如何面对天天要写的业务代码和梦想成为技术大牛两者间的矛盾与纠结

为了讨论标题表达的这个问题，我们可以探讨如下几个话题。

1. 什么是业务
2. 业务和技术的关系
3. 业务和因解决业务而衍生的业务
4. 对业务的态度因你在团队中的角色而不同
5. 如何从写业务代码中跳出来，做你所谓的有技术含量的工作

1. 什么是业务

简单讲，“业务”就是需要处理的各种事务，通常指每个公司的产品和服务需要涉及的事务，“业务”最终的目的是完成工作所做的所有事务。

比如取款就是一种业务，ATM 机内运转的软件，要解决的业务就是取款。
比如挂号、预约、查检查报告，都是业务，趣医网的 App 就可以用来解决这些业务。
比如买火车票也是业务，12306 这个网站就是为解决买车票的业务服务的。

2. 业务和技术的关系

软件是用来解决现实世界中的业务给人们的工作带来便利的。

比如到火车站买票，要坐车、提前、排队，又麻烦又消耗时间又浪费精力，而 12306 网站和 App ，通过把买火车票这种现实业务虚拟化，为人们省去了奔波、排队、耗时的麻烦。

比如大家都想到好医院看病，人人都想挂专家号，很多人为了挂到某个医生的号，通宵排队，非常辛苦，而现在的各种网上挂号网站、微信公众号、 App ，通过软件技术手段，把专家大夫这种资源虚拟化，让大家随时随地能挂号，还不用到医院、不用通宵憋尿排队、不用担心被医托和黄牛忽悠，给患者带来了极大便利。

软件是现实业务虚拟化的载体，技术最终是为了解决业务问题的。从这个角度讲，所有的开发者，其工作最终都是指向某个特定业务问题的。没有业务，技术的存在就没有意义。技术不能解决实际问题，不能给人们带来便利，就没有价值。

但从另一方面来讲，技术是现实业务虚拟化的必要条件，没有技术，现实中的业务就无法被虚拟化。而且，同一种技术又可以实现多种业务的虚拟化。所以，很多初阶的开发者才会有种“错觉”：技术牛 X ，因为没有技术就无法实现业务，业务很 Low ，技术牛 X 了，随随便便就能搞定。

实际上，这些感觉虽然在一定阶段有其道理，但并不是真理哦。

关于业务和技术的关系，这里下个结论：

技术是为了解决业务问题的，只有在实现业务、给人们带来便利的前提下，技术的存在才有意义，所以，多数时候，是业务决定技术、业务统领技术。
没有技术，业务就无法被虚拟化，生产效率就很难有效提升。
业务和技术具有相互促进、相互依存的关系。

我们回到开发者身上来看，写业务代码多一些，还是所谓的技术代码多一些，没有高下之分，只有个人取向和组织分工的不同。

3. 业务和因解决业务而衍生的业务

很多开发者会用割裂的眼光来看待业务和技术，比如把增删改查（CRUD）看作是无意义的业务代码，把实现 libuv 或 Redis 这样的框架看作是有技术含量的事情。

比如京东上《程序员的成长课》这本书的详情页，是这样的：

它对应的架构是这样的：

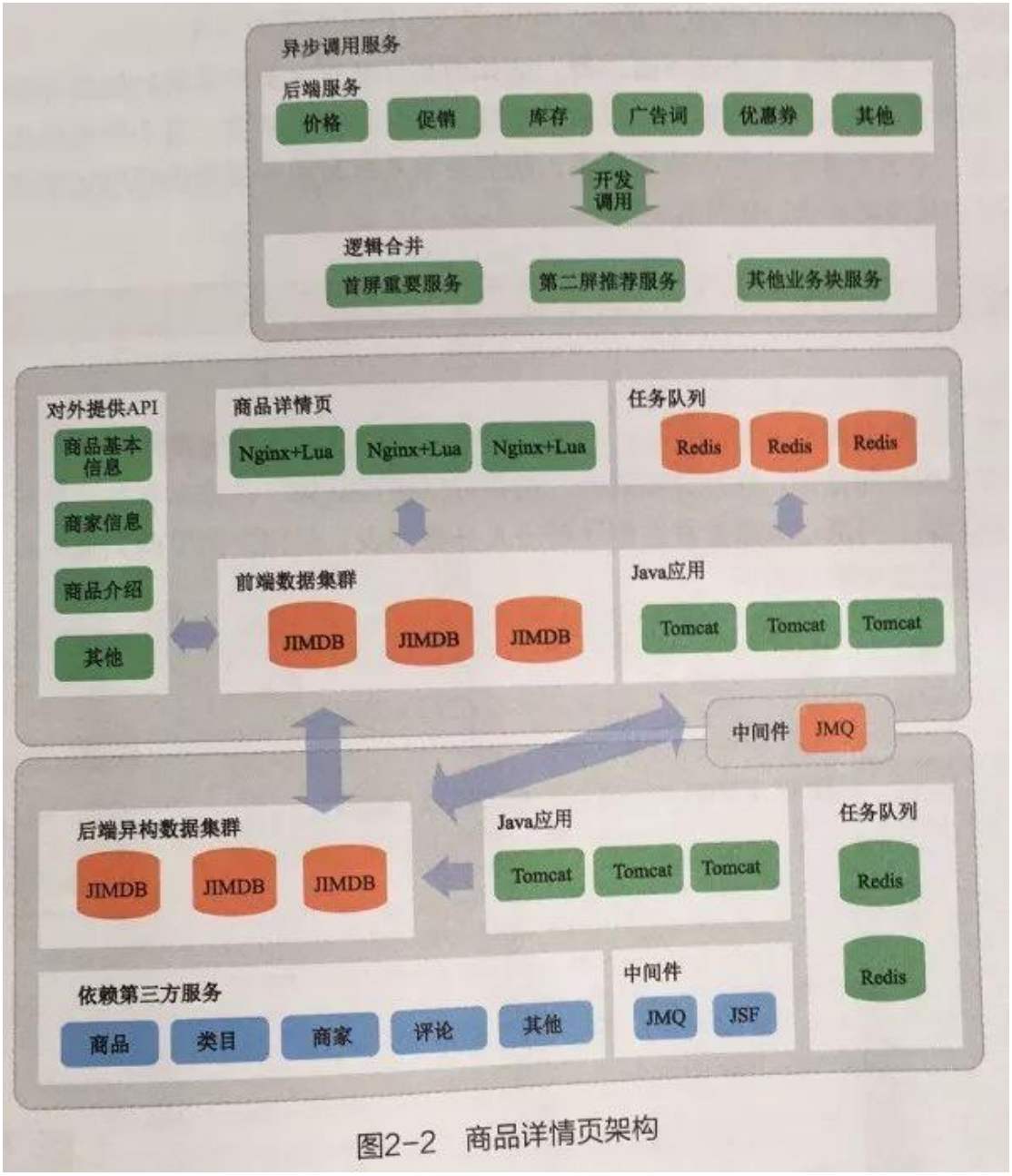

图2-2 商品详情页架构

很多开发者会觉得，写那些用来展示《程序员的成长课》的图书封面、优惠券、促销等相关信息的代码是没什么技术含量的，因为那些是业务代码。

他们会觉得，写商品详情页架构中的 Redis、JMQ 或 JIMDB 是有技术含量的，是真正的技术代码。

但实际上，所谓的业务代码和技术代码，它们的区别，仅仅是和业务的距离远近不同而已：业务代码离业务更近，技术代码离业务稍远。它们最终都是指向业务实现的。

而且，你换一种视角来看业务，就会发现，其实每一层代码，都服务于它的上一层代码，上一层代码，就是它的业务！

比如详情页架构的第 2 层“对外提供 API”中的商品介绍个 API ，它的服务对象，就是前端页面，要解决的业务，就是“响应前端页面的查询，提供商品介绍”

而第 2 层底部的前端数据集群（JIMDB），它的服务对象，就是商品介绍，要解决的业务，就是“存储商品或代理商品介绍信息”。

简单说，每一层技术实现，都服务于上一层，都以上一层的需求为业务。从这个角度讲，现实中的业务在被虚拟化的过程中，会在技术实现层面引发分层，产生中间性、对用户不可见的新业务。

从这个广义业务的视角来看，每一层代码，都是业务代码！
但是为什么很多开发者又觉得所做的技术实现越接近现实业务越没技术含量呢？

这是因为，你越接近用户业务：

细节越多，繁琐度越高，越不容易做好，越容易因为一点小瑕疵而被否定，让人觉得自己的劳动没价值
现实性越强，变化几率越高，越容易来回修改代码，越让人觉得自己的掌控感低下
实现的代码可迁移性越差，劳动成果被复用的概率越低

而当你远离用户业务时：

你用到的技术，多数都是被高度抽象过的、用来解决从用户业务衍生出的技术性业务的，它们比具体的用户业务稳定，它们的适用面更广，也更容易被迁移到其它的业务领域
你的劳动成果因为具有抽象属性，被复用的概率会更高，你会更愿意打磨它，会更有成就感
你受到压力，经过距离用户近的几层同事的传递，得到了衰减，没那么大
你打交道的对象，多数时候是内部同事、是技术人群，更容易达成一致

4. 对业务的态度因你在团队中的角色而不同

你对业务的态度，会因你在团队中承担的角色不同而不同。这是由开发团队的组织结构和职责分工导致的。

下面是我绘制的“团队结构、能力与职责”图：

在一个开发团队中，架构师这个角色，会负责业务拆分和软件架构的工作，并且领导团队来实现满足业务的软件。

注 1 ：有的研发团队里有业务架构师和软件架构师两种角色，业务拆分由业务架构师或业务分析师完成。
注 2 ：软件架构师和业务架构师这两个角色也可能由没有架构师头衔的研发经理兼任。

架构师一定是要以业务为导向的，要搞懂业务的。所以，在架构师这个阶段，在团队管理者这个阶段，业务的重要性，往往是高于技术的，在他们的眼中，业务统领技术，技术是用来实现业务的。

当团队完成业务架构和软件架构之后，就会选择不同的开发者来负责不同功能模块的实现。

负责不同功能模块实现的开发者，必须能够理解业务，并且要熟悉某个技术栈，能够进行模块设计和任务拆分，我称这样的开发者为“熟练开发者”。

熟练开发者会承接由架构师分派的子业务，负责模块设计和拆分，把拆分后的小任务，交给普通程序员来完成。

当你是一个熟练开发者时，业务和技术几乎同等重要，因为：

你不理解业务，就很难将子业务模块映射到软件实现上，也很难做进一步的业务拆分。
你不具备完整的技术栈和相应的知识体系，就很难找到合适的技术来实现业务，也很难做软件模块的拆分。
熟练开发者完成了子业务和软件模块的拆分，会形成一系列的叶子型任务，并把它们分派给具备特定专项技术能力的普通程序员。

普通程序员要做的事情比较简单，就是接受别人分派的任务，实现特定的业务细节。

注意当你是一个普通程序员的时候，团队要求你具备一定的专项技术能力，能够完成任务即可，你的角色，就拿把螺丝刀拧螺丝，拧好螺丝就 Ok。

这个时候，你内心是痛苦的，对不停地写业务代码是拒绝的，因为你要再找工作时，别的组织看重你的专项技术能力甚于业务能力（他们有人做业务拆分，你过去了能拧螺丝即可），而你在现有组织中，却因为深陷业务代码的编写而无法持续淬炼你的技能能力。

所以普通程序员最纠结写业务代码这件事！

那么，该如何才能解脱呢？

5. 如何从写业务代码中跳出来

孔子说过一段话：“弟子入则孝，出则悌，谨而信，泛爱众而亲仁，行有余力，则以学文。”

翻译成现代文，是这个意思：“年轻人，在家就要孝顺父母，出门在外就要尊敬兄长，行为谨慎，言语有信，博爱众人，亲近仁者。这样都做到之后还有余力的话，就可以去学习从政，做更大的事业。”

这段话呢，给普通程序员指明了方向：轻松搞定你的业务代码，还有余力，就可以做更重要的事情。

也就是说，当下你能力不够，组织上不可能给你更复杂的模块让你负责（再说团队里已经有更厉害的人在做那些事了），你得先轻松且漂亮地搞定手上的任务再说。
很多普通程序员天天抱怨老写业务代码没长进，可手上的任务却总是敷衍了事，完成得凑凑合合，那是很难摆重复简单业务任务的泥沼的。

那怎样才能做到轻松、漂亮地搞定任务呢？ 4 点：

1) 在深度和广度两个方面提升技术能力（如果当下任务繁重，就利用业余时间练习）
2) 把自己的做的事情放在全局理解，提升业务理解能力
3) 培养好的工作习惯，比如计划、回顾等
4) 做好汇报和展示，让领导知道你的能力

当你慢慢做了上面 4 点之后，每次拿到任务，都能轻松又漂亮地搞定，超出领导的预期，还有未发挥完的火力，那团队就一定会给你复杂一点的任务，如果你还能轻松、漂亮地搞定并且还有余力，那团队就会给你复杂度再高一些的任务……

往复循环，你就可以跳出最简单的业务代码编写，做越来越重要的事情，人也变得越来越重要。

6. 小结

前面我们分 5 个部分阐述了业务和技术的关系，总结一下，关键的其实有 3 点：
技术是手段，业务是目的；软件开发工作是以业务为导向的，但是没有技术又无法实现业务。
业务和技术的关系，随着开发者角色的变化而变化。
刚入行时作为普通程序员，技术是基础，有技术才能实现业务，公司在招人时也以技术水平为门槛，从这点出发，一定要在短期内迅速提升技术。

工作了 3 、5 年，成了熟练开发者，可以独自负责一个业务模块时，需要更好地理解业务，这样才能更好的从技术上实现，此时业务和技术并重。

从熟练开发者往前发展，有两条路，技术专家和架构师，如果你选择架构师的路线，则应该调整思维，以业务为导向，把业务放在更重要的位置，因为架构是从业务拆分出来的，如果你选择技术专家路线，则需要在深耕技术的同时保持对业务的敏感。

普通程序员要想从业务代码的泥沼中跳出，要从技术水平、做事的方法、习惯和自我展示几方面入手，努力做到搞定任务有余力，进入正向循环，慢慢获得做重要事情的机会，让自己变得重要。