百度App网络深度优化：DNS 和连接优化

作者丨蔡锐（百度 App 资深工程师）

编辑丨覃云

网络优化是客户端几大技术方向中公认的一个深度领域，百度 App 也不例外，今天，我们在这里向大家介绍百度 App 网络深度优化的实践经验，内容主要包括 DNS 优化和连接优化，希望对大家在网络方向的学习和实践有所帮助。

一、DNS 优化

百度起家于搜索，整个公司的网络架构和部署都是基于标准的 internet 协议，目前已经是全栈 HTTPS，来到移动互联网时代后，总的基础架构不变，但在客户端上需要做很多优化工作。

DNS（Domain Name System），它的作用是根据域名查出 IP 地址，它是 HTTP 协议的前提，只有将域名正确的解析成 IP 地址后，后面的 HTTP 流程才能进行，所以一般做网络优化会首选优化 DNS。

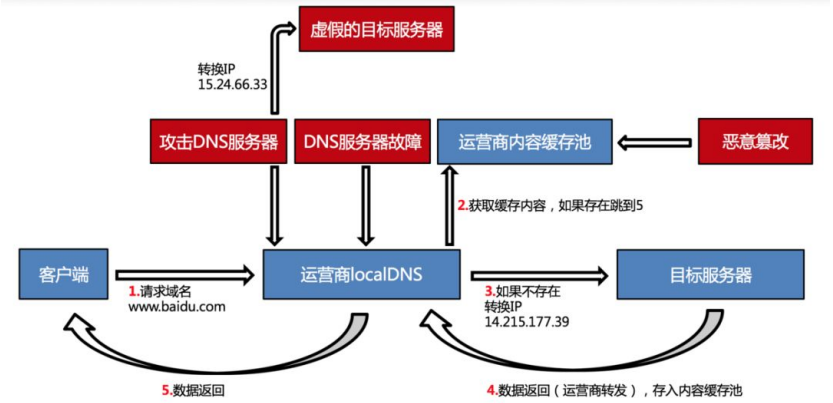
1. 背景

DNS 优化核心需要解决的问题有两点：

【1】由于 DNS 劫持或故障造成的服务不可用，进而影响用户体验，影响公司的收入。

【2】由于 DNS 调度不准确导致的性能退化，进而影响用户体验。

百度 App 承载着亿级流量，每年都会遇到运营商 DNS 劫持或运营商 DNS 故障，整体影响非常不好，所以 DNS 优化刻不容缓，通过下图会更直观的了解运营商劫持或故障的原理。

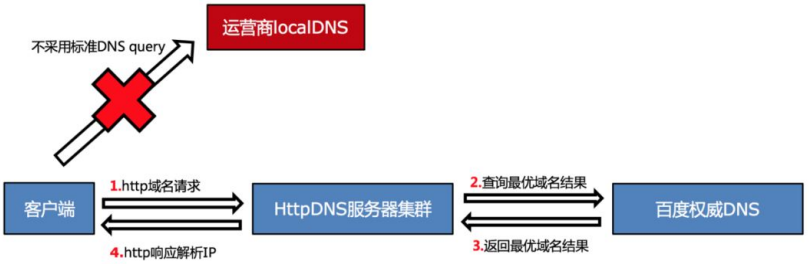


运营商劫持或故障的原理

2. HTTPDNS

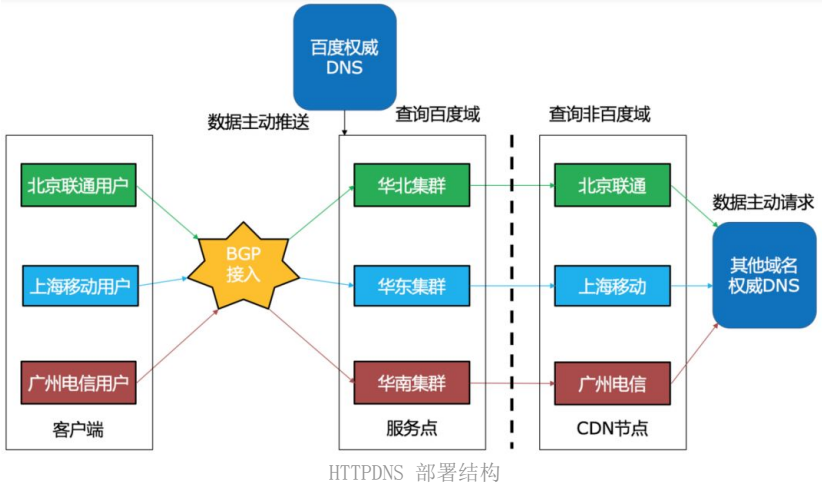
既然我们面临这么严峻的问题，那么我们如何优化 DNS 呢？答案就是 HTTPDNS。

大部分标准 DNS 都是基于 UDP 与 DNS 服务器交互的，HTTPDNS 则是利用 HTTP 协议与 DNS 服务器交互，绕开了运营商的 Local DNS 服务，有效防止了域名劫持，提高域名解析效率，下图是 HTTPDNS 的原理。



HTTPDNS 原理

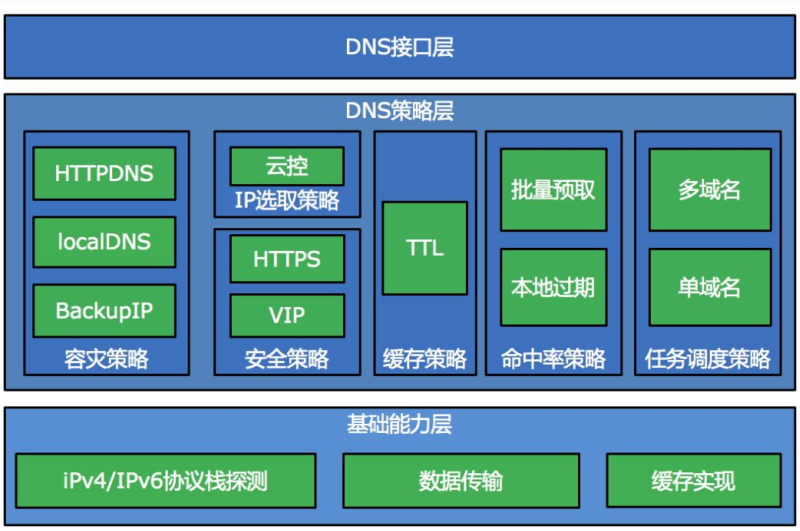
百度 App HTTPDNS 端上的实现是基于百度 SYS 团队的 HTTPDNS 服务，下图介绍了 HTTPDNS 的服务端部署结构。



HTTPDNS 部署结构

HTTPDNS 服务是基于 BGP 接入的，BGP 英文 Border Gateway Protocol，即边界网关协议，是一种在自治系统之间动态的交换路由信息的路由协议，BGP 可以根据当前用户的运营商路由到百度服务点的对应集群上，对于第三方域名，服务点会通过百度部署在运营商的 CDN 节点向其他域名权威 DNS 发起查询，查询这个运营商下域名的最优 IP。

百度 App 独立实现了端的 HTTPDNS SDK，下图介绍了端 HTTPDNS 的整体架构。



端 HTTPDNS 的整体架构

DNS 接口层

DNS 接口层解决的问题是屏蔽底层的细节，对外提供简单整洁的 API，降低使用者的上手成本，提高开发效率。

DNS 策略层

DNS 策略层通过多种策略的组合，使 HTTPDNS 服务在性能，稳定性，可用性上均保持较高的水准，下面讲解下每个策略设计的初衷和具体实现。

容灾策略

这是一个非常关键的策略，主要解决 HTTPDNS 服务可用性的问题，实践证明，这个策略帮助百度 App 在异常情况下挽救回很多流量。

【1】当 HTTPDNS 服务不可用并且本地也没有缓存或者缓存失效的时候，会触发降级策略，降级成运营商的 localDNS 方案，虽然存在运营商事故或者劫持的风险，但保障了 DNS 服务的可用性。

【2】当 HTTPDNS 服务和 localDNS 服务双双不可用的情况下，会触发 backup 策略，使用端上的 backup IP。

什么是 backup IP？backup IP 是多组根据域名分类的 IP 列表，可云端动态更新，方便后续运维同学调整服务端的节点 IP，不是所有域名都有对应的 backup IP 列表，目前百度 App 只能保证核心域名的可用性。

既然是一组 IP，便有选取问题，backup IP 选取机制是怎样的呢？我们的中心思想就是要在端上利用最小的代价，并且考虑服务端的负载均衡，得到相对正确或者合理的选取结果。通过运营商和地理信息，可以选择一个相对较优的 IP，但获取地理信息需要很大耗时，外加频次很高，代价很大，所以我们选择了 RR 算法来代替上面的方法（RR 算法是 Round-Robin，轮询调度），这样客户端的代价降低到最小，服务端也实现了负载均衡。

安全策略

【1】HTTPDNS 解决的核心问题就是安全，标准的 DNS 查询大部分是基于 UDP 的，但也有基于 TCP 的，如果 UDP 被封禁，就需要使用 TCP。不管是 UDP 还是 TCP，安全性都是没有保障的，HTTPDNS 查询是基于标准的 HTTP 协议，为了保证安全我们会在 HTTP 上加一层 TLS（安全传输层协议），这便是 HTTPS。

【2】解决了传输层协议的安全性后，我们要解决下域名解析的问题，上面我们提到 HTTPDNS 服务是基于 BGP 接入的，在端上采用 VIP 方式请求 HTTPDNS 数据（VIP 即 Virtual IP，VIP 并没有与某设备存在必定的绑定关系，会跟随主备切换之类的情况发生而变换，VIP 提供的服务是对应到某一台或若干台服务器的），既然请求原始数据需要使用 IP 直连的方式，那么就摆脱了运营商 localDNS 的解析限制，这样即使运营商出现了故障或者被劫持，都不会影响百度 App 的可用性。

任务调度策略

HTTPDNS 服务提供了两类 HTTP 接口，用于请求最优域名结果。第一种是多域名接口，针对不同的产品线，下发产品线配置的域名，第二种是单域名接口，只返回你要查询的那个域名结果，这样的设计和标准的 DNS 查询基本是一样的，只不过是从 UDP 协议变成了 HTTP 协议。

【1】多域名接口会在 App 冷启动和网络切换的时候请求一次，目的是在 App 的网络环境初始化或者变化的时候预先获取域名结果，这样也会减少单域名接口的请求次数。

【2】单域名接口会在本地 cache 过期后，由用户的操作触发网络请求，进而做一次单域名请求，用户这次操作的 DNS 结果会降级成 localDNS 的结果，但在没有过期的情况下，下次会返回 HTTPDNS 的结果。

IP 选取策略

IP 选取策略解决的核心问题是最优 IP 的选取，避免因为接入点的选取错误造成的跨运营商耗时。HTTPDNS 服务会将最优 IP 按照顺序下发，客户端默认选取第一个，这里没有做客户端的连通性校验的原因，主要还是担心端上的性能问题，不过有容灾策略兜底，综合评估还是可以接受的。

缓存策略

大家对于 DNS 缓存并不陌生，它主要是为了提升访问效率，操作系统，网络库等都会做 DNS 缓存。

DNS 缓存中一个重要的概念就是 TTL（Time-To-Live），在 localDNS 中针对不同的域名，TTL 的时间是不一样的，在 HTTPDNS 中这个值由服务端动态下发，百度 App 目前所有的域名 TTL 的配置是 5 分钟，过期后如果没有新的 IP 将继续沿用老的 IP，当然也可以选择不沿用老的 IP，而降级成 localDNS 的 IP，那么这就取决于 localDNS 对于过期 IP 的处理。

命中率策略

如果 HTTPDNS 的命中率是 100%，在保证 HTTPDNS 服务稳定高效的前提下，我们就可以做到防劫持，提升精准调度的能力。

【1】为了提升 HTTPDNS 的命中率，我们选择使用多域名接口，在冷启动和网络切换的时候，批量拉取域名结果并缓存在本地，便于接下来的请求使用。

【2】为了再一次提升 HTTPDNS 的命中率，当用户操作触发网络请求，获取域名对应的 IP 时，会提前进行本地过期时间判断，时间是 60s，如果过期，会发起单域名的请求并缓存起来，这样会持续延长域名结果的过期时间。本地过期时间与上面提到的 TTL 是客户端和服务端的双重过期时间，目的是在异常情况下可以双重保证过期时间的准确性。

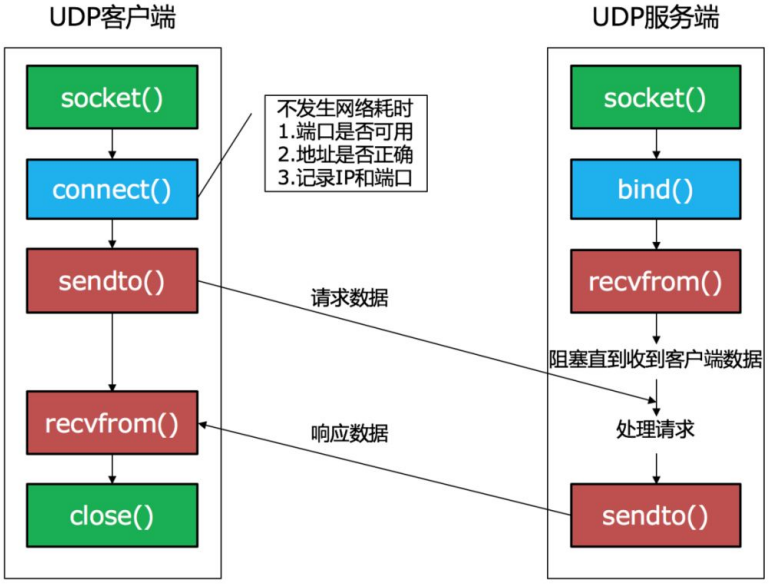
基础能力层

基础能力层主要提供给 DNS 策略层所需要的基础能力，包括 IPv4/IPv6 协议栈探测的能力，数据传输的能力，缓存实现的能力，下面将讲解每种能力的具体实现

IPv4/IPv6 协议栈探测

百度 App 的 IPv6 改造正在如火如荼的进行中，端上在 HTTPDNS 的 IP 选取上如何知道目前属于哪个协议栈成为关键性问题，并且这种判断要求性能极高，因为 IP 选取的频次实在是太高了。

我们选取的方案是 UDP Connect，那么何为 UDP Connect？大家都知道 TCP 是面向连接的，传输数据前客户端都要调用 connect 方法通过三次握手建立连接，UDP 是面向无连接的，无需建立连接便能收发数据，但是如果我们调用了 UDP 的 connect 方法会发生什么呢？当我们调用 UDP 的 connect 方法时，系统会检测其端口是否可用，地址是否正确，然后记录对端的 IP 地址和端口号，返回给调用者，所以 UDP Connect 不会像 TCP Connect 发起三次握手，发生网络真实损耗，UDP 客户端只有调用 send 或者 sendto 方法后才会真正发起真实网络损耗。



UDP Connect 原理

有了 UDP Connect 的基础保障，我们在上层做了缓存机制，用来减少系统调用的损耗，时机上目前仅在冷启动和网络切换会触发探测，在同一种网络制式下探测一次基本可以确保当前网络是 IPv4 栈还是 IPv6 栈。

目前百度 App 客户端对于 IPv4/IPv6 双栈的策略是保守的，仅在 IPv6-only 的情况下使用 v6 的 IP，其余使用的都是 v4 的 IP，双栈下的方案后续需要优化，业内目前标准的做法是 happy eyeball 算法，什么叫 happy eyeball 呢？就是不会因为 IPv4 或 IPv6 的故障问题，导致用户的眼球一直在等待加载或者出错，这就是 happy eyeball 名字的由来。

happy eyeball 有 v1 版本 RFC6555 和 v2 版本 RFC8305，前者是 Cisco 提出来的，后者是苹果提出来的。happy eyeball 解决的核心问题是，复杂环境下 v4 和 v6 IP 选取的问题，它是一套整体解决方案，对于域名查询的处理，地址的排序，连接的尝试等方面均做出了规定，感兴趣的同学可以查看：

- https://tools.ietf.org/html/rfc6555
- https://tools.ietf.org/html/rfc8305

2. 数据传输

数据传输主要提供网络请求的能力和数据解析的能力。

【1】网络请求失败重试的机制，获取 HTTPDNS 结果的成功率会大大影响 HTTPDNS 的命中率，所以客户端会有一个三次重试的机制，保障成功率。

【2】数据解析异常的机制，如果获取的 HTTPDNS 的结果存在异常，将不会覆盖端上的缓存。

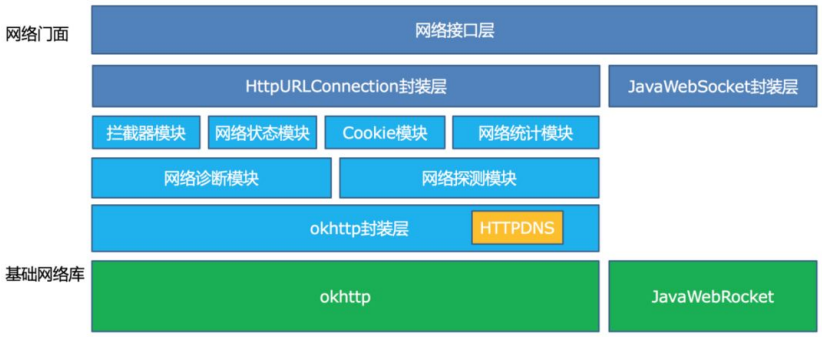
3. 缓存实现

缓存的实现基本可以分为磁盘缓存和内存缓存，对于 HTTPDNS 的缓存场景，我们是选其一还是都选择呢？百度 App 选择的是内存缓存，目的是防止我们自己的服务出现问题，运维同学在紧急情况下切换流量，如果做了磁盘缓存，会导致百度 App 在重启后也可能不可用，但这种问题会导致 APP 在冷启动期间，HTTPDNS 结果未返回前，还是存在故障或者劫持的风险，综合评估来看可以接受，如果出现这种极端情况，影响的是冷启动阶段的一些请求，但只要 HTTPDNS 结果返回后便会恢复正常。

3.HTTPDNS 的最佳实践

百度 App 目前客户端网络架构由于历史原因还未统一，不过我们正朝着这个目标努力，下面着重介绍下 HTTPDNS 在 Android 和 iOS 网络架构中的位置及实践。

HTTPDNS 在 Android 网络架构的位置及实践百度 App 的 Android 网络流量都在 okhttp 之上，上层进行了网络门面的封装，封装内部的实现细节和对外友好的 API，供各个业务和基础模块使用，在 okhttp 上我们扩展了 DNS 模块，使用 HTTPDNS 替换了原有的系统 DNS。

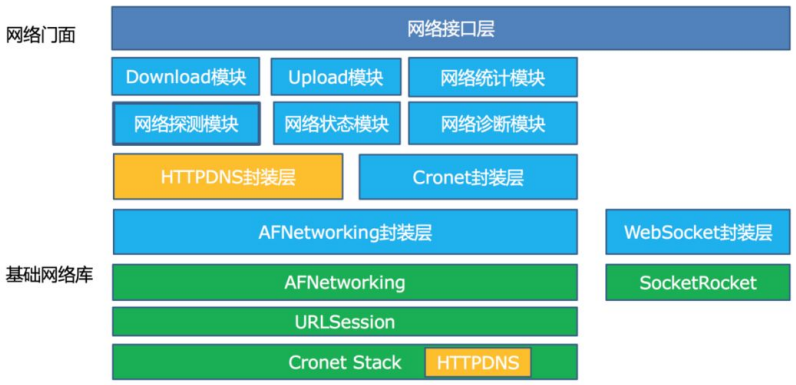


HTTPDNS 在 Android 网络架构的位置

HTTPDNS 在 iOS 网络架构的位置及实践

百度 App 的 iOS 网络流量都在 cronet（chromium 的 net 模块）之上，上层我们使用 AOP 的方式将 cronet stack 注入进 URLSession 里，这样我们就可以直接使用 URLSession 的 API 进行网络的操作而且更易于系统维护，在上层封装了网络门面，供各个业务和基础模块使用，在 cronet 内部我们修改了 DNS 模块，除了原有的系统 DNS 逻辑外，还添加了 HTTPDNS 的逻辑。

iOS 上还有一部分流量是在原生 URLSession 上，主要是有些第三方业务没有使用 cronet 但还想单独使用 HTTPDNS 的能力，所以就有了下面的 HTTPDNS 封装层，方法是在上层直接将域名替换成 IP，域名对于底层很多机制是至关重要的，比如 https 校验，cookie，重定向，SNI（Server Name Indication）等，所以将域名修改成了 IP 直连后，我们又处理了以上三种情况，保证请求的可用性。



HTTPDNS 在 iOS 网络架构的位置

4. 收益

DNS 优化的收益主要有两点，一是防止 DNS 的劫持（在出问题时显得尤为重要），降低网络时延（在调度不准确的情况下，会增大网络的时延，降低用户的体验），这两点收益需要结合业务来说，以百度 App Feed 业务为例，第一点上我们取得了比较大的效果，iOS 劫持率由 0.12% 降低到 0.0002%，Android 劫持率由 0.25% 降低到 0.05%，第二点的收益不明显，原因在于 Feed 业务主要目标群体在国内，百度在国内节点布局相对丰富，服务整体质量也较高，即使出现调度不准确的情况，差值也不会太大，但如果在国外情况可能会差很多。

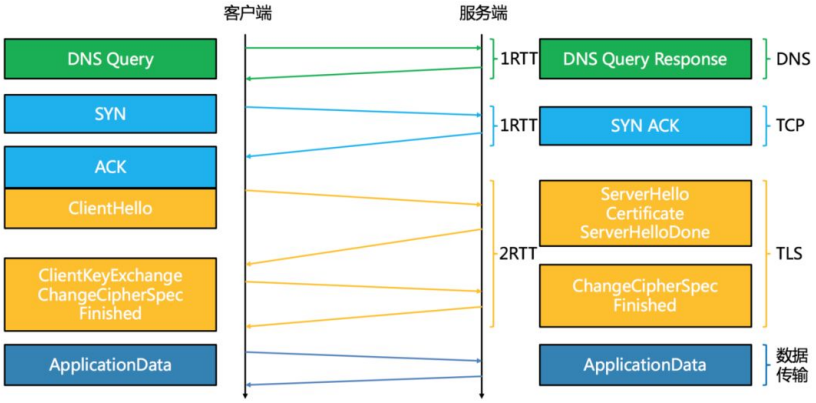
二、连接优化

1. 背景

连接优化需要解决两个核心问题：

1. 连接建立耗时较长，导致请求的总时长变长，进而影响用户体验。
2. 在多变的网络环境下，连接建立的过程可能会失败，导致成功率下降，进而影响用户体验。

百度 App 承载着亿级流量，对于每一个请求都需要追求耗时短，成功率高的体验。从协议角度出发，如何才能做到这一点呢？首先我们来看下建立连接耗时的原理。



建立连接耗时的原理

从上图我们能清晰的看出：

1.DNS Query 需要 1 个 RTT（Round-Trip Time，即往返时间），百度 App 都是基于 HTTPDNS 服务的，所以大部分会命中缓存，如果降级走了系统 DNS，也会命中缓存，命中不了的由于是基于 UDP 协议，所以在连接耗时上没有太大的影响，线上的数据也能说明这点。

2.TCP 要经历 SYN，SYN/ACK，ACK 三次握手的 1.5 个 RTT，不过 ACK 和 ClientHello 合并了，所以就是 1 个 RTT。

3.TLS（Transport Layer Security，即传输层安全性协议）需要经过握手和密钥交换 2 个 RTT。

综上所述，DNS，TLS，TCP 握手阶段用了 4 个 RTT 才到了 ApplicationData 阶段，也就是数据开始传输阶段。

通过上面的分析可以总结出，如果我们能尽量的将 TLS 和 TCP 的 RTT 减少，将会大大降低连接耗时的时间。

2. 连接优化我们都能做什么？

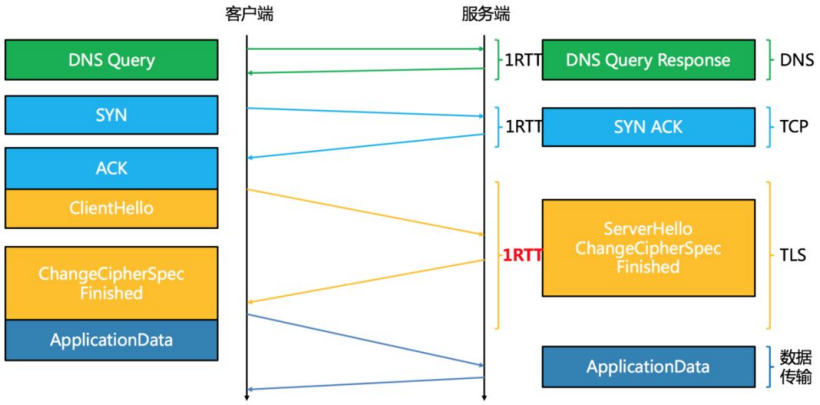
百度 App 的优化目标分为两类，一类是 TLS 的连接优化，一类是 TCP 的连接优化。

TLS 的连接优化

TLS 的连接优化，需要服务端和客户端都需要支持，共同完成优化手段，包括 Session Resumption 和 False Start。

Session Resumption

Session Resumption 中文意思是会话复用，下图讲解了 Session Resumption 的协议原理。



Session Resumption 的协议原理

通过上图可以看出 TLS 密钥协商交换的过程没有了，但具体是如何实现的呢？包含两种方式，一种是 Sesssion Identifier，一种是 Session Ticket。

1）Session Identifier

Session Identifier 中文为会话标识符，更像我们熟知的 session 的概念。是 TLS 握手中生成的 Session ID。服务端会将 Session ID 保存起来，客户端也会存储 Session ID，在后续的 ClientHello 中带上它，服务端如果能找到匹配的信息，就可以完成一次快速握手。

2）Session Ticket

Session Identifier 存在一些弊端，比如客户端多次请求如果没有落在同一台机器上就无法找到匹配的信息，但 Session Ticket 可以。Session Ticket 更像我们熟知的 cookie 的概念，Session Ticket 用只有服务端知道的安全密钥加密过的会话信息，保存在客户端上。客户端在 ClientHello 时带上了 Session Ticket，服务器如果能成功解密就可以完成快速握手。

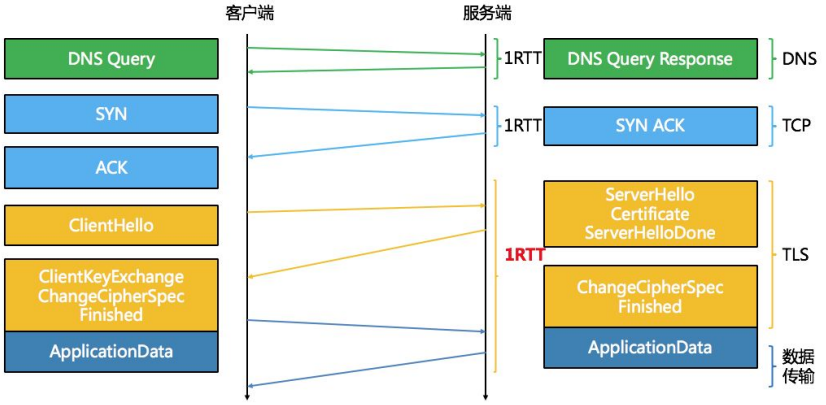
不管是 Session Identifier 还是 Session Ticket 都存在时效性问题，不是永久生效，对于这两种方式大家可以查看：

https://tools.ietf.org/html/rfc5077

百度 App 的网络协议层对这两种方式都是支持的，省去了 TLS 握手过程中证书下载，密钥协商交换的环节，节省了 1 个 RTT 的时间。

False Start

False Start 的中文意思是抢跑，下图讲解了 False Start 的协议原理。



False Start 的协议原理

上图很清晰的说明在 TLS 第一步握手成功后，客户端在发送 Change Cipher Spec Finished 的同时开始数据传输，服务端在 TLS 握手完成时直接返回应用数据。应用数据的发送实际上并未等到握手全部完成，所以称之为抢跑。

从结果看省去了 1 个 RTT 的时间。False Start 有两个前提条件，一是要通过应用层协议协商 ALPN（Application Layer Protocol Negotiation）握手，二是要支持前向安全的加密算法。False Start 在未完成握手的情况下就发送了数据，前向安全可以提高安全性，具体协议实现，大家可以查看：

https://tools.ietf.org/html/rfc7918

百度 App 的网络协议层对 False Start 是支持的。

这里说句题外话，其实 TCP 层有个类似的连接优化手段叫 Fast Open，感兴趣的同学，可以查看：

https://tools.ietf.org/html/rfc7413

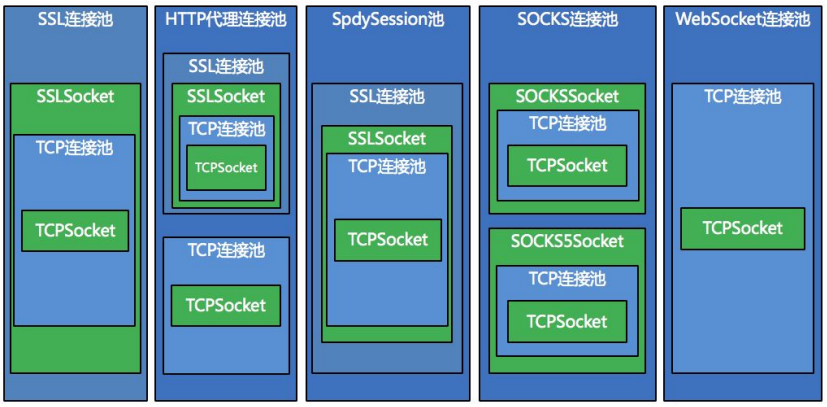
Session Resumption 和 False Start 的区别

两者对于 TLS 来说都是节省一个 RTT，Session Resumption 在第一次握手时还是需要 2 个 RTT，在第二次握手时才能复用减少到 1 个 RTT。False Start 是端上的行为，故每次都会减少到 1 个 RTT。

TCP 的连接优化

TCP 的连接优化，我们先从连接池说起，首先让我们来认识下连接池都有哪些类型。

1. 连接池



连接池的类型

上图展示了连接池的不同类型，都是大家耳熟能详的协议连接池，有低级连接池，包含 TCP 连接池（管理 HTTP 请求的连接）和 WebSocket 连接池（管理 WebSocket 连接）。

有高级连接池，包括 HTTP 代理连接池（管理 HTTP 代理请求的连接），SpdySession 连接池（管理 SPDY 和 HTTP/2 请求的连接），SOCKS 连接池（管理 SOCKS 和 SOCKS5 代理的连接），SSL 连接池（管理 HTTPS 请求的连接）。

不同类型的连接池以组合的形式互相复用能力。

1）SSL 连接池管理的是 SSLSocket，但 SSLSocket 又依赖于 TCP 连接池提供的 TCPSocket。

2）HTTP 代理连接池如果走 HTTP 协议，那么就需要 TCP 连接池提供 TCPSocket，如果走 HTTPS 协议，那么就需要 SSL 连接池提供 SSLSocket。

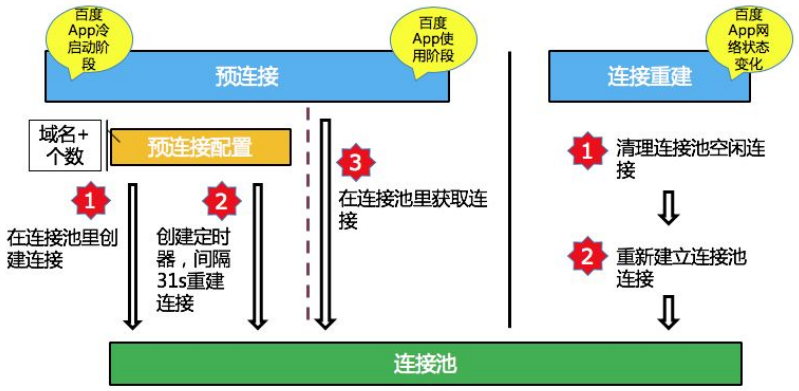
3）SpdySession 连接池依赖 SSL 连接池提供 SSLSocket，这里需要说明下，虽然 HTTP/2 协议没有强制绑定 HTTPS，但是在实际开发中确实都是绑定 HTTPS，百度 App 使用 ALPN 来协商 HTTP/2。

4）SOCKS 连接池管理的 SOCKSSocket 和 SOCKS5Socket 都需要依赖 TCP 连接池提供的 TCPSocket，虽然 SOCKS5 支持 UDP，但 cronet 网络库暂时没有实现。

5）WebSocket 连接池依赖 TCP 连接池提供的 TCPSocket，声明下这里没有说明 WSS（Web Socket Secure）的情况。

TCP 连接优化是一个比较复杂的内容，百度 App 做了针对性场景优化，包括预连接，连接重建，备用连接，复合连接。

2. 预连接



预连接和连接重建

预连接，预先创建好的连接。它解决的场景是在 App 使用阶段可以无耗时的获取连接。下面用四个问答来解释预连接。

问题一：预连接是否能解决所有网络请求的提前连接建立？

答：答案是否定的，预连接需要业务方进行核心业务的评估，针对核心的域名进行预连接的建立。

问题二：预连接既然针对的是特定的域名，那么是如何配置的呢？

答：采用域名 + 连接数的方式进行配置，比如 https://a.baidu.com|2，表示给 a.baidu.com 这个域名配置两条预连接，这里要说明下，在 HTTP/1.x 协议下，网络库的实现都会对于单域名有最大连接数的限制，不同网络库的个数限制不一样，有 5 个也有 6 个，但对于 HTTP/2 协议，这个连接数就只能是 1 个。

问题三：预连接是如何建立的？

答：在网络库初始化的时候，会根据使用者的配置延迟 5s 进行预连接的建立，主要是考虑网络库在冷启动下对于启动性能的影响，为了保证网络库的整体性能，预连接的总个数限制在 20 个。

问题四：预连接是如何保持的？

答：在网络库初始化的时候，除了进行预连接的建立，还会创建一个预连接的定时器，这个定时器会每隔 31s，这个值的设定取决于 BFE（Baidu Front End，是七层流量的统一接入系统）和 BGW（Baidu Gate Way，百度自主研发的四层负载均衡平台）对超时的最小值设定，根据使用者的配置重新建立连接。

3. 连接重建

连接重建，将连接重新建立。它解决的场景是 App 网络状态发生变化，IP 地址变化，导致连接不可用。下面用三个问答来解释连接重建。

问题一：连接重建是否针对连接池里的所有连接？

答：答案是肯定的。

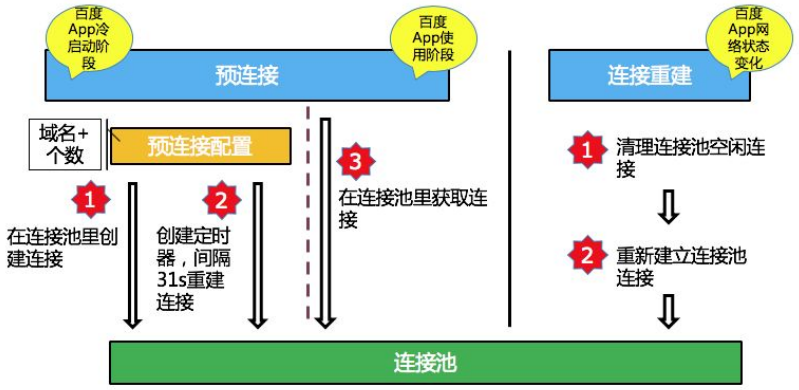
问题二：连接重建的过程是什么样的？

答：在网络状态变化的时候，第一步会清除掉连接池里的 idle socket，何为 idle socket？即空闲 socket，对于从未使用过的空闲 socket 超过 60 秒清除，对于使用过的空闲 socket 超过 90 秒清除。第二步重建连接需要等待 200ms，目的是等待 DNS 先重建完成。

问题三：连接重建对于性能有影响吗？

答：出于性能考虑，连接重建的连接个数限制是 100 个。

4. 备用连接



备用连接和复合连接

备用连接，预备的连接。它解决的场景是正常发送一个请求当 group 内无连接可用的时候（何为 group？group 是管理 socket 的最小单元，内部包含活跃 socket，空闲 socket，连接任务，等待请求）。下面用三个问答来解释备用连接。

问题一：备用连接是否针对所有请求？

答：答案是肯定的。

问题二：备用连接的过程是什么样的？

答：当有请求来临时，连接池内无连接可用，会启动一个定时器开启备用连接，定时器的间隔时间是 250ms，与主连接进行竞争，如果主连接因为网络抖动或者网络状态不好，导致连接失败，那么备用连接就直接发送请求。如果主连接成功，那么备用连接就被取消掉。

问题三：备用连接的目的是什么？

答：在连接池无连接的情况下，务必是要创建连接的，在主连接之外加一个备用连接，会大大提升创建连接的成功率，从而提升用户体验。

5. 复合连接

复合连接，即多条连接。它解决的场景是为了多个 IP 地址的连接选取问题。下面用三个问答来解释复合连接。

问题一：复合连接是否针对所有请求？

答：答案是肯定的。复合连接可以全局开关，百度 App 现阶段暂时没有开启复合连接。

问题二：复合连接的过程是什么样的？

答：众所周知域名 DNS 查询一般情况下会返回多个 IP，我们以域名查询返回两个 IP 为例

1）如果结果中存在 IPv6 的地址，那么会优先选用 IPv6 的地址，这个规则 follow HappyEyeBall 机制（可参考系列一对于 HappyEyeBall 的介绍）。

2） 接下来这两个 IP 会按照顺序尝试建立连接，如果第一个 IP 返回失败，将立即开始连接第二个 IP。

3）如果第一个 IP 率先成功返回，那么第二个 IP 将被加入连接尝试列表并停止所有尝试连接。

4）如果第一个 IP 失败，会立刻开始第二个 IP 的连接。

5）如果第一个 IP 处于 pending 状态，那么会启动一个定时器，默认延迟 2s 会发起第二个 IP 的连接，如果是多个 IP 将会递归连接，需要特别说明下，不同的网络制式延迟时间会不一样，这样体验也会更好。

问题三：复合连接的目的是什么？

答：复合连接的好处是提供最优的 IP 选取机制，但也会带来服务端的高负载，所以使用的时候需要进行综合评估。

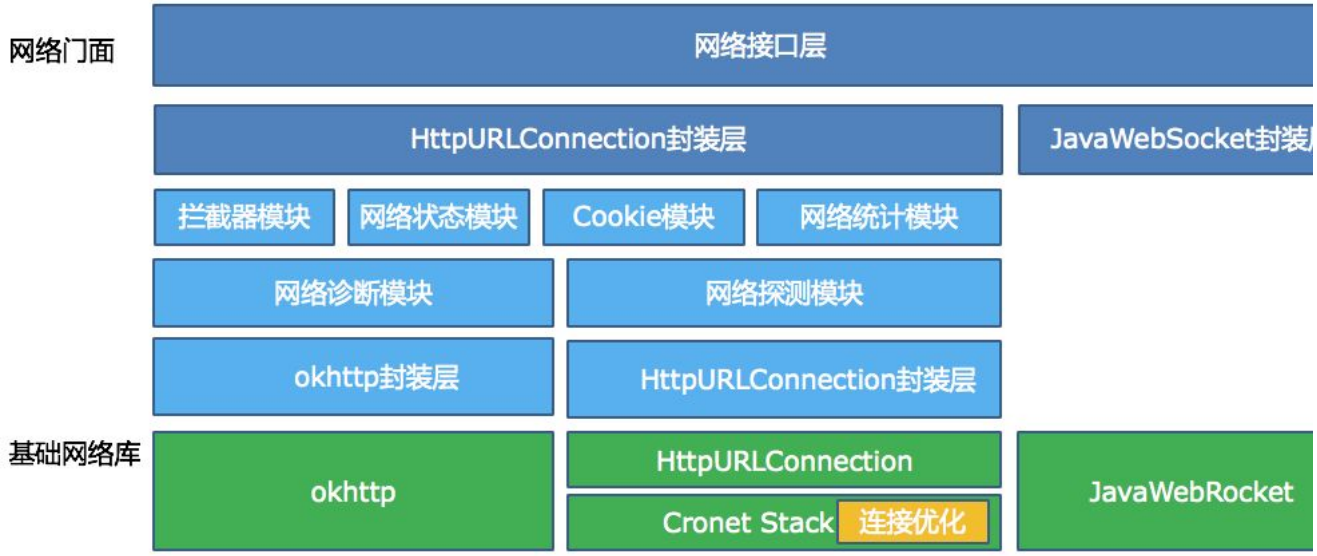
3. 连接优化的最佳实践

百度 App 目前客户端网络架构由于历史原因还未统一，不过我们正朝着这个目标努力。

我们的中心思想是以系统网络库的 API 调用接口为中心，上层建立网络门面，供外部便捷调用，底层通过系统机制以 AOP 的方式将 cronet（chromium 的 net 模块）注入进系统网路库，达到双端网络架构统一，能力复用。

下面着重介绍下连接优化在 Android 和 iOS 网络架构中的位置及实践。

1. 连接优化在 Android 网络架构的位置及实践

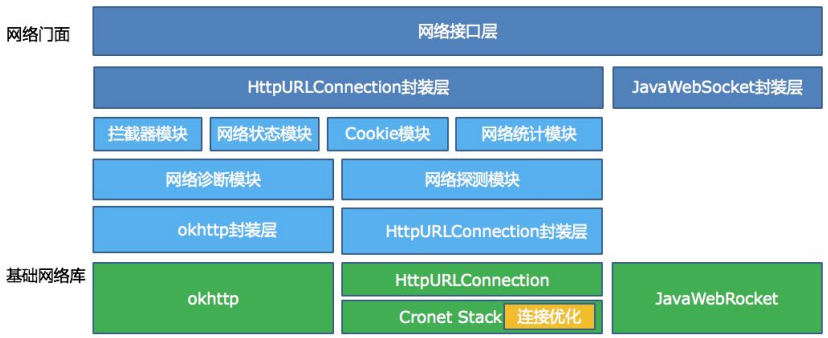


连接优化在 Android 网络架构的位置

百度 App 的 Android 网络流量目前都在 okhttp 之上，上层进行了网络门面的封装，封装内部的实现细节和对外友好的 API，目前我们正在进行重构，默认采用 Android 标准的网络接口 HttpURLConnection，它的底层由系统提供的 okhttp 的实现。

订制方面利用 URL Stream Protocol 机制将 HttpURLConnection 底层网络协议栈接管为 cronet，供各个业务和基础模块使用，连接优化的所有内容在 cronet 网络库内部实现。

2. 连接优化在 iOS 网络架构的位置及实践



连接优化在 iOS 网络架构的位置

百度 App 的 iOS 网络流量目前都在 cronet 之上，上层我们使用 iOS 的 URL Loading System 机制将 cronet stack 注入进 URLSession 里，这样我们就可以直接使用 URLSession 的 API 进行网络的操作而且更易于系统维护，在上层封装了网络门面，供各个业务和基础模块使用。

在 cronet 内部实现了预连接（主要针对百度 App 的几个核心域名进行预连和保活），连接重建（针对所有请求），备用连接（针对所有请求），复合连接（iOS 上暂时没有开启），Session Resumption（针对所有请求），False Start（针对所有请求）。

4. 收益

连接优化的收益主要体现在网络时延和网络成功率上，这两点收益需要结合业务来说，以百度 App Feed 刷新这个典型业务场景为例。

Feed 刷新文本请求网络时延降低 16%，Feed 刷新图片请求网络时延降低 12%，可谓收益相当明显。

成功率方面，Feed 刷新文本请求成功率提升 0.29%，Feed 刷新图片请求成功率提升 0.23%，也是非常不错的收益。

三、结语

DNS 优化和连接优化是个持续性的话题，没有最优只有更优。上面介绍的百度 App 的一些经验和做法并不见得完美，但我们会继续深入的优化下去，持续提升百度 App 的网络性能。

以上优化由百度 App 团队，内核团队，OP 团队共建完成。最后感谢大家的辛苦阅读，希望对你有所帮助。

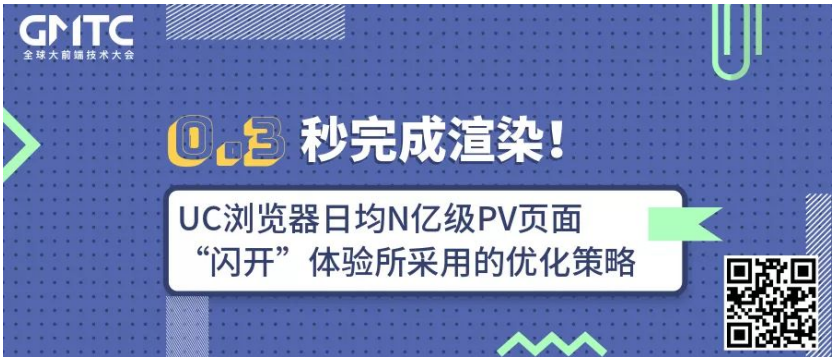
参考资料

- https://tools.ietf.org/html/rfc7858
- https://tools.ietf.org/html/rfc6555
- https://tools.ietf.org/html/rfc8305
- https://tools.ietf.org/html/rfc7918
- https://tools.ietf.org/html/rfc5077
- https://tools.ietf.org/html/rfc7413

作者简介

蔡锐，9 年移动客户端开发经验，在百度先后主导过订制 ROM 领域、多屏互动领域、Hybrid 跨平台领域等多个技术领域的开发，目前担任百度 App 的客户端资深工程师，参与基础技术的研究，专攻动态化和网络优化方向。

GMTC 全球大前端技术大会上，我们邀请到了来自 Google、BAT、美团、京东、滴滴、字节跳动等 60+ 一线技术专家与你共话前端那些事，涵盖小程序、Flutter、前端安全、工程化、性能优化等 20+ 热点技术，不可错过。欢迎点击“阅读原文”了解详情。



阅读原文