

高德技术 2020年刊合辑

CODE A BETTER TRAVEL

智慧出行最佳技术实践和总结，覆盖大前端、算法
汽车工程、架构、质量、顶会论文……



— 高德技术出品 —

序

年味逾浓，春节渐近。

高德技术年刊如期而至。

过去的一年里，高德业务快速发展，国庆出行节的日活跃用户数突破1.5亿，又创新高。高德技术人在支撑业务快速发展的同时也在持续创新：导航个性化、引擎服务化、基建架构升级、全面上云、共享打车、信息服务、汽车前后装、大云图等方面实现新的突破，车道级导航、高精地图落地等领域取得行业领先，车载环境下的手机定位技术获得国际顶赛冠军……在这些过程中，我们做了大量的技术实践总结和思考，并以文章等形式沉淀下来，与大家一起分享成功的经验和踩坑的教训。

在2021年春节即将到来之际，我们精选了几十篇高德技术的干货文章及数篇国际顶会论文，制作成了一本厚达750页+的电子书，作为新年礼物赠给大家。

这本电子书内容覆盖了大前端、算法、架构、汽车工程、质量等多个领域，以及数篇高德入选顶会论文的解读，希望能对大家拓展技术思路有所帮助。

如果大家有意愿针对书中的技术问题深入探讨，想与我们做更多交流，或者有更好的建议，欢迎扫描下方高德技术公众号二维码与我们联系。

希望大家喜欢本书，欢迎推荐给身边感兴趣的朋友。

衷心感谢大家一直以来的支持和陪伴！

最后，祝大家牛年快乐，平安健康，阖家幸福。

高德技术微信公众号（amap_tech）



分享来自于高德技术的原创文章，发布技术活动、组织文化、热招岗位信息，和技术圈小伙伴一起学习成长。

关注高德技术微信公众号

回复“2020 年刊” 获取 2020 高德技术年刊电子书

目录

序	2
大前端篇	8
高德前端这五年：动态化技术的研发历程和全面落地实践	9
高德智慧景区随身听播放器框架设计与实现	25
高德最佳实践：Serverless 规模化落地有哪些价值？	35
3 倍+提升，高德地图极致性能优化之路	43
高德 APP 启动耗时剖析与优化实践（iOS 篇）	56
iOS 代码染色原理及技术实践	93
Android 端代码染色原理及技术实践	111
八个维度对低代码能力度量模型的思考	139
以高德为例，超级 APP 启动提速的实践和思考	152
高德地图驾车导航内存优化原理与实战	199
高德智慧交通地图空间可视化 SDK 设计与实现	214
地图建筑群的光影效果原理和应用实践	234
算法篇	247
导航定位向高精定位的演进与实践	248

关于卫星定位，你想知道的一切.....	262
业内首发车道级导航背后——详解高精定位技术演进.....	286
地图兴趣点聚合算法的探索与实践.....	303
卫星影像识别技术在高德数据建设中的探索与实践.....	340
高德地图首席科学家任小枫：高精算法推动高精地图落地....	350
揭秘！文字识别在高德地图数据生产中的演进.....	361
高德算法工程一体化实践和思考.....	379
深度学习在高德 POI 鲜活度提升中的演进.....	389
混合时空图卷积网络：能“推导”未来路况的智能算法.....	404
高德 AR & 车道级导航技术演进与实践.....	416
深度学习在高德 ETA 应用的探索与实践.....	425
机器学习在高德地图轨迹分类的探索和应用.....	432
高德地理位置兴趣点现势性增强演进之路.....	448
漫话地图之高精地图生产中的坐标系.....	467
盘点 有哪些大数据处理工具？	484
任小枫 QA 答疑汇总 视觉+地图技术有哪些新玩法？	505
汽车工程篇.....	517
高德车载导航自研图片格式的探索和实践.....	518
高德车载导航的差分更新优化实践.....	529

远程调试在 Linux 车机中的应用 538

浅析云控平台画面传输的视频流方案 545

基于 Rust 的 Android Native 内存分析方案 551

高德车载导航 Android 平台 DR 回放技术方案 569

架构篇 580

高德深度信息接入的平台化演进 581

高德云图异步反应式技术架构探索和实践 595

高德 SD 地图数据生产自动化技术的路线与实践（道路篇） 611

质量篇 621

高德技术评测建设之路 622

高德全链路压测——语料智能化演进之路 636

高德全链路压测——精准控压的建设实践 650

如何规范你的 Git commit? 664

单元测试在高德在线导航业务中的实践 675

顶会论文篇 687

KDD 论文解读 | 混合时空图卷积网络：更精准的时空预测模型
..... 688

CIKM 论文解读 | 破解高架区域偏航检测难题，高德提出工业级
轻量模型 ERNet 709

大前端篇

高德前端这五年：动态化技术的研发历程和全面落地实践

作者：北萧

前言

2015 年–2020 年，历经 5 年发展，高德地图应用开发前端团队在业务快速发展中不断成长。一路走来，从小团队主要负责短期运营活动开发的散兵游勇，到现在团队规模 100 人+，覆盖高德 5 大业务线，上百个模块的坚甲利兵。本文将分享随着业务快速增长高德前端的技术发展历程，总结动态化技术的落地实践，以及高德前端未来的发展方向。

高德（应用开发）前端技术的发展按照时间线来看，大致可以分为 4 个阶段：

- 2015 年，业务上大量拉新的诉求，活动需求暴增，应用前端开始登上高德技术大舞台。
- 2016 年 – 2017 年，业务高速发展，对于效率以及双端一致性的诉求，带来了前端发展的契机，动态化技术开始落地。

- 2017 年 – 2019 年，动态化在高德全面落地，前端开发的角色越来越重要，业务半径不断延展。
- 2019 年 – 至今，这是目前的发展阶段，更加关注支撑的稳定性和延展性，让业务更好的活在未来。

一言蔽之就是“顺势而为，乘风破浪”。



2015 年：小荷才露尖尖角

2014 年底，高德地图提出专注用户需求，专注做地图导航产品和导航产品的技术研发，未来三年无商业化目标的新战略。没有了商业化的压力，一心专注产品和用户体验的高德地图，技术就此踏上了高速发展的轨道。

运营活动开发需求暴增，“工程、效率”解题

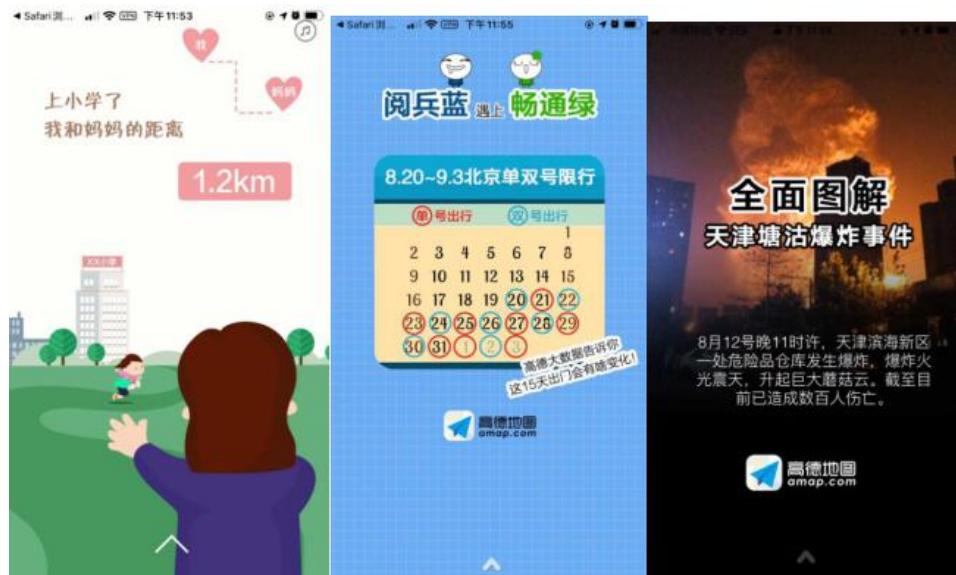
围绕促进日活和留存的战略，大量运营活动的开发需求应运而

生，这个阶段的活动特点是“短平快”，开发周期短，一周内交付验证，活动时效期过后即可下线，完全不需要维护。对于代码的可维护性、技术上的创新要求不高，目标是快速响应热点事件，完成活动开发。

这种模式给当时人员稀少的前端团队带来了非常大的考验，因为很多运营活动搭建需要在前端完成。而此时的前端团队在技术沉淀上较为薄弱，重复劳动明显。基于此，我们首先要完成的是效率上的提升，主要工作包括：

- **组件化**：和运营同学一起规范、建设活动常用组件。
- **模版化**：拼图，通过模版化解决简单页面的搭建问题。
- **流程化**：CLI 加速工作流。

正是在基础能力上的耕耘，在之后多个热点事件时，我们才能游刃有余，在短时间内完成业务开发、上线。



十一全民出行节，“性能、体验”沉淀

时间回到 2015 年 10 月 1 日，为了配合“十一全民出行节”，第一个大型运营活动“十一挖宝”就此诞生，也拉开了每年出行高峰必有大型活动的序幕。我们通过开发更加简单、有趣的交互设计提升用户的游戏体验，强烈的社交属性例如 PK，排行榜等促进用户之间传播。

这一年的活动在公司内外进行了大范围的运营推广，“寻宝嘉年华，十六台 Smart 汽车送不停”，“斗鱼主播全场直播挖宝”小伙伴们应该还有印象，当时直播间非常火热，我们却战战兢兢，如履薄冰，特别担心直播时出现卡顿、白屏等问题，把火热的“全网挖宝”变成全网大型吐槽节目，运气比较好，我们担心的事情没有发生。

尽管如此，后背发凉的回忆使我们意识到，技术上如何完善复杂

游戏的性能体验必将成为日后的课题，基于此我们又完成了基础技术（体验、性能）能力上的沉淀，包括：音频语音交互解决方案，大型游戏性能的最佳实践。



到 2015 年底整个前端团队初具雏形，团队开始建立规范化，标准化，体系化的思维，在技术上也积攒了不少家底。为了应对可预见的考验，前端团队也招入了很多有能力的新人。正是这些人才，使我们在接下来的多线作战中游刃有余。

2016 年 – 2017 年：忽如一夜春风来

随着高德地图业务沿着扩品类、在垂直品类做精做细，景区、酒

店、银行商铺、充电桩等个性化定制需求凸显，对前端展现提出了更高的要求，对“快速应变”要求也越来越高，这段时间主要面临以下痛点：

- 业务要求快速发版试错。
- 研发资源越来越无法满足业务的快速增长。

契机，高德动态化技术诞生

这些问题也在不断地督促我们去反思，到底有没有一种架构既能像 H5 一样快速的开发、发布又能保持原生 Native 的体验？实际上，在 2015 年，我们就开始做动态化了，那时候业内有 React Native，团队做了技术调研，发现不能完全满足业务上的需要，尤其是性能方面，因此我们决定自研一套动态化技术。在项目伊始就有一些难点摆在我面前：

- 布局怎么做？RN 的 yoga、iOS 的绝对布局还是 Android 的 RelativeLayout？
- Runtime framework 放在哪里？C++、JS 谁来承载？

- 模块化的机制是什么样的？Node Require、Webpack
Require？
- 通信、动画怎么做等诸多问题需要我们探索，抉择。

经过团队内部多次思想上的碰撞、激烈的讨论，最终确定以下核心设计思想。

核心

核心处理尽量下沉动态化引擎层，双端尽量做薄，动态化引擎（C++）以 Webkit、Node 为参考，即可以通过 HTML、CSS、JavaScript 编写原生应用，又可以像 Nodejs 一样使用文件操作等与原生应用的交互能力。这样的设计在上层对接前端生态时更加灵活，在处理复杂、频繁交互的大型页面时也会有更好的性能。

优化

除常规动画外，还设计了关联动画解决高频联动动画，关联动画本质上并非是一种播放类型的动效，是基于观察者模式设计的，被观察者的属性变化会影响观察者的属性变化，它将关联关系提前绑定好，一次性由 JS 线程传递给 UI 线程，这样能够很好的保证交互性能。

在方案明确后，整个团队也投入到能力建设中来，尽管每周都在发现问题、解决问题中艰难前行，但大家仍然乐此不疲，对于这种打怪升级的过程乐在其中。在基础能力、辅助工具齐备的背景下，我们开始着手动态化业务的落地实践，最终我们选择了 POI 业务。POI 即 (Point of Interest) 兴趣点，如学校、酒店、饭店、加油站、超市等，高德地图上有数千万的 POI。

起航，动态化技术落地 POI



首先看一看 POI 业务的特点：

- UI 复杂，多品类，多种多样的展现形式。

- 与地图有存在交互。
- 性能要求高，长列表，数据量大。
- 富交互，大量手势交互，关联动画。

多人协作开发问题

为了快速验证能力，项目的排期非常紧张，为此前端同学 All in，业务上看尽管 POI 只有一个页面，但是却有多个行业，而行业是由多个模块拼接而成，每个模块在不同行业展现形式也不尽相同，如何解决协作问题就成为项目成败的关键之一。

为此我们完成了 Framework 框架开发，可以用 JSX 语法实现基本组件化，在组件这个级别进行 CURD 解决了模块化开发的问题。

调试问题

在项目之初我们并不存在完备的调试方式，甚至可以说不存在调试能力，只是通过 `print` 将 `log` 输出到手机端展现。这在开发 POI 时遇到了极大的问题，业务场景复杂大量实时日志无法查看，导致效率极低。

为此我们完成了 websdk, mock 能力，在浏览器端完成了 POI 页面的预览，调试。

尽管 POI 落地过程中，遇到了各种各样的问题，但结果是美好的，动态化技术也经受住了业务的考验，新的 POI 不仅完全覆盖了之前 H5 的功能，在手势动画、List 展现上还体现了更加卓越的交互体验和性能。伴随着业务上线，基建一期也基本完成，这个阶段以满足业务为中心，主要围绕支撑能力的设计和基本的开发体验。



POI 的圆满落地也标志着前端技术有能力在高德地图中承担更复杂、更核心、更大的业务场景，前端开发也即将迎来春天。

2017 年 – 2019 年：千树万树梨花开

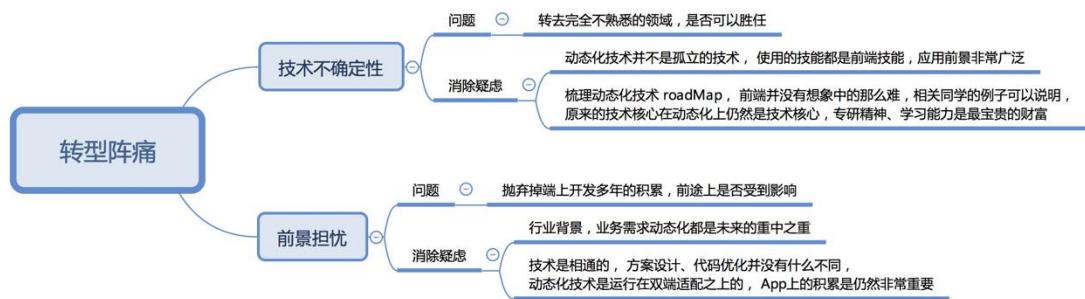
POI 业务上的成功落地，标志着动态化技术解决方案趋于稳定，可以应对各种各样的复杂业务，随之而来的是大量业务的考验。

随着动态化技术应用的深入、主要业务模块的全面接入，支撑能力不完善、动态化技术开发人员缺乏导致改造压力越来越大。

团队壮大，“小前端”到“大前端”

人员的问题主要从内外两方面解决，外部启动招聘，大量吸纳有相关背景的前端同学。内部 Native 同学加强技术培训，转向动态化技术开发也正式提上日程。

不少同学一定有这样的经历，如果让其去调研一门新的技术大家一定非常乐意，充满干劲，对未知领域的探索，求知渴望是研发的共同点。不过如果让其持续朝着这门技术发展就会有非常多的疑虑。

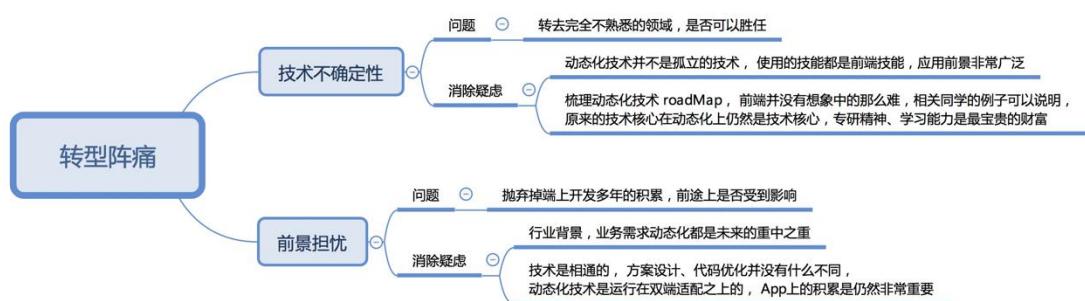


通过培训等方式，解决大家对于技术不确定性和前景的担忧，大量同学开始转向动态化技术，到 2019 年初整个动态化“大前端”团队得到快速增加。

基建完善，“研发闭环，逐个突破”

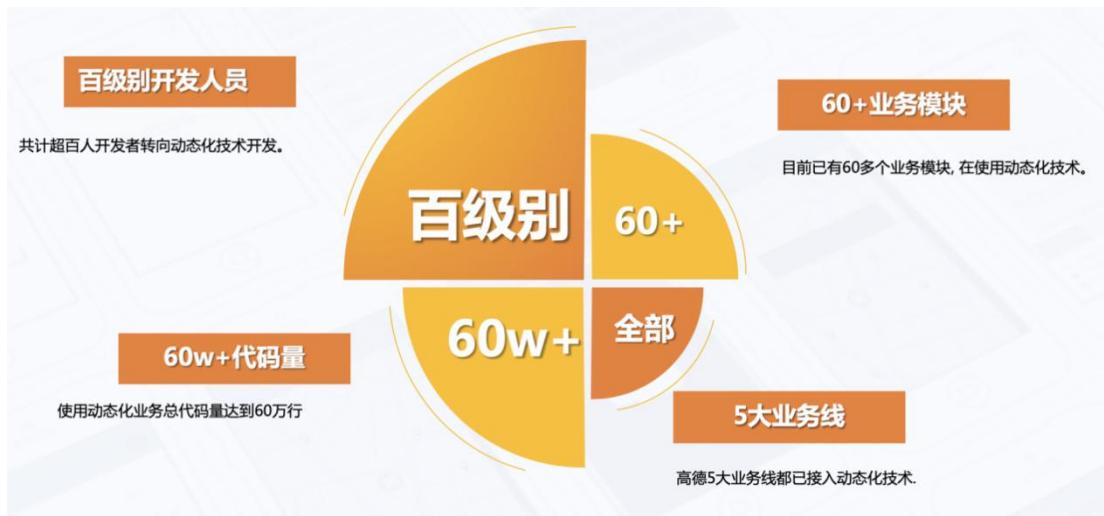


- 通过 IDE 将开发、调试能力打通。
- 通过工程平台进行发布、回滚、更新、监控、分析操作。



业务增长，“横跨 5 大业务线，高德核心业务全覆盖”

富有战斗力的团队、渐渐完善的基建使业务从小步慢走到大步快跑，从用户的核心诉求我在哪（主图，定位），我去哪（搜索，POI），怎么去（规划，导航）到用户的延伸诉求怎么去（打车）怎么玩（景区）高德 5 大业务线 60 多个模块全面接入动态化技术。



这个阶段动态化业务发展迅猛，“前端团队”不断壮大，由“小前端”转变为“大前端”，基建方面也是围绕业务全面展开，不断完善。

有了稳定的开发环境，2017 年 – 2018 年，不到 2 年的时间我们完成代码量从 3W 到 60W，模块数量从 1 到 60+，开发人员井喷式增长。业务发版频次渐渐加快、加密，从单月版→快迭双周版。

2019 年 – 至今：九层之台，起于垒土

面对着越来越复杂的业务，仍有不少细节问题需要进一步解决，如何更好的为业务赋能再次成为重点，阿里前端大咖玉伯之前的分享中有句话给我们印象颇深：愿等花开，坚持长期主义，要快，但不能急。

回到自身来说，前面几年都是保证业务赢在当下，支撑上都是大刀阔斧快速建设，完成 0 到 1 的过程。接下来应该帮助业务更好的活在未来，在当前基础能力具备的情况下，需要闭关审视自身，从功能的完整性，延展性等方面做到精细化。

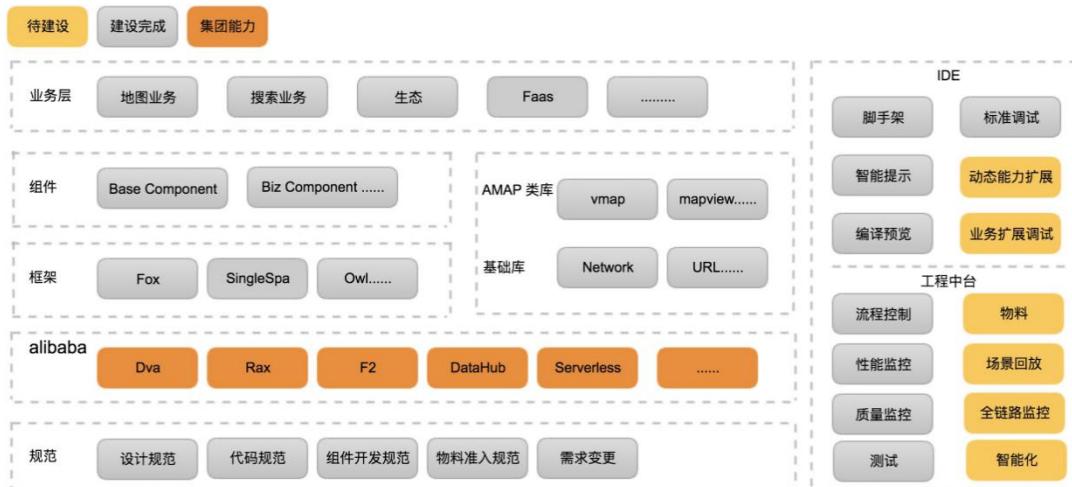
未来，我们也将从研发生态、工程中台、智能化 3 大方向上以精细化、标准化、差异化为基础要求，不断补足能力，逐渐完成中台化、智能化的基础建设工作，围绕 IDE 打造更好的一站式场景化开发体验。

五化基建方针

- 精细化：切中痛点，系统解题。
- 标准化：集团生态、业界标准。
- 差异化：标准化基础上，提供 Amap 能力扩展。

- 智能化：低代码，物料复用，UI 自动生成。
- 中台化：前台通用能力下沉到中台，不再局限高德。

技术大图



接下来的重点方向

- 工具链路稳定性、延展性持续优化。
- 平台能力中台化。
- 全链路监控：快速分析、定位问题。
- 物料：缩短开发到资源路径，沉淀更多基础能力。

- 智能化：低代码、零代码。

以上是 5 年来高德地图前端技术的发展历程，过程中有失有得，我们还在路上，未来会更加努力，让出行更美好。

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德智慧景区随身听播放器框架设计与实现

作者：王玉鹏

一、背景

“远看山有色，近听水‘有’声”，景区语音导览是智慧景区重点业务之一，以用地图可以边走边听景区各景点的语音介绍为主要诉求，实现高德智慧景区地图不仅可以看，还可以听，从而使用户交互体验得到跨越式提高。

我们想要让“技术有温度”，让讲解更加有感情和内涵，最好可以通过讲解构造一个“UGC 景区讲解生态圈”，并且还能帮助讲解创作者有一定的收益，以达到“生态圈的正向循环”，让线上导游“天下没有难做的生意”。

试想一下，当游客走进故宫，这时，高德地图的语音包可以播放：“故宫有 180 万件宝贝，青铜馆、陶瓷馆……”这段话的讲解人，是著名收藏家、古董鉴赏家马未都，是不是更加吸引你关注？另外，当你漫步到延禧宫，语音包则会立刻讲一讲延禧宫与大热的电视剧《延禧攻略》有什么关系，并且有背景音插入，是多么生动形象。

所以，我们开发选型并没有采用传统的 TTS 技术（由文本内容生成机器语音），而是采用了更加通用音频格式(比如 mp3)，作为讲解的音频输入源，方便讲解者进行二次创作。本文将简单回顾高德智慧景区随身听播放器的框架设计与实现。

二、架构设计前思考

“夫未战而庙算胜者，得算多也；未战而庙算不胜者，得算少也”，拉开战斗序幕之前我们应该尽量去“庙算”，提前预防和判断并保证技术风险可控，俗称“防火”。“防火”更能看出本事，而“救火”只是能力。开发应尽量做到“不打无准备之仗”。

首先，如何提升开发和后续迭代效率？此问题涉及到是纯 Native 开发还是用跨平台混合技术开发。如果用纯 Native，双端开发人力可能会使工作量翻倍，后期可维护性也差，经常需要双端同步拉齐。但纯 Native 开发声音相关的技术方案成熟且风险较小。而用跨平台混合技术开发，优点和缺点正好与单纯 Native 开发相反。

经过小组多次技术讨论，看长远利益，最终确定用跨平台技术方案，用该方案虽然技术挑战和风险大（比如需要和

跨平台架构支撑团队一起“无中生有”的去打通 JS 的播放链路和各种音频中断能力回调等），但这个方案有个强有力的好处，就是可以“Write Once，Run Everywhere”（这里的 Everywhere 主要是指移动端操作系统），这样可以天然的拉齐双端业务代码能力，大大节约开发周期和人力，对业务快速功能迭代很有优势，再苦再累再难也值得为此努力。

其次，**如何节省 CPU 和内存资源？**做移动开发的同学都知道，音频播放是耗系统软硬件资源的（比如 CPU、内存还有电量等），另外音频播放不仅仅是涉及到单个 App 的事情，还涉及到第三方 App 音频播放的影响（比如系统来电声音焦点抢占，其他音乐 App 播放焦点抢占问题等）。

所以，业务层开发，要对底层播放器提供的播放能力进行二次封装，一是要控制播放器实例的随意创建。二是要处理各第三方 App 的音频播放焦点的申请和释放等逻辑业务。由此可见，搭建一个通用的业务播放器框架势在必行，受益良多。

再次，**如何使业务与音频本身的播放框架能力隔离？**业务多变，而音频播放能力相对来说是稳定的，其基本能力包括但不限于（首次&续接）播放，暂停，抢占，打断，音

量调节（渐渐变强），物理（如耳机）按键响应，打断后场景恢复，缓存，预加载，强弱网络和播放异常等。

这些音频本身的技术能力，最好应该是和纯业务是解耦的，尽量做到“高内聚，低耦合”。

经过深思熟虑，我们认为设计模式中的“ObserverPattern 观察者模式”，比较切合这一技术背景。纯业务和音频框架本身制定通用的接口协议。

在纯业务自由注册监听器到音频播放框架中，根据关心的回调事件自由处理自己的业务，而音频框架本身只做主要的焦点抢占，现场恢复和事件分发等事情，非常符合 SRP 原则（单一职责），后续调试和维护都很方便。

最后，如何实现跨 Page 播放能力？如下图所示：



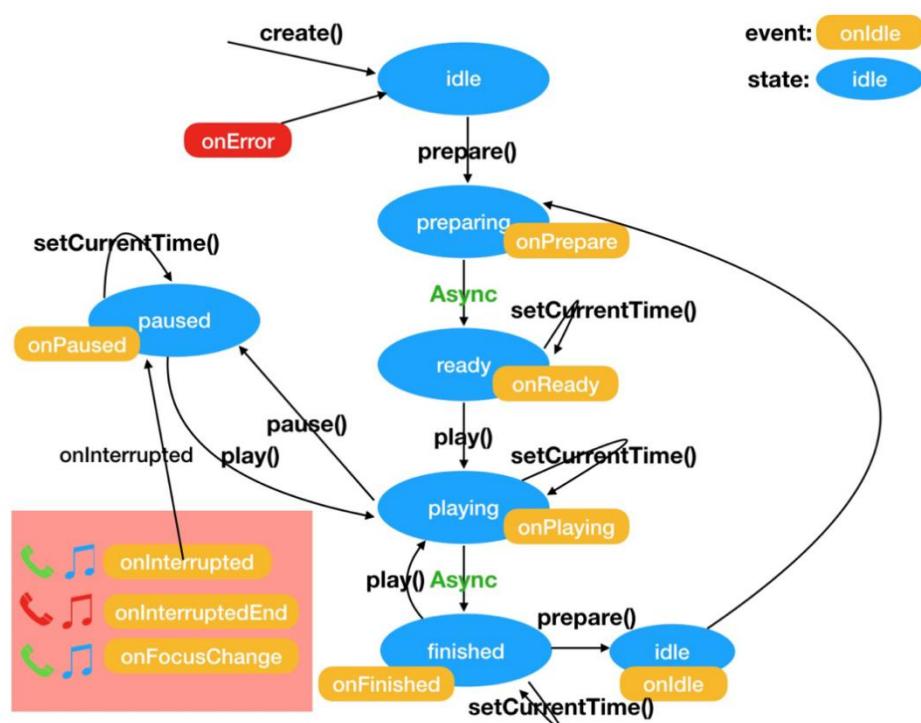
随身听很多业务是有跨 Page 播放要求的，如果将播放能力直接提供出来，由各个页面的 Page 自己维护，势必会生出很多的 Audio，混乱而且页面相互通信交换信息成本高。后经过讨论，就有了如下图的架构方式设计：



结合跨平台底层播放器的特性，虚拟出来一个 BizService 放在跨平台框架的 Service 容器（和安卓里面的 Service 概念差不多，提供一个无界面的可以处理公共业务的容器）里面，处理 Page 页面业务管理和信息交换以及缓存管理，BizService 只和 BizVoiceMediaCenter 交互管理音频数据，也就是说 BizVoiceMediaCenter 是通用播放器框架对外一个“门面”（Facade 门面设计模式）。BizVoiceMediaCenter 里面有且仅有一个 VoiceMediaAlbum 实例（播放专辑，提供“上一曲”，“下一曲”，顺序播放，续播等能力）。

三、架构设计和开发

首先，我们先简单看下跨平台底层播放器的生命周期，如下图所示：



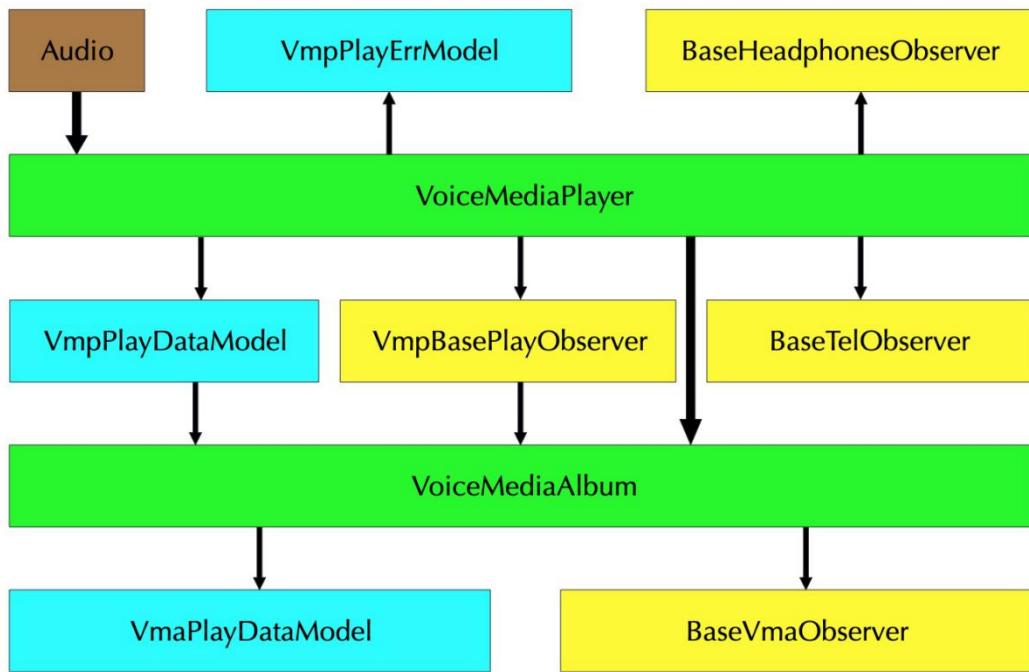
熟悉 Native 开发的同学应该知道，跨平台底层播放器的架构和生命周期，和 Android 本身系统播放器非常相似，差异点是音频焦点被抢占和恢复的回调部分，iOS 设备是 `onInterrupted`，当音频被其他应用打断开始时回调，如电话铃声响起触发此回调（在此回调中保存播放器状态，以便在 `onInterruptedEnd` 回调中恢复播放）。

`onInterruptedEnd`，当音频被其他应用打断结束时回调，如挂断后触发此回调。

而 Android 是 `onFocusChanged`，当音频焦点变化后回调。当然还有其它一些细微差别，比如双端，播放错误码不一致，播放异常超时逻辑不一致等。

但这些都可以通过在业务层构建自己 `VoiceMediaPlayer` 来拉齐以及处理通用音频焦点抢占和丢失场景的逻辑。

通过上面分析，我们可以大体搭出如下图业务播放器的整体框架图(图中箭头表示数据流的方向)。

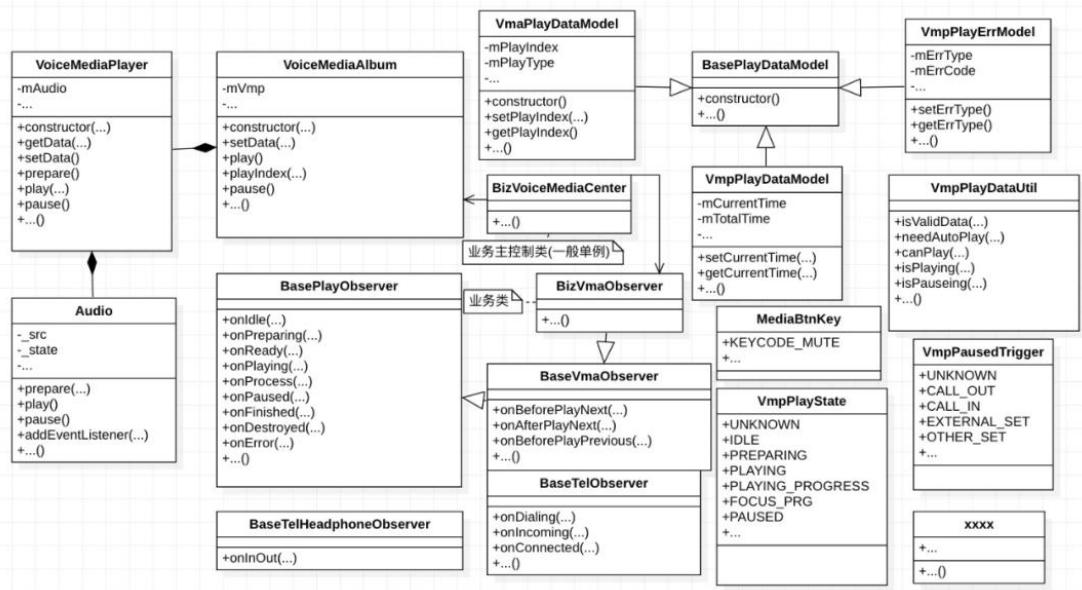


我们可以很容易的看出，业务对跨平台底层播放器 Audio 进行了二次封装为 VoiceMediaPlayer，拉齐和处理通用业务场景（比如抢焦点，播放，现场恢复，播放异常，蓝牙或耳机物理按键响应等）。

VoiceMediaPlayer 再上层是 VoiceMediaAlbum（播放专辑）， VoiceMediaAlbum 专辑类，主要是处理顺序播放，上一曲，下一曲，整个专辑播放事件（单曲播放信息和进度，整体播放进度透出，自动切换顺序，循环或业务指定下一曲播放等）， VoiceMediaAlbum 和业务层的 BizVoiceMediaCenter 打交道，当然 BizVoiceMediaCenter 也可以直接和 VoiceMediaPlayer 打交道，但我们一般不建议这么做，即便

是就播放一首音频，我们也希望，把这首音频当成一个专辑来包装和调用（随身听业务也确实是这么做的），这样更加规范和方便以后扩展。

最后，我们来看看整体架构的详细类设计图，如下图所示：



四、落地产出

高德智慧景区随身听播放器框架完成后，很好的支撑了随身听后续版本的开发。此外，后续因业务需求对产品做了多次迭代和变更，但播放器的架构几乎不需要做很大调整和升级（即使后面又增加了离线播放能力），很好验证了其稳定性和可扩展能力。

下面一系列图，我们可以看出这颗“种子”(景区随身听播放器框架)，开出的美丽的“花”，如下图所示：



以上各个页面底层都共用了这个播放器框架，很方便的实现了音频的跨页面播放和管理，以及异常中断的统一处理。高效满足了相关音频业务的播放能力要求，也为高德智慧景区随身听业务后续迭代开发打下了坚实的地基。

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德最佳实践：Serverless 规模化落地有哪些 价值？

作者：以燃

1. 导读

曾经看上去很美、一直被观望的 Serverless，现已逐渐进入落地的阶段。今年的“十一出行节”，高德在核心业务规模化落地 Serverless，由 Serverless 支撑的业务在流量高峰期的表现十分优秀。传统应用也能带来同样的体验，那么 Serverless 的差异化价值又是什么呢？本文分享高德 Serverless 规模化落地背后的实践总结。

随着 Serverless 概念的进一步普及，开发者已经从观望状态进入尝试阶段，更多的落地场景也在不断解锁。“Serverless 只适合小场景吗？”、“只能被事件驱动吗？”这些早期对 Serverless 的质疑正在逐渐消散，用户正在更多的核心场景中，开始采用 Serverless 技术达到提效、弹性、成本优化等目的。

作为地图应用的领导者，高德为带给用户更好的出行体验，不断在新技术领域进行探索，在核心业务规模化落地 Serverless，现已取得显著成效。

2020 年的“十一出行节”期间，高德地图再创新记录，当日活跃用户数突破 1 亿的时间比 2019 年 10 月 1 日提前 3 个多小时。

期间，Serverless 作为其中一个核心技术场景，平稳扛住了流量高峰期的考验。值得一提的是，由 Serverless 支撑的业务在流量高峰期的表现十分优秀，每分钟函数调用量接近两百万次。

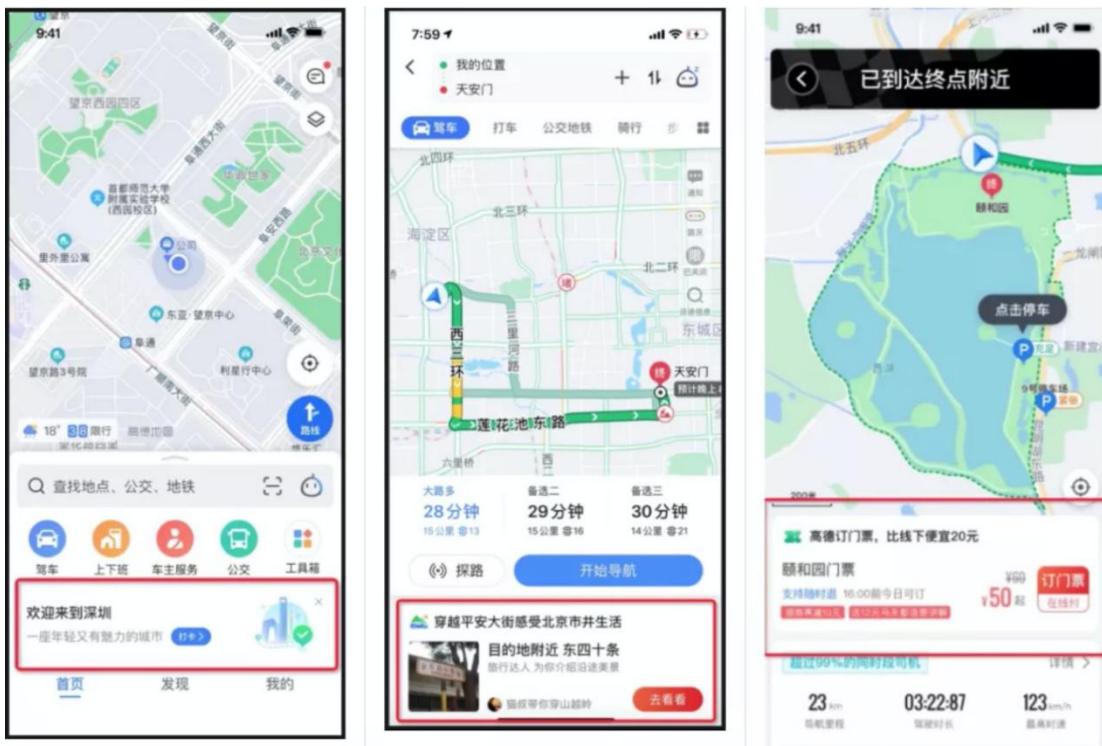
这再次验证了 Serverless 基础技术的价值，进一步拓展了技术场景。

2. 业务场景

自主出行是高德地图的核心业务，涉及到用户出行相关的功能诉求，承载了高德地图 APP 内最大的用户流量。

下图为自主出行核心业务中应用 Node FaaS 的部分场景，从左至右依次为：主图场景页、路线规划页、导航结束页。

大前端篇-高德最佳实践：Serverless 规模化落地有哪些价值？

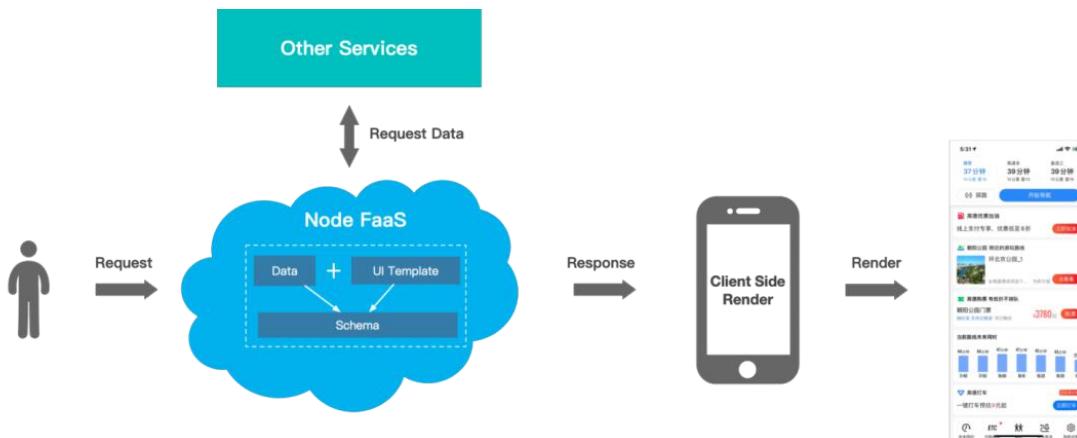


随着功能的进一步拓展，高德地图从导航工具升级为出行服务平台和生活信息服务入口，进一步拓展了出行相关的生活信息服务场景，带给用户更全面的用户体验。上图功能为场景推荐卡片，旨在根据用户出行意图推荐信息，提升用户出行体验。此功能需具备快速迭代，样式调整高灵活性的能力。

因此，将卡片样式模版存放于云端，通过服务下发的形式渲染至客户端无疑为最优选择，可以满足业务快速灵活迭代的目的。

经过方案评估判断，此场景类型属于无状态服务，基于阿里云 Serverless 成熟的生态，高德最终选择接入 Node FaaS（阿里云函数计算）服务能力，出行前端搭建了场景推荐卡片服务。卡片

的 UI 模版获取、数据请求聚合&逻辑处理、拼接生成 Schema 的能力均在 FaaS 层得到实现，客户端根据服务下发的 Schema 直接渲染展示，达到更加轻便灵活的目标。



那么，Serverless 场景在“十一出行节”峰值场景中的具体表现如何？

整体服务成功率均大于 99.99%，总计 100W+ 次触发/分钟，QPS 2W+，各场景的服务平均响应时间均在 60ms 以下，服务稳定性超出预期。

3. 业务价值

从对以上业务场景的支撑中，我们可以看出 Serverless 的表现非常优秀。当然你也会问，传统的应用也能带来同样的体验，那么 Serverless 的差异化价值又是什么呢？

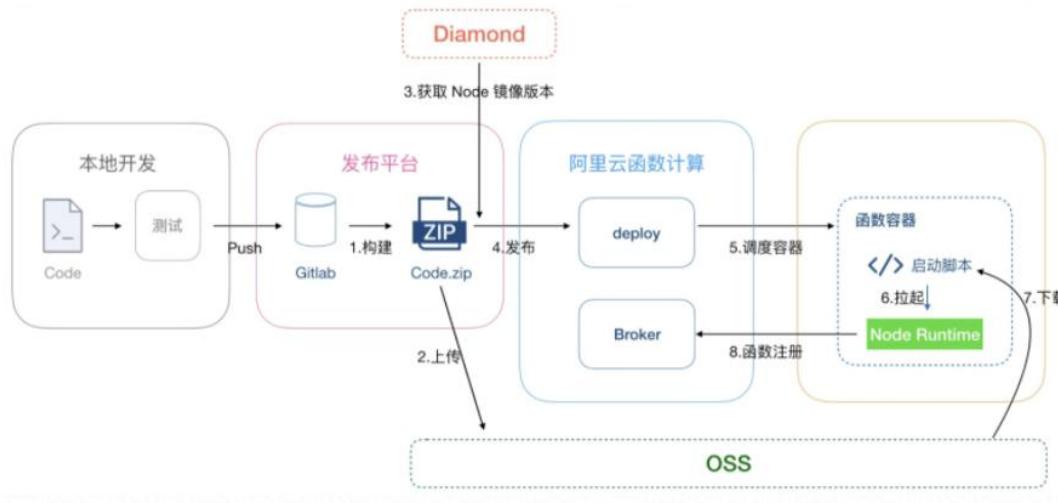
简单提效

传统 BFF（Back-end For Front-end）层应用会随着时间推移以及业务需求的增加，其 BFF 层也会逐渐的变“富”，冗余的代码会逐渐变多，最后就会变成开发者的噩梦——“牵一发而动全身”。随着人员迭代变化，模块的开发者也会变化，BFF 层就会慢慢变成一个无人知晓，无人敢动的模块。

当 BFF 层转换成 SFF（Serverless For Front-end）层之后，会有什么变化？SFF 的职责会变的单一、零运维、成本更低，这些是 Serverless 本身自带的能力，而这些能力可以帮助前端进一步释放生产潜能。开发者不再需要一个富 BFF 层，而只需一个接口或一个 SFF 就可以实现功能，天然解决了“牵一发而动全身”的问题。

如果接口停服或者没有流量，那么所用的实例会自动缩零，也就很容易分辨出是哪一个接口函数，后期就可以删掉此接口的函数，有效提升资源利用率。

高德在 Serverless 应用上非常先进，实现了 FaaS 层与研发体系的完全对接，因此，应用从开发、测试、灰度、上线的全生命周期，到具备流控、弹性、容灾等标准化能力，所用的时间较以前缩短了 40%，大大提高了人效。

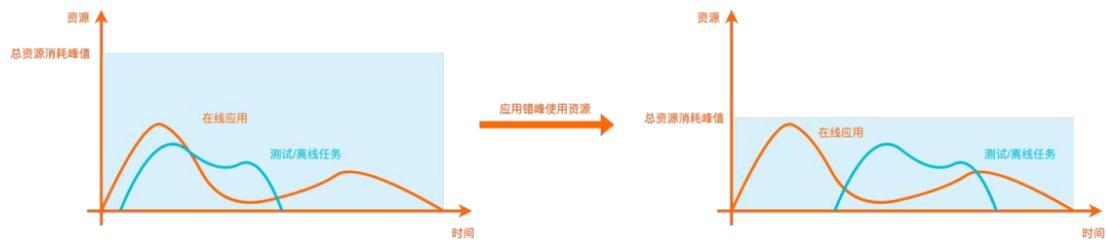


弹性以及成本

通过流量趋势数据，我们可以观察到地图场景流量特点——高峰与低峰的落差十分明显。按照传统应用的资源准备，我们需要根据最高峰的流量进行资源准备，所以到了流量低峰期，多准备的机器会有很多冗余，这就造成了成本的浪费。

针对以上情况，高德使用了阿里云函数计算，可以根据流量变化自动扩缩容。

然而，提升扩缩容速度的复杂性较大，一直是大企业的专属，但函数计算可以通过毫秒级别的启动优势，将快上快下的扩缩容能力普及给用户，轻松帮助用户实现了计算资源的弹性利用，并且大大降低了成本。



可观测性

可观测性是应用上线诊断平台的必备属性，要让用户观察到 RT 变化、资源的使用率、系统应用的全链路调用，从而快速诊断出系统应用的瓶颈问题。阿里云函数计算率先与日志服务、云监控、tracing 平台以及函数工作流编排做了完美的融合，用户只需要配置一次，就可以完完整整的享受到以上这些功能，大大降低了用户的学习成本，实现了对应用程序的快速诊断。

This screenshot shows the AliCloud Function Compute tracing interface. At the top, there are tabs: 概述, 代码执行, 触发器, 日志查询, 函数指标, 调用链查询, 异步配置. The '调用链查询' tab is selected. Below it, there's a search bar with 'fc-tracing' and a time range selector '最近30分钟'. The main area has two sections: '时间变化趋势' (Time Change Trend) and 'Span耗时分布' (Span Duration Distribution). The '时间变化趋势' section shows a line chart of duration vs. time. The 'Span耗时分布' section shows a histogram of duration ranges with green bars labeled 'Success'. At the bottom, there's a table of trace logs with columns: TraceId, 时间 (Time), Span名称 (Span Name), 耗时 (Duration), and IP. The table lists four entries: TracingService/tracing, InvokeFunction, RuntimeInitialization, and PrepareCode, all from 2020-11-18 10:33:23.

Serverless 规模化落地的序幕已经拉开，更多场景正在各行各业中解锁。Serverless 在高德的规模化落地，对于业务方而言，业务迭代更快更灵活了，为业务创新创造了前提条件；对于前端

开发者而言，进一步激活了开发者的生产潜能，提升了极大的能力自信。高德出行业务从 2020 年初的能力试点到“十一出行节”的自主出行核心场景，期间接入了阿里云函数计算，积累了非常宝贵的云原生落地经验，为未来业务整体上云打下了良好基础。

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

3 倍+提升，高德地图极致性能优化之路

作者：景尧

1. 导读

随着移动互联网的成熟发展，移动应用技术上呈现出多样化的趋势，业务上倾向打造平台及超级入口，超级应用应运而生。但业务快速扩张与有限的系统资源必然是冲突的，如何实现多（能力服务的高增长）、快（体验流畅）、好（兼容稳定）、省（资源成本低），让大象也能跳舞，成为摆在超级应用面前必须解决的问题。

伴随着高德地图 APP 近几年的高速发展，也面临到这些问题，从 2019 年开始，我们开启了一系列性能优化专项，对高德地图 APP 进行了深入性能分析和极致优化，取得比较显著的效果。

在这个过程中总结了一系列优化思路和技术方案，希望对同样面临超级应用性能问题的你有所帮助。

经过一系列优化动作，我们在保证业务需求正常迭代新增的基础上，启动、核心链路交互、行中内存、包大小等多方面均实现了性能的成倍提升，尤其是低端机上达到了 3 倍+的提升，从多个维度改善了用户性能体验。



- 启动攻坚**：启动耗时降低 70%+，实现 2s 地图元素完成展示，并管控保持在稳定低水位，呈下降趋势。
- 核心链路交互优化**：在搜索、路线等链路上实现中高端机型秒开、低端机型 2s 内打开，整体提升用户流畅交互体验。
- 行中内存优化**：全机型优化了 30% 左右，提高稳定性。

- **包大小攻坚**：双端体积降低 20%，提高安装转换率。

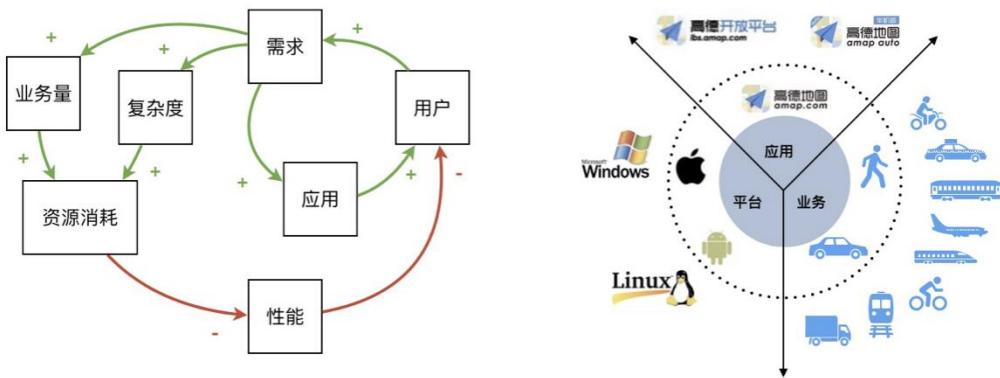
2. 性能优化业务背景

某段时间，高德地图 APP 面临着性能恶化、管控困难的问题。以启动耗时为例，双端启动等待体感明显，并且历史上治理后出现反复，整体呈上升趋势，我们思考问题背后的问题，主要有以下几个方面：

业务庞大

超级应用一般都经历这样的发展过程。首先，应用提供服务给用户，用户开始增长，增长的用户会产生更多的需求。应用为满足新增需求不断迭代，提供新的服务。新的服务推动用户进一步增长，进入下一个循环。正是在这个正循环发展中，应用像滚雪球一样越滚越大，终于成为超级应用。

然而，随着业务需求的不断增长，业务量和复杂度也随着上升，系统资源会越占越多。但机器资源是有限的，资源的争夺不可避免地会导致性能问题，从而影响用户体验和业务扩展，成为超级应用正循环发展的拦路虎。



高德地图也同样经历了这样的过程，随着这几年的快速发展，应用从手机扩展到了车机，平台从 iOS、Android 扩展了 Windows 和 Linux，覆盖 10 多种出行方式的同时，还在不断提供组队、视频、语音、AR 等新服务。与此相应的是单端代码行超百万行，线程上百，任务上千，造成了持续的性能压力。

环境复杂

性能问题面临的另一个主要挑战是超级应用的环境复杂。一方面，随着移动设备的长线发展，系统碎片化情况越来越严重，Android 系统横跨 11 个主版本，iOS 横跨 14 个主版本，加之设备厂商对系统进行各种各样的改造，进一步增加了系统的碎片化；另一方面，用户移动设备的环境是非常不稳定的，电量、温度的变化以及其他应用的抢占都会造成内存、CPU、GPU 等资

源波动。复杂不可控的环境为一致的性能体验的保持增加了很大的难度。

但作为大用户体量的超级应用，任何环境的体验都要保证。特别是地图领域，用户习惯对不同产品直接对比，任何环境下性能体验问题，都会直接影响产品的整体口碑，导致用户流失。所以需要兼顾所有环境，不只是主流机型系统和场景，在长尾场景与机型系统上也必须流畅运行，这就要求超级应用这头大象不但要在舞台上跳舞，在凳子上、甚至在水里也能跳舞。

技术链路长

为了满足研发效率提升、产品动态化等多样需求，移动应用技术上除支持原生开发外，也要支持小程序、Web H5、C 基础库等跨平台、容器化、动态化开发。从高德 APP 来看，最顶层业务除了 OC、Java 外，还支持 JS 开发。支撑层提供了 AJAX、小程序、原生、C 等多种容器框架，同时还涉及 JS、JNI 等桥接层。最下面则用 C++ 提供地图各个引擎能力，这里包括 OpenGL、定位传感器融合等多种底层能力。技术链路自上而下开始变得长且复杂，链路上任何一环都可能导致性能问题，原有的单技术语言的排查工具已经无

法定位明确性能卡点模块，为性能排查和管控带来挑战。

3. 解法：低成本优化迁移，长线管控

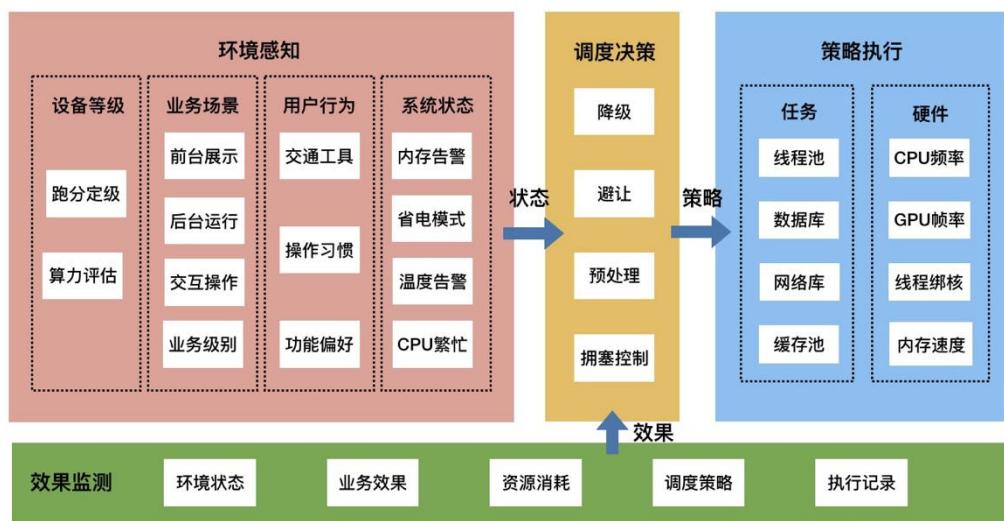
基于上面的问题，原有传统的一招鲜的优化方案，显然解决不了需求日益增长和复杂环境下的性能一致体验。所以，我们在专项实践过程中，沉淀了一套自适应资源调度框架，解决历史性能问题的同时，能够在不影响现有的研发效率的情况下，低成本优化迁移，实现新业务高性能的开发。此外，从系统底层进行全维度资源监控，自动定位分发问题，来实现长线管控，避免先治理后反弹的情况。

自适应资源调度框架

自适应资源调度框架在应用运行过程中，感知采集运行环境。然后对不同环境状态进行不同的调度决策，生成相应的性能优化策略，最终根据优化策略执行对应优化功能。与此同时，监测调度上下文以及调度策略执行效果，并将其反馈给调度决策系统，从而为进一步的决策调优提供信息输入。这样，可以做到在不同的运行环境下都能达到可预期的极致性能体验。并

且，整个过程，对业务无需额外开发，做到无感接入，避免影响业务开发效率。

自适应资源调度



•环境感知

感知环境分为硬件设备、业务场景、用户行为和系统状态四个维度：

- 。硬件设备上，一方面通过集团实验室对已知设备进行评测跑分确定高中低端机型，另一方面在用户设备上本地对硬件进行实时算力评估。

- 业务场景上，将业务分为前台展示、后台运行、交互操作等几类，一般情况下前台正在进行交互操作的业务场景优先级最高，后台数据预处理业务场景优先级最低。对于同类别业务场景，根据业务 UV、交易量、资源消耗等维度进行 PK，确定细分优先级。
- 用户行为上，结合服务用户画像和本地实时推算，确定用户功能偏好和操作习惯，为下一步针对用户的精准优化决策做准备。
- 系统状态上，一方面通过系统提供接口获取诸如内存警告、温度警告及省电模式等来获取系统极端状态，另一方面通过对内存、线程、CPU 和电量进行监控，来实时确定系统性能资源情况。

• 调度决策

感知到环境状态之后，调度系统将结合各种状态与调度规则，进行业务以及资源调配决策：

- 降级规则：在低端设备上或者系统资源紧张告警（如内存、温度告警）时，关闭高耗能功能或者低优先级功能。
- 避让规则：高优先级功能运行时，低优先级功能进行避让，如用户点击搜索框时到搜索结果完全展示到时间段内，后台低优任务进行暂停避让，保证用户交互体验。
- 预处理规则：依据用户操作及习惯进行预处理，如某用户通常在启动 3s 后，点击搜索，则在 3s 之前对该用户搜索结果进行预加载，从而在用户点击时呈现极致的交互体验效果。
- 拥塞控制规则：在设备资源紧张时，主动降低资源申请量，如 CPU 繁忙时，主动降低线程并发量；这样在高优任务到来时，避免出现资源紧缺申请不到资源性能体验问题。

• 策略执行

策略执行分为任务执行和硬件调优：其中任务执行，主要是通过内存缓存、数据库、线程池和网络库对相

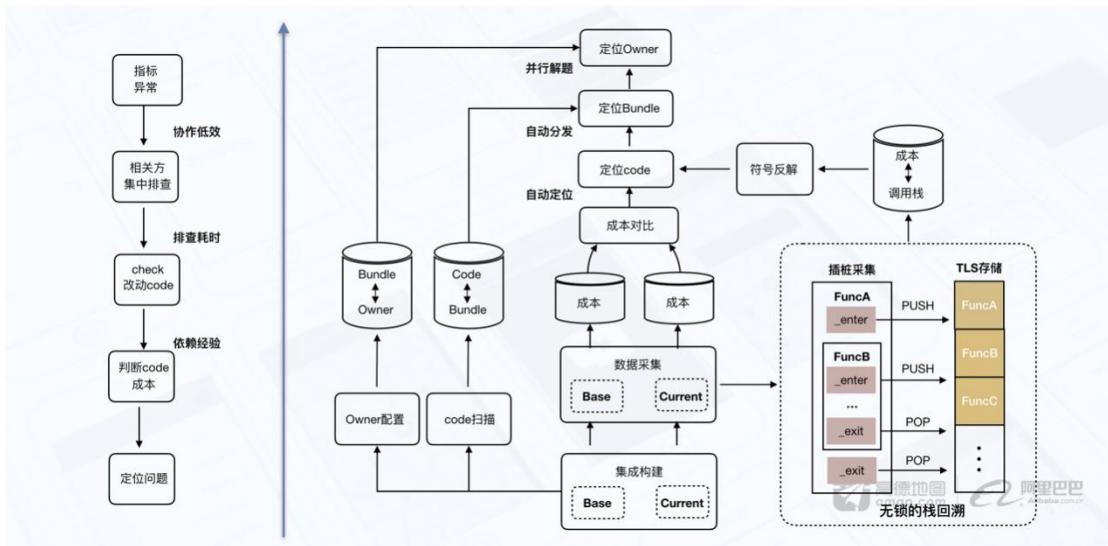
应任务的进行运行控制，来间接实现对各类资源的调度控制。而硬件调优，则是通过与系统厂商合作，直接对硬件资源进行控制，如 CPU 密集的高优业务开始运行时，将提高 CPU 频率，并将其运行线程绑定到大核上，避免线程来回切换损耗性能，最大化地调度系统资源来提升性能。

- 效果监测

在资源调度过程中对各个模块进行监测，并将环境状态、调度策略、执行记录、业务效果、资源消耗等情况反馈给调度系统，调度系统以此来评判本次调度策略的优劣，以做进一步的调优。

全维度资源监控

由于技术链路长、关联模块复杂，原来出现性能问题时，需要所有相关方集中排查，check 所有的改动代码，依赖个人经验判断代码的成本来定位问题，协作和排查成本都很高，导致性能管控有效落地阻力很大。所以我们就思考，性能问题的根本是硬件资源的竞争，那能不能逆向解题，反过来对资源成本进行监控，如果发现异常再回溯产生成本的代码，以及分发给相应 owner。

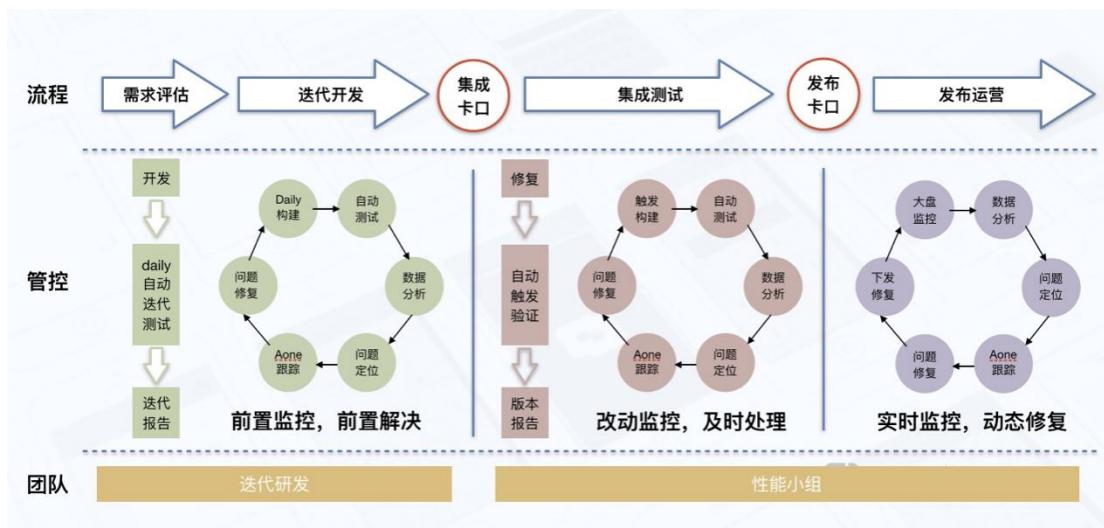


那基于这个思路，在构建的时候，首先通过代码扫描建立代码模块关联库。然后，进行成本和调用栈采集。采集完成后，对基线版本和当前版本的成本进行对比，如果发现异常，则通过符号反解异常成本的调用栈直接定位到问题代码。另外，基于问题代码查找代码模块关联库，来定位问题模块，最后将问题准确分发给模块相应的 owner，最终实现问题的自动定位和分发，支持团队并行解题。

管控流程体系

性能的有效长线管控，除了上面的资源监控平台，还需要配套的流程体系及组织保障。所以在 APP 的生命周期每个阶段都建立了从测试分析到修复验证的闭环管控。前置监控在迭代开发阶段，早发现早解决。在

集成阶段监控每一个改动，保证及时处理。线上通过实时监控和动态下发，实现快速修复。



4. 总结与思考

决心大于方案

超级应用的性能问题往往关联多方业务，需要多方团队协作，所以自上而下对性能的重视程度和优化决心是决定成败的关键，打通任督二脉，才能事半功倍，把优化方案顺利落地。

攻城难，守城更难

业务与技术都在快速迭代，要想保证优化成果防止反弹，管控是必须的，而管控就会有束缚和效率影响，管控过程中就难免会遇到各种各样的阻力。所以一方

面技术上，建立标准规则，配合提效工具和优化流程，尽量避免影响业务开发。另一方面，团队需要具有共同认知，性能体验与功能体验同等重要，用户对比心智很强，性能体验往往与产品口碑直接挂钩。

性能优化永远在路上

目前，我们很多优化策略以及数据参数还是从实验室调校而来。未来，我们会进一步探索云端一体、端智能等技术，做到更懂用户，贴合业务和用户特点，实现性能体验的个性化提升。

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德 APP 启动耗时剖析与优化实践（iOS 篇）

作者：戴铭

前言

最近高德地图 APP 完成了一次启动优化专项，超预期将双端启动的耗时都降低了 65%以上，iOS 在 iPhone7 上速度达到了 400 毫秒以内。就像产品们用后说的，快到不习惯。算一下每天为用户省下的时间，还是蛮有成就感的，本文做个小结。

如何对 iOS 启动耗时分析

多维度分析

主线程耗时

CPU

内存

I/O

网络

启动类型

冷启：内存无、无进程

热启：内存有部分、无进程

暂停：内存有、进程有





(文中配图均为多才多艺的技术哥哥手绘)

启动阶段性能多维度分析

要优化，首先要做到的是对启动阶段的各性能纬度做分析，包括主线程耗时、CPU、内存、I/O、网络。这样才能更加全面的掌握启动阶段的开销，找出不合理的方法调用。

启动越快，更多的方法调用就应该做成按需执行，将启动压力分摊，只留下那些启动后方法都会依赖的方法和库的初始化，比如网络库、Crash 库等。而剩下那些需要预加载的功能可以放到启动阶段后再执行。

启动有哪几种类型，有哪些阶段呢？

启动类型分为：

- **Cold**: APP 重启后启动，不在内存里也没有进程存在。
- **Warm**: APP 最近结束后再启动，有部分在内存但没有进程存在。
- **Resume**: APP 没结束，只是暂停，全在内存中，进程也存在。

分析阶段一般都是针对 Cold 类型进行分析，目的就是要让测试环境稳定。为了稳定测试环境，有时还需要找些稳定的机型，对于 iOS 来说 iPhone7 性能中等，稳定性也不错就很适合，Android 的 Vivo 系列也相对稳定，华为和小米系列数据波动就比较大。

除了机型外，控制测试机温度也很重要，一旦温度过高系统还会降频执行，影响测试数据。有时候还会设置飞行模式采用 Mock 网络请求的方式来减少不稳定的网络影响测

试数据。最好是重启后退 iCloud 账号，放置一段时间再测，更加准确些。

了解启动阶段的目的就是聚焦范围，从用户体验上来确定哪个阶段要快，以便能够让用户可视和响应用户操作的时间更快。

简单来说 iOS 启动分为加载 Mach-O 和运行时初始化过程，加载 Mach-O 会先判断加载的文件是不是 Mach-O，通过文件第一个字节，也叫魔数来判断，当是下面四种时可以判定是 Mach-O 文件：

- 0xfeedface 对应的 loader.h 里的宏是 MH_MAGIC
- 0xfeedfact 宏是 MH_MAGIC_64
- NXSwapInt(MH_MAGIC) 宏 MH_GIGAM
- NXSwapInt(MH_MAGIC_64) 宏 MH_GIGAM_64

Mach-O 主要分为：

- 中间对象文件 (MH_OBJECT)
- 可执行二进制 (MH_EXECUTE)
- VM 共享库文件 (MH_FVMLIB)
- Crash 产生的 Core 文件 (MH_CORE)

- preload (MH_PRELOAD)
- 动态共享库 (MH_DYLIB)
- 动态链接器 (MH_DYLINKER)
- 静态链接文件 (MH_DYLIB_STUB) 符号文件和调试信息 (MH_DSYM) 这几种。

确定是 Mach-O 后，内核会 fork 一个进程，execve 开始加载。检查 Mach-O Header。随后加载 dyld 和程序到 Load Command 地址空间。通过 dyld_stub_binder 开始执行 dyld，dyld 会进行 rebase、binding、lazy binding、导出符号，也可以通过 DYLD_INSERT_LIBRARIES 进行 hook。

dyld_stub_binder 给偏移量到 dyld 解释特殊字节码 Segment 中，也就是真实地址，把真实地址写入到 la_symbol_ptr 里，跳转时通过 stub 的 jump 指令跳转到真实地址。dyld 加载所有依赖库，将动态库导出的 trie 结构符号执行符号绑定，也就是 non lazybinding，绑定解析其他模块功能和数据引用过程，就是导入符号。

Trie 也叫数字树或前缀树，是一种搜索树。查找复杂度 $O(m)$ ， m 是字符串的长度。和散列表相比，散列最差复杂度是 $O(N)$ ，一般都是 $O(1)$ ，用 $O(m)$ 时间评估 hash。散列缺点是会分

配一大块内存，内容越多所占内存越大。Trie 不仅查找快，插入和删除都很快，适合存储预测性文本或自动完成词典。

为了进一步优化所占空间，可以将 Trie 这种树形的确定性有限自动机压缩成确定性非循环有限状态自动体（DAFSA），其空间小，做法是会压缩相同分支。

对于更大内容，还可以做更进一步的优化，比如使用字母缩减的实现技术，把原来的字符串重新解释为较长的字符串；使用单链式列表，节点设计为由符号、子节点、下一个节点来表示；将字母表数组存储为代表 ASCII 字母表的 256 位的位图。

尽管 Trie 对于性能会做很多优化，但是符号过多依然会增加性能消耗，对于动态库导出的符号不宜太多，尽量保持公共符号少，私有符号集丰富。

这样维护起来也方便，版本兼容性也好，还能优化动态加载程序到进程的时间。

然后执行 attribute 的 constructor 函数。举个例子：

```
1. #include <stdio.h>
2.
3. __attribute__((constructor))
4. static void prepare() {
5.     printf("%s\n", "prepare");
6. }
7.
8. __attribute__((destructor))
9. static void end() {
10.    printf("%s\n", "end");
11. }
12.
13. void showHeader() {
14.     printf("%s\n", "header");
15. }
```

运行结果：

```
1. ming@mingdeMacBook-Pro macho_demo % ./main "hi"
2. prepare
3. hi
4. end
```

运行时初始化过程分为：

- 加载类扩展。
- 加载 C++ 静态对象。
- 调用 +load 函数。
- 执行 main 函数。
- Application 初始化，到 applicationDidFinishLaunchingWithOptions 执行完。
- 初始化帧渲染，到 viewDidAppear 执行完，用户可见可操作。



也就是说对启动阶段的分析以 viewDidAppear 为截止。这次优化之前已经对 Application 初始化之前做过优化，效果并

不明显，没有本质的提高，所以这次主要针对 Application 初始化到 `viewDidAppear` 这个阶段各性能多纬度进行分析。

工具的选择其实目前看来是很多的，Apple 提供的 System Trace 会提供全面系统的行为，可以显示底层系统线程和内存调度情况，分析锁、线程、内存、系统调用等问题。总的来说，通过 System Trace 能清楚知道每时每刻 APP 对系统资源的使用情况。

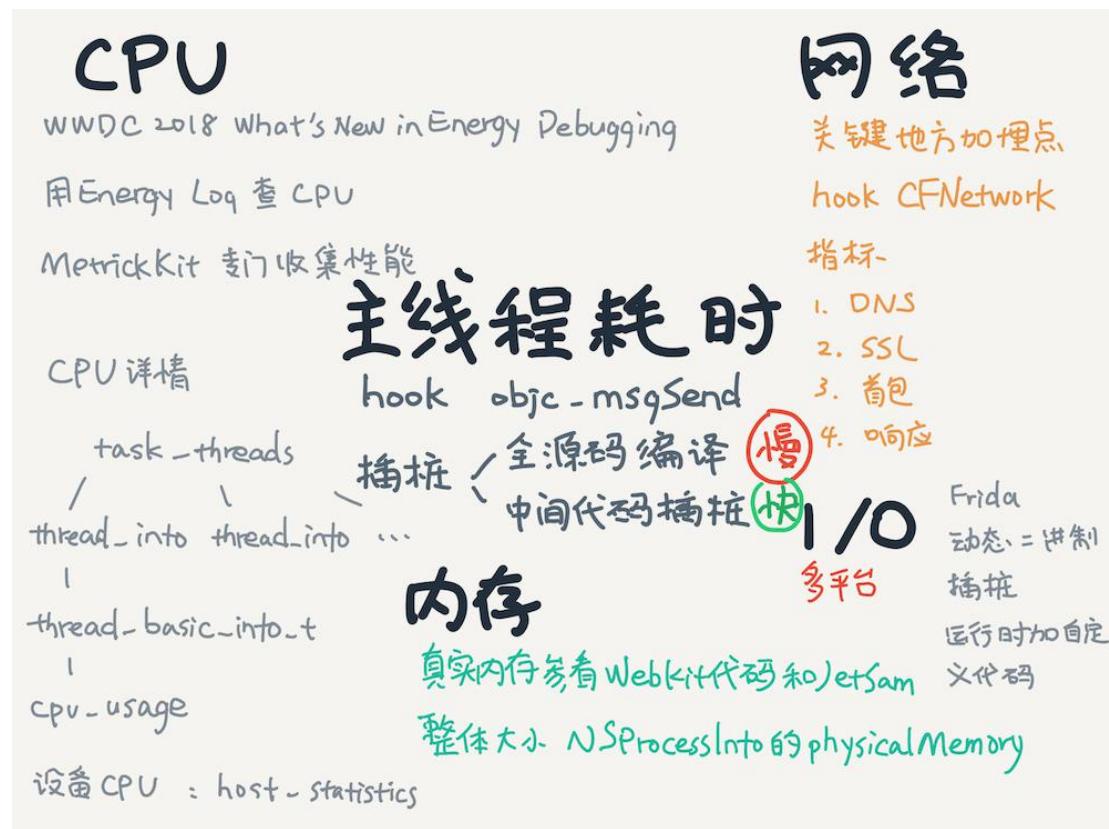
System Trace 能查看线程的状态，可以了解高优线程使用相对于 CPU 数量是否合理，可以看到线程在执行、挂起、上下文切换、被打断还是被抢占的情况。虚拟内存使用产生的耗时也能看到，比如分配物理内存，内存解压缩，无缓存时进行缓存的耗时等。甚至是发热情况也能看到。

System Trace 还提供手动打点进行信息显式，在你的代码中导入 `sys/kdebug_signpost.h` 后。

配对 `kdebug_signpost_start` 和 `kdebug_signpost_end` 就可以了。这两个方法有五个参数，第一个是 `id`，最后一个颜色，中间都是预留字段。

Xcode11 开始 XCTest 还提供了测量性能的 API。苹果在 [2019 年 WWDC 启动优化专题](#) 也介绍了 Instruments 里的最新模板 App launch 如何分析启动性能。但是要想达到对启动数据进行留存取均值、Diff、过滤、关联分析等自动化操作，App launch 目前还没法做到。

下面针对主线程耗时、CPU、网络、内存、I/O 等多维度进行分析：



- **主线程耗时**

多个纬度性能分析中最重要、最终用户体验感到的是主线程耗时分析。对主线程方法耗时可以直接使用 Massier，这是 everettjf 开发的[一个 Objective-C 方法跟踪工具](#)。

生成 trace json 进行分析，或者参看这个代码

[GCDFetchFeed/SMCallTraceCore.c at](#)

[master · ming1016/GCDFetchFeed · GitHub](#)

自己手动 hook objc_msgSend 生成一份 Objective-C 方法耗时数据进行分析。还有种插桩方式，可以解析 IR（加快编译速度），然后在每个方法前后插入耗时统计函数。

文章后面我会着重介绍如何开发工具进一步分析这份数据，以达到监控启动阶段方法耗时的目的。

hook 所有的方法调用，对详细分析时很有用，不过对于整个启动时间影响很大，要想获取启动每个阶段更准确的时间消耗还需要依赖手动埋点。

为了更好的分析启动耗时问题，手动埋点也会埋的越来越多，也会影响启动时间精确度，特别是当团队很多，模块很多时，问题会突出。但是每个团队在排查启动耗时往往只会关注自己或相关某几个模块的分析，基于此，可以把

不同模块埋点分组，灵活组合，这样就可以照顾到多种需求了。

- CPU

为什么分析启动慢除了分析主线程方法耗时外，还要分析其它纬度的性能呢？

我们先看看启动慢的表现，启动慢意味着界面响应慢、网络慢（数据量大、请求数多）、CPU 超负荷降频（并行任务多、运算多），可以看出影响启动的因素很多，还需要全面考虑。

对于 CPU 来说，WWDC 的

[What's New in Energy Debugging — WWDC 2018 — Videos](#)
[— Apple Developer](#)

介绍了用 Energy Log 来查 CPU 耗电，当前台三分钟或后台一分钟 CPU 线程连续占用 80%以上就判定为耗电，同时记录耗电线程堆栈供分析。还有一个 MetrickKit 专门用来收集电源和性能统计数据，每 24 小时就会对收集的数据进行汇总上报，Matty 在 NSHipster 网站上也[发了篇文章](#)专门进行

介绍：

那么，CPU 的详细使用情况如何获取呢？也就是说哪个方法用了多少 CPU。

有好几种获取详细 CPU 使用情况的方法。线程是计算机资源调度和分配的基本单位。CPU 使用情况会提现到线程这样的基本单位上。`task_struct` 的 `act_list` 数组包含所有线程，使用 `thread_info` 的接口可以返回线程的基本信息，这些信息定义在 `thread_basic_info_t` 结构体中。这个结构体内的信息包含了线程运行时间、运行状态以及调度优先级，其中也包含了 CPU 使用信息 `cpu_usage`。

获取方式参看：

[objective c — Get detailed iOS CPU usage with different states](#)

[— Stack Overflow](#)

[GT GitHub — Tencent/GT](#) 也有获取 CPU 的代码。

整体 CPU 占用率可以通过 `host_statistics` 函数取到 `host_cpu_load_info`，其中 `cpu_ticks` 数组是 CPU 运行的时钟脉冲数量。通过 `cpu_ticks` 数组里的状态，可以分别获取

CPU_STATE_USER、CPU_STATE_NICE、CPU_STATE_SYSTEM 这三个表示使用中的状态，除以整体 CPU 就可以取到 CPU 的占比。

通过 NSProcessInfo 的 activeProcessorCount 还可以得到 CPU 的核数。线上数据分析时会发现相同机型和系统的手机，性能表现却截然不同，这是由于手机过热或者电池损耗过大后系统降低了 CPU 频率所致。

所以，如果取得 CPU 频率后也可以针对那些降频的手机来进行针对性的优化，以保证流畅体验。获取方式可以[参考这里](#)。

• 内存

要想获取 APP 真实的内存使用情况可以[参看 WebKit 的源码](#)

JetSam 会判断 APP 使用内存情况，超出阈值就会杀死 APP，JetSam 获取阈值的[代码在这里](#)。整个设备物理内存大小可以通过 NSProcessInfo 的 physicalMemory 来获取。

• 网络

对于网络监控可以使用 Fishhook 这样的工具 Hook 网络底层库 CFNetwork。网络的情况比较复杂，所以需要定些和时间相关的关键的指标，指标如下：

- DNS 时间
- SSL 时间
- 首包时间
- 响应时间

有了这些指标才能够有助于更好的分析网络问题。启动阶段的网络请求是非常多的，所以 HTTP 的性能是非常要注意的。以下是 WWDC 网络相关的 Session：

[Your App and Next Generation Networks – WWDC 2015 – Videos](#)

— Apple Developer

[Networking with NSURLSession – WWDC 2015 – Videos](#)

— Apple Developer

[Networking for the Modern Internet – WWDC 2016 – Videos](#)

— Apple Developer

[Advances in Networking, Part 1 – WWDC 2017 – Videos](#)

— Apple Developer

[Advances in Networking, Part 2 – WWDC 2017 – Videos – Apple Developer](#)

[Optimizing Your App for Today's Internet – WWDC 2018 – Videos – Apple Developer](#)

- I/O

对于 I/O 可以使用

[Frida • A world-class dynamic instrumentation framework | Inject JavaScript to explore native apps on Windows, macOS, GNU/Linux, iOS, Android, and QNX](#)

这种动态二进制插桩技术，在程序运行时去插入自定义代码获取 I/O 的耗时和处理的数据大小等数据。Frida 还能够在其它平台使用。

关于多维度分析更多的资料可以看看历届 WWDC 的介绍。

下面我列下 16 年来 WWDC 关于启动优化的 Session，每场都很精彩。

[Using Time Profiler in Instruments — WWDC 2016 — Videos — Apple Developer](#)

[Optimizing I/O for Performance and Battery Life — WWDC 2016 — Videos — Apple Developer](#)

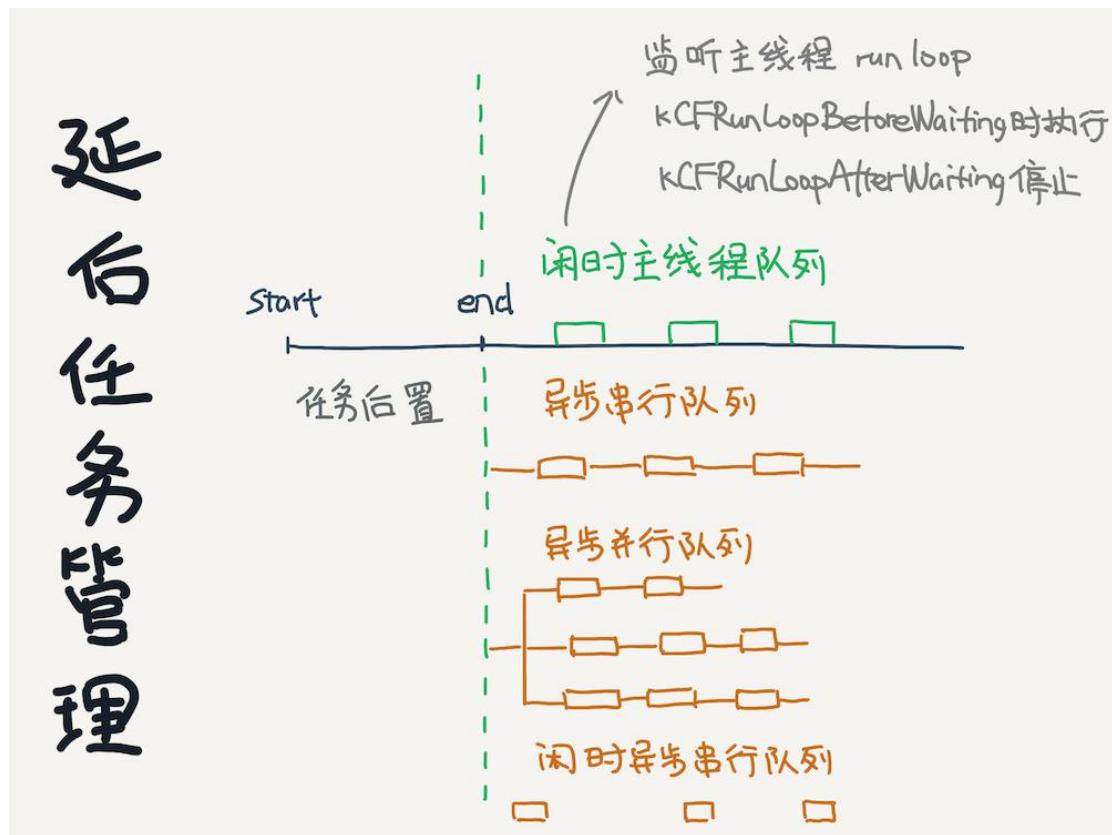
[Optimizing App Startup Time — WWDC 2016 — Videos — Apple Developer](#)

[App Startup Time: Past, Present, and Future — WWDC 2017 — Videos — Apple Developer](#)

[Practical Approaches to Great App Performance — WWDC 2018 — Videos — Apple Developer](#)

[Optimizing App Launch — WWDC 2019 — Videos — Apple Developer](#)

延后任务管理



经过前面所说的对主线程耗时方法和各个纬度性能分析后，对于那些分析出来没必要在启动阶段执行的方法，可以做成按需或延后执行。

任务延后的处理不能粗犷的一口气在启动完成后在主线程一起执行，那样用户仅仅只是看到了页面，依然没法响应操作。那该怎么做呢？套路一般是这样，创建四个队列，分别是：

- 异步串行队列
- 异步并行队列
- 闲时主线程串行队列

- 闲时异步串行队列

有依赖关系的任务可以放到异步串行队列中执行。异步并行队列可以分组执行，比如使用 `dispatch_group`，然后对每组任务数量进行限制，避免 CPU、线程和内存瞬时激增影响主线程用户操作，定义有限数量的串行队列，每个串行队列做特定的事情，这样也能够避免性能消耗短时间突然暴涨引起无法响应用户操作。

使用 `dispatch_semaphore_t` 在信号量阻塞主队列时容易出现优先级反转，需要减少使用，确保 QoS 传播。可以用 `dispatch_group` 替代，性能一样，功能不差。

异步编程可以直接 GCD 接口来写，也可以使用阿里的协程框架

[coobjc GitHub — alibaba/coobjc](#)

闲时队列实现方式是监听主线程 runloop 状态，在 `kCFRunLoopBeforeWaiting` 时开始执行闲时队列里的任务，在 `kCFRunLoopAfterWaiting` 时停止。

优化后如何保持？

攻易守难，就像刚到新团队时将包大小减少了 48 兆，但是一年多一直能够守住，除了决心还需要有手段。

对于启动优化来说，将各性能纬度通过监控的方式盯住是必要的，但是发现问题后快速、便捷的定位到问题还是需要找些突破口。

我的思路是将启动阶段方法耗时多的按照时间线一条一条排出来，每条包括方法名、方法层级、所属类、所属模块、维护人。考虑到便捷性，最好还能方便的查看方法代码内容。

接下来我通过开发一个工具，详细介绍下怎么实现这样的效果。

• 解析 json

如前面所说在输出一份 Chrome trace 规范的方法耗时 json 后，先要解析这份数据。这份 json 数据类似下面的样子：

```
1. {"name": "[SMVeilweaa]upVeilState:", "cat": "catname", "ph": "B", "pid": 2381, "tid": 0, "ts": 21},
```

```
2. {"name": "[SMVeilweaa]tatLaunchState:", "cat": "catname", "ph": "B", "pid": 238
     1, "tid": 0, "ts": 4557},
3. {"name": "[SMVeilweaa]tatTimeStamp:state:", "cat": "catname", "ph": "B", "pid": 2381,
     2381, "tid": 0, "ts": 4686},
4. {"name": "[SMVeilweaa]tatTimeStamp:state:", "cat": "catname", "ph": "E", "pid": 2381,
     2381, "tid": 0, "ts": 4727},
5. {"name": "[SMVeilweaa]tatLaunchState:", "cat": "catname", "ph": "E", "pid": 238
     1, "tid": 0, "ts": 5732},
6. {"name": "[SMVeilweaa]upVeilState:", "cat": "catname", "ph": "E", "pid": 2381,
     "tid": 0, "ts": 5815},
7. ...
```

通过 Chrome 的 Trace-Viewer 可以生成一个火焰图。其中 name 字段包含了类、方法和参数的信息，cat 字段可以加入其它性能数据，ph 为 B 表示方法开始，为 E 表示方法结束，ts 字段表示。

很多工程在启动阶段会执行大量方法，很多方法耗时很少，可以过滤那些小于 10 毫秒的方法，让分析更加聚焦。

iOS 15秒内T1到T5阶段大于10毫秒耗时方法						
阶段	耗时 (ms)	方法	外部耗时	Bundle	Owner	业务线
T0	14	① [SMAddaess]tatAptDaawaiveaQustSeavea	0	other	暂无	暂无
T0	14	② - [SMAddaess]tatSeaveaAddaessFoaKey:	0	other	暂无	暂无
T0	13	③ -- [SMAunAddaessSide]shaaedaunAddaessSide	0	other	暂无	暂无
T0	13	4 --- [SMAunAddaessSide]loudaunAddaess	0	other	暂无	暂无
T0	13	5 ---- [SMAunAddaessSide]loadOftheaauAddaess:	13	other	暂无	暂无
T0	15	① [SMSaltSlipContao]takeOveavContaoLifeCycle	0	other	暂无	暂无
T0	740	① [SMlication]tatApSMaunchea:	3	other	暂无	暂无
T0	46	② - [SMVeilweea]accoadName:jack:	3	other	暂无	暂无
T0	13	③ -- [SMLaageSideuaation]upHotelHostLatt	0	other	暂无	暂无
T0	13	4 --- [SMAun]addL.tattenea:comSMete:	3	other	暂无	暂无
T0	18	③ -- [SMLaageSideuaation]tutupMouseHosts	0	other	暂无	暂无
T0	18	4 --- [SMLaageSideuaation]LaageEnviaoment	0	other	暂无	暂无
T0	18	5 ---- [SMPeakMoonSideuaation]tataelceaseVasion	0	other	暂无	暂无
T0	16	6 ----- [SMJustSideMicao]lJustSide	11	other	暂无	暂无
T0	26	② - [SMVeilweea]accoadName:jack:	1	other	暂无	暂无
T0	21	③ -- [SMAun]addL.tattenea:comSMete:	0	other	暂无	暂无
T0	20	4 --- [SMDaySMSeavea]handleSM:value:	20	other	暂无	暂无
T0	567	② - [SMVeilweea]accoadName:jack:	0	other	暂无	暂无
T0	252	③ -- [SMDeskDayBoyaun]initDeskDay	39	other	暂无	暂无
T0	81	4 --- [SMASLLaageweaalImp]inteanStaanit	1	other	暂无	暂无
T0	79	5 ---- [SMLaageacability]staatweaaa	0	other	暂无	暂无
T0	79	6 ----- [SMLaageacability]staatl_aage	0	other	暂无	暂无
T0	79	7 ----- [SMLaageacability]staatacability	5	other	暂无	暂无
T0	48	8 ----- [SMacabilityGial]tomeachabilityStatus	44	other	暂无	暂无
T0	25	8 ----- [SMacabilityGial]tomstaatNotifiea	25	other	暂无	暂无
T0	122	4 --- [SMDeskDayBoyaun]cacateInitPaaam:	1	other	暂无	暂无
T0	22	5 ---- [SMDayFileGial]tatDayPotaivea	22	other	暂无	暂无
T0	86	5 ---- [KeychainItemWaappealInitStavDiuCaow:accessGaoup:	86	other	暂无	暂无
T0	314	③ -- [SMMyy]shaaedSaltv	0	other	暂无	暂无

耗时的高低也做了颜色的区分。外部耗时指的是子方法以外系统或没源码的三方方法的耗时，规则是父方法调用的耗时减去其子方法总耗时。

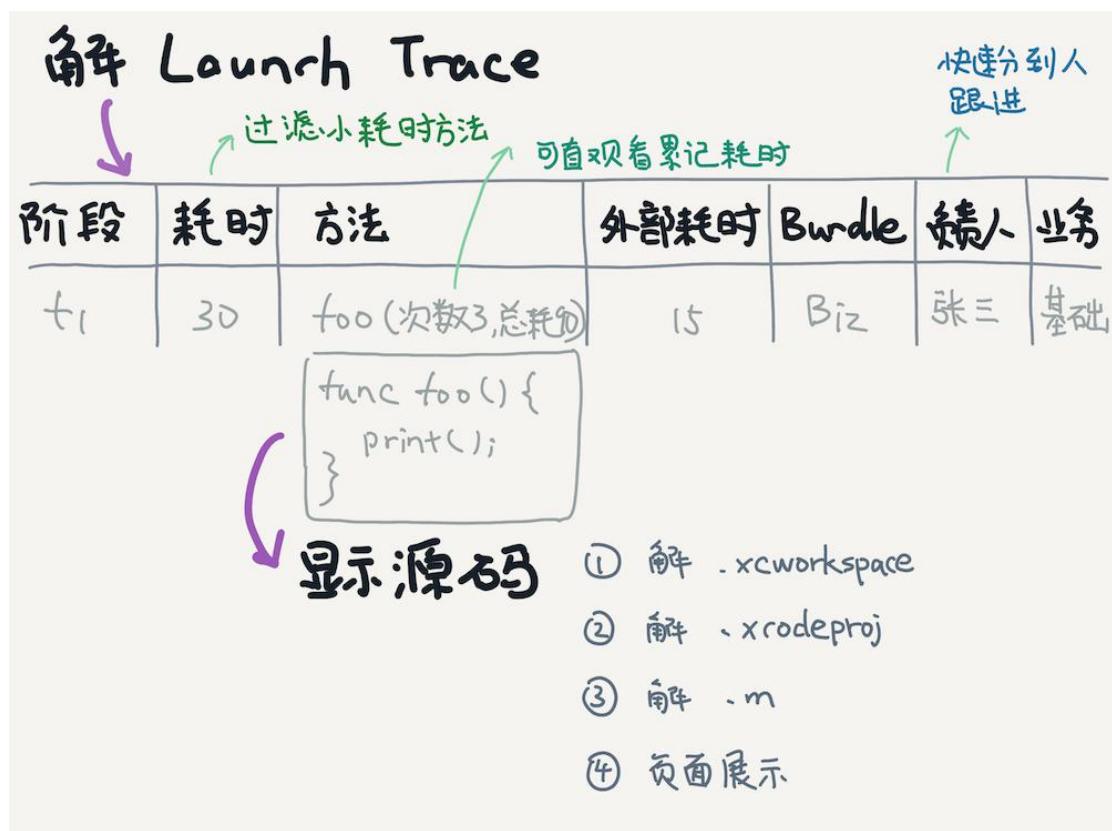
目前为止通过过滤耗时少的方法调用，可以更容易发现问题方法。但是，有些方法单次执行耗时不多，但是会执行很多次，累加耗时会大，这样的情况也需要体现在展示页面里。

另外，外部耗时高时或者碰到自己不了解的方法时，是需要到工程源码里去搜索对应的方法源码进行分析的，有的

方法名很通用时还需要花大量时间去过滤无用信息。

因此，接下来还需要做两件事情，首先累加方法调用次数和耗时，体现在展示页面中，另一个是从工程中获取方法源码能够在展示页面中进行点击显示。

完整思路如下图：



• 展示方法源码

在页面上展示源码需要先解析 .xcworkspace 文件，通过 .xcworkspace 文件取到工程里所有的 .xcodeproj 文件。分

析 .xcodeproj 文件取到所有 .m 和 .mm 源码文件路径，解析源码，取到方法的源码内容进行展示。

解析 .xcworkspace

开 .xcworkspace，可以看到这个包内主要文件是 contents.xcworkspacedata。内容是一个 xml：

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <Workspace
3.   version = "1.0">
4.   <FileRef
5.     location = "group:GCDFetchFeed.xcodeproj">
6.   </FileRef>
7.   <FileRef
8.     location = "group:Pods/Pods.xcodeproj">
9.   </FileRef>
10.  </Workspace>
```

.xcworkspace

content.xcworkspacedata

XMLTagNode

```

type : XMLTagNodeType
value : String
name : String
attributes : [XMLTagAttribute]

```

XMLTagNodeType

```

xml
single // <a click="go();"/> 需考虑<! [CDATA[
start // <p>
value // <p> value </p> 这种 cdata 标签
end   // </p>

```

解析 .xcodeproj

.xcodeproj

目标：取源码文件路径

Section

PBXBuildFile	: 文件
PBXFileReference	: 源码、资源、库
PBXGroup	: 文件夹与文件关系
PBXSourceBuildPhase	: 编译文件

构造 Section

PBXProject : Project 设置信息

XCBuildConfiguration : XCode Build Setting 对应

问题1：
老工程目录下有较多不在 Proj 里的文件

问题2：
.xcodeproj 类似 JSON
却不全相同

通过 XML 的解析可以获取 FileRef 节点内容，xcodeproj 的文件路径就在 FileRef 节点的 location 属性里。

每个 xcodeproj 文件里会有 project 工程的源码文件。为了能够获取方法的源码进行展示，那么就先要取出所有 project 工程里包含的源文件的路径。

xcodeproj 的文件内容看起来大概是下面的样子。

```
// !$*UTF8*$!
{
    archiveVersion = 1;
    classes = {
    };
    objectVersion = 50;
    objects = {

/* Begin PBXBuildFile section */
3AF0A7C6231E61B30080E07C /* AppDelegate.m in Sources */ = {isa =
    PBXBuildFile; fileRef = 3AF0A7C5231E61B30080E07C /* AppDelegate.m */; };
3AF0A7C9231E61B30080E07C /* ViewController.m in Sources */ = {isa =
    PBXBuildFile; fileRef = 3AF0A7C8231E61B30080E07C /* ViewController.m */;
};
3AF0A7CC231E61B30080E07C /* Main.storyboard in Resources */ = {isa =
    PBXBuildFile; fileRef = 3AF0A7CA231E61B30080E07C /* Main.storyboard */; };
3AF0A7CE231E61B50080E07C /* Assets.xcassets in Resources */ = {isa =
    PBXBuildFile; fileRef = 3AF0A7CD231E61B50080E07C /* Assets.xcassets */; };
3AF0A7D1231E61B50080E07C /* LaunchScreen.storyboard in Resources */ = {isa =
    PBXBuildFile; fileRef = 3AF0A7CF231E61B50080E07C /*
    LaunchScreen.storyboard */; };
3AF0A7D4231E61B50080E07C /* main.m in Sources */ = {isa = PBXBuildFile;
    fileRef = 3AF0A7D3231E61B50080E07C /* main.m */; };
3AF0A7DE231E61B50080E07C /* SampleProjectTests.m in Sources */ = {isa =
    PBXBuildFile; fileRef = 3AF0A7DD231E61B50080E07C /* SampleProjectTests.m
*/; };
}
```

其实内容还有很多，需要一个个解析出来。

考虑到 xcodeproj 里的注释很多，也都很有用，因此会多设计些结构来保存值和注释。思路是根据 XcodeprojNode 的类型来判断下一级是 key value 结构还是 array 结构。

如果 XcodeprojNode 的类型是 dicStart 表示下级是 key value 结构。如果类型是 arrStart 就是 array 结构。当碰到类型是 dicEnd，同时和最初 dicStart 是同级时，递归下一级树结构。而 arrEnd 不用递归，xcodeproj 里的 array 只有值类型的数据。

有了基本节点树结构以后就可以设计 xcodeproj 里各个 section 的结构。主要有以下的 section：

- PBXBuildFile：文件，最终会关联到 PBXFileReference。
- PBXContainerItemProxy：部署的元素。
- PBXFileReference：各类文件，有源码、资源、库等文件。
- PBXFrameworksBuildPhase：用于 framework 的构建。
- PBXGroup：文件夹，可嵌套，里面包含了文件与文件夹的关系。
- PBXNativeTarget：Target 的设置。
- PBXProject：Project 的设置，有编译工程所需信息。
- PBXResourcesBuildPhase：编译资源文件，有 xib、storyboard、plist 以及图片等资源文件。

- PBXSourcesBuildPhase：编译源文件（.m）。
- PBXTargetDependency：Taget 的依赖。
- PBXVariantGroup：. storyboard 文件。
- XCBuildConfiguration：Xcode 编译配置，对应 Xcode 的 Build Setting 面板内容。
- XCConfigurationList：构建配置相关，包含项目文件和 target 文件。

得到 section 结构 Xcodeproj 后，就可以开始分析所有源文件的路径了。根据前面列出的 section 的说明，PBXGroup 包含了所有文件夹和文件的关系，Xcodeproj 的 pbxGroup 字段的 key 是文件夹，值是文件集合，因此可以设计一个结构体 XcodeprojSourceNode 用来存储文件夹和文件关系。

接下来需要取得完整的文件路径。通过 recursiveFatherPaths 函数获取文件夹路径。这里需要注意的是需要处理 ... / 这种文件夹路径符。

解析 .m .mm 文件



对 Objective-C 解析可以参考 LLVM，这里只需要找到每个方法对应的源码，所以自己也可以实现。分词前先看看 LLVM 是怎么定义 token 的。[定义文件在这里](#)。

根据这个定义我设计了 token 的结构体，主体部分如下：

```

1. // 切割符号 []().&=+-<>~!/%^?;,:#@
```

```

2. public enum OCTK {
3.     case unknown // 不是 token
4.     case eof // 文件结束
5.     case eod // 行结束
6.     case codeCompletion // Code completion marker
7.     case cxxDefaultargEnd // C++ default argument end marker
}

```

```
8.    case comment // 注释  
9.    case identifier // 比如 abcde123  
10.   case numericConstant(OCTkNumericConstant) // 整型、浮点 0x123,  
解释计算时用，分析代码时可不用  
11.  case charConstant // 'a'  
12.  case stringLiteral // "foo"  
13.  case wideStringLiteral // L"foo"  
14.  case angleStringLiteral // <foo> 待处理需要考虑作为小于符号的  
问题  
15.  
16. // 标准定义部分  
17. // 标点符号  
18. case punctuators(OCTkPunctuators)  
19.  
20. // 关键字  
21. case keyword(OCTKKeyword)  
22.  
23. // @关键字  
24. case atKeyword(OCTKAtKeyword)  
25. }
```

完整的定义在这里：

[MethodTraceAnalyze/ParseOCTokensDefine.swift](#)

分词过程可以参看 LLVM 的实现：

[clang: lib/Lex/Lexer.cpp Source File](#)

我在处理分词时主要是按照分隔符一一对应处理，针对代码注释和字符串进行了特殊处理，一个注释一个 token，一个完整字符串一个 token。我的分词实现代码：

[MethodTraceAnalyze/ParseOCTokens.swift](#)

由于只要取到类名和方法里的源码，所以语法分析时，只需要对类定义和方法定义做解析就可以，语法树中节点设计：

```
1. // OC 语法树节点  
2. public struct OCNode {  
3.     public var type: OCNodeType  
4.     public var subNodes: [OCNode]  
5.     public var identifier: String // 标识  
6.     public var lineRange: (Int, Int) // 行范围  
7.     public var source: String // 对应代码  
8. }  
9. // 节点类型  
10. public enum OCNodeType {
```

```
11.     case `default`  
12.     case root  
13.     case `import`  
14.     case `class`  
15.     case method  
16. }
```

其中 lineRange 记录了方法所在文件的行范围，这样就能够从文件中取出代码，并记录在 source 字段中。

解析语法树需要先定义好解析过程的不同状态：

```
1. private enum RState {  
2.     case normal  
3.     case eod           // 换行  
4.     case methodStart    // 方法开始  
5.     case methodReturnEnd // 方法返回类型结束  
6.     case methodNameEnd  // 方法名结束  
7.     case methodParamStart // 方法参数开始  
8.     case methodContentStart // 方法内容开始  
9.     case methodParamTypeStart // 方法参数类型开始  
10.    case methodParamTypeEnd // 方法参数类型结束  
11.    case methodParamEnd    // 方法参数结束  
12.    case methodParamNameEnd // 方法参数名结束
```

```
13.  
14.    case at           // @  
15.    case atImplementation // @implementation  
16.  
17.    case normalBlock    // oc 方法外部的 block {}, 用  
        于 c 方法  
18. }
```

完整解析出方法所属类、方法行范围的代码在这里：

[MethodTraceAnalyze/ParseOCNodes.swift](#)

解析.m 和.mm 文件，一个一个串行解的话，对于大工程，每次解的速度很难接受，所以采用并行方式去读取解析多个文件。

经过测试，发现每组在 60 个以上时能够最大利用我机器（2.5 GHz 双核 Intel Core i7）的 CPU，内存占用只有 60M，一万多.m 文件的工程大概 2 分半能解完。

使用的是 dispatch group 的 wait，保证并行的一组完成再进入下一组。



现在有了每个方法对应的源码，接下来就可以和前面 trace 的方法对应上。页面展示只需要写段 js 就能够控制点击时展示对应方法的源码。

页面展示

在进行 HTML 页面展示前，需要将代码里的换行和空格替换成 HTML 里的对应的和 。

```

1. let allNodes = ParseOC.ocNodes(workspacePath: "/Users/ming/Downloads
/GCDFetchFeed/GCDFetchFeed/GCDFetchFeed.xcworkspace")

```

```

2. var sourceDic = [String:String]()

```

```
3. for aNode in allNodes {  
4.     sourceDic[aNode.identifier] = aNode.source.replacingOccurrences(of: "  
\n", with: "</br>").replacingOccurrences(of: " ", with: "&nbsp;")  
5. }
```

用 p 标签作为源码展示的标签，方法执行顺序的编号加方法名作为 p 标签的 id，然后用 display: none; 将 p 标签隐藏。方法名用 a 标签，click 属性执行一段 js 代码，当 a 标签点击时能够显示方法对应的代码。这段 js 代码如下：

```
1. function sourceShowHidden(sourceIdName) {  
2.     var sourceCode = document.getElementById(sourceIdName);  
3.     sourceCode.style.display = "block";  
4. }
```

最终效果如下图：

iOS 15秒内T1到T5阶段大于1毫秒耗时方法						
阶段	耗时 (ms)	方法	外部耗时	Bundle	Owner	业务线
T0	14	① [SMAddaess]tatAptDaawaiveaQustSeavea (总次数4、总耗时15)	0	other	暂无	暂无
T0	14	② -[SMAddaess]tatSeaveaAddaessFoaKey: (总次数30、总耗时24)	0	other	暂无	暂无
T0	13	③ --[SMaunAddaessSide]shaaeauAddaessSide (总次数30、总耗时14)	0	other	暂无	暂无
T0	13	4 ----[SMaunAddaessSide]loaddaunAddaess	0	other	暂无	暂无
T0	13	5 -----[SMaunAddaessSide]loadOtheaaunAddaess:	13	other	暂无	暂无
T0	15	① [SMSaltSlipContao]takeOveavContaoLifeCycle	0	other	暂无	暂无
		- (void)takeOveavContaoLifeCycle { RACScheduler *scheduler = [RACScheduler schedulerWithPriority:RACSchedulerPriorityDefault]; [[[SMDB shareInstance] markFeedAllItemsAsRead:self.feedModel.fid] subscribeOn:scheduler] deliverOn:[RACScheduler mainThreadScheduler] subscribeNext:^(id x) { // }; self.feedModel.unReadCount = 0; [self.navigationController popViewControllerAnimated:YES]; }				
T0	740	① [SMIcation]tatApSMaunchea:	3	other	暂无	暂无
T0	46	② -[SMVeilweea]aecoadName:jack: (总次数13、总耗时1907)	3	other	暂无	暂无
T0	13	③ --[SMLageSideuation]upHotelHostLtat	0	other	暂无	暂无
T0	13	4 ----[SMaun]addLattenea:comSMete: (总次数10、总耗时43)	3	other	暂无	暂无
T0	18	③ --[SMLageSideuation]tatupMouseHosts	0	other	暂无	暂无
T0	18	4 ----[SMLageSideuation]LageEnviaoment (总次数3、总耗时19)	0	other	暂无	暂无
T0	18	5 -----[SMPeakMoonSideuation]tataelaseVession (总次数3、总耗时18)	0	other	暂无	暂无
T0	16	6 -----[SMJustSideMica]CoJustSide (总次数81、总耗时18)	11	other	暂无	暂无
T0	26	② -[SMVeilweea]aecoadName:jack: (总次数13、总耗时1907)	1	other	暂无	暂无
T0	21	③ --[SMaun]addLattenea:comSMete: (总次数10、总耗时43)	0	other	暂无	暂无

将动态分析和静态分析进行了结合，后面可以通过不同版本进行对比，发现哪些方法的代码实现改变了，能展示在页面上。还可以进一步静态分析出哪些方法会调用到 I/O 函数、起新线程、新队列等，然后展示到页面上，方便分析。

读到最后，可以看到这个方法分析工具并没有用任何一个轮子，其实有些是可以使用现有轮子的，比如 json、xml、xcodeproj、Objective-C 语法分析等，之所以没有用是因为不同轮子使用的语言和技术区别较大，当格式更新时如果使用的单个轮子没有更新会影响整个工具。

开发这个工具主要工作是在解析上，所以使用自有解析技术也能够让所做的功能更聚焦，不做没用的功能，减少代码维护量，所要解析格式更新后，也能够自主去更新解析方式。更重要的一点是可以亲手接触下这些格式的语法设计。

结语

本文小结了启动优化的技术手段，总的来说，对启动进行优化的决心的重要程度是远大于技术手段的，决定着是否能够优化的更多。技术手段有很多，我觉得手段的好坏区别只是在效率上，最差的情况全用手动一个个去查耗时也是能够解题的。

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

iOS 代码染色原理及技术实践

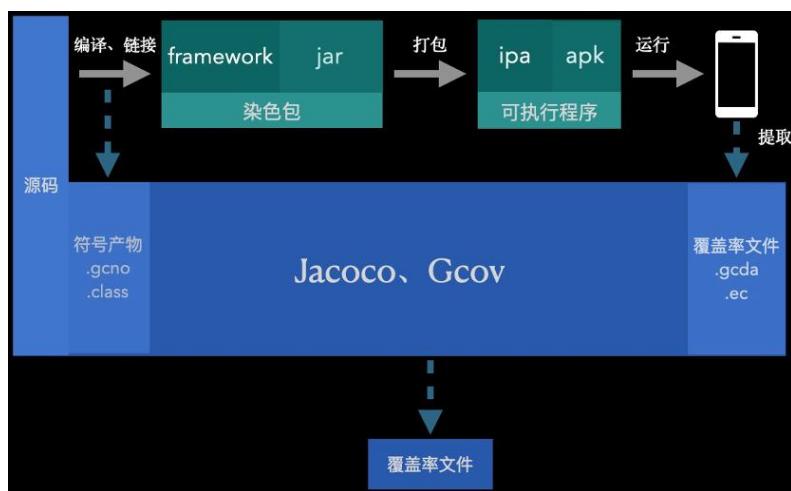
作者：罗诺

背景

随着业务的迅速发展，业务代码逻辑的复杂度增加。QA 测试的质量对于产品上线后的稳定性更加重要。一般 QA 测试的工作流程分为两大项：自动化测试和人工测试。这两种测试后都需要得到代码覆盖率。自动化测试的覆盖率，在双端都有比较成熟的方案。

本文着重介绍人工测试过程中，怎么得到对应的代码覆盖率。涉及到的技术主要是代码染色。以下会先介绍整体的工作流程，再对涉及到的技术一一阐述。

染色流程



流程图中涉及到了双端的关键节点以及技术点。我们重点介绍编译阶段。

- 编译阶段：生成染色包（对 IR 文件插桩）

需要在编译中增加编译选项，编译后会为每个可执行文件生成对应的 .gcno 文件。

- 运行阶段：生成二进制覆盖率文件。

在测试代码中调用覆盖率分发函数，会生成对应的 .gcda 文件。

- 解析阶段：将二进制覆盖率文件可视化。

编译阶段

在上文可以看出，编译阶段最核心的操作是对 IR 文件进行插桩。

什么是 IR 文件？插桩逻辑是什么？我们往下看。

语言处理系统

一个完整的语言处理系统中，从源程序到可执行的机器代码，如下图所示，历经几个重要模块。而我们上文提到的 IR 文件，是编译器模块中的产物，插桩处理也是在这个模块中进行。这里重点讨论下编译器。



编译器

说起编译器，我们了解到的传统编译器架构分为前端、优化器和后端。

传统编译器的劣势是：前端和后端没有完全分离，耦合在了一起，因而如果要支持一门新的语言或硬件平台，需要做大量的工作。一种更加灵活，适应性更好的编译器套件应运而生——LLVM.

LLVM

官网：<http://www.aosabook.org/en/llvm.html>

LLVM 是一个开源的，模块化和可重用的编译器和工具链技术的集合，或者说是一个编译器套件。

可以使用 LLVM 来编译 Kotlin , Ruby , Python , Haskell , Java , D , PHP , Pure , Lua 和许多其他语言。

LLVM 核心库还提供一个优化器，对流行的 CPU 做代码生成支持。

LLVM 同时支持 AOT 预先编译和 JIT 即时编译。

2012 年，LLVM 获得美国计算机协会 ACM 的软件系统大奖，和 UNIX , WWW , TCP/IP , Tex , JAVA 等齐名。

LLVM 和传统编译器最大的不同点在于，前端输入的任何语言，在经过编译器前端处理后，生成的中间码都是 IR 格式的。接下来看下 LLVM 架构下的巨大优势，iOS&MacOS 平台的编译器。

LLVM 编译器

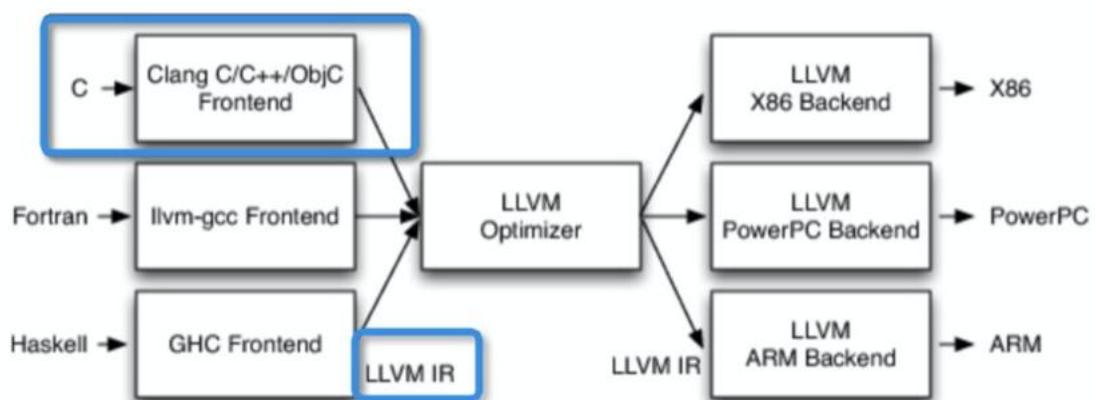


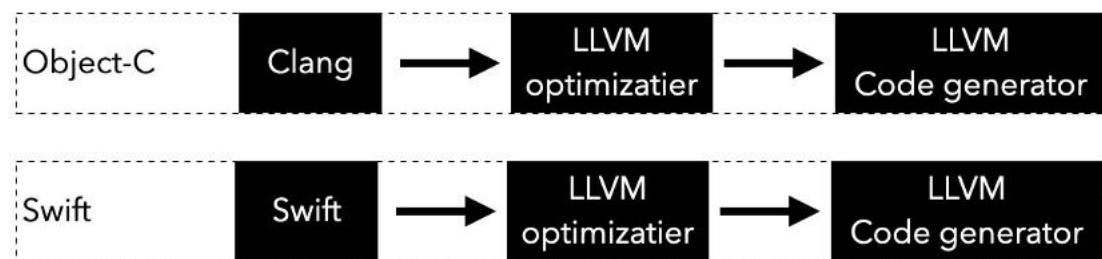
Figure 11.3: LLVM's Implementation of the Three-Phase Design

iOS&MacOS 平台编译器

iOS、MacOS 平台开发用的 IDE：Xcode。在 Xcode 5 版本前使用的是 GCC 编译器，在 Xcode 5 中将 GCC 彻底抛弃，替换为 LLVM。LLVM 包含了编译器前端、优化器和编译器后端三大模块。

其中 Swift 除了在编译器前端和 Objective-C 稍有不同，其他模块都是相同的。

如下图所示，能看出 LLVM 的优势，对于一门新的编程语言，只需要提供对应的编译前端，生成 IR。就可以完成整个新语言的处理。



聊过了 IR 文件在整个语言处理过程中的位置，下面我们看下 IR 文件生成逻辑以及插桩相关的逻辑。这不得不提到 Clang。

Clang 是 LLVM 的子项目，是 C、C++ 和 Objective-C 的编译器。Clang 在整个 Objective-C 编译过程中扮演了编译器前端的角色，同时也参与到了 Swift 编译过程中的 Objective-C API 映射阶段。

Clang 的特点是编译速度快，模块化，代码简单易懂，诊断信息可读性强，占用内存小以及容易扩展和重用等。

Clang 的主要功能是输出代码对应的抽象语法树（AST），针对用户发生的编译错误准确地给出建议，并将代码编译成 LLVM IR。

以 Xcode 为例，Clang 编译 Objective-C 代码的速度是 Xcode 5 版本前使用的 GCC 的 3 倍，其生成的 AST 所耗用掉的内存仅仅是 GCC 的五分之一左右。

关于 iOS 项目可以使用对应的命令获取，本文不作详细介绍。

关于编译器前端的主要工作项，感兴趣的读者阅读《编译原理》——龙书。

介绍完了 IR 的“生成器”。接下来我们详细介绍 IR 文件。

LLVM IR

LLVM Intermediate Representation。 LLVM 的中间代码，是编译器前端的输出，和编译器后端的输入。是连接编译器前端与 LLVM 后端的一个桥梁。

通常常见的文件格式为 `ll` 和 `bt`。做过 iOS 开发的读者应该了解 `bitcode`。`bt` 就是编译器开启 `bitcode` 后的一种中间代码格式。

IR 提供了独立于任何特定机器架构的源语，因此它是 LLVM 优化和进行代码生成的关键，也是 LLVM 有别于其他编译器的最大特点。 LLVM 的核心功能都是围绕 IR 建立的。

通常中间代码的表示形式分为：语法树（syntax tree）、三地址指令序列。为了更好的了解 IR 文件。这里介绍下三地址指令。

三地址指令

也可以称为三地址代码。之所以被称为三地址指令，是源于它的指令形式： $x = y \text{ op } z$ ，其中 op 是一个二目运算

符， y 和 z 是运算分量的地址， x 是运算结果的存放地址。
三地址指令最多只执行一个运算，通常是计算，比较或者
分支跳转运算。

三地址代码拆分了多运算符算术表达式以及控制流语句的
嵌套结构，所以适用于目标代码的生成和优化。

```
1. //像  $x+y*z$  这样的源代码被翻译成三地址指令序列:  
2. t1=y*z  
3. t2=x+t1  
4.  
5. //源码: do i = i + 1; while(a[i] < 10); 被翻译成如下的三地址指令  
6. i = i + 1  
7. t1 = a[i]  
8. if t1 < 10 goto 6  
9. 其中 t1, t2 是编译器产生的临时名字。
```

但是程序运行过程中，每个模块并不是完全独立的。存在着模块间的跳转。这些被翻译出的三地址指令，又被组合成另一种便于理解的形式——BB 块。

基本块

基本块(Basic Block)是满足下列条件的最大的连续三地址指令序列：

- 控制流只能从基本块中的第一个指令进入该块。
- 除了基本块的最后一个指令，控制流在离开基本块之前不会停机或者跳转。
- 只要基本块中的第一个指令被执行，那么基本块中的所有指令都会得到执行

其中中间代码指令序列生成 BB 块的算法如下：

- 确定中间代码序列中哪些指令是首指令
 - 中间代码的第一个三地址指令是一个首指令。
 - 任意一个条件或无条件转移指令之后的目标指令是一个首指令。
 - 紧跟在一个条件或无条件转移指令之后的指令是一个首指令。

- 每个首指令对应的基本块包括了从它自己开始，直到下一个首指令（不含）或者中间代码的结尾指令之间的所有指令。

举例：

```
1. i = 1 //第一个三地址指令，所以作为首指令  
2. j = 1 //第 11 行，跳转语句的目标指令。所以作为首指令  
3. t1 = 10*i  
4. t2 = t1+j  
5. t3 = 8*t2  
6. t4 = t3-88  
7. a[t4] = 0.0  
8. j = j+1  
9. if j<=10 goto (3) //本身作为跳转指令，所以是首指令  
10. i = i+1  
11. if i<=10 goto (2) //本身作为跳转指令，所以是首指令  
12. i = 1  
13. t5 = i - 1 //第 17 行，跳转语句的目标指令。所以是首指令  
14. t6 = 88*t5  
15. a[t6] = 1.0  
16. i = i+1  
17. if i<=10 goto (13)//本身作为跳转指令，所以是首指令
```

```
18.  
19. //把一个 10x10 的矩阵设置成单位矩阵中的中间代码  
20. for(i=1; i<=10; i++){  
21.     for(j=1; j<=10; j++){  
22.         a[i, j] = 0.0;  
23.     }  
24. }  
25. for(i=1; i<=10; i++){  
26.     a[i, j] = 1.0;  
27. }
```

对应被划分的 BB 块：

```
1 i = 1 BB1  
2 j = 1 BB2  
3 t1 = 10*i  
4 t2 = t1+j  
5 t3 = 8*t2  
6 t4 = t3-88  
7 a[t4] = 0.0  
8 j = j+1  
9 if j<=10 goto (3)  
10 i = i+1 BB4  
11 if i<=10 goto (2)  
12 i = 1 BB5  
13 t5 = i - 1  
14 t6 = 88*t5  
15 a[t6] = 1.0 BB6  
16 i = i+1  
17 if i<=10 goto (13)
```

在了解了 BB 块之后。我们距离怎么对 IR 文件进行插桩的真相已经越来越近了，下面我们来看下最后一个最重要的

环节。

流图

当将一个中间代码程序划分成为基本块之后，我们用一个流图来表示它们之间的控制流。

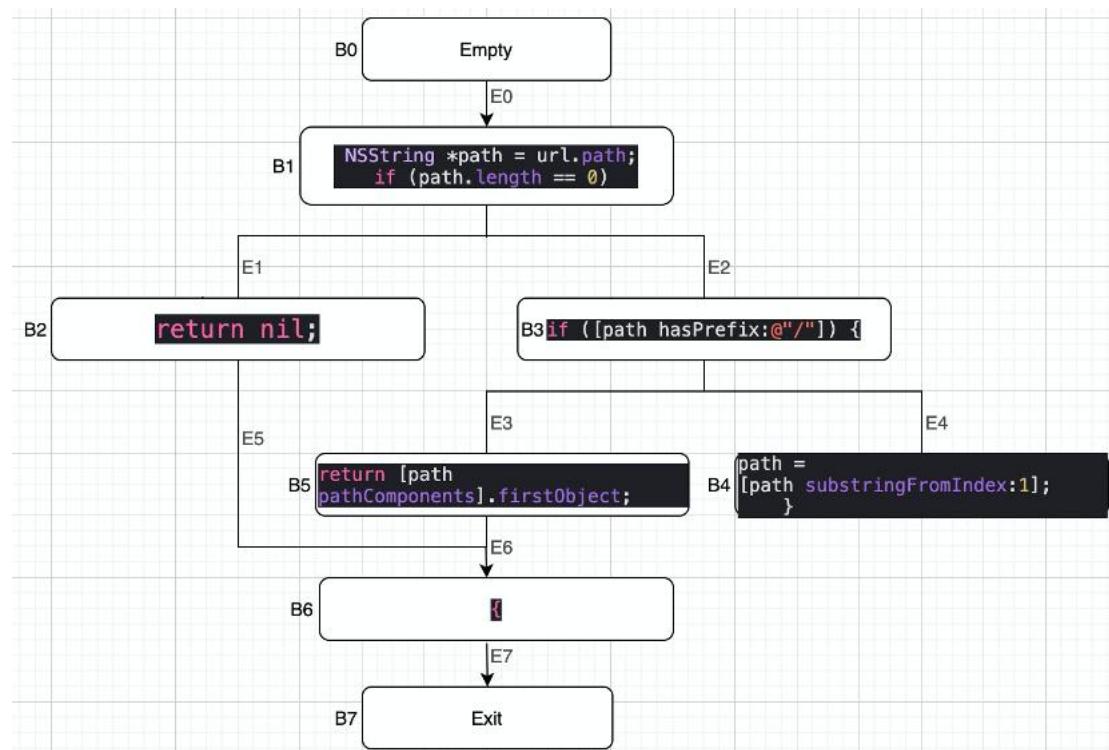
流图(flow graph)的结点就是这些基本块。流图就是通常的图，它可以用任何适合表示图的数据结构来表示。

从基本块 B 到基本块 C 之间有一条边当且仅当基本块 C 的第一个指令紧跟在 B 的最后一个指令之后执行。存在这样一条边的原因有两种：

- 有一个从 B 的结尾跳转到 C 的开头的条件或无条件跳转语句。
- 按照原来的三地址语句序列中的顺序，C 紧跟在 B 之后，且 B 的结尾不存在无条件跳转语句。

我们说 B 是 C 的前驱(predecessor)，而 C 是 B 的一个后继(successor)。

通常会增加两个分部称为入口（entry）和出口（exit）的结点。它们不和任何可执行的中间指令对应。从入口到流图的第一个可执行结点有一条边（edges）。从任何包含了可能是程序的最后执行指令的基本块到出口有一条边。如果程序的最后指令不是一个无条件转移指令，那么包含了程序的最后一条指令的基本块是出口结点的一个前驱。但任何包含了跳转到程序之外的跳转指令的基本块也是出口结点的前驱。



其中 B0–B7 是 BB 块。E0–E7 是边（edges）

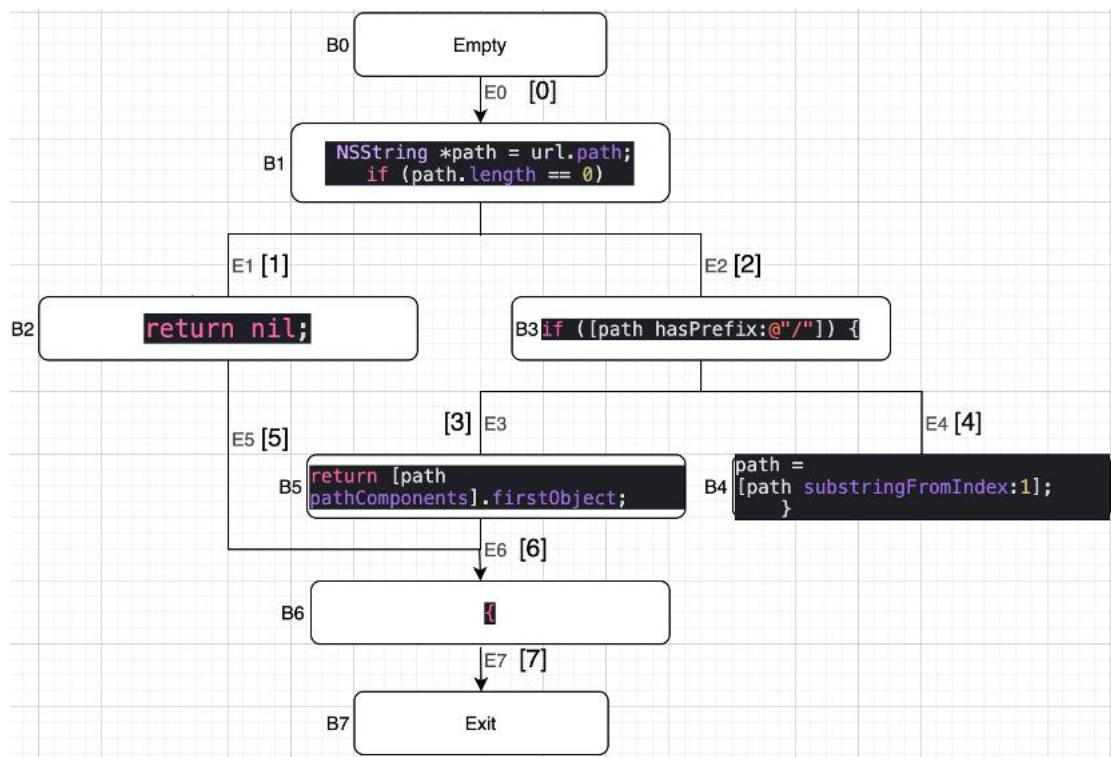
插桩逻辑

覆盖率计数指令的插入会进行两次循环，外层循环遍历编译单元中的函数，内层循环遍历函数的基本块。函数遍历用来向 `gcno` 文件中写入函数位置信息。

一个函数中基本块的插桩方法如下：

- 统计所有 BB 的后继数 n，创建和后继数大小相同的数组 `ctr[n]`。
- 以后继数编号为序号将执行次数依次记录在 `ctr[i]` 位置，对于多后继情况根据条件判断插入。

根据生成流图的规则，可以很容易得到桩点位置，[]处就是插入的桩点序号。



关于工程配置可以参考 GCOV 的官网：

<https://gcc.gnu.org/onlinedocs/gcc/Gcov.html>

下面简单介绍下 gcov, gcno, gcda 这三个 gcc 家族的关键成员。

GCOV

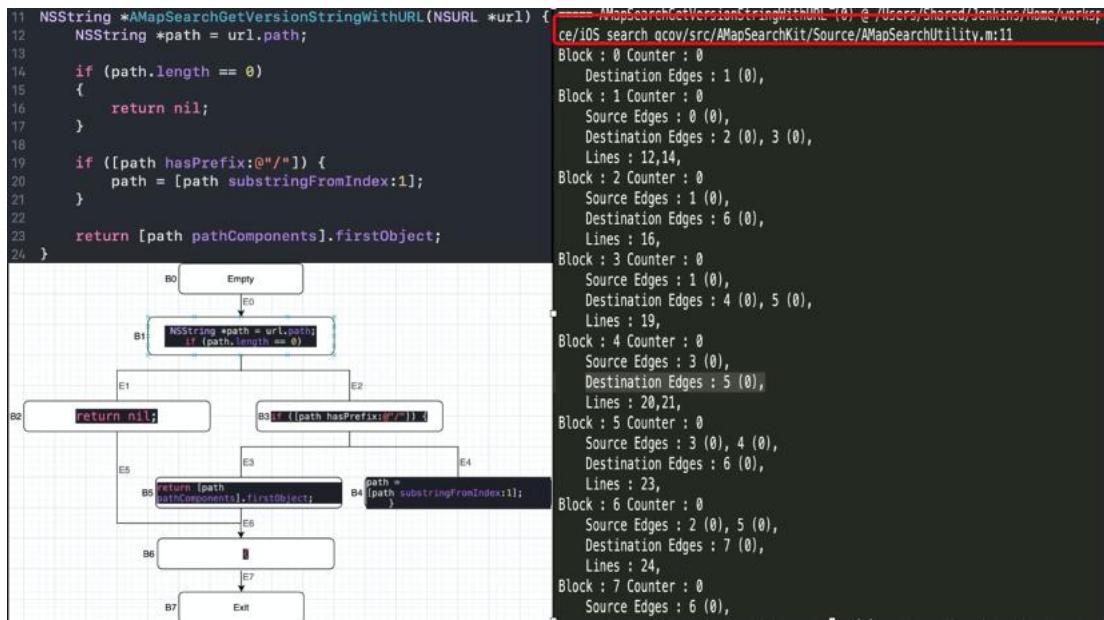
GCOV 是一个 GNU 的本地覆盖测试工具，伴随 GCC 发布，配合 GCC 共同实现对 C 或者 C++ 文件的语句覆盖和分支覆盖测试。是一个命令行方式的控制台程序。需要工具链的支持。

GCNO

利用 Clang 分别生成源文件的 AST 和 IR 文件，对比发现，AST 中不存在计数指令，而 IR 中存在用来记录执行次数的代码。

覆盖率映射关系生成源码是 LLVM 的一个 Pass，用来向 IR 中插入计数代码并生成 .gcno 文件（关联计数指令和源文件）。

大前端篇—iOS 代码染色原理及技术实践



上图右侧。即为 gcno 的可视化格式。

本质上 gcno 是二进制内容。需要借助 gcov 工具(gcov -dump xxx.gcno)将文件转换为这种可视的格式。

其中每个字段的含义

- 函数所在文件的绝对路径（如上图红框所示）。
- Block : 0–7 代表 BB 文件的编号。
- Counter 为插桩后生成的存储执行次数的字段。
- Source Edges 是前继。
- Destination 是后继。
- Lines 是指令在代码文件中行数。

GCDA

gcda 是由加了-fprofile-arcs 编译参数的编译后的文件运行所产生的，它包含了弧跳变的次数和其他的概要信息。

借助 gcov 工具可以查看 gcda 文件的大致内容：

gcda 文件已经是一个包括了函数执行情况的文件。剩余的工作就是将执行情况更加可视化，和源码进行匹配。

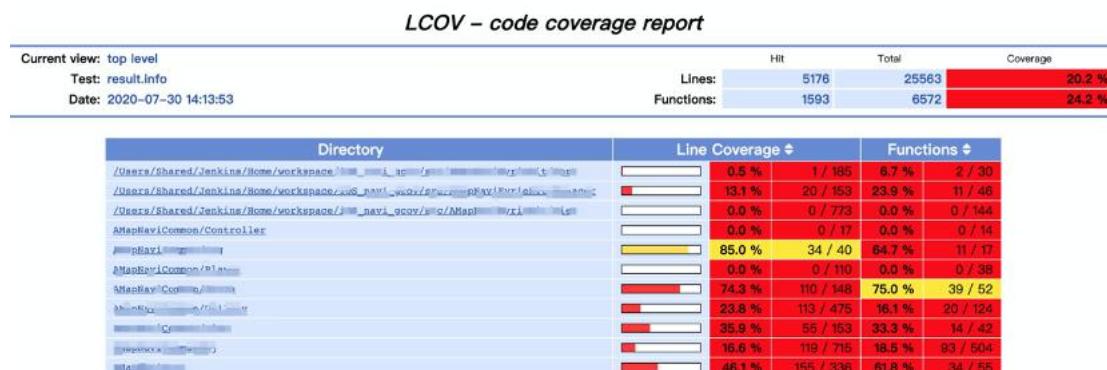
```
j1@JdeMacBook-Pro ~% cd ~/Downloads/gcho_AMapSearchKit_arm64/; gcov -f '/Users/j1/Downloads/gcho_AMapSearchKit_arm64/AMapSearchUtility.gcda'
Function 'AMapSearchGetVersionStringWithURL'
Lines executed:87.50% of 8

Function 'AMapSearchRemovingKeysWithValue'
Lines executed:85.71% of 14

File '/Users/Shared/Jenkins/Home/workspace/iOS_search_gcov/src/AMapSearchKit/Source/AMapSearchUtility.m'
Lines executed:86.36% of 22
/Users/Shared/Jenkins/Home/workspace/iOS_search_gcov/src/AMapSearchKit/Source/AMapSearchUtility.m:creating 'AMapSearchUtility.m.gcov'
```

了解了三个 gc 的重要成员。借助一些前端工具，我们就可以得到一份详细的覆盖率报告了。关于前端工具，大家可以自行搜索。

最后附上覆盖的一个报告片段



技术扩展

了解上述基础知识后，我们更加容易理解 LLVM 中的架构及各个模块的功能。我们可以在插桩过程中，修改原有的插桩逻辑。我们可以编写 XCode 编译器插件。在一个语言的任意处理阶段，我们都可以对其进行定制，甚至我们可以创造一个自己的专属语言。

源码参考：

https://github.com/llvm-mirror/llvm/blob/release_70/lib/Transforms/Instrumentation/GCOVProfiling.cpp

https://llvm.org/doxygen/group__LLVMCoreValueBasicBlock.html#ga444a4024b92a990e9ab311c336e74633

<https://gcc.gnu.org/onlinedocs/gcc/Gcov.html>

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向

Android 端代码染色原理及技术实践

作者：世杰

导读

高德地图开放平台产品不断迭代，代码逻辑越来越复杂，现有的测试流程不能保证完全覆盖所有业务代码，测试不到的代码及分支，会存在一定的风险。为了保证测试全面覆盖，需要引入代码覆盖率做为测试指标，需要对 SDK 代码进行染色，测试结束后可生成代码覆盖率报告，作为发版前的一项重要卡点指标。本文小结了 Android 端代码染色原理及技术实践。

JaCoCo 工具

JaCoCo 有以下优点：

- 支持 Ant 和 Gradle 打包方式，可以自由切换。
- 支持离线模式，更贴合 SDK 的使用场景。
- JaCoCo 文档比较全面，还在持续维护，有问题便于解决。

JaCoCo 主要是通过 ASM 技术对 Java 字节码进行处理和插桩，ASM 和 Java 字节码技术不是本文重点，感兴趣的朋友可以自行了解。下面重点介绍 JaCoCo 的插桩原理。

Jacoco 探针

由于 Java 字节码是线性的指令序列，所以 JaCoCo 主要是利用 ASM 处理字节码，在需要的地方插入一些特殊代码。

我们通过 Test1 方法观察一下 JaCoCo 做的处理。

```
1. //原始java方法
2. public static int Test1(int a, int b) {
3.     int c = a + b;
4.     int d = c + a;
5.     return d;
6. }
7. //-----我是分割线-----
//-----
8. //jacoco 处理后的方法
9. private static transient /* synthetic */ boolean[] $jacocoData;
```

```
10.  
11.    public static int Test1(final int a, final int b) {  
12.        final boolean[] $jacocoInit = $jacocoInit();  
13.        final int c = a + b;  
14.        final int n;  
15.        final int d = n = c + a;  
16.        $jacocoInit[3] = true;  
17.        return n;  
18.    }  
19.    private static boolean[] $jacocoInit() {  
20.        boolean[] $jacocoData;  
21.        if (($jacocoData = TestInstrument.$jacocoData) == null) {  
22.            $jacocoData = (TestInstrument.$jacocoData =  
23.                Offline.getProbes(-6846167369868599525  
24.                L,  
25.                "com/jacoco/test/T  
26.                estInstrument", 4));  
27.        }  
28.        return $jacocoData;  
29.    }
```

可以看出代码中插入了多个 Boolean 数组赋值，自动添加了 jacocoInit 方法和 jacocoData 数组声明。

JaCoCo 统计覆盖率就是标记 Boolean 数组，只要执行过的代码，就对相应角标的 Boolean 数组进行赋值，最后对 Boolean 进行统计即可得出覆盖率，这个数组官方的名字叫探针（Probe）。

探针是由以下四行字节码组成，探针不改变该代码的行为，只记录他们是否已被执行，从理论上讲，可以在每行代码都插入一个探针，但是探针本身需要多个字节码指令，这将增加几倍的类文件的大小和执行速度，所以 JaCoCo 有一定的插桩策略。

1. ALOAD probearray
2. xPUSH probeid
3. ICONST_1
4. BASTORE

探针插桩策略

探针的插入需要遵循一定策略，大体可分成以下三个策略：

- 统计方法的执行情况。
- 统计分支语句的执行情况。
- 统计普通代码块的执行情况。

方法的执行情况

这个比较容易处理，在方法头或者方法尾加就可以了。

- 方法尾加：能说明方法被执行过，且说明了探针上面的方法被执行了，但是这种处理比较麻烦，可能有多个 return 或者 throw。
- 方法头加：处理简单，但只能说明方法有进去过。

通过分析源码，发现 JaCoCo 是在方法结尾处插入探针，return 和 throw 之后都会加入探针。

```
1. public void visitInsn(final int opcode) {  
2.     switch (opcode) {
```

```
3.     case Opcodes.IRETURN:  
4.     case Opcodes.LRETURN:  
5.     case Opcodes.FRETURN:  
6.     case Opcodes.DRETURN:  
7.     case Opcodes.ARETURN:  
8.     case Opcodes.RETURN:  
9.     case Opcodes.ATHROW:  
10.        probesVisitor.visitInsnWithProbe(opcode, idGenerator.nextId());  
11.        break;  
12.    default:  
13.        probesVisitor.visitInsn(opcode);  
14.        break;  
15.    }  
16. }
```

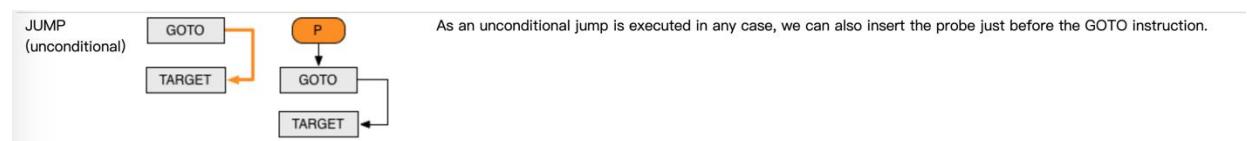
分支的执行情况

Java 字节码通过 Jump 指令来控制跳转，分为有条件 Jump 和无条件 Jump。

- 无条件 Jump (goto)

这种一般出现在 `continue`, `break` 中，由于在任何情况下都执行无条件跳转，因此在 `GOTO` 指令之前插入探针。

官方文档中介绍



示例代码

```

    return d;
}

public static void Test2(int a, int b) {
    boolean[] var2 = $jacocoInit();
    switch(a) {
        case 1:
            b = a + 1;
            var2[5] = true;
            break;
        case 2:
            b = a + 2;
            var2[6] = true;
            break;
        default:
            var2[4] = true;
    }

    var2[7] = true;
}

```

Bytecode View:

```

3: iload_0
4: lookupswitch 2
      1: 39 (+34)
      2: 50 (+45)
      default: 32 (+27)
5: 32 aload_2
6: 33 iconst_4
7: 34 iconst_1
8: 35 bastore
9: 36 goto 59 (+23)
10: 39 iload_0
11: 40 iconst_1
12: 41 iadd
13: 42 istore_1
14: 43 aload_2
15: 44 iconst_5
16: 45 iconst_1
17: 46 bastore
18: 47 goto 59 (+12)
19: 50 iload_0
20: 51 iconst_2
21: 52 iadd
22: 53 istore_1
23: 54 aload_2
24: 55 bipush 6
25: 57 iconst_1
26: 58 bastore
27: 59 aload_2
28: 60 bipush 7
29: 62 iconst_1
30: 63 bastore
31: 64 return

```

有条件 Jump (`if-else`)

这种经常出现于 `if` 等有条件的跳转语句，JaCoCo 会对 `if` 语句进行反转，将字节码变成 `if not` 的逻辑结构。

为什么要对 if 进行反转？下面示例将说明原因。

Test4 方法是一个普通的单条件 if 语句，可以看到 JaCoCo 将 >10 的条件反转成 ≤ 10 ，为什么要进行反转而不是直接在原有 if 后面增加 else 块呢？继续往下看复杂一点的情况。

```
1. //源码
2. public static void Test4(int a) {
3.     if(a>10){
4.         a=a+10;
5.     }
6.     a=a+12;
7. }
8.
9. //jacoco 处理后的字节码
10. public static void Test4(int a) {
11.     boolean[] var1 = $jacocoInit();
12.     if (a <= 10) {
13.         var1[11] = true;
14.     } else {
15.         a += 10;
```

```
16.             var1[12] = true;  
17.         }  
18.         a += 12;  
19.         var1[13] = true;  
20.     }
```

Test5 方法是一个多条件的 if 语句，可以看出来将两个组合条件拆分成单一条件，并进行反转。

这样做好处：可以完整统计到每个条件分支的执行情况，各种条件都会插入探针，保证了完整的覆盖，而反转操作再配合 GOTO 指令可以更简单的插入探针，这里可以看出 JaCoCo 的处理非常巧妙。

```
1. //源码, if 有多个条件  
2. public static void Test5(int a, int b) {  
3.     if(a>10 || b>10){  
4.         a=a+10;  
5.     }  
6.     a=a+12;  
7. }
```

```
8.  
9. //jacoco 处理后的字节码。  
10. public static void Test5(int a, int b) {  
11.     boolean[] var2;  
12.     label15: {  
13.         var2 = $jacocolnit();  
14.         if (a > 10) {  
15.             var2[14] = true;  
16.         } else {  
17.             if (b <= 10) {  
18.                 var2[15] = true;  
19.                 break label15;  
20.             }  
21.             var2[16] = true;  
22.         }  
23.         a += 10;  
24.         var2[17] = true;  
25.     }  
26.     a += 12;  
27.     var2[18] = true;  
28. }
```

可以通过测试报告看出来，标记为黄色代表分支执行情况覆盖不完整，标记为绿色代表分支所有条件都执行完整了。

```
2623.  
2624.     public static void Test3(int a) {  
2625.         ◆ if(a>10 && a>12){  
2626.             a=a+10;  
2627.         }  
2628.         a=a+12;  
2629.     }  
2630.  
623.  
624.     public static void Test3(int a) {  
625.         ◆ if(a>10 && a>12){  
626.             a=a+10;  
627.         }  
628.         a=a+12;  
629.     }  
630.
```

代码块的执行情况

理论上只要在每行代码前都插入探针即可，但这样会有性能问题。JaCoCo 考虑到非方法调用的指令基本都是按顺序执行的，因此对非方法调用的指令不插入探针，而对方法调用的指令之前都插入了探针。

Test6 方法内在调用 Test 方法前都插入了探针。

```
1. public static void Test6(int a, int b) {
```

```
2.         boolean[] var2 = $jacocoInit();
3.         a += b;
4.         b = a + a;
5.         var2[19] = true;
6.         Test();
7.         int var10000 = a + b;
8.         var2[20] = true;
9.         Test();
10.        var2[21] = true;
11.    }
```

源码解析

通过上面的示例，我们暂时通过表面现象理解了探针插入策略。知其然不知其所以然，我们通过源码分析论证一下JaCoCo的真实逻辑，看看JaCoCo是如何通过ASM，来实现探针插入策略的。

源码 MethodProbesAdapter.java 类中，通过 needsProbe 方法判断 Lable 前面是否需要插入探针。

```
1. @Override
```

```
2.     public void visitLabel(final Label label) {  
3.         if (LabelInfo.needsProbe(label)) {  
4.             if (tryCatchProbeLabels.containsKey(label)) {  
5.                 probesVisitor.visitLabel(tryCatchProbeLabels.get(label));  
6.             }  
7.             probesVisitor.visitProbe(idGenerator.nextId());  
8.         }  
9.         probesVisitor.visitLabel(label);  
10.    }
```

下面看一下 needsProbe 方法，主要的限制条件有三个 successor、multiTarget、methodInvocationLine。

```
1.     public static boolean needsProbe(final Label label) {  
2.         final LabelInfo info = get(label);  
3.         return info != null && info.successor  
4.             && (info.multiTarget || info.methodInvocationLine);  
5.     }
```

先看到 successor 属性。顾名思义，表示当前的 Label 是否是前一条 Label 的继任者，也就是说当前指令和上一

条指令是否是连续的，两条指令中间没有插入GOTO或者return。

LabelFlowAnalyzer.java类中，对每行指令进行流程分析，对successor属性赋值。

```
1. boolean successor = false; //默认是 false
2. boolean first = true; //默认是 true
3.
4. @Override
5. public void visitJumpInsn(final int opcode, final Label label) {
6.     LabelInfo.setTarget(label);
7.     if (opcode == Opcodes.JSR) {
8.         throw new AssertionError("Subroutines not supported.");
9.     }
10.    //如果是 GOTO 指令, successor=false, 表示前后两条指令是断开的。
11.    successor = opcode != Opcodes.GOTO;
12.    first = false;
13. }
14.
15. @Override
16. public void visitInsn(final int opcode) {
```

```
17.     switch (opcode) {  
18.         case Opcodes.RET:  
19.             throw new AssertionError("Subroutines not supported.");  
20.         case Opcodes.IRETURN:  
21.         case Opcodes.LRETURN:  
22.         case Opcodes.FRETURN:  
23.         case Opcodes.DRETURN:  
24.         case Opcodes.ARETURN:  
25.         case Opcodes.RETURN:  
26.         case Opcodes.ATHROW:  
27.             successor = false; //return 或者 throw, 表示两条指令是断开的  
28.             break;  
29.     default:  
30.         successor = true; //普通指令的话, 表示前后两条指令是连续的  
31.         break;  
32.     }  
33.     first = false;  
34. }
```

35.

```
36.     @Override  
37.     public void visitLabel(final Label label) {  
38.         if (first) {
```

```
39.         LabelInfo.setTarget(label);  
40.     }  
41.     if (successor) { //这里设置当前指令是不是上一条指令的继任者。  
42.         //源码中，只有这一个地方会触发这个条件赋值，也  
        就是访问每个 label 的第一条指令。  
43.         LabelInfo.setSuccessor(label);  
44.     }  
45. }
```

再看一下 methodInvocationLine 属性，当 ASM 访问到 visitMethodInsn 方法的时候，就标记当前 Label 代表调用一个方法，将 methodInvocationLine 赋值为 True。

```
1. @Override  
2. public void visitLineNumber(final int line, final Label start) {  
3.     lineStart = start;  
4. }  
5.  
6. @Override  
7. public void visitMethodInsn(final int opcode, final String owner,  
8.     final String name, final String desc, final boolean itf) {
```

```
9.     successor = true;  
10.    first = false;  
11.    markMethodInvocationLine();  
12. }  
13.  
14. private void markMethodInvocationLine() {  
15.     if (lineStart != null) {  
16.         //lineStart 就是当前这个Label  
17.         LabelInfo.setMethodInvocationLine(lineStart);  
18.     }  
19. }  
20.  
21. LabelInfo.java 类  
22. public static void setMethodInvocationLine(final Label label) {  
23.     create(label).methodInvocationLine = true;  
24. }
```

再看一下 multiTarget 属性，它表示当前指令是否可能从多个来源跳转过来。源码在下面。

当执行到一条 Jump 语句时，第二个参数表示要跳转到的 Label，这时就会标记一次来源，后续分析流到了

该 Label，如果它还是一条继任者指令，那么就将它标记为多来源指令。

```
1. public void visitJumpInsn(final int opcode, final Label label) {  
2.     LabelInfo.setTarget(label); //Jump 语句 将 Label 标记一次为 true  
3.     if (opcode == Opcodes.JSR) {  
4.         throw new AssertionError("Subroutines not supported.");  
5.     }  
6.     successor = opcode != Opcodes.GOTO;  
7.     first = false;  
8. }  
9.  
10. //如果当设置它是否是上一条指令的后续指令时，再一次设置它为  
11. //multiTarget=true，表示至少有 2 个来源  
12. public static void setSuccessor(final Label label) {  
13.     final LabelInfo info = create(label);  
14.     info.successor = true;  
15.     if (info.target) {  
16.         info.multiTarget = true;  
17.     }  
18. }
```

特殊问题解答

有了前面对源码的分析，再来看一些特殊情况。

问：else 块结尾为什么会插入探针？

答：L3 的来源有两处，一处是 GOTO 来的，一处是 L1 顺序执行来的，使得 multiTarget = true 条件成立，所以在 L3 之前插入探针，表现在 Java 代码中就是在 else 块结尾增加了探针。

```

        b = a + 1;
        var2[5] = true;
        break;
    case 2:
        b = a + 2;
        var2[6] = true;
        break;
    default:
        var2[4] = true;
    }

    var2[7] = true;
}

public static void Test3(int a, int b) {
    boolean[] var2 = $jacocoInit();
    if (a > 10) {
        a = b + 1;
        var2[8] = true;
    } else {
        a = b + 1;
        var2[9] = true;
    }
    var2[10] = true;
}

public static void Test4(int a) {
    boolean[] var1 = $jacocoInit();
    if (a <= 10) {
        var1[11] = true;
    } else {
        a += 10;
        var1[12] = true;
    }
    a += 12;
    var1[13] = true;
}

```

```

142 // access flags 0x9
143 public static Test3(II)V
144     INVOKESTATIC com/jacoco/test/TestInstrument.$jacocoInit ()[Z
145         ASTORE 2
146         L0
147         LINENUMBER 29 L0
148         ILOAD 0
149         BIPUSH 10
150         IF_ICMPLE L1
151         L2
152         LINENUMBER 30 L2
153         ILOAD 1
154         ICONST_1
155         IADD
156         ISTORE 0
157         ALOAD 2
158         BIPUSH 8
159         ICONST_1
160         BASTORE
161         GOTO L3
162         L1
163         LINENUMBER 32 L1
164         ILOAD 1
165         ICONST_1
166         IADD
167         ISTORE 0
168         ALOAD 2
169         BIPUSH 9
170         ICONST_1
171         BASTORE
172         L3
173         LINENUMBER 34 L3
174         ALOAD 2
175         BIPUSH 10
176         ICONST_1
177         BASTORE
178         RETURN
179
180         LOCALVARIABLE a I L0 L4 0
181         LOCALVARIABLE b T L0 L4 1

```

问：为什么 case 1 条件里第一个 Test 方法前不插入探针？

答：L1 上一条是指 GOTO 指令，使得 successor = false，所以该方法调用前无需插入探针。

```

        b = a * i;
        var2[5] = true;
        break;
    case 2:
        b = a + 2;
        var2[6] = true;
        break;
    default:
        var2[4] = true;
    }
    var2[7] = true;
}

public static void Test3(int a, int b) {
    boolean[] var2 = $jacocoInit();
    if (a > 10) {
        a = b + 1;
        var2[8] = true;
    } else {
        a = b + 1;
        var2[9] = true;
    }
    var2[10] = true;
}

public static void Test4(int a) {
    boolean[] var1 = $jacocoInit();
    if (a <= 10) {
        var1[11] = true;
    } else {
        a += 10;
        var1[12] = true;
    }
    a += 12;
    var1[13] = true;
}

```

// access flags 0x9
public static Test3(II)V
 INVOKESTATIC com/jacoco/test/TestInstrument.\$jacocoInit ()[Z
 ASTORE 2
 L0
 LINENUMBER 29 L0
 ILOAD 0
 BIPUSH 10
 IF_ICMPLE L1
 L2
 LINENUMBER 30 L2
 ILOAD 1
 ICONST_1
 IADD
 ISTORE 0
 ALOAD 2
 BIPUSH 8
 ICONST_1
 BASTORE
GOTO L3
 L1
 LINENUMBER 32 L1
 ILOAD 1
 ICONST_1
 IADD
 ISTORE 0
 ALOAD 2
 BIPUSH 9
 ICONST_1
 BASTORE
L3
 LINENUMBER 34 L3
 ALOAD 2
 BIPUSH 10
 ICONST_1
 BASTORE
 RETURN
L4
 LOCALVARIABLE a I L0 L4 0
 LOCALVARIABLE b T L0 L4 1

探针插桩结论

通过以上分析得出结论，代码块中探针的插入策略：

- return 和 throw 之前插入探针。
- 复杂 if 语句，为统计分支覆盖情况，会进行反转成 if not，再对个分支插入探针。
- 当前指令是上一条指令的连续，并且当前指令是触发方法调用，则插入探针。

- 当前指令和上一条指令是连续的，并且是有多个来源的时候，则插入探针。

构建 SDK 染色包

利用 JaCoCo 提供的 Ant 插件，在原有打包脚本上进行修改。

- Ant 脚本根节点增加 JaCoCo 声明。
- 引入 jacocoant 自定义 task。
- 在 compile task 完成之后，运行 instrument 任务，对原始 classes 文件进行插桩，生成新的 classes 文件。
- 将插桩后的 classes 打包成 jar 包，不需要混淆，就完成了染色包的构建。

```
1. <project name="Example" xmlns:jacoco="antlib:org.jacoco.ant"> //增加  
    jacoco 声明  
2.      //引入自定义 task
```

```
3.      <taskdef uri="antlib:org.jacoco.ant" resource="org/jacoco/ant/antlib.xml">
4.          <classpath path="path_to_jacoco/lib/jacocoant.jar"/>
5.      </taskdef>
6.
7.      ...
8.      //对 classes 插桩
9.      <jacoco:instrument destdir="target/classes-instr" depends="compile"
   >
10.         <fileset dir="target/classes" includes="**/*.class"/>
11.     </jacoco:instrument>
12.
13. </project>
```

测试工程配置

将生成的染色包放入测试工程 lib 库中，测试工程 build.gradle 配置中开启覆盖率统计开关。

官方 gradle 插件默认自带 JaCoCo 支持，需要开启开关。

```
testCoverageEnabled = true //开启代码染色覆盖率统计
```

收集覆盖率报告的方式有两种，一种是用官方文档里介绍的：配置 jacoco-agent.properties 文件，放 Demo 的 resources 资源目录下。

文件配置生成覆盖率产物的路径，然后测试完 Demo，在终止 JVM 也就是退出应用的时候，会自动将覆盖率数据写入，这种方式不方便对覆盖率文件命名自定义，多轮测试产物不明确。

```
destfile=/sdcard/jacoco/coverage.ec
```

另一种方式是利用反射技术：反射调用 jacoco.agent.rt.RT 类的 getExecutionData 方法，获取上文中探针的执行数据，将数据写入 sdcard 中，生成 ec 文件。这段代码可以在应用合适位置触发，推荐退出之前调用。

```
1.      /**
2.      * 生成 ec 文件
3.      */
4.      public static void generateEcFile(boolean isNew, Context context) {
5.          File file = new File(DEFAULT_COVERAGE_FILE_PATH);
6.          if(!file.exists()) {
```

```
7.         file.mkdir();
8.     }
9.     DEFAULT_COVERAGE_FILE = DEFAULT_COVERAGE_FILE_PATH +
   File.separator+ "coverage"+getDate()+" .ec";
10.    Log.d(TAG, "生成覆盖率文
件: " + DEFAULT_COVERAGE_FILE);
11.    OutputStream out = null;
12.    File mCoverageFilePath = new File(DEFAULT_COVERAGE_FILE);
13.    try {
14.        if (!mCoverageFilePath.exists()) {
15.            mCoverageFilePath.createNewFile();
16.        }
17.        out = new FileOutputStream(mCoverageFilePath.getPath(), t
rue);
18.
19.        Object agent = Class.forName("org.jacoco.agent.rt.RT")
20.                .getMethod("getAgent")
21.                .invoke(null);
22.
23.        out.write([byte[]]) agent.getClass().getMethod("getExecutionD
ata", boolean.class)
24.                .invoke(agent, false));
```

```
25.             Log.d(TAG, "写入" + DEFAULT_COVERAGE_FILE + "完成!");
26.             Toast.makeText(context, "写入" + DEFAULT_COVERAGE_FILE + "完成!", Toast.LENGTH_SHORT).show();
27.         } catch (Exception e) {
28.             Log.e(TAG, "generateEcFile: " + e.getMessage());
29.             Log.e(TAG, e.toString());
30.         } finally {
31.             if (out == null)
32.                 return;
33.             try {
34.                 out.close();
35.             } catch (IOException e) {
36.                 e.printStackTrace();
37.             }
38.         }
39.     }
40. }
```

覆盖率报告生成

JaCoCo 支持将多个 ec 文件合并，利用 Ant 脚本即可。

```
1. <jacoco:merge destfile="merged.exec">  
2.     <fileset dir="executionData" includes="*.exec"/>  
3. </jacoco:merge>
```

将 ec 文件从手机导出，配合插桩前的 classes 文件、源码文件(可选)，配置 Ant 脚本中，就可以生成 Html 格式的覆盖率报告。

```
1. <jacoco:report>  
2.  
3.     <executiondata>  
4.         <file file="jacoco.exec"/>  
5.     </executiondata>  
6.  
7.     <structure name="Example Project">  
8.         <classfiles>  
9.             <fileset dir="classes"/>  
10.        </classfiles>  
11.        <sourcefiles encoding="UTF-8">  
12.            <fileset dir="src"/>  
13.        </sourcefiles>  
14.    </structure>
```

```
15.  
16.    <html destdir="report" />  
17.  
18. </jacoco:report>
```

熟悉 Java 字节码技术、ASM 框架、理解 JaCoCo 插桩原理，可以有各种手段玩转 SDK，例如在不修改源码的情况下，在打包阶段可以动态插入和删除相应代码，完成一些特殊需求。

参考链接

<https://www.jacoco.org/jacoco/trunk/doc/index.html>

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

八个维度对低代码能力度量模型的思考

作者：子羽

写在前面

low-code 大旗之下，各式各样的低代码平台熙熙攘攘：

- 应用场景：PC 中后台、移动 H5、小程序，也有 React Native 等跨端；
- 核心功能：UI 编排、（逻辑）流程编排，甚至服务编排；
- 交互方式：表单配置、拖拽，甚至还有富文本扩展。

作为低代码领域的探索者，或许都有过共同的困惑：

- 现有的商业化成熟产品能否满足我们的实际业务需要？

- 站在巨人的肩膀上，我们面临的真正问题是什
么？
- 我们当下处于哪个阶段？下一阶段在哪里？如何
通往下一阶段？

为了解开这些疑惑，我们尝试建立一个能力度量模型，
让低代码平台的变化有迹可循。

1. 业务场景

能力模型的第一维是业务场景，覆盖到的业务场景越
多，低代码能力越强。

从不同角度可对业务场景进行不同的划分，例如：

- 产品：2C、中后台；
- 业务：营销活动、反馈表单、常规图文展现、复
杂富交互；
- 端：移动端、PC Web、小程序。

这样的大类不一定适合所有业务，可根据业务重要性（核心、重要、边缘），差异程度（所用的技术体系、面向的用户群体）等进行具体划分。必要的话，还可以细分出各个子维度。目的是将现有的低代码能力对目标业务场景的支持程度定量地描述清楚，平台已经能满足哪几类业务场景，未来还能够满足哪些？

在垂直场景划分的基础上，还可以衍生出跨业务线投放、跨端搭投、一搭多投等混合的探索方向。

2. 用户群体

第二维是低代码平台面向的用户群，用户群体越大，低代码能力越强。

一般按用户的专业程度分为：

- 特定技术人员：前端、后端、DBA 等专业人员；
- 一般技术人员：有一定逻辑编码能力的开发人员，能够快速理解并运用表达式、事件等概念；

- 非技术人员：没有开发经验的产品、运营、商务、行政人员。

如果平台的最终目标是面向非技术人员，那么就要求对功能进行高度抽象，屏蔽下层技术细节，以降低使用门槛，将更广泛的用户纳入进来。

另一方面，用户的数量也是衡量低代码能力的重要指标，覆盖用户量越大、不同属性（团队、部门、第三方）的用户越多，越能体现低代码平台的成熟度。

3. 能力完整性

第三维是能力完整性（即技术表达力的完备程度），完整性越高，低代码能力越强。

目标业务场景中，能力完备的低代码开发平台具有与源码开发等同的技术表达力，即，人工写代码能实现的东西，就一定能通过低代码开发平台来完成。

以 Web App 为例，能力完整性要求低代码平台能够表达 UI（含交互效果）、前端业务逻辑、接口调用、甚至后端业务逻辑、数据模型等，能够替代源码开发，

目标用户可通过平台完成目标需求的全部开发工作，而不是受限于平台能力，只能完成某一部分工作。

4. 原料包容性

第四维是原料包容性，即低代码平台对不同输入的接受能力，例如：

- 是否支持录入现有业务组件、模块？
- 是否支持引用任意第三方模块？
- 是否支持引入非标准模块？

源码开发的一大优势在于能够最大限度地复用现有代码，无论是公共组件/业务组件、第三方模块，甚至非标准模块，都可以随时通过封装引入，甚至源码拷贝的方式来复用。而低代码平台则不同，对于组件、模块大都有明确的准入规则，只有符合标准的“原料”才能进入到池子中，供平台用户复用。

一种简单的做法是按组件的通用程度分为公共组件与业务组件（模块的处理与组件类似，本节不再严格区

分），平台只收录通用的公共组件，极大地简化了组件版本管理，但这种划分对于长期持续迭代的业务并不适用，由于无法复用现成的代码，低代码模式下开发效率远低于高度复用的源码开发。

因此，更好的做法是按标准程度将组件分为标准组件与定制组件：

- 标准组件：平台预置的组件；
- 定制组件：平台用户可随时引入的其它组件。

只有允许用户录入定制组件，才能够满足其代码复用需求，让开发效率重新回到同一水平。因为对于长期迭代的业务而言，日常使用最频繁的一定是业务组件，而不是通用的公共组件。这种情况下，如何录入定制组件、如何支持定制组件与标准组件混用是值得深入探索的方向。

5. 产物丰富度

第五维是产物丰富度，平台输出的产物形态越丰富，低代码能力越强。

输出产物可分为 3 类：

- 最终产物：功能模块、页面、应用；
- 中间产物：业务组件、区块、模版；
- 初级产物：UI 组件。

最终产物的完成度最高，但可复用程度最低，初级产物与之相反。多种形态的输出产物意味着强大的可复用性和灵活的集成方式，例如：

- 低代码开发与源码开发混合使用，允许平滑过渡；
- 基于低代码平台产出的半成品二次开发，减轻一部分工作量。

也就是说，能力完整性决定了目标应用场景，而产物的丰富度决定着低代码平台的实际应用场景。

6. 链路覆盖度

第六维是链路覆盖度，表示对完整生产链路的覆盖程度，覆盖度越高，低代码能力越强。

完整的生产链路一般包括需求-设计-开发-测试-发布-运维，低代码平台对生产链路的覆盖越完整，协作流程越顺畅，效率提升也越明显。不同业务环境中，具体的生产链路可能不尽相同，但都需要明确低代码平台的链路覆盖范围，不断优化覆盖范围内的环节，同时尽可能降低与范围外各个环节的协作成本。

具体的，提升链路覆盖度有 2 种方式：

- 并入：将范围外的环节也拿进来，比如在支持开发的基础上，提供测试、发布流程管理，以及相应的一键部署能力，对必要流程提供尽可能完善的支持，避免将低代码平台与生产链路上下游的接缝暴露给用户，由人工来填补；
- 连通：与范围外的环节连接起来，例如，一个表达力很有限的低代码平台可能需要与源码开发模式配合使用，此时可以考虑与源码开发中的代码

仓库联动，将产物一键上传至代码库，或者反过来将低代码能力嵌入到 IDE 中，辅助源码开发。

要覆盖生产全链路不一定非要把所有环节都纳入到低代码开发平台中，只需要打通数据链路，与现有工具、平台联动起来即可，例如：

```
1      原料协议  
2 物料资产 -----> 低代码平台  
3      产物协议  
4 低代码平台 -----> 发布平台/代码仓库  
5      中间产物协议  
6 UED设计工具 -----> 低代码平台  
7      接口描述协议  
8 API管理平台 -----> 低代码平台  
9      数据描述协议  
10 低代码平台 -----> 数据Mock平台
```

7. 协作效率

第七维是协作效率，指的是不同角色在低代码模式下的协同工作效率，协作效率越高，低代码能力越强。

不同于源码开发，低代码开发作为一种新的研发模式，在协作效率方面有很大的想象空间，例如：

- 产品经理：可通过低代码平台产出高保真原型，交由研发人员进一步开发，甚至能够自行快速调整文案、图片素材等；
- UED：设计工具对接低代码平台，无需人工标注、走查效果。

Design2Code（设计稿转代码）是解决 UED 与研发人员的协作效率问题的另一种思路，相比之下，低代码平台的核心优势在于降低了专业性要求，使得产品经理、UED 等非技术人员也有能力自主调整，甚至独立完成部分需求。

8. 智能程度

第八维是智能程度，越智能，低代码能力越强。

首先，如何定义智能？

简单定义为帮助甚至代替人工决策的能力，也就是说，程序能够自动做出（我也认为正确的）决定，那么它就是智能的。例如现代 IDE 能够根据海量代码库词频特征、当前输入上下文、用户编码习惯等信息综合计

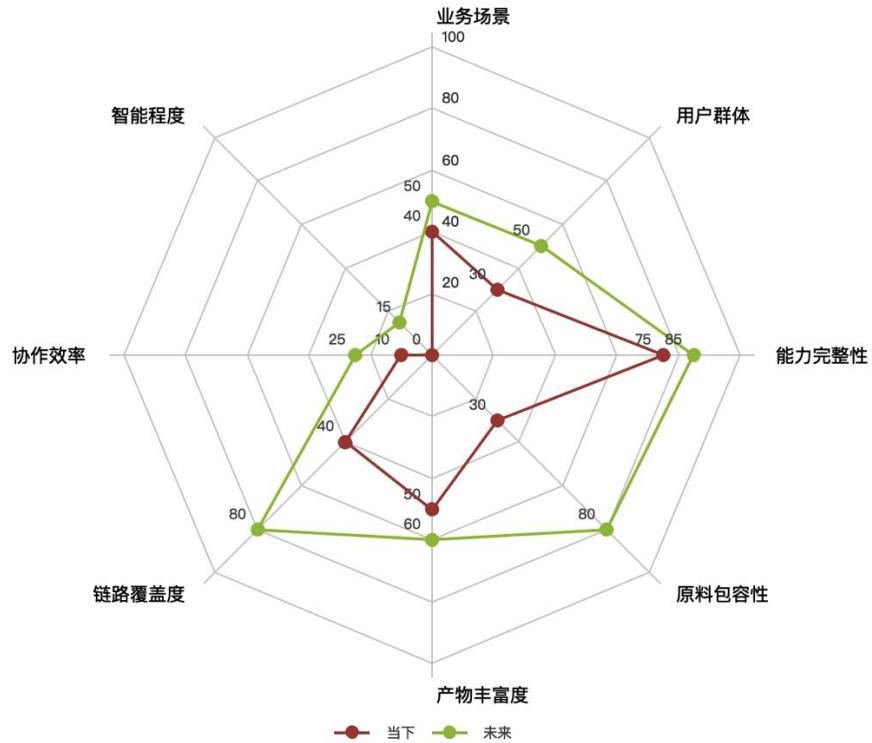
算得到最有可能的几个备选项作为补全提示，大概率是我要输入的内容，所以称之为智能提示。

配置化（数据化）的低代码开发是走向智能化开发的必经之路，因为智能的基础是数据，基于大数据集分析得出的规律是程序决策的重要依据。而源码开发由于其灵活性，并不能提供细致的有效输入，低代码平台限制了人工编码的灵活性，提供了一种配置化的程序表达方式，产生的配置数据能够作为推荐算法的输入，进而帮助人工决策：

- 自动推荐/选择内容，如布局模版、组件样式（字号、颜色）、图片素材等；
- 自动推荐/选择文案，如关键字、句式、风格等；
- 甚至自动产生大量 UI 组合，均等投放验证效果，根据效果反馈自动选择最佳。

让部分生产环节从人工决策走向自动化的数据驱动决策，低代码平台在这样的智能化进程中起着不可替代的作用。

至此，八维低代码能力度量模型已经建立起来了：



- 现有的商业化成熟产品能否满足我们的实际业务需要？
- 站在巨人的肩膀上，我们面临的真正问题是什么？
- 我们当下处于哪个阶段？下一阶段在哪里？如何通往下一阶段？

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

以高德为例，超级 APP 启动提速的实践和思考

作者：戴铭



前言

启动是门面，好的印象也助于留存率提高。苹果也在系统启动上不断努力，提升用户体验，最新的 M1 宣传中还特别强调了翻盖秒开 macOS，可以看出其对极致启动速度的追求。

[这篇文章](#)提到，据 [Akamai 调查](#)，每多 1 秒等待，转化率会下

降 7%，KissMetrics 的一份报告说启动超 5 秒，会使 19% 的用户放弃等待并卸载 APP。



高德 APP 启动优化专项完成后已经有一段时间了，一直保持实属不易，我一年前在[这篇文章](#)里也做了些总结。我这里再补充些启动优化用到的一些手段和想法，希望能够对你有帮助。

通过 Universal Links 和 APP Links 优化唤端启动体验

APP 都会存在拉新和导流的诉求，如何提高这样场景的用户体验呢？这里会用到唤端技术。包含选择什么样的唤端协议，我们先看看唤端路径，如下：



唤端的协议分为自定义协议和平台标准协议，自定义协议在 iOS 端会有系统提示弹框，在 Android 端 Chrome 25 后自定义协议失效，需用 Intent 协议包装才能打开 APP。如果希望提高体验最好使用平台标准协议。平台标准协议在 iOS 平台叫 Universal Links，在 iOS 9 开始引入的，所以 iOS 9 及以上系统都支持，如果用户安装了要跳的 APP 就会直接跳到 APP，不会有系统弹框提示。相对应的 Android 平台标准协议叫 APP Links，Android 6 以上都支持。

这里需要注意的是 iOS 的 Universal Links 不支持自动唤端，也就是页面加载后自动执行唤端是不行的，需要用户主动点击进行唤端。对于自定义协议和平台标准协议在有些 APP 里是遇到屏蔽或者那些 APP 自定义弹窗提示，这就只能通过沟通加白来解决了。

另外对于启动时展示 H5 启动页，或唤端跳转特定功能页，可以将拦截判断置前，判断出启动去往功能页，优先加载功能页的任务，主图相关任务项延后再加载，以提升启动到特定页面的速度。

H5 启动页

现在 APP 启动会在有活动时先弹出活动运营 H5 页面提高活动曝光率。但如果 H5 加载慢势必非常影响启动的体验。iOS 的话可以使用 ODR(On-Demand Resources) 在安装后先下载下来，点击启动前实际上就可以直接加载本地的了。

ODR 安装后立刻下载的模式，下载资源会被清除，所以需要将下载内容移动到自定义的地方，同时还需要做自己兜底的

下载来保证在 On-Demand Resources 下载失败时，还能够再从自己兜底服务器上拉下资源。

On-Demand Resources 还能够放很多资源，甚至包括脚本代码的预加载，可以减少包体积。由于使用的是苹果服务器，还能够减少 CDN 产生的峰值成本。

如果不使用 On-Demand Resources 也可以对 WKWebView 进行预加载，虽然安装后第一次还是需要从服务器上加载一次，不过后面就可以从本地快速读取了。

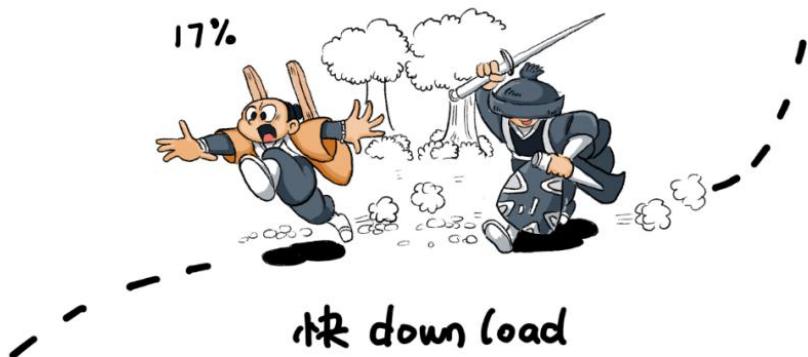
iOS 有三套方案，一套是通过 WKBrowsingContextController 注册 scheme，使用 URLProtocol 进行网络拦截。第二套是基于 WKURLSchemeHandler 自定义 scheme 拦截请求。第三套是在本地搭建 local server，拦截网络请求重定向到本地 server。第三套搭建本地 server 成本高，启动 server 比较耗时。第二套 WKURLSchemeHandler 使用自定义 scheme，对于 H5 适配成本很高，而且需要 iOS 11 以上系统支持。

第一套方案是使用了 WKBrowsingContextController 的 registerSchemeForCustomProtocol：这个方法，这个方法的参数设

置为 http 或 https 然后执行，后面这类 scheme 就能够被 NSURLProtocol 处理了，具体实现可以[在这里](#)看到。

Android 通过系统提供的资源拦截Api即可实现加载拦截，拦截后根据请求的url识别资源类型，命中后设置对应的mimeType、encoding、fileStream即可。

下载速度



APP 安装前的下载速度也直接影响到了用户从选择你的 APP 到使用的体验，如果下载大小过大，用户没有耐心等待，可能就放弃了你的 APP，4G5G 环境下超 200MB 会弹窗提示是否继续下载，严重影响转化率。

因此还对下载大小做了优化，将 `_TEXT` 字段迁移到自定义段，使得 iPhone X 以前机器的下载大小减少了 $1/3$ 的大小，这招之所以对 iPhone X 以前机器管用的原因是因为先前机器是按照先加密再压缩，压缩率低，而之后机器改变了策略因此下载大小就会大幅减少。Michael Eisel 这篇博客 [《One Quick Way to Drastically Reduce your iOS APP's Download Size》](#) 提出了这套方案，你可以立刻应用到自己应用中，提高老机器下载速度，效果立竿见影。

Michael Eisel 还用 Swift 包装了 simdjson 写了个库 ZippyJSONDecoder 比系统自带 JSONDecoder 快三倍，很符合本篇“要更快”的主题，人类对速度的追求是没有止境的，最近 YY 大神 ibireme 也在写 JSON 库 [YYJSON](#) 速度比 simdjson 还快。

Michael 还写了个提速构建的自制链接器 zld，项目说明还描述了如何开发定制自己的链接器。还有主线程阻塞（ANR）检测的 swift 类 [ANRChecker](#)，还有通过 hook 方式记录系统错误日志的 [例子](#) 展示如何通过截获自动布局错误，函数是 `UIAlertViewForUnsatisfiableConstraints`，`malloc` 问题替换函数为 `malloc_error_break` 即可。Michael 的这些性能问题处理手段非常实用。

通过每月新增激活量、浏览到新增激活转换率、下载到激活转换率、转换率受体积因素影响占比、每个用户获取成本，使用公式计算能够得到每月成本收益，把你们公司对应具体参数数值套到公式中，算出来后你会发现如果降低了 50 多 MB，每月就会有非常大的收益。

对于 Android 来说，很多功能是可以放在云端按需下载使用，后面的方向是重云轻端，云端一体，打通云端链路。

下载和安装完成后，就要分析 APP 开始启动时如何做优化了，我接下来跟你说说 Android 启动 so 库加载如何做监控和优化。

Android so 库加载优化

编译阶段—静态分析优化

依托自动化构建平台，通过构建配置实现对源码模块的灵活配置，进行定制化编译。

1. `-ffunction-sections -fdata-sections` // 实现按需加载
2. `-fvisibility=hidden -fvisibility-inlines-hidden` // 实现符号隐藏

这样可以避免无用模块的引入，效果如下图：

[Nr]	Name	Type	Size	ES	Flg	Lk	Inf	Al
[0]		NULL	000000 00		0	0	0	0
[1]	.dynsym	DYNSYM	0156b0 18	A	2	1	4	
[2]	.dynstr	STRTAB	044f58 00	A	0	0	1	
[3]	.hash	HASH	0095c0 04	A	1	0	4	
[4]	.gnu.version	VERSYM	002a06 02	A	1	0	2	
[5]	.gnu.version_d	VERDEF	00001c 00	A	2	1	4	
[6]	.gnu.version_r	VERNEED	000060 00	A	2	3	4	
[7]	.rel.dyn	REL	075918 00	A	1	0	4	
[8]	.rel.plt	REL	000720 00	A1	3	4		
[9]	.plt	PROGBITS	00082a0 00	AX	0	0	4	
[10]	.text	PROGBITS	12de814 00	AX	0	0	16	
[11]	.ARM.extab	PROGBITS	0fa04 00	A	0	0	4	
[12]	.ARM.exidx	ARM_EXIDX	077ec8 00	AL	18	0	4	
[13]	.rodata	PROGBITS	055c8c 00	A	0	0	16	
[14]	.data.rel.ro.loca	PROGBITS	000d90 00	WA	0	0	8	
[15]	.fini_array	FINI_ARRAY	000008 00	WA	0	0	4	
[16]	.data.rel.ro	PROGBITS	0444b4 00	WA	0	0	16	
[17]	.init_array	INIT_ARRAY	000500 00	WA	0	0	4	
[18]	.dynamic	DYNAMIC	000130 00	WA	2	0	4	
[19]	.data	PROGBITS	230b00 00	WA	0	0	16	
[20]	.got	PROGBITS	0002d0 00	WA	0	0	4	
[21]	.note	NODET	020000 00	WA	0	0	16	
[22]	.comment	PROGBITS	0000d6 01	MS	0	0	1	
[23]	.note.gnu.gold-ve	NOTE	00001c 00		0	0	4	
[24]	.ARM.attributes	ARM_ATTRIBUTES	00003c 00		0	0	1	
[25]	.shstrtab	STRTAB	000103 00		0	0	1	

[Nr]	Name	Type	Size	ES	Flg	Lk	Inf	Al
[0]		NULL	000000 00		0	0	0	0
[1]	.dynsym	DYNSYM	00b750 18	A	2	1	4	
[2]	.dynstr	STRTAB	01c9e0 00	A	0	0	1	
[3]	.hash	HASH	004df0 04	A	1	0	4	
[4]	.gnu.version	VERSYM	0016ea 02	A	1	0	2	
[5]	.gnu.version_d	VERDEF	00001c 00	A	2	1	4	
[6]	.gnu.version_r	VERNEED	000060 00	A	2	3	4	
[7]	.rel.dyn	REL	0745a8 00	A	1	0	4	
[8]	.rel.plt	REL	000720 00	A1	3	4		
[9]	.plt	PROGBITS	00082a0 00	AX	0	0	4	
[10]	.text	PROGBITS	12b1fd0 00	AX	0	0	16	
[11]	.ARM.extab	PROGBITS	0f8d70 00	A	0	0	4	
[12]	.ARM.exidx	ARM_EXIDX	07e5f0 00	AL	18	0	4	
[13]	.rodata	PROGBITS	055694 00	A	0	0	16	
[14]	.data.rel.ro.loca	PROGBITS	000d00 00	WA	0	0	8	
[15]	.fini_array	FINI_ARRAY	000008 00	WA	0	0	4	
[16]	.data.rel.ro	PROGBITS	0444dc 00	WA	0	0	16	
[17]	.init_array	INIT_ARRAY	000500 00	WA	0	0	4	
[18]	.dynamic	DYNAMIC	000130 00	WA	2	0	4	
[19]	.data	PROGBITS	23eb44 00	WA	0	0	16	
[20]	.got	PROGBITS	002d74 00	WA	0	0	4	
[21]	.bss	NODET	020000 00	WA	0	0	16	
[22]	.comment	PROGBITS	0000d6 01	MS	0	0	1	
[23]	.note.gnu.gold-ve	NOTE	00001c 00		0	0	4	
[24]	.ARM.attributes	ARM_ATTRIBUTES	00003c 00		0	0	1	
[25]	.shstrtab	STRTAB	000103 00		0	0	1	

运行阶段-hook 分析优化

Android Linker 调用流程如下：



注意，find_library 加载成功后返回 soinfo 对象指针，然后调用其 call_constructors 来调用 so 的 init_array。call_constructors

调用 call_array，其内部循环调用 call_funtion 来访问 init_array 数组的调用。

高德 Android 小伙伴们基于 [frida-gum](#) 的 hook 引擎开发了线下性能监控工具，可以 hook c++ 库，支持 macos、android、ios，针对 so 的全局构造时间和链接时间进行 hook，对关键 so 加载的关键节点耗时进行分析。dlopen 相关 hook 监控点如下：

```
1. static target_func_t android_funcs_22[] = {  
2.     {"__dl_dlopen", 0, (void *)my_dlopen},  
3.     {"__dl_ZL12find_libraryPKciPK12android_dlextinfo", 0, (void *)my_fin  
d_library},  
4.     {"__dl_ZN6sinfo16CallConstructorsEv", 0, (void *)my_sinfo_CallCon  
structors},  
5.     {"__dl_ZN6sinfo9CallArrayEPKcPPFvvEjb", 0, (void *)my_sinfo_Call  
Array}  
6. };  
7.  
8. static target_func_t android_funcs_28[] = {  
9.     {"__dl_Z9do_dlopenPKciPK17android_dlextinfoPKv", 0, (void *)my_do  
_dlopen_28},  
10.    {"__dl_Z14find_librariesP19android_namespace"},
```

```
11.     {"__dl_ZN6sinfo17call_constructorsEv", 0, (void *)my_sinfo_CallConstructors} ,  
12.     {"__dl_ZL10call_arrayIPFviPPcS1_EEvPKcPT_jbS5_", 0, (void *)my_call_array<constructor_func>} ,  
13.     {"__dl_ZN6sinfo10link_imageERK10LinkListIS"} ,  
14.     {"__dl_g_argc", 0, 0} ,  
15.     {"__dl_g_argv", 0, 0} ,  
16.     {"__dl_g_envp", 0, 0}  
17. } ;
```

Android 版本不同对应 hook 方法有所不同，要注意当 so 有其他外部链接依赖时，针对 dlopen 的监控数据，不只包括自身部分，也包括依赖的 so 部分。在这种情况下，so 加载顺序也会产生很大的影响。

JNI_OnLoad 的 hook 监控代码如下：

```
1. #ifdef ABTOR_ANDROID  
2. jint my_JNI_ONLoad(JavaVM* vm, void* reserved) {  
3.     asl::HookEngine::HookContext *ctx = asl::HookEngine::getHookContext();
```

```
4.  
5.     uint64_t start = PerfUtils::getTickTime();  
6.     jint res = asl::CastFuncPtr(my_JNI_OnLoad, ctx->org_func)(vm, res  
erved);  
7.     int duration = (int)(PerfUtils::getTickTime() - start);  
8.  
9.     LibLoaderMonitorImpl *monitor = (LibLoaderMonitorImpl*)LibLoaderMoni  
tor::getInstance();  
10.    monitor->addOnloadInfo(ctx->user_data, duration);  
11.    return res;  
12. }  
13. #endif
```

如上代码所示，linker 的 `dlopen` 完成加载，然后调用 `dlsym` 来调用目标 so 的 `JNI_OnLoad`，完成 JNI 涉及的初始化操作。

加载 so 需要注意并行出现 `loadLibrary0` 锁的问题，这样会让多线程发生等锁现象。可以减少并发加载，但不能简单把整个加载过程放到串行任务里，这样耗时可能会更长，并且没法充分利用资源。比较好的做法是，将耗时少的串行起来同时并行耗时长的 so 加载。

至此完成了 so 的初始化和链接的监控。

说完 Android，那么 iOS 的加载是怎样的，如何优化呢？我接着跟你说。

APP 加载

_dyld_start 之前做了什么，dyld_start 是谁调用的，通过查看 xnu 的源码可以理出，当 APP 点击后会通过 __mac_execve 函数 fork 进程，加载解析 Mach-O 文件，调用 exec_activate_image() 开始激活 image 的过程。先根据 image 类型来选择 imgact，开始 load_machfile，这个过程会先解析 Mach-O，解析后依据其中的 LoadCommand 启动 dyld。最后使用 activate_exec_state() 处理结构信息，thread_setentrypoint() 设置 entry_point APP 的入口点。

_dyld_start 之后要少些动态库，因为链接耗时；少些 +load、C 的 constructor 函数和 C++ 静态对象，因为这些会在启动阶段执行，多了就会影响启动时间。

因此，没有用的代码就需要定期清理和线上监控。通过元类中 flag 的方式进行监控然后定期清理。

iOS 主线程方法调用时长检测

+load 方法时间统计，使用运行时 swizzling 的方式，将统计代码放到链接顺序的最前面即可。静态初始化函数在 DATA 的 mod_init_func 区，先把里面原始函数地址保存，前后加上自定义函数记录时间。

在 Linux 上有 strace 工具，还有库跟踪工具 ltrace，OSX 有包装了 dtrace 的 instruments 和 dtruss 工具，不过在某些场景需求下不好用。objc_msgSend 实际上会通过在类对象中查找选择器到函数的映射来重定向执行到实现函数。一旦它找到了目标函数，它就会简单地跳转到那里，而不必重新调整参数寄存器。这就是为什么我把它称为路由机制，而不是消息传递。Objective-C 的一个方法被调用时，堆栈和寄存器是为 objc_msgSend 调用配置的，objc_msgSend 路由执行。objc_msgSend 会在类对象中查找函数表对应定向到的函数，找到目标函数就跳转，参数寄存器不会重新调整。

因此可以在这里 hook 住做统一处理。hook objc_msgSend 还可以获取启动方法列表，用于二进制重排方案中所需要的 APPOrderFiles，不过 APPOrderFiles 还可以通过 Clang SanitizerCoverage 获得，具体可以看宝藏男孩 Michael Eisel

这个这篇博客 [《Improving APP Performance with Order Files》](#) 的介绍。

`objc_msgSend` 可以通过 `fishhook` 指定到你定义的 `hook` 方法中，也可以使用创建跳转 `page` 的方式来 `hook`。做法是先用 `mmap` 分配一个跳转的 `page`，这个内存后面会用来执行原函数，使用特殊指令集将 CPU 重定向到内存的任意位置。创建一个内联汇编函数用来放置跳转的地址，利用 C 编译器自动复制跳转 `page` 的结构，指向 `hook` 的函数，之前把指令复制到跳转 `page` 中。ARM64 是一个 RISC 架构，需要根据指令种类检查分支指令。可以在 `_objc_msgSend` 里找到 `b` 指令的检查。相关代码如下：

```
1. ENTRY _objc_msgSend
2.     MESSENGER_START
3.
4.     cmp x0, #0          // nil check and tagged pointer check
5.     b.le    LNNilOrTagged // (MSB tagged pointer looks negative)
6.
7.     ldr x13, [x0]        // x13 = isa
8.
9.     and x9, x13, #ISA_MASK // x9 = class
```

检查通过就可以用这个指针读取偏移量，并修改指向跳转地址，跳转 page 完成，hook 函数就可以被调用了。

接下来看下 hook _objc_msgSend 的函数，这个我在以前博客 [《深入剖析 iOS 性能优化》](#) 写过，不过多赘述，只做点补充说明。从[这里的源码](#)可以看实现，其中的 attribute((naked)) 表示无参数准备和栈初始化，asm 表示其后面是汇编代码，volatile 是让后面的指令避免被编译优化到缓存寄存器中和改变指令顺序，volatile 使其修饰变量被访问时都会在共享内存里重新读取，变量值变化时也能写到共享内存中，这样不同线程看到的变量都是一个值。如果你发现不加 volatile 也没有问题，你可以把编译优化选项调到更优试试。stp 表示操作两个寄存器，中括号部分表示压栈存入 sp 偏移地址，! 符号表合并了压栈指令。

save() 的作用是把传递参数寄存器入栈保存，call(b, value) 用来跳到指定函数地址，call(blr, &before_objc_msgSend) 是调用原 _objc_msgSend 之前指定执行函数，call(blr, orig_objc_msgSend) 是调用 objc_msgSend 函数，call(blr, &after_objc_msgSend) 是调用原 _objc_msgSend 之后指定执

行函数。`before_objc_msgSend` 和 `after_objc_msgSend` 分别记录时间，差值就是方法调用执行的时长。

调用之间通过 `save()` 保存参数，通过 `load()` 来读取参数。

`call` 的第一个参数是 `blr`，`blr` 是指跳转到寄存器地址后会返回，由于 `blr` 会改变 `lr` 寄存器 X30 的值，影响 `ret` 跳到原方法调用方地址，崩溃堆栈找方法调研栈也依赖 `lr` 在栈上记录的地址，所以需要在 `call()` 之前对 `lr` 进行保存，`call()` 都调用完后再进行恢复。跳转到 `hook` 函数，`hook` 函数可以执行我们自定义的事情，完成后恢复 CPU 状态。

进入主图后的优化

进入主图后，用户就可以点击按钮进入不同功能了，是否能够快速响应按钮点击操作也是启动体验感知很重要的事情。按钮点击的两个事件 `didTouchUp` 和 `didTouchDown` 之间也会有延时。

因此可以在 `didTouchDown` 时在主线程先 `async` 初始化下一个 VC，把初始化提前完成，这样做可以提高 50ms—100ms 的速度，甚至更多，具体收益依赖当前主线程繁忙情况和下一个页面 `viewDidLoad` 等初始化方法里的耗时，启动阶段主线程一定不会闲，即使点击后主线程阻塞，使用 `async` 也能保证下一个页面的初始化不会停。

线程调度和任务编排

整体思路

对于任务编排有 种打法，就是先把所有任务滞后，然后再看哪个是启动开始必须要加载的。效果立竿见影，很快就能看到最好的结果，后面就是反复斟酌，严格把关谁才是必要的启动任务了。

启动阶段的任务，先理出相关依赖关系，在框架中进行配置，有依赖的任务有序执行，无依赖独立任务可以在非密集任务执行期串行分组，组内并发执行。

这里需要注意的是 Android 的 SharedPreferences 文件加载导致的 ContextImpl 锁竞争，一种解法是合并文件，不过后期维护成本会高，另一种是使用串行任务加载。你可能会疑惑，我没怎么用锁，那是不是就不会有锁等待的问题了。其实不然，比如在 iOS 中，dispatch_once 里有 dispatch_atomic_barrier 方法，此方法就有锁的作用，因此锁其实存在各个 API 之下，如不用工具去做检查，有时还真不容易发现这些问题。

有 IO 操作的任务除了锁等待问题，还有效率方面也需要特别注意，比如 iOS 的 Foundation 库使用的是 NSData

`writeToFile:atomically:` 方法，此方法会调用系统提供的 `fsync` 函数将文件描述符 `fd` 里修改的数据强写到磁盘里，`fsync` 相比较与 `fcntl` 效率高但写入物理磁盘会有等待，可能会在系统异常时出现写入顺序错乱的情况。系统提供的 `write()` 和 `mmap()` 函数都会用到内核页缓存，是否写入磁盘不由调用返回是否成功决定，另外 c 的标准库的读写 API `read` 和 `fwrite` 还会在系统内核页缓存同步对应由保存了缓冲区基地址的 `FILE` 结构体的内部缓冲区。因此启动阶段 IO 操作方法需要综合做效率、准确和重要性三方面因素的权衡考虑，再进行有 IO 操作的任务编排。

针对初始化耗时的库，比如埋点库，可以延后初始化，先将所需要的数据存储到内存中，待到埋点库初始化时再进行记录。对一些主图上业务网络可以延后请求，比如闪屏、消息盒子、主图天气、限行控件数据请求、开放图层数据、Wi-Fi 信息上报请求等。

多线程共享数据的问题

并发任务编排缺少一个统一的异步编程模型，并发通信共享数据方式的手段，比如代理和通知会让处理到处飞，闭包这种匿名函数排查问题不方便，而且回调中套回调前期设计后期维护和理解很困难，调试、性能测试也乱。这些通过回调来处理异步，不光

复杂难控，还有静态条件、依赖关系、执行顺序这样的额外复杂度，为了解决这些额外复杂度，还需要使用更多的复杂机制来保证线程安全，比如使用低效的 mutex、超高复杂度的读写锁、双重检查锁定、底层原子操作或信号量的方式来保护数据，需要保证数据是正确锁住的，不然会有内存问题，锁粒度要定还要注意避免死锁。

并发线程通信一般都会使用 libdispatch (GCD) 这样的共享数据方式来处理，也就异步再回调的方式。libdispatch 的 async 策略是把任务的 block 放到队列链表，使用时会在底层的线程池里找可用线程，有就直接用，没有就新建一个线程源码，监控线程池，队列调度，使用这样的策略来减少线程创建。当并发任务多时，比如启动期间，即使线程没爆，但 CPU 在各个线程切换处理任务时也是会有时间开销的，每次切换线程，CPU 都需要执行调度程序增加调度成本和增加 CPU 使用率，并且还容易出现多线程竞争问题。单次线程切换看起来不长，但整个启动，切换频率高的话，整体时间就会增大。

多线程的问题以及处理方式，带来了开发和排查问题的复杂性，以及出现问题机率的提高，资源和功能云化也有类似的问题，云化和本地的耦合依赖、云化之间的关系处理、版本兼容问题会带来更复杂的开发以及测试挑战，还有问题排查

的复杂度。这些都需要去做权衡，对基础建设方案提出了更高的要求，对容错回滚的响应速度也有更高的要求。

这里[有个 book](#) 专门来说并行编程难的，并告诉你该怎么做。

这里[有篇文章](#) 列出了苹果公司 libdispatch 的维护者 Pierre Habouzit 关于 libdispatch 的讨论邮件。

说了一堆共享数据方式的问题，没有体感，下面我说个最近碰到的多线程问题，你也看看排查有多费劲。

一个具体多线程问题排查思路

问题是工程引入一个系统库，暂叫 A 库，出现的问题现象是 CoreMotion 不回调，网络请求无法执行，除了全局并发队列会 pending block 外主线程和其它队列工作正常。

第一阶段，排查思路看是否跟我们工程相关，首先看是不是各个系统都有此问题，发现 iOS14 和 iOS13 都有问题。然后把 A 库放到一个纯净 Demo 工程中，发现没有出问题了。基于上面两种情况，推测只有将 A 库引入我们工程才会出现问题。在纯净 Demo 工程中，A 库使用时 CPU 会占用 60%–80%，集成到我们工程后涨到 100%，所以下个阶段排查方向就是性能。

第二阶段的打法是看是否是由性能引起的问题。先在纯净工程中创建大量线程，直到线程打满，然后进行大量浮点运算使 CPU 到 100%，但是没法复现，任务通过 libdispatch 到全局并发队列能正常工作。

怎么在 Demo 里看出线程已爆满了呢？

libdispatch 可以使用线程数是有上限的，在 libdispatch 的源码里可以看到 libdispatch 的队列初始化时使用 pthread 线程池相关代码：

```
1. #if DISPATCH_USE_PTHREAD_POOL  
2. static inline void  
3. _dispatch_root_queue_init_pthread_pool(dispatch_queue_global_t dq,  
4.                                         int pool_size, dispatch_priority_t pri)  
5. {  
6.     dispatch_pthread_root_queue_context_t pqc = dq->do_ctxt;  
7.     int thread_pool_size = DISPATCH_WORKQ_MAX_PTHREAD_COUNT;  
  
8.     if (!(pri & DISPATCH_PRIORITY_FLAG_OVERCOMMIT)) {  
9.         thread_pool_size = (int32_t)dispatch_hw_config(active_cpus);  
10.    }  
}
```

```
11.     if (pool_size && pool_size < thread_pool_size) thread_pool_size =  
          pool_size;  
12.     ... // 省略不相关代码  
13. }
```

如上面代码所示，`dispatch_hw_config` 会用 `dispatch_source` 来监控逻辑 CPU、物理 CPU、激活 CPU 的情况计算出线程池最大线程数量。

如果当前状态是 `DISPATCH_PRIORITY_FLAG_OVERCOMMIT`，也就是会出现 `overcommit` 队列时，线程池最大线程数就按照 `DISPATCH_WORKQ_MAX_PTHREAD_COUNT` 这个宏定义的数量来，这个宏对应的值是255。因此通过查看是否出现 `overcommit` 队列可以看出线程池是否已满。

什么时候 `libdispatch` 会创建一个新线程？

当 `libdispatch` 要执行队列里 `block` 时会去检查是否有可用的线程，发现有可用线程时，在可用线程去执行 `block`，如果没有，通过 `pthread_create` 新建一个线程，在上面执行，函数关键代码如下：

```
1. static void
2. _dispatch_root_queue_poke_slow(dispatch_queue_global_t dq, int n, int flo
   or)
3. {
4.     ...
5.     // 如果状态是 overcommit，那么就继续添加到 pending
6.     bool overcommit = dq->dq_priority & DISPATCH_PRIORITY_FLAG_O
   VERCOMMIT;
7.     if (overcommit) {
8.         os_atomic_add2o(dq, dgq_pending, remaining, relaxed);
9.     } else {
10.         if (!os_atomic_cmpxchg2o(dq, dgq_pending, 0, remaining, rela
   xed)) {
11.             _dispatch_root_queue_debug("worker thread request still pen
   ding for "
12.             "global queue: %p", dq);
13.         return;
14.     }
15. }
16. ...
17. t_count = os_atomic_load2o(dq, dgq_thread_pool_size, ordered);
18. do {
```

```
19.         can_request = t_count < floor ? 0 : t_count - floor;
20.         // 是否有可用
21.         if (remaining > can_request) {
22.             _dispatch_root_queue_debug("pthread pool reducing request f
rom %d to %d",
23.                                         remaining, can_request);
24.             os_atomic_sub2o(dq, dgq_pending, remaining - can_reques
t, relaxed);
25.             remaining = can_request;
26.         }
27.         // 线程满
28.         if (remaining == 0) {
29.             _dispatch_root_queue_debug("pthread pool is full for root qu
eue: "
30.                                         "%p", dq);
31.             return;
32.         }
33.     } while (!os_atomic_cmpxchgvw2o(dq, dgq_thread_pool_size, t_cou
nt,
34.                                         t_count - remaining, &t_count, acquire));
35.
36.     ...

```

```
37.     do {
38.         _dispatch_retain(dq); // 在 _dispatch_worker_thread 里取任务
    并执行
39.         while ((r = pthread_create(pthr, attr, _dispatch_worker_thread,
    dq))) {
40.             if (r != EAGAIN) {
41.                 (void)dispatch_assume_zero(r);
42.             }
43.             _dispatch_temporary_resource_shortage();
44.         }
45.     } while (--remaining);
46.     ...
47. }
```

如上面代码所示，can_request 表示可用线程数，通过当前最大可用线程数减去已用线程数获得，赋给 remaining 后，用来判断线程是否满和控制线程创建。`dispatch_worker_thread` 会取任务并执行。

当 libdispatch 使用的线程池中线程过多，并且有 pending 标记，当等待超时；也就是

libdispatch 里 DISPATCH_CONTENTION_USLEEP_MAX 宏定义的时间后，也会触发创建一个新的待处理线程。

libdispatch 对应函数关键代码如下：

```
1. static bool  
2. __DISPATCH_ROOT_QUEUE_CONTENDED_WAIT__(dispatch_queue_global_t dq,  
3.                                              int (*predicate)(dispatch_queue_global_t dq))  
4. {  
5.     ...  
6.     bool pending = false;  
7.     ...  
8.     do {  
9.         ...  
10.        if (!pending) {  
11.            // 添加 pending 标记  
12.            (void)os_atomic_inc2o(dq, dgq_pending, relaxed);  
13.            pending = true;  
14.        }  
15.        _dispatch_contention_usleep(sleep_time);  
16.        ...  
17.        sleep_time *= 2;
```

```
18.     } while (sleep_time < DISPATCH_CONTENTION_USLEEP_MAX);  
19.     ...  
20.     if (pending) {  
21.         (void)os_atomic_dec2o(dq, dgq_pending, relaxed);  
22.     }  
23.     if (status == DISPATCH_ROOT_QUEUE_DRAIN_WAIT) {  
24.         _dispatch_root_queue_poke(dq, 1, 0); // 创建新线程  
25.     }  
26.     return status == DISPATCH_ROOT_QUEUE_DRAIN_READY;  
27. }
```

如上所示，在创建新的待处理线程后，会退出当前线程，负载没了就会去用新建的线程。

接下来使用 Instruments 进行分析 Trace 文件，发现启动阶段立刻开始使用 A 库的话，CPU 会突然上升，如果使用 A 库稍晚些，CPU 使用率就是稳定正常的。

这说明在第一个阶段性能相关结论只是偶现情况才会出现，出问题时，并没有出现系统资源紧张的情况，可以得出并不是性能问题的结论。那么下一个阶段只能从A库的使用和排查我们工程其它功能的问题。

第三个阶段的思路是使用功能二分排查法，先排出 A 库使用问题，做法是在使用最简单的 A 库初始化一个页面在首屏也会复现问题。

我们的功能主要分为渲染、引擎、网络库、基础功能、业务几个部分。将渲染、引擎、网络库拉出来建个 Demo，发现这个 Demo 不会出现问题。那么有问题的就可能在基础功能、业务上。

先去掉的功能模块有 CoreMotion、网络、日志模块、定时任务（埋点上传），依然复现。接下来去掉队列里的 libdispatch 任务，队列里的任务主要是由 Operation 和 libdispatch 两种方式放入。其中 Operation 最后是使用 libdispatch 将任务 block 放入队列，期间会做优先级和并发数的判断。对于 libdispatch 可以 Hook 住可以把任务 block 放到队列的 libdispatch 方法，有 dispatch_async 、 dispatch_after 、 dispatch_barrier_async 、 dispatch_APPIy 这些方法。任务直接返回，还是有问题。

推测验证基础能力和业务对出现问题队列有影响，instruments 只能分析线程，无法分析队列，因此需要写工具分析队列情况。

接下来进入第四个阶段。

先 hook 时截获任务 block 使用的 libdispatch 方法、执行队列名、优先级、做唯一标识的入队时间、当前队列的任务数、还有执行堆栈的信息。通过截获的内容按照时间线看，当出现全局并发队列 pending block 数量堆积时，新的使用 libdispatch 加入的部分任务可以得到执行，也有没执行的，都执行了也会有问题。

然后去掉 Operation 的任务：通过日志还能发现 Operation 调用 libdispatch 的任务直接 hook libdispatch 的方法是获取不到的，可能是 Operation 调用方法有变化。另外在无法执行任务的线程上新建的 libdispatch 任务也无法执行，无法执行的 Operation 任务达到所设置的 maxConcurrentOperationCount，对应的 OperationQueue 就会在 Operation 的队列里 pending。

由此可以推断出，在全局并发队列 pending 的 block 包含了直接使用 libdispatch 的和 Operation 的任务，pending 的任务。因此还需要 hook 住 Operation，过滤掉所有添加到 Operation Queue 的任务，但结果还是复现问题。

此时很崩溃，本来做好了一个一个下掉功能的准备（成本高），这时，有同学发现前阶段两个不对的结论。

这个阶段定为第五阶段。

第一个不对的结论是经 QA 同学长时间多轮测试，只在14.2及以上系统版本有问题，由于只有这个版本才开始有此问题，推断可能是系统 bug；第二个不对的是只有渲染、引擎、网络库的 Demo 再次检查，可复现问题，因此可以针对这个 Demo 进行进一步二分排查。

于是，咱们针对两个先前错误结论，再次出发，同步进行验证。对 Demo 排除了网络库依然复现，后排除引擎还是复现，同时使用了自己的示例工程在iOS14.2上复现了问题，和第一阶段纯净Demo的区别是往全局并发队列里方式，官方 Demo 是 Operation，我们的是 libdispatch。

因此得出结论是苹果系统升级问题，原因可能在 OperationQueue，问题重现后，不再运行其中的 operation。14.3beta 版还没有解决。

那么看下 Operation 实现，分析下系统 bug 原因。

APPPortableFoundation 里有Operation的开源实现 NSOperation.m，相比较 GNUstep 和 Cocotron 更完善，可以看到 Operation 如何在 _schedulerRun 函数里通过 libdispatch 的 async 方法将 operation 的任务放到队列执行。

swift源码里的fundation也有实现 Operation，我们看看 _schedule 函数的关键代码：

```
1. internal func _schedule() {  
2.     ...  
3.     // 按优先级顺序执行  
4.     for prio in Operation.QueuePriority.priorities {  
5.         ...  
6.         while let operation = op?.takeUnretainedValue() {  
7.             ...  
8.             let next = operation.__nextPriorityOperation  
9.             ...  
10.            if Operation.__NSOperationState.enqueued == operation._state && operation._fetchCachedIsReady(&retest) {  
11.                if let previous = prev?.takeUnretainedValue() {  
12.                    previous.__nextPriorityOperation = next  
13.                } else {  
14.                    _setFirstPriorityOperation(prio, next)  
15.                }  
16.            ...  
17.            if __mainQ {  
18.                queue = DispatchQueue.main  
19.            } else {
```

```
20.           queue = __dispatch_queue ?? __synthesizeBacking
21.           Queue()
22.
23.           if let schedule = operation.__schedule {
24.               if operation is _BarrierOperation {
25.                   queue.async(flags: .barrier, execute: {
26.                       schedule.perform()
27.                   })
28.               } else {
29.                   queue.async(execute: schedule)
30.               }
31.           }
32.
33.           op = next
34.       } else {
35.           ... // 添加
36.       }
37.   }
38. }
39. ...
40. }
```

上述代码可见，可以看到 `_schedule` 函数根据 `Operation.QueuePriority.priorities` 优先级数组顺序，从最高 `barrier` 开始到 `veryHigh`、`high`、`normal`、`low` 到最低的 `veryLow`，根据 `operation` 属性设置决定 libdispatch 的 `queue` 是什么类型的，最后通过 `async` 函数分配到对应的队列上执行。

查看 `operation` 代码更新情况，最新 `operation` 提交修复了一个问题，commit [在这里](#)，根据修复问题的描述来看，和 A 库引入导致队列不可添加 `OperationQueue` 的情况非常类似。修复的地方可以看下图：



如图所示，在先前 `_schedule` 函数里使用 `nextOperation` 而不用 `nextPriorityOperation` 会导致主操作列表里的不同优先级操作列表交叉连接，可能会在执行后面操作时被挂起，而 A

库里的 OperationQueue 都是高优的，如果有其它优先级的 OperationQueue 加进来就会出现挂起的问题。

从提交记录看，19 年 6 月 12 日的那次提交变更了很多代码逻辑，描述上看是为了更接近 objc 的实现，changePriority 函数就是那个时候加进去的。提交的 commit 如下图所示：

The screenshot shows a GitHub commit for a Swift file named 'Operation.swift' located in the 'Foundation' directory. The commit message is 'Refactor Operation, and OperationQueue to be closer to the objc implementation with new barrier and progress APIs (#2331)'. The commit date is highlighted with a red box as '2019年6月12日 GMT+8 上午4:00:03'. The code editor shows several lines of Swift code, with line 991 highlighted by a blue box. The specific line of code is: 'previous._nextOperation = next'. This line is part of a loop that updates the '_nextOperation' pointer for each operation in the queue.

```
972 +     internal func _schedule() {
973 +         var retestOps = [Operation]()
974 +         _lock()
975 +         var slotsAvail = _actualMaxNumOps - _numExecOps
976 +         for prio in Operation.QueuePriority.priorities {
977 +             if 0 >= slotsAvail || _suspended {
978 +                 break
979 +             }
980 +             var op = _firstPriorityOperation(prio)
981 +             var prev: Unmanaged<Operation>?
982 +             while let operation = op?.takeUnretainedValue() {
983 +                 if 0 >= slotsAvail || _suspended {
984 +                     break
985 +                 }
986 +                 let next = operation._nextPriorityOperation
987 +                 var retest = false
988 +                 // if the cached state is possibly not valid then the isReady value in
989 +                 // if Operation._NSOperationState.enqueued == operation._state && opera
990 +                 // if let previous = prev?.takeUnretainedValue() {
991 +                     previous._nextOperation = next
992 +                 } else {
993 +                     _setFirstPriorityOperation(prio, next)
994 +                 }
995 +                 if next == nil {
996 +                     _setLastPriorityOperation(prio, prev)
```

怀疑（只是怀疑，苹果官方并没有说）可能是在 iOS14 引入 swift 版的 Operation，因此这个 Operation 针对 objc 调用做了适配。之所以 14.2 之前 Operation 重构后的 bug 没有引起问题，可能是当时 A 库的 Queue 优先级还没调高，14.2 版本 A 库的 Queue 优先级开始调高了，所以出现了优先级交叉挂起的情况。

从这次排查可以发现，目前对于并发的监测还是非常复杂的。那么并发问题在 iOS 的将来会得到解决吗？

多线程并行计算模型

既然共享数据方式问题多，那还有其它选择吗？

实际上在服务端大量使用着 Actor 这样的并行计算模型，在并行世界里，一切都是 actor，actor 就像一个容器，会有自己的状态、行为、串行队列的消息邮箱。actor 之间使用消息来通信，会把消息发到接受消息 actor 的消息邮箱里，消息盒子可并行接受消息，消息的处理是依次进行，当前处理完才处理下一个，消息邮箱这套机制就好像 actor 们的大管家，让 actor 之间的沟通井然有序。

[Swift 并发路线图](#)也预示着 Swift 要加入 actor，Chris Lattner 也希望 Swift 能够在多核机器，还有大型服务集群能够得到方便的使用，分布式硬件的发展趋势必定是多核，去共享内存的硬件的，因为共享内存的编程不光复杂而且原子性访问比非原子性要慢近百倍。

提案中设计到 actor 的设计是把 actor 设计成一种特殊类，让这个类有引用语义，能形成 map，可以 weak 或 unowned 引用。actor 类中包含一些只有 actor 才有的方法，这些方法提供 actor 编程模型所需安全性。

但 actor 类不能继承自非 actor 类，因为这样 actor 状态可能有机会以不安全的方式泄露。actor 和它的函数和属性之间是静态关系，这样可以通过编译方式避免数据竞争，对数据隔离，如果不是安全访问 actor 属性的上下文，编译器可以处理切换到那个上下文中。

对于 actor 隔离会借鉴强制执行对内存的独占访问提案的思想，比如局部变量、inout 参数、结构体属性编译器可以分析变量的所有访问，有冲突就可以报错，类属性和全局变量要在运行时可以跟踪在进行的访问，有冲突报错。而全局内存还是没法避免数据竞争，这个需要增加一个全局 actor 保护。

按 actor 模型对任务之间通讯重新调整，不用回调代理等手段，将发送消息放到消息邮箱里进行类似 RxSwift 那样 next 的方式一个一个串行传递。说到 RxSwift，那 RxSwift 和 Combine 这样的框架能替代 actor 吗？

对这些响应式框架来说解决线程通信只是其中很小的一部分，其还是会面临闭包、调试和维护复杂的问题，而且还要使用响应式编程范式，显然还是有些重了，除非你已经习惯了响应式编程。

任务都按 actor 模型方式来写，还能够做到功能之间的解耦，如果是服务器应用，actor 可以布到不同的进程甚至是不同机器上。

actor 中消息邮件在同一时间只能处理一个消息，这样等待返回一个值的方式，需要暂停，内部有返回再继续执行，这要怎么实现呢？

答案是使用 Coroutine。

在 Swift 并发路线提案里还提到了基于 coroutine 的 `async/await` 语法，这种语法风格已经被广泛采纳，比如 Python、Dart、JavaScript 都有实现，这样能够写出简洁好维护的并发代码。

上述只是提案，最快也需要两个版本的等待，那么语言上的支持还没有来，怎么能提前享用 coroutine 呢？

处理暂停恢复操作，可以使用 context 处理函数 `setjmp` 和 `longjmp`，但 `setjmp` 和 `longjmp` 较难实现临时切换到不同的执行路径，然后恢复到停止执行的地方，所以服务器用一般都会使用 `ucontext` 来实现，gnu 的举的例子 GNU C Library：

[Complete Context Control](#)，这个例子在于创建 context 堆栈，swapcontext 来保存 context，这样可以在其它地方能执行回到原来的地方。创建 context 堆栈代码如下：

```
1. uc[1].uc_link = &uc[0];  
2. uc[1].uc_stack.ss_sp = st1;  
3. uc[1].uc_stack.ss_size = sizeof st1;  
4. makecontext (&uc[1], (void (*) (void)) f, 1, 1);
```

上面代码中 uc_link 表示的是主 context。保存 context 的代码如下：

```
swapcontext (&uc[n], &uc[3 - n]);
```

但是在 Xcode 里一试，出现错误提示如下：

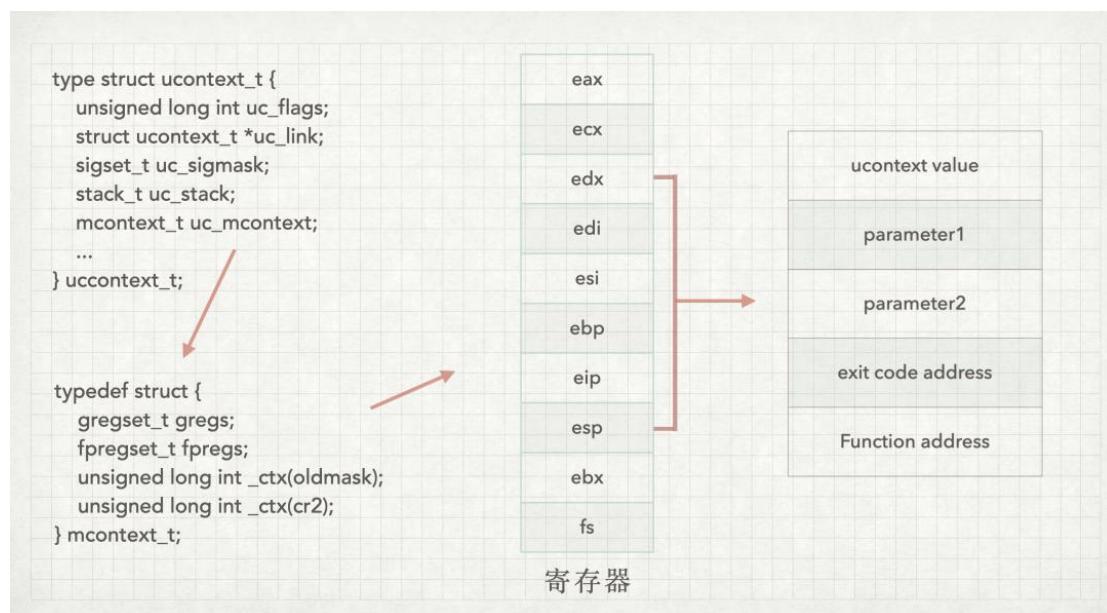
```
implicit declaration of function 'swapcontext' is invalid in c99
```

原来最新的 POSIX 标准已经没有这个函数了，IEEE Std 1003.1-2001 / Cor 2-2004，应用了项目 XBD/TC2/D6/28，标注 getcontext()、makecontext()、setcontext() 和 swapcontext() 函数过时了。在 POSIX 2004 第 743 页说明了原因，大概意思就是建议使用 pthread 这种系统编程上，后来的 Rust 和 Swift coroutine 的提案里都是使用的系统编程来实现 coroutine，长期看系统编程实现 coroutine 肯定是趋势。那么

在 swift 升级之前还有办法在 iOS 用 ucontext 这种轻量级的 coroutine 吗？

其实也是有的，可以考虑临时过渡一下。具体可以看看 ucontext 的汇编实现，重新在自己工程里实现出来就可以了。getcontext、setcontext、makecontext、swapcontext 在 Linux 系统代码里能看到。ucontext_t 结构体里的 uc_stack 会记录 context 使用的栈。getcontext() 是把各个寄存器保存到内存结构体里，setcontext() 是把来自 makecontext() 和 getcontext() 的各寄存器恢复到当前 context 的寄存器里。switchcontext() 合并了 getcontext() 和 setcontext()。

ucontext_t 的结构体设计如下：



如上图所示，ucontext_t 还包含了一个更高层次的 context 封装 uc_mcontext，uc_mcontext 会保存调用线程的寄存器。上图中 eax 是函数入参地址，寄存器值入栈操作代码如下：

```
1. movl    $0, oEAX(%eax)  
2. movl    %ecx, oECX(%eax)  
3. movl    %edx, oEDX(%eax)  
4. movl    %edi, oEDI(%eax)  
5. movl    %esi, oESI(%eax)  
6. movl    %ebp, oEBP(%eax)
```

以上代码中 oECX、oEDX 等表示相应寄存器在内存结构体里的位置。esp 指向返回地址值，由 eip 字段记录，代码如下：

```
1. movl    (%esp), %ecx  
2. movl    %ecx, oEIP(%eax)
```

edx 是 getcontext() 的栈寄存器会记录 ucontext_t.uc_stack.ss_sp 栈顶的值，oSS_SIZE 是栈大小，通过指令 addl 可以找到栈底。

makecontext() 会根据 ecx 里的参数去设置栈，setcontext() 是 getcontext 的逆操作，设置当前 context，栈顶在 esp 寄存器。

轻量级的 `coroutine` 实现了，下面咱们可以通过 Swift `async/await` 提案(已加了编号 0296, 表示核心团队已经认可，上线可期) 看下系统编程的 `coroutine` 是怎么实现的。Swift `async/await` 提案中的思路是让开发者编写异步操作逻辑，编译器用来转换和生成所需的隐式操作闭包。可以看作是个语法糖，并像其它实现那样会改变完成处理程序被调用的队列。工作原理类似 `try`，也不需要捕获 `self` 的转义闭包。挂起会中断原子性，比如一个串行队列中任务要挂起，让其它任务在一个串行队列中交错运行，因此异步函数最好是不阻塞线程。将异步函数当作一般函数调用，这样的调用会暂时离开线程，等待当前线程任务完成再从它离开的地方恢复执行这个函数，并保证是在先前的 `actor` 里执行完成。

启动性能分析工具

iOS 官方工具

Instruments 中 Time Profiles 中的 Profile 可以方便的分析模块中每个方法的耗时。Time Profiles 中的 Samples 分析将更加准确的显示出 APP 启动后每一个 CPU 核心在一个时间片内所执行的代码。如果在模块开发中有以下的需求，可以考虑使用 Samples 分析：

- 希望更精确的分析某个方法具体执行代码的耗时

- 想知道一个方法到另一个方法的耗时情况（跨方法耗时分析）

MetricKit 2.0 开始加强了诊断特性，通过收集调用栈信息能够方便我们来进行问题的诊断，通过 didReceive 回调 MXMetricPayload 性能数据，可包含 MXSignpostMetric 自定义采集数据，甚至是捕获不到的崩溃信号的系统强杀崩溃信息传到自己服务器进行分析和报警。

如何在 iOS 真机和模拟器上实现自动化性能分析

苹果有个 usbmux 协议会给自己 macOS 程序和设备进行通信，场景有备份 iPhone 还有真机调试。macOS 对应的是 /System/Library/PrivateFrameworks/MobileDevice.framework/Versions/A/Resources/ 下的 usbmuxd 程序，usbmuxd 是 IPC socket 和 TCP socket 用来进行进程间通信，[这里有他的一个开源实现](#)。对于在手机端是 lockdown 来起服务。因此利用 usbmuxd 的协议，就可以自建和设备通信的应用比如 lookin，实现方式可以参考这个 [demo](#)。使用 usbmux 协议的 libimobiledevice（相当于 Android 的 adb）提供了更多能力，可以获取设备的信息、搭载 ifuse 访问设备文件系统（没越

狱可访问照片媒体、沙盒、日志）、与调试服务器连接远程调试。无侵入的库还有 gamebench 也用到了 libimobiledevice。

instruments 可以导出 .trace 文件，以前只能用 instruments 打开，Xcode12 提供了 xctrace 命令行工具可以导出可分析的数据。Xcode12 之前的时候是能使用 TraceUtility 这个库，TraceUtility 的做法是链上 Xcode 里 instruments 用的那些库，比如 DVTFoundation 和 InstrumentsKit 等，调用对应的方法去获取 .trace 文件。使用 libimobiledevice 能构造操作 instruments 的应用，将 instruments 的能力自动化。

perfdog 就是使用了 libimobiledevice 调用了 instruments 的接口，[实现代码](#)来实现 instruments 的一些功能，并进行了扩展定制，无侵入的构建本地性能监控并集成到自动测试中出数据，减少人工成本。无侵入的另一个好处就是可以方便用同一套标准看到其他 APP 的表现情况。

要到具体场景去跑 case 还需要流程自动化。APPIum 使用的是 Facebook 开发的一套基于 W3C 标准交互协议 WebDriver 的库 WebDriverAgent，Python 版可以看[这个](#)，不过后来 Facebook 开发了新的一套命令行工具 idb(iOS Development Bridge)，归档了 WebDriverAgent。

idb 可以对 iOS 模拟器和设备跑自动化测试，idb 主要有两个基于 macOS 系统库 CoreSimulator.framework、MobileDevice.framework，包装的 FBSimulatorControl 和 FBDeviceControl 库。

FBSimulatorControl 包含了 iOS 模拟器的所有功能，Xcode 和 simctl 都是用的 CoreSimulator，自动化中输入事件是逆向了 iOS 模拟器 Indigo 服务的协议，Indigo 是模拟器通过 mach IPC 通道 mach_msg_send 接受触摸等输入事件的协议。破解后就可以模拟输入事件了。

MobileDevice.framework 也是 macOS 的私有库，macOS 上的 Finder、Xcode、Photos 这些会使用 iOS 设备的应用都是用了 MobileDevice，文件读写用的是包装了 AMDServiceConnection 协议的 AFC 文件操作 API，idb 的 instruments 相关功能是在 [这里](#) 实现了 DTXConnectionServices 服务协议。
libmobiledevice 可以看作是重新实现了 MobileDevice.framework。pymobiledevice、MobileDevice、C 编写的 SDMMobileDevice，还有 Objective-C 编写的 MobileDeviceAccess，这些库也是用的 MobileDevice.framework。

Android Profiler

Android Profiler 是 Android 中常用的耗时分析工具，以各种图表的形式展示函数执行时间，帮助开发者分析耗时问题。

启动优化着实是牵一发动全身的事情，手段既琐碎又复杂。如何能够将监控体系建设起来，并融入到整个研发到上线流程中，是个庞大的工程。下面给你介绍下我们是如何做的吧。

管控流程体系保障平台建设

APM 自动化管控和流程体系保障平台，目标是通过稳定环境更自动化的测试，采集到的性能数据能够通过分析检测，发现问题能够更低成本定位分发告警，同时大盘能够展示趋势和详情。

开发过程会 daily 出迭代报告，开发完成后，会有集成卡口，提前卡住迭代性能问题。

集成后，在集成构建平台能够构建正式包和线下性能包，进行线下测试和线上性能数据采集，线下支持录制回放、Monkey 等自动化测试手段，测试期间会有生成版本报告，发布上线前也会有发布卡口，及时处理版本问题。

发布后，通过云控进行指标配置、阈值配置还有采集比例等。性能数据上传服务经异常检测发现问题会触发报警，自动在 Bug 平台创建工单进行跟踪，以便及时修复问题减少用户体验损失。服务还会做统计、分级、基线对比、版本关联以及过滤等数据分析操作，这些分析后的性能数据最终会通过版本、迭代趋势等统计报表方式在大盘上展示，还能展示详情，包括对比展示、问题详情、场景分类、条件查询等。

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德地图驾车导航内存优化原理与实战

作者：禹方

背景

根据 Apple 官方 WWDC 的回答，减少内存可以让用户体验到更快的启动速度，不会因为内存过大而导致 Crash，可以让 APP 存活的更久。

对于高德地图来说，根据线上数据的分析，内存过高会导致导航过程中系统强杀 OOM。尤其区别于其他 APP 的地方是，一般 APP 只需要关注前台内存过高的系统强杀 FOOM，高德地图有不少用户使用后台导航，所以也需要关注后台的内存过高导致的系统强杀 BOOM，且后台强杀较前台强杀更为严重。为了提升用户体验，内存治理迫在眉睫。

原理剖析

OOM

OOM 是 Out of Memory 的缩写。在 iOS APP 中如果内存超了，系统会把 APP 直接杀死，一种另类的 Crash，

且无法捕获。发现 OOM 时，我们可以从设备->隐私->分析与改进->分析数据 中找到以 JetsamEvent 开头的日志，日志里面记录了很多信息：手机设备信息、系统版本、内存大小、CPU 时间等。

Jetsam

Jetsam 是 iOS 系统的一种资源管理机制。不同于 MacOS、Linux、Windows 等，iOS 中没有内存交换空间，所以在设备整体内存紧张时，系统会将一些优先级不高或者占用内存过大的直接 Kill 掉。

通过 iOS 开源的 XNU 内核源码可以分析到：

- 每个进程在内核中都存在一个优先级列表，JetSam 在受到内存压力时会从优先级列表最低的进程开始尝试杀死，直到内存水位恢复到正常水位。
- Jetsam 是通过 `get_task_phys_footprint` 获取到 `phys_footprint` 的值，来决定要不要杀掉应用。

Jetsam 机制清理策略可以总结为以下几点：

- 单个 APP 物理内存占用超过上限会被清理，不同的设备内存水位线不一样。
- 整个设备物理内存占用受到压力时，优先清理后台应用，再清理前台应用。
- 优先清理内存占用高的应用，再内存占用低的应用。
- 相比系统应用，会优先清理用户应用。

Android 端为 Low Memory Killer：

- 根据 APP 的优先级和使用总内存的多少，系统会在设备内存吃紧情况下强杀应用。
- 内存吃紧的判断取决于系统 RSS(实际使用物理内存，包含共享库占用的全部内存)的大小。
- 关键参数有 3 个：

- oom_adj：在 Framework 层使用，代表进程的优先级，数值越高，优先级越低，越容易被杀死。
- oom_adj threshold：在 Framework 层使用，代表 oom_adj 的内存阈值。Android Kernel 会定时检测当前剩余内存是否低于这个阈值，若低于则杀死 $\text{oom_adj} \geq \text{该阈值对应的 oom_adj}$ 中，数值最大的进程，直到剩余内存恢复至高于该阈值的状态。
- oom_score_adj：在 Kernel 层使用，由 oom_adj 换算而来，是杀死进程时实际使用的参数。

数据分析

phys_footprint 获取 iOS 应用总的物理内存，具体可以参考官方说明 iOS Memory Deep Dive.

```
1. std::optional<size_t> memoryFootprint()  
2. {  
3.     task_vm_info_data_t vmlInfo;  
4.     mach_msg_type_number_t count = TASK_VM_INFO_COUNT;
```

```
5.     kern_return_t result = task_info(mach_task_self(), TASK_VM_INFO, (task_info_t) &vmlInfo, &count);  
6.     if (result != KERN_SUCCESS)  
7.         return std::nullopt;  
8.     return static_cast<size_t>(vmlInfo.phys_footprint);  
9. }
```

Instruments—VM Tracker 可以用来分析具体内存分类，比如 Malloc 部分是堆内存，WebKit Malloc 部分是 JavaScriptCore 占用的内存等。需要注意的是每个分类的内存值 = Dirty Size + Swapped。

通过 Instruments VM Tracker 抓取导航中内存分布进行对比分析。导航前台静置时，高德地图的总内存数值非常高，其中 IOKit、WebKit Malloc 和 Malloc 堆内存为内存占用大头。

在分析过程中可以使用的工具很多，各有优缺点，需要配合使用，相互弥补。我们在分析的过程中主要用到 Instruments VM Tracker、Allocations、Capture GPU Frame、MemGraph、dumpsys meminfo、Graphics API Debugger、Arm Mobile Studio、AJX 内存分析工具、自研 Malloc 分析工具等。

- IOKit 内存为地图渲染显存部分。
- WebKit Malloc 内存为 AJX JS 业务内存。
- Malloc 堆内存，我们通过 Hook Malloc 分配内存的 API，通过抓取堆栈分析具体内存消费者。

治理优化

根据上面的数据分析，很容易做出从大头开始抓起的思路。我们在治理过程中的大体思路：

- 分析数据：从内存大头开始，分析各内存归属业务，以便业务进一步分析优化。
- 内存治理：优化技术方案减少内存开销、高低端机功能分级和智能容灾（即内存告警时通过功能降级等策略释放内存）。

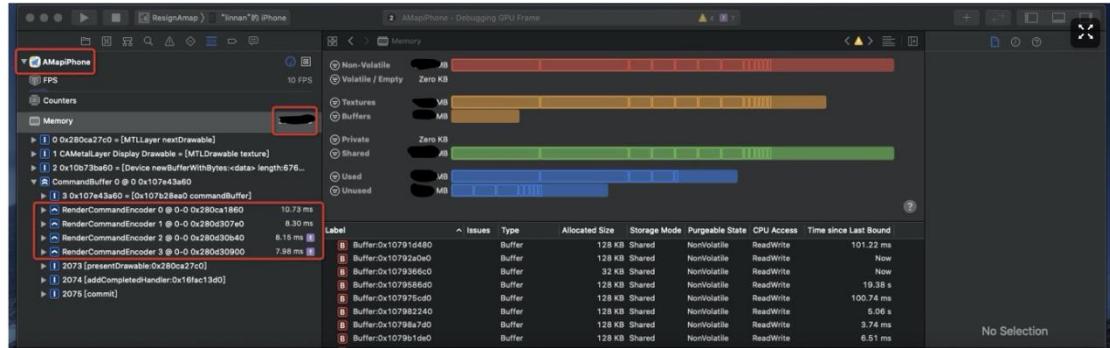
分而治之

据数据分析，高德地图三大内存消耗分别是地图渲染（Graphic 显存）、功能业务（JavaScriptCore）和通用业务（Malloc）。我们也主要从这三个方面入手优化。

地图 Graphic 显存优化

Xcode 自带 Debug 工具 Capture GPU Frame，可以分析出具体显存占用，显存主要分为纹理 Texture 部分和 Buffer 部分，通过详细的地址信息分析具体消耗。

Android 端类似分析显存工具可以用 Google 的 Graphics API Debugger。



根据分析，Texture 部分我们通过 FBO 绘制方式调整、矢量路口大图背景优化、图标跨页面释放、文字纹理优化、低端机关闭全屏抗锯齿等减少显存消耗。Buffer 部分通过开启低显存模式、关闭四叉树预加载、切后台释放缓存资源等。

Webkit Malloc 优化

高德地图使用的是自研的动态化方案，依赖于 iOS 系统提供的框架 JavaScriptCore，使用的业务内存消耗大

多会被系统归类到 WebKit Malloc，从系统工具 Instruments 上的 VM Tracker 可以看出。此处有两个思路，一个是业务自身优化内存消耗，第二个是动态化引擎和框架优化内存消耗。

业务自身优化，动态化方案的 IDE 提供内存分析工具可以清晰的输出具体业务内存消耗在什么地方，便于业务同学分析是否合理。

动态化引擎和框架优化，我们通过优化对系统库 JavaScriptCore 的使用方式，即多个 JSContextRef 上下文共享同一份 JSContextGroupRef 的方式。多个页面可以共享一份框架代码，从而减少内存开销。

Malloc 堆内存优化

iOS 端堆内存分配基本上使用的 libmalloc 库，其中包含以下几个内存操作接口：

```
1. // c 分配方法  
2. void *malloc(size_t __size) __result_use_check __alloc_size(1);  
3. void *calloc(size_t __count, size_t __size) __result_use_check __alloc_size(1, 2);
```

```
4. void      free(void *);  
5. void      *realloc(void *__ptr, size_t __size) __result_use_check __alloc_si  
ze(2);  
6. void      *malloc(size_t) __alloc_size(1);  
7.  
8. // block 分配方法  
9. // Create a heap based copy of a Block or simply add a reference to an  
existing one.  
10. // This must be paired with Block_release to recover memory, even wh  
en running  
11. // under Objective-C Garbage Collection.  
12. BLOCK_EXPORT void *_Block_copy(const void *aBlock)  
13.     __OSX_AVAILABLE_STARTING(__MAC_10_6, __IPHONE_3_2);
```

通过hook内存操作API记录下内存分配的堆栈、大小，即可分析内存使用情况。

同时源码中还存在一个全局钩子函数 malloc_logger，可输出 Malloc 过程中的日志，定义如下：

```
1. // We set malloc_logger to NULL to disable logging, if we encounter er  
rors
```

```
2. // during file writing  
3. typedef void(malloc_logger_t)(uint32_t type,  
4.           uintptr_t arg1,  
5.           uintptr_t arg2,  
6.           uintptr_t arg3,  
7.           uintptr_t result,  
8.           uint32_t num_hot_frames_to_skip);  
9. extern malloc_logger_t *malloc_logger;
```

iOS 堆内存分析方案，可通过 hook malloc 系列 API，也可以设置 malloc_logger 的函数实现，即可记录下堆内存使用情况。

此方案有几个难点问题，每秒钟内存分配的量级大、内存有分配有释放需要高效查询和堆栈反解聚合。为此我们设计了一套完整的 Malloc 堆内存分析方案，来满足快速定位堆内存归属，以便分发到各自业务 Owner 分析优化。

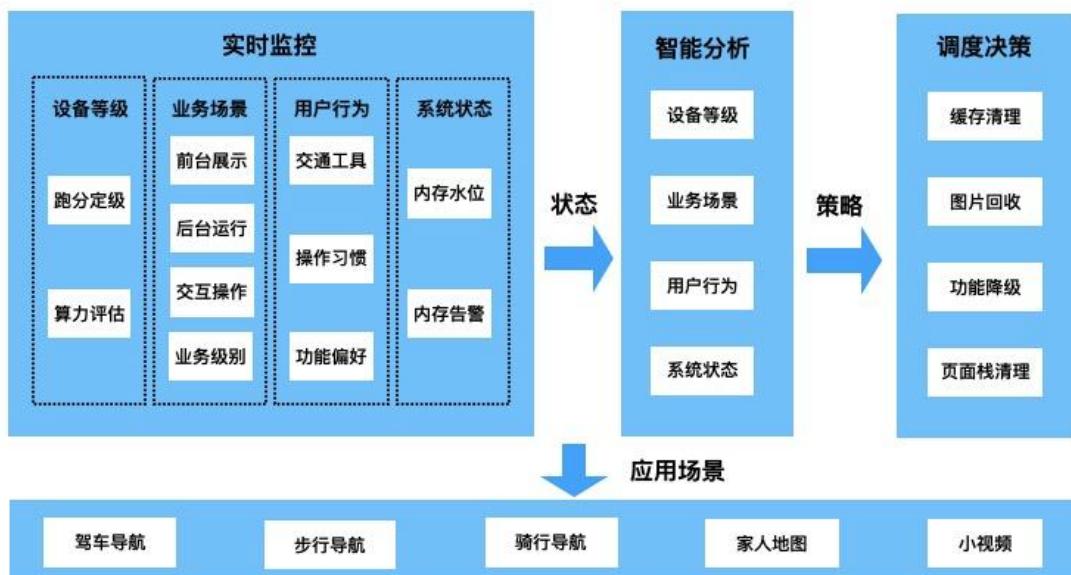
统一管理

随着业务的增长给高德地图这个超级 APP 带来了极大资源压力，因此我们沉淀了一套自适应资源管理框架，

来满足不同业务场景在有限资源下能够做到功能和体验极致均衡。主要的设计思路是通过监测用户设备等级、系统状态、当前业务场景以及用户行为，利用调度算法进行实时推算，统一管理协调 APP 当前资源状态分配，对用户当前不可见的内存等资源进行回收。

自适应资源管理框架—内存部分

可以根据不同的设备等级、业务场景、用户行为和系统状态来管理资源。各业务都可以很容易的接入此框架，目前已经应用到多个业务场景，均有不错的收益。



数据验收

通过三个版本的连续治理，前后台导航场景均有 50% 的收益，同时 Abort 率也有 10%~20% 的收益。

整体收益算是比较乐观，但是随之而来的挑战是我们该如何守住成果。

长线管控

所谓打江山容易守江山难，如果没有长线管控的方案，随着业务的版本迭代，不出三五个版本就会将先前的优化消耗。为此我们构建了一套 APM 性能监控平台，在研发测试阶段发现并解决问题，不把问题带上线。

APM 性能监控平台

为了将 APP 的性能做到日常监控，我们建设了一套线下「APM 性能监控平台」，平台能够支持常规业务场景的性能监控，包括：内存、CPU、流量等，能够及时的发现问题并进行报警。再配合性能跟进流程，为客户端性能保障把好最后一关。

内存分析工具

Xcode memory gauge：在 Xcode 的 Debug navigator 中，可以粗略查看内存占用的情况。

Instruments – Allocations：可以查看虚拟内存占用、堆信息、对象信息、调用栈信息、VM Regions 信息等。可以利用这个工具分析内存，并针对性地进行优化。

Instruments – Leaks：用于检测内存泄漏。

Instruments – VM Tracker：可以查看内存占用信息，查看各类型内存的占用情况，比如 dirty memory 的大小等等，可以辅助分析内存过大、内存泄漏等原因。

Instruments – Virtual Memory Trace：有内存分页的具体信息，具体可以参考 WWDC 2016 – System Trace in Depth。

Memory Resource Exceptions：从 Xcode 10 开始，内存占用过大时，调试器能捕获到 EXC_RESOURCE RESOURCE_TYPE_MEMORY 异常，并断点在触发异常抛出的地方。

Xcode Memory Debugger：Xcode 中可以直接查看所有对象间的相互依赖关系，可以非常方便的查找循环引用的问题。同时，还可以将这些信息导出为 memgraph 文件。

memgraph + 命令行指令：结合上一步输出的 memgraph 文件，可以通过一些指令来分析内存情况。vmmap 可以打印出进程信息，以及 VMRegions 的信息等，结合 grep 可以查看指定 VMRegion 的信息。Leaks 可追踪堆中的对象，从而查看内存泄漏、堆栈信息等。heap 会打印出堆中所有信息，方便追踪内存占用较大的对象。malloc_history 可以查看 heap 指令得到的对象的堆栈信息，从而方便地发现问题。

总结：malloc_history ==> Creation； leaks ==> Reference； heap & vmmap ==> Size。

MetricKit：iOS 13 新推出的监控框架，用于收集和处理电池和性能指标。当用户使用 APP 的时候，iOS 会记录各项指标，然后发送到苹果服务端上，并自动生成相关的可视化报告。通过 Window -> Organizer -> Metrics 可查，包括电池、启动时间、卡顿情况、内存情况、磁盘读写五部分。也可以 MetricKit 集成到工程里，将数据上传到自己的服务进行分析。

MLeaksFinder：通过判断 UIViewController 被销毁后其子 view 是否也都被销毁，可以在不入侵代码的情况下检测内存泄漏。

Graphics API Debugger：Google 开源的一系列的 Graphics 调试工具，可以检查、微调、重播应用对图形驱动的 API 调用。

Arm Mobile Studio：专业级 GPU 分析工具。

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德智慧交通地图空间可视化 SDK 设计与实现

作者：峻山

一、背景

地图空间可视化作为高德智慧交通前端业务中最重要的功能之一，承担着城市交通大脑、全境智能大屏等业务中大量的地图渲染需求。

作为向用户展示交通数据的窗口，我们需要展现省、市、区、商圈、自定义区域多种场景，包括所有交通事件、拥堵指数、辖区等多种维度的数据，呈现着数据量大、元素种类多、逻辑展现重等特点。

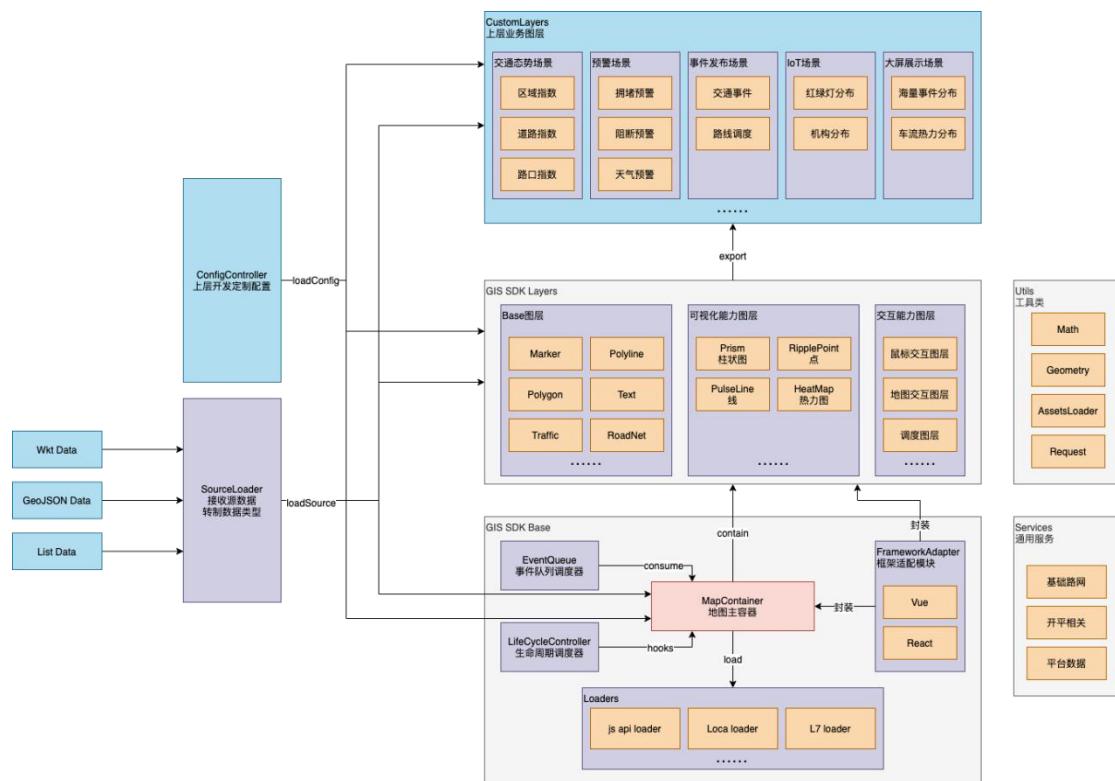
JSAPI 作为高德地图前端战线的引擎，涵盖着渲染地图、展示覆盖物等底层能力，但对于行业应用领域的开发来说，存在着开发难度大、适配成本高、纯原生 JS 实现与主流框架结合不紧密，无行业图层能力的问题。

基于以上原因，我们设计了具有适用于垂直行业的、可复用、可扩展、二次开发简单等特点的地图 SDK，已经成为智慧交通地图空间可视化能力的首选方案。

二、方案设计

整体框架设计方案

高德智慧交通团队经过大量项目实践和思考，以交通行业为切入点，面向整个前端行业地图设计了一套地图空间可视化开发的 SDK，整体功能架构设计如下图所示：



- (1) MapContainer 是整个 SDK 的基座，用于承载地图引擎，装载在其上渲染的覆盖物图层，加载所需要的框架模块，在整个架构中起到中流砥柱的作用。
- (2) 配置控制器负责传入用户配置，包括地图应用 key 配置、加载可选功能配置、样式配置等，在用户变更这些配置后，它会把更新后的配置信息传递到流程中的其他模块中。
- (3) 接受数据的工作由 SourceLoader 完成，设计了一套 SDK 内部使用的标准化的数据格式，Loader 负责将用户传入的不同类型的数据(已经支持 GeoJSON、WKT、数据列表等形式)转化成专用标准格式数据，分发到地图容器及各图层中。
- (4) 为了支持不同的主流应用框架，将框架适配层单独拆分，由它将主要模块封装成 Vue、React 等框架兼容的组件形式，实现多框架扩展。
- (5) 地图 API 调用有着严格的顺序限制，而封装框架对于图层各个生命周期的触发是异步的，乱序的，存在无法保证流程一致性的问题，为了应对这种情况我

们在 SDK 中引入了事件队列机制。

状态驱动方案的实现

1. 生命周期设计

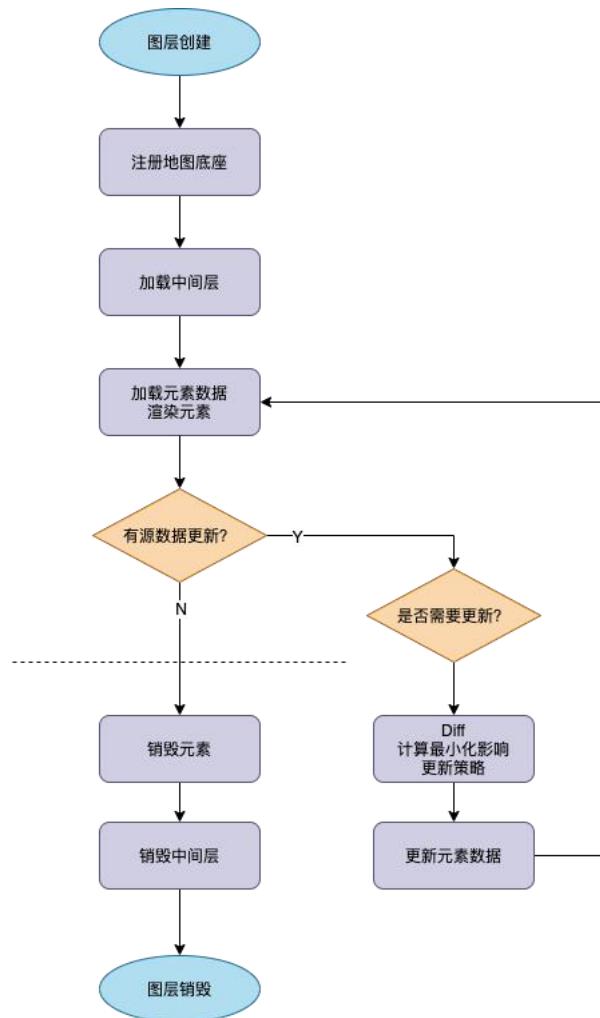
地图 JS API 的调用逻辑与原生 Javascript 一样，是命令式调用设计，像下面这样：

```
1. // 创建地图  
2. const map = new AMap.Map(options);  
3.  
4. // 添加覆盖物  
5. const marker = new AMap.Marker(markerOptions);  
6. map.add(marker);  
7.  
8. // 修改覆盖物属性  
9. marker.setContext(newContext);  
10.  
11. // 移除覆盖物  
12. map.remove(marker);  
13. map.destroy();
```

这样的 API 调用方式，与上层项目的开发框架，如 Vue、React 等不匹配，如果在一个状态驱动的框架下充斥着大量命令式驱动的代码，会大幅度降低这个项目的可维护性、可扩展性。

为了更好地支撑开发的需要，所有业务图层抽象出了一套完整的生命周期流程。不同的图层，渲染逻辑的步骤不完全相同，各图层的额外能力，如支持交互事件的能力、动画能力也不尽相同，但都可以囊括在这一套生命周期结构内。

SDK 图层组件生命周期定义如下：



(1) 地图注册

在地图底座加载完毕后，会通知各个图层的 RegisterMap 流程，这是图层组件生命周期的第一步，图层中包含的所有元素都在这之后才会开始渲染。

(2) 中间层加载

部分类型的元素需要分组批量加载，因此在渲染这些元素之前，需要先将对应的组图层加载出来。因此，我们设计了组图层相关的生命周期，相关逻辑只需在

beforeAppendGroup, appendGroup, afterAppendGroup 这些流程中实现即可。

(3)元素加载

beforeAppendComponent, appendComponent, afterAppend Component, 这些是元素图层中最重要的流程，用于实现图层元素加载的主逻辑。

其中，对于一些元素需要有前置检查，有数据校验，可以把相应的检查逻辑放入 beforeAppend 中；有的元素需要注册交互事件，或者需要有添加动画 scheme 能力，这部分的实现逻辑可以放到 afterAppend 流程中。

与之对应的，还有元素的销毁流程，
beforeRemoveComponent, removeComponent, afterRemoveComponent。如果元素绑定了交互事件，将会在 beforeRemove 的时候解绑；如果元素注册了动画或者周期调用，也会在 beforeRemove 的时候销毁周期 timer。

(4)元素更新

shouldUpdate, diff, updateComponent，用于实现组件数据动态更新后图层元素的 diff、更新过程。

其中为了防止源数据中只有一小部分修改导致整个图层全部重绘的情况，我们在其中加入了 diff 的算法，通过各图层的校验数据 key 的方法，筛选出变更前后一致的数据项，只重绘不同的数据，大大提升了渲染流程的效率。

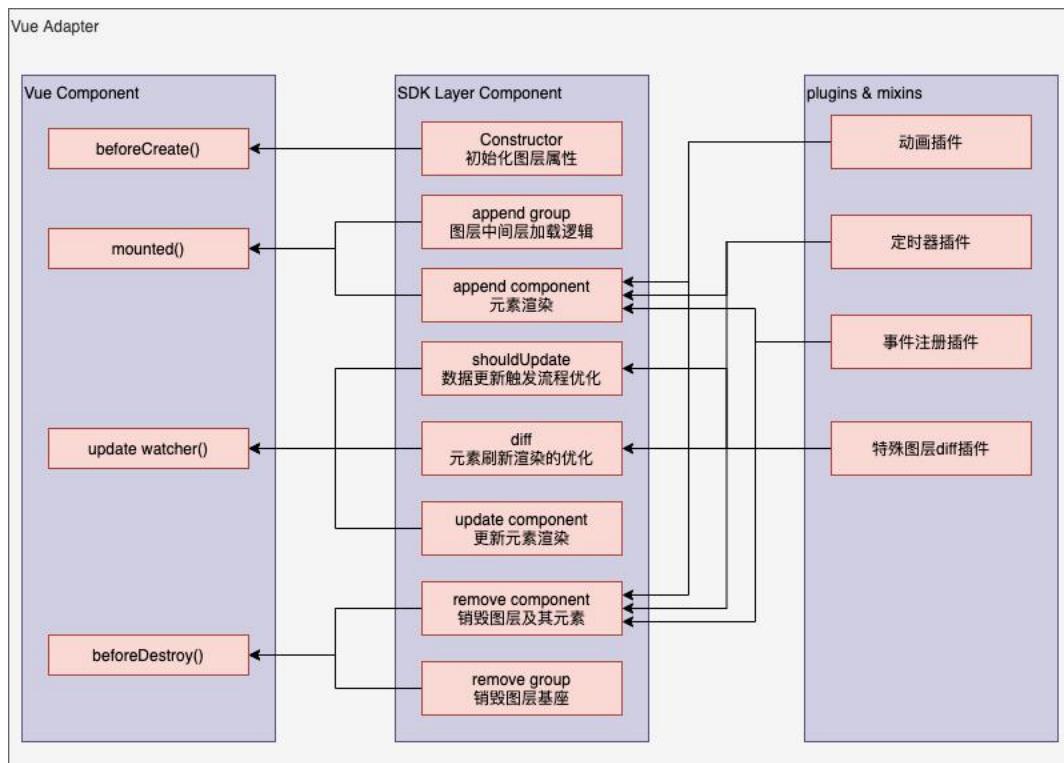
2. 插件的实现

不同图层之间存在共同处理流程和共同的属性，对此，我们设计了各种可复用的内置插件，供各图层根据自身特性组合使用。

例如，有实现定时刷新效果的 scheme 插件，实现动画效果的 animate 插件，实现注册交互事件的 event 插件等。这些插件的设计，必须遵守组件生命周期的规范，插件功能的实现逻辑，也全部以注册上述的生命周期函数的方式完成。

这些生命周期需要与主流框架的生命周期设计适配。以目前我们项目中正在使用的 Vue 框架举例，Vue 也有其自己的组件生命周期，它的设计基本能够与我们

的周期函数相匹配。因此，针对 Vue 的适配过程其实并不怎么难：



Vue 自身有一套不同层级的组件之间的加载控制流程，父子层级、兄弟层级之间的组件有着严格的触发顺序。例如，父组件的 `beforeCreate` 总是在子组件 `beforeCreate` 之前触发，而父组件的 `mounted` 又总是在子组件 `mounted` 之后才会响应，这与我们的多层次图层之间想要的触发顺序相符。因此，SDK 图层的各生命周期总能在 Vue 中找到与之对应的触发时间点。

经过封装后，用 SDK 实现的地图模块在项目中生成的组件树结构如下：

```
▼ <Root>
  ▼ <NewDemo>
    ▼ <CommonMap>
      <BaseLayerTab>
    ▼ <ExtendLayerTab>
      <DistrictMaskPolygonLayer>
      <DistrictHiddenPolygonLayer>
      <EventElasticMarkerLayer>
      <MilePostLabelsLayer>
      <EarlyWarningElasticMarkerLayer>
      <WeatherEarlyWarningElasticMarkerLayer>
      <PcheckEarlyWarningElasticMarkerLayer>
      <CrossTurnPolylineLayer>
      <CrossTurnArrowMarkerLayer>
    <Base3DLayerTab>
    <Extend3DLayerTab>
    <ToolLayerTab>
```

3. 异步流程的一致性设计

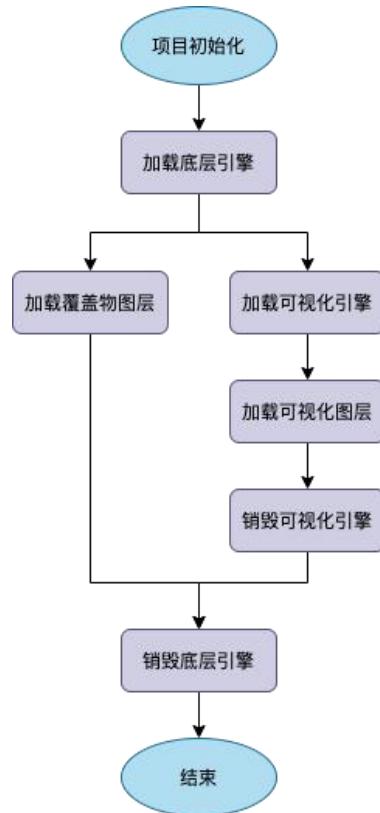
我们使用的底层地图引擎，对于流程逻辑的顺序有着严格的要求：

(1) 地图底座的创建须在所有其他流程之前。

(2) Loca、L7 底座的初始化须在地图底座创建完成之后。

(3) 地图元素需要在地图底座加载完成后才能够开始加载，销毁也需要在地图底座销毁之前完成。

(4) 需要确保状态与结果的一致性，如果在短时间内触发了大量的更新数据的操作，即使底层引擎处理需要很长的时间，也要保证最终的展示结果与更新的顺序完全一致。



不幸的是，虽然主流的框架有完善的生命周期管理机制，能够确保各个流程的执行顺序不出差错，但这些流程之间都是异步的、并发的，而绘图引擎在处理这些渲染指令时，会由于处理时长的不确定性，导致各指令返回的顺序有所变化，这可能会导致下面的情况出现：

- 地图容器的加载时间过长，导致加载后续元素时，地图仍没有渲染完成而出错；
- 在短时间内对同一份数据进行变更，如果引擎处理第一次变更花的时间比后一次更长，就会导致第一次更新的结果渲染出来时，会把更早完成的第二次渲染结果覆盖掉。

为了避免上述情况，我们在 SDK 中实现了事件队列控制器，处理顺序问题：



(1)所有图层组件中需要调用底座引擎的事件，例如 `append component`, `remove component` 等，不会直接调用底座的相关接口，而是在队列控制器中 push 一个对应类型的事件。

(2)队列控制器中的所有事件类型，全部封装成同步方法实现。由控制器收集所有涂层的调用消息，单线程逐一消费。

(3)在控制器中写入特殊的控制逻辑，地图基座的加载需要在其他图层加载之前，则把基座加载的事件的响应优先级设置为最高。

(4)引入筛选机制，针对队列中存在同一图层的互逆操作，如短时间内加载一份数据，之后又 remove 掉，由于这一对操作不会对当前的结果有任何影响，因此这一对操作将会被过滤逻辑删除，达到优化渲染性能的效果。

4. 地图控制指令的优化

地图底座支持用户通过调用相关方法控制地图展示的视野，SDK 在这种设计上加以优化，通过在地图底座组件上配置相应的属性状态，来实现定位到选定元素、定位到整个辖区范围、定位到特定地点及缩放级别等多种视野类型。

同时，地图的其他控制方法，例如设置周边避让区域、设置光标形状、设置自定义地图样式等方法，也全部改为传递 props 属性的方式实现。

	未使用SDK时 修改地图属性的方式	使用SDK修改地图属性方式
触发方式	指令驱动	状态驱动
示例	<pre> 1 // 初始化地图 2 const map = new AMap.Map(options); 3 // 设置各属性 4 map.setZoom(zoom); 5 map.setMapStyle(mapstyle); 6 ... </pre>	<pre> 1 <Map 2 :zoom="zoom" 3 :mapStyle="mapStyle" 4 ... 5 /> </pre>
特点	调用繁琐，不易封装，需要开发者手动维护	现代化API，修改方便，开发者无需关注底层

三、其他优化

地图实例缓存

就我们使用的底层地图引擎来说，创建、销毁一个地图底座需要消耗大量的性能，而有时候这样的操作是可以避免的。有时候我们只是切换了一个页面路由，图面上展示物并不需要有什么变化，但仍然会触发地图底座的销毁与重新生成。这个流程是多余的。

为了优化这个问题，我们设计了可以容纳 2 个底座实例的缓存容器。每次在执行销毁地图的命令时，我们并不会真正的销毁它，而是把它隐藏掉并存入缓存中。

下次需要创建实例时，直接在缓存中找到符合要求的实例拿来用。

多实例环境隔离

随着下游业务项目的功能迭代，产品提出了在同一个页面内展示多个 SDK 底座实例的要求。对此，我们对 SDK 进行了一系列的优化：

- 改造消息队列控制器，原来的单线程模式已经不再适用，现在已可以支持实例隔离，不同实例之间独享事件队列和流程控制逻辑。
- 优化图层与底座的从属判定机制，在多个底座之间存在父子关系的情况下，能够让图层在最合适的底座上展现。

GL 渲染 Context 没有正确 GC 回收导致的崩溃问题

在为 L7 编写加载器时，遇到了内存泄露的问题：如果在项目中使用了 L7 相关图层，销毁时 L7 使用的 WebGLRenderingContext 资源不会正确释放，反复创建销毁几次后，浏览器会因为内部的 renderingContext 资源不足而渲染崩溃。

分析 L7 源码后发现，L7 为了实现与地图同步 resize，在地图容器 DOM 上注册了一个 resize 事件，并把这个事件的处理函数绑定在了这个容器 DOM 的一个叫 __resize__trigger__ 的属性上。

如果开发者在项目中使用 Vue 作为前端框架，Vue 的模板更新机制会引起 DOM 的重绘，在一次数据变更之后，它会把原来的容器 DOM 销毁，替换为一个新的。

但由于注册的事件函数中含有 DOM 对象引用的缘故，虽然旧的 DOM 对象已经从 DOM tree 上移除，但并不会被 GC 回收，而是仍然被 __resize__trigger__ 这个函数引用着，同时由于新生成的 DOM 不具有该属性，导致在 L7 引擎销毁的时候，由于 L7 找不到这个函数，resize 事件解绑也会失败。在开发者触发多次切换引擎操作之后，有大量的未被实际引用的容器 DOM 无法被回收，而这些 DOM 中又都包含着 webGL Canvas 对象，导致浏览器的 GLRendering 资源不足的问题出现。

解决方法：我们无法修改 L7 的源码，因此也无法更改它注册、解绑事件的逻辑。但我们可以通过在每次 Vue

刷新之前，对即将被移除的 canvas 的 width 和 height 设置为 0，以此来直接释放 renderingContext 资源，实测有效。

最佳解决方案：目前我们已经有自行实现的 3D 图形类，且也扩展了对 Loca 等其他可视化库的支持，可以摆脱对单一库的依赖，实现相同的能力。

四、多维数据比对

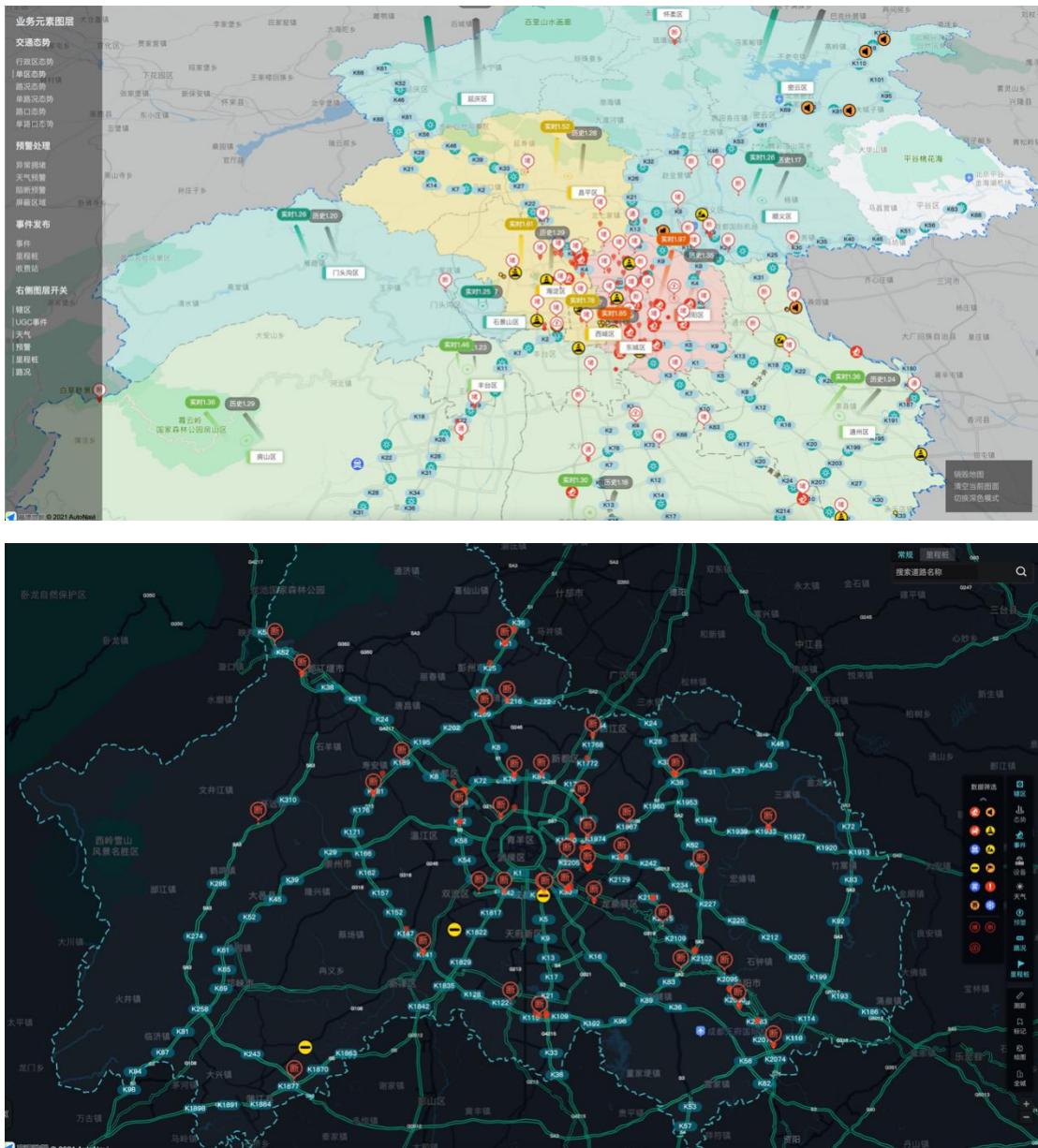
经过高德智慧交通大量的项目实践和数据比对，充分证明了地图空间可视化 SDK 开发的必要性，业务价值和技术价值都经历了项目的考验。

以高德交通大脑和全境智能大屏的数据比对可以得到使用 SDK 之前和之后的数据比较：

JSAPI原生地图渲染方案	SDK方案	
图层能力	比较原生、单薄	丰富
行业场景支持覆盖	无	覆盖交通行业、易扩展
功能开发代码量	-	下降65%
首屏渲染性能	-	提升200%+
页面切换性能		

项目落地效果：

大前端篇-高德智慧交通地图空间可视化 SDK 设计与实现



使用 SDK 后的项目开发代码：

```
1. <template>  
2.     <!-- 地图底座 -->  
3.     <CommonMap  
4.         :center="center"  
5.         :zoom="zoom"
```

```
6.      :city="fitViewCity"
7.      view-mode="3D"
8.      :map-style="mapStyle"
9.    >
10.   <!-- 场景 1 -->
11.   <TrafficScene>
12.     <!-- 地图元素层 -->
13.     <TrafficPointLayer :list="trafficPointList" />
14.     <!-- 带有事件监听的地图元素层 -->
15.     <TrafficRoadMarkerLayer
16.       :list="trafficRoadList"
17.       @click="handleTrafficRoadMarkerClick"
18.     />
19.   </TrafficScene>
20.   <!-- 场景 2 -->
21.   <PublishScene>
22.     <PublishMarkerLayer
23.       :list="publishList"
24.       @mouseover="handlePublishMouseOver"
25.       @mouseout="handlePublishMouseOut"
26.       @click="handlePublishClick"
27.     />
```

```
28.    </PublishScene>
29.    <!-- 可视化场景 -->
30.    <VisualizationScene use="amap-loca">
31.    <!-- 可视化数据层 -->
32.    <TrafficRoadLineLayer :list="trafficRoadList" />
33.    </VisualizationScene>
34.    </CommonMap>
35. </template>
```

五、展望

经过高德智慧交通大量项目的实践，SDK 的建设已经趋于成熟，开发简单、稳定性高、性能好的特点可以很好地降低开发者使用高德开放平台 JSAPI 来开发地图空间可视化项目的成本。因此，我们计划以开发者官网的形式对外输出，更好地服务于开发者。

地图建筑群的光影效果原理和应用实践

作者：叶其

背景

高德开放平台在 2020 年初推出了 AMap JSAPI 2.0 版本，现在版本已经稳定下来。在 JSAPI 2.0 版本中我们采用了新的渲染管线，在每个渲染流程中都针对性的进行了性能优化。

因此，各方面相较于上一个稳定版本都有了很大的提升。为了还原更加真实的世界，我们希望地图中的建筑元素拥有更多的光影效果，来模拟真实世界中无处不在的光照。并且，真实世界的楼体建筑也应该拥有不同的材质，对光源有不同的响应。

今天我们就来聊一下地图上立体建筑元素的光照效果如何渲染。

现状

目前主流地图服务商的地图渲染效果都有各自的侧重点，主要的方案有以下几种：

方案一：楼体没有高度信息，只有贴地的多边形色块代表。



方案一效果图

方案二：拥有盒子形状楼体渲染，支持太阳光模拟光照的简单立体效果。在某些角度下会出立面无法区分的情况。



方案二效果图

方案三：猜测利用倾斜摄影资源，类似 3DTile 的加载方式，实现了接近真实世界的渲染效果，而且性能优异。



方案三效果图

当时，高德使用的也是类似方案二中的简单光照盒子效果的立体建筑，我们希望在新的数据资源下能够渲染出更加真实的效果。具体希望达到以下效果：

- 实现更加柔和立体的光照效果，解决立面在某些角度下会由于光照强度和角度一致导致糊成一片，丧失立体效果。
- 实现照亮指定的区域，模拟城市的夜晚效果。

- 满足某些特殊场景下立体物体的金属光泽特效。

方案调研

Blinn-Phong 光照模型

Blinn-Phong 光照模型中规定了一个片元的颜色是有三个不同的光反射量组成：

• 环境光

环境光是模拟现实世界中各种光源在不同的物体表面无休止的反射而最终生成的光，因为现实世界光的反射太过于复杂，在冯氏光照中我们直接定义一个光源，无差别的反射在每个物体的表面来模拟环境光。

$$\langle \text{环境光} \rangle = \text{环境光强度} \times \text{环境光颜色}.$$

• 漫射光

漫射光是模拟物体对一个光源的反射强度。从下面的图中我们能看到，漫射光的反射强度取决于物体表面的法向量和入射光的夹角，夹角越大物体反射越弱，反之，越强。

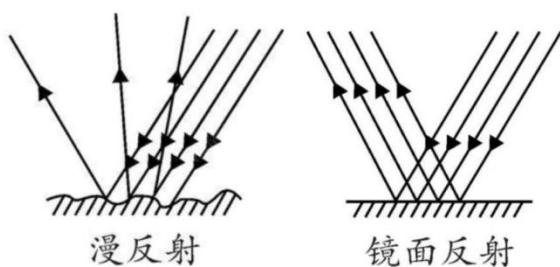
$\langle\text{漫射光}\rangle = \text{光强度} \times \text{光颜色} \times \cos(\text{光和平面法向量的夹角})$ 。

• 镜面反射光

镜面反射主要针对表面拥有粗糙度的物体，当物体表面比较光滑的时候——比如一个由抛光金属制成的物体，那么光照射到物体表面会形成光点或者光斑。

$\langle\text{镜面反射光}\rangle = \text{光强度} \times \text{光颜色} \times \cos(\text{光在平面上的反射夹角})^{\wedge}\text{平面光泽度}$ 。

一个片段颜色就是三个反射的和：总反射 = 环境反射 + 漫反射 + 镜面反射。



光源和材质

灯光按照不同的类型大致分为四类：

- 环境光

上面冯氏光照中已经介绍。

- 平行光

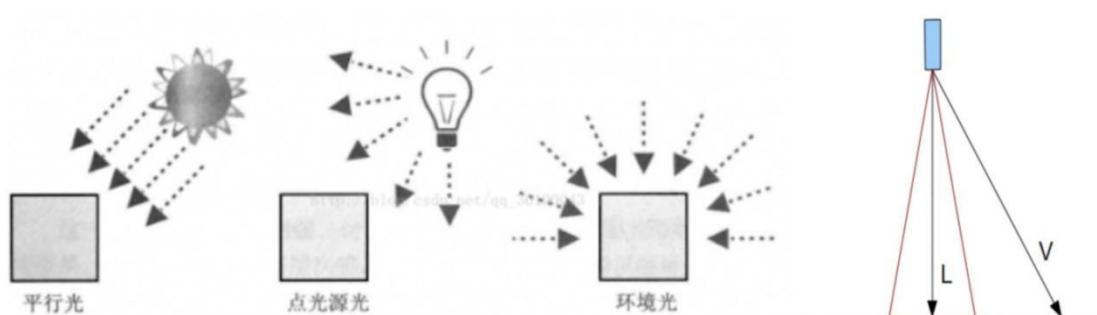
是一种理想型光源，主要模拟真实世界中的太阳光，因为太阳距离地球很远，射向地球的光可以近似的看做平行光。

- 点光

点光源主要模拟的是现实世界中的灯泡光。也可以理解为太阳是整个太阳系中的一个巨大点光源。

- 聚光灯

聚光灯光源也可以认为是锥形光源，一般使用在一些特殊场景中。

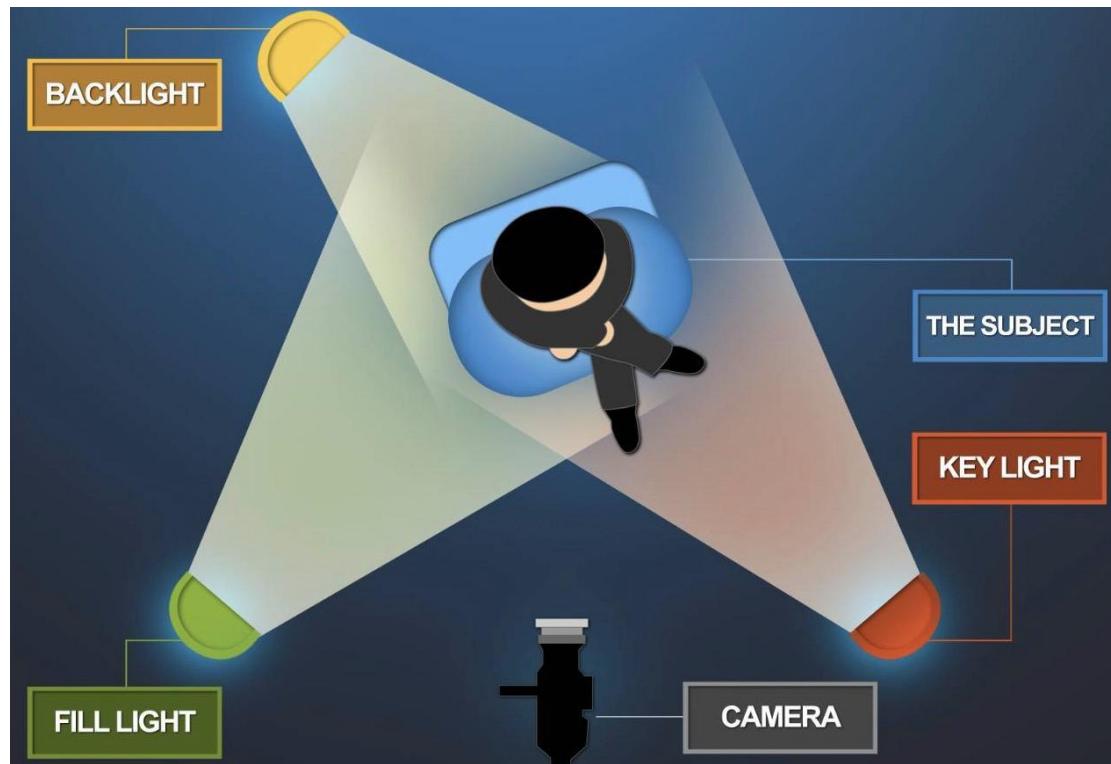


(图片来源于网络)

三点布光法

为了解决单光源导致某些角度下的光线死角问题，我们发现了一种经典和常用的打光方式：三点布光法。

它侧重指灯位在不同的三个点上。用三个不同强度和方向的光分布在一个场景中不同的位置将整个场景照的层次分明。



(本图来源于网络)

主要分为：

主光：它规定了方向、角度、与范围，规定了照明光轴与照射角。起着主要造型和确定光影格调的作用，确定了整个环境的主体明暗基调。

辅助光：它起着辅助主光未照明的区域并通过副光来调整光比，柔化主光形成的阴影。避免光的背面一片漆黑而失去立体感。

轮廓光：前两度光完成后，需要把物体与环境隔开，产生一种深度与层次。增强整个环境的柔和度和层次感。

对比和实施

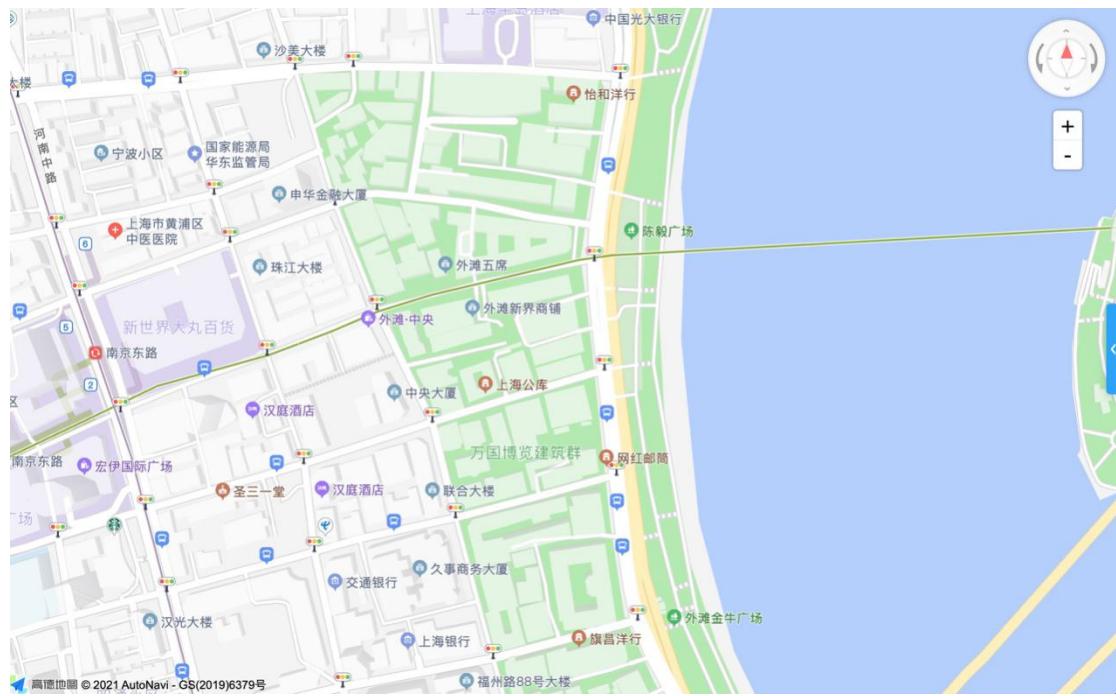
为了得到比较好的物体光照效果和场景渲染，我们最终决定对上面的三种技术方案都进行实现。

使用 Blinn-Phong 光照模型模拟物体反射，使用环境光、平行光、点光进行光照模拟，最后使用三点布光法进行光源分布。这样在场景中能保证最好光照效果。

高德 JSAPI 2.0 版本中，我们绘制了简单的立体楼块。

下面，我们开始使用新的光照方案。

大前端篇—地图建筑群的光影效果原理和应用实践

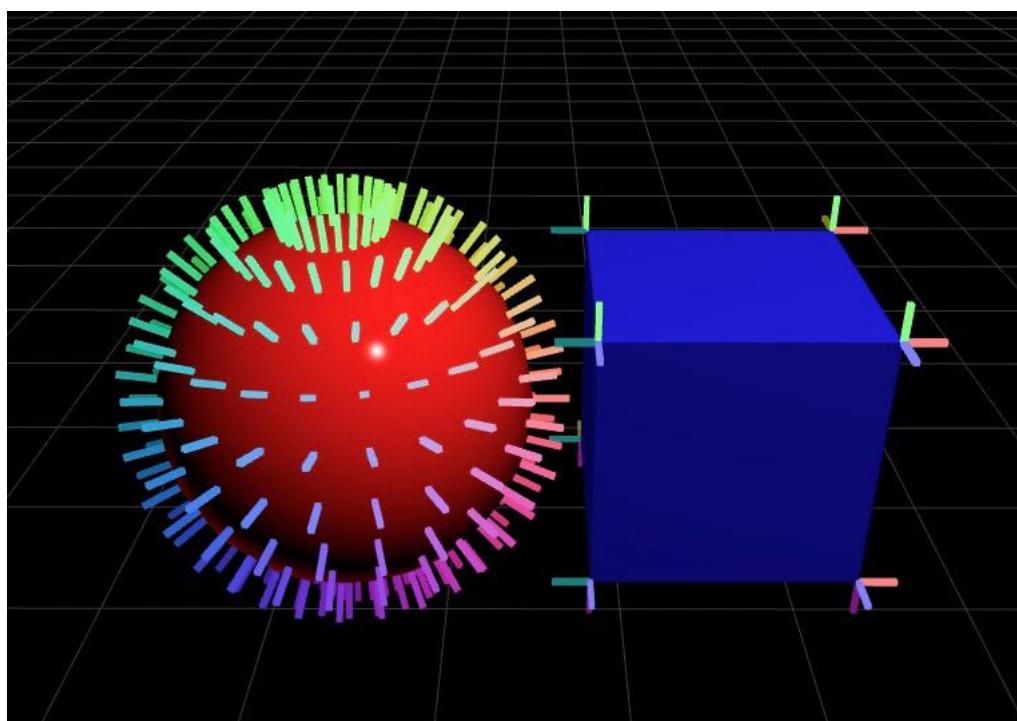


Blinn-Phong 光照模型是基于每个面的法线和光照的方向两个向量，对这两个向量进行各种运算来模拟真实世界中的光照对物体的影响。

我们先着手准备每个平面的法向量。每一个平面其实都有两个法向量，这两个法向量代表平面的“正面”和“反面”。平面中计算法向量的公式就是平面上两个任意向量的叉乘。

$\langle \text{平面法向量} \rangle = \text{cross}(\text{平面向量 A}, \text{平面向量 B})$ 。

根据平面自身的缠绕方向，我们需要计算出楼块面朝外的法向量。

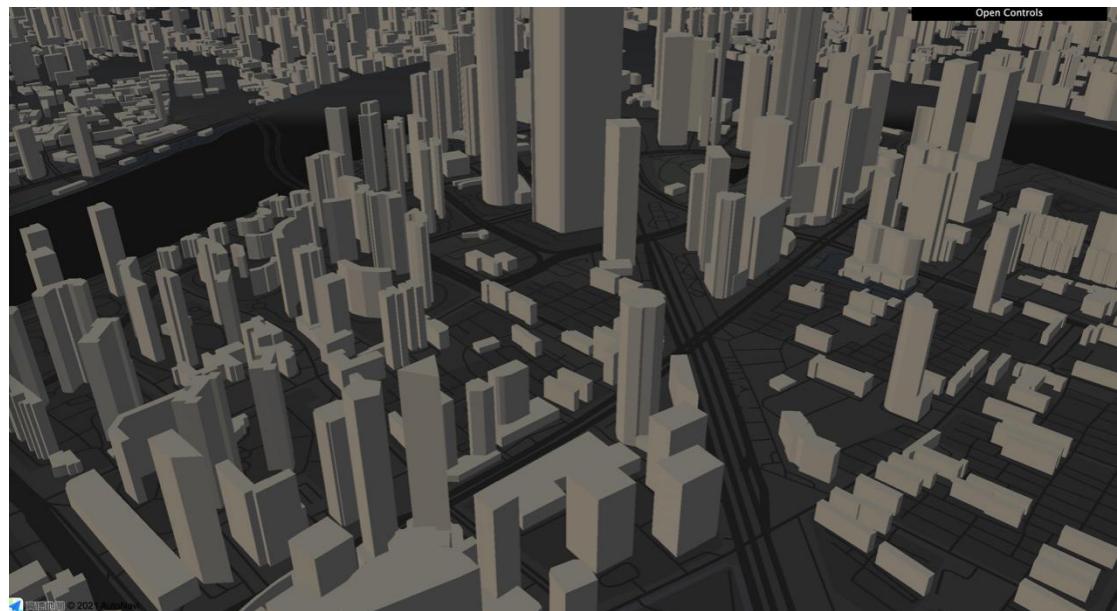


(本图来源于网络)

对于环境光和平行光来说是比较简单的，只需要光照强度、光的颜色、光的方向（平行光）。但是对于点

光源来说稍微复杂一点，还需要有光的衰减。通常来说一个点光源对不同距离远的物体的光照强度是不一样的，因此需要对物体上的每一个片元都进行衰减计算。

加上平行光和环境光之后我们有了这样一个楼体效果：



为了能聚焦到主要物体上面，应该使用点光源将需要聚焦的范围点亮，让整个场景更加有层次感。

点光源的位置就设置在东方明珠上方 1000 米处的空中，设置光的衰减距离为一万米。这样他就会照亮整个一万米范围的城市建筑。从下图中我们就可以看出点光源照亮了整个东方明珠的区域。



至此，地图楼体建筑的光照效果已经基本完成。

小结

我们首先使用平行光从正南方向照射进场景中，来模拟太阳的正点照射。但是仅仅有这一个光源会导致上

面提到的在某些角度下楼体的立面还是会出现整片的暗色。因此，我们需要在地图的北方增加一个弱化的背面平行光，用来点亮楼体背面。最后，为了突出东方明珠这块区域，又增加了一个半径为十公里的点光源，这样整个区域的光照就能很好的突出层次感。

在地图渲染这个方向上，有很多基础理论可以借鉴其他领域中已经很成熟的理论和最佳实践，来帮助我们渲染出更加接近真实世界的地图。

参考文献

- [1] Andreas Anyuru [美] 著，吴文国译《WebGI 高级编程——开发 Web 3D 图形》. 北京：清华大学出版社，2013.06. 第一版
- [2] Kouichi Matsuda [美]，Rodger Lea [美] 著，谢光磊译《WebGL 编程指南》. 北京：电子工业出版社，2014.6

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

算法篇

导航定位向高精定位的演进与实践

作者：子路

导读

地图软件现在已成为人们出行必备的重要辅助工具。为了实现准确的导航，首先必须准确确定人或车的当前位置。因此，定位技术就是实现导航功能的基石。

本文较系统的介绍了手机、车机导航定位中使用的关键技术，以及高德地图在这些关键技术中的进展。最后，讨论了在传统导航向自动驾驶的演进过程中，定位技术的演进路径。

1. 导航定位框架

导航定位的核心业务目标是为导航服务提供连续可靠的定位依据，包括：当前在哪条路上，是否偏离路线，距离下一个路口有多远，等等。

为实现这一目标，首先需要接收定位信号输入。最常见的定位信号是 GPS，其可以提供全域米级精度（5~10m）的位置信息。在此基础上，大部分手机同时配置了惯性传感器

(陀螺仪、加速度计) 和磁力计，还有部分手机配置了气压计，可以感知高程方向的位置变化。

对于车机，通过 CAN 总线获取的车速脉冲、方向盘转角等信息是另一类重要的定位输入。基于上述定位信号，应用姿态融合、航位推算等算法，计算出连续可靠的位置和姿态。再依据地图数据将人/车的实际位置与地图道路关联，实时判断当前是否已经偏离导航路线，或更新当前在导航路线中的相对位置。

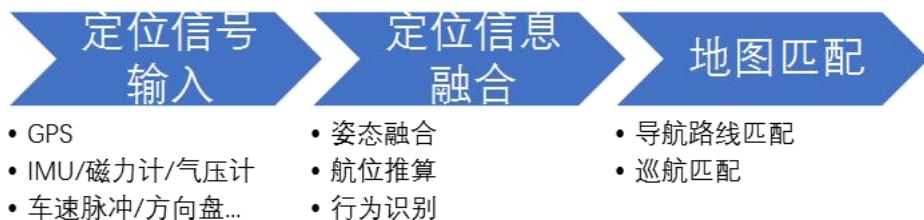


图 1 导航定位框架

在上述定位框架中，针对不同端的形态（手机/车机），输入定位信号的配置不同，使用的定位技术及覆盖的定位场景也不同。

对于手机，存在步行、骑行、驾车等多种使用场景，需要对用户行为进行识别。在步行场景下，由于速度较低，GPS 方向不准确，手机姿态通过融合惯导和磁力计计算实现。

在驾车场景下，位置和姿态主要由 GPS 提供，针对 GPS 跳跃、漂移等复杂情况设计可靠的地图匹配算法是手机定位重点要解决的问题。

对于车机，只存在驾车使用场景。同时，由于车机具备稳定的安装状态并可以提供更丰富的车辆 CAN 总线信息，基于这些信息设计航位推算及融合算法，解决隧道、高架、平行路等复杂场景的连续定位问题是车机定位的重点。

2. 手机导航定位

2.1. 姿态融合技术

常用的姿态融合技术又称为 AHRS (Attitude and heading reference system)。对于六轴惯性传感器融合，包含陀螺仪和加速度计，其 AHRS 算法如下图所示。陀螺仪测量的是角速度，角速度积分即可得到某一时间段内的角度变化。加速度计测量的是物体的加速度，包含重力加速度，当静止时，通过获得重力加速度在三个轴上的分量可以计算相对倾斜角度。AHRS 算法采用滤波方法，如互补滤波、Kalman 滤波，对不同传感器姿态进行融合。

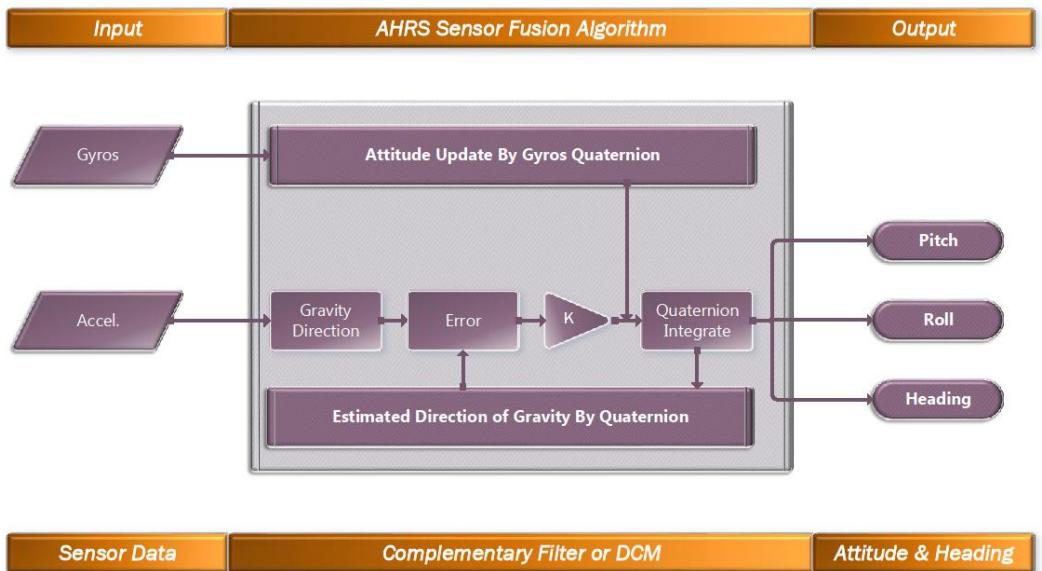


图 2 AHRS 融合算法

对于九轴传感器，额外提供了三个轴向的磁力计方向，同样利用上面的算法框架进行融合。

2.2. 地图匹配技术

传统的地图匹配方法是在定位点附近道路中，根据距离、方向接近等一些判断准则找到最可能是汽车行驶道路的匹配道路。这种方法实现简单，但通常 GPS 定位误差是十米，在信号干扰、遮挡的情况下可以达到几十米甚至上百米，而地图测绘误差、地图简化误差同样可以达到十几到几十米。在各种误差条件下，单纯依靠距离、方向这样的几何特征判断做策略匹配是很不稳定的。

对于一个好的地图匹配算法，为了稳定准确的确定匹配道路，需要综合利用定位源和地图的各种输入数据，做融合计算，并根据汽车行驶的特点对各种特殊场景做处理。对于多源信息融合，隐马尔可夫（HMM）是一个比较常用并且有效的方法，因此我们采用 HMM 作为匹配算法的核心，并辅以场景策略算法，实现地图匹配。

在 HMM 地图匹配算法中，匹配道路是未知的，作为隐藏变量 z_n 。每个时刻观测到的 GPS 定位信息是观测变量 x_n 。地图匹配的目标是在已知定位信息的情况下对匹配道路进行估计：

$$\max_{z_1, \dots, z_n} P(z_n, \dots, z_1 | x_n, \dots, x_1)$$

对上述问题，可以采用维特比算法用递推的方法进行计算。

在 HMM 框架下建立地图匹配模型，核心在于确定发射概率模型和转移概率模型。发射概率模型的确定依据定位位置和方向。

1) 对定位位置，与道路距离越接近概率越大，反之概率越小。同时考虑匹配道路的选择对横向距离误差较敏感，对纵向距离误差不敏感。采用正态分布建立模型。

2) 对定位方向，与道路方向越接近概率越大，反之概率越小。同时概率与速度有关，速度越大越可信。采用 Von Mises 分布，以速度为超参数建立模型。

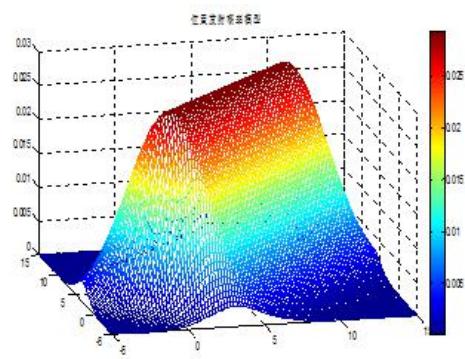


图 3 位置发射概率

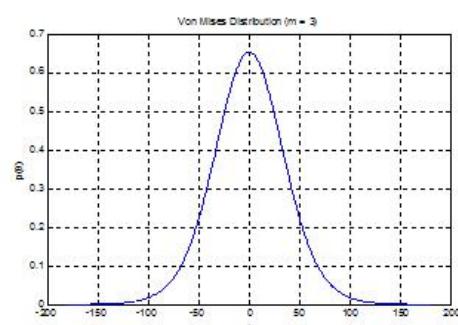


图 4 方向发射概率

转移概率模型的确定依据道路距离、道路转角对汽车行驶的约束建立。

道路转弯的角度越大，速度较大的概率越低。采用 Von Mises 分布，以速度为超参数建立模型。根据车速和时间差计算移动距离，该距离和路径移动距离越接近，概率越大。采用指数分布建立模型。

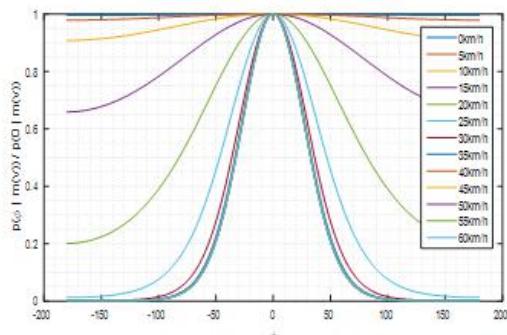


图 5 速度转移概率

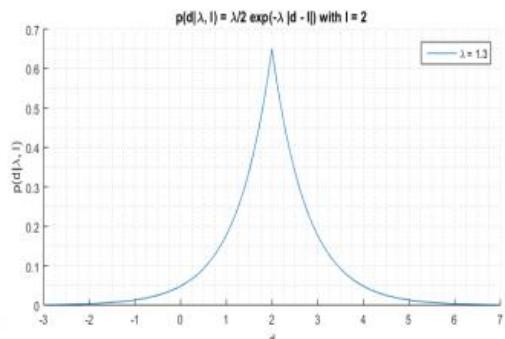


图 6 移动距离转移概率

上述算法在高德地图手机 APP 上实现了落地，为驾车导航提供准确的定位匹配结果，用于导航的引导播报等功能。对比原来使用策略的地图匹配方法，HMM 算法在匹配准确率和稳定性上都有显著提升。

3. 车机导航定位

3.1. 车机定位方案

对于车机导航，如何充分利用车辆传感器和总线信息，优化驾车导航各种复杂场景体验是定位要解决的核心问题。其中，复杂场景包括：隧道、地下停车场定位失效，城市峡谷区域定位漂移等。

解决上述痛点问题的关键在于多传感器融合技术。例如，当 GPS 漂移或失效时，利用车速脉冲与惯导融合的航位推算技术进行持续定位，但航位推算会产生累计误差，需要

地图数据进行反馈矫正，同时地图数据和 GPS 又可以对惯导参数进行标定，提升航位推算精度。

在实际车机导航项目中，传感器配置不同，又会衍生出不同的定位方案，如下表所示。

定位解决方案	硬件成本	定位效果	功能特性								
			信号漂移抑制	无GNSS信号推算	小角度分叉快速匹配	精准起点定位	环岛高精度匹配	精准隧道推算	地下停车场持续定位	上下高架桥识别	进出主辅路识别
GNSS	低	基础	*								
前端融合	最高	良好	★★	★★	*	*	*	★★	*	★	-
车辆模型	较低	良好	★★☆	★★☆	*	★★	★★	★★	★★	-	★
后端融合**	较高	优秀	★★☆	★★☆	★★☆	★★☆	★★☆	★★☆	★★☆	★★☆	★★☆

其中，纯 GNSS 方案无法使用任何传感器融合手段，定位效果最差。前端融合方案实现了惯导和车速的前置融合，可以满足部分 GPS 失效场景的持续定位，但由于累积误差的影响，提升有限。车辆模型方案和后端融合方案都实现了传感器、GPS 与地图数据的完整融合，因此定位效果更好，其中车辆模型方案使用车辆 CAN 总线的传感器数据，而后端融合使用车机安装的惯导传感器，在对传感器选型及安装使用方式进行精细适配之后，可获得最佳的定位效果。

3.2. 传感器融合技术

以后端融合为例，传感器融合算法框架如下。

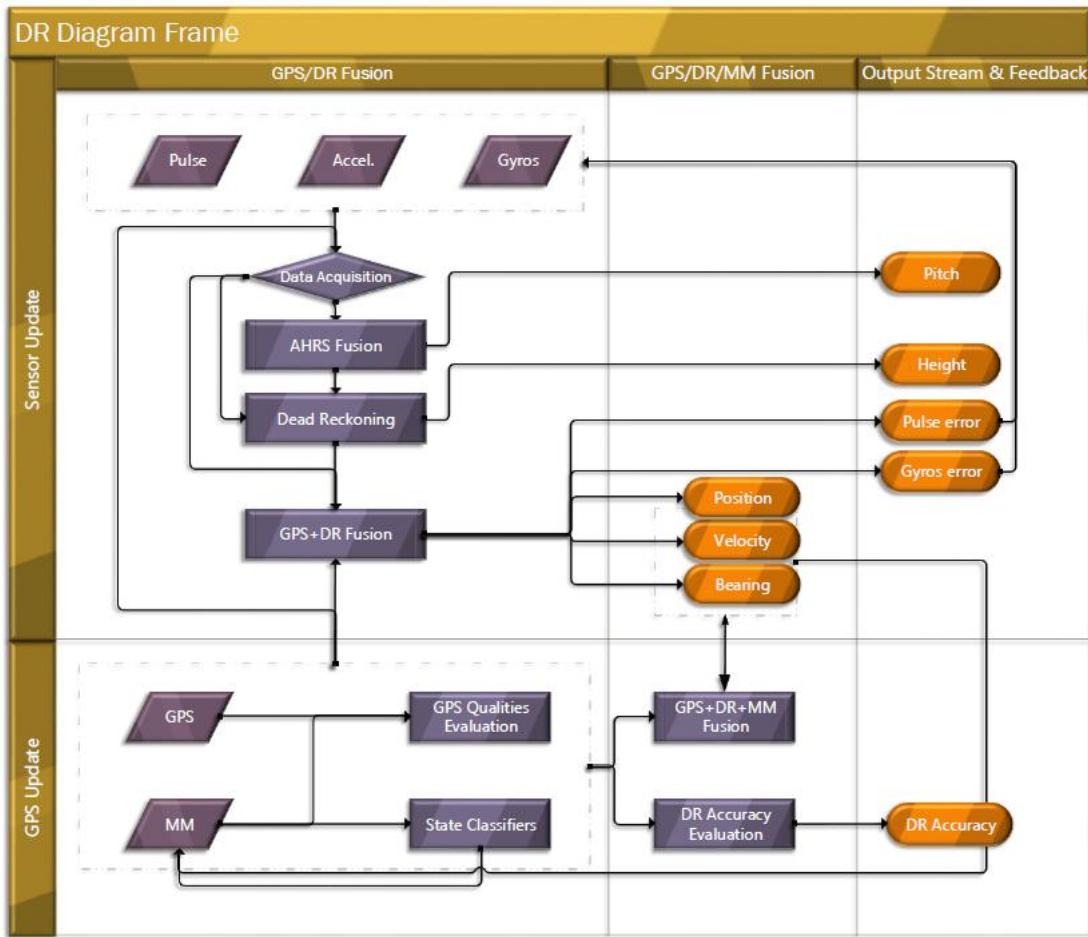


图 7 传感器融合算法框架

融合算法有两个目的：第一，将不同技术的导航信息融合成唯一导航信息，使之可靠性高于未融合前的；第二，估计器件误差（陀螺仪零偏、测速仪尺度误差等）。

融合算法基于 Kalman 滤波实现，其关键在于模型建立和模型参数设置。Kalman 滤波模型由状态转移方程和观测方程构成。状态转移方程表示相邻导航状态之间的转移关系，它通过构建导航误差微分方程实现；模型参数是指状态转

移噪声和观测噪声，观测噪声的设置与 GPS 质量评估模块相关。经 Kalman 滤波处理后，得到导航误差的最优估计。

实现了完整信息融合的传感器融合技术可以在使用低成本传感器条件下达到甚至超过高成本专业惯导设备的定位效果。

下图展示了采用后端融合方案的车机导航定位效果。图中蓝色是 GPS 位置，红色是高精度基准设备的定位轨迹，绿色是车机导航定位轨迹。可以看到，在 GPS 被遮挡的停车场，或 GPS 被干扰的区域，车机导航定位始终可以持续稳定的输出高精度的定位位置，保证了车载导航功能的可靠运行。



图 8 停车场定位效果

图 9 信号干扰区定位效果

4. 高精定位演进

传统的导航定位仅需要解决道路级的定位问题，对定位精度的要求不高。但随着辅助驾驶、自动驾驶等越来越多的应用场景出现，对定位精度的要求也不断提高，如下图。

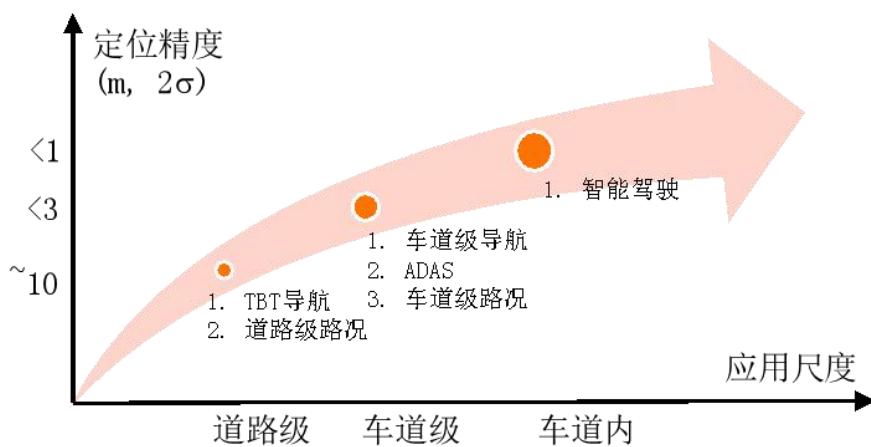


图 10 高精定位演进

对于车道级导航，定位需要能分辨出汽车当前所在的车道，这就要求定位精度达到米甚至亚米级，而对于更先进的智能驾驶应用，为保证安全，需要定位精度达到厘米级。

为达到更高精度的定位能力，需要对现有的定位手段进行升级。一种升级的方法是，对现有的定位输入源直接升级，如普通 GPS 升级为更高精度的 RTK-GPS，低成本 MEMS 惯导升级为高精度惯导，这样可以在基本不改变原有算法框

架的基础上直接获得高精度的定位能力。但缺点是，原有算法的缺陷，如长时间 GPS 丢失场景的累积误差问题依然存在，且成本较高。另一种方法是引入新的定位源，如激光雷达、毫米波雷达、摄像头等。这就需要针对这些新的传感器开发新的融合定位算法。不同的高精定位方案对比如下表。

方案		代表厂商	精度	优势	缺点	成本
绝对	RTD/RTK	千寻	1~3m(RTD), 0.1~1m(RTK)	1. 静态精度高 2. 不受天气影响	1. 场景局限 2. 动态精度下降 3. 依赖地图绝对精度	1000~2000 需额外加装
传感器 地图						
相对	Lidar	Grid Map	Google	<5cm	1. 精度高 2. 数据量较小 2. 可靠性较高	1. 栅格图数据量大 2. 天气影响，成本高 1. 传感器成本高 2. 单独图层
	Lidar/ Depth camera	RoadDNA	TomTom	<15cm (Lat) <50cm (Long)	1. 成本较低 2. 可靠性较高	\$30~60万(Velodyne , 32/64线) \$500(Ultra Puck , 32线，2020目标) \$250(Quanergy, 8线) 自动驾驶功能加装
	Radar	Radar Map	Bosch	<10cm (Lat) <40cm (Long) (+ camera)	1. 分辨率低，无法单独使用 2. 单独图层	2000~N ADAS功能加装
	Camera	HAD Map	Here, Volkswagen	20cm (Lat) 1m (Long)	1. 成本较低 2. 受光线影响 2. 纵向精度不保证	几千元 ADAS功能加装

上面方案中，高德与千寻合作开发了基于千寻 RTK 服务的高精定位解决方案“知途”，实际道路评测在高速及普通道路场景下的定位精度可达到 10cm 以内。该方案不依赖于其他任何传感器或地图数据，具有全域高精度的特点，可作为独立的高精度定位解决方案。



图 11 “知途”高精定位样机

在基于环境特征匹配的相对定位方案中，激光雷达方案是较成熟可靠的，也是自动驾驶早期原型阶段最普遍采用的定位方案。但受制于激光雷达的成本和可靠性问题，量产落地仍存在风险。基于视觉的相对定位方案成本更低，同时受益于近年来视觉算法和计算芯片领域突飞猛进的发展，在当前的量产自动驾驶中已经逐渐成为主流的定位方案。基于高德自身的图像及定位能力建设，将在三个业务方向上进行高精定位的业务实践。

1) 面向 L3 自动驾驶的系统级定位：基于外部输入的视觉语义信息（如 Mobileye 发送的车道线形状、类型等），与高精地图数据（HD Map）匹配，并结合 GPS/RTK 和 IMU 等其他定位源，实时计算车道级高精定位结果，并驱动高精数据播发引擎（EHP）发送高精数据，为自动驾驶功能提供定位及数据服务。

2) 车道级导航定位：基于自研的视觉算法和云端图像定位能力，实现全域覆盖的车道级定位能力，驱动传统道路级导航向车道级导航升级。

3) 用于高精数据众包采集的软硬一体化高精定位：基于自研的低成本视觉+RTK+IMU 硬件，实现基于 vSlam 技术的高精度绝对定位，为高精数据的采集、重建，并最终快速更新迭代提供支撑。

小结

传统导航定位采用 10m 精度的 GPS 定位为基础，针对手机 / 车机不同端，考虑他们独特的运动特征及输入信号配置，设计传感器融合算法、行为判断算法、地图匹配算法，最终满足导航对于全场景道路级定位的精度要求。未来面向半自动、全自动驾驶应用，要求定位精度向车道级甚至厘米级演进，这需要在考虑实际落地场景的基础上进行传感器和算法迭代，这是下一阶段定位技术演进的重要方向。

招聘

阿里巴巴高德地图在线引擎和安全运维中心团队长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

关于卫星定位，你想知道的一切

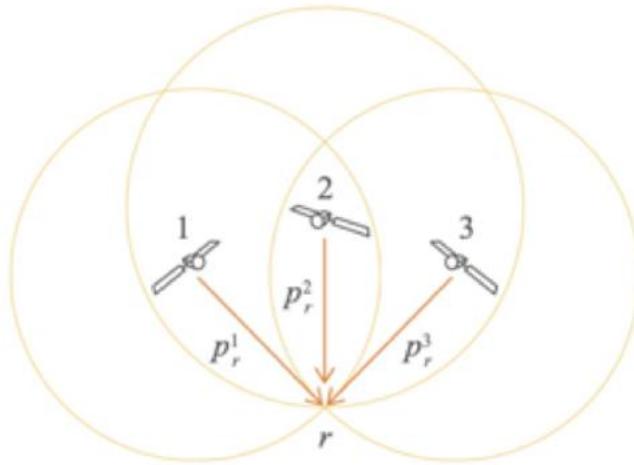
作者：方兴

5G 和北斗，是国之重器。北斗作为卫星定位系统，目前在国际上已处于领先地位，而且已经渗透到我们工作和生活的方方面面。本文将简要介绍卫星定位的原理和应用情况，方便大家对北斗、卫星定位有更多的了解。

卫星定位的原理

卫星定位系统的英文是 Global Navigation Satellite System(GNSS)，虽然直接翻译过来是导航卫星系统，但它真正提供的能力是定位，能定位后，导航就变得相对简单了。

卫星定位的原理，是利用卫星播发时间信号，当设备接收到后，可以根据信号发射时间和本地时间，计算出信号传输时间，再结合光速获得卫星-设备距离。



有了多颗卫星的信号，可以列出一组方程，求解 4 个未知数：设备的三维坐标 $x/y/z$ ，以及本地时间与 GNSS 系统的时间差。

式中的代表卫星 j 的三维坐标，这个坐标可以通过卫星星历计算获得。

$$R^j = \sqrt{(x^j - x)^2 + (y^j - y)^2 + (z^j - z)^2} + c \cdot dt_r, j = 1, 2, \dots, n (n \geq 4).$$

星历是描述卫星运行轨道的一组参数，卫星轨道是一个椭圆，通过几个参数和时间，可以唯一确定卫星的准确位置。

星历的获取有两种方式，一种是卫星直接播发，这种方式的好处是定位过程不依赖卫星信号以外的任何输入，即使没有网络也可以定位成功，但问题是卫星链路带宽很小，要下载完整星历，需要 30 秒左右的时间。

早期的手机和一些车载设备定位过程很慢，就是由于这个原因。

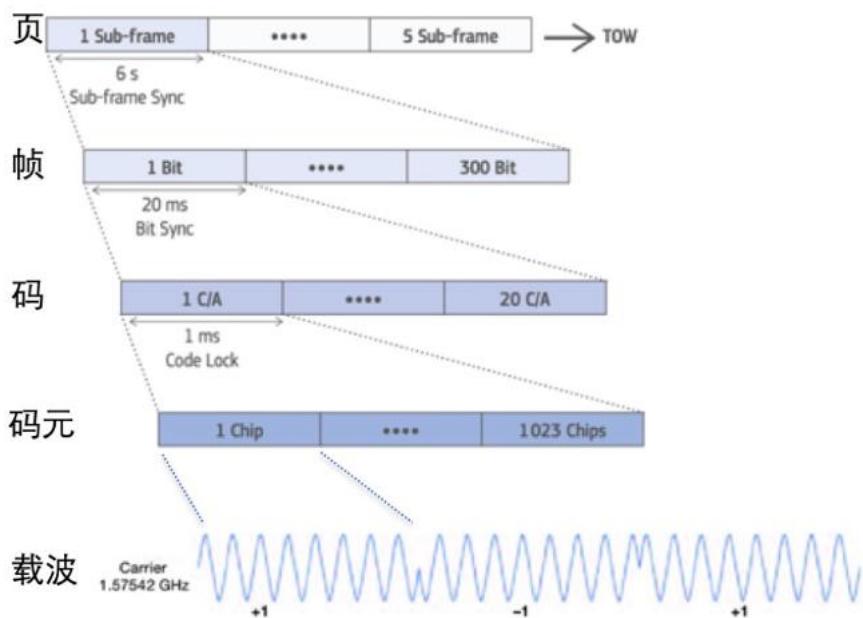
另一种方式，是通过互联网播发，这种方式叫 A-GNSS，具体的传输协议叫 SUPL(Secure User Plane Location)，这种数据一般不对应用层透出，在手机上，操作系统会在底层定时请求 SUPL 数据，然后将获得的星历注入 GNSS 芯片。有了 A-GNSS，设备就可以在秒级获得定位，不需要任何等待过程，目前所有的手机都支持这种方式。A-GNSS 的服务提供商，主要是通信运营商，以及一些定位服务商，比如谷歌、千寻等。

卫星不间断的向地面广播信号，这个信号主要包括以下信息：

- 卫星编号。用于从星历中查找卫星轨道，再结合时间戳获得当前卫星位置
- 当前时间戳。用于获得卫星位置，另一方面计算伪距。伪距是(本地时间-信号发射时间)*光速，之所以叫伪距，是因为本地时间与卫星时间不同步，所以这个距离并不是真正的设备-卫星距离。

- 星历数据。用于计算卫星位置。

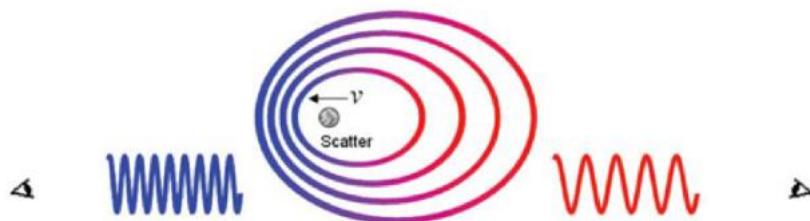
像其他所有的通信技术一样，这些信息也是以报文的形式发送的，以 GPS 为例，卫星会每隔 6 秒发出一个包，而这个包会分解为数据位—CA 码序列—载波波形，通过天线发射到地面。地面设备持续锁定卫星，在解算时，计算每颗卫星当前时刻的时间戳（用最近一次收到的时间戳加上报文偏移量），然后进行位置解算。



载波的频率是 1.5G 左右，波长 20 厘米左右，比移动通信的波长稍长一些，所以信号的穿透性还是比较好的（波长越长，越容易绕开障碍物），可以穿透比较薄的墙壁或屋顶，所以在一些情况下即使无法直接看

到天空，也是能定位的。但是卫星信号是从上往下，在室内很难穿越多层建筑。

卫星定位的另一个特点是可以解算出速度，其依据是多普勒频移原理（与交警用的测速仪原理一样）。当信号源与接收设备存在相对运动时，接收到的信号频率会发生变化。



频率变化量与相对速度存在如下公式：

$$\begin{aligned}-f_d * \lambda &= diff_v + c \cdot dt \\&= (v - v^j) \cdot l^j + c \cdot dt\end{aligned}$$

其中，公式左边是频差和波长， v 是设备运动速度（矢量）， v^j 是卫星运动速度（矢量）， l^j 是卫星的投影方向， dt 是本地设备的频漂速度。只要测量了 4 颗星的频差，就可以解出本地设备的运动速度（与设备姿态无关）。

除了定位和测速，定位卫星还可以完成全球授时（解算过程中获得本地钟差），这也是目前成本最低的高精度授时方法，比绝大部分设备自带的时钟都要准确。

一般而言，伪距测量值精度不如频率测量精度高（伪距定位精度在 10 米左右，而多普勒定速精度可以达到 0.2 米/秒以内，授时精度在 20ns），原因是伪距测量容易受到多种路径误差影响（后面会介绍），而频率测量的干扰因素少很多。

卫星定位发展历程

最早的卫星定位系统，是美国在 1960 年代开发的子午仪系统，后续在 70 年代开发出了 GPS 定位系统，目前的 GPS 系统由 24 颗卫星构成。除了 GPS，世界多国也开发出了自己的卫星定位系统，主要的有中国的北斗系统、欧盟的伽利略系统、俄罗斯的格洛纳兹系统，此外日本和印度在开发区域定位系统。

	国家	卫星数	状态	特点
GPS	美国	24	已建成	建成时间最久，用户渗透率最高
北斗 Beidou/BDS	中国	30	已建成	亚太地区覆盖好，支持短报文通信
格洛纳兹 Glonass	俄罗斯	24	已建成	高纬度地区覆盖好
伽利略 Galileo	欧盟	27	已建成	多频段、多业务
QZSS	日本	4	已建成	对GPS扩增，播发厘米级修正信息，仅针对日本周边地区
IRNSS	印度	7	建设中	仅针对印度周边地区

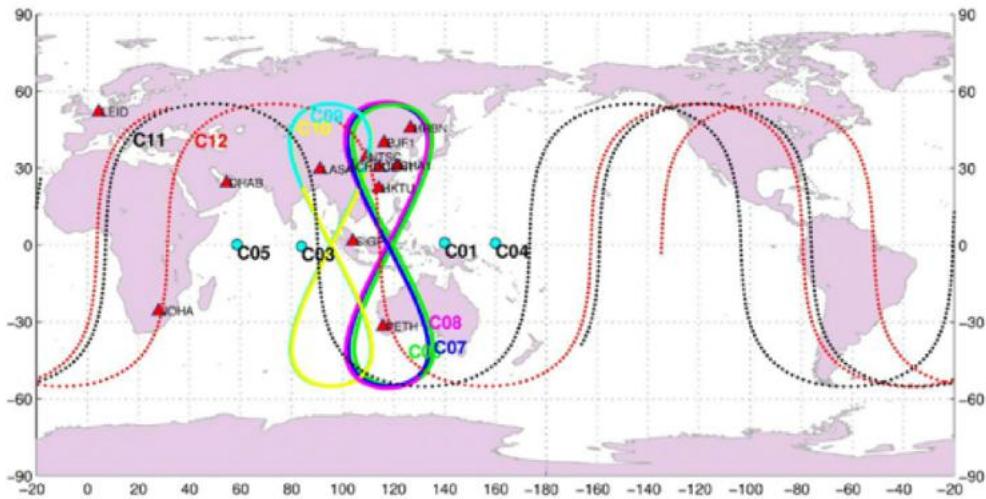
除了天上的卫星，各定位系统还需要地面站对卫星的运行进行监测，包括健康度、轨道参数（计算完成后要注入卫星实现全球播发）、信号质量等，另外还需要对卫星进行控制。

各种卫星定位系统使用的技术类似，大多采用中轨道卫星（MEO，卫星高度2万公里），少数采用了地球同步轨道（GEO，卫星高度4万公里）和地球倾斜同步轨道（IGSO）。同时，信号播发大多采用CDMA技术，实现在同一个频率上传输多颗卫星的信号。为了让地面设备能够较好的接收来自几万公里外的信号，信号的数据速率都比较低，比如GPS L1频段的数据传输速率只有50字节/s，根据香农定理， $C=B\log_2(1+S/N)$ ，在频率带宽B固定的情况下，随着传输速率C的降低，接收端在信噪比(S/N)比较低的

时候也可以解出正确的信号，有利于持续的锁定、跟踪卫星信号。

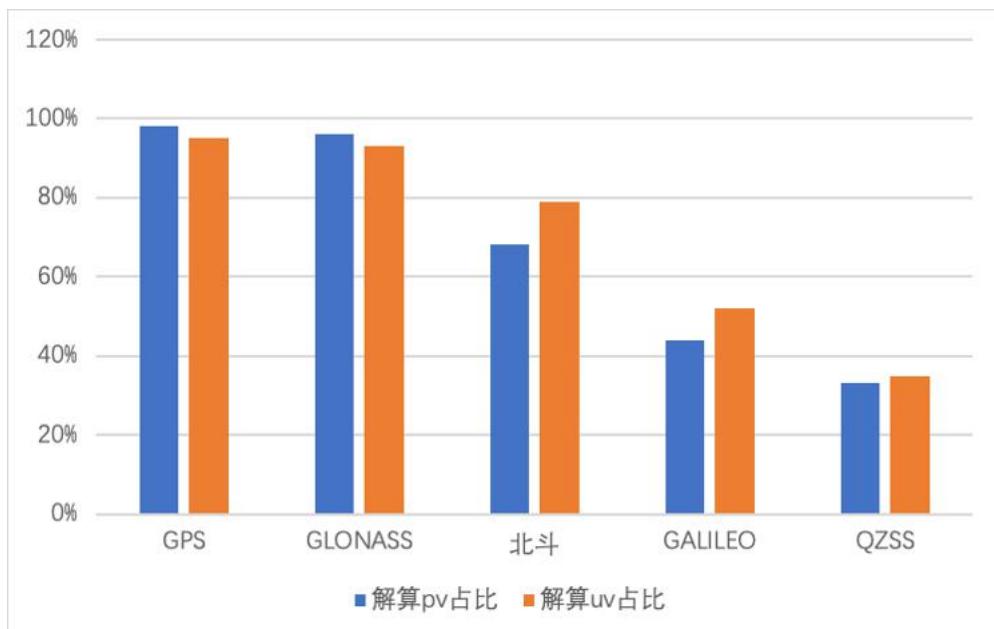
与其他定位系统相比，北斗的特点主要有：

- 亚太地区覆盖好。北斗系统由 3 颗地球同步卫星、3 颗地球倾斜轨道同步卫星和 24 颗中轨道卫星构成，与 GPS 相比，北斗有 6 颗星持续覆盖亚太地区，极大提升了亚太地区可见卫星的颗数，一方面提高定位成功率，另一方面也能提升精度（改善了 GDOP，减少了误差）。
- 北斗的同步卫星可用来进行通信，地面设备可以将短报文发送到卫星（只用 GEO 卫星支持短报文）上，然后转发给目标终端，这种通信是免费的，但是需要专门的天线和设备（需要将信号发射到 4 万公里远的地方，普通手机肯定是不行的）。



多个卫星定位系统的信号同时被收到时，所有的卫星可以一同参与解算（每增加一个系统，只需增加一个新的参数，即这个系统相对于 GPS 系统的时间差），使得定位精度可以获得提升。目前手机上无法选择参与定位的星座或者卫星，所以我们无法指定只用北斗或者不用 GPS 定位。

我们对比了手机端 GNSS 定位时，使用不同系统的占比，可以看出 GPS 和格洛纳兹由于发展的比较早，在手机芯片侧的渗透率比较高，因此被使用的比例也最高，其次就是北斗。

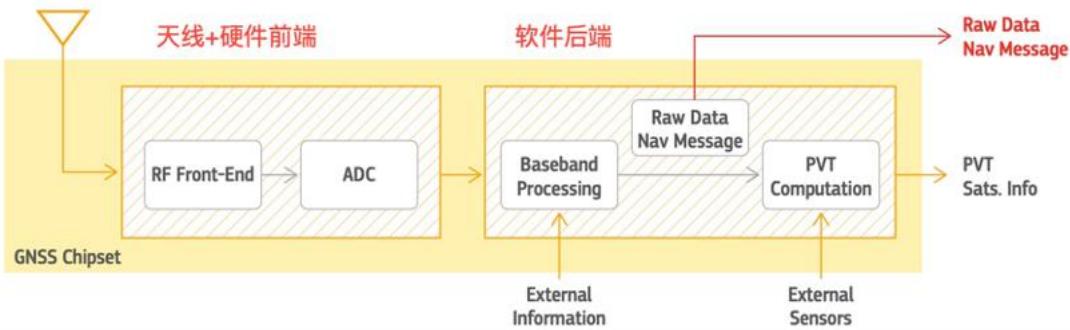


按参与定位的卫星颗数统计，北斗排在第二位，仅次于 GPS。

	锁定卫星占比	锁定卫星数	解算卫星占比	解算卫星数
GPS	34%	8.6	38%	7.8
北斗	26%	6.6	24%	4.8
GLONASS	22%	5.7	23%	4.8
GALILEO	10%	2.6	11%	2.2
QZSS	7%	1.7	4%	0.9

因为各系统技术类似，其定位精度也是类似的，北斗也不例外，水平定位误差一般在 10 米以内。垂直定位精度一般会差一些，主要是由于卫星都分布在设备的一侧，垂直方向上的误差难以修正。

卫星定位接收机构成



卫星定位接收机的原理图如上图所示，主要的模块包括：

1. 天线

用于接收卫星信号。由于卫星信号微弱，天线当然是越大越好，但是由于接收机需要移动，天线尺寸受到制约。天线的主要作用是放大信号和抑制多径，主要的类型有以下几种：



左边的是比较常见的天线，内部是陶瓷天线，外部带磁铁，可以吸附在车顶；中间的是专业天线，旁边带扼流圈，可以抑制来自四周和地面反射的信号，只接收从天顶方向来的信号，这种天线的效果最好，一般用于专业研究和高精测绘；右侧是手机天线，长度只有几厘米，效果最差。

卫星信号的电磁波是圆极化的（传播时在垂直于传播方向的一个平面上波动），因此，采用圆极化天线（如平面的陶瓷天线）接收效果最好。但手机上天线尺寸太小，只能采用线极化天线，信号捕获能力大幅下降，再加上缺乏信号屏蔽（扼流圈），极易受到多径效应以及其他信号干扰。

2. 射频前端

这个模块主要是将原始信号进行下变频、功率放大以及滤波，提取真正有用的信号，便于解码处理。

3. 基带处理

这个模块是对卫星信号进行解码，获得卫星报文。每颗卫星的信号需要一个单独的通道进行处理，如果有 100 颗卫星，2 个频段，那可能需要 200 个通道才能有效处理这些信息。通道数越多，可以获得的卫星观测值也就越丰富，定位精度也就越高。

解码的过程，分为搜索—锁定—跟踪三步，首先生成每颗卫星的伪码，然后与信号进行自相关操作，相关度达到一定程度就可以锁定卫星，然后进行码锁定、位同步、帧同步，最终提取出报文。这个过程要持续进

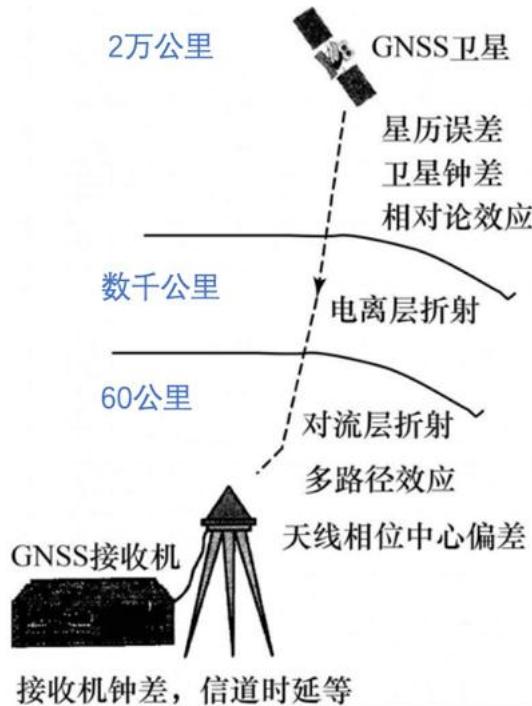
行，因为多普勒效应，信号的频率会不断变化，所以本地生成的伪码也要不断变换频率去适配卫星的变化。一旦失去锁定，就会丢失信号，也就无法定位了。

4. PVT 解算

PVT 包括 Position, Velocity 和 Time。这一步是真正进行定位的步骤，是利用基带解码获得的报文，提取出时间戳、星历等信息，代入公式进行计算，然后将计算结果输出给应用程序。

定位误差来源与精度提升

卫星定位虽然已经很准确了，但是在某些场景下，还是无法满足需求，比如，打车的时候定位点离车辆有一定距离、步导的时候难以区分方向甚至会定位到马路对面、静止的时候定位点总数飘来飘去、室内的时候定位点乱飘。这需要从卫星信号的发射、传输、接收过程来解释。



卫星信号从发射到被设备接收，需要经过大气层，其中，大气电离层有数千公里厚，这部分大气非常稀薄，但是存在大量被电离的电子，这部分电子会让电磁波变慢一点，从而产生延迟。在对流层，也会产生一定的延迟。在地表附近，由于各种建筑、山体、水面的影响，卫星信号可能被反射或折射（多径效应），产生延迟。

在卫星信号发射侧和接收侧，也有很多系统相关的误差，比如时钟偏差、处理延迟等，这些延迟加上传输延迟，使得卫星信号的传输时间，并不是准确的等于物理距离/光速，另一方面，卫星的星历也有误差，

卫星位置和真实位置存在偏差，最终造成了定位结果产生偏差。

要提升定位精度，需要想办法消除这些误差，主要有以下几种方案。

双频 GNSS

不同频率的电磁波通过电离层时会有不同的延迟，人们发现，对两个或多个频率的观测值进行线性组合，可以消除电离层误差，从而能提升精度。这就是双频 GNSS 定位的原理。小米 8 是业界第一款支持双频 GNSS 定位的手机，后续各大厂商均进行了跟进，一些高端手机均采用双频定位。消除电离层误差后，定位精度可以提升到 5 米以内。

地基/星基增强

星历误差、卫星时钟误差、甚至是电离层和对流层误差都是可以观测或建模的，一旦计算出了实时的误差值，就可以通过一个单独的通道进行播发，接收设备在定位过程中使用这些修正项，就可以提升定位精度。播发的通道一般有两种，一种是直接通过卫星播发，称为 SBAS(Satellite-Based Augmentation System)，好

处是覆盖广，但设备需要增加额外的信号接收通道；另一种是地基增强，比如通过互联网，这需要设备具备联网能力。

这些增强方式对于精度提升是有限的，还是有很多误差项无法消除，比如电离层误差。

高精定位—差分定位(RTK)

RTK 是 Real - time kinematic 的缩写，是一种差分定位。其原理是利用一个参考站提供基准观测值，然后用设备的观测值与基准站的观测值进行差分，差分后可以消掉星历误差、卫星钟差、电离层误差，再进行星间差分后可以进一步消除掉设备的钟差，最终可以算出设备相对基准站的相对坐标，如果基准站位置已知，就可以完成准确的绝对坐标，精度可以达到厘米级甚至毫米级。

RTK 能提升精度的另一个原因是引入了载波相位观测，相比伪距观测值，载波相位观测值的误差更小。

使用 RTK，需要在附近 20km 内有参考站（距离太远，电离层误差不一样，做差分无法完全消除误差），同

时需要持续不断的获得参考站的观测数据（一般通过互联网传输，使用 RTCM 协议），因此相对普通的定位，RTK 定位成本较高，但对于一些对精度要求很高的场景，比如车道级定位、自动驾驶等，是必不可少的。

RTK 服务一般由专业服务商提供，如千寻、六分，这些服务商在全国范围内部署了数千个基准站，持续对订阅用户播发数据。

高精定位—精密单点定位(PPP)

RTK 需要布设密集的参考站，有没有办法不依赖参考站？PPP(precise point positioning)就是一种方法，它的原理是对每一种误差进行准确建模，最终求解出卫星和设备之间的准确距离。为了确定准确的误差，PPP 定位时需要不断的迭代内部参数，而且，一些卫星的误差只有当卫星位置变化后才能体现出来，所以 PPP 需要比较长的收敛时间，一般需要 30 分钟才能收敛到理想的精度，如何更快的收敛是目前学术界的一个研究热点。

组合定位

卫星定位的一个最大问题，就是丢失卫星信号后如何定位，这就需要其他定位方式来补充。组合定位是利用卫星信号和其他定位技术，比如惯性导航，来完成定位，二者相互配合。最简单的一个例子，就是卫星定位是有一个最高频率的，一般最多是 10Hz，在两次定位之间，可用惯导来进行位置推算，获得更高频率的位置输出。而组合导航最重要的作用，是提升精度，比如，利用卡尔曼滤波方法，用惯导计算推算位置，用卫星定位提供观测量，对推算位置进行修正，这可以让定位结果更加平滑，而且可以对异常的卫星观测量进行过滤或降权。

手机上的卫星定位

在移动互联网出现以前，卫星定位终端是一个很专业的领域，只有测绘、军事等领域会应用这种技术，定位需要使用专用的接收机，比如 Trimble、ublox 等。随着智能手机将卫星定位芯片集成，卫星定位的应用得到爆发式增长，终端数量一下子提升到几十亿量级，也产生了海量的位置数据。

手机上的卫星定位与专业接收机，还是存在比较大的差异，主要体现在：

- 手机受限于尺寸，天线比较小，对原始信号的捕获、锁定、去噪能力都比较差，造成接收到的信号质量天然不如专业接收机。
- 手机上芯片成本比较低，支持的通道数比较有限，一次定位能够解码的卫星数量和系统数量都比较少，主要是单频，少数是双频，没有三频。
- 手机上对功耗、性能开销的要求比较高，不能花费大量资源在定位上，解算算法的复杂度比较低，效果也比较有限，精度比较差。

苹果手机

苹果手机的定位能力是完全封闭的，对外只透出定位结果，外部基本无法拿到任何定位相关的原始观测量，比如卫星数量、类型等。好消息是，iPhone12 终于开始支持北斗了。从苹果的 API 上，外界甚至无法区分定位结果到底是来自卫星定位还是网络定位（目前仅能通过速度的符号来判断，但苹果对此没有任何承诺）。

所以，基于苹果手机，我们基本无法做出优化，苹果手机上高德地图的定位点都是 iOS 底层直接提供的。

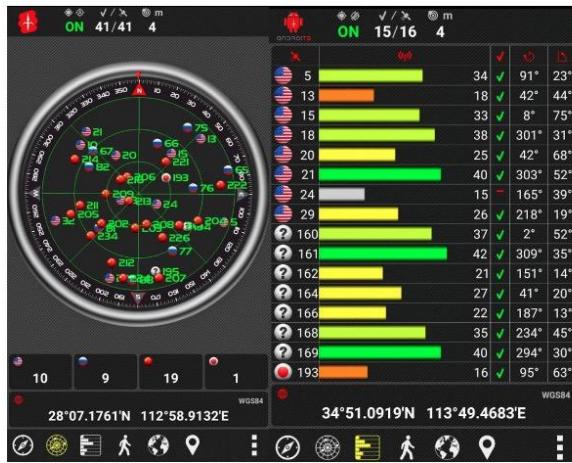
安卓手机

安卓手机比苹果手机开放的多，在定位能力方面提供了一系列 API：

- 可以单独获取卫星定位结果或网络定位结果，也可以同时进行两种定位。
- 提供了 NMEA 格式（一种卫星定位结果的规范化表达）的结果数据，可以获取每颗卫星的 ID、类型、信号强度，以及 xDOP 等细粒度的误差描述。
- 提供了 GnssStatus 来描述每颗卫星的状态，内容比 NMEA 更全面。
- 提供了 GnssMeasurement 来描述原始观测量，包括伪距测量值、载波相位测量值、卫星锁定状态等。
- 提供了 GnssClock 描述本地时钟的状态。

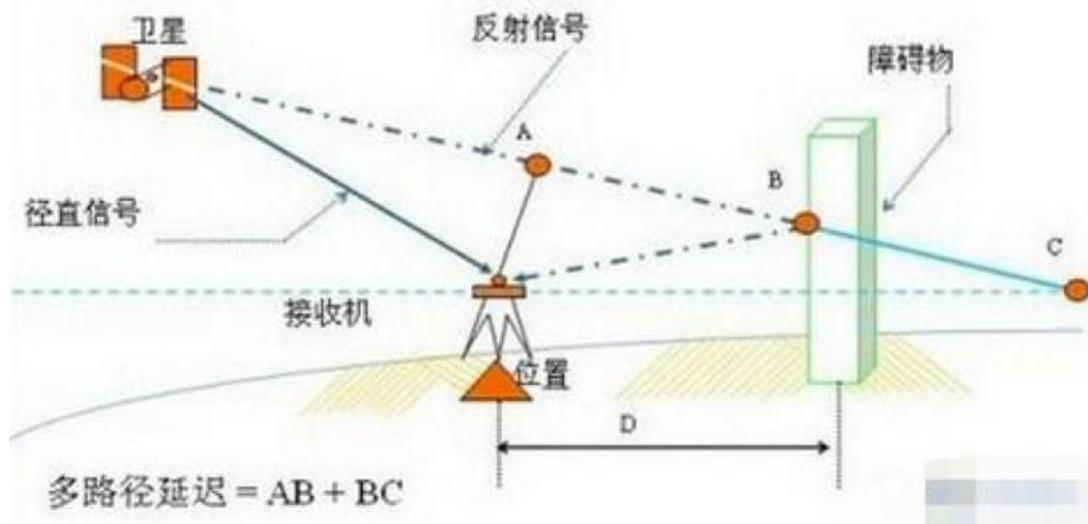
- 提供了 GnssNavigation 透出最原始的未解码报文。

有了这些信息，通过一些 App 就可以实时看到当前的卫星状态，例如 Androits gps test，GPSTest 等。



另外，我们还可以进行卫星定位的软解算，对卫星定位结果进行修正，甚至替代。我们主要尝试解决两类问题：

- 定位不准：对卫星定位结果进行质量判断，识别其中的大误差点，优化精度，或者优化精度半径，使下游使用定位点的时候，能够差异化的处理。



定位不准的原因，主要是来自卫星信号中含有误差，而影响最严重，也是最难抑制的，就是多径造成的影响。

另一类定位不准的问题，是系统将其他定位结果伪造为卫星定位结果。比如，将网络点冒充为卫星定位点。

- 无法定位：系统不输出定位结果时，尝试进行软件解算。

无法定位最主要的原因，是信号接收条件不好，比如室内遮挡、高架遮挡、高楼遮挡。在开阔地带无法定位，一般是设备 Bug，重启设备后一般都能解决。

卫星定位未来展望

随着移动用户量持续增长，以及物联网的大范围普及，卫星定位技术还会持续快速发展。

在卫星侧，将出现低轨定位卫星（距地面几百公里）。传统上的定位卫星由于要覆盖较大的地理范围，高度一般都比较高，运行在中轨轨道上。随着火箭发射技术的革命，卫星发射成本急剧下降，向太空发射大批量低成本卫星的方案成为可能。比如 spacex 已经发射了上千颗“星链”卫星。

低轨卫星进行定位有几个好处：

- 距离近，信号更强，设备侧接收到的卫星信号更好。
- 可以传输更多的数据，比如各种修正数据。
- 位于电离层底部，电离层误差小。
- 卫星仰角变化快，PPP 定位可以更快收敛。

在设备侧，高精定位将大范围普及，华为 P40 是首个支持 RTK 的智能手机，可以做到 0.5 米的精度。高通也即将发布支持 RTK 的移动芯片，在 2021 年上半年，更多支持 RTK 的智能手机将会上市。

在应用侧，高精定位的应用场景会不断涌现，现在的一些典型应用场景包括：

- 传统测绘
- 精准农业，机械化自动化种植和收割
- 车道级导航和自动驾驶
- 共享单车的精准停放
- 无人机导航

参考资料

北斗官方网站 <http://www.beidou.gov.cn/>

GPS 官方网站 <https://www.gps.gov/>

伽利略官方网站 <https://www.gsa.europa.eu/>

业内首发车道级导航背后——详解高精定位技术演进

作者：子路

导读

10月30日，华为联合高德、千寻发布了业内首家面向手机用户的车道级导航应用。在这背后是高精度定位技术不断演进发展，最终走向成熟量产的过程。本文将结合高德地图在车道级导航及自动驾驶等领域的工作，分享我们对于高精度定位技术演进的思考，以及在高精定位实际落地应用中的一些实践。



一、高德定位技术概述

定位技术是支撑高德地图的导航、交通等核心业务的关键基础技术，他的主要任务是确定物体（通常是人或车）在一个相对固定的坐标系中的位置和姿态。我们用手机高德地图作为例子来说明都有哪些技术在实际应用场景中发挥作用。

通常手机的基础定位能力是由手机的 GNSS 芯片提供的，它为我们在室外的绝大部分场景下提供了 5~10 米的定位精度。但是在卫星信号不好的时候，定位可能会漂移，我们需要识别出这种情况。

另外，当信号受到干扰的时候，位置可能出现有规律的偏移，我们也要识别出干扰，并且在可能的情况下重新解算出正确的位置。当 GNSS 定位不准的时候，想要持续定位，可以利用传感器识别出步行/驾车状态，再进行航位推算（PDR、VDR）。当进入室内卫星信号丢失了，常用的定位方法是根据手机扫描的基站和 Wifi、蓝牙等信号做网络定位。

上面这些技术提供了基础的位置坐标，而在导航过程中，我们更关心的是行驶在哪条路上，有没有偏航，

距离下个路口有多远，想得到这些信息就需要用到地图匹配技术。在一些非常复杂的场景下，比如高架桥、主辅路，判断道路变得非常困难，这时候还需要用到一些专门的识别模型来解决匹配问题。

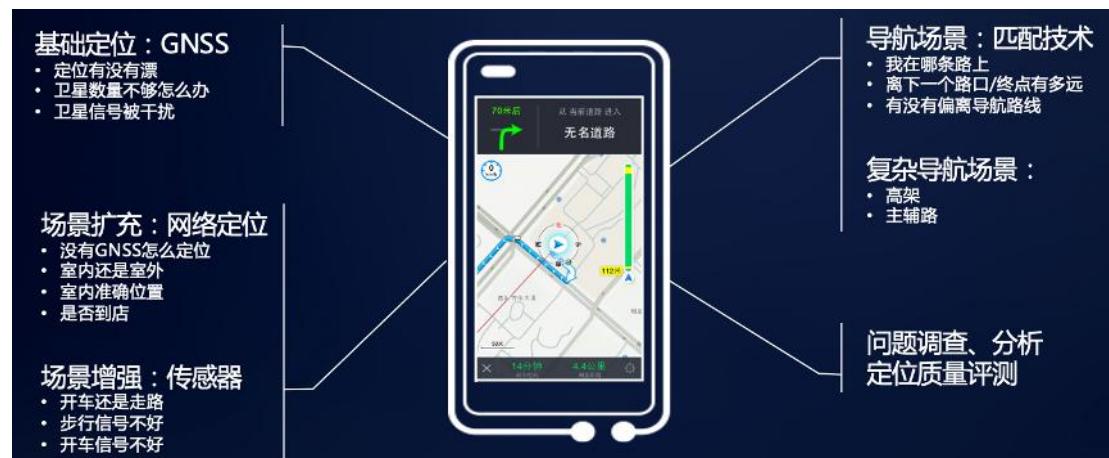


图 1 手机高德地图中的定位技术

上面提到的只是具体一个业务场景中的部分技术，下面展示了高德定位技术的一个更完整的大图。总的来讲，我们是通过构建一套“云+端+数据”的完整的技术体系，并建设质量迭代系统保证各技术模块的持续更新迭代，来支撑众多的定位业务应用。



图 2 高德定位技术大图

二、定位技术如何向高精演进

回到图 1，我们可以看到这里提到的定位技术尽管扩展了用户场景，但在定位精度方面并没有明显提升。如果要实现前面提到的车道级导航，乃至更加智能化的自动驾驶，就要求定位精度显著提高到亚米级，甚至厘米级。那么怎么做到这一点？我们下面会从技术的角度做一个分析梳理。

首先，我们把图 2 中涉及的，以及其他更广泛的定位技术，按照定位原理分成三类，分别是航位推算、几何定位和特征定位。针对不同的定位类型分析影响精度的因素，总结提升精度的方法，最终期望找到实现高精定位的技术路径。

方案	定位类型	典型应用	外部依赖	精度影响因素	局限性	提升精度手段	场景
航位推算	相对位置	IMU-DR	无	器件精度	无法确定绝对位置	提高器件精度	全场景
		汽车模型 DR PDR		模型精度 推算时间/距离	提高误差补偿模型精度		
几何定位	绝对位置	GNSS UWB、蓝牙、5G	卫星 无线电设施	定位设施的数量、分布 几何测量误差	依赖定位设施 受信号遮挡、反射等影响	多频、多星座、差分技术 精细时间测量	开阔天空 基站布设范围内
特征定位	绝对位置 (特征匹配)	Wifi 指纹 Lidar 点云匹配 图像匹配	特征地图	特征数量、质量、区分度	依赖地图 受环境影响大	场景特征丰富 提高数据精度	数据采集范围内
	相对位置 (SLAM)	Lidar-SLAM 视觉 SLAM			无法确定绝对位置 受环境影响大	场景特征丰富	大部分场景 (除雨雪天等)

表 1 不同定位技术的分析汇总

1. 航位推算

航位推算的基本原理是从上一时刻位置出发，根据运动方向和距离推算下一时刻的位置。显然这种定位方法需要一个已知的起始位置，否则就只能得到相对的位置变化。同时在推算过程中定位误差会不断累积增大，所以影响精度的直接因素就是推算时间或距离。

此外，推算精度还受到每个时刻的测量精度的影响，对于惯性传感器，这就直接由惯性器件的精度决定。例如，精度最高的战略级惯导，随时间发散的位置误差可达 $30m/hr$ ，相比之下，战术级惯导精度要差 3 个数量级，而我们常用的消费级微机械（MEMS）惯导精度比战术级要再差 1~2 个数量级。

除了器件精度，推算过程中的模型精度也会影响定位精度，这包含两个方面：一是对器件测量误差的补偿模型，二是对计算误差的补偿模型。通常只有当器件本身的精度足够高时，才需要考虑更精确的补偿模型。

2. 几何定位

几何定位是对已知位置的参考设备进行测距或者测角，再通过几何计算确定自身位置。根据几何计算的方式不同，包括 RSS（信号强度）、TOA（到达时间）、AOA（到达角）、TDOA（到达时间差）等多种方法。对于测角定位方法，一个小的角度测量误差可能在距离定位设施较远的地方产生很大的位置误差，因此这种方法（如采用 AOA 方法的蓝牙定位）的定位精度通常受范围限制。在测距方法中，采用时间到达模型（如采用 TOA 方法的 GNSS 定位，采用 TDOA 方法的 UWB 定位）比信号强度模型（如采用 RSS 方法的蓝牙和 Wifi 定位）更有可能获得较高的定位精度。但在实际应用中，最终的定位精度受到距离测量精度的影响，尤其是在卫星定位这一类长距离信号传播的场景中，如何消除信号传播路径上的测量误差，就成为决定定位精度的关键。此外，几何定位的精度也受到定位设

施数量和分布的影响，同时观测的设施越多、分布越均匀，精度通常也越高。

3. 特征定位

特征定位方法首先获取周围环境的若干特征，如基站 id、Wifi 指纹、地磁场、图像、Lidar 点云等。接下来有两种处理方式，一种是把接收到的特征和事先采集的特征地图进行匹配，确定在特征地图中的位置；另一种是没有特征地图，通过对比前后帧的特征变化来进行位置姿态推算（即 SLAM 技术），达到类似航位推算的相对定位效果。显然，影响特征定位精度的直接因素是特征的数量、质量和区分度。

因此，采用信号指纹特征（如 Wifi 指纹）的定位方法因为指纹的稀疏性通常精度有限。基于环境感知特征的定位方法在采集的特征足够密集的情况下（如高线数 Lidar，中高分辨率图像等）可以达到很高的精度，但是在实际应用中受环境影响较大，当环境特征单一的时候（如天空、雪天）精度就会下降甚至无法定位。另外，特征地图匹配方法的定位精度也受到特征地图精度的限制，特征推算方法（如视觉 SLAM）的定位误差会随距离累积，具有类似航位推算的发散效果。

综合上面的分析，可以筛选出具备高精度定位能力的技术选型。完整的高精定位方案首先需要至少一项高精度绝对定位技术，如几何定位中的 GNSS 定位，特征定位中的 Lidar 点云匹配等；其次，针对这些方案中的场景限制，辅助相对定位手段，如 DR、SLAM 技术等，进行补充。

		绝对定位		相对定位	
		几何定位	特征匹配	航位推算	SLAM
室外	GNSS、5G	Lidar 点云匹配			
		图像匹配	IMU-DR	Lidar-SLAM	
室内	UWB、5G	Lidar 点云匹配	汽车模型 DR	视觉 SLAM	
		图像匹配			

表 2 高精定位的技术路线

三、高精定位业务场景与解决方案

上面从技术的角度分析了高精定位可能的实现路径，接下来我们从高德的具体业务场景出发，看一看这些技术在实际业务中是如何落地的。

1. 实际业务场景需要什么样的高精定位能力

出行场景是高德地图的核心业务场景。以驾车出行为例，传统的出行应用是以 TBT(Turn-by-turn) 导航为代表的道路级应用，它对于定位精度的要求在 10m 左右。更精细的导航体验，如车道级导航，需要将汽车定位到车道上，这就需要位置精度达到 1 米以内。对于智

能驾驶场景，为了保证机器自动驾驶的安全性，对定位精度的要求更高，一般在道路横向的精度需要小于20厘米。

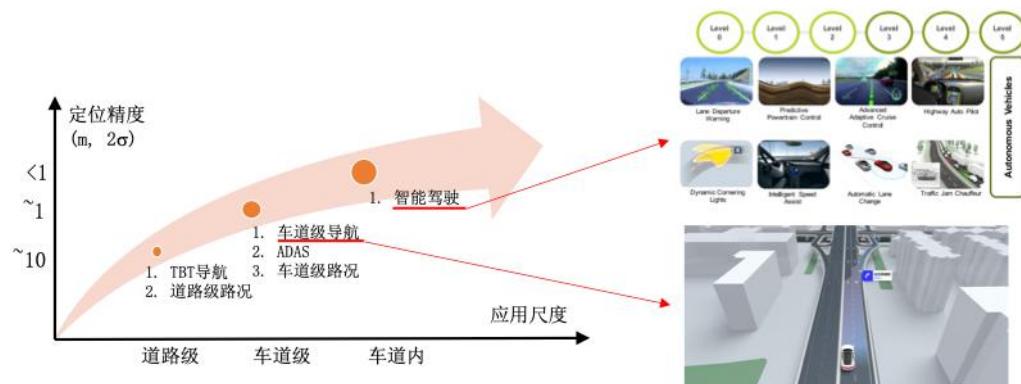


图 3 驾车出行场景的高精应用

除了对精度的要求，不同业务场景对于高精定位能力还提出了其他维度的要求。

1) 可靠性（或完整性）：这主要用来衡量定位系统是不是有能力发现可能的错误，这对于依赖定位进行智能驾驶的应用尤其重要。比如，系统需要对当前的位置给出一个精度半径，当实际的位置精度小于这个半径时，系统就是可靠的。所以，对于可靠性要求比较高的应用，这个半径的估计通常是保守的。

2) 可用性：如果系统能够准确的判断当前的定位精度满足导航、自动驾驶等业务的要求，这时系统就是可

用的。显然，可用性要求精度半径的估计不能太大，否则系统会频繁认为定位不可靠，导致相关的功能无法使用。

3) 算力：作为传统导航应用的升级，车道级导航对于算力的敏感度较高，通常要求满足目前的手机、车机导航的算力限制。智能驾驶根据不同的智能程度分级（SAE Level1~Level5），对算力的要求也不同。通常低等级智能驾驶搭载的传统汽车电气架构无法提供更多的算力资源，而高等级智能驾驶使用的集中计算单元可以提供的算力资源更丰富。

除此之外还有许多与实际应用相关的需求，比如输出定位结果的时间稳定性，定位能够覆盖的场景范围等。

总结一下，车道级导航需要的核心能力是识别当前车道，这一般要求定位精度小于1米，同时作为导航应用，需要在传统导航的基础上提高道路匹配的准确率。低等级智能驾驶（L3以下）要求车道识别和道路匹配（这主要是为了保证智能驾驶只在允许的道路范围内打开，如高速路）的准确率更高，更进一步要求横向

位置精度达到 20 厘米，另外对系统的可靠性要求也更高。

上面的两类应用是我们目前最主要的业务场景，它们都要求在较低算力条件下实现高精定位功能。为了满足这些业务的需要，我们开发了轻量级的一体化融合定位引擎。

2. 轻量级的一体化融合定位解决方案

这里我们使用的定位技术主要包括：RTK-GNSS 技术，图像语义匹配，IMU 或汽车模型 DR 技术等。其中，图像语义匹配使用视觉设备（通常为车上装载的智能摄像头，如 mobileye 等）识别的车道线、地面标志等信息作为输入，与高精地图数据进行匹配定位，这一过程处理的语义要素有限，所以算力消耗不大。至于其他技术在传统导航定位中已经涉及，所以方案整体的算力消耗可以控制在和普通导航同一量级。

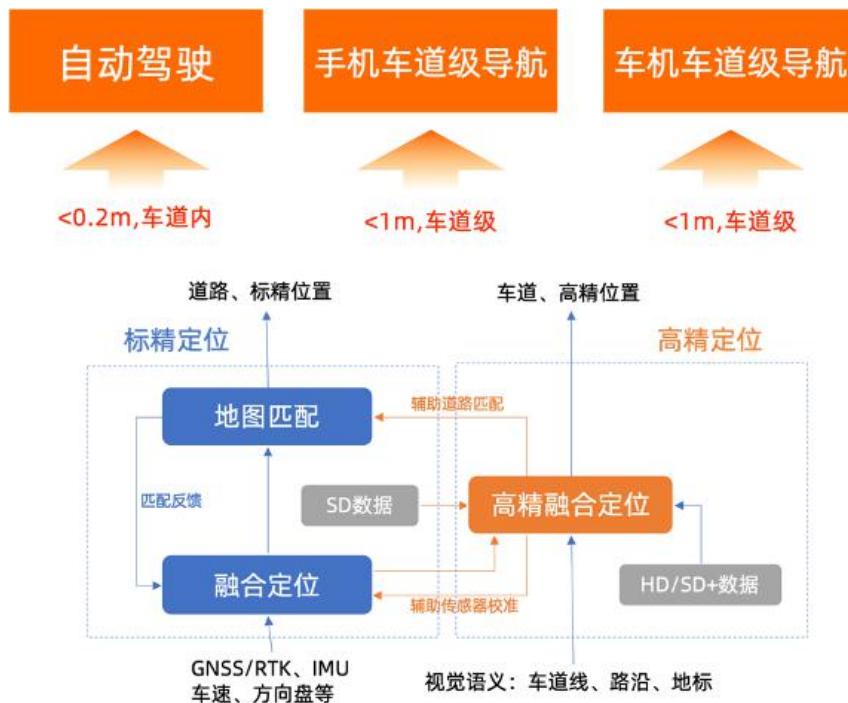


图 4 一体化融合定位解决方案

上面的一体化方案框架可以接收全部或部分定位信号输入，同时提供道路级和车道级的定位结果，保证了在全场景下定位的连续性。在具体应用中落地时，方案的形式又有所不同。

对于智能驾驶应用，高可靠性要求系统具备更多的冗余信息来容错，因此通常需要 RTK-GNSS、IMU、视觉语义等所有信息输入，在收到这些信息之后需要解决两个问题：1) 如何判断输入信号哪些是可靠的，2) 如何充分利用所有信息进行融合定位。

这里我们采用了基于多假设的粒子滤波作为高精融合定位的基本算法，并且设计实现了下面的算法改进：

- 根据假设特征缩减粒子维度，减少计算量；
- 采用分层归一化解决微小系统误差导致的粒子退化问题；
- 基于上下文的后验置信度计算，解决输入信号置信度缺失或错误；
- 基于信号窗口的输入信号延迟和乱序处理；
- 利用高精卫星定位和高精地图数据辅助传感器校准，提升 DR 能力。

目前该算法已经在一款 L3 级别智能驾驶车型上落地，正在进行大规模实验验证。

对于车道级导航应用，由于成本和使用条件的限制，通常无法获取所有输入信息。但是根据表 2，我们至少需要 RTK-GNSS 或者视觉语义其中之一进行高精度

绝对定位。在手机终端上，比较便捷的实现方案是升级手机 GNSS 芯片支持 RTK 差分来提升精度。在前面华为手机上首发的车道级导航应用就是综合了华为和千寻的相关技术和服务来实现高精度的绝对定位。

对于车机，可以将车上用于低等级智能驾驶功能的智能摄像头信息接入导航，直接升级车道级导航功能。

在这个应用场景下的高精融合定位仍然是以上面的粒子滤波算法作为基础，但需要在算法和工程上灵活适配各种不同的输入信号类型和信号特征。

另外，针对导航场景的特点，一体化融合定位还根据车道级定位结果，比如汽车是不是在出口车道上，来辅助判断主辅路、高架桥等复杂路况条件下的偏航情况，提升用户导航的整体体验。

目前基于华为手机的车道级导航要发布上线，车机的车道级项目也正在实施落地，不久就会和大家见面。

3. 面向复杂场景的多元紧耦合 SLAM 技术

上面的轻量级融合定位方案可以解决室外大部分遮挡少、语义清晰场景的高精度定位问题，但是对于更复

杂的场景，比如室内、停车场、城市复杂路口等，高精度 GNSS 可能无效，视觉语义信息也较少，这时候就需要融合更丰富的定位手段。在高等级智能驾驶（L4 以上）中通常使用 Lidar 点云匹配，和 / 或高精度惯导 DR 来保证持续的高精度定位，但 Lidar 和高精惯导的成本很高，大规模应用受限。

SLAM 技术可以用低成本视觉传感器，持续推算高精度位置和姿态，可以作为低成本高精定位的有效手段。相比上面的轻量级方案，它的算力成本较高，但是在目前终端算力持续升级的大环境下，仍然具有很好的落地潜力。

我们的思路是用一套紧耦合的方案，尽可能充分利用各种低成本传感器：GNSS、IMU、视觉等提供的信息，依据这些信息在不同维度上的误差特征，建立最优化模型，实现最优的位置姿态估计。

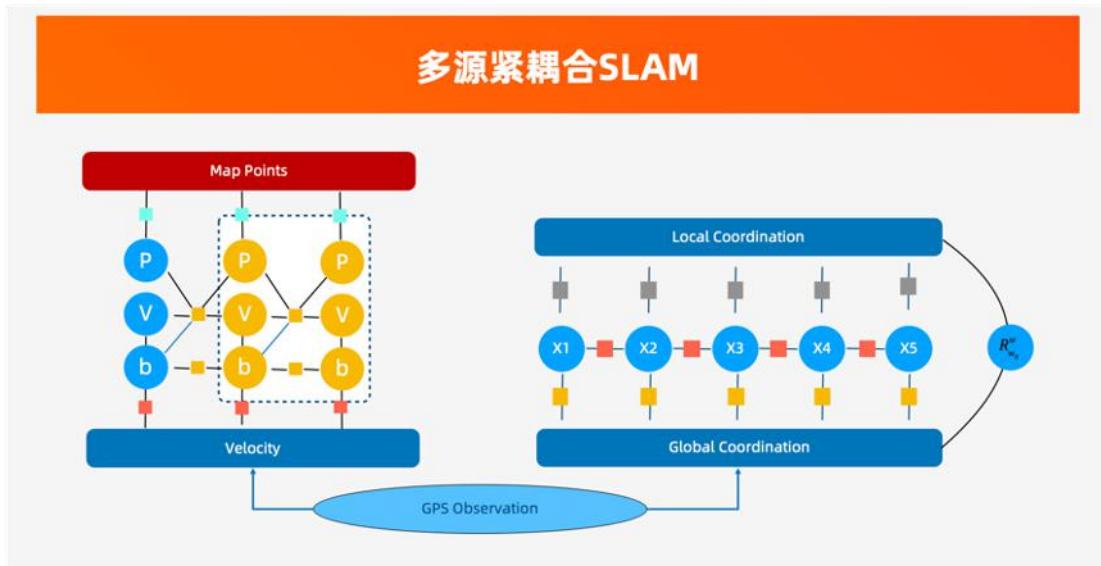


图 5 多元紧耦合 SLAM 算法框架

我们在公开数据集 EuRoc 和 Kaist 上对比了这一套多元紧耦合 SLAM 算法和目前流行的视觉-IMU 融合算法、视觉-IMU-GNSS 融合算法的效果，其位置精度提升 1 倍以上。接下来我们将在手机、车机终端上优化算法的算力消耗，并在未来面向全场景的高精度导航、智能驾驶应用中落地。

四、总结与展望

定位技术的发展由来已久。事实上，如果我们回到二三十年前甚至更早，那时候就已经产生了用于测绘的专业的高精定位技术，所以定位精度本身并不是问题。但是今天在人们出行方式深刻变革的背景下再提高精

定位，我们面临的问题是怎么样构建用户用得起，又真正能够为出行提供便利的技术和应用。

所以，未来的高精定位首先需要扩展场景应用，从室外到室内，从驾车到步行，最终达到全场景覆盖。针对场景应用的特点，明确对高精定位在精度、可靠性、成本等各个维度上的需求。充分结合当前快速发展的传感、通讯、计算等领域技术，设计最佳的解决方案。可能的研发方向包括：

- 更低成本的高精 GNSS 技术，如 PPP-RTK 技术等；
- 基于最新通信技术（如 5G）的高精度定位；
- 基于最新感知技术（如低成本 Lidar）的高精度定位；
- 对各种定位技术的更深度的融合方案（如 IMU、视觉辅助 RTK 解算）。

地图兴趣点聚合算法的探索与实践

作者：昕远

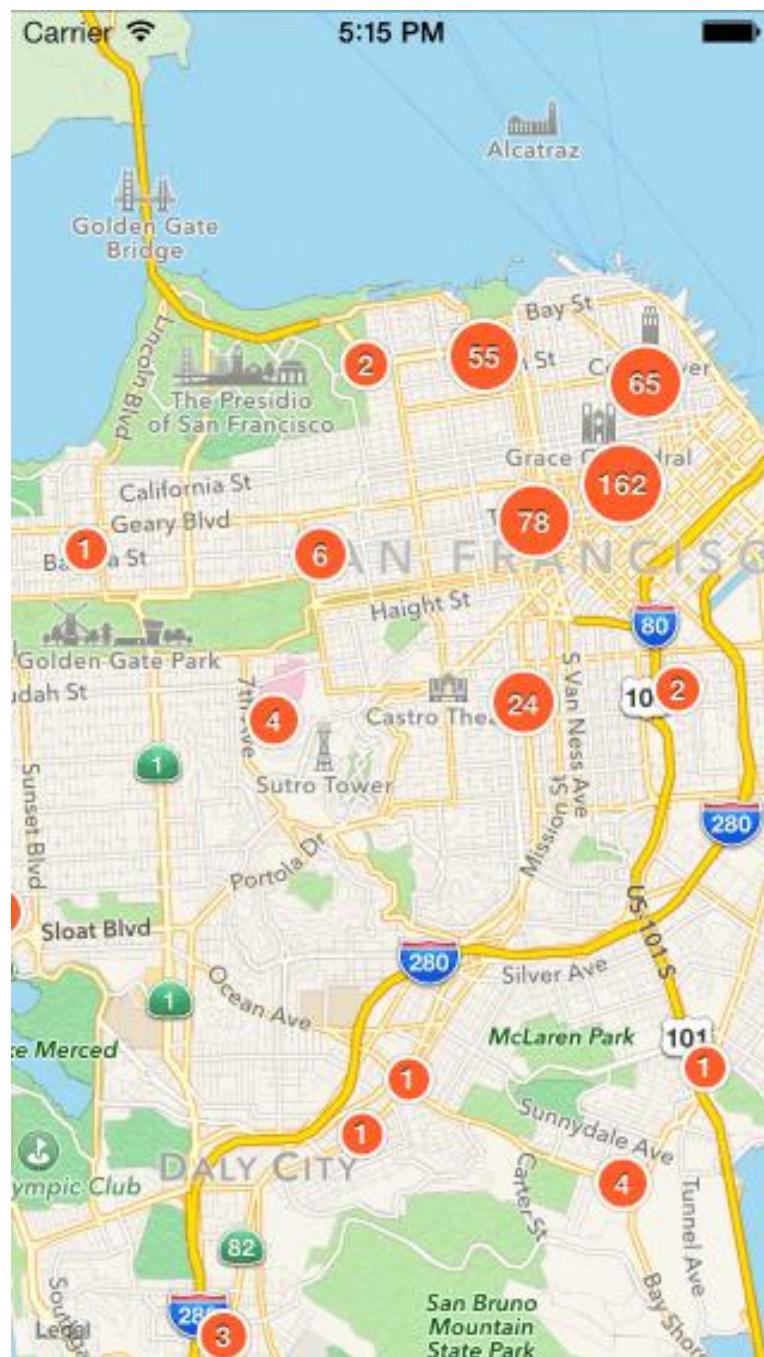
一、导读

在实现基于地图的业务时，当地图上需要展示的兴趣点（POI）过多时，一般会基于图面效果和渲染性能的考虑，在大比例尺展示完整的业务数据，而在小比例尺展示聚合态数据。在处理不同数量级、不同分布形态的POI时，如何通过算法取得更加合理的聚合效果，同时既能支持离线的预处理聚合，也能较好的满足实时聚合的性能要求是本文主要讨论的内容。

注：兴趣点（Point of Interest，通常缩写成POI）是指电子地图上的某个地标、景点，用以标示出该地所代表的政府机关、商业机构（加油站、超市、餐厅、酒店等）、风景名胜、公共厕所、交通设施等处所。

本文通过对现有聚合算法进行预研，并结合实际业务对其进行改进，构建了一套能够满足多种业务需求的数据聚合方案。其中包含多种针对不同业务场景、不

同数量级的 POI 聚合算法，并将聚合算法的配置进行了抽象，实现了聚合效果好，性能高且使用方便的聚合算法库，目前已应用到离线聚合和线上实时聚合的各个业务场景中。



二、点聚合算法比较

本节介绍目前使用较多的点聚合算法，并对不同点聚合算法的性能和聚合效果进行横向对比。针对不同的场景给出了参考使用的点聚合算法。在此基础上构建了一套相对通用的点聚合算法工具，使用者可以根据自身业务的特点，通过一定的配置来自定义具体使用的算法、策略和参数等等。

2.1 点聚合算法

这里对使用较多的点聚合方案做简单介绍。

2.1.1 kmeans 算法

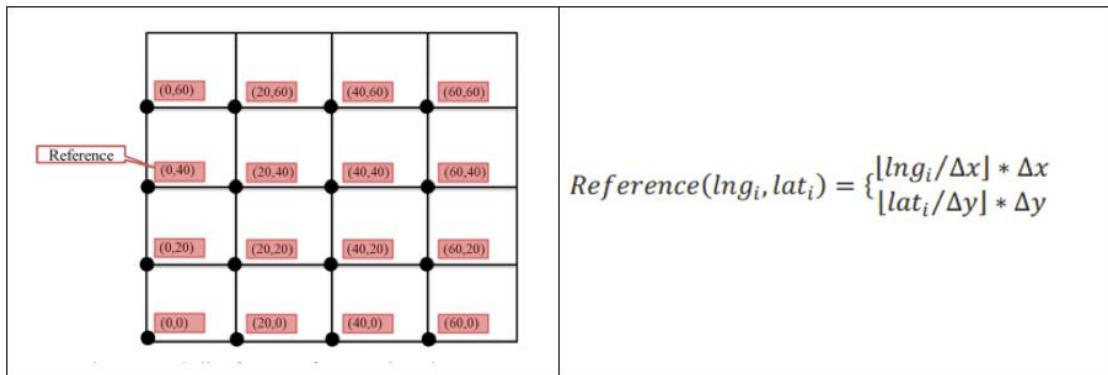
kmeans 算法主要通过初始指定 k 个聚类质心，而后按照“距离”来聚拢“相近”的数据点，而后重新计算新的质心，以此往复。不断迭代计算 k 个聚类点的质心，最终回归到 k 个聚类结果中，主要有以下两个缺点：

性能，kmeans 是计算密集型算法，模型需要迭代很多次才能够完善，且大量的距离计算比较消耗 cpu。

效果，kmeans 不能解决重叠覆盖问题（算法设计指标里边没有对覆盖问题的评估指标）。

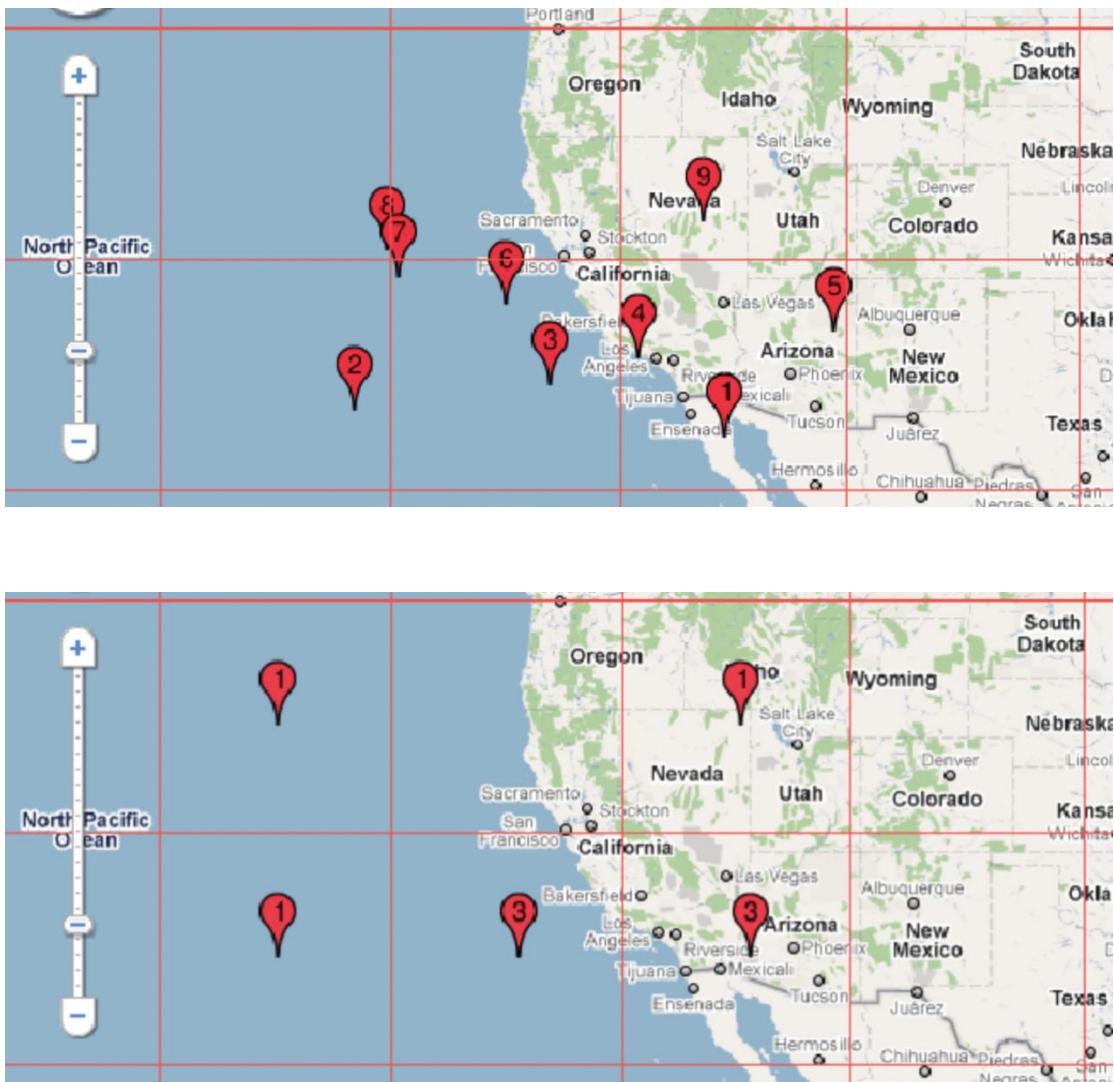
2.1.2 直接网格算法 (Grid-based Clustering)

将地图划分为若干个网格，将落在对应网格中的点聚合在小网格的中心点。每个小网格只显示一个中心点，值为网格内的点数量。具体计算公式如下：



优点，运算速度快，不涉及两个点之间的距离计算，只有点是否处在网格内的一次性计算就能拿到结果。

缺点，如下图示，明明相近的两个点，因为划分在了不同的网格，而被迫分开计算在不同的网格中心。同时网格的中心，不一定就是网格内点的聚类中心，不能真实地反映聚类的中心点。

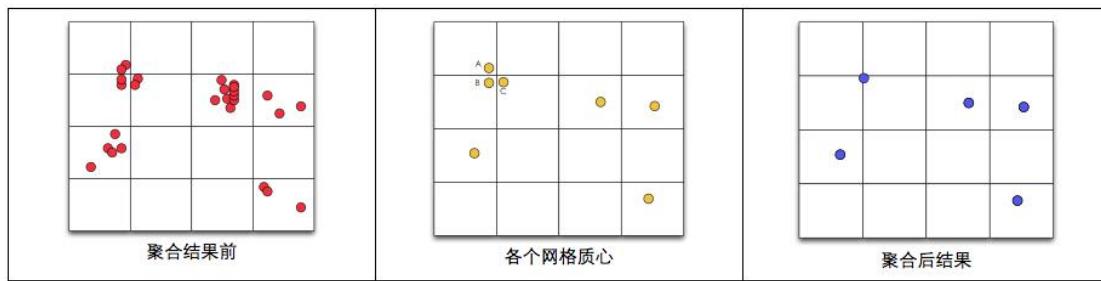


2.1.3 网格质心合并算法

与 2.1.2 方法基本一致，不同点在于，算法在将点划分到不同的网格中以后，会对每个网格的质心重新计算，得到更精确的聚类中心点。

此外，还对经过质心计算的网格聚类中心点，进行了合并。（如果不进行合并，可能导致不同网格质心相近，造成覆盖）。网格质心的合并算法以一个网格质

心为中心，画一个圆圈（或方格），将在这个范围内的网格质心都进行合并。圆圈或者方格的覆盖范围，可以作为配置来调整。



优点，运算速度较快（计算质心和合并质心不会带来特别大的计算量）

缺点，增加了计算量，同时也存在一定的误差，网格的划分，依旧可能会将原本聚集的点，强制分离开。

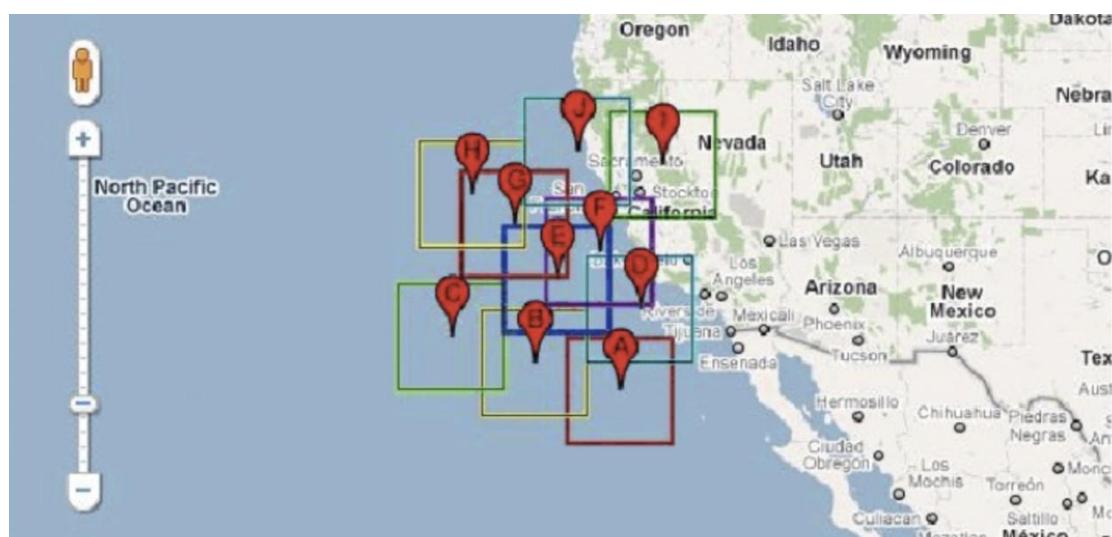
2.1.4 网格距离算法 (Grid-Distance-based Clustering)

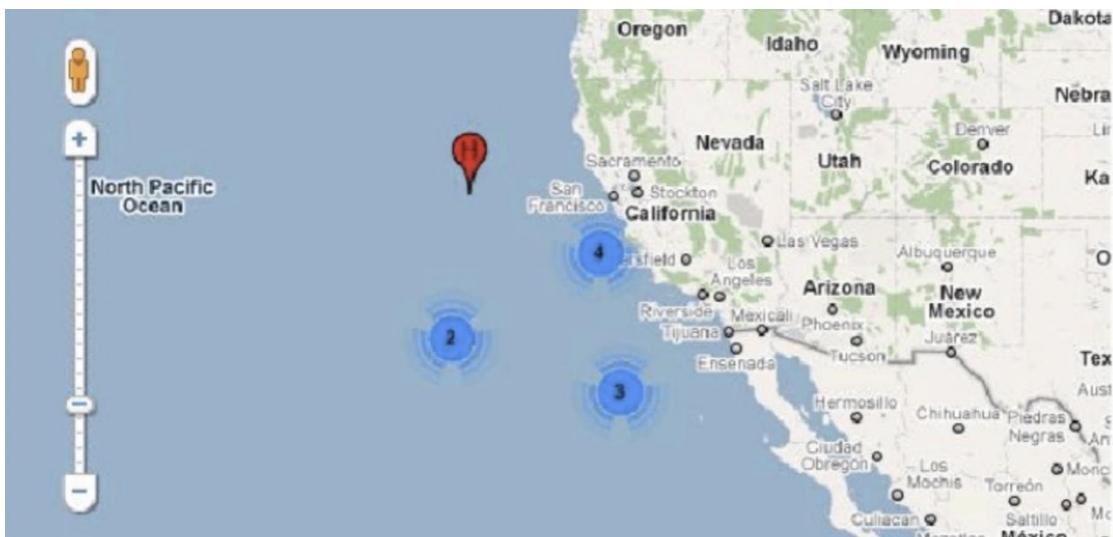
初始时没有任何已知聚合点，然后对每个点进行迭代，计算一个点的外包正方形，若此点的外包正方形与现有的聚合点的外包正方形不相交，则新建聚合点（这里不是计算点与点间的距离，而是计算一个点的外包正方形，正方形的边长由用户指定或程序设置一个默认值），若相交，则把该点聚合到该聚合点中，若点与多个已知的聚合点的外包正方形相交，则计算该点

到到聚合点的距离，聚合到距离最近的聚合点中，如此循环，直到所有点都遍历完毕。

优点，运算速度相对较快，每个原始点只需计算一次，可能会有点与点距离计算，聚合点较精确的反映了所包含的原始点要素的位置信息。

缺点，速度不如完全基于网格的速度快等，同时各个点迭代顺序不同导致最终结果不同。因此涉及到制定迭代顺序的问题。同时聚类的中心点，是第一个加入该类的点的位置，并不能够代表整个聚类的中心位置，存在一定的不准确性。





2.1.5 四叉树算法（提供快速查找 POI 点的能力）+ 网格聚合算法（提供聚合 POI 点的具体策略）

首先明确，四叉树算法本身不提供数据聚合的能力，它的存在是为了配合之前的网格点聚合算法，提供快速查找到某一个网格下的所有 POI 数据的能力。

对应于二叉树，二叉查找树适合用来存储和查找一维数据，可以达到 $O(\log n)$ 的效率。在用二叉查找树进行插入数据时，查找一个数据只需要和树结点值的对比，选择二叉树的两个叉之一向下，直到叶子结点，查找时使用二分法也可以迅速找到需要的数据。对应于地图数据的二维坐标来说，四叉树可以很好地对二维数据进行存储和查找。它能够将数据分成四个象限，在查找数据时，通过对二维属性（一般是 x, y ）进行判

断，选择四个分叉（象限）之一向下查找，直至叶子节点。（同样的对于三维数据来说，使用八叉树来进行存储和查询）。

2.2.5.1四叉树的操作

1. 节点分裂

当满足特定条件时，为了获得更好的检索效率，四叉树的节点会发生分裂，分裂的做法是：以当前节点的矩形为中心，划分出四个等大的矩形，作为四个子节点，每个节点深度=当前节点深度+1，根节点深度为0；并且将当前节点的元素重新插入到对应的子节点。

2. 插入元素

从根节点开始，如果当前节点无子节点，将元素插入到当前节点。如果存在子节点（定义为K），并且元素能够完全被子节K点所“包围”，就将元素插入到子节点K，对于子节点K进行上述递归操作（即查看K节点是否有子节点，如没有子节点，则将数据存储在K节点上，如果有子节点，则下沉继续查找匹配）；如果元素跨越了多个子节点，那就将元素存储在当前

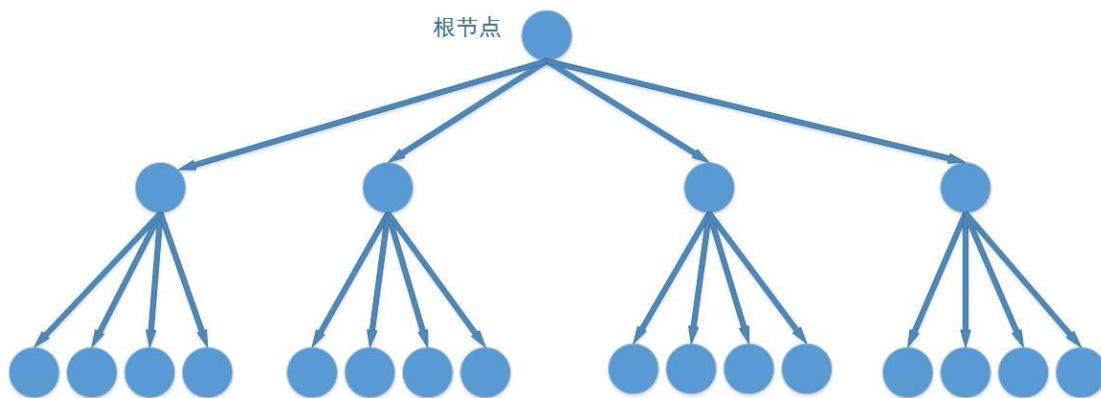
节点（即可以将跨越多个节点的数据存储在多个节点上）。

如果某一节点元素超过特定值，并且深度小于四叉树允许的最大深度，分裂当前节点。

3. 查找元素

对于给定检索区域，从根节点起，如果检索区域与当前节点有交集，且当前节点没有子节点，则返回当前节点的所有元素。

如果当前节点还有子节点，递归检索与检索区域有交集的子节点。



2.1.5.2 基于四叉树的点聚合方案具体实现：

采用网格质心算法来进行点的聚合，四叉树提供数据查询的能力。简单来说就是把屏幕分割成若干个区域，

每个区域最多显示一个标注，每个区域的数据内容由四叉树来进行查询和提供。然后根据地图缩放比例去设置每个网格区域的大小以达到较好的展示效果。

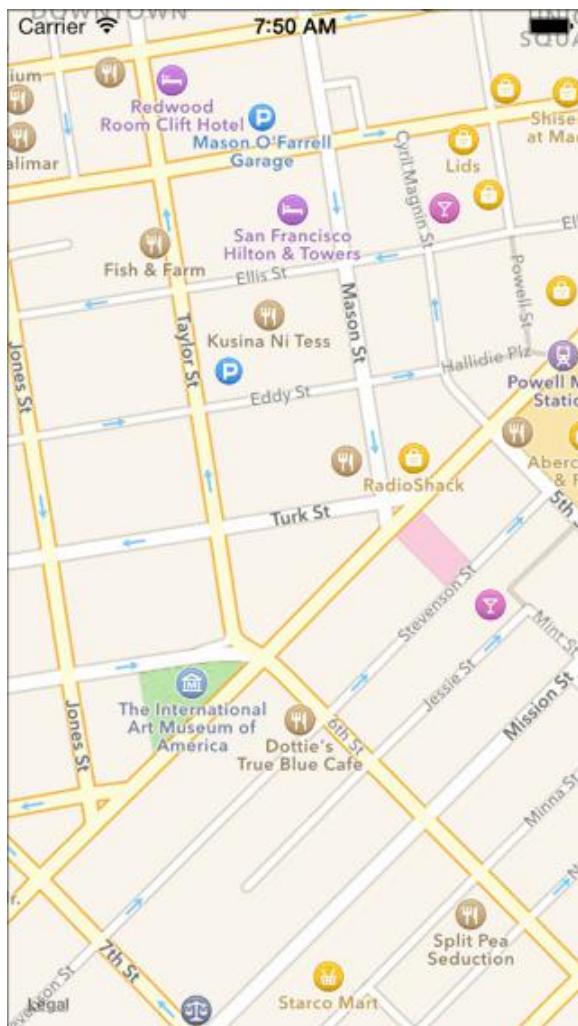
1. 使用基于四叉树的点聚合的方案首先需要建立四叉树。具体实现原理如下：

a. 首先对 POI 数据建立四叉树索引，四叉树的结构可以用如下代码表示。（创建四叉树的代价比较大）

```
1. typedef struct TBQuadTreeNodeData { //四叉树中存储的数据点(即poi  
信息) (一个四叉树节点可能包含多个数据点)  
2.     double x; //经纬度坐标  
3.     double y;  
4.     void* data; //额外的补充信息  
5. } TBQuadTreeNodeData;  
6. TBQuadTreeNodeData TBQuadTreeNodeDataMake(double x, double y, void*  
data);  
7.  
8. typedef struct TBBoundingBox { //每个四叉树节点所表示的区间范围  
9.     double x0, double y0; //横纵坐标的最小值  
10.    double xf, double yf; //横纵坐标的最大值  
11. } TBBoundingBox;
```

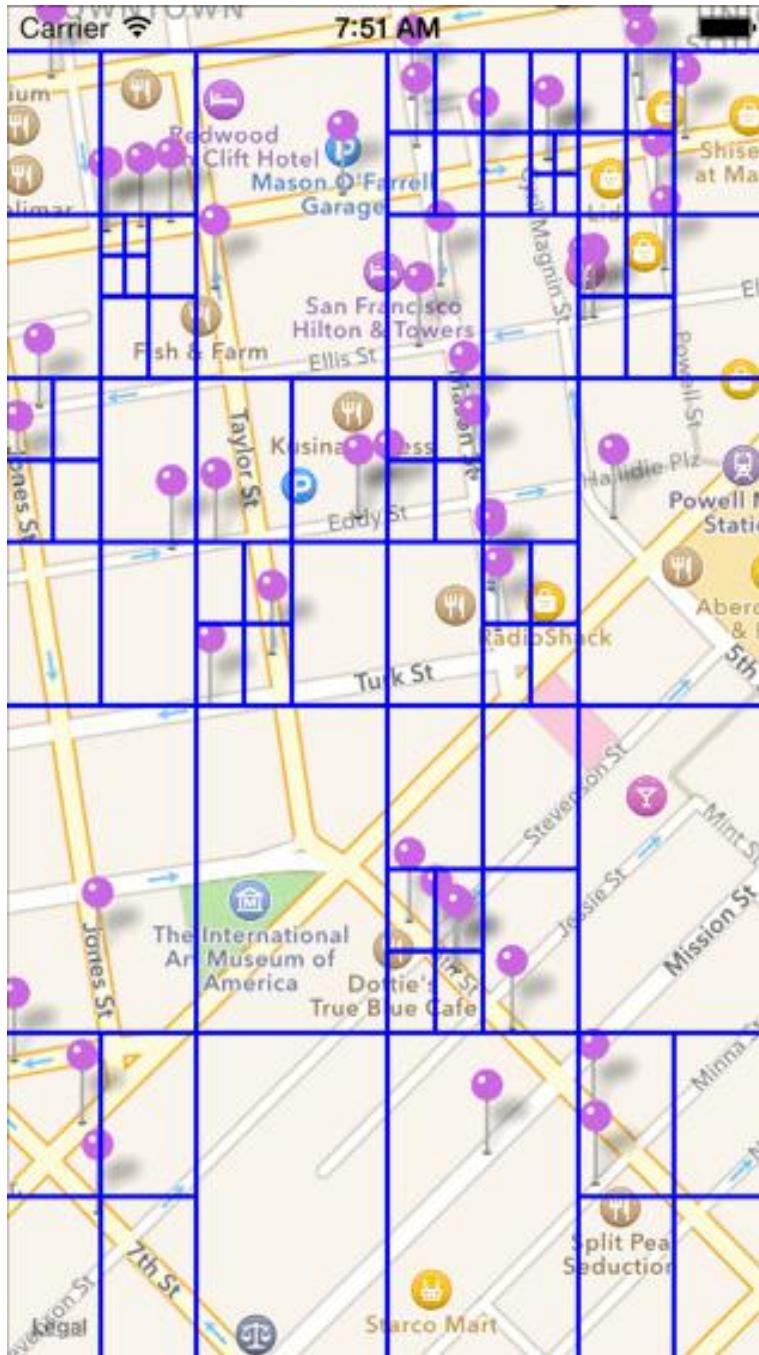
```
12. TBBoundingBox TBBoundingBoxMake(double x0, double y0, double xf, do  
uble yf);  
13.  
14. typedef struct quadTreeNode { //四叉树节点  
15.     struct quadTreeNode* northWest; //西北象限 (子节点)  
16.     struct quadTreeNode* northEast; //东北象限 (子节点)  
17.     struct quadTreeNode* southWest; //  
18.     struct quadTreeNode* southEast; //  
19.     TBBoundingBox boundingBox; //本节点所表示的数据范围  
20.     int bucketCapacity; //本节点的数据容纳量 (上限)  
21.     TBQuadTreeNodeData *points; //本节点存储的数据点信息 (poi 信息)  
22.     int count; //目前已经存储的数据点 (poi 点) 个数  
23. } TBQuadTreeNode;  
24. TBQuadTreeNode* TBQuadTreeNodeMake(TBBoundingBox boundary, int buc  
ketCapacity);
```

b. 建立四叉树的过程如下图所示，在到达节点的容量上限之后节点就会将其表达的数据范围从中心点开始划分为四个象限，构成它的四个子节点，子节点如果达到容量上限，同样重复这一过程。



c. 查找一定范围内的 POI 数据的过程如下图所示（如下图中查找红色边框圈出来的范围）。

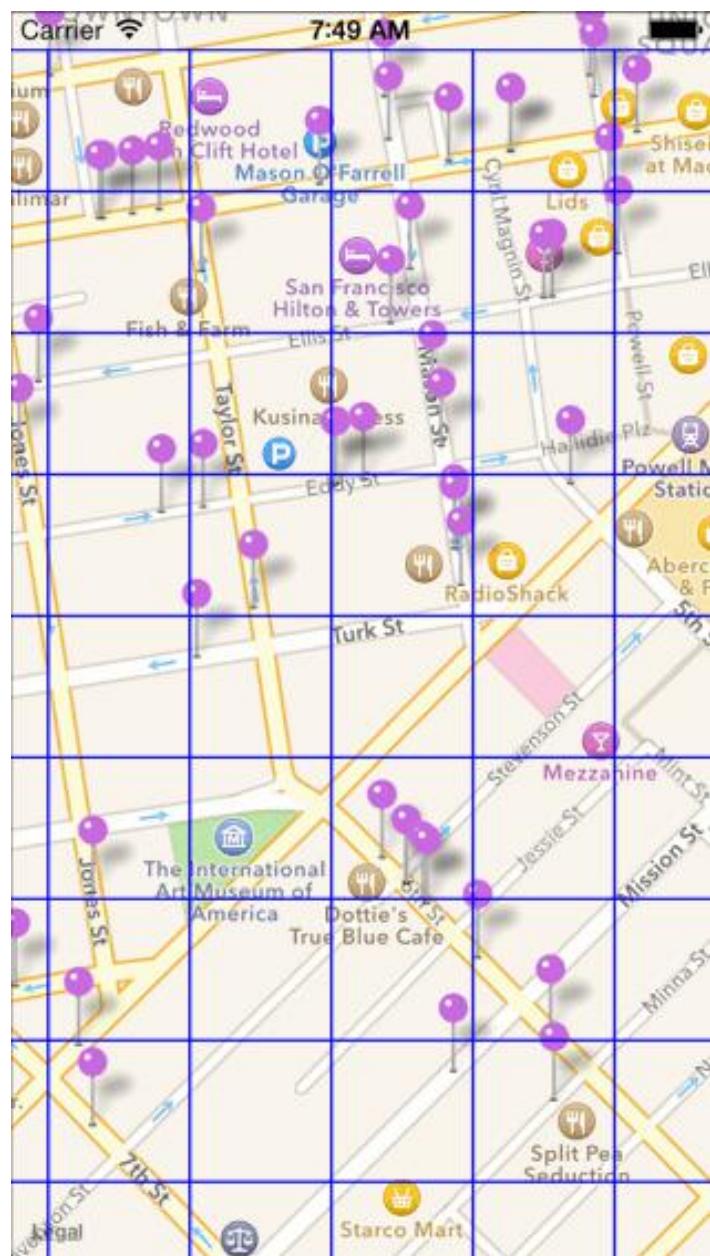
比较红色区域是否和四叉树元素的根节点有交集，无交集则舍弃，有交集继续向四叉树的分支中进行查找。



2. 四叉树建好并且能够进行区域查询

接下来点聚合算法就可以开始执行了，点聚合算法首先会先将当前屏幕划分为若干个网格（grid），然后对每一个网格通过四叉树来查找该网格内的POI，等

找到同一个网格中的所有 POI 数据之后，计算其平均质心，并统计该网格中一共存在多少数据，即可完成聚合。



2.1.6 K-D 树算法（同四叉树算法，提供快速查找的能力）

K-D 树 (k-dimensional 树的简称) 是一种分割 k 维数据空间的数据结构，主要应用于多维空间数据的搜索（如：范围搜索和最近邻搜索）。了解 K-D 树可以从理解线段树开始。

2.1.6.1 线段树的实现

1. 线段树的本质是一棵维护一段区间的平衡二叉树。线段树维护的是一个区间内的最大值。比如树根是 8，维护的是整个区间的最大值，每一个中间节点的值都是以它为树根的子树中所有元素的最大值。其构建过程可以使用如下伪代码表示：

```
1. class Node:  
2.     def __init__(self, value, lchild, rchild):  
3.         self.value = value  
4.         self.lchild = lchild  
5.         self.rchild = rchild  
6.  
7.     def build(arr):  
8.         n = len(arr)  
9.         left, right = arr[:n//2], arr[n//2:]
```

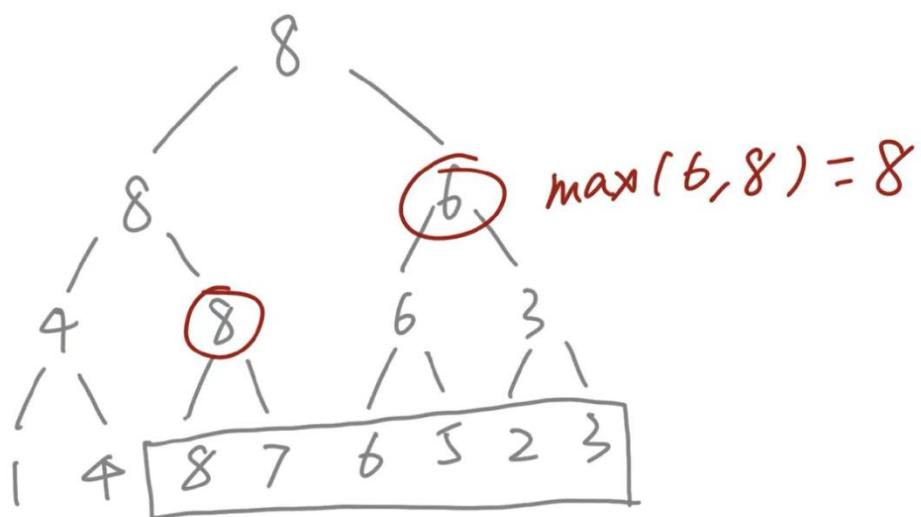
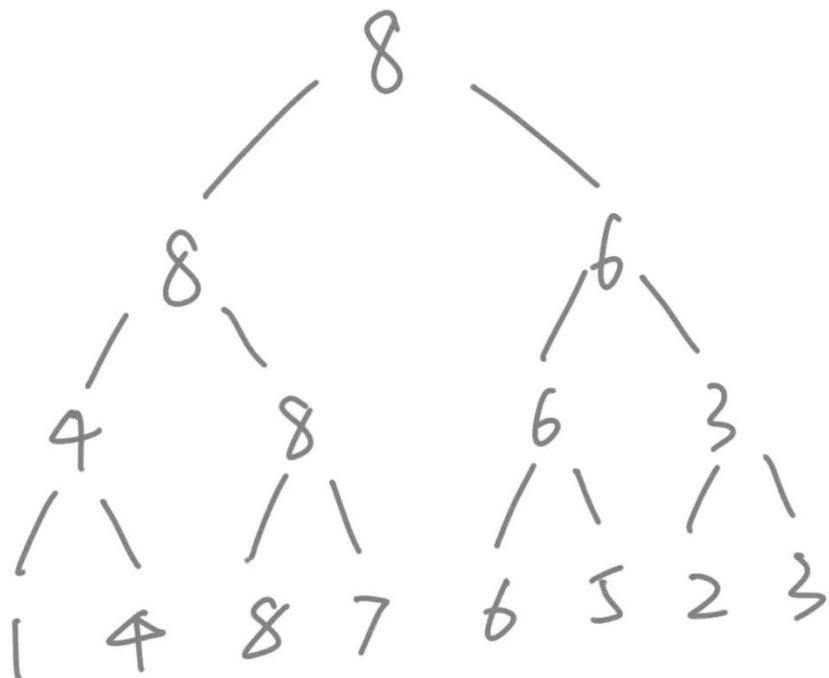
```
10.     lchild, rchild = self.build(left), self.build(right)  
11.     return Node(max(lchild.value, rchild.value), lchild, rchild)
```

2. 通过线段树，可以在 $O(\log N)$ 的时间内计算出某一个连续区间的最大值。如下图所示：

3. 当需要查找树底层被方框圈起来的区间中的最大值时，我们只需要找到能够覆盖这个区间的中间节点就行。可以发现被红框框起来的两个节点的子树刚好覆盖这个区间，于是整个区间的最大值，就是这两个元素的最大值。

这样，我们就把一个需要 $O(N)$ 查找的问题降低成了 $O(\log N)$ ，不但如此，也还可以做到 $O(\log N)$ 时间复杂度内的更新，也就是说不仅可以快速查询，还可以快速

更新线段当中的元素。



2.1.6.2 K-D 树的实现

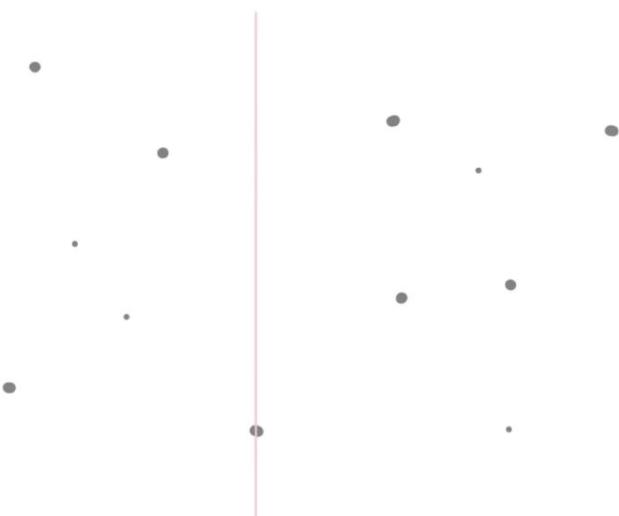
线段树维护的是一个区间内的元素，是一个一维的序列。如果我们将数据的维度扩充一下，扩充到多维呢？KD-Tree 就可以理解成是线段树拓展到多维空间当中的情况。以二维空间数据来说明 K-D 树如何建立。

1. K-D 树建立过程

a. 一个二维的平面中分布着若干个点坐标。



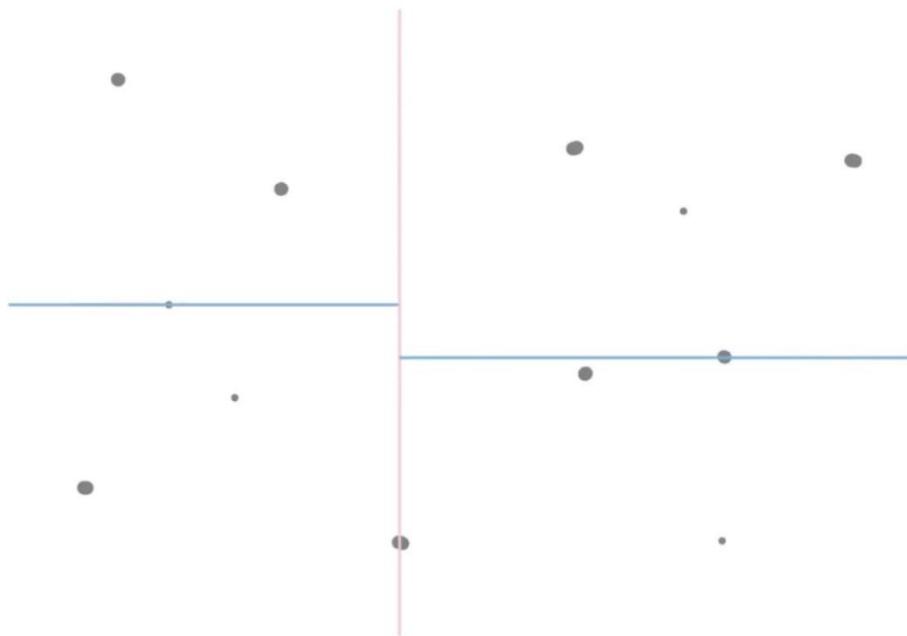
b. 选择一个维度(比如 X 轴)，将数据一分为二。



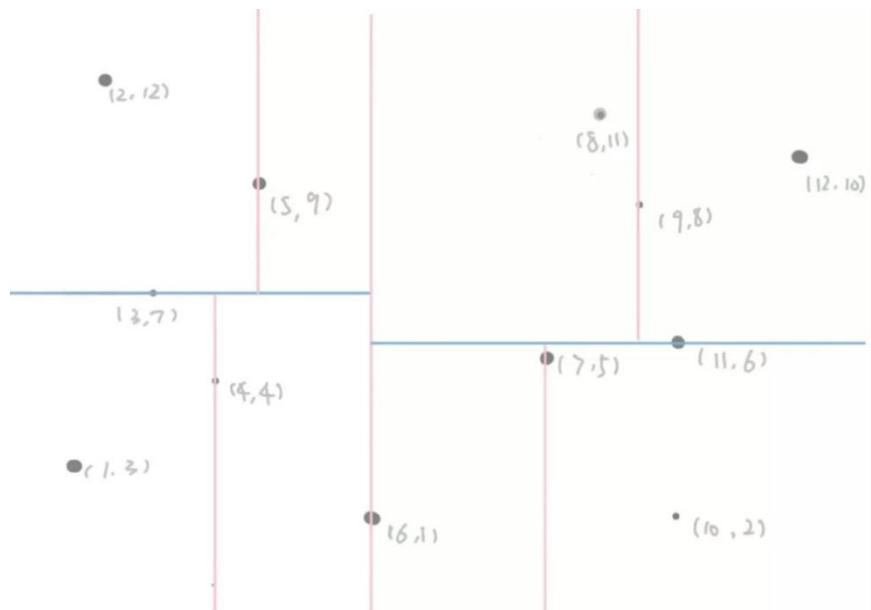
c. 经过切分之后，左右两侧的点被分成了两棵子树。

对于这两个部分的数据来说我们更换一个维度在进行二分。其实可以选择方差最大的维度进行划分，以此来保证更平均的分配，和更好的区分度）。

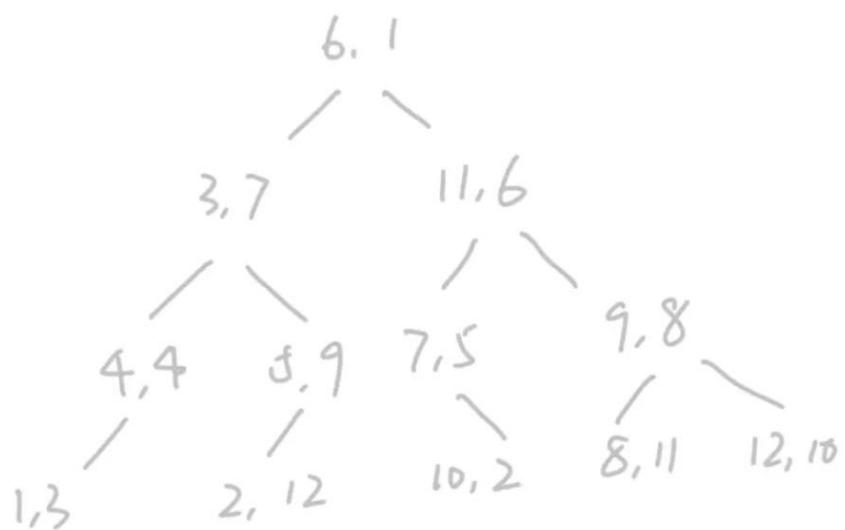
此处我们选择 y 轴进行划分，就可以得到以下图形：



d. 重复上述过程，一直到点不能进行分割为止。即可得到一颗 KD 树。因为每次划分都是选择中位数来进行划分，所以可以保证根节点到叶子节点的深度不会超过 $\log(N)$ 。



最终建立的 K-D 树存储上的形式如下图所示：

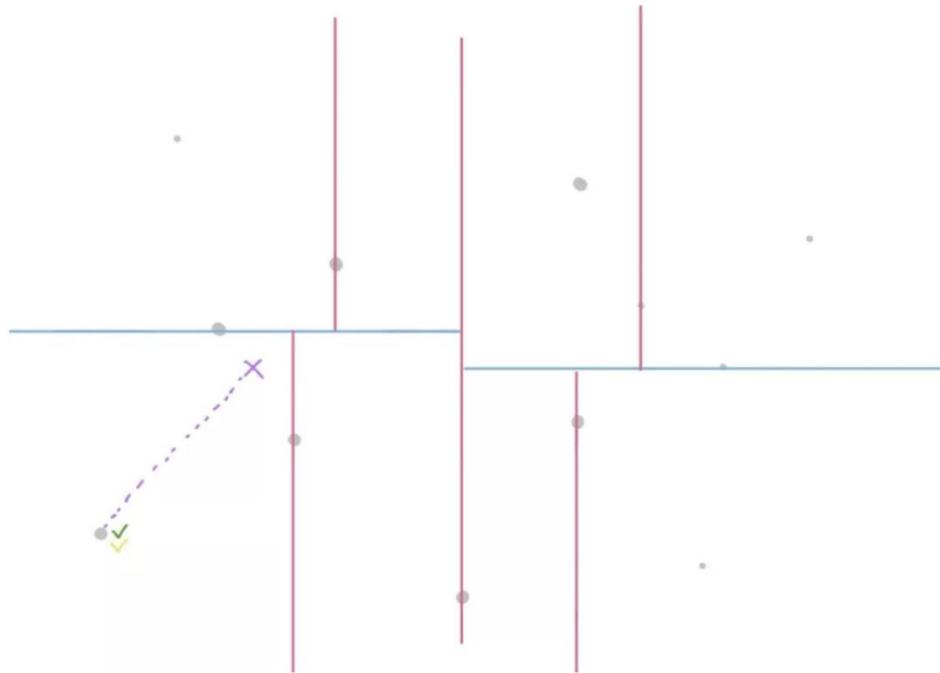


2. K-D 树查询过程

K-D 树的一个最大的优点在于，能够快速的进行批量

查询，如查询 K 个距离最近的数据有哪些、查询距离满足一定阈值的数据有哪些等。

a. 假设我们要查找距离给定目标点最近的 3 个点。首先会创建一个候选集来存储答案。当找到叶子节点（叶子节点代表一块儿小区域）之后，这个区域当中只有一个点，把它加入候选集。如下图所示，“绿色”表示样本被放入了候选集当中，“黄色”表示已经访问过。



b. 通过判断样本和当前分割轴的距离，来确定分割轴的另一侧有没有更临近的数据点。

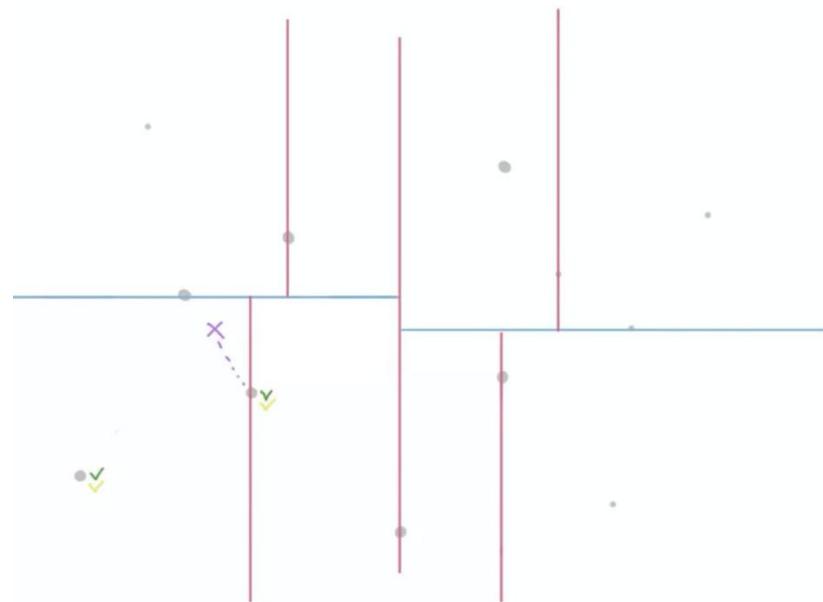
如果是叶子节点，没有轴的另一侧，则向上从父节点开始查找。

如果是非叶子节点，且当前候选集中已经存在 K 个最小值，则计算候选集中与目标点距离最远的距离 (d_2)，与目标点到分割轴的距离 (d_1) 谁更大。

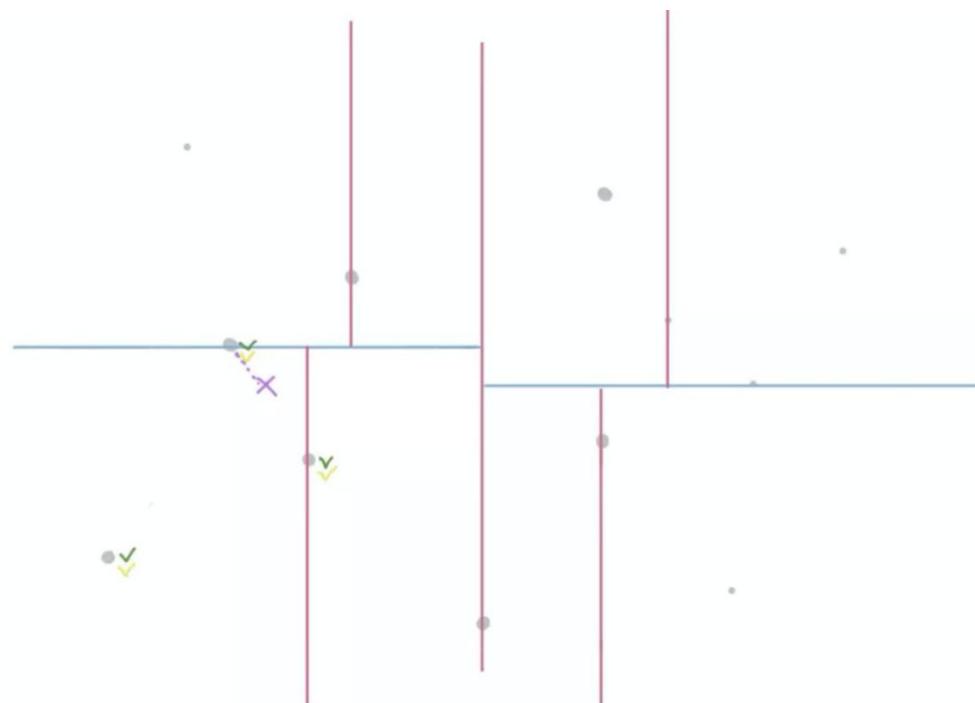
如果目标点到分割轴的距离 (d_1) 更大，即 $d_1 > d_2$ ，则另一侧没有比候选集中距离（目标点）更小的点。不需要遍历另一侧的数据点。（因为另一侧距离目标点最近的距离，至少是目标点到分界轴的距离 d_1 ，还得加上另一侧到分界轴的距离 x ，即 $d_1+x > d_2$ ）。

如果目标点到分割周的距离 (d_1) 更小，即 $d_1 < d_2$ ，则分割轴的另一侧可能存在距离目标点更小的点，需要进行遍历。

(a 步骤中的节点为叶子节点，本轮向上查找父节点，同时候选集中不满 3 个，父节点加入候选集)

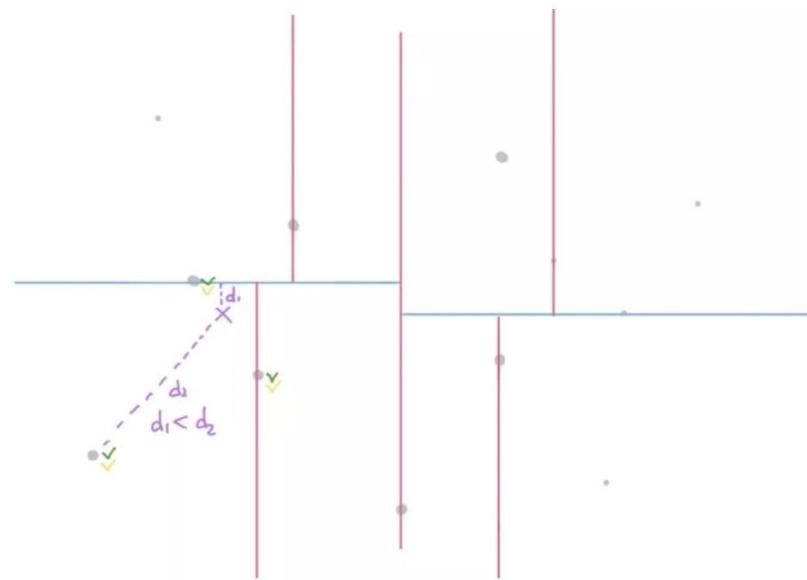


c. 分析 b 步骤中的节点，虽然是父节点，但是另一侧没有数据，所以也向上查找他的父节点，同时目前候选集不满 3 个，也加入候选集。

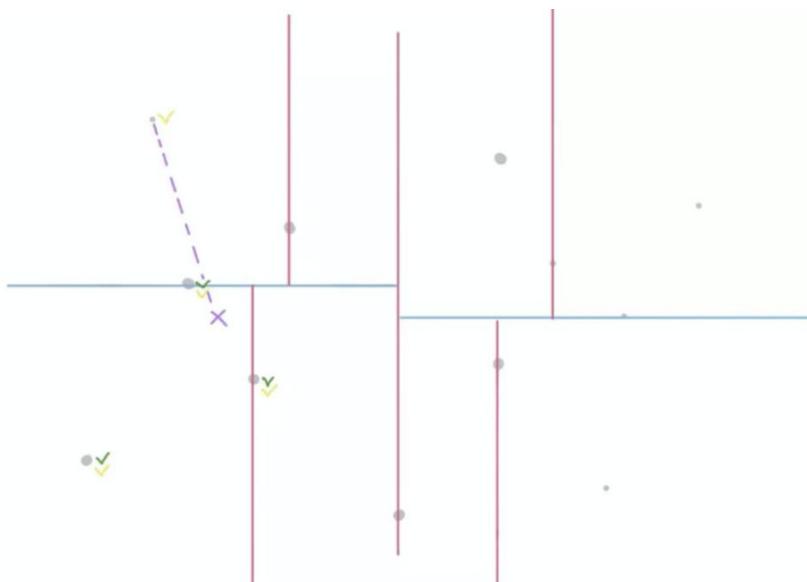


d. 当前节点作为父节点，且右子节点不为空，所以需要判断目标点到分界线的距离，目标点到分界线的距

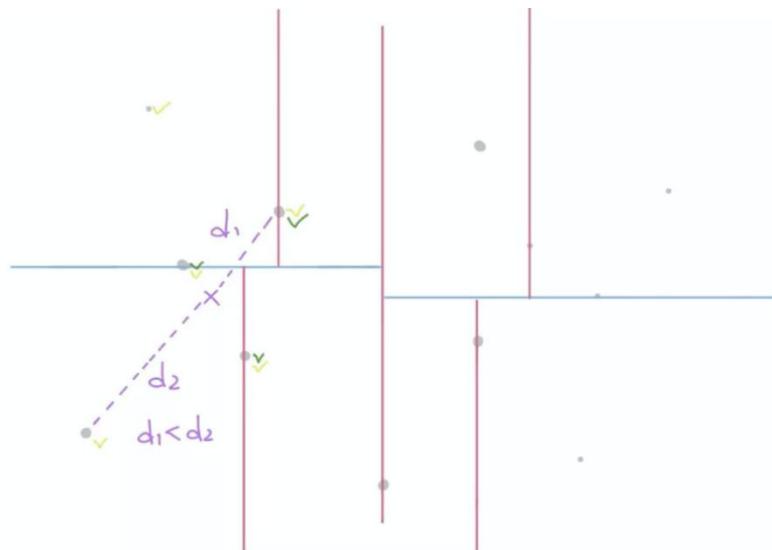
离 d_1 小于目标点到最远候选集中点的距离 d_2 ，所以分界线另一侧有可能有更小的值，需要遍历。



e. 找到边界另一侧的叶子节点，发现他到目标点距离，大于候选集到目标点的距离，同时候选集已经有了 K 个值，所以不能构成新的答案。需要继续向上查找他的父节点。



f. 发现其父节点到目标点的距离，比已有的候选集中的点更小，更新候选集。



2.2 点聚合实施方案

依托于之前进行的调研，开始对点聚合方案进行实现。本文尝试了多种算法的实现，并就其性能和效果进行了对比。其中包含：

- 直接欧式距离算法

计算每个点，与周围点的距离是否在指定阈值之内，如果满足则共同构建聚合点，聚合点的坐标取平均值。

将步骤 a 中的聚合点，从原有数据集中剔除。重新从原始 POI 点中选择一个点，再次进行步骤 a 构建新的聚合点。

重复 a, b 两个过程，直至所有 POI 点都完成了聚合。

- 基于四叉树的直接欧式距离算法

与直接欧式距离算法基本一致，不同点在于，计算与周围点的距离是否在指定阈值之内时，是通过四叉树的查询来完成的。

- 网格质心算法

与 3.1.2 中算法基本一致，不同点在于，本算法计算了每个网格中数据点的平均质心作为聚合点的位置坐标。

- 网格质心合并聚合算法

在网格质心算法的基础上，新增了聚合点的合并逻辑，通过指定阈值来完成聚合点的合并。

- 基于四叉树的网格质心算法

与网格质心算法基本一致，不同点在于，查询一个网格内对应的数据时，使用的是四叉树来进行查询。

- 网格距离算法

与 3.1.4 中算法一致。

- 基于四叉树的网格距离算法

与 3.1.4 中算法基本一致，只是使用了聚合点来构建了四叉树，来加快遍历查询的速度。

- 基于 KD 树的网格距离算法

与 3.1.4 中算法基本一致，只是使用了聚合点来构建了 KD 树，来加快遍历查询的速度。

2.3 聚合算法流程：

1. 算法的输入

请求参数中需要包含 data 和 config

data 指的是需要聚合的 poi 数据，POI 数据需要包含位置信息。config 指的是聚合算法的配置，如包含多少层级，每个层级聚合算法的相关参数等等。

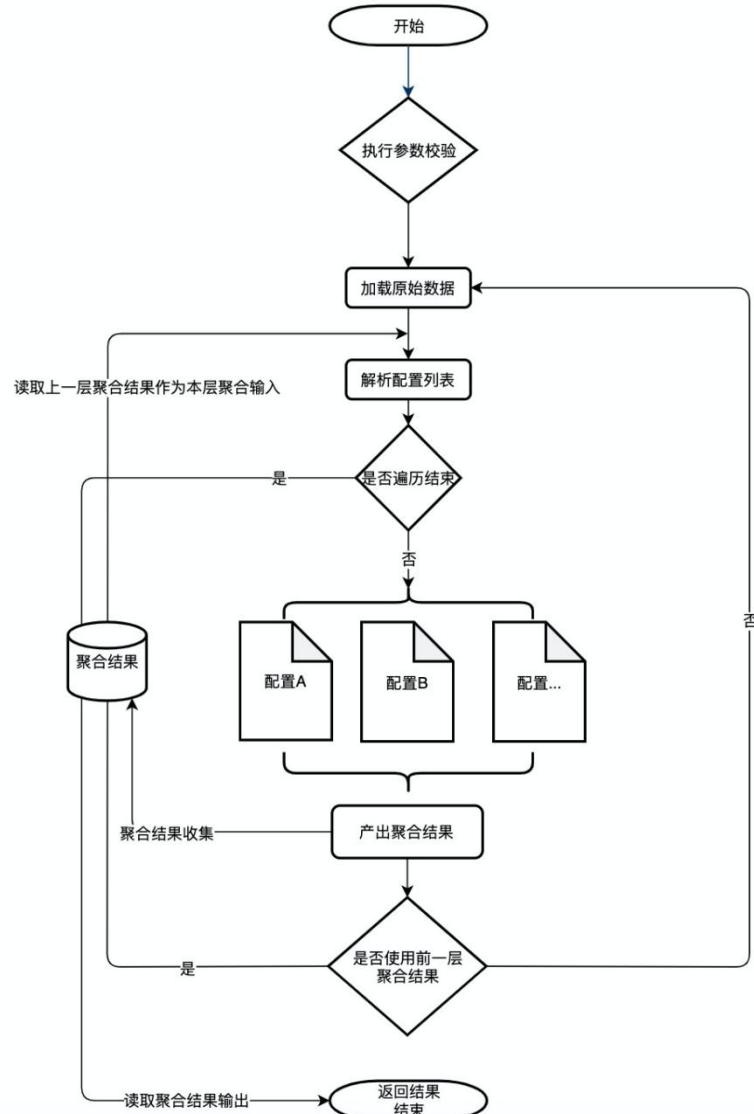
2. 算法的输出

点聚合的结果数据中，包含了不同层级的点聚合结果，以 `Map < String, List < AggregationDTO > >` 的形式下发，`String` 代表不同聚合层级的唯一标识，`List < AggregationDTO > >` 代表着该层多个聚合点的数据。

每一层的聚合结果由多个聚合点来组成，其中每一个聚合点（AggregationDTO）需要包含如下信息：

聚合点的位置（经纬度或坐标），聚合点包含的原始POI个数，其他信息（目前填充的是该聚合点聚合了哪些原始的POI数据信息）

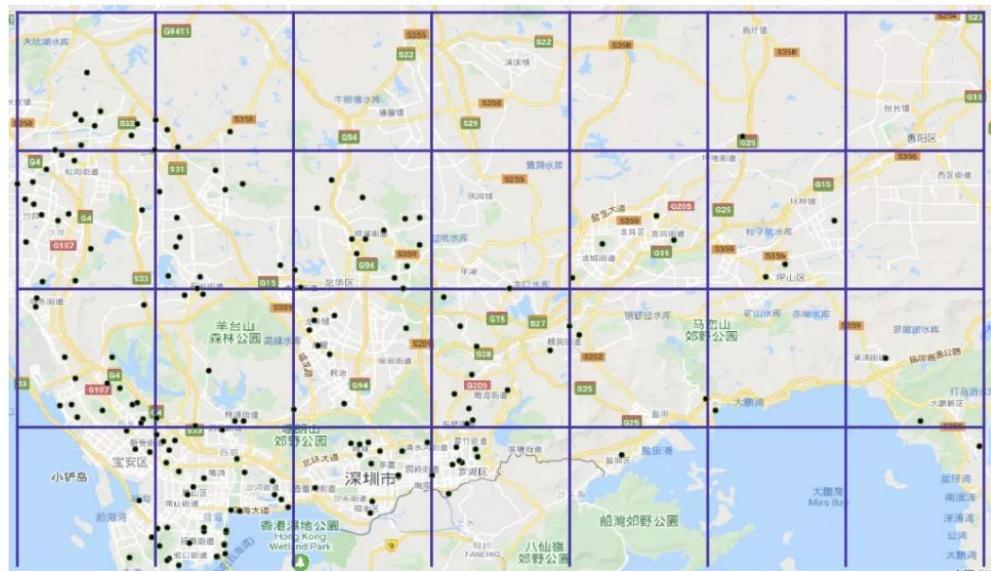
3. 整个算法的流程如下



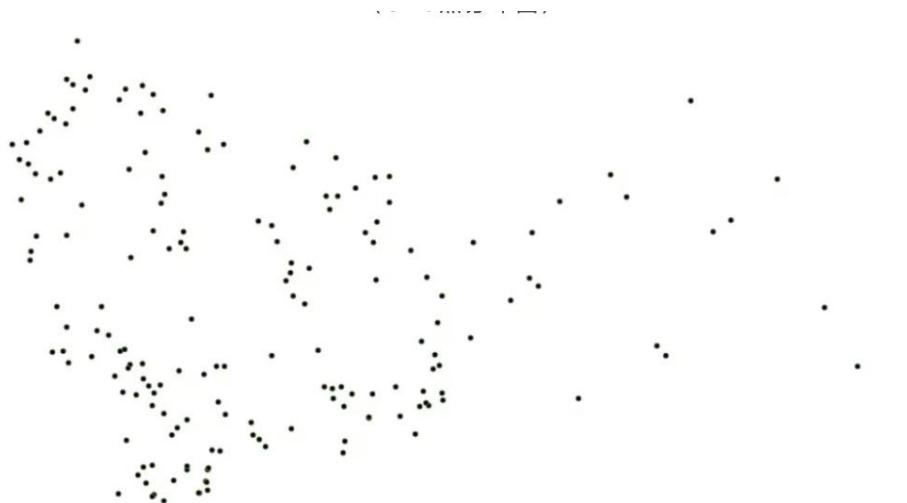
三. 结果&效果

1. 效果测试数据集

如下图所示，目前效果测试使用的数据集为GPS点坐标信息，下图中每个黑色的圆点代表一个坐标数据，共175个。



GPS 点分布图



同上图，无地图信息

2. 评价指标

对于聚类结果的评价指标，核心的思想分为两部分：
一是紧凑度，即簇内样本距离尽量的近；二是分离度，
即簇与簇之间的样本尽量的远。

一篇谷歌学术引用量 600+ 的论文做了解析， Liu Y, Li Z, Xiong H, et al. Understanding of internal clustering validation measures[C]//2010 IEEE International Conference on Data Mining. IEEE, 2010: 911–916.

结论是：一种“S_Dbw”聚类评价指标对于各种噪声，不同密度的数据集等等干扰项的评价结果鲁棒性最强，对比其他评价指标有显著优势。

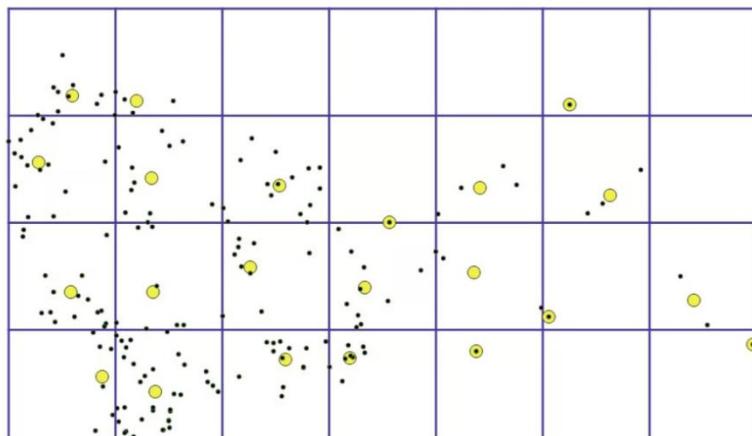
“S_Dbw”公式分为两部分， $Dens_bw(c)$ 用来衡量各个类的平均密度关系，该值较小表明，聚类簇集的类间区分度较好。

$Scat(c)$ 用来计算类内距离，距离越小同一类中数据对象间的相似性越高。具体公式中每个部分的解释可参考论文，此处不再赘述。

$$Dens_bw(c) = \frac{1}{c \cdot (c - 1)} \sum_{i=1}^n \left(\sum_{\substack{j=1 \\ j \neq i}}^c \frac{density(u_{ij})}{\max(density(v_i), density(v_j))} \right)$$
$$Scat(c) = \frac{1}{c} \sum_{i=1}^c \frac{\|\delta(v_i)\|}{\|\delta(S)\|}$$

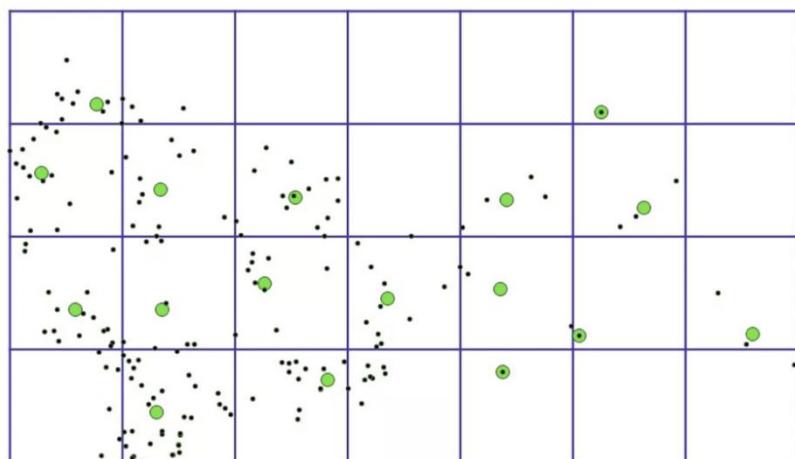
3. 算法效果（部分算法直观演示）

a. 网格质心算法（划分网格，然后计算每个网格的质心作为聚类中心）由于网格是固定的，出现了“聚集点被网格割裂”的情况。



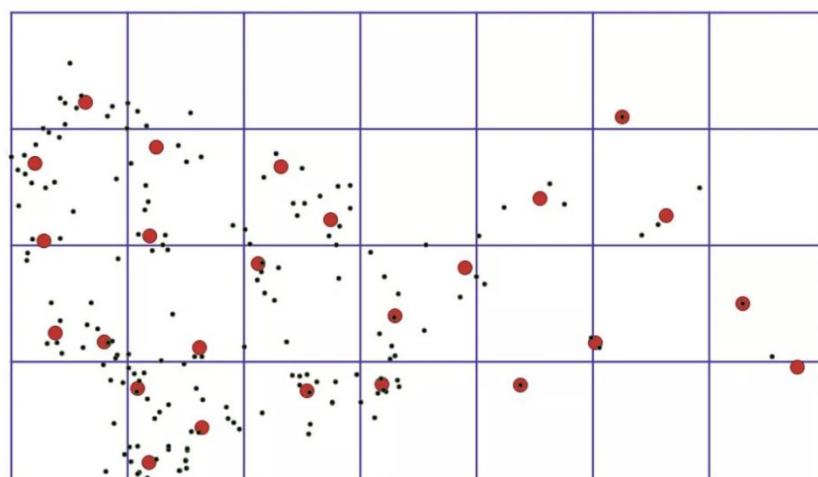
b. 网格质心合并算法（在网格质心算法的基础上，对两个距离较近的聚合点，进行合并）

将点聚合的结果，再进行一次聚合，以期望被网格割裂开来的聚集点数据，能够重新聚合在一起。

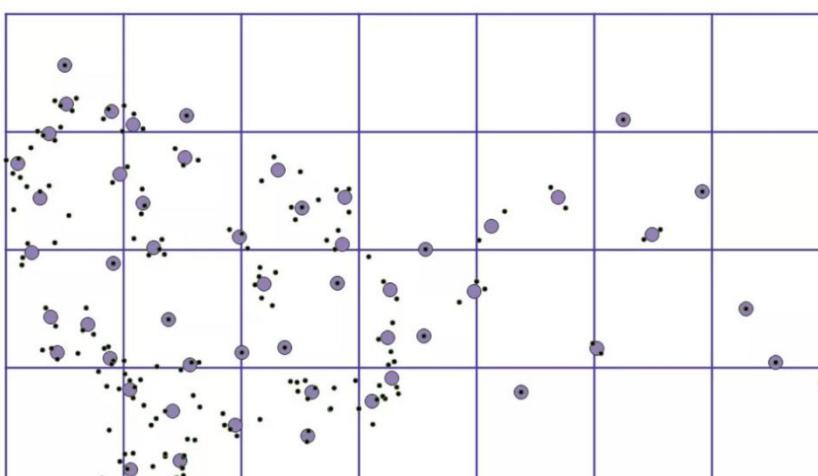


c. 网格距离算法（即 2.2.4 中的算法）

以原始数据点自身为中心，按照一定的距离范围，去聚合周围距离最近的聚合点。与其他算法不同，网格距离算法的设计思想是原始数据点和聚合点的合并，而不是原始数据点之间的相互合并。如果没有在聚合范围内找到聚合点，就以自身位置新建一个聚合点。



参数：0.04



参数：0.03

d 效果对比：

在使用直接距离聚合算法和网格质心算法中，有些场景下会将距离较远的两个点聚合在一起，网格距离算法能够有效改善这种情况。原因有两个：

因为网格距离算法是将原始数据点聚合到“距离最近的”聚合点中，能够在一定程度上避免原始数据点聚合到“距离更远”的聚合点。

网格距离算法中，对于每一个原始数据点的遍历中，它能够影响的只有自己这一个原始数据点。而直接距离算法中每一个原始数据点的遍历，可能会影响到多个原始数据点。虽然两种算法都会受到原始数据点的遍历顺序影响，但影响的程度相差很大。总体来说，网格距离算法表现更好。

网格距离算法能够在一定程度上避免“网格将原本聚集的数据点割裂”开的这种情况。

如网格距离算法中所展示的两张图，聚合点的分布并没有被网格所局限。而对比网格质心算法所展示的效果图，其每个网格只能出现了一个聚合点。

4. 不同算法使用建议

原始POI数据量	聚合点数量	准确率要求	效率要求	稳定性要求(POI顺序) & 数据分布要求	推荐使用算法
大	多	高	高	-	四叉树+网格距离算法；KD树+网格距离算法
大	多	高	低	-	网格距离算法
大	多	低	高	-	网格质心算法
大	少	高	高	-	网格距离算法
大	少	高	低	-	网格距离算法
大	少	低	高	-	网格质心算法；网格质心合并算法
小	-	-	-	-	网格距离算法

其中，原始POI数据量的大小分界线，以10K为分界线，超过10K可以认为数据量较大，算法之间的性能会有差别。聚合点数量的多少，以1K为边界，超过1K个聚合点可以认为聚合点数量较大。准确率和效率的要求，需要使用者根据业务的具体要求来做判断。

参考文献

- [1] 丁立国, 熊伟, 周斌. 专题图空间点聚合可视化算法研究[J]. 地理空间信息, 2017, 15(5): 6—9.
- [2] Theodore Calmes. How To Efficiently Display Large Amounts of Data on iOS Maps.
- [3] Liu Y, Li Z, Xiong H, et al. Understanding of internal clustering validation measures[C]//2010 IEEE International Conference on Data Mining. IEEE, 2010: 911—916.
- [4] Viky. 在线地图的点聚合算法及现状.

招聘

阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

卫星影像识别技术在高德数据建设中的探索与实践

作者：柒侠

导读

对于地图服务而言，地图数据的准确率和覆盖率是服务质量的关键因素，而地图数据的更新，依赖于多种信息源，如轨迹热力，实采图像，卫星影像等。近年来，由于遥感卫星数量的增多及高分辨率光谱相机的出现，以及卫星影像图自身覆盖广、视角好、信息丰富的特点，卫星影像作为地图数据更新的信息源起到了越来越重要的作用。

对于卫星影像的使用方式，高德经历了由前端用户展示，到人工数据作业参考，再到主动发现更新地图数据的进化过程，这同时也是我们不断挖掘影像数据价值的过程。本文会介绍高德视觉团队将卫星影像从被动参考升级为主动发现的过程中的探索和实践。

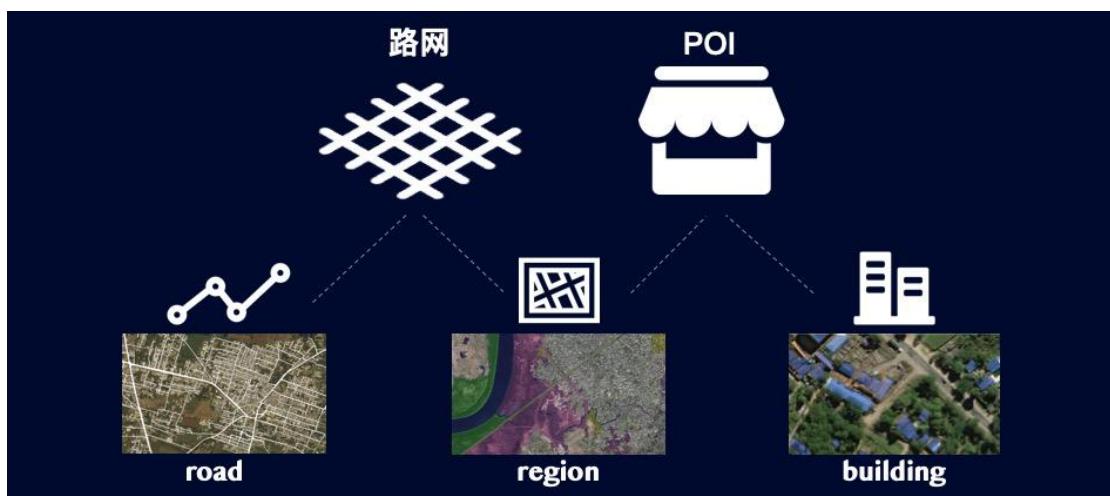
卫星影像关键元素

按照几何结构划分，影像元素可分为三大类：道路元素(road)，地物元素(region)，建筑物元素(building)：

道路元素：包含普通道路，精细道路（主/辅路/非机动车道，提前右转路），连接点（贯穿路、出入口、掉头口、路口等）。

地物元素：包含建筑区域、拆迁区域、水域、农田、山区、林地、大棚等。

建筑物元素：建筑物楼块。



卫星影像在数据更新上的优势

路网是地图数据的基础，所有的道路属性、动态事件、POI引导都需要基于准确的路网数据信息。而卫星影像由于具有上帝视角，对区域内路网的连接关系、复杂的路口关系、平立交关系的判断具有全局而丰富的信息支撑。同时，由

于卫星影像覆盖广、成本低的特点，对于热力稀疏或者采集车难以覆盖的区域，可以进行很好的路网数据补充。



路网三大信息源：热力、卫星影像、实采

作为用户导航的终点，POI（“Point of interest”的缩写，在地图数据中，一个POI可以是一栋房子、一个商铺、一个公交站等）坐标位置的准确性十分重要。通过高德POI中Top1000w的统计，70%的POI需要与楼块进行绑定，POI到达点与沿街楼块具有强依赖关系。



POI与楼块强相关性

卫星影像识别技术探索实践

卫星影像精细语义分割(Semantic)

在语义分割上，为了提升算法精度，我们将主要方向聚焦在上下文信息的结合，如使用了 U-Net 结构、ASPP、Non-local 等对信息的聚合具有作用的结构。同时引入了 Attention 加强了网络对图像显著区域，即当前分割任务所关注的类别进行了注意力聚焦，使得效果达到进一步提升。

- U-Net 结构

由于影像图像语义较为简单、结构较为固定，高级语义信息和低级特征都显得很重要，因此我们选用了 U-Net 作为网络的基础结构。Encoder-Decoder 分别下采样 4 次+上采样 4 次，将 Encoder 得到的高级语义特征图恢复到原图片的分辨率。

相比于 FCN 和 Deeplab 等，U-Net 共进行了 4 次上采样，并在同一个 Stage 使用了 Skip Connection，而不是直接在高级语义特征上进行监督和 Loss 反传。

这样，就保证了最后恢复出来的特征图融合了更多 Low-Level 的 Feature，也使得不同 Scale 的 Feature 得到了融合，从而可以进行多尺度预测和 DeepSupervision。4 次上采样也使得分割图恢复边缘等信息更加精细。

- ASPP

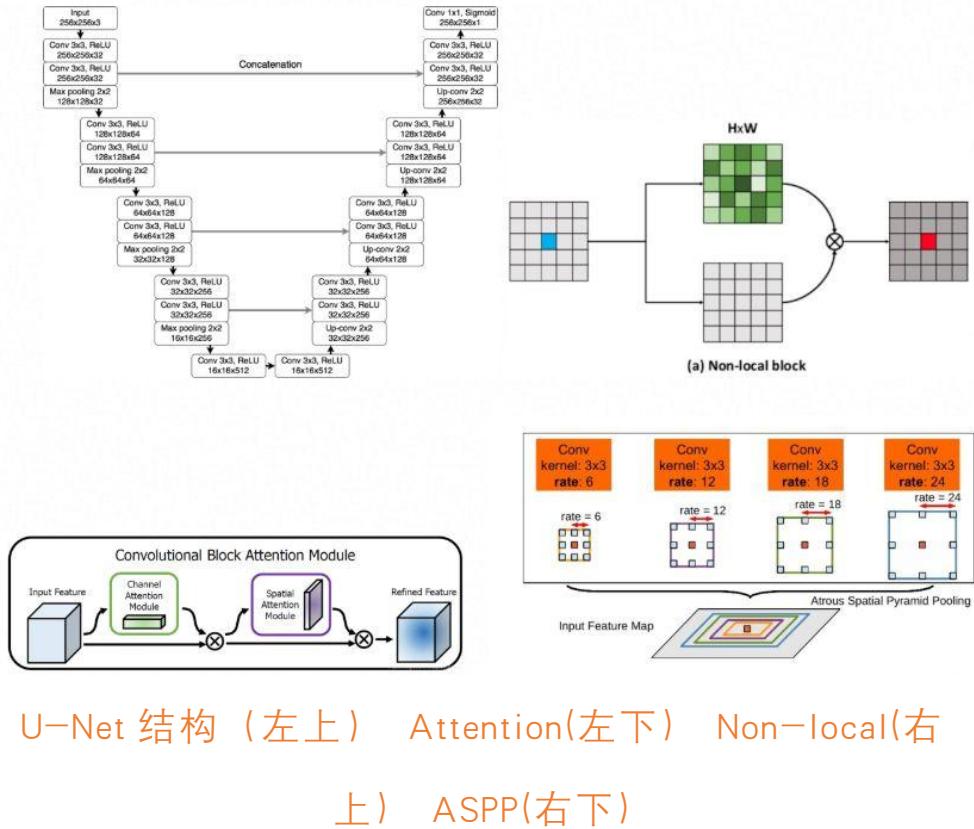
使用不同扩张率的扩张卷积，并进行特征结合，得到多尺度特征，同时得到全局信息和局部信息。

- Attention

关注图像显著区域，将 U-Net 的浅层和对应的深层进行信息结合后，得到 Attention 的参数，再作用于当前深层，得到最终 Attention 的结果输出。

- Non-local

特定层的卷积核在原图上的感受野(local)是有限的，Non-local 通过将空间中不同像素间的关系编码到当前层的输出，从而将全局信息加入到输出结果中，就能很好地解决 local 操作无法看清全局的情况，为后面的层带去更丰富的信息。



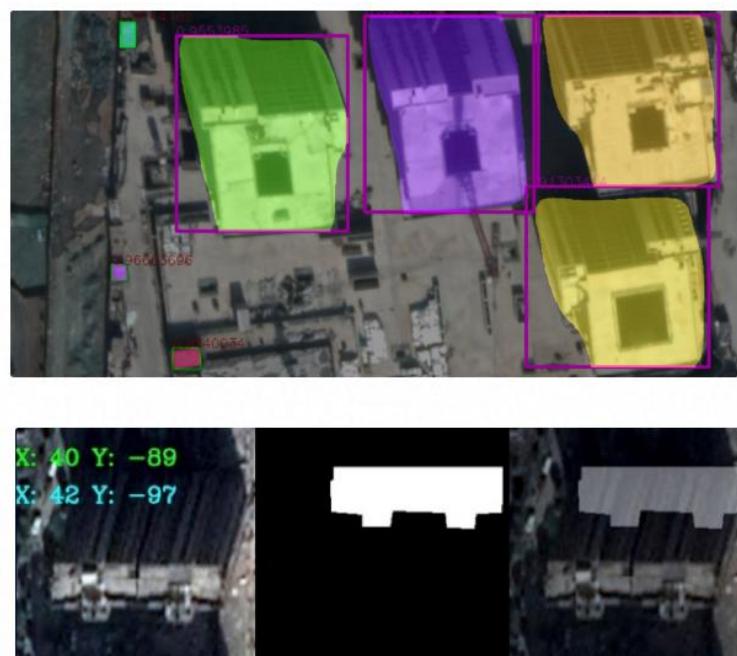
影像楼块实例分割(Instance)

实例分割有两种主流方法，第一种是基于目标检测，在得到目标检测框之后再在框内做语义分割前景和背景，由于这种方法需要借助目标检测中的区域提议，因此该方法称为 Proposal-Based 方法。

另一种方法是，在语义分割图的基础上，将像素聚集到不同的实例上，这种被称为 Proposal-Free 方法。我们对两种主流方法进行了对比实验，由于楼块具有多样性、“矮胖结构”的特点，Proposal-Based 方法效果要优于 Proposal-Free 方法。

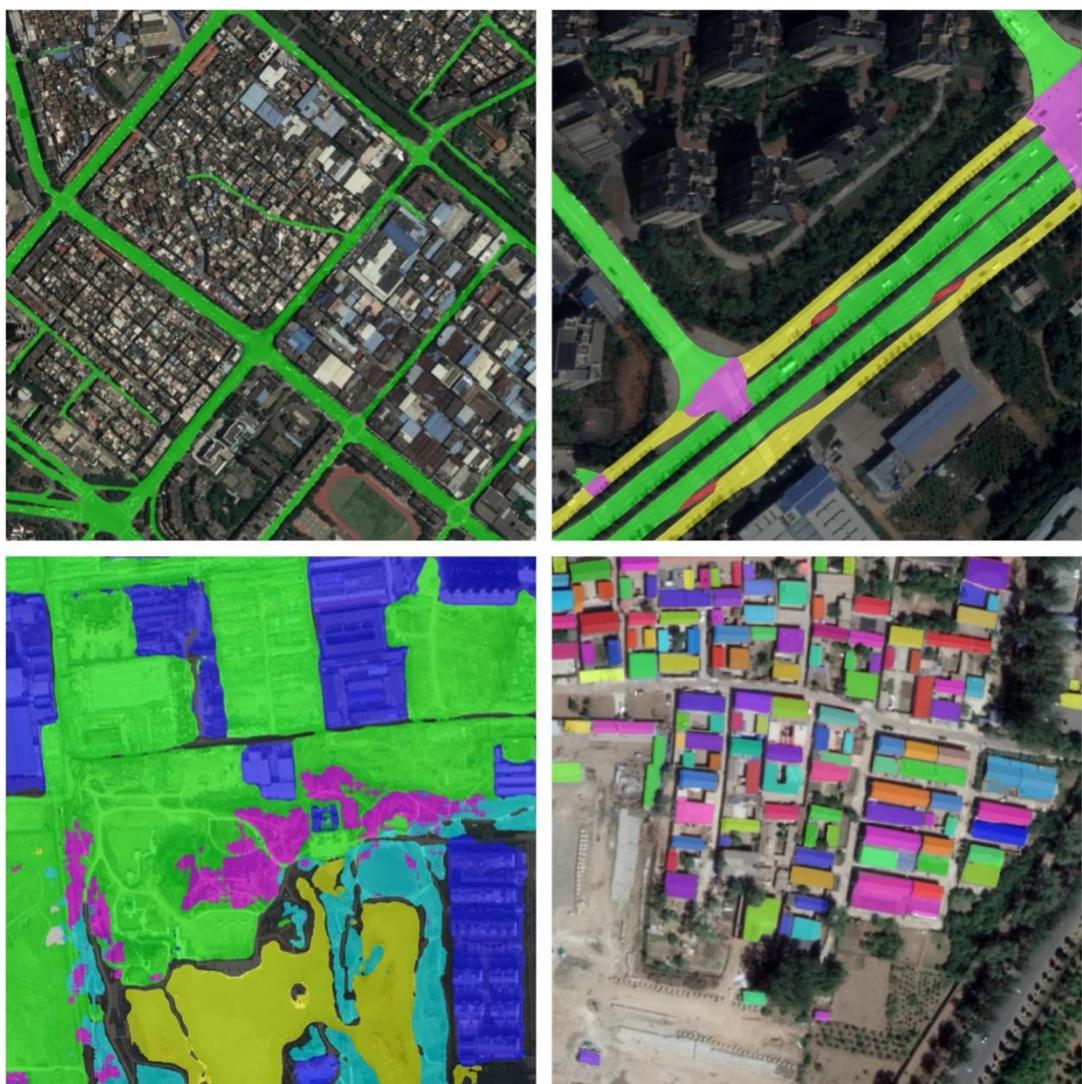
对于楼块数据而言，重要的表达内容是楼块的底座位置及其形状。然而由于影像拍摄视角问题，部分高楼在视觉上呈现斜射的效果，部分基座边缘被遮挡，为识别造成了极大的难度。

经过数据分析与推算，我们发现绝大多数的楼块底座形状是和楼顶形状一致的，因此我们采用了楼顶分割+楼顶到基座偏移量的多任务学习方案，将分割出的楼顶形状加上一个楼顶到基座的偏移向量，对基座的形状和位置进行了一个比较理想的还原。



多元素识别效果展示

针对卫星影像不同元素的图像特征与拓扑结构关系，我们设计了多个识别模型，包含普通道路识别、精细路网识别、地物分类识别、楼块识别等，作用于高德多种类别的数据更新。



普通道路识别（左上） 精细路识别(右上) 地物分类(左下) 楼块识别(右下)

未来展望&挑战

• 路网数据的准确/快速更新

用户在使用导航过程中可能会遇到一些场景：比如为什么这里有条新路却给导航了一条绕远的路？为什么导航了一条已经不能走的路？为什么本来这里可以掉头却还要往前多走几公里才能掉头？这些由路网数据错误导致的导航偏差，是我们未来需要解决的核心问题，也是业界的难题。

未来我们期望通过视觉算法层面的优化，通过多采集源的融合预测，通过提前发现建设中道路等一系列手段，来快速感知到现实世界中发生的路网变化。

• 数字城市中的楼块与 AOI 建设

对于数字城市来说，楼块和 AOI（兴趣区，Area Of interest）是重要的元素之一：如用户想要前往某个店铺，实际导航的到达点是店铺所在的楼块；用户想要前往某个小区的某个楼，实际导航的到达点是小区的入口，因此楼块与 AOI 的准确与完备直接影响到用户导航最后几百米的使用感受。同时结合最近的疫情防控，数字城市中的楼块和 AOI 信息可以对写字楼、小区等区域的疫情防控提供有力的数据支持。

未来，我们期望通过结合卫星影像的发现能力，进一步完善数字城市的数据建设，连接真实世界，让出行更美好。

招聘

高德地图视觉技术中心火热招聘中：计算机视觉、图像识别、点云、三维、AR、视频等算法类职位，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德地图首席科学家任小枫：高精算法推动高精地图落地

作者：高德技术整理

2020 云栖大会于 9 月 17 日 -18 日在线上举行，阿里巴巴高德地图携手合作伙伴精心组织了“智慧出行”专场，为大家分享高德地图在打造基于 DT+AI 和全面上云架构下的新一代出行生活服务平台过程中的思考和实践，并重点分享了「高精地图、高精算法、智能时空预测模型、自动驾驶、AR 导航、车道级技术」等话题。

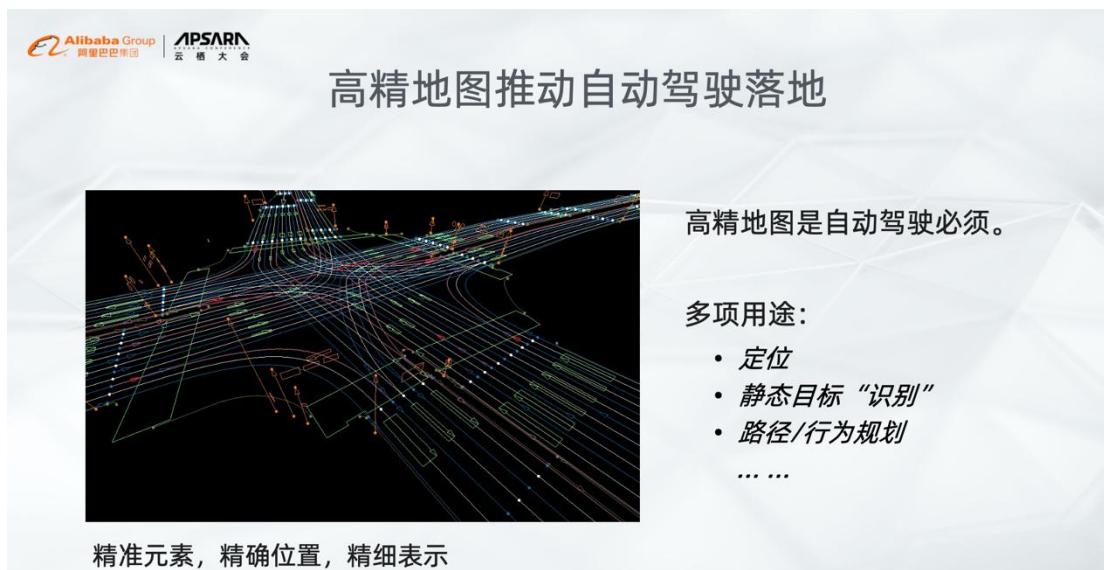
「高德技术」把本场讲师分享的主要内容整理成文并陆续发布出来，本文为其中一篇。

查看：[【演讲视频播放地址】](#)

阿里巴巴高德地图首席科学家任小枫分享的话题是《高精算法推动高精地图落地》。任小枫从算法出发，介绍了高精地图制作和落地的挑战，以及高德如何打磨和突破关键技术，把高精地图做到业界领先。

去年云栖大会，任小枫给大家整体介绍了视觉技术在高德的应用，主要分享了常规地图和 AR 导航、视觉定位方面的工作。今年时间比较短（每位讲师只有 12 分钟的演讲时间），主要介绍高精算法方面的工作。

什么是高精地图？主要是给自动驾驶场景用的地图。它和我们平时开车时所用到的地图很不一样。一个关键就是要精准。高精地图需要有精准元素，精确位置，精细表示。例如下图中的杆子、车道线、行驶轨迹等都需要在高精地图中精确的表示出来。



自动驾驶要成功落地，高精地图是必须做成的事情。高精地图有很多用途：给自动驾驶用来定位；静态目

标的建模“识别”，路径/行为规划。这些对于自动驾驶来说是非常重要的功能。

高精地图落地的挑战主要在哪里？

精度！精度！精度！（重要的话说 3 遍）其他的挑战还有规模、成本、时效。

首先是精度。高德对高精地图制作的首要原则就是要准！要做到很高的精度，百米相对精度 10 厘米。就是在真实世界中，如果有两个元素在真实世界中相距 100 米，那他们的相对位置，跟真实世界比起来的误差必须在 10 厘米之内。这是一个非常高的目标。在这样一个高目标下，生产效率和成本就成了非常大的挑战，具体细节这里不展开解释了，目前高德已经做到了行业领先的水平。此外，还有鲜度和更新，最早期的时候以年为单位，后来到了月更新，周更新，日更新甚至更快，这样才能真正做到地图信息是准确有效的。

算法在解决高精地图面临的挑战中有很重要的作用。

算法相关的工作主要有三部分：

- 资料精度与对齐
- 识别和生产自动化
- 变化发现与更新

资料精度

高精地图的首要原则是精准。高德从一开始就做了很大的投入，使用了很贵的采集车。用的高精度 Riegl 采集车，高精度激光雷度，它的测距精度能达到 5mm, 1M/sec；高精度的组合惯导；千寻基站解算…但即使这样大的投入和采集车配备，也并不意味着采集的数据资料就没问题了，还是会出问题。比如轨迹，静止或运动的时候都会出错，出错的时候并不是很多，可能只有 0.5% 的比例，但出错的时候就会造成点云资料出问题，比如点云分层，工程师们需要做很多算法方面的工作去检测，在点云分层的时候把它修复。

经过一系列工作后，能在轨迹错误率上有明显的降低。单趟资料采集精度的问题解决后，很快就会遇到多趟资料对齐的问题。举例来说，一条路上很可能做多次采集，多次采集回来的点云，如果不做处理的话就会出现非常明显的重影，它是不贴在一起的，需要用算

法的手段把它对齐。这也有很大的挑战，因为对精度的要求很高，在 5cm 以内。

各种场景的挑战很多，比如植被的影响。在不同季节的时候采集数据就会遇到这种情况，这些树和灌木会造成非常大的影响。同时在对齐的时候需要保持轨迹的刚性。因为原来采集回来的时候，轨迹的相对精度是很好的。在对齐过程中不能破坏和降低相对精度。而是要在原有基础上去提高相对精度，包括上下行的场景，如果是一条路，上下行不同两个方向和观测角度采集到的资料，以及桥上桥下，因为共视区域有限，对齐的挑战就更高了。

资料对齐

资料对齐怎么做？分为前端和后端两个部分。

前端有一个比较核心的算法就是**点云匹配**。比较常见的比如 ICP 或者 GICP 算法。但光是注意点云匹配是不够的，要把这个题解好，还需要解决很多其他的问题。比如稀疏点云特征提取，快速点云语义分割，快速车道线分割。这两块跟语义相关，在效率上是比较大的

挑战。因为点云的资料数据量非常大，在计算时间上不能花太多时间。

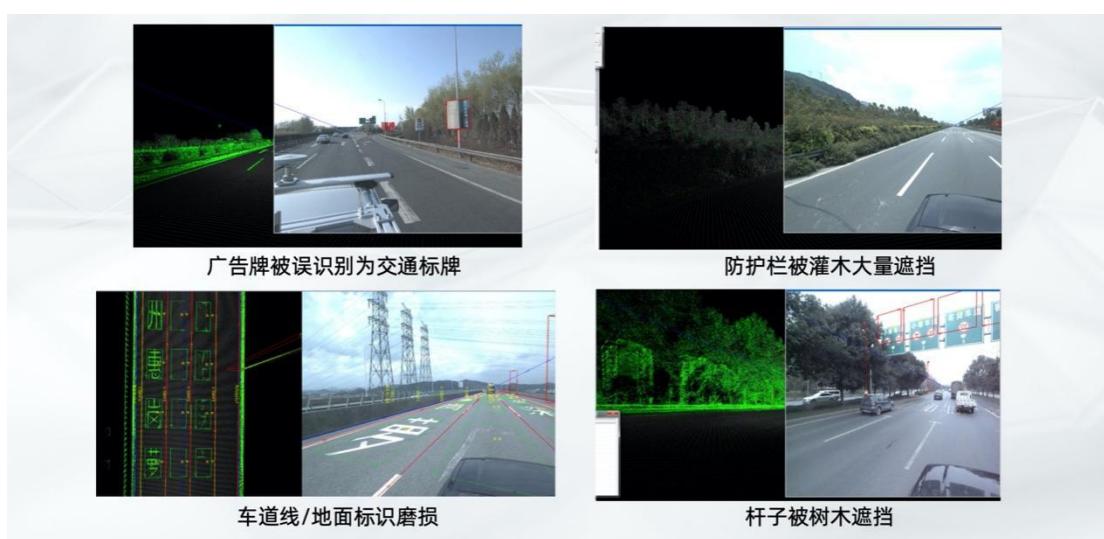
前端做了很多工作，那对齐后端做什么呢，**后端主要做大规模优化**。因为轨迹的修整不能在单点做，需要很多条轨迹，甚至在整个城市的规模上一起去做调整。

高德也花了很多时间在做优化算法。比如做了一条基于样条曲线的稀疏优化算法。把它稀疏化以后就能达到百倍级的加速。能够比较好的解决在一个城市规模下的点云对齐问题。

识别和生产自动化

对齐问题解决后就是生产效率，即识别和生产自动化的问题。高精地图里元素很多，比如线性元素里就有车道线、护栏、路缘石、自然边界，还有所谓的 OBJ，即地面标识、杆状物、交通标牌、桥、龙门架等等。这些都需要制作出来，采用自动化的方法，算法的方法就是非常重要的环节。它的输入有点云和图像识别。可以通过算法来生成 HD 地图元素。可以用算法来提高人工效率。

举几个有挑战性的例子。比如说下图的 4 种情况：



怎么解决？

输入主要是点云和图像。高德花了很多时间在优化模型能力和提高精度。包括：

- 点云语义分割(多级随机聚合网络)
- 图像全景分割(检测/分割深度融合)
- 点云/图像融合(前融合+后融合)
- 传统算法辅助(e.g. 拟合，三维)
- 矢量化及建模(深度特征+图模型)

高德在这些方法做到了很高的水准：高准确度召回 $>98\%$ ；部分实现跳点作业，达到 $>99.5\%$ 召回；部分实现免人工检查，达到 >99.5 准确率。

变化发现与更新

在地图更新方面，高德有两套方案都在做，应用于不同的场景。一种是用激光的方法，一种是用视觉的方法。

激光的方法。因为要控制成本，所以使用了相对低成本的激光和相对低成本的组合惯导。输入的资料质量比较低，需要做很多事情去提高资料的精度。包括：

- 紧耦合 Lidar SLam/LIO
- 实时语义分割
- 定位地图：多重特征图层
- In-the-loop 重定位
- 协方差模型
- 全局位姿优化
- 定位图层更新

上面这些都有相对比较成熟的解决方案。虽然输入资料差，但在更新场景下还是能达到很好的精度。

视觉的方法。由极低成本消费级相机和极低成本消费级组合惯导来做，这中间有很多视觉的算法。包括：

- 紧耦合 Visual Slam/VIO
- 定位特征图层和语义图层
- 特征+语义重定位
- 全局位姿优化(融合 VIO/重定位)
- 定位特征图层及更新

现在已经做到 15 厘米，在严格评测的条件下已经是业界领先水平了。后续会继续提高精度。

视觉更新技术可以直接应用于构建地图。下面的图显示的是视觉建图和卫星影像的重叠。



变化发现的问题，高精地图如何做到更高的精度和更
高频率的更新？

假设有一个极低成本的方案去采集图像，采集回来的肯定是一些质量很差的图像，在这些图像基础上要做对比，和真实世界的变化，比如下图里的两张图片里都有个电子眼，其实是同一个电子眼，但在图片上看着很不一样，需要用算法的手段来判断这是否为同一个电子眼。这里面就有很多图像算法的工作要做。包括：



这些方面高德都在做，也已经有了比较好的结果。这是高精地图制作过程中的一个必需的过程。这也是高德的特色所在，因为使用这些低成本的设备，能用现有的低成本的资料发现物理世界的变化。

高精地图是高德未来的一个重要方向。它的制作和落地是一个系统性的工程，除了算法以外，还有很多其他的关键工作要做。让我们一起努力。

招聘

高德地图视觉技术中心火热招聘中：计算机视觉、图像识别、点云、三维、AR、视频等算法类职位，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

揭秘！文字识别在高德地图数据生产中的演进

作者：虽寿

1. 导读

丰富准确的地图数据大大提升了我们在使用高德地图出行的体验。相比于传统的地图数据采集和制作，高德地图大量采用了图像识别技术来进行数据的自动化生产，而其中场景文字识别技术占据了重要位置。商家招牌上的艺术字、LOGO 五花八门，文字背景复杂或被遮挡，拍摄的图像质量差，如此复杂的场景下，如何解决文字识别技术全、准、快的问题？

本文分享文字识别技术在高德地图数据生产中的演进与实践，介绍了文字识别自研算法的主要发展历程和框架，以及未来的发展和挑战。

2. 背景

作为一个 DAU 过亿的国民级软件，高德地图每天为用户提供海量的查询、定位和导航服务。地图数据的丰富性和准确性决定了用户体验。传统的地图数据的采集和制作过程，是在数据采集设备实地采集的基础上，再对采集资料进行人工编辑和上线。这样的模式下，数据更新慢、加工成本高。为解

决这一问题，高德地图采用图像识别技术从采集资料中直接识别地图数据的各项要素，实现用机器代替人工进行数据的自动化生产。通过对现实世界高频的数据采集，运用图像算法能力，在海量的采集图片库中自动检测识别出各项地图要素的内容和位置，构建出实时更新的基础地图数据。而基础地图数据中最为重要的是 POI（Point of Interest）和道路数据，这两种数据可以构建出高德地图的底图，从而承载用户的行为与商家的动态数据。

图像识别能力决定了数据自动化生产的效率，其中场景文字识别技术占据了重要位置。不同采集设备的图像信息都需要通过场景文字识别（Scene Text Recognition，STR）获得文字信息。这要求我们致力于解决场景文字识别技术全、准、快的问题。在 POI 业务场景中，识别算法不仅需要尽可能多的识别街边新开商铺的文字信息，还需要从中找出拥有 99% 以上准确率的识别结果，从而为 POI 名称的自动化生成铺平道路；在道路自动化场景中，识别算法需要发现道路标志牌上细微的变化，日处理海量回传数据，从而及时更新道路的限速、方向等信息。与此同时，由于采集来源和采集环境的复杂性，高德场景文字识别算法面对的图像状况往往复杂的多。主要表现为：

- 文字语言、字体、排版丰富：商家招牌上的艺术字体，LOGO 五花八门，排版形式各式各样。
- 文字背景复杂：文字出现的背景复杂，可能有较大的遮挡，复杂的光照与干扰。
- 图像来源多样：图像采集自低成本的众包设备，成像设备参数不一，拍摄质量差。图像往往存在倾斜、失焦、抖动等问题。

由于算法的识别难度和识别需求的复杂性，已有的文本识别技术不能满足高德高速发展的业务需要，因此高德自研了场景文字识别算法，并迭代多年，为多个产品提供识别能力。

3. 文字识别技术演进与实践

STR 算法发展主要历程

场景文字识别（STR）的发展大致可以分为两个阶段，以 2012 年为分水岭，分别是传统图像算法阶段和深度学习算法阶段。

传统图像算法

2012 年之前，文字识别的主流算法都依赖于传统图像处理技术

和统计机器学习方法实现，传统的文字识别方法可以分为图像预处理、文字识别、后处理三个阶段：

- 图像预处理：完成文字区域定位，文字矫正，字符切割等处理，核心技术包括连通域分析，MSER，仿射变换，图像二值化，投影分析等；
- 文字识别：对切割出的文字进行识别，一般采用提取人工设计特征（如 HOG 特征等）或者 CNN 提取特征，再通过机器学习分类器（如 SVM 等）进行识别；
- 后处理：利用规则，语言模型等对识别结果进行矫正。

传统的文字识别方法，在简单的场景下能达到不错的效果，但是不同场景下都需要独立设计各个模块的参数，工作繁琐，遇到复杂的场景，难以设计出泛化性能好的模型。

深度学习算法

2012 年之后，随着深度学习在计算机视觉领域应用的不断扩大，文字识别逐渐抛弃了原有方法，过渡到深度学习算法方案。在深度学习时代，文字识别框架也逐渐简化，目前主流的方案主要有

两种，一种是文本行检测与文字识别的两阶段方案，另一种是端到端的文字识别方案。

1) 两阶段文字识别方案

主要思路是先定位文本行位置，然后再对已经定位的文本行内容进行识别。文本行检测从方法角度主要分为基于文本框回归的方法[1]，基于分割或实例分割的方法[2]，以及基于回归、分割混合的方法[3]，从检测能力上也由开始的多向矩形框发展到多边形文本[2]，现在的热点在于解决任意形状的文本行检测问题。文本识别从单字检测识别发展到文本序列识别，目前序列识别主要又分为基于 CTC 的方法[4]和基于 Attention 的方法[5]。

2) 端到端文字识别方案[6]

通过一个模型同时完成文本行检测和文本识别的任务，既可以提高文本识别的实时性，同时因为两个任务在同一个模型中联合训练，两部分任务可以互相促进效果。

文字识别框架

高德文字识别技术经过多年的发展，已经有过几次大的升级。从最开始的基于 FCN 分割、单字检测识别的方案，逐渐演进到现有基于实例分割的检测，再进行序列、单字检测识别结合的方案。与学术界不同，我们没有采用 End-to-End 的识别框架，是由于

业务的现实需求所决定的。End-to-End 框架往往需要足够多高质量的文本行及其识别结果的标注数据，但是这一标注的成本是极为高昂的，而合成的虚拟数据并不足以替代真实数据。因此将文本的检测与识别拆分开来，有利于分别优化两个不同的模型。

如下图所示，目前高德采用的算法框架由文本行检测、单字检测识别、序列识别三大模块构成。文本行检测模块负责检测出文字区域，并预测出文字的掩模用于解决文本的竖直、畸变、弯曲等失真问题，序列识别模块则负责在检测出的文字区域中，识别出相应的文字，对于艺术文本、特殊排列等序列识别模型效果较差的场景，使用单字检测识别模型进行补充。



文本行检测

自然场景中的文字区域通常是多变且不规则的，文本的尺度大小

各异，成像的角度和成像的质量往往不受控制。同时不同采集来源的图像中文本的尺度变化较大，模糊遮挡的情况也各不相同。我们根据实验，决定在两阶段的实例分割模型的基础上，针对实际问题进行了优化。

文本行检测可同时预测文字区域分割结果及文字行位置信息，通过集成 DCN 来获取不同方向的文本的特征信息，增大 mask 分支的 feature 大小并集成 ASPP 模块，提升文字区域分割的精度。并通过文本的分割结果生成最小外接凸包用于后续的识别计算。在训练过程中，使用 online 的数据增广方法，在训练过程中对数据进行旋转、翻转、mixup 等，有效的提高了模型的泛化能力。具体检测效果如下所示：



检测结果示例

目前场景文本检测能力已经广泛应用于高德 POI、道路等多个产品中，为了验证模型能力，分别在 ICDAR2013(2018 年 3 月)、

算法篇-揭秘！文字识别在高德地图数据生产中的演进

ICDAR2017-MLT(2018 年 10 月)、ICDAR2019-ReCTS 公开数据集中进行验证，并取得了优异的成绩。

The figure displays three screenshots from the ICDAR competition interface, each showing a different stage of text detection performance:

- ICDAR2013-文本定位第一**: Shows a screenshot of a building facade with various signs and text elements.
- ICDAR2017-MLT文本定位第一**: Shows a screenshot of a road sign with Korean text.
- ICDAR2019-ReCTS文本定位第三**: Shows a screenshot of a building facade with multiple text elements in different languages.

文本行检测竞赛成绩

文字识别

根据背景的描述，POI 和道路数据自动化生产对于文字识别的结果有两方面的需求，一方面是希望文本行内容尽可能完整识别，另外一方面对于算法给出的结果能区分出极高准确率的部分（准确率大于 99%）。不同于一般文字识别评测以单字为维度，我们在业务使用中，更关注于整个文本行的识别结果，因此我们定义了符合业务使用需求的文字识别评价标准：

- 文本行识别全对率：表示文字识别正确且读序正确的文本行在所有文本行的占比。

- 文本行识别高置信占比：表示识别结果中的高置信度部分（准确率大于 99%）在所有文本行的占比。

文本行识别全对率主要评价文字识别在 POI 名称，道路名称的整体识别能力，文本行识别高置信占比主要评价算法对于拆分出识别高准确率部分的能力，这两种能力与我们的业务需求紧密相关。为了满足业务场景对文字识别的需求，我们针对目前主流的文字识别算法进行了调研和选型。

文字识别发展到现在主要有两种方法，分别是单字检测识别和序列识别。单字检测识别的训练样本组织和模型训练相对容易，不被文字排版的顺序影响。缺点在某些“上下结构”，“左右结构”的汉字容易检测识别错误。相比之下序列识别包含更多的上下文信息，而且不需要定位单字精确的位置，减小因为汉字结构导致的识别损失。

但是，现实场景文本的排版复杂，“从上到下”，“从左到右”排版会导致序列识别效果不稳定。结合单字检测识别和序列识别各自

的优缺点，采用互补的方式提高文字识别的准确率。



单字检测识别和序列识别结果融合

1) 单字检测识别

单字检测采用 Faster R-CNN 的方法，检测效果满足业务场景需求。单字识别采用 SENet 结构，字符类别支持超过 7000 个中英文字符和数字。在单字识别模型中参考 identity mapping 的设计和 MobileNetV2 的结构，对 Skip Connections 和激活函数进行了优化，并在训练过程中也加入随机样本变换，大幅提升文字识别的能力。在 2019 年 4 月，为了验证在文字识别的算法能力，我们在 ICDAR2019-ReCTS 文字识别竞赛中获得第二名的成绩（准确率与第一名相差 0.09%）。

单字检测识别效果



ICDAR2019-ReCTS单字识别第二

Task 1. Character Recognition in the Signboard

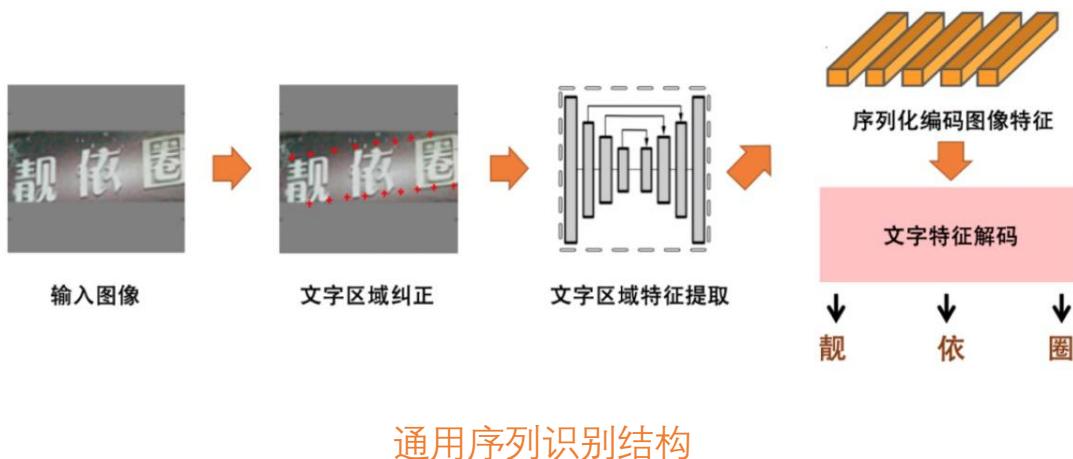
Ranking	User_Id	Affiliation	Accuracy
1	user_0792	MEITI University of Sciences and Technology of China	0.9737
2	user_10431	Alibaba AMAR	0.9729
3	user_10465	JD.com Team, NAVIER/Int Corp	0.9722
4	user_10466	South China University of Technology, The University of Adelaide, Northwestern Polytechnical University, Lenovo, HUAWEI	0.9694
5	user_3347	Tencent (Data Platform Precision Recommendation)	0.9612
6	user_4330	• Huazhong University of Science and Technology (Bohan Li, et al.)	0.9473
7	user_10467	• Tsinghua University	0.9460
8	user_10468	• JD.com Team, Alibaba Cloud (Data Lab: Image Recognition)	0.9459
9	user_10775	• Institute of Information Engineering, Chinese Academy of Sciences, University of Science & Technology Beijing	0.9355
10	user_10981	• XJU University of Technology (Zhu Hong Digital Image Processing Laboratory)	0.9347
11	user_5409	• Harbin Institute of Technology	0.9319
12	user_10469	• Tsinghua University	0.9309
13	user_10112	• Zhejiang University	0.9302
14	user_5425	Creditlink	0.9292
15	user_3284	• University of Chinese Academy of Sciences	0.9179
16	user_3074	• Sichuan University	0.9057
17	user_10470	• Tsinghua University	0.8954
18	user_10662	Wynnsearch Info Tech., Ltd.	0.8882
19	user_9423	• Institute of Information Engineering, Chinese Academy of Sciences	0.8886
20	user_10798	• Huazhong University of Science and Technology (Shan Xu, et al.)	0.8833
21	user_10986	Junior	0.8668
▼ means student contestants			

单字检测识别效果图

2) 文本序列识别

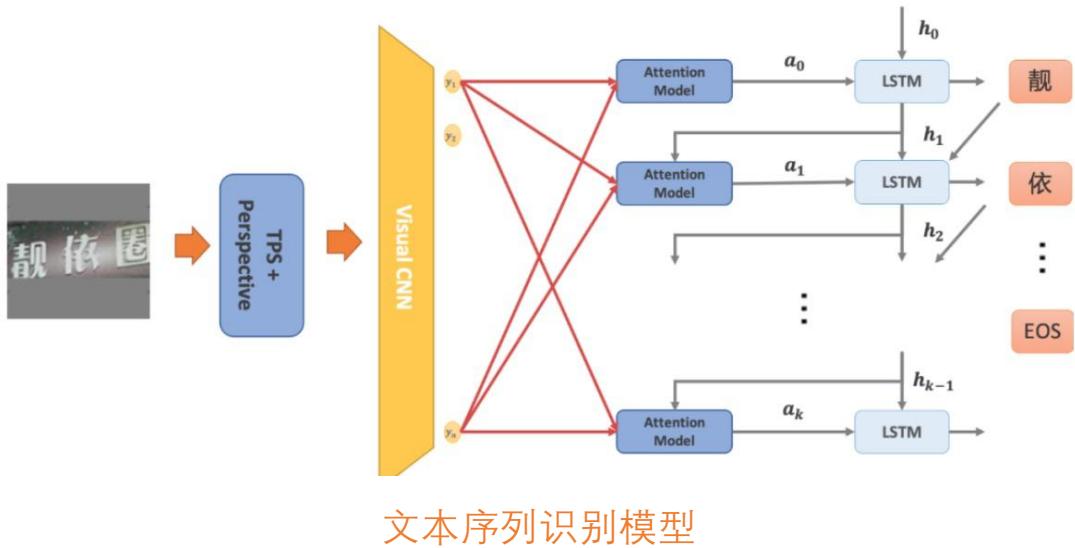
近年来，主流的文本序列识别算法如 Aster、DTRT 等，可以分解为文字区域纠正，文字区域特征提取、序列化编码图像特征和文字特征解码四个子任务。文字区域纠正和文字区域特征提取将变形的文本行纠正为水平文本行并提取特征，降低了后续识别算法的识别难度。序列化编码图像特征和文字特征解码

(Encoder—Decoder 的结构)能在利用图像的纹理特征进行文字识别的同时，引入较强的语义信息，并利用这种上下文的语义信息来补全识别结果。



在实际应用中，由于被识别的目标主要以自然场景的短中文本为主，场景文本的几何畸变、扭曲、模糊程度极为严重。同时希望在一个模型中识别多个方向的文本，因此我们采用的是的 TPS-Inception-BiLSTM-Attention 结构来进行序列识别。主要结

构如下所示：



对于被检测到的文本行，基于角点进行透视变换，再使用 TPS 变换获得水平、竖直方向的文本，按比例缩放长边到指定大小，并以灰色为背景 padding 为方形图像。这一预处理方式既保持了输入图像语义的完整，同时在训练和测试阶段，图像可以在方形范围内自由的旋转平移，能够有效的提高弯曲、畸变文本的识别性能。将预处理完成的图像输入 CNN 中提取图像特征。再使用 BiLSTM 编码成序列特征，并使用 Attention 依次解码获得预测结果。

如下图所示，这一模型通过注意力机制在不同解码阶段赋予图像特征不同的权重，从而隐式表达预测字符与特征的对齐关系，实现在一个模型中同时预测多个方向文本。文本序列识别模型目前已覆盖英文、中文一级字库和常用的繁体字字库，对于艺术文本、

模糊文本具有较好的识别性能。

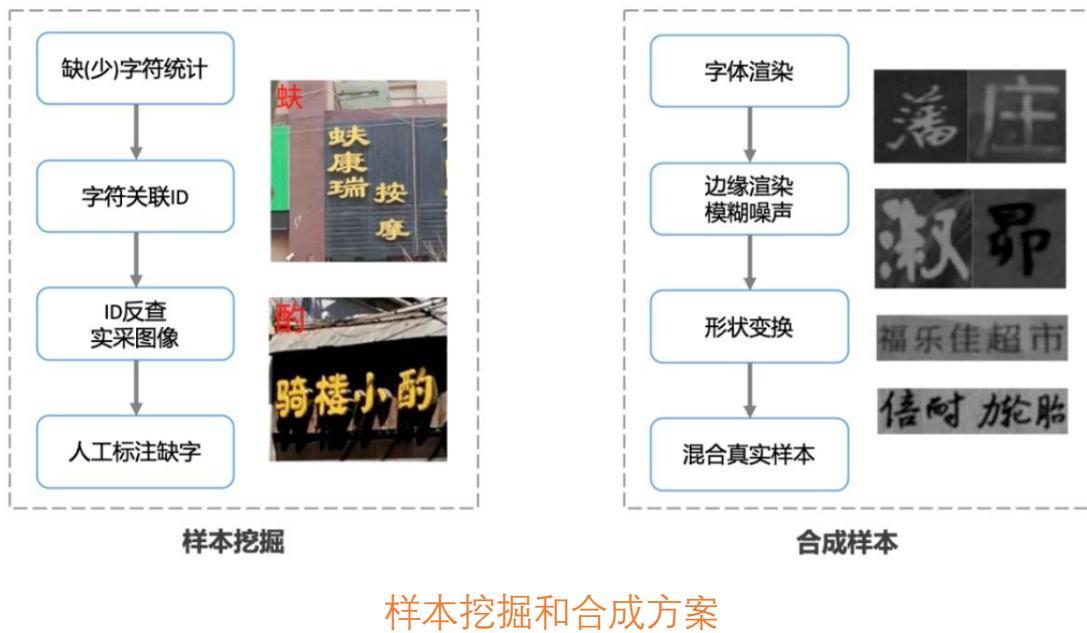


序列识别效果

3) 样本挖掘&合成

在地图数据生产业务中经常会在道路标志牌中发现一些生僻的地点名称或者在POI牌匾中发现一些不常见的字甚至是繁体字，因此在文字识别效果优化中，除了对于模型的优化外，合理补充缺字、少字的样本也是非常重要的环节。为了补充缺字、少字的样本，我们从真实样本挖掘和人工样本合成两个方向入手，一方面结合我们业务的特点，通过数据库中已经完成制作的包含生僻字的名称，反向挖掘出可能出现生僻字的图像进行人工标注，另一方面，我们利用图像渲染技术人工合成文字样本。实际使用中，

将真实样本和人工合成样本混合使用，大幅提升文字识别能力。



样本挖掘和合成方案

4. 文字识别技术小结

高德文字识别算法通过对算法结构的打磨，和多识别结果的融合，满足不同使用场景的现实需要。同时以文字识别为代表的计算机视觉技术，已广泛应用于高德数据自动化生产的各个角落，在部分采集场景中，机器已完全代替人工进行数据的自动化生产。POI数据中超过 70%的数据都是由机器自动化生成上线，超过 90%的道路信息数据通过自动化更新。数据工艺人员的技能极大简化，大幅节约了培训成本和支出开销。

5. 未来发展和挑战

目前高德主要依赖深度学习的方式解决场景文字的识别问题，相

对国外地图数据，国内汉字的基数大，文字结构复杂导致对数据多样性的要求更高，数据不足成为主要痛点。

另外，图像的模糊问题往往会影响自动化识别的性能和数据的制作效率，如何识别模糊和对模糊的处理也是高德的研究课题之一。我们分别从数据、模型设计层面阐述如何解决数据不足和模糊识别的问题，以及如何进一步提高文字识别能力。

数据层面

数据问题很重要，在没有足够的人力物力标注的情况下，如何自动扩充数据是图像的一个通用研究课题。其中一个思路是通过数据增广的方式扩充数据样本。Google DeepMind 在 CVPR 2019 提出 AutoAugment 的方法，主要通过用强化学习的方法寻找最佳的数据增广策略。另一种数据扩充的解决办法是数据合成，例如阿里巴巴达摩院的 SwapText 利用风格迁移的方式完成数据生成。

模型层面

模糊文本的识别

模糊通常造成场景识别文本未检测和无法识别的问题。在学术界超分辨率是解决模糊问题的主要方式之一，TextSR 通过 SRGAN 对文本超分的方式，还原高清文本图像，解决模糊识别的问题。

对比 TextSR，首尔大学和马萨诸塞大学在 Better to Follow 文中提出通过 GAN 对特征的超分辨率方式，没有直接生成新的图像而是将超分辨率网络集成在检测网络中，在效果接近的同时，由于其采用 End-to-End 的模式，计算效率大幅提高。

文字语义理解

通常人在理解复杂文字时会参考一定的语义先验信息，近年来随着 NLP（Natural Language Processing）技术的发展，使得计算机也拥有获得语义信息的能力。参考人理解复杂文字的方式，如何利用语义的先验信息和图像的关系提高文字识别能力是一个值得研究的课题。例如 SEED 在 CVPR 2020 提出将语言模型添加到识别模型中，通过图像特征和语义特征综合判断提高文字识别能力。

其他发展

除此之外，从云到端也是模型发展的一个趋势，端上化的优势在于节约资源，主要体现在节约上传至云端的流量开销和云端服务器的计算压力。在端上化设计上，针对 OCR 算法的研究和优化，探索高精度、轻量级的检测和识别框架，压缩后模型的大小和速度满足端上部署的需要，也是我们今后需要研究的一个课题。

6. 参考文献

1. Liao M et al. Textboxes++: A single-shot oriented scene text detector[J]. IEEE transactions on image processing , 2018.
2. Lyu P , Liao M , Yao C , et al. Mask TextSpotter: An End-to-End Trainable Neural Network for Spotting Text with Arbitrary Shapes[J]. 2018.
3. Zhou X , Yao C , Wen H , et al. EAST: An Efficient and Accurate Scene Text Detector[J]. 2017.
4. Shi B , Bai X , Yao C . An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition[J]. ieee transactions on pattern analysis & machine intelligence , 2017 , 39(11):2298–2304 .
5. Wojna Z , Gorban A N , Lee D S , et al. Attention-Based Extraction of Structured Information from Street View Imagery[C]// 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR). IEEE , 2018.

6.Li H , Wang P , Shen C . Towards End-to-end Text Spotting with Convolutional Recurrent Neural Networks[J]. 2017.

招聘

高德地图视觉技术中心火热招聘中：计算机视觉、图像识别、点云、三维、AR、视频等算法类职位，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德算法工程一体化实践和思考

作者：承道

背景

随着高德地图业务的快速开展，除了导航本身的算法业务外，其他中小型业务对算法策略的需求越来越多、越来越快，近两年参与的一些新项目从算法调研到应用上线都在一周级，例如与共享出行相关的各种算法服务，风控、调度、营销等各个体系的业务需求。类似于传统导航中成熟的长周期、高流量、低时延的架构和开发方式已无法满足此类业务初期的快速试错和优化改进诉求，找到合适的为业务赋能的算法服务方式就变得势在必行。

在落地实施的过程中，采用一体化架构。所谓的一体化是指整个算法、工程一体化，涉及数据、系统等全链路打通，实现数据流的系统化流动；算法业务调研兼顾工程服务开发，测试、验证过程自动化、智能化，从而形成业务闭环，推动业务的快速迭代。

项目初期，需要快速试错和策略优化。此时，业务需求的 QPS 往往不高 (<1k)，因此，传统的应用开发和部署方式，一方面拖慢了业务节奏，另一方面浪费了大量资源。

在此过程中，我们希望达到的目标就是离线策略调研即服务逻辑开发，离线调研完成即服务化完成，这样就能够显著降低算法调研到策略上线的时间。因为性能 (QPS、RT) 压力不大，离线用 Python 进行快速的开发就成为可能。

从长期看，随着业务逐步成熟，算法快速组合、服务调用量和服务性能成为衡量算法服务重要的评价标准，此时合理的优化就应该被向前推进，例如核心算法逻辑高性能化将会变成重要的工作。

因此，算法工程一体化的建设过程也就是满足业务从初创期到成熟期迭代的过程。

系统总体逻辑架构



整个平台类似于上图所示，主要由几个逻辑部分组成：

- 统一接入网关服务
- 业务算法透出服务
- 算法模型及代码管理服务
- 质量保障体系

注：GBFC 为数据服务层，主要用于获取各种算法所需要的数据和特征，让业务算法服务达到无状态的条件，同时也便于数据在各条业务线的共享和共建。

统一接入网关

网关服务目的是将各种算法 API 进行隔离，提供原子服务和组合服务。业务端需要调用各种算法如价值判

断、风险预估、营销推荐时，统一网关对各个业务提供统一算法 API，避免各个服务重复调用。算法网关对算法进行统一监控，包括服务性能，接口可用性等，同时对数据进行统一收集，便于数据管理和特征生产，进行实时在线学习，提升用户体验，保障算法效果。

网关服务一方面可以进行一些共用的预处理操作，例如鉴权、路由、限流、降级、熔断、灰度、AB、陪跑等，用于保障服务的可用性和扩展性。同时又能进行服务组合，例如语音识别、图像处理等，使得各个算法服务能够有机的结合在一起，这样使得业务算法层只需要维护原子性服务，即可进行复杂的业务处理。

因此，网关服务必然需要一个灵活、弹性、轻量、无状态的算法业务层的支撑，在技术选型方面，目前火热的 Serverless 架构刚好能够很好的满足上述需求。

Serverless 架构上的业务算法服务

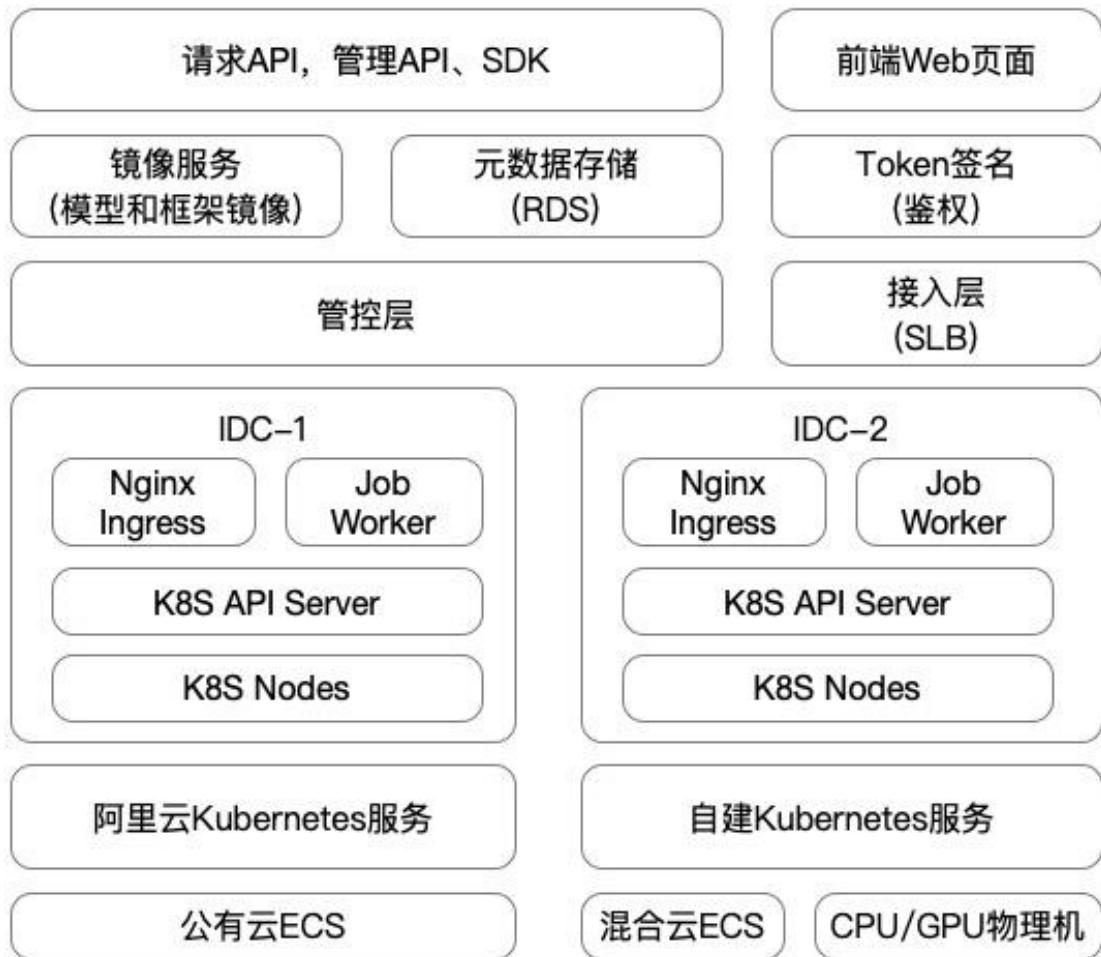
2009 年，伯克利以独特的视角定义了云计算，尤其是最近的四年，这篇文章被大量引用，其中的观点刚好非常契合业务初期的技术场景，比如减少服务化的

工作，只关心业务逻辑或算法逻辑，实现快速迭代、按需计算等。

2019 年，伯克利又进一步提出：Serverless 所提供的接口，简化了云计算的编程，其代表了程序员生产力的又一次变革，一如编程语言从汇编时代演变为高级语言时代。

在目前关于 Serverless 的探索中，FaaS 基本被认为是最佳范例，这与其自身特点有关。FaaS 非常轻量级，无状态，运行快，对于纯粹的无状态应用特别合适，虽然在冷启动层面存在一些瓶颈，但这种架构还是解决了很多问题，而业务初期的算法快速实践，刚好与这种架构的特点相契合。

在 Serverless 架构的基础之上，我们对算法服务按照下图的方式进行了部署。



按照上图所示，算法服务在本地开发完成后，可以直接作为 Function 进行发布，即之前所说的离线策略调研完成的同时就是服务代码完成。

这个特性也很好的解决了算法同学和业务脱节的问题，很多算法同学无法独立完成整个工程服务开发，需要将代码提交给工程人员进行整合、包装、发布，但这种协作方式在整个业务链条中是不合理的。

一方面算法同学无法独立完成业务支撑，另一方面，工程同学不仅要处理逻辑调用，还需要花时间去了解当前算法实现方式、原理、所需数据、异常情况等各种问题，经常是一个相对简单的算法服务，会议和沟通占据了绝大多数时间，因此引入 FaaS 不仅简化了这个业务开发流程，同时也让算法部署尽量是原子化，方便业务间组装复用。

我们在实现的路径上，首先完成了 Python 的运行时环境的构建工作，能够将 Python 代码及相关模型直接服务化，模型部分支持 PMML, TensorFlow 等，这样就实现了业务初期所需要的快速迭代，减少试错成本的诉求。

此后会逐步完善基于 Golang, Java 等运行时环境，选择 Golang 的原因是对于并行性支持良好，不仅可以实现业务所需要的性能诉求，同时又保留着开发相对简单的特点。

经过算法原子化、函数化后，就赋予了算法同学独立负责线上业务的能力，但也带了几个严肃的问题：服务的稳定性，工程的质量，服务的正确性。如果简单

的把这些扔给算法同学，就仅是工作量的转移，并且还可能引起整个业务的宕机风险。因此，质量保障体系建设就变成了重要的一环。

质量保障体系建设

很多人会认为，要做质量保障，就是提交到测试人员进行测试或回归测试，其实不然。前两部分省却的人工成本被转移到测试同学的身上，因为算法同学的工程化能力相对偏弱，是不是就要引入更多的测试同学做验证？

如果抱着这种思维，那么，业务的迭代速度依然无法快速起来。

因此就必须考虑：这种质量保障能否完全的自动化呢？答案是肯定的。

在策略的调研过程中一般会经历以下几个步骤，1. 数据分析；2. 算法设计；3. 数据验证；4. 人工／自动评测；5. 策略迭代重复步骤1-4。在这个过程中，刚好包含了质量保障体系所需要的数据、数据集、测试集及验证集。

算法同学可以通过使用三个集合实现自动化的压测流程，输出 QPS、RT、一致率等相关信息，使用数据集，实现稳定性测试；使用测试集，实现了逻辑正确性验证；通过验证集实现了效果测试。

当然，上述只是质量保障的一部分。一般业务快速迭代的过程中，常常需要进行 AB 验证，或者是全量陪跑验证，在过程实施中通过引流和过程链，我们实现了全量陪跑的过程，对待上线的算法实现了全方位的质量评估。

小结

算法、工程的统一化建设基本实现了业务初期的快速迭代需求。在项目开展的过程中，对工程同学的挑战除了来自于工程化实践外，同时也需要对业务所处的阶段要有清晰的认识。

当前情况下，算法和工程之所以可以独立开展，有一个必要的前提是数据和算法分离，也就是算法服务是无状态、函数化的。正是因为如此，也需要花费不少的时间在特征服务层的建设上。同样，特征服务层的建设也可以遵循相似的逻辑实现共建、共享。

最后需要再次说明的是，算法工程一体化的架构设计能够基本满足业务类算法的诉求。但对于一些对计算量要求巨大的 AI 项目，频繁的数据获取导致的算力、功耗瓶颈已经明显制约算法创新。业内正在通过将数据存储单元和计算单元融为一体，实现计算存储一体化的硬件架构革新，突破 AI 算力瓶颈。

招聘

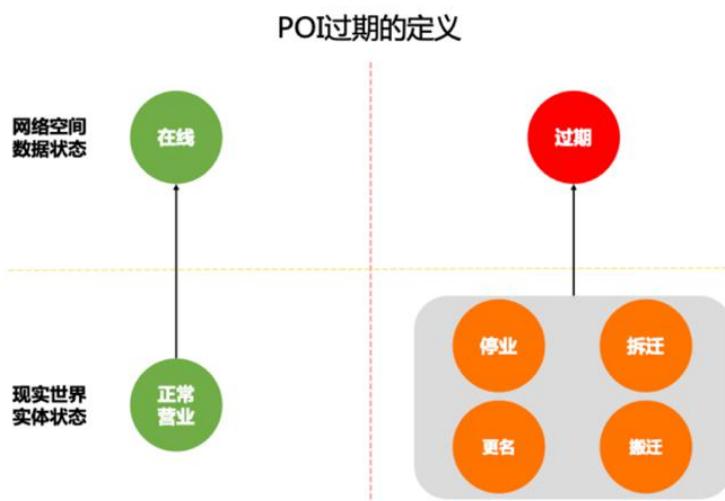
阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位 地 点：北 京。欢 迎 投 递 简 历 到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

深度学习在高德 POI 鲜活度提升中的演进

作者：懿林

1. 导读

高德地图拥有着数千万的 POI (Point of Interest) 兴趣点，如学校、酒店、加油站、超市等。其中伴随着众多 POI 创建的同时，会有大量的 POI 过期，如停业、拆迁、搬迁、更名。这部分 POI 对地图鲜活度和用户体验有着严重的负面影响，需要及时有效地识别并处理。



由于实地采集的方式成本高且时效性低，挖掘算法则显得格外重要。其中基于趋势大数据的时序模型，能够覆盖大

部分挖掘产能，对 POI 质量提升有着重要意义。

过期 POI 识别本质上可以抽象为一个数据分布非对称的二分类问题。项目中以多源趋势特征为基础，并在迭代中引入高维度稀疏的属性、状态特征，构建符合业务需求的混合模型。

本文将对深度学习技术在高德地图落地的过程中遇到的业务难点，和经过实践检验的可行方案进行系统性的梳理总结。

2. 特征工程

过期挖掘的实质是感知伴随 POI 过期而发生的变化，进行事后观测式挖掘，一般都会伴随着 POI 相关活跃度的下降。因此时序模型的关键是构建相关联的特征体系。同时在实践中我们也构造了一些有效的非时序特征进行辅助校正。

2.1 时序特征

时序特征方面，建立了 POI 和多种信息的关联关系，并分别整合为月级的统计值，作为时序模型的输入；时间序列窗口方面，考虑到一些周期性的规律的影响，需要两年以上的序列长度来训练模型。

2.2 辅助特征

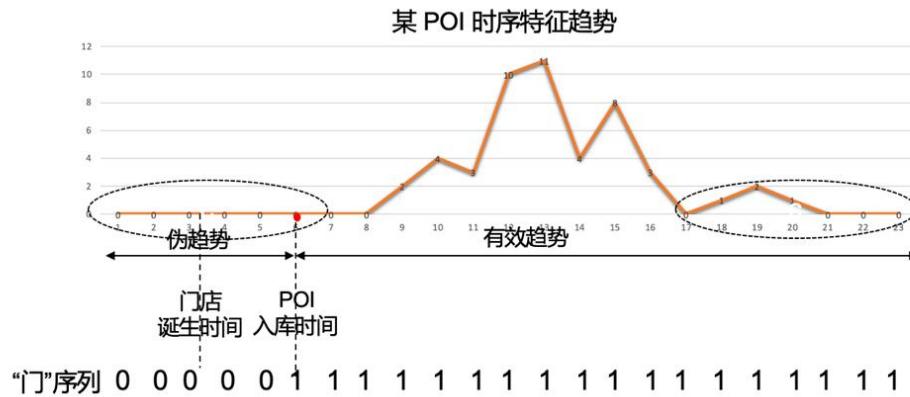
辅助特征方面，首先是将人工核实历史数据进行有效利用。方式是构造一个时间序列长度的 One-Hot 向量，将最后一次人工核实存在的月份标记为 1，其他月份为 0。人工核实存在表示该时间结点附近过期概率较低，若人工更新在趋势下降之后，说明趋势表征过期的概率不高。

其次，调研发现不同行业类型的 POI 有着不同的过期概率，如餐饮和生活服务类过期概率较高，而地名或公交站点等类型则相对低很多。因此将行业类型编号构建为一个时间序列长度的等值向量，作为静态辅助特征。

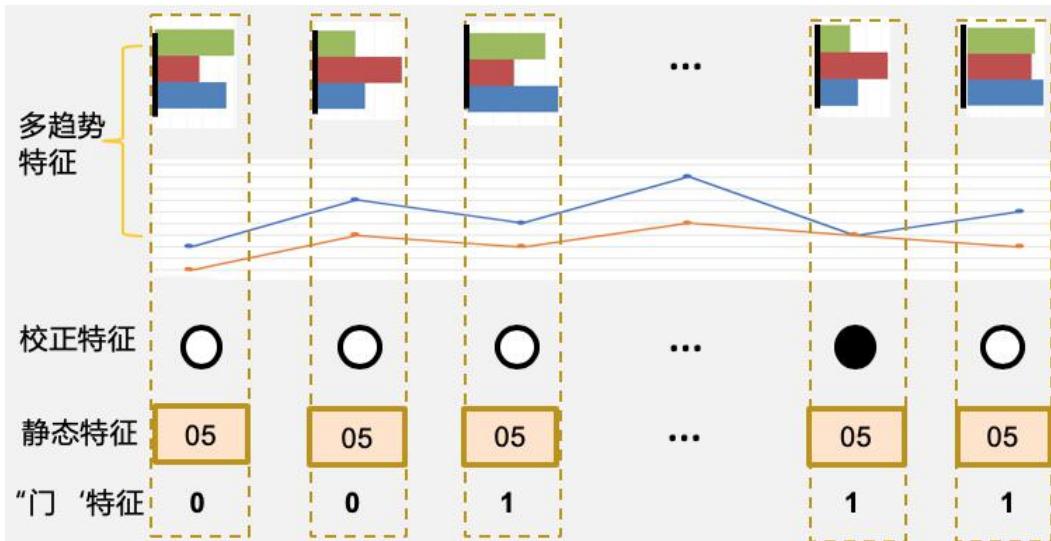
第三种辅助特征是在分析业务中的漏召回问题时总结构造的。发现有相当部分的新诞生 POI，其入库创建后至今的时长于序列长度。意味着这部分序列前期存在较多数值为零的伪趋势，会对尾部的真实下降趋势造成干扰从而误判。对此提出了两种优化思路：

- 采用可变长度的 RNN 模型，只截取 POI 创建时间之后部分的序列作为输入。

- 序列长度不变，添加一维“门”序列特征，序列在 POI 创建时间之前的部分数值为 0，之后为 1。如图所示。



对比采用第二种方案效果更优。考虑到我们只有 POI 的入库创建时间信息，而不了解门店的具体诞生时间，直接按入库时间截取序列，会造成门店诞生和 POI 创建时间段内的特征信息损失；而添加“门”序列则可以在保持信息完备的同时约束高可信区间。最后构建的混合特征示意图如下所示。

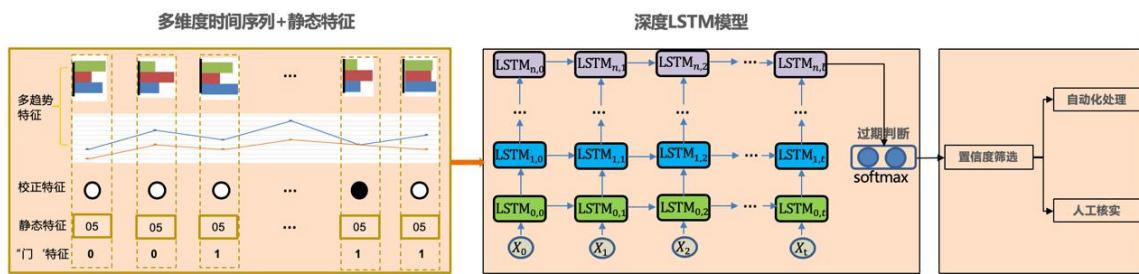


3. RNN 阶段

循环神经网络（RNN, Recurrent Neural Network）凭借强大的表征能力在序列建模问题上有非常突出的表现，业务中采用了其变种模型 LSTM。

3.1 RNN1.0

以前述的时序特征和辅助特征为基础，我们采用多层 LSTM 搭建了第一版 RNN 过期挖掘模型，结构如图所示。主要逻辑为，将逐时间点对齐后的特征输入到深度 LSTM 中，在网络最后时刻的输出后，接入一层 SoftMax 计算过期概率。最后根据结果匹配不同的置信度区段，分别进行自动化处理或人工作业等任务。模型初步验证了 RNN 在过期趋势挖掘领域落地的可行性和优势。



3.2 RNN2.0

高德地图基于导航、搜索或点击等操作频度对 POI 进行了热度排名。头部的热门 POI 如果过期但未及时发现对用户体验的伤害更大。2.0 版本模型升级的主要目标便是进一步提升头部热门段位的过期 POI 发现能力。

分析发现热门 POI 的数据分布相比尾部有较大差异性。头部 POI 的数据量丰富，且数值为 0 的月份很少；相反尾部 POI 则数据稀疏，且有数值月份量级可能也仅为个位数。对于这种头部效应特别明显的状况，单独开发了高热度段特征的头部 RNN 模型，实现定制化挖掘。

另一方面，对于单维度特征缺失的情况，也区分热度采用了不同的填充方式。头部 POI 特征信息丰富，将缺失维度补零让其保持“静默”防止干扰；而尾部特征稀疏，本身已有较多零值，需要插值处理使缺失特征和整体保持相近趋势。方法为将其他维度的数据规范化处理后，采用加权的方式得到插值。

	T0	T1	...	Tt-3	Tt-2	Tt-1	Tt	
维度A	a ₀	a ₁	...	a _{t-3}	a _{t-2}	a _{t-1}	a _t	数值规范化 $a'_i = \frac{a_i}{\sqrt{\sum_{i=0}^t a_i^2}}$
维度B	b ₀	b ₁	...	b _{t-3}	b _{t-2}	b _{t-1}	b _t	头部模型 缺失值填充 $c'_i = 0$
维度C	*	*		*	*	*	*	尾部模型 缺失值填充 $c'_i = \frac{w_1 a'_i + w_2 b'_i + w_3 d'_i}{w_1 + w_2 + w_3}$
维度D	d ₀	d ₁	...	d _{t-3}	d _{t-2}	d _{t-1}	d _t	

2.0 版模型对头部和尾部的召回能力都有提升，对头部的自动化能力提升尤为明显。

4. Wide&Deep 阶段

RNN 模型能够充分发掘时序特征的信息，但特征丰富度不足成为制约自动化能力进一步提升的瓶颈。因此整合业务中的其他数据，从多源信息融合角度升级模型便成为新阶段的工作重点。主要的整合目标包括非时序的静态信息和状态信息，以及新开发的时序特征信息。

模型升级主要借鉴了 Wide&Deep 的思想，并做了很多结合业务实际情况的应用创新。首先我们要把已有的 RNN 模型封装为 Deep 模块后和 Wide 部分联合，相当于重新构建了一个混合模型，涉及到模型结构维度的整合。其次，既有 Deep 的时序信息，又有 Wide 部分的实时状态信息，涉及到数据时间维度的整合。最后是 Wide 部分包含大量的不可

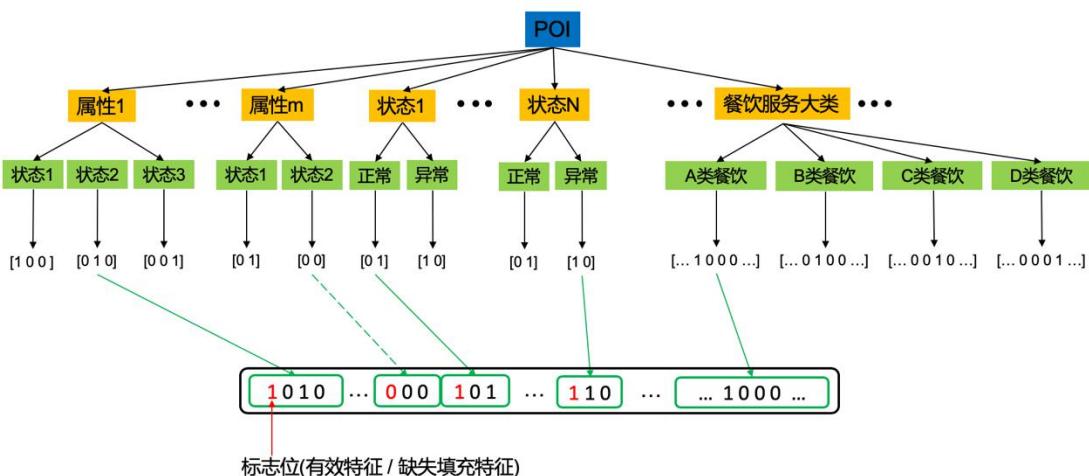
量化或比较的类型特征需要编码表征处理，涉及到数据属性维度的整合。

4.1 Wide & LSTM

- 特征编码

我们将非时序特征经过编码后构建 Wide 模块。主要包括属性、状态，以及细分行业类型三种特征。

考虑到某些 POI 属性存在缺失的情况，故编码中第 1 位表示特征是否存在的标志位，后面则为 One-Hot 编码后的对应的属性类型；对于状态特征，同样有一位表示是否特征缺失的标志位，而后面的 One-Hot 编码则表示最新时刻的状态类型；由于不同行业类型有着不同的背景过期率，我们将细分的行业类型做 One-Hot 编码后作为第三种特征。最后将各特征编码依次连接，得到一个高维度的稀疏向量。特征编码的过程如图所示。



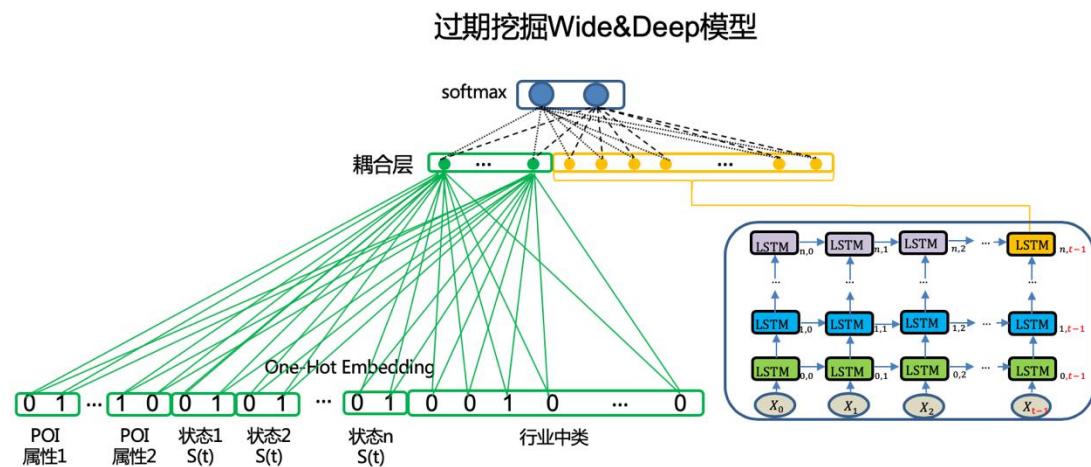
• 特征耦合

特征完备之后，将各类特征耦合及模型训练便成为关键。

耦合点选在了 SoftMax 输出的前一层。对于 Deep 部分的 RNN 结构，参与耦合的便是最后时间节点的隐层。

而对于 Wide 部分的高维度稀疏向量，我们通过一层全连接网络来降维，便得到 Wide 部分的隐层。最后将两部分的隐层连接，输出到 SoftMax 来计算过期概率。

模型采用同步输入 Wide 和 Deep 部分特征的方式联合训练，并调节两部分的耦合隐层的维度来平衡两部分的权重。过期挖掘场景的 Wide & LSTM 模型结构如图所示。

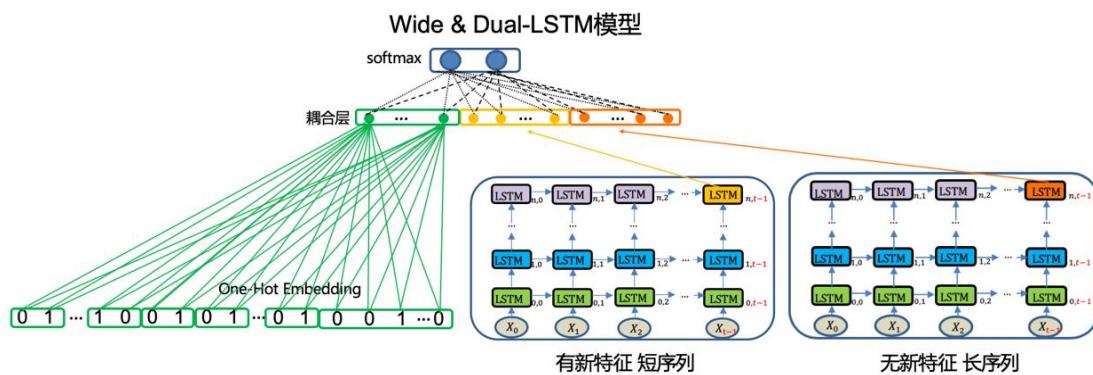


模型经过多次迭代优化后稳定投产，已成为过期挖掘业务中覆盖行业广、自动化解题能力突出的综合性模型。

4.2 Wide & Dual-LSTM

在做模型升级迭代的同时，基础特征的建设工作也在同步进行。在扩充新的趋势特征的时候面临这样一个问题，新特征维数较多且时间序列较短，这样将长时序特征和短时序特征逐时间点匹配时会出现很大部分的数值缺失。

由于新特征缺失部分较多且维度较大，缺失值填充的负面影响会过于严重而不适合采用。项目中采用了分而治之的方案，分别建立两个 RNN 模块，其中长 RNN 模块输入无新特征的长序列，短 RNN 模块输入有新特征的短序列，最后将双 RNN 的 Hidden 层和 Wide 部分一起耦合，得到了 Wide & Dual-RNN 模型，结构如图所示。



双 RNN 结构能够很好地将新特征融入到现有模型并提升判断准确率，不足的地方是结构较复杂影响计算效率。故后

期进行了新阶段的研发，采用更灵活的时序模型 TCN 进行迭代。

4.3 Wide & Attention-TCN

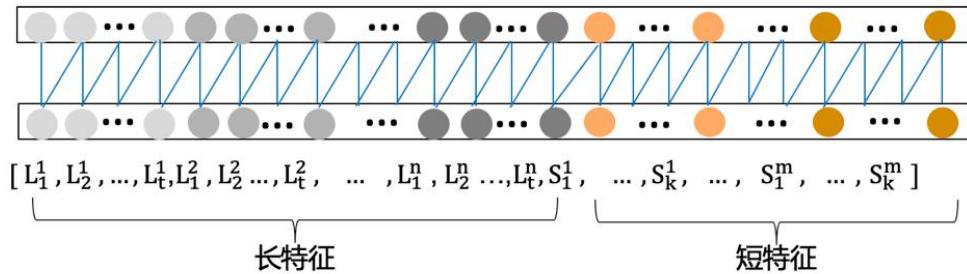
TCN 主要有如下三方面优点使其能胜任时间序列的建模：

首先，架构中的卷积存在因果关系，即从未来到过去不会存在信息泄漏。其次，卷积架构可以将任意长度的序列映射到固定长度的序列。另外，它还利用残差模块和空洞卷积来构建长期依赖关系。

性能对比上，TCN 可以将时间序列作为向量并行化处理，相比 RNN 的逐时间点顺序计算的方式有更快的计算速度。此外，TCN 可以输入延展成一维的序列，从而避免了特征需要逐时间点对齐。

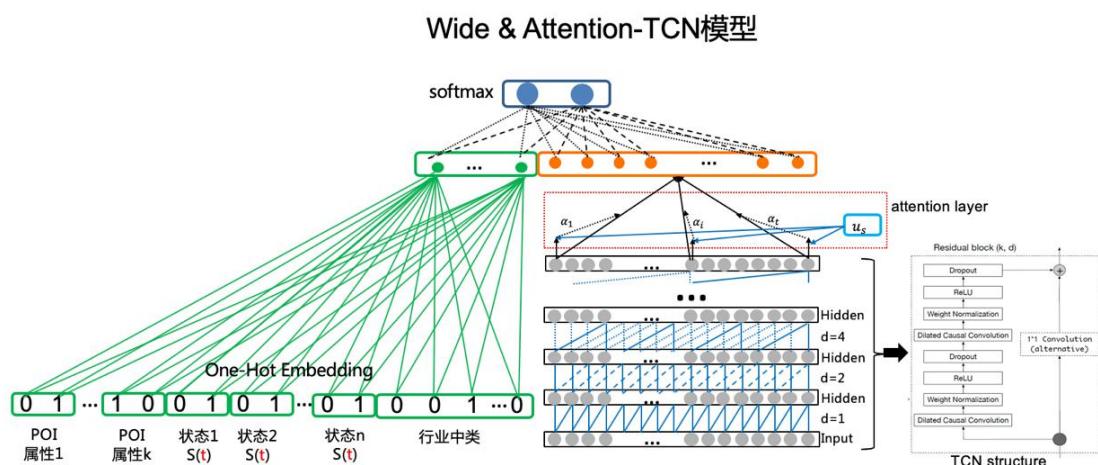
因此，在验证了 Wide&Deep 的思路有效后，我们尝试将 Deep 部分的 RNN 结构升级为 TCN。

首先，对于输入部分的特征进行了 Flatten 处理，即将每个维度的时间序列依次首尾相连，如图所示，拼接成为一个长向量后作为输入。这样便实现了长特征和短特征的有效整合。



其次对于输出结构，引入序列维度的 Attention 机制进行优化。主要思想是不再只读取序列最后节点的隐向量的浓缩信息，而是对所有序列节点的隐向量信息加权处理后，得到汇总的隐向量信息，使所有节点的学习结果得到充分利用。

最后，将 Attention-TCN 后得到的汇总隐向量和 Wide 部分的隐层进行耦合，得到的 Wide&Attention-TCN 模型结构如图所示。



通过引入新的轻量 TCN 时序模型和 Attention 机制，新的模型性能有了进一步提高，但调优过程相对 RNN 更加复杂。

多轮参数调整与结构优化后，最终落地版本与 Wide & Dual-LSTM 版相比，计算效率和业务扩招回能力均有可观提升。

5. 总结与展望



深度学习在过期挖掘场景中的落地，经历了不断摸索尝试、总结问题、优化方案、验证效果的迭代演进的过程。期间以提升过期发现能力为核心目标，对特征扩展、特征构造和模型结构优化的角度都进行了探索，并总结了如上的业务场景落地经验。其中，丰富可靠的特征、合适的特征表征方式和符合场景的模型结构设计是提升业务问题解决能力的关键。

当前模型主要是基于信息和趋势进行宏观性的规律总结，并判断具备这类特征情况下的 POI 过期的概率。而现实生活中 POI 的具体地理环境、自身经营状况、周边竞争态势等个性化因素的影响往往不可忽略。因此，未来规划将综合考虑整体规律性特征和个体差异性，实现精细化挖掘。

6. 参考文献

1. Sepp Hochreiter and Jurgen Schmidhuber, Long Short-Term Memory, Neural Computation 1997
2. B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. CoRR, abs/1511.06939, 2015
3. Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, GlenAnderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide&Deep learning for recommender systems. In Proc. 1st Workshop on Deep Learning for Recommender Systems, pages 7 – 10, 2016.

4.Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems. ACM, 191 – 198, 2016.

5.Bai S , Kolter J Z , Koltun V . An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling[J]. 2018.

6.Pappas N , Popescu-Belis A . Multilingual Hierarchical Attention Networks for Document Classification[J]. 2017.

招聘

阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位 地 点：北 京。欢 迎 投 递 简 历 到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

混合时空图卷积网络：能“推导”未来路况的智能算法

作者：高德技术整理

2020 云栖大会于 9 月 17 日 -18 日在线上举行，阿里巴巴高德地图携手合作伙伴精心组织了“智慧出行”专场，为大家分享高德地图在打造基于 DT+AI 和全面上云架构下的新一代出行生活服务平台过程中的思考和实践，并重点分享了「高精地图、高精算法、智能时空预测模型、自动驾驶、AR 导航、车道级技术」等话题。

「高德技术」把本场讲师分享的主要内容整理成文并陆续发布出来，本文为其中一篇。

查看：[【演讲视频播放地址】](#)

阿里巴巴高级算法专家冀晨光带来的课题是《混合时空图卷积网络：更精准的时空预测模型》。冀晨光分享了高德提出的时空图卷积算法，巧妙利用海量用户的导航规划信息，“推导”出未来拥堵状况，显著提升

预测准确度，并重点介绍了这一业界领先的技术及其在高德业务中的应用。

作为一款国民级出行生活服务平台，高德拥有 5.3 亿+月活用户，在过去一年间，为出行用户节省至少 19.3 亿+小时拥堵时间，创造了巨大的社会效益。这里广为大家所熟知的躲避拥堵功能背后的核心技术，就是交通路况预测算法。

冀晨光的分享主要分三部分展开。

- 路况预测是什么。
- 混合时空图卷积网络：能“推导”未来路况的算法。
- 应用前瞻：从“路况预测”到“交通调度”。

路况预测是什么

下面的三幅图，展示了北京西单金融街附近的区域，在三个相邻时间点上的交通状况。其中绿色代表畅通，黄色代表缓行，红色代表拥堵。假设现在是 18 点整，路况预测的目标就是预估未来时刻上的交通状况，例

如半小时之后、18:30 的路况。从三个时刻的路况状态及彼此的关联中可以观察到，拥堵会在时空上演化和传播，路况预测就是要对这背后的规律进行精确的刻画和建模。



路况预测是什么

某周五，北京西单金融街附近的路况演化

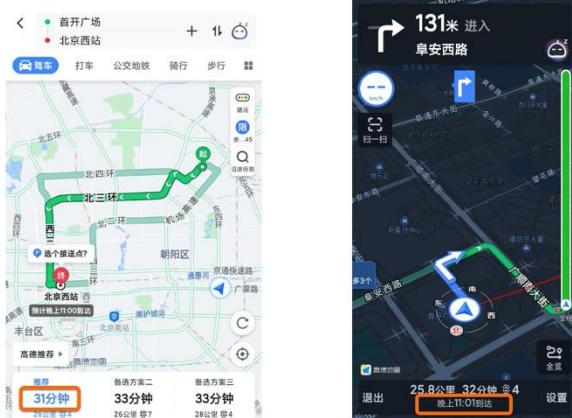


路况预测技术支撑了高德地图很多的核心功能。冀晨光举了两个代表性的案例。第一个案例是路线的旅行时间预估，术语上叫做 ETA。

下方左图是驾车路线规划页面，右图是导航的页面。其中红框高亮的部分就是预估的 ETA。ETA 是路线排序的重要因子，ETA 预估的准确与否，直接决定了能否帮助用户有效躲避拥堵。

路况预测是什么

应用案例一：路线旅行时间预测（ETA, Estimated time of arrival）



第二个案例是随时间推演的路线规划。北京有外地小客车早晚高峰限行的政策，命中该政策的车在 7 到 9 点间不能进入五环。假设有一个这样的车在 6:45 分出发。如下图左图所示，常规路线规划只能基于当前时刻的限行信息算路，而 6:45 分时限行政策尚未生效，所以算出的路线就是穿越五环区域的路线，会导致用户在 7 点后违章。

为了解决这个问题，高德设计了随时间推演的路线规划算法，这个算法具有未来视角，能帮助用户避让未来即将来到的限行，如下图右图所示，推演路线会引导用户在 7 点前驶出五环，走六环到达目的地。因此未来路况预测的是否精准，会影响到对用户驶离五环时刻的判断，也就会直接影响到用户是否违章。

路况预测是什么

应用案例二：随时间推演路线规划（Time dependent routing）

北京市限行政策：五环及以内外地机动车早晚高峰限行，限行时间：工作日7:00-9:00, 17:00-20:00限行



在如何预测准未来路况的问题上，业界主要有两类方法。一类是**交通仿真**，在车辆行驶的起终点信息之上，结合交通动力学理论，联合仿真、预估车辆的行驶路径和交通路况。这类方法属于知识驱动的方法。另一类是**数据驱动的方法**，通过训练模型学习历史路况和未来路况之间的统计关联进而进行预测。这类方法是目前各大出行科技公司所主要依赖的方法。

业界现有的数据驱动方案，主要以历史交通状况为特征，辅助以部分事件类特征，例如异常天气、体育赛事等，常常难以预测准拥堵的起始时刻，导致预测延迟的问题。以下图中右图为例，横轴代表一天的不同时刻，纵轴代表一条道路的旅行时间。红色曲线代表道路的实际旅行时间，即真值。绿色曲线代表模型提

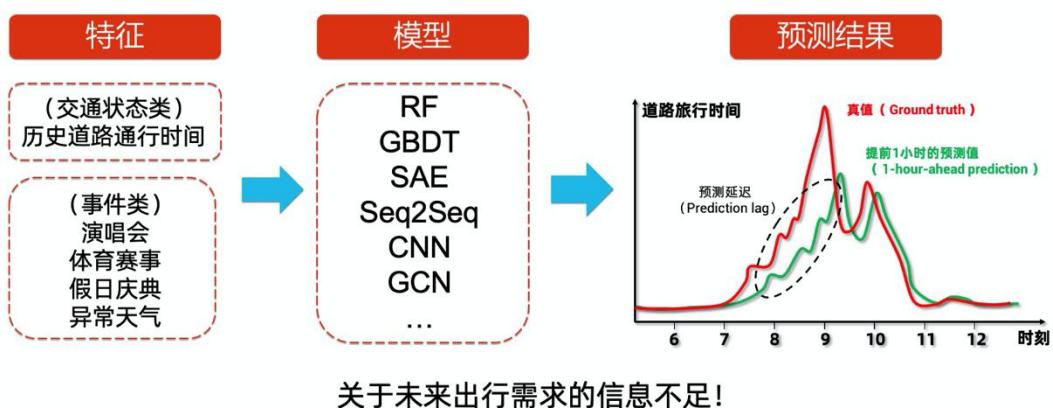
前一小时给出的预测值。可以看到，在拥堵发生的阶段，大约 8–9 点之间，模型的预测表现出了显著的延迟。

这类问题不能通过单纯升级模型的表达能力来彻底解决，不管是早期的经典机器学习模型，如随机森林、GBDT，抑或是初期的深度学习模型，如 stacked autoencoder、sequence-to-sequence model，还是近期提出的基于卷积网络、图卷积网络的更先进的结构等等，就预测效果而言都存在类似的缺陷。



路况预测是什么

业界的数据驱动方案及其缺陷



究其原因，从物理上讲，拥堵来源于车流量的增大，现有方案的事件类特征时空颗粒度较粗，不能在分钟级别、路段级别上充分表达未来车流量的信息，也即

缺乏一个预示拥堵发生的提前量信号，因此无法从根本上克服这个问题。

随后，冀晨光重点介绍了高德提出的解法。这项研究成果发表在今年的数据挖掘顶会 KDD 上。

混合时空图卷积网络：能“推导”未来路况的算法

第一步，从高德海量的实时驾车导航数据中预估出每条道路未来的车流量。如下图中左图的所示，在导航开始时刻，以及行中每隔固定的时间段，用户的高德客户端都会与云端的导航引擎进行交互，请求引擎更新剩余旅程的 ETA。对应的，在导航引擎侧，根据这同一个 ETA，就能预估出单个用户未来贡献的流量值。

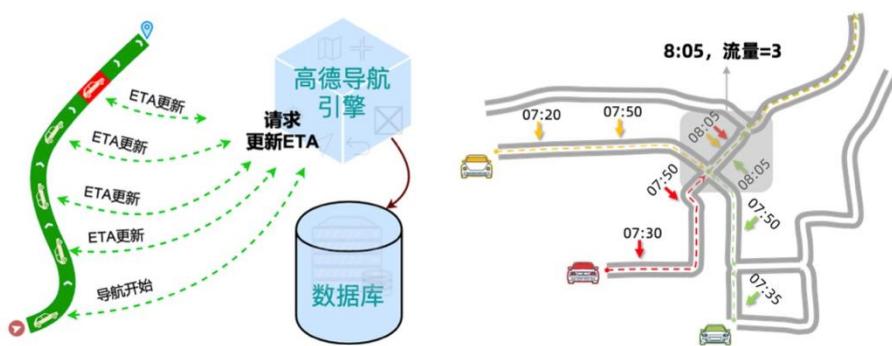
接下来，高德对全体用户的未来流量值按道路进行聚合，就获取了未来任一时刻全路网的车流量分布。以下图中右图为例，黄、红、绿三辆车在出发时均使用了高德导航，对应颜色的虚线代表高德给出的规划路线，箭头代表预计到达该道路的时间点。可以看到，在 8 点 05 分，三辆车将一同到达灰色方框里的道路。通过这样的方式，高德可以提前几十分钟预判到这条

道路在 8 点 05 分的车流量为 3。实际中，高德导航行业领先的路面渗透率，让这种方式预估的未来车流量能够较好的近似真实的未来车流量。



混合时空图卷积网络：能“推导”未来路况的算法

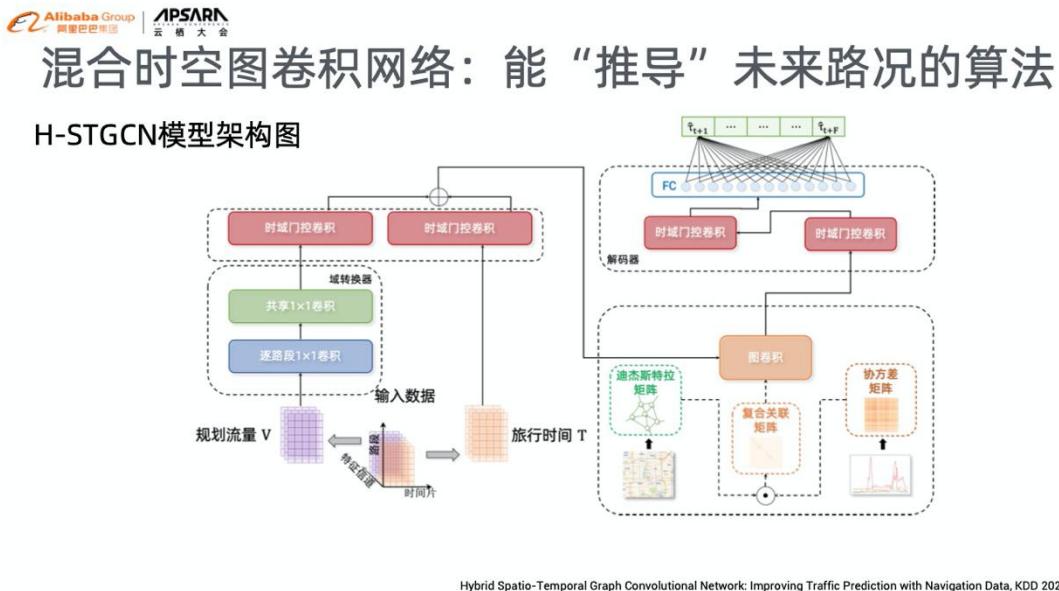
从导航规划数据中获取未来流量的信息



第二步，高德提出了一个独特的混合时空图卷积网络，简称 HSTGCN，能够有效的将这个未来车流量的特征与传统的道路旅行时间特征整合处理。

具体来说，高德提出了一个域转换器结构，将未来车流量特征转换成对应的道路通行时间特征。接下来，一维时域门控卷积从这两部分时间特征上提取出更高维的模式。之后，基于提出的复合关联矩阵的图卷积网络会捕捉住路网上各处拥堵之间的空间关联关系。最后，两个额外的时域门控卷积和全连接网络进一步

处理前述信号后进行解码，输出全路网道路未来的通行时间。



直观的讲，HSTGCN 可以学习到车流量增大、道路变拥堵的交通动力学规律，能够基于规划的流量，推导出未来路况。这个方案，作为知识驱动和数据驱动方法的有机结合体，兼备了交通仿真和统计学习这两种主流方法双方的优势。

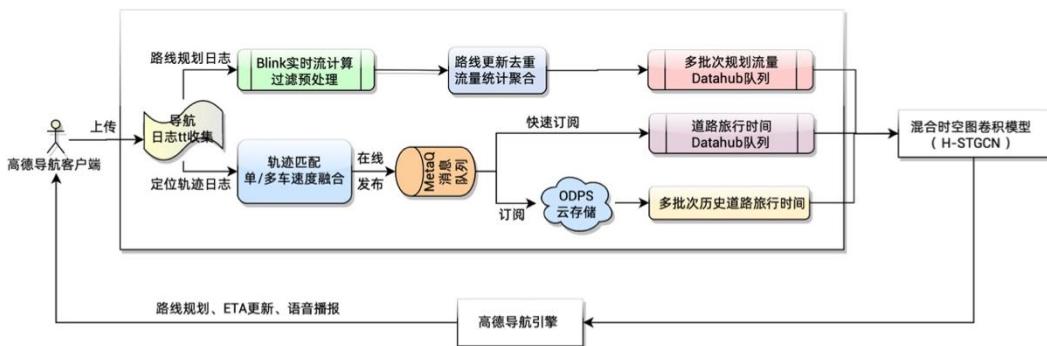
下图是模型布署的工程架构图。从左至右，导航客户端实时上传相关数据。其中路线规划数据经过 Blink 实时流处理，生成前面提到的全路网未来的流量分布。定位轨迹数据，经过路网匹配和处理，生成各个道路的旅行时间。HSTGCN 接入这两个特征源，实时预测、

发布未来的交通路况，进而帮助导航引擎更精准的计算 ETA 和规划路线。



混合时空图卷积网络：能“推导”未来路况的算法

工程架构图

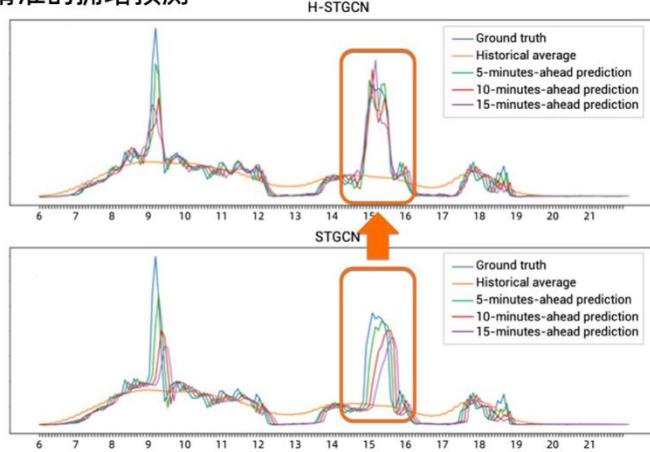


冀晨光介绍了一个模型预测突发拥堵的实例。下图里有上下两张图，分别是 HSTGCN 和此前学术界的代表性路况预测模型 STGCN 的预测结果。图的横轴是一天的全部时刻，纵轴是一条道路的旅行时间。蓝色曲线代表真实的路况，橙色曲线代表路况的历史均值，其他颜色的曲线对应模型提前 5–15 分钟的预测结果。

可以看到，HSTGCN 克服了上一代方法的不足，很好的解决了突发拥堵的预测问题。

混合时空图卷积网络：能“推导”未来路况的算法

模型收益：更精准的拥堵预测



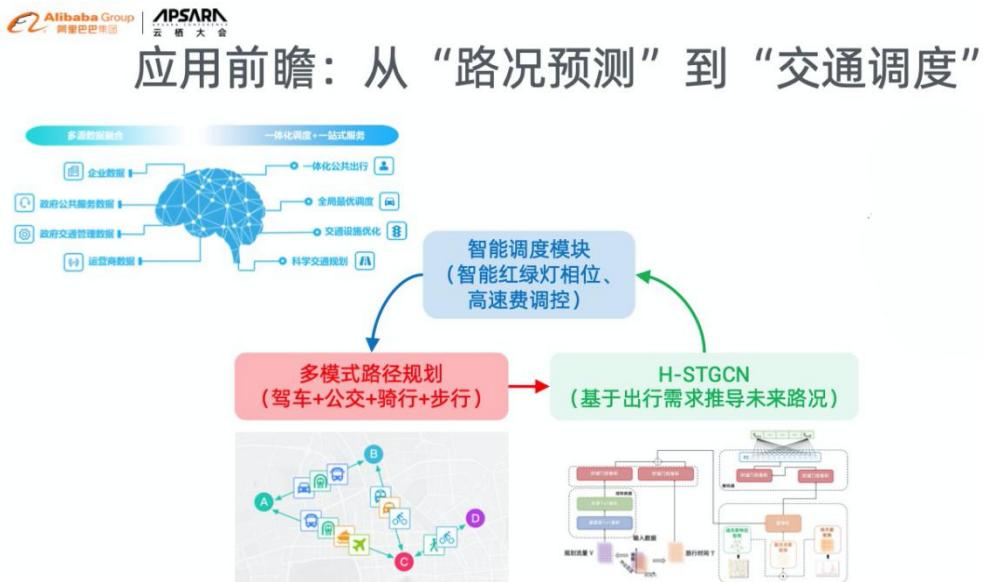
此外，因为 HSTGCN 的预测是基于规划车流量的，而这个预测值又会影响到下一个时刻的路线规划，所以 HSTGCN 带来了动态分流、消解拥堵的能力。

应用前瞻：从“路况预测”到“交通调度”

今天，各大出行科技公司所做的路况预测主要是被动式的统计预测，而 HSTGCN 则为主动式的交通调动第一次提供了工业界可落地的、数据驱动的解决方案。

可以畅想一下未来的情景。路线规划引擎会从今天的单模态规划演变成多模态规划，综合考虑驾车、公交、骑行、步行等多种出行方式。HSTGCN 利用规划引擎产出的路线及预估时间，推导出未来路网上的交通状况。接下来，智能调度模块，例如红绿灯相位调控引

擎或是高速费调控引擎，基于预估的路况进行决策，决策的结果又反馈回路线规划引擎，干预路线推荐。



整个系统实时、动态的运转，形成交通调度的智能闭环，有希望大大缓解交通拥堵，节约交通资源和成本，提高全局的运输效率。

招聘

阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德 AR & 车道级导航技术演进与实践

作者：高德技术整理

2020 云栖大会于 9 月 17 日 -18 日在线上举行，阿里巴巴高德地图携手合作伙伴精心组织了“智慧出行”专场，为大家分享高德地图在打造基于 DT+AI 和全面上云架构下的新一代出行生活服务平台过程中的思考和实践，并重点分享了「高精地图、高精算法、智能时空预测模型、自动驾驶、AR 导航、车道级技术」等话题。

「高德技术」把讲师分享的主要内容整理成文并陆续发布出来，本文为其中一篇。

阿里巴巴高级地图技术专家王前卫分享的话题是《AR&车道级导航技术演进与实践》。他为大家介绍了这些领域的核心技术、阶段成果及未来方向。

查看：[【演讲视频播放地址】](#)

王前卫主要分享了三部分的内容：

- 技术背景
- 当前进展
- 核心技术

以往，高德通过全球卫星定位系统和数字化的电子地图为用户提供了一款道路级的导航服务，帮助用户方便快捷的到达目的地。现在通过引入更能理解环境，感知环境的视觉感知系统，以及通过引入更贴近现实，更精细的车道级数据，为用户精心打造了一款基于实景的**车道级导航产品**。它能为用户带来一种全新的导航体验，做到所见即所得。

这款产品包含哪些功能呢？AR 导航通过视频增强技术实现了引导信息与现实世界更完美的贴合，为用户提供简单易懂的方向性指引。这样用户再也不会因为走到复杂路口而走错路；在距离路口较近，且用户行驶在非正确的车道上时，高德 AR 导航也能进行及时准确的变道提醒；在路口等红绿灯的时候，帮助用户实时观察周边环境，及时提醒用户，红灯已变绿灯，或者前车已经启动。AR 导航功能一经上线就获得了用户的好评。

核心技术揭秘

高德 AR 导航需要具备三方面的能力：

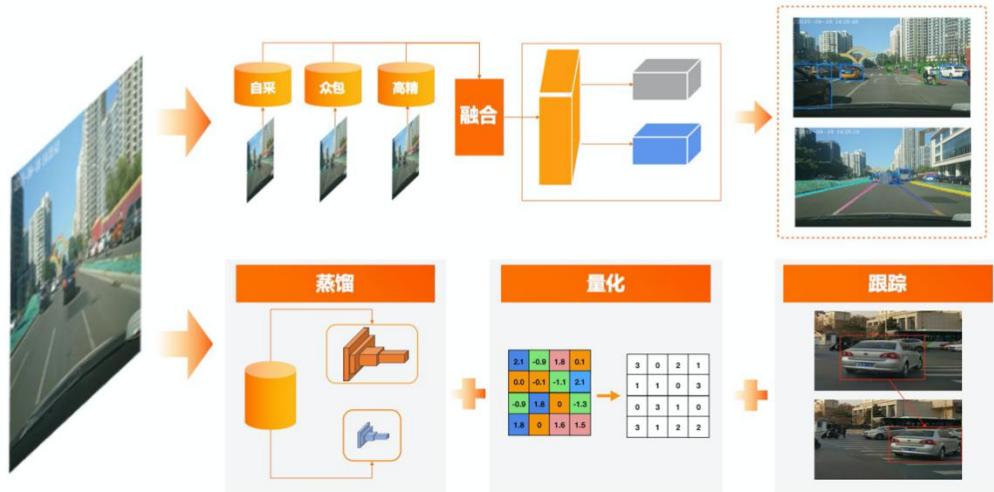
- 对周边环境实时的感知能力
- 车道级的高精定位能力
- 道路数据的精细化表达能力

环境实时感知

在环境感知上，高德 AR 导航选择了成本较低但目前使用广泛的视觉技术，通过深度学习算法来感知周围的环境。其中最大的挑战在于如何设计一款轻量化的深度学习模型，既能在低算力的设备上实时运行，同时能保证较高识别精度。高德主要在三个方面进行了优化：

第一，在数据上，高德采用了海量多源大数据的融合和提取来保证训练样本的多样性和覆盖度；第二，在算法上，主要通过优化网络模型，特征共享等方法来保证算法的准确度；第三，在性能上，通过知识蒸馏，模型的量化算法，多任务的跟踪等方法来保证在低算力上能流畅运行。

轻量化深度学习模型

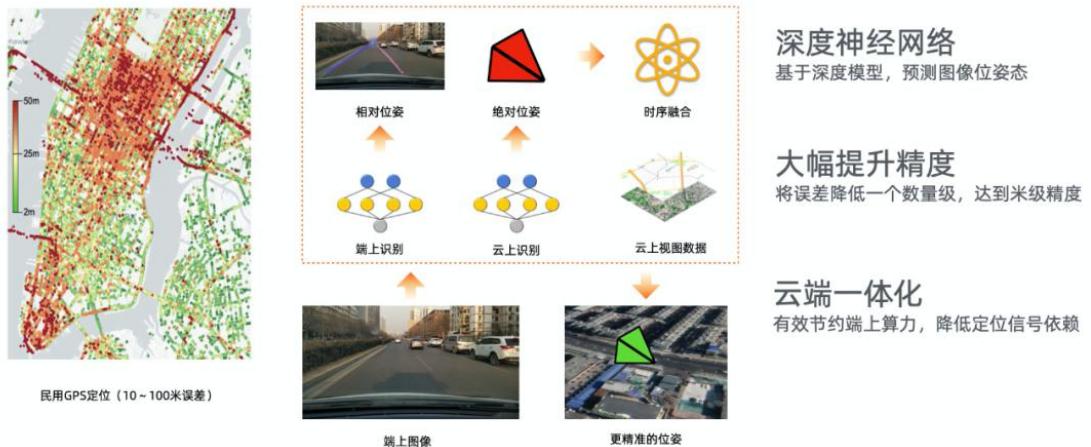


高精定位

GPS 定位精度不足，信号干扰大，特别是在遇到城市森林或者是天气不好的时候，会产生信号漂移、精度无法保证。目前精度不足已经成为大多数导航产品用户体验提升的瓶颈。

基于此，高德提出了一种基于云端一体化视觉定位技术，基于端上图像，结合云端视图大数据，通过神经网络回归出设备绝对位姿。与此同时，通过端上识别车道线、道路边沿等标识，进行相对定位。最终结合时空一致性，进行云和端的融合，大幅提升了定位精度，将定位误差提升了一个数量级。

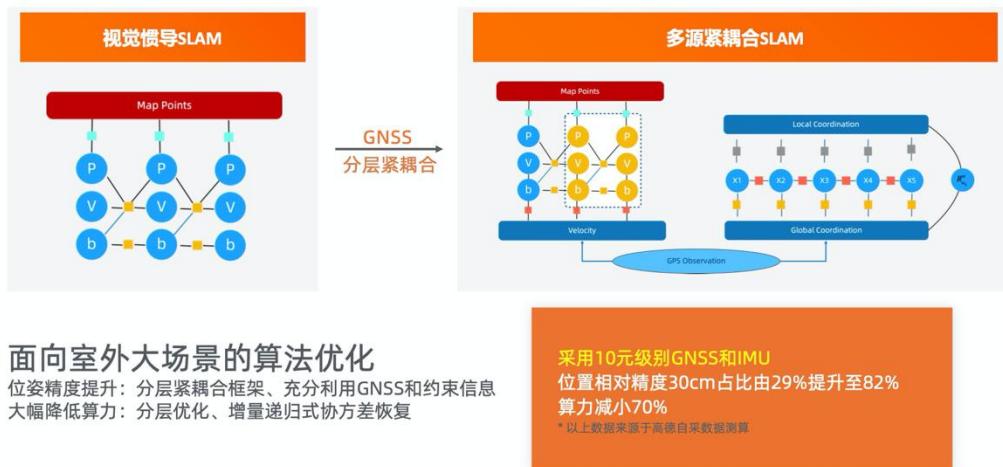
云端一体化视觉定位



在没有网络的时候，如何使用高精度定位呢？基于成本较低的 GPS、惯导和视觉传感器，高德设计研发了**多源紧耦合 SLAM (MT-SLAM)** 技术，通过算法的深度融合实现低成本高精度的位置姿态估计，为高精地图众包采集、车道级 AR 导航等业务提供很好的能力支撑。其相对位置精度 30cm 占比在 82%以上。

位置姿态的提升，主要是根据 GNSS 不同信息的精度特性，采用分层紧耦合的融合框架，对信息充分利用，同时考虑运动约束，在减少优化维度的同时也提高精度；根据实际场景的精度特点，缩减内层优化对象，来提升优化效率；根据协方差应用场景，采用增量递归的方式提升协方差恢复效率。

多源紧耦合SLAM (MT-SLAM)



在实际的用户场景中，定位遇到的环境是比较复杂的，在实现方式上，有的是基于手机 RTK 技术，有的是基于视觉传感技术。在不同场景下，有的需要标准精度定位，也有的需要高精度的定位能力。

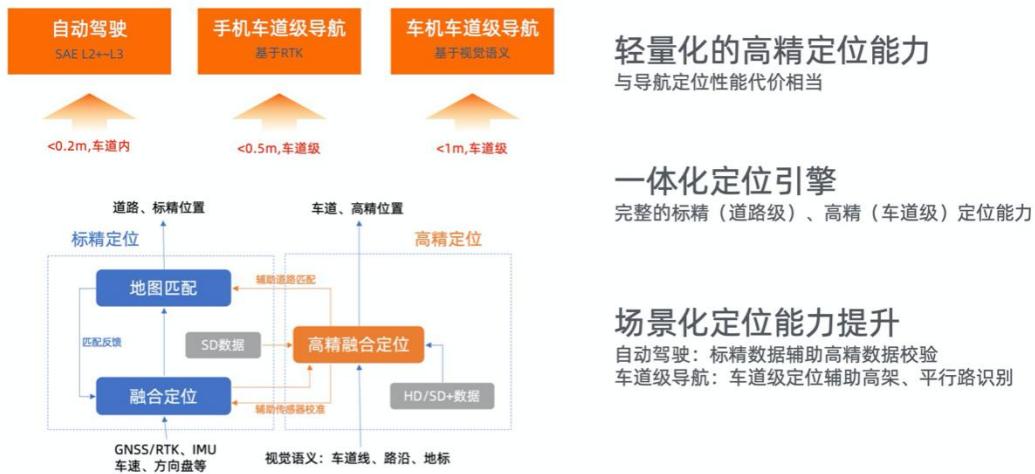
如何降低成本，提升效率，以成本最小化的方式来实现一体化定位技术应用和落地呢。

高德设计了一套高精/高标一体化融合定位系统。基于成熟的差分卫星定位或环境语义信息，构建轻量化的高精定位能力，并且和标精的导航定位结合形成一体化的融合定位引擎，满足自动驾驶、车道级导航等不同业务的需要。

一体化定位引擎，已具备完整的道路级标精、车道级高精定位能力，高精、标精定位结果独立输出又相互关联，为导航和自动驾驶联动提供便利，确保在全场景下的定位结果输出，保证定位连续性。



高精/标精一体化融合定位

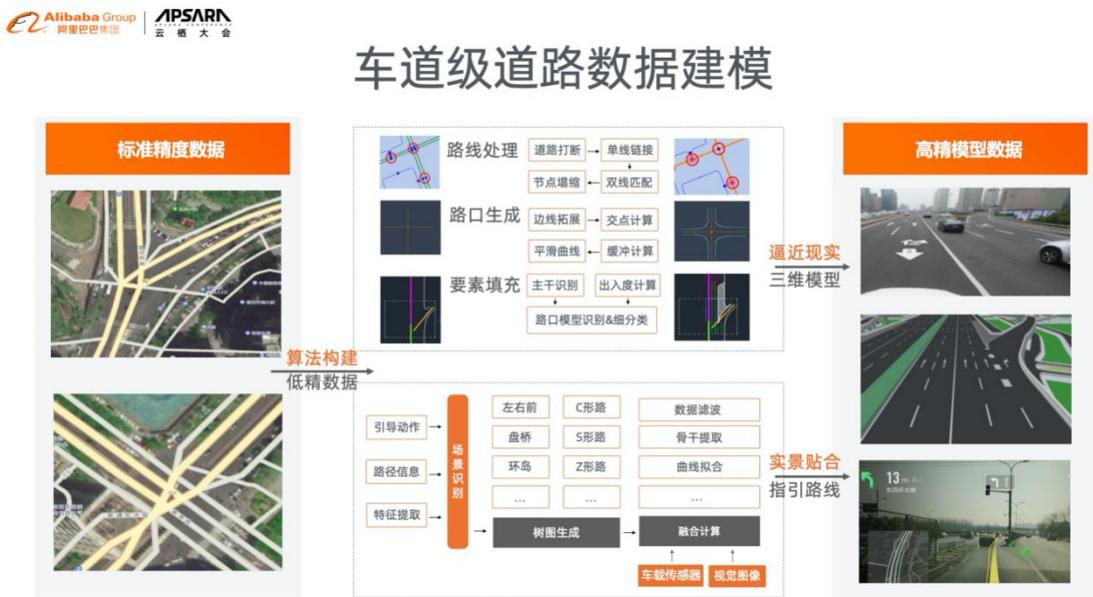


道路数据的精细化表达

现在有了车道级高精的定位，也有了对周边环境的实时感知，最后还需要考虑的是如何把标准精度数据表达得更加精细，如何通过建立道路模型，使引导信息的表达更加贴近现实场景。

大家首先能想到的是通过高精数据。高精数据的厘米级精度，确实能更真实的反映真实世界。然而，为了

追求低成本，高覆盖，高德选择了利用标准数据精度，加上道路属性信息，通过算法来构建高精道路数据模型。



高德主要通过两个方面来进行模型构建，一是道路的**模型**，主要是利用 SD 的形点数据，结合道路的车道属性信息，通过对路口的切分、建模、还原等算法来建立道路的三维模型。二是**实景中的引导信息展示**，主要利用规划路径信息和引导信息，结合实时的道路图像特征提取信息，再加融合的高精定位，在不同的场景下来分别构建对应的引导线模型。

目前高德的这套模型构建算法，已在实际项目中落地。其车道级三维模型已经能够很好的反映真实世界，更

加逼近现实世界，其 AR 导航的指示引导的铺路线和引导线，在绝大多数场景已经做了和实景道路的贴合。

招聘

阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位 地 点：北 京。欢 迎 投 递 简 历 到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

深度学习在高德 ETA 应用的探索与实践

作者：大晨

1. 导读

驾车导航是数字地图的核心用户场景，用户在进行导航规划时，高德地图会提供给用户 3 条路线选择，由用户根据自身情况来决定按照哪条路线行驶。



同时各路线的 ETA (estimated time of arrival, 预估到达时间) 会直接显示给用户，这是用户关心的核心点之一。用户给定起点和终点后，我们的任务是预测起终点的 ETA，ETA 的准确率越高，给用户带来的出行体验越好。

2. 基于深度学习模型的探索和实践

2.1 模型选择

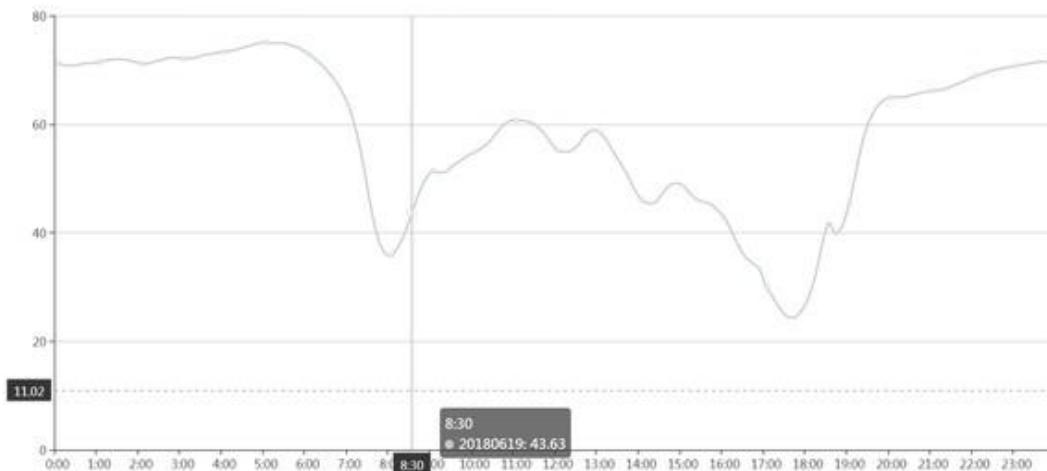
传统机器学习模型在 ETA 中，比较常用的有线性回归、RF（随机森林）、GBDT（梯度提升决策树）等回归预测类模型。线性模型表达能力较差，需要大量特征工程预先分析出有效的特征；RF 通过样本随机和特征随机的方式引入更多的随机性，解决了决策树泛化能力弱的问题；GBDT 是通过采用加法模型（即基函数的线性组合），以及不断减小训练过程产生的残差来达到回归的算法。

传统机器学习模型相对简单易懂，也能达到不错的效果，但存在两个问题：

- 模型的表达能力跟选取的特征有关，需要人工事先分析出有效的特征。

- 没有考虑上游对下游路段的影响，产生了如丢失上下游关联信息、下游受上游影响导致的不确定性等问题。

第一个问题很好理解，深度学习模型能很好地弥补这方面。针对第二个问题，以历史速度信息选取存在的不确定性为例来说明一下，历史速度信息是一个区分周一到周日七个个工作日、10 分钟间隔的历史平均时间，可以根据该路段的预计进入时间所在 10 分钟区间来选定。如下图（历史平均速度）从 0:00–24:00 的变化曲线，可以看到一天中特别是早晚高峰，速度值存在较大波动。

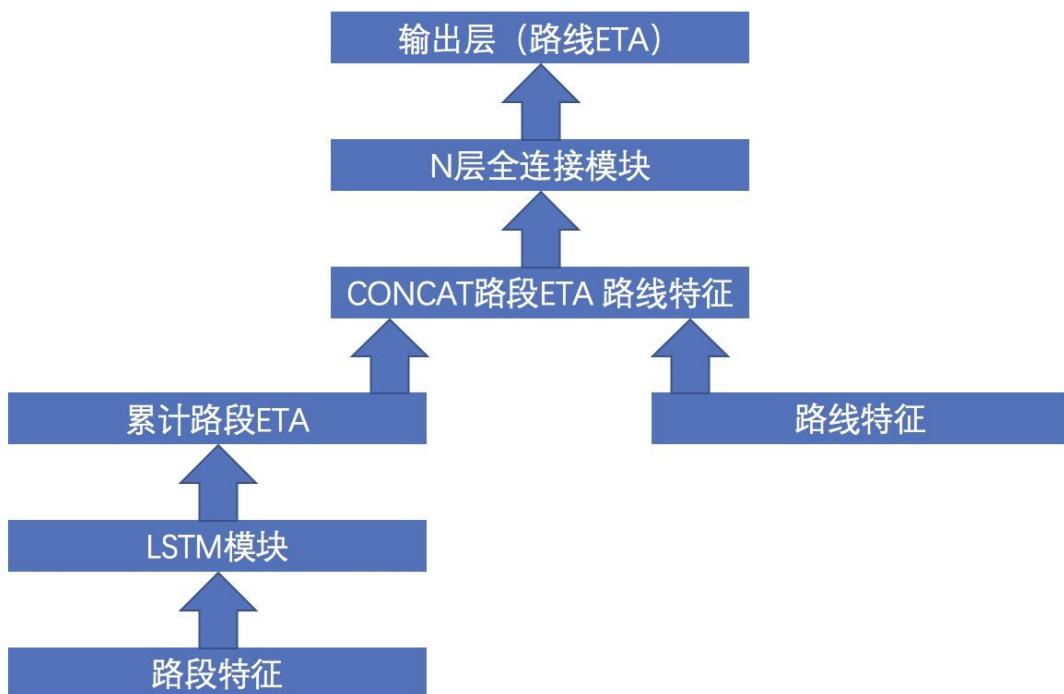


而在选取历史平均时间时，依赖的是预计进入时间，这个时间依赖于上游路段的预计通行时间，因此其选取存在不确定性，进而导致 ETA 计算不准确。

考虑到以上问题的存在，我们选择利用 RNN 的时间序列思想将路线中上下游路段串联起来进行路段 ETA 的预测。

另外考虑到 RNN 存在的长依赖问题，且结合实际业务情况，我们选择使用 LSTM 模型来进行建模，LSTM 的门结构具有的选择性还能让模型自行学习选择保留哪些上游的特征信息进行预测。

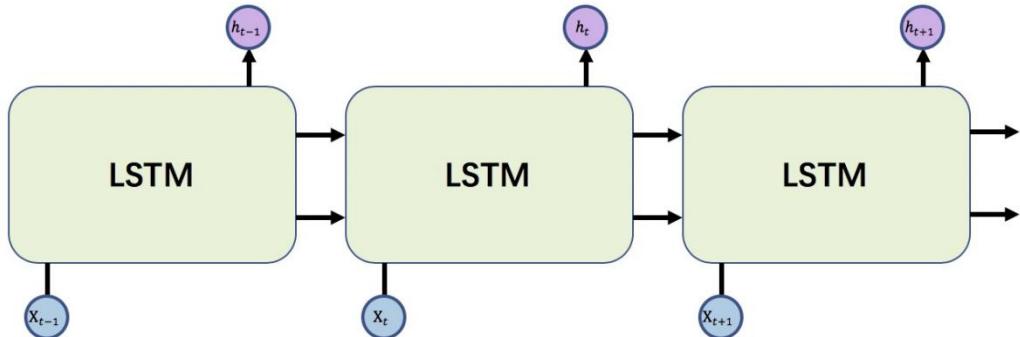
2.2 网络架构



上图为整个模型的框架图，主要分为两部分，使用 LSTM 模块对路线中的路段 ETA 的预测和最终使用 N 层全连接模

块对累计路段 ETA 及路线各特征进行完整路线的 ETA 预测。

2.3 路段 ETA 预测



上图为各路段 ETA 预测使用的 LSTM 结构图， X_t 为路线中第 t 个路段的特征信息，主要包含对应的实时路况信息、历史路况信息、路段的静态特征等。

LSTM 本是输入时间序列数据的模型，我们利用该思想，将路线中各路段序列依次输入模型。

2.4 完整路线 ETA 预测

在 LSTM 模块得到累计路线 ETA 预测值后，结合该路线的静态属性，使用全连接模块将其整合成最终输出的完整路线 ETA 预测值。

路线的属性特征主要指一些人工提取的特征，如该路线的长度、导航规划发起特征日、是否早晚高峰时段等，用以加强模型在不同场景下的表达能力。

损失函数选用线性回归常用的平方形式：MSE，公式如下：

$$MSE = \frac{1}{N} \sum_{j=1}^N ((ETA_{\text{路线}j} - \text{用户实走}_j) / \text{用户实走}_j)^2$$

其中，N 是路线数量，ETA 路线 j 为路线 ETA，即预测值；用户实走 j 为用户在该路线的实走时间，即真值。

3. 模型效果

衡量模型效果，即路线上 ETA 的预测值时，主要考虑的是准确率。一般情况下，用户对 ETA 偏长和偏短的容忍度不同，对偏长容忍度更高。比如用户要去机场，ETA 给的时间偏短 10 分钟比偏长 10 分钟对用户的损害更大。因此准确度的指标设计倾向于 ETA 偏长，定义为满足用户一定容忍范围的请求比例，即准确率作为主要衡量指标。

在北京市上的实验结果显示，ETA 准确率得到提升，MSE loss 下降比例 28.2%，效果有了明显的提升。

4. 小结

本文介绍了引入深度学习模型，帮助建模导航规划的预估到达时间预测，成功解决了线性模型的不足，也为后续引入更多特征、进行更多探索打开了空间，如历史速度信息的不确定度、时效性、周期性、突发事件、路网结构等。

招聘

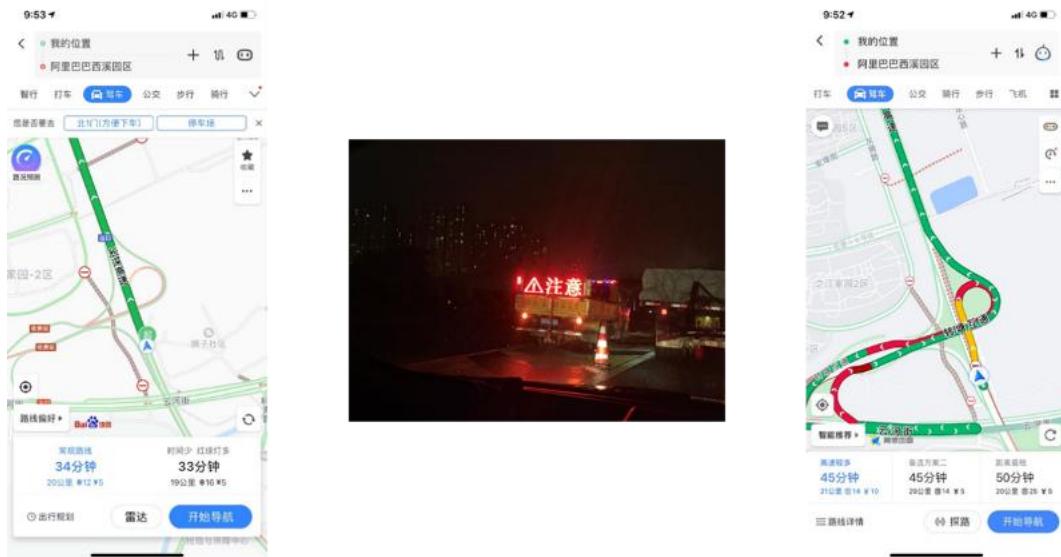
阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位 地 点：北 京。欢 迎 投 递 简 历 到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

机器学习在高德地图轨迹分类的探索和应用

作者：流明

1. 背景

当我们打开导航，开车驶向目的地的过程中，有时候会碰到这样的问题：前方明明没有路，可能在施工封闭，可是导航仍然让我们往前开车，以至于我们无法顺利到达目的地。全国道路千千万，每天都有巨量的道路变得不可通行，那么如何动态的识别出哪些道路走不通了呢？



图中所示即为因封路事件导致的导航路线改变

道路不通常往往导致该条道路汽车流量突然降低。监控汽车流量的变化是挖掘封路事件的重要指标。但是，目前业务中遇到的一个重要问题是，针对汽车无法通行的封路事件，行人、自行车可能都可以穿行，这些行人、自行车等的噪声流量大大削弱了道路流量变化。

因此，如果能够对行人、自行车、汽车的轨迹进行分类，就可以对道路流量的噪声进行过滤，仅仅关注汽车流量，流量随着封路事件的变化将更为显著，从而便于道路封闭的挖掘。本文主要针对非机动车、机动车分类探索轨迹分类问题。

2. 样本获取与标签制定

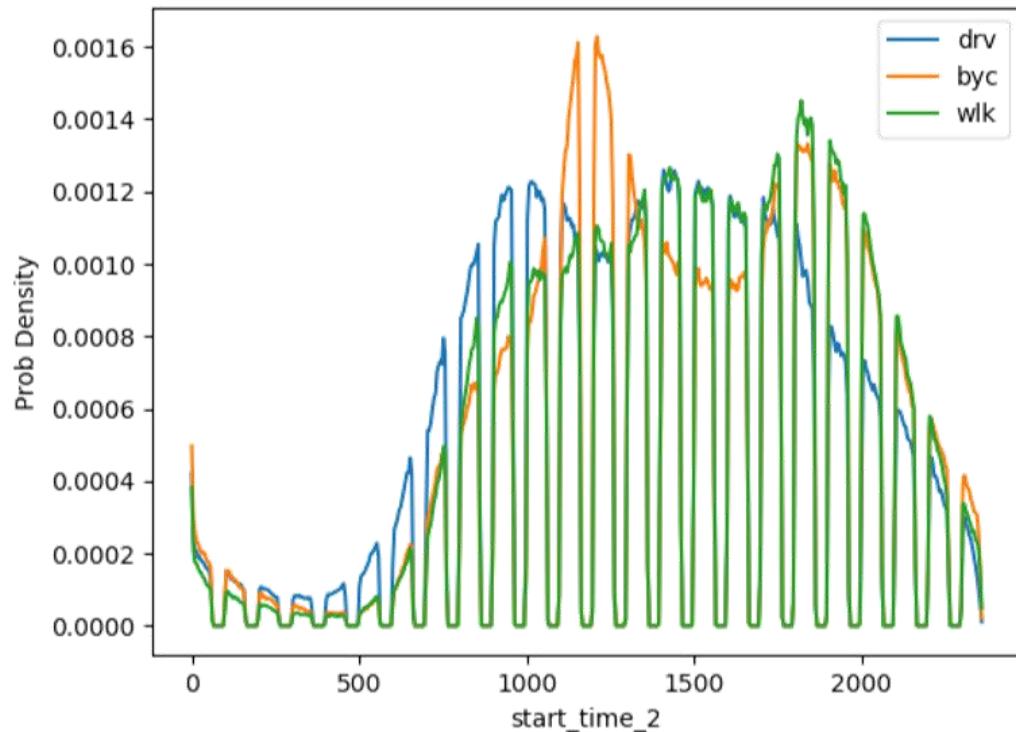
由于轨迹数据缺少原始真值，我们将用户导航模式作为轨迹分类的伪标签。例如当时用户采用汽车导航，其轨迹对应的标签即为汽车。由于汽车导航数据远远多于非机动车，不同伪标签样本比例差异巨大，存在严重的样本不平衡问题。此外，用户导航模式与用户实际出行方式可能并不一致。比如有些用户可以根据汽车导航步行到达目的地。下文介绍的标签-概率混合贝叶斯模型将分析并试图解决上述 2 个问题。

3. 特征分析

可以将轨迹分类相关特征划分为 5 类。分别是：

- 轨迹概况特征集，包括轨迹耗时、轨迹长度、轨迹开始时间等。
- 速度相关特征集，包括最大速度、平均速度、速度标准差等。
- 时间相关特征集，包括道路末端等待红灯时间，调头时间，左转时间等。
- 行为相关特征集，包括调头行为，往复活动，左转减右转时间等。
- 用户画像特征集，包括用户职业、有车概率。

下面以轨迹开始时间特征为例，解释该特征的物理意义。其概率密度函数如下所示（drv，汽车；byc，自行车；wlk，步行）：



早晨 (5:00~10:00) 汽车轨迹概率较高，可能是早高峰导致。

午时 (11:00~13:00) 自行车轨迹概率较高，可能是由于外卖送餐。

傍晚 (17:00~20:00) 步行、自行车概率均较高，可能是由于下班散步以及外卖送餐。

4. 贝叶斯模型的概率分布视角

选择基于贝叶斯分类器进行改进的原因如下：

- 贝叶斯分类器属于生成模型，依赖于条件概率密度函数，具有明确的统计学意义。此外，如前面提到的条件概率图示，通过观察不同轨迹、标签的概率密度函数，能够逐个分析、说明特征的有效性。

- 贝叶斯分类器可以表示为：

$$y = \arg \max_{c_k} P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)} | Y = c_k)$$

可以通过调整 $P(Y = c_k)$ 快速适应不同的样本不平衡场景。

- 通过增添、删减、改动特征的概率密度函数，可以快速完成贝叶斯分类器的迭代改进。并且相较于决策树，贝叶斯分类器不会对某一个特征的变动过于敏感。
- 贝叶斯分类器最终输出为概率值，可以作为置信度。

4.1 标签-概率混合贝叶斯模型

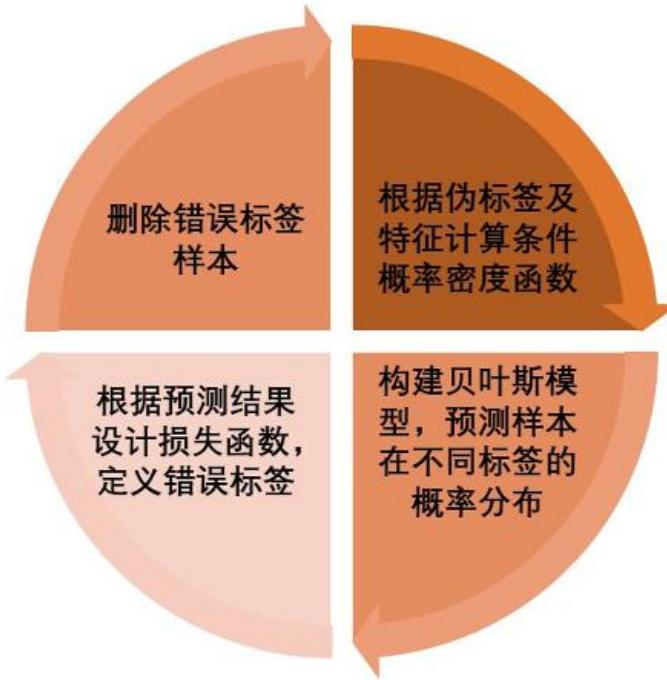
当前轨迹分类问题为样本不平衡（Data Unbalanced）
标签不准确（Noisy Label）问题。其中，样本不平衡
问题可以通过调整 $P(Y = c_k)$ 进行处理。而解决标签不准

确问题的主要策略之一即为设计样本清洗方法，删除错误样本[1]。

Tanaka 等人基于卷积神经网络提出了伪标签损失函数，通过交叉熵损失函数与伪标签损失函数的迭代优化完成了错误标签的修正[2]。

受此启发，贝叶斯模型同样能够建立伪标签损失函数完成样本清洗。

我们基于贝叶斯分类器的分类结果，提出基于伪标签极大似然估计的伪标签损失函数，完成错误样本清洗，再迭代完成贝叶斯分类。该模型的迭代流程如下图所示。



标签-概率混合优化贝叶斯模型迭代流程

首先，基于当前伪标签样本集搭建朴素贝叶斯模型。设样本特征向量与当前伪标签分别为 X_i , Y_i 。其中， Y_i 为 one-hot 标签， $X_i = \{x_i^{(j)}\}$ 。当前朴素贝叶斯模型对于特征向量 X_i 属于类别 c_k 的概率估计如下式： \leftarrow

$$P(y_k = 1 | X = X_i) = \frac{P(y_k = 1) \prod_j P(x_i^{(j)} | y_k = 1)}{\sum_k P(y_k = 1) \prod_j P(x_i^{(j)} | y_k = 1)}$$

假设真值标签为 $Y_i^* = \{y_{i,k} + \xi_{i,k}\}$ ，其中 $\xi_{i,k}$ 是伪标签的修正项，应保证 Y_i^* 仍然是 one-hot 标签。则以 $\xi_{i,k}$ 为变量，已构建贝叶斯模型计算出的条件概率为真值，构建基于当前贝叶斯模型输出概率的极大似然函数，如下式： \leftarrow

$$MLE = \max_{\xi_{i,k}} \prod_i \prod_k P(y_k = 1 | X = X_i)^{y_{i,k} + \xi_{i,k}}$$

对似然函数取对数，并且取其负值，可得损失函数为： \leftarrow

$$L = \min_{\xi_{i,k}} \sum_i \sum_k -(y_{i,k} + \xi_{i,k}) P(y_k = 1 | X = X_i)$$

为防止过拟合，添加一范数，取 α 为一范数的系数。 \leftarrow

$$L = \min_{\xi_{i,k}} \sum_i \sum_k \alpha |\xi_{i,k}| - (y_{i,k} + \xi_{i,k}) P(y_k = 1 | X = X_i)$$

可通过逐个分析样本求解此损失函数。考虑任一样本 X_i , 伪标签 $y_{i,k1} = 1$ 。为保证

$Y_i^* = \{y_{i,k} + \xi_{i,k}\}$ 仍然为 one-hot 标签, $\{\xi_{i,k}\}$ 有两种情况。[«](#)

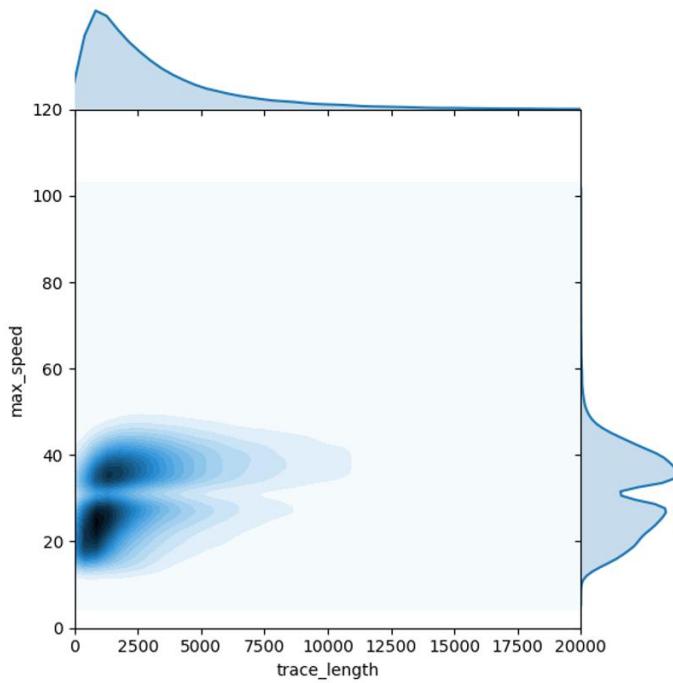
- $\{\xi_{i,k}\}$ 均为 0, 此时损失函数认为伪标签正确, 样本 X_i 损失函数值为 $-P(y_{k1} = 1 | X = X_i)$ 。[«](#)
- $\xi_{i,k1} = -1, \xi_{i,k2} = 1$, 其余值为 0, 此时损失函数认为伪标签错误, 样本 X_i 损失函数值为 $2\alpha - P(y_{k2} = 1 | X = X_i)$ 。[«](#)

因此, 如果存在 $k2$ 使得 $P(y_{k2} = 1 | X = X_i) - P(y_{k1} = 1 | X = X_i) > 2\alpha$, 则损失函数认为标签存在错误, 可删除错误样本, 完成样本清洗。[«](#)

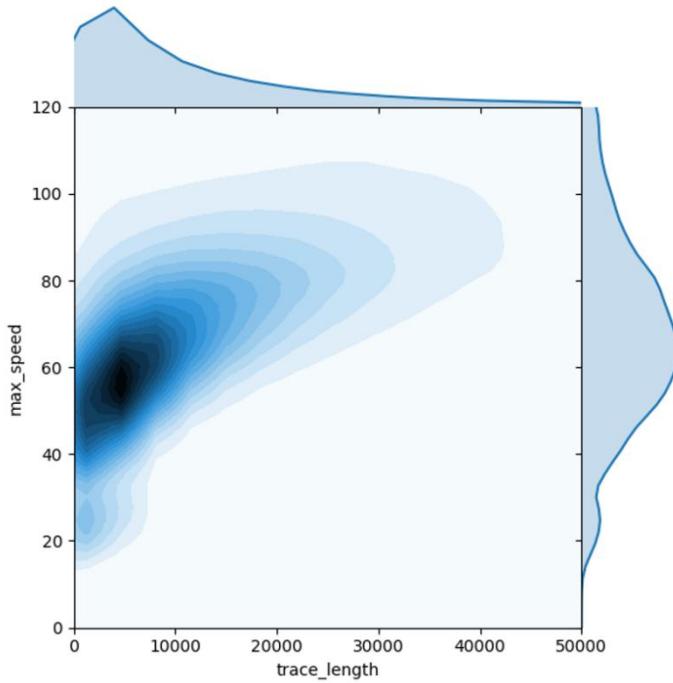
利用清洗过的样本重新训练贝叶斯模型, 获得新的预测概率 $P'(y_k = 1 | X = X_i)$, 之后可以迭代清洗样本, 直至满足要求为止。[«](#)

4.2 联合概率密度函数计算

由于贝叶斯分类器假设各变量相互独立, 因此不可避免的会对一些场景下的样本产生错误预测。例如, 外卖骑手以及快递员应当被判定为非机动车。这种类型轨迹长度可能较长(超过 10 公里), 最大速度适中(小于 50 公里每小时)。在假设行驶距离与最大速两个特征相互独立的情况下, 容易错误地把外卖骑手以及快递员的行驶轨迹判定为汽车。



自行车轨迹行驶距离以及最大速度联合概率密度函数



汽车轨迹行驶距离以及最大速度联合概率密度函数

但是，行驶距离与最大速两个维度的特征并不相互独立。上图构建了针对汽车轨迹这两个维度的联合概率密度函数，可以发现，对汽车轨迹而言，行驶距离越长，最大速可能越高，当汽车行驶距离超过 10 公里时，其最大速度小于 50 公里每小时的可能性很低。因此，通过构建行驶距离与最大速度两个维度的联合概率密度函数，替换两个独立概率的相乘，可以帮助解决长距离非机动车轨迹被误判为汽车的问题。

4.3 基于贝叶斯的轨迹分类实验结果

评测团队抽样约 100 条数据并人工标记真值。最终模型分类效果如表所示。

	朴素贝叶斯	联合概率贝叶斯	联合概率贝叶斯+样本删除
机动车准确率	低	中	中
机动车召回率	中	中	高
非机动车准确率	中	中	高
非机动车召回率	低	中	中
综合分类精度	低	中	高

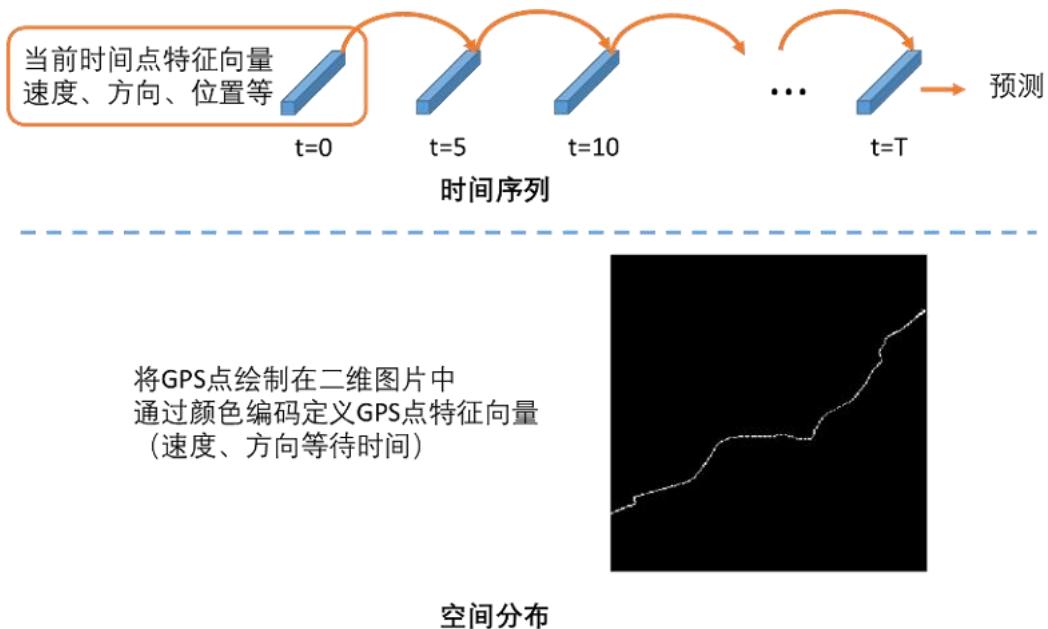
5. 深度学习模型的图像编码视角

由于轨迹数据缺少原始真值，我们将用户导航模式作为轨迹分类的伪标签。例如当时用户采用汽车导航，其轨迹对应的标签即为汽车。本次基于深度学习的探

索不考虑标签噪声的问题。

5.1 轨迹信息的两种观察方式

深度学习的优势在于能够从原始数据中学习到有效信息，无需人工挖掘特征。针对轨迹数据的特点，存在两种观察轨迹的方式，分别是时间序列与空间分布。



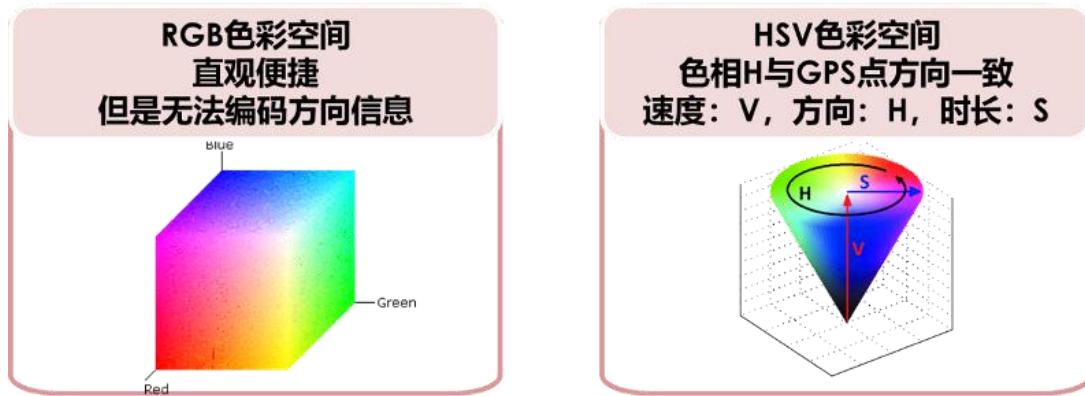
时间序列：轨迹当中的 GPS 点数据随时间推移依次上传至数据库中。轨迹数据天然具备时间序列属性。因此，可以采用 TCN 或 RNN 构建模型，学习轨迹中的时间序列信息，完成轨迹分类。

空间分布：将轨迹数据绘制在地图中，则轨迹构成图片中的一条线。如果能够将速度、方向、等待时间编码到线的颜色当中，则能够采用 CNN 从轨迹图像中学习到有效信息[3]。

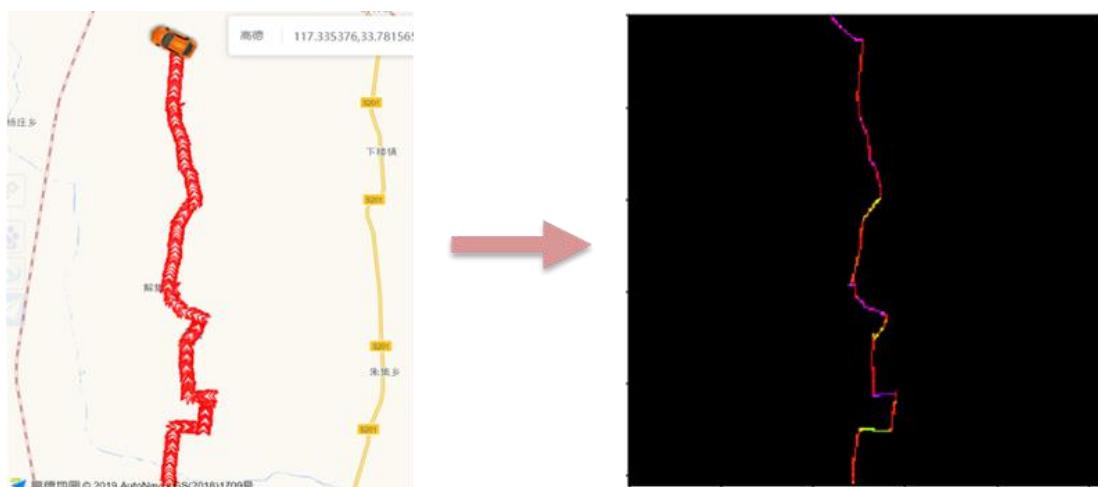


图左侧大概率为快递员轨迹，图右侧大概率为汽车轨迹。GPS 点的空间分布能够为轨迹分类提供有效信息。因此，我们采用空间分布模式构建模型，探索基于深度学习的轨迹分类。

轨迹颜色编码：从 GPS 点中获取的主要信息为速度、方向、等待时长。将这 3 个维度的信息进行轨迹颜色编码有 2 种方式，分别是 RGB 编码与 HSV 编码。其中 HSV 即色相 (Hue)、饱和度 (Saturation)、亮度 (Value)。



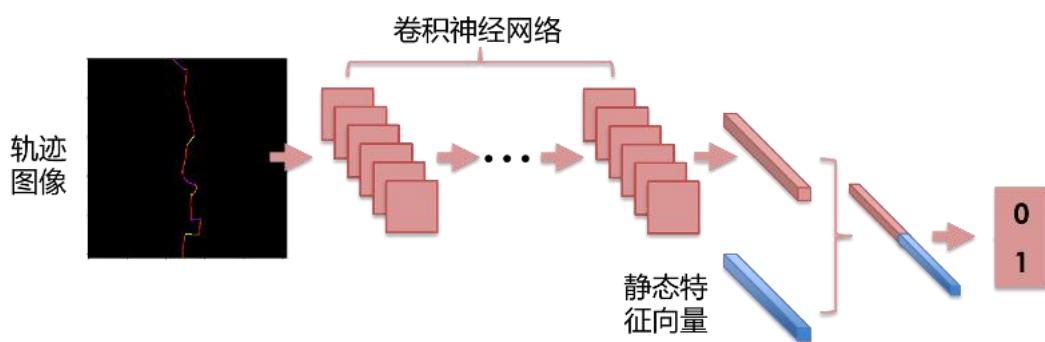
由于方向信息为 0~360 度的角度值，与 HSV 色彩空间中的色相 H 完全一致。因此，本文采用 HSV 色彩空间对速度、方向、等待时长进行编码，编码方式为速度：V，方向：H，时长：S。编码后将轨迹缩放为 256×256 的图片。对地图轨迹的编码结果如下图所示。



5.2 双流神经网络模型

基于编码生成的轨迹图片依然缺失一些重要信息，包括将轨迹缩放至 256×256 图片的缩放因子，以及 GPS

点所在的位置。可以将轨迹匹配结果中通过国道、省道、城市快速路等不同类型道路的比例构造出特征集表征 GPS 点所在的位置信息，加入缩放因子构造一个一维静态特征向量。



将卷积神经网络学习到的特征向量以及该一维特征向量合并，最终通过全连接层完成轨迹分类。最终卷积神经网络选择 ResNet50 结构。

5.3 基于深度学习的轨迹分类实验结果

评测团队抽样约 100 个样本，人工标记真值。

步行街场景	提出的优化贝叶斯模型	深度学习模型
机动车准确率	中	中
机动车召回率	高	高
非机动车准确率	高	高
非机动车召回率	中	中
综合分类精度	高	较高

如上表所示，以人工标记标签为真值验证深度学习模型，深度学习模型能够取得有效的轨迹分类精度，但是最终分类效果弱于提出的贝叶斯模型。可能的原因有如下几点：

- ResNet50 并不是学习轨迹图像的最优模型。
- 仅采用 8 月 17 日的样本训练模型，样本多样性不足。
- 所选择的特殊场景样本分布与深度学习中的训练集样本分布差异巨大。

6. 小结

轨迹分类对于准确及时地挖掘道路封闭事件具有重要意义。本文从给予概率密度分布的贝叶斯模型视角与基于轨迹点图像编码的深度学习视角分别探索了轨迹分类可能的技术方案。

未来轨迹分类模型还可以从聚焦应用场景，优化应用以及拓展上游数据，优化特征两个方面进行改进。

7. 参考文献

[1]. Fr é nay B, Verleysen M. Classification in the presence of label noise: a survey[J]. IEEE transactions on neural networks and learning systems, 2013, 25(5): 845–869.

[2]. Tanaka D, Ikami D, Yamasaki T, et al. Joint optimization framework for learning with noisy labels[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018: 5552–5560.

招聘

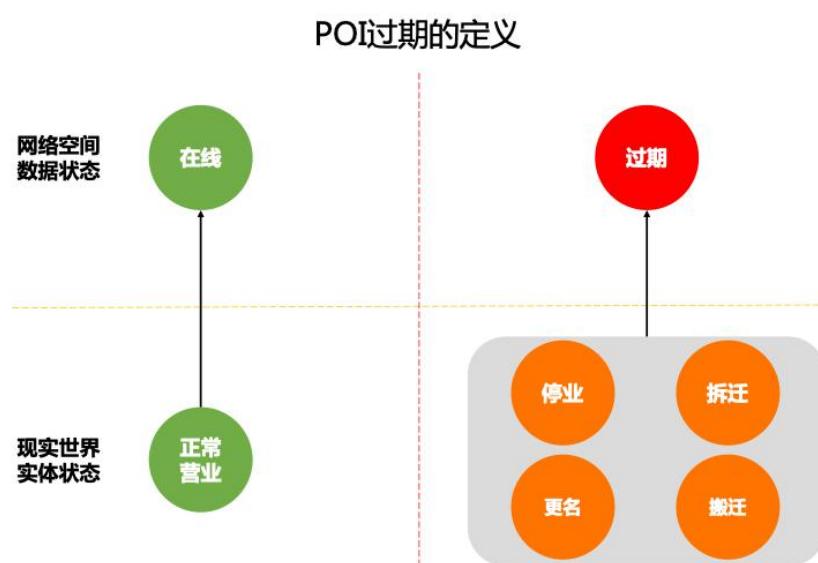
阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德地理位置兴趣点现势性增强演进之路

作者：致斌

1. 导读

人们在高德地图上会看到很多地理位置兴趣点（Point of Interest，缩写为 POI），例如餐厅、超市、景点、酒店、车站、停车场等。对 POI 数据的评价维度包括现势性、准确性、完备性和丰富性。其中，现势性就是地图所提供的地理空间信息反映当前最新情况的程度，简而言之，增强现势性就是指尽可能快速地发现已停业、搬迁、更名、拆迁的过期冗余 POI 数据，并将其处理成下线状态的过程。

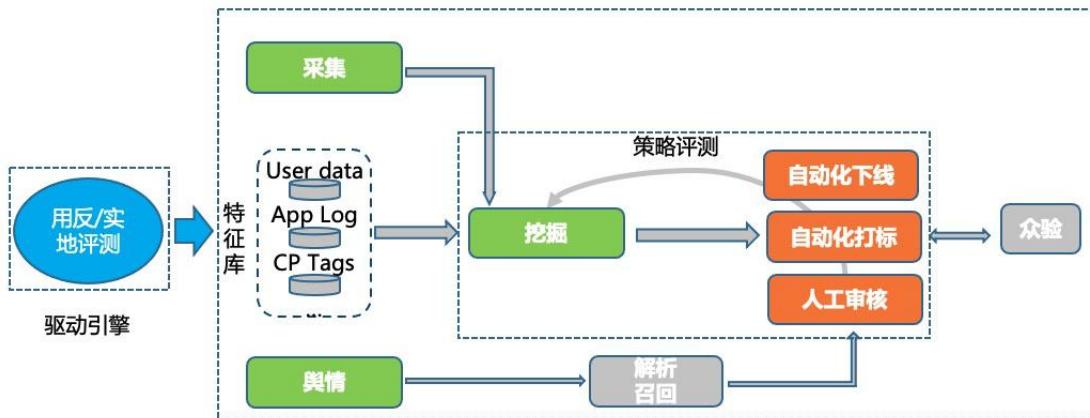


在线的过期冗余数据会伤害用户体验，经过推算，头部在线数据的一个过期率百分点年度影响用户体验 3 亿次。因此，POI 过期问题的解决势在必行，以增强现势性，减少用户伤害。

POI 过期问题的解决分为发现和处理两个环节。发现采用挖掘线主导，采集线和舆情线补位的方式。采集的天然优势是自带核实资料，劣势也很明显，成本高、下发频率低，因而发现的时效性不高，且采集线发现过期需要经过挖掘线；互联网舆情时效性高，但覆盖少且 ROI 低；作为覆盖高、时效高、成本低的大数据挖掘是绝对主力。

处理环节有人工核实、自动化打标和自动化下线三种手段。人工核实存在缺少核实资料从而导致核实率低的问题，这是因为挖掘所依赖的观测资料不能提供过期的实锤证据，观测资料不等于核实资料，且过期的数据更倾向于核实不到，即能发现、难处理，从而伴随着挖掘的演进，衍生了自动化打标和自动化下线两种处理手段以及相应的风险控制机制（打标回捞、下线回捞），一方面提高了处理能力，另一方面降低了人工成本。

挖掘主力、采集补位、舆情兜底



注：自动化打标是一种与前端搜索联动灰度处理高疑似过期数据的方式—打标数据非精搜不展现，精搜伴有话术提示。其背后的核心思路是由传统图商的实锤思维向互联网思维转变，及时触达用户，快上快下。



本篇文章将主要介绍挖掘的演进历程，作为过期发现的核心手段，挖掘在不同阶段分别面临缺资料、提准时、资料薄三大问题，站在今天回首过去，这个过程可以分为三个阶段：

- 基于自身属性的 POI 过期挖掘
- 基于使用行为的 POI 过期挖掘
- 基于人地关系的 POI 过期挖掘

我们利用策略、机器学习和深度学习等数据挖掘技术，从点到面、由粗到精地攻克 POI 过期挖掘业务，POI 现势性增强的模式已经发生了深刻的变化。

2. 数据挖掘手段的演进

2.1 阶段一：基于自身属性的 POI 过期挖掘

早期的主要矛盾是缺少挖掘资料，如果同时做资料的 POI 挂接和基于挂接资料的挖掘策略会导致挖掘链路长、项目风险高。因此，在提升新资料的 POI 聚合能力的同时，以 POI 自身属性作为主要的挖掘资料。高德 POI 团队在信息聚合、融合方面有长时而丰富的积

累，POI 属性可大致分为三类：基础信息、深度/动态信息和关系信息。

基础信息：表征实体，包括名称、坐标、地址、行业、电话、时间、来源等。

深度/动态信息：增加 POI 数据丰富性，包括：图片、评分、评论、团购、报价等。

关系信息：POI 间通过语义、时空建立的关联，包括：亲子、引用、共现等。

针对不同的属性，我们设计不同的策略去挖掘过期 POI。根据复杂程度，我们将策略主要分为：基于单 POI 的挖掘和基于多 POI 的挖掘。二者的最大区别在于是否使用 POI 间的关系信息。下面介绍几个比较典型的策略：

策略		基础信息									深度/动态信息			关系信息				
大类	小类	名称	坐标	地址	电话	类别	来源	时间	状态	...	评论	评分	图片	...	亲子	引用	区块	...
基于单POI挖掘	评论过期语义挖掘										◎							
	来源状态异常					△	◎	△	△									
	电话异常				◎	△	△	△	△									
	来源独有				△	△	◎				△	△	△					
	沉睡数据				△	△	△	◎			△	△	△					
	...																	
基于多POI挖掘	原关系挖掘	◎	△	△		△									△			
	同地址挖掘	△	△	◎		△	△	△						△	△			
	同电话挖掘	△	△		◎	△	△	△						△	△			
	实采网格差分		△			△	◎	△	△						◎			
	亲子联动挖掘	△	△			△			△					◎				
	拆迁区域挖掘		◎			△	△	△	◎						△			
...																		

策略特征使用表

◎代表关键特征，△表示辅助特征

评论过期语义挖掘是比较典型的基于单 POI 的挖掘策略。深度/动态信息中的评论是获取用户对 POI 反馈的有效途径之一，其中也包括对过期 POI 的反馈，我们通过匹配关键词很容易找到这种评论。上、下文的语境会导致关键词的语义发生变化，为此，我们利用 TextCNN 模型实现语义分类以达到消歧的目的，筛选出真正表达 POI 过期的评论。如下所示：



“原”关系挖掘使用 POI 间的引用关系，是一种基于多 POI 的挖掘策略。我们在含有“原”关键字的 POI 名称、别名或地址中通过实体抽取技术，得到“原”关系（新旧关系）的两个 POI 名称，通过聚合技术找到旧名称所对应的过期 POI。

同地址策略则利用地址门牌冲突关系来挖掘。其逻辑是：相同的门牌号（包括室内水牌）上通常只有唯一的经营实体，若地址上存在两个或多个实体且不是聚集实体（商场、园区等），则应当存在过期 POI。我们采用图论对问题建模，取门牌相同的 POI 集合，将 POI 视为节点，POI 间关系（亲子、兄弟、共现、参考引用等）为边。利用最大连通分解算法将集合划分为 K 个连通子图，每个子图看作是一个实体或聚合实体。若 K=2，则将更新时间较早的子图作为疑似过期集合输出。

同电话策略是为数不多能与具体过期现象对应的策略。选取有相同电话的 POI 的集合，与同地址策略类似，通过名称语义计算、空间计算、共现关系、亲子关系等，剔除掉聚集实体、连锁店、疑似重复数据等噪音，并根据名称相似性和距离关系，分辨出更名和搬迁现象。电话实际上代表着 POI 背后真正的人，通过人的行为变化可以判断一个 POI 过期与否，甚至可以推断出该 POI 具体的过期现象。

伴随着 POI 聚合多种新数据源的能力的日渐成熟，新的挖掘资料已具备。我们的重心也逐步转移至基于使用 POI 行为的挖掘。

2.2 阶段二：基于使用行为的 POI 过期挖掘

步入阶段二，缺少挖掘资料已不再是解题的主要矛盾，人工核实率低、处理能力不足的问题凸显，从而迫切需要建立自动化打标/下线能力（提准）。过期挖掘的实质是感知伴随 POI 过期而发生的变化，进行事后观测式挖掘，比如，过期一般都会伴随着 POI 活跃度（运单量等）的下降。

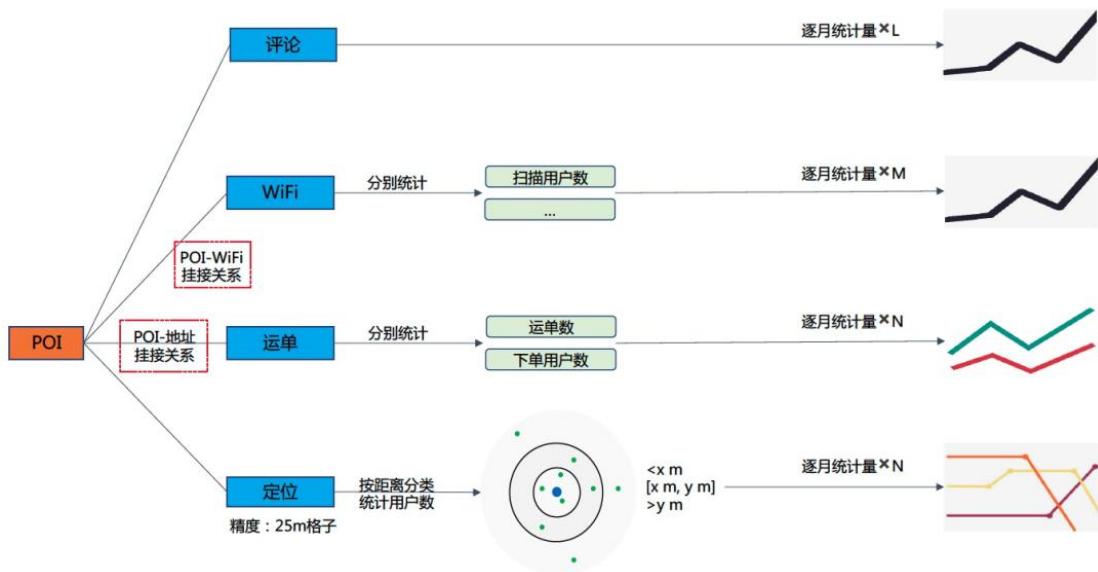
前文已提到，挖掘所依赖的观测资料不能提供过期的实锤证据（比如，运单消失并不是过期实锤），且过期强相关因子种类偏少、天然引入上游误差以及真实世界存在贝叶斯误差；外加随着解题推进，在线 POI 数据现势性增强、过期率下降，在观测资料固定的前提下，过期挖掘的产量及精确率均随过期率的下降而自然下降，上述这些都会导致精确率难提高，因此，提准难成为该阶段的主要矛盾。

特征层面我们通过去噪、精细化加以应对，受篇幅所限，本文暂不做展开介绍。而算法层面则是通过技术升级来应对。路线图：从规则到模型；从浅层模型到深度模型；从单源决策到多源信息融合；从决策层多源信息融合到特征层多源信息融合。

根据是否需要参考历史情况，我们将基于使用行为的 POI 过期挖掘划分为时序异常和事件异常两类。

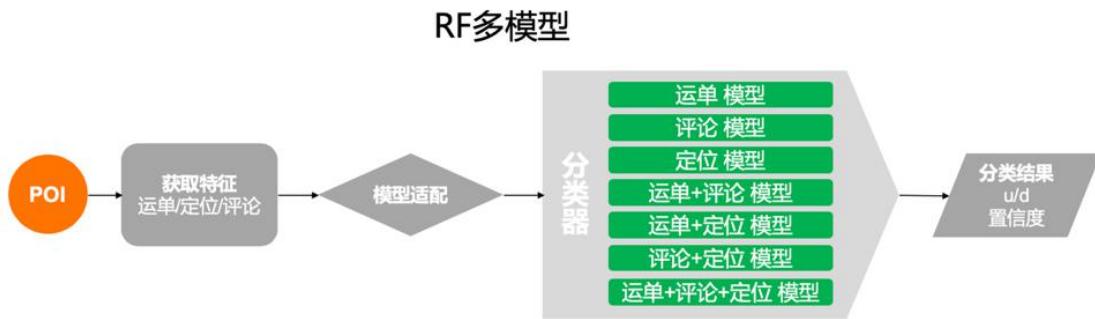
2.2.1 时序异常

POI 的存活状态可以通过关联的使用行为量活跃度间接反映出来，从使用行为量的趋势角度尝试迭代解题。



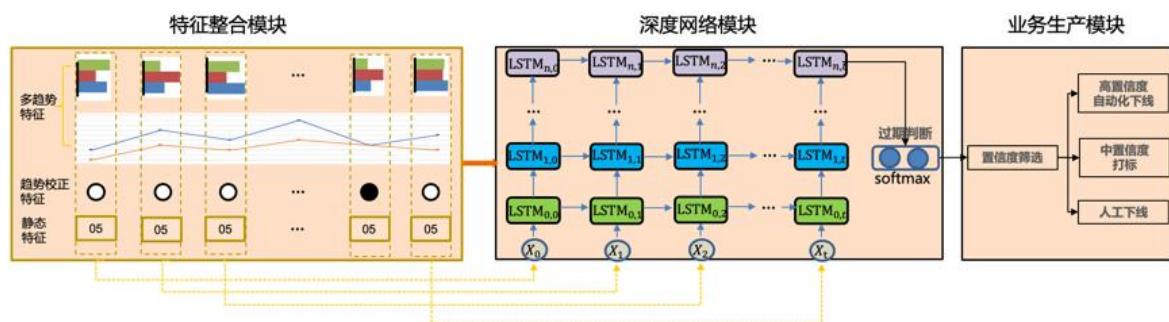
趋势模型的主要思想是，统计某个时间窗口关联的使用行为量活跃度来衡量 POI 的存活状态，并通过分析活跃度相对于历史情况的衰减程度来判断 POI 是否过期，其基本假设是时序趋势下降与 POI 过期正相关。以已知活跃度信息的逐月统计量时间序列为特征，我们完成了 RF→RNN→模型融合→Wide&Deep 四个迭代阶段的研发。

鉴于 RF 在分类决策问题中表现出的精度高、不易过拟合、对数据集适应能力强、落地高效以及对于规则思维的天然吻合度，可成为验证解题方案可行性的首选。方案是将每种特征的每个时间节点值作为一个输入维度来构建模型。RF 凭借高准确和高产出落地投产，验证了行为量趋势应用于过期挖掘的重要意义。

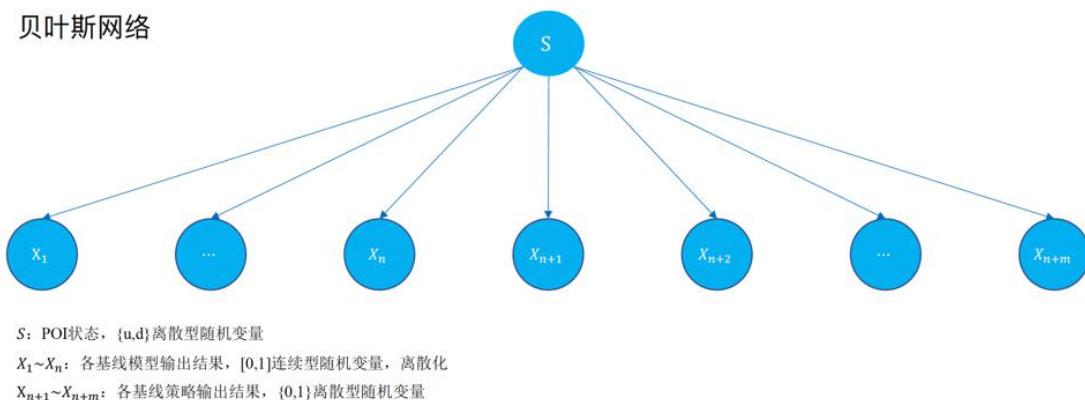


之后，针对 RF 存在的一些不足来做进一步的技术升级。首先，模型无法学到连续时间节点之间的趋势关联性，时序信息未得到充分利用；其次，对于不同种类特征缺失、长短序列融合等问题需要建立定制化模型来解决，多模型增加了维护负担。因此，要选择时序领域优势明显的 RNN 模型进行迭代升级。

通过构建多层 LSTM 深度网络实现了趋势关联信息的深度挖掘，同时针对不同热度分段的数据分布差异性，采取各自最优的缺失特征填充方式，避免了多模型式的解题方案，便于业务维护。RNN 模型使发现能力，特别是自动化能力得到较大提升。



虽然 RNN 相对于 RF 提升了对于时序特征的学习能力，但信息不足依然限制了模型的自动化能力。我们进一步开发了能够实现多源信息融合决策的融合模型。思想是将 RF、RNN、拆迁区域等现有各基线模型、策略以及白名单作为子分类器纳入统一框架内考虑，在此基础上构建贝叶斯网络，做决策层的多源信息融合。相较于特征层的多源信息融合，它落地快且效果明确，为过期业务提供了稳定的高准确自动化下线产出，自动化能力大幅提升。

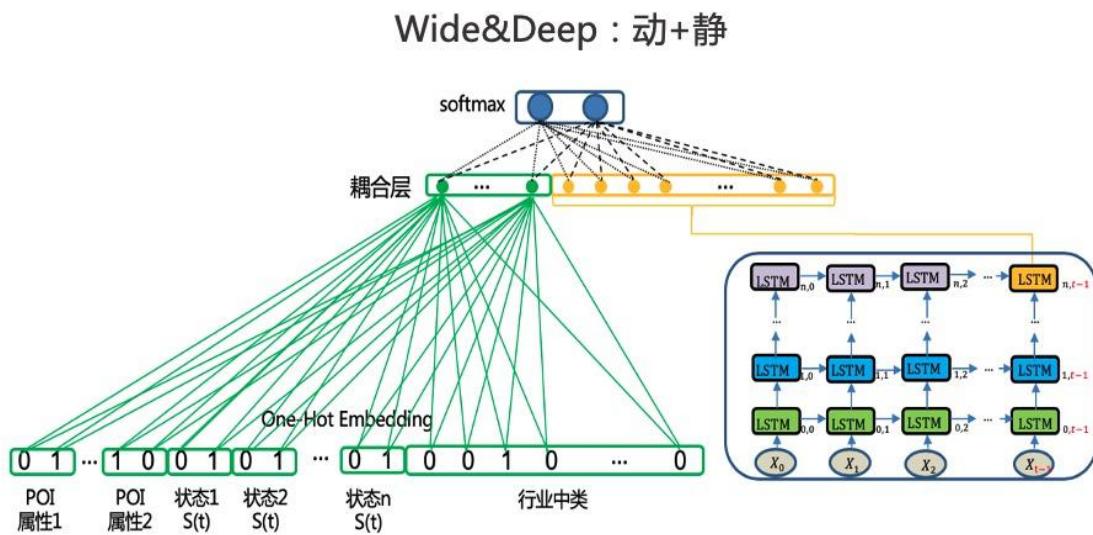


第四个阶段是从多源信息融合的角度进一步优化。一方面，决策层融合相比特征层融合存在更多的信息损失；另一方面，一些模型/策略只在部分品类的 POI 上满足业务投产的准确率标准，导致不达标品类的产出结果未得到充分利用。因此，从实现特征层多源信

息融合的角度出发，借鉴 Wide&Deep 思想搭建新业务模型。

整体思路是，将众多不可量化或比较的属性特征和状态信息特征进行编码表征，再经过一层全连接层降维后作为 Wide 部分；将 RNN 模型作为 Deep 部分，最后将两部分耦合。

模型经过多轮迭代优化可稳定投产，自动化能力得到进一步提升，已成为过期挖掘业务中覆盖行业广、自动化解题能力突出的综合性模型。



综合以上，人机解题比大幅下降，解决了人工核实率低、处理能力不足的问题，并且大幅降低了成本。

2.2.2 事件异常

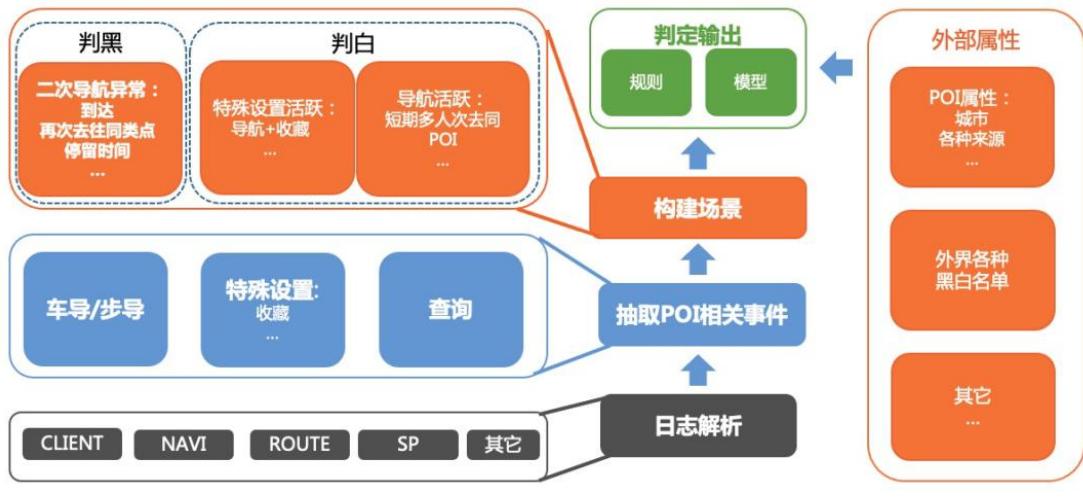
现有的时序异常模型主要依赖于使用行为量的趋势特征做判断，存在挖掘资料覆盖上的天花板，以加油站、ATM、公共厕所等为例，这些类型的 POI 因自身属性的原因导致无挖掘资料，趋势模型无能为力。

因而提出基于日志（Session）的异常事件模型，统计陌生群体到达过期 POI 后需求不满足引发的异常事件，补位时序异常模型的挖掘盲区，即无需参考历史情况，仅利用日志抽取 POI 关联的异常行为事件，累积近期异常事件衡量 POI 的存活状态是否正常。

日志挖掘难点

海量的日志行为。直接使用不仅消耗资源大，且有大量的冗余数据造成干扰。如何在海量行为中抽取与过期相关的特征是一个艰难的工程。

行为随机性大。例如，很多情景里快到终点前会提前结束导航从而无法判断是否到达目的地；有些情景是规划去一个目的地但从末端轨迹可以判断实际去的地点天差地别。



解题框架

针对上述问题，主要通过实地评测的过期 POI case 分析来构建具体的异常事件场景，例如到达后试图报错、到达后快速发起二次同质化导航等，以上统计量作为特征输入，由此可聚焦相关日志片段并降低随机行为噪音。整体解题框架如上图所示，从不同的 Session 源解析与 POI 相关的事件，按照时间顺序组合成场景 1、场景 2、场景…，加入外部属性如类型、城市等，以目的地 POIID 按照时间窗口归并生成相应的统计特征，输入 LR 模型，输出 POI 的过期得分。目前采用 LR，优点是简单粗暴压住噪声。

挖掘效果

Session 异常事件模型有效补充其他手段未能覆盖的解题集合，专攻汽车服务、生活服务、娱乐场所、金

融保险服务等品类 POI，是过期挖掘不可或缺的组成部分，且未来仍有较大的泛化召回空间。

2.3 阶段三：基于人地关系的 POI 过期挖掘

2.3.1 人地关系建设

趋势特征丰富（厚）的过期 POI，容易被趋势模型挖出。而当趋势特征（使用行为）稀少（薄）时，模型发现能力较差。所以该阶段需要解决资料薄的问题，通过对关键群体（ >2 ）线索的捕捉，降低对资料厚度的依赖。洞察 POI 的关键群体的行为，有可能找到发现甚至解释 POI 过期的特征。

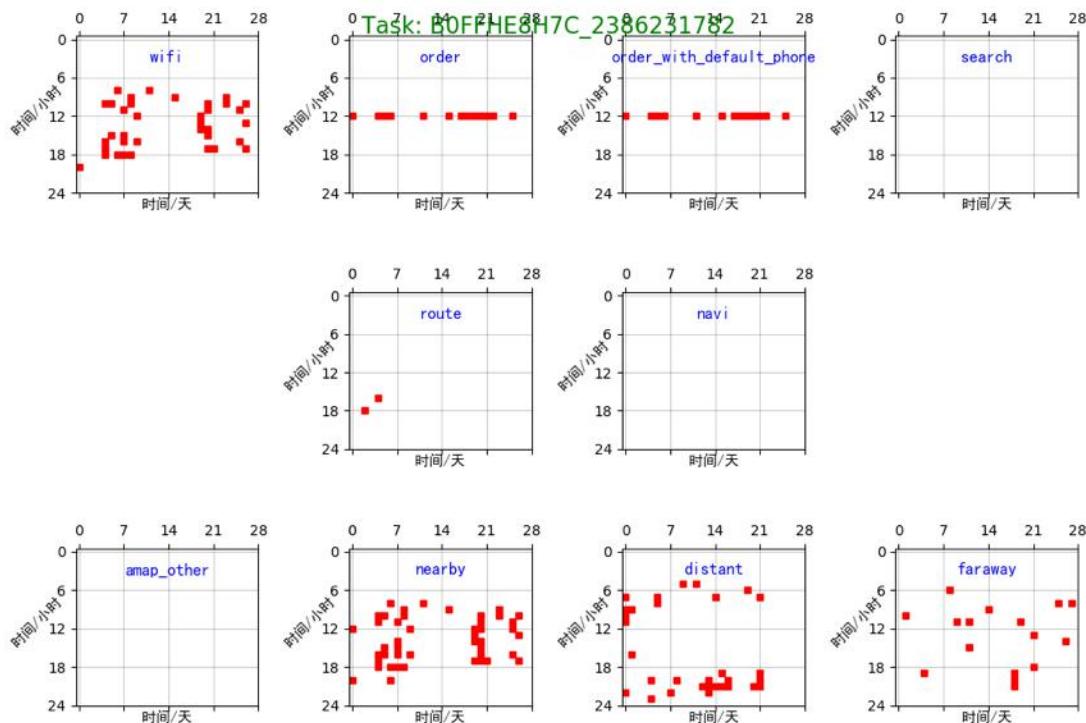
因此，第一步我们需要建设人地关系，找出所谓的关键群体，称之为内部群体，是指：对 POI 有依赖的群体，这种人地关系，我们称之为内部关系，其它均为外部关系。

第二步基于内部群体的时空运动模式的变化发现过期 POI，补位趋势特征稀疏时的召回问题，局限性：内部群体基本不变的 POI 更名等场景不可解。第一步人地

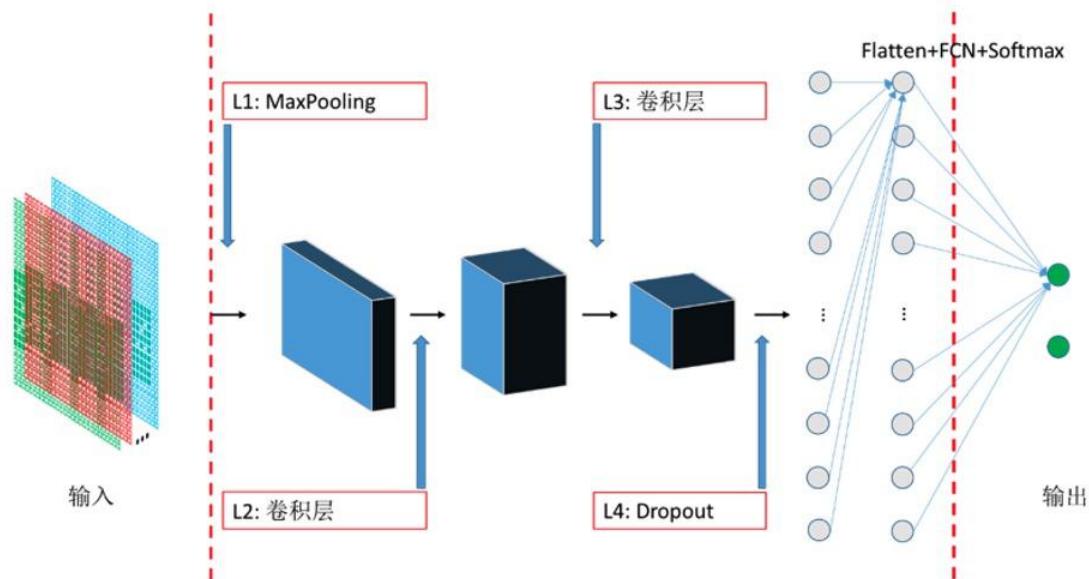
关系建设基本完成，大体分为数据层、行为层和模型层三层，分别介绍如下：

在数据层，收集可能与 POI 相关的数据源，打通各个数据孤岛，将不同类型的数据关联到高德 POI 上。

在行为层，将行为特征表达在窗口为 X 天的二维矩阵上，如图所示。矩阵表示能够更加清楚地反映行为的周期性规律。不同行为序列可以看作是不同通道的矩阵，很好地适配行为数据的异步性，同时保持可扩展性（每多一种行为，可增加一个通道表示）。



在模型层，面向多通道的矩阵特征，采用深度卷积网络完成分类任务，其基本结构如下：



该结构一定程度地缓解由于数据不完备导致的特征稀疏，有效地学习行为的时间规律，取得符合预期的结果，验证了模型的可用性。在模型层，通过补充的召回策略，帮助将内部关系对 POI 的覆盖度进一步提升，完成从 $0 \rightarrow 1$ 的建设。

总结

过期挖掘已经成为增强 POI 现势性的绝对主力手段。这条以大数据挖掘为主导的路线还远远没有达到终局，未来的演进方向至少有以下几个：内部群体时空转移本质化通盘解题，降低对资料厚度的依赖；面向过期

现象的定向挖掘能力提升；POI生命力画像构建；生态探索，从逆向解题向逆向+正向解题渗透。我们将致力于为提供给用户更美好的出行服务体验而努力。

招聘

阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位 地 点：北 京。欢 迎 投 递 简 历 到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

漫话地图之高精地图生产中的坐标系

作者：晶阳

导读

高精地图区别于普通电子地图，一般来说高精地图的精度要求在亚米级甚至厘米级，其所含有的道路交通元素也更丰富和细致。这就造成高精地图数据生产过程中会遇到很多普通地图没有的问题和困难，更需要有专业的测绘知识解题。

本文从现实世界到地图数据的抽象基础—坐标系着手，介绍相关领域知识，希望对大家深入高精地图领域有所裨益。

本文属于漫话地图系列，将会持续介绍地图行业一些有趣的知识点，欢迎持续关注。

1. 如何定义地球

1.1 地球椭球体

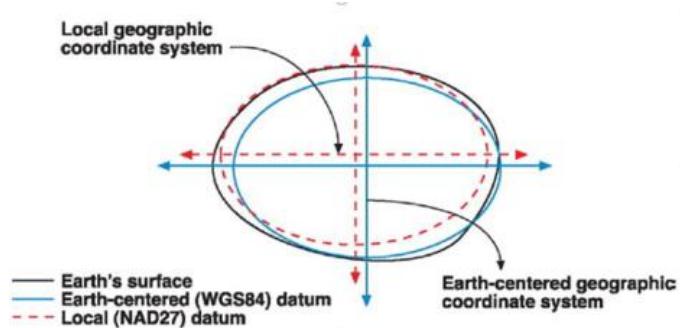
众所周知地球表面凸凹不平，是一个无法用数学公式表达的曲面，这样的曲面不能作为测量制图的基准面。那么假想一个扁率极小的椭圆，绕大地球体短轴旋转所形成的规则椭球体称之为地球椭球体。

1.2 大地水准面

大地水准面是海平面向陆地内部的自然延伸形成的闭合面，由于地球重力面分布不均匀，大地水准面仍然不是一个标准的数学模型，无法作为制图表达。

1.3 大地基准面

大地基准面是设计为最密合部分或全部大地水准面的数学模式，它由椭球体本身及椭球体和地表原点间关系来定义，从而衍生出参考椭球体(参考于标准椭球体)的概念。此关系能以 6 个量来定义，通常是大地纬度、大地经度、原点高度、原点垂线偏差之两分量及原点至某点的大地方位角。

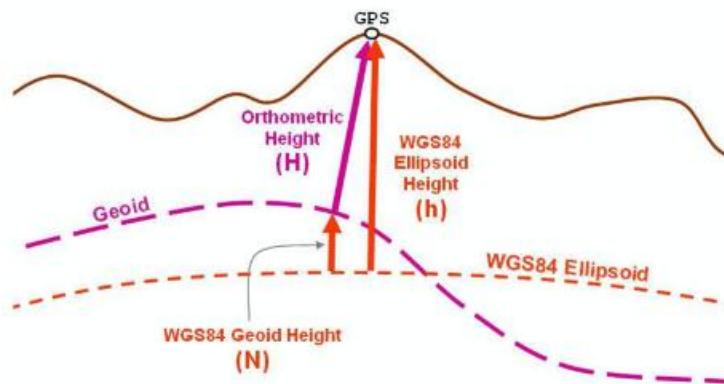


1.4 参心坐标系/地心坐标系

地心坐标系是以地球质心为原点建立的空间直角坐标系 XYZ，或以球心与地球质心重合的地球椭球面为基准面所建立的大地坐标系 BLH；参心坐标系是以参考椭球体的几何中心为原点的坐标系。

1.4.1 常见的坐标系—WGS 84 坐标系

- 原点：地球质心。
- Z 轴：BIH (1984.0) 定义的地极 (CTP) 方向，即国际协议原点 CIO，它由 IAU 和 IUGG 共同推荐。
- X 轴：指向 BIH 定义的零度(本初)子午面和 CTP 赤道的交点。
- Y 轴：和 Z, X 轴构成右手坐标系。
- WGS84 椭球体：国际大地测量与地球物理联合会第 17 届大会测量常数推荐值，长轴 6378137.000m，短轴 6356752.314，扁率 1/298.257223563。



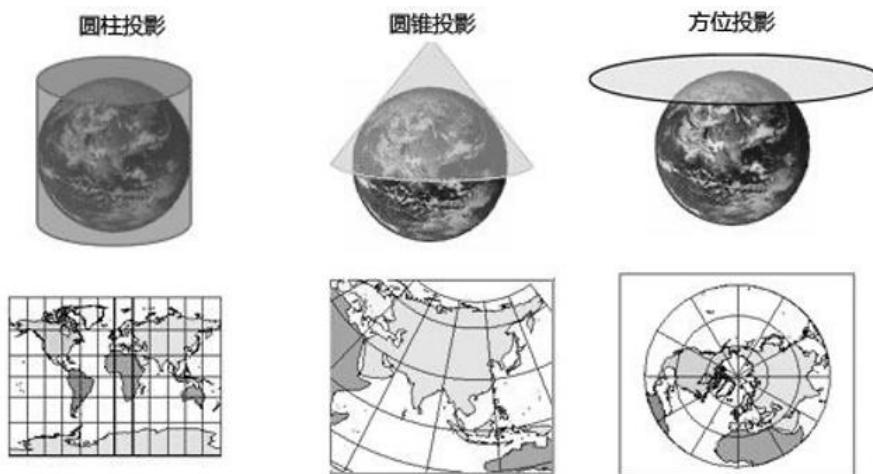
1.4.2 北京 54/西安 80/GCJ2000

- 北京 54：前苏联克拉索夫斯基椭球体，是前苏联 1942 坐标系的延伸，原点不在北京而是在前苏联的普尔科沃。
- 西安 80：国际大地测量与地球物理联合会推荐值 (IAG75 椭球体)。西安大地原点设在陕西省泾阳县永乐镇，位于西安市西北。1985 国家高程基准采用青岛验潮站 1952—1979 年确定的黄海平均海水面。
- GCJ2000：地心坐标系在我国的具体体现，和 84 椭球体仅有 0.11mm 的误差。

2 如何定义二维地图

2.1 地图投影

利用一定数学法则把地球表面转换到平面上的理论和方法称为地图投影。由于地球表面是一个不可展平的曲面，所以运用任何数学方法进行投影转换都会产生误差和变形，为按照不同的需求缩小误差，就产生了各种投影方式，如圆柱投影、圆锥投影、等角投影、等面积投影、切投影、割投影等。

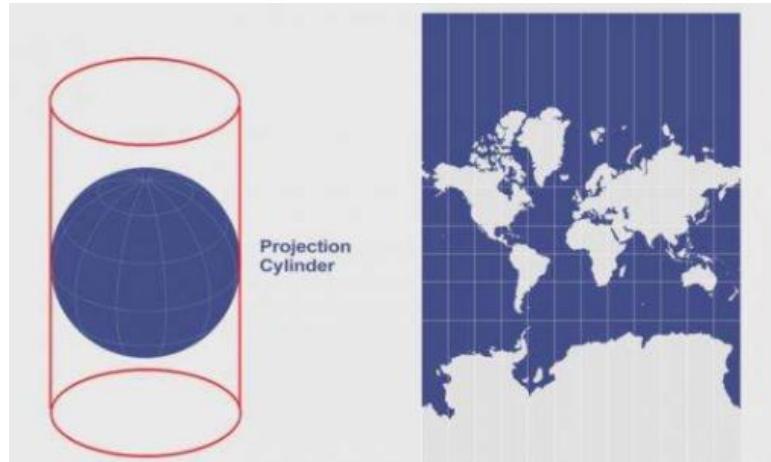


2.2 常见的几种投影方式

2.2.1 墨卡托 / Web 墨卡托

一种正轴等角切圆柱投影。

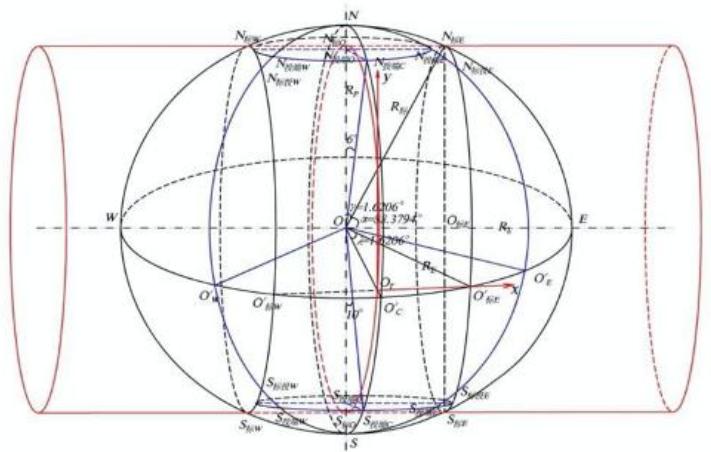
- 等角：保证对象形状不变以及方向位置正确。
- 圆柱：保证纬线经线平行相互垂直且经线间隔相同。
- 缺点：纬线间隔从赤道向两级逐渐增大，面积变形大。
- Web 墨卡托：Google 首创，把地球模拟为球体而非椭球体，近似等角。



2.2.2 高斯投影/UTM 投影

两个投影很相似，高斯投影为等角横切椭圆柱投影，前苏联、中国和德国等所采用。UTM 投影(通用横轴墨

卡托）是一种等角横割椭圆柱投影，为世界上大部分国家采用。



3. 如何定义高精地图

相对于普通电子地图，高精地图采集方式发生了质的变化。从误差米级到厘米级能力提升主要源于高精采集车上更加丰富和立体的传感器设备，包括但不限于激光雷达、RTK、惯导、摄像机等，有能力反映更加精细的真实世界。

高精地图以采集车为基础，可以分为如下几个坐标系。

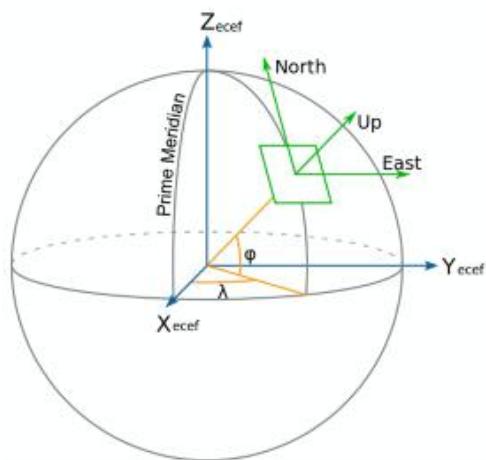
3.1 ECEF—地心地固坐标系

原点位于地球质心， z 轴沿着地轴指向北极， y 轴沿着赤道平面与格林威治子午面的交线上， y 轴在赤道平

面与 x 轴 z 轴满足右手法则，该坐标系一般和 WGS84 坐标系相互转换，属于同一基准下不同表达。

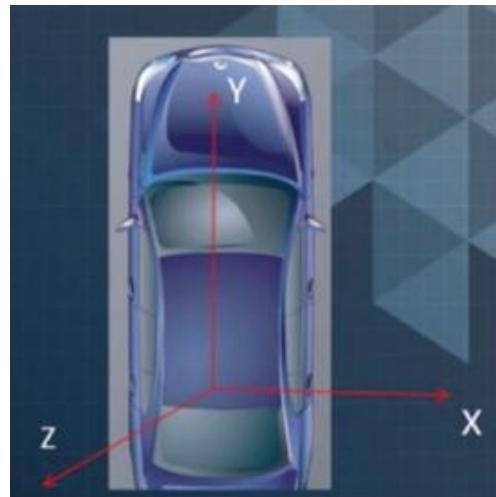
3.2 东北天/当地水平坐标系

当地水平坐标系的原点位于载体所在的地球表面， x 轴和 y 轴在当地水平面内，分别指向东向和北向， z 轴垂直向上，与 x 轴 y 轴满足右手法则，称为东-北-天 (e-n-u) 坐标系，相对另一坐标系(北东地)。



3.3 车体坐标系

车体坐标系原点在载体质量中心与载体固链， x 轴沿载体轴指向右， y 轴指向前， z 轴和 xy 满足右手坐标法则指天，又称为右-前-上 (r-f-u) 坐标系



3.4 激光雷达坐标系

和选用雷达类型及安装方式有关，一般来说原点位于多线束旋转轴的交点处，z 轴沿着轴线向上，其测量的点坐标是在激光雷达坐标系下的三维坐标。雷达与载体固链，通过坐标系外参和车体姿态可以得到激光点的世界坐标系。

3.5 IMU 坐标系

IMU 坐标系的坐标原点在陀螺仪和加速度计的坐标原点，xyz 三个轴方向分别与陀螺仪和加速度计的对应轴向平行。在解算惯性导航系统中 IMU 与车体固链，因此在不考虑安装误差角的情况下，载体坐标系也即为 IMU 坐标系。

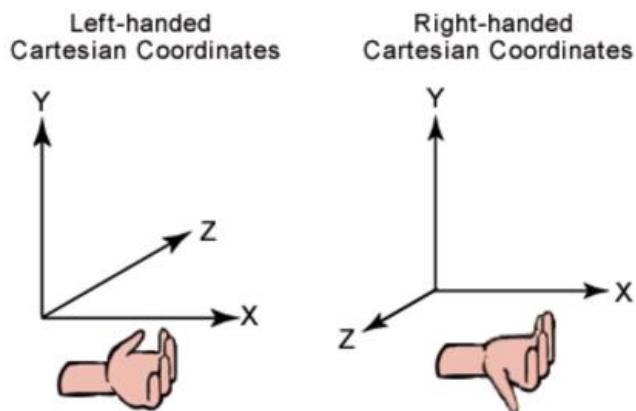
3.6 相机坐标系

以自己的光心为原点， xy 一般由相平面决定， x 朝右， y 轴朝下建立一个局部坐标系，一般与载体固连，通过外参进行刚性变化转换。

3.7 什么是右手坐标系

右手系(right-hand system)是在空间中规定直角坐标系的方法之一，两种定义方式：

- a. 把右手放在原点的位置，使大姆指，食指和中指互成直角，把大姆指指向 x 轴的正方向，食指指向 y 轴的正方向时，中指所指的方向就是 z 轴的正方向。
- b. 如果当右手(左手)的大拇指指向第一个坐标轴(x 轴)的正向，而其余手指以第二个轴(y 轴)绕第一轴转动的方向握紧，就与第三个轴(z 轴)重合，与之相反则为左手系。



4. 坐标系之间的转换

4.1 地学概念上的坐标转换

坐标系变换就是在相同空间点在不同椭球体下的不同坐标表达形式的数值换算，主要分为三种：

- 大地坐标系与空间直角坐标系的相互转换(经纬度转 ECEF)。
- 空间直角坐标系与站心坐标系的转换(ECEF 转工
程)。
- 大地坐标系与平面坐标系的转换(经纬度转投影
坐标)

4.1.1 相同基准面下的变化-经纬度转笛卡尔坐标公 式

相同基准下，将大地坐标系转换为空间直角坐标系的公式为

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} (N + H) \cos B \cos L \\ (N + H) \cos B \sin L \\ [N(1 - e^2) + H] \sin B \end{bmatrix}$$

其中：

$$N \text{ 为卯酉圈的半径}, N = \frac{a}{\sqrt{1 - e^2 \sin^2 B}} \quad e^2 = \frac{a^2 - b^2}{a^2}$$

a 为地球椭球的长半轴；

b 为地球椭球的短半轴。

<https://baike.com>

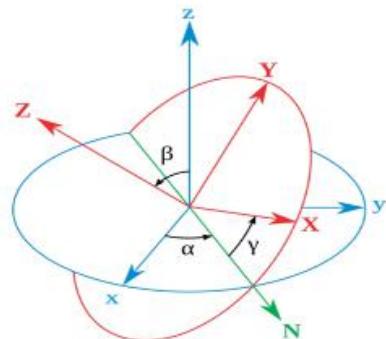
4.1.2 不同椭球体间的变化—布尔沙七参数模型

需要 7 个参数

- 三个坐标平移量 (ΔX , ΔY , ΔZ)，即两个空间坐标系的坐标原点之间坐标差值。
- 三个坐标轴的旋转角度 ($\Delta \alpha$, $\Delta \beta$, $\Delta \gamma$)，通过按顺序旋转三个坐标轴指定角度，可以使两个空间直角坐标系的 XYZ 轴重合在一起。
- 尺度因子 m ，即两个空间坐标系内的同一段直线的长度比值，通常 m 值等于 1

4.2 三维空间变换相关的一些概念

4.2.1 欧拉角 / 欧拉旋转

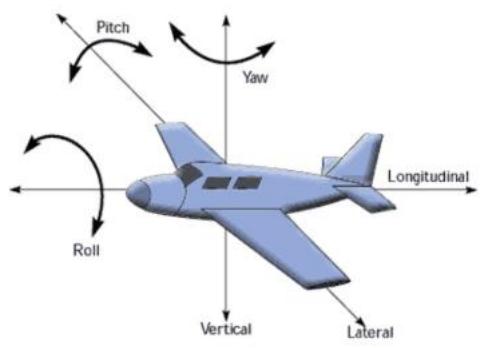


旋转步骤：

- 物体绕全局的 z 轴旋转 α 角。
- 继续绕自己的 X 轴（也就是图中的 N 轴）旋转 β 角。
- 最后绕自己的 Z 轴旋转 γ 角。

详细介绍请[看这里](#)。

4.2.2 万向节死锁



沿着机身右方轴（X）进行旋转，称为 Pitch 倾仰。

沿着机头上方轴（Y）进行旋转，称为 Yaw 偏航。

沿着机头前方轴（Z）进行旋转，称为 Roll 桶滚。

详细介绍[请见这里](#)。

4.2.3 旋转矩阵

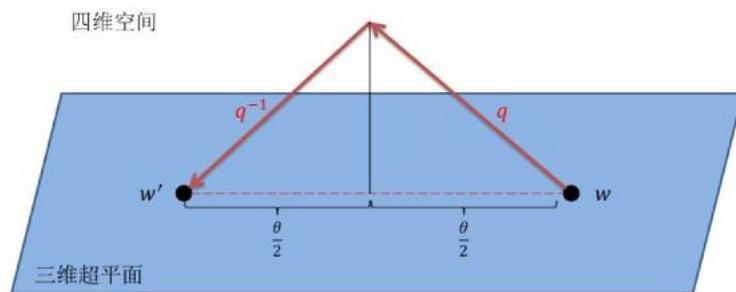
任何维的旋转可以表述为向量与合适尺寸的方阵的乘积，最终一个旋转等价于在另一个不同坐标系下对点位置的重新表述。

推导过程[请见这里](#)。

4.2.4 旋转向量

向量旋转公式最早由 Rodrigues 提出，用一个三维向量来表示三维旋转变换，该向量的方向是旋转轴，其模则是旋转角度，设旋转向量的单位向量为 r ，模为 θ ，三维点 p 在旋转向量 r 的作用下变换至 p'

4.2.5 四元数



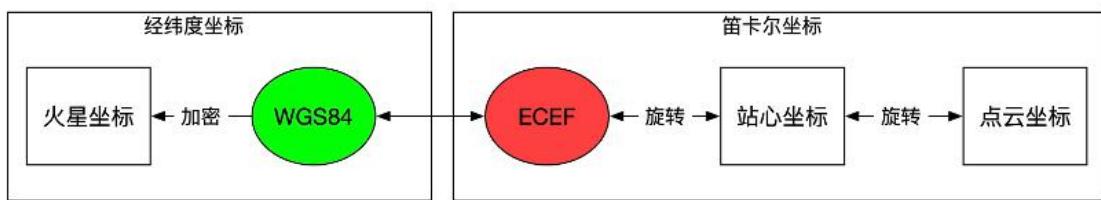
四元数优势：

- 解决万向节死锁（Gimbal Lock）问题。
- 仅需存储 4 个浮点数，相比矩阵更加轻量。
- 四元数无论是求逆、串联等操作相比矩阵更加高效。

相关资料

- [四元数与三维旋转](#)
- [理解四元数](#)
- [四元数的可视化](#)

5. 坐标系转换关系



6. 关于坐标系的一些周边

6.1 EPSG 欧洲石油测绘组织

European Petroleum Survey Group: 成立于 1986 年，它负责维护并发布坐标参考系统的数据集参数，以及坐标转换描述，该数据集被广泛接受并使用，通过 Web 发布平台进行分发。

SRID: OGC 标准中空间参考系统的唯一 ID，详见[这里](#)

6.2 WGS 1984 Web Mercator

EPSG:3785 EPSG 在 2008 给 Web Mercator 设立的 WKID，但是这个坐标系的基准面是正圆球，不是 WGS 1984，存在了一段时间后被弃用。

EPSG:3857 EPSG 为 Web Wercator 最终设立的 WKID，也就是现在我们常用的 Web 地图的坐标系，并且给定官方命名 WGS 84 / Pseudo-Mercator

OPENLAYER:900913 Web Mercator 已经成为 Web 地图领域的事实标准，尽管这个坐标系由于精度问题一度得不到官方认证，Google 为 Web Mercator 任性地制定了这个 ID。

6.3 Proj.4

Proj.4 是开源 GIS 最著名的地图投影库，许多开源软件的投影都直接使用了 Proj.4 的库文件。该项目遵循 MIT license，用 C 语言编写，由 USGS 的 Gerald I. Evenden 在 1980 年代创立并一直维护到退休，目前有 C、Java、Python、JS 等多语言版本维护。功能主要有经纬度坐标与地理坐标的转换，坐标系的转换，包括基准变换等。

7. 参考资料

- [无人驾驶中的坐标系](#)
- [空间坐标与投影系统](#)
- [如何理解 3D 动画中的欧拉角以及死锁](#)
- [如何形象地理解四元数](#)
- [刚体在三维空间的旋转](#)

盘点 | 有哪些大数据处理工具?

作者：降云

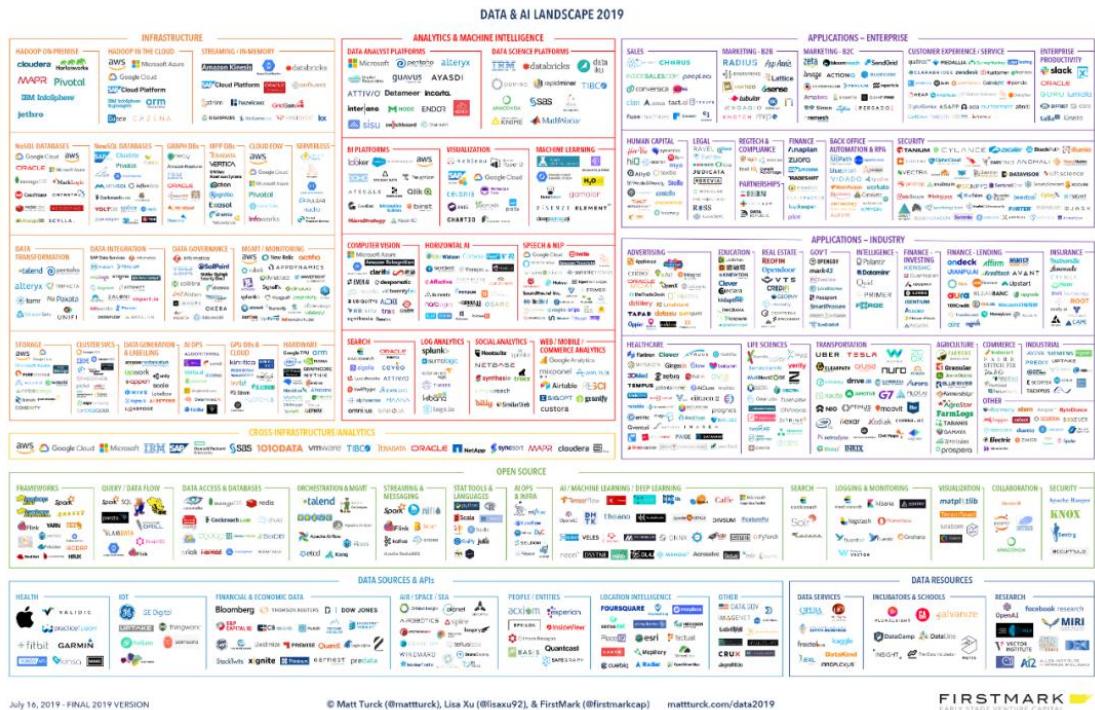
导读

近几年里，大数据行业发展势头迅猛，故而相应的分布式产品和架构层出不穷，本文分享作者在大数据系统实践过程中接触过的一些工具及使用感受，抛砖引玉，和同学们一起构建一个分布式产品的全景图。

下图是由著名的数据观察家 Matt Turck 在他的 [BLOG](#) 里发出的 2019 年人工智能和大数据产业图，他从 2012 年开始每年都会绘制一张，大致描述这个产业里的公司及其数据相关的产品，以及所属问题的领域。

这里面大部分是商业软件，而对于绝大多数互联网公司，中间绿色的开源产品可能大家接触的更多一些，而这些产品里，绝大多数都属于 Apache 基金会。

算法篇-盘点 | 有哪些大数据处理工具？



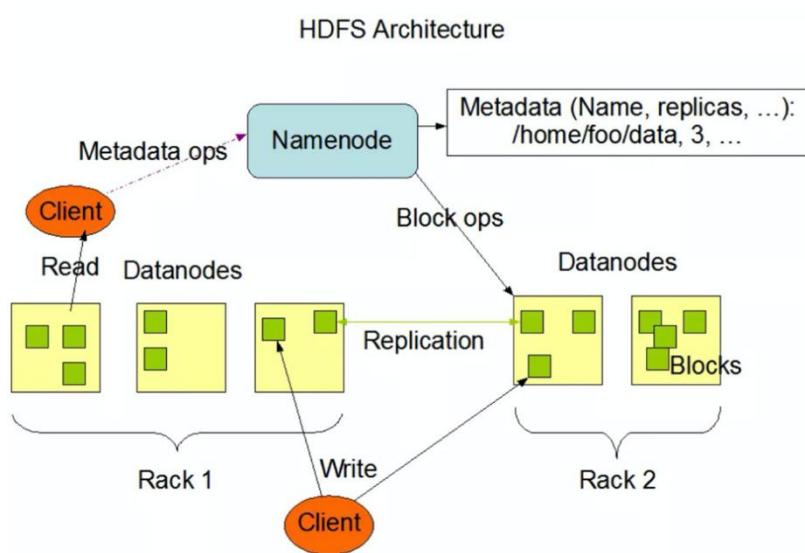
下面我从中挑选一些东西随便聊聊，因为是随便聊聊，所以知识点并不全，也不能帮助大家知道如何搭建和使用，以及如何避坑，只是谈谈我对这些东西的印象，描述一个大概的轮廓，如有使用需求可以搜索网上其它文章，资料还是很多的。当然，大家对其中的内容有兴趣可以随时找我交流讨论，对文中如有描述错误的地方也欢迎大家斧正，共同学习。

Apache Hadoop

官网：<http://hadoop.apache.org/>

Hadoop项目下包含了很多子项目，从计算到存储都有，比如HDFS、MapReduce、YARN、HBase。

HDFS 全称叫做 Hadoop 分布式文件系统，其主要由一个 NameNode(NN)和多个 DataNode(DN)组成，数据文件会分成多个 Block，这些 Block 按照不同主机，不同机架的策略以默认一备三的情况下分布存储在各个节点。现在每个 Block 大小默认是 128MB，以后随着磁盘寻址速度的增加，这个 Block 也会不断增大。而 NN 里面则存储了这些 Block 元数据的信息，这样客户端进行数据查询的时候，DN 告知所需数据的位置。从这种结构上能看出一些比较明显的问题就是 NN 节点的单点问题，所以在 Hadoop 2.x 的时候，针对 NN 做了一些改进。



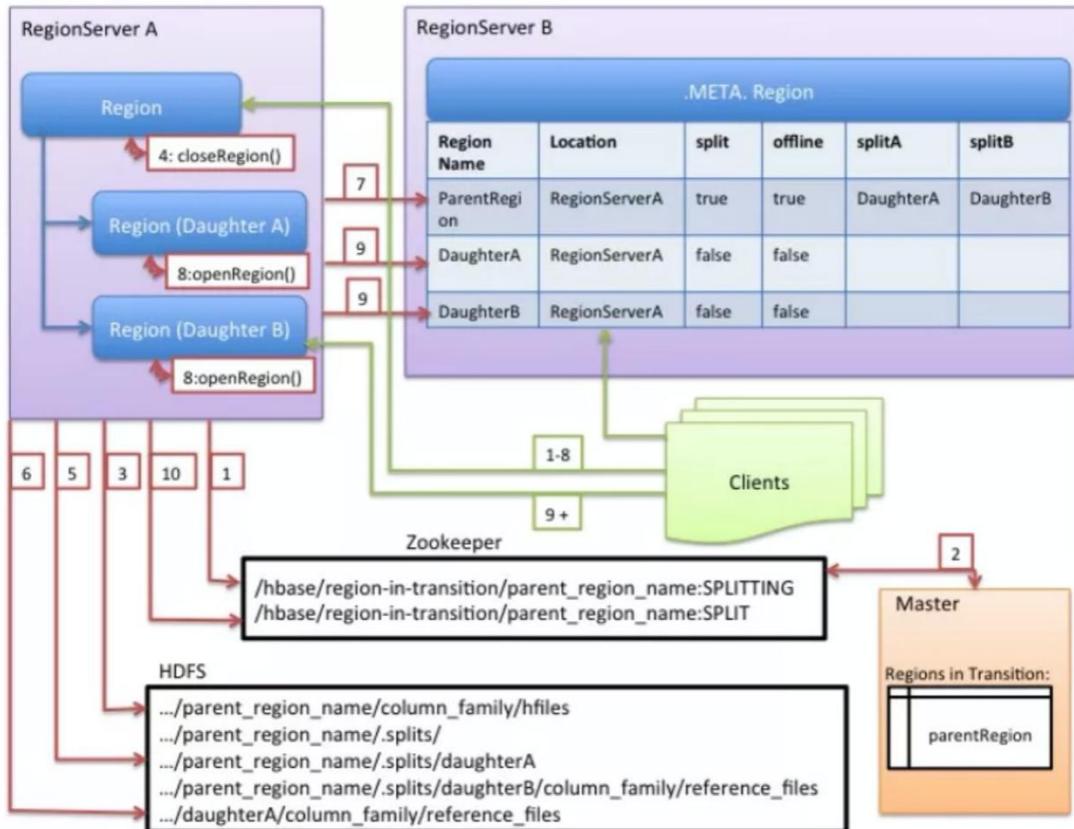
首先是在系统可用性上，增加了一个 StandBy 状态的 NN，作为服务中 NN（Active NN）的备机，当服务中的 NN 挂掉后，由 StandBy 的 NN 自动接替工作。而 NN 节点状态的健康和服务切换，由 ZKFC 负责。主备 NN 之间的信息同步则由 Quorum Journal Node 负责。

其次，由于单台 NN 中存储了大量的元数据信息，所以随着 HDFS 数据量的不断增加，显然 NN 必将成为系统的瓶颈，为了解决这个问题，Hadoop 2.x 增加了 Federation，该技术允许系统中有多台 NN 同时对外提供服务，这多台 NN 将 DN 中的所有文件路径进行了横向拆分，每个 DN 负责不同的路径，达到了横向扩展的效果。

除了 HDFS，Hadoop 2.x 也引入了 YARN，该工具负责对集群中的资源进行管理和任务的协调。该工具分成一个 ResourceManager(RM)和多个 NodeManager(NM)，当一个任务提交给 YARN 之后，会先在某一服务器上启动一个 ApplicationMaster(AM)，AM 向 RM 申请资源，RM 通过 NM 寻找集群中空闲的资源，NM 将资源打包成一个个 Container，交给 AM。AM 将数据和程序分发到对应节点上处理，如果某个 Container 中的任务执行失败了，AM 会重新向 RM 申请新的 Container。

Apache Hadoop HBase & Kudu

官网: <http://hbase.apache.org/>



众所周知，HBase 一个分布式列式存储系统，同样属于 Hadoop 的子项目，列式存储的优劣在这里不说了，提一下 HBase 的 WAL 和 LSM。WAL 全称为 Write Ahead Log，只是在数据修改操作前，会先将此操作记录在日志中，这样一旦服务崩溃，通过该日志即可进行数据的恢复，提到这里有些人就会联想到 MySQL，因为 InnoDB 引擎的 redo log 就是典型的 WAL 应用。而在 HBase 中该功能是由叫做 HLog 的模块所完成的。

再说 LSM，其全称为 Log Structured Merge Trees，介绍原理的文章也有很多，在 HBase 中，LSM 树是 MemStore 模块的底层存储结构，而 MemStore 有三个作用，一是当有数据写入的时候，直接写到 MemStore 中，从而提升写数据的效率。二是充当读取数据时的缓存。三是定期对数据操作去重，并进行数据落盘。HBase 的主要角色分别有 HMaster 和 HRegionServer，同样是一对多的关系，而各节点的状态全都交由 Zookeeper 负责。Kudu 是一个和 HBase 非常类似的产品，其不同之处在于 Kudu 不依赖 Zookeeper 来管理自己的集群，并且 HBase 的数据是保存在 HDFS 上的，而 Kudu 拥有自己的数据文件格式。

Apache Spark

官网：<https://spark.apache.org/>

Spark 是由加州大学伯克利分校推出的分布式计算引擎，在 Spark 的官方主页上有一张和 Hadoop 的性能对比图，姑且不谈这张图中数据的准确性，但是 Spark 的确将 Hadoop（主要是指 MapReduce）的性能提升了一个量级。我理解这主要得益于两个方面：第一个是 Spark 计算过程中生成的中间数据不再落盘，没有了 Spill 的阶段。第二个是引入 DAG 对任务进行拆解，一个完整的 Job 被分成多个 Stage，每个

Stage 里面又有多个 Task，通过一张有向无环图，使得没有依赖关系的 Task 可以并行运行。

Spark 不只是在批处理上有所成绩，而是更加注重整个生态圈的建设，其拥有流式处理框架 SparkStreaming，采用微批的形式达到类似流处理的效果，现在又推出了 Structured Streaming，实现基于状态的流处理框架。此外还拥有 SparkSQL 来帮助非开发人员更加便捷的调用 Spark 的服务和 Spark MLlib 这个机器学习库。

Spark 虽好，但其对内存资源消耗也很大，同时也使得他在稳定性上不如 MapReduce，所以有些大公司数仓的日常任务仍旧采用传统 MapReduce 的方式执行，不求最快，但求最稳。我们的系统在刚从 MapReduce 上切到 Spark 时，每天夜里也是任务异常频发，最后调整了任务和资源分配，再加上一个很粗暴的重试机制解决了。

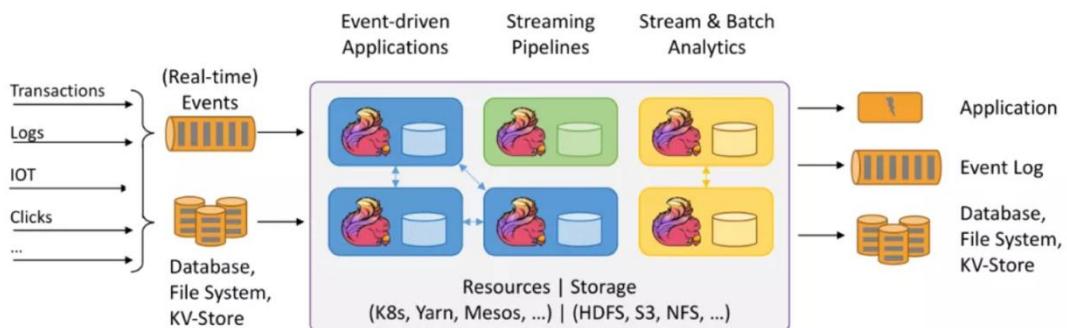
Apache Flink

官网：<https://flink.apache.org/>

Flink 是德国 Data Artisans 公司开发一款分布式计算系统，该公司于 19 年初被阿里巴巴集团收购。包括 Spark 和 Kafka，

也都看到了未来流式计算的前景是非常巨大的，纷纷建立属于自己的流式计算生态圈。

Flink 和 Spark Streaming 相比，前者是真正的流式计算，而后者是微批处理，虽然批次足够小，但其本质毕竟还是批处理，这就导致有些场景 SparkStreaming 注定无法满足，虽然 Spark 现在将重心转移到了 Structured Streaming，它弥补了 Spark Streaming 很多的不足，但是在处理流程上仍然是微批处理。



而 Flink 在设计之初就同时考虑了批处理和流处理这两种需求，所以使用者也可以只通过一个计算引擎，就能实现批处理和流处理两种计算场景，其主要几个需要清楚的特性我觉得分别是：State 状态管理，CheckPoint 容错机制，Window 滑动窗口，和 Watermark 乱序解决。这些内容网上都有很多介绍，不再阐述。

Apache Impala

官网：<https://impala.apache.org/>

Impala 是 Cloudera 公司用 C++ 开发的支持 SQL 语义的查询系统，可以用来查询 HDFS、HBase、Kudu 的内容，也支持多种序列化和压缩格式，因为也是基于内存的计算，比传统 MapReduce 快很多。不过因为已经使用了 Spark，所以组里并没有对 Impala 进行大规模的应用。经过一些零散的调研和了解，好像其它公司对 Impala 的应用也不是非常多。

Apache Zookeeper

官网：<https://zookeeper.apache.org/>

Zookeeper 无论在数据系统还是在其它后端系统的使用场景都非常广，它可以用作分布式锁服务，可以用做系统的配置中心，可以协助完成一致性算法的选主过程，可以用于 ZKFC 做节点健康情况的探查，总之用处还有很多。而它的工作机制，基本就是 ZAB 协议的机制，一个支持崩溃恢复的原子广播协议，其主要组成也是由一个 Leader 和多个 Follower 组成的，数据的提交遵循 2PC 协议。当 Leader 崩溃时，Follower 会自动切换状态开始重新选主，重新选完之后再进行多节点的数据对齐。

Apache Sqoop

官网：<https://sqoop.apache.org/>

一款用于在传统关系型数据库和 HDFS 之间互相进行数据传递的工具，无论是 import 还是 export 都提供了大量的参数，因为是分布式执行，数据传输的速度也非常快。只是在使用的过程中需要注意数据源中的异常数据，会比较容易造成数据传递过程中的异常退出。为了弥补 Sqoop 的功能单一，推出了 Sqoop 2，架构上比 Sqoop 1 复杂了很多，不过我没有用过。

Apache Flume

官网：<http://flume.apache.org/>

分布式数据传输工具，支持包含文件、Netcat、JMS、HTTP 在内的多种数据源。其结构上分成 Source、Channel、Sink 三部分，Source 将获取到的数据缓存在 Channel 中，这个 Channel 可以是文件，可以是内存，也可以使用 JDBC，Sink 从 Channel 消费数据，传递给系统中的其他模块，比如 HBase、HDFS、Kafka 等等。

Apache Kafka

官网：<http://kafka.apache.org/>

曾经是一款由 Scala 开发的分布式消息队列产品，现在生态已经扩展了，因为它推出了 Kafka Streaming，所以现在也应该被称作是一个流处理平台了。

Kafka 的队列按照 Topic 划分，每个 Topic 下由多个 Partition 组成，在单个 Partition 中的消息保证是有序的。这种结构下确保了消息是在磁盘顺序写入的，节省了磁盘寻址的时间，所以数据落盘的速度非常快。加之采用了 mmap 的方式，减少了用户态和内核态之间的数据拷贝次数，mmap 是一种将文件内容和内存地址映射的技术，提效十分明显。Kafka 和 Flume 的配合使用，形成了流式处理领域里的经典框架。

Apache Ranger & Sentry

官网：<http://ranger.apache.org/>

官网：<http://sentry.apache.org/>

Ranger 和 Sentry 都是分布式的数据安全工具，这两个产品的功能也基本是一样的，就是去管理大数据计算生态圈产

品的权限，Sentry 是采用插件的形式，将自己集成到 Impala、Hive、HDFS、Solr 等产品上，当用户向这些产品发起请求，产品会先向 Sentry Server 进行校验，Sentry 也可以和 Kerberos 配合使用，从而完成跨平台统一权限管理。而 Ranger 所提供的功能也类似，但是所支持的产品更加多样，包括 HDFS、HBase、Hive、YARN、Storm、Solr、Kafka、Atlas 等，其同样也是采用一个 Ranger Admin 连接多个集成到产品上的 Ranger 插件完成的权限验证过程。

Apache Atlas

官网：<https://atlas.apache.org/>

Apache Atlas 是数据治理体系中比较重要的一个产品，它主要负责元数据的管理，这个元数据就是指用来描述数据的数据，比如数据的类型、名称、属性、作用、生命周期、有效范围、血缘关系等等，在大数据系统中，元数据有着非常大的价值，一个比较成熟的数据系统中一般都会存在着这么一个元数据管理平台，元数据除了能让业务人员更加方便快捷理解我们的数据和业务，也有着帮助我们提升数据质量，消除信息不对称，以及快速定位数据问题等作用，所以如何有效的利用好这些元数据，使这些数据产生更大的价值，也是很多人一直在思考的事情。现在 Atlas

支持的数据源有 Hive、Sqoop、Storm，其导入方式有 HOOK 和 Batch 两种方式，首次使用是 Batch 的同步方式，之后 Atlas 会利用 HOOK 主动获取到数据源的变化，并更新自身数据。

Apache Kylin

官网：<http://kylin.apache.org/>

The screenshot shows the Apache Kylin Cube Designer interface. At the top, there is a header with various status indicators and a timestamp. Below the header, there are tabs for Grid, SQL, JSON(Cube), Notification, and Storage. The main area is titled 'Cube Designer' and shows a flowchart with seven steps: 1. Cube Info (highlighted in blue), 2. Dimensions, 3. Measures, 4. Refresh Setting, 5. Advanced Setting, 6. Configuration Overwrites, and 7. Overview. Under the 'Cube Info' step, there are fields for Model Name (set to 'test_streaming_table_model_desc'), Notification Email List, and Notification Events. A 'Next' button is located at the bottom right of the form.

Kylin是一个为OLAP场景量身定制的分布式数据仓库产品，提供多维分析的功能，并可以和很多BI分析工具无缝对接，比如Tableau、Superset等。Kylin提供了前端平台，使用者可以在该平台上去定义自己的数据维度，Kylin会定时完整分析所需数据的预算算，形成多个Cube，并将之保存在HBase

中，所以部署Kylin的时候需要HBase环境的支持。在数据与计算的时候，对其所在设备的资源消耗也比较大。

Apache Hive & Tez

官网：<https://hive.apache.org/>

官网：<https://tez.apache.org/>

Hive 应该是最有名气的数据仓库工具了吧，他将 HDFS 上的数据组织成关系型数据库的形式，并提供了 HiveSQL 进行结构化查询，使得数据分析人员可以从传统的关系型数据库几乎无缝的过渡到 HDFS 上，但其个别函数和传统 SQL 还是有区别的，并且默认也不支持 update 和 delete 操作。但开发人员可以开发 UDF，为 HiveSQL 扩充属于自己的功能函数。

Hive 本身的计算是基于 MapReduce 的，后来为了应对 SparkSQL 的出现，开发组推出了 Hive on Spark，使得 SQL 的解释、分析、优化还是在 Hive 上，而执行阶段交由 Spark 去完成，从而以达到和 SparkSQL 近似的速度。

Tez 是对 Hive 的另一项优化，为其引入了 DAG 的概念，增加任务并行度从而提升 Hive 的查询速度，但其本质仍旧是

MapReduce，所以提升效果相比 Hive on Spark 来讲并不足够明显。

Apache Presto

官网：<https://prestodb.io/>

Presto 是由 Facebook 公司开发的一款分布式查询引擎，其主要特点是支持了非常多的 Connector，从而实现在一个平台上连接多个数据源，并且可以将这些数据源的内容进行聚合计算，同时 Presto 也支持使用者自行开发新的 Connector。

并且，Presto 的计算过程全程是基于内存的，所以速度也是非常快的，但其实 Presto 也只是针对个别计算场景的性能优化会非常明显，网上有非常详细的分析文章。之前使用该工具是为了将离线数仓和实时数仓的数据进行联合查询，提供给实时数据平台使用。

在使用过程中我觉得有点不好的地方有三点。一是因为 Presto 基于内存计算，所以在资源紧张的情况下经常 Crash 导致任务失败。二是 Presto 任务为串行提交，所以会出现大任务阻塞小任务的情况出现。或许通过调参可以解决该问题吧，但没有再深入调研了。三是没有找到一个比较好

的 Web 平台去查询 Presto，网上有 Hue 通过 PostgreSQL 去链接 Presto 的方案，觉得有点麻烦，看上去比较成熟的 Airpal 平台也已不再更新了。最后使用了 yanagishima，基本功能可以满足，但该平台没有用户管理功能，没法控制权限。

Apache Parquet & Orc

官网：<https://parquet.apache.org/>

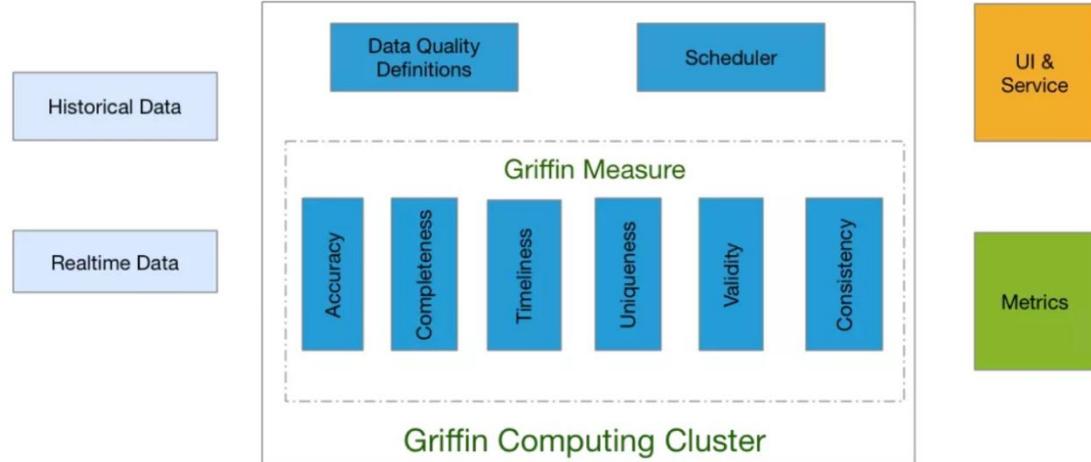
官网：<https://orc.apache.org/>

Parquet 和 ORC 是两种比较应用比较多的列式存储格式，列式存储不同于传统关系型数据库中行式存储的模式，这种主要的差别可能由于联机事务处理（OLTP）和联机分析处理（OLAP）的需求场景不同所造成的。在 OLTP 场景多是需要存储系统能满足快速的 CRUD，这种操作对象都是以行为单位的。而在 OLAP 场景下，主要的特征是数据量巨大，而对实时性的要求并不高。而列式存储正式满足了这一需求特征。因为当数据以列的方式存储，在查询的时候引擎所读取的数据量将会更小，而且同一列的数据往往内容类似，更加便于进行数据压缩，但列式存储不适于更新和删除频繁的场景。

Parquet 和 Orc 同为列式存储，但他们的存储格式并不相同，这种差异造成了两者在存储不同类型的数据时所出现的性能差异，从网上的一些文章看，Orc 的性能要比 Parquet 好一点，但是 Impala 是不支持 Orc 的，并且诸如 Delta Lake 这种数据湖产品，也是基于 Parquet 去做的。所以在选择采用哪种列式存储格式时，还是要根据自身的业务特点来决定。

Apache Griffin

官网：<http://griffin.apache.org/>

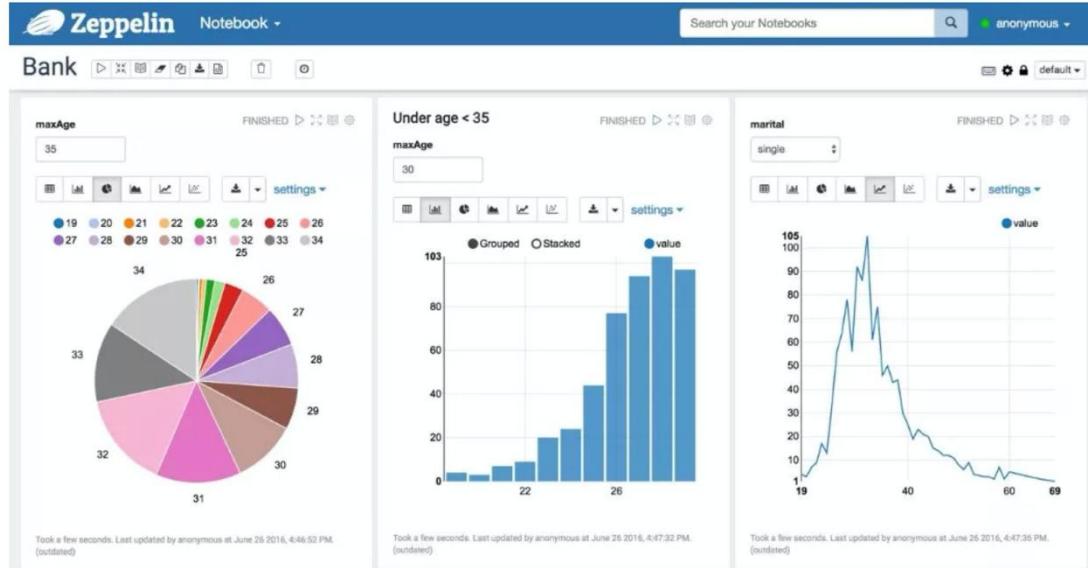


数据质量管理是数据系统中不可或缺的一环，初期的时候我们往往在 ETL 的各个阶段，加入一些简单的脚本来对生成的数据进行检查，而 Apache Griffin 也是一款这样的产品，它是由 eBay 开发的一个数据质量监控平台，后上升为 Apache 顶级项目。它提供了数据校验和报警的功能，也支

持一些参数的可视化展现，相关的配置步骤都可以在 Griffin 的页面上完成。除了能完成一些最基本最简单的诸如是否存在异常值的数据检查，也能完成一些诸如最值、中值的数据统计需求等等，并且提供了专业的图表报告。

Apache Zeppelin

官网：<http://zeppelin.apache.org/>



Zeppelin 是一款非常方便的在线笔记本，使用体验有点像 Python 的 Jupyter NoteBook，可以从图中看到使用者可以在线执行，并绘制简单的图表。并且 Zeppelin 有了用户的概念，使得多人协同工作更加方便。Zeppelin 支持了非常多的数据源，通过该平台，可以调用 Hive、Cassandra、R、Kylin、Flink、Spark、ElasticSearch、HBase、Python、Shell 等等。

我在使用时出现了 Spark 连接不稳的情况，需要使用者反复登录才可以。但总之我还是非常喜欢这款工具的。

Apache Superset

官网：<http://superset.apache.org/>

Superset 是一款开源的可视化工具，使用该工具可以方便快速的创建数据 Dashboard，同类型的产品还有 Redash、Metabase，但调研过后个人还是更喜欢 Superset 一些。不过因为同期引入了 Tableau，Superset 并没有在实际项目中使用。

Tableau

官网：<https://www.tableau.com/>

和介绍的其它软件不同，Tableau 是一款商用软件，根据购买的账号数量按年付费，之所以这里提到它，也是因为 Tableau 在 BI 领域内的名气着实有点高。Tableau 分为 Server 端和本地客户端，使用者通过在客户端上的拖拽，即可快速生成一个数据 Dashboard，使得 Dashboard 的开发工作从开发侧下放到了需求方。

并且，Tableau 也提供了完备的用户管理功能，还支持了非常多的数据源。商业软件和开源软件比起来，无论功能完备性上还是使用体验上，的确都有明显的提升。我觉得唯一的难度可能就是如何把这个开发维护的工作在需求方落地吧，毕竟它还是有一些学习成本的。

TPCx-BB

官网：<http://www.tpc.org/>

TPC 全称是事务处理性能委员会，是由数十家公司组成的非盈利性组织，负责订制各个行业的基准测试规范，阿里巴巴的 MaxCompute 和 OceanBase 都参加过该项测试，并取得了非常好的成绩。TPCx-BB 是一个大数据基准测试工具，该工具模拟了一个网上零售的场景，首先工具会先向被测系统中插入预定好的表和数据，然后经过一系列的 SQL 操作，来对大数据集群的性能进行评估。

TPC 针对不同的被测场景，提供了很多现成的工具，大家可以在[这里下载](#)使用。

招聘

阿里巴巴高德地图智能技术中心长期招聘机器学习算法、
C++、Java 资深工程师/技术专家/高级专家，职位地点：
北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主
题为：姓名-应聘团队-应聘方向。

高德地图首席科学家任小枫 QA 答疑汇总 |

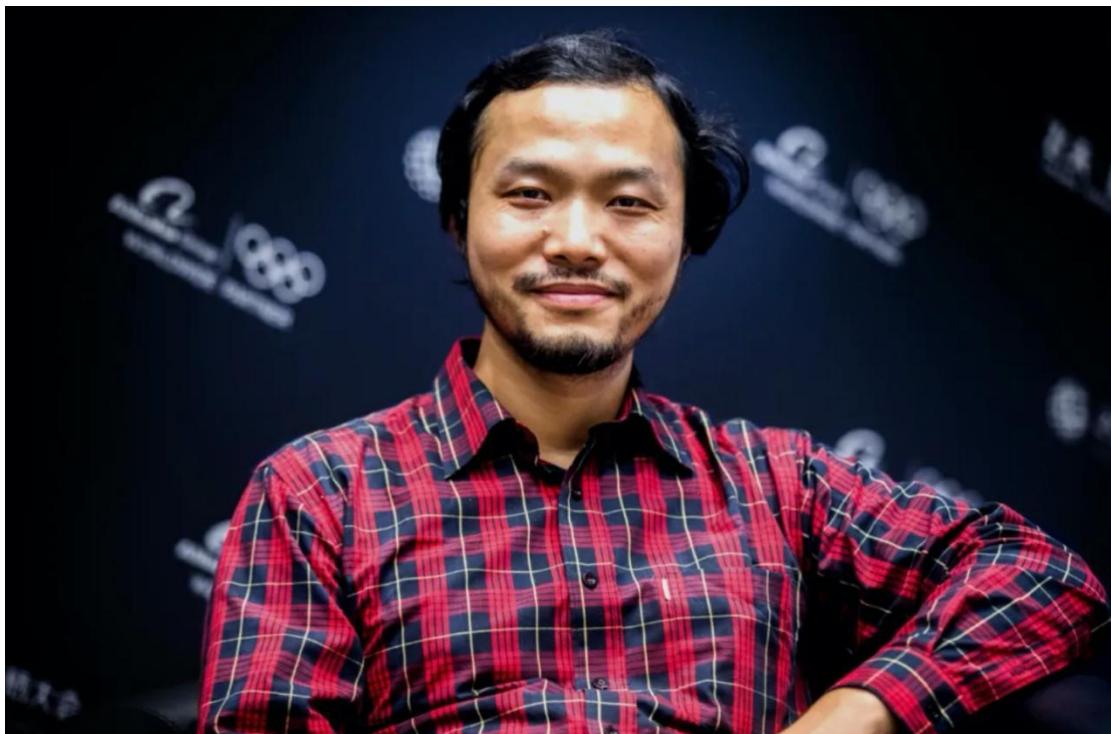
视觉+地图技术有哪些新玩法?

高德技术整理

2020 年 4 月份的时候，阿里巴巴高德地图首席科学家任小枫在#大咖学长云对话#的在线直播活动上就计算机视觉相关技术发展以及在地图出行领域的应用与大家做技术交流，直播间互动火爆，尤其在 QA 环节，学弟学妹们纷纷就感兴趣的视觉应用、AR 导航、定位技术、5G、职业发展等话题提问，任小枫做了精彩回答。我们整理了问答内容，分享给大家。

视频回放地址：

<https://vku.youku.com/live/ilproom?id=8064786>



任小枫博士，现任阿里巴巴高德地图首席科学家，研究员，主要负责视觉技术在地图和出行领域的应用和创新。加入阿里巴巴前，他在 2013 到 2017 年间供职于亚马逊，是亚马逊的资深主任科学家和 AMAZON GO 的算法负责人。浙江大学本科毕业，加州大学伯克利分校博士，华盛顿大学计算机系客座教授，CVPR/ICCV/AAAI 等会议领域主席，IEEE PAMI 副主编。

视觉技术发展及应用

提问：计算机视觉在高精度地图构建中的应用有哪些？

任小枫：视觉算法对于高精度地图构建是核心的技术，主要应用在资料对齐和精度保证、识别和地图数据自动化生成、视觉定位和高精地图更新等。

提问：您觉得现有的基础学科研究水平与硬件水平能否保证视觉技术的快速发展？视觉技术发展在近期会不会遇到较难突破的瓶颈？

任小枫：经过了前几年深度学习技术在视觉各个领域的快速发展，一定程度上说，深度学习和视觉的基础技术现在都遇到了瓶颈。或者说，没有开始的时候发展的那么快，有很多难题需要解决，也可能需要创造新的技术。对于应用而言，我觉得基础技术和硬件水平目前大致是够用的，更重要的是如何把技术用好，有针对性的去突破技术瓶颈。

提问：单目标跟踪 SOT（给定模版跟踪单个目标，类别无关/可跨域）近两年的进展非常显著，具有解决快速跟踪的潜质，想请问目前有没有在地图业务这边比如视觉定位（VO 中跟踪路标）/AR 导航（短时跟踪）中应用的前景？

如果有的话，请问需要解决什么样的需求问题（鲁棒/速度等）？

任小枫：跟踪是一个视觉基础技术，在很多场景都有应用。对于导航和出行，确实在 AR 导航、定位上能起到核心作用，减少识别（检测）的计算需求，并增加鲁棒性和平滑性。但是在很多实际应用中，跟踪的使用和需求和学术界单目标跟踪的设置会有所不同。

提问：视觉特征是否能结合语义给地图的导航出行服务带来更好的体验呢？

任小枫：视觉可以提供高精度的定位，也可以提供场景的语义理解，肯定可以带来导航和出行更好的体验。但是具体的产品体验和技术实现还需要进一步的探索和积累。

提问：计算机视觉下一步的重难点是哪个方向？未来的前景如何？

任小枫：计算机视觉是一种通用的感知手段，信息量很大，可以用于多种感知任务，可以远距离观测，应用的前景是很广阔和美好的。下一步的难点，除了基础技术需要进步和突破外。还有：如何找到视觉能发挥核心作用的应用场景，如何根据实际问题综合各类算法设计整体方案，如何

较好的解决计算资源的问题，如何结合其他传感器和先验知识等问题。

AR 导航

提问： AR 导航是实时图像计算的吗？设备算力可以打标吗？

任小枫： AR 导航是实时图像计算，在低算力的条件下实现导航和辅助驾驶功能。我们也尽可能的进行“预算算”，事先计算好环境中的一些元素，来配合实时计算。

提问： AR 导航最后通过什么来展示内容？显示屏还是 HUD？

任小枫： AR 导航有多种产品形态：中控屏、HUD、后视镜、仪表盘，这些都是正在使用/潜在使用的展示方式。

提问： 有一个非技术性的问题，AR 导航会不会过度吸引驾驶员的注意力，导致他/她忽略车辆两侧的交通？

任小枫： 这是一个产品设计的好问题，也是我们一直在打磨和寻求平衡的问题。一个设计的好的 AR 导航产品，会考虑到不过多吸引注意力。

提问：安全辅助驾驶会有疲劳驾驶检测吗？

任小枫：高德的 AR 导航目前只有朝外的单目相机，没有支持疲劳驾驶检测。对车内的监控，包括疲劳检测，是视觉技术在安全辅助驾驶的一个重要应用。

定位技术

提问：室内定位现在主流实现技术有哪些？基于声信号的室内导航前景好吗？

任小枫：室内定位有多种基于传感器的技术，包括 WiFi, Bluetooth, RFID, Ultra-Wideband，也包括声信号。我觉得室内定位的发展，如果需要部署传感器，很大程度上不是取决于技术和定位精度，而是是否有好的应用。WiFi 定位的普及是因为室内网络需要 WiFi。iPhone 11 装了 UWB 芯片可以近距离文件传输。

提问：GPS 定位那么大的差距是什么原因导致的？因为多路径效应吗？

任小枫：GPS 定位不准有多个原因，主要是在“城市峡谷”（高楼林立）的场景。多路径效应是其中最重要的因素，

因为环境的折射（特别是像玻璃这样的高反光材料），导致 GPS 位置计算不准。其他方面还有因为楼宇/高架桥的遮挡导致能观察到的卫星数降低，空气（特别是带电离子和水蒸气）的干扰，等多种原因。

提问：高德如何解决 GPS 漂移的问题？

任小枫：这是一个复杂的问题。基于手机传感器，我们结合实际的驾驶和步行场景做了很多优化，包括 GPS 置信度分析，和 IMU 结合，和路网结合等。视觉定位是我们在开拓的解决定位不准的一个新方向。

地图基础技术

提问：目前高德地图图层有哪些？是语义级高精度地图吗？

任小枫：高德地图有多种地图数据形态，从标准地图（高德 App 上看到的），到车道级地图，到高精地图。精度不同，对应的应用不同。多种地图中都有语义信息，但是语义信息的内容和精度会有不同。

提问：深度相机和普通的相机有什么区别？

任小枫: 普通相机获取的信息是二维 RGB 图像，没有三维信息。深度相机在每个像素上，除了 RGB 颜色之外，也同时获取深度（距离）信息，一般是利用主动模式（time-of-flight, structured light 等）。现在很多主流手机上都已经配备了深度相机。

提问: 高德地图对道路信息是怎么采集的，道路有变化地图会实时更新么？

任小枫: 高德地图道路信息有多个来源，主要是依靠低成本的车载视频资料。道路相关信息是在随时变化的，我们会不断的采集最新资料并制作更新地图数据，及时上线应用。

提问: 室内三维空间（比如多层的商业大楼）地图绘制的难点有哪些？

任小枫: 室内三维地图绘制最大的难点在于数据采集。三维重建的方法需要有多个角度的图像。基于深度相机的移动建模方法精度上不一定能满足需求。

新人职业成长

提问：从视觉和图像领域的学术研究领域到公司商业计算机视觉应用技术开发需要补充哪些知识？

任小枫：我觉得主要要考虑的不是补充具体的知识，而是要注意培养自己的各方面的能力：（1）对实际问题的分析和解决的能力；（2）动手能力；（3）快速学习和拓展知识的能力。

提问：从事计算机视觉领域该如何制定职业规划？

任小枫：和其他行业和技术方向的职业规划没有本质的区别，要结合自身的长/短处和兴趣，找到自己合适的工作方向，逐步提高技术深度，广度，高度，综合能力，一步步做出实际结果发展职业。

提问：请问现在从事视觉领域工作是否一定要具备深度学习的技能？

任小枫：计算机视觉现在大量的使用深度学习技术，深度学习的知识和技术我觉得是必须的。有一些和几何相关的子领域，比如三维重建、SLAM/VIO，深度学习应用的还不多，但是（1）后续预计会有更多的深度学习应用；（2）

从提高技术广度和视野出发，也需要一定程度上了解深度学习。

业界热点及其他

提问：自动驾驶会用到 5G 技术吗？

任小枫：目前看来，5G 技术会在自动驾驶上有多种应用，但对于 L4/L5 全自动驾驶，我觉得 5G 并不能从根本上解决自动驾驶安全性（和舒适性）的难题。

提问：跟踪和定位中的计算端和云如何配合？

任小枫：大体上来说，实时性要求高的，和传感器结合密切的，会在端上完成；和地图结合密切的，需要用到大量参考数据的，会在云上完成。

提问：谷歌地图有一个街景地图的模块用到了许多图像识别的技术，街景地图怎么拼成的？以及街景发展趋势是怎样的？

任小枫: 谷歌地图的街景地图主要来自于谷歌自己的街景采集车，车上载有高质量的相机和组合惯导等传感器。街景地图主要是一个拼接的过程。街景地图很有意思，但还没有对导航和出行的体验带来根本的变化。谷歌最近的 AR 步行导航（这个和高德的车载 AR 导航不同）是基于街景地图的一个新应用。

提问：可穿戴设备（类似眼镜、智慧助手等）在视觉技术上如何更好的落地以及产品化？

任小枫: 硬件（AR 展示，算力）和体验是可穿戴设备要真正落地和普及的主要问题。Google Glass 作为一个超前的产品，在硬件上受限制太大。目前 AR 眼镜的应用主要在企业场景。我个人觉得可穿戴设备作为个人助手（包括导航，信息展示等）的应用前景是很好的，但现在硬件条件可能还不成熟。

招聘

高德地图视觉技术中心火热招聘中：计算机视觉、图像识别、点云、三维、AR、视频等算法类职位，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

汽车工程篇

高德车载导航自研图片格式的探索和实践

作者：桂凯

背景

随着近年来车内多媒体设备从无屏向有屏的发展，市场上出现了各种形状、尺寸和分辨率的车机屏幕，其丰富程度远远超过 Android 适配的手机屏幕。

高德车载导航过去采用的多套 UI 图片资源，通过拉伸、缩小来适应各种车机屏幕以便减少内部 UI 资源开发和管理成本的方式受到越来越大的挑战：软件包的 Size 不断增加，对安装空间和用户流量提出更高要求、多套 UI 资源的维护成本越来越高、拉伸或者缩小导致适配效果上存在各种失真的情况。

本文小结了高德技术团队在车载导航自研图片格式解决方案上的探索和实践，希望对大家有所启发。

本地化方案和动态化方案的比较

行业上的解决方案基本上分成两大类：

本地化方案：UI 资源存在于软件包中，然后从图片格式入手，解决使用过程中的性能和成本问题。

	举例	特征描述
图片压缩	PNG、WEBP	该方案是在性能和大小上零和博弈，压缩比带来的解码性能问题会加大车载设备的性能要求
	.9.png	对图片质量要求高，应用中占比小，主要设计目的是解决图片的拉升变形问题，同时能平衡图片的大小和性能
图片矢量化	SVG	通过矢量方式表示图形，能有效解决大屏模式下显示效果问题，需要设计支撑。扁平化设计风格对该方案友好

动态化方案：通过在线识别目标机器的硬件配置，动态下载合适的 UI 资源，通过在线的方式动态生成。

	举例	特征描述
动态图片	常见于游戏和Web应用	通过启动时或者运行时下载资源来显示，已达到最优尺寸资源显示，需要UE交互支持该模式
动态包	苹果手机应用	开发将IPA上传到应用商店，由App Store在不同目标设备下载对应资源模式的包，在安装时已经感知完成，依赖应用商店架构支持

从车机应用的角度思考，这两种方案的优缺点如下：

方案	优点	缺点
本地化	不依赖于网络和生态，离线情况下也能得到相应的显示效果	视觉效果的设计会对资源大小和显示性能产生较大的影响
动态化	对不同设备友好性高、具备在线主题的扩展能力	对网络和生态支持要求较高

从以上可以看出，当前阶段，业内的车载生态体系的建设并不健全，采用本地化方案更具有现实意义。在大屏时代，随着硬件性能的逐步提升，矢量化的图片方案会成为未来应用的趋势。于是，我们决定在图片矢量化方向上开展建设，以确定适合高德车载的图片适配方案。

矢量化方案的探索与实践

以下是常用的矢量图片方案的能力支持情况：

矢量化方案	颜色通道	Path图形	渐变	阴影
iconfont	单色	支持	否	否
VectorDrawabe	多色	支持	支持	否
SVG	多色	支持	支持	支持
Shape+LayerList	多色	只支持简单的Path (*注)	特定角度的渐变	只能渐变伪装

注：Shape 支持简单的 Path 指的是：矩形、圆角矩形、椭圆、线。

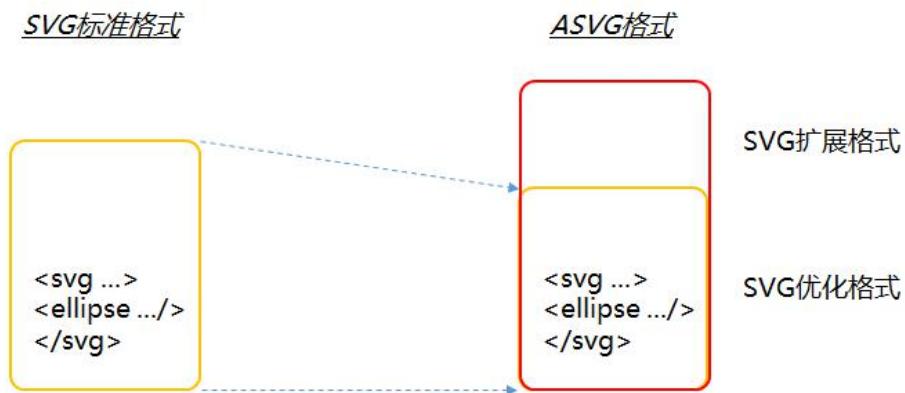
通过除动画外的常用图形设计元素进行对比，可以看出除 SVG 外，其它表达方式在当下个性化诉求下，存在一定的不满足性。因此，SVG 应该是高德车机在矢量图上的最优选择。

SVG 在车载业务上的适用性分析

车机硬件属于嵌入式硬件，车机上的导航应用，同样要遵循一般的规则，即用最少的资源（磁盘、内存、CPU）取得最大的运行效果。结合高德业务的需求，通过对 SVG 格式的分析，我们发现了 SVG 格式存在如下的不足：

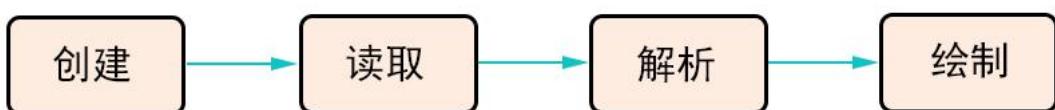
- SVG 文件类型为文本文件，读写效率存在一定的提升空间；
- SVG 数据中存在大量非必要数据，对文件大小和读写效率存在影响；
- 高德业务中，存在大量差异性较小的资源，需要提供更优的解决方案；
- UED 在提供的设计资源中可能会提供 .9 图片，需要考虑合适的解决方案；
- 不支持颜色描述主题化、圆角信息 ID 化等为支持高德动态 HMI 所需的需求；

基于以上分析，为了满足业务的需求，我们考虑扩展标准的 SVG 格式创建高德自用的图片格式，格式名称定义为 ASVG (Amapauto SVG)。



ASVG 在车载业务上的实践

ASVG 在车机导航中，需要经历如下四个环节。我们主要从创建、解析和绘制环节对 ASVG 的使用进行了优化。



1. 针对创建环节，从 ASVG 格式的角度进行优化：

• 数据结构优化

在 SVG 的设计意图本身是更加倾向于让使用者更易用，所以在表达矢量化过程中存在大量冗余的意图数据，这导致

使用过程中存在较高的解析成本。在嵌入式系统上，我们需要充分利用格式中的每一个字节，有效提高读取效率并降低资源大小。

对 SVG 进行解析并将对应的节点和属性变成一个特定的中间结构，并扁平化存放，除 magic 等数据外，其它数据可直接用于图形运算，去掉非必要数据，从而达到压缩和提高解析效率的目的。如下表格：

	层级关系表达代码	用于理解的符号	用于描述图片特征的说明
原始格式	存在	存在	存在
扁平化	不存在	存在	存在
抽象数据结构	不存在	不存在	不存在

完成优化后，整个运算过程中不再进行字符串运算，在连续内存中读取顺序变量 Buffer 并进行赋值，解码时间降到微秒级别，资源大小平均缩减 60% 左右：

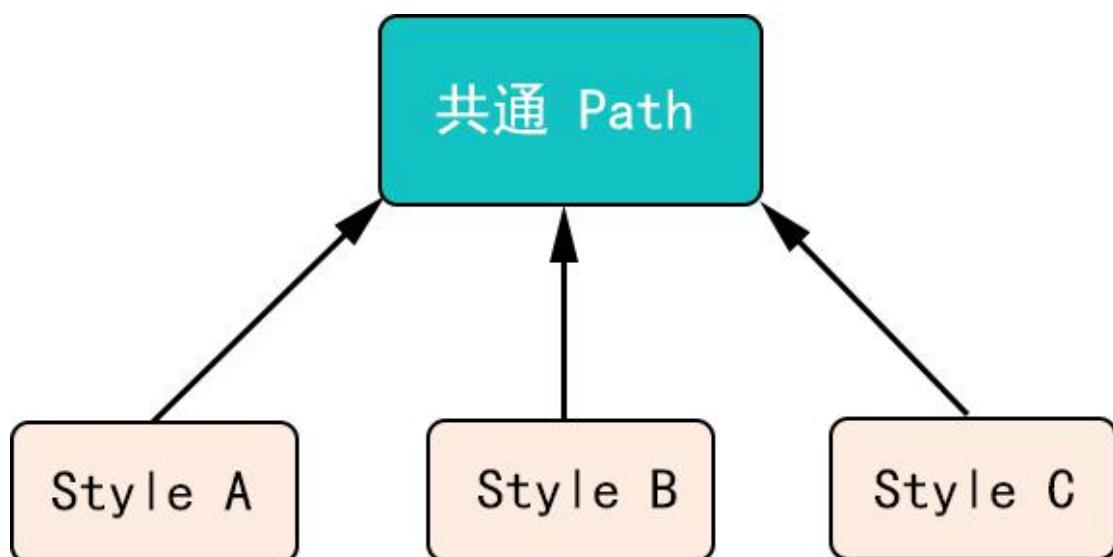
PNG (29*29)	SVG	ASVG
681 Byte	506 Byte	345 Byte

• 结合业务去冗余

在车机导航应用中，同一个控件对象在不同的场景下，UED 会制作不同的图片资源。如下图，设计师根据昼夜不同的色彩饱和度和阴影效果来达到车标 Icon 的设计效果。

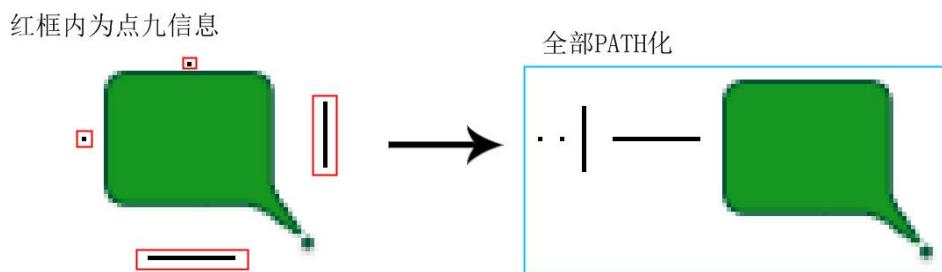


这种情况下，形状描述相关信息基本一致，通过设计中间结构在图形状态及主题上进行去冗余。



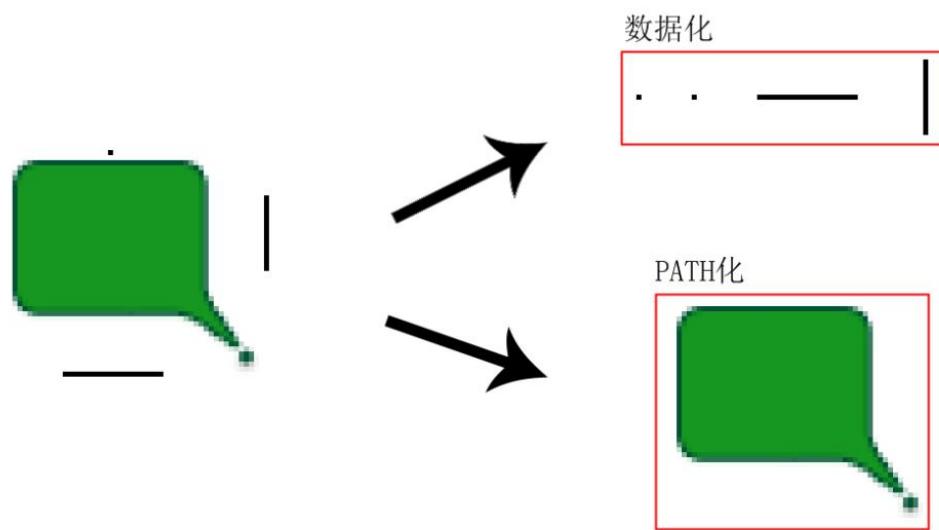
- 点 9 图片 ASVG 化

点九图片是 PNG 栅格图片中一类特殊的图片。将点九图片 ASVG 化，面临点九图片信息管理的问题。



对策方案：

- a. 从数据结构上，将点九信息从点九图片中拆分出来。点九信息直接存入 ASVG 文件中，将点九信息直接数据化。



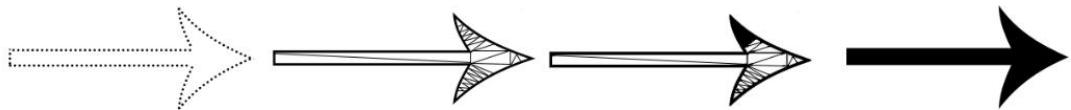
- b. 从规则上，制作点九图片时，保证点九信息的点九长度只有一个像素，点九信息周围一个像素点也可以做点九信息点。保证在矢量图片缩放过程中，解决进退位计算带来的点九长度位置误差。

2. 针对解析和绘制环节，选择合适的矢量方案：

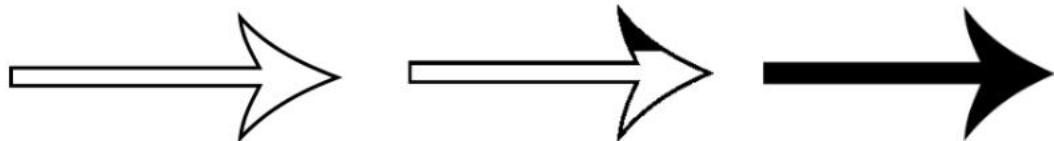
- 矢量图形的解析方案

矢量图形的解析就是将矢量图形中的 Path 部分翻译成可由 GPU 绘制形状。方案主要有两种：

方案 A. 将 Path 通过分格化（tessellation）分格成多个小的凸多边形。最后，利用标准渲染 API 直接完成绘制。

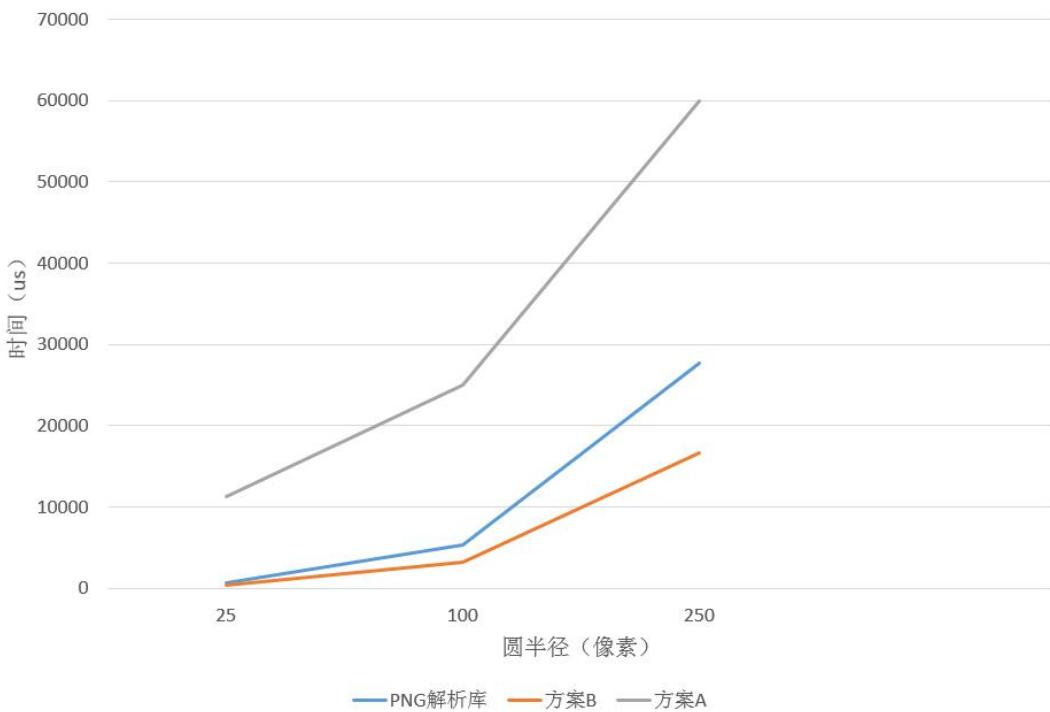


方案 B. 将 Path 通过路径解析利用扫描线的方式，将 Path 转成一张 bitmap。最后，利用标准渲染 API 直接完成绘制。



- 矢量图形解析方案的效能对比

下图是各矢量图形绘制方案的解析效率对比。在同等试验条件下，方案 B 的加载和解析效率要高很多。



• 矢量图形方案的绘制效果对比

在显示效果上，车载终端的设备差异较大，部分设备不支持抗锯齿能力。使用方案 A 绘制出来的图形在处理斜多边形部分会出现锯齿，而使用方案 B 绘制出来的图形却平滑很多：



综合考虑，我们采用方案 B 作为 ASVG 矢量图的最终方案。

小结

通过 ASVG 的使用，高德车载导航业务取得了较大收益，在界面显示效果、图片加载效率、资源维护效率等方面有了很大提升，同时 APK 包大小大幅下降。面对车载行业个性化需求及 5G 时代的到来，“显示”作为用户在车载交互体验中重要的一环，我们将在为用户提供感官上更加真实的体验方面不断创新，后续也会在该方案基础上融入更多能力和提供相关工具，在动态化、主题定制等场景扩大使用范围。

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德车载导航的差分更新优化实践

作者：迷珑

导读

随着车载设备联网化，越来越多的车载设备从离线走到了线上。高德车载导航也早已从过去的离线安装包更新演进到了在线迭代更新。但原车载设备的 Android 硬件配置远低于手机，主要表现在处理器主频低、内存和存储空间有限，导致车载导航在车机上会出现无法下载新版本数据包、更新过程耗时长导致卡顿的情况，对导航应用的性能提出了要求。

为提高用户体验，高德技术团队立项解决了该问题。本文小结了高德车载导航在版本自更新演进过程中二进制差分解决方案的性能优化实践。

差分更新方案比较

对于应用程序的版本更新迭代，除了分发全量的安装包，还有一种更低成本的方式是分发增量包，即通过下发前后两个版本的差异部分(这个过程下面简称 Diff)，然后在客户

端对原版本进行补丁更新(这个过程下面简称 Patch)。因此也叫差分更新。

业内比较流行的差分方案主要有：bsdiff、Xdelta3 和 Courgette。最后一个方案 Courgette 来自于谷歌，主要解决的是可执行文件的差分，而导航更新资源不仅包含可执行文件，还包含了图片等各种资源文件。所以，我们主要对比 bsdiff 和 Xdelta3 方案。

bsdiff 和 Xdelta3 方案比较

下面是我们对选取的几个文件做的 bsdiff 和 Xdelta3 差分性能对比：

文件	bsdiff	bsdiff patch	bsdiff patch	Xdelta3	Xdelta3 patch	Xdelta3 patch
	压缩比	耗时	内存	压缩比	耗时	内存
文件1	14.2	334ms	23MB	4.83	77ms	71MB
文件2	7.61	7.1s	42MB	5.33	376ms	79MB
文件3	27.14	5.15s	11MB	9.5	12.9ms	66MB

bsdiff 的优势是压缩比高，生成的差分文件非常小，但 Patch 过程耗时。而 Xdelta3 的优势是 Patch 过程耗时极短，但内存消耗非常大。

相比高德车载导航自身运行内存开销不足 100MB 的情况，Xdelta3 的 Patch 内存消耗无法接受。因此我们选用了 bsdiff 作为自更新方案。

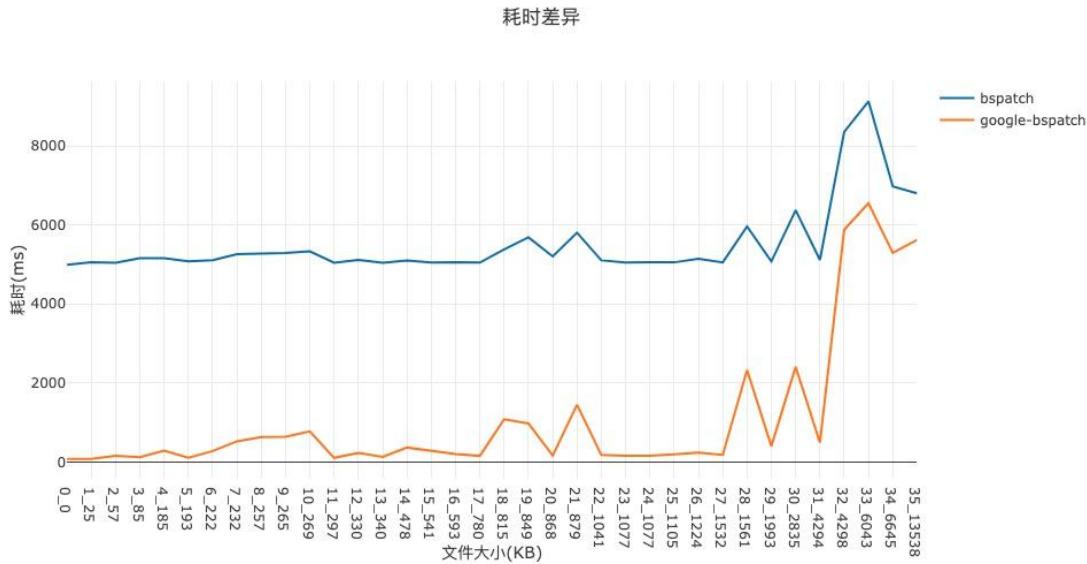
原生 bsdiff 方案缺陷与改进

原生 bsdiff 方案使得压缩比问题得到解决，但在车载导航自更新中还存在下面两个缺陷：

- 内存消耗大，整个过程会占用 10~35MB 左右的内存。
- 耗时长，整个包更新时间在 3 分钟左右。

在对 bsdiff 的优化探索中我们发现 Chromium 开源项目中存在一份基于 bsdiff 的优化版本。该版本将 bsdiff 的默认 sufsort 算法替换成 divsufsort 算法，在 Patch 时间上有了一较大的提升。





使用 Chromium 版本的 bsdiff，高德车载导航的自更新性能如下：

- 内存消耗在 10~20MB。
- 整个 Patch 过程耗时仍然长达 25 秒左右。

耗时依然很长。

由于 Patch 过程是一个 CPU 密集型的操作，且车载设备 CPU 的性能普遍不足，这意味着在整个升级过程中用户能明显感受到导航的操作卡顿。

同时，更新时间越长意味着遭遇断电的可能性也越大。

基于以上分析，我们决定对Chromium版本的bsdiff进行CPU和内存上的性能优化。

bsdiff 在车载自更新业务中的性能优化实践

在车载自更新业务上，我们设定的目标是整体更新时间小于 6 秒，且内存开销小于 2MB。整个优化的过程就是围绕时间和空间的取舍。

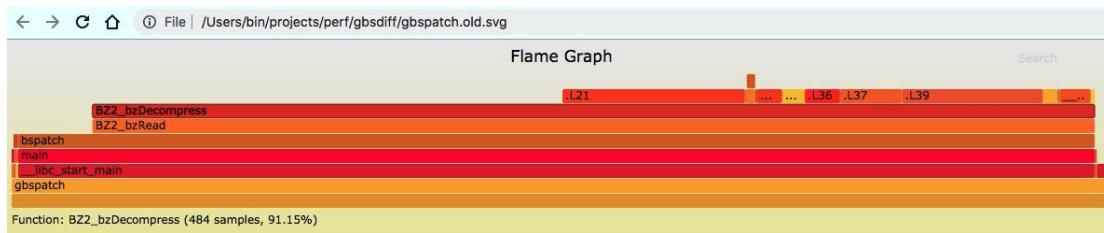
内存优化方案

经过对 bsdiff 源码的分析，其 Patch 内存主要开销来自文件内容在内存中的读写暂存和 Bzip2 的解压开销。通过调整 Bzip2 参数可以降低部分内存，但无法达到期望。而文件读写的内存占用主要来自于其在内存中的暂存。

基于 bsdiff 差分 Patch 包的文件格式，我们增加了滑动窗口缓冲区的 Patch 特性，使其在文件的流式处理上能够有更好的内存消耗可控性。每次读取和写入指定的滑动窗口大小数据，使数据即来即走。

算法优化方案

经过上述的优化后，Patch 过程的主要性能瓶颈在于 Bzip2 的解压算法中，即使调整 Bzip2 参数也无法减少本身的计算量。



bsdiff 差分算法的一个特性就是差分出的 Patch 数据包含了大量连续的 01 冗余数据，而 Bzip2 算法的优点就是对这类数据可以做到高度的压缩，这也是 bsdiff 压缩比高的原因。不过现在是目前的瓶颈。

此外，我们会制作软件整体的压缩差分包(即生成 tar.bz2 或 zip 格式文件)，也就是说针对每个 Bzip2 压缩后的差分文件还要再经过一次压缩归档。这也意味着在客户段要进行两次的解压。

替换压缩算法

类似的冗余压缩算法有 RLE(Run-length encoding)，这个算法也是 Bzip2 算法的第一步。简单来说 RLE 算法就是针对连

续多个冗余字节去掉其冗余字节，仅保留冗余的长度信息。这个算法相对更简单。

因此，我们将 Bzip2 压缩算法替换成 RLE 算法，实际结果发现生成的 Patch 文件很大，压缩比很低。但是可以通过再次压缩归档制作一次差分包，就可以达到和 Bzip2 几乎相同的压缩比效果。唯一的不足就是在客户端解压后会占用多一些磁盘空间，而这个代价相对廉价多了。

优化性能对比

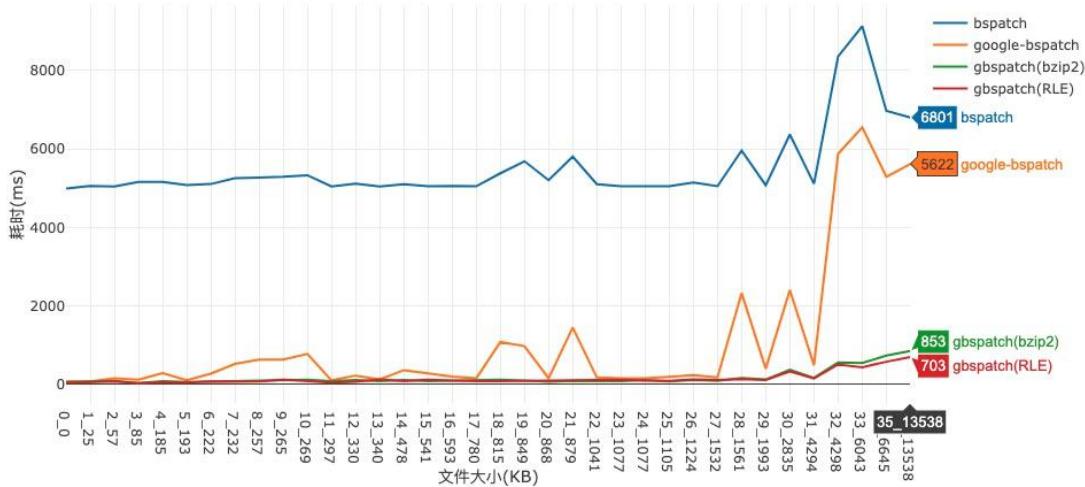
经过上述整体优化后，性能对比如下：



经过内存优化后的方案空间复杂度将为了 O(1)。

汽车工程篇-高德车载导航的差分更新优化实践

耗时差异



上面的耗时差异在 ARM 车机会更明显：

```
Performance counter stats for './gbspatch_old ./libGNaviDice.so.1280 ./libGNaviDice.so.new_old ./libGNaviDice.so.patch_old':  
    781.544011 task-clock      # 0.939 CPUs utilized  
        263 context-switches   # 0.000 M/sec  
          0 CPU-migrations    # 0.000 M/sec  
        835 page-faults       # 0.001 M/sec  
 771178225 cycles           # 0.987 GHz  
        0 stalled-cycles-frontend # 0.00% frontend cycles idle  
        0 stalled-cycles-backend  # 0.00% backend cycles idle  
        0 instructions          # 0.00 insns per cycle  
        0 branches              # 0.000 M/sec  
        0 branch-misses         # 0.00% of all branches  
  
 0.832428333 seconds time elapsed  
  
NaviDice.so.new ./libGNaviDice.so.patch  
<  
Performance counter stats for './gbspatch ./libGNaviDice.so.1280 ./libGNaviDice.so.new ./libGNaviDice.so.patch':  
    244.430000 task-clock      # 0.939 CPUs utilized  
        21 context-switches   # 0.000 M/sec  
          0 CPU-migrations    # 0.000 M/sec  
        611 page-faults       # 0.002 M/sec  
 242975831 cycles           # 0.994 GHz  
        0 stalled-cycles-frontend # 0.00% frontend cycles idle  
        0 stalled-cycles-backend  # 0.00% backend cycles idle  
        0 instructions          # 0.00 insns per cycle  
        0 branches              # 0.000 M/sec  
        0 branch-misses         # 0.00% of all branches  
  
 0.260288333 seconds time elapsed  
root@topeet_6dq:/data/perf #
```

最终优化收益：内存消耗控制在 2MB 以内，整体 Patch 更新耗时 3~5 秒。

小结

通过对 bsdiff 的优化，高德车载导航在自更新性能上取得了较大收益。大幅缩短了用户下载和更新时间，降低了对 ARM 车机的硬件资源要求。为推动车载导航 OTA 更新提供了技术基础，对未来高德车载导航在分发新功能、新业务上铺平了道路。

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

远程调试在 Linux 车机中的应用

作者：仟哩

导读

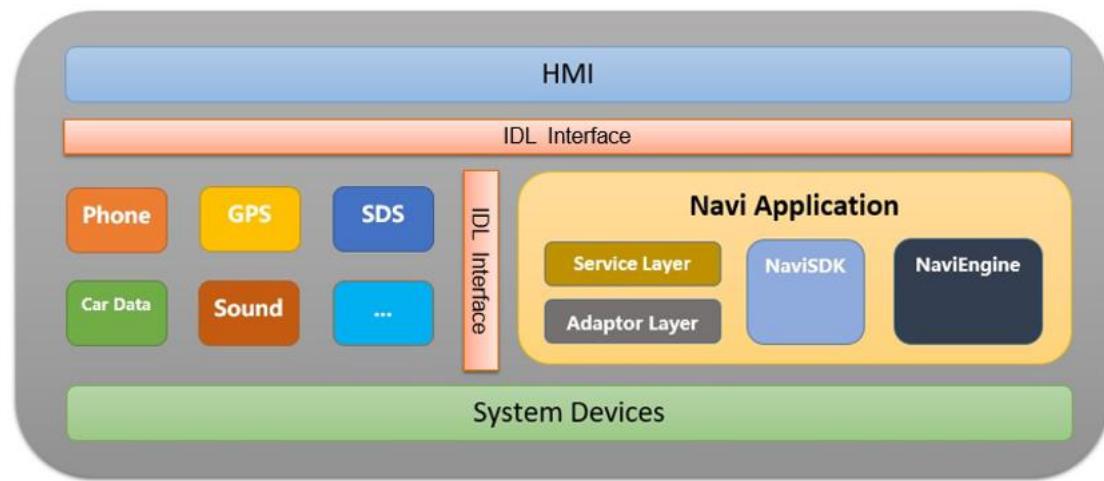
在软件开发过程中，调试是必不可少的环节，嵌入式操作系统的调试与桌面操作系统的调试相比有很大差别，嵌入式系统的可视化调试能力比桌面操作系统要弱一点。对于导航这种业务场景比较复杂的程序开发，可视化调试环境能让我们业务场景开发事半功倍，也能快速定位导航业务与车机中其他模块交互出现的问题，提高开发过程中的调试效率。

远程调试是真机调试中最便捷的一种，开发者只需借用在 PC 端强大的调试器就能完成业务场景的调试。

背景

Thrift 是一种接口描述语言和二进制通讯协议，它被用来定义和创建跨语言的服务，是一种 RPC（远程过程调用）通信框架，由 Facebook 为“大规模跨语言服务开发”。在车机系统中，各模块之间也可以使用 Thrift 通信框架进行通信。导航作为一个单独的为进程提供服务的模块，只提供导航

相关的业务以及地图渲染的能力，导航的 HMI 界面是车机系统中统一的操作界面，系统 HMI 界面与导航之间的交互接口则是通过已经定义好的接口描述语言（IDL），使用自动化工具生成本地可调用的接口，然后使用 Thrift 框架传输完成系统 HMI 与导航之间的通信。



调试手段

为了开发过程中调试方便，我们在 PC 上做了一套模拟器，能在 PC 上进行地图渲染。还实现了一套在 PC 上使用的系统 HMI 模拟命令发送工具，模拟工具是作为客户端连接导航提供的服务，这样能在 PC 端模拟发送命令，帮助导航简单业务的开发，但这种方式存在着以下弊端：

- 模拟命令工具，只能模拟简单的业务场景，有多个交互的场景无法模拟。

- 无法操作地图 HMI，也看不到 HMI 界面显示以及过程中的反馈。
- 无法滚动地图，后面接了 Win32 上面的鼠标事件，能用鼠标实现滚动，但这种方式与车机中滚动流程不一致。
- PC 端无法使用车机中的设备，如导航过程中没有导航音，无法使用 USB 等。
- PC 端拿不到车机中的数据，比如车身数据、GPS 信号等。

调试方案优化

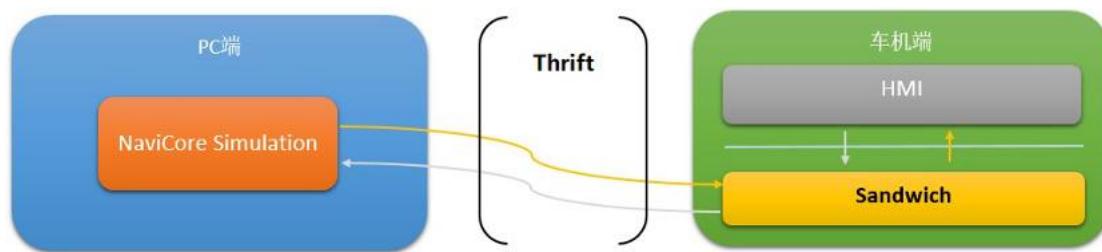
针对当前调试手段存在的以上问题。我们对调试方案进行了优化，我们可以借助车机中系统 HMI 来与导航进行交互。实现了使用车机环境中的信号对 PC 端导航业务场景进行调试。主要有以下几点功能：

1. PC 端模拟器接收车机发来的信号

在该项目中，导航的相关业务都是作为服务端向车机中其他模块提供服务，在车机系统中，系统 HMI 连接导航服务

的地址是固定的，我们在车机中开发了一个代理——Sandwich，主要作用是启动导航服务，这个导航服务并不实现真正的导航业务，而是启动了一个空服务，让车机中其他模块能成功建立连接，同时 Sandwich 作为客户端连接 PC 端模拟器提供的导航服务，PC 端的导航服务真正实现导航业务，Sandwich 负责接收车机发送过来的业务请求，并将请求转发给 PC 端模拟器，这样 PC 端模拟器就能接收到车机中的信号，收到的业务请求并做处理再将处理结果通过 Thrift 反馈到车机端。

这个流程我们打通了车机中信号发送到 PC 端模拟器，并可以将处理完的数据反馈给车机端。

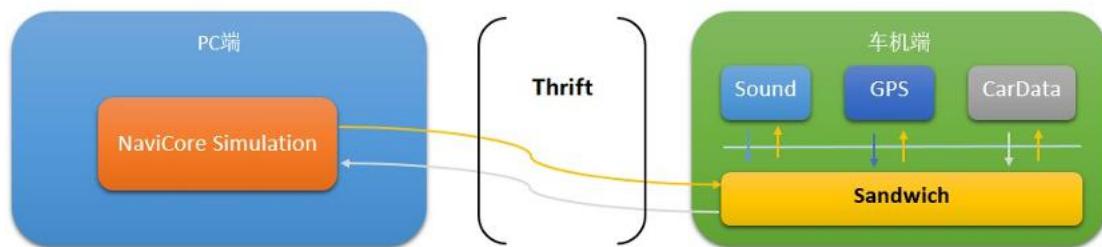


2. PC 端模拟器向车机发送信号

导航也需要连接车机中其他模块提供的服务，如，获取车身数据、获取 GPS 定位信号，将导航语音数据发送到车机语音播放模块等。

PC 端模拟器需要作为客户端来连接车机中的服务，真正连接的是车机中 Sandwich 提供的服务，Sandwich 作为客户端连接车机中其他模块的服务，比如 Sandwich 连接 Sound 模块，GPS 模块，CarData 模块等。

PC 端模拟器需要使用车机设备播放导航音，需要将播放内容发送给 Sandwich，Sandwich 收到播放内容后，再发送给车机中的 Sound 模块，导航音就能播放了。PC 端连接车机中其他模块的工作原理也是一样。



3. 将 PC 端模拟器中显示的地图投射到车机端显示

实现了以上两步，一个使用车机信号调试 PC 端导航程序的环境基本完成了。已经能实现车机信号与 PC 进行双向接收，但是此时导航的渲染能力还是停留在 PC 端，车机中还只是显示了一个系统 HMI 界面，无法看到导航地图展现的效果，这样就会带来一个问题，一些需要强依赖地图的操作可能就无法精准操作，比如点击地图上某个 POI 等。此时需要将 PC 端的展现同步到车机侧。

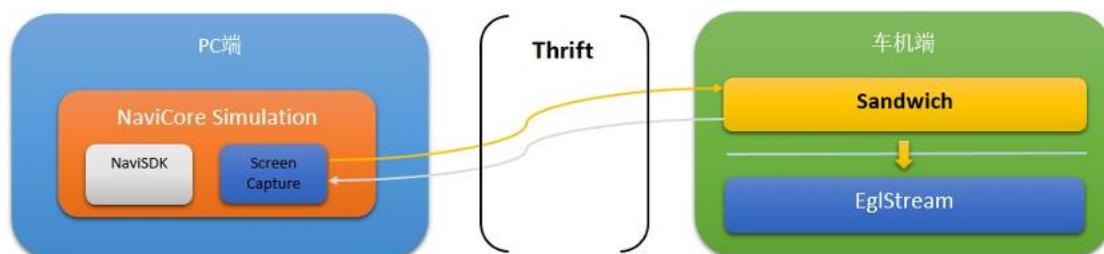
要实现这一目的，一般我们有两种方法：

- **车机与 PC 同步渲染**

车机中的导航正常运行，当导航接收到系统模块业务请求时，先是车机导航进行处理，处理完毕后将信号转发到 PC 端处理，这种方案两端导航业务逻辑并行运行，复杂的业务场景下，两端会同时跟车机进行交互，此时可能会产生互斥，会有两端逻辑不同步的场景，达不到预期效果。

- **将车机中渲染的数据投射到车机端**

在这里我们可以将 PC 上程序每渲染一帧地图则将结果传到车机端，由车机端 Sandwich 负责接收，当 Sandwich 接收到一帧地图像素数据后，负责将此帧数据渲染到车机屏幕上，此时车机中呈现的效果跟 PC 端一致。在该项目中我们采用了这一方案，这种方案中，真正的导航业务逻辑是来自 PC 端，车机中只是一个转发过程，所以不会存在第一种方案中的问题。



但在某些特定的环境下，导航描画会很频繁，发送给车机的数据也会很多，频繁的数据发送可能会带来一定的性能开销，表现上可能会出现延迟。这里可以使用降低图像质量来减少图像数据，例如，可以使用 16 位或者 8 位 BMP 来传输，还可以压缩传输，这样 1920*720 分辨率图像传输大小能控制在 30–50k 左右。

小结

基于车机系统中 Thrift 通信框架，实现的这套远程调试方案，实质是在车机中使用 Sandwich 程序接管车机系统中与导航有交互的全部接口处理，通过 RPC 通信转发，实现了使用真实车机信号调试导航的目的。有了这套调试环境，我们甚至可以直接在真车上边路测边调试，跟以前的路测拿 Log 回来分析、重现相比，整个调试过程，简单，便捷，直观。大大提高了开发效率。

基于 RPC 通信的特性，我们还可以对调试方案再进一步优化，可以加入多客户端调试功能，使用同一台车机环境，不同的模块负责人可以同时进行复杂业务场景的联合调试。

浅析云控平台画面传输的视频流方案

作者：东滔，元机

背景

ARC（高德车机云控平台）是一个基于车载设备业务深度定制的云控平台，通过该平台我们能够实现远程使用不同类型的车载设备。为了让远程使用者像在本地一样使用车载设备，需要将车载设备的画面及时的传回给使用者。因此，画面传输能力是 ARC 平台的一个核心组件。

起初我们采用行业内普遍在用的画面传输开源方案（minicap）。该方案获取到屏幕数据后压缩生成 JPG 图像，逐帧传输到 Web 端进行展示。

由于车机性能比手机差很多，压缩图片消耗 CPU 性能大，在部分低端车机设备上压缩图片能消耗 80% 左右的 CPU，容易使设备使用出现卡顿。同时图像压缩率不算很高，传输消耗带宽大，在低带宽下造成用户看到的画面过度延迟。

因此，我们需要一个解决方案能够平衡传回的画面质量和车机端的 CPU 资源消耗。本文将小结本次云控平台画面传输的视频流方案。



思路方法

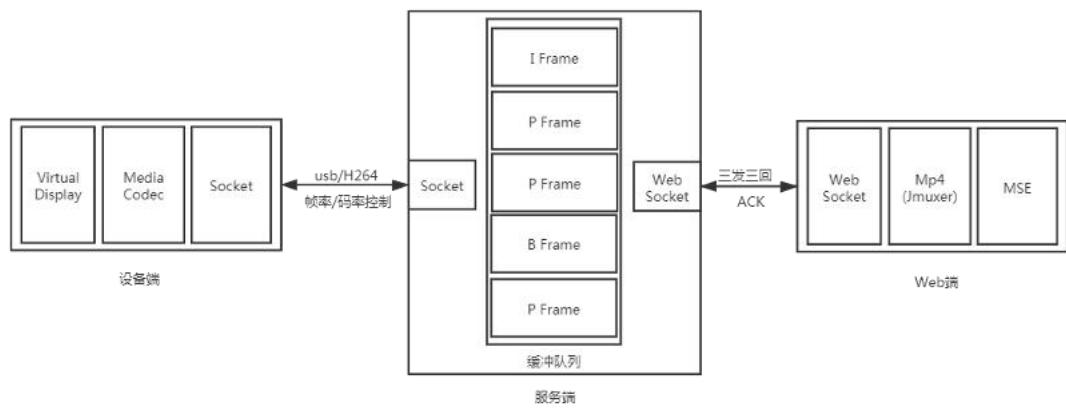


基于图像数据的基本传输链路，为了能够不消耗设备端 CPU 资源，首先想到了图像不进行压缩，先传输到服务端进行处理。但是经过调研，车机的 USB 带宽传

输根本无法满足高清图像不压缩进行传输，高清原始数据非常大，基本1秒只能传输三帧左右的数据。

另一个思路是采用设备端的硬件编码器减少CPU资源的消耗。经过调研Android 4.1开始基本都自带了H264视频编码器。因此，决定尝试采用视频流的方案，在设备端通过硬件编码器编码成视频流，通过服务端转发到Web端进行解码展示。

实现方案



整个实现方案可以分为以下三个部分：

- **设备端**：负责画面的获取和编码。

- **服务端**：处理视频流的传输和控制。

- **Web 端**：视频流的解码和展示。

画面的获取和编码

图像画面的获取直接采用的是 Android 的 Virtual Display。编码方式有多种实现方法：

编码方式	最低API	优缺点
cpp mediacodec (系统api)	4.1	CPU性能消耗较大，不同车机视频硬件编码器支持的色彩模式不同，需要对图像进行色彩转换
cpp mediacodec+surface (系统api)	4.3	无SDK支持，开发难度大
Java mediacodec+surface	5	SDK支持，开发简单，不支持5.0以下版本系统，API兼容性好，无需通过源码编译

由于 Java 方案只能支持 Android 5.0 以上机器，而目前车载市场 Android 4.x 的占比还比较大，无法忽略。因此只能使用 cpp 的方案，最低可兼容 Android 4.3 版本。

视频流的传输和控制

Web 端最常见的直播方案是 rtmp/hls/flvjs 等。但是这些方案最低都有 1–3s 的延迟。对于一般直播平台没有影响，但是对于有实时交互场景的云控平台，要求达到毫秒级延迟。所以，最终决定采用 H264 裸流通

过 Socket 传输的方案，设备端编码 H264 视频流直接传输到 Web 端进行播放。

同时，为了提高使用体验，对视频流的传输增加了弹性控制。通过在服务端加入缓存队列用来监控前端带宽负载情况，根据带宽状况自动调节帧率和码率，优先保证使用者的流畅感。

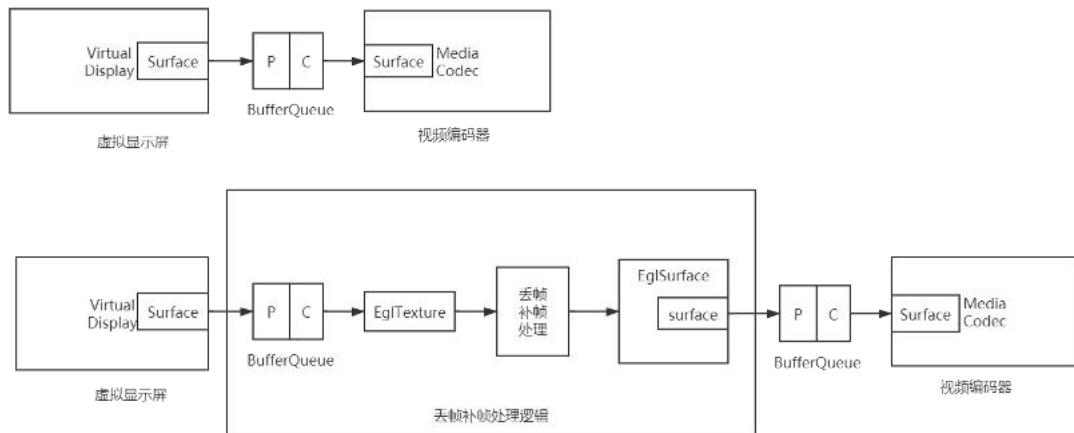
Web 端展示和解码

Web 端展示使用 media source extensiton(MSE) + fragment mp4 的方案，把 H264 裸流封装成 fragment mp4 后，通过 MSE api 进行解码播放，具体实现是参考了开源的 Jmuxer 方案。

丢帧和补帧

默认情况下 Android Virtual Display 产生的最大帧率是 60fps，而我们肉眼 30fps 就能感觉流畅。为了能够节省带宽，我们定义了视频流最大输出帧率是 30fps。在网络带宽较差的情况下，我们还能够降低帧率来最大限度的避免延迟。同时，Android MediaCodec 不支持控制帧率，帧率是由每秒送入的帧数量决定的。因此，我们需要通过实现丢帧来进行帧率控制。

Win7 硬件解码器没有低延迟模式，需要大概 10 帧左右数据才能开始播放，而 VirtualDisplay 是画面有变化才会产生图像帧，因此需要实现补帧来消除解码延迟。



我们通过创建一个 EGLSurface 对丢帧补帧进行处理，通过时间间隔控制 EGLTexture 绘入 EGLSurface 的频度进行丢帧，通过重复绘入最后一帧数据进行补帧。

总结

该方案在 ARC 平台上的使用，在保证传输质量的同时，有效的提升了使用者操作的流畅感。该方案理论上也可以应用于其他类似的云控平台上，如果不需要支持 Android 4.x 设备，采用 Java 层 API 来获取视频流数据，则可以降低开发和适配成本。

基于 Rust 的 Android Native 内存分析方案

作者：迷珑

背景

高德地图车机版运行的车载系统环境绝大部分都是基于安卓的定制系统，且高德车机版底层代码均为 C/C++ Native 代码。因此，在安卓上需要有一种通用的 Native 内存性能分析方案。内存塔(MemTower)是一个基于开源项目 memory-profiler 并移植安卓且优化改进后的方案，解决了之前方案存在的痛点问题，满足了通用 Native 内存性能分析需求。该项目采用 Rust 语言编写，并利用了 Rust 的一些特性来完成对 Native 内存访问的 Hook。

1. Android Native 内存分析痛点与诉求

这一节主要介绍我们为什么要做这件事以及对于这件事我们期望达到什么样的目标。

1.1 现有工具缺陷

Android 在 Java 层面有很完善的性能分析工具，但是在 Native 层面没有完整的解决方案。主要表现在：

- 不支持 Android 4.x，线上统计数据显示 4.x 版本的车机仍占有较大比重，因此这点成为了无法忽视的问题。
- 安卓自带的 malloc_debug 功能在不同的版本上行为不同，而且车机安卓系统大多经过了系统厂商的定制，不能保证这些功能可用。

因此，无法基于 Android 系统自有的功能做到 Native 内存性能分析。

我们团队之前也在这方面做出了一些成果，但还是存在下面几个问题：

- 通过修改编译参数对 Native 代码函数入口/结束位置插桩来进行 Hook，导致了性能严重下降。

- 由于是侵入式分析，对内存问题分析需要单独编译出包分析，解决效率大幅降低，一个内存泄漏问题的排查成本按天计算。
- 缺少精准内存使用数据。

1.2 打造一套完整的 Native 内存性能分析方案

结合上门的问题痛点，我们希望能够有一套完整的 Native 内存性能分析方案。具体诉求表现在下面几点：

- 1) 支持安卓 4.x 在内的绝大多数安卓系统。
- 2) 无侵入式分析，内存问题的发现与精准定位同时完成。
- 3) 性能优异，overhead 低。
- 4) 支持长时间内存泄漏压测。包括车厂客户在内的研发团队都会对导航进行压测，需要能够支持长时间的压测并定位内存泄漏问题。

5) 函数级内存使用数据。原先的方案重点在于解决内存泄漏的问题，获取的内存使用数据不够精确。而我们希望新的方案能够获得详细的内存使用数据，用来支持内存性能优化。

2. 内存塔(MemTower)方案

本节主要介绍 memory-profiler 项目的实现和内存塔(MemTower)方案在移植该项目至 Android 平台上的过程和对原方案的改进。阐述我们是如何实现并满足上述的诉求。

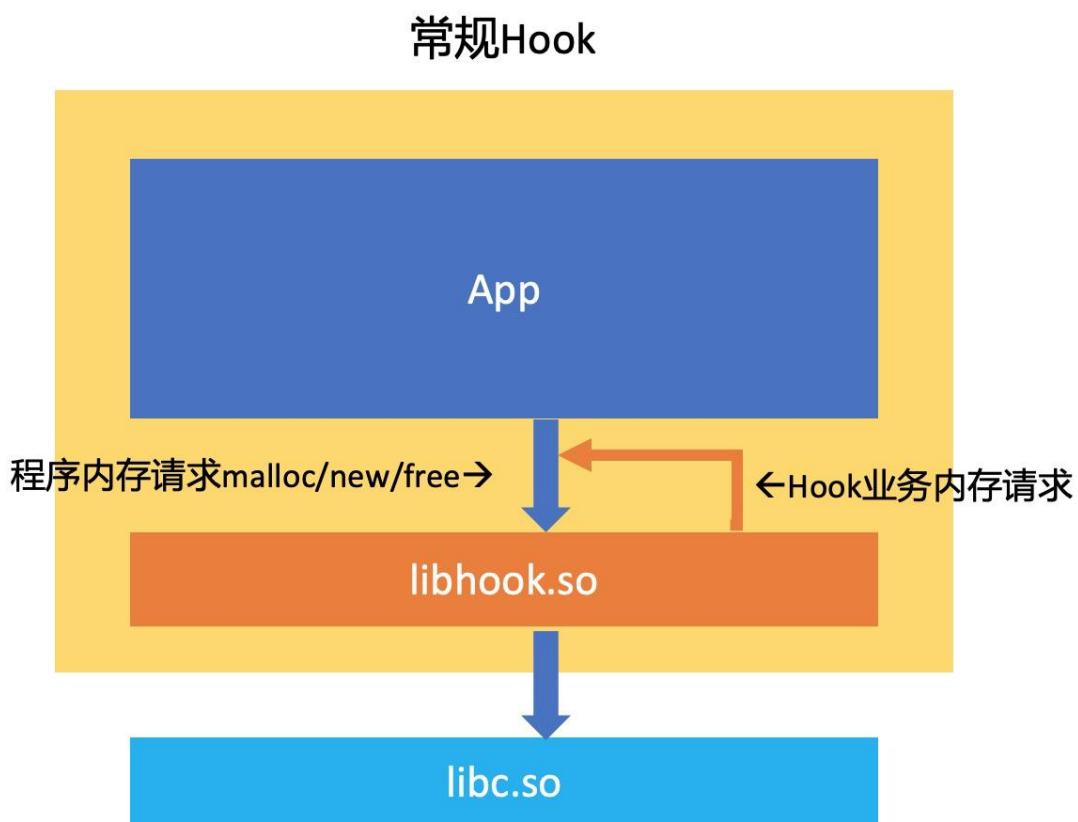
2.1 选择 Rust & Memory-profiler

针对上门的诉求，期望能够找到一种新的解决方案。当时正好在研究 Rust，因此在 GitHub 上结合关键字搜索便发现了 memory-profiler (以下简称 mp)项目，作者 koute 是前 Nokia 工程师。接着才有了后面的内存塔。本节主要阐述 mp 如何结合 Rust 实现内存 Profile 的相关原理和功能。

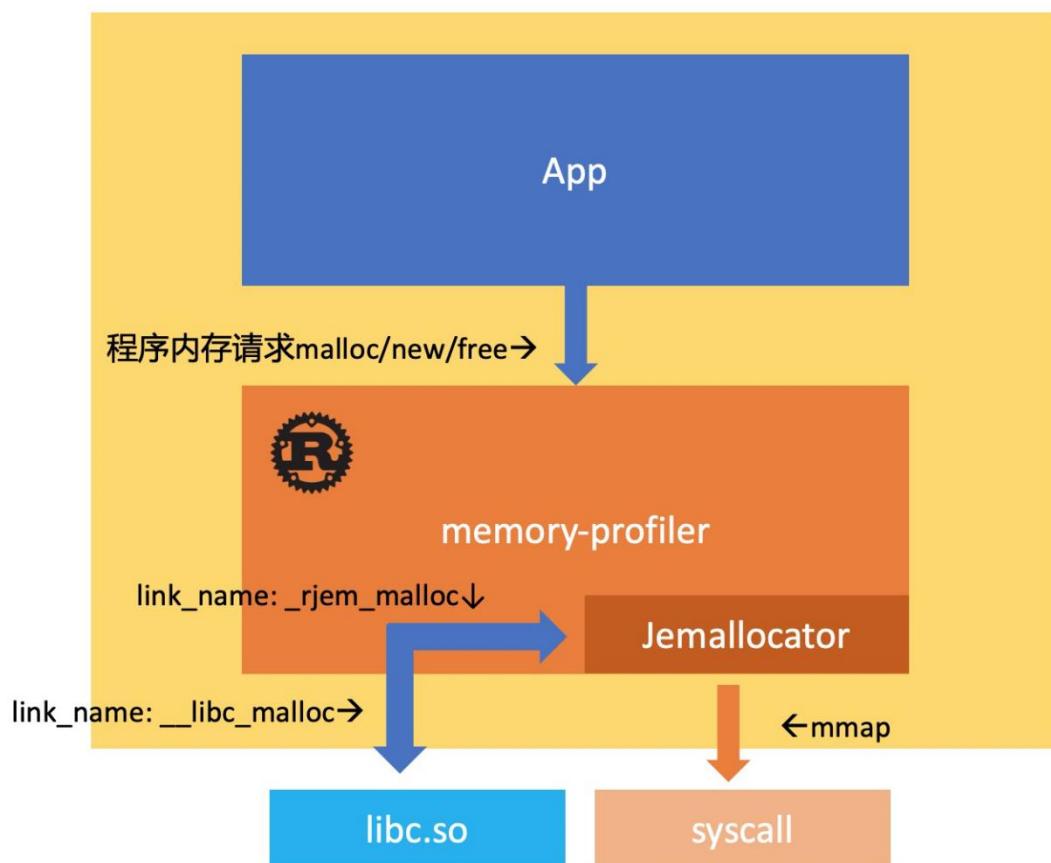
2.1.1 Hook 实现

通常对 Native 内存性能分析使用的方案是 Hook `malloc` 和 `free` 等内存调用请求。`mp` 的原理也是如此，利用 `LD_PRELOAD` 预加载自定义库实现对内存操作函数的 Hook。这种方案最大的问题是容易引发循环 `malloc` 调用。

如下图，Hook 了程序内存请求后，Hook 业务自身的内存请求也会触发内存请求，从而造成了 `malloc` 循环调用，引发栈崩溃。



mp 的做法利用了 Rust 的可自定义内存分配器(Allocator)的特性，将曾经的 Rust 默认内存分配器jemalloc 作为自定义分配器，并在 jemalloc-sys 的 c 代码中将最终的内存申请 mmap 替换成自定义的函数入口(从而也区分应用和自身的 mmap 调用)，最终调用 mmap 系统调用。



将 Rust 内存请求转发给系统调用后，还需要将应用的内存请求继续传递给系统 libc。mp 的做法是通过 Rust 的 feature 开关，可以自行选择两种方式处理应用内存

请求，这两种方式都是通过在 Rust 中指定 `link_name` 属性实现：

1) 直接通过 `libc::malloc` 的 `link_name` 将应用内存请求转发给 libc

2) 通过指定成 `jemallocator` 的函数入口 `rjem_malloc`，使应用和 Rust 共用 `jemalloc`。

最终可以使 Hook 业务使用完整的 Rust 语言功能而不用担心 Rust 自身代码引起的循环调用崩溃。

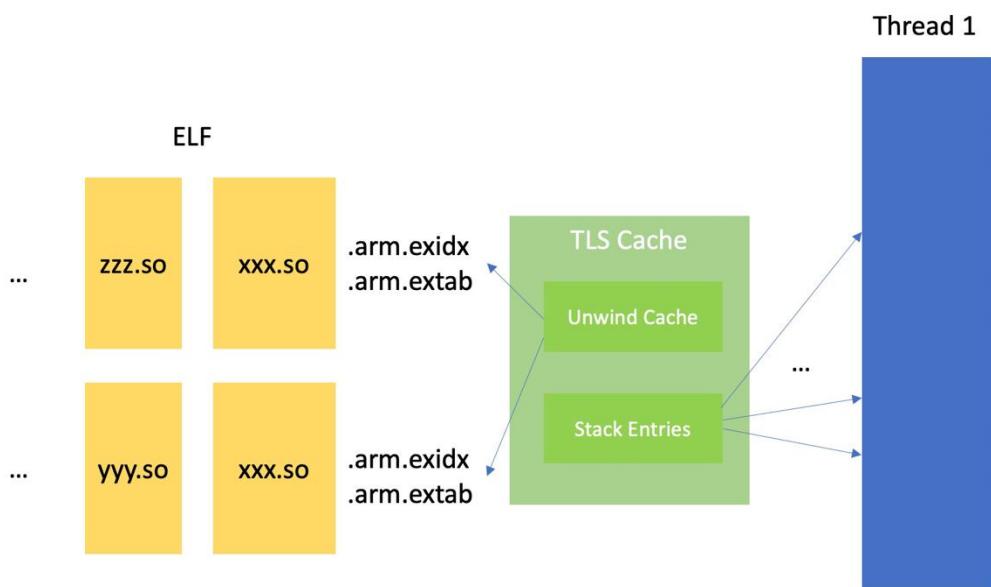
2.1.2 高性能堆栈反解

除了利用 Rust 系统编程语言特性避开内存循环调用之外，作者还利用 Rust 的高性能特点实现了几种高性能堆栈反解。

1) 利用 ELF 的 `eh_frame` 节 (C++ 异常处理机制) 提供的栈回溯信息。

2) 基于 `ARM::exidx + ARM::extab` 的栈回溯，这个是 ARM 提供的 `unwind table`。

具体实现可以看作者的这个 Crate `not-perf`。这里选择第二种做说明，如图下，对每个线程的堆栈都用线程局部存储维护了一套栈帧缓存，这个缓存来自于 ELF 文件中的 `unwind table` 信息，当堆栈的帧在缓存未命中时会把对应二进制的 `unwind` 表被加载到内存，而命中的时候，就不需要去读取文件。通常二进制被加载后它的地址空间就不会发生变化，所以缓存的效率很高。缺点是每个线程都有一套完整的缓存。从系统层面看占用的内存 overhead 很大。



2.1.3 强大的数据分析功能

从 mp 的页面可以看到它除了内存 Profile 外，还有一个对应的数据分析 Server 端，采用 actix-web 框架，

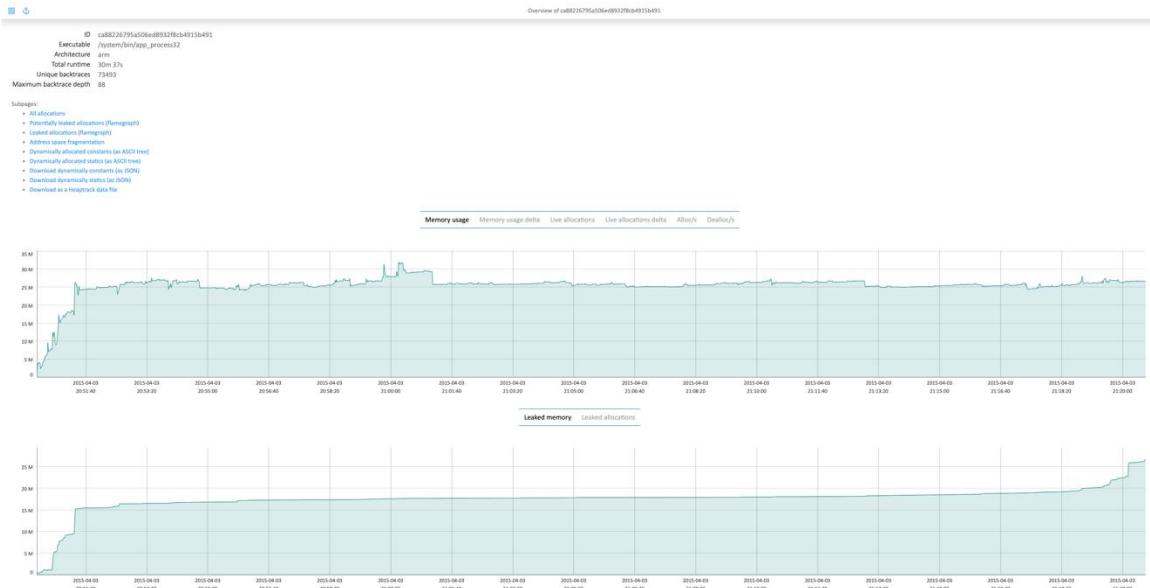
且具备一个非常强大的分析功能。主要特性有下面几点：

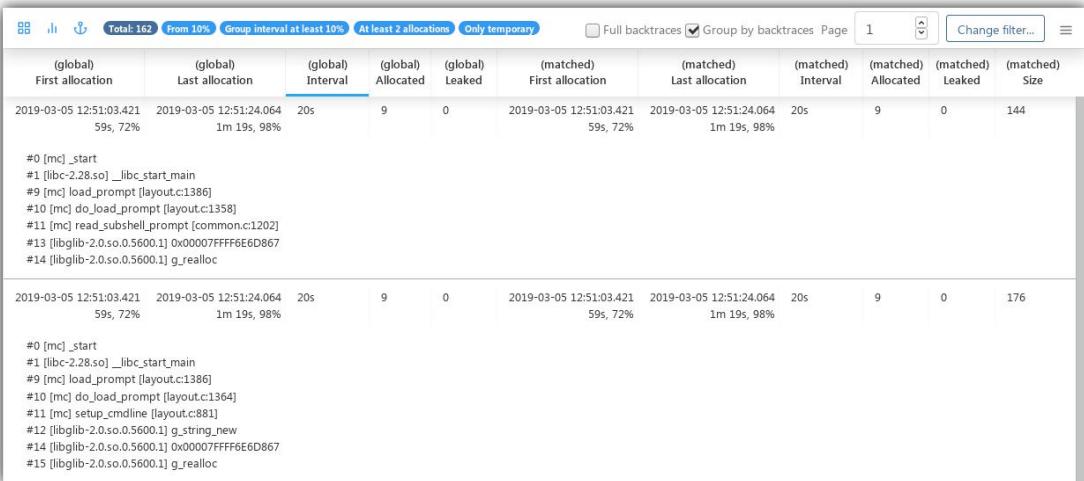
1) 内存使用量和泄漏两种视角的时序曲线非常直观。

2) 搭配了一个非常强大的过滤器，可以实现针对内存生命周期、函数、时间等多维度做过滤查询及其对应的内存火焰图功能。

3) 所有功能具备 RESTful API 接口，可以非常容易的实现定制。

详细的使用说明这里不做过多的介绍。





2.2 移植

了解完 mp 的基本原理后，本节我们主要阐述在移植安卓平台过程中遇到的各种问题(坑)。

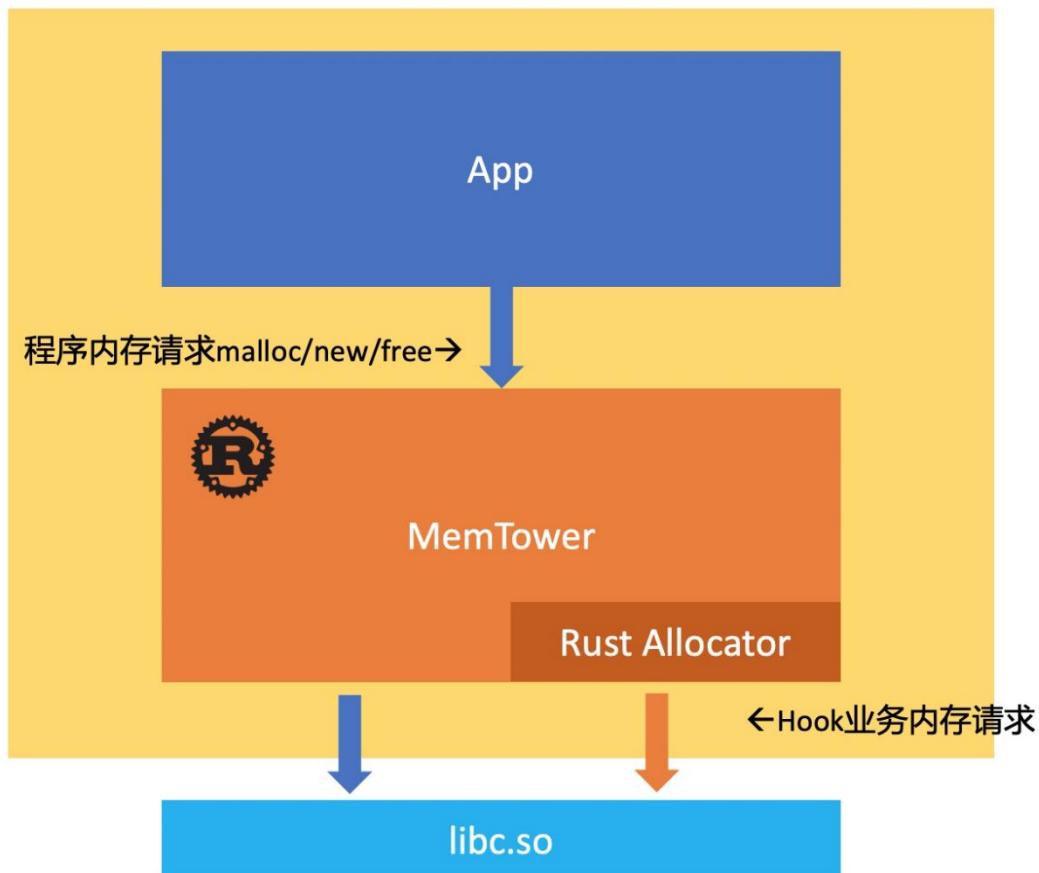
2.2.1 自定义 Allocator

mp 的 Hook 方案在 Android 平台上存在较多问题，主要体现在下面几点：

1)Jemalloc 本身也才是 Android 5.0 开始引入安卓，mp 自带的 jemalloc-sys 会导致一个应用里存在两个jemalloc，最终表现为在不同的版本上有着各种各样的异常崩溃，问题排查成了阻碍。

2) .ARM.extab 是 glibc 提供的 malloc 函数入口别名，但在 Android 平台没有对应这类实现。

因此，我们采用最原始的 `dlsym` 方法获取内存相关函数入口，再将其封装成 Rust Allocator。应用的内存请求也使用这些函数地址。如下图，最终所有内存请求都传给 libc，这样 Rust 的业务代码对 libc 来说是透明的。



2.2.2 栈回溯

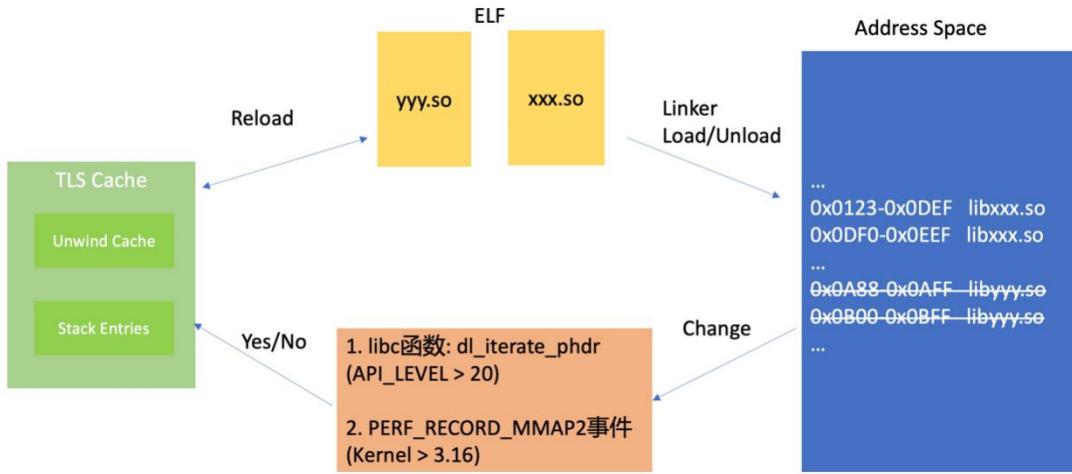
栈回溯这块同样有一些移植修改。上面说到作者提供了基于C++ 异常处理机制的栈回溯方法，但是这个方案要求依赖C++库。而 C 在 Android 8.0 之后才会成为默认依赖。这要求在 8.0 之前的版本运行时应用必须也依赖C++库。因此我们移除了这个栈回溯方案，舍去了这个依赖。

2.2.3 地址空间重载

在程序启动或调用 `dlopen/dlclose` 时链接器会加载(或卸载)ELF 文件，相应的，程序的地址空间会发生变化，这时候栈回溯缓存里的地址空间就可能会失效，需要重新加载(reload)，reload 操作扫描整个地址空间的变更，这个成本很高。与此同时还需要一种低成本获取地址空间变化的方式。`mp` 的实现主要有两种方式：

1) libc 提供的接口 `dl_iterate_phdr`。Android API_LEVEL 低于 21(即 5.0 之前)没有，5.0 之后这个函数的结构体和在高版本 Android 的实现不同。所以 Rust 定义的单一 C 结构体格式会导致读取到脏数据作为 reload 依据，导致非常高频繁地 `reload`；

2) Perf 的 `PERF_RECORD_MMAP2` 事件，这个要求内核版本大于 3.16。因此这在 Android 4.x 上也不具备。



实际运行过程中程序在加载完所有依赖 ELF 后，地址空间几乎很少再变。因此，我们修改为只有在新的 ELF 被加载时才进行地址空间重载。火焰图结果显示可以大幅降低 Hook 时的计算成本。

2.3 改进

到目前为止，内存塔已经可以在支持 `LD_PRELOAD` 的 Android 版本上正确运行了(含 4.x)。但是上面诉求中还有一点无法满足：长时间内存泄漏压测。而且在数据分析过程中，我们希望有更多维度的信息。因此，本小节主要介绍我们对内存塔的改进。

2.3.1 内存泄漏压测

mp原先的定位正如它的名称表述，是一款内存性能分析工具，它记录的是全量内存信息。这点决定了它的数据量规模。在长时间压测一小时的多个业务场景中，根据内存使用量不同，生成的采样数据文件有1GB~7GB之多。这样的数据量无法满足业务的需要。

因此，我们增加了内存泄漏检测模式(`ONLY LEAKED`)，这个模式的原理如下：

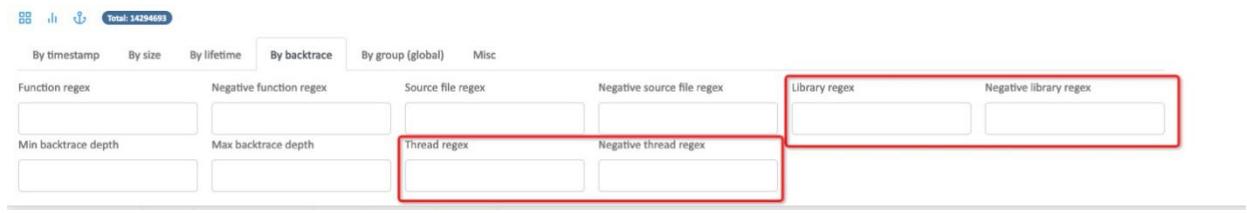
- 1)将记录到内存开辟的每一层栈帧记录到一个字典树(Trie Tree)中，同时记录开辟的内存大小。
- 2)内存释放时更新字典树对应的节点信息。当前泄漏是否达到某个阈值(如 100MB)，是则停止采样。
- 3)在结束采样时把整个字典树存储的未释放内存记录写入文件。

这种模式的优点是最终的数据量非常的小，实际压测一小时数据文件大小在 100~200MB 之间。再进过 mp自带的 `postprocess` 子命令压缩后，大小不足 100MB。不足之处是内存塔需要在内存中缓存一个全量的堆栈

历史数据，当没有新的栈帧记录出现后这个内存增长才会趋于稳定。

2.3.2 增强分析过滤器

导航的业务模块划分和线程很多，因此增加了按线程和库正则筛选过滤器选项。

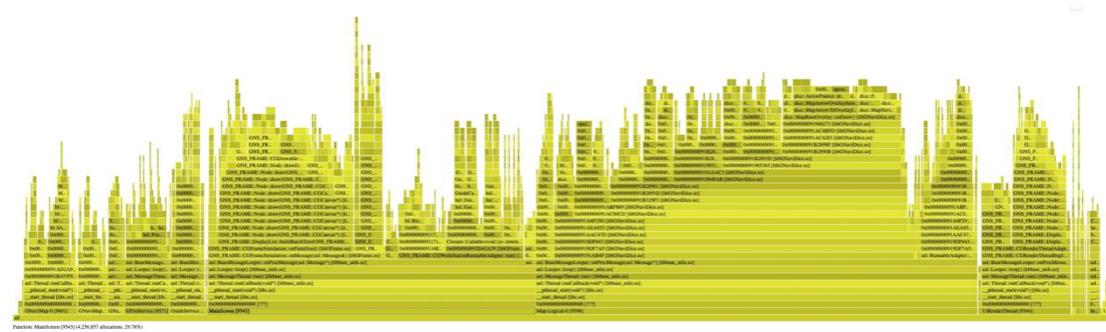


2.3.3 内存火焰图完善

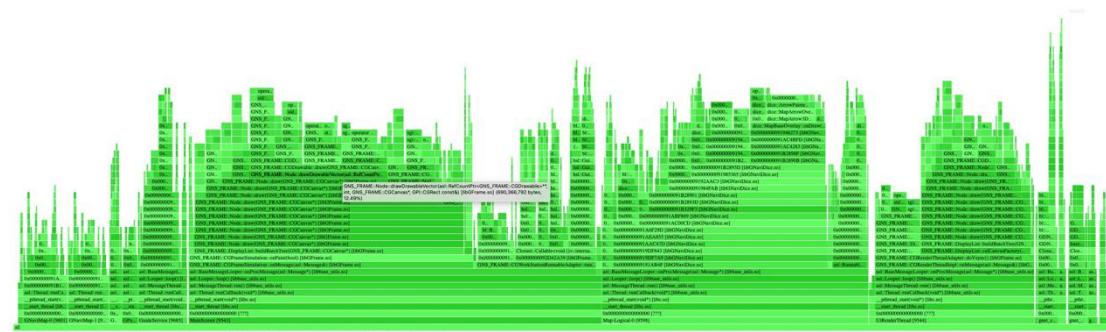
mp 原方案的内存火焰图是以内存大小(allocated)作为火焰图维度，在分析内存性能时内存开辟次数(allocations)也是一个很重要的指标，因此加入内存开辟次数火焰图。这是当初最早改进的功能，而且火焰图的形状类似塔状，就把该项目重命名为：内存塔(MemTower)。

最后一点是原方案的火焰图信息没有以线程为单位划分，我们把堆栈信息按线程区分后会更加直观。

分配次数火焰图



分配大小火焰图



3. 内存塔的能力及更多可能

最后一节介绍下内存塔提供了什么样的能力、收益以及还有哪些可能。

3.1 能力

内存塔(MemTower)在Android 8.0以下依赖 `setprop wifap_com.xxx:xxx` 和 root 权限的能力，8.0以上版

本如果没有 root 权限还可以通过配置 Android 项目 `wrap.sh` 来加载内存塔库。另外，由于 mp 原生支持 Linux 的原因，我们也成功适配了奔驰戴姆勒这类嵌入式 Linux 项目车机。

- 支持平台：Android 4.x、5.1.1 和 7 或更高以上版本(5.0 和 6 系统存在 Bug，无法设置 `setprop`)。Linux x86_64, AArch64, Arm.
- 采样方式：非侵入式。非 Root 设备可选侵入式方式。
- 采样模式：常规性能分析模式和内存泄漏压测模式。
- 特点：高性能堆栈反解、完善的内存分析 Insight 体验(多维度过滤器分析、内存火焰图等)。

原先发现内存泄漏问题重新出包二次压测分析，再推断可能泄漏点的流程耗费时间按天计算。利用内存塔(MemTower)做一遍测试后几分钟即可解析出精细化数据，大幅降低了内存性能问题分析成本。

mp 提供的这套 Hook 思路和高性能堆栈反解其实可以不仅仅局限在内存方面的分析，还可以针对 IO 性能分析或其它问题上。

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德车载导航 Android 平台 DR 回放技术方案

作者：亦勤

导读

DR：（英文为 Dead Reckoning，航迹推算）。用于推算的传感器大致有：陀螺仪、四轮速、车速脉冲、3D 加速度计等。在车载导航中，航位推算是使用先前确定的位置，通过测量移动的距离和方位，计算出经过的时间后的位置。航位推算受累加误差影响，随着时间推移，推算出的位置误差会越来越大，需要配合 GNSS 位置修正。

高德车载导航和业内主流的汽车厂商有很多合作，在前装项目中，汽车厂商为了能更好地为用户提供更为准确的导航定位功能，大多都会选择 DR+GNSS 的定位方案，而这种定位方案数据源除了 GNSS 信号外，还有传感器的信号（陀螺、加速度计和车速等）。市面上现有的 GPS 回放 APP 和方案只有针对 GNSS 的，这样车载导航内部在进行需求验证、问题修复验证时就需要实车路测，需要占用很大的人力投入和时间。但

在真实的业务场景中，不是每个项目都能满足有实车的条件。

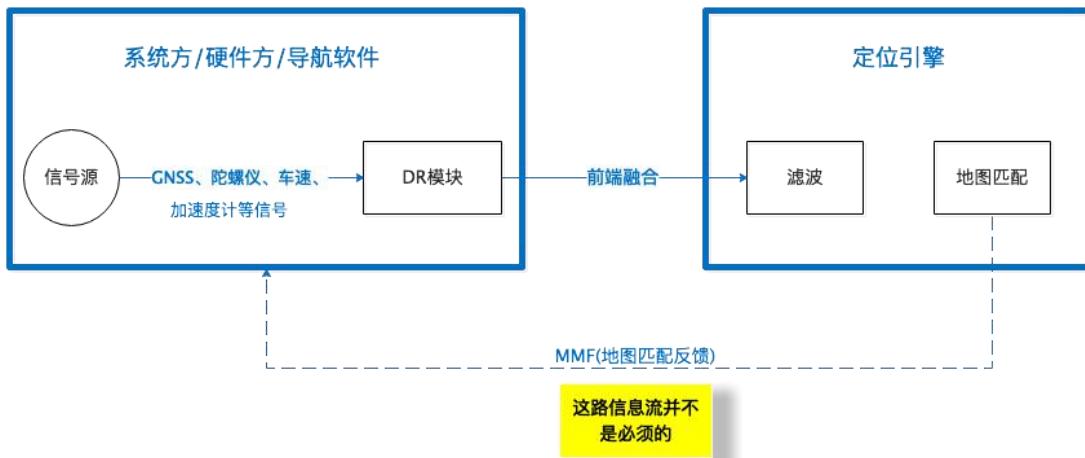
本文将介绍目前高德车载导航 Android 平台的 DR+GNSS 回放方案。

高德车载导航 DR 模式

前端融合模式

DR 又分为前端融合和后端融合。前端融合指的是信号融合的工作是由系统方/硬件方负责的，高德车载导航只负责地图匹配工作。也就是车载导航收到系统传过来的信号是经过系统方/芯片融合计算后的信号，与纯 GPS 的区别在于未定位状态下也会有信号传到 HMI 层。前端融合模式也包含前端融合+MMF 模式，MMF 是指地图匹配反馈信息，由于 GPS 和陀螺仪等信号都不能保证百分百准确，前端融合的推算也会产生偏航，尤其在隧道场景，所以需要 MMF 在适当时机纠正融合信号。

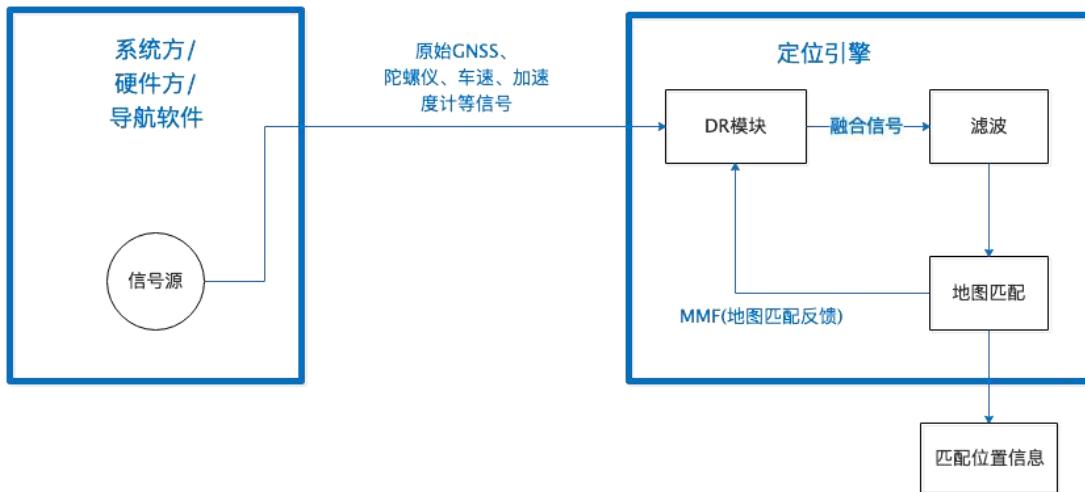
融合信号是将 GPS 信号、陀螺仪、加速度计和脉冲等信息综合处理后得到的轨迹信号。



后端融合模式

简单点说是指融合信号的生成工作是在车载导航定位引擎内部实现的。对于定位引擎来说，后端融合模式的输入信息是 GPS、陀螺仪、脉冲、加速度计等原始信号。

所谓“后端融合”是指惯导系统推算出的融合信号，是与地图数据和导航引擎通过通信协议紧密配合的，导航系统仅在汽车启动时使用 GPS 信号进行定位，车辆行驶过程中始终以惯导系统推算出的融合经纬度信息为主进行导航，只有当导航出现偏差时，才选用 GPS 信号精准时的数值进行实时校正。因此，在 GPS 信号出现折射、反射、衍射、漂移、甚至失去 GPS 信号时仍能精准导航。

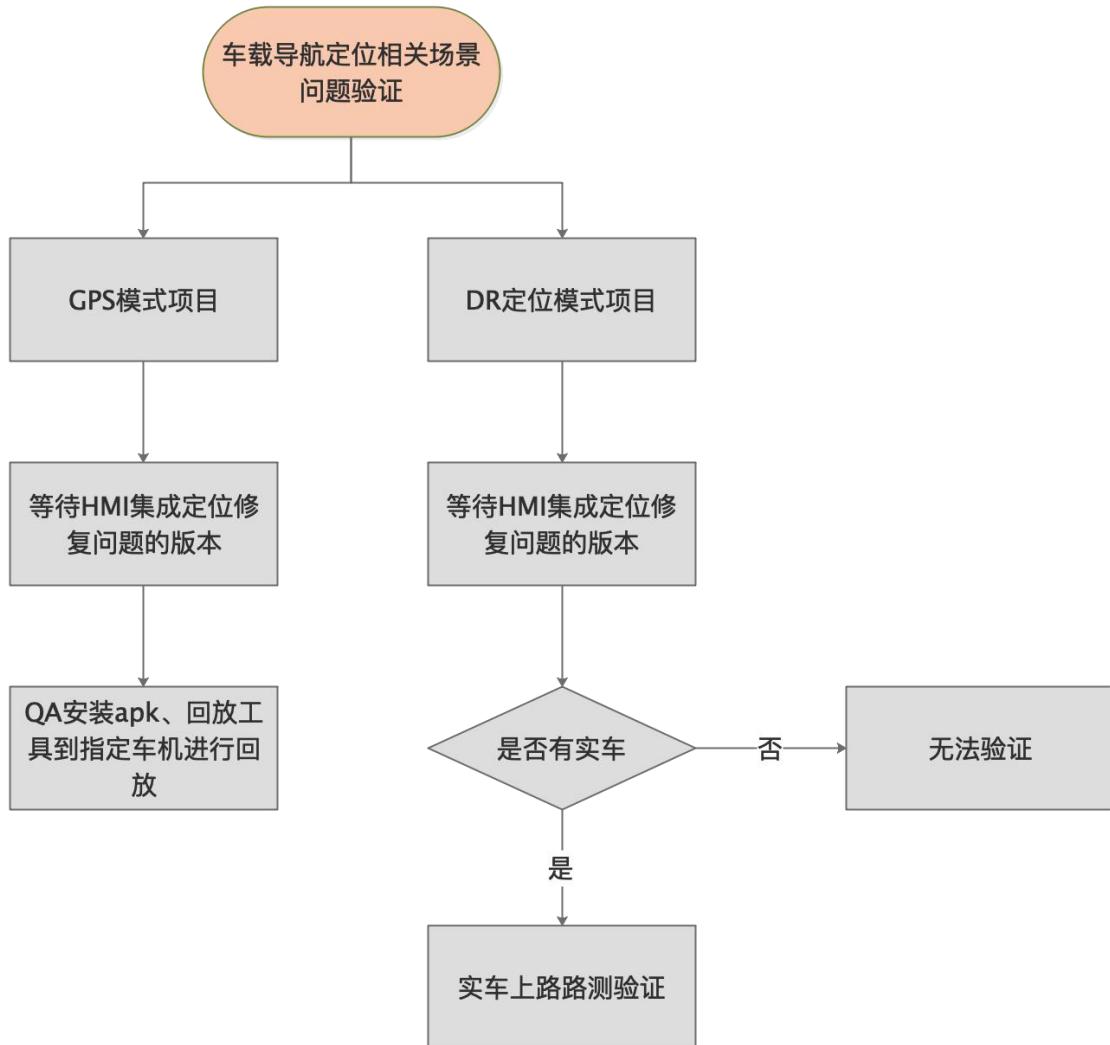


之前了解到的 Android 平台信号回放都是通过记录的经纬度等信息，通过调用 Android 原生模拟信号接口进行设置，而应用从原生接口接收到的定位信号就是模拟信号。

缺点

依赖原生模拟信号的方式只支持 GPS 模式，对于 DR 定位模式还需要依赖传感器信号的输入，目前 Android 原生还未支持这块数据的模拟。

在 DR 回放工具之前车载导航定位相关场景问题验证流程如下图：



DR 回放工具前的 QA 验证流程

对于 DR 定位模式项目，之前都是上路验证测试，人力和时间成本高。

解决方案设计

高德车载导航信号输入方案大部分都是跨进程通讯，所以回放工具设计最大化模拟了原来车载导航数据获取的方式，使用了独立进程进行回放。

回放工具分为前端融合、后端融合模式，目前两种模式回放的通讯方式都是通过自定义 AIDL 协议。

回放文件选择

回放文件的选择有两种，一种是车载导航 HMI 将收到的信息自定义格式保存到文件中；另一种是使用定位引擎已有的定位信息日志文件。

日志方案对比

自定义定位文件

优点：

信息格式定义自由，内容存放为需要字段

缺点：

引擎字段变更需要及时修改，同步性较差，且高德车载导航增加了一种日志文件运行时的读写，对车机性能有一定消耗

定位引擎定位文件

优点：

定位引擎字段变更无需HMI自行修改；减少一份写入频繁的文件操作

缺点：

定位引擎日志文件存放了较多无需关注的信息，文件较大，拷贝读取相对比较耗时；引擎如果字段顺序变更，回放工具解析需要修改

综合考虑到两种回放文件的优缺点，决定复用引擎的定位信息文件，车机性能比较吃紧，不能因为这个功能增加车机消耗。

回放方案选择

实现回放有两种方式：

一是通过跨进程方式，外部程序通过 AIDL 传输数据给到车载导航。

另一种是将回放功能集成到车载导航中，做为车载导航中的一个彩蛋功能。

以下为两种方案的优缺点：

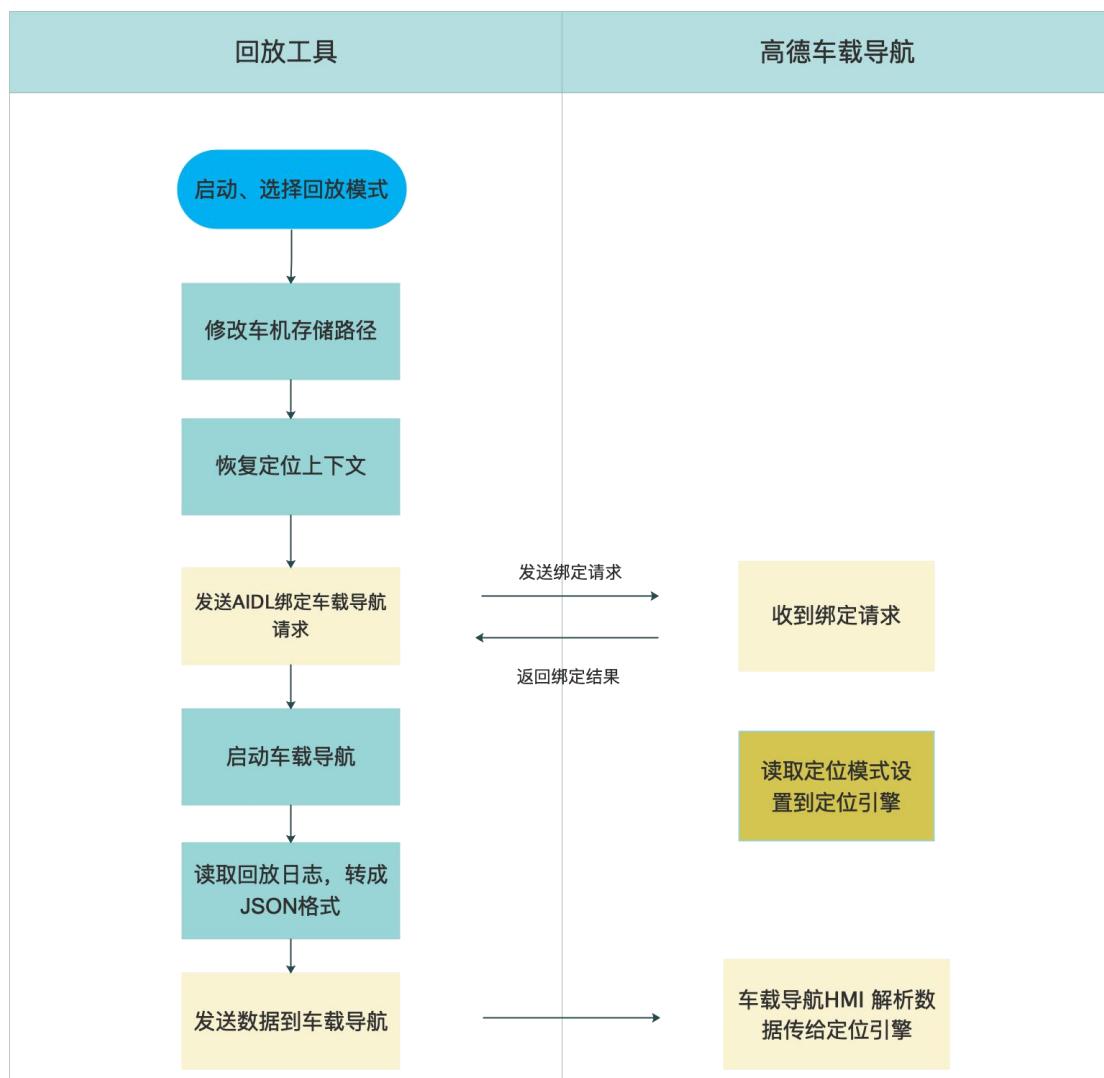
回放方案选择

独立进程AIDL回放	车载导航内部自行回放
优点 与原始信号接收方式相似，能较大程度还原信号输入 独立，方便修改维护和集成 对不同的高德车载导航大版本都能方便修改兼容使用	优点 使用者无需关注内部需要操作步骤（如车机自定义路径），较为简单
缺点 车机的日志文件数据路径需要配置 因定位引擎恢复上下文与初始化有时序要求，回放工具回放步骤需要注意，否则容易出错	缺点 使车载导航 CPU 占用升高，对性能差车机有一定的影响 不易于维护同步，不同版本的车载导航需要各自维护

综合以上问题，我们最终选择了独立进程回放的方式。

该方案是通过模拟跨进程传输信号方式，自定义的 JSON 格式复用了高德车载导航标准的后端融合传输方案的格式，最大程度还原了后端融合高德车载导航 APP 端接收、解析的流程。

工具回放实现流程如下图：



从上图可知，DR 工具实现了 DR 项目台面回放验证问题能力，大大缩短了 QA 同学验证定位场景问题的时间，如原本实车路测需要 2 个小时，而现在只需要 0.5 小时则可以实现从打包到回放查看。而对于外地路测发现的问题复现和问题修复验证则从无法验证到本地实现验证。

旧方案反馈和新的优化

旧方案反馈

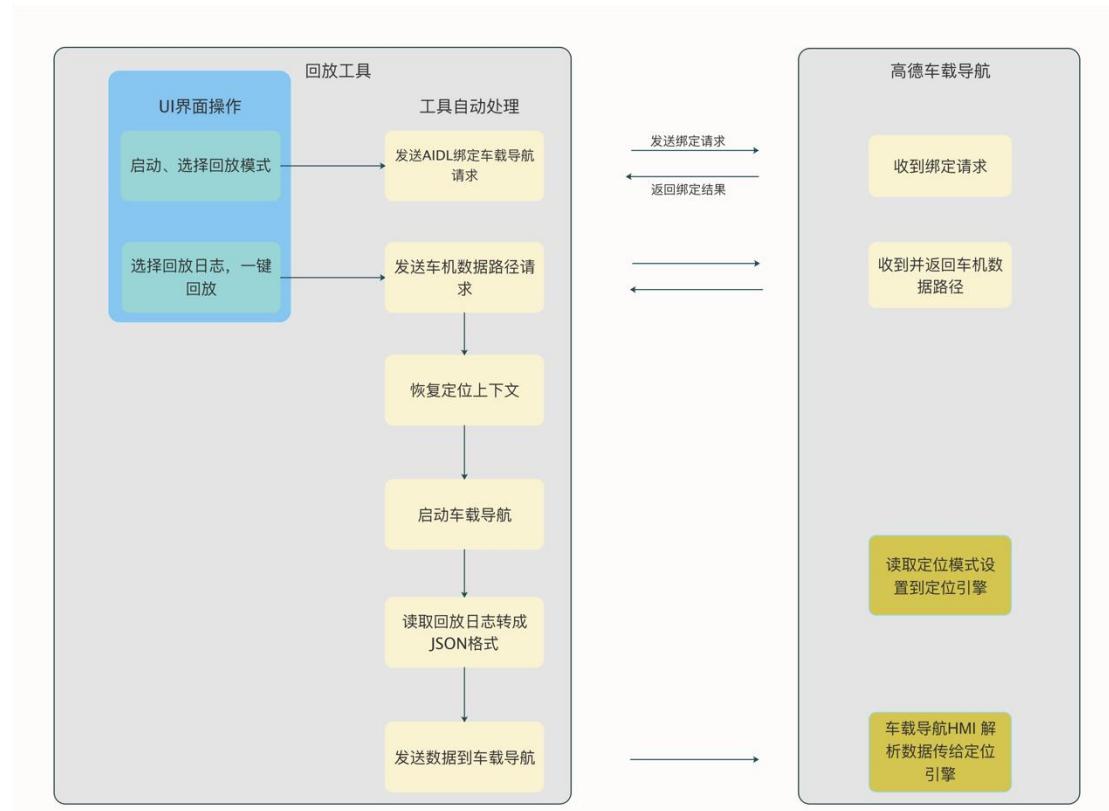
以前的工具已经在 Android 平台高德车载导航项目上广泛使用，带来便利的同时，也发现了一些需要优化改进的地方，该工具在后端融合回放时存在几个问题：

- 1、人为操作步骤多，需要操作 5 步才会开始回放。
- 2、恢复定位上下文经常失败，且失败原因没法快速调查定位到。
- 3、前装车机自定义数据存储路径情况较多，导致每次使用工具回放都需要先去了解车机具体路径再进行修改。

优化规划

针对以上问题，对工具执行步骤进行缩减，将 5 步缩减至一键回放，对于车机自定义路径问题，通过已有 AIDL 通道新增获取车机自定义数据路径，去掉人为填写必要。至于恢复定位上下文经常失败问题，目前总结多是因为自定义数据路径没有权限导致，可通过命令方式尝试赋值权限，如果失败则做个原因提醒。

优化后的方案如下：



小结

通过回放工具的推广使用，大大缩短了高德车载导航 Android 平台的定位问题修复验证、路测问题的回放复现等工作量，大大减少了实车路测带来的时间成本和人力消耗，提高了研发和 QA 团队的效率。

招聘

阿里巴巴高德地图工程技术中心长期招聘 Java、Golang、Python、Android、iOS 前端资深工程师和技术专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

架构篇

高德深度信息接入的平台化演进

作者：Quark

导读

本文介绍了高德地图中 POI 深度信息接入在平台化过程中的一些思考和实践，从最开始的单体应用，随着业务发展面临挑战，从业务角度提出解决问题的思路和方案，进而转化成技术设计并落地实现的过程。

背景

POI 是 Point of Interest 的缩写，即我们通常理解的地点信息。对普通用户而言，POI 数据除包含名称、坐标等基本信息外，还会展示图片、评论、营业信息等内容，这些我们统称为深度信息。作为真实世界在线上的直接体现，其丰富度、准确度、新鲜度对用户的出行决策起到了至关重要的作用，也是高德地图从生活服务等多方面服务大众的基础。

为了丰富深度信息，我们通过多种途径对接采集数据。每个数据接入源称之为一个 CP(Content Provider)。最初只有

少量 CP 的时候，每个 CP 建立一个应用，完全独立的存储、独立的代码，甚至采用的是完全不同的技术栈。

然而，随着接入规模不断上涨，这种单体应对模式逐渐无力支撑，无法批量生产、更新、运维、监控等问题成为了业务迭代路上的绊脚石，大家花在基础维护等事务上的精力占比甚至超过了业务迭代。

用一组数据说明下深度业务的发展速度：一个季度工作日 130 天左右，新接入的任务数量却多达到 120 个以上。截止目前接入的任务总数是研发人数的 100 倍以上，单日处理数据量达十亿规模。基于对这个趋势的预判，深度团队提前开始了平台化的探索。

平台化实践

平台化的思路是明确的，但是平台化的具体设计实施却有诸多不同的选择。

大多数数据接入系统的设计目标都相对比较纯粹：作为接入系统，只要把数据拿到并输入到本业务体系内就可以，剩余的如数据解析，业务处理都由下游的其他系统再次加工才可形成真正的业务数据，即接入系统从设计之初就是无状态的，对数据本身的理解也基本与业务无关。

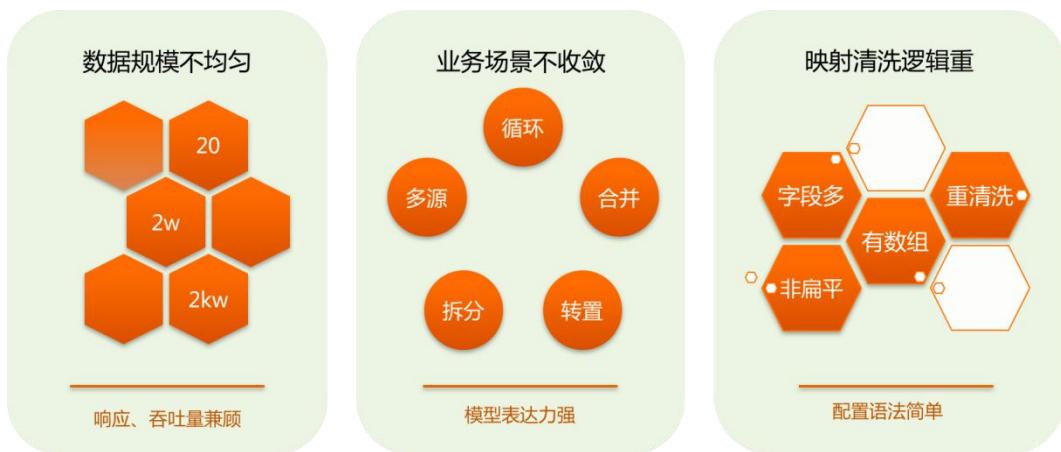
但是考虑高德深度信息接入业务的特殊性，我们平台化时并没有采用这个方案，而是采用一种更集约化的思路，接入平台本身对数据就需要有充分的理解，不仅负责数据接入，还要负责数据解析、维度对齐、规格映射及生命周期维护等相关内容，平台直接内置了深度信息处理流程的全部管控逻辑。

另外，不同于一般的接入系统，除研发(RD)外，产品(PM)也是系统的第一用户，平台需要有能力让 PM 在了解有限技术约束的条件下自主完成全流程数据接入、分析和调试，这就对平台所见即所得的实时设计调试能力提出了极高的要求。从平台设计角度要解决以下一些难点：

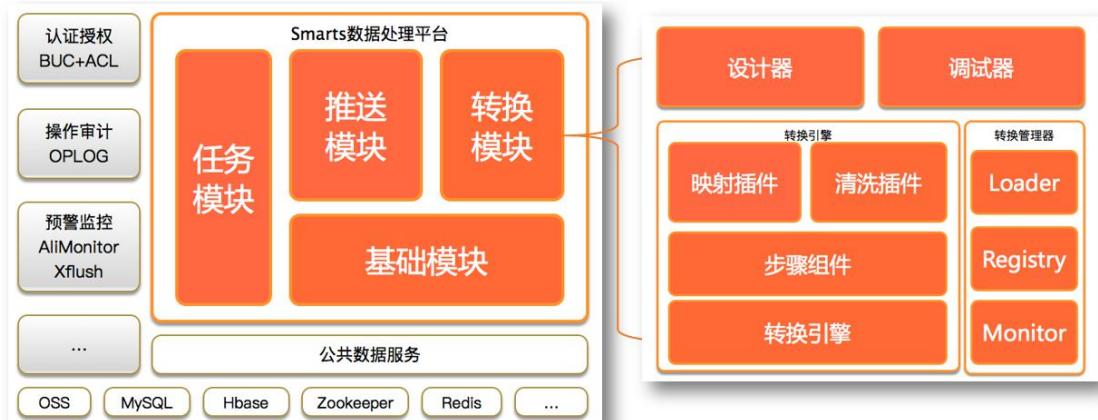
- **数据规模不均匀**：不同 CP 的数据量和数据体积相差巨大，有的源数据量有几亿条，最少的 CP 甚至只有一条数据。具体到每条数据大小也差距悬殊，如部分数据单条达到 7.5M，有的则只有一个字段，仅几个字节。
- **业务场景不收敛**：深度数据来源多且杂：有三方合作接口、离线文件、经济体内 OSS、ODPS、MetaQ 等，

且 CP 数据结构和关联匹配规则多种多样、无法预知，需要平台在设计上能支持各种场景下的维度对齐。

- **映射清洗逻辑复杂**：这里还有一个和常规业务不同的点，高德深度数据采用 Schema 比较松散的 JSON 方式组织，有多层嵌套对象及数组字段，且不同行业的规格并不一样，平台最终需要把数据组织成近百套不同规格的数据，这种松散的、非扁平二维表的数据处理也是挑战之一，尤其是存在数组上下文的场景里。



最终我们设计出如图所示的平台架构，平台集成了基础、转换、推送和任务调度四个模块，配合完成深度信息接入的全部工作。



平台分为几个模块：

基础模块：负责 CP、行业、规格、权限等基础信息的在线化，实现统一管理。

转换模块：负责数据获取、维度对齐、规格映射等处理。

推送模块：负责转换后规格数据推送至下游准入服务。

任务模块：负责对任务的管理，如任务类型、积压策略和数据差分等。

转换引擎设计

转换模块由转换引擎、转换管理器、设计器和调试器四部分组成。

为了降低系统的设计复杂度，所有业务规则的自定义部分均由转换模块支持。转换模块作为业务自由度最高的模块，使用相同的底层支持了上层业务的预转换、转换和数据分

析三种场景，是系统能支持各种复杂业务场景的核心部分，转换引擎要支持数据获取、维度对齐、规格映射清洗等配置化及调试功能最复杂多变的部分。

数据获取

数据获取能力不仅要支持常见的 HTTP、OSS、ODPS、MTOP、MetaQ 及 Push 服务等多种方式，而且还要支持组合叠加。

比如，先从 OSS 下载一个文件，解析文件行，根据解析的数据，再调用 HTTP 服务等场景。

为了支持近乎无限的业务叠加能力和所见即所得的设计效果，我们调研了阿里经济体内外的多种解决方案，如 Blink、Stream 平台等，没有发现可以直接满足我们业务需求的组件，主要问题为：

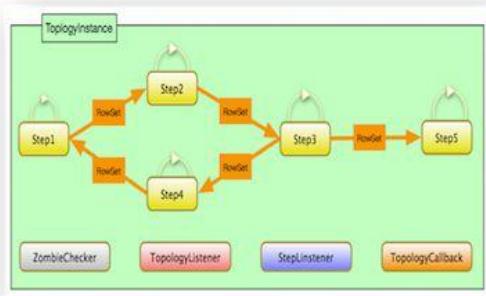
- 基于技术维度组织，需要大量写代码或理解技术语义，无法提供业务视角，对数据 PM 的理解和使用有极大的障碍。

- 步骤数据视图是扁平二维表，无法实现松散结构传递和处理。如果在步骤间自定义业务约束及协议则过于复杂。
- 无法支持实时无副作用调试，运行流程和调试流程数据会互相污染。

基于以上分析，我们决定不在上述平台上进行二次开发，而且直接基于当前业务场景定制一套引擎，虽然这些引擎无法直接使用，但是PDI的步骤组织及驱动方式和我们的业务场景比较匹配，从自由度、表达力和直观性几个角度考虑，转换引擎舍弃了DAG这种依赖计算和并行调度都相对容易的技术模型，使用和PDI类似的有向图模型进行组织。



为了最大限度的支持 PM 直接对业务场景进行描述，我们最终采纳了 PDI 的转换引擎设计思路，直接以原始有向图方式对步骤进行驱动执行，最大限度保持设计直觉和运行时的逻辑一致，从而不需要实现引擎层面的翻译器、优化器、执行器等复杂组件。



步骤调度机制

- 多头多尾
- 生命周期自治
- 自主数据交换
- 超时探活

数据传递机制

- 进程内通信
- 纯内存交互
- 异步并行
- 背压阻塞

1

数据规模不均匀

秒级初始化，每秒百万级处理能力

2

业务模型不收敛

插件化的业务模型拓展能力

为了保证引擎的执行效率和安全性，我们保证步骤间数据传递不会跨进程，所有数据交互全部在内存内完成，且步骤之间均为异步并行执行，通过背压感知机制从后向前传导，平衡各步骤间的处理速度差异。

维度对齐

维度对齐是指把不同数据源、不同维度的数据通过给定的业务规则关联整合成某一种维度的数据，比如深度信息业

务一般需要整合成 POI 维度的数据。理论上有了引擎提供，能直观表达并堆叠业务的能力可以实现维度对齐的需求。

但是，深度信息还有一个问题要解，即面对数据 PM 使用实时调试的需求，所以无论复杂还是简单的转换都需要能随时调试，并直观地展示结果，方便数据 PM 快速分析和排查。

常规 ETL 里都会涉及维度对齐的问题，但是由于常规业务一般都是二维数据表间的关联整合，所以像 PDI 之类的方法基本都是通过 SQL+临时表的方案进行处理，在设计时即绑定了输入输出，调试和运行并无本质的区别，或者调试时需要修改配置，强制输出到一个临时存储，这意味转换引擎需要依赖特定的外部环境（如特定的数据库表），造成调试和运行时的数据会互相污染。

我们的数据天然不是二维结构，无法平铺到表中，自然也就无法使用这种方案。我们采用的是只依赖本机内存+磁盘的方式进行数据处理，如 flatten 这种数据打散的需求直接用内存实现，不需要借助存储；像 merge join 等可能全量数据交叉运算的直接采用本地磁盘做辅助，实现了全部

功能都不需要外部特殊环境支持，秉承这个思路，我们最终实现了具备如下能力的转换引擎：

纯引擎

- 不写数据，不生成执行记录。
- 不依赖任务及特殊执行环境。
- 可以随时初始化并执行。

数据透出

- 转换配置不需要指定输出，数据输出步骤动态挂接。
- 多种场景管理器：任务场景会写到 DB，调试场景通过 WebSocket 回传到前端页面。

有了这样的引擎，综合考虑调试场景的要求，转换在设计时不再需要指定输出目标，输出目标会在运行时由场景管理器根据调试场景和正常运行场景动态挂接，避免数据互相污染。平台在 Web 端支持几乎所有层次的所见即所得的调试分析功能，覆盖了预转换、转换、清洗、推送等几乎所有环节。

规格映射清洗

为了支持松散 Schema 映射和透明扩展，转换的行模型 (RowSchema) 创新性的设计为双容器结构。

- 主数据：承载上游步骤的直接结果数据。
- 数据托盘：承载转换参数、步骤变量、映射结果等内容。



映射步骤通过映射类型、映射规则和清洗参数支持映射清洗一体化。

- 正向映射：自上而下进行数据提取，以扩展 JSONPath 表达式（扩展了主数据、数据托盘、数组循环 item 等上下文语义）为主，多种映射类型为辅。
- 反向清洗：自下而上逐层清洗，可任意叠加策略。

转换模块通过步骤、映射、字段清洗三个层次对数据进行处理，PM 使用时只需要通过 Web 界面拖拽对应组件并简单填写一些业务参数即可完成配置。

为了避免业务黑盒问题，系统设计不同于 Stream 平台的一个地方是系统组件会向后兼容，步骤插件、映射插件、清洗插件都没有版本的概念。系统不支持的自定义业务在各个系统模块均可以写脚本(Groovy)的方式托底实现，但是不允许上传二进制包，代码必须以配置形式直接体现，避免后期的维护问题。

生命周期管理

生命周期是指系统要在适当的时机触发数据的新增、更新、删除操作。站在数据接入的角度，删除是一个较为复杂的过程，业务术语称之为下线。要说清楚下线问题得先说下深度信息的任务模型。

目前我们支持批处理和流处理两种模型，如大家直观理解的，批处理任务每次执行都会递增一个批次号，比如常见的定时任务类型。流模型指任务一旦打开就会始终保持运行，数据一般是通过 MetaQ、Push 服务等方式被动接收的，没有批次概念。

为了满足业务需求我们支持批次过期、时间过期、条件下线三种策略，且支持多策略叠加使用。而这些策略设计时也有各自要考虑的内容，如批次过期怎样避免扫描全批次的历史记录、历史和重试场景批次号的共享递增问题；时间过期如何避免对每条记录绑定定时器造成的定时器数量爆炸等等。

生命周期管理涉及到比较多的任务模块设计内容，比如任务调度模型及多机分片机制设计，任务预警熔断逻辑设计，存储表的设计等，由于深度信息业务的集成需求，接入平台没有选用开源或阿里经济体现有的任务调度框架，而是自己定制开发了一个，篇幅有限这里不再展开论述。

小结与展望

深度信息接入平台见证了高德深度接入飞速发展的几年，以极低的人力投入支撑了高德在各垂类领域的深耕拓源，为高德向生活服务类高频应用拓展提供了底层数据支持。未来我们还将在全链路 Debug、运营精细化场景支持、非标数据处理、自由业务编排平台等方面继续深化和演进。

招聘

阿里巴巴高德地图信息研发中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德云图异步反应式技术架构探索和实践

作者：喜洲

背景

高德云图是高德地理信息基础能力的出口，对外提供包含搜索和导航等服务接口数量超 700 个，接入应用达 40 万以上，日均处理请求量超百亿，日均 QPS 峰值过百万。高德云图服务端包含开放平台、苹果地图和多类行业解决方案，服务客户包括个人与企业开发者、企业专有用户，以及手淘、天猫、支付宝、飞猪、Lazada 等阿里经济体团队。

传统服务端架构一般采用同步阻塞模型，这符合常人思维模式，但同步等待浪费系统资源，且通过分配更多线程来支撑更多请求的方式，会导致上下文切换和锁竞争，拉低资源利用率。基于一个请求一个线程的服务模式无法做到动态伸缩，难以应对突发流量。

云图核心目标是稳定高效地输出高德基础能力，为最大化利用服务器资源、提高服务稳定性和优化终端用户体验，云图服务端基于不同业务场景在异步与反应式技术架构上

做了一系列探索和实践。本文将介绍在优化升级过程中的探索与思考，希望为有类似需求的团队带来帮助。

异步与反应式

异步任务处理机制可分为两大类，第一类是线程池，第二类是事件驱动。第一类将阻塞式任务从一个线程交由另一个线程处理，避免影响当前线程，但无法解决线程膨胀问题。第二类即基于 epoll 事件驱动和多路复用，在处理 I/O 阻塞任务时可以实现服务内非阻塞响应。



Reactive 反应式的“反应”体现在：1. 对事件立即响应；2. 对失败场景立即响应；3. 对用户请求立即响应。其本质是面向事件流、基于 Reactor 模型、可保证服务在任何场景

下都能保持实时响应的架构

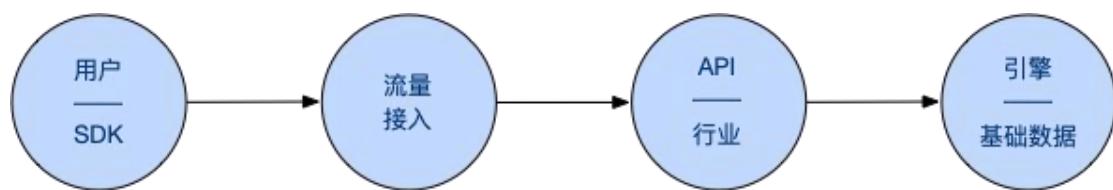
(<https://www.reactivemanifesto.org/>)。

在任务处理角度，它基于事件驱动模型（比如 Java NIO/Selector）提供任务异步处理能力，在编程模型角度，它提供面向事件流的一系列操作组合（Operator），与目前流行的函数式编程、CQRS 和 EventSourcing 相辅相成。

随着 Java 8 和 Lambda 的普及，反应式技术框架大量出现，Java 9 已经提供 Reactive Streams API 实现，Spring 也推出基于 Reactor 的 WebFlux 框架，反应式正在成为互联网服务可靠方案。

轻量业务前置

云图线上服务整体可以抽象为下列四层。



流量接入层主要由用户鉴权、流控等轻量业务组成，特点是网络 I/O 密集，无长耗时逻辑且不依赖复杂中间件。为

充分利用 Nginx 事件驱动模型和 Lua 开发效率的优势，我们选择 OpenResty lua-nginx-module 来前置这层业务。

- 优点：OpenResty 可以使用少量服务器资源支撑极高请求量。
- 缺点：Lua 语言偏小众，且部分中间件客户端缺乏对 Lua 支持，目前使用场景主要集中在流量接入层和 API 层内轻量业务场景。

从 Servlet 到去 Servlet

云图业务层以 Java 技术栈为主，其中很多 Web 服务基于同步阻塞式 Java Servlet，部分服务基于异步 Servlet。同步模型易于开发和问题追踪，但难以平稳应对高并发场景，虽然 Servlet 3.0 支持在非容器线程中处理请求，帮助尽快释放容器线程池中的线程，但由于其底层 I/O 依旧是阻塞操作（InputStream/OutputStream），在将结果写回响应流时仍会阻塞处理线程。

如果业务包含阻塞逻辑，在面对高并发场景时压力只是从容器线程池转移到业务自定义线程池。Servlet 3.1 支持读写 Socket 时不阻塞线程（NIO），但由于围绕着 HTTP 请

求响应语义模型来设计，在接口设计上并非纯粹异步，Tomcat 等 Servlet 容器无法最大程度发挥 NIO 优势。

苹果地图请求分发服务

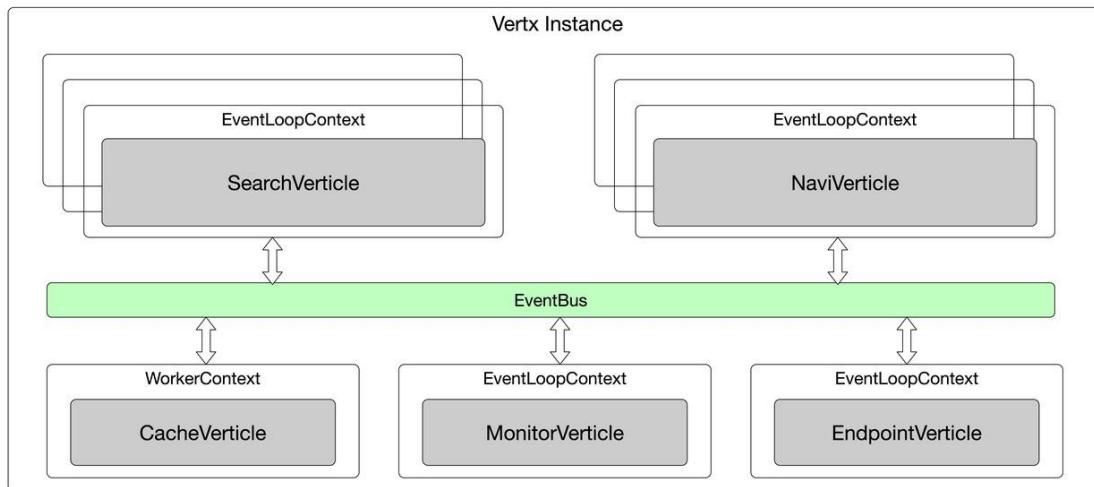
云图苹果地图请求分发服务负责对苹果地图用户请求进行处理、分发和结果聚合，以支持苹果海外搜索与导航，其对并发能力、服务耗时和稳定性有着较高要求。在技术选型时考虑到 Servlet 的约束，以及项目周期、学习曲线和实现成本等因素，最终选择 [Vert.x](#)。

在我们看来，Vert.x 的优点包含：

- 轻量、高性能。
- 易读的源码、低学习曲线。
- Actor 模型，基于 Verticle 和 EventBus 可以实现服务模块解耦与资源隔离。

由于该服务链路较短，最终没有选择 RxJava 而使用 Java 8 CompletableFuture 来实现异步服务接口定义和任务编排，

同时基于 Vert.x Verticle 封装不同业务单元，基于 EventBus 实现业务单元间消息传递。



Verticle 类似于 Akka Actor，每个 Verticle 实例会绑定一个 EventLoopContext 对象，且每个 Context 会被分配一个 EventLoop 线程，Verticle 实例内部逻辑总会由该线程执行，Vert.x 通过这一机制确保 Verticle 内部线程安全。

同时，由于 EventLoopGroup 在 Vertx 实例化时已经完成定义，不论当前有多少个 Verticle 实例，它们都会共享这个固定大小的 EventLoopGroup，通过控制不同业务单元对应的 Verticle 实例数量，可以实现业务单元线程资源隔离，达到模拟服务多租户的目的。

基于苹果请求分发服务，我们将 Vert.x 进行落地，达到凭借少量服务资源来稳定支撑苹果地图入口流量分发的目标。如果服务对 Spring 技术生态没有强需求，推荐使用 Vert.x 来做为反应式技术初体验。

从 Future 到 Reactive

通过 CompletableFuture 和 Lambda 表达式，可以快速实现轻量业务异步封装与编排，与 Callback 相比可以避免方法多层嵌套问题，但面对相对复杂业务逻辑时仍存在以下局限：

- 难以简单优雅实现多异步任务编排；
- 难以处理实时流式场景；
- 难以支持高级异常处理；
- 不支持任务延迟执行。

使用 Reactive 模型能够解决上述 Future 的局限。假设要实现下面这个需求（例子修改自 Project Reactor Reference Guide）：筛选出符合特定条件的一组产品 Id（findIds 方

法），再通过 Id 找到对应产品名称（`findProductName` 方法）与成交均价（`findAvgPrice` 方法），最终输出统计结果。

基于 Future 实现：

```
1. public CompletableFuture<List<String>> getStatisticOfFruits(){
2.     CompletableFuture<List<String>> ids = findIds("fruits");
3.
4.     CompletableFuture<List<String>> result
5.     = ids.thenComposeAsync(l -> {
6.         Stream<CompletableFuture<String>> zip = l.stream().map(i
7.             ->{
8.                 CompletableFuture<String> nameTask = findProductName(i);
9.
10.                CompletableFuture<Integer> priceTask = findAvgPrice(i);
11.
12.                return nameTask.thenCombineAsync(priceTask,
13.                    (name, price)-> "Name: " + name + " - price: " + p
14.                    rice);
15.            });
16.
17.            List<CompletableFuture<String>> combinationList
18.            = zip.collect(Collectors.toList());
19.
```

```
15.  
16.     CompletableFuture<String>[] combinationArray  
17.     = combinationList.toArray()  
18.     new CompletableFuture[combinationList.size()];  
19.  
20.     CompletableFuture<Void> allDone  
21.     = CompletableFuture.allOf(combinationArray);  
22.  
23.     return allDone.thenApply(v -> combinationList.stream()  
24.         .map(CompletableFuture::join)  
25.         .collect(Collectors.toList()));  
26. );  
27.  
28.     return result;  
29. }
```

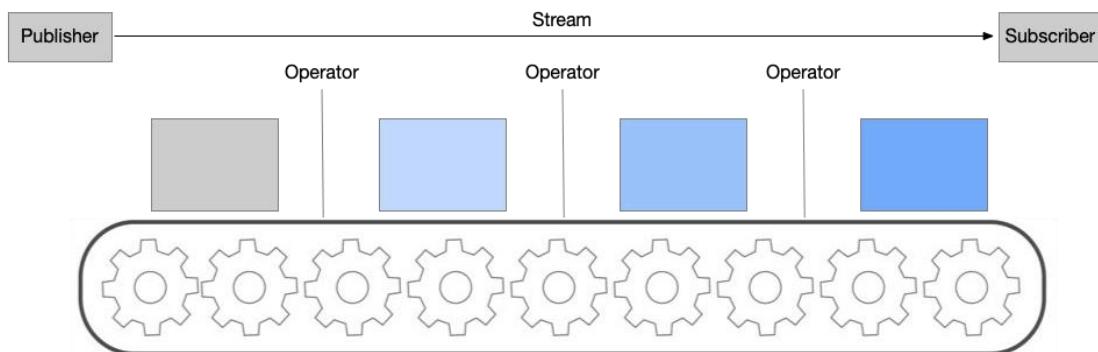
基于 Reactive (Project Reactor) 实现：

```
1. Flux<String> ids = rxFindIds("fruits");  
2. Flux<String> combinations =  
3.     ids.flatMap(id -> {  
4.         Mono<String> nameTask = rxFindProductName(id);
```

```
5.         Mono<Integer> priceTask = rxFindAvgPrice(id);  
6.         return nameTask.zipWith(priceTask,  
7.             (name, price) -> "Name " + name + " - price " + pric  
e);  
8.     });  
9.     return combinations.collectList();  
10. }
```

从上面简单对比可以看出，相比 Future，基于 Reactive 模型丰富的操作符组合

(filter/map/flatMap/zip/onErrorResume 等高阶函数) 代码清晰易读，搭配 Lamda 可以轻松实现复杂业务场景任务编排。



可以将 Reactive 系统想象成现实中的一条生产线，系统原始输入可以看作是生产线起点端的原始材料（Publisher），原始材料被生产线向下游运输（Push）。

对于原始输入每个处理步骤可以看作是生产线上运送 / 装卸 / 加工 / 检验等一系列操作（Operator/Processor），生产线加工出的产品则会投递给预定（Subscription）该批产品的用户（Subscriber），通过预定，终端消费者无需反复询问上游产品源（Pull）。

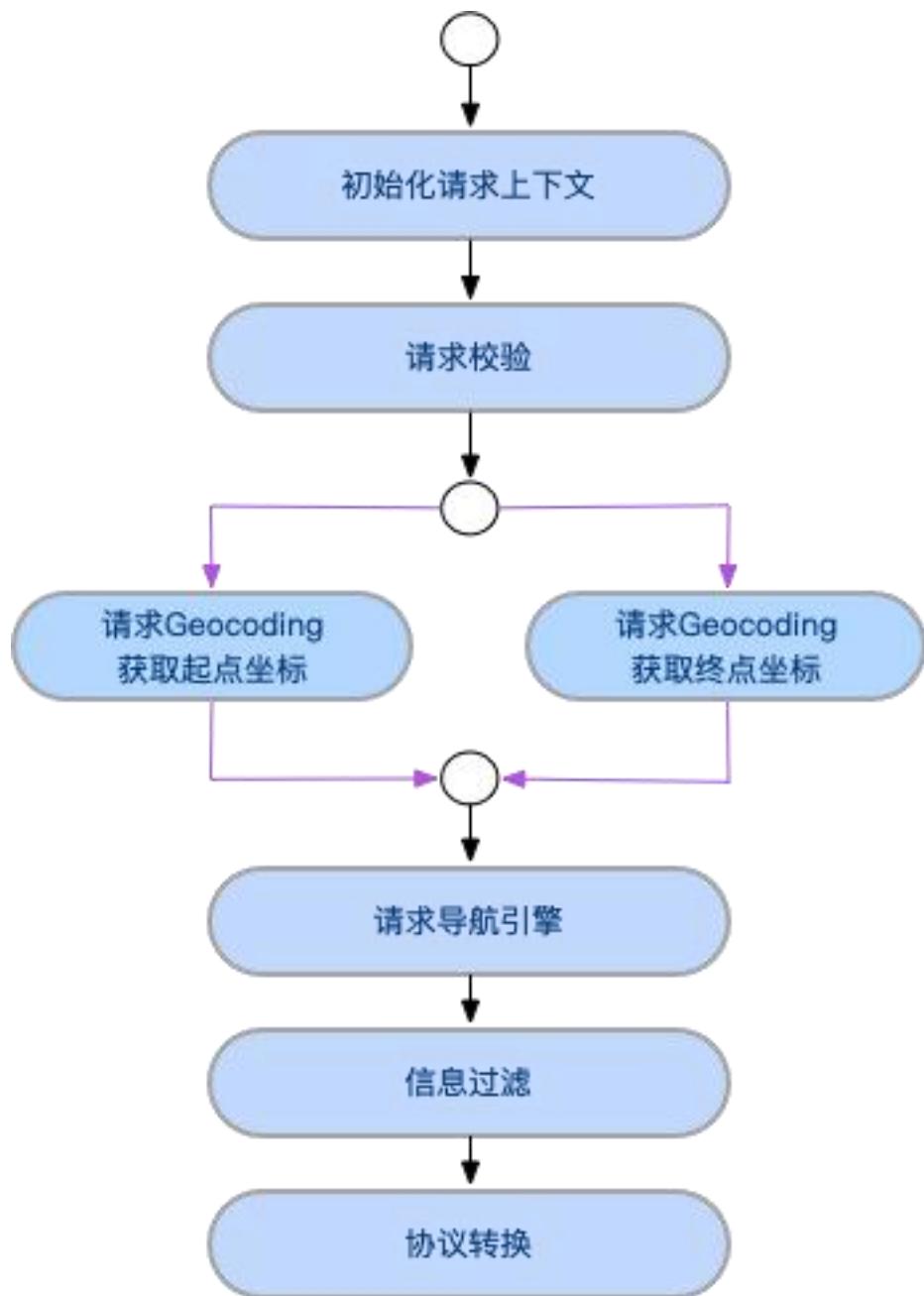
同时，生产线上每一步都可以向前反馈当前处理能力，避免上游投递物料数量超出当前加工能力（Back Pressure），从而保证生产线平稳运行。

开放平台 LBS API 服务

云图开放平台 LBS API 服务负责对外透出搜索和导航等基础 LBS 能力，实现请求校验、结果拼装、协议转换等逻辑。

随着业务快速发展，底层业务域服务粒度和复杂度不断增加，虽然提高了整体服务横向扩展能力，但给上层业务聚合增加困难，往往一个服务场景就涉及多次分布式调用。

这里用一个简单业务场景举例。在请求导航服务时可以将地址设置为起终点，如下图所示，请求处理链路包含请求校验、并发调用地理编码（Geocoding）服务获取经纬度坐标、请求导航引擎、基于业务规则信息处理与过滤、协议转换等步骤，其中涉及多数据源聚合操作。



使用传统编程模式来处理类似上述业务场景，往往涉及复杂并发管理和异常处理。

使用 Reactive 则能简化业务逻辑组合，提高代码可读性：

```
1. public Mono<RoutingResponse> processRoutingRequest(String originalRequest)
2. {
3.     return
4.         validate(originalRequest)
5.         .then(Mono
6.             .zip(getOriginCoordinate(originalRequest),
7.                 getDestinationCoordinate(originalRequest)))
8.             .flatMap((coordinates) -> callRoutingEngine(
9.                 coordinates.getT1(), coordinates.getT2())))
10.            .flatMap(this::filter)
11.            .flatMap(this::convert)
12.            .onErrorResume(this::handleException),
13. }
```

我们选择 [Project Reactor](#) 和 Spring Webflux 来做为 LBS API 服务反应式架构升级基础，从之前 Servlet 技术栈转向 Reactive 技术栈，主要工作包括：

1. 业务逻辑梳理与异步化改造，主要包含业务重构分拆、任务异步化编排与改造、中间件客户端异步化替换等，以及适配与异步化不兼容逻辑，比如使用 Reactor Context 替换 ThreadLocal 来传递上下文；

2. 服务框架重构。提高业务抽象粒度，开发者只需实现具体业务步骤，业务处理链任务编排交由框架层实现。

升级重构之后，服务性能较之前 Servlet 架构得到了明显提升。以短途驾车导航场景为例，压测结果显示在高并发请求压力下新应用 QPS 较之前提升 480%。



通过业务梳理重构与服务异步反应式改造，LBS API 服务解决了早期面临的资源利用率问题，单机可支撑 QPS 较之前有数倍提升，在系统资源正常情况下，面对流量突增场景服务耗时仍能保持平稳，同时通过任务异步编排替代多任务排队，有效降低了部分复杂业务链路服务耗时。后续规划包括：

1. 当前 API 与行业层各微服务之间缺乏对 Backpressure 的支持，导致很难彻底避免上游请求压垮下游服务的潜在

风险。目前我们在调研 RSocket 协议，希望通过全链路协议反应式改造完成服务整体对 Backpressure 的支持；

2. 行业服务层很多系统，业务复杂度远超其他两层，计划通过领域驱动设计完成业务梳理与反应式改造。

总结与展望

云图服务端通过在异步反应式技术上的尝试与探索，为不同业务场景找到了相匹配的优化方案，解决了原有同步阻塞模型资源利用率低、系统稳定性易波动的问题。但异步反应式并非银弹，相对同步模型，它对开发同学思维模式有着更高要求，且整条服务链路均需进行反应式改造。

在适合的业务场景下，反应式技术架构能够有效提升服务吞吐能力，降低业务编排复杂度，帮助构建云原生时代整体系统快速即时反应能力。希望与对反应式技术感兴趣的的同学和团队多多交流。

高德 SD 地图数据生产自动化技术的路线与实践（道路篇）

作者：喆嘉

一、背景及现状

近些年，国内道路交通及相关设施的基础建设日新月异。广大用户日常出行需求旺盛，对所使用到的电子地图产品的数据质量和现势性提出了更高的要求。传统的地图数据采集和生产过程，即通过采集设备实地采集后对采集资料进行人工处理的模式，其数据更新慢、加工成本高等问题矛盾日益突显。

高德地图凭借视觉 AI 和大数据技术优势引领地图数据产业变革，通过图像 AI 技术从采集资料中直接识别提取各类数据要素，为实现机器代替人的作业模式提供最坚实的技术基础。

高德地图通过对现实世界高频高密度的数据采集，运用图像视觉 AI 能力，在海量的采集图片库中自动检测识别并确定出各种交通标志标线标牌的内容及位置，再通过与历史资料信息的对比，能快速发现现实世界的变化信息，同时

结合强大而专业的数据融合能力，实现 100% 信息融入，从而构建出高现势性的全国基础地图。

综上，通过算法、地图工程的深度技术合作，以及与资料采集、数据生产的业务拉通，搭建一条以图像识别、位置服务、差分过滤、数据融合等为核心技术的基础地图数据生产全自动化产线，从而建立起从真实世界到地图应用终端，高效高质量的数据信息流水线生产通道。

二、自动化产线的可行性及重点

从图像物体分类和检测进展来看，图像物体的分类和检测已经有几十年的历史，涌现了一系列经典的算法。

近些年随着图像识别技术特别是深度学习技术的快速发展及 GPU 计算能力的发展，分类和检测技术有了极大的提升。

从自动化需要的大数据来看，高德地图专注地图数据制作十几年，积累了覆盖全国、丰富且准确的数据，加之每天拥有大量采集信息的汇入，这些数据都成为算法训练天然的样本池；同时一整套专业化、标准化的地图生产作业规范为数据融合打下了坚实的理论基础。

因此，从算法储备能力、数据和工艺的积累来看，自动化产线搭建具有较强的可行性，其重点围绕着以下四部分组成：

图像识别：图像识别的目标是从输入图像中解析出地图数据相关的现实信息，通过检测、识别图片中的交通标志标线标牌信息，细分其类型，并理解其中的数字和文字，以文本形式表达内容。此外，由于输入的是连续图像，单个标志标线标牌可以在多个图像上观察到，因此整合多张图像中的同一信息，并选择最合适的图像作为主图展示。

位置服务：基于低精度 GPS 和采集图像，位置服务推算出自身和场景物体的精确位置，并映射到地图数据中。其中包括图像道路理解、标志位置解析、采集轨迹匹配等核心能力。依据轨迹特性和道路连通性，建立对定位位置、角度、速度等与候选道路关系的匹配概率模型，将轨迹关联到地图数据上。通过对多张图片中场景的理解，给出图片相对于路口的相对位置，结合地图道路数据的形态，进一步确定物体的作用位置。

图像差分和语义过滤：目的是将新采集资料与已有母库中的数据进行一致性对比，自动将相同的信息进行差分与过

滤操作，留下变化的信息。两者不同之处在于前者是检测相同位置新一次采集的图片相对于历史采集图片是否有变化，从轨迹和图像本身的角度对比；后者从数据的角度看图像识别后内容，对于母库数据是否有变化，从地图语义的角度做比较。

基于位置的数据融合：图像识别的成果，结合位置服务提供的作用位置，获取到作用道路。通过抽象路口的模型，在该道路或路口做数据融合，即新增或者更新地图数据。

三、 关键技术能力

1. 图像识别

图像识别主要面临三大挑战：一方面场景多样，类型繁多。待检测对象种类繁多，如交通标志标牌、地面引导线、电子眼等。比如正常的方向信息标牌如下图：



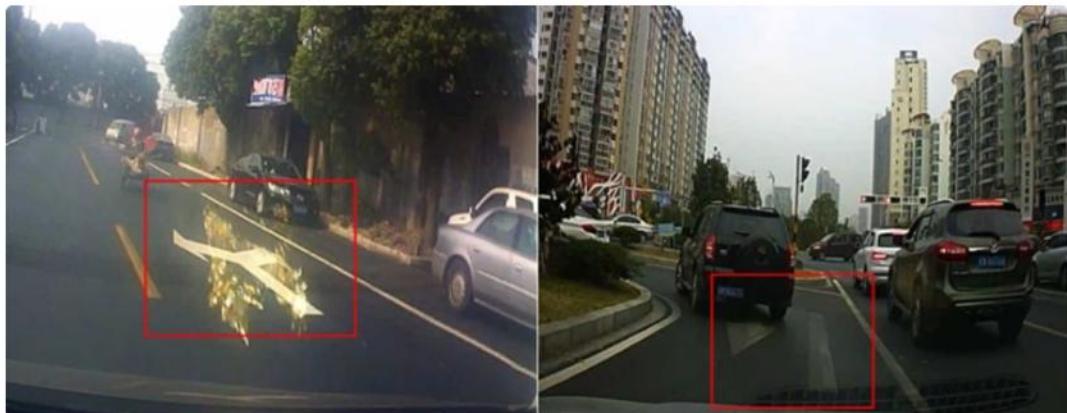
特殊的方向信息牌标牌：



而同类检测对象的样式也是繁杂的，国标通用的交通标志有几百个类型，而各地也会有一些地方特色的交通标志，所以需支持定制化检测识别。常见标牌形状多样，三角、圆形、方形、菱形、八边形等，同时颜色分布广泛，如黄色、红色、蓝色、绿色、黑色、白色等，另外，还需要排除自然场景内一些类似交通标志的标语、广告牌等，以减少对识别准确率的影响。



另一方面，在自然场景下图片质量差异巨大，其中很多图像质量偏低。再加上面临遮挡、逆光、雨雪天等极端户外场景。这些在检测环节都是要重点考虑及解决的问题。



最后，待检测对象的尺寸差异较大，大如方牌（几百个像素大小），小如电子眼、交通灯（十几个像素大小）。而小尺度检测，辨识度很差，对检测算法有比较高的要求。

综上，对于算法能力本身而言，交通标志检测实际上是一个多类型的目标检测任务，主流的方法是基于深度学习的 End2End 方案，在一个网络中同时完成检测与细分类任务。常用的 dataset 一般是 PASCAL VOC(20 类)和 COCO(90 类)等。

根据业务的实际需要，整个方案分为目标检测与精细分类两部分组成，目标检测阶段通过 Faster-RCNN 在图片中检

测所有的交通标志，该阶段要求极高的召回率和执行速度，相应在准确率方面可以放宽要求；精细分类阶段对目标检测阶段得到候选框，然后进行精细分类并滤除噪声，最终保证极高的召回率和准确率。

2. 位置服务

轨迹漂移对位置匹配地图的准确性一直都是极大的挑战，一方面平行路、高架场景，尤其是主辅路这种距离 1–2 个车道的平行路，需要很高的定位精度，常规的 GPS 定位精度在 5–10m，很难达到 80% 的主辅路识别率。另外基础地图数据本身也存在 GPS 精度问题。

通过规则及隐马尔科夫模型的学习、推理以及维特比算法等基础理论以外，合理地抵抗定位漂移问题，是轨迹匹配成功的关键。通过对轨迹形态进行学习和总结，找出其规律，建立符合其特性的概率模型，精准地表达匹配建立过程，合理地平衡匹配准确性和抗漂移能力二者之间的关系。另外，通过长轨迹的连通性和图像识别车道数或道路位置关系，以解决平行路的部分场景的问题。

而对作用道路和作用位置的确定，目前依赖于图像识别对于路口位置的识别及融合对地图数据场景的理解和判断，

例如标牌对路或路口的相对位置靠识别本身很难确定，需要融合对数据路网数据特性的理解和判断，这种判断比较复杂，人一眼就看明白了，但是机器很难用规则去描述。所以，通过路段中直行、路口中直行及拐弯等场景的分析，对比地图路段或路口的模型，来确定作业道路，根据不同属性计算作用位置。

3. 图像差分和语义过滤

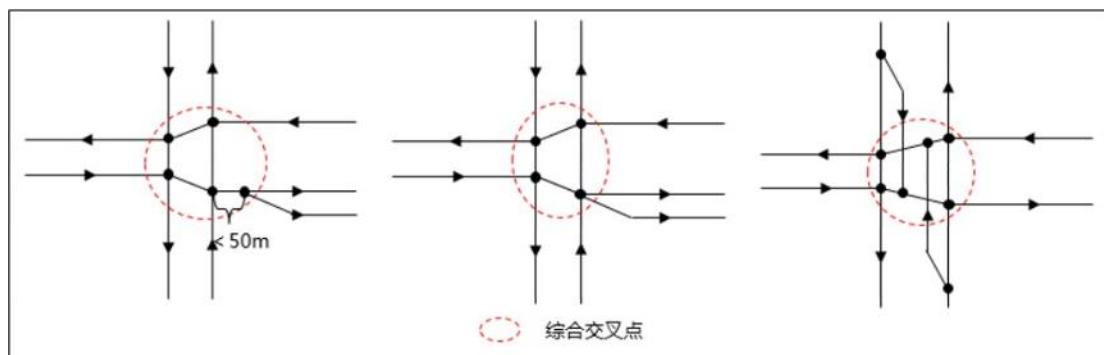
图像差分主要会面临资料对齐问题，即同一位置的多次采集资料，会受 GPS 自身精度及因卫星信号遮挡导致的漂移带来的所在道路判断偏差的影响。另外，在语义识别上，受自然环境下的环境因素，如遮挡、模糊、阴影、雨雪天气、视角变化等，会影响后续算法对图像的深层语义信息(如类型、内容等)的解析。两种因素的叠加，在多张图像和语义的一致性比对时，难度就提高了不少。

这方面，算法大幅快速提升了识别和一致性判断的准确率，以避免错误匹配对数据更新的影响。图像差分分为资料对齐和局部匹配两部分，资料对齐回答两次采集图像是否在同一位置、视角等，通过 GPS 轨迹粗筛、图像匹配等手段，判断两张图像的位置关系。局部匹配则需要回答两个物体是否为同一类型，对于有文本内容的物体，还需要检测版

式、文本的一致性。因此除引入常见的点特征匹配技术外，也使用了基于深度学习的图像匹配网络。对于文本内容部分，借助 OCR 能力完成内容的理解和解析，最终判断两次采集的内容完全一致性。

4. 基于位置的数据融合

由于现实世界的复杂性，地图生产经验积累形成了大量标准化地图数据制作规范，这些都是能合理抽象、准确表达现实世界的无形资产。即便现实路网形态千奇百怪，但都能通过模型进行抽象归类，建立不同场景下相对通用的地图数据模型，从而在其上建立沉淀大量的地图数据处理的工具类和方法，以确保数据自动化融合能力的广泛使用。



四、总结

高德 SD 基础地图数据生产自动化实现，本质上就是在基础地图数据生产过程中，引入图像 AI 技术和数据融合技术，结合多年地图数字化生产作业规范及经验，创新出一套面

向资料的自动化生产线，形成自动化解放人工持续提供高效高质量的地图数据，以解决地图供应商生产产线专业化程度高、人工成本大、作业效率低等产线问题，最终满足广大用户出行过程对电子地图产品数据现势性的需求。

招聘

阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

质量篇

高德技术评测建设之路

作者：燕鸣

前言

近几十年是互联网高速发展的时代。随着互联网行业的发展壮大，必然会出现角色的细分，从而演化出了不同的职能岗位。随着日益激烈的市场竞争，修炼内功，提升产品效果也成为了各公司发展的重要工作。产品效果如何评估？用户体验如何度量？本文试图阐述评测这一新岗位在高德的主要职责，发展进化过程，以及这一岗位所负责的产品效果评估手段与体系搭建。

当你在各搜索引擎输入评测二字时，看到的相关搜索通常有这样的：

相关搜索

评测和测评有什么区别	是叫测评还是评测	评测啥意思
手机对比评测网	华为体脂秤2pro不支持ios	家居评测网站
产品测评网站	rbk852评测	豪爵铃木en150呆子评测

99%的人还搜了

超时空评测	手机测评	手机测评网站	评测是啥意思	产品测评网站
测评和评测的区别	评测手机的软件	评测和测评	手机评测视频	
手机评测网站哪个好	笔记本评测	手机对比评测平台	大米评测的个人频道	

这些问题其实能代表大部分人对评测的了解——就是除了游戏评测、手机评测、汽车评测、生活用品评测之外，人们对评测其实不太了解。互联网公司里 Title 是评测的同学又是做什么的呢？也许大家的了解就更少了。

做了三年多的评测，在第一年经常面对的灵魂拷问就是：“你们评测是做什么的？”这种问题回答起来，基本类似于哲学的终极三问了：“你是谁？你从哪儿来？你到哪儿去？”

评测是谁？这是评测的定位问题。评测从哪儿来？这是评测的根基和起源。评测要到哪儿去？这是评测的发展目标和方向。

评测是谁？

简单地说，评测是评估产品效果的团队。希望能站在用户的角度，在上线前验证需求效果，在上线后通过对自身、用户数据和竞品的全面分析，建立起产品立体的效果评估体系，也就是评测体系。

评测从哪儿来？

要回答这个问题，其实就是——为什么要评测？

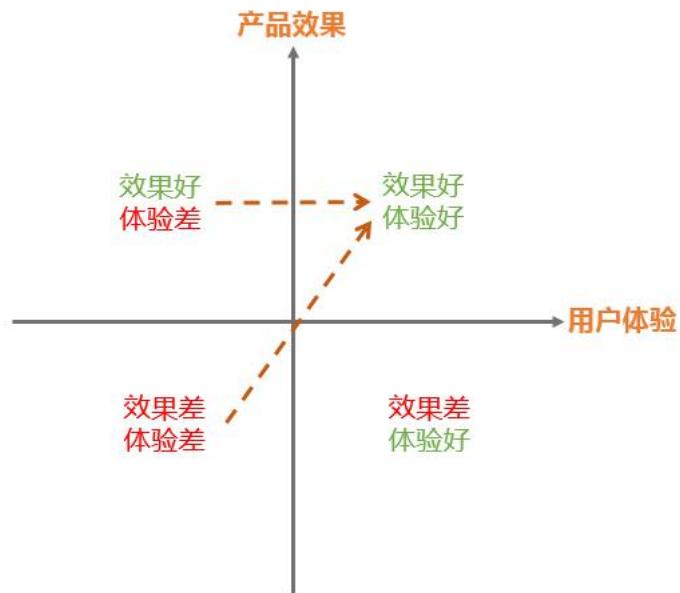
如同每个版本更新，我们都会关心性能如何一样，当上线了新的策略时，大家也会同样关心产品的效果。产品效果如何评估？策略相关的需求开发完成之后，研发实现的实际效果是否和产品经理的预期一致？实际效果又是否和用户的预期一致？

在理想情况下，这三者应该是无差异的。但我们也应该有衡量它们之间是否有差异的方式，给出效果变化是否正向的结论，以更好地保障用户的使用体验。

此外，即使上线前，所有人都一致给出了正向结论，认为需求上线后一定会给用户体验带来极大提升。真实的产品体验如何，仍然得用户说了算。比较大的修改可以通过 AB 实验的方式圈出小部分用户，快速收集用户数据，进一步对需求效果是否正向做出评价。或者直接上线，通过对行为数据及用户反馈的分析来完成线上评估。

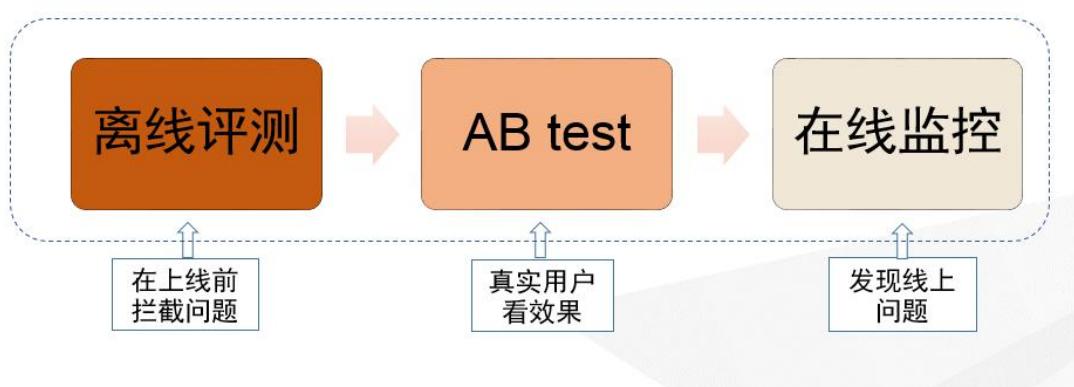
同时，要在市场上找准自己的位置，对竞品的分析必不可少。

有了这些效果评估及分析的需求，就有了评测团队。



如何进行评测

上线前的离线效果评测及分析、AB实验及分析、上线后的指标监控及问题分析、问题挖掘，竞品监控和分析是常见的评测手段。



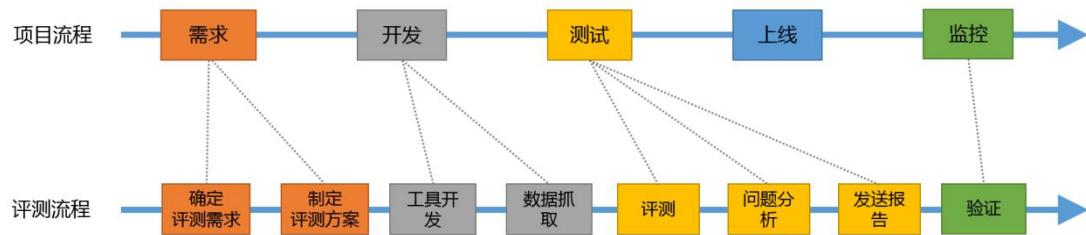
一、离线评测

上线前，针对产品的需求，评测的职责是通过各种方式分析及验证产品效果，给出是否能达到上线标准的结论，同时分析出头部问题所在。

技术评测团队成立之初，主要建设的部分有：确定合作流程、建设评测专业能力和建设评测工具。

• 合作流程

对标一个版本开发的项目流程，从需求确定到开发，到测试验证再到上线。评测从需求串讲阶段开始，明确有哪些需求涉及到效果变化。再根据变化情况制定评测方案，同时检查工具是否符合需要，如否则进入工具快速开发阶段。然后获取评测数据，进入评估验证阶段，最后发送报告，给出需求是否通过评测的结论，并对出现的问题进行总结分类。



对于评测介入的不同业务线来说，评测的流程大致相同。但由于业务不同，评测方案与方式会有很大不同。

• 评测方案

根据产品需求，明确效果修改影响范围，从而确定评测样本、评测方式和评测标准。

• 评测样本

评测样本通常会根据需求影响范围的不同，区分为随机语料和特定语料。

特定语料一般针对需求修改的特定维度、类型进行抽取，目的是保证评测任务的覆盖率。随机语料则是为了反映需求的真实影响范围。当一个评测任务需要使用特定语料时。通常建议使用特定及随机语料各一份，以同时保证足够的覆盖，同时了解真实影响范围，确保不会出现不符合预期的变化。

除真实语料外，在特定场景下也会使用自己构建的语料。通常原因为：1) 策略上线之前没有真实线上语料；2) 影响的场景太小，在真实语料中很难找到足够的 Case。

• 评测标准

评测标准通常涉及到一个概念，即真值。当某类数据在现实世界中有唯一正确答案时，即有绝对真值存在，如数据

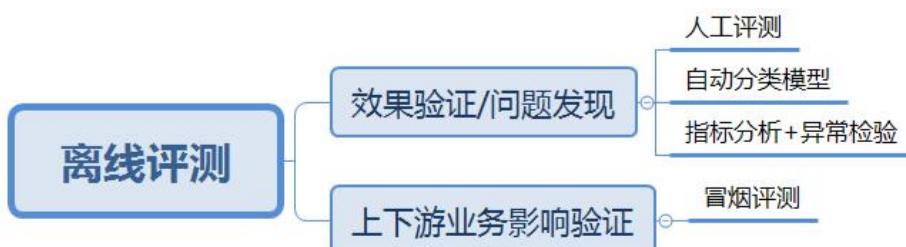
信息。因此我们对这类数据的评价标准就是是否跟真值一致。

另一类是相对真值。来源可以是用户日志。例如，当我们在判断提供给用户的预计到达时间（ETA）是否正确时，可以用用户在起终点之间的真实行驶时间作为真值和我们的预估时间进行对比。但由于单一用户的实际行驶时间受个人行驶习惯以及单次的行驶情况所影响，并不是完全准确的。因此是相对真值。在搜索等业务线，用户的点击行为，也可以成为相对真值，从而成为效果评测的标准。

是否有真值，真值是否容易获取，能否大批量自动化的获取，是在确认评测标准时需要做的判断。

• 评测方式

对应不同的评测目的，我们给出不同的离线评测方式。有真值的业务，通过真值的自动获取或者标注，可以实现自动化评测。而无真值的业务线，判断效果好坏的成本较高，通常需要进行人工评测或者半自动化评测。



人工评测，顾名思义，就是靠人力打分。各搜索公司大概是最早对自己的产品进行效果评估的，谷歌、微软、百度、苹果等，都采用了类似的方式对质量进行评价。

Google 曾经发布过长达 164 页的人工质量评估指南。百度和必应也发布过类似的文档。

苹果在介绍自己的评测体系时，也曾经专门解释过 Human Judgement metrics, why we track them?

- 可以在上线前发现版本问题。
- 人工评测的指标与定量指标紧密关联。
- 可以定义一个版本的整体质量，并可持续跟进效果变更。
- 比用户反馈更详细，更容易定位问题。

人工评测缺点不用多说，成本高、覆盖面小、效率偏低。因为它的优点，目前仍然是各公司评测体系不可缺少的一部分。与别的评测手段结合使用时，能起到很好的效果。

要保证人工评测的质量和效率，有三个关键点，一是标准，二是流程，三是工具。

标准文档，类似于操作手册，目的是降低人员培训成本，并在一些较难判断的 Case 上，尽量减少大家认知上的差异。所以标准文档应该越傻瓜越好。

定义明确、所有的特殊和例外场景都有示例、在实践中反复检验，并且保持更新频率。文档更新应该有专人负责，并且明确更新周期，同时将更新点同步到所有评估人员。

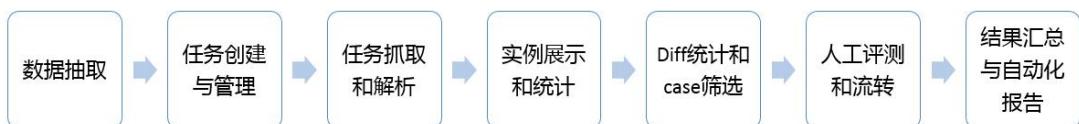
人工操作错误在所难免，没人能达到百分百的准确。同时需要人工评测的评测对象，通常本身没有客观统一的确定答案，因此大家难免在判断上有差异。这些问题都需要从流程上加以保障。如同一 Case 必须多人标注，仅保留一致率较高的 Case，否则便丢弃。或者采用初审复审制，经验较少的人员进行初审，高级人员进行复审。

盲审，这种方式通常在对比时使用，去掉新旧版或者左右版的标识，并且让结果随机出现，从而保证评测人员的客观性，不受主观因素影响。

人工评测中的人，通常也有两种身份。一种是普通用户，一种是专家。专家评测需要站在更专业的视角，结合自己对业务的理解和经验才能得出结论。另一种则是普通用户也能站在自己的视角给出效果好坏。后一种可以进行众测，达到较大范围的收取用户体验与反馈，同时获得一些真实数据支持迭代优化的效果。地图导航由于其专业性，通常需要进行专家评测。

• 评测工具

评测工具是评测效率和质量的保证。核心功能包括，数据仓库、任务管理、任务的抓取和解析，diff统计和筛选，任务实例的展示、评测、流转，抽样、分配，结果管理、自动化报告。



通用流程之外的任务类型、打分方式、Case形态都可以自己定义。由于大部分是对比类的评测任务，如何做 diff 也非常关键，尽量把业务关注的各个重点都进行 diff 差分。以便快速了解迭代效果影响面，以及快速定位问题。专家型评测在分析和定位问题时，还需要辅助分析或者判断的数据及工具。工具的接入常常能极大地提高评测效率。

人工评测能够良好运行，有了一定的评测经验积累和业务了解之后，开始进行半自动化和自动化的评测建设。

方式包括定义指标波动阈值和极端 Case 的冒烟评测，及模拟人工评测的自动打分模型。

自动打分模型通过学习人工评测的特征，自动给出 GSB 的评分，统计评分结果，对评测任务的效果进行初步判定。目前可以成为辅助判断的参考手段。

■ 自动评测-GSB打分 参考指标

维度	略好	一致	略差
case数	683	80899	3575
占比	0.79%	94.15%	4.16%

冒烟评测先定义出业务核心关注的场景和维度，设定指标。并根据既往评测经验计算出可接受的波动阈值。另外定义出在效果变化上不可接受的恶劣 Case。对于部分需要快速验证上线的实验，可以实现缩短评测周期，并保证无异常的效果。在部分业务线借此实现了自动发布上线的过程。

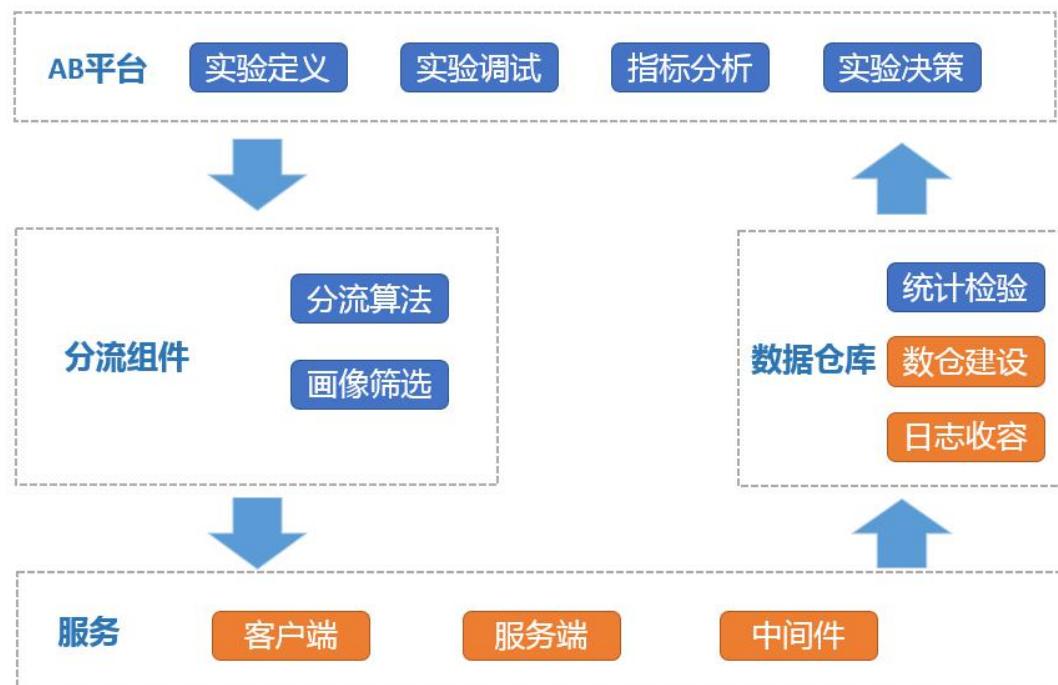
指标分析+异常检验的评测方式，是目前无真值业务线离线评测的最佳实践方式之一。通过定义整体指标、场景指

标、异常指标，形成较为全面的指标体系。观察新版本在不同情况下的指标整体波动和分布变化。在过程中筛选出异常 Case 再进行人工校验。最终根据指标变化情况和人工检验结果给出结论。如无异常则可以快速通过评测。

最后，路测是导航产品效果验证的终极手段。从用户视角体验并评估全过程。虽然成本高，效率低，但必不可少，与其他手段并用，也是上线前效果保障的方式之一。

二、AB 实验

部分需求尤其是模型调优。需要上线观察效果。因此在快速通过离线评测之后，进入 AB 阶段进行效果评估。



AB 的核心链路是分流打标、指标观测和实验结论产出。关键点是实验的科学性。效果评估链路中，AB 能力的具备不难，但 AB 实验的建设是个长期的过程，在此不赘述。

三、线上验证

经过离线验证、AB 实验，证明效果都是正向之后，需求通常全量上线，上线之后的效果如何，需要对线上指标进行分析，并观察用户反馈情况，了解是否在核心指标上有预期的收益，以及观察指标是否有异常变化。

一个产品的核心是满足用户需求，创造用户价值。因此是否满足了用户需求，用户满意度如何，产品在市场上的情况怎么样，必然是一个产品创造者要长期关注和回答的问题。以上便是我们试图去回答这些问题的方式。

结语

评测的建设过程，其实也是产品效果评估立体体系的搭建过程。这个职责在任何一个互联网公司都需要有人承担。不过角色也许是测试、也许是产品、也许是运营。

在高德，之所以把这个角色独立出来，源于对用户体验和产品效果的重视。这一体系当然远远未臻完美，还在不断

搭建进化的过程中，我们始终希望能够通过不断努力，让出行更美好。

招聘

阿里巴巴高德地图质量部热招测试开发高级工程师/专家，Java、C++高级工程师/专家，算法岗位。职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德全链路压测——语料智能化演进之路

作者：睿光

背景

高德地图作为日活过亿的国民级出行生活服务平台，承载着海量用户服务的是后台的超大规模集群。从用户角度，如果出问题，影响会很大。3机房异地部署造成线上环境复杂，链路复杂。在这样的条件下，如何避免因故障造成用户的伤害，以及在复杂链路条件下做好容量规划，做好灾备，并在第一时间发现问题，通过流量控制和预案演练做应急响应就显得至关重要，而所有的工作都不能等到事情发生之后才做，我们需要有一种验证手段来做好提前性能摸底，这就是全链路压测，让真实的流量提前到来。

全链路压测作为线上服务稳定性保障的重要手段，对高德来说也是非常重要的。高德全链路压测平台TestPG从无到有，在经历过常态化压测后，已基本可以保障高德的所有全链路压测和日常压测，达到了平台初期快速、准确压测和全链路压测的目标。而语料

生产(流量处理)作为全链路压测的重要环节，本文将对此做重点介绍。

一次全链路压测可简单总结为 3 步：压测前的流量处理（也就是生产语料）、压测中确定压力模型启动压测、压测后的结果分析与问题定位。每次全链路压测，压测前的流量处理是整个压测过程中最耗时的一环。

过去往往由运维采集日志交给测试同学写脚本处理，耗时相当严重、成本巨大，且存在请求过期等诸多问题。基于这些问题，高德全链路压测平台 TestPG 前期已规范了高德压测的语料格式，统一了高德压测的流量处理流程。但随着高德全链路压测的演进，后续面临两个主要问题：

- **语料生产流程缺乏统一管控。** 虽然平台前期已规范了语料格式，但各业务只是按照语料规范处理流量，生产流程缺乏统一、标准化管控，导致语料生产成本依然很大。尤其对于全链路压测来说，语料准备是最耗时的环节。
- **接口级别的精准控压无法满足需求。** 高德作为国民级的出行应用，流量受天气、地形、节假日的

影响比较大。比如拿驾车导航来说，日常大多都是短距离的驾车导航，而国庆、春节大多都是长距离的驾车导航，而长距离的驾车导航对后端算力的要求是非线性增加的，甚至是成倍增加。但长短距离的驾车导航对压测平台来说是同一个接口，而平台目前的精准控压只能做到接口级别，无法模拟接口特征级别的压测。

基于以上两大问题，高德全链路压测团队设立语料智能化专项，重点解决以上相关问题。

解题思路和路径

引流标准化

高德的全链路压测彼时已基本拉通大多业务，但还属于一个演进阶段。对于语料处理，主要由各业务自行处理后用来压测，语料处理的来源缺乏统一性，日志、ODPS、流量等处理来源司空见惯。对于语料生产流程的统一管控，我们首先想到的是统一语料处理来源，必须选择一个低成本、高效率的方式作为语料生产的输入，而流量录制的方式就很切合。经过调研，发现高德其他业务场景对流量录制也有很大的需求。但高

德过去的流量录制方式并不统一，各业务线自行拷贝流量经常会引起线上机器不稳定等问题。所以首先要做的是统一高德的流量录制，标准化引流。

语料生产平台化

要统一管控语料的生产流程，上面已经统一了语料生产的输入，接下来就是如何把流量转化为符合平台规范的语料，把整个转化流程平台化。但对于高德业务来说，各个业务都有其自身的特点，如果让平台为每个业务提供定制化的处理逻辑成本巨大，再加上平台对各个业务并不是特别熟悉，也很容易出错。

而整个语料处理过程也存在一些通用的处理逻辑，所以我们必须提供一种既支持各业务定制化需求，又可以满足平台通用处理逻辑的方案。我们最终选择通过 Flink 来完成整个流量处理逻辑。

引流已经标准化，业务方只需查看流量的格式内容，编写 Flink 的 UDF(用户自定义函数)，处理自身业务定制化的需求即可，而后续通用的语料存储等逻辑可通过 Flink 的 sink 插件来完成。这样既可以提供通用处

理逻辑，又给业务的特殊需求提供了支持，扩展性良好。

语料智能化

上面已经提到高德这种国民级出行应用受各种环境影响比较大，如何达到接口特征级别的精准控压，是当时面临的又一大难题。平台已具备接口级别的精准控压，只需把接口按照特征分类，提供真实流量的特征分布即可。但流量的特征分布是实时变化的，如何提供符合流量高峰的特征分布是语料智能化的最终目标。

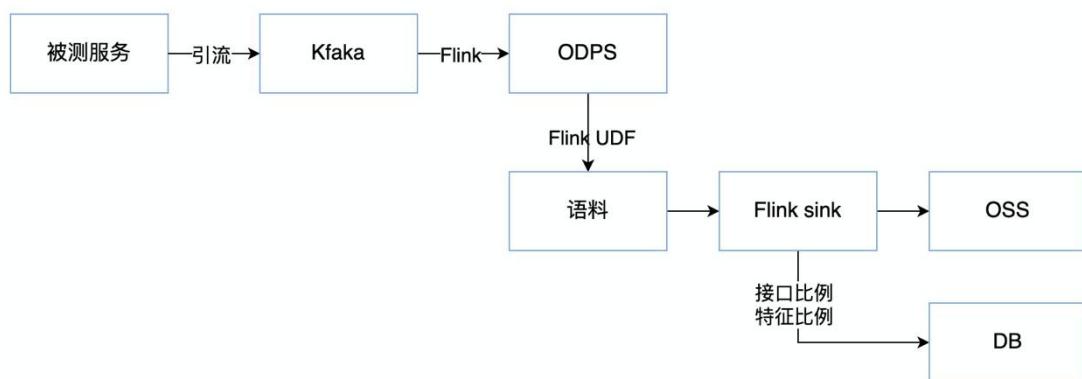
要实现语料智能化需要经历 3 个阶段。第一阶段是**流量特征统计**。我们需要明确影响流量变化的因素，体现到流量上就是具体的参数分布，具体有哪些参数会随着外界环境的变化而变化。当然这块高德大多业务线都有一些粗略的分析结果，前期可以直接采用，后期就需要有更细粒度的特征分析。

第二阶段是**流量特征提取**。有了具体的特征参数后，就需要对特征参数进行提取统计，后续可用来做智能预测。但特征参数的提取到底应该如何去做呢？

经过综合分析发现放到语料生产的环节最合适。引流拷贝流量，语料生产环节用来处理流量，在这个环节提取特征参数再好不过了。而整个语料生产扩展性良好，对用户的特殊需求通过 UDF 完成，整个流量特征提取刚好可以在通用逻辑里面完成。

第三阶段就是智能预测与机器学习。有了特征参数的统计数据，就可以借助往年高德地图国庆或春节的流量特征，加上今年随着业务的流量变化趋势，智能预测出符合今年国庆或春节流量特征的数据，做到接口特征级别的精准压测，做到真正意义上的全链路压测，为高德地图服务的稳定性保驾护航。后续也可以借助机器学习自动发现影响流量变化的特征参数，自动采集分析，做到真正意义的语料智能化。

整体方案



整个引流工作将由开发的统一引流平台来完成，引流平台通过引流插件把流量缓存到 Kafka，最终落盘到 ODPS。而整个语料生产服务直接对接引流平台，处理来自 ODPS 的流量即可。

语料生产服务的整体处理过程都由 Flink 来完成。用户只需编写 Flink 的 UDF 来完成自己业务线定制化的需求即可。而且整个 Flink 的 UDF 支持多参数传递，用户可灵活编写 UDF，在执行过程中动态传递相关参数，解决请求过期等问题。

Flink sink 是由平台开发的一个 Flink 源表解析插件，主要包括流量的特征分析与提取，以及把生产好的语料按照接口命名写入 OSS 供平台压测使用。目前流量的特征由各业务线自己提供，通过在平台添加完成。

Flink sink 在执行过程中调用平台开放 API 获取特征数据进行采集，最终上报给平台，平台后续再根据这些数据进行机器学习，智能预测出符合流量高峰的流量特征，供全链路压测使用。

核心功能介绍

Iflow引流平台

基于上面的问题分析，高德工程效率团队积极迎接挑战，短短几个月开发了Iflow引流平台，对高德的引流进行了统一管控，具体如下图所示：

The screenshot shows the Iflow Task Management interface. At the top, there are tabs for '新增' (Add), '我的' (My), and '全部' (All). A search bar is labeled '请输入任务ID/名称' (Enter task ID/name) with a magnifying glass icon. Below the search bar is a refresh button. The main area is titled '任务列表' (Task List) and contains a table with the following data:

ID	名称	类型	创建时间	创建者	状态	操作
1	[REDACTED]	gor	2020-07-03 17:39:22	[REDACTED]	用户终止	提交审批 执行 停止 ...
5	[REDACTED] -01 [REDACTED] 验证	gor	2020-08-13 14:09:19	[REDACTED]	引流完成	提交审批 执行 停止 ...
4	[REDACTED]	gor	2020-08-04 16:33:57	[REDACTED]	引流完成	提交审批 执行 停止 ...
15	[REDACTED] 上线	gor	2020-10-21 10:46:55	[REDACTED]	引流完成	提交审批 执行 停止 ...
10	[REDACTED]	gor	2020-10-14 15:07:57	[REDACTED]	引流中	提交审批 执行 停止 ...
14	[REDACTED]	gor	2020-10-20 19:47:17	[REDACTED]	审批未通过	提交审批 执行 停止 ...

Iflow 引流平台以任务的方式对高德的引流进行管理。目前采用引流插件的方式进行流量拷贝(后续将支持更多引流方式)，流量通过 Kafka 缓存，最终写入 ODPS 供大家使用。用户只需要从 ODPS 提取需要的数据即可。而启动引流需要相关负责人审批，周知到关联业务，有效的降低了引流引起事故后排查的成本。

TestPG 语料智能化

高德全链路压测平台语料智能化主要由 3 个模块组成：业务线管理、压测名单管理和接口比例管理。业务线

管理主要用来管理高德各个链路的相关数据，包括关联引流任务、启动引流、引流记录、语料路径、压测header管理和触发语料生产等功能。一条业务线就是一条压测链路，从引流到语料生产以及语料特征分析等都是在业务线维度完成的。

具体如下图所示：

ID	业务线名称	描述	状态	应用名	修改时间	操作
7	【测试】	【测试】	成功	【测试】-gate【测试】	2020/10/28 15:10:09	编辑 删除 更多
6	【测试】	【测试】	成功	【测试】-word【测试】	2020/10/28 14:50:09	编辑 删除 更多
5	【测试】	【测试】	跳过	【测试】-test【测试】	2020/07/17 15:49:49	编辑 删除 更多
4	【测试】	【测试】	跳过	【测试】-test【测试】	2020/07/09 16:32:29	编辑 删除 更多
3	【测试】	【测试】	跳过	【测试】-test【测试】	2020/06/24 15:58:59	编辑 删除 更多
2	【测试】	【测试】	跳过	【测试】-test【测试】	2020/06/24 11:36:08	编辑 删除 更多
1	【测试】	【测试】	跳过	【测试】-platform【测试】	2020/06/24 11:35:47	编辑 删除 更多

功能介绍：

- **关联引流任务**：主要完成和引流平台任务的关联以及配置相关的参数。
- **启动引流任务**：启动引流平台任务，在引流结束后会自动触发语料生产，通过执行用户编写的

Flink UDF 和平台开发的 Flink 插件，完成语料的生产和特征参数的提取。

- 语料路径：在每次启动引流触发语料生产后平台会自动生成语料路径，用户可在创建语料的时候自主选择。
- 压测 header 管理：每条业务线都有自身的业务特点，在 header 上的体现也不同，这里主要用来管理压测 http 服务发送的 header 内容。
- 触发语料生产：语料生产有 2 条途径，一是关联好引流任务启动引流后会自动触发语料生产，包括特征参数提取等一系列的操作；二是在引流成功后，用户可能对 UDF 等参数有所修改，也可以通过此按钮来触发语料生产。

压测名单管理主要用来管理压测的接口。一个公司开始做压测，业务肯定是需要跟着去适配的，随之而来的就是业务改造，这是一个漫长的过程。

为了方便管理，高德全链路压测平台对高德这边的接口进行统一管理。具体如下图所示：

The screenshot shows the 'TESTPG' application interface. On the left, there's a sidebar with various management options like '任务' (Tasks), '场景' (Scenarios), '语料' (Corpora), '集群管理' (Cluster Management), '报告管理' (Report Management), 'AccessKey', '压测熔断事件' (Pressure Test Breakthrough Events), '配置管理' (Configuration Management) which is expanded to show '业务线管理' (Business Line Management), '压测名单管理' (Pressure Test List Management) which is selected and highlighted in blue, and '接口比例管理' (Interface Proportion Management). At the bottom of the sidebar, there's a link to 'jmeter4.0下载' (Download jmeter4.0).

The main content area is titled '生产环境 v 生产环境' (Production Environment v Production Environment). It shows a table for '配置管理 / 压测名单管理' (Configuration Management / Pressure Test List Management). The table has columns: 'ID', '接口path' (Interface Path), '路径' (Path), '接口名称' (Interface Name), '类型' (Type), '业务线' (Business Line), '负责人' (Responsible Person), '状态' (Status), and '操作' (Operations). There are four entries in the table:

ID	接口path	路径	接口名称	类型	业务线	负责人	状态	操作
1275	/v2/test/api/navi/line/hand/service-en	高德地图服务端	高德地图服务端	压测接口	amaps	高德	已完成	编辑 删除
1274	/v2/test/amap-navigation/card-service-route-plan/	高德地图导航卡服务端	高德地图导航卡服务端	压测接口	amaps	高德	已完成	编辑 删除
1230	/v2/test/api/navi/line/hand/service-en	高德地图服务端	高德地图服务端	压测接口	amaps	高德	已完成	编辑 删除
1227	/ws/gis/line/hand/service-en	POI聚合接口	POI聚合接口	压测接口	amaps	高德	已完成	编辑 删除

压测名单是在引流过程中自动上报的，引流只要发现未在压测名单的接口就会自动上报压测平台，平台根据关联应用去关联对应的负责人，并推动确认。

如果可压测就确认为压测名单，下次语料生产作为白名单正常引流。如果不能压测就区分为免压接口或待跟进接口。待跟进接口平台后续会以消息通知的形式推动业务线改造，最终达到真正意义的接口覆盖全、链路覆盖全的全链路压测。

接口比例管理前期主要是用来管理 BI 提供的、以及每次全链路压测调整的比较贴近真实情况的接口比例数据，作为后续全链路压测的一个参考。后期将通过语

料生产提取流量特征的统计数据，智能分析预测出符合真实情况的流量比例，供全链路压测直接使用，具体如下图所示：

ID	path	qps	比例	操作
1209	/watertransfer/navigation/euwww.../...	1.000000	25.67210000	添加特征
1210	/...	1.000000	11.25720000	添加特征
1213	...	1.000000	6.41800000	添加特征
1211	/ws/feer/auth...	1.000000	5.91000000	添加特征
1212	/www/amap/guoruan...	1.000000	3.80130000	添加特征
1214	...	1.000000	2.87860000	添加特征
1220	/ws/mapping/www/infilit...	1.000000	2.79270000	添加特征

平台优势

语料平台化生产

整个语料生产对接了引流平台，并通过 Flink 来完成。既支持了业务方定制化的需求，也支持平台通用化的处理逻辑，扩展性良好。通用逻辑通过 Flink sink 来实现，并加入了流量特征提取等功能，推动了语料智能化的顺利进行。用户只需要学习 Flink 完成 UDF 的编写，然后在平台完成相关配置即可。很大程度上提高了语料生产的效率和质量，是语料从格式标准化向生产流程标准化的一大飞跃。

语料智能化

平台在整个语料生产的过程中，通过 Flink 插件完成了特征参数的统计汇总。目前用户只需在平台完成相关特征的配置，平台在语料生产过程中就会分析特征并统计汇总。有了特征参数的统计数据，将有助于平台后续的智能分析与预测，达到接口特征级别的精准控压，最终达到完全意义的全链路压测。

平台目前已经完成了语料的自动生产，并加入了语料智能化相关的工作。整个压测名单也是通过引流自动上报，后续将通过消息通知自动拉通业务线改造解决。接口比例管理模块也已支持接口比例的展示和调整，最终通过语料特征的智能预测，即可生产出符合流量高峰真实特征的语料。这些都将推动高德全链路压测智能化的演进。

未来展望

高德全链路压测平台语料智能化发展已经有一段时间了，通过大家的不懈努力，语料智能化已完成了语料的自动生产，以及特征参数的汇总和提取，为后续智能化奠定了基础。未来平台将通过机器学习的方式分析学习采集到的特征数据，根据往年流量高峰的特征

情况，加今年流量的变化趋势预测出符合今年流量高峰的特征情况，做到接口特征级别的精准控压，完全模拟真实流量压测达到真正意义的全链路压测。

此外，平台将会借助机器学习自动分析发现影响流量变化的参数，自动提取分析，提高语料生产的准确性。

平台也会有置信度评估系统，分别对比真实的流量特征和预测的流量特征，分析产生误差的原因，进一步提高预测的精准度，做到完全真实的流量生产。后续配合平台的精准压测、压力模型和监控等功能达到真正意义的无人化、智能化的全链路压测。

招聘

阿里巴巴高德地图质量部热招测试开发高级工程师/专家，Java、C++高级工程师/专家，算法岗位。职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

高德全链路压测——精准控压的建设实践

作者：南照

导读

作为国民级出行生活服务平台，高德服务的稳定性不论是平时还是节假日都是至关重要的，服务稳定性一旦出问题，可能影响千万级甚至上亿用户。春节、十一等节假日激增的用户使用量，给高德整体服务的稳定性带来了不小的挑战。每年在大型节假日前我们都会做整体服务的全链路压测。通过常态化全链路压测项目的推进，已具备了月度级别的常态化全链路压测能力，把战前演练提到日常，持续推进稳定性保障建设。

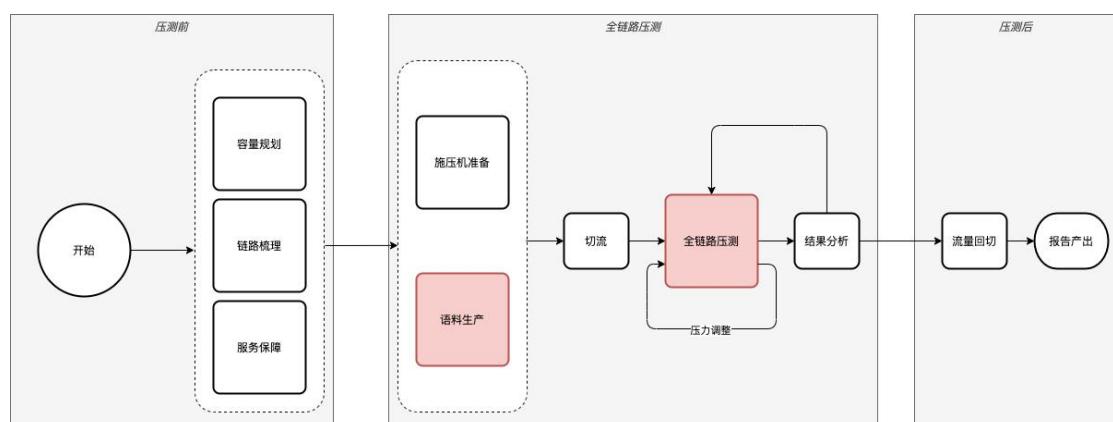
TestPG 压测平台 2018 年 9 月启动，在 2019 年春节第一次支撑高德全链路压测任务，当时前后花了近 2 周的时间才完成 3 个机房的压测任务。后来成立了常态化全链路压测项目，通过对流程的优化以及压测平台技术能力的升级，现在已经具备了 1 天内完成全国 3 个机房全链路压测的能力。大型节假日的全链路压测周期缩短到了 3 天。

全链路压测的常态化不仅对高德整体服务的稳定性建设起到了推动作用，而且也推动了流程上的优化以及压测平台在技术能力上的改进提升。

具体而言，我们主要从压测前的语料准备和压测过程中的压力调控两个方面入手，通过语料平台化生产，规范语料生成流程，对语料生产提效；通过压测过程中对发压能力的精准调控，使我们能够灵活调整压力模型，从而使其更加接近于线上真实情况，让全链路压测过程平滑流畅，效率提升。本文会重点介绍 TestPG 压测平台在发压能力精准调控方面的建设实践。

压力调控的两个主要问题

下图是现阶段高德全链路压测的一个大致流程



高德全链路压测涉及到接入层的上百个接口，每个压测接口的压测流量需要在链路梳理阶段由运维和相关

同学事先预估给定。压测前会在压测场景里对每个接口的流量进行分配，通过流量占比事先配置好，这是我们全链路压测的压力模型。压测启动之后，会逐步加压，直到整体压测流量到达预估值，在整个过程中，可能会对事先配置的压力模型不断做出调整，最终使压测流量模型符合预期的线上流量模型。

高德全链路压测涉及的链路比较长，压测流量流经接入层，服务层最终到达引擎层，有可能出现实际到达引擎的流量与预估流量之间存在差异。这时候会先停止压测，更新压测场景里的接口压测流量占比配置，然后再启动压测。

在高峰期，压测集群中可能有百十台施压机同时施压，停止压测，更改压测场景里的接口流量占比配置，再启动压测，然后再一次逐步加压，这样一套流程会非常耗时而且效率低下，有时只为了更新一两个接口的流量占比配置，不得不把前面的步骤再走一次，导致过多耗费时间精力。不断起停也使整个过程不连贯，无端拉长了全链路压测周期。

再者，在全链路压测过程中，整体压力是逐步增加的，多轮加压，每一轮加压后都会监控服务的各项指标来

决定是否进一步增加压力。TestPG 压测平台采用分布式集群施压模式，增压是通过往压测集群里调度新的施压机实现的，这样带来的问题就是，增加压测流量的时候，需要相关人员根据压测时候的单机施压能力大致估算出需要增加多少机器。由于增减的压力 = 单机施压能力 * n 台施压机，这样的压力调控粒度也是比较粗糙的。比如要加压 1w qps，单机的施压能力是 3k qps，那么需要增加 3 台或 4 台施压机，那么实际增加的压力为 9k 或者 1.2w。

最后，也是最重要的，高德业务系统对接口特征参数（对服务的功能/性能有重大影响的接口参数）尤为敏感，比如长短距离导航对后台服务的算力和资源消耗都是不同，这就提出了更高的要求，我们需要有能力在全链路压测中对接口的压力调控精确到特征参数级别。

简单总结一下，在压测过程中，针对压力调控，我们面临两个主要问题，一是压力调控效率低下，二是无法做到细粒度的精准调控。

实现路径

全链路压测是以真实流量提前对系统进行验证。只有以接近于线上真实流量的压力模型对系统进行压测，才更能发现可能隐藏的稳定性问题，压测才能更有价值。由于我们的压力模型是预估给出的，难免会与实际服务预期的流量存在差异。所以压测过程中通过对差异流量做调整，最终使压测流量模型符合预期的线上流量模型。

既然现阶段压测过程中，压力调整在所难免，并且由于其效率低下，已经影响到了全链路压测的顺利进行，那么就可以以此为抓手，在压力调控能力上做技术改进。

首先是解决压力调整效率低下问题，保障全链路压测的流畅进行，提升效率。然后结合语料智能化项目，实现精准压测，使压测流量（压力模型+压测场景）更加接近于线上真实流量。未来可以进一步探索，提供丰富的压力模型以更好地支撑场景化压测的诉求。

接下来会介绍 TestPG 压测平台，在施压能力精准调控建设上的技术方案和落地成果。

技术方案

TestPG 压测平台采用分布式集群施压模式，多台施压机构成一个压测集群。平台的整个架构是典型的 master-slaver 结构，压测的时候 master 调度 slaver 进行施压。这样的架构其实非常适合我们实现压力动态调控的一个自动控制系统。



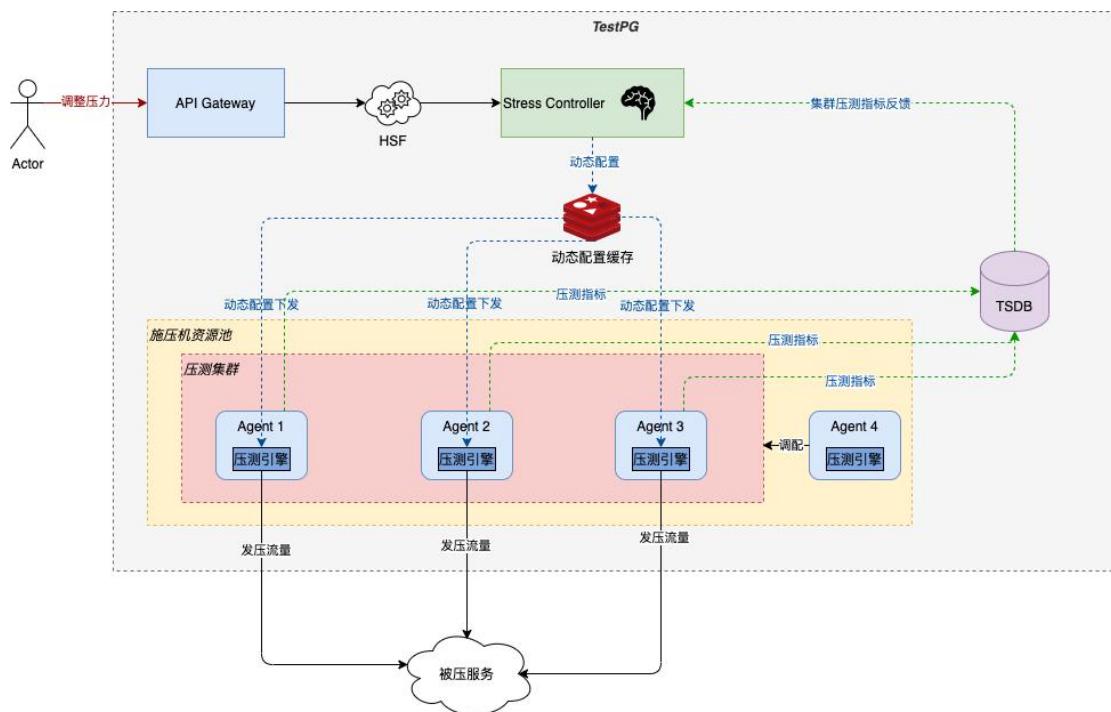
上图是压力调控系统的一个抽象示意图。

压力调控中心是压力调控系统的大脑。其主要作用是根据压力调控策略向压测集群中的施压机下发压力调控指令，并且能够根据调控反馈数据，决定进一步的调控策略。

压测集群作为压力调控对象，接收压力调控中心下发的调控指令，并实施具体的压力调控动作。

压测集群与压力调控中心之间存在一个反馈渠道，这样压力调控中心就可以知道调控的效果，并根据反馈数据，对下一步的调控进行决策。

落地架构



以上是 TestPG 压测平台精准控压建设的一个落地架构示意图。

API Gateway 模块承接用户压力调控指令，并把压力调控指令转发到 Stress Controller(压力调控中心)。

Stress Controller 是压力调控的大脑，会根据压力调控策略向压测集群中的施压机下发调控指令，并根据反馈数据，决定下一步调控策略。

TestPG 基于 Redis 实现了动态配置缓存。压力调控指令通过动态配置缓存下发到压测集群，更具体来说是下发到压测集群中的每台施压机上，目前 TestPG 采用两种方式实现动态配置的下发，分别是施压机主动拉取和通过发布订阅模式进行实时推送。

压力调控指令下达到施压机后，压测引擎运行实例会加载压力调控配置，实时调整压力。

每个压测引擎都会实时上报自己到压测指标（比如 qps, rt 等）和施压机的性能指标（比如 cpu 占用率，load 率等）到 TSDB 时序数据库，TSDB 建立了压测集群和压力调控中心之间的反馈渠道。Stress Controller 定期查询 TSDB，获取每台施压机以及整个压测集群的压测指标作为反馈数据，根据这些反馈数据判定单机压力

调控成功与否，整个压测集群压力调控成功与否，并且会根据反馈数据决策是否进行进一步的调控。

基于以上架构，TestPG 压测平台在精准控压建设上，已实现了集群和接口两个级别的精准调控。

集群压力调控

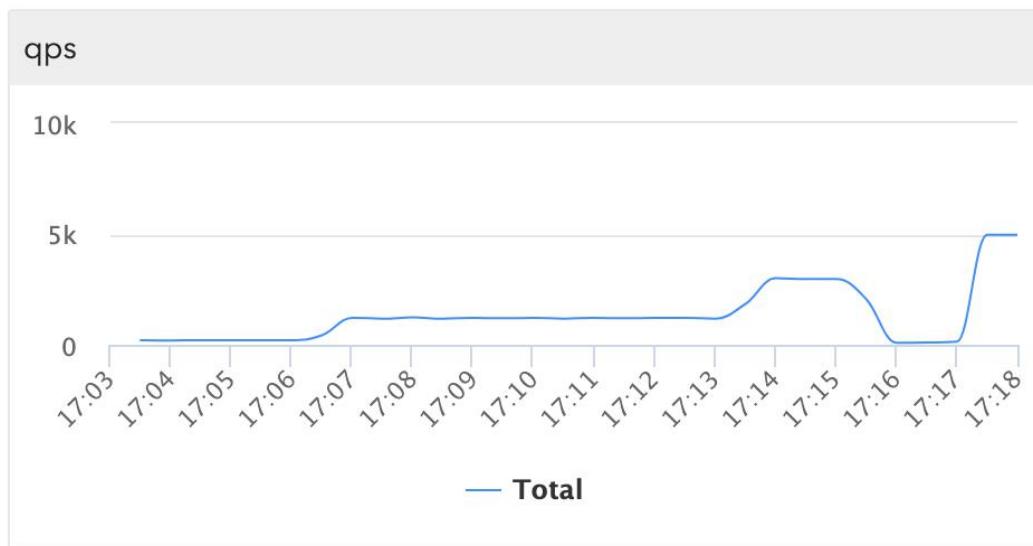
通过集群压力精准调控，在压测过程中，可以随时指定要压的预期 qps，如果是下调压力，平台会非常快的把压测集群的输出流量，压到指定的预期 qps。如果是上调压力，当前压测集群中的施压机整体输出压力上调后不能达到预期的 qps，系统会从施压机公共资源池调配一定数量的施压机进入压测集群，确保在服务容量以内，压测流量能够上调到预期 qps。通过这个功能，压测过程中增减压力，我们不再需要人工根据单机施压能力估算要增减多少施压机，整个压力调控过程完全是由系统自动进行的。集群的整体压力也实现了精准调控，达到平台在压力输出能力上指哪打哪能力。

功能介绍

在压测过程进行中，可以对压测任务预期 qps 进行指定。



压测集群整体压力输出根据指定的预期 qps 实时调整。



接口压力调控

接口级别的压力调控，目前平台已经落地了接口流量占比动态调整能力，压测过程中可以随时对接口差异流量做调整，实现了在不停压测的情况下对接口流量

占比的即时调整能力。该功能对保障全链路压测的顺利进行起到了关键作用，让我们可以方便高效地对压力模型做调整，使全链路压测得以平滑顺利进行，提升了全链路压测参与各方的幸福感。

今年十一全链路压测，由于业务增长较快，导致压力预估模型与实际预期有较大出入，通过接口比例动态调整功能，在不停压测的情况下我们对压力模型进行了近百次调整，保证了全链路压测的流畅度，保障了全链路压测的顺利进行。

针对于接口特征参数级别的精准调控，我们的解决方案是：通过语料智能化生产，对接口特征参数提取和统计，可以把压测接口按特征参数分布拆分为多个，结合接口调压能力，最终在接口级别压力调控上，可以进一步实现接口特征参数级别的更精细调控，这对于高德业务系统的全链路压测来说是非常重要，可以使我们实施更加精准的压测。

功能介绍

在压测过程中，对压测接口流量占比实时调整。

质量篇—高德全链路压测——精准控压的建设实践

接口比例调整 压力控制

输入名称或api进行搜索 q

背景颜色 代表您修改当前记录的接口比例

查看帮助

接口名称	api	接口比例
netty-get-0	/?\${params}	10.000000
netty-get-1	/?\${params}	90.000000

重新计算接口比例 保存接口比例 查看推送结果 保存接口比例快照

接口发压流量占比实时调整效果。



总结

通过发压能力精准调控的建设，目前 TestPG 压测平台具备了集群和接口两个级别的高效精准压力调控能力。提升了全链路压测效率，有效缩短了全链路压测的周期，在全链路压测过程中可以使我们方便高效地对压力模型做调整，使其更加接近于线上真实情况。

全链路压测的目的在于验证服务稳定性保障措施是否符合预期，提前发现服务稳定性问题。压测的真实性至关重要，这里的真实性是指压测的语料数据与线上

用户场景相符合，压力模型也要与线上相符合。目前我们的压力模型是通过计算和预估的方式给出的，往往与线上压力模型存在出入；而压测数据虽然是基于线上流量生产的，但与春节、十一当天的场景还是有差异的。

在真实性保障上，目前 TestPG 压测平台也已经有了相应的解决方案，那就是语料智能化生产加上精准控压。未来，我们期望通过语料智能化项目的推进，借助机器学习等手段，通过对压力模型，以及用户场景的预测，并结合精准控压技术，让全链路压测的压测流量模型更加接近于春节、十一等节假日线上真实情况，实现真正意义上的精准压测。

未来思考

精准控压技术，赋予了平台对压力的精准调控能力，解决了全链路压测过程中，压力调控效率低下问题，保障了全链路压测的顺畅度；并且通过对压力模型的方便高效调整，也使得全链路压测的压力模型更加符合线上真实情况。但是精准控压技术的用武之地不应该局限于此。

未来我们可以在压测类型上做进一步探索。目前从 TestPG 压测平台的使用情况来看，日常压测上大家并没有有意识的区分压测类型。绝大部分情况下大家都是以固定 qps 持续对系统施压。但是，线上系统面临的真实流量有时候并不是固定，有可能出现极限尖峰脉冲等情况。平时系统如果没有压测过这种极限场景，那么线上系统遇到这种异常流量时就有可能被打挂。

基于精准控压技术，我们可以在压力模型上做进一步深挖，未来可以在平台上提供多种压测类型支持，比如负载测试、压力测试、系统容量预估测试等等。并为每种压测类型配备相应的压力模型，比如可以提供负载递增的压力模型，这样可以方便我们探查系统容量极限；又比如通过提供极限脉冲压力模型，可以有效测试系统在遭遇异常流量时候的表现。我们期望通过提供丰富的压测类型支持，把系统性能的各方面都测到，把系统压测做全做得更专业。

如何规范你的 Git commit?

作者：睿光

导读

commit message 应该如何写才更清晰明了？团队开发中有没有遇到过让人头疼的 git commit？高德技术团队的同学结合自己的经验，撰写了本文分享在 git commit 规范建设上的实践。

背景

Git 每次提交代码都需要写 commit message，否则就不允许提交。一般来说，commit message 应该清晰明了，说明本次提交的目的，具体做了什么操作……但是在日常开发中，大家的 commit message 千奇百怪，中英文混合使用、fix bug 等各种笼统的 message 司空见怪，这就导致后续代码维护成本特别大，有时自己都不知道自己的 fix bug 修改的是什么问题。

基于以上这些问题，我们希望通过某种方式来监控用户的 git commit message，让规范更好的服务于质量，提高大家的研发效率。

规范建设

规范梳理

初期我们在互联网上搜索了大量有关 git commit 规范的资料，但只有 Angular 规范是目前使用最广的写法，比较合理和系统化，并且有配套的工具（IDEA 就有插件支持这种写法）。最后综合阿里巴巴高德地图相关部门已有的规范总结出了一套 git commit 规范。

commit message 格式

```
<type>(<scope>): <subject>
```

type(必须)

用于说明 git commit 的类别，只允许使用下面的标识。

feat: 新功能 (feature) 。

fix/to: 修复 bug，可以是 QA 发现的 BUG，也可以是研发自己发现的 BUG。

- fix: 产生 diff 并自动修复此问题。适合于一次提交直接修复问题。
- to: 只产生 diff 不自动修复此问题。适合于多次提交。最终修复问题提交时使用 fix.

docs: 文档 (documentation)。

style: 格式 (不影响代码运行的变动)。

refactor: 重构 (即不是新增功能，也不是修改 bug 的代码变动)。

perf: 优化相关，比如提升性能、体验。

test: 增加测试。

chore: 构建过程或辅助工具的变动。

revert: 回滚到上一个版本。

merge: 代码合并。

sync: 同步主线或分支的 Bug。

scope(可选)

scope 用于说明 commit 影响的范围，比如数据层、控制层、视图层等等，视项目不同而不同。

例如，在 Angular，可以是 location, browser, compile, compile,rootScope, ngHref, ngClick, ngView 等。

如果你的修改影响了不止一个 scope，你可以使用*代替。

subject(必须)

subject 是 commit 目的的简短描述，不超过 50 个字符。

- 建议使用中文（感觉中国人用中文描述问题能更清楚一些）。
- 结尾不加句号或其他标点符号。

根据以上规范 git commit message 将是如下的格式：

1. fix(DAO): 用户查询缺少 username 属性
2. feat(Controller): 用户查询接口开发

以上就是我们梳理的 git commit 规范，那么我们这样规范 git commit 到底有哪些好处呢？

- 便于程序员对提交历史进行追溯，了解发生了什么情况。

- 一旦约束了 commit message，意味着我们将慎重的进行每一次提交，不能再一股脑的把各种各样的改动都放在一个 git commit 里面，这样一来整个代码改动的历史也将更加清晰。
- 格式化的 commit message 才可以用于自动化输出 Change log。

监控服务

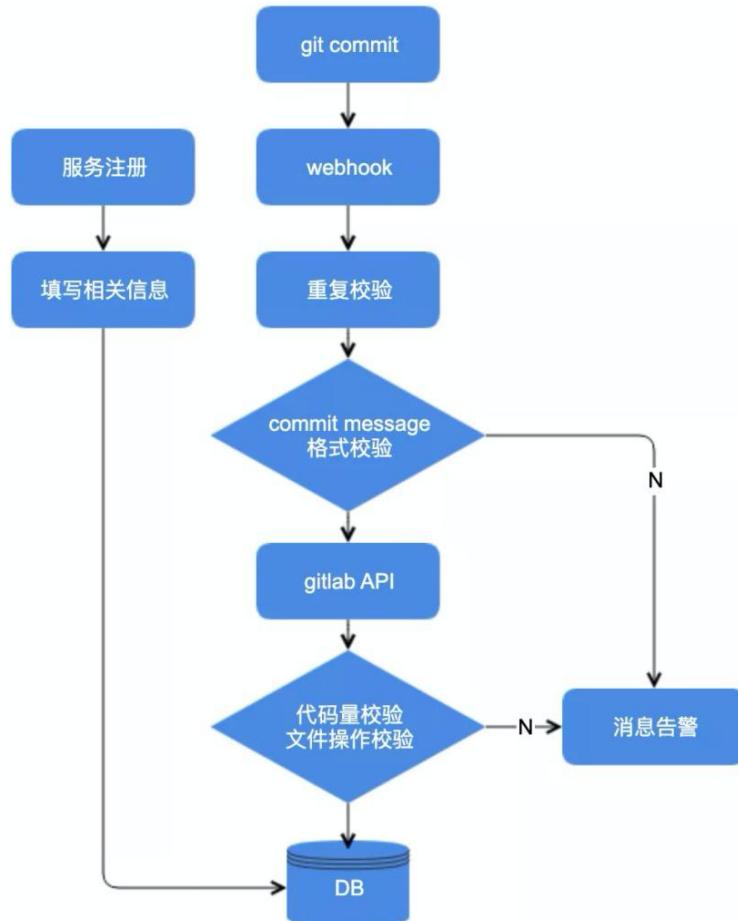
通常提出一个规范之后，为了大家更好的执行规范，就需要进行一系列的拉通，比如分享给大家这种规范的优点、能带来什么收益等，在大家都认同的情况下最好有一些强制性的措施。

当然，git commit 规范也一样，前期我们分享完规范之后考虑从源头进行强制拦截，只要大家提交代码的 commit message 不符合规范，直接不能提交。

但由于代码仓库操作权限的问题，我们最终选择了使用 webhook 通过发送警告的形式进行监控，督促大家按照规范执行代码提交。

除了监控 git commit message 的规范外，我们还加入了大代码量提交监控和删除文件监控，减少研发的代码误操作。

整体流程



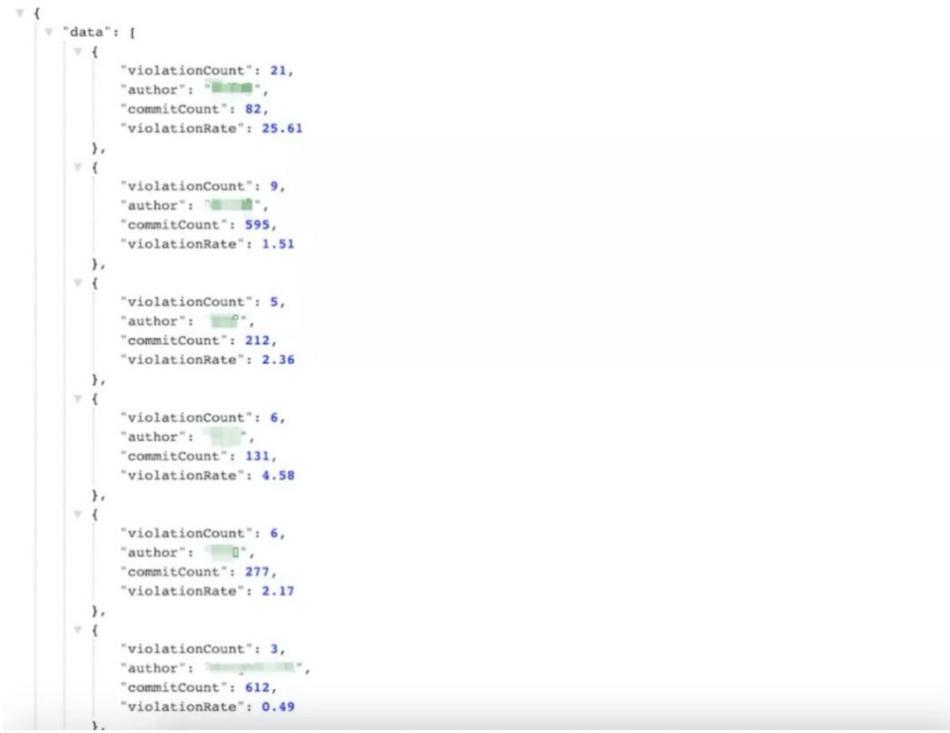
- 服务注册：服务注册主要完成代码库相关信息的添加。
- 重复校验：防止 merge request 再走一遍验证流程。
- 消息告警：对不符合规范以及大代码量提交、删除文件等操作发送告警消息。
- DB：存项目信息和 git commit 信息便于后续统计 commit message 规范率。

webhook 是作用于代码库上的，用户提交 git commit，push 到仓库的时候就会触发 webhook， webhook 从用户的 commit 信息里面获取到 commit message，校验其是否满足 git commit 规范，如果不满足就发送告警消息；如果满足规范，调用 gitlab API 获取提交的 diff 信息，验证提交代码量，验证是否有重命名文件和删除文件操作，如果存在以上操作还会发送告警消息，最后把所有记录都入库保存。

以上就是我们整个监控服务的相关内容，告警信息通过如下形式发送到对应的钉钉群里：



我们也有整体 git commit 的统计，统计个人的提交次数、不规范次数、不规范率等如下图：



```
{ "data": [ { "author": "User A", "commitCount": 82, "violationCount": 21, "violationRate": 25.61 }, { "author": "User B", "commitCount": 595, "violationCount": 9, "violationRate": 1.51 }, { "author": "User C", "commitCount": 212, "violationCount": 5, "violationRate": 2.36 }, { "author": "User D", "commitCount": 131, "violationCount": 6, "violationRate": 4.58 }, { "author": "User E", "commitCount": 277, "violationCount": 6, "violationRate": 2.17 }, { "author": "User F", "commitCount": 612, "violationCount": 3, "violationRate": 0.49 } ] }
```

未来思考

git hooks 分为客户端 hook 和服务端 hook。客户端 hook 又分为 pre-commit、prepare-commit-msg、commit-msg、post-commit 等，主要用于控制客户端 git 的提交工作流。用户可以在项目根目录的 .git 目录下面配置使用，也可以配置全局 git template 用于个人 pc 上的所有 git 项目使用。服务端 hook 又分为 pre-receive、post-receive、update，主要在服务端接受提交对象时进行调用。

以上这种采用 webhook 的形式对 git commit 进行监控就是一种 server 端的 hook，相当于 post-receive。这种方式并不能阻止代码的提交，它只是通过告警的形式来约束用户的行为，但最终不规

范的 commit message 还是被提交到了服务器，不利于后面 change log 的生成。

由于公司代码库权限问题，我们目前只能添加这种 post-receive 类型的 webhook。如大家有更高的代码库权限，可以采用 server 端 pre-receive 类型的 webhook，直接拒绝不规范的 git commit message。只要 git commit 规范了，我们甚至可以考虑把代码和 bug、需求关联等等。

当然，这块我们也可以考虑客户端的 pre-commit，pre-commit 在 git add 提交之后，然后执行 git commit 时执行，脚本执行没错就继续提交，反之就会驳回。客户端 git hooks 位于每个 git 项目下的隐藏文件 .git 中的 hooks 文件夹里。我们可以通过修改这块的配置文件添加我们的规则校验，直接阻止不规范 message 的提交，也可以通过客户端 commit-msg 类型的 hook 进行拦截，把不规范扼杀在萌芽之中。

修改每个 git 项目下面 .git 目录中的 hooks 文件大家肯定觉得浪费时间，其实这里可以采用配置全局 git template 来完成。但是这又会涉及到 hooks 配置文件同步的问题。hooks 配置文件在本地，如何让 hooks 配置文件的修改能同步到所有使用的项目又成为一

个问题。所以使用服务端 hook 还是客户端 hook 需要根据具体需求做适当的权衡。

git hook 不光可以用来做规范限制，它还可以做更多有意义的事情。一次 git commit 提交的信息量很大，有作者信息、代码库信息、commit 等信息。我们的监控服务就根据作者信息做了 git commit 的统计，这样不仅可以用来监控 commit message 的规范性，也可以用来监控大家的工作情况。

我们也可以把 git commit 和相关的 bug 关联起来，我们查看 bug 时就可以查看解决这个 bug 的代码修改，很有利于相关问题的追溯。当然我们用同样的方法也可以把 git commit 和相关的需求关联起来，比如我们定义一种格式 feat *786990（需求的 ID），然后在 git commit 的时候按照这种格式提交， webhook 就可以根据这种格式把需求和 git commit 进行关联，也可以用来追溯某个需求的代码量，当然这个例子不一定合适，但足以证明 git hook 功能之强大，可以给我们的流程规范带来很大的便利。

总结

编码规范、流程规范在软件开发过程中是至关重要的，它可以使我们在开发过程中少走很多弯路。Git commit 规范也是如此，确实也是很有必要的，几乎不花费额外精力和时间，但在之后查找

问题的效率却很高。作为一名程序员，我们更应注重代码和流程的规范性，永远不要在质量上将就。

招聘

阿里巴巴高德地图质量部热招测试开发高级工程师/专家，Java、C++高级工程师/专家，算法岗位。职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

单元测试在高德在线导航业务中的实践

作者：棋李

导读：TDD(Test Driven Development)是一种强调测试先行的开发方式，通过编写单元测试用例，有效保障存量复杂系统在开发、重构上的质量。在本文中，高德智能技术中心的同学通过分析现有测试方法面临的问题，分享如何使用 GTest 框架进行单元测试，以及在单元测试中的一些实践心得。

一 业务背景

高德在线导航服务作为有很强业务特性和多年历史积累的存量系统，不可避免的存在大量的不合理代码，而业务演进对系统性能、算法、底层架构等不断提出更高要求，存量的各种业务代码和算法、架构快速演进的诉求存在严重冲突，如何有效保障质量地进行快速重构式演进，成为业务发展面临的首要工程难题。

二 现有质量保障方法问题与分析

1 现有测试方法的问题

常规方法是对新老服务批量进行请求比较 diff，这种方式简单有效，是我们一直在用的方法，但存在以下问题：

- 无效 diff 问题：以公交规划引擎为例，依赖步导引擎、搜索、公交突发事件、路况等多个下游服务，获取结果的差异导致很多无效 diff。
- 运行时间较长：case 量较多时运行时间较长，在 10 分钟级别。由于这一步成本较高，一般开发人员跑 diff 的频率不会太高，无法进行“每次一小步”的测试。
- 排查困难：当发现 diff 后进行排查非常困难，因为是整个请求级别的 diff，中间步骤可能都存在问题。

2 业界主流方法实践

ThoughtWorks、Google 等公司使用 TDD 方式进行敏捷开发，通过编写单元测试用例保障开发、重构的质量，目前已经成为主流最佳实践。

三 单元测试介绍

1 什么是单元测试？

单元测试是对一个模块、一个函数或者一个类进行正确性检验的测试工作。

测试的粒度更小更轻量，运行时间在秒级，特别适合渐进式重构中的“每次一小步”的质量保障。

由于单元测试用例针对的是一个函数、类更细粒度的目标，所以当某个用例不通过时，可以快速锁定问题点。

2 单元测试框架

常见单元测试框架有 xUnit 系列，多种语言都有对应实现，如 CppUnit、JUnit、NUnit...

GTest 是 Google 开发的单元测试框架，此框架具有一些高级功能，如 death test，mock 等。

我们选择的是 GTest 框架。

3 单元测试、重构、TDD 与敏捷

TDD(Test Driven Development)是强调测试先行的开发方式，这种方式的好处在于编写任何函数、修改任何代码时可以通过编写一个单元测试用例代码来表达要实现的代码功能，一个测试用例本

身就是一个代码表达的需求。而积累起来的测试用例可以有效保障开发及后续重构演进的质量。

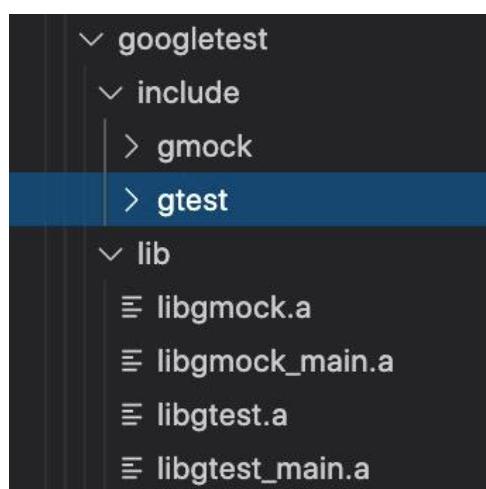
重构和 TDD 是敏捷方法的核心构成要素，脱离了 TDD 的敏捷是危险的，没有用例保障的重构一旦启动，就像一匹脱缰的野马。而单元测试和 TDD 则是缚住野马的缰绳。

四 公交服务单元测试实践

1 GTest 框架集成

Git 库地址：<https://github.com/google/googletest>

GTest 框架集成非常简单，把 googletest 库加入到工程中，增加链接 libgtest 即可：



通过如下代码即可驱动用例执行：

```
1. int RCUnitTest::Excute()
2. {
3.     int argc = 2;
4.     char* argv[] = {const_cast<char*>(""), const_cast<char*>("--gtest_out-
put=\"xml:./testAll.xml\"")};
5.     ::testing::InitGoogleTest(&argc, argv);
6.
7.     return RUN_ALL_TESTS();
```

开关控制：为避免影响到正式版本，可以考虑通过编译控制，也可以增加一个配置项开关。

我们在使用时是在入口处通过一个配置项控制是否触发单元测试用例，编译时默认只链接入口文件，需要运行单元测试时添加上单元测试用例文件进行链接运行。

2 测试代码编写

通过实现一个 Test 类的派生类，然后使用 TEST_F 宏添加测试函数即可，如下示例：

质量篇—单元测试在高德在线导航业务中的实践

```
1. class DateTimeUtilTest : public ::testing::Test  
2. {  
3. protected:  
4.     virtual void SetUp()  
5. {  
6. }  
7.  
8. virtual void TearDown()  
9. {  
10. }  
11. };  
12.  
13. TEST_F(DateTimeUtilTest, TestAddSeconds_leap)  
14. {  
15.     //闰年测试 2020-02-28  
16.     tm tt;  
17.     tt.tm_year = (2020 - 1900);  
18.     tt.tm_mon = 1;  
19.     tt.tm_mday = 28;  
20.     tt.tm_hour = 23;  
21.     tt.tm_min = 59;  
22.     tt.tm_sec = 50;
```

```
23.  
24.     DateTimeUtil::AddSeconds(tt, 30);  
25.     EXPECT_TRUE(tt.tm_sec == 20);  
26.     EXPECT_TRUE(tt.tm_min == 0);  
27.     EXPECT_TRUE(tt.tm_hour == 0);  
28.     EXPECT_TRUE(tt.tm_mday == 29);  
29.     EXPECT_TRUE(tt.tm_mon == 1);  
30.  
31. //非闰年测试 2019-02-28  
32. tm tt1;  
33. tt1.tm_year = (2019 - 1900);  
34. tt1.tm_mon = 1;  
35. tt1.tm_mday = 28;  
36. tt1.tm_hour = 23;  
37. tt1.tm_min = 59;  
38. tt1.tm_sec = 50;  
39. DateTimeUtil::AddSeconds(tt1, 30);  
40. EXPECT_TRUE(tt1.tm_sec == 20);  
41. EXPECT_TRUE(tt1.tm_min == 0);  
42. EXPECT_TRUE(tt1.tm_hour == 0);  
43. EXPECT_TRUE(tt1.tm_mday == 1);  
44. EXPECT_TRUE(tt1.tm_mon == 2);
```

```
45. };
```

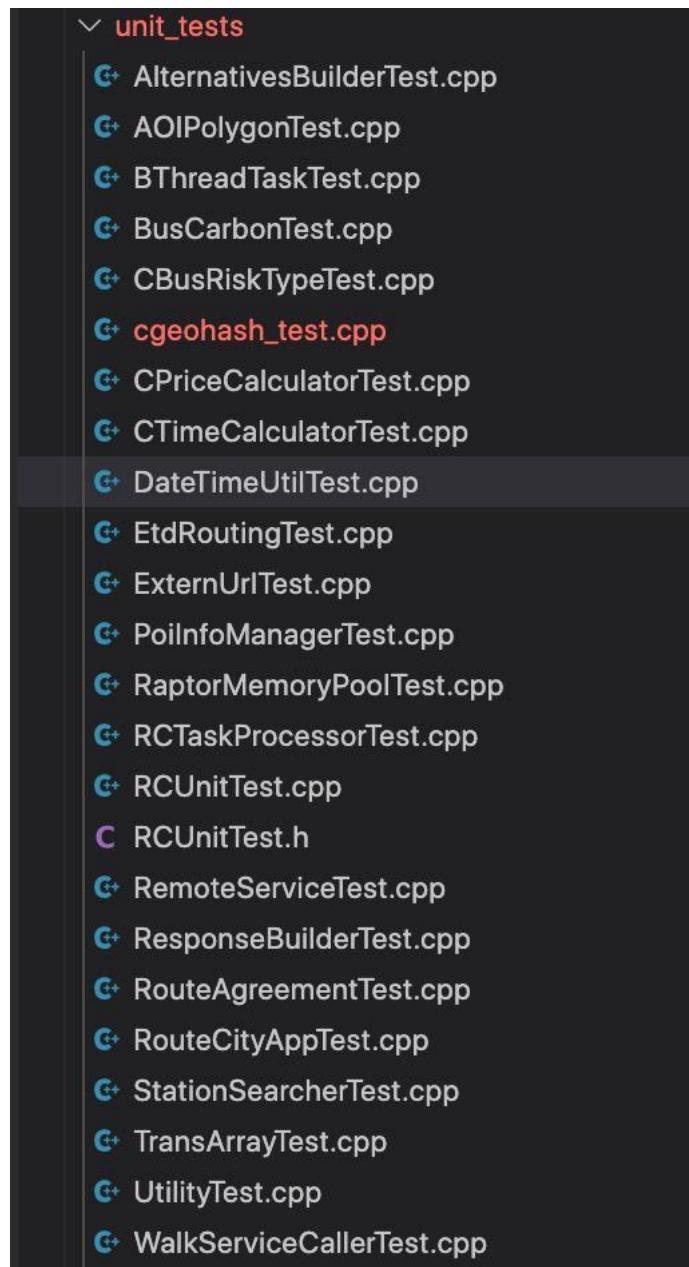
测试用例执行效果：

```
11223 21:00:40.620003 9250 src/brpc/server.cpp:1042] Server[route..RouteBrokerService] is serving o
W1223 21:00:40.620082 9250 src/brpc/server.cpp:1048] Builtin services are disabled according to Ser
WARNING: unrecognized output format "'xml'" ignored.
[=====] Running 9 tests from 2 test cases.
[-----] Global test environment set-up.
[-----] 6 tests from StationSearcherTest
[ RUN   ] StationSearcherTest.TestOriginSubPoi
start_walktost_count0:343
term_walktost_count0:340
[ OK   ] StationSearcherTest.TestOriginSubPoi (4 ms)
[ RUN   ] StationSearcherTest.TestTermSubPoi
start_walktost_count1:335
term_walktost_count1:344
[ OK   ] StationSearcherTest.TestTermSubPoi (2 ms)
[ RUN   ] StationSearcherTest.TestNoOrigin
[ OK   ] StationSearcherTest.TestNoOrigin (0 ms)
[ RUN   ] StationSearcherTest.TestAladdin
[ OK   ] StationSearcherTest.TestAladdin (1 ms)
[ RUN   ] StationSearcherTest.TestLoopLine
[ OK   ] StationSearcherTest.TestLoopLine (0 ms)
[ RUN   ] StationSearcherTest.TestEnlargeSearchRange
term_walktost_count2:54
[ OK   ] StationSearcherTest.TestEnlargeSearchRange (1 ms)
[-----] 6 tests from StationSearcherTest (8 ms total)

[-----] 3 tests from AOIPolygonTest
[ RUN   ] AOIPolygonTest.TestGetBoundStations
[ OK   ] AOIPolygonTest.TestGetBoundStations (3 ms)
[ RUN   ] AOIPolygonTest.TestGetInnerStations
[ OK   ] AOIPolygonTest.TestGetInnerStations (0 ms)
[ RUN   ] AOIPolygonTest.TestGetInnerStations_haiouisland
[ OK   ] AOIPolygonTest.TestGetInnerStations_haiouisland (0 ms)
[-----] 3 tests from AOIPolygonTest (3 ms total)

[-----] Global test environment tear-down
[=====] 9 tests from 2 test cases ran. (11 ms total)
[ PASSED ] 9 tests.
```

目前公交引擎已经积累了 23 个模块测试用例，基本覆盖了寻站、寻路、ETA、票价、风险停运等核心功能，持续积累中。通过单元测试保障，每个版本开发活动中都在进行渐进式重构活动，能够有效保障质量，提测迭代次数和线上新增代码引入问题数量持续较低。



3 问题与难点

数据依赖问题

在线导航引擎是对数据重度依赖的业务，多组数据结构之间互相关联，字段繁多，很难脱离数据构建有效的单元测试。通过 mock

方式构造假数据成本很高。而数据变化将导致用例不能通过。

我的实践：能够简单构造假数据的通过构造假数据来搞定。

对于很难构建假数据的情况，直接使用真实数据即可。数据变化可能导致这部分用例不通过，没有关系，只需要保障在每次重构前把相关的用例调通即可，这样仍可以确保重构过程的质量。即：不需要做到用例随时随地都能运行通过，而是保证重构前后都可以通过。

4 常见错误认知

对于没有真正实践过单元测试和 TDD 开发方式的同学来说，有一些认知上的常见误区，比如：

开发时间都不够，哪有时间编写单元测试？

我的理解：

- 首先 TDD 的开发方式强调的是测试先行，编写测试代码是在前面的，这个过程等于是理解需求的过程。即想清楚你要实现的是什么功能？这个测试代码是理清需求的

产物，如此而已，不存在更多时间成本。

- TDD 开发方式属于典型的一次投入，持续受益的事情，用例积累越多，越容易在早期发现问题，重构有了质量保障，代码越来越整洁清晰，开发同学们再也不用哀叹历史代码。

历史代码那么多，怎么补单元测试？

那就从添加第一个用例开始。我的做法是对应本次修改涉及到的代码添加用例，逐步积累。

添加用例的过程是理解现有代码的过程，对于存量的历史代码，各种硬性编码侵入，各种耦合，全局变量或长生命周期大对象，通过编写单元测试用例能够有效理清函数真正的输入输出，也为重构增加了有效保障。

五 存量复杂系统代码渐进式重构

对于我们一线码农，每天大部分时间都在和代码打交道，如果你维护的代码结构合理、易读易扩展，那么恭喜你！但大部分情况我们面对的是存在各种历史“积淀”的存量工程，各种牵一发而动

全身，这种情况下小改动还可以靠多花时间，认真仔细来搞定，但想要做一些大的系统升级就难了。

而对于巨型业务系统来说，重写在成本和质量控制方面显得更不现实。那么设置几个大的节点，通过渐进式重构逐渐优化，变量变为质变，是综合来看最优的方式。

而单元测试和 TDD，则是渐进式重构有效开展的必选方法。

招聘

阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

顶会论文篇

KDD 论文解读 | 混合时空图卷积网络：更精准的时空预测模型

作者：高德智能技术中心

导读

时空预测在天气预报、运输规划等领域有着重要的应用价值。交通预测作为一种典型的时空预测问题，具有较高的挑战性。以往的研究中主要利用通行时间这类交通状态特征作为模型输入，很难预测整体的交通状况，本文提出的混合时空图卷积网络，利用导航数据大大提升了时空预测的效果（本文中所提到的技术解决方案论文已被收录到国际 AI 顶会 KDD2020）。

论文下载地址：<https://arxiv.org/abs/2006.12715>

日常通勤中的规律往往相对容易挖掘，但交通状况还会受很多其他因素影响，之前的研究主要利用通行时间这类交通状态作为特征，少量研究引入事件，不能很好地预测实际交通流量。

为解决这一问题，本文从高德导航引擎中获取了「计划中交通流量」，并将其扩充到机器学习模型当中。

计划中交通流量来自导航数据，反映了用户出行意图中蕴含的未来交通流量。由于拥有海量用户，高德地图中的导航规划数据能够较为全面地反应正在发生的通行需求，并且信息粒度较事件级别的特征更精细。

具体来说，规划的路线产生了计划中交通流量，而计划中交通流量可以用来指导对未来通行时间的预测。 vol_f 代表当前可获取的导航路径在 f 个时间步后在此路段产生的计划中交通流量。计划中交通流量的迅速飙升意味着即将到来的交通拥堵。

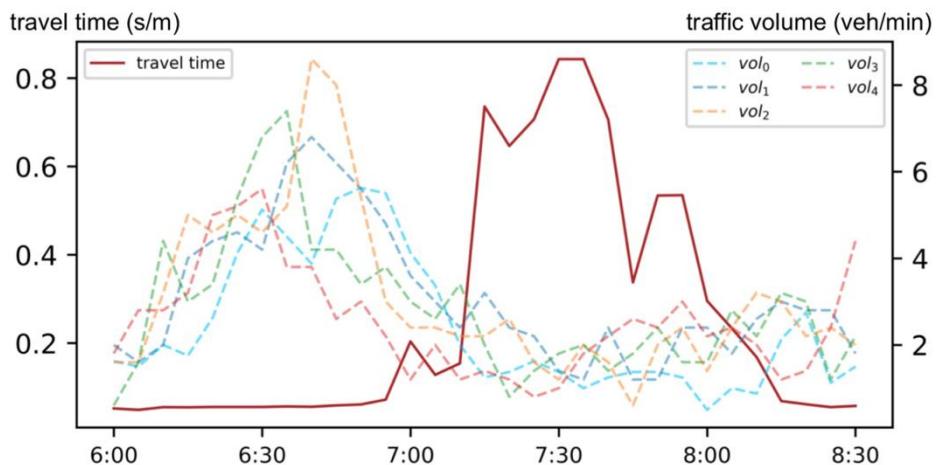


图 1 北京某路段在 2019 年 10 月 28 日早高峰期间通行时间和计划中交通流量

为了将交通流量这一异质信号整合到通行时间预测模型当中，我们创新性地设计出一种**域转换器**(domain transformer)结构，用于将交通流量信息转化为通行时间信息。

交通流理论中，路段的交通流量和车辆密度呈三角形曲线映射关系，而曲线的参数是因路段而异的。图 2 展示了现实世界中的例子。为了利用这一转换关系，我们设计了将流量转化为通行时间的转换器，该转换器由两层网络构成，分别用于提取全局共享信息和学习不同路段的精细化信息。

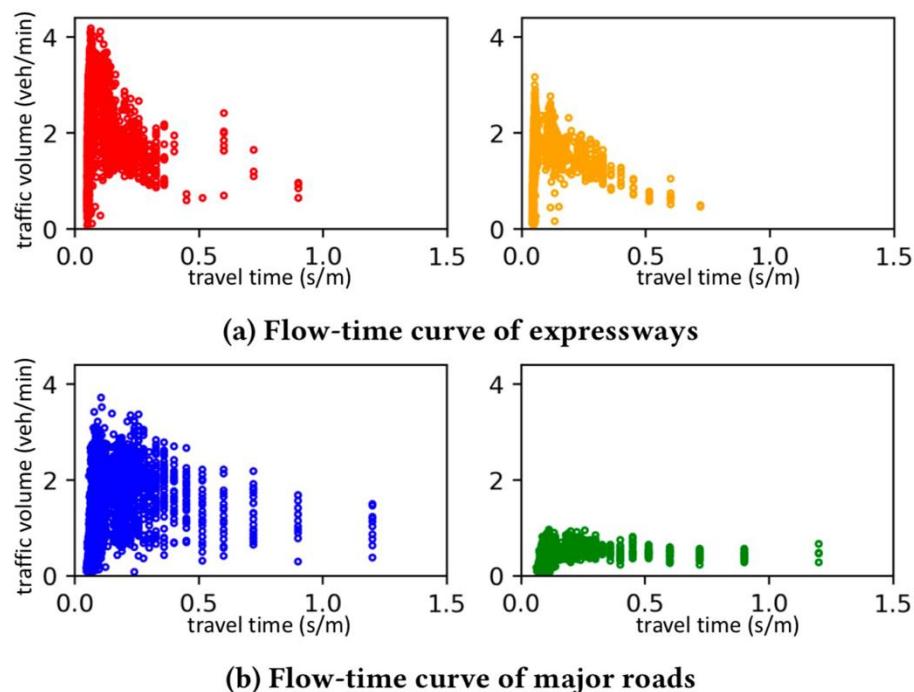


图 2 四个不同路段的流量时间曲线

另一方面，由于交通网络的非欧几里得特性，我们利用图卷积(graph convolution)结构提取空间依赖性特征，并设

计了一种新的邻接矩阵用于更好地体现路段间的交通邻近性。

在以往的研究中[6]，邻接矩阵的权重只按距离衰减，并没有考虑到路段间固有的交通邻近性（图 3 给出了距离近但交通状态相差较大的例子）。为解决这一问题，我们设计了一种复合邻接矩阵（compound adjacency matrix），在距离衰减的基础上进一步引入了路段通行时间的协方差。



图 3 相邻道路间拥堵不一定会传播

本文提出的混合时空图卷积网络（Hybrid Spatio-Temporal Graph Convolutional Network，H-STGCN）是综合利用上述技术的交通预测框架。

在H-STGCN中，转换器将未来交通流量信号转化为通行时间信号。路段间参数共享的门控卷积用于提取时间依赖信

息。

基于复合邻接矩阵的图卷积从合并后的通行时间信号中捕捉空间依赖信息。H-STGCN经由端到端的训练，可具备基于计划中交通流量信息预测未来拥堵的能力。利用真实交通数据集进行实验可验证，H-STGCN的效果显著优于各种前沿模型。

混合时空图卷积网络，独创域转换器和复合邻接矩阵整体框架

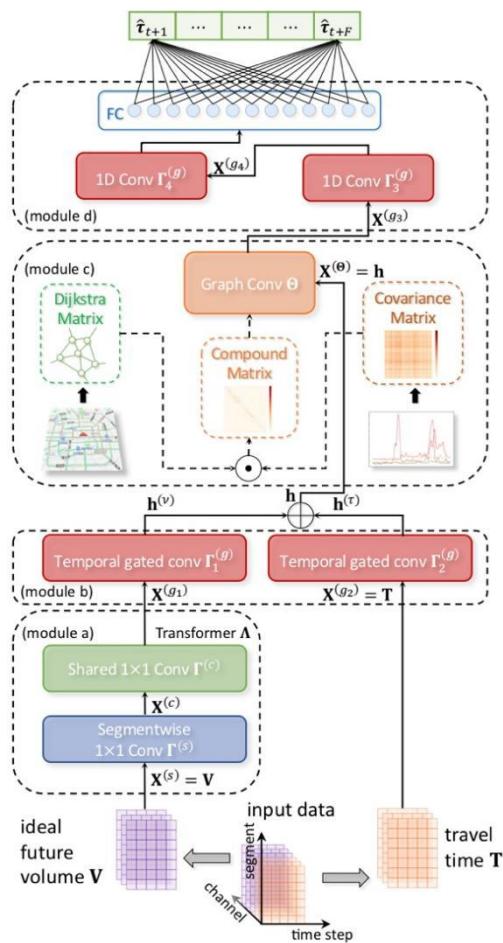


图 4 H-STGCN 模型框架

图 4 展示了 H-STGCN 的整体框架。模型输入由两个特征张量组成，理想未来流量 V 和通行时间张量 T 。 V 和 T 均包含三个维度：空间维度、时间维度、通道维度，分别对应路段、所使用的时间片和特征。

域转换器（模块 a）首先将 V 转化为通行时间信号，输出未来通行时间张量 $X^{(g1)}$ 。接下来，两个独立的门控卷积（模块 b）分别作用于 $X^{(g1)}$ 和 T 的时间维度以提取更高层级的时域特征。

将每个路段视为一个节点，基于复合邻接矩阵的图卷积（模块 c）作用在合并（concatenation）后的信号 $h = h^V \oplus h^T$ 上。两个门控卷积继续扩大时域上的感知范围，并最终经由一个全连接层（FC）输出预测结果。

模型输入与数据处理

输入特征张量 X 的每个切片对应了一个单独的时间片 t ($\leq t_0$)。每个切片又由两部分组成：理想未来流量和通行时间。

理想未来流量。作为对真实未来流量这一无法获取信息的近似，理想未来流量 $v_{i,t_0,f}$ 可以通过在线导航引擎获取。图 5 示意了高德导航系统的架构。导航过程中，车辆每秒钟与云服务器同步自身坐标，与此同时，为保证用户获取到最新的交通状态信息，云服务器对 ETA 进行几乎实时的持续更新。

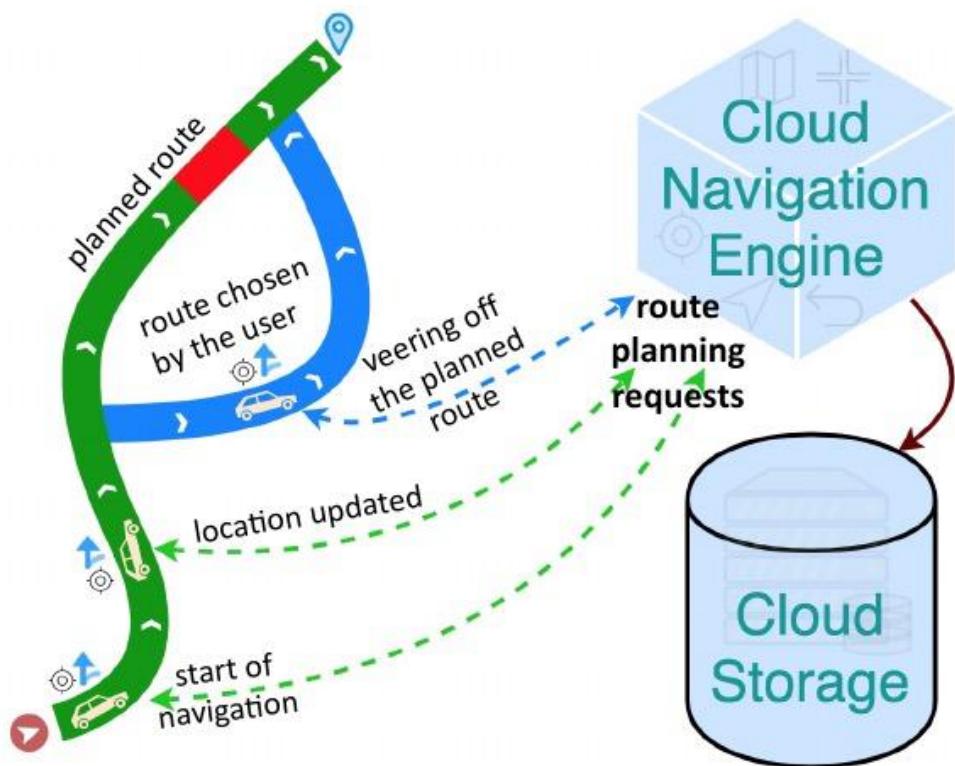


图 5 高德导航系统架构示意

高德导航引擎中原始数据的形式为

$$\mathcal{L} = \{(r, \{(\rho_{r,l}, \delta_{r,l}, \psi_r) | l \in [0, M_r)\}) | r \in [0, N_{\mathcal{L}})\},$$

其中 r 是导航进程的索引号， ψ_r 是导航 r 的发起时间， $\rho_{r,l}$ 代表规划路线中的第 l 个路段， $\delta_{r,l}$ 是到达 $\rho_{r,l}$ 的预估时间， M_r 是路线中路段的总数量， N_φ 是导航进程的总数量。ETA 来自机器学习模型的预测（利用历史轨迹等数据训练得到）。算法 1 展示了从导航路线集合中推算理想未来流量的方法。

Algorithm 1 Route aggregation algorithm to obtain the ideal future volume

Input: The list of route records from the dataset \mathcal{L}
Output: Ideal future volume v

```

1: Initialize  $Z$  as an empty set
2: for each  $r \leftarrow 0, 1, \dots, N_{\mathcal{L}} - 1$  do
3:   for each  $l \leftarrow 0, 1, \dots, M_r - 1$  do
4:      $s \leftarrow \rho_{r,l}$     $\{s \text{ is the id of a road segment}\}$ 
5:      $t \leftarrow \delta_{r,l}$     $\{t \text{ is a time slot}\}$ 
6:     for  $f \leftarrow 0, 1, 2, \dots, F$  do
7:       if  $t \geq \psi_r$  then
8:          $\zeta \leftarrow (s, t, f)$ 
9:         Add  $\zeta$  to  $Z$ 
10:      else
11:        break
12:      end if
13:       $t \leftarrow t - \Delta t$     $\{\Delta t \text{ stands for the length of a single time slot}\}$ 
14:    end for
15:  end for
16: end for
17: for each  $s_0 \leftarrow 0, 1, \dots, n - 1$  do
18:   for each  $t_0 \leftarrow 0, 1, \dots, S_{\text{train}} + S_{\text{test}} - 1$  do
19:     for each  $f_0 \leftarrow 0, 1, \dots, F$  do
20:        $v_{s_0, t_0, f_0} = \text{cardinality}(\{\zeta | \zeta.s = s_0, \zeta.t = t_0, \zeta.f = f_0, \forall \zeta \in Z\})$ 
21:     end for
22:   end for
23: end for
24: return  $v$ 

```

H-STGCN 中，与预测时间窗口相对应的理想未来流量和历史平均流量同时被输入：

$$\mathbf{V}_{i,t} = \left[\nu_{i,t,0}, \nu_{i,t,1}, \dots, \nu_{i,t,F}, \nu_{i,t,0}^{(h)}, \nu_{i,t+1,0}^{(h)}, \dots, \nu_{i,t+F,0}^{(h)} \right],$$

其中 i 是路段的索引号。

通行时间。通行时间通过完成地图匹配的 GPS 点数据整合计算得到。

H-STGCN 中，通行时间及其与预测时间窗口相对应的历史均值同时被用于模型的输入：

$$\mathbf{T}_{i,t} = \left[\tau_{i,t}, \tau_{i,t}^{(h)}, \tau_{i,t+1}^{(h)}, \dots, \tau_{i,t+F}^{(h)} \right],$$

其中 i 是路段的索引号。

域转换器。域转换器由串联的两层网络组成，即逐路段 1×1 卷积（segmentwise 1×1 convolution）和路段间共享 1×1 卷积（shared 1×1 convolution），图 6 呈现了这一结构。

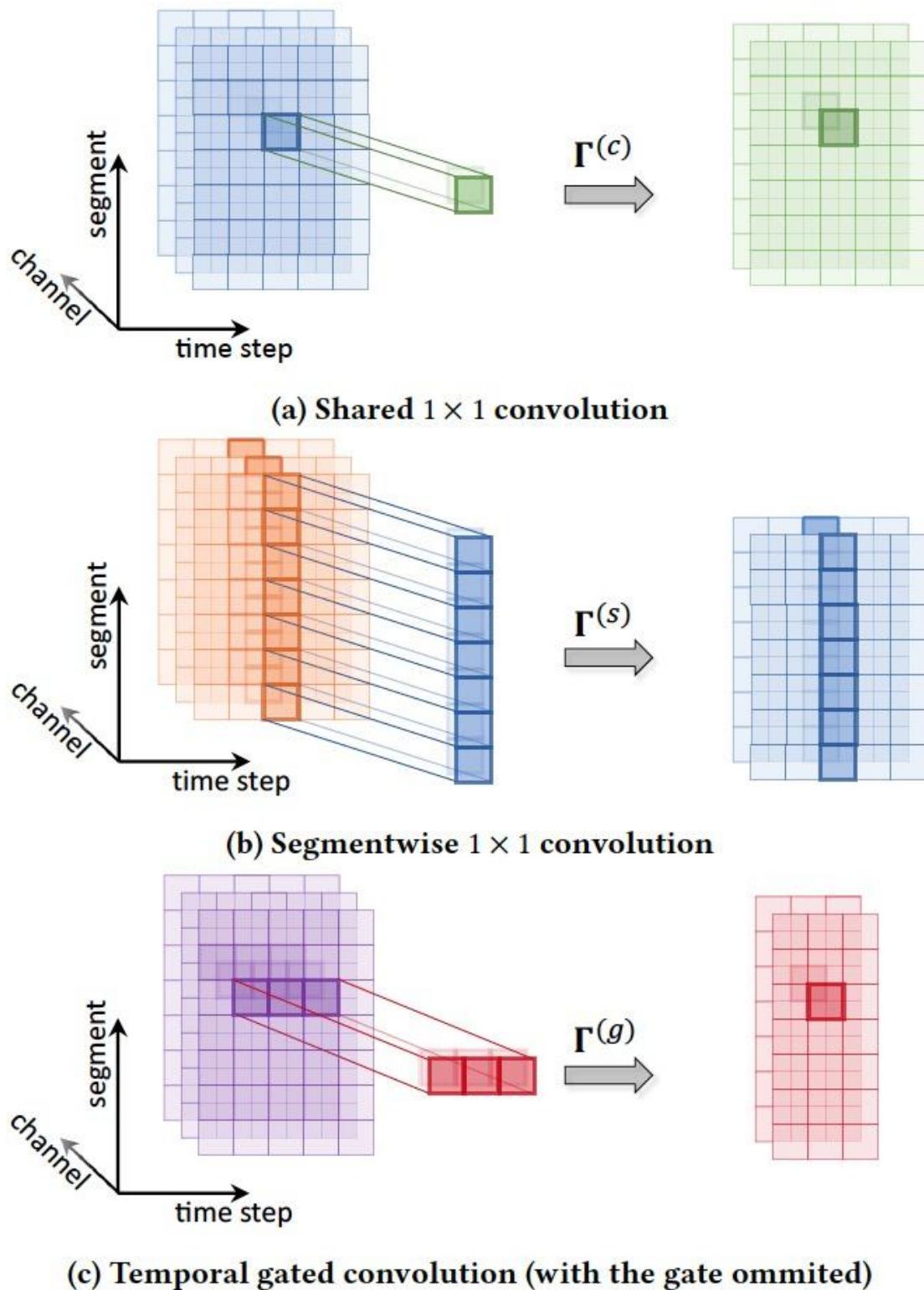


图 6 H-STGCN 中的各种卷积运算

共享 1×1 卷积。路段及时间片间参数共享的 1×1 卷积层 $\Gamma^{(c)}$ 位于域转换器的顶部，该卷积运算阐释如图 6a，旨在捕捉全局的三角形曲线映射关系。记这一层的输入和输出为 $\mathbf{X}_{i,t,:}^{(c)} \in \mathbb{R}^{C^{(c_{in})}}$ 与 $\mathbf{Y}_{i,t,:}^{(c)} \in \mathbb{R}^{C^{(c_{out})}}$ ，则有：

$$\mathbf{Y}_{i,t,:}^{(c)} = \Gamma^{(c)}(\mathbf{X}_{i,t,:}^{(c)}) = \sigma(\mathbf{X}_{i,t,:}^{(c)} \cdot \mathbf{F}^{(c)} + \mathbf{b}^{(c)}),$$

其中 $\mathbf{F}^{(c)} \in \mathbb{R}^{C^{(c_{in})} \times C^{(c_{out})}}$ 为权重， $\mathbf{b}^{(c)} \in \mathbb{R}^{C^{(c_{out})}}$ 为偏置项， σ 为ELU (Exponential Linear Unit) 激活函数。

逐路段 1×1 卷积。为保证模型能够充分提取精细到路段级别的特征，路段参数个性化的 1×1 卷积层 $\Gamma^{(s)}$ 位于域转换器的底部（共享 1×1 卷积前面一层），该卷积运算阐释如图 6b。记这一层的输入和输出为 $\mathbf{X}_{i,t,:}^{(s)} \in \mathbb{R}^{C^{(s_{in})}}$ 与 $\mathbf{Y}_{i,t,:}^{(s)} \in \mathbb{R}^{C^{(s_{out})}}$ ，则有：

$$\mathbf{Y}_{i,t,:}^{(s)} = \Gamma^{(s)}(\mathbf{X}_{i,t,:}^{(s)}) = \sigma(\mathbf{X}_{i,t,:}^{(s)} \cdot \mathbf{F}_{i,:}^{(s)} + \mathbf{b}_{i,:}^{(s)}),$$

其中，为权重， $\mathbf{b}^{(s)} \in \mathbb{R}^{n \times C^{(s_{out})}}$ 为偏置项， σ 是ELU激活函数。

逐路段 1×1 卷积。为保证模型能够充分提取精细到路段级别的特征，路段参数个性化的 1×1 卷积层 $\Gamma^{(s)}$ 位于域转换器的底部（共享 1×1 卷积前面一层），该卷积运算阐释如图 6b。记这一层的输入和输出为 $\mathbf{X}_{i,t,:}^{(s)} \in \mathbb{R}^{C^{(s_{in})}}$ 与 $\mathbf{Y}_{i,t,:}^{(s)} \in \mathbb{R}^{C^{(s_{out})}}$ ，则有：

$$\mathbf{Y}_{i,t,:}^{(s)} = \Gamma^{(s)}(\mathbf{X}_{i,t,:}^{(s)}) = \sigma(\mathbf{X}_{i,t,:}^{(s)} \cdot \mathbf{F}_{i,:}^{(s)} + \mathbf{b}_{i,:}^{(s)}),$$

其中，为权重， $\mathbf{b}^{(s)} \in \mathbb{R}^{n \times C^{(s_{out})}}$ 为偏置项， σ 是ELU激活函数。|

基于复合邻接矩阵的图卷积

复合邻接矩阵。以往研究[6,7]中的邻接矩阵假设节点间的接近性简单地依距离衰减：

$$w_{ij}^{(d)} = \begin{cases} \exp\left(-\frac{d_{ij}^2}{\sigma^2}\right) & , \exp\left(-\frac{d_{ij}^2}{\sigma^2}\right) \geq \epsilon \\ 0 & , \text{otherwise} \end{cases},$$

其中 d_{ij} 为路段 s_i 与 s_j 的最短路距离， σ 控制衰减速率， ϵ 为控制矩阵稀疏性的截断阈值。我们将 $\mathbf{W}^{(d)}$ 称为迪杰斯特拉矩阵 (Dijkstra matrix)。在很多场景下，单纯的空间接近程度并不能反映真实的交通邻近性。更具体而言，交通拥堵对交通分流的影响取决于邻近路段的若干种属性，包括道路等级、路况等。可见，拥堵的传播在空间上并不均匀。由此，我们提出了复合邻接矩阵 $\mathbf{W}^{(c)}$ ：

$$w_{ij}^{(c)} = \sigma_{ij} \cdot w_{ij}^{(d)}, 1 \leq i \leq n, 1 \leq j \leq n,$$

$$\sigma_{ij} = \sum_{t \in [0, S_{\text{train}}]} (\tau_{i,t} - \bar{\tau}_i)_+ (\tau_{j,t} - \bar{\tau}_j)_+,$$

其中 $(\cdot)_+ = \max \{0, \cdot\}$, $\bar{\tau}_i = \sum_{t \in [0, S_{\text{train}}]} \tau_{i,t} / S_{\text{train}}$ 。

图卷积。我们将交通路网视为一个以路段为节点的图。归一化图拉普拉斯 (normalized graph Laplacian) 矩阵 \mathbf{L} 和缩放变换的图拉普拉斯 (scaled graph Laplacian) 矩阵 $\tilde{\mathbf{L}}$ 分别表示为：

$$\mathbf{L} = \mathbf{I}_n - \mathbf{D}^{-\frac{1}{2}} \mathbf{W}^{(c)} \mathbf{D}^{-\frac{1}{2}},$$

$$\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{\max} - \mathbf{I}_n,$$

其中 \mathbf{I}_n 为单位阵, $\mathbf{W}^{(c)}$ 为复合邻接矩阵, 对角阵 \mathbf{D} 为 $\mathbf{W}^{(c)}$ 的度矩阵 (degree matrix), λ_{\max} 是 \mathbf{L} 的最大特征值。图卷积层 Θ 通过 $\tilde{\mathbf{L}}$ 的切比雪夫多项式 (Chebyshev polynomials) 参数化。记这一层的输入和输出为

$$\mathbf{X}_{:,t,:}^{(\Theta)} \in \mathbb{R}^{n \times C^{(\Theta_{\text{in}})}} \text{ 和 } \mathbf{Y}_{:,t,:}^{(\Theta)} \in \mathbb{R}^{n \times C^{(\Theta_{\text{out}})}}$$

则：

$$\mathbf{Y}_{:,t,j}^{(\Theta)} = \sigma \left(\sum_{m=1}^{C^{(\Theta_{\text{in}})}} \sum_{k=0}^{K-1} \Theta_{k,m,j} T_k(\tilde{\mathbf{L}}) \mathbf{X}_{:,t,m}^{(\Theta)} + \mathbf{b}_j^{(\Theta)} \right) \in \mathbb{R}^n, \forall j = 1, 2, \dots, C^{(\Theta_{\text{out}})}$$

其中, $T_k(\tilde{\mathbf{L}})$ 是切比雪夫多项式第 k 阶项, K 是卷积核大小, $\Theta \in \mathbb{R}^{K \times C^{(\Theta_{\text{in}})} \times C^{(\Theta_{\text{out}})}}$ 为权重张量, $\mathbf{b}_j^{(\Theta)}$ 为偏置项, σ 是 ELU 激活函数。

时域门控卷积

如图 6c 所示, 路段间参数共享的一维卷积将输入 $\mathbf{X}^{(g)} \in \mathbb{R}^{n \times P \times C^{(g_{\text{in}})}}$ 转化为张量:

$$[\mathbf{A} \ \mathbf{B}] \in \mathbb{R}^{n \times (P-K_t+1) \times (2C^{(g_{\text{out}})})} = \mathbf{F}^{(g)} * \mathbf{X}^{(g)} + \mathbf{b}^{(g)},$$

其中 $*$ 表示一维卷积运算符。是卷积核, K_t 是卷积核的大小, P 是输入时序长度, $\mathbf{b}^{(g)}$ 是偏置项。 \mathbf{A} 和 \mathbf{B} 形状相同、通道数均为 $C^{(g_{\text{out}})}$ 。我们使用 GLU (gated linear unit) 进一步引入非线性:

$$\Gamma^{(g)} \left(\mathbf{X}^{(g)} \right) = \mathbf{A} \odot \sigma(\mathbf{B}) \in \mathbb{R}^{n \times (P-K_t+1) \times C^{(g_{\text{out}})}}$$

" \odot " 表示哈达玛积 (Hadamard product)。

与STGCN的关系

时空图卷积网络（Spatio-Temporal Graph Convolutional Network, STGCN）[7]将空域图卷积层和时域门控卷积层交替地进行堆叠以同时捕捉时间和空间的依赖性。将H-STGCN的流量特征分支和邻接矩阵中的协方差项去掉，则H-STGCN退化为只有单个时空卷积块（ST-Conv block）的STGCN模型。

模型训练

数据扩充。我们将高斯噪音叠加到流量通道中小于 ϵ_n 的值上，以提升模型的泛化能力。|

优化目标。对于本文中的多时间步预测，我们使用L1损失函数：

$$\mathcal{L} = \frac{1}{n \times S_{\text{train}} \times F} \sum_{\substack{i \in [0, n) \\ t \in [0, S_{\text{train}}) \\ f \in (0, F]}} |\hat{\tau}_{i,t+f} - \tau_{i,t+f}|,$$

其中 $\hat{\tau}_{i,t+f}$ 是模型的输出， $\tau_{i,t+f}$ 为真值。

基于真实路况测试，各项指标均优于传统模型

数据集

实验数据集W3-715和E5-2907，分别对应西三环附近的715个路段和东五环附近的2907个路段（如图7所示）。数据集的时间跨度为2018年12月24日至2019年4月21日（其中包含的节假日被移除，共十周数据），保留的时段为每天的06:00至22:00。前八周数据作为训练集，后两周作为测试集。



图 7 实验路网空间分布

对比模型

基线模型，包括历史均值（HA）、线性回归（LR）、GBRT、MLP、Seq2Seq、STGCN（包含单个时空卷积块）。用于对比实验的变种模型。

- STGCN (Im)：换用复合邻接矩阵的 STGCN（用于和原始的迪杰斯特拉矩阵对比）。
- H-STGCN (1)：将流量特征张量 V 全部设成 1。

评价指标 我们在三种测试集上进行模型效果的比对：

- 全测试集（如 4.1 节中所描述）。
- 高流量路段的拥堵时期，用 C 表示。
- 高流量路段的突发拥堵时期，用尾缀 NRC 表示。

效果比较 表 1 展示了在全测试集、测试集 C、测试集 NRC 上不同模型的表现。

评估标准包括 MAE (s/m)、MAPE (%) 和 RMSE (s/m)。H-STGCN 在各项指标上均显著优于不同的对标模型，在突发拥堵的预测方面优势尤为明显。

表 1 模型效果对比

Dataset	Model	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE
		Test set (Full)			Test set (C)			Test set (NRC)		
W3-715	HA	0.03886	20.73	0.09285	0.07040	34.36	0.10479	0.10303	39.39	0.16486
	LR	0.03334	16.58	0.08467	0.06469	33.52	0.10582	0.09080	39.57	0.14768
	GBRT	0.03264	16.10	0.08409	0.06236	32.08	0.10479	0.09085	39.35	0.14945
	MLP	0.03272	16.57	0.08269	0.06096	31.84	0.10190	0.08733	38.71	0.14427
	Seq2Seq	0.03231	15.81	0.08252	0.06033	28.79	0.10174	0.08599	34.04	0.14467
	STGCN	0.03219	16.01	0.08182	0.05975	30.48	0.09901	0.08599	38.72	0.14004
	STGCN (Im)	0.03200	15.83	0.08196	0.05965	29.96	0.09995	0.08539	36.71	0.14197
E5-2907	H-STGCN (1)	0.03138	15.52	0.08099	0.05804	29.14	0.09806	0.08373	34.71	0.14012
	H-STGCN	0.03114	15.36	0.08045	0.05711	28.34	0.09644	0.08124	33.22	0.13711
	HA	0.04615	21.22	0.11405	0.09786	44.95	0.16729	0.13161	46.96	0.21769
	LR	0.04096	17.03	0.10732	0.08229	41.69	0.14270	0.10747	47.01	0.18192
	GBRT	0.04032	16.61	0.10680	0.07997	39.51	0.14465	0.10657	44.68	0.18593
	MLP	0.04031	17.16	0.10547	0.08025	41.26	0.14229	0.10580	45.84	0.18236
	Seq2Seq	0.04087	17.52	0.10631	0.08413	41.72	0.14703	0.10981	44.81	0.18722
STGCN	0.03984	16.95	0.10296	0.07561	38.13	0.13677	0.09966	43.28	0.17563	
	STGCN (Im)	0.03957	16.85	0.10221	0.07498	37.80	0.13579	0.09843	42.74	0.17399
	H-STGCN (1)	0.03870	16.31	0.10095	0.07380	37.07	0.13455	0.09750	42.32	0.17257
	H-STGCN	0.03861	16.28	0.10067	0.07254	36.31	0.13308	0.09528	40.82	0.17030

复合邻接矩阵。分析表 1 可知，和 STGCN 相比，STGCN (Im) 在 W3-715 数据集上有着更低的 MAE、MAPE，在 E5-2907 数据集上有着更低的 MAE、MAPE 及 RMSE，证明了复合邻接矩阵的有效性。

图 8 以 E5-2907 数据集为例，对不同邻接矩阵进行可视化。图中颜色代表的值为 $lg(w_{ij} + 1)$ (a) 为迪杰斯特拉矩阵，(b) 为协方差矩阵，(c) 为复合邻接矩阵。

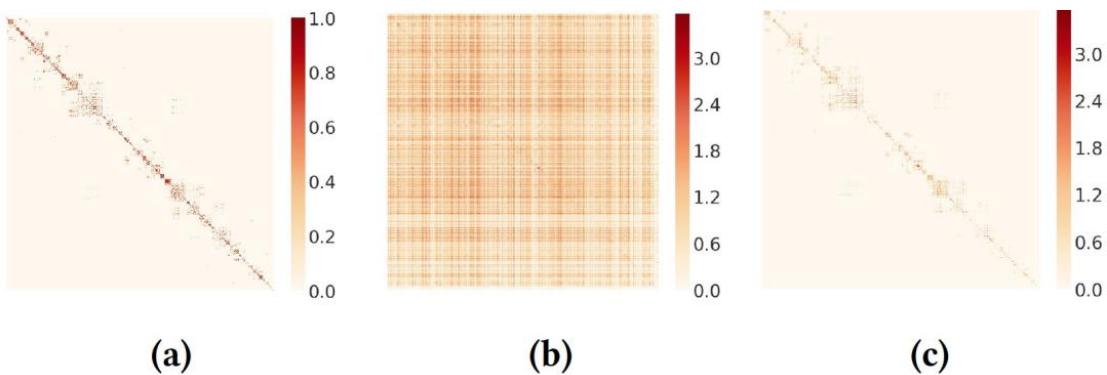


图 8 E5-2970 的各种邻接矩阵

未来流量特征和域转换器。如表 1 所示，和 STGCN (Im)相比，H-STGCN 有着稳定的更优表现，从而证实了利用未来流量数据带来的收益。由于域转换器中逐路段卷积结构的存在，H-STGCN 的模型表达能力是强于 STGCN (Im)的。为了消除这一影响以针对未来流量特征带来的收益做更公平的分析，我们进一步将 H-STGCN 与 H-STGCN (1)进行对比。

在测试集 C、测试集 NRC 上，不难发现未来流量特征在对拥堵的预测上有显著更优的表现。

如图 9 所示，随着预测时间跨度的拉长，未来流量特征带来的收益会起主导作用。

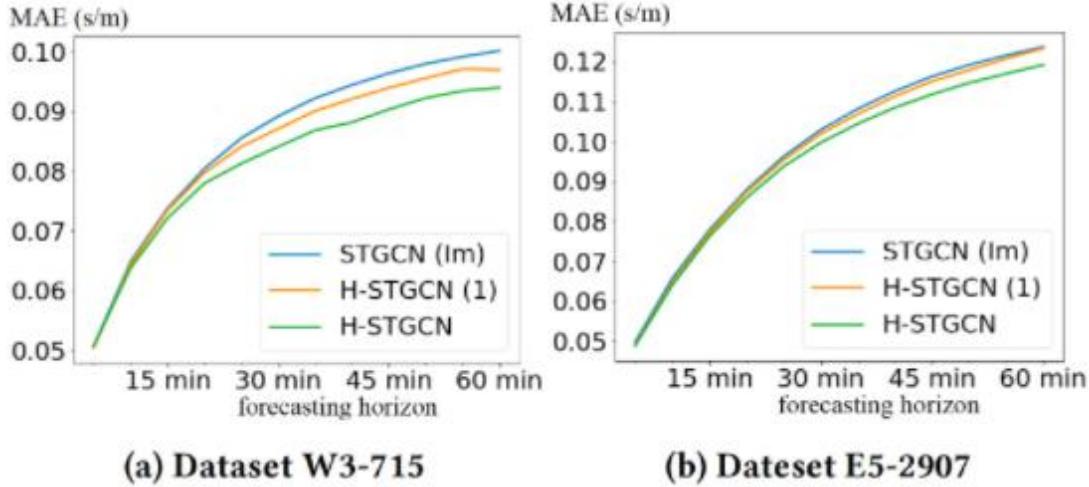
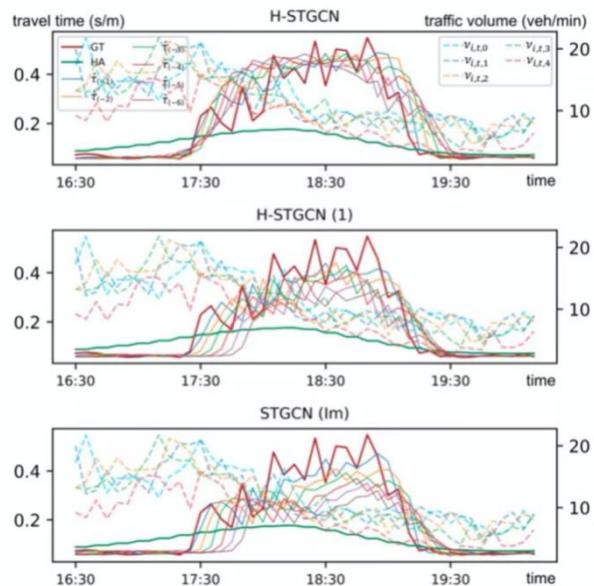


图 9 测试集 NRC 上效果比对

为了更加直观地对H-STGCN的原理加以剖析，我们这里展示一个突发拥堵预测的案例（如图10所示）。这个案例来自2018年4月16日某一高速路段。GT代表真值，HA代表历史均值， $\hat{\tau}_{(-f)}$ 是 f 个时间步以前对当前通行时间的预测值， $\nu_{i,t,f}$ 是对应 f 个时间步后的理想未来流量。

17:30至18:00拥堵加剧的阶段，H-STGCN (1) 提前多个时间步的预测结果和真值相比有明显的时间滞后。相比之下，H-STGCN由于有理想未来流量中的信息，甚至有能力在30min以前就对拥堵有着准确的预测。

我们可以这样理解这一现象： $\nu_{i,t,3}$ 对应的曲线代表了对15min之后交通流量的近似推算，该曲线在17:15就开始拉升了。基于导航引擎中只有当前时刻已经发起的导航行程这一事实，实际的未来流量要比理想未来流量更高。所以， $\nu_{i,t,3}$ 的飙升预示着有较大的交通流量正在涌来，这就使H-STGCN能够在没有历史数据做参考的情况下预知未来的拥堵。



模型可扩展性

模型在 W3-715 和 E5-2097 两数据集的预测时间不超过 100ms。为了在实际线上应用场景中平衡推演效率和预测效果，我们将城市路网切分成最多包含几千个路段的子路网，每个子路网在线上部署一个模型。

未来将在主动交通管理方面发挥重要作用

H-STGCN 已在高德驾车路线的旅行时间预测 (ETA)（见图 11）中落地[9,10]，并将偏差严重的案例数量降低了 15%。H-STGCN 首次以数据驱动的方式建模了用户出行意图与交通路况演化之间的相互作用，未来可以广泛的应用在主动式的交通管理领域，例如智能红绿灯调控[9]、智能道路收费系统[10]等。



图 11 ETA 预测结果的展示 本文提出了一种新的用于通行时间预测的深度学习框架：混合时空图卷积网络（H-STGCN），该框架利用从导航数据中推演出的计划中交通流量提升模型效果。在真实场景数据集上进行的实验证实 H-STGCN 和对标的模型相比取得了更优的效果，在突发拥堵的预测上优势尤为明显。

混合时空图卷积网络提供了一种将物理知识嵌入数据驱动模型的新范式，创新性地应用了复合邻接矩阵和域转换器结构，很容易推广到一般的时空预测任务当中，未来将在智能交通管理等领域发挥重要作用。

主要参考文献

- [1] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting.
- [2] YishengLv, YanjieDuan, WenwenKang, Zhengxi Li, Fei-Yue Wang, et al. 2015. Traffic flow prediction with big data: A deep learning approach. IEEE Trans. Intelligent Transportation Systems 16, 2 (2015), 865 – 873.

- [3] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio—Temporal Graph Convolutional Neural Network: A Deep Learning Framework for Traffic Forecasting. In Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI).
- [4] JingruiHe, WeiShen, Phani Divakaruni, Laura Wynter, and Rick Lawrence. 2013. Improving Traffic Prediction with Tweet Semantics. In Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI). 1387 – 1393.
- [5] Binbing Liao, Jingqing Zhang, Chao Wu, Douglas McIlwraith, Tong Chen, Shengwen Yang, Yike Guo, and Fei Wu. 2018. Deep Sequence Learning with Auxiliary Information for Traffic Prediction. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM.
- [6] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion convolutional recurrent neural network: Data—driven traffic forecasting.

- [7] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio—Temporal Graph Convolutional Neural Network: A Deep Learning Framework for Traffic Forecasting. In Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI).
- [8] Moshe Ben—Akiva, Michel Bierlaire, Haris Koutsopoulos, and Rabi Mishalani. 1998. DynaMIT: A simulation—based system for traffic prediction. In DACCORD Short Term Forecasting Workshop. Delft, The Netherlands, 1 – 12.
- [9] Wei, H., Zheng, G., Yao, H. and Li, Z., 2018. Intellilight: A reinforcement learning approach for intelligent traffic light control. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.
- [10] https://en.wikipedia.org/wiki/Congestion_pricing

CIKM 论文解读 | 破解高架区域偏航检测难题，高德提出工业级轻量模型 ERNet

作者：高德智能技术中心

导读

在本文中，高德在线引擎中心定位研发部对高架区域中的偏航检测难题进行了探究，并提出以一种真正工业级的轻量级神经网络模型 ERNet 来解决这一问题。实验结果表明，ERNet 在实际操作中取得了非常显著的效果。本文已被学术会议 ACM CIKM 2020 收录。

手机导航是手机地图中非常重要的功能。偏航检测（车辆是否偏离规划路线）是手机导航中至关重要的任务。传统偏航检测方法通过检测车辆的位置（低精度）或运动方向是否偏离规划路线，判断车辆是否偏航。

而在高架区域（包含高架桥和平行辅路）中，识别车辆是否偏航是非常困难的。因此，识别车辆是否在高架上行驶，能够极大地提升高架区域的偏航检测效果。

在本文中，高德地图在线引擎中心提出了高架道路网络 (Elevated

Road Network, ERNet），一个轻量级且真正工业级别的神经网络模型，用于解决高架区域中的偏航检测难题。

具体而言，对于同一组高架道路片段和平行辅路片段，ERNet 以 4 种高语义级别的特征作为输入，学习得到两个 10 维的道路描述子向量（A 和 B）。

在推断阶段，对于车辆的一个位置，ERNet 预测一个 10 维的 embedding 向量（C）。最后通过比较 $\|A-C\|$ 和 $\|B-C\|$ ，并结合置信度约束（confidence constraint），我们能够识别出车辆是否在高架上行驶。

在大量的实验中，ERNet 取得了很好的效果。目前，ERNet 已经在高德地图手机客户端上线，支持北京、上海和广州三个城市的高架区域偏航检测。

序言

手机导航是手机地图的核心功能。在高德地图（中国最流行的手机地图 APP），人们每天请求海量的导航服务。在一次旅程中，如果车辆偏离了导航服务为其推荐的最佳规划路线，手机导航系统需要及时地检测出这一偏航事件，并规划一条新的路线。

传统的偏航检测方法（在手机地图中）主要依赖 GNSS 系统提供的车辆位置和行驶方向信息。一般地，如果车辆的位置（或方向）距离规划路线连续地超过 T_d 米或(T_a 角度)，一次偏航事件可以被检测出，如下图 1 所示。

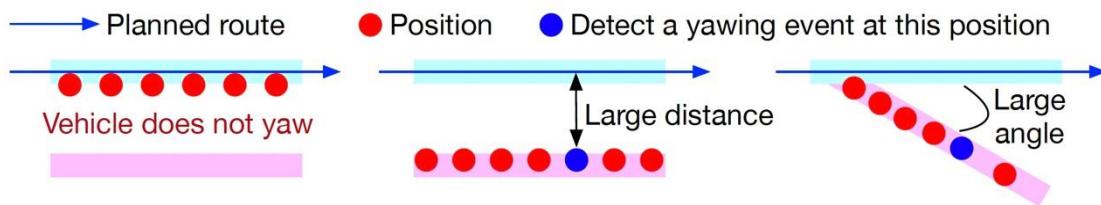


图 1：传统偏航检测方法可以在中间图（距离超出 T_d ）和右侧图（角度超过 T_a ）检测出偏航事件。

在中国，高架道路作为城市快速交通的关键组成部分，被大量建造。尤其在上海，有超过 622km 的高架道路，几乎覆盖全城。在高架道路附近，GNSS 系统提供的位置和行驶方向精度较低（可能误差会超过 20 米，如下图 2 所示）。

因此 T_d 和 T_a 必须足够大，否则偏航检测方法可能会误报偏航。这就导致在高架区域（高架桥附近存在较近的平行辅路）中，传统的偏航检测方法无法正确检测出偏航事件。

如果我们能够正确识别出车辆是否在高架桥上（高架道路识别），势必会给偏航检测方法带来质的提升。

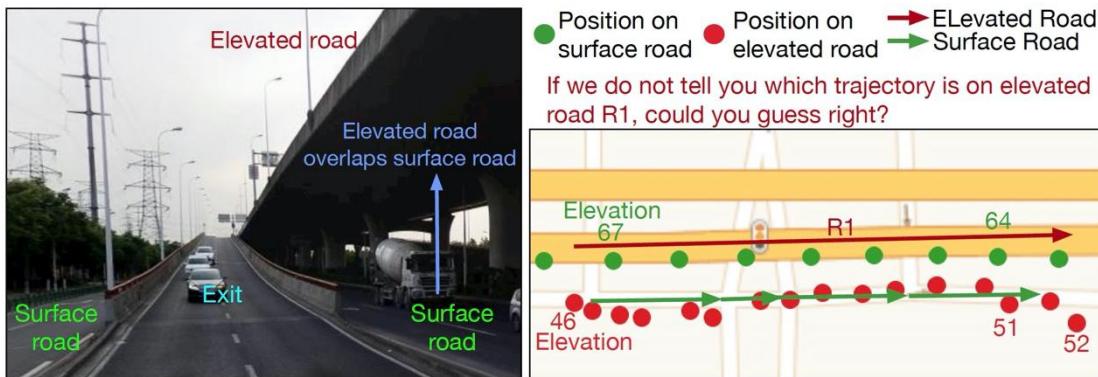


图 2：左子图：高架道路（Elevated road）跟旁边的辅路（Surface road）距离较近，而且方向平行。右子图：两条轨迹存在较大的水平位置误差。红色轨迹来自行驶在高架桥 R1 的车辆，但是其位置却落在辅路上。同样地，绿色轨迹来自行驶在辅路上的车辆，其位置落在高架桥上。需要注意的是，红色轨迹的位置（高架桥是水平的，没有高度差）的高度取值范围从 46 米到 52 米。而且，红色轨迹的高度取值小于绿色轨迹（真实在辅路上的位置）的高度取值。

现有高架状态识别方法的局限

目前存在的高架状态识别方法大致可以分为三类：地图匹配方法、高程计算方法和特征识别方法。

地图匹配方法使用 Map-Matching 技术[9, 10, 14]，考虑时序的车辆位置、方向、路网等信息，在路网中为车辆运行的轨迹选取合适的道路。这种方法在一般场景中工作良好，但是在高架区域表现糟糕。这主要是由于高架桥与平行辅路的方向一致，且距离很近，而地图匹配方法依赖的车辆位置和方向在高架区域精度较

低，不足以精确区分轨迹是否在高架桥上。而路网的拓扑信息在高架桥存在较多的出入口时，也无法精确建模，如下图 3 所示。



图 3：当路网拓扑信息在高架区域存在出入口时，由于 GNSS 位置精度较低，地图匹配方法很难确认一条轨迹是来自高架桥上行驶的车辆还是来自平行辅路上行驶的车辆。

高程计算方法主要使用高度或高度差信息，例如使用 GNSS 系统提供的高度或使用气压计测量值计算的高度差。不过，GNSS 系统提供的高度信息基本不可用[3]（如上图 2 所示）。

至于气压计，仅 iPhone 和少数高端的 Android 手机才装备。并且，气压计在车辆窗口打开后，读数非常不稳定（如下图 4 所示）。汽车速度越快，气压计读数越低，测量的高度越高。

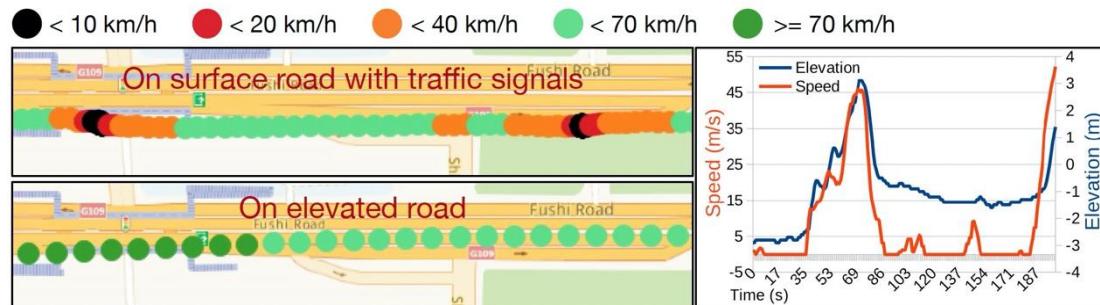


图 4：左子图：一辆车在平行辅路上频繁地加速和减速（上侧），在高架桥上保持较快的速度（下侧）；右子图：当车辆窗户打开后，气压计测量的高度变化在平面道路上出现了较大的差异（超过 7 米）。

特征识别方法提取能够区分车辆在高架桥上和桥下的平行辅路行驶的本质差异特征，并将这些特征作为模型（HMM 或分类器）的输入。

模型的输出为车辆的状态：行驶在高架桥上和行驶在高架桥下的平行辅路。方法 [2] 使用了非常简单的特征，例如车辆速度的方差、可追踪卫星（GNSS 系统由多颗卫星组成）的数量等。

然而，速度和卫星信号可以提供非常有意义的信息，用于捕获车辆行驶在高架桥上和桥下平行辅路的本质差异，见图 4 和图 5。因此更加复杂的特征应该被考虑进来。

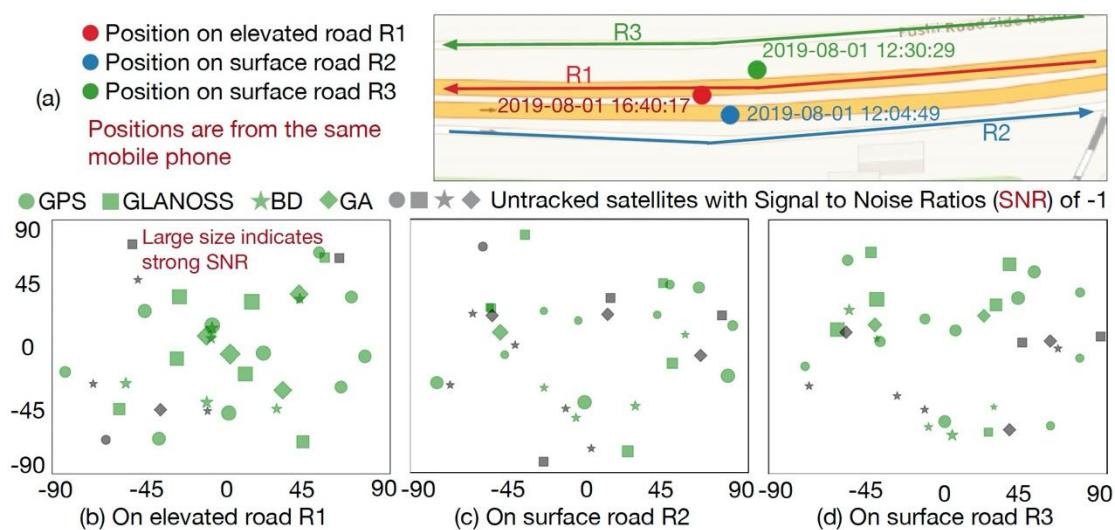


图 5：卫星信号在高架桥上和平行辅路上存在较大差异。 (a) 三个位置分别在高架桥 R1 上，在平行辅路 R2 和 R3 上。(b) 在 R1 上，大多数的卫星信噪比比较强（图标尺寸比较大），有 7 个卫星未被追踪到（无法用来进行 GNSS 定位）。(c) 由于 R2 北方被高架桥遮挡，北方的卫星的信噪比比较低，且 12 个卫星未被追踪到。(d) 跟 (c) 非常类似，南侧的卫星的信噪比比较低，且 9 个卫星未被追踪到。

如何设计有效的高架状态识别方法？

为了设计一个有效的高架状态识别方法，我们面临以下三个主要挑战：

1. 识别方法必须是轻量级且计算代价低。因为我们希望该方法能够在手机端运行，以便手机导航系统及时的检测出车辆的偏航事件；

2. 获取大量精确标记的样本（车辆是否在高架桥上）是非常困难的。由于我们面临的问题是一个全国范围内（甚至是全球范围内）的问题，因此标记的样本必须足够多才能覆盖如此大范围的场景；

3. 对卫星信号和速度建模同样比较困难，这主要是因为卫星信号和速度在不同时间、不同地点的差异非常大（特征空间巨大）。

近年来，以神经网络为代表的模型在多个任务中取得了极佳的效果，所以有足够的动机去尝试它们。

对于挑战 1，我们更倾向于使用轻量级（层数较少）的神经网络，而不是深度神经网络（有可能取得更佳的实验效果，但是计算代价巨大）；

对于挑战 2，我们可以使用车辆在手机导航过程中上传的 GNSS 信息（车辆位置、运动方向，卫星空间位置、卫星信噪比等），设计一个自动标注算法，标注出车辆实际行驶的道路序列。

对于挑战 3，我们需要设计一个有效的特征提取方法来处理卫星信号。此外，在许多局部区域，高架桥下接收的卫星信号和高架桥上接收的卫星信号差异巨大。而在某些局部区域，差异较小。例如，当高架桥下的平行辅路距离高架桥较远时，遮挡情况较弱，因此平行辅路上行驶的车辆也能接收到较强的卫星信号。

而在高楼林立的市区，即使车辆行驶在高架桥上，也会因为受到

高楼的遮挡，继而接收到的卫星信号较弱，如下图 6 所示。为了对这些区域差异进行建模，我们可以使用局部编码技术，为每个局部区域编码生成一个维度为 d 的 embedding 向量。

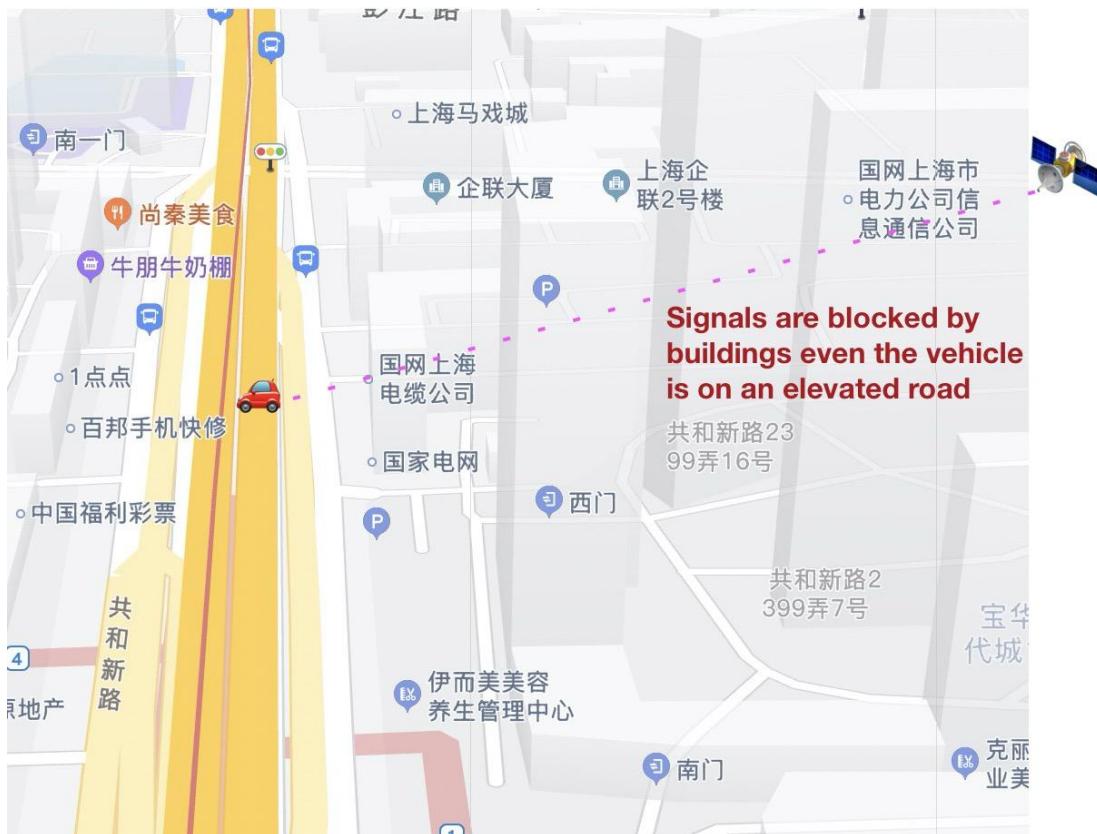


图 6：即使车辆行驶在高架桥上，在某些区域，卫星信号的传输会受到高楼的遮挡，因此车辆内手机接收到的卫星的信噪比和追踪到的卫星个数可能也会比较低（跟桥下行驶的车辆没有明显差异）。

同时，为了学习到有区分性的特征（同类样本在学习到的特征空间距离较近，不同类样本在学习到的特征空间距离较远），一些损失函数，例如 center loss [13]，triplet loss [11] 和 triplet-center

loss [5] 被提出。使用这些损失函数训练的技术称为 metric learning 技术。

高架道路识别是一个局部问题。对于每一个位置，我们都可以计算出其属于哪个局部区域。我们在学习过程中，只需要确保属于同一个局部区域的不同类别的样本在特征空间上距离较远，同一个布局区域的同类别的样本在特征空间上距离较近。

Softmax loss 则要求不同类别的样本都要在特征空间上距离较远，同类别的样本都要在特征空间上距离较近，而不管样本是否属于同一个局部区域。

根据之前的分析，研究者提出了一个特征识别方法。该方法基于 Elevated Road Network (ERNet)，一个轻量级而且真正意义上工业级别的模型，从根本上解决高架道路识别问题。

ERNet 在一个大的自动标注的训练样本上进行训练，并在一个手工标注的样本集上进行验证和测试。ERNet 输入 4 类高语义的特征，并使用 triplet loss 进行训练。ERNet 为同一个局部区域（我们称之为 group）的高架桥上片段和桥下平行辅路片段各学习一个 γ 维的道路描述子 A 和 B。在推断阶段，ERNet 为每一

个待预测的位置预测一个 γ 维的特征 embedding C。如果 $\|A-C\| < \|B-C\|$ ，则可以简单地认为车辆行驶在高架桥上。

不过为了提升识别的准确率，研究者提出了一个置信度约束的机制。本文同样提出了一个基于 ERNet 的偏航检测方法。

本研究的贡献可以总结为以下四点：

1. 提出了 ERNet，一个轻量级而且真正工业级别的模型，使用 triplet loss 进行训练。该模型为局部区域（group）的桥上和桥下路段学习道路描述子；
2. 使用 group embedding 技术建模卫星信号在不同局部区域的多样性；
3. 提出了一个新的特征卫星平面投影特征（Satellite Plane Projection，SPP），用于建模卫星信号分布；
4. 提出了一个基于 ERNet 的偏航检测方法。该方法已经成功应用在高德地图（上线北京、上海、广州三个城市）。

ERNet 模型详解

离线数据处理

在一次旅行中，为了请求导航信息（例如规划路线、交通拥堵信息），手机导航系统会上传 GNSS 信息到我们的地图服务器。因此地图服务器拥有大量的 GNSS 信息和规划路线信息，这些被称为原始数据。

由于研究者希望使用 group embedding 技术，所以需要将原始数据转换为 group 数据。group 数据的每个实例包含：group ID、高架道路距离、GNSS 信息和道路类别。

具体地，研究者首先提炼出车辆真正行驶的路线 travel route，然后将车辆行驶轨迹的每个点投影到 travel route 上计算 group ID、高架道路距离和道路类别。前者他们称为 route refinement，后者称为 group projection。

Route refinement

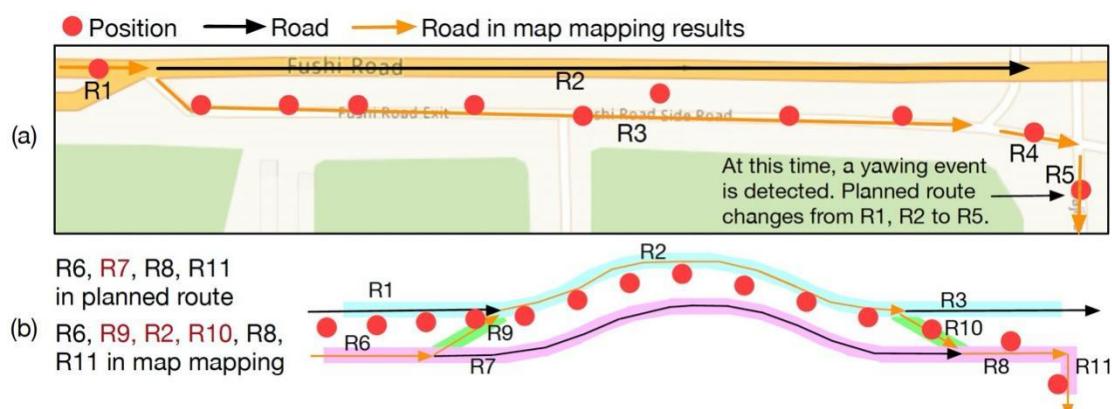


图 7：(a) 地图匹配方法将轨迹点匹配到邻近的道路上（黄色道路）。

(b) 一辆车行驶在 R6、R7、R8 和 R11 上，并没有偏航（规划路线也是如此）。

在一次旅程中，如果车辆偏航，将会有不止一条规划路线。由于 travel route 无法直接合并多个规划路线，如上图 7 (a) 所示。因此研究者需要提炼出 travel route。

具体地，研究者使用地图匹配方法将一条轨迹的全部位置匹配到附近的道路上，从而得到一组连续的道路，见图 7 (a)。由于地图匹配方法使用的是低精度位置，因此匹配道路可能是错误的。例如，图 7 (b) 中，travel route 实际上包含 R6、R7、R8 和 R11。而地图匹配的结果为 R6、R9、R2、R10 和 R8。其中 R9、R2 和 R10 并不是车辆实际行驶的道路。

在高架区域中，对于平行的高架桥上道路和桥下的平行辅路，如果存在地图匹配道路和规划路线道路不一致的问题，则更倾向于相信规划路线的道路。这主要因为在高架区域，仅有 3% 的车辆会偏航（而地图匹配的错误率要远高于 3%）。

例如，图 7 (b) 中，更愿意相信车辆行驶在规划路线中的 R7 上，而不是地图匹配路线中的 R9、R2 和 R10。当然，相信规划路线

会导致大约 3% 的道路并不是车辆真正行驶的道路（相当于训练样本存在 3% 的噪声数据）。不过，这对于神经网络而言，仍然可以训练得到精度较高的模型，只要能够更好地处理过拟合问题。

Group projection

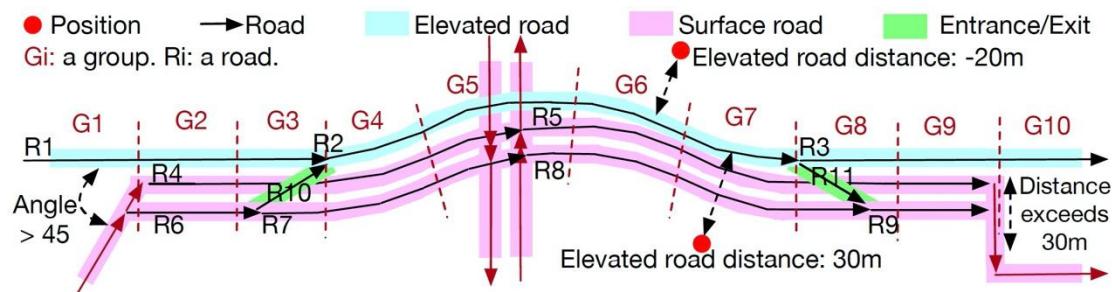


图 8：我们将一条高架道路切分为多个 group。每个 group 的长度接近 α 米。每个 group 包含一个高架桥片段，和一个或多个平行辅路片段。如果一个 group 内的平行辅路片段到高架桥片段的距离超过 30 米或者夹角大于 45 度，我们舍弃掉这个 group，例如 G1 和 G10。

问题的关键在于如何为每个位置计算其所属的 group。上图 8 显示将一个高架桥切分为多个 group。每个 group 的长度为 α 。对于一次旅程的轨迹，我们将全部位置顺序的投影到 travel route 上，寻找每个位置最佳的匹配道路，如下图 9 所示。

如果最佳匹配道路即为高架道路，那么投影距离即为高架道路距离，投影处对应的 group 即为高位置所属的 group。

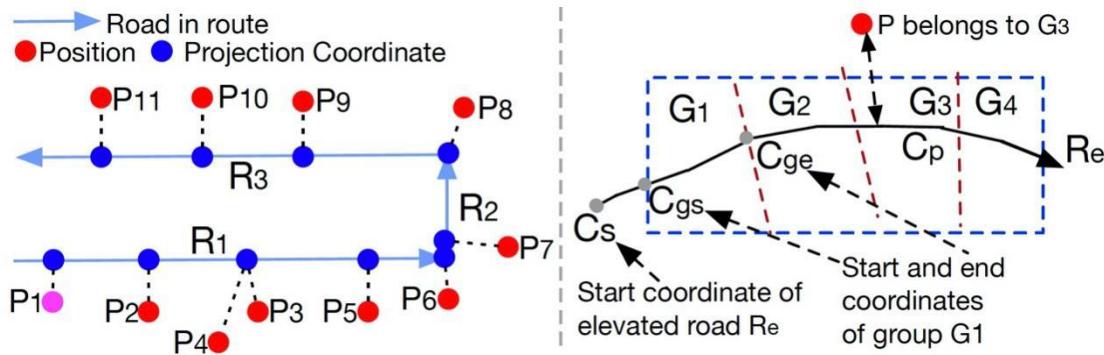


图 9：左子图：位置被顺序地投影到 travel route 上。每个位置匹配到最近的近似平行道路上（道路的方向和运动方向夹角接近）。注意，对于回退位置（例如 P4），研究者将其投影到 P3 投影的道路上；右子图：一个 group 的范围被表示为 $[G_s, G_e]$ ，这里 G_s 或 G_e 指的是 C_s 到 C_{gs} 或 C_{ge} 的距离。

对于最佳匹配道路为平行辅路的位置，研究者使用一个叫 road map 的结构。road map 的每个实例为平行辅路的 ID 和对应的高架桥道路的 ID。这里对应指的是平行辅路和高架桥是同向而且是临近重叠的。例如，在图 8 中，平行辅路 R6 的对应高架桥道路 ID 为 R1，而平行辅路 R7 对应的高架桥道路 ID 为 R1 和 R2。使用 road map，最佳匹配道路为平行辅路时，我们获取对应的高架桥道路，并将位置投影到高架桥道路上，获得高架道路距离和 group ID。

特征

ERNet 使用 4 类特征：1) 卫星平面投影特征 (Satellite Plane

Projection, SPP) ; 2) group ID; 3) 高架道路距离; 4) 序列速度特征。

SPP 特征捕获了卫星信号分布。在高架桥上接收和桥下平行辅路接收的卫星信号分布差异很大, 如图 5 所示。group ID 可以提供局部信息。大的高架道路距离表示车辆行驶在距离高架桥较远的平行辅路上(没有卫星信号遮挡)或行驶在距离高架桥较近的平行辅路上(卫星信号遮挡验证, 因此位置飘逸严重)。

下图 10 (c) 显示了序列速度特征的提取方式。尽管每个 bin 长度设置地更小能够提供更丰富的内容信息, 用来区分车辆加减速是由于红灯导致还是由于交通拥堵导致。当速度较快时, 一些 bin 可能没有位置数据。为了避免这种情况, 研究者设置每个 bin 的长度为 30 米(车辆速度低于 108km/h 时, 每个 bin 都存在位置数据)。

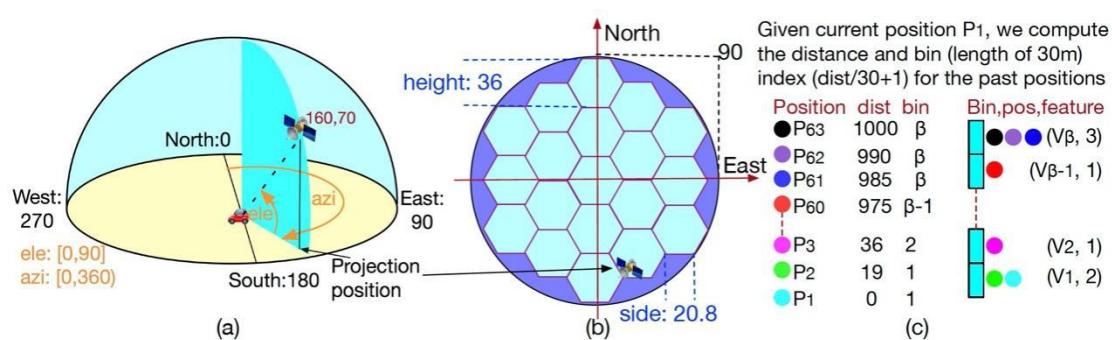


图 10: (a) 卫星的空间位置是(160, 70), 160 是指方位角, 70 指俯仰角; (b) 卫星的投影平面被切分为 19 个正六边形; (c) 时序速

度特征包含 β 个 bin。其中每个 bin 包含属于该 bin 的位置的平均速度和位置的个数。

对于每个位置，我们能够非常容易计算出其所属的 group、高架道路距离和序列速度特征。因此这里重点讨论下 SPP 特征。

具体地，研究者将投影平面切分为 19 个正六边形（图 10 (b) 所示）。六边形网格切分空间在地图领域经常被使用[12]。卫星的空间位置 (azi, ele) 对应到投影平面的坐标 p 可以按照如下方式计算：

$$p_x = \sin(\text{radian}(azi)) * (90 - ele), p_y = \cos(\text{radian}(azi)) * (90 - ele).$$

对于每一秒接收到的卫星信号，在每个六边形 H 中，研究者为每类卫星 T 计算 3 维特征：

$$f_1 = \sum_{s_i \neq -1} w_i, \quad f_2 = \sum_{s_i = -1} w_i, \quad f_3 = \frac{\sum_{i \neq -1} w_i \times s_i}{f_1 \times m}$$

这里 s_i 表示卫星 i 的信噪比（信噪比数值越大，则卫星信号越强）。m 代表手机的 GPS 接收机能够接收到的该类卫星 (T) 的最大信噪比（可以统计最近 20km 内，该类卫星 T 接收到的

最强 SNR）。 W_i 是高斯加权，通过如下方式计算：

$$\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{d_i^2}{2\sigma^2}\right)$$

这里是卫星 d_i 的投影坐标 p 到六边形 H 中心的距离。研究者设置 σ 为六边形边长的 2 倍。

对于一个六边形 H ，一个大的 f_1 (f_2) 表示更多可追踪（不可追踪）卫星的投影坐标更接近 H 的中心，这捕获了卫星的位置分布。 f_3 是卫星信噪比的加权平均值（使用最大卫星信噪比标准化）。

大的 f_3 表示投影坐标靠近 H 中心的卫星的信噪比接近该类卫星的最大信噪比。因此 f_3 捕获了卫星的信号强度分布。

由于研究中使用了 4 类卫星，因此 SPP 特征的维度总数为 $4*3*19 = 228$ 。SPP 特征是高语义级别的特征而且计算非常高效，因此非常适合手机设备。

网络结构

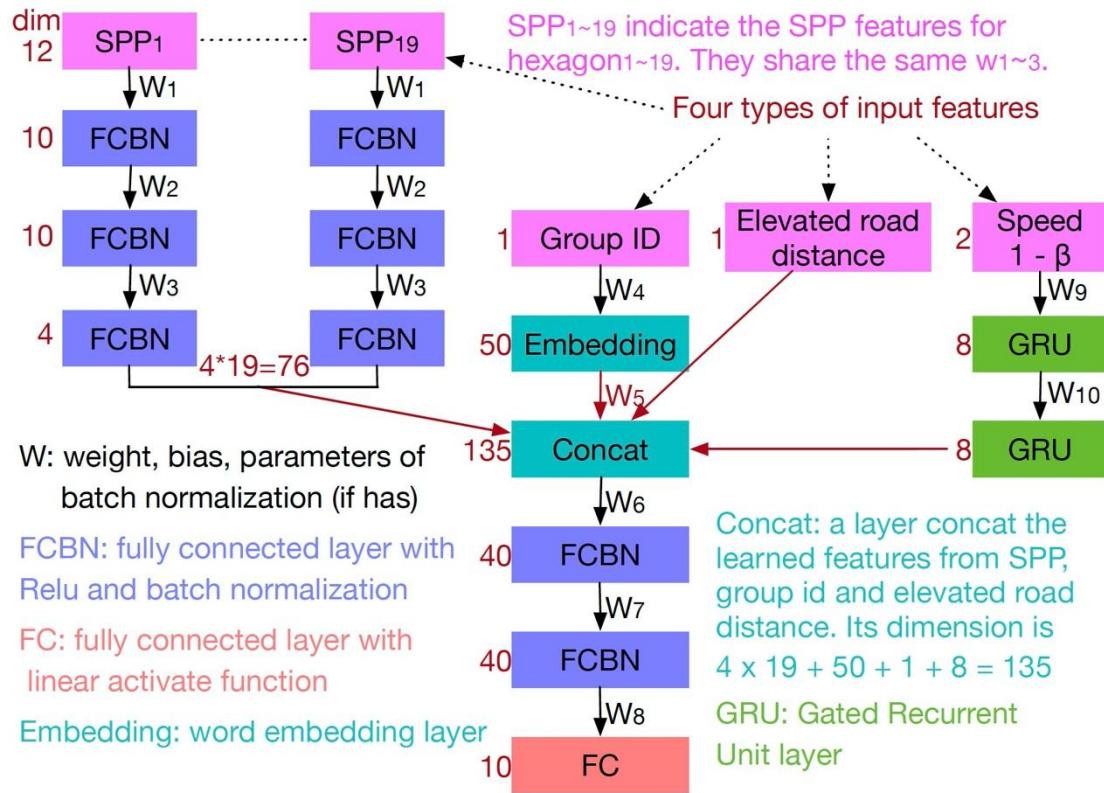


图 11：ERNet 包含 10 个权重层。SPP 特征在不同六边形的特征分量共享 w_1 、 w_2 和 w_3 。

ERNet 被设计为在手机设备上运行的轻量级网络，如上图 11 所示。对于 group ID 特征，研究者使用 group embedding 技术将 group ID 编码为一个 50 维的 embedding 向量。高架道路识别可以直接使用 softmax loss 训练一个二分类模型。

不过，高架道路识别是一个局部问题，可以为每一个待识别的位置计算其所属的局部区域（也就是 group）。因此只有属于同一个 group 的不同类别的样本才需要在特征空间存在区分度，如下

图 12 所示。Triplet 和 triplet-center loss 非常适合解决局部问题。

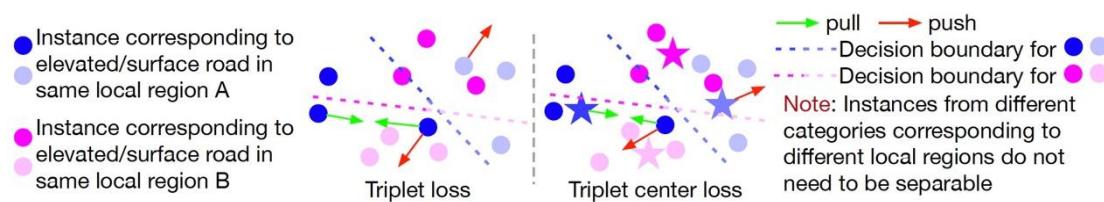


图 12：不像传统的损失函数 Softmax 强制要求所有不同类别的样本在特征空间分开，triplet 和 triplet-center loss 可以做到只要求属于同一个局部区域的不同样本在特征空间分开，因此更适合解决局部问题。

由于在实验中，研究者发现使用 triplet loss 训练的效果比使用 triplet-center loss 训练的效果更好，因此本研究更倾向使用 triplet loss 训练 ERNet。

研究者使用 Batch All [6] 策略在 group 内部构造 triplet 样本。所谓 group 内，指的是任意一个 triplet 的 anchor, positive 和 negative 样本均属于同一个 group。ERNet 为每一个 group 学习得到一个 γ 维的桥上道路描述子(属于该 group 的桥上训练样本的 ERNet 的特征 embedding 的均值)和 γ 维的桥下道路描述子。本文 γ 被设置为 10。

此外，考虑到卷积神经网络的参数共享机制的成功，研究者对

SPP 特征也使用了参数共享机制。具体地，每个六边形的输入 SPP 特征（12 维）单独通过 3 个全连接层（ w_1 、 w_2 和 w_3 ）。19 个六边形的 SPP 特征共享相同的 w_1 、 w_2 和 w_3 。SPP 的参数共享不仅能够带来更高的性能，同样能够极大的降低计算代价，详见对比实验部分。

置信度约束

对于一个 group G，ERNet 学习一个 γ 维的桥上道路描述子 A 和桥下道路描述子 B，见下图 13。在预测阶段，对于一个属于 G 的位置，ERNet 预测得到一个 γ 维的特征 embedding C。如果 $\|A-C\| < \|B-C\|$ ，该位置可以被识别为位于高架道路上。否则，位置可以被识别为在桥下。尽管这样做，ERNet 已经能够取得较高的准确率，但是对于工业级别的应用而言，准确率仍然略有不足。

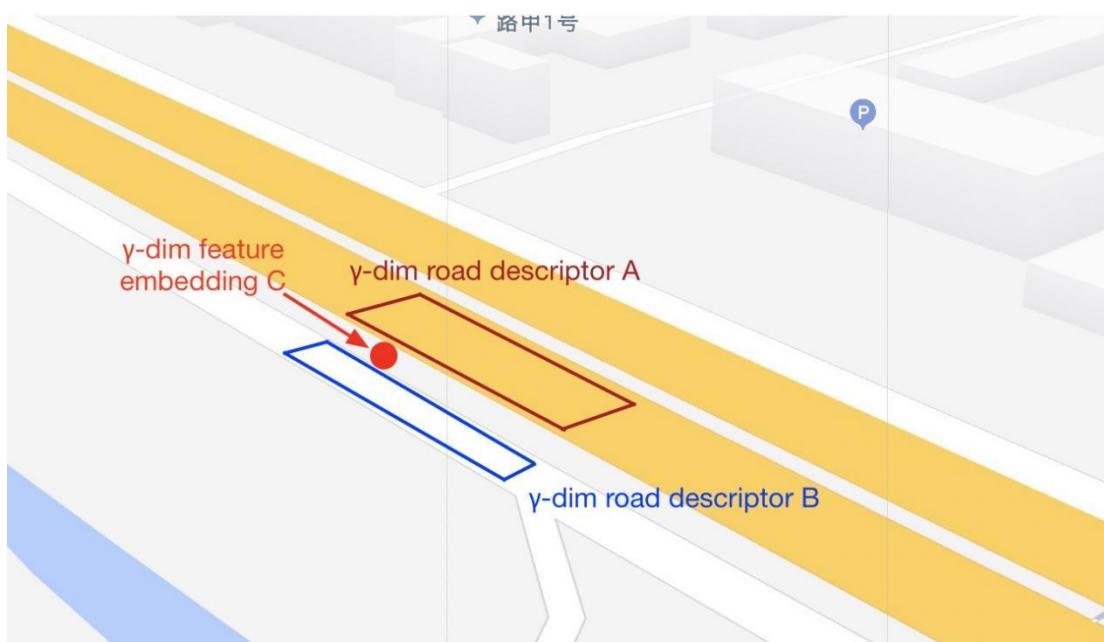


图 13: ERNet 为每个 group 学习桥上道路描述子 A 和桥下道路描述子 B。预测阶段，ERNet 为每个位置预测特征 embedding C。

因此，研究者提出使用 confidence constraint（置信度约束技术）提供 ERNet 的预测准确率（98.6% 到 99.2%，详见实验部分）。具体地，在训练集中，他们为每一个样本的预测结果计算 confidence：

$$\|A - C\|_2^2 / (\|A - C\|_2^2 + \|B - C\|_2^2)$$

置信度等于 0 (或 1) 表示该位置更像是在高架桥上 (或高架桥下)。在一个 group 中，我们将预测结果进行排序。在每一个预测的 confidence c 上，计算

$$F_{\beta}\text{-score} = \frac{(1+\beta^2) \times p \times r}{\beta^2 \times p + r}$$

这里 $p = L / M$, $r = L / N$ 。N 是类别为高架桥上的样本个数，M 是 confidence 小于 c 的样本的个数，L 是 M 中类别为高架桥上的样本个数。 β 是用来平衡 p 和 r 的值。 β 越小，表示越看重准确率。本文中，研究者设置 β 为 0.25，因为希望准确率更高些。

对应最大 $F_{\beta} - \text{score}$ 的置信度值即为 group G 的桥上约束值 V_e 。在推断阶段，只有样本的置信度小于 V_e ，样本才能被识别为桥上。类似地，group G 也有一个桥下约束值 V_s 。在推断阶段，只有样本的置信度大于 V_s ，样本才能被识别为桥下。如果样本既不能识别为桥上，也不能识别为桥下，那么样本被认为是未知状态。

下图 14 是对置信度约束的一个形象描述。所谓置信度约束，可以认为是为桥上的道路描述子和桥下的道路描述子各加了一个置信度半径。对于桥上样本，样本的特征 embedding 不仅需要到桥上的描述子更近，而且需要在桥上描述子的置信度半径内。

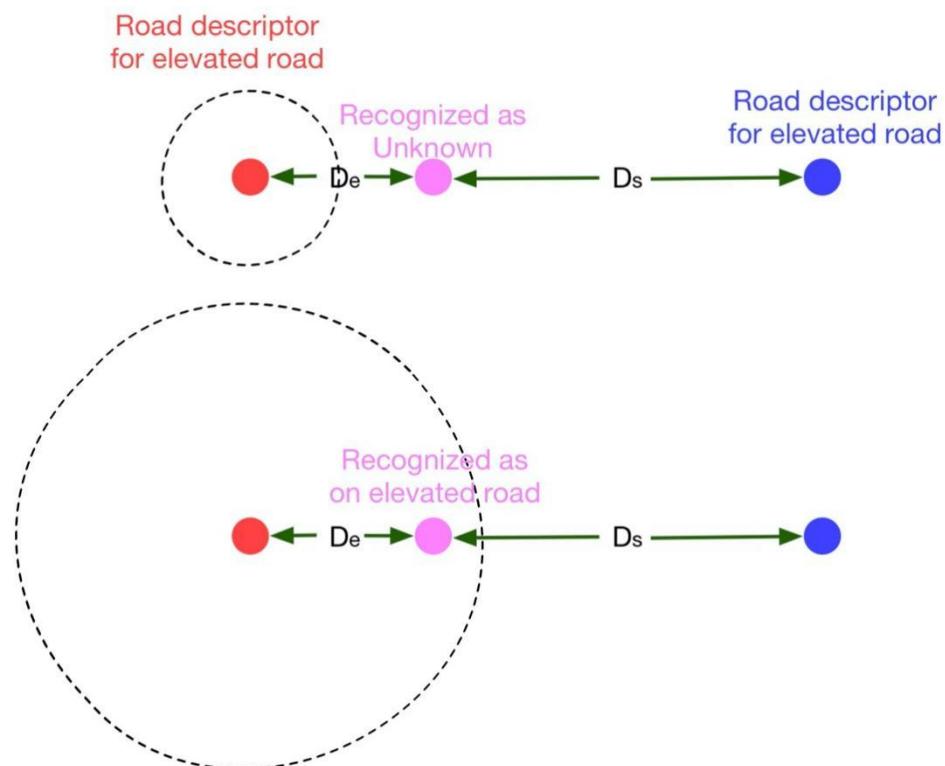


图 14：置信度约束可以被视为在道路描述子外侧加上置信度半径。
一个位置被识别为在桥上，不仅需要该位置的特征 embedding 距离

桥上道路描述子更近，而且需要在桥上道路描述子的置信度半径内。

基于 ERNet 的偏航检测方法

研究者基于 ERNet，研究者提出了一个新的偏航检测方法。该检测方法运行在手机端。在导航过程中，ERNet 会不断地预测车辆当前的状态（行驶在高架桥上还是桥下）。在一个没有重叠的 k 秒的时间窗口内，如果车辆规划行驶在桥上 / 桥下，而 ERNet 连续识别车辆行驶在桥下 / 桥上，一个偏航事件即可被检测出来。

下图 15 是一个偏航检测的示例：车辆规划行驶在高架桥上，但是车辆实际行驶在高架桥下的平行辅路。ERNet 连续识别出车辆行驶在桥下，因此一次偏航事件即可以被检测出来。

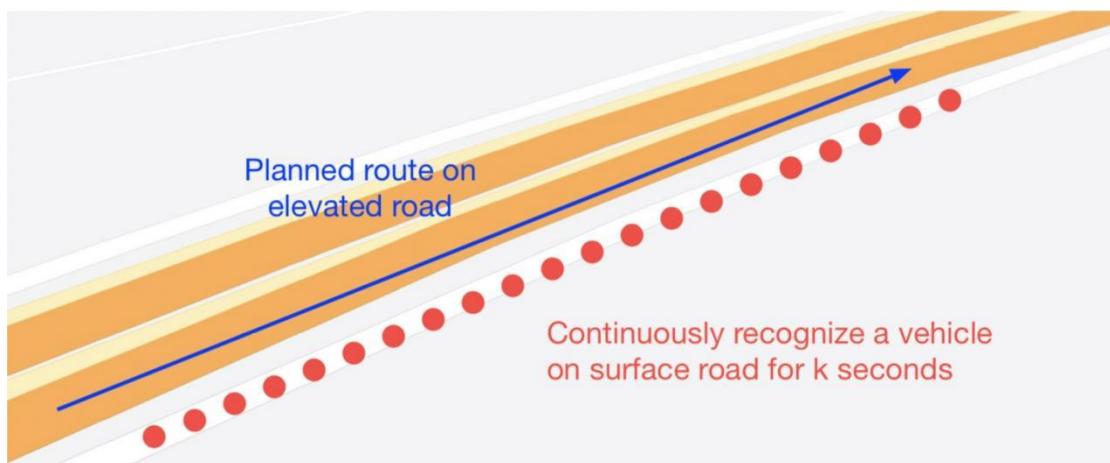


图 15：规划路线为桥上时，ERNet 连续 (k 秒) 识别车辆行驶在桥下，一次偏航事件被检测出。

实验

研究者在北京、上海和广州三个城市，测试了高架道路网络在高架道路识别和偏航检测两个任务上的效果。下图 16 为北上广三个城市高架区域（绿色矩形框内）。

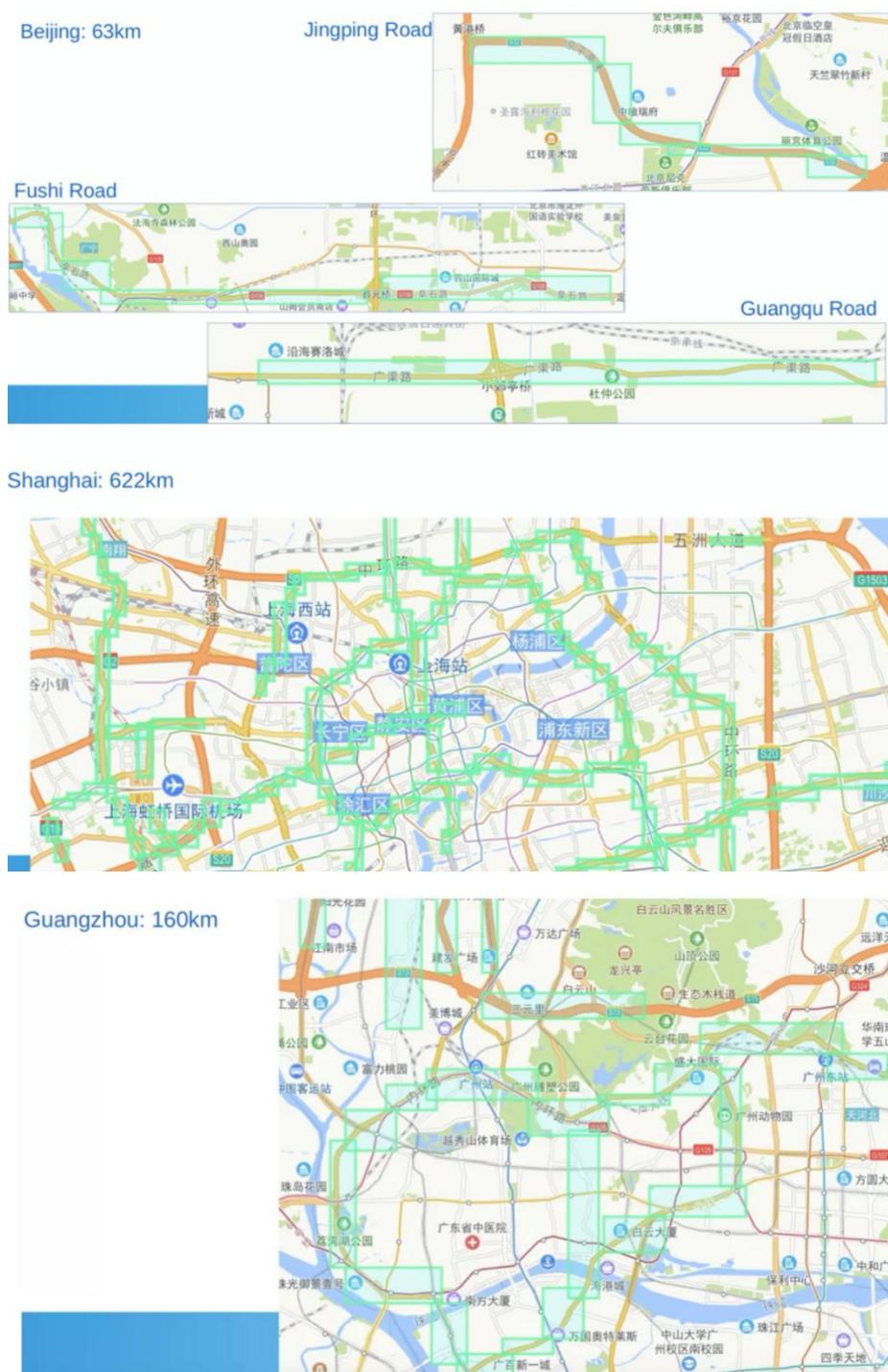


图 16：北京、上海、广州三个城市的高架桥区分布。

数据集

由于暂无公开的数据集，因此研究者在 Android 手机上搜集数据，构造本文的训练数据集和测试数据集。对于 iPhone 手机，无法获取到卫星的底层信号（如卫星的空间位置和信噪比等），因此本文方法只针对 Android 手机适用。

训练数据集

研究者使用一个非常大的训练数据集。该数据集的类别是自动标注的，存在大约 3% 的标注类别噪声。自动标注算法总共标注了数十亿个样本（每个样本对应一个位置）。对于每一个 group，研究者只采样最多 4000 个样本用于训练。当然，如果一个 group 的样本仅包含桥上或桥下类别，则会舍弃该 group。训练数据总共包含约一亿个样本。

测试数据集

研究者的测试数据集按照轨迹组织。70% 的轨迹用作评估集，30% 用作验证集。每个实例包含：A) SPP 特征、高架道路距离、group ID 和序列速度特征；B) 道路标注类别（高架桥上 / 高架桥下）；C) 经度、纬度、高度、速度和运动方向；D) 规划道路类别（规

划行驶桥上 / 规划行驶桥下）。A、B、和 C 用来评估特征识别方法和匹配识别方法。D 用来评估偏航检测方法。

为了构造测试数据集，研究者同样搜集 Android 手机的 GNSS 和规划路线数据，跟构造训练集类似。

但是，为了确保真值的准确性，研究者手动标注车辆行驶的道路。标注参考车辆行驶的速度、交通信号灯、路网拓扑等信息。尽管标注也会存在一些误标注数据，但总体上要更为可靠些。

由于北京、上海和广州三个城市的高架道路长度分别是 63km、622km 和 160km，因此测试数据集的轨迹在三个城市的分布为 1：10：3。如果一条轨迹包含标注道路类别和规划路线道路类别不一致的位置，则称该轨迹为偏航轨迹。

评估协议

对于高架道路识别任务，其目标为识别为车辆是否在高架桥上行驶。研究者定义识别准确率 $p=T / (T + F)$ 和识别有效率 $r=(T+F) / (T + F + U)$ 。T 为正确识别的位置个数，F 为错误识别的位置个数，U 为识别为未知状态的数量。下图 17 中对应的 $p=10 / 12$, $r = 12 / 14$ 。

	Real road type (manual label)	ERNet's recognition result
◆	Elevated road	Elevated road ✓
◆	Elevated road	Surface road ✗
◆	Elevated road	Unknown ?
●	Surface road	Elevated road ✗
●	Surface road	Surface road ✓
●	Surface road	Unknown ?
Recognition result:		◆ ◆ ◆ ◆ ? ◆ ◆ ◆ ◆ ? ✓ ✗ ✗ ✓ ? ✓ ✗ ✗ ✓ ✓ ✓

图 17：14 个位置中，10 个位置被正确识别，12 个位置存在识别结果。

对于偏航检测，研究者考虑偏航准确率 yaw_p 和偏航召回率 yaw_r 。其中 yaw_p 为正确偏航的次数除以总共偏航的次数， yaw_r 为正确偏航的轨迹数除以偏航轨迹的数量（注意，一条轨迹仅偏航一次，因为线上实际应用时，偏航后会立即重新规划路线）。

参数实验

ERNet 有三个超参数： α 、 β 和 γ 。 α 代表 group 的长度， β 代表序列速度的 bin 数量， γ 代表道路描述子的大小。基于 ERNet 的偏航检测方法有一个超参数： k ，表示无重叠时间窗口的大小。

在验证测试集上，经过多组参数对比实验，如下图 18，研究者选择了最优的参数组合 $\alpha = 20, \beta = 35, \gamma = 10, k = 15$ 。

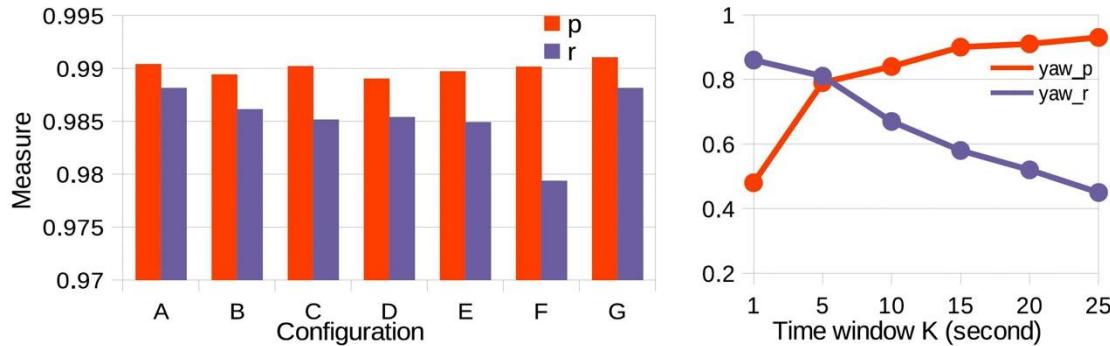


图 18：(a) α 、 β 和 γ 在 A: 20、35、10，在 B: 10、35、10，在 C: 40、35、10，在 D: 20、20、10，在 E: 20、50、10，在 F: 20、35、5，在 G: 20、35、20；(b) 我们设置 k 为 1、5、10、15、20、25。

高架状态识别实验结果

研究者用如下方法对比 ERNet 在高架状态识别的实验结果：

- ERNet-1：ERNet 不使用 group ID 特征；
- ERNet-2：ERNet 不使用 SPP 特征；
- ERNet-3：ERNet 不使用序列速度特征；
- ERNet-4：ERNet 不使用置信度约束；
- ERNet-5：ERNet 不使用 SPP 的参数共享机制；
- SoftmaxNet：ERNet 使用 softmax loss 训练；
- TCNet：ERNet 使用 triplet-center loss 训练；

- XGB：使用 XGBoost [1] 训练一个模型，使用 SPP，高架道路距离，序列速度特征三类特征。没有使用 group ID 特征。树的总数为 30 颗，深度为 20；
- Map-matching：地图匹配方法；
- GNSS-rule[2]：论文设计的直接基于 GPS 高度的方法；
- Bar-rule：基于气压计的方法。注意该方法仅在出入口生效；
- DEEL[2]：论文提出的一个基于 HMM 模型的特征识别方法。

GNSS-rule 和 DEEL 是论文 [2] 提出的方法，在论文 [2] 的私有数据集上进行评估。

该数据集仅包含 4 个 Android 手机在上海的数据。Bar-rule 是研究者实现的一个基于气压计的方法，其在他们私有的 iPhone 数据集上进行评估。其余的方法均在评估测试集上进行评估。

Methods	$p(\%)$	$r(\%)$	$p_e(\%)$	$r_e(\%)$	$p_s(\%)$	$r_s(\%)$
ERNet	99.2	98.3	99.4	98.4	98.8	98.3
ERNet-1	99.3	94.7	99.4	94.3	99.3	95.5
ERNet-2	98.6	94.2	98.7	94.4	98.5	93.9
ERNet-3	98.2	95.7	98.5	96.1	97.7	95
ERNet-4	98.6	100	99	100	97.9	100
ERNet-5	98.9	98.2	99.1	98.6	98.5	97.8
SoftmaxNet	98.3	100	98.9	100	97.3	100
TCNet	99.2	94.4	99.2	93.8	99.2	95.4
XGB	95.1	100	94.9	100	95.6	100
Map-matching	95.1	100	98.7	100	88.8	100
Bar-rule	90	20	89	21	90	20
GNSS-rule	\	\	64	100	57	100
DEEL	92	100	93	100	91	100

表 1：高架道路识别的对比方法实验结果。

表 1 显示 ERNet 获得了最优的效果。对比 ERNet-1, ERNet 的 r 提升了 3.6%，而 p 仅降低了 0.1%，这证明了 group ID 的有效性。对比 ERNet-2, ERNet 的 r 提升 4.1%，p 提升 0.6%，同样证明了 SPP 特征的有效性。对比 ERNet-3, ERNet 的 r 提升了 2.6%，p 提升了 1%，序列速度特征同样很重要。

不使用置信度约束时，ERNet-4 的 r 是 100%，但是 p 为 98.6%。不过研究者更希望准确率能够有足够大的提升，因此置信度约束也是有必要的。对比 ERNet-5，ERNet 的 SPP 特征的共享机制不仅轻微提升了 p 和 r，而且 ERNet'的计算代价仅为 ERNet-5 的 1/5。

研究者比较 ERNet-4 和 SoftmaxNet，可以发现使用 triplet loss 训练使得 p 略微增加 0.3%。不过对于桥下样本的 p 增加为

0.6%。因此 triplet loss 的引入也是有一定价值的，能够体现出高架道路识别局部问题的属性。TCNet 的效果较差，这值得进一步的讨论分析。

至于其余的对比方法，效果明显弱于 ERNet 和 ERNet 的变种。

下图 19 显示了 ERNet 在 10 款流行手机和 3 个城市的实验效果。可以看出，ERNet 对于不同机型和不同城市的性能比较稳定。注意横轴的刻度是非常细的。

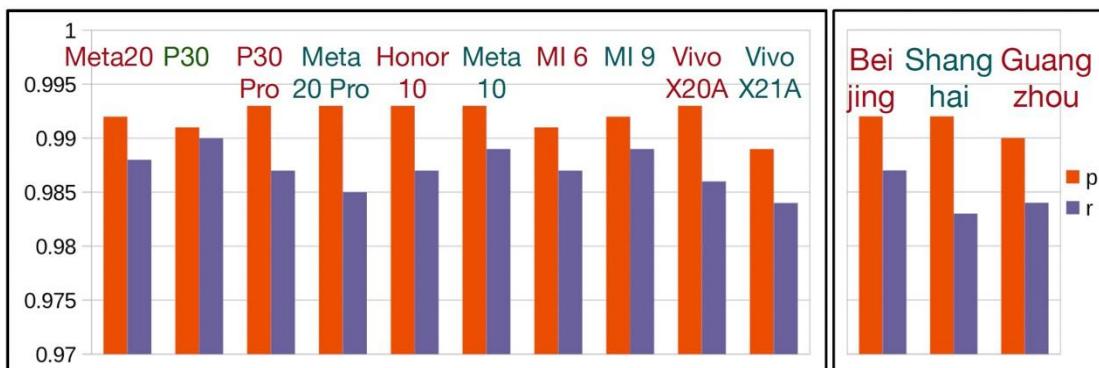


图 19：ERNet 在 10 款流行手机（左子图）和 3 个城市（右子图）的实验结果。

下图 20 展示了两个识别例子。图 20(a)是一个 good-case。ERNet 的识别结果是非常好的，即使在入口处；而图 20 (b) 则是一个 bad-case。车辆从高架区域外的道路驶入高架区域，中间没有经过入口。

按照常识，此处应该全部识别为高架桥下。但是 ERNet 并没有考虑道路的拓扑关系（道路的联通性、出入口等），后续可以考虑引入道路的拓扑关系解决这类 bad-case。

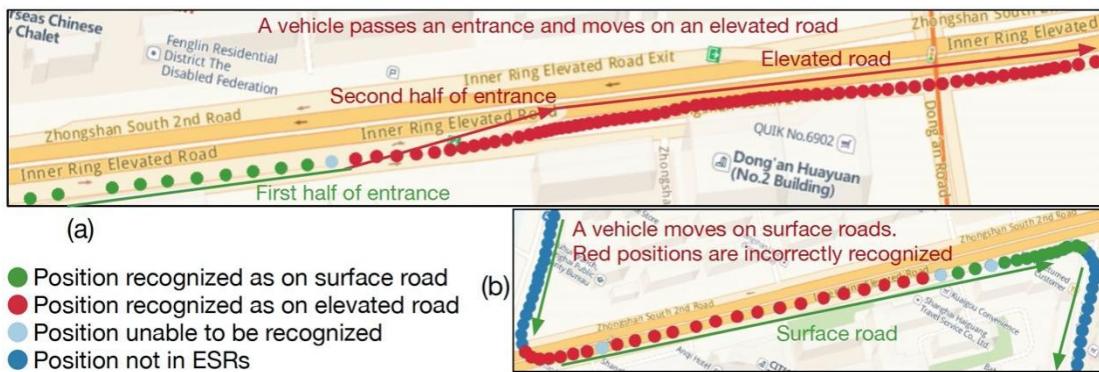


图 20: (a) ERNet 输出稳定的识别结果，即使在入口处；(b) Bad-cases

也许可以通过引入道路拓扑关系解决（此处没有入口，因此车辆不可能在高架桥上行驶）。

T-SNE 可视化

卫星信号特征的特征空间非常巨大，因此较难学习到区分度非常强的特征表示（76 维的卫星信号表示，SPP 参数共享的最后一层），下如图 21 (a)。而结合其它三类特征后，学习得到的道路描述子的可分度非常强，如图 21 (b)。

此外，高架道路识别是一个局部问题，ERNet 在学习过程中不要求属于不同 group 的同类样本距离彼此非常近，而只要求同一个 group 内的同类样本在特征空间上靠近彼此，如图 21 (c)。

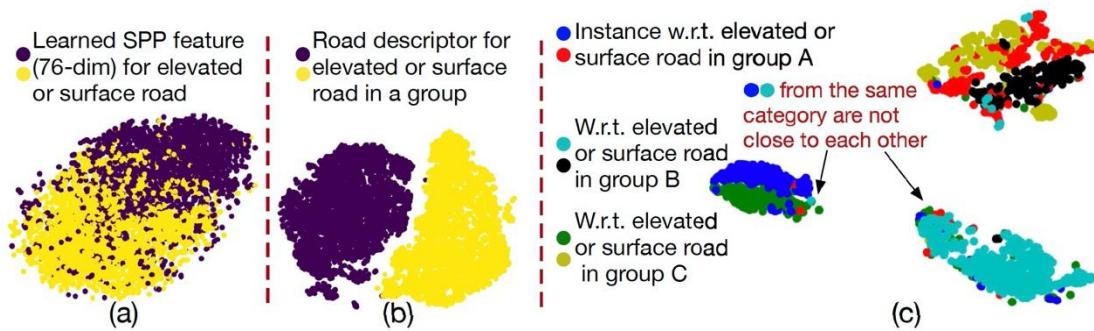


图 21：单独依靠 SPP 特征，无法学习足够区分度的特征 (a)。而结合其余三类特征后，ERNet 能够学习出足够区分度的道路描述子 (b) 和特征 embedding (c)。

偏航检测实验结果

研究者对比基于气压计的偏航检测方法和传统的偏航检测方法（高德线上偏航算法）。气压计的偏航检测方法容易受车辆开窗关窗影响（车窗打开后，车辆速度较快的情况下，气压计读数会在短时间内降低，对应测量的高度短时间内增加）。

下表 2 显示基于 ERNet 的偏航检测方法在准确率和召回率上都远超基于气压计的偏航检测方法。需要注意的是，传统的偏航检测方法无法检测出高架场景的偏航事件。

Measures	ERNet-based	Bar-based	Traditional
$yaw_p(\%)$	91.1	69.4	0
$yaw_r(\%)$	58.4	21.5	0

表 2：在评估测试集上的偏航检测结果

未来展望

在本文中，研究者提出了 ERNet，一个轻量级而且真正工业级的模型，使用 triplet loss 进行训练，从根本上解决高架道路识别问题。

在未来，研究者会将 ERNet 扩展到全国。并探索如何使用道路拓扑信息降低误识别。此外，现在的偏航检测方法至少需要 15 秒才能识别出车辆偏航。这对于用户的体验并不是最佳的。因此，研究者考虑使用循环神经网络对偏航检测方法进行建模，期望能够在更短时间检测出偏航。

引文

- [1] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In KDD. 785 – 794.
- [2] Yaoming Gong, Yanmin Zhu, and Jiadi Yu. 2015. DEEL: Detecting Elevation of Urban Roads with Smartphones on Wheels. In IEEE International Conference on Sensing, Communication, and Networking.
- [3] GPS Accuracy 2017. GPS.
<https://www.gps.gov/systems/gps/performance/>

[4] GPS Overview 2017. GPS.

<https://www.gps.gov/systems/gnss/>

[5] Xinwei He, Yang Zhou, Zhichao Zhou, Song Bai, and Xiang Bai. 2018. Triplet-Center Loss for Multi-View 3D Object Retrieval. In CVPR. 1945 – 1954.

[6] Alexander Hermans, Lucas Beyer, and Bastian Leibe. 2017. In Defense of the Triplet Loss for Person Re-Identification. In arXiv.

[7] Ping-Fan Ho, Chia-Che Hsu, Jyh-Cheng Chen, and Tao Zhang. 2018. Poster: Using Barometer on Smartphones to Improve GPS Navigation Altitude Accuracy. In MobiCom. 741 – 743.

[8] Diederik P Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In ICLR.

[9] Yin Lou, Chengyang Zhang, Xing Xie, Yu Zheng, Wei Wang, and Yan Huang. 2009. Map-Matching for Low-Sampling-Rate GPS Trajectories. In SIGSPATIAL.

[10] Paul E Newson and John Krumm. 2009. Hidden Markov map matching through noise and sparseness. In International Conference on Advances in Geographic Information Systems. 336 – 343.

[11] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In CVPR. 815 – 823.

[12] Xiaocheng Tang, Zhiwei Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, and Jieping Ye. 2019. A Deep Value–network Based Approach for Multi–Driver Order Dispatching. In KDD. 1780 – 1790.

[13] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. 2016. A Discriminative Feature Learning Approach for Deep Face Recognition. In ECCV. 499 – 515.

[14] Can Yang and Gyozo Gidofalvi. 2018. Fast map matching, an algorithm integrating hidden Markov model with precomputation. International Journal of Geographical Information Science 32, 3 (2018), 547 – 570.[15] Yu Zheng. 2015. Trajectory Data Mining: An

Overview. International Journal of Geographical Information Science
6, 3 (2015).

招聘

阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位地点：北京。欢迎投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘团队-应聘方向。

AAAI 论文解读 | LRC-BERT：对比学习潜在语义知识蒸馏

作者：高德智能技术中心

导读

高德智能技术中心研发团队在工作中设计了对比学习框架进行知识蒸馏，并在此基础上提出 COS-NCE LOSS，该论文已被 AI 顶会 AAAI2021 接收。

论文链接：<https://arxiv.org/abs/2012.07335>

NLP 自然语言处理在高德各个业务线发挥重要作用，例如动态事件命名实时识别，搜索场景用户语义理解，共享出行通话文本自动判责等。

而 NLP 领域近期最重要的进展当属预训练模型，Google 发布的 BERT 预训练语言模型一经推出就霸占了 NLP 各大榜单，提升了诸多 NLP 任务的性能，在 11 种不同 NLP 测试中创出最佳成绩，预训练模型成为自然语言理解主要趋势之一。

预训练模型通常包括两个阶段：第一阶段是在大型语料库根据给

定上下文预测特定文本。第二阶段是在特定的下游任务进行 finetuning。

BERT 的强大毫无疑问，但由于模型有上亿参数量体型庞大，单个样本计算一次的开销动辄上百毫秒，因而给部署线上服务带来很大的困扰，如何让 BERT 瘦身是工业界以及学术界重点攻坚问题。Hinton 的文章 "Distilling the Knowledge in a Neural Network" 首次提出了知识蒸馏的概念，将 teacher 知识压缩到 student 网络，student 网络与 teacher 网络具有相同的预测能力但拥有更快的推理速度，极大节省了计算资源。

目前前沿的技术有微软的 BERT-PKD (Patient Knowledge Distillation for BERT)，huggingface 的 DistilBERT，以及华为 TinyBERT。其基本思路都是减少 transformer encoding 的层数和 hidden size 大小，实现细节上各有不同，主要差异体现在 loss 的设计上。

然而知识蒸馏最核心问题是如何捕捉到模型潜在语义信息，而之前工作焦点在 loss 设计上，而这种方式让模型关注在单个样本的表达信息细节上，对于捕捉潜在语义信息无能为力。

高德智能技术中心研发团队在工作中设计了对比学习框架进行知识蒸馏，并在此基础上提出 COS-NCE LOSS，通过优化 COS-NCE LOSS 拉近正样本，并拉远负样本距离，能够让模型有效的学习到潜在语义表达信息（LRC-BERT 对比 DistillBERT，BERT-PKD 并不限制模型的结构，student 网络可以灵活的选择模型结构以及特征维度）。

同时，为进一步让 LRC-BERT 更加有效的学习，我们设计了两阶段训练过程。

最后，LRC-BERT 在 word vector embedding layer 引入梯度扰动技术提升模型鲁棒性。本文的主要贡献点概括如下：

- 提出了对比学习框架进行知识蒸馏，在此基础上提出 COS-NCE LOSS 可以有效的捕捉潜在语义信息。
- 梯度扰动技术首次引入到知识蒸馏中，在实验中验证其能够提升模型的鲁棒性。
- 提出使用两阶段模型训练方法更加高效的提取中间层潜在语义信息。

- 本文在 General Language Understanding Evaluation (GLUE) 评测集合取得了蒸馏模型的 SOTA 效果。



背景介绍

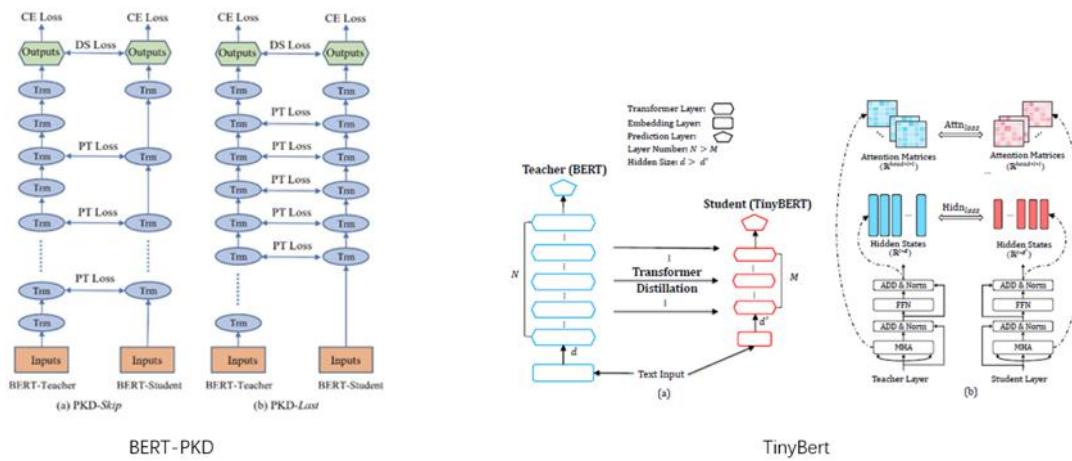
当前深度学习模型压缩方法的研究主要可以分为以下几个方向：裁剪、因子分解、权重共享、量化、知识蒸馏。裁剪—移除网络中不必要的部分。剪裁方法包括 weight 裁剪、attention head 裁剪、layer 裁剪，一些方法还在训练过程中通过正则化，以增加可靠性(layer dropout)。因子分解—通过将参数矩阵分解成两个较小矩阵的乘积来逼近原始参数矩阵。

这给矩阵施加了低秩约束，权重因子分解既可以应用于输入嵌入

层（这节省了大量磁盘内存），也可以应用于前馈/自注意力层的参数（为了提高速度）。

知识蒸馏是一种模型压缩常见方法，用于模型压缩指的是在 teacher-student 框架中，将复杂、学习能力强的网络学到的特征表示“知识蒸馏”出来，传递给参数量小、学习能力弱的网络。

例如 BERT-PKD 首次提出 student 学习 teacher 中间层表达，分别尝试两种方式 skip, last 模式如下图。TinyBert 提出了基于 MSE 的 transformer 层 Attention loss、hidden loss 以及 emb loss 来进行知识蒸馏。



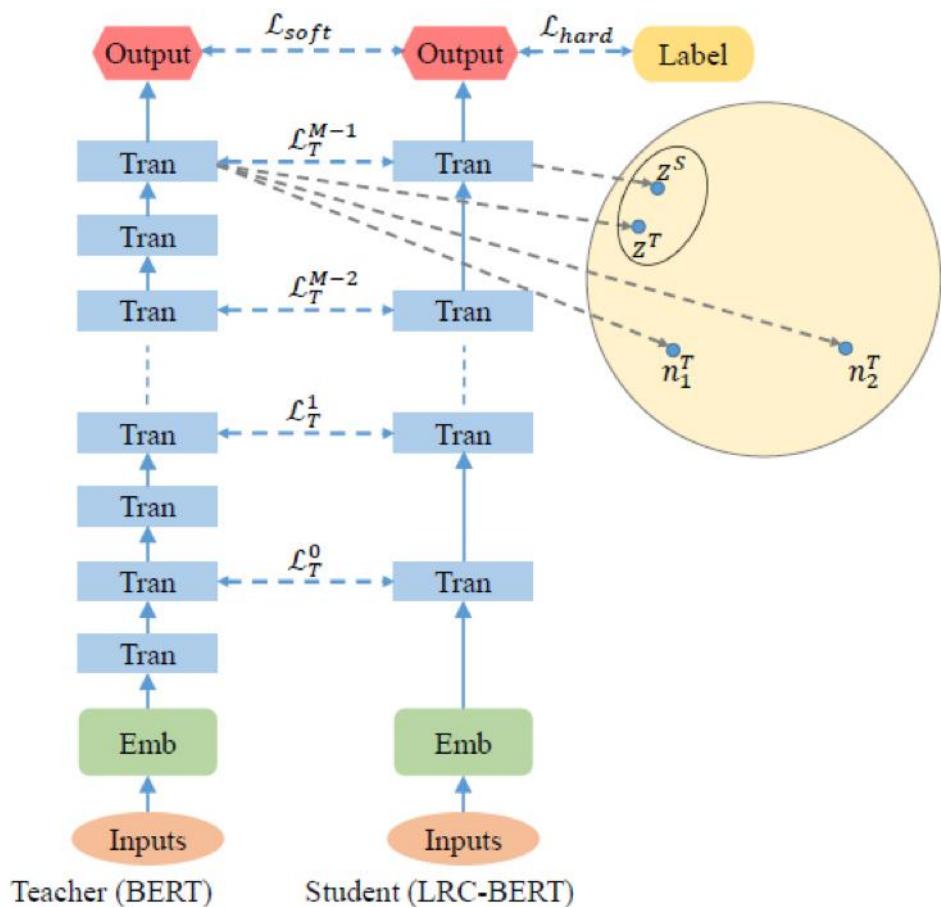
方法

3.1 问题定义

teacher 网络定义为 $f^T(x, \theta)$: x 为模型输入， θ 为模型参数，模型输出为 Z^T 。student 网络定义为 $f^S(x, \theta')$ 同时输出为 Z^S 。目标是 student 的 $f^S(x, \theta')$ 表达更加贴近 $f^T(x, \theta)$ 表达，同时最小化 prediction layer loss，使 student 与 teacher 具有同样的性能。

3.2 模型蒸馏结构

如下图即为 LRC-BERT 结构，对比学习作用在中间层表达使 student 能够学习到 teacher 潜在语义信息。举例来说，对于 student 表达 Z^S 与 teacher 特征表达 Z^T 靠近，而要远离负例 n_1^T 和 n_2^T 。



3.3 COS-based NCE loss

对比学习的概念很早就有，但真正成为热门方向是在 2020 年的 2 月份，Hinton 组的 Ting Chen 提出了 SimCLR[9]，用该框架训练出的表示以 7% 的提升刷爆了之前的 SOTA，甚至接近有监督模型的效果。

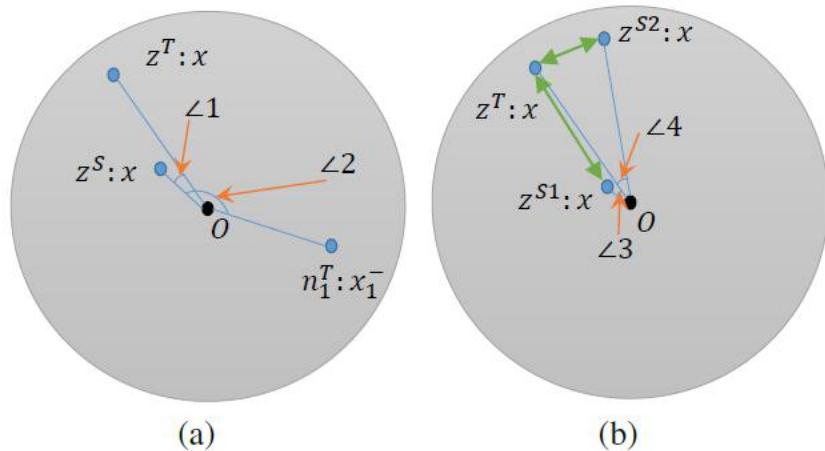
对比学习目标是为输入 X 学习一个表示 Z （最好的情况就是知道 Z 就能知道 X ），衡量方式采用互信息 $I(X, Z)$ ，最大化互信息的目标进行推导就会得到对比学习的 loss（也称 InfoNCE），其核心是通过计算样本表示间的距离，拉近正样本，拉远负样本获取 X 深层信息表达。

本文设计了对比损失 COS-NCE 用于中间层知识蒸馏。对于一个给定的 teacher 网络 $f^T(x, \theta)$ 以及 student 网络 $f^S(x, \theta')$ ，任何一个正例随机选择 K negative samples $N = \{n_1^-, n_2^-, \dots, n_K^-\}$ ，因此得到 teacher 中间层表达 Z^T ，student 中间层表达 Z^S ，以及 K 个负例表达 $N = \{n_1^-, n_2^-, \dots, n_K^-\}$ 。

不同于之前对比学习采用 Euclidean distance or mutual information 作为 loss，本文提出 cos 角度度量方式进行对比学习。如下图所示：

(a) 在特征空间 Z^S 与 Z^T 角度更加贴近，而与负例 $n^T \cos$ 角度差异变大。

(b) 对于不同的 student f^{s1} f^{s2} (student f^{s1} 语义与 teacher 语义更相似)：在 Euclidean distance (绿色) 上 Z^{s2} 更加贴近 Z^T ，然而在基于 cos-based 距离衡量上 Z^{s1} 相对 Z^{s2} 更贴合 Z^T ，可见 cos-based 更加符合语义特征表达度量。



COS-based NCE loss 公式如下， $g(\dots) \rightarrow [0, 2]$ 用来衡量两个向量角度 distance， $g(x, y)$ 越小代表两个向量越相似， $g(x, y)=2$ 代表两个向量不相似状态的边界。

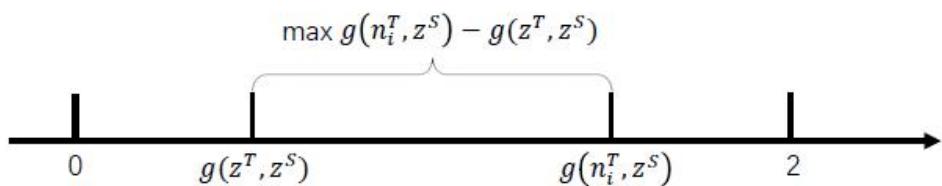
COS-NCE 设计动机是最小化 Z^S 和 Z^T 的角度 distance，最大化 Z^S 和 N^T 角度 distance。

如下图所示， $g(n_i^T, z^S)$ 与 $g(z^T, z^S)$ 的距离需要被放大，本文的处理是将最大化问题转换成最小化问题，其具体定义为：

$$2 - (g(n_i^T, z^S) - g(z^T, z^S))。$$

$$\mathcal{L}_C(z^S, z^T, N) = \frac{\sum_{i=1}^K (2 - (g(n_i^T, z^S) - g(z^T, z^S))) + g(z^T, z^S)}{2K} \quad (1)$$

$$g(x, y) = 1 - \frac{x \cdot y}{\|x\| \|y\|}, \quad (2)$$



3.4 Distillation for transformer-layer

COS-NCE 用于 transformer-layer 的蒸馏，每个 transformer-layer 包含 multi head attention 以及 FFN，本文针对 FFN 的输出进行蒸馏。这里假定 teacher 有 N -transformer layer，student 有 M -transformer layer。

这里我们选择使用 uniform 的方式完成 teacher 的 N -transformerlayer 与 student 的 M -transformerlayer 之间的映射。

公式如下， $h_i^S \in \mathbb{R}^{l \times d}$ 表示 student 网络 i -th transformer layer 的输出。 $h\phi^T \in \mathbb{R}^{l \times d'}$ 表示 teacher 网络 ϕ i -th transformer 的输出。

$j = \phi_i$ 即为层数映射函数 student 学习 teacher 对应层数输出， i 表示文本长度， $d' < d$ 表示 teacher student hidden size (d 的维度少于 d')。

$H_i^T = \{ h_0, i^T, h_1, i^T, \dots, h_{k-1}, i^T \}$ 对应 teacher 网络 i -th transformer Knegative 样本。 $W \in \mathbb{R}^{d \times d'}$ 为维度映射参数，目的将 student 与 teacher hidden size 对齐。

$$\mathcal{L}_T^i = \mathcal{L}_C(h_i^S W, h_{\phi_i}^T, H_{\phi_i}^T), \quad (3)$$

3.5 Distillation for predict-layer

更好的适配下游预测任务，本文采用 student 的预测层输出学习 teacher 的预测层输出，即 softloss。同时 student 学习 real label，即 hardloss。其中 KL divergence 用于 student 学习 teacher 预测分布，cross-entropy loss 用于 student 学习 real label。

y_S, y_T 分别是 student、teacher 的预测输出， t controls the smoothness of the output distribution， y 为真实 label。公式 6 是最终损失函数， α, β, γ 分别是不同损失的加权系数。

为了让模型更加高效的学习中间层表达，本文采用两阶段训练方法，在第一阶段，我们先关注中间层的对比损失 α, β, γ 设置为 1, 0, 0。

在第二阶段， β 、 γ 权重设置大于 0 保证模型有能力预测下游任务。

$$\mathcal{L}_{soft} = softmax(y^T/\tau) \cdot \log\left(\frac{softmax(y^T/\tau)}{softmax(y^S/\tau)}\right), \quad (4)$$

$$\mathcal{L}_{hard} = -softmax(y/\tau)\log(softmax(y^S)), \quad (5)$$

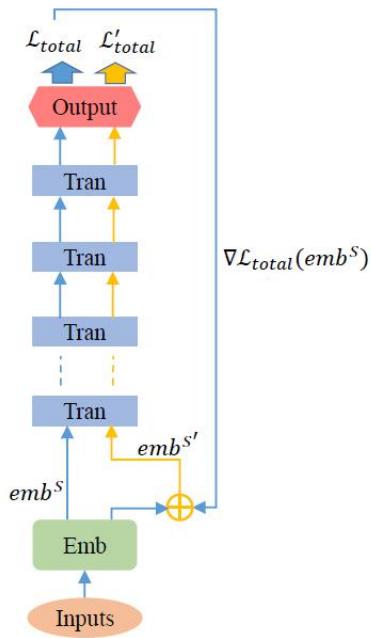
$$\mathcal{L}_{total} = \alpha \sum_{i=0}^{M-1} \mathcal{L}_T^i + \beta \mathcal{L}_{soft} + \gamma \mathcal{L}_{hard}, \quad (6)$$

3.6 Training based on Gradient Perturbation

模型结构是影响鲁棒性重要因素，因此如何让模型更加鲁棒成为模型压缩算法中一个重要的关注点。之前在模型压缩剪枝算法中引入过正则化以增加可靠性，而本文引入了梯度扰动技术增强 LRC-BERT 的鲁棒性。

下图展示梯度扰动过程，本文没有直接使用 L_{total} 对 model 进行反向梯度传播，而是优先计算 emb 的梯度 $\nabla L_{total}(emb^S)$ 将其作用于 emb^S 输入进行扰动。

最终，使用梯度扰动之后 loss 对 model 进行参数更新。如下公式 emb^S 是增加梯度扰动的表示。



$$emb^{S'} = emb^S + \nabla L_{total}(emb^S). \quad (7)$$

4. 实验

4.1 数据集

GLUE Benchmark <https://gluebenchmark.com/>（通用语言理解评估基准）是衡量自然语言理解技术水平的重要指标。数据集包含了自然语言推断、语义相似度、问答匹配、情感分析等 9 项任务。本文将 LRC-BERT 在 GLUE 数据集合上进行评测。

4.2 实验参数设置

本文采用 BERT-Base 作为 teacher，其包含 12 层 transformer，每层包含 12 attention head，768 hidden size，3072 intermediate size。

student 网络采用 4 层 transformer，每层包含 12 attention head，312 hidden size，1200 intermediate size。为了更好验证 LRC-BERT 有效性，本文设置了两组模型：LRC-BERT 包含预训练（使用 Wikipedia corpus）、specific tasks 蒸馏；LRC-BERT1 直接进行 specific tasks 蒸馏。

蒸馏实验中，学习率选择 { 5e-5, 1e-4, 3e-4 }，batch size 16。MRPC RTE CoLA 训练数据少于 10K 的数据集采用 90 epoch，其他数据集合采用 18 epochs。两阶段实验设置，80% steps 使用第一阶段 { $\alpha : \beta : \gamma = 1 : 0 : 0$ }，剩下 20% step 使用第二阶段训练 { $\alpha : \beta : \gamma = 1 : 1 : 3$ }，t 设置 1.1。

4.3 主要实验结果

主要实验结果如下：

(1) LRC-BERT 明显优于 DistilBERT、BERT-PKD、TinyBERT。平均预测效果 LRC-BERT 保留 BERT-base 97.4% 的性能。说明 LRC-BERT 的有效性。

(2) 训练数据量较大的数据集合上 ($>100K$)，LRC-BERT1 直接在下游任务上蒸馏。对比 TinyBERT 分别在 MNLI-m, MNLI-mm, QQP, QNLI 上分别提高 0.3%，0.8%，0.6%，0.6%。

(3) LRC-BERT 对比 LRC-BERT1s 在 MRPC, RTE, CoLA 分别提高 4%, 12.1%, 14.9%。

同时另一个重要参考指标是模型推理速度，下图所示 LRC-BERT 取得了 $9.6 \times$ 速度提升，modelsize 上取得了 $7.5 \times$ 收益。

Model	MNLI-m (393k)	MNLI-mm (393k)	QQP (364k)	SST-2 (67k)	QNLI (105k)	MRPC (3.7k)	RTE (2.5k)	CoLA (8.5k)	STS-B (5.7k)	Average
BERT-base (teacher)	84.3	83.8	71.4	93.6	90.9	88.0	66.4	53.0	84.8	79.6
DistilBERT	78.9	78.0	68.5	91.4	85.2	82.4	54.1	32.8	76.1	71.9
BERT-PKD	79.9	79.3	70.2	89.4	85.1	82.6	62.3	24.8	79.8	72.6
TinyBERT	82.5	81.8	71.3	92.6	87.7	86.4	62.9	43.3	79.9	76.5
LRC-BERT ₁	82.8	82.6	71.9	90.7	88.3	83.0	51.0	31.6	79.8	73.5
LRC-BERT	83.1	82.7	72.2	92.9	88.7	87.0	63.1	46.5	81.2	77.5

Model	transformer layers	hidden size	Param	inference time(s)
BERT-base	12	768	109M	121.4
LRC-BERT	4	312	14.5M	12.7

4.4 消融实验

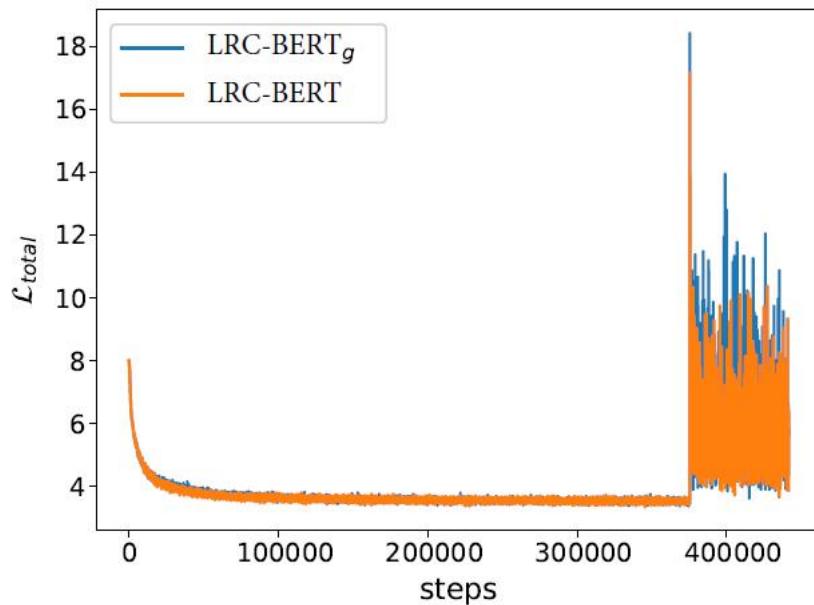
消融实验在 MNLI-m, MNLI-mm, MPRC, CoLA 数据集合上从 loss function 以及 梯度扰动上进行分析。

Effect of different loss function, 本文分别移除 COS-NCE, softloss, hardloss 验证模型效果，分别用 LRC-BERTC LRC-BERTS LRC-BERTH 表示。如下图所示，COS-NCE 影响权重最大去除后在如下实验中效果最差，特别在 CoLA 数据集合上去除 COS-NCE 效果从 50 到 37。而 softloss,

hardloss 对最终效果影响有限。综上三种损失对 LRC-BERT 均有效。

Model	MNLI-m	MNLI-mm	MRPC	CoLA
LRC-BERT	83.4	83.5	89.0	50.0
$LRC-BERT_C$	78.0	78.2	81.5	37.0
$LRC-BERT_S$	82.7	83.0	89.4	48.8
$LRC-BERT_H$	83.0	83.5	88.7	48.6

Effect of gradient perturbation，梯度扰动能够在训练过程中影响中间层数据分布，同样验证效果采用 $LRC-BERT_g$ 作为去除 gradient perturbation 模型。下图展示在 MNLI-m 训练过程中 training loss 变化，在第二阶段 $LRC-BERT$ 损失振幅相对比 $LRC-BERT_g$ 减弱， $LRC-BERT$ 趋于稳定状态。



Analysis of Two-stage Training Method, 采用两阶段的目的是为了让 student 在训练开始阶段更加专注于学习 teacher 中间层表达，同样设置 LRC-BERT2 去除二阶段训练直接采用 { $\alpha : \beta : \gamma = 1 : 1 : 3$ } 在 MNLI-m 进行训练。效果如下图所示，去除两阶段效果下降明显，同样也能够说明 COS-NCE 在中间层蒸馏的作用。

Model	Accuracy
LRC-BERT	83.4
LRC-BERT ₂	79.4

Analysis of Two-stage Training Method, COS-NCE 采用 cos 角度 distance 对中间层 transformer 蒸馏，同时本文采用 BERT_M (中间层使用 MSE 替换 COS-NCE 作为 loss) 为对比。随机抽取 case 分析，前两个 case LRC-BERT 以及 BERT_M 预测正确，而后面两个 BERT_M 预测比较波动导致预测错误，LRC-BERT angular distance 在预期范围内，由此可见 LRC-BERT 能够有效捕捉到深层语义信息。

sentence1	sentence2	MSE	angular distance(g)	label	prediction (BERT _M)
Paper goods.	Paper products.	1.532	0.505	entailment	entailment
oh constantly	Rarely	1.557	0.525	contradiction	contradiction
The truth?	Will you tell the truth?	1.730	0.496	neutral	contradiction
I'm not exactly sure.	I'm not exactly sure if you're aware of your issues.	2.108	0.484	entailment	neutral

在高德具体业务场景的落地

动态事件是指由于道路通行能力变化影响用户出行事件，包括封闭、施工、事故等。作为高德交通动态事件重要获取途径 NLP 事件抽取业务，主要流程为收集交通官方平台以及各个媒体平台信息，经过命名实体识别、事件拆分组合最终输出动态事件，影响 Amap 用户出行规划路线，如下图示例。

本文提出的方法已在高德交通动态事件抽取中落地，LRC-BERT 保留了 BERT-base 97% 的性能，人工评测平日准确率提高 4%，召回率提高 3%；节假日准确率提高 5%，召回率提高 7%。

此外，本方法训练方法简单，复现成本低，可广泛应用于使用自然语言理解（NLU）各个业务线，提高模型推理速度降低部署成本。



总结

本文创新提出了对比学习框架进行知识蒸馏，并在基础上提出 COS-NCE LOSS 可以有效的捕捉潜在语义信息。梯度扰动技术首次引入到知识蒸馏中，在实验中验证其能够提升模型的鲁棒性。

为了更加高效提取中间层潜在语义信息采用使用两阶段模型训练方法。GLUE Benchmark 实验结果表明 LRC-BERT 模型有效性。

招聘

阿里巴巴高德地图智能技术中心长期招聘机器学习算法、C++、Java 资深工程师/技术专家/高级专家，职位地点：北京。欢迎

投递简历到 gdtech@alibaba-inc.com，邮件主题为：姓名-应聘
团队-应聘方向。