过度设计的架构师们，应该拿去祭天

托尼学长 2023-10-08 41,682 阅读10分钟

智能总结 复制 重新生成

文章列举了多种过度设计的架构师案例，包括过度拆分微服务、设计过多数据库表和库、盲目追求大中台、滥用消息队列和缓存、沉迷设计模式、建立复杂关系、强调数据库可移植性、过度数据校验和过度配置化等，指出应遵循 KISS 原则，保持简单。

关联问题: 如何避免过度设计 怎样才算适度架构 微服务怎样算适度

基于该文章内容继续向AI提问

我发现一个非常有趣的现象。

十多年前，那时“美女”这个称谓还是非常稀缺值钱的，被这么称呼的女性同胞占比，也就是不到10%的样子。

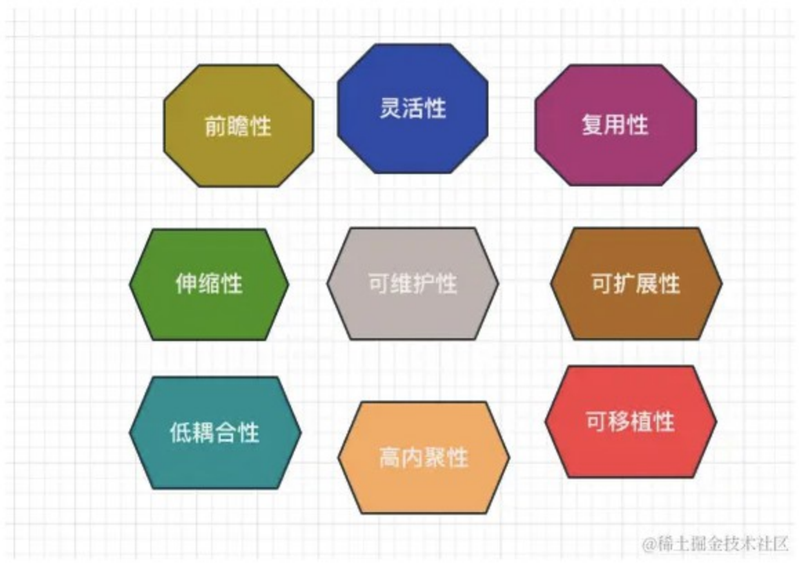
后来就愈发不可收拾了，只要是个女的活的，下至5岁上至50岁的，99%都被人称呼过“美女”。

当然，现在互联网行业的架构师，也越来越“美女化”了，基本上有个两三年工作经验的，带两三个应届生负责过一两个QPS不过十，用户量不过千的小系统的，把项目用SSM框架给搭建起来的，也都成架构师了。

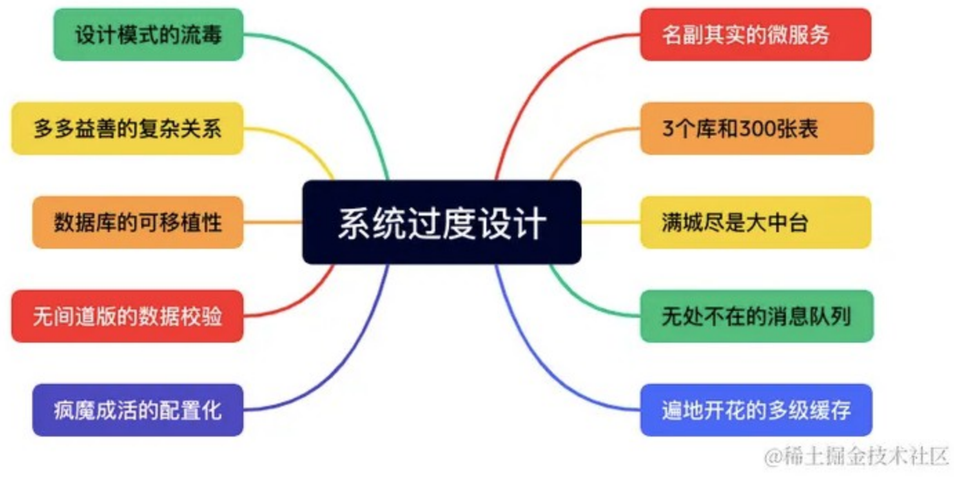
而这些所谓的“架构师”们，如果仅仅是title上的改动，平时工作中该撸代码就撸代码，该摸鱼看网页就看网页，其实也真的没什么。

最最最最怕的就是，他们觉得自己的身份已经变了，是时候该体现出自己作为系统架构师价值的时候了，那一切就会变得不可收拾了。

这些架构师们体现价值的方式当然是做架构设计。按照他们的话说，系统架构要具备前瞻性、灵活性、复用性、伸缩性、可维护性、可扩展性、低耦合性、高内聚性、可移植性。当然，基本上90%都是过度设计。



下面让我们来细数一下，那些年，我所经历过的过度设计。



名副其实的微服务

不久前我面试过一个中小厂架构师，看他的简历上赫然写着，“主导XX系统从单体服务往微服务架构演进工作”。

然后我问他的问题是：“详细说下微服务拆分这件事情，包括：微服务拆分的原因、时机和拆分后的粒度。”

托尼学长 LV.5
榜上有名 优秀作者
77 文章 371k 阅读 1.6k 粉丝
关注 私信

目录 收起

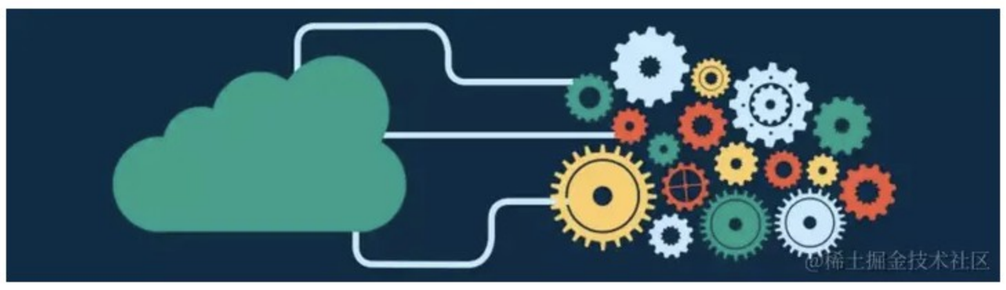
相关推荐
架构师之道：具备什么样的技能才能成...
1.8k阅读 · 10点赞
为什么大部分人做不了架构师?
44k阅读 · 265点赞
架构师之道：架构设计为什么如此重要
1.4k阅读 · 11点赞
架构师之道：为什么需要架构师
4.5k阅读 · 25点赞
架构师1-架构认知、核心能力和基本原则
1.9k阅读 · 9点赞

精选内容
OpenHarmony（鸿蒙南向开发）——小...
塞尔维亚大汉 · 15阅读 · 0点赞
MongoDB聚合查询解析：多维数据关联...
思考的Joey · 23阅读 · 1点赞
MongoDB学习笔记：CRUD基本操作
思考的Joey · 18阅读 · 1点赞
MongoDB：打破数据枷锁的灵活数据库
思考的Joey · 36阅读 · 1点赞
Terraform：像写代码一样管理你的云资源
Y11_推特同名 · 38阅读 · 1点赞

这个架构师说的第一句话就把我雷到了："微服务拆分的粒度，我认为越细越好，不然为什么叫微服务呢？而且，现在的一个很小的微服务，随着业务的持续迭代演进，未来都有可能变得非常庞大，我们做架构设计的，必须要具备前瞻性。"

他接着说："我们的微服务不但按照业务模型进行的拆分，而且我还按照controller层、service层和dao层也做了拆分，这样可以提升代码复用性，你想用我哪层的代码，就可以调用我哪层的API。"

最终，一个单体服务就被他拆分成了这样。



我问他："微服务的'三个火枪手原则'了解吗？"

他摇了摇头，说不清楚。

我心里感慨到，今年阿里云和腾讯云业绩能不能达标，全看这类架构师的了，他们是真费机器啊。

3个库和300张表

去年，跟一个三方公司临时组建了一个项目组，共同开发孵化A项目。

项目联调期间，我跟三方公司的小A说："我刚调用了你们项目的XX接口，新增了20条交易数据，你看看你们接口的业务处理正常吗？数据库里面有这20条数据吗？"

小A说："好的，稍等，我看看。"

20分钟过去了，我问小A看得怎么样了。

小A说："业务处理是正常的，数据我正在一条条找，20条已经找到17条了，我在找剩下的3条。"

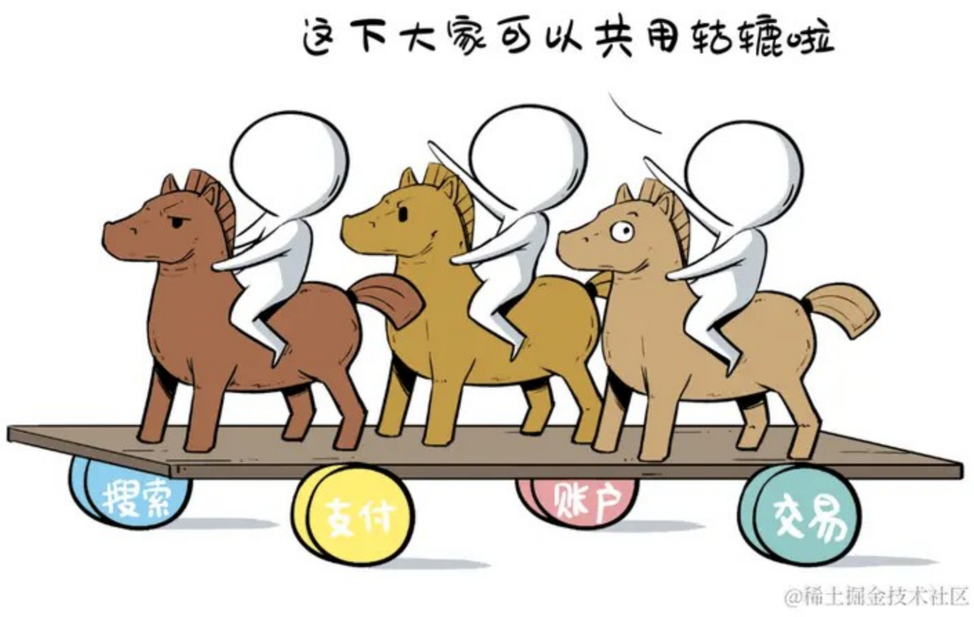
我听得有些懵逼，问小A："你直接从你们订单表里，不能一下子看到20分钟前写入的20条数据吗？为什么还需要一条条找啊？"

小A说："我们的架构师老张，按照每天三百万订单的数据增量，做了一个五年架构规划，已经分好了3个库和300张表。我现在正在根据他的路由规则，一条条地找这些数据。"

满城尽是大中台

呵呵，忽如一夜春风来，满城尽是大中台。

2015年福厂正式提出了"大中台、小前台"的中台战略，通过将原本分散到各个业务的支持部门，比如技术部门、数据部门集中到一起，进行快速的服务迭代，以期更高效地支撑前线，大幅降低支持部门的重复投资建设。



三年后，各个大小互联网公司纷纷跟进，争相建设自己家的中台，也就在这时，某独角兽公司的架构师老范过来找我取经。

我跟老范说："你们的两个主业务是机票和酒店，业务差别太大了，且创新孵化业务并不多，并不适合中台策略。"

老范说："不，中台这个我们一定要搞，因为既是研发团队的政治任务，也是我个人的技术追求。"

半年后，我问老范搞得怎么样了，老范说："唉，讨论了半年哪些职责属于大中台，哪些职责属于小前端，现在还没讨论明白呢。"

无处不在的消息队列

福厂收购了某公司，在收购后的一次技术交流中，我听到对方公司的首席架构师说："MQ是个好东西，能异步处理，能消峰，能解耦，还是应该在项目中多用用的。"

后来发现，大首席架构师的下级执行力真强，MQ真的在他们的项目中无处不在：

- 发短信验证码的场景用MQ，且其生产者和消费者是同一个服务，就为了用MQ异步调用短信服务的SDK；
- 打业务日志的场景用MQ，且其生产者和消费者是同一个服务，就是为了用MQ异步打一行日志；
- TPS个位数的约课场景用MQ，且其生产者和消费者是同一个服务，其美名曰进行消峰；
- 各服务间的通信基本上80%都用了MQ，而不是RPC，其美名曰系统解耦；

牛逼Class！

遍地开花的多级缓存

对，对，还是上次的那个首席架构师，他除了爱用消息队列外，还特别喜欢用缓存，而且是Guava Cache + Redis的多级缓存。

据同事说，这种多级缓存策略在这位首席架构师的熏陶下，已经遍布了OA系统、公司官网、消息中心、结算系统、供应链系统、CRM系统。

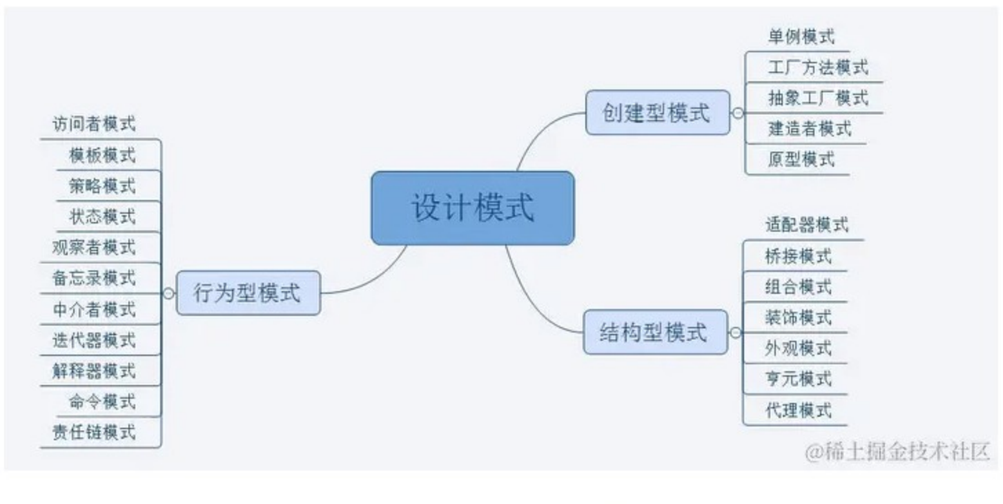
首席架构师说："缓存不仅能提升性能，还能帮助数据库抗压，提升系统可用性，绝对是个好东西，应该多用一用。"

然后，公司的系统就经常发生多种缓存的数据与数据库的数据一致性问题。

首席架构师又说："任何架构都是有利有弊的，但只要利大于弊就好，不要太在意。"

设计模式的流毒

记得我刚上班不久，组内有一个架构师同事，写的代码巨复杂，各种技巧、设计模式、高级语法满天飞，还沾沾自喜的给我们炫耀。



一次Code Review的时候，我嘴欠问他这里咋这么设计，他就鄙视的说："你连这个都不知道，这是设计模式中的建造者模式啊。"

当时觉得他好牛逼，而我好low。

以后，每次进行Code Review，只要看到其他同事代码里有几个if else，架构师同事就质问道：“为什么不用策略模式优化if else？”

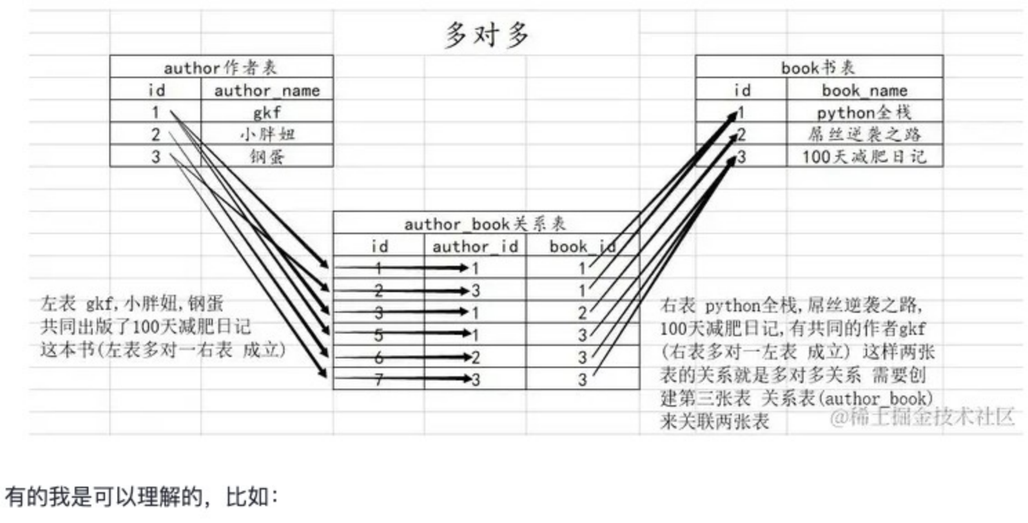
当然，还有其他的质问，类似于：这块为什么不用抽象工厂模式？这块为什么不用代理模式？这块为什么不用观察者模式？

后来我们就给他起了个外号，叫“设模”（se mo）。

多多益善的复杂关系

前面说的那些架构师们，他们过度设计所带来的后果是浪费服务器和研发资源，但架构师老邓不一样，他的过度设计是浪费表。

之前见过某在线教育公司设计的表结构，基本上所有表之间的外键关系都是按照多对多方式设计的，也就是加一个中间的关系映射表。



有的我是可以理解的，比如：

- 一个学生会出现在多个不同的班级里，而一个班级里也会有不同的学生；
- 一个学生可以学习多门课程，而每门课程又会对多个学生进行学习；
- 一个学生可以上多个老师的课，而一个老师又可以教多个学生；

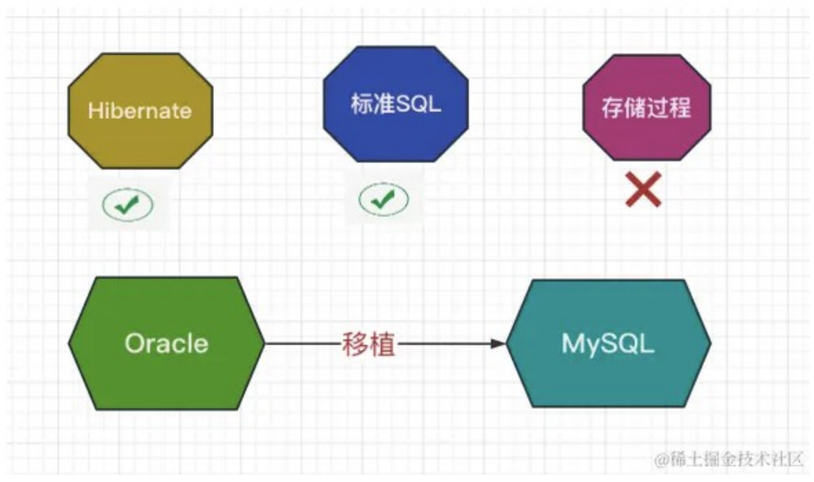
但是，但是。

- 一个学生可以有多个考试成绩，难道一个考试成绩还能属于多个学生吗？
- 一个学生有多个课程的课时余额，难道一个课时余额还能属于多个学生吗？

老邓说：“万一以后业务变化了呢？一切皆有可能啊。”

数据库的可移植性

还在上大学的时候，在CSDN上看某著名架构师在极力强调数据库的可移植性。



我记得当时的原话大概是：

- Hibernate的HQL可以帮我们保证不同数据库之间的移植性，比如：MySQL中的limit和Oracle中的rownum。
- 为什么不能写存储过程？一个重要的原因就是业务逻辑放到数据库里会导致数据库移植成本变大。
- 程序内尽量采用标准SQL语法，因为我们要考虑将来的移植风险。

当时听了，觉得这个大架构师简直就是YYDS。然后我工作了这么多年，也没遇到过一次数据库移植。

无间道版的数据校验

我厂某团队的架构师老李素以严谨著称，其经常放在嘴边的一句话就是："工程师不仅仅是一项有创造性的职业，也是一门严谨审慎的职业。"

这话说的确实没毛病，我也看过他们团队的工程代码，程序的边界处理、异常处理和容错处理做得都特别好，入参校验也是特别细致入微。

就像老李所说的那样："All input is evil。"

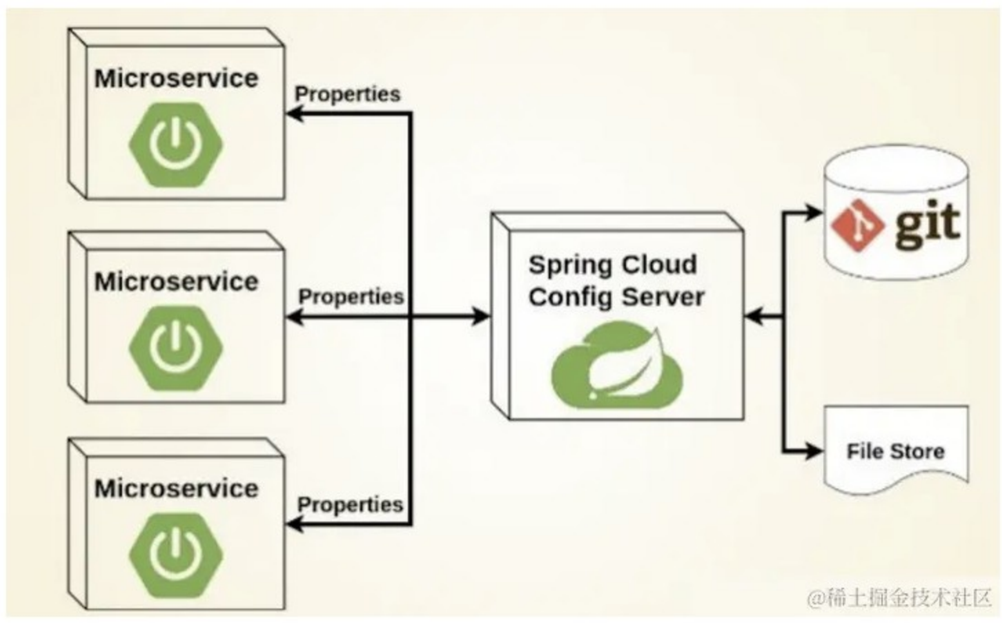
不，等等，入参校验没问题，但怎么从数据库里读出来的数据，为什么还要再校验一遍？难道不是在写入的时候校验吗？

老李面无表情地说："如果数据库中的数据，并没有经过应用程序处理，而是不知道被谁直接改库了呢？"

卧槽，这是泥马数据校验无间道版吗？

疯魔成活的配置化

还是上面的那个架构师老李，他要求团队代码中带数字的地方，全部走配置中心，这样可以不发布代码就直接进行修改。



然后，我就看到了这样的现象：

- 如果某个HashMap的size大于0，则进行xxxx，这个0写到了配置中心里。
- 如果用户性别等于1（男性），则进行男装推荐，这个1写到了配置中心里。
- 如果商品状态等于2（已下线），则进行xxxx，这个2写到了配置中心里。

配置中心啊，你的责任真的好重大。

总结

遇到这种类型的架构师，真的特别想把他们祭天了，因为我是Kiss原则的忠实拥趸。

Keep it simple，stupid，即：保持简单、愚蠢。

保持简单就能让系统运行更好，更容易维护扩展，越是资深的人，越明白这个道理。

标签： 后端 设计模式 架构

评论 235

登录 / 注册 即可发布评论!

最热 | 最新

用户1945244712263 热评
数据库的可移植性，这个必须的，你遇到的少不代表就是错的
1年前 28 38 ...

托尼学长 作者 : 你对，我错
1年前 9 回复 ...

用户29278130... 回复 托尼学长 作者 : 不不，是你对，他错，上面评论必须的，只是在评论区装逼的，不用理会
1年前 4 回复 ...

查看全部 38 条回复

我是你的小饼干 热评
这类文章其实没啥营养，看看爽文就行了，别太认真。单说数据库可移植性这块，可能是作者没接触过，或者产品是自己公司使用没有什么强制要求，金融、医药行业的程序员应该是深有体会，这两年适配各种数据库有多么痛苦。
1年前 40 9 ...

404 子物 : 信创
1年前 3 回复 ...

我是你的小饼干 回复 用户73717808... : 这不叫换，叫兼容
1年前 点赞 回复 ...

查看全部 9 条回复

baifangkual 热评
是不是过度设计 应由时间检验 只能说在你的视角和实际经历中这些故事确实可以说大部分都是过度设计，而且设计模式那个故事 用设计模式本身并没有错 从这块故事来看 仅仅是你（们）和那个设摸关系不好被他顶过罢了
1年前 31 6 ...

imdongrui : 过度用设计模式和spring的高级用法等等，真的很影响后续维护，理解起来是有成本的，更别说一大堆复杂用法凑在一块，因为封装太多设计得太复杂还会影响后续拓展，我现在有个同事就是这样，说他厉害吧，他总是把简单事情复杂化了，说不厉害吧，他确实会很多高级用法，我自己的代码都是尽量简洁，能不用设计模式和高级用法就不用，keep it simple, stupid
1年前 4 回复 ...

用户17714749... : 设计模式 适合基础框架类的东西，业务上的代码还是适合简单的， 让新人快速上手才是主要的， 主要也是由于技术水平参差不齐
1年前 5 回复 ...

查看全部 6 条回复

查看全部 235 条评论

为你推荐

企业架构设计方法论
致问笔记 | 3年前 | 3.2k 9 评论 架构 后端

什么是架构，什么是架构师?
源字节1号 | 2年前 | 3.8k 23 评论 架构

作为一个架构师，我是不是应该有很多职责?
架构师修行之路 | 4年前 | 407 2 评论 微服务 后端

一文搞懂应用架构的3个核心概念
AI架构师汤师爷 | 6月前 | 66 1 评论 架构

架构师之道：架构的演变之路
MobotStone | 10月前 | 2.1k 4 评论 架构 设计

架构师的道法术：架构师是否需要下场写代码?
不惑_ | 1月前 | 598 14 2 后端 前端 架构

一个前端对架构设计的唠嗑
yodfz | 1年前 | 2.6k 33 6 架构 前端

《吃透微服务》 - 服务网关之Gateway
蔡农曰 | 3年前 | 1.7k 19 评论 Java

如何设计实现一个通用的微服务架构? 高可靠、高可用思维模型
程序员斌哥 | 3年前 | 994 3 评论 后端

唯有代码可以治愈一切|2021年中总结
新一代程序员 | 3年前 | 504 4 2 程序员

一个小型公司怎么落地微服务
快乐的提千万 | 3年前 | 1.2k 9 评论 后端

为何项目做到最后都成为“屎山”，代码无法改动
行云创新 | 3年前 | 4.8k 24 17 架构 后端

在成为架构师之前所需了解的一些知识
元闰子 | 4年前 | 2.8k 20 2 架构

架构师之道：为什么需要架构师
MobotStone | 9月前 | 4.5k 25 14 架构 设计

向架构进发
tom_Z | 3年前 | 443 点赞 评论 架构