为了研发一个无侵入的 iOS 页面停留时长采集 SDK，您可以采用**AOP（面向切面编程）**的方法，结合 **Runtime** 动态替换 `UIViewController` 的生命周期方法，实现无侵入的埋点方案。

下面是一个全面的 SDK 实现步骤，主要包括 Runtime 方法交换和自动埋点功能，确保不需要开发者在每个 `UIViewController` 中手动添加代码。

1. SDK 实现思路

- **Runtime 交换方法**：使用 Objective-C Runtime 替换系统的 `viewDidAppear:` 和 `viewWillDisappear:` 方法，以插入埋点逻辑。
- **自动采集页面信息**：通过 `UIViewController` 的类名自动识别页面，不需要开发者传入页面标识。
- **数据上报**：封装网络请求，将采集的页面停留时长发送到服务器。
- **无侵入**：开发者无需手动集成代码，SDK 自动管理所有页面的时长统计。

2. SDK 代码实现

2.1 `PageTrackingManager.h`

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

@interface PageTrackingManager : NSObject

+ (instancetype)sharedInstance;
- (void)startTracking; // 启动页面时长埋点监控

@end
```

2.2 `PageTrackingManager.m`

```
#import "PageTrackingManager.h"
#import <objc/runtime.h>

@implementation PageTrackingManager

+ (instancetype)sharedInstance {
    static PageTrackingManager *sharedInstance = nil;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        sharedInstance = [[PageTrackingManager alloc] init];
    });
    return sharedInstance;
}

- (void)startTracking {
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        [self swizzleUIViewControllerLifecycle];
    });
}

- (void)swizzleUIViewControllerLifecycle {
```

```
    Class class = [UIViewController class];

    // 替换 viewDidAppear
    SEL originalSelector = @selector(viewDidAppear:);
    SEL swizzledSelector = @selector(tracked_viewDidAppear:);
    [self swizzleMethodInClass:class originalSelector:originalSelector
swizzledSelector:swizzledSelector];

    // 替换 viewWillDisappear
    SEL originalDisappearSelector = @selector(viewWillDisappear:);
    SEL swizzledDisappearSelector = @selector(tracked_viewWillDisappear:);
    [self swizzleMethodInClass:class originalSelector:originalDisappearSelector
swizzledSelector:swizzledDisappearSelector];
}

- (void)swizzleMethodInClass:(Class)class originalSelector:(SEL)originalSelector
swizzledSelector:(SEL)swizzledSelector {
    Method originalMethod = class_getInstanceMethod(class, originalSelector);
    Method swizzledMethod = class_getInstanceMethod(class, swizzledSelector);

    BOOL didAddMethod = class_addMethod(class,
                                        originalSelector,

 method_getImplementation(swizzledMethod),
                                        method_getTypeEncoding(swizzledMethod));

    if (didAddMethod) {
        class_replaceMethod(class,
                            swizzledSelector,
                            method_getImplementation(originalMethod),
                            method_getTypeEncoding(originalMethod));
    } else {
        method_exchangeImplementations(originalMethod, swizzledMethod);
    }
}

@end
```

2.3 UIViewController 类别（Category）实现替换逻辑

```
#import "PageTrackingManager.h"
#import <objc/runtime.h>

@implementation UIViewController (Tracking)

- (void)tracked_viewDidAppear:(BOOL)animated {
    [self tracked_viewDidAppear:animated];  // 调用原方法 (由于方法交换，此处是原始方
法)

    // 记录页面进入时间
    NSString *pageName = NSStringFromClass([self class]);
    NSLog(@"Enter page: %@", pageName);
    objc_setAssociatedObject(self, @selector(tracked_viewDidAppear:), [NSDate
date], OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (void)tracked_viewWillDisappear:(BOOL)animated {
```

```
    [self tracked_viewWillDisappear:animated];  // 调用原方法

    // 获取页面进入时间并计算停留时长
    NSDate *enterTime = objc_getAssociatedObject(self,
@selector(tracked_viewDidAppear:));
    if (enterTime) {
        NSTimeInterval stayDuration = [[NSDate date]
timeIntervalSinceDate:enterTime];
        NSString *pageName = NSStringFromClass([self class]);
        NSLog(@"Exit page: %@, Stay Duration: %.2f seconds", pageName,
stayDuration);

        // 上报数据
        [self reportPageStayDuration:pageName duration:stayDuration];
    }
}

- (void)reportPageStayDuration:(NSString *)pageName duration:
(NSTimeInterval)duration {
    // 模拟网络请求，将数据发送到后台
    NSLog(@"Reporting: %@ stayed for %.2f seconds", pageName, duration);

    // 此处可以封装实际的网络请求，将 pageName 和 duration 发送到服务器
    // 使用 NSURLSession 或者其他第三方网络库，比如 AFNetworking
}

@end
```

2.4 启动 SDK

在应用启动时调用 SDK 的 `startTracking` 方法，开始埋点监控。

```
#import "PageTrackingManager.h"

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions {
    // 启动页面停留时长埋点 SDK
    [[PageTrackingManager sharedInstance] startTracking];
    return YES;
}
```

3. 考虑因素

- **App 切换前后台的处理**：可以通过监听 `UIApplicationWillResignActiveNotification` 和 `UIApplicationDidBecomeActiveNotification` 来判断 App 进入后台或返回前台，调整停留时长的统计逻辑。
- **崩溃处理**：当 App 意外崩溃时，页面停留时长可能无法被正确上报，可以结合崩溃日志工具来确保数据的完整性。
- **性能优化**：确保方法交换不会影响应用性能，通过异步网络请求上报数据来减少 UI 线程的压力。

4. 总结

通过 Runtime 方法交换的无侵入方式，SDK 可以自动采集 iOS 应用的页面停留时长，开发者无需在每个页面中手动调用代码。SDK 具备很好的扩展性，可以方便地加入更多的埋点采集功能，如按钮点击、页面滑动等。