snippet at a wrong place, you will run into problems.

Step 4: Tell Facebook your bundle identifier on the Quick Starts page.

=> [How to get bundle identifier](#)

Step 5: Edit your AppDelegate.swift

a.

```
import FBSDKCoreKit
```

b.

```

func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
    FBSDKApplicationDelegate.sharedInstance().application(application,
    didFinishLaunchingWithOptions: launchOptions)
    return true
}

func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?,

```

```

annotation: AnyObject) -> Bool {
    return FBSDKApplicationDelegate.sharedInstance().application(application, openURL: url,
sourceApplication: sourceApplication, annotation: annotation)
}

```

第104.3节：获取Facebook用户数据

用户在您的应用中登录Facebook后，现在是时候获取您请求的数据了
FBButton.readPermissions.

Swift:

```

enum FacebookParametesField : String
{
    case FIELDS_KEY = "fields"
    case FIELDS_VALUE = "id, email, picture, first_name, last_name"
}

if FBSDKAccessToken.currentAccessToken() != nil
{
    // 获取用户的Facebook数据
    FBSDKGraphRequest(graphPath: "me",
parameters: [FacebookParametesField.FIELDS_KEY.rawValue :
FacebookParametesField.FIELDS_VALUE.rawValue])
.startWithCompletionHandler({ (graphConnection : FBSDKGraphRequestConnection!, result : AnyObject!, error : NSError!) -> Void in

        if error == nil
        {
            print("Facebook Graph phaze")

            let email = result["email"]
            let facebookToken = FBSDKAccessToken.currentAccessToken().tokenString
            let userFacebookId = result["id"]
            let firstName = result["first_name"]
            let lastName = result["last_name"]

            if let result = result as? Dictionary<String, AnyObject>
            {
                if let picture = result["picture"] as? Dictionary<String,AnyObject>
                {
                    if let data = picture["data"] as? Dictionary <String,AnyObject>
                    {
                        if let url = data["url"] as? String
                        {
                            // 头像图片 URL
                            let profilePictureURL = url
                        }
                    }
                }
            }
        }
    })
}

```

```

annotation: AnyObject) -> Bool {
    return FBSDKApplicationDelegate.sharedInstance().application(application, openURL: url,
sourceApplication: sourceApplication, annotation: annotation)
}

```

Section 104.3: Fetching the facebook user data

After the user signed in to Facebook at your app, now it's time to fetch the data you requested at the
FBButton.readPermissions.

Swift:

```

enum FacebookParametesField : String
{
    case FIELDS_KEY = "fields"
    case FIELDS_VALUE = "id, email, picture, first_name, last_name"
}

if FBSDKAccessToken.currentAccessToken() != nil
{
    // Getting user facebook data
    FBSDKGraphRequest(graphPath: "me",
parameters: [FacebookParametesField.FIELDS_KEY.rawValue :
FacebookParametesField.FIELDS_VALUE.rawValue])
.startWithCompletionHandler({ (graphConnection : FBSDKGraphRequestConnection!, result : AnyObject!, error : NSError!) -> Void in

        if error == nil
        {
            print("Facebook Graph phaze")

            let email = result["email"]
            let facebookToken = FBSDKAccessToken.currentAccessToken().tokenString
            let userFacebookId = result["id"]
            let firstName = result["first_name"]
            let lastName = result["last_name"]

            if let result = result as? Dictionary<String, AnyObject>
            {
                if let picture = result["picture"] as? Dictionary<String,AnyObject>
                {
                    if let data = picture["data"] as? Dictionary <String,AnyObject>
                    {
                        if let url = data["url"] as? String
                        {
                            // Profile picture URL
                            let profilePictureURL = url
                        }
                    }
                }
            }
        }
    })
}

```

第105章：AFNetworking

第105.1节：在自定义线程上调度完成块

每当使用AFNetworking时，调用会被调度到AFNetworking提供的自定义线程。当调用返回到完成块时，它会在主线程上执行。

此示例设置了一个调度到完成块的自定义线程：

AFNetworking 2.xx :

```
// 使用您的名称和DISPATCH_QUEUE_SERIAL作为标志创建dispatch_queue_t
dispatch_queue_t myQueue = dispatch_queue_create("com.CompanyName.AppName.methodTest",
DISPATCH_QUEUE_SERIAL);

// 初始化AFNetworking的AFHTTPRequestOperation
operation = [[AFHTTPRequestOperation alloc] initWithRequest:request];

// 设置FMDB属性以在主线程之外运行
[operation setCompletionQueue:myQueue];
```

AFNetworking 3.xx :

```
AFHTTPSessionManager *manager = [[AFHTTPSessionManager alloc] init];
[self setCompletionQueue:myQueue];
```

Chapter 105: AFNetworking

Section 105.1: Dispatching completion block on a custom thread

Whenever AFNetworking is used the call is dispatched on a custom thread provided by AFNetworking. When the call returns to the completion block, it gets executed on the main thread.

This example sets a custom thread that dispatch to the completion block:

AFNetworking 2.xx:

```
// Create dispatch_queue_t with your name and DISPATCH_QUEUE_SERIAL as for the flag
dispatch_queue_t myQueue = dispatch_queue_create("com.CompanyName.AppName.methodTest",
DISPATCH_QUEUE_SERIAL);

// init AFHTTPRequestOperation of AFNetworking
operation = [[AFHTTPRequestOperation alloc] initWithRequest:request];

// Set the FMDB property to run off the main thread
[operation setCompletionQueue:myQueue];
```

AFNetworking 3.xx:

```
AFHTTPSessionManager *manager = [[AFHTTPSessionManager alloc] init];
[self setCompletionQueue:myQueue];
```

第106章：CTCallCenter

第106.1节：CallKit - iOS 10

```
//头文件  
  
<CallKit/CXCallObserver.h>  
  
CXCallObserver *callObserver = [[CXCallObserver alloc] init];  
  
// 如果队列为nil，则回调将在主队列上执行  
  
[callObserver setDelegate:self queue:nil];  
  
// 不要忘记保存对callObserver的引用，以防止其被释放  
  
self.callObserver = callObserver;  
  
// 获取通话状态  
- (void)callObserver:(CXCallObserver *)callObserver callChanged:(CXCall *)call {  
    if (call.hasConnected) {  
        // 执行必要的操作  
    }  
}
```

第106.2节：即使在后台也能拦截来自您应用的通话

来自苹果文档：

使用 CTCallCenter 类获取当前蜂窝电话的列表，并响应电话状态的变化，例如从拨号状态到连接状态。这些状态变化称为蜂窝电话事件。

CTCallCenter 的目的是让开发者有机会在通话期间暂停应用状态，以便为用户提供最佳体验。

Objective-C:

首先，我们将在想要处理拦截的类中定义一个新的类成员：

```
@property (atomic, strong) CTCallCenter *callCenter;
```

在类的初始化方法（构造函数）中，我们将为类成员分配新的内存：

```
[self setCallCenter:[CTCallCenter new]];
```

随后，我们将调用实际处理拦截的新方法：

```
- (void)registerPhoneCallListener  
{  
    [[self callCenter] setCallEventHandler:^(CTCall * _Nonnull call) {  
        NSLog(@"CallEventHandler called - interception in progress");  
    }];  
}
```

Chapter 106: CTCallCenter

Section 106.1: CallKit - ios 10

```
//Header File  
  
<CallKit/CXCallObserver.h>  
  
CXCallObserver *callObserver = [[CXCallObserver alloc] init];  
  
// If queue is nil, then callbacks will be performed on main queue  
  
[callObserver setDelegate:self queue:nil];  
  
// Don't forget to store reference to callObserver, to prevent it from being released  
  
self.callObserver = callObserver;  
  
// get call status  
- (void)callObserver:(CXCallObserver *)callObserver callChanged:(CXCall *)call {  
    if (call.hasConnected) {  
        // perform necessary actions  
    }  
}
```

Section 106.2: Intercepting calls from your app even from the background

From Apple documentation:

Use the CTCallCenter class to obtain a list of current cellular calls, and to respond to state changes for calls such as from a dialing state to a connected state. Such state changes are known as cellular call events.

The purpose of CTCallCenter is to give the developer the opportunity to pause his app state during a call in order to give the user the best experience.

Objective-C:

First, we will define a new class member inside the class we want to handle the interceptions:

```
@property (atomic, strong) CTCallCenter *callCenter;
```

Inside our class init (constructor) we will allocate new memory for our class member:

```
[self setCallCenter:[CTCallCenter new]];
```

Afterwards, we will invoke our new method that actually handles the interceptions:

```
- (void)registerPhoneCallListener  
{  
    [[self callCenter] setCallEventHandler:^(CTCall * _Nonnull call) {  
        NSLog(@"CallEventHandler called - interception in progress");  
    }];  
}
```

```

if ([call.callState isEqualToString: CTCallStateConnected])
{
    NSLog(@"已连接");
}
else if ([call.callState isEqualToString: CTCallStateDialing])
{
    NSLog(@"拨号中");
}
else if ([call.callState isEqualToString: CTCallStateDisconnected])
{
    NSLog(@"已断开");
}
} else if ([call.callState isEqualToString: CTCallStateIncoming])
{
    NSLog(@"来电中");
}
];
}

```

就是这样，如果用户使用你的应用并接到电话，你可以拦截此电话并处理你的应用以保存状态。

值得一提的是，有4种通话状态你可以拦截：

CTCallStateDialing
 CTCallStateIncoming
 CTCallStateConnected
 CTCallStateDisconnected

Swift:

在相关类中定义你的类成员并进行定义：

```

self.callCenter = CTCallCenter()
self.callCenter.callEventHandler = { call in
    // 处理你的拦截
    if call.callState == CTCallStateConnected
    {
    }
}

```

如果你的应用在后台，并且你需要在应用处于后台时拦截通话，会发生什么？

例如，如果你开发的是一个企业应用，你基本上只需在“功能”标签中添加两个功能（VOIP 和后台获取）：

你的项目目标 -> 功能 -> 后台模式 -> 勾选“语音 over IP”和“后台获取”

```

if ([call.callState isEqualToString: CTCallStateConnected])
{
    NSLog(@"Connected");
}
else if ([call.callState isEqualToString: CTCallStateDialing])
{
    NSLog(@"Dialing");
}
else if ([call.callState isEqualToString: CTCallStateDisconnected])
{
    NSLog(@"Disconnected");
}
else if ([call.callState isEqualToString: CTCallStateIncoming])
{
    NSLog(@"Incomming");
}
];
}

```

That's it, if the user will use your app and will receive a phone call you could intercept this call and handle your app for a save state.

It is worth mentioning that there are 4 call states you can intercept:

CTCallStateDialing
 CTCallStateIncoming
 CTCallStateConnected
 CTCallStateDisconnected

Swift:

Define your class member at the relevant class and define it:

```

self.callCenter = CTCallCenter()
self.callCenter.callEventHandler = { call in
    // Handle your interception
    if call.callState == CTCallStateConnected
    {
    }
}

```

What will happen if your app is in the background and you need to intercept calls while the app is in the background?

For example, if you develop an **enterprise** app you can basically just add 2 capabilities (VOIP & Background fetch) in the Capabilities tab:

Your project target -> Capabilities -> Background Modes -> mark Voice over IP & Background fetch

第107章：推送通知

参数	描述
userInfo	一个包含远程通知信息的字典，可能包括应用图标的徽章数字、提醒声音、提醒消息、通知标识符和自定义数据。

第107.1节：注册设备以接收推送通知

要注册设备以接收推送通知，请将以下代码添加到您的AppDelegate文件中的didFinishLaunchingWithOptions方法：

Swift

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
    // 应用启动后自定义的重写点。
    if UIDevice.currentDevice().systemVersion.compare(v, options: .NumericSearch) == NSOrderedAscending {
        // 如果运行的是iOS 8以下版本，则注册推送通知
        if application.respondsToSelector("registerUserNotificationSettings:") {
            let types: UIUserNotificationType = (.Alert | .Badge | .Sound)
            let settings: UIUserNotificationSettings = UIUserNotificationSettings(forTypes: types, categories: nil)

            application.registerUserNotificationSettings(settings)
            application.registerForRemoteNotifications()
        } 否则 {
            // 在 iOS 8 之前注册推送通知
            application.registerForRemoteNotificationTypes(.Alert | .Badge | .Sound)
        }
    } else {
        var center = UNUserNotificationCenter.currentNotificationCenter()
        center.delegate = self
    }
    center.requestAuthorizationWithOptions((UNAuthorizationOptionSound |
    UNAuthorizationOptionAlert | UNAuthorizationOptionBadge)) {(granted: Bool, error: NSError) -> Void in
        if !error {
            UIApplication.sharedApplication().registerForRemoteNotifications()
            // 需要此操作才能让应用处理推送通知
            print("推送注册成功。")
        } else {
            print("推送注册失败")
            print("错误: \(error.localizedDescription!) - \(error.localizedDescription)")
            print("建议: \(error.localizedRecoveryOptions) - \(error.localizedRecoverySuggestion!)")
        }
    }
    return true
}
```

Objective-C

```
#define SYSTEM_VERSION_LESS_THAN(v) ([[UIDevice currentDevice] systemVersion] compare:v
options:NSNumericSearch] == NSOrderedAscending)

if( SYSTEM_VERSION_LESS_THAN( @"10.0" ) )
{
    if ([application respondsToSelector:@selector(isRegisteredForRemoteNotifications)])
    {
```

Chapter 107: Push Notifications

Parameter	Description
userInfo	A dictionary that contains remote notification info, potentially including a badge number for the app icon, alert sound, alert message, a notification identifier, and custom data.

Section 107.1: Registering device for Push Notifications

To register your device for push notifications, add the following code to your AppDelegate file in didFinishLaunchingWithOptions method:

Swift

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
    // Override point for customization after application launch.
    if UIDevice.currentDevice().systemVersion.compare(v, options: .NumericSearch) == NSOrderedAscending {
        // Register for Push Notifications, if running iOS < 8
        if application.respondsToSelector("registerUserNotificationSettings:") {
            let types: UIUserNotificationType = (.Alert | .Badge | .Sound)
            let settings: UIUserNotificationSettings = UIUserNotificationSettings(forTypes: types, categories: nil)

            application.registerUserNotificationSettings(settings)
            application.registerForRemoteNotifications()
        } else {
            // Register for Push Notifications before iOS 8
            application.registerForRemoteNotificationTypes(.Alert | .Badge | .Sound)
        }
    } else {
        var center = UNUserNotificationCenter.currentNotificationCenter()
        center.delegate = self
        center.requestAuthorizationWithOptions((UNAuthorizationOptionSound |
        UNAuthorizationOptionAlert | UNAuthorizationOptionBadge)) {(granted: Bool, error: NSError) -> Void in
            if !error {
                UIApplication.sharedApplication().registerForRemoteNotifications()
                // required to get the app to do anything at all about push notifications
                print("Push registration success.")
            } else {
                print("Push registration FAILED")
                print("ERROR: \(error.localizedDescription!) - \(error.localizedDescription)")
                print("SUGGESTIONS: \(error.localizedRecoveryOptions) - \(error.localizedRecoverySuggestion!)")
            }
        }
        return true
    }
}
```

Objective-C

```
#define SYSTEM_VERSION_LESS_THAN(v) ([[UIDevice currentDevice] systemVersion] compare:v
options:NSNumericSearch] == NSOrderedAscending)

if( SYSTEM_VERSION_LESS_THAN( @"10.0" ) )
{
    if ([application respondsToSelector:@selector(isRegisteredForRemoteNotifications)])
    {
```

```

// iOS 8 通知
[application registerUserNotificationSettings:[UIUserNotificationSettings
settingsForTypes:(UIUserNotificationTypeSound | UIUserNotificationTypeAlert |
UIUserNotificationTypeBadge) categories:nil]];

[application registerForRemoteNotifications];
}

else
{
    // iOS 8 以下通知
    [application registerForRemoteNotificationTypes:
     (UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeAlert |
UIRemoteNotificationTypeSound)];
}

}
else
{
UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
center.delegate = self;
[center requestAuthorizationWithOptions:(UNAuthorizationOptionSound |
UNAuthorizationOptionAlert | UNAuthorizationOptionBadge) completionHandler:^(BOOL granted, NSError *
*_Nullable error)
{
    if( !error )
    {
        [[UIApplication sharedApplication] registerForRemoteNotifications]; // 需要此操作才能让应用处理推送通知
        NSLog( @"推送注册成功。" );
    }
    else
    {
        NSLog( @"推送注册失败" );
        NSLog( @"错误: %@ - %@", error.localizedDescription, error.localizedFailureReason );
        NSLog( @"建议: %@ - %@", error.localizedRecoveryOptions,
error.localizedRecoverySuggestion );
    }
}];
}

// 检查应用是否由推送通知启动
//-----
// 处理推送通知
if (launchOptions != nil)
{
    // 这里应用将由推送通知启动
    // 远程通知
    NSDictionary* dictionary1 = [launchOptions
objectForKey:UIApplicationLaunchOptionsRemoteNotificationKey];
    // 本地通知
    NSDictionary* dictionary2 = [launchOptions
objectForKey:UIApplicationLaunchOptionsLocalNotificationKey];
    if (dictionary1 != nil)
    {
        //远程通知负载
        NSLog(@"从推送通知启动: %@", dictionary1);
        //在这里处理你的推送通知
    }
    if (dictionary2 != nil)
    {
        NSLog(@"从dictionary2dictionary2dictionary2通知启动: %@", dictionary2);
    }
}

```

```

// iOS 8 Notifications
[application registerUserNotificationSettings:[UIUserNotificationSettings
settingsForTypes:(UIUserNotificationTypeSound | UIUserNotificationTypeAlert |
UIUserNotificationTypeBadge) categories:nil]];

[application registerForRemoteNotifications];
}

else
{
    // iOS < 8 Notifications
    [application registerForRemoteNotificationTypes:
     (UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeAlert |
UIRemoteNotificationTypeSound)];
}

}
else
{
UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
center.delegate = self;
[center requestAuthorizationWithOptions:(UNAuthorizationOptionSound |
UNAuthorizationOptionAlert | UNAuthorizationOptionBadge) completionHandler:^(BOOL granted, NSError *
*_Nullable error)
{
    if( !error )
    {
        [[UIApplication sharedApplication] registerForRemoteNotifications]; // required to
get the app to do anything at all about push notifications
        NSLog( @"Push registration success." );
    }
    else
    {
        NSLog( @"Push registration FAILED" );
        NSLog( @"ERROR: %@ - %@", error.localizedDescription, error.localizedFailureReason );
        NSLog( @"SUGGESTIONS: %@ - %@", error.localizedRecoveryOptions,
error.localizedRecoverySuggestion );
    }
}];
}

// to check if your App lunch from Push notification
//-----
// Handel Push notification
if (launchOptions != nil)
{
    // Here app will open from pushnotification
    // RemoteNotification
    NSDictionary* dictionary1 = [launchOptions
objectForKey:UIApplicationLaunchOptionsRemoteNotificationKey];
    // LocalNotification
    NSDictionary* dictionary2 = [launchOptions
objectForKey:UIApplicationLaunchOptionsLocalNotificationKey];
    if (dictionary1 != nil)
    {
        // RemoteNotification Payload
        NSLog(@"Launched from push notification: %@", dictionary1);
        // here handle your push notification
    }
    if (dictionary2 != nil)
    {
        NSLog(@"Launched from dictionary2dictionary2dictionary2 notification: %@", dictionary2);
    }
}

```

```

dictionary2);
    double delayInSeconds = 7;
dispatch_time_t popTime = dispatch_time(DISPATCH_TIME_NOW, (int64_t)(delayInSeconds *
NSEC_PER_SEC));
dispatch_after(popTime, dispatch_get_main_queue(), ^(void){
    // [self addMessageFromRemoteNotification:dictionary2 updateUI:NO];
});
}

else
{}
//-----

```

上述代码将尝试与APNs服务器通信以获取设备令牌（前提是您的iOS配置文件中已启用APNs）。

一旦与APNs服务器建立了可靠连接，服务器将为您提供设备令牌。

在添加上述代码后，向AppDelegate类中添加以下方法：

Swift

```

func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    print("DEVICE TOKEN = \(deviceToken)")
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print(error)
}

```

Objective-C

```

- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
NSString * deviceTokenString = [[[deviceToken description]
                                stringByReplacingOccurrencesOfString: @"<" withString: @""]
                                stringByReplacingOccurrencesOfString: @">>" withString: @""]
                                stringByReplacingOccurrencesOfString: @" " withString: @""];

NSLog(@"生成的设备令牌字符串是 : %@",deviceTokenString);
}

- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error
{
NSLog(@"获取令牌失败，错误 : %@", error.description);
}

```

上述方法根据注册成功或失败的场景被调用。

成功场景调用：

Swift

```

func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    print("DEVICE TOKEN = \(deviceToken)")
}

```

```

dictionary2);
    double delayInSeconds = 7;
dispatch_time_t popTime = dispatch_time(DISPATCH_TIME_NOW, (int64_t)(delayInSeconds *
NSEC_PER_SEC));
dispatch_after(popTime, dispatch_get_main_queue(), ^(void){
    // [self addMessageFromRemoteNotification:dictionary2 updateUI:NO];
});
}

else
{}
//-----

```

The above code will try to communicate with APNs server to get device token (prerequisites are you have APNs enabled in your iOS provisioning profile).

Once it establishes reliable connection with APNs server, the server provides you a device token.

After adding the code above, add these methods to the AppDelegate class:

Swift

```

func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    print("DEVICE TOKEN = \(deviceToken)")
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print(error)
}

```

Objective-C

```

- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
NSString * deviceTokenString = [[[deviceToken description]
                                stringByReplacingOccurrencesOfString: @"<" withString: @""]
                                stringByReplacingOccurrencesOfString: @">>" withString: @""]
                                stringByReplacingOccurrencesOfString: @" " withString: @"";

NSLog(@"The generated device token string is : %@",deviceTokenString);
}

- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error
{
    NSLog(@"Failed to get token, error: %@", error.description);
}

```

The above methods are called according to registration success or failure scenario.

Success scenario calls:

Swift

```

func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    print("DEVICE TOKEN = \(deviceToken)")
}

```

```
}
```

在 Swift3 中：

```
@objc(userNotificationCenter:willPresentNotification:withCompletionHandler:) @available(iOS 10.0, *)
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void)
{
    //在前台显示通知。
    print("Userinfo2 \(notification.request.content.userInfo)")
}
```

Objective-C

```
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
if(application.applicationState == UIApplicationStateInactive) {
    NSLog(@"非活动状态 - 用户在应用关闭或后台时点击了通知");

    //执行一些任务
    [self handlePushNotification(userInfo];
}

else if (application.applicationState == UIApplicationStateBackground) {
    NSLog(@"应用后台 - 应用在后台时收到通知");
    [self handlePushNotification(userInfo];
}

else {
    NSLog(@"应用激活 - 应用打开时收到通知");
    //显示应用内横幅
    //执行任务
}
}
```

失败场景调用：

Swift

```
func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print(error)
}
```

Objective-C

```
- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error
```

注意

如果上述方法都没有被调用，说明您的设备无法与APNs服务器建立可靠连接，可能是由于网络访问问题。

```
}
```

In Swift3:

```
@objc(userNotificationCenter:willPresentNotification:withCompletionHandler:) @available(iOS 10.0, *)
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void)
{
    //To show notifications in foreground.
    print("Userinfo2 \(notification.request.content.userInfo)")
}
```

Objective-C

```
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
if(application.applicationState == UIApplicationStateInactive) {
    NSLog(@"Inactive - the user has tapped in the notification when app was closed or in
background");
    //do some tasks
    [self handlePushNotification(userInfo];
}

else if (application.applicationState == UIApplicationStateBackground) {
    NSLog(@"application Background - notification has arrived when app was in background");
    [self handlePushNotification(userInfo];
}

else {
    NSLog(@"application Active - notification has arrived while app was opened");
    //Show an in-app banner
    //do tasks
}
}
```

Failure scenario calls:

Swift

```
func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print(error)
}
```

Objective-C

```
- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error
```

Note

If none of the above methods are getting called, your device is not able to create reliable connection with APNs server, which might be because of internet access problems.

第107.2节：注册应用ID以用于推送通知

所需物品

Section 107.2: Registering App ID for use with Push Notifications

Things you need

- 已付费的苹果开发者计划会员资格一个有效的应
- 用程序ID和标识符（如 com.example.MyApp），且之前未被使用过
- 访问developer.apple.com和会员中心
- 一台用于测试的iOS设备（因为推送通知在模拟器上无法使用）

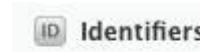
在苹果开发者中心为应用程序ID启用APNs访问权限

1- 登录developer.apple.com会员中心（首页上的账户链接）

 Account

2- 进入“证书”

3- 从左侧面板选择“应用程序ID”

 Identifiers

 App IDs

4- 点击右上角的“+”



5- 添加带有推送通知选项勾选的应用程序ID

6- 点击已创建的应用程序ID并选择编辑

7- 在推送通知面板点击配置

8- 在你的Mac上打开钥匙串访问应用

9- 在钥匙串访问菜单中，点击证书助理 -> 从证书颁发机构请求证书

10- 在第一个文本框中输入你的邮箱

11- 在第二个文本框中输入你的姓名

- A paid Apple Developer Program Membership
- A valid App ID and identifier for your app (like com.example.MyApp) which is not used before anywhere
- Access to developer.apple.com and Member Center
- An iOS Device to test (as Push Notifications don't work on Simulator)

Enabling the APNs access for App ID in Apple Developer Center

1- Log in to developer.apple.com Member Center (the Account link on the home page)

 Account

2- Go to "Certificates"

3- Select "App ID" from left panel

 Identifiers

 App IDs

4- Click on "+" on top right



5- Add App ID with Push Notifications option checked

6- Click on created App ID and select Edit

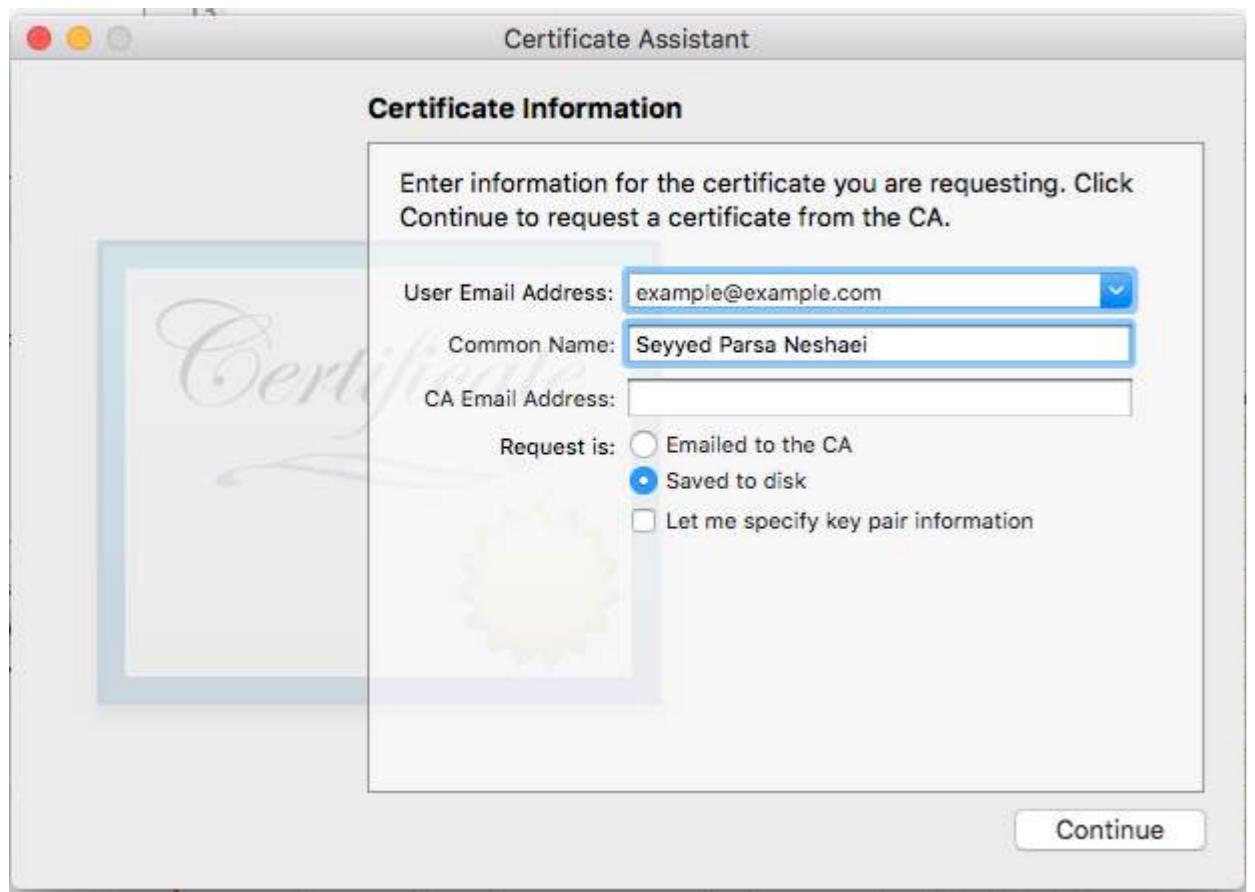
7- Click Configure in Push Notifications panel

8- Open Keychain Access app in your Mac

9- From Keychain Access menu, click Certificate Assistant -> Request a Certificate from a Certificate Authority

10- Enter your mail in the first text field

11- Enter your name in the second text field



12- CA邮箱地址留空

13- 选择保存到磁盘，而不是通过邮件发送给CA

14- 点击继续并上传生成的文件

15- 下载Apple生成的文件并在钥匙串访问打开时打开它

在Xcode中启用APNs访问

1- 选择你的项目

2- 打开功能 (Capabilities) 标签

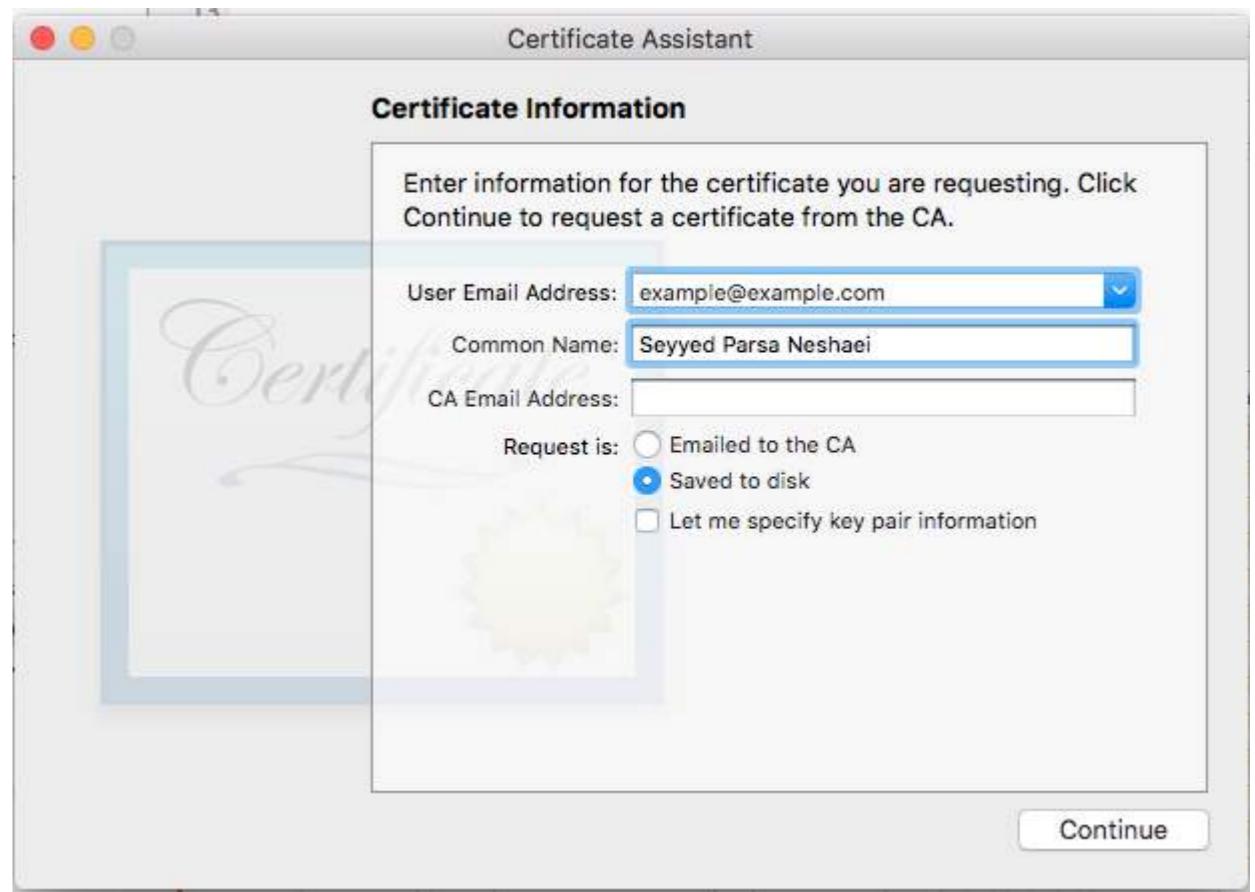
3- 找到推送通知并开启

4- 找到后台模式并开启，勾选远程通知

第107.3节：测试推送通知

即使在服务器端尚未准备好之前，测试推送通知的工作原理始终是一个好习惯，以确保你这边的设置正确。使用以下PHP脚本给自己发送推送通知是相当简单的。

1. 将脚本保存为文件（例如 send_push.php），并与您的证书（开发或
生产环境）放在同一文件夹中
2. 编辑它以填写您的设备令牌和证书密码
3. 选择用于建立连接的正确路径，dev_path 或 prod_path（这是脚本中“打开与 APNS 服务器的连接”发生的地方）
4. 在终端中切换到该文件夹并运行命令 'php send_push'



12- Leave CA Email Address empty

13- Select Saved to disk rather than Emailed to the CA

14- Click Continue and upload the generated file

15- Download the generated file by Apple and open it while Keychain Access is open

Enabling the APNs access in Xcode

1- Select your project

2- Open Capabilities tab

3- Find Push Notifications and turn it on

4- Find Background Modes and turn it on and check Remote Notifications

Section 107.3: Testing push notifications

It is always a good practice to test how push notifications work even before you have your server side ready for them, just to make sure that everything is set up correctly on your side. It is quite easy to send yourself a push notification using a following PHP script.

1. Save the script as a file (send_push.php for example) in the same folder as your certificate (development or production)
2. Edit it to put your device token, password from the certificate
3. Choose the correct path for opening a connection, dev_path or prod_path (this is where 'Open a connection to the APNS server' happens in the script)
4. cd to the folder in Terminal and run command 'php send_push'

5. 在您的设备上接收通知

```
<?php

// 在此处填写您的设备令牌 (无空格) :
$deviceToken = '20128697f872d7d39e48c4a61f50cb11d77789b39e6fc6b4cd7ec80582ed5229';
// 将您的最终pem证书名称放在这里。它应该与此脚本位于同一文件夹中
$cert_name = 'final_cert.pem';
// 在此处填写您的私钥密码短语：
$passphrase = '1234';

// 示例点
$alert = 'Hello world!';
$event = 'new_incoming_message';

// 您可以根据所使用的证书类型选择任一路径
$dev_path = 'ssl://gateway.sandbox.push.apple.com:2195';
$prod_path = 'ssl://gateway.push.apple.com:2195';

////////////////////////////////////////////////////////////////

$cxt = stream_context_create();
stream_context_set_option($cxt, 'ssl', 'local_cert', $cert_name);
stream_context_set_option($cxt, 'ssl', 'passphrase', $passphrase);

// 打开与 APNS 服务器的连接
$fp = stream_socket_client(
    $dev_path, $err,
    $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT, $cxt);

if (!$fp)
    exit("连接失败: $err $errstr" . PHP_EOL);

echo '已连接到 APNS' . PHP_EOL;

// 创建负载主体
// 应尽可能简短
// 如果通知未能送达，很可能是
// 生成的消息太长
$body['aps'] = array(
    'alert' => $alert,
    'sound' => 'default',
    'event' => $event
);

// 将负载编码为 JSON
	payload = json_encode($body);

// 构建二进制通知
$msg = chr(0) . pack('n', 32) . pack('H*', $deviceToken) . pack('n', strlen($payload)) . $payload;

// 发送到服务器
$result = fwrite($fp, $msg, strlen($msg));

if (!$result)
    echo '消息未发送' . PHP_EOL;
else
    echo '消息发送成功' . PHP_EOL;

// 关闭与服务器的连接
fclose($fp);
```

5. Receive the notification on your device

```
<?php

// Put your device token here (without spaces):
$deviceToken = '20128697f872d7d39e48c4a61f50cb11d77789b39e6fc6b4cd7ec80582ed5229';
// Put your final pem cert name here. it is supposed to be in the same folder as this script
$cert_name = 'final_cert.pem';
// Put your private key's passphrase here:
$passphrase = '1234';

// sample point
$alert = 'Hello world!';
$event = 'new_incoming_message';

// You can choose either of the paths, depending on what kind of certificate you are using
$dev_path = 'ssl://gateway.sandbox.push.apple.com:2195';
$prod_path = 'ssl://gateway.push.apple.com:2195';

////////////////////////////////////////////////////////////////

$cxt = stream_context_create();
stream_context_set_option($cxt, 'ssl', 'local_cert', $cert_name);
stream_context_set_option($cxt, 'ssl', 'passphrase', $passphrase);

// Open a connection to the APNS server
$fp = stream_socket_client(
    $dev_path, $err,
    $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT, $cxt);

if (!$fp)
    exit("Failed to connect: $err $errstr" . PHP_EOL);

echo 'Connected to APNS' . PHP_EOL;

// Create the payload body
// it should be as short as possible
// if the notification doesn't get delivered that is most likely
// because the generated message is too long
$body['aps'] = array(
    'alert' => $alert,
    'sound' => 'default',
    'event' => $event
);

// Encode the payload as JSON
$payload = json_encode($body);

// Build the binary notification
$msg = chr(0) . pack('n', 32) . pack('H*', $deviceToken) . pack('n', strlen($payload)) . $payload;

// Send it to the server
$result = fwrite($fp, $msg, strlen($msg));

if (!$result)
    echo 'Message not delivered' . PHP_EOL;
else
    echo 'Message successfully delivered' . PHP_EOL;

// Close the connection to the server
fclose($fp);
```

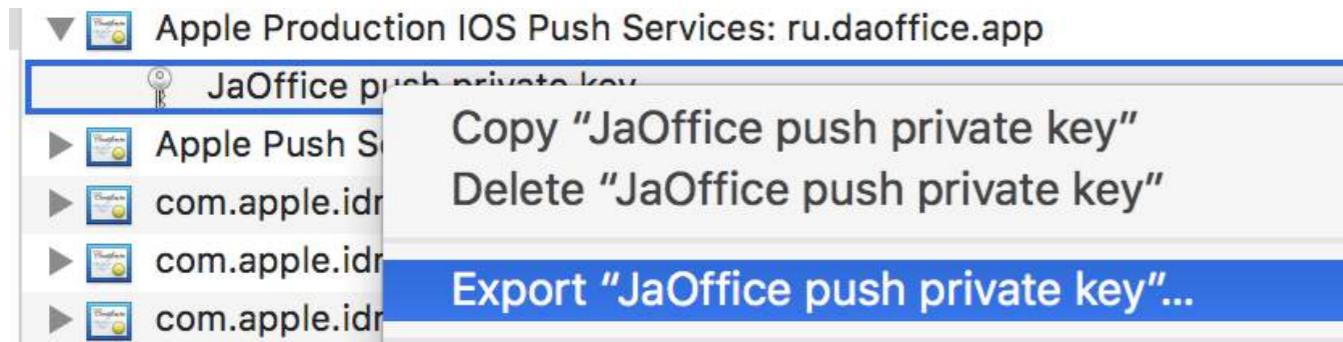
第107.4节：检查您的应用是否已注册推送通知

Swift

```
let isPushEnabled = UIApplication.sharedApplication().isRegisteredForRemoteNotifications()
```

第107.5节：从您的.cer文件生成.pem证书，以传递给服务器开发人员

1. 将aps.cer保存到一个文件夹中
2. 打开“钥匙串访问”，将该证书下的密钥导出为.p12文件（命名为key.p12）。操作方法是右键点击该密钥，选择导出。保存到与步骤1相同的文件夹。导出时会提示输入密码，随便设置一个并记住它。



3. 在终端中切换到该文件夹并执行以下命令：
4. 将.cer转换为.pem证书

```
openssl x509 -in aps.cer -inform der -out aps.pem
```

5. 将您的密钥转换为.pem格式。打开密钥时，输入步骤2中导出时设置的密码。然后，输入另一个密码以保护导出的文件。系统会提示您输入两次以确认。

```
openssl pkcs12 -nocerts -out key.pem -in key.p12
```

6. 将文件合并为一个最终文件

```
cat key.pem aps.pem > final_cert.pem
```

7. final_cert.pem 是最终结果。将其连同步骤5中的密码一起交给服务器开发人员，这样他们才能使用受保护的证书。

第107.6节：取消注册推送通知

要以编程方式取消注册远程通知，可以使用

Objective-C

```
[[UIApplication sharedApplication] unregisterForRemoteNotifications];
```

Swift

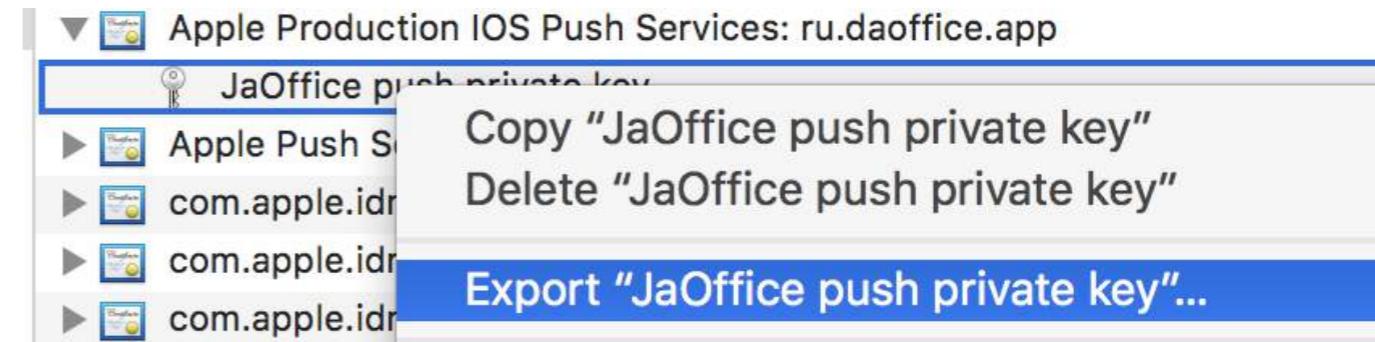
Section 107.4: Checking if your app is already registered for Push Notification

Swift

```
let isPushEnabled = UIApplication.sharedApplication().isRegisteredForRemoteNotifications()
```

Section 107.5: Generating a .pem certificate from your .cer file, to pass on to the server developer

1. Save aps.cer to a folder
2. Open "Keychain access" and export the key that is under that certificate to a .p12 file (call it key.p12). To do that right click on it and choose Export. Save it to the same folder as step 1. On export you will be prompted for a password. Make something up and memorize it.



3. cd to that folder in Terminal and execute the following commands:
4. Convert .cer to a .pem certificate

```
openssl x509 -in aps.cer -inform der -out aps.pem
```

5. Convert your key to .pem format. To open the key, enter the password you exported it with from the keychain, in step 2. Then, enter another password that will protect the exported file. You will be prompted to enter it twice for confirmation.

```
openssl pkcs12 -nocerts -out key.pem -in key.p12
```

6. Merge the files into one final file

```
cat key.pem aps.pem > final_cert.pem
```

7. The final_cert.pem is the final result. Pass it on to server developers with the password from step 5, so that they will be able to use the protected certificate.

Section 107.6: Unregistering From Push Notifications

To unregister from Remote Notifications programmatically you can use

Objective-C

```
[[UIApplication sharedApplication] unregisterForRemoteNotifications];
```

Swift

```
UIApplication.sharedApplication().unregisterForRemoteNotifications()
```

这类似于进入手机设置，手动关闭该应用的通知。

注意：在某些罕见情况下，您可能需要这样做（例如：当您的应用不再支持推送通知时）

如果您只是想允许用户暂时禁用通知，您应该实现一个方法，在服务器的数据库中移除设备令牌。否则，如果您仅在设备本地禁用通知，服务器仍会发送消息。

第107.7节：设置应用图标徽章数字

使用以下代码片段从应用程序内部设置徽章数字（假设 someNumber 已经声明）：

Objective-C

```
[UIApplication sharedApplication].applicationIconBadgeNumber = someNumber;
```

Swift

```
UIApplication.shared.applicationIconBadgeNumber = someNumber
```

要完全移除徽章，只需将 someNumber = 0 设置即可。

第107.8节：注册（非交互式）推送通知

建议将注册推送通知的逻辑添加到AppDelegate.swift中，因为回调函数（成功、失败）会在那里被调用。注册只需执行以下操作：

```
let application = UIApplication.sharedApplication()
let settings = UIUserNotificationSettings(forTypes: [.Alert, .Badge, .Sound], categories: nil)
application.registerUserNotificationSettings(settings)
```

然后会调用回调函数didRegisterUserNotificationSettings，在这种情况下，你只需这样触发注册：

```
func application(application: UIApplication, didRegisterUserNotificationSettings
notificationSettings: UIUserNotificationSettings) {
    application.registerForRemoteNotifications()
}
```

在这种情况下，系统会弹出提示，询问是否允许接收推送通知。以下回调函数之一将被调用：

```
func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    let tokenChars = UnsafePointer<CChar>(deviceToken.bytes)
    var tokenString = ""

    for i in 0..
```

```
UIApplication.sharedApplication().unregisterForRemoteNotifications()
```

this is similar to going into the setting of your phone and manually switching off Notifications for the application.

NOTE: There may be rare cases where you would need this(eg: when your app no longer supports push notifications)

If you just want to allow the user to temporarily disable Notifications. You should implement a method to remove device token in the database on your server. else if you only disable Notification locally on your device your server will still send messages.

Section 107.7: Setting the application icon badge number

Use the following piece of code to set the badge number from within your application (suppose someNumber has been declared before):

Objective-C

```
[UIApplication sharedApplication].applicationIconBadgeNumber = someNumber;
```

Swift

```
UIApplication.shared.applicationIconBadgeNumber = someNumber
```

In order to *remove* the badge completely, just set someNumber = 0.

Section 107.8: Registering for (Non Interactive) Push Notification

The logic of registering for push notification is recommended to be added in AppDelegate.swift as the callback functions (success, failure) will be called their. To register just do the following:

```
let application = UIApplication.sharedApplication()
let settings = UIUserNotificationSettings(forTypes: [.Alert, .Badge, .Sound], categories: nil)
application.registerUserNotificationSettings(settings)
```

Then the callback function didRegisterUserNotificationSettings will be called and in that case you just trigger the register like this:

```
func application(application: UIApplication, didRegisterUserNotificationSettings
notificationSettings: UIUserNotificationSettings) {
    application.registerForRemoteNotifications()
}
```

And in that case and system alert will be shown asking for permission to receive push notification. One of the following callback functions will be called:

```
func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    let tokenChars = UnsafePointer<CChar>(deviceToken.bytes)
    var tokenString = ""

    for i in 0..
```

```

        print("推送令牌: \(tokenString)")
    }

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print("注册远程通知失败, 错误: \(error)")
}

```

在极少数情况下，既不会调用成功回调函数，也不会调用失败回调函数。这种情况发生在您遇到网络连接问题或APNS沙盒服务宕机时。系统会向APNS发起API调用进行验证，验证失败将导致两个回调函数都不会被调用。请访问Apple系统状态以确认服务是否正常。

第107.9节：处理推送通知

用户点击推送通知后，将调用以下回调函数。您可以解析JSON以获取后端发送的任何特定信息，这将有助于实现深度链接：

Swift

```

func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject :
AnyObject]) {
    print("收到通知: \(userInfo)")
}

```

Objective C

```

- (void)application:(UIApplication *)application didReceiveRemoteNotification: (NSDictionary
*)userInfo
{
    NSLog(@"收到通知: %@", userInfo);
}

```

iOS 10

```

#define SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO(v) ([[UIDevice currentDevice] systemVersion]
compare:v options:NSNumericSearch] != NSOrderedAscending)

-(void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
*)userInfo fetchCompletionHandler:(void
(^)(UIBackgroundFetchResult))completionHandler
{
    // iOS 10 将通过其他方法处理通知
    NSLog(@"收到通知: %@", userInfo);

    if( SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO( @"10.0" ) )
    {
        NSLog( @"iOS 版本 >= 10. 让 NotificationCenter 处理这个." );
        // 设置成员变量以告知新代理这是后台
        return;
    }
}

```

```

        print("Push token: \(tokenString)")
    }

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print("didFailToRegisterForRemoteNotificationsWithError: \(error)")
}

```

In very rare cases, neither success or failure callback functions are called. This happens when you have internet connection problems or the APNS Sandbox is down. The system do an API call to APNS to do some verification, failing to do so will lead to none of the two callbacks functions will be called. Visit [Apple system status](#) to make sure its fine.

Section 107.9: Handling Push Notification

Once user clicks on a push notification, the following callback function will be called. You can parse the JSON to get any specific info sent from backend that will help you in deep linking:

Swift

```

func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject :
AnyObject]) {
    print("Received notification: \(userInfo)")
}

```

Objective C

```

- (void)application:(UIApplication *)application didReceiveRemoteNotification: (NSDictionary
*)userInfo
{
    NSLog(@"Received notification: %@", userInfo);
}

```

iOS 10

```

#define SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO(v) ([[UIDevice currentDevice] systemVersion]
compare:v options:NSNumericSearch] != NSOrderedAscending)

-(void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
*)userInfo fetchCompletionHandler:(void
(^)(UIBackgroundFetchResult))completionHandler
{
    // iOS 10 will handle notifications through other methods
    NSLog(@"Received notification: %@", userInfo);

    if( SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO( @"10.0" ) )
    {
        NSLog( @"iOS version >= 10. Let NotificationCenter handle this one." );
        // set a member variable to tell the new delegate that this is background
        return;
    }
}

```

```

NSLog( @"处理推送, didReceiveRemoteNotification: %@", userInfo );

// 自定义代码处理通知内容

if( [UIApplication sharedApplication].applicationState == UIApplicationStateInactive )
{
    NSLog( @"非活动状态" );
    completionHandler( UIBackgroundFetchResultNewData );
}
else if( [UIApplication sharedApplication].applicationState == UIApplicationStateBackground )
{
    NSLog( @"后台" );
    completionHandler( UIBackgroundFetchResultNewData );
}
else
{
    NSLog( @"foreground" );
    completionHandler( UIBackgroundFetchResultNewData );
}

- (void)userNotificationCenter:(UNUserNotificationCenter *)center
    willPresentNotification:(UNNotification *)notification
    withCompletionHandler:(void (^)(UNNotificationPresentationOptions options))completionHandler
{
    NSLog( @"处理前台推送" );
    // 自定义代码处理应用处于前台时的推送
    NSLog(@"%@", notification.request.content.userInfo);
}

- (void)userNotificationCenter:(UNUserNotificationCenter *)center
    didReceiveNotificationResponse:(UNNotificationResponse *)response
    withCompletionHandler:(void (^)(() ))completionHandler
{
    NSLog( @"处理后台或关闭状态下的推送" );// 如果你在 didReceiveRemoteNotification 中设置了成员变量, 你将知道这是来自关闭还是后台
    NSLog(@"%@", response.notification.request.content.userInfo);
}

```

```

NSLog( @"HANDLE PUSH, didReceiveRemoteNotification: %@", userInfo );

// custom code to handle notification content

if( [UIApplication sharedApplication].applicationState == UIApplicationStateInactive )
{
    NSLog( @"INACTIVE" );
    completionHandler( UIBackgroundFetchResultNewData );
}
else if( [UIApplication sharedApplication].applicationState == UIApplicationStateBackground )
{
    NSLog( @"BACKGROUND" );
    completionHandler( UIBackgroundFetchResultNewData );
}
else
{
    NSLog( @"foreground" );
    completionHandler( UIBackgroundFetchResultNewData );
}

- (void)userNotificationCenter:(UNUserNotificationCenter *)center
    willPresentNotification:(UNNotification *)notification
    withCompletionHandler:(void (^)(UNNotificationPresentationOptions options))completionHandler
{
    NSLog( @"Handle push from foreground" );
    // custom code to handle push while app is in the foreground
    NSLog(@"%@", notification.request.content.userInfo);
}

- (void)userNotificationCenter:(UNUserNotificationCenter *)center
    didReceiveNotificationResponse:(UNNotificationResponse *)response
    withCompletionHandler:(void (^)(() ))completionHandler
{
    NSLog( @"Handle push from background or closed" );
    // if you set a member variable in didReceiveRemoteNotification, you will know if this is from
    // closed or background
    NSLog(@"%@", response.notification.request.content.userInfo);
}

```

第108章：丰富推送的扩展 通知 - iOS 10。

iOS 10 为我们带来了UserNotifications.framework，这是本地/远程通知的新API。它支持直接从通知中查看媒体附件或回复消息。

通知内容包括：标题、副标题、正文和附件。附件可以包含最大50MB的图片/GIF/视频。

第108.1节：通知内容扩展

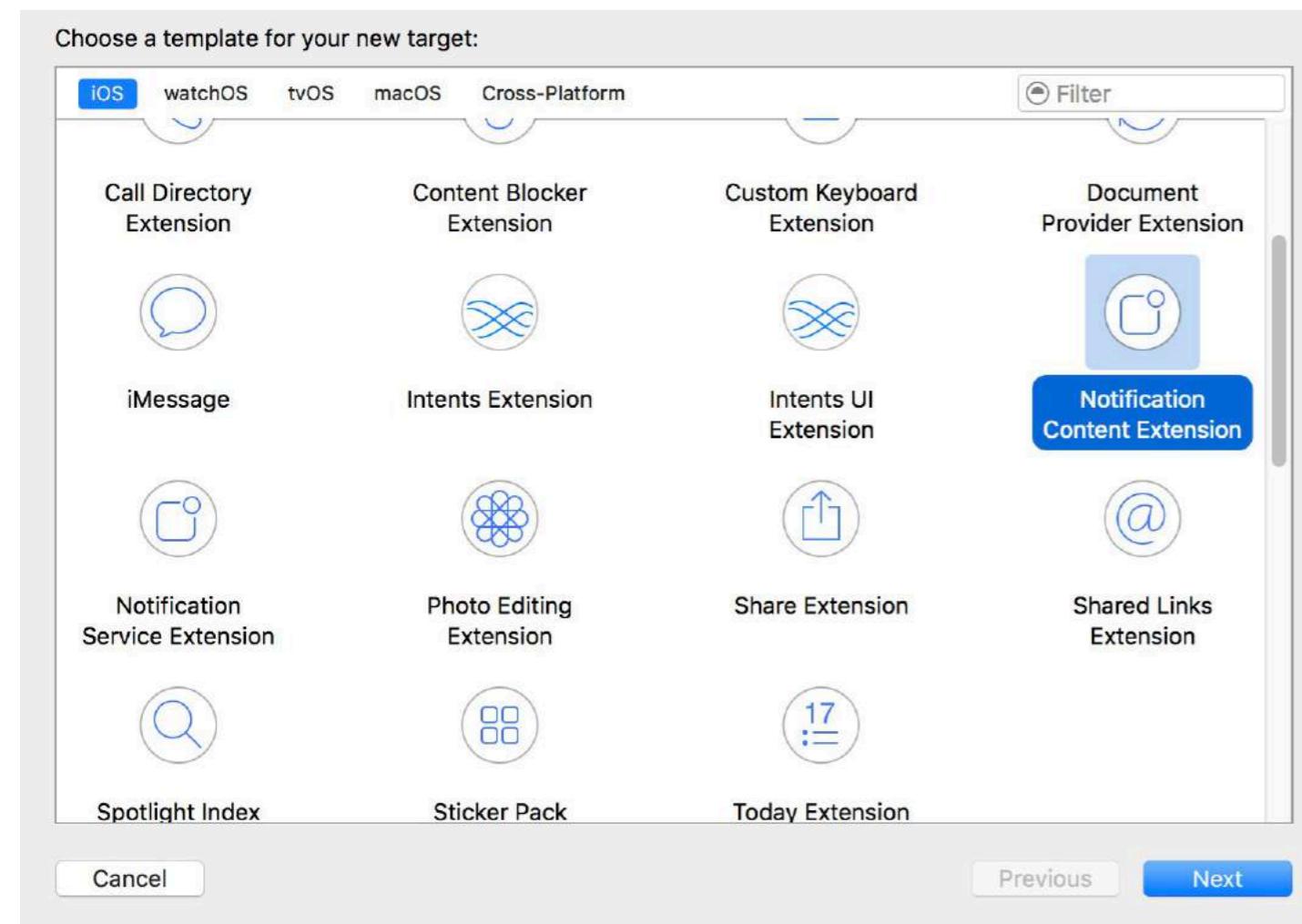
我们为什么需要它？

内容扩展帮助我们在通知展开时创建自定义用户界面。

您可以使用此框架定义一个扩展，该扩展接收通知数据并提供相应的视觉呈现。您的扩展还可以响应与这些通知相关的自定义操作。

第108.2节：实现

1. 在xCode导航器窗口中，进入目标部分。点击添加新目标。
2. 选择通知内容扩展模板：



3. 在您的info.plist文件中，为UNNotificationExtensionCategory键设置标识符：

Chapter 108: Extension for rich Push Notification - iOS 10.

iOS 10 gave us UserNotifications.framework, the new API for local/remote notifications. It offers viewing media attachments or responding to messages right from the notification.

Notification content consists of: title, subtitle, body and attachment. Attachment can contain images/gifs/videos up to 50 mb.

Section 108.1: Notification Content Extension

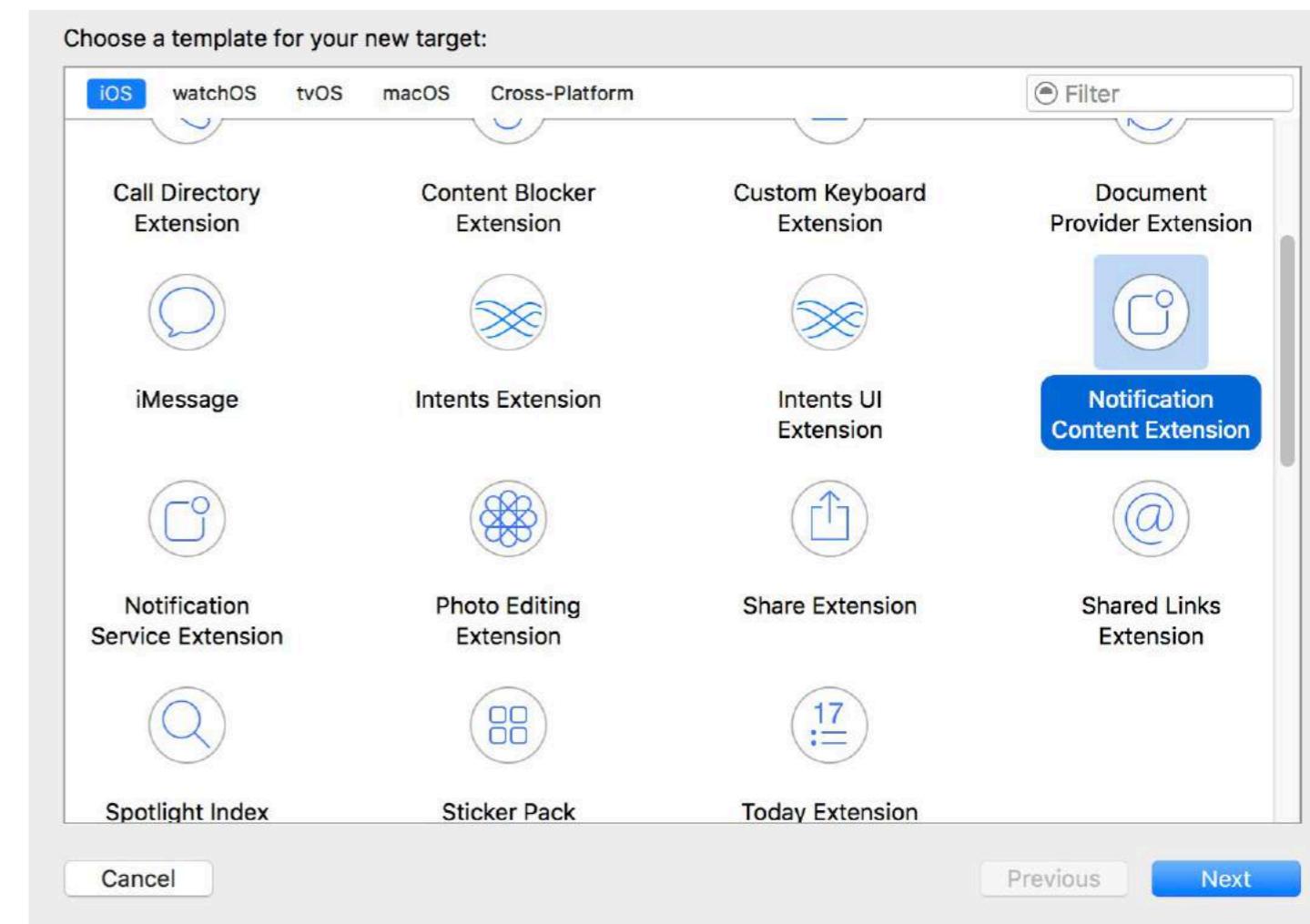
Why do we need it?

Content extension helps us to create custom user interface upon notification expansion.

You use this framework to define an extension that receives the notification data and provides the corresponding visual representation. Your extension can also respond to custom actions associated with those notifications.

Section 108.2: Implementation

1. In xCode Navigator window go to Targets section. Press Add New Target.
2. Select Notification Content Extension template:



3. In your info.plist file set the identifier for UNNotificationExtensionCategory key:

▼ NSExtension	Dictionary (3 items)
▼ NSExtensionAttributes	Dictionary (3 items)
UNNotificationExtensionDefaultContentHidden	Boolean YES
UNNotificationExtensionCategory	String customContentIdentifier
UNNotificationExtensionInitialContentSizeRatio	Number 0.7
NSExtensionMainStoryboard	String MainInterface
NSExtensionPointIdentifier	String com.apple.usernotifications.content-extension

NSExtensionAttributes :

UNNotificationExtensionCategory (必填)

该键的值是一个字符串或字符串数组。每个字符串包含由应用使用 UNNotification Category 类声明的类别标识符。

UNNotificationExtensionInitialContentSizeRatio (必填)

表示视图控制器视图初始大小的数字，按其高度与宽度的比例表示。

UNNotificationExtensionDefaultContentHidden (可选) 设置为 YES

S 时，系统仅在通知界面显示您的自定义视图控制器。设置为 NO 时，系统会在显示您的视图控制器内容的同时显示默认通知内容。

UNNotificationExtensionOverridesDialogTitle (可选) 该键的值为布

尔值。设置为 true 时，系统使用您的视图控制器的 title 属性作为通知标题。设置为 false 时，系统将通知标题设置为您的应用名称。如果未指定此键，默认值为 false。

4. 在 NotificationViewController.swift 文件中创建自定义视图

5. 添加新的category key，并将其值设置为我们在 Info.plist (步骤3) 中输入的内容：

推送：

```
{
  aps: {
    alert: { ... },
    类别: 'io.swifting.notification-类别'
  }
}
```

本地：

```
let mutableNotificationContent = UNMutableNotificationContent()
mutableNotificationContent.category = "io.swifting.notification-category"
mutableNotificationContent.title = "Swiftling.io 通知"
mutableNotificationContent.subtitle = "Swiftling.io 呈现"
mutableNotificationContent.body = "自定义通知"
```

另请查看官方 API 参考：

https://developer.apple.com/reference/usernotificationsui/unnotificationcontentextension?utm_source=swifting.io&utm_medium=web&utm_campaign=blog%20post

▼ NSExtension	Dictionary (3 items)
▼ NSExtensionAttributes	Dictionary (3 items)
UNNotificationExtensionDefaultContentHidden	Boolean YES
UNNotificationExtensionCategory	String customContentIdentifier
UNNotificationExtensionInitialContentSizeRatio	Number 0.7
NSExtensionMainStoryboard	String MainInterface
NSExtensionPointIdentifier	String com.apple.usernotifications.content-extension

NSExtensionAttributes:

UNNotificationExtensionCategory (Required)

The value of this key is a string or an array of strings. Each string contains the identifier of a category declared by the app using the UNNotificationCategory class.

UNNotificationExtensionInitialContentSizeRatio (Required)

Number that represents the initial size of your view controller's view expressed as a ratio of its height to its width.

UNNotificationExtensionDefaultContentHidden (Optional)

When set to YES, the system displays only your custom view controller in the notification interface. When set to NO, the system displays the default notification content in addition to your view controller's content.

UNNotificationExtensionOverridesDialogTitle (Optional)

The value of this key is a Boolean. When set to true, the system uses the title property of your view controller as the title of the notification. When set to false, the system sets the notification's title to the name of your app. If you do not specify this key, the default value is set to false.

4. Create custom view in NotificationViewController.swift file

5. Add new category key and set its value to what we typed in the Info.plist (step 3):

Push:

```
{
  aps: {
    alert: { ... },
    category: 'io.swifting.notification-category'
  }
}
```

Local:

```
let mutableNotificationContent = UNMutableNotificationContent()
mutableNotificationContent.category = "io.swifting.notification-category"
mutableNotificationContent.title = "Swiftling.io Notifications"
mutableNotificationContent.subtitle = "Swiftling.io presents"
mutableNotificationContent.body = "Custom notifications"
```

Also check out the official API reference:

https://developer.apple.com/reference/usernotificationsui/unnotificationcontentextension?utm_source=swifting.io&utm_medium=web&utm_campaign=blog%20post

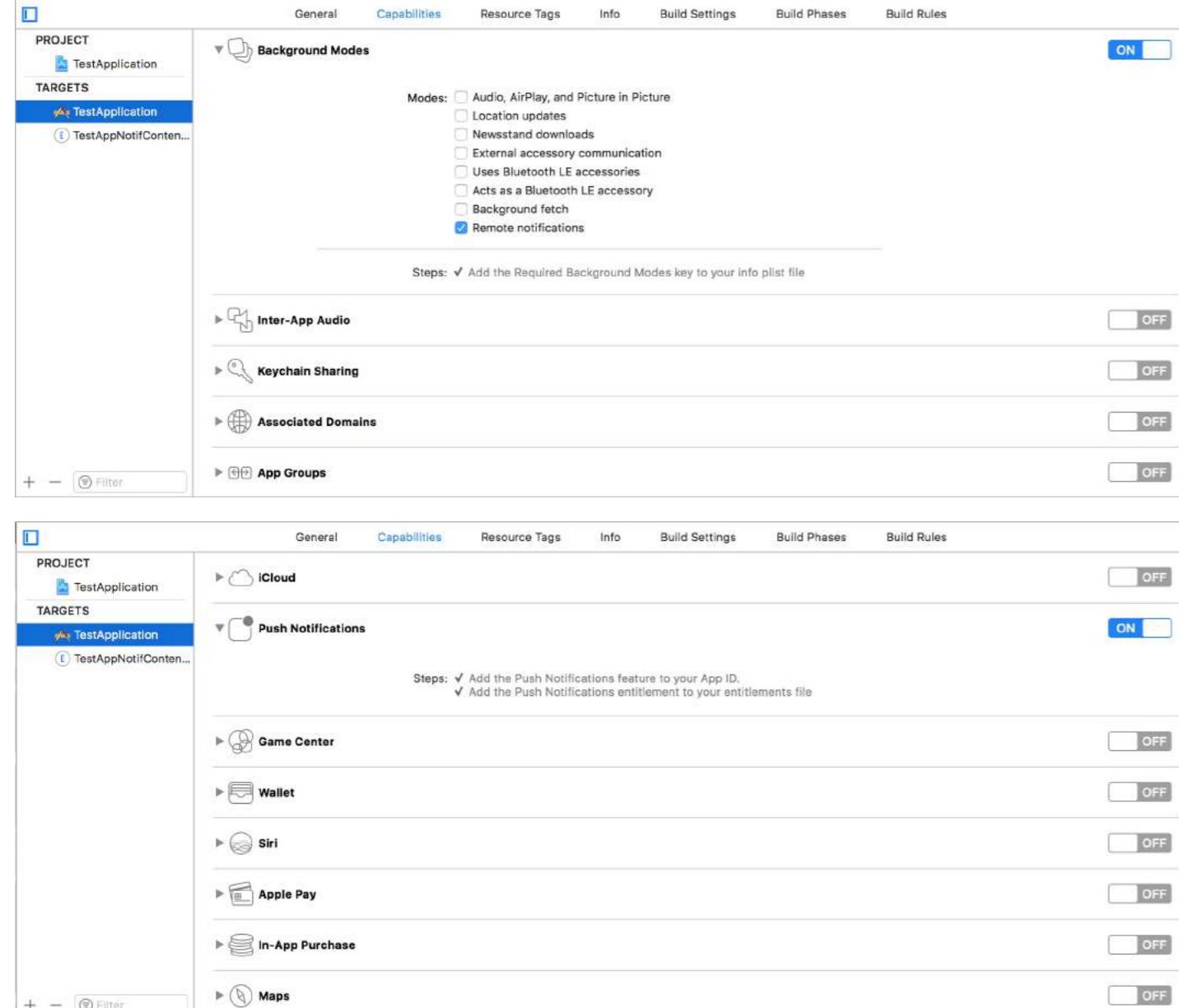
第109章：丰富通知

丰富通知让您可以在本地和远程通知在用户设备上出现时自定义通知的外观。丰富通知主要包括 UNNotificationServiceExtension 和 UNNotificationContentExtension，即以扩展的方式显示普通通知。

第109.1节：创建一个简单的 UNNotificationContentExtension

步骤 1

使环境适合通知。确保已启用后台模式和推送通知



步骤 2：创建 UNNotificationContentExtension

点击底部的+图标，创建一个目标模板并选择通知内容扩展 -> 下一步
-> 为内容扩展创建一个名称 -> 完成

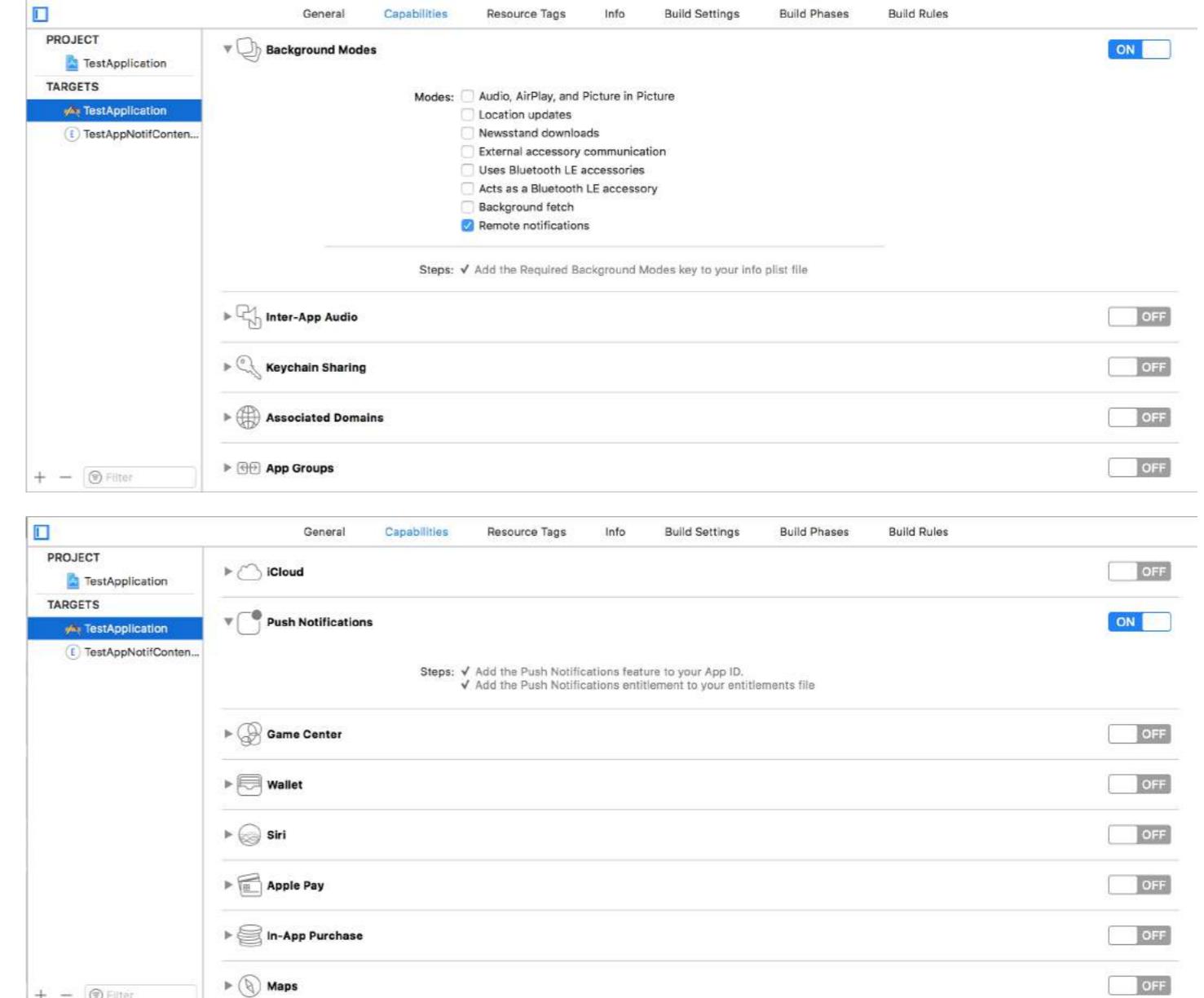
Chapter 109: Rich Notifications

The Rich Notifications lets you customize the appearance of local and remote notifications when they appear on the user's device. Rich notification mostly includes UNNotificationServiceExtension and UNNotificationContentExtension ie displaying the normal notification in an extended manner

Section 109.1: Creating a simple UNNotificationContentExtension

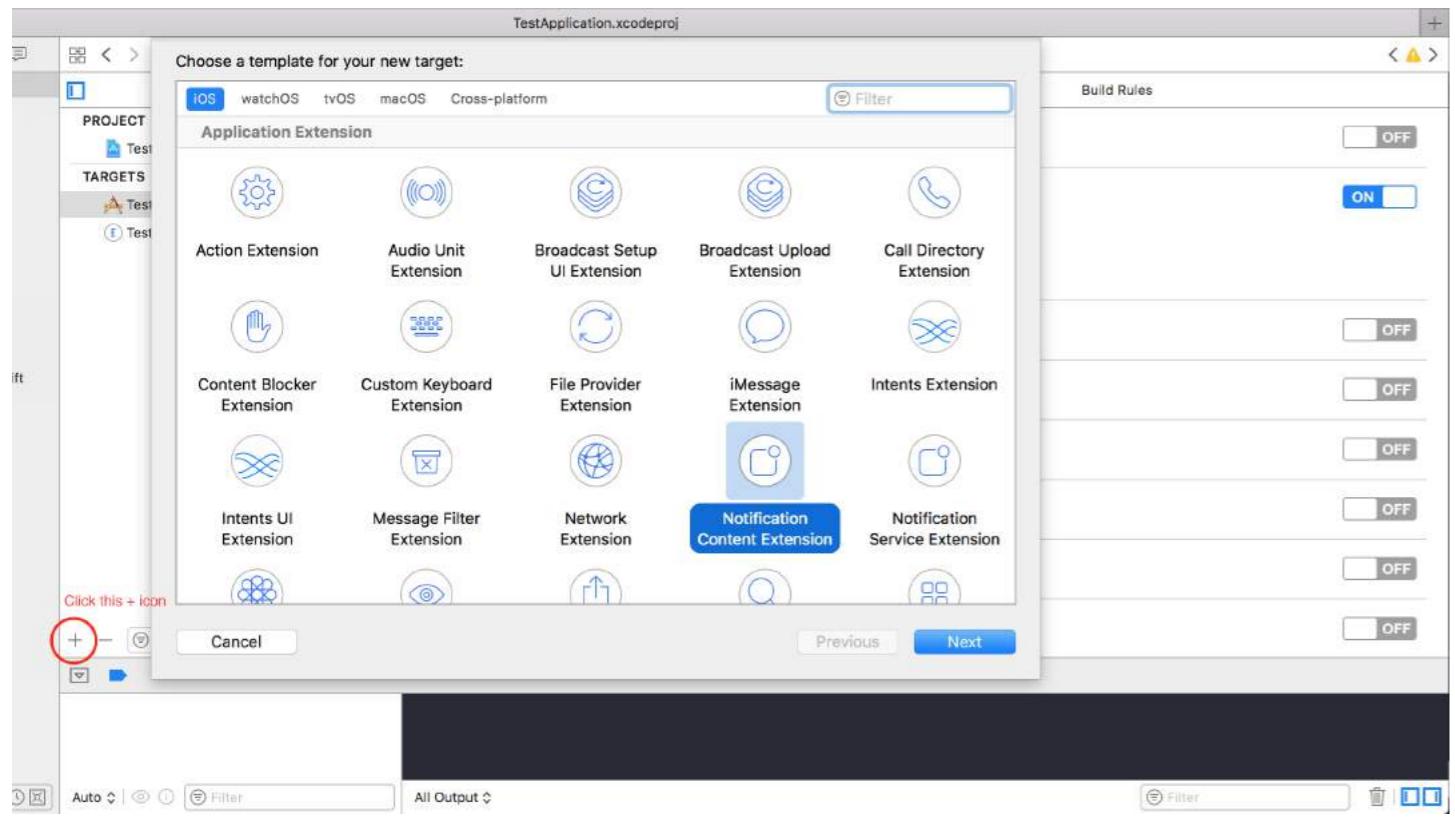
Step 1

Making the environment suitable for Notification. Make sure you enabled **Background Modes** and **Push Notification**

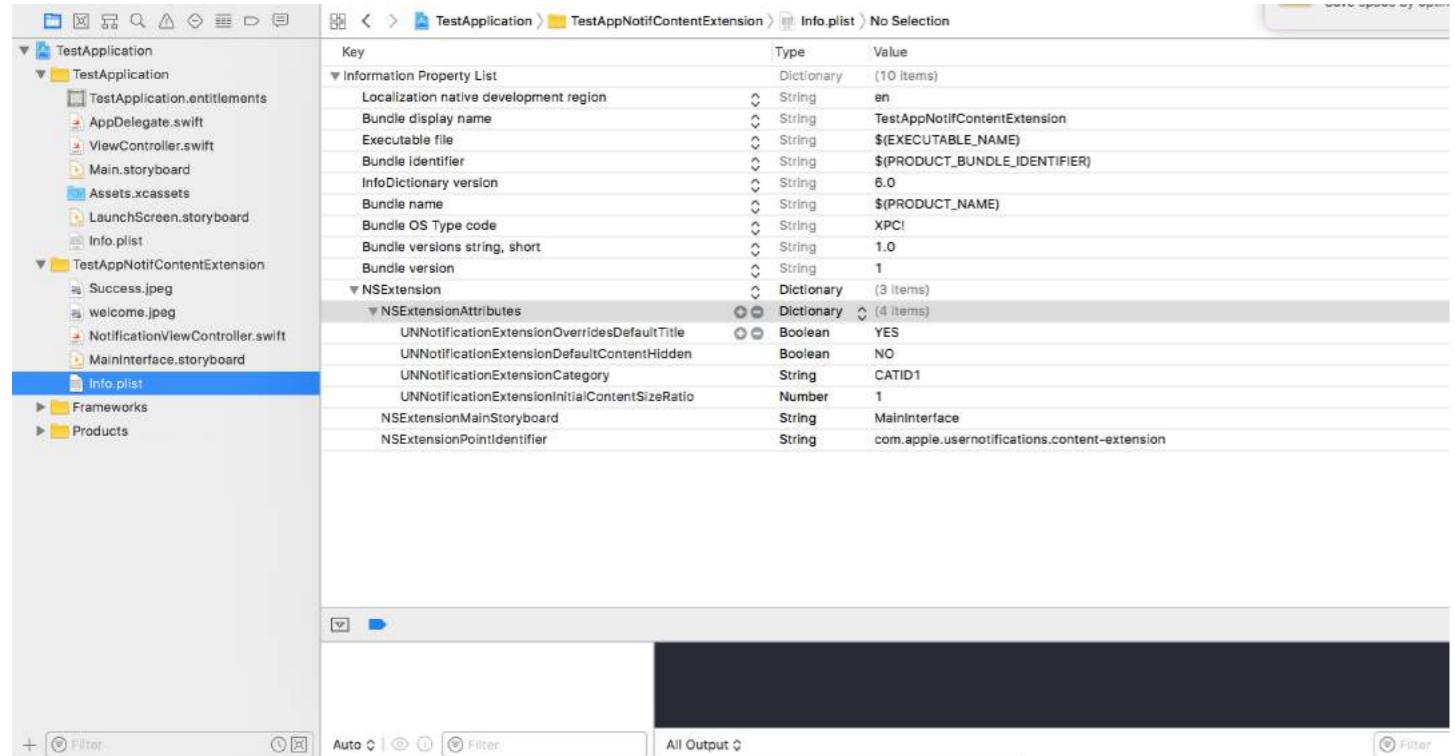


Step 2: Creating an UNNotificationContentExtension

Click on the + icon in the bottom which creates a target template and select Notification Content Extension -> next -> create a name for the content extension -> finish

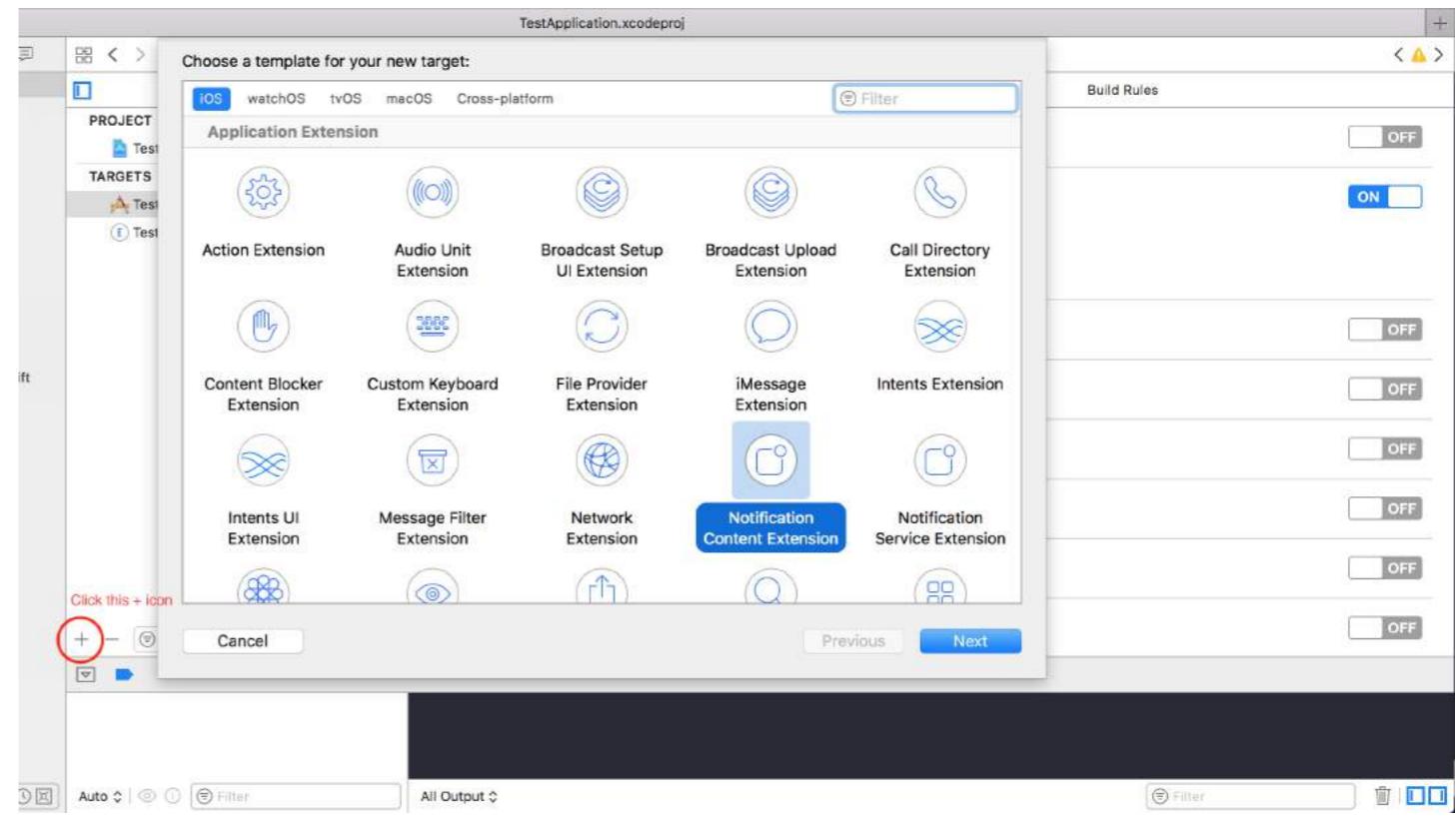


步骤 3：配置已创建扩展的 info.plist 文件

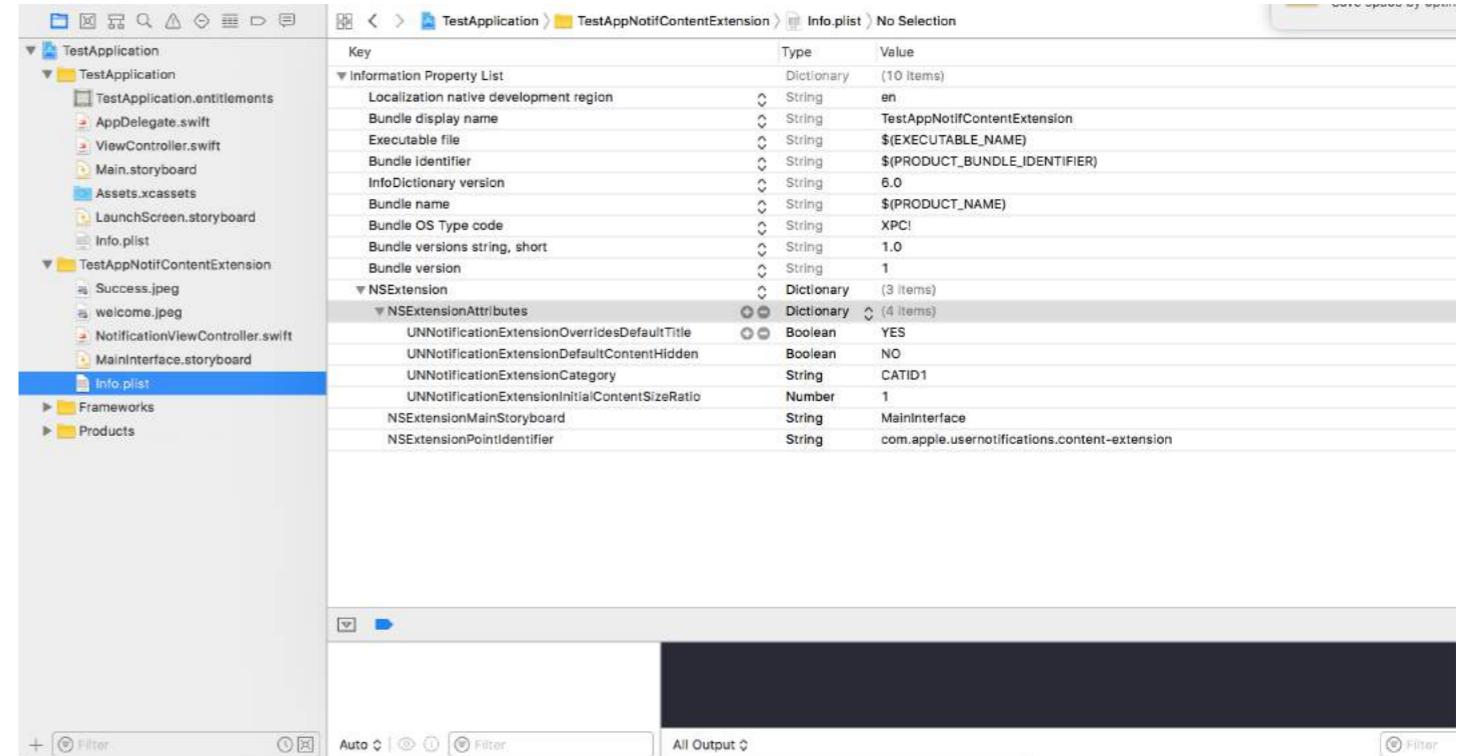


NSExtension 中的字典表示通知内容的显示方式，这些操作在长按收到的通知时执行

- UNNotificationExtensionOverridesDefaultTitle：我们可以为通知自定义标题，默认显示应用程序自身的名称。`title = myTitle`
- UNNotificationDefaultContentHidden：此布尔值决定是否隐藏通知的默认正文
- UNNotificationCategory：类别在应用程序的UNUserNotificationCenter中创建。这里可以是字符串或字符串数组，因此每个类别可以提供不同类型的数据，我们可以



Step 3: Configuring the info.plist file of the created extension



The dictionary in NSExtension signifies how the notification content is displayed, these are performed on long pressing the received notification

- UNNotificationExtensionOverridesDefaultTitle: We can give custom title for our notification by default it displays the name of the application `self.title = myTitle`
- UNNotificationDefaultContentHidden: This boolean determines whether the default body of the notification is to be hidden or not
- UNNotificationCategory: Category is created in UNUserNotificationCenter in your application. Here it can be either a string or an array of strings, so each category can give different types of data from which we can

创建不同的用户界面。我们发送的负载必须包含类别名称，以便显示此特定的扩展

- UNNotificationExtensionInitialContentSizeRatio：初始内容的大小，即首次显示ContentExtension时，相对于设备宽度的初始大小。这里1表示高度将等于宽度

步骤4：在我们的应用中创建UNNotificationAction和UNNotificationCategory

在应用的AppDelegate.swift中的didFinishLaunchingWithOptions函数中添加

```
let userNotificationAction:UNNotificationAction = UNNotificationAction.init(identifier: "ID1",
标题: "வணக்கம்", 选项: .destructive)
let userNotificationAction2:UNNotificationAction = UNNotificationAction.init(identifier: "ID2",
标题: "Success", 选项: .destructive)

let notifCategory:UNNotificationCategory = UNNotificationCategory.init(identifier: "CATID1",
动作: [userNotificationAction,userNotificationAction2], intentIdentifiers: ["ID1", "ID2"] ,
选项:.customDismissAction)

UNUserNotificationCenter.current().delegate = self
UNUserNotificationCenter.current().setNotificationCategories([notifCategory])
UIApplication.shared.registerForRemoteNotifications()
```

我们创建了两个UNNotificationAction，标识符分别为ID1和ID2，并将这些动作添加到一个UNNotificationCategory，标识符为CATID1（ContentExtension的info.plist文件中的categoryID相同，我们这里创建的应在负载和plist文件中使用）。我们将该类别设置到应用的UNUserNotificationCenter中，下一行我们注册通知，这会调用didRegisterForRemoteNotificationsWithDeviceToken函数，在那里我们获取设备令牌

注意：别忘了在AppDelegate.swift中import UserNotifications并添加UNUserNotificationCenterDelegate

步骤5：NotificationContent的示例负载

```
'aps': {
'badge': 0,
>alert': {
'title': "丰富通知",
'body': "丰富通知的正文",
},
'sound' : "default",
'category': "CATID1",
'mutable-content':'1',
},
'attachment': "2"
```

步骤6：配置ContentExtension

当执行通知操作时，类别对应的操作会自动显示。
让我们看看代码是如何实现的

```
import UIKit
import UserNotifications
import UserNotificationsUI

class NotificationViewController: UIViewController, UNNotificationContentExtension {

@IBOutlet var imageView: UIImageView?
```

create different UI's. The payload we send must contain the category name in order to display this particular extension

- UNNotificationExtensionInitialContentSizeRatio: The size of the initial content ie when displaying the ContentExtension for the first time the initial size with respect to width of the device. here 1 denotes the height will be equal to the width

Step 4: Creating UNNotificationAction and UNNotificationCategory in our application

In your app's AppDelegate.swift didFinishLaunchingWithOptions function add

```
let userNotificationAction:UNNotificationAction = UNNotificationAction.init(identifier: "ID1",
title: "வணக்கம்", options: .destructive)
let userNotificationAction2:UNNotificationAction = UNNotificationAction.init(identifier: "ID2",
title: "Success", options: .destructive)

let notifCategory:UNNotificationCategory = UNNotificationCategory.init(identifier: "CATID1",
actions: [userNotificationAction,userNotificationAction2], intentIdentifiers: ["ID1", "ID2"] ,
options:.customDismissAction)

UNUserNotificationCenter.current().delegate = self
UNUserNotificationCenter.current().setNotificationCategories([notifCategory])
UIApplication.shared.registerForRemoteNotifications()
```

We created two UNNotificationAction with identifiers ID1 and ID2 and added those actions to a UNNotificationCategory with identifier CATID1 (the categoryID in ContentExtension's info.plist file are same, what we created here should be used in payload and the plist file). We set the category to our application's UNUserNotificationCenter and in next line we are registering for the notification which calls the didRegisterForRemoteNotificationsWithDeviceToken function where we get the device token

Note: dont forget to import UserNotifications in your AppDelegate.swift and add UNUserNotificationCenterDelegate

Step 5: Sample payload for the NotificationContent

```
'aps': {
'badge': 0,
>alert': {
'title': "Rich Notification",
'body': "Body of RICH NOTIFICATION",
},
'sound' : "default",
'category': "CATID1",
'mutable-content':'1',
},
'attachment': "2"
```

Step 6: Configuring the ContentExtension

The corresponding actions for the category is automatically displayed while the notification action is performed.
Lets us see the code how its being performed

```
import UIKit
import UserNotifications
import UserNotificationsUI

class NotificationViewController: UIViewController, UNNotificationContentExtension {

@IBOutlet var imageView: UIImageView?
```

```

override func viewDidLoad() {
    super.viewDidLoad()
}

func didReceive(_ notification: UNNotification) {
    self.title = "Koushik"
    imageView?.backgroundColor = UIColor.clear
    imageView?.image = #imageLiteral(resourceName: "welcome.jpeg")
}

func didReceive(_ response: UNNotificationResponse, completionHandler completion: @escaping
(UNNotificationContentExtensionResponseOption) -> Void) {

    self.title = "Koushik"
    imageView?.image = UIImage.init(named: "Success.jpeg")

    if(response.actionIdentifier == "ID1")
    {
        imageView?.image = UIImage.init(named: "Success.jpeg")
    }
    else
    {
        imageView?.image = UIImage.init(named: "welcome.jpeg")
    }
}

```

第7步：结果

收到并长按/点击查看通知后，通知显示如下

```

override func viewDidLoad() {
    super.viewDidLoad()
}

func didReceive(_ notification: UNNotification) {
    self.title = "Koushik"
    imageView?.backgroundColor = UIColor.clear
    imageView?.image = #imageLiteral(resourceName: "welcome.jpeg")
}

func didReceive(_ response: UNNotificationResponse, completionHandler completion: @escaping
(UNNotificationContentExtensionResponseOption) -> Void) {

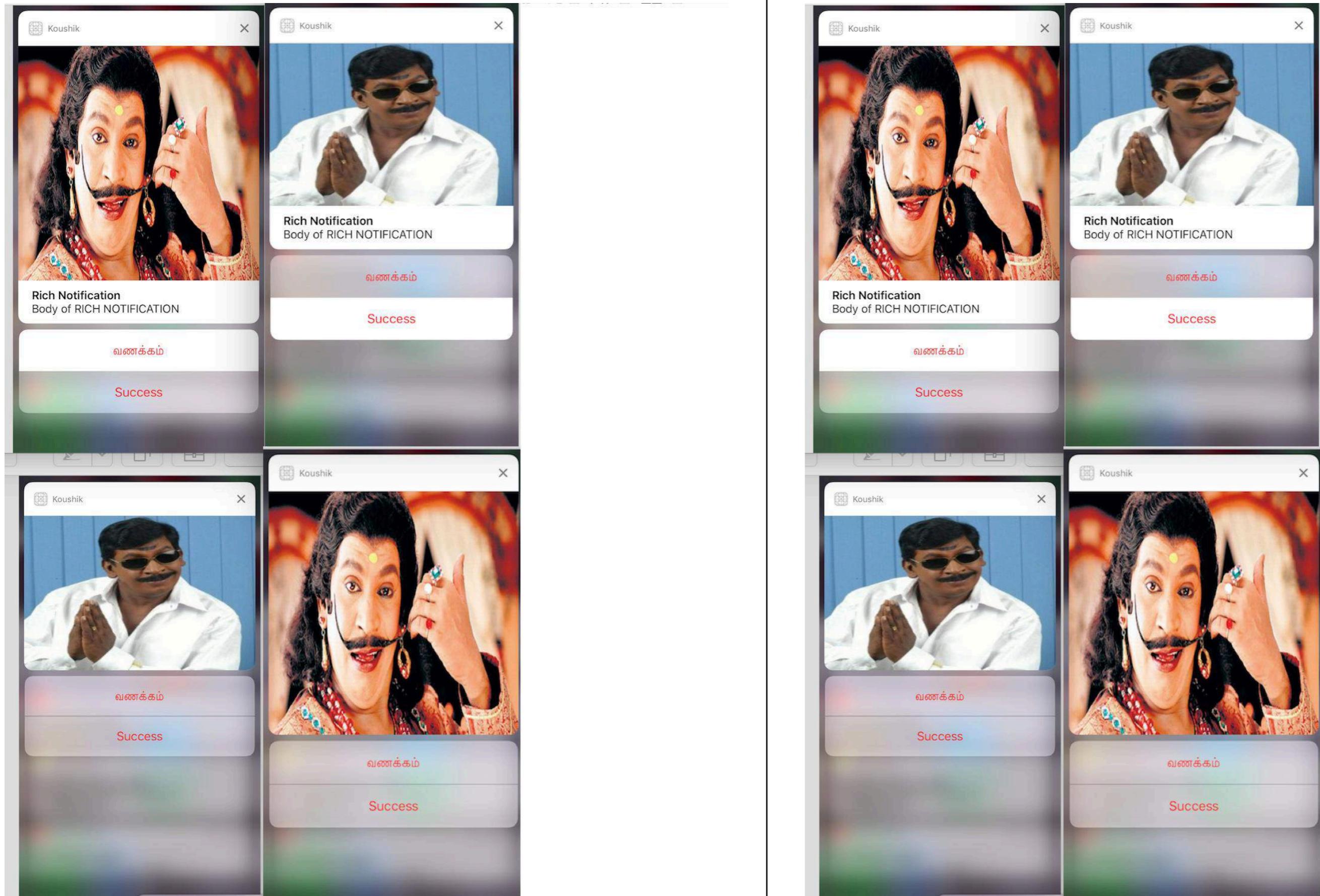
    self.title = "Koushik"
    imageView?.image = UIImage.init(named: "Success.jpeg")

    if(response.actionIdentifier == "ID1")
    {
        imageView?.image = UIImage.init(named: "Success.jpeg")
    }
    else
    {
        imageView?.image = UIImage.init(named: "welcome.jpeg")
    }
}

```

Step 7: Result

After receiving and long press/Clicking View notification, the notification looks like this



标题是“Koushik”，因为我们设置了 `self.title = "Koushik"` 并且 `UNNotificationExtensionOverrideDefaultTitle` 为 YES。在第3步中我们将 `UNNotificationExtensionDefaultContentHidden` 设置为 NO，如果是 YES，通知将显示为图3和图4的样子。

The title is "Koushik" since we gave `self.title = "Koushik"` and `UNNotificationExtensionOverrideDefaultTitle` as YES. In step 3 we gave `UNNotificationExtensionDefaultContentHidden` as NO if its YES then the notification will look like images 3 and 4.

第110章：键值编码-键值观察

KVC :- 键值编码

通常实例变量通过属性或访问器访问，但KVC提供了另一种通过字符串访问变量的方式。这样你的类就像一个字典，属性名例如“age”变成了键，该属性持有的值变成了该键对应的值。

例如，你有一个员工类，带有“age”属性。通常我们这样访问。

```
emp.age = @"20";
NSString age = emp.age;
```

但KVC的工作方式如下：

```
[emp valueForKey:@"age"];
[emp setValue:@"25" forKey:@"age"];
```

KVO : 键值观察者

当对象的某个属性发生变化时，通知该变化的机制称为KVO。

例如：键盘通知

例如，person对象希望在BankAccount对象的accountBalance属性发生变化时收到通知。为此，Person对象必须通过发送addObserver: forKeyPath: options: context:消息，注册为BankAccount对象accountBalance属性的观察者。

第110.1节：观察NSObject子类的属性

大多数KVO和KVC功能默认已在所有NSObject子类中实现。

要开始观察名为firstName的属性，属于名为personObject的对象，在观察类中执行以下操作：

```
[personObject addObserver:self
    forKeyPath:@"firstName"
    options:NSKeyValueObservingOptionNew
    context:nil];
```

上述代码中self所指的对象将在被观察的键路径发生变化时接收到observeValueForKeyPath:ofObject:change:context:消息。

```
- (void)observeValueForKeyPath:(NSString *)keyPath
    ofObject:(id)object
    change:(NSDictionary<NSString *, id> *)change
    context:(void *)context
{
    NSLog(@"new value of %@ is: %@", keyPath, change[NSKeyValueChangeNewKey]);
}
```

“键路径”是KVC的术语。NSObject子类默认实现KVC功能。

名为(firstName的实例变量可以通过@"firstName"键路径访问。

访问@"firstName"键路径时，会调用名为firstName的getter方法，无论是否存在

Chapter 110: Key Value Coding-Key Value Observation

KVC :- Key-Value Coding

Normally instance variables are accessed through properties or accessors but KVC gives another way to access variables in form of strings. In this way your class acts like a dictionary and your property name for example "age" becomes key and value that property holds becomes value for that key.

```
For example, you have employee class with "age" property. Normally we access like this.
emp.age = @"20";
NSString age = emp.age;
```

But KVC works like this:

```
[emp valueForKey:@"age"];
[emp setValue:@"25" forKey:@"age"];
```

KVO :- Key-Value Observer

The mechanism through which objects are notified when there is change in any of property is called KVO.

Ex.: keyboard notification

For example, person object is interested in getting notification when accountBalance property is changed in BankAccount object. To achieve this, Person Object must register as an observer of the BankAccount's accountBalance property by sending an addObserver: forKeyPath: options: context: message.

Section 110.1: Observing a property of a NSObject subclass

Most KVO and KVC functionality is already implemented by default on all **NSObject** subclasses.

To start observing a property named firstName of an object named personObject do this in the observing class:

```
[personObject addObserver:self
    forKeyPath:@"firstName"
    options:NSKeyValueObservingOptionNew
    context:nil];
```

The object that self in the above code refers to will then receive a observeValueForKeyPath:ofObject:change:context: message whenever the observed key path changes.

```
- (void)observeValueForKeyPath:(NSString *)keyPath
    ofObject:(id)object
    change:(NSDictionary<NSString *, id> *)change
    context:(void *)context
{
    NSLog(@"new value of %@ is: %@", keyPath, change[NSKeyValueChangeNewKey]);
}
```

"Key path" is a KVC term. **NSObject** subclasses implement KVC functionality by default.

An instance variable named _firstName will be accessible by the @"firstName" key path.

A getter method named firstName will be called when accessing the @"firstName" key path, regardless of there

_firstName实例变量或setFirstName方法。

第110.2节：KVO观察中上下文的使用

```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary<NSString *, id> *)change context:(void *)context
```

如果你将类交给他人使用，Context（上下文）非常重要。Context 让你的类的观察者能够验证调用的是你的观察者。

不传递观察者的问题是，如果有人继承你的类并为同一个对象、同一个键注册观察者，但他没有传递上下文，那么超类的观察者可能会被多次调用。

一个唯一且供你内部使用的变量是一个很好的上下文。

更多信息请参见。

[重要性和良好的上下文](#)

being a _firstName instance variable or setFirstName setter method.

Section 110.2: Use of context for KVO Observation

```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary<NSString *, id> *)change context:(void *)context
```

Context is important if you ship your class for others to use. Context lets your class observer verify that its your observer which is being called.

The problem with not passing an observer is, if some one subclass your class and register an observer for the same object,same key and he does not passes a context ,then the super class observer can be called multiple time.

A variable which is unique and internal for your use is a good context.

For more information.

[importance and good context](#)

第111章：初始化惯用法

第111.1节：带块的工厂方法

```
internal func Init<Type>(value : Type, block: @noescape (object: Type) -> Void) -> Type
{
    block(object: value)
    return value
}
```

用法：

```
Init(UILabel(frame: CGRect.zero)) {
    $0.backgroundColor = UIColor.blackColor()
}
```

第111.2节：设置元组以避免代码重复

通过一行代码设置变量元组，避免构造函数中的代码重复：

```
class 联系人: UIView
{
    private var message: UILabel
    private var phone: UITextView

    required init?(coder aDecoder: NSCoder) {
        (message, phone) = self.dynamicType.setUp()
        super.init(coder: aDecoder)
    }

    override func awakeFromNib() {
        (message, phone) = self.dynamicType.setUp()
        super.awakeFromNib()
    }

    override init(frame: CGRect) {
        (message, phone) = self.dynamicType.setUp()
        super.init(frame: frame)
    }

    private static func setUp(){
        let message = UILabel() // ...
        let phone = UITextView() // ...
        return (message, phone)
    }
}
```

第111.3节：使用位置常量初始化

```
let mySwitch: UISwitch = {
    view.addSubview($0)
    $0.addTarget(self, action: "action", forControlEvents: .TouchUpInside)
    return $0
}(UISwitch())
```

Chapter 111: Initialization idioms

Section 111.1: Factory method with block

```
internal func Init<Type>(value : Type, block: @noescape (object: Type) -> Void) -> Type
{
    block(object: value)
    return value
}
```

Usage:

```
Init(UILabel(frame: CGRect.zero)) {
    $0.backgroundColor = UIColor.blackColor()
}
```

Section 111.2: Set to tuples to avoid code repetition

Avoid code repetition in constructors by setting a tuple of variables with a one liner:

```
class Contact: UIView
{
    private var message: UILabel
    private var phone: UITextView

    required init?(coder aDecoder: NSCoder) {
        (message, phone) = self.dynamicType.setUp()
        super.init(coder: aDecoder)
    }

    override func awakeFromNib() {
        (message, phone) = self.dynamicType.setUp()
        super.awakeFromNib()
    }

    override init(frame: CGRect) {
        (message, phone) = self.dynamicType.setUp()
        super.init(frame: frame)
    }

    private static func setUp(){
        let message = UILabel() // ...
        let phone = UITextView() // ...
        return (message, phone)
    }
}
```

Section 111.3: Initialize with positional constants

```
let mySwitch: UISwitch = {
    view.addSubview($0)
    $0.addTarget(self, action: "action", forControlEvents: .TouchUpInside)
    return $0
}(UISwitch())
```

第111.4节：在didSet中初始化属性

```
@IBOutlet weak var title: UILabel! {
    didSet {
        label.textColor = UIColor.redColor()
        label.font = UIFont.systemFontOfSize(20)
        label.backgroundColor = UIColor.blueColor()
    }
}
```

也可以同时设置值并初始化：

```
private var loginButton = UIButton() {
    didSet(oldValue) {
        loginButton.addTarget(self, action: #selector(LoginController.didClickLogin),
        forControlEventss: .TouchUpInside)
    }
}
```

第111.5节：在自定义NSObject中分组outlet

将每个outlet移动到一个NSObject中。然后从库中拖动一个对象到故事板的控制器场景中，并在那里连接元素。

```
class ContactFormStyle: NSObject
{
    @IBOutlet private weak var message: UILabel!
    didSet {
        message.font = UIFont.systemFontOfSize(12)
        message.textColor = UIColor.blackColor()
    }
}

class ContactFormVC: UIViewController
{
    @IBOutlet private var style: ContactFormStyle!
}
```

第111.6节：使用then初始化

这在语法上类似于使用位置常量初始化的示例，但需要Then扩展，来自<https://github.com/devxoul/Then>（附在下方）。

```
let label = UILabel().then {
    $0.textAlignment = .Center
    $0.textColor = UIColor.blackColor()
    $0.text = "Hello, World!"
}
```

Then 扩展：

```
import Foundation

public protocol Then {}

extension Then
```

Section 111.4: Initialize attributes in didSet

```
@IBOutlet weak var title: UILabel! {
    didSet {
        label.textColor = UIColor.redColor()
        label.font = UIFont.systemFontOfSize(20)
        label.backgroundColor = UIColor.blueColor()
    }
}
```

It's also possible to both set a value and initialize it:

```
private var loginButton = UIButton() {
    didSet(oldValue) {
        loginButton.addTarget(self, action: #selector(LoginController.didClickLogin),
        forControlEventss: .TouchUpInside)
    }
}
```

Section 111.5: Group outlets in a custom NSObject

Move every outlet to an NSObject. Then drag an Object from the library to the controller scene of the storyboard and hook the elements there.

```
class ContactFormStyle: NSObject
{
    @IBOutlet private weak var message: UILabel!
    didSet {
        message.font = UIFont.systemFontOfSize(12)
        message.textColor = UIColor.blackColor()
    }
}

class ContactFormVC: UIViewController
{
    @IBOutlet private var style: ContactFormStyle!
}
```

Section 111.6: Initialize with then

This is similar in syntax to the example that initializes using positional constants, but requires the Then extension from <https://github.com/devxoul/Then> (attached below).

```
let label = UILabel().then {
    $0.textAlignment = .Center
    $0.textColor = UIColor.blackColor()
    $0.text = "Hello, World!"
}
```

The Then extension:

```
import Foundation

public protocol Then {}

extension Then
```

```
{  
    public func then(@noescape block: inout Self -> Void) -> Self {  
        var copy = self  
        block(&copy)  
        return copy  
    }  
}  
  
extension NSObject: Then {}
```

```
{  
    public func then(@noescape block: inout Self -> Void) -> Self {  
        var copy = self  
        block(&copy)  
        return copy  
    }  
}  
  
extension NSObject: Then {}
```

第112章：Storyboard

通常，Storyboard中的视图控制器会根据Storyboard本身定义的操作自动实例化和创建。然而，你也可以使用Storyboard对象来实例化Storyboard文件中的初始视图控制器，或者实例化你想要以编程方式呈现的其他视图控制器。下面你将看到这两种用例的示例。

第112.1节：初始化

```
//Swift  
let storyboard = UIStoryboard(name: "Main", bundle: NSBundle.mainBundle())  
  
//Objective-c  
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:[NSBundle mainBundle]];
```

第112.2节：获取初始视图控制器

```
//Swift  
let initialScreen = storyboard.instantiateInitialViewController()  
  
//Objective-c  
UIViewController *initialScreen = [storyboard instantiateInitialViewController];
```

第112.3节：获取视图控制器

```
//Swift  
let viewController = storyboard.instantiateViewControllerWithIdentifier("identifier")  
  
//Objective-c  
UIViewController *viewController = [storyboard  
instantiateViewControllerWithIdentifier:@"identifier"];
```

Chapter 112: Storyboard

Normally, view controllers in a storyboard are instantiated and created automatically in response to actions defined within the storyboard itself. However, you can use a storyboard object to instantiate the initial view controller in a storyboard file or instantiate other view controllers that you want to present programmatically. Below you will find examples of both use cases.

Section 112.1: Initialize

```
//Swift  
let storyboard = UIStoryboard(name: "Main", bundle: NSBundle.mainBundle())  
  
//Objective-c  
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:[NSBundle mainBundle]];
```

Section 112.2: Fetch Initial ViewController

```
//Swift  
let initialScreen = storyboard.instantiateInitialViewController()  
  
//Objective-c  
UIViewController *initialScreen = [storyboard instantiateInitialViewController];
```

Section 112.3: Fetch ViewController

```
//Swift  
let viewController = storyboard.instantiateViewControllerWithIdentifier("identifier")  
  
//Objective-c  
UIViewController *viewController = [storyboard  
instantiateViewControllerWithIdentifier:@"identifier"];
```

第113章：后台模式和事件

第113.1节：后台播放音频

在属性列表 (.plist) 文件中添加一个名为Required background modes的键。

如下图所示。

Key	Type	Value
▼ Information Property List	Dictionary	(16 items)
Localization native development r...	String	en
Bundle display name	String	[REDACTED]
Executable file	String	\$(EXECUTABLE_NAME)
► Icon files	Array	(14 items)
Bundle identifier	String	[REDACTED]
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.1
Bundle creator OS Type code	String	????
Bundle version	String	1.1
Application requires iPhone envir...	Boolean	YES
▼ Required background modes	Array	(1 item)
Item 0	String	App plays audio or streams audio/video using AirPlay
Icon already includes gloss effects	Boolean	YES
► Required device capabilities	Array	(1 item)
► Supported interface orientations	Array	(1 item)

Chapter 113: Background Modes and Events

Section 113.1: Play Audio in Background

Add a key named **Required background modes** in property list (.plist) file ..

as following picture..

Key	Type	Value
▼ Information Property List	Dictionary	(16 items)
Localization native development r...	String	en
Bundle display name	String	[REDACTED]
Executable file	String	\$(EXECUTABLE_NAME)
► Icon files	Array	(14 items)
Bundle identifier	String	[REDACTED]
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.1
Bundle creator OS Type code	String	????
Bundle version	String	1.1
Application requires iPhone envir...	Boolean	YES
▼ Required background modes	Array	(1 item)
Item 0	String	App plays audio or streams audio/video using AirPlay
Icon already includes gloss effects	Boolean	YES
► Required device capabilities	Array	(1 item)
► Supported interface orientations	Array	(1 item)

并在以下文件中添加代码

AppDelegate.h

```
#import <AVFoundation/AVFoundation.h>
#import <AudioToolbox/AudioToolbox.h>
```

AppDelegate.m

在 application didFinishLaunchingWithOptions 中

```
[[AVAudioSession sharedInstance] setDelegate:self];
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryPlayback error:nil];
[[AVAudioSession sharedInstance] setActive:YES error:nil];
[[UIApplication sharedApplication] beginReceivingRemoteControlEvents];

UInt32 size = sizeof(CFStringRef);
CFStringRef route;
AudioSessionGetProperty(kAudioSessionProperty_AudioRoute, &size, &route);
NSLog(@"route = %@", route);
```

如果你想根据事件进行更改，需要在 AppDelegate.m 中添加以下代码

```
- (void)remoteControlReceivedWithEvent:(UIEvent *)theEvent {
```

And add following code in

AppDelegate.h

```
#import <AVFoundation/AVFoundation.h>
#import <AudioToolbox/AudioToolbox.h>
```

AppDelegate.m

in application didFinishLaunchingWithOptions

```
[[AVAudioSession sharedInstance] setDelegate:self];
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryPlayback error:nil];
[[AVAudioSession sharedInstance] setActive:YES error:nil];
[[UIApplication sharedApplication] beginReceivingRemoteControlEvents];

UInt32 size = sizeof(CFStringRef);
CFStringRef route;
AudioSessionGetProperty(kAudioSessionProperty_AudioRoute, &size, &route);
NSLog(@"route = %@", route);
```

If you want changes as per events you have to add following code in AppDelegate.m

```
- (void)remoteControlReceivedWithEvent:(UIEvent *)theEvent {
```

```

if (theEvent.type == UIEventTypeRemoteControl) {
    switch(theEvent.subtype) {
        case UIEventSubtypeRemoteControlPlay:
            [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
            break;
        case UIEventSubtypeRemoteControlPause:
            [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
            break;
        case UIEventSubtypeRemoteControlStop:
            break;
        case UIEventSubtypeRemoteControlTogglePlayPause:
            [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
            break;
        default:
            return;
    }
}

```

需要根据通知进行处理。

```

if (theEvent.type == UIEventTypeRemoteControl) {
    switch(theEvent.subtype) {
        case UIEventSubtypeRemoteControlPlay:
            [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
            break;
        case UIEventSubtypeRemoteControlPause:
            [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
            break;
        case UIEventSubtypeRemoteControlStop:
            break;
        case UIEventSubtypeRemoteControlTogglePlayPause:
            [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
            break;
        default:
            return;
    }
}

```

Based on notification have to work on it..

第114章：Fastlane

第114.1节：fastlane工具

fastlane 是一个面向Android和iOS开发者的开源构建自动化工具。它可以减少你的构建生成时间。它是一个使用Ruby的命令行工具，因此你的电脑需要安装Ruby。大多数Mac电脑默认已经安装了Ruby。

安装fastlane

1. 打开终端。
2. 运行 `sudo gem install fastlane --verbose`
3. 如果您还没有安装 Xcode 命令行工具，请运行 `xcode-select --install` 来安装它们
4. 现在，`cd` 到你的项目文件夹（输入 `cd [末尾带空格]` 并将你的项目文件夹拖入终端）
5. 运行 `fastlane init` 来设置 fastlane。
6. 现在你可以使用所有 Fastlane 工具了：

iOS 工具

- [deliver](#)：上传截图、元数据和你的应用到 App Store
- [snapshot](#)：自动化在每个设备上为你的 iOS 应用拍摄本地化截图
- [frameit](#)：快速将截图放入正确的设备框架中
- [pem](#)：自动生成和更新推送通知配置文件
- [sigh](#)：因为你更愿意花时间开发而不是处理配置问题
- [produce](#)：使用命令行在 iTunes Connect 和开发者门户创建新的 iOS 应用
- [cert](#)：自动创建和维护 iOS 代码签名证书
- [gym](#)：构建你的 iOS 应用从未如此简单
- [match](#)：使用 Git 轻松同步团队的证书和配置文件
- [scan](#)：运行 iOS 和 Mac 应用测试的最简单方法
- [spaceship](#)：访问 Apple 开发者中心和 iTunes Connect 的 Ruby 库

iOS TestFlight 工具

- [pilot](#)：从终端管理您的TestFlight测试人员和构建的最佳方式
- [boarding](#)：邀请您的TestFlight测试版测试人员的最简单方法

Android工具

- [supply](#)：上传您的Android应用及其元数据到Google Play
- [screengrab](#)：自动在每个设备上为您的Android应用拍摄本地化截图

Chapter 114: Fastlane

Section 114.1: fastlane tools

[fastlane](#) is an open source build automation tool for Android and iOS for developers. It reduce your build generation time. It is a command line tool that uses [Ruby](#), so you need Ruby on your computer. Most Macs already have Ruby installed by default.

Install fastlane

1. Open a terminal.
2. Run `sudo gem install fastlane --verbose`
3. If you haven't installed the Xcode command-line tools yet, run `xcode-select --install` to install them
4. Now, `cd` into your project folder (type `cd` [with the space at the end] and drag your project folder into the terminal)
5. Run `fastlane init` to get fastlane setup.
6. Now you can able to use all the Fastlane tools:

iOS Tools

- [deliver](#): Upload screenshots, metadata, and your app to the App Store
- [snapshot](#): Automate taking localized screenshots of your iOS app on every device
- [frameit](#): Quickly put your screenshots into the right device frames
- [pem](#): Automatically generate and renew your push notification profiles
- [sigh](#): Because you would rather spend your time building stuff than fighting provisioning
- [produce](#): Create new iOS apps on iTunes Connect and Dev Portal using the command line
- [cert](#): Automatically create and maintain iOS code signing certificates
- [gym](#): Building your iOS apps has never been easier
- [match](#): Easily sync your certificates and profiles across your team using Git
- [scan](#): The easiest way to run tests for your iOS and Mac apps
- [spaceship](#): Ruby library to access the Apple Dev Center and iTunes Connect

iOS TestFlight Tools

- [pilot](#): The best way to manage your TestFlight testers and builds from your terminal
- [boarding](#): The easiest way to invite your TestFlight beta testers

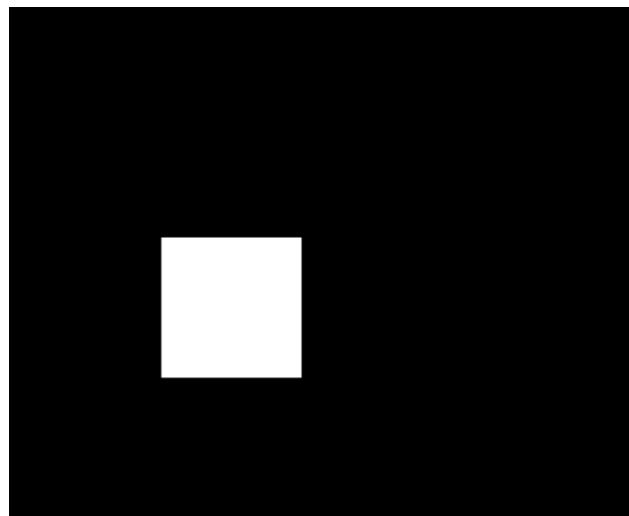
Android Tools

- [supply](#): Upload your Android app and its metadata to Google Play
- [screengrab](#): Automate taking localized screenshots of your Android app on every device

第115章：CAShapeLayer

第115.1节：绘制矩形

```
CAShapeLayer *mask = [[CAShapeLayer alloc] init];  
  
mask.frame = CGRectMake(50, 50, 100, 100);  
  
CGFloat 宽度 = 100;  
  
CGFloat 高度 = 100;  
  
CGMutablePathRef path = CGPathCreateMutable();  
  
CGPathMoveToPoint(path, nil, 30, 30);  
  
CGPathAddLineToPoint(path, nil, 宽度, 30);  
  
CGPathAddLineToPoint(path, nil, 宽度, 高度);  
  
CGPathAddLineToPoint(path, nil, 30, 高度);  
  
CGPathAddLineToPoint(path, nil, 30, 30);  
  
CGPathCloseSubpath(path);  
  
mask.path = path;  
  
CGPathRelease(path);  
  
self.view.layer.mask = mask;
```



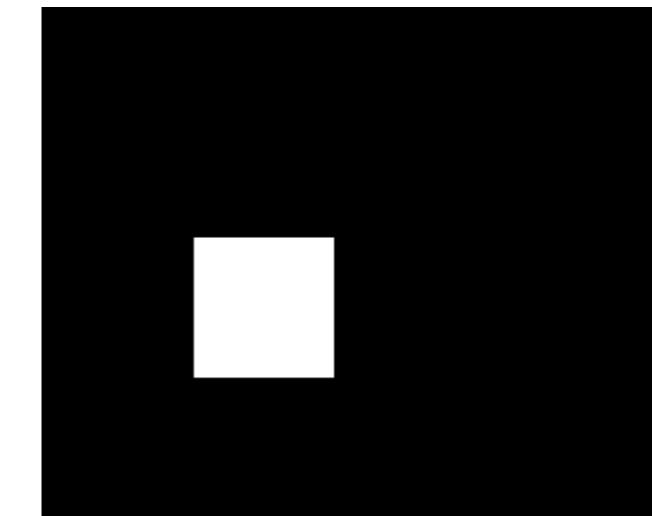
第115.2节：绘制圆形

```
CAShapeLayer *circle = [CAShapeLayer layer];  
  
[circle setPath:[[UIBezierPath bezierPathWithOvalInRect:CGRectMake(100, 100, 150, 150)]  
CGPath]];  
  
[circle setStrokeColor:[[UIColor blueColor] CGColor]];
```

Chapter 115: CAShapeLayer

Section 115.1: Draw Rectangle

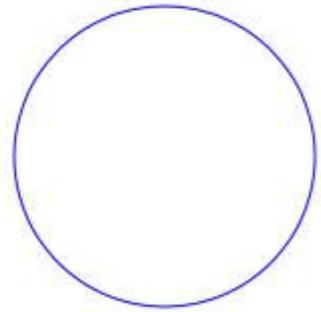
```
CAShapeLayer *mask = [[CAShapeLayer alloc] init];  
  
mask.frame = CGRectMake(50, 50, 100, 100);  
  
CGFloat width = 100;  
  
CGFloat height = 100;  
  
CGMutablePathRef path = CGPathCreateMutable();  
  
CGPathMoveToPoint(path, nil, 30, 30);  
  
CGPathAddLineToPoint(path, nil, width, 30);  
  
CGPathAddLineToPoint(path, nil, width, height);  
  
CGPathAddLineToPoint(path, nil, 30, height);  
  
CGPathAddLineToPoint(path, nil, 30, 30);  
  
CGPathCloseSubpath(path);  
  
mask.path = path;  
  
CGPathRelease(path);  
  
self.view.layer.mask = mask;
```



Section 115.2: Draw Circle

```
CAShapeLayer *circle = [CAShapeLayer layer];  
  
[circle setPath:[[UIBezierPath bezierPathWithOvalInRect:CGRectMake(100, 100, 150, 150)]  
CGPath]];  
  
[circle setStrokeColor:[[UIColor blueColor] CGColor]];
```

```
[circle setFillColor:[[UIColor clearColor] CGColor]];
[[self.view layer] addSublayer:circle];
```



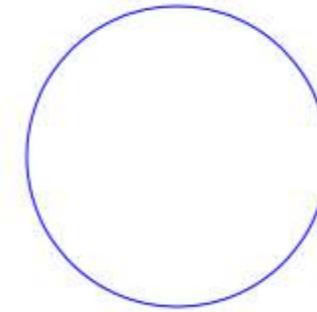
第115.3节：CAShapeLayer动画

```
CAShapeLayer *circle = [CAShapeLayer layer];
[circle setPath:[[UIBezierPath bezierPathWithOvalInRect:CGRectMake(100, 100, 150, 150)] CGPath]];
[circle setStrokeColor:[[UIColor blueColor] CGColor]];
[circle setFillColor:[[UIColor clearColor] CGColor]];
[[self.view layer] addSublayer:circle];

CABasicAnimation *pathAnimation = [CABasicAnimation animationWithKeyPath:@"strokeEnd"];
pathAnimation.duration = 1.5f;
pathAnimation.fromValue = [NSNumber numberWithFloat:0.0f];
pathAnimation.toValue = [NSNumber numberWithFloat:1.0f];
pathAnimation.repeatCount = 10;
pathAnimation.autoreverses = YES;
[circle addAnimation:pathAnimation
    forKey:@"strokeEnd"];
```

)

```
[circle setFillColor:[[UIColor clearColor] CGColor]];
[[self.view layer] addSublayer:circle];
```



Section 115.3: CAShapeLayer Animation

```
CAShapeLayer *circle = [CAShapeLayer layer];
[circle setPath:[[UIBezierPath bezierPathWithOvalInRect:CGRectMake(100, 100, 150, 150)] CGPath]];
[circle setStrokeColor:[[UIColor blueColor] CGColor]];
[circle setFillColor:[[UIColor clearColor] CGColor]];
[[self.view layer] addSublayer:circle];

CABasicAnimation *pathAnimation = [CABasicAnimation animationWithKeyPath:@"strokeEnd"];
pathAnimation.duration = 1.5f;
pathAnimation.fromValue = [NSNumber numberWithFloat:0.0f];
pathAnimation.toValue = [NSNumber numberWithFloat:1.0f];
pathAnimation.repeatCount = 10;
pathAnimation.autoreverses = YES;
[circle addAnimation:pathAnimation
    forKey:@"strokeEnd"];
```

)

第115.4节：基本CAShapeLayer操作

使用UIBezierPath创建圆形路径ShapeLayer

```
CAShapeLayer *circleLayer = [CAShapeLayer layer];
[circleLayer setPath:[[UIBezierPath bezierPathWithOvalInRect:
CGRectMake(50, 50, 100, 100)] CGPath]];
circleLayer.lineWidth = 2.0;
[circleLayer setStrokeColor:[[UIColor redColor] CGColor]];
[circleLayer setFillColor:[[UIColor clearColor] CGColor]];
circleLayer.lineJoin = kCALineJoinRound; //4种类型可用于创建线条样式
circleLayer.lineDashPattern = [NSArray arrayWithObjects:
[NSNumber numberWithInt:2],[NSNumber numberWithInt:3 ], nil];
// self.origImage 是父视图
[[self.view layer] addSublayer:circleLayer];
self.currentShapeLayer = circleLayer; // 用于保存该形状图层引用的公共值

self.view.layer.borderWidth = 1.0f;
self.view.layer.borderColor = [[UIColor blueColor]CGColor]; // 这将在主视图中绘制
```

移除 ShapeLayer

保持对该形状图层的引用。例如，你可能有一个属性 currentShapeLayer：现在你有了引用，就可以轻松移除该图层：

类型 1：

```
[self.currentShapeLayer removeFromSuperlayer];
```

类型 2：

```
self.view.layer.sublayers = nil ; // 移除所有之前的形状
```

其他操作

//绘制正方形形状

```
CAShapeLayer *squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(20, 20, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = nil;
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];
```

//绘制圆形形状

```
CAShapeLayer *circleShape = [CAShapeLayer layer];
circleShape.frame = CGRectMake(160, 20, 120, 120);
circleShape.lineWidth = 2.0;
circleShape.fillColor = nil;
circleShape.strokeColor = [[UIColor redColor] CGColor];
circleShape.path = [UIBezierPath bezierPathWithOvalInRect:circleShape.bounds].CGPath;
[[self.view layer] addSublayer:circleShape];
```

//子路径

//UIBezierPath 可以有任意数量的“路径段”（或子路径），因此你可以有效地绘制为

Section 115.4: Basic CAShapeLayer Operation

UIBezierPath using to create a circular path ShapeLayer

```
CAShapeLayer *circleLayer = [CAShapeLayer layer];
[circleLayer setPath:[[UIBezierPath bezierPathWithOvalInRect:
CGRectMake(50, 50, 100, 100)] CGPath]];
circleLayer.lineWidth = 2.0;
[circleLayer setStrokeColor:[[UIColor redColor] CGColor]];
[circleLayer setFillColor:[[UIColor clearColor] CGColor]];
circleLayer.lineJoin = kCALineJoinRound; //4 types are available to create a line style
circleLayer.lineDashPattern = [NSArray arrayWithObjects:
[NSNumber numberWithInt:2],[NSNumber numberWithInt:3 ], nil];
// self.origImage is parentView
[[self.view layer] addSublayer:circleLayer];
self.currentShapeLayer = circleLayer; // public value using to keep that reference of the shape
Layer
self.view.layer.borderWidth = 1.0f;
self.view.layer.borderColor = [[UIColor blueColor]CGColor]; // that will plotted in the mainview
```

Remove ShapeLayer

Keep a reference to that shape layer. For example, you might have a property currentShapeLayer: Now that you have a reference, you can easily remove the layer:

Type 1:

```
[self.currentShapeLayer removeFromSuperlayer];
```

Type 2:

```
self.view.layer.sublayers = nil ; //removed all earlier shapes
```

Other Operation

//Draw Square Shape

```
CAShapeLayer *squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(20, 20, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = nil;
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];
```

//Draw Circle Shape

```
CAShapeLayer *circleShape = [CAShapeLayer layer];
circleShape.frame = CGRectMake(160, 20, 120, 120);
circleShape.lineWidth = 2.0;
circleShape.fillColor = nil;
circleShape.strokeColor = [[UIColor redColor] CGColor];
circleShape.path = [UIBezierPath bezierPathWithOvalInRect:circleShape.bounds].CGPath;
[[self.view layer] addSublayer:circleShape];
```

//Subpaths

//UIBezierPath can have any number of “path segments” (or subpaths) so you can effectively draw as

在单个路径对象中绘制任意多的形状或线条

```
CAShapeLayer *shapeLayer = [CAShapeLayer layer];
shapeLayer.frame = CGRectMake(20, 140, 200, 200);
shapeLayer.lineWidth = 2.0;
shapeLayer.fillColor = nil;
shapeLayer.strokeColor = [[UIColor redColor] CGColor];

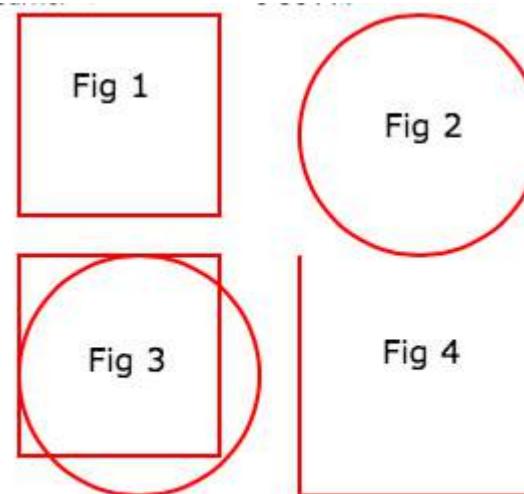
CGMutablePathRef combinedPath= CGPathCreateMutableCopy(circleShape.path);
CGPathAddPath(combinedPath, NULL, squareLayer.path);

shapeLayer.path = combinedPath;
[[self.view layer] addSublayer:shapeLayer];

//开放路径
// 路径不需要将终点连接回起点。连接回起点的路径称为闭合路径，不连接回起点的称为开放路径。

shapeLayer = [CAShapeLayer layer];
shapeLayer.frame = CGRectMake(160, 140, 300, 300);
shapeLayer.lineWidth = 2.0;
shapeLayer.fillColor = nil;
shapeLayer.strokeColor = [[UIColor redColor] CGColor];

UIBezierPath *linePath=[UIBezierPath bezierPath];
[linePath moveToPoint:CGPointZero];
[linePath addLineToPoint:CGPointMake(0 , 120)];
[linePath addLineToPoint:CGPointMake(120 , 120)];
[linePath addLineToPoint:CGPointMake(120 , 0)];
shapeLayer.path = linePath.CGPath;
[[self.view layer] addSublayer:shapeLayer];
```



填充概念 //填充颜色

```
CAShapeLayer *squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(20, 30, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor] CGColor];
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];
```

many shapes or lines as you want in a single path object

```
CAShapeLayer *shapeLayer = [CAShapeLayer layer];
shapeLayer.frame = CGRectMake(20, 140, 200, 200);
shapeLayer.lineWidth = 2.0;
shapeLayer.fillColor = nil;
shapeLayer.strokeColor = [[UIColor redColor] CGColor];

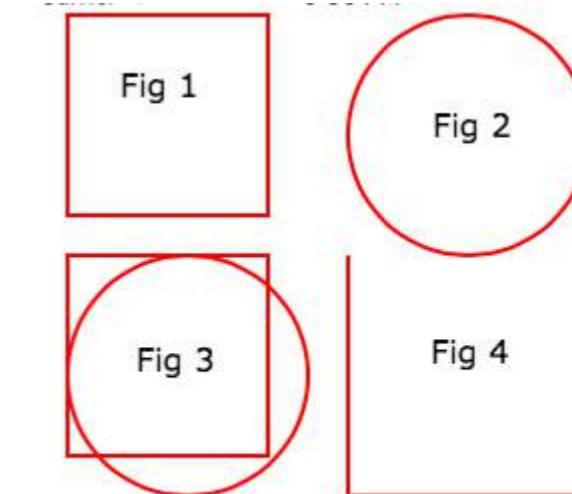
CGMutablePathRef combinedPath= CGPathCreateMutableCopy(circleShape.path);
CGPathAddPath(combinedPath, NULL, squareLayer.path);

shapeLayer.path = combinedPath;
[[self.view layer] addSublayer:shapeLayer];

//Open Path
// Paths do not need to connect their end points back to their starting points. A path that connects back to its starting point is called a closed path, and one that does not is called an open path.

shapeLayer = [CAShapeLayer layer];
shapeLayer.frame = CGRectMake(160, 140, 300, 300);
shapeLayer.lineWidth = 2.0;
shapeLayer.fillColor = nil;
shapeLayer.strokeColor = [[UIColor redColor] CGColor];

UIBezierPath *linePath=[UIBezierPath bezierPath];
[linePath moveToPoint:CGPointZero];
[linePath addLineToPoint:CGPointMake(0 , 120)];
[linePath addLineToPoint:CGPointMake(120 , 120)];
[linePath addLineToPoint:CGPointMake(120 , 0)];
shapeLayer.path = linePath.CGPath;
[[self.view layer] addSublayer:shapeLayer];
```



Fill Concepts //Fill Color

```
CAShapeLayer *squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(20, 30, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor] CGColor];
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];
```

```

//填充图案颜色
//images.jpeg

squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(140, 30, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor colorWithPatternImage:[UIImage imageNamed:@"images.jpeg"]]CGColor];
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];

//填充规则

//类型 1: kCAFillRuleNonZero
squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(0, 140, 150, 150);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.fillRule = kCAFillRuleNonZero; // 指定规则类型
squareLayer.strokeColor = [[UIColor redColor] CGColor];
UIBezierPath *outerPath = [UIBezierPath bezierPathWithRect:CGRectMakeInset(squareLayer.bounds, 20.0, 20.0)];
UIBezierPath *innerPath = [UIBezierPath bezierPathWithRect:CGRectMakeInset(squareLayer.bounds, 50.0, 50.0)];
CGMutablePathRef combinedPath= CGPathCreateMutableCopy(outerPath.CGPath);
CGPathAddPath(combinedPath, NULL, innerPath.CGPath);
squareLayer.path = combinedPath;
[[self.view layer] addSublayer:squareLayer];

//类型 2: kCAFillRuleEvenOdd
squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(140, 140, 150, 150);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.fillRule = kCAFillRuleEvenOdd; // 指定规则类型
squareLayer.strokeColor = [[UIColor redColor] CGColor];
outerPath = [UIBezierPath bezierPathWithRect:CGRectMakeInset(squareLayer.bounds, 20.0, 20.0)];
innerPath = [UIBezierPath bezierPathWithRect:CGRectMakeInset(squareLayer.bounds, 50.0, 50.0)];
combinedPath= CGPathCreateMutableCopy(outerPath.CGPath);
CGPathAddPath(combinedPath, NULL, innerPath.CGPath);
squareLayer.path = combinedPath;
[[self.view layer] addSublayer:squareLayer];

```

```

//Fill Pattern Color
//images.jpeg

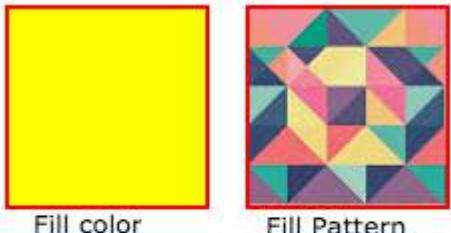
squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(140, 30, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor colorWithPatternImage:[UIImage imageNamed:@"images.jpeg"]]CGColor];
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];

//Fill Rule

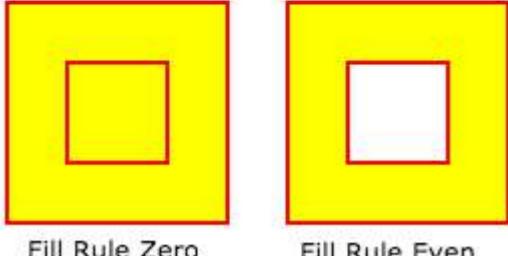
//Type 1: kCAFillRuleNonZero
squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(0, 140, 150, 150);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.fillRule = kCAFillRuleNonZero; // indicate the rule type
squareLayer.strokeColor = [[UIColor redColor] CGColor];
UIBezierPath *outerPath = [UIBezierPath bezierPathWithRect:CGRectMakeInset(squareLayer.bounds, 20.0, 20.0)];
UIBezierPath *innerPath = [UIBezierPath bezierPathWithRect:CGRectMakeInset(squareLayer.bounds, 50.0, 50.0)];
CGMutablePathRef combinedPath= CGPathCreateMutableCopy(outerPath.CGPath);
CGPathAddPath(combinedPath, NULL, innerPath.CGPath);
squareLayer.path = combinedPath;
[[self.view layer] addSublayer:squareLayer];

//Type 2: kCAFillRuleEvenOdd
squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(140, 140, 150, 150);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.fillRule = kCAFillRuleEvenOdd; // indicate the rule type
squareLayer.strokeColor = [[UIColor redColor] CGColor];
outerPath = [UIBezierPath bezierPathWithRect:CGRectMakeInset(squareLayer.bounds, 20.0, 20.0)];
innerPath = [UIBezierPath bezierPathWithRect:CGRectMakeInset(squareLayer.bounds, 50.0, 50.0)];
combinedPath= CGPathCreateMutableCopy(outerPath.CGPath);
CGPathAddPath(combinedPath, NULL, innerPath.CGPath);
squareLayer.path = combinedPath;
[[self.view layer] addSublayer:squareLayer];

```



Fill color Fill Pattern



Fill Rule Zero Fill Rule Even

列出了访问样式属性

填充颜色

根据绘制的形状填充颜色。

填充规则

填充规则，有两种规则应用于绘制形状。

1. `kCAFillRuleNonZero`
2. `kCAFillRuleEvenOdd`

线帽样式

以下类型用于更改线条的样式。

1. `KCALineCapButt`
2. `KCALineCapRound`
3. `KCALineCapSquare`

线段虚线模式

应用于形状路径描边时的虚线模式。

创建虚线样式当你描边该线时。

线段虚线相位

应用于形状路径描边时的虚线相位。可动画。

线段连接方式

形状路径的线段连接样式。以下样式用于绘制线段连接样式。

1. `KCALineJoinMiter`
2. `KCALineJoinRound`
3. `KCALineJoinBevel`

线宽

用于设置线宽。

斜接限制

描边形状路径时使用的斜接限制。可动画。

描边颜色

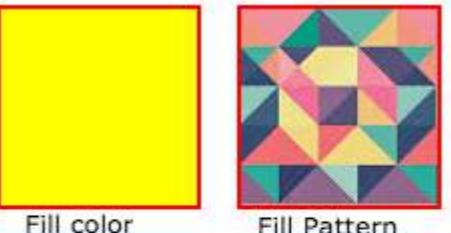
根据线条路径设置描边颜色。

描边起点

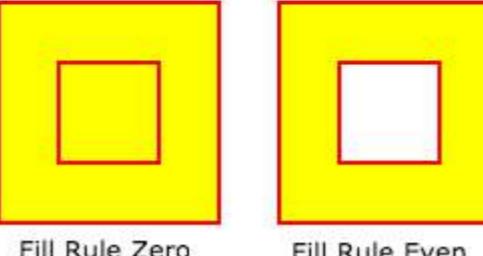
描边开始的位置。

描边终点

描边结束的位置。



Fill color Fill Pattern



Fill Rule Zero Fill Rule Even

Listed the accessing Style properties

`fillColor`

Fill the color based on the drawn shape.

`fillRule`

Fill Rule the there are two rule `is` applied to draw the shape.

1. `kCAFillRuleNonZero`
2. `kCAFillRuleEvenOdd`

`lineCap`

Below type used to change the style of the line.

1. `KCALineCapButt`
2. `KCALineCapRound`
3. `KCALineCapSquare`

`lineDashPattern`

The dash pattern applied to the shape's path when stroked.

Create DashStyle `while` you will stroke the line.

`lineDashPhase`

The dash phase applied to the shape's path when stroked. Animatable.

`lineJoin`

Line join style `for` the shape path. Below style use to draw the line join style.

1. `KCALineJoinMiter`
2. `KCALineJoinRound`
3. `KCALineJoinBevel`

`lineWidth`

Which using to `set` the line width.

`miterLimit`

The miter limit used when stroking the shape's path. Animatable.

`strokeColor`

`Set` the stroke color based on the path of the line.

`strokeStart`

When the stroke will start.

`strokeEnd`

When the stroke will end.

第116章：WKWebView

WKWebView是iOS 8和OS X Yosemite中引入的现代WebKit API的核心。它取代了UIKit中的UIWebView和AppKit中的WebView，在两个平台上提供了一致的API。

WKWebView拥有流畅的60fps滚动、内置手势、应用与网页之间的简化通信，以及与Safari相同的JavaScript引擎，是WWDC 2014发布的最重要的内容之一。

第116.1节：添加从应用程序包加载的自定义用户脚本

```
let configuration = WKWebViewConfiguration()

if let path = NSBundle.mainBundle().pathForResource("customUserScript", ofType: "js"),
    source = try? NSString(contentsOfFile: path, encoding: NSUTF8StringEncoding) as String {

    let userScript = WKUserScript(source: source, injectionTime:
        WKUserScriptInjectionTime.AtDocumentStart, forMainFrameOnly: false)

    let userContentController = WKUserContentController()
    userContentController.addUserScript(userScript)
    configuration.userContentController = userContentController
}

let webView = WKWebView(frame: self.view.bounds, configuration: configuration)
```

在WKUserScriptInjectionTime枚举中的任何值都是有效的：`.AtDocumentStart`, `.AtDocumentEnd`

第116.2节：从JavaScript发送消息并在本地端处理

可以使用以下代码从JavaScript发送消息

```
window.webkit.messageHandlers.{NAME}.postMessage()
```

以下是创建脚本消息处理器以处理消息的方法：

```
class NotificationScriptMessageHandler: NSObject, WKScriptMessageHandler {
    func userContentController(userContentController: WKUserContentController,
        didReceiveScriptMessage message: WKScriptMessage!) {
        if message.name == "{NAME}" {
            // 确保处理正确的消息
            print(message.body)
        }
    }
}
```

以下是在WKWebView中配置脚本消息处理器的方法：

```
let configuration = WKWebViewConfiguration()
let userContentController = WKUserContentController()
let handler = NotificationScriptMessageHandler()
userContentController.addScriptMessageHandler(handler, name: "{NAME}")
configuration.userContentController = userContentController
```

Chapter 116: WKWebView

WKWebView is the centerpiece of the modern WebKit API introduced in iOS 8 & OS X Yosemite. It replaces UIWebView in UIKit and WebView in AppKit, offering a consistent API across the two platforms.

Boasting responsive 60fps scrolling, built-in gestures, streamlined communication between app and webpage, and the same JavaScript engine as Safari, WKWebView is one of the most significant announcements to come out of WWDC 2014.

Section 116.1: Adding custom user script loaded from app bundle

```
let configuration = WKWebViewConfiguration()

if let path = NSBundle.mainBundle().pathForResource("customUserScript", ofType: "js"),
    source = try? NSString(contentsOfFile: path, encoding: NSUTF8StringEncoding) as String {

    let userScript = WKUserScript(source: source, injectionTime:
        WKUserScriptInjectionTime.AtDocumentStart, forMainFrameOnly: false)

    let userContentController = WKUserContentController()
    userContentController.addUserScript(userScript)
    configuration.userContentController = userContentController
}

let webView = WKWebView(frame: self.view.bounds, configuration: configuration)
```

Any value in the `WKUserScriptInjectionTime` enum is valid: `.AtDocumentStart`, `.AtDocumentEnd`

Section 116.2: Send messages from JavaScript and Handle them on the native side

Messages can be sent from JavaScript using the following code

```
window.webkit.messageHandlers.{NAME}.postMessage()
```

Here how to create a script message handler to handle the messages:

```
class NotificationScriptMessageHandler: NSObject, WKScriptMessageHandler {
    func userContentController(userContentController: WKUserContentController,
        didReceiveScriptMessage message: WKScriptMessage!) {
        if message.name == "{NAME}" {
            // to be sure of handling the correct message
            print(message.body)
        }
    }
}
```

Here how to configure the script message handler in the WKWebView:

```
let configuration = WKWebViewConfiguration()
let userContentController = WKUserContentController()
let handler = NotificationScriptMessageHandler()
userContentController.addScriptMessageHandler(handler, name: "{NAME}")
configuration.userContentController = userContentController
```

```
let webView = WKWebView(frame: self.view.bounds, configuration: configuration)
```

注意：多次使用`addScriptMessageHandler:name:`添加相同的“{NAME}”处理器，会导致`NSInvalidArgumentException`异常。

第116.3节：创建一个简单的网页浏览器

```
import UIKit
import WebKit

class ViewController: UIViewController, UISearchBarDelegate, WKNavigationDelegate, WKUIDelegate {

    var searchBar: UISearchBar! //所有网页浏览器都有搜索栏。
    var webView: WKWebView! //我们将使用的WKWebView。
    var toolbar: UIToolbar! //底部的工具栏，就像Safari中的一样。
    var activityIndicator: UIActivityIndicatorView! //活动指示器，用于告知用户页面正在加载。

    override func viewDidLoad() {
        super.viewDidLoad()

        self.initControls()
        self.setTheme()
        self.doLayout()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    func initControls() {
        self.searchBar = UISearchBar()

        //WKUserContentController 允许我们向 webView 添加 JavaScript 脚本，这些脚本将在页面加载开始时或结束时运行。
        let configuration = WKWebViewConfiguration()
        let contentController = WKUserContentController()
        configuration.userContentController = contentController

        //使用自定义配置创建 webView。
        self.webView = WKWebView(frame: .zero, configuration: configuration)

        self.toolbar = UIToolbar()
        self.layoutToolbar()

        self.activityIndicator = UIActivityIndicatorView(activityIndicatorStyle: .gray)
        self.activityIndicator.hidesWhenStopped = true
    }

    func setTheme() {
        self.edgesForExtendedLayout = UIRectEdge(rawValue: 0)
        self.navigationController?.navigationBar.barTintColor = UIColor.white()

        //设置键盘和搜索栏的主题。设置代理。
        self.searchBar.delegate = self
        self.searchBar.returnKeyType = .go
    }
}
```

```
let webView = WKWebView(frame: self.view.bounds, configuration: configuration)
```

NOTE: adding same "{NAME}" handler with `addScriptMessageHandler:name:` more than once, results in `NSInvalidArgumentException` exception.

Section 116.3: Creating a Simple WebBrowser

```
import UIKit
import WebKit

class ViewController: UIViewController, UISearchBarDelegate, WKNavigationDelegate, WKUIDelegate {

    var searchBar: UISearchBar! //All web-browsers have a search-bar.
    var webView: WKWebView! //The WKWebView we'll use.
    var toolbar: UIToolbar! //Toolbar at the bottom just like in Safari.
    var activityIndicator: UIActivityIndicatorView! //Activity indicator to let the user know the page is loading.

    override func viewDidLoad() {
        super.viewDidLoad()

        self.initControls()
        self.setTheme()
        self.doLayout()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    func initControls() {
        self.searchBar = UISearchBar()

        //WKUserContentController allows us to add Javascript scripts to our webView that will run either at the beginning of a page load OR at the end of a page load.

        let configuration = WKWebViewConfiguration()
        let contentController = WKUserContentController()
        configuration.userContentController = contentController

        //create the webView with the custom configuration.
        self.webView = WKWebView(frame: .zero, configuration: configuration)

        self.toolbar = UIToolbar()
        self.layoutToolbar()

        self.activityIndicator = UIActivityIndicatorView(activityIndicatorStyle: .gray)
        self.activityIndicator.hidesWhenStopped = true
    }

    func setTheme() {
        self.edgesForExtendedLayout = UIRectEdge(rawValue: 0)
        self.navigationController?.navigationBar.barTintColor = UIColor.white()

        //Theme the keyboard and searchBar. Setup delegates.
        self.searchBar.delegate = self
        self.searchBar.returnKeyType = .go
    }
}
```

```

self.searchbar.searchBarStyle = .prominent
self.searchbar.placeholder = "搜索或输入网站名称"
self.searchbar.autocapitalizationType = .none
self.searchbar.autocorrectionType = .no

//设置 WebView 的代理。
self.webView.navigationDelegate = self //处理页面导航的代理self.webView.uiDelegate = self //处理新标签页、窗口、弹出窗口、布局等的代理

    self.activityIndicator.transform = CGAffineTransform(scaleX: 1.5, y: 1.5)
}

func 布局工具栏() {
    //浏览器通常有后退按钮、前进按钮、刷新按钮和
    新标签页/新窗口按钮。

    var items = Array<UIBarButtonItem>()

    let space = UIBarButtonItem(barButtonSystemItem: .flexibleSpace, target: nil, action: nil)

    items.append(UIBarButtonItem(title: "<", style: .plain, target: self, action:
#selector(onBackPressed)))
    items.append(space)
    items.append(UIBarButtonItem(title: ">", style: .plain, target: self, action:
#selector(onForwardPressed)))
        items.append(space)
    items.append(UIBarButtonItem(barButtonSystemItem: .refresh, target: self, action:
#selector(onRefreshPressed)))
    items.append(space)
    items.append(UIBarButtonItem(barButtonSystemItem: .organize, target: self, action:
#selector(onTabPressed)))

        self.toolbar.items = items
}

func doLayout() {
    //将搜索栏添加到导航栏。
    self.navigationItem.titleView = self.searchbar

    //将所有其他子视图添加到 self.view。
    self.view.addSubview(self.webView)
    self.view.addSubview(self.toolbar)
    self.view.addSubview(self.activityIndicator)

    //设置将要约束的视图。

    let views: [String: AnyObject] = ["webView": self.webView, "toolbar": self.toolbar,
"activityIndicator": self.activityIndicator];
    var constraints = Array<String>();

    constraints.append("H:|-0-[webView]-0-|")
    constraints.append("H:|-0-[toolbar]-0-|")
    constraints.append("V:|-0-[webView]-0-[toolbar(50)]-0-|")

    //使用上述视觉约束约束子视图。

    for constraint in constraints {
        self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views))
    }
}

```

```

self.searchbar.searchBarStyle = .prominent
self.searchbar.placeholder = "Search or enter website name"
self.searchbar.autocapitalizationType = .none
self.searchbar.autocorrectionType = .no

//Set the WebView's delegate.
self.webView.navigationDelegate = self //Delegate that handles page navigation
self.webView.uiDelegate = self //Delegate that handles new tabs, windows, popups, layout,
etc..

    self.activityIndicator.transform = CGAffineTransform(scaleX: 1.5, y: 1.5)
}

func layoutToolbar() {
    //Browsers typically have a back button, forward button, refresh button, and
newTab/newWindow button.

    var items = Array<UIBarButtonItem>()

    let space = UIBarButtonItem(barButtonSystemItem: .flexibleSpace, target: nil, action: nil)

    items.append(UIBarButtonItem(title: "<", style: .plain, target: self, action:
#selector(onBackPressed)))
    items.append(space)
    items.append(UIBarButtonItem(title: ">", style: .plain, target: self, action:
#selector(onForwardPressed)))
    items.append(space)
    items.append(UIBarButtonItem(barButtonSystemItem: .refresh, target: self, action:
#selector(onRefreshPressed)))
    items.append(space)
    items.append(UIBarButtonItem(barButtonSystemItem: .organize, target: self, action:
#selector(onTabPressed)))

        self.toolbar.items = items
}

func doLayout() {
    //Add the searchBar to the navigationBar.
    self.navigationItem.titleView = self.searchbar

    //Add all other subViews to self.view.
    self.view.addSubview(self.webView)
    self.view.addSubview(self.toolbar)
    self.view.addSubview(self.activityIndicator)

    //Setup which views will be constrained.

    let views: [String: AnyObject] = ["webView": self.webView, "toolbar": self.toolbar,
"activityIndicator": self.activityIndicator];
    var constraints = Array<String>();

    constraints.append("H:|-0-[webView]-0-|")
    constraints.append("H:|-0-[toolbar]-0-|")
    constraints.append("V:|-0-[webView]-0-[toolbar(50)]-0-|")

    //constrain the subviews using the above visual constraints.

    for constraint in constraints {
        self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views))
    }
}

```

```

        for view in self.view.subviews {
            view.translatesAutoresizingMaskIntoConstraints = false
        }

        //将活动指示器约束到视图的中心。
        self.view.addConstraint(NSLayoutConstraint(item: self.activityIndicator, attribute: .centerX, relatedBy: .equal, toItem: self.view, attribute: .centerX, multiplier: 1.0, constant: 0.0))
        self.view.addConstraint(NSLayoutConstraint(item: self.activityIndicator, attribute: .centerY, relatedBy: .equal, toItem: self.view, attribute: .centerY, multiplier: 1.0, constant: 0.0))
    }

    //搜索栏代理方法

    func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {
        self.searchbar.resignFirstResponder()

        if let searchText = self.searchbar.text, url = URL(string: searchText) {
            //从搜索栏获取URL。使用该URL创建一个新的NSURLRequest，并告诉webView导航到该URL/页面。同时指定超时时间以防页面加载过久。还处理了cookie/缓存策略。

            let request = URLRequest(url: url, cachePolicy: .useProtocolCachePolicy, timeoutInterval: 30)
            self.webView.load(request)
        }
    }

    //工具栏代理

    func onBackButtonPressed(button: UIBarButtonItem) {
        if (self.webView.canGoBack) { //允许用户返回上一页。
            self.webView.goBack()
        }
    }

    func onForwardButtonPressed(button: UIBarButtonItem) {
        if (self.webView.canGoForward) { //允许用户前进到下一页。
            self.webView.goForward()
        }
    }

    func onRefreshPressed(button: UIBarButtonItem) {
        self.webView.reload() //重新加载当前页面。
    }

    func onTabPressed(button: UIBarButtonItem) {
        //TODO: 打开一个新标签页或网页。
    }

    //网页视图代理

    func webView(_ webView: WKWebView, decidePolicyFor navigationAction: WKNavigationAction, decisionHandler: (WKNavigationActionPolicy) -> Void) {
        decisionHandler(.allow) //允许用户导航到请求的页面。
    }

```

```

        for view in self.view.subviews {
            view.translatesAutoresizingMaskIntoConstraints = false
        }

        //constraint the activity indicator to the center of the view.
        self.view.addConstraint(NSLayoutConstraint(item: self.activityIndicator, attribute: .centerX, relatedBy: .equal, toItem: self.view, attribute: .centerX, multiplier: 1.0, constant: 0.0))
        self.view.addConstraint(NSLayoutConstraint(item: self.activityIndicator, attribute: .centerY, relatedBy: .equal, toItem: self.view, attribute: .centerY, multiplier: 1.0, constant: 0.0))
    }

    //Searchbar Delegates

    func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {
        self.searchbar.resignFirstResponder()

        if let searchText = self.searchbar.text, url = URL(string: searchText) {
            //Get the URL from the search bar. Create a new NSURLRequest with it and tell the webView to navigate to that URL/Page. Also specify a timeout for if the page takes too long. Also handles cookie/caching policy.

            let request = URLRequest(url: url, cachePolicy: .useProtocolCachePolicy, timeoutInterval: 30)
            self.webView.load(request)
        }
    }

    //Toolbar Delegates

    func onBackButtonPressed(button: UIBarButtonItem) {
        if (self.webView.canGoBack) { //allow the user to go back to the previous page.
            self.webView.goBack()
        }
    }

    func onForwardButtonPressed(button: UIBarButtonItem) {
        if (self.webView.canGoForward) { //allow the user to go forward to the next page.
            self.webView.goForward()
        }
    }

    func onRefreshPressed(button: UIBarButtonItem) {
        self.webView.reload() //reload the current page.
    }

    func onTabPressed(button: UIBarButtonItem) {
        //TODO: Open a new tab or web-page.
    }

    //WebView Delegates

    func webView(_ webView: WKWebView, decidePolicyFor navigationAction: WKNavigationAction, decisionHandler: (WKNavigationActionPolicy) -> Void) {
        decisionHandler(.allow) //allow the user to navigate to the requested page.
    }

```

```

func webView(_ webView: WKWebView, decidePolicyFor navigationResponse: WKNavigationResponse,
decisionHandler: (WKNavigationResponsePolicy) -> Void) {
    decisionHandler(.allow) //允许 webView 处理响应。
}

func webView(_ webView: WKWebView, didStartProvisionalNavigation navigation: WKNavigation!) {
    self.activityIndicator.startAnimating()
}

func webView(_ webView: WKWebView, didFailProvisionalNavigation navigation: WKNavigation!,
withError error: NSError) {
    self.activityIndicator.stopAnimating()

    //处理错误。向用户显示一个警告，告诉他们发生了什么。
    let alert = UIAlertController(title: "错误", message: error.localizedDescription,
preferredStyle: .alert)
    let action = UIAlertAction(title: "确定", style: .default) { (action) in
        alert.dismiss(animated: true, completion: nil)
    }
    alert.addAction(action)
    self.present(alert, animated: true, completion: nil)
}

func webView(_ webView: WKWebView, didFinish navigation: WKNavigation!) {
    self.activityIndicator.stopAnimating()

    //使用网页的最终端点 URL 更新我们的搜索栏。
    if let url = self.webView.url {
        self.searchbar.text = url.absoluteString ?? self.searchbar.text
    }
}

func webView(_ webView: WKWebView, didReceiveServerRedirectForProvisionalNavigation navigation: WKNavigation!) {
    //当 WebView 收到跳转到不同页面或端点时，会调用此方法。
}

func webView(_ webView: WKWebView, didCommit navigation: WKNavigation!) {
    //当网页内容开始加载时，会调用此方法。
}

func webView(_ webView: WKWebView, didFail navigation: WKNavigation!, withError error: NSError) {
}

func webView(_ webView: WKWebView, didReceive challenge: URLAuthenticationChallenge,
completionHandler: (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {
    completionHandler(.performDefaultHandling, .none) //默认处理 SSL 连接。
    //我们没有进行 SSL 固定或自定义证书处理。
}

//WebView 的 UINavigation 代理方法

//当 WebView 或已加载页面想要打开新窗口/标签页时，会调用此方法。
func webView(_ webView: WKWebView, createWebViewWith configuration: WKWebViewConfiguration, for

```

```

func webView(_ webView: WKWebView, decidePolicyFor navigationResponse: WKNavigationResponse,
decisionHandler: (WKNavigationResponsePolicy) -> Void) {
    decisionHandler(.allow) //allow the webView to process the response.
}

func webView(_ webView: WKWebView, didStartProvisionalNavigation navigation: WKNavigation!) {
    self.activityIndicator.startAnimating()
}

func webView(_ webView: WKWebView, didFailProvisionalNavigation navigation: WKNavigation!,
withError error: NSError) {
    self.activityIndicator.stopAnimating()

    //Handle the error. Display an alert to the user telling them what happened.
    let alert = UIAlertController(title: "Error", message: error.localizedDescription,
preferredStyle: .alert)
    let action = UIAlertAction(title: "OK", style: .default) { (action) in
        alert.dismiss(animated: true, completion: nil)
    }
    alert.addAction(action)
    self.present(alert, animated: true, completion: nil)
}

func webView(_ webView: WKWebView, didFinish navigation: WKNavigation!) {
    self.activityIndicator.stopAnimating()

    //Update our search bar with the webPage's final endpoint-URL.
    if let url = self.webView.url {
        self.searchbar.text = url.absoluteString ?? self.searchbar.text
    }
}

func webView(_ webView: WKWebView, didReceiveServerRedirectForProvisionalNavigation navigation: WKNavigation!) {
    //When the webview receives a "Redirect" to a different page or endpoint, this is called.
}

func webView(_ webView: WKWebView, didCommit navigation: WKNavigation!) {
    //When the content for the webpage starts arriving, this is called.
}

func webView(_ webView: WKWebView, didFail navigation: WKNavigation!, withError error: NSError) {
}

func webView(_ webView: WKWebView, didReceive challenge: URLAuthenticationChallenge,
completionHandler: (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {
    completionHandler(.performDefaultHandling, .none) //Handle SSL connections by default. We
aren't doing SSL pinning or custom certificate handling.
}

//WebView's UINavigation Delegates

//This is called when a webView or existing loaded page wants to open a new window/tab.
func webView(_ webView: WKWebView, createWebViewWith configuration: WKWebViewConfiguration, for

```

```

navigationAction: WKNavigationAction, windowFeatures: WKWindowFeatures) -> WKWebView? {
    // 表示新标签页/窗口的视图。该视图左上角会有一个关闭按钮 + 一个 webView。
    let container = UIView()

    // 新标签页需要一个退出按钮。
    let XButton = UIButton()
    XButton.addTarget(self, action: #selector(onWebViewExit), for: .touchUpInside)
    XButton.layer.cornerRadius = 22.0

    // 创建新的 webView 窗口。
    let webView = WKWebView(frame: .zero, configuration: configuration)
    webView.navigationDelegate = self
    webView.uiDelegate = self

    // 布局标签页。
    container.addSubview(XButton)
    container.addSubview(webView)

    let views: [String: AnyObject] = ["XButton": XButton, "webView": webView];
    var constraints = Array<String>()

    约束条件。append("H:|-(-22)-[XButton(44)]")
    约束条件。append("H:|-0-[webView]-0-|")
    约束条件。append("V:|-(-22)-[XButton(44)]-0-[webView]-0-|")

    //约束子视图。
    for constraint in constraints {
        容器.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
            options: NSLayoutConstraintOptions(rawValue: 0), metrics: nil, views: views))
    }

    for 视图 in 容器.子视图 {
        view.translatesAutoresizingMaskIntoConstraints = false
    }

    //TODO: 将containerView添加到self.view或用新控制器呈现。跟踪标签页..
}

func onWebViewExit(button: UIButton) {
    //TODO: 销毁标签页。从当前窗口或控制器中移除新标签页。
}
}

```

在键盘中显示自定义的GO按钮：

```

navigationAction: WKNavigationAction, windowFeatures: WKWindowFeatures) -> WKWebView? {
    //The view that represents the new tab/window. This view will have an X button at the top
    //left corner + a webView.
    let container = UIView()

    //New tabs need an exit button.
    let XButton = UIButton()
    XButton.addTarget(self, action: #selector(onWebViewExit), for: .touchUpInside)
    XButton.layer.cornerRadius = 22.0

    //Create the new webView window.
    let webView = WKWebView(frame: .zero, configuration: configuration)
    webView.navigationDelegate = self
    webView.uiDelegate = self

    //Layout the tab.
    container.addSubview(XButton)
    container.addSubview(webView)

    let views: [String: AnyObject] = ["XButton": XButton, "webView": webView];
    var constraints = Array<String>()

    constraints.append("H:|-(-22)-[XButton(44)]")
    constraints.append("H:|-0-[webView]-0-|")
    constraints.append("V:|-(-22)-[XButton(44)]-0-[webView]-0-|")

    //constrain the subviews.
    for constraint in constraints {
        container.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
            options: NSLayoutConstraintOptions(rawValue: 0), metrics: nil, views: views))
    }

    for view in container.subviews {
        view.translatesAutoresizingMaskIntoConstraints = false
    }

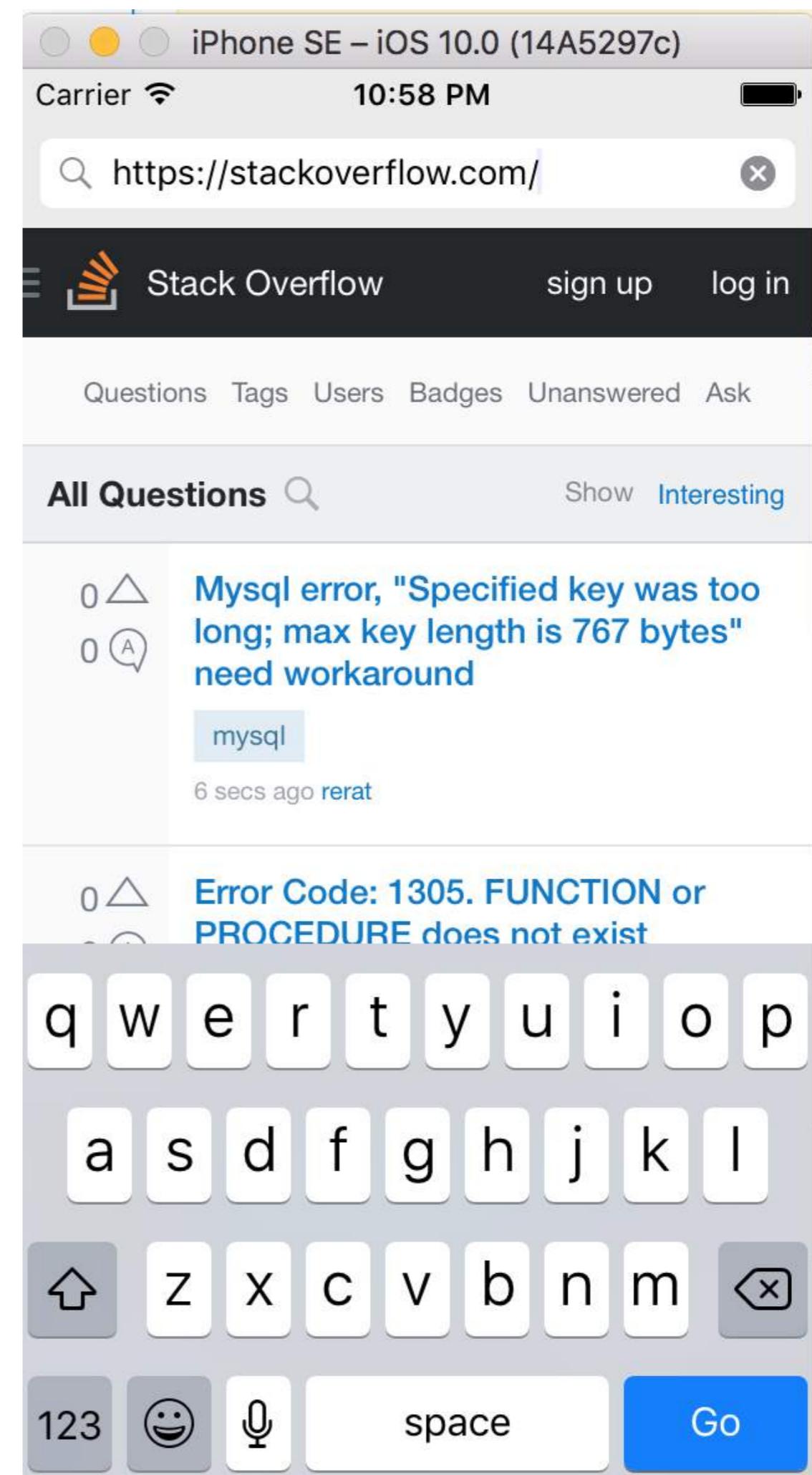
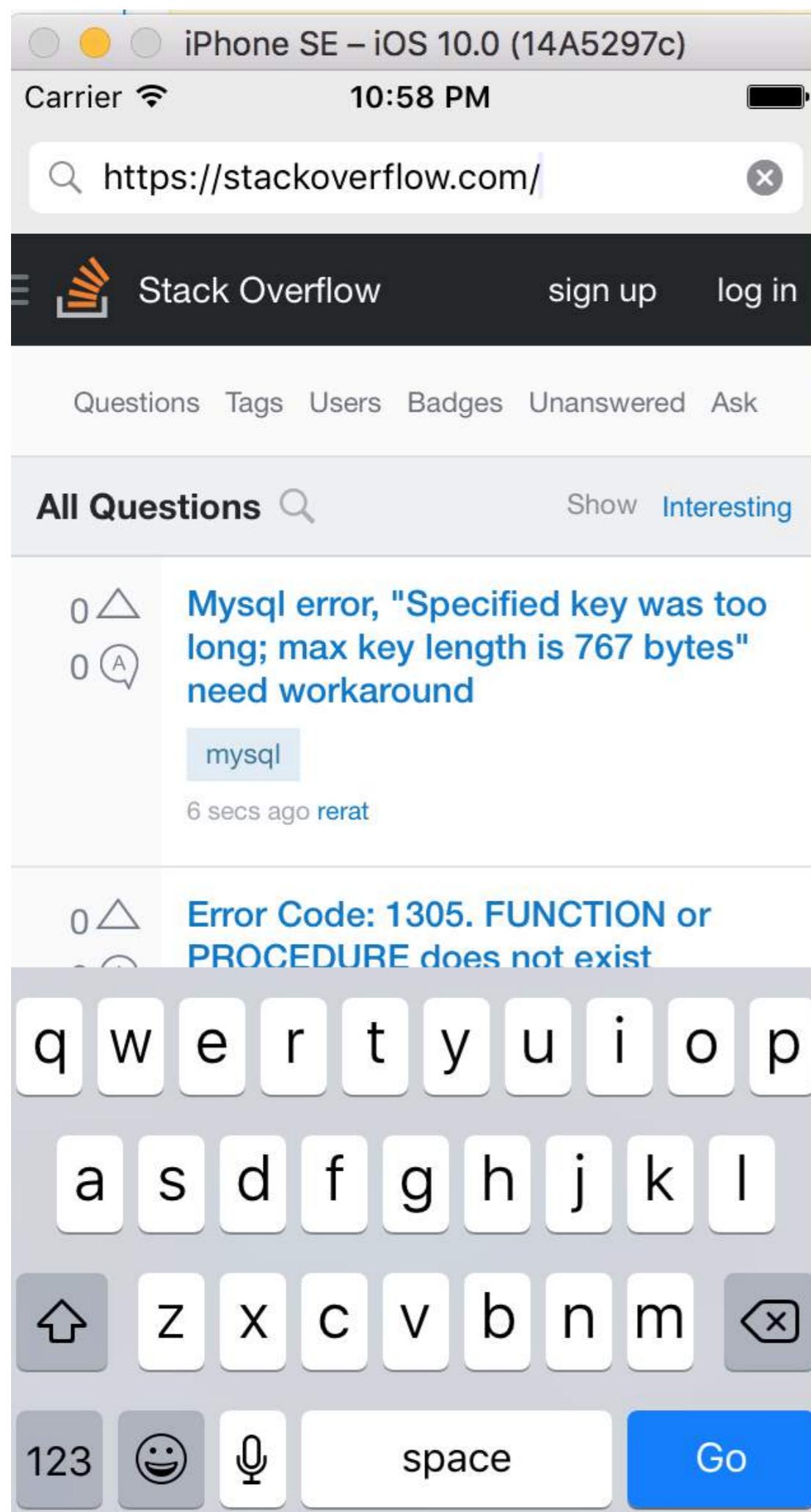
    //TODO: Add the containerView to self.view or present it with a new controller. Keep track
    //of tabs..

    return webView
}

func onWebViewExit(button: UIButton) {
    //TODO: Destroy the tab. Remove the new tab from the current window or controller.
}
}

```

Showing the custom GO button in the keyboard:



显示工具栏和完全加载的页面。

Showing the toolbar and fully loaded page.

iPhone SE – iOS 10.0 (14A5297c)
Carrier 10:58 PM

https://stackoverflow.com/

Stack Overflow sign up log in

Questions Tags Users Badges Unanswered Ask

All Questions Show Interesting

0 ▲ Mysql error, "Specified key was too long; max key length is 767 bytes" need workaround

6 secs ago rerat

0 ▲ Error Code: 1305. FUNCTION or PROCEDURE does not exist

8 secs ago neubert

0 ▲ cqlengine python lib is not recognizing batchquery object

14 secs ago gotham

< >

iPhone SE – iOS 10.0 (14A5297c)
Carrier 10:58 PM

https://stackoverflow.com/

Stack Overflow sign up log in

Questions Tags Users Badges Unanswered Ask

All Questions Show Interesting

0 ▲ Mysql error, "Specified key was too long; max key length is 767 bytes" need workaround

6 secs ago rerat

0 ▲ Error Code: 1305. FUNCTION or PROCEDURE does not exist

8 secs ago neubert

0 ▲ cqlengine python lib is not recognizing batchquery object

14 secs ago gotham

< >

第117章：UUID（通用唯一标识符）

第117.1节：苹果的IFA与IFV（苹果广告标识符与供应商标识符）

- 您可以使用IFA来衡量广告点击量，使用IFV来衡量应用安装量。
- IFA具有内置的隐私机制，非常适合用于广告。相比之下，IFV供开发者内部使用，用于衡量安装其应用的用户。

IFA

- ASIdentifierManager类提供
 - advertisingIdentifier: UUID: 一个唯一的字母数字字符串，专属于每台设备，仅用于投放广告。
 - isAdvertisingTrackingEnabled : 一个布尔值，表示用户是否限制了广告追踪。

IFV

- ASIdentifierManager类提供
 - identifierForVendor : UUID : 一个字母数字字符串，用于唯一标识设备对应用程序的供应商。

在这里找到您的设备IFA和IFV。[here](#)

第117.2节：为iOS设备创建UUID字符串

这里我们可以在一行内创建UUID字符串。

表示UUID字符串，可用于唯一标识类型、接口和其他项目。

Swift 3.0

```
print(UUID().uuidString)
```

它对于使用唯一ID识别多个设备非常有用。

第117.3节：生成UUID

随机 UUID

Swift

```
func randomUUID() -> NSString{
    return NSUUID.UUID().UUIDString()
}
```

Objective-C

```
+ (NSString *)randomUUID {
    if(NSClassFromString(@"NSUUID")) { // 仅在 iOS >= 6.0 可用
        return [[NSUUID UUID] UUIDString];
    }
    CFUUIDRef uuidRef = CFUUIDCreate(kCFAllocatorDefault);
```

Chapter 117: UUID (Universally Unique Identifier)

Section 117.1: Apple's IFA vs. IFV (Apple Identifier for Advertisers vs. Identifier for Vendors)

- You can use the IFA for measuring ad clicks and the IFV for measuring app installs.
- IFA has built-in privacy mechanisms that make it perfect for advertising. In contrast, the IFV is for developers to use internally to measure users who install their apps.

IFA

- ASIdentifierManager class provides
 - advertisingIdentifier: UUID:** An alphanumeric string unique to each device, used only for serving advertisements.
 - isAdvertisingTrackingEnabled:** A Boolean value that indicates whether the user has limited ad tracking.

IFV

- ASIdentifierManager class provides
 - identifierForVendor: UUID:** An alphanumeric string that uniquely identifies a device to the app's vendor.

Find your device IFA and IFV [here](#).

Section 117.2: Create UUID String for iOS devices

Here we can create UUID **String** with in one line.

Represents UUID strings, which can be used to uniquely identify types, interfaces, and other items.

Swift 3.0

```
print(UUID().uuidString)
```

It is very useful for identify multiple devices with unique id.

Section 117.3: Generating UUID

Random UUID

Swift

```
func randomUUID() -> NSString{
    return NSUUID.UUID().UUIDString()
}
```

Objective-C

```
+ (NSString *)randomUUID {
    if(NSClassFromString(@"NSUUID")) { // only available in iOS >= 6.0
        return [[NSUUID UUID] UUIDString];
    }
    CFUUIDRef uuidRef = CFUUIDCreate(kCFAllocatorDefault);
```

```
CFStringRef cfuuid = CFUUIDCreateString(kCFAllocatorDefault, uuidRef);
CFRelease(uuidRef);
NSString *uuid = [((__bridge NSString *) cfuuid) copy];
CFRelease(cfuuid);
return uuid;
}
```

第117.4节：供应商标识符

版本 ≥ iOS 6

在一行代码中，我们可以获得如下的UUID：

Swift

```
let UDIDString = UIDevice.currentDevice().identifierForVendor?.UUIDString
```

Objective-C

```
NSString *UDIDString = [[[UIDevice currentDevice] identifierForVendor] UUIDString];
```

identifierForVendor 是一个唯一标识符，在单个设备上同一供应商的所有应用中保持不变，除非该设备上该供应商的所有应用都被删除。请参阅 [Apple的文档](#) 了解该

UUID 何时会发生变化。

```
CFStringRef cfuuid = CFUUIDCreateString(kCFAllocatorDefault, uuidRef);
CFRelease(uuidRef);
NSString *uuid = [((__bridge NSString *) cfuuid) copy];
CFRelease(cfuuid);
return uuid;
}
```

Section 117.4: Identifier for vendor

Version ≥ iOS 6

Within a single line, we can get an UUID like below:

Swift

```
let UDIDString = UIDevice.currentDevice().identifierForVendor?.UUIDString
```

Objective-C

```
NSString *UDIDString = [[[UIDevice currentDevice] identifierForVendor] UUIDString];
```

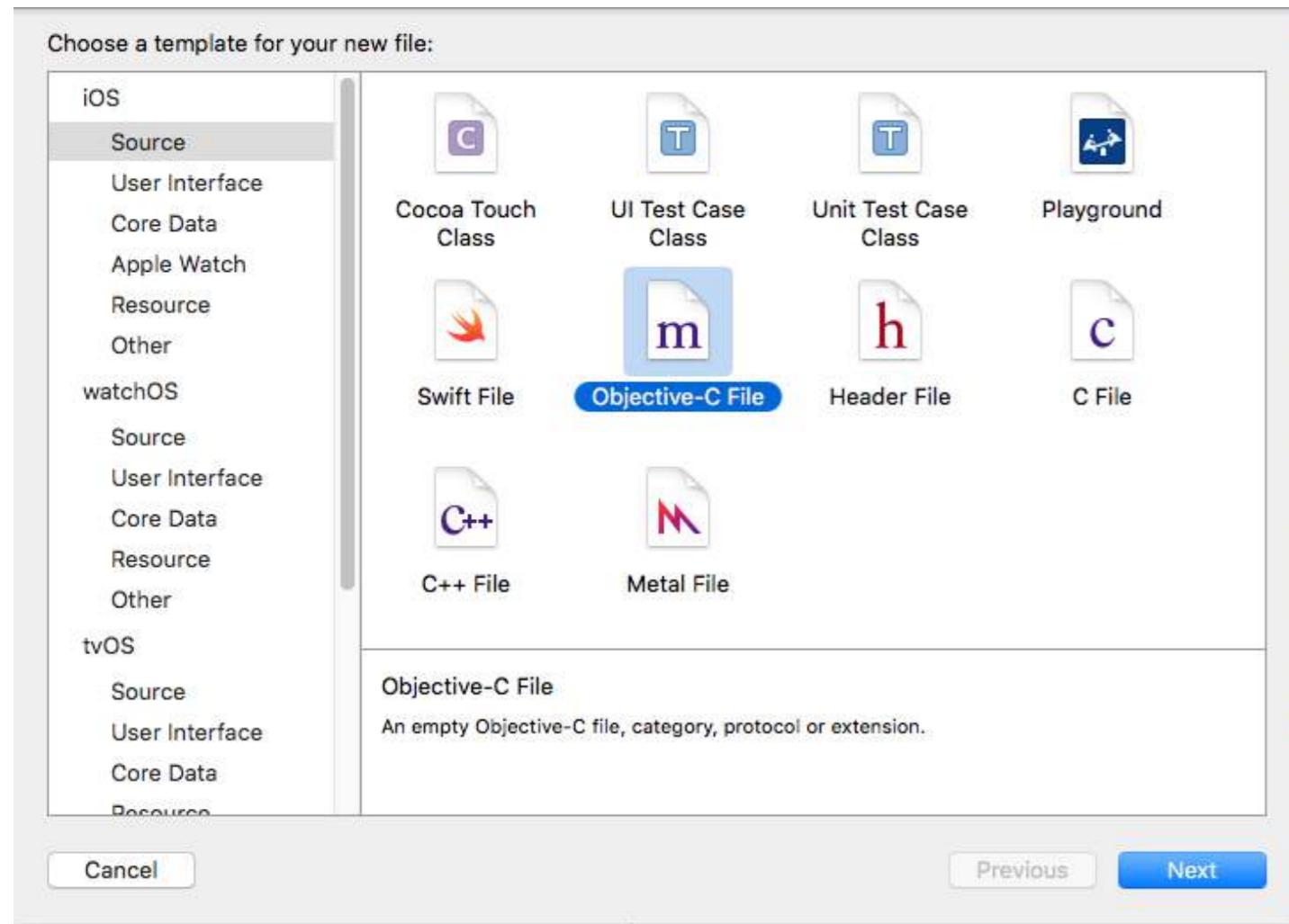
The `identifierForVendor` is an unique identifier that stays the same for every app of a single vendor on a single device, unless all of the vendor's apps are deleted from this device. See [Apple's documentation](#) about when this UUID changes.

第118章：类别

第118.1节：创建类别

类别提供了在不进行子类化或更改实际对象的情况下，为对象添加额外功能的能力。

例如，我们想设置一些自定义字体。让我们创建一个为 `UIFont` 类添加功能的类别。打开你的Xcode项目，点击文件 -> 新建 -> 文件，选择Objective-C文件，点击下一步，输入你的类别名称，比如“CustomFont”，选择文件类型为类别，类为`UIFont`，然后点击“下一步”，最后点击“创建”。

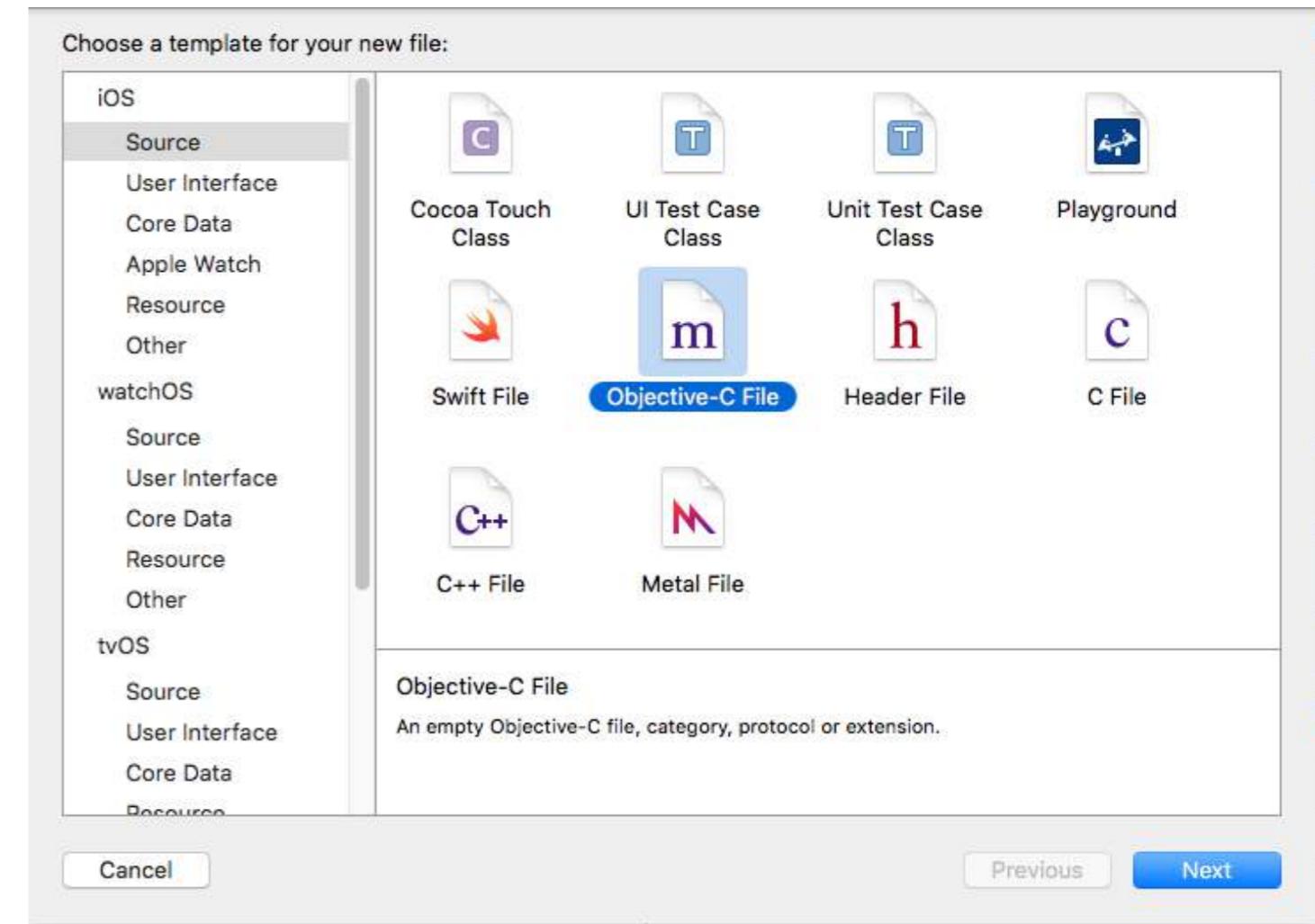


Chapter 118: Categories

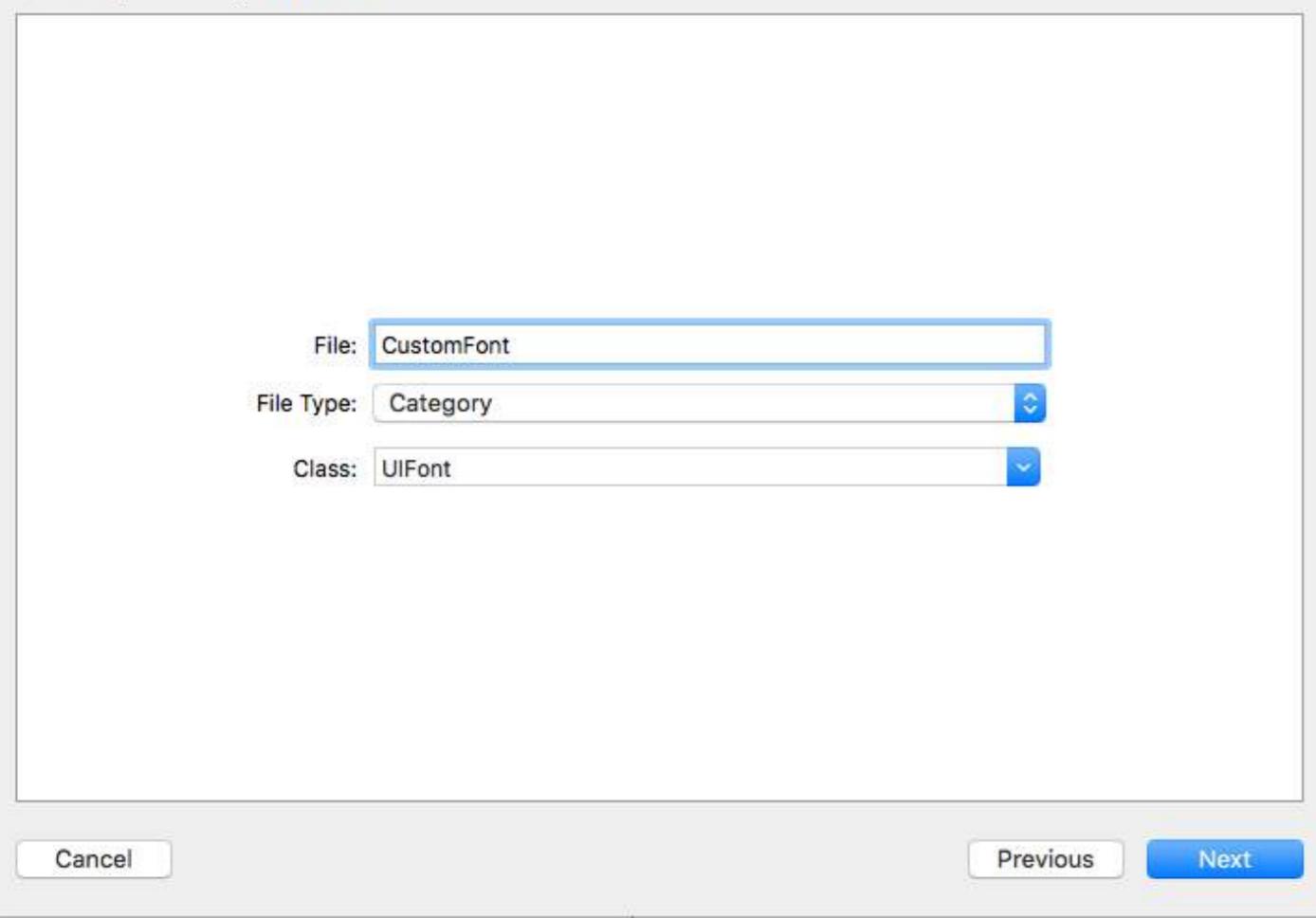
Section 118.1: Create a Category

Categories provide the ability to add some extra functionality to an object without subclassing or changing the actual object.

For example we want to set some custom fonts. Lets create a category that add functionality to `UIFont` class. Open your Xcode project, click on File -> New -> File and choose Objective-C file , click Next enter your category name say "CustomFont" choose file type as Category and Class as `UIFont` then Click "Next" followed by "Create."



Choose options for your new file:



声明类别方法：

点击 "UIFont+CustomFonts.h" 查看新类别的头文件。在接口中添加以下代码以声明该方法。

```
@interface UIFont (CustomFonts)  
+(UIFont *)productSansRegularFontOfSize:(CGFloat)size;  
@end
```

现在实现类别方法：

点击 "UIFont+CustomFonts.m" 查看类别的实现文件。添加以下代码以创建一个设置 ProductSansRegular 字体的方法。

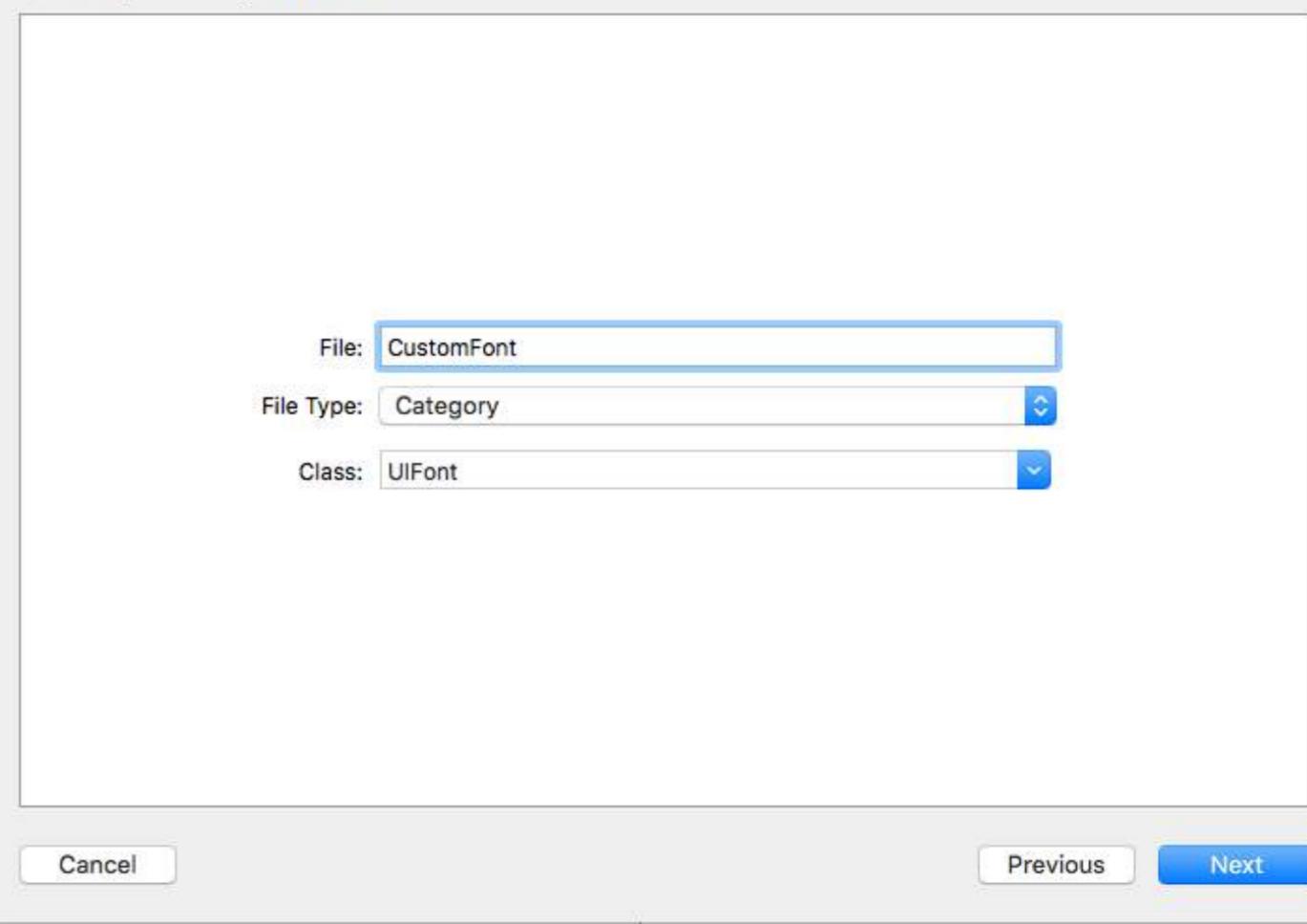
```
+ (UIFont *)productSansRegularFontOfSize:(CGFloat)size{  
    return [UIFont fontWithName:@"ProductSans-Regular" size:size];  
}
```

导入你的类别

```
#import "UIFont+CustomFonts.h"
```

现在设置标签字体

Choose options for your new file:



Declare the Category Method:

Click "UIFont+CustomFonts.h" to view the new category's header file. Add the following code to the interface to declare the method.

```
@interface UIFont (CustomFonts)  
+(UIFont *)productSansRegularFontOfSize:(CGFloat)size;  
@end
```

Now Implement the Category Method:

Click "UIFont+CustomFonts.m" to view the category's implementation file. Add the following code to create a method that will set ProductSansRegular Font.

```
+ (UIFont *)productSansRegularFontOfSize:(CGFloat)size{  
    return [UIFont fontWithName:@"ProductSans-Regular" size:size];  
}
```

Import your category

```
#import "UIFont+CustomFonts.h"
```

Now set the Label font

```
[self.label setFont:[UIFont productSansRegularFontOfSize:16.0]]; 
```

```
[self.label setFont:[UIFont productSansRegularFontOfSize:16.0]]; 
```

第119章：处理URL方案

参数	含义
aUrl	a NSURL实例，存储内置或自定义的方案字符串

第119.1节：使用内置URL方案打开邮件应用

Swift:

```
if let url = URL(string: "mailto://azimov@demo.com") {  
    if UIApplication.shared.canOpenURL(url) {  
        UIApplication.shared.openURL(url)  
    } else {  
        print("无法打开URL")  
    }  
}
```

Objective-C:

```
NSURL *url = [NSURL URLWithString:@"mailto://azimov@demo.com"];  
if ([[UIApplication sharedApplication] canOpenURL:url]) {  
    [[UIApplication sharedApplication] openURL:url];  
} else {  
    NSLog(@"无法打开URL");  
}
```

第119.2节：苹果URL方案

这些是iOS、OS X和watchOS 2及更高版本的原生应用支持的URL方案。

在Safari中打开链接：

Objective-C

```
NSString *stringURL = @"http://stackoverflow.com/";  
NSURL *url = [NSURL URLWithString:stringURL];  
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "http://stackoverflow.com/"  
if let url = URL(string: stringURL) {  
    UIApplication.shared.openURL(url)  
}
```

开始电话通话

Objective-C

```
NSString *stringURL = @"tel:1-408-555-5555";  
NSURL *url = [NSURL URLWithString:stringURL];  
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "tel:1-408-555-5555"  
if let url = URL(string: stringURL) {  
    UIApplication.shared.openURL(url)
```

Chapter 119: Handling URL Schemes

Parameter	Meaning
aUrl	a NSURL instance which stores a built-in or custom scheme string

Section 119.1: Using built-in URL scheme to open Mail app

Swift:

```
if let url = URL(string: "mailto://azimov@demo.com") {  
    if UIApplication.shared.canOpenURL(url) {  
        UIApplication.shared.openURL(url)  
    } else {  
        print("Cannot open URL")  
    }  
}
```

Objective-C:

```
NSURL *url = [NSURL URLWithString:@"mailto://azimov@demo.com"];  
if ([[UIApplication sharedApplication] canOpenURL:url]) {  
    [[UIApplication sharedApplication] openURL:url];  
} else {  
    NSLog(@"Cannot open URL");  
}
```

Section 119.2: Apple URL Schemes

These are URL schemes supported by native apps on iOS, OS X, and watchOS 2 and later.

Opening link in Safari:

Objective-C

```
NSString *stringURL = @"http://stackoverflow.com/";  
NSURL *url = [NSURL URLWithString:stringURL];  
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "http://stackoverflow.com/"  
if let url = URL(string: stringURL) {  
    UIApplication.shared.openURL(url)  
}
```

Starting a phone conversation

Objective-C

```
NSString *stringURL = @"tel:1-408-555-5555";  
NSURL *url = [NSURL URLWithString:stringURL];  
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "tel:1-408-555-5555"  
if let url = URL(string: stringURL) {  
    UIApplication.shared.openURL(url)
```

```
}
```

HTML

```
<a href="tel:1-408-555-5555">1-408-555-5555</a>
```

开始FaceTime通话

Objective-C

```
NSString *stringURL = @"facetime:14085551234";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "facetime:14085551234"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="facetime:14085551234">使用FaceTime连接</a>
<a href="facetime:user@example.com">使用FaceTime连接</a>
```

打开信息应用以撰写短信给收件人：

Objective-C

```
NSString *stringURL = @"sms:1-408-555-1212";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "sms:1-408-555-1212"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="sms:">启动信息应用</a>
<a href="sms:1-408-555-1212">新短信</a>
```

打开邮件应用以撰写邮件给收件人：

Objective-C

```
NSString *stringURL = @"mailto:foo@example.com";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
}
```

HTML

```
<a href="tel:1-408-555-5555">1-408-555-5555</a>
```

Starting a FaceTime conversation

Objective-C

```
NSString *stringURL = @"facetime:14085551234";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "facetime:14085551234"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="facetime:14085551234">Connect using FaceTime</a>
<a href="facetime:user@example.com">Connect using FaceTime</a>
```

Opening Messages App to compose an sms to recipient:

Objective-C

```
NSString *stringURL = @"sms:1-408-555-1212";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "sms:1-408-555-1212"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="sms:">Launch Messages App</a>
<a href="sms:1-408-555-1212">New SMS Message</a>
```

Opening Mail app to compose an email to recipient:

Objective-C

```
NSString *stringURL = @"mailto:foo@example.com";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "mailto:foo@example.com"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="mailto:frank@wwdcdemo.example.com">约翰·弗兰克</a>
```

您还可以在“收件人”、“抄送”和“密送”字段中包含主题字段、消息和多个收件人。（在 iOS 中，from 属性会被忽略。）以下示例展示了一个包含多个不同属性的 mailto URL：

```
mailto:foo@example.com?cc=bar@example.com&subject=来自库比蒂诺的问候！&body=希望你在这里！
```

注意：也可以使用MFMailComposeViewController在应用内呈现撰写邮件对话框。

```
let stringURL = "mailto:foo@example.com"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="mailto:frank@wwdcdemo.example.com">John Frank</a>
```

You can also include a subject field, a message, and multiple recipients in the To, Cc, and Bcc fields. (In iOS, the from attribute is ignored.) The following example shows a mailto URL that includes several different attributes:

```
mailto:foo@example.com?cc=bar@example.com&subject=Greetings%20from%20Cupertino!&body=Wish%20you%20were%20here!
```

Note: Compose email dialog can also be presented within app using MFMailComposeViewController.

第120章：Realm

第120.1节：带主键的RLMObject基类模型 - Objective-C

一个使用主键和一些通用默认属性的RLMObject基类模型示例。
子类随后可以设置其特定需求的元数据。

```
@interface BaseModel : RLMObject

@property NSString *uuid;
@property NSString *metadata;

@end

@implementation BaseModel

+ (NSString *)primaryKey
{
    return @"uuid";
}

+ (NSDictionary *)defaultPropertyValues
{
    NSMutableDictionary *defaultPropertyValues = [NSMutableDictionary dictionaryWithDictionary:[super defaultPropertyValues]];
    NSString *uuid = [[NSUUID UUID] UUIDString];
    [defaultPropertyValues setValue:@"" forKey:@"metadata"];
    [defaultPropertyValues setValue:uuid forKey:@"uuid"];
    return defaultPropertyValues;
}

+ (NSArray *)ignoredProperties
{
    return @[];
}

@end
```

Chapter 120: Realm

Section 120.1: RLMObject Base Model Class with Primary Key - Objective-C

An example of a RLMObject base model class that uses a primary key and some generic default properties.
Subclasses can then set metadata specific to their needs.

```
@interface BaseModel : RLMObject

@property NSString *uuid;
@property NSString *metadata;

@end

@implementation BaseModel

+ (NSString *)primaryKey
{
    return @"uuid";
}

+ (NSDictionary *)defaultPropertyValues
{
    NSMutableDictionary *defaultPropertyValues = [NSMutableDictionary dictionaryWithDictionary:[super defaultPropertyValues]];
    NSString *uuid = [[NSUUID UUID] UUIDString];
    [defaultPropertyValues setValue:@"" forKey:@"metadata"];
    [defaultPropertyValues setValue:uuid forKey:@"uuid"];
    return defaultPropertyValues;
}

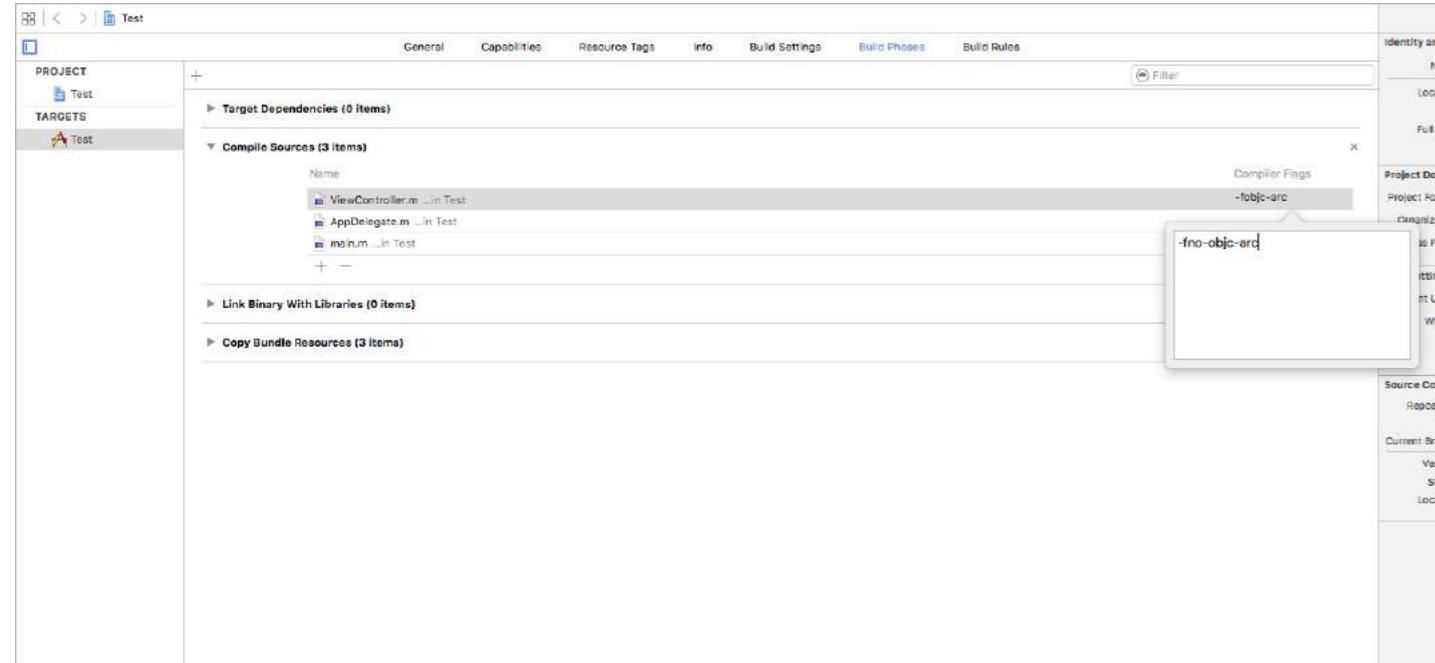
+ (NSArray *)ignoredProperties
{
    return @[];
}

@end
```

第121章：ARC（自动引用计数）

第121.1节：在文件上启用/禁用ARC

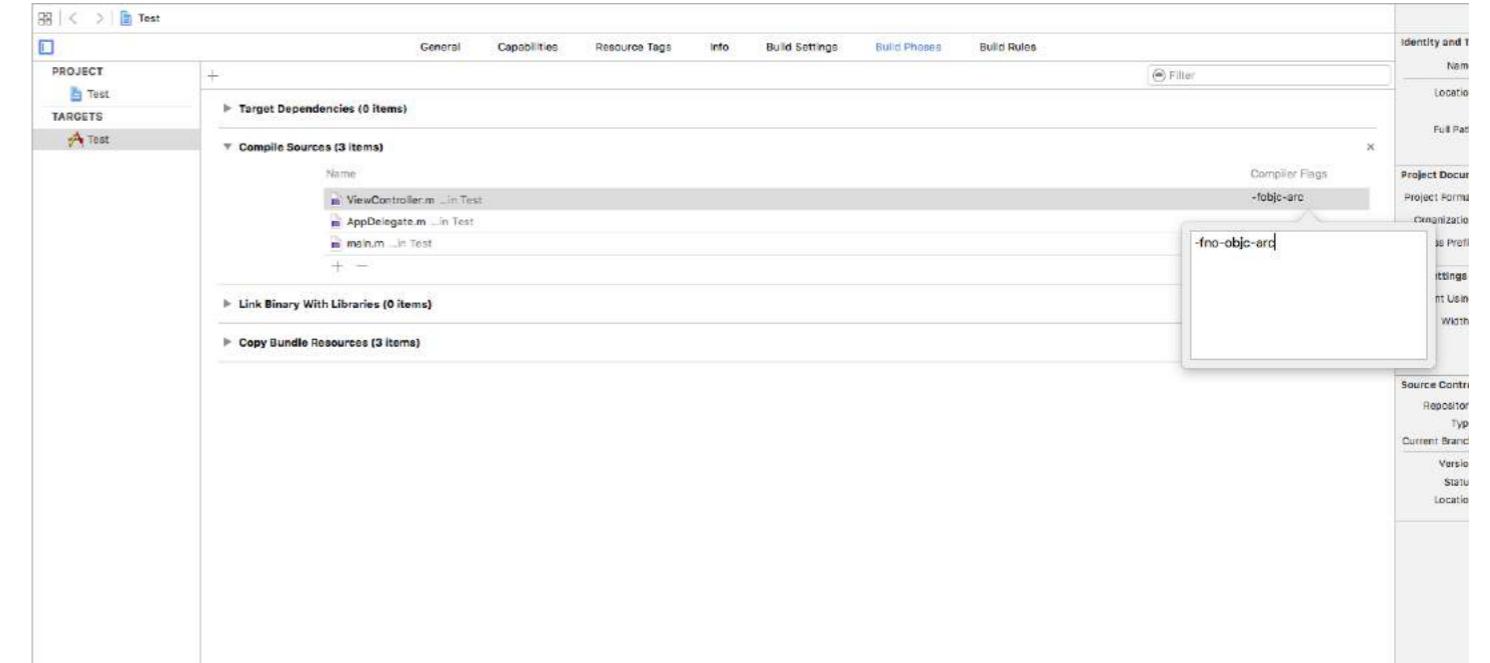
可以通过为每个文件添加-fno-objc-arc编译器标志来禁用单个文件的ARC。相反，也可以在Targets>Build Phases>Compile Sources中添加。



Chapter 121: ARC (Automatic Reference Counting)

Section 121.1: Enable/Disable ARC on a file

ARC can be disabled for individual files by adding the `-fno-objc-arc` compiler flag for each file. Conversely it can be added in Targets > Build Phases > Compile Sources



第122章：动态字体

第122.1节：在WKWebView中匹配动态字体大小

WKWebView会调整网页内容的字体大小，以使完整的网页适应设备的外形尺寸。如果你希望网页文本在纵向和横向模式下的大小都与用户偏好的阅读大小相似，则需要显式设置。

Swift

```
// 构建用于动态字体和响应式设计的HTML头部
func buildHTMLHeader() -> String {

    // 获取HTML样式的首选动态字体大小
    let bodySize = UIFont.preferredFont(forTextStyle: UIFontTextStyle.body).pointSize
    let h1Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title1).pointSize
    let h2Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title2).pointSize
    let h3Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title3).pointSize

    // 在iPad上，横屏文本比首选字体大小更大
    var portraitMultiplier = CGFloat(1.0)
    var landscapeMultiplier = CGFloat(0.5)

    // iPhone上的文本被缩小
    if UIDevice.current.model.range(of: "iPhone") != nil {
        portraitMultiplier = CGFloat(3.0)
        landscapeMultiplier = CGFloat(1.5)
    }

    // 开始HTML头部文本
    let patternText = "<html> <head> <style>"

    // 匹配本页的动态字体。
    + "body { background-color: \$(backgroundColor); }"
    + "@media all and (orientation:portrait) {img {max-width: 90%; height: auto;}"
    + "p, li { font: -apple-system-body; font-family: Georgia, serif; font-size:calc(\$(bodySize * portraitMultiplier)px + 1.0vw); font-weight: normal; color: \$(fontColor) }"
    + "h1 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h1Size * portraitMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    + "h2 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h2Size * portraitMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    + "h3, h4 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h3Size * portraitMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    "
    + "@media all and (orientation:landscape) {img {max-width: 65%; height: auto;}"
    + "p, li { font: -apple-system-body; font-family: Georgia, serif; font-size:calc(\$(bodySize * landscapeMultiplier)px + 1.0vw); font-weight: normal; color: \$(fontColor) }"
    + "h1 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h1Size * landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    + "h2 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h2Size * landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    + "h3, h4 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h3Size * landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    } </style>"
    + "</head><body>"
    + "<meta name=\"viewport\" content=\"width: device-width\>"

    return patternText
}
```

Chapter 122: Dynamic Type

Section 122.1: Matching Dynamic Type Font Size in WKWebView

WKWebView resizes the fonts on web content so that a full-sized web page will fit on the device's form factor. If you want the web text in both portrait and landscape to be similar in size to the user's preferred reading size, you need to set it explicitly.

Swift

```
// build HTML header for dynamic type and responsive design
func buildHTMLHeader() -> String {

    // Get preferred dynamic type font sizes for html styles
    let bodySize = UIFont.preferredFont(forTextStyle: UIFontTextStyle.body).pointSize
    let h1Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title1).pointSize
    let h2Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title2).pointSize
    let h3Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title3).pointSize

    // On iPad, landscape text is larger than preferred font size
    var portraitMultiplier = CGFloat(1.0)
    var landscapeMultiplier = CGFloat(0.5)

    // iPhone text is shrunk
    if UIDevice.current.model.range(of: "iPhone") != nil {
        portraitMultiplier = CGFloat(3.0)
        landscapeMultiplier = CGFloat(1.5)
    }

    // Start HTML header text
    let patternText = "<html> <head> <style>"

    // Match Dynamic Type for this page.
    + "body { background-color: \$(backgroundColor); }"
    + "@media all and (orientation:portrait) {img {max-width: 90%; height: auto;}"
    + "p, li { font: -apple-system-body; font-family: Georgia, serif; font-size:calc(\$(bodySize * portraitMultiplier)px + 1.0vw); font-weight: normal; color: \$(fontColor) }"
    + "h1 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h1Size * portraitMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    + "h2 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h2Size * portraitMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    + "h3, h4 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h3Size * portraitMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    "
    + "@media all and (orientation:landscape) {img {max-width: 65%; height: auto;}"
    + "p, li { font: -apple-system-body; font-family: Georgia, serif; font-size:calc(\$(bodySize * landscapeMultiplier)px + 1.0vw); font-weight: normal; color: \$(fontColor) }"
    + "h1 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h1Size * landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    + "h2 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h2Size * landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    + "h3, h4 { font: -apple-system-headline; font-family: Verdana, sans-serif; font-size:calc(\$(h3Size * landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \$(headFontColor) }"
    } </style>"
    + "</head><body>"
    + "<meta name=\"viewport\" content=\"width: device-width\>"

    return patternText
}
```

}

第122.2节：获取当前内容大小

Swift

```
UIApplication.sharedApplication().preferredContentSizeCategory
```

Objective-C

```
[UIApplication sharedApplication].preferredContentSizeCategory;
```

这将返回一个内容大小类别常量，或一个辅助功能内容大小类别常量。

第122.3节：在iOS 10上无通知情况下处理首选文本大小变化

UILabel, UITextField, & UITextView 类从iOS 10开始新增了一个属性，用于在用户更改首选阅读大小时自动调整字体，名为 `adjustsFontForContentSizeCategory`。

Swift

```
@IBOutlet var label:UILabel!  
  
if #available(iOS 10.0, *) {  
    label.adjustsFontForContentSizeCategory = true  
} else {  
    // 监听 UIContentSizeCategoryDidChangeNotification 并手动处理  
    // 因为 adjustsFontForContentSizeCategory 属性不可用。  
}
```

第122.4节：文本大小变化通知

您可以注册设备文本大小更改的通知。

Swift

```
NSNotificationCenter.defaultCenter().addObserver(self, selector: #selector(updateFont), name:  
name:UIContentSizeCategoryDidChangeNotification, object: nil)
```

Objective-C

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(updateFont)  
name:UIContentSizeCategoryDidChangeNotification object:nil];
```

通知的userInfo对象包含了新大小，键为UIContentSizeCategoryNewValueKey。

}

Section 122.2: Get the Current Content Size

Swift

```
UIApplication.sharedApplication().preferredContentSizeCategory
```

Objective-C

```
[UIApplication sharedApplication].preferredContentSizeCategory;
```

This returns a content size category constant, or an accessibility content size category constant.

Section 122.3: Handling Preferred Text Size Change Without Notifications on iOS 10

UILabel, UITextField, & UITextView classes have a new property starting from iOS 10 for automatically resizing their font when a user changes their preferred reading size named `adjustsFontForContentSizeCategory`.

Swift

```
@IBOutlet var label:UILabel!  
  
if #available(iOS 10.0, *) {  
    label.adjustsFontForContentSizeCategory = true  
} else {  
    // Observe for UIContentSizeCategoryDidChangeNotification and handle it manually  
    // since the adjustsFontForContentSizeCategory property isn't available.  
}
```

Section 122.4: Text Size Change Notification

You can register for notifications of when the device text size is changed.

Swift

```
NSNotificationCenter.defaultCenter().addObserver(self, selector: #selector(updateFont), name:  
name:UIContentSizeCategoryDidChangeNotification, object: nil)
```

Objective-C

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(updateFont)  
name:UIContentSizeCategoryDidChangeNotification object:nil];
```

The notification userInfo object contains the new size under UIContentSizeCategoryNewValueKey.

第123章：SWRevealViewController

第123.1节：使用

SWRevealViewController 设置基本应用

使用 Swift 语言创建一个带有单视图应用模板的基础应用程序

添加 SWRevealViewController.h 和 SWRevealViewController.m

然后点击创建桥接头文件按钮

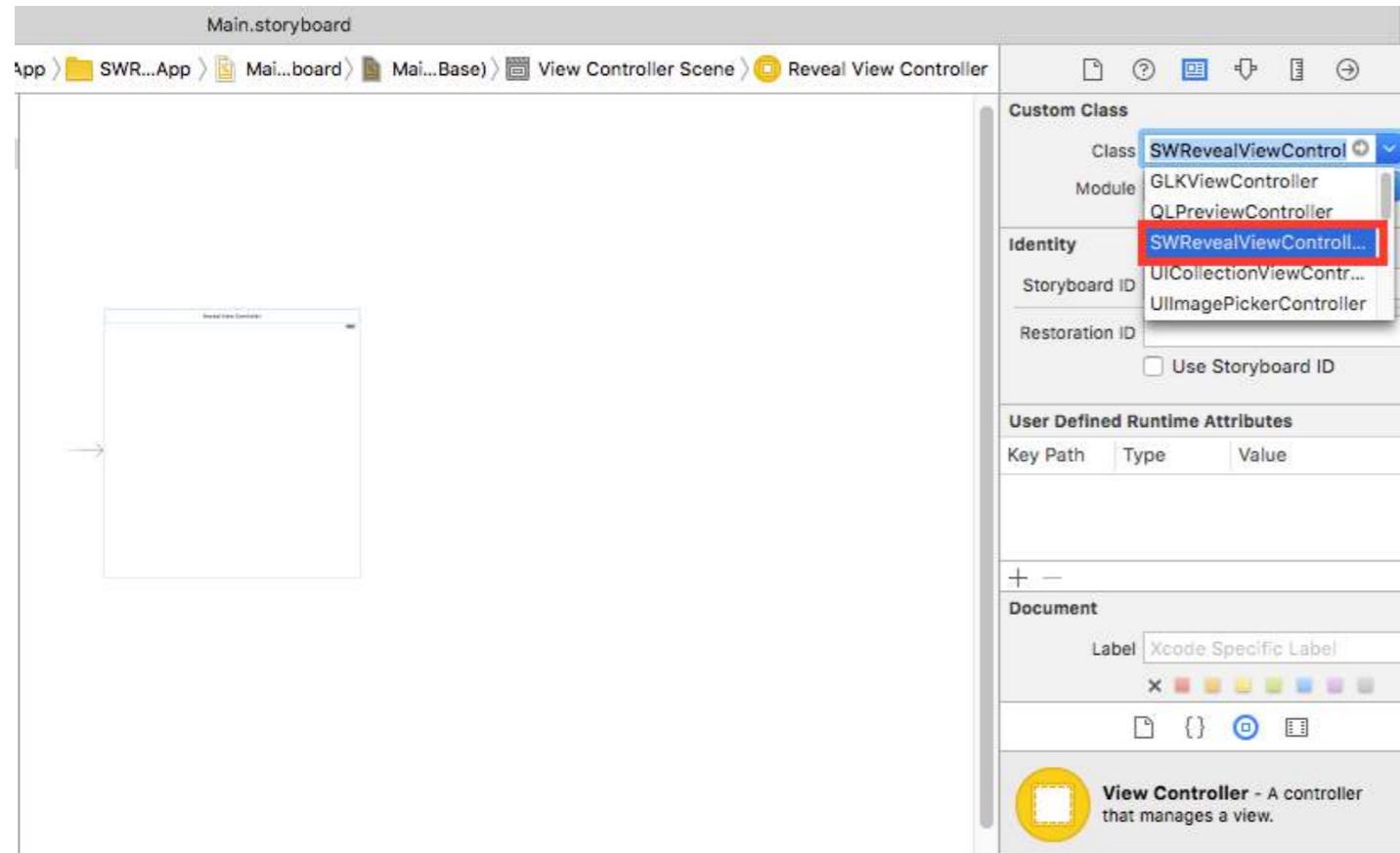


并添加

```
#import "SWRevealViewController.h"
```

到桥接头文件中

然后在故事板中选择viewController并将类改为SWRevealViewController



然后将文件中的viewController重命名为MainViewController，并添加一个名为
RightViewController的新ViewController

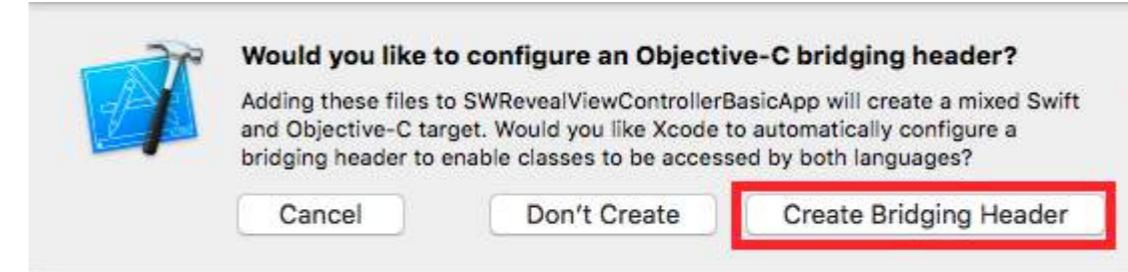
Chapter 123: SWRevealViewController

Section 123.1: Setting up a basic app with SWRevealViewController

Create a basic application with single view application template with swift as language

Add SWRevealViewController.h and SWRevealViewController.m

then click on Create Bridging Header button

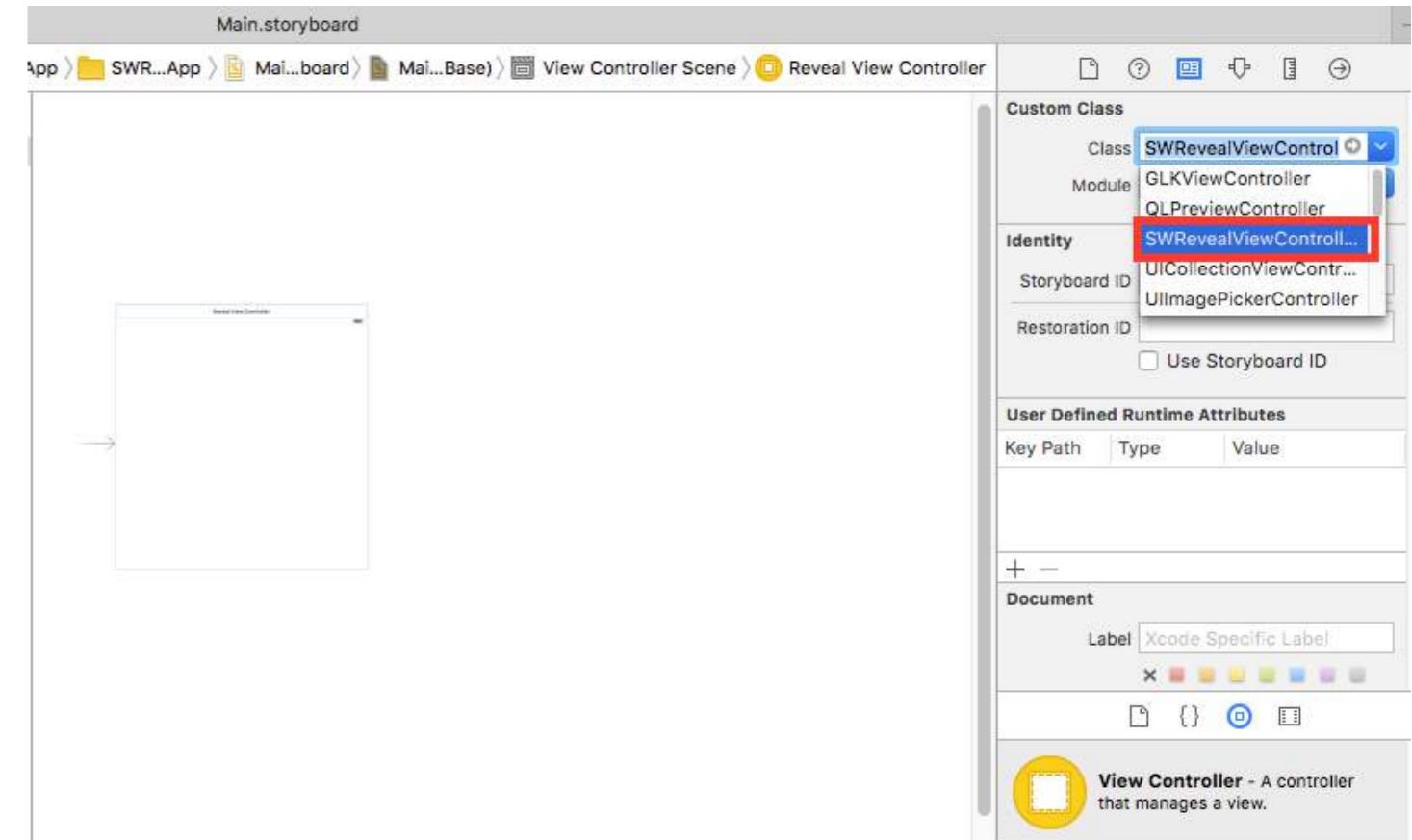


and add

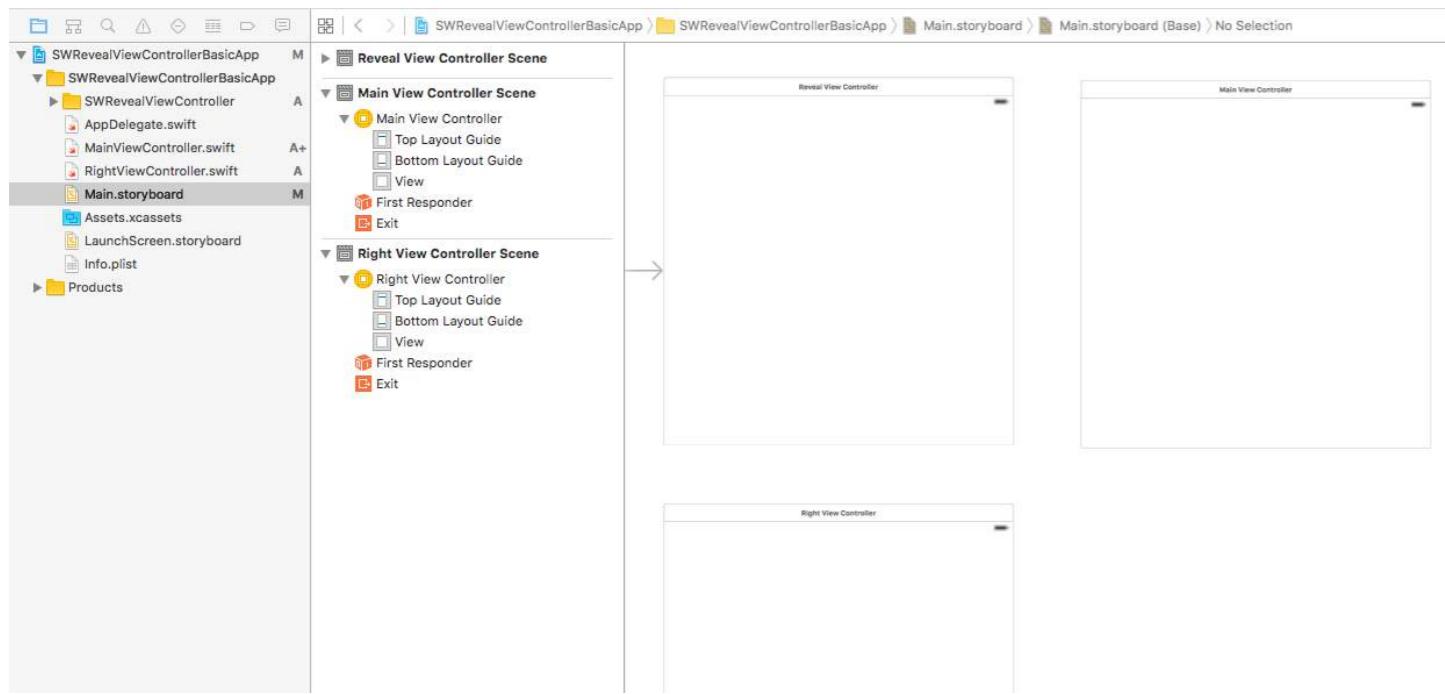
```
#import "SWRevealViewController.h"
```

on the Bridging header

Then select viewController on storyboard and change class to SWRevealViewController

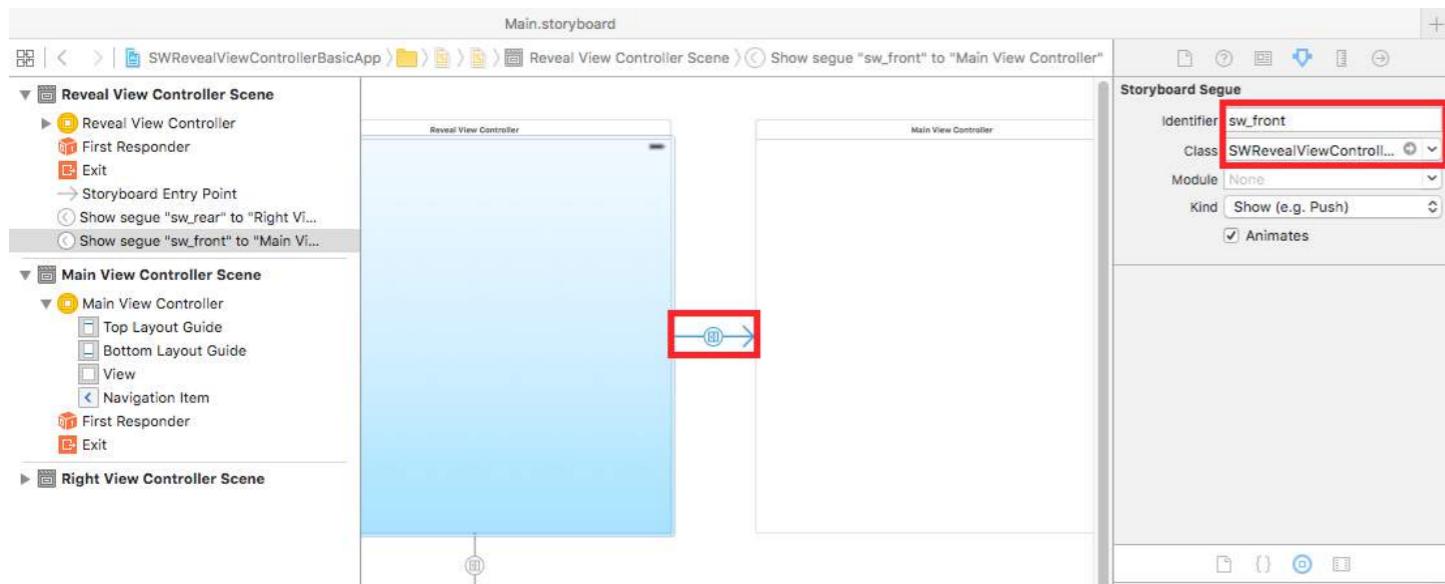


Then rename the viewController on files to MainViewController and add new ViewController with
RightViewController name



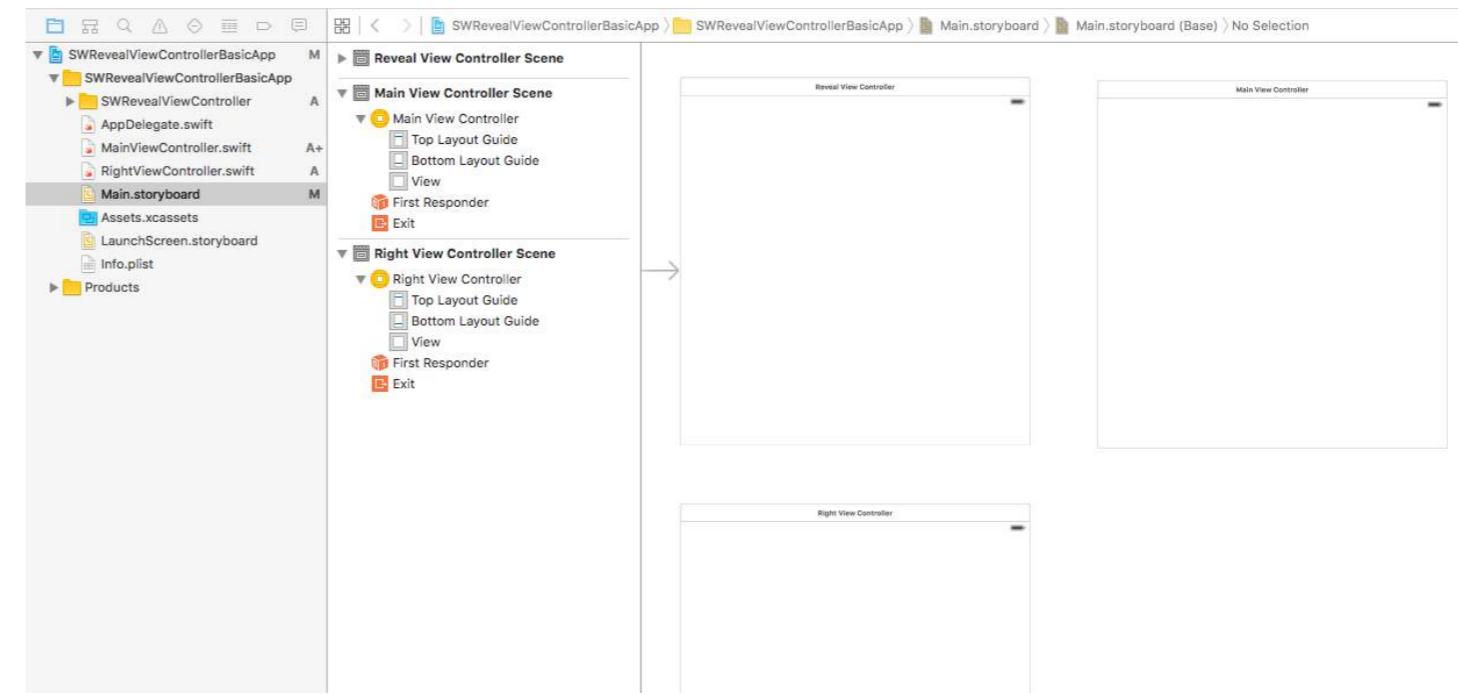
接着我们从SWRevealViewController分别添加两个segue到MainViewController和RightViewController，然后需要选择第一个（从SWRevealViewController到MainViewController）并编辑属性

在标识符中设置 sw_front 在类中设置 SWRevealViewControllerSegueSetController



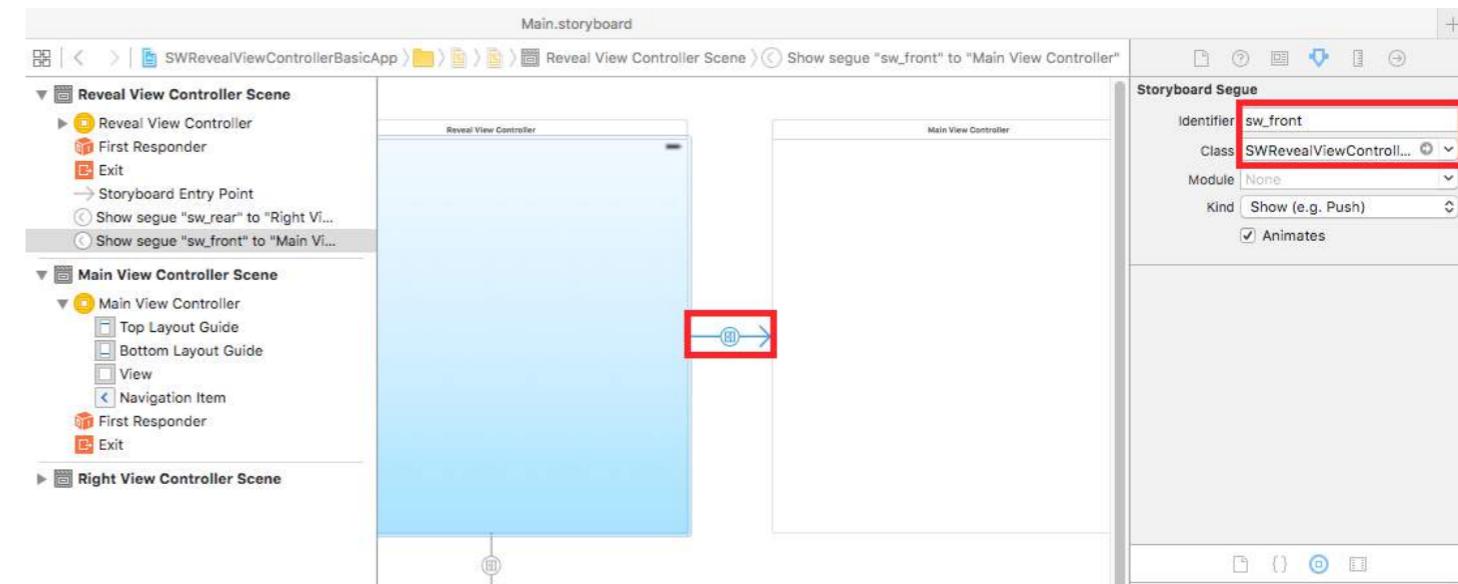
之后我们需要对segue（从SWRevealViewController到RightViewController）做同样操作

在标识符中设置 sw_rear 在类中设置 SWRevealViewControllerSegueSetController



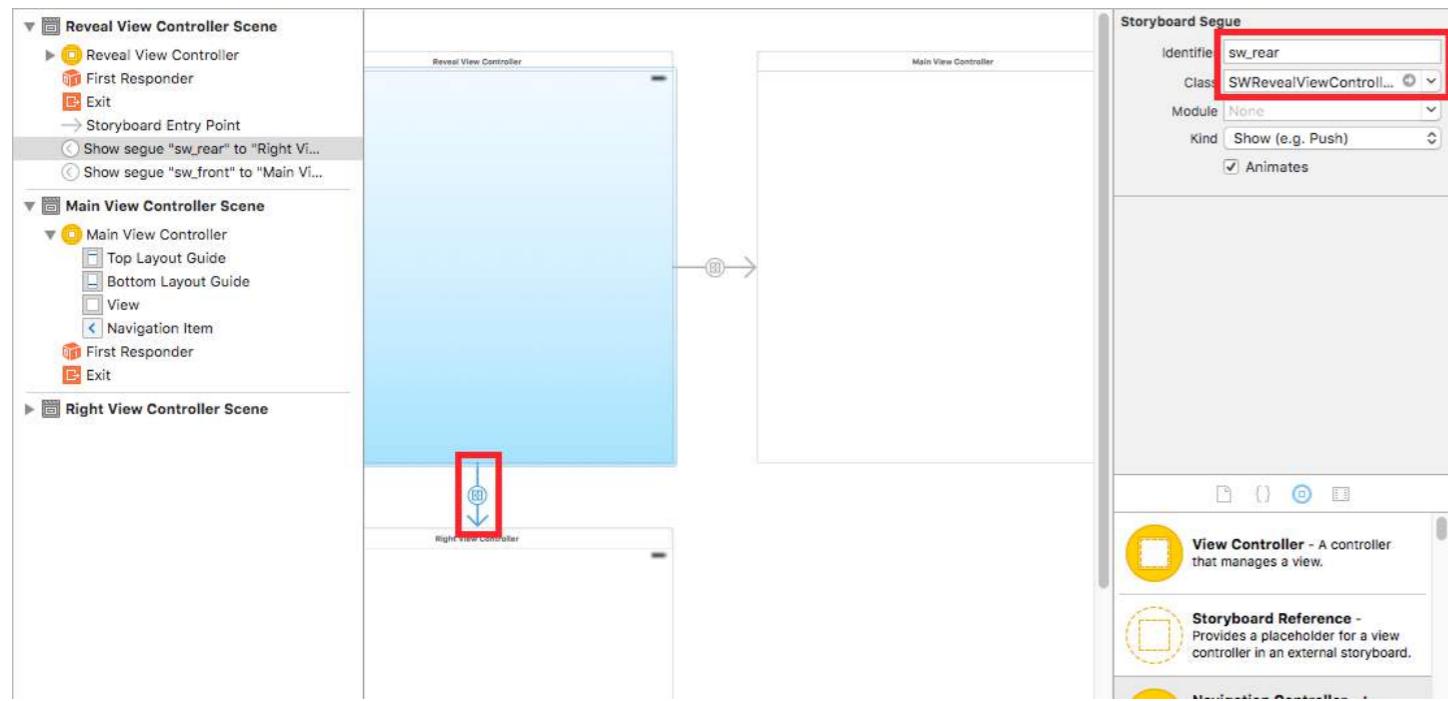
then we add two segues from SWRevealViewController to MainViewController and from SWRevealViewController to RightViewController, then we need to select the first (from SWRevealViewController to MainViewController) and edit properties

on identifier set sw_front on Class set SWRevealViewControllerSegueSetController



after this we need to do the same with the segue (from SWRevealViewController to RightViewController)

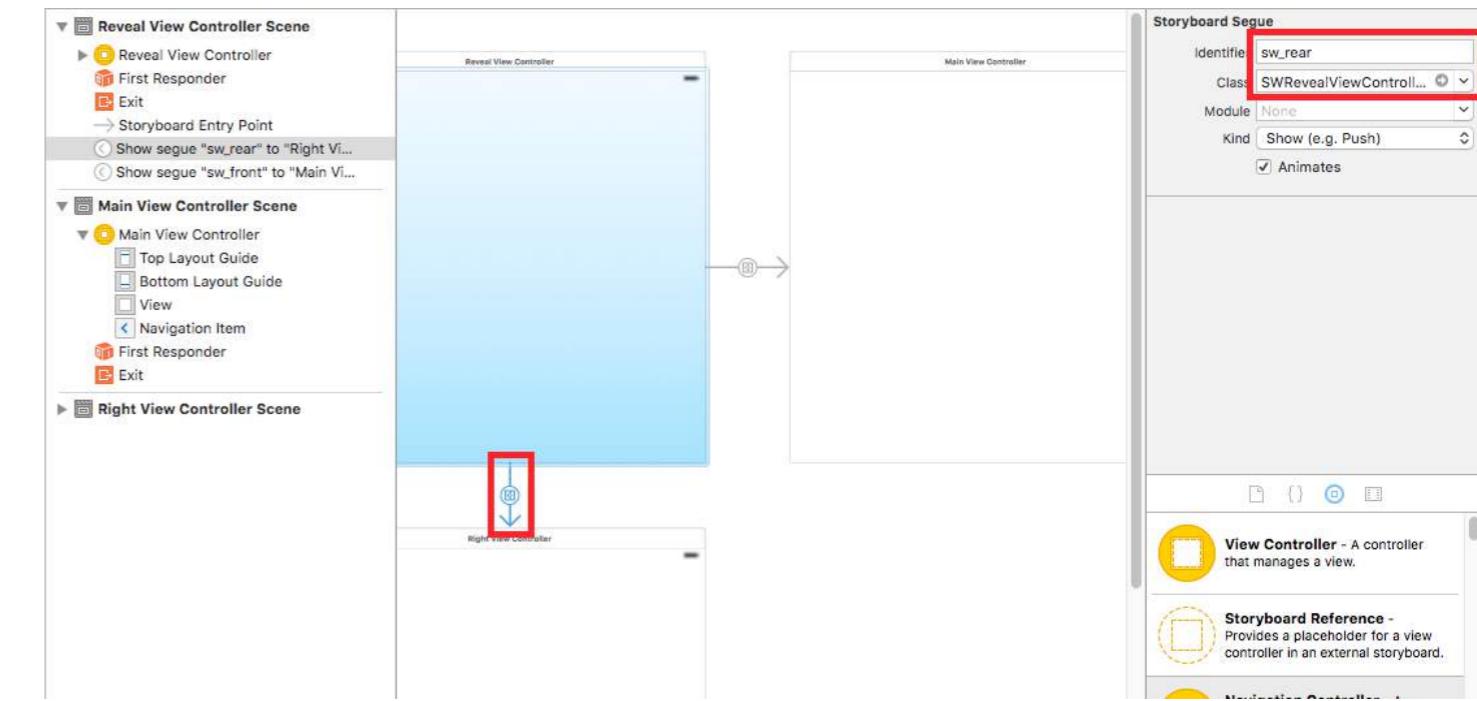
on identifier set sw_rear on Class set SWRevealViewControllerSegueSetController



然后在 MainViewController 的 viewDidLoad 方法中添加这一行

```
self.view.addGestureRecognizer(self.revealViewController().panGestureRecognizer());
```

就是这样，你已经有了一个集成了 SWRevealViewController 的基础应用，你可以向右滑动以显示 RightViewController 作为侧边菜单



then on MainViewController add this line on viewDidLoad method

```
self.view.addGestureRecognizer(self.revealViewController().panGestureRecognizer());
```

And this is all, you have a basic app with SWRevealViewController integrated, you can swipe to right to show RightViewController as lateral menu

第124章：DispatchGroup

相关主题：

Grand Central Dispatch

并发

第124.1节：介绍

假设你有多个线程在运行。每个线程都在执行一个任务。你希望在所有任务线程完成时，主线程或另一个线程能够收到通知。

解决此类问题的最简单方法是使用DispatchGroup。

使用DispatchGroup时，对于每个请求，您需要enter该组，对于每个完成的请求，您需要leave该组。

当组内不再有请求时，您将被notify（通知）。

用法：

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let dispatchGroup = DispatchGroup() //为任务创建一个组。
        let session: URLSession = URLSession.shared

        dispatchGroup.enter() //第一个任务进入组。

        let firstTask = session.dataTask(with: URLRequest(url: URL(string: "https://stackoverflow.com")!)) { (data, response, error) in
            //处理响应..

            dispatchGroup.leave() //第一个任务离开组。
        }

        dispatchGroup.enter() //进入第二个任务的组。

        let secondTask = session.dataTask(with: URLRequest(url: URL(string: "https://google.ca")!)) { (data, response, error) in
            //处理响应..

            dispatchGroup.leave() //离开第二个任务的组。

            //当上述所有任务全部完成时，在主线程/队列上收到通知。
            dispatchGroup.notify(queue: DispatchQueue.main) {
        }
    }
}
```

Chapter 124: DispatchGroup

Related topics:

Grand Central Dispatch

Concurrency

Section 124.1: Introduction

Suppose you have multiple threads running. Each thread is doing one task. You want to get notified either on the mainThread OR another thread, when all the task-threads are completed.

The simplest solution to such a problem is a DispatchGroup.

When using a DispatchGroup, for each request, you enter the group and for each completed request, you leave the group.

When there are no longer requests in the group, you will be notify (notified).

Usage:

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let dispatchGroup = DispatchGroup() //Create a group for the tasks.
        let session: URLSession = URLSession.shared

        dispatchGroup.enter() //Enter the group for the first task.

        let firstTask = session.dataTask(with: URLRequest(url: URL(string: "https://stackoverflow.com")!)) { (data, response, error) in
            //Process Response..

            dispatchGroup.leave() //Leave the group for the first task.
        }

        dispatchGroup.enter() //Enter the group for the second task.

        let secondTask = session.dataTask(with: URLRequest(url: URL(string: "https://google.ca")!)) { (data, response, error) in
            //Process Response..

            dispatchGroup.leave() //Leave the group for the second task.
        }

        //Get notified on the main thread/queue.. when ALL of the tasks above has been completed.
        dispatchGroup.notify(queue: DispatchQueue.main) {
    }
}
```

```

    print("所有任务已完成")
}

//开始任务。
firstTask.resume()
secondTask.resume()
}

override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
}
}

```

通过上述方法，你不必无限期等待所有任务完成。你可以在所有任务开始之前显示加载器，并在所有任务完成后关闭加载器。这样，主线程不会被阻塞，代码也保持整洁。

现在假设你还想让任务有序执行，或者将它们的响应依次添加到数组中。你可以这样做：

```

import UIKit

//锁机制..
func synchronized(_ lock: AnyObject, closure: () -> Void) {
    objc_sync_enter(lock)
    closure()
    objc_sync_exit(lock)
}

class ViewController: UIViewController {

    let lock = NSObject() //用于加锁的对象。
    var responseArray = Array<Data?>() //响应数组。

    override func viewDidLoad() {
        super.viewDidLoad()

        let dispatchGroup = DispatchGroup()
        let session: URLSession = URLSession.shared

        dispatchGroup.enter() //第一个任务进入组。

        let firstTask = session.dataTask(with: URLRequest(url: URL(string: "https://stackoverflow.com")!)) { (data, response, error) in
            //处理响应..

            synchronized(self.lock, closure: { () -> Void in
                self.responseArray[0] = data ?? nil
            })
        }

        dispatchGroup.leave() //第一个任务离开组。
    }

    dispatchGroup.enter() //进入第二个任务的组。

    let secondTask = session.dataTask(with: URLRequest(url: URL(string: "https://google.ca")!))
    { (data, response, error) in

```

```

        print("Every task is complete")

    }

//Start the tasks.
firstTask.resume()
secondTask.resume()
}

override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
}
}

```

With the above, you don't have to wait infinitely until all the tasks are completed. You can display a loader BEFORE all the tasks have started and dismiss the loader AFTER all tasks are completed. This way, your main thread does not get blocked and your code remains clean.

Now suppose you also wanted the tasks to be ordered or add their responses to an array sequentially. You could do the following:

```

import UIKit

//Locking mechanism..
func synchronized(_ lock: AnyObject, closure: () -> Void) {
    objc_sync_enter(lock)
    closure()
    objc_sync_exit(lock)
}

class ViewController: UIViewController {

    let lock = NSObject() //Object to lock on.
    var responseArray = Array<Data?>() //Array of responses.

    override func viewDidLoad() {
        super.viewDidLoad()

        let dispatchGroup = DispatchGroup()
        let session: URLSession = URLSession.shared

        dispatchGroup.enter() //Enter the group for the first task.

        let firstTask = session.dataTask(with: URLRequest(url: URL(string: "https://stackoverflow.com")!)) { (data, response, error) in
            //Process Response..

            synchronized(self.lock, closure: { () -> Void in
                self.responseArray[0] = data ?? nil
            })
        }

        dispatchGroup.leave() //Leave the group for the first task.
    }

    dispatchGroup.enter() //Enter the group for the second task.

    let secondTask = session.dataTask(with: URLRequest(url: URL(string: "https://google.ca")!))
    { (data, response, error) in

```

```

//处理响应..

synchronized(self.lock, closure: { () -> Void in
    self.responseArray[1] = data ?? nil
})

dispatchGroup.leave() //离开第二个任务的组。
}

//在主线程上接收通知.. 当上述所有请求都完成时。
dispatchGroup.notify(queue: DispatchQueue.main) {

    print("所有任务已完成..")

    for i in 0..

```

注意事项

每个条目在 DispatchGroup 中必须有对应的退出。如果你在 enter 后忘记调用 leave，后果自负。任务完成时你将永远不会收到通知。

enter 的次数必须等于 leave 的次数。

```

//Process Response..

synchronized(self.lock, closure: { () -> Void in
    self.responseArray[1] = data ?? nil
})

dispatchGroup.leave() //Leave the group for the second task.
}

//Get notified on the main thread.. when ALL of the requests above has been completed.
dispatchGroup.notify(queue: DispatchQueue.main) {

    print("Every task is complete..")

    for i in 0..

```

Notes

Every entry must have an exit in a DispatchGroup. If you forget to leave after entering, you are setting yourself up. You will NEVER be notified when the tasks are completed.

The amount of enter must equal the amount of leave.

第125章：GCD (Grand Central Dispatch)

Grand Central Dispatch (GCD) 是苹果针对多线程的解决方案。它是一个轻量级框架，用于在队列中同步或异步执行任务，并在后台为你管理 CPU 线程。

相关主题：并发

第125.1节：调度信号量

DispatchSemaphore 提供了传统计数信号量的高效实现，可用于控制多个执行上下文对资源的访问。

使用信号量的场景之一是当你进行文件读写时，如果多个任务同时尝试读写文件，让每个任务按顺序等待可以提高性能，避免过度负担 I/O 控制器。

Swift 3

```
func do2TasksAtATime () {
    print("开始长时间运行的任务 (每次2个)") let sem = DispatchSemaphore(value: 2) //该信号量只允许同时运行2个任务 (资源计数)

    for i in 0...7 {
        I().async { //在后台线程运行任务
            if !sem.wait() //启动一堆任务
                sleep(2) //如果没有可用资源，则在此等待
            print("长任务 \(i) 完成！\((Date()))") //执行一些长时间任务，例如文件
            sem.signal() //通知信号量该资源现在可用
        }
    }
}
```

示例输出：(注意时间戳)

```
开始长时间运行的任务 (每次2个)
长任务0完成！2017-02-16 07:11:53 +0000
长任务1完成！2017-02-16 07:11:53 +0000
长任务2完成！2017-02-16 07:11:55 +0000
长任务3完成！2017-02-16 07:11:55 +0000
长任务4完成！2017-02-16 07:11:57 +0000
长任务5完成！2017-02-16 07:11:57 +0000
长任务6完成！2017-02-16 07:11:59 +0000
长任务7完成！2017-02-16 07:11:59 +0000
```

更多信息，请参阅[Apple Docs](#)

第125.2节：Dispatch Group (调度组)

DispatchGroup允许对工作进行整体同步。您可以使用它们提交多个不同的工作项，并跟踪它们全部完成的时间，即使它们可能在不同的队列上运行。当所有指定任务完成之前无法取得进展时，这种行为非常有用。

Chapter 125: GCD (Grand Central Dispatch)

Grand Central Dispatch (GCD) is Apple's answer to multithreading. It is a lightweight framework for performing tasks synchronously or asynchronously in queues and handles CPU threads for you behind the scenes.

Related Topic: Concurrency

Section 125.1: Dispatch Semaphore

DispatchSemaphore provides an efficient implementation of a traditional counting semaphore, which can be used to control access to a resource across multiple execution contexts.

A scenario for when to use a semaphore could be if you are doing some file reading/writing, if multiple tasks are trying to read and write from file at the same time, it could increase your performance to make each task wait its turn so as to not overburden the I/O controller.

Swift 3

```
func do2TasksAtATime () {
    print("starting long running tasks (2 at a time)")
    let sem = DispatchSemaphore(value: 2) //this semaphore only allows 2 tasks to run at the same time (the resource count)
    for i in 0...7 {
        DispatchQueue.global().async {
            sem.wait() //launch a bunch of tasks
            sleep(2) //run tasks on a background thread
            are just sleeping for a 2 seconds for demonstration purposes)
            print("long task \(i) done! \((Date()))") //wait here if no resources available
            sem.signal() //do some long task eg file access (here we
        }
    }
}
```

Example output: (notice the time stamps)

```
starting long running tasks (2 at a time)
long task 0 done! 2017-02-16 07:11:53 +0000
long task 1 done! 2017-02-16 07:11:53 +0000
long task 2 done! 2017-02-16 07:11:55 +0000
long task 3 done! 2017-02-16 07:11:55 +0000
long task 4 done! 2017-02-16 07:11:57 +0000
long task 5 done! 2017-02-16 07:11:57 +0000
long task 6 done! 2017-02-16 07:11:59 +0000
long task 7 done! 2017-02-16 07:11:59 +0000
```

For more info, refer to the [Apple Docs](#)

Section 125.2: Dispatch Group

DispatchGroup allows for aggregate synchronization of work. You can use them to submit multiple different work items and track when they all complete, even though they might run on different queues. This behavior can be helpful when progress can't be made until all of the specified tasks are complete.

一个适用场景是，如果您有多个网络服务调用都需要完成后才能继续。
例如，您需要下载多组数据，这些数据需要由某个函数进行处理。
您必须等待所有网络服务完成后，才能调用函数处理所有接收到的数据。

Swift 3

```
func doLongTasksAndWait () {
    print("starting long running tasks")
    let group = DispatchGroup()          //创建一个组，用于管理我们即将执行的一系列任务
    for i in 0...3 {                     //启动一系列任务（例如一堆需要完成后才能继续到下一个视图控制器的网络服务调用）
        group.enter()                   //通知组有任务被添加
        DispatchQueue.global().async {   //在后台线程运行任务
            sleep(arc4random() % 4)      //执行一些耗时任务，例如网络服务或数据库查询（这里仅为演示目的随机休眠一段时间）
            print("long task \(i) done!")
        }
        group.leave()                  //通知组该任务已完成
    }
    group.wait()                      //阻塞当前线程，直到上述所有任务完成（因此建议不要在主线程使用此函数）
    print("all tasks done!")
}
```

或者，如果你不想等待组内任务全部完成，而是想在所有任务完成后执行某个函数，可以使用 `notify` 函数替代 `group.wait()`

```
group.notify(queue: DispatchQueue.main) { //queue: 参数指定该代码块在哪个队列执行，如果需要更新UI，请使用主队列
    print("all tasks done!")           //当所有任务都离开组时执行该代码块
}
```

示例输出：

```
开始长时间运行的任务
长任务 0 完成！
长任务 3 完成！
长任务 1 完成！
长任务 2 完成！
所有任务完成！
```

更多信息，请参阅[Apple Docs](#)或相关主题

第125.3节：获取主队列

主队列是执行所有UI更新的调度队列，涉及UI更改的代码都放在该队列中。

您需要获取主队列，以便在异步过程（如）完成时更新UI
`NSURLSession`

主要有两种类型的队列调用：同步和异步。当你同步调用某个操作时，意味着发起该操作的线程会等待任务完成后继续。异步则表示不会等待。

Objective-C 代码

A Scenario when this could be useful is if you have multiple webservice calls that all need to finish before continuing. For example, you need to download multiple sets of data that needs to be processed by some function. You have to wait for all webservices to complete before calling the function to process all the received data.

Swift 3

```
func doLongTasksAndWait () {
    print("starting long running tasks")
    let group = DispatchGroup()          //create a group for a bunch of tasks we are about to do
    for i in 0...3 {                     //launch a bunch of tasks (eg a bunch of webservice calls
        group.enter()                   //let the group know that something is being added
        DispatchQueue.global().async {   //run tasks on a background thread
            sleep(arc4random() % 4)      //do some long task eg webservice or database lookup (here
            we are just sleeping for a random amount of time for demonstration purposes)
            print("long task \(i) done!")
        }
        group.leave()                  //let group know that the task is finished
    }
    group.wait()                      //will block whatever thread we are on here until all the
    above tasks have finished (so maybe don't use this function on your main thread)
    print("all tasks done!")
}
```

Alternatively, if you do not want to wait for the groups to finish, but instead want to run a function once all the tasks have completed, use the `notify` function in place of the `group.wait()`

```
group.notify(queue: DispatchQueue.main) { //the queue: parameter is which queue this block will run
on, if you need to do UI updates, use the main queue
    print("all tasks done!")           //this will execute when all tasks have left the group
}
```

Example output:

```
starting long running tasks
long task 0 done!
long task 3 done!
long task 1 done!
long task 2 done!
all tasks done!
```

For more info, refer to the [Apple Docs](#) or the related topic

Section 125.3: Getting the Main Queue

The main queue is the dispatch queue in which all the UI updates take place and the code involving UI changes are placed.

You need to get to the main queue in order to update UI on completion of an asynchronous process like `NSURLSession`

There are two types of main queue calls synchronous and asynchronous. When you invoke something synchronously, it means that the thread that initiated that operation will wait for the task to finish before continuing. Asynchronous means that it will not wait.

Code Objective-C

同步主队列调用

```
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

异步主队列调用

```
dispatch_async(dispatch_get_main_queue(), ^{
    // 通常在这里执行工作以更新用户界面
});
```

SWIFT 3

异步主队列调用

```
DispatchQueue.main.async {
}
```

同步主队列调用

```
DispatchQueue.main.sync {
}
```

第125.4节：创建一个调度队列

你可以使用 `dispatch_queue_create`

Objective-C 创建你自己的队列

```
dispatch_queue_t queue = dispatch_queue_create("com.example.myqueue", DISPATCH_QUEUE_SERIAL);
```

Swift

```
// Swift 3 之前
let queue = dispatch_queue_create("com.example.myqueue", DISPATCH_QUEUE_SERIAL)
// Swift 3
let queue = DispatchQueue(label: "com.example.myqueue") //默认是串行队列，除非
.concurrent 被指定为属性
```

第125.5节：串行队列与并发队列

Swift 3

串行队列

```
func serialQueues () {
    let serialQueue = DispatchQueue(label: "com.example.serial") //默认队列类型是串行
队列
    let start = Date ()
    for i in 0...3 {
        //启动一堆任务
        //在后台线程上运行任务，使用我们的串行队列
        serialQueue.async {
            sleep(2)
            //执行一些耗时任务，例如网络服务
或数据库查询
            let timeTaken = Date().timeIntervalSince(start)
        }
    }
}
```

Synchronous Main Queue call

```
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

Asynchronous Main Queue call

```
dispatch_async(dispatch_get_main_queue(), ^{
    // do work here to Usually to update the User Interface
});
```

SWIFT 3

Asynchronous Main Queue call

```
DispatchQueue.main.async {
}
```

Synchronous Main Queue call

```
DispatchQueue.main.sync {
}
```

Section 125.4: Create a dispatch queue

You can create your own queue using `dispatch_queue_create`

Objective-C

```
dispatch_queue_t queue = dispatch_queue_create("com.example.myqueue", DISPATCH_QUEUE_SERIAL);
```

Swift

```
// Before Swift 3
let queue = dispatch_queue_create("com.example.myqueue", DISPATCH_QUEUE_SERIAL)
// Swift 3
let queue = DispatchQueue(label: "com.example.myqueue") //default is serial queue, unless
.concurrent is specified as an attribute otherwise
```

Section 125.5: Serial vs Concurrent Dispatch Queues

Swift 3

Serial Queue

```
func serialQueues () {
    let serialQueue = DispatchQueue(label: "com.example.serial") //default queue type is a serial
queue
    let start = Date ()
    for i in 0...3 {
        serialQueue.async {
            sleep(2)
            //launch a bunch of tasks
            //run tasks on a background
            thread, using our serial queue
            or database lookup
            let timeTaken = Date().timeIntervalSince(start)
        }
    }
}
```

```

        print("串行耗时任务 \(\i) 完成！总耗时: \(timeTaken)")
    }
}

```

示例输出：

```

串行耗时任务 0 完成！总耗时: 2.07241100072861
串行耗时任务 1 完成！总耗时: 4.16347700357437
串行耗时任务 2 完成！总耗时: 6.23209798336029
串行耗时任务 3 完成！总耗时: 8.30682599544525

```

并发队列

```

func concurrentQueues () {
    let concurrentQueue = DispatchQueue(label: "com.example.concurrent", attributes: .concurrent)
//显式指定队列为并发队列
    let start = Date ()
    for i in 0...3 {           //启动一组任务
        concurrentQueue.async { //在后台线程上运行任务，使用我们的并发队列
            sleep(2)           //执行一些耗时任务，例如网络服务或数据库查询
            let timeTaken = Date().timeIntervalSince(start)
            print("并发耗时任务 \(\i) 完成！总耗时: \(timeTaken)")
        }
    }
}

```

示例输出：

```

并发长任务 3 完成！总耗时: 2.07092100381851
并发长任务 0 完成！总耗时: 2.07087397575378
并发长任务 2 完成！总耗时: 2.07086700201035
并发长任务 1 完成！总耗时: 2.07089096307755

```

讨论

从上面的例子可以看出，串行队列会按照提交到队列的顺序完成每个任务。每个任务都会等待前一个任务完成后才执行。至于并发队列，每个任务不会等待队列中的其他任务，而是尽快执行；其优点是队列中的所有任务会在不同线程上同时运行，使得并发队列所需时间少于串行队列。

如果任务的执行顺序不重要，始终使用并发队列以获得最佳效率。

```

        print("serial long task \(\i) done! total time taken: \(timeTaken)")
    }
}

```

Example output:

```

serial long task 0 done! total time taken: 2.07241100072861
serial long task 1 done! total time taken: 4.16347700357437
serial long task 2 done! total time taken: 6.23209798336029
serial long task 3 done! total time taken: 8.30682599544525

```

Concurrent Queue

```

func concurrentQueues () {
    let concurrentQueue = DispatchQueue(label: "com.example.concurrent", attributes: .concurrent)
//explicitly specify the queue to be a concurrent queue
    let start = Date ()
    for i in 0...3 {           //launch a bunch of tasks
        concurrentQueue.async { //run tasks on a background thread, using our concurrent queue
            sleep(2)           //do some long task eg webservice or database lookup
            let timeTaken = Date().timeIntervalSince(start)
            print("concurrent long task \(\i) done! total time taken: \(timeTaken)")
        }
    }
}

```

Example output:

```

concurrent long task 3 done! total time taken: 2.07092100381851
concurrent long task 0 done! total time taken: 2.07087397575378
concurrent long task 2 done! total time taken: 2.07086700201035
concurrent long task 1 done! total time taken: 2.07089096307755

```

Discussion

As we can see from the examples above, a serial queue will complete each task in the order they are submitted to the queue. Each task will wait for the previous task to finish before executing. As for the concurrent queue, each task does not wait on the others in the queue and executes as soon as possible; the advantage is that all tasks on the queue will run at the same time on separate threads, making a concurrent queue take less time than a serial queue.

If order of execution of tasks is not important, always use a concurrent queue for the best efficiency.

第126章：尺寸类别与适应性

第126.1节：特征集合

在iOS应用中，用户界面可以呈现几种不同的一般形状和尺寸。这些通过尺寸类别定义，可以通过视图或视图控制器的特征集合获得。

苹果定义了两种尺寸类别：常规和紧凑。这两种尺寸类别在设备的两个轴线上（水平和垂直）均可用。您的应用在其生命周期中可能处于这四种状态中的任何一种。作为简写，开发者通常通过先写水平轴的尺寸类别，再写垂直轴的尺寸类别来描述尺寸类别组合：“紧凑/常规”描述的是水平紧凑但垂直常规的界面。

在您的应用中，使用UITraitEnvironment协议上的方法检查当前的尺寸类别并响应变化：

```
class MyViewController: UIViewController {
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        print("水平尺寸类别: \(traitCollection.horizontalSizeClass)")
        print("垂直尺寸类别: \(traitCollection.verticalSizeClass)")
    }

    override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
        super.traitCollectionDidChange(previousTraitCollection)
        print("特征集合已更改；尺寸类别可能不同。")
    }
}
```

UIView 和 UIViewController 都遵循 UITraitEnvironment，因此你可以查看当前的特征集合并在任一子类中处理更改。

第126.2节：使用特征集合更改更新自动布局

使应用程序自适应——即通过响应尺寸类别的变化来改变布局——通常需要大量自动布局系统的帮助。应用程序变得自适应的主要方式之一是当视图的尺寸类别发生变化时，更新活动的自动布局约束。

例如，考虑一个使用 UIStackView 来排列两个 UILabel 的应用程序。我们可能希望在水平紧凑环境中，这些标签垂直堆叠，而在水平常规环境中有多空间时，它们并排放置。

```
class ViewController: UIViewController {
    var stackView: UIStackView!

    override func viewDidLoad() {
        super.viewDidLoad()

        stackView = UIStackView()
        for text in ["foo", "bar"] {
            let label = UILabel()
            label.translatesAutoresizingMaskIntoConstraints = false
            label.text = text
            stackView.addArrangedSubview(label)
        }
    }
}
```

Chapter 126: Size Classes and Adaptivity

Section 126.1: Trait Collections

In an iOS app, your user interface can take on one of a few different general shapes and sizes. These are defined using **size classes**, which are available through a view or view controller's **trait collection**.

Apple defines two size classes: **regular** and **compact**. Each of these size classes are available on both axes of the device (**horizontal** and **vertical**). Your app may exist in any of these four states throughout its lifetime. As a shorthand, developers often describe a size class combination by saying or writing the two size classes, with the horizontal axis first: "Compact/Regular" describes an interface that is horizontally compact but vertically regular.

In your app, use methods on the UITraitEnvironment protocol to check your current size class and respond to changes:

```
class MyViewController: UIViewController {
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        print("Horizontal size class: \(traitCollection.horizontalSizeClass)")
        print("Vertical size class: \(traitCollection.verticalSizeClass)")
    }

    override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
        super.traitCollectionDidChange(previousTraitCollection)
        print("Trait collection changed; size classes may be different.")
    }
}
```

Both UIView and UIViewController conform to UITraitEnvironment, so you can look at your current trait collection and handle changes in subclasses of either.

Section 126.2: Updating Auto Layout with Trait Collection Changes

Making an app **adaptive** – that is, responding to size class changes by changing your layout – often involves lots of help from the Auto Layout system. One of the primary ways apps become adaptive is by updating the active Auto Layout constraints when a view's size class changes.

For example, consider an app that uses a UIStackView to arrange two UILabels. We might want these labels to stack on top of each other in horizontally compact environments, but sit next to each other when we have a little more room in horizontally regular environments.

```
class ViewController: UIViewController {
    var stackView: UIStackView!

    override func viewDidLoad() {
        super.viewDidLoad()

        stackView = UIStackView()
        for text in ["foo", "bar"] {
            let label = UILabel()
            label.translatesAutoresizingMaskIntoConstraints = false
            label.text = text
            stackView.addArrangedSubview(label)
        }
    }
}
```

```

view.addSubview(stackView)
stackView.translatesAutoresizingMaskIntoConstraints = false
    stackView.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true
    stackView.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive = true
}

override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
updateAxis(forTraitCollection: traitCollection)
}

override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)
updateAxis(forTraitCollection: traitCollection)
}

private func updateAxis(forTraitCollection traitCollection: UITraitCollection) {
    switch traitCollection.horizontalSizeClass {
        case .regular:
            stackView.axis = .horizontal
        case .compact:
            stackView.axis = .vertical
        case .unspecified:
            print("未指定的尺寸类别！")
            stackView.axis = .horizontal
    }
}
}

```

```

view.addSubview(stackView)
stackView.translatesAutoresizingMaskIntoConstraints = false
    stackView.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true
    stackView.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive = true
}

override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
updateAxis(forTraitCollection: traitCollection)
}

override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)
updateAxis(forTraitCollection: traitCollection)
}

private func updateAxis(forTraitCollection traitCollection: UITraitCollection) {
    switch traitCollection.horizontalSizeClass {
        case .regular:
            stackView.axis = .horizontal
        case .compact:
            stackView.axis = .vertical
        case .unspecified:
            print("Unspecified size class!")
            stackView.axis = .horizontal
    }
}
}

```

第126.3节：支持iPad上的iOS多任务处理

现代iOS应用适应性的关键部分是支持iPad上的多任务处理。默认情况下，Xcode 7及更高版本创建的应用将配置为支持多任务处理：它们将拥有一个使用自动布局（Auto Layout）的LaunchScreen.storyboard文件。

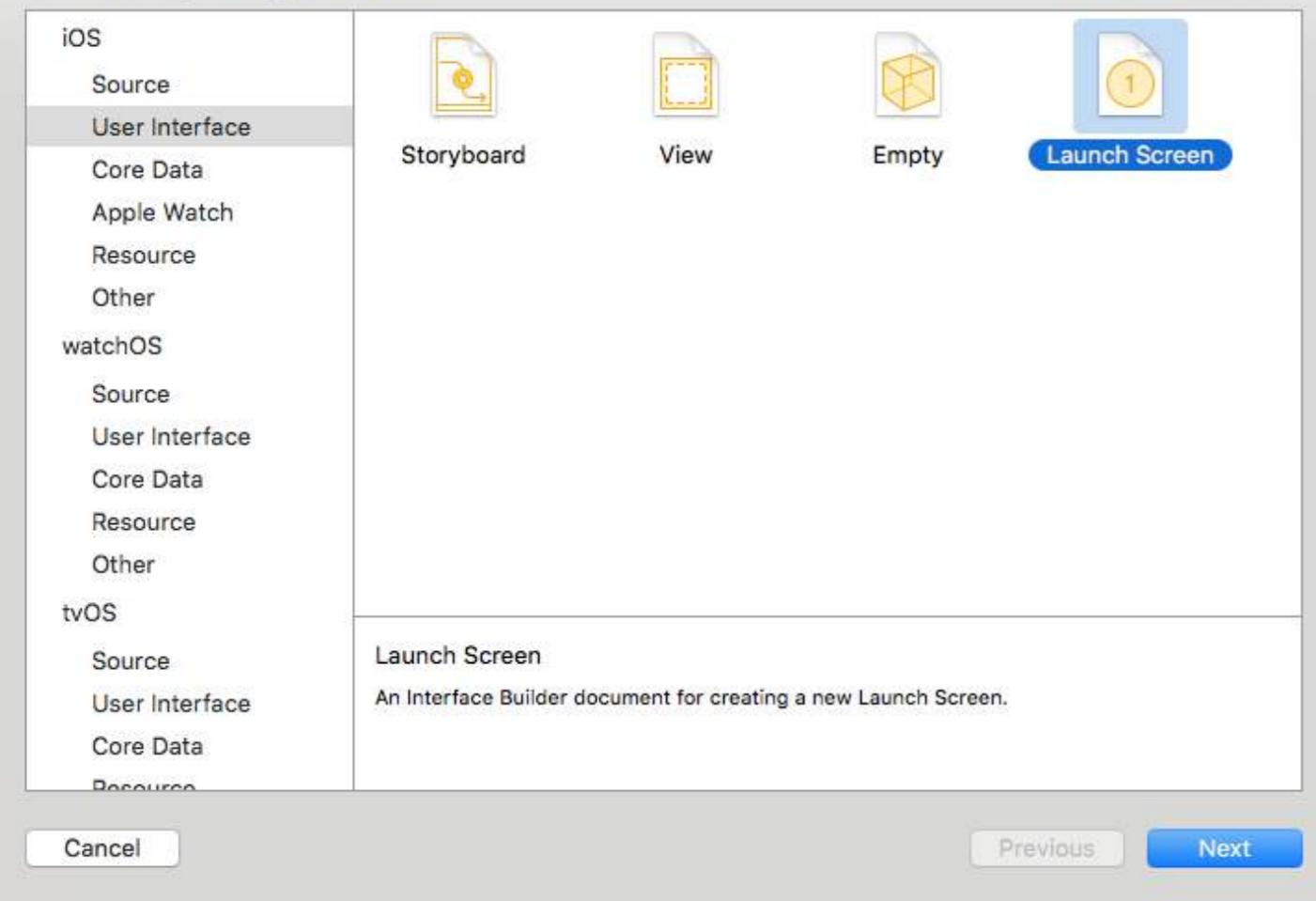
现有应用启用多任务处理的最简单方法是创建这样的storyboard，然后将其设置为项目的启动屏幕（Launch Screen）：

Section 126.3: Supporting iOS Multitasking on iPad

A key piece of adaptivity in a modern iOS app is supporting multitasking on iPad. By default, apps created in Xcode 7 and newer will be configured to support multitasking: they'll have a LaunchScreen.storyboard file that uses Auto Layout.

The easiest way for existing apps to opt in to multitasking is to create such a storyboard, then set it as the project's Launch Screen:

Choose a template for your new file:



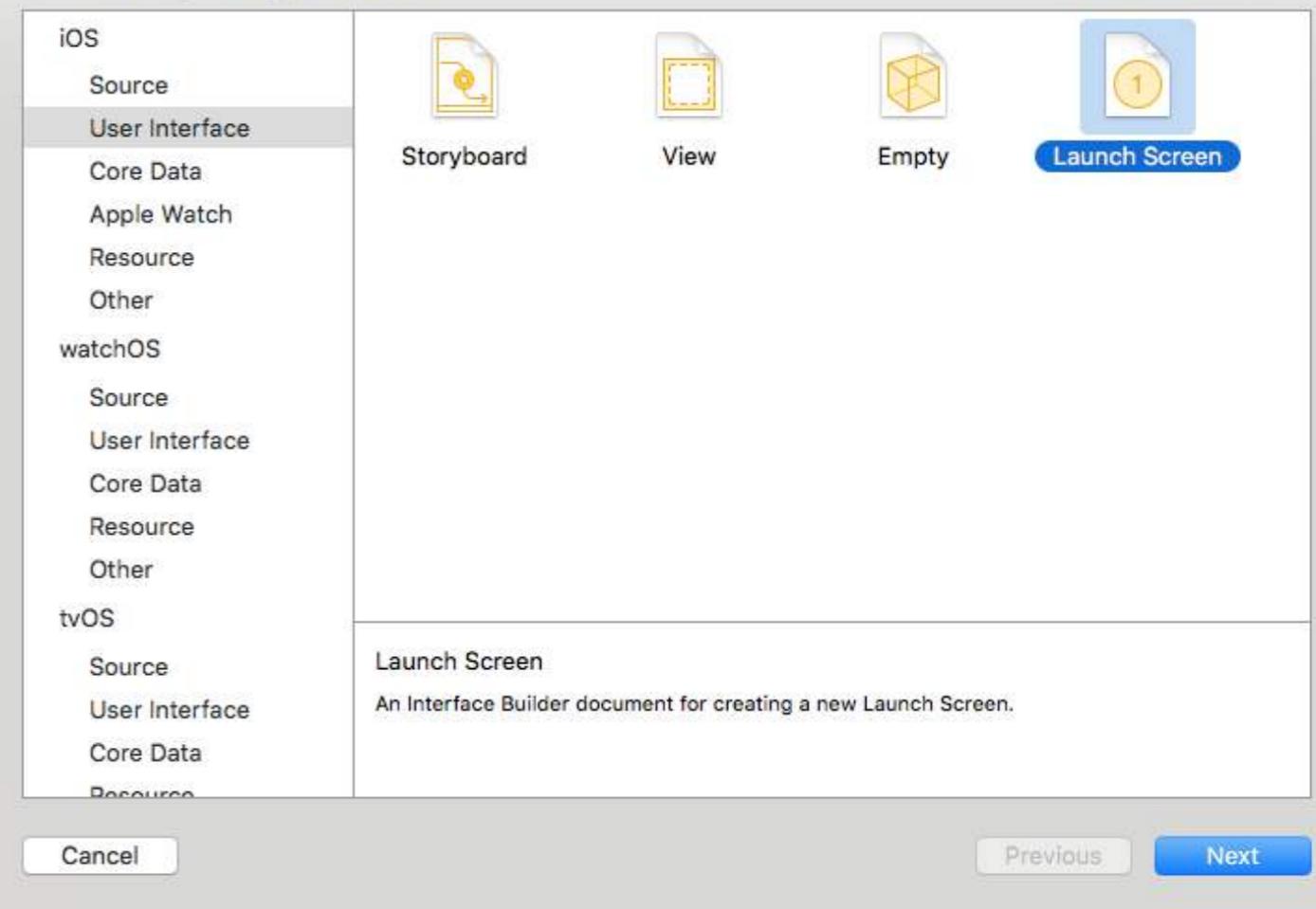
App Icons Source AppIcon

Launch Images Source Use Asset Catalog

Launch Screen File LaunchScreen

一旦您的应用支持iPad多任务处理，请审核现有的视图和视图控制器，确保它们使用自动布局并能支持各种尺寸类别组合。

Choose a template for your new file:



App Icons Source AppIcon

Launch Images Source Use Asset Catalog

Launch Screen File LaunchScreen

Once your app supports iPad multitasking, audit existing views and view controllers to make sure that they use Auto Layout and can support a variety of size class combinations.

第127章：IBOutlets

第127.1节：在UI元素中使用IBOutlet

一般来说，IBOutlet 用于将用户界面对象连接到另一个对象，在本例中是 UIViewController。该连接用于允许通过代码或事件以编程方式影响该对象。这可以通过使用故事板的助手，按住 Control 键从元素拖动到视图控制器的 .h 属性部分来简单完成，但也可以通过编程方式手动将 IBOutlet 代码连接到右侧实用工具栏中对象的“connections”标签来完成。以下是一个带有标签 outlet 的 UIViewController 的 Objective-C 示例：

```
//ViewController.h
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

//这是 outlet 的声明
@property (nonatomic, weak) IBOutlet UILabel *myLabel;

@end

//ViewController.m
#import "ViewController.h"

@implementation ViewController

@synthesize myLabel;

-(void) viewDidLoad {
    [super viewDidLoad];
    //编辑 outlet 的属性
    myLabel.text = @"TextHere";
}

@end
```

Swift 版本：

```
import UIKit
class ViewController: UIViewController {
    //这是 outlet 的声明
    @IBOutlet weak var myLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        //编辑出口的属性
        myLabel.text = "TextHere"
    }
}
```

如果.h文件中outlet声明左侧的点被填充，则可以验证storyboard对象与编程对象之间已连接。空心圆表示连接不完整。

Chapter 127: IBOutlets

Section 127.1: Using an IBOutlet in a UI Element

In general, IBOutlets are used to connect an user interface object to another object, in this case a UIViewController. The connection serves to allow for the object to be affected by code or events programmatically. This can be done simply by using the assistant from a storyboard and control-clicking from the element to the view controller's .h property section, but it can also be done programmatically and manually connecting the IBOutlet code to the "connections" tab of the object the utility bar on the right. Here is an objective-c example of a UIViewController with a label outlet:

```
//ViewController.h
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

//This is the declaration of the outlet
@property (nonatomic, weak) IBOutlet UILabel *myLabel;

@end

//ViewController.m
#import "ViewController.h"

@implementation ViewController

@synthesize myLabel;

-(void) viewDidLoad {
    [super viewDidLoad];
    //Editing the properties of the outlet
    myLabel.text = @"TextHere";
}

@end
```

And swift:

```
import UIKit
class ViewController: UIViewController {
    //This is the declaration of the outlet
    @IBOutlet weak var myLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        //Editing the properties of the outlet
        myLabel.text = "TextHere"
    }
}
```

The connection between the storyboard object, and the programmed object can be verified as connected if the dot to the left of the declaration of the outlet in the .h is filled. An empty circle implied an incomplete connection.

第128章：AWS SDK

第128.1节：使用AWS SDK上传图片或视频到S3

在开始示例之前，我建议创建一个带有代理类成员的单例，这样你可以实现后台上传文件的用例，让用户在文件上传时继续使用你的应用，即使应用处于后台。

首先，我们应该创建一个表示S3配置的枚举：

```
enum S3Configuration : String
{
    case IDENTITY_POOL_ID = "YourIdentityPoolId"
    case BUCKET_NAME = "YourBucketName"
    case CALLBACK_KEY = "YourCustomStringForCallBackWhenUploadingInTheBackground"
    case CONTENT_TYPE_IMAGE = "image/png"
    case CONTENT_TYPE_VIDEO = "video/mp4"
}
```

现在，我们应该在应用首次启动时设置凭证，因此，我们应将它们设置在
在 AppDelegate 的 didFinishLaunchingWithOptions 方法中（注意你应该在 regionType 参数中设置你的区域）：

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool
{
    let credentialProvider = AWSCognitoCredentialsProvider(regionType: .EUWest1, identityPoolId:
S3Configuration.IDENTITY_POOL_ID.rawValue)
    let configuration = AWSServiceConfiguration(region: .EUWest1, credentialsProvider:
credentialProvider)
    AWSS3TransferUtility.registerS3TransferUtilityWithConfiguration(configuration, forKey:
S3Configuration.CALLBACK_KEY.rawValue)
}
```

由于我们已经在 AppDelegate 中，我们应该实现由 AWS SDK 处理的后台回调：

```
func application(application: UIApplication, handleEventsForBackgroundURLSession identifier:
String, completionHandler: () -> Void)
{
    // 将打印您在枚举中设置的标识符：.CALLBACK_KEY
    print("标识符: " + identifier)
    // 存储完成处理程序。
    AWSS3TransferUtility.interceptApplication(application,
                                                handleEventsForBackgroundURLSession: identifier,
                                                completionHandler: completionHandler)
}
```

现在，当用户将应用移到后台时，您的上传将继续实际上传。

为了使用AWS SDK上传文件，我们必须将文件写入设备并提供给SDK实际的
路径。举例来说，假设我们有一个UIImage（也可以是视频..），我们将其写入一个临时
文件夹：

```
// 一些图片....
```

Chapter 128: AWS SDK

Section 128.1: Upload an image or a video to S3 using AWS SDK

Before starting with the example I'd recommend to create a Singleton with a delegate class member so you could achieve a use case of uploading a file in the background and let the user keep using your app while the files are being uploaded even when the app is the background.

Let's start, first, we should create an enum that represent the S3 configuration:

```
enum S3Configuration : String
{
    case IDENTITY_POOL_ID = "YourIdentityPoolId"
    case BUCKET_NAME = "YourBucketName"
    case CALLBACK_KEY = "YourCustomStringForCallBackWhenUploadingInTheBackground"
    case CONTENT_TYPE_IMAGE = "image/png"
    case CONTENT_TYPE_VIDEO = "video/mp4"
}
```

Now, we should set the credentials when your app launch for the first time, thus, we should set them inside the
AppDelegate at the didFinishLaunchingWithOptions method (pay attention that you should set your region at the
regionType param):

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool
{
    let credentialProvider = AWSCognitoCredentialsProvider(regionType: .EUWest1, identityPoolId:
S3Configuration.IDENTITY_POOL_ID.rawValue)
    let configuration = AWSServiceConfiguration(region: .EUWest1, credentialsProvider:
credentialProvider)
    AWSS3TransferUtility.registerS3TransferUtilityWithConfiguration(configuration, forKey:
S3Configuration.CALLBACK_KEY.rawValue)
}
```

Since we are already inside the AppDelegate, we should implement the background callback that is handled by the
AWS SDK:

```
func application(application: UIApplication, handleEventsForBackgroundURLSession identifier:
String, completionHandler: () -> Void)
{
    // Will print the identifier you have set at the enum: .CALLBACK_KEY
    print("Identifier: " + identifier)
    // Stores the completion handler.
    AWSS3TransferUtility.interceptApplication(application,
                                                handleEventsForBackgroundURLSession: identifier,
                                                completionHandler: completionHandler)
}
```

Now, when the user will move the app to the background your upload will continue the actual upload.

In order to upload the file using the AWS SDK we will have to write the file to the device and give the SDK the actual
path. For the sake of the example, imagine we have a UIImage (could be a video also..) and we will write it to a temp
folder:

```
// Some image....
```

```

let image = UIImage()
let fileURL = NSURL(fileURLWithPath: NSTemporaryDirectory()).URLByAppendingPathComponent(fileName)
let filePath = fileURL.path!
let imageData = UIImageJPEGRepresentation(image, 1.0)
imageData!.writeToFile(filePath, atomically: true)

```

FileURL 和 fileName 将用于后续的实际上传。

我们需要定义两个由 AWS SDK 提供的闭包，

1. AWSS3TransferUtilityUploadCompletionHandlerBlock - 一个在上传完成时（或未完成时）通知的闭包
2. AWSS3TransferUtilityUploadProgressBlock - 一个通知每个已发送字节的闭包

如果你计划使用单例模式，应该将这些类型定义为类成员。实现应如下所示：

```

var completionHandler : AWSS3TransferUtilityUploadCompletionHandlerBlock? =
{ (task, error) -> Void in

    if ((error) != nil)
    {
        print("上传失败")
    }
    else
    {
        print("文件上传成功")
    }
}

var progressBlock : AWSS3TransferUtilityUploadProgressBlock? =
{ [unowned self] (task, bytesSent:Int64, totalBytesSent:Int64, totalBytesExpectedToSend:Int64) -> Void in

    let progressInPercentage = Float(Double(totalBytesSent) / Double(totalBytesExpectedToSend)) * 100
    print(progressInPercentage)
}

```

注意：如果您使用的是单例模式，您可能需要定义一个代理，用于报告进度或文件完成情况。

如果您不使用单例模式，可以创建一个静态方法，包含相关类型：

```

static func uploadImageToS3(fileURL : NSURL,
                            fileName : String,
                            progressFunctionUpdater : Float -> Void,
                            resultBlock : (NSError?) -> Void)
{
    // 实际实现 ....
    // ...
    // ...
}

```

1. progressFunctionUpdater - 将通过函数回调报告进度。
2. resultBlock - 如果返回nil，则上传成功，否则返回错误对象。

女士们，先生们，实际上传开始：

```
let fileData = NSData(contentsOfFile: fileURL.relativePath!)
```

```

let image = UIImage()
let fileURL = NSURL(fileURLWithPath: NSTemporaryDirectory()).URLByAppendingPathComponent(fileName)
let filePath = fileURL.path!
let imageData = UIImageJPEGRepresentation(image, 1.0)
imageData!.writeToFile(filePath, atomically: true)

```

FileURL and fileName will be used for the actual uploading later.

There are 2 closures we will have to define that are provided by the AWS SDK,

1. AWSS3TransferUtilityUploadCompletionHandlerBlock - A closure that notifies when the upload is done (or not)
2. AWSS3TransferUtilityUploadProgressBlock - A closure that notifies each byte sent

If you plan to have a Singleton you should define those types as class members. The implementation should look like this:

```

var completionHandler : AWSS3TransferUtilityUploadCompletionHandlerBlock? =
{ (task, error) -> Void in

    if ((error) != nil)
    {
        print("Upload failed")
    }
    else
    {
        print("File uploaded successfully")
    }
}

var progressBlock : AWSS3TransferUtilityUploadProgressBlock? =
{ [unowned self] (task, bytesSent:Int64, totalBytesSent:Int64, totalBytesExpectedToSend:Int64) -> Void in

    let progressInPercentage = Float(Double(totalBytesSent) / Double(totalBytesExpectedToSend)) * 100
    print(progressInPercentage)
}

```

NOTE: If you are using a Singleton you might want to define a delegate that will report back with progress or when the file is done. If you are not using a Singleton you can create a static method that would have the relevant types:

```

static func uploadImageToS3(fileURL : NSURL,
                            fileName : String,
                            progressFunctionUpdater : Float -> Void,
                            resultBlock : (NSError?) -> Void)
{
    // Actual implementation ....
    // ...
    // ...
}

```

1. progressFunctionUpdater - will report back to a function with progress.
2. resultBlock - If you return nil then upload was successfully else, you send the error object

Ladies and gentlemen, the actual upload:

```
let fileData = NSData(contentsOfFile: fileURL.relativePath!)
```

```

let expression = AWSS3TransferUtilityUploadExpression()
expression.uploadProgress = progressBlock

let transferUtility =
AWSS3TransferUtility.S3TransferUtilityForKey(S3Configuration.Callback_KEY.rawValue)

transferUtility?.uploadData(fileData!,
bucket: S3Configuration.BUCKET_NAME.rawValue,
key: fileName,
contentType: S3Configuration.CONTENT_TYPE_IMAGE.rawValue,
expression: expression,
completionHandler: completionHandler).continueWithBlock
{ (task : AWSTask) -> AnyObject? in

    if let error = task.error
    {
        print(error)
    }
    if let exception = task.exception
    {
        print("异常: " + exception.description)
    }
    if let uploadTask = task.result as? AWSS3TransferUtilityUploadTask
    {
        print("上传开始...")
    }

    return nil
}

```

愉快的 S3 上传 :)

```

let expression = AWSS3TransferUtilityUploadExpression()
expression.uploadProgress = progressBlock

let transferUtility =
AWSS3TransferUtility.S3TransferUtilityForKey(S3Configuration.Callback_KEY.rawValue)

transferUtility?.uploadData(fileData!,
bucket: S3Configuration.BUCKET_NAME.rawValue,
key: fileName,
contentType: S3Configuration.CONTENT_TYPE_IMAGE.rawValue,
expression: expression,
completionHandler: completionHandler).continueWithBlock
{ (task : AWSTask) -> AnyObject? in

    if let error = task.error
    {
        print(error)
    }
    if let exception = task.exception
    {
        print("Exception: " + exception.description)
    }
    if let uploadTask = task.result as? AWSS3TransferUtilityUploadTask
    {
        print("Upload started...")
    }

    return nil
}

```

Happy S3 uploading :)

第129章：调试崩溃

第129.1节：调试 EXC_BAD_ACCESS

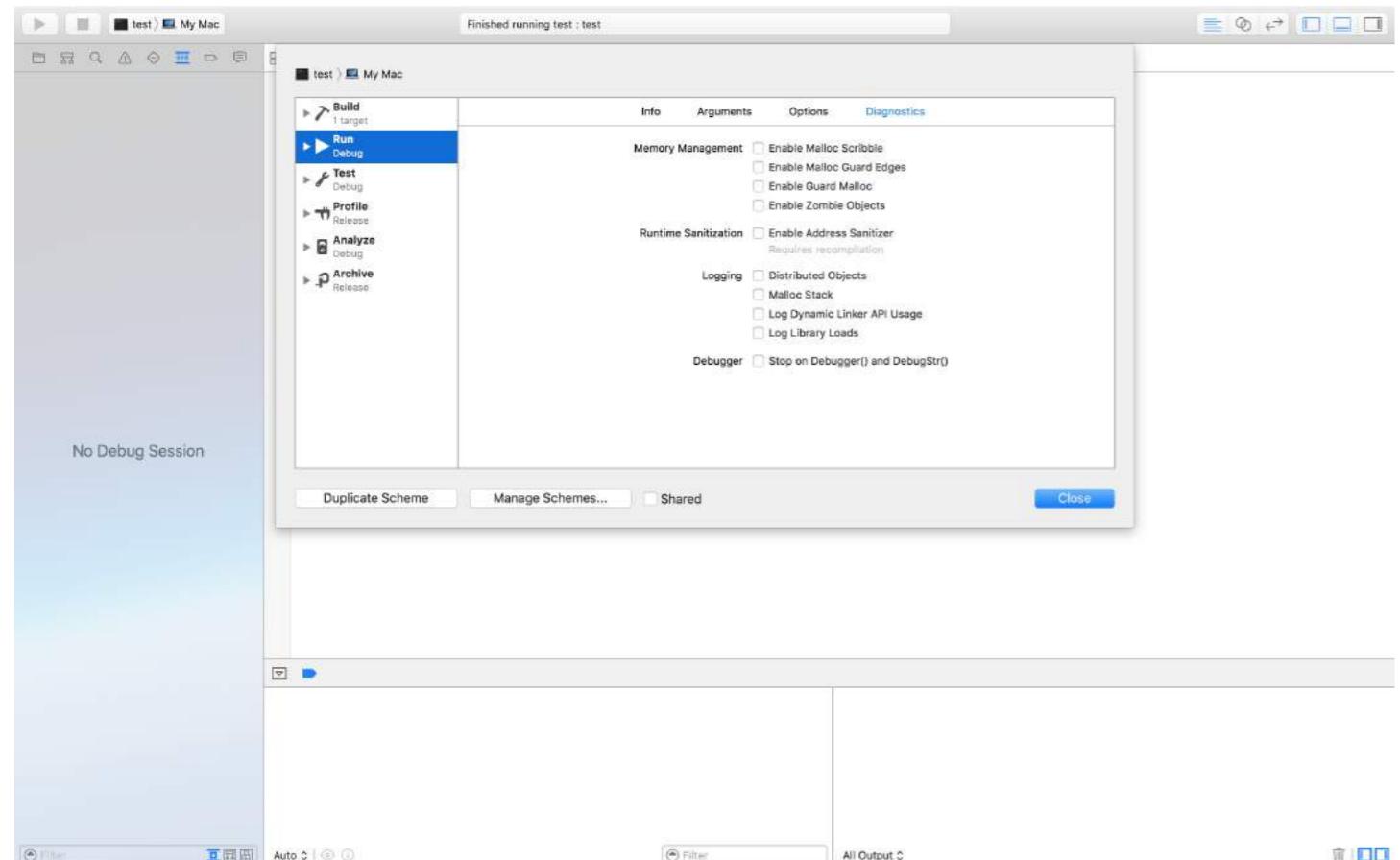
EXC_BAD_ACCESS 意味着进程尝试以无效方式访问内存，比如解引用一个 NULL 指针或写入只读内存。这是最难调试的崩溃类型，因为它通常没有错误信息，有些崩溃可能非常难以重现和/或发生在与问题完全无关的代码中。这个错误在 Swift 中非常罕见，但如果发生，通常可以通过降低编译器优化来获得更易调试的崩溃。

大多数 EXC_BAD_ACCESS 错误是由于尝试解引用一个 NULL 指针引起的。如果是这种情况，红色箭头中列出的地址通常是一个低于正常内存地址的十六进制数字，通常是 0x0。在调试器中设置断点或偶尔添加 printf/NSLog 语句，以找出该指针为何为 NULL。

不太稳定发生或完全没有意义的 EXC_BAD_ACCESS 可能是内存管理问题导致的。常见导致此问题的原因有：

- 使用已被释放的内存
- 尝试写入超出 C 数组或其他类型缓冲区末尾的内容
- 使用未初始化的指针

在方案编辑器的诊断部分，Xcode 包含一些有用的工具来帮助调试内存问题：



地址消毒器（Address Sanitizer）增加了许多检查，一旦发生内存问题就会停止应用并提供详细说明具体发生了什么的错误信息。僵尸对象（Zombie Objects）检测已释放的 Objective-C 对象的问题，但开启自动引用计数（Automatic Reference Counting）后通常不会出现这类问题。

Chapter 129: Debugging Crashes

Section 129.1: Debugging EXC_BAD_ACCESS

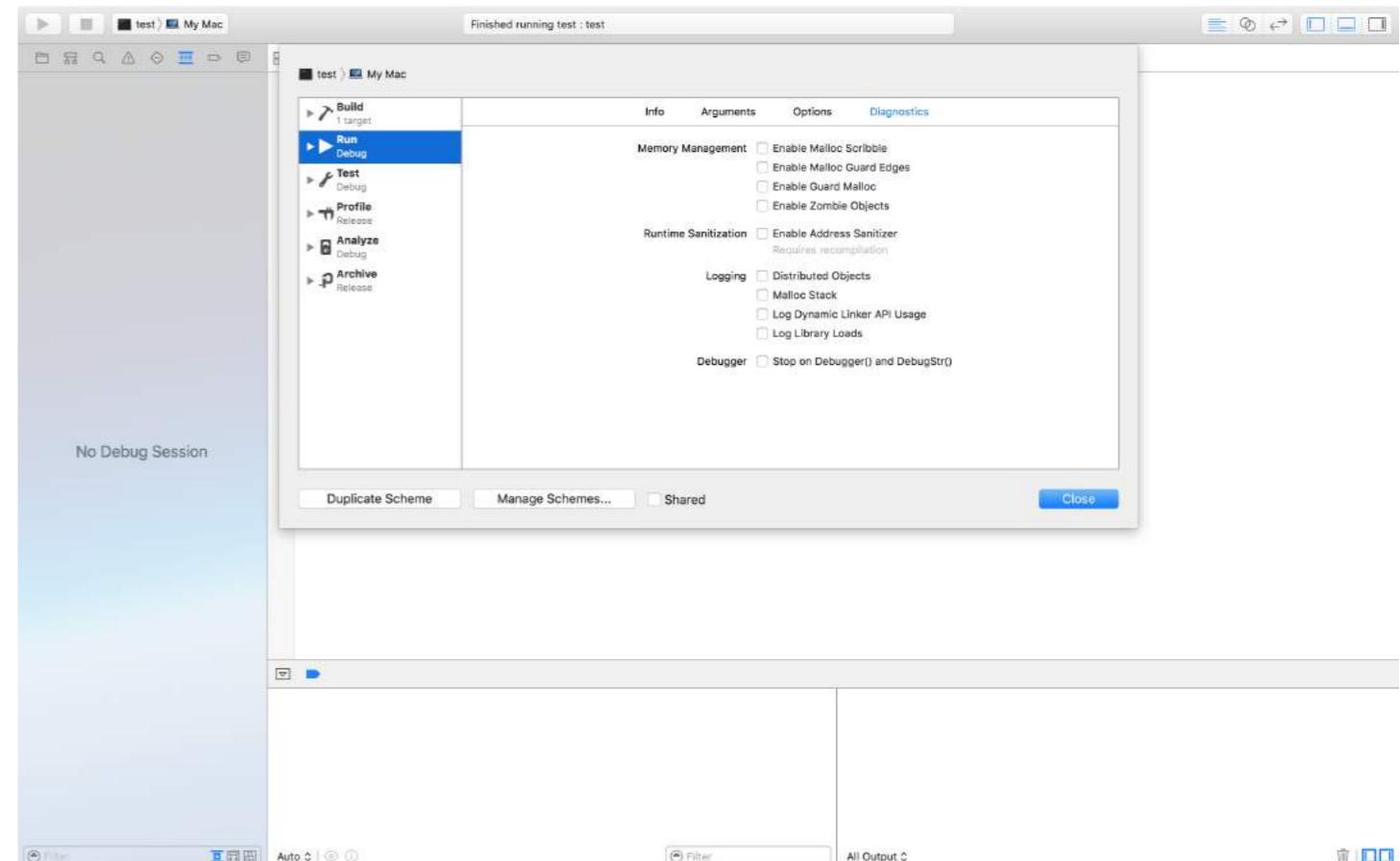
EXC_BAD_ACCESS means the process tried to access memory in an invalid way, like dereferencing a NULL pointer or writing to read-only memory. This is the hardest kind of crash to debug, because it usually does not have an error message, and some crashes can be very difficult to reproduce and/or occur in code completely unrelated to the problem. This error is very rare in Swift, but if it occurs, you can often get easier-to-debug crashes by reducing compiler optimizations.

Most EXC_BAD_ACCESS errors are caused by trying to dereference a NULL pointer. If this is the case, the address listed in the red arrow will usually be a hexadecimal number that is lower than a normal memory address, often 0x0. Set breakpoints in the debugger or add occasional printf/NSLog statements to find out why that pointer is NULL.

An EXC_BAD_ACCESS that occurs less reliably or makes no sense at all could be the result of a memory management problem. Common problems that can cause this are:

- Using memory that has been deallocated
- Trying to write past the end of a C array or other kind of buffer
- Using a pointer which has not been initialized

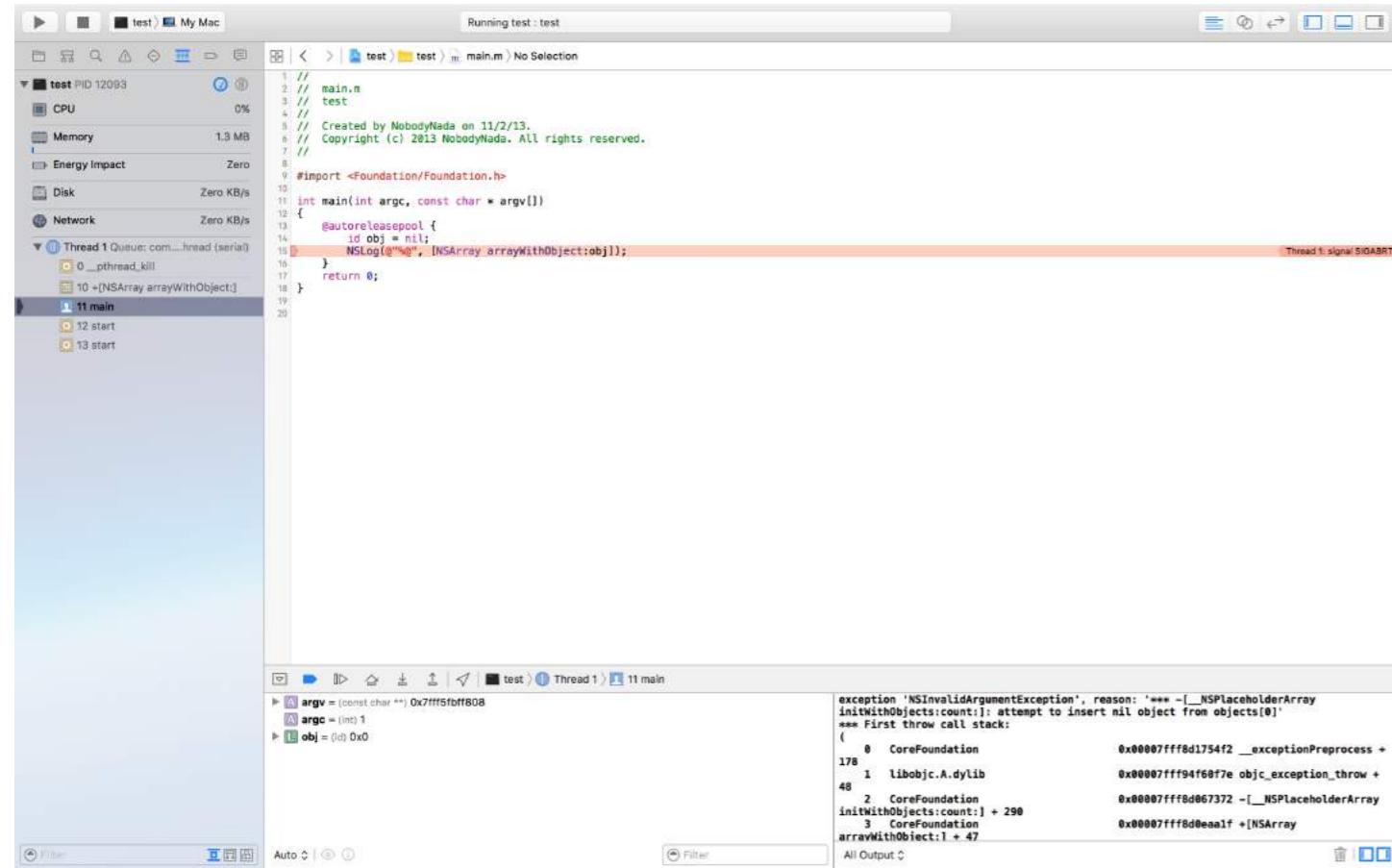
In the Diagnostics section of the Scheme Editor, Xcode includes a few useful tools to help debug memory problems:



The Address Sanitizer adds lots of checks that will stop the app whenever memory problems occur and provide a helpful error message detailing exactly what happened. Zombie Objects detects problems with deallocated Objective-C objects, but you shouldn't get these kinds of problems with Automatic Reference Counting turned on.

第129.2节：查找崩溃信息

当您的应用崩溃时，Xcode会进入调试器并显示有关崩溃的更多信息：



最重要的部分是：

红色箭头

红色箭头显示了哪一行代码崩溃以及崩溃的原因。

调试器控制台

许多崩溃会将更多信息记录到调试器控制台。应用程序崩溃时，调试器应自动出现，

但如果未出现，可以通过选择来显示调试器



在Xcode右上角的按钮，显示

通过点击控制台



调试器右下角的按钮。

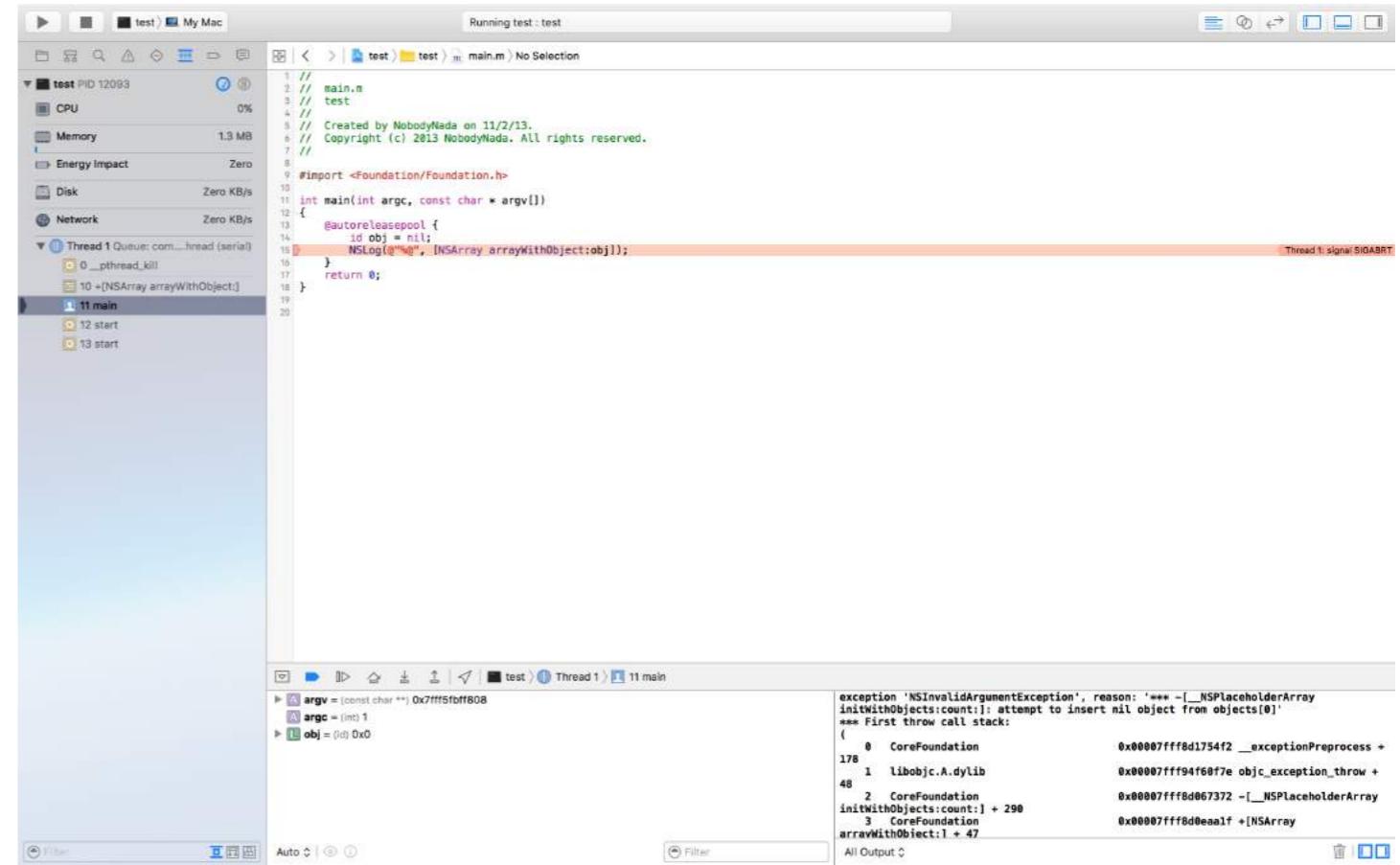
堆栈跟踪

堆栈跟踪列出了程序在到达崩溃代码之前调用过的函数。

部分堆栈跟踪显示在屏幕左侧的调试导航器中，调试器控件允许您选择一个堆栈帧以在调试器中查看：

Section 129.2: Finding information about a crash

When your app crashes, Xcode will enter the debugger and show you more information about the crash:



The most important parts are:

The red arrow

The red arrow displays which line of code crashed & why it crashed.

The debugger console

Many crashes log more information to the debugger console. It should automatically appear when the app crashes,

but if it's not there, show the debugger by selecting the

button in the top-right corner of Xcode, and show the

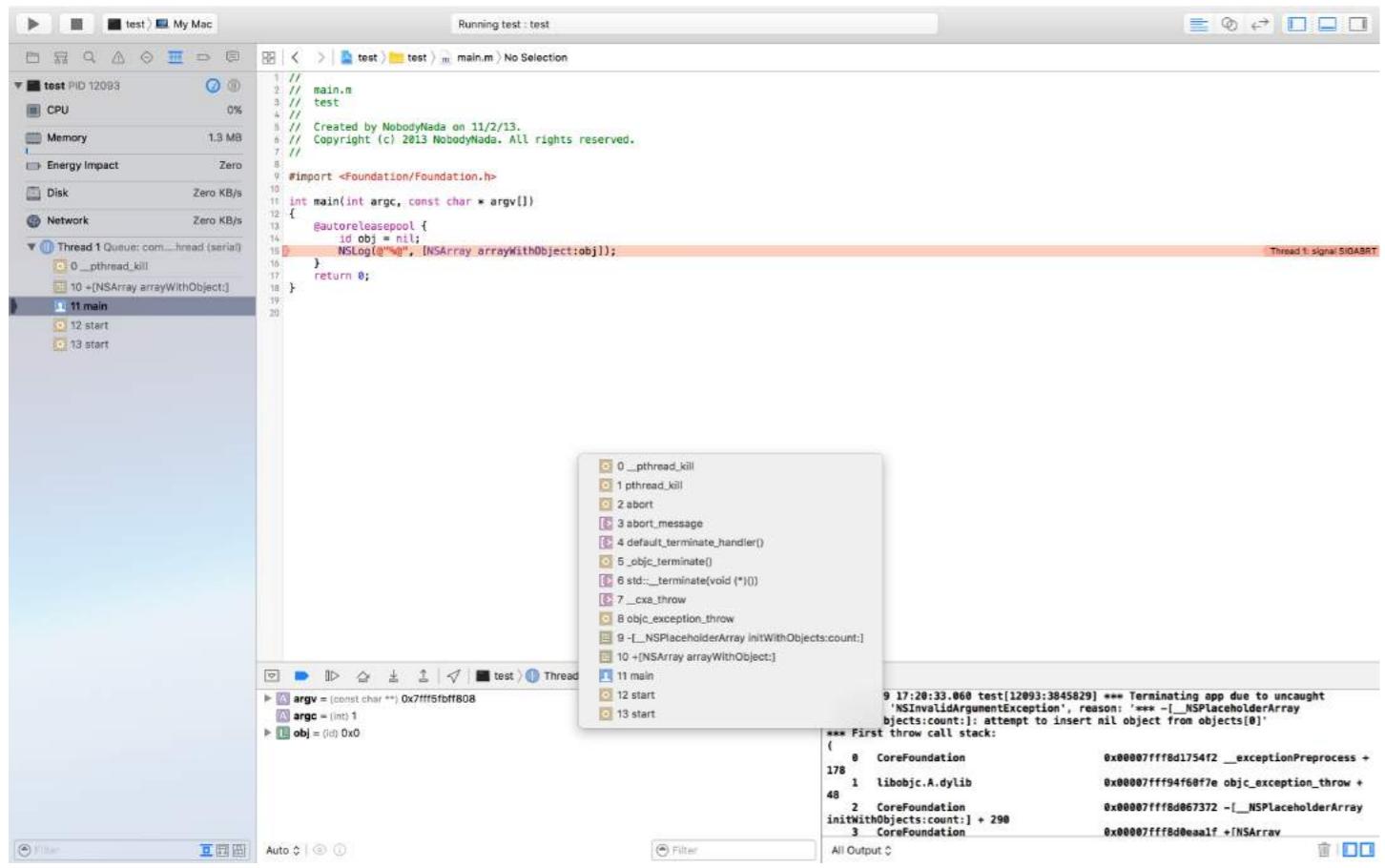
console by clicking the

button in the bottom-right corner of the debugger.

The stack trace

The stack trace lists the functions the program came from before it got to the code that crashed.

Part of the stack trace is displayed in the Debug Navigator on the left of the screen, and the debugger controls allow you to select a stack frame to view in the debugger:



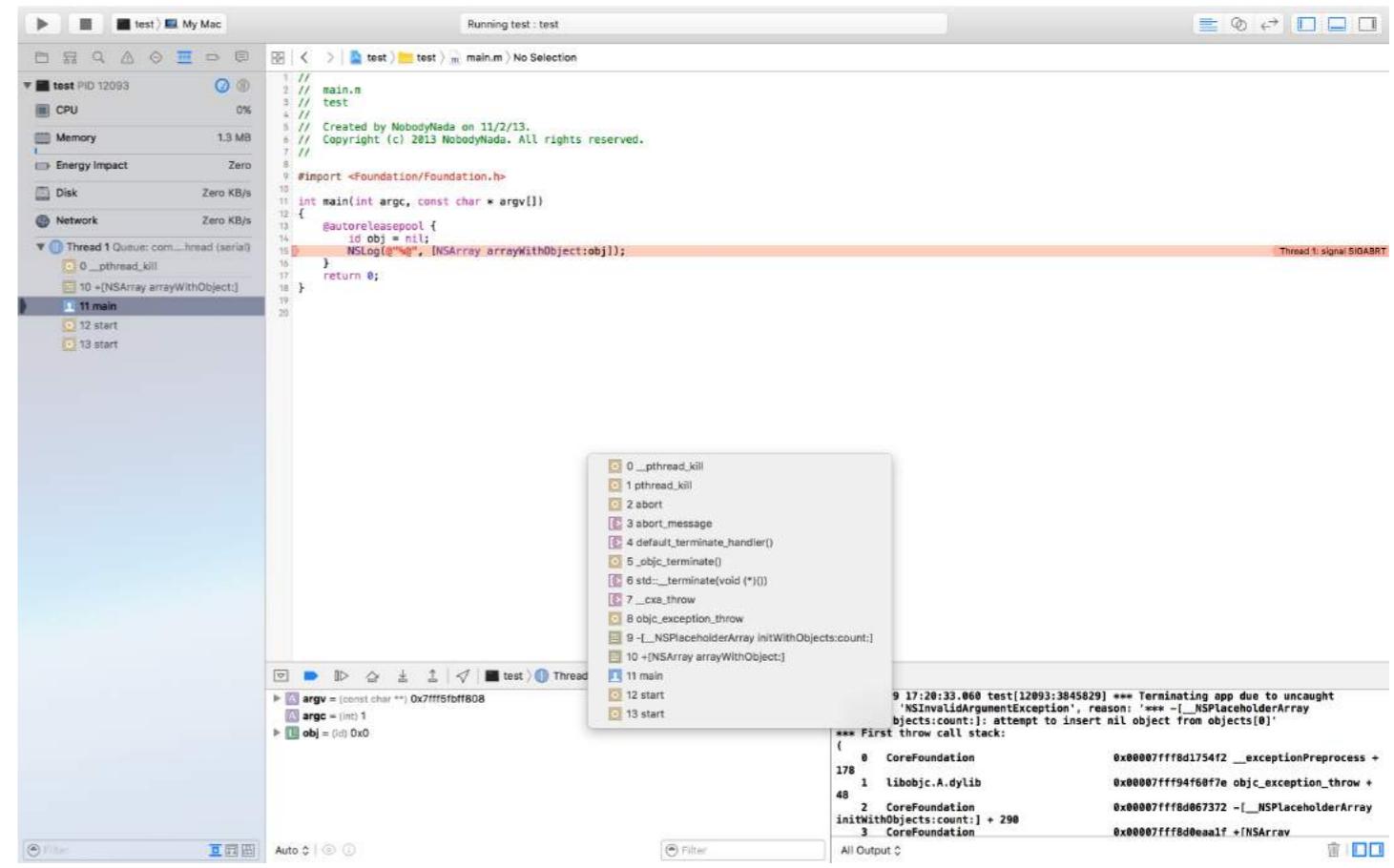
如果你在调试器的 (lldb) 提示符下输入 `bt` 命令并按下 `return`，你将获得一个堆栈跟踪的文本表示，可以复制粘贴：

```
(lldb) bt
* 线程 #1: 线程ID = 0x3aaec5, 0x00007fff91055f06 libsystem_kernel.dylib`__pthread_kill + 10, 队列 =
  'com.apple.main-thread', 停止原因 = 信号 SIGABRT
帧 #0: 0x00007fff91055f06 libsystem_kernel.dylib`__pthread_kill + 10
  帧 #1: 0x000000010008142d libsystem_pthread.dylib`pthread_kill + 90
  帧 #2: 0x00007fff96dc76e7 libsystem_c.dylib`abort + 129
帧 #3: 0x00007fff8973bf81 libc++abi.dylib`abort_message + 257
  帧 #4: 0x00007fff89761a47 libc++abi.dylib`default_terminate_handler() + 267
  帧 #5: 0x00007fff94f636ae libobjc.A.dylib`_objc_terminate() + 103
帧 #6: 0x00007fff8975f19e libc++abi.dylib`std::__terminate(void (*)) + 8
  帧 #7: 0x00007fff8975ec12 libc++abi.dylib`__cxa_throw + 121
帧 #8: 0x00007fff94f6108c libobjc.A.dylib`objc_exception_throw + 318
  帧 #9: 0x00007fff8d067372 CoreFoundation`-[__NSPlaceholderArray initWithObjects:count:] + 290
帧 #10: 0x00007fff8d0eaaf CoreFoundation`+[NSArray arrayWithObject:] + 47
* frame #11: 0x000000010001b54 test`main(argc=1, argv=0x00007fff5fbff808) + 68 at main.m:15
帧 #12: 0x00007fff8bea05ad libdyld.dylib`start + 1
帧 #13: 0x00007fff8bea05ad libdyld.dylib`start + 1
```

第129.3节：调试 SIGABRT 和 EXC_BAD_INSTRUCTION 崩溃

一个SIGABRT或EXC_BAD_INSTRUCTION通常意味着应用程序因某些检查失败而故意崩溃。它们应该会在调试器控制台记录一条包含更多信息的消息；请在那里查看更多信息。

许多SIGABRT是由未捕获的Objective-C异常引起的。异常可能被抛出的原因有很多，且它们总是会在控制台记录大量有用的信息。



If you enter the `bt` command at the (lldb) prompt in the debugger and press `return`，you will get a textual representation of the stack trace that you can copy and paste:

```
(lldb) bt
* thread #1: tid = 0x3aaec5, 0x00007fff91055f06 libsystem_kernel.dylib`__pthread_kill + 10, queue =
  'com.apple.main-thread', stop reason = signal SIGABRT
frame #0: 0x00007fff91055f06 libsystem_kernel.dylib`__pthread_kill + 10
frame #1: 0x000000010008142d libsystem_pthread.dylib`pthread_kill + 90
frame #2: 0x00007fff96dc76e7 libsystem_c.dylib`abort + 129
frame #3: 0x00007fff8973bf81 libc++abi.dylib`abort_message + 257
frame #4: 0x00007fff89761a47 libc++abi.dylib`default_terminate_handler() + 267
frame #5: 0x00007fff94f636ae libobjc.A.dylib`_objc_terminate() + 103
frame #6: 0x00007fff8975f19e libc++abi.dylib`std::__terminate(void (*)) + 8
frame #7: 0x00007fff8975ec12 libc++abi.dylib`__cxa_throw + 121
frame #8: 0x00007fff94f6108c libobjc.A.dylib`objc_exception_throw + 318
frame #9: 0x00007fff8d067372 CoreFoundation`-[__NSPlaceholderArray initWithObjects:count:] + 290
frame #10: 0x00007fff8d0eaaf CoreFoundation`+[NSArray arrayWithObject:] + 47
* frame #11: 0x000000010001b54 test`main(argc=1, argv=0x00007fff5fbff808) + 68 at main.m:15
frame #12: 0x00007fff8bea05ad libdyld.dylib`start + 1
frame #13: 0x00007fff8bea05ad libdyld.dylib`start + 1
```

Section 129.3: Debugging SIGABRT and EXC_BAD_INSTRUCTION crashes

A SIGABRT or an EXC_BAD_INSTRUCTION usually means the app crashed itself intentionally because some check failed. These should log a message to the debugger console with more information; check there for more information.

Many SIGABRTs are caused by uncaught Objective-C exceptions. There are a lot of reasons exceptions can be thrown, and they will always log a lot of helpful information to the console.

- `NSInvalidArgumentException`, 表示应用程序向方法传递了无效参数
- `NSRangeException`, 表示应用程序试图访问对象（如 `NSArray` 或 `NSString`）的越界索引
- `NSInternalInconsistencyException` 表示对象发现自身处于意外状态。
- `NSUnknownKeyException` 通常表示你在 XIB 中有错误的连接。尝试一些对[这个问题](#)的解答。

- `NSInvalidArgumentException`, which means the app passed an invalid argument to a method
- `NSRangeException`, which means the app tried to access an out-of-bounds index of an object such as an `NSArray` or an `NSString`
- `NSInternalInconsistencyException` means an object discovered it was in an unexpected state.
- `NSUnknownKeyException` usually means you have a bad connection in an XIB. Try some of the answers to [this question](#).

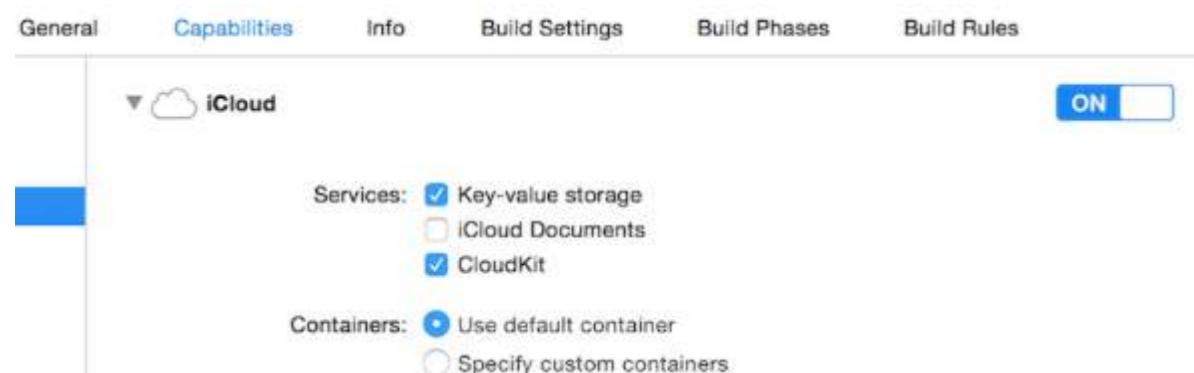
第130章：CloudKit

第130.1节：注册应用以使用CloudKit

您需要获取一个权限文件，以便应用能够访问您的iCloud并使用CloudKit写入记录。

按照以下步骤授予应用访问iCloud的权限：

- 1- 在项目导航器中选择项目，然后打开“常规”标签页。
- 2- 在“身份”部分，将您的开发者Apple ID设置到“团队”下拉菜单中。（如果不可用，请在Xcode菜单 -> 偏好设置 -> 账户中添加。）
- 3- 进入项目属性中的“功能”标签页，开启iCloud。然后选择“键值存储”和“CloudKit”。



4- 确保这些项目已被勾选：

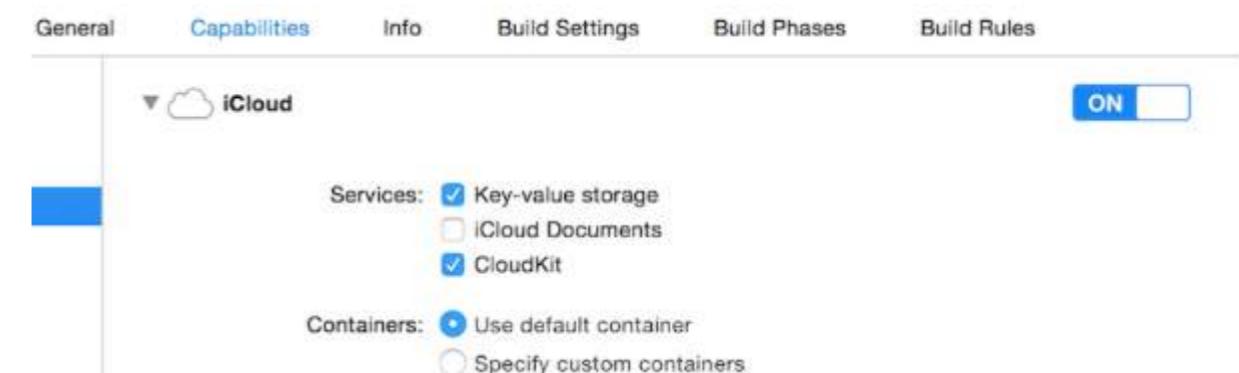
Chapter 130: CloudKit

Section 130.1: Registering app for use with CloudKit

What you need is to get an entitlements file so the app can access your iCloud and write records using CloudKit.

Follow the steps to grant access to iCloud from your app:

- 1- Select the project in the Project Navigator, and then open the General tab.
- 2- In the Identity section, set your developer Apple ID to the Team dropdown menu. (If it is not available, add it in Xcode menu -> Preferences -> Accounts.)
- 3- Go to Capabilities tab in the project properties and turn iCloud on. Then, select "Key-Value Storage" and "CloudKit".



4- Make sure these items are checked:

如果所有项目都已勾选，则您的应用已准备好使用CloudKit。

第130.2节：使用CloudKit仪表盘

所有使用CloudKit相关代码创建的记录都可以在CloudKit仪表盘中预览、编辑甚至删除。要访问CloudKit仪表盘，请前往 [here](#)。

仪表盘中有几个部分：

- 记录类型（稍后将讨论）
- 安全角色（您可以在这里设置数据库为公开或私有）订阅类型（您的应用可以注册苹果推送通知（APNs），以便在记录更改时通知您）

记录类型

在这里，您可以看到应用中所有现有的记录类型列表。当您第一次打开某个应用的CloudKit仪表板时，那里有一个名为Users的记录类型，您可以使用它，也可以删除它并使用您自己的记录类型。

在此页面，您可以手动设置数据类型。当然，在大多数情况下这没有意义，因为iOS SDK

Section 130.2: Using CloudKit Dashboard

All the records created using CloudKit-related code can be previewed, edited and even removed in CloudKit Dashboard. To access CloudKit Dashboard, go [here](#).

There are several parts in the dashboard:

- Record Types (which will be discussed later)
- Security Roles (which is where you can set databases as public or private)
- Subscription Types (which your app could register for Apple Push Notifications (APNs) to notify you when a record is changed)

Record Types

Here, you get a list of all the existing record types in the app. When you first open CloudKit Dashboard for an app, there's a record type called Users there, which you can use it or just delete it and use your own.

In this page, you could make your data typed manually. Of course, in most cases this is pointless, because iOS SDK

比仪表板处理得更好，但如果您愿意，这个功能也存在。此页面最常用的功能是预览类型。

第130.3节：将数据保存到CloudKit

要将数据保存到CloudKit，我们必须创建：

- 一个CKRecordID（您唯一记录的键）
- 一个CKRecord（包含数据）

制作记录键

为了确保每个新记录标识符都是唯一的，我们使用当前的时间戳，它是唯一的。我们使用NSDate的方法timeIntervalSinceReferenceDate()获取时间戳。它的形式是###.### (#代表数字)，我们将使用整数部分。为此，我们将字符串拆分：

Swift

```
let timestamp = String(format: "%f", NSDate.timeIntervalSinceReferenceDate())
let timestampParts = timestamp.componentsSeparatedByString(".")
let recordID = CKRecordID(recordName: timestampParts[0])
```

制作记录

制作记录时，我们应指定记录类型（在使用CloudKit仪表盘中有说明）为Users，ID为刚才制作的内容，以及数据。这里，我们将向记录中添加示例文本、一张图片和当前日期：

Swift

```
let record = CKRecord(recordType: "Users", recordID: recordID)
record.setObject("Some Text", forKey: "text")
record.setObject(CKAsset(fileURL: someValidImageURL), forKey: "image")
record.setObject(NSDate(), forKey: "date")
```

Objective-C

```
CKRecord *record = [[CKRecord alloc] initWithRecordType: "Users" recordID: recordID];
[record setObject: "Some Text" forKey: "text"];
[record setObject: [CKAsset assetWithFileURL: someValidImageURL] forKey: "image"];
[record setObject: [[NSDate alloc] init] forKey: "date"];
```

注意

这里，我们没有直接将UIImage添加到记录中，因为如备注中所述，CloudKit不直接支持图像格式，因此我们将UIImage转换成了CKAsset。

访问容器

Swift

```
let container = CKContainer.defaultContainer()
let database = container.privateCloudDatabase // 或者 container.publicCloudDatabase
```

将记录保存到 CloudKit 数据库

Swift

```
database.saveRecord(record, completionHandler: { (_, error) -> Void in
    print(error ?? "")
})
```

can handle it way better than the dashboard, but the functionality is also there if you prefer. The most use of this page is for previewing types.

Section 130.3: Saving data to CloudKit

To save date to CloudKit, we must make:

- A CKRecordID (the key of your unique record)
- A CKRecord (which includes data)

Making a record key

To ensure that every new record identifier is unique, we use the current *timestamp*, which is unique. We get the timestamp using NSDate's method `timeIntervalSinceReferenceDate()`. It is in form of ###.### (# are numbers), which we will use the integer part. To do this, we split the string:

Swift

```
let timestamp = String(format: "%f", NSDate.timeIntervalSinceReferenceDate())
let timestampParts = timestamp.componentsSeparatedByString(".")
let recordID = CKRecordID(recordName: timestampParts[0])
```

Making the record

To make the record, we should specify the record type (explained in Using CloudKit Dashboard) as Users, the ID as the thing we made just now and the data. Here, we will add a sample text, a picture and the current date to the record:

Swift

```
let record = CKRecord(recordType: "Users", recordID: recordID)
record.setObject("Some Text", forKey: "text")
record.setObject(CKAsset(fileURL: someValidImageURL), forKey: "image")
record.setObject(NSDate(), forKey: "date")
```

Objective-C

```
CKRecord *record = [[CKRecord alloc] initWithRecordType: "Users" recordID: recordID];
[record setObject: "Some Text" forKey: "text"];
[record setObject: [CKAsset assetWithFileURL: someValidImageURL] forKey: "image"];
[record setObject: [[NSDate alloc] init] forKey: "date"];
```

Note

Here, we didn't add the UIImage directly to the record, because as mentioned in Remarks, image format isn't directly supported in CloudKit, so we have converted UIImage into CKAsset.

Accessing the container

Swift

```
let container = CKContainer.defaultContainer()
let database = container.privateCloudDatabase // or container.publicCloudDatabase
```

Saving the records to CloudKit database

Swift

```
database.saveRecord(record, completionHandler: { (_, error) -> Void in
    print(error ?? "")
})
```

第131章：GameplayKit

第131.1节：生成随机数

虽然GameplayKit（随 iOS 9 SDK 引入）主要用于实现游戏逻辑，但它也可以用来生成随机数，这在应用和游戏中非常有用。

除了以下章节中使用的GKRandomSource.sharedRandom之外，开箱即用的还有三种额外类型的GKRandomSource。

- GKARC4RandomSource 使用 ARC4 算法GKLinearCongruentialRandomSource 是一种快速但随机性较差的GKRandomSourceGKMersenneTwisterRandomSource 实现了梅森旋转算法。它速度较慢但随机性更强。

在接下来的章节中，我们只使用GKRandomSource的nextInt()方法。除此之外，还有nextBool() -> Bool和nextUniform() -> Float

生成

首先，导入GameplayKit：

Swift

```
import GameplayKit
```

Objective-C

```
#import <GameplayKit/GameplayKit.h>
```

然后，使用以下代码生成一个随机数：

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt()
```

Objective-C

```
int randomNumber = [[GKRandomSource sharedRandom] nextInt];
```

注意

当nextInt()函数不带参数使用时，会返回一个介于-2,147,483,648和2,147,483,647之间的随机数，包含这两个数，因此我们不能确定它总是一个正数或非零数。

生成一个从0到n的数字

为实现这一点，您应将 n 传递给 nextIntWithUpperBound() 方法：

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt(upperBound: 10)
```

Objective-C

Chapter 131: GameplayKit

Section 131.1: Generating random numbers

Although GameplayKit (which is introduced with iOS 9 SDK) is about implementing game logic, it could also be used to generate random numbers, which is very useful in apps and games.

Beside the GKRandomSource.sharedRandom which is used in the following chapters there are three additional types of GKRandomSource's out of the box.

- **GKARC4RandomSource** Which uses the the ARC4 algorithm
- **GKLinearCongruentialRandomSource** Which is a fast but not so random GKRandomSource
- **GKMersenneTwisterRandomSource** Which implements a MersenneTwister algorithm. It is slower but more random.

In the following chapter we only use the nextInt() method of a GKRandomSource. In addition to this there is the nextBool() -> Bool and the nextUniform() -> Float

Generation

First, import GameplayKit:

Swift

```
import GameplayKit
```

Objective-C

```
#import <GameplayKit/GameplayKit.h>
```

Then, to generate a random number, use this code:

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt()
```

Objective-C

```
int randomNumber = [[GKRandomSource sharedRandom] nextInt];
```

Note

The nextInt() function, when used without parameters, will return a random number between -2,147,483,648 and 2,147,483,647, including themselves, so we are not sure that it is always a positive or non-zero number.

Generating a number from 0 to n

To achieve this, you should give n to nextIntWithUpperBound() method:

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt(upperBound: 10)
```

Objective-C

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: 10];
```

这段代码会给我们一个介于0到10之间的数字，包括0和10。

生成从m到n的数字

为此，你需要创建一个带有GKRandomSource的GKRandomDistribution对象，并传入边界值。一个GKRandomDistribution可以用来改变分布行为，比如GKGaussianDistribution或GKShuffledDistribution。

之后，该对象可以像普通的GKRandomSource一样使用，因为它也实现了GKRandom协议。

Swift

```
let randomizer = GKRandomDistribution(randomSource: GKRandomSource(), lowestValue: 0, highestValue:  
6)  
let randomNumberInBounds = randomizer.nextInt()
```

Objective-C outdated

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: n - m] + m;
```

例如，要生成一个介于3到10之间的随机数，可以使用以下代码：

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt(upperBound: 7) + 3
```

Objective-C outdated

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: 7] + 3;
```

第131.2节：GKEntity和GKComponent

实体代表游戏中的一个对象，比如玩家角色或敌人角色。由于这个对象没有手脚，它本身不能做太多事情，我们可以给它添加组件。为了创建这个系统，苹果提供了GKEntity和GKComponent类。

假设我们在接下来的章节中有以下类：

```
class Player: GKEntity{}  
class PlayerSpriteComponent: GKComponent {}
```

GKEntity

实体是组件的集合，并提供多个函数来添加、移除和与其组件交互。

虽然我们可以直接使用 GKEntity，但通常会为特定类型的游戏实体对其进行子类化。

重要的是，每个类的组件只能添加一次。如果你添加了第二个相同类型的组件，它将覆盖 GKEntity 中已存在的第一个组件。

```
let otherComponent = PlayerSpriteComponent()  
var player = Player()  
player.addComponent(PlayerSpriteComponent())  
player.addComponent(otherComponent)
```

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: 10];
```

This code will give us a number between 0 and 10, including themselves.

Generating a number from m to n

To do this you create a GKRandomDistribution object with a GKRandomSource and pass in the bounds. A GKRandomDistribution can be used to change the distribution behaviour like GKGaussianDistribution or GKShuffledDistribution.

After that the object can be used like every regular GKRandomSource since it does implement the GKRandom protocol too.

Swift

```
let randomizer = GKRandomDistribution(randomSource: GKRandomSource(), lowestValue: 0, highestValue:  
6)  
let randomNumberInBounds = randomizer.nextInt()
```

Objective-C outdated

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: n - m] + m;
```

For example, to generate a random number between 3 and 10, you use this code:

Swift

```
let randomNumber = GKRandomSource.sharedRandom().nextInt(upperBound: 7) + 3
```

Objective-C outdated

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: 7] + 3;
```

Section 131.2: GKEntity and GKComponent

An entity represents an object of a game like a player figure or an enemy figure. Since this object does not do much without arms and legs we can add the components to this. To create this system apple has the GKEntity and GKComponent classes.

Lets assume we have the following classe for the following chapters:

```
class Player: GKEntity{}  
class PlayerSpriteComponent: GKComponent {}
```

GKEntity

An entity is a collection of components and offers several functions to add, remove and interact with components of it.

While we could just use the GKEntity it is common to Subclass it for a specific type of game entity.

It is important that it is only possible to add a component of a class once. In case you add a second component of the same class it will override the first existing component inside of the GKEntity

```
let otherComponent = PlayerSpriteComponent()  
var player = Player()  
player.addComponent(PlayerSpriteComponent())  
player.addComponent(otherComponent)
```

```
print(player.components.count) //将打印 1  
print(player.components[0] == otherComponent) // 将打印 true
```

你可能会问为什么。原因是调用了名为component(for: T.Type)的方法，该方法返回实体中特定类型的组件。

```
let component = player.component(ofType: PlayerSpriteComponent.self)
```

除了组件方法外，它还有一个update方法，用于将游戏逻辑的增量时间或当前时间委托给其组件。

```
var player = Player()  
player.addComponent(PlayerSpriteComponent())  
player.update(deltaTime: 1.0) // 将调用添加到它的 PlayerSpriteComponent 的 update 方法
```

GKComponent

组件表示实体的某个部分，例如视觉组件或逻辑组件。

如果调用实体的 update 方法，它会将此调用委托给所有组件。重写此方法用于操作实体。

```
class PlayerSpriteComponent: GKComponent {  
    override func update(deltaTime seconds: TimeInterval) {  
        // 根据更新时间移动精灵  
    }  
}
```

此外，可以重写方法 didAddToEntity 和 willRemoveFromEntity 来通知其他组件其被添加或移除。

要操作组件内的其他组件，可以获取该组件所属的 GKEntity。

```
override func update(deltaTime seconds: TimeInterval) {  
    let controller = self.entity?.component(ofType: PlayerControlComponent.self)  
    // 调用控制器上的方法  
}
```

虽然这是可能的，但这不是一种常见的模式，因为它将两个组件连接在一起。

GK组件系统

虽然我们刚才讨论了使用GKEntity的更新委托机制来更新GKComponents，但还有一种不同的方式来更新GKComponents，称为GKComponentSystem。

当需要一次性更新所有特定类型的组件时，会使用它。

为特定类型的组件创建一个GKComponentSystem。

```
let system = GKComponentSystem(componentClass: PlayerSpriteComponent.self)
```

要添加组件，可以使用add方法：

```
system.addComponent(PlayerSpriteComponent())
```

```
print(player.components.count) //will print 1  
print(player.components[0] == otherComponent) // will print true
```

You may ask why. The reason for this is the methods called component(for: T.Type) which returns the component of a specific type of the entity.

```
let component = player.component(ofType: PlayerSpriteComponent.self)
```

In addition to the components-methods it has an update method which is used to delegate the delta time or current time of the game logic to its components.

```
var player = Player()  
player.addComponent(PlayerSpriteComponent())  
player.update(deltaTime: 1.0) // will call the update method of the PlayerSpriteComponent added to it
```

GKComponent

A component represents something of an entity for example the visual component or the logic component.

If an the update method of an entity is called it will delegate this to all of its components. Overriding this method is used to manipulate an Entity.

```
class PlayerSpriteComponent: GKComponent {  
    override func update(deltaTime seconds: TimeInterval) {  
        //move the sprite depending on the update time  
    }  
}
```

In addition to this it is possible to override the method didAddToEntity and willRemoveFromEntity to inform other components about its removal or add.

To manipulate a other component inside of a component it is possible to get the GKEntity which the component is added to.

```
override func update(deltaTime seconds: TimeInterval) {  
    let controller = self.entity?.component(ofType: PlayerControlComponent.self)  
    //call methods on the controller  
}
```

While this is possible it is **not** a common pattern since it wires the two components together.

GKComponentSystem

While we just talked about using the update delegate mechanism of the GKEntity to update the GKComponents there is a different way to update GKComponents which is called GKComponentSystem.

It is used in case it is needed that all components of a specific type need to be updated in one go.

A GKComponentSystem is created for a specific type of component.

```
let system = GKComponentSystem(componentClass: PlayerSpriteComponent.self)
```

To add a component you can use the add method:

```
system.addComponent(PlayerSpriteComponent())
```

但更常见的方式是将创建的实体及其组件传递给GKComponentSystem，它会在实体中找到匹配的组件。

```
system.addComponent(foundIn: player)
```

要更新特定类型的所有组件，请调用更新：

```
system.update(deltaTime: delta)
```

如果您想使用GKComponentSystem而不是基于实体的更新机制，则必须为每个组件拥有一个GKComponentSystem，并对所有系统调用更新。

But a more common way is to pass the created entity with it's components to the GKComponentSystem and it will find a matching component inside of the entity.

```
system.addComponent(foundIn: player)
```

To update all components of a specific type call the update:

```
system.update(deltaTime: delta)
```

In case you want to use the GKComponentSystem instead of a entity based update mechanism you have to have a GKComponentSystem for every component and call the update on all of the systems.

第132章：从命令行进行Xcode构建与归档

选项	描述
-project	构建项目 name.xcodeproj。
-scheme	构建工作区时必需。
-目的地	使用目标设备
-configuration	使用构建配置
-sdk	指定的 SDK

第 132.1 节：构建与归档

构建：

```
xcodetool -exportArchive -exportFormat ipa \
-archivePath "/Users/username/Desktop/MyiOSApp.xcarchive" \
-exportPath "/Users/username/Desktop/MyiOSApp.ipa" \
-exportProvisioningProfile "MyCompany Distribution Profile"
```

归档：

```
xcodetool -project <ProjectName.xcodeproj>
-scheme <ProjectName>
-sdk iphonesimulator
-configuration Debug
-destination "platform=iOS Simulator,name=<Device>,OS=9.3"
clean build
```

Chapter 132: Xcode Build & Archive From Command Line

Option	Description
-project	Build the project name.xcodeproj.
-scheme	Required if building a workspace.
-destination	Use the destination device
-configuration	Use the build configuration
-sdk	specified SDK

Section 132.1: Build & Archive

Build:

```
xcodetool -exportArchive -exportFormat ipa \
-archivePath "/Users/username/Desktop/MyiOSApp.xcarchive" \
-exportPath "/Users/username/Desktop/MyiOSApp.ipa" \
-exportProvisioningProfile "MyCompany Distribution Profile"
```

Archive:

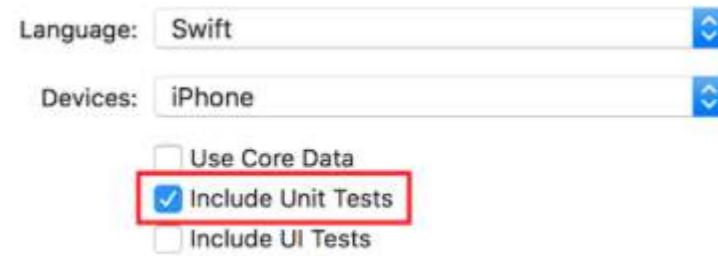
```
xcodetool -project <ProjectName.xcodeproj>
-scheme <ProjectName>
-sdk iphonesimulator
-configuration Debug
-destination "platform=iOS Simulator,name=<Device>,OS=9.3"
clean build
```

第133章：XCTest框架 - 单元测试

第133.1节：向Xcode项目添加测试文件

创建项目时

您应该在项目创建对话框中勾选“包含单元测试”。



创建项目后

如果您在创建项目时错过了勾选该项，您仍然可以稍后添加测试文件。操作步骤如下：

1- 进入 Xcode 中的项目设置

2- 进入“Targets”（目标）

3- 点击“Add Target”（添加目标）

4- 在“Other”（其他）下，选择“Cocoa Touch Unit Test Testing Bundl

e”最后，您应该会有一个名为[您的应用名称]Tests.swift的文件。若是 Objective-C，则应有两个文件，分别命名为[您的应用名称]Tests.h和[您的应用名称]Tests.m。

[您的应用名称]Tests.swift或.m文件默认包含：

- 一个XCTest模块导入
- 一个继承自XCTestCase的[您的应用名称]Tests类，包含setUp、tearDown、testExample、testPerformanceExample方法

Swift

```
import XCTest

class MyProjectTests: XCTestCase {

    override func setUp() {
        super.setUp()
        // 在此处放置初始化代码。此方法在类中每个测试方法调用之前被调用。
    }

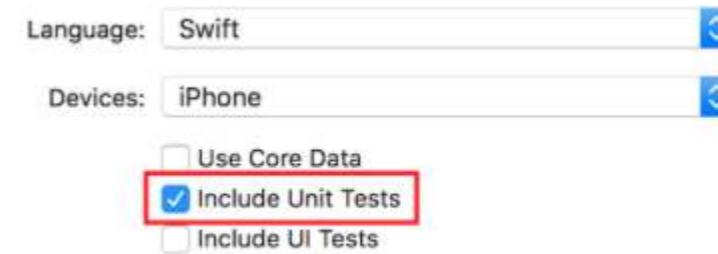
    override func tearDown() {
        // 在此处放置清理代码。此方法在类中每个测试方法调用之后被调用。
        super.tearDown()
    }
}
```

Chapter 133: XCTest framework - Unit Testing

Section 133.1: Adding Test Files to Xcode Project

When creating the project

You should check "Include Unit Tests" in the project creation dialog.



After creating the project

If you missed checking that item while creating your project, you could always add test files later. To do so:

1- Go to your project settings in Xcode

2- Go to "Targets"

3- Click "Add Target"

4- Under "Other", select "Cocoa Touch Unit Test Testing Bundle"

At the end, you should have a file named [Your app name]Tests.swift. In Objective-C, you should have two files named [Your app name]Tests.h and [Your app name]Tests.m instead.

[Your app name]Tests.swift or .m file will include by default :

- A XCTest module import
- A [Your app name]Tests class which extends XCTestCase
- setUp, tearDown, testExample, testPerformanceExample methods

Swift

```
import XCTest

class MyProjectTests: XCTestCase {

    override func setUp() {
        super.setUp()
        // Put setup code here. This method is called before the invocation of each test method in the class.
    }

    override func tearDown() {
        // Put teardown code here. This method is called after the invocation of each test method in the class.
        super.tearDown()
    }
}
```

```

func testExample() {
    // 这是一个功能测试用例的示例。
    // 使用 XCTAssert 及相关函数验证测试是否产生正确结果。
}

func testPerformanceExample() {
    // 这是一个性能测试用例的示例。
    self.measure {
        // 在这里放置你想要测量时间的代码。
    }
}

```

Objective-C

```

#import <XCTest/XCTest.h>

@interface MyProjectTests : XCTestCase

@end

@implementation MyProjectTests

- (void)setUp {
    [super setUp];
    // 在此处放置初始化代码。此方法在类中每个测试方法调用之前被调用。
}

- (void)tearDown {
    // 在这里放置清理代码。此方法在每个测试方法调用后执行。
    [super tearDown];
}

- (void)testExample {
    // 这是一个功能测试用例的示例。
    // 使用 XCTAssert 及相关函数验证测试是否产生正确结果。
}

- (void)testPerformanceExample {
    // 这是一个性能测试用例的示例。
    [self measureBlock:^{
        // 在这里放置你想要测量时间的代码。
    }];
}

```

第133.2节：添加测试方法

根据苹果公司：

测试方法

测试方法是测试类的实例方法，以前缀test开头，不带参数，返回void，例如(void)testColorIsRed()。测试方法执行项目中的代码，如果代码未产生预期结果，则使用一组断言API报告失败。

```

func testExample() {
    // This is an example of a functional test case.
    // Use XCTAssert and related functions to verify your tests produce the correct results.
}

func testPerformanceExample() {
    // This is an example of a performance test case.
    self.measure {
        // Put the code you want to measure the time of here.
    }
}

```

Objective-C

```

#import <XCTest/XCTest.h>

@interface MyProjectTests : XCTestCase

@end

@implementation MyProjectTests

- (void)setUp {
    [super setUp];
    // Put setup code here. This method is called before the invocation of each test method in the
    // class.
}

- (void)tearDown {
    // Put teardown code here. This method is called after the invocation of each test method in the
    // class.
    [super tearDown];
}

- (void)testExample {
    // This is an example of a functional test case.
    // Use XCTAssert and related functions to verify your tests produce the correct results.
}

- (void)testPerformanceExample {
    // This is an example of a performance test case.
    [self measureBlock:^{
        // Put the code you want to measure the time of here.
    }];
}

```

Section 133.2: Adding test methods

According to Apple:

Test Methods

A test method is an instance method of a test class that begins with the prefix test, takes no parameters, and returns void, for example, (void)testColorIsRed(). A test method exercises code in your project and, if that code does not produce the expected result, reports failures using a set of assertion APIs. For

例如，函数的返回值可能会与预期值进行比较，或者测试可能断言对某个类的方法的不当使用会抛出异常。

因此，我们使用“test”作为方法前缀添加测试方法，如：

Swift

```
func testSomething() {  
}
```

Objective-C

```
- (void)testSomething {  
}
```

为了实际测试结果，我们使用XCTAssert()方法，该方法接受一个布尔表达式，如果为真，则标记测试为成功，否则标记为失败。

假设我们在视图控制器类中有一个名为 sum() 的方法，用于计算两个数字的和。为了测试它，我们使用这个方法：

Swift

```
func testSum(){  
    let result = viewController.sum(4, and: 5)  
    XCTAssertEqual(result, 9)  
}
```

Objective-C

```
- (void)testSum {  
int result = [viewController sum:4 and:5];  
    XCTAssertEqual(result, 9);  
}
```

注意

默认情况下，如果视图控制器类中的标签、文本框或其他UI元素最初是在Storyboard文件中创建的，那么测试类无法访问它们。这是因为它们是在视图控制器类的loadView()方法中初始化的，而该方法在测试时不会被调用。调用loadView()及所有其他所需方法的最佳方式是访问我们viewController属性的view属性。你应该在测试UI元素之前添加这行代码：

```
XCTAssertNotNil(viewController.view)
```

第133.3节：编写测试类

```
import XCTest  
@testable import PersonApp  
  
class PersonTests: XCTestCase {  
    func test_completeName() {  
        let person = Person(firstName: "Josh", lastName: "Brown")  
        XCTAssertEqual(person.completeName(), "Josh Brown")  
    }  
}
```

example, a function's return value might be compared against an expected value or your test might assert that improper use of a method in one of your classes throws an exception.

So we add a test method using "test" as the prefix of the method, like:

Swift

```
func testSomething() {  
}
```

Objective-C

```
- (void)testSomething {  
}
```

To actually test the results, we use XCTAssert() method, which takes a boolean expression, and if true, marks the test as succeeded, else it will mark it as failed.

Let's say we have a method in View Controller class called sum() which calculates sum of two numbers. To test it, we use this method:

Swift

```
func testSum(){  
    let result = viewController.sum(4, and: 5)  
    XCTAssertEqual(result, 9)  
}
```

Objective-C

```
- (void)testSum {  
int result = [viewController sum:4 and:5];  
    XCTAssertEqual(result, 9);  
}
```

Note

By default, you can't access label, text box or other UI items of the View Controller class from test class if they are first made in Storyboard file. This is because they are initialized in loadView() method of the View Controller class, and this will not be called when testing. The best way to call loadView() and all other required methods is accessing the view property of our viewController property. You should add this line before testing UI elements:

```
XCTAssertNotNil(viewController.view)
```

Section 133.3: Writing a test class

```
import XCTest  
@testable import PersonApp  
  
class PersonTests: XCTestCase {  
    func test_completeName() {  
        let person = Person(firstName: "Josh", lastName: "Brown")  
        XCTAssertEqual(person.completeName(), "Josh Brown")  
    }  
}
```

```
}
```

现在让我们来讨论这里发生了什么。`import XCTest` 这一行允许我们扩展 `XCTestCase` 并使用 `XCTAssertEqual`（以及其他断言）。扩展 `XCTestCase` 并且测试函数名前缀为 `test` 将确保 Xcode 在运行项目测试时（⌘U 或 Product > Test）自动执行该测试。`@testable import PersonApp` 这一行将导入我们的 `PersonApp` 目标，以便我们可以测试并使用其中的类，例如上面示例中的 `Person`。最后，`XCTAssertEqual` 将确保 `person.completeName()` 等于字符串 "Josh Brown"。

第133.4节：将Storyboard和视图控制器作为实例添加到测试文件中

要开始单元测试，这将在测试文件中进行，并且将测试视图控制器和 Storyboard，我们应当将这两个文件引入测试文件中。

定义视图控制器

Swift

```
var viewController : ViewController!
```

引入Storyboard并初始化视图控制器

将以下代码添加到 `setUp()` 方法中：

Swift

```
let storyboard = UIStoryboard(name: "Main", bundle: nil)
viewController = storyboard.instantiateInitialViewController() as! ViewController
```

Objective-C

```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
viewController = (ViewController *) [storyboard instantiateInitialViewController];
```

这样，你可以编写测试方法，它们就知道去哪里检查错误。在这种情况下，有视图控制器和故事板。

第133.5节：导入可测试的模块

类、结构体、枚举及其所有方法默认都是 `internal` 的。这意味着它们只能从同一个模块访问。测试用例在不同的目标中，这意味着它们在不同的模块中。为了能够访问你想测试的方法，你需要使用 `@testable` 关键字导入要测试的模块。

假设我们有一个名为 `ToDo` 的主模块，我们想为它编写测试。我们会这样导入该模块：

```
@testable import ToDo
```

在包含此导入语句的文件中的所有测试方法现在都可以访问 `ToDo` 模块中所有 `internal` 的类、结构体、枚举及其所有 `internal` 方法。

你绝不应该将包含你想测试的元素的文件添加到测试目标中，因为这可能导致难以调试的错误。

```
}
```

Now let's discuss what's going on here. The `import XCTest` line will allow us to extend `XCTestCase` and use `XCTAssertEqual` (among other assertions). Extending `XCTestCase` and prefixing our test name with `test` will ensure that Xcode automatically runs this test when running the tests in the project (⌘U or **Product > Test**). The `@testable import PersonApp` line will import our `PersonApp` target so we can test and use classes from it, such as the `Person` in our example above. And finally, our `XCTAssertEqual` will ensure that `person.completeName()` is equal to the string "Josh Brown".

Section 133.4: Adding Storyboard and View Controller as instances to test file

To get started with unit testing, which will be done in the tests file and will be testing the View Controller and Storyboard, we should introduce these two files to the test file.

Defining the View Controller

Swift

```
var viewController : ViewController!
```

Introducing the Storyboard and initializing the View Controller

Add this code to the `setUp()` method:

Swift

```
let storyboard = UIStoryboard(name: "Main", bundle: nil)
viewController = storyboard.instantiateInitialViewController() as! ViewController
```

Objective-C

```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
viewController = (ViewController *) [storyboard instantiateInitialViewController];
```

This way, you could write test methods, and they will know where to check for errors. In this case, there are View Controller and the Storyboard.

Section 133.5: Import a module that it can be tested

Classes, structs, enums and all their methods are `internal` by default. This means they can be only accessed from the same module. The test cases are in a different target and this means they are in a different module. To be able to access the method you want to test, you need to import the module to be tested using the `@testable` keyword.

Let's say we have a main module called `ToDo` and we want to write tests for it. We would import that module like this:

```
@testable import ToDo
```

All test methods in the file with this import statement can now access all `internal` classes, structs, enums and all their `internal` methods of the `ToDo` module.

You should never add the files with the elements you want to test to the test target because that can lead to hard to debug errors.

第133.6节：触发视图加载和显示

视图加载

在视图控制器的测试中，有时你需要触发`loadView()`或`viewDidLoad()`的执行。这可以通过访问视图来实现。假设你在测试中有一个名为`sut`（被测试系统）的视图控制器实例，那么代码如下：

```
XCTAssertNotNil(sut.view)
```

视图显示

你也可以通过添加以下代码来触发`viewWillAppear(_:)`和`viewDidAppear(_:)`方法：

```
sut.beginAppearanceTransition(true, animated: true)  
sut.endAppearanceTransition()
```

第133.7节：开始测试

测试特定方法

要测试特定方法，点击方法定义旁边的方框。

测试所有方法

要测试所有方法，请点击类定义旁边的方框。

查看测试结果

如果定义旁边有绿色勾号，测试已成功。



如果定义旁边有红色叉号，测试失败。



运行所有测试

产品 -> 测试 或 Cmd + U

它将运行所有测试目标中的所有测试！

Section 133.6: Trigger view loading and appearance

View loading

In a test for a view controller you want sometimes to trigger the execution of `loadView()` or `viewDidLoad()`. This can be done by accessing the view. Let's say you have view controller instance in your test called `sut` (system under test), then the code would look like this:

```
XCTAssertNotNil(sut.view)
```

View appearance

You can also trigger the methods `viewWillAppear(_:)` and `viewDidAppear(_:)` by adding the following code:

```
sut.beginAppearanceTransition(true, animated: true)  
sut.endAppearanceTransition()
```

Section 133.7: Start Testing

Testing a specific method

To test a specific method, click the square next to the method definition.

Testing all methods

To test all methods, click the square next to the class definition.

See the testing result

If there is a green check next to the definition, the test has succeeded.



If there is a red cross next to the definition, the test has failed.



Running all tests

Product -> Test OR Cmd + U

It will run all the tests from all the test targets!

第134章：AVPlayer和AVPlayerViewController

第134.1节：使用AVPlayer和AVPlayerLayer播放媒体

Objective C

```
NSURL *url = [NSURL URLWithString:@"YOUR URL"];
AVPlayer *player = [AVPlayer playerWithURL:videoURL];
AVPlayerLayer *playerLayer = [AVPlayerLayer playerLayerWithPlayer:player];
playerLayer.frame = self.view.bounds;
[self.view.layer addSublayer:playerLayer];
[player play];
```

Swift

```
let url = NSURL(string: "YOUR URL")
let player = AVPlayer(URL: videoURL!)
let playerLayer = AVPlayerLayer(player: player)
playerLayer.frame = self.view.bounds
self.view.layer.addSublayer(playerLayer)
player.play()
```

第134.2节：使用AVPlayerViewController播放媒体

Objective-C

```
NSURL *url = [[NSURL alloc] initWithString:@"YOUR URL"]; // url可以是远程或本地的
AVPlayer *player = [AVPlayer playerWithURL:url];
// 创建播放器视图控制器

AVPlayerViewController *controller = [[AVPlayerViewController alloc] init];
[self presentViewController:controller animated:YES completion:nil];
controller.player = player;
[player play];
```

Swift

```
let player = AVPlayer(URL: url) // url可以是远程或本地的

let playerViewController = AVPlayerViewController()
// 创建播放器视图控制器
playerViewController.player = player
self.presentViewController(playerViewController, animated: true) {
    playerViewController.player!.play()
}
```

第134.3节：AVPlayer示例

```
AVPlayer *avPlayer = [AVPlayer playerWithURL:[NSURL URLWithString:@"YOUR URL"]];

AVPlayerViewController *avPlayerCtrl = [[AVPlayerViewController alloc] init];
avPlayerCtrl.view.frame = self.view.frame;
avPlayerCtrl.player = avPlayer;
avPlayerCtrl.delegate = self;
```

Chapter 134: AVPlayer and AVPlayerViewController

Section 134.1: Playing Media using AVPlayer and AVPlayerLayer

Objective C

```
NSURL *url = [NSURL URLWithString:@"YOUR URL"];
AVPlayer *player = [AVPlayer playerWithURL:videoURL];
AVPlayerLayer *playerLayer = [AVPlayerLayer playerLayerWithPlayer:player];
playerLayer.frame = self.view.bounds;
[self.view.layer addSublayer:playerLayer];
[player play];
```

Swift

```
let url = NSURL(string: "YOUR URL")
let player = AVPlayer(URL: videoURL!)
let playerLayer = AVPlayerLayer(player: player)
playerLayer.frame = self.view.bounds
self.view.layer.addSublayer(playerLayer)
player.play()
```

Section 134.2: Playing Media Using AVPlayerViewController

Objective-C

```
NSURL *url = [[NSURL alloc] initWithString:@"YOUR URL"]; // url can be remote or local
AVPlayer *player = [AVPlayer playerWithURL:url];
// create a player view controller

AVPlayerViewController *controller = [[AVPlayerViewController alloc] init];
[self presentViewController:controller animated:YES completion:nil];
controller.player = player;
[player play];
```

Swift

```
let player = AVPlayer(URL: url) // url can be remote or local

let playerViewController = AVPlayerViewController()
// creating a player view controller
playerViewController.player = player
self.presentViewController(playerViewController, animated: true) {
    playerViewController.player!.play()
}
```

Section 134.3: AVPlayer Example

```
AVPlayer *avPlayer = [AVPlayer playerWithURL:[NSURL URLWithString:@"YOUR URL"]];
```

```
AVPlayerViewController *avPlayerCtrl = [[AVPlayerViewController alloc] init];
avPlayerCtrl.view.frame = self.view.frame;
avPlayerCtrl.player = avPlayer;
avPlayerCtrl.delegate = self;
```

```
[avPlayer play];
[self presentViewController:avPlayerCtrl animated:YES completion:nil]
```

```
[avPlayer play];
[self presentViewController:avPlayerCtrl animated:YES completion:nil]
```

第135章：iOS中的深度链接

第135.1节：为自己的应用添加URL方案

假设你正在开发一个名为MyTasks的应用，你想允许通过传入的URL来创建一个带有标题和正文的新任务。你设计的URL可能看起来像这样：

```
mytasks://create?title=hello&body=world
```

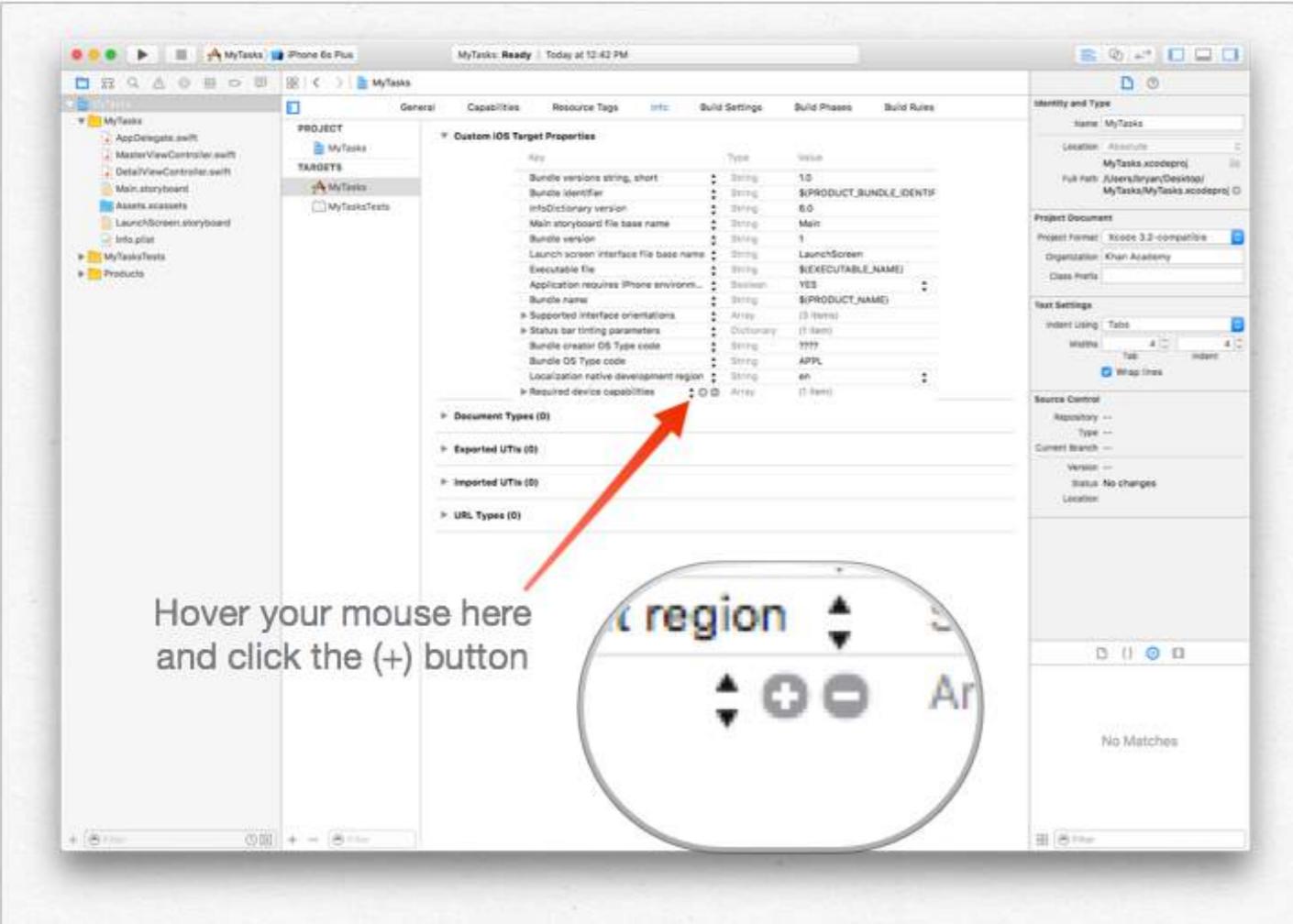
(当然，text 和 body 参数用于填充我们正在创建的任务！)

将此 URL 方案添加到您的项目中的主要步骤如下：

1. 在应用的 Info.plist 文件中注册一个 URL 方案，这样系统才能知道何时将 URL 路由到您的应用。
2. 向您的 UIApplicationDelegate 添加一个函数，用于接收和处理传入的 URL。
3. 执行打开该 URL 时需要进行的任何任务。

第一步：在 Info.plist 中注册 URL 方案：

首先，我们需要在 Info.plist 文件中添加一个“URL Types”条目。点击这里的（+）按钮：



...然后输入您的应用的唯一标识符，以及您想使用的 URL 方案。要具体！您不想让 URL 方案与其他应用的实现冲突。这里宁可长一点也不要太短！

Chapter 135: Deep Linking in iOS

Section 135.1: Adding a URL scheme to your own app

Let's say you're working on an app called MyTasks, and you want to allow inbound URLs to create a new task with a title and a body. The URL you're designing might look something like this:

```
mytasks://create?title=hello&body=world
```

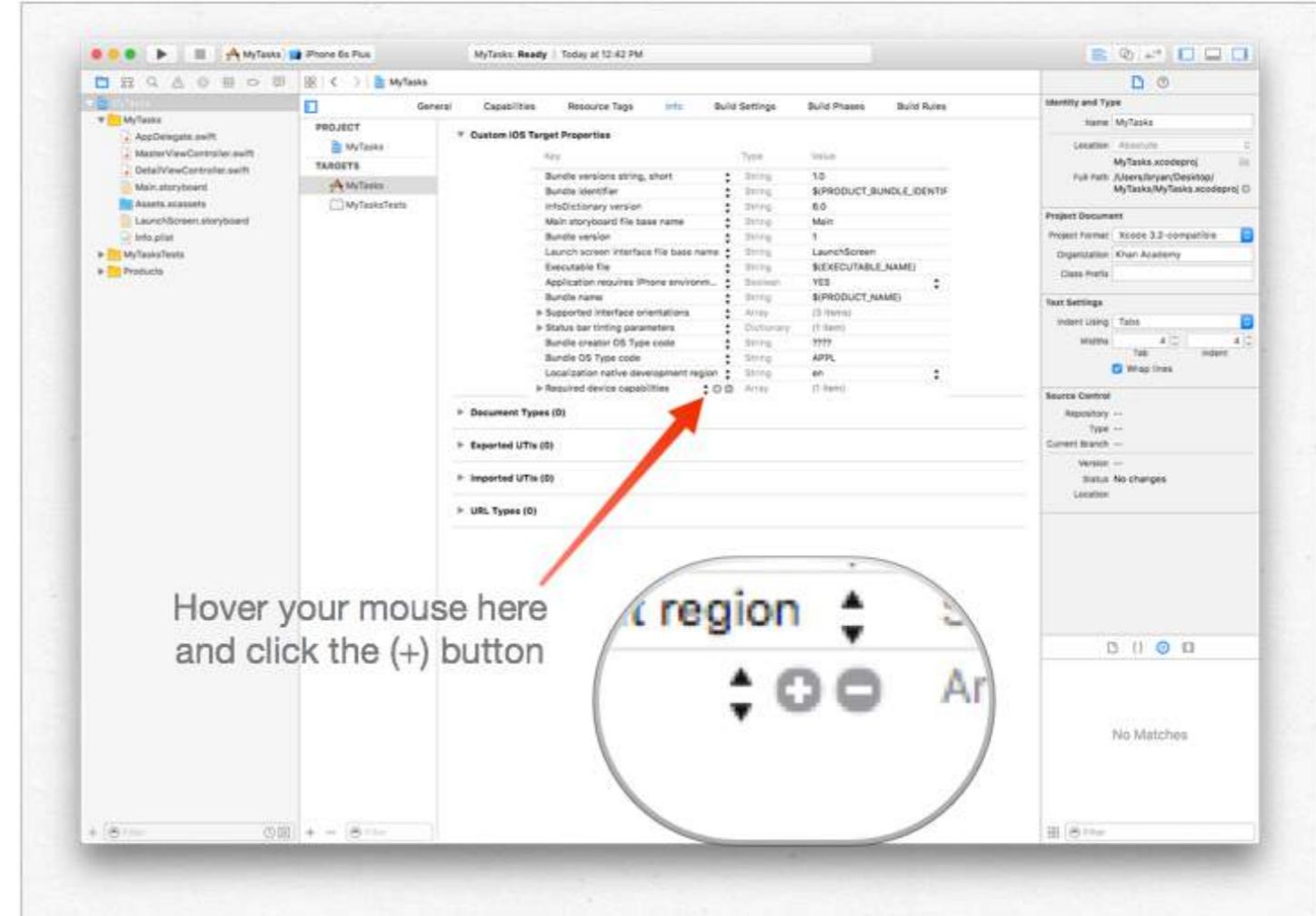
(Of course, the text and body parameters are used to populate our task that we're creating!)

Here are the Big Steps to adding this URL scheme to your project:

1. Register a URL scheme in your app's `Info.plist` file, so the system knows when to route a URL to your app.
2. Add a function to your `UIApplicationDelegate` that accepts and handles incoming URLs.
3. Perform whatever task needs to occur when that URL is opened.

Step One: Register a URL scheme in Info.plist:

First, we need to add a "URL Types" entry to our Info.plist file. Click the (+) button here:



...then enter a unique identifier for your app, as well as the URL scheme you want to use. Be specific! You don't want the URL scheme to conflict with another app's implementation. Better to be too-long here than too-short!

▼ URL types	Array	(5 items)
▼ Item 0	Dictionary	(2 items)
URL identifier	String	com.mycompany
▼ URL Schemes	Array	(1 item)
Item 0	String	mytasks
▼ Item 1	Dictionary	(1 item)

步骤二：处理 UIApplicationDelegate 中的 URL

我们需要在我们的UIApplicationDelegate上实现application:openURL:options:。我们将检查传入的URL，看看是否有可以执行的操作！

一种实现方式如下：

```
func application(app: UIApplication, openURL url: NSURL, options: [String : AnyObject]) -> Bool {
    if url.scheme == "mytasks" && url.host == "create" {
        let title = // 使用你选择的方法从URL的查询中获取标题
        let body = // 使用你选择的方法从URL的查询中获取正文
        self.rootViewController.createTaskWithTitle(title, body: body)
        return true
    }

    返回 false
}
```

第三步：根据URL执行任务。

当用户通过URL打开你的应用时，他们可能期望某些事情发生。也许是导航到某个内容，也许是创建一个新项目——在这个例子中，我们将创建一个新的任务！

在上面的代码中，我们可以看到调用了self.rootViewController.createTaskWithTitle(:body:)——所以假设你的AppDelegate有一个指向其根视图控制器的指针，并且该控制器正确实现了该函数，那么你就准备好了！

第135.2节：基于URL scheme打开应用

要使用定义的URL scheme odolist://打开应用

Objective-C

```
NSURL *myURL = [NSURL URLWithString:@"todolist://there/is/something/to/do"];
[[UIApplication sharedApplication] openURL:myURL];
```

Swift

```
let stringURL = "todolist://there/is/something/to/do"
if let url = NSURL(string: stringURL) {
    UIApplication.shared().openURL(url)
}
```

HTML

```
<a href="todolist://there/is/something/to/do">New SMS Message</a>
```

注意：检查链接是否可以打开很有用，否则可以向用户显示适当的信息。

▼ URL types	Array	(5 items)
▼ Item 0	Dictionary	(2 items)
URL identifier	String	com.mycompany
▼ URL Schemes	Array	(1 item)
Item 0	String	mytasks
▼ Item 1	Dictionary	(1 item)

Step Two: Handle the URL in the UIApplicationDelegate

We need to implement application:openURL:options: on our `UIApplicationDelegate`. We'll inspect the incoming URL and see if there's an action we can take!

One implementation would be this:

```
func application(app: UIApplication, openURL url: NSURL, options: [String : AnyObject]) -> Bool {
    if url.scheme == "mytasks" && url.host == "create" {
        let title = // get the title out of the URL's query using a method of your choice
        let body = // get the title out of the URL's query using a method of your choice
        self.rootViewController.createTaskWithTitle(title, body: body)
        return true
    }

    return false
}
```

Step Three: Perform a task depending on the URL.

When a user opens your app via a URL, they probably expected *something* to happen. Maybe that's navigating to a piece of content, maybe that's creating a new item - in this example, we're going to create a new task in the app!

In the above code, we can see a call to `self.rootViewController.createTaskWithTitle(:body:)` - so assuming that your AppDelegate has a pointer to its root view controller which implements the function properly, you're all set!

Section 135.2: Opening an app based on its URL scheme

To open an app with defined URL scheme `todolist://`:

Objective-C

```
NSURL *myURL = [NSURL URLWithString:@"todolist://there/is/something/to/do"];
[[UIApplication sharedApplication] openURL:myURL];
```

Swift

```
let stringURL = "todolist://there/is/something/to/do"
if let url = NSURL(string: stringURL) {
    UIApplication.shared().openURL(url)
}
```

HTML

```
<a href="todolist://there/is/something/to/do">New SMS Message</a>
```

Note: It's useful to check if link can be opened to otherwise display an appropriate message to the user.

这可以使用 canOpenURL: 方法来完成。

第135.3节：为您的应用设置深度链接

为您的应用设置深度链接很简单。您只需要一个用于打开应用的小型URL。

按照以下步骤为您的应用设置深度链接。

1. 让我们创建一个项目并命名为 DeepLinkPOC。
2. 现在选择你的项目目标。
3. 选择目标后，点击“信息”标签。
4. 向下滚动到底部，直到看到“URL 类型”选项
5. 点击“+”选项。
6. 你会看到URL 方案，添加一个字符串，用于打开你的应用。我们添加“DeepLinking”到 URL 方案中。

因此，要打开你的应用，可以在 Safari 中输入"DeepLinking://"来启动。你的深度链接字符串格式如下。

[scheme]://[host]/[path] --> DeepLinking://path/Page1

其中，Scheme（方案）："DeepLinking" Host（主机）："path" path（路径）："Page1"

注意：即使不添加主机和路径，也能启动应用，所以不用担心。但你可以添加主机和路径，以便应用启动后额外重定向到特定页面。

7. 现在将以下方法添加到你的 AppDelegate 中。

Swift:

```
func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?, annotation: AnyObject) -> Bool
```

Objective-c :

```
-(BOOL)application:(UIApplication *)application  
    openURL:(NSURL *)url  
sourceApplication:(NSString *)sourceApplication  
annotation:(id)annotation
```

每当你应用通过你为其设置的深度链接字符串启动时，都会调用上述方法。

8. 现在是安装你的应用的时候了，但在你直接点击运行按钮之前，先对方案的应用启动方法做一个小改动。
 - 选择并编辑你的方案，如下所示

This can be done using canOpenURL : method.

Section 135.3: Setting up deeplink for your app

Setting up deep-linking for your app is easy. You just need a small url using which you want to open your app.

Follow the steps to set up deep-linking for your app.

1. Lets create a project and name it DeepLinkPOC.
2. Now select your project target.
3. After selecting target,select the 'info' tab.
4. Scroll down to the bottom until you see an option of **URL Types**
5. Click '+' option.
6. You will see **URL schemes** add a string using which you want to open your app.Lets add "**DeepLinking**" in URL schemes.

So, to open your app you can launch it by typing "**DeepLinking://**" into your safari. Your deep-linking string has following format.

[scheme]://[host]/[path] --> DeepLinking://path/Page1

where, Scheme : "DeepLinking" Host : "path" path : "Page1"

Note : Even if don't add host and path it will launch the app, so no worries. But you can add host and path to additionally redirect to particular page after application launch.

7. Now add following method to your appdelegate.

Swift:

```
func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?, annotation: AnyObject) -> Bool
```

Objective-c :

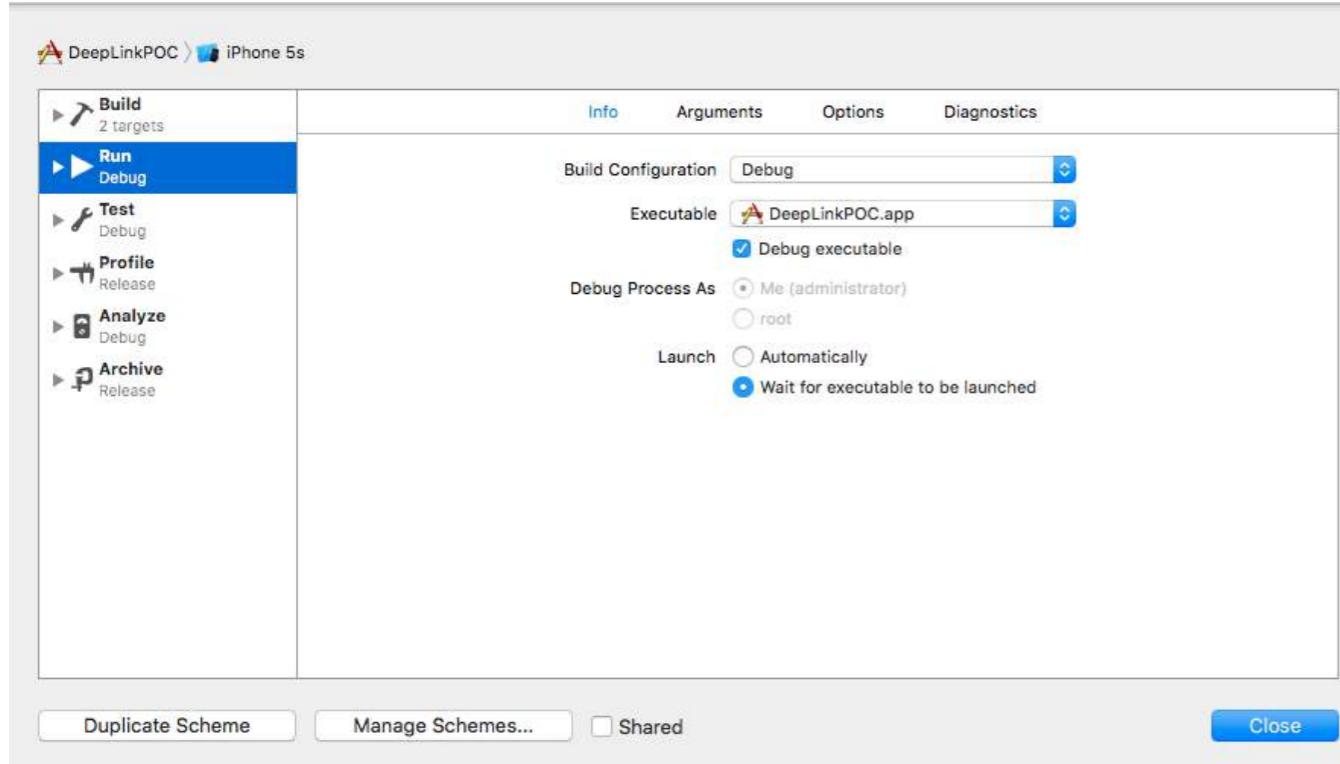
```
-(BOOL)application:(UIApplication *)application  
    openURL:(NSURL *)url  
sourceApplication:(NSString *)sourceApplication  
annotation:(id)annotation
```

The above method is called whenever your app is launched using a deep-linking string you set for your app.

8. Now is the time to install your app but wait before you directly jump to run button. Lets do a small change in scheme's app-launch method.
 - Select and edit your scheme as



- 更改其启动类型并关闭



9. 现在点击运行按钮（如果需要，可以在 `didFinishLaunchingWithOptions` 和 `openURL` 方法中添加断点以观察值）

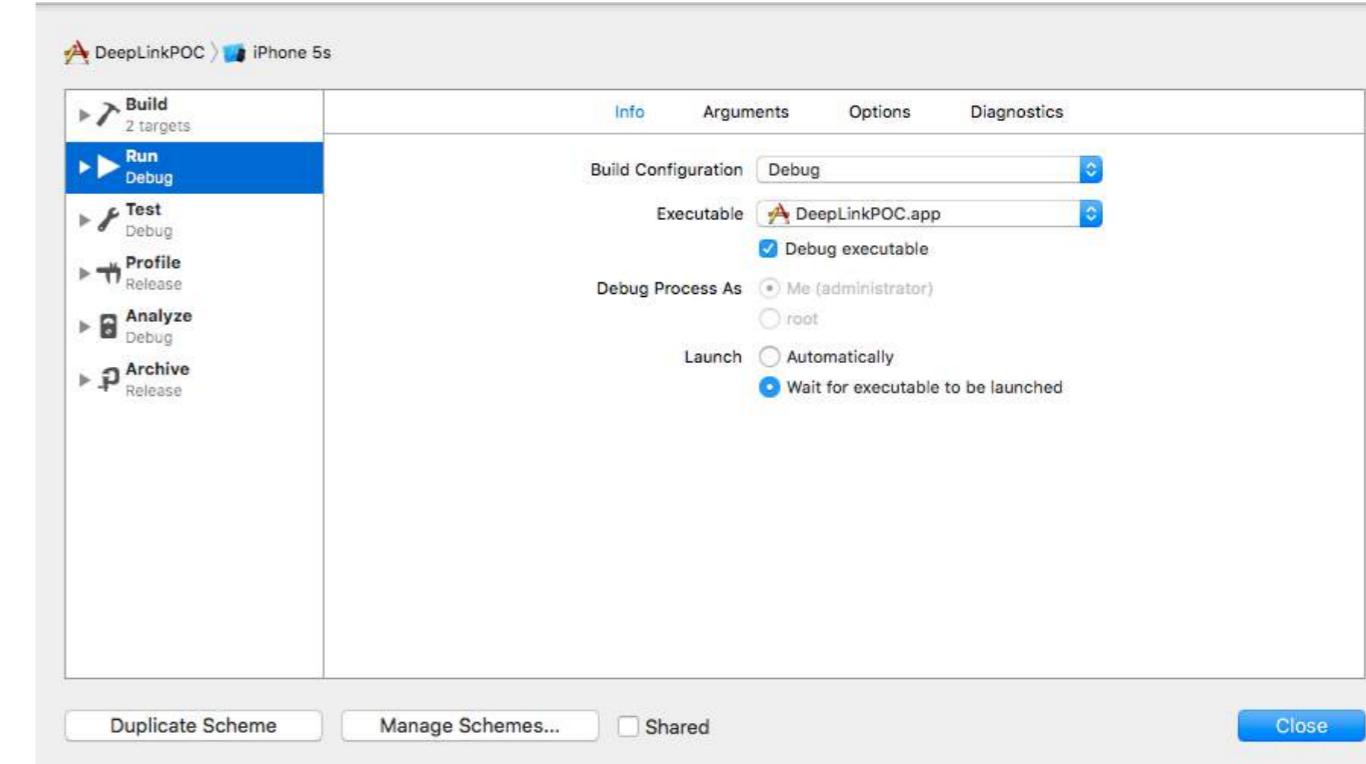
10. 你会看到一条消息“等待 DeepLinkPOC（或你的应用名称）启动”。

11. 打开 Safari，在搜索栏输入“DeepLinking://”，这将显示提示“在 DeepLinkPOC 中打开此页面，点击打开以启动你的应用。

希望你已经了解了如何为你的应用设置深度链接 :)



- change its launch type and close



9. Now click the Run button (if you want you can add breakpoint to your `didFinishLaunchingWithOptions` and `openURL` methods to observe values)

10. You'll see a message "Waiting for DeepLinkPOC(or your app name) to launch".

11. Open safari and type in "**DeepLinking://**" into the search bar this will show prompt "open this page in DeepLinkPOC" click open to launch your app.

Hope you got to know how to set up deep-linking for your app :)

第136章：核心图形

第136.1节：创建核心图形上下文

核心图形上下文

核心图形上下文是一个画布，我们可以在上面绘图并设置一些属性，比如线条粗细。

创建上下文

要创建上下文，我们使用`UIGraphicsBeginImageContextWithOptions()` C函数。然后，当绘制完成后，调用`UIGraphicsEndImageContext()`来结束上下文：

Swift

```
let size = CGSize(width: 256, height: 256)  
UIGraphicsBeginImageContextWithOptions(size, false, 0)  
  
let context = UIGraphicsGetCurrentContext()  
  
// 在这里绘制代码  
  
UIGraphicsEndImageContext()
```

Objective-C

```
CGSize size = [CGSize width:256 height:256];  
  
UIGraphicsBeginImageContextWithOptions(size, NO, 0);  
  
CGContext *context = UIGraphicsGetCurrentContext();  
  
// 在这里绘制代码  
  
UIGraphicsEndImageContext();
```

在上面的代码中，我们向`UIGraphicsBeginImageContextWithOptions()`函数传递了3个参数：

- 一个`CGSize`对象，用于存储上下文（画布）的整体大小
- 一个布尔值，如果为`true`，上下文将是不透明的
- 一个整数值，用于设置缩放比例（1表示非视网膜屏幕，2表示视网膜屏幕，3表示视网膜高清屏幕）。如果设置为0，系统会根据目标设备自动处理缩放比例。

第136.2节：向用户呈现绘制的画布

Swift

```
let image = UIGraphicsGetImageFromCurrentImageContext()  
imageView.image = image //假设 imageView 是一个有效的 UIImageView 对象
```

Objective-C

```
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
```

Chapter 136: Core Graphics

Section 136.1: Creating a Core Graphics Context

Core Graphics context

A Core Graphics context is a canvas which we can draw in it and set some properties like the line thickness.

Making a context

To make a context, we use the `UIGraphicsBeginImageContextWithOptions()` C function. Then, when we are done with drawing, we just call `UIGraphicsEndImageContext()` to end the context:

Swift

```
let size = CGSize(width: 256, height: 256)  
UIGraphicsBeginImageContextWithOptions(size, false, 0)  
  
let context = UIGraphicsGetCurrentContext()  
  
// drawing code here  
  
UIGraphicsEndImageContext()
```

Objective-C

```
CGSize size = [CGSize width:256 height:256];  
  
UIGraphicsBeginImageContextWithOptions(size, NO, 0);  
  
CGContext *context = UIGraphicsGetCurrentContext();  
  
// drawing code here  
  
UIGraphicsEndImageContext();
```

In the code above, we passed 3 parameters to the `UIGraphicsBeginImageContextWithOptions()` function:

- 一个`CGSize`对象，用于存储上下文（画布）的整体大小
- 一个布尔值，如果为`true`，上下文将是不透明的
- 一个整数值，用于设置缩放比例（1表示非视网膜屏幕，2表示视网膜屏幕，3表示视网膜高清屏幕）。如果设置为0，系统会根据目标设备自动处理缩放比例。

Section 136.2: Presenting the Drawn Canvas to User

Swift

```
let image = UIGraphicsGetImageFromCurrentImageContext()  
imageView.image = image //assuming imageView is a valid UIImageView object
```

Objective-C

```
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
```

```
imageView.image = image; //假设 imageView 是一个有效的 UIImageView 对象
```

```
imageView.image = image; //assuming imageView is a valid UIImageView object
```

第137章 : Segue (界面跳转)

第137.1节 : 使用 Segue 在导航栈中向后导航

回退 Segue

回退 Segue 让你可以“回退”导航栈并指定返回的目标。

此函数的签名是 Interface Builder 识别它的关键。它必须有一个返回值为

IBAction, 且带有一个参数 **UIStoryboardSegue** 函数名无关紧要。

实际上, 该函数甚至不必执行任何操作。它只是作为一个标记, 指明哪个

UIViewController 是回退 Segue 的目标。[source][1]

解除segue所需的签名

Objective C :

```
- (IBAction)prepareForUnwind:(UIStoryboardSegue *)segue {  
}
```

Swift:

```
@IBAction func prepareForUnwind(segue: UIStoryboardSegue) {  
}
```

第137.2节 : 概述

摘自苹果文档 :

UIStoryboardSegue对象负责执行两个视图控制器之间的视觉过渡。此外, segue对象用于准备从一个视图控制器到另一个视图控制器的过渡。segue对象包含有关过渡中涉及的视图控制器的信息。

当segue被触发但视觉过渡尚未发生时, 故事板运行时会调用当前视图控制器的prepareForSegue:sender:方法, 以便将任何所需数据传递给即将显示的视图控制器。

属性

Swift

```
sourceViewController: UIViewController {get}  
destinationViewController: UIViewController {get}  
identifier: String? {get}
```

参考文献 :

- [UIViewController 类参考](#)
- [UIStoryboardSegue 类参考](#)

Chapter 137: Segues

Section 137.1: Using Segues to navigate backwards in the navigation stack

Unwind Segues

Unwind Segues give you a way to “unwind” the navigation stack and specify a destination to go back to. The signature of this function is key to Interface Builder recognizing it. **It must have a return value of IBAction and take one parameter of UIStoryboardSegue**. The name of the function does not matter. In fact, the function does not even have to do anything. It's just there as a marker of which UIViewController is the destination of the Unwind Segue. [source][1]

Required signature of an unwind segue

Objective C:

```
- (IBAction)prepareForUnwind:(UIStoryboardSegue *)segue {  
}
```

Swift:

```
@IBAction func prepareForUnwind(segue: UIStoryboardSegue) {  
}
```

Section 137.2: An Overview

From the Apple documentation:

A UIStoryboardSegue object is responsible for **performing the visual transition between two view controllers**. In addition, segue objects are used to prepare for the transition from one view controller to another. **Segue objects contain information about the view controllers involved in a transition**. When a segue is triggered, but before the visual transition occurs, the storyboard runtime calls the current view controller's prepareForSegue:sender: method so that it can pass any needed data to the view controller that is about to be displayed.

Attributes

Swift

```
sourceViewController: UIViewController {get}  
destinationViewController: UIViewController {get}  
identifier: String? {get}
```

References:

- [UIViewController Class Reference](#)
- [UIStoryboardSegue Class Reference](#)

第137.3节：在触发Segue之前准备你的视图控制器

PrepareForSegue:

```
func prepareForSegue(_ segue:UIStoryboardSegue, sender sender:AnyObject?)
```

通知视图控制器即将执行segue

参数

segue: segue对象。

sender: 初始化segue的对象。

Swift示例

如果segue的标识符是"SomeSpecificIdentifier", 则执行某个任务

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {  
    if segue.identifier == "SomeSpecificIdentifier" {  
        // 执行特定任务  
    }  
}
```

第137.4节：决定是否执行调用的Segue

ShouldPerformSegueWithIdentifier:

```
func shouldPerformSegueWithIdentifier(_ identifier:String, sender sender:AnyObject?) -> Bool
```

确定是否应该执行具有指定标识符的segue。

参数

Identifier: 触发segue的标识字符串

Sender: 初始化segue的对象。

Swift示例

仅当标识符为"SomeSpecificIdentifier"时执行segue

```
override func shouldPerformSegueWithIdentifier(identifier:String, sender: AnyObject?) -> Bool {  
    if identifier == "SomeSpecificIdentifier" {  
        return true  
    }  
    返回 false  
}
```

第137.5节：以编程方式触发Segue

PerformSegueWithIdentifier:

GoalKicker.com – iOS® 开发者专业笔记

Section 137.3: Preparing your view controller before a triggering a Segue

PrepareForSegue:

```
func prepareForSegue(_ segue: UIStoryboardSegue, sender sender: AnyObject?)
```

Notifies the view controller that a segue is about to be performed

Parameters

segue: The segue object.

sender: The object that initialized the segue.

Example in Swift

Perform a task if the identifier of the segue is "SomeSpecificIdentifier"

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {  
    if segue.identifier == "SomeSpecificIdentifier" {  
        // Do specific task  
    }  
}
```

Section 137.4: Deciding if an invoked Segue should be performed

ShouldPerformSegueWithIdentifier:

```
func shouldPerformSegueWithIdentifier(_ identifier:String, sender sender: AnyObject?) -> Bool
```

Determines whether the segue with the specified identifier should be performed.

Parameters

Identifier: String that identifies the triggered segue

Sender: The object that initialized the segue.

Example in Swift

Only perform segue if the identifier is "SomeSpecificIdentifier"

```
override func shouldPerformSegueWithIdentifier(identifier:String, sender: AnyObject?) -> Bool {  
    if identifier == "SomeSpecificIdentifier" {  
        return true  
    }  
    return false  
}
```

Section 137.5: Trigger Segue Programmatically

PerformSegueWithIdentifier:

GoalKicker.com – iOS® Developer Notes for Professionals

```
func performSegueWithIdentifier(_ identifier:String, sender sender:AnyObject?)
```

从当前视图控制器的故事板文件中启动具有指定标识符的segue

参数

标识符: 用于识别触发的segue的字符串

发送者: 将启动segue的对象。

Swift示例

从表视图行选择执行标识符为"SomeSpecificIdentifier"的segue：

```
func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {  
    performSegueWithIdentifier("SomeSpecificIdentifier", sender: indexPath.item)  
}
```

```
func performSegueWithIdentifier(_ identifier:String, sender sender:AnyObject?)
```

Initiates the segue with the specified identifier from the current view controller's storyboard file

Parameters

Identifier: String that identifies the triggered segue

Sender: The object that will initiate the segue.

Example in Swift

Performing a segue with identifier "SomeSpecificIdentifier" from a table view row selection:

```
func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {  
    performSegueWithIdentifier("SomeSpecificIdentifier", sender: indexPath.item)  
}
```

第138章：EventKit

第138.1节：访问不同类型的日历

访问日历数组

要访问EKCalendar数组，我们使用calendarsForEntityType方法：

Swift

```
let calendarsArray = eventStore.calendarsForEntityType(EKEntityType.Event) as! [EKCalendar]
```

遍历日历

只需使用一个简单的for循环：

Swift

```
for calendar in calendarsArray{  
    //...  
}
```

访问日历标题和颜色

Swift

```
let calendarColor = UIColor(CGColor: calendar.CGColor)  
let calendarTitle = calendar.title
```

Objective-C

```
UIColor *calendarColor = [UIColor initWithCGColor: calendar.CGColor];  
NSString *calendarTitle = calendar.title;
```

第138.2节：请求权限

您的应用程序在未获得权限的情况下无法访问您的提醒事项和日历。相反，它必须向用户显示警告，请求用户授予应用访问事件的权限。

首先，导入EventKit框架：

Swift

```
import EventKit
```

Objective-C

```
#import <EventKit/EventKit.h>
```

创建一个EKEEventStore

然后，我们创建一个EKEEventStore对象。这个对象用于访问日历和提醒事项数据：

Swift

```
let eventStore = EKEEventStore()
```

Objective-C

```
EKEEventStore *eventStore = [[EKEEventStore alloc] init];
```

注意

Chapter 138: EventKit

Section 138.1: Accessing different types of calendars

Accessing the array of calendars

To access the array of EKCalendars, we use the calendarsForEntityType method:

Swift

```
let calendarsArray = eventStore.calendarsForEntityType(EKEntityType.Event) as! [EKCalendar]
```

Iterating through calendars

Just use a simple for loop:

Swift

```
for calendar in calendarsArray{  
    //...  
}
```

Accessing the calendar title and color

Swift

```
let calendarColor = UIColor(CGColor: calendar.CGColor)  
let calendarTitle = calendar.title
```

Objective-C

```
UIColor *calendarColor = [UIColor initWithCGColor: calendar.CGColor];  
NSString *calendarTitle = calendar.title;
```

Section 138.2: Requesting Permission

Your app can't access your reminders and your calendar without permission. Instead, it must show an alert to user, requesting him/her to grant access to events for the app.

To get started, import the EventKit framework:

Swift

```
import EventKit
```

Objective-C

```
#import <EventKit/EventKit.h>
```

Making an EKEEventStore

Then, we make an EKEEventStore object. This is the object from which we can access calendar and reminders data:

Swift

```
let eventStore = EKEEventStore()
```

Objective-C

```
EKEEventStore *eventStore = [[EKEEventStore alloc] init];
```

Note

每次需要访问日历时都创建一个EKEventStore对象效率不高。尽量只创建一次并在代码中到处使用它。

Making an EKEventStore object every time we need to access calendar is not efficient. Try to make it once and use it everywhere in your code.

检查可用性

可用性有三种不同状态：已授权、已拒绝和未确定。未确定表示应用程序需要请求访问权限。

要检查可用性，我们使用EKEventStore对象的authorizationStatusForEntityType()方法：

Swift

```
switch EKEventStore.authorizationStatusForEntityType(EKEntityTypeEvent){  
    case .Authorized: //...  
    case .Denied: //...  
    case .NotDetermined: //...  
    default: break  
}
```

Objective-C

```
switch ([EKEventStore authorizationStatusForEntityType:EKEntityTypeEvent]) {  
    case EKAuthorizationStatus.Authorized:  
        //...  
        break;  
    case EKAuthorizationStatus.Denied:  
        //...  
        break;  
    case EKAuthorizationStatus.NotDetermined:  
        //...  
        break;  
    default:  
        break;  
}
```

请求权限

将以下代码放入NotDetermined情况：

Swift

```
eventStore.requestAccessToEntityType(EKEntityTypeEvent, completion: { [weak self]  
(userGrantedAccess, _) -> Void in  
    if userGrantedAccess{  
        //访问日历  
    }  
}
```

Checking Availability

Availability has three different status: Authorized, Denied and Not Determined. Not Determined means the app needs to grant access.

To check availability, we use authorizationStatusForEntityType() method of the EKEventStore object:

Swift

```
switch EKEventStore.authorizationStatusForEntityType(EKEntityTypeEvent){  
    case .Authorized: //...  
    case .Denied: //...  
    case .NotDetermined: //...  
    default: break  
}
```

Objective-C

```
switch ([EKEventStore authorizationStatusForEntityType:EKEntityTypeEvent]) {  
    case EKAuthorizationStatus.Authorized:  
        //...  
        break;  
    case EKAuthorizationStatus.Denied:  
        //...  
        break;  
    case EKAuthorizationStatus.NotDetermined:  
        //...  
        break;  
    default:  
        break;  
}
```

Requesting Permission

Put the following code in NotDetermined case:

Swift

```
eventStore.requestAccessToEntityType(EKEntityTypeEvent, completion: { [weak self]  
(userGrantedAccess, _) -> Void in  
    if userGrantedAccess{  
        //access calendar  
    }  
}
```

第138.3节：添加事件

创建事件对象

Swift

```
var event = EKEvent(eventStore: eventStore)
```

Objective-C

```
EKEvent *event = [EKEvent initWithEventStore:eventStore];
```

设置相关日历、标题和日期

Swift

Section 138.3: Adding an event

Creating the event object

Swift

```
var event = EKEvent(eventStore: eventStore)
```

Objective-C

```
EKEvent *event = [EKEvent initWithEventStore:eventStore];
```

Setting related calendar, title and dates

Swift

```
event.calendar = calendar
event.title = "事件标题"
event.startDate = startDate //假设startDate是有效的NSDate对象
event.endDate = endDate //假设endDate是有效的NSDate对象
```

将事件添加到日历

Swift

```
try {
    do eventStore.saveEvent(event, span: EKSpan.ThisEvent)
} catch let error as NSError {
    //错误
}
```

Objective-C

```
NSError *error;
BOOL *result = [eventStore saveEvent:event span:EKSpanThisEvent error:&error];
if (result == NO){
    //错误
}
```

```
event.calendar = calendar
event.title = "Event Title"
event.startDate = startDate //assuming startDate is a valid NSDate object
event.endDate = endDate //assuming endDate is a valid NSDate object
```

Adding event to calendar

Swift

```
try {
    do eventStore.saveEvent(event, span: EKSpan.ThisEvent)
} catch let error as NSError {
    //error
}
```

Objective-C

```
NSError *error;
BOOL *result = [eventStore saveEvent:event span:EKSpanThisEvent error:&error];
if (result == NO){
    //error
}
```

第139章：SiriKit

第139.1节：向应用添加Siri扩展

要在您的应用中集成Siri功能，您应当像创建iOS 10小组件（旧版今日视图扩展）或自定义键盘时那样添加扩展。

增加能力

1- 在项目设置中，选择你的 iOS 应用目标并进入功能（Capabilities）标签页

2- 启用 Siri 功能

添加扩展

1- 进入 文件 -> 新建 -> 目标...

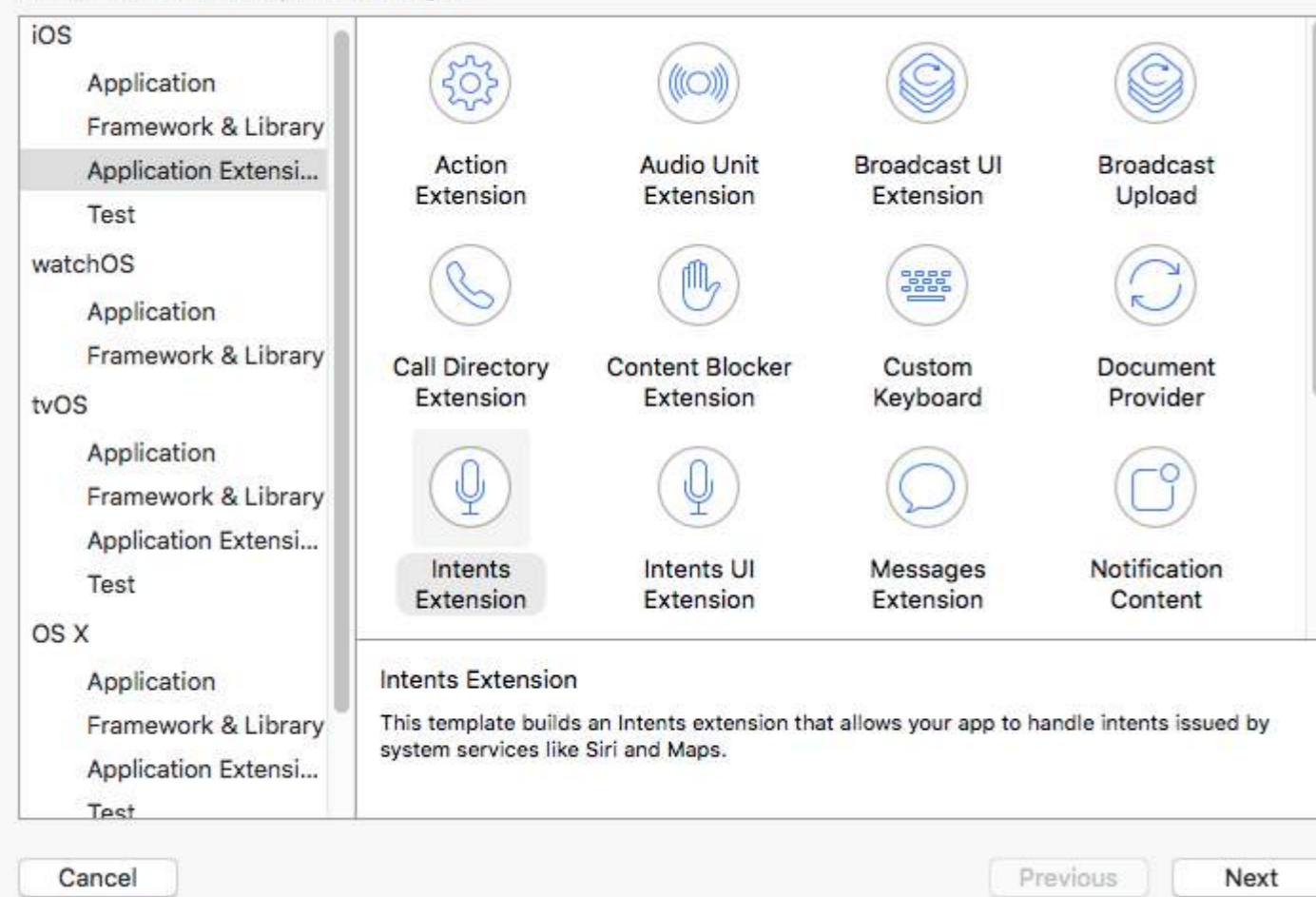
2- 从左侧面板选择 iOS -> 应用扩展

3- 双击右侧的 Intents 扩展

根据苹果公司：

Intents 扩展模板构建了一个 Intents 扩展，允许你的应用处理由系统服务（如 Siri 和地图）发出的意图。

Choose a template for your new target:



Chapter 139: SiriKit

Section 139.1: Adding Siri Extension to App

To integrate Siri capabilities in your app, you should add an extensions as you would do while creating an iOS 10 Widget (old Today View Extension) or a custom keyboard.

Adding capability

1- In the project settings, select your iOS app target and go to Capabilities tab

2- Enable the Siri capability

Adding the extension

1- Go to File -> New -> Target...

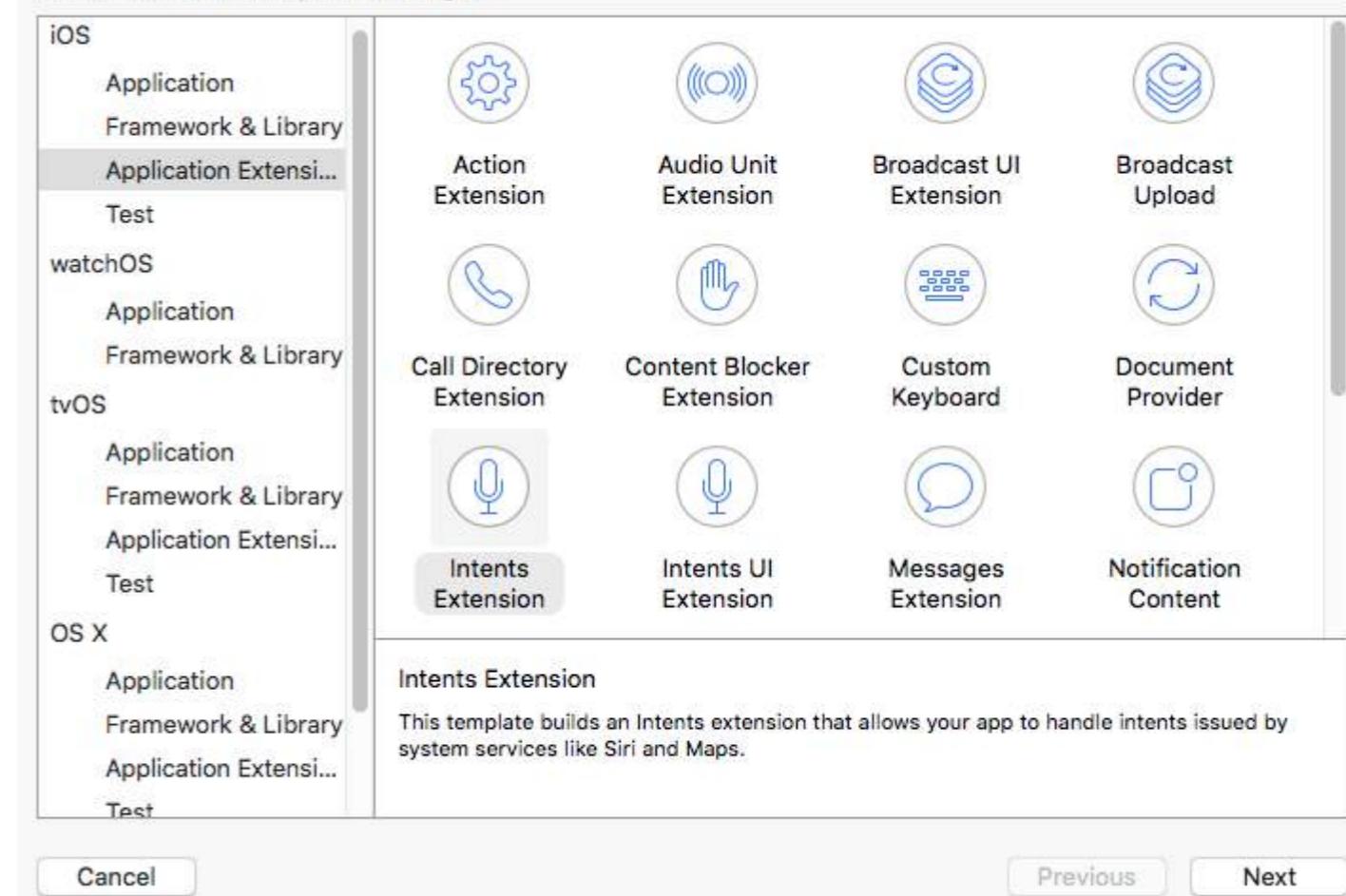
2- Select iOS -> Application Extension from the left pane

3- Double-click Intents Extension from right

According to Apple:

Intents Extension template builds an Intents extension that allows your app to handle intents issued by system services like Siri and Maps.

Choose a template for your new target:



4- 选择一个名称，并确保勾选“包含 UI 扩展”



完成这些步骤后，会创建两个新目标（Intents 扩展和 UI 扩展），默认包含锻炼意图代码。有关不同类型的 Siri 请求，请参见备注。

注意

每当你想调试扩展时，只需从可用方案中选择 Intent 方案。

注意

你无法在模拟器中测试 SiriKit 应用。你需要一台真实设备。

4- Choose a name, and be sure to check "Include UI Extension"



By doing this steps, two new targets (Intents Extension and UI Extension) are created, and by default they contain Workout Intent code. For different types of Siri requests, see Remarks.

Note

Anytime you want to debug your extension, just select the Intent scheme from the available schemes.

Note

You can't test SiriKit apps in the Simulator. Instead, you need a real device.

第140章：联系人框架

第140.1节：添加联系人

Swift

```
import Contacts

// 创建一个可变对象以添加联系人
let contact = CNMutableContact()

contact.imageData = NSData() // 作为NSData对象的头像图片

contact.givenName = "约翰"
contact.familyName = "苹果种子"

let homeEmail = CNLabeledValue(label:CNLabelHome, value:"john@example.com")
let workEmail = CNLabeledValue(label:CNLabelWork, value:"j.appleseed@icloud.com")
contact.emailAddresses = [homeEmail, workEmail]

contact.phoneNumbers = [CNLabeledValue(
    label:CNLabelPhoneNumberiPhone,
    value:CNPhoneNumber(stringValue:"(408) 555-0126"))]

let homeAddress = CNMutablePostalAddress()
homeAddress.street = "无限循环路1号"
homeAddress.city = "库比蒂诺"
homeAddress.state = "CA"
homeAddress.postalCode = "95014"
contact.postalAddresses = [CNLabeledValue(label:CNLabelHome, value:homeAddress)]

let birthday = NSDateComponents()
birthday.day = 1
birthday.month = 4
birthday.year = 1988 // 你可以省略年份值以表示无年份生日
contact.birthday = birthday

// 保存新创建的联系人
let store = CNContactStore()
let saveRequest = CNSaveRequest()
saveRequest.addContact(contact, toContainerWithIdentifier:nil)
try! store.executeSaveRequest(saveRequest)
```

第140.2节：授权联系人访问权限

导入框架

Swift

```
import Contacts
```

Objective-C

```
#import <Contacts/Contacts.h>
```

检查无障碍

Swift

```
switch CNContactStore.authorizationStatusForEntityType(CNEntityType.Contacts){
case .Authorized: //访问联系人
case .Denied, .NotDetermined: //请求权限
default: break
```

Chapter 140: Contacts Framework

Section 140.1: Adding a Contact

Swift

```
import Contacts

// Creating a mutable object to add to the contact
let contact = CNMutableContact()

contact.imageData = NSData() // The profile picture as a NSData object

contact.givenName = "John"
contact.familyName = "Appleseed"

let homeEmail = CNLabeledValue(label:CNLabelHome, value:"john@example.com")
let workEmail = CNLabeledValue(label:CNLabelWork, value:"j.appleseed@icloud.com")
contact.emailAddresses = [homeEmail, workEmail]

contact.phoneNumbers = [CNLabeledValue(
    label:CNLabelPhoneNumberiPhone,
    value:CNPhoneNumber(stringValue:"(408) 555-0126"))]

let homeAddress = CNMutablePostalAddress()
homeAddress.street = "1 Infinite Loop"
homeAddress.city = "Cupertino"
homeAddress.state = "CA"
homeAddress.postalCode = "95014"
contact.postalAddresses = [CNLabeledValue(label:CNLabelHome, value:homeAddress)]

let birthday = NSDateComponents()
birthday.day = 1
birthday.month = 4
birthday.year = 1988 // You can omit the year value for a yearless birthday
contact.birthday = birthday

// Saving the newly created contact
let store = CNContactStore()
let saveRequest = CNSaveRequest()
saveRequest.addContact(contact, toContainerWithIdentifier:nil)
try! store.executeSaveRequest(saveRequest)
```

Section 140.2: Authorizing Contact Access

Importing the framework

Swift

```
import Contacts
```

Objective-C

```
#import <Contacts/Contacts.h>
```

Checking accessibility

Swift

```
switch CNContactStore.authorizationStatusForEntityType(CNEntityType.Contacts){
case .Authorized: //access contacts
case .Denied, .NotDetermined: //request permission
default: break
```

```
}
```

Objective-C

```
switch ([CNContactStore authorizationStatusForEntityType:CNEntityType.Contacts]){
    case CNAuthorizationStatus.Authorized:
        //访问联系人
        break;
    case CNAuthorizationStatus.Denied:
        //请求权限
        break;
    case CNAuthorizationStatus.NotDetermined:
        //请求权限
        break;
}
```

请求权限

Swift

```
var contactStore = CKContactStore()
contactStore.requestAccessForEntityType(CKEntityType.Contacts, completionHandler: { (ok, _) -> Void
in
    if access{
        //访问联系人
    }
}
```

第140.3节：访问联系人

应用过滤器

要访问联系人，我们应该对之前在授权联系人访问示例中定义的contactStore变量应用类型为NSPredicate的过滤器。例如，这里我们想筛选出姓名与我们自己匹配的联系人：

Swift

```
let predicate = CNContact.predicateForContactsMatchingName("Some Name")
```

Objective-C

```
NSPredicate *predicate = [CNContact predicateForContactsMatchingName:@"Some Name"];
```

指定要获取的键

这里，我们想获取联系人的名字、姓氏和头像：

Swift

```
let keys = [CNContactGivenNameKey, CNContactFamilyNameKey, CNContactImageDataKey]
```

获取联系人

Swift

```
do {
    let contacts = try contactStore.unifiedContactsMatchingPredicate(predicate, keysToFetch: keys)
} catch let error as NSError {
    //...
}
```

访问联系人详情

Swift

```
print(contacts[0].givenName)
```

```
}
```

Objective-C

```
switch ([CNContactStore authorizationStatusForEntityType:CNEntityType.Contacts]){
    case CNAuthorizationStatus.Authorized:
        //access contacts
        break;
    case CNAuthorizationStatus.Denied:
        //request permission
        break;
    case CNAuthorizationStatus.NotDetermined:
        //request permission
        break;
}
```

Requesting Permission

Swift

```
var contactStore = CKContactStore()
contactStore.requestAccessForEntityType(CKEntityType.Contacts, completionHandler: { (ok, _) -> Void
in
    if access{
        //access contacts
    }
}
```

Section 140.3: Accessing Contacts

Applying a filter

To access contacts, we should apply a filter of type `NSPredicate` to our `contactStore` variable which we defined it in Authorizing Contact Access example. For example, here we want to sort out contacts with name matching with our own:

Swift

```
let predicate = CNContact.predicateForContactsMatchingName("Some Name")
```

Objective-C

```
NSPredicate *predicate = [CNContact predicateForContactsMatchingName:@"Some Name"];
```

Specifying keys to fetch

Here, we want to fetch the contact's first name, last name and profile image:

Swift

```
let keys = [CNContactGivenNameKey, CNContactFamilyNameKey, CNContactImageDataKey]
```

Fetching contacts

Swift

```
do {
    let contacts = try contactStore.unifiedContactsMatchingPredicate(predicate, keysToFetch: keys)
} catch let error as NSError {
    //...
}
```

Accessing contact details

Swift

```
print(contacts[0].givenName)
```

```
print(contacts[1].familyName)
let image = contacts[2].imageData
```

```
print(contacts[1].familyName)
let image = contacts[2].imageData
```

第141章：iOS 10语音识别API

第141.1节：语音转文本：识别捆绑包中包含的音频录音的语音

```
//import Speech
//import AVFoundation

// 创建一个文本框以显示语音输出
@IBOutlet weak var transcriptionTextField: UITextView!
// 我们需要这个音频播放器来播放音频
var audioPlayer: AVAudioPlayer!

override func viewDidLoad()
{
    super.viewDidLoad()
}

// 该函数用于在音频播放完成时停止音频，否则会重复播放同一音频
func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool)
{
    player.stop()
}

// 该函数用于获取语音识别器，之后向语音识别器发起请求
func requestSpeechAuth()
{
    SFSpeechRecognizer.requestAuthorization { authStatus in
        if authStatus == SFSpeechRecognizerAuthorizationStatus.authorized {
            if let path = Bundle.main.url(forResource: "mpthree", withExtension: "m4a") {
                do {
                    let sound = try AVAudioPlayer(contentsOf: path)
                    self.audioPlayer = sound
                    self.audioPlayer.delegate = self
                    sound.play()
                } catch {
                    print("error")
                }
            }
        }
    }
}

let recognizer = SFSpeechRecognizer()
let request = SFSpeechURLRecognitionRequest(url:path)
recognizer?.recognitionTask(with: request) { (result, error) in
    if let error = error {
        print("there is a error\((error))")
    } else {
        // here you are printing out the audio output basically showing it on uitext field
        self.transcriptionTextField.text =
result?.bestTranscription.formattedString
    }
}

// 这里你在 UIButton 按下时调用 requestSpeechAuth 函数
@IBAction func playButtonPress(_ sender: AnyObject)
{
```

Chapter 141: iOS 10 Speech Recognition API

Section 141.1: Speech to text: Recognize speech from a bundle contained audio recording

```
//import Speech
//import AVFoundation

// create a text field to show speech output
@IBOutlet weak var transcriptionTextField: UITextView!
// we need this audio player to play audio
var audioPlayer: AVAudioPlayer!

override func viewDidLoad()
{
    super.viewDidLoad()
}

// this function is required to stop audio on audio completion otherwise it will play same audio
again and again
func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool)
{
    player.stop()
}

// this function is required to get a speech recognizer and after that make and request to speech
recognizer
func requestSpeechAuth()
{
    SFSpeechRecognizer.requestAuthorization { authStatus in
        if authStatus == SFSpeechRecognizerAuthorizationStatus.authorized {
            if let path = Bundle.main.url(forResource: "mpthree", withExtension: "m4a") {
                do {
                    let sound = try AVAudioPlayer(contentsOf: path)
                    self.audioPlayer = sound
                    self.audioPlayer.delegate = self
                    sound.play()
                } catch {
                    print("error")
                }
            }
        }
    }
}

let recognizer = SFSpeechRecognizer()
let request = SFSpeechURLRecognitionRequest(url:path)
recognizer?.recognitionTask(with: request) { (result, error) in
    if let error = error {
        print("there is a error\((error))")
    } else {
        // here you are printing out the audio output basically showing it on uitext field
        self.transcriptionTextField.text =
result?.bestTranscription.formattedString
    }
}

// here you are calling requestSpeechAuth function on UIButton press
@IBAction func playButtonPress(_ sender: AnyObject)
{
```

```
requestSpeechAuth()  
}
```

```
requestSpeechAuth()  
}
```

第142章：StoreKit

第142.1节：从App

StoreKit
获取本地化的产品信息

使用SKProductsRequest从一组产品标识符字符串中获取本地化的产品信息：

```
import StoreKit

let productIdentifierSet = Set(["yellowSubmarine", "pennyLane"])
let productsRequest = SKProductsRequest(productIdentifiers: productIdentifierSet)
```

为了处理来自productsRequest的产品，我们需要为请求分配一个代理来处理响应。该代理需要遵循SKProductsRequestDelegate协议，这意味着它必须继承自NSObject（即任何Foundation对象）并实现productsRequest方法：

```
class PaymentManager: NSObject, SKProductsRequestDelegate {

    var products: [SKProduct] = []

    func productsRequest(request: SKProductsRequest,
                         didReceiveResponse response: SKProductsResponse) {
        products = response.products
    }
}
```

为了启动productsRequest，我们将PaymentManager设置为产品请求的代理，并调用请求的start()方法：

```
let paymentManager = PaymentManager()
productsRequest.delegate = paymentManager
productsRequest.start()
```

如果请求成功，产品将会在paymentManager.products中。

Chapter 142: StoreKit

Section 142.1: Get localized product information from the App Store

Get localized product information from a set of product identifier strings using SKProductsRequest:

```
import StoreKit

let productIdentifierSet = Set(["yellowSubmarine", "pennyLane"])
let productsRequest = SKProductsRequest(productIdentifiers: productIdentifierSet)
```

In order to process the products from the productsRequest, we need to assign a delegate to the request that handles the response. The delegate needs to conform to the SKProductsRequestDelegate protocol, which means that it must inherit from `NSObject` (i.e. any Foundation object) and implement the `productsRequest` method:

```
class PaymentManager: NSObject, SKProductsRequestDelegate {

    var products: [SKProduct] = []

    func productsRequest(request: SKProductsRequest,
                         didReceiveResponse response: SKProductsResponse) {
        products = response.products
    }
}
```

To initiate the productsRequest we assign PaymentManager as the products-request's delegate, and call the start() method on the request:

```
let paymentManager = PaymentManager()
productsRequest.delegate = paymentManager
productsRequest.start()
```

If the requests succeeds the products will be in paymentManager.products.

第143章：代码签名

第143.1节：配置描述文件

为了在XCode中构建IPA文件，您需要使用证书和配置描述文件对应用进行签名。

这些可以在 <https://developer.apple.com/account/ios/profile/create> 创建

配置描述文件类型

配置描述文件分为两种类型，开发和发布：

开发

- iOS 应用开发 / tvOS 应用开发 - 用于开发过程中将您的应用安装到测试设备上。

分发

- App Store / tvOS App Store - 用于为您的应用签名以便上传到应用商店。
- 企业内部分发 - 用于企业内部将您的应用分发到公司内的设备。
- Ad Hoc / tvOS Ad Hoc - 用于将您的应用分发到有限数量的特定设备（例如，您需要知道想要安装应用的设备的 UDID）。

Chapter 143: Code signing

Section 143.1: Provisioning Profiles

In order to build an IPA file in XCode, you require to sign your application with a certificate and provisioning profile. These can be created at <https://developer.apple.com/account/ios/profile/create>

Provisioning Profile Types

Provisioning Profiles are split into two types, Development, and Distribution:

Development

- iOS App Development / tvOS App Development - used in development in order to install your app onto a test device.

Distribution

- App Store / tvOS App Store - used to sign your application for app store upload.
- In House - Used for Enterprise distribution of your app, to devices within your business.
- Ad Hoc / tvOS Ad Hoc - Used to distribute your app to a limited number of specific devices (e.g. you will need to know the UDIDs of the devices you want to install your app to).

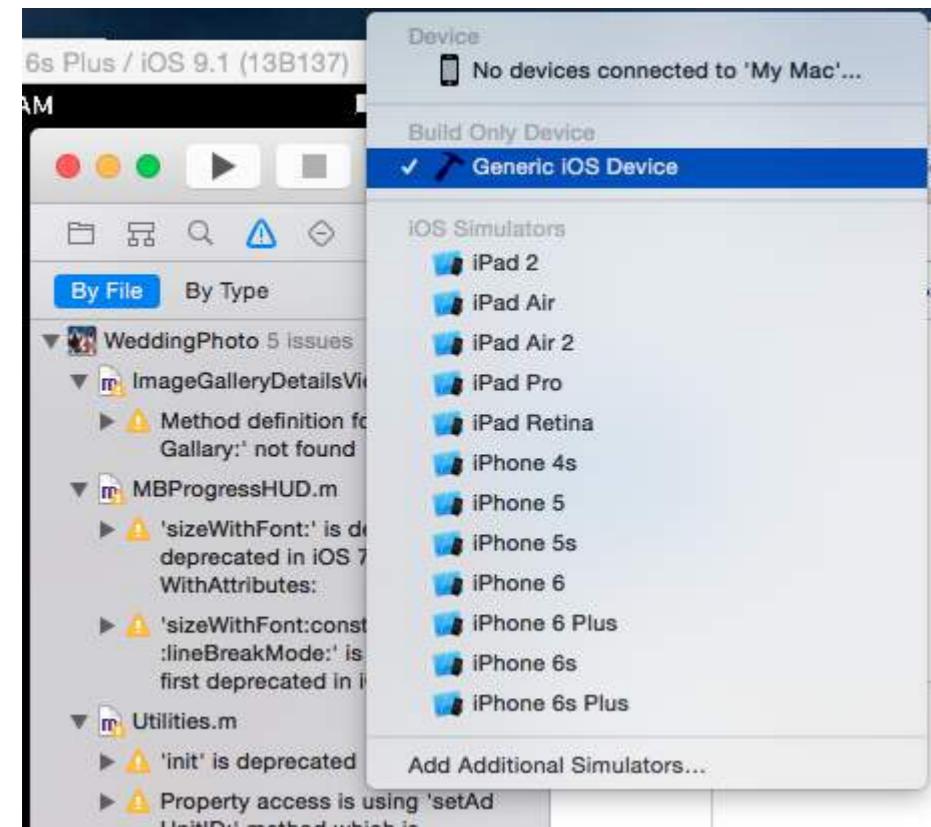
第144章：使用 Applicationloader 创建上传到 App Store 的.ipa 文件

第144.1节：使用

Application Loader 创建上传到 App Store 的.ipa 文件

如果您想上传 .ipa 文件到 iTunes Connect 而不想在 Xcode 中集成开发者账号，并且想使用 Application Loader，那么您可以使用 iTunes 生成 .ipa 文件。

步骤 1：选择设备而非模拟器。



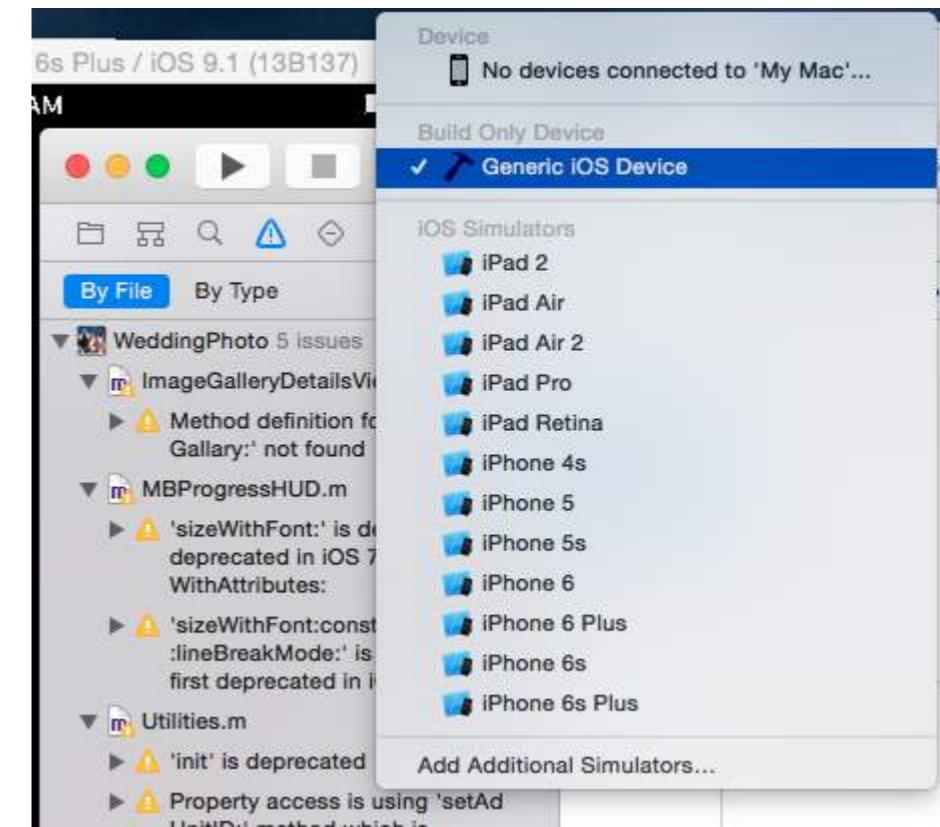
步骤 2：进入产品 -> 选择归档

Chapter 144: Create .ipa File to upload on appstore with Applicationloader

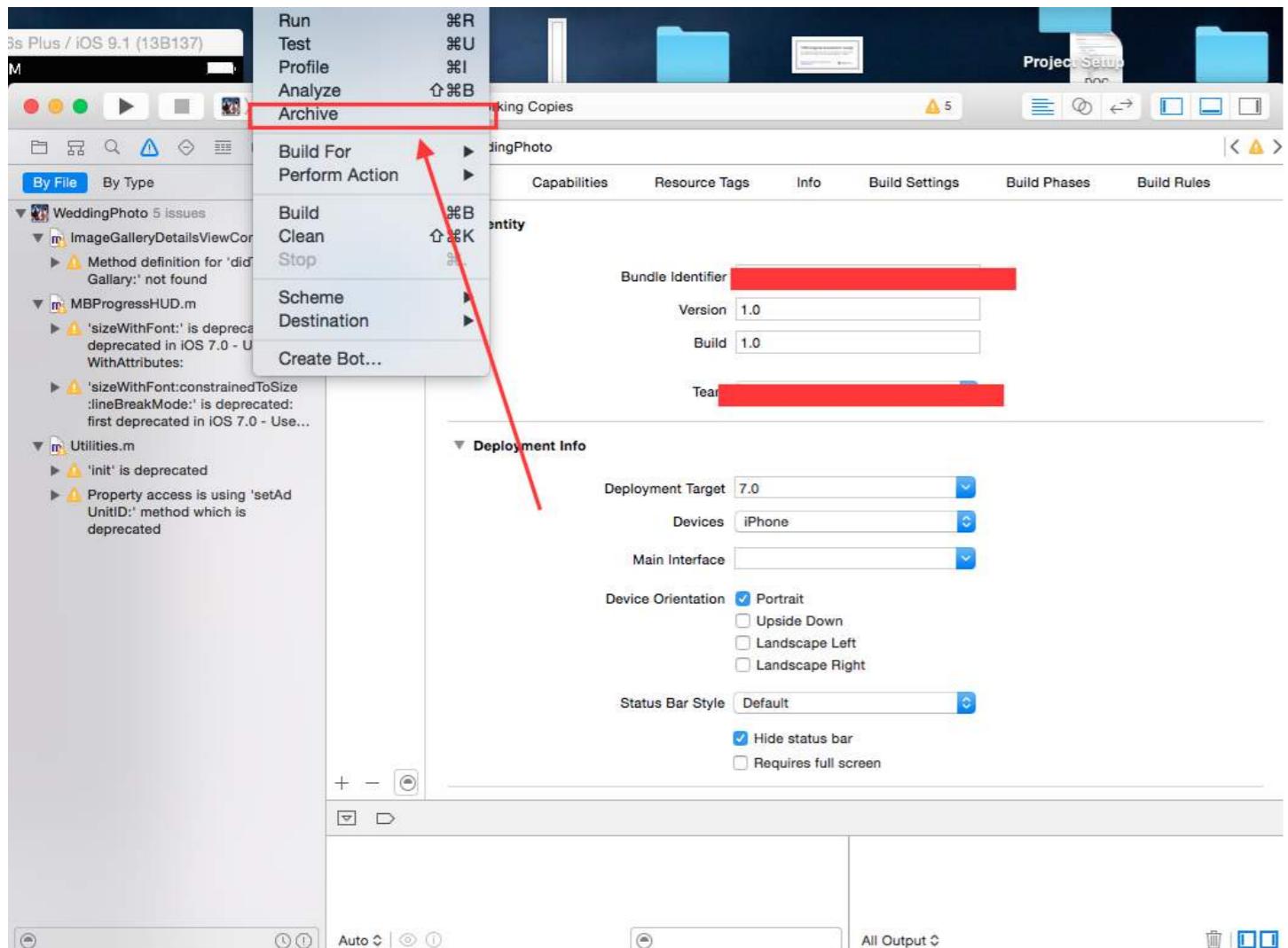
Section 144.1: create .ipa file to upload app to appstore with Application Loader

If you want to upload .ipa file to itunesconnect **without integrating developer account in Xcode** and you want to use **application loader**. then you can **generate .ipa with iTunes** .

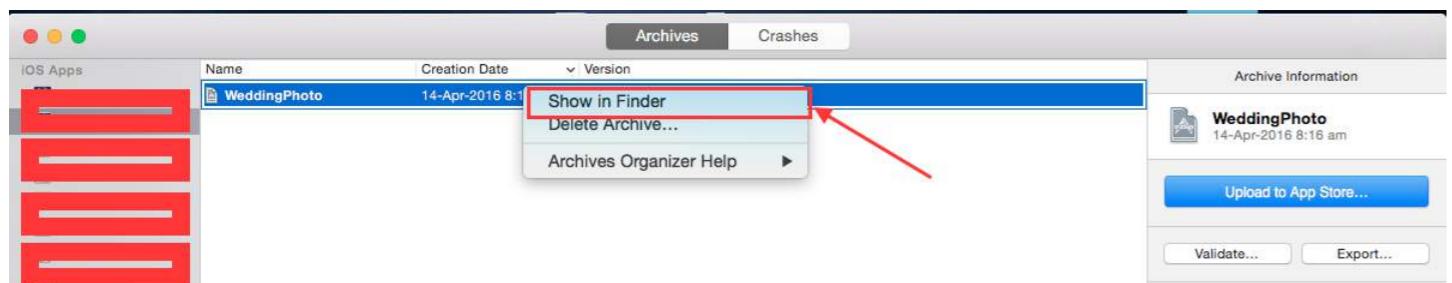
Step 1: Select device inplace of simulator.



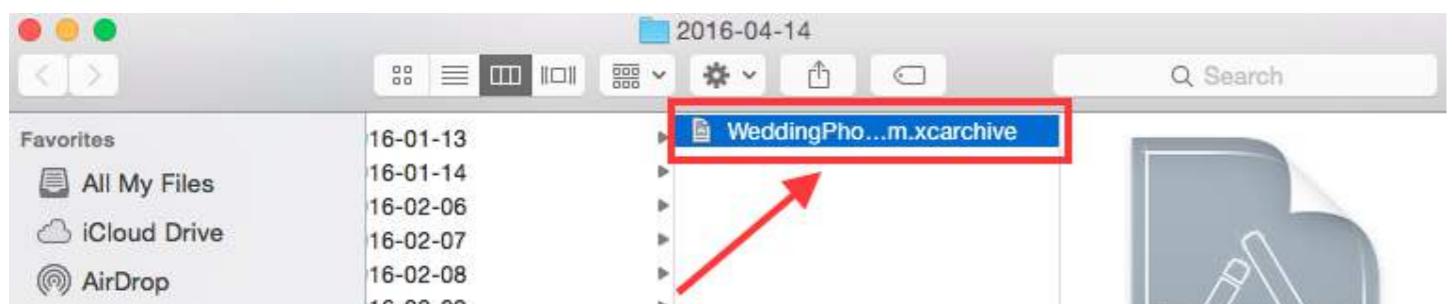
Step 2: Go to Product -> select Archive



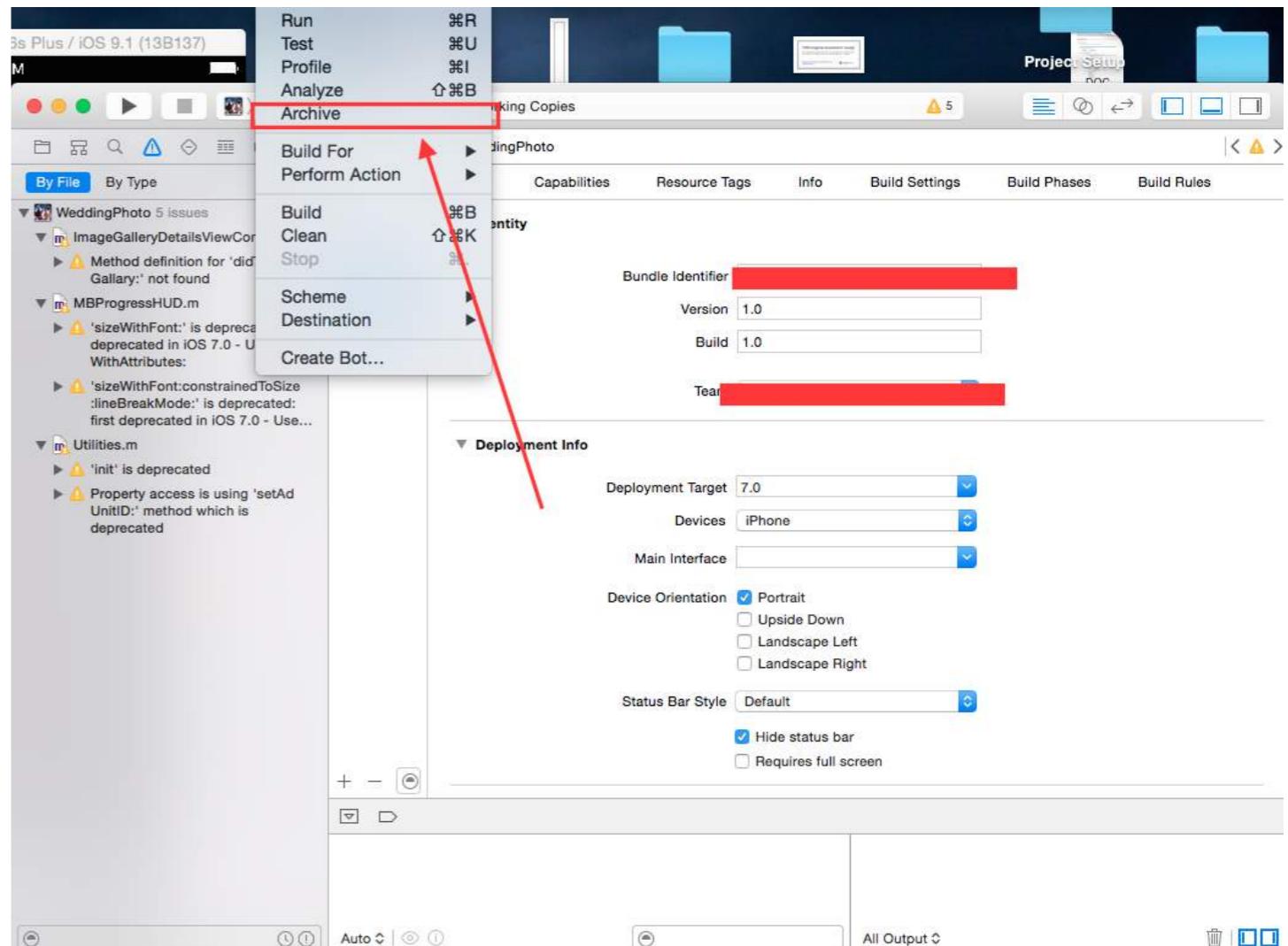
步骤 3：过程完成后，右键点击你的归档 -> 选择在 Finder 中显示



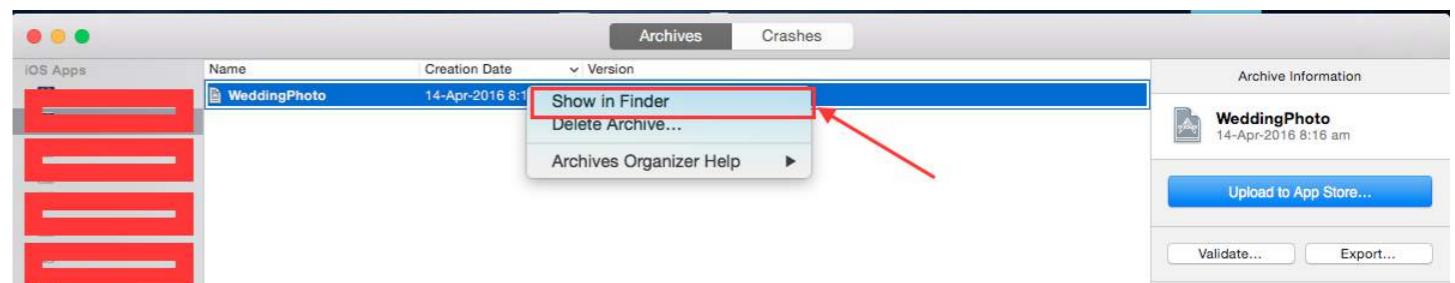
步骤 4：点击“在 Finder 中显示”后，你将被重定向到归档文件夹，界面如下所示



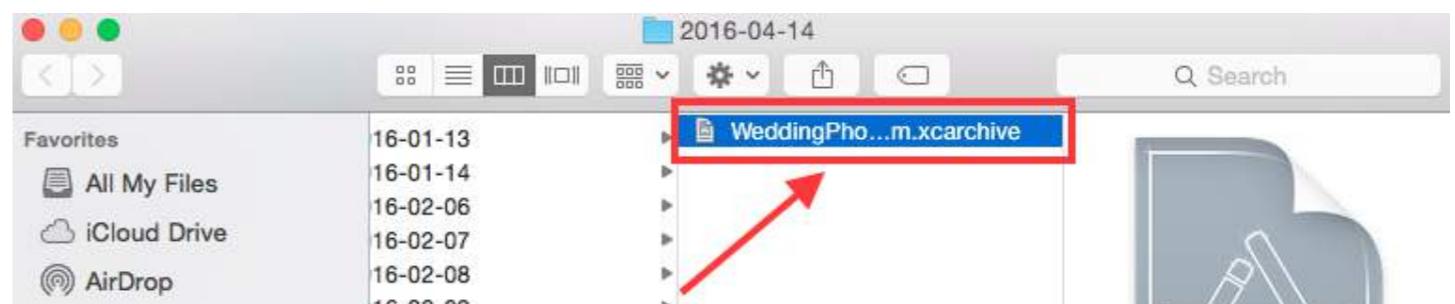
步骤 5：右键点击 .xarchive 文件 -> 选择“在 Finder 中显示”选项。



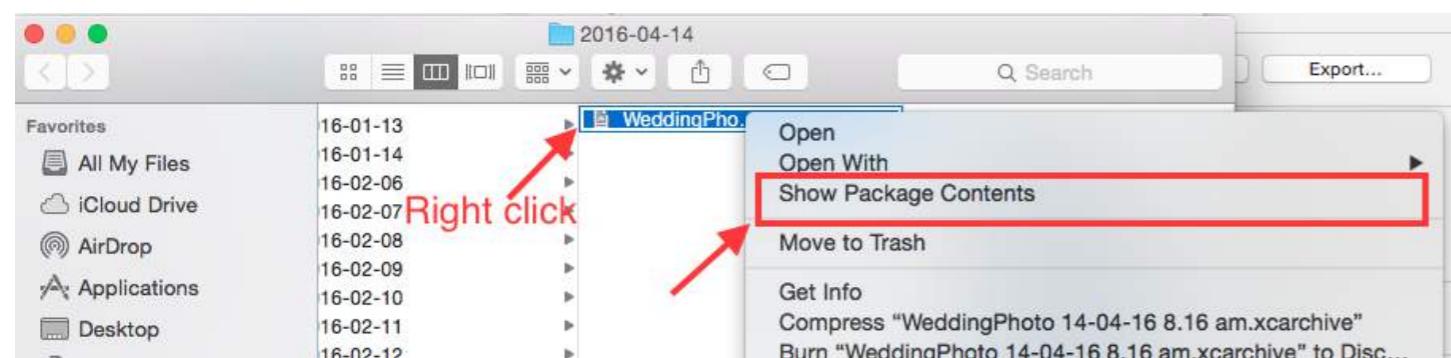
Step 3: After completed process right click to your Archive -> and select show in Finder



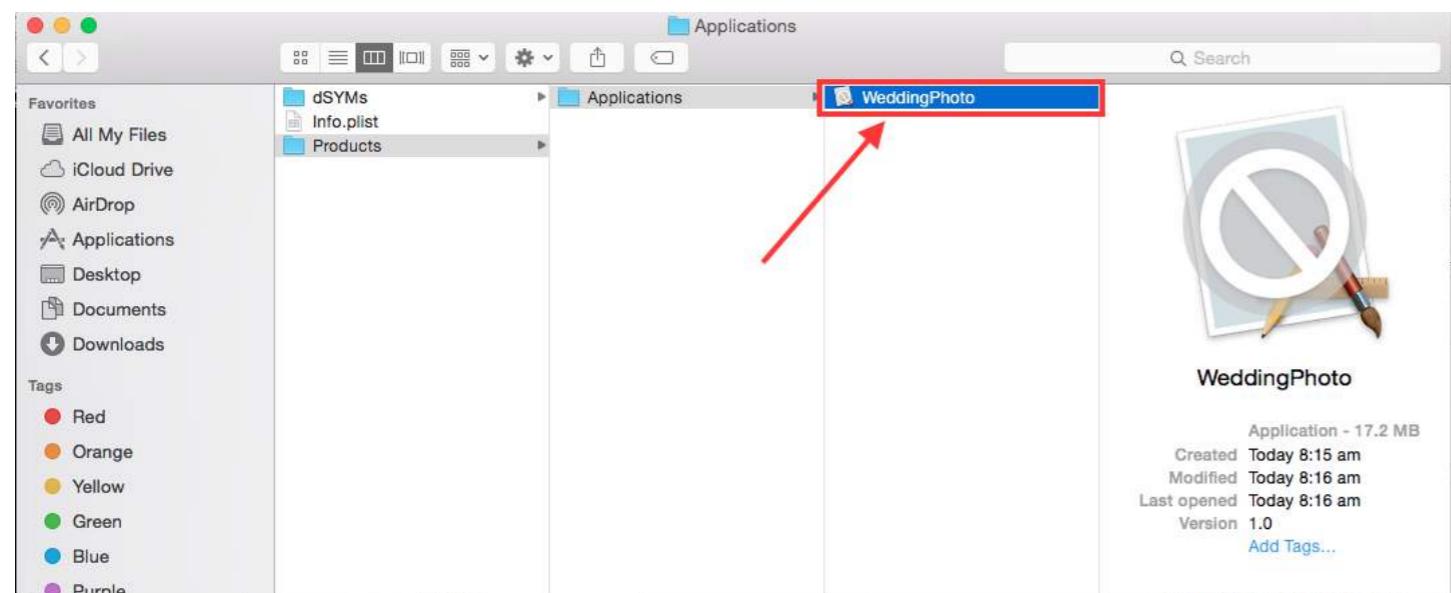
Step 4: when you click on show in finder you will redirect to Archive folder, looks like this



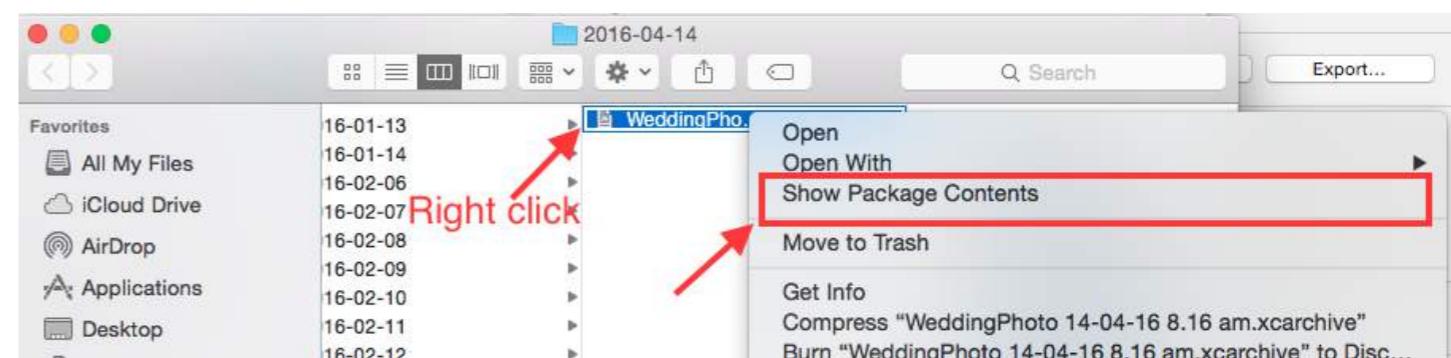
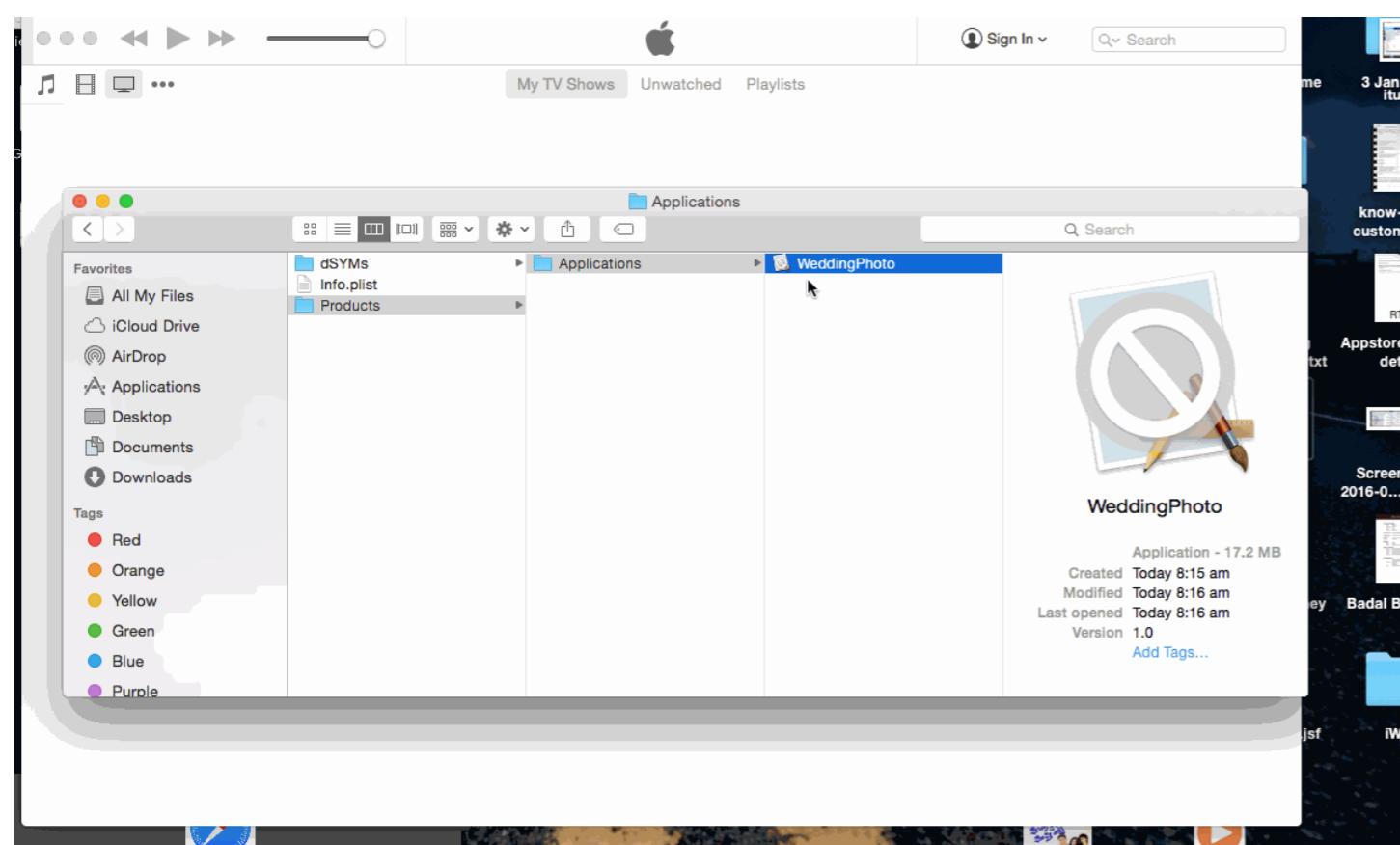
Step 5: Right click on .xarchive file -> select Show in finder option.



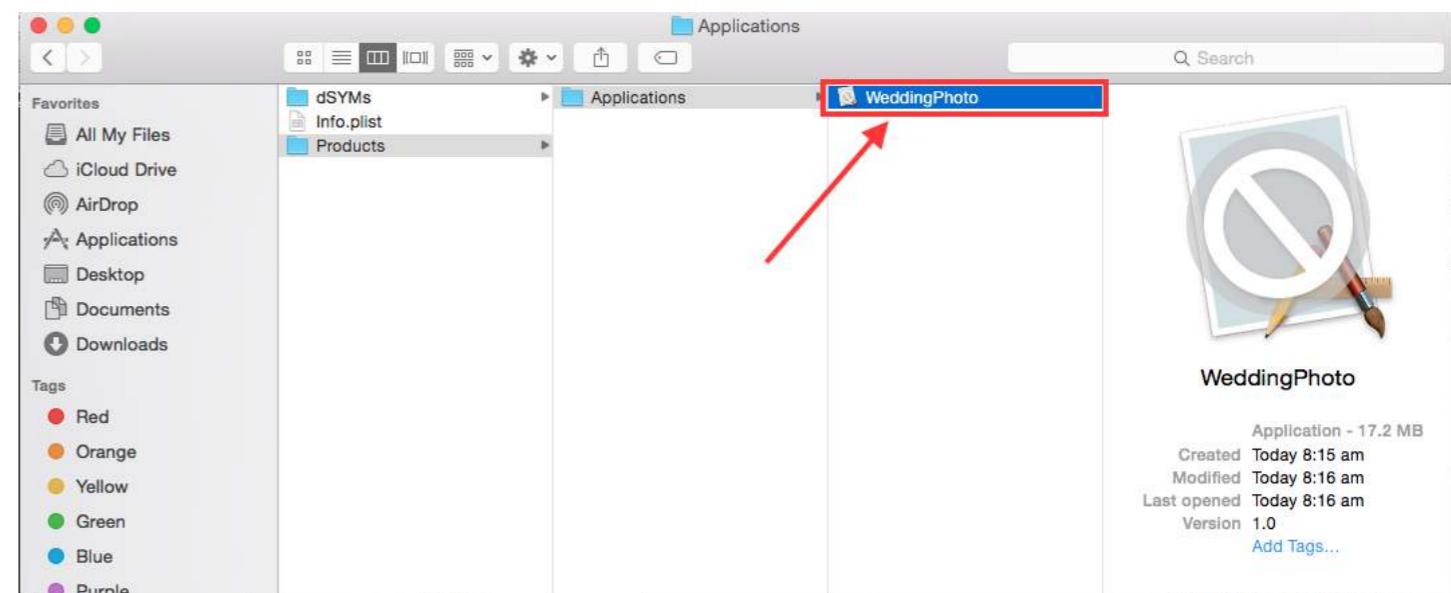
步骤 6：进入产品文件夹 -> 应用程序文件夹 -> 你会找到 yourprojectname.app



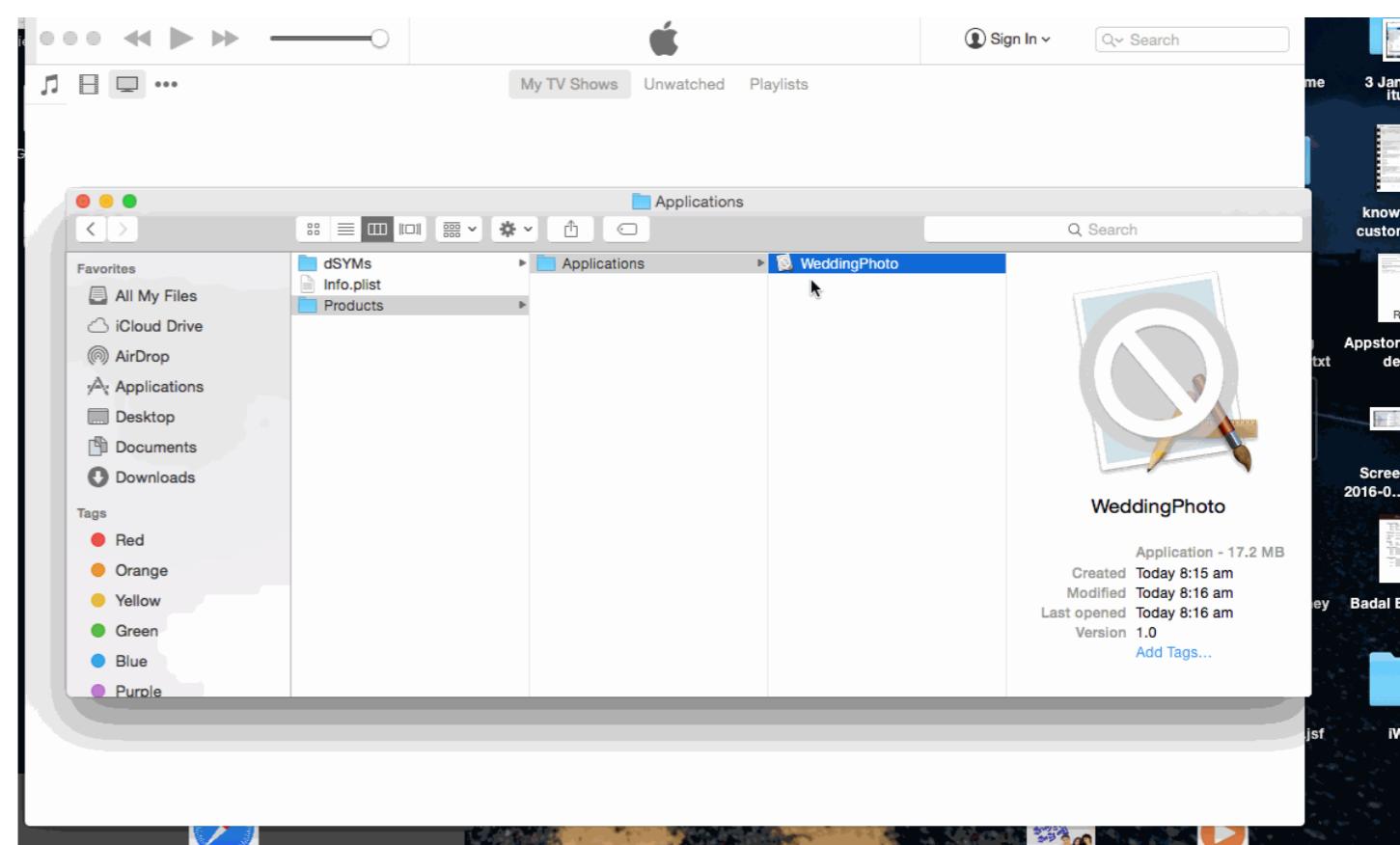
步骤 7：现在将 .app 转换为 .ipa，只需将其拖放到 iTunes 中。请查看下图，



Step 6: Go to Product Folder -> Application Folder -> You will find yourprojectname.app



Step 7: Now to convert .app to .ipa just drag and drop into itunes . check below image ,



步骤8：现在将此 .ipa 文件放在安全的地方，上传时使用应用程序加载器。

注意：如果你想了解如何使用应用加载器上传应用，请查看此内容，

[使用应用加载器上传应用](#)

编辑：

警告：不要通过将扩展名从 .aap 改为 .zip 再从 .zip 改为 .ipa 来制作 .ipa 文件。

我在许多回答中看到，有人建议先压缩 .app 文件，然后将扩展名从 .zip 改为 .ipa 。这种方法现在已经不行了。使用这种方法你会遇到如下错误，

IPA 无效，未包含 payload 目录。

Step 8: Now put this .ipa file in safe place and use when upload with application loader .

Note: if you want to know how to upload app with application loader then check this ,

[Upload app with application Loader](#)

EDIT:

WARNING: Don't make .ipa with changing extension from .aap to .zip and .zip to .ipa.

I have seen in many answer that , they have suggest compress .app file and then change the extension from .zip to .ipa . It is not working now . By this method you will get Error like ,

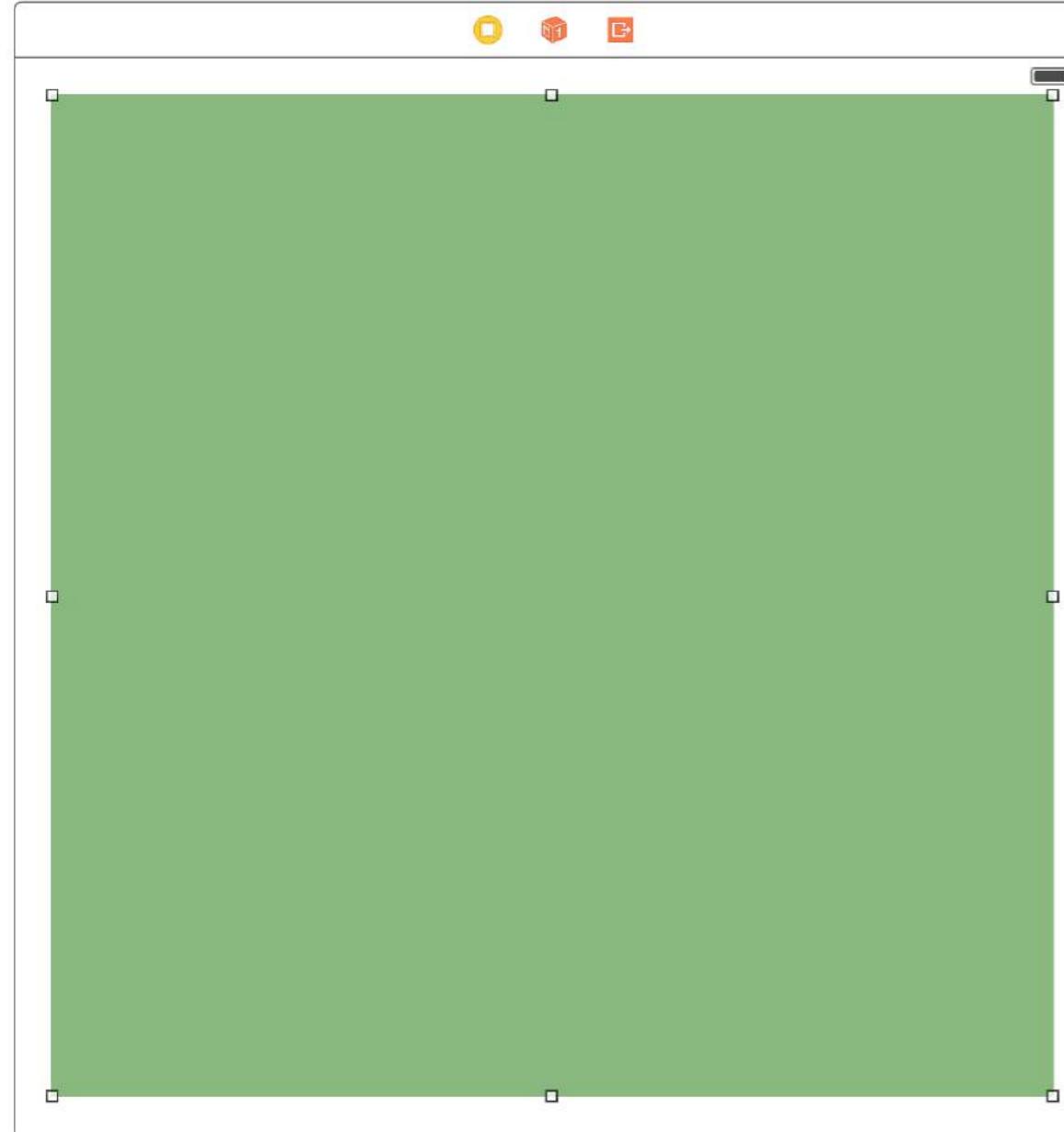
IPA is invalid, it does not include a payload directory.

第145章：尺寸类别和自适应

第145.1节：通过Storyboard实现尺寸类别和自适应

我们可以为任何 `UIView` 的子类添加自适应，该子类是在 `nib` 文件中添加到视图控制器上的。
让我们以使用尺寸类别为视图添加自适应为例。

1. 在视图控制器中添加一个视图，如下所示：



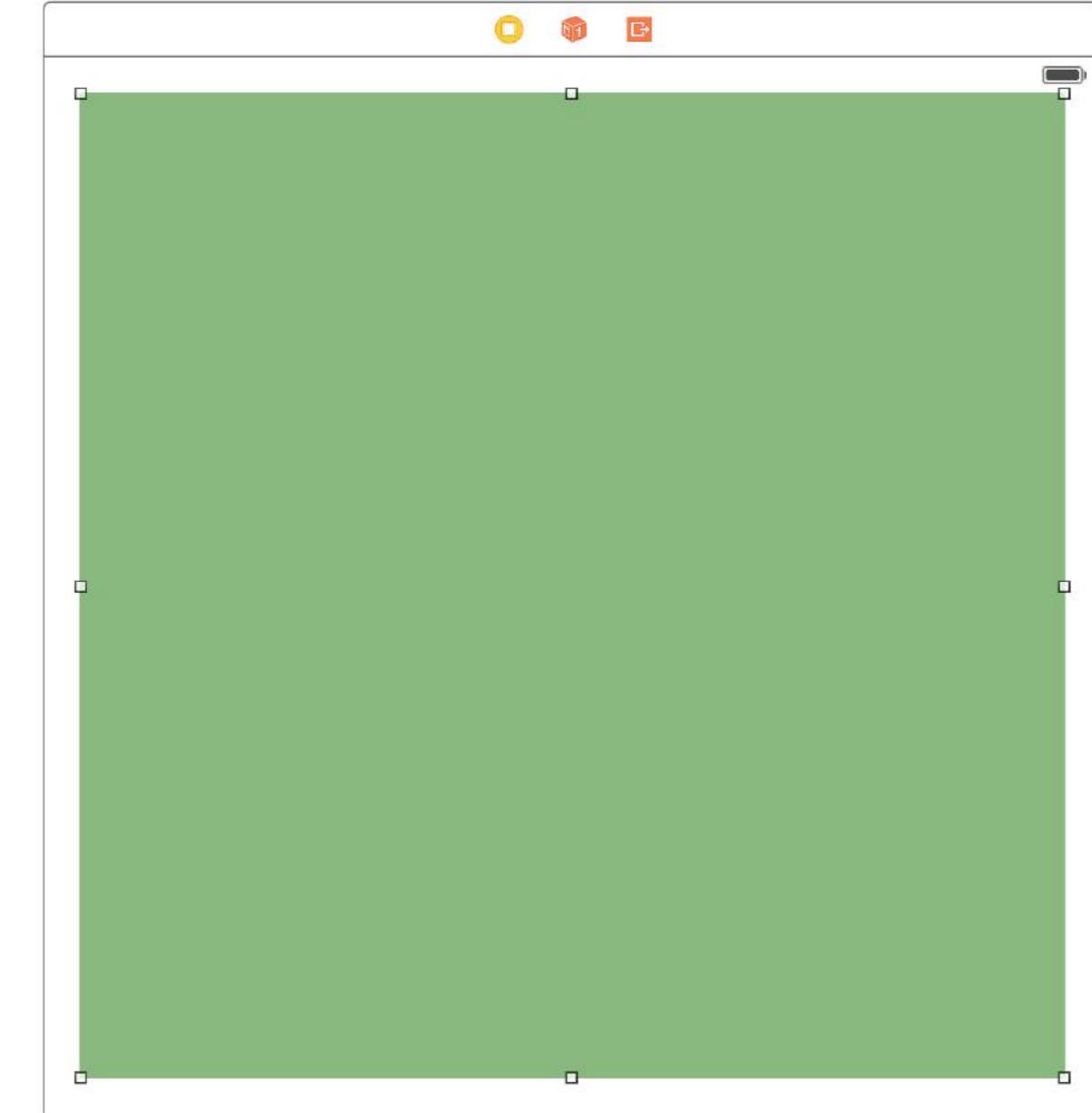
2. 现在我们需要将该视图固定到其父视图，以使用约束来确定其大小和位置，如下所示：

Chapter 145: Size Classes and Adaptivity

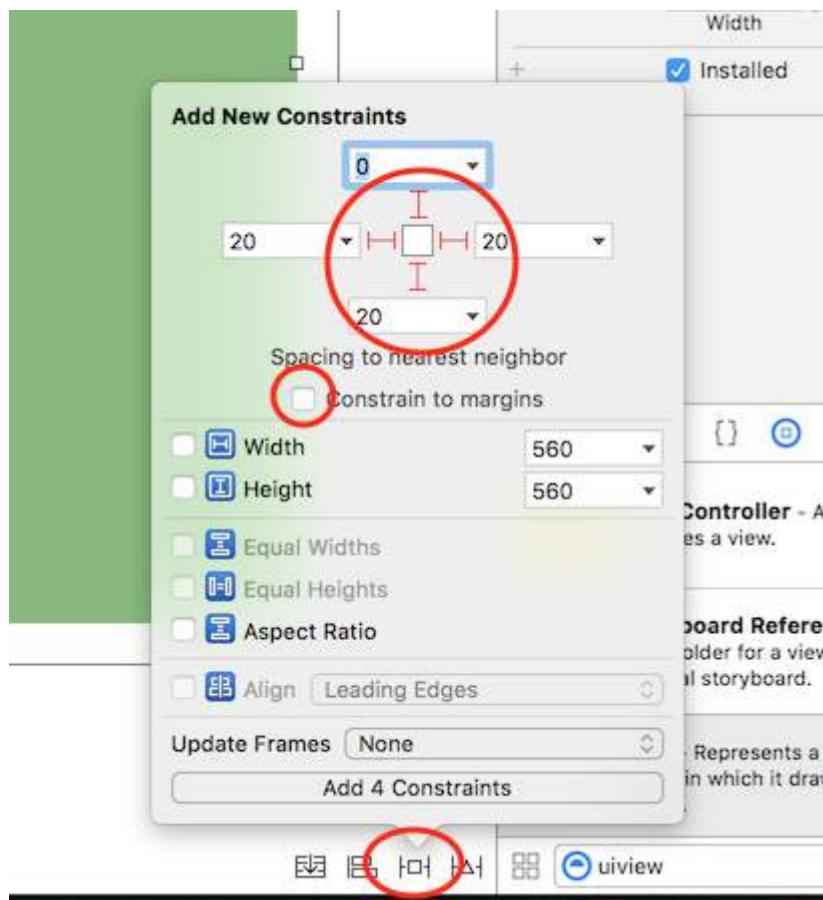
Section 145.1: Size Classes and Adaptivity through Storyboard

We can add adaptivity to any subclass of `UIView` which we add on view controller in nib file.
Lets take an example of adding adaptivity using size classes to a view.

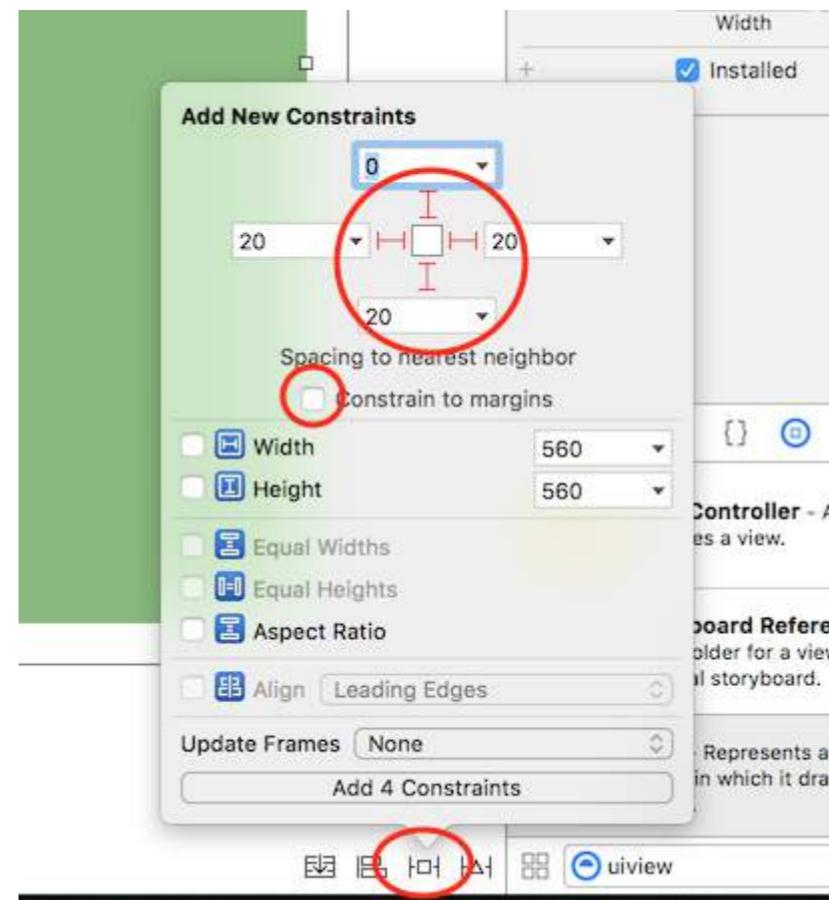
1. Add a view on view controller as:



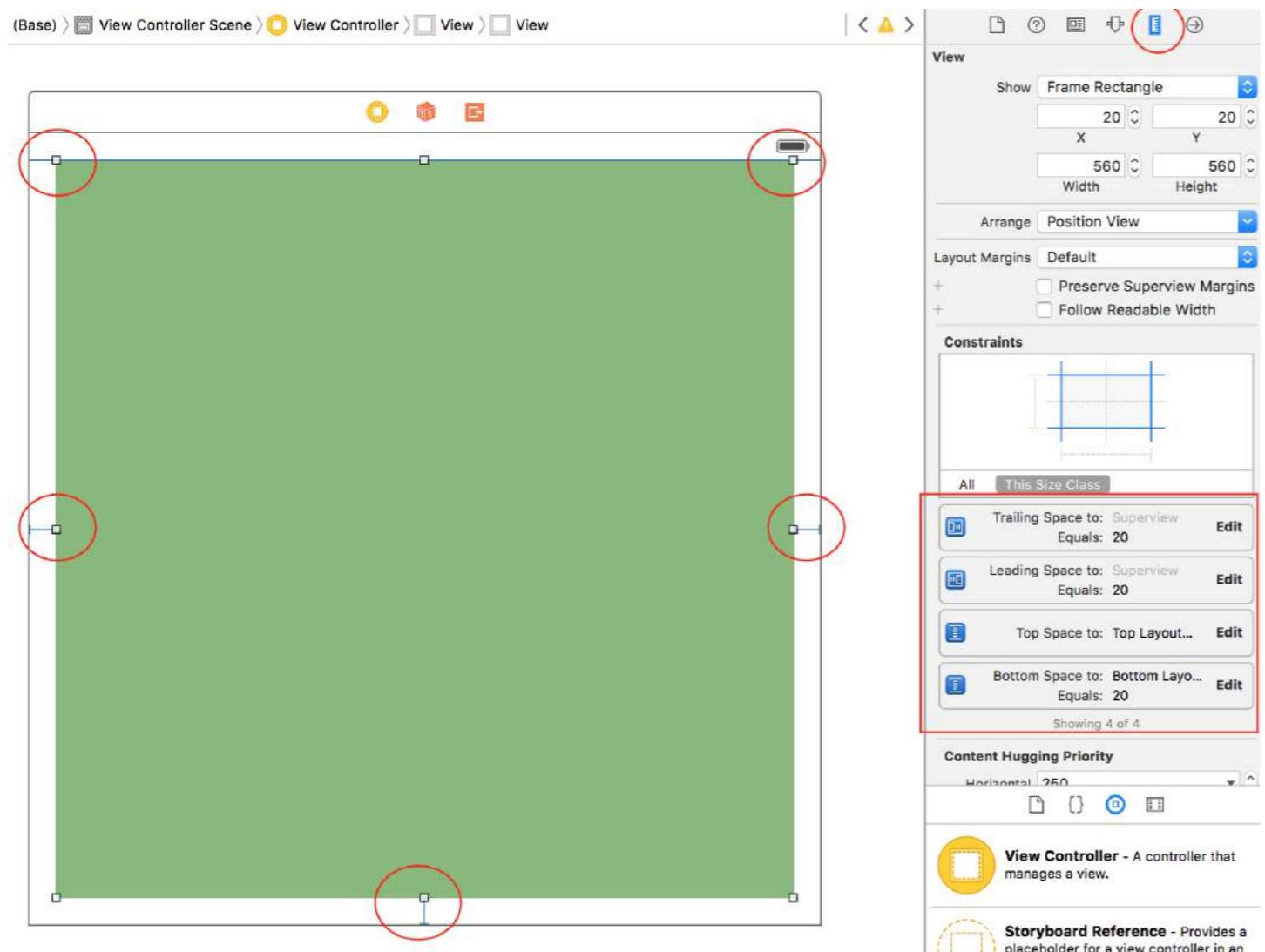
2. Now we need to pin this view to its superview for fixing its size and position using **constraints** as:



3. 我们可以看到添加的约束如下：



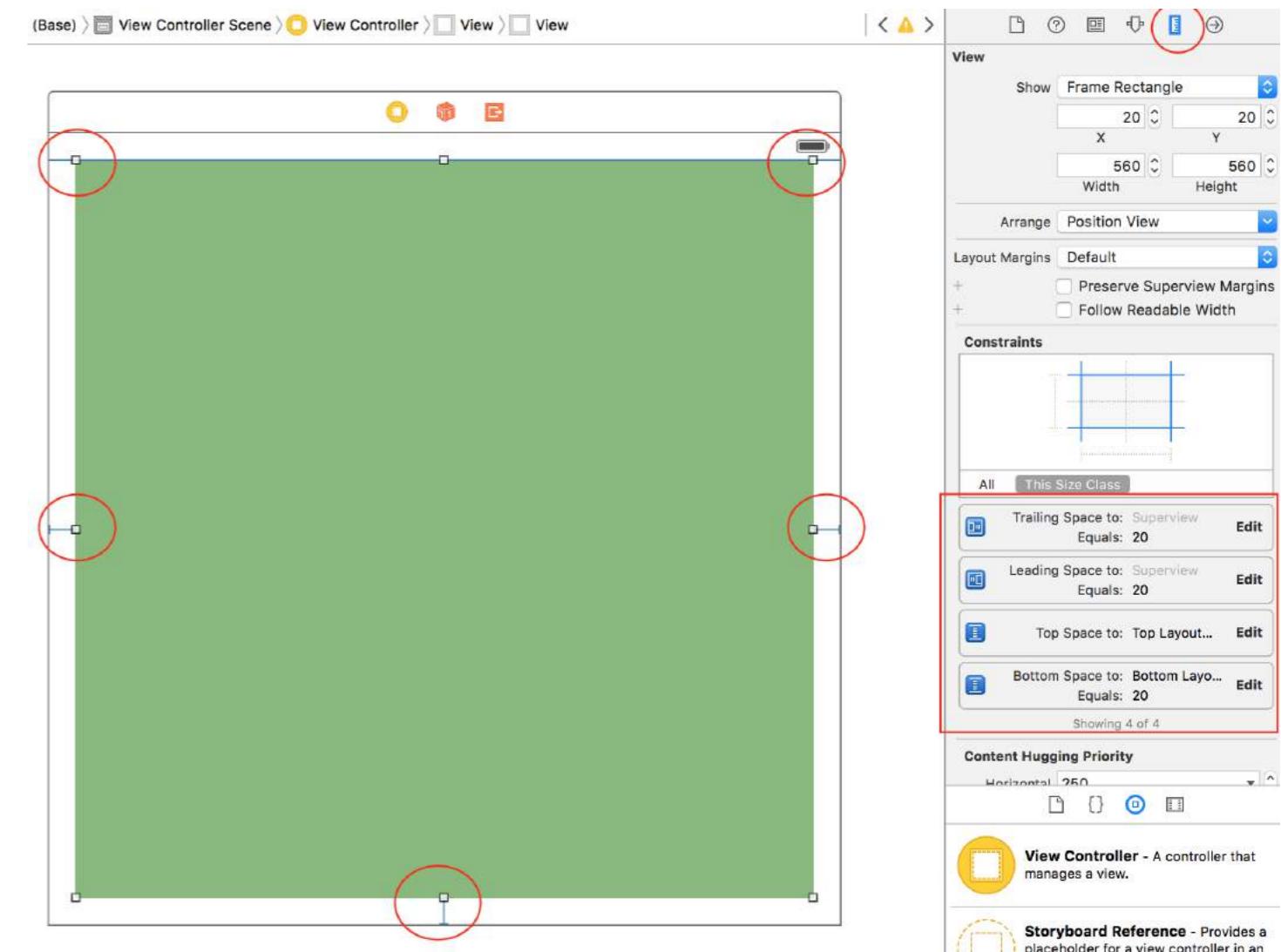
3. We can see the added constraints as:



这些约束定义了添加的视图将在其父视图中放置为

```
CGRect(20, 0, superview.width - 20, superview.height - 20)
```

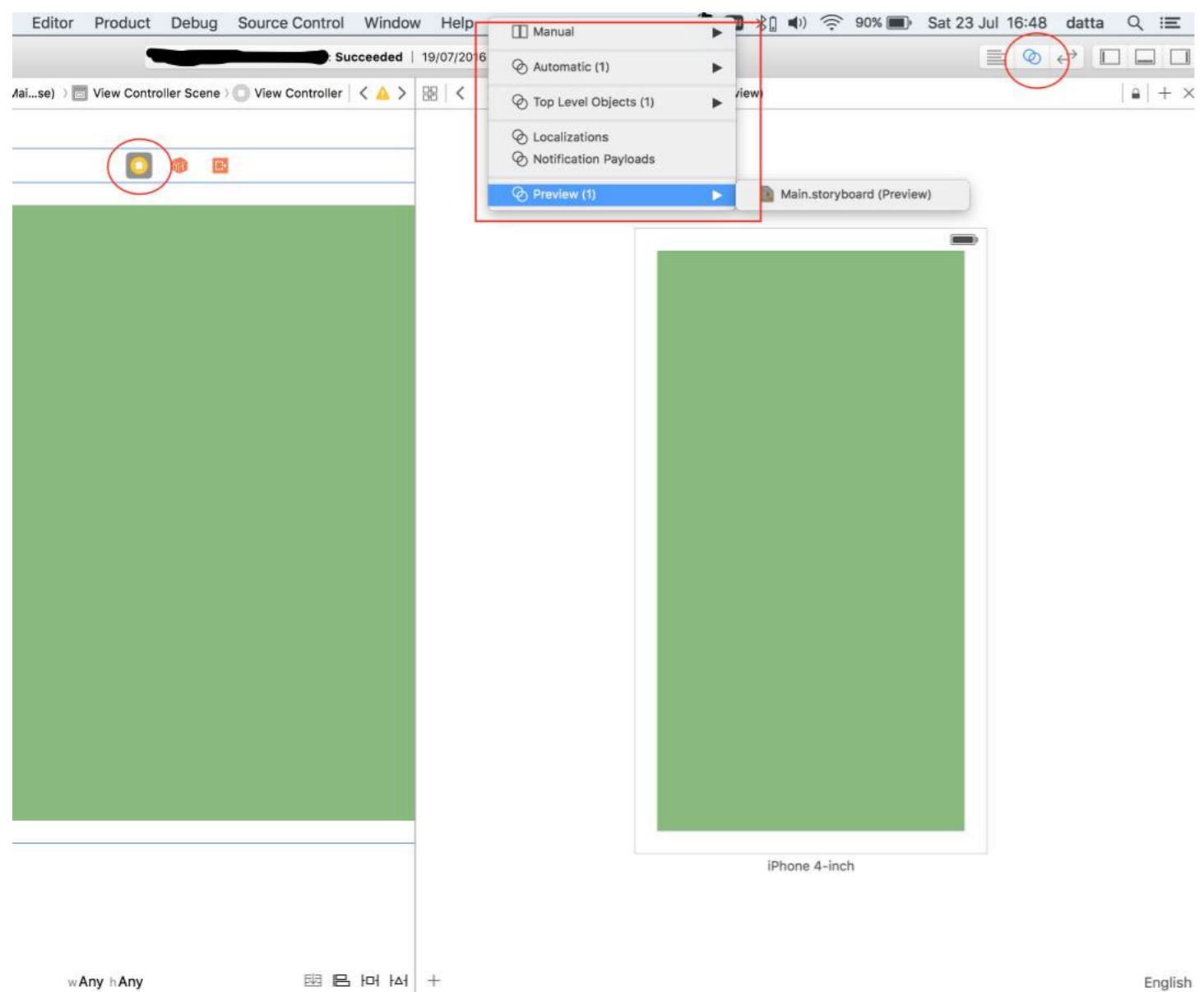
4. 要在屏幕上查看这些添加的约束的预览，我们可以使用辅助编辑器，如下所示；



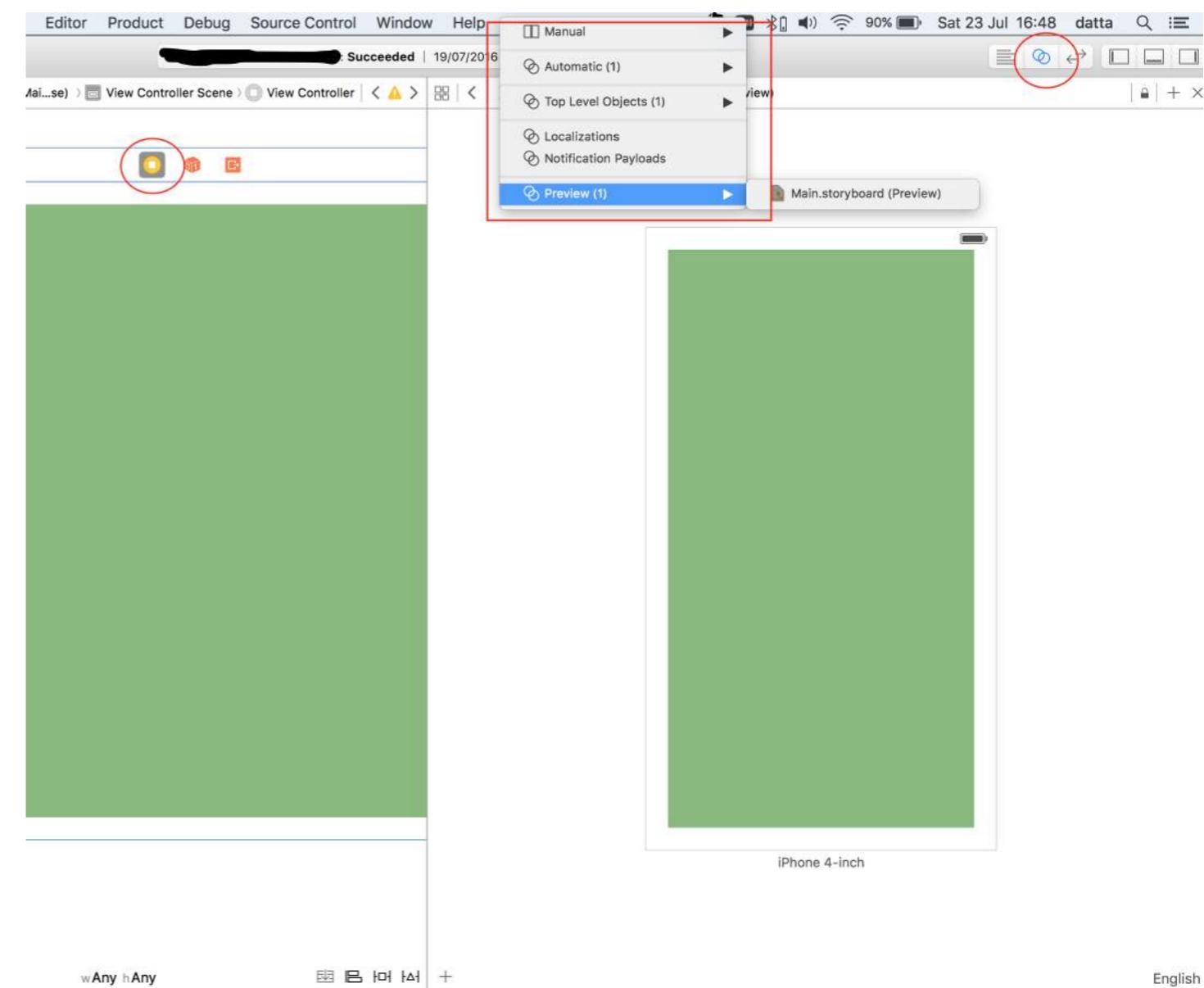
These constraints defines that the added view will be placed in it's superview as

```
CGRect(20, 0, superview.width - 20, superview.height - 20)
```

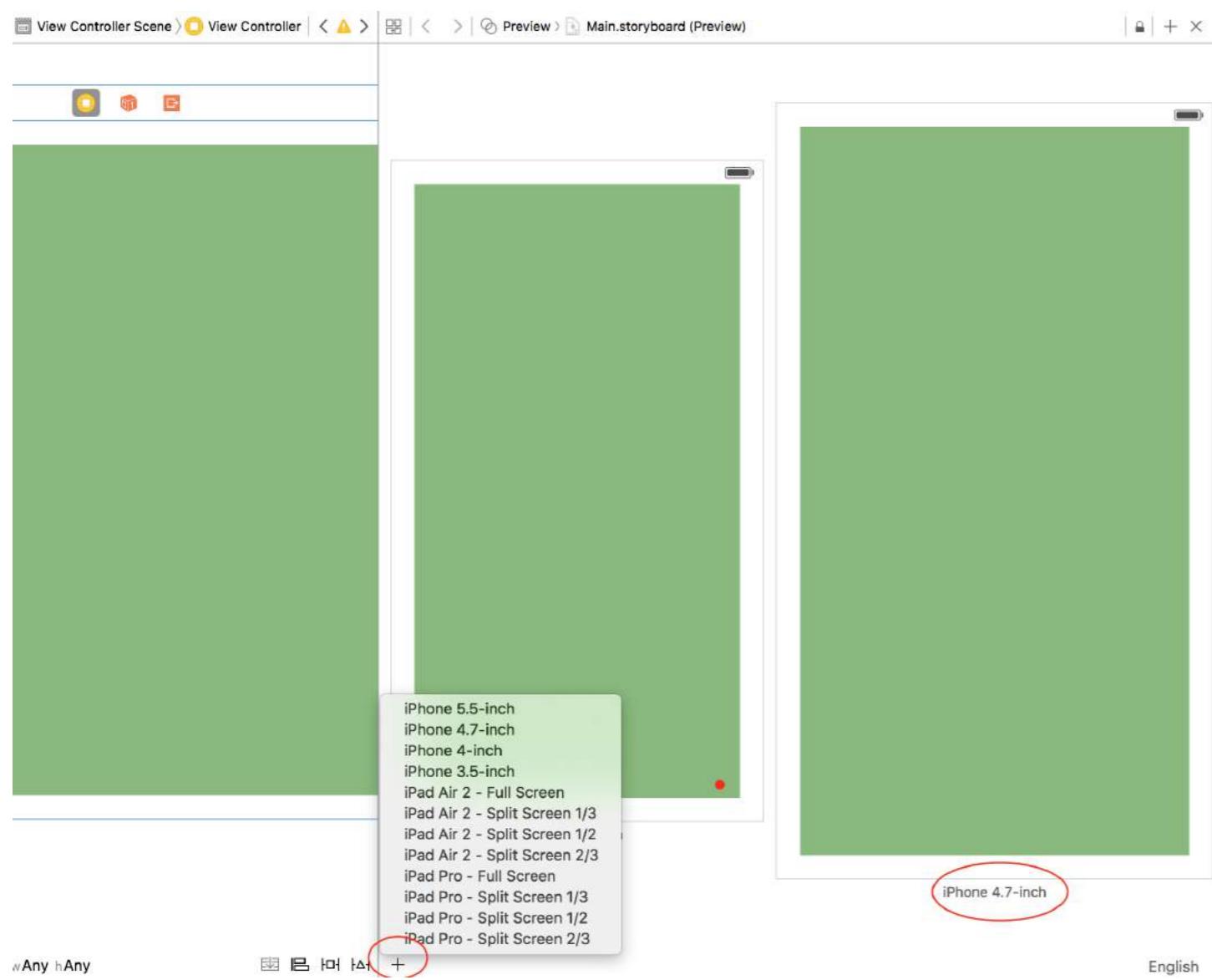
4. To see the preview on screen of these added constraints we can use **Assistant Editor** as;



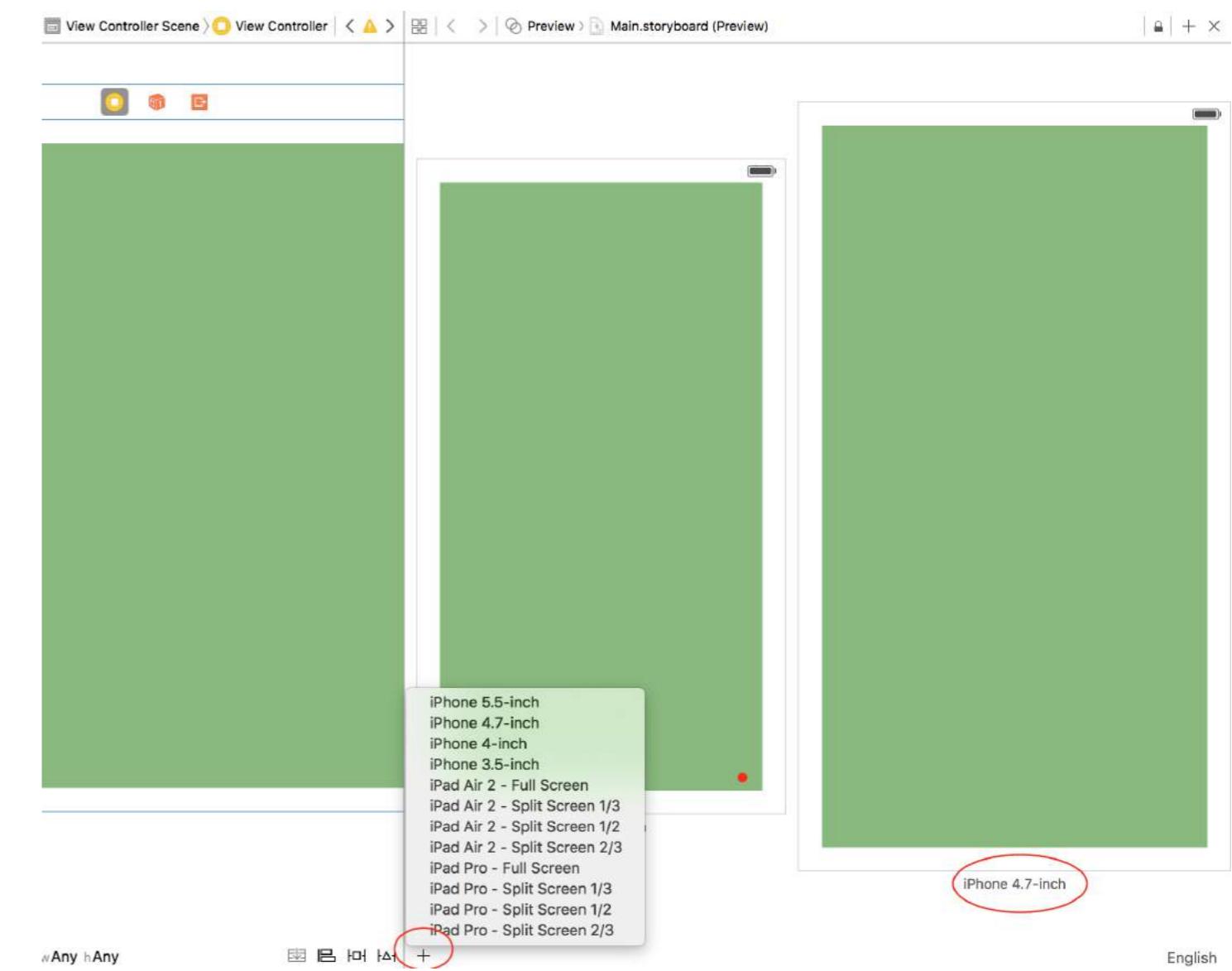
5. 我们可以添加更多屏幕以查看预览，如下所示：



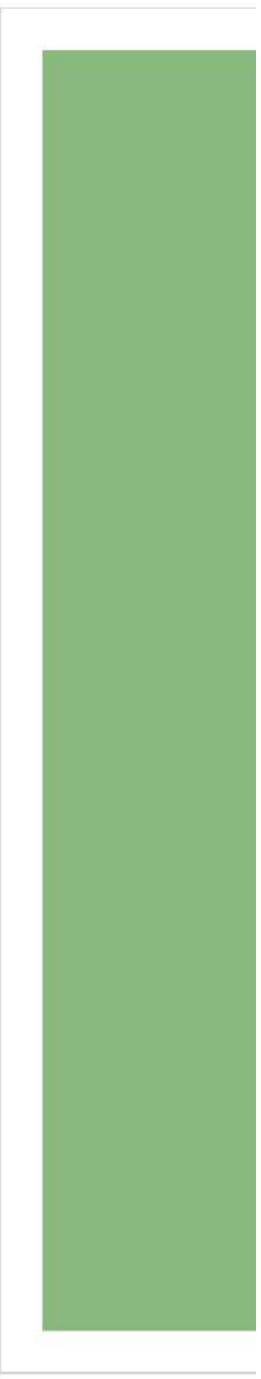
5. We can add more screen to see preview like:



我们还可以通过将鼠标移动到设备名称上并点击旋转按钮来查看横屏模式的预览，如下所示：



We can also see the preview with landscape mode by moving mouse on the name of device and clicking the rotation button as:



第146章：MKDistanceFormatter

第146.1节：从距离生成字符串

给定一个CLLocationDistance（简单来说是表示米数的Double），输出一个用户可读的字符串：

```
let distance = CLLocationDistance(42)
let formatter = MKDistanceFormatter()
let answer = formatter.stringFromDistance(distance)
// answer = "150 feet"
```

Objective-C

```
CLLocationDistance distance=42;
MKDistanceFormatter *formatter=[[MKDistanceFormatter alloc] init];
NSString *answer=[formatter stringFromDistance:distance];
// answer = "150 feet"
```

默认情况下，这会遵循用户的区域设置。

第146.2节：距离单位

import Mapkit 将 units 设置为以下之一.Default, .Metric, .Imperial, .ImperialWithYards :

```
formatter.units = .Metric
var answer = formatter.stringFromDistance(distance)
// "40 m"

formatter.units = .ImperialWithYards
answer = formatter.stringFromDistance(distance)
// "50 yards"
```

Objective-C

```
MKDistanceFormatter *formatter=[[MKDistanceFormatter alloc] init];
formatter.units=MKDistanceFormatterUnitsMetric;
NSString *answer=[formatter stringFromDistance:distance];
//40 m

formatter.units=MKDistanceFormatterUnitsImperialWithYards;
NSString *answer=[formatter stringFromDistance:distance];
//50 yards
```

第146.3节：单位样式

将unitStyle设置为以下之一：.Default、.Abbreviated、.Full：

```
formatter.unitStyle = .Full
var answer = formatter.stringFromDistance(distance)
// "150 英尺"

formatter.unitStyle = .Abbreviated
answer = formatter.stringFromDistance(distance)
// "150 英尺"
```

Chapter 146: MKDistanceFormatter

Section 146.1: String from distance

Given a CLLocationDistance (simply a Double representing meters), output a user-readable string:

```
let distance = CLLocationDistance(42)
let formatter = MKDistanceFormatter()
let answer = formatter.stringFromDistance(distance)
// answer = "150 feet"
```

Objective-C

```
CLLocationDistance distance=42;
MKDistanceFormatter *formatter=[[MKDistanceFormatter alloc] init];
NSString *answer=[formatter stringFromDistance:distance];
// answer = "150 feet"
```

By default, this respects the user's locale.

Section 146.2: Distance units

import Mapkit Set units to one of .Default, .Metric, .Imperial, .ImperialWithYards:

```
formatter.units = .Metric
var answer = formatter.stringFromDistance(distance)
// "40 m"

formatter.units = .ImperialWithYards
answer = formatter.stringFromDistance(distance)
// "50 yards"
```

Objective-C

```
MKDistanceFormatter *formatter=[[MKDistanceFormatter alloc] init];
formatter.units=MKDistanceFormatterUnitsMetric;
NSString *answer=[formatter stringFromDistance:distance];
//40 m

formatter.units=MKDistanceFormatterUnitsImperialWithYards;
NSString *answer=[formatter stringFromDistance:distance];
//50 yards
```

Section 146.3: Unit style

Set unitStyle to one of .Default, .Abbreviated, .Full:

```
formatter.unitStyle = .Full
var answer = formatter.stringFromDistance(distance)
// "150 feet"

formatter.unitStyle = .Abbreviated
answer = formatter.stringFromDistance(distance)
// "150 ft"
```

Objective-C

```
formatter.unitStyle=MKDistanceFormatterUnitStyleFull;
NSString *answer=[formatter stringFromDistance:distance];
// "150 英尺"

formatter.unitStyle=MKDistanceFormatterUnitStyleAbbreviated;
NSString *answer=[formatter stringFromDistance:distance];
// "150 英尺"
```

Objective-C

```
formatter.unitStyle=MKDistanceFormatterUnitStyleFull;
NSString *answer=[formatter stringFromDistance:distance];
// "150 feet"

formatter.unitStyle=MKDistanceFormatterUnitStyleAbbreviated;
NSString *answer=[formatter stringFromDistance:distance];
// "150 ft"
```

第147章：3D触控

第147.1节：使用Swift的3D触控

3D触控首次出现在iPhone 6s Plus中。这个新的界面层增加了两种交互行为：
Peek和Pop。

Peek 和 Pop 简要说明

Peek - 用力按压

Pop - 更用力按压

检查是否支持 3D 触控

你应该检查设备是否支持 3D 触控。你可以通过检查 `UITraitCollection` 对象的 `forceTouchCapability` 属性值来实现。`UITraitCollection` 描述了你的应用的 iOS 界面环境。

```
if (traitCollection.forceTouchCapability == .Available) {  
    registerForPreviewingWithDelegate(self, sourceView: view)  
}
```

实现代理

你需要在你的类中实现 `UIViewControllerPreviewingDelegate` 的两个方法。其中一个方法用于 peek，另一个用于 pop 行为。

用于 peek 的方法是 `previewingContext`。

```
func previewingContext(previewingContext: UIViewControllerPreviewing, viewControllerForLocation  
location: CGPoint) -> UIViewController? {  
  
    guard let indexPath = self.tableViewIndexPathForRowAtPoint(location), cell =  
self.tableView.cellForRowAtIndexPath(indexPath) as? <YourTableViewCell> else {  
        return nil  
    }  
  
    guard let detailVC =  
storyboard?.instantiateViewControllerWithIdentifier("<YourViewControllerIdentifier>") as?  
<YourViewController> else {  
        return nil  
    }  
  
    detailVC.peekActive = true  
    previewingContext.sourceRect = cell.frame  
  
    // 执行操作  
  
    return detailVC  
}
```

实现pop方法的是`previewingContext`。:)

```
func previewingContext(previewingContext: UIViewControllerPreviewing, commitViewController  
viewControllerToCommit: UIViewController) {
```

Chapter 147: 3D Touch

Section 147.1: 3D Touch with Swift

3D touch has been introduced with iPhone 6s Plus. There are two behaviors added with this new interface layer:
Peek and Pop.

Peek and Pop in a nutshell

Peek - Press hard

Pop - Press really hard

Checking for 3D support

You should check if the device has a 3D touch support. You can do this by checking the value of `forceTouchCapability` property of a `UITraitCollection` object. `UITraitCollection` describes the iOS interface environment for your app.

```
if (traitCollection.forceTouchCapability == .Available) {  
    registerForPreviewingWithDelegate(self, sourceView: view)  
}
```

Implementing the delegate

You need to implement the two methods of `UIViewControllerAnimatedDelegate` in your class. One of the methods is for `peek` and the other one is for `pop` behavior.

The method to be implemented for the `peek` is `previewingContext`.

```
func previewingContext(previewingContext: UIViewControllerPreviewing, viewControllerForLocation  
location: CGPoint) -> UIViewController? {  
  
    guard let indexPath = self.tableViewIndexPathForRowAtPoint(location), cell =  
self.tableView.cellForRowAtIndexPath(indexPath) as? <YourTableViewCell> else {  
        return nil  
    }  
  
    guard let detailVC =  
storyboard?.instantiateViewControllerWithIdentifier("<YourViewControllerIdentifier>") as?  
<YourViewController> else {  
        return nil  
    }  
  
    detailVC.peekActive = true  
    previewingContext.sourceRect = cell.frame  
  
    // Do the stuff  
  
    return detailVC  
}
```

The method to be implemented for the `pop` is `previewingContext`.:)

```
func previewingContext(previewingContext: UIViewControllerPreviewing, commitViewController  
viewControllerToCommit: UIViewController) {
```

```

let balanceViewController = viewControllerToCommit as! <YourViewController>

// 执行操作

navigationController?.pushViewController(balanceViewController, animated: true)
}

```

如你所见，它们是重载方法。你可以通过实现这些方法以任何方式使用3D触控。

Objective-C

```

// 检查3D Touch是否可用
if ([self.traitCollection respondsToSelector:@selector(forceTouchCapability)] &&
    (self.traitCollection.forceTouchCapability == UIForceTouchCapabilityAvailable))
{
    [self registerForPreviewingWithDelegate:self sourceView:self.view];
}

// 预览 (Peek)
- (UIViewController *)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
    viewControllerForLocation:(CGPoint)location {
    NSIndexPath *indexPath = [self.tableView indexPathForRowAtPoint:location];
    Country *country = [self countryForIndexPath:indexPath];
    if (country) {
        CountryCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
        if (cell) {
            previewingContext.sourceRect = cell.frame;
            UINavigationController *navController = [self.storyboard
                instantiateViewControllerWithIdentifier:@"UYLCountryNavController"];
            [self configureNavigationController:navController withCountry:country];
            return navController;
        }
    }
    return nil;
}

// 弹出 (Pop)
- (void)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
    commitViewController:(UIViewController *)viewControllerToCommit {
    [self showDetailViewController:viewControllerToCommit sender:self];
}

```

```

let balanceViewController = viewControllerToCommit as! <YourViewController>

// Do the stuff

navigationController?.pushViewController(balanceViewController, animated: true)
}

```

As you can see they are overloaded methods. You can use 3D touch in any way implementing these methods.

Objective-C

```

//Checking for 3-D Touch availability
if ([self.traitCollection respondsToSelector:@selector(forceTouchCapability)] &&
    (self.traitCollection.forceTouchCapability == UIForceTouchCapabilityAvailable))
{
    [self registerForPreviewingWithDelegate:self sourceView:self.view];
}

//Peek
- (UIViewController *)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
    viewControllerForLocation:(CGPoint)location {
    NSIndexPath *indexPath = [self.tableView indexPathForRowAtPoint:location];
    Country *country = [self countryForIndexPath:indexPath];
    if (country) {
        CountryCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
        if (cell) {
            previewingContext.sourceRect = cell.frame;
            UINavigationController *navController = [self.storyboard
                instantiateViewControllerWithIdentifier:@"UYLCountryNavController"];
            [self configureNavigationController:navController withCountry:country];
            return navController;
        }
    }
    return nil;
}

//Pop
- (void)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
    commitViewController:(UIViewController *)viewControllerToCommit {
    [self showDetailViewController:viewControllerToCommit sender:self];
}

```

第147.2节：3D Touch Objective-C示例

Objective-C

```

// 检查3D Touch是否可用
if ([self.traitCollection respondsToSelector:@selector(forceTouchCapability)] &&
    (self.traitCollection.forceTouchCapability == UIForceTouchCapabilityAvailable))
{
    [self registerForPreviewingWithDelegate:self sourceView:self.view];
}

// 预览 (Peek)
- (UIViewController *)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
    viewControllerForLocation:(CGPoint)location {
    NSIndexPath *indexPath = [self.tableView indexPathForRowAtPoint:location];
    Country *country = [self countryForIndexPath:indexPath];
    if (country) {

```

Section 147.2: 3 D Touch Objective-C Example

Objective-C

```

//Checking for 3-D Touch availability
if ([self.traitCollection respondsToSelector:@selector(forceTouchCapability)] &&
    (self.traitCollection.forceTouchCapability == UIForceTouchCapabilityAvailable))
{
    [self registerForPreviewingWithDelegate:self sourceView:self.view];
}

//Peek
- (UIViewController *)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
    viewControllerForLocation:(CGPoint)location {
    NSIndexPath *indexPath = [self.tableView indexPathForRowAtPoint:location];
    Country *country = [self countryForIndexPath:indexPath];
    if (country) {

```

```

CountryCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
    if (cell) {
previewingContext.sourceRect = cell.frame;
    UINavigationController *navController = [self.storyboard
instantiateViewControllerWithIdentifier:@"UYLCountryNavController"];
        [self configureNavigationController:navController withCountry:country];
        return navController;
    }
}
return nil;
}

// 弹出 (Pop)
- (void)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
commitViewController:(UIViewController *)viewControllerToCommit {
    [self showDetailViewController:viewControllerToCommit sender:self];
}

```

```

CountryCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
    if (cell) {
previewingContext.sourceRect = cell.frame;
    UINavigationController *navController = [self.storyboard
instantiateViewControllerWithIdentifier:@"UYLCountryNavController"];
        [self configureNavigationController:navController withCountry:country];
        return navController;
    }
}
return nil;
}

//Pop
- (void)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
commitViewController:(UIViewController *)viewControllerToCommit {
    [self showDetailViewController:viewControllerToCommit sender:self];
}

```

第148章：GameCenter排行榜

第148.1节：GameCenter排行榜

先决条件：

1. Apple开发者账号
2. 通过iTunesConnect设置GameCenter排行榜

设置GameCenter排行榜：

1. 登录 iTunesConnect
2. 进入我的应用。为您的项目创建一个应用，然后进入功能。
3. 点击游戏中心
4. 点击排行榜旁边的加号。
5. 选择单一排行榜作为排行榜类型。
6. 创建一个排行榜参考名称以供参考。
7. 为您的应用创建一个排行榜ID，用于提交分数时引用。
8. 将分数格式设置为整数
9. 分数提交方式为最佳分数
10. 点击添加语言并填写条目。

复制你创建的LeaderboardID，然后我们前往Xcode。

使用Xcode

我们将使用4个函数。

1. 导入框架并设置协议
2. 检查用户是否已登录GameCenter
3. 向GameCenter报告分数
4. 查看排行榜
5. 导入GameKit import GameKit 协议 GKGameCenterControllerDelegate
6. 现在我们要检查用户是否已登录GameCenter

```
func authenticateLocalPlayer() {  
  
    let localPlayer = GKLocalPlayer.localPlayer()  
    localPlayer.authenticateHandler = { (viewController, error) -> Void in  
  
        if viewController != nil {  
            //如果用户尚未登录GameCenter，我们让他们登录  
            let vc:UIViewController = self.view!.window!.rootViewController!  
            vc.presentViewController(viewController!, animated: true, completion: nil)  
  
        } else {  
            //如果需要，可以在这里执行其他操作  
        }  
    }  
}
```

Chapter 148: GameCenter Leaderboards

Section 148.1: GameCenter Leaderboards

Prerequisites:

1. Apple Developers Account
2. Setup GameCenter Leaderboards with iTunesConnect

Setting up GameCenter Leaderboards:

1. Sign in to iTunesConnect
2. Go to My Apps. Create an app for your project then go to Features.
3. Click on Game Center
4. Click the plus sign next to Leaderboards.
5. Choose Single Leaderboard for Leaderboard types.
6. Create a Leaderboard Reference Name for your reference.
7. Create a Leaderboard ID for your app to refer to when reporting scores.
8. Set score format to Integer
9. Score Submission will be Best Score
10. Click Add language and fill the entries.

Copy your LeaderboardID that you made and lets head over to Xcode.

Working with Xcode

There are 4 functions that we will be working with.

1. Importing the framework and setting up the protocols
2. Checking if the user is signed in to GameCenter
3. Reporting the scores to GameCenter
4. Viewing leaderboards
5. Import GameKit import GameKit Protocols GKGameCenterControllerDelegate
6. Now we want to check if the user is signed in to GameCenter

```
func authenticateLocalPlayer() {  
  
    let localPlayer = GKLocalPlayer.localPlayer()  
    localPlayer.authenticateHandler = { (viewController, error) -> Void in  
  
        if viewController != nil {  
            //If the user is not signed in to GameCenter, we make them sign in  
            let vc:UIViewController = self.view!.window!.rootViewController!  
            vc.presentViewController(viewController!, animated: true, completion: nil)  
  
        } else {  
            //Do something here if you want  
        }  
    }  
}
```

3. 现在用户正在使用应用，突然用户获得了新的高分，我们通过调用下面的函数来报告高分。

下面的函数有两个参数。

Identifier 是一个字符串，用于输入你在iTunesConnect中创建的排行榜ID。

score 是一个整数，表示用户提交到iTunesConnect的分数。

```
func saveHighScore(identifier:String, score:Int) {  
    if GKLocalPlayer.localPlayer().authenticated {  
        let scoreReporter = GKScore(leaderboardIdentifier: identifier)  
        scoreReporter.value = Int64(score)  
        let scoreArray:[GKScore] = [scoreReporter]  
        GKScore.reportScores(scoreArray, withCompletionHandler: {  
            error -> Void in  
                if error != nil {  
                    print("Error")  
                } else {  
                }  
            })  
    }  
}
```

4. 现在如果用户想查看排行榜，调用下面的函数

```
//如果调用此函数，将显示GameCenter排行榜和成就。  
func showGameCenter() {  
    let gameCenterViewController = GKGameCenterViewController()  
    gameCenterViewController.gameCenterDelegate = self  
  
    let vc:UIViewController = self.view!.window!.rootViewController!  
    vc.presentViewController(gameCenterViewController, animated: true, completion:nil)  
  
}  
  
//此函数在显示后关闭游戏中心。  
func gameCenterViewControllerDidFinish(gameCenterViewController: GKGameCenterViewController) {  
    gameCenterViewController.dismissViewControllerAnimated(true, completion: nil)  
    self.gameCenterAchievements.removeAll()  
}
```

3. Now the user is using the app and suddenly the user has a new high score, we report the high score by calling the function below.

The function below has 2 parameters.

Identifier which is defined as a string and used to enter your leaderboardID that you made in iTunesConnect.

score which is defined as an Int which will be the users score to submit to iTunesConnect

```
func saveHighScore(identifier:String, score:Int) {  
    if GKLocalPlayer.localPlayer().authenticated {  
        let scoreReporter = GKScore(leaderboardIdentifier: identifier)  
        scoreReporter.value = Int64(score)  
        let scoreArray:[GKScore] = [scoreReporter]  
        GKScore.reportScores(scoreArray, withCompletionHandler: {  
            error -> Void in  
                if error != nil {  
                    print("Error")  
                } else {  
                }  
            })  
    }  
}
```

4. Now if the user wants to view leaderboards, call the function below

```
//This function will show GameCenter leaderboards and Achievements if you call this function.  
func showGameCenter() {  
    let gameCenterViewController = GKGameCenterViewController()  
    gameCenterViewController.gameCenterDelegate = self  
  
    let vc:UIViewController = self.view!.window!.rootViewController!  
    vc.presentViewController(gameCenterViewController, animated: true, completion:nil)  
  
}  
  
//This function closes gameCenter after showing.  
func gameCenterViewControllerDidFinish(gameCenterViewController: GKGameCenterViewController) {  
    gameCenterViewController.dismissViewControllerAnimated(true, completion: nil)  
    self.gameCenterAchievements.removeAll()  
}
```

第149章：钥匙串

第149.1节：向钥匙串添加密码

每个钥匙串项目通常表示为一个CFDictionary。但你也可以在Objective-C中简单使用NSDictionary并利用桥接，或者在Swift中使用Dictionary并显式转换为CFDictionary。

你可以用以下字典构造一个密码：

Swift

```
var dict = [String : AnyObject]()
```

首先，您需要一个键/值对，让钥匙串知道这是一个密码。请注意，因为我们的字典键是一个字符串所以我们必须在Swift 3中显式地将任何CFString转换为String。CFString不能用作Swift字典的键，因为它不可哈希。

Swift

```
dict[kSecClass as String] = kSecClassGenericPassword
```

接下来，我们的密码可能有一系列属性来描述它并帮助我们以后找到它。以下是通用密码的属性列表。

Swift

```
// 密码仅在设备解锁时可访问  
dict[kSecAttrAccessible as String] = kSecAttrAccessibleWhenUnlocked  
// 标签可能帮助您以后找到它  
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString  
// 用户名  
dict[kSecAttrAccount as String] = "我的名字" as CFString  
// 服务名称  
dict[kSecAttrService as String] = "MyService" as CFString
```

最后，我们需要实际的私有数据。请确保不要在内存中保留太久。这必须是CFData。

Swift

```
dict[kSecValueData as String] = "my_password!!".data(using: .utf8) as! CFData
```

最后，钥匙串服务的添加函数需要知道它应该如何返回新构建的钥匙串项目。由于你不应该在内存中长时间持有数据，下面是如何只返回属性的方法：

Swift

```
dict[kSecReturnAttributes as String] = kCFBooleanTrue
```

现在我们已经构建了项目。让我们添加它：

Swift

```
var result: AnyObject?  
let status = withUnsafeMutablePointer(to: &result) {  
    SecItemAdd(dict as CFDictionary, UnsafeMutablePointer($0))  
}  
let newAttributes = result as! Dictionary<String, AnyObject>
```

Chapter 149: Keychain

Section 149.1: Adding a Password to the Keychain

Every Keychain Item is most often represented as a [CFDictionary](#). You can, however, simply use [NSDictionary](#) in Objective-C and take advantage of bridging, or in Swift you may use [Dictionary](#) and explicitly cast to [CFDictionary](#).

You could construct a password with the following dictionary:

Swift

```
var dict = [String : AnyObject]()
```

First, you need a key/value pair that lets the Keychain know this is a password. Note that because our dict key is a [String](#) we must cast any [CFString](#) to a [String](#) explicitly in Swift 3. CFString may not be used as the key to a Swift Dictionary because it is not Hashable.

Swift

```
dict[kSecClass as String] = kSecClassGenericPassword
```

Next, our password may have a series of attributes to describe it and help us find it later. [Here's a list of attributes for generic passwords](#).

Swift

```
// The password will only be accessible when the device is unlocked  
dict[kSecAttrAccessible as String] = kSecAttrAccessibleWhenUnlocked  
// Label may help you find it later  
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString  
// Username  
dict[kSecAttrAccount as String] = "My Name" as CFString  
// Service name  
dict[kSecAttrService as String] = "MyService" as CFString
```

Finally, we need our actual private data. Be sure not to keep this around in memory for too long. This must be CFData.

Swift

```
dict[kSecValueData as String] = "my_password!!".data(using: .utf8) as! CFData
```

Finally, the Keychain Services add function wants to know how it should return the newly constructed keychain item. Since you shouldn't be holding on to the data very long in memory, here's how you could only return the attributes:

Swift

```
dict[kSecReturnAttributes as String] = kCFBooleanTrue
```

Now we have constructed our item. Let's add it:

Swift

```
var result: AnyObject?  
let status = withUnsafeMutablePointer(to: &result) {  
    SecItemAdd(dict as CFDictionary, UnsafeMutablePointer($0))  
}  
let newAttributes = result as! Dictionary<String, AnyObject>
```

这将新的属性字典放入result中。SecItemAdd接收我们构建的字典，以及一个指向我们希望存放结果的位置的指针。该函数随后返回一个OSStatus，指示成功或错误代码。结果代码在此处有描述。

第149.2节：钥匙串访问控制（带密码回退的TouchID）

钥匙串允许保存带有特殊 SecAccessControl 属性的项目，该属性仅在用户通过 Touch ID（或如果允许此类备用方案，则通过密码）进行身份验证后才能从钥匙串中获取项目。应用程序仅会收到身份验证是否成功的通知，整个用户界面由 iOS 管理。

首先，应创建 SecAccessControl 对象：

Swift

```
let error: Unmanaged<CFError>?

guard let accessControl = SecAccessControlCreateWithFlags(kCFAllocatorDefault,
kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly, .userPresence, &error) else {
    fatalError("Something went wrong")
}
```

接下来，将其添加到字典中，使用 kSecAttrAccessControl 键（该键与您在其他示例中使用的 kSecAttrAccessible 键互斥）：

Swift

```
var dictionary = [String : Any]()

dictionary[kSecClass as String] = kSecClassGenericPassword
dictionary[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
dictionary[kSecAttrAccount as String] = "My Name" as CFString
dictionary[kSecValueData as String] = "new_password!!".data(using: .utf8) as! CFData
dictionary[kSecAttrAccessControl as String] = accessControl
```

然后像之前一样保存它：

Swift

```
let lastResultCode = SecItemAdd(query as CFDictionary, nil)
```

要访问存储的数据，只需查询钥匙串中的密钥。钥匙串服务将向用户显示身份验证对话框，并根据是否提供了合适的指纹或匹配了密码返回数据或 nil。

可选地，可以指定提示字符串：

Swift

```
var query = [String: Any]()

query[kSecClass as String] = kSecClassGenericPassword
query[kSecReturnData as String] = kCFBooleanTrue
query[kSecAttrAccount as String] = "我的名字" as CFString
query[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
query[kSecUseOperationPrompt as String] = "请将手指放在那个按钮上" as CFString

var queryResult: AnyObject?
let status = withUnsafeMutablePointer(to: &queryResult) {
    SecItemCopyMatching(query as CFDictionary, UnsafeMutablePointer($0))
```

This places the new attributes dict inside result. SecItemAdd takes in the dictionary we constructed, as well as a pointer to where we would like our result. The function then returns an OSStatus indicating success or an error code. Result codes are described [here](#).

Section 149.2: Keychain Access Control (TouchID with password fallback)

Keychain allows to save items with special SecAccessControl attribute which will allow to get item from Keychain only after user will be authenticated with Touch ID (or passcode if such fallback is allowed). App is only notified whether the authentication was successful or not, whole UI is managed by iOS.

First, SecAccessControl object should be created:

Swift

```
let error: Unmanaged<CFError>?

guard let accessControl = SecAccessControlCreateWithFlags(kCFAllocatorDefault,
kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly, .userPresence, &error) else {
    fatalError("Something went wrong")
}
```

Next, add it to the dictionary with kSecAttrAccessControl key (which is mutually exclusive with kSecAttrAccessible key you've been using in other examples):

Swift

```
var dictionary = [String : Any]()

dictionary[kSecClass as String] = kSecClassGenericPassword
dictionary[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
dictionary[kSecAttrAccount as String] = "My Name" as CFString
dictionary[kSecValueData as String] = "new_password!!".data(using: .utf8) as! CFData
dictionary[kSecAttrAccessControl as String] = accessControl
```

And save it as you've done before:

Swift

```
let lastResultCode = SecItemAdd(query as CFDictionary, nil)
```

To access stored data, just query Keychain for a key. Keychain Services will present authentication dialog to the user and return data or nil depending on whether suitable fingerprint was provided or passcode matched.

Optionally, prompt string can be specified:

Swift

```
var query = [String: Any]()

query[kSecClass as String] = kSecClassGenericPassword
query[kSecReturnData as String] = kCFBooleanTrue
query[kSecAttrAccount as String] = "My Name" as CFString
query[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
query[kSecUseOperationPrompt as String] = "Please put your fingers on that button" as CFString

var queryResult: AnyObject?
let status = withUnsafeMutablePointer(to: &queryResult) {
    SecItemCopyMatching(query as CFDictionary, UnsafeMutablePointer($0))
```

```
}
```

注意，若用户拒绝、取消或授权失败，status 将为 err。

Swift

```
if status == noErr {
    let password = String(data: queryResult as! Data, encoding: .utf8)!
    print("密码: \(password)")
} else {
    print("授权未通过")
}
```

第149.3节：在钥匙串中查找密码

要构造查询，我们需要将其表示为一个`CFDictionary`。你也可以在Objective-C中使用`NSDictionary`，或在Swift中使用`Dictionary`并转换为`CFDictionary`。

我们需要一个类键：

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
```

接下来，我们可以指定属性以缩小搜索范围：

Swift

```
// 标签
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// 用户名
dict[kSecAttrAccount as String] = "我的名字" as CFString
// 服务名称
dict[kSecAttrService as String] = "MyService" as CFString
```

我们还可以指定此处描述的特殊搜索修饰键。

最后，我们需要说明希望如何返回数据。下面，我们将请求仅返回私有密码本身，作为一个`CFData`对象：

Swift

```
dict[kSecReturnData as String] = kCFBooleanTrue
```

现在，开始搜索：

Swift

```
var queryResult: AnyObject?
let status = withUnsafeMutablePointer(to: &queryResult) {
    SecItemCopyMatching(dict as CFDictionary, UnsafeMutablePointer($0))
}
// 不要长时间将此保存在内存中！！
let password = String(data: queryResult as! Data, encoding: .utf8)!
```

这里，`SecItemCopyMatching` 接受一个查询字典和一个指向你希望结果存放位置的指针。它返回一个`OSStatus`结果代码。以下是可能的结果。

```
}
```

Pay attention that status will be err if user declined, canceled or failed authorization.

Swift

```
if status == noErr {
    let password = String(data: queryResult as! Data, encoding: .utf8)!
    print("Password: \(password)")
} else {
    print("Authorization not passed")
}
```

Section 149.3: Finding a Password in the Keychain

To construct a query, we need to represent it as a `CFDictionary`. You may also use `NSDictionary` in Objective-C or `Dictionary` in Swift and cast to `CFDictionary`.

We need a class key:

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
```

Next, we can specify attributes to narrow down our search:

Swift

```
// Label
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
dict[kSecAttrAccount as String] = "My Name" as CFString
// Service name
dict[kSecAttrService as String] = "MyService" as CFString
```

We can also specify special search modifier keys described [here](#).

Finally, we need to say how we'd like our data returned. Below, we'll request that just the private password itself be returned as a `CFData` object:

Swift

```
dict[kSecReturnData as String] = kCFBooleanTrue
```

Now, let's search:

Swift

```
var queryResult: AnyObject?
let status = withUnsafeMutablePointer(to: &queryResult) {
    SecItemCopyMatching(dict as CFDictionary, UnsafeMutablePointer($0))
}
// Don't keep this in memory for long!!
let password = String(data: queryResult as! Data, encoding: .utf8)!
```

Here, `SecItemCopyMatching` takes in a query dictionary and a pointer to where you'd like the result to go. It returns an `OSStatus` with a result codes. [Here](#) are the possibilities.

第149.4节：在钥匙串中更新密码

和往常一样，我们首先需要一个CFDictionary来表示我们想要更新的项目。这个字典必须包含该项目的所有旧值，包括旧的私有数据。然后它接受一个CFDictionary，包含你想要更改的任何属性或数据本身。

所以首先，让我们构建一个类键和一个属性列表。这些属性可以缩小我们的搜索范围，但如果要更改它们，必须包含任何属性及其旧值。

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
// 标签
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// 用户名
dict[kSecAttrAccount as String] = "My Name" as CFString
```

现在我们必须添加旧数据：

Swift

```
dict[kSecValueData as String] = "my_password!!".data(using: .utf8) as! CFData
```

现在让我们创建相同的属性，但使用不同的密码：

Swift

```
var newDict = [String : AnyObject]()
newDict[kSecClass as String] = kSecClassGenericPassword
// 标签
newDict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// 用户名
newDict[kSecAttrAccount as String] = "我的名字" as CFString
// 新密码
newDict[kSecValueData as String] = "new_password!!".data(using: .utf8) as! CFData
```

现在，我们只需将其传递给钥匙串服务：

Swift

```
let status = SecItemUpdate(dict as CFDictionary, newDict as CFDictionary)
```

SecItemUpdate 返回一个状态码。结果描述见 [here](#)。

第149.5节：从钥匙串中移除密码

我们只需要一件事来从钥匙串中删除一个项目：一个包含描述要删除项目属性的 CFDictionary。任何匹配查询字典的项目都会被永久删除，因此如果你只打算删除单个项目，请确保查询足够具体。和往常一样，我们可以在Objective-C中使用 NSDictionary，或者在Swift中使用 Dictionary 然后转换为 CFDictionary。

在此上下文中，查询字典仅包含一个类键来描述项目类型和描述项目信息的属性。不允许包含诸如 kSecMatchCaseInsensitive 之类的搜索限制。

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
```

Section 149.4: Updating a Password in the Keychain

As usual, we first need a [CFDictionary](#) to represent the item we want to update. This must contain all of the old values for the item, including the old private data. Then it takes a [CFDictionary](#) of any attributes or the data itself that you would like to change.

So first, let's construct a class key and a list of attributes. These attributes can narrow our search but you must include any attributes and their old values if you will be changing them.

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
// Label
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
dict[kSecAttrAccount as String] = "My Name" as CFString
```

Now we must add the old data:

Swift

```
dict[kSecValueData as String] = "my_password!!".data(using: .utf8) as! CFData
```

Now let's create the same attributes but a different password:

Swift

```
var newDict = [String : AnyObject]()
newDict[kSecClass as String] = kSecClassGenericPassword
// Label
newDict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
newDict[kSecAttrAccount as String] = "My Name" as CFString
// New password
newDict[kSecValueData as String] = "new_password!!".data(using: .utf8) as! CFData
```

Now, we just pass it to Keychain Services:

Swift

```
let status = SecItemUpdate(dict as CFDictionary, newDict as CFDictionary)
```

SecItemUpdate returns a status code. Results are described [here](#).

Section 149.5: Removing a Password from the Keychain

We need only one thing in order to delete an item from the Keychain: a [CFDictionary](#) with attributes describing the items to be deleted. Any items that match the query dictionary will be deleted permanently, so if you are only intending to delete a single item be sure to be specific with your query. As always, we can use an [NSDictionary](#) in Objective-C or in Swift we can use a [Dictionary](#) and then cast to [CFDictionary](#).

A query dictionary, in this context exclusively includes a class key to describe what the item is and attributes to describe information about the item. Inclusion of search restrictions such as [kSecMatchCaseInsensitive](#) is not allowed.

Swift

```
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
```

```
// 标签
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// 用户名
dict[kSecAttrAccount as String] = "My Name" as CFString
```

现在我们可以简单地将其移除：

Swift

```
let status = SecItemDelete(dict as CFDictionary)
```

SecItemDelete 返回一个 OSStatus。结果代码描述见 [here](#)。

第149.6节：使用一个文件进行钥匙串的添加、更新、删除和查找操作

Keychain.h

```
#import <Foundation/Foundation.h>
typedef void (^KeychainOperationBlock)(BOOL successfulOperation, NSData *data, OSStatus status);

@interface Keychain : NSObject

-(id) initWithService:(NSString *) service_ withGroup:(NSString*)group_;

-(void)insertKey:(NSString *)keyWithData:(NSData *)data
withCompletion:(KeychainOperationBlock)completionBlock;
-(void)updateKey:(NSString*)keyWithData:(NSData*) data
withCompletion:(KeychainOperationBlock)completionBlock;
-(void)removeDataForKey:(NSString*)key withCompletionBlock:(KeychainOperationBlock)completionBlock;
-(void)findDataForKey:(NSString*)key withCompletionBlock:(KeychainOperationBlock)completionBlock;

@end
```

Keychain.m

```
#import "Keychain.h"
#import <Security/Security.h>

@implementation Keychain

{
    NSString * keychainService;
    NSString * keychainGroup;
}

-(id) initWithService:(NSString *)service withGroup:(NSString*)group
{
    self =[super init];
    if(self) {
        keychainService = [NSString stringWithString:service];
        if(group) {
            keychainGroup = [NSString stringWithString:group];
        }
    }
    return self;
}

-(void)insertKey:(NSString *)key
```

```
// Label
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
dict[kSecAttrAccount as String] = "My Name" as CFString
```

And now we can simply remove it:

Swift

```
let status = SecItemDelete(dict as CFDictionary)
```

SecItemDelete returns an OSStatus. Result codes are described [here](#).

Section 149.6: Keychain Add, Update, Remove and Find operations using one file

Keychain.h

```
#import <Foundation/Foundation.h>
typedef void (^KeychainOperationBlock)(BOOL successfulOperation, NSData *data, OSStatus status);

@interface Keychain : NSObject

-(id) initWithService:(NSString *) service_ withGroup:(NSString*)group_;

-(void)insertKey:(NSString *)keyWithData:(NSData *)data
withCompletion:(KeychainOperationBlock)completionBlock;
-(void)updateKey:(NSString*)keyWithData:(NSData*) data
withCompletion:(KeychainOperationBlock)completionBlock;
-(void)removeDataForKey:(NSString*)key withCompletionBlock:(KeychainOperationBlock)completionBlock;
-(void)findDataForKey:(NSString*)key withCompletionBlock:(KeychainOperationBlock)completionBlock;

@end
```

Keychain.m

```
#import "Keychain.h"
#import <Security/Security.h>

@implementation Keychain

{
    NSString * keychainService;
    NSString * keychainGroup;
}

-(id) initWithService:(NSString *)service withGroup:(NSString*)group
{
    self =[super init];
    if(self) {
        keychainService = [NSString stringWithString:service];
        if(group) {
            keychainGroup = [NSString stringWithString:group];
        }
    }
    return self;
}

-(void)insertKey:(NSString *)key
```

```

withData:(NSData *)data
    withCompletion:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary * dict =[self prepareDict:key];
    [dict setObject:data forKey:(__bridge id)kSecValueData];
    [dict setObject:keychainService forKey:(id)kSecAttrService];

OSStatus status = SecItemAdd((__bridge CFDictionaryRef)dict, NULL);
    if(errSecSuccess != status) {
        DLog(@"Unable add item with key %@ error:%d",key,(int)status);
        if (completionBlock) {
            completionBlock(errSecSuccess == status, nil, status);
        }
    }
    if (status == errSecDuplicateItem) {
        [self updateKey:key withData:data withCompletion:^(BOOL successfulOperation, NSData *updateData, OSStatus updateStatus) {
            if (completionBlock) {
                completionBlock(successfulOperation, updateData, updateStatus);
            }
        }];
        DLog(@"发现重复项 -- 使用数据更新键");
    }
}

-(void)findDataForKey:(NSString *)key
    withCompletionBlock:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary *dict = [self prepareDict:key];
    [dict setObject:(__bridge id)kSecMatchLimitOne forKey:(__bridge id)kSecMatchLimit];
    [dict setObject:keychainService forKey:(id)kSecAttrService];
    [dict setObject:(id)kCFBooleanTrue forKey:(__bridge id)kSecReturnData];
    CFTypeRef result = NULL;
OSStatus status = SecItemCopyMatching((__bridge CFDictionaryRef)dict,&result);

    if( status != errSecSuccess) {
        DLog(@"无法获取键 %@ 的项目，错误码:%d",key,(int)status);
        if (completionBlock) {
            completionBlock(errSecSuccess == status, nil, status);
        }
    } else {
        if (completionBlock) {
            completionBlock(errSecSuccess == status, (__bridge NSData *)result, status);
        }
    }
}

-(void)updateKey:(NSString *)key
   WithData:(NSData *)data
    withCompletion:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary * dictKey =[self prepareDict:key];

    NSMutableDictionary * dictUpdate =[[NSMutableDictionary alloc] init];
    [dictUpdate setObject:data forKey:(__bridge id)kSecValueData];
    [dictUpdate setObject:keychainService forKey:(id)kSecAttrService];
    OSStatus status = SecItemUpdate((__bridge CFDictionaryRef)dictKey, (__bridge CFDictionaryRef)dictUpdate);
    if( status != errSecSuccess) {
        DLog(@"无法删除键 %@ 的项目，错误码:%d",key,(int)status);
    }
    if (completionBlock) {
}

```

```

   WithData:(NSData *)data
    withCompletion:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary * dict =[self prepareDict:key];
    [dict setObject:data forKey:(__bridge id)kSecValueData];
    [dict setObject:keychainService forKey:(id)kSecAttrService];

OSStatus status = SecItemAdd((__bridge CFDictionaryRef)dict, NULL);
    if(errSecSuccess != status) {
        DLog(@"Unable add item with key %@ error:%d",key,(int)status);
        if (completionBlock) {
            completionBlock(errSecSuccess == status, nil, status);
        }
    }
    if (status == errSecDuplicateItem) {
        [self updateKey:key withData:data withCompletion:^(BOOL successfulOperation, NSData *updateData, OSStatus updateStatus) {
            if (completionBlock) {
                completionBlock(successfulOperation, updateData, updateStatus);
            }
        }];
        DLog(@"Found duplication item -- updating key with data");
    }
}

-(void)findDataForKey:(NSString *)key
    withCompletionBlock:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary *dict = [self prepareDict:key];
    [dict setObject:(__bridge id)kSecMatchLimitOne forKey:(__bridge id)kSecMatchLimit];
    [dict setObject:keychainService forKey:(id)kSecAttrService];
    [dict setObject:(id)kCFBooleanTrue forKey:(__bridge id)kSecReturnData];
    CFTypeRef result = NULL;
OSStatus status = SecItemCopyMatching((__bridge CFDictionaryRef)dict,&result);

    if( status != errSecSuccess) {
        DLog(@"Unable to fetch item for key %@ with error:%d",key,(int)status);
        if (completionBlock) {
            completionBlock(errSecSuccess == status, nil, status);
        }
    } else {
        if (completionBlock) {
            completionBlock(errSecSuccess == status, (__bridge NSData *)result, status);
        }
    }
}

-(void)updateKey:(NSString *)key
   WithData:(NSData *)data
    withCompletion:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary * dictKey =[self prepareDict:key];

    NSMutableDictionary * dictUpdate =[[NSMutableDictionary alloc] init];
    [dictUpdate setObject:data forKey:(__bridge id)kSecValueData];
    [dictUpdate setObject:keychainService forKey:(id)kSecAttrService];
    OSStatus status = SecItemUpdate((__bridge CFDictionaryRef)dictKey, (__bridge CFDictionaryRef)dictUpdate);
    if( status != errSecSuccess) {
        DLog(@"Unable to remove item for key %@ with error:%d",key,(int)status);
    }
    if (completionBlock) {
}

```

```

completionBlock(errSecSuccess == status, nil, status);
}

-(void)removeDataForKey:(NSString *)key
withCompletionBlock:(KeychainOperationBlock)completionBlock {
    NSMutableDictionary *dict = [self prepareDict:key];
OSStatus status = SecItemDelete((__bridge CFDictionaryRef)dict);
    if( status != errSecSuccess) {
        DLog(@"无法删除键 %@ 的项目，错误码:%d",key,(int)status);
    }
    if (completionBlock) {
completionBlock(errSecSuccess == status, nil, status);
    }
}

#pragma mark 内部方法

-(NSMutableDictionary*) prepareDict:(NSString *) key {

    NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
    [dict setObject:(__bridge id)kSecClassGenericPassword forKey:(__bridge id)kSecClass];

    NSData *encodedKey = [key dataUsingEncoding:NSUTF8StringEncoding];
    [dict setObject:encodedKey forKey:(__bridge id)kSecAttrGeneric];
    [dict setObject:encodedKey forKey:(__bridge id)kSecAttrAccount];
    [dict setObject:keychainService forKey:(__bridge id)kSecAttrService];
    [dict setObject:(__bridge id)kSecAttrAccessibleAlwaysThisDeviceOnly forKey:(__bridge id)kSecAttrAccessible];

    //这是用于跨应用共享数据
    if(keychainGroup != nil) {
        [dict setObject:keychainGroup forKey:(__bridge id)kSecAttrAccessGroup];
    }

    return dict;
}

@end

```

```

completionBlock(errSecSuccess == status, nil, status);
}

-(void)removeDataForKey:(NSString *)key
withCompletionBlock:(KeychainOperationBlock)completionBlock {
    NSMutableDictionary *dict = [self prepareDict:key];
OSStatus status = SecItemDelete((__bridge CFDictionaryRef)dict);
    if( status != errSecSuccess) {
        DLog(@"Unable to remove item for key %@ with error:%d",key,(int)status);
    }
    if (completionBlock) {
completionBlock(errSecSuccess == status, nil, status);
    }
}

#pragma mark Internal methods

-(NSMutableDictionary*) prepareDict:(NSString *) key {

    NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
    [dict setObject:(__bridge id)kSecClassGenericPassword forKey:(__bridge id)kSecClass];

    NSData *encodedKey = [key dataUsingEncoding:NSUTF8StringEncoding];
    [dict setObject:encodedKey forKey:(__bridge id)kSecAttrGeneric];
    [dict setObject:encodedKey forKey:(__bridge id)kSecAttrAccount];
    [dict setObject:keychainService forKey:(__bridge id)kSecAttrService];
    [dict setObject:(__bridge id)kSecAttrAccessibleAlwaysThisDeviceOnly forKey:(__bridge id)kSecAttrAccessible];

    //This is for sharing data across apps
    if(keychainGroup != nil) {
        [dict setObject:keychainGroup forKey:(__bridge id)kSecAttrAccessGroup];
    }

    return dict;
}

@end

```

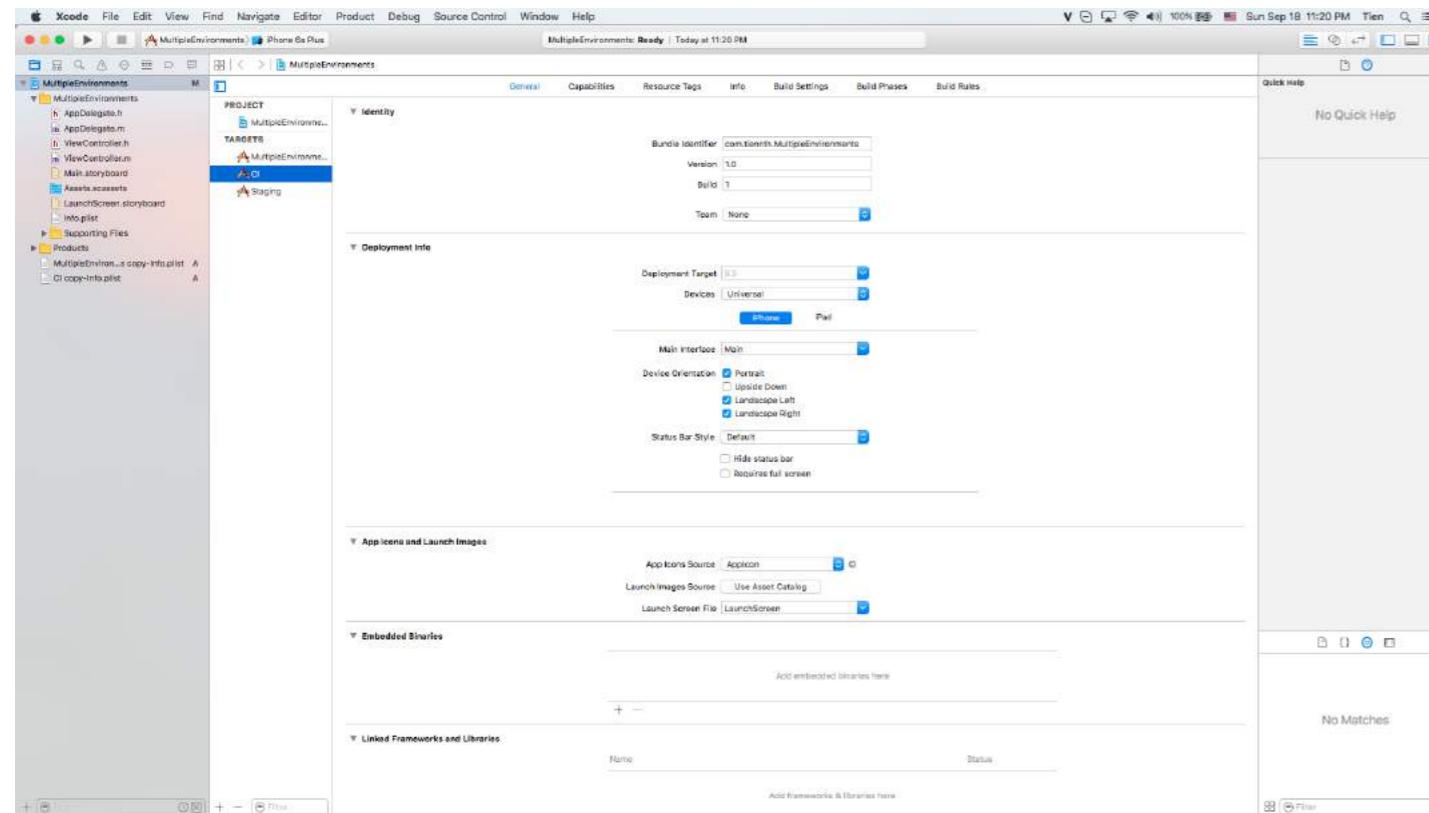
第150章：使用宏处理多环境

第150.1节：使用多个目标和宏处理多环境

例如，我们有两个环境：CI - 预发布，并希望为每个环境添加一些自定义设置。

这里我将尝试自定义服务器URL、应用名称。

首先，我们通过复制主目标创建两个环境的两个目标：



对于每个目标，我们将定义一个自定义宏。这里我将在目标 CI 的构建设置中定义名为“CI”的宏，在目标 Staging 中定义名为“STAGING”的宏。

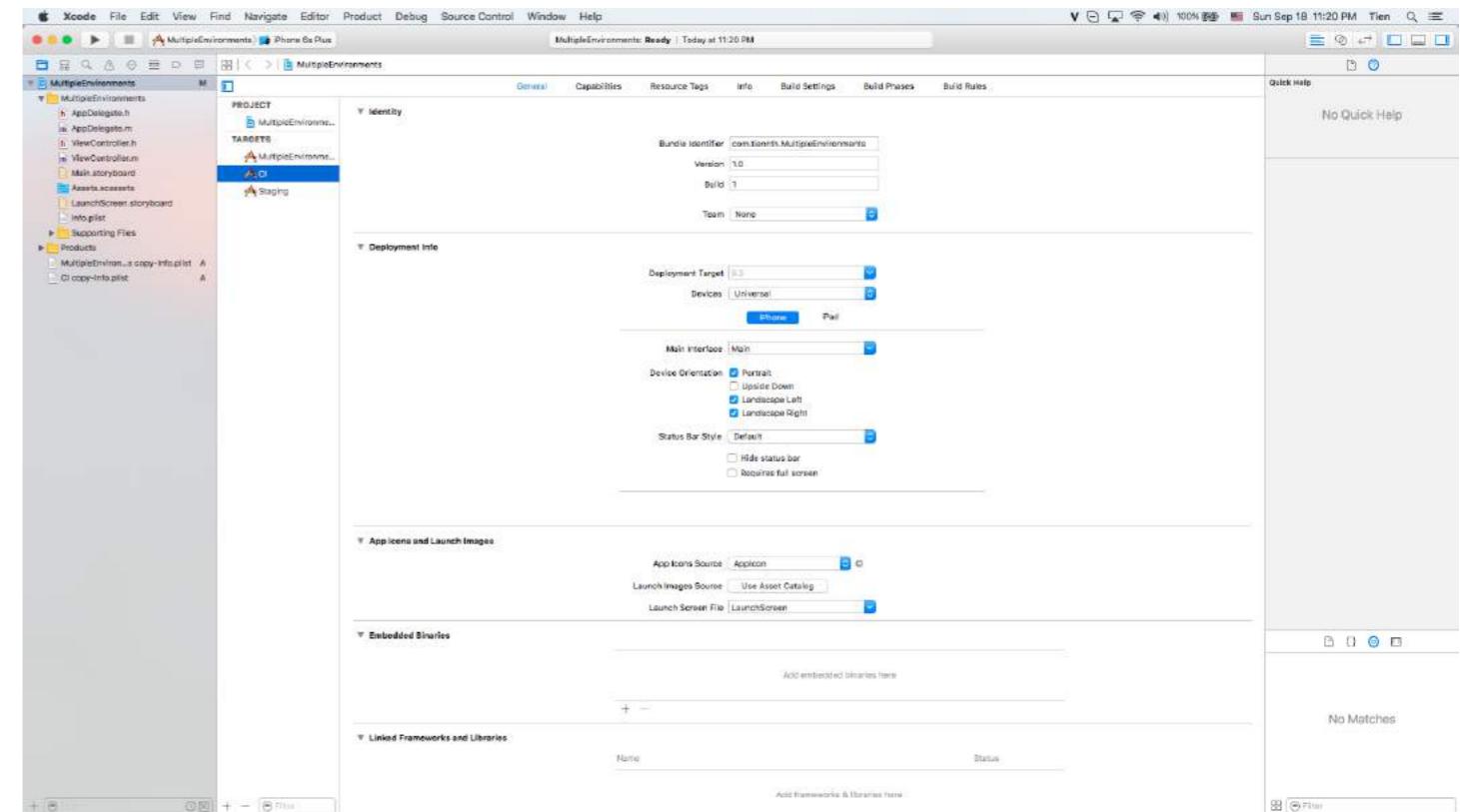
开发目标（MultipleEnvironment 目标）：

Chapter 150: Handle Multiple Environment using Macro

Section 150.1: Handle multiple environment using multiple target and macro

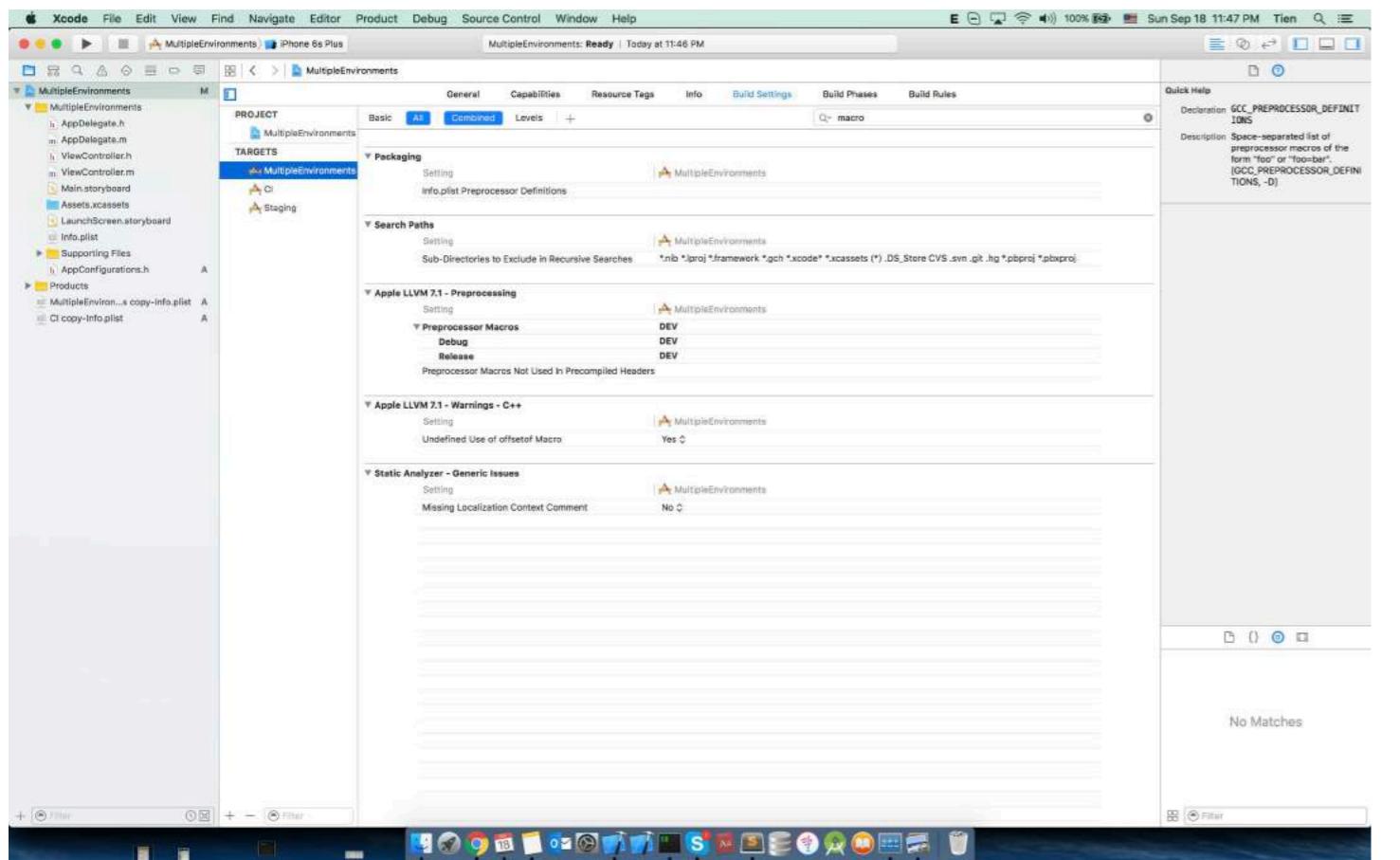
For example, we have two environments: CI - Staging and want to add some customizations for each environment. Here I will try to customize server URL, app name.

First, we create two targets for 2 environments by duplicating the main target:

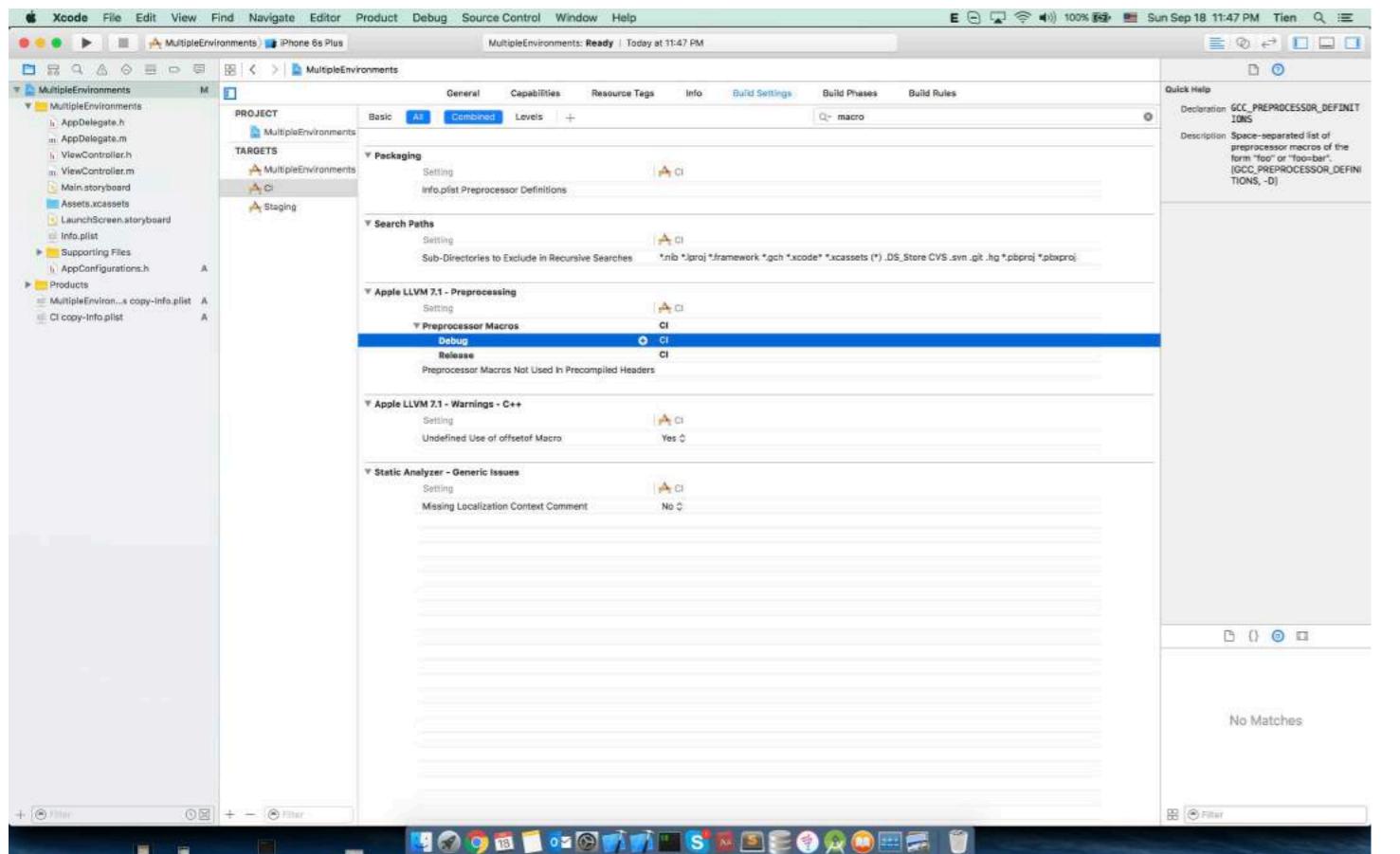


For each target, we will define a custom macro. Here I will define macro named "CI" in build settings of target CI, macro named "STAGING" for target Staging.

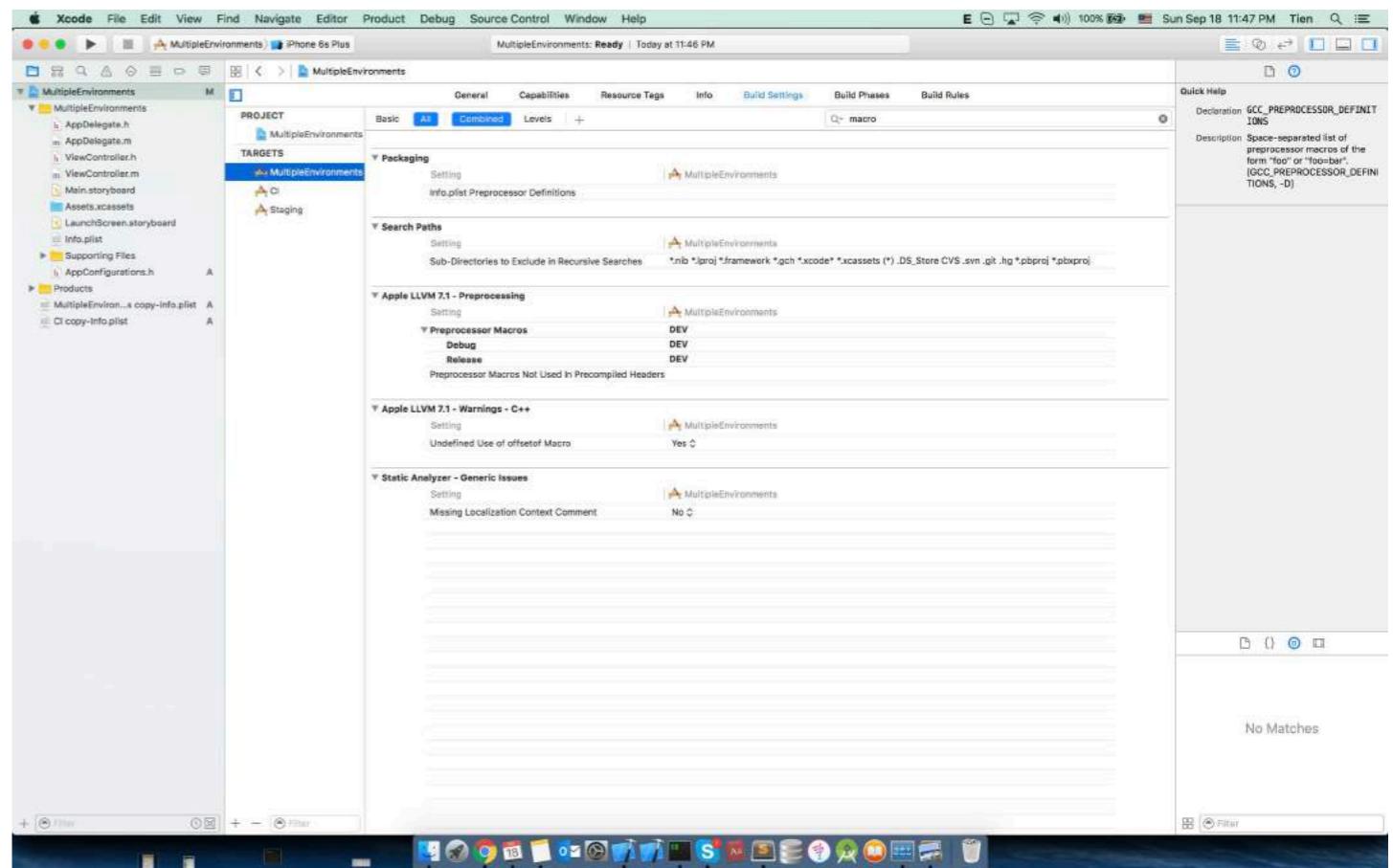
The development target (MultipleEnvironment target):



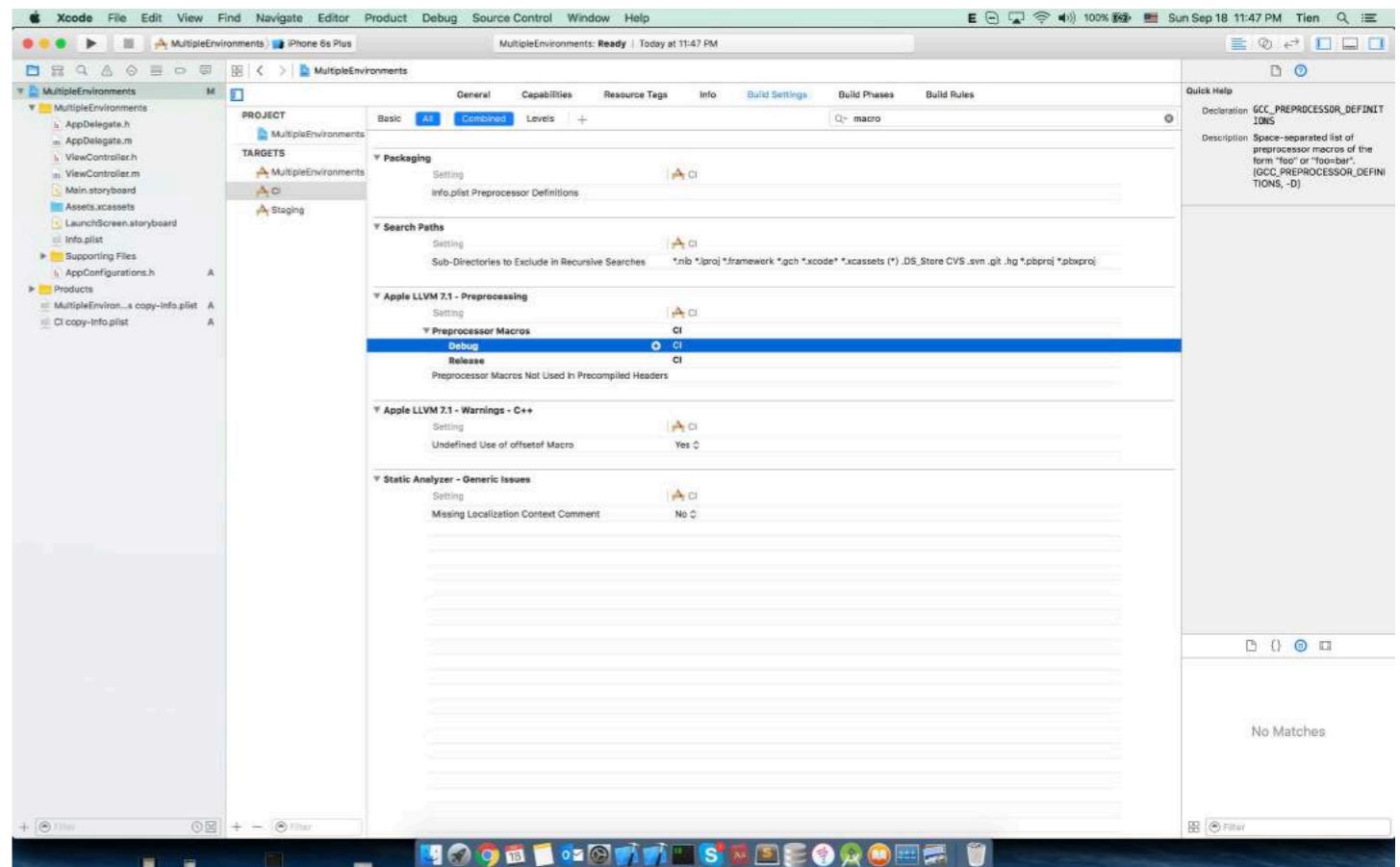
目标 CI :



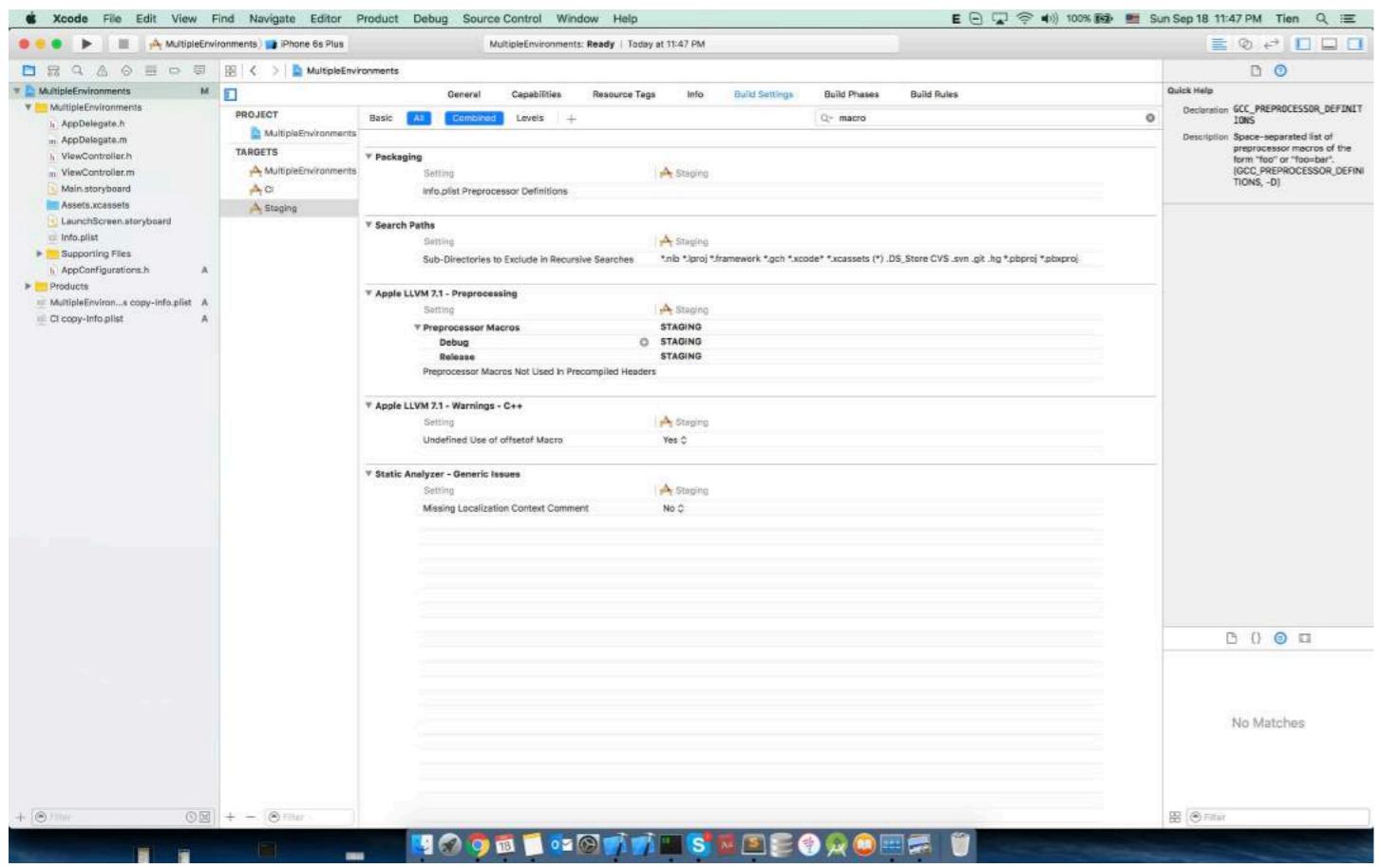
目标 Staging :



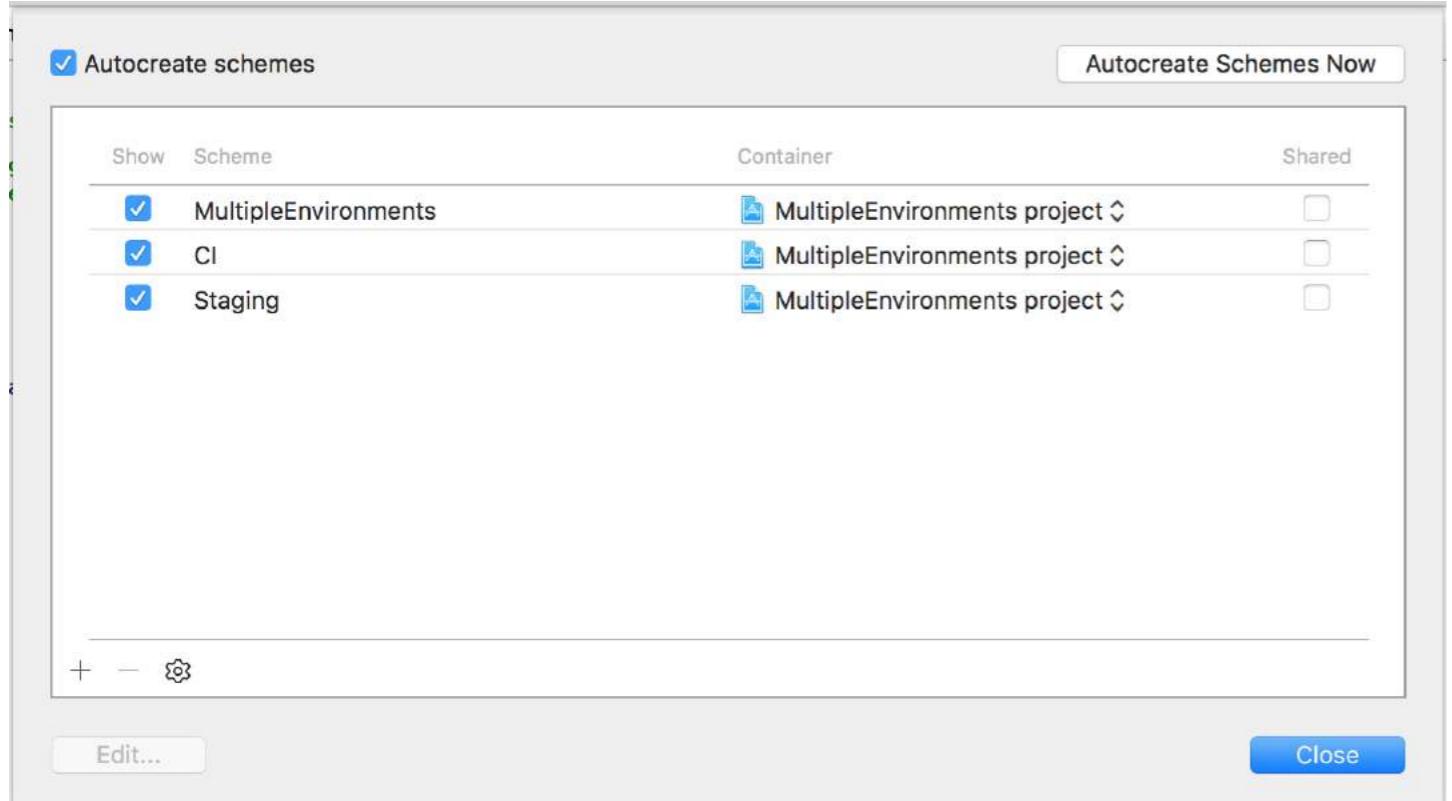
Target CI:



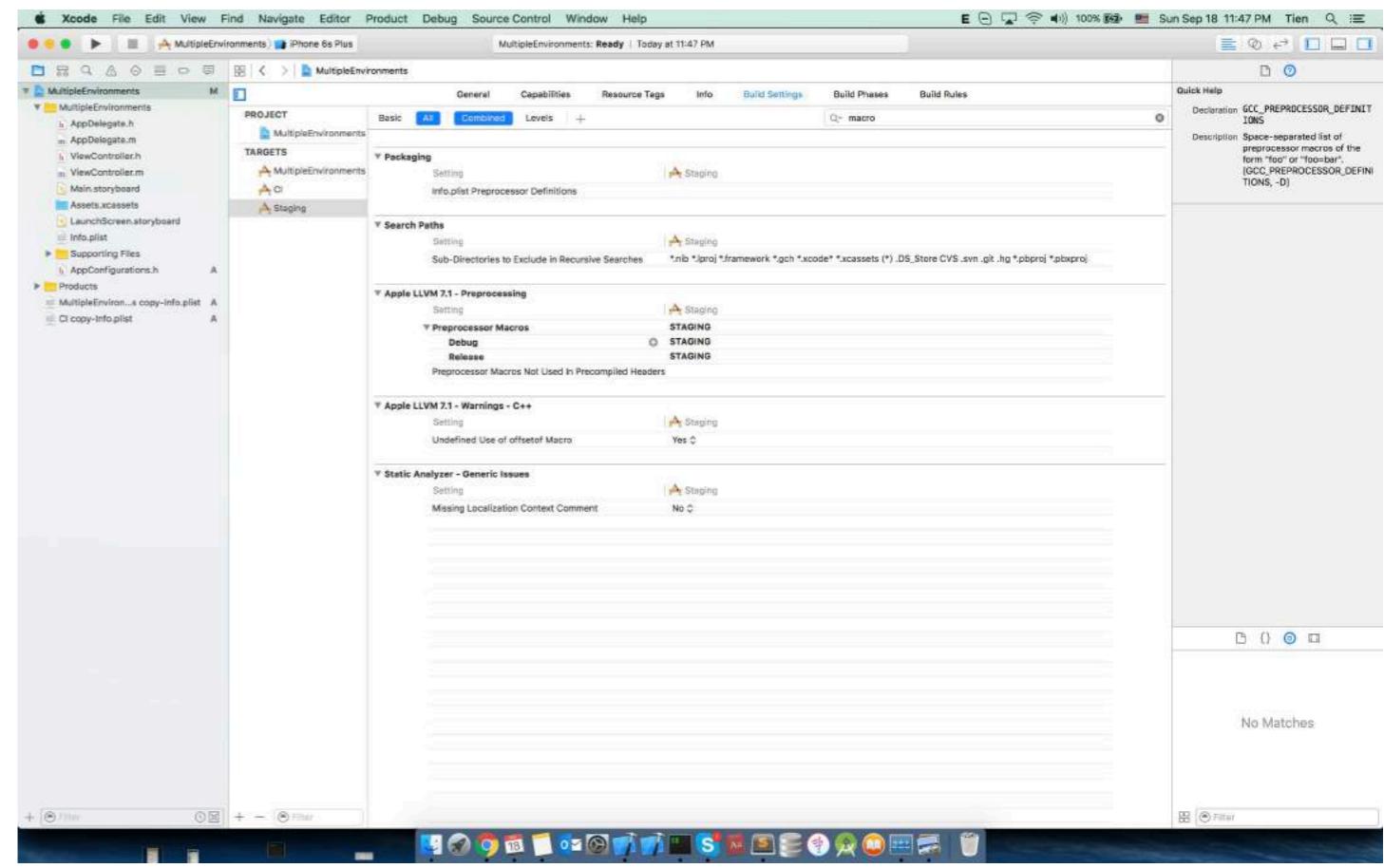
Target Staging:



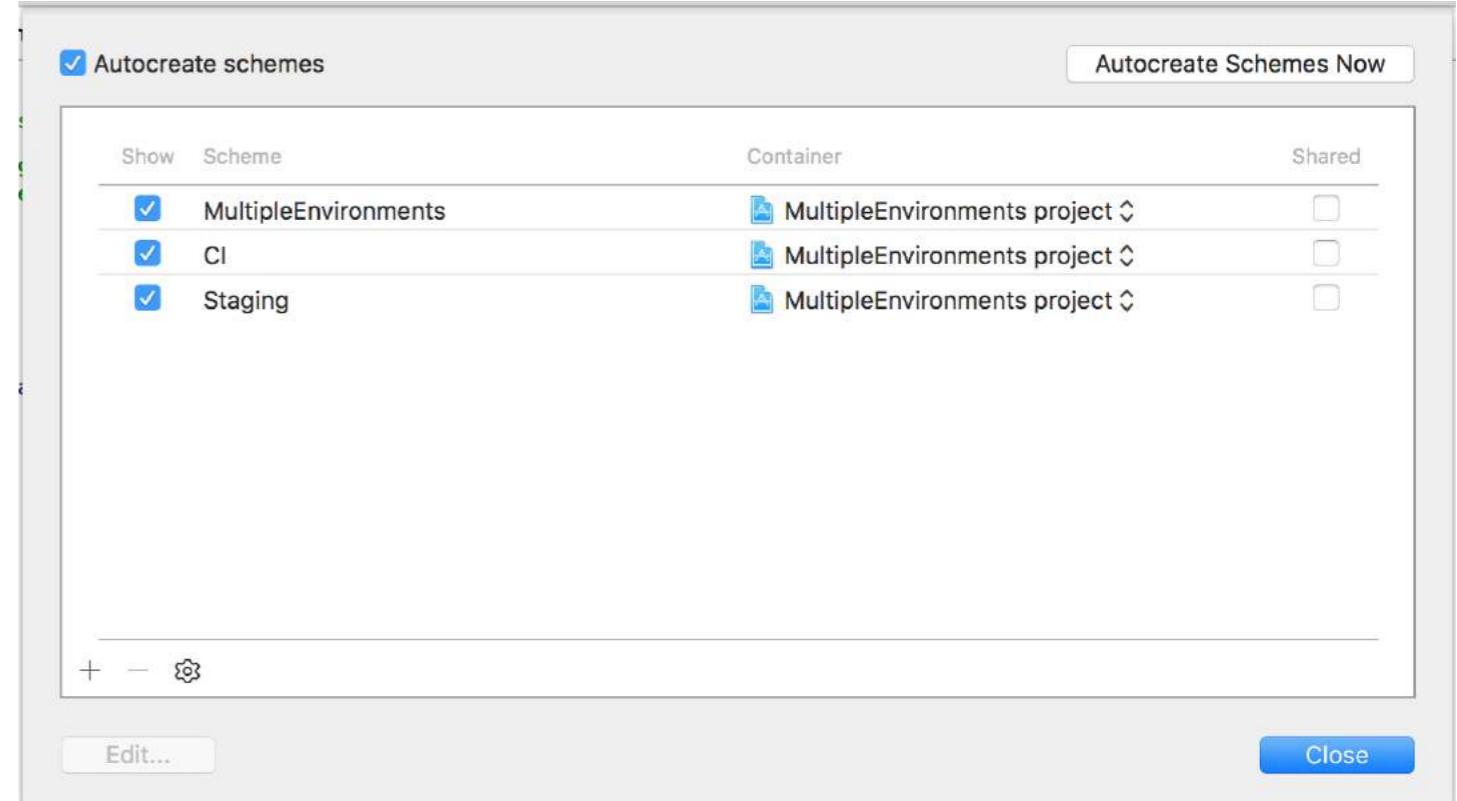
为每个目标创建方案：



我们将创建一个头文件来定义服务器 URL，如下所示：



Create scheme for each target:



We will create a header file to define SERVER URL as below:

```

// AppConfigurations.h
// MultipleEnvironments
// Created by Tien on 9/30/16.
// Copyright © 2016 tien. All rights reserved.

#ifndef AppConfigurations_h
#define AppConfigurations_h
#import <MultipleEnvironments/MultipleEnvironments.h>

// Define configurations for development environment.
#if defined(DEV)
#define SERVER_URL @"http://192.168.10.10:8080/"
#endif

// Define configurations for CI environment.
#if defined(CI)
#define SERVER_URL @"http://ci.api.example.com/"
#endif

// Define configurations for Staging environment.
#if defined(STAGING)
#define SERVER_URL @"http://stg.api.example.com/"
#endif

#endif // AppConfigurations_h

```

这意味着，

- 如果我们使用默认目标（MultipleEnvironment）进行运行/归档，SERVER_URL 是 <http://192.168.10.10:8080/>
- 如果我们使用 CI 目标运行/归档，SERVER_URL 是 <http://ci.api.example.com/>
- 如果我们使用 STAGING 目标运行/归档，SERVER_URL 是 <http://stg.api.example.com/>

如果你想做更多自定义，例如：为每个目标更改应用名称：

```

// AppConfigurations.h
// MultipleEnvironments
// Created by Tien on 9/30/16.
// Copyright © 2016 tien. All rights reserved.

#ifndef AppConfigurations_h
#define AppConfigurations_h
#import <MultipleEnvironments/MultipleEnvironments.h>

// Define configurations for development environment.
#if defined(DEV)
#define SERVER_URL @"http://192.168.10.10:8080/"
#endif

// Define configurations for CI environment.
#if defined(CI)
#define SERVER_URL @"http://ci.api.example.com/"
#endif

// Define configurations for Staging environment.
#if defined(STAGING)
#define SERVER_URL @"http://stg.api.example.com/"
#endif

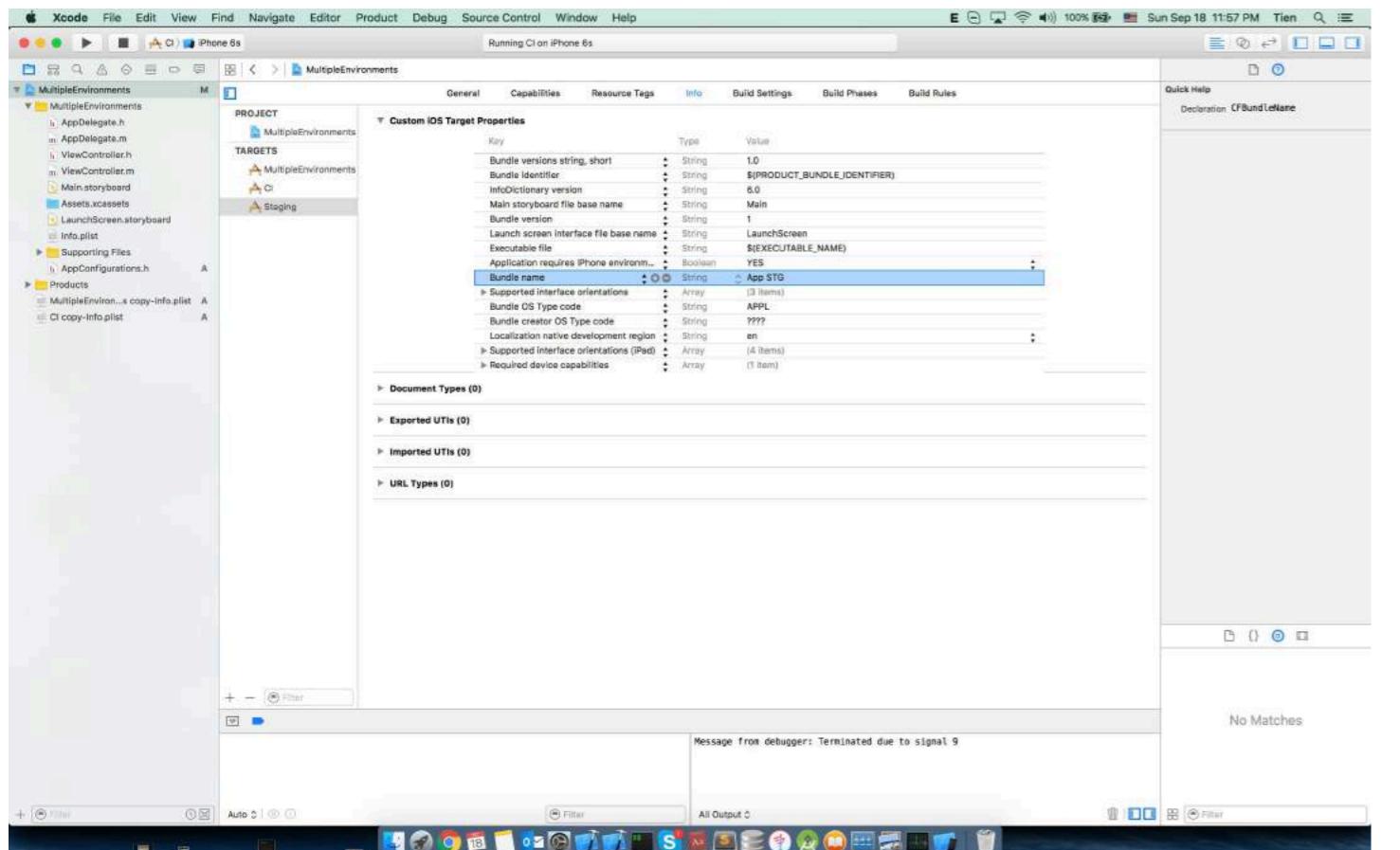
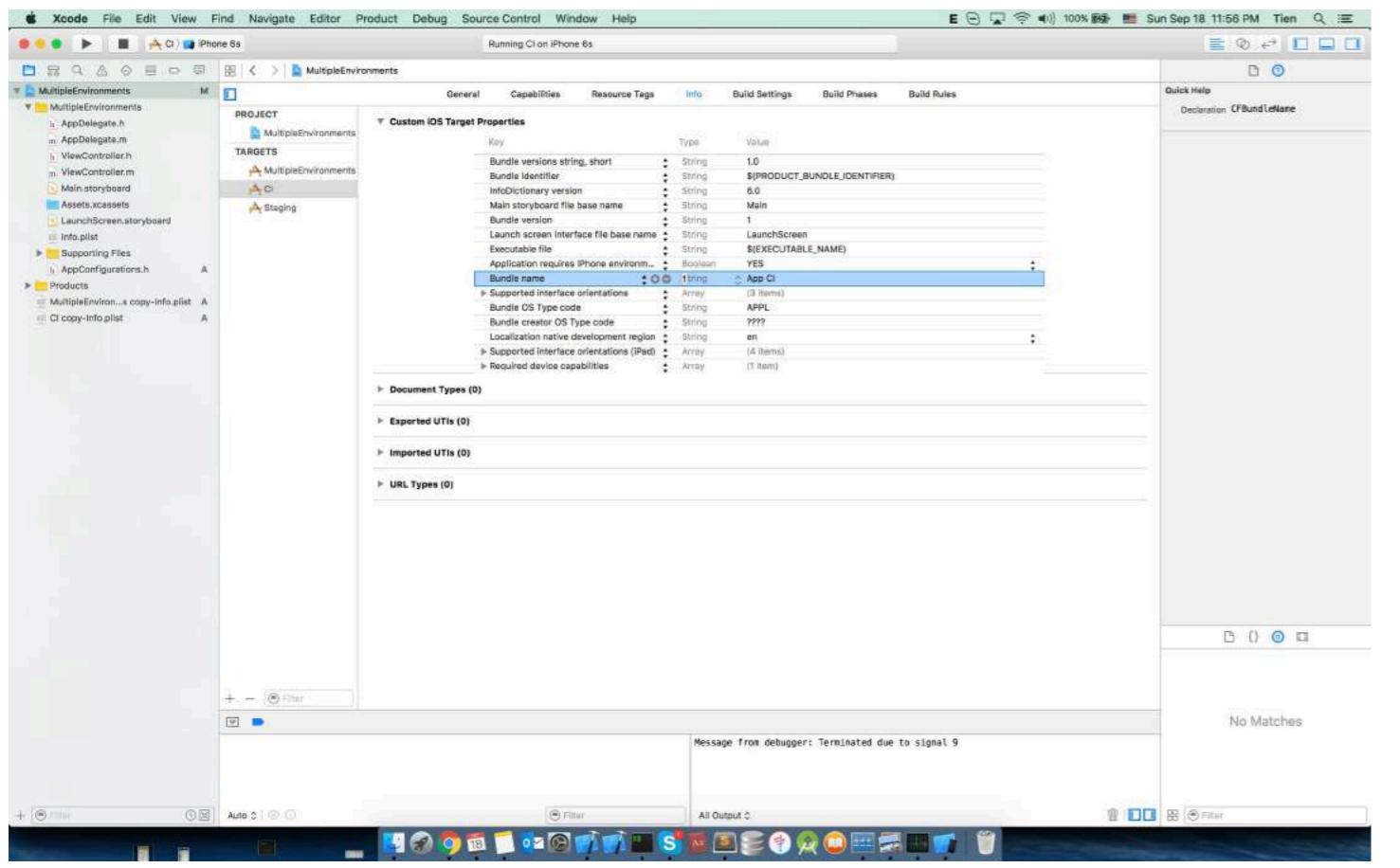
#endif // AppConfigurations_h

```

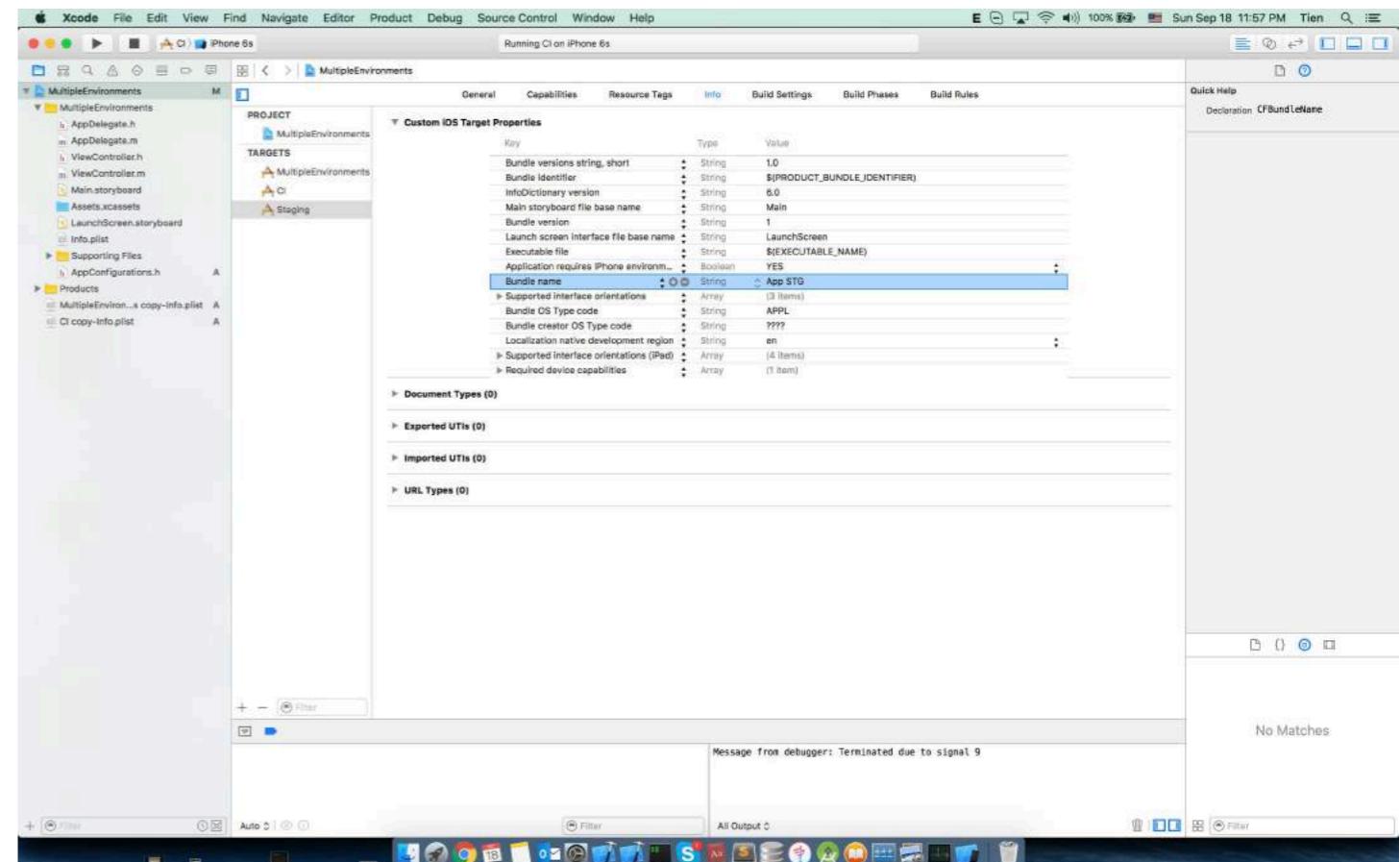
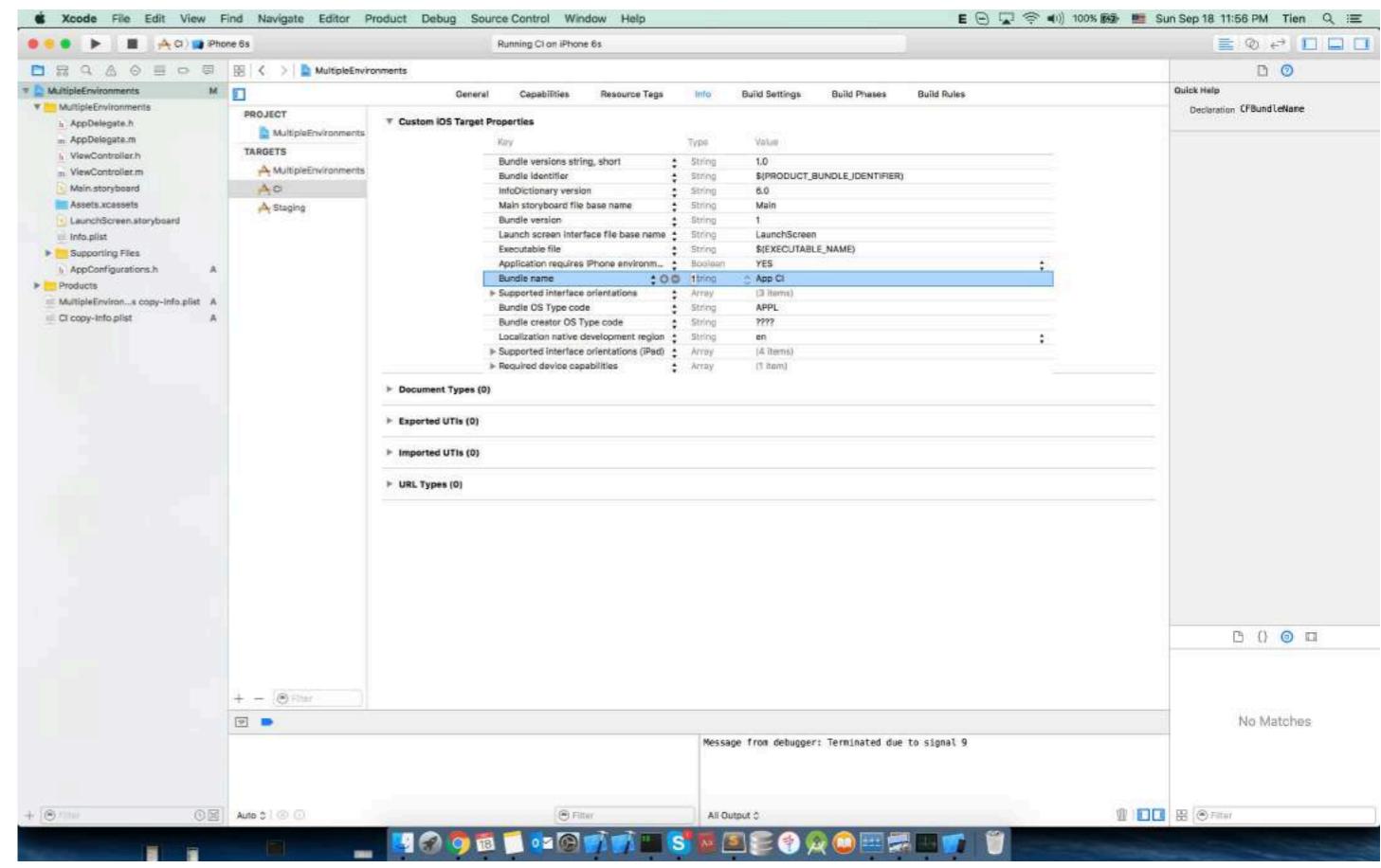
It means,

- If we run/archive using the default target (MultipleEnvironment), the SERVER_URL is <http://192.168.10.10:8080/>
- If we run/archive using CI target, the SERVER_URL is <http://ci.api.example.com/>
- If we run/archive using STAGING target, the SERVER_URL is <http://stg.api.example.com/>

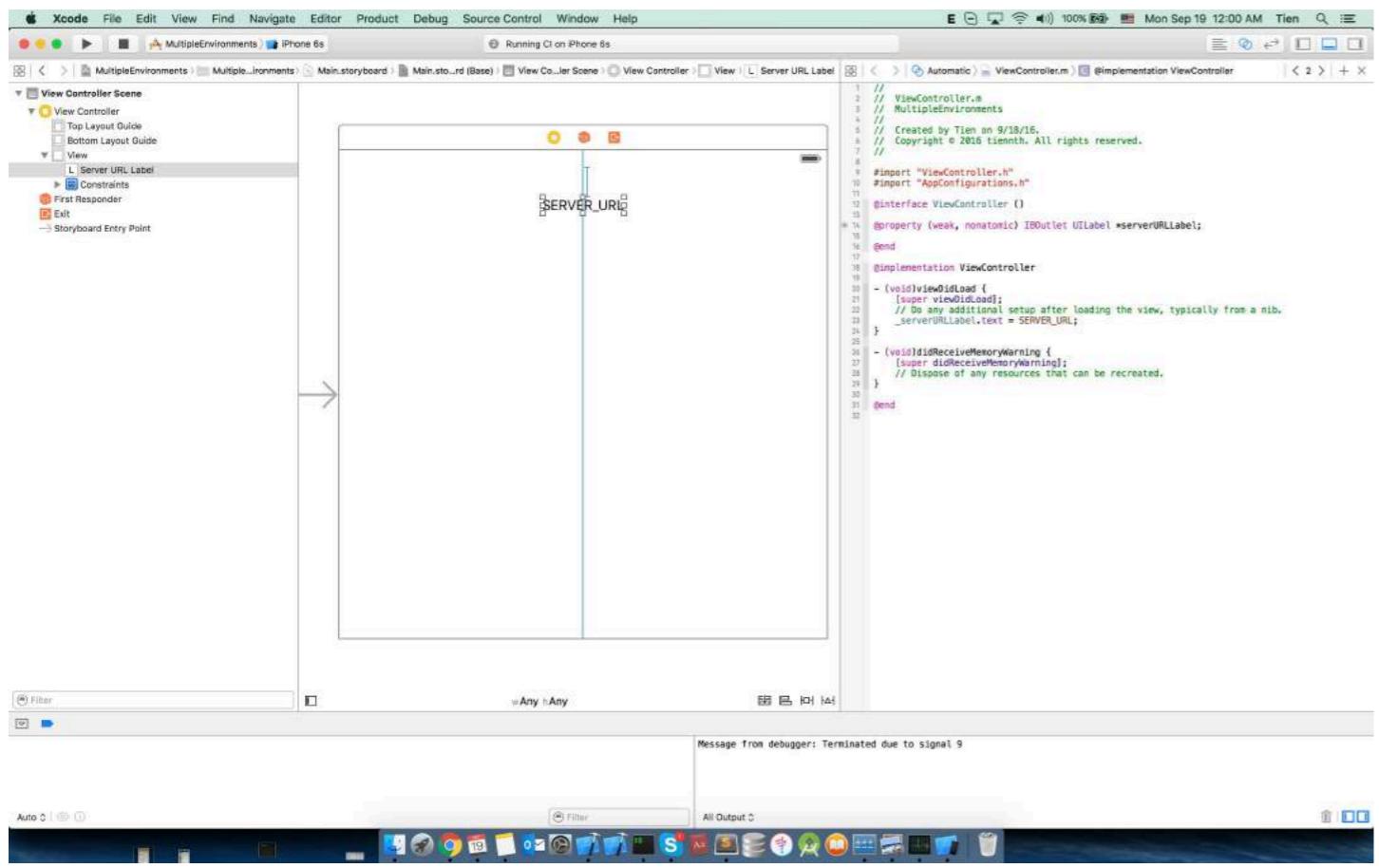
If you want to do more customize, for example: Change app name for each target:



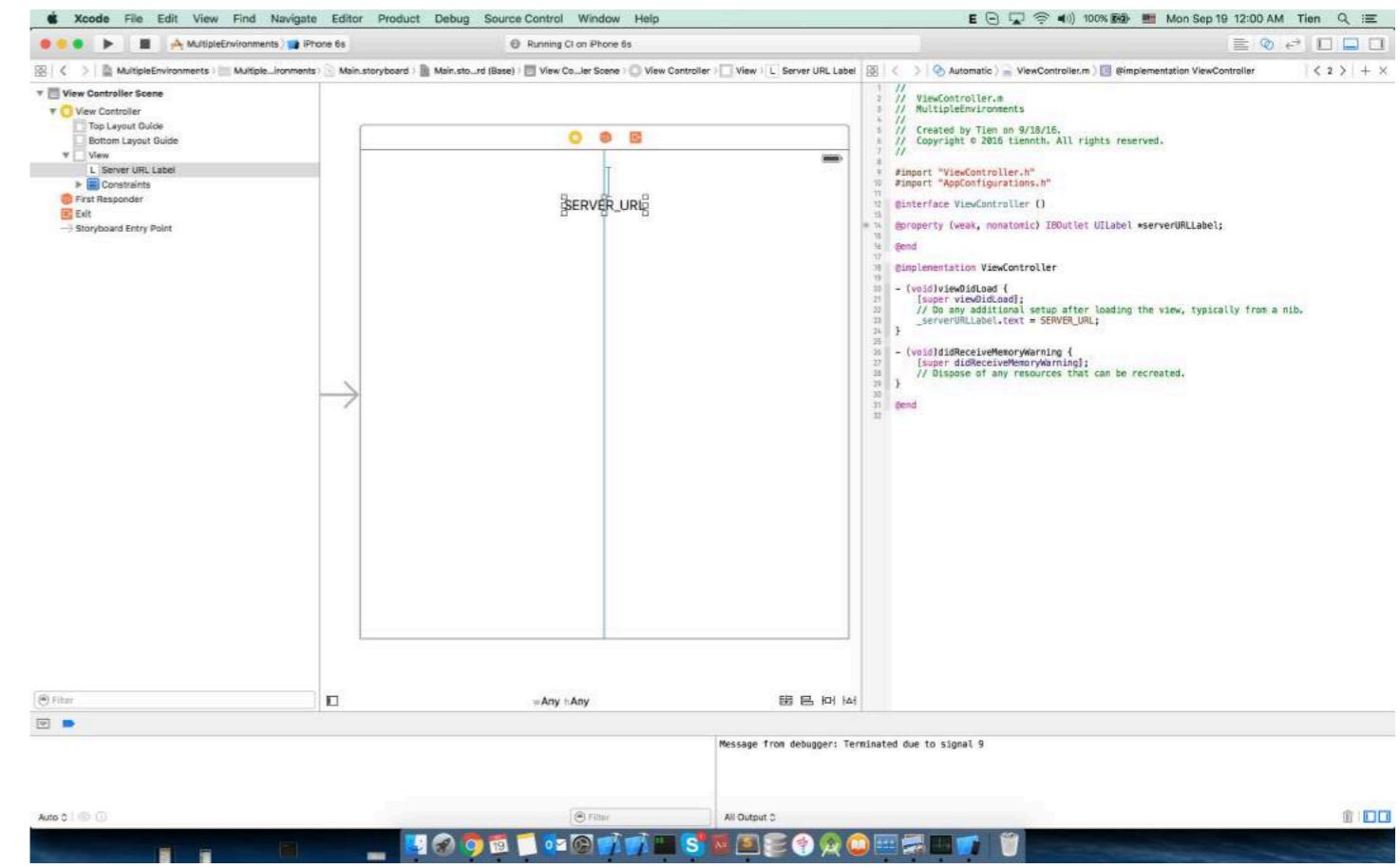
快完成了。现在我们想在主屏幕显示当前的 SERVER_URL：



Almost done. Now we want to show current SERVER_URL to main screen:



现在，让我们看看如果使用默认目标（MultipleEnvironment）运行应用会怎样

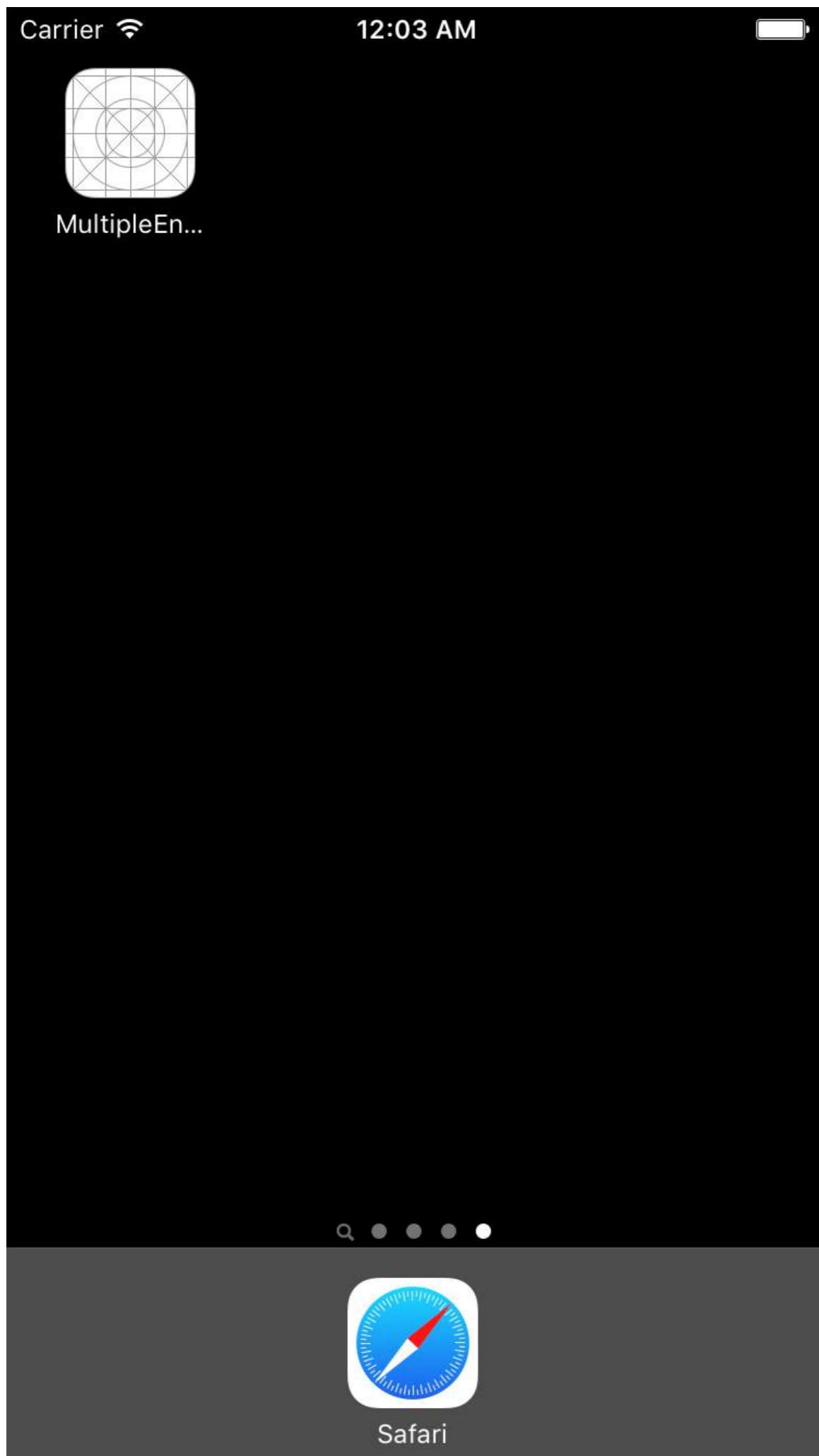


Now, let's see if we run the app with the default target (MultipleEnvironment)



<http://192.168.10.10:8080/>

<http://192.168.10.10:8080/>



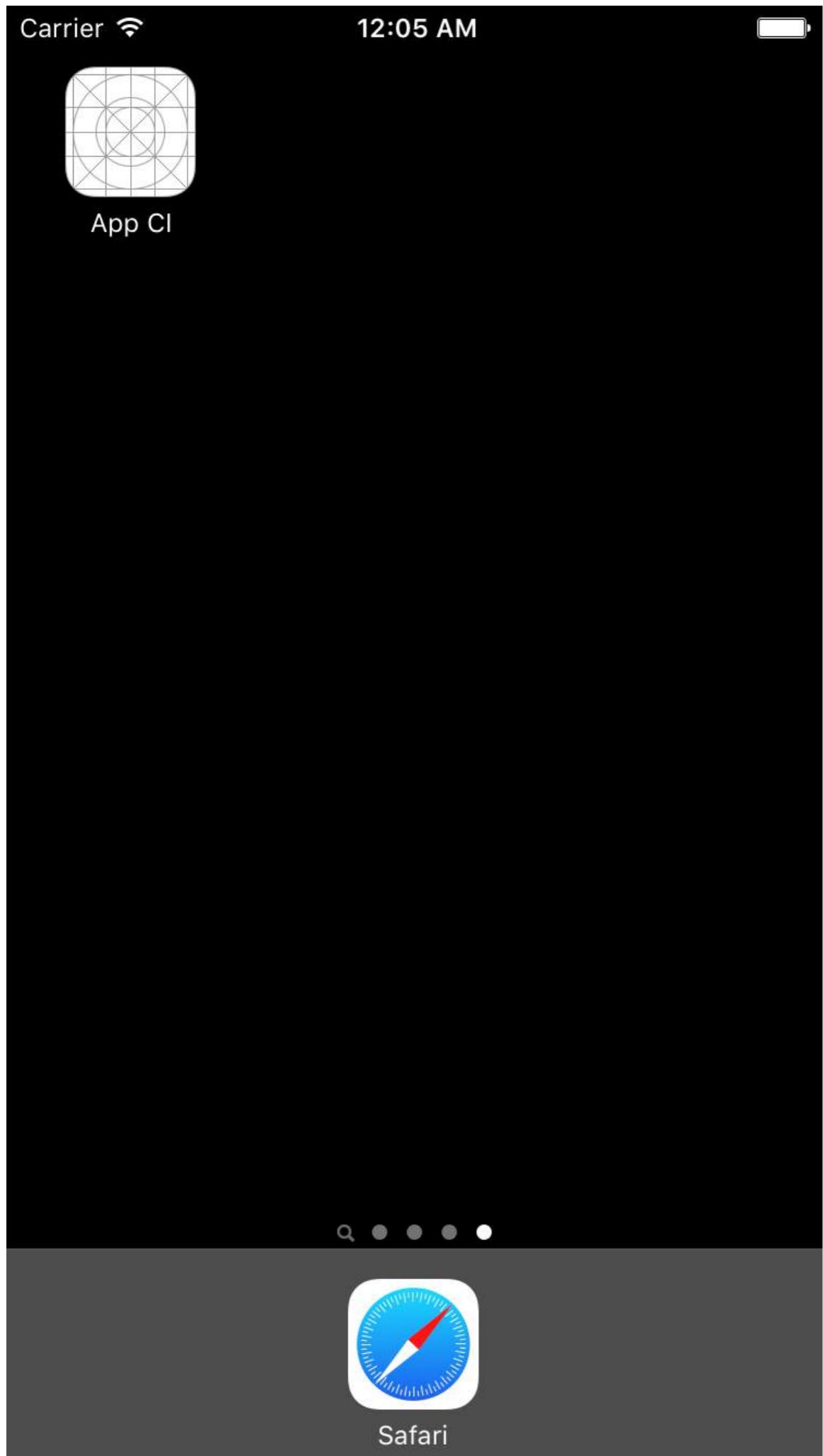
CI 目标：

CI target:



<http://ci.api.example.com/>

<http://ci.api.example.com/>

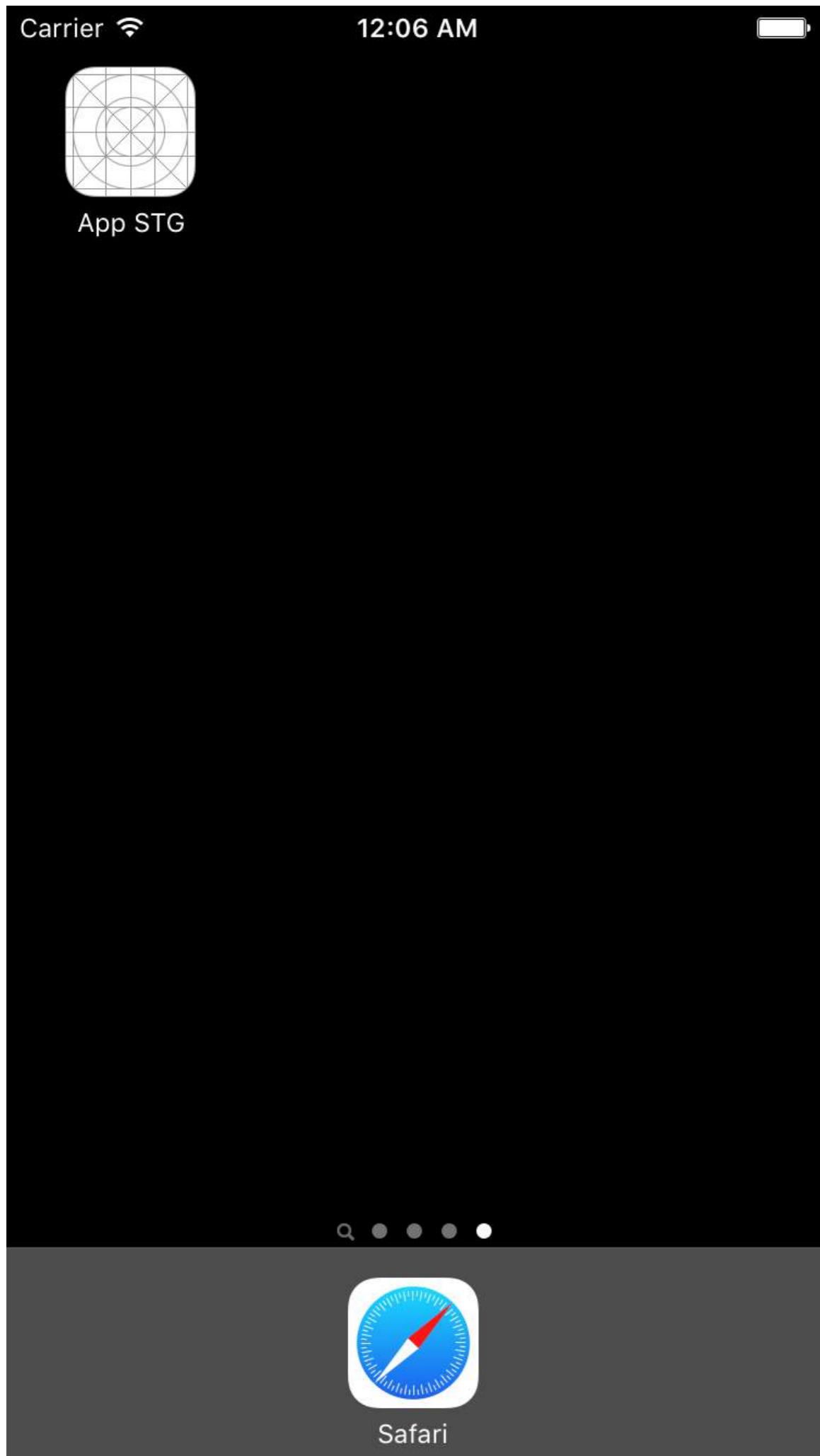


预发布目标:

Staging target:

<http://stg.api.example.com/>

<http://stg.api.example.com/>



如您所见，SERVER_URL 和应用名称的值会根据不同目标而变化 :)

As you can see, value of SERVER_URL and app name is changed for each target :)

第151章：设置视图背景

第151.1节：填充UIImageView的背景图像

Objective-C

```
UIGraphicsBeginImageContext(self.view.frame.size);
[[UIImage imageNamed:@"image.png"] drawInRect:self.view.bounds];
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
self.view.backgroundColor = [UIColor colorWithPatternImage:image];
```

第151.2节：创建渐变背景视图

要创建带有渐变的背景，可以使用CAGradientLayer类：

Swift 3.1：

```
func createGradient() {
    let caLayer = CAGradientLayer()
    caLayer.colors = [UIColor.white, UIColor.green, UIColor.blue]
    caLayer.locations = [0, 0.5, 1]
    caLayer.bounds = self.bounds
    self.layer.addSublayer(caLayer)
}
```

这可以在viewDidLoad()中这样调用：

```
override func viewDidLoad() {
    super.viewDidLoad()
    createGradient()
}
```

CAGradientLayer 的 locations 和 bounds 变量可以接受多个值，以创建包含任意多种颜色的渐变层。根据文档说明：

默认情况下，颜色均匀分布在图层上，但你也可以选择性地指定 locations，以控制渐变中颜色的位置。

第151.3节：使用图片设置视图背景

```
self.view.backgroundColor = [UIColor colorWithPatternImage:[UIImage imageNamed:@"Background.png"]];
```

第151.4节：设置视图背景

Objective C：

```
view.backgroundColor = [UIColor redColor];
```

Swift:

```
view.backgroundColor! = UIColor.redColor()
```

Chapter 151: Set View Background

Section 151.1: Fill background Image of a UIView

Objective-C

```
UIGraphicsBeginImageContext(self.view.frame.size);
[[UIImage imageNamed:@"image.png"] drawInRect:self.view.bounds];
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
self.view.backgroundColor = [UIColor colorWithPatternImage:image];
```

Section 151.2: Creating a gradient background view

To create a background with a gradient you can use the [CAGradientLayer](#) class:

Swift 3.1:

```
func createGradient() {
    let caLayer = CAGradientLayer()
    caLayer.colors = [UIColor.white, UIColor.green, UIColor.blue]
    caLayer.locations = [0, 0.5, 1]
    caLayer.bounds = self.bounds
    self.layer.addSublayer(caLayer)
}
```

This can be called on viewDidLoad() like so:

```
override func viewDidLoad() {
    super.viewDidLoad()
    createGradient()
}
```

The CAGradientLayer locations and bounds variables can take multiple values to create a gradient layer with however many colors you desire. From the documentation:

By default, the colors are spread uniformly across the layer, but you can optionally specify locations for control over the color positions through the gradient.

Section 151.3: Set View background with image

```
self.view.backgroundColor = [UIColor colorWithPatternImage:[UIImage imageNamed:@"Background.png"]];
```

Section 151.4: Set View background

Objective C:

```
view.backgroundColor = [UIColor redColor];
```

Swift:

```
view.backgroundColor! = UIColor.redColor()
```

```
view.backgroundColor = UIColor.redColor
```

```
view.backgroundColor = UIColor.redColor
```

第152章：块

第152.1节：自定义方法的自定义完成块

1- 定义你自己的自定义块

```
typedef void(^myCustomCompletion)(BOOL);
```

2- 创建一个自定义方法，该方法以你的自定义完成块作为参数。

```
-(void) customMethodName:(myCustomCompletion) compblock{
    // 执行操作
    // 检查完成块是否存在；如果不检查会抛出异常
    if(complblock)
        complblock(YES);
}
```

3- 如何在你的方法中使用块

```
[self customMethodName:^(BOOL finished) {
    if(finished){
        NSLog(@"success");
    }
}];
```

第152.2节：UIView动画

```
[UIView animateWithDuration:1.0
    animations:^{
        someView.alpha = 0;
        otherView.alpha = 1;
    }
    completion:^(BOOL finished) {
        [someView removeFromSuperview];
    }];
}
```

插入符号“^”字符定义了一个代码块。例如，`^{ ... }`就是一个代码块。更具体地说，它是一个返回“void”且不接受任何参数的代码块。它相当于一个类似于“`- (void)something;`”的方法，但代码块本身没有内在的名称。

定义一个可以接受参数的代码块的方式非常相似。要向代码块传递参数，你可以这样定义代码块：`^(BOOL someArg, NSString someStr) { ... }*`。当你使用支持代码块的API调用时，尤其是在动画代码块或NSURLConnection代码块中，如上例所示，你会编写类似这样的代码块。

第152.3节：修改捕获变量

Block 会捕获出现在同一词法作用域中的变量。通常这些变量会被捕获为“const”值：

```
int val = 10;
void (^blk)(void) = ^{
    val = 20; // 错误！val 是一个常量值，不能被修改！
};
```

Chapter 152: Block

Section 152.1: Custom completion block for Custom Methods

1- Define Your own custom Block

```
typedef void(^myCustomCompletion)(BOOL);
```

2- Create custom method which takes your custom completion block as a parameter.

```
-(void) customMethodName:(myCustomCompletion) compblock{
    //do stuff
    // check if completion block exist; if we do not check it will throw an exception
    if(complblock)
        complblock(YES);
}
```

3- How to use block in your Method

```
[self customMethodName:^(BOOL finished) {
    if(finished){
        NSLog(@"success");
    }
}];
```

Section 152.2: UIView Animations

```
[UIView animateWithDuration:1.0
    animations:^{
        someView.alpha = 0;
        otherView.alpha = 1;
    }
    completion:^(BOOL finished) {
        [someView removeFromSuperview];
    }];
}
```

The carat “^” character defines a block. For example, `^{ ... }` is a block. More specifically, it is a block that returns “void” and accepts no arguments. It is equivalent to a method such like: “`- (void)something;`” but there is no inherent name associated with the code block.

Define a block that can accept arguments work very similarly. To supply an argument to a block, you define the block like so: `^(BOOL someArg, NSString someStr) { ... }*`. When you use API calls that support blocks, you'll be writing blocks that look similar to this, especially for animation blocks or NSURLConnection blocks as shown in the above example.

Section 152.3: Modify captured variable

Block will capture variables that appeared in the same lexical scope. Normally these variables are captured as “const” value:

```
int val = 10;
void (^blk)(void) = ^{
    val = 20; // Error! val is a constant value and cannot be modified!
};
```

为了修改变量，需要使用 `_block` 存储类型修饰符。

```
_block int val = 10;
void (^blk)(void) = ^{
    val = 20; // 正确！val 现在可以像普通变量一样被修改。
};
```

In order to modify the variable, you need to use the `_block` storage type modifier.

```
_block int val = 10;
void (^blk)(void) = ^{
    val = 20; // Correct! val now can be modified as an ordinary variable.
};
```

第153章：自动布局中的内容拥抱/内容压缩

第153.1节：定义：固有内容尺寸

在自动布局出现之前，你总是需要告诉按钮和其他控件它们应该有多大，要么通过设置它们的frame或bounds属性，要么通过在界面构建器中调整它们的大小。但事实证明，大多数控件完全能够根据其内容确定它们需要多少空间。

一个标签知道它的宽度和高度，因为它知道设置在其上的文本长度，以及该文本的字体大小。按钮也是如此，按钮可能将文本与背景图像和一些内边距结合起来。

分段控件、进度条和大多数其他控件也是如此，尽管有些控件可能只有预定的高度，但宽度未知。

这被称为固有内容尺寸，这是自动布局中的一个重要概念。自动布局会询问你的控件它们需要多大空间，并基于这些信息来布局屏幕。

通常你会想使用固有内容尺寸，但在某些情况下你可能不想这样做。你可以通过在控件上设置明确的宽度或高度约束来防止这种情况。

想象一下，当你在UIImageView上设置一张比屏幕大得多的图片时会发生什么。除非你想让视图调整到图片的尺寸，否则通常你会给图片视图一个固定的宽度和高度，并缩放内容。

参考：<https://www.raywenderlich.com/115444/auto-layout-tutorial-in-ios-9-part-2-constraints>

Chapter 153: Content Hugging/Content Compression in Autolayout

Section 153.1: Definition: Intrinsic Content Size

Before Auto Layout, you always had to tell buttons and other controls how big they should be, either by setting their frame or bounds properties or by resizing them in Interface Builder. But it turns out that most controls are perfectly capable of determining how much space they need, based on their content.

A **label** knows how wide and tall it is because it knows the length of the text that has been set on it, as well as the font size for that text. Likewise for a **button**, which might combine the text with a background image and some padding.

The same is true for segmented controls, progress bars, and most other controls, although some may only have a predetermined height but an unknown width.

This is known as the intrinsic content size, and it is an important concept in Auto Layout. Auto Layout asks your controls how big they need to be and lays out the screen based on that information.

Usually you want to use the `intrinsic content size`, but there are some cases where you may not want to do that. You can prevent this by setting an explicit Width or Height constraint on a control.

Imagine what happens when you set an image on a UIImageView if that image is much larger than the screen. You usually want to give image views a fixed width and height and scale the content, unless you want the view to resize to the dimensions of the image.

Reference: <https://www.raywenderlich.com/115444/auto-layout-tutorial-in-ios-9-part-2-constraints>

第154章：iOS谷歌地点API

第154.1节：从当前位置获取附近地点

前提条件

1. 在项目中安装pods
2. 安装GooglePlaces SDK
3. 启用定位服务

首先，我们需要通过获取用户当前的经度和纬度来获取用户的位置。

1. 导入GooglePlaces和GooglePlacePicker

```
import GooglePlaces
import GooglePlacePicker
```

2. 添加CLLocationManagerDelegate协议

```
class ViewController: UIViewController, CLLocationManagerDelegate { }
```

3. 创建你的CLLocationManager()

```
var currentLocation = CLLocationManager()
```

4. 请求授权

```
currentLocation = CLLocationManager()
currentLocation.请求始终授权()
```

5. 创建一个按钮以调用 GooglePlacePicker 方法

```
@IBAction func placePickerAction(sender: AnyObject) {

    if CLLocationManager.authorizationStatus() == .AuthorizedAlways {

        let center = CLLocationCoordinate2DMake((currentLocation.location?.coordinate.latitude)!, (currentLocation.location?.coordinate.longitude)!)
        let northEast = CLLocationCoordinate2DMake(center.latitude + 0.001, center.longitude + 0.001)
        let southWest = CLLocationCoordinate2DMake(center.latitude - 0.001, center.longitude - 0.001)
        let viewport = GMSCoordinateBounds(coordinate: northEast, coordinate: southWest)
        let config = GMSPlacePickerConfig(viewport: viewport)
        placePicker = GMSPlacePicker(config: config)

        placePicker?.pickPlaceWithCallback({ (place: GMSPlace?, error: NSError?) -> Void in
            if let error = error {
                print("选择地点错误: \(error.localizedDescription)")
                return
            }

            if let place = place {
                print("地点名称: \(place.name)")
                print("地址: \(place.formattedAddress)")
            }
        })
    }
}
```

Chapter 154: iOS Google Places API

Section 154.1: Getting Nearby Places from Current Location

Prerequisites

1. Install pods in your project
2. Install the GooglePlaces SDK
3. Enable location services

First we need to get the users location by getting their current longitude and latitude.

1. Import GooglePlaces and GooglePlacePicker

```
import GooglePlaces
import GooglePlacePicker
```

2. Add the CLLocationManagerDelegate protocol

```
class ViewController: UIViewController, CLLocationManagerDelegate { }
```

3. create your CLLocationManager()

```
var currentLocation = CLLocationManager()
```

4. Request authorization

```
currentLocation = CLLocationManager()
currentLocation.requestAlwaysAuthorization()
```

5. Create a button to call the GooglePlacePicker method

```
@IBAction func placePickerAction(sender: AnyObject) {

    if CLLocationManager.authorizationStatus() == .AuthorizedAlways {

        let center = CLLocationCoordinate2DMake((currentLocation.location?.coordinate.latitude)!, (currentLocation.location?.coordinate.longitude)!)
        let northEast = CLLocationCoordinate2DMake(center.latitude + 0.001, center.longitude + 0.001)
        let southWest = CLLocationCoordinate2DMake(center.latitude - 0.001, center.longitude - 0.001)
        let viewport = GMSCoordinateBounds(coordinate: northEast, coordinate: southWest)
        let config = GMSPlacePickerConfig(viewport: viewport)
        placePicker = GMSPlacePicker(config: config)

        placePicker?.pickPlaceWithCallback({ (place: GMSPlace?, error: NSError?) -> Void in
            if let error = error {
                print("Pick Place error: \(error.localizedDescription)")
                return
            }

            if let place = place {
                print("Place name: \(place.name)")
                print("Address: \(place.formattedAddress)")
            }
        })
    }
}
```

```
    } else {
        print("地点名称: nil")
        print("地址: nil")
    }
}
```

```
    } else {
        print("Place name: nil")
        print("Address: nil")
    }
}
```

第155章：导航栏

第155.1节：SWIFT示例

```
navigationController?.navigationBar.titleTextAttributes = [NSForegroundColorAttributeName:  
UIColor.white, NSFontAttributeName:UIFont(name: "HelveticaNeue-CondensedBold", size: 17)!,  
navigationController?.navigationBar.tintColor = .white  
navigationController?.navigationBar.barTintColor = .red  
navigationController?.navigationBar.isTranslucent = false  
navigationController?.navigationBar.barStyle = .black
```

第155.2节：自定义默认导航栏外观

```
// 应用程序中默认的 UINavigationBar 外观  
[[UINavigationBar appearance] setTitleTextAttributes:@{NSForegroundColorAttributeName: [UIColor  
whiteColor],  
NSFontAttributeName : [UIFont  
fontWithName:@"HelveticaNeue-CondensedBold" size:17],  
}];  
  
[[UINavigationBar appearance] setTintColor:[UIColor whiteColor]];  
[[UINavigationBar appearance] setBarTintColor:[UIColor KNGRed]];  
[[UINavigationBar appearance] setTranslucent:NO];  
[[UINavigationBar appearance] setBarStyle: UIBarStyleBlack];  
[[UIBarButtonItem appearanceWhenContainedIn: [UISearchBar class], nil] setTintColor:[UIColor  
KNGGray]];
```

Chapter 155: Navigation Bar

Section 155.1: SWIFT Example

```
navigationController?.navigationBar.titleTextAttributes = [NSForegroundColorAttributeName:  
UIColor.white, NSFontAttributeName:UIFont(name: "HelveticaNeue-CondensedBold", size: 17)!,  
navigationController?.navigationBar.tintColor = .white  
navigationController?.navigationBar.barTintColor = .red  
navigationController?.navigationBar.isTranslucent = false  
navigationController?.navigationBar.barStyle = .black
```

Section 155.2: Customize default navigation bar appearance

```
// Default UINavigationBar appearance throughout the app  
[[UINavigationBar appearance] setTitleTextAttributes:@{NSForegroundColorAttributeName: [UIColor  
whiteColor],  
NSFontAttributeName : [UIFont  
fontWithName:@"HelveticaNeue-CondensedBold" size:17],  
}];  
  
[[UINavigationBar appearance] setTintColor:[UIColor whiteColor]];  
[[UINavigationBar appearance] setBarTintColor:[UIColor KNGRed]];  
[[UINavigationBar appearance] setTranslucent:NO];  
[[UINavigationBar appearance] setBarStyle: UIBarStyleBlack];  
[[UIBarButtonItem appearanceWhenContainedIn: [UISearchBar class], nil] setTintColor:[UIColor  
KNGGray]];
```

第156章：应用范围内的操作

第156.1节：获取最顶层的 UIViewController

获取最顶层的UIViewController的常用方法是获取活动

UIWindow的RootViewController。我写了一个扩展来实现这个功能：

```
extension UIApplication {  
  
    func topViewController(_ base: UIViewController? = UIApplication.shared.keyWindow?.rootViewController) -> UIViewController {  
  
        if let nav = base as? UINavigationController {  
            return topViewController(nav.visibleViewController)  
        }  
  
        if let tab = base as? UITabBarController {  
            if let selected = tab.selectedViewController {  
                return topViewController(selected)  
            }  
        }  
  
        if let presented = base?.presentedViewController {  
            return topViewController(presented)  
        }  
  
        return base!  
    }  
}
```

第156.2节：拦截系统事件

使用iOS的NotificationCenter，它非常强大，您可以拦截某些全应用范围的事件：

```
NotificationCenter.default.addObserver(  
    self,  
    selector: #selector(ViewController.do(_:)),  
    name: NSNotification.Name.UIApplicationDidBecomeActive,  
    object: nil)
```

您可以注册更多事件，详情请查看

<https://developer.apple.com/reference/foundation/nsnotification.name>。

Chapter 156: App wide operations

Section 156.1: Get the top most UIViewController

A common approach to get the top most UIViewController is to get the RootViewController of your active UIWindow. I wrote an extension for this:

```
extension UIApplication {  
  
    func topViewController(_ base: UIViewController? = UIApplication.shared.keyWindow?.rootViewController) -> UIViewController {  
  
        if let nav = base as? UINavigationController {  
            return topViewController(nav.visibleViewController)  
        }  
  
        if let tab = base as? UITabBarController {  
            if let selected = tab.selectedViewController {  
                return topViewController(selected)  
            }  
        }  
  
        if let presented = base?.presentedViewController {  
            return topViewController(presented)  
        }  
  
        return base!  
    }  
}
```

Section 156.2: Intercept System Events

Using the NotificationCenter of iOS, which can be very powerful, you are able to intercept certain app-wide events:

```
NotificationCenter.default.addObserver(  
    self,  
    selector: #selector(ViewController.do(_:)),  
    name: NSNotification.Name.UIApplicationDidBecomeActive,  
    object: nil)
```

You can register for a lot of more events, just take a look at

<https://developer.apple.com/reference/foundation/nsnotification.name>.

第157章：CoreImage滤镜

第157.1节：Core Image滤镜示例

Objective-C

只需记录此日志，查看如何使用特定滤镜

```
NSArray *properties = [CIFilter filterNamesInCategory:kCICategoryBuiltIn];  
  
for (NSString *filterName in properties)  
{  
    CIFilter *fltr = [CIFilter filterWithName:filterName];  
    NSLog(@"%@", [fltr attributes]);  
}
```

以CISepiaTone为例，系统日志如下

```
CIAtributeFilterDisplayName = "Sepia Tone";  
CIAtributeFilterName = CISepiaTone;  
CIAtributeReferenceDocumentation =  
"http://developer.apple.com/cgi-bin/apple_ref.cgi?apple_ref=/apple_ref/doc/filter/ci/CISepiaTone";  
inputImage = {  
    CIAtributeClass = CIImage;  
CIAtributeDescription = "用作输入图像的图像。对于也使用  
背景图像的滤镜，这是前景图像。";  
CIAtributeDisplayName = 图像；  
CIAtributeType = CIAtributeTypeImage ;  
};  
inputIntensity = {  
    CIAtributeClass = NSNumber ;  
CIAtributeDefault = 1 ;  
CIAtributeDescription = "棕褐色效果的强度。值为1.0时创建单色棕褐色图像。值为0.0时对图像无影响。";  
CIAtributeDisplayName = 强度；  
CIAtributeIdentity = 0 ;  
CIAtributeMin = 0 ;  
    CIAtributeSliderMax = 1 ;  
    CIAtributeSliderMin = 0 ;  
    CIAtributeType = CIAtributeTypeScalar ;  
};  
}
```

使用上述系统日志，我们按如下方式设置滤镜：

```
CIIImage *beginImage = [CIIImage imageWithCGImage:[myImageView.image CGImage]];  
CIContext *context = [CIContext contextWithOptions:nil];  
//选择滤镜名称和强度  
CIFilter *filter = [CIFilter filterWithName:@"CISepiaTone" keysAndValues: kCIInputImageKey,  
beginImage, @"inputIntensity", [NSNumber numberWithFloat:0.8], nil];  
CIIImage *outputImage = [filter outputImage];  
  
CGImageRef cgimg = [context createCGImage:outputImage fromRect:[outputImage extent]];  
UIImage *newImg = [UIImage imageWithCGImage:cgimg];  
  
[myImageView1 setImage:newImg];  
  
CGImageRelease(cgimg);
```

Chapter 157: CoreImage Filters

Section 157.1: Core Image Filter Example

Objective-C

Just log this see how to use a particular filter

```
NSArray *properties = [CIFilter filterNamesInCategory:kCICategoryBuiltIn];  
  
for (NSString *filterName in properties)  
{  
    CIFilter *fltr = [CIFilter filterWithName:filterName];  
    NSLog(@"%@", [fltr attributes]);  
}
```

In case of CISepiaTone the system log is as follows

```
CIAtributeFilterDisplayName = "Sepia Tone";  
CIAtributeFilterName = CISepiaTone;  
CIAtributeReferenceDocumentation =  
"http://developer.apple.com/cgi-bin/apple_ref.cgi?apple_ref=/apple_ref/doc/filter/ci/CISepiaTone";  
inputImage = {  
    CIAtributeClass = CIImage;  
CIAtributeDescription = "The image to use as an input image. For filters that also use a  
background image, this is the foreground image。";  
CIAtributeDisplayName = Image;  
CIAtributeType = CIAtributeTypeImage ;  
};  
inputIntensity = {  
    CIAtributeClass = NSNumber ;  
CIAtributeDefault = 1 ;  
CIAtributeDescription = "The intensity of the sepia effect. A value of 1.0 creates a  
monochrome sepia image. A value of 0.0 has no effect on the image。";  
CIAtributeDisplayName = Intensity;  
CIAtributeIdentity = 0 ;  
CIAtributeMin = 0 ;  
    CIAtributeSliderMax = 1 ;  
    CIAtributeSliderMin = 0 ;  
    CIAtributeType = CIAtributeTypeScalar ;  
};  
}
```

Using the above system log we set up the filter as below:

```
CIIImage *beginImage = [CIIImage imageWithCGImage:[myImageView.image CGImage]];  
CIContext *context = [CIContext contextWithOptions:nil];  
//select Filter Name and Intensity  
CIFilter *filter = [CIFilter filterWithName:@"CISepiaTone" keysAndValues: kCIInputImageKey,  
beginImage, @"inputIntensity", [NSNumber numberWithFloat:0.8], nil];  
CIIImage *outputImage = [filter outputImage];  
  
CGImageRef cgimg = [context createCGImage:outputImage fromRect:[outputImage extent]];  
UIImage *newImg = [UIImage imageWithCGImage:cgimg];  
  
[myImageView1 setImage:newImg];  
  
CGImageRelease(cgimg);
```

以上代码生成的图像



Image generated by the above code



设置滤镜的另一种方法

```
UIImageView *imageView1=[[UIImageView alloc]initWithFrame:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height/2)];
UIImageView *imageView2=[[UIImageView alloc]initWithFrame:CGRectMake(0,
self.view.frame.size.height/2, self.view.frame.size.width, self.view.frame.size.height/2)];
imageView1.image=[UIImage imageNamed:@"image.png"];

CIImage *beginImage = [CIImage imageWithCGImage:[imageView1.image CGImage]];
CIContext *context = [CIContext contextWithOptions:nil];
//选择滤镜名称和强度

CIFilter *filter = [CIFilter filterWithName:@"CIColorPosterize"];
[filter setValue:beginImage forKey:kCIInputImageKey];
[filter setValue:[NSNumber numberWithFloat:8.0] forKey:@"inputLevels"];
CIImage *outputImage = [filter outputImage];

CGImageRef cgimg = [context createCGImage:outputImage fromRect:[outputImage extent]];
UIImage *newImg = [UIImage imageWithCGImage:cgimg];

[imageView2 setImage:newImg];

CGImageRelease(cgimg);
[self.view addSubview:imageView1];
```

An alternative way to set up a filter

```
UIImageView *imageView1=[[UIImageView alloc]initWithFrame:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height/2)];
UIImageView *imageView2=[[UIImageView alloc]initWithFrame:CGRectMake(0,
self.view.frame.size.height/2, self.view.frame.size.width, self.view.frame.size.height/2)];
imageView1.image=[UIImage imageNamed:@"image.png"];

CIImage *beginImage = [CIImage imageWithCGImage:[imageView1.image CGImage]];
CIContext *context = [CIContext contextWithOptions:nil];
//select Filter Name and Intensity

CIFilter *filter = [CIFilter filterWithName:@"CIColorPosterize"];
[filter setValue:beginImage forKey:kCIInputImageKey];
[filter setValue:[NSNumber numberWithFloat:8.0] forKey:@"inputLevels"];
CIImage *outputImage = [filter outputImage];

CGImageRef cgimg = [context createCGImage:outputImage fromRect:[outputImage extent]];
UIImage *newImg = [UIImage imageWithCGImage:cgimg];

[imageView2 setImage:newImg];

CGImageRelease(cgimg);
[self.view addSubview:imageView1];
```

```
[self.view addSubview:imageView2];
```

由此代码生成的图像



所有可用滤镜如下

```
/* CIAccordionFoldTransition,
CIAdditionCompositing,
CIAffineClamp,
CIAffineTile,
CIAffineTransform,
CIAreaAverage,
CIAreaHistogram,
CIAreaMaximum,
CIAreaMaximumAlpha,
CIAreaMinimum,
CIAreaMinimumAlpha,
CIAztecCodeGenerator,
CIBarsSwipeTransition,
CIBlendWithAlphaMask,
CIBlendWithMask,
CIBloom,
CIBoxBlur,
CIBumpDistortion,
CIBumpDistortionLinear,
CICheckerboardGenerator,
CICircleSplashDistortion,
CICircularScreen,
CICircularWrap,
CICMYKHalftone,
CICode128BarcodeGenerator,
```

```
[self.view addSubview:imageView2];
```

Image generated from this code



All Available Filters are as below

```
/* CIAccordionFoldTransition,
CIAdditionCompositing,
CIAffineClamp,
CIAffineTile,
CIAffineTransform,
CIAreaAverage,
CIAreaHistogram,
CIAreaMaximum,
CIAreaMaximumAlpha,
CIAreaMinimum,
CIAreaMinimumAlpha,
CIAztecCodeGenerator,
CIBarsSwipeTransition,
CIBlendWithAlphaMask,
CIBlendWithMask,
CIBloom,
CIBoxBlur,
CIBumpDistortion,
CIBumpDistortionLinear,
CICheckerboardGenerator,
CICircleSplashDistortion,
CICircularScreen,
CICircularWrap,
CICMYKHalftone,
CICode128BarcodeGenerator,
```

CIColorBlendMode,
CIColorBurnBlendMode,
CIColorClamp,
CIColorControls,
CIColorCrossPolynomial,
CIColorCube,
CIColorCubeWithColorSpace,
CIColorDodgeBlendMode,
CIColorInvert,
CIColorMap,
CIColorMatrix,
CIColorMonochrome,
CIColorPolynomial,
CIColorPosterize,
CIColumnAverage,
CIComicEffect,
CIConstantColorGenerator,
CIConvolution3X3,
CIConvolution5X5,
CIConvolution7X7,
CIConvolution9Horizontal,
CIConvolution9Vertical,
CICopyMachineTransition,
CICrop,
CICrystallize (晶体化) ,
CIDarkenBlendMode (变暗混合模式) ,
CIDepthOfField (景深) ,
CIDifferenceBlendMode (差值混合模式) ,
CIDiscBlur (圆盘模糊) ,
CIDisintegrateWithMaskTransition (带蒙版过渡的瓦解) ,
CIDisplacementDistortion (位移扭曲) ,
CIDissolveTransition (溶解过渡) ,
CIDivideBlendMode (除法混合模式) ,
CIDotScreen (点状网屏) ,
CIDroste (德罗斯特效应) ,
CIEdges (边缘) ,
CIEdgeWork (边缘处理) ,
CIEightfoldReflectedTile (八重反射平铺) ,
CIEclusionBlendMode (排除混合模式) ,
CIEposureAdjust (曝光调整) ,
CIFalseColor (伪彩色) ,
CIFlashTransition (闪光过渡) ,
CIFourfoldReflectedTile (四重反射平铺) ,
CIFourfoldRotatedTile (四重旋转平铺) ,
CIFourfoldTranslatedTile (四重平移平铺) ,
CIGammaAdjust (伽马调整) ,
CIGaussianBlur (高斯模糊) ,
CIGaussianGradient (高斯渐变) ,
CIGlassDistortion (玻璃扭曲) ,
CIGlassLozenge (玻璃菱形) ,
CIGlideReflectedTile (滑动反射平铺) ,
CIGloom (阴影) ,
CIHardLightBlendMode (强光混合模式) ,
CIHatchedScreen (阴影线网) ,
CIHeightFieldFromMask (从蒙版生成高度场) ,
CIHexagonalPixelate (六边形像素化) ,
CIHighlightShadowAdjust (高光阴影调整) ,
CIHistogramDisplayFilter (直方图显示滤镜) ,
CIHoleDistortion (孔洞扭曲) ,
CIHueAdjust (色相调整) ,
CIHueBlendMode (色相混合模式) ,
CIKaleidoscope (万花筒) ,

CIColorBlendMode,
CIColorBurnBlendMode,
CIColorClamp,
CIColorControls,
CIColorCrossPolynomial,
CIColorCube,
CIColorCubeWithColorSpace,
CIColorDodgeBlendMode,
CIColorInvert,
CIColorMap,
CIColorMatrix,
CIColorMonochrome,
CIColorPolynomial,
CIColorPosterize,
CIColumnAverage,
CIComicEffect,
CIConstantColorGenerator,
CIConvolution3X3,
CIConvolution5X5,
CIConvolution7X7,
CIConvolution9Horizontal,
CIConvolution9Vertical,
CICopyMachineTransition,
CICrop,
CICrystallize,
CIDarkenBlendMode,
CIDepthOfField,
CIDifferenceBlendMode,
CIDiscBlur,
CIDisintegrateWithMaskTransition,
CIDisplacementDistortion,
CIDissolveTransition,
CIDivideBlendMode,
CIDotScreen,
CIDroste,
CIEdges,
CIEdgeWork,
CIEightfoldReflectedTile,
CIEclusionBlendMode,
CIEposureAdjust,
CIFalseColor,
CIFlashTransition,
CIFourfoldReflectedTile,
CIFourfoldRotatedTile,
CIFourfoldTranslatedTile,
CIGammaAdjust,
CIGaussianBlur,
CIGaussianGradient,
CIGlassDistortion,
CIGlassLozenge,
CIGlideReflectedTile,
CIGloom,
CIHardLightBlendMode,
CIHatchedScreen,
CIHeightFieldFromMask,
CIHexagonalPixelate,
CIHighlightShadowAdjust,
CIHistogramDisplayFilter,
CIHoleDistortion,
CIHueAdjust,
CIHueBlendMode,
CIKaleidoscope,

CI Lanczos Scale Transform (兰氏缩放变换),
CI Lenticular Halo Generator (透镜光晕生成器),
CI Light Blend Mode (变亮混合模式),
CI Light Tunnel (光隧道),
CI Linear Burn Blend Mode (线性加深混合模式),
CI Linear Dodge Blend Mode (线性减淡混合模式),
CI Linear Gradient (线性渐变),
CI Linear To sRGB Tone Curve (线性到sRGB色调曲线),
CI Line Overlay (线条叠加),
CI Line Screen (线条网点),
CI Luminosity Blend Mode (亮度混合模式),
CI Masked Variable Blur (带蒙版的可变模糊),
CI Mask To Alpha (蒙版转Alpha),
CI Maximum Component (最大分量),
CI Maximum Compositing (最大合成),
CI Median Filter (中值滤波),
CI Minimum Component (最小分量),
CI Minimum Compositing (最小合成),
CI Mod Transition (模运算过渡),
CI Motion Blur (运动模糊),
CI Multiply Blend Mode (正片叠底混合模式),
CI Multiply Compositing (正片叠底合成),
CI Noise Reduction (降噪),
CI Op Tile (操作平铺),
CI Overlay Blend Mode (叠加混合模式),
CI Page Curl Transition (翻页过渡),
CI Page Curl With Shadow Transition (带阴影的翻页过渡),
CI Parallelogram Tile (平行四边形平铺),
CIPDF417条码生成器,
CI Perspective 校正,
CI Perspective 平铺,
CI Perspective Transform,
CI Perspective Transform With Extent,
CI Photo Effect Chrome,
CI Photo Effect Fade,
CI Photo Effect Instant,
CI Photo Effect Mono,
CI Photo Effect Noir,
CI Photo Effect Process,
CI Photo Effect Tonal,
CI Photo Effect Transfer,
CI Pinch Distortion,
CI Pin Light Blend Mode,
CI Pixellate,
CI Pointillize,
CI QR Code Generator,
CI Radial Gradient,
CI Random Generator,
CI Ripple Transition,
CI Row Average,
CI Saturation Blend Mode,
CI Screen Blend Mode,
CI Sepia Tone,
CI Shaded Material,
CI Sharpen Luminance,
CI Sixfold Reflected Tile,
CI Sixfold Rotated Tile,
CI Smooth Linear Gradient,
CI Soft Light Blend Mode,
CI Source Atop Compositing,
CI Source In Compositing,
CI Source Out Compositing,

CI Lanczos Scale Transform,
CI Lenticular Halo Generator,
CI Light Blend Mode,
CI Light Tunnel,
CI Linear Burn Blend Mode,
CI Linear Dodge Blend Mode,
CI Linear Gradient,
CI Linear To sRGB Tone Curve,
CI Line Overlay,
CI Line Screen,
CI Luminosity Blend Mode,
CI Masked Variable Blur,
CI Mask To Alpha,
CI Maximum Component,
CI Maximum Compositing,
CI Median Filter,
CI Minimum Component,
CI Minimum Compositing,
CI Mod Transition,
CI Motion Blur,
CI Multiply Blend Mode,
CI Multiply Compositing,
CI Noise Reduction,
CI Op Tile,
CI Overlay Blend Mode,
CI Page Curl Transition,
CI Page Curl With Shadow Transition,
CI Parallelogram Tile,
CIPDF417 Barcode Generator,
CI Perspective Correction,
CI Perspective Tile,
CI Perspective Transform,
CI Perspective Transform With Extent,
CI Photo Effect Chrome,
CI Photo Effect Fade,
CI Photo Effect Instant,
CI Photo Effect Mono,
CI Photo Effect Noir,
CI Photo Effect Process,
CI Photo Effect Tonal,
CI Photo Effect Transfer,
CI Pinch Distortion,
CI Pin Light Blend Mode,
CI Pixellate,
CI Pointillize,
CI QR Code Generator,
CI Radial Gradient,
CI Random Generator,
CI Ripple Transition,
CI Row Average,
CI Saturation Blend Mode,
CI Screen Blend Mode,
CI Sepia Tone,
CI Shaded Material,
CI Sharpen Luminance,
CI Sixfold Reflected Tile,
CI Sixfold Rotated Tile,
CI Smooth Linear Gradient,
CI Soft Light Blend Mode,
CI Source Atop Compositing,
CI Source In Compositing,
CI Source Out Compositing,

```
CISourceOverCompositing,  
CISpotColor,  
CISpotLight,  
CISRGRToneCurveToLinear,  
CIStarShineGenerator,  
CIStraightenFilter,  
CISretchCrop,  
CISripesGenerator,  
CISubtractBlendMode,  
CISunbeamsGenerator,  
CISwipeTransition,  
CITemperatureAndTint,  
CIToneCurve,  
CITorusLensDistortion,  
CITriangleKaleidoscope,  
CITriangleTile,  
CITwelvefoldReflectedTile,  
CITwirlDistortion,  
CIUnsharpMask,  
CIVibrance,  
CIVignette,  
CIVignetteEffect,  
CIVortexDistortion,  
CIWhitePointAdjust,  
CIZoomBlur*/
```

```
CISourceOverCompositing,  
CISpotColor,  
CISpotLight,  
CISRGRToneCurveToLinear,  
CIStarShineGenerator,  
CIStraightenFilter,  
CISretchCrop,  
CISripesGenerator,  
CISubtractBlendMode,  
CISunbeamsGenerator,  
CISwipeTransition,  
CITemperatureAndTint,  
CIToneCurve,  
CITorusLensDistortion,  
CITriangleKaleidoscope,  
CITriangleTile,  
CITwelvefoldReflectedTile,  
CITwirlDistortion,  
CIUnsharpMask,  
CIVibrance,  
CIVignette,  
CIVignetteEffect,  
CIVortexDistortion,  
CIWhitePointAdjust,  
CIZoomBlur*/
```

第158章：使用CoreImage/OpenCV进行人脸检测

第158.1节：人脸与特征检测

Objective-C

导入以下内容到你的视图控制器

```
#import <CoreImage/CoreImage.h>
#import <CoreImage/CoreImage.h>
#import <QuartzCore/QuartzCore.h>
```

调用函数

```
[self faceDetector];
```

函数定义：

```
-(void)faceDetector
{
    // 加载用于人脸检测的图片
    UIImageView* image = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"download.jpeg"]];

    // 绘制人脸检测图片
    [self.view addSubview:image];

    // 在后台执行用于标记人脸的方法
    [self performSelectorInBackground:@selector(markFaces:) withObject:image];

    // 在y轴翻转图片以匹配Core Image使用的坐标系
    [image setTransform:CGAffineTransformMakeScale(1, -1)];

    // 翻转整个窗口使所有内容正立
    [self.view setTransform:CGAffineTransformMakeScale(1, -1)];
}
```

标记人脸函数

```
// 添加人脸方框及眼睛和嘴巴的颜色遮罩
-(void)markFaces:(UIImageView *)facePicture
{
    // 使用之前加载的人脸检测图片绘制CI图像
    CIImage* image = [CIImage imageWithCGImage:facePicture.image.CGImage];

    // 创建人脸检测器 - 由于速度不是问题，我们将使用高精度
    // 检测器
    CIDetector* detector = [CIDetector detectorOfType:CIDetectorTypeFace
                                                context:nil options:[NSDictionary
dictionaryWithObject:CIDetectorAccuracyHigh forKey:CIDetectorAccuracy]];

    // 创建一个数组，包含检测器检测到的所有人脸
    NSArray* features = [detector featuresInImage:image];
    NSLog(@"人脸数量 %d",[features count]);
}
```

Chapter 158: Face Detection Using CoreImage/OpenCV

Section 158.1: Face and Feature Detection

Objective-C

Import the following to your ViewController

```
#import <CoreImage/CoreImage.h>
#import <CoreImage/CoreImage.h>
#import <QuartzCore/QuartzCore.h>
```

Call the function

```
[self faceDetector];
```

Function definition:

```
-(void)faceDetector
{
    // Load the picture for face detection
    UIImageView* image = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"download.jpeg"]];

    // Draw the face detection image
    [self.view addSubview:image];

    // Execute the method used to markFaces in background
    [self performSelectorInBackground:@selector(markFaces:) withObject:image];

    // flip image on y-axis to match coordinate system used by core image
    [image setTransform:CGAffineTransformMakeScale(1, -1)];

    // flip the entire window to make everything right side up
    [self.view setTransform:CGAffineTransformMakeScale(1, -1)];
}

}
```

Mark Face Function

```
// Adds face squares and color masks to eyes and mouth
-(void)markFaces:(UIImageView *)facePicture
{
    // draw a CI image with the previously loaded face detection picture
    CIImage* image = [CIImage imageWithCGImage:facePicture.image.CGImage];

    // create a face detector - since speed is not an issue we'll use a high accuracy
    // detector
    CIDetector* detector = [CIDetector detectorOfType:CIDetectorTypeFace
                                                context:nil options:[NSDictionary
dictionaryWithObject:CIDetectorAccuracyHigh forKey:CIDetectorAccuracy]];

    // create an array containing all the detected faces from the detector
    NSArray* features = [detector featuresInImage:image];
    NSLog(@"Number of faces %d",[features count]);
}
```

```

// 我们将遍历每一个检测到的人脸。CIFaceFeature为我们提供了// 整张脸的宽度，以及每只眼睛和嘴巴的坐标（如果检测到的话）。// 还提供了眼睛和嘴巴的BOOL值，以便我们检查它们是否存在。
//   for (features in image)
//   {
for(CIFaceFeature* faceFeature in features)
{
    // 获取人脸的宽度
    CGFloat faceWidth = faceFeature.bounds.size.width;

    // 使用人脸的边界创建一个UIView
    UIView* faceView = [[UIView alloc] initWithFrame:faceFeature.bounds];

    // 在新创建的UIView周围添加边框
    faceView.layer.borderWidth = 1;
    faceView.layer.borderColor = [[UIColor redColor] CGColor];

    // 将新视图添加到视图中以在脸部周围创建一个框
    [self.view addSubview:faceView];

    if(faceFeature.hasLeftEyePosition)
    {
        // 根据脸部宽度创建一个UIView
        UIView* leftEyeView = [[UIView alloc]
initWithFrame:CGRectMake(faceFeature.leftEyePosition.x-faceWidth*0.15,
faceFeature.leftEyePosition.y-faceWidth*0.15, faceWidth*0.3, faceWidth*0.3)];
        // 更改眼睛视图的背景颜色
        [leftEyeView setBackgroundColor:[[UIColor blueColor] colorWithAlphaComponent:0.3]];
        // 根据脸部位置设置leftEyeView的位置
        [leftEyeView setCenter:faceFeature.leftEyePosition];
        // 圆角处理
        leftEyeView.layer.cornerRadius = faceWidth*0.15;
        // 将视图添加到窗口
        [self.view addSubview:leftEyeView];
    }

    if(faceFeature.hasRightEyePosition)
    {
        // 根据脸部宽度创建一个UIView
        UIView* rightEye = [[UIView alloc]
initWithFrame:CGRectMake(faceFeature.rightEyePosition.x-faceWidth*0.15,
faceFeature.rightEyePosition.y-faceWidth*0.15, faceWidth*0.3, faceWidth*0.3)];
        // 更改眼睛视图的背景颜色
        [rightEye setBackgroundColor:[[UIColor blueColor] colorWithAlphaComponent:0.3]];
        // 根据脸部设置 rightEyeView 的位置
        [rightEye setCenter:faceFeature.rightEyePosition];
        // 圆角处理
        rightEye.layer.cornerRadius = faceWidth*0.15;
        // 将新视图添加到窗口
        [self.view addSubview:rightEye];
    }

    if(faceFeature.hasMouthPosition)
    {
        // 创建一个 UIView，大小基于脸宽
        UIView* mouth = [[UIView alloc] initWithFrame:CGRectMake(faceFeature.mouthPosition.x-
faceWidth*0.2, faceFeature.mouthPosition.y-faceWidth*0.2, faceWidth*0.4, faceWidth*0.4)];
        // 将嘴部背景色改为绿色
        [mouth setBackgroundColor:[[UIColor greenColor] colorWithAlphaComponent:0.3]];
        // 根据脸部设置 mouthView 的位置
        [mouth setCenter:faceFeature.mouthPosition];
    }
}

```

```

// we'll iterate through every detected face. CIFaceFeature provides us
// with the width for the entire face, and the coordinates of each eye
// and the mouth if detected. Also provided are BOOL's for the eye's and
// mouth so we can check if they already exist.
//   for (features in image)
//   {
for(CIFaceFeature* faceFeature in features)
{
    // get the width of the face
    CGFloat faceWidth = faceFeature.bounds.size.width;

    // create a UIView using the bounds of the face
    UIView* faceView = [[UIView alloc] initWithFrame:faceFeature.bounds];

    // add a border around the newly created UIView
    faceView.layer.borderWidth = 1;
    faceView.layer.borderColor = [[UIColor redColor] CGColor];

    // add the new view to create a box around the face
    [self.view addSubview:faceView];

    if(faceFeature.hasLeftEyePosition)
    {
        // create a UIView with a size based on the width of the face
        UIView* leftEyeView = [[UIView alloc]
initWithFrame:CGRectMake(faceFeature.leftEyePosition.x-faceWidth*0.15,
faceFeature.leftEyePosition.y-faceWidth*0.15, faceWidth*0.3, faceWidth*0.3)];
        // change the background color of the eye view
        [leftEyeView setBackgroundColor:[[UIColor blueColor] colorWithAlphaComponent:0.3]];
        // set the position of the leftEyeView based on the face
        [leftEyeView setCenter:faceFeature.leftEyePosition];
        // round the corners
        leftEyeView.layer.cornerRadius = faceWidth*0.15;
        // add the view to the window
        [self.view addSubview:leftEyeView];
    }

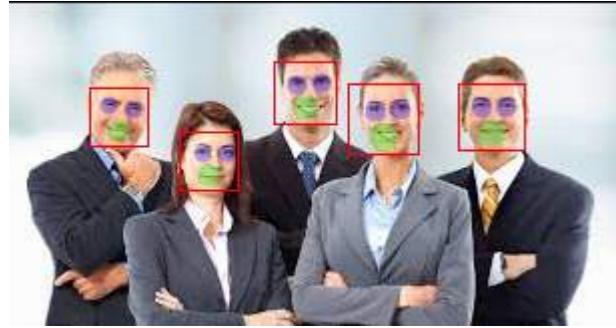
    if(faceFeature.hasRightEyePosition)
    {
        // create a UIView with a size based on the width of the face
        UIView* rightEye = [[UIView alloc]
initWithFrame:CGRectMake(faceFeature.rightEyePosition.x-faceWidth*0.15,
faceFeature.rightEyePosition.y-faceWidth*0.15, faceWidth*0.3, faceWidth*0.3)];
        // change the background color of the eye view
        [rightEye setBackgroundColor:[[UIColor blueColor] colorWithAlphaComponent:0.3]];
        // set the position of the rightEyeView based on the face
        [rightEye setCenter:faceFeature.rightEyePosition];
        // round the corners
        rightEye.layer.cornerRadius = faceWidth*0.15;
        // add the new view to the window
        [self.view addSubview:rightEye];
    }

    if(faceFeature.hasMouthPosition)
    {
        // create a UIView with a size based on the width of the face
        UIView* mouth = [[UIView alloc] initWithFrame:CGRectMake(faceFeature.mouthPosition.x-
faceWidth*0.2, faceFeature.mouthPosition.y-faceWidth*0.2, faceWidth*0.4, faceWidth*0.4)];
        // change the background color for the mouth to green
        [mouth setBackgroundColor:[[UIColor greenColor] colorWithAlphaComponent:0.3]];
        // set the position of the mouthView based on the face
        [mouth setCenter:faceFeature.mouthPosition];
    }
}

```

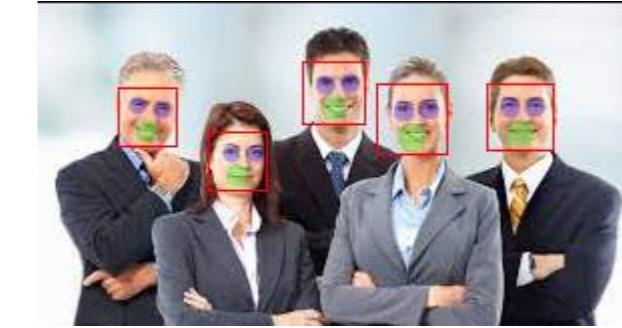
```
// 圆角处理  
嘴部。layer.cornerRadius = faceWidth*0.2;  
// 将新视图添加到窗口  
[self.view addSubview:mouth];  
}  
}  
// }
```

功能的模拟器截图



```
// round the corners  
mouth.layer.cornerRadius = faceWidth*0.2;  
// add the new view to the window  
[self.view addSubview:mouth];  
}  
}  
// }
```

The Simulator ScreenShot for the Function



第159章：MPMediaPickerControllerDelegate

第159.1节：使用 MPMediaPickerControllerDelegate加载音乐并用 AVAudioPlayer播放

按照以下步骤进行：

- 在Info.plist中添加“NSAppleMusicUsageDescription”以获取隐私权限。
- 确保你的音乐已存储在iPhone中。模拟器中无法使用。

版本 ≥ iOS 10.0.1

```
import UIKit
import AVFoundation
import MediaPlayer

class ViewController: UIViewController, MPMediaPickerControllerDelegate {

    var avMusicPlayer: AVAudioPlayer!
    var mpMediapicker: MPMediaPickerController!
    var mediaItems = [MPMediaItem]()
    let currentIndex = 0

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool){
        //该怎么办？
    }

    func mediaPicker(_ mediaPicker: MPMediaPickerController, didPickMediaItems mediaItemCollection: MPMediaItemCollection) {
        mediaItems = mediaItemCollection.items
        updatePlayer()
        self.dismiss(animated: true, completion: nil)
    }

    func updatePlayer(){
        let item = mediaItems[currentIndex]
        // 尝试使用路径设置 AVAudioPlayer，如果成功，则设置 AVMusicPlayer 和歌曲相关值。

        if let path: NSURL = item.assetURL as NSURL? {
            do {
                avMusicPlayer = try AVAudioPlayer(contentsOf: path as URL)
                avMusicPlayer.enableRate = true
            } catch {
                avMusicPlayer = nil
            }
        }
    }

    @IBAction func Play(_ sender: AnyObject) {

```

Chapter 159: MPMediaPickerControllerDelegate

Section 159.1: Load music with MPMediaPickerControllerDelegate and play it with AVAudioPlayer

Go through the steps:

- Add 'NSAppleMusicUsageDescription' to your Info.plist for the privacy authority.
- Make sure your music is available in your iPhone. It will not work in the simulator.

Version ≥ iOS 10.0.1

```
import UIKit
import AVFoundation
import MediaPlayer

class ViewController: UIViewController, MPMediaPickerControllerDelegate {

    var avMusicPlayer: AVAudioPlayer!
    var mpMediapicker: MPMediaPickerController!
    var mediaItems = [MPMediaItem]()
    let currentIndex = 0

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool){
        //What to do?
    }

    func mediaPicker(_ mediaPicker: MPMediaPickerController, didPickMediaItems mediaItemCollection: MPMediaItemCollection) {
        mediaItems = mediaItemCollection.items
        updatePlayer()
        self.dismiss(animated: true, completion: nil)
    }

    func updatePlayer(){
        let item = mediaItems[currentIndex]
        // DO-TRY-CATCH try to setup AVAudioPlayer with the path, if successful, sets up the
        // AVMusicPlayer, and song values.
        if let path: NSURL = item.assetURL as NSURL? {
            do {
                avMusicPlayer = try AVAudioPlayer(contentsOf: path as URL)
                avMusicPlayer.enableRate = true
                avMusicPlayer.rate = 1.0
                avMusicPlayer.numberOfLoops = 0
                avMusicPlayer.currentTime = 0
            } catch {
                avMusicPlayer = nil
            }
        }
    }

    @IBAction func Play(_ sender: AnyObject) {

```

```
//AVMusicPlayer.deviceCurrentTime
avMusicPlayer.play()
}

@IBAction func Stop(_ sender: AnyObject) {
    avMusicPlayer.stop()
}

@IBAction func picker(_ sender: AnyObject) {
    mpMediapicker = MPMediaPickerController.self(mediaTypes:MPMediaType.music)
    mpMediapicker.allowsPickingMultipleItems = false
    mpMediapicker.delegate = self
    self.present(mpMediapicker, animated: true, completion: nil)
}

}
```

```
//AVMusicPlayer.deviceCurrentTime
avMusicPlayer.play()
}

@IBAction func Stop(_ sender: AnyObject) {
    avMusicPlayer.stop()
}

@IBAction func picker(_ sender: AnyObject) {
    mpMediapicker = MPMediaPickerController.self(mediaTypes:MPMediaType.music)
    mpMediapicker.allowsPickingMultipleItems = false
    mpMediapicker.delegate = self
    self.present(mpMediapicker, animated: true, completion: nil)
}

}
```

第160章：图表 (Coreplot)

第160.1节：使用CorePlot制作图表

Core Plot提供了podspec，因此您可以使用cocoapods作为您的库管理器，这将使安装和更新更加简单

在您的系统上安装cocoapods

在项目目录中通过输入pod `init`添加一个名为Podfile的文本文件到您的项目中

在Podfile中添加这一行 `pod 'CorePlot', '~> 1.6'`

在终端中，切换到您的项目目录并运行pod `install`

Cocoapods将生成一个xcworkspace文件，您应使用该文件启动项目 (.xcodeproj文件将不包含pod库)

打开由 CocoaPods 生成的 .xcworkspace

在 ViewController.h 文件中

```
#import <CorePlot/ios/CorePlot.h>
//#import "CorePlot-CocoaTouch.h" 或上述导入语句
@interface ViewController : UIViewController<CPTPlotDataSource>
```

在 ViewController.m 文件中

```
-(void)loadView
{
    [super loadView];
    // 我们需要一个 hostview，你可以在 IB 中创建一个（并创建一个 outlet），或者直接这样做：
    CPTGraphHostingView* hostView = [[CPTGraphHostingView alloc] initWithFrame:CGRectMake(10, 40, 300, 400)];
    hostView.backgroundColor=[UIColor whiteColor];
    self.view.backgroundColor=[UIColor blackColor];
    [self.view addSubview: hostView];
    // 创建一个 CPTGraph 对象并添加到 hostView
    CPTGraph* graph = [[CPTXYGraph alloc] initWithFrame:CGRectMake(10, 40, 300, 400)];
    hostView.hostedGraph = graph;
    // 从图表中获取（默认）plotspace，以便我们设置其 x/y 范围
    CPTXYPlotSpace *plotSpace = (CPTXYPlotSpace *) graph.defaultPlotSpace;
    // 注意这些 CPTPlotRange 是由起点和长度定义的（不是起点和终点）！！
    [plotSpace setYRange: [CPTPlotRange plotRangeWithLocation:CPTDecimalFromFloat( 0 )
length:CPTDecimalFromFloat( 20 )]];
    [plotSpace setXRange: [CPTPlotRange plotRangeWithLocation:CPTDecimalFromFloat( -4 )
length:CPTDecimalFromFloat( 8 )]];
    // 创建图表（我们尚未定义实际的 x/y 值，这些将由数据源提供...）

    CPTScatterPlot* plot = [[CPTScatterPlot alloc] initWithFrame:CGRectMakeZero];
    // 我们保持简单，让此类充当数据源（因此我们实现了
    <CPTPlotDataSource>
    plot.dataSource = self;
    // 最后，将创建的图表添加到我们之前创建的 CPTGraph 对象的默认绘图空间中

    [graph addPlot:plot toPlotSpace:graph.defaultPlotSpace];
}
// 该方法存在是因为此类也作为我们图表的数据源
```

Chapter 160: Graph (Coreplot)

Section 160.1: Making graphs with CorePlot

Core Plot provides a podspec, so you can use cocoapods as your library manager which should make installing and updating much simpler

Install cocoapods on your system

In the project directory add a text file to your project called Podfile by typing pod `init` in your project directory

In the Podfile add the line `pod 'CorePlot', '~> 1.6'`

In the terminal, cd to your project directory and run pod `install`

Cocoapods will generate a xcworkspace file, which you should use for launching your project (the .xcodeproj file will not include the pod libraries)

Open the .xcworkspace generated by CocoaPods

In the ViewController.h File

```
#import <CorePlot/ios/CorePlot.h>
//#import "CorePlot-CocoaTouch.h" or the above import statement
@interface ViewController : UIViewController<CPTPlotDataSource>
```

In the ViewController.m File

```
-(void)loadView
{
    [super loadView];
    // We need a hostview, you can create one in IB (and create an outlet) or just do this:
    CPTGraphHostingView* hostView = [[CPTGraphHostingView alloc] initWithFrame:CGRectMake(10, 40, 300, 400)];
    hostView.backgroundColor=[UIColor whiteColor];
    self.view.backgroundColor=[UIColor blackColor];
    [self.view addSubview: hostView];
    // Create a CPTGraph object and add to hostView
    CPTGraph* graph = [[CPTXYGraph alloc] initWithFrame:CGRectMake(10, 40, 300, 400)];
    hostView.hostedGraph = graph;
    // Get the (default) plotspace from the graph so we can set its x/y ranges
    CPTXYPlotSpace *plotSpace = (CPTXYPlotSpace *) graph.defaultPlotSpace;
    // Note that these CPTPlotRange are defined by START and LENGTH (not START and END) !!
    [plotSpace setYRange: [CPTPlotRange plotRangeWithLocation:CPTDecimalFromFloat( 0 )
length:CPTDecimalFromFloat( 20 )]];
    [plotSpace setXRange: [CPTPlotRange plotRangeWithLocation:CPTDecimalFromFloat( -4 )
length:CPTDecimalFromFloat( 8 )]];
    // Create the plot (we do not define actual x/y values yet, these will be supplied by the
datasource...)
    CPTScatterPlot* plot = [[CPTScatterPlot alloc] initWithFrame:CGRectMakeZero];
    // Let's keep it simple and let this class act as datasource (therefore we implemnt
<CPTPlotDataSource>
    plot.dataSource = self;
    // Finally, add the created plot to the default plot space of the CPTGraph object we created
before
    [graph addPlot:plot toPlotSpace:graph.defaultPlotSpace];
}
// This method is here because this class also functions as datasource for our graph
```

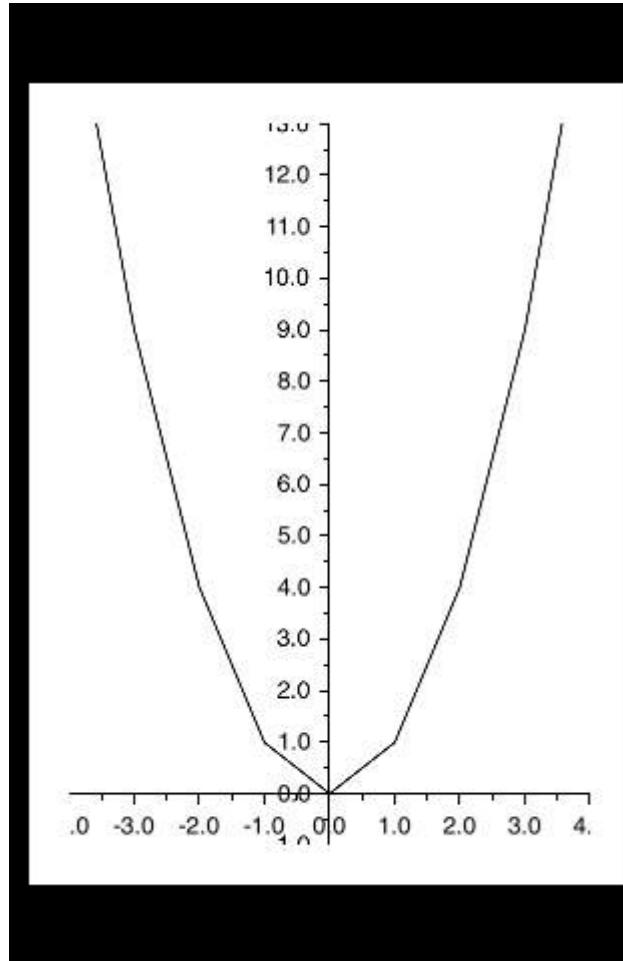
```

// 因此此类实现了 CPTPlotDataSource 协议
-(NSUInteger)numberOfRecordsForPlot:(CPTPlot *)plot numberOfRowsInSection:(NSUInteger)index
{
    // 我们需要为每个索引提供一个X或Y值 (此方法将为每个调用一次)
    int x = index - 4;
    // 实际上此方法对图中的每个点调用两次, 一次用于X值, 一次用于Y值

    if(fieldEnum == CPTScatterPlotFieldX)
    {
        // 返回x值, 根据索引, x值将在-4到4之间
        return [NSNumber numberWithInt: x];
    } else
    {
        // 返回y值, 此示例中我们绘制的是y = x * x
        return [NSNumber numberWithInt: x * x];
    }
}

```

生成的输出如下所示：

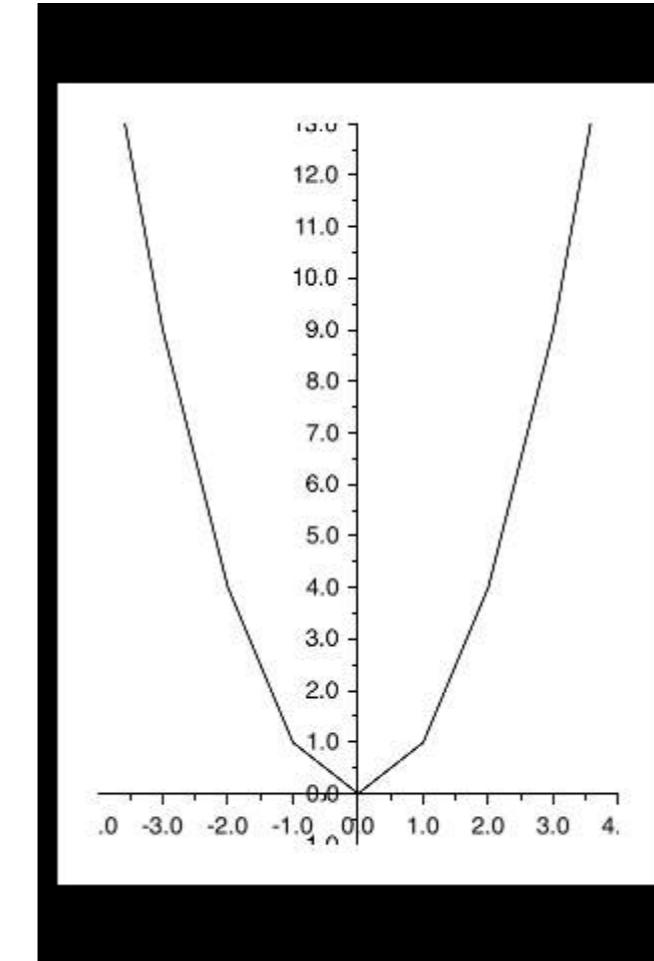


```

// Therefore this class implements the CPTPlotDataSource protocol
-(NSUInteger)numberOfRecordsForPlot:(CPTPlot *)plot numberOfRowsInSection:(NSUInteger)index
{
    // We need to provide an X or Y (this method will be called for each) value for every index
    int x = index - 4;
    // This method is actually called twice per point in the plot, one for the X and one for the Y
    value
    if(fieldEnum == CPTScatterPlotFieldX)
    {
        // Return x value, which will, depending on index, be between -4 to 4
        return [NSNumber numberWithInt: x];
    } else
    {
        // Return y value, for this example we'll be plotting y = x * x
        return [NSNumber numberWithInt: x * x];
    }
}

```

The generated Output is as given below:



第161章：Swift中的FCM消息传递

第161.1节：在Swift中初始化FCM

按照以下步骤将FCM添加到您的Swift项目中

1- 如果您还没有Xcode项目，请现在创建一个。如果没有Podfile，请创建一个：

```
$ cd your-project 目录  
$ pod init
```

2- 添加您想安装的pods。您可以在Podfile中这样包含一个Pod：

```
pod 'Firebase/Core'  
pod 'Firebase/Messaging'
```

3- 安装pods并打开.xcworkspace文件以在Xcode中查看项目。

```
$ pod install  
$ open your-project.xcworkspace
```

4- 从plist下载 GoogleService-Info.plist 文件并将其包含在您的应用中。

5- 将APNs证书上传到Firebase。[APN证书](#)

6- 在项目的AppDelegate文件中添加 "import Firebase"

7- 在你的 "application:didFinishLaunchingWithOptions" 中添加 "FIRApp.configure()"

8- 注册远程通知

```
if #available(iOS 10.0, *) {  
    let authOptions : UNAuthorizationOptions = [.Alert, .Badge, .Sound]  
    UNUserNotificationCenter.currentNotificationCenter().requestAuthorizationWithOptions(  
        authOptions,  
        completionHandler: {_ in })  
  
    // 对于iOS 10显示通知（通过APNS发送）  
    UNUserNotificationCenter.currentNotificationCenter().delegate = self  
    // 对于iOS 10数据消息（通过FCM发送）  
    FIRMessaging.messaging().remoteMessageDelegate = self  
  
} else {  
    let settings: UIUserNotificationSettings =  
        UIUserNotificationSettings(forTypes: [.Alert, .Badge, .Sound], categories: nil)  
    application.registerUserNotificationSettings(settings)  
}  
  
application.registerForRemoteNotifications()
```

9- 获取注册令牌使用

Chapter 161: FCM Messaging in Swift

Section 161.1: Initialize FCM in Swift

follow below step to add FCM in your swift Project

1- If you don't have an Xcode project yet, create one now. Create a Podfile if you don't have one:

```
$ cd your-project directory  
$ pod init
```

2- Add the pods that you want to install. You can include a Pod in your Podfile like this:

```
pod 'Firebase/Core'  
pod 'Firebase/Messaging'
```

3- Install the pods and open the .xcworkspace file to see the project in Xcode.

```
$ pod install  
$ open your-project.xcworkspace
```

4- Download a GoogleService-Info.plist file from [plist](#) and include it in your app.

5- Upload APNs certificate to Firebase. [APN Cert](#)

6- add "import Firebase" in your AppDelegate file of project

7- add this "FIRApp.configure()" in your "application:didFinishLaunchingWithOptions"

8- register for remote notification

```
if #available(iOS 10.0, *) {  
    let authOptions : UNAuthorizationOptions = [.Alert, .Badge, .Sound]  
    UNUserNotificationCenter.currentNotificationCenter().requestAuthorizationWithOptions(  
        authOptions,  
        completionHandler: {_ in })  
  
    // For iOS 10 display notification (sent via APNS)  
    UNUserNotificationCenter.currentNotificationCenter().delegate = self  
    // For iOS 10 data message (sent via FCM)  
    FIRMessaging.messaging().remoteMessageDelegate = self  
  
} else {  
    let settings: UIUserNotificationSettings =  
        UIUserNotificationSettings(forTypes: [.Alert, .Badge, .Sound], categories: nil)  
    application.registerUserNotificationSettings(settings)  
}  
  
application.registerForRemoteNotifications()
```

9- to get register token use

```
let token = FIRInstanceID.instanceID().token()!
```

10- 如果你想监控 token 变化, 请在 appDelegate 文件中使用以下代码

```
func tokenRefreshNotification(notification: NSNotification) {
if let refreshedToken = FIRInstanceID.instanceID().token() {
    print("InstanceID token: \(refreshedToken)")
}

// 连接到 FCM, 因为在获取 token 之前尝试连接可能失败。
connectToFcm()
}
```

11- 要接收来自 FCM 的消息, 请在 appDelegate 中添加以下代码

```
func connectToFcm() {
FIRMessaging.messaging().connectWithCompletion { (error) in
    if (error != nil) {
        print("无法连接到 FCM。 \(error)")
    } else {
        print("已连接到 FCM。")
    }
}
```

12- 和用于断开连接

```
func applicationWillEnterBackground(application: UIApplication) {
    FIRMessaging.messaging().disconnect()
    print("已断开与 FCM 的连接。")
}
```

在你的 appDelegate 中。

初始化完成, 客户端已准备好接收来自 FCM 面板的消息或通过第三方服务器的令牌发送的消息

```
let token = FIRInstanceID.instanceID().token()!
```

10- and if you want monitor for token change use below code in appDelegate file

```
func tokenRefreshNotification(notification: NSNotification) {
if let refreshedToken = FIRInstanceID.instanceID().token() {
    print("InstanceID token: \(refreshedToken)")

}

// Connect to FCM since connection may have failed when attempted before having a token.
connectToFcm()
}
```

11- to receive message from fcm add below code in appDelegate

```
func connectToFcm() {
FIRMessaging.messaging().connectWithCompletion { (error) in
    if (error != nil) {
        print("Unable to connect with FCM. \(error)")
    } else {
        print("Connected to FCM.")
    }
}
```

12- and for disconnect use

```
func applicationWillEnterBackground(application: UIApplication) {
    FIRMessaging.messaging().disconnect()
    print("Disconnected from FCM.")
}
```

in your appDelegate.

the initialization complete and client ready to receive message from fcm panel or send by token from third party server

第162章：在 iOS 中创建自定义框架

第162.1节：用 Swift 创建框架

按照以下步骤在 Swift-iOS 中创建自定义框架：

1. 在 Xcode 中创建一个新项目
2. 选择 iOS/Framework & Library/Cocoa Touch Framework 来创建一个新框架
3. 点击下一步并设置产品名称
4. 点击下一步并选择目录以在那里创建项目
5. 向创建的项目添加代码和资源

框架创建成功

要将创建的框架添加到另一个项目，首先应创建一个工作区

将“目标项目”和“框架项目”添加到工作区，然后：

1. 转到目标项目的常规选项卡
2. 将框架项目的产品文件夹中的“*.framework”文件拖到“嵌入式二进制文件”部分
3. 要在任何视图控制器或类中使用，只需在每个文件中导入框架

Chapter 162: Create a Custom framework in iOS

Section 162.1: Create Framework in Swift

follow these step to creating Custom Framework in Swift-IOS:

1. Create a new project. In Xcode
2. Choose iOS/Framework & Library/Cocoa Touch Framework to create a new framework
3. click next and set the productName
4. click next and choose directory to create Project there
5. add code and resources to created project

Framework created successfully

to add created framework to another project, first you should create a workspace
add "target project" and "framework project" to workspace, then :

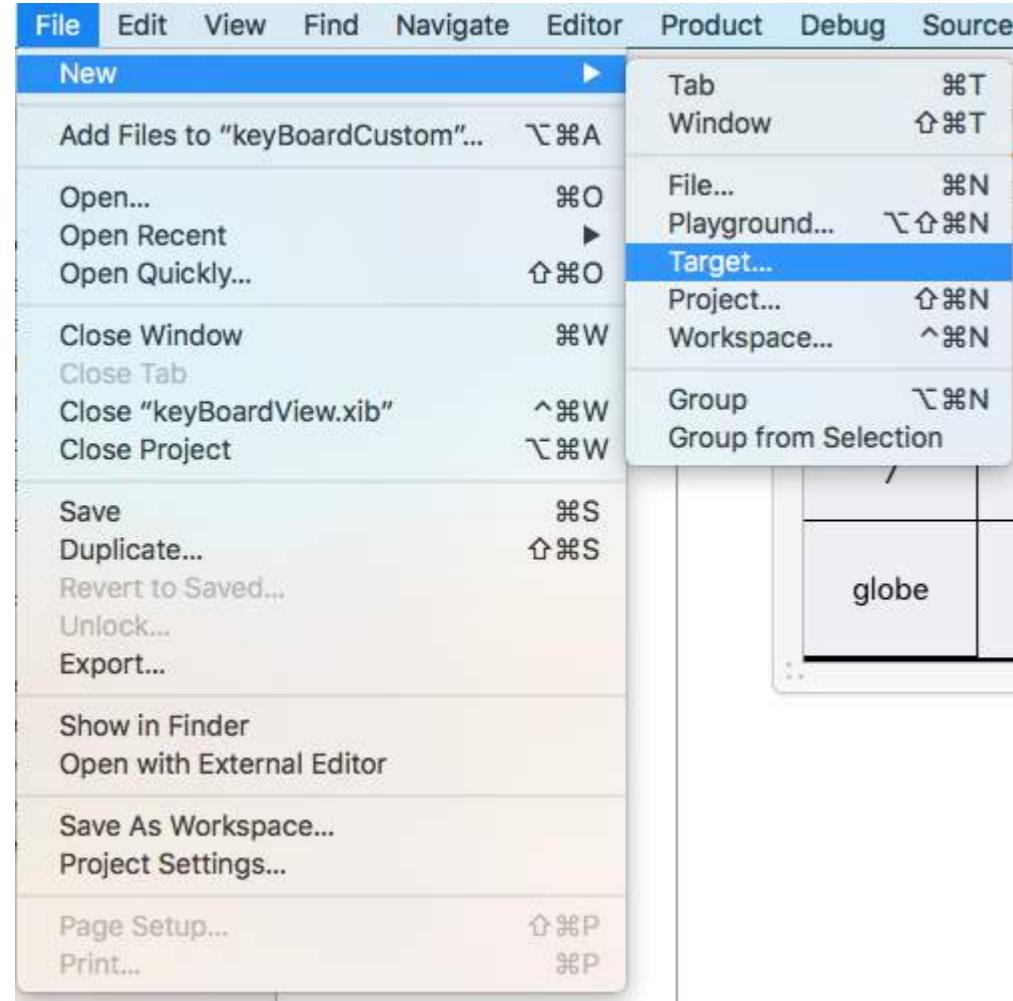
1. go to the general tab of target project
2. drag the "*.framework" file in product folder of framework project to "Embedded Binaries" section
3. to use in any ViewController or class just import framework at each file

第163章：自定义键盘

第163.1节：自定义键盘示例

Objective-C 和 Xib

向现有的XCode项目添加目标



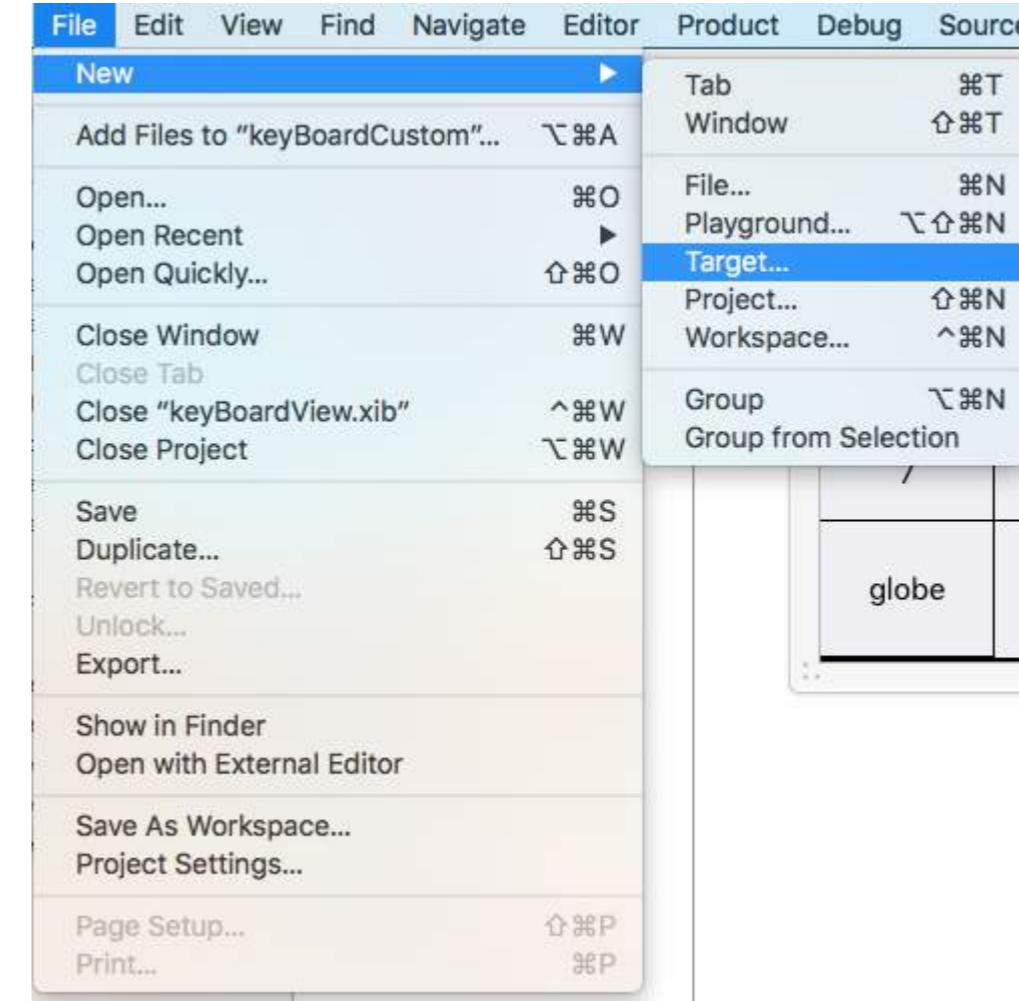
在“添加目标”中选择“自定义键盘”

Chapter 163: Custom Keyboard

Section 163.1: Custom KeyBoard Example

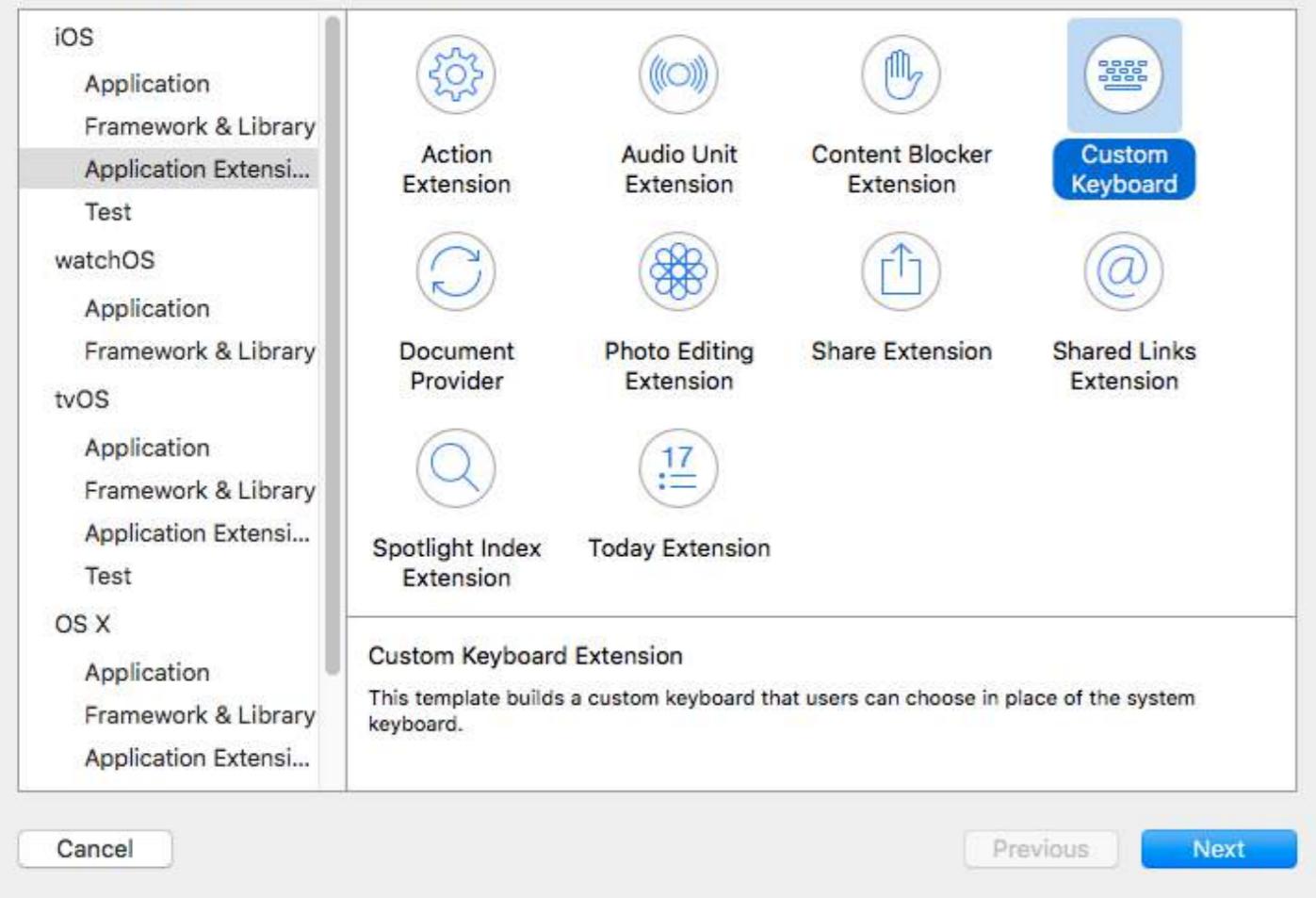
Objective-C and Xib

Add a target to an existing XCode project



In the Add Target select Custom KeyBoard

Choose a template for your new target:

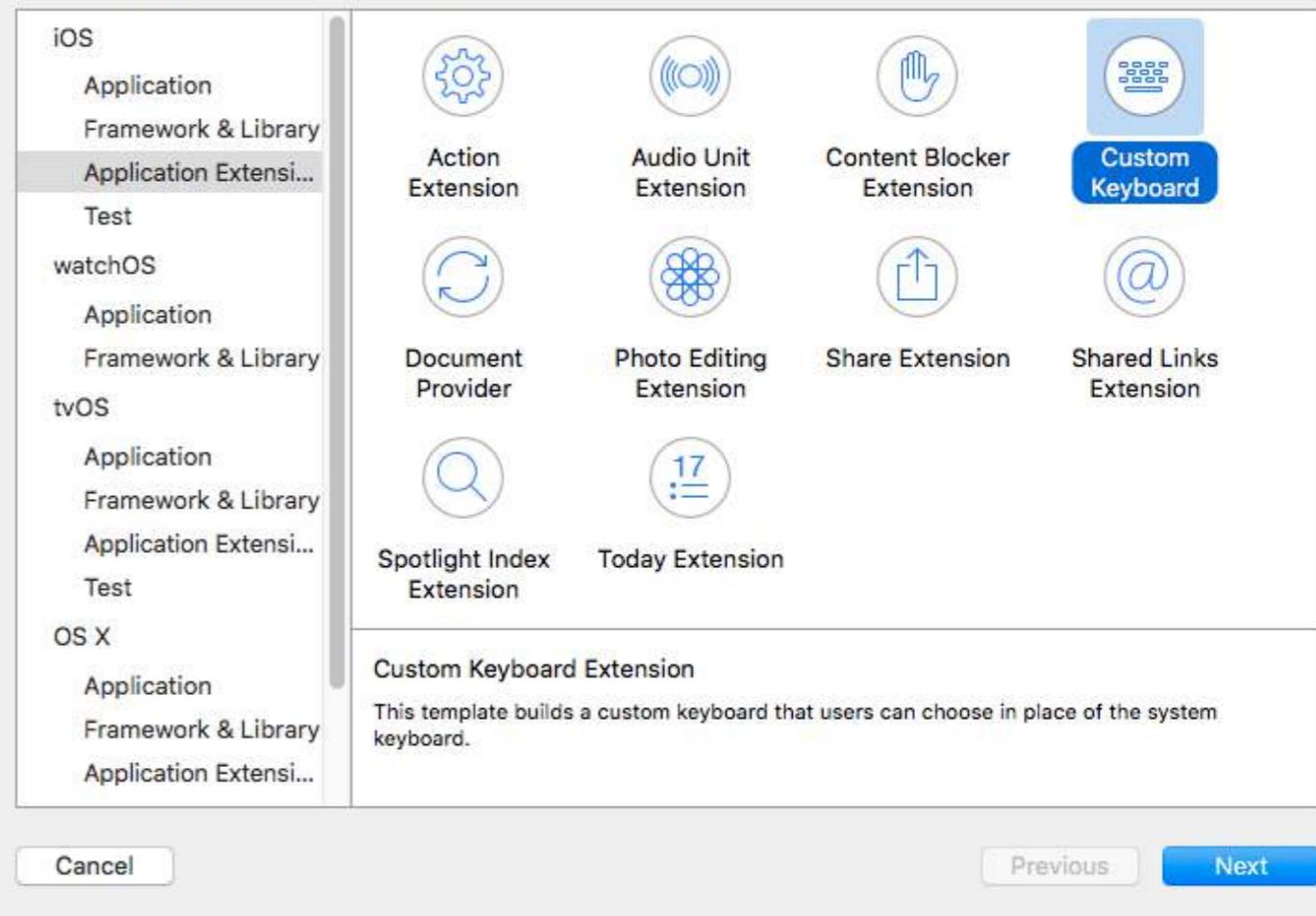


Cancel

Previous

Next

Choose a template for your new target:



Cancel

Previous

Next

像这样添加目标：

Add the target like this:

Choose options for your new target:

Product Name: myKeyboard

Organization Name: ibrahim

Organization Identifier: com.capanicus.keyBoardCustom

Bundle Identifier: com.capanicus.keyBoardCustom.myKeyboard

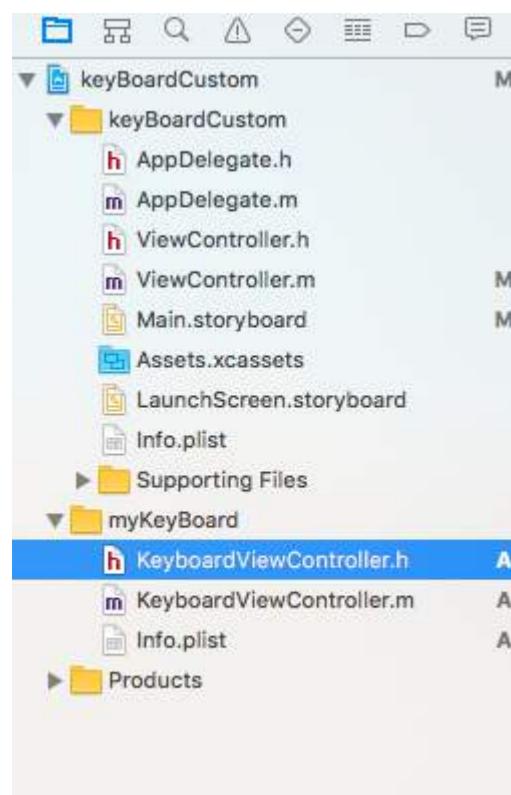
Language: Objective-C

Project: keyBoardCustom

Embed in Application: keyBoardCustom

Cancel Previous Finish

你的项目文件目录应该看起来像这样



这里 myKeyboard 是添加的目标名称

添加一个类型为 UIView 的新的 Cocoatouch 文件，并添加一个接口文件

Choose options for your new target:

Product Name: myKeyboard

Organization Name: ibrahim

Organization Identifier: com.capanicus.keyBoardCustom

Bundle Identifier: com.capanicus.keyBoardCustom.myKeyboard

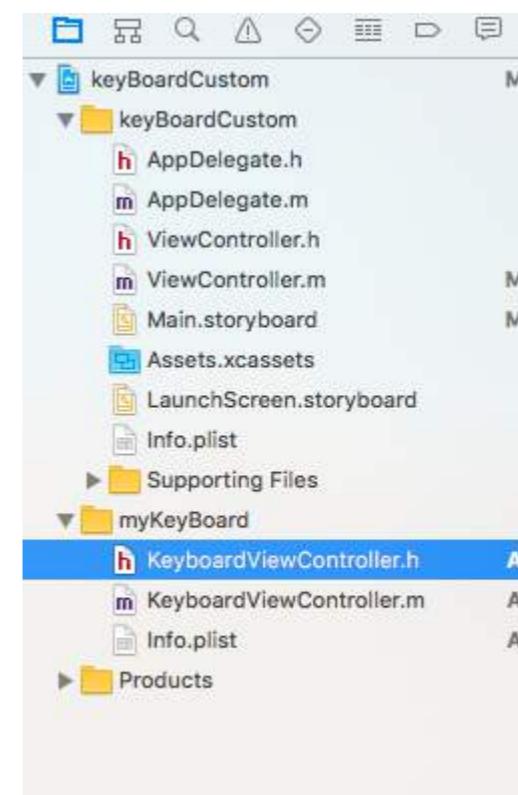
Language: Objective-C

Project: keyBoardCustom

Embed in Application: keyBoardCustom

Cancel Previous Finish

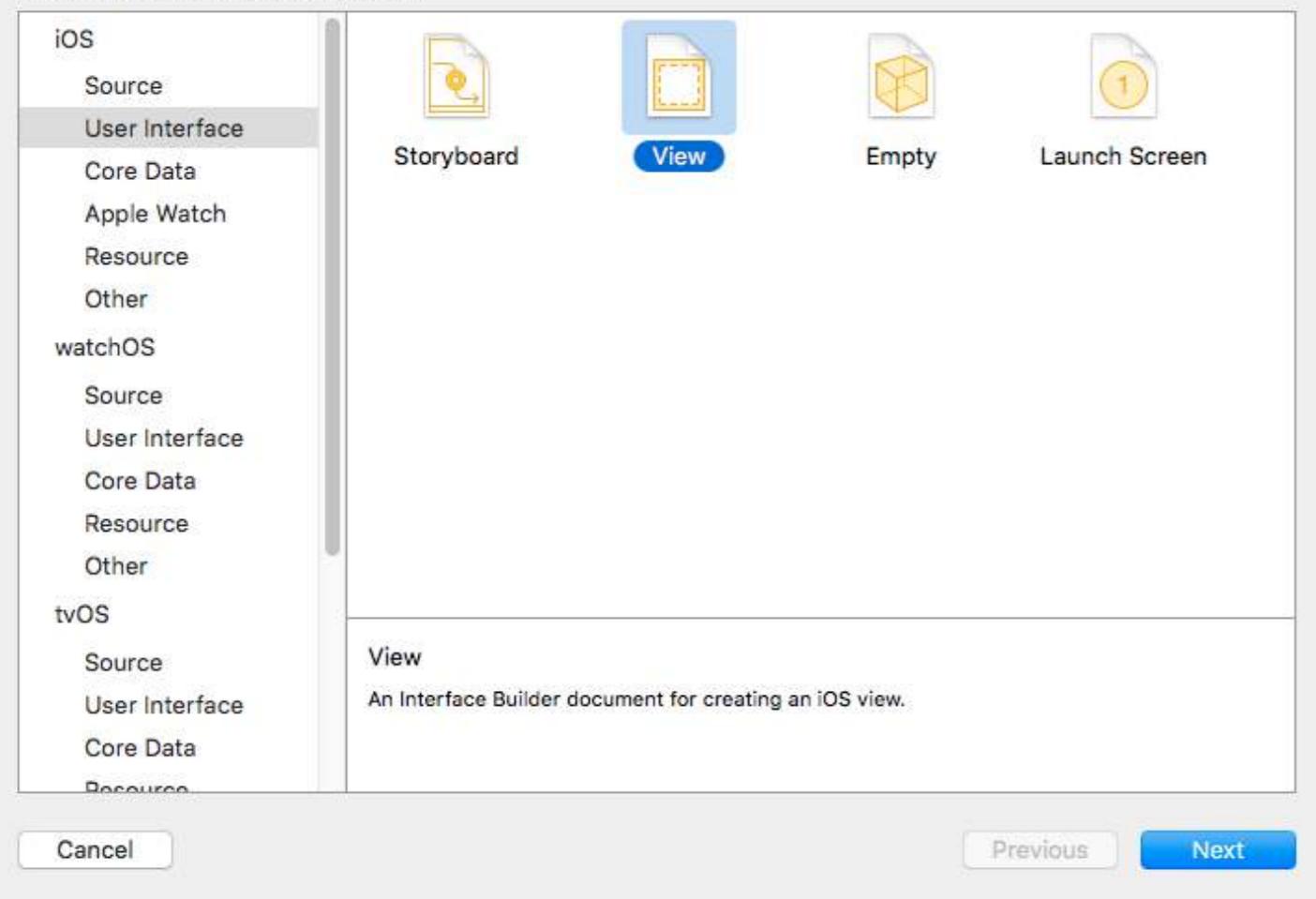
Your project file directory should look something like this



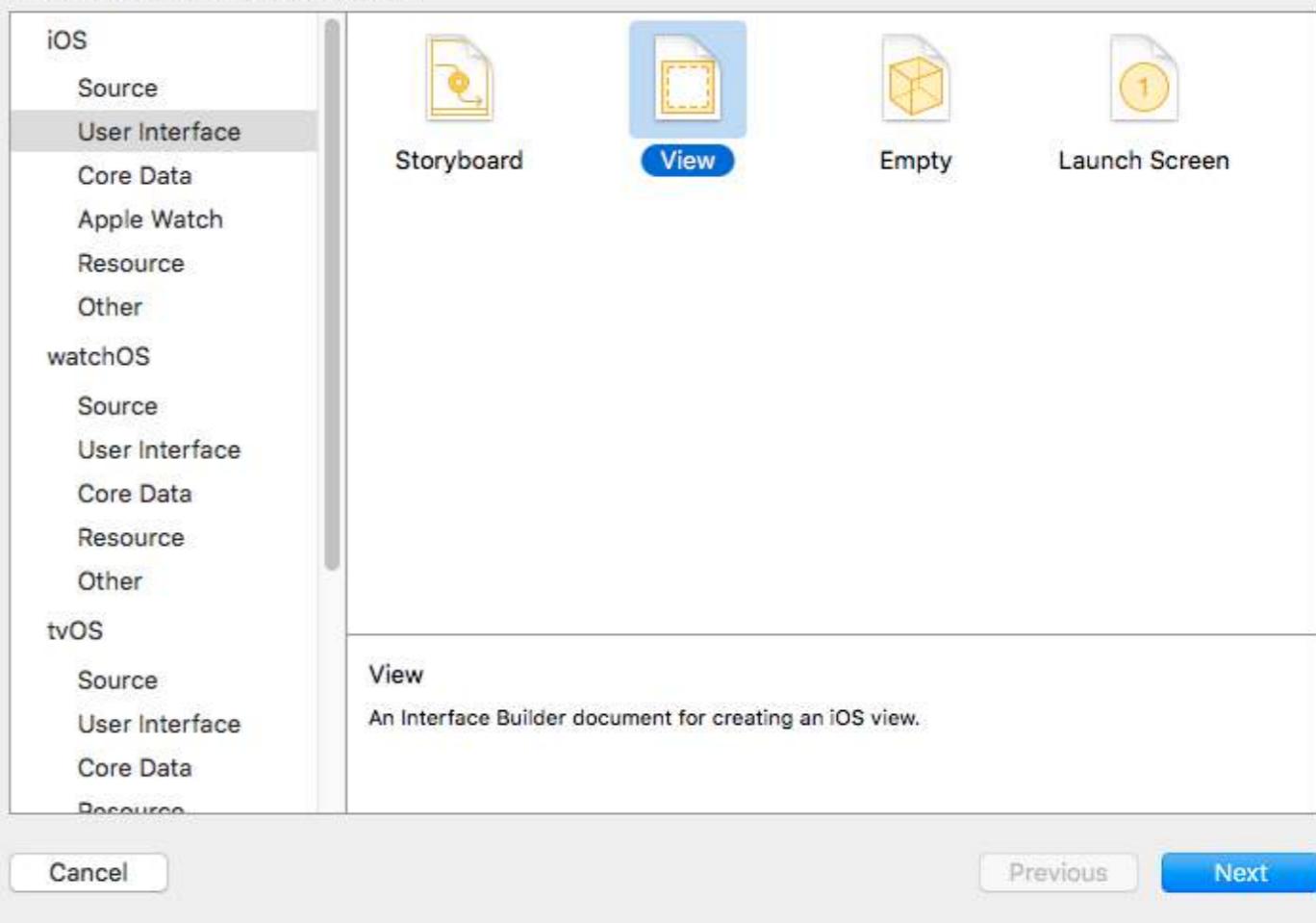
Here myKeyboard is the name of the added Target

Add new Cocoatouch file of type of type UIView and add an interface file

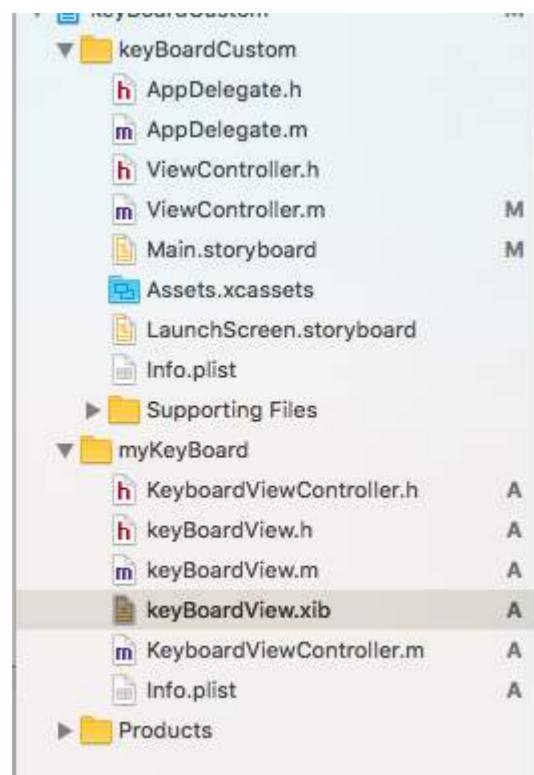
Choose a template for your new file:



Choose a template for your new file:

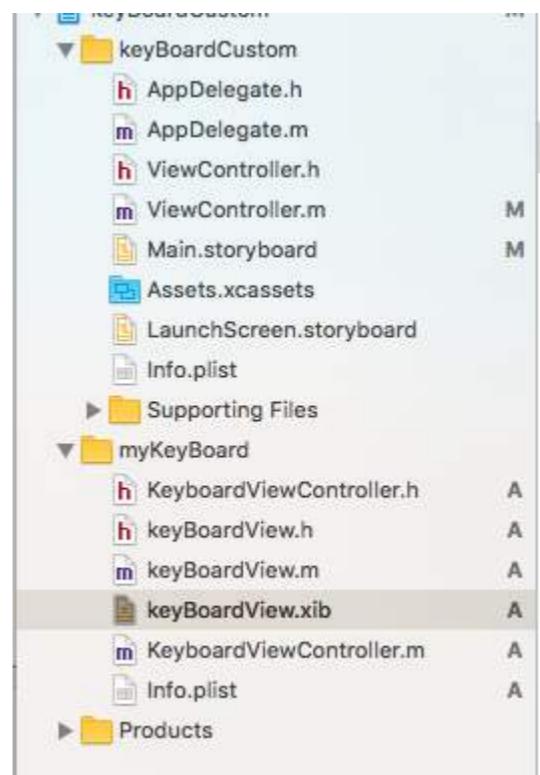


最后你的项目目录应该看起来像这样

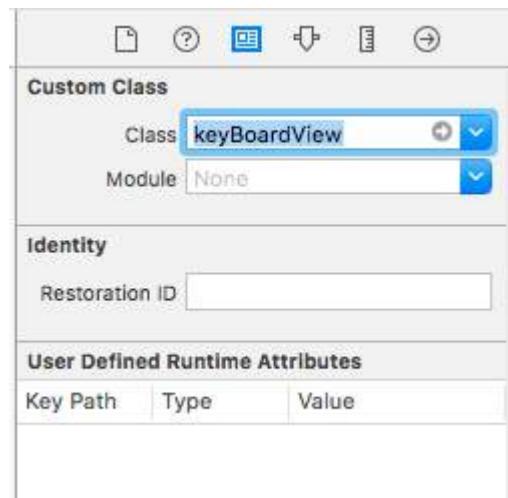


将keyBoardView.xib设置为keyBoardView的子类

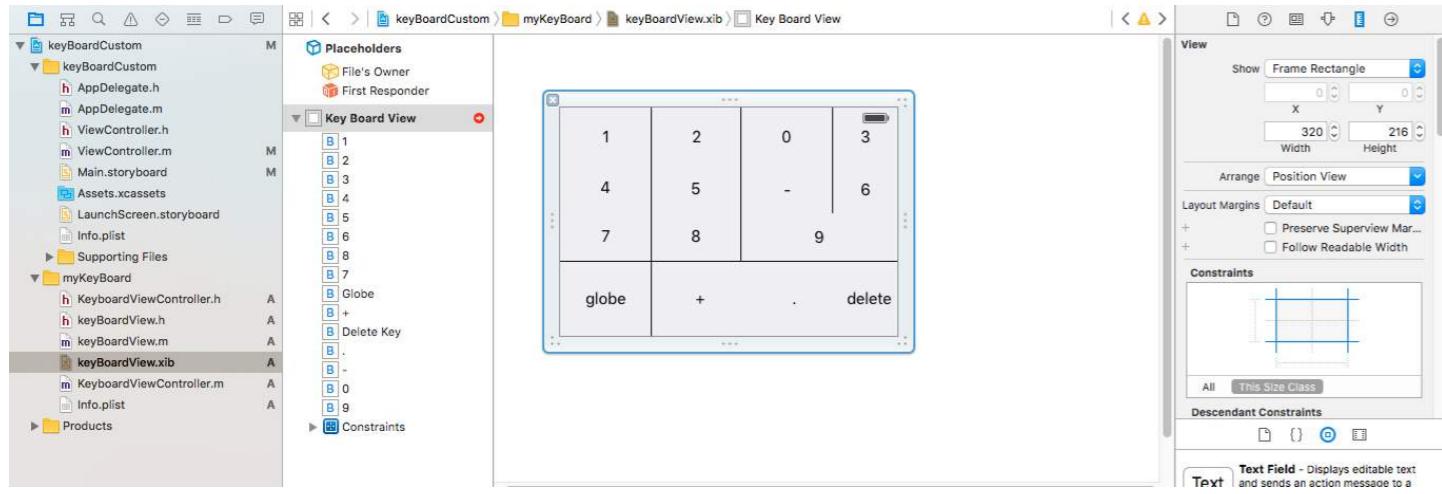
Finally your project directory should look like this



make the keyBoardView.xib a subclass of keyBoardView



在keyBoardView.xib文件中创建接口



将连接从keyBoardView.xib到keyBoardView.h文件

keyBoardView.h应如下所示

```
#import <UIKit/UIKit.h>

@interface keyBoardView : UIView

@property (weak, nonatomic) IBOutlet UIButton *deleteKey;
// 删除键的IBOutlet
@property (weak, nonatomic) IBOutlet UIButton *globe;
// 标题为globe的键的Outlet，用于切换键盘类型
@property (strong, nonatomic) IBOutletCollection(UIButton) NSArray *keys;
// 包含所有按键的集合，包括'0'到'9' '+' '-' 和 '.'

@end
```

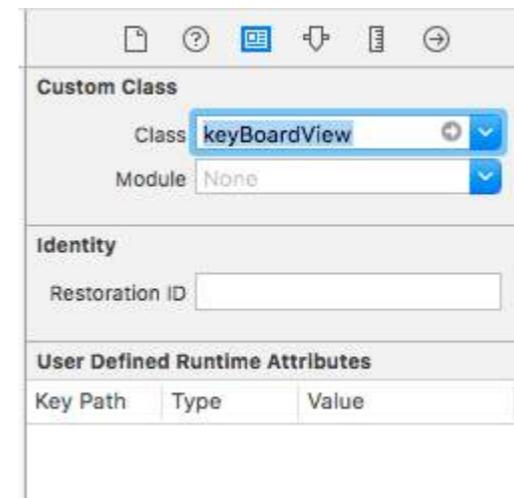
在keyBoardViewController.h文件中导入#import "keyBoardView.h"

声明一个键盘属性@property (strong, nonatomic)keyBoardView *keyboard;

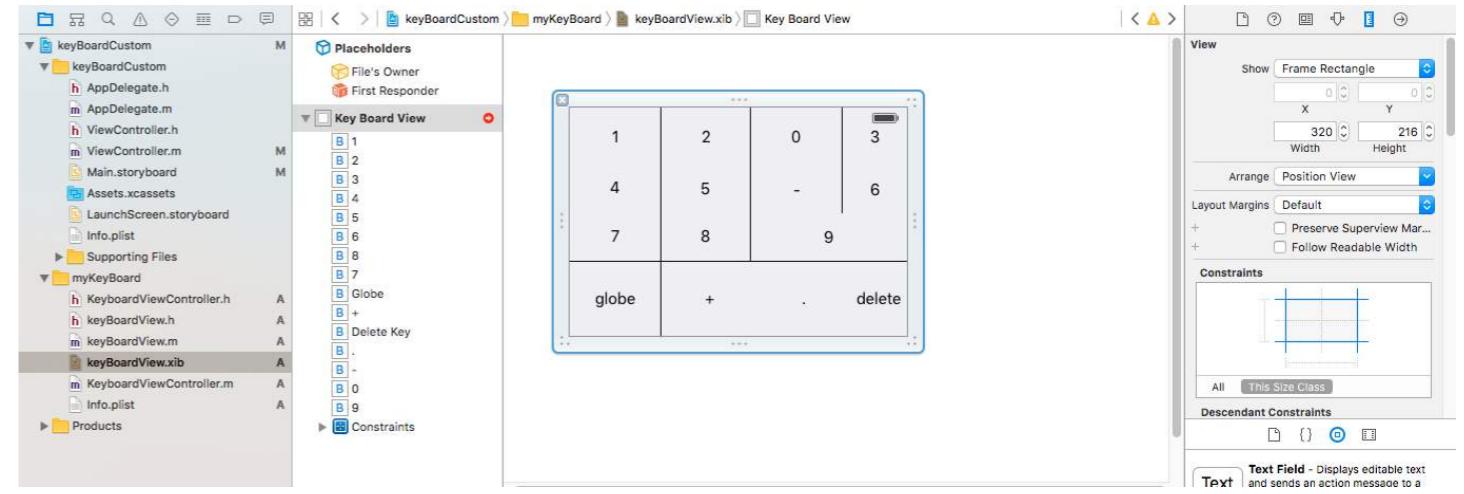
注释掉

```
@property (nonatomic, strong) UIButton *nextKeyboardButton
```

KeyboardViewController.m文件中的viewDidLoad()函数应如下所示



Make interface in the keyBoardView.xib file



Make connections to from the keyBoardView.xib to keyBoardView.h file

keyBoardView.h should look like

```
#import <UIKit/UIKit.h>

@interface keyBoardView : UIView

@property (weak, nonatomic) IBOutlet UIButton *deleteKey;
//IBOutlet for the delete Key
@property (weak, nonatomic) IBOutlet UIButton *globe;
//Outlet for the key with title globe which changes the keyboard type
@property (strong, nonatomic) IBOutletCollection(UIButton) NSArray *keys;
//Contains a collection of all the keys '0' to '9' '+' '-' and '.'

@end
```

In the keyBoardViewController.h file import #import "keyBoardView.h"

Declare a property for keyboard @property (strong, nonatomic)keyBoardView *keyboard;

Comment out the

```
@property (nonatomic, strong) UIButton *nextKeyboardButton
```

The KeyboardViewController.m file's viewDidLoad() function should look like this

```

- (void)viewDidLoad {
    [super viewDidLoad];
    self.keyboard=[[NSBundle mainBundle]loadNibNamed:@"keyBoardView" owner:nil
options:nil]objectAtIndex:0];
    self.inputView=self.keyboard;
    [self 添加手势到键盘];

    // 在此执行自定义UI设置
    // self.nextKeyboardButton = [UIButton buttonWithType:UIButtonTypeSystem];
    //
    // [self.nextKeyboardButton setTitle:NSLocalizedString(@"Next Keyboard", @"下一个键盘'按钮的标题") forState:UIControlStateNormal];
    // [self.nextKeyboardButton sizeToFit];
    // self.nextKeyboardButton.translatesAutoresizingMaskIntoConstraints = NO;
    //
    // [self.nextKeyboardButton addTarget:self action:@selector(advanceToNextInputMode)
    forControlEvents:UIControlEventTouchUpInside];
    //
    // [self.view addSubview:self.nextKeyboardButton];
    //
    // [self.nextKeyboardButton.leftAnchor constraintEqualToAnchor:self.view.leftAnchor].active =
    YES;
    // [self.nextKeyboardButton.bottomAnchor constraintEqualToAnchor:self.view.bottomAnchor].active
    = YES;
}

```

函数addGestureToKeyboard、pressDeleteKey、keyPressed定义如下

```

-(void) addGestureToKeyboard
{
    [self.keyboard.deleteKey 添加目标 : self 动作 : @selector(pressDeleteKey)
事件 : UIControlEventTouchUpInside];
    [self.keyboard.globe 添加目标 : self 动作 : @selector(advanceToNextInputMode)
事件 : UIControlEventTouchUpInside];

    for (UIButton *key 在 self.keyboard.keys中)
    {
        [key 添加目标 : self 动作 : @selector(keyPressed:)
事件 : UIControlEventTouchUpInside];
    }

    [self.textDocumentProxy 向后删除];
}

-(void)keyPressed:(UIButton *)key
{
    [self.textDocumentProxy 插入文本 : [key currentTitle]];
}

```

运行主应用程序，进入设置->通用->键盘->添加新键盘->从第三方键盘部分添加键盘（显示的键盘名称为 keyBoardCustom）

键盘名称可以通过添加一个名为Bundle display name的键来更改，在值字符串中输入主项目键盘的所需名称。

```

- (void)viewDidLoad {
    [super viewDidLoad];
    self.keyboard=[[NSBundle mainBundle]loadNibNamed:@"keyBoardView" owner:nil
options:nil]objectAtIndex:0];
    self.inputView=self.keyboard;
    [self addGestureToKeyboard];

    // Perform custom UI setup here
    // self.nextKeyboardButton = [UIButton buttonWithType:UIButtonTypeSystem];
    //
    // [self.nextKeyboardButton setTitle:NSLocalizedString(@"Next Keyboard", @"Title for 'Next
    // Keyboard' button") forState:UIControlStateNormal];
    // [self.nextKeyboardButton sizeToFit];
    // self.nextKeyboardButton.translatesAutoresizingMaskIntoConstraints = NO;
    //
    // [self.nextKeyboardButton addTarget:self action:@selector(advanceToNextInputMode)
    // forControlEvents:UIControlEventTouchUpInside];
    //
    // [self.view addSubview:self.nextKeyboardButton];
    //
    // [self.nextKeyboardButton.leftAnchor constraintEqualToAnchor:self.view.leftAnchor].active =
    YES;
    // [self.nextKeyboardButton.bottomAnchor constraintEqualToAnchor:self.view.bottomAnchor].active
    = YES;
}

```

The functions addGestureToKeyboard, pressDeleteKey, keyPressed are defined below

```

-(void) addGestureToKeyboard
{
    [self.keyboard.deleteKey addTarget:self action:@selector(pressDeleteKey)
forControlEvents:UIControlEventTouchUpInside];
    [self.keyboard.globe addTarget:self action:@selector(advanceToNextInputMode)
forControlEvents:UIControlEventTouchUpInside];

    for (UIButton *key in self.keyboard.keys)
    {
        [key addTarget:self action:@selector(keyPressed:)
forControlEvents:UIControlEventTouchUpInside];
    }
}

-(void) pressDeleteKey
{
    [self.textDocumentProxy deleteBackward];
}

-(void)keyPressed:(UIButton *)key
{
    [self.textDocumentProxy insertText:[key currentTitle]];
}

```

Run the Main Application and go to Settings->General->Keyboard->Add New Keyboard-> and add keyboard from the third party keyboard section (The displayed keyboardName would be keyBoardCustom)

The keyboard name can be changed by adding a key called Bundle display name and in the Value String Value enter the desired name for the keyboard of the main Project.

	Key	Type	Value
Information Property List			
	Localization native development re...	String	en
	Executable file	String	\$(EXECUTABLE_NAME)
	Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
	InfoDictionary version	String	6.0
	Bundle name	String	\$(PRODUCT_NAME)
	Bundle OS Type code	String	APPL
	Bundle versions string, short	String	1.0
	Bundle display name	String	keyBoardMi
	Bundle creator OS Type code	String	????
	Bundle version	String	1
	Application requires iPhone enviro...	Boolean	YES
	Launch screen interface file base...	String	LaunchScreen
	Main storyboard file base name	String	Main
► Required device capabilities			
► Supported interface orientations			

你也可以观看这个Youtube视频 [_____](#)

	Key	Type	Value
Information Property List			
	Localization native development re...	String	en
	Executable file	String	\$(EXECUTABLE_NAME)
	Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
	InfoDictionary version	String	6.0
	Bundle name	String	\$(PRODUCT_NAME)
	Bundle OS Type code	String	APPL
	Bundle versions string, short	String	1.0
	Bundle display name	String	keyBoardMi
	Bundle creator OS Type code	String	????
	Bundle version	String	1
	Application requires iPhone enviro...	Boolean	YES
	Launch screen interface file base...	String	LaunchScreen
	Main storyboard file base name	String	Main
► Required device capabilities			
► Supported interface orientations			

You can also watch this [Youtube Video](#)

第164章：AirDrop

第164.1节：AirDrop

Objective-C

AirDrop可以通过UIActivityViewController使用。UIActivityViewController类是一个标准视图控制器，提供多种标准服务，例如将项目复制到剪贴板、分享到社交媒体网站、通过信息发送项目、AirDrop以及一些第三方应用程序。

在这种情况下，我们将通过UIActivityViewController发送一张图片

```
UIImage*hatImage = [UIImage imageNamed:@"logo.png"];
if (hatImage)//检查图片文件是否不为nil
{
//初始化一个UIActivityViewController
UIActivityViewController*controller = [[UIActivityViewController alloc]
initWithActivityItems:@[hatImage] applicationActivities:nil];
//从UIActivityViewController菜单中排除以下选项
NSArray*excludeActivities = @[@UIActivityTypePostToWeibo,UIActivityTypePrint,
UIActivityTypeMail,UIActivityTypeMessage,UIActivityTypePostToTwitter,UIActivityTypePostToFacebook,
UIActivityTypeCopyToPasteboard,UIActivityTypeAssignToContact,
UIActivityTypeSaveToCameraRoll,UIActivityTypeAddToReadingList,
UIActivityTypePostToFlickr,UIActivityTypePostToVimeo,
UIActivityTypePostToTencentWeibo];
controller.excludedActivityTypes = excludeActivities;
[self presentViewController:controller animated:YES completion:nil];
}
```

Swift

```
if ((newImage) != nil)
{
    let activityVC = UIActivityViewController(activityItems: [newImage],
applicationActivities: nil)
activityVC.excludedActivityTypes =[UIActivityTypeAddToReadingList]
    self.presentViewController(activityVC, animated: true, completion: nil)
}
```

Chapter 164: AirDrop

Section 164.1: AirDrop

Objective-C

Airdrop can be used from [UIActivityViewController](#). The [UIActivityViewController](#) class is a standard view controller that provides several standard services, such as copying items to the clipboard, sharing content to social media sites, sending items via Messages, AirDrop and some third party applications.

In this case we would be sending an image via [UIActivityViewController](#)

```
UIImage *hatImage = [UIImage imageNamed:@"logo.png"];
if (hatImage)//checks if the image file is not nil
{
//Initialise a UIActivityViewController
UIActivityViewController *controller = [[UIActivityViewController alloc]
initWithActivityItems:@[hatImage] applicationActivities:nil];
//Excludes following options from the UIActivityViewController menu
NSArray *excludeActivities = @[@UIActivityTypePostToWeibo,UIActivityTypePrint,
UIActivityTypeMail,UIActivityTypeMessage,UIActivityTypePostToTwitter,UIActivityTypePostToFacebook,
UIActivityTypeCopyToPasteboard,UIActivityTypeAssignToContact,
UIActivityTypeSaveToCameraRoll,UIActivityTypeAddToReadingList,
UIActivityTypePostToFlickr,UIActivityTypePostToVimeo,
UIActivityTypePostToTencentWeibo];
controller.excludedActivityTypes = excludeActivities;
[self presentViewController:controller animated:YES completion:nil];
}
```

Swift

```
if ((newImage) != nil)
{
    let activityVC = UIActivityViewController(activityItems: [newImage],
applicationActivities: nil)
activityVC.excludedActivityTypes =[UIActivityTypeAddToReadingList]
    self.presentViewController(activityVC, animated: true, completion: nil)
}
```

第165章：SLComposeViewController

第165.1节：用于Twitter、Facebook、新浪微博和腾讯微博的SLComposeViewController

Objective-C

首先将Social Framework添加到XCode项目中。

导入#import "Social/Social.h"类到所需的ViewController

带文本、图片和链接的Twitter

```
//- 在推特上分享文本 --
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeTwitter])
{
    //推文
    SLComposeViewController *twitterVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeTwitter];
    //发送链接和文本
    [twitterVC addURL:[NSURL URLWithString:@"https://twitter.com/IbrahimH_ss_n"]];
    //为链接添加图片
    [twitterVC addImage:[UIImage imageNamed:@"image"]];
    //发送带链接和图片的推文
    [twitterVC setInitialText:text];
    /* 在推文中添加链接和图片时，推文的有效长度即
    用户可输入的字符数会减少。
    推文的最大长度为140个字符*/
    [self presentViewController:twitterVC animated:YES completion:nil];
}
else
{//如果未登录Twitter则显示警告
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"社交分享"
message:@"您尚未登录Twitter."preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}
}
```

带文本、图片和链接的Facebook分享

```
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeFacebook])
{
    SLComposeViewController *fbVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeFacebook];
    [fbVC setInitialText:text];
    //发送链接和文本
    [fbVC addURL:[NSURL URLWithString:@"https://twitter.com/IbrahimH_ss_n"]];
    //添加链接的照片
    [fbVC addImage:[UIImage imageNamed:@"image"]];
    [self presentViewController:fbVC animated:YES completion:nil];
}
else
{//如果未登录Twitter则显示警告
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"您尚未登录推特。"preferredStyle:UIAlertControllerStyleAlert];
}
```

Chapter 165: SLComposeViewController

Section 165.1: SLComposeViewController for Twitter, facebook, SinaWeibo and TencentWeibo

Objective-C

First add the Social Framework to the XCode project.

Import the #import "Social/Social.h" class to the required ViewController

Twitter with text, image and link

```
//- - To Share text on twitter --
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeTwitter])
{
    //Tweet
    SLComposeViewController *twitterVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeTwitter];
    //To send link together with text
    [twitterVC addURL:[NSURL URLWithString:@"https://twitter.com/IbrahimH_ss_n"]];
    //To add a photo to a link
    [twitterVC addImage:[UIImage imageNamed:@"image"]];
    //Sending link and Image with the tweet
    [twitterVC setInitialText:text];
    /* While adding link and images in a tweet the effective length of a tweet i.e.
    the number of characters which can be entered by the user decreases.
    The default maximum length of a tweet is 140 characters*/
    [self presentViewController:twitterVC animated:YES completion:nil];
}
else
{//Shows alert if twitter is not signed in
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to twitter."preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}
}
```

Facebook with Text, Image and Link

```
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeFacebook])
{
    SLComposeViewController *fbVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeFacebook];
    [fbVC setInitialText:text];
    //To send link together with text
    [fbVC addURL:[NSURL URLWithString:@"https://twitter.com/IbrahimH_ss_n"]];
    //To add a photo to a link
    [fbVC addImage:[UIImage imageNamed:@"image"]];
    [self presentViewController:fbVC animated:YES completion:nil];
}
else
{//Shows alert if twitter is not signed in
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to twitter."preferredStyle:UIAlertControllerStyleAlert];
}
```

```

[self presentViewController:alertCont animated:YES completion:nil];
UIAlertAction *okay=[UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleDefault
handler:nil];
[alertCont addAction:okay];
}

```

新浪微博

```

// - 新浪微博 -
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeSinaWeibo]){

    SLComposeViewController *SinaWeiboVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeSinaWeibo];
    [SinaWeiboVC setInitialText:text];

    [self presentViewController:SinaWeiboVC animated:YES completion:nil];
}
else
{
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"社交分享"
message:@"您尚未登录新浪微博。"preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}

```

腾讯微博

```

// -腾讯微博文本分享
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeTencentWeibo])
{
    SLComposeViewController *tencentWeiboVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeTencentWeibo];
    [tencentWeibo setInitialText:text];
    [self presentViewController:tencentWeibo animated:YES completion:nil];
}
else
{
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"社交分享"
message:@"您尚未登录新浪微博。"preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}

```

```

[self presentViewController:alertCont animated:YES completion:nil];
UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
[alertCont addAction:okay];
}

```

SinaWeibo

```

// - SinaWeibo -
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeSinaWeibo]){

    SLComposeViewController *SinaWeiboVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeSinaWeibo];
    [SinaWeiboVC setInitialText:text];

    [self presentViewController:SinaWeiboVC animated:YES completion:nil];
}
else
{
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to SinaWeibo."preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}

```

TencentWeibo

```

// -TencentWeibo text share
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeTencentWeibo])
{
    SLComposeViewController *tencentWeiboVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeTencentWeibo];
    [tencentWeibo setInitialText:text];
    [self presentViewController:tencentWeibo animated:YES completion:nil];
}
else
{
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to SinaWeibo."preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}

```

第166章：iOS中的AirPrint教程

第166.1节：AirPrint打印横幅文本

Objective-C

向ViewController.h文件添加代理和文本格式化器

```
@interface ViewController : UIViewController <UIPrintInteractionControllerDelegate> {
    UISimpleTextPrintFormatter *_textFormatter;
}
```

在ViewController.m文件中定义以下常量

```
#define DefaultFontSize 48
#define PaddingFactor 0.1f
```

打印文本的函数如下：

```
-(IBAction)print:(id)sender {
    /* 获取 UIPrintInteractionController，这是一个共享对象 */
    UIPrintInteractionController *controller = [UIPrintInteractionController
sharedPrintController];
    if(!controller){
        NSLog(@"无法获取共享的 UIPrintInteractionController !");
        return;
    }

    /* 将此对象设置为代理，以便使用
printInteractionController:cutLengthForPaper: 代理方法 */
    controller.delegate = self;

    UIPrintInfo *printInfo = [UIPrintInfo printInfo];
    printInfo.outputType = UIPrintInfoOutputGeneral;

    /* 对于横幅使用横向打印方向，使文本沿纸张长边打印。
*/
    printInfo.orientation = UIPrintInfoOrientationLandscape;

    printInfo.jobName = self.textField.text;
    controller.printInfo = printInfo;

    /* 使用用户在文本字段中提供的文本创建 UISimpleTextPrintFormatter */
    _textFormatter = [[UISimpleTextPrintFormatter alloc] initWithText:self.textField.text];

    /* 根据用户选择设置文本格式化器的颜色和字体属性 */
    _textFormatter.color = [self chosenColor];
    _textFormatter.font = [self chosenFontWithSize:DefaultFontSize];

    /* 在控制器上设置此 UISimpleTextPrintFormatter */
    controller.printFormatter = _textFormatter;

    /* 设置完成处理程序块。如果打印任务在排队前出现错误，将在此处处理。 */
    void (^completionHandler)(UIPrintInteractionController *, BOOL, NSError *) =
^(UIPrintInteractionController *printController, BOOL completed, NSError *error) {
```

Chapter 166: AirPrint tutorial in iOS

Section 166.1: AirPrint printing Banner Text

Objective-C

Add the delegate and a text-formatter to the ViewController.h file

```
@interface ViewController : UIViewController <UIPrintInteractionControllerDelegate> {
    UISimpleTextPrintFormatter *_textFormatter;
}
```

In the ViewController.m file define the following constants

```
#define DefaultFontSize 48
#define PaddingFactor 0.1f
```

The function which prints the text is as follows:

```
-(IBAction)print:(id)sender {
    /* Get the UIPrintInteractionController, which is a shared object */
    UIPrintInteractionController *controller = [UIPrintInteractionController
sharedPrintController];
    if(!controller){
        NSLog(@"Couldn't get shared UIPrintInteractionController!");
        return;
    }

    /* Set this object as delegate so you can use the
printInteractionController:cutLengthForPaper: delegate */
    controller.delegate = self;

    UIPrintInfo *printInfo = [UIPrintInfo printInfo];
    printInfo.outputType = UIPrintInfoOutputGeneral;

    /* Use landscape orientation for a banner so the text print along the long side of the paper.
*/
    printInfo.orientation = UIPrintInfoOrientationLandscape;

    printInfo.jobName = self.textField.text;
    controller.printInfo = printInfo;

    /* Create the UISimpleTextPrintFormatter with the text supplied by the user in the text field
*/
    _textFormatter = [[UISimpleTextPrintFormatter alloc] initWithText:self.textField.text];

    /* Set the text formatter's color and font properties based on what the user chose */
    _textFormatter.color = [self chosenColor];
    _textFormatter.font = [self chosenFontWithSize:DefaultFontSize];

    /* Set this UISimpleTextPrintFormatter on the controller */
    controller.printFormatter = _textFormatter;

    /* Set up a completion handler block. If the print job has an error before spooling, this is
where it's handled. */
    void (^completionHandler)(UIPrintInteractionController *, BOOL, NSError *) =
^(UIPrintInteractionController *printController, BOOL completed, NSError *error) {
```

```

if(completed && error)
NSLog( @"打印失败，错误域为 %@，错误代码为 %lu。本地化描述：%@，失败原因：%@", error.domain, (long)error.code,
error.localizedDescription, error.localizedFailureReason );
};

if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad)
[controller presentFromRect:self.printButton.frame inView:self.view animated:YES
completionHandler:completionHandler];
else
[controller presentAnimated:YES completionHandler:completionHandler]; // iPhone
}

```

设置打印页面的代理函数：

```

- (CGFloat)printInteractionController:(UIPrintInteractionController *)printInteractionController
cutLengthForPaper:(UIPrintPaper *)paper {

/* 创建一个任意大小的字体，以便计算文本高度的每屏幕点大致对应的字体点数。 */
UIFont *font = _textFormatter.font;
CGSize size = [self.textField.text sizeWithAttributes:@{NSFontAttributeName: font}];

float approximateFontPointPerScreenPoint = font.pointSize / size.height;

/* 使用一个能填满纸张宽度的大小创建新字体 */
font = [self chosenFontWithSize: paper.printableRect.size.width *
approximateFontPointPerScreenPoint];

/* 使用最终字体大小计算文本的高度和宽度 */
CGSize finalTextSize = [self.textField.text sizeWithAttributes:@{NSFontAttributeName: font}];

/* 将UISimpleTextFormatter的字体设置为计算出的字体大小 */
_textFormatter.font = font;

/* 计算卷筒的边距。卷筒打印机在切割前后可能有不可打印区域。我们必须将其加到切割长度中
，以确保可打印区域有足够的空间容纳我们的文本。 */
CGFloat lengthOfMargins = paper.paperSize.height - paper.printableRect.size.height;

/* 切割长度是文本宽度，加上边距，再加上一些填充 */
return finalTextSize.width + lengthOfMargins + paper.printableRect.size.width * PaddingFactor;
}

```

```

if(completed && error)
NSLog( @"Printing failed due to error in domain %@ with error code %lu. Localized
description: %@", error.domain, (long)error.code,
error.localizedDescription, error.localizedFailureReason );
};

if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad)
[controller presentFromRect:self.printButton.frame inView:self.view animated:YES
completionHandler:completionHandler];
else
[controller presentAnimated:YES completionHandler:completionHandler]; // iPhone
}

```

The delegate function which sets up the print page:

```

- (CGFloat)printInteractionController:(UIPrintInteractionController *)printInteractionController
cutLengthForPaper:(UIPrintPaper *)paper {

/* Create a font with arbitrary size so that you can calculate the approximate
font points per screen point for the height of the text. */
UIFont *font = _textFormatter.font;
CGSize size = [self.textField.text sizeWithAttributes:@{NSFontAttributeName: font}];

float approximateFontPointPerScreenPoint = font.pointSize / size.height;

/* Create a new font using a size that will fill the width of the paper */
font = [self chosenFontWithSize: paper.printableRect.size.width *
approximateFontPointPerScreenPoint];

/* Calculate the height and width of the text with the final font size */
CGSize finalTextSize = [self.textField.text sizeWithAttributes:@{NSFontAttributeName: font}];

/* Set the UISimpleTextFormatter font to the font with the size calculated */
_textFormatter.font = font;

/* Calculate the margins of the roll. Roll printers may have unprintable areas
before and after the cut. We must add this to our cut length to ensure the
printable area has enough room for our text. */
CGFloat lengthOfMargins = paper.paperSize.height - paper.printableRect.size.height;

/* The cut length is the width of the text, plus margins, plus some padding */
return finalTextSize.width + lengthOfMargins + paper.printableRect.size.width * PaddingFactor;
}

```

第167章 : Carthage iOS 设置

第167.1节 : Mac 上的 Carthage 安装

Carthage 设置

从给定链接下载最新版本的 Carthage [下载链接](#)

在下载部分下载**Carthage.pkg**文件。

下载完成后，双击下载的 **pkg** 文件进行安装。

要检查是否下载成功，请在终端运行以下命令 `carthage version` 应显示安装的版本，如`0.18-19-g743fa0f`

Chapter 167: Carthage iOS Setup

Section 167.1: Carthage Installation Mac

Carthage Setup

Download the latest version of Carthage from the given link [Download Link](#)

Down in the Download section download the **Carthage.pkg** file.

Once the download is complete install it by double clinking on the download **pkg** file.

To check for successful download run the following command in your Terminal `carthage version` This should give the version installed like `0.18-19-g743fa0f`

第168章 : Healthkit

第168.1节 : HealthKit

Objective-C

首先进入Target->Capabilities并启用HealthKit。这将设置info.plist条目。

新建一个CocoaClass，类型为NSObject。我给的文件名是GSHealthKitManager，头文件如下所示

GSHealthKitManager.h

```
#import <Foundation/Foundation.h>
#import <HealthKit/HealthKit.h>
@interface GSHealthKitManager : NSObject

+ (GSHealthKitManager *)sharedManager;

- (void)requestAuthorization;

- (NSDate *)readBirthDate;
- (void)writeWeightSample:(double)weight;
- (NSString *)readGender;

@end
```

GSHealthKitManager.m

```
#import "GSHealthKitManager.h"
#import <HealthKit/HealthKit.h>

@interface GSHealthKitManager ()

@property (nonatomic, retain) HKHealthStore *healthStore;

@end
```

```
@implementation GSHealthKitManager

+ (GSHealthKitManager *)sharedManager {
    static dispatch_once_t pred = 0;
    static GSHealthKitManager *instance = nil;
    dispatch_once(&pred, ^{
        instance = [[GSHealthKitManager alloc] init];
        instance.healthStore = [[HKHealthStore alloc] init];
    });
    return instance;
}
```

```
- (void)requestAuthorization {

    if ([HKHealthStore isHealthDataAvailable] == NO) {
        // 如果我们的设备不支持HealthKit -> 返回。
        return;
    }
}
```

Chapter 168: Healthkit

Section 168.1: HealthKit

Objective-C

First go the Target->Capabilities and enable HealthKit. This would setup the info.plist entry.

Make a new CocoaClass of type `NSObject` The filename I gave is `GSHealthKitManager` and the header file is as shown below

GSHealthKitManager.h

```
#import <Foundation/Foundation.h>
#import <HealthKit/HealthKit.h>
@interface GSHealthKitManager : NSObject

+ (GSHealthKitManager *)sharedManager;

- (void)requestAuthorization;

- (NSDate *)readBirthDate;
- (void)writeWeightSample:(double)weight;
- (NSString *)readGender;

@end
```

GSHealthKitManager.m

```
#import "GSHealthKitManager.h"
#import <HealthKit/HealthKit.h>

@interface GSHealthKitManager ()

@property (nonatomic, retain) HKHealthStore *healthStore;

@end
```

```
@implementation GSHealthKitManager

+ (GSHealthKitManager *)sharedManager {
    static dispatch_once_t pred = 0;
    static GSHealthKitManager *instance = nil;
    dispatch_once(&pred, ^{
        instance = [[GSHealthKitManager alloc] init];
        instance.healthStore = [[HKHealthStore alloc] init];
    });
    return instance;
}
```

```
- (void)requestAuthorization {

    if ([HKHealthStore isHealthDataAvailable] == NO) {
        // If our device doesn't support HealthKit -> return.
        return;
    }
}
```

```
NSArray *readTypes = @[[HKObjectType  
characteristicTypeForIdentifier:HKCharacteristicTypeIdentifierDateOfBirth],[HKObjectType  
characteristicTypeForIdentifier:HKCharacteristicTypeIdentifierBiologicalSex]];
```

```
[self.healthStore requestAuthorizationToShareTypes:nil readTypes:[NSSet setWithArray:readTypes] completion:nil];  
}
```

```
- (NSDate *)readBirthDate {  
    NSError *error;  
    NSDate *dateOfBirth = [self.healthStore dateOfBirthWithError:&error]; // HKHealthStore 的便捷方法，直接获取出生日期。  
}
```

```
if (!dateOfBirth) {  
    NSLog(@"获取用户年龄信息时发生错误，或者尚未存储任何信息。在您的应用中，请尝试优雅地处理此情况。");  
}
```

```
return dateOfBirth;  
}
```

```
- (NSString *)readGender {  
    NSError *error;  
    HKBiologicalSexObject *gen=[self.healthStore biologicalSexWithError:&error];  
    if (gen.biologicalSex==HKBiologicalSexMale)  
    {  
        return(@"男");  
    }  
    else if (gen.biologicalSex==HKBiologicalSexFemale)  
    {  
        return (@"女");  
    }  
    else if (gen.biologicalSex==HKBiologicalSexOther)  
    {  
        return (@"其他");  
    }  
    else{  
        return (@"未设置");  
    }  
}
```

```
@end
```

从视图控制器调用

```
- (IBAction)pressed:(id)sender {  
  
    [[GSHealthKitManager sharedManager] requestAuthorization];  
    NSDate *birthDate = [[GSHealthKitManager sharedManager] readBirthDate];  
    NSLog(@"出生日期 %@", birthDate);  
    NSLog(@"gender 2131321 %@", [[GSHealthKitManager sharedManager] readGender]);  
  
}
```

日志输出

```
NSArray *readTypes = @[[HKObjectType  
characteristicTypeForIdentifier:HKCharacteristicTypeIdentifierDateOfBirth],[HKObjectType  
characteristicTypeForIdentifier:HKCharacteristicTypeIdentifierBiologicalSex]];
```

```
[self.healthStore requestAuthorizationToShareTypes:nil readTypes:[NSSet setWithArray:readTypes] completion:nil];  
}
```

```
- (NSDate *)readBirthDate {  
    NSError *error;  
    NSDate *dateOfBirth = [self.healthStore dateOfBirthWithError:&error]; // Convenience method of HKHealthStore to get date of birth directly.
```

```
if (!dateOfBirth) {  
    NSLog(@"Either an error occurred fetching the user's age information or none has been stored yet. In your app, try to handle this gracefully.");  
}
```

```
return dateOfBirth;  
}
```

```
- (NSString *)readGender {  
    NSError *error;  
    HKBiologicalSexObject *gen=[self.healthStore biologicalSexWithError:&error];  
    if (gen.biologicalSex==HKBiologicalSexMale)  
    {  
        return(@"Male");  
    }  
    else if (gen.biologicalSex==HKBiologicalSexFemale)  
    {  
        return (@"Female");  
    }  
    else if (gen.biologicalSex==HKBiologicalSexOther)  
    {  
        return (@"Other");  
    }  
    else{  
        return (@"Not Set");  
    }  
}
```

```
@end
```

Calling from ViewController

```
- (IBAction)pressed:(id)sender {  
  
    [[GSHealthKitManager sharedManager] requestAuthorization];  
    NSDate *birthDate = [[GSHealthKitManager sharedManager] readBirthDate];  
    NSLog(@"birthdate %@", birthDate);  
    NSLog(@"gender 2131321 %@", [[GSHealthKitManager sharedManager] readGender]);  
  
}
```

Log Output

2016-10-13 14:41:39.568 random[778:26371] 出生日期 1992-11-29 18:30:00 +0000
2016-10-13 14:41:39.570 random[778:26371] 性别 2131321 男性

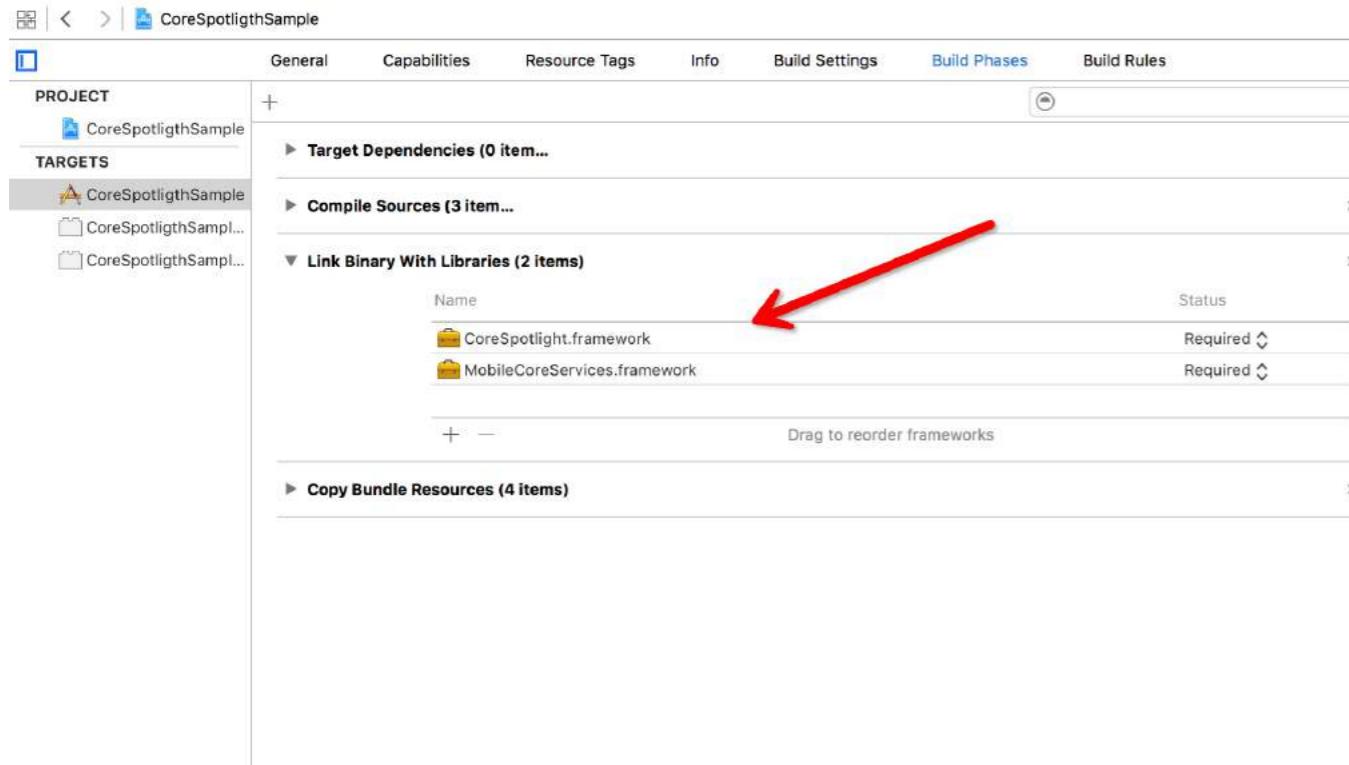
2016-10-13 14:41:39.568 random[778:26371] birthdate 1992-11-29 18:30:00 +0000
2016-10-13 14:41:39.570 random[778:26371] gender 2131321 Male

第169章：iOS中的核心聚焦（Core SpotLight）

第169.1节：核心聚焦（Core-Spotlight）

Objective-C

1. 创建一个新的iOS项目，并将CoreSpotlight和MobileCoreServices框架添加到项目中。



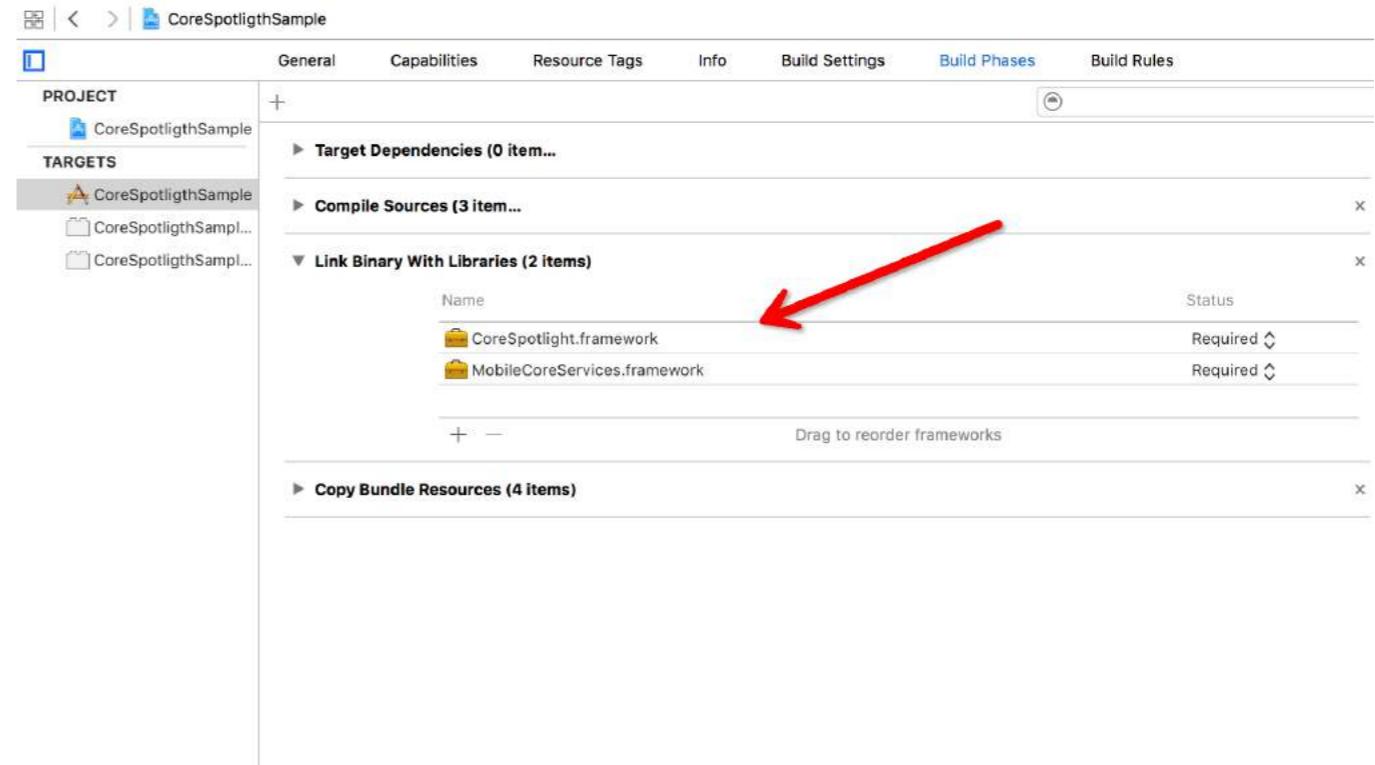
2. 创建实际的CSearchBarItem，并关联uniqueIdentifier、domainIdentifier和attributeSet。最后，使用[[CSearchBarIndex defaultSearchableIndex]...]对CSearchBarItem进行索引，如下所示。

Chapter 169: Core SpotLight in iOS

Section 169.1: Core-Spotlight

Objective-C

1. Create a new iOS project and add *CoreSpotlight* and *MobileCoreServices* framework to your project.



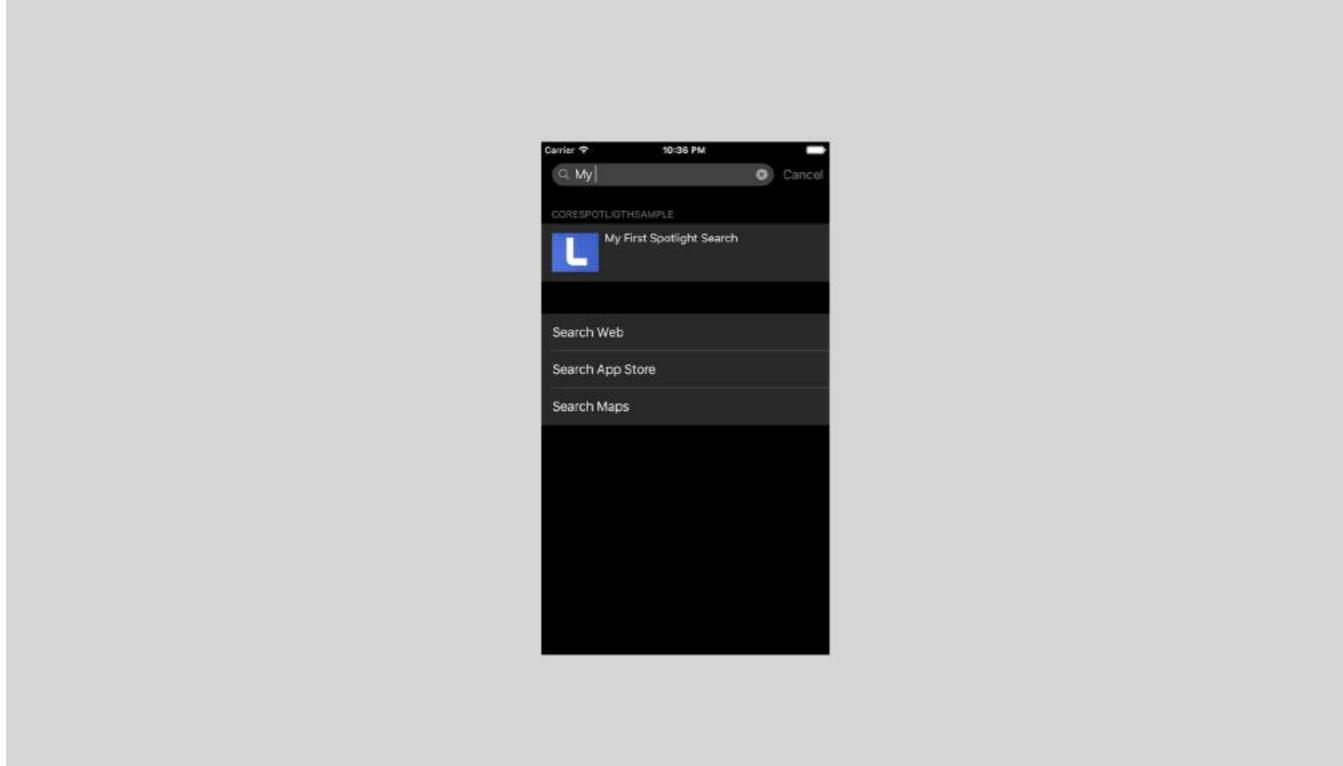
2. Create the actual CSearchBarItem and associating the uniqueIdentifier, domainIdentifier and the attributeSet. Finally index the CSearchBarItem using [[CSearchBarIndex defaultSearchableIndex]...] as show below.

```

4 // Created by Mayqiyue on 7/10/15.
5 // Copyright © 2015 mayqiyue. All rights reserved.
6 //
7 //
8
9 #import "ViewController.h"
10 #import <CoreSpotlight/CoreSpotlight.h>
11 #import <MobileCoreServices/MobileCoreServices.h>
12
13 @interface ViewController : UIViewController
14
15 @end
16
17 @implementation ViewController
18
19 - (void)viewDidLoad {
20     [super viewDidLoad];
21     [self setupCoreSpotlightSearch];
22 }
23
24 - (void)setupCoreSpotlightSearch {
25     CSSearchableItemAttributeSet *attributeSet = [[CSSearchableItemAttributeSet alloc] initWithItemContentType:
26         (NSString *)kUTTypeImage];
27     attributeSet.title = @"My First Spotlight Search";
28     attributeSet.contentDescription = @"";
29     attributeSet.keywords = [NSArray arrayWithObjects:@"Hello", @"Welcome", @"Spotlight", nil];
30     UIImage *image = [UIImage imageNamed:@"l"];
31     NSData *imageData = [NSData dataWithData:UIImagePNGRepresentation(image)];
32     attributeSet.thumbnailData = imageData;
33
34     CSSearchableItem *item = [[CSSearchableItem alloc] initWithUniqueId:@"com.deeplink"
35         domainIdentifier:@"spotlight.sample" attributeSet:attributeSet];
36
37     [[CSSearchableIndex defaultSearchableIndex] indexSearchableItems:@[item] completionHandler: ^(NSError * _Nullable
38         error) {
39         if (!error)
40             NSLog(@"Search item indexed");
41     }];
42 }
43 - (void)didReceiveMemoryWarning {
44     [super didReceiveMemoryWarning];
45 }

```

3. 好的！测试索引！

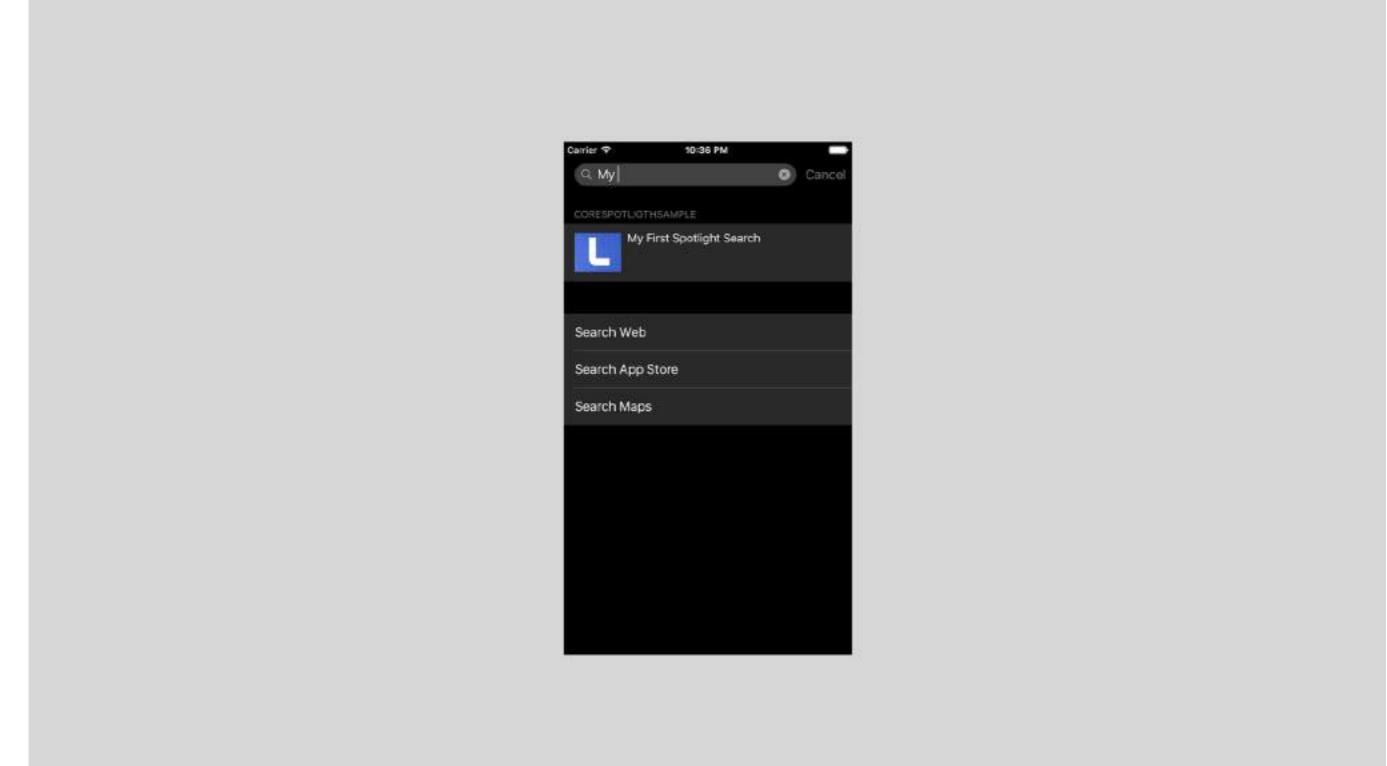


```

4 // Created by Mayqiyue on 7/10/15.
5 // Copyright © 2015 mayqiyue. All rights reserved.
6 //
7 //
8
9 #import "ViewController.h"
10 #import <CoreSpotlight/CoreSpotlight.h>
11 #import <MobileCoreServices/MobileCoreServices.h>
12
13 @interface ViewController : UIViewController
14
15 @end
16
17 @implementation ViewController
18
19 - (void)viewDidLoad {
20     [super viewDidLoad];
21     [self setupCoreSpotlightSearch];
22 }
23
24 - (void)setupCoreSpotlightSearch {
25     CSSearchableItemAttributeSet *attributeSet = [[CSSearchableItemAttributeSet alloc] initWithItemContentType:
26         (NSString *)kUTTypeImage];
27     attributeSet.title = @"My First Spotlight Search";
28     attributeSet.contentDescription = @"";
29     attributeSet.keywords = [NSArray arrayWithObjects:@"Hello", @"Welcome", @"Spotlight", nil];
30     UIImage *image = [UIImage imageNamed:@"l"];
31     NSData *imageData = [NSData dataWithData:UIImagePNGRepresentation(image)];
32     attributeSet.thumbnailData = imageData;
33
34     CSSearchableItem *item = [[CSSearchableItem alloc] initWithUniqueId:@"com.deeplink"
35         domainIdentifier:@"spotlight.sample" attributeSet:attributeSet];
36
37     [[CSSearchableIndex defaultSearchableIndex] indexSearchableItems:@[item] completionHandler: ^(NSError * _Nullable
38         error) {
39         if (!error)
40             NSLog(@"Search item indexed");
41     }];
42 }
43 - (void)didReceiveMemoryWarning {
44     [super didReceiveMemoryWarning];
45 }

```

3. OK! Test the index!



第170章：核心运动

第170.1节：访问气压计以获取相对高度

Swift

导入Core Motion库：

```
import CoreMotion
```

接下来，我们需要创建一个CMAltimeter对象，但一个常见的错误是在viewDidLoad()中创建它。如果那样做，气压计在我们需要调用其方法时将无法访问。不过，先在viewDidLoad()之前创建你的CMAltimeter对象：

```
let altimeter = CMAltimeter()
```

现在：

1. 我们需要使用以下方法检查relativeAltitude是否可用：
`CMAltimeter.isRelativeAltitudeAvailable.`
2. 如果返回true，则可以开始使用
`startRelativeAltitudeUpdatesToQueue`
3. 如果没有错误，您应该能够从relativeAltitude和压力属性中获取数据。

下面是一个按钮动作的定义，用于开始使用我们的气压计进行监测。

```
@IBAction func start(sender: AnyObject){  
if CMAltimeter.isRelativeAltitudeAvailable() {  
    // 1  
    altimeter.startRelativeAltitudeUpdatesToQueue(NSOperationQueue.mainQueue(), withHandler: {  
        data, error in  
            // 2  
            if (error == nil) {  
                println("相对高度: \(data.relativeAltitude)")  
                println("压力: \(data.pressure)")  
            }  
    })  
}
```

Chapter 170: Core Motion

Section 170.1: Accessing Barometer to get relative altitude

Swift

Import the Core Motion library:

```
import CoreMotion
```

Next, we need to create a CMAltimeter object, but a common pitfall is to create it in the viewDidLoad(). If done that way, the altimeter won't be accessible when we need to call a method on it. Nevertheless, go ahead and create your CMAltimeter object just before the viewDidLoad():

```
let altimeter = CMAltimeter()
```

Now:

1. We need to check if relativeAltitude is even available with the following method:
`CMAltimeter.isRelativeAltitudeAvailable.`
2. If that returns true, you can then begin monitoring altitude change with
`startRelativeAltitudeUpdatesToQueue`
3. If there are no errors, you should be able to retrieve data from the relativeAltitude and pressure properties.

Given below is the definition of a button action to begin monitoring with our barometer.

```
@IBAction func start(sender: AnyObject){  
if CMAltimeter.isRelativeAltitudeAvailable() {  
    // 1  
    altimeter.startRelativeAltitudeUpdatesToQueue(NSOperationQueue.mainQueue(), withHandler: {  
        data, error in  
            // 2  
            if (error == nil) {  
                println("Relative Altitude: \(data.relativeAltitude)")  
                println("Pressure: \(data.pressure)")  
            }  
    })  
}
```

第171章：二维码扫描器

二维码（Quick Response，快速响应）码是广泛应用于机器可读光学标签的二维条码。iOS从7版本开始提供了使用AVFoundation框架读取二维码的方法。该框架提供了一套API，用于设置/打开摄像头并从摄像头视频流中读取二维码。

第171.1节：使用AVFoundation框架扫描二维码

在iOS 7之前，如果想扫描二维码，可能需要依赖第三方框架或库，如zBar或zXing。但苹果从iOS 7开始引入了AVCaptureMetadataOutput用于读取条形码。

要使用AVFoundation读取二维码，需要设置/创建AVCaptureSession并使用captureOutput:didOutputMetadataObjects:fromConnection:代理方法。

步骤 1

导入AVFoundation框架并遵循AVCaptureMetadataOutputObjectsDelegate协议

```
import AVFoundation  
class ViewController: UIViewController, AVCaptureMetadataOutputObjectsDelegate
```

步骤2

二维码读取完全基于视频捕捉。因此，为了捕捉连续视频，需要创建一个AVCaptureSession并设置设备输入和输出。在视图控制器的viewDidLoad方法中添加以下代码

```
// 创建一个 AVCaptureDevice 实例，并将视频作为媒体类型参数提供。  
let captureDevice = AVCaptureDevice.defaultDevice(withMediaType: AVMediaTypeVideo)  
  
do {  
    // 使用设备对象创建 AVCaptureDeviceInput 类的实例并初始化  
    // 捕获会话  
    let input = try AVCaptureDeviceInput(device: captureDevice)  
    captureSession = AVCaptureSession()  
    captureSession?.addInput(input)  
  
    // 创建 AVCaptureMetadataOutput 对象的实例并将其设置为捕获会话的输出设备。  
  
    let captureMetadataOutput = AVCaptureMetadataOutput()  
    captureSession?.addOutput(captureMetadataOutput)  
    // 使用默认的调度队列设置代理  
    captureMetadataOutput.setMetadataObjectsDelegate(self, queue: DispatchQueue.main)  
    // 设置元数据对象类型为二维码，这里也可以添加多种类型  
    captureMetadataOutput.metadataObjectTypes = [AVMetadataObjectTypeQRCode]  
  
    // 初始化视频预览图层并将其作为子图层添加到视图控制器视图的图层中。  
  
    videoPreviewLayer = AVCaptureVideoPreviewLayer(session: captureSession)  
    videoPreviewLayer?.videoGravity = AVLayerVideoGravityResizeAspectFill  
    videoPreviewLayer?.frame = view.layer.bounds  
    view.layer.addSublayer(videoPreviewLayer!)  
  
    // 启动捕获会话。  
    captureSession?.startRunning()  
} catch {  
    // 如果发生任何错误，通知用户。示例中仅打印错误信息  
}
```

Chapter 171: QR Code Scanner

QR (Quick Response) codes are two-dimensional barcodes which are widely used on machine-readable optical labels. iOS do provide a way to read the QR codes by using AVFoundation framework from iOS 7 onwards. This framework provides set of API's to setup/open the camera and read QR codes from the camera feed.

Section 171.1: Scanning QR code with AVFoudation framework

Prior to iOS 7 when you want to scan a QR code, we might need to rely on third party frameworks or libraries like zBar or zXing. But Apple introduced AVCaptureMetadataOutput from iOS 7 for reading barcodes.

To read QR code using AVFoundation we need to setup/create AVCaptureSession and use captureOutput:didOutputMetadataObjects:fromConnection: delegate method.

Step 1

Import AVFoundation framework and confirm to AVCaptureMetadataOutputObjectsDelegate protocol

```
import AVFoundation  
class ViewController: UIViewController, AVCaptureMetadataOutputObjectsDelegate
```

Step 2

QR code reading is totally based on video capture. So to capture continuous video create an AVCaptureSession and set up device input and output. Add the below code in view controller viewDidLoad method

```
// Create an instance of the AVCaptureDevice and provide the video as the media type parameter.  
let captureDevice = AVCaptureDevice.defaultDevice(withMediaType: AVMediaTypeVideo)  
  
do {  
    // Create an instance of the AVCaptureDeviceInput class using the device object and intialise  
    // capture session  
    let input = try AVCaptureDeviceInput(device: captureDevice)  
    captureSession = AVCaptureSession()  
    captureSession?.addInput(input)  
  
    // Create a instance of AVCaptureMetadataOutput object and set it as the output device the  
    // capture session.  
    let captureMetadataOutput = AVCaptureMetadataOutput()  
    captureSession?.addOutput(captureMetadataOutput)  
    // Set delegate with a default dispatch queue  
    captureMetadataOutput.setMetadataObjectsDelegate(self, queue: DispatchQueue.main)  
    //set meta data object type as QR code, here we can add more then one type as well  
    captureMetadataOutput.metadataObjectTypes = [AVMetadataObjectTypeQRCode]  
  
    // Initialize the video preview layer and add it as a sublayer to the viewcontroller view's  
    // layer.  
    videoPreviewLayer = AVCaptureVideoPreviewLayer(session: captureSession)  
    videoPreviewLayer?.videoGravity = AVLayerVideoGravityResizeAspectFill  
    videoPreviewLayer?.frame = view.layer.bounds  
    view.layer.addSublayer(videoPreviewLayer!)  
  
    // Start capture session.  
    captureSession?.startRunning()  
} catch {  
    // If any error occurs, let the user know. For the example purpose just print out the error  
}
```

```
print(error)
return
}
```

步骤 3

实现 AVCaptureMetadataOutputObjectsDelegate 委托方法以读取二维码

```
func captureOutput(_ captureOutput: AVCaptureOutput!, didOutputMetadataObjects metadataObjects: [Any]!, from connection: AVCaptureConnection!) {

    // 检查 metadataObjects 数组是否至少包含一个对象。如果没有，则视频捕获中没有二维码

    if metadataObjects == nil || metadataObjects.count == 0 {
        // 未检测到二维码。
        return
    }

    // 获取元数据对象并将其转换为 `AVMetadataMachineReadableCodeObject`
    let metadataObj = metadataObjects[0] as! AVMetadataMachineReadableCodeObject

    if metadataObj.type == AVMetadataObjectTypeQRCode {
        // 如果找到的元数据等于二维码元数据，则从元数据中获取字符串值

        let barCodeObject = videoPreviewLayer?.transformedMetadataObject(for: metadataObj)

        if metadataObj.stringValue != nil {
            // metadataObj.stringValue 是我们的二维码
        }
    }
}
```

这里元数据对象还可以为你提供摄像头画面中读取到的二维码的边界。要获取边界，只需将元数据对象传递给videoPreviewLayer的transformedMetadataObject方法，如下所示。

```
let barCodeObject = videoPreviewLayer?.transformedMetadataObject(for: metadataObj)
qrCodeFrameView?.frame = barCodeObject!.bounds
```

第171.2节：UIViewController 扫描二维码并显示视频输入

```
import AVFoundation
class QRScannerViewController: UIViewController,
    AVCaptureMetadataOutputObjectsDelegate {

    func viewDidLoad() {
        self.initCaptureSession()
    }

    private func initCaptureSession() {
        let captureDevice = AVCaptureDevice
            .defaultDevice(withMediaType: AVMediaTypeVideo)
        do {
            let input = try AVCaptureDeviceInput(device: captureDevice)
            let captureMetadataOutput = AVCaptureMetadataOutput()
            self.captureSession?.addOutput(captureMetadataOutput)
            captureMetadataOutput.setMetadataObjectsDelegate(self,
                queue: DispatchQueue.main)
            captureMetadataOutput
        .metadataObjectTypes = [AVMetadataObjectTypeQRCode]
```

```
print(error)
return
}
```

Step 3

Implement AVCaptureMetadataOutputObjectsDelegate delegate method to read the QR code

```
func captureOutput(_ captureOutput: AVCaptureOutput!, didOutputMetadataObjects metadataObjects: [Any]!, from connection: AVCaptureConnection!) {

    // Check if the metadataObjects array contains at least one object. If not no QR code is in our
    // video capture
    if metadataObjects == nil || metadataObjects.count == 0 {
        // NO QR code is being detected.
        return
    }

    // Get the metadata object and cast it to `AVMetadataMachineReadableCodeObject`
    let metadataObj = metadataObjects[0] as! AVMetadataMachineReadableCodeObject

    if metadataObj.type == AVMetadataObjectTypeQRCode {
        // If the found metadata is equal to the QR code metadata then get the string value from
        // meta data
        let barCodeObject = videoPreviewLayer?.transformedMetadataObject(for: metadataObj)

        if metadataObj.stringValue != nil {
            // metadataObj.stringValue is our QR code
        }
    }
}
```

here metadata object can give you the bounds of the QR code read on the camera feed as well. To get the bounds simply pass the metadata object to videoPreviewLayer's transformedMetadataObject method like below.

```
let barCodeObject = videoPreviewLayer?.transformedMetadataObject(for: metadataObj)
qrCodeFrameView?.frame = barCodeObject!.bounds
```

Section 171.2: UIViewController scanning for QR and displaying video input

```
import AVFoundation
class QRScannerViewController: UIViewController,
    AVCaptureMetadataOutputObjectsDelegate {

    func viewDidLoad() {
        self.initCaptureSession()
    }

    private func initCaptureSession() {
        let captureDevice = AVCaptureDevice
            .defaultDevice(withMediaType: AVMediaTypeVideo)
        do {
            let input = try AVCaptureDeviceInput(device: captureDevice)
            let captureMetadataOutput = AVCaptureMetadataOutput()
            self.captureSession?.addOutput(captureMetadataOutput)
            captureMetadataOutput.setMetadataObjectsDelegate(self,
                queue: DispatchQueue.main)
            captureMetadataOutput
        .metadataObjectTypes = [AVMetadataObjectTypeQRCode]
```

```

        self.videoPreviewLayer =
AVCaptureVideoPreviewLayer(session: self.captureSession)
        self.videoPreviewLayer?
.videoGravity = AVLayerVideoGravityResizeAspectFill
        self.videoPreviewLayer?.frame =
            self.view.layer.bounds

        self._viewController?.view.layer
.addSublayer(videoPreviewLayer!)
        self.captureSession?.startRunning()
    } catch {
        //TODO: 处理输入打开错误
    }
}

private func 关闭捕获会话() {
    if let running = self.captureSession?.isRunning, running {
        self.captureSession?.stopRunning()
    }
    self.captureSession = nil
    self.videoPreviewLayer?.removeFromSuperLayer()
    self.videoPreviewLayer = nil
}

func captureOutput(_ captureOutput: AVCaptureOutput,
    didOutputMetadataObjects metadataObjects: [Any]!,
    from connection: AVCaptureConnection) {
guard metadataObjects != nil && metadataObjects.count != 0 else {
    //未捕获到任何内容
    返回
}

if let metadataObj =
metadataObjects[0] as? AVMetadataMachineReadableCodeObject {
    guard metadataObj.type == AVMetadataObjectTypeQRCode else {
        return
    }

    let barCodeObject = videoPreviewLayer?
.transformedMetadataObject(for:
metadataObj as AVMetadataMachineReadableCodeObject)
        as! AVMetadataMachineReadableCodeObject

    if let qrValue = metadataObj.stringValue {
        self.handleQRRead(value: qrValue)
    }
}

private handleQRRead(value: String) {
    //TODO: 处理读取的二维码
}
private captureSession: AVCaptureSession?
private videoPreviewLayer: AVCaptureVideo
}

```

handleQRRead - 在扫描成功时调用 initCaptureSession - 初始化二维码扫描和摄像头输入
dismissCaptureSession - 隐藏摄像头输入并停止扫描

```

        self.videoPreviewLayer =
AVCaptureVideoPreviewLayer(session: self.captureSession)
        self.videoPreviewLayer?
.videoGravity = AVLayerVideoGravityResizeAspectFill
        self.videoPreviewLayer?.frame =
            self.view.layer.bounds

        self._viewController?.view.layer
.addSublayer(videoPreviewLayer!)
        self.captureSession?.startRunning()
    } catch {
        //TODO: handle input open error
    }
}

private func dismissCaptureSession() {
    if let running = self.captureSession?.isRunning, running {
        self.captureSession?.stopRunning()
    }
    self.captureSession = nil
    self.videoPreviewLayer?.removeFromSuperLayer()
    self.videoPreviewLayer = nil
}

func captureOutput(_ captureOutput: AVCaptureOutput,
    didOutputMetadataObjects metadataObjects: [Any]!,
    from connection: AVCaptureConnection) {
guard metadataObjects != nil && metadataObjects.count != 0 else {
    //Nothing captured
    return
}

if let metadataObj =
metadataObjects[0] as? AVMetadataMachineReadableCodeObject {
    guard metadataObj.type == AVMetadataObjectTypeQRCode else {
        return
    }

    let barCodeObject = videoPreviewLayer?
.transformedMetadataObject(for:
metadataObj as AVMetadataMachineReadableCodeObject)
        as! AVMetadataMachineReadableCodeObject

    if let qrValue = metadataObj.stringValue {
        self.handleQRRead(value: qrValue)
    }
}

private handleQRRead(value: String) {
    //TODO: Handle the read qr
}
private captureSession: AVCaptureSession?
private videoPreviewLayer: AVCaptureVideo
}

```

handleQRRead - will be called on a successful scan initCaptureSession - initialize scanning for QR and camera input
dismissCaptureSession - hide the camera input and stop scanning

第172章：plist iOS

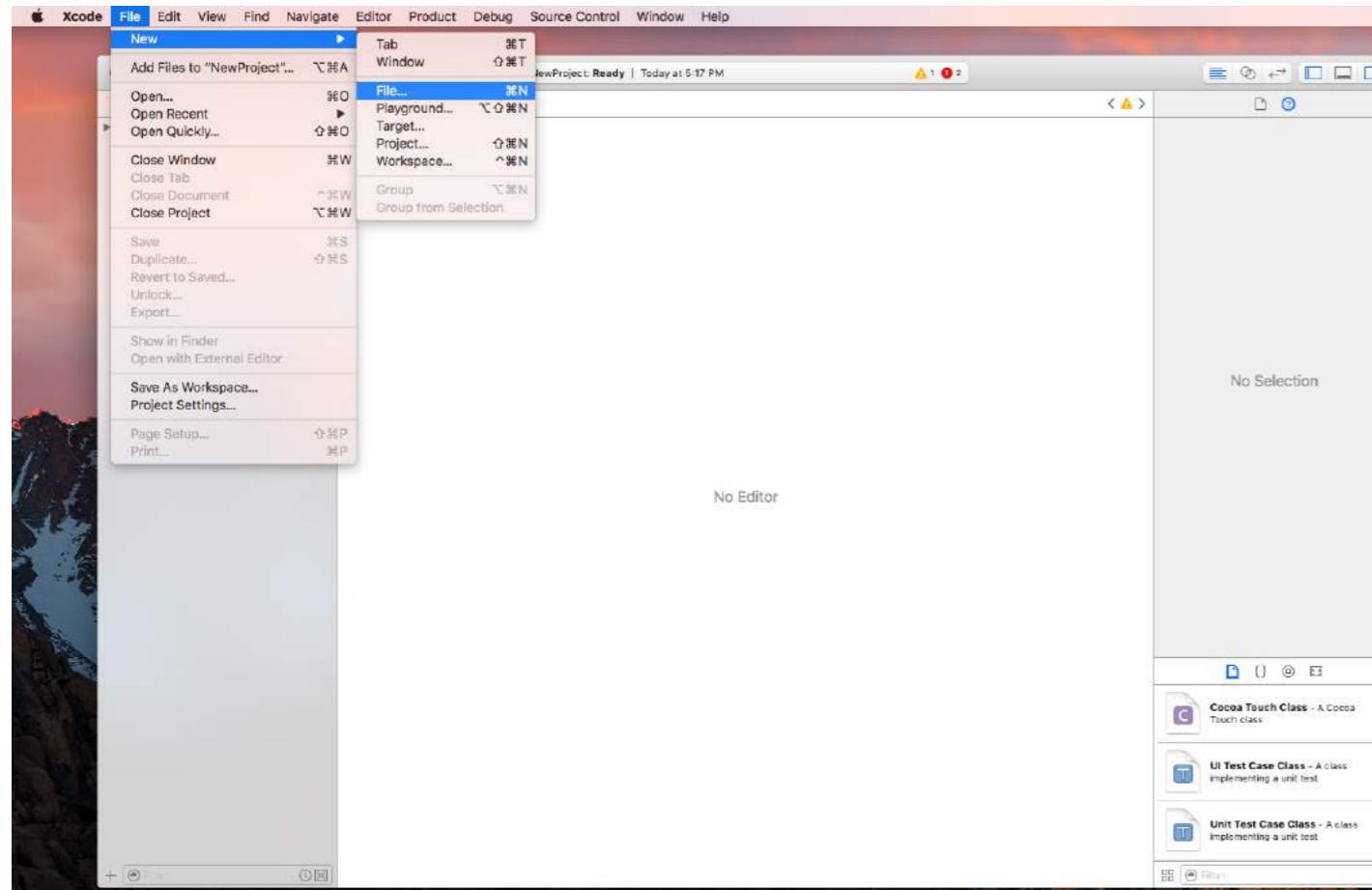
Plist用于iOS应用中的数据存储。Plist以数组和字典的形式保存数据。在plist中我们可以保存数据为：
1. 应用中使用的静态数据。2. 来自服务器的数据。

第172.1节：示例：

1. 应用中使用的静态数据。

保存静态数据到plist请遵循以下方法：

a) 添加一个新文件



b) 点击资源中的属性列表 (Property list)

Chapter 172: plist iOS

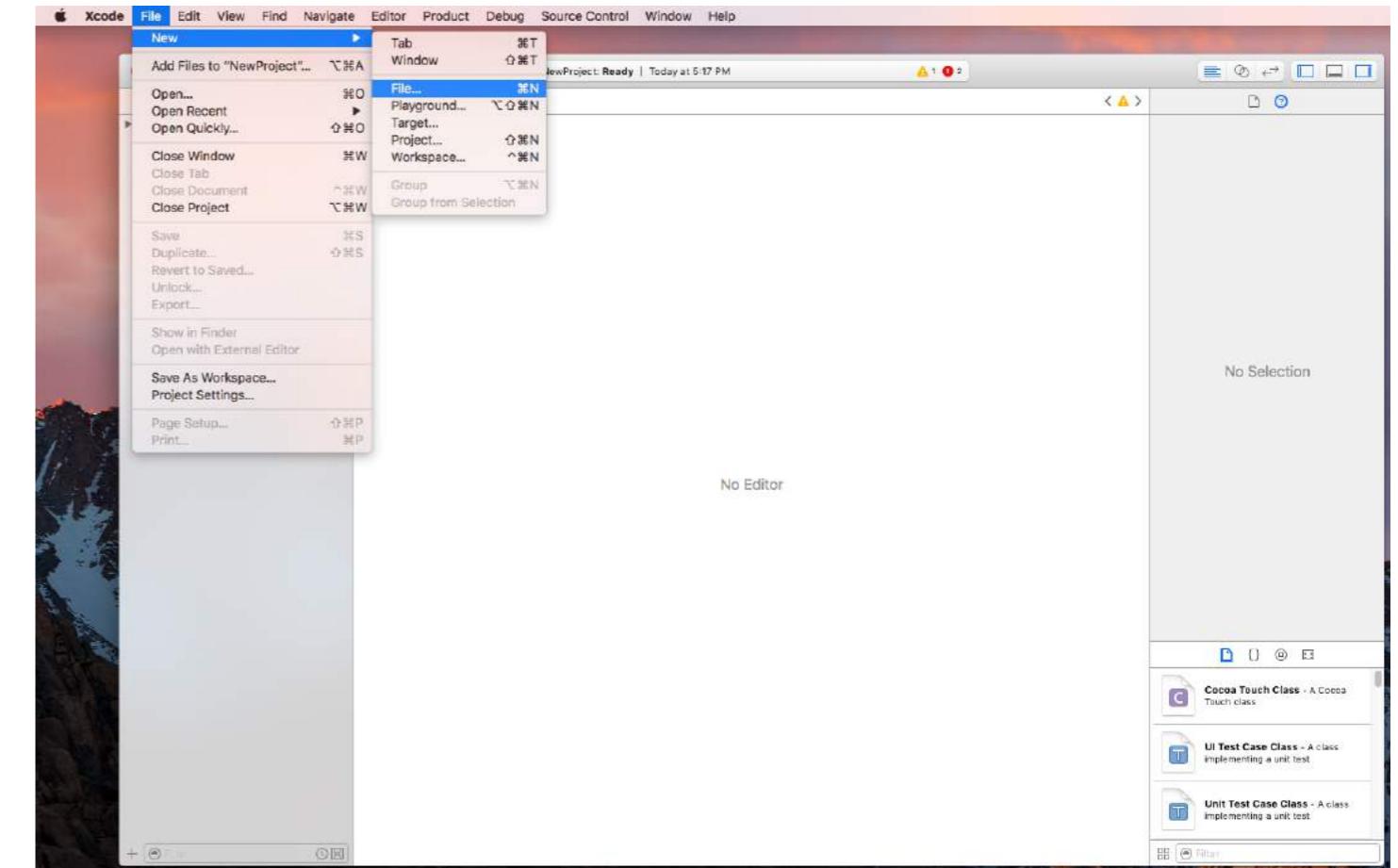
Plist is used for storage of data in iOS app. Plist save data in form of Array and Dictionaries. In plist we can save data as: 1. Static data to be used in app. 2. Data that will be coming from server.

Section 172.1: Example:

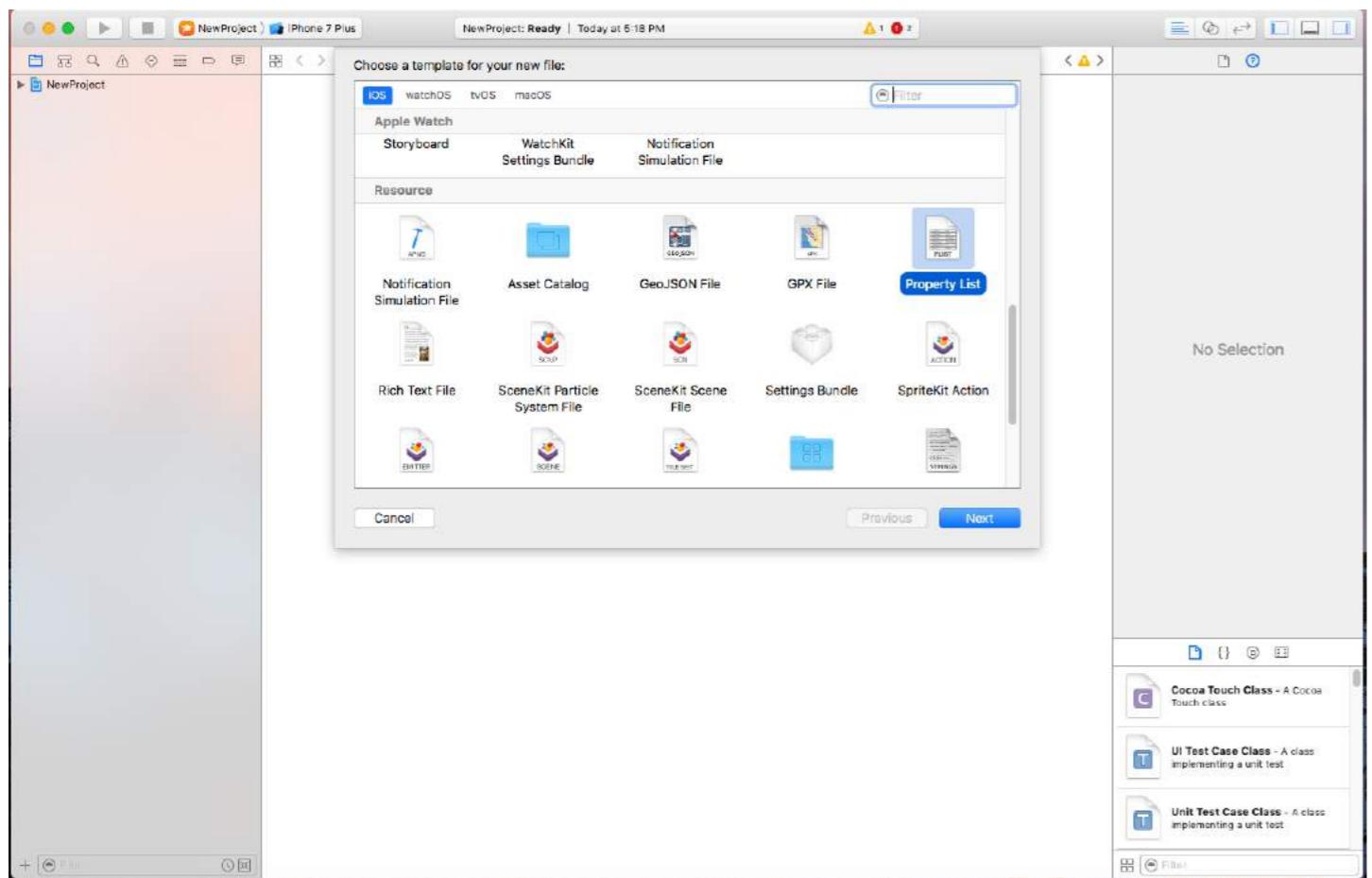
1. Static data to be used in app.

To save static data in plist follow these methods:

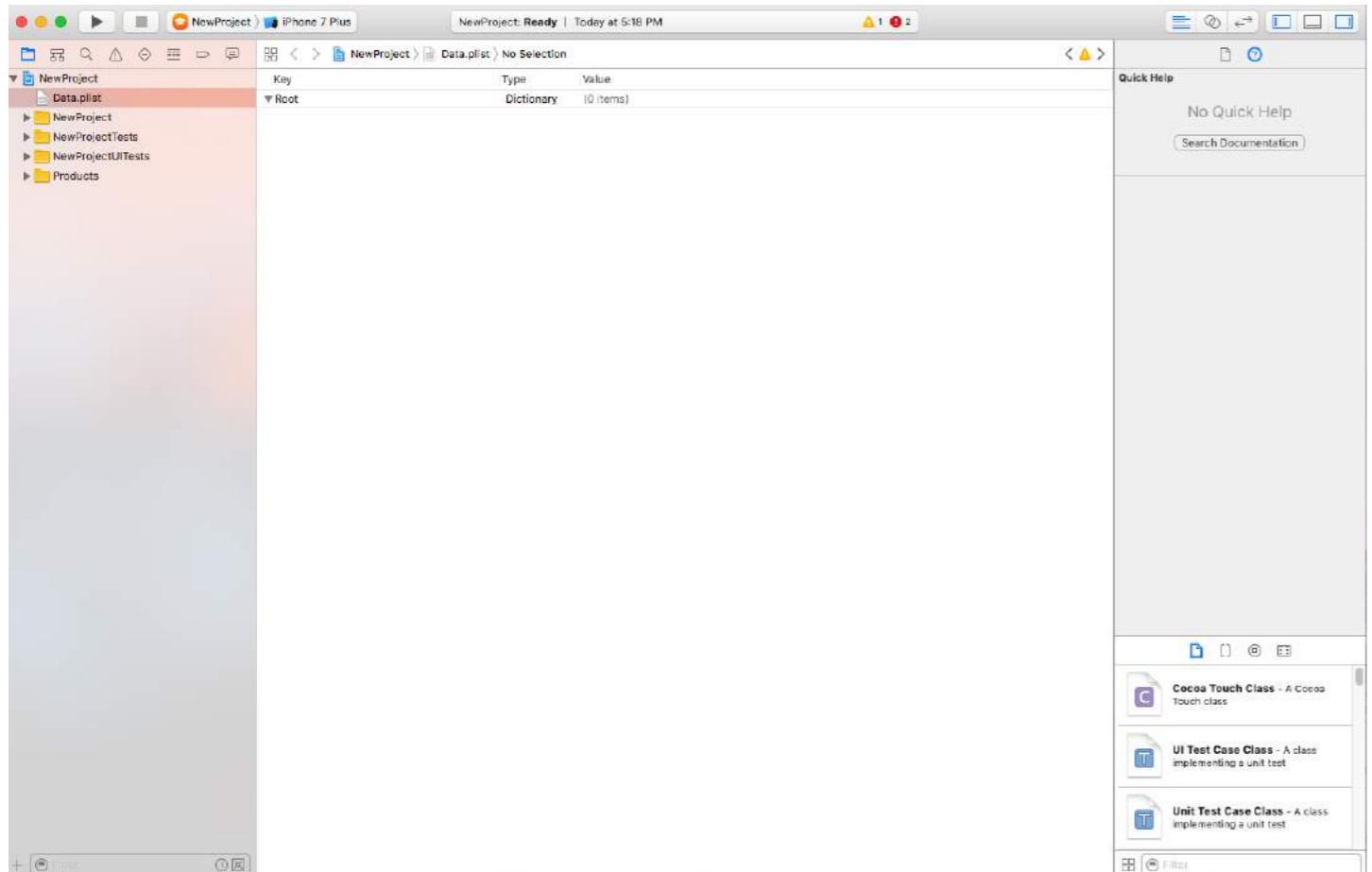
a) Add a new file



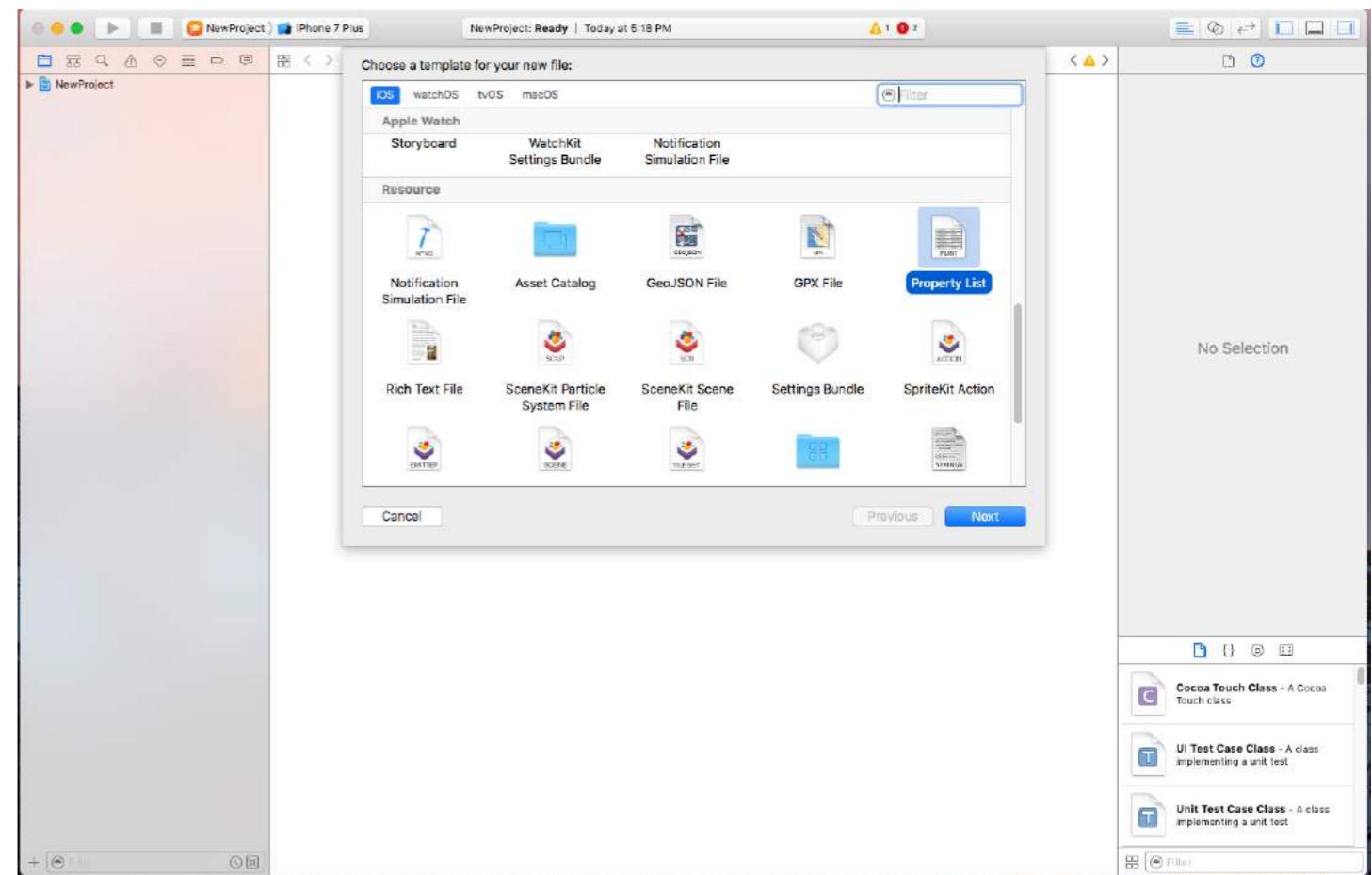
b) Click Property list in Resources



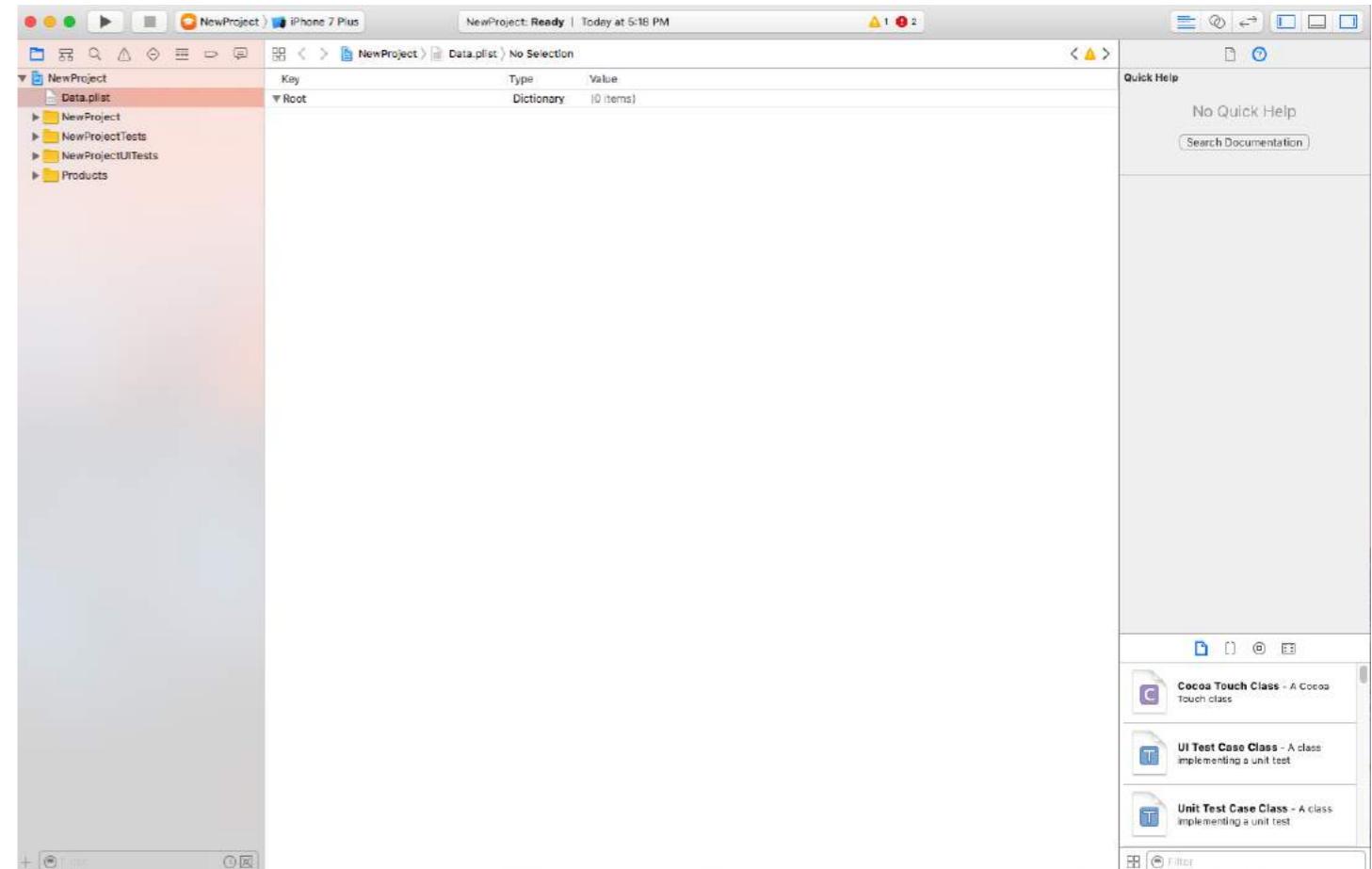
c) 命名属性列表，系统将创建一个文件（此处为data.plist）



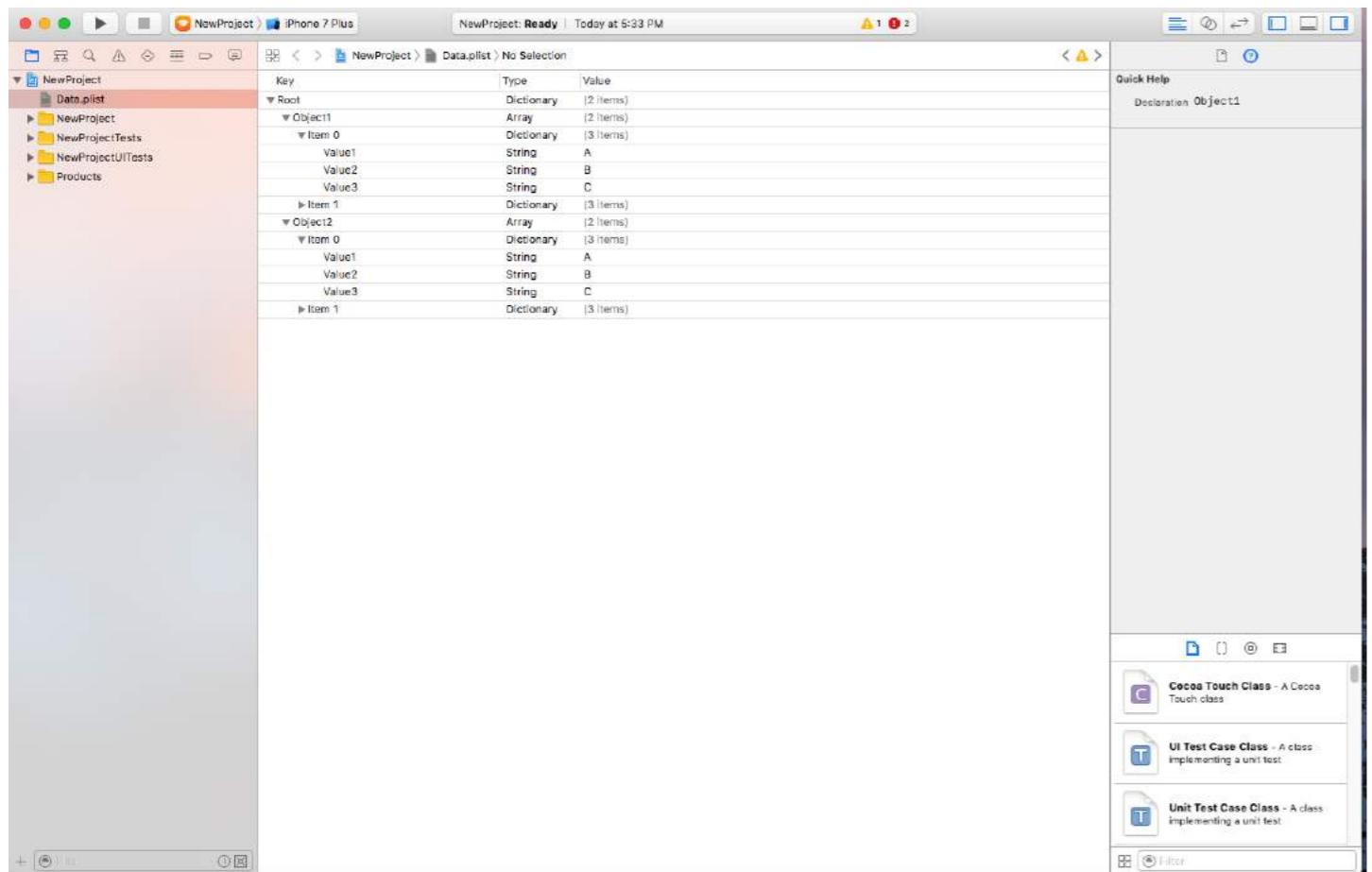
d) 你可以创建包含数组和字典的plist，如下：



c) Name the propertylist and a file will be created as(data.plist here)



d) You can create a plist of Arrays and Dictionaries as:



// 从bundle中读取plist并获取根字典

```
NSDictionary *dictRoot = [NSDictionary dictionaryWithContentsOfFile:[[NSBundle mainBundle] pathForResource:@"Data" ofType:@"plist"]];
```

// 你的字典包含一个字典数组 // 现在从中取出一个数组。

```
NSArray *arrayList = [NSArray arrayWithArray:[dictRoot objectForKey:@"Object1"]];

for(int i=0; i< [arrayList count]; i++)
{
    NSMutableDictionary *details=[arrayList objectAtIndex:i];
}
```

第172.2节：从Plist保存和编辑/删除数据

您已经创建了一个plist。该plist在应用中将保持不变。如果您想编辑此plist中的数据，添加新的数据或从plist中删除数据，您无法直接修改此文件。

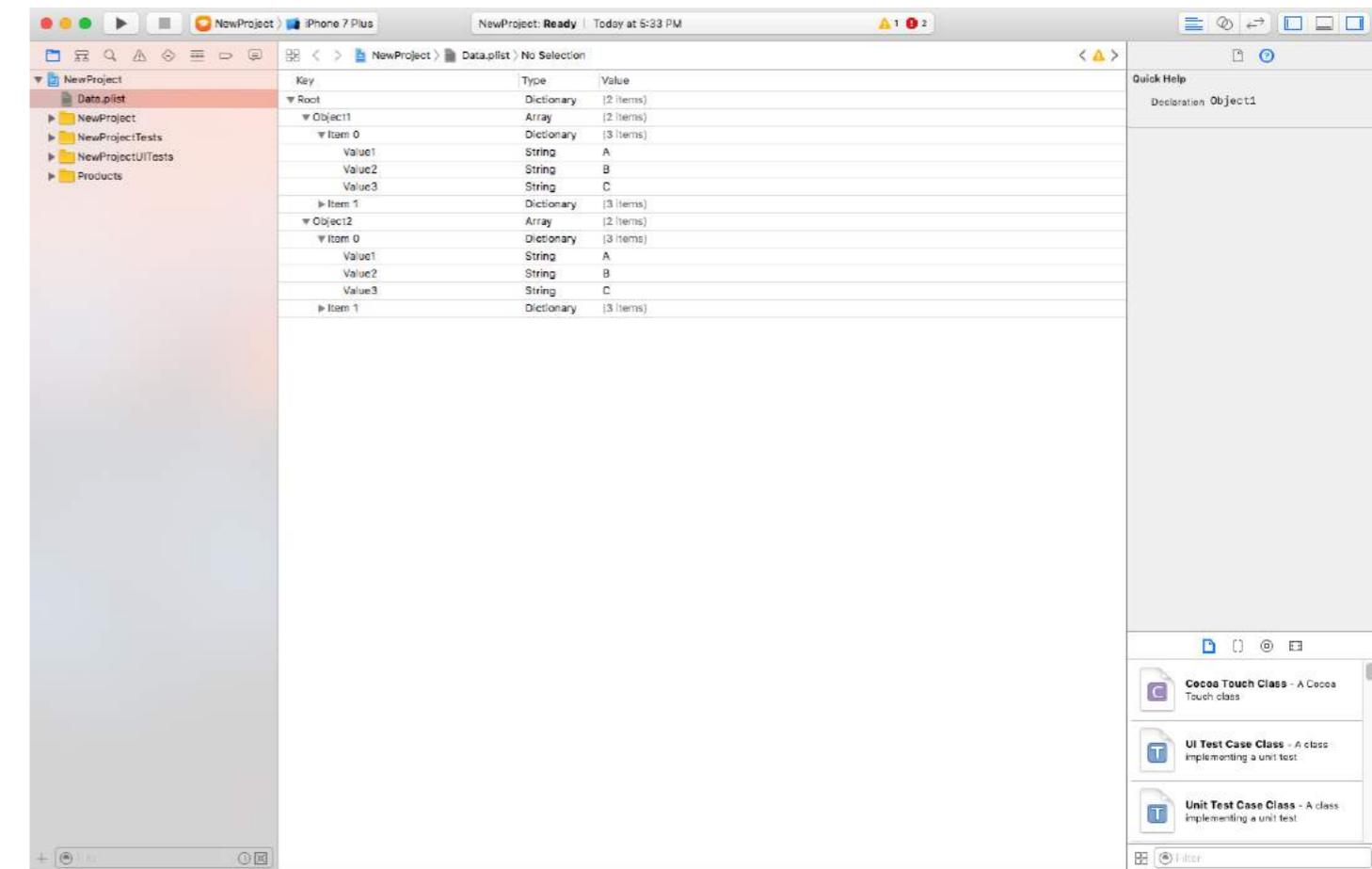
为此，您需要将plist存储在文档目录中。您可以编辑保存在文档目录中的plist。

将plist保存到文档目录，如下所示：

```
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"Data" ofType:@"plist"];

NSDictionary *dict = [[NSDictionary alloc] initWithContentsOfFile:filePath];

NSDictionary *plistDict = dict;
```



// Read plist from bundle and get Root Dictionary out of it

```
NSDictionary *dictRoot = [NSDictionary dictionaryWithContentsOfFile:[[NSBundle mainBundle] pathForResource:@"Data" ofType:@"plist"]];
```

// Your dictionary contains an array of dictionary // Now pull an Array out of it.

```
NSArray *arrayList = [NSArray arrayWithArray:[dictRoot objectForKey:@"Object1"]];

for(int i=0; i< [arrayList count]; i++)
{
    NSMutableDictionary *details=[arrayList objectAtIndex:i];
}
```

Section 172.2: Save and edit/delete data from Plist

You have already created a plist. This plist will remain same in app. If you want to edit the data in this plist, add new data in plist or remove data from plist, you can't make changes in this file.

For this purpose you will have to store your plist in Document Directory. You can edit your plist saved in document directory.

Save plist in document directory as:

```
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"Data" ofType:@"plist"];

NSDictionary *dict = [[NSDictionary alloc] initWithContentsOfFile:filePath];

NSDictionary *plistDict = dict;
```

```

NSFileManager *fileManager = [NSFileManager defaultManager];
NSString *error = nil;

NSData *plistData = [NSPropertyListSerialization dataFromPropertyList:plistDict
format:NSPropertyListXMLFormat_v1_0 errorDescription:&error];

if (![fileManager fileExistsAtPath: plistPath]) {

    if(plistData)
    {
        [plistData writeToFile:plistPath atomically:YES];
    }
} else
{
}

```

从 Plist 中检索数据如下：

```

NSArray *paths = NSSearchPathForDirectoriesInDomains (NSDocumentDirectory, NSUserDomainMask,
YES);
NSString *documentsPath = [paths objectAtIndex:0];
NSString *plistPath = [documentsPath stringByAppendingPathComponent:@"Data.plist"];
NSDictionary *dict = [[NSDictionary alloc] initWithContentsOfFile:plistPath];

NSArray *usersArray = [dict objectForKey:@"Object1"];

```

您可以根据需要编辑、删除、添加新数据，然后将 plist 重新保存到文档目录。

```

NSFileManager *fileManager = [NSFileManager defaultManager];
NSString *error = nil;

NSData *plistData = [NSPropertyListSerialization dataFromPropertyList:plistDict
format:NSPropertyListXMLFormat_v1_0 errorDescription:&error];

if (![fileManager fileExistsAtPath: plistPath]) {

    if(plistData)
    {
        [plistData writeToFile:plistPath atomically:YES];
    }
} else
{
}

```

Retrieve data from Plist as:

```

NSArray *paths = NSSearchPathForDirectoriesInDomains (NSDocumentDirectory, NSUserDomainMask,
YES);
NSString *documentsPath = [paths objectAtIndex:0];
NSString *plistPath = [documentsPath stringByAppendingPathComponent:@"Data.plist"];
NSDictionary *dict = [[NSDictionary alloc] initWithContentsOfFile:plistPath];

NSArray *usersArray = [dict objectForKey:@"Object1"];

```

You can edit remove, add new data as per your requirement and save the plist again to Document Directory.

第173章 : WCSessionDelegate

WCSessionDelegate 适用于使用 WatchConnectivity 的 watch OS2 及以上版本。var watchSession : WCSession? func startWatchSession(){ if(WCSession.isSupported()){ watchSession = WCSession.default() watchSession!.delegate = self watchSession!.activate() } } 实现必需的方法：-didReceiveApplicationContext

第173.1节 : Watch kit 控制器 (WKInterfaceController)

```
import WatchConnectivity

var watchSession : WCSession?

override func awake(withContext context: Any?) {
    super.awake(withContext: context)
    // 在此配置界面对象。
startWatchSession()
}

func startWatchSession(){

    if(WCSession.isSupported()){
        watchSession = WCSession.default()
        watchSession!.delegate = self
        watchSession!.activate()
    }
}

// 当 iOS 应用触发事件时，下面的代理方法回调
func session(_ session: WCSession, didReceiveApplicationContext applicationContext: [String : Any])
{
    print("did ReceiveApplicationContext at watch")
}
```

Chapter 173: WCSessionDelegate

WCSessionDelegate works with watch OS2 + using WatchConnectivity. var watchSession : WCSession? func startWatchSession(){ if(WCSession.isSupported()){ watchSession = WCSession.default() watchSession!.delegate = self watchSession!.activate() } } Implement the required method:- didReceiveApplicationContext

Section 173.1: Watch kit controller (WKInterfaceController)

```
import WatchConnectivity

var watchSession : WCSession?

override func awake(withContext context: Any?) {
    super.awake(withContext: context)
    // Configure interface objects here.
startWatchSession()
}

func startWatchSession(){

    if(WCSession.isSupported()){
        watchSession = WCSession.default()
        watchSession!.delegate = self
        watchSession!.activate()
    }
}

//Callback in below delegate method when iOS app triggers event
func session(_ session: WCSession, didReceiveApplicationContext applicationContext: [String : Any])
{
    print("did ReceiveApplicationContext at watch")
}
```

第174章：AppDelegate

AppDelegate 是一个协议，定义了由单例 **UIApplication** 对象在应用程序生命周期中的重要事件发生时调用的方法。

通常用于在应用启动时执行任务（设置应用环境，分析（例如：Mixpanel/GoogleAnalytics/Crashlytics）、数据库堆栈等）以及关闭（例如：保存数据库上下文）、处理 URL 打开请求和类似的全应用程序任务。

第174.1节：通过AppDelegate方法的所有应用状态

要获取更新或在应用上线前执行某些操作，您可以使用以下方法。

AppDidFinishLaunching

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // 在应用启动前编写代码
    return YES;
}
```

当应用进入前台时：

```
- (void)applicationWillEnterForeground:(UIApplication *)application {
    // 当应用从后台切换到前台激活状态时调用；在这里可以撤销进入后台时所做的许多更改。
}
```

当应用启动以及从后台切换到前台时，会调用以下方法：

```
- (void)applicationDidBecomeActive:(UIApplication *)application {
    // 重新启动在应用处于非活动状态时暂停（或尚未开始）的任何任务。
    // 如果应用之前处于后台状态，可以选择刷新用户界面。
}
```

当应用进入后台时：

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    // 使用此方法释放共享资源，保存用户数据，失效定时器，并存储足够的应用状态信息，以便在应用被终止后恢复到当前状态。
    // 如果您的应用支持后台执行，当用户退出时，将调用此方法，而不是applicationWillTerminate。
}
```

当应用即将失去激活状态时

```
- (void)applicationWillResignActive:(UIApplication *)application {
    // 当应用即将从激活状态转为非激活状态时发送。此情况可能发生在某些临时中断（如来电或短信）时，或用户退出应用并开始切换到后台状态时。
    // 使用此方法暂停正在进行的任务，禁用定时器，并使图形渲染回调失效。游戏应使用此方法暂停游戏。
}
```

Chapter 174: AppDelegate

AppDelegate is a protocol which defines methods that are called by the singleton **UIApplication** object in response to important events in the lifetime of an app.

Normally used to perform tasks on application startup (setup app environment, analytics (ex.: Mixpanel/GoogleAnalytics/Crashlytics), DB stack etc.) and shutdown (ex.: save DB context), handling URL open requests and similar application-wide tasks.

Section 174.1: All States of Application through AppDelegate methods

For Getting updated or to do something before app goes live to user you can use below method.

AppDidFinishLaunching

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Write your code before app launch
    return YES;
}
```

While App Enter in Foreground:

```
- (void)applicationWillEnterForeground:(UIApplication *)application {
    // Called as part of the transition from the background to the active state; here you can undo many of the changes made on entering the background.
}
```

When App Launching and also background to Foreground hit below method:

```
- (void)applicationDidBecomeActive:(UIApplication *)application {
    // Restart any tasks that were paused (or not yet started) while the application was inactive.
    // If the application was previously in the background, optionally refresh the user interface.
}
```

While App Enter in background:

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
    // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
}
```

While app resign active

```
- (void)applicationWillResignActive:(UIApplication *)application {
    // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
    // Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering callbacks. Games should use this method to pause the game.
}
```

应用终止时：

```
- (void)applicationWillTerminate:(UIApplication *)application {
    // 当应用即将终止时调用。如有必要，保存数据。另见
    applicationDidEnterBackground:;
}
```

第174.2节：AppDelegate的角色：

- AppDelegate包含您的应用启动代码。
- 它响应应用状态的关键变化。具体来说，它响应临时中断以及应用执行状态的变化，例如应用从前台切换到后台时。
- 它响应来自应用外部的通知，例如远程通知（也称为推送通知）、低内存警告、下载完成通知等。
- 它决定是否应进行状态保存和恢复，并在需要时协助保存和恢复过程。
- 它响应针对应用本身的事件，而非特定于应用视图或视图控制器的事件。
您可以使用它来存储应用的核心数据对象或任何没有所属视图控制器的内容。

第174.3节：打开指定URL的资源

请求代理打开由 URL 指定的资源，并提供一个启动选项字典。

使用示例：

```
func application(_ app: UIApplication, open url: URL, options: [UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool {
    return SomeManager.shared.handle(
        url,
        sourceApplication: options[.sourceApplication] as? String,
        annotation: options[.annotation]
    )
}
```

第174.4节：处理本地和远程通知

使用示例：

```
/* 你自定义的 APNs/本地通知管理器实例 */
private var pushManager: AppleNotificationManager!
```

注册：

```
func application(application: UIApplication, didRegisterUserNotificationSettings notificationSettings: UIUserNotificationSettings) {
    // 调用以告知代理可以使用哪些类型的通知来获取用户的注意力
    pushManager.didRegisterSettings(notificationSettings)
}

func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken deviceToken: NSData) {
    // 告诉代理应用已成功注册到苹果推送通知服务 (Apple Push Notification service)
```

While app terminate:

```
- (void)applicationWillTerminate:(UIApplication *)application {
    // Called when the application is about to terminate. Save data if appropriate. See also
    applicationWillEnterBackground:;
}
```

Section 174.2: AppDelegate Roles:

- AppDelegate contains your app's startup code.
- It responds to key changes in the state of your app. Specifically, it responds to both temporary interruptions and to changes in the execution state of your app, such as when your app transitions from the foreground to the background.
- It responds to notifications originating from outside the app, such as remote notifications (also known as push notifications), low-memory warnings, download completion notifications, and more.
- It determines whether state preservation and restoration should occur and assists in the preservation and restoration process as needed.
- It responds to events that target the app itself and are not specific to your app's views or view controllers. You can use it to store your app's central data objects or any content that does not have an owning view controller.

Section 174.3: Opening a URL-Specified Resource

Asks the delegate to open a resource specified by a URL, and provides a dictionary of launch options.

Example of usage:

```
func application(_ app: UIApplication, open url: URL, options: [UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool {
    return SomeManager.shared.handle(
        url,
        sourceApplication: options[.sourceApplication] as? String,
        annotation: options[.annotation]
    )
}
```

Section 174.4: Handling Local and Remote Notifications

Example of usage:

```
/* Instance of your custom APNs/local notification manager */
private var pushManager: AppleNotificationManager!
```

Registration:

```
func application(application: UIApplication, didRegisterUserNotificationSettings notificationSettings: UIUserNotificationSettings) {
    // Called to tell the delegate the types of notifications that can be used to get the user's
    attention
    pushManager.didRegisterSettings(notificationSettings)
}

func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken deviceToken: NSData) {
    // Tells the delegate that the app successfully registered with Apple Push Notification service
```

```
(APNs)
pushManager.didRegisterDeviceToken(deviceToken)

}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    // 当苹果推送通知服务无法成功完成注册过程时发送给代理
    pushManager.didFailToRegisterDeviceToken(error)
}
```

远程通知处理：

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject :
AnyObject] {
    // 远程通知到达，有数据需要获取
    // 进行处理
    pushManager.handleNotification(userInfo,
background: application.applicationState == .Background
    )
}
```

本地通知处理：

```
func application(application: UIApplication, didReceiveLocalNotification notification:
UILocalNotification) {
    pushManager.handleLocalNotification(notification, background: false)
}
```

处理操作（已弃用）：

```
func application(application: UIApplication, handleActionWithIdentifier identifier: String?,
forRemoteNotification userInfo: [NSObject : AnyObject],
completionHandler: () -> Void) {
    pushManager.handleInteractiveRemoteNotification(userInfo, actionIdentifier: identifier,
completion: completionHandler)
}
```

```
(APNs)
pushManager.didRegisterDeviceToken(deviceToken)

}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    // Sent to the delegate when Apple Push Notification service cannot successfully complete the
registration process.
    pushManager.didFailToRegisterDeviceToken(error)
}
```

Remote notifications handling:

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject :
AnyObject]) {
    // Remote notification arrived, there is data to be fetched
    // Handling it
    pushManager.handleNotification(userInfo,
background: application.applicationState == .Background
    )
}
```

Local notifications handling:

```
func application(application: UIApplication, didReceiveLocalNotification notification:
UILocalNotification) {
    pushManager.handleLocalNotification(notification, background: false)
}
```

Handling action (deprecated):

```
func application(application: UIApplication, handleActionWithIdentifier identifier: String?,
forRemoteNotification userInfo: [NSObject : AnyObject],
completionHandler: () -> Void) {
    pushManager.handleInteractiveRemoteNotification(userInfo, actionIdentifier: identifier,
completion: completionHandler)
}
```

第175章：应用提交流程

本教程涵盖了上传 iOS 应用到 App Store 所需的所有必要步骤。

第175.1节：设置配置描述文件

在以前的版本中，配置描述文件的设置是手动完成的。您需要生成分发描述文件，下载它，然后分发您的应用。这必须针对每台开发机器进行，非常耗时。然而，在大多数情况下，现在的 Xcode 8 会为您完成大部分工作。确保您使用将用于分发应用的账户登录，然后只需在“Targets -> General”中选择“自动管理代码签名”。

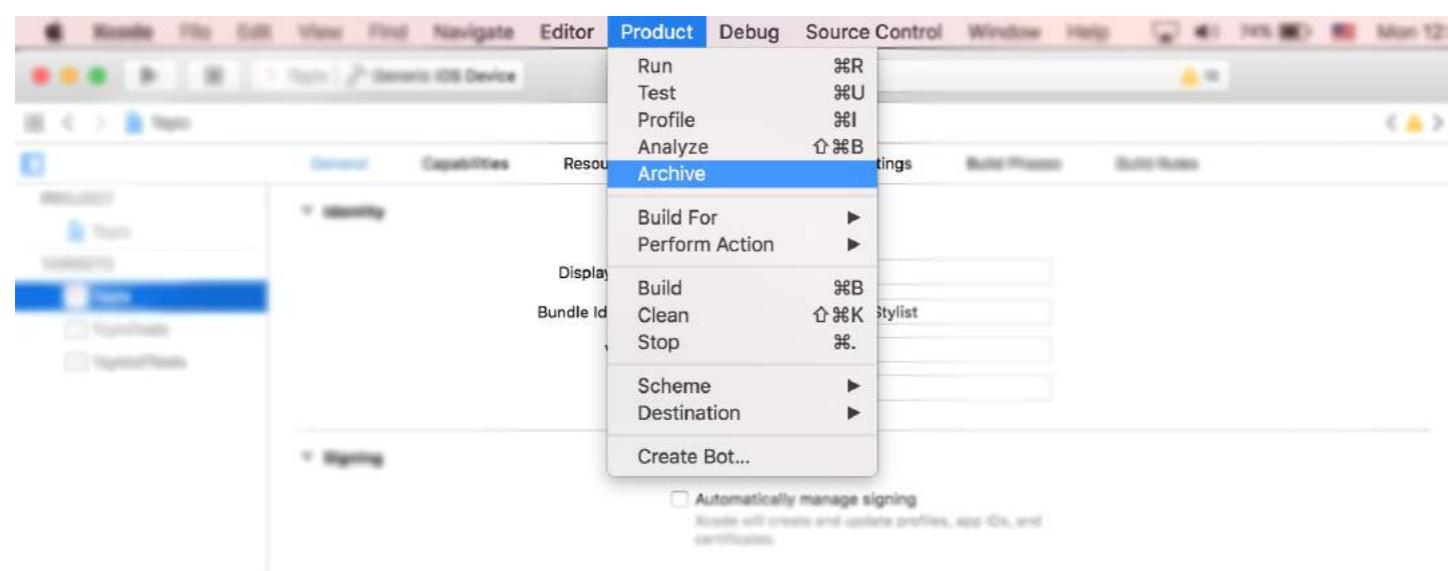
▼ Signing

- Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

第175.2节：归档代码

配置描述文件全部设置完成后，提交应用的下一步是归档您的代码。

从设备和模拟器的下拉菜单中选择“通用 iOS 设备”选项。然后，在“Product”菜单下选择“归档”选项。



如果提交的是对商店中现有应用的更新，请确保构建号高于当前版本且版本号不同。例如，当前应用的构建号为30，版本标签为1.0。下一次更新应至少为构建号31，版本标签为1.0.1。在大多数情况下，您应在版本号中添加第三位小数以应对紧急的错误修复或小补丁，第二位小数主要用于功能更新，而第一位小数在重大应用更新时递增。

第175.3节：导出 IPA 文件

完成后，您可以在 Xcode 归档管理器中找到您的归档文件。这里保存并管理着您所有之前的版本和归档构建，除非您删除它们。您会立即注意到一个大蓝色按钮写着“上传到 App Store...”，但在90%的情况下，由于各种原因（主要是 Xcode 的错误），此按钮无法正常工作。解决方法是导出归档文件，然后使用另一个名为 Application Loader 的 Xcode 工具上传。

但是，由于 Application Loader 是上传 IPA 文件到 App Store 的工具，归档文件需要导出为正确的格式。这是一个简单的任务，大约需要半小时。点击右侧面板中的“导出”按钮。

Chapter 175: App Submission Process

This tutorial covers all the necessary steps required to upload an iOS app to the App Store.

Section 175.1: Setup provisioning profiles

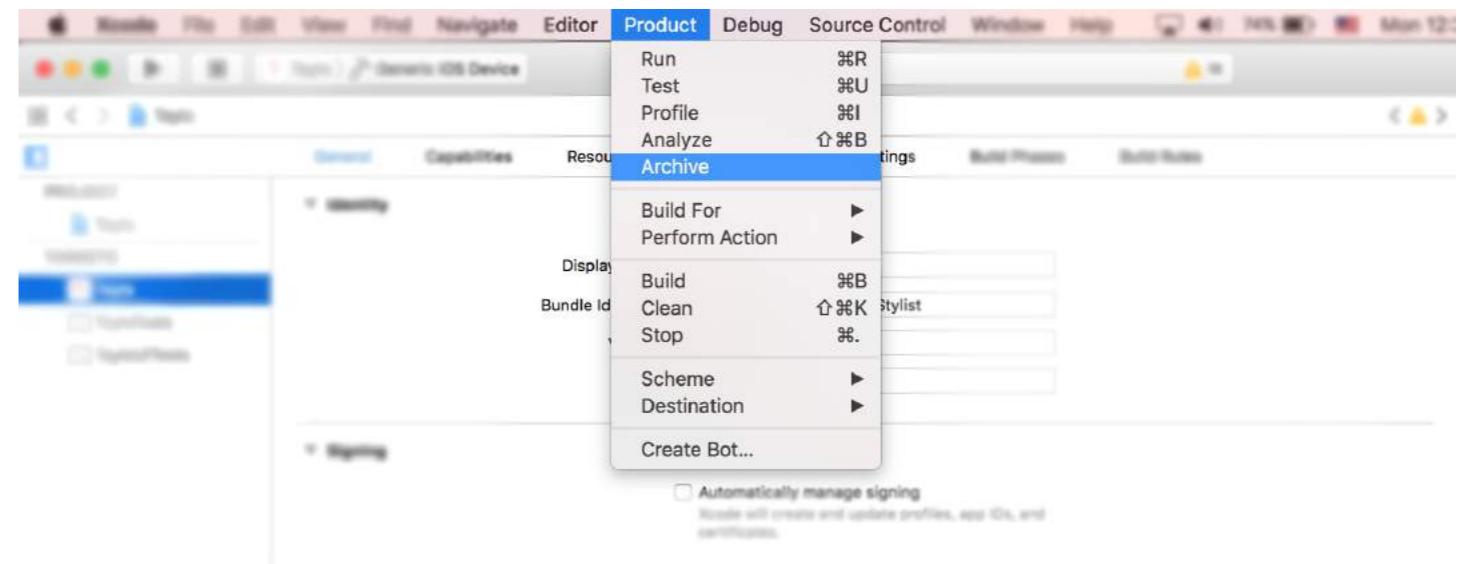
In previous versions, setting up provisioning profiles was done manually. You generate distribution profile, download it and then distribute your app. This had to be done for every development machine which was extremely time consuming. However, in most situations nowadays, Xcode 8 will do most of this work for you. Make sure you sign in with the account which will be used for distributing the app and then simply select "Automatically manage code signing" in Targets -> General.

▼ Signing

- Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Section 175.2: Archive the code

Once the provisioning profiles are all set, next step in the process of submitting the app is to archive your code. From the dropdown of devices and simulators select option "Generic iOS device". Then, under "Product" menu select option "Archive".

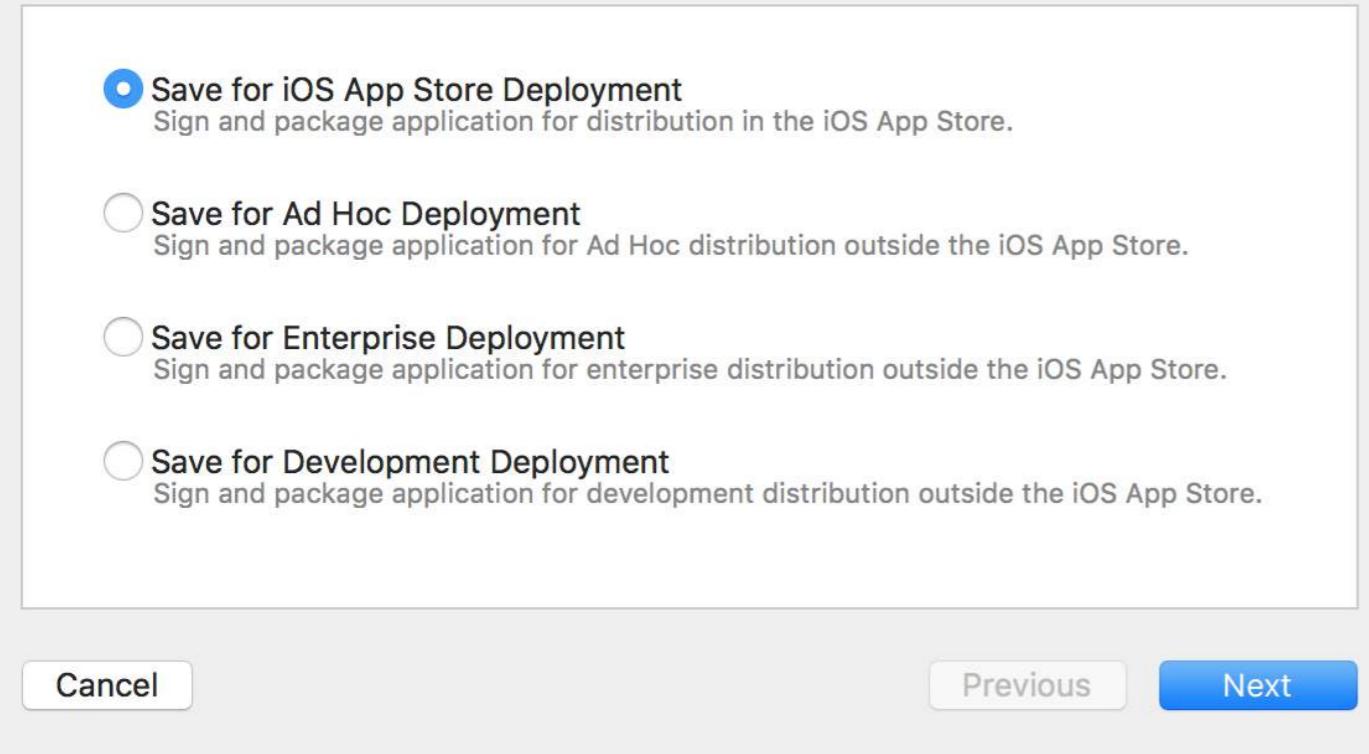


In case where the submission is an update to the existing app on the store, make sure that the build number is higher than the current and that the version number is different. For example, current application has build number 30 and version label 1.0. The next update should have at least build number 31 and version label 1.0.1. In most cases, you should add third decimal to your version in case of some urgent bug fixes or small patches, second decimal is mostly reserved for feature updates while first decimal is incremented in case of a major app update.

Section 175.3: Export IPA file

Once it's done, you can find your archive in the Xcode organizer. This is where all your previous versions and archive builds are saved and organized in case you do not delete them. You will immediately notice a large blue button saying "Upload to App Store..." however in 9/10 cases this will not work due to various reasons (Xcode bugs mostly). Workaround is to export your archive and upload it using another Xcode tool called Application Loader. However, since Application loader uploads IPA files to the App Store, archive needs to be exported to the correct format. This is a trivial task which might take ~ half an hour. Click on the "Export" button in the right side panel.

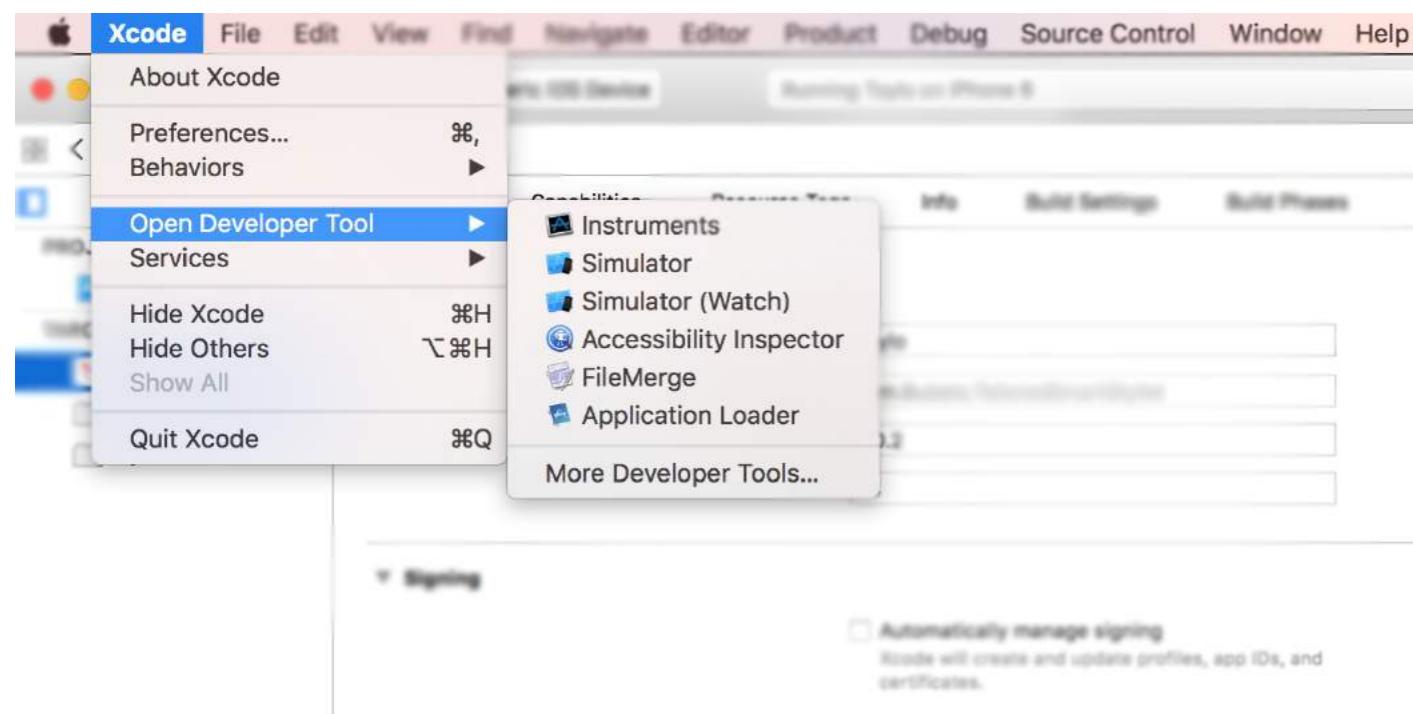
Select a method for export:



如果您要将应用上传到App Store，选择第一个选项并点击下一步。再次登录并验证您的代码，然后去喝杯咖啡。导出过程中，系统会询问您保存生成的IPA文件的位置。通常，桌面是最方便的选择。

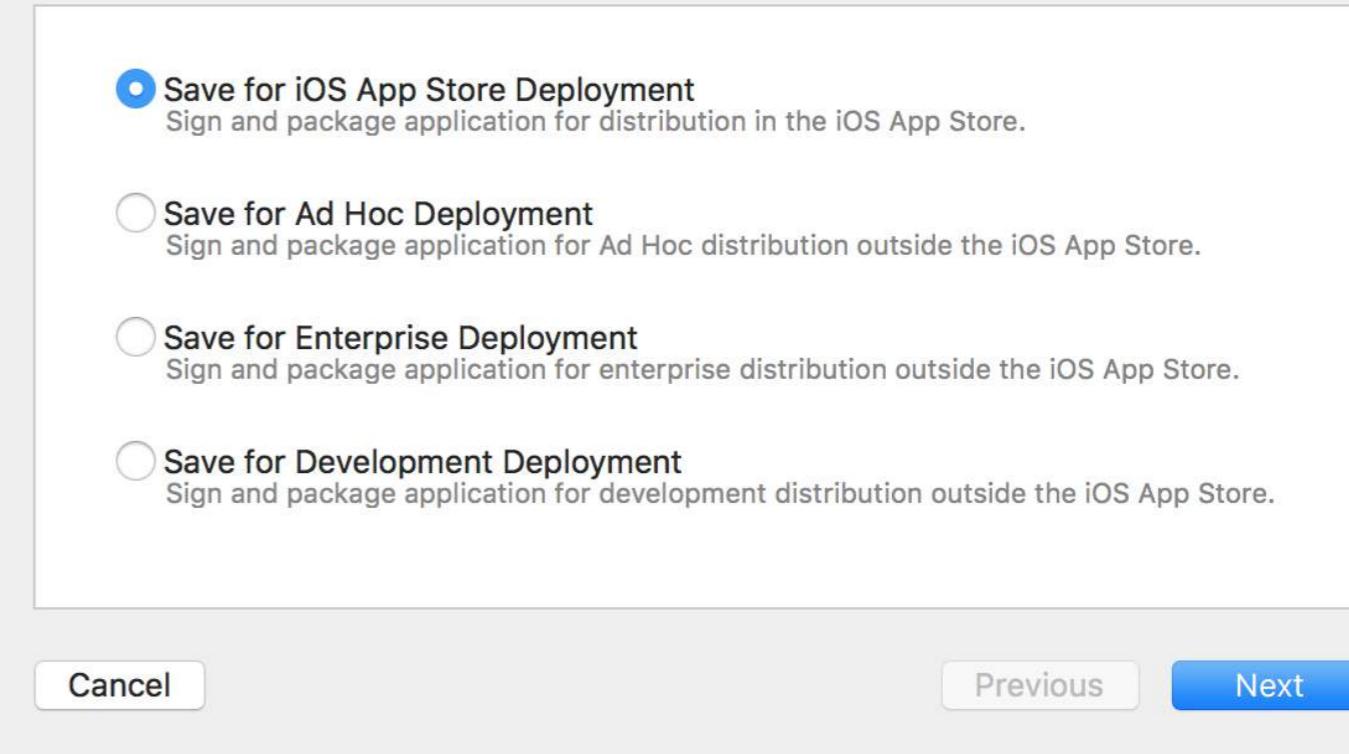
第175.4节：使用Application Loader上传IPA文件

IPA文件生成后，打开Xcode，导航到开发者工具并打开Application Loader。



如果您的Xcode中有多个账户，系统会要求您选择。自然地选择您在第一步中用于代码签名的账户。选择“交付您的应用”并上传代码。上传完成后，可能需要最多一小时才能在iTunes Connect的构建列表中显示。

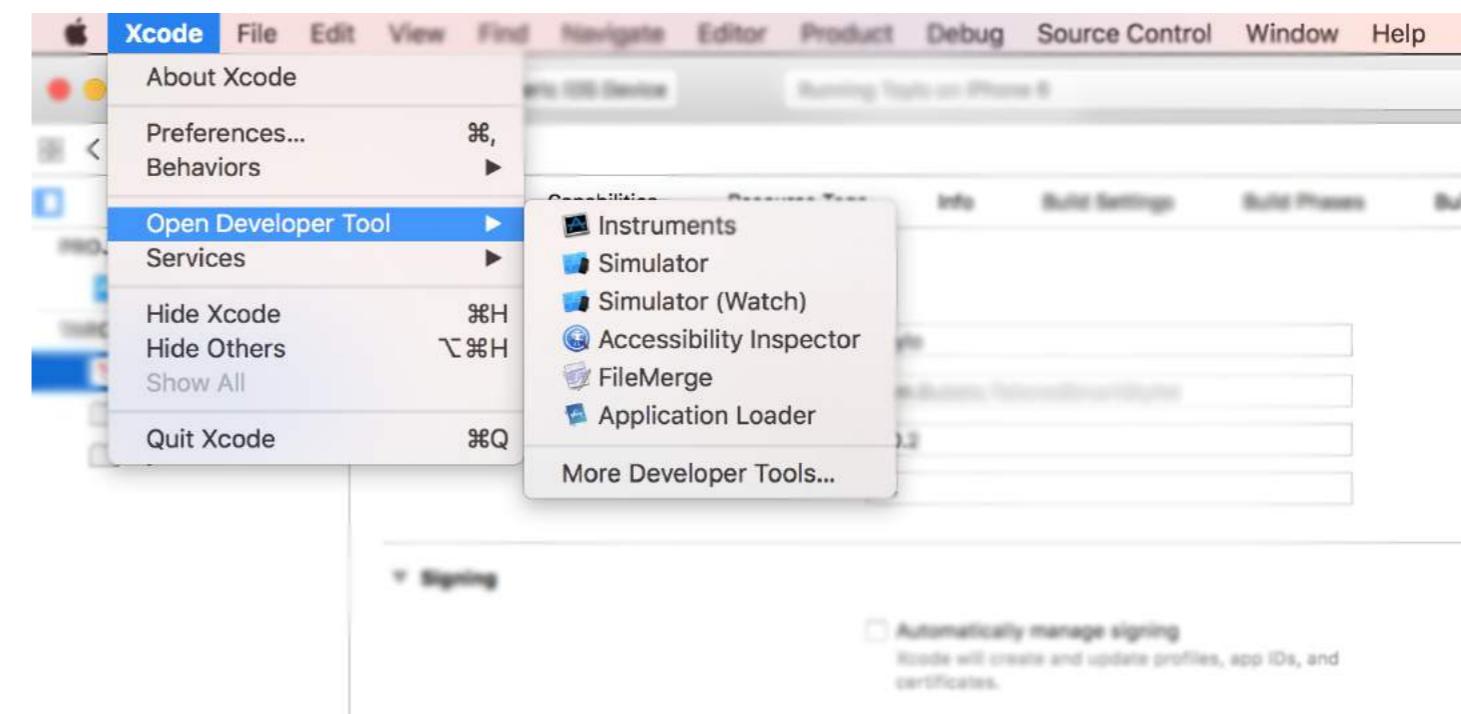
Select a method for export:



If you are uploading app to the App Store, select the first option and click Next. Sign in and validate your code once again and go grab a cup of coffee. Once the exporting process is done, you will be asked where to save the generated IPA file. Usually, desktop is the most convenient choice.

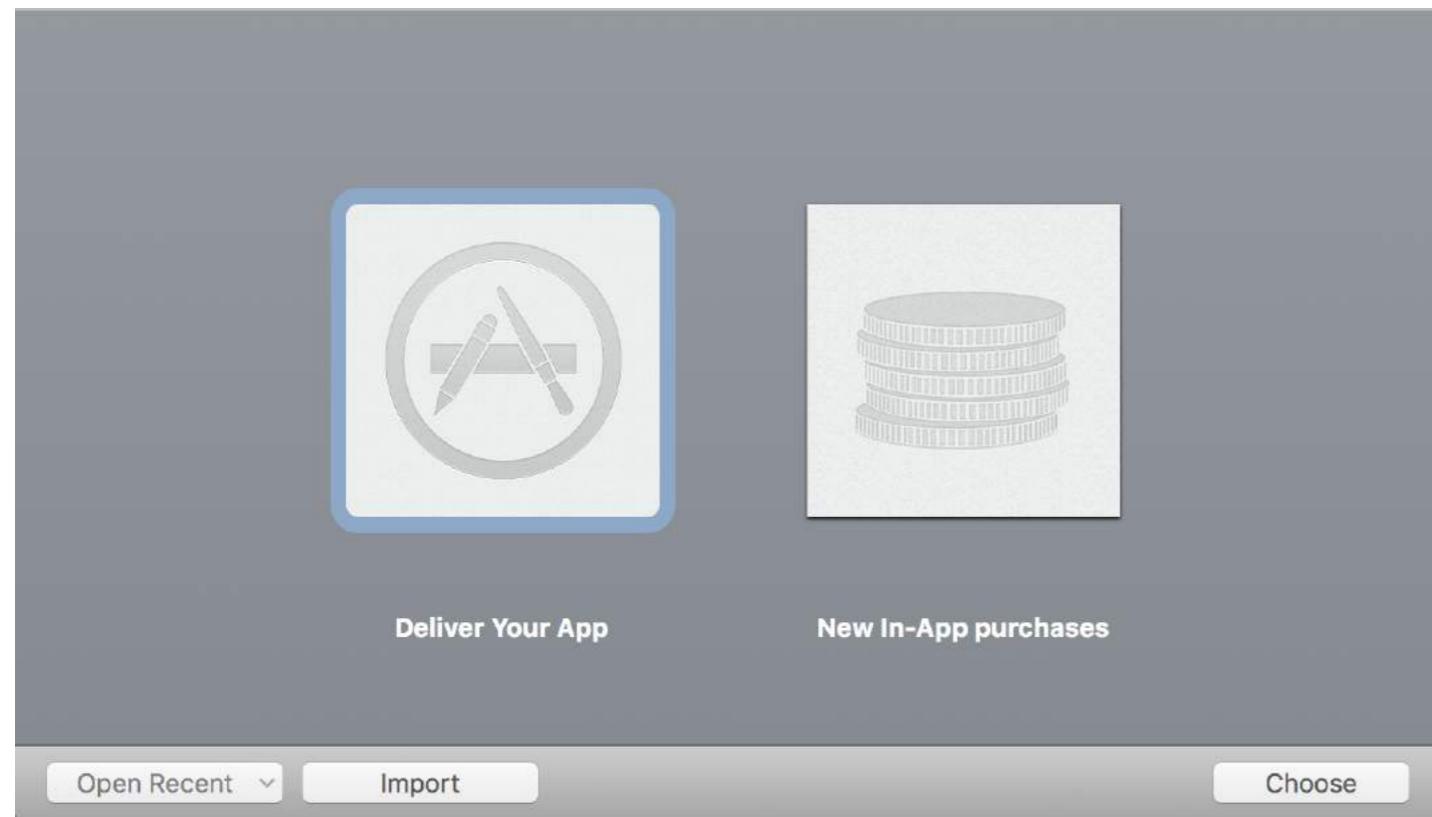
Section 175.4: Upload IPA file using Application Loader

Once the IPA file is generated, open Xcode, navigate to developer tools and open Application Loader.

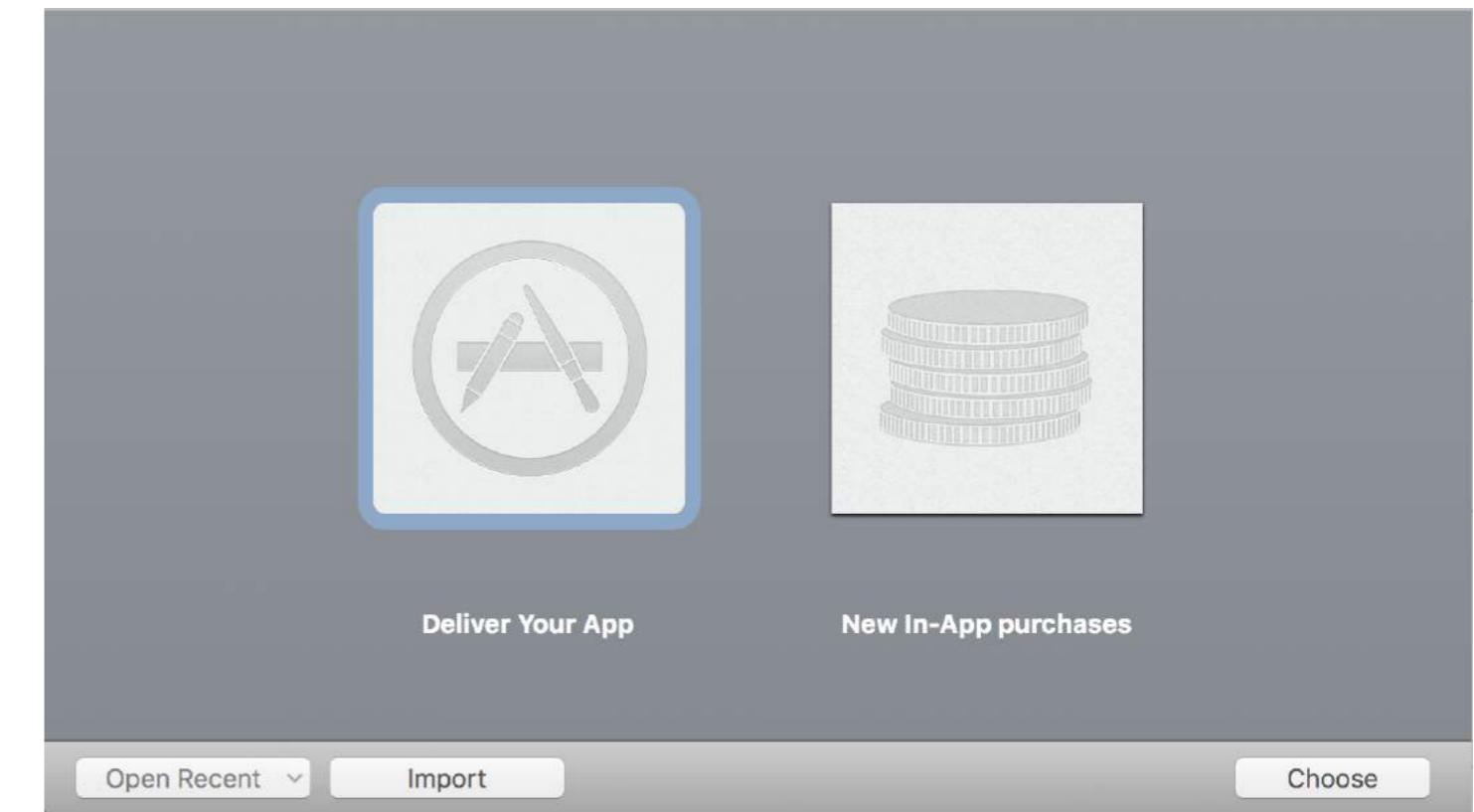


If you have multiple accounts in your Xcode, you will be asked to choose. Naturally pick "Deliver your app" and upload the code. Once the upload is done it may take up to one

可能需要最多一小时才能在iTunes Connect的构建列表中显示。



hour to appear in your build list on iTunes Connect.



第176章：FileHandle

从文档目录分块读取文件

第176.1节：从文档目录分块读取文件

我从文档目录获取文件路径，并以1024字节为块读取该文件，然后保存（追加）到NSMutableData对象，或者你也可以直接写入套接字。

```
// MARK: - 获取文件数据块的方法。
func getFileDataInChunks() {

    let documentDirectoryPath = NSSearchPathForDirectoriesInDomains(.documentDirectory,
    .userDomainMask, true)[0] as NSString
    let filePath = documentDirectoryPath.appendingPathComponent("video.mp4")

    // 检查路径下文件是否存在。
    if FileManager.default.fileExists(atPath: filePath) {

        let chunkSize = 1024 // 将数据分成1 KB

        // 创建NSMutableData对象以保存读取的数据。
        let ReadData = NSMutableData()

        do {

            // 打开文件以供读取。
            outputFileHandle = try FileHandle(forReadingFrom: URL(fileURLWithPath: filePath))

            // 获取第一个数据块
            var datas = outputFileHandle?.readData(ofLength: chunkSize)

            // 检查下一个数据块是否为空。
            while !(datas?.isEmpty)! {

                // 这里我将数据块写入ReadData，或者你也可以直接写入socket。
                ReadData.append(datas!)

                // 获取下一个数据块
                datas = outputFileHandle?.readData(ofLength: chunkSize)

                print("Running: \(ReadData.length)")

            }

            // 读取数据完成后关闭outputFileHandle。
            outputFileHandle?.closeFile()

            print("文件读取完成")

        }catch let error as NSError {
            print("错误 : \(error.localizedDescription)")
        }
    }
}
```

文件读取完成后，你将在ReadData变量中获得文件数据，这里outputFileHandle是FileHandle的一个对象

Chapter 176: FileHandle

Read file in chunks from document directory

Section 176.1: Read file from document directory in chunks

I get the file path from document directory and read that file in chunks of 1024 and save (append) to `NSMutableData` object or you can directly write to socket.

```
// MARK: - Get file data as chunks Methode.
func getFileDataInChunks() {

    let documentDirectoryPath = NSSearchPathForDirectoriesInDomains(.documentDirectory,
    .userDomainMask, true)[0] as NSString
    let filePath = documentDirectoryPath.appendingPathComponent("video.mp4")

    //Check file exists at path or not.
    if FileManager.default.fileExists(atPath: filePath) {

        let chunkSize = 1024 // divide data into 1 kb

        //Create NSMutableData object to save read data.
        let ReadData = NSMutableData()

        do {

            //open file for reading.
            outputFileHandle = try FileHandle(forReadingFrom: URL(fileURLWithPath: filePath))

            // get the first chunk
            var datas = outputFileHandle?.readData(ofLength: chunkSize)

            //check next chunk is empty or not.
            while !(datas?.isEmpty)! {

                //here I write chunk data to ReadData or you can directly write to socket.
                ReadData.append(datas!)

                // get the next chunk
                datas = outputFileHandle?.readData(ofLength: chunkSize)

                print("Running: \(ReadData.length)")

            }

            //close outputFileHandle after reading data complete.
            outputFileHandle?.closeFile()

            print("File reading complete")

        }catch let error as NSError {
            print("Error : \(error.localizedDescription)")
        }
    }
}
```

After file reading complete you will get file Data in ReadData variable Here outputFileHandle is a object of FileHandle

```
var outputFileHandle:FileHandle?
```

```
var outputFileHandle:FileHandle?
```

第177章：基础文本文件输入输出

第177.1节：从文档文件夹读取和写入

Swift 3

```
import UIKit

// 保存字符串到文件
let fileName = "TextFile"
let documentDirectory = try FileManager.default.urlForDirectory(.documentDirectory, in:
    .userDomainMask, appropriateFor: nil, create: true)

var fileURL = try documentDirectory.appendingPathComponent(fileName).appendingPathExtension("txt")

print("文件路径: \(fileURL.path)")

var toFileString = "要写入的文本"
do {
    // 写入文件
    try toFileString.writeToURL(fileURL, atomically: true, encoding: NSUTF8StringEncoding)
} catch let error as NSError {
    print("写入URL失败: \(fileURL), 错误: \(error.localizedDescription)")
}

// 读取
var fromFileString = ""
do {
    fromFileString = try String(contentsOfURL: fileURL)
} catch let error as NSError {
    print("从URL读取失败: \(fileURL), 错误: \(error.localizedDescription)")
}
print("来自文件的文本输入: \(fromFileString)")
```

Swift 2

```
import UIKit

// 保存字符串到文件
let fileName = "TextFile"
let DocumentDirectoryURL = try! NSFileManager.defaultManager().URLForDirectory(.DocumentDirectory,
    inDomain: .UserDomainMask, appropriateForURL: nil, create: true)

let fileURL =
    DocumentDirectoryURL.URLByAppendingPathComponent(fileName).URLByAppendingPathExtension("txt")
print("文件路径: \(fileURL.path)")

var toFileString = "要写入的文本"
do {
    // 写入文件
    try toFileString.writeToURL(fileURL, atomically: true, encoding: NSUTF8StringEncoding)
} catch let error as NSError {
    print("写入URL失败: \(fileURL), 错误: \(error.localizedDescription)")
}

// 读取
var fromFileString = ""
do {
    fromFileString = try String(contentsOfURL: fileURL)
```

Chapter 177: Basic text file I/O

Section 177.1: Read and write from Documents folder

Swift 3

```
import UIKit

// Save String to file
let fileName = "TextFile"
let documentDirectory = try FileManager.default.urlForDirectory(.documentDirectory, in:
    .userDomainMask, appropriateFor: nil, create: true)

var fileURL = try documentDirectory.appendingPathComponent(fileName).appendingPathExtension("txt")

print("FilePath: \(fileURL.path)")

var toFileString = "Text to write"
do {
    // Write to file
    try toFileString.writeToURL(fileURL, atomically: true, encoding: NSUTF8StringEncoding)
} catch let error as NSError {
    print("Failed writing to URL: \(fileURL), Error: \(error.localizedDescription)")
}

// Reading
var fromFileString = ""
do {
    fromFileString = try String(contentsOfURL: fileURL)
} catch let error as NSError {
    print("Failed reading from URL: \(fileURL), Error: \(error.localizedDescription)")
}
print("Text input from file: \(fromFileString)")
```

Swift 2

```
import UIKit

// Save String to file
let fileName = "TextFile"
let DocumentDirectoryURL = try! NSFileManager.defaultManager().URLForDirectory(.DocumentDirectory,
    inDomain: .UserDomainMask, appropriateForURL: nil, create: true)

let fileURL =
    DocumentDirectoryURL.URLByAppendingPathComponent(fileName).URLByAppendingPathExtension("txt")
print("FilePath: \(fileURL.path)")

var toFileString = "Text to write"
do {
    // Write to file
    try toFileString.writeToURL(fileURL, atomically: true, encoding: NSUTF8StringEncoding)
} catch let error as NSError {
    print("Failed writing to URL: \(fileURL), Error: \(error.localizedDescription)")
}

// Reading
var fromFileString = ""
do {
    fromFileString = try String(contentsOfURL: fileURL)
```

```
} catch let error as NSError {
    print("从URL读取失败: \(fileURL), 错误: " + error.localizedDescription)
}
print("来自文件的文本输入: \(fromFileString)")
```

```
} catch let error as NSError {
    print("Failed reading from URL: \(fileURL), Error: " + error.localizedDescription)
}
print("Text input from file: \(fromFileString)")
```

第178章：iOS 语音合成 (TTS)

了解如何在iOS设备上将文本合成为语音

第178.1节：文本转语音

Objective C

```
AVSpeechSynthesizer *synthesizer = [[AVSpeechSynthesizer alloc] init];
AVSpeechUtterance *utterance = [AVSpeechUtterance speechUtteranceWithString:@"Some text"];
[utterance setRate:0.2f];
[synthesizer speakUtterance:utterance];
```

Swift

```
let synthesizer = AVSpeechSynthesizer()
let utterance = AVSpeechUtterance(string: "Some text")
utterance.rate = 0.2
```

你也可以这样更改语音：

```
utterance.voice = AVSpeechSynthesisVoice(language: "fr-FR")
```

然后进行朗读

- 在Swift 2中：`synthesizer.speakUtterance(utterance)`
- 在Swift 3中：`synthesizer.speak(utterance)`

别忘了导入AVFoundation

有用的方法

您可以使用以下两种方法停止或暂停所有语音：

```
- (BOOL)pauseSpeakingAtBoundary:(AVSpeechBoundary)boundary;
- (BOOL)stopSpeakingAtBoundary:(AVSpeechBoundary)boundary;
```

`AVSpeechBoundary` 指示语音是应立即暂停或停止 (`AVSpeechBoundaryImmediate`)，还是应在当前正在朗读的单词之后暂停或停止 (`AVSpeechBoundaryWord`)。

Chapter 178: iOS TTS

Find how to produce synthesized speech from text on an iOS device

Section 178.1: Text To Speech

Objective C

```
AVSpeechSynthesizer *synthesizer = [[AVSpeechSynthesizer alloc] init];
AVSpeechUtterance *utterance = [AVSpeechUtterance speechUtteranceWithString:@"Some text"];
[utterance setRate:0.2f];
[synthesizer speakUtterance:utterance];
```

Swift

```
let synthesizer = AVSpeechSynthesizer()
let utterance = AVSpeechUtterance(string: "Some text")
utterance.rate = 0.2
```

You can also change the voice like this :

```
utterance.voice = AVSpeechSynthesisVoice(language: "fr-FR")
```

And then speak

- In Swift 2 : `synthesizer.speakUtterance(utterance)`
- In Swift 3 : `synthesizer.speak(utterance)`

Don't forget to import AVFoundation

Helpful methods

You can Stop or Pause all speech using these two methods :

```
- (BOOL)pauseSpeakingAtBoundary:(AVSpeechBoundary)boundary;
- (BOOL)stopSpeakingAtBoundary:(AVSpeechBoundary)boundary;
```

The `AVSpeechBoundary` indicates if the speech should pause or stop immediately (`AVSpeechBoundaryImmediate`) or it should pause or stop after the word currently being spoken (`AVSpeechBoundaryWord`).

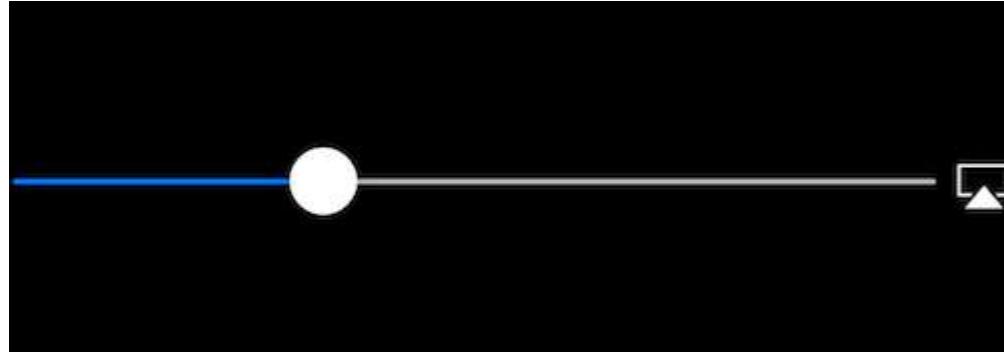
第179章：MPVolumeView

MPVolumeView 类是一个音量视图，向用户展示一个滑块控件用于设置系统音频输出音量，以及一个按钮用于选择音频输出路径。

第179.1节：添加 MPVolumeView

```
// 在一个容器视图中添加 MPVolumeView
let mpVolumeHolderView = UIView(frame: CGRect(x: 0, y: view.bounds.midY, width: view.bounds.width,
height: view.bounds.height))
// 设置容器视图的背景色为透明
mpVolumeHolderView.backgroundColor = .clear
let mpVolume = MPVolumeView(frame: mpVolumeHolderView.bounds)
mpVolume.showsRouteButton = true
mpVolumeHolderView.addSubview(mpVolume)
view.addSubview(mpVolumeHolderView)
// 音量视图是白色的，为了在此示例中更好地显示它，将父视图背景设置为黑色
view.backgroundColor = .black
```

!!! 一个非常重要的注意事项是，MPVolumeView 仅在真实设备上有效，模拟器上无效。



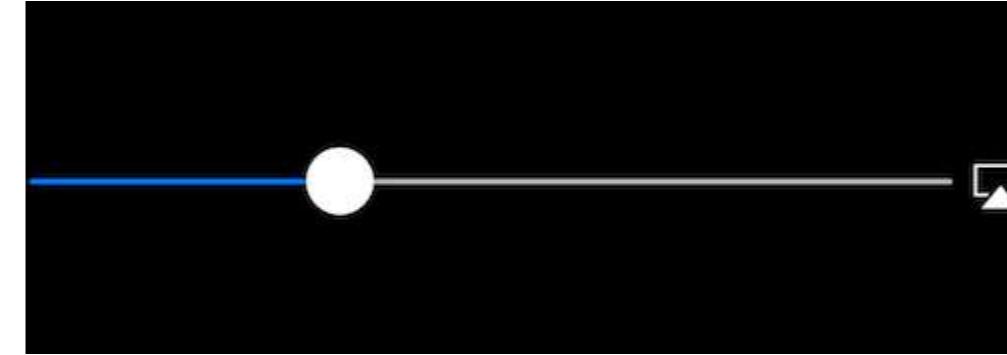
Chapter 179: MPVolumeView

The MPVolumeView class is volume view to presents the user with a slider control for setting the system audio output volume, and a button for choosing the audio output route.

Section 179.1: Adding a MPVolumeView

```
// Add MPVolumeView in a holder view
let mpVolumeHolderView = UIView(frame: CGRect(x: 0, y: view.bounds.midY, width: view.bounds.width,
height: view.bounds.height))
// Set the holder view's background color to transparent
mpVolumeHolderView.backgroundColor = .clear
let mpVolume = MPVolumeView(frame: mpVolumeHolderView.bounds)
mpVolume.showsRouteButton = true
mpVolumeHolderView.addSubview(mpVolume)
view.addSubview(mpVolumeHolderView)
// the volume view is white, set the parent background to black to show it better in this example
view.backgroundColor = .black
```

!!! A very important note is that the MPVolumeView only works on an actual device and not on a simulator.



第180章：Objective-C 关联对象

参数

object 你想要修改的现有对象

key 这基本上可以是任何具有固定内存地址的指针，但一个好的做法是在这里使用一个计算属性（getter）

value 您想要添加的对象

policy 这个newValue的内存策略，即是否应被保留/赋值、复制等，就像你声明的其他属性一样属性

首次在 iOS 3.1 中作为 Objective-C 运行时的一部分引入，关联对象提供了一种向现有类对象添加实例变量的方法（无需子类化）。

这意味着你可以将任何对象附加到任何其他对象上，而无需子类化。

第180.1节：基本关联对象示例

假设我们需要向SomeClass添加一个 NSString 对象（我们不能子类化）。

在此示例中，我们不仅创建了一个关联对象，还将其包装在类别中的计算属性中，以实现更整洁的代码

```
#import <objc/runtime.h>

@interface SomeClass (MyCategory)
// 这是包装关联对象的属性。下面我们实现了 setter 和 getter
//，实际上利用了对象关联机制
@property (nonatomic, retain) NSString *associated;
@end

@implementation SomeClass (MyCategory)

- (void)setAssociated:(NSString *)object {
    objc_setAssociatedObject(self, @selector(associated), object,
                           OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (NSString *)associated {
    return objc_getAssociatedObject(self, @selector(associated));
}
```

现在使用该属性就这么简单

```
SomeClass *instance = [SomeClass alloc] init];
instance.associated = @"this property is an associated object under the hood";
```

Chapter 180: Objective-C Associated Objects

Param

object The existing object you want to modify

key This can basically be any pointer that has a constant memory address, but a nice practice is to use here a computed property (getter)

value The object you want to add

policy The memory policy for this new value i.e. should it be retained / assigned, copied etc.. just like any other property you'd declare

First introduced in iOS 3.1 as part of the Objective-C runtime, associated objects provide a way to add instance variables to an existing class object (w/o subclassing).

This means you'll be able to attach any object to any other object without subclassing.

Section 180.1: Basic Associated Object Example

Assume we need to add an NSString object to SomeClass (we can't subclass).

In this example we not only create an associated object but also wrap it in a computed property in a category for extra neatness

```
#import <objc/runtime.h>

@interface SomeClass (MyCategory)
// This is the property wrapping the associated object. Below we implement the setter and getter
// which actually utilize the object association
@property (nonatomic, retain) NSString *associated;
@end

@implementation SomeClass (MyCategory)

- (void)setAssociated:(NSString *)object {
    objc_setAssociatedObject(self, @selector(associated), object,
                           OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (NSString *)associated {
    return objc_getAssociatedObject(self, @selector(associated));
}
```

Now it would be as easy as this to use the property

```
SomeClass *instance = [SomeClass alloc] init];
instance.associated = @"this property is an associated object under the hood";
```

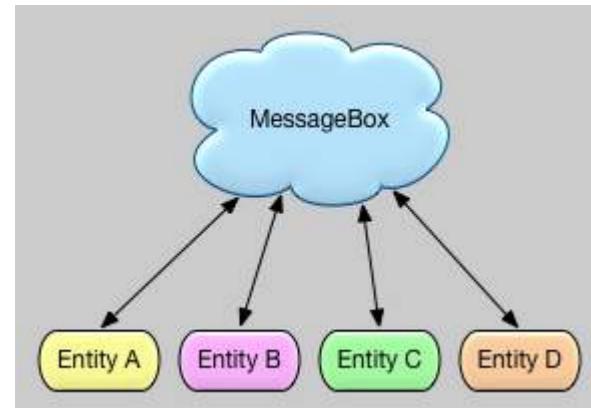
第181章：视图控制器之间传递数据（基于 MessageBox 概念）

MessageBox是一个用于解耦实体的简单概念。

例如，实体A可以发送一条消息，实体B可以在合适的时候读取。

一个视图控制器想要与另一个视图控制器通信，但你不想创建强引用或弱引用关系。

第181.1节：简单示例用法



```
let messageBox:MessageBox = MessageBox()  
  
// 设置  
messageBox.setObject("TestObject1", forKey:"TestKey1")  
  
// 获取  
// 但不移除它，保持存储状态，以便稍后仍可检索  
let someObject:String = messageBox.getObject(forKey:"TestKey1", removeIfFound:false)  
  
// 获取  
// 并移除它  
let someObject:String = messageBox.getObject(forKey:"TestKey1", removeIfFound:true)
```

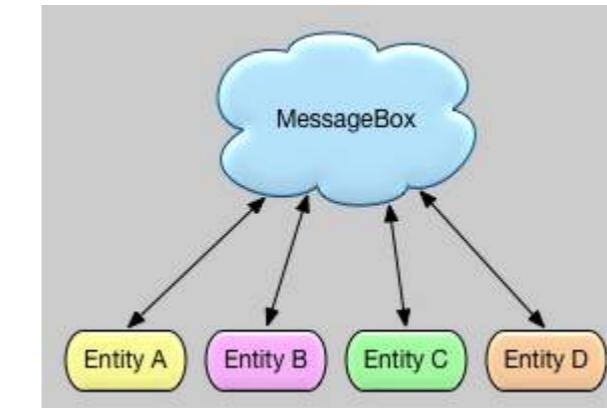
Chapter 181: Passing Data between View Controllers (with MessageBox-Concept)

MessageBox is a simple concept for decoupling entities.

For example entity A can place a message that entity B can read whenever suitable.

A view controller would like to talk to another view controller, but you don't want to create a strong or weak relationship.

Section 181.1: Simple Example Usage

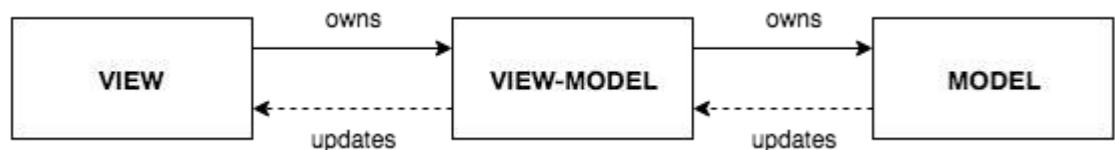


```
let messageBox:MessageBox = MessageBox()  
  
// set  
messageBox.setObject("TestObject1", forKey:"TestKey1")  
  
// get  
// but don't remove it, keep it stored, so that it can still be retrieved later  
let someObject:String = messageBox.getObject(forKey:"TestKey1", removeIfFound:false)  
  
// get  
// and remove it  
let someObject:String = messageBox.getObject(forKey:"TestKey1", removeIfFound:true)
```

第182章：MVVM

第182.1节：无响应式编程的MVVM

我将先简短地解释一下什么是Model-View-ViewModel (MVVM) 设计模式以及为什么在你的iOS应用中使用它。当iOS刚出现时，苹果建议使用MVC (Model-View-Controller) 作为设计模式。他们在所有示例中都展示了它，所有初期开发者都很乐意使用它，因为它很好地将业务逻辑和用户界面分离开来。随着应用变得越来越大和复杂，一个新的问题出现了，恰当地称为“庞大视图控制器” (Massive View Controllers, MVC)。因为所有业务逻辑都被添加到了ViewController中，随着时间推移，它们通常变得过于庞大和复杂。为了解决MVC问题，iOS世界引入了一种新的设计模式——Model-View-ViewModel (MVVM) 模式。



上图展示了MVVM的结构。你有一个标准的ViewController + View（在Storyboard、XIB或代码中），它作为MVVM的View（后文中“View”将指MVVM的View）。View持有对ViewModel的引用，ViewModel中包含我们的业务逻辑。需要注意的是，ViewModel对View一无所知，也从不持有对View的引用。ViewModel持有对Model的引用。

理论部分关于MVVM就讲到这里。更多内容可以阅读 [here](#)。

MVVM的一个主要问题是，当ViewModel没有任何引用且甚至对View一无所知时，如何通过ViewModel更新View。

本示例的主要部分是展示如何使用MVVM（更准确地说，如何绑定ViewModel和View）而不使用任何响应式编程（ReactiveCocoa、ReactiveSwift或RxSwift）。顺便提一句：如果你想使用响应式编程，那更好，因为使用它可以非常轻松地完成MVVM绑定。但本示例是关于如何在不使用响应式编程的情况下使用MVVM。

让我们创建一个简单的示例来演示如何使用MVVM。

我们的MVVMEmployeeViewController是一个简单的ViewController，包含一个标签和一个按钮。当按钮被按下时，标签文本应设置为“Hello”。由于决定用户交互时该做什么是业务逻辑的一部分，ViewModel将负责决定用户按下按钮时该做什么。MVVM的View不应包含任何业务逻辑。

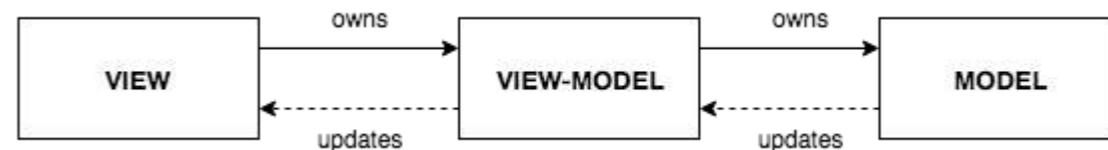
```
class MVVMEmployeeViewController: UIViewController {  
  
    @IBOutlet weak var helloLabel: UILabel!  
  
    var viewModel: MVVMEmployeeViewModel?  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
    }  
  
    @IBAction func sayHelloButtonPressed(_ sender: UIButton) {  
        viewModel?.userTriggeredSayHelloButton()  
    }  
}
```

MVVMEmployeeViewModel 是一个简单的视图模型 (ViewModel)。

Chapter 182: MVVM

Section 182.1: MVVM Without Reactive Programming

I'll start with a really short explanation what is and why use Model-View-ViewModel (MVVM) design pattern in your iOS apps. When iOS first appeared, Apple suggested to use MVC (Model-View-Controller) as a design pattern. They showed it in all of their examples and all first developers were happy using it because it nicely separated concerns between business logic and user interface. As applications became larger and more complex a new problem appeared appropriately called Massive View Controllers (MVC). Because all business logic was added in the ViewController, with time they usually became too large and complex. To avoid MVC issue, a new design pattern was introduced to the world of iOS - Model-View-ViewModel (MVVM) pattern.



This is enough with a theoretical part of the MVVM. More about it can be read [here](#).

One of the **main issues with MVVM** is how to update View via the ViewModel when ViewModel doesn't have any references and doesn't even know anything about the View.

The main part of this example is to show how to use MVVM (more precisely, how to bind ViewModel and View) without any reactive programming (ReactiveCocoa, ReactiveSwift or RxSwift). Just as a note: if you want to use Reactive programming, even better since MVVM bindings are done really easy using it. But this example is on how to use MVVM without Reactive programming.

Let's create a simple example to demonstrate how to use MVVM.

Our MVVMEmployeeViewController is a simple ViewController with a label and a button. When button is pressed, the label text should be set to 'Hello'. Since deciding what to do on user interaction is part of the business logic, ViewModel will have to decide what to do when user presses the button. MVVM's View shouldn't do any business logic.

```
class MVVMEmployeeViewController: UIViewController {  
  
    @IBOutlet weak var helloLabel: UILabel!  
  
    var viewModel: MVVMEmployeeViewModel?  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
    }  
  
    @IBAction func sayHelloButtonPressed(_ sender: UIButton) {  
        viewModel?.userTriggeredSayHelloButton()  
    }  
}
```

MVVMEmployeeViewModel is a simple ViewModel.

```

class MVVMEExampleViewModel {

    func userTriggeredSayHelloButton() {
        // 当没有视图的引用时，如何更新视图的标签？
    }
}

```

你可能会想，如何在视图中设置视图模型的引用。我通常在视图控制器（ViewController）初始化时或显示之前进行设置。对于这个简单的例子，我会在AppDelegate中这样做：

```

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    if let rootVC = window?.rootViewController as? MVVMEExampleViewController {
        let viewModel = MVVMEExampleViewModel()
        rootVC.viewModel = viewModel
    }
}

返回 true

```

现在真正的问题是：如何在不将视图的引用传给视图模型的情况下，从视图模型更新视图？（记住，我们不会使用任何iOS的响应式编程库）

你可以考虑使用KVO，但那样会让事情变得过于复杂。一些聪明的人思考过这个问题，提出了Bond库。这个库起初看起来可能有些复杂且难以理解，所以我只取其中一小部分，让我们的MVVM完全可用。

让我们介绍Dynamic类，它是我们简单但功能齐全的MVVM模式的核心。

```

class Dynamic<T> {
    typealias Listener = (T) -> Void
    var listener: Listener?

    func bind(_ listener: Listener?) {
        self.listener = listener
    }

    func bindAndFire(_ listener: Listener?) {
        self.listener = listener
        listener?(value)
    }

    var value: T {
        didSet {
            listener?(value)
        }
    }

    init(_ v: T) {
        value = v
    }
}

```

Dynamic 类使用泛型和闭包将我们的视图模型与视图绑定。我不会详细讲解这个类，我们可以在评论中讨论（以使这个示例更简短）。现在让我们更新我们的 MVVMEExampleViewController 和 MVVMEExampleViewModel 用于使用这些类。

我们更新的 MVVMEExampleViewController

```

class MVVMEExampleViewModel {

    func userTriggeredSayHelloButton() {
        // How to update View's label when there is no reference to the View??
    }
}

```

You might wonder, how to set the ViewModel's reference in the View. I usually do it when ViewController is being initialized or before it will be shown. For this simple example, I would do something like this in the AppDelegate:

```

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    if let rootVC = window?.rootViewController as? MVVMEExampleViewController {
        let viewModel = MVVMEExampleViewModel()
        rootVC.viewModel = viewModel
    }

    return true
}

```

The real question now is: how to update View from ViewModel without giving a reference to the View to the ViewModel? (Remember, we won't use any of the Reactive Programming iOS libraries)

You could think about using KVO, but that would just complicate things too much. Some smart people have thought about the issue and came up with the [Bond library](#). The library might seem complicated and a little harder to understand at first, so I'll just take one small part of it and make our MVVM fully functional.

Let's introduce the Dynamic class which is the core of our simple yet fully functional MVVM pattern.

```

class Dynamic<T> {
    typealias Listener = (T) -> Void
    var listener: Listener?

    func bind(_ listener: Listener?) {
        self.listener = listener
    }

    func bindAndFire(_ listener: Listener?) {
        self.listener = listener
        listener?(value)
    }

    var value: T {
        didSet {
            listener?(value)
        }
    }

    init(_ v: T) {
        value = v
    }
}

```

Dynamic class is using Generics and Closures to bind our ViewModel with our View. I won't go into details about this class, we can do it in the comments (to make this example shorter). Let's now update our MVVMEExampleViewController and MVVMEExampleViewModel to use those classes.

Our updated MVVMEExampleViewController

```

class MVVMEExampleViewController: UIViewController {

    @IBOutlet weak var helloLabel: UILabel!

    var viewModel: MVVMEExampleViewModel?

    override func viewDidLoad() {
        super.viewDidLoad()
        bindViewModel()
    }

    func bindViewModel() {
        if let viewModel = viewModel {
            viewModel.helloText.bind({ (helloText) in
                DispatchQueue.main.async {
                    // 当 ViewModel 中 helloText 动态变量的值被设置或更改时
                    // 这段代码将被执行
                    self.helloLabel.text = helloText
                }
            })
        }
    }

    @IBAction func sayHelloButtonPressed(_ sender: UIButton) {
        viewModel?.userTriggeredSayHelloButton()
    }
}

```

更新的 MVVMEExampleViewModel :

```

class MVVMEExampleViewModel {

    // 我们必须用想要的数据类型初始化 Dynamic 变量
    var helloText = Dynamic("")

    func userTriggeredSayHelloButton() {
        // 设置 Dynamic 变量的值
        // 将触发我们在视图中定义的闭包
        helloText.value = "Hello"
    }
}

```

就是这样。你的ViewModel现在能够更新View，而无需持有对View的引用。

这是一个非常简单的示例，但我想你已经了解了它的强大之处。我不会详细讲解 MVVM 的好处，但一旦你从MVC切换到MVVM，就不会再回头。试试看，你会明白的。

```

class MVVMEExampleViewController: UIViewController {

    @IBOutlet weak var helloLabel: UILabel!

    var viewModel: MVVMEExampleViewModel?

    override func viewDidLoad() {
        super.viewDidLoad()
        bindViewModel()
    }

    func bindViewModel() {
        if let viewModel = viewModel {
            viewModel.helloText.bind({ (helloText) in
                DispatchQueue.main.async {
                    // When value of the helloText Dynamic variable
                    // is set or changed in the ViewModel, this code will
                    // be executed
                    self.helloLabel.text = helloText
                }
            })
        }
    }

    @IBAction func sayHelloButtonPressed(_ sender: UIButton) {
        viewModel?.userTriggeredSayHelloButton()
    }
}

```

Updated MVVMEExampleViewModel:

```

class MVVMEExampleViewModel {

    // we have to initialize the Dynamic var with the
    // data type we want
    var helloText = Dynamic("")

    func userTriggeredSayHelloButton() {
        // Setting the value of the Dynamic variable
        // will trigger the closure we defined in the View
        helloText.value = "Hello"
    }
}

```

That is it. Your ViewModel is now able to update View without it having a reference to the View.

This is a really simple example, but I think you have an idea how powerful this can be. I won't go into details about benefits of the MVVM, but once you switch from MVC to MVVM, you won't go back. Just try it and see for yourself.

第183章：缓存在线图片

第183.1节：AlamofireImage

使用AlamofireImage缓存在线图片。它基于Swift中的Alamofire。通过

cocoapods安装AlamofireImage

```
pod 'AlamofireImage', '~> 3.1'
```

设置：

1. 导入AlamofireImage和Alamofire
2. 设置图像缓存：`let imageCache = AutoPurgingImageCache(memoryCapacity: 111_111_111, preferredMemoryUsageAfterPurge: 90_000_000)`
3. 发起请求并将图像添加到缓存：

```
Alamofire.request(self.nameUrl[i]).responseImage { response in
    if response.result.value != nil {
        let image = UIImage(data: response.data!, scale: 1.0)!
        imageCache.add(image, withIdentifier: self.nameUrl[i])
    }
}
```

4. 从缓存中获取图像：

```
if let image = imageCache.image(withIdentifier: self.nameUrl[self.a])
{
    self.localImageView.image = image
}
```

更多信息请访问此链接[_____](#)

Chapter 183: Cache online images

Section 183.1: AlamofireImage

Caching Online Images Using AlamofireImage. It works on top of Alamofire in Swift. Install AlamofireImage using cocoapods

```
pod 'AlamofireImage', '~> 3.1'
```

SetUp:

1. Import AlamofireImage and Alamofire
2. SetUp the Image cache:`let imageCache = AutoPurgingImageCache(memoryCapacity: 111_111_111, preferredMemoryUsageAfterPurge: 90_000_000)`
3. Making a request and adding the Image to Cache:

```
Alamofire.request(self.nameUrl[i]).responseImage { response in
    if response.result.value != nil {
        let image = UIImage(data: response.data!, scale: 1.0)!
        imageCache.add(image, withIdentifier: self.nameUrl[i])
    }
}
```

4. Retrieve Images From Cache:

```
if let image = imageCache.image(withIdentifier: self.nameUrl[self.a])
{
    self.localImageView.image = image
}
```

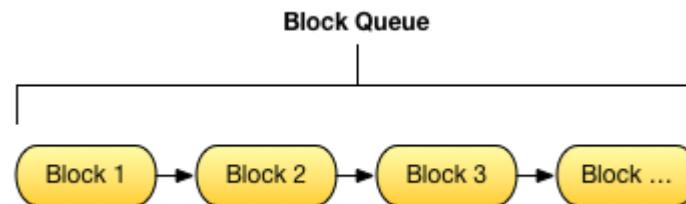
For more Info follow [this link](#)

第184章：队列中的链式块（使用MKBlockQueue）

MKBlockQueue允许你创建一个块链，并按顺序一个接一个地执行它们。与NSOperation相比，使用MKBlockQueue时，你可以自己决定一个块何时完成以及何时让队列继续。你还可以将数据从一个块传递到下一个块。

<https://github.com/MKGitHub/MKBlockQueue>

第184.1节：示例代码



```
// 创建将传递给块的字典
var myDictionary:Dictionary<String, Any> = Dictionary<String, Any>()
myDictionary["InitialKey"] = "InitialValue"

// 创建块队列
let myBlockQueue:MKBlockQueue = MKBlockQueue()

// 块1
let b1:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

        print("块1开始, 字典内容 : \(dictionary)")
        dictionary["Block1Key"] = "Block1Value"

        // 告诉该块已完成
        blockQueueObserver.blockCompleted(with:&dictionary)
}

// 块 2
let b2:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

        var copyOfDictionary:Dictionary<String, Any> = dictionary

        // 测试在主线程异步调用, 带延迟
        DispatchQueue.main.asyncAfter(deadline:(.now() + .seconds(1)), execute:
        {
            print("块 2 开始, 字典内容 : \(copyOfDictionary)")

            copyOfDictionary["Block2Key"] = "Block2Value"

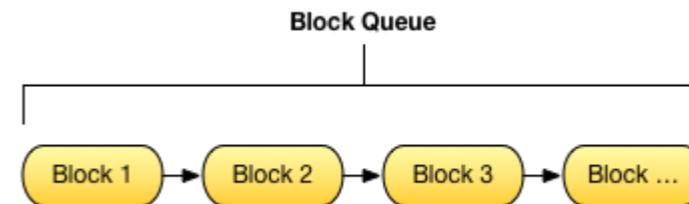
            // 告诉该块已完成
            blockQueueObserver.blockCompleted(with:&copyOfDictionary)
        })
}
```

Chapter 184: Chain Blocks in a Queue (with MKBlockQueue)

MKBlockQueue allows you to create a chain of blocks and execute them one after the other in a queue. Compared with NSOperation, with MKBlockQueue you decide yourself when a block is complete and when you want the queue to continue. You can also pass data from one block to the next.

<https://github.com/MKGitHub/MKBlockQueue>

Section 184.1: Example Code



```
// create the dictionary that will be sent to the blocks
var myDictionary:Dictionary<String, Any> = Dictionary<String, Any>()
myDictionary["InitialKey"] = "InitialValue"

// create block queue
let myBlockQueue:MKBlockQueue = MKBlockQueue()

// block 1
let b1:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

        print("Block 1 started with dictionary: \(dictionary)")
        dictionary["Block1Key"] = "Block1Value"

        // tell this block is now completed
        blockQueueObserver.blockCompleted(with:&dictionary)
}

// block 2
let b2:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

        var copyOfDictionary:Dictionary<String, Any> = dictionary

        // test calling on main thread, async, with delay
        DispatchQueue.main.asyncAfter(deadline:(.now() + .seconds(1)), execute:
        {
            print("Block 2 started with dictionary: \(copyOfDictionary)")

            copyOfDictionary["Block2Key"] = "Block2Value"

            // tell this block is now completed
            blockQueueObserver.blockCompleted(with:&copyOfDictionary)
        })
}
```

```

// 块 3
let b3:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in
        var copyOfDictionary:Dictionary<String, Any> = dictionary

    // 测试在全局后台队列上异步调用，带延迟
    DispatchQueue.global(qos:.background).asyncAfter(deadline:(.now() + .seconds(1)), execute:
    {
        print("Block 3 started with dictionary: \(copyOfDictionary)")

        copyOfDictionary["Block3Key"] = "Block3Value"

        // 告诉该块已完成
        blockQueueObserver.blockCompleted(with:&copyOfDictionary)
    })
}

// 向队列添加块
myBlockQueue.addBlock(b1)
myBlockQueue.addBlock(b2)
myBlockQueue.addBlock(b3)

// 为队列添加完成块
myBlockQueue.queueCompletedBlock(
{
    (dictionary:Dictionary<String, Any>) in
    print("Queue completed with dictionary: \(dictionary)")
})

// 运行队列
print("Queue starting with dictionary: \(myDictionary)")
myBlockQueue.run(with:&myDictionary)

```

```

// block 3
let b3:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in
        var copyOfDictionary:Dictionary<String, Any> = dictionary

    // test calling on global background queue, async, with delay
    DispatchQueue.global(qos:.background).asyncAfter(deadline:(.now() + .seconds(1)), execute:
    {
        print("Block 3 started with dictionary: \(copyOfDictionary)")

        copyOfDictionary["Block3Key"] = "Block3Value"

        // tell this block is now completed
        blockQueueObserver.blockCompleted(with:&copyOfDictionary)
    })
}

// add blocks to the queue
myBlockQueue.addBlock(b1)
myBlockQueue.addBlock(b2)
myBlockQueue.addBlock(b3)

// add queue completion block for the queue
myBlockQueue.queueCompletedBlock(
{
    (dictionary:Dictionary<String, Any>) in
    print("Queue completed with dictionary: \(dictionary)")
})

// run queue
print("Queue starting with dictionary: \(myDictionary)")
myBlockQueue.run(with:&myDictionary)

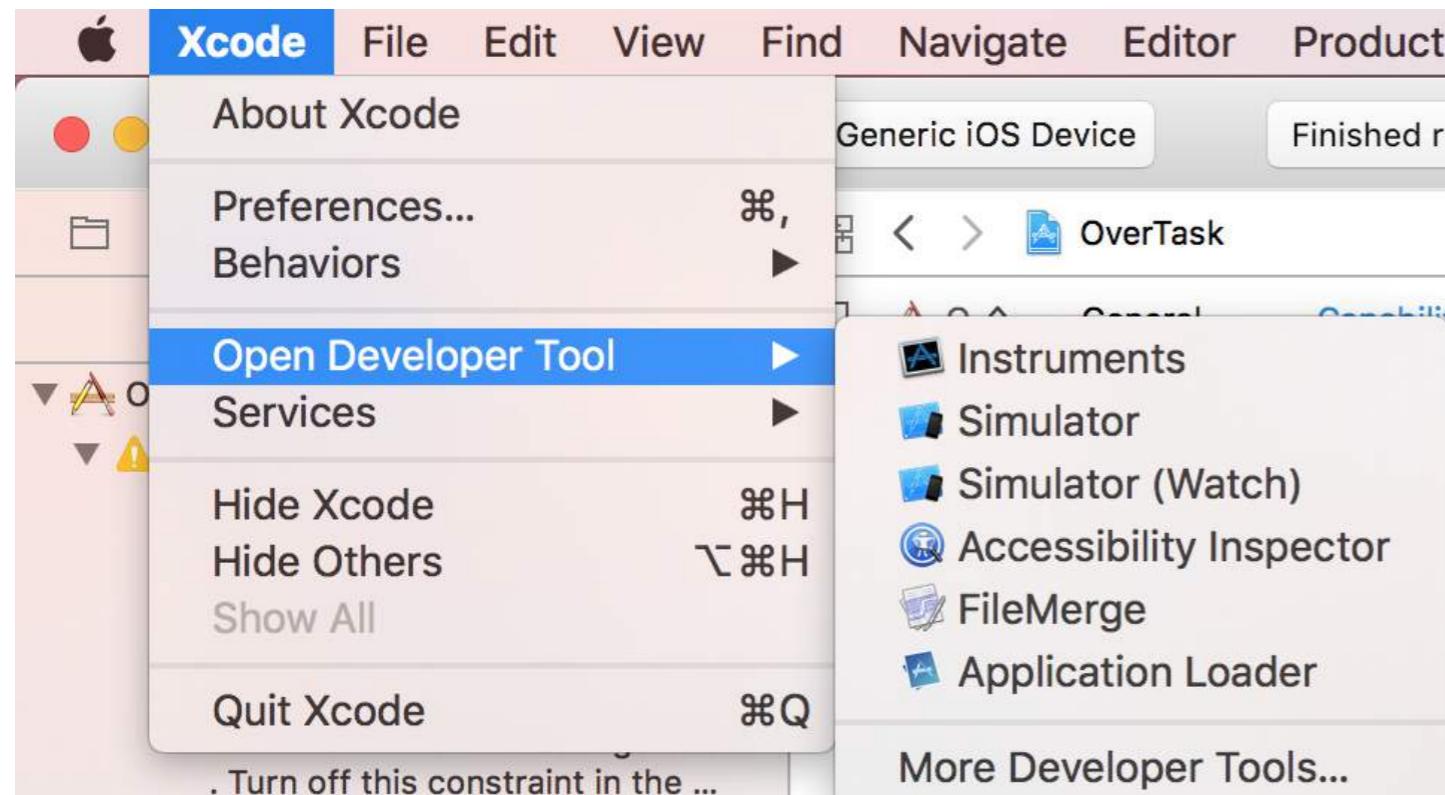
```

第185章：模拟器

iOS、watchOS 和 tvOS 模拟器是测试应用程序而无需使用真实设备的绝佳方式。这里我们将讨论如何使用模拟器。

第185.1节：启动模拟器

你应该进入Xcode -> 打开开发者工具：



使用“Simulator”，你可以运行iOS和tvOS，但“Simulator (Watch)”只能运行watchOS。

第185.2节：3D / 力触摸模拟

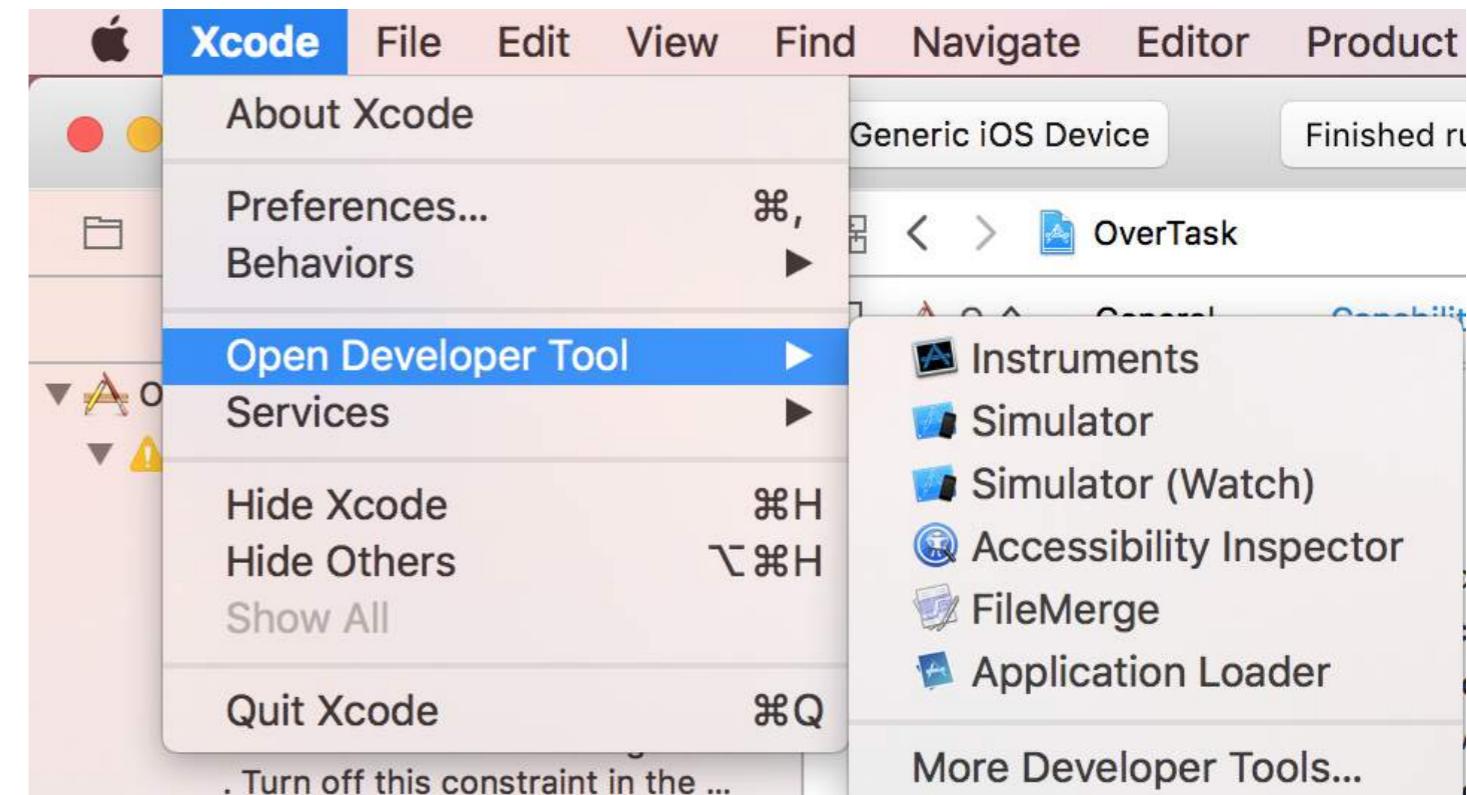
进入硬件 -> 触摸压力：

Chapter 185: Simulator

iOS, watchOS and tvOS Simulators are great ways to test your apps without using an actual device. Here we will talk about working with Simulators.

Section 185.1: Launching Simulator

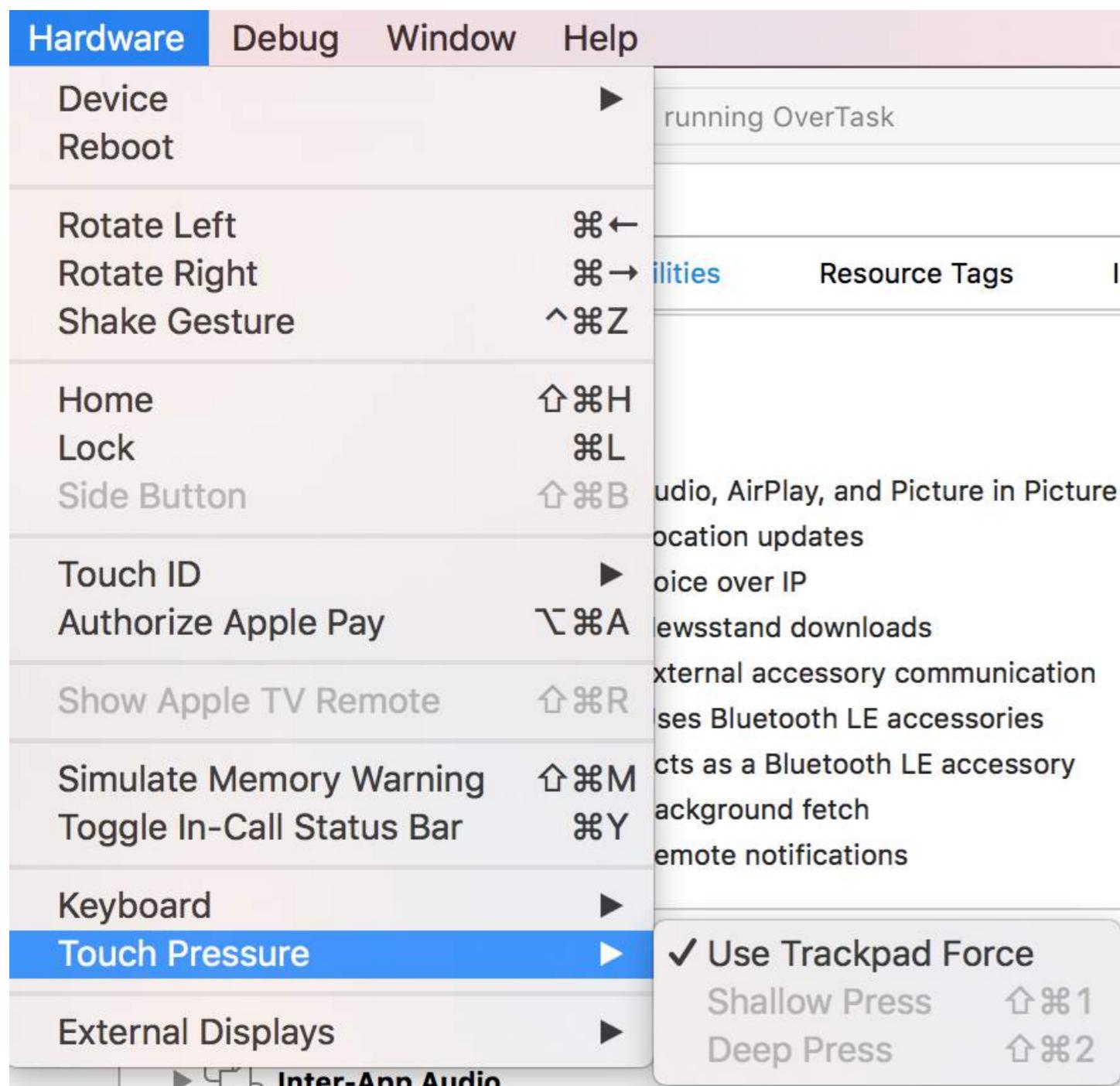
You should go to Xcode -> Open Developer Tool:



Using "Simulator", you could run iOS and tvOS, but "Simulator (Watch)" will only run watchOS.

Section 185.2: 3D / Force Touch simulation

Go to Hardware -> Touch Pressure:

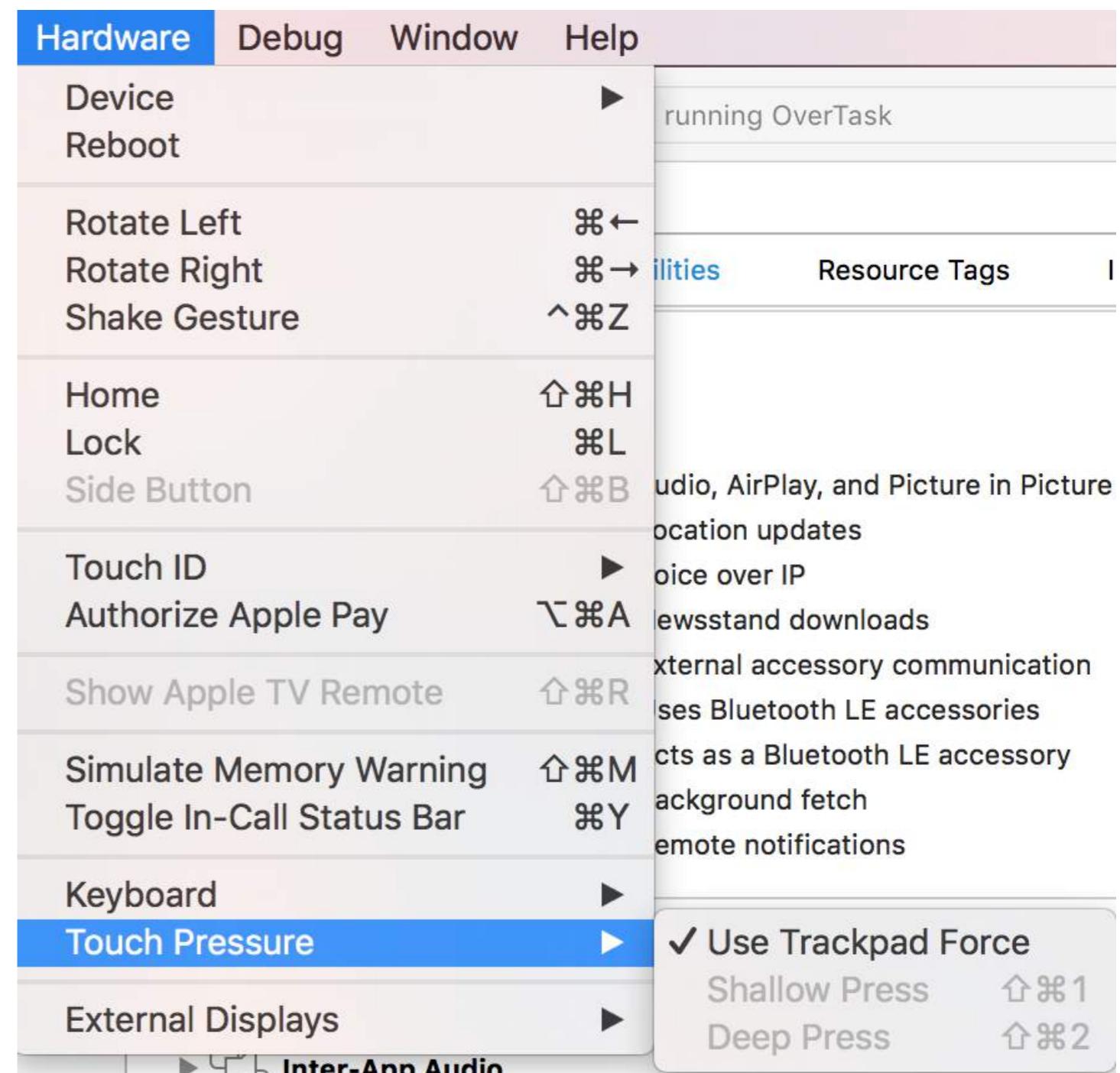


模拟普通按压，使用Shift-Command-1，模拟深度按压，使用Shift-Command-2。

如果你的MacBook配备了Force Touch触控板，你可以使用触控板的压力来模拟3D / 力触摸。

第185.3节：更改设备型号

进入硬件 -> 设备：

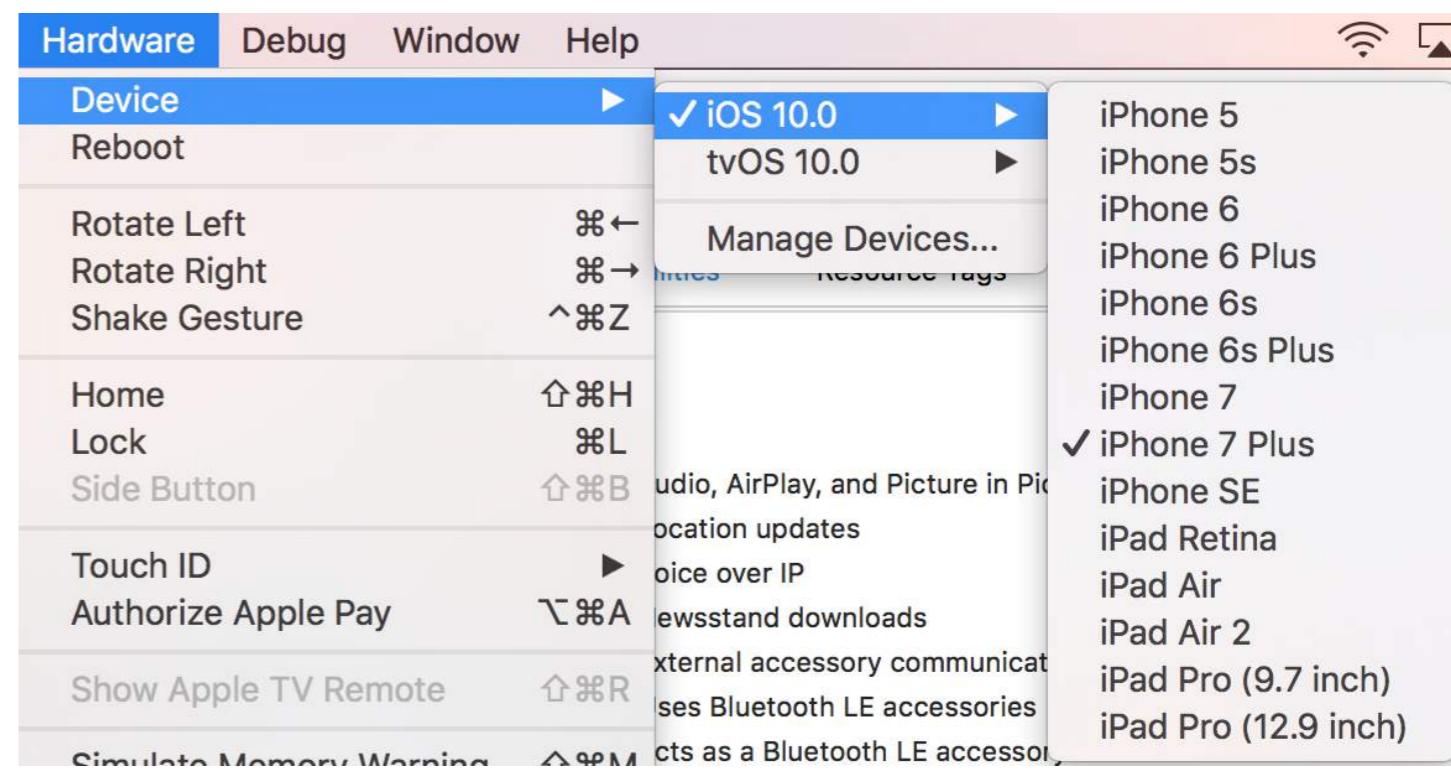
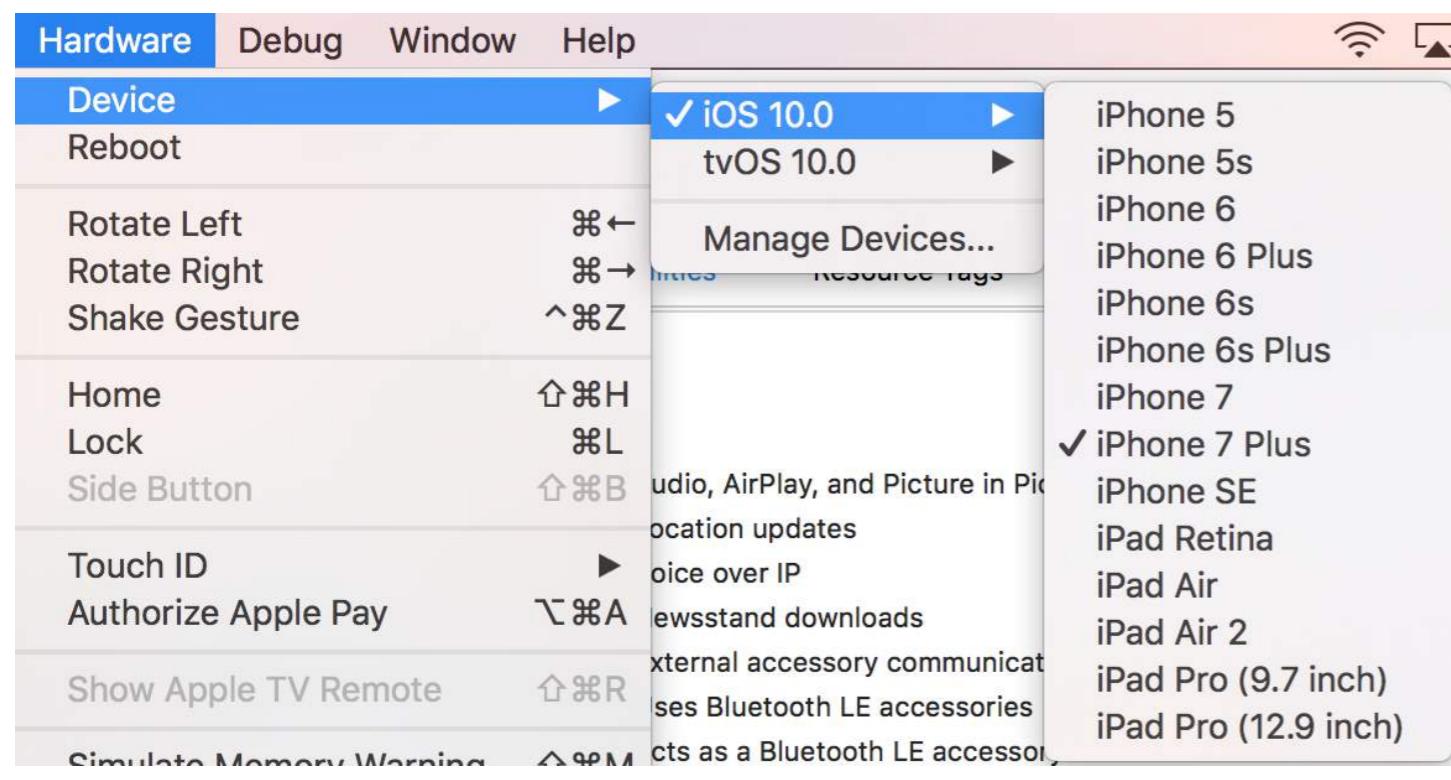


To simulate a normal press, use Shift-Command-1 and a deep one, Shift-Command-2.

If your MacBook has a Force Touch trackpad, you could use your trackpad force to simulate 3D / Force Touch.

Section 185.3: Change Device Model

Go to Hardware -> Device:



第185.4节：导航模拟器

主页按钮

关闭应用时应使用Shift-Command-H以显示动画效果。

锁定

要锁定设备，请使用Command-L。

旋转

使用Command键配合方向键。

Section 185.4: Navigating Simulator

Home button

You should use Shift-Command-H to have an animation when closing an app.

Lock

To lock the device, use Command-L.

Rotation

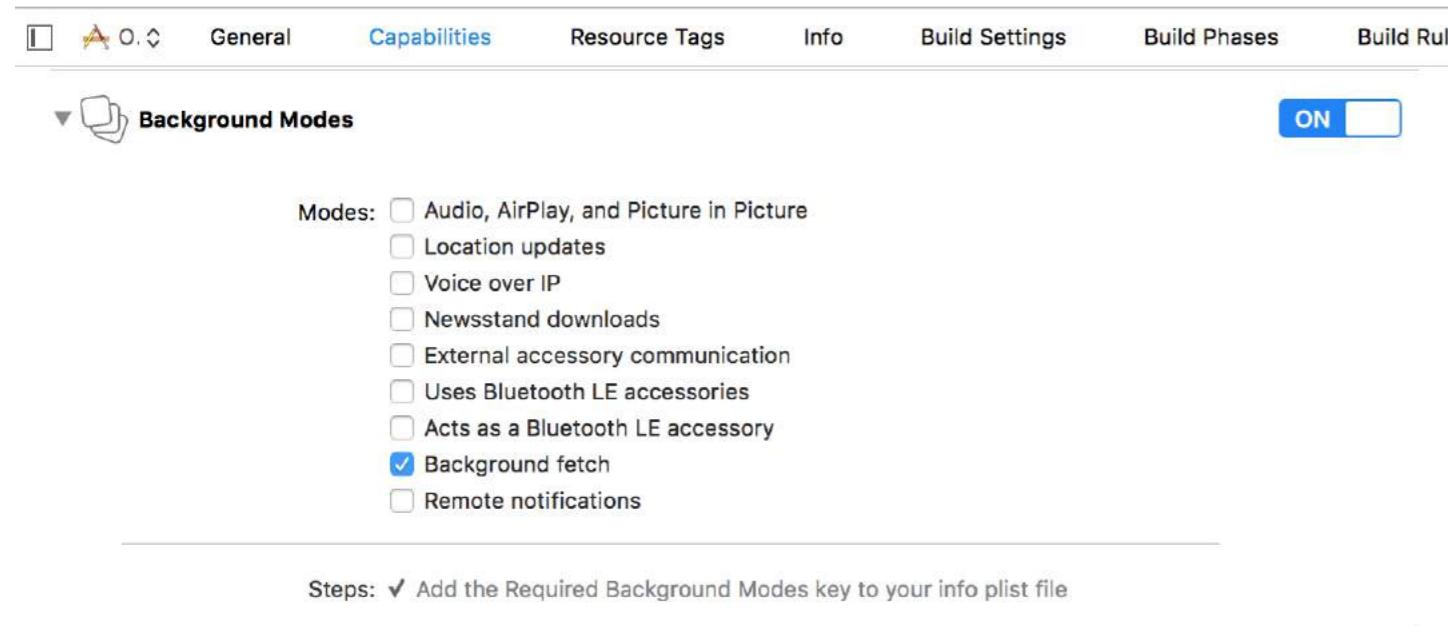
Use Command with arrow keys.

第186章：后台模式

响应迅速是每个应用程序的需求。用户希望在打开应用时内容已经准备好，因此开发者应使用后台模式使他们的应用更加用户友好。

第186.1节：开启后台模式功能

1. 打开Xcode并打开你的项目。
2. 在你的应用目标中，导航到“功能”标签页。
3. 开启后台模式。



第186.2节：后台获取

后台获取是一种新模式，允许你的应用始终保持最新信息，同时最大限度减少对电池的影响。你可以利用此功能在固定时间间隔内下载数据。

开始操作：

- 1- 在Xcode的功能界面中勾选“后台获取”。
- 2- 在AppDelegate中的application(_:didFinishLaunchingWithOptions:)方法中，添加：

Swift

```
UIApplication.shared.setMinimumBackgroundFetchInterval(UIBackgroundFetchIntervalMinimum)
```

Objective-C

```
[[UIApplication shared]
setMinimumBackgroundFetchInterval:UIBackgroundFetchIntervalMinimum]
```

你可以使用任何CGFloat值来设置

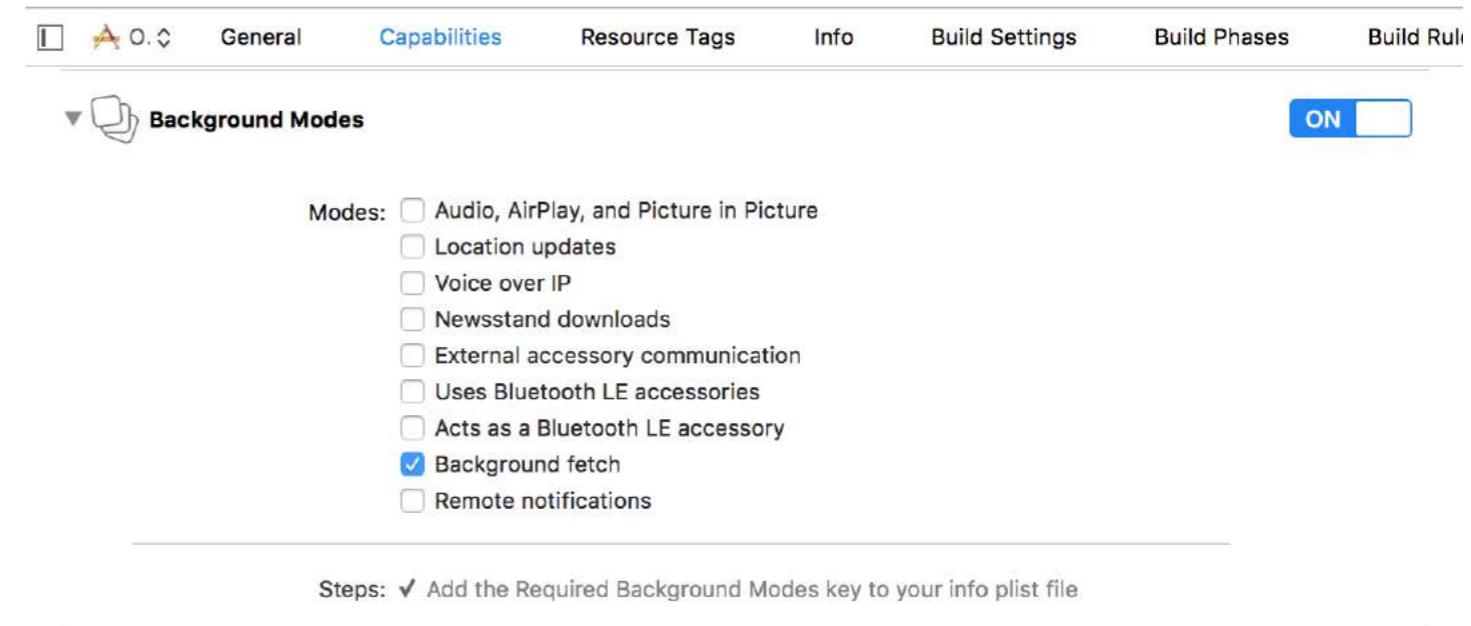
- 3- 你必须实现application(_:performFetchWithCompletionHandler:)。将其添加到你的AppDelegate中：

Chapter 186: Background Modes

Being responsive is a need for every app. Users want to have apps which have their content ready when they open it, so developers should use Background Modes to make their apps more user friendly.

Section 186.1: Turning on the Background Modes capability

1. Go to Xcode and open your project.
2. In your app target, navigate to Capabilities tab.
3. Turn on Background Modes.



Section 186.2: Background Fetch

Background fetch is a new mode that lets your app appear always up-to-date with the latest information while minimizing the impact on battery. You could download feeds within fixed time intervals with this capability.

To get started:

- 1- Check Background Fetch in capabilities screen in Xcode.
- 2- In application(_:didFinishLaunchingWithOptions:) method in AppDelegate, add:

Swift

```
UIApplication.shared.setMinimumBackgroundFetchInterval(UIBackgroundFetchIntervalMinimum)
```

Objective-C

```
[[UIApplication shared]
setMinimumBackgroundFetchInterval:UIBackgroundFetchIntervalMinimum]
```

Instead of UIBackgroundFetchIntervalMinimum, you could use any CGFloat value to set fetch intervals.

- 3- You must implement application(_:performFetchWithCompletionHandler:). Add that to your AppDelegate:

Swift

```
func application(_ application: UIApplication, performFetchWithCompletionHandler completionHandler:  
@escaping (UIBackgroundFetchResult) -> Void) {  
    // 你的代码写在这里  
}
```

第186.3节：测试后台获取

1- 在真机上运行应用并连接Xcode调试器。

2- 从调试菜单中选择模拟后台获取：

Swift

```
func application(_ application: UIApplication, performFetchWithCompletionHandler completionHandler:  
@escaping (UIBackgroundFetchResult) -> Void) {  
    // your code here  
}
```

Section 186.3: Testing background fetch

1- Run the app on a real device and attach it to Xcode debugger.

2- From Debug menu, select **Simulate Background Fetch**:

Debug Source Control Window Help

Pause ^⌘Y
Continue To Current Line ^⌘C
Step Over F6
Step Into F7
Step Out F8
Step Over Instruction ^ F6
Step Over Thread ^⇧F6
Step Into Instruction ^ F7
Step Into Thread ^⇧F7

Capture GPU Frame
GPU Overrides ▶
Simulate Location ▶
Simulate Background Fetch
Simulate UI Snapshot
iCloud ▶
View Debugging ▶

Deactivate Breakpoints ⌘Y
Breakpoints ▶

Debug Workflow ▶

Attach to Process by PID or Name...
Attach to Process ▶
Detach

Debug Source Control Window Help

Pause ^⌘Y
Continue To Current Line ^⌘C
Step Over F6
Step Into F7
Step Out F8
Step Over Instruction ^ F6
Step Over Thread ^⇧F6
Step Into Instruction ^ F7
Step Into Thread ^⇧F7

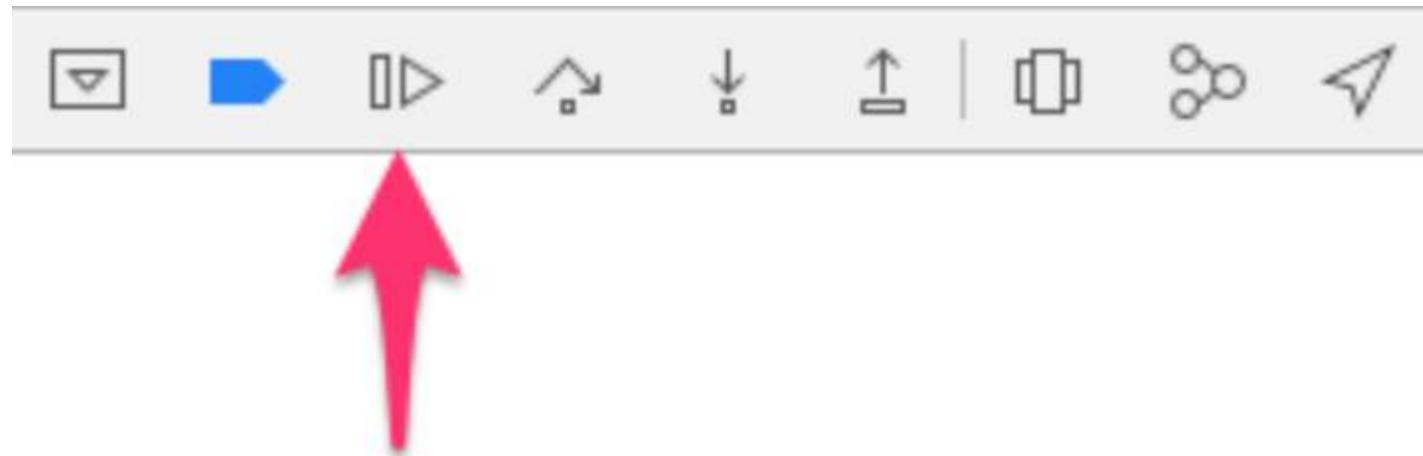
Capture GPU Frame
GPU Overrides ▶
Simulate Location ▶
Simulate Background Fetch
Simulate UI Snapshot
iCloud ▶
View Debugging ▶

Deactivate Breakpoints ⌘Y
Breakpoints ▶

Debug Workflow ▶

Attach to Process by PID or Name...
Attach to Process ▶
Detach

3- 现在 Xcode 会用 SIGSTOP 信号暂停应用。只需点击继续按钮，让应用执行后台获取。



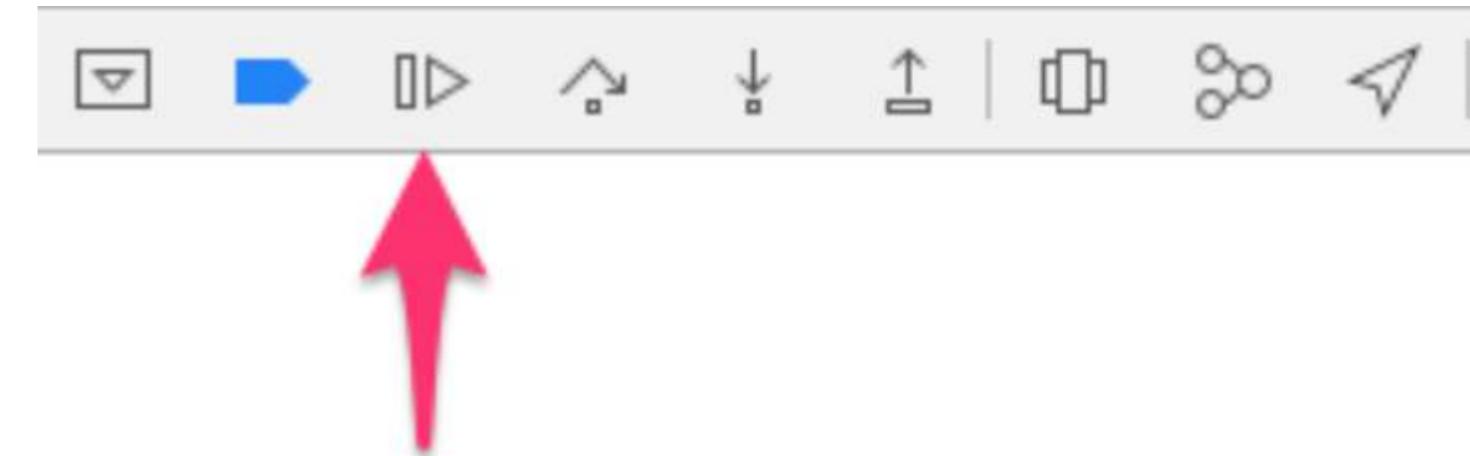
现在你会看到数据已被获取并准备好供你使用。

第186.4节：后台音频

默认情况下，当你在播放音频时，退出应用会停止播放，但你可以通过在 Xcode 的后台功能页面中勾选第一个复选框来防止这种情况发生。

iOS 会自动为你处理这个问题，你无需编写任何代码！

3- Now Xcode will pause the app with SIGSTOP signal. Just tap the continue button to let the app do the background fetch.



Now you will see that data is fetched and ready for you.

Section 186.4: Background Audio

By default, when you are streaming an audio, by exiting the app it will stop, but you can prevent this by turning on the first check box in Background capability page in Xcode.

iOS will automatically handle this for you, and you don't need to write any code!

第187章：OpenGL

OpenGL ES 是 iOS 用于进行3D渲染的图形库。

第187.1节：示例项目

一个示例项目 ([Git 仓库](#))，可用作进行一些3D渲染的起点。设置OpenGL和着色器的代码相当长且繁琐，因此不适合放在此示例格式中。后续部分可以在单独的示例中展示，详细说明每段代码的具体作用，但现在这里是Xcode项目。

Chapter 187: OpenGL

OpenGL ES is a graphics library that iOS uses to do 3D rendering.

Section 187.1: Example Project

An [example project \(Git repo\)](#) that can be used as a starting point for doing some 3D rendering. The code for setting up OpenGL and the shaders is quite long and tedious so it won't fit well under this example format. Later pieces of it can be shown in separate examples detailing what exactly is going on with each piece of code, but for now here is the Xcode project.

第188章：MVP架构

组件：



- 模型 是负责领域数据的接口（用于在GUI中显示或其他操作）
- 视图 负责表现层（GUI）
- 演示者 是模型和视图之间的“中间人”。它响应用户在视图上执行的操作，从模型中检索数据，并将其格式化以在视图中显示

组件职责：

模型	视图	演讲者
与数据库层通信，呈现数据		对模型执行查询
触发适当的事件	接收事件	格式化模型数据
		非常基础的验证逻辑，发送格式化数据到视图
		复杂的验证逻辑

MVC和MVP的区别：

- MVC中的视图与控制器紧密耦合，MVP的视图部分由UIViews和UIViewController组成
- MVP视图尽可能简单，几乎不包含逻辑（类似MVVM），MVC视图包含一些业务逻辑并且可以查询模型
- MVP视图处理用户手势并将交互委托给Presenter，MVC中控制器处理手势并命令模型
- MVP模式高度支持单元测试，MVC支持有限
- MVC控制器依赖大量UIKit，而MVP的Presenter没有依赖

优点：

- MVP使UIViewController成为视图组件的一部分，它是简单的、被动的且.....不那么庞大 ;]由于视图简
- 单，大部分业务逻辑被封装，这带来了极佳的可测试性。可以引入模拟对象来测试领域部分。
- 分离的实体更容易理解，职责划分清晰。

缺点

- 你需要编写更多代码。
- 对于没有经验的开发者或尚未使用该模式的人来说是一个障碍。

Chapter 188: MVP Architecture

Components:



- **Model** is an interface responsible for the domain data (to be displayed or otherwise acted upon in the GUI)
- **View** is responsible for the presentation layer (GUI)
- **Presenter** is the "middle-man" between Model and View. It reacts to the user's actions performed on the View, retrieves data from the Model, and formats it for display in the View

Component duties:

Model	View	Presenter
Communicates with DB layer	Renders data	Performs queries to the Model
Raising appropriate events	Receives events	Formats data from Model
		Very basic validation logic
		Sends formatted data to the View
		Complex validation logic

Differences between MVC and MVP:

- View in MVC is tightly coupled with the Controller, the View part of the MVP consists of both UIViews and UIViewController
- MVP View is as dumb as possible and contains almost no logic (like in MVVM), MVC View has some business logic and can query the Model
- MVP View handles user gestures and delegates interaction to the Presenter, in MVC the Controller handles gestures and commands Model
- MVP pattern highly supports Unit Testing, MVC has limited support
- MVC Controller has lots of UIKit dependencies, MVP Presenter has none

Pros:

- MVP makes UIViewController a part of the View component it's dumb, passive and...less massive ;]
- Most of the business logic is encapsulated due to the dumb Views, this gives an excellent testability. Mock objects can be introduced to test the domain part.
- Separated entities are easier to keep in head, responsibilities are clearly divided.

Cons

- You will write more code.
- Barrier for unexperienced developers or for those who don't yet work with the pattern.

MVP是一种架构模式，是模型-视图-控制器（Model-View-Controller）的衍生。它由三个独立的组件组成：模型、视图和Presenter。它的设计目的是便于自动化单元测试并改善表现逻辑中的关注点分离。

示例中你会找到一个以MVP模式构建的简单项目。

第188.1节：Dog.swift

```
import Foundation

enum Breed: String {
    case 牛头犬 = "Bulldog"
    case 杜宾犬 = "Doberman"
    case 拉布拉多 = "Labrador"
}

struct Dog {
    let name: String
    let breed: String
    let age: Int
}
```

第188.2节：DoggyService.swift

```
import Foundation

typealias Result = ([Dog]) -> Void

class DoggyService {

    func deliverDoggies(_ result: @escaping Result) {

        let firstDoggy = Dog(name: "阿尔弗雷德", breed: Breed.拉布拉多.rawValue, age: 1)
        let secondDoggy = Dog(name: "维尼", breed: Breed.杜宾犬.rawValue, age: 5)
        let thirdDoggy = Dog(name: "幸运", breed: Breed.拉布拉多.rawValue, age: 3)

        let delay = DispatchTime.now() + Double(Int64(Double(NSEC_PER_SEC)*2)) /
Double(NSEC_PER_SEC)

        DispatchQueue.main.asyncAfter(deadline: delay) {
            result([firstDoggy,
secondDoggy,
thirdDoggy])
        }
    }
}
```

第188.3节：DoggyPresenter.swift

```
import Foundation

class DoggyPresenter {

    // 标记: - 私有
fileprivate let dogService: DoggyService
weak fileprivate var dogView: DoggyView?

init(dogService: DoggyService){
```

MVP is an architectural pattern, a derivation of the Model-View-Controller. It's represented by three distinct components: Model, View and the Presenter. It was engineered to facilitate automated unit testing and improve the separation of concerns in presentation logic.

In examples you'll find a simple project built with MVP pattern in mind.

Section 188.1: Dog.swift

```
import Foundation

enum Breed: String {
    case bulldog = "Bulldog"
    case doberman = "Doberman"
    case labrador = "Labrador"
}

struct Dog {
    let name: String
    let breed: String
    let age: Int
}
```

Section 188.2: DoggyService.swift

```
import Foundation

typealias Result = ([Dog]) -> Void

class DoggyService {

    func deliverDoggies(_ result: @escaping Result) {

        let firstDoggy = Dog(name: "Alfred", breed: Breed.labrador.rawValue, age: 1)
        let secondDoggy = Dog(name: "Vinny", breed: Breed.doberman.rawValue, age: 5)
        let thirdDoggy = Dog(name: "Lucky", breed: Breed.labrador.rawValue, age: 3)

        let delay = DispatchTime.now() + Double(Int64(Double(NSEC_PER_SEC)*2)) /
Double(NSEC_PER_SEC)

        DispatchQueue.main.asyncAfter(deadline: delay) {
            result([firstDoggy,
secondDoggy,
thirdDoggy])
        }
    }
}
```

Section 188.3: DoggyPresenter.swift

```
import Foundation

class DoggyPresenter {

    // MARK: - Private
fileprivate let dogService: DoggyService
weak fileprivate var dogView: DoggyView?

init(dogService: DoggyService){
```

```

        self.dogService = dogService
    }

    func attachView(_ attach: Bool, view: DoggyView?) {
        if attach {
            dogView = nil
        } else {
            if let view = view { dogView = view }
        }
    }

    func getDogs(){
        self.dogView?.startLoading()

        dogService.deliverDoggies { [weak self] doggies in
            self?.dogView?.finishLoading()

            if doggies.count == 0 {
                self?.dogView?.setEmpty()
            } else {
                self?.dogView?.setDoggies(doggies.map {
                    return DoggyViewData(name: "($0.name) \($0.breed)",
                                         age: "($0.age)")
                })
            }
        }
    }

    struct DoggyViewData {
        let name: String
        let age: String
    }
}

```

第188.4节：DoggyView.swift

```

import Foundation

协议 DoggyView: NSObjectProtocol {
    func startLoading()
    func finishLoading()
    func setDoggies(_ doggies: [DoggyViewData])
    func setEmpty()
}

```

第188.5节：DoggyListViewController.swift

```

import UIKit

类 DoggyListViewController: UIViewController, UITableViewDataSource {

    @IBOutlet 弱引用 var emptyView: UIView?
    @IBOutlet 弱引用 var tableView: UITableView?
    @IBOutlet 弱引用 var spinner: UIActivityIndicatorView?

    fileprivate let dogPresenter = DoggyPresenter(dogService: DoggyService())
    fileprivate var dogsToDisplay = [DoggyViewData]()

    override func viewDidLoad() {
        super.viewDidLoad()
}

```

```

        self.dogService = dogService
    }

    func attachView(_ attach: Bool, view: DoggyView?) {
        if attach {
            dogView = nil
        } else {
            if let view = view { dogView = view }
        }
    }

    func getDogs(){
        self.dogView?.startLoading()

        dogService.deliverDoggies { [weak self] doggies in
            self?.dogView?.finishLoading()

            if doggies.count == 0 {
                self?.dogView?.setEmpty()
            } else {
                self?.dogView?.setDoggies(doggies.map {
                    return DoggyViewData(name: "($0.name) \($0.breed)",
                                         age: "($0.age)")
                })
            }
        }
    }

    struct DoggyViewData {
        let name: String
        let age: String
    }
}

```

Section 188.4: DoggyView.swift

```

import Foundation

protocol DoggyView: NSObjectProtocol {
    func startLoading()
    func finishLoading()
    func setDoggies(_ doggies: [DoggyViewData])
    func setEmpty()
}

```

Section 188.5: DoggyListViewController.swift

```

import UIKit

类 DoggyListViewController: UIViewController, UITableViewDataSource {

    @IBOutlet 弱引用 var emptyView: UIView?
    @IBOutlet 弱引用 var tableView: UITableView?
    @IBOutlet 弱引用 var spinner: UIActivityIndicatorView?

    fileprivate let dogPresenter = DoggyPresenter(dogService: DoggyService())
    fileprivate var dogsToDisplay = [DoggyViewData]()

    override func viewDidLoad() {
        super.viewDidLoad()
}

```

```

tableView?.dataSource = self
    spinner?.hidesWhenStopped = true
    dogPresenter.attachView(true, view: self)
    dogPresenter.getDogs()
}

// 标记: 数据源
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return dogsToDisplay.count
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = UITableViewCell(style: .subtitle, reuseIdentifier: "Cell")
    let userViewData = dogsToDisplay[indexPath.row]
    cell.textLabel?.text = userViewData.name
    cell.detailTextLabel?.text = userViewData.age
    return cell
}

extension DoggyListViewController: DoggyView {

    func startLoading() {
        spinner?.startAnimating()
    }

    func finishLoading() {
        spinner?.stopAnimating()
    }

    func setDoggies(_ doggies: [DoggyViewData]) {
        dogsToDisplay = doggies
        tableView?.isHidden = false
        emptyView?.isHidden = true;
        tableView?.reloadData()
    }

    func setEmpty() {
        tableView?.isHidden = true
        emptyView?.isHidden = false;
    }
}

```

```

tableView?.dataSource = self
    spinner?.hidesWhenStopped = true
    dogPresenter.attachView(true, view: self)
    dogPresenter.getDogs()
}

// MARK: DataSource
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return dogsToDisplay.count
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = UITableViewCell(style: .subtitle, reuseIdentifier: "Cell")
    let userViewData = dogsToDisplay[indexPath.row]
    cell.textLabel?.text = userViewData.name
    cell.detailTextLabel?.text = userViewData.age
    return cell
}

extension DoggyListViewController: DoggyView {

    func startLoading() {
        spinner?.startAnimating()
    }

    func finishLoading() {
        spinner?.stopAnimating()
    }

    func setDoggies(_ doggies: [DoggyViewData]) {
        dogsToDisplay = doggies
        tableView?.isHidden = false
        emptyView?.isHidden = true;
        tableView?.reloadData()
    }

    func setEmpty() {
        tableView?.isHidden = true
        emptyView?.isHidden = false;
    }
}

```

第189章：使用CoreBluetooth配置信标

如何读取和写入蓝牙低功耗设备的数据。

第189.1节：显示所有蓝牙低功耗

(BLE) 的名称

- 在此示例中，我有一个受控房间，启用了单个BLE设备。
- 您的类应继承CBCentralManagerDelegate。
- 实现方法：centralManagerDidUpdateState(_ central: CBCentralManager)。
- 使用全局队列以避免在搜索设备时屏幕冻结。
- 实例化CBCentralManager并等待回调centralManagerDidUpdateState响应。

```
class BLEController: CBCentralManagerDelegate{

var cb_manager: CBCentralManager!
var bles : [CBPeripheral] = []

override func viewDidLoad() {
    super.viewDidLoad()
cb_manager = CBCentralManager(delegate: self, queue: DispatchQueue.global())
}

func centralManagerDidUpdateState(_ central: CBCentralManager) {
    print("更新状态 - \(central)")
}
}
```

回调 centralManagerDidUpdateState 表示 CoreBluetooth 已准备好，因此您现在可以搜索 BLE 设备。

更新 centralManagerDidUpdateState 代码，在准备好时搜索所有 BLE 设备。

```
func centralManagerDidUpdateState(_ central: CBCentralManager) {
    print("更新状态 - \(central)")
SearchBLE()
}

func SearchBLE(){
cb_manager.scanForPeripherals(withServices: nil, options: nil)
    StopSearchBLE()
}

func StopSearchBLE() {
    let when = DispatchTime.now() + 5 // 将 5 改为所需的秒数
DispatchQueue.main.asyncAfter(deadline: when) {
    self.cb_manager.stopScan()
}
}
```

- SearchBLE() 用于搜索 BLE 设备，并在 5 秒后停止搜索
- cb_manager.scanForPeripherals(withServices: nil, options: nil) 会搜索范围内的所有 BLE 设备。
- StopSearchBLE() 会在 5 秒后停止搜索。
- 每发现一个 BLE 设备，都会回调函数 centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber)

```
func centralManager(_ central: CBCentralManager, didDiscover peripheral:
```

Chapter 189: Configure Beacons with CoreBluetooth

Hot to read and write data to a bluetooth low energy device.

Section 189.1: Showing names of all Bluetooth Low Energy (BLE)

- For this example I have a controlled room with a single BLE device enable.
- Your class should extend CBCentralManagerDelegate.
- Implement the method: centralManagerDidUpdateState(_ central: CBCentralManager).
- Use global queue to not freeze the screen while searching for a device.
- Instantiate CBCentralManager and wait for callback centralManagerDidUpdateState response.

```
class BLEController: CBCentralManagerDelegate{

var cb_manager: CBCentralManager!
var bles : [CBPeripheral] = []

override func viewDidLoad() {
    super.viewDidLoad()
    cb_manager = CBCentralManager(delegate: self, queue: DispatchQueue.global())
}

func centralManagerDidUpdateState(_ central: CBCentralManager) {
    print("UPDATE STATE - \(central)")
}
}
```

Callback to centralManagerDidUpdateState indicates that CoreBluetooth is ready, so you can search for BLE now. Update centralManagerDidUpdateState code to search for all BLE device when it is ready.

```
func centralManagerDidUpdateState(_ central: CBCentralManager) {
    print("UPDATE STATE - \(central)")
    SearchBLE()
}

func SearchBLE(){
    cb_manager.scanForPeripherals(withServices: nil, options: nil)
    StopSearchBLE()
}

func StopSearchBLE() {
    let when = DispatchTime.now() + 5 // change 5 to desired number of seconds
    DispatchQueue.main.asyncAfter(deadline: when) {
        self.cb_manager.stopScan()
    }
}
```

- SearchBLE() search for BLE devices and stop searching after 5s
- cb_manager.scanForPeripherals(withServices: nil, options: nil) looks for every BLE in range with you.
- StopSearchBLE() will stop the search after 5s.
- Each BLE found will callback func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber)

```
func centralManager(_ central: CBCentralManager, didDiscover peripheral:
```

```

CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber) {
    guard let name = peripheral.name else {
        返回
    }
    print(name)
    bles.append(peripheral)
}

```

第189.2节：连接并读取major值

- 我在一个受控房间内，只有一个使用IBEACON协议的minew信标。
- BLEController需要扩展CBPeripheralDelegate
- 我将在搜索停止后使用第一个BLE进行连接。
- 修改方法StopSearchBLE()

```

class BLEController: CBCentralManagerDelegate, CBPeripheralDelegate{
//...
func StopSearchMiniewBeacon() {
    let when = DispatchTime.now() + 5 // 将2改为所需的秒数
    DispatchQueue.main.asyncAfter(deadline: when) {
        self.cb_manager.stopScan()
        self.cb_manager.connect(bles.first)
    }
//...
}

```

- 在您的BLE设备文档中，您应查找服务UUID和主UUID特征

```

var service_uuid = CBUUID(string: "0000fff0-0000-1000-8000-00805f9b34fb")
var major_uuid = CBUUID(string: "0000fff2-0000-1000-8000-00805f9b34fb")
func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
    peripheral.delegate = self
    peripheral.discoverServices([service_uuid])
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {print("服务: \(service_uuid)\n错误: \(error)")
peripheral.discoverCharacteristics([major_uuid], for: (peripheral.services?[0])!)
}

```

- 创建变量 'service_uuid' 和 'major_uuid'，如上面的代码。'-0000-1000-8000-00805f9b34fb' 是标准的一部分。`'fff0'` 是我的服务 UUID，`'fff2'` 是我的主 UUID 特征，`'0000'` 是填充4字节 UUID 所需的部分。
- 调用 `discoverCharacteristics([major_uuid], for: (peripheral.services?[0])!)` 会从我的设备 GATT 服务器获取主特征，目前其值为 NIL。
- `(peripheral.services?[0])!` - 这里是 0，因为在调用 `peripheral.discoverServices([service_uuid])` 后会返回单个值。

```

func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService, error: Error?) {
    for characteristic in service.characteristics! {print("特征: \(characteristic)\n错误: \(error)"if(characteristic.uuid.uuidString == "FFF2"){
        peripheral.readValue(for: characteristic)
    }
}

```

```

CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber) {
    guard let name = peripheral.name else {
        return
    }
    print(name)
    bles.append(peripheral)
}

```

Section 189.2: Connect and read major value

- I'm in a controlled room with a single minew beacon that use IBEACON protocol.
- BLEController needs to extend CBPeripheralDelegate
- I'll use the first BLE to connect after the search has stop.
- Modify the method StopSearchBLE()

```

class BLEController: CBCentralManagerDelegate, CBPeripheralDelegate{
//...
func StopSearchMiniewBeacon() {
    let when = DispatchTime.now() + 5 // change 2 to desired number of seconds
    DispatchQueue.main.asyncAfter(deadline: when) {
        self.cb_manager.stopScan()
        self.cb_manager.connect(bles.first)
    }
}
//...
}

```

- In the documentation of your BLE device, you should look for the SERVICE UUID and MAJOR UUID CHARACTERISTIC

```

var service_uuid = CBUUID(string: "0000fff0-0000-1000-8000-00805f9b34fb")
var major_uuid = CBUUID(string: "0000fff2-0000-1000-8000-00805f9b34fb")
func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
    peripheral.delegate = self
    peripheral.discoverServices([service_uuid])
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
    print("Service: \(service_uuid)\n error: \(error)")
    peripheral.discoverCharacteristics([major_uuid], for: (peripheral.services?[0])!)
}

```

- Create a variable 'service_uuid' and 'major_uuid' like code above. '-0000-1000-8000-00805f9b34fb' is part of the standard. 'fff0' is my SERVICE UUID, 'fff2' is my MAJOR UUID characteristic and '0000' are required to fill the 4 bytes uuid 1° block.
- `discoverCharacteristics([major_uuid], for: (peripheral.services?[0])!)` will get major characteristic from my device gatt server and it will have NIL as value for now.
- `(peripheral.services?[0])!` - 0 because will return a single value once I did `peripheral.discoverServices([service_uuid])`

```

func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService, error: Error?) {
    for characteristic in service.characteristics! {
        print("Characteristic: \(characteristic)\n error: \(error)")
        if(characteristic.uuid.uuidString == "FFF2"){
            peripheral.readValue(for: characteristic)
        }
}

```

```

    }

}

func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("读取特征: \(characteristic) 错误: \(error)")let major = UInt16.init(bigEndian:
    UInt16(data: characteristic.value!)!)print("major: \(major)")

}

```

- 特征值只有在调用 `peripheral.readValue(for: characteristic)` 后才能读取
- `readValue` 的结果会触发 `func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?)`，错误：带有数据类型值的错误（`Error?`）。

第189.3节：写入主值

- 您需要发现服务和特征
- 写入特征之前无需读取其值。
- 将继续，例如，在读取值之后。修改函数 `peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?)`
- 添加变量 `new_major` 和 `reset_characteristic`

```

var reset_characteristic : CBCharacteristic!
func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService,
error: Error?) {
    for characteristic in service.characteristics! {print("特征: \(c
    haracteristic) 错误: \(error)")if(characteristic.uuid.uuidString == "FFF2"){

peripheral.readValue(for: characteristic)
    }
    if(characteristic.uuid.uuidString == "FFFF"){
        reset_characteristic = characteristic
    }
}
let new_major : UInt16 = 100
func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("读取特征: \(characteristic) 错误: \(error)")let major = UInt16.init(bigEndian:
    UInt16(data: characteristic.value!)!)print("major: \(major)")

peripheral.writeValue(new_major.data, for: characteristic, type:
CBCharacteristicWriteType.withResponse)
}

```

- iPhone 默认以小端格式发送和接收字节，但我的设备 MINEW 使用的芯片组 NRF51822 采用 ARM 架构，需要大端格式的字节，因此我必须进行字节交换。
- BLE 设备文档会说明每个特征的输入和输出类型，以及是否可以像上面那样读取 (`CBCharacteristicWriteType.wi
thResponse`)。

```

func peripheral(_ peripheral: CBPeripheral, didWriteValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("特征写入: \(characteristic) 错误: \(error)")if(characteristic.uuid.uuidString ==
    "FFF2"){
        print("重置中")
peripheral.writeValue("minew123".data(using: String.Encoding.utf8)!, for:
reset_characteristic, type: CBCharacteristicWriteType.withResponse)
    }
    if(characteristic.uuid.uuidString == "FFFF"){

}

```

```

    }

}

func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic read: \(characteristic)\n error: \(error)")
    let major = UInt16.init(bigEndian: UInt16(data: characteristic.value!)!)
    print("major: \(major)")

}

```

- Characteristic value will only be readable after call `peripheral.readValue(for: characteristic)`
- `readValue` will result in `func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?)` with value in Data type.

Section 189.3: Write major value

- You need discover the services and characteristic
- You don't need read value from the characteristic before writing over it.
- will continue for, for this example, after read value. Modify `func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?)`
- Add a variable `new_major` and `reset_characteristic`

```

var reset_characteristic : CBCharacteristic!
func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService,
error: Error?) {
    for characteristic in service.characteristics! {
        print("Characteristic: \(characteristic)\n error: \(error)")
        if(characteristic.uuid.uuidString == "FFF2"){
            peripheral.readValue(for: characteristic)
        }
        if(characteristic.uuid.uuidString == "FFFF"){
            reset_characteristic = characteristic
        }
    }
}
let new_major : UInt16 = 100
func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic read: \(characteristic)\n error: \(error)")
    let major = UInt16.init(bigEndian: UInt16(data: characteristic.value!)!)
    print("major: \(major)")
    peripheral.writeValue(new_major.data, for: characteristic, type:
CBCharacteristicWriteType.withResponse)
}

```

- iPhone by default will send and receive bytes in Little Endian format, but my device MINEW witch chipset NRF51822 have ARM architecture and need bytes in Big Endian format, so I have to swap it.
- BLE Device documentation will say what type of input and output each characteristic will have and if you can read it like above (`CBCharacteristicWriteType.withResponse`).

```

func peripheral(_ peripheral: CBPeripheral, didWriteValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic write: \(characteristic)\n error: \(error)")
    if(characteristic.uuid.uuidString == "FFF2"){
        print("Resetting")
        peripheral.writeValue("minew123".data(using: String.Encoding.utf8)!, for:
reset_characteristic, type: CBCharacteristicWriteType.withResponse)
    }
    if(characteristic.uuid.uuidString == "FFFF"){

}

```

```

    print("重启完成")
    cb_manager.cancelPeripheralConnection(peripheral)
}
}

```

- 要更新gatt服务器信息，您必须通过编程方式重启它，或者保存数据到服务器后手动关闭并重新开启。
- FFFF 是该设备的特征。
- “minew123” 是在此情况下重启以保存信息的默认密码。
- 运行你的应用并查看控制台是否有错误，希望没有，但你还看不到新值。

```

func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("读取特征: \(characteristic) 错误: \(error)")let major = UInt16.init(bigEndian:
    UInt16(data: characteristic.value!)!)print("major: \(major)")

    //peripheral.writeValue(new_major.data, for: characteristic, type:
    CBCharacteristicWriteType.withResponse)

}

```

- 最后一步是在 didUpdateValueFor 方法中注释掉最后一行代码并重新运行应用，现在你将看到新值。

```

    print("Reboot finish")
    cb_manager.cancelPeripheralConnection(peripheral)
}
}

```

- To update a gatt server information you have to reboot it programmatically or save data to it and turn off and turn on manually.
- FFFF is characteristic that do it in this device.
- 'minew123' is the default password for reboot o save information in this case.
- run your app and watch you console for any error, I hope none, but you will not see the new value yet.

```

func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic read: \(characteristic)\n error: \(error)")
    let major = UInt16.init(bigEndian: UInt16(data: characteristic.value!)!)
    print("major: \(major)")
    //peripheral.writeValue(new_major.data, for: characteristic, type:
    CBCharacteristicWriteType.withResponse)

}

```

- Last step is to comment last line in method didUpdateValueFor and rerun the app, now you will see the new value.

第190章：核心数据

核心数据是你应用程序的模型层，涵盖了最广泛的意义。它是贯穿 iOS SDK 的模型-视图-控制器模式中的模型部分。

核心数据不是你的应用程序的数据库，也不是用于将数据持久化到数据库的 API。核心数据是一个管理对象图的框架。就是这么简单。核心数据可以通过写入磁盘来持久化该对象图，但这不是该框架的主要目标。

第190.1节：核心数据操作

获取上下文：

```
NSManagedObjectContext *context = ((AppDelegate*)[[UIApplication sharedApplication] delegate]).persistentContainer.viewContext;
```

获取数据：

```
NSFetchRequest<EntityName *> *fetchRequest = [EntityName fetchRequest];
NSError *error ;
NSArray *resultArray= [context executeFetchRequest:fetchRequest error:&error];
```

带排序获取数据：

```
NSFetchRequest<EntityName *> *fetchRequest = [EntityName fetchRequest];
NSSortDescriptor *sortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"someKey"
ascending:YES];
fetchRequest.sortDescriptors = @[sortDescriptor];
NSError *error ;
NSArray *resultArray= [context executeFetchRequest:fetchRequest error:&error];
```

添加数据：

```
NSManagedObject *entityNameObj = [NSEntityDescription insertNewObjectForEntityForName:@"EntityName"
inManagedObjectContext:context];
[entityNameObj setValue:@"someValue" forKey:@"someKey"];
```

保存上下文：

```
[((AppDelegate*)[[UIApplication sharedApplication] delegate]) saveContext];
```

Chapter 190: Core Data

Core Data is the model layer of your application in the broadest sense possible. It's the Model in the Model-View-Controller pattern that permeates the iOS SDK.

Core Data isn't the database of your application nor is it an API for persisting data to a database. Core Data is a framework that manages an object graph. It's as simple as that. Core Data can persist that object graph by writing it to disk, but that is not the primary goal of the framework.

Section 190.1: Operations on core data

To Get context:

```
NSManagedObjectContext *context = ((AppDelegate*)[[UIApplication sharedApplication] delegate]).persistentContainer.viewContext;
```

To fetch data:

```
NSFetchRequest<EntityName *> *fetchRequest = [EntityName fetchRequest];
NSError *error ;
NSArray *resultArray= [context executeFetchRequest:fetchRequest error:&error];
```

To fetch data with sorting:

```
NSFetchRequest<EntityName *> *fetchRequest = [EntityName fetchRequest];
NSSortDescriptor *sortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"someKey"
ascending:YES];
fetchRequest.sortDescriptors = @[sortDescriptor];
NSError *error ;
NSArray *resultArray= [context executeFetchRequest:fetchRequest error:&error];
```

To add data:

```
NSManagedObject *entityNameObj = [NSEntityDescription insertNewObjectForEntityForName:@"EntityName"
inManagedObjectContext:context];
[entityNameObj setValue:@"someValue" forKey:@"someKey"];
```

To save context:

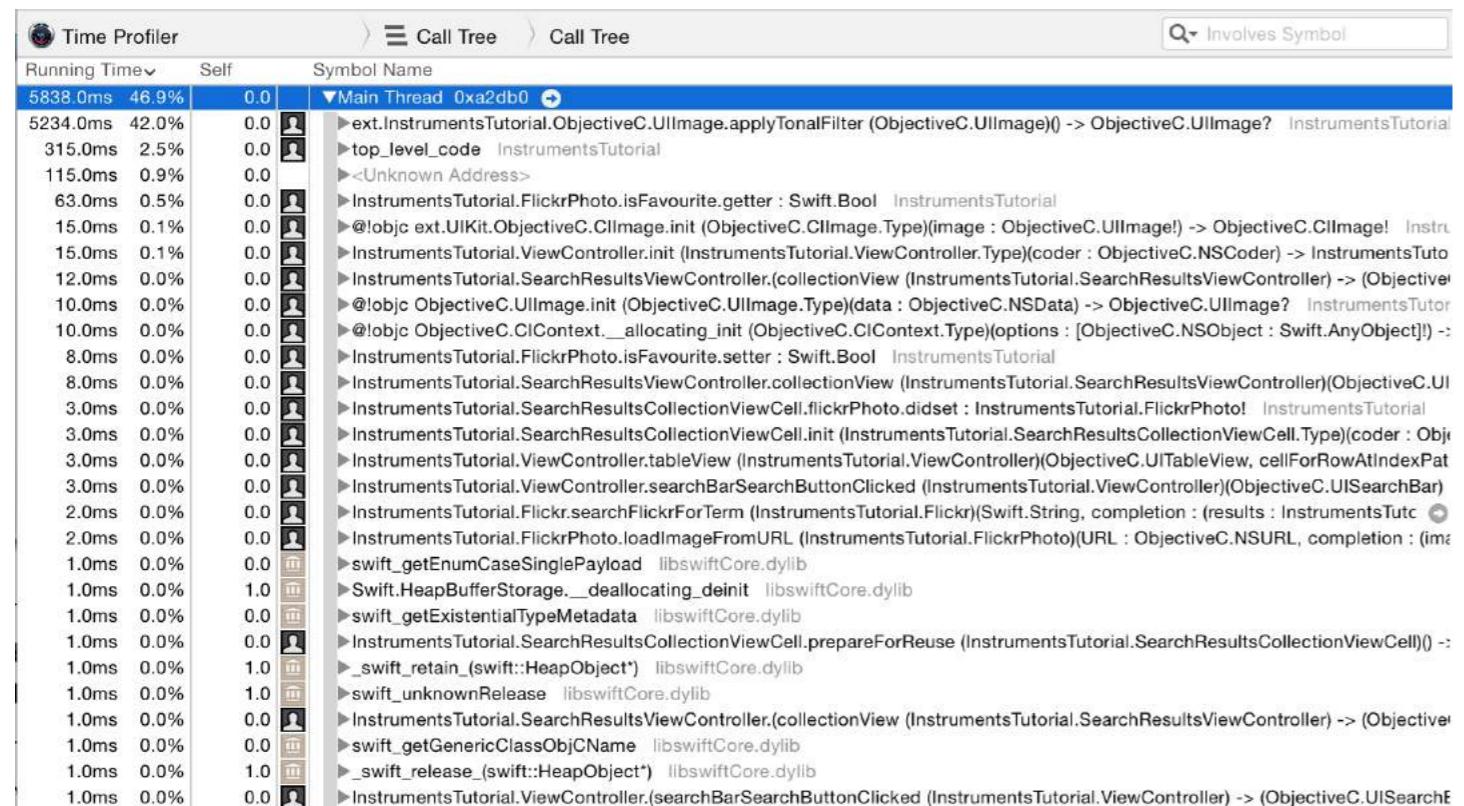
```
[((AppDelegate*)[[UIApplication sharedApplication] delegate]) saveContext];
```

第191章：使用Instruments进行性能分析

Xcode包含一个名为Instruments的性能调优应用程序，您可以使用它通过各种不同的指标来分析您的应用程序性能。它提供了检查CPU使用率、内存使用、内存泄漏、文件/网络活动以及能耗等工具，仅举几例。从Xcode启动应用性能分析非常简单，但有时理解分析结果不容易，这使得一些开发者无法充分利用该工具的全部潜力。

第191.1节：时间分析器

您将首先查看的工具是时间分析器。Instruments会在测量间隔内暂停程序执行，并对每个运行线程进行堆栈跟踪。可以把它看作是在Xcode调试器中按下暂停按钮。以下是时间分析器的预览：



此界面显示调用树。调用树展示了应用程序中各个方法执行所花费的时间。每一行代表程序执行路径中经过的不同方法。通过分析分析器停止在每个方法的次数，可以确定在每个方法中花费的时间。例如，如果

100 采样间隔为1毫秒，且某个特定方法被发现位于堆栈顶部时

10个样本，那么你可以推断出大约10%的总执行时间—10毫秒—花费在该方法中。这是一个相当粗略的估算，但它有效！

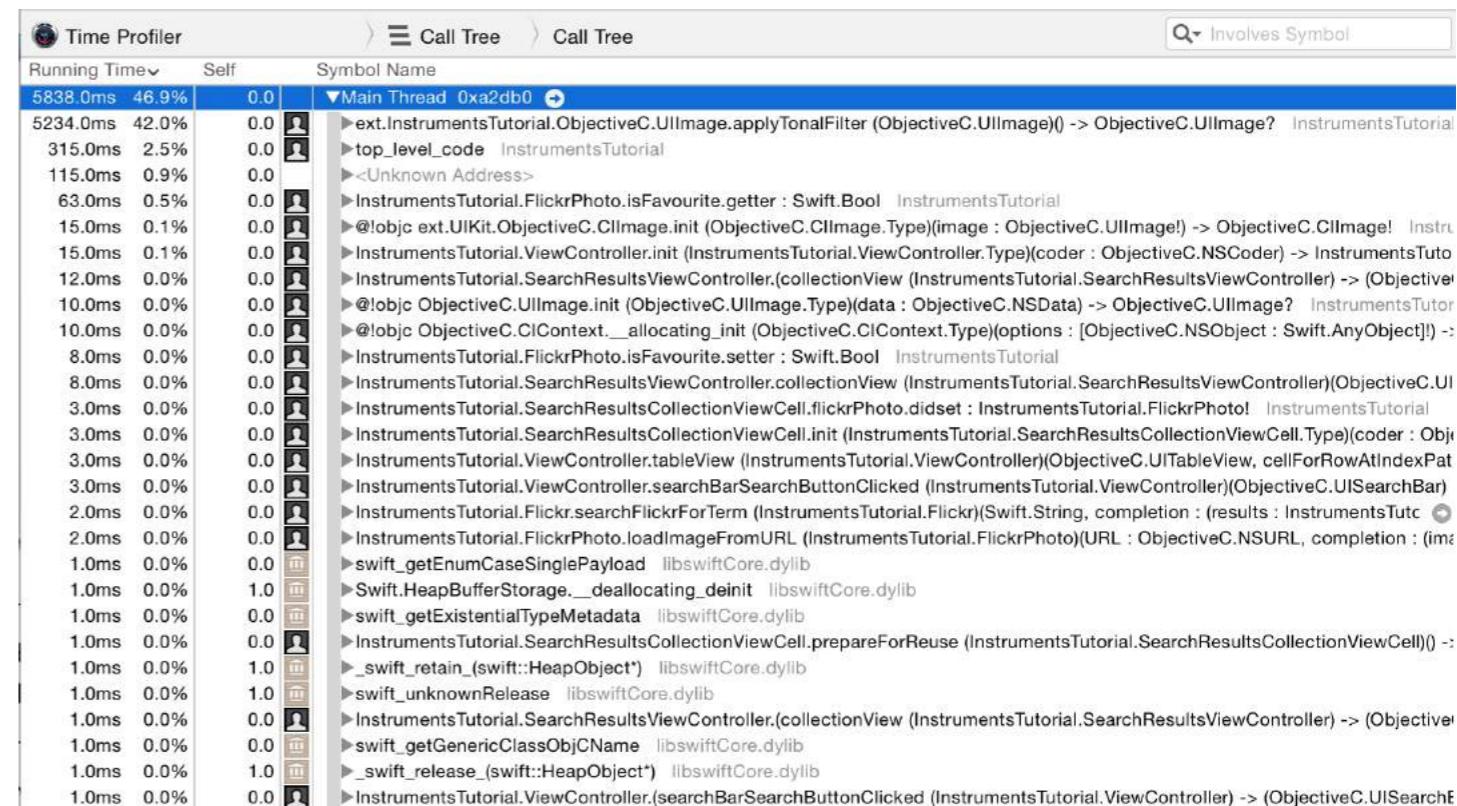
从Xcode的菜单栏中，选择Product\Profile，或按下⌘I。这将构建应用并启动Instruments。你将看到一个如下所示的选择窗口：

Chapter 191: Profile with Instruments

Xcode includes a performance tuning application named Instruments that you can use to profile your application using all sorts of different metrics. They have tools to inspect CPU usage, memory usage, leaks, file/network activity, and energy usage, just to name a few. It's really easy to start profiling your app from Xcode, but it's sometimes not as easy to understand what you see when it's profiling, which deters some developers from being able to use this tool to its fullest potential.

Section 191.1: Time Profiler

The first instrument you'll look at is the Time Profiler. At measured intervals, Instruments will halt the execution of the program and take a stack trace on each running thread. Think of it as pressing the pause button in Xcode's debugger. Here's a sneak preview of the Time Profiler :-



This screen displays the Call Tree. The Call Tree shows the amount of time spent executing in various methods within an app. Each row is a different method that the program's execution path has followed. The time spent in each method can be determined from the number of times the profiler is stopped in each method. For instance, if **100** samples are done at **1 millisecond intervals**, and a particular method is found to be at the top of the stack in 10 samples, then you can deduce that approximately **10%** of the total execution time — **10 milliseconds** — was spent in that method. It's a fairly crude approximation, but it works!

From Xcode's menu bar, select Product\Profile, or press ⌘I. This will build the app and launch Instruments. You will be greeted with a selection window that looks like this:



这些都是Instruments自带的不同模板。

选择Time Profiler工具并点击选择。这将打开一个新的Instruments文档。点击左上角的红色录制按钮开始录制并启动应用。你可能会被要求输入密码以授权Instruments分析其他进程—不用担心，这里提供密码是安全的！在Instruments窗口中，你可以看到时间在计数，屏幕中央图表上方有一个小箭头从左向右移动。这表示应用正在运行。

现在，开始使用应用。搜索一些图片，并深入查看一个或多个搜索结果。你可能已经注意到，进入搜索结果非常缓慢，滚动搜索结果列表也非常令人烦躁—这是一个非常笨拙的应用！

好消息是，你很幸运，因为你即将开始修复它！不过，首先你将快速了解一下你在Instruments中看到的内容。首先，确保工具栏右侧的视图选择器中两个选项都被选中，如下所示：



这将确保所有面板都打开。现在请查看下面的截图及其下方对各部分的说明：



These are all different templates that come with Instruments.

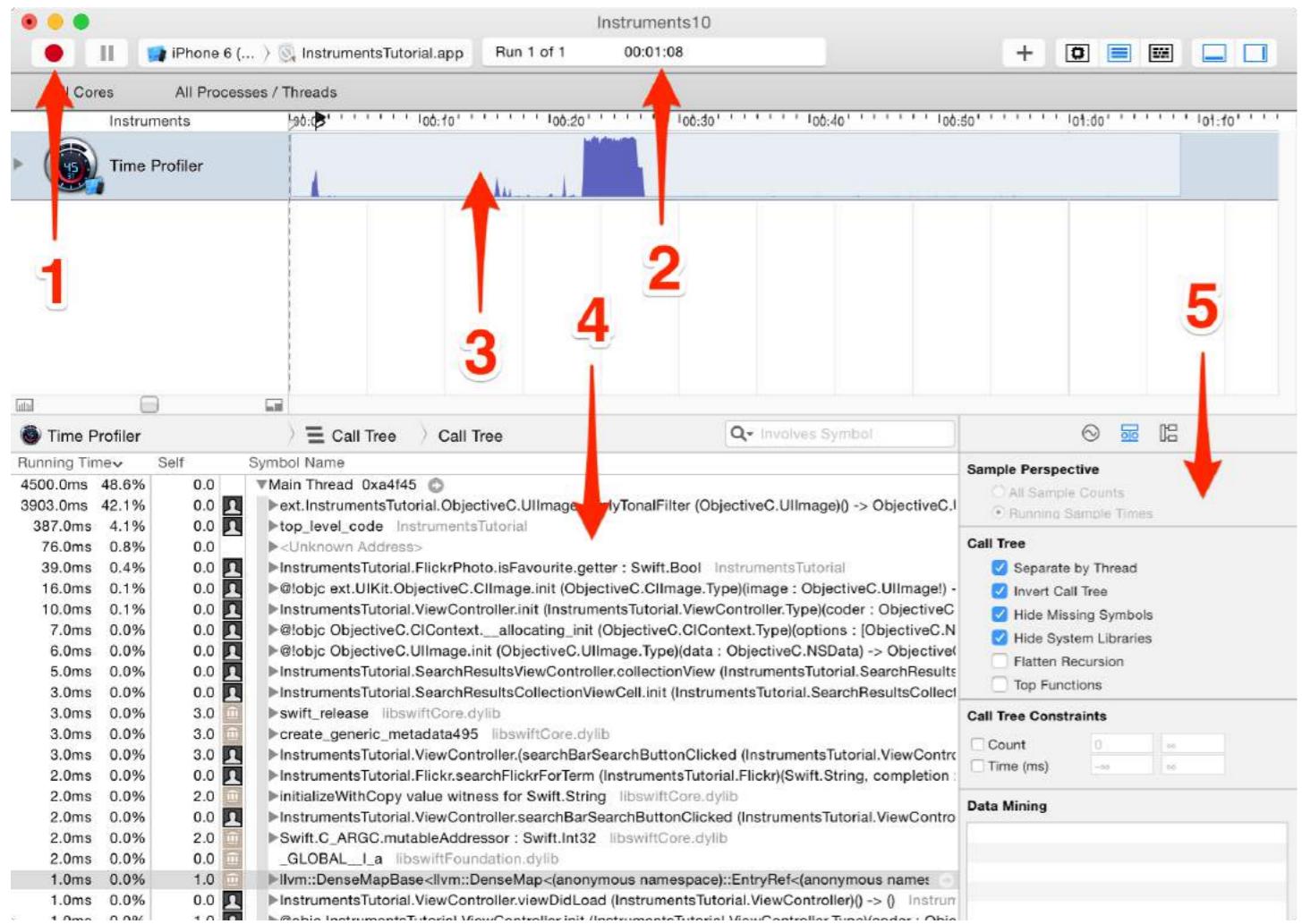
Select the Time Profiler instrument and click Choose. This will open up a new Instruments document. Click the red **record button** in the top left to start recording and launch the app. You may be asked for your password to authorize **Instruments** to analyze other processes — fear not, it's safe to provide here! In the **Instruments window**, you can see the time counting up, and a little arrow moving from left to right above the **graph** in the center of the screen. This indicates that the app is running.

Now, start using the app. Search for some images, and drill down into one or more of the search results. You have probably noticed that going into a search result is tediously slow, and scrolling through a list of search results is also incredibly annoying – it's a terribly clunky app!

Well, you're in luck, for you're about to embark on fixing it! However, you're first going to get a quick run down on what you're looking at in **Instruments**. First, make sure the view selector on the right hand side of the toolbar has both options selected, like so:



That will ensure that all panels are open. Now study the screenshot below and the explanation of each section beneath it:



1. 这些是录制控制。红色的‘录制’按钮将停止和启动当前被分析的应用，当它被点击时（在录制图标和停止图标之间切换）。暂停按钮的功能正如你所期望的那样，暂停应用程序的当前执行。

2. 这是运行计时器。计时器会统计被分析的应用程序运行了多长时间，以及运行了多少次。如果你使用录制控制停止然后重新启动应用程序，那将开始一个新的运行，显示屏将显示“运行 2 的 2”。

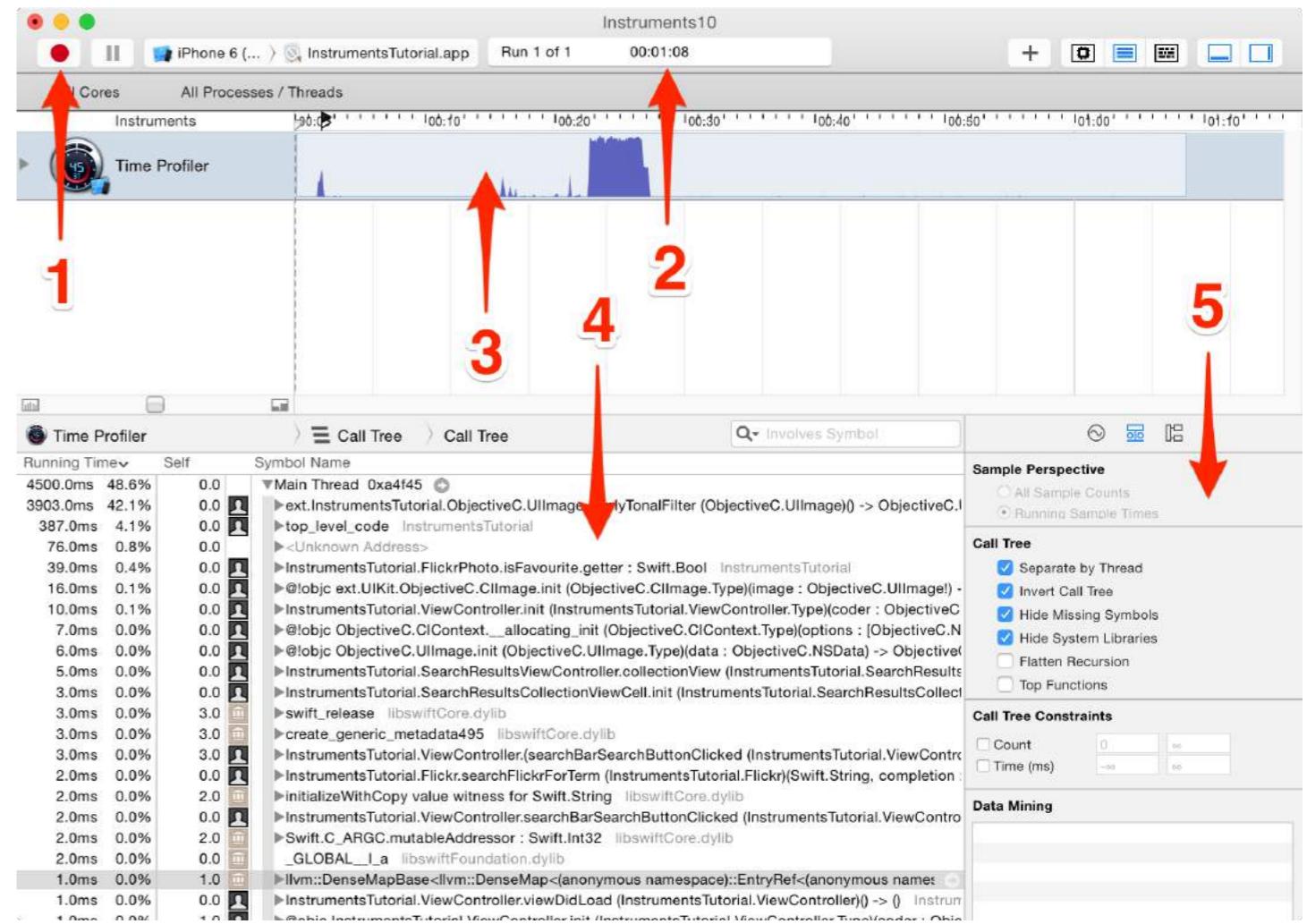
3. 这称为轨迹。在你选择的时间分析器模板中，只有一个工具，因此只有一个轨迹。你将在教程后面了解这里显示的图表的具体内容。

4. 这是详细面板。它显示你正在使用的特定工具的主要信息。在本例中，它显示的是“最热”的方法——也就是使用了最多 CPU 时间的方法。如果你点击顶部写着“调用树”（左边那个）的栏，并选择“样本列表”，你将看到数据的另一种视图。该视图显示每一个样本。点击几个样本，你将在扩展详细信息检查器中看到捕获的堆栈跟踪。

5. 这是检查器面板。有三个检查器：录制设置、显示设置和扩展详细信息。你很快就会学习到这些选项中的一些内容。

深入挖掘

执行图像搜索，并深入查看结果。我个人喜欢搜索“狗”，但你可以选择任何你喜欢的——你可能是那些喜欢猫的人之一！



1. These are the **recording controls**. The red ‘record’ button will stop & start the app currently being profiled when it is clicked (it toggles between a record and stop icon). The pause button does exactly what you’d expect and pauses the current execution of the app.

2. This is the run timer. The timer counts how long the app being profiled has been running, and how many times it has been run. If you stop and then restart the app using the recording controls, that would start a new run and the display would then show Run 2 of 2.

3. This is called a track. In the case of the Time Profiler template you selected, there’s just one instrument so there’s just one track. You’ll learn more about the specifics of the graph shown here later in the tutorial.

4. This is the detail panel. It shows the main information about the particular instrument you’re using. In this case, it’s showing the methods which are “hottest” — that is, the ones that have used up the most CPU time. If you click on the bar at the top which says Call Tree (the left hand one) and select Sample List, then you are presented with a different view of the data. This view is showing every single sample. Click on a few samples, and you’ll see the captured stack trace appear in the Extended Detail inspector.

5. This is the inspectors panel. There are three inspectors: Record Settings, Display Settings, and Extended Detail. You’ll be learning more about some of these options shortly.

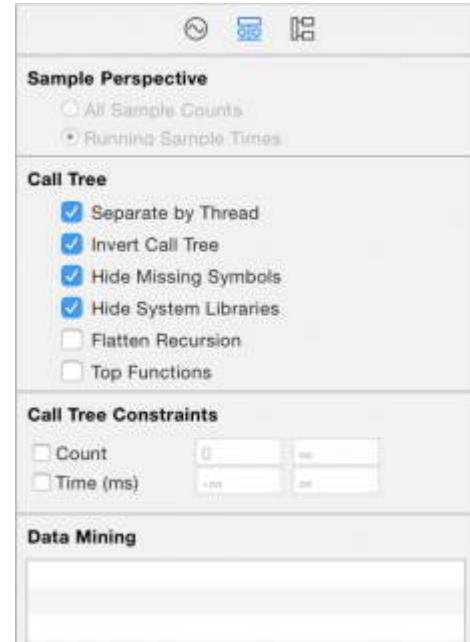
Drilling Deep

Perform an image search, and drill into the results. I personally like searching for “dog”, but choose whatever you wish – you might be one of those cat people!

现在，上下滚动列表几次，这样你就在时间分析器中获得了足够的数据。你应该会注意到屏幕中间的数字在变化，图表也在填充；这表明**CPU周期**正在被使用。

你真的不会期望任何用户界面像这样笨拙，任何表格视图在滚动时都应该像黄油一样顺滑，才能准备好发布！为了帮助定位问题，你需要设置一些选项。

在右侧，选择显示设置检查器（或按⌘+2）。在检查器中，调用树部分，选择按线程分开，反转调用树，隐藏缺失符号和隐藏系统库。界面将显示如下：



以下是每个选项对左侧表格中显示数据的作用：

按线程分开：每个线程应单独考虑。这使你能够了解哪些线程负责最多的**CPU**使用。

反转调用树：启用此选项后，堆栈跟踪将从上到下考虑。这通常是你想要的，因为你想看到**CPU**花费时间最深的方法。

隐藏缺失符号：如果你的应用或系统框架找不到dSYM文件，那么表格中不会显示方法名（符号），而是显示对应于二进制内部地址的十六进制值。如果选择此选项，则只显示完全解析的符号，未解析的十六进制值将被隐藏。这有助于简化呈现的数据。

隐藏系统库：选择此选项时，只显示你自己应用的符号。通常选择此选项很有用，因为你通常只关心CPU在你自己代码中花费的时间—你无法控制系统库使用了多少CPU！

扁平化递归：此选项将递归函数（即调用自身的函数）视为每个堆栈跟踪中的一个条目，而不是多个。

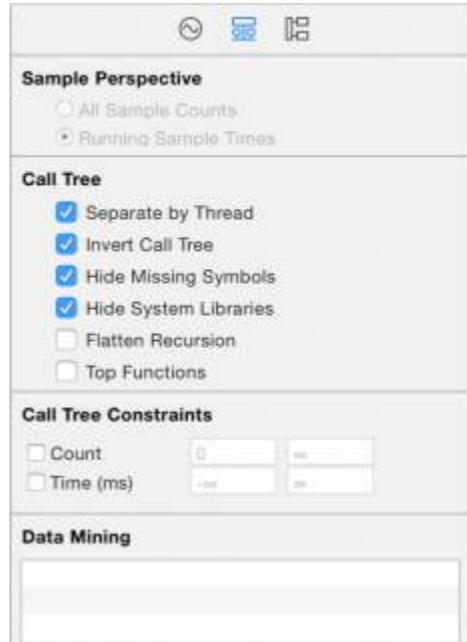
主要功能：启用此功能后，仪器会将函数中花费的总时间视为该函数内直接花费的时间与该函数调用的其他函数中花费时间的总和。

所以如果函数A调用了B，那么A的时间报告为在A中花费的时间加上在B中花费的时间。这非常有用，因为它让你每次深入调用栈时都能选出最大的时间数字，从而聚焦于你最

Now, scroll up and down the list a few times so that you've got a good amount of data in the Time Profiler. You should notice the numbers in the middle of the screen changing and the **graph** filling in; this tells you that **CPU cycles** are being used.

You really wouldn't expect any UI to be as clunky as this no table view is ready to ship until it scrolls like butter! To help pinpoint the problem, you need to set some options.

On the right hand side, select the **Display Settings inspector** (or press ⌘+2). In the **inspector**, under the Call Tree section, select Separate by Thread, Invert Call Tree, Hide Missing Symbols and Hide System Libraries. It will look like this:



Here's what each option is doing to the data displayed in the table to the left:

Separate by Thread: Each thread should be considered separately. This enables you to understand which threads are responsible for the greatest amount of **CPU** use.

Invert Call Tree: With this option, the stack trace is considered from top to bottom. This is usually what you want, as you want to see the deepest methods where the **CPU** is spending its time.

Hide Missing Symbols: If the dSYM file cannot be found for your app or a system framework, then instead of seeing method names (symbols) in the table, you'll just see hex values corresponding to addresses inside the binary. If this option is selected, then only fully resolved symbols are displayed and the unresolved **hex** values are hidden. This helps to declutter the data presented.

Hide System Libraries: When this option is selected, only symbols from your own app are displayed. It's often useful to select this option, since usually you only care about where the **CPU** is spending time in your own code – you can't do much about how much **CPU** the system libraries are using!

Flatten Recursion: This option treats **recursive** functions (ones which call themselves) as one entry in each stack trace, rather than multiple.

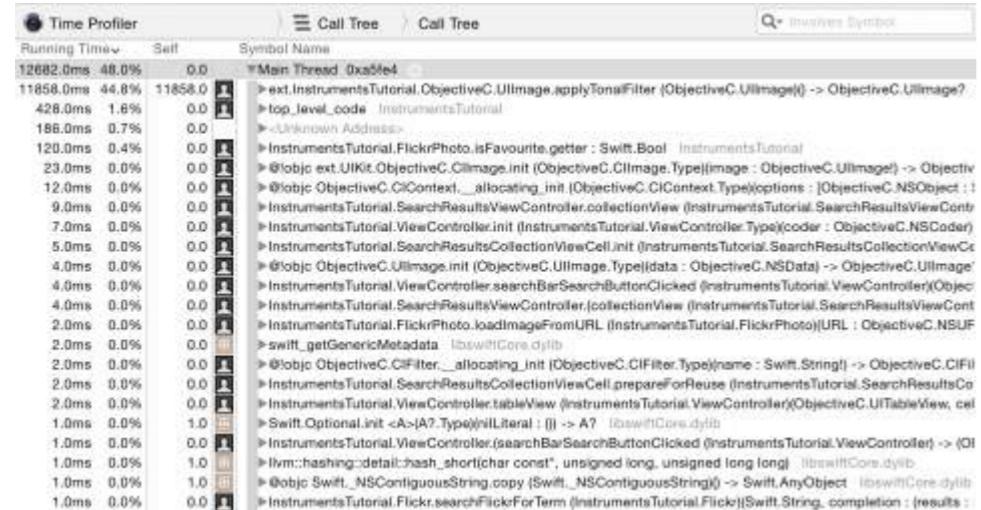
Top Functions: Enabling this makes Instruments consider the total time spent in a function as the sum of the time directly within that function, as well as the time spent in functions called by that function.

So if function A calls B, then A's time is reported as the time spent in A PLUS the time spent in B. This can be really useful, as it lets you pick the largest time figure each time you descend into the call stack, zeroing in on your most

耗时的方法。

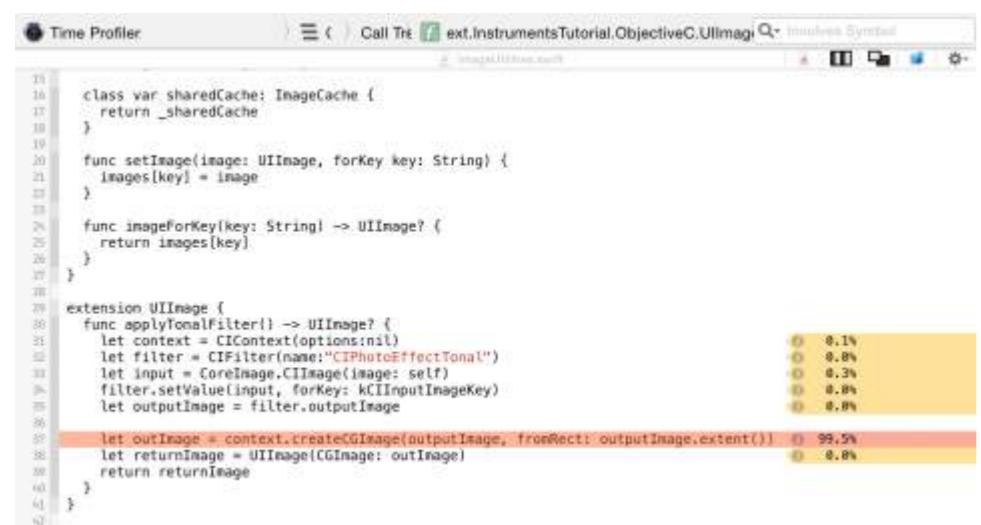
如果你正在运行一个Objective-C应用程序，还有一个“仅显示Obj-C”的选项：如果选中此项，则只显示Objective-C方法，而不显示任何C或C++函数。你的程序中没有这些，但如果你查看一个OpenGL应用程序，例如，它可能会包含一些C++代码。

虽然某些数值可能略有不同，但启用上述选项后，条目的顺序应与下表类似：



嗯，这看起来确实不太好。绝大多数时间都花在了对缩略图应用“调子”滤镜的方法上。这对你来说不应该太意外，因为表格加载和滚动是界面中最笨重的部分，而这正是表格单元格不断被更新的时候。

要了解该方法内部发生了什么，双击表格中的该行。这样会弹出以下视图：



这很有趣，不是吗！`applyTonalFilter()`是添加到`UIImage`的一个扩展方法，几乎100%的时间都花在应用图像滤镜后创建`CGImage`输出上。

这部分其实没什么太多加速空间：创建图像是一个相当密集的过程，耗时就是耗时。我们试着退一步，看看`applyTonalFilter()`是从哪里调用的。点击代码视图顶部面包屑导航中的“调用树”返回到上一个界面：

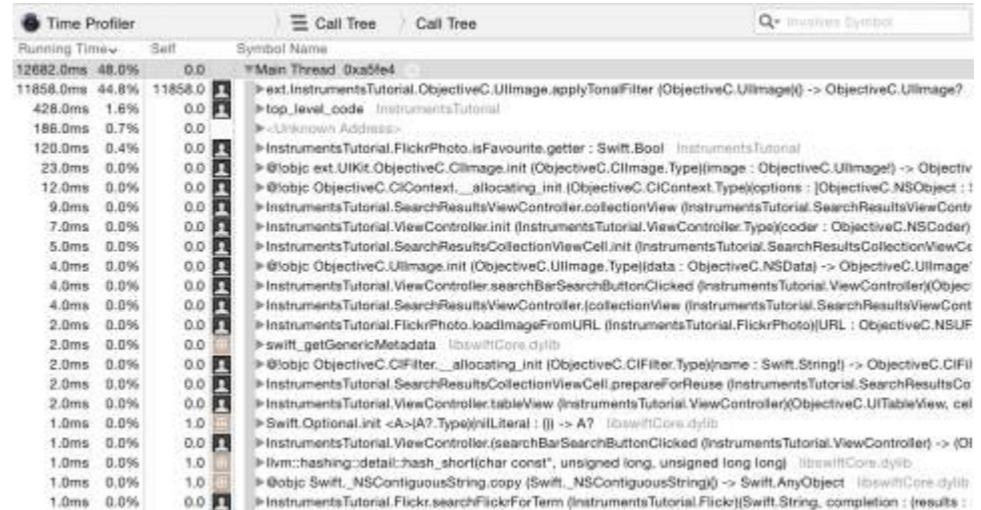


现在点击表格顶部applyTonalFilter行左侧的小箭头。这将展开调用树

time-consuming methods.

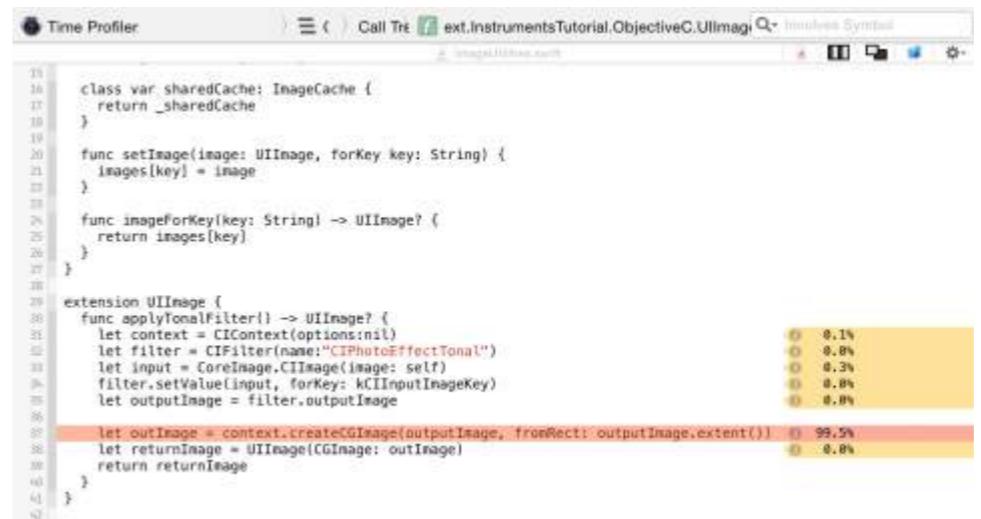
If you're running an Objective-C app, there's also an option of Show **Obj-C Only**: If this is selected, then only objective-C methods are displayed, rather than any C or C++ functions. There are none in your program, but if you were looking at an OpenGL app, it might have some C++, for example.

Although some values may be slightly different, the order of the entries should be similar to the table below once you have enabled the options above:



Well, that certainly doesn't look too good. The vast majority of time is spent in the method that applies the 'tonal' filter to the thumbnail photos. That shouldn't come as too much of a shock to you, as the table loading and scrolling were the clunkiest parts of the UI, and that's when the table cells are constantly being updated.

To find out more about what's going on within that method, double click its row in the table. Doing so will bring up the following view:



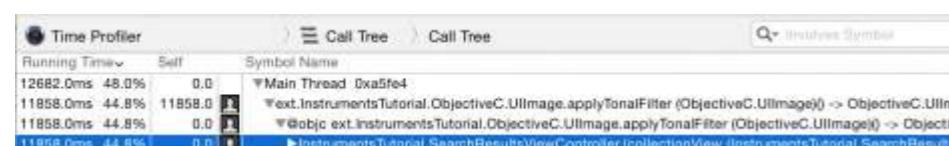
Well that's interesting, isn't it! `applyTonalFilter()` is a method added to `UIImage` in an extension, and almost **100%** of the time spent in it is spent creating the `CGImage` output after applying the image filter.

There's not really much that can be done to speed this up: creating the image is quite an intensive process, and takes as long as it takes. Let's try stepping back and seeing where `applyTonalFilter()` is called from. Click Call Tree in the breadcrumb trail at the top of the code view to get back to the previous screen:



Now click the small arrow to the left of the `applyTonalFilter` row at the top of the table. This will unfold the Call Tree

显示 `applyTonalFilter` 的调用者。你可能还需要展开下一行；在分析 Swift 时，调用树中有时会出现重复的行，前缀为 `@objc`。你感兴趣的是第一个以你的应用'目标名称 (InstrumentsTutorial) 为前缀的行：

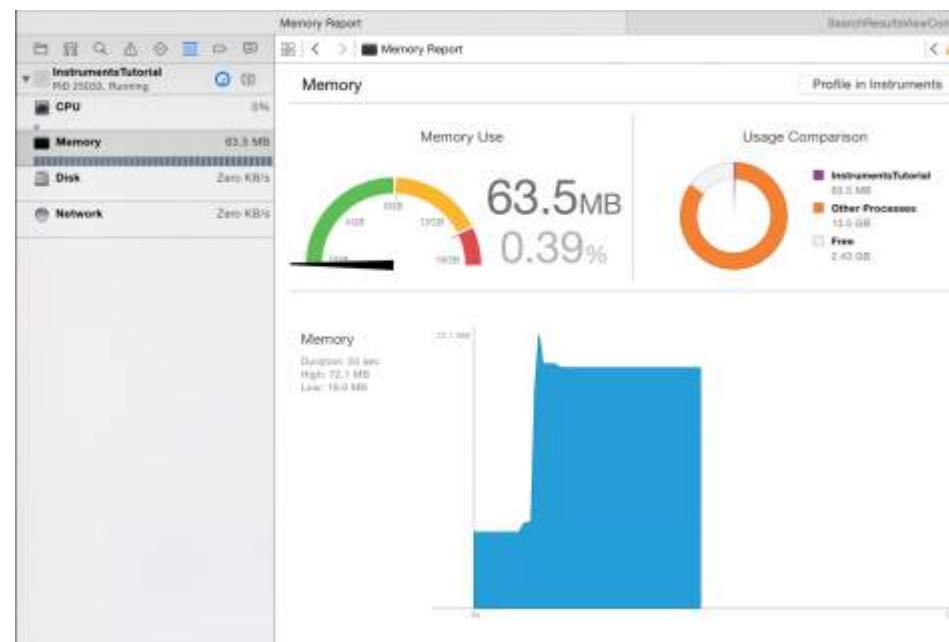


在这种情况下，这一行指的是结果集合视图的 `cellForItemAtIndexPath`。双击该行以查看项目中相关的代码。

现在你可以看到问题所在。应用色调滤镜的方法执行时间较长，并且它'是直接从 `cellForItemAtIndexPath` 调用的，这会阻塞主线程（因此阻塞整个 UI），每次请求滤镜图像时都会如此。

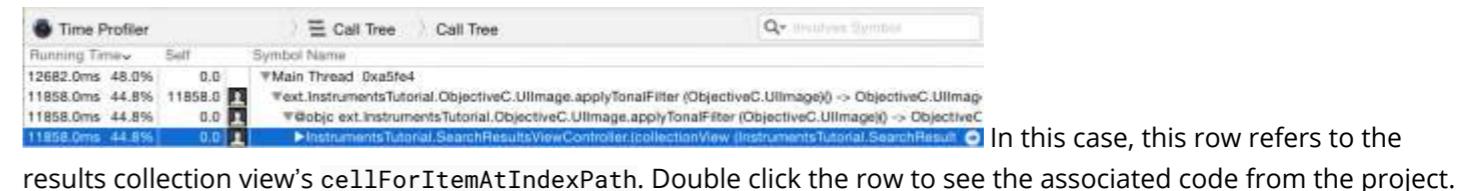
分配

这里有关于所有被创建的对象及其所占用内存的详细信息；它还显示了每个对象的引用计数。要重新开始一个新的 Instruments 配置，请退出 Instruments 应用。这次，构建并运行应用，然后在导航区域打开调试导航器。接着点击内存，在主窗口显示内存使用图表：



这些图表有助于快速了解你的应用性能。但你需要更强大的功能。点击 Instruments 中的 Profile 按钮，然后选择 Transfer 将此会话导入 Instruments。Allocations 仪器将自动启动。

to show the caller of `applyTonalFilter`. You may need to unfold the next row too; when profiling Swift, there will sometimes be duplicate rows in the Call Tree, prefixed with `@objc`. You're interested in the first row that's prefixed with your app's target name (`InstrumentsTutorial`):

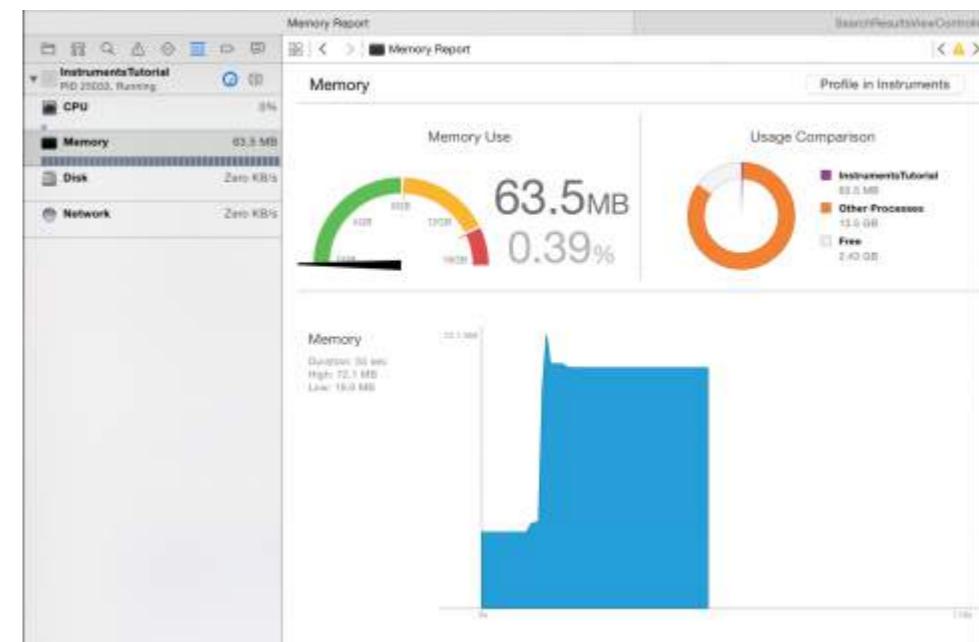


In this case, this row refers to the results collection view's `cellForItemAtIndexPath`. Double click the row to see the associated code from the project.

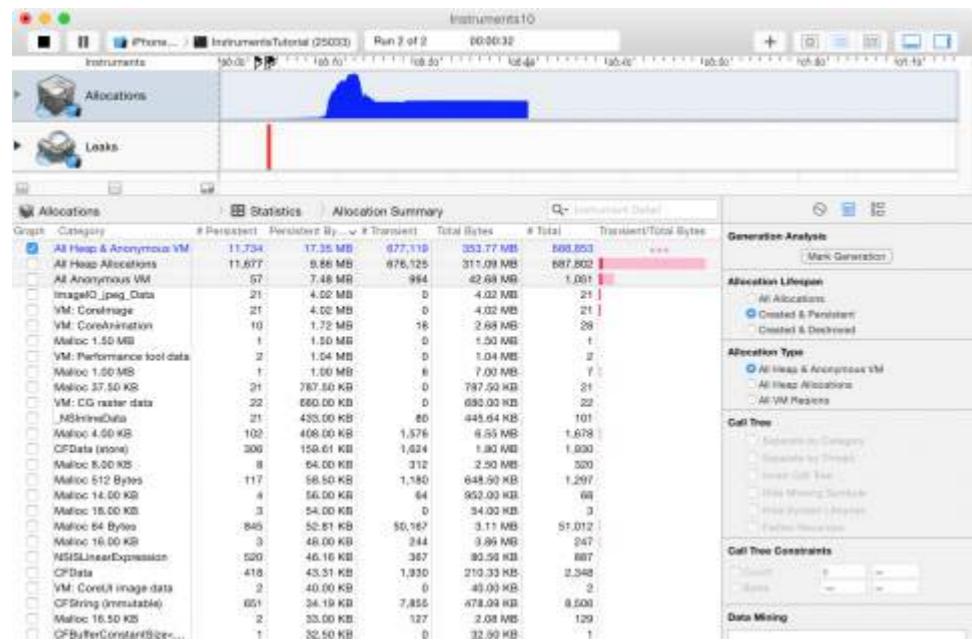
Now you can see what the problem is. The method to apply the tonal filter takes a long time to execute, and it's called directly from `cellForItemAtIndexPath`, which will block the main thread (and therefore the entire UI) each time it's ask for a filtered image.

Allocations

There are detailed information about all the **objects** that are being created and the memory that backs them; it also shows you retain counts of each object. To start afresh with a new instruments profile, quit the Instruments app. This time, build and run the app, and open the Debug Navigator in the Navigators area. Then click on **Memory** to display graphs of memory usage in the main window:



These graphs are useful for to get a quick idea about how your app is performing. But you're going to need a bit more power. Click the Profile in Instruments button and then Transfer to bring this session into **Instruments**. The **Allocations instrument** will start up automatically.



这次你会注意到两个轨迹。一个叫做分配（Allocations），另一个叫做泄漏（Leaks）。分配轨迹将在后面详细讨论；泄漏轨迹通常在Objective-C中更有用，本教程不会涉及。所以你接下来要追踪的是什么错误？项目中隐藏着一些你可能不知道的东西。你可能听说过内存泄漏，但你可能不知道实际上有两种泄漏：

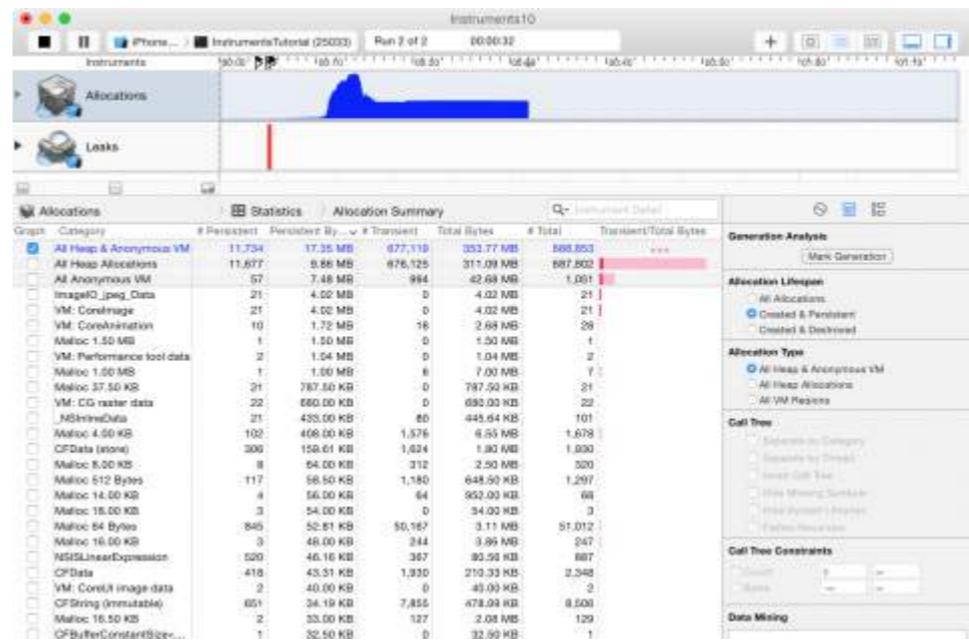
真正的内存泄漏是指一个对象不再被任何东西引用，但仍然被分配着——这意味着内存永远无法被重用。即使有Swift和自动引用计数（ARC）帮助管理内存，最常见的内存泄漏类型是保留环（retain cycle）或强引用环（strong reference cycle）。这是指两个对象相互持有强引用，导致彼此无法被释放。这意味着它们的内存永远不会被释放！

无限制的内存增长是指内存持续被分配，但从未有机会被释放。如果这种情况持续下去，系统的内存最终会被占满，你将面临严重的内存问题。在iOS中，这意味着应用程序会被系统强制终止。

在应用程序上运行分配（Allocations）工具时，进行五次不同的搜索，但暂时不要深入查看结果。确保搜索有一些结果！然后等待几秒钟，让应用程序稳定下来。

你应该注意到分配轨迹中的图表一直在上升。这告诉你内存正在被分配。正是这个特性将引导你发现无限制的内存增长。

你将执行的是一次代分析（generation analysis）。为此，按下名为“标记代”（Mark Generation）的按钮。你可以在显示设置检查器的顶部找到该按钮：



This time you'll notice two tracks. One is called Allocations, and one is called Leaks. The Allocations track will be discussed in detail later on; the Leaks track is generally more useful in Objective-C, and won't be covered in this tutorial. So what bug are you going to track down next? There's something hidden in the project that you probably don't know is there. You've likely heard about memory leaks. But what you may not know is that there are actually two kinds of leaks:

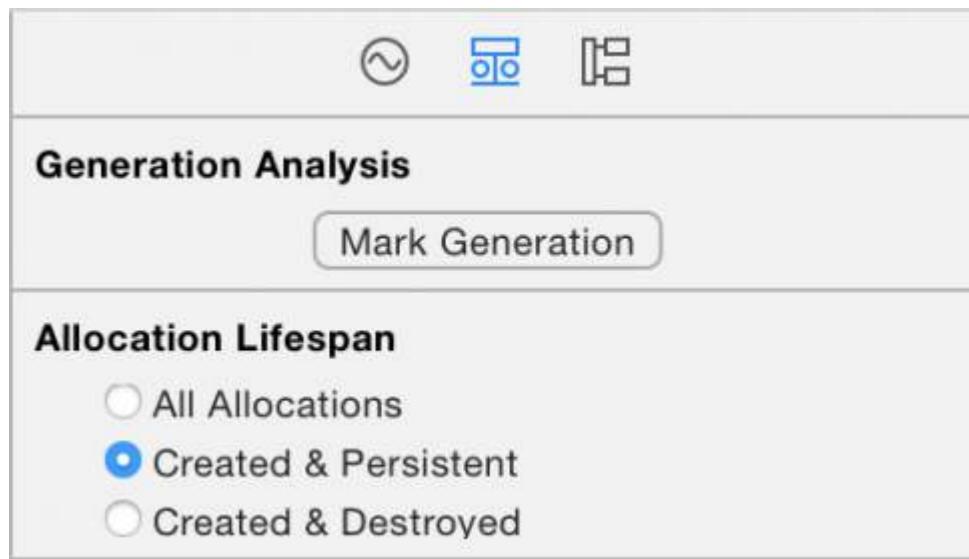
True memory leaks are where an object is no longer referenced by anything but still allocated – that means the memory can never be re-used. Even with Swift and ARC helping manage memory, the most common kind of memory leak is a retain cycle or strong reference cycle. This is when two objects hold strong references to one another, so that each object keeps the other one from being deallocated. This means that their memory is never released!

Unbounded memory growth is where memory continues to be allocated and is never given a chance to be deallocated. If this continues forever, then at some point the system's memory will be filled and you'll have a big memory problem on your hands. In iOS this means that the app will be killed by the system.

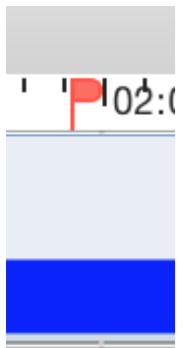
With the Allocations **instrument** running on the app, make five different searches in the app but do not drill down into the results yet. Make sure the searches have some results! Now let the app settle a bit by waiting a few seconds.

You should have noticed that the **graph** in the Allocations track has been rising. This is telling you that memory is being allocated. It's this feature that will guide you to finding unbounded memory growth.

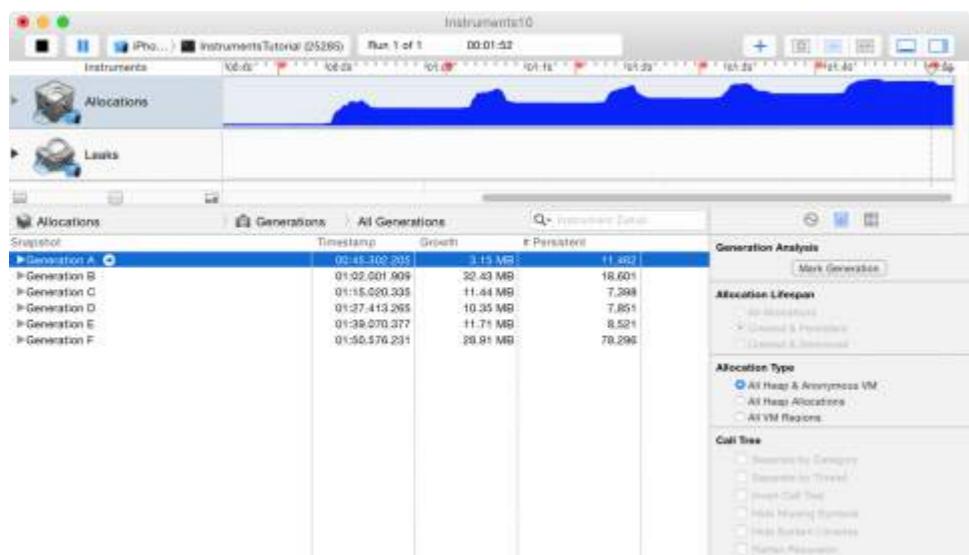
What you're going to perform is a generation analysis. To do this, press the button called Mark Generation. You'll find the button at the top of the Display Settings inspector:



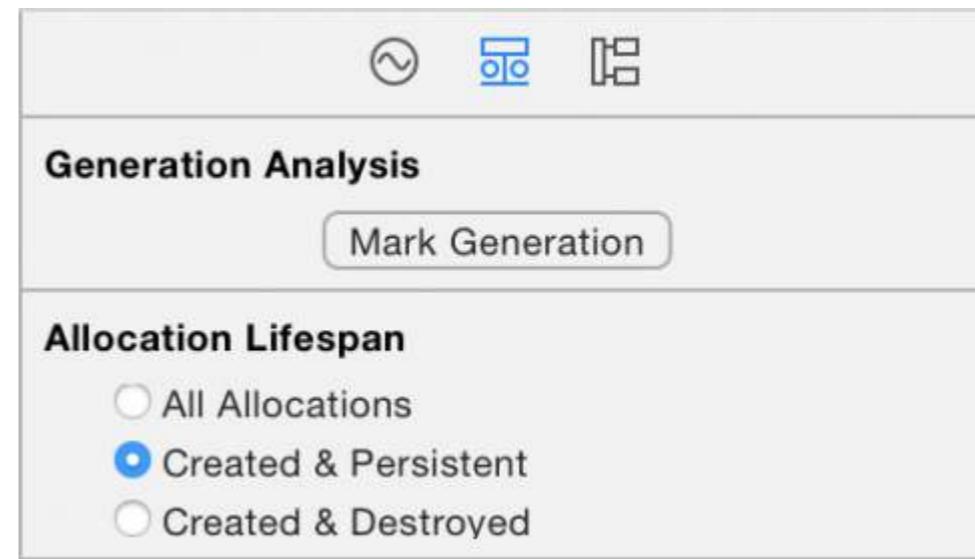
按下它后，你会看到轨迹中出现一个红旗，如下所示：



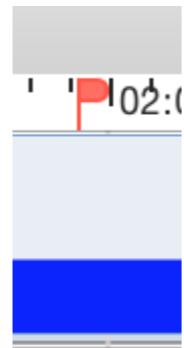
代分析的目的是多次执行某个操作，观察内存是否以无限制的方式增长。深入某个搜索，等待几秒钟让图片加载，然后返回主页面。然后再次标记代。对不同的搜索重复此操作。深入几次搜索后，Instruments会显示如下：



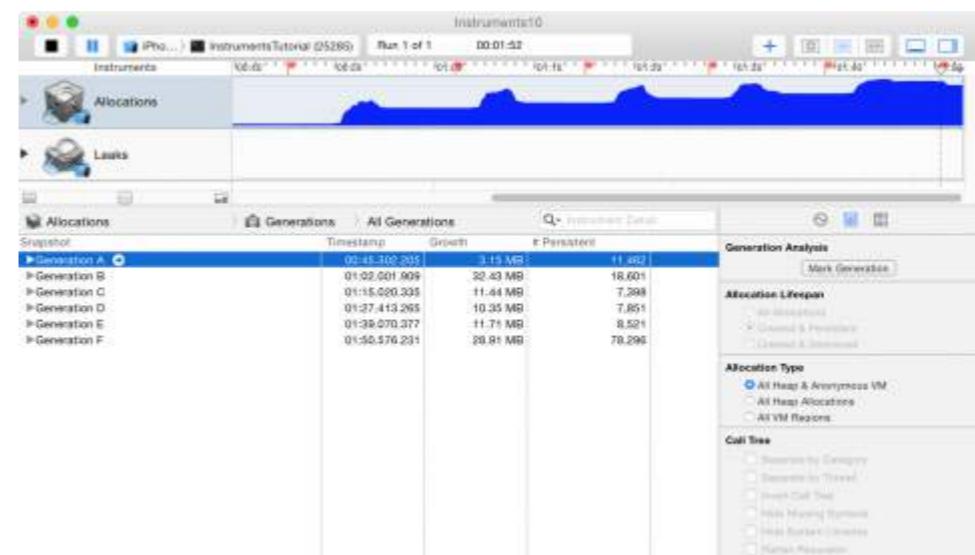
此时，你应该开始怀疑了。注意蓝色图表随着每次深入搜索而上升。这显然是不好的。但等等，内存警告 (memory warnings) 呢？你知道那些，对吧？
内存警告是iOS告诉应用程序内存紧张，需要清理一些内存的方式。



Press it and you will see a red flag appear in the track, like so:



The purpose of generation analysis is to perform an action multiple times, and see if memory is growing in an unbounded fashion. Drill into a search, wait a few seconds for the images to load, and then go back to the main page. Then mark the generation again. Do this repeatedly for different searches. After a drilling into a few searches, Instruments will look like this:



At this point, you should be getting suspicious. Notice how the blue graph is going up with each search that you drill into. Well, that certainly isn't good. But wait, what about memory warnings? You know about those, right? Memory warnings are iOS's way of telling an app that things are getting tight in the memory department, and you need to clear out some memory.

这种增长可能不仅仅是由于你的应用程序；也可能是UIKit深层次的某些部分占用了内存。在指责系统框架或你的应用程序之前，先给它们一个机会来清理内存。

通过在Instruments菜单栏选择“Instrument\Simulate Memory Warning”，或在模拟器菜单栏选择“Hardware\Simulate Memory Warning”来模拟内存警告。你会注意到内存使用量会稍微下降，或者可能根本没有变化，当然不会回到理想状态。所以某处仍然存在无界的内存增长。

在每次深入搜索迭代后标记一代的原因是，你可以看到每代之间分配了多少内存。查看详细面板，你会看到一堆代。

It's possible that this growth is not just due to your app; it could be something in the depths of UIKit that's holding onto memory. Give the system frameworks and your app a chance to clear their **memory** first before pointing a finger at either one.

Simulate a memory warning by selecting Instrument\Simulate Memory Warning in Instruments' menu bar, or Hardware\Simulate Memory Warning from the simulator's menu bar. You'll notice that memory usage dips a little, or perhaps not at all. Certainly not back to where it should be. So there's still **unbounded memory growth** happening somewhere.

The reason for marking a generation after each iteration of drilling into a search is that you can see what **memory** has been allocated between each generation. Take a look in the detail panel and you'll see a bunch of generations.

第192章：应用评分/评论请求

从iOS 10.3开始，不再需要跳转到苹果商店进行评分/评论。苹果在StoreKit框架中引入了SKStoreReviewController类，开发者只需调用SKStoreReviewController类的requestReview()类方法，系统会为你处理整个过程。

此外，你仍然可以在应用的设置或配置界面中包含一个持久链接，深度链接到你的App Store产品页面。以自动打开

第192.1节：评分/评论iOS应用程序

只需在您希望用户对您的应用进行评分/评论的位置输入以下一行代码。

```
SKStoreReviewController.requestReview()
```

Chapter 192: Application rating/review request

Now from iOS 10.3 , there is no need to navigate application to apple store for rating/review. Apple has introduced SKStoreReviewController class in storekit framework. In which developer just need to call requestReview() class method of SKStoreReviewController class and the system handles the entire process for you.

In addition, you can continue to include a persistent link in the settings or configuration screens of your app that deep-links to your App Store product page. To automatically ope

Section 192.1: Rate/Review iOS Application

Just type below one line code from where you want user to rate/review your application.

```
SKStoreReviewController.requestReview()
```

第193章：MyLayout

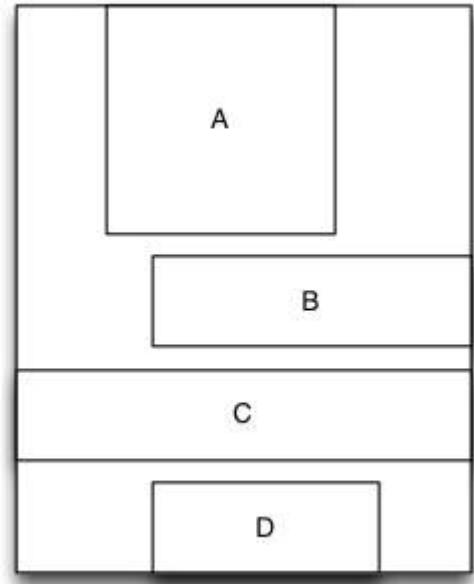
MyLayout是一个简单易用的iOS视图布局Objective-C框架。MyLayout提供了一些简单的功能来构建各种复杂界面。它集成了包括iOS的Autolayout和SizeClass、Android的五种布局类、HTML/CSS的float、flex-box和bootstrap等功能。
你可以访问：

Objective-C: <https://github.com/youngsoft/MyLinearLayout> Swift: <https://github.com/youngsoft/TangramKit>

第193.1节：使用MyLayout的简单示例

- 有一个容器视图S，宽度为100，高度根据所有子视图高度自适应。里面有四个子视图A、B、C、D，从上到下排列。
- 子视图A的左边距是S宽度的20%，右边距是S宽度的30%，高度等于A的宽度。
- 子视图B的左边距是40，宽度填充S的剩余宽度，高度为40。子视图C的宽度填充S，高度为
- 子视图D的右边距是20，宽度是S宽度的50%，高度为40

如下面图所示：



```
MyLinearLayout *S = [MyLinearLayout linearLayoutWithOrientation:MyLayoutViewOrientation_Vert];
S.subviewSpace = 10;
S.widthSize.equalTo(@100);

UIView *A = UIView.new;
A.leftPos.equalTo(@0.2);
A.rightPos.equalTo(@0.3);
A.heightSize.equalTo(A.widthSize);
[S addSubview:A];

UIView *B = UIView.new;
B.leftPos.equalTo(@40);
B.widthSize.equals(@60);
B.heightSize.equalTo(@40);
[S addSubview:B];

UIView *C = UIView.new;
```

Chapter 193: MyLayout

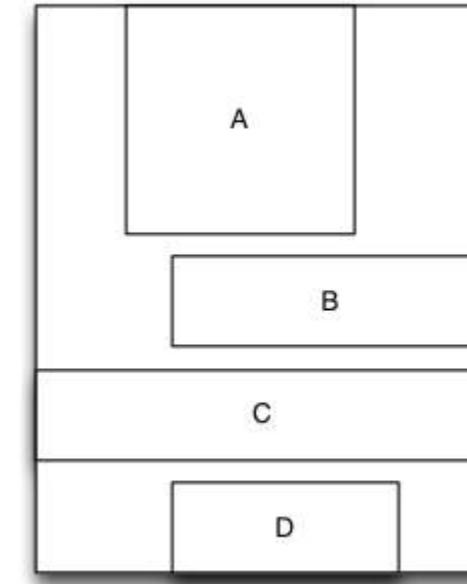
MyLayout is a simple and easy objective-c framework for iOS view layout. MyLayout provides some simple functions to build a variety of complex interface. It integrates the functions including: Autolayout and SizeClass of iOS, five layout classes of Android, float and flex-box and bootstrap of HTML/CSS. you can visit from:

Objective-C: <https://github.com/youngsoft/MyLinearLayout> Swift: <https://github.com/youngsoft/TangramKit>

Section 193.1: A Simple Demo to use MyLayout

- There is a container view S which width is 100 and height is wrap to all subviews height. there are four subviews A,B,C,D arranged from top to bottom.
- Subview A's left margin is 20% width of S, right margin is 30% width of S, height is equal to width of A.
- Subview B's left margin is 40, width is filled in to residual width of S,height is 40. Subview C's width is filled in to S, height is
- Subview D's right margin is 20, width is 50% width of S, height is 40

like below figure:



```
MyLinearLayout *S = [MyLinearLayout linearLayoutWithOrientation:MyLayoutViewOrientation_Vert];
S.subviewSpace = 10;
S.widthSize.equalTo(@100);

UIView *A = UIView.new;
A.leftPos.equalTo(@0.2);
A.rightPos.equalTo(@0.3);
A.heightSize.equalTo(A.widthSize);
[S addSubview:A];

UIView *B = UIView.new;
B.leftPos.equalTo(@40);
B.widthSize.equalTo(@60);
B.heightSize.equalTo(@40);
[S addSubview:B];

UIView *C = UIView.new;
```

```
C.leftPos.equalTo(@0);
C.rightPos.equalTo(@0);
C.heightSize.equalTo(@40);
[S addSubview:C];
```

```
UIView *D = UIView.new;
D.rightPos.equalTo(@20);
D.widthSize.equalTo(S.widthSize).multiply(0.5);
D.heightSize.equalTo(@40);
[S addSubview:D];
```

```
C.leftPos.equalTo(@0);
C.rightPos.equalTo(@0);
C.heightSize.equalTo(@40);
[S addSubview:C];
```

```
UIView *D = UIView.new;
D.rightPos.equalTo(@20);
D.widthSize.equalTo(S.widthSize).multiply(0.5);
D.heightSize.equalTo(@40);
[S addSubview:D];
```

第194章：模拟器构建

在哪里可以找到模拟器构建？

前往 ~/Library/Developer/CoreSimulator/Devices/

你会发现带有字母数字名称的目录

然后点击其中一个目录，进行以下选择

Data/Containers/Bundle/Application/

你会再次看到带有字母数字名称的目录，点击后你会找到

模拟器构建文件

注意：

在模拟器上安装iOS设备构建是行不通的。

iPhone模拟器使用i386架构，iPad模拟器构建使用x8架构

第194.1节：在模拟器上手动安装构建

```
xcrun simctl install booted *.app
```

Chapter 194: Simulator Builds

Where to find simulator build ?

Go to ~/Library/Developer/CoreSimulator/Devices/

You will find directories with alphanumeric names

then click on the one of the directories and make following selection

Data/Containers/Bundle/Application/

Again you will find directories with alphanumeric names if you click on that you will find

Simulator build over there

Note:

Installing iOS device build on simulator will not work out.

iPhone simulator uses i386 architecture iPad simulator builds uses x8

Section 194.1: Installing the build manually on simulator

```
xcrun simctl install booted *.app
```

第195章：使用GPX文件在iOS上模拟位置

第195.1节：您的.gpx文件：MPS_HQ.gpx

```
<gpx xmlns="http://www.topografix.com/GPX/1/1"
  xmlns:gpxx = "http://www.garmin.com/xmlschemas/GpxExtensions/v3"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.topografix.com/GPX/1/1
    http://www.topografix.com/GPX/1/1/gpx.xsd
    http://www.garmin.com/xmlschemas/GpxExtensions/v3
    http://www8.garmin.com/xmlschemas/GpxExtensions/v3/GpxExtensionsv3.xsd"
  version="1.1"
  creator="gpx-poi.com">
  <wpt lat="38.9072" lon="77.0369">38.9072/-77.0369
  <time>2015-04-16T22:20:29Z</time>
  <name>华盛顿特区</name>
  <extensions>
    <gpxx:WaypointExtension>
      <gpxx:Proximity>10</gpxx:Proximity>
      <gpxx:Address>
        <gpxx:StreetAddress>华盛顿特区</gpxx:StreetAddress>
        <gpxx:City>华盛顿</gpxx:City>
        <gpxx:State>DC</gpxx:State>
        <gpxx:Country>美国</gpxx:Country>
        <gpxx:PostalCode> 20005 </gpxx:PostalCode>
      </gpxx:Address>
    </gpxx:WaypointExtension>
  </extensions>
</gpx>
```

第195.2节：设置此位置：

1. 进入编辑方案。
2. 选择运行 -> 选项。
3. 勾选“允许位置模拟”。
4. 从“默认位置”下拉列表中选择 *.GPX 文件名。

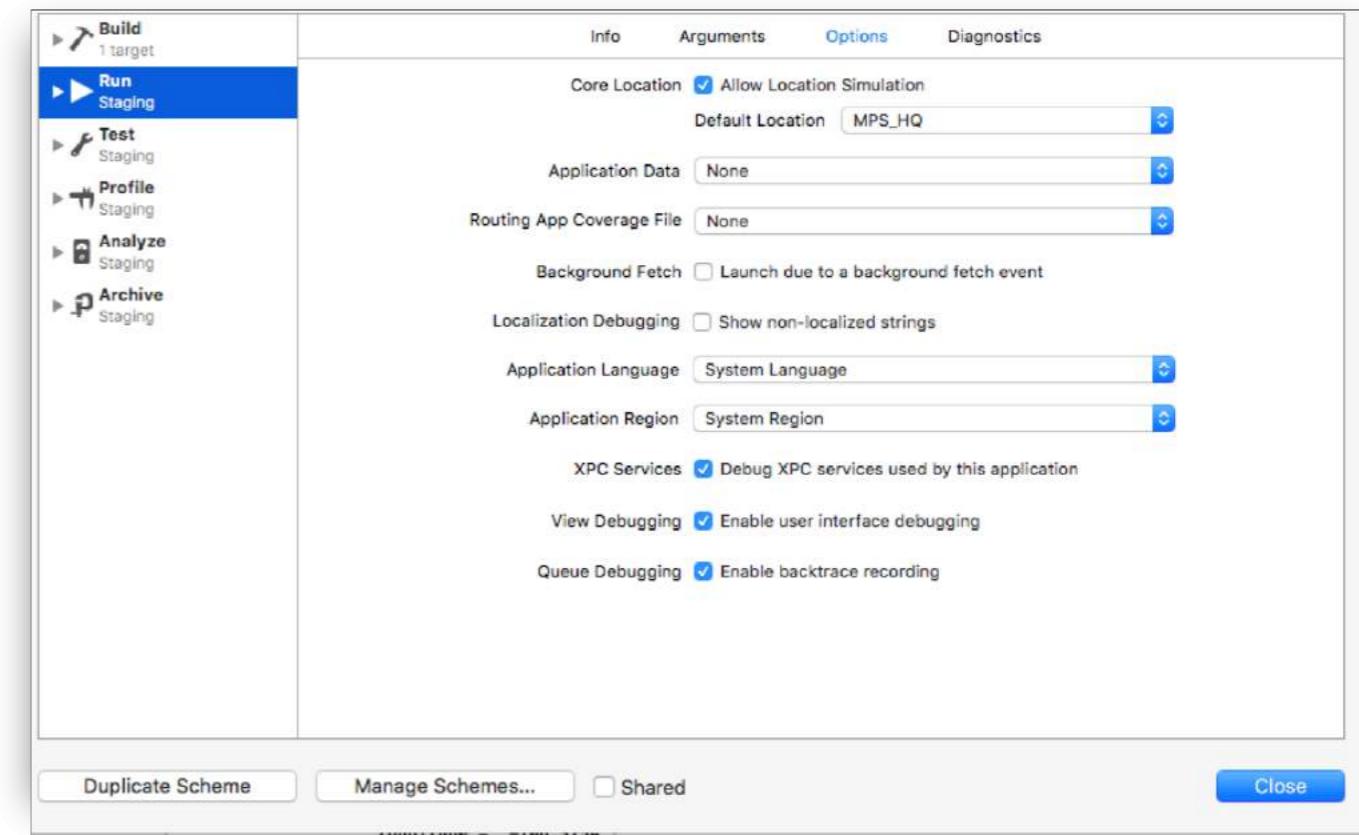
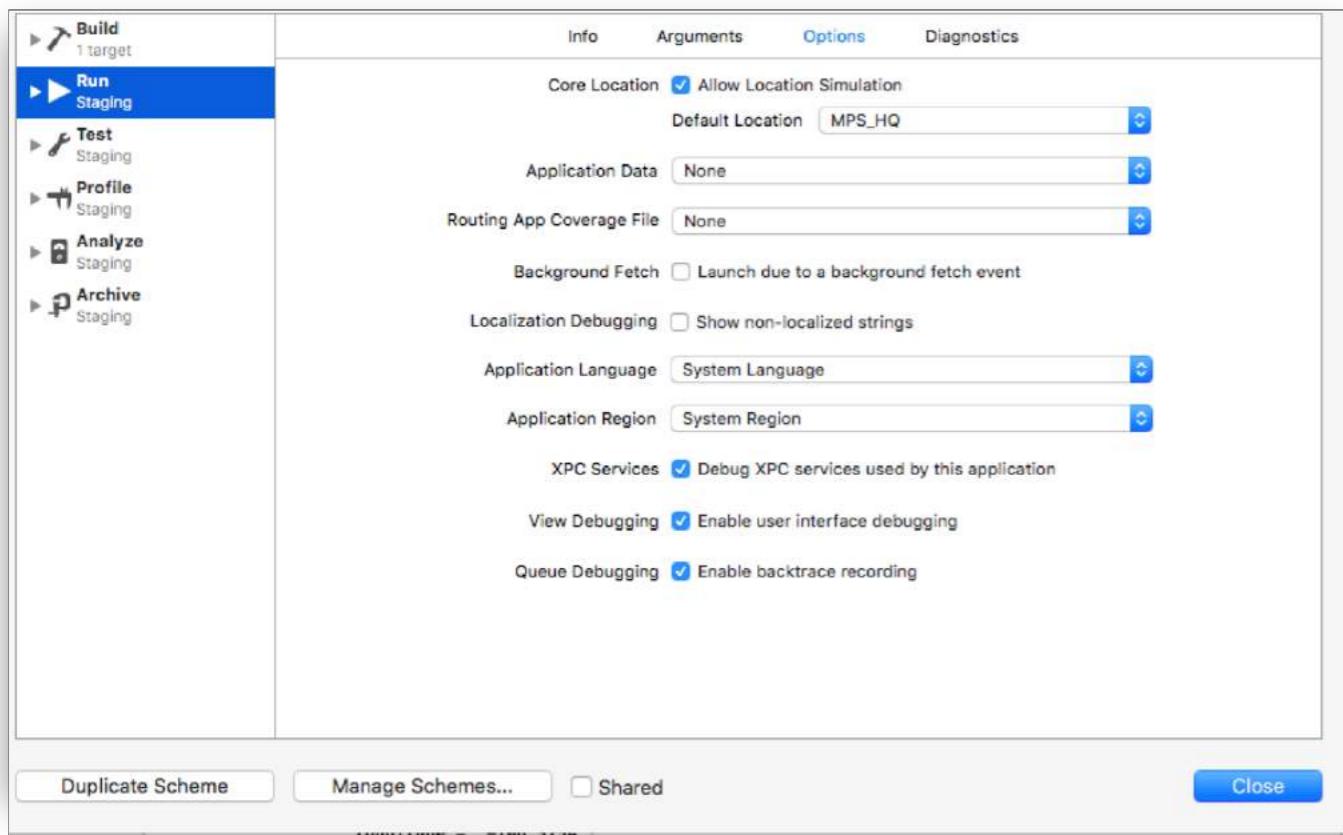
Chapter 195: Simulating Location Using GPX files iOS

Section 195.1: Your .gpx file: MPS_HQ.gpx

```
<gpx xmlns="http://www.topografix.com/GPX/1/1"
  xmlns:gpxx = "http://www.garmin.com/xmlschemas/GpxExtensions/v3"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.topografix.com/GPX/1/1
    http://www.topografix.com/GPX/1/1/gpx.xsd
    http://www.garmin.com/xmlschemas/GpxExtensions/v3
    http://www8.garmin.com/xmlschemas/GpxExtensions/v3/GpxExtensionsv3.xsd"
  version="1.1"
  creator="gpx-poi.com">
  <wpt lat="38.9072" lon="77.0369">38.9072/-77.0369
  <time>2015-04-16T22:20:29Z</time>
  <name>Washington, DC</name>
  <extensions>
    <gpxx:WaypointExtension>
      <gpxx:Proximity>10</gpxx:Proximity>
      <gpxx:Address>
        <gpxx:StreetAddress>Washington DC</gpxx:StreetAddress>
        <gpxx:City>Washington</gpxx:City>
        <gpxx:State>DC</gpxx:State>
        <gpxx:Country>United States</gpxx:Country>
        <gpxx:PostalCode> 20005 </gpxx:PostalCode>
      </gpxx:Address>
    </gpxx:WaypointExtension>
  </extensions>
</gpx>
```

Section 195.2: To set this location:

1. Go to Edit Scheme.
2. Select Run -> Options.
3. Check "Allow Location Simulation".
4. Select the *.GPX File Name from the "Default Location" drop down list.



第196章：SqlCipher 集成

SQLite 已经是 iOS 应用中持久数据存储的流行 API，因此对开发来说优势显而易见。作为程序员，你使用的是一个稳定且文档完善的 API，且该 API 在 Objective-C 中有许多优秀的封装库，如 FMDB 和加密核心数据。所有安全相关的问题都与应用代码清晰分离，由底层框架管理。

第196.1节：代码集成：

使用密码集成打开数据库。

```
- (void)checkAndOpenDB{
    sqlite3 *db;
    NSString *strPassword = @"password";

    if (sqlite3_open_v2([[databaseURL path] UTF8String], &db, SQLITE_OPEN_READWRITE | SQLITE_OPEN_CREATE, NULL) == SQLITE_OK) {
        const char* key = [strPassword UTF8String];
        sqlite3_key(db, key, (int)strlen(key));
        if (sqlite3_exec(db, (const char*) "SELECT count(*) FROM sqlite_master;", NULL, NULL, NULL) == SQLITE_OK) {
            NSLog(@"密码正确，或者已初始化新数据库");
        } else {
            NSLog(@"密码错误！");
        }
        sqlite3_close(db);
    }
}

- (NSURL *)databaseURL
{
    NSArray *URLs = [[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory
inDomains:NSUTFUserDomainMask];
    NSURL *directoryURL = [URLs firstObject];
    NSURL *databaseURL = [directoryURL URLByAppendingPathComponent:@"database.sqlite"];
    return databaseURL;
}
```

Chapter 196: SqlCipher integration

SQLite is already a popular API for persistent data storage in iOS apps so the upside for development is obvious. As a programmer you work with a stable, well-documented API that happens to have many good wrappers available in Objective-C, such as FMDB and Encrypted Core Data. All security concerns are cleanly decoupled from application code and managed by the underlying framework.

Section 196.1: Integration of code:

Integration to open database using password.

```
- (void)checkAndOpenDB{
    sqlite3 *db;
    NSString *strPassword = @"password";

    if (sqlite3_open_v2([[databaseURL path] UTF8String], &db, SQLITE_OPEN_READWRITE | SQLITE_OPEN_CREATE, NULL) == SQLITE_OK) {
        const char* key = [strPassword UTF8String];
        sqlite3_key(db, key, (int)strlen(key));
        if (sqlite3_exec(db, (const char*) "SELECT count(*) FROM sqlite_master;", NULL, NULL, NULL) == SQLITE_OK) {
            NSLog(@"Password is correct, or a new database has been initialized");
        } else {
            NSLog(@"Incorrect password!");
        }
        sqlite3_close(db);
    }
}

- (NSURL *)databaseURL
{
    NSArray *URLs = [[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory
inDomains:NSUTFUserDomainMask];
    NSURL *directoryURL = [URLs firstObject];
    NSURL *databaseURL = [directoryURL URLByAppendingPathComponent:@"database.sqlite"];
    return databaseURL;
}
```

第197章：安全性

iOS中的安全性涉及数据安全、传输安全、代码安全等方面

第197.1节：保护iTunes备份中的数据安全

如果我们希望应用数据在iTunes备份中受到保护，就必须让应用数据不被iTunes备份。

每当iOS设备使用macOS上的iTunes备份时，所有应用存储的数据都会被复制到该备份中，并存储在备份电脑上。

但我们可以使用URLResourceKey.isExcludedFromBackupKey键将应用数据排除在此备份之外。

以下是我们应用的目录结构：



注意：通常敏感数据存储在“应用支持”目录中。

例如，如果我们想排除存储在**Application Support**目录中的所有数据，则可以按如下方式使用上述提到的键：

```
let urls = FileManager.default.urls(for: .applicationSupportDirectory, in: .userDomainMask)
let baseURL = urls[urls.count-1];

        let bundleIdentifier = Bundle.main.object(forInfoDictionaryKey: "CFBundleIdentifier") as!
String
        let pathURL = baseURL.appendingPathComponent(bundleIdentifier)
        let persistentStoreDirectoryPath = pathURL.path
        if !FileManager.default.fileExists(atPath: persistentStoreDirectoryPath) {
            do {
try FileManager.default.createDirectory(atPath: path, withIntermediateDirectories:
true, attributes: nil)
            }catch {
                //handle error
            }
        }
        let dirURL = URL.init(fileURLWithPath: persistentStoreDirectoryPath, isDirectory: true)
        do {
try (dirURL as NSURL).setResourceValue((true), forKey: .isExcludedFromBackupKey)
        } catch {
            //处理错误
        }
```

有很多工具可以查看iTunes备份中的所有备份数据，以确认上述方法是否有效。

Chapter 197: Security

Security in iOS is related to data security, transport security, code security, etc

Section 197.1: Securing Data in iTunes Backups

If we want our app data to be protected against iTunes backups, we have to skip our app data from being backed up in iTunes.

Whenever iOS device backed up using iTunes on macOS, all the data stored by all the apps is copied in that backup and stored on backing computer.

But we can exclude our app data from this backup using URLResourceKey.`isExcludedFromBackupKey` key. Here is the directory structure of our app:



Note: Generally sensitive data is stored in 'Application Support' directory.

e.g. If we want to exclude all our data stored in **Application Support** directory then we can use above mentioned key as follow:

```
let urls = FileManager.default.urls(for: .applicationSupportDirectory, in: .userDomainMask)
let baseURL = urls[urls.count-1];

        let bundleIdentifier = Bundle.main.object(forInfoDictionaryKey: "CFBundleIdentifier") as!
String
        let pathURL = baseURL.appendingPathComponent(bundleIdentifier)
        let persistentStoreDirectoryPath = pathURL.path
        if !FileManager.default.fileExists(atPath: persistentStoreDirectoryPath) {
            do {
try FileManager.default.createDirectory(atPath: path, withIntermediateDirectories:
true, attributes: nil)
            }catch {
                //handle error
            }
        }
        let dirURL = URL.init(fileURLWithPath: persistentStoreDirectoryPath, isDirectory: true)
        do {
try (dirURL as NSURL).setResourceValue((true), forKey: .isExcludedFromBackupKey)
        } catch {
            //handle error
        }
```

There are lots of tools available to see iTunes backups for all the backed up data to confirm whether above approach works or not.

iExplorer 是一个很好的工具，用于浏览iTunes备份。

第197.2节：使用SSL的传输安全

iOS应用需要以一种方式编写，以保障通过网络传输的数据的安全性。

SSL是实现这一点的常用方法。

每当应用尝试调用网络服务以拉取或推送数据到服务器时，应使用基于HTTP的SSL，即HTTPS。

为此，应用必须调用 `https://server.com/part` 这样的网络服务，而不是 `http://server.com/part`。

在这种情况下，应用程序需要信任使用 SSL 证书的服务器 `server.com`。

以下是验证服务器信任的示例-

实现URLSessionDelegate如下：

```
func urlSession(_ session: URLSession, didReceive challenge: URLAuthenticationChallenge, completionHandler: @escaping (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {  
  
    if challenge.protectionSpace.authenticationMethod == NSURLAuthenticationMethodServerTrust {  
        let serverTrust:SecTrust = challenge.protectionSpace.serverTrust!  
  
        func acceptServerTrust() {  
            let credential:URLCredential = URLCredential(trust: serverTrust)  
            challenge.sender?.use(credential, for: challenge)  
            completionHandler(.useCredential, URLCredential(trust:  
challenge.protectionSpace.serverTrust!))  
        }  
  
        let success = SSLTrustManager.shouldTrustServerTrust(serverTrust, forCert:  
"Server_Public_SSL_Cert")  
        if success {  
            acceptServerTrust()  
            return  
        }  
    }  
    else if challenge.protectionSpace.authenticationMethod ==  
NSURLAuthenticationMethodClientCertificate {  
        completionHandler(.rejectProtectionSpace, nil);  
        return  
    }  
    completionHandler(.cancelAuthenticationChallenge, nil)  
}
```

这是信任管理器：（未找到Swift代码）

```
@implementation SSLTrustManager  
+ (BOOL)shouldTrustServerTrust:(SecTrustRef)serverTrust forCert:(NSString*)certName {  
// 加载捆绑的证书。  
NSString *certPath = [[NSBundle mainBundle] pathForResource:certName ofType:@"der"];  
NSData *certData = [[NSData alloc] initWithContentsOfFile:certPath];  
CFDataRef certDataRef = (_bridge_retained CFDataRef)certData;  
SecCertificateRef cert = SecCertificateCreateWithData(NULL, certDataRef);  
  
// 建立以我们捆绑证书为锚点的信任链。  
CFArrayRef certArrayRef = CFArrayCreate(NULL, (void *)&cert, 1, NULL);  
SecTrustSetAnchorCertificates(serverTrust, certArrayRef);  
  
// 验证该信任。  
SecTrustResultType trustResult;
```

iExplorer is good one to explore iTunes backups.

Section 197.2: Transport Security using SSL

iOS apps needs to be written in a way to provide security to data which is being transported over network.

SSL is the common way to do it.

Whenever app tries to call web services to pull or push data to servers, it should **use SSL over HTTP, i.e. HTTPS**.

To do this, **app must call https://server.com/part** such web services and not http://server.com/part.

In this case, app needs to trust the server `server.com` using SSL certificate.

Here is the example of validating server trust-

Implement URLSessionDelegate as:

```
func urlSession(_ session: URLSession, didReceive challenge: URLAuthenticationChallenge, completionHandler: @escaping (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {  
  
    if challenge.protectionSpace.authenticationMethod == NSURLAuthenticationMethodServerTrust {  
        let serverTrust:SecTrust = challenge.protectionSpace.serverTrust!  
  
        func acceptServerTrust() {  
            let credential:URLCredential = URLCredential(trust: serverTrust)  
            challenge.sender?.use(credential, for: challenge)  
            completionHandler(.useCredential, URLCredential(trust:  
challenge.protectionSpace.serverTrust!))  
        }  
  
        let success = SSLTrustManager.shouldTrustServerTrust(serverTrust, forCert:  
"Server_Public_SSL_Cert")  
        if success {  
            acceptServerTrust()  
            return  
        }  
    }  
    else if challenge.protectionSpace.authenticationMethod ==  
NSURLAuthenticationMethodClientCertificate {  
        completionHandler(.rejectProtectionSpace, nil);  
        return  
    }  
    completionHandler(.cancelAuthenticationChallenge, nil)  
}
```

Here is trust manager: (couldn't found Swift code)

```
@implementation SSLTrustManager  
+ (BOOL)shouldTrustServerTrust:(SecTrustRef)serverTrust forCert:(NSString*)certName {  
// Load up the bundled certificate.  
NSString *certPath = [[NSBundle mainBundle] pathForResource:certName ofType:@"der"];  
NSData *certData = [[NSData alloc] initWithContentsOfFile:certPath];  
CFDataRef certDataRef = (_bridge_retained CFDataRef)certData;  
SecCertificateRef cert = SecCertificateCreateWithData(NULL, certDataRef);  
  
// Establish a chain of trust anchored on our bundled certificate.  
CFArrayRef certArrayRef = CFArrayCreate(NULL, (void *)&cert, 1, NULL);  
SecTrustSetAnchorCertificates(serverTrust, certArrayRef);  
  
// Verify that trust.  
SecTrustResultType trustResult;
```

```
SecTrustEvaluate(serverTrust, &trustResult);

// 清理。
CFRelease(certArrayRef);
CFRelease(cert);
CFRelease(certDataRef);

// 我们自定义的信任链是否成功评估？
return trustResult == kSecTrustResultUnspecified;
}
@end
```

Server_Public_SSL_Cert.der 是服务器的公用SSL密钥。

通过这种方法，我们的应用可以确保它正在与预期的服务器通信，且没有人拦截应用与服务器之间的通信。

```
SecTrustEvaluate(serverTrust, &trustResult);

// Clean up.
CFRelease(certArrayRef);
CFRelease(cert);
CFRelease(certDataRef);

// Did our custom trust chain evaluate successfully?
return trustResult == kSecTrustResultUnspecified;
}
@end
```

Server_Public_SSL_Cert.der is servers' public SSL key.

Using this approach our app can make sure that it is communicating to the intended server and no one is intercepting the app-server communication.

第198章：应用传输安全 (ATS)

参数	详细信息
NSAppTransportSecurity	配置ATS 设置为YES以在所有地方禁用ATS。在iOS 10及更高版本和macOS 10.12及更高版本中，如果您的应用程序的Info.plist文件中存在以下任意键，则此键的值将被忽略： NSAllowsArbitraryLoadsInMedia, NSAllowsArbitraryLoadsInWebContent, NSAllowsLocalNetworking
NSAllowsArbitraryLoads	设置为YES以禁用通过AV Foundation框架API加载的媒体的ATS。 （iOS 10+，macOS 10.12+）
NSAllowsArbitraryLoadsInMedia	设置为YES以禁用应用程序的网页视图（WKWebView、UIWebView、WebView）中的ATS，而不影响NSURLSession连接。 （iOS 10+，macOS 10.12+）
NSAllowsArbitraryLoadsInWebContent	设置为YES以禁用对未限定域名和.local域的连接的ATS。 （iOS 10+，macOS 10.12+）
NSAllowsLocalNetworking	为特定域配置例外
NSExtensionDomains	设置为YES以将例外应用于所选域的所有子域。
NSIncludesSubdomains	设置为YES以要求针对域上的服务器（X.509）证书，必须提供来自已知CT日志的有效签名证书透明度（CT）时间戳。 （iOS 10+，macOS 10.12+）
NSRequiresCertificateTransparency	设置为YES以允许所选域使用HTTP。 默认值为YES；设置为NO以禁用前向保密并接受更多密码套件。
NSExtensionAllowsInsecureHTTPLoads	默认值为TLSv1.2；可能的值有：TLSv1.0、TLSv1.1、 TLSv1.2
NSExtensionRequiresForwardSecrecy	类似于 NSExtensionAllowsInsecureHTTPLoads，但适用于您无法控制的域名
NSExtensionMinimumTLSVersion	类似于 NSExtensionRequiresForwardSecrecy，但适用于您无法控制的域名
NSExtensionAllowsInsecureHTTPLoads	类似于 NSExtensionRequiresForwardSecrecy，但适用于您无法控制的域名
NSExtensionRequiresForwardSecrecy	类似于 NSExtensionMinimumTLSVersion，但适用于您无法控制的域名
NSExtensionMinimumTLSVersion	

第198.1节：加载所有HTTP内容

苹果在iOS 9中引入了ATS，作为一项新的安全功能，以提升应用与网络服务之间的隐私和安全。ATS默认会拒绝所有非HTTPS请求。虽然这对生产环境非常有利，但在测试过程中可能会带来不便。

ATS在目标的Info.plist文件中通过NSAppTransportSecurity字典（Xcode Info.plist编辑器中的App Transport Security Settings）进行配置。要允许所有HTTP内容，添加Allow Arbitrary Loads布尔值（NSAllowsArbitraryLoads）并设置为YES。生产应用不推荐这样做，如果需要HTTP内容，建议选择性启用。

第198.2节：选择性加载HTTP内容

与启用所有HTTP内容类似，所有配置均在App Transport Security Settings下进行。向顶层ATS设置中添加Exception Domains字典（NSExtensionDomains）。

对于每个域，向“例外域”中添加一个字典项，其中键为相应的域名。设置
NSExtensionAllowsInsecureHTTPLoads 设置为 YES，以禁用该域的 HTTPS 要求。

Chapter 198: App Transport Security (ATS)

Parameter	Details
NSAppTransportSecurity	Configure ATS Set to YES to disable ATS everywhere. In iOS 10 and later, and macOS 10.12 and later, the value of this key is ignored if any of the following keys are present in your app's Info.plist file: NSAllowsArbitraryLoadsInMedia, NSAllowsArbitraryLoadsInWebContent, NSAllowsLocalNetworking
NSAllowsArbitraryLoads	Set to YES to disable ATS for media loaded using APIs from the AV Foundation framework. (iOS 10+, macOS 10.12+)
NSAllowsArbitraryLoadsInMedia	Set to YES to disable ATS in your app's web views (WKWebView, UIWebView, WebView) without affecting your URLSession connections. (iOS 10+, macOS 10.12+)
NSAllowsArbitraryLoadsInWebContent	Set to YES to disable for connections to unqualified domains and to .local domains. (iOS 10+, macOS 10.12+)
NSAllowsLocalNetworking	Configure exceptions for specific domains
NSExtensionDomains	Set to YES to apply the exceptions to all subdomains of the selected domain.
NSIncludesSubdomains	Set to YES to require that valid, signed Certificate Transparency (CT) timestamps, from known CT logs, be presented for server (X.509) certificates on a domain. (iOS 10+, macOS 10.12+)
NSRequiresCertificateTransparency	Set to YES to allow HTTP on the selected domain.
NSExtensionAllowsInsecureHTTPLoads	Defaults to YES; Set to NO to disable Forward Secrecy and accept more ciphers.
NSExtensionRequiresForwardSecrecy	Defaults to TLSv1.2; Possible values are: TLSv1.0, TLSv1.1, TLSv1.2
NSExtensionMinimumTLSVersion	Similar to NSExtensionAllowsInsecureHTTPLoads, but for domains that you have no control over
NSExtensionAllowsInsecureHTTPLoads	Similar to NSExtensionRequiresForwardSecrecy, but for domains that you have no control over
NSExtensionRequiresForwardSecrecy	Similar to NSExtensionMinimumTLSVersion, but for domains that you have no control over
NSExtensionMinimumTLSVersion	

Section 198.1: Load all HTTP content

Apple introduced ATS with iOS 9 as a new security feature to improve privacy and security between apps and web services. ATS by default fails all non HTTPS requests. While this can be really nice for production environments, it can be a nuisance during testing.

ATS is configured in the target's Info.plist file with the NSAppTransportSecurity dictionary (App Transport Security Settings in the Xcode Info.plist editor). To allow all HTTP content, add the Allow Arbitrary Loads boolean (NSAllowsArbitraryLoads) and set it to YES. This is not recommended for production apps, and if HTTP content is required, it is recommended that it be selectively enabled instead.

Section 198.2: Selectively load HTTP content

Similar to enabling all HTTP content, all configuration happens under the App Transport Security Settings. Add the Exception Domains dictionary (NSExtensionDomains) to the top level ATS settings.

For every domain, add a dictionary item to the Exception Domains, where the key is the domain in question. Set NSExtensionAllowsInsecureHTTPLoads to YES to disable the HTTPS requirement for that domain.

第198.3节：端点需要 SSL

自 iOS 9 起，所有端点必须遵守 HTTPS 规范。

任何未使用 SSL 的端点将在控制台日志中出现警告。对于您的应用程序来说，网络连接将显示为失败。

要配置例外，请将以下内容放入您的 Info.plist 文件：

- 仅允许特定域 (testdomain.com)：

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSEceptionDomains</key>
<dict>
<key>testdomain.com</key>
<dict>
<key>NSIncludesSubdomains</key>
<true/>
<key>NSEceptionAllowsInsecureHTTPLoads</key>
<true/>
</dict>
</dict>
</dict>
```

允许此类行为的键是 NSEceptionAllowsInsecureHTTPLoads。在此情况下，应用程序将仅允许与指定域 (testdomain.com) 的 HTTP 连接，并阻止所有其他 HTTP 连接。

关键字 NSIncludesSubdomains 指定提到的域名 (testdomain.com) 的任何及所有子域名也应被允许。

- 允许任何域名：

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSAllowsArbitraryLoads</key>
<true/>
</dict>
```

在这种情况下，应用程序将允许对任何域名的 HTTP 连接。自 2017 年 1 月 1 日起，使用此标志将导致 App Store 的严格审核，应用开发者必须解释他们为何首先需要使用此例外。可能的解释包括：

- 加载不包含个性化信息的加密媒体内容的应用程序。
- 连接到无法升级为使用安全连接的设备。
- 连接到由其他实体管理且不支持安全连接的服务器。

Section 198.3: Endpoints require SSL

Introduced in iOS 9, all endpoints must adhere to the HTTPS specification.

Any endpoints not using SSL will fail with a warning in the console log. To your application it will appear that the internet connection failed.

To configure exceptions: Place the following in your Info.plist file:

- Allow particular domain (testdomain.com) **only**:

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSEceptionDomains</key>
<dict>
<key>testdomain.com</key>
<dict>
<key>NSIncludesSubdomains</key>
<true/>
<key>NSEceptionAllowsInsecureHTTPLoads</key>
<true/>
</dict>
</dict>
</dict>
```

The key that permits such behavior is NSEceptionAllowsInsecureHTTPLoads. In this case, app will allow HTTP connection to mentioned domain (testdomain.com) only and block all other HTTP connections.

The key NSIncludesSubdomains specifies that any and all **subdomains** of the mentioned domain (testdomain.com) should also be allowed.

- Allow any domain:

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSAllowsArbitraryLoads</key>
<true/>
</dict>
```

In this case, app will allow HTTP connection to **any** domain. As of January 1st 2017, using this flag will cause thorough App Store review and the app developers will have to explain why they need to use this exception in the first place. Possible explanations include:

- An application that loads encrypted media content that contains no personalized information.
- Connections to devices that cannot be upgraded to use secure connections.
- Connection to a server that is managed by another entity and does not support secure connections.

第199章：选择最佳iOS架构模式的指南

揭开MVC、MVP、MVVM和VIPER或其他设计模式的神秘面纱，选择构建应用程序的最佳方法

第199.1节：MVP模式

```
import UIKit

结构体 Person { // 模型
    let firstName: String
    let lastName: String
}

协议 GreetingView: 类 {
    函数 setGreeting(greeting: 字符串)
}

协议 GreetingViewPresenter {
    初始化(view: GreetingView, person: Person)
    函数 showGreeting()
}

类 GreetingPresenter : GreetingViewPresenter {
    无主引用 让 view: GreetingView
    让 person: Person
    必需初始化(view: GreetingView, person: Person) {
        self.view = view
        self.person = person
    }
    函数 showGreeting() {
        让 greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.view.setGreeting(greeting)
    }
}

class GreetingViewController : UIViewController, GreetingView {
    var presenter: GreetingViewPresenter!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents: .TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        self.presenter.showGreeting()
    }

    func setGreeting(greeting: String) {
        self.greetingLabel.text = greeting
    }

    // 布局代码写在这里
}
// MVP的组装
```

Chapter 199: Guideline to choose best iOS Architecture Patterns

Demystifying MVC, MVP, MVVM and VIPER or any other design patterns to choose the best approach to building an app

Section 199.1: MVP Patterns

```
import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

protocol GreetingView: class {
    func setGreeting(greeting: String)
}

protocol GreetingViewPresenter {
    init(view: GreetingView, person: Person)
    func showGreeting()
}

class GreetingPresenter : GreetingViewPresenter {
    unowned let view: GreetingView
    let person: Person
    required init(view: GreetingView, person: Person) {
        self.view = view
        self.person = person
    }
    func showGreeting() {
        let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.view.setGreeting(greeting)
    }
}

class GreetingViewController : UIViewController, GreetingView {
    var presenter: GreetingViewPresenter!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents: .TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        self.presenter.showGreeting()
    }

    func setGreeting(greeting: String) {
        self.greetingLabel.text = greeting
    }

    // layout code goes here
}
// Assembling of MVP
```

```

let model = Person(firstName: "大卫", lastName: "布莱恩")
let view = GreetingViewController()
let presenter = GreetingPresenter(view: view, person: model)
view.presenter = presenter

```

第199.2节：MVVM模式

```

import UIKit

结构体 Person { // 模型
    let firstName: String
    let lastName: String
}

协议 GreetingViewModelProtocol: 类 {
    变量 greeting: 字符串? { 获取 }
    变量 greetingDidChange: ((GreetingViewModelProtocol) -> ())? { 获取 设置 } // 当greeting改变时调用的函数

    初始化(person: Person)
    函数 showGreeting()
}

类 GreetingViewModel : GreetingViewModelProtocol {
    常量 person: Person
    变量 greeting: 字符串? {
        属性观察器 {
            self.greetingDidChange?(self)
        }
    }
    变量 greetingDidChange: ((GreetingViewModelProtocol) -> ())?
    必需初始化(person: Person) {
        self.person = person
    }
    函数 showGreeting() {
        self.greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
    }
}

class GreetingViewController : UIViewController {
    var viewModel: GreetingViewModelProtocol! {
        didSet {
            self.viewModel.greetingDidChange = { [unowned self] viewModel in
                self.greetingLabel.text = viewModel.greeting
            }
        }
    }
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self.viewModel, action: "showGreeting", forControlEvents: .TouchUpInside)
    }
    // 布局代码写在这里
}
// 组装MVVM
let model = Person(firstName: "David", lastName: "Blaine")
let viewModel = GreetingViewModel(person: model)
let view = GreetingViewController()

```

```

let model = Person(firstName: "David", lastName: "Blaine")
let view = GreetingViewController()
let presenter = GreetingPresenter(view: view, person: model)
view.presenter = presenter

```

Section 199.2: MVVM Pattern

```

import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

protocol GreetingViewModelProtocol: class {
    var greeting: String? { get }
    var greetingDidChange: ((GreetingViewModelProtocol) -> ())? { get set } // function to call
    when greeting did change
        init(person: Person)
        func showGreeting()
}

class GreetingViewModel : GreetingViewModelProtocol {
    let person: Person
    var greeting: String? {
        didSet {
            self.greetingDidChange?(self)
        }
    }
    var greetingDidChange: ((GreetingViewModelProtocol) -> ())?
    required init(person: Person) {
        self.person = person
    }
    func showGreeting() {
        self.greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
    }
}

class GreetingViewController : UIViewController {
    var viewModel: GreetingViewModelProtocol! {
        didSet {
            self.viewModel.greetingDidChange = { [unowned self] viewModel in
                self.greetingLabel.text = viewModel.greeting
            }
        }
    }
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self.viewModel, action: "showGreeting", forControlEvents: .TouchUpInside)
    }
    // layout code goes here
}
// Assembling of MVVM
let model = Person(firstName: "David", lastName: "Blaine")
let viewModel = GreetingViewModel(person: model)
let view = GreetingViewController()

```

第199.3节：VIPER模式

```

import UIKit

结构体 Person { // 实体 (通常更复杂, 例如 NSManagedObject)
    let firstName: String
    let lastName: String
}

struct GreetingData { // 传输数据结构 (非实体)
    let greeting: String
    let subject: String
}

protocol GreetingProvider {
    func provideGreetingData()
}

protocol GreetingOutput: class {
    func receiveGreetingData(greetingData: GreetingData)
}

class GreetingInteractor : GreetingProvider {
    weak var output: GreetingOutput!

    func provideGreetingData() {
        let person = Person(firstName: "David", lastName: "Blaine") // 通常来自数据访问层

        let subject = person.firstName + " " + person.lastName
        let greeting = GreetingData(greeting: "Hello", subject: subject)
        self.output.receiveGreetingData(greeting)
    }
}

protocol GreetingViewEventHandler {
    func didTapShowGreetingButton()
}

protocol GreetingView: class {
    func setGreeting(greeting: String)
}

class GreetingPresenter : GreetingOutput, GreetingViewEventHandler {
    weak var view: GreetingView!
    var greetingProvider: GreetingProvider!

    func didTapShowGreetingButton() {
        self.greetingProvider.provideGreetingData()
    }

    func receiveGreetingData(greetingData: GreetingData) {
        let greeting = greetingData.greeting + " " + greetingData.subject
        self.view.setGreeting(greeting)
    }
}

class GreetingViewController : UIViewController, GreetingView {
    var eventHandler: GreetingViewEventHandler!
    let showGreetingButton = UIButton()
}

```

Section 199.3: VIPER Pattern

```

import UIKit

struct Person { // Entity (usually more complex e.g. NSManagedObject)
    let firstName: String
    let lastName: String
}

struct GreetingData { // Transport data structure (not Entity)
    let greeting: String
    let subject: String
}

protocol GreetingProvider {
    func provideGreetingData()
}

protocol GreetingOutput: class {
    func receiveGreetingData(greetingData: GreetingData)
}

class GreetingInteractor : GreetingProvider {
    weak var output: GreetingOutput!

    func provideGreetingData() {
        let person = Person(firstName: "David", lastName: "Blaine") // usually comes from data
access layer
        let subject = person.firstName + " " + person.lastName
        let greeting = GreetingData(greeting: "Hello", subject: subject)
        self.output.receiveGreetingData(greeting)
    }
}

protocol GreetingViewEventHandler {
    func didTapShowGreetingButton()
}

protocol GreetingView: class {
    func setGreeting(greeting: String)
}

class GreetingPresenter : GreetingOutput, GreetingViewEventHandler {
    weak var view: GreetingView!
    var greetingProvider: GreetingProvider!

    func didTapShowGreetingButton() {
        self.greetingProvider.provideGreetingData()
    }

    func receiveGreetingData(greetingData: GreetingData) {
        let greeting = greetingData.greeting + " " + greetingData.subject
        self.view.setGreeting(greeting)
    }
}

class GreetingViewController : UIViewController, GreetingView {
    var eventHandler: GreetingViewEventHandler!
    let showGreetingButton = UIButton()
}

```

```

let greetingLabel = UILabel()

override func viewDidLoad() {
    super.viewDidLoad()
    self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
}

func didTapButton(button: UIButton) {
    self.eventHandler.didTapShowGreetingButton()
}

func setGreeting(greeting: String) {
    self.greetingLabel.text = greeting
}

// 布局代码写在这里
}

// 组装 VIPER 模块，无 Router
let view = GreetingViewController()
let presenter = GreetingPresenter()
let interactor = GreetingInteractor()
view.eventHandler = presenter
presenter.view = view
presenter.greetingProvider = interactor
interactor.output = presenter

```

第199.4节：MVC 模式

```

import UIKit

struct Person { // 模型
    let firstName: String
    let lastName: String
}

class GreetingViewController : UIViewController { // 视图 + 控制器
    var person: Person!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.greetingLabel.text = greeting
    }

    // 布局代码写在这里
}

// MVC的组装
let model = Person(firstName: "大卫", lastName: "布莱恩")
let view = GreetingViewController()
view.person = model

```

```

let greetingLabel = UILabel()

override func viewDidLoad() {
    super.viewDidLoad()
    self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
}

func didTapButton(button: UIButton) {
    self.eventHandler.didTapShowGreetingButton()
}

func setGreeting(greeting: String) {
    self.greetingLabel.text = greeting
}

// layout code goes here
}

// Assembling of VIPER module, without Router
let view = GreetingViewController()
let presenter = GreetingPresenter()
let interactor = GreetingInteractor()
view.eventHandler = presenter
presenter.view = view
presenter.greetingProvider = interactor
interactor.output = presenter

```

Section 199.4: MVC pattern

```

import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

class GreetingViewController : UIViewController { // View + Controller
    var person: Person!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.greetingLabel.text = greeting
    }

    // layout code goes here
}

// Assembling of MVC
let model = Person(firstName: "David", lastName: "Blaine")
let view = GreetingViewController()
view.person = model

```

第200章：多播委托

为现有iOS控件添加多播功能的模式。添加多播可以提高清晰度和代码重用性。

第200.1节：适用于任何控件的多播委托

通过委托将消息从一个对象转发给另一个对象，将这些消息多播给多个观察者。

步骤1：创建 `NSObject` 类 `RRMulticastDelegate`

步骤2：在 `RRMulticastDelegate.h` 文件中实现以下代码

```
#import <Foundation/Foundation.h>

@interface RRMulticastDelegate : NSObject

{
    //处理委托的多个观察者
    NSMutableArray* _delegates;
}

// 将委托方法的实现转发给观察者列表
- (void)addDelegate:(id)delegate;
- (void)removeDelegate:(id)delegate;

// 获取多个代理
-(NSArray *)delegatesObjects;

@end
```

步骤3：在 `RRMulticastDelegate.m` 文件中实现以下代码

```
#import "RRMulticastDelegate.h"

@implementation RRMulticastDelegate

- (id)init
{
    if (self = [super init])
    {
        _delegates = [NSMutableArray array];
    }
    return self;
}

-(NSArray *)delegatesObjects
{
    return _delegates;
}

- (void)removeDelegate:(id)delegate
{
    if ([_delegates containsObject:delegate])
        [_delegates removeObject:delegate];
}

- (void)addDelegate:(id)delegate
```

Chapter 200: Multicast Delegates

Pattern for adding multicasting capabilities to existing iOS controls. Adding multicasting allows for improved clarity and code re-use.

Section 200.1: Multicast delegates for any controls

Forward messages one object to another by delegates, multicasting these messages to multiple observers..

Step 1: Create `NSObject` class of `RRMulticastDelegate`

Step 2: Following code implement in `RRMulticastDelegate.h` file

```
#import <Foundation/Foundation.h>

@interface RRMulticastDelegate : NSObject

{
    //Handle multiple observers of delegate
    NSMutableArray* _delegates;
}

// Delegate method implementation to the list of observers
- (void)addDelegate:(id)delegate;
- (void)removeDelegate:(id)delegate;

// Get multiple delegates
-(NSArray *)delegatesObjects;

@end
```

Step 3: Following code implement in `RRMulticastDelegate.m` file

```
#import "RRMulticastDelegate.h"

@implementation RRMulticastDelegate

- (id)init
{
    if (self = [super init])
    {
        _delegates = [NSMutableArray array];
    }
    return self;
}

-(NSArray *)delegatesObjects
{
    return _delegates;
}

- (void)removeDelegate:(id)delegate
{
    if ([_delegates containsObject:delegate])
        [_delegates removeObject:delegate];
}

- (void)addDelegate:(id)delegate
```

```

{
    if (![_delegates containsObject:delegate])
        [_delegates addObject:delegate];
}

- (BOOL)respondsToSelector:(SEL)aSelector
{
    if ([super respondsToSelector:aSelector])
        return YES;

    // 如果任何代理响应此选择器，则返回YES
    for(id delegate in _delegates)
    {
        if (!delegate)
            continue;

        if ([delegate respondsToSelector:aSelector])
        {
            return YES;
        }
    }
    return NO;
}

- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
{
    // 这个类能创建签名吗？
    NSMethodSignature* signature = [super methodSignatureForSelector:aSelector];

    // 如果不能，尝试我们的代理
    if (!signature)
    {
        for(id delegate in _delegates)
        {
            if (!delegate)
                continue;

            if ([delegate respondsToSelector:aSelector])
            {
                return [delegate methodSignatureForSelector:aSelector];
            }
        }
    }
    return signature;
}

- (void)forwardInvocation:(NSInvocation *)anInvocation
{
    // 将调用转发给每个代理
    for(id delegate in _delegates)
    {
        if (!delegate)
            continue;

        if ([delegate respondsToSelector:[anInvocation selector]])
        {
            [anInvocation invokeWithTarget:delegate];
        }
    }
}

@end

```

```

{
    if (![_delegates containsObject:delegate])
        [_delegates addObject:delegate];
}

- (BOOL)respondsToSelector:(SEL)aSelector
{
    if ([super respondsToSelector:aSelector])
        return YES;

    // if any of the delegates respond to this selector, return YES
    for(id delegate in _delegates)
    {
        if (!delegate)
            continue;

        if ([delegate respondsToSelector:aSelector])
        {
            return YES;
        }
    }
    return NO;
}

- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
{
    // can this class create the signature?
    NSMethodSignature* signature = [super methodSignatureForSelector:aSelector];

    // if not, try our delegates
    if (!signature)
    {
        for(id delegate in _delegates)
        {
            if (!delegate)
                continue;

            if ([delegate respondsToSelector:aSelector])
            {
                return [delegate methodSignatureForSelector:aSelector];
            }
        }
    }
    return signature;
}

- (void)forwardInvocation:(NSInvocation *)anInvocation
{
    // forward the invocation to every delegate
    for(id delegate in _delegates)
    {
        if (!delegate)
            continue;

        if ([delegate respondsToSelector:[anInvocation selector]])
        {
            [anInvocation invokeWithTarget:delegate];
        }
    }
}

@end

```

步骤4：创建 NSObject 类别类 RRProperty

步骤5：在 NSObject+RRProperty.h 文件中实现以下代码

```
#import <Foundation/Foundation.h>

#import "RRMulticastDelegate.h"

@interface NSObject (RRProperty)<UITextFieldDelegate,UITableViewDataSource>

-(void)setObject:(id)block forKey:(NSString *)key;
-(id)objectForKey:(NSString *)key;

#pragma mark - 多播代理

- (RRMulticastDelegate *)multicastDelegate;
- (RRMulticastDelegate *)multicastDatasource;

-(void)addDelegate:(id)delegate;
-(void)addDataSource:(id)datasource;

@end
```

步骤6：以下代码实现于NSObject+RRProperty.m文件中

```
#import "NSObject+RRProperty.h"

#import <objc/message.h>
#import <objc/runtime.h>

#pragma GCC diagnostic ignored "-Wprotocol"

static NSString *const MULTICASTDELEGATE = @""MULTICASTDELEGATE";
static NSString *const MULTICASTDATASOURCE = @""MULTICASTDATASOURCE";

@implementation NSObject (RRProperty)

-(void)setObject:(id)block forKey:(NSString *)key
{
objc_setAssociatedObject(self, (__bridge const void *)(key), block, 0BJC_ASSOCIATION_RETAIN);
}

-(id)objectForKey:(NSString *)key
{
    return objc_getAssociatedObject(self, (__bridge const void *)(key));
}

#pragma mark - 多播代理

- (RRMulticastDelegate *)multicastDelegate
{
id multicastDelegate = [self objectForKey:MULTICASTDELEGATE];
if (multicastDelegate == nil) {
multicastDelegate = [[RRMulticastDelegate alloc] init];

    [self setObject:multicastDelegate forKey:MULTICASTDELEGATE];
}

return multicastDelegate;
}
```

Step 4: Create NSObject category class of RRProperty

Step 5: Following code implement in NSObject+RRProperty.h file

```
#import <Foundation/Foundation.h>

#import "RRMulticastDelegate.h"

@interface NSObject (RRProperty)<UITextFieldDelegate,UITableViewDataSource>

-(void)setObject:(id)block forKey:(NSString *)key;
-(id)objectForKey:(NSString *)key;

#pragma mark - Multicast Delegate

- (RRMulticastDelegate *)multicastDelegate;
- (RRMulticastDelegate *)multicastDatasource;

-(void)addDelegate:(id)delegate;
-(void)addDataSource:(id)datasource;

@end
```

Step 6: Following code implement in NSObject+RRProperty.m file

```
#import "NSObject+RRProperty.h"

#import <objc/message.h>
#import <objc/runtime.h>

#pragma GCC diagnostic ignored "-Wprotocol"

static NSString *const MULTICASTDELEGATE = @""MULTICASTDELEGATE";
static NSString *const MULTICASTDATASOURCE = @""MULTICASTDATASOURCE";

@implementation NSObject (RRProperty)

-(void)setObject:(id)block forKey:(NSString *)key
{
objc_setAssociatedObject(self, (__bridge const void *)(key), block, 0BJC_ASSOCIATION_RETAIN);
}

-(id)objectForKey:(NSString *)key
{
    return objc_getAssociatedObject(self, (__bridge const void *)(key));
}

#pragma mark - Multicast Delegate

- (RRMulticastDelegate *)multicastDelegate
{
id multicastDelegate = [self objectForKey:MULTICASTDELEGATE];
if (multicastDelegate == nil) {
multicastDelegate = [[RRMulticastDelegate alloc] init];

    [self setObject:multicastDelegate forKey:MULTICASTDELEGATE];
}

return multicastDelegate;
}
```

```

- (RRMulticastDelegate *)multicastDatasource
{
id multicastDatasource = [self objectForKey:MULTICASTDATASOURCE];
    if (multicastDatasource == nil) {
multicastDatasource = [[RRMulticastDelegate alloc] init];

    [self setObject:multicastDatasource forKey:MULTICASTDATASOURCE];
}

return multicastDatasource;
}

-(void)addDelegate:(id)delegate
{
    [self.multicastDelegate addDelegate:delegate];

    UITextField *text = (UITextField *) self;
    text.delegate = self.multicastDelegate;
}

-(void)addDataSource:(id)datasource
{
    [self.multicastDatasource addDelegate:datasource];
    UITableView *text = (UITableView *) self;
    text.dataSource = self.multicastDatasource;
}

@end

```

最后你可以对任何控件使用多播代理...

例如...

在NSObject+RRProperty.h文件中导入你的视图控制器类，以访问其方法来设置多播代理/数据源。

```

UITextView *txtView = [[UITextView alloc]initWithFrame:txtframe];
[txtView addDelegate:self];

UITableView *tblView = [[UITableView alloc]initWithFrame:tblframe];
[tblView addDelegate:self];
[tblView addDataSource:self];

```

```

- (RRMulticastDelegate *)multicastDatasource
{
    id multicastDatasource = [self objectForKey:MULTICASTDATASOURCE];
    if (multicastDatasource == nil) {
        multicastDatasource = [[RRMulticastDelegate alloc] init];

        [self setObject:multicastDatasource forKey:MULTICASTDATASOURCE];
    }

    return multicastDatasource;
}

-(void)addDelegate:(id)delegate
{
    [self.multicastDelegate addDelegate:delegate];

    UITextField *text = (UITextField *) self;
    text.delegate = self.multicastDelegate;
}

-(void)addDataSource:(id)datasource
{
    [self.multicastDatasource addDelegate:datasource];
    UITableView *text = (UITableView *) self;
    text.dataSource = self.multicastDatasource;
}

@end

```

Finally you can use multicast delegate for any controls...

For ex...

Import your viewcontroller class in [NSObject+RRProperty.h](#) file to access its methods to **set multicast delegate/datasource**.

```

UITextView *txtView = [[UITextView alloc]initWithFrame:txtframe];
[txtView addDelegate:self];

UITableView *tblView = [[UITableView alloc]initWithFrame:tblframe];
[tblView addDelegate:self];
[tblView addDataSource:self];

```

第201章：使用图像资源

图像资源用于在我们的iOS应用中通过Xcode管理和组织不同类型的图像资源。

这些资源可以是应用图标、启动图像、应用中使用的图像、全尺寸图像、随机尺寸图像等。

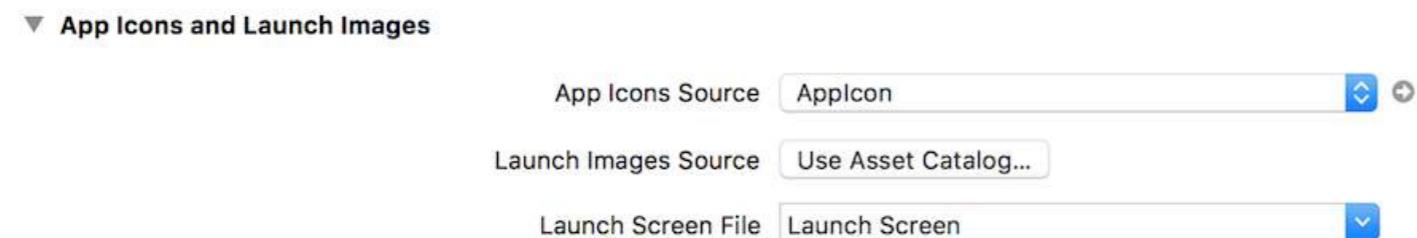
第201.1节：使用图像资源的启动图像

启动屏幕是在启动应用时出现的屏幕，持续到应用的第一个屏幕出现为止。

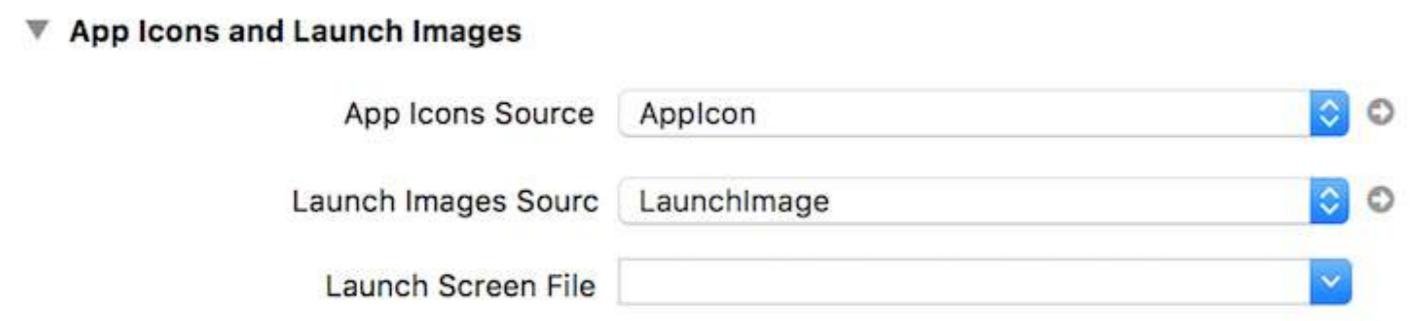
[在这里了解更多关于启动屏幕及其指南的信息。](#)

类似于应用图标，我们必须在项目设置中说明使用图像资源作为启动屏幕图像。

默认项目设置如下：



我们必须改成这样：



一旦我们更改这些设置，Xcode 会自动提示我们迁移到资产并在资产中创建 LaunchImage 文件，如下所示：

Chapter 201: Using Image Assets

Image assets are used to manage and organize different types of image assets in our iOS app using Xcode.

These assets can be **App Icons, Launch Images, images used throughout the app, full size images, random sized images** etc.

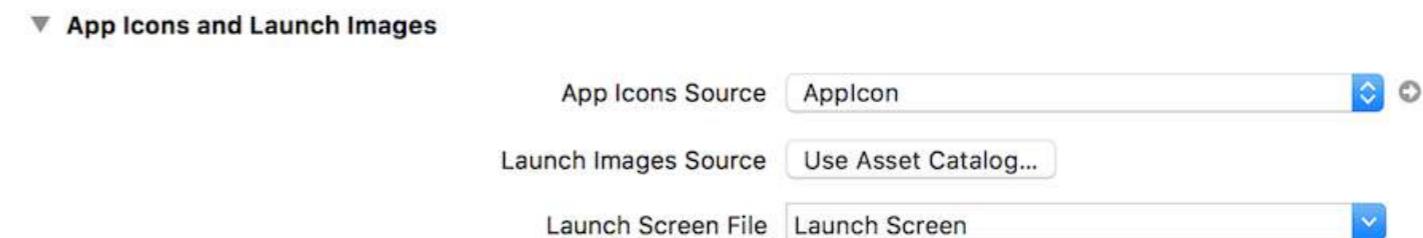
Section 201.1: LaunchImage using Image Assets

Launch screen is a screen which appears while launching app and lasts till first screen of app appears.

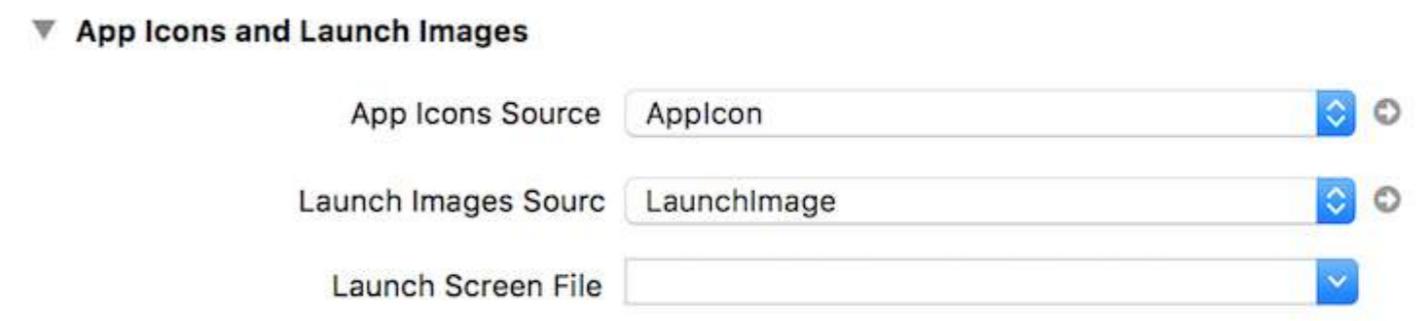
Learn more about [Launch Screen and guidelines here](#).

Similar to AppIcons we have to mention in project settings about using image assets for launch screen image.

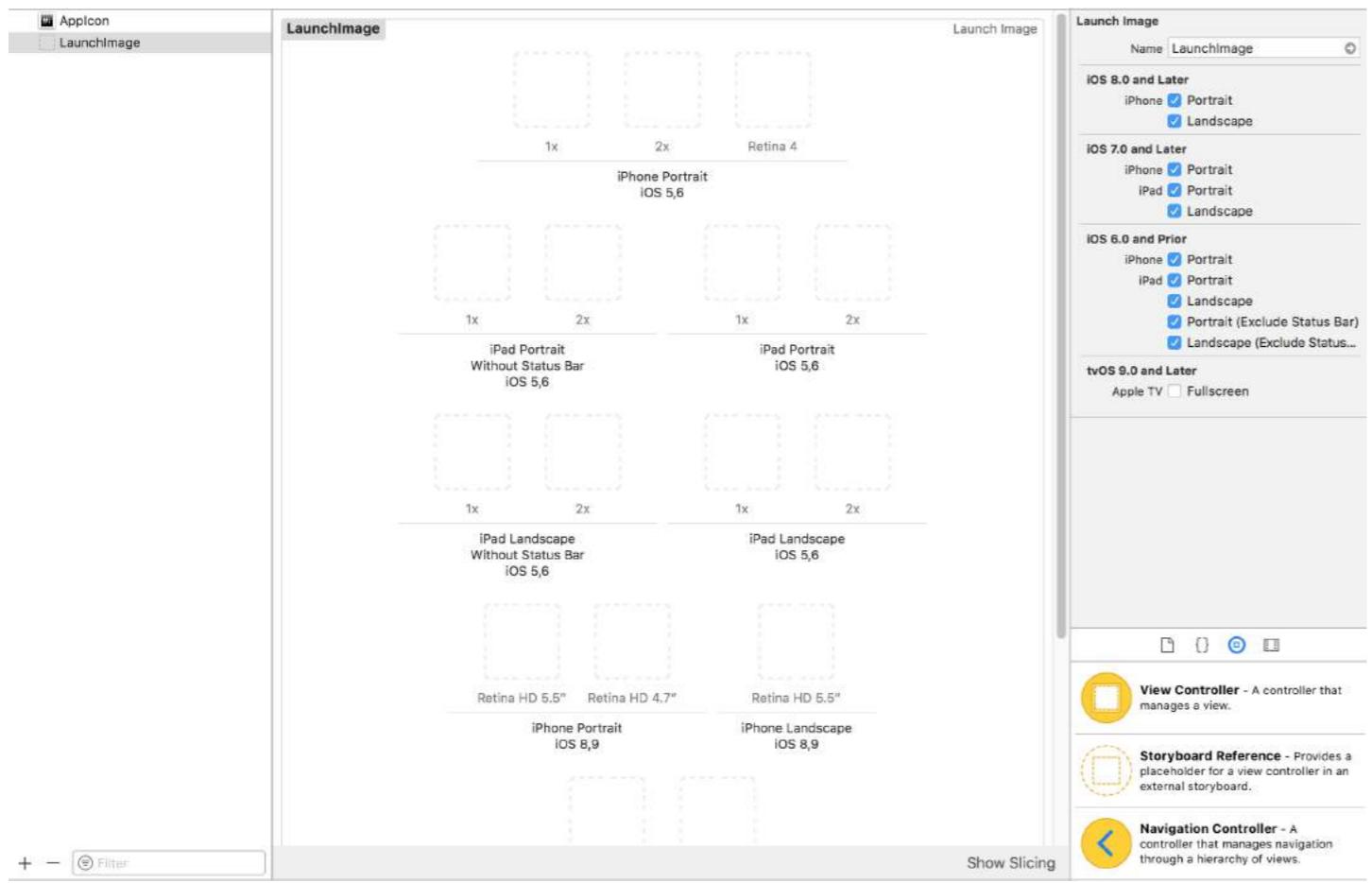
By default project settings are like:



We have to change to like this:



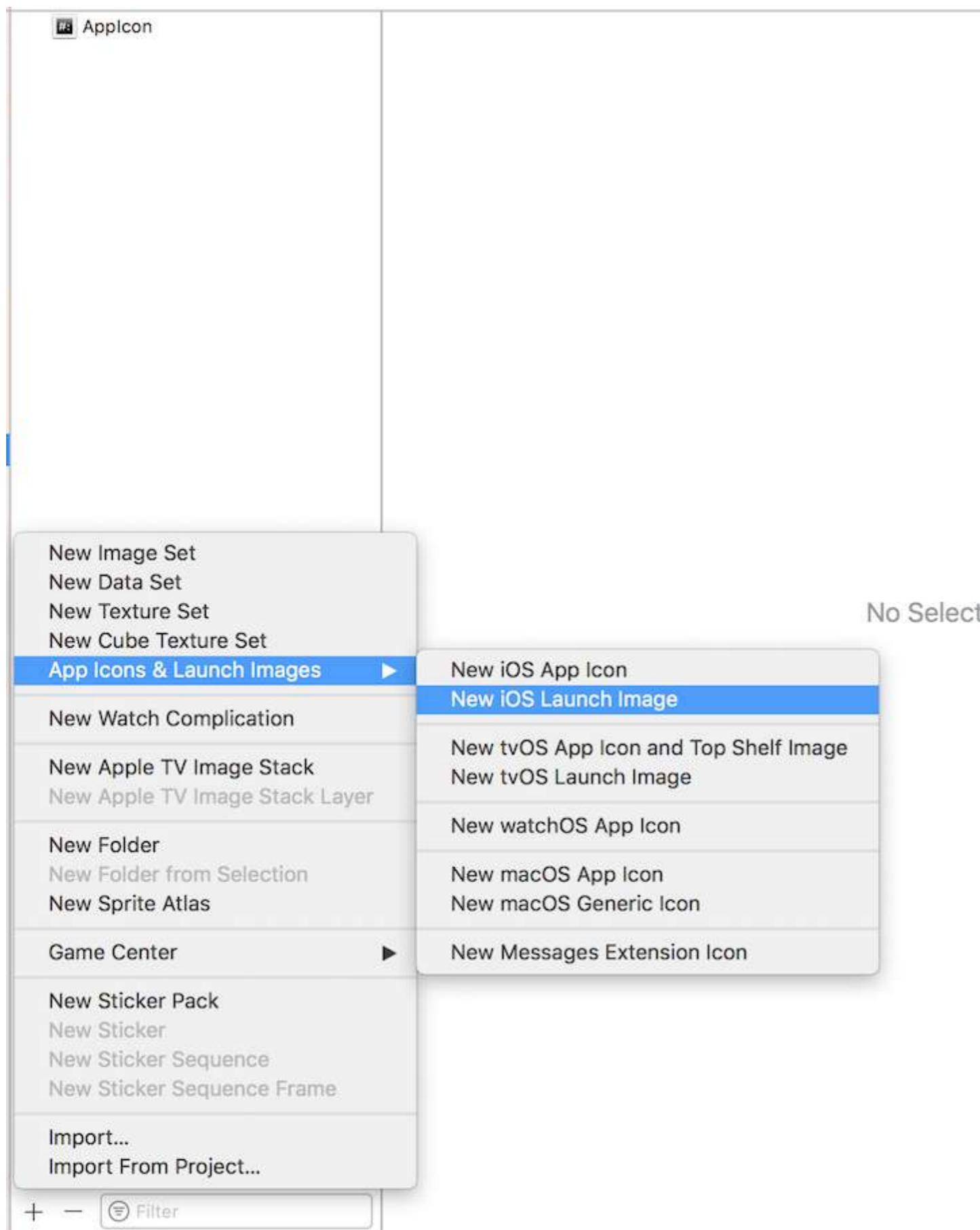
Once we change these settings, Xcode will ask us to migrate to assets and create LaunchImage file in assets automatically as:



如果未创建，我们可以通过点击底部的+按钮手动创建，如下所示：

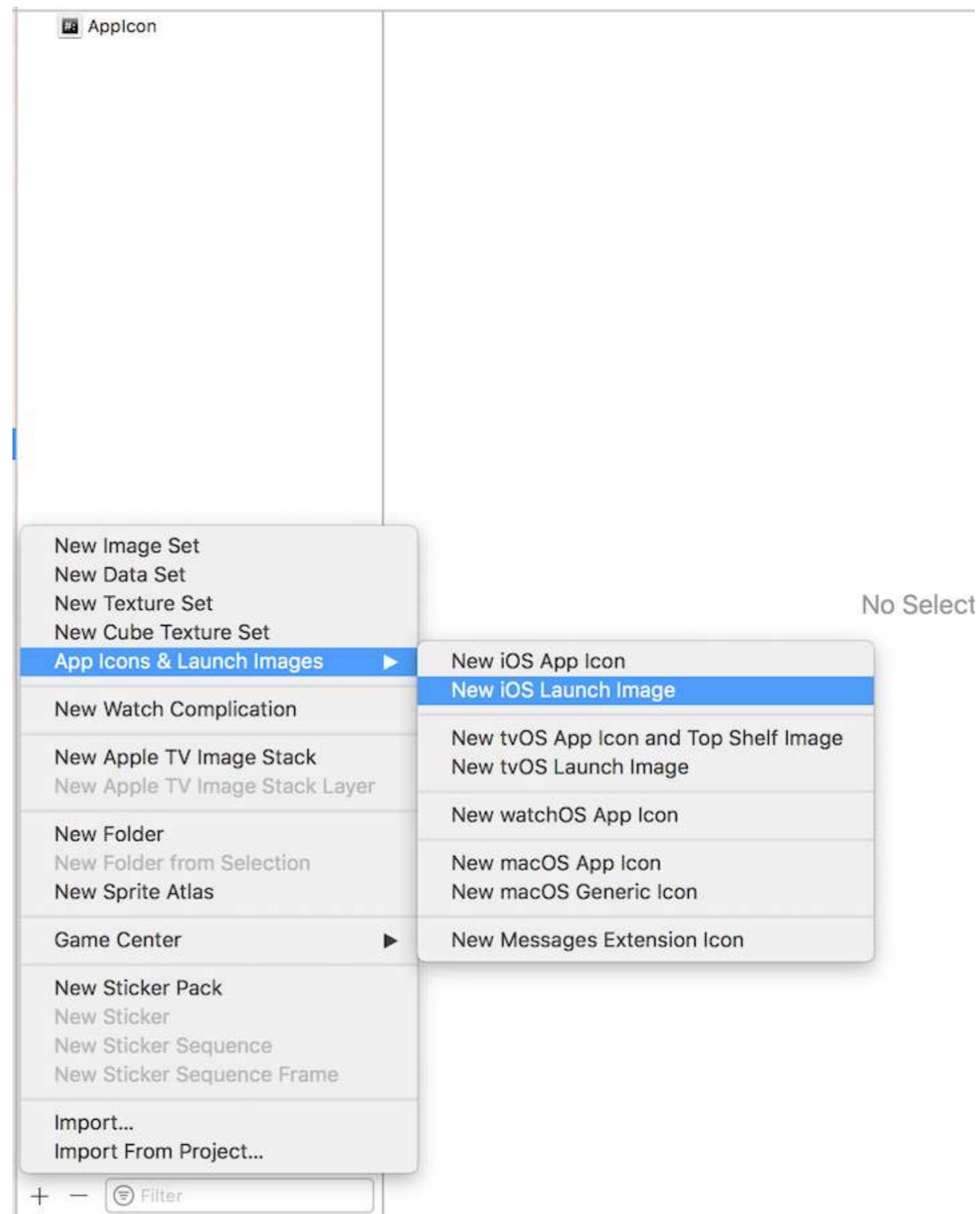


If not created, we can manually create one by clicking + button at the bottom as:



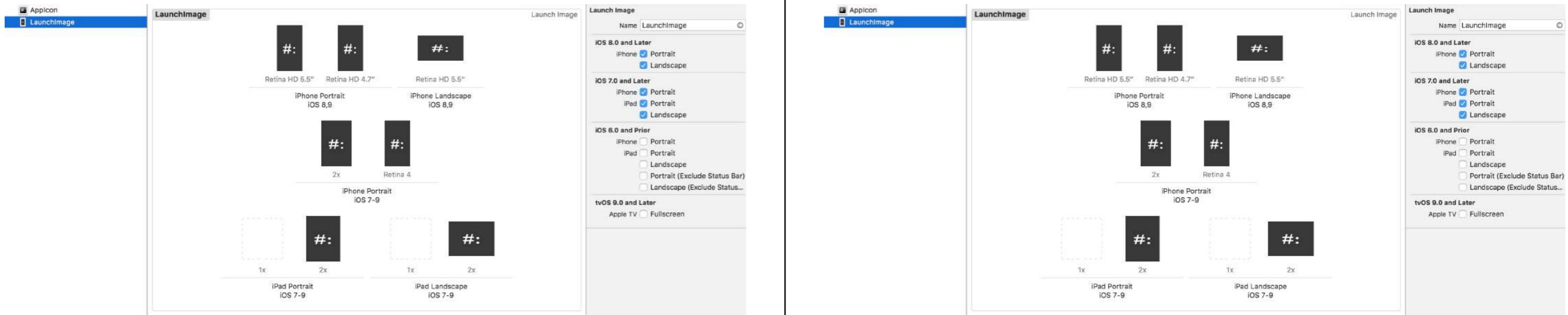
之后，根据我们的需求，可以通过属性检查器勾选/取消勾选框来将空白框更改为支持的设备。

我为屏幕尺寸从4英寸到5.5英寸的iPhone以及所有iPad填充了这些图片，如下所示：



After this, according to our requirement we can change the empty boxes to devices which we support using attributes inspector by checking/unchecking boxes.

I filled these images for iPhones of 4" screen to 5.5" and for all iPads as:



以下是所有启动图片的尺寸：

Retina HD 5.5 英寸 iPhone 坚屏 - iPhone (6, 6S, 7) Plus - 1242x2208 像素

Retina HD 4.7" iPhone 坚屏 - iPhone 6、6S、7 - 750x1334 像素

Retina HD 5.5 英寸 iPhone 横屏 - iPhone (6, 6S, 7) Plus - 2208x1242 像素

2倍 iPhone 坚屏 - (3.5") iPhone 4S - 640x960 像素

Retina 4 英寸 iPhone 坚屏 - (4") iPhone 5, 5S, 5C, iPod Touch, SE - 640x1136 像素

2倍 iPad 坚屏 - 所有 Retina iPad - 1536x2048 像素

2倍 iPad 横屏 - 所有 Retina iPad - 2048x1536 像素

Here are sizes of all launch images:

Retina HD 5.5" iPhone Portrait - iPhone (6, 6S, 7)Plus - 1242x2208px

Retina HD 4.7" iPhone Portrait - iPhone 6, 6S, 7 - 750x1334px

Retina HD 5.5" iPhone Landscape - iPhone (6, 6S, 7)Plus - 2208x1242px

2x iPhone Portrait - (3.5") iPhone 4S - 640x960px

Retina 4 iPhone Portrait - (4") iPhone 5, 5S, 5C, iPod Touch, SE - 640x1136px

2x iPad Portrait - All Retina iPads - 1536x2048px

2x iPad Landscape - All Retina iPads - 2048x1536px

注意：

1 非 Retina iPad: 我留空了 1倍 iPad 坚屏和横屏 因为非 Retina iPad 会通过缩放使用 2倍 启动画面图片

2 12.9英寸 iPad Pro: 这个 iPad 没有对应的方形图标，因为该 iPad 也会通过缩放使用 2倍 iPad 图片

3 Retina HD 5.5英寸: iPad 应该使用 1920x1080像素 坚屏和 1080x1920像素 横屏，但 Xcode 会给出警告，且启动画面不会在这些设备上显示

4 分屏视图: 由于我们使用的是 LaunchImage Asset 而非 LaunchScreen XIB，我们的应用将不支持 iPad 和 5.5英寸横屏 iPhone 的 分屏视图 功能

5 重新安装: 如果我们的应用已经安装在设备上，并尝试使用这些新添加的启动画面资源运行应用，有时设备启动应用时不会显示启动画面。此时只需从设备删除应用，清理+构建项目并运行，即可显示新的启动画面

第201.2节：使用图像资源的应用图标

每当我们在 Xcode 中为新应用创建新项目时，它会提供各种内置类、目标、测试、plist 文件等。同样，它还会提供一个 Assets.xcassets 文件，用于管理项目中的所有图像资源。

这是该文件在文件导航器中的显示方式：

Notes:

1 **non-retina iPads:** I left blank 1x iPad Portrait and Landscape because non-retina iPads will use 2x launch images by scaling

2 **12.9" iPad Pro:** there is no square for this iPad because this iPad will also use 2x iPad images by scaling them

3 **Retina HD 5.5":** iPads should have 1920x1080px for portrait and 1080x1920px for landscape but Xcode will give warning and launch image will not be shown on those devices

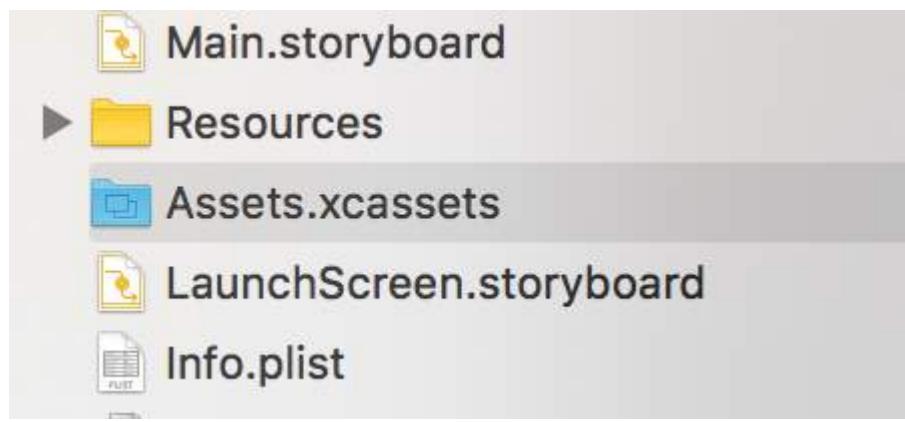
4 **SplitView:** as we are using LaunchImage Asset instead of LaunchScreen XIB, our app will not support SplitView on iPads and landscape 5.5" iPhones

5 **Reinstall:** if our app is already installed on device and we try to run with these newly added launch image assets, then sometimes device will not show launch images while launching app. In this case just delete app from device, clean+build project and run it, it'll show new launch images

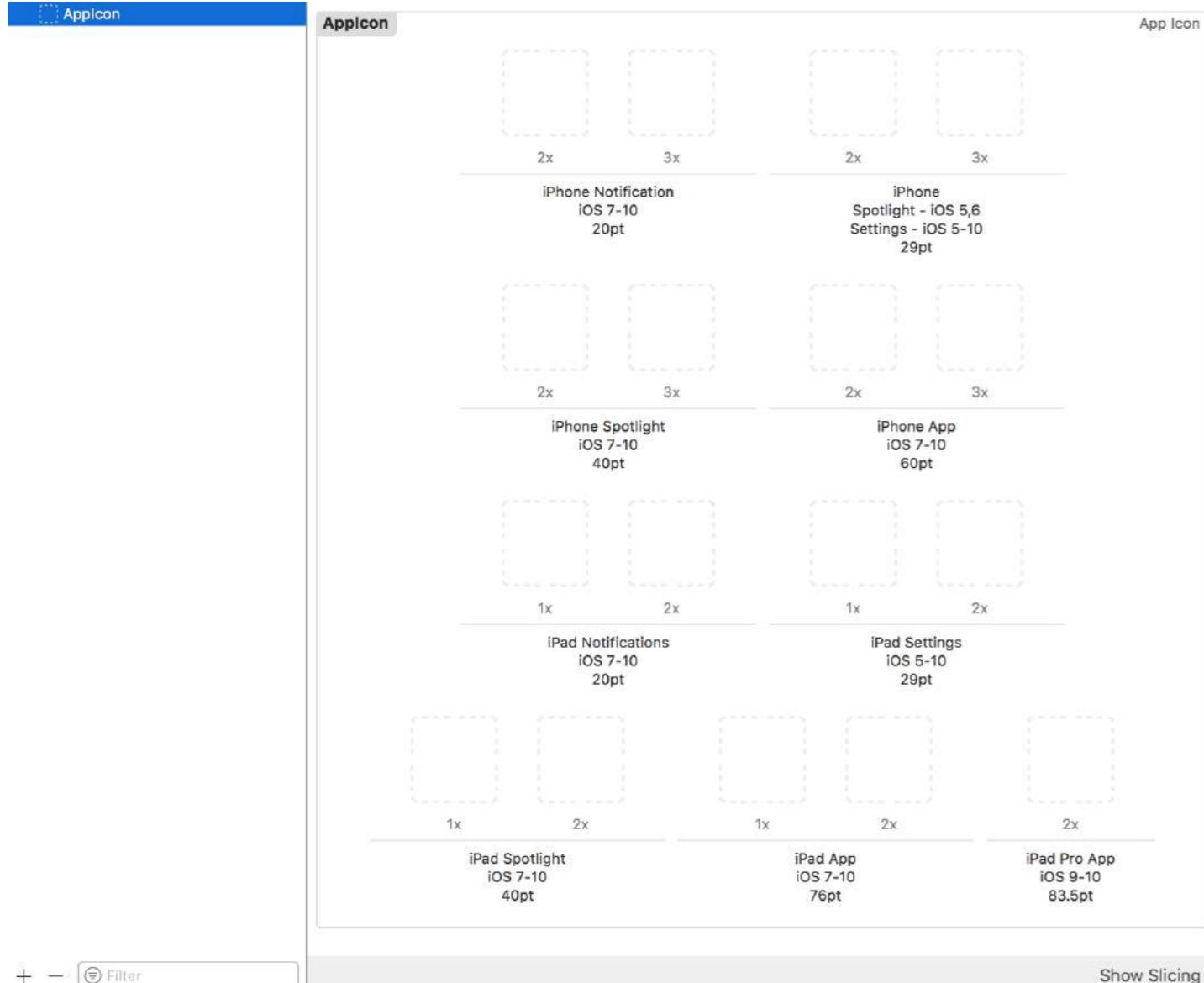
Section 201.2: App Icon using Image Assets

Whenever we create a new project in Xcode for our new app, it gives us various in-built classes, targets, tests, plist file, etc. Similarly it also gives us an Assets.xcassets file, which manages all the image assets in our project.

This is how this file looks like in file navigator:



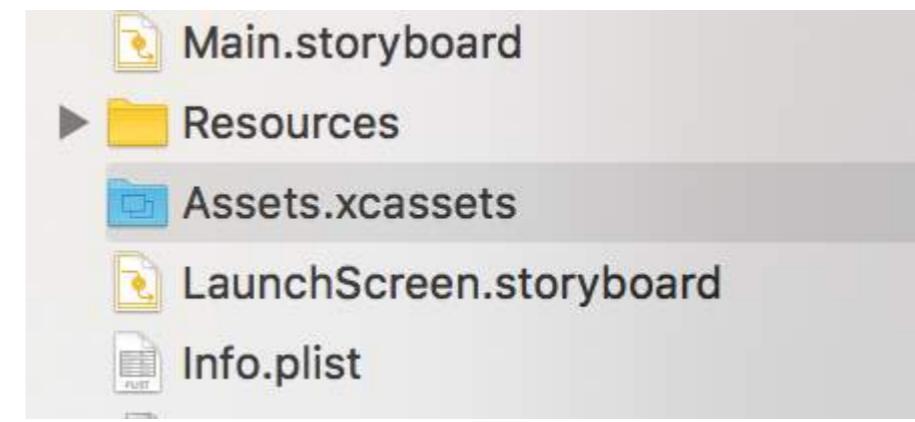
如果我们点击它，它会是这样的：



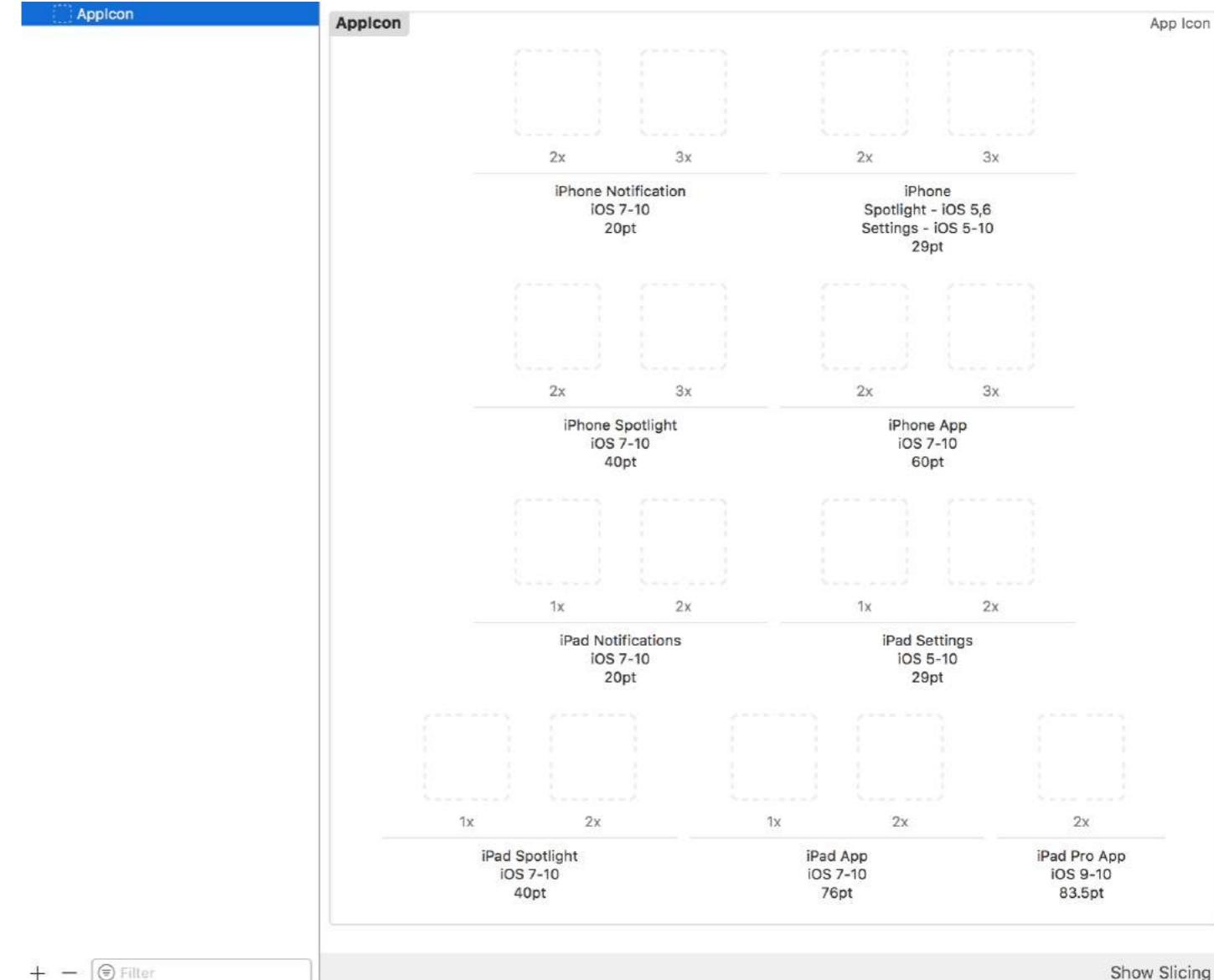
正如我所说，AppIcon 资源已经为我们创建好了。

我们只需将相应的图片拖放到每个空白的方块中。每个黑色方块会告诉我们图片的尺寸，尺寸写在方块正下方。

将所有图片拖放到所有方块后，它会是这样的：



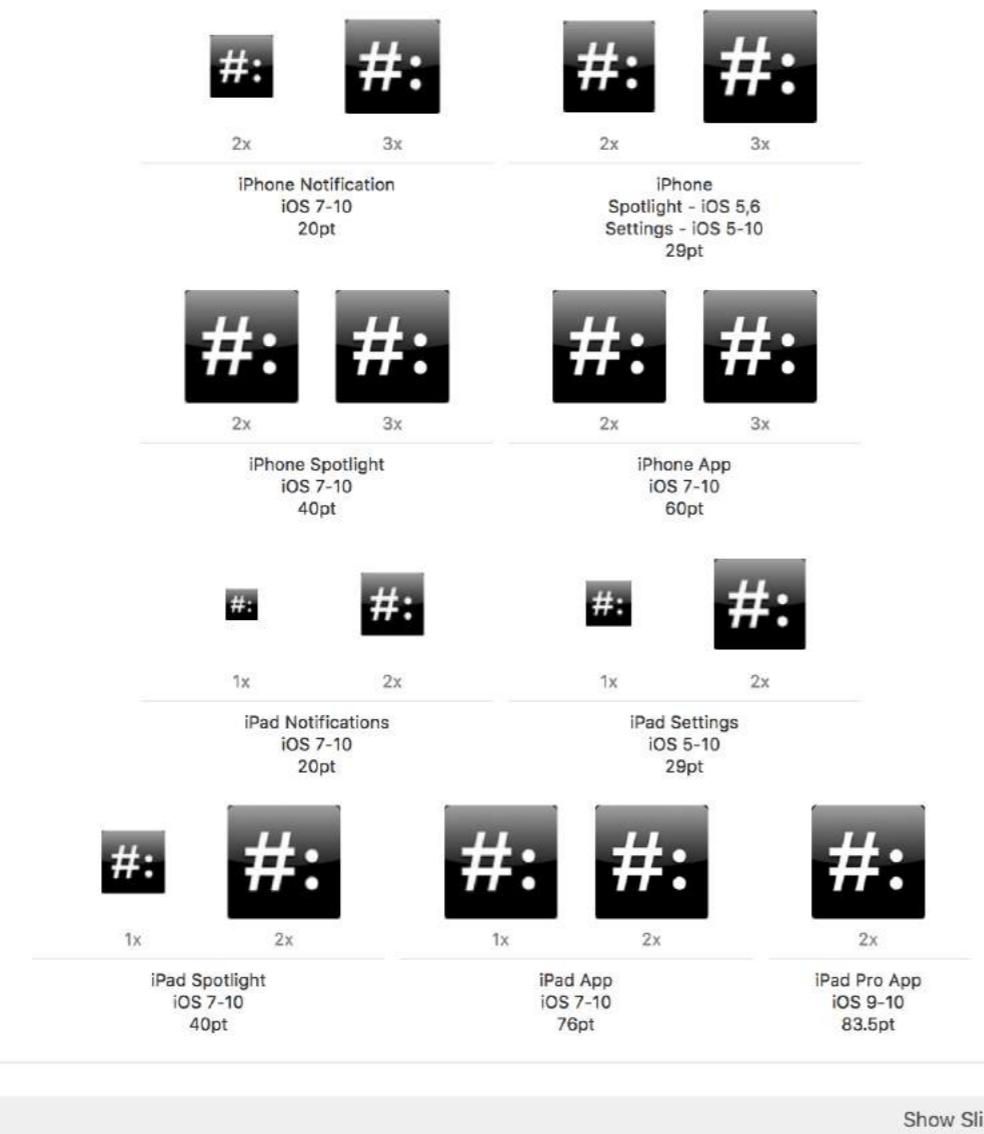
If we click it, it'll look like this:



As I said, AppIcon asset is already created for us.

We just have to **drag and drop** respective image on each empty square block. Each black will tell us what size that image should be, it's written just below it.

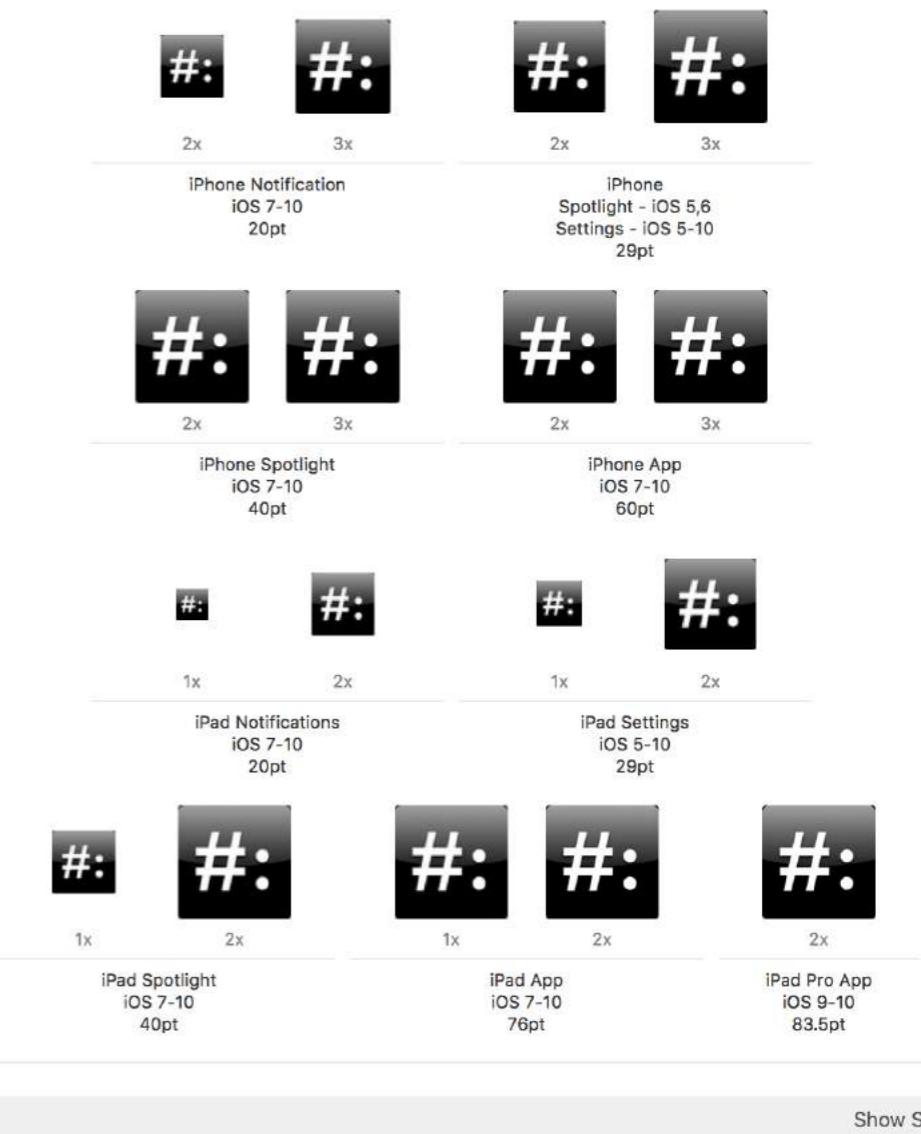
After dragging and dropping all the images in all the squares, it'll look like this:



+ - Filter

Show Slicing

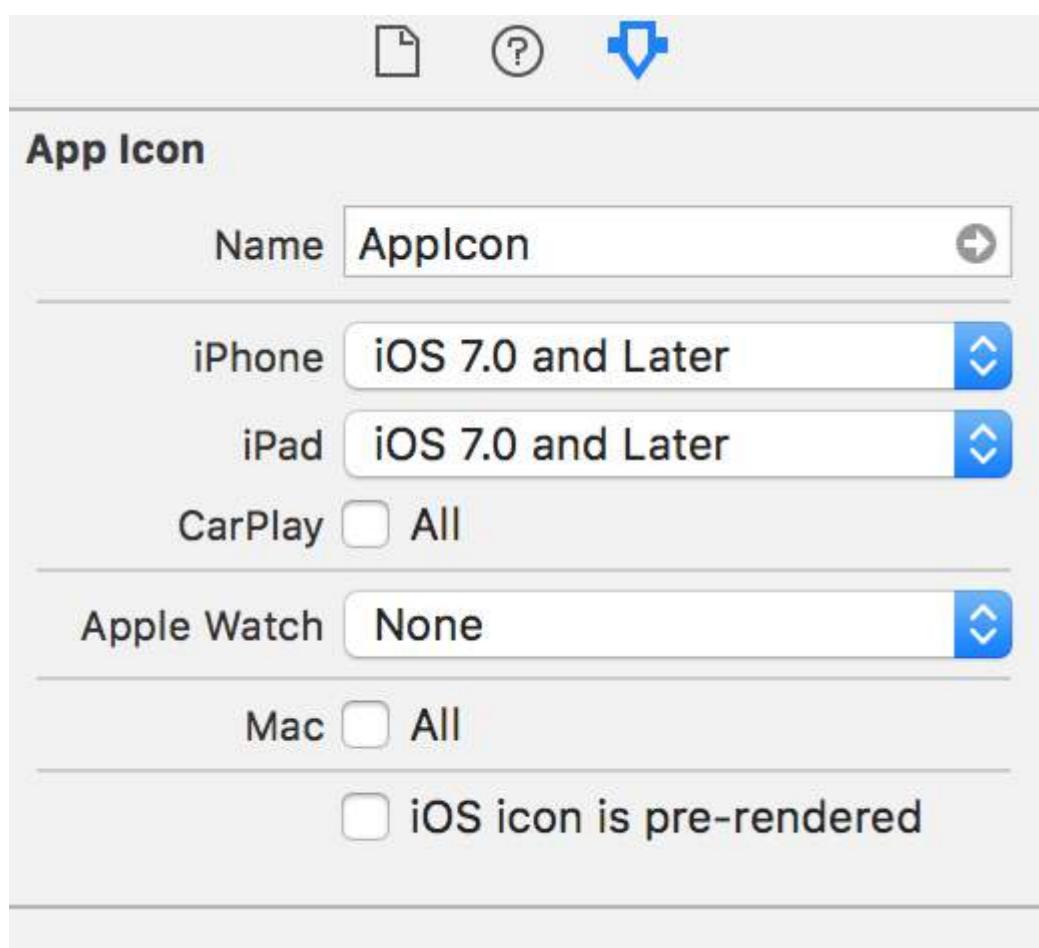
我们也可以在实用工具 -> 属性检查器中更改设备设置，针对图标资源，设置如下：



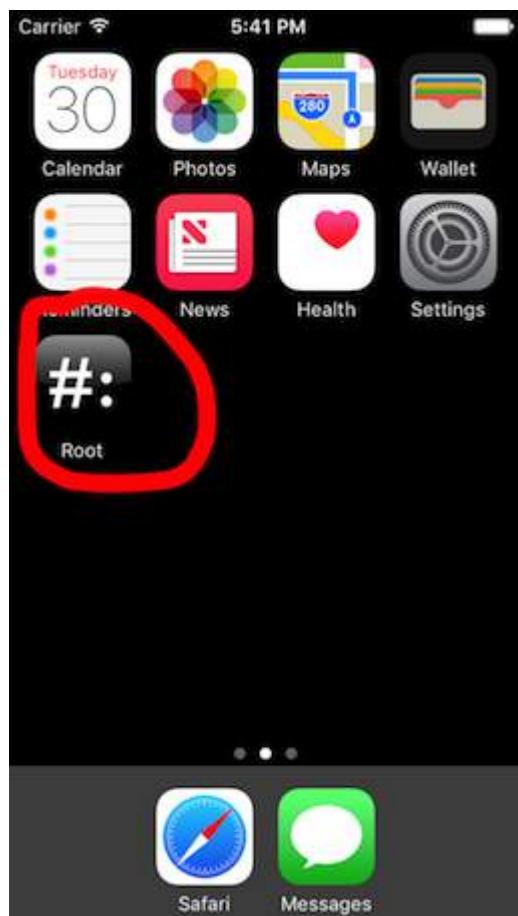
+ - Filter

Show Slicing

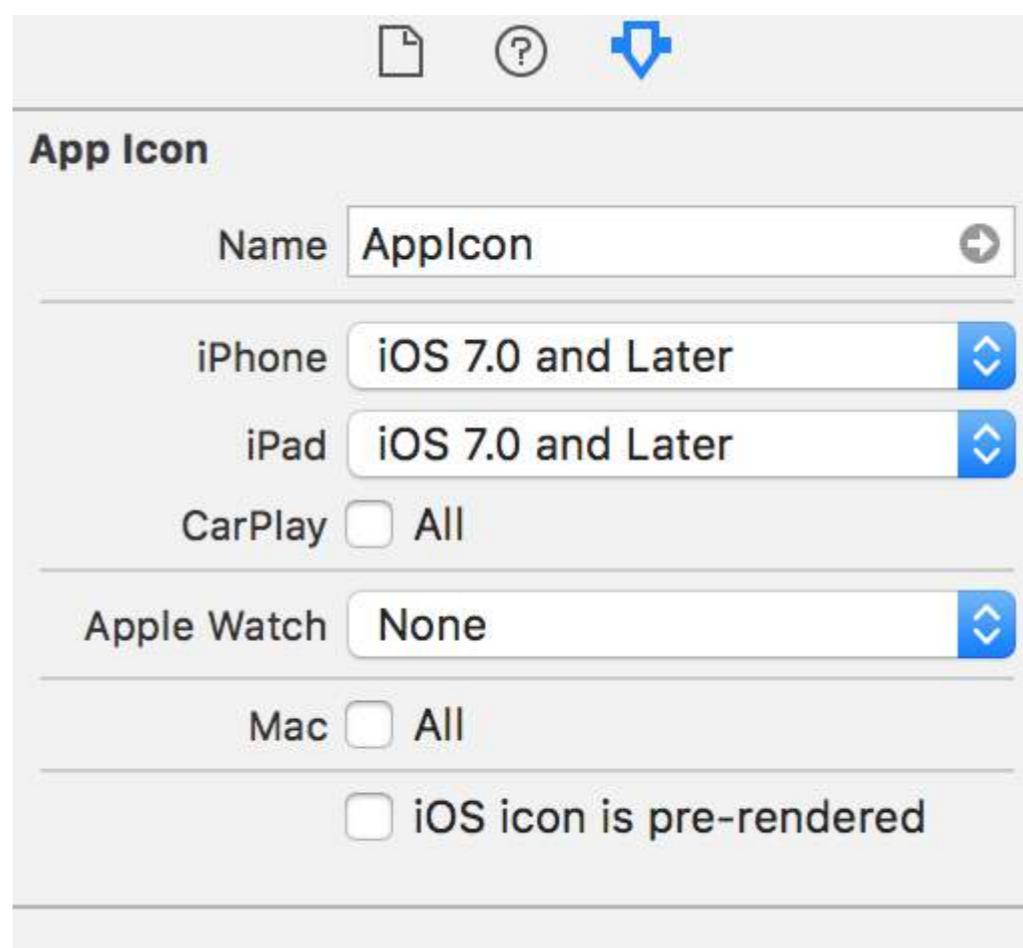
We can change the devices setting also for icon assets in Utilities -> Attributes Inspector as:



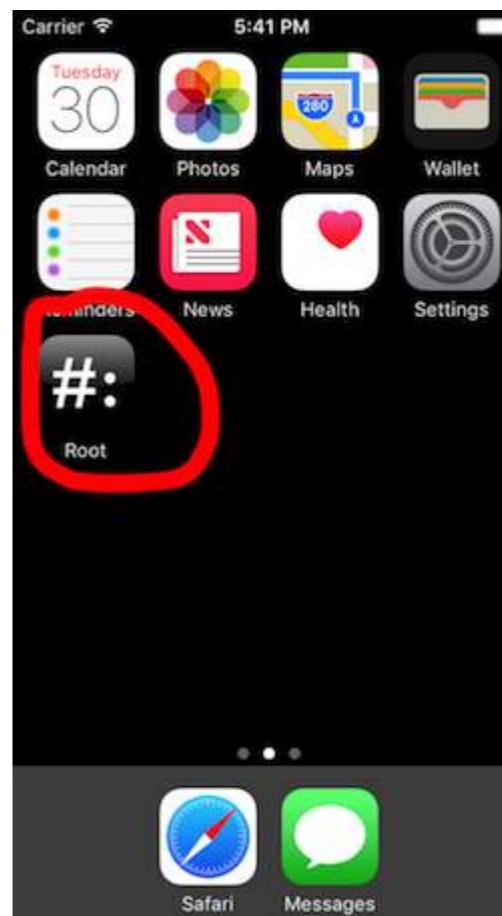
完成这些后，只需运行应用程序，我们就会有一个漂亮的应用图标，如下所示：



它默认存在，但如果没有，请确保在目标 -> 常规设置中此设置如下：



Once we finished this, just run an app and we'll be having nice icon to app as this:



It is there by default, but if it's not then make sure this settings is as in Target->General settings:

▼ App Icons and Launch Images

App Icons Source  

▼ App Icons and Launch Images

App Icons Source  

第202章：Objective-C中的运行时

第202.1节：使用关联对象

当你想为现有类添加需要保存状态的功能时，关联对象非常有用。

例如，给每个UIView添加一个活动指示器：

Objective-C实现

```
#import <objc/runtime.h>

static char ActivityIndicatorKey;

@implementation UIView (ActivityIndicator)

- (UIActivityIndicatorView *)activityIndicator {
    return (UIActivityIndicatorView *)objc_getAssociatedObject(self, &ActivityIndicatorKey);
}

- (void)setActivityIndicator: (UIActivityIndicatorView *)activityIndicator {
    objc_setAssociatedObject(self, &ActivityIndicatorKey, activityIndicator,
    OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (void)showActivityIndicator {
    UIActivityIndicatorView *activityIndicator = [[UIActivityIndicatorView alloc]
initWithActivityIndicatorStyle: UIActivityIndicatorViewStyleGray];

    [self setActivityIndicator:activityIndicator];

    activityIndicator.center = self.center;
    activityIndicator.autoresizingMask = UIViewAutoresizingFlexibleTopMargin |
    UIViewAutoresizingFlexibleLeftMargin | UIViewAutoresizingFlexibleRightMargin |
    UIViewAutoresizingFlexibleBottomMargin;

    [activityIndicator startAnimating];

    [self addSubview: activityIndicator];
}

- (void)hideActivityIndicator {
    UIActivityIndicatorView *activityIndicator = [self activityIndicator];

    if (activityIndicator != nil) {
        [[self activityIndicator] removeFromSuperview];
    }
}

@end
```

你也可以通过 Swift 访问 Objective-C 运行时：

Swift 代码

```
extension UIView {
    private struct AssociatedKeys {
        static var activityIndicator = "UIView.ActivityIndicatorView"
    }
}
```

Chapter 202: Runtime in Objective-C

Section 202.1: Using Associated Objects

Associated objects are useful when you want to add functionality to existing classes which requires holding state.

For example, adding a activity indicator to every UIView:

Objective-C Implementation

```
#import <objc/runtime.h>

static char ActivityIndicatorKey;

@implementation UIView (ActivityIndicator)

- (UIActivityIndicatorView *)activityIndicator {
    return (UIActivityIndicatorView *)objc_getAssociatedObject(self, &ActivityIndicatorKey);
}

- (void)setActivityIndicator: (UIActivityIndicatorView *)activityIndicator {
    objc_setAssociatedObject(self, &ActivityIndicatorKey, activityIndicator,
    OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (void)showActivityIndicator {
    UIActivityIndicatorView *activityIndicator = [[UIActivityIndicatorView alloc]
initWithActivityIndicatorStyle: UIActivityIndicatorViewStyleGray];

    [self setActivityIndicator:activityIndicator];

    activityIndicator.center = self.center;
    activityIndicator.autoresizingMask = UIViewAutoresizingFlexibleTopMargin |
    UIViewAutoresizingFlexibleLeftMargin | UIViewAutoresizingFlexibleRightMargin |
    UIViewAutoresizingFlexibleBottomMargin;

    [activityIndicator startAnimating];

    [self addSubview: activityIndicator];
}

- (void)hideActivityIndicator {
    UIActivityIndicatorView *activityIndicator = [self activityIndicator];

    if (activityIndicator != nil) {
        [[self activityIndicator] removeFromSuperview];
    }
}

@end
```

You can also access the Objective-C runtime through Swift:

Swift Code

```
extension UIView {
    private struct AssociatedKeys {
        static var activityIndicator = "UIView.ActivityIndicatorView"
    }
}
```

```

private var activityIndicatorView: UIActivityIndicatorView? {
    get {
        return objc_getAssociatedObject(self, &AssociatedKeys.activityIndicator) as?
UIActivityIndicatorView
    }
    set (activityIndicatorView) {
objc_setAssociatedObject(self, &AssociatedKeys.activityIndicator,
activityIndicatorView, .OBJC_ASSOCIATION_RETAIN_NONATOMIC)
    }
}

func showActivityIndicator() {
activityIndicatorView = UIActivityIndicatorView(activityIndicatorStyle: .gray)
    activityIndicatorView.center = center
activityIndicatorView.autoresizingMask = [.flexibleLeftMargin, .flexibleRightMargin,
.flexibleTopMargin, .flexibleBottomMargin]

activityIndicatorView.startAnimating()

    addSubview(activityIndicatorView)
}

func hideActivityIndicator() {
    activityIndicatorView.removeFromSuperview()
}
}

```

```

private var activityIndicatorView: UIActivityIndicatorView? {
    get {
        return objc_getAssociatedObject(self, &AssociatedKeys.activityIndicator) as?
UIActivityIndicatorView
    }
    set (activityIndicatorView) {
objc_setAssociatedObject(self, &AssociatedKeys.activityIndicator,
activityIndicatorView, .OBJC_ASSOCIATION_RETAIN_NONATOMIC)
    }
}

func showActivityIndicator() {
activityIndicatorView = UIActivityIndicatorView(activityIndicatorStyle: .gray)
    activityIndicatorView.center = center
activityIndicatorView.autoresizingMask = [.flexibleLeftMargin, .flexibleRightMargin,
.flexibleTopMargin, .flexibleBottomMargin]

    activityIndicatorView.startAnimating()

    addSubview(activityIndicatorView)
}

func hideActivityIndicator() {
    activityIndicatorView.removeFromSuperview()
}
}

```

第203章：模型展示样式

模态展示样式用于从一个视图控制器过渡到另一个视图控制器。有两种实现此自定义的方法。一种是通过代码，另一种是通过界面构建器（使用segue）。

此效果是通过将modalPresentationStyle变量设置为UIModalPresentationStyle枚举的一个实例来实现的。modalPresentationStyle属性是UIViewController的一个类变量，用于指定视图控制器在屏幕上的展示方式。

第203.1节：使用界面构建器探索ModalPresentationStyle

这是一个非常基础的应用程序，用于演示iOS中不同的ModalPresentationStyle。根据此处找到的文档，UIModalPresentationStyle有9种不同的取值，具体如下，

1. 全屏 (fullScreen)
2. 页面表 (pageSheet)
3. 表单表 (formSheet)
4. 当前上下文 (currentContext)
5. 自定义 (custom)
6. 覆盖全屏 (overFullScreen)
7. 覆盖当前上下文 (overCurrentContext)
8. 弹出框 (popover)
9. 无

要设置一个项目，只需创建一个普通的 iOS 项目并添加两个ViewControllers。在你的初始 ViewController中放置一个UIButton，并通过Target -> Action机制将其连接到第二个ViewController。为了区分两个 ViewControllers，设置ViewController中UIView的背景属性为其他颜色。如果一切顺利，你的 Interface Builder 应该看起来像这样，

Chapter 203: ModelPresentationStyles

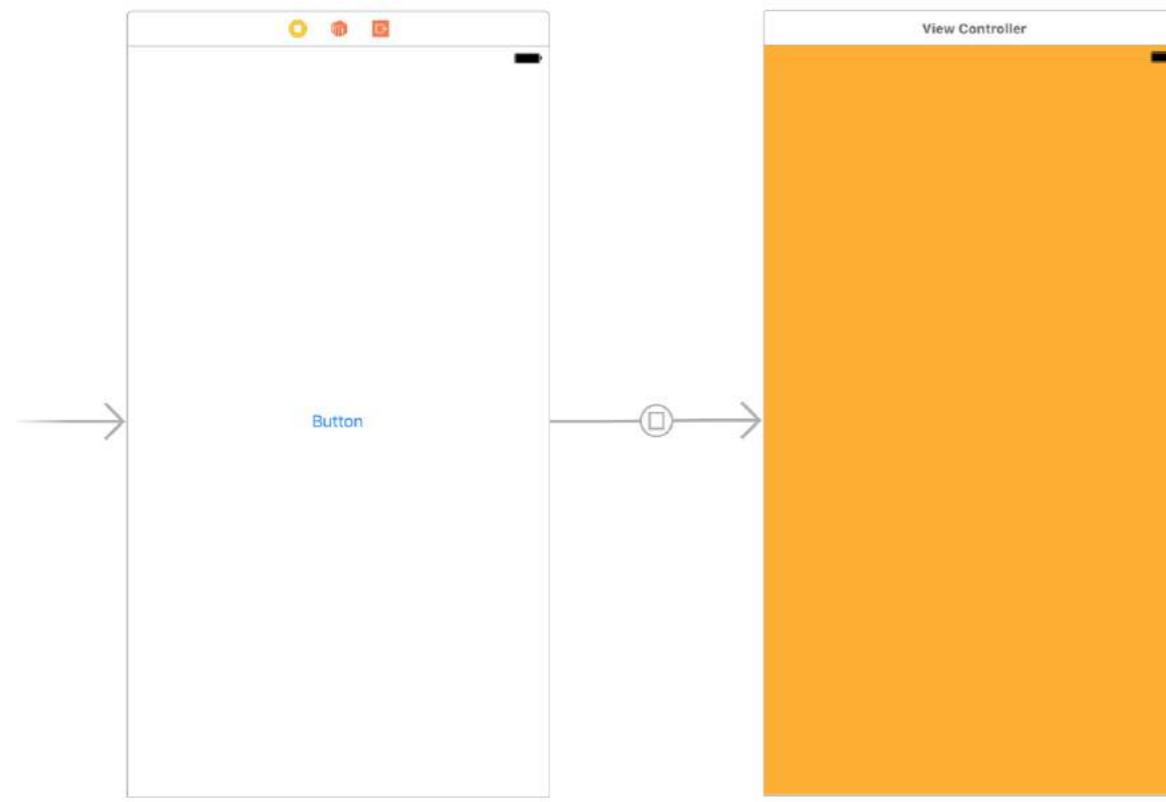
Modal Presentation styles are used when you are transitioning from one view controller to another. There are 2 ways of achieving this customization. One is through code and another through Interface Builder(using segues). This effect is achieved by setting modalPresentationStyle variable to an instance of UIModalPresentationStyle enum. modalPresentationStyle property is a class variable of UIViewController and is used to specify how a ViewController is presented on screen.

Section 203.1: Exploring ModalPresentationStyle using Interface Builder

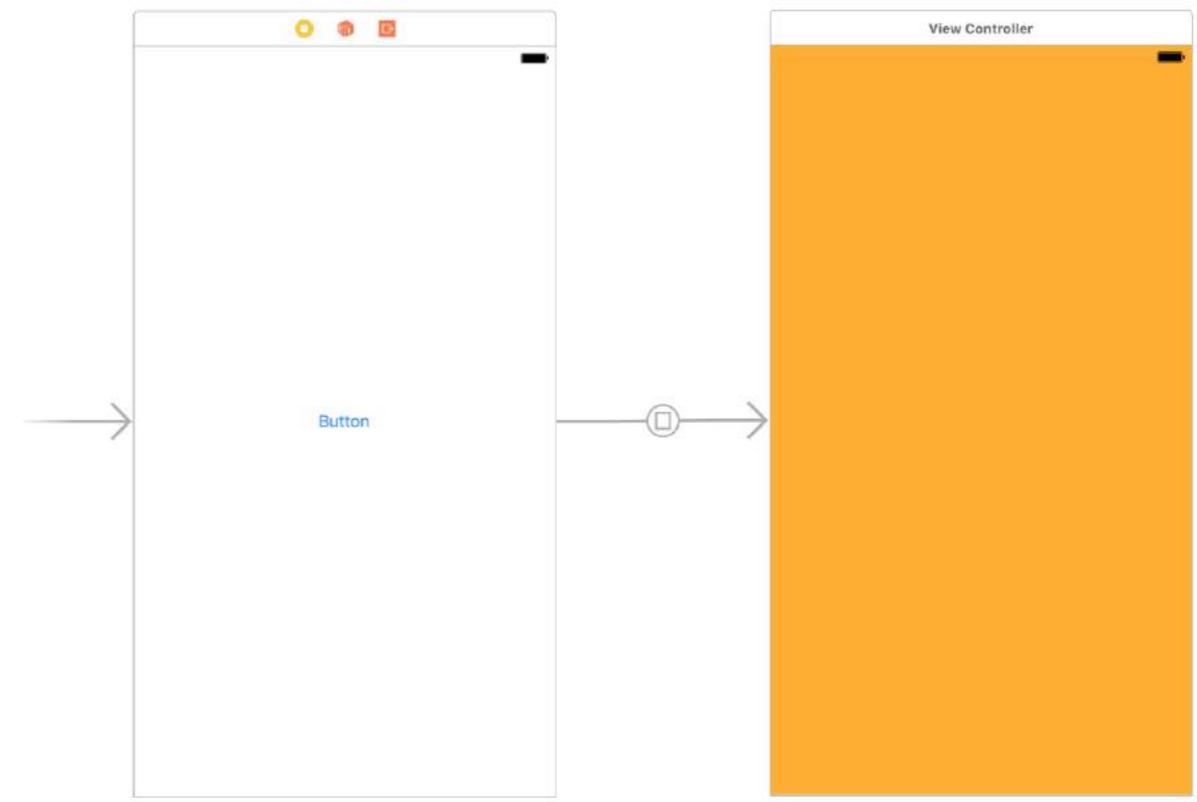
This will be a very basic app which will illustrate different ModalpresentationStyle in iOS. According to documentation found [here](#), There are 9 different values for UIModalPresentationStyle which are as follows,

1. fullScreen
2. pageSheet
3. formSheet
4. currentContext
5. custom
6. overFullScreen
7. overCurrentContext
8. popover
9. none

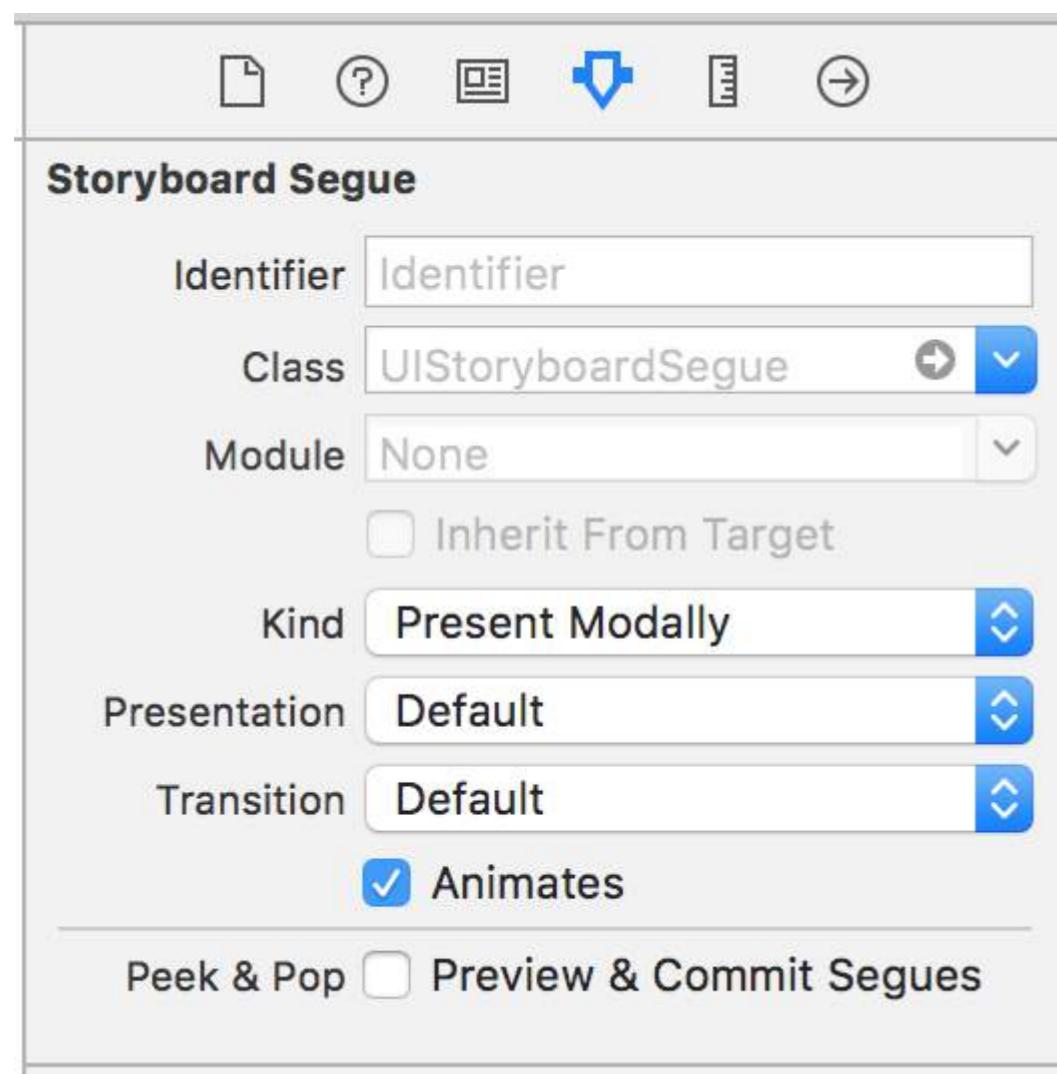
To setup a project, just create a normal iOS project and add 2 ViewControllers. Put a UIButton in your initial ViewController and connect it to 2nd ViewController via a Target -> Action mechanism. To distinguish both ViewControllers, set background property of UIView in ViewController some other color. If all goes well, your Interface Builder should look something like this,



确保你构建此项目并在iPad上运行（关于为什么选择 iPad 的详细信息，请参阅备注部分）。设置好项目后，选择 segue 并进入attributes inspector。你应该能看到



Make sure you build this project and run it on **iPad** (For details on why iPad, refer to Remarks section). Once you are done setting up your project, select the segue and go to the attributes inspector. You should be able to see

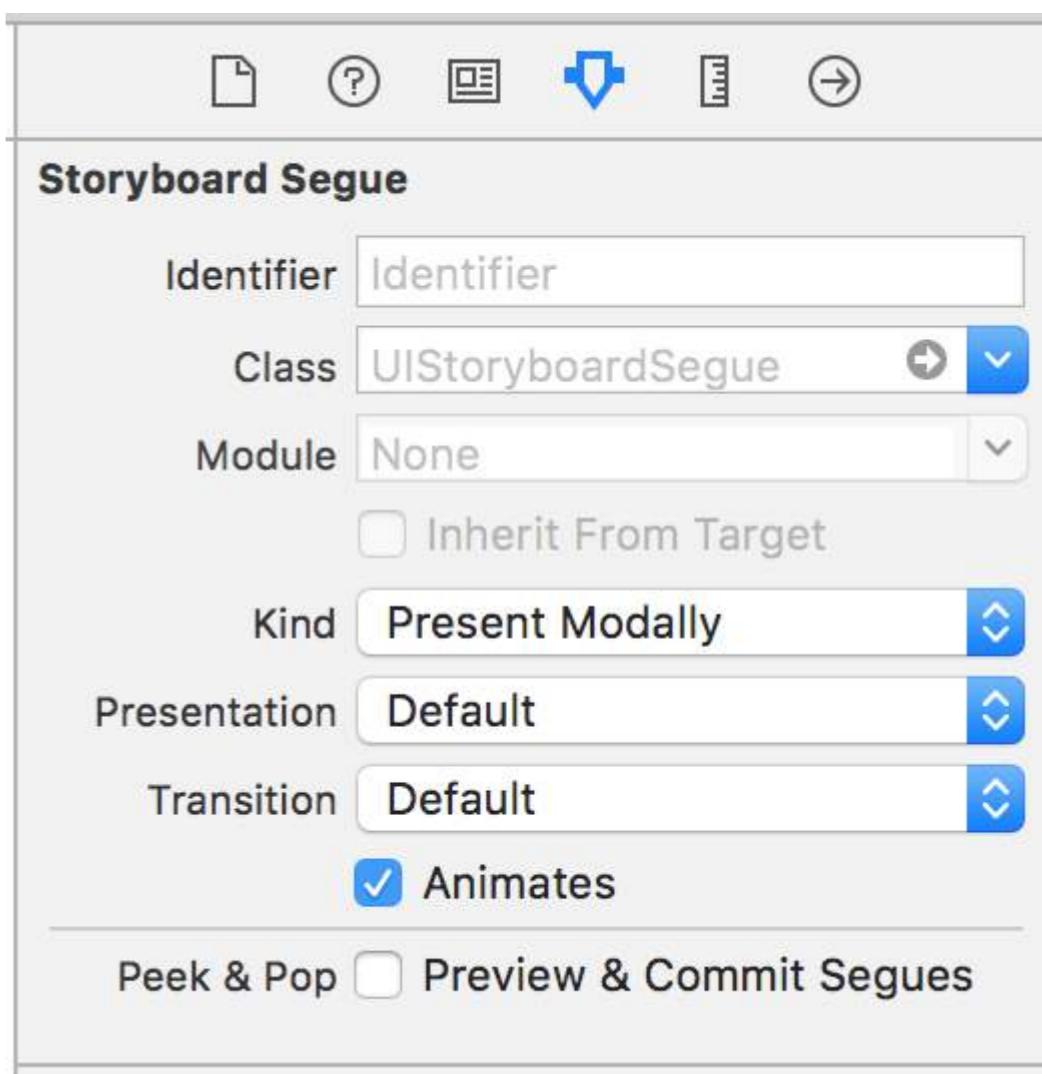


类似这样的内容，

将 kind 属性设置为 Present Modally。

现在，在这个示例中我们看不到所有效果，因为其中一些需要少量代码。

让我们从fullscreen开始。当你在Kind标签中选择Present Modally时，这个效果默认被选中。构建并运行后，第二个ViewController将占据你 iPad 的全屏。



something like this,

Set the kind property to Present Modally.

Now, we won't see all of the effects in this example as some of them requires little bit of code.

Let's start with fullscreen. This effect is selected by default when you select Present Modally in Kind tab. When you build and run, the 2nd ViewController would occupy the full screen of your iPad.

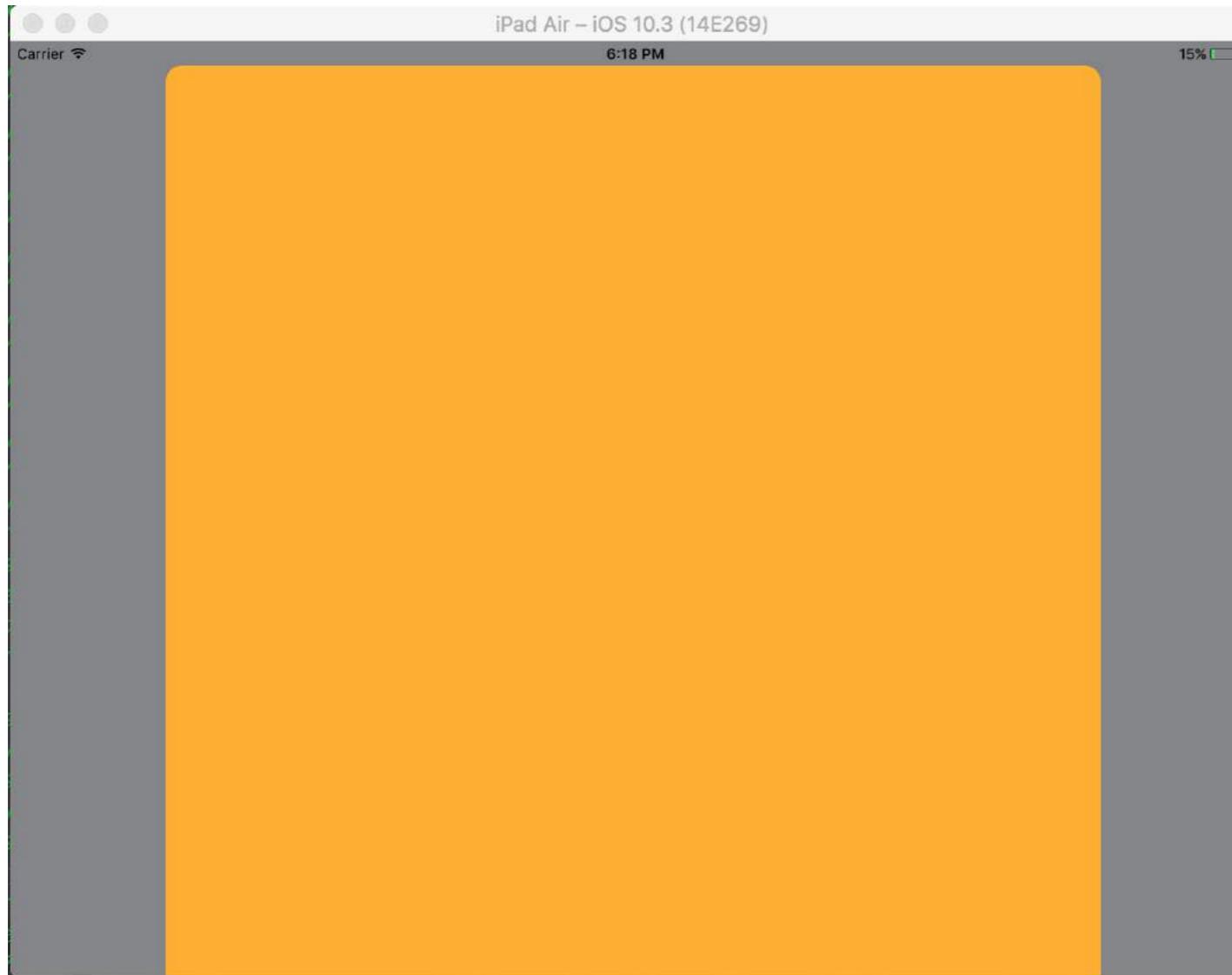


接下来是pageSheet。您可以从Presentation选项卡中选择此选项。在此选项中，当设备处于纵向时



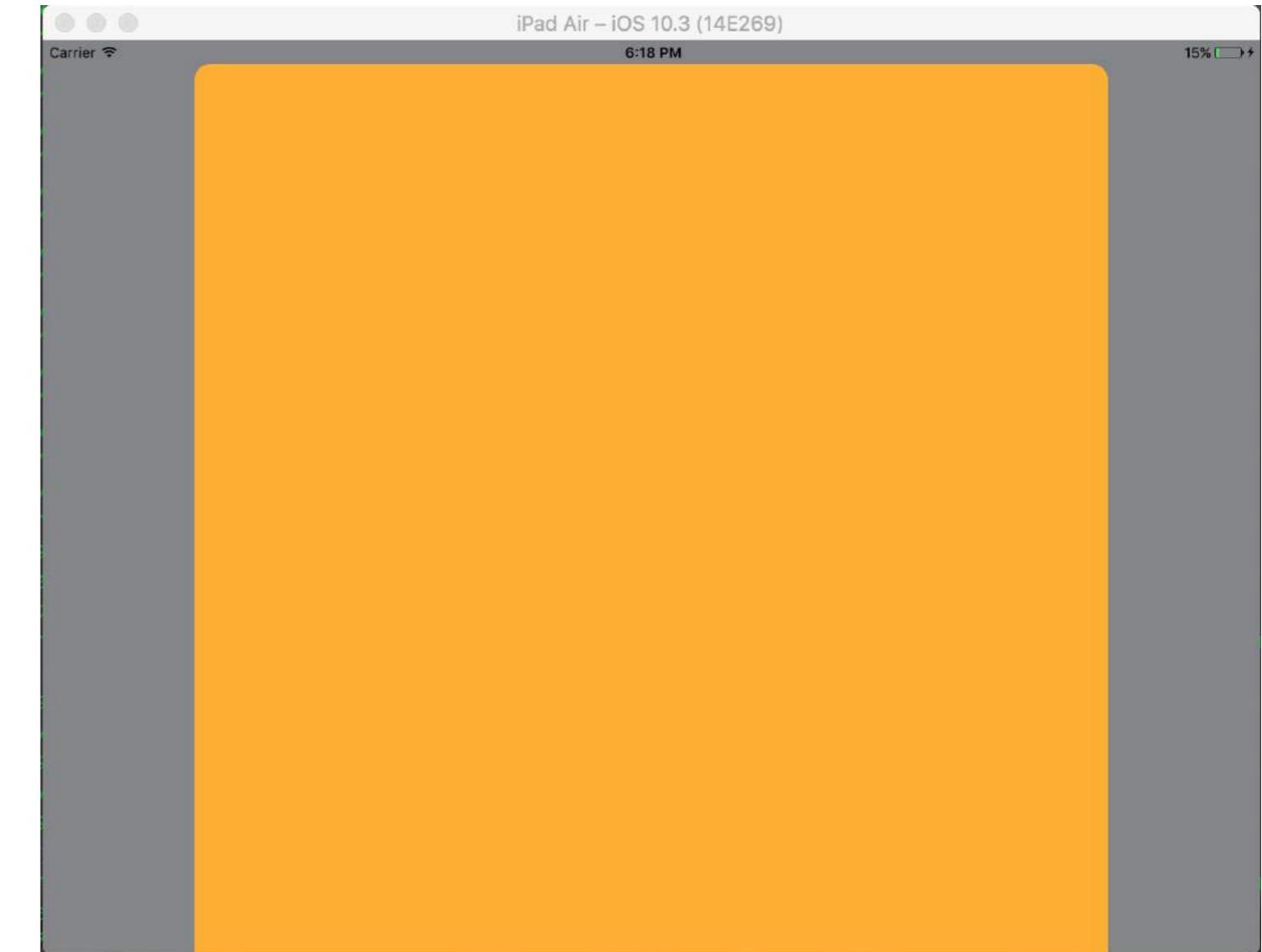
Next is pageSheet. You can select this option from Presentation tab. In this option, when device is in portrait

模式下，第二个ViewController类似于全屏，但为横屏模式，第二个ViewController的宽度远小于设备宽度。未被第二个ViewController覆盖的内容将被遮暗。



对于formSheet样式，第二个ViewController位于设备中心，大小小于设备尺寸。
当设备处于横屏模式且键盘可见时，视图位置会向上调整以显示
ViewController。

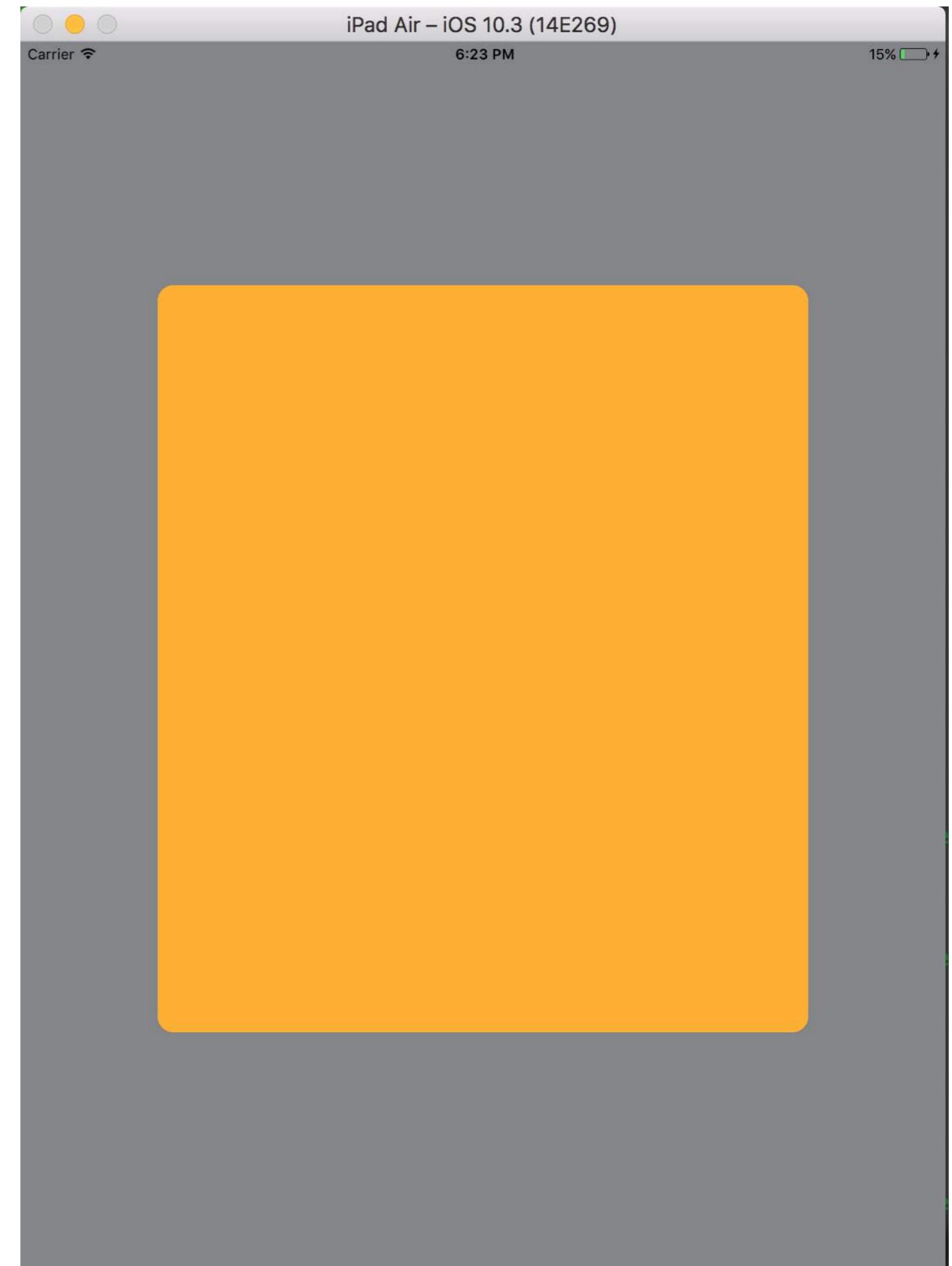
mode, the 2nd ViewController is similar to full screen but in landscape mode, 2nd ViewController is much narrower than the device width. Also, any content not covered by 2nd ViewController will be dimmed.



For formSheet style, the 2nd ViewController is placed in center of device and the size is smaller to that of device.
Also when device is in landscape mode and keyboard is visible position of view is adjusted upwards to show the
ViewController.

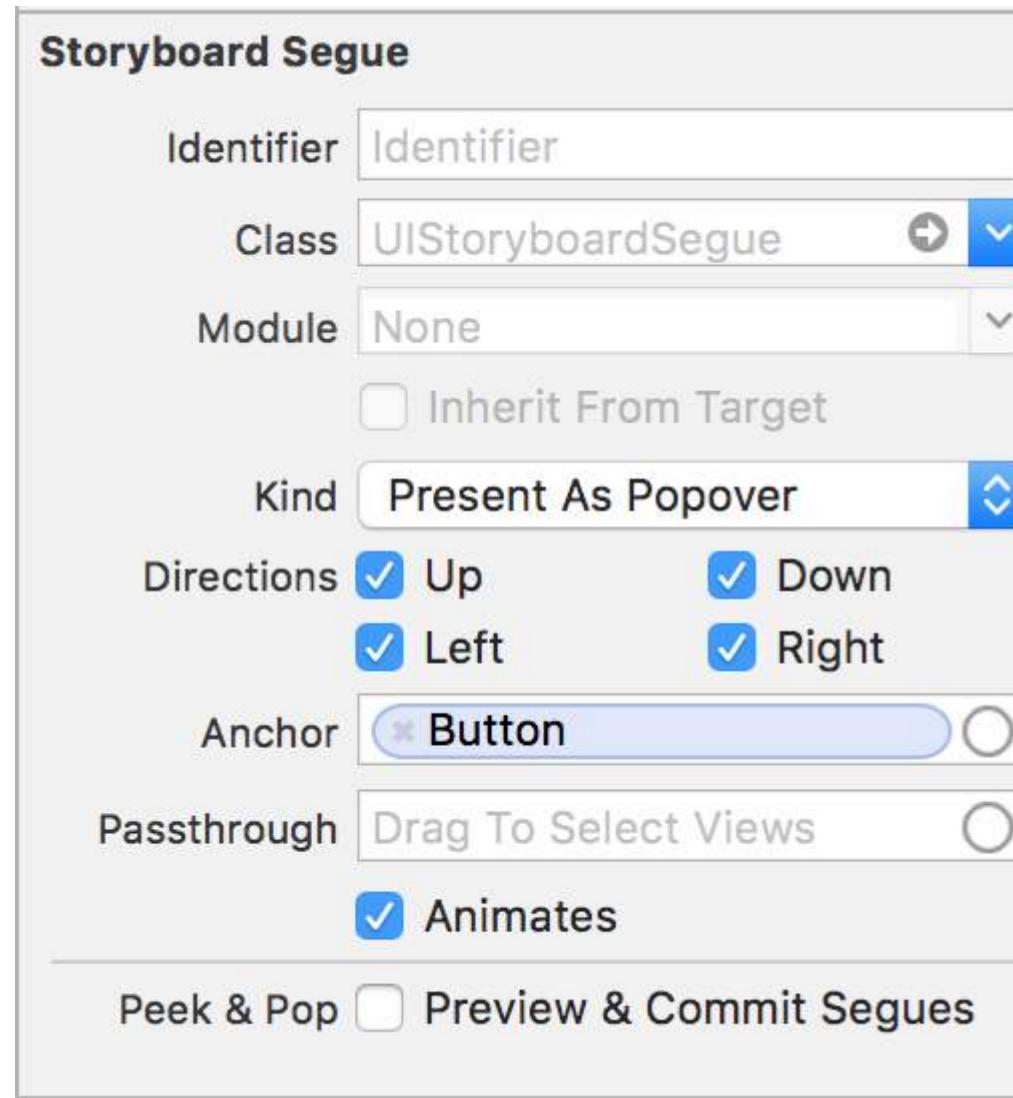


我们将尝试的最后一种样式是popover。要选择此样式，请在Kind标签中选择Present as Popover。第二个



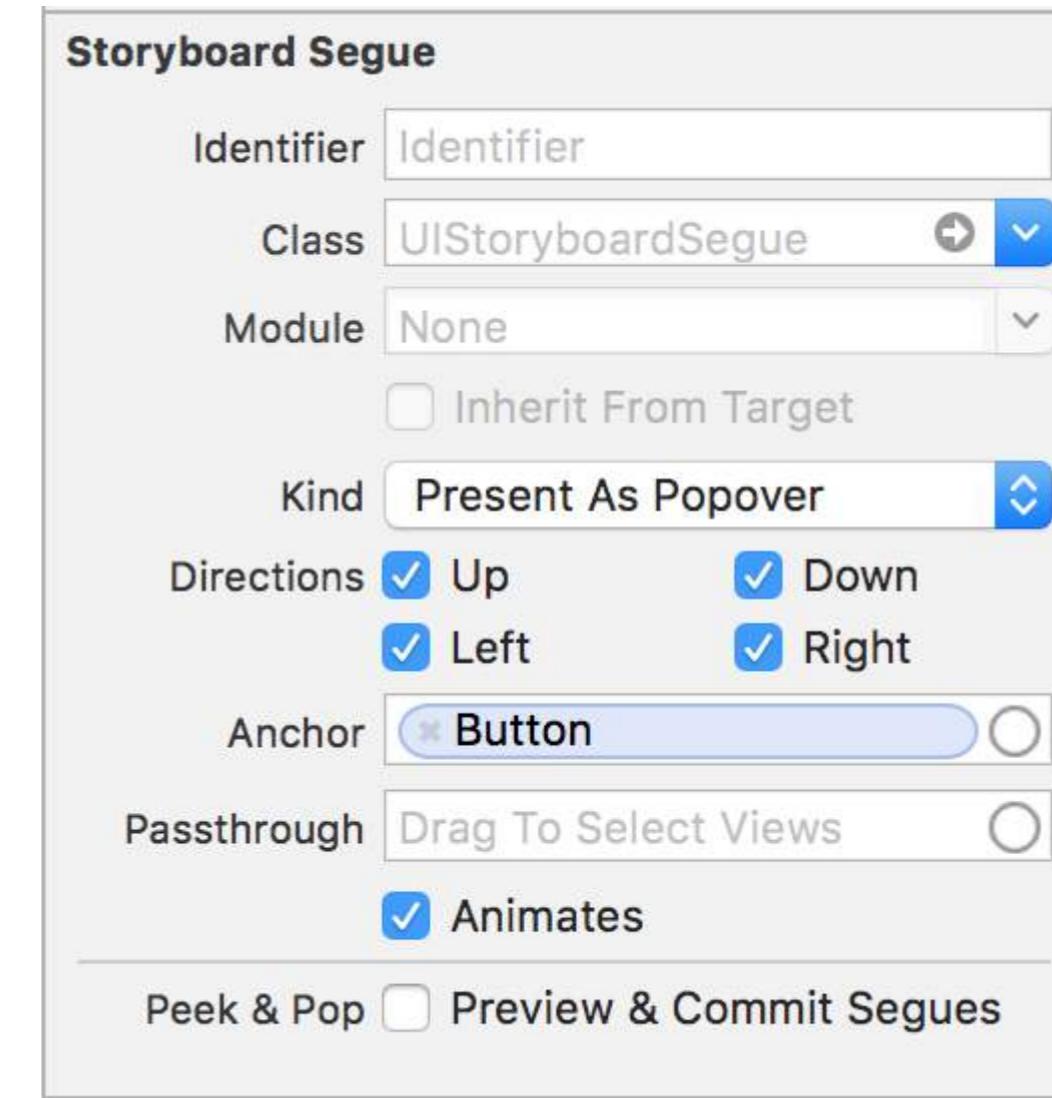
Last style which we are going to try is popover. To select this style, select Present as Popover in Kind tab. The 2nd

ViewController以小型弹出视图形式呈现（大小可设置）。背景内容会被遮暗。点击弹出视图外部任意位置将关闭弹出视图。你的Attributes Inspector应类似如下，

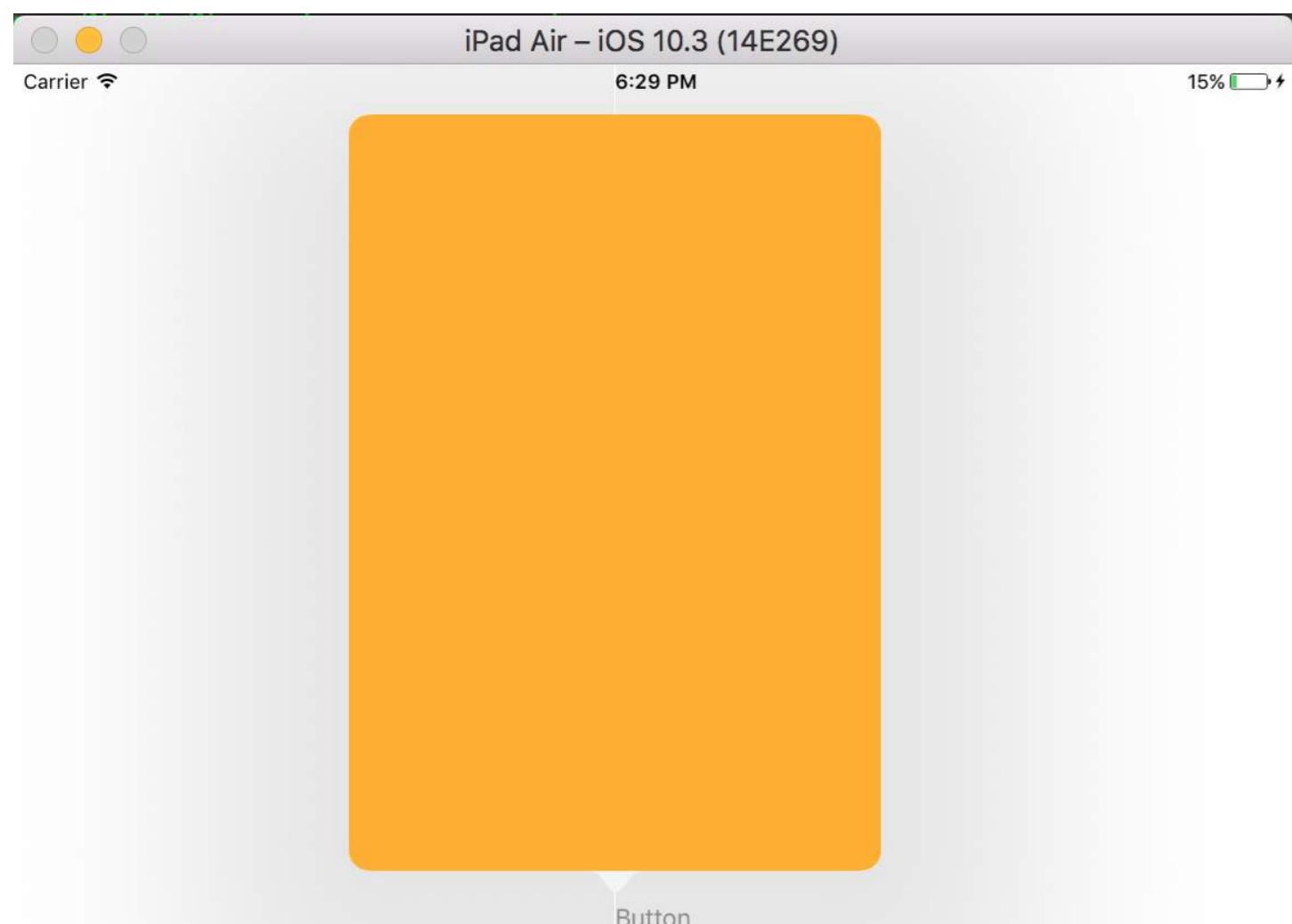


Anchor是你希望弹出视图箭头指向的UI元素。Directions是允许弹出视图Anchor指向的方向。

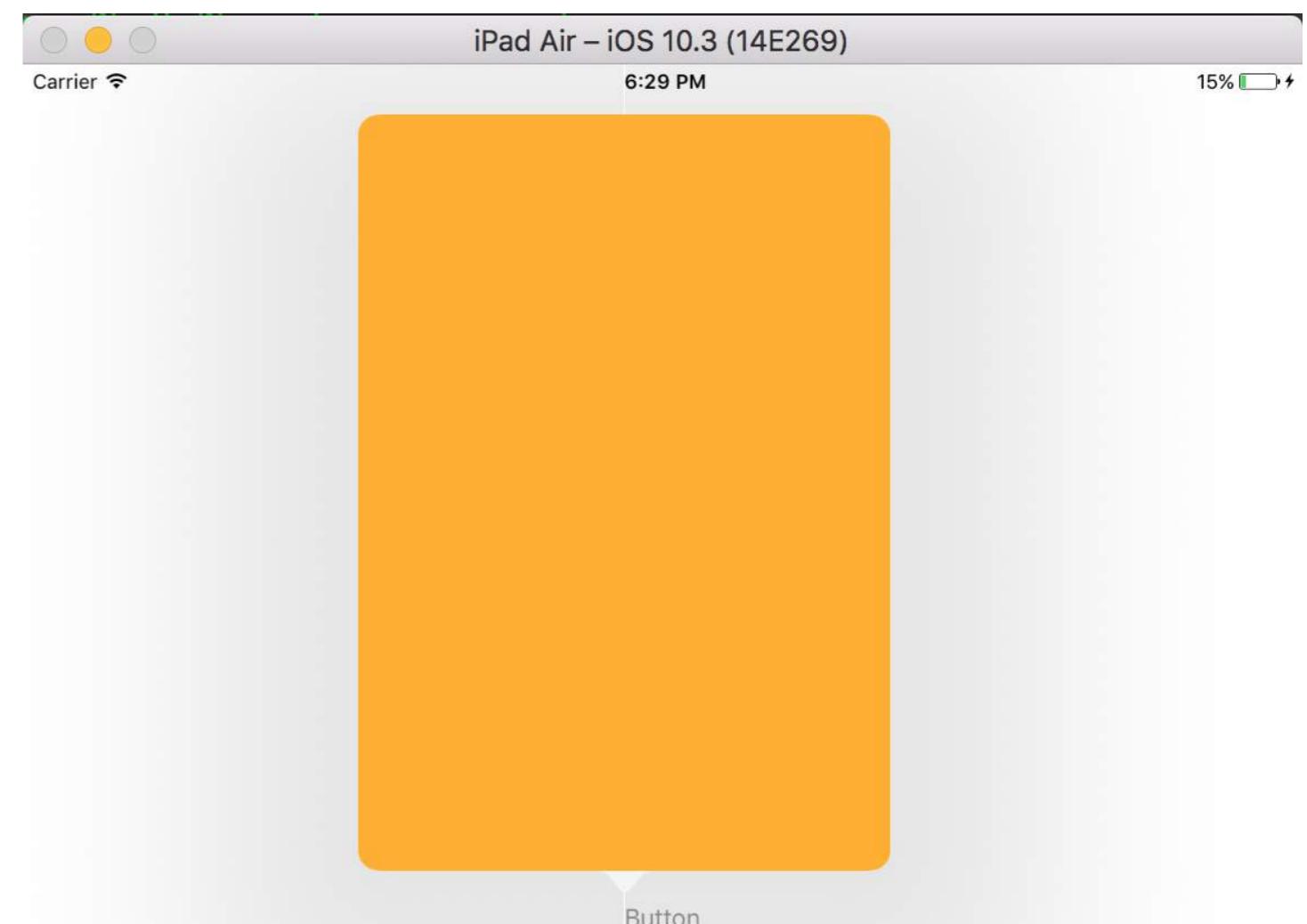
ViewController is presented as a small popover(size can be set). The background content is dimmed. Any tap outside the popover would dismiss the popover. Your Attributes Inspector should look something like this,



Anchor is the UI element to which you want your popover arrow to point. Directions are the directions you allow your popover Anchor to point in.



除了这些基本的模态展示样式外，还有更多样式，但它们实现起来比较复杂，需要一些代码。



There more then these basic Modal Presentation Styles but they are little complicated to achieve and require some

更多细节可以在苹果文档中找到。

code. More details can be found in the Apple Documentation.

第204章：CydiaSubstrate插件

学习如何为越狱的iPhone创建CydiaSubstrate插件。

这些插件将使你能够修改操作系统的 behavior，使其按照你希望的方式运行。

第204.1节：使用Theos创建新插件

使用nic创建一个新项目

在终端中输入此命令

```
$THEOS/bin/nic.pl
```

NIC 2.0 - 新实例创建器

```
[1.] iphone/activator_event  
[2.] iphone/application_modern  
[3.] iphone/cyget  
[4.] iphone/flipswitch_switch  
[5.] iphone/framework  
[6.] iphone/ios7_notification_center_widget  
[7.] iphone/library  
[8.] iphone/notification_center_widget  
[9.] iphone/preference_bundle_modern  
[10.] iphone/tool  
[11.] iphone/tweak  
[12.] iphone/xpc_service
```

请选择一个模板（必填）：

选择模板 [11.] iphone/tweak

填写详细信息后，将创建以下文件：

```
-rw-r--r--@ 1 gkpln3 staff 214B 6月 12 15:09 Makefile  
-rw-r--r--@ 1 gkpln3 staff 89B 6月 11 22:58 TorchonFocus.plist  
-rw-r--r-- 1 gkpln3 staff 2.7K 6月 12 16:10 Tweak.xm  
-rw-r--r-- 1 gkpln3 staff 224B 6月 11 16:17 control  
drwxr-xr-x 3 gkpln3 staff 102B 6月 11 16:18 obj  
drwxr-xr-x 16 gkpln3 staff 544B 6月 12 16:12 packages
```

重写 iOS 截图保存方法

使用你喜欢的代码编辑器打开Tweak.xm文件。

钩住操作系统中的某个方法。

```
%hook SBScreenShotter  
- (void)saveScreenshot:(BOOL)screenshot  
{  
    %orig;  
    NSLog(@"saveScreenshot: 被调用");  
}  
%end
```

注意你可以选择是否调用原始函数，例如：

Chapter 204: CydiaSubstrate tweak

Learn how to create cydia substrate tweaks for jailbroken iPhones.

Those tweaks will enable you to modify the operating system's behavior to act the way you would like it to.

Section 204.1: Create new tweak using Theos

Use nic to create a new project

Enter this command in your terminal

```
$THEOS/bin/nic.pl
```

NIC 2.0 - New Instance Creator

```
[1.] iphone/activator_event  
[2.] iphone/application_modern  
[3.] iphone/cyget  
[4.] iphone/flipswitch_switch  
[5.] iphone/framework  
[6.] iphone/ios7_notification_center_widget  
[7.] iphone/library  
[8.] iphone/notification_center_widget  
[9.] iphone/preference_bundle_modern  
[10.] iphone/tool  
[11.] iphone/tweak  
[12.] iphone/xpc_service
```

Choose a Template (required):

Choose template [11.] iphone/tweak

Fill in the details and you will get the following files created:

```
-rw-r--r--@ 1 gkpln3 staff 214B Jun 12 15:09 Makefile  
-rw-r--r--@ 1 gkpln3 staff 89B Jun 11 22:58 TorchonFocus.plist  
-rw-r--r-- 1 gkpln3 staff 2.7K Jun 12 16:10 Tweak.xm  
-rw-r--r-- 1 gkpln3 staff 224B Jun 11 16:17 control  
drwxr-xr-x 3 gkpln3 staff 102B Jun 11 16:18 obj  
drwxr-xr-x 16 gkpln3 staff 544B Jun 12 16:12 packages
```

Override iOS screenshots saving method

open the Tweak.xm file using your favorite code editor.

hook to a certain method from the operating system.

```
%hook SBScreenShotter  
- (void)saveScreenshot:(BOOL)screenshot  
{  
    %orig;  
    NSLog(@"saveScreenshot: is called");  
}
```

Note you can choose whether or not the original function should be called, for example:

```
%hook SBScreenShotter
- (void)saveScreenshot:(BOOL)screenshot
{
    NSLog(@"saveScreenshot: 被调用");
}
%end
```

将覆盖该函数而不调用原始函数，因此导致截图无法保存。

```
%hook SBScreenShotter
- (void)saveScreenshot:(BOOL)screenshot
{
    NSLog(@"saveScreenshot: is called");
}
%end
```

will override the function without calling the original one, thus causing screenshots not being saved.

第205章：从图像创建视频

使用AVFoundation从图像创建视频

第205.1节：从UIImage创建视频

首先你需要创建AVAssetWriter

```
NSError *error = nil;
NSURL *outputURL = <#表示你想保存视频的URL的NSURL对象#>;
AVAssetWriter *assetWriter = [AVAssetWriter assetWriterWithURL:outputURL
fileType:AVFileTypeQuickTimeMovie error:&error];
if (!assetWriter) {
    // 处理错误
}
```

AVAssetWriter至少需要一个资产写入输入。

```
NSDictionary *writerInputParams = [NSDictionary dictionaryWithObjectsAndKeys:
AVVideoCodecH264, AVVideoCodecKey,
[NSNumber numberWithInt:renderSize.width],
AVVideoWidthKey,
[NSNumber numberWithInt:renderSize.height],
AVVideoHeightKey,
AVVideoScalingModeResizeAspectFill,
AVVideoScalingModeKey,
nil];

AVAssetWriterInput *assetWriterInput = [AVAssetWriterInput
assetWriterInputWithMediaType:AVMediaTypeVideo outputSettings:writerInputParams];
if ([assetWriter canAddInput:assetWriterInput]) {
    [assetWriter addInput:assetWriterInput];
} else {
    // 显示错误信息
}
```

要向 AVAssetWriterInput 添加 CVPixelBufferRef，需要创建
AVAssetWriterInputPixelBufferAdaptor

```
NSDictionary *attributes = [NSDictionary dictionaryWithObjectsAndKeys:
    [NSNumber numberWithUnsignedInt:kCVPixelFormatType_32ARGB],
    (NSString*)kCVPixelBufferPixelFormatKey,
    [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGImageCompatibilityKey,
    [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGBitmapContextCompatibilityKey,
    nil];
AVAssetWriterInputPixelBufferAdaptor *writerAdaptor = [AVAssetWriterInputPixelBufferAdaptor
assetWriterInputPixelBufferAdaptorWithAssetWriterInput:assetWriterInput
sourcePixelBufferAttributes:attributes];
```

现在我们可以开始写入了

```
[assetWriter startWriting];
[assetWriter startSessionAtSourceTime:kCMTimeZero];
[assetWriterInput requestMediaDataWhenReadyOnQueue:exportingQueue usingBlock:^{
    for (int i = 0; i < images.count; ++i) {
        while (!assetWriterInput isReadyForMoreMediaData) {
```

Chapter 205: Create a video from images

Create a video from images using AVFoundation

Section 205.1: Create Video from UIImages

First of all you need to create AVAssetWriter

```
NSError *error = nil;
NSURL *outputURL = <#NSURL object representing the URL where you want to save the video#>;
AVAssetWriter *assetWriter = [AVAssetWriter assetWriterWithURL:outputURL
fileType:AVFileTypeQuickTimeMovie error:&error];
if (!assetWriter) {
    // handle error
}
```

AVAssetWriter needs at least one asset writer input.

```
NSDictionary *writerInputParams = [NSDictionary dictionaryWithObjectsAndKeys:
    AVVideoCodecH264, AVVideoCodecKey,
    [NSNumber numberWithInt:renderSize.width],
    AVVideoWidthKey,
    [NSNumber numberWithInt:renderSize.height],
    AVVideoHeightKey,
    AVVideoScalingModeKey,
    nil];

AVAssetWriterInput *assetWriterInput = [AVAssetWriterInput
assetWriterInputWithMediaType:AVMediaTypeVideo outputSettings:writerInputParams];
if ([assetWriter canAddInput:assetWriterInput]) {
    [assetWriter addInput:assetWriterInput];
} else {
    // show error message
}
```

To append CVPixelBufferRef's to AVAssetWriterInput we need to create
AVAssetWriterInputPixelBufferAdaptor

```
NSDictionary *attributes = [NSDictionary dictionaryWithObjectsAndKeys:
    [NSNumber numberWithUnsignedInt:kCVPixelFormatType_32ARGB],
    (NSString*)kCVPixelBufferPixelFormatKey,
    [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGImageCompatibilityKey,
    [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGBitmapContextCompatibilityKey,
    nil];
AVAssetWriterInputPixelBufferAdaptor *writerAdaptor = [AVAssetWriterInputPixelBufferAdaptor
assetWriterInputPixelBufferAdaptorWithAssetWriterInput:assetWriterInput
sourcePixelBufferAttributes:attributes];
```

Now we can start writing

```
[assetWriter startWriting];
[assetWriter startSessionAtSourceTime:kCMTimeZero];
[assetWriterInput requestMediaDataWhenReadyOnQueue:exportingQueue usingBlock:^{
    for (int i = 0; i < images.count; ++i) {
        while (!assetWriterInput isReadyForMoreMediaData) {
```

```

[NSThread sleepForTimeInterval:0.01];
// 可以检查是否尝试避免创建无限循环
}

UIImage *uImage = images[i];

CVPixelBufferRef buffer = NULL;
CVRReturn err = PixelBufferCreateFromImage(uImage.CGImage, &buffer);
if (err) {
    // 处理错误
}

// 帧持续时间是单张图像的持续时间，单位为秒
CMTime presentationTime = CMTimeMakeWithSeconds(i * frameDuration, 0000000);

[writerAdaptor appendPixelBuffer:buffer withPresentationTime:presentationTime];

CVPixelBufferRelease(buffer);
}

[assetWriterInput markAsFinished];
[assetWriter finishWritingWithCompletionHandler:^{
    if (assetWriter.error) {
        // 显示错误信息
    } else {
        // outputURL
    }
}];
}

```

这是一个从CGImageRef获取CVPixelBufferRef的函数

```

CVRReturn PixelBufferCreateFromImage(CGImageRef imageRef, CVPixelBufferRef *outBuffer) {
    CIContext *context = [CIContext context];
    CIIImage *ciImage = [CIIImage imageWithCGImage:imageRef];

    NSDictionary *attributes = [NSDictionary dictionaryWithObjectsAndKeys:
        [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGBitmapContextCompatibilityKey,
        [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGImageCompatibilityKey
        ,nil];

    CVReturn err = CVPixelBufferCreate(kCFAllocatorDefault, CGImageGetWidth(imageRef),
    CGImageGetHeight(imageRef), kCVPixelFormatType_32ARGB, (_bridge CFDictionaryRef
    _Nullable)(attributes), outBuffer);
    if (err) {
        return err;
    }

    if (outBuffer) {
        [context render:ciImage toCVPixelBuffer:*outBuffer];
    }

    return kCVReturnSuccess;
}

```

```

[NSThread sleepForTimeInterval:0.01];
// can check for attempts not to create an infinite loop
}

UIImage *uImage = images[i];

CVPixelBufferRef buffer = NULL;
CVRReturn err = PixelBufferCreateFromImage(uImage.CGImage, &buffer);
if (err) {
    // handle error
}

// frame duration is duration of single image in seconds
CMTime presentationTime = CMTimeMakeWithSeconds(i * frameDuration, 1000000);

[writerAdaptor appendPixelBuffer:buffer withPresentationTime:presentationTime];

CVPixelBufferRelease(buffer);
}

[assetWriterInput markAsFinished];
[assetWriter finishWritingWithCompletionHandler:^{
    if (assetWriter.error) {
        // show error message
    } else {
        // outputURL
    }
}];
}

```

Here is a function to get CVPixelBufferRef from CGImageRef

```

CVRReturn PixelBufferCreateFromImage(CGImageRef imageRef, CVPixelBufferRef *outBuffer) {
    CIContext *context = [CIContext context];
    CIIImage *ciImage = [CIIImage imageWithCGImage:imageRef];

    NSDictionary *attributes = [NSDictionary dictionaryWithObjectsAndKeys:
        [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGBitmapContextCompatibilityKey,
        [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGImageCompatibilityKey
        ,nil];

    CVReturn err = CVPixelBufferCreate(kCFAllocatorDefault, CGImageGetWidth(imageRef),
    CGImageGetHeight(imageRef), kCVPixelFormatType_32ARGB, (_bridge CFDictionaryRef
    _Nullable)(attributes), outBuffer);
    if (err) {
        return err;
    }

    if (outBuffer) {
        [context render:ciImage toCVPixelBuffer:*outBuffer];
    }

    return kCVReturnSuccess;
}

```

第206章： Codable

[Codable](#) 是在Xcode 9、iOS 11和Swift 4中新增的。 Codable 用于使你的数据类型可编码和可解码，以便与JSON等外部表示兼容。

Codable 用于支持编码和解码，声明遵循 Codable 协议，该协议结合了 Encodable 和 Decodable 协议。这个过程称为使你的类型可编码。

第206.1节：在Swift 4中使用 Codable 与 JSONEncoder 和 JSONDecoder

让我们以电影结构体为例，这里我们将结构体定义为 Codable。因此，我们可以轻松地对其进行编码和解码。

```
struct Movie: Codable {
    enum MovieGenre: String, Codable {
        case horror, sciFi, comedy, adventure, animation
    }

    var name: String
    var moviesGenre: [MovieGenre]
    var rating: Int
}
```

我们可以像下面这样创建一个电影对象：

```
let upMovie = Movie(name: "Up", moviesGenre: [.comedy, .adventure, .animation], rating: 4)
```

upMovie 包含名称“Up”，其电影类型为喜剧、冒险和动画，评分为4（满分5分）。

编码

JSONEncoder 是一个将数据类型实例编码为 JSON 对象的对象。 JSONEncoder 支持 Codable 对象。

```
// 编码数据
let jsonEncoder = JSONEncoder()
do {
    let jsonData = try jsonEncoder.encode(upMovie)
    let jsonString = String(data: jsonData, encoding: .utf8)
    print("JSON String : " + jsonString!)
}
catch { }
```

JSONEncoder 会给我们 JSON 数据，用于获取 JSON 字符串。

输出字符串将会是：

```
{
    "name": "Up",
    "moviesGenre": [
        "comedy",
        "adventure",
        "animation"
    ]}
```

Chapter 206: Codable

[Codable](#) is added with Xcode 9, iOS 11 and Swift 4. Codable is used to make your data types encodable and decodable for compatibility with external representations such as JSON.

Codable uses both encoding and decoding, declare conformance to Codable, which combines the Encodable and Decodable protocols. This process is known as making your types codable.

Section 206.1: Use of Codable with JSONEncoder and JSONDecoder in Swift 4

Let's take an example with the structure of Movie, here we have defined the structure as Codable. So, we can encode and decode it easily.

```
struct Movie: Codable {
    enum MovieGenre: String, Codable {
        case horror, sciFi, comedy, adventure, animation
    }

    var name: String
    var moviesGenre: [MovieGenre]
    var rating: Int
}
```

We can create an object from movie like as:

```
let upMovie = Movie(name: "Up", moviesGenre: [.comedy, .adventure, .animation], rating: 4)
```

The upMovie contains the name "Up" and its movieGenre is comedy, adventure and animation which contains 4 rating out of 5.

Encode

JSONEncoder is an object that encodes instances of a data type as JSON objects. JSONEncoder supports the Codable object.

```
// Encode data
let jsonEncoder = JSONEncoder()
do {
    let jsonData = try jsonEncoder.encode(upMovie)
    let jsonString = String(data: jsonData, encoding: .utf8)
    print("JSON String : " + jsonString!)
}
catch { }
```

JSONEncoder will give us the JSON data which is used to retrieve JSON string.

Output string will be like :

```
{
    "name": "Up",
    "moviesGenre": [
        "comedy",
        "adventure",
        "animation"
    ]}
```

```
],  
    "rating": 4  
}
```

解码

JSONDecoder 是一个用于从 JSON 对象解码数据类型实例的对象。我们可以从 JSON 字符串中获取该对象。

```
do {  
    // 将数据解码为对象  
  
    let jsonDecoder = JSONDecoder()  
    let upMovie = try jsonDecoder.decode(Movie.self, from: jsonData)  
    print("评分 : \(upMovie.name)")  
    print("评分 : \(upMovie.rating)")  
}  
catch {  
}
```

通过解码 JSONData，我们将收到 Movie 对象。因此我们可以获取保存在该

对象中的所有值。

输出将会是：

```
名称 : Up  
评分 : 4
```

```
],  
    "rating": 4  
}
```

Decode

JSONDecoder is an object that decodes instances of a data type from JSON objects. We can get the object back from the JSON string.

```
do {  
    // Decode data to object  
  
    let jsonDecoder = JSONDecoder()  
    let upMovie = try jsonDecoder.decode(Movie.self, from: jsonData)  
    print("Rating : \(upMovie.name)")  
    print("Rating : \(upMovie.rating)")  
}  
catch {  
}
```

By decoding the JSONData we will receive the Movie object back. So we can get all the values which is saved in that object.

Output will be like:

```
Name : Up  
Rating : 4
```

第207章：异步加载图片

第207.1节：最简单的方法

创建这个最简单的方法是使用Alamofire及其UIImageViewExtension。我们需要的是一个带有 imageView 的表格视图单元格，称之为imageView。

在 tableView 的 cellForRowAt: 函数中，我们将以以下方式下载图片并设置：

```
let url = URL(string: "https://httpbin.org/image/png")!
let placeholderImage = UIImage(named: "placeholder")!

imageView.af_setImage(withURL: url, placeholderImage: placeholderImage)
```

url 应指向你想下载的图片，placeHolder 图片应为已存储的图片。

然后我们调用 af_setImage 方法在 imageView 上，该方法会下载指定 url 的图片，在下载过程中会显示占位图片。一旦图片下载完成，请求的图片就会显示出来。

第 207.2 节：下载后检查单元格是否仍然可见

有时下载时间比单元格显示时间长。在这种情况下，可能会出现下载的图片显示在错误的单元格中。为了解决这个问题，我们不能使用 UIImageView Extension。

我们仍然会使用 Alamofire，但会使用完成处理器来显示图片。

在这种情况下，我们仍然需要一个带有 imageView 的 tableView 单元格。在 cellForRowAt: 方法中，我们会用以下代码下载图片：

```
let placeholderImage = UIImage(named: "placeholder")!
imageView.image = placeholderImage

let url = URL(string: "https://httpbin.org/image/png")!

Alamofire.request(url!, method: .get).responseImage { response in
    guard let image = response.result.value else { return }

    if let updateCell = tableView.cellForRow(at: indexPath) {
        updateCell.imageView.image = image
    }
}
```

在这个例子中，我们首先将图像设置为占位符图像。随后，我们使用 Alamofire 的 request 方法下载图像。我们将 URL 作为第一个参数传入，由于我们只想获取图像，因此使用.get HTTP 方法。因为我们下载的是图像，所以希望响应为图像，因此使用.responseImage 方法。

图像下载完成后，闭包被调用，首先确认下载的图像确实存在。然后通过检查 cellForRow(at: indexPath) 不返回 nil，确保该单元格仍然可见。如果返回 nil，则不做任何操作；如果不为 nil，则将刚下载的图像赋值给该单元格。

最后这个 if 语句确保单元格仍然可见，如果用户已经滚动过该单元格，updateCell 将为 nil，if 语句返回 nil。这有助于防止在单元格中显示错误的图像。

Chapter 207: Load images async

Section 207.1: Easiest way

The most simple way to create this is to use [Alamofire](#) and its [UIImageViewExtension](#). What we need is a tableview with a cell that has an imageView in it and lets call it imageView.

In the cellForRowAt: function of the tableView we would download the image and set in the following way:

```
let url = URL(string: "https://httpbin.org/image/png")!
let placeholderImage = UIImage(named: "placeholder")!

imageView.af_setImage(withURL: url, placeholderImage: placeholderImage)
```

The url should point to the image that you want to download and the placeHolder image should be a stored image. We then call the af_setImage method on the imageView which downloads the image at the given url and during the download the placeholder image will be shown. As soon as the image is downloaded the requested image is displayed

Section 207.2: Check that the cell is still visible after download

Sometimes the download takes longer than the cell is being displayed. In this case it can happen, that the downloaded image is shown in the wrong cell. To fix this we can not use the [UIImageView Extension](#).

We still will be using [Alamofire](#) however we will use the completion handler to display the image.

In this scenario we still need a tableView with a cell which has a imageView in it. In the cellForRowAt: method we would download the image with the following code:

```
let placeholderImage = UIImage(named: "placeholder")!
imageView.image = placeholderImage

let url = URL(string: "https://httpbin.org/image/png")!

Alamofire.request(url!, method: .get).responseImage { response in
    guard let image = response.result.value else { return }

    if let updateCell = tableView.cellForRow(at: indexPath) {
        updateCell.imageView.image = image
    }
}
```

In this example we first set the image to the placeholder image. Afterwards we download the image with the request method of [Alamofire](#). We pass the url as the first argument and since we just want to get the image we will use the .get HTTP method. Since we are downloading an image we want the response to be an image therefore we use the .responseImage method.

After the image has been downloaded the closure gets called and first of all we make sure that the downloaded image actually exists. Then we make sure that the cell is still visible by checking that the cellForRow(at: indexPath) doesn't return nil. If it does nothing happens, if it doesn't we assign the recently downloaded image.

This last if statement ensures that the cell is still visible if the user already scrolled over the cell the updateCell will be nil and the if statement returns nil. This helps us prevent displaying the wrong image in a cell.

第208章：添加Swift桥接头文件

第208.1节：如何手动创建Swift桥接头文件

- 在Xcode中添加新文件（文件 > 新建 > 文件），然后选择“源文件”并点击“头文件”。
- 将文件命名为“YourProjectName-Bridging-Header.h”。例如：在我的应用Station中，文件名为“Station-Bridging-Header”。
- 创建文件。
- 导航到你的项目构建设置，找到“Swift 编译器-代码生成”部分。你可以在搜索框中输入“Swift 编译器”来更快地缩小搜索结果。注意：如果你没有“Swift 编译器-代码生成”部分，这意味着你可能还没有向项目中添加任何 Swift 类。添加一个 Swift 文件，然后再试一次。
- 在“Objective-C 桥接头文件”旁边，你需要添加头文件的名称/路径。如果你的文件位于项目的根文件夹，只需填写头文件的名称。示例：“ProjectName/ProjectName-Bridging-Header.h”或简单地“ProjectName-Bridging-Header.h”。
- 打开你新创建的桥接头文件，使用#import语句导入你的 Objective-C 类。该文件中列出的任何类都可以从你的 Swift 类中访问。

第208.2节：Xcode 自动创建

向你的 Xcode 项目添加一个新的 Swift 文件。随意命名，系统会弹出一个提示框，询问你是否想创建桥接头文件。注意：如果你没有收到添加桥接头文件的提示，可能是你之前拒绝过此消息，你需要手动添加头文件（见下文）。

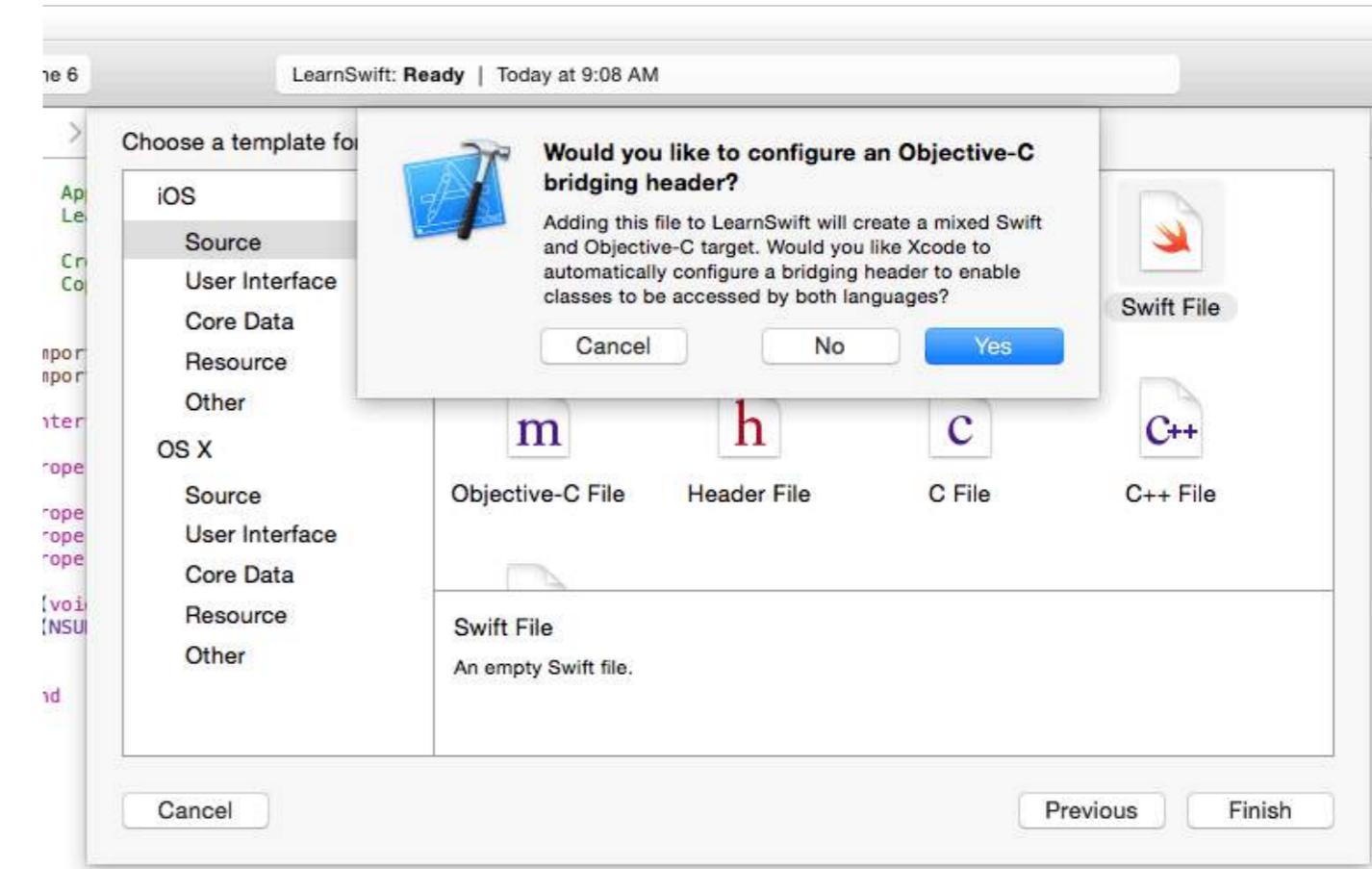
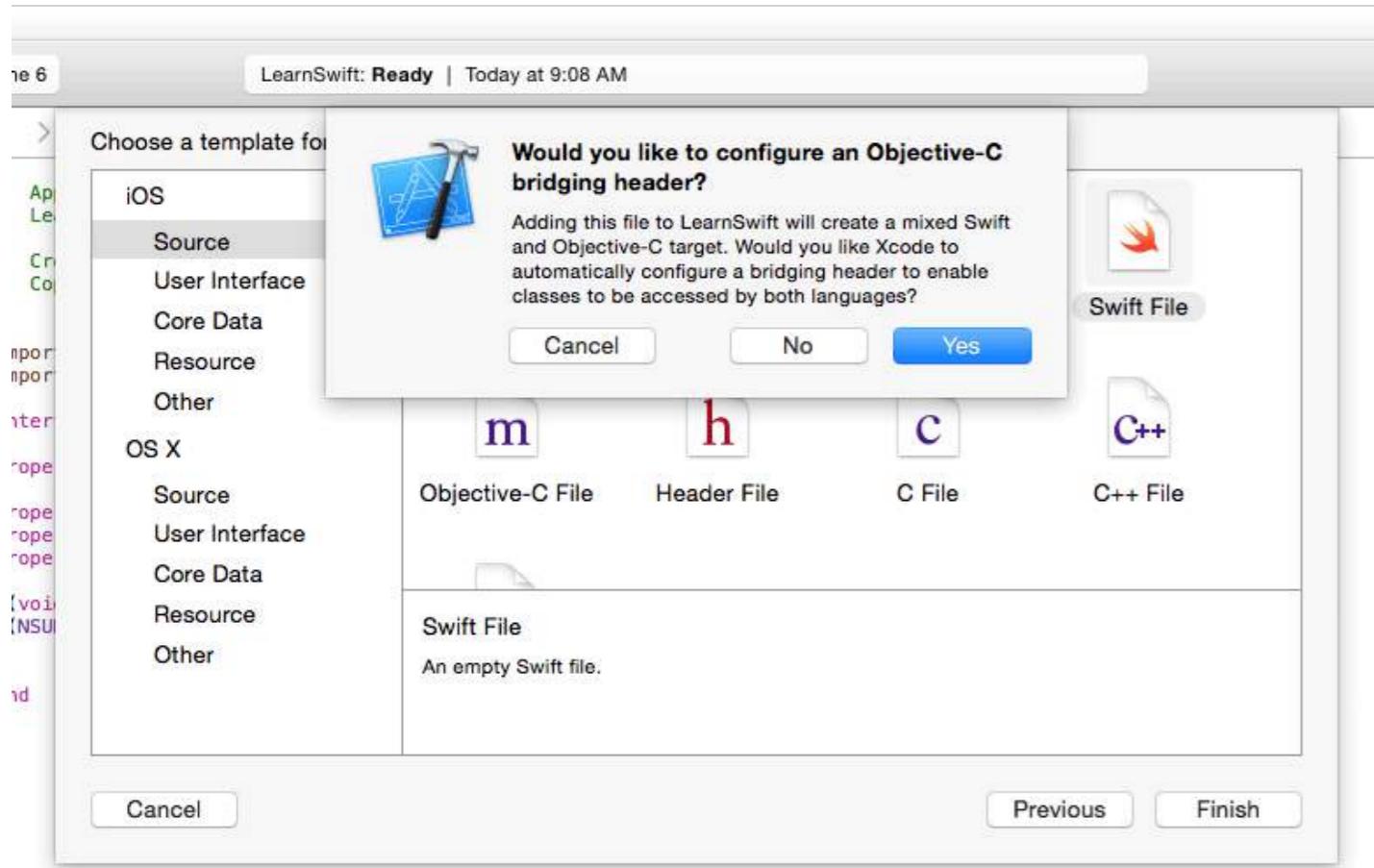
Chapter 208: Adding a Swift Bridging Header

Section 208.1: How to create a Swift Bridging Header Manually

- Add a new file to Xcode (File > New > File), then select “Source” and click “Header File”.
- Name your file “YourProjectName-Bridging-Header.h”. Example: In my app Station, the file is named “Station-Bridging-Header”.
- Create the file.
- Navigate to your project build settings and find the “Swift Compiler – Code Generation” section. You may find it faster to type in “Swift Compiler” into the search box to narrow down the results. Note: If you don’t have a “Swift Compiler – Code Generation” section, this means you probably don’t have any Swift classes added to your project yet. Add a Swift file, then try again.
- Next to “Objective-C Bridging Header” you will need to add the name/path of your header file. If your file resides in your project’s root folder simply put the name of the header file there. Examples: “ProjectName/ProjectName-Bridging-Header.h” or simply “ProjectName-Bridging-Header.h”.
- Open up your newly created bridging header and import your Objective-C classes using #import statements. Any class listed in this file will be able to be accessed from your swift classes.

Section 208.2: Xcode create automatically

Add a new Swift file to your Xcode project. Name it as you please and you should get an alert box asking if you would like to create a bridging header. Note: If you don’t receive a prompt to add a bridging header, you probably declined this message once before and you will have to add the header manually (see below)



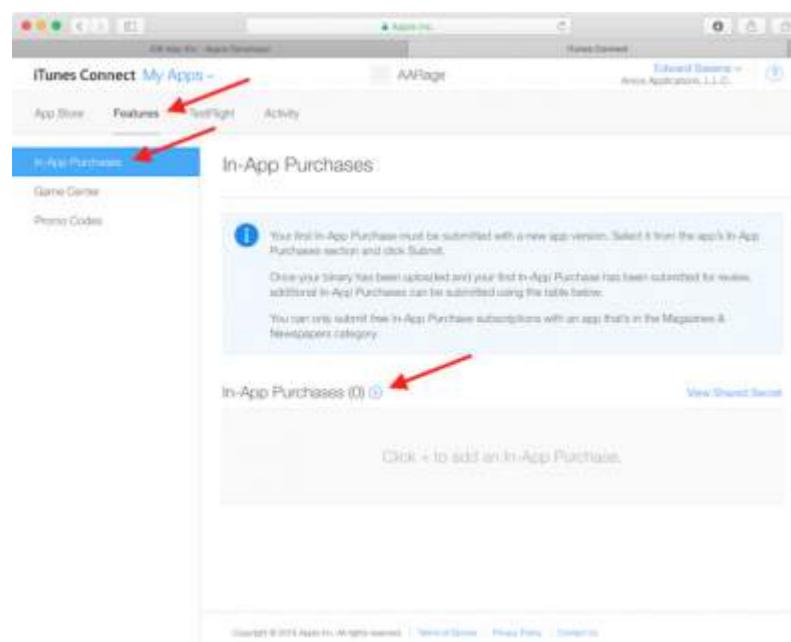
第209章：创建应用程序ID

第209.1节：创建应用内购买产品

- 在应用内提供IAP时，您必须先在iTunes Connect中为每个单独的购买添加一个条目。如果您曾经在商店中上架过应用程序，流程类似，包括选择购买的定价等级。当用户进行购买时，App Store会处理向用户的iTunes账户收费的复杂过程。您可以添加多种不同类型的IAP：

- 消耗型：这些可以多次购买并且会被消耗掉。比如额外生命、游戏内货币、临时增强道具等。
- 非消耗型：一次购买，期望永久拥有的内容，如额外关卡和可解锁内容。
- 非自动续订订阅：在固定时间内可用的内容。
- 自动续订订阅：重复订阅，例如每月的raywenderlich.com订阅。

您只能为数字商品提供应用内购买，不能用于实物商品或服务。有关详细信息，请查看Apple关于创建应用内购买产品的完整文档。现在，在查看您应用在iTunes Connect中的条目时，点击“功能”标签，然后选择“应用内购买”。要添加新的IAP产品，点击“应用内购买”右侧的“+”号。



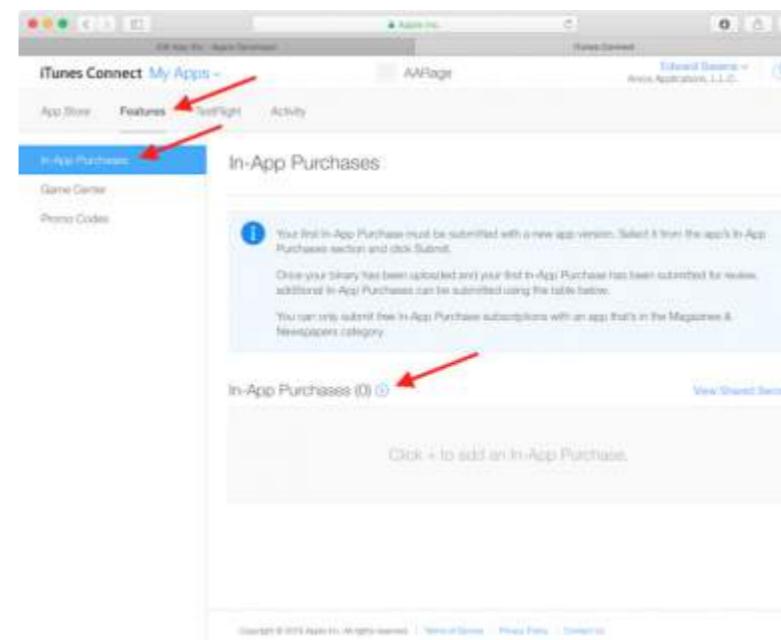
您将看到以下对话框出现：

Chapter 209: Creating an App ID

Section 209.1: Creating In-App Purchase Products

- When offering IAP within an app, you must first add an entry for each individual purchase within iTunes Connect. If you've ever listed an app for sale in the store, it's a similar process and includes things like choosing a pricing tier for the purchase. When the user makes a purchase, the App Store handles the complex process of charging the user's iTunes account. There are a whole bunch of different types of IAP you can add:
 - Consumable:** These can be bought more than once and can be used up. These are things such as extra lives, in-game currency, temporary power-ups, and the like.
 - Non-Consumable:** Something that you buy once, and expect to have permanently such as extra levels and unlockable content.
 - Non-Renewing Subscription:** Content that's available for a fixed period of time.
 - Auto-Renewing Subscription:** A repeating subscription such as a monthly raywenderlich.com subscription.

You can only offer In-App Purchases for digital items, and not for physical goods or services. For more information about all of this, check out Apple's full documentation on Creating In-App Purchase Products. Now, while viewing your app's entry in iTunes Connect, click on the Features tab and then select In-App Purchases. To add a new IAP product, click the + to the right of In-App Purchases.



You will see the following dialog appear:

Select the In-App Purchase you want to create.

Consumable

A product that is used once, after which it becomes depleted and must be purchased again.

Example: Fish food for a fishing app.

Non-Consumable

A product that is purchased once and does not expire or decrease with use.

Example: Race track for a game app.

Auto-Renewable Subscription

A product that allows users to purchase dynamic content for a set period. This type of subscription renews automatically unless cancelled by the user.

Example: Monthly subscription for an app offering a streaming service.

Non-Renewing Subscription

A product that allows users to purchase a service with a limited duration. The content of this in-app purchase can be static. This type of subscription does not renew automatically.

Example: Annual subscription to a catalog of archived articles.

[Learn more about In-App Purchases.](#)

[Cancel](#)

[Create](#)

当用户在你的应用中购买一个rage漫画时，你会希望他们始终能够访问它，所以选择非消耗型（Non-Consumable），然后点击创建。接下来，按如下填写IAP的详细信息：

- 参考名称：在iTunes Connect中识别IAP的昵称。此名称不会出现在应用的任何地方。你将通过此次购买解锁的漫画标题是“鼓手的女朋友”，所以在这里输入该名称。
- 产品ID：这是识别IAP的唯一字符串。通常最好以Bundle ID开头，然后附加一个特定于该可购买项目的唯一名称。对于本教程，请确保附加“GirlfriendOfDrummerRage”，因为稍后应用中会用它来查找要解锁的漫画。例如：com.theNameYouPickedEarlier.Rage.GirlFriendOfDrummerRage。
- 允许销售：启用或禁用IAP的销售。你需要启用它！
- 价格等级：IAP的价格。选择等级1。

现在向下滚动到本地化部分，注意默认有一个英语（美国）的条目。将“鼓手的女朋友”输入到显示名称和描述中。点击保存。太好了！你已经创建了第一个IAP产品。

Select the In-App Purchase you want to create.

Consumable

A product that is used once, after which it becomes depleted and must be purchased again.

Example: Fish food for a fishing app.

Non-Consumable

A product that is purchased once and does not expire or decrease with use.

Example: Race track for a game app.

Auto-Renewable Subscription

A product that allows users to purchase dynamic content for a set period. This type of subscription renews automatically unless cancelled by the user.

Example: Monthly subscription for an app offering a streaming service.

Non-Renewing Subscription

A product that allows users to purchase a service with a limited duration. The content of this in-app purchase can be static. This type of subscription does not renew automatically.

Example: Annual subscription to a catalog of archived articles.

[Learn more about In-App Purchases.](#)

[Cancel](#)

[Create](#)

When a user purchases a rage comic in your app, you'll want them to always have access to it, so select Non-Consumable, and click Create. Next, fill out the details for the IAP as follows:

- Reference Name:** A nickname identifying the IAP within iTunes Connect. This name does not appear anywhere in the app. The title of the comic you'll be unlocking with this purchase is “**Girlfriend of Drummer**”, so enter that here.
- Product ID:** This is a unique string identifying the IAP. Usually it's best to start with the Bundle ID and then append a unique name specific to this purchasable item. For this tutorial, make sure you append “**GirlfriendOfDrummerRage**”, as this will be used later within the app to look up the comic to unlock. So, for example: com.theNameYouPickedEarlier.Rage.GirlFriendOfDrummerRage.
- Cleared for Sale:** Enables or disables the sale of the IAP. You want to enable it!
- Price Tier:** The cost of the IAP. Choose Tier 1.

Now scroll down to the Localizations section and note that there is a default entry for English (U.S.). Enter “**Girlfriend of Drummer**” for both the Display Name and the Description. Click Save. Great! You've created your first IAP product.

Localizations

English (U.S.)

Display Name 	Girlfriend of Drummer
Description 	Girlfriend of Drummer

234

在深入编写代码之前，还有一步需要完成。当在应用的开发版本中测试应用内购买时，苹果提供了一个测试环境，允许你‘购买’你的IAP产品而无需进行真实的财务交易。

第209.2节：创建沙盒用户

在 iTunes Connect 中，点击窗口左上角的 iTunes Connect 返回主菜单。选择用户与角色，然后点击沙盒测试人员标签。点击“测试人员”标题旁的 + 号。



The screenshot shows the 'iTunes Connect' interface with the 'Users and Roles' section selected. The 'Sandbox Testers' tab is active. A single tester named 'Tester (1)' is listed. A red arrow points to the 'Tester (1)' link.

填写信息并在完成后点击保存。您可以为测试用户编造一个名字和姓氏，但所选的电子邮件地址必须是真实的，因为苹果会向该地址发送验证邮件。

收到邮件后，请务必点击其中的链接以验证您的地址。您输入的电子邮件地址也不应已关联任何 Apple ID 账户。提示：如果您有 Gmail 账户，可以使用地址别名，而无需创建全新的账户。

Localizations

English (U.S.)

Display Name 	Girlfriend of Drummer
Description 	Girlfriend of Drummer

234

There's one more step required before you can delve into some code. When testing in-app purchases in a development build of an app, Apple provides a test environment which allows you to 'purchase' your IAP products without creating financial transactions.

Section 209.2: Creating a Sandbox User

In iTunes Connect, click iTunes Connect in the top left corner of the window to get back to the main menu. Select Users and Roles, then click the Sandbox Testers tab. Click + next to the "Tester" title.



The screenshot shows the 'iTunes Connect' interface with the 'Users and Roles' section selected. The 'Sandbox Testers' tab is active. A single tester named 'Tester (1)' is listed. A red arrow points to the 'Tester (1)' link.

Fill out the information and click Save when you're done. You can make up a first and last name for your test user, but the email address chosen must be a real email address as a verification will be sent to the address by Apple.

Once you receive that email, be sure to click the link in it to verify your address. The email address you enter should also NOT already be associated with an Apple ID account. Hint: if you have a gmail account, you can simply use an address alias instead of having to create a brand new account

第210章：Swift：在 AppDelegate 中更改 rootViewController 以呈现主界面或登录/引导流程

向新用户展示首次使用体验通常很有用。这可能出于多种原因，例如提示他们登录（如果您的情况需要）、解释如何使用应用，或仅仅是告知他们更新中的新功能（如 iOS11 中的备忘录、照片和音乐）。

第210.1节：选项1：切换根视图控制器（良好）

切换根视图控制器有其好处，尽管过渡选项仅限于 UIViewAnimationOptions 支持的那些，因此根据您希望在流程间如何过渡，可能需要实现自定义过渡——这可能比较繁琐。

您可以通过简单设置 UIApplication.shared.keyWindow.rootViewController 来显示引导流程。解除显示则通过使用 UIView.transition(with:) 并传入过渡样式作为 UIViewAnimationOptions，这里使用的是交叉淡入淡出（Cross Dissolve）。（翻转和卷曲效果也支持）。

在切换回主视图之前，您还必须设置主视图的 frame，因为这是您第一次实例化它。

```
// MARK: - 引导流程

extension AppDelegate {

    func 显示引导流程() {
        if let window = UIApplication.shared.keyWindow, let onboardingViewController = UIStoryboard(name: "Onboarding", bundle: nil).instantiateInitialViewController() as? OnboardingViewController {
            onboardingViewController.delegate = self
            window.rootViewController = onboardingViewController
        }
    }

    func 隐藏引导流程() {
        if let window = UIApplication.shared.keyWindow, let mainViewController = UIStoryboard(name: "Main", bundle: nil).instantiateInitialViewController() {
            mainViewController.view.frame = window.bounds
            UIView.transition(with: window, duration: 0.5, options: .transitionCrossDissolve, animations: {
                window.rootViewController = mainViewController
            }, completion: nil)
        }
    }
}
```

第210.2节：选项2：以模态方式呈现替代流程（更好）

在最简单的实现中，入职流程可以直接以模态窗口的形式呈现，因为从语义上讲，用户处于单一的流程中。

Chapter 210: Swift: Changing the rootViewController in AppDelegate to present main or login/onboarding flow

It is often useful to present a first-run experience to new users of your App. This could be for any number of reasons, such as prompting them to sign in (if required for your situation), explaining how to use the App, or simply informing them of new features in an update (as Notes, Photos and Music do in iOS11).

Section 210.1: Option 1: Swap the Root View Controller (Good)

There are benefits to switching the root view controller, although the transition options are limited to those supported by UIViewAnimationOptions, so depending on how you wish to transition between flows might mean you have to implement a custom transition - which can be cumbersome.

You can show the Onboarding flow by simply setting the `UIApplication.shared.keyWindow.rootViewController`

Dismissal is handled by utilizing `UIView.transition(with:)` and passing the transition style as a `UIViewAnimationOptions`, in this case Cross Dissolve. (Flips and Curls are also supported).

You also have to set the frame of the Main view before you transition back to it, as you're instantiating it for the first time.

```
// MARK: - Onboarding

extension AppDelegate {

    func showOnboarding() {
        if let window = UIApplication.shared.keyWindow, let onboardingViewController = UIStoryboard(name: "Onboarding", bundle: nil).instantiateInitialViewController() as? OnboardingViewController {
            onboardingViewController.delegate = self
            window.rootViewController = onboardingViewController
        }
    }

    func hideOnboarding() {
        if let window = UIApplication.shared.keyWindow, let mainViewController = UIStoryboard(name: "Main", bundle: nil).instantiateInitialViewController() {
            mainViewController.view.frame = window.bounds
            UIView.transition(with: window, duration: 0.5, options: .transitionCrossDissolve, animations: {
                window.rootViewController = mainViewController
            }, completion: nil)
        }
    }
}
```

Section 210.2: Option 2: Present Alternative Flow Modally (Better)

In the most straightforward implementation, the Onboarding flow can simply be presented in a modal context, since semantically the User is on a single journey.

[Apple 人机界面指南 – 模态][1]:

仅在必须引起用户注意、任务必须完成或放弃才能继续使用应用程序，或需要保存重要数据时，才考虑创建模态上下文。

以模态方式呈现允许在流程结束时简单地关闭，避免了切换控制器的繁琐操作。

标准方式也支持自定义过渡，因为这使用了ViewController.present() API：

```
// MARK: - 引导流程

extension AppDelegate {

    func 显示引导流程() {
        if let window = window, let onboardingViewController = UIStoryboard(name: "Onboarding",
bundle: nil).instantiateViewController() as? OnboardingViewController {
            onboardingViewController.delegate = self
            window.makeKeyAndVisible()
            window.rootViewController?.present(onboardingViewController, animated: false,
completion: nil)
        }
    }

    func 隐藏引导流程() {
        if let window = UIApplication.shared.keyWindow {
            window.rootViewController?.dismiss(animated: true, completion: nil)
        }
    }
}
```

[Apple Human Interface Guidelines – Modality][1]:

Consider creating a modal context only when it's critical to get someone's attention, when a task must be completed or abandoned to continue using the app, or to save important data.

Presenting modally allows the simple option of dismissal at the end of the journey, with little of the cruft of swapping controllers.

Custom transitions are also supported in the standard way, since this uses the ViewController.present() API:

```
// MARK: - Onboarding

extension AppDelegate {

    func showOnboarding() {
        if let window = window, let onboardingViewController = UIStoryboard(name: "Onboarding",
bundle: nil).instantiateViewController() as? OnboardingViewController {
            onboardingViewController.delegate = self
            window.makeKeyAndVisible()
            window.rootViewController?.present(onboardingViewController, animated: false,
completion: nil)
        }
    }

    func hideOnboarding() {
        if let window = UIApplication.shared.keyWindow {
            window.rootViewController?.dismiss(animated: true, completion: nil)
        }
    }
}
```

致谢

非常感谢所有来自Stack Overflow Documentation的人员帮助提供这些内容，
更多更改可以发送至web@petercv.com以发布或更新新内容

4oby	第2章
阿比吉特	第192章
abjurato	第149章
亚当·埃伯巴赫	第11章和第38章
亚当·普雷布尔	第27章
阿德南·阿夫塔布	第79章
阿德里安娜·卡雷利	第46章和第151章
艾哈迈德·阿卜杜拉	第41章、第89章和第107章
艾哈迈德·哈拉夫	第28章
AJ9	第14章
ajmccall	第65章
Ajumal	第15章
阿基兰·阿拉苏	第2章
alaphao	第20章和第67章
亚历克斯·卡拉姆	第74章
亚历克斯·科希	第2、14、34、38、74和95章
亚历克斯·劳斯	第61章
亚历山大·特卡琴科	第64章
阿列克西	第40和92章
阿里·阿巴斯	第178章
阿里·比德尔	第1章
阿里·埃尔索卡里	第34章
阿利斯特拉	第89章
艾尔文·阿比亚	第122章
阿曼普里特	第107章
阿马尔	第67、96和97章
阿南德·尼姆杰	第1、38、40、74、76和117章
阿纳托利	第1章
安德烈亚斯	第25、45和151章
安德烈斯·卡内拉	第2章
安德烈斯·基耶夫斯基	第14章
安德里·切尔年科	第2章
安·范	第11和20章
阿尼什·吴	第20章、第72章和第97章
安基特·乔汉	第190章
ApolloSoftware	第79章
Arefly	第5章和第96章
阿希什·卡卡德	第107章、第113章和第206章
阿舒托什·戴夫	第1章
askielboe	第142章
AstroCB	第2章
阿西莫夫	第47、56和119章
反斜杠	第52章
巴达尔·沙阿	第30、67和144章
巴拉古鲁巴兰	第77章
豆	第53章和第67章
本塞·帕托加托	第74章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

4oby	Chapter 2
Abhijit	Chapter 192
abjurato	Chapter 149
Adam Eberbach	Chapters 11 and 38
Adam Preble	Chapter 27
Adnan Aftab	Chapter 79
Adriana Carelli	Chapters 46 and 151
Ahmed Abdallah	Chapters 41, 89 and 107
Ahmed Khalaf	Chapter 28
AJ9	Chapter 14
ajmccall	Chapter 65
Ajumal	Chapter 15
Akilan Arasu	Chapter 2
alaphao	Chapters 20 and 67
Alex Kallam	Chapter 74
Alex Koshy	Chapters 2, 14, 34, 38, 74 and 95
Alex Rouse	Chapter 61
Alexander Tkachenko	Chapter 64
Alexi	Chapters 40 and 92
Ali Abbas	Chapter 178
Ali Beadle	Chapter 1
Ali Elsokary	Chapter 34
Alistra	Chapter 89
Alvin Abia	Chapter 122
Amanpreet	Chapter 107
amar	Chapters 67, 96 and 97
Anand Nimje	Chapters 1, 38, 40, 74, 76 and 117
Anatoliy	Chapter 1
Andreas	Chapters 25, 45 and 151
Andres Canella	Chapter 2
Andres Kievsky	Chapter 14
Andrii Chernenko	Chapter 2
Anh Pham	Chapters 11 and 20
Anish 吴	Chapters 20, 72 and 97
Ankit chauhan	Chapter 190
ApolloSoftware	Chapter 79
Arefly	Chapters 5 and 96
Ashish Kakkad	Chapters 107, 113 and 206
Ashutosh Dave	Chapter 1
askielboe	Chapter 142
AstroCB	Chapter 2
azimov	Chapters 47, 56 and 119
backslash	Chapter 52
Badal Shah	Chapters 30, 67 and 144
balagurubaran	Chapter 77
Bean	Chapters 53 and 67
Bence Pattogato	Chapter 74

[BennX](#)
[本特福德](#)
[贝托·卡尔达斯](#)
[beyowulf](#)
[巴德雷什·卡蒂里亚](#)
[巴文·拉马尼](#)
[布米特·梅赫塔](#)
[布莱克德沃罗](#)
[蓝翼](#)
[bluey31](#)
[bmike](#)
[Boletrone](#)
[Bonnie](#)
[Brandon](#)
[breakingobstacles](#)
[布伦登·罗伯托](#)
[布雷特·庞德](#)
[布赖恩](#)
[光明的未来](#)
[bryanjclark](#)
作者 : Jeevan
[凯勒布·克莱维特](#)
[查尔斯](#)
[查图兰加·席尔瓦](#)
[奇拉格·德赛](#)
[克里斯·布兰兹马](#)
[Cin316](#)
[cleverbit](#)
[代码战士](#)
[代码更改者](#)
[科里·威尔海特](#)
克里斯
[克里斯蒂娜](#)
[西里尔·伊瓦尔·加西亚](#)
[达塔特拉亚](#)
[丹尼尔·奥尔梅诺](#)
[dannyzlo](#)
[danshevluk](#)
[达尔希特·沙阿](#)
[dasdom](#)
[ddb](#)
[DeyaEldeen](#)
[deyanm](#)
[dgatwood](#)
[Dima Deploy](#)
[迪内什·拉贾](#)
[迪彭·潘查萨拉](#)
[迪尚特·卡帕迪亚](#)
[争议](#)
[文档](#)
[道格拉斯·斯塔恩斯](#)
[杜利·金斯基](#)
[敦雅·拉利奇](#)

第131章
第2章
第189章
第55、57和59章
第107章和第151章
第50章
第2章、第7章、第38章、第67章和第93章
第67章
第171章
第75章
第1章
第24章
第73、107和134章
第33、43、116和124章
第65章和第198章
第14章和第83章
第23章
第2、7、10、14、68、104和119章
第47章
第135章
第2章和第78章
第2章、第11章和第20章
第73章
第2章
第134章
第2章
第210章
第2章和第52章
第174章
第20章
第24章
第40章
第101、148和154章
第7、20、35、133、145、197和201章
第137章
第27章
第2和61章
第21章
第2、5、23、26、36、75和133章
第2、5、20和38章
第2、11、20和23章
第1章
第75章
第14章、第63章和第67章
第67章
第8章
第203章
第63章
第86章
第20章
第34章、第75章和第97章
第8章、第24章、第27章、第36章、第73章、第119章和第135章

[BennX](#)
[bentford](#)
[Beto Caldas](#)
[beyowulf](#)
[Bhadresh Kathiriya](#)
[Bhavin Ramani](#)
[Bhumit Mehta](#)
[BlackDeveraux](#)
[Bluewings](#)
[bluey31](#)
[bmike](#)
[Boletrone](#)
[Bonnie](#)
[Brandon](#)
[breakingobstacles](#)
[Brendon Roberto](#)
[Brett Ponder](#)
[Brian](#)
[Bright Future](#)
[bryanjclark](#)
[byJeevan](#)
[Caleb Kleveter](#)
[Charles](#)
[Chathuranga Silva](#)
[Chirag Desai](#)
[Chris Brandsma](#)
[Cin316](#)
[cleverbit](#)
[Code.Warrior](#)
[CodeChanger](#)
[Cory Wilhite](#)
[Cris](#)
[Cristina](#)
[Cyril Ivar Garcia](#)
[D4ttatraya](#)
[Daniel Ormeño](#)
[dannyzlo](#)
[danshevluk](#)
[Darshit Shah](#)
[dasdom](#)
[ddb](#)
[DeyaEldeen](#)
[deyanm](#)
[dgatwood](#)
[Dima Deploy](#)
[Dinesh Raja](#)
[Dipen Panchasara](#)
[Dishant Kapadiya](#)
[dispute](#)
[Doc](#)
[Douglas Starnes](#)
[Duly Kinsky](#)
[Dunja Lalic](#)

Chapter 131
Chapter 2
Chapter 189
Chapters 55, 57 and 59
Chapters 107 and 151
Chapter 50
Chapters 2, 7, 38, 67 and 93
Chapter 67
Chapter 171
Chapter 75
Chapter 1
Chapter 24
Chapters 73, 107 and 134
Chapters 33, 43, 116 and 124
Chapters 65 and 198
Chapters 14 and 83
Chapter 23
Chapters 2, 7, 10, 14, 68, 104 and 119
Chapter 47
Chapter 135
Chapters 2 and 78
Chapters 2, 11 and 20
Chapter 73
Chapter 2
Chapter 134
Chapters 2, 11, 20 and 201
Chapter 137
Chapter 27
Chapters 2 and 61
Chapter 21
Chapters 2, 5, 23, 26, 36, 75 and 133
Chapters 2, 5, 20 and 38
Chapters 2, 11, 20 and 23
Chapter 1
Chapter 75
Chapters 14, 63 and 67
Chapter 67
Chapter 8
Chapter 203
Chapter 63
Chapter 86
Chapter 20
Chapters 34, 75 and 97
Chapters 8, 24, 27, 36, 73, 119 and 135

[杜赖·阿穆坦.H](#)
[埃弗拉伊姆·韦斯](#)
[埃隆·陈](#)
[Emptyless](#)
[Eonil](#)
[ERbittuu](#)
[埃里克](#)
[埃里克·戈达德](#)
[欧文](#)
[esthepiking](#)
[法比奥](#)
[法比奥·贝尔格 \(Fabio Berger\)](#)
[法希姆·帕尔卡尔 \(Fahim Parkar\)](#)
[法兰加尼 \(Faran Ghani\)](#)
[费利克斯](#)
[FelixSFD](#)
[Felix](#)
[gadu](#)
[乔治·李](#)
[ggrana](#)
[gkpln3](#)
[格洛芬戴尔](#)
[格雷格](#)
[gvuksic](#)
[H. M. 马德罗内](#)
[HaemEternal](#)
[halil_g](#)
[hankide](#)
[HariKrishnan.P](#)
[Harshal Bhavsar](#)
[Heberti Almeida](#)
[hgwhittle](#)
[霍萨姆·加里布](#)
[侯赛因·贝布迪·拉德](#)
[鱼人马](#)
[伊丹](#)
[idobn](#)
[idocode](#)
[ignotusverum](#)
[伊戈尔·比迪纽克](#)
[邪恶的火腿博士](#)

[伊马努·佩蒂](#)
[iphonic](#)
[伊尔凡](#)
[J.F](#)
[J.帕拉维奇尼](#)
[杰克·奈](#)
[雅各班克斯](#)
[雅科波·彭佐](#)
[杰克·伦泽](#)
[詹姆斯](#)

第194章
第171章
第7章
第76章
第24章
第73章
第2章
第1章和第14章
第2章
第2章和第198章
第127章
第34章
第2章
第118章
第2章
第1、2、5、8、11、22、34、37、54、68、117和198章
第125和187章
第2章和第24章
第159章
第2章
第204章
第14章
第64章
第2章
第122章
第143章
第202章
第2章
第115章
第23章和第76章
第8章
第7章、第20章和第28章
第107章
第76章
第74章
第177章
第2章
第52章
第89章
第53章
第11章
第2章
第20章和第23章
第121章
第107章和第114章
第207章
第1章
第2章
第74章
第63章
第23章
第1章和第5章

[Durai Amuthan.H](#)
[Efraim Weiss](#)
[ElonChan](#)
[Emptyless](#)
[Eonil](#)
[ERbittuu](#)
[Eric](#)
[Erik Godard](#)
[Erwin](#)
[esthepiking](#)
[Fabio](#)
[Fabio Berger](#)
[Fahim Parkar](#)
[Faran Ghani](#)
[Felix](#)
[FelixSFD](#)
[Fonix](#)
[gadu](#)
[George Lee](#)
[ggrana](#)
[gkpln3](#)
[Glorfindel](#)
[Greg](#)
[gvuksic](#)
[H. M. Madrone](#)
[HaemEternal](#)
[halil_g](#)
[hankide](#)
[HariKrishnan.P](#)
[Harshal Bhavsar](#)
[Heberti Almeida](#)
[hgwhittle](#)
[Hossam Ghareeb](#)
[Husein Behboodi Rad](#)
[Ichthyocentaur](#)
[Idan](#)
[idobn](#)
[idocode](#)
[ignotusverum](#)
[Igor Bidiniuc](#)
[il Malvagio Dottor Prosciutto](#)
[Imanou Petit](#)
[iphonic](#)
[Irfan](#)
[J.F](#)
[J.Paravicini](#)
[Jack Ngai](#)
[Jacobanks](#)
[Jacopo Penzo](#)
[Jake Runzer](#)
[JAL](#)
[James](#)

[詹姆斯·P](#)
[简·阿塔克](#)
[雅诺](#)
[吉米·詹姆斯](#)
[李金环](#)
[约翰·莱昂纳多](#)
[johnpenning](#)
[jojodmo](#)
[乔什·布朗](#)
[乔什·卡斯韦尔](#)
[约书亚](#)
[约书亚·小麦金农](#)
[JPetric](#)
[jrf](#)
[juanjo](#)
[Julian135](#)
[Justin](#)
[卡米尔·哈拉西莫维奇](#)
[Kampai](#)
[kamwysoc](#)
[keithbhunter](#)
[肯德尔·利斯特](#)
[凯文·迪特拉格利亚](#)
[基利安·科尔茨施](#)
[基雷因](#)
[基里特·莫迪](#)
[基里特·瓦格赫拉](#)
[Kof](#)
[近田·亚达夫](#)
[小川康介](#)
[考希克](#)
[克鲁纳尔](#)
[林纳斯·格法斯](#)
[洛西奥瓦蒂](#)
[迷失海洋的约书亚](#)
[卢卡·达尔贝尔蒂](#)
[路易斯](#)
[路易斯·恩里克·吉马良斯](#)
[第34章、第70章和第89章](#)
[卢克·帕特森](#)
[卢米亚尔克](#)
[玛迪ヅヅ](#)
[马赫什](#)
[马哈茂德·亚当](#)
[马尼](#)
[曼西·潘查尔](#)
[马克](#)
[mattblessed](#)
[马修·考利](#)
[马修·西曼](#)
[maxkonovalov](#)
[马尤里·R·塔拉维亚](#)
[MCMatan](#)

第5、74、76、85、98、119和122章
第133章
第67和111章
第8章
第70章
第27章
第2章
第2章、第5章和第20章
第2章和第133章
第76章
第2章、第5章、第34章、第83章、第94章、第98章和第152章
第2章
第182章
第1章
第2章和第67章
第41章
第66章
第18章、第19章、第62章和第89章
第1章
第7章
第67章
第63章
第38章和第86章
第88章
第2章
第22章、第70章、第74章和第76章
第112章
第30章和第34章
第134章
第68章
第44章、第48章和第109章
第69章
第29章
第1章
第73章和第179章
第2章、第14章、第20章、第23章和第63章
第14章
第2章
第2章
第34章、第72章和第92章
第68章
第2章、第8章和第116章
第106章
第70章和第100章
第38章
第11章
第1章
第149章
第8章、第23章、第29章、第73章和第80章
第47章和第103章
第54章

[James P](#)
[Jan ATAC](#)
[Jano](#)
[Jimmy James](#)
[Jinhuan Li](#)
[John Leonardo](#)
[johnpenning](#)
[jojodmo](#)
[Josh Brown](#)
[Josh Caswell](#)
[Joshua](#)
[Joshua J. McKinnon](#)
[JPetric](#)
[jrf](#)
[juanjo](#)
[Julian135](#)
[Justin](#)
[Kamil Harasimowicz](#)
[Kampai](#)
[kamwysoc](#)
[keithbhunter](#)
[Kendall Lister](#)
[Kevin DiTraglia](#)
[Kilian Koeltzsch](#)
[Kireyin](#)
[Kirit Modi](#)
[Kirit Vaghela](#)
[Kof](#)
[Konda Yadav](#)
[Kosuke Ogawa](#)
[Koushik](#)
[Krunal](#)
[LinusGeffarth](#)
[Losiowaty](#)
[lostAtSeaJoshua](#)
[Luca D'Alberti](#)
[Luis](#)
[Luiz Henrique Guimaraes](#)
[Luke Patterson](#)
[Lumialxk](#)
[Maddyヅヅ](#)
[Mahesh](#)
[Mahmoud Adam](#)
[MANI](#)
[Mansi Panchal](#)
[Mark](#)
[mattblessed](#)
[Matthew Cawley](#)
[Matthew Seaman](#)
[maxkonovalov](#)
[Mayuri R Talaviya](#)
[MCMatan](#)

Chapters 5, 74, 76, 85, 98, 119 and 122
Chapter 133
Chapters 67 and 111
Chapter 8
Chapter 70
Chapter 27
Chapter 2
Chapters 2, 5 and 20
Chapters 2 and 133
Chapter 76
Chapters 2, 5, 34, 83, 94, 98 and 152
Chapter 2
Chapter 182
Chapter 1
Chapters 2 and 67
Chapter 41
Chapter 66
Chapters 18, 19, 62 and 89
Chapter 1
Chapter 7
Chapter 67
Chapter 63
Chapters 38 and 86
Chapter 88
Chapter 2
Chapters 22, 70, 74 and 76
Chapter 112
Chapters 30 and 34
Chapter 134
Chapter 68
Chapters 44, 48 and 109
Chapter 69
Chapter 29
Chapter 1
Chapters 73 and 179
Chapters 2, 14, 20, 23 and 63
Chapter 14
Chapters 34, 70 and 89
Chapter 2
Chapter 2
Chapters 34, 72 and 92
Chapter 68
Chapters 2, 8 and 116
Chapter 106
Chapters 70 and 100
Chapter 38
Chapter 11
Chapter 1
Chapter 149
Chapters 8, 23, 29, 73 and 80
Chapters 47 and 103
Chapter 54

穆罕默德·易卜拉欣·哈桑 第2、3、8、9、11、13、16、20、21、24、25、27、37、41、45、50、51、54、59、71、8
梅胡尔·楚阿汉 第39、40、50、83、103、104、110、117、149、152、153、155和198章
梅胡尔·塔卡尔 第68和73章
梅尔特·布兰 第67章
米敦·MP 第93章
绿 第25和156章
穆罕默德·拉纳 第16章
莫谢 第2章
穆罗德里戈 第32章
Xcoder先生 第46、49、61、64和85章
mtso 第23、39和103章
穆阿祖德 第20和63章
穆罕默德·齐山 第8章
穆罕默德·佐海布·埃赫桑 第22章和第110章
纳伦德拉·潘迪 第20章、第22章、第75章、第85章、第100章、第102章和第115章
内特·李 第5章
内森 第198章
内森·莱维特 第2章
Nef10 第50章
内尔敏·塞希奇 第175章
尼克莱什·巴格迪亚 第176章
尼基塔·库尔廷 第22章
尼拉夫 第196章
尼拉夫·巴特 第52章
恩朱里 第2、11、24、91、92和98章
诺亚姆 第180章
NobodyNada 第129章
NSNoob 第5、20、22、23、67和73章
Nykholas 第73章
OhadM 第63、73、104、105、106和128章
Oleh Zayats 第26、108、174和188章
Ollie 第2章
奥努尔·图纳 第147章
奥特温·根茨 第41章
奥兹古尔 第67章
帕尔塞克 第110章
P. Pawluś 第5、34和60章
帕布莱罗斯 第20、50和65章
帕维尔·加蒂洛夫 第6章
帕维尔·古罗夫 第107章
pckill 第20章
彼得·德维斯 第20章
法尼·赛 第199章
pkc456 第173章
quant24 第103章
Quantm 第2章和第11章
布朗·拉达加斯特 第2章
拉胡尔 第12章、第17章、第31章、第85章和第200章
拉胡尔·维亚斯 第20章和第117章
拉姆库马尔·钦塔拉 第41章和第50章
regetskcob 第2章

Md. Ibrahim Hassan Chapters 2, 3, 8, 9, 11, 13, 16, 20, 21, 24, 25, 27, 37, 41, 45, 50, 51, 54, 59, 71, 81, 88, 125, 140, 146, 147, 151, 155, 157, 158, 160, 163, 164, 165, 166, 167, 168, 169, 170 and 183
Mehul Chuahan Chapters 39, 40, 50, 83, 103, 104, 110, 117, 149, 152, 153, 155 and 198
Mehul Thakkar Chapters 68 and 73
Mert Buran Chapter 67
Midhun MP Chapter 93
midori Chapters 25 and 156
Mohammad Rana Chapter 16
Moshe Chapter 2
mourodrigo Chapter 32
Mr. Xcoder Chapters 46, 49, 61, 64 and 85
mtso Chapters 23, 39 and 103
muazhud Chapters 20 and 63
Muhammad Zeeshan Chapter 8
Muhammad Zohaib Ehsan Chapters 22 and 110
Narendra Pandey Chapters 20, 22, 75, 85, 100, 102 and 115
Nate Lee Chapter 5
nathan Chapter 198
Nathan Levitt Chapter 2
Nef10 Chapter 50
Nermin Sehic Chapter 175
Nikhlesh Bagdiya Chapter 176
Nikita Kurtin Chapter 22
Nirav Chapter 196
Nirav Bhatt Chapter 52
njuri Chapters 2, 11, 24, 91, 92 and 98
Noam Chapter 180
NobodyNada Chapter 129
NSNoob Chapters 5, 20, 22, 23, 67 and 73
Nykholas Chapter 73
OhadM Chapters 63, 73, 104, 105, 106 and 128
Oleh Zayats Chapters 26, 108, 174 and 188
Ollie Chapter 2
Onur Tuna Chapter 147
Ortwin Gentz Chapter 41
ozgur Chapter 67
Pärserk Chapter 110
P. Pawluś Chapters 5, 34 and 60
pableiros Chapters 20, 50 and 65
Pavel Gatilov Chapter 6
Pavel Gurov Chapter 107
pckill Chapter 20
Peter DeWeese Chapter 20
Phani Sai Chapter 199
pkc456 Chapter 173
quant24 Chapter 103
Quantm Chapters 2 and 11
Radagast the Brown Chapter 2
Rahul Chapters 12, 17, 31, 85 and 200
Rahul Vyas Chapters 20 and 117
Ramkumar chintala Chapters 41 and 50
regetskcob Chapter 2

[雷尼尔·梅利安](#)

第11章、第23章、第34章、第68章和第123章
第23章和第29章
第64章
第101章
第11章和第85章
第8章
第38章
第141章
第5章、第24章和第76章
第8章
第25章
第161章和第162章
第8章和第75章
第90章
第23章、第67章和第73章
第87章
第23章
第34章
第8、23、73和98章
第29、107、110和135章
第20、30和34章
第99章
第174章
第14章
第1、2、8、22、73、76、80、107、117、130、131、133、136、138、139、140、
、185和186章
第2章
第132章
第127章
第22、34和61章
第33章
第97章
第14章和第118章
第125章
第64章
第114章
第172章
第2章
第21章
第2章
第40章
第58、181和184章
第14、61、63和64章
第120章
第8、14、27、29和97章
第37章
第2、8、11、29、41和136章
第1、2、11、14、22、24、26、27、28、29、34、36、38、53、63、64、67和89章
第2、24、34、65、103和117章
第14和152章
第74章

[雷克斯](#)

[理查德·阿什](#)

[rigdonmr](#)

[罗布](#)

[rob180](#)

[罗德里戈·德·圣地亚哥](#)

[rohit90](#)

[罗兰·基索姆](#)

[罗纳克·查尼亚拉](#)

[Ruby](#)

[赛义德](#)

[sage444](#)

[萨赫布·罗伊](#)

[萨莉](#)

[萨姆·费舍尔](#)

[萨默·穆拉德](#)

[塞缪尔·斯宾塞](#)

[塞缪尔·特费拉](#)

[samwize](#)

[桑迪](#)

[sanman](#)

[sasquatch](#)

[satheeshwaran](#)

[萨米尔·沙阿](#)

[塞斯林](#)

[赛义德·帕尔萨·内沙伊](#)

[沙哈布丁·万西瓦拉](#)

[沙杜尔](#)

[鲨鱼诱饵胡哈哈](#)

[shim](#)

[施里坎特·K](#)

[悉达特·苏尼尔](#)

[simple_code](#)

[skyline75489](#)

[slxl](#)

[SM18](#)

[SNarula](#)

[solidcell](#)

[SpaceDog](#)

[Sravan](#)

[Srinija](#)

[StackUnderflow](#)

[史蒂夫·莫泽 \(Steve Moser\)](#)

[subv3rsion](#)

[苏贾尼娅](#)

[苏贾伊](#)

[苏尼尔·夏尔马](#)

[苏拉格奇](#)

[苏山特·贾格塔普](#)

[塔马罗斯](#)

[塔伦·西拉](#)

[Reinier Melian](#)

[Rex](#)

[Richard Ash](#)

[rigdonmr](#)

[Rob](#)

[rob180](#)

[Rodrigo de Santiago](#)

[rohit90](#)

[Roland Keesom](#)

[Ronak Chaniyara](#)

[Ruby](#)

[Saeed](#)

[sage444](#)

[Saheb Roy](#)

[Sally](#)

[Sam Fischer](#)

[Samer Murad](#)

[Samuel Spencer](#)

[Samuel Teferra](#)

[samwize](#)

[Sandy](#)

[sanman](#)

[sasquatch](#)

[satheeshwaran](#)

[Saumil Shah](#)

[Seslyn](#)

[Seyyed Parsa Neshaei](#)

[Shahabuddin Vansiwala](#)

[Shardul](#)

[SharkbaitWhahaha](#)

[shim](#)

[Shrikant K](#)

[Siddharth Sunil](#)

[simple_code](#)

[skyline75489](#)

[slxl](#)

[SM18](#)

[SNarula](#)

[solidcell](#)

[SpaceDog](#)

[Sravan](#)

[Srinija](#)

[StackUnderflow](#)

[Steve Moser](#)

[subv3rsion](#)

[Sujania](#)

[Sujay](#)

[Sunil Sharma](#)

[Suragch](#)

[sushant jagtap](#)

[Tamarous](#)

[Tarun Seera](#)

Chapters 11, 23, 34, 68 and 123
Chapters 23 and 29

Chapter 64

Chapter 101

Chapters 11 and 85

Chapter 8

Chapter 38

Chapter 141

Chapters 5, 24 and 76

Chapter 8

Chapter 25

Chapters 161 and 162

Chapters 8 and 75

Chapter 90

Chapters 23, 67 and 73

Chapter 87

Chapter 23

Chapters 7, 11, 14 and 82

Chapters 72, 73 and 98

Chapter 34

Chapters 8, 23, 73 and 98

Chapters 29, 107, 110 and 135

Chapters 20, 30 and 34

Chapter 99

Chapter 174

Chapter 14

Chapters 1, 2, 8, 22, 73, 76, 80, 107, 117, 130, 131, 133, 136, 138, 139, 140,
185 and 186

Chapter 2

Chapter 132

Chapter 127

Chapters 22, 34 and 61

Chapter 33

Chapter 97

Chapters 14 and 118

Chapter 125

Chapter 64

Chapter 114

Chapter 172

Chapter 2

Chapter 21

Chapter 2

Chapter 40

Chapters 58, 181 and 184

Chapters 14, 61, 63 and 64

Chapter 120

Chapters 8, 14, 27, 29 and 97

Chapter 37

Chapters 2, 8, 11, 29, 41 and 136

Chapters 1, 2, 11, 14, 22, 24, 26, 27, 28, 29, 34, 36, 38, 53, 63, 64, 67 and 89

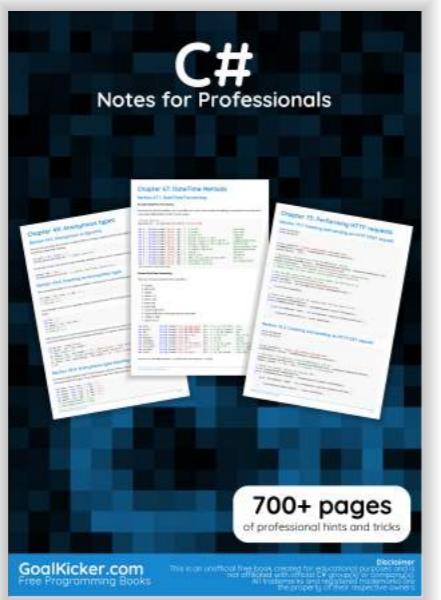
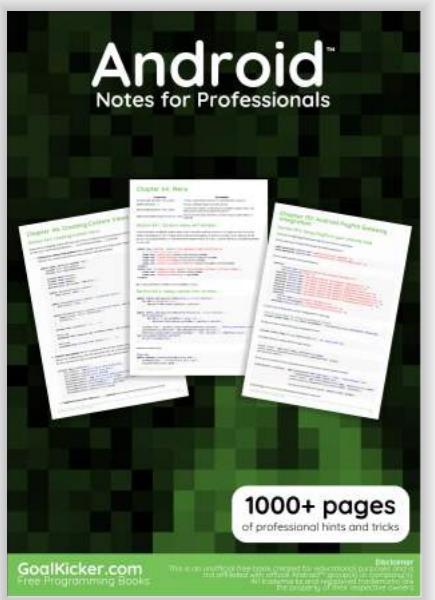
Chapters 2, 24, 34, 65, 103 and 117

Chapters 14 and 152

Chapter 74

塔沃·梅塞普	第39章	Tarvo Mäesepp	Chapter 39
塔西纳里	第36章	tassinari	Chapter 36
特贾·南达穆里	第53章	Teja Nandamuri	Chapter 53
tfrank377	第72章	tfrank377	Chapter 72
tharkay	第34、73、114和123章	tharkay	Chapters 34, 73, 114 and 123
The Curry Man	第8章	The Curry Man	Chapter 8
田 (Tien)	第86和150章	Tien	Chapters 86 and 150
Tiko	第205章	Tiko	Chapter 205
tilo	第107章	tilo	Chapter 107
蒂姆	第70章和第126章	Tim	Chapters 70 and 126
蒂姆·埃本泽尔	第70章	Tim Ebenezer	Chapter 70
timbroder	第63章	timbroder	Chapter 63
tktsubota	第7章	tktsubota	Chapter 7
tobeiosdev	第73章和第147章	tobeiosdev	Chapters 73 and 147
托米·C.	第23章和第92章	Tommie C.	Chapters 23 and 92
乌玛	第195章	Uma	Chapter 195
撤销	第2、72、85和146章	Undo	Chapters 2, 72, 85 and 146
user1374	第151和152章	user1374	Chapters 151 and 152
user3480295	第2、8、22和67章	user3480295	Chapters 2, 8, 22 and 67
user3760892	第76章	user3760892	Chapter 76
维克多·M	第47章	Victor M	Chapter 47
维格南	第1章	Vignan	Chapter 1
维克托·辛科	第5章、第20章、第23章和第89章	Viktor Simkó	Chapters 5, 20, 23 and 89
维尼特·乔德哈里	第99章	Vineet Choudhary	Chapter 99
维诺德·库马尔	第191章	Vinod Kumar	Chapter 191
维韦克·莫尔卡尔	第42章	Vivek Molkar	Chapter 42
vp2698	第4章	vp2698	Chapter 4
wdywayne	第84章	wdywayne	Chapter 84
william205	第2、5、7、10、11、20、24、26、28、30、34、36、64、72、76、89和103章	william205	Chapters 2, 5, 7, 10, 11, 20, 24, 26, 28, 30, 34, 36, 64, 72, 76, 89 and 103
WMios	第2、5、23、24、26、34、61、94和101章	WMios	Chapters 2, 5, 23, 24, 26, 34, 61, 94 and 101
金刚狼	第7章	Wolverine	Chapter 7
氙	第7章	Xenon	Chapter 7
亚格内什·多巴里亚	第2、23和63章	Yagnesh Dobariya	Chapters 2, 23 and 63
叶夫亨·杜比宁	第116章	Yevhen Dubinin	Chapter 116
约格什·瓦德瓦	第95、208和209章	yogesh wadhwa	Chapters 95, 208 and 209
东安	第67章	Đông An	Chapter 67
欧阳大哥	第193章	欧阳大哥	Chapter 193

你可能也喜欢



You may also like

