为了模拟神策（Sensors Analytics）SDK 的实现逻辑，特别是在无侵入的埋点采集方面，可以借鉴神策的常用方法，包括：

1. **AOP 面向切面编程**：通过动态交换 `UIViewController` 和 `UIControl` 等关键类的方法，自动埋点，降低开发者的使用成本。
2. **自动追踪事件**：页面浏览、点击事件、应用启动、退出等均为自动采集事件，不需要手动埋点。
3. **事件数据管理**：通过本地持久化或内存管理来存储埋点数据，统一在合适时机发送到服务端。
4. **网络请求管理**：将埋点数据封装为 JSON 并通过网络请求上报至服务器。

下面的示例将模拟神策的实现，主要功能包括页面停留时长采集、按钮点击事件追踪、数据持久化和上报逻辑。

1. 模拟神策的核心实现逻辑

1.1 `SensorsAnalyticsManager.h`

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

@interface SensorsAnalyticsManager : NSObject

+ (instancetype)sharedInstance;

// 开始 SDK 监控
- (void)startTracking;

// 上报事件
- (void)trackEvent:(NSString *)eventName withProperties:(NSDictionary
*)properties;

@end
```

1.2 `SensorsAnalyticsManager.m`

```objc
#import "SensorsAnalyticsManager.h"
#import <objc/runtime.h>
#import <UIKit/UIKit.h>

@implementation SensorsAnalyticsManager

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

- (void)startTracking {
    // 启动页面和点击事件的埋点追踪
    [self swizzleUIViewControllerLifecycle];
    [self swizzleUIControlEventTracking];
}
```

```
#pragma mark - 页面停留时长追踪

- (void)swizzleUIViewControllerLifecycle {
    Class class = [UIViewController class];

    SEL originalAppearSelector = @selector(viewDidAppear:);
    SEL swizzledAppearSelector = @selector(sa_viewDidAppear:);
    [self swizzleMethodInClass:class originalSelector:originalAppearSelector
swizzledSelector:swizzledAppearSelector];

    SEL originalDisappearSelector = @selector(viewWillDisappear:);
    SEL swizzledDisappearSelector = @selector(sa_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
```

1.3 UIViewController 类别实现页面埋点逻辑

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

@implementation UIViewController (SensorsAnalytics)

- (void)sa_viewDidAppear:(BOOL)animated {
    [self sa_viewDidAppear:animated];  // 调用原始的 viewDidAppear

    // 记录进入页面的时间
    NSString *pageName = NSStringFromClass([self class]);
    NSLog(@"Enter page: %@", pageName);
    objc_setAssociatedObject(self, @selector(sa_viewDidAppear:), [NSDate date],
OBJC_ASSOCIATION_RETAIN_NONATOMIC);

    // 发送页面浏览事件
    [[SensorsAnalyticsManager sharedInstance] trackEvent:@"PageView"
```

```
                                                  withProperties:@{@"pageName":
pageName}];
}

- (void)sa_viewWillDisappear:(BOOL)animated {
    [self sa_viewWillDisappear:animated];  // 调用原始的 viewWillDisappear

    // 计算页面停留时长
    NSDate *enterTime = objc_getAssociatedObject(self,
@selector(sa_viewDidAppear:));
    if (enterTime) {
        NSTimeInterval stayDuration = [[NSDate date]
timeIntervalSinceDate:enterTime];
        NSString *pageName = NSStringFromClass([self class]);
        NSLog(@"Exit page: %@, Stay Duration: %.2f seconds", pageName,
stayDuration);

        // 发送页面停留时长事件
        [[SensorsAnalyticsManager sharedInstance] trackEvent:@"PageStay"
                                                  withProperties:@{@"pageName":
pageName, @"duration": @(stayDuration)}];
    }
}

@end
```

1.4 追踪 UIControl 点击事件

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

@implementation UIControl (SensorsAnalytics)

+ (void)load {
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        [self swizzleControlEvents];
    });
}

+ (void)swizzleControlEvents {
    SEL originalSelector = @selector(sendAction:to:forEvent:);
    SEL swizzledSelector = @selector(sa_sendAction:to:forEvent:);
    Method originalMethod = class_getInstanceMethod([self class],
originalSelector);
    Method swizzledMethod = class_getInstanceMethod([self class],
swizzledSelector);

    method_exchangeImplementations(originalMethod, swizzledMethod);
}

- (void)sa_sendAction:(SEL)action to:(id)target forEvent:(UIEvent *)event {
    [self sa_sendAction:action to:target forEvent:event];  // 调用原始方法

    // 发送点击事件埋点
    NSString *actionName = NSStringFromSelector(action);
    NSString *controlName = NSStringFromClass([self class]);
    [[SensorsAnalyticsManager sharedInstance] trackEvent:@"UIControlClick"
```

```
                                                    withProperties:@{@"control":
controlName, @"action": actionName}];
}

@end
```

1.5 事件数据的上报和存储

神策的 SDK 会先将事件存储在本地，等积累到一定量或者满足某些条件时再统一上报服务器。

```
@implementation SensorsAnalyticsManager

// 埋点事件追踪方法
- (void)trackEvent:(NSString *)eventName withProperties:(NSDictionary
*)properties {
    NSMutableDictionary *eventData = [NSMutableDictionary dictionary];
    eventData[@"eventName"] = eventName;
    eventData[@"properties"] = properties;
    eventData[@"timestamp"] = @([[NSDate date] timeIntervalSince1970]);

    // 模拟将事件存储到本地
    [self saveEvent:eventData];

    // 如果达到一定数量，或满足条件，触发上报逻辑
    if ([self shouldUploadEvents]) {
        [self uploadEvents];
    }
}

- (void)saveEvent:(NSDictionary *)eventData {
    // 模拟本地存储，可以使用文件、数据库等方式持久化
    NSLog(@"Saved event: %@", eventData);
}

- (BOOL)shouldUploadEvents {
    // 模拟条件，比如超过 10 条事件，或者达到时间阈值
    return YES;
}

- (void)uploadEvents {
    // 模拟事件上报，可以使用 NSURLSession 或其他网络库
    NSLog(@"Uploading events...");
}

@end
```

2. 总结

这个实现模拟了神策埋点 SDK 的核心逻辑，通过 Runtime 方法交换实现无侵入的页面停留时长采集和 UI 事件追踪。埋点数据会本地存储，并根据条件统一上报服务器。在实际生产环境中，还需要处理更多的边界情况，如网络失败重试、数据加密、日志管理等，确保 SDK 的健壮性。