# 境外业务性能优化实践

## 前言

### 性能问题简介

应用性能是产品用户体验的基石，性能优化的终极目标是优化用户体验。当我们谈及性能，最直观能想到的一个词是“快”，Strangeloop 在对众多的网站做性能分析之后得出了一个著名的3s定律“页面加载速度超过3s，57%的访客会离开”，可见页面加载速度对于互联网产品的重要性。速度在 Google、百度等搜索引擎的 PR 评分中也占有一定的比例，会影响到网站的 SEO 排名。“天下武功、唯快不破”，套在性能上面也非常适用。

**性能指标**

性能优化是个系统性工程，涉及到后端、前端、移动端、系统网络及各种基础设施，每一块都需要做各自的性能优化。当我们系统的分析性能问题时，可以通过以下指标来衡量：



- Web 端：首屏时间、白屏时间、可交互时间、完全加载时间等。

首屏时间是指从用户打开网页开始到浏览器第一屏渲染完成的时间，是最直接的用户感知体验指标，也是性能领域公认的最重要的核心指标。

首屏时间 = DNS 时间 + 建立连接时间 + 后端响应时间 + 网络传输时间 + 首屏页面渲染时间。

- 移动端：Crash 率、内存使用率、FPS（ Frames Per Second，每秒传输帧数）、端到端响应时间等。

Native 相比于 H5 在交互体验方面有更多的优势，FPS 是体现页面顺畅程度的一个重要指标，另外端上的开发同学还需要关注 App 进程的 CPU 使用率、内存使用率等系统性能指标。端到端响应时间是衡量一个 API 性能的关键指标，比纯后端响应时间更全面，它会受到 DNS、网络带宽、网络链路、HTTP payload 等多个因素的影响。端到端响应时间是 DNS 解析时间、网络传输时间及后端响应时间的总和。

- 后端：响应时间（ RT ）、吞吐量（ TPS ）、并发数等。

后端系统响应时间是指系统对请求做出响应的时间（ 应用延迟时间 ），对于面向用户的 Web 服务，响应时间能很好度量应用性能，会受到数据库查询、RPC 调用、网络 IO、逻辑计算复杂度、JVM 垃圾回收等多方面因素影响。对于高并发的应用和系统，吞吐量是

个非常重要的指标，它与 request 对 CPU、内存资源的消耗，调用的外部接口及 IO 等紧密关联。

**影响性能的因素**

互联网产品是创意、研发、系统、网络、硬件、维护等众多资源相互交织的集合体，性能受多方面因素影响，犹如一只木桶，木桶能盛多少水，并不取决于最长的那块木板，而是缺与最短的那块木板，也可称之为短板效应。影响产品性能的因素有：

（1）产品逻辑与用户行为

产品逻辑过于复杂、功能交互过于丰富、产品设计过于绚丽、页面元素素材过多等都会影响产品性能。

（2）基础网络

中国的基础网络是世界上最复杂的基础网络，国内的网络运营商众多且各自为政，互联互通成本很高。对于境外业务来说更是要面对国内国际网络交互的情况，再加上 GFW 的存在，网络延迟、丢包现象非常严重。

（3）代码及应用

开发语言瓶颈、代码质量及系统架构等都会影响系统性能，常见的代码及应用问题有：



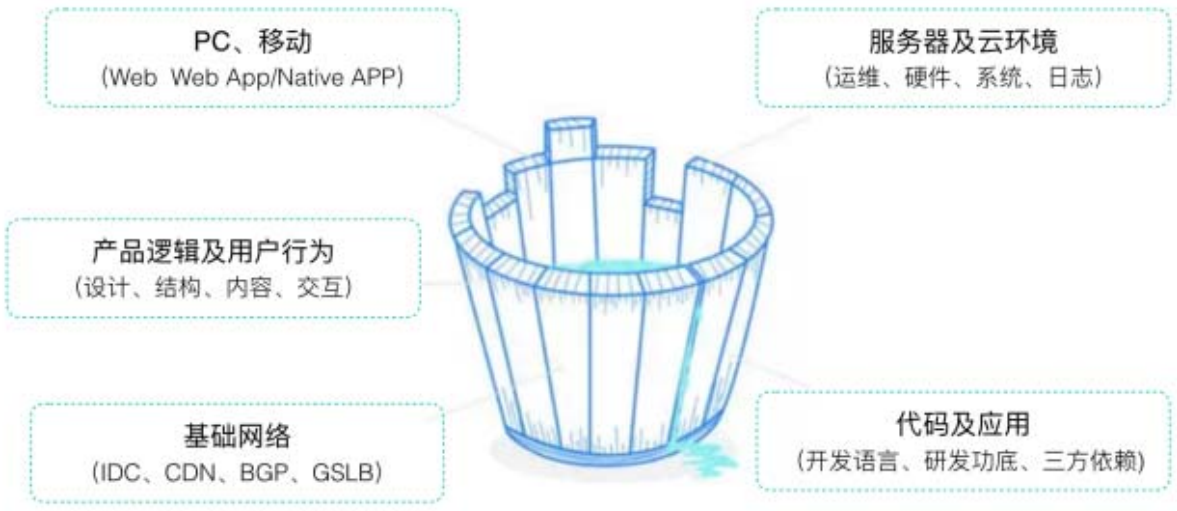
- 架构不严谨，业务发展超越架构支撑能力而导致系统负荷过载，进而导致出现系统奔溃、响应超时等现象；
- 不合理、不适用的架构必将影响速度，如单点、无 Cache、应用混部署、没有考虑分布式、集群化等；
- 研发“功力”和经验不足，开发的 App、Server 效率和性能较低、不稳定也是常见的事情；
- 没有性能意识，业务上量后系统出现连锁反应，导致性能问题叠加，直接影响用户体验；
- 多数的性能问题发生在数据库上，由慢 SQL、过多查询等原因造成的数据库瓶颈，没有做读写分离、分库分表等。

（4）移动端环境

移动互联网时代，移动端的环境的复杂性对产品的性能影响也很大，比如用户的设备类型、设备性能、操作系统类型、系统版本及网络类型（电信/移动/联通, wifi/4G/3G/2G）等。

（5）服务器及云环境

硬件的发展遵循着摩尔定律，硬件的生命周期一般都很短，服务器老化或其他硬件问题经常会导致应用故障。IDC、机架、服务器、内存、磁盘、网卡等不同硬件和操作系统上运行的应用性能差距可以达到数十倍之多。




## 境外业务的特点

境外业务与其他境内业务相比，区别主要表现在以下及方面：

- 用户在境外访问

境外业务很大一部分流量来自境外访问，国外网络情况十分复杂，一些国家的网络基础设施很差，以越南泰国等东南亚国家为例，很多国家 4G 覆盖率很低，从国外访问国内机房，不仅网络链路长，还涉及到跨网跨运营商跨 GFW 的访问情况，访问延迟、网络丢包等情况非常严重；



- 大量使用 Hybrid 实现

由于业务发展很快，业务新增及变更也相对频繁，为适应业务的快速发展，我们大量采用的是H5方式实现，大量使用 Hybrid 模式。

- 与境外商家对接

除了用户在境外访问，境外业务还会和很多境外商家、供应商或代理商有合作对接，同样面临着跨国网络访问的问题。

基于以上背景，如何产品性能，做到像国内业务一样，其中面临了很多的技术挑战。本文将从网络优化、前端优化、后端优化等角度来介绍境外业务在性能优化方面的做过的一些事情。

# 网络优化

影响网络性能的问题有很多，常见的网络问题有以下几类。

**问题一：DNS 问题**

DNS 问题最容易被大家所忽视，而实际上 DNS 出问题的概率非常大，DNS 问题主要有2类：

- 一类是 DNS 解析慢或解析失败，我们统计过一些数据，我们的域名在国内 DNS 解析耗时大概在30-120ms之间，而国外网络下耗时达到200-300ms左右，在2G/3G等弱网环境下，DNS 解析失败非常常见，DNS 对于首次网络访问的耗时及网络成功率会有很大的影响。

下图是我们利用页面测速工具（gtmatrix）在加拿大温哥华节点测试的一个页面首次访问时的网络请求情况，可以看出当用户在加拿大第一次访问且运营商 LocalDNS 无 NS 缓存记录时，DNS 解析是非常慢的。


- 另一类常见问题是 DNS 劫持或失效，乌云（WooYun）上曾报过多起因域名服务商安全漏洞被黑客利用导致网站 NS 记录被篡改的 case。而更多的 DNS 劫持确实来自于网络运营商的作恶，主要有以下几种：



（1）本地缓存，各运营商确保网内访问，同时减少跨网结算，运营商在网内搭建了内容缓存服务器，通过把域名强行指向内容缓存服务器。

（2）内容劫持，有部分 LocalDNS 会把部分域名解析指向内容缓存，并替换成第三方广告联盟的广告。

（3）解析转发，运营商的 LocalDNS 还存在解析转发的形象。指运营商自身不进行域名递归解析，而是把域名解析请求转发到其他运营商的递归 DNS 上，解析请求的来源 IP 成了其他运营商的 IP，从而导致用户请求跨网访问，性能变差。

**问题二：网络链路问题**

链路过长、请求经过的路由转发跳数过多、跨网访问等都是影响网络传输性能的关键因素。另外网络攻击（主要是 DDos、CC 攻击等洪水攻击）流量也影响着网络链路的稳定性。据统计，骨干网上每天有数百G的流量为攻击流量。

**问题三：传输 Payload 大小**

移动设备的网络在非 Wi-Fi 环境下时通常不太稳定，再加上有 TCP 拥塞策略的限制，数据传输量越大传的就越慢，我们需要尽量的减少数据传输量。通常的做法有：数据压缩、图片压缩、选择更高效的序列化算法（比如 protocol Buffer）等。

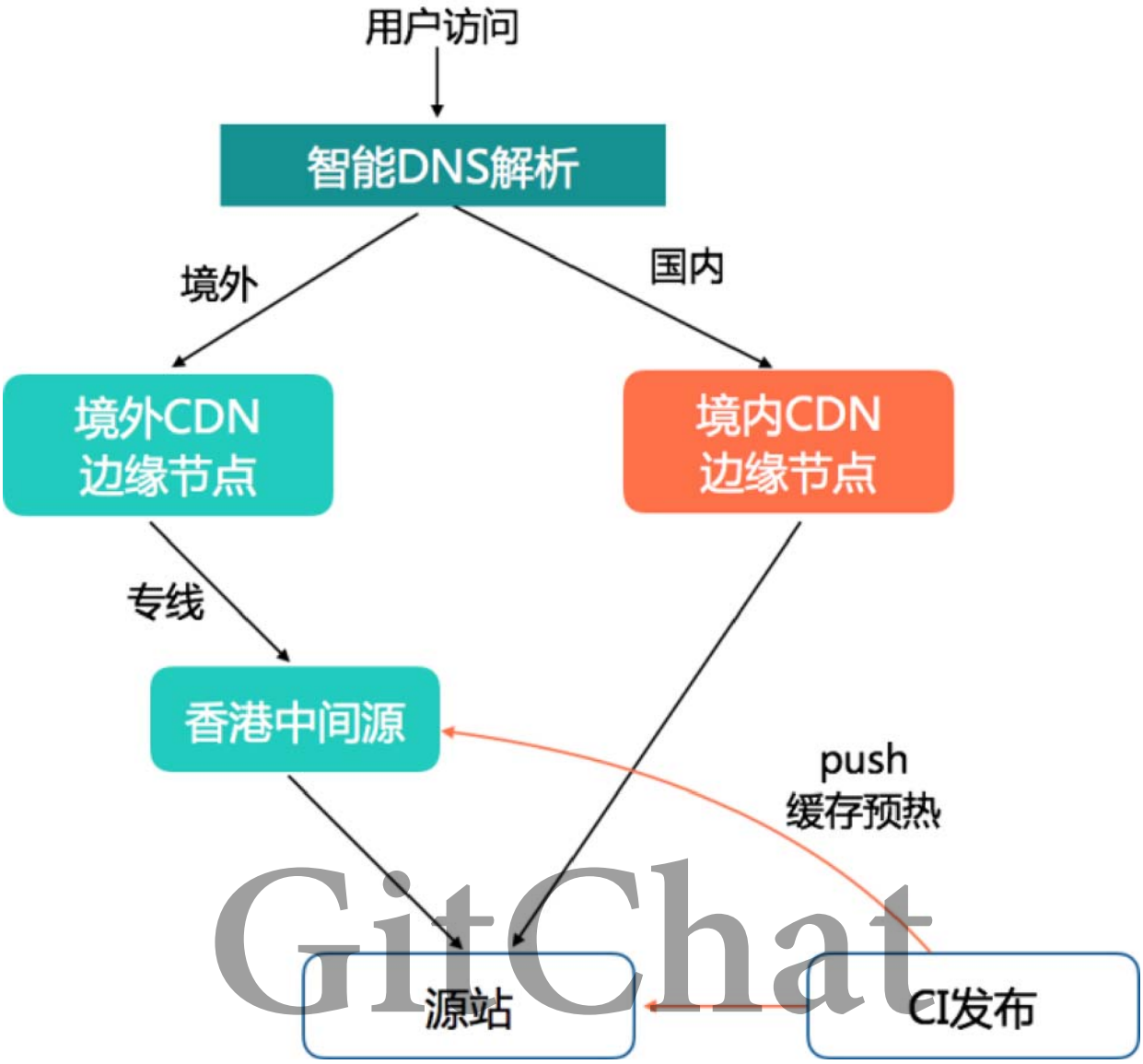
我们在网络优化方面主要做了以下几件事情：

- CDN 优化：海外 CDN 加速、CDN 缓存预热；
- DNS Prefetch：DNS 预热，刷新移动设备系统 /VM 的 DNS 缓存；
- 长连接：”代理长连接”Shark，专线链路优化，并且有效解决了 DNS 的瓶颈问题。
- 网络链路优化：通过专线和代理，解决公网链路长及网络抖动不稳定的问题。

## CDN 优化

CDN 服务的好坏主要取决于节点部署的覆盖程度、带宽以及调度算法。对国内业务而言，使用蓝汛、网宿、帝联等老牌 CDN 服务商或腾讯云、阿里云、UPYUN 等云 CDN 服务商性能都很好，但这些 CDN 服务商在境外的节点就很少了。为了提升境外的静态资源加速效果，我们尝试对接了很多国外的知名 CDN 服务商（每一家在不同区域的服务能力不一样，但都很贵！）。通过智能 DNS 解析用户的 IP 来源，如果是境外访问则 CNAME 到国外 CDN，国内访问时仍然走的是国内 CDN。

CDN 也是一种缓存，是缓存就不得不谈命中率的问题。 如果用户在境外访问时 CDN 未命中，静态资源从境外回源到国内源站获取，成本非常高。为了提升缓存命中率，我们的做法是在香港搭了一个 CDN 中间源，在前端资源发布时会调用 cdn 的 push 接口把资源预热到中间源，保证当境外边缘节点缓存未命中时无需再回源到国内 IDC，只需从中间源获取。






DNS Prefetch

由于 DNS 的种种问题，腾讯推出了 HTTPDNS 服务，使用 HTTP 协议向 DNS 服务器的80端口进行请求，代替传统的 DBS 协议向 DNS 服务器的53端口进行请求，绕开 Local DNS，避免网络劫持和跨网访问等问题。但 HTTPDNS 需要能够获取 CDN 边缘节点的 IP，这就限制了只有像腾讯、阿里这种有自建 CDN 的大厂才能实现。

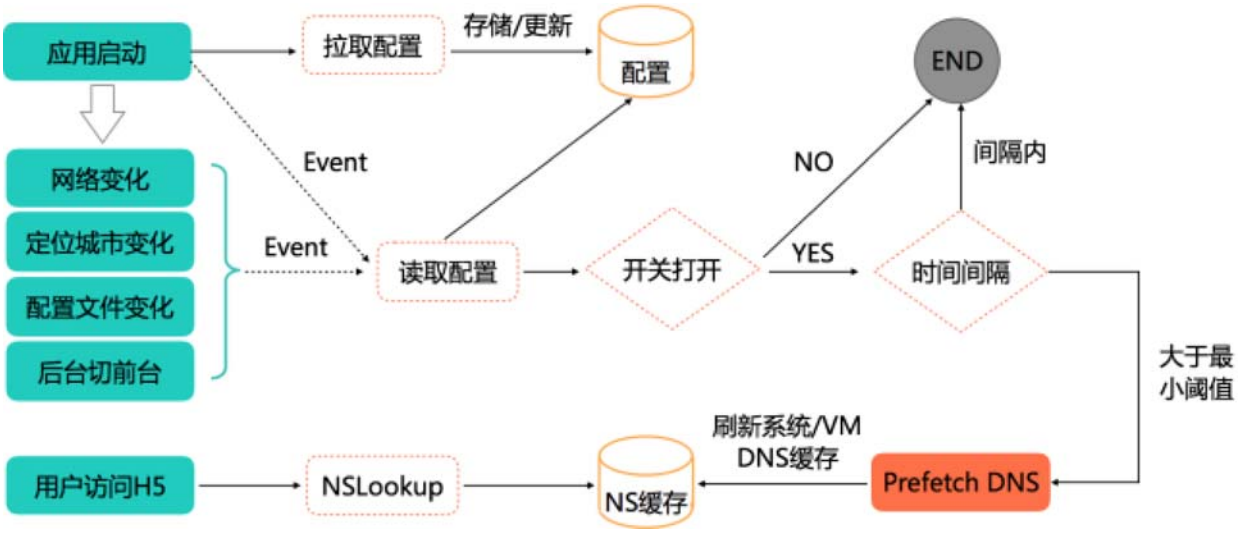
另外 W3C 也提供了 DNS 预读的方案，可以通过在服务器端发送 X-DNS-Prefetch-Control 报头，或是在文档中使用值为 http-equiv 的<meta> 标签：

```
<meta http-equiv="x-dns-prefetch-control" content="on">
```

的方式来打开浏览器的DNS预读取功能，但是该API功能目前在移动端浏览器内核中实现支持的较少。

我们采取的是一种轻量级的方案，如下：

1. 利用APP启动时的config接口，下发DNS Prefetch的配置参数：开关、时间间隔、需要进行prefetch的域名列表等；
2. 监听APP启动、网络变化、定位城市变化、配置文件变化、前后台切换等事件，在独立的线程中执行DNS Prefetch的逻辑；
3. 如果开关打开，且上次prefetch的时间距离当前的时间大于阈值，则刷新DNS，粗发操作系统/VM层的缓存功能。




DNS Prefetch上线后我们海外域名解析时间RT90从350ms下降至250ms左右。

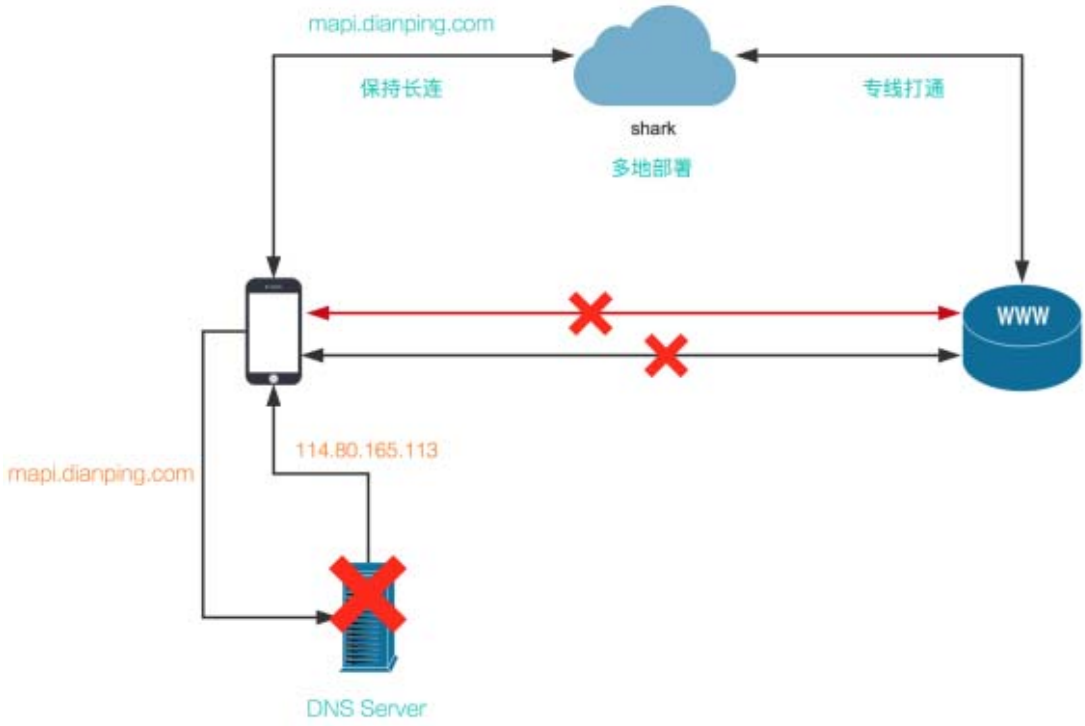
## 长连接



HTTP请求重度依赖DNS，DNS劫持、移动端网络不稳定导致建连失败以及公网链路质量差等因素导致移动端的网络成功率一直不高。HTTP 2.0可以通过SSE、WebSocket等方式与服务端保持长连接，并且可以做到请求多路复用，但HTTP 2.0对运维、前端、后端的改造成本非常高。基于此背景公司架构组推出了Shark，一种“代理长连接”的模式，主要用于解决移动设备网络通信质量差的问题。

- Shark在国内和境外部署了多个接入点，类似于CDN的就近访问，用户可以就近连接到Shark节点；
- Shark各节点的ip会在APP启动时加载到设备，客户端通过“跑马测试”（ping各节点）的方式选择最优节点，建立并保持TCP长连接。
- Shark节点和IDC之间通过内网专线打通，从而保证了网络链路的质量；
- 客户端Shark的网络层会拦截APP内的HTTP请求，通过特定的协议格式将HTTP请求信息转化成TCP包传到Shark节点，Shark节点再将TCP请求“还原”成HTTP，再通过专线请求后端服务。
- 当Shark通道出问题的时候，可以failover到普通的HTTP模式，从而实现高可用

这种“代理长连接”的模式，对后端业务是无感知的，业务无需做任何改造。另外也巧妙的绕开了DNS、公网质量差等问题，极大的提升了native api请求的网络成功率。



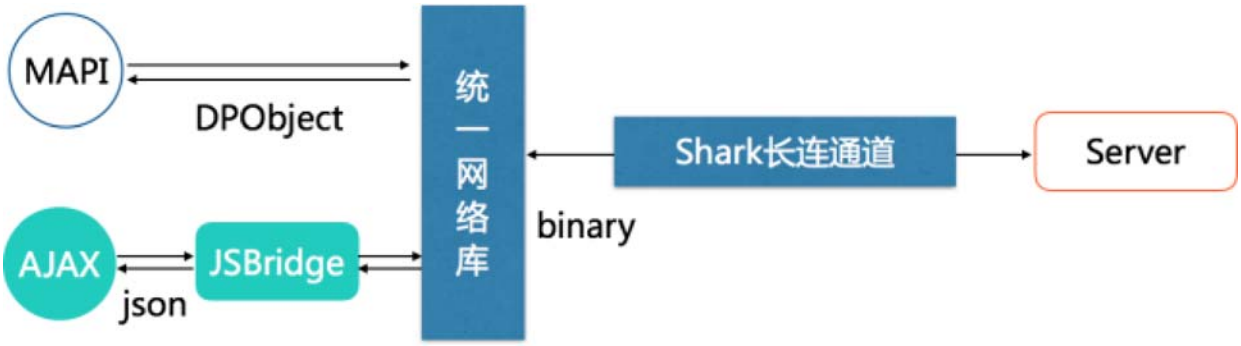

**Ajax 接长连**

目前美团点评大部分的APP都接入了Shark SDK基础网络库，Native API（我们内部叫Mobile API，MAPI）的网路请求由Shark SDK统一解决，使用的是自定义的序列化方式（内部称DPObject，比json效率高）。但对于H5页面中的Ajax请求，是没法直接享受到Shark带来的“福利”的。先看一下Hybrid模式下一次Ajax请求的过程：




上图可以看出，一次普通的Ajax请求会由WebView的内置浏览器内核来发送接受请求，一般是json格式的数据，和PC浏览器的一次HTTP请求过程差别不大，都是要经过DNS、TCP建连以及要面对公网链路差的问题,另外Ajax请求服务复用TCP连接，意味着每次请求都要重新建连。有了MAPI的经验，我们很容易想到，能否像MAPI一样利用长连通道来提升性能呢？

一种方式是在WebView中拦截页面的HTTP请求，在容器层做请求代理并处理序列化反序列化等事情。这种方式对业务比较友好，业务方几乎不需要做什么事情。但WKWebView的限制比较多，所以方案一目前很难推行。另一种方式是通过JsBridge来实现，缺点是对业务侵入性较高，业务方需要手动控制桥API的调用，一期我们选择的是“较笨拙”的方案二。




网络链路优化

**用户境外访问**

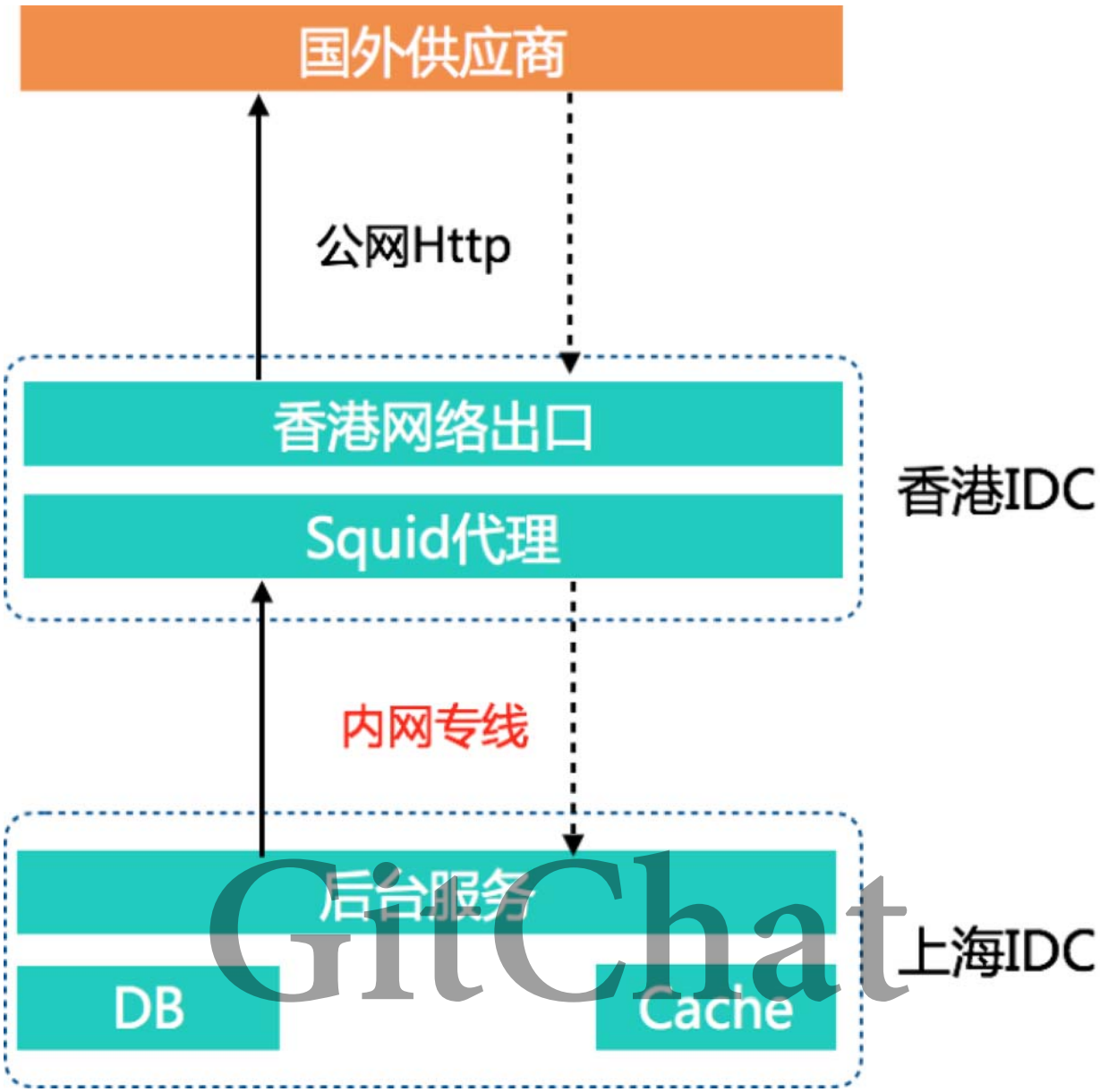
Ajax接长连解决了App内的H5场景，而对于App外的入口场景，如M站、微信朋友圈分享等，我们没法使用到Shark长连接，跨网跨国访问的问题依然存在，这种情况下我们目前的优化主要是“利用专线”：

- 我们的后台应用和数据部署在上海机房，在香港机房内部署了一组SLB（七层负载均衡，基于tenginx实现）。
- 利用专线连上海和香港的机房，解决GFW拦截过滤、跨境网络访问及公网链路差的问题。
- 当用户在境外访问时，智能DNS会解析出香港机房的ip，请求经香港slb走专线转发到境内服务器；而当用户在境外访问时，则直接请求到上海的机房。



**境外直连对接**

另一个场景是，我们和很多的境外供应商有直连对接，通过HTTPClient的方式后端发起调用对方的Open API接口，这种场景优化前接口延迟及网络成功率都非常不理想（很多和国外对接的业务应该都遇到过类似的问题），我们的优化方案是：在香港部署一个正向代理squid，请求先由内网专线转发到香港的squid服务器，再从香港机房的网络出口出去。以与香港迪士尼的对接为例，优化前的 api接口rt95：9s+（迪士尼接口传输的数据非常多），优化后降到2.3s，效果非常明显。

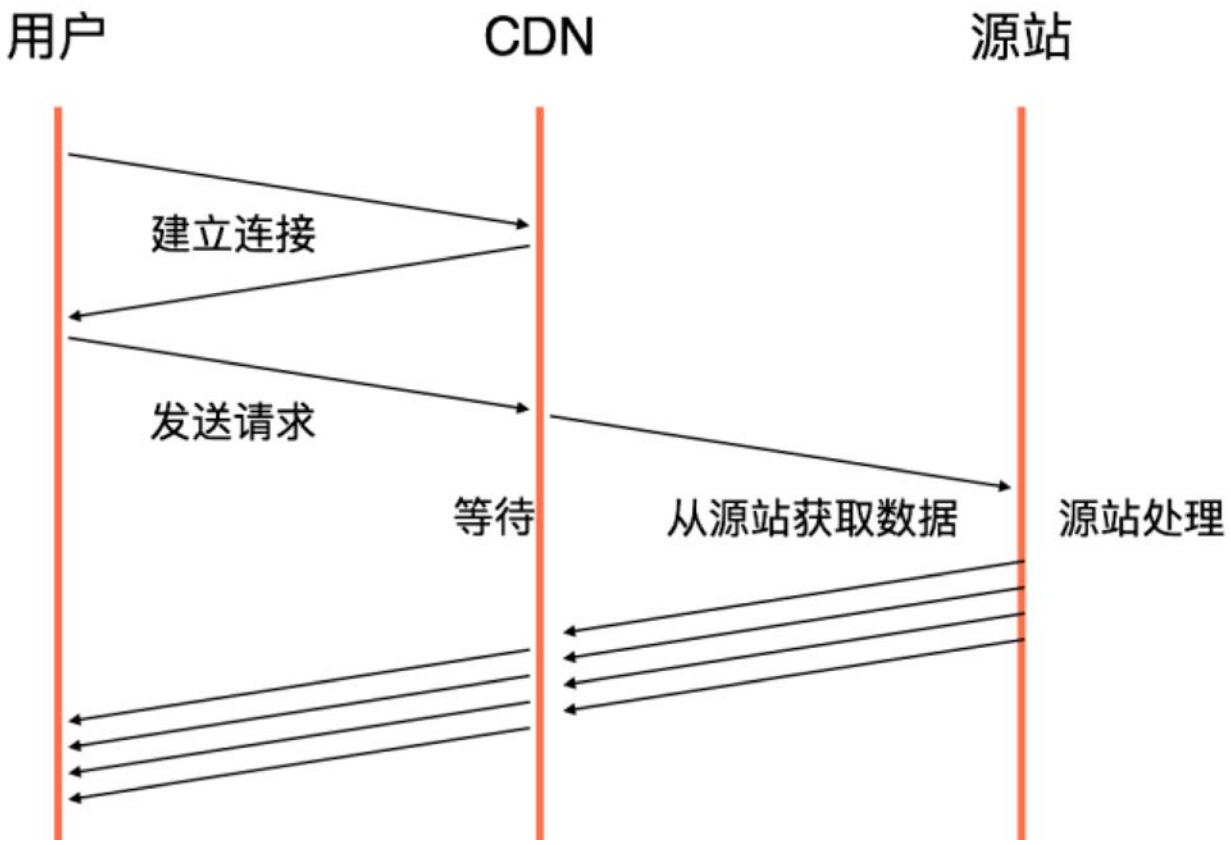





**CDN 动态加速**

除了专线方案，我们还测试 CDN 动态加速。

CDN 不仅可以用来对静态资源做缓存加速，也可以对动态数据接口起加速作用，原理如下：




用户请求耗时=用户和边缘交互的时间 + 1*RTT（边缘到源站） + 源站处理时间。

HTTP 请求中，建连是个非常耗时的事，普通 HTTP 请求 TCP 3次握手要消耗2个RTT时间，如果用户和服务器位置很远，比方说用户在美国，服务在中国，请求走海底光缆的话一次RTT理论值是150ms左右（光波信号传输速度结合物理距离计算得出），实际肯定大于理论值。



CDN 动态加速主要在以下几方面起到优化效果：

1. 用户与服务器的建连改成与CDN边缘节点建连（就近访问），缩短了建连时间，同时也提升了建连成功率。
2. CDN与源站之间通信相比公网网络链路质量有保证。
3. CDN节点和源站的连接可复用。

我们实测下来CDN动态加速在部分国家和地区有明显的加速效果，但整体的效果不够明显，所以最终未投入规模使用。

## 前端优化

前端优化我们主要做的几件事情：

- 前后端分离
- 图片优化
- 域名收敛、减少请求
- 离线化

- 首屏node后端同构渲染

## 前后端分离

在之前的项目中，页面是由Java后端项目中通过FTL模板引擎拼装，前端团队会维护另外一个前端的项目，存放相应的CSS和JS文件，最后通过公司内部的Cortex系统打包发布。

这个流程的问题在于前端对于整个页面入口没有控制力，需要依赖后端的FTL拼装，页面的内容需要更改时，前后端同学就要反复沟通协调，整体效率比较差，容易出错，也不方便实现前端相关的优化。

前后端分离的关键点在于前端拥有完整独立的开发、测试、部署的流程，与后端完全分离。我们把页面的组装完全放置到了前端项目，后端只提供Ajax的接口用于获取和提交数据。前端页面完全静态化，构建完毕之后连同相应的静态资源通过CI直接发布到CDN。这样的好处有：

1. 前后端同学的开发工作解耦，只需要约定好API，两边即可并行开发；
2. 后端API可以做到多端复用，比如PC、H5、M站、小程序等；
3. 前端主文档HTML页面可以利用CDN加速。

## 图片优化



在一些重体验的网页上，图片资源的占比通常较大，一些高清大图动则几十上百K大小。 针对图片这块我们主要做了以下几点优化：

- 图片尺寸按屏幕大小自适应：

原先我们图片的尺寸都是由后端控制，由服务在代码中写死下发给前端，这样带来问题是：

1. 不同设备不同大小的屏幕上使用的图片尺寸一样，无法满足所有设备；
2. 一些没经验的开发同学经常乱用尺寸，比如我们POI列表页的商户头图只需要200 * 200就够了，而新手工程师可能使用的是800 * 800的图片，导致页面加载慢、用户流量白白被浪费且客户端还需要做图片压缩剪裁。

美团云的图片服务提供了实时剪裁功能，后端在下发图片URL时不需要指定尺寸，由客户端根据屏幕尺寸做自适应计算，这样可保证每台设备上的图片都“刚好合适”。

- CDN加速：前面CDN优化章节已介绍，通过接入境外CDN服务商及CDN预热的方式做CDN加速。
- 图片压缩：境外业务内部已在全面使用WebP，经测试WebP格式能够优化图片大小25-50%，清晰度基本没有影响。
- 懒加载：图片资源通常比较大，选用懒加载可有效缩短页面可交互时间。

## 域名收敛、减少请求

域名过多会带来以下问题：

1. DNS解析成本高；
2. CDN加速一般都是按域名来做配置，过多的域名无形增加了CDN接入的成本；
3. 浏览器的并发加载数限制，浏览器对单域名的并发度是有限的，超过限制的请求需要等待串行加载，页面加载速度会变慢。

我们做了以下几件事:

1. 去掉一些直接引入的第三方脚本，将脚本直接打包到我们的代码中（可以利用CDN）；
2. 所有静态资源共用一个域名；
3. 将一些尺寸较小的脚本和CSS构建过程中内联到主文档中。

通过域名收敛/减少请求数，我们商品详情页的页面请求数从8个减少到4个、域名数也减少一半，页面完全加载时间下降了约1000ms。

### 离线化

离线化可以减少网络请求，加速页面加载速度。另外当用户网络不稳定或断网时也可以访问已被缓存的页面和资源，我们先后使用了2种离线化方案：



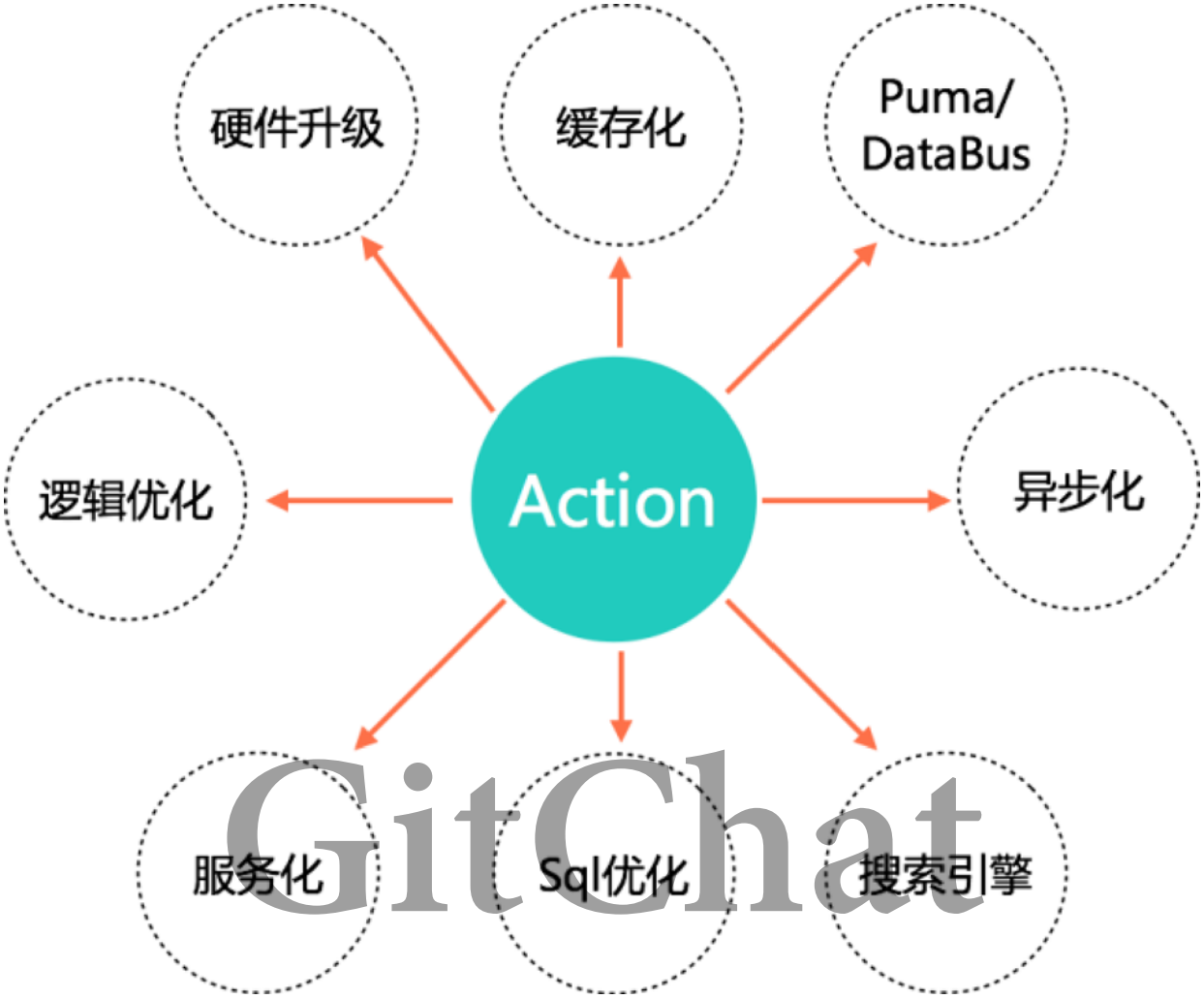
1. AppCache（HTML Application Cache API），在前端项目构建流程中，通过分析页面资源依赖关系，自动生成资源manifest文件，这样就能够确保页面及资源发生变更时，manifest文件内容同步更新。当浏览器监测到manifest文件有更新时，会自动重新下载manifest里面的文件。AppCache的一个缺点是缓存文件会越来越多，缓存不容易清理。AppCache未来会逐步被Service Worker所取代，无论从灵活性还是可扩展性而言，SW都更胜一筹。
2. 目前在使用的是公司平台自研的离线包框架，相比于AppCache，离线包框架在资源更新，离线配置，内存管理等方面都做了很大的改善。另外AppCache对于用户第一次加载页面是没有加速效果的，因为只有第一次访问之后才会把资源缓存下来。而离线包框架则可以做到真正的预加载，它会监听APP正常启动事件，当APP启动后即可开始加载更新离线资源。

### Node 服务端同构渲染

同构渲染，主要用来解决首屏加载的问题。相比于服务端，移动端设备的性能较弱，页面在服务端渲染比在前端渲染会快很多。配合上Ajax长连等网络优化技术，node同构首屏后端渲染提升了首屏加载速度。node同构直出和一开始的java freemark后端渲染最大的区别在于：node项目可由前端同学来维护，用的是前端工程师熟悉的js语言。另外前端生态较好，React、Vue等框架都提高了丰富的渲染模板供前端工程师选择。

## 后端优化

后端优化的思路相对比较比较通用，和境外业务的特点关系性并不大，稳重的前言部分“影响性能的因素”章节有简单描述，本文将不对各种后端优化手段做详细介绍，只挑几件我们做过的事情做下简单介绍。





- 硬件升级

硬件问题对性能的影响不容忽视，早期的时候，我们的一个DB集群经常有慢SQL报警，业务排查下来发现SQL都很简单，该做的索引优化也都做了。后来DBA同学帮忙定位到问题是硬件过旧导致，将机械硬盘升级成固态硬盘之后报警立马消失了，效果立竿见影!

- 缓存化

缓存可以称的上是性能优化的利器，使用缓存时需要考虑缓存命中率、缓存更新、数据一致性、缓存穿透及雪崩、value过大等问题，可以通过mutiGet将多次请求合并一次、异步访问等方式来提升缓存读取的性能。

- 逻辑优化

举个例子，我们的商家系统有一个商家导出历史订单的功能，接口经常超时。排查下来是由于功能上默认是导出商家的全部历史订单，数据量过大导致。而实际业务上几个月之前已完成的订单对商家来说并没有什么意义，如果是要看统计数据我们有专门的统计功能。后来和PM商量后在导出时增加了时间段选择项，默认只会导出最近3个月的订单，超时问题顺利解决。

- 服务化

我们做服务化最基础的是按业务做服务拆分，避免跨业务间的互相影响，数据和服务同时拆分。同一个业务内部我们还按计算密集型/IO密集型的服务拆分、C端/B端服务拆分、核心/非核心服务拆分、高频服务单独部署等原则做拆分。

- 异步化

异步化可以利用线程池、消息队列等方式实现。使用线程池的时候一定要注意核心参数的设置，可以通过监控工具去观测实际创建、活跃、空闲的线程数，结合CPU、内存的使用率情况来做线程池调优。

另一种是通过NIO实现异步化，一切网络IO皆可异步：RPC框架、Servlet3.0提供的异步技术、Apache HttpAsyncClient、缓存异步接口等等。

- 搜索引擎

复杂查询以及一些聚合计算不适合在数据库中做，可以利用搜索引擎来实现，另外搜索引擎还可以帮我们很好的解决跨库、跨数据源检索的场景。

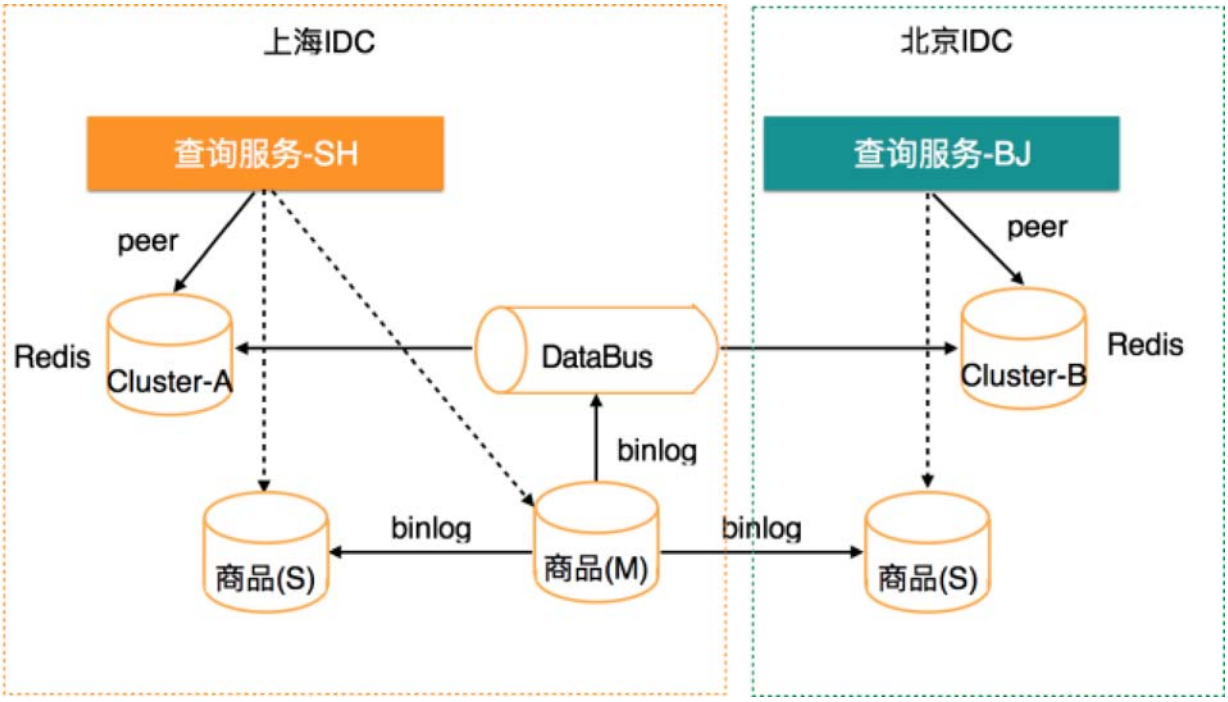
## 服务架构



美团点评有上海、北京两地都有机房，以商品服务为例，我们的商品服务会同时被上海和北京的两侧的应用依赖和调用。一开始我们的服务只部署在上海机房，由于上海、北京两地机房距离间隔较远（北京-上海专线ping延迟30ms左右），当北京的应用调服务时就会有较高的延迟，为此我们做了如下的缓存双集群加异地部署：

1. 商品静态信息全缓存，缓存未命中时再查DB；
2. DB以主从的方式部署，北京机房也部署了一套从库；
3. 缓存数据更新使用DataBus，基于binglog数据同步的方式，保障DB和Redis数据的“准实时”同步；
4. 服务、缓存、DB均两地部署，调用方就近访问（RPC、cache、dal等基础组件支持就近路由）；
5. 缓存更新利用Client的双写模式，Databus监听到数据更新后，消费程序会往两个缓存集群同时写入。

通过这种双缓存加异地部署的“异地多活”模式（实际是异地只读），提升了我们服务在跨地域场景下调用时的性能。




# 总结

结合境外业务特点，本文从网络优化、前端优化、后端优化几个角度介绍了境外业务在性能优化上的一些实践，重点篇幅放在了网络优化部分。性能优化是一个系统性工程，需要RD、FE、SRE的配合协作才能做好。得益于公司强大的高性能前端框架、BGP网络、高性能应用组件、云平台等基础设施，以及在靠谱的运维保障SRE团队、基础架构团队以及平台团队支持下，我们境外业务的性能优化取得了阶段性的成果，后续还要继续努力！




最后，打一波广告，美团点评境外度假业务正在招人，欢迎志同道合的朋友加入我们（简历传送门：**yunshuang.tao#dianping.com**），加入美团旅行，一起来做改变世界的事~~

关注 **美团点评技术团队** 微信公众号，收获更多干货！


GitChat