天天写「业务代码」，如何成为「技术大牛」？

“不管是开发、测试、运维，每个技术人员心理多多少少都有一个成为技术大牛的梦，毕竟“梦想总是要有的，万一实现了呢”！正是对技术梦的追求，促使我们不断地努力和提升自己。然而……

然而“梦想是美好的，现实却是残酷的”，很多同学在实际工作后就会发现，梦想是成为大牛，但做的事情看起来跟大牛都不沾边，例如，程序员说“天天写业务代码还加班，如何才能成为技术大牛”，测试说“每天都有执行不完的测试用例”，运维说“扛机器接网线敲shell命令，这不是我想要的运维人生”……

由于我是程序员，所以以下的一些例子都是基于程序开发的，但大道理是相通的，测试、运维都可以借鉴。

几个典型的误区

拜大牛为师

知乎上有人认为想成为技术大牛最简单直接、快速有效的方式是“拜团队技术大牛为师”，让他们平时给你开小灶，给你分配一些有难度的任务。

我个人是反对这种方法的，主要的原因有几个：

1. 大牛很忙，不太可能单独给你开小灶，更不可能每天都给你开1个小时的小灶；而且一个团队里面，如果大牛平时经常给你开小灶，难免会引起其他团队成员的疑惑，我个人认为如果团队里的大牛如果真正有心的话，多给团队培训是最好的。

然而做过培训的都知道，准备一场培训是很耗费时间的，课件和材料至少2个小时（还不能是碎片时间），讲解1个小时，大牛们一个月做一次培训已经是很高频了。

2. 因为第一个原因，所以一般要找大牛，都是带着问题去请教或者探讨。因为回答或者探讨问题无需太多的时间，更多的是靠经验和积累，这种情况下大牛们都是很乐意的，毕竟影响力是大牛的一个重要指标嘛。

然而也要特别注意：如果经常问那些书本或者google能够很容易查到的知识，大牛们也会很不耐烦的，毕竟时间宝贵。经常有网友问我诸如“jvm的-Xmn参数如何配置”这类问题，我都是直接回答“请直接去google”，因为这样的问题实在是太多了，如果自己不去系统学习，每个都要问是非常浪费自己和别人的时间的。

3. 大牛不多，不太可能每个团队都有技术大牛，只能说团队里面会有比你水平高的人，即使他每天给你开小灶，最终你也只能提升到他的水平；而如果是跨团队的技术大牛，由于工作安排和分配的原因，直接请教和辅导的机会是比较少的，单凭参加几次大牛的培训，是不太可能就成为技术大牛的。

综合上述的几个原因，我认为对于大部分人来说，要想成为技术大牛，首先还是要明白“主要靠自己”这个道理，不要期望有个像武功师傅一样的大牛手把手一步一步的教你。适当的时候可以通过请教大牛或者和大牛探讨来提升自己，但大部分时间还是自己系统性、有针对性的提升。

业务代码一样很牛逼

知乎上有的回答认为写业务代码一样可以很牛逼，理由是业务代码一样可以有各种技巧，例如可以使用封装和抽象使得业务代码更具可扩展性，可以通过和产品多交流以便更好的理解和实现业务，日志记录好了问题定位效率可以提升10倍……等等。

业务代码一样有技术含量，这点是肯定的，业务代码中的技术是每个程序员的基础，但只是掌握了这些技巧，并不能成为技术大牛。就像游戏中升级打怪一样，开始打小怪，经验值很高，越到后面经验值越少，打小怪已经不能提升经验值了。这个时候就需要打一些更高级的怪，刷一些有挑战的副本了，没看到哪个游戏只要一直打小怪就能升到顶级的。

成为技术大牛的路也是类似的，你要不断的提升自己的水平，然后面临更大的挑战，通过应对这些挑战从而使自己水平更上一级，然后如此往复，最终达到技术大牛甚至业界大牛的境界，写业务代码只是这个打怪升级路上的一个挑战而已，而且我认为是比较初级的一个挑战。

所以我认为：**业务代码都写不好的程序员肯定无法成为技术大牛，但只把业务代码写好的程序员也还不能成为技术大牛。**

上班太忙没时间自学

很多人认为自己没有成为技术大牛并不是自己不聪明，也不是自己不努力，而是中国的这个环境下，技术人员加班都太多了，导致自己没有额外的时间进行学习。

这个理由有一定的客观性，毕竟和欧美相比，我们的加班确实要多一些，但这个因素只是一个需要克服的问题，并不是不可逾越的鸿沟，毕竟我们身边还是有那么多的大牛也是在中国这个环境成长起来的。

我认为有几个误区导致了这种看法的形成：

1、上班做的都是重复工作，要想提升必须自己额外去学习

形成这个误区的主要原因还是在于认为“写业务代码是没有技术含量的”，而我现在上班就是写业务代码，所以我在工作中不能提升。

2、学习需要大段的连续时间

很多人以为要学习就要像学校上课一样，给你一整天时间来上课才算学习，而我们平时加班又比较多，周末累的只想睡懒觉，或者只想去看看电影打打游戏来放松，所以就没有时间学习了。

实际上的做法正好相反：首先我们应该在工作中学习和提升，因为学以致用或者有实例参考，学习的效果是最好的；其次工作后学习不需要大段时间，而是要挤出时间，利用时间碎片来学习。我会在接下来的篇幅讲“如何在工作中学习提升”，至于如何利用时间碎片来学习，可以参考：技术人的小目标：10000小时理论落地，你就是大牛

正确的做法

Do more

做的更多，做的比你主管安排给你的任务更多。

我在HW的时候，负责一个版本的开发，这个版本的工作量大约是2000行左右，但是我除了做完这个功能，还将关联的功能全部掌握清楚了，代码（大约10000行）也全部看了一遍，做完这个版本后，我对这个版本相关的整套业务全部很熟悉了。

经过一两次会议后，大家发现我对这块掌握最熟了，接下来就有趣了：产品讨论需求找我、测试有问题也找我、老大对外支撑也找我；后来，不是我负责的功能他们也找我，即使我当时不知道，我也会看代码或者找文档帮他们回答……最后我就成了我这个系统的“专家”了。虽然这个时候我还是做业务的，还是写业务代码，但是我已经对整个业务都很熟悉了。
以上只是一个简单的例子，其实就是想说：要想有机会，首先你得从人群中冒出来，要想冒出来，你就必须做到与众不同，要做到与众不同，你就要做得更多！

怎么做得更多呢？可以从以下几个方面着手：

1、熟悉更多业务，不管是不是你负责的；熟悉更多代码，不管是不是你写的
这样做有很多好处，举几个简单的例子：

- 需求分析的时候更加准确，能够在需求阶段就识别风险、影响、难点
- 问题处理的时候更加快速，因为相关的业务和代码都熟悉，能够快速的判断问题可能的原因并进行排查处理
- 方案设计的时候考虑更加周全，由于有对全局业务的理解，能够设计出更好的方案

2、熟悉端到端
比如说你负责web后台开发，但实际上用户发起一个http请求，要经过很多中间步骤才到你的服务器（例如浏览器缓存、DNS、nginx等），服务器一般又会经过很多处理才到你写的绑陪分代码（路由、权限等）这整个流程中的很多系统或者步骤，绝大部分人是不可能去参与写代码的，但掌握了这些知识对你的综合水平有很大作用，例如方案设计、线上故障处理这些更加有含金量的技术工作都需要综合技术水平。
“系统性”、“全局性”、“综合性”这些字眼看起来比较虚，但其实都是技术大牛的必备的素质，要达到这样的境界，必须去熟悉更多系统、业务、代码。

3、自学
一般在比较成熟的团队，由于框架或者组件已经进行了大量的封装，写业务代码所用到的技术确实也比较少，但我们要明白“唯一不变的只有变化”，框架有可能要改进，组件可能要替换，现有技术可能已经无法满足业务需求，或者你换了一家公司，新公司既没有组件也没有框架，要你从头开始来做。
这些都是机会，也是挑战，而机会和挑战只会分配给有准备的人，所以这种情况下我们更加需要自学更多东西，因为真正等到要用的时候再来学已经没有时间了。
以java为例，大部分业务代码就是if-else加个数据库操作，但我们完全可以自己学些更多java的知识，例如垃圾回收，调优，网络编程等，这些可能暂时没用，但真要用的时候，不是google一下就可以了，这个时候谁已经掌握了相关知识和技能，机会就是谁的。
以垃圾回收为例，我自己平时就抽时间学习了这些知识，学了1年都没用上，但后来用上了几次，每次都解决了卡死的大问题，而有的同学，写了几年的java代码，对于stop-the-world是什么概念都不知道，更不用说去优化了。
特别是很多开源软件，更加需要自己平时去自学，例如Nginx、Redis、Mongodb、ElasticSearch等，在合适的时机引入这些技术，能够带来很大的价值。

Do better
要知道这个世界上没有完美的东西，你负责的系统和业务，总有不合理和可以改进的地方，这些“不合理”和“可改进”的地方，都是更高级别的怪物，打完后能够增加更多的经验值。识别出这些地方，并且给出解决方案，然后向主管提出，一次不行两次，多提几次，只要有一次落地了，这就是你的机会。
例如：

- 重复代码太多，是否可以引入设计模式？
- 系统性能一般，可否进行优化？
- 目前是单机，如果做成双机是否更好？
- 版本开发质量不高，是否引入高效的单元测试和集成测试方案？
- 目前的系统太庞大，是否可以通过重构和解耦改为3个系统？
- 阿里中间件有一些系统感觉我们也可以用，是否可以引入 ？

只要你去想，其实总能发现可以改进的地方的；如果你觉得系统哪里都没有改进的地方，那就说明你的水平还不够，可以多学习相关技术，多看看业界其它公司怎么做，BAT都怎么做。
我2013年调配到九游，刚开始接手了一个简单的后台系统，每天就是配合前台做数据增删改查，看起来完全没意思，是吧？如果只做这些确实没意思，但我们接手后做了很多事情：

- 解耦，将一个后台拆分为2个后台，提升可扩展性和稳定性；
- 双机，将单机改为双机系统，提高可靠性；
- 优化，将原来一个耗时5小时的接口优化为耗时5分钟

还有其它很多优化，后来我们这个组承担了更多的系统，后来这个小组5个人，负责了6个系统。

Do exercise
在做职业等级沟通的时候，发现有很多同学确实也在尝试Do more、Do better，但在执行的过程中，几乎每个人都遇到同一个问题：光看不用效果很差，怎么办？
例如：

- 学习了jvm的垃圾回收，但是线上比较少出现FGC导致的卡顿问题，就算出现了，恢复业务也是第一位的，不太可能线上出现问题然后让每个同学都去练一下手，那怎么去实践这些jvm的知识和技能呢？
- Netty我也看了，也了解了Reactor的原理，但是我不可能参与Netty开发，怎么去让自己真正掌握Reactor异步模式呢？
- 看了《高性能MySQL》，但是线上的数据库都是DBA管理的，测试环境的数据库感觉又是随便配置的，我怎么去验证这些技术呢？
- 框架封装了DAL层，数据库的访问我们都不需要操心，我们怎么去了解分库分表实现？

诸如此类问题还有很多，我这里分享一下个人的经验，其实就是3个词：learning、trying、teaching！

1、Learning

这个是第一阶段，看书、google、看视频、看别人的博客都可以，但要注意一点是“系统化”，特别是一些基础性的东西，例如JVM原理、Java编程、网络编程，HTTP协议等等，这些基础技术不能只通过google或者博客学习，我的做法一般是先完整的看完一本书全面的了解，然后再通过google、视频、博客去有针对性的查找一些有疑问的地方，或者一些技巧。

2、Trying

这个步骤就是解答前面提到的很多同学的疑惑的关键点，形象来说就是“自己动手丰衣足食”，也就是自己去尝试搭建一些模拟环境，自己写一些测试程序。例如：

- Jvm垃圾回收：可以自己写一个简单的测试程序，分配内存不释放，然后调整各种jvm启动参数，再运行的过程中使用jstack、jstat等命令查看jvm的堆内存分布和垃圾回收情况。这样的程序写起来很简单，简单一点的就几行，复杂一点的也就几十行。
- Reactor原理：自己真正去尝试写一个Reactor模式的Demo，不要以为这个很难，最简单的Reactor模式代码量（包括注释）不超过200行（可以参考Doug Lee的PPT）。自己写完后，再去看看netty怎么做，一对比理解就更加深刻了。
- MySQL：既然有线上的配置可以参考，那可以直接让DBA将线上配置发给我们（注意去掉敏感信息），直接学习；然后自己搭建一个MySQL环境，用线上的配置启动；要知道很多同学用了很多年MySQL，但是连个简单的MySQL环境都搭不起来。
- 框架封装了DAL层：可以自己用JDBC尝试去写一个分库分表的简单实现，然后与框架的实现进行对比，看看差异在哪里。
- 用浏览器的工具查看HTTP缓存实现，看看不同种类的网站，不同类型的资源，具体是如何控制缓存的；也可以自己用Python写一个简单的HTTP服务器，模拟返回各种HTTP Headers来观察浏览器的反应。

还有很多方法，这里就不一一列举，简单来说，就是要将学到的东西真正试试，才能理解更加深刻，印第安人有一句谚语：I hear and I forget. I see and I remember. I do and I understand，而且“试试”其实可以比较简单，很多时候我们都可以自己动手做。

当然，如果能够在实际工作中使用，效果会更好，毕竟实际的线上环境和业务复杂度不是我们写个模拟程序就能够模拟的，但这样的机会可遇不可求，大部分情况我们还真的只能靠自己模拟，然后等到真正业务要用的时候，能够信手拈来。

3、Teaching

一般来说，经过Learning和Trying，能掌握70%左右，但要真正掌握，我觉得一定要做到能够跟别人讲清楚。因为在讲的时候，我们既需要将一个知识点系统化，也需要考虑各种细节，这会促使我们进一步思考和学习。同时，讲出来后看或者听的人可以有不同的理解，或者有新的补充，这相当于继续完善了整个知识技能体系。

这样的例子很多，包括我自己写博客的时候经常遇到，本来我觉得自己已经掌握很全面了，但一写就发现很多点没考虑到；组内培训的时候也经常看到，有的同学写了PPT，但是讲的时候，大家一问，或者一讨论，就会发现很多点还没有讲清楚，或者有的点其实是理解错了。写PPT、讲PPT、讨论PPT，这个流程全部走一遍，基本上对一个知识点掌握就比较全面了。

总结

成为技术大牛梦想虽然很美好，但是要付出很多，不管是Do more还是Do better还是Do exercise，都需要花费时间和精力，这个过程中可能很苦逼，也可能很枯燥，这里我想特别强调一下：前面我讲的都是一些方法论的东西，但真正起决定作用的，其实还是我们对技术的热情和兴趣！