



下载APP



加餐4 | 如何理解分布式系统？

2020-07-27 张帆

系统性能调优必知必会

[进入课程 >](#)



讲述：张浩

时长 12:59 大小 11.91M



你好，我是陶辉。课程到现在也已经接近尾声，我看到有的同学已经开始在掉队了。所以今天这讲，我准备来回答大家的一些高频问题。

咱们目前正在学习的这一模块叫“分布式系统优化”，我给你讲了监控、CAP、负载均衡、一致性哈希，说实话，这些知识都不简单，你如果觉得有点难，那也别气馁，因为它确实得多琢磨，我自己一开始学习的时候也是这样。

不过，我发现，在这个模块中，很多同学似乎对分布式有什么误解，有的人说分布式就是多台机器，有的人说分布式就是微服务，总之，大家各有自己的理解。于是，我就想着你写篇加餐，来系统聊聊这个话题。



不过，在查资料的过程中，我发现 InfoQ 上已经有一篇文章很好地回答了这个问题。于是，经过编辑冬青的努力，我们找到了作者张帆，申请到了那篇文章的授权，在这里交付给你。

如果现在让你阐述一下什么是“分布式系统”，你脑子里第一下跳出来的是什么？我想，此时可以用苏东坡先生的一句诗，来形象地描述大家对分布式系统的认识：

横看成岭侧成峰，远近高低各不同。

“分布式系统”等于 SOA、ESB、微服务这些东西吗？

我觉得每个人脑子里一下子涌现出来的肯定是非常具象的东西，就像下面这些：

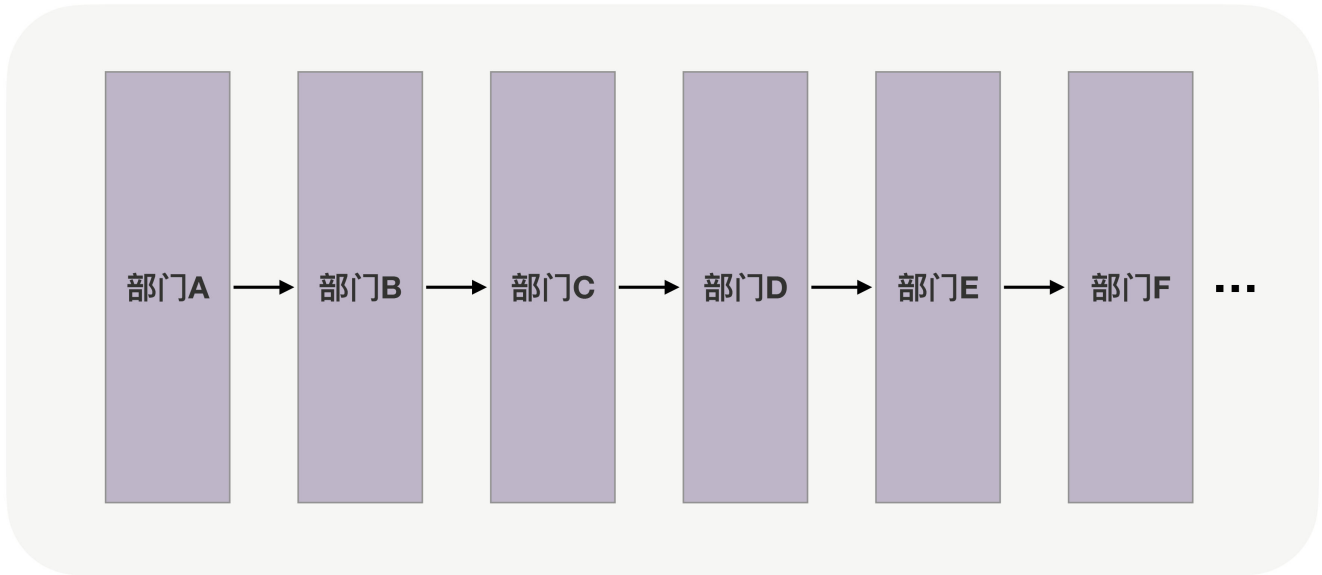
“分布式系统”等于 SOA、ESB、微服务这些东西吗？

如果你一下子想到的是 XX 中心、XX 服务，意味着你把服务化的模式（SOA、ESB、微服务）和分布式系统错误地划上了等号。

那么，什么是“服务化”呢？服务化就像企业当中将相同岗位的人员划分到同一个部门管理，以此来收敛特定的工作入口，再进行二次分配，以提高人员利用率和劳动成果的复用度。服务化的本质是“分治”，而“分治”的前提是先要拆，然后才谈得上如何治。这时，高内聚、低耦合的思想在拆分过程中起到了一个非常重要的作用，因为这可以尽可能地降低拆分后不同组件间进行协作的复杂度。所以重要的是“怎么拆”，还有如何循序渐进地拆，而在这个过程中你究竟是采用了何种服务化模式（比如 SOA、ESB、微服务等）并不是关键。

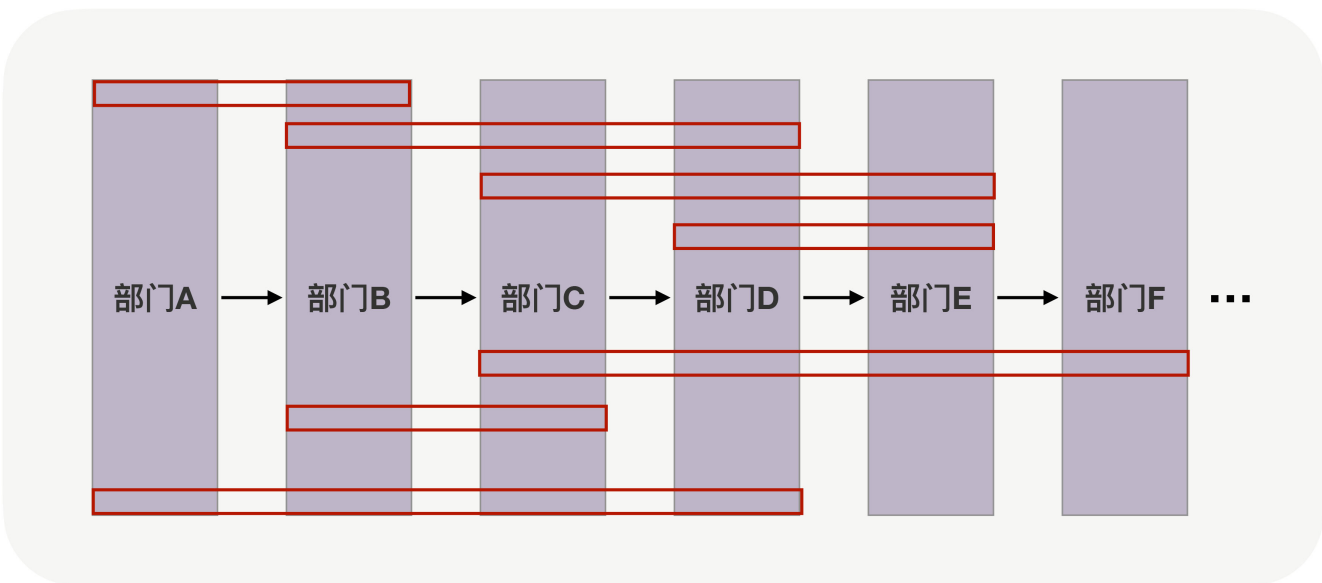
为什么说“怎么拆”最重要呢？我来举个例子，企业的组织架构包括三种模型：职能型、项目型、矩阵型。你可以把这里的企业理解为一个“分布式系统”，把后面的 3 种模型理解为这个分布式系统的 3 种形态。作为这个“系统”的所有人，你需要考虑如何拆分它，才能使得各功能组件相互之间可以更好地协作。假设，你要将一个总计 10000 名员工的企业按“职能型”拆分成 20 个部门，得到的结果是每个部门 500 人。

这时，如果工作是流水线式的上下游关系。一个部门完工了再交给下一个部门。



那么这时候是高内聚、低耦合的。因为一个工种只与另一个工种产生了关联，并且仅有一次。

但如果工作需要频繁的由不同职能的人员同时进行，就会导致同一个部门可能与多个部门产生联系。



那么，这时是低内聚、高耦合的。因为一个工种需要和其他多个工种产生关联并且远不止一次。

可以看到服务化体现了“分治”的效果，这也是分布式系统的核心思想，因此从“分治”这个本质上来看，服务化的确是分布式系统，但分布式系统不仅仅停留在那些服务化的模式上。

我相信，你在工作中参与开发的任何软件系统，到处都存在着需要拆分的地方，除非它的功能极简到只需要计算一个 $1+1$ 。比如，当我们在电商平台点击“提交订单”的时候，会涉及生成订单、扣除积分、扣除库存等等动作。电商系统初期所有的功能可能都在一个系统里面，那么这些操作可以写在一个方法体里吗？我想只要代码能够成功运行，大部分人是不会管你怎么写的。但是如果这时需要增加一个红包功能呢？相信你或多或少遇到过在几百上千行代码中去增改功能的事情，其中的痛苦应该深有体会。

要解决这个问题就是要做拆分，通过梳理、归类，将不同的紧密相关的部分收敛到一个独立的逻辑体中，这个逻辑体可以是函数、类以及命名空间，等等。所以，从这个角度来说“分治”的问题其实早就存在我们的工作中，就看我们是否有去关注它了。因此，这并不只是我们在进行服务化时才需要考虑的问题。

那么如何才能做好这个事情，更好的拆分能力正是我们需要掌握的。如果只是因为看到其他人这么拆，我也这么拆，根据“二八原则”，或许“依样画葫芦”可以达到 80% 的契合度，但是往往那剩下的 20% 会是耗费我们 80% 精力的“大麻烦”。要知道，**只有掌握了核心主旨，才能更快地找到最理想的高内聚、低耦合方案。**

“分布式系统”是各种中间件吗？

又或许，听到分布式系统，你想到了某某 MQ 框架、某某 RPC 框架、某某 DAL 框架，把运用中间件和分布式系统错误地划上了等号。

这里需要你搞清楚的是，中间件起到的是标准化的作用。中间件只是承载这些标准化想法的介质、工具，可以起到引导和约束的效果，以此大大降低系统的复杂度和协作成本。我们来分别看一下：

MQ 框架标准化了不同应用程序间非实时异步通信的方式。

RPC 框架标准化了不同应用程序间实时通讯的方式。

DAL (Data Access Layer , 数据访问层) 框架标准化了应用程序和数据库之间通讯的方式。

所以，**虽然分布式系统中会运用中间件，但分布式系统却不仅仅停留在用了什么中间件上。**你需要清楚每一类中间件背后是对什么进行了标准化，它的目的是什么，带来了哪些


副作用，等等。只有如此，你才能真正识别不同技术框架之间的区别，找到真正适合当前系统的技术框架。

那么标准是拍脑袋决定的吗？肯定不是，正如前面所说每一次标准化都是有目的的，需要产生价值。比如，大部分中间件都具备这样一个价值：

为了在软件系统的迭代过程中，避免将精力过多地花费在某个子功能下众多差异不大的选项中。

在现实中，这点更多时候出现在技术层面的中间件里，比如，数据库访问框架的作用是为了标准化操作不同数据库的差异，使得上层应用程序不用纠结于该怎么与 MySQL 交互或者该怎么与 SQL SERVER 交互。因为与业务相比，技术层面“稳定”多了，所以做标准化更有价值，更能获得长期收益。但“稳定”是相对的，哪怕单纯在业务层面也存在相对稳定的部分。

比如，你可以想象一下“盛饭”的场景，在大多数情况下其中相对稳定的是什么，不稳定的的是什么。想完之后看下面的示例：

 复制代码

```
1  ...
2  基类：人
3  继承基类的子类：男人、女人
4
5  基类：碗
6  继承基类的子类：大碗、小碗、汤碗
7
8  基类：勺子
9  继承基类的子类：铁勺、陶瓷勺、塑料勺
10
11 function 盛饭(参数 人, 参数 碗, 参数 勺子){
12     do 人拿起碗
13     do 人拿起勺子
14     do 人用勺子舀起饭
15     do 人把勺子放到碗的上方并倒下
16
17 }
18 ...
```

从这个示例里我们发现，不稳定的部分都已经成为变量了，那么剩下的这个方法体起到的作用和前面提到的中间件是一样的，它标准化了盛饭的过程。所以识别相对稳定的部分是什么，如何把它们提炼出来，并且围绕这些点进行标准化，才是我们需要掌握的能力。而锻炼这个能力和需要这个能力的地方同样并不局限于分布式系统。

列举这些现象只是想说，我们在认知一个分布式系统的时候，内在胜于表象，掌握一个扎实的理论基本功更为重要。而且，这些训练场无处不在。

海市蜃楼般的“分布式系统”

我相信，自从进入移动时代以来，各种高大上的系统架构图越来越频繁地出现，你的眼前充斥着各种主流、非主流的眼花缭乱的技术框架。你不由得肃然起敬一番，心中呐喊着：“对，这就是我想去的地方，我想参与甚至实现一个这样牛逼的分布式系统，再也不想每天只是增删改查了。”

得不到的事物总是美好的，但往往我们也会过度地高估它的美好。与此类似，高大上的架构图背后呈现的系统的确也是一个成熟分布式系统的样貌，但我们要清楚一点：罗马不是一日建成的。

而且，“分布式”这个词只是意味着形态上是散列状的，而“一分为二”和“一分为N”本质上并没有区别。所以，很多小项目或者大型项目的初期所搭配的基础套餐“单程序 + 单数据库”，同样可以理解为分布式系统，其中遇到的问题很多同样也存在于成熟的分布式系统中。

想象一下，下面的场景是否在“单程序 + 单数据库”项目中出现过？

log 记录执行成功，但是数据库的数据没发生变化；

进程内的缓存数据更新了，但是数据库更新失败了。

这里我们停顿 30 秒，思考一下为什么会出现这些问题？

这里需要我们先思考一下“软件”是什么。软件的本质是一套代码，而代码只是一段文字，除了提供文字所表述的信息之外，本身无法“动”起来。但是，想让它“动”起来，使其能够完成一件我们指定的事情，前提是需要一个宿主来给予它生命。这个宿主就是计

算机，它可以让代码变成一连串可执行的“动作”，然后通过数据这个“燃料”的触发，“动”起来。这个持续的活动过程，又被描述为一个运行中的“进程”。

那么除了我们开发的系统是软件，数据库也是软件，前者负责运算，后者负责存储运算后的结果（也可称为“状态”），分工协作。

所以，“单程序 + 单数据库”为什么也是分布式系统这个问题就很明白了。因为我们所编写的程序运行时所在的进程，和程序中使用到的数据库所在的进程，并不是同一个。也因此导致了，让这两个进程（系统）完成各自的部分，而后最终完成一件完整的事，变得不再像由单个个体独自完成这件事那么简单。这就如“两人三足”游戏一样，如何尽可能地让外部看起来像是一个整体、自然地前进。

所以，我们可以这么理解，涉及多个进程协作才能提供一个完整功能的系统就是“分布式系统”。

那么再回到上面举例的两个场景，我们在思考“单程序 + 单数据库”项目中遇到的这些问题背后的原因和解决它的过程时，与我们在一个成熟的分布式系统中的遭遇是一样的，例如数据一致性。当然，这只是分布式系统核心概念的冰山一角。

维基百科对“分布式系统”的宏观定义是这样的：

分布式系统是一种其组件位于不同的联网计算机上的系统，然后通过互相传递消息来进行通信和协调。为了达到共同的目标，这些组件会相互作用。

我们可以再用大小关系来解释它：把需要进行大量计算的工程数据分割成小块，由多台计算机分别计算，然后将结果统一合并得出数据结论的科学。这本质上就是“分治”。而“单程序 + 单数据库”组合的系统也包含了至少两个进程，“麻雀虽小五脏俱全”，这也是“分布式系统”。

小结

现在，我们搞清楚了，看待一个“分布式系统”的时候，内在胜于表象。以及，只要涉及多个进程协作才能提供一个完整功能的系统，就是“分布式系统”。

我相信还有很多其他景象出现你的脑海中，但这大多数都是分布式系统的本质产生的“化学反应”，进而形成的结果。如果停留在这些表象上，那么我们最终将无法寻找到“分布式系统”的本质，也就无法得到真正的“道”，更不会真正具备驾驭这些形态各异的“分布式系统”的能力。

所以，希望你在学习分布式系统的时候，不要因追逐“术”而丢了“道”。没有“道”只有“术”是空壳，最终会走火入魔，学得越多，会越混乱，到处都是矛盾和疑惑。

以上就是张帆老师的分享，他的观点与本专栏也是不谋而合的。他认为：我们不仅要清楚具体场景下的最佳实践，还要明白为什么这样做，以及该如何去权衡不同方案。我们务必要修炼好自己的内功，形成一套完整的知识体系，完成核心“骨架”的塑造。而在此之后，你自己在课外学习时，就可以去填充“血肉”部分，逐渐丰满自己。未来，大家的区别就在于胖一点和瘦一点，但只要能很好地完成工作，胖瘦又有何影响呢？

最后，有关“分布式系统优化”你还有什么问题吗？欢迎在留言区中一起讨论。

提建议

更多课程推荐

设计模式之美

前 Google 工程师手把手教你写高质量代码

王争

前 Google 工程师

《数据结构与算法之美》专栏作者



涨价倒计时 🕒

限时秒杀 **¥149**，7月31日涨价至 **¥299**

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 29 | 流式计算：如何通过集群实现实时计算？

精选留言 (2)

写留言



webmin

2020-07-27

看待“计算机”视角，进程从程序角度去看就是一台计算机，其实如果粒度再小一点的线程或协程也可以视为一台一台计算机。

在同一台计算机内进程、线程或协程间可以全局变量来共享状态，其实如果把整个分布系统看成一台计算机数据库（关系型DB、NoSQL）就是一个全局变量集。

当然这里说全局变量只是为了讨论方便，实际工作我们还是要通过消息来传递状态，而...
展开 ∨



Jeff.Smile

2020-07-27

看待一个“分布式系统”的时候，内在胜于表象。以及，只要涉及多个进程协作才能提供一个完整功能的系统，就是“分布式系统”。

学到了，至今为止最为简要准确的表述。

展开 ✓

