1. 现状及瓶颈

1.1 瓶颈

有没有觉得过时间过得真快？有没有觉得这一年都没做过什么就过去了？有没有在和manager聊薪酬的时候发现自己没有什么调薪理由可以说？有没有觉得在公司工作没有成长？

这就是瓶颈。

1.2 思考

有没有思考过应该如何走出这样的职业瓶颈?

1.3 解决瓶颈

1.3.1 能力

你必须了解行业里各个level的人都拥有哪些能力。没有知识没有眼界，你永远也不知道你缺的是什么。

1.3.2 目标

设定当前对你来说可实现的目标。短期、可执行，这是最关键的，否则会打击积极性和可持续的意志。

1.3.3 执行

坚持执行，达到目标。然后重新设定目标并迭代。

2. 后端程序员需要的能力

2.1 技术能力

- PHP:
    - 基础：值传递、引用传递，内存消耗、释放，内存拷贝
    - 性能调优：软件层面，代码层面
    - profile工具使用，代码执行细节分析
    - 常用debug手段
- 数据库:
    - MySQL:
        - 引擎差别，对应的应用场景
        - 索引使用，索引算法
        - explain使用
        - 常用数据库高负载解决方法
    - Redis:
        - 不同数据结构内存消耗效率
        - 不同接口算法复杂度

        - 内存分配策略，优化方法
        - 内存消耗完毕之后的行为，如何处理
- JavaScript：
    - js里的OO怎么处理，有哪几种方法，优劣各是什么
    - js里如何进行类继承
    - js里如何进行类型判断
    - js里的全局变量陷阱
    - js里的闭包，有什么陷阱，如何避免
- 缓存：
    - memcached内存分配策略
    - memcached如何计算内存浪费
    - memcached如何减少内存浪费率
    - memcached如何观测slab分布
- CSS：
    - 什么是LESS和SASS
    - 优劣？如何选择？
    - 比起原生CSS，它们解决了什么问题？
- HTML5：...
- Linux：
    - vim快捷键
    - 简单编程环境搭建
    - bash是什么
    - sudo是什么
    - awk是什么
    - sed是什么
    - 简单的系统监控命令

2.2 对程序通用特性的敏感度

掌握一门新的语言其实很简单，计算机资源来来去去就是CPU、内存、硬盘、网络等几项。对应到编程语言内的资源，其实也就是那么几项。

在学新语言、新技术的时候，不在于你学没学会语法，这种东西查查手册分分钟就解决了。关键在于你对于一系列每种程序都通用的特性掌握得如何，一般来说有这几点：

- 该技术擅长什么不擅长什么
- CPU密集运算优化
- 内存管理策略
- debug工具使用
- profile工具使用
- 并发解决策略

理解了上面几点内容，你就不会胡乱说自己掌握了一门新技术了。至少对于自己没掌握的部分，能有一个大概的了解。

2.3 DRY

DRY：Don't Repeat Yourself

程序员都得有这个素养，不懒的程序员不是好程序员。每次发布太复杂？做工具解决。每次配置表更新都出错？做工具解决。

很多团队工作几年之后都能开源出不少好用的工具，这就是技术积累，就是财富。

2.4 解决问题的能力

带项目的永远不会欢迎只会问“十万个为什么”的程序员。简单的问题，就要有能力自主解决。在不得不向上级询问的时候，带上解决方案会比直接去问要好上一万倍。

请自行想象：

- “xxx，我有一个问题解决不了了，....” 结束
- “xxx，我有一个问题，现在我找到两种方法解决，第一种是...，第二种是...，我觉得第一种有...优势，...劣势；第二种有...优势，...劣势。你觉得哪种会比较好点，我倾向于...” 结束

2.5 学习能力

技术的世界千变万化，实在太精彩。身在其中的程序员就不得不面对这快速演变的技术世界。学习能力强弱是至关重要的分水岭。能快速学习新技术的，就能活下去，否则就是被淘汰。

2.6 英语

接上一点，太多新的技术在进入国内铺开之前，都只有国外的英语资料和文档，英语水平不好，就只能干瞪眼了。当一门技术热门到国内遍地翻译文档和技术书的时候，你再入手也就不吃香了。

2.7 管理能力

管理无处不在，不是有title了才算是管理，只要项目组你主要负责，而且同职能的不止一人，那么你就是管理。简单讲几点可能平时大家没想到的：

- 有没有写文档，别人写的文档有没有review
- 写代码之前，项目组有没有过过程序设计案
- 有没有定期的code review，有没有强制的code review
- 有没有定期的项目组内人员工作report
- 有没有定期了解项目组内人员工作情况，1：1谈话
- 有没有尝试解决项目组内的技术债
- 有没有尝试将项目内的技术模块抽离，并发展开来

思路要打开，甩脱自己只是一个写代码的人这样的认识。

2.8 基本素养

- 文档先行
- coding style

- do things right at the first time
- 拒绝拖延症
- 设计完成了再写代码

2.9 掌握最新资讯并选择正确的技术方向

技术发展太快，新的语言和新的技术出现也太快。不了解这些新的技术，不了解这些新技术擅长什么不擅长什么，你就很难在以后技术投资的时候选择正确的方向。

对一种有潜力的技术进行技术储备和投资，才是关乎到将来薪酬水平的重要砝码。最好的例子就是object-c，诡异的语法，但又如何。市场供需关系决定了object-c的程序员大量需求，却又大量缺少。薪资水平当然水涨船高。

每天有固定关注什么技术新闻站点？有关注业界大牛博客？如何了解最新的技术资讯？

3. 如何培养能力提升自己

3.1 做好本职工作

看了我上面对后端程序员能力的描述，你应该了解自己仍旧还有多少东西有欠缺。即便做的是最初级的工作，要把这份工作真的做好，做完美，还是很有难度的。不要因为自己做的东西简单，就觉得很容易做好。其实并非如此。

我第一份工作，做web外包的时候，在项目组中，我能完全把从设计到实现的整个流程一个人完成掉，我的上司啥都不用干，只需要给我做review就好了。长此以往我才获得了更多的机会，才一步一个脚印爬上去的。世上没有什么轻松的事情，设定目标，完成、努力，慢慢的你就会有进步。

3.2 抓住管理的机会

我在管理的能力那点已经提到过，管理并不是一个title，日常工作中，其实很多人就已经有管理的实践机会了。有没有抓住这个机会，做出成就，完全取决于你自己。

3.3 学会读代码

工作中，可能你只是一个编写项目逻辑代码的底层程序员。但这不代表你拿不到更高层次的代码，比如开源项目的源代码。

然后，你应该明白我要说的，一切都取决于你自己。github上有无数大牛的开源账号，里面有各种各样值得你学习的源代码。你有没有去看呢？

3.4 写代码的习惯

找出生活中的需求，什么都可以，自己想要研究的，自己需要用的，设立一个你自己的项目。然后养成每天写代码的习惯，不管时间多么少，总要挤出一点时间来写自己的代码。

长此以往，你就会积累一笔宝贵的财富。建议建立一个你自己的github账号，以后出去面试的时候就有了利器。

3.5 学会沟通和写作

当你有了技术积累想要更向上走的时候，你必须培养自己沟通的能力，一个无法与人好好沟通的人是无法胜任manager职责的，而且无法与人好好交流沟通，技术钻研上也必然会遇到瓶颈。

记住，一个人的能量总是有限的。

写作是另一个很重要的素养。当你认为你很理解一个技术点的时候，尝试把它写下来，写成一篇技术博客。你会发现你自己认为之前很理解的东西其实很难被整理描述出来。而一旦你成功整理一篇技术博客，你对这个技术点的认识就会大幅提升。

3.7 give and take

看了上面的描述，不知道你有没有发现，提升自己其实很简单，关键在于你肯不肯付出时间和努力，没有什么事情那么美好，能让你在公司里干活的时候技术水平也一起嗖嗖提升。

很多时候你需要在闲暇时付出大量时间和努力，来研究当下可能并不能立刻对你的职业生涯有帮助的技术。和投资理财很相似，不是么？

工作年限并不是衡量一人的标准，能力才是。工作年限也不是决定你薪资的标准，你对公司的贡献才是。