人人车业务平台从最初典型的LNMP单机单服务部署架构，发展到如今分布式服务化架构，五百多台虚拟机部署，一路走来，踩过不少坑，也遇到过不少挑战，特别是对于基于云服务进行业务开发的场景，以及从零开始服务化与SOA之路更是颇有心得，希望通过此次分享全面回顾人人车业务平台技术架构发展之路，重点分享如下内容：

创业初期技术架构选型思考，那些年我们趟过的坑，云服务与三方服务使用心得；

O2O型互联网创业公司，重线下团队技术型公司，技术架构优化之路，分享我们人人车是如何做服务拆分、如何做服务化、如何做SOA、如何做DB慢SQL优化等；

人人车HTTP服务管理框架介绍相关使用经验分享；

人人车业务平台技术架构规划；

人人车是由一帮百度兄弟们，于 2014年4月成立，作为二手车C2C模式首创者，以及典型的O2O二手车互联网电商企业， 创业两年多，目前业务范围已经覆盖全国数十个城市了，今年年底将覆盖到过百城市。

我们业务平台负责全部核心业务系统研发工作，包括：CRM系统、发布系统、编辑系统、评估系统、销售系统，金融支付等业务系统，承担人人车所有线下业务团队收车、卖车等核心环节技术支持！

好了，接下来，就进入到今天的第一个话题，分享下我们创业初期的技术架构和技术选型思考。

创业初期技术架构选型思考

为了快速开发迭代，初期人人车业务平台所有服务全部基于LNMP进行搭建，从前到后一套系统，2台云主机机器，非常简单清晰，V 0.1版本架构如下图所示：

从上图可以看出创业初期我们的V0.1版本的技术架构非常简单，主要基于一下几点考虑：

研发资源的限制，我们必须集中精力做最重要的事情，业务需求满足是第一位的；

产品快速迭代的需要，创业初期整个技术团队技术能力成熟度不高，因此我们更是倾向简单、清晰、研发熟练度高的技术框架；

初期业务规模较小，技术架构需求较低，简单框架完全可以满足业务需求，适合业务需求的架构就是好架构；

轻量级LNMP框架，选择足够简单灵活的PHP CI框架， 云MySQL RDS服务，简单易上手，运维部署成本非常低，大大节省了我们运维成本，以及开发人员的学习成本；

为了能够快速搭建服务把创业项目搞上线，我们之前一直坚持的几个原则也跟大家分享下：

大量使用云服务，包括：虚拟机、DB、MQ中间件、Redis等基础服务，大大降低了这些技术能力要求高的基础主件、存储服务、消息中间件等服务的部署运维代价；其实就是：花钱买服务，花钱买时间，花钱买研发能力；

大量使用第三方收费服务，以代价换时间，例如：Teambition服务、BDP报表服务、ODPS数据管理、图片存储服务、语音通话服务等等；

权衡自研系统与三方采购系统代价，对于非核心业务系统，如果能通过采购满足需求，就快速采购服务，应用到一线业务中去，例如：HR系统、OA系统、培训系统、云报销系统、工作流引擎等等；

拥抱开源，GitHub是个金矿，值得每一位创业者去挖掘，去发现自己的需求，其实很多同行已经做了我们要做的基础研发事情；

O2O创业项目，线下业务系统是第一位的，所有资源”all in” 去提升线下团队工作效率，优化线下业务系统流程，提升线索转化漏斗，提高转化率；

低频冷门电商业务，口碑推广，营销推广，广告投放，SEM等都是需要精细运营的，是流量运营最重要的几个关键点；

虽然我们坚持了上面提到的很多原则，不经意间我们还是踩了很多坑，例如：

云服务不靠谱，出问题造成雪崩，导致整个站点不可用。这块我的经验是不能把整个站点的生死寄托在别人手中，一定要多给自己留个B计划，容灾方案不管是公司发展到哪一步都需要的；云服务作为基础服务在做架构部署时，一定要多考虑几家互为容灾备份；

第三方服务在使用时，一定要与自己业务服务解耦，能简单封装一个proxy代理层那就最好了，避免直接嵌套第三方SDK到业务代码中，这样就相当于跟第三方捆绑在一条船上了，友谊的小船说翻就翻啊；

核心系统容灾，核心系统在选择第三方服务时，一定要多挑选几个合作伙伴，系统架构上，最好能支持容灾切换，在一个第三方服务出问题时，能够快速进行切换，避免将核心系统生死权掌握到第三方服务上去。例如我们人人车是一个重电销型公司，呼叫中心就是我们的核心，这套系统之前我们设计初期就仅仅只支持一家供应商，导致后面只要合作伙伴出问题，我们就跟着挂了，非常非常的被动；真是不怕神一样的对手，就怕猪一样的队友啊！

重视DB设计，这个问题，可能在初期大家觉得MySQL DB数据库表可以随便设计，表的字段个数，字段类型都影响不大，其实初期严格把关DB设计，后期就会少填很多坑，我们曾经有个大json\_data字段，历史原因设计不友好，导致我们花了2年时间才把这个字段拆掉，真的是非常痛苦的经历；

人人车业务平台技术架构优化之路

随着我们业务不断发展，线下团队出现了爆炸性的增长，销售、评估师、客服人员等都增长不止一个量级，我们的车辆数、订单量、客服量等都是飞速增长，同时由于前期主要关注产品需求，而对技术架构的优化工作相对滞后，导致这个阶段我们业务系统问题不断，这使得我们陷入了非常被动的局面。

这种背景下，我们意识到必须要缓一缓业务需求开发了，不能再一直堆代码了，我们需要进行重构需要梳理我们之前的系统架构，因此这才有了我们全面梳理人人车业务系统架构与系统重构这项工作，希望重点从技术架构合理性上去梳理现有系统！

回头大家可以反观当时这个阶段我们的技术架构，我叫它为V0.2版本架构，如下图所示：

相对于1年前的V0.1版技术架构，此时整个业务端架构，做了部分系统拆分，但是这种拆分仅仅是代码层面的复制，重新创建出了很多业务模块，而真正的底层DB、数据库表、缓存等都还是未进行拆分的；

由于所有业务都使用一个业务DB库，因此DB成了整个业务的单点与雷区，随时可能被引爆，同时由于业务复杂度的不断增大，DB慢查询也不断涌现，DB连接数、CPU、IOPS等核心资源都出现了竞争与争抢，只要一个业务系统出问题，则所有服务都将受影响，出现雪崩效应，正是由于这些问题使得我们业务进展越来越艰难！

在启动服务架构优化项目之前，我们首先从如下两个方面进行梳理，全面诊断我们系统架构存在的问题：

1、流程规范与运维部署问题梳理：

全部服务部署在2台云主机，高度耦合，相互影响，资源竞争；

发布方式原始，svn check、scp覆盖；

回滚流程与工具缺失，遇到问题回滚不方便、回滚不完全；

上线流程不规范，无通报相关机制、无回滚预案；

上线后回归验证方法不全面，不易确定功能是否完整；

2、服务耦合与服务拆分梳理：

所有项目copy自car\_publish这个项目，大量冗余代码，框架使用不纯净；

通用库、工具、类未能复用，未提取部署到统一路径独立部署维护；

API分层逻辑不严格，跨层、跃层调用现象明显，业务调用出口不统一，不便于后续维护；

项目耦合太紧密，未按功能进行服务化拆分，服务调用关系混乱，难以管理；

Cache层缺失，缺少核心逻辑缓存加速，系统有时响应缓慢；

DB数据库耦合严重，未按业务独立拆分，部分字段设计不合理包含大JSON、长字段、慢查询；

日志打印不规范，关键路径日志缺失，日志文件切分与清理机制缺少

接口级别监控缺少，超时、流量变化等业务级别监控缺少；

为了解决上面分析的问题，我们从如下几个方面重点跟进，对人人车业务平台进行有史一来最大的一次优化重构，涉及到：服务拆分、运维部署流程优化、服务监控、DB拆分、慢SQL 优化、同步转异步等事项。

1、服务拆分

从上面分析可以看出，现有系统最大问题，就是各模块之间耦合太高，模块之间依赖太重，因此我们首先启动的项目，就是服务拆分，从服务部署上物理拆分开，避免一个模块出问题造成整个业务系统雪崩。

服务拆分的核心原则如下：

各模块Git代码层面独立管理与维护

线上独立SLB、独立部署、独立运维

DB层面数据隔离，独立从库

通用功能抽取到API层

核心业务逻辑增加cache层

通过以上的拆分工作，我们的服务部署架构变成如下图所示，各个模块独立部署，通过SLB进行负载均衡，从物理部署上将服务拆分开来。

2、DB慢查询优化，我们从如下几个方面着手：

第一，每天通过SQL工具分析出top 5慢查询，要求研发同学必须完成优化，将慢查询优化作为例行事项，每天去review，到后期我们规范化慢SQL排查流程，从如下几个维度分析周级别慢SQL：执行耗时、扫描行数、返回行数等条件，综合分析出高优先级SQL提交给研发同学参考优化；

第二，DB读写分离，所有读请求切换到从库，同时各个服务按优先级使用不同从库，这样主库只剩下更新操作，基本可以杜绝大部分慢SQL，通过慢SQL每天分析报告可以清楚看到需要重点优化的问题SQL；

第三，对于DB轮询操作相关业务逻辑优化，调整轮询为通知机制，降低程序轮询给DB带来的压力，通过更新通知机制保证数据实时通知到业务方，再通过 API查询详细数据，减少各业务模块对主库的直接SQL查询；

第四，加强SQL审核，所有上线SQL，都必须通过SQL审核才可以到线上DB执行，之前由于业务迭代过快，线上DB运维与权限把控不够，导致研发同学直接线上操作DB， 各种慢SQL层出不穷；

第五，加强DB表设计review ，所有对线上数据库DDL相关操作，都必须通过线上DB schema审核才可执行；对于DB设计我们同样归纳出统一的 mysql DB设计规范，所有线上设计都必须满足规范才可执行！

人人车DB Schema规范如下：

所有数据库的DDL操作全部收回，由OP统一管理；

每张表/视图要有注释（comment），格式为 模块|用途|负责人|创建日期 ，例如：贷款|记录贷款用户身份证号码|张三|2016-03-25；

每个字段要有注释（comment），格式为 用途|负责人|创建日期 ， 例如：记录用户性别@1:男@0:女@2:未知|李四|2016-03-25；

每张表三个必加字段，id （自动增长）， create\_time ， update\_time 用作存储主键，记录创建时间，记录更新时间 ， 这三个字段由OP维护，RD无需处理；

有逻辑删除需求的表，可选增加delete\_flag字段（int类型，默认值为0），用来标识字段是否被逻辑删除；

收回RD的delete权限，所有数据不能做物理删除（逻辑上删除的需求可以通过flag标签的方式来处理）；

字段来源需要说明，来自哪张表的那个字段，例如：记录车辆ID，#取自cp\_used\_car.car\_id|李四|2016-03-25；

如果字段为枚举类型，或普通数据类型当做枚举使用，需要列举枚举范围并说明每个枚举值的含义，例如：记录用户性别，#系统根据用户身份证号码判断@1:男@0:女@2:未知|李四|2016-03-25 （ | 用来分割不同项目 ， # 用来标识数据来源 ，@ 用来分割多个枚举 ，: 用来分割枚举名称和值 ， 正常的描述中不要包含 | # @ :）

3、DB拆分：

由于前期业务相对简单，所有业务数据全部集中存放在一个DB里面，这样导致重要数据与普通日志数据都混在一个库中，各业务模块数据也全部落在一个库中，导致业务主库压力过大，随时可能因为一条SQL导致的DB问题，将整个人人车业务平台都给拖垮，因此DB拆分对于我们来说，也迫在眉睫。

我们执行DB拆分的核心原则如下：

各实体分库分表设计

DB主从拆分

数据加密杜绝明文存储

表与DB设计：必须遵循人人车DB设计规范与schema设计规范

索引设计：合理使用索引，杜绝滥用索引

为了能够更好的兼容之前的系统，做到平滑迁移，减少对业务系统的影响，在做DB拆分时，我们首先对业务系统进行实体拆分，再抽取API，对于业务方只需要将之前SQL查询方式迁移到API调用即可。

基于人人车业务实际场景，我们将之前的DB库拆分为如下实体：

Json\_data 数据拆分 （解决DB 中json\_data字段内容过大造成性能问题而拆分）

Car 实体抽取

User 实体抽取

Order 实体抽取

Clue 实体抽取

对于我们最核心的car车辆实体拆分，基本是将之前主库中一个核心表与json\_data全部重新拆开设计的，由原来的一张表扩展到如下图所示的实体设计：

人人车服务化与SOA架构

一、服务化与SOA架构：

在我们做完服务部署架构拆分后，服务化与SOA架构规划其实已经排上了日程，我们做服务化与SOA架构主要出于以下考虑：

1、业务平台视角

多样的需求直达 （业务端各类系统需求的满足）

一致的用户体验 （用户使用与行为习惯的延续，减少培训成本）

快速迭代 （各子系统、业务线只关注自有特性开发、专人专事，避免“重复造轮子”）

资源的最有效利用 （便于人力资源、软硬件资源的高效共用与复用）

系统稳定性与服务质量的提升

问题避免与定位、解决速度的提升

2、公司全局视角

SOA与服务化架构的探索

公司级服务化框架的试点与应用

技术积累与沉淀的需要，研发人员技能提升的必由之路

打造高效O2O线上、线下平台的基石

提升研发效率、降低研发成本的利剑

人人车服务化架构设计的核心指标如下表所示：

对于人人车来说，服务化架构是必行之路，前期的高度耦合的技术架构，已经在很大程度上制约整个公司业务的发展，对于产品需求，响应速度越来越慢，需求堆积越来越多，业务抱怨也越来越大！

因此我们对现有技术团队做了简单的组织架构调整，单独抽取精干力量，搭建专门负责服务化的基础架构团队，开始启动人人车服务化架构之路，我们主要从如下几个方面着手开展工作的：

现有系统功能划分与子系统梳理；

DB层拆分：各子系统DB独立，解耦降低依赖；

所有子系统、应用间的交互都要通过服务的方式来进行，不允许其他方式如：读库、同步脚步、搭建私服、跨层调用等；

所有子系统、应用，都以服务的方式将其数据与功能开放出来，通过API的形式提供访问；

服务的实现方式，初期以现有的HTTP协议为准，后续扩展到RPC、自定义协议等；

通过上面这一系列的优化重构，人人车业务平台技术架构终于站到了 V1.0版本，如下图所示：

从上图我们可以看到，我们的V1.0版本系统架构，基本达到如下效果：

服务已经完全拆分开，模块化与组件化也初步建立；

产出多个基础服务如：基于zookeeper的通用配置管理服务、通用反作弊服务、通用验证码服务等；

产出一批执行规范：通用错误码规范、通用日志管理规范、MySQL规范、PHP编码规范等；

对于一些通用库与通用配置，也完成拆分，单独通过Git项目进行管理，创建comm libs， comm configs等多个通用配置与代码库。

可以说这一阶段，是我们人人车业务研发团队最艰难的阶段，同样也是大家收获最大的阶段，从零开始，我们全程见证了自己的服务，从开始的混乱不堪，逐步变成规范与稳定，每一步拆分，每一个优化，其实都凝聚着所有业务平台研发同学的无限心血！

通过服务化相关项目的推进，我们在项目研发过程中，发现之前的研发流程与研发思路也需要进行调整与转变，主要表现在如下几点：

转变需要解决的主要问题

从简单“堆代码”实现业务需求，到模块化开发“堆积木”满足需求；

从简单代码copy，到模块服务化与基础库沉淀，便于后续各类系统快速开发；

转变的关键点

通用流程业务的抽象；

核心流程的拆分与解耦 （服务、子系统、模块级别拆分）；

服务的可定制 -> SAAS；

服务的可插拔 -> SOA；

二、人人车HTTP微服务管理框架：

在做服务化过程中，我们基于实体拆分，抽取了多个实体API服务，为了能够统一管理这些HTTP API，我们提取了统一的服务接入层，对我们所有http-api进行管理，通过统一的微服务管理框架，实现如下核心功能：

统一的API接入层：所有业务实体拆分后对应API全部接入到服务管理框架，实现对外接口统一，对内统一管理；

统一的权限管理与鉴权：可以实现多种API访问的权限管理，如：jwt、OAuth 2.0、IP白名单机制等等；

统一的API分析与管理：通过WEB上实现服务注册，简单配置下，即可实现服务路由、API分析、流量控制、服务容灾、failover等通用功能；

统一的API监控与统计：统一实行qps统计、日志、监控、openfalcon打通上报数据等功能；

采用Nginx+lua的模式，功能模块采用lua可灵活扩展，基于Nginx实现达到高性能与高可靠，在最上层实现功能需求，减少对业务的侵入与性能的损耗；

人人车微服务管理框架，架构图如下：

人人车业务平台微服务管理框架，基于开源OpenResty框架进行搭建，通过Lua脚本进行功能扩展，实现个性化需求，所有 HTTP请求调用全部通过微服务管理框架，由框架进行路由、鉴权等功能；

由于微服务管理的请求都是无状态的HTTP请求，因此在微服务框架层面可以灵活的扩展，避免单点问题。对于一些核心业务，我们甚至可以灵活的从物理层面独立部署微服务管理框架，从而隔离各个核心模块之间的相互影响。

三、人人车V2.0版本架构：

虽然人人车业务平台V1.0的架构，阶段性的满足了快速发展的业务需求，但是从技术架构角度看，还是存在诸多问题的，例如：

跨层调用大量存在；

API化不够彻底；

DB拆分仍不完整；

HTTP服务缺乏统一管理，缺少统一的服务治理框架；

对于HTTP服务所有模块鉴权、安全、监控等都有大量重复工作。

因此，在2016年上半年我们花了大概小半年的时间，通过进行上面提到的的SOA与服务化，HTTP api统一接入层的加入， 微服务管理框架的加入等一系列工作，终于将人人车业务平台技术架构再次往前推进一步到V2.0版架构，V2.0架构图如下所示：

人人车业务平台技术架构规划

对于下一步人人车业务平台技术架构的走向，总体规划如下图所示：

概括的讲，我们将重点关注服务稳定性、异地容灾、数据存储隔离、基础服务组件化、通用业务模块化、持续集成、运维与安全协同等方面，重点跟进如下几点：

基础支撑项目服务化与组件化：重点跟进呼叫平台化、派单服务化、CRM平台化、配置服务化、反作弊通用服务、验证码服务化等基础业务支撑项目；

基础运维与安全：协同OP一起构建更加方便灵活的运维安全体系，对整个人人车业务平台所有服务进行安全护航；

持续集成支撑：协同QA团队一起打造从Git提交代码开始的自动化流程，包括：自动化测试，自动化发布，自动化回归验证等核心指标；

前端架构分离与服务化：协同FE团队一起打造服务化的前端架构，彻底做到前后端分离，真正实现前后端数据隔离，进一步打造人人车自己的前端开发组件和框架；

数据存储隔离：协同DBA团队一起构建人人车业务系统数据访问层，对于底层DB真正做到业务隔离与上层透明，增强DB数据安全性；

异地容灾支撑系统：基于各家云服务提供商，构建人人车自有资源定位层，实行各云服务商之间灾备，同时通过自建proxy服务方式，实现自建服务与云服务之间的灾备功能；

服务质量评价体系建立：基于现有的业务系统，构建统一的服务质量评价体系，能够实现各服务缺陷管理、稳定性度量、稳定性评价指标统一监控等功能；

作者介绍

徐章健，南开大学计算机软件与理论硕士研究生毕业，2015年加入人人车，担任业务平台总架构师，总体负责业务平台的架构及技术规划，目前重点推进人人车业务平台服务化、SOA、业务服务平台化、数据平台化等基础项目研发管理工作；徐章健同学拥有多年互联网研发管理工作，历任百度ksarch核心研发工程师，360搜索onebox组架构师，美团INF基础架构组架构师。多年来专注于基础架构方面的研究，对于大规模web架构、搜索架构算法、大规模集群基础设施建设，都有一定的见解； 对于高并发、大流量分布式服务、RPC框架、服务治理、服务SOA化等都有丰富的实战经验，对项目管理、团队建设、人才培养等也都颇为擅长！

人人车DB Schema规范如下：

所有数据库的DDL操作全部收回，由OP统一管理；

每张表/视图要有注释（comment），格式为 模块|用途|负责人|创建日期 ，例如：贷款|记录贷款用户身份证号码|张三|2016-03-25；

每个字段要有注释（comment），格式为 用途|负责人|创建日期 ， 例如：记录用户性别@1:男@0:女@2:未知|李四|2016-03-25；

每张表三个必加字段，id （自动增长）， create\_time ， update\_time 用作存储主键，记录创建时间，记录更新时间 ， 这三个字段由OP维护，RD无需处理；

有逻辑删除需求的表，可选增加delete\_flag字段（int类型，默认值为0），用来标识字段是否被逻辑删除；

收回RD的delete权限，所有数据不能做物理删除（逻辑上删除的需求可以通过flag标签的方式来处理）；

字段来源需要说明，来自哪张表的那个字段，例如：记录车辆ID，#取自cp\_used\_car.car\_id|李四|2016-03-25；

如果字段为枚举类型，或普通数据类型当做枚举使用，需要列举枚举范围并说明每个枚举值的含义，例如：记录用户性别，#系统根据用户身份证号码判断@1:男@0:女@2:未知|李四|2016-03-25 （ | 用来分割不同项目 ， # 用来标识数据来源 ，@ 用来分割多个枚举 ，: 用来分割枚举名称和值 ， 正常的描述中不要包含 | # @ :）