加餐四 | 学习基础技术：你对“基础”的理解准确吗？

2021-01-15 李运华

大厂晋升指南 进入课程 >

讲述：安晓辉

时长 16:58 大小 15.55M

你好，我是华仔。

如果说 IT 技术领域有哪个说法最深入人心，那一定是“基础很重要”；而如果说有哪个说法让很多人花费了大量时间去学习，却没什么效果的话，那么多半也是这句话。

我相信你曾经被人谆谆教诲过：做技术，基础很重要，一定要打好基础，比如说数据结构和算法、操作系统、编译原理等等；而且很多公司面试的时候，也采用了“面试造航母，工作拧螺丝”的方式，对基础能力的考察远远超过实际工作需要。

结果，很多人费了很大的力气来提升所谓的“基础能力”，但是却发现根本看不到提升效果，工作中也用不上，白白浪费时间和精力。

难道说“基础很重要”这个说法不对吗？其实这个说法本身没有问题，但是它模糊太笼统了，很难准确地理解，再加上一些口口相传的经验误导，搞得很多人都掉到坑里去了。

这一讲，我来跟你聊聊到底什么才是“基础”，怎么提升基础技术才能事半功倍。

典型的错误观点

基础能力确实很重要，但是对于什么才是“基础”，业界并没有统一的定义。不过，有几个错误的观点流传很广，误导了很多人，其中最典型的就是以下三个：

1. 基础 = 底层

有些人以为越底层的东西越基础，比如操作系统内核（管控程序的运行），编译原理（所有编程语言的基础），CPU 的指令和内存（程序运行的基础）……毕竟从字面意思来理解，底层的东西当然是基础了，而且是越底层越重要，因为越底层越通用。

2. 基础 = 源码

有些人喜欢“Show me the code”，认为只有源码才是最基础的东西，源码面前没秘密，要学基础就一定要去看源码，要自己能写出来才算真正掌握了“基础能力”。

3. 基础 = 不变

有些人认为不变的东西才是基础，比如数学、算法和数据结构、计算机组成原理、汇编语言、甚至包括离散数学和逻辑电路等，把这些学好了，以后无论做什么都用得上。

很多人抱着这样的想法去提升基础，结果却没什么效果。

我有个同事花了 6 个月时间去研究编译原理，感觉没什么收获，然后找我来讨论原因。

我也有位朋友花了大量的时间来看 Linux 内核源码，看完好像知道了一些源码，但线上出了问题之后，连 Linux 定位工具都不会用。

也有很多技术人员用了很多时间来背算法和数据结构的源码，但在实际工作中，要么不知道什么时候用什么算法，要么就滥用算法，明明一个很简单的逻辑也要硬套一个算法。

核心就是工作相关

要想打好基础能力，首先要明确什么才是真正的“基础能力”。

我的观点是“**基础能力是指工作任务相关的基础能力，不是整个计算机技术的基础能力**”，核心就是“**工作相关**”，千万不要单纯照搬别人口中的基础能力。

基于这个观点，我们来澄清一下前面提到的几个错误观点。

1. 基础！= 底层

如果底层技术和当前的工作内容没有关系，那就不是工作要求的基础能力。

比如 CPU 指令和内存寻址，对于做嵌入式开发来说是基础，而对于做 Android/iOS 业务开发来说就不是基础了。

2. 基础！= 源码

如果当前的工作并不需要我们去修改其源码或者理解其源码细节，那就不是工作要求的基础能力。

比如 Linux 内核源码和 Hotspot 虚拟机源码，对于做虚拟机开发来说肯定是基础，但是对于 Java 业务开发来说就不是基础了。

3. 基础！= 不变

不变的东西确实应用很广，但是随着技术的发展，不变的东西越来越稳定，封装也越来越抽象，基本上就可以认为不再需要关注它了。

这就像电一样，我们天天用，电学的原理，你只要上过中学基本都知道。但是我相信，现在没有人在使用电器的时候，还要去翻一翻物理课本吧。

很多人为了证明“基础很重要”，都会举建房子的例子，因为地基是房子的基础。

但其实认真思考一下，就算是建房子，打地基的方式也是不断变化的，古人用夯土打地基，后来用石头打地基，现在用钢筋水泥打地基，而且工人在用钢筋水泥打地基的时候，也不需要知道“如何制造水泥”“如何炼钢”这样的基础知识。

麻省理工大学以前有一门非常火的课程，叫“计算机程序的构造和解释”（SICP，Structure and Interpretation of Computer Programs），但后来他们停止了这门课，给出的原因如下（Why MIT stopped teaching SICP）：

> They felt that the SICP curriculum no longer prepared engineers for what engineering is like today.
> Sussman said that in the 80s and 90s, engineers built complex systems by combining simple and well-understood parts. The goal of SICP was to provide the abstraction language for reasoning about such systems.
> Today, this is no longer the case. Sussman pointed out that engineers now routinely write code for complicated hardware that they don’ t fully understand (and often can’ t understand because of trade secrecy.)
> The same is true at the software level, since programming environments consist of gigantic libraries with enormous functionality.

简单来说，就是 SICP 课程不是为今天的程序员准备的，而是为 20 世纪 80～90 年代的程序员准备的。

这是因为那个时候的程序员是通过组合简单和深刻理解的部件（其实就是指从底层开始构建）来构建复杂系统，而现在的程序员在复杂的商业硬件和大型的开发库上面来构建复杂系统，就算程序员想了解这些底层硬件和开发库，也可能因为商业秘密等原因无法做到。

举例说明

按照工作相关这个原则，我举两个常见的例子对比说明一下。

1. Java 业务开发 vs AJDK 开发

它们都是 Java 相关的开发，我们假设一个是用 Java 来在 Linux 平台上基于 Spring Boot 框架完成业务开发，另一个是要负责阿里的 AJDK（基于 Hotspot 实现，目前已经开源）开发，那么它们的基础能力差异如下表所示：

Java业务开发	AJDK开发
Java语言特性和API是基础 C++不是基础	Java语言特性和API是基础 C++也是基础，因为Hotspot是用C++开发的
Java虚拟机垃圾回收原理是基础 Hotspot虚拟机源码不是基础	Java虚拟机垃圾回收原理和实现是基础 Hotspot虚拟机源码也是基础
Linux环境的工具和进程管理是基础 Linux内核不是基础	Linux系统编程是基础，包括进程/线程和锁 Linux内核实现机制和原理也是基础
数据库是基础，包括SQL和索引等	数据库不是基础，因为不相关
Spring Boot框架的特性、原理是基础 Spring Boot的源码不是基础	Spring Boot不是基础，因为不相关

2. Android 业务开发 vs Android 动态化框架

它们都是 Android 相关的开发，我们假设一个是做业务开发，一个是开发动态化框架，那么它们的基础能力差异如下表所示：

Android业务开发	Android动态化框架
Java语言特性和API是基础 编译原理不是基础，因为不会去修改Java语言	Java语言特性、API是基础 编译原理也是基础，因为动态框架可能要用DSL
Android的基本原理是基础 源码不是基础	Android的原理和源码都是基础，比如用反射来修改Android的一些代码从而实现动态化
常见的库是基础，比如网络库OkHttp、图片库Fresco和事件库EventBus等	可能会用到网络库OkHttp 但总体上对各种库的要求不多也不高
JavaScript不是关键基础 主要应用是JSBridge	JavaScript是基础，比如React Native和Weex都是基于JavaScript的跨平台动态化框架

细化基础范围：技能图谱

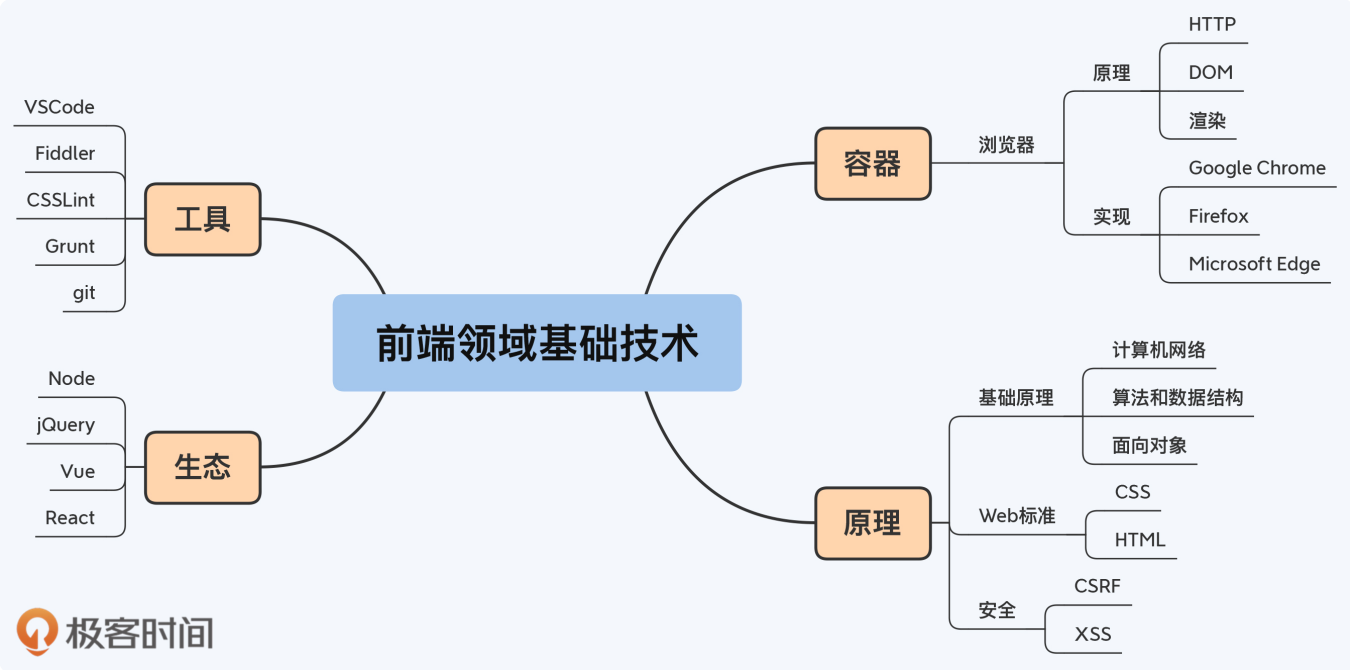
明确了“工作相关”这个原则之后，提升基础的第一步，就是使用**技能图谱**的方式，从以下 4 个维度来细化基础能力的范围。

1. **工具**：工作中常用的工具，比如 IDE、编程语言、问题定位工具和版本管理工具等。
2. **生态**：系统或者产品运行时依赖的所有组件或者系统，比如第三方库、中间件、数据库、文件系统和游戏引擎等。
3. **容器**：系统或者产品在哪里运行，比如 Android、iOS、Linux、浏览器和云服务器等。

4. **原理**：需要掌握的原理知识，常见的有计算机网络会让数据结构等。

工具	工作中常用的工具
生态	系统或者产品运行时依赖的所有组件或者系统
容器	系统或者产品运行的环境
原理	需要掌握的原理知识

我们以前端开发为例，基础能力的范围如下图所示：



注：

1. 上图仅为示例，不代表完整的前端领域基础能力范围。
2. 图中没有涵盖“JavaScript 编程语言”，因为这个可以说是核心能力，不是基础能力。

有了技能图谱之后，我们就能够大致地了解每个技术领域的基础能力到底包括哪些了。

提升技术的技巧

明确了基础能力的定义和范围，我们就可以把有限的时间和精力用在更有价值的地方，避免眉毛胡子一把抓，从而实现投入产出效率的最大化。

但是，单个基础技术怎么学，这也很关键，否则学习还是可能事倍功半。

常见学习误区

常见的学习误区有两个。第一个是认为，**既然是基础技术，那肯定是掌握得越深越好**，比如数据结构和算法、计算机网络和操作系统这些，几乎是所有程序员的基础，所以每一项都应该深入了解。

但是这样做还是会导致你浪费很多时间和精力。

以数据结构和算法为例，很多人学习的时候都采用了背代码的方式，认为只有自己能手写这些代码，才算是真正的掌握。

而且，有些面试官在面试的时候喜欢让应聘者写简单的算法代码，进一步强化了这样的认知。

我就曾经跟几个这样的面试官聊过，对话过程几乎一模一样，很有意思：

问：“为什么你们要用这种方式来判断应聘者的水平？”

答：“如果一个程序员连个简单的算法都写不出，那就说明他肯定不合格！”

问：“那你们要自己修改算法和写数据结构吗？”

答：“怎么可能？直接用 Java 库里面的，自己写的质量怎么跟 Java 库的比啊？”

问：“一般你们考什么算法和数据结构？”

答：“冒泡啊、快排啊、链表、字符串之类的？”

问：“那为什么你们不要求手写 B+ 树，不手写 ConcurrentHashMap 这些呢？这些难度更高。”

答：这个要求有点高啊，现场写不完，其实我自己也写不出……

在上面的对话中，我们可以看到，其实这种面试方式就属于“面试造航母，工作拧螺丝”，不能判断应聘者水平高低，只能反映出他们面试准备程度的高低；而且能考的也就是这几个简单的数据结构和算法题目，因为面试官自己也只能看懂这几个。

第二个误区是认为，**要完全掌握基础，一定要掌握源码**。

这个观点更加容易导致你投入大量时间却没什么收获。尤其是 Linux 内核源码、JVM 虚拟机源码和 MySQL 源码这些，如果你不具备深厚的 C/C++ 的开发功力，基本上连看都看不懂，更不用说考虑代码规模和复杂度了。

即便是 Netty 这些代码相对少一些的开源项目，就算你拥有很强的 Java 开发技术，要想每一行代码都了解，也要花非常多的时间的。

因为一个成熟的开源项目，都是几十个人用了很多年的时间慢慢积累的，你一个人想一下子就全部搞懂所有代码，这是不现实的。而如果你把时间浪费在这个地方，用来提升其他更有用的技术的时间就没有了。

如何判断学习深度？

所以说，就算是同一个基础技术，不同的技术人员学习的深度也是不同的。核心的原则还是之前提到的“工作相关”，根据工作内容来决定基础技术的学习深度。

下面，我举几个常见的例子来说明。

1. 数据结构和算法

对于绝大部分开发人员来说，主要是熟悉数据结构和算法的原理、优缺点与应用场景，还有自己所用的编程语言提供的算法和数据结构。

而对于中间件开发的技术人员来说，在做极致的性能优化的时候，Java 的 ConcurrentHashMap 之类的并发数据结构，就需要掌握算法的原理和代码实现细节了。

2. 计算机网络

对于绝大部分开发人员来说，能够熟练掌握抓包工具抓取 TCP/IP 包，并且能够看懂包信息，定位网络问题就行了。

而对于运维人员来说，抓包、路由协议、组网配置等就需要深入掌握了。

3. 操作系统

对于绝大部分开发人员来说，掌握基本的操作系统原理和概念，能够使用操作系统提供的工具来定位程序问题就行了。

而对于驱动开发、内核模块开发的技术人员来说，操作系统原理、实现机制和代码都需要深入掌握。

如何让理解更加深入？

明确学习深度之后，因为基础知识点比较多，看起来比较散，所以你可能学了很多知识，但是不知道它们之间的关联关系，理解不够全面和深入。

应对这个问题办法就是第 19 讲中介绍的“链式学习法”，通过领域分层将基础技术和顶层的实用技术关联起来，形成系统化的理解，这样能够理解得更深，记得更牢固。

基础积累会不会浪费？

看到这里，你可能会有疑问：判断基础能力范围和基础技术学习深度的原则都是“工作相关”，那么如果工作发生变化，岂不是很多基础技术的积累都白费了？

这里就要看所谓的“变”具体是怎么变。

如果是前后两个工作的领域基本一致，那么基础技术的积累基本上是可以通用的。比如我曾经从 PHP 服务端开发转为 Java 服务端开发，在数据结构和算法、计算机网络、数据库和操作系统方面的积累完全可以通用。

但如果前后两个工作领域差异很大，那么基础技术的积累确实可能无法通用。比如我的一位同事从 Android 开发转为服务端后台开发，虽然数据结构和算法、计算机网络可以通用，但是 SQLite 数据库和 Android 操作系统这些就不能通用了。

所以跨领域转岗一定要慎重，要转的话就尽早转，越晚损失越大。

我的实际经历

我刚去 UC 的时候，是用 C/C++ 做中间件，对高性能和网络都有比较高的要求。于是我深入地学习了 CPU 和网络的一些基础知识，最典型的就是 SMP 架构的 CPU 的 False Sharing（伪共享）问题。

这个知识点理论上属于计算机组成原理，但是计算机组成原理一般只会写 CPU 有 L1/L2/L3 Cache，很少提到多核 CPU 的的 Cache Line（缓存行）对齐会导致 False Sharing 问题，并且对性能有很大影响。MySQL 也被这个问题给坑过，而 Disruptor 的高性能则是采用 padding 避免了这个坑。

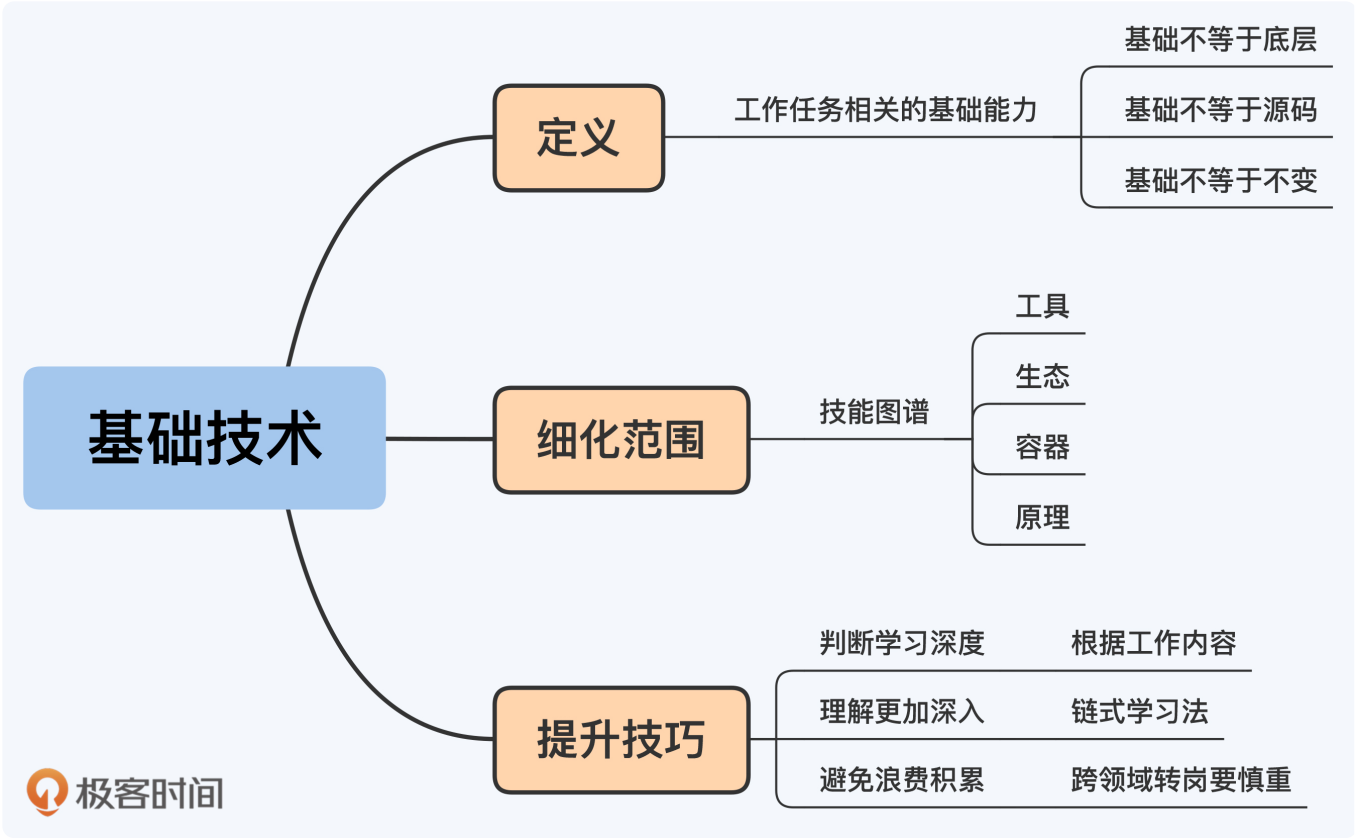
你看到这里，可能会急着想去看看我说的 False Sharing 到底是什么。别急，这个基础知识点在我后来负责业务开发的时候就没用了，因为接触不到这个深度。

但是做业务开发的时候，MySQL 的索引原理和 Elasticsearch 的倒排索引这些基础理论就很有用了，因为你要设计合理的索引和存储方案。不过这个时候，你也不需要把 B+ tree 的数据结构写出来，只需要知道原理，就可以设计合理的索引和存储方案了。

小结

现在，我们回顾一下这一讲的重点。

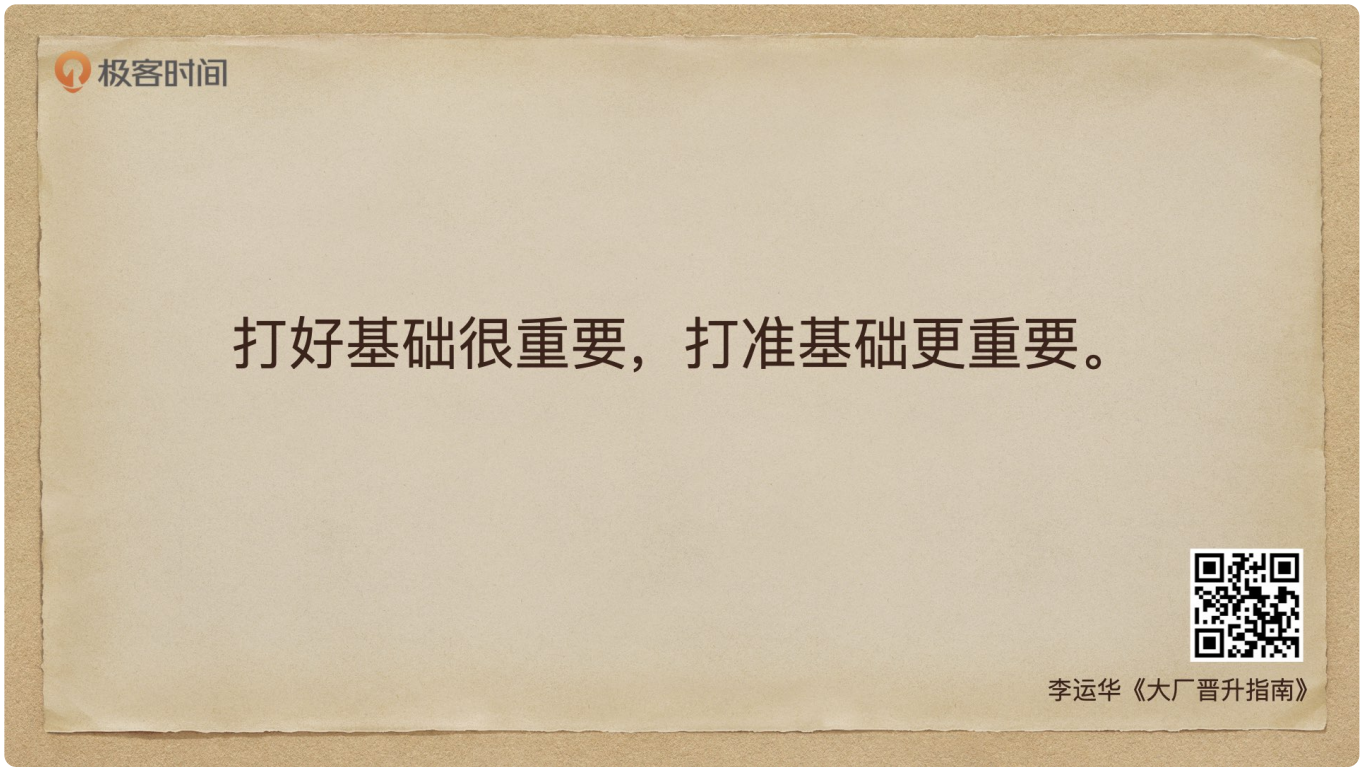
1. 基础能力是指工作任务相关的基础能力，不是整个计算机技术的基础能力。基础不等于底层，不等于源码，也不等于不变。
2. 提升基础的第一步，就是使用技能图谱的方式，从工具、生态、容器和原理这 4 个维度细化基础能力的范围。
3. 提升基础技术的技巧包括：根据工作内容来决定基础技术的学习深度；通过链式学习法将基础技术和实际用到的技术系统串起来；跨领域转岗要慎重，要转的话就尽早转。



思考题

这就是今天的全部内容，留一道课后思考题给你吧。按照这一讲的内容，你能够整理出你当前工作岗位要求基础技术包括哪些，以及你需要学到怎样的深度吗？

欢迎你把答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。



提建议

上一篇 加餐三 | 10000小时定律：成为大牛的秘密是什么?

下一篇 放学别走 | 如何画好领域分层图?

精选留言 (9)

写留言

humor
2021-01-15

虽然道理是这样，但是如果没有学过某项基础技术，比如操作系统和网络，我怎么知道我学到什么深度才算合适呢？或者我后来的工作遇到了难题，要解决这个难题需要用到比较深的基础技术，但是我又没去学这么深，我怎么能想到要去学这项技术，然后应用呢？

展开

作者回复: 第一个问题：
如果你完全都没学过某个技术，连它包含什么内容，涉及哪些概念，有什么应用场景都完全不清楚，是不可能看一眼就知道应该学到什么深度的。

要想自己不学就知道其深度，只有让别人来告诉你，即使别人来告诉你，你也只是知道了结果，不知道为什么有的是要深度学习，有的是要大概了解。

所以说学习不是一次性的，而是要逐步迭代深入的。以计算机网络为例，第一次学习你至少要系统的找本书或者资料系统学习一下，你就能够知道其内容涵盖DNS、TCP、IP、BGP、ARP、RARP、RIP、OSPF……这一大串技术，此时你已经知道每个技术点的大概含义和适应范围了，你再结合自己的工作来判断哪些是大概了解，那些是要深入学习的。

例如，你是做Java后台业务开发的，那DNS、TCP、IP是需要深入了解的，但是ARP、OSPF、BGP就不需要深入了解；但如果你是运维，这些基本都要深入理解。

第二个问题：
要么别人告诉你，要么自己带着问题去探索，然后找到对应的技术点，再深入学习。
例如，false sharing这个案例，一开始我也不知道，我是去查"multi-thread high performance"这个关键字，然后无意中发现的，发现后我就深入的去学习了一下，又了解了MESI等关联知识。

4

黄立
2021-01-15

毕竟不是做科研，还是以应用为主，学到用到最好

展开

作者回复: 可以这么说，毕竟时间和精力有限，要考虑投入产出比

1

humor
2021-01-15

基础技术就算暂时用不到，学习深入一点也会有好处吧，比如可以学习前人的设计思想，将优秀的设计思想引入自己的业务中，或者为以后应用的机会做准备😁

展开

作者回复: 都可以引入自己的业务了，为啥还说用不到呢?

1

奇小易
2021-01-19

对于基础这个还是有点懵，我读完后理解的基础技术是工作上用到的或者有直接联系的技术。那是不是工作中所有用到的技术都是基础技术？老师说的核心技术是怎么区分？是这个岗位必须牢牢掌握的基础技术？
请老师解惑。

展开

作者回复: 你再仔细研究文中的案例，就是前端那个图。

Y丶z
2021-01-18

说的没错，但是其中有个度很难衡量，就是现在工作中用不到，但是后续突然用到了，如果当时不会，可能就会错失一些机会。

展开

作者回复: 先把常用的，流行的，成熟的学会了，基本不用担心突然要用，因为可以说每个技术都有多个选择，即使突然要用，你积累多了，学起来也很快

1

威龙
2021-01-17

“打好基础很重要，打准基础更重要”总结的非常到位，虽然自己不是做技术的同学，但是从这篇加餐中get到学以致用的思想。结合日常做产品工作来思考，就是要找准产品和项目的核心价值，找准业务方核心需求，保持Less is more的设计思想。

展开

作者回复: 很有见地：）

主要是我们的时间和精力有限，如果什么都想学很深，是不现实的，而且投入产出比低。

qinsi
2021-01-16

那么问题来了，面试或晋升的时候，如果所有候选人在岗位划定的基础能力范围内水平都是一样的，那要如何决定谁被录取或晋升呢？

展开

作者回复: 这个没有标准考试，不会出现两个人的分数完全一样的情况，实际评价的时候，不可能认为两个人水平完全一模一样，一般都是“两个差不多”这种评价，然后如果一定要排名，一定能找到理由区分高低的。

1

我来也
2021-01-15

老师真敢说😁
上次只说了编译原理，这次说了一大堆。

作者回复: 你觉得说的有道理不？😂

1

术子米德
2021-01-15

🤔☕🤔☕🤔
基础:=你当前使用技术的支撑技术，理解当前层的支撑层，这样的技术够基础、够实用
基础:=你当前思维方式的思维框架，理解你的思维哪里来，受约束哪些思维方式，这样去思考，就像万维钢老师说的Something bigger than yourself，这样的思维够基础、够实用

展开

作者回复: 有点抽象，我看不太懂😂