聊聊HTTPS环境DNS优化：美图App请求耗时节约近半案例

> 活动预告：12月22~23，GIAC全球互联网架构大会将于上海举行，本周新增阿里，京东、LinkedIn等多名讲师出席，参看文末了解更多详情。

导读：移动互联网时代，APP 厂商之间的竞争非常激烈，而良好的用户体验是必须优先考虑的，美图产品以高颜值著称，对产品的用户体验非常重视。从技术的角度来看，客户端的体验优化当中 DNS 优化是非常关键的一环，怎么降低 DNS 的耗时，怎么减少域名劫持等问题，都是大家需要重点解决的研发问题。本文介绍美图 DNS 优化的实践，作者从原理到效果，整体讲解的非常全面，值得学习和借鉴。

DNS 服务作用于网络连接之前，将域名解析为 IP 地址供后续流程进行连接。

DNS 查询时，会先在本地缓存中尝试查找，如果不存在或是记录过期，就继续向 DNS 服务器发起递归查询,这里的 DNS 服务器一般就是运营商的 DNS 服务器。

在这过程中，会产生一些不可控的问题。

美图的移动端产品在实际用户环境下会面临 DNS 劫持、

耗时波动等问题，这些 DNS 环节的不稳定因素，导致后续网络请求被劫持或是直接失败, 对产品的用户体验产生不好的影响。

为此，我们对移动端产品的 DNS 解析进行了优化探索，产生了相应的 SDK。在这过程中，我们参考借鉴了业内的主流方案，也进行了一些实践上的思考。

下面的内容会主要以 Android 平台来进行说明。

LocalDNS VS HTTP DNS

在长期的实践中，互联网公司发现 LocalDNS 会存在如下几个问题：

- 域名缓存: 运营商 DNS 缓存域名解析结果，将用户导向网内缓存服务器;
- 解析转发 & 出口 NAT: 运营商 DNS 转发查询请求或是出口 NAT 导致流量调度策略失效;

为了解决 LocalDNS 的这些问题，业内也催生了 HTTP DNS 的概念，它的基本原理如下:

原本用户进行 DNS 解析是向运营商的 DNS 服务器发起 UDP 报文进行查询，而在 HTTP DNS 下，我们修改为用户带上待查询的域名和本机 IP 地址直接向 HTTP WEB 服务器发起 HTTP 请求，这个 HTTP WEB 将返回域名解析后的 IP 地址。

比如 DNSPod 的实现原理如下:

相比 LocalDNS, HTTP DNS 会具备如下优势:

- 根治域名解析异常: 绕过运营商的 DNS，向具备 DNS 解析功能的 HTTP WEB 服务器发起查询;
- 调度精准: HTTP DNS 能够直接获取到用户的 IP 地址，从而实现准确导流;
- 扩展性强: 本身基于 HTTP 协议，可以实现更强大的功能扩展;

那么，是否直接全部走 HTTP DNS 呢?

美图移动端 DNS 优化策略探索

HTTP DNS 相比 LocalDNS 存在一些优势, 然而 HTTP DNS 本身也是存在一定的成本问题。

美图的产品线丰富，涉及的域名也较为广泛，为了适应各产品的实际场景，在实践中我们设计了较为灵活的策略控

制。

首先，在策略上我们并未完全放弃 LocalDNS。

一个 App 涉及的域名众多，在策略上我们能够配置其核心 API 域名走 HTTP DNS，而对于非核心请求我们仍希望它先尝试走 LocalDNS, 在异常情况下才升级走 HTTP DNS。

那么如何判断 LocalDNS 的异常情况呢?

我们选择了几个指标来衡量一个 DNS 服务器的质量情况:

- IP 记录的 TTL 时间: 在 DNS 劫持发生的情况下，返回的 TTL 可能会有非常大的值;
- 解析耗时: 如果一个 DNS 服务器解析耗时不理想，那么它也不是我们希望的;
- 返回的 IP 的可连接性: 对返回的 IP 进行质量测试，如果连接状况不佳，那么这个 DNS 服务器有劫持的可疑;

在 Android 平台上，通过系统方法获得的解析结果信息是非常有限的，上面的指标有的将无法获取，因此在实践中我们会自己去构造 DNS 查询报文，向运营商的多个 DNS 服务器发起查询。

通过上面几个指标的综合评定，当 LocalDNS 表现不佳的时候，策略上我们将升级走 HTTP DNS，尝试让用户获取更好的 DNS 解析效果。

在 DNS 解析环节，还有一个我们比较关心的指标，那就

是 DNS 解析的耗时：

LocalDNS 在过期的情况下，会发起递归查询，这个时间是不可控的，在部分情况下甚至能达到数秒级别； HTTP DNS 相对会好一些，但正常来看，也会有200ms 左右的耗时。 这个时间能否再优化一些呢?

我们 SDK 在本地构建了自己的记录缓存池，每次通过 LocalDNS 或是 HTTP DNS 解析得到记录都存在缓冲池中。

当然，这个是普遍的做法，系统底层的 netdb 库也是这样实现。

区别在于我们做了一个小改动：对于过期的记录我们采用懒更新的策略，当查到过期的缓存记录时，先返回过期记录给用户，同时再异步重新发起 DNS 查询更新缓存记录。

这个小改动能够保证我们二次解析时都能命中本地缓存，极大地降低 DNS 解析耗时，不过它也带来了一定的风险性。

因此实践中，我们也会添加异步定期的 DNS 记录缓存池扫描功能，及时发现缓存中的过期记录并进行更新，也降低 App 命中过期记录的情况。

无侵入的 SDK 接入方式探索

在 DNS 优化的实践中，我们遇到最大的问题，倒不是策

略层面设计问题，而是我们的 DNS SDK 运用到实际 App 产品业务上的姿势问题。

业内对 HTTP DNS 在实际业务中的接入方式多采用 IP 直连的形式，

即原本直接请求 http://www.meitu.com，现在我们先调用 SDK 进行域名解析，拿到 IP 地址比如 1.1.1.1，然后替换域名为: http://1.1.1.1/；

这样操作之后， 由于 URL 中 HOST 已经是 IP 地址，网络请求库将跳过域名解析环节，直接向 1.1.1.1 服务器发起 HTTP 请求。

在实际操作中，对于 IP 直连的方案我们踩了不少的坑。

首先，对于 HTTP 请求，采用 IP 直连的方案后，我们还是需要进行的一个操作是手动配置 Header 中的 HOST :

```
URL htmlUrl = new URL("http://1.1.1.1/");

HttpURLConnection connection =
(HttpURLConnection) htmlUrl.openConnection();

connection.setRequestProperty("Host","www.meitu.com");
```

HTTP 协议相对比较容易，只需要处理 HOST，那么 HTTPS 呢?

发起HTTPS请求首先需要进行 SSL/TLS 握手，其流程如下:

1. 客户端发送 Client Hello，携带随机数、支持的加密算法等信息;
2. 服务端收到请求后，选择合适的加密算法，连同公钥证书、随机数等信息返回给客户端;
3. 客户端检验服务端证书的合法性，计算产生随机数并用证书公钥加密发送给服务端;
4. 服务端通过私钥获取随机数信息，基于之前的交互信息计算得到协商密钥并通知给客户端;
5. 客户端验证服务端发送的数据和密钥，通过后双方握手完成，开始进行加密通信;

在我们采用 IP 直连的形式后，上述 HTTPS 的第三步会发生问题, 客户端检验服务端下发的证书这动作包含两个步骤:

1. 客户端用本地保存的根证书解开证书链，确认服务端的证书是由可信任的机构颁发的。
2. 客户端需要检查证书的 Domain 域和扩展域是否包含本次请求的 HOST。

证书的验证需要这两个步骤都检验通过才能够进行后续流程，否则 SSL/TLS 握手将在这里失败结束。

由于在 IP 直连下，我们给网络请求库的 URL 中 host 部分

已经被替换成了 IP 地址，

因此证书验证的第二步中，默认配置下“本次请求的 HOST”会是一个 IP 地址，这将导致 domain 检查不匹配，最终 SSL/TLS 握手失败。

那么该如何解决这个问题?

解决 SSL/TLS 握手中域名校验问题的方法在于我们重新配置 HostnameVerifier, 让请求库用实际的域名去做域名校验，

代码示例如下:

```
final URL htmlUrl = new URL("https://1.1.1.1/");

HttpsURLConnection connection =
(HttpsURLConnection) htmlUrl.openConnection();

connection.setRequestProperty("Host","www.meipai.com");

connection.setHostnameVerifier(new
HostnameVerifier() {

    @Override
```

```
    public boolean verify(String hostname, SSLSession
session) {

        return
HttpsURLConnection.getDefaultHostnameVerifier()

                .verify("www.meipai.com",session);

    }

});
```

我们又解决了一个问题，那么 IP 直连下， HTTPS 的问题都搞定了吗?

没有，HTTPS 还有 SNI 的场景要特殊处理。

SNI（Server Name Indication）是为了解决一个服务器使用多个域名和证书的SSL/TLS扩展。它的基本工作原理如下:

1. 服务端配置有多个域名和对应的证书。客户端在与服务器建立SSL链接之时,先发送自己要访问站点的域名。

2. 服务器根据这个域名返回一个合适的证书。

跟上面 Domain 校验的情况类似，这里的网络请求库默认发送给服务端的 "要访问站点的域名" 就是我们替换后的 IP 地址。

服务端在收到这样一个 IP 地址形式的域名后将是一脸懵

逼，找不到对应的证书，最后只好下发一个默认的域名证书回来。

接下来发生的是，客户端在检验证书的 Domain 域时，怎么也检查不通过，因为服务端下发的证书本来就不是对应该域名的。

最后 SSL/TLS 握手失败告终。

上述这个 SNI 场景下的问题，我们是否有办法解决呢?

可以解决，需用客户端重新定制 SSLSocketFactory , 不过修改的代码相对较多，这里就不列举了。

如果我们 SDK 要接入到 App 实际业务中，到 HTTPS SNI 场景处理这里，相信很多同学都崩溃了，接入的工作量其实也不低。

很多情况下可能就做了妥协，只有 Okhttp 场景才使用这个 SDK，因为 Okhttp 本身支持 DNS 替换，没有上面那些问题。

在美图的实践中，我们不仅仅希望 Okhttp 的请求才进行这个 DNS 优化，我们希望在 App H5 页面加载、播放器播放等场景也能应用相应的优化。

在这样的需求下，IP 直连的接入方案带来的接入工作量其实不低，甚至需要改动到部分轮子。

在最初的实践中，我们也的确尝试了落实 IP 直连 到各个模块，然而即使克服了改造的工作量问题，实际运行上还是会有不少坑。

那么，有没有更合适的一种技术方案，能够降低 我们 DNS SDK 的接入工作量，也能兼顾各种使用场景，比如 HTTPS、RTMP 协议等?

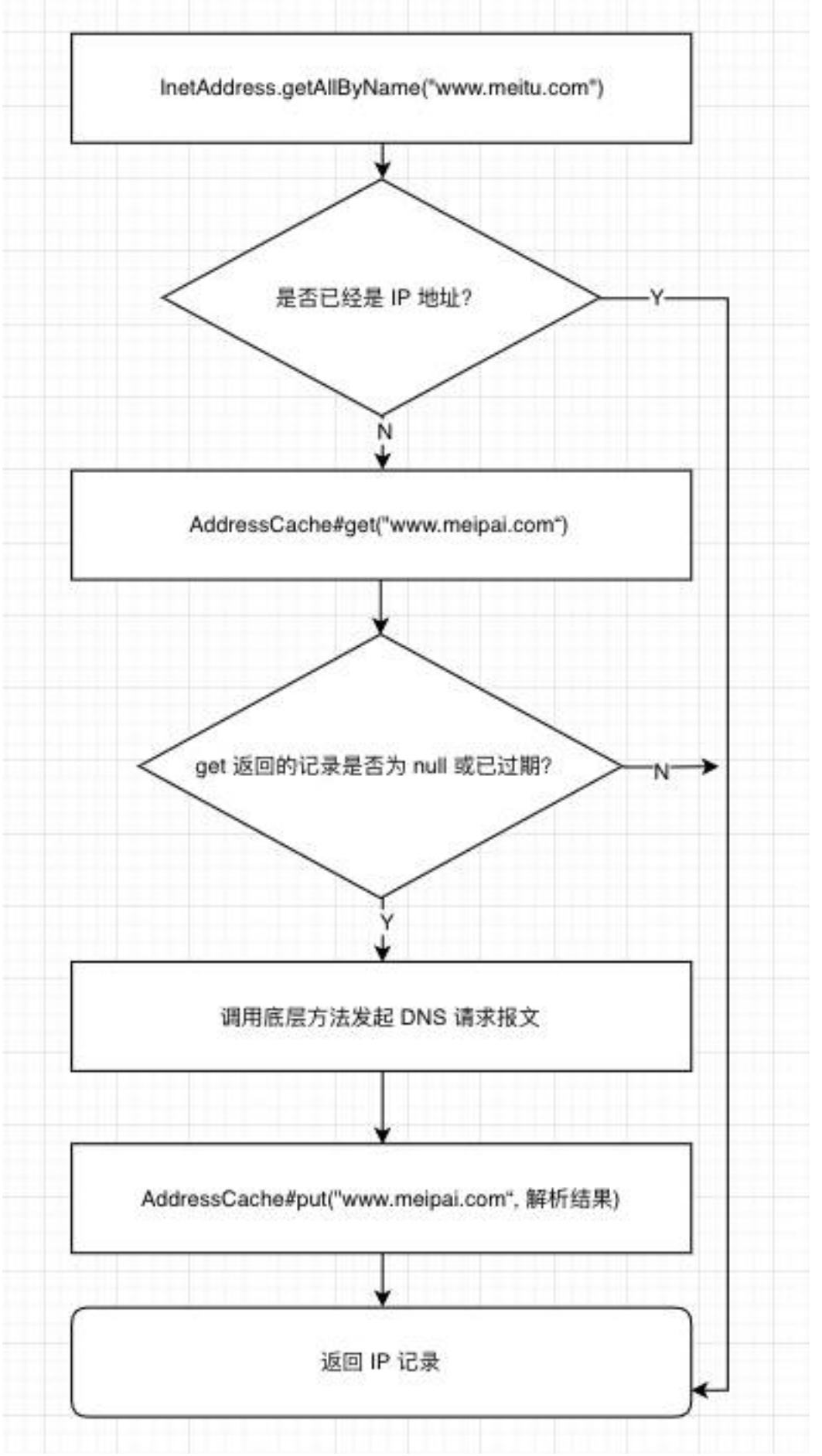
基于这样的目标，我们在实践中尝试探索了一种对业务集成友好的无侵入式 DNS SDK 集成方案。下面我们以 Android 平台进行说明。

我们知道在 Java 层面上进行 DNS 解析的基本方式是调用如下方法:

```
InetAddress.getAllByName("www.meipai.com");
```

Android 平台上常用的 Okhttp、HttpUrlConnection 等网络请求库都会依赖这个形式的 DNS 解析。

我们深入分析 InetAddress 的运行流程，其大致如下:



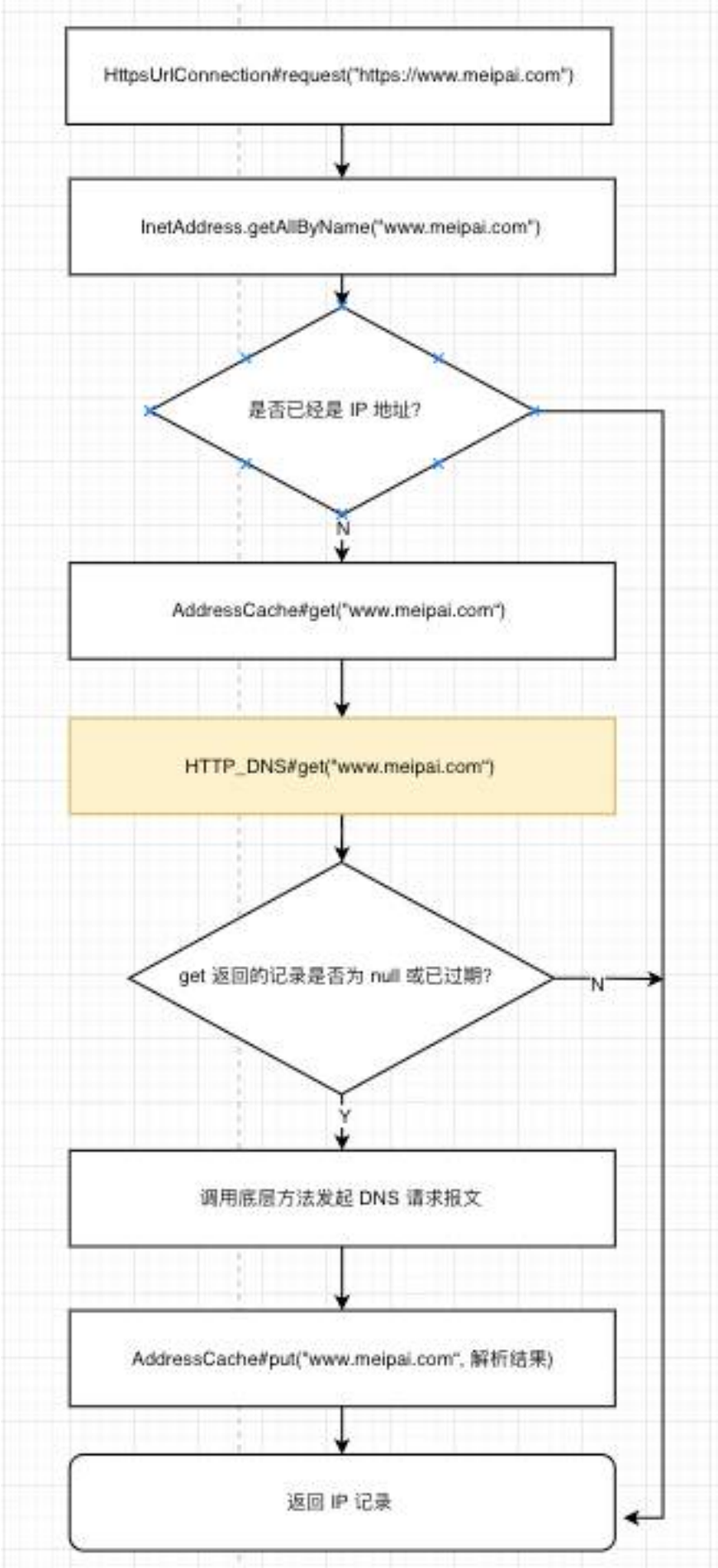
在上述流程中我们可以知道，InetAddress 会有到 AddressCache 尝试获取已缓存记录的动作，而这里 AddessCache 是一个 static 的 map 结构变量。

因此，在这里我们来对它做点小手脚：

- 模仿系统的 AddressCache 构造一个我们自己的 AddressCahce 结构，不过它的 get 方法被替换为从我们 SDK 获取解析记录。

- 通过反射的形式用我们修改后的 AddressCache 替换掉系统的 AddressCache 变量。

这个偷天换日的操作之后，HttpsUrlConnection 等 Java 层网络请求在进行 DNS 解析时就会是这样一个流程:



通过这个形式，我们能够完美解决 Java 层的 DNS SDK 接入问题，对于业务方来说，他们并不需要做任何 URL 替换操作，对应的 HTTPS 场景下的问题也不复存在。

Java 层的接入解决了， 那么 Native 层呢?

我们知道在 Android 平台上，像 WebView、播放器等模块他们进行网络连接的操作都是在 native 层进行的，并不会调用到 Java 层的 InetAddress 方法。

首先在 C/C++ 层，我们知道进行 DNS 解析会使用 getaddrinfo 或是 gethostbyname2 这两个函数。

另外我们还知道，在 Android 等 Linux 系统下，对于 .so 这类可共享对象文件会是 ELF 的文件格式。

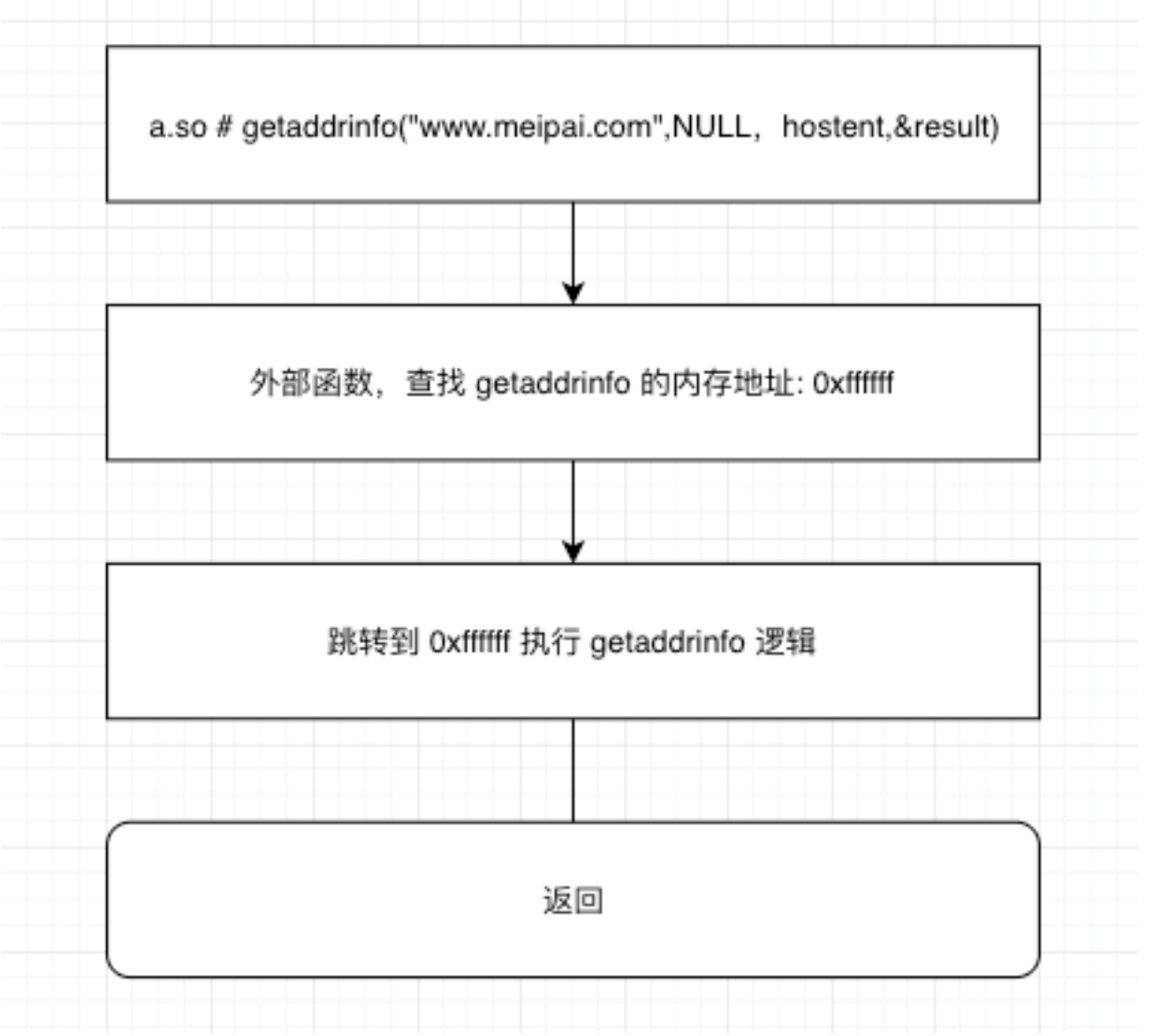
因此从这些已知信息，我们可以得到下列一些情况:

我们的 App 中 a.so 中直接使用到了系统 libc.so 中的 getaddrinfo 函数，

那么根据 ELF 文件规范，在 a.so 的 .rel.plt 表中会有如下关系定义: getaddrinfo ==> 0xFFFFFF 。

.rel.plt 表中的映射关系为 a.so 的运行指出了 getaddrinfo 这个外部符号在当前内存空间中的绝对地址。

正常情况下，a.so 中执行到 getaddrinfo 的函数流程是这样的:

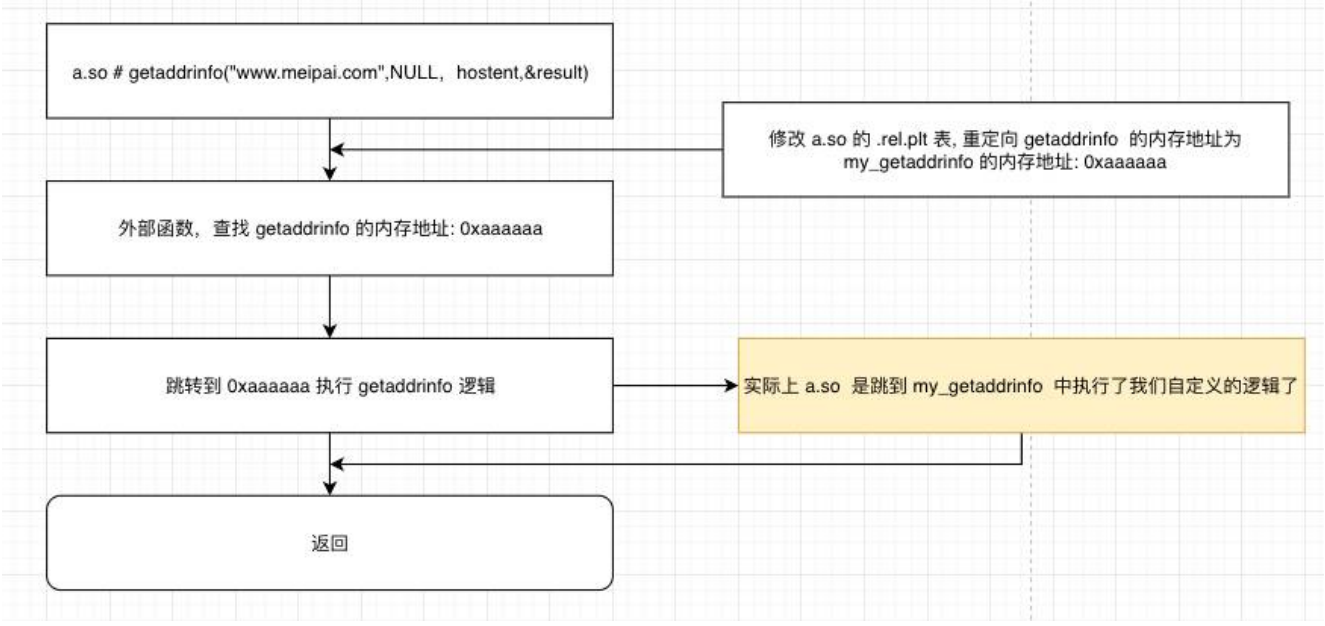


那么在这里，我们是否可以手动修改这个映射表内容，把 getaddrinfo 的内存地址替换成我们的 my_getaddrinfo 地址呢?

这样，a.so 在实际运行时会被拐到我们的 my_getaddrinfo 中?

实际上，确实是可行的。 我们尝试在 SDK 启动后，对 a.so 的 .rel.plt 表进行修改，达到接管 a.so DNS 的目的,

修改后的 a.so 运行流程如下:

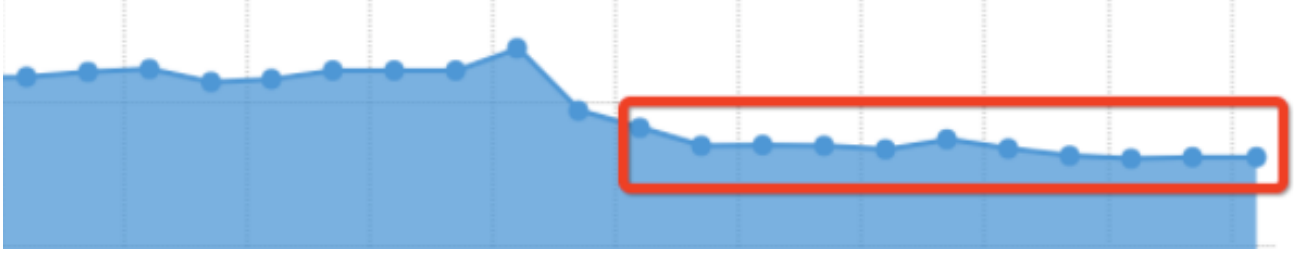


通过上面的方式，我们能够比较完美地接管 App 在 Java 层 和 Native 层 DNS 过程，实现业务方无任何额外改动的情况下运用我们的 DNS SDK 优化效果。

SDK 上线后的效果表现

在实际运用中，我们取得了比较好的效果。得益于 DNS SDK 在命中本地缓存率上的策略优化，我们的移动端产品在网络请求中 DNS 解析环节耗时得到降低，

从实际监控数据来看，**完整网络请求的耗时也能够降低 100ms 左右。**



通过 HTTP DNS 的引入和 LocalDNS 优化升级策略，我们的网络请求成功率有提升，在未知主机等具体错误率表现出下降的趋势。

由于 SDK 层面本身做好了灵活的策略配置，我们通过线上监控和配置也让各产品在效益和成本之间取得一个最佳的平衡点。

号外：

美图架构，专注于虚拟化平台建设、流媒体、云存储、千万同时在线的通讯服务、音视频编解码等基础设施建设，现急需相关领域爱好者加入，工作地点可自由选择北京、厦门、深圳，待遇从优，美女多多。现紧缺岗位如下：

- Go/C 开发工程师：我们 50% 以上代码使用 Go
- Java 开发工程师
- 音视频编解码研究人员
- Docker 虚拟化底层研发工程师

有兴趣者请联系：yt@meitu.com or yxq@meitu.com or zl3@meitu.com

美图技术沙龙深圳站于12月底开启，欢迎关注官方公众号，关注最新动向。微信号: MTtechsalon

参考阅读：

- GIAC全球互联网架构大会最新日程
- App域名劫持之DNS高可用 - 开源版HttpDNS方案详解

- 手机QQ上传速度提升8倍秘诀：解决速度与成功率的“鱼翅”项目
- 腾讯TMQ团队移动App的网络优化：24小时流量优化到原来15%历程

活动预告：

12 月 22 ~ 23 日，GIAC 全球互联网架构大会将于上海举行。GIAC 是高可用架构技术社区推出的面向架构师、技术负责人及高端技术从业人员的技术架构大会。GIAC 于 2016 年 12 月成功举办了第一届，今年的 GIAC 已经有腾讯、阿里巴巴、百度、平安、饿了么、携程、七牛、蚂蚁金服、罗辑思维、摩拜、唯品会，LinkedIn, Pivotal, Mesosphere, AdMaster, Hulu 等公司专家出席。

本期 GIAC 大会上，移动开发相关部分精彩的议题如下：

移动开发最佳实践

不同行业、不同时期的团队的移动最佳实践

出品人

戴 铭

滴滴出行技术专家

Swift 将 Web 代码转成 60 帧满帧原生应用的方案及实践

戴铭

滴滴出行技术专家

Android框架虚拟化实战

董福源

360手机卫士技术经理

移动Android应用高可用性探索

邹迪飞(之羲)

阿里巴巴高级技术专家

注：出品人及演讲议题以最新官网为准，全部最新演讲议题请点击“阅读原文”至官网查看。

参加 GIAC，盘点年底最新技术，目前单人购买优惠 600 元，多人购买有更多优惠。点击“阅读原文”了解大会更多详情。