摘要:

2017年7月，我参加了某互联网公司的网约车出行平台的开发，系统架构师设计师一职，该平台主要是将司机乘客资源匹配解决用户出行问题，提供了专车、快车、出租车等多种出行方式。本文结合作者实践，以出行平台为例，讨论NoSQL数据库技术以及其应用，NoSQL数据库泛指非关系型数据库。按照数据模型可以分为4类：键值（Key-Value）存储数据库、列存储数据库、文档存型数据库、图（Graph）数据库。主要是为了弥补，关系型数据库：性能低、容量有限、和运维成不低的缺点。在平台中我们选择键值数据库，来做内容缓存。具体在用户登录注册、司机位置上报、订单执行过程等场景增加键值数据库同步，降低业务数据库的读写压力，提升平台性能。使平台达上线预期性能目标，稳定运行的一年多时间。无较大生产故障得到用户的一致好评。是对我工作的认可和肯定。

正文:

2017年7月，我参加了某互联网公司的网约车出行平台的开发，该平台主要是为了改善目前：黑车乱象、乘客出行安全无法得到保障背景下。为了构建一个安全、规范的网约车环境。解决用户打车难、提升用户的出行幸福感，拉动了产业链的发展，构建了一个完整的出行生态链的目标。而构建的一个网约车出行平台。平台包括乘客端、司机端、后台管理系统三部分组成。乘客端供乘客查询车辆、发布订单、支付车费、评论司机；司机端供司机车辆信息认证、出车接单、乘客接送、车费提现等；管理系统主要是提供系统报表查询、规则配置、乘客管理、司机管理、分公司管理、账务管理等。本平台提供了专车、快车、出租车业务，乘客可以通过自身的需求来选择不同类型业务出行，司机需要上传自身拥有的运营车辆等证照信息到平台审核，只有当平台审核通过后才能正常的再平台上合法运营。在该项目中本人担任系统架构师设计师一职，负责项目的架构设计以及软件开发的部分工作。

在NoSQL的技术选型的过程中我们分析了常见的4种数据库，1、键值（Key-Value）存储数据库，使用场景在内容缓存，主要用于处理大量数据库的高访问负载也用于一些日志系统等等。数据模型包括：Key指向Value的键值对，通常通过hash table来实现。优点查询速度快。缺点数据无结构化，通常只被当做字符串或二进制数据，如：Redis、Oracle BDB。2、列存储数据库，使用场景在分布式文件系统，以列簇形式存储，将同一列数据存储在一起。查找速度快、可拓展性强，更容易分布式拓展。如：HBase, Cassandra。3、文档型数据库。使用场景在Web应用中。它是一种键值存储相类似，该类型的数据模型是版本化的文档，半结构化的文档以特定的格式存储。比如JSON。如：MongoDb, CouchDB。4、图形数据库（Graph）使用场景在社交网络、推荐系统等专注构件关系的图谱。采用图结构，利用图解狗相关算法，比如最短路径寻址，N度关系查找等。如：Neo4J, InfoGrid。在本平台中要是通过内容缓存来提升平台性能，所以我们选择了键值数据库Redis。

在平台中我主要是在用户中心、位置中心、订单中心使用键值数据库。

在用户中心，我们把用户登录状态、用户基本信息、用户核心业务等信息。存储到Redis中方便网关校验用户登录状态、用户访问权限。在位置中心，司机端每间隔3s上报一次司机位置信息。把司机的当前位置存储到Redis中、并且将经纬度信息转化为地理位置栅格方便司机实时位置和周边车辆信息的查询。在订单中心，需要将乘客的订单分派给对应的司机在派单的过程中将订单存储到业务数据库同时写入Redis。用于多轮派单查询、超时关闭订单和乘客取消订单的情况。这些设计可以事先在订单创建后，乘客和实际的快速匹配。同时在派单过程中需要频繁的查询司机、司机车辆信息、司机订单状态等信息。对比关系型数据库Redisc采用hash table的方式数据存储在内存中查询效率大大高于关系型数据库SQL查询，能够有效的提升平台的性能。

个人中心，主要是负责登录注销用户信息维护等。我们采用业务数据库+ Redis 内容缓存的方式。在用户登录我们主要采用手机号+验证码方式登录，用户输入手机号，请求验证码我们在返回验证码之前先将"crc-"加手机号作为Key，验证码作为Value的内容存储到Redis数据库中，设置有效期。然后用户输入信息后验证码后，然后通过手机号加验证码前缀"crc-"来获取验证码对比，如果能够匹配上返回用户登录成功，并且通过"token-"+手机号，存储用户信息的JSON字符串，包含：手机号、用户状态、执行订单ID等。返回用户令牌token，反之登录失败。在网关判断用户是否在请求头中带有效的token, 鉴定用户的登录状态。拦截非法请求。在用户的订单流程过程中，会将用户的订单信息放入token中，且用户创建订单的时候可以判断用户当前状态是否能够创建订单。

位置中心, 主要是负责司机的实时位置存储，在位置中心由于实时位置数据具有实时性且查询频繁，我们选择将数据存在Redis中可以提供高效的查询。司机登录后，每间隔3s将司机的实时位置通过报到位置中心，首先将位置信息存储到"pos-"加手机号存储对应的司机位置信息、"ts-"加手机号存储当前上报的时间戳信息分别存入Redis设置过期时间为3分钟。且将位置信息栅格化后将司机手机号存储到"rid-xx-xx"的栅格key的list中，便于快速查询周边车辆。如乘客登录的时候通过当前经纬度信息来查周边5km内的车辆，查询周围25个栅格来查询订单信息。用户下线和用户位置信息过期会将对应的栅格中的司机手机号删除，在业务查询过程中回去个人中心获获取司机的登录状态和司机的类型来做查询条件。

订单中心，主要是负责将乘客创建的订单按照匹配程度推送给一个或者多个司机。我们采用业务数据库存储和Redis数据同步的方式。在订单服务启动后会将业务的系统中的字典信息加载存储到Redis中，字典信息包含分公司派单距离、订单过期时间、司机排序规则等参数。在专车的派单过程中首先会在订单创建成功在位置中心去请求符合条件的司机，然后计算司机的实时位置到乘客的距离是否有N公里，这里参考字典中的设置。拿到周边的司机后，然后判断司机的服务分，订单量，好评率等维度来对司机作排序，然后将订单推送给最符合条件的司机，并将司机和乘客的令牌中更新该订单。完成一次派单。如果当前一次派单没有成功，会按照字典的规则做扩大范围，直到订到超期关闭订单。然后然后进入订单执行过程。

以上的架构设计使得平台达到了预期性能目标，发挥了NoSQL数据库的避免不必要的复杂性、吞吐量、高水平拓展能力和低端硬件集群，避免高贵的对象-关系映射等优势。首先这些设计，对于平台的整体提升的性能非常大。使得平台更加的灵活，更容易拓展，更加高可用，提升查询效率，提升了平台承载体量，提升了公司核心产品竞争力。其次为公司节约了硬件资源和运维成本，为公司节约了运营成本。在过程中我们做了几次相关的技术分享，总结了在开发过程中遇到的问题和分享了一些优秀的解决方案。从而提升了开发人员的技术水平，同时也提升了团队的凝聚力。每个开发成员应对分布式系统的设计有了新的认识。

当然，NoSQL自身也有数据模型和查询语言没有经过数学验证，不支持ACID特征，功能简单，没有统一的查询模型等缺点。在项目中我们开发了内容缓存构建保证内容缓存和业务数据库的数据一致性。所以我们应该继续优化NoSQL的使用设计，做到更进一步，争取能够设计出更多高质量，高性能的软件产品。