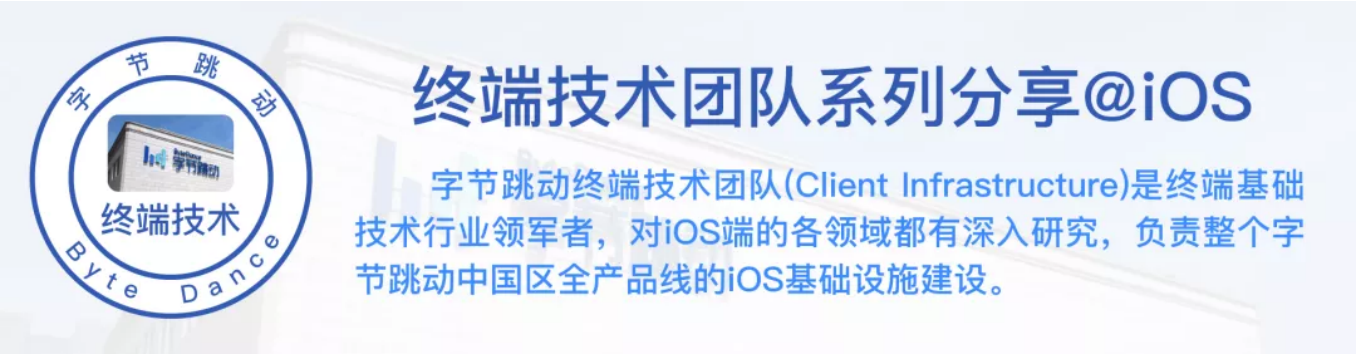
无埋点核心技术：iOS Hook在字节的实践经验

段文斌 字节跳动技术团队 今天



前言

众所周知，字节跳动的推荐在业内处于领先水平，而精确的推荐离不开大量埋点，常见的埋点采集方案是在响应用户行为操作的路径上进行埋点。但是由于 App 通常会有比较多界面和操作路径，主动埋点的维护成本就会非常大。所以行业的做法是无埋点，而无埋点实现需要 AOP 编程。

一个常见的场景，比如想在 `UIViewController` 出现和消失的时刻分别记录时间戳用于统计页面展现的时长。要达到这个目标有很多种方法，但是 AOP 无疑是最简单有效的方法。Objective-C 的 Hook 其实也有很多种方式，这里以 Method Swizzle 给个示例。

```
@interface UIViewController (MyHook)

@end

@implementation UIViewController (MyHook)

+ (void)load {
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        /// 常规的 Method Swizzle封装
        swizzleMethods(self, @selector(viewDidAppear:), @selector(my_viewDidAppear:));
        /// 更多Hook
    });
}

- (void)my_viewDidAppear:(BOOL)animated {
  /// 一些Hook需要的逻辑

  /// 这里调用Hook后的方法，其实现其实已经是原方法了。
```

```
  [self my_viewDidAppear: animated];
}

@end
```

接下来我们探讨一个具体场景：

`UICollectionView` 或者 `UITableView` 是 iOS 中非常常用的列表 UI 组件，其中列表元素的点击事件回调是通过 `delegate` 完成的。这里以 `UICollectionView` 为例， `UICollectionView` 的 `delegate` ，有个方法声明， `collectionView:didSelectItemAtIndexPath:` ，实现这个方法我们就可以给列表元素添加点击事件。

我们的目标是 Hook 这个 delegate 的方法，在点击回调的时候进行额外的埋点操作。

方案迭代

方案 1 Method Swizzle

通常情况下，Method Swizzle 可以满足绝大部分的 AOP 编程需求。因此首次迭代，我们直接使用 Method Swizzle 来进行 Hook。

```
@interface UICollectionView (MyHook)

@end

@implementation UICollectionView (MyHook)

// Hook, setMyDelegate:和setDelegate:交换过
- (void)setMyDelegate:(id)delegate {
    if (delegate != nil) {
        /// 常规Method Swizzle
        swizzleMethodsXXX(delegate, @selector(collectionView:didSelectItemAtIndexPath:), self, @select

    }

    [self setMyDelegate:nil];
}

- (void)my_collectionView:(UICollectionView *)ccollectionView didSelectItemAtIndexPath:(NSIndexPath *)
  /// 一些Hook需要的逻辑
```

```
    /// 这里调用Hook后的方法，其实现其实已经是原方法了。
    [self my_collectionView:ccollectionView didSelectItemAtIndexPath:index];
}

@end
```

我们把这个方案集成到今日头条 App 里面进行测试验证，发现没法办法验证通过。

主要原因今日头条 App 是一个庞大的项目，其中引入了非常多的三方库，比如 IGListKit 等，这些三方库通常对 `UICollectionView` 的使用都进行了封装，而这些封装，恰恰导致我们不能使用常规的 Method Swizzle 来 Hook 这个 delegate。直接的原因总结有以下两点：

1. `setDelegate` 传入的对象不是实现 `UICollectionViewDelegate` 协议的那个对象

img

如图示， `setDelegate` 传入的是一个代理对象 proxy，proxy 引用了实际的实现 `UICollectionViewDelegate` 协议的 `delegate` ，proxy 实际上并没有实现 `UICollectionViewDelegate` 的任何一个方法，它把所有方法都转发给实际的 `delegate` 。这种情况下，我们不能直接对 proxy 进行 Method Swizzle

1. 多次 `setDelegate`

img

在上述图例中，使用方存在连续调用两次 `setDelegate` 的情况，第一次是真实 `delegate` ， 第二次是 `proxy` ， 我们需要区别对待。

代理模式和 NSProxy 介绍

使用 proxy 对原对象进行代理，在处理完额外操作之后再调用原对象，这种模式称为代理模式。而 Objective-C 中要实现代理模式，使用 NSProxy 会比较高效。详细内容参考下列文章。

- 代理模式
- NSProxy 使用

这里面 `UICollectionView` 的 `setDelegate` 传入的是一个 `proxy` 是非常常见的操作，比如 IGListKit，同时 App 基于自身需求，也有可能会做这一层封装。

在 `UICollectionView` 的 `setDelegate` 的时候，把 `delegate` 包裹在 `proxy` 中，然后把 proxy 设置给 `UICollectionView` ,使用 `proxy` 对 `delegate` 进行消息转发。

img

方案 2 使用代理模式

方案 1 已经无法满足我们的需求了，我们考虑到既然对 `delegate` 进行代理是一种常规操作，我们何不也使用代理模式，对 `proxy` 再次代理。

代码实现

- 先 Hook `UICollectionView` 的 `setDelegate` 方法
- 代理 `delegate`

简单的代码示意如下

```
/// 完整封装了一些常规的消息转发方法
@interface DelegateProxy : NSProxy

@property (nonatomic, weak, readonly) id target;

@end

/// 为 CollectionView delegate转发消息的proxy
@interface BDCollectionViewDelegateProxy : DelegateProxy

@end

@implementation BDCollectionViewDelegateProxy <UICollectionViewDelegate>

- (void)collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath *)inde
    //track event here
    if ([self.target respondsToSelector:@selector(collectionView:didSelectItemAtIndexPath:)]) {
        [self.target collectionView:collectionView didSelectItemAtIndexPath:indexPath];

    }
}

- (BOOL)bd_isCollectionViewTrackerDecorator {
    return YES;
}

// 还有其他的消息转发的代码 先忽略
- (BOOL)respondsToSelector:(SEL)aSelector {
    if (aSelector == @selector(bd_isCollectionViewTrackerDecorator)) {
        return YES;
    }

    return [self.target respondsToSelector:aSelector];
}


@end

@interface UICollectionView (MyHook)

@end

@implementation UICollectionView (MyHook)

- (void) setDd_TrackerProxy:(BDCollectionViewDelegateProxy *)object {
    objc_setAssociatedObject(self, @selector(bd_TrackerProxy), object, OBJC_ASSOCIATION_RETAIN_NONATOM
```

```
}

- (BDCollectionViewDelegateProxy *) bd_TrackerProxy {
    BDCollectionViewDelegateProxy *bridge = objc_getAssociatedObject(self, @selector(bd_TrackerProxy))

    return bridge;
}

// Hook, setMyDelegate:和setDelegate:交换过了
- (void)setMyDelegate:(id)delegate {
    if (delegate == nil) {
        [self setMyDelegate:delegate];
        return
    }

    // 不会释放，不重复设置
    if ([delegate respondsToSelector:@selector(bd_isCollectionViewTrackerDecorator)]) {
         [self setMyDelegate:delegate];
         return;
    }

    BDCollectionViewDelegateProxy *proxy = [[BDCollectionViewDelegateProxy alloc] initWithTarget:deleg
    [self setMyDelegate:proxy];
    self.bd_TrackerProxy = proxy;

}

@end
```

模型

下图实线表示强引用，虚线表示弱引用。

情况一

如果使用方没有对 `delegate` 进行代理，而我们使用代理模式

- `UICollectionView` ，其 `delegate` 指针指向 DelegateProxy
- DelegateProxy，被 UICollectionView 用 runtime 的方式强引用，其 target 弱引用真实 Delegate

img

情况二

如果使用方也对 `delegate` 进行代理，我们使用代理模式

- 我们只需要保证我们的 DelegateProxy 处于代理链中的一环即可

img

从这里我们可以看出，代理模式有很好的扩展性，它允许代理链不断嵌套，只要我们都遵循代理模式的原则即可。

到这里，我们的方案已经在今日头条 App 上测试通过了。但是事情远还没有结束。

踩坑之旅

目前的还算比较可以，但是也不能完全避免问题。这里其实不仅仅是**UICollectionView 的 delegate，包括:**

- **UIWebView**
- **WKWebView**
- **UITableView**
- **UICollectionView**
- **UIScrollView**
- **UIActionSheet**
- **UIAlertView**

我们都采用相同的方法来进行 Hook。**同时我们将方案封装一个 SDK 对外提供，以下统称为 MySDK。**

第一次踩坑

某客户接入我们的方案之后，在集成过程中反馈有必现 Crash，下面详细介绍一下这一次踩坑的经历。

堆栈信息

重点信息是 `[UIWebView webView:decidePolicyForNavigationAction:request:frame:decisionListener:]` 。

```
Thread 0 Crashed:

0   libobjc.A.dylib   0x000000018198443c objc_msgSend + 28

1   UIKit             0x000000018be05b4c -[UIWebView webView:decidePolicyForNavigationAction:request:f

2   CoreFoundation    0x0000000182731cd0 __invoking___ + 144

3   CoreFoundation    0x000000018261056c -[NSInvocation invoke] + 292

4   CoreFoundation    0x000000018261501c -[NSInvocation invokeWithTarget:] + 60

5   WebKitLegacy      0x000000018b86d654 -[_WebSafeForwarder forwardInvocation:] + 156
```

从堆栈信息不难判断出 crash 原因是 UIWebView 的 delegate 野指针，那为啥出现野指针呢?

这里先说明一下 crash 的直接原因，然后再来具体分析为什么就出现了问题。

1. MySDK 对 setDelegate 进行了 Hook
2. 客户也对 setDelegate 进行了 Hook
3. 先执行 MySDK 的 Hook 逻辑调用，然后执行客户的 Hook 逻辑调用

客户 Hook 的代码

```
@interface UIWebView (JSBridge)

@end

@implementation UIWebView (JSBridge)

- (void)setJsBridge:(id)object {
    objc_setAssociatedObject(self, @selector(jsBridge), object, OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (WebViewJavascriptBridge *)jsBridge {
    WebViewJavascriptBridge *bridge = objc_getAssociatedObject(self, @selector(jsBridge));
    return bridge;
}

+ (void)load {
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        swizzleMethods(self, @selector(setDelegate:), @selector(setJSBridgeDelegate:));
        swizzleMethods(self, @selector(initWithFrame:), @selector(initJSWithFrame:));
    });

}

- (instancetype)initJSWithFrame:(CGRect)frame {
    self = [self initJSWithFrame:frame];
    if (self) {
        WebViewJavascriptBridge *bridge = [WebViewJavascriptBridge bridgeForWebView:self];
        [self setJsBridge:bridge];
    }
    return self;
}

/// webview.delegate = xxx 会被调用多次且传入的对象不一样
- (void)setJSBridgeDelegate:(id)delegate {
    WebViewJavascriptBridge *bridge = self.jsBridge;
    if (delegate == nil || bridge == nil) {
        [self setJSBridgeDelegate:delegate];
    } else if (bridge == delegate) {
        [self setJSBridgeDelegate:delegate];
    } else {
        /// 第一次进入这里传入 bridge
        /// 第二次进入这里传入一个delegate
        if (![delegate isKindOfClass:[WebViewJavascriptBridge class]]) {
            [bridge setWebViewDelegate:delegate];
            /// 下面这一行代码是客户缺少的
            /// fix with this
            [self setJSBridgeDelegate:bridge];
        } else {
            [self setJSBridgeDelegate:delegate];
        }
```

```
    }
}

@end
```

MySDK Hook 代码

```
@interface UIWebView (MyHook)

@end

@implementation UIWebView (MyHook)

// Hook, setWebViewDelegate:和setDelegate:交换过
- (void)setWebViewDelegate:(id)delegate {
    if (delegate == nil) {
        [self setWebViewDelegate:delegate];
    }
    BDWebViewDelegateProxy *proxy = [[BDWebViewDelegateProxy alloc] initWithTarget:delegate];
    self.bd_TrackerDecorator = proxy;
    [self setWebViewDelegate:proxy];
}

@end
```

野指针原因

UIWebView 有两次调用 setDelegate 方法，第一次是传的 WebViewJavascriptBridge，第二次传的另一个实际的 WebViewDelegate。暂且称第一次传了 bridge 第二次传了实际上的 delegate。

1. 第一次调用，MySDK Hook 的时候会用 DelegateProxy 包装住 bridge，所有方法通过 DelegateProxy 转发到 bridge，这里传给 `setJSBridgeDelegate:(id)delegate` 的 delegate 实际上是 DelegateProxy**而非 bridge**。

img

这里需要注意，UIWebView 的 delegate 指向 DelegateProxy 是客户给设置上的，且这个属性 **assign 而非 weak，这个 assign 很关键，assigin 在对象释放之后不会自动变为 nil。**

1. 第二次调用，MySDK Hook 的时候会用新的 DelegateProxy 包装住 delegate 也就是 WebViewDelegate，这个时候 MySDK 的逻辑是把新的 DelegateProxy 给强引用中，老的 DelegateProxy 就失去了强引用因此释放了。

img

此时的状态如果不做任何处理，当前状态就如图示：

- delegate 指向已经释放的 DelegateProxy，野指针
- UIWebview 触发回调就导致 crash

修复方法

如果补上那一句， `setJSBridgeDelegate:(id)delegate` 在判断了 delegate 不是 bridge 之后，把 UIWebView 的 delegate 设置为 bridge 就可以完成了。

注释中 fix with this 下一行代码

修复后模型如下图

img

总结

使用 Proxy 的方式虽然也可以解决一定的问题，但是也需要使用方遵循一定的规范，要意识到第三方 SDK 也可能 `setDelegate` 进行 Hook，也可能使用 Proxy

第二次踩坑

先补充一些参考资料

- RxCocoa 源码参考 https://github.com/ReactiveX/RxSwift
- rxcocoa 学习-DelegateProxy

RxCocoa 也使用了代理模式，对 delegate 进行了代理，按道理应该没有问题。但是 RxCocoa 的实现有点出入。

RxCocoa

img

如果单独只使用了**RxCocoa**的方案，和方案是一致，也就不会有任何问题。

RxCocoa+MySDK

img

RxCocoa+MySDK 之后，变成这样子。UICollectionView 的 delegate 直接指向谁在于谁调用的 `setDelegate` 方法后调。

理论也应该没有问题，就是引用链多一个 poxy 包装而已。但是实际上有两个问题。

问题 1

RxCocoa 的 delegate 的 get 方法命中 assert

```
//  UIScrollView+Rx.swift
extension Reactive where Base: UIScrollView {
    public var delegate: DelegateProxy<UIScrollView, UIScrollViewDelegate> {
        return RxScrollViewDelegateProxy.proxy(for: base)
        // base可以理解为一个UIScrollView 实例
    }
```

```
    }
}

open class RxScrollViewDelegateProxy {

    public static func proxy(for object: ParentObject) -> Self {
        let maybeProxy = self.assignedProxy(for: object)
        let proxy: AnyObject
        if let existingProxy = maybeProxy {
            proxy = existingProxy
        } else {
            proxy = castOrFatalError(self.createProxy(for: object))
            self.assignProxy(proxy, toObject: object)
            assert(self.assignedProxy(for: object) === proxy)
        }
        let currentDelegate = self._currentDelegate(for: object)
        let delegateProxy: Self = castOrFatalError(proxy)
        if currentDelegate !== delegateProxy {
            delegateProxy._setForwardToDelegate(currentDelegate, retainDelegate: false)
            assert(delegateProxy._forwardToDelegate() === currentDelegate)
            self._setCurrentDelegate(proxy, to: object)
           /// 命中下面这一行assert
            assert(self._currentDelegate(for: object) === proxy)
            assert(delegateProxy._forwardToDelegate() === currentDelegate)
        }
        return delegateProxy
    }
}
```

重点逻辑

- delegateProxy 即使 RxDelegateProxy
- currentDelegate 为 RxDelegateProxy 指向的对象
- RxDelegateProxy._setForwardToDelegate 把 RxDelegateProxy 指向真实的 Delegate
- 标红的前面一句执行的时候，是调用 setDelegate 方法，把 RxDelegateProxy 的 proxy 设置给 UIScrollView(其实是一个 UICollectionView 实例)
- 然后进入了 MySDK 的 Hook 方法，把 RxDelegateProxy 给包了一层
- 最终结果如下图
- 然后导致 self._currentDelegate(for: object) 是 DelegateProxy 而非 RxDelegateProxy，**触发标红断言**

img

这个断言就很霸道，相当于 RxCocoa 认为就只有它能够去使用 Proxy 包装 delegate，其他人不能这样做，只要做了，就断言。

进一步分析

- 当前状态

img

- 再次进入 Rx 的方法
  - currentDelegate 是 UICollectionView 指向的 DelegateProxy（MySDK 的包装）
  - delegateProxy 指向还是 RxDelegateProxy
  - 触发 Rx 的 if 判断,Rx 会把其指向真实的 delegate 改向 UICollectionView 指向的 DelegateProxy
  - 导致循环指向，引用链中真实的 Delegate 丢失了

img

问题 2

上面提到多次调用导致了循环指向，而循环指向导致了在实际的方法转发的时候变成了死循环。

img

responds 代码

```
open class RxScrollViewDelegateProxy {

    override open func responds(to aSelector: Selector!) -> Bool {

        return super.responds(to: aSelector)

            || (self._forwardToDelegate?.responds(to: aSelector) ?? false)

            || (self.voidDelegateMethodsContain(aSelector) && self.hasObservers(selector: aSelector))

        }
}
```

```
@implementation BDCollectionViewDelegateProxy

- (BOOL)respondsToSelector:(SEL)aSelector {
    if (aSelector == @selector(bd_isCollectionViewTrackerDecorator)) {
        return YES;
    }
    return [super respondsToSelector:aSelector];
}

@end
```

似乎只要不多次调用就没有问题了？

关键在于 Rx 的 setDelegate 方法也调用了 get 方法，导致一次 get 就触发第二次调用。也就是多次调用是无法避免。

解决方案

问题的原因比较明显，如果改造 RxCocoa 的代码，把第三方可能的 Hook 考虑进来，完全可以解决问题。

解决方案 1

参考 MySDK 的 proxy 方案，在 proxy 中加入一个特殊方法，来判断 RxDelegateProxy 是否已经在引用链中，而不去主动改变这个引用链。

img

```
open class RxScrollViewDelegateProxy {
    public static func proxy(for object: ParentObject) -> Self {
        ...
        let currentDelegate = self._currentDelegate(for: object)
        let delegateProxy: Self = castOrFatalError(proxy)
        //if currentDelegate !== delegateProxy
        if !currentDelegate.responds(to: xxxMethod) {
            delegateProxy._setForwardToDelegate(currentDelegate, retainDelegate: false)
            assert(delegateProxy._forwardToDelegate() === currentDelegate)
```

```
            self._setCurrentDelegate(proxy, to: object)
            assert(self._currentDelegate(for: object) === proxy)
            assert(delegateProxy._forwardToDelegate() === currentDelegate)
        } else {
            return currentDelegate
        }

        return delegateProxy
    }

}
```

类似这样的改造，就可以解决问题。我们与 Rx 团队进行了沟通，也提了 PR，可惜最终被拒绝合入了。Rx 给出的说明是，Hook 是不优雅的方式，不推荐 Hook 系统的任何方法，也不想兼容任何第三方的 Hook。

解决方案 2

有没有可能，RxCocoa 不改代码，MySDK 来兼容?

刚才提到，有可能是两种状态。

- 状态 1
  - setDelegate 的时候，先进 Rx 的方法，后进 MySDK 的 Hook 方法，
  - 传给 Rx 的就是 delegate
  - 传给 MySDK 的是 RxDelegateProxy
  - Delegate 的 get 调用就触发 bug

img

- 状态 2

- setDelegate 的时候，先进 MySDK 的 Hook 方法，后进 Rx 的方法?
- 传给 Rx 的就是 DelegateProxy

img

其实如果是状态 2，似乎 Rxcocoa 的 bug 是不会复现的。

但是仔细查看 Rxcocoa 的 setDelegate 代码

```
extension Reactive where Base: UIScrollView {
    public func setDelegate(_ delegate: UIScrollViewDelegate)

    -> Disposable {
        return RxScrollViewDelegateProxy.installForwardDelegate(delegate, retainDelegate: false, onPro
    }
}

open class RxScrollViewDelegateProxy {
    public static func installForwardDelegate(_ forwardDelegate: Delegate, retainDelegate: Bool, onPro
        weak var weakForwardDelegate: AnyObject? = forwardDelegate as AnyObject
        let proxy = self.proxy(for: object)
        assert(proxy._forwardToDelegate() === nil, "")
        proxy.setForwardToDelegate(forwardDelegate, retainDelegate: retainDelegate)
        return Disposables.create {
            ...
        }
    }
}
```

emmm? Rx 里面，UICollectionView 的 setDelegate 和 Delegate 的 get 方法**不是 Hook...**

```
collectionView.rx.setDelegate(delegate)
```

```
let delegate = collectionView.rx.delegate
```

最终流程就只能是

- setDelegate 的时候，先进 Rx 的方法，传给 Rx 真实的 delegate
- 后进 MySDK 的 Hook 方法
- 传给 MySDK 的是 RxDelegateProxy
- Rx 里面获取 CollectionView 的 delegate 触发判断
- Delegate 的 get 调用就触发 bug

如果 MySDK 还是采用当前的 Hook 方案，就没法在 MySDK 解决了。

解决方案 3

仔细看了一下，发现 Rx 里面是通过重写 RxDelegateProxy 的 forwardInvocation 来达到方法转发的目的，即

- RxDelegateProxy 没有实现 `UICollectionViewDelegate` 的任何方法
- forwardInvocation 中处理 `UICollectionViewDelegate` 相关回调

回顾消息转发机制

img

我们可以在 forwardingTargetForSelector 这一步进行处理，这样可以避开与 Rx 相关的冲突，处理完再直接跳过。

- forwardingTargetForSelector 中针对 delegate 的回调，target 返回一个 SDK 处理的类，比 DelegateProxy
- DelegateProxy 上报完成之后，直接调用跳到 RxDelegateProxy 的 forwardInvocation 方法

这个解决方案其实也不完美，只能暂时规避与 Rx 的冲突。如果后续有其他 SDK 也来在这个阶段处理 Hook 冲突，也容易出现问题。

总结

确实如 Rx 团队描述的那样，Hook 不是很优雅的方式，任何 Hook 都有可能存在兼容性问题。

1. 谨慎使用 Hook
2. Hook 系统接口一定要遵循一定的规范，不能假想只有你在 Hook 这个接口
3. 不要假想其他人会怎么处理，直接把多种方案集成到一起，构建多种场景，测试兼容性

文章列举的方案可能不全或者不完善，如果有更好的方案，欢迎讨论。

参考文档

- NSProxy 使用
- 代理模式
- rxcocoa 学习-DelegateProxy
- https://github.com/ReactiveX/RxSwift

关于字节移动平台团队

字节跳动移动平台团队(Client Infrastructure)是大前端基础技术行业领军者，负责整个字节跳动的中国区大前端基础设施建设，提升公司全产品线的性能、稳定性和工程效率，支持的产品包括但不限于抖音、今日头条、西瓜视频、火山小视频等，在移动端、Web、Desktop 等各终端都有深入研究。

就是现在！**客户端 / 前端 / 服务端 / 端智能算法 / 测试开发** 面向全球范围招聘！**一起来用技术改变世界**，感兴趣可以联系邮箱 **chenxuwei.cxw@bytedance.com**，邮件主题 **简历-姓名-求职意向-期望城市-电话**。