殚精竭力

2017年，做得越多越觉得自己不会得越多，有种殚精竭力的感觉。这一年在技术上的思考和实践的比较多，也大胆的尝试做了跨角色跨职能的架构。同时也有点什么都想做的冲动，所以反而有些事情没做好、没做精。

初悟编程

这一年并没有花多少时间在写代码上面，倒是CodeReview的代码不少。之前一直写Java，今年也写了两个月Vue，后面又写了段时间React Native，有跨语言的对比后，对编程有种得心应手的感觉。

规范

`命名即思想`，往往是想明白后命名会很自然，反过来看着不舒服的命名至少说明可能存在很多问题：

- 职责不清楚
- 职责不单一
- 需求理解不正确
- 思想不够简洁直接

规范是其实是用来解决问题的手段，而不是约束。

- 增加可维护性。不需要额外再做翻译。几个人写出来的代码是一样的。虽然说没有永远是对的，但是总有相对正确的写法。
- 增加写代码的成本。不用一直在思考重复的内容。
- 避免一些低级的debug。
- 提升性能

有规范的前提下，可以通过自动化的方式来解决重复的劳动。

- IDE 的代码片段插件
- 标准的Demo的场景
- 代码生成器

JAVA开发规范

推荐

- 《阿里巴巴Java开发手册》
- 《重构》

树状编程

很多小伙伴是线性的编程方式，写一个方法，要把所有的细节放在里面。这种方式的弊端

- 关注点太高。必须知道所有的关注点才能完成这个方法。
- 回来不断的切换思维
- 如果改动一个地方，需要从头索引定位
- 不方便协作

如果把编程方式换成树状。写一个方法，考虑同一个层次的需求点，然后在进一步细化需求点。这里比较难的是怎么判断是同一个层次的抽象。

推荐

- 《代码整洁之道》

防御式编程

很多小伙伴是线性的编程方式。在正常的逻辑里会有各种特殊情况判断。这种弊端：

- 在一开始写程序的时候很容易只考虑正常，然后后面随着发现问题，不断的添加各种if处理，或者代码，破坏原来的结构。
- 正常的逻辑和异常、特殊的逻辑耦合在一块，思考关注点高。

防御式编程：把所有异常、特殊的情况都处理完，代码最后只考虑正常的逻辑。而且要永远认为程序是不安全的，不要人为的假设是正常的。

推荐

- 《代码大全》

用自然的语言写代码

一直发现一个很奇怪的问题：很多人会把一个自然的需求，通过自己的加工，然后加上自己理解的算法，最后会很复杂的代码把自己弄晕了。

在写复杂算法代码和复杂sql的时候特别明显。

其实需求明确后，用自然的语言转为伪代码，最后的伪代码，只是根据不同的语言特性，纯翻译一下。

进阶Java

以前只停留在会使用Java语法的阶段。后来发现在高性能、高并发或者架构设计上的时候，只是了解语法是力不从心的。

java虚拟机

- 内存。跟踪内存泄露，调优应用。
- 类加载。解决类冲突。
- 字节码。不入侵业务的添加监控。远程调试。

多线程

- 锁。
- 线程。
- 异步。

力荐

- 《深入理解Java虚拟机》

深入面向对象

面向对象是一种抽象简化问题思想方式。同时它通过经典的特性：封装、继承、多态来解决在面向对象抽象简化过程中常见的问题。

[漫淡面向对象——POJO对象]

[漫淡面向对象——算法|设计模式]

[漫淡面向对象——抽象问题域分析]

提升面向对象的方式，多想！多想！多想！快捷提升的方法

- 领域驱动设计
- UML
- 设计模式

力荐

- 《Thinking in UML》
- 《UML基础、应用和案例》
- 《领域驱动设计》
- 《Head First设计模式》
- 《大话设计模式》

使用面向切面

是一种减少重复代码，减少关注度，降低复杂度的不入侵业务的思想方式。可以让业务更简单、更专注，能够降低复杂度。比如下面的应用场景：

- 权限判断
- 数据聚合
- 数据格式
- 自定义标签引擎
- 事务
- 日志
- 统计
- 监控
- 数据收集等
- 缓存

> 这里指的并不是springMVC或者应用级别。指的是整个系统的各个环节或者解决问题的思考方式。比如：有的缓存，可能会放在代理的节点再加一中间层。监控可能会放到容器级别通过代理去实现。

推荐

- 《spirng in action》里对于AOP的描述
- 《JavaEE互联网轻量级框架整合开发》里面对于AOP的描述

形成架构原则

其实架构本身就是一种思维方式和能力。是对技术规划，选择，以最优的资源真实解决问题，同时又能在扩展和后续的一种平衡之术

推荐

- 《软件构架设计——程序员向架构师转型必备》
- 《软件架构师12项修炼》

定位问题

明确要架构的边界和要解决的问题，是架构的第一步也是很重要的一步。

可以从下面4个维护去思考，找到问题。

- 重复。可以通过封装和自动化或者工具化来解决。
- 关注高。通过封装和分层，流程和规范约定来解决。
- 复杂高。通过封装工具和分层和规范约定来解决。
- 耦合高。通过分层、分步骤和规范约定来解决。

规划

根据解决问题会有多少收益比。节约时间/花费时间：来明确做事的优先级。不要根据技术的好坏和牛逼与否。解决问题才是关键。

合理的拆分，会让后期落地更可控。

- 水平拆分。分层
- 垂直拆分。分步骤

迭代的思维。一步到位和每次验证实践步伐太大，都容易造成夭折。

- 先有再完美。
- 伤其十指，不如断其一指。

毕竟是商业，生产环境，所以要求稳。

- 小范围试验。一般来说找用户容忍度高的，商业价值影响相对小的去试验。
- 做好回退的方案或者备用方案。

设计

设计是为了把风险降到最低，对一些风险高的地方提前准备和集中解决，以免后面落地的时候返工、甚至推翻重做。不是所有的内容都需要设计，这样就是过度设计。

开源

优先使用第三方，尽量不要重复造轮子。直接使用开源的第三方，需要考虑下面的问题。

- 生态（社区）
- 先进性
- 成功案例
- 活跃性
- 扩展性
- 易用性
- 合理性
- 优缺点

自研

如果找不到适合的轮子。可以通过下面的方式去思考，分析，分解设计要做的事。

- 分而治之

    - 分层
    - 分步骤
    - 职责清晰
- 抽象、建模
- 设计模式
- 思想
    - 面向对象
    - 面向切面
    - 面向插件
    - 面向服务

实施

- 质量
    - 代码质量
    - 软件质量
    - 需求质量
- 研发
    - 核心代码
    - 算法
    - 关键节点的实现（复杂度高的）
- 指导
    - 有现成的解决方案

验证

演示是最好的办法。一来能show一下成果，二来群众的眼睛是雪亮的，发现问题的角度更不同。

优化

在解决完问题后。还需要从易用性、扩展性去考虑。

- 文档化
    - 文档
    - Demo
- 易用性
    - 暴露出去需要使用者知道的参数越少越好。
    - 常用的一些场景对应的参数越自然赵好。不要让用户思考太多。
- 扩展性
    - 在不需要知道太多细节的情况下，更方便的扩展

落地跨职能架构

今年一共落地前端Web（Vue），后台（SpringMVC+mybatis），混合（React Native）以及优化应用架构。每种架构对应的领域技术和需要的能力有所不同。但是架构原则几乎是一样的。具体的后面再补充

- 前端Web架构（Vue和React，其实抽象上是一致的，细节不太一样）。
- 混合架构（React Native)。和前端的架构绝大数雷同，只是额外需要考虑和原生交互和移动端原生等问题。
- 后台架构（Java和Nodejs，涉及到的领域也是一样的，细节和语法、工具、周边不太一样）
- 应用架构。通过项目管理，版本管理，自动化持续集成等方式把一个产品以更快、最优方式转成互联网可以提供服务的服务。
- 服务化架构。今年主要对分层职责定义，服务边界定义更明确了，数据聚合和设计，只是通过jar包的方式解决，下一步可以实践使用服务化和微服务相关的技术落地。

优化效率

多一个技能，能节省后面很多的时间，时间质量会越来越高。后面会专门花点时间做些这方面的专题。

办公

- typora(markdown)
- everything(找文件)
- xmind（帮助思考、固化和解决问题的神器）
- licecap（制作gif图的神器）
- excel (在一些转数据的场景，帮了大忙，有的写sql或者程序要几个小时甚至几天的，用excel几乎几分钟内就完成)
- ...

客户端

- redis-desktop-manager
- navicat
- SecureCRT
- ...

IDEA

- Intellj idea
- webstorm
- Vim配置
- notepad++
- ...

开发辅助

- jd-gui
- fiddler
- chorme控制台
- beyond compare
- ...

网站

- github：管理知识和资源
- 百度网盘：管理资源和软件
- 为知：管理网文

- doit：时间管理
- worktile|teambition：项目管理
- 博客园：知识固化和自我营销。
- ....

了解大数据

- 是一种分布式计算的解决方案和生态圈。
- 和传统的数据库的建模和理解是雷同的。
- 了解了能解决的问题，以及企业对大数据的诉求有哪些。

团队招聘

明确要招过来的人职责是什么，具体工作和定位是什么。然后就是怎么验证他们是否符合这些能力。

开放式问题

封闭的答案往往，不能体现一个人的能力而且容易是背出来的。

往下继续问三级或者连环问三个以上的问题。

- 了解知道的知识面和关注度
- 思维是否清晰
- 是否真实解决过问题。

有价值的问题

很容易知道关注度还有就是层次。

比如去年最有价值的事，去年做过最难的事。

相对多彩的生活

越努力越精彩。滑雪、高尔夫、泡温泉、张北草原、攀岩、承德避暑、键盘控、文具控.....有点遗憾的是陪伴家人的时间还是比较少。主要也是因为一些事限制自己，没有明确的目标，时间黑洞比较多，所以只能用加班的方式来完成了。

宏图大业

2017年过后，有种什么感觉呢——尴尬！感觉高不成，低不就，后面反思了一下，还是因为没有很扎实的迈入高端的职业生涯里，所以计划2018年沉下心来，优先攻技术。

另外越来越觉得资源很重要，解决问题的途径并不仅限于自己，能解决问题就是本事，所以想扩大圈子，也想做些自己的事，虽然我也没太想明白要做什么，但是总归还是要开始做些。

有的时候感觉自己的激情越来越少，一回头发现原来是因为生活

实战微服务

- 搭建环境，熟悉相关的技术栈。

- 用于生产环境。整理出遇到的问题。
- 整理系列的知识体系并且固化。

实战大数据

- 搭建环境，熟悉相关的技术栈。不涉及到具体的大数据的分析算法。
- 用于生产环境。整理出遇到的问题。
- 整理系列的知识体系并且固化。

完善应用架构

希望真实的落地到具体的应用和生产环境中，并且完善细节。现在比较尴尬，一说都知道，有的也在用，但是没有做到极致。

技术解决方案

- 消息队列
- 全文索引
- 日志系统

企业解决方案

- 工作流
- 权限

经典框架

- 完整的企业后台系统

扩大圈子

- 好友
- 校友
- 同事
- 特定群体

营销自我

- 博客（网站）
- github（知识管理和更新）
- 平时的积累和总结，系统出视频。需要一个牛逼的工具

出国旅游

世界这么大，想出去走走。需要更明确的预算和计划

再照婚纱

主题：花海。圆原来的承诺

健康身体

- 减肥。也是今年的一个遗憾，都减下来了，又因为一些事情反弹上去了。主要还是要保持一个良好的心情和合理的饮食和生活习惯。
- 看病。拒绝安慰自己，有不舒服的都去看看，并且支持把小毛病看怎么处理下。比如原来的牙疼哈。