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。**

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。

============================================================================================

从2013年毕业转眼工作接近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。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。

## 怎样看代码

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。

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，也就是说用兴趣和习惯去积累一个月就可以了解一个专业的一门专业课，同样地，在统一个领域的持续投入会有持续的提高。所以重在方法，以及积累 -- -- "坚持"是一个听起来悲催的词。

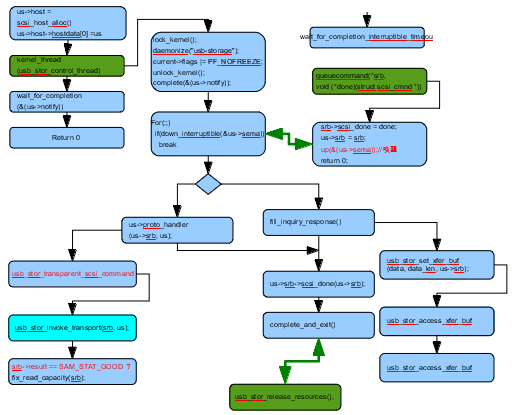
比如需要跟上下游模块交互的代码，以网卡驱动软件为例，上层软件调用驱动软件的接口向远端设备发送数据，数据发送完了这个阶段就完成了，如果埋头看代码那么看到这就断片了——数据发出后是整个过程完成了，还是只是一个阶段完成了？如果只是阶段完成，远端设备会给我回复什么样的数据，处理这些数据的入口在哪？没有对整个通信过程的了解很难理解驱动的代码逻辑。

此外，上下文的衔接也是一个令人头疼的问题：工作中你所要负责的代码通常要耗费一周或者几周的时间才能初步有所理解。如果没有一种很好的方法记录下分析代码的进展，每次分析都从头开始就会重复阅读开始的一部分代码而进展缓慢；在代码分析过程中还涉及到大量相互关联、错综复杂的数据结构：如一些变量的初始化和真正使用并不在一个阶段里面，真正使用的时候早已忘了这个变量被初始化成什么值，这时候再去翻初始化的代码非常耗时，还有可能翻到了初始化的部分又忘记了当前进展。

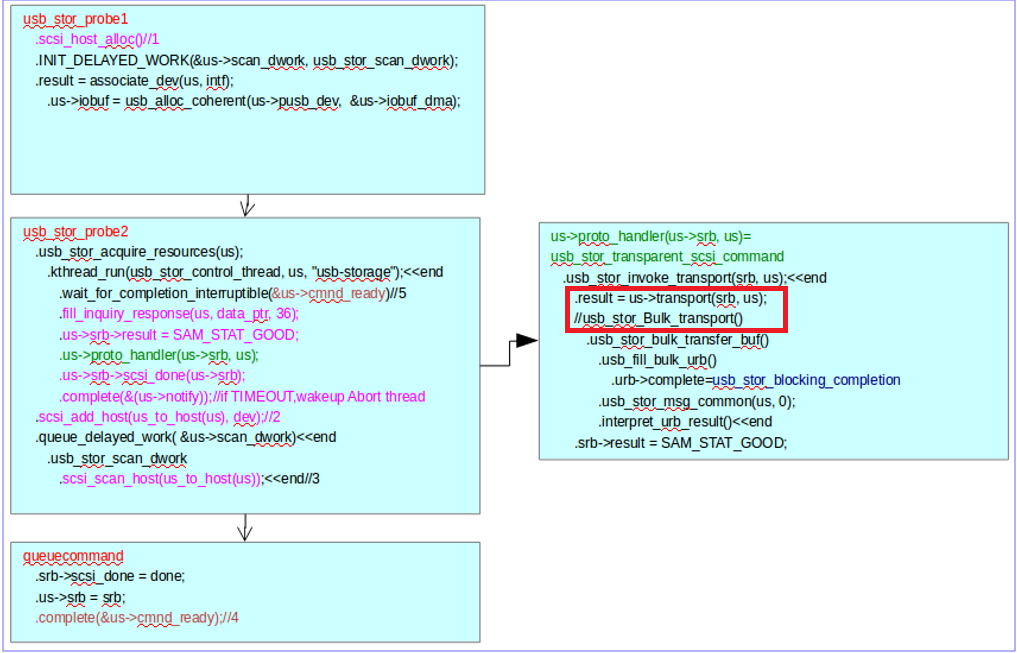
根据上面两段提出的问题也就知道了高效阅读代码的两个关键点：**一是理清逻辑框架；二是做好代码记录。**

**理清逻辑框架**首先要了解代码和处理逻辑分为几个阶段，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。像考古一样阅读代码——先用铁锹挖去周边的泥土露出文物的轮廓，再用铲子清理露出文物的表面，最后用毛刷去除尘土露出文物的纹理。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，对于驱动软件开发一般的芯片手册会有对芯片使用场景的介绍或者demo代码，可以了解大致的流程。

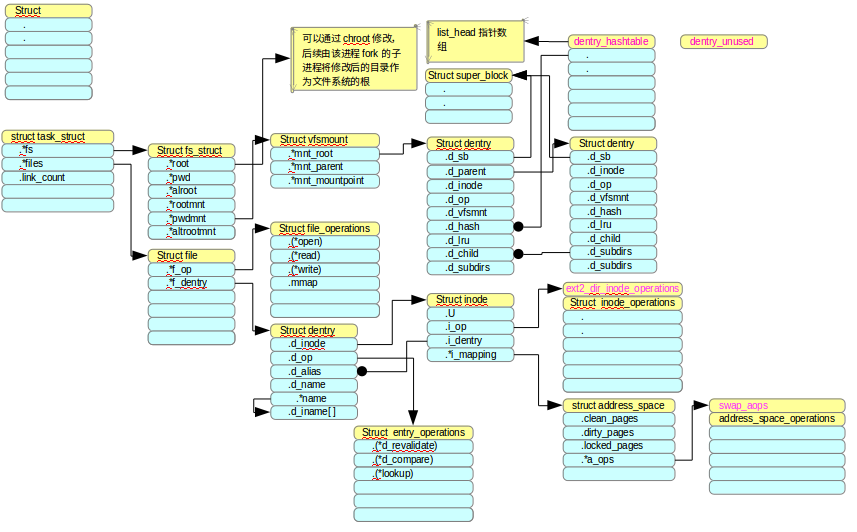
**做好代码记录**建议建立两个图，一个是代码执行顺序/调用关系图，一个是数据结构组织图。相信不少人对记录代码执行顺序/调用关系做过尝试，如下图是以前采用的一种方式，优点是能够清晰的展现出一些流程分支等逻辑细节，缺点是这种方式画图非常耗时，信息密度（一个作图页面能记录的内容）也很低。



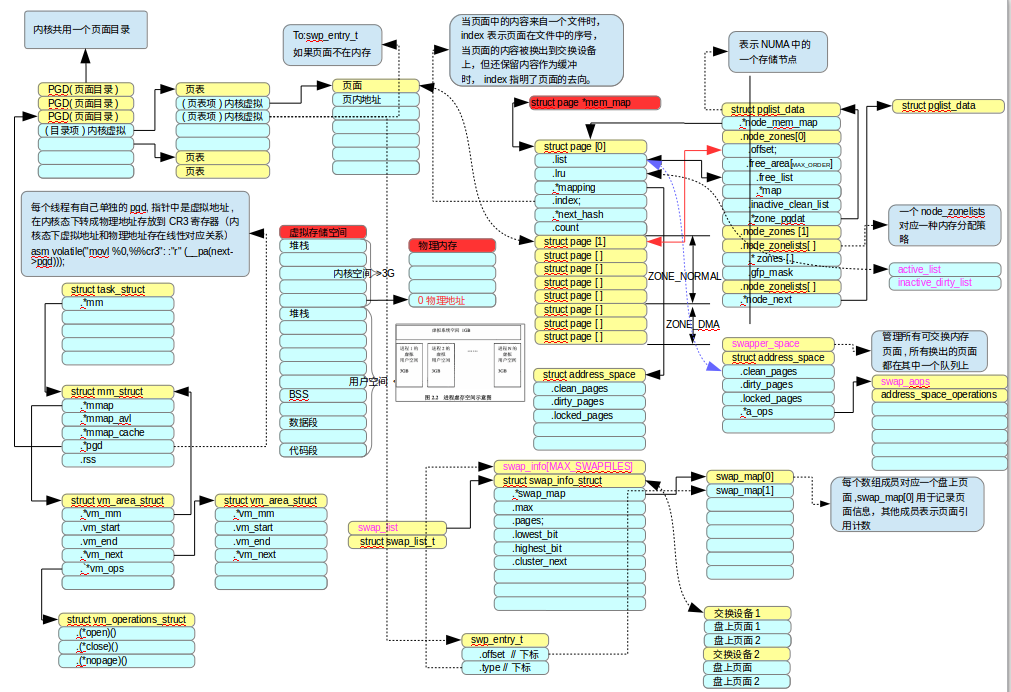
个人习惯用libreoffice draw画图工具记录代码执行顺序/调用关系：自上而下表示顺序执行，通过缩进表示调用关系，缩进层次过深时在右边另起一个区域。遇到函数指针时，在同一级缩进下用注释符号”//”记录实际执行的函数，如下图红色矩形框标注的部分。利用这种方式代码的**阶段划分**、**调用关系**以及**函数指针取值**都能清晰的表现出来；其次信息密度较高，通常一个模块也就几个主要流程，每个流程用一个框图就能将关键信息展示出来，且可以随时添加关键信息，显著提升阅读代码效率。



阅读代码时通常遇到函数一层层调用，参数一次次传递；在分析最内层函数时早已忘记函数参数的指代对象是什么。数据组织结构图可以快速帮助找到函数参数的指代对象，并形成整个软件结构的整体认识。一个数据组织结构图示例如下图所示，一些重要的数据成员可以添加注释，一些全局变量可以重点标注，对代码理解很有帮助。



内存管理的数据组织结构图示例：



2.成长

## 效率提升

**个人:** 工欲善其事必先利其器，好的工具可以起到事半功倍的效果，顺手的工具主要有git、有道云、libreoffice draw、tasksteper、notepad++、正则表达式等。

git管理代码（廖雪峰）

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。

有道云笔记，draw画图记录代码，中断后可以连续，问题归档（文件和问题单，【20180203】XX局点XX设备发生XX问题），使用git的案例（版本记录）。

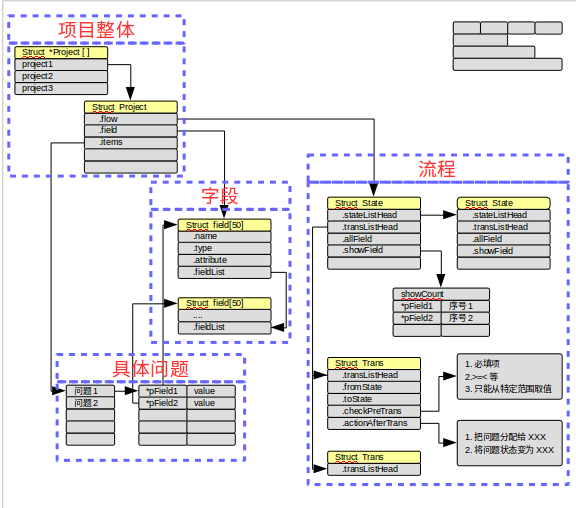
正则表达式，同时过滤多个关键词分析日志

怎样快速查找某次修改的commit\_id?

1.git log [xxxfile];2.过滤出commit xxxxx信息；3.替换，将commit XXXX替换为git show xxxx；4.再次替换将(.\*)替换为echo “\1” >>logfile\n\1 >>logfile 5.执行命令 6.打开logfile用”(commit)|(关键词)”搜索，可以快速找到修改的内容以及对应的commit。

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，像管理bug一样管理每一项日常事务，反对邮件。



2.2聚少成多：零散时间的利用和知识体系建设，网页书签的保存

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，慢慢来，比较快

版本记录、保存home/debug用于搭建调试环境等