1. 起因

**想总结一下，但是没有大块的时间，一直在构思懒的动笔，或者动笔了但是一次没写完就丢下了，后面再也捡不起来了。**

**想看某个部分的代码，但是没坚持多久就因为各种各样的事情偷懒了一段时间，再次想继续的时候接不上了。**

**怎么系统的建设自己的知识体系？**

**工程的最终目的—解决问题。**

从2013年毕业转眼工作接近4.5年，在小整年的时间节点做个小结，为日复一日的工作加点仪式感，大整年（10年）能更上一个台阶。恭喜自己选了这一行，相对有意思的一个行业，相比其他行业有更多按照自己的想法干活的自由，以及社会平均水平以上的收入。

代码并不是像小说一样从头到尾顺序写下来的，小说也有倒叙、插叙，但是你从第一个页看到最后一页总能看懂；但是多数代码并不是从第一行看到最后一行就一定能啃下来的，以现在的软件规模也不现实——工作之前多数同学都会对自己的精力有些盲目自信，觉得我可以加班、可以比别人付出更多的时间——但是现实很骨感：大多数人都很努力，而且真的有忙不完的事情，你怎么会有比比人更多的时间；再说，如果把全部空闲时间全部砸在这上面，搞得自己非常疲惫，换来的成长有什么意义。

所以重在方法，以及积累；现在很多网文推崇一万小时定律——任何领域投入一万小时就可以成为专家，但是一万小时太恐怖，若每天拿出一小时的空闲时间全年365天不间断，一万小时大约需要三十年，你可能会铆足劲头奔着30年后称为某领域专家一直努力下去，抱歉我是做不到。但是想一下大学的课程，一门课多数集中在30—60学时，用兴趣和习惯去积累。

需要跟上下游模块交互的代码，比如一个网卡驱动软件，上层软件调用驱动软件的接口向远端设备发送数据，数据发送完了这个阶段就完成了，如果埋头看代码那么看到这就断片了——数据发出后是整个过程就完成了，还是只是一个阶段完成了？如果只是阶段完成，远端设备会给我回复什么样的数据，处理这些数据的入口在哪？没有对整个通信过程的了解很难理解驱动的代码逻辑。

所以对于逻辑框架的了解至关重要。首先了解代码和处理逻辑分为几个部分、几个阶段，再把每个阶段的入口拎出来。完成这两步对于负责的模块就了解了一半，剩下的无非就是了解每个阶段的实现细节，理解有快有慢但总有一天会熟悉。对于逻辑框架的理解可以通过老员工讲解、模块文档、芯片手册、概要设计等方式获取，

考古式阅读

怎么都看不懂代码，理不清的数据结构，各种钩子（函数指针），后面用到的时候已经不记得初始化阶段给设置成什么值了。

汉光善于总结、去维护组时不要荒废了C语言，由于太过零散边总结边丢弃，并没有形成系统的体系。

得益于github,工作流引擎（JIRA）,goole的浏览器书签，libreoffice draw的画图。

目标：帮助自己更好的成长，规格严格、训练有素的职场人，以及有更多的空闲时间过更有趣的生活；

帮助刚入行的程序员同行们更快的成长。

2.成长

2.1效率提升

个人:git管理代码（廖雪峰），有道云笔记，draw画图记录代码，中断后可以连续。

可能很多人都想总结一下，但是一旦一鼓作气没完成，后面就会不知道丢到哪去，再也找不回来了。Git的优点在于多台设备同步，这样有灵感的时候记录的一些琐碎的事务、杂乱无章的随手记，心血来潮的时候整理一点，随手push一下；到另外一台设备上pull一下就是上次终结的地方，修改完后再随手push一下。。。慢慢会形成一篇像模像样的东西。

知乎上有很多非程序员使用git的讨论，可见它的价值不菲。

团队:版本跟踪记录，标准化运作

2.2聚少成多：零散时间的利用和知识体系建设

地铁上用图片《内核设计与实现》

兴趣，每日半小时，python，django，mysql，nginx部署，阿里云，虚拟化

3.娱乐

单车、耳机

4.英语

5.不足

业务场景，存储怎么使用，块、文件、对象存储，虚拟化场景下有哪些可以提升，如文件系统，VAAI特性应该进一步了解。

系统的学习内核；

调试手段欠缺，makefile编写等

版本记录、保存home/debug用于搭建调试环境等