二

用户故事 从“心”出发，我还有无数个可能

你好，我是 Longslee，很高兴与大家一起学习《高并发系统设计 40 问》。

我从事软件相关的职业已经有九年时间了，之前在一家税务行业类公司工作，目前在一家电信行业相关的公司，从事开发和运维工作。

我并不算“极客时间”的老用户，因为接触“极客时间”只有短短几个月，一开始只抱着试试看的心态，尝试着订阅了几门课程，后来便自然而然地将它当作工作之余，获取信息的必需品。

要说跟这门课结缘，还是在今年 10 月份，那时，我偶然打开“极客时间”，看到了《高并发系统设计 40 问》的课程，被开篇词的题目**“你为什么要学习高并发系统设计”**吸引了。开篇词中提到：

公司业务流量平稳，并不表示不会遇到一些高并发的需求场景；为了避免遇到问题时手忙脚乱，你有必要提前储备足够多的高并发知识，从而具备随时应对可能出现的高并发需求场景的能力……

这些信息着实戳中了我。

回想起来，自己所处的行业是非常传统的 IT 行业，几乎与“互联网”不着边，所以我平时特别难接触一线的技术栈。然而，虽然行业传统，但并不妨碍日常工作中高并发的出现，**比如，偶尔出现的线上促销活动。**

单纯从我自己的角度出发，除了因为开篇词戳中之外，选择这个课程，还在于自己想拓宽视野、激发潜能，另一方面，当真的遇到“高并发”时，不至于望洋兴叹，脑海一片空白。

在课程设计上，每一节课的标题都是以问号结束，这种看似寻常的设计，很容易让我在学习时，联想到自己的实际工作，从而先问问自己：我们为什么要架构分层？如何避免消息重复？等等，自己有了一些答案后，再进入正式的学习，对概念性的知识查漏补缺。

我个人认为，这也算是这门课程的一个小的特色。唐扬老师抛出问题，并用自己的经验进行回答，让这篇文章有了一个很好的闭环。

目前来说，我所在的行业和项目，为了应对日益复杂的业务场景，和日渐频繁的促销活动，也在慢慢地转变，更多地引入互联网行业知识，产品也更加与时俱进。

作为这个行业的一员，在日常工作中，我自然也遇到了一些难题，碰到了一些瓶颈，但是在寻找解决方式的时候，往往局限在自己擅长的技术体系和历史的过往经验上。而在学习了这门课之后，我拓宽了眼界，会不自主地思考“是不是可以用今天学到的方式解决某些问题？”“当初选用的中间件和使用方式合不合理？”等等。

而且，就像我提到的，自己所处的行业在不断改变，其实，就目前的趋势来看，很早就存在的信息化产品和目前主流的互联网产品渐渐难以界定了。就比如高校的教务系统，听起来好像跟我们接触的各类网站大不一样，但是在开学的时候，又有多少选课系统能扛住同学们瞬间的巨大流量呢？

《17 | 消息队列：秒杀时如何处理每秒上万次的下单请求？》讲的就是各厂处理可预见且短时间内大流量的“套路”，而我认为，这个“套路”也可以应用到大学的选课系统。因为教务系统在通常情况下都是很闲的，如果整体升级来提高 QPS 性价比太低，所以只要保证在选课时，服务的稳定性就好了。这里可以引入消息队列，来缓解数据库的压力，再通过异步拆分，提高核心业务的处理速度。

其实，还有好多节课都给我留下了深刻的印象，比如，第 2 讲、第 10 讲、第 13 讲等等。

单看《02 | 架构分层：我们为什么一定要这么做？》这个题目，我一开始会觉得“老生常谈”，软件分层在实际项目中运用的太多太多了，老师为什么单独拿出来一讲介绍呢？然而当我看到“如果业务逻辑很简单的话，可不可以从表示层直接到数据访问层，甚至直接读数据库呢？”这句话时，**联系到了自己的实际业务：**

我所参与的一个工程，确实因为业务逻辑基本等同数据库逻辑，所以从表示层直接与数据访问层交互了。但是如果数据库或者数据访问层发生改动，那将要修改表示层的多个地方，万一漏掉了需要调整的地方，连问题都不好查了，并且如果以后再无意地引入逻辑层，修改的层次也将变多。

对我而言，这篇文章能够有触动我的地方，引发我的思考，所以在接下来的项目中，我坚持选用分层架构。

而《10 | 发号器：如何保证分库分表后 ID 的全局唯一性》**给我的项目提供了思路：**我的需求不是保证分库分表后，主键的唯一性，但由于需要给各个客户端分配唯一 ID，用客户端策略难免重复，所以在读到：

一种是嵌入到业务代码里，也就是分布在业务服务器中。这种方案的好处是业务代码在使用的时候不需要跨网络调用，性能上会好一些，但是就需要更多的机器 ID 位数来支持更多的

业务服务器。另外，由于业务服务器的数量很多，我们很难保证机器 ID 的唯一性，所以就需要引入 ZooKeeper 等分布式一致性组件，来保证每次机器重启时都能获得唯一的机器 ID……

我采取了类似发号器的概念，并且摒弃了之前 UUID 似的算法。采用发号器分发的 ID 后，在数据库排序性能有所提升，业务含义也更强了。

除此之外，在学习《13 | 缓存的使用姿势（一）：如何选择缓存的读写策略？》之前，我的项目中没有过多地考虑，数据库与缓存的一致性。比如，我在写入数据时，选择了先写数据库，再写缓存，考虑到写数据库失败后事务回滚，缓存也不会被写入；如果缓存写入失败，再设计重试机制。

看起来好像蛮 OK 的样子，但是因为没有考虑到在多线程更新的情况下，确实会造成双方的不一致，**所造成的后果是：有时候从前端查询到的结果与真实数据不符。**后来，根据唐扬老师提到的 Cache Aside（旁路缓存）策略，我顿然醒悟，然后将这一策略用于该工程中，效果不错。这节课，我从唐扬老师的亲身经历中，学到了不少的经验，直接用到了自己的项目中。

真的很感谢唐扬老师，也很开心能够遇到这门课程，在这里，想由衷地表达自己的感谢之情。

那么我是怎么学习这门课程的呢？在这里，我想分享几点：

知行合一

学完课程后，除了积极思考“能否用”“怎么用”“何时用”这些问题外，一定要趁热打铁，要么继续深入话题，翻阅其他资料，巩固下来；要么敲敲代码实现一遍，化为自己的技能；如果时间充裕，甚至可以立马着手改进项目。

留言区 = 挖宝区

每节课结束，我都会在留言板留下疑问，或者分享体验，我喜欢问问题其实是跟自己在大学时，参加的一场宣讲会有关。当时，来招聘的负责人是一位美国留学回来的台湾工程师，他介绍完后问大家有没有疑问，并没有人回答。

后来，他讲了一个经历，使我感慨良多。他说当他刚去美国大学的时候，教授讲完课就要答疑，一个白人学生提了一个，在中国学生看来十分简单且幼稚的问题，以后的每节课，这位白人同学都要提问，渐渐地，提的问题他都听不懂了！再后来，教授也不懂了。

所以，我会不断地发问，不懂就问，把留言区当作挖宝区，看大家的留言，进行思考。比如@李冲同学的几个跟帖，就解答了我对布隆过滤器的误解，并且还知道了另一种布谷鸟过

滤器。

勤做笔记

有的时候，我当时理解的比较透彻，可过了两三天之后，就有些模糊了，所以后来，我根据自己的理解写成思维导图形式，随时随地都可以翻阅。另外，在实现这些方案的代码后面，也可以写下相应的注释，Review 的时候还可以温故知新。

在最后，我也想分享一下自己为什么用专栏这种形式来学习。善用搜索引擎的同学们都有体会，搜索出来的知识分布在各处，雷同的不少，有经验的介绍甚少，我没办法在有限的时间内，将搜索到的知识形成体系。

当然了，要想系统地学习可以借助书籍。**但是对我来说，**书籍类学习周期长，章节之间的关联性也不大，容易学了这里忘了那里。书籍多是讲一个专业点，对于跨专业的知识经常一笔带过，而专栏，是有作者自己的理解在里边，前后之间有贯通，学习起来轻松愉悦。

就拿一致性 Hash 这个知识点来说，我从网上看了不少关于一致性 Hash 的文章，但没有看到应用，更别谈应用中的缺陷，有的描述甚至让我误认为节点变化后，数据也会跟着迁移。唐扬老师的《14 | 缓存的使用姿势（二）：缓存如何做到高可用？》，倒是给了我网络上看不到的盲区，通过在留言区与老师交流后，颇有一种豁然开朗的收获感。

当然了，这些只是我个人的感受，见仁见智，**你或许有自己的学习方法，也或许大家的起点不同，**在这里，我只想把自己的真实感受分享出来，也十分感谢大家倾听我的故事。

总的来说，想要提升自己，并没有捷径，只有一步一步地踏实前行，从踩过的坑中，努力地爬出来。

对我来说，唐扬老师的《高并发系统设计 40 问》犹如及时雨一般的存在，弥补了我高并发相关知识上的缺陷，我相信，认真学完课程之后，自己的视野一定有所开拓，职业生涯也会进入新的篇章。

上一页 下一页