

# 必读

## 作者简介

哈喽，大家好。我是仲一，目前在某大厂搬砖，从事驱动开发相关工作。很遗憾，本硕都是双非，没有耀眼的学历。因此，我在2020年3月份就开始准备秋招了。

**幸运，自己的努力没有白费。庆幸，在重要关头，我都能遇到贵人相助。**所以，我把自己秋招春招总结的内容都分享了出来。希望可以帮助到更多的人。凭借这份资料，最后顺利拿到了**oppo, 小米, 兆易创新, 全志科技, 海康威视等十余家家公司的offer**。

如果大家在网上看到了不错的资料，或者在笔试面试中遇到了资料中没有的知识点，大家可以关注**我的公众号**联系我，我替大家整理。

资料如有错误或者不合适的地方，可以在github向我提交issues。由于精力有限，所以只会用心维护好**github和公众号**两个平台。资料中的**勘误**也会同步更新在github中。

[github链接（点击跳转）](#)

[gitee链接（点击跳转）](#)



我自己运营了一个公众号，主要分享**计算机基础, 操作系统, 笔试面试, 驱动开发**等相关文章。非常欢迎大家关注我的公众号。

当然，想要加我微信好友进一步聊天的，也可以关注我的公众号，里面有我的**联系方式**。

## 嵌入式软件工程师笔试面试指南简介

嵌入式软件工程师笔试面试指南，详细分成了**笔试面试准备, 笔试面试八股文总结, 面经总结, 名企笔试真题解析**等四个部分。

其中，八股文又分成了C/C++，数据结构与算法分析，Arm体系与架构，Linux驱动开发，操作系统，网络编程等六个部分。

资料未经作者授权严禁各类培训机构私下传阅，如有发现，必将追究责任！

## 如何使用这份资料

---

嵌入式软件开发分为**应用层开发**和**驱动开发**。如果你想应聘应用开发，Linux驱动开发部分可以不看。如果你想应聘驱动开发，C++部分和网络编程部分可以不看。阅读顺序没有严格要求，各个部分都是独立的。像**操作系统**，**数据结构与算法**等内容都是通用的，是我们必须学习的。

资料内容很多，全部背下来不太现实，我建议大家以理解的方式来记忆，实在理解不了的再去背诵。此外，大家在准备秋招的时候，将重点放在自己最薄弱的部分，补齐短板。

## 嵌入式软件工程师笔试面试指南的价值

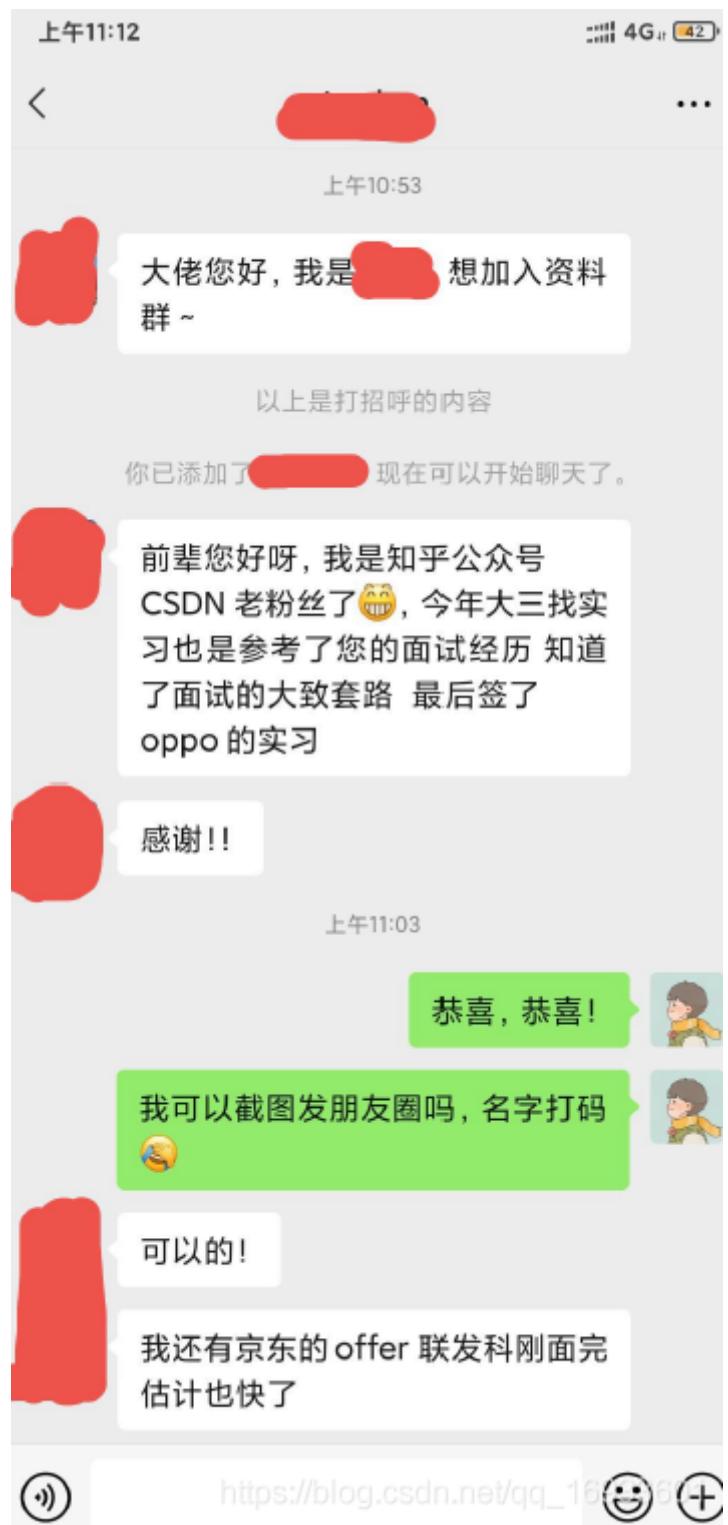
---

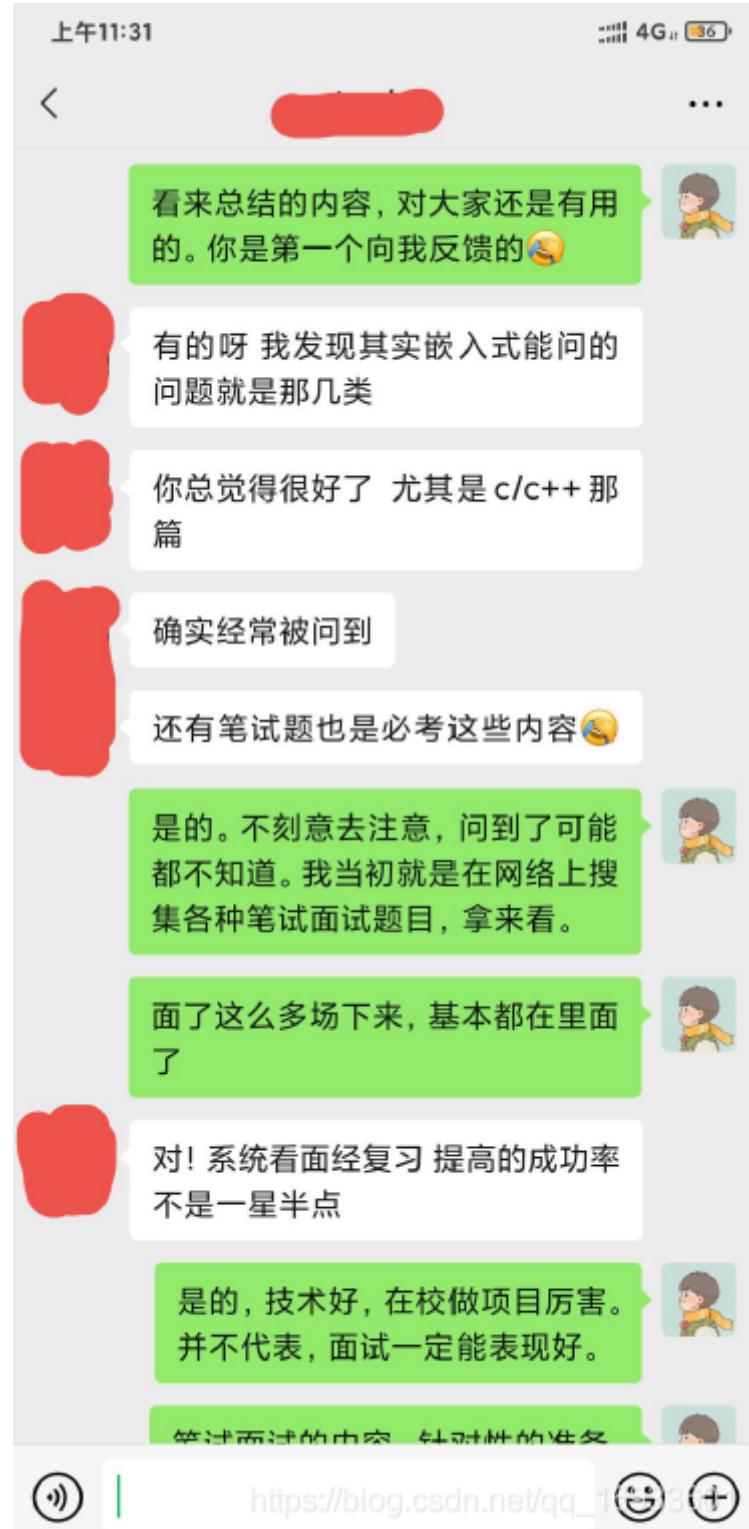
为了方便大家交流，互相学习。我建立了一个技术交流群，群内大佬很多，大家的问题都可以得到及时解答。截至到2021年5月12号，已经有多位同学收到了**华为**，**大疆**，**oppo**，**联发科**等大厂的offer，真心为这些同学感到开心。看下这些同学是怎么说的吧。

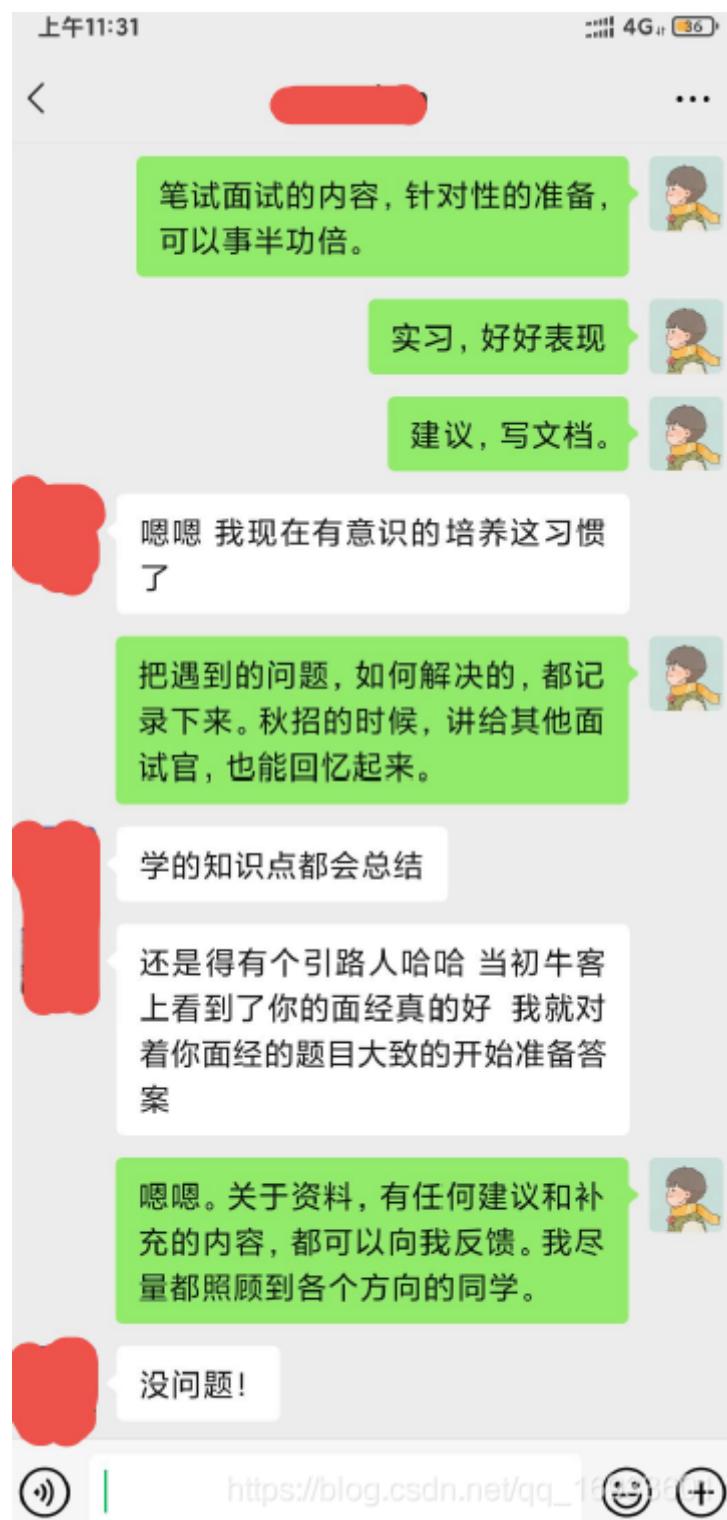
## 某外企面试官对这份资料的评价（20210514）



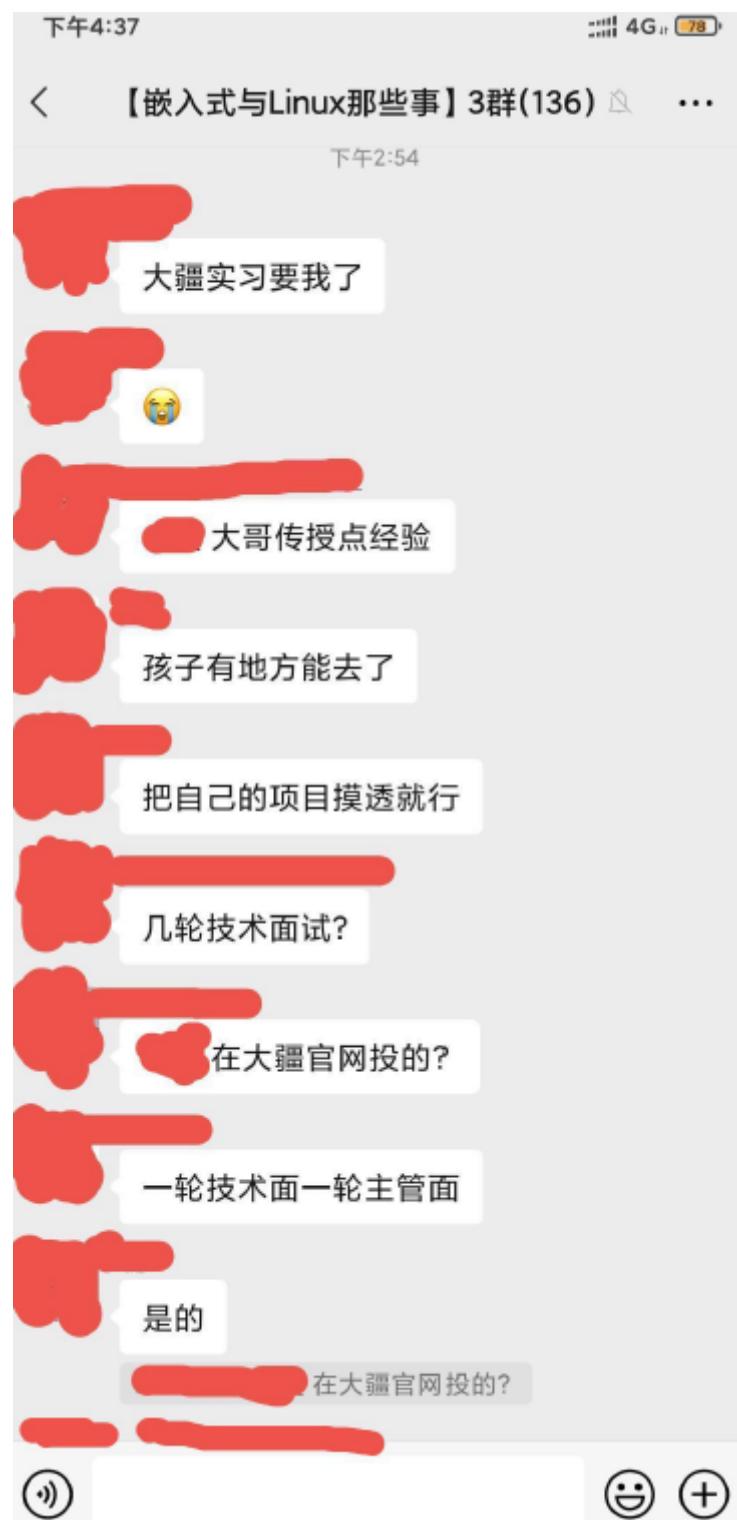
## 收获 oppo 联发科 京东offer应届生的评价 (20210430)

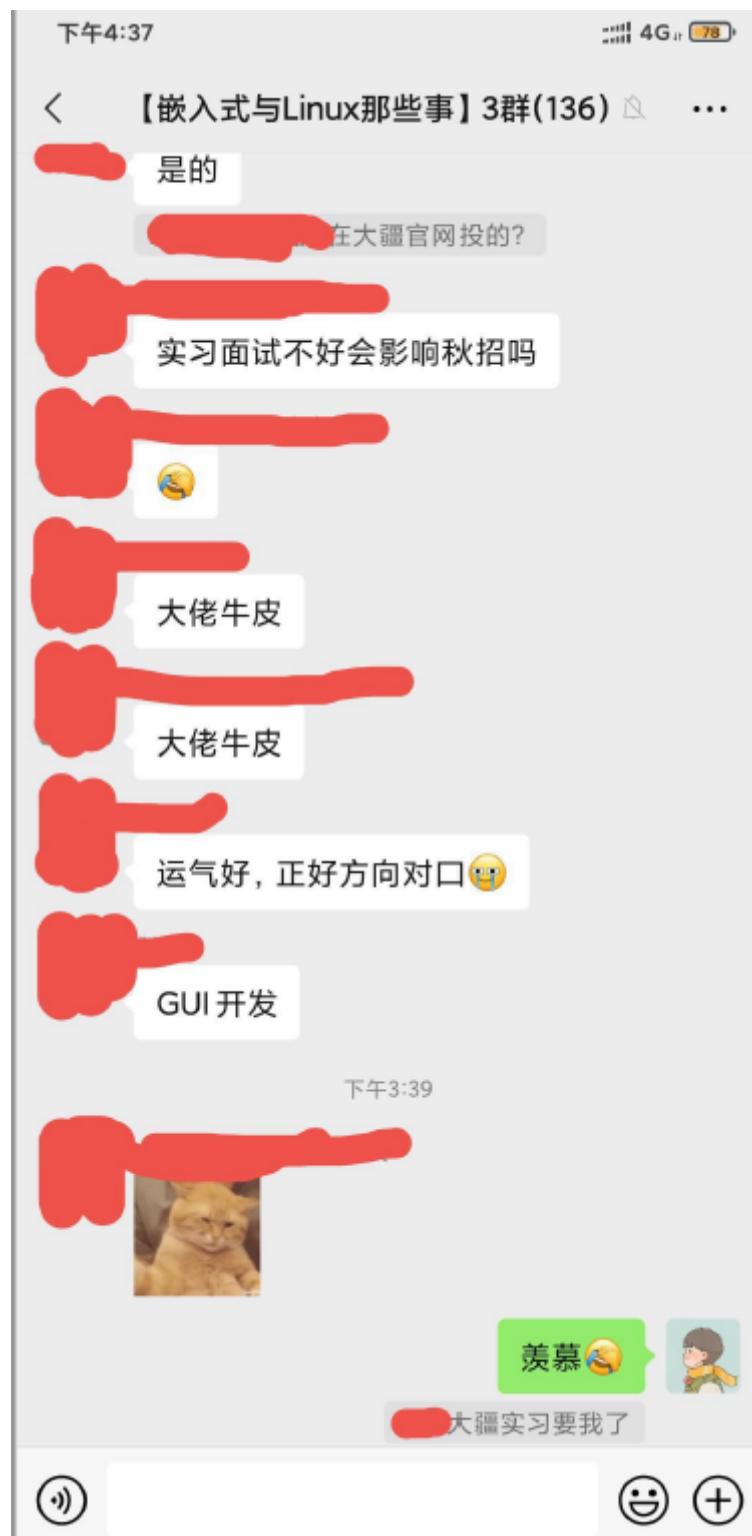






大疆offer (20210430)





华为offer (20210430)

华为下午技术面 + 主管面

感觉不难

主管面问了个项目直接跟我谈薪了

7k 一个月



华为。

??

是不是外包啊

假华为吧

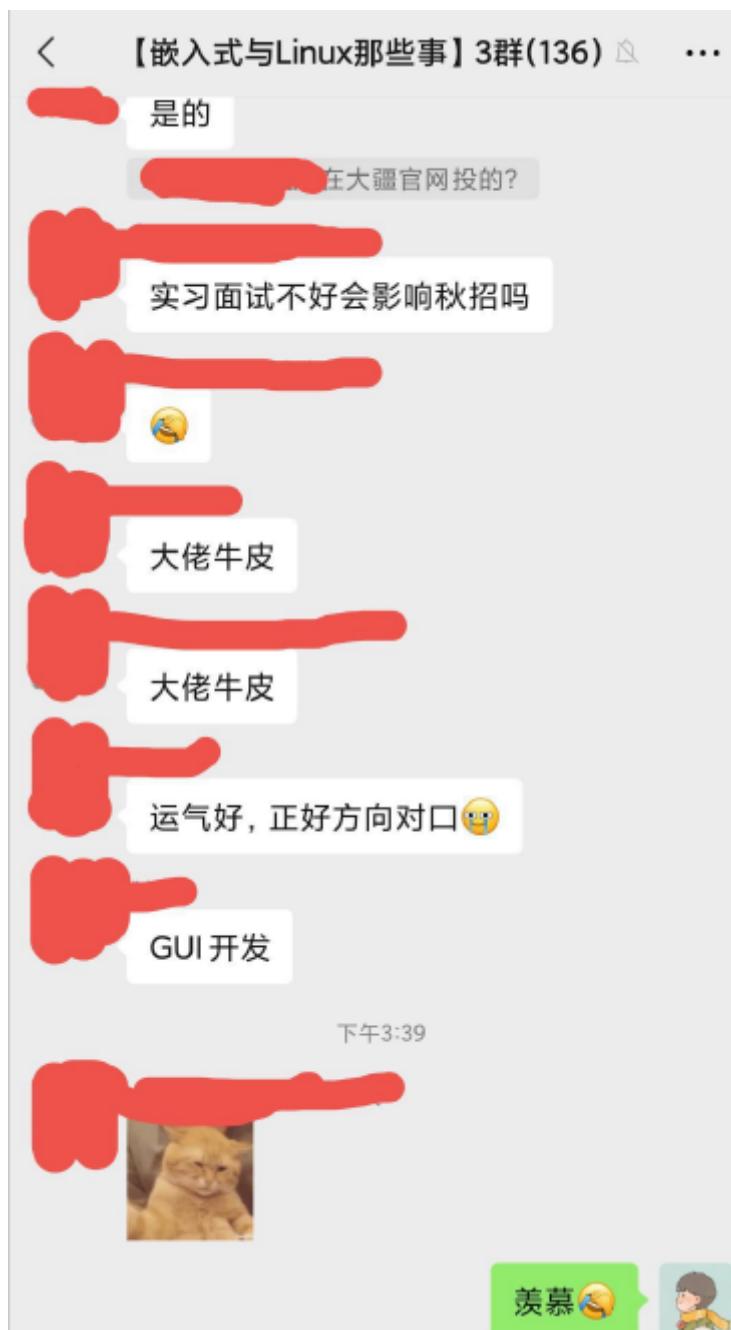
实习

"████████" 撤回了一条消息

不是春招

██████ 哪部门





欢迎各位小伙伴加我微信好友，一起聊天唠嗑，共同学习进步。

## 笔试面试准备

### 如何获取校招信息

#### 就业网站

第一个渠道就是学校和学院的就业网站：

关注宣讲和招聘信息；另外就是秋招和春招的时候学校都会举行招聘会。不要忘记参加。

#### 企业官网

第二个渠道就是企业的官方途径：

主要是指各大中型公司的招聘官网。还建议关注企业招聘公众号、微博号等，还可以直接跟公众号互动，留言，一些疑问有的时候可能会被解答。

这些地方的招聘信息通常最全，不仅包括招聘岗位、网申笔试面试的时间、还有招聘流程，校园宣讲会时间地点等。

## 招聘APP

第三个渠道就是招聘网站和APP，它们综合了大量企业的招聘信息，可以获得比较多的校招信息。下面是我整理的质量还不错的招聘网站，一般来说知名度高的网站肯定是信息量最大的地方：

### 1、前程无忧（51job）：

一个非常老牌招聘网站。

### 2、智联招聘：

老牌招聘网站，和51job差不多。

### 3、拉勾校招：

专注于互联网行业招聘，能够聊天。

### 4、boss直聘网：

boss上互联网企业比较多，可以直接跟HR沟通，运气好的话，可能跟boss直接沟通了

### 5、大街网：

一个综合类招聘网站

### 6、实习僧：

校招和实习的好网站，雇主信息质量比较高

### 7、牛客网：

牛客网是不仅可以看到各企业的招聘信息，而且还有强大的面试经验分享，往年笔试真题分享等，一些互联网公司HR经常会直接在上面发帖，如果你在上面勾搭到已经入职的小哥哥小姐姐们，说不定还可以让帮忙内推到心仪企业。

### 8、应届生求职网：

这个网站校招信息全，也可以查询到各地各校的宣讲会，而且它的文库里面有求职大礼包，上面整理了一些企业的详细介绍，面试流程和经验等；

### 9、海投网：

海投网宣讲会查询系统对应届生很实用，可以看到各城市各高校近期将举办的宣讲会，相当于资源整合，方便大家选择，也有笔面试经验和真题分享，还有简历在线制作，模板下载。

### 10、刺猬实习：

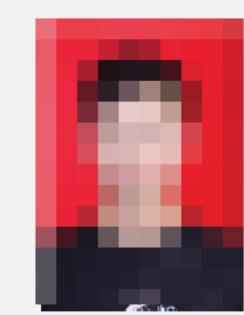
类似于实习僧，个人感觉信息和雇主信息比不上实习僧。



# 程序员如何写一份合格的简历？

今天不聊技术，来聊一聊如何写一份合格的简历。前两天，在交流群看到了一个同学问如何写简历。于是，我就让他把简历发给了我。简历的制作过程考验了一个人的两个能力，逻辑能力和细节能力。而不考验设计能力。下面就这份简历存在的问题，以及如何写简历做个简单总结。

## 1. 原始简历



④ 男 22岁 西安  
 ⑤ 本科  
 ⑥ 学生  
 ⑦ 133[REDACTED]0882  
 ⑧ cee[REDACTED]b@qq.com  
 ⑨ 身份证 [REDACTED]0877  
 ⑩ 籍贯 陕西渭南  
 ⑪ 政治面貌 中共党员

**自我评价**

本人为物联网工程专业2021届毕业生；在西安粤嵌科技受到嵌入式软件开发培训一年，系统学习了嵌入式软件开发相关课程，在技术方面打下了良好的基础。在校期间注重专业课知识学习的同时也积极参加各种学科竞赛及专业能力比赛：中国工程机器人大赛、互联网+、挑战杯、全国大学生物联网竞赛、全国大学生电子设计竞赛等等；在校专业课成绩优秀，赢得了各老师的认可；有良好的沟通和团队协作能力，热爱集体，对工作充满热情；分析和学习能力强，有较强的责任感。

**教育经历**  
 2017.08 - 2021.06 本科 [REDACTED] 物联网工程 专业排名1%~5%

---

**校内经历**  
 2018.11 - 2019.11 [REDACTED] 特训营(国旗护卫队) 特训营宣传部部长  
 负责社团对外的宣传和外交工作，主要包括抖音、微博、公众号等各大平台的运营及团队的管理；配合校团委对全校师生进行红色爱国主义宣传教育。工作期间获得校级新媒体最佳运营奖。

---

**求职意向**  
 西安、可接受调剂 技术支持工程师、解决方案工程师

---

**项目经历**  
 2020.05 - 2020.07 基于zigbee和wifi技术的复杂环境下救援机器人 嵌入式软件工程师、c/c++软件开发  
 1.通过wifi来传输摄像头的视频画面，通过zigbee模块来传输环境  
 2.采用GPS定位模块来获取目标的位置信息  
 3.采用仿生蜘蛛的六足结构，灵活应对各种复杂地形  
 4.通过超声波和红外进行自主避障，也可通过2.4G的无线遥控进行多元化控制  
 5.通过人体红外传感器进行疑似生命体征的检测，并向上层报告位置信息  
 6.利用zigbee自组网优势可将一个机器人扩展为一个机器人集群，扩大搜索规模  
 项目成果：获得2020华为杯大学生物联网设计竞赛国赛三等奖、西北赛区特等奖

负责模块：项目前中期的市场调研、系统硬件环境搭建及调试。

该系统是基于深度学习和物联网应用构建的果树病虫害智能预警与诊断系统。

项目研究的关键问题：

1.优化果树病虫害智能诊断的策略，实现病虫害快速诊断。动态实时地监控作物生长状况，实时、定点进行病虫害监测和预警。

2.通过物联网的视频监控终端、自动监控、防灾减灾。

3.实现农户与农户、农户与专家之间的交流，提高农作物病虫害的综合防治水平。

项目成果：获得2019大学生创新创业大赛校级一等奖、2019互联网+大赛校级一等奖。

### 获奖情况

2020.12 2020年中国工程机器人大赛暨国际公开赛双足竞步窄足项目全国二等奖-两项

2020.12 2020年中国工程机器人大赛暨国际公开赛双足竞步交叉足项目全国三等奖

2020.12 2019-2020学年 国家励志奖学金

2020.08 全国大学生物联网设计竞赛（华为杯）西北赛区决赛特等奖

2020.08 全国大学生物联网设计竞赛（华为杯）国赛三等奖

2019.12 全国高校计算机能力挑战赛（c语言程序设计）西北赛区三等奖。

2019.10 “互联网+”大学生创新创业大赛校赛一等奖

2019.05 挑战杯陕西省大学生课外学术科技作品竞赛三等奖。

2019.05 校级优秀学生干部

### 个人技能

有一定的Linux基础，熟悉Linux环境下的IO编程、进程线程、网络等；

有扎实的C/C++编程基础和项目经验，具有良好的代码编写习惯；

具有扎实的数字电路、模拟电路、微机原理等硬件专业理论基础知识；

熟悉专业相关方面技术文档的编写；熟悉软件项目的一般开发流程；

熟悉Mysql、Sql Server，能够熟练使用SQL语言；

### 论文发表

截至目前，已申请实用新型专利5项。其中3项是第一发明人两项已受理，一项是第二发明人(已授权)。

## 1.1 存在问题及改进建议

存在问题	修改意见
发给我的时候，简历命名为“我的简历+王五”	简历命名：岗位+学校+姓名+邮箱。 嵌入式软件工程师+清华大学+王五+ <a href="mailto:XX@163.com">XX@163.com</a>
教育经历跨行书写	最好写成一行，包括起止时间，学校，专业，学历，GPA(选填) 2018.09 - 2021.06 清华大学 信息与通信工程 硕士 GPA:3.63/4.0 (20%) 2014.09 - 2018.06 清华大学 电子信息工程 本科 GPA:3.72/4.0 (5%)
应聘的是技术岗位，校内经历和岗位关系不大	校内经历可以不写。如果简历放得下，这段经历往后放
求职意向在简历中放的位置不合适	求职意向和个人信息放在一起。不要穿插在校内经历和项目经历之间；非必要情况下，不要只写一个期望工作地点
左侧个人信息中“西安”表述不明确	工作地点：西安
左侧个人信息中邮箱名字太长	建议换一个163邮箱，邮箱名字可以是自己英文名字，中文名字拼音，电话等。
左侧个人信息中身份证号不需要写	身份证号就不需要写了
左侧个人信息中自我评价太长，没有重点	有较强的学习能力：成绩排名前5%，获得三次一等奖学金 有较强动手能力和团队协调能力：作为负责人，多次组织参与互联网+、挑战杯等大型比赛，获国家级奖励2次，省部级奖励5次
项目经历中，重点不突出	建议换一种格式书写。具体格式见下；项目经历主要写自己做了那些，而不是写项目介绍
项目经历中，没有项目开发环境	可以补上项目开发环境，比如，GCC3.4.2, Linux3.4.2内核, Ubuntu16.04, S3C2440开发板
第二页简历中左侧空白太多	很多网站导出的简历格式都是有问题的，建议自己可以用Typora或Word做一份
第二页简历中获奖情况写的太乱	建议相同类型的放在一起。优先级如下：竞赛类>奖学金>论文>专利>其他
第二页简历中个人技能排放位置太靠后	个人技能可以放在第一个。建议按照以下顺序排列：个人技能，项目经验，获奖，论文，个人评价。 个人技能尽量写和岗位匹配度高的。不知道怎么写的，可以去智联招聘看下相关岗位要求。
简历总共三页，太长了	简历最多两页。专利可以单独列一栏，也可以和获奖情况放一起。视个人情况而定。

## 2. 书写简历注意问题

### 2.1 个人信息

姓名，出生年月，联系方式，籍贯，电子邮件，政治面貌，毕业院校，专业，求职意向（选填），期望工作地点（选填）。

在官网投递简历时，一般会选择一个确定的工作岗位。面试官面试的时候，拿到的简历都是一个岗位的。

### 2.2 邮箱

最好使用163邮箱。邮箱名字可以是姓名全称，姓名简写，电话号码等。千万不要用[aeeeeeeee@163.com](mailto:aeeeeeeee@163.com)。HR看了会骂人的！

### 2.3 教育经历

时间，学校，专业，学历，GPA。教育经历一般由高到低写，如果绩点比较高，可以写上。

### 2.4 专业技能

尽量写和岗位匹配度高的。如果你应聘的是Linux驱动工程师，可以按照以下方式写。

- 具有一定硬件基础知识，能看懂原理图和Dataheet
- -熟练掌握c语言，具有良好编程风格，掌握Gcc、Shell等开发工具

如果你应聘的是嵌入式应用工程师，可以按照以下方式写。

- 熟练Qt与C++编程，熟悉QSS，能够使用样式对界面和控件进行美化处理，可以独立编制定制控件，有良好的产品交互意识
- 熟悉TCP/UDP的Socket编程、Http协议、Xml解析，串口等相关知识

如果你不知道怎么描述这些技能，可以去智联招聘看看对应岗位的一些要求。

#### 防止恶意转载

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：[https://blog.csdn.net/qq\\_16933601/article/details/112982065](https://blog.csdn.net/qq_16933601/article/details/112982065)

### 2.5 实习经历

主要从以下几个方面来写。

**公司名称：**XXX

**部门/岗位职责：**你所在的部门，以及你的岗位，你担任该岗位具体的工作内容是什么。其中的工作内容可以往你将要面试岗位的要求上靠

**实习时间：**2020.03~2020.06

**参与项目简介：**主要介绍项目主要内容，4~6句话即可。

**负责事宜：**写自己做了哪些内容。**第一**，XXXX；**第二**，XXXX；**第三**，XXX。

**实习收获：**这段实习经历给你带来的最大收获。这一点里，你可以适当写一些你今后的职业规划以及打算。

## 2.6 项目经验

一定要清晰明了，重点突出！千万不要写一大段话，面试官是没有耐心看这么多的。如果自己做过一些项目的话，建议写两到三个自己做过的项目，主要从以下几个方面介绍：

**项目名称：**基于XXX的XXX

**个人角色：**项目负责人/模块负责人

**起止时间：**2020.03~2020.06

**项目描述：**主要介绍项目主要内容，4句话即可。

**编程语言和环境：**Gcc3.4.2, Linux3.4.2内核, Ubuntu16.04, S3C2440开发板

**负责事宜：**写自己做了哪些内容。**第一**, XXXX; **第二**, XXXX; **第三**, XXX。

## 2.7 荣誉及奖项

相同类型的奖项最好放在一起。如果应聘的是开发岗，可以按照以下优先级来写：**竞赛>奖学金>论文>专利**。如果应聘的是算法岗，按照以下优先级来写：**顶刊/顶会>专利>竞赛>奖学金**。

## 2.8 个人博客

如果有博客且内容比较充实，可以放一个链接。博客不仅能看出水平，还可以看出态度。

面试的时间很短，面试官有时候一天面十几个人，对于应试者的能力考查未必很全面，有些优点在那么短的时间内展示不出来。如果有博客、Github等，面试官可以提前用碎片时间了解，获得大致的印象。

## 2.9 自我评价

主要从以下几个方面写：**性格，学习能力，自我驱动能力**等。推荐采用三段式总分结构进行自我评价。下面给出一个模版。

- 有较强的学习能力：成绩排名前5%，获得三次一等奖学金。
- 有强烈的好奇心，对嵌入式底层比较感兴趣：自学了Linux驱动开发相关内容，移植了Linux3.4.2内核到S3C2440平台。
- 拥有良好的沟通和协调能力，注重团队协作：在校期间，作为负责人，曾多次组织同学参与项目书的编写，并成功申请到国家级基金项目1项，省级重大专项2项。

## 2.10 其他注意事项

- 英文请核对正确，注意大小写，比如GitHub，而非github，以官网为准
- 时间建议统一格式为yyyy.mm - yyyy.mm，比如：2018.01 - 2018.12，而非 2018.1 - 2018.12
- 简历颜色最多三种
- 简历不要花哨，清除图标、进度条等附加元素
- 了解、熟悉、精通等词汇谨慎使用，尤其是精通一词尽量不用
- 应届生写清楚毕业时间
- 如果有头像请放西服衬衫证件照

## 3. 简历模版推荐

### 3.1 [Markdown-Resume](#)

一个非常简洁的在线制作简历网站。缺点是模版比较少。

### 3.2 极简polebrief

免费且无需登录，制作方便，直接在模板上编辑，并且最后还可以导出Word、Pdf、Jpg等不同格式的简历。

### 3.3 五百丁简历

汇聚了各个行业的简历，涉及的职业类别也非常宽泛，简历非常有设计感。模板数量也很多，部分可能需要付费。

### 3.4 个人使用的简历模版

下面这个模版是我个人使用的一个，参照Markdown-Resume使用Word制作的。有需要的可以在公众号回复【简历】自取，如果需要帮忙修改下简历，也可以加我微信，私发给我。

#### 基本信息

姓名	嵌入式与 Linux 那些事	出生年月	1998. 3
联系方式	123456789	籍贯	北京东城
电子邮件	XXXX@163. com	政治面貌	中共党员
毕业院校	清华大学	专业	信息与通信工程



#### 教育经历

2018.09 - 2021.06	清华大学	信息与通信工程	硕士	GPA:3.63/4.0 (15%)
2014.09 - 2018.06	清华大学	电子信息工程	本科	GPA:3.72/4.0 (5%)

#### 专业技能

- 熟悉 Linux 内核的总线设备驱动模型，输入子系统框架，字符设备驱动框架，块设备驱动框架。
- 熟悉 Bootloader 工作原理。
- 熟悉 C 语言，Matlab 语言，熟悉常用的数据结构与算法。
- 熟悉 Keil，Matlab，VS2015，微信 Web 开发者工具，Linux 下项目开发编程环境。
- 熟悉基本硬件原理，能看懂芯片手册，能利用 Altium Designer 绘制四层普通电路板。
- 熟练使用示波器，频谱仪，具备软硬件联合调试的基本能力。
- 英语通过六级，能熟练的阅读英文文献。
- 了解 C++，汇编语言，Arm 体系与架构，操作系统基本概念。
- 了解 JavaScript，CSS，HTML 语言。



### 如何投递简历

对于应届生来说，投递简历的途径有很多。

第一种就是参加校园宣讲会的方式进行投递。

因为很多宣讲会结束后有笔试，如果万一错过了网申或者之前网申没过，但是笔试过了，照样可以赢得面试机会。参加宣讲会可以更多的了解企业的信息，岗位信息等，一些企业的宣讲会都会有互动活动，可以领奖品，而且有些互动活动会赠送类似于免笔试，直通面试卡的东西。运气好可以拿到。

去的时候简历带够，然后准备好笔试的东西就行了。

### **第二种就是网申。**

一般通过招聘网站或者企业官网投递，需要在系统上填写简历信息。但是强烈建议还是要准备一份自己的简历模板。

渠道在获取校招信息章节已经提到，请自行翻看。

### **第三种就是内推。**

- 1、找已经入职的学长学姐or朋友；或者已经获得offer或者有过实习关系的同学等；找他们帮忙内推，我今年就内推了好几个学弟。
- 2、没有熟人的话，可以自己去牛客网、知乎、或者一些论坛、贴吧等去找信息，这些地方会有一些分享经验或者帮忙内推的人，去勾搭他们。
- 3、还有就是提前参与企业一些活动，比如当校园天使，帮企业在同学间宣传，帮忙这样企业会给内推名额或者免笔试直接进面试的名额之类的；
- 4、关注一些公众号关于春招的推文，下面经常会发放一些内推码；

**不要以为内推就是麻烦了别人，其实基本上每个大厂内推成功都是有奖金的，你内推成功，内推员也是会拿到奖金的。所以内推本质是双赢的。**

## **如何在面试中介绍自己的项目经验**

在面试时，经过寒暄后，一般面试官会让介绍项目经验。常见的问法是，说下你最近的（或最拿得出手的）一个项目。

根据我们的面试经验，发现有不少候选人对此没准备，说起来磕磕巴巴，甚至有人说出项目经验从时间段或技术等方面和简历上的不匹配，这样就会造成如下的后果。

1 第一印象就不好了，至少会感觉该候选人表述能力不强。

2 一般来说，面试官会根据候选人介绍的项目背景来提问题，假设面试时会问10个问题，那么至少有5个问题会根据候选人所介绍的项目背景来问，候选人如果没说好，那么就没法很好地引导后继问题了，就相当于把提问权完全交给面试官了。

面试时7份靠能力，3份靠技能，而刚开始时的介绍项目又是技能中的重中之重，所以本文将从“介绍”和“引导”两大层面告诉大家如何准备面试时的项目介绍。

好了，如下是正文内容。

### **1. 在面试前准备项目描述，别害怕，因为面试官什么都不知道**

面试官是人，不是神，拿到你的简历的时候，是没法核实你的项目细节的（一般公司会到录用后，用背景调查的方式来核实）。更何况，你做的项目是以月为单位算的，而面试官最多用30分钟来从你的简历上了解你的项目经验，所以你对项目的熟悉程度要远远超过面试官，所以你一点也不用紧张。如果你的工作经验比面试官还丰富的话，甚至还可以控制整个面试流程（笔者在面试方面成精后也经常干这种事情，大家一定也能行）。

	你	面试官
对你以前的项目和技能	很了解	只能听你说，只能根据你说的内容做出判断
在面试过程中的职责	在很短的时间内防守成功即可	如果找不出漏洞，就只能算你以前做过
准备时间	面试前你有充足的时间准备	一般在面试前用30分钟阅读你的简历
沟通过程	你可以出错，但别出关键性的错误	不会太为难你，除非你太差
技巧	你有足够的技巧，也可以从网上找到足够的面试题	其实就问些通用的有规律的问题

既然面试官无法了解你的底细，那么他们怎么来验证你的项目经验和技术？下面总结了一些常用的提问方式。

提问方式	目的
让你描述工作经历和项目（极有可能是最近的），看看你说的是否和简历上一致	看你是否真的做过这些项目
看你简历上项目里用到的技术，比如框架、数据库，然后针对这些技术提些基本问题	还是验证你是否做过项目，同时看你是否了解这些技术，为进一步提问做准备
针对某个项目，不断深入地问一些技术上的问题，或者从不同侧面问一些技术实现，看你前后回答里面是否有矛盾	深入核实你的项目细节
针对某技术，问些项目里一定会遇到的问题，比如候选人事做过数据库，那么就会问索引方面的问题	通过这类问题，核实候选人是否真的有过项目经验（或者还仅仅是学习经验）

## 2. 准备项目的各种细节，一旦被问到了，就说明你没做过

一般来说，在面试前，大家应当准备项目描述的说辞，自信些，因为这部分你说了算，流利些，因为你经过充分准备后，可以知道你要说些什么。而且这些是你实际的项目经验（不是学习经验，也不是培训经验），那么一旦让面试官感觉你都说不上来，那么可信度就很低了。

不少人是拘泥于“项目里做了什么业务，以及代码实现的细节”，这就相当于把后继提问权直接交给面试官。下表列出了一些不好的回答方式。

回答方式	后果
我在XX软件公司做了XX门户网站项目，这个项目做到了XX功能，具体是XX和XX模块，各模块做了XX功能，客户是XX，最后这个项目挣了XX钱	直接打断，因为业务需求我不需要了解，我会直接问他项目里的技术
(需要招聘一个Java后端开发，会Spring MVC) 最近一个项目我是用C# (或其他非Java技术) 实现的，实现了.....或者我最近做的不是开发，而是测试.....或者我最近的项目没有用到Spring MVC	提问，你最近用到SSH技术的项目是什么时候，然后在评语上写：最近XX时间没接触过SSH
在毕业设计的时候（或者在读书的时候，在学习的时候，在XX培训学校，在XX实训课程中），.....	直接打断，提问你这个是否是商业项目，如果不是，你有没有其他的商业经验。如果没商业项目经验，除非是校招，否则就直接结束面试
描述项目时，一些关键要素（比如公司、时间、所用技术等）和简历上的不匹配	我们会深究这个不一致的情况，如果是简历造假，那么可能直接中断面试，如果真的是笔误，那么就需要提供合理的解释

在避免上述不好的回答的同时，大家可以按下表所给出的要素准备项目介绍。如果可以，也请大家准备一下用英语描述。其实刚毕业的学生，或者工作经验较少的人，英语能力都差不多，但你说了，这就是质的进步。

要素	样式
控制在1分钟里面，讲出项目基本情况，比如项目名称，背景，给哪个客户做，完成了基本的事情，做了多久，项目规模多大，用到哪些技术，数据库用什么，然后酌情简单说一下模块。重点突出背景，技术，数据库和其他和技术有关的信息。	我在XX公司做了XX外汇保证金交易平台，客户是XX银行，主要完成了挂盘，实盘成交，保证金杠杆成交等功能，数据库是Oracle，前台用到S等技术，后台用到java的SSH，几个人做了X个月。不需要详细描述各功能模块，不需要说太多和业务有关但和技术无关的。如果面试官感兴趣，等他问。
要主动说出你做了哪些事情，这部分的描述一定需要和你的技术背景一致。	我做了外汇实盘交易系统，挂单成交系统，XXX模块，做了X个月
描述你在项目里的角色	我主要是做了开发，但在开发前，我在项目经理的带领下参与了业务调研，数据库设计等工作，后期我参与了测试和部署工作。
可以描述用到的技术细节，特别是你用到的技术细节，这部分尤其要注意，你说出口的，一定要知道，因为面试官后面就根据这个问的。你如果做了5个模块，宁可只说你能熟练说上口的2个。	用到了Java里面的集合，JDBC，...等技术，用到了Spring MVC等框架，用技术连接数据库。
这部分你风险自己承担，如果可以，不露声色说出一些热门的要素，比如Linux，大数据，大访问压力等。但一旦你说了，面试官就会直接问细节。	这个系统里，部署在Linux上，每天要处理的数据量是XX，要求是在4小时，1G内存的情况下处理完5千万条数据。平均访客是每分钟XXX。

面试前，你一定要准备，一定要有自信，但也要避免如下的一些情况。

要避免的情况	正确的做法	原因
回答很简单。问什么答什么，往往就用一句话回答	把你知道的都说出来，重点突出你知道的思想，框架	问：你SSH用过吗？答：用过。问：在什么项目里用到？答：一个保险项目
说得太流利	适当停顿，边思考边说	让面试官感觉你在背准备的东西，这样后面问题就很难
项目介绍时什么都想说，	就说些刚才让准备的一些，而且要有逻辑地说	会让面试官感觉你思路太乱
别太多介绍技术细节，就说你熟悉的技术	技术面点到为止，等面试官来问	你说的所有技术要点，都可能会被深问。面试官一般会有自己的面试节奏，如果你在介绍时就太多说技术细节，很有可能被打断，从而没法说出你准备好的亮点。

### 3. 不露痕迹地说出面试官爱听的话

在项目介绍的时候（当然包括后继的面试），面试官其实很想要听一些关键点，只要你说出来，而且回答相关问题比较好，这绝对是加分项。我在面试别人的时候，一旦这些关键点得到确认，我是绝对会在评语上加上一笔的。

下面列些面试官爱听的关键点和对应的说辞。

关键点	说辞
能考虑到代码的扩展性，有参与框架设计的意识	我的项目XX保险项目，用到SSH技术，数据库是Oracle，（这个是铺垫），开发的时候，我会先和项目经理一起设计框架，并参与了框架的构建，连接数据库的时候，我们用到了DAO，这样做的理由是，把SQL语句封装到DAO层，一旦要扩展功能模块，就可以不用做太多的改动。
有调优意识，能通过监控发现问题点，然后解决	在开发阶段，我就注意到内存的性能问题和SQL运行的时间问题，在压力测试阶段，我会通过xx工具来监控内存和数据库，发现待提升的代码点，然后通过查资料来优化。最后等项目上线后，我们会部署监控系统，一旦发现内存和数据库问题，我们会第一时间解决。
动手能力很强，肯干活，会的东西比较多，团队合作精神比较好	在项目里，我不仅要做开发的工作，而且需要自己测试，需要自己根据一些日志的输出到数据库或Java端去debug，当我开好一个模块时，需要自己部署到Linux上测试。或者，一旦遇到问题，如果是业务方面的，我会及时和项目经理沟通，如果是技术方面的，我会自己查资料，如果是测试方面的，我会及时和测试的人沟通。
责任心比较强，能适应大压力的环境	被问“你如果在项目里遇到问题怎么办？”回答：遇到问题我先查资料，如果实在没法解决，不会拖，会及时问相关的人，即使加班，也会在规定的时间内解决。
有主见，能不断探索新的知识	在项目里，我会在保证进度的前提下和项目经理说我的想法，提出我的解决方案。在开发过程中，我会先思考一下，用一种比较好的方式，比如效率最高的方法实现。

## 4.一定要主动，面试官没有义务挖掘你的亮点

我去面试人家的时候，往往会特别提问：你项目里有什么亮点？或者你作为应聘者，有什么其他加分项能帮你成功应聘到这个岗位。即使这样问，还有些人直接说没有。

我这样问已经是处于角色错位了，作为面试者，应当主动说出，而不是等着问，但请注意，说的时候要有技巧，找机会说，通常是找一些开放性的问题说。

比如：在这个项目里用到了什么技术？你除了说一些基本的技术，比如Spring MVC，Hibernate，还有数据库方面的常规技术时，还得说，用到了Java内存管理，这样能减少对虚拟机内存的压力，或者说用到了大数据处理技术等。也就是说，得找一切机会说出你拿得出手的而且当前也非常热门的技术。

或者找个相关的问题做扩展性说明，比如被问到：你有没有用到过一对多和多对多？你除了说基本知识点以外，还可以说，一般我还会根据需求适当地设置cascade和inverse关键字，随后通过一个实际的案例来说明合理设计对你项目的帮助，这样就能延伸性地说明你的技能了。相反如果你不说，面试话一定会认为你只会简单的一对一和一对多操作。

面试的时候，如果候选人回答问题很简单，有一说一，不会扩展，或者用非常吝啬的语句来回答我的问题，那么我一般会给机会让他们深入讲述（但我不敢保证不是每个面试官都会深入提问），如果回答再简洁，那么也会很吝啬地给出好的评语。

记住：面试官不是你的亲戚，面试官很忙，能挖掘出你的亮点的面试官很少，而说出你的亮点是你的义务。

我在面试别人过程中，根据不同的情况一般会给出如下的评语。

1 回答很简答，但回答里能证明出他对框架等技术确实是做过，我会在评语里些“对框架了解一般，不知道一些深层次的知识（我都问了多次了你都回答很简答，那么对不起了，我只能这么写，或许你确实技术很强，那也没办法，谁让你不肯说呢？）”，同时会加一句“表达能力很一般，沟通能力不强”，这样即使他通过技术面试，后面的面试他也会很吃力。

2 回答很简单，通过回答我没法验证他是在项目里做过这个技术，还是仅仅在平时学习中学过这个技术。我就会写“在简历中说用过XX技术，但对某些细节说不上来，没法看出在项目里用到这个技术”，如果这个技术是职务必需点，那么他通过面试的可能性就非常小。

3 回答很简单，而且只通过嗯啊之类的虚词回答，经过提醒还这样，我会敷衍几句结束面试，直接写“技术很薄弱，没法通过面试”。

经理斟酌”。这样通过后继综合面试的机会就一般了，毕竟综合面试会着重考察表达能力交往能力等非技术因素。

**不管怎样，一旦回答简单，不主动说出你的擅长点，或没有条理很清楚地说出你的亮点，就算我让你通过面试，也不会写上“框架细节了解比较深，数据库应用比较熟练”等之类的好评语，你即使通过技术和后面的综合面试，工资也是比较低的。**

## 5.一旦有低级错误，可能会直接出局

面试过程中有些方面你是绝对不能出错，所以你在准备过程中需要尤其注意如下的因素。下面列了些会导致你直接出局的错误回答。

错误类型	导致的后果
前后矛盾，后面的回答无法证明你的项目描述，比如一开始说用到了Spring MVC，后面没法说出最基本的实现，比如不知道Spring有哪些类，或者没法说出项目的细节。	我会怀疑这个项目的真实性，我就会进一步问：数据库用什么，数据量多少？多少人做了多少时间，一旦再出现明显漏洞，比如一个小项目用到非常多的时间，那么就不仅仅是技术问题，而是在面试过程中企图“蒙混过关”的性质了。
项目里一定会用到的基本概念性问题都回答不上，Spring的依赖注入概念是什么，怎么用的，或者Hibernate的一对多怎么实现	一旦被我发现概念不知道，我就会通过更多问题确认，如果被我确认很弱，这就相当严重，因为技术能力差和技术没用过是两个截然不同的状况，技术没用过会导致直接出局。
面试时说出的工作经验和简历上的不一致	我会直接怀疑简历是编的，我会让候选人解释，即使是说简历写错了，我也会问比较深入的问题来核实他的技能和能力。
简历上的技能描述和回答出来的明显不一致，比如明明是只会简单的Linux，但吹得天花乱坠	我会通过一些比较深的问题核实其他技能，找出其他方面吹嘘的水分。所以建议，你可以适当夸张，但别过分，比如你在项目里没搭建框架但平时学习时搭建过，你可以写“XX项目的框架是你搭建的”，但你不能说你是一个架构师，非常了解项目的底层。
让面试官感觉你不稳定，很浮躁，比如说话不庄重，或者面试时打扮非常不正规，就穿背心来。	即使你技术再好，这个可能导致你直接出局。我对油嘴滑舌的候选人一般会直接写上不好的评语，这样很难过后面项目经理的面试。我还遇到一个人，简历上工作是半年一换，我问他为什么经常换，他直接说是待遇问题，这个人我是直接Fail掉。
明说不能加班，不能出差	其实虽然有这一问，但公司里未必真的会加班会出差。但听到这类回答，说明这个人不能承受大压力的工作，或者责任心不强，大多数公司是不会要这种人的。

## 6.引导篇：准备些加分点，在介绍时有意提到，但别说全

在做项目介绍的时候，你可以穿插说出一些你的亮点，但请记得，不论在介绍项目还是在回答问题，你当前的职责不是说明亮点而是介绍项目，一旦你详细说，可能会让面试官感觉你跑题了。

所以这时你可以一笔带过，比如你可以说，“我们的项目对数据要求比较大，忙的时候平均每小时要处理几十万条数据”，这样就可以把面试官引入“大数据”的方向。

你在面试前可以根据职位的需求，准备好这种“一笔带过”的话。比如这个职位的需求点是Spring MVC框架，大数据高并发，要有数据库调优经验，那么介绍以往项目时，你就最好突出这些方面你的实际技能。

再给大家举个例子，比如Java虚拟机内存管理和数据库优化是绝大多数项目都要遇到的两大问题，大家都可以在叙述项目经验时说，在这个项目里，我们需要考虑内存因素，因为我们的代码只允许在2G内存环境中运行，而且对数据库性能要求比较高，所以我们经常要监控优化内存和数据库里的SQL语句。这样当面试官深入提问时，就能抛出自己准备好的虚拟机内存优化和数据库优化方面的说辞。

实在不行，你也可以说“我除了做开发，也做了了解需求，测试和部署的工作，因为这个项目人手比较少，压力比较大”，这样你也能展示你有过独挡一面的经历。

我在面试过程中，一旦听到有亮点，就会等到他说好当前问题后，顺口去问，一般技术面试最多办半小时，你把时间用在回答准备好的问题点上的时候，被问其他问题的时间就会少了。

## 7.你可以引导，但不能自说自话

我面试的时候，也会遇到些有准备的人，其实如果你真的想应聘的话，一定要事先准备，这点我能够理解，甚至赞同，你只要别露出太明显的痕迹，我不会写上“似乎有准备，没法考察真实技能”这种话，更何况未必每个面试官都能感觉出你准备过。但你不能凭着有准备而太强势，毕竟面试是面试官主导的。

1 &nbsp;&nbsp;我遇到个别面试的人，他们说话太多，一般会主动扩展，比如我问他数据库用什么，他不仅回答数据库是什么，自己做了什么，甚至顺便会把大数据处理技术都说出来。

其实过犹不及，我就会重点考察你说的每个细节，因为我怀疑你说的都是你从网上看的，而不是你项目中用到的，我甚至会直接威胁：“你先和我说实话这个技术你真在项目里用到，我后面会重点考察，一旦被认为你项目里没做，这个性质就是蒙混过关了”，往往这些人会主动坦白。

不过话说回来，他如果仅仅说，数据量比较大，但点到为止，不继续说后面的话，我就会深入去问，他自然有机会表达。同时请注意，一般在面试过程中，一旦你亮出加分点，但面试官没接嘴，这个加分点可能就不是项目必备的，也不是他所关注的，当前你就可以别再说了，或者等到你提问题的时候再说。

## 8.不是结尾的总结

两句话，第一，面试前一定要准备，第二，本文给出的是方法，不是教条，大家可以按本文给出的方向结合自己的项目背景做准备，而不是死记硬背本文给出的一些说辞。

当大家介绍好项目背景后，面试才刚刚开始，哪怕你说得再好，哪怕你把问题引导到你准备的范围里，这也得应付Java Web（比如Spring MVC,ORM等）、Java Core（多线程、集合、JDBC等）和数据库等方面的问题。

那么本文的价值体现在哪呢？如果引导不好，你根本没机会展示自己的能力。这就是本文给出的方法价值所在。说句自夸的话，本文给出的一些方法和说辞不是拍脑袋想出来的，而是从面试上百个候选人的经历中抽取出来的，其中有不少血泪，也有不少人成功的途径，这篇文章多少对大家（尤其是经验不满3年的初级程序员）有帮助。



微信搜一搜

嵌入式与Linux那些事

## HR面试常见问题汇总

### 1. 语言表达、仪表

序号	题目	面试要点参考
1	简单的谈一下自己（自我介绍）	观察应试者的语言是否流畅、有条理、层次分明，讲话的风度如何。
2	请你告诉我你的一次失败经历。	如果能迅速作答，则应试者反应灵敏，或可能是应试者善于总结教训。
3	你有什么优点和缺点。	应试者对自己的判断是否中肯，自信、自卑和自傲倾向如何。
4	请讲述一次你做的不错的成绩。	考察应试者是否有能力。

### 2. 工作经验

序号	题目	面试要点参考
1	你现在或最近所做的工作，其职责是什么？在团队中是什么位置？	应试者是否曾关注自己的工作，是否了已工作的重点，表述是否简明扼要。
2	你认为你在工作中的成就是什么？	了解对方对“成就”的理解，了解对方能力的突出点，是否能客观的总结回顾自我
3	你以前在日常工作中主要处理些什么问题？	通过对方对自己工作的归纳判断其对业务的熟练程度和关注度。可依此继续追问细节。
4	以前工作中有过什么良好的建议和计划？	了解对方对工作的改善能力。要追问细节，避免对方随意编造或夸夸其谈。

### 3. 应聘动机与期望

序号	题目	面试要点参考
1	你最喜欢的工作是什么？为什么？请谈谈你在选择工作时都考虑哪些因素？如何看待待遇和工作条件？	同时可判断对方的分析能力和自知力
2	你为什么选择来我公司工作？你对我公司了解些什么？你为什么应聘这个职位？	只为找到一份工作糊口而盲目求职的培养潜质不高，但对公司的不了解不应成为重点
3	你对我公司提供的工作有什么希望和要求？	能大胆而客观地提出要求的优先，提出不切实际要求的可不予考虑
4	你喜欢什么样的领导和同事？	喜欢什么样的人，自己也将最终成为那种人
5	你认为在一个理想的工作单位里，个人事业的成败是由什么决定的？	价值观的一种。不同的职位需要不同价值观的人，但基本观念不能和企业文化相差太远
6	你为什么要选读这个专业？你所学的专业和我们的工作有何关系？	当对方专业与本职位关联不大时使用本条
7	你更喜欢什么样的公司？	判断对方在本公司的适应性和稳定性。

### 4. 事业心、进取心、自信心

序号	题目	面试要点参考
1	你个人有什么抱负和理想？你准备怎样实现它？	追问题，避免对方夸夸其谈
2	你认为这次面试能通过吗？理由是什么？	理想情况是既自信又不狂妄。
3	你认为成功的决定性因素是什么？	追问题：你认为自己具备其中的哪些？
4	你的职业发展计划是什么？如何实现这个计划？	有计划的人才是真正有进取心，但要看对方所描述的是否适合本职位。

### 5. 工作态度、组织纪律性、诚实可靠性

序号	题目	面试要点参考
1	你认为公司管得松一些好还是紧一点好？	无标准答案，关键在于对方思路
2	你在工作中喜欢经常与主管沟通、汇报工作，还是最终才做一次汇报？	无标准答案，工作习惯问题
3	你如何看待超时和周末、休息日加班？	理想情况是既能接受加班，又不赞成加班
4	你认为制定制度的作用是什么？怎样才能保证制度的有效性？	观察对方是否言不由衷

## 6. 分析判断能力

序号	题目	面试要点参考
1	你认为自己适合什么样的工作？为什么？	希望对方能切实结合自己的性格、能力、经历特点有条理地分析
2	你认为怎样才能跟上飞速发展的时代（行业）而不落后？	追问题：你平时主要采取一些什么学习方式
3	“失去监督的权力必然产生腐败”，对于这句话你怎么理解？	虽与工作无关，但主要观察对方的观察问题的角度与推导的思路
4	吸烟有害健康，但烟草业对国家的税收有很大的贡献，你如何看待政府采取的禁烟措施？	虽与工作无关，但主要观察对方的观察问题的角度与推导的思路

## 7. 应变能力（可选题）

序号	题目	面试要点参考
1	在实际生活中，你做了一件好事，不但没人理解，反而遭到周围人的讽刺和挖苦，这时你会如何处理？	反馈的时间应作为主要参考因素，若对方在20秒内还没有回答，自然转入下一个问题
2	在一次重要的会议上，领导做报告时将一个重要的数据念错了，如不纠正会影响工作。这时你会怎么办？	反馈的时间应作为主要参考因素，若对方在20秒内还没有回答，自然转入下一个问题

## 8. 自知力、自控力

序号	题目	面试要点参考
1	你认为自己的长处和短处是什么？怎样才能做到扬长避短？	关注对方对自己短处（长处）的描述
2	你听见有人在背后议论你或说风凉话，你怎么处理？	关注对方思维的出发点
3	领导和同事批评你时，你如何对待？	观察对方是否言不由衷
4	假如这次面试你未被录取，你今后会做哪些努力？	观察对方提到问题时瞬间的反应

## 9. 组织协调能力、人际关系与适应能力

序号	题目	面试要点参考
1	你担任过什么社团工作？	顺势追问细节，全面观察对方
2	你喜欢和什么样的人交朋友	营造轻松氛围，尽量让对方放低戒心，展开阐述，从中观察细节
3	从一个熟悉的环境转入陌生的环境，你会怎样努力去适应？大概需要多久？	不妨先举个实例引导对方，如：想象你到了一个陌生的城市拓展市场业务……
4	你更喜欢主动地开展工作还是由上级指挥工作？你喜欢独立工作还是与别人合作？	两类人都有可取的地方，当对方选择其中一个时，可追问他对他对另一类人的看法

## 10. 精力、活力与兴趣、爱好

序号	题目	面试要点参考
1	你喜欢什么运动？	将对方的兴趣分为身体接触对抗型、不接触对抗型、非竞争型、静止型、独享趣味型等再进一步分析
2	你业余时间怎么度过？你喜欢什么电视节目？喜欢读哪些书籍？	将爱好与应聘的职位一起分析，试寻找共同点，判断对方今后对职业感兴趣的可能性
3	你一般什么时候休息？什么时候起床？	休息有规律者优先
4	你经常会玩很晚才休息吗？	能熬夜是精力充沛的表现，但若是经常“玩”得很晚则上进心不足

## 11. 专业知识水平及特长

序号	题目	面试要点参考
1	你认为自己最擅长的是什么	与应聘职位一起综合考察，寻求共同点
2	谈谈你对本专业现时发展情况的了解。你认为业界今后的发展如何？	时刻掌握专业最新资讯的有培养潜力
3	你有什么级别的专业资格证书和能力证明？你认为它们能证明你能应付工作中的什么具体问题？	对本专业的深度理解
4	你最近阅读、写作或发表了什么专业文章或书籍？有何收获？	一般侧重于阅读的收获

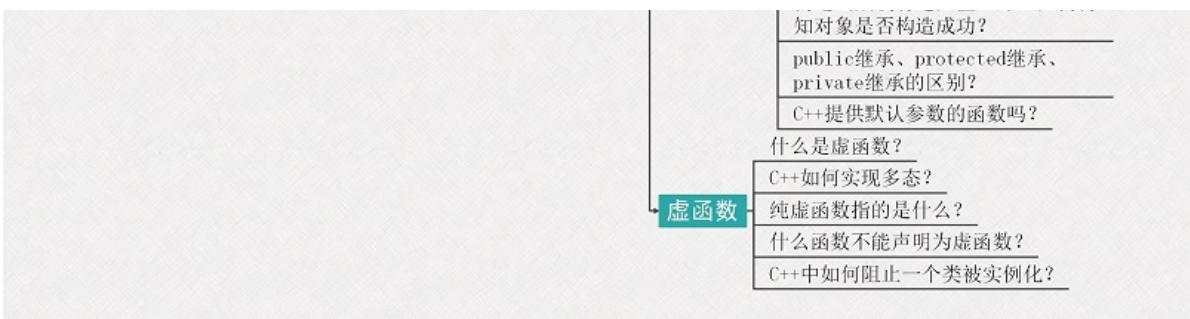
## 嵌入式知识点总结

### C/C++

---

	C语言宏中“#”和“##”的用法
	关键字volatile有什么含义？并举出三个不同的例子？
	关键字static的作用是什么？
	static全局变量与普通的全局变量有什么区别？
	static函数与普通函数有什么区别？
	在C语言中，为什么 static变量只初始化一次？
	extern “C”的作用是什么？
	const有什么作用？
	什么情况下使用const关键字？
	strcpy和strlen有什么区别？
	new/delete与malloc/free的区别是什么？
	strlen("\0") =? sizeof("\0")=?
	sizeof和strlen有什么区别？
	不使用 sizeof，如何求int占用的字节数？
	C语言中 struct与 union的区别是什么
	左值和右值是什么？
	什么是短路求值？
	++a和a++有什么区别？两者是如何实现的？
	C内存分配的方式有几种？
	堆与栈有什么区别？
	栈在C语言中有什么作用？
	C++函数栈空间的最大值是多少？
	C语言参数压栈顺序是怎样的？
→ 内存	C++如何处理返回值？
	C++中拷贝赋值函数的形参能否进行值传递？
	C++的内存管理是怎样的？
	什么是内存泄漏？
	如何判断内存泄漏？
	数组指针和指针数组有什么区别？
	函数指针和指针函数有什么区别？
	数组名和指针的区别与联系是什么？
	指针进行强制类型转换后与地址进行加法运算，结果是什么？
→ 指针	常量指针，指向常量的指针，指向常量的常量指针有什么区别？
	指针和引用的异同是什么？如何相互转换？
	野指针是什么？
	如何避免野指针？
	C++中的智能指针是什么？
	智能指针的内存泄漏如何解决？
	预处理器标识#error的目的是什么？
	定义常量谁更好？#define还是const？
	typedef和 define有什么区别？
	如何使用 define声明个常数，用以表明1年中有多少秒（忽略闰年问题）
→ 预处理	# include< filename.h>和#include "filename.h"有什么区别
	头文件的作用有哪些？
	在头文件中定义静态变量是否可行，为什么？
	写一个“标准”宏MIN，这个宏输入两个参数并返回较小的一个？
	不使用流程控制语句，如何打印出1~1000的整数？
	全局变量和静态变量的区别是什么？
	全局变量可不可以定义在可被多





## 关键字

### C语言宏中“#”和“##”的用法

#### 1. (#) 字符串化操作符

作用：将宏定义中的传入参数名转换成用一对双引号括起来参数名字符串。其只能用于有传入参数的宏定义中，且必须置于宏定义体中的参数名前。

如：

```
1 #define example( instr ) printf( "the input string is:\t%s\n", #instr )
2 #define example1( instr ) #instr当使用该宏定义时：
```

```
1 example( abc ); // 在编译时将会展开成: printf("the input string is:\t%s\n","abc")
2 string str = example1( abc ); // 将会展成: string str="abc"
```

#### 2. (##) 符号连接操作符

作用：将宏定义的多个形参转换成一个实际参数名。

如：

```
1 #define exampleNum( n ) num##n
```

使用：

```
1 int num9 = 9;
2 int num = exampleNum( 9 ); // 将会扩展成 int num = num9
```

注意：

a. 当用##连接形参时，##前后的空格可有可无。

如：

```
1 #define exampleNum( n )      num ## n
2 // 相当于 #define exampleNum( n )      num##n
```

b. 连接后的实际参数名，必须为实际存在的参数名或是编译器已知的宏定义。

c. 如果##后的参数本身也是一个宏的话，##会阻止这个宏的展开。

```
1 #include <stdio.h>
2 #include <string.h>
3
```

```

4 #define STRCPY(a, b)    strcpy(a ## _p, #b)
5 int main()
6 {
7     char var1_p[20];
8     char var2_p[30];
9     strcpy(var1_p, "aaaa");
10    strcpy(var2_p, "bbbb");
11    STRCPY(var1, var2);
12    STRCPY(var2, var1);
13    printf("var1 = %s\n", var1_p);
14    printf("var2 = %s\n", var2_p);
15
16 //STRCPY(STRCPY(var1,var2),var2);
17 //这里是否会展开为: strcpy(strcpy(var1_p,"var2")_p,"var2") ? 答案是否定的:
18 //展开结果将是: strcpy(STRCPY(var1,var2)_p,"var2")
19 //## 阻止了参数的宏展开!如果宏定义里没有用到 # 和 ##, 宏将会完全展开
20 // 把注释打开的话, 会报错:implicit declaration of function 'STRCPY'
21 return 0;
22 }

```

结果：

```

1 var1 = var2
2 var2 = var1

```

## 关键字volatile有什么含意？并举出三个不同的例子？

- 并行设备的硬件寄存器。** 存储器映射的硬件寄存器通常加volatile，因为寄存器随时可以被外设硬件修改。当声明指向设备寄存器的指针时一定要用volatile，它会告诉编译器不要对存储在这个地址的数据进行假设。
- 一个中断服务程序中修改的供其他程序检测的变量。** volatile提醒编译器，它后面所定义的变量随时都有可能改变。因此编译后的程序每次需要存储或读取这个变量的时候，**都会直接从变量地址中读取数据**。如果没有volatile关键字，则编译器可能优化读取和存储，可能暂时使用寄存器中的值，如果这个变量由别的程序更新了的话，将出现不一致的现象。
- 多线程应用中被几个任务共享的变量。** 单地说就是防止编译器对代码进行优化.比如如下程序：

```

1 XBYTE[2]=0x55;
2 XBYTE[2]=0x56;
3 XBYTE[2]=0x57;
4 XBYTE[2]=0x58;

```

对外部硬件而言，上述四条语句分别表示不同的操作，会产生四种不同的动作，但是编译器却会对上述四条语句进行优化，**认为只有XBYTE[2]=0x58（即忽略前三条语句，只产生一条机器代码）**。如果键入volatile，编译器会逐一的进行编译并产生相应的机器代码（产生四条代码）。

## 关键字static的作用是什么？

- 在函数体，**只会被初始化一次**，一个被声明为静态的变量在这一函数被调用过程中维持其值不变。
- 在模块内（但在函数体外），一个被声明为**静态的变量**可以被模块内所用函数访问，但不能被模块外其它函数访问。它是一个**本地的全局变量**（只能被当前文件使用）。
- 在模块内，一个被声明为**静态的函数**只可被这一模块内的其它函数调用。那就是，这个函数被限制在声明它的模块的本地范围内使用（**只能被当前文件使用**）。

## 在C语言中，为什么 static 变量只初始化一次？

对于所有的对象（不仅仅是静态对象），**初始化都只有一次**，而由于静态变量具有“记忆”功能，初始化后，一直都没有被销毁，**都会保存在内存区域中**，所以不会再次初始化。存放在静态区的变量的生命周期一般比较长，它与整个程序“同生死、共存亡”，所以它只需初始化一次。而 auto 变量，即自动变量，由于它**存放在栈区**，一旦函数调用结束，就会**立刻被销毁**。

## extern "C" 的作用是什么？

extern "C" 的主要作用就是为了能够正确实现 C++ 代码调用其他 C 语言代码。加上 extern "C" 后，会**指示编译器这部分代码按 C 语言的进行编译**，而不是 C++ 的。

## const 有什么作用？

1. 定义变量（局部变量或全局变量）为常量，例如：

```
1 const int N=100; //定义一个常量N
2 N=50; //错误，常量的值不能被修改
3 const int n; //错误，常量在定义的时候必须初始化
```

2. 修饰函数的参数，表示在函数体内不能修改这个参数的值。

3. 修饰函数的返回值。

a. 如果给用 const 修饰**返回值的类型为指针**，那么函数返回值（即指针）的内容是**不能被修改的**，而且这个返回值只能赋给被 const 修饰的指针。例如：

```
1 const char GetString() //定义一个函数
2 char *str= GetString() //错误，因为str没有被 const 修饰
3 const char *str=GetString() //正确
```

b. 如果用 const 修饰**普通的返回值**，如返回 int 变量，由于这个返回值是一个临时变量，在函数调用结束后这个临时变量的生命周期也就结束了，因此把**这些返回值修饰为 const 是没有意义的**。

4. 节省空间，避免不必要的内存分配。例如：

```
1 #define PI 3.14159 //该宏用来定义常量
2 const double Pi=3.14159 //此时并未将Pi放入只读存储器中
3 double i=Pi //此时为Pi分配内存，以后不再分配
4 double I=PI //编译期间进行宏替换，分配内存
5 double j=Pi //没有内存分配再次进行宏替换，又一次分配内存
```

## 什么情况下使用 const 关键字？

1. 修饰一般常量。一般常量是指简单类型的常量。这种常量在定义时，修饰符 const 可以用在类型说明符前，也可以用在类型说明符后。例如：

```
1 int const x=2; const int x=2
```

2. 修饰常数组。定义或说明一个常数组可以采用如下格式：

```
1 int const a[8]={1,2,3,4,5,6,7,8}
2 const int a[8]={1,2,3,4,5,6,7,8}
```

3. 修饰常对象。常对象是指对象常量，定义格式如下：

```

1 | class A:
2 | const A a;
3 | A const a;

```

定义常对象时，同样要进行初始化，并且该对象不能再被更新。修饰符 `const` 可以放在类名后面，也可以放在类名前面。

#### 4. 修饰常指针

```

1 | const int*p; //常量指针，指向常量的指针。即p指向的内存可以变，p指向的数值内容不可变
2 | int const*p; //同上
3 | int*const p; //指针常量，本质是一个常量，而用指针修饰它。即p指向的内存不可以变，但是p内存
    | 位置的数值可以变
4 | const int* const p; //指向常量的常量指针。即p指向的内存和数值都不可变

```

5. 修饰常引用。被 `const` 修饰的引用变量为常引用，一旦被初始化，就不能再指向其他对象了。

6. 修饰函数的常参数。`const` 修饰符也可以修饰函数的传递参数，格式如下：

```

1 | void Fun (const int Var)

```

告诉编译器 `Var` 在函数体中不能被改变，从而防止了使用者一些无意的或错误的修改。

7. 修饰函数的返回值。`const` 修饰符也可以修饰函数的返回值，表明该返回值不可被改变，格式如下：

```

1 | const int Fun1 ();
2 | const MyClass Fun2 ();

```

8. 在另一连接文件中引用 `const` 常量。使用方式有

```

1 | extern const int i;
2 | extern const int j=10;

```

## new/delete与malloc/free的区别是什么？

1. `new`、`delete` 是 C++ 中的操作符，而 `malloc` 和 `free` 是标准库函数。
2. 对于非内部数据对象来说，只使用 `malloc` 是无法完成动态对象要求的，一般在创建对象时需要调用构造函数，对象消亡时，自动的调用析构函数。而 `malloc free` 是库函数而不是运算符，不在编译器控制范围之内，不能够自动调用构造函数和析构函数。而 `new` 在为对象申请分配内存空间时，可以自动调用构造函数，同时也可以完成对对象的初始化。同理，`delete` 也可以自动调用析构函数。而 `malloc` 只是做一件事，只是为变量分配了内存，同理，`free` 也只是释放变量的内存。
3. `new` 返回的是指定类型的指针，并且可以自动计算所申请内存的大小。而 `malloc` 需要我们计算申请内存的大小，并且在返回时强行转换为实际类型的指针。

## `strlen("\0")=? sizeof("\0")=?`

`strlen("\0") = 0, sizeof("\0") = 2.`

`strlen` 用来计算字符串的长度（在 C/C++ 中，字符串是以 "\0" 作为结束符的），它从内存的某个位置（可以是字符串开头，中间某个位置，甚至是某个不确定的内存区域）开始扫描直到碰到第一个字符串结束符 \0 为止，然后返回计数器值。`sizeof` 是 C 语言的关键字，它以字节的形式给出了其操作数的存储大小，操作数可以是一个表达式或括在括号内的类型名，操作数的存储大小由操作数的类型决定。

## sizeof和strlen有什么区别？

strlen与 sizeof的差别表现在以下5个方面。

1. sizeof是运算符（是不是被弄糊涂了？事实上， sizeof既是关键字，也是运算符，但不是函数），而 strlen是函数。 sizeof后如果是类型，则必须加括弧，如果是变量名，则可以不加括弧。
2. sizeof运算符的结果类型是 size\_t，它在头文件中 typedef 为 unsigned int 类型。该类型保证能够容纳实现所建立的最大对象的字节大小
3. sizeof可以用类型作为参数， strlen只能用char\*作参数，而且必须是以“0结尾的。 sizeof还可以以函数作为参数，如int g ()，则 sizeof (g ()) 的值等于 sizeof (int)的值，在32位计算机下，该值为4。
4. 大部分编译程序的 sizeof都是在**编译**的时候计算的，所以可以通过 sizeof (x) 来定义数组维数。而 strlen则是在**运行期**计算的，用来计算字符串的实际长度，不是类型占内存的大小。例如， char str[20] = "0123456789"，字符数组str是**编译期**大小已经固定的数组，在32位机器下，为 sizeof (char) \*20=20，而其 strlen大小则是在**运行期**确定的，所以其值为字符串的实际长度10。
5. 当数组作为参数传给函数时，传递的是指针，而不是数组，即传递的是数组的首地址。

## 不使用 sizeof，如何求int占用的字节数？

```

1 #include <stdio.h>
2 #define MySizeof(value)  ((char *)(&value+1)-(char*)&value)
3
4 int main()
5 {
6     int i ;
7     double f;
8     double *q;
9     printf("%d\r\n",MySizeof(i));
10    printf("%d\r\n",MySizeof(f));
11    printf("%d\r\n",MySizeof(a));
12    printf("%d\r\n",MySizeof(q));
13    return 0;
14 }
```

输出为：

```
1 | 4 8 32 4
```

上例中，`(char*) & value` 返回 Value 的地址的第一个字节，`(char*) (& value+1)` 返回 value 的地址的下一个地址的第一个字节，所以它们之差为它所占的字节数。

## C语言中 struct与 union的区别是什么？

struct（结构体）与 union（联合体）是C语言中两种不同的数据结构，两者都是常见的复合结构，其区别主要表现在以下两个方面。

1. 结构体与联合体虽然都是由多个不同的数据类型成员组成的，但不同之处在于联合体中所有成员**共用一块地址空间**，即联合体只存放了一个被选中的成员，而结构体中所有成员占用空间是累加的，其所有成员都存在，不同成员会存放在不同的地址。在计算一个结构型变量的总长度时，其内存空间大小等于所有成员长度之和（需要考虑字节对齐），而在联合体中，所有成员不能同时占用内存空间，它们不能同时存在，所以一个联合型变量的长度等于其最长的成员的长度。
2. 对于联合体的不同成员赋值，**将会对它的其他成员重写**，原来成员的值就不存在了，而对结构体的不同成员赋值是互不影响的。

举个例子。下列代码执行结果是多少？

```

1 | typedef union {double i; int k[5]; char c;}DATE;
2 | typedef struct data(int cat; DATE cow;double dog;) too;
3 | DATE max;
4 | printf ("%d", sizeof(too)+sizeof(max));

```

假设为32位机器，int型占4个字节，double型占8个字节，char型占1个字节，而DATE是一个联合型变量，联合型变量共用空间，union里面最大的变量类型是int[5]，所以占用20个字节，它的大小是20，而由于union中double占了8个字节，因此union是要8个字节对齐，所占内存空间为8的倍数。为了实现8个字节对齐，所占空间为24.而data是一个结构体变量，每个变量分开占用空间，依次为 sizeof (int) + sizeof (DATE) + sizeof ( double) =4+24+8=36按照8字节对齐，占用空间为40，所以结果为40+24=64。

## 左值和右值是什么？

左值是指可以出现在等号左边的变量或表达式，它最重要的特点就是可写（可寻址）。也就是说，它的值可以被修改，如果一个变量或表达式的值不能被修改，那么它就不能作为左值。

右值是指只可以出现在等号右边的变量或表达式。它最重要的特点是可读。一般的使用场景都是把一个右值赋值给一个左值。

通常，左值可以作为右值，但是右值不一定是左值。

## 什么是短路求值？

```

1 | #include <stdio.h>
2 | int main()
3 | {
4 |     int i = 6;
5 |     int j = 1;
6 |     if(i>0 || (j++)>0);
7 |     printf("%D\r\n",j);
8 |     return 0;
9 | }

```

输出结果为1。

输出为什么不是2，而是1呢？其实，这里就涉及一个短路计算的问题。由于if语句是个条件判断语句，里面是有两个简单语句进行或运算组合的复合语句，因为或运算中，只要参与或运算的两个表达式的值都为真，则整个运算结果为真，而由于变量i的值为6，已经大于0了，而该语句已经为true，则不需要执行后续的j++操作来判断真假，所以后续的j++操作不需要执行，j的值仍然为1。

因为短路计算的问题，对于&&操作，由于在两个表达式的返回值中，如果有任何一个为假则整个表达式的值都为假，如果前一个语句的返回值为false，则无论后一个语句的返回值是真是假，整个条件判断都为假，不用执行后一个语句，而a>b的返回值为false，程序不执行表达式n=c>d，所以，n的值保持为初值2。

## ++a和a++有什么区别？两者是如何实现的？

a++的具体运算过程为

```

1 | int temp = a;
2 | a=a+1;
3 | return temp;

```

++a的具体运算过程为

```

1 | a=a+1;
2 | return a;

```

后置自增运算符需要把原来变量的值复制到一个**临时的存储空间**，等运算结束后才会返回这个临时变量的值。所以前置自增运算符效率比后置自增要高



## 内存

### C语言中内存分配的方式有几种？

#### 1. 静态存储区分配

内存分配在程序编译之前完成，且在程序的整个运行期间都存在，例如全局变量、静态变量等。

#### 2. 栈上分配

在函数执行时，函数内的局部变量的存储单元在栈上创建，函数执行结束时这些存储单元自动释放。

#### 3. 堆上分配

### 堆与栈有什么区别？

#### 1. 申请方式

栈的空间由操作系统自动分配/释放，堆上的空间手动分配/释放。

#### 2. 申请大小的限制

栈空间有限。在Windows下，栈是向**低地址**扩展的数据结构，是一块**连续的内存的区域**。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在WINDOWS下，栈的大小是2M（也有的说是1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示**overflow**。因此，能从**栈获得的空间较小**

堆是很大的自由存储区。堆是向**高地址**扩展的数据结构，是不连续的内存区域。这是由于系统是用**链表来存储的空闲内存地址的**，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小**受限于计算机系统中有效的虚拟内存**。由此可见，堆获得的空间比较灵活，也比较大。

#### 3. 申请效率

栈由系统自动分配，速度较快。但程序员是无法控制的。

堆是由new分配的内存，一般速度比较慢，而且容易产生内存碎片，不过用起来最方便。

### 栈在C语言中有什么作用？

1. C语言中栈用来存储临时变量，临时变量包括**函数参数和函数内部定义的临时变量**。函数调用中和函数调用相关的**函数返回地址**，函数中的临时变量，寄存器等均保存在栈中，函数调动返回后从栈中恢复寄存器和临时变量等函数运行场景。
2. 多线程编程的基础是栈，**栈是多线程编程的基石**，每一个线程都最少有一个自己专属的栈，用来存储本线程运行时各个函数的临时变量和维系函数调用和函数返回时的函数调用关系和函数运行场景。操作系统最基本的功能是支持多线程编程，支持中断和异常处理，每个线程都有专属的栈，中断和异常处理也具有专属的栈，栈是操作系统多线程管理的基石。

## C语言函数参数压栈顺序是怎样的？

从右至左。

C语言参数入栈顺序的好处就是可以动态变化参数个数。自左向右的入栈方式，最前面的参数被压在栈底。除非知道参数个数，否则是无法通过栈指针的相对位移求得最左边的参数。这样就变成了左边参数的个数不确定，正好和动态参数个数的方向相反。因此，C语言函数参数采用自右向左的入栈顺序，主要原因是支持可变长参数形式。

## C++如何处理返回值？

C++函数返回可以按值返回和按常量引用返回，偶尔也可以按引址返回。多数情况下不要使用引址返回。

## C++中拷贝赋值函数的形参能否进行值传递？

可以。但是调用拷贝赋值函数的时候，首先要将实参传递给形参，此时会多调用一次拷贝构造，并不会无限递归。

在C++ primer 第五版13.3节介绍了 copy and swap技术就是利用值传递和swap函数实现异常安全的拷贝赋值。以下是书中代码：

```

1 HasPtr& HasPtr::operator=(HasPtr rhs)
2 {
3     swap(*this, rhs);
4     return *this;
5 }
```

在拷贝赋值中只需要调用拷贝构造函数，并会产生无限的循环往复。

## C++的内存管理是怎样的？

在C++中，虚拟内存分为**代码段、数据段、BSS段、堆区、文件映射区以及栈区**六部分。

**代码段**：包括只读存储区和文本区，其中只读存储区存储字符串常量，文本区存储程序的机器代码。

**数据段**：存储程序中已初始化的全局变量和静态变量

**BSS段**：存储未初始化的全局变量和静态变量（局部+全局），以及所有被初始化为0的全局变量和静态变量。

**堆区**：调用new/malloc函数时在堆区动态分配内存，同时需要调用delete/free来手动释放申请的内存。

**映射区**：存储动态链接库以及调用mmap函数进行的文件映射

**栈**：使用栈空间存储函数的返回地址、参数、局部变量、返回值

## 什么是内存泄漏？

简单地说就是申请了一块内存空间，使用完毕后没有释放掉。

它的一般表现方式是程序运行时间越长，占用内存越多，最终用尽全部内存，整个系统崩溃。由程序申请的一块内存，且没有任何一个指针指向它，那么这块内存就泄露了。

## 如何判断内存泄漏？

1. 良好的编码习惯，尽量在涉及内存的程序段，检测出内存泄露。当程式稳定之后，在来检测内存泄露时，无疑增加了排除的困难和复杂度。使用了内存分配的函数，一旦使用完毕，要记得要使用其相应的函数释放掉。
2. 将分配的内存的指针以链表的形式自行管理，使用完毕之后从链表中删除，程序结束时可检查改链表。
3. Boost 中的smart pointer。
4. 一些常见的工具插件，如ccmalloc、Dmalloc、Leaky等等。



## 指针

### 数组指针和指针数组有什么区别？

数组指针就是指向数组的指针，它表示的是一个指针，这个指针指向的是一个数组，它的重点是指针。例如，`int (*pa)[8]` 声明了一个指针，该指针指向了一个有8个int型元素的数组。下面给出一个数组指针的示例。

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 void main()
4 {
5     int b[12]={1,2,3,4,5,6,7,8,9,10,11,12};
6     int (*p)[4];
7     p = b;
8     printf ("%d\n", **(++p));
9 }
```

程序的输出结果为 5。

上例中，p是一个数组指针，它指向一个**包含有4个int类型数组的指针**，刚开始p被初始化为指向数组b的首地址，`++p`相当于把p所指向的地址向后移动4个int所占用的空间，此时p指向数组{5,6,7,8}，语句`*(++p)`表示的是这个数组中**第一个元素**的地址(可以理解p为指向二维数组的指针，{1,2,3,4}，{5,6,7,8}，{9,10,11,12})。p指向的就是{1,2,3,4}的地址，`*p`就是指向元素，{1,2,3,4}，`**p`指向的就是1)，语句`** (++p)`会输出这个数组的第一个元素5。

指针数组表示的是一个数组，而数组中的元素是指针。下面给出另外一个指针数组的示例

```

1 #include <stdio.h>
2 int main()
3 {
4     int i;
5     int *p[4];
6     int a[4]={1,2,3,4};
7     p[0] = &a[0];
8     p[1] = &a[1];
```

```

9  p[2] = &a[2];
10 p[3] = &a[3];
11 for(i=0;i<4;i++)
12     printf ("%d", *p[i]);
13 printf("\n");
14 return 0;
15 }
```

程序的输出结果为1234。

## 函数指针和指针函数有什么区别？

### 1. 函数指针

如果在程序中定义了一个函数，那么在编译时系统就会为这个函数代码分配一段存储空间，这段存储空间的首地址称为这个**函数的地址**。而且函数名表示的就是这个地址。既然是地址我们就可以定义一个指针变量来存放，这个指针变量就叫作**函数指针变量**，简称**函数指针**。

```
1 | int(*p)(int, int);
```

这个语句就定义了一个指向函数的指针变量 p。首先它是一个指针变量，所以要有一个“\*”，即 `(*p)`；其次前面的 int 表示这个指针变量可以指向返回值类型为 int 型的函数；后面括号中的两个 int 表示这个指针变量可以指向有两个参数且都是 int 型的函数。所以合起来这个语句的意思就是：定义了一个指针变量 p，该指针变量可以指向返回值类型为 int 型，且有两个整型参数的函数。p 的类型为 `int(*)(int, int)`。

我们看到，函数指针的定义就是将“函数声明”中的“函数名”改成“（指针变量名）”。但是这里需要注意的是：“（指针变量名）”**两端的括号不能省略**，括号改变了运算符的优先级。如果省略了括号，就不是定义函数指针而是一个函数声明了，即声明了一个返回值类型为指针型的函数。

**最后需要注意的是，指向函数的指针变量没有 ++ 和 -- 运算。**

```

1 # include <stdio.h>
2 int Max(int, int); //函数声明
3 int main(void)
4 {
5     int(*p)(int, int); //定义一个函数指针
6     int a, b, c;
7     p = Max; //把函数Max赋给指针变量p，使p指向Max函数
8     printf("please enter a and b:");
9     scanf("%d%d", &a, &b);
10    c = (*p)(a, b); //通过函数指针调用Max函数
11    printf("a = %d\nb = %d\nmax = %d\n", a, b, c);
12    return 0;
13 }
14 int Max(int x, int y) //定义Max函数
15 {
16     int z;
17     if (x > y)
18     {
19         z = x;
20     }
21     else
22     {
23         z = y;
24     }
25 }
```

```

25     return z;
26 }
27

```

## 2. 指针函数

首先它是一个函数，只不过这个函数的返回值是一个地址值。函数返回值必须用同类型的指针变量来接受，也就是说，指针函数一定有“函数返回值”，而且，在主调函数中，函数返回值必须赋给同类型的指针变量。

```
类型名 *函数名(函数参数列表);
```

其中，后缀运算符括号“()”表示这是一个函数，其前缀运算符星号“\*”表示此函数为指针型函数，其函数值为指针，即它带回来的值的类型为指针，当调用这个函数后，将得到一个“指向返回值为...的指针（地址），“类型名”表示函数返回的指针指向的类型”。

“(函数参数列表)”中的括号为函数调用运算符，在调用语句中，即使函数不带参数，其参数表的一对括号也不能省略。其示例如下：

```
1 | int *pfun(int, int);
```

由于“\*”的优先级低于“()”的优先级，因而pfun首先和后面的“()”结合，也就意味着，pfun是一个函数。即：

```
1 | int *(pfun(int, int));
```

接着再和前面的“\*”结合，说明这个函数的返回值是一个指针。由于前面还有一个int，也就是说，pfun是一个返回值为整型指针的函数。

```

1 | #include <stdio.h>
2 | float *find(float*pionter)[4],int n;//函数声明
3 | int main(void)
4 |
5 | {
6 |     static float score[][4]={{60,70,80,90},{56,89,34,45},{34,23,56,45}};
7 |     float *p;
8 |     int i,m;
9 |     printf("Enter the number to be found:");
10 |    scanf("%d",&m);
11 |    printf("the score of NO.%d are:\n",m);
12 |    p=find(score,m-1);
13 |    for(i=0;i<4;i++)
14 |        printf("%5.2f\t",*(p+i));
15 |
16 |    return 0;
17 |
18 | float *find(float(*pionter)[4],int n)/*定义指针函数*/
19 |
20 | {
21 |     float *pt;
22 |     pt=*(pionter+n);
23 |     return(pt);
24 |

```

共有三个学生的成绩，函数find()被定义为指针函数，其形参pointer是指针指向包含4个元素的一维数组的指针变量。pointer+n指向score的第n+1行。\*(pointer+1)指向第一行的第0个元素。pt是一个指针变量，它指向浮点型变量。main()函数中调用find()函数，将score数组的首地址传给pointer。

## 数组名和指针的区别与联系是什么？

### 1. 数据保存方面

指针保存的是地址（保存目标数据地址，自身地址由编译器分配），内存访问偏移量为4个字节，无论其中保存的是何种数据均已地址类型进行解析。

数组保存的数据。数组名表示的是第一个元素的地址，内存偏移量是保存数据类型的内存偏移量；只有对数组名取地址（&数组名）时数组名才表示整个数组，**内存偏移量是整个数组的大小（sizeof(数组名)）**。

### 2. 数据访问方面

指针对数据的访问方式是间接访问，需要用到解引用符号（\*数组名）。

数组对数据的访问则是直接访问，可通过下标访问或数组名+元素偏移量的方式

### 3. 使用环境

指针多用于动态数据结构（如链表，等等）和动态内存开辟。

数组多用于存储固定个数且类型统一的数据结构（如线性表等等）和隐式分配。

## 指针进行强制类型转换后与地址进行加法运算，结果是什么？

假设在32位机器上，在对齐为4的情况下,sizeof(long)的结果为4字节，sizeof(char\*)的结果为4字节，sizeof(short int)的结果与 sizeof(short)的结果都为2字节， sizeof(char)的结果为1字节， sizeof(int)的结果为4字节，由于32位机器上是4字节对齐，以如下结构体为例：

```

1 struct BBB
2
3 {
4
5     long num;
6
7     char *name;
8
9     short int data;
10
11    char ha;
12
13    short ba[5];
14
15 }*p;

```

当 p=0x100000; 则 p+0x200=? (ulong)p+0x200=? (char\*)p+0x200=?

其实在32位机器下,sizeof(struct BBB)=sizeof(\*p)=4+4+2+2+1+3/\*补齐\*/+2\*5+2/\*补齐\*/=24字节，而 p=0x100000,那么 p+0x200=0x1000000+0x200\*24 指针加法，加出来的是指针所指类型的字节长度的整倍数，就是p偏移sizeof(p)\*0x200。

(ulong)p+0x200=0x10000010+0x200 经过ulong后,已经不再是指针加法，而变成一个数值加法了。

(char\*)p+0x200=0x1000000+0x200\*sizeof(char) 结果类型是char\*。

## 指针常量，常量指针，指向常量的常量指针有什么区别？

### 1. 指针常量

```
1 | int * const p
```

先看const再看\*，p是一个常量类型的指针，**不能修改这个指针的指向**，但是这个指针所指向的地址上存储的**值可以修改**。

### 2. 常量指针

```
1 | const int *p
2 | int const *p
```

先看\*再看const，定义一个指针指向一个常量，不能通过指针来修改这个指针**指向的值**

### 3. 指向常量的常量指针

```
1 | const int *const p
```

对于“指向常量的常量指针”，就必须同时满足上述1和2中的内容，**既不可以修改指针的值，也不可以修改指针指向的值**。

## 指针和引用的异同是什么？如何相互转换？

### 相同

1. 都是地址的概念，指针指向某一内存、它的内容是所指内存的地址；引用则是某块内存的别名。
2. 从内存分配上看：两者都占内存，程序为指针会分配内存，一般是4个字节；而引用的本质是指针常量，指向对象不能变，但指向对象的值可以变。两者都是地址概念，所以本身都会占用内存。

### 区别

1. 指针是实体，而引用是别名。
2. 指针和引用的自增（++）运算符意义不同，指针是对内存地址自增，而引用是对值的自增。
3. 引用使用时无需解引用(\*)，指针需要解引用；（关于解引用大家可以看看这篇博客，传送门）。
4. 引用只能在定义时被初始化一次，之后不可变；指针可变。
5. 引用不能为空，指针可以为空。
6. “sizeof 引用”得到的是所指向的变量(对象)的大小，而“sizeof 指针”得到的是指针本身的大小，在32位系统指针变量一般占用4字节内存。

```
1 | #include "stdio.h"
2 |
3 | int main(){
4 |     int x = 5;
5 |     int *p = &x;
6 |     int &q = x;
7 |
8 |     printf("%d %d\n", *p, sizeof(p));
9 |     printf("%d %d\n", q, sizeof(q));
10 }
```

```
1 | //结果
2 | 5 8
3 | 5 4
```

由结果可知，引用使用时无需解引用(\*)，指针需要解引用；我用的是64位操作系统，“sizeof 指针”得到的是指针本身的大小，及8个字节。而“sizeof 引用”得到的是的对象本身的大小及int的大小，4个字节。

## 转换

1. **指针转引用**：把指针用\*就可以转换成对象，可以用在引用参数当中。
2. **引用转指针**：把引用类型的对象用&取地址就获得指针了。

```

1 int a = 5;
2 int *p = &a;
3 void fun(int &x){} //此时调用fun可使用： fun (*p);
4 //p是指针，加个*号后可以转换成该指针指向的对象，此时fun的形参是一个引用值，
5 //p指针指向的对象会转换成引用x。

```

## 野指针是什么？

1. 野指针是指向不可用内存的指针，当指针被创建时，指针不可能自动指向NULL，这时，默认值是随机的，此时的指针成为野指针。
2. 当指针被free或delete释放掉时，如果没有把指针设置为NULL，则会产生野指针，因为释放掉的仅仅是手指向的内存，并没有把指针本身释放掉。
3. 第三个造成野指针的原因是指针操作超越了变量的作用范围。

## 如何避免野指针？

1. 对指针进行初始化。

```

1 //将指针初始化为NULL。
2 char * p = NULL;
3 //用malloc分配内存
4 char * p = (char *)malloc(sizeof(char));
5 //用已有合法的可访问的内存地址对指针初始化
6 char num[ 30 ] = {0};
7 char *p = num;

```

2. 指针用完后释放内存，将指针赋NULL。

```

1 delete(p);
2 p = NULL;

```

注：malloc函数分配完内存后需注意：

- a. 检查是否分配成功（若分配成功，返回内存的首地址；分配不成功，返回NULL。可以通过if语句来判断）
- b. 清空内存中的数据（malloc分配的空间里可能存在垃圾值，用memset或bzero 函数清空内存）

```

1 //s是 需要置零的空间的起始地址； n是 要置零的数据字节个数。
2 void bzero (void *s, int n) ;
3 // 如果要清空空间的首地址为p, value为值, size为字节数。
4 void memset(void *start, int value, int size);

```

## C++中的智能指针是什么？

智能指针是一个类，用来存储指针（指向动态分配对象的指针）。

C++程序设计中使用堆内存是非常频繁的操作，堆内存的申请和释放都由程序员自己管理。程序员自己管理堆内存可以提高了程序的效率，但是整体来说堆内存的管理是麻烦的，C++11中引入了智能指针的概念，方便管理堆内存。使用普通指针，容易造成堆内存泄露（忘记释放），二次释放，程序发生异常时内存泄露等问题等，使用智能指针能更好的管理堆内存。

## 智能指针的内存泄漏如何解决？

为了解决循环引用导致的内存泄漏，引入了弱指针 `weak_ptr`，`weak_ptr` 的构造函数不会修改引用计数的值，从而不会对对象的内存进行管理，其类似一个普通指针，但是不会指向引用计数的共享内存，但是可以检测到所管理的对象是否已经被释放，从而避免非法访问。



## 预处理

预处理器标识#error的目的是什么？

#error预处理指令的作用是，编译程序时，只要遇到#error就会生成一个**编译错误提示消息，并停止编译**。其语法格式为：#error error-message。

下面举个例子：

程序中往往有很多的预处理指令

```
1 #ifdef XXX
2 ...
3 #else
4 #endif
```

当程序比较大时，往往有些宏定义是在外部指定的（如makefile），或是在系统头文件中指定的，当你不太确定当前是否定义了XXX时，就可以改成如下这样进行编译：

```
1 #ifdef XXX
2 ...
3 #error "XXX has been defined"
4 #else
5 #endif
```

这样，如果编译时出现错误，输出了XXX has been defined，表明宏XXX已经被定义了。

定义常量谁更好？#define还是const？

尺有所短，寸有所长，define与const都能定义常量，效果虽然一样，但是各有侧重。

define既可以替代常数值，又可以替代表达式，甚至是代码段，但是容易出错，而 const的引入可以增强程序的可读性，它使程序的维护与调试变得更加方便。具体而言，它们的差异主要表现在以下3个方面。

1. define只是用来进行**单纯的文本替换**， define常量的**生命周期止于编译期，不分配内存空间**，它存在于程序的**代码段**，在实际程序中，它只是一个常数；而 const常量存在于程序的**数据段**，并在**堆栈中分配了空间**， const常量在程序中确确实实存在，并且可以被调用、传递
2. const常量有数据类型，而 define常量没有数据类型。编译器可以对 const常量进行类型安全检查，如类型、语句结构等，而 define不行。
3. 很多IDE**支持调试** const定义的常量，而不支持 define定义的常量由于 const修饰的变量可以排除程序之间的不安全性因素，保护程序中的常量不被修改，而且对数据类型也会进行相应的检查，极大地提高了程序的**健壮性**，所以一般**更加倾向于用const来定义常量类型**。

## typedef和 define有什么区别？

typedef与 define都是**替一个对象取一个别名**，以此来增强程序的可读性，但是它们在使用和作用上也存在着以下4个方面的不同。

### 1. 原理不同

#define是C语言中定义的语法，它是预处理指令，在预处理时进行简单而机械的字符串替换，**不做正确性检查**，不管含义是否正确照样代入，只有在编译已被展开的源程序时，才会发现可能的错误并报错。

例如，`# define PI 3.1415926`，当程序执行 `area=PI*r` 语句时，PI会被替换为3.1415926。于是该语句被替换为 `area=3.1415926*r*r`。如果把# define语句中的数字9写成了g，预处理也照样代入，而不去检查其是否合理、合法。

typedef是关键字，它在编译时处理，所以 typedef具有类型检查的功能。它在自己的作用域内给一个已经存在的类型一个别名，但是不能在一个函数定义里面使用标识符 typedef。例如，`typedef int INTEGER`，这以后就可用 INTEGER来代替int作整型变量的类型说明了，例如：`INTEGER a,b;`

用 typedef定义数组、指针、结构等类型将带来很大的方便，不仅使程序书写简单而且使意义更为明确，因而增强了可读性。例如：`typedef int a[10];`

表示a是整型数组类型，数组长度为10。然后就可用a说明变量，例如：语句 `a s1,s2;` 完全等效于语句 `int s1[10],s2[10].` 同理，`typedef void (*p) (void)` 表示p是一种指向void型的指针类型。

### 2. 功能不同

typedef用来定义类型的别名，这些类型不仅包含内部类型（int、char等），还包括自定义类型（如 struct），可以起到使类型易于记忆的功能。

例如：`typedef int (*PF)(const char *, const char*)`

定义一个指向函数的指针的数据类型PF，其中函数返回值为int，参数为 const char\*。typedef还有另外一个重要的用途，那就是定义机器无关的类型。例如，可以定义一个叫REAL的浮点类型，在目标机器上它可以获得最高的精度：`typedef long double REAL`，在不支持 long double的机器上，该 typedef看起来会是下面这样：`typedef double real`，在 double都不支持的机器上，该 typedef看起来会是这样：`typedef float REAL`。

#define不只是可以为类型取别名，还可以定义常量、变量、编译开关等。

### 3. 作用域不同

#define没有作用域的限制，只要是之前预定义过的宏，在以后的程序中都可以使用，而 typedef有自己的作用域。

程序示例如下：

```

1 void fun()
2 {
3     #define A int
4 }
5 void gun()
6 {
7     //这里也可以使用A，因为宏替换没有作用域，但如果上面用的是 typedef，那这里就不能用
8     //A，不过，一般不在函数内使用 typedef
9 }
10

```

#### 4. 对指针的操作不同

两者修饰指针类型时，作用不同。

```

1 #define INTPTR1 int*
2 typedef int* INTPTR2;
3 INTPTR1 p1, p2;
4 INTPTR2 p3, p4;

```

INTPTR1 p1, p2和INTPTR2 p3, p4的效果截然不同。INTPTR1 p1, p2进行字符串替换后变成  
`int*p1,p2`，要表达的意义是声明一个指针变量p1和一个整型变量p2。而INTPTR2 p3, p4，由于  
INTPTR2是具有含义的，告诉我们是一个指向整型数据的指针，那么p3和p4都为指针变量，这句相当于  
`int*p1, *p2`。从这里可以看出，进行宏替换是不含任何意义的替换，仅仅为字符串替换；而用**typedef**  
为一种数据类型起的别名是带有一定含义的。

程序示例如下

```

1 #define INTPTR1 int*
2 typedef int* INTPTR2
3 int a=1;
4 int b=2;
5 int c=3;
6 const INTPTR1 p1=&a;
7 const INTPTR2 p2=&b;
8 INTPTR2 const p3=&c;

```

上述代码中，`const INTPTR1 p1`表示p1是一个常量指针，即不可以通过p1去修改p1指向的内容，但是  
p1可以指向其他内容。而对于`const INTPTR2 p2`，由于INTPTR2表示的是个指针类型，因此用`const`去  
限定，表示封锁了这个指针类型，因此p2是一个指针常量，不可使p2再指向其他内容，但可以通过p2修  
改其当前指向的内容。`INTPTR2 const p3`同样声明的是一个指针常量。

### 如何使用 **define** 声明个常数，用以表明1年中有多少秒（忽略闰年问题）

```
1 #define SECOND_PER_YEAR (60*60*24*365UL)
```

### # include<filename.h>和# include"filename.h"有什么区别？

对于`#include<filename.h>`，编译器先从标准库路径开始搜索`filename.h`，使得系统文件调用较快。而  
对于`#include"filename.h"`，编译器先从用户的工作路径开始搜索`filename.h`，然后去寻找系统路  
径，使得自定义文件较快。

## 头文件的作用有哪些？

头文件的作用主要表现为以下两个方面：

1. 通过头文件来调用库功能。出于对源代码保密的考虑，源代码不便（或不准）向用户公布，只要向用户提供头文件和二进制的库即可。用户只需要按照头文件中的接口声明来调用库功能，而不必关心接口是怎么实现的。编译器会从库中提取相应的代码。
2. 头文件能加强类型安全检查。当某个接口被实现或被使用时，其方式与头文件中的声明不一致，编译器就会指出错误，大大减轻程序员调试、改错的负担。

## 在头文件中定义静态变量是否可行，为什么？

不可行，如果在头文件中定义静态变量，会造成资源浪费的问题，同时也可能引起程序错误。因为如果在使用了该头文件的每个C语言文件中定义静态变量，按照编译的步骤，在每个头文件中都会单独存在一个静态变量，从而会引起空间浪费或者程序错误所以，不推荐在头文件中定义任何变量，当然也包括静态变量。

## 写一个"标准"宏MIN，这个宏输入两个参数并返回较小的一个？

```
1 | #define MIN(A,B) ((A) <= (B) ? (A) : (B))
```

## 不使用流程控制语句，如何打印出1~1000的整数？

宏定义多层嵌套 (10 \* 10 \* 10)，printf多次输出。

```
1 | #include <stdio.h>
2 | #define B P,P,P,P,P,P,P,P,P
3 | #define P L,L,L,L,L,L,L,L,L
4 | #define L I,I,I,I,I,I,I,I,I,N
5 | #define I printf ("%3d", i++)
6 | #define N printf ("\n")
7 | int main()
8 | {
9 |     int i = 1;
10 |     B;
11 |     return 0;
12 | }
```

简便写法,同样使用多层嵌套

```
1 | #include<stdio.h>
2 | #define A(x) x;x;x;x;x;x;x;x;x;
3 | int main()
4 | {
5 |     int n=1;
6 |     A(A(A printf("%d", n++));
7 |     return 0;
8 | }
```



## 变量

### 全局变量和局部变量的区别是什么？

1. 全局变量的作用域为程序块，而局部变量的作用域为当前函数。
2. 内存存储方式不同，全局变量（静态全局变量，静态局部变量）分配在全局数据区（静态存储空间），后者分配在栈区。
3. 生命周期不同。全局变量随主程序创建而创建，随主程序销毁而销毁，局部变量在局部函数内部，甚至局部循环体等内部存在，退出就不存在了。
4. 使用方式不同。通过声明为全局变量，程序的各个部分都可以用到，而局部变量只能在局部使用。

### 全局变量可不可以定义在可被多个.C文件包含的头文件中？为什么？

可以，在不同的C文件中以static形式来声明同名全局变量。

可以在不同的C文件中声明同名的全局变量，前提是其中只能有一个C文件中对此变量赋初值，此时连接不会出错。

### 局部变量能否和全局变量重名？

能，局部会屏蔽全局。

局部变量可以与全局变量同名，在函数内引用这个变量时，会用到同名的局部变量，而不会用到全局变量。

对于有些编译器而言，在同一个函数内可以定义多个同名的局部变量，比如在两个循环体内都定义一个同名的局部变量，而那个局部变量的作用域就在那个循环体内。



## 函数

### 请写个函数在main函数执行前先运行

**attribute**可以设置函数属性（Function Attribute）、变量属性（Variable Attribute）和类型属性（Type Attribute）。

gnu对于函数属性主要设置的关键字如下：

```

1      aligned:    设置函数对齐方式。
2      always_inline/gnu_inline:
3          函数是否是内联函数。
4      constructor/destructor:
5          主函数执行之前、之后执行的函数。
6      format:
7          指定变参函数的格式输入字符串所在函数位置以及对应格式输出的位置。
8      noreturn:
9          指定这个函数没有返回值。
10         请注意，这里的没有返回值，并不是返回值是void。而是像
11         _exit/exit/abord那样
12         执行完函数之后进程就结束的函数。
13         weak: 指定函数属性为弱属性，而不是全局属性，一旦全局函数名称和指定的函数名称
14         命名有冲突，使用全局函数名称。

```

完整示例代码如下：

```

1 #include <stdio.h>
2
3 void before() __attribute__((constructor));
4 void after() __attribute__((destructor));
5
6 void before() {
7     printf("this is function %s\n",__func__);
8     return;
9 }
10
11 void after(){
12     printf("this is function %s\n",__func__);
13     return;
14 }
15
16 int main(){
17     printf("this is function %s\n",__func__);
18     return 0;
19 }
20
21 // 输出结果
22 // this is function before
23 // this is function main
24 // this is function after

```

## 为什么析构函数必须是虚函数？

将可能会被继承的父类的析构函数设置为虚函数，可以保证当我们new一个子类，然后使用基类指针指向该子类对象，释放基类指针时可以释放掉子类的空间，防止内存泄漏。

## 为什么C++默认的析构函数不是虚函数？

C++默认的析构函数不是虚函数是因为虚函数需要额外的虚函数表和虚表指针，占用额外的内存。而对于不会被继承的类来说，其析构函数如果是虚函数，就会浪费内存。因此C++默认的析构函数不是虚函数，而是只有当需要当作父类时，设置为虚函数。

## C++中析构函数的作用？

如果构造函数打开了一个文件，最后不需要使用时文件就要被关闭。析构函数允许类自动完成类似清理工作，不必调用其他成员函数。

析构函数也是特殊的类成员函数。简单来说，析构函数与构造函数的作用正好相反，它用来完成对象被删除前的一些清理工作，也就是专门的扫尾工作。

## 静态函数和虚函数的区别？

静态函数在编译的时候就已经确定运行时机，虚函数在运行的时候动态绑定。虚函数因为用了虚函数表机制，调用的时候会增加一次内存开销。

## 重载和覆盖有什么区别？

1. 覆盖是子类和父类之间的关系，垂直关系；重载同一个类之间方法之间的关系，是水平关系。
2. 覆盖只能由一个方法或者只能由一对方法产生关系；重载是多个方法之间的关系。
3. 覆盖是根据对象类型（对象对应存储空间类型）来决定的；而重载关系是根据调用的实参表和形参表来选择方法体的。

## 虚函数表具体是怎样实现运行时多态的？

### 原理：

虚函数表是一个类的虚函数的地址表，每个对象在创建时，都会有一个指针指向该类虚函数表，每一个类的虚函数表，按照函数声明的顺序，会将函数地址存在虚函数表中，当子类对象重写父类的虚函数的时候，父类的虚函数表中对应的位置会被子类的虚函数地址覆盖。

### 作用：

在用父类的指针调用子类对象成员函数时，虚函数表会指明要调用的具体函数是哪个。

## C语言是怎么进行函数调用的？

大多数CPU上的程序实现**使用栈来支持函数调用操作**，栈被用来传递函数参数、存储返回信息、临时保存寄存器原有的值以备恢复以及用来存储局部变量。

函数调用操作所使用的栈部分叫做**栈帧结构**，每个函数调用都有属于自己的栈帧结构，栈帧结构由两个指针指定，帧指针（指向起始），栈指针（指向栈顶），函数对大多数数据的访问都是基于帧指针。下面是结构图：

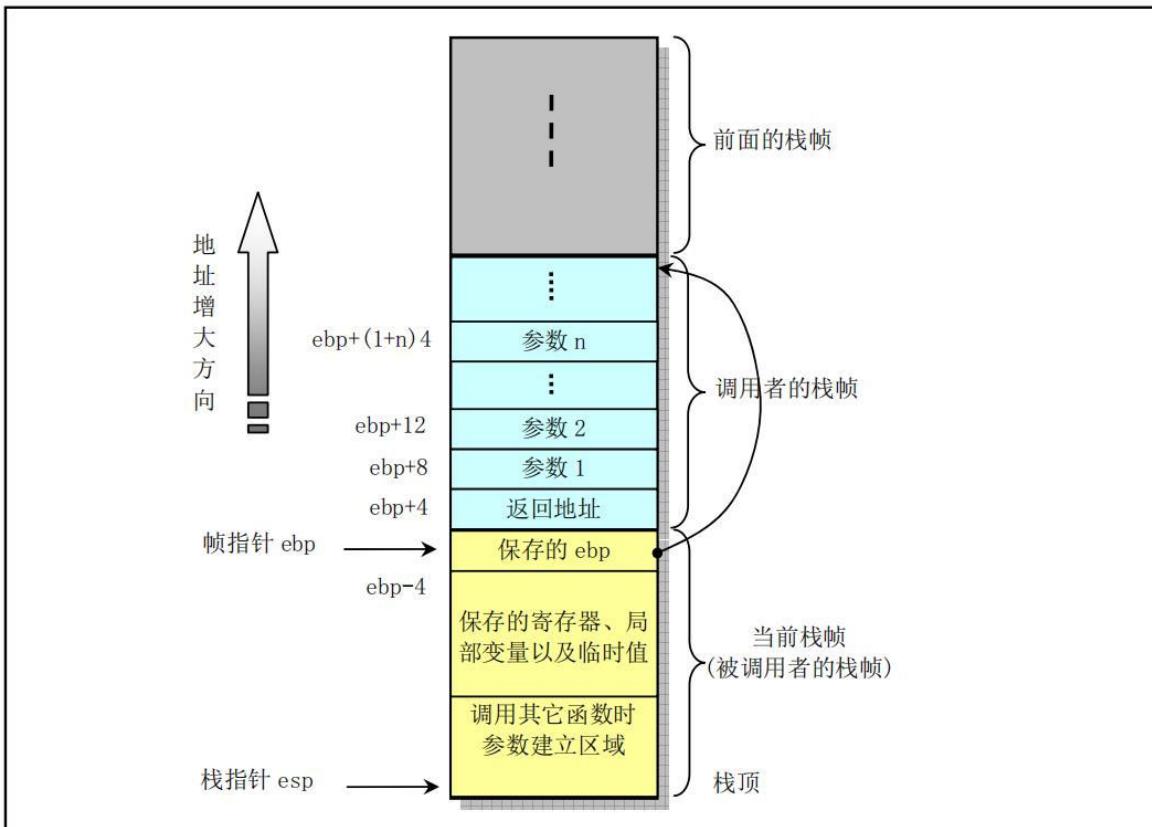


图 3-4 栈中帧结构示意图

[https://blog.csdn.net/weixin\\_42462202](https://blog.csdn.net/weixin_42462202)

栈指针和帧指针一般都有专门的寄存器，通常使用ebp寄存器作为帧指针，使用esp寄存器做栈指针。

**帧指针指向栈帧结构的头，存放着上一个栈帧的头部地址，栈指针指向栈顶。**

## 请你说一说select

### 1. select函数原型

```
1 | int select(int maxfdp, fd_set *readfds, fd_set *writefds, fd_set
   *errorfds, struct timeval *timeout);
```

### 2. 文件描述符的数量

单个进程能够监视的文件描述符的数量存在最大限制，通常是1024，当然可以更改数量；(在linux内核头文件中定义：#define \_FD\_SETSIZE 1024)

### 3. 就绪fd采用轮询的方式扫描

select返回的是int，可以理解为返回的是ready(准备好的)一个或者多个文件描述符，应用程序需要遍历整个文件描述符数组才能发现哪些fd句柄发生了事件，由于select采用轮询的方式扫描文件描述符（不知道那个文件描述符读写数据，所以需要把所有的fd都遍历），文件描述符数量越多，性能越差

### 4. 内核 / 用户空间内存拷贝

select每次都会改变内核中的句柄数据结构集(fd集合)，因而每次调用select都需要从用户空间向内核空间复制所有的句柄数据结构(fd集合)，产生巨大的开销

### 5. select的触发方式

select的触发方式是水平触发，应用程序如果没有完成对一个已经就绪的文件描述符进行IO操作，那么之后每次调用select还是会将这些文件描述符通知进程。

### 6. 优点

- a. select的可移植性较好，可以跨平台；
- b. select可设置的监听时间timeout精度更好，可精确到微秒，而poll为毫秒。

### 7. 缺点：

- a. select支持的文件描述符数量上限为1024，不能根据用户需求进行更改；
- b. select每次调用时都要将文件描述符集合从用户态拷贝到内核态，开销较大；
- c. select返回的就绪文件描述符集合，需要用户循环遍历所监听的所有文件描述符是否在该集合中，当监听描述符数量很大时效率较低。

## 请你说说fork,wait,exec函数

父进程产生子进程使用fork拷贝出来一个父进程的副本，此时只拷贝了父进程的页表，两个进程都读同一块内存，当有进程写的时候使用写实拷贝机制分配内存，exec函数可以加载一个elf文件去替换父进程，从此父进程和子进程就可以运行不同的程序了。fork从父进程返回子进程的pid，从子进程返回0.调用了wait的父进程将会发生阻塞，直到有子进程状态改变,执行成功返回0，错误返回-1。exec执行成功则子进程从新的程序开始运行，无返回值，执行失败返回-1。

## 数组

### 以下代码表示什么意思？

```
1 | *(a[1]+1)、*(&a[1][1])、(*(a+1))[1]
```

第一个：因为a[1]是第2行的地址，a[1]+1偏移一个单位（得到第2行第2列的地址），然后解引用取值，得到a[1][1]；

第二个：[]优先级高，a[1][1]取地址再取值。

第三个：a+1相当于&a[1]，所以\*(a+1)=a[1]，因此\*(a+1)[1]=a[1][1]

### 数组下标可以为负数吗？

可以，因为下标只是给出了一个与当前地址的偏移量而已，只要根据这个偏移量能定位得到目标地址即可。下面给出一个下标为负数的示例：

数组下标取负值的情况：

```
1 | #include <stdio.h>
2 | int main()
3 | {
4 |     int i;
5 |     int a[5]={0,1,2,3,4};
6 |     int *p=&a[4];
7 |     for (i=-4; i<=0; i++)
8 |         printf("%d %x\n", p[i], &p[i]);
9 |     return 0.
10 |
11 //输出结果为
12 //0 b3ecf480
13 //1 b3ecf484
14 //2 b3ecf488
15 //3 b3ecf48c
16 //4 b3ecf490
```

从上例可以发现，在C语言中，数组的下标并非不可以为负数，当数组下标为负数时，编译可以通过，而且也可以得到正确的结果，只是它表示的意思却是从当前地址向前寻址。



## 位操作

### 如何求解整型数的二进制表示中1的个数？

程序代码如下：

```

1 #include <stdio.h>
2 int func(int x)
3 {
4     int countx = 0;
5     while(x)
6     {
7         countx++;
8         x = x&(x-1);
9     }
10    return countx;
11 }
12 int main()
13 {
14     printf("%d\n", func(9999));
15     return 0;
16 }
```

程序输出的结果为8。

在上例中，函数func()的功能是将x转化为二进制数，然后计算该二进制数中含有的1的个数。首先以9为例来分析，9的二进制表示为1001,8的二进制表示为1000，两者执行&操作之后结果为1000，此时1000再与0111（7的二进制位）执行&操作之后结果为0。

为了理解这个算法的核心，需要理解以下两个操作：

- 1) 当一个数被减1时，它最右边的那个值为1的bit将变为0，同时其右边的所有的bit都会变成1。
- 2) 每次执行x&(x-1)的作用是把x对应的二进制数中的最后一位1去掉。因此，循环执行这个操作直到x等于0的时候，循环的次数就是x对应的二进制数中1的个数。

### 如何求解二进制中0的个数

图示分析(以25为例)：

十进制整数

25

二进制整数

0 1 1 0 0 1

第一次 :  $25 \% 2 = 1$      $rest = 25 >> 1 = 12$      $count++$ 第二次 :  $rest \% 2 = 0$      $rest1 = rest >> 1$ 

.....

第三十二次 :  $rest31 \% 2 = \dots$ 

( count 为计数初始值为 0 )

```

1 int CountZeroBit(int num)
2 {
3     int count = 0;
4
5     while (num + 1)
6     {
7         count++;
8         num |= (num + 1); // 算法转换
9     }
10    return count;
11 }
12
13 int main()
14 {
15     int value = 25;
16     int ret = CountZeroBit(value);
17     printf("%d 的二进制位中 0 的个数为 %d\n", value, ret);
18     system("pause");
19     return 0;
20 }
```

**交换两个变量的值，不使用第三个变量。即  $a=3, b=5$ , 交换之后  $a=5, b=3$ ;**有两种解法，一种用算术算法，一种用 $\wedge$ (异或)。

```

1 a = a + b;
2 b = a - b;
3 a = a - b;
```

```

1 a = a^b; // 只能对 int, char..
2 b = a^b;
3 a = a^b;
4 or
5 a ^= b ^= a;
```

给定一个整型变量a，写两段代码，第一个设置a的bit 3，第二个清除a 的bit 3。在以上两个操作中，要保持其它位不变。

```

1 #define BIT3 (0x1<<3)
2 static int a;
3 void set_bit3(void)
4 {
5     a |= BIT3;
6 }
7 void clear_bit3(void)
8 {
9     a &= ~BIT3;
10}

```



## 容器和算法

**map和set有什么区别？分别又是怎么实现的？**

map和set都是C++的关联容器，其底层实现都是红黑树（RB-Tree）。

由于 map 和set所开放的各种操作接口，RB-tree 也都提供了，所以几乎所有的 map 和set的操作行为，都只是转调 RB-tree 的操作行为。

**map和set的区别在于：**

map中的元素是**key-value**（键值对）对：**关键字起到索引的作用**，值则表示与索引相关联的数据；Set与之相对就是关键字的简单集合，**set中每个元素只包含一个关键字**。

set的迭代器是const的，不允许修改元素的值；map允许修改value，但不允许修改key。

其原因是因为map和set是根据关键字排序来保证其有序性的，如果允许修改key的话，那么首先需要删除该键，然后调节平衡，再插入修改后的键值，调节平衡，如此一来，严重破坏了map和set的结构，导致iterator失效，不知道应该指向改变前的位置，还是指向改变后的位置。所以**STL中将set的迭代器设置成const，不允许修改迭代器的值；而map的迭代器则不允许修改key值，允许修改value值**。

map支持下标操作，set不支持下标操作。

map可以用key做下标，map的下标运算符[ ]将关键码作为下标去执行查找，如果关键码不存在，则插入一个具有该关键码和mapped\_type类型默认值的元素至map中，因此**下标运算符[ ]在map应用中需要慎用**，const\_map不能用，只希望确定某一个关键值是否存在而不希望插入元素时也不应该使用，mapped\_type类型没有默认值也不应该使用。如果find能解决需要，尽可能用find。

## STL的allocator有什么作用？

STL的分配器用于封装STL容器在内存管理上的底层细节。在C++中，其内存配置和释放如下：

`new`运算分两个阶段：(1)调用`::operator new`配置内存;(2)调用对象构造函数构造对象内容

`delete`运算分两个阶段：(1)调用对象析构函数；(2)调用`::operator delete`释放内存

为了精密分工，STL allocator将两个阶段操作区分开来：内存配置有`alloc::allocate()`负责，内存释放由`alloc::deallocate()`负责；对象构造由`::construct()`负责，对象析构由`::destroy()`负责。

同时为了提升内存管理的效率，减少申请小内存造成的内存碎片问题，SGI STL采用了两级配置器，当分配的空间大小超过128B时，会使用第一级空间配置器；当分配的空间大小小于128B时，将使用第二级空间配置器。第一级空间配置器直接使用`malloc()`、`realloc()`、`free()`函数进行内存空间的分配和释放，而第二级空间配置器采用了内存池技术，通过空闲链表来管理内存。

## STL迭代器如何删除元素？

对于序列容器`vector`,`deque`来说，使用`erase(iterator)`后，后边的每个元素的迭代器都会失效，但是后边每个元素都会往前移动一个位置，但是`erase`会返回下一个有效的迭代器；

对于关联容器`map` `set`来说，使用了`erase(iterator)`后，当前元素的迭代器失效，但是其结构是红黑树，删除当前元素的，不会影响到下一个元素的迭代器，所以在调用`erase`之前，记录下一个元素的迭代器即可。

对于`list`来说，它使用了不连续分配的内存，并且它的`erase`方法也会返回下一个有效的`iterator`，因此上面两种正确的方法都可以使用。

## STL中MAP数据如何存放的？

红黑树。`unordered_map`底层结构是哈希表

## STL中`map`与`unordered_map`有什么区别？

`map`在底层使用了红黑树来实现，`unordered_map`是C++11标准中新加入的容器，它的底层是使用hash表的形式来完成映射的功能，`map`是按照`operator<`比较判断元素是否相同，以及比较元素的大小，然后选择合适的位置插入到树中。所以，如果对`map`进行遍历（中序遍历）的话，输出的结果是有序的。顺序就是按照`operator<`定义的大小排序。

而`unordered_map`是计算元素的Hash值，根据Hash值判断元素是否相同。所以，对`unordered_map`进行遍历，结果是无序的。

使用`map`时，需要为`key`定义`operator<`。而`unordered_map`的使用需要定义`hash_value`函数并且重载`operator==`。对于内置类型，如`string`，这些都不用操心，可以使用默认的。对于自定义的类型做`key`，就需要自己重载`operator<`或者`hash_value()`了。

所以说，当不需要结果排好序时，最好用`unordered_map`，插入删除和查询的效率要高于`map`。

## `vector`和`list`的区别是什么？

1. `vector`底层实现是数组；`list`是双向链表。
2. `vector`支持随机访问，`list`不支持。
3. `vector`是顺序内存，`list`不是。
4. `vector`在中间节点进行插入删除会导致内存拷贝，`list`不会。
5. `vector`一次性分配好内存，不够时才进行2倍扩容；`list`每次插入新节点都会进行内存申请。
6. `vector`随机访问性能好，插入删除性能差；`list`随机访问性能差，插入删除性能好。

## STL中迭代器有什么作用？有指针为何还要迭代器？

### 1、迭代器

Iterator（迭代器）模式又称Cursor（游标）模式，用于提供一种方法顺序访问一个聚合对象中各个元素，而又不需暴露该对象的内部表示。或者说可能更容易理解：Iterator模式是运用于聚合对象的一种模式，通过运用该模式，使得我们可以在不知道对象内部表示的情况下，按照一定顺序（由iterator提供的方法）访问聚合对象中的各个元素。

由于Iterator模式的以上特性：与聚合对象耦合，在一定程度上限制了它的广泛运用，一般仅用于底层聚合支持类，如STL的list、vector、stack等容器类及ostream\_iterator等扩展iterator。

### 2、迭代器和指针的区别

迭代器不是指针，是类模板，表现的像指针。他只是模拟了指针的一些功能，通过重载了指针的一些操作符，->、\*、++、--等。迭代器封装了指针，是一个“可遍历STL（Standard Template Library）容器内全部或部分元素”的对象，本质是封装了原生指针，是指针概念的一种提升（lift），提供了比指针更高级的行为，相当于一种智能指针，他可以根据不同类型的数据结构来实现不同的++、--等操作。

迭代器返回的是对象引用而不是对象的值，所以cout只能输出迭代器使用\*取值后的值而不能直接输出其自身。

### 3、迭代器产生原因

Iterator类的访问方式就是把不同集合类的访问逻辑抽象出来，使得不用暴露集合内部的结构而达到循环遍历集合的效果。

## epoll的原理是什么？

调用顺序：

```

1 | int epoll_create(int size);
2 | int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);
3 | int epoll_wait(int epfd, struct epoll_event *events, int maxevents, int
   | timeout);
```

首先创建一个epoll对象，然后使用epoll\_ctl对这个对象进行操作，把需要监控的描述添加进去，这些描述如将会以epoll\_event结构体的形式组成一颗红黑树，接着阻塞在epoll\_wait，进入大循环，当某个fd上有事件发生时，内核将会把其对应的结构体放入到一个链表中，返回有事件发生的链表。

## STL里resize和reserve的区别是什么？

改变当前容器内含有元素的数量(size())，eg: vector v; v.resize(len); v的size变为len,如果原来v的size小于len，那么容器新增 (len-size) 个元素，元素的值为

默认为0.当v.push\_back(3);之后，则是3是放在了v的末尾，即下标为len，此时容器是size为len+1；

改变当前容器的最大容量(capacity)，它不会生成元素，只是确定这个容器允许放入多少对象，如果reserve(len)的值大于当前的capacity()，那么会重新分配一块能存len个对象的空间，然后把之前v.size()个对象通过copy constructor复制过来，销毁之前的内存；



## 类和数据抽象

### C++中类成员的访问权限？

C++通过 public、protected、private 三个关键字来控制成员变量和成员函数的访问权限，它们分别表示公有的、受保护的、私有的，被称为成员访问限定符。在类的内部（定义类的代码内部），无论成员被声明为 public、protected 还是 private，都是可以互相访问的，没有访问权限的限制。在类的外部（定义类的代码之外），只能通过对对象访问成员，并且通过对对象只能访问 public 属性的成员，不能访问 private、protected 属性的成员。

### C++中struct和class的区别是什么？

在C++中，可以用struct和class定义类，都可以继承。区别在于：structural的默认继承权限和默认访问权限是public，而class的默认继承权限和默认访问权限是private。另外，class还可以定义模板类形参，比如template。

### C++类内可以定义引用数据成员吗？

可以，必须通过成员函数初始化列表初始化。

### 面向对象与泛型编程是什么？

1. 面向对象编程简称OOP,是一种程序设计思想。OOP把对象作为程序的基本单元,一个对象包含了数据和操作数据的函数。
2. 面向过程的程序设计把计算机程序视为一系列的命令集合，即一组函数的顺序执行。为了简化程序设计，面向过程把函数继续切分为子函数，即把大块函数通过切割成小块函数来降低系统的复杂度。
3. 泛型编程: 让类型参数化,方便程序员编码。

类型参数化: 使的程序(算法)可以从逻辑功能上抽象,把被处理对象(数据)的类型作为参数传递。

### 什么是右值引用，跟左值又有什么区别？

#### 左值和右值的概念：

左值：能对表达式取地址、或具名对象/变量。一般指表达式结束后依然存在的持久对象。

右值：不能对表达式取地址，或匿名对象。一般指表达式结束就不再存在的临时对象。

#### 右值引用和左值引用的区别：

1. 左值可以寻址，而右值不可以。
2. 左值可以被赋值，右值不可以被赋值，可以用来给左值赋值。
3. 左值可变,右值不可变（仅对基础类型适用，用户自定义类型右值引用可以通过成员函数改变）。

## 析构函数可以为 virtual 型，构造函数则不能，为什么？

构造函数不能声明为虚函数，析构函数可以声明为虚函数，而且有时是必须声明为虚函数。不建议在构造函数和析构函数里面调用虚函数。

构造函数不能声明为虚函数的原因是：

虚函数的主要意义在于被派生类继承从而产生多态。派生类的构造函数中，编译器会加入构造基类的代码，如果基类的构造函数用到参数，则派生类在其构造函

数的初始化列表中必须为基类给出参数，就是这个原因。虚函数的意思就是开启动态绑定，程序会根据对象的动态类型来选择要调用的方法。然而在构造函数运行的时候，这个对象的动态类型还不完整，没有办法确定它到底是什么类型，故构造函数不能动态绑定。（动态绑定是根据对象的动态类型而不是函数名，在调用构造函数之前，这个对象根本就不存在，它怎么动态绑定？）

## C++中空类默认产生哪些类成员函数？

C++中空类默认会产生以下6个函数：默认构造函数、复制构造函数、析构函数、赋值运算符重载函数、取址运算法重载函数、const取址运算符重载函数等。

```

1 class Empty
2 {
3     public:
4         Empty(); // 缺省构造函数
5         Empty( const Empty& ); // 拷贝构造函数
6         ~Empty(); // 析构函数
7         Empty& operator=( const Empty& ); // 赋值运算符
8         Empty* operator&(); // 取址运算符
9         const Empty* operator&() const; // 取址运算符 const
10    };

```



## 面向对象

### 面向对象和面向过程有什么区别？

面向对象与面向过程有以下四个方面的不同：

1. 出发点不同

面向对象使用符合常规思维的方式来处理客观世界的问题，强调把解决问题领域的“动作”直接映射到对象之间的接口上。而面向过程则强调的是过程的抽象化与模块化，是以过程为中心构造或处理客观世界问题。

2. 层次逻辑关系不同

面向对象使用计算机逻辑来模拟客观世界中的物理存在，以对象的集合类作为处理问题的单位，尽可能地使计算机世界向客观世界靠拢，以使处理问题的方式更清晰直接，面向对象使用类的层次结构来体现类之间的继承与发展。面向过程处理问题的基本单位是能清晰准确地表达过程的模块，用模块的层次结构概括模块或模块间的关系与功能，把客观世界的问题抽象成计算机可以处理的过程。

### 3. 数据处理方式与控制程序方式不同

面向对象将数据与对应的代码封装成一个整体，原则上其他对象不能直接修改其数据，即对象的修改只能由自身的成员函数完成，控制程序方式上是通过“事件驱动”来激活和运行程序的。而面向过程是直接通过程序来处理数据，处理完毕后即可显示处理的结果，在控制方式上是按照设计调用或返回程序，不能自由导航，各模块之间存在着控制与被控制，调动与被调用的关系。

### 4. 分析设计与编码转换方式不同

面向对象贯穿于软件生命周期的分析、设计及编码中，是一种平滑的过程，从分析到设计再到编码是采用一致性的模型表示，实现的是一种无缝连接。而面向过程强调分析、设计及编码之间按规则进行转换贯穿于软件生命周期的分析、设计及编码中，实现的是一种有缝的连接。

## 面向对象的基本特征有哪些？

面向对象的编程方法有四个基本特性：

1. 抽象：就是忽略一个主题中与当前目标无关的方面，以便更充分地注意与当前目标有关的方面。抽象并不打算了解全部问题，而只是选择其中的一部分，暂时不用部分细节。抽象包括两个方面，一是过程抽象，二是数据抽象。

过程抽象是指任何一个明确定义功能的操作都可被使用者看作单个的实体看待，尽管这个操作实际上可能由一系列更低级的操作来完成。数据抽象定义了数据类型和施加于该类型对象上的操作，并限定了对象的值，只能通过使用这些操作修改和观察。

2. 继承：这是一种联结类的层次模型，并且允许和鼓励类的重用，它提供了一种明确表述共性的方法。对象的一个新类可以从现有的类中派生，这个过程称为类继承。新类继承了原始类的特性，新类称为原始类的派生类（子类），而原始类称为新类的基类（父类）。

派生类可以从它的基类那里继承方法和实例变量，并且类可以修改或增加新的方法使之更适合特殊的需要。这也体现了大自然中一般与特殊的关系。继承性很好地解决了软件的可重用性问题。

3. 封装：就是把过程和数据包围起来，对数据的访问只能通过已定义的接口。面向对象的计算始于这个基本概念，即现实世界可以被描绘成一系列完全自治、封装的对象，这些对象通过一个受保护的接口访问其他对象。一旦定义了一个对象的特性，则有必要决定这些特性的可见性，即哪些特性对外部世界是可见的，哪些特性用于表示内部状态。

在这个阶段定义对象的接口。通常，应禁止直接访问一个对象的实际表示，而应通过操作接口访问对象，这称为信息隐藏。封装保证了模块具有较好的独立性，使得程序维护修改较为容易。对应用程序的修改仅限于类的内部，因而可以将应用程序修改带来的影响减少到最低限度。

4. 多态：是指允许不同类的对象对同一消息做出响应。比如同样的复制-粘贴操作，在字处理程序和绘图程序中有不同的效果。多态性包括参数化多态性和包含多态性。多态性语言具有灵活、抽象、行为共享、代码共享的优势，很好地解决了应用程序函数同名问题。

## 什么是深拷贝？什么是浅拷贝？

深拷贝是彻底的拷贝，两对象中所有的成员都是独立的一份，而且，成员对象中的成员对象也是独立一份。

浅拷贝中的某些成员变量可能是共享的，深拷贝如果不彻底，就是浅拷贝。

## 什么是友元？

有成员只能在类的成员函数内部访问，如果想在别处访问对象的私有成员，只能通过类提供的接口（成员函数）间接地进行。这固然能够带来数据隐藏的好处，利于将来程序的扩充，但也会增加程序书写的麻烦。

C++是从结构化的C语言发展而来的，需要照顾结构化设计程序员的习惯，所以在对私有成员可访问范围的问题上不可限制太死。

C++设计者认为，如果有的程序员真的非常怕麻烦，就是想在类的成员函数外部直接访问对象的私有成员，那还是做一点妥协以满足他们的愿望为好，这也算是眼前利益和长远利益的折中。因此，C++就有了**友元 (friend)**的概念。打个比方，这相当于是说：朋友是值得信任的，所以可以对他们公开一些自己的隐私。

友元提供了一种普通函数或者类成员函数访问另一个类中的私有或保护成员的机制。也就是说有两种形式的友元：

- (1) 友元函数：普通函数对一个访问某个类中的私有或保护成员。
- (2) 友元类：类A中的成员函数访问类B中的私有或保护成员。

## 基类的构造函数/析构函数是否能被派生类继承？

基类的构造函数/析构函数不能被派生类继承。

基类的构造函数不能被派生类继承，派生类中需要声明自己的构造函数。设计派生类的构造函数时，不仅要考虑派生类所增加的数据成员初始化，也要考虑基类的数据成员的初始化。声明构造函数时，只需要对本类中新增成员进行初始化，对继承来的基类成员的初始化，需要调用基类构造函数完成。

基类的析构函数也不能被派生类继承，派生类需要自行声明析构函数。声明方法与一般（无继承关系时）类的析构函数相同，不需要显式地调用基类的析构函数，系统会自动隐式调用。需要注意的是，析构函数的调用次序与构造函数相反。

## 初始化列表和构造函数初始化的区别？

构造函数初始化列表以一个冒号开始，接着是以逗号分隔的数据成员列表，每个数据成员后面跟一个放在括号中的初始化式。例如：

```
1 | Example::Example() : ival(0), dval(0.0) {} //ival 和dval 是类的两个数据成员
```

上面的例子和下面不用初始化列表的构造函数看似没什么区别：

```
1 | Example::Example()
2 | {
3 |     ival = 0;
4 |     dval = 0.0;
5 | }
```

的确，这两个构造函数的结果是一样的。但区别在于：上面的构造函数（使用初始化列表的构造函数）显示的初始化类的成员；而没使用初始化列表的构造函数是对类的成员赋值，并没有进行显示的初始化。

初始化和赋值对内置类型的成员没有什么大的区别，像上面的任一个构造函数都可以。但有的时候必须用带有初始化列表的构造函数：

1. 成员类型是没有默认构造函数的类。若没有提供显示初始化式，则编译器隐式使用成员类型的默认构造函数，若类没有默认构造函数，则编译器尝试使用默认构造函数将会失败。
2. const成员或引用类型的员。因为const对象或引用类型只能初始化，不能对他们赋值。

## C++中有那些情况只能用初始化列表，而不能用赋值？

构造函数初始化列表以一个冒号开始，接着是以逗号分隔的数据成员列表，每个数据成员后面都跟一个放在括号中的初始化式。例如，Example:Example ival (o,dva (0.0) {}，其中ival与dva是类的两个数据成员。

在C++语言中，赋值与初始化列表的原理不一样，赋值是删除原值，赋予新值，初始化列表开辟空间和初始化是同时完成的，直接给予一个值

所以，在C++中，赋值与初始化列表的使用情况也不一样，只能用初始化列表，而不能用赋值的情况一般有以下3种：

1. 当类中含有 const (常量)、reference (引用) 成员变量时，只能初始化，不能对它们进行赋值。常量不能被赋值，只能被初始化，所以必须在初始化列表中完成，C++的引用也一定要初始化，所以必须在初始化列表中完成。
2. 派生类在构造函数中要对自身成员初始化，也要对继承过来的基类成员进行初始化当基类没有默认构造函数的时候，通过在派生类的构造函数初始化列表中调用基类的构造函数实现。
3. 如果成员类型是没有默认构造函数的类，也只能使用初始化列表。若没有提供显式初始化时，则编译器隐式使用成员类型的默认构造函数，此时编译器尝试使用默认构造函数将会失败

## 类的成员变量的初始化顺序是什么？

1. 成员变量在使用初始化列表初始化时，与构造函数中初始化成员列表的顺序无关，只与定义成员变量的顺序有关。因为成员变量的初始化次序是根据变量在内存中次序有关，而内存中的排列顺序早在编译期就根据变量的定义次序决定了。这点在EffectiveC++中有详细介绍。
2. 如果不使用初始化列表初始化，在构造函数内初始化时，此时与成员变量在构造函数中的位置有关。
3. 注意：类成员在定义时，是不能初始化的
4. 注意：类中const成员常量必须在构造函数初始化列表中初始化。
5. 注意：类中static成员变量，必须在类外初始化。
6. 静态变量进行初始化顺序是基类的静态变量先初始化，然后是它的派生类。直到所有的静态变量都被初始化。这里需要注意全局变量和静态变量的初始化是不分次序的。这也不难理解，其实静态变量和全局变量都被放在公共内存区。可以把静态变量理解为带有“作用域”的全局变量。在一切初始化工作结束后，main函数会被调用，如果某个类的构造函数被执行，那么首先基类的成员变量会被初始化。

## 当一个类为另一个类的成员变量时，如何对其进行初始化？

示例程序如下：

```

1 class ABC
2 {
3 public:
4     ABC(int x, int y, int z);
5 private :
6     int a;
7     int b;
8     int c;
9 };
10
11 class MyClass
12 {
13 public:
14     MyClass():abc(1,2,3)
15     {
16

```

```

17 }
18
19 private:
20 ABC abc;
21 };

```

上例中，因为ABC有了显式的带参数的构造函数，那么它是无法依靠编译器生成无参构造函数的，所以必须使用初始化列表:abc(1,2,3)，才能构造ABC的对象。

## C++能设计实现一个不能被继承的类吗？

在Java中定义了关键字final，被final修饰的类不能被继承。但在C++中没有final这个关键字，要实现这个要求还是需要花费一些精力。

首先想到的是在C++中，子类的构造函数会自动调用父类的构造函数。同样，子类的析构函数也会自动调用父类的析构函数。要想一个类不能被继承，我们只要把它的构造函数和析构函数都定义为私有函数。那么当一个类试图从它那继承的时候，必然会由于试图调用构造函数、析构函数而导致编译错误。

可是这个类的构造函数和析构函数都是私有函数了，我们怎样才能得到该类的实例呢？这难不倒我们，我们可以通过定义静态来创建和释放类的实例。

基于这个思路，我们可以写出如下的代码：

```

1 // Define a class which can't be derived from /// class FinalClass1
2 {
3     public :
4     static FinalClass1* GetInstance()
5     {
6         return new FinalClass1;
7     }
8     static void DeleteInstance( FinalClass1* pInstance)
9     {
10        delete pInstance;
11        pInstance = 0;
12    }
13     private :
14     FinalClass1() {}
15     ~FinalClass1() {}
16 };

```

这个类是不能被继承，但在总觉得它和一般的类有些不一样，使用起来也有点不方便。比如，我们只能得到位于堆上的实例，而得不到位于栈上实例。能不能实现一个和一般类除了不能被继承之外其他用法都一样的类呢？办法总是有的，不过需要一些技巧。请看如下代码：

```

1 // Define a class which can't be derived from /// template <typename
2 T> class MakeFinal
3 {
4     friend T;
5     private :
6     MakeFinal() {}
7     ~MakeFinal() {}
8 };
9 class FinalClass2 :
10     virtual public MakeFinal<FinalClass2>
11 {
12     public :
13     FinalClass2() {}

```

```

13     ~FinalClass2() {}
14 };

```

这个类使用起来和一般的类没有区别，可以在栈上、也可以在堆上创建实例。尽管类 `MakeFinal <FinalClass2>` 的构造函数和析构函数都是私有的，但由于类 `FinalClass2` 是它的友元函数，因此在 `FinalClass2` 中调用 `MakeFinal <FinalClass2>` 的构造函数和析构函数都不会造成编译错误。但当我们试图从 `FinalClass2` 继承一个类并创建它的实例时，却不能通过编译。

```

1 class Try : public FinalClass2
2 {
3     public :
4         Try() {}
5         ~Try() {}
6     };     Try temp;

```

由于类 `FinalClass2` 是从类 `MakeFinal <FinalClass2>` 虚继承过来的，在调用 `Try` 的构造函数的时候，会直接跳过 `FinalClass2` 而直接调用 `MakeFinal <FinalClass2>` 的构造函数。非常遗憾的是 `Try` 不是 `MakeFinal <FinalClass2>` 的友元，因此不能调用其私有的构造函数。

基于上面的分析，试图从 `FinalClass2` 继承的类，一旦实例化，都会导致编译错误，因此是 `FinalClass2` 不能被继承。这就满足了我们设计要求。

## 构造函数没有返回值，那么如何得知对象是否构造成功？

这里的“构造”不单指分配对象本身的内存，而是指在建立对象时做的初始化操作（如打开文件、连接数据库等）。

因为构造函数没有返回值，所以通知对象的构造失败的唯一方法就是在构造函数中抛出异常。构造函数中抛出异常将导致对象的析构函数不被执行，当对象发生部分构造时，已经构造完毕的子对象将会逆序地被析构。

## Public继承、protected继承、private继承的区别？

`public`（公有）继承、`protected`（保护）继承和 `private`（私有）继承是常见的3种继承方式。

### 1. 公有继承

对于子类的对象而言，采用公有继承时，基类成员对子类对象的可见性与一般类成员对对象的可见性相同，公有成员可见，其他成员不可见。

对于子类而言，基类的公有成员和保护成员可见；基类的公有成员和保护成员作为派生类的成员时，它们都维持原有的可见性（基类 `public` 成员在子类中还是 `public`，基类 `protected` 成员在子类中还是 `protected`）；基类的私有成员不可见，基类的私有成员依然是私有的，子类不可访问。

### 2. 保护继承

保护继承的特点是：基类的所有公有成员和保护成员都成为派生类的保护成员，并且只能被它的派生类成员函数或友元访问。基类的私有成员仍然是私有的。由此可以看出，基类的所有成员对子类的对象都是不可见的。

### 3. 私有继承

私有继承的特点是，基类的公有成员和保护成员都作为派生类的私有成员，并且不能被这个派生类的子类所访问。

## C++提供默认参数的函数吗？

C++可以给函数定义默认参数值。在函数调用时没有指定与形参相对应的实参时，就自动使用默认参数。

默认参数的语法与使用：

- (1) 在函数声明或定义时，直接对参数赋值，这就是默认参数。
- (2) 在函数调用时，省略部分或全部参数。这时可以用默认参数来代替。

通常调用函数时，要为函数的每个参数给定对应的实参。例如：

```

1 void delay(int loops=1000); //函数声明
2
3 void delay(int loops) //函数定义
4 {
5     if(loops==0)
6     {
7         return;
8     }
9     for(int i=0;i<loops;i++)
10    ;
11 }
```

在上例中，如果将delay()函数中的loops定义成默认值1000，这样，以后无论何时调用delay()函数，都不用给loops赋值，程序都会自动将它当做值1000进行处理。例如，当执行delay(2500)调用时，loops的参数值为显性化的，被设置为2500；当执行delay()时，loops将采用默认值1000。

默认参数在函数声明中提供，当有声明又有定义时，定义中不允许默认参数。如果函数只有定义，则默认参数才可出现在函数定义中。例如：

```

1 oid point(int=3, int=4); //声明中给出默认值
2
3 void point(int x, int y) //定义中不允许再给出默认值
4 {
5     cout<<x<<endl;
6     cout<<y<<endl;
7 }
```

如果一组重载函数（可能带有默认参数）都允许相同实参数个数的调用，将会引起调用的二义性。例如：

```

1 void func(int); //重载函数之一
2 void func(int, int=4); //重载函数之二，带有默认参数
3 void func(int=3, int=4); //重载函数三，带有默认参数
4 func(7); //错误：到底调用3个重载函数中的哪个？
5 func(20, 30); //错误：到底调用后面两个重载函数的那个？
```



## 虚函数

### 什么是虚函数？

指向基类的指针在操作它的多态类对象时，可以根据指向的不同类对象调用其相应的函数，这个函数就是虚函数。

虚函数的作用：在基类定义了虚函数后，可以在派生类中对虚函数进行重新定义，并且可以通过基类指针或引用，在程序的运行阶段动态地选择调用基类和不同派生类中的同名函数。（如果在派生类中没有对虚函数重新定义，则它继承其基类的虚函数。）

下面是一个虚函数的实例程序：

```

1 #include "stdafx.h"
2 #include<iostream>
3 using namespace std;
4
5 class Base
6 {
7 public:
8     virtual void Print() //父类虚函数
9     {
10         printf("This is Class Base!\n");
11     }
12 };
13
14 class Derived1 :public Base
15 {
16 public:
17     void Print() //子类1虚函数
18     {
19         printf("This is Class Derived1!\n");
20     }
21 };
22
23 class Derived2 :public Base
24 {
25 public:
26     void Print() //子类2虚函数
27     {
28         printf("This is Class Derived2!\n");
29     }
30 };
31
32 int main()
33 {
34     Base Cbase;

```

```

35     Derived1 Cderived1;
36     Derived2 Cderived2;
37     Cbase.Print();
38     Cderived1.Print();
39     Cderived2.Print();
40
41     cout << "-----" << endl;
42     Base *p1 = &Cbase;
43     Base *p2 = &Cderived1;
44     Base *p3 = &Cderived2;
45     p1->Print();
46     p2->Print();
47     p3->Print();
48 }
49
50 /*
51 输出结果:
52 This is Class Base!
53 This is Class Derived1!
54 This is Class Derived2!
55 -----
56 This is Class Base!
57 This is Class Derived1!
58 This is Class Derived2!
59 */

```

需要注意的是，虚函数虽然非常好用，但是在使用虚函数时，并非所有的函数都需要定义成虚函数，因为实现虚函数是有代价的。在使用虚函数时，需要注意以下几个方面的内容：

- (1) 只需要在声明函数的类体中使用关键字virtual将函数声明为虚函数，而定义函数时不需要使用关键字virtual。
- (2) 当将基类中的某一成员函数声明为虚函数后，派生类中的同名函数自动成为虚函数。
- (3) 非类的成员函数不能定义为虚函数，全局函数以及类的成员函数中静态成员函数和构造函数也不能定义为虚函数，但可以将析构函数定义为虚函数。
- (4) 基类的析构函数应该定义为虚函数，否则会造成内存泄漏。基类析构函数未声明virtual，基类指针指向派生类时，delete指针不调用派生类析构函数。有virtual，则先调用派生类析构再调用基类析构。

## C++如何实现多态？

C++中通过虚函数实现多态。虚函数的本质就是通过基类指针访问派生类定义的函数。每个含有虚函数的类，其实例对象内部都有一个虚函数表指针。该虚函数表指针被初始化为本类的虚函数表的内存地址。所以，在程序中，不管对象类型如何转换，该对象内部的虚函数表指针都是固定的，这样才能实现动态地对对象函数进行调用，这就是C++多态性的原理。

## 纯虚函数指的是什么？

纯虚函数是一种特殊的虚函数，格式一般如下

```

1 class <类名>
2 {
3     virtual()函数返回值类型 虚函数名(形参表) =0;
4     ...
5 };
6 class <类名>

```

由于在很多情况下，基类中不能对虚函数给出有意义的实现，只能把函数的实现留给派生类。例如，动物作为一个基类可以派生出老虎、孔雀等子类，但是动物本身生成对象不合情理，此时就可以将动物类中的函数定义为纯虚函数，如果基类中有纯虚函数，那么在子类中必须实现这个纯虚函数，否则子类将无法被实例化，也无法实现多态。

含有纯虚函数的类称为抽象类，抽象类不能生成对象。纯虚函数永远不会被调用，它们主要用来统一管理子类对象。

## 什么函数不能声明为虚函数？

常见的不能声明为虚函数的有：普通函数（非成员函数）；静态成员函数；内联成员函数；构造函数；友元函数。

### 1. 为什么C++不支持普通函数为虚函数？

普通函数（非成员函数）只能被overload，不能被override，声明为虚函数也没有什么意思，因此编译器会在编译时邦定函数。

### 2. 为什么C++不支持构造函数为虚函数？

这个原因很简单，主要是从语义上考虑，所以不支持。因为构造函数本来就是为了明确初始化对象成员才产生的，然而virtual function主要是为了再不完全了解细节的情况下也能正确处理对象。另外，virtual函数是在不同类型的对象产生不同的动作，现在对象还没有产生，如何使用virtual函数来完成你想完成的动作。（这不就是典型的悖论）

### 3. 为什么C++不支持内联成员函数为虚函数？

其实很简单，那内联函数就是为了在代码中直接展开，减少函数调用花费的代价，虚函数是为了在继承后对象能够准确的执行自己的动作，这是不可能统一的。（再说了，inline函数在编译时被展开，虚函数在运行时才能动态的邦定函数）

### 4. 为什么C++不支持静态成员函数为虚函数？

这也很简单，静态成员函数对于每个类来说只有一份代码，所有的对象都共享这一份代码，他也没有要动态邦定的必要性。

### 5. 为什么C++不支持友元函数为虚函数？

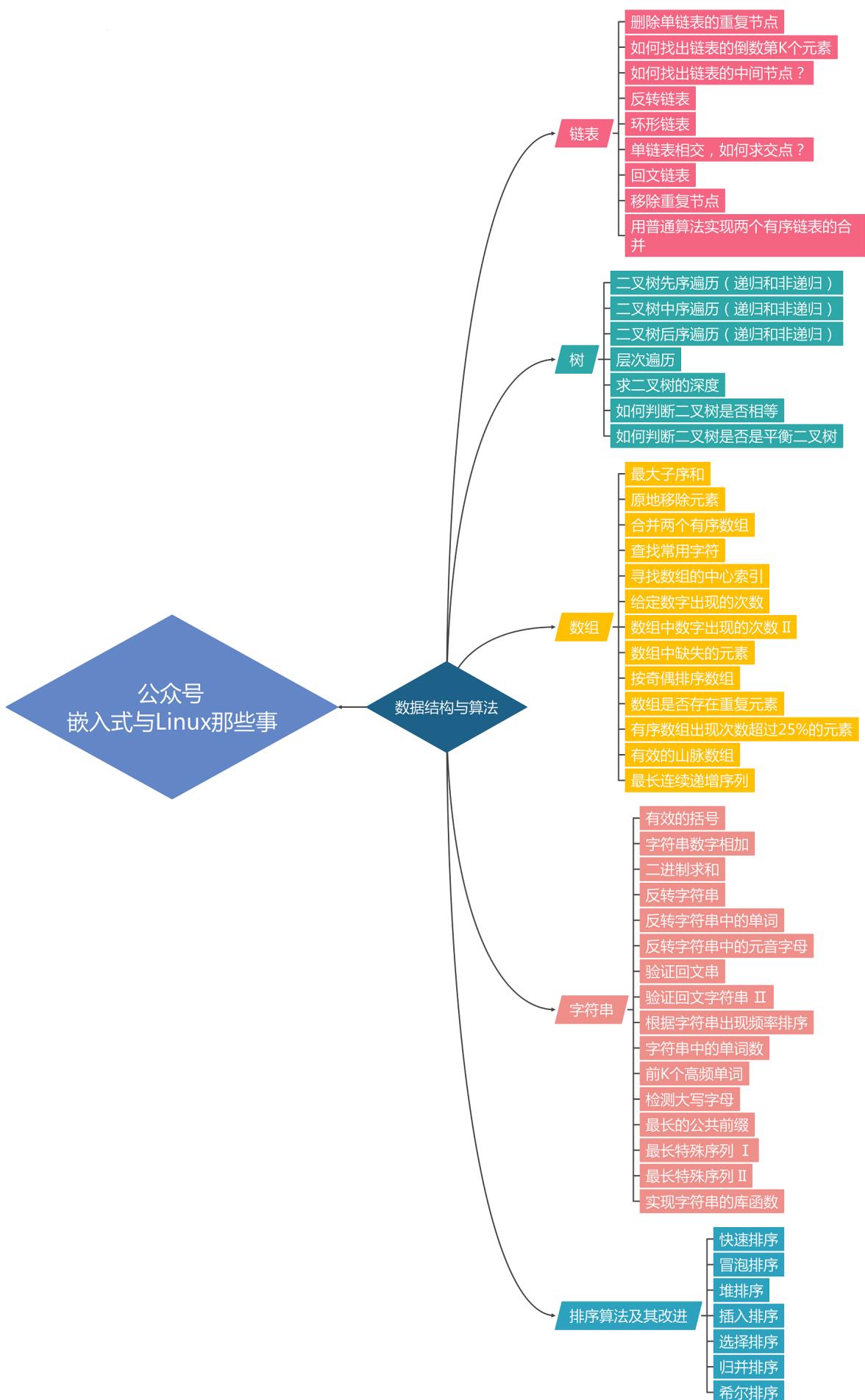
因为C++不支持友元函数的继承，对于没有继承特性的函数没有虚函数的说法。

## C++中如何阻止一个类被实例化？

C++中可以通过使用抽象类，或者将构造函数声明为private阻止一个类被实例化。抽象类之所以不能被实例化，是因为抽象类不能代表一类具体的事物，它是对多种具有相似性的具体事物的共同特征的一种抽象。例如，动物作为一个基类可以派生出老虎、孔雀等子类，但是动物本身生成对象不合情理。

## 数据结构与算法

---



说下数据结构，我建议用C++来刷题，可以避免重复造轮子。当然，C其实也可以，有些题目写起来会复杂点。

在刷题策略上。坚持一个原则：熟能生巧。

第一遍：不会就看答案。学习他人的最优解，建立思维体系，了解所有最优解，方法技巧第一。做题套路，以印象为主。

第二遍：自己想，锻炼逻辑基础。过easy题，记住；做medium，重点题背，反复背。刷过一遍，已经有了自己的思维，现在就是要培养自己的独立做题能力了。自己先尝试写答案，多多少少能写出来一些。写不出来的部分，看下题解，分析下自己卡在哪里。

第三遍：做easy题；做部分medium题，hard题有思路。夯实medium基础。熟练运用做题套路，以做题为主。

第四遍：做面经，开阔思路，了解出题形式。基础牢轻松，不牢就痛苦。

在笔试的时候，很多公司的大题的笔试系统其实并不是像力扣一样只写个子函数就行了，而是和ACM竞赛的类型一样，需要自己处理输入输出。这部分一定要提前练习。如果不熟悉系统，笔试的时候虽然有思路，但是也写不出来。建议提前熟悉下输入输出。

## [生客OJ输入输出训练](#)

# 链表

## 删除单链表的重复节点

### [力扣链接](#)

#### 题目描述

编写代码，移除未排序链表中的重复节点。保留最开始出现的节点。

示例1: 输入: [1, 2, 3, 3, 2, 1]

输出: [1, 2, 3]

示例2: 输入: [1, 1, 1, 1, 2]

输出: [1, 2]

提示: 链表长度在[0, 20000]范围内。链表元素在[0, 20000]范围内。

进阶:

如果不得使用临时缓冲区，该怎么解决？

#### 解题思路

思路一：定义两个指针current和p来逐个遍历链表，current元素依次和p比较，直到p为NULL，current向后移动一个。

思路二：利用标记数组，标记当前值是否出现过。

#### 代码实现

方法一：

```

1 struct ListNode* removeDuplicateNodes(struct ListNode* head){
2     //判断是否需要遍历
3     if (head == NULL) return NULL;
4     struct ListNode* current = head;
5     //双指针逐个比较
6     while (current) {
7         struct ListNode* p = current;
8         while(p->next) {
9             if (p->next->val == current->val) {
10                 p->next = p->next->next;
11             } else {
12                 p = p->next;
13             }
}

```

```

14     }
15     current = current->next;
16 }
17 return head;
18 }
```

## 方法二：

```

1 struct ListNode* removeDuplicateNodes(struct ListNode* head){
2     //判断是否需要遍历
3     if(head==NULL || head->next==NULL)
4         return head;
5     //利用val的值在0~20000之间，标记是否出现过
6     int index[20001] = {0};
7     index[head->val] = 1;
8     struct ListNode *pre = head, *q = head->next;
9     while(q)
10    {
11        //当前val未出现过，保存当前val，修改pre和p指针
12        if(index[q->val]==0)
13        {
14            index[q->val] = 1;
15            pre = q;
16            q = q->next;
17        }
18        //当前值已经出现过了，删除当前节点
19        else
20        {
21            pre->next = q->next;
22            q = pre->next;
23        }
24    }
25    return head;
26 }
```

## 如何找出链表的倒数第K个元素？

### 题目描述

#### [力扣链接](#)

输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点。

例如，一个链表有 6 个节点，从头节点开始，它们的值依次是 1、2、3、4、5、6。这个链表的倒数第 3 个节点是值为 4 的节点。

示例：

给定一个链表: 1->2->3->4->5, 和 k = 2.

返回链表 4->5.

## 解题思路

快慢指针，先让快指针走k步，然后两个指针同步走，当快指针走到头时，慢指针就是链表倒数第k个节点。

## 代码

```

1 struct ListNode* getKthFromEnd(struct ListNode* head, int k){
2     /*双指针*/
3     struct ListNode* former = head;
4     struct ListNode* latter = head;
5     int i = 0;
6     for(i = 0; i < k; i++)
7     {
8         former = former->next;
9     }
10    while(former)
11    {
12        latter = latter->next;
13        former = former->next;
14    }
15    return latter;
16 }
```

## 如何找出链表的中间节点

### 题目描述

#### [力扣链接](#)

给定一个带有头结点 head 的非空单链表，返回链表的中间结点。

如果有两个中间结点，则返回第二个中间结点。

示例 1：

输入：[1,2,3,4,5]

输出：此列表中的结点 3(序列化形式：[3,4,5])

返回的结点值为 3。 (测评系统对该结点序列化表述是 [3,4,5])。

注意，我们返回了一个 ListNode 类型的对象 ans，这样：

ans.val = 3, ans.next.val = 4, ans.next.next.val = 5, 以及 ans.next.next.next = NULL.

示例 2：

输入：[1,2,3,4,5,6]

输出：此列表中的结点 4 (序列化形式：[4,5,6])

由于该列表有两个中间结点，值分别为 3 和 4，我们返回第二个结点。

提示：给定链表的结点数介于 1 和 100 之间。

## 解题思路

这是一个典型的双指针问题，定义一个快指针，一个慢指针。快指针一次走两步，慢指针一次走一步。当快指针走到结尾时，慢指针指向的就是中间节点。这里要注意一个细节：题目要求：「两个中间结点的时候，返回第二个中间结点」。此时可以在草稿纸上写写画画，就拿自己的左右手的两根指头同步移动，可以得出：快指针可以前进的条件是：**当前快指针和当前快指针的下一个结点都非空**。如果题目要求「在两个中间结点的时候，返回第一个中间结点」，此时快指针可以前进的条件是：**当前快指针的下一个结点和当前快指针的下下一个结点都非空**。注意体会以上二者的不同之处。

## 代码

```

1 struct ListNode* middleNode(struct ListNode* head){
2     struct ListNode* slower = head;
3     struct ListNode* faster = head;
4     //注意边界条件
5     while((faster!=NULL)&&(faster->next!=NULL))
6     {
7
8         faster=faster->next->next;
9         slower=slower->next;
10    }
11    return slower;
12 }
```

## 反转链表

### 题目描述

#### [力扣链接](#)

定义一个函数，输入一个链表的头节点，反转该链表并输出反转后链表的头节点。

示例:

输入: 1->2->3->4->5->NULL

输出: 5->4->3->2->1->NULL

限制: 0 <= 节点个数 <= 5000

### 解题思路

反转链表实际就是重新指向结构体中的next指针，我们需要修改下一个节点的next指针指向前一个节点。所以，在遍历链表时我们要逐个修改链表的指针指向。这个题目也可以用递归来做，一直递归到链表的最后一个结点，该结点就是反转后的头结点，记作head。

此后，每次函数在返回的过程中，让当前结点的下一个结点的 next 指针指向当前节点。同时让当前结点的 next 指针指向 NULL，从而实现从链表尾部开始的局部反转。当递归函数全部出栈后，链表反转完成。

## 代码

### 非递归

```

1 struct ListNode* reverseList(struct ListNode* head){
2
3     //判断节点长度
4     if(head == NULL || head->next == NULL)
5         return head;
6     else
7     {
8         struct ListNode* former = NULL;
9         struct ListNode* mid = head;
10        struct ListNode* latter = NULL;
11        while(mid != NULL)
12        {
13            //保存下一个节点
14            latter = mid->next;
15            //让该节点指向上一个节点
16            mid->next = former;
```

```

17         //上一个节点走到当前节点
18         former = mid;
19         //当前节点走到下一个节点
20         mid = latter;
21     }
22     return former;
23 }
```

递归版本

```

1 struct ListNode* reverseList(struct ListNode* head){
2
3     //递归版本
4     if(head == NULL || head->next == NULL){
5         return head;
6     }
7     else{
8         //保存头节点
9         struct ListNode* mid = head;
10        //保存下一个节点
11        struct ListNode* latter = mid->next;
12        //递归到最后一个几点，返回转置后链表的地址
13        head = reverseList(latter);
14        //让后面的节点指向前一个节点
15        latter->next = mid;
16        //每次递归返回都赋值为空，最后一次返回将转置后的节点的next赋值为空
17        mid->next = NULL;
18        return head;
19    }
20 }
```

## 环形链表

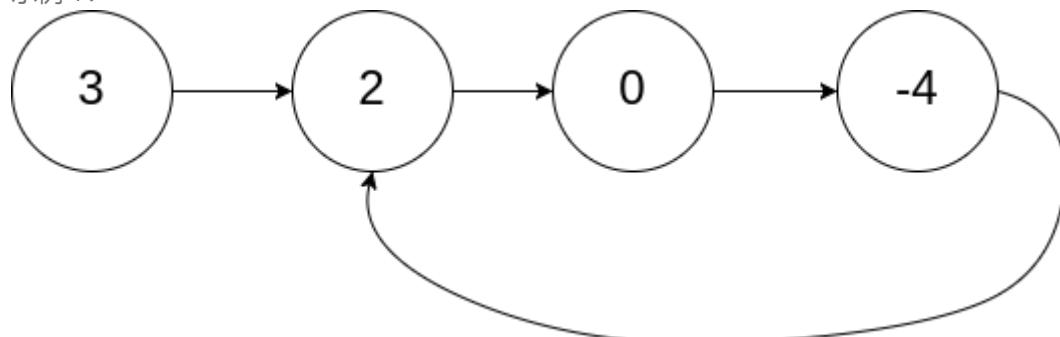
### 题目描述

#### [力扣链接](#)

给定一个链表，返回链表开始入环的第一个节点。如果链表无环，则返回 null。  
为了表示给定链表中的环，我们使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始）。如果 pos 是 -1，则在该链表中没有环。

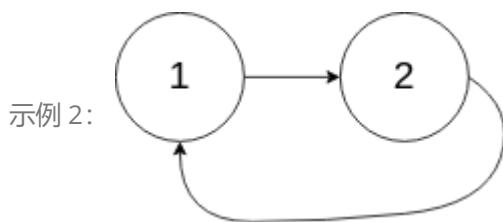
说明：不允许修改给定的链表。

示例 1：



输入：head = [3,2,0,-4], pos = 1 输出：tail connects to node index 1

解释：链表中有一个环，其尾部连接到第二个节点。



输入: head = [1,2], pos = 0 输出: tail connects to node index 0

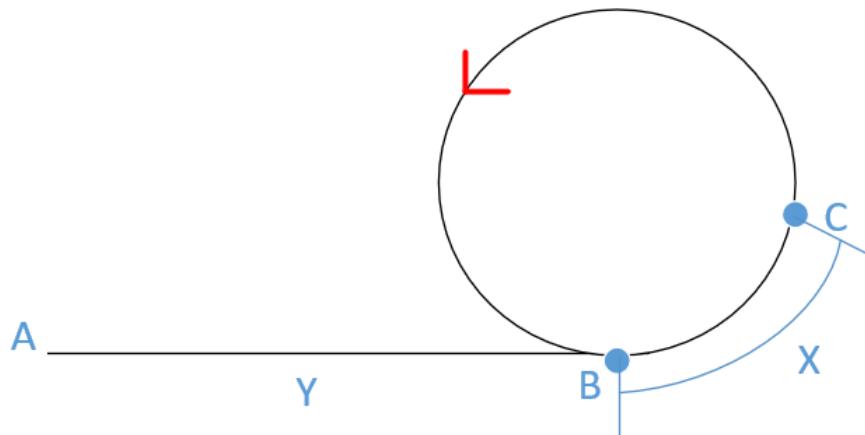
解释: 链表中有一个环, 其尾部连接到第一个节点。



输入: head = [1], pos = -1 输出: no cycle 解释: 链表中没有环。

进阶: 你是否可以不用额外空间解决此题?

### 解题思路



[https://blog.csdn.net/qq\\_16933601](https://blog.csdn.net/qq_16933601)

假设有两个指针, 分别为快慢指针fast和slow, 快指针每次走两步, 慢指针每次前进一步, 如果有环则两个指针必定相遇;

A:链表起点

B:环起点

C:相遇点

X:环起点到相遇点距离

Y:链表起点到环起点距离

R: 环的长度

S:第一次相遇时走过的路程

慢指针slow第一次相遇走过的路程  $S1 = Y + X; (1)$

快指针fast第一次相遇走过的路程  $S2 = 2S1 = Y + X + NR; (2)$

说明: 快指针的速度是慢指针的两倍, 相同时间内路程应该是慢指针的两倍,  $Y + X + NR$ 是因为快指针可能经过N圈后两者才相遇;

把 (1) 式代入 (2) 式得:  $Y = NR - X;$

表明头节点和C点的慢指针会在B点相遇, 此处就是环节点.

### 代码

```

1 struct ListNode *detectCycle(struct ListNode *head) {
2     struct ListNode *slow = head;
3     struct ListNode *fast = head;
4     while (fast != NULL && fast->next != NULL) {
  
```

```

5         slow = slow->next;
6         fast = fast->next->next;
7         if (slow == fast) {
8             fast = head;
9             while (slow != fast) {
10                 slow = slow->next;
11                 fast = fast->next;
12             }
13             return slow;
14         }
15     }
16     return NULL;
17 }
```

## 单链表相交，如何求交点？

### 题目描述

#### 力扣链接

给定两个（单向）链表，判定它们是否相交并返回交点。请注意相交的定义基于节点的引用，而不是基于节点的值。换句话说，如果一个链表的第k个节点与另一个链表的第j个节点是同一节点（引用完全相同），则这两个链表相交。

示例 1：

输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB = 3

输出：Reference of the node with value = 8

输入解释：相交节点的值为 8 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。

示例 2：

输入：intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1

输出：Reference of the node with value = 2

输入解释：相交节点的值为 2

（注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。

示例 3：

输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2

输出：null

输入解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。解释：这两个链表不相交，因此返回 null。

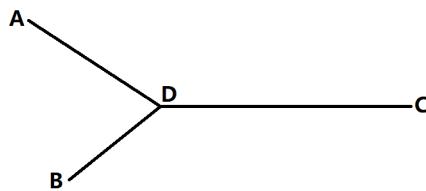
注意：

如果两个链表没有交点，返回 null。在返回结果后，两个链表仍须保持原有的结构。可假定整个链表结构中没有循环。程序尽量满足 O(n)

时间复杂度，且仅用 O(1) 内存。

## 解题思路

根据题意,两个链表相交的点是指: 两个指针指向的内容相同,则说明该结点记在A链表上又在B链表上,进而说明A和B是相交的而对于相交的情况,两条链表一定是这种结构:



[https://blog.csdn.net/wq\\_16933601](https://blog.csdn.net/wq_16933601)

为什么呢?

因为如果链表A和链表B相交于D的话,那么说明D结点即在A上又在B上,而D之后的元素自然也就均在A和B上了,因为他们是通过next指针相连的.

如果有相交的结点D的话,每条链的头结点先走完自己的链表长度,然后回头走另外的一条链表,那么两结点一定为相交于D点,因为这时每个头结点走的距离是一样的,都是  $AD + BD + DC$ ,而他们每次又都是前进1,所以距离相同,速度又相同,固然一定会在相同的时间走到相同的结点上,即D点。

## 代码

```

1 struct ListNode *getIntersectionNode(struct ListNode *headA, struct ListNode
2 *headB) {
3     struct ListNode *p=headA;
4     struct ListNode *q=headB;
5     while(p!=q)
6     {
7         if(p==NULL)
8             p=headB;
9         else
10            p=p->next;
11         if(q==NULL)
12             q=headA;
13         else
14             q=q->next;
15     }
16     return q;
}
  
```

## 回文链表

### 题目描述

#### 力扣链接

编写一个函数, 检查输入的链表是否是回文的。

示例 1：

输入： 1->2

输出： false

示例 2：

输入： 1->2->2->1

输出： true

进阶：

你能否用  $O(n)$  时间复杂度和  $O(1)$  空间复杂度解决此题？

## 解题思路

利用双指针找到链表的中间位置，然后将后半段反转，将后半段依次与前半段比较。

## 代码

```

1  bool isPalindrome(struct ListNode* head){
2      struct ListNode* slow = head;
3      struct ListNode* fast = head;
4      struct ListNode* p = head;
5      while((fast!=NULL) &&(fast->next!=NULL))
6      {
7          slow = slow->next;
8          fast = fast->next->next;
9      }
10     struct ListNode *cur = slow;
11     struct ListNode *pre = NULL;
12     //将链表的后半段反转
13     while(cur!=NULL){
14         //保存下一个的值
15         struct ListNode* temp = cur->next;
16         //重新连接当前节点
17         cur->next = pre;
18         //更新pre
19         pre = cur;
20         //cur指向下一个
21         cur = temp;
22     }
23     // 将后半段反转的链表与前半段依次比较
24     while(pre&&head){
25         if(pre->val!=head->val)
26             return false;
27         pre=pre->next;head=head->next;
28     }
29     return true;
30 }
31 }
```

## 移除重复节点

### 题目描述

[力扣链接](#)

编写代码，移除未排序链表中的重复节点。保留最开始出现的节点。

示例1: 输入: [1, 2, 3, 3, 2, 1]

输出: [1, 2, 3]

示例2: 输入: [1, 1, 1, 1, 2]

输出: [1, 2]

提示: 链表长度在[0, 20000]范围内。 链表元素在[0, 20000]范围内。

进阶:

如果不得使用临时缓冲区, 该怎么解决?

## 解题思路

思路一: 定义两个指针current和p来逐个遍历链表, current元素依次和p比较, 直到p为NULL, current向后移动一个。

思路二: 利用标记数组, 标记当前值是否出现过

## 代码

方法一:

```

1 struct ListNode* removeDuplicateNodes(struct ListNode* head){
2     //判断是否需要遍历
3     if (head == NULL) return NULL;
4     struct ListNode* current = head;
5     //双指针逐个比较
6     while (current) {
7         struct ListNode* p = current;
8         while(p->next) {
9             if (p->next->val == current->val) {
10                 p->next = p->next->next;
11             } else {
12                 p = p->next;
13             }
14         }
15         current = current->next;
16     }
17     return head;
18 }
19 123456789101112131415161718

```

方法二:

```

1 struct ListNode* removeDuplicateNodes(struct ListNode* head){
2     //判断是否需要遍历
3     if(head==NULL || head->next==NULL)
4         return head;
5     //利用val的值在0~20000之间, 标记是否出现过
6     int index[20001] = {0};
7     index[head->val] = 1;
8     struct ListNode *pre = head, *q = head->next;
9     while(q)
10    {
11        //当前val未出现过, 保存当前val, 修改pre和p指针
12        if(index[q->val]==0)
13        {
14            index[q->val] = 1;
15            pre = q;
16            q = q->next;
17        }
18    }
19 }

```

```

17     }
18     //当前值已经出现过了，删除当前节点
19     else
20     {
21         pre->next = q->next;
22         q = pre->next;
23     }
24 }
25 return head;
26 }
```

## 用普通算法实现两个有序链表的合并

### 题目描述

#### 力扣链接

将两个升序链表合并为一个新的升序链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。

示例：

输入：1->2->4, 1->3->4

输出：1->1->2->3->4->4

### 解题思路

#### 方法一：非递归

两个链表是排序过得，所以我们只需要同时遍历两个链表，比较链表元素的大小，将小的连接到新的链表中即可。最后，可能有一个链表会先遍历完，此时，我们直接将另一个链表连接到新链表的最后即可。

#### 方法二：递归

在两表都不为空的情况下，如果一个表当前value更小，把该表的next修改为（next）与（另一个表）中更小者

### 代码

#### 方法一：

```

1 struct ListNode* mergeTwoLists(struct ListNode* l1, struct ListNode* l2){
2
3
4     if(l1 == NULL)//判空
5         return l2;
6     if(l2 == NULL)
7         return l1;
8     struct ListNode* head = (struct ListNode*)malloc(sizeof(struct
9     ListNode));
10    struct ListNode* pre= head;
11    //先连接小的元素
12    while (l1 && l2) {
13        if (l1->val <= l2->val) {
14            pre->next = l1;
15            l1 = l1->next;
16        } else {
17            pre->next = l2;
18            l2 = l2->next;
19        }
20    }
21    return head;
22 }
```

```

18     }
19     pre = pre->next;
20 }
21 //将剩下元素直接接在最后面
22 pre->next = (NULL == l1 ? l2 : l1);
23 return head->next;
24 }
```

方法二：递归

```

1 /**
2 * Definition for singly-linked list.
3 * struct ListNode {
4 *     int val;
5 *     struct ListNode *next;
6 * };
7 */
8
9
10 struct ListNode* mergeTwoLists(struct ListNode* l1, struct ListNode* l2){
11     if(l1 == NULL) return l2;
12     if(l2 == NULL) return l1;
13     if(l1->val < l2->val){
14         l1->next = mergeTwoLists(l1 ->next, l2);
15         return l1;
16     }else{
17         l2->next = mergeTwoLists(l1, l2->next);
18         return l2;
19     }
20 }
```

< Empty Math Block >



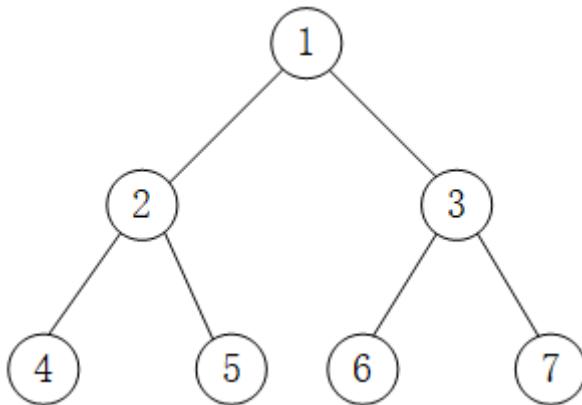
## 树

### 二叉树先序遍历（递归和非递归）

#### 先序遍历规则

先序遍历的核心思想：1.访问根节点；2.访问当前节点的左子树；3.若当前节点无左子树，则访问当前节点的右子树；即考察到一个节点后，即刻输出该节点的值，并继续遍历其左右子树。（根左右）

## 先序遍历举例



如图所示，采用先序遍历访问这颗二叉树的详细过程为：

- 1.访问该二叉树的根节点，找到 1；
- 2.访问节点 1 的左子树，找到节点 2；
- 3.访问节点 2 的左子树，找到节点 4；
- 4.由于访问节点 4 左子树失败，且也没有右子树，因此以节点 4 为根节点的子树遍历完成。但节点 2 还没有遍历其右子树，因此现在开始遍历，即访问节点 5；
- 5.由于节点 5 无左右子树，因此节点 5 遍历完成，并且由此以节点 2 为根节点的子树也遍历完成。现在回到节点 1，并开始遍历该节点的右子树，即访问节点 3；
- 6.访问节点 3 左子树，找到节点 6；
- 7.由于节点 6 无左右子树，因此节点 6 遍历完成，回到节点 3 并遍历其右子树，找到节点 7；
- 8.节点 7 无左右子树，因此以节点 3 为根节点的子树遍历完成，同时回归节点 1。由于节点 1 的左右子树全部遍历完成，因此整个二叉树遍历完成；

因此，图中二叉树采用先序遍历得到的序列为：1 2 4 5 3 6 7

## 先序遍历代码（递归）

```

1  /*
2   * @Description:
3   * @Version:
4   * @Autor: Carlos
5   * @Date: 2020-05-29 16:55:41
6   * @LastEditors: Carlos
7   * @LastEditTime: 2020-05-30 17:03:23
8   */
9  #include <stdio.h>
10 #include <string.h>
11 #include <stdlib.h>
12 #define TElmType int
13 //构造结点的结构体
14 typedef struct BiTNode{
15     TElmType data;//数据域
16     struct BiTNode *lchild,*rchild;//左右孩子指针
17 }BiTNode,*BiTree;
18
19 /**
20  * @Description: 初始化树的函数
  
```

```

21 * @Param: BiTree *T 结构体指针的指针
22 * @Return: 无
23 * @Author: Carlos
24 */
25 void CreateBiTree(BiTree *T){
26     *T=(BiTree)malloc(sizeof(BiTNode));
27     //根节点
28     (*T)->data=1;
29     (*T)->lchild=(BiTree)malloc(sizeof(BiTNode));
30     (*T)->rchild=(BiTree)malloc(sizeof(BiTNode));
31     //1节点的左孩子2
32     (*T)->lchild->data=2;
33     (*T)->lchild->lchild=(BiTree)malloc(sizeof(BiTNode));
34     (*T)->lchild->rchild=(BiTree)malloc(sizeof(BiTNode));
35     //2节点的右孩子5
36     (*T)->lchild->rchild->data=5;
37     (*T)->lchild->rchild->lchild=NULL;
38     (*T)->lchild->rchild->rchild=NULL;
39     //1节点的右孩子3
40     (*T)->rchild->data=3;
41     (*T)->rchild->lchild=(BiTree)malloc(sizeof(BiTNode));
42     //3节点的左孩子6
43     (*T)->rchild->lchild->data=6;
44     (*T)->rchild->lchild->lchild=NULL;
45     (*T)->rchild->lchild->rchild=NULL;
46     (*T)->rchild->rchild=(BiTree)malloc(sizeof(BiTNode));
47     //3节点的右孩子7
48     (*T)->rchild->rchild->data=7;
49     (*T)->rchild->rchild->lchild=NULL;
50     (*T)->rchild->rchild->rchild=NULL;
51     //2节点的左孩子4
52     (*T)->lchild->lchild->data=4;
53     (*T)->lchild->lchild->lchild=NULL;
54     (*T)->lchild->lchild->rchild=NULL;
55 }
56
57 /**
58 * @Description: 模拟操作结点元素的函数，输出结点本身的数值
59 * @Param:BiTree elem 就结构体指针
60 * @Return: 无
61 * @Author: Carlos
62 */
63 void PrintBiT(BiTree elem){
64     printf("%d ",elem->data);
65 }
66
67 /**
68 * @Description: 先序遍历
69 * @Param: BiTree T 结构体指针
70 * @Return: 无
71 * @Author: Carlos
72 */
73 void PreOrderTraverse(BiTree T){
74     if (T) {
75         PrintBiT(T); //调用操作结点数据的函数方法
76         PreOrderTraverse(T->lchild); //访问该结点的左孩子
77         PreOrderTraverse(T->rchild); //访问该结点的右孩子
78     }
79 }
80
81 /**
82 * @Description: 中序遍历
83 * @Param: BiTree T 结构体指针
84 * @Return: 无
85 * @Author: Carlos
86 */
87 void InOrderTraverse(BiTree T){
88     if (T) {
89         InOrderTraverse(T->lchild); //访问该结点的左孩子
90         PrintBiT(T); //调用操作结点数据的函数方法
91         InOrderTraverse(T->rchild); //访问该结点的右孩子
92     }
93 }
94
95 /**
96 * @Description: 后序遍历
97 * @Param: BiTree T 结构体指针
98 * @Return: 无
99 * @Author: Carlos
100 */
101 void PostOrderTraverse(BiTree T){
102     if (T) {
103         PostOrderTraverse(T->lchild); //访问该结点的左孩子
104         PostOrderTraverse(T->rchild); //访问该结点的右孩子
105         PrintBiT(T); //调用操作结点数据的函数方法
106     }
107 }
108
109 /**
110 * @Description: 层次遍历
111 * @Param: BiTree T 结构体指针
112 * @Return: 无
113 * @Author: Carlos
114 */
115 void LevelOrderTraverse(BiTree T){
116     if (T) {
117         Queue q;
118         InitQueue(&q);
119         EnQueue(&q,T);
120         while (!IsEmptyQueue(&q)) {
121             DeQueue(&q,T);
122             PrintBiT(T);
123             if (T->lchild) {
124                 EnQueue(&q,T->lchild);
125             }
126             if (T->rchild) {
127                 EnQueue(&q,T->rchild);
128             }
129         }
130     }
131 }
132
133 /**
134 * @Description: 深度优先搜索
135 * @Param: BiTree T 结构体指针
136 * @Return: 无
137 * @Author: Carlos
138 */
139 void DepthFirstSearch(BiTree T){
140     if (T) {
141         Stack s;
142         InitStack(&s);
143         Push(&s,T);
144         while (!IsEmptyStack(&s)) {
145             Pop(&s,T);
146             PrintBiT(T);
147             if (T->lchild) {
148                 Push(&s,T->lchild);
149             }
150             if (T->rchild) {
151                 Push(&s,T->rchild);
152             }
153         }
154     }
155 }
156
157 /**
158 * @Description: 宽度优先搜索
159 * @Param: BiTree T 结构体指针
160 * @Return: 无
161 * @Author: Carlos
162 */
163 void BreadthFirstSearch(BiTree T){
164     if (T) {
165         Queue q;
166         InitQueue(&q);
167         EnQueue(&q,T);
168         while (!IsEmptyQueue(&q)) {
169             DeQueue(&q,T);
170             PrintBiT(T);
171             if (T->lchild) {
172                 EnQueue(&q,T->lchild);
173             }
174             if (T->rchild) {
175                 EnQueue(&q,T->rchild);
176             }
177         }
178     }
179 }
180
181 /**
182 * @Description: 求二叉树的深度
183 * @Param: BiTree T 结构体指针
184 * @Return: 无
185 * @Author: Carlos
186 */
187 int TreeDepth(BiTree T){
188     if (T) {
189         int ldepth = TreeDepth(T->lchild);
190         int rdepth = TreeDepth(T->rchild);
191         if (ldepth > rdepth) {
192             return ldepth + 1;
193         } else {
194             return rdepth + 1;
195         }
196     } else {
197         return 0;
198     }
199 }
200
201 /**
202 * @Description: 求二叉树的节点数
203 * @Param: BiTree T 结构体指针
204 * @Return: 无
205 * @Author: Carlos
206 */
207 int TreeCount(BiTree T){
208     if (T) {
209         int lcount = TreeCount(T->lchild);
210         int rcount = TreeCount(T->rchild);
211         return lcount + rcount + 1;
212     } else {
213         return 0;
214     }
215 }
216
217 /**
218 * @Description: 求二叉树的叶子数
219 * @Param: BiTree T 结构体指针
220 * @Return: 无
221 * @Author: Carlos
222 */
223 int LeafCount(BiTree T){
224     if (T) {
225         int lcount = LeafCount(T->lchild);
226         int rcount = LeafCount(T->rchild);
227         if (T->lchild == NULL && T->rchild == NULL) {
228             return 1;
229         } else {
230             return lcount + rcount;
231         }
232     } else {
233         return 0;
234     }
235 }
236
237 /**
238 * @Description: 求二叉树的分支数
239 * @Param: BiTree T 结构体指针
240 * @Return: 无
241 * @Author: Carlos
242 */
243 int BranchCount(BiTree T){
244     if (T) {
245         int lcount = BranchCount(T->lchild);
246         int rcount = BranchCount(T->rchild);
247         if (T->lchild != NULL && T->rchild != NULL) {
248             return lcount + rcount + 1;
249         } else {
250             return lcount + rcount;
251         }
252     } else {
253         return 0;
254     }
255 }
256
257 /**
258 * @Description: 求二叉树的满度数
259 * @Param: BiTree T 结构体指针
260 * @Return: 无
261 * @Author: Carlos
262 */
263 int FullDegree(BiTree T){
264     if (T) {
265         int ldegree = FullDegree(T->lchild);
266         int rdegree = FullDegree(T->rchild);
267         if (T->lchild == NULL && T->rchild == NULL) {
268             return 0;
269         } else {
270             return ldegree + rdegree + 1;
271         }
272     } else {
273         return 0;
274     }
275 }
276
277 /**
278 * @Description: 求二叉树的空度数
279 * @Param: BiTree T 结构体指针
280 * @Return: 无
281 * @Author: Carlos
282 */
283 int EmptyDegree(BiTree T){
284     if (T) {
285         int ldegree = EmptyDegree(T->lchild);
286         int rdegree = EmptyDegree(T->rchild);
287         if (T->lchild != NULL && T->rchild != NULL) {
288             return 0;
289         } else {
290             return ldegree + rdegree + 1;
291         }
292     } else {
293         return 0;
294     }
295 }
296
297 /**
298 * @Description: 求二叉树的平衡度数
299 * @Param: BiTree T 结构体指针
300 * @Return: 无
301 * @Author: Carlos
302 */
303 int BalanceDegree(BiTree T){
304     if (T) {
305         int ldegree = BalanceDegree(T->lchild);
306         int rdegree = BalanceDegree(T->rchild);
307         if (ldegree - rdegree == 1 || rdegree - ldegree == 1) {
308             return 1;
309         } else {
310             return 0;
311         }
312     } else {
313         return 0;
314     }
315 }
316
317 /**
318 * @Description: 求二叉树的平衡性
319 * @Param: BiTree T 结构体指针
320 * @Return: 无
321 * @Author: Carlos
322 */
323 int Balance(BiTree T){
324     if (T) {
325         int lbalance = BalanceDegree(T->lchild);
326         int rbalance = BalanceDegree(T->rchild);
327         if (lbalance == 1 && rbalance == 1) {
328             return 1;
329         } else {
330             return 0;
331         }
332     } else {
333         return 0;
334     }
335 }
336
337 /**
338 * @Description: 求二叉树的对称性
339 * @Param: BiTree T 结构体指针
340 * @Return: 无
341 * @Author: Carlos
342 */
343 int Symmetric(BiTree T){
344     if (T) {
345         int lsymmetric = Symmetric(T->lchild);
346         int rsymmetric = Symmetric(T->rchild);
347         if (lsymmetric == 1 && rsymmetric == 1) {
348             return 1;
349         } else {
350             return 0;
351         }
352     } else {
353         return 0;
354     }
355 }
356
357 /**
358 * @Description: 求二叉树的完全性
359 * @Param: BiTree T 结构体指针
360 * @Return: 无
361 * @Author: Carlos
362 */
363 int Complete(BiTree T){
364     if (T) {
365         int lcomplete = Complete(T->lchild);
366         int rcomplete = Complete(T->rchild);
367         if (lcomplete == 1 && rcomplete == 1) {
368             return 1;
369         } else {
370             return 0;
371         }
372     } else {
373         return 0;
374     }
375 }
376
377 /**
378 * @Description: 求二叉树的深度
379 * @Param: BiTree T 结构体指针
380 * @Return: 无
381 * @Author: Carlos
382 */
383 int Depth(BiTree T){
384     if (T) {
385         int ldepth = Depth(T->lchild);
386         int rdepth = Depth(T->rchild);
387         if (ldepth > rdepth) {
388             return ldepth + 1;
389         } else {
390             return rdepth + 1;
391         }
392     } else {
393         return 0;
394     }
395 }
396
397 /**
398 * @Description: 求二叉树的节点数
399 * @Param: BiTree T 结构体指针
400 * @Return: 无
401 * @Author: Carlos
402 */
403 int Count(BiTree T){
404     if (T) {
405         int lcount = Count(T->lchild);
406         int rcount = Count(T->rchild);
407         return lcount + rcount + 1;
408     } else {
409         return 0;
410     }
411 }
412
413 /**
414 * @Description: 求二叉树的叶子数
415 * @Param: BiTree T 结构体指针
416 * @Return: 无
417 * @Author: Carlos
418 */
419 int Leaf(BiTree T){
420     if (T) {
421         int lcount = Leaf(T->lchild);
422         int rcount = Leaf(T->rchild);
423         if (T->lchild == NULL && T->rchild == NULL) {
424             return 1;
425         } else {
426             return lcount + rcount;
427         }
428     } else {
429         return 0;
430     }
431 }
432
433 /**
434 * @Description: 求二叉树的分支数
435 * @Param: BiTree T 结构体指针
436 * @Return: 无
437 * @Author: Carlos
438 */
439 int Branch(BiTree T){
440     if (T) {
441         int lcount = Branch(T->lchild);
442         int rcount = Branch(T->rchild);
443         if (T->lchild != NULL && T->rchild != NULL) {
444             return lcount + rcount + 1;
445         } else {
446             return lcount + rcount;
447         }
448     } else {
449         return 0;
450     }
451 }
452
453 /**
454 * @Description: 求二叉树的满度数
455 * @Param: BiTree T 结构体指针
456 * @Return: 无
457 * @Author: Carlos
458 */
459 int Full(BiTree T){
460     if (T) {
461         int ldegree = Full(T->lchild);
462         int rdegree = Full(T->rchild);
463         if (T->lchild == NULL && T->rchild == NULL) {
464             return 0;
465         } else {
466             return ldegree + rdegree + 1;
467         }
468     } else {
469         return 0;
470     }
471 }
472
473 /**
474 * @Description: 求二叉树的空度数
475 * @Param: BiTree T 结构体指针
476 * @Return: 无
477 * @Author: Carlos
478 */
479 int Empty(BiTree T){
480     if (T) {
481         int ldegree = Empty(T->lchild);
482         int rdegree = Empty(T->rchild);
483         if (T->lchild != NULL && T->rchild != NULL) {
484             return 0;
485         } else {
486             return ldegree + rdegree + 1;
487         }
488     } else {
489         return 0;
490     }
491 }
492
493 /**
494 * @Description: 求二叉树的平衡度数
495 * @Param: BiTree T 结构体指针
496 * @Return: 无
497 * @Author: Carlos
498 */
499 int BalanceDegree(BiTree T){
500     if (T) {
501         int ldegree = BalanceDegree(T->lchild);
502         int rdegree = BalanceDegree(T->rchild);
503         if (ldegree - rdegree == 1 || rdegree - ldegree == 1) {
504             return 1;
505         } else {
506             return 0;
507         }
508     } else {
509         return 0;
510     }
511 }
512
513 /**
514 * @Description: 求二叉树的平衡性
515 * @Param: BiTree T 结构体指针
516 * @Return: 无
517 * @Author: Carlos
518 */
519 int Balance(BiTree T){
520     if (T) {
521         int lbalance = BalanceDegree(T->lchild);
522         int rbalance = BalanceDegree(T->rchild);
523         if (lbalance == 1 && rbalance == 1) {
524             return 1;
525         } else {
526             return 0;
527         }
528     } else {
529         return 0;
530     }
531 }
532
533 /**
534 * @Description: 求二叉树的对称性
535 * @Param: BiTree T 结构体指针
536 * @Return: 无
537 * @Author: Carlos
538 */
539 int Symmetric(BiTree T){
540     if (T) {
541         int lsymmetric = Symmetric(T->lchild);
542         int rsymmetric = Symmetric(T->rchild);
543         if (lsymmetric == 1 && rsymmetric == 1) {
544             return 1;
545         } else {
546             return 0;
547         }
548     } else {
549         return 0;
550     }
551 }
552
553 /**
554 * @Description: 求二叉树的完全性
555 * @Param: BiTree T 结构体指针
556 * @Return: 无
557 * @Author: Carlos
558 */
559 int Complete(BiTree T){
560     if (T) {
561         int lcomplete = Complete(T->lchild);
562         int rcomplete = Complete(T->rchild);
563         if (lcomplete == 1 && rcomplete == 1) {
564             return 1;
565         } else {
566             return 0;
567         }
568     } else {
569         return 0;
570     }
571 }
572
573 /**
574 * @Description: 求二叉树的深度
575 * @Param: BiTree T 结构体指针
576 * @Return: 无
577 * @Author: Carlos
578 */
579 int Depth(BiTree T){
580     if (T) {
581         int ldepth = Depth(T->lchild);
582         int rdepth = Depth(T->rchild);
583         if (ldepth > rdepth) {
584             return ldepth + 1;
585         } else {
586             return rdepth + 1;
587         }
588     } else {
589         return 0;
590     }
591 }
592
593 /**
594 * @Description: 求二叉树的节点数
595 * @Param: BiTree T 结构体指针
596 * @Return: 无
597 * @Author: Carlos
598 */
599 int Count(BiTree T){
600     if (T) {
601         int lcount = Count(T->lchild);
602         int rcount = Count(T->rchild);
603         return lcount + rcount + 1;
604     } else {
605         return 0;
606     }
607 }
608
609 /**
610 * @Description: 求二叉树的叶子数
611 * @Param: BiTree T 结构体指针
612 * @Return: 无
613 * @Author: Carlos
614 */
615 int Leaf(BiTree T){
616     if (T) {
617         int lcount = Leaf(T->lchild);
618         int rcount = Leaf(T->rchild);
619         if (T->lchild == NULL && T->rchild == NULL) {
620             return 1;
621         } else {
622             return lcount + rcount;
623         }
624     } else {
625         return 0;
626     }
627 }
628
629 /**
630 * @Description: 求二叉树的分支数
631 * @Param: BiTree T 结构体指针
632 * @Return: 无
633 * @Author: Carlos
634 */
635 int Branch(BiTree T){
636     if (T) {
637         int lcount = Branch(T->lchild);
638         int rcount = Branch(T->rchild);
639         if (T->lchild != NULL && T->rchild != NULL) {
640             return lcount + rcount + 1;
641         } else {
642             return lcount + rcount;
643         }
644     } else {
645         return 0;
646     }
647 }
648
649 /**
650 * @Description: 求二叉树的满度数
651 * @Param: BiTree T 结构体指针
652 * @Return: 无
653 * @Author: Carlos
654 */
655 int Full(BiTree T){
656     if (T) {
657         int ldegree = Full(T->lchild);
658         int rdegree = Full(T->rchild);
659         if (T->lchild == NULL && T->rchild == NULL) {
660             return 0;
661         } else {
662             return ldegree + rdegree + 1;
663         }
664     } else {
665         return 0;
666     }
667 }
668
669 /**
670 * @Description: 求二叉树的空度数
671 * @Param: BiTree T 结构体指针
672 * @Return: 无
673 * @Author: Carlos
674 */
675 int Empty(BiTree T){
676     if (T) {
677         int ldegree = Empty(T->lchild);
678         int rdegree = Empty(T->rchild);
679         if (T->lchild != NULL && T->rchild != NULL) {
680             return 0;
681         } else {
682             return ldegree + rdegree + 1;
683         }
684     } else {
685         return 0;
686     }
687 }
688
689 /**
690 * @Description: 求二叉树的平衡度数
691 * @Param: BiTree T 结构体指针
692 * @Return: 无
693 * @Author: Carlos
694 */
695 int BalanceDegree(BiTree T){
696     if (T) {
697         int ldegree = BalanceDegree(T->lchild);
698         int rdegree = BalanceDegree(T->rchild);
699         if (ldegree - rdegree == 1 || rdegree - ldegree == 1) {
700             return 1;
701         } else {
702             return 0;
703         }
704     } else {
705         return 0;
706     }
707 }
708
709 /**
710 * @Description: 求二叉树的平衡性
711 * @Param: BiTree T 结构体指针
712 * @Return: 无
713 * @Author: Carlos
714 */
715 int Balance(BiTree T){
716     if (T) {
717         int lbalance = BalanceDegree(T->lchild);
718         int rbalance = BalanceDegree(T->rchild);
719         if (lbalance == 1 && rbalance == 1) {
720             return 1;
721         } else {
722             return 0;
723         }
724     } else {
725         return 0;
726     }
727 }
728
729 /**
730 * @Description: 求二叉树的对称性
731 * @Param: BiTree T 结构体指针
732 * @Return: 无
733 * @Author: Carlos
734 */
735 int Symmetric(BiTree T){
736     if (T) {
737         int lsymmetric = Symmetric(T->lchild);
738         int rsymmetric = Symmetric(T->rchild);
739         if (lsymmetric == 1 && rsymmetric == 1) {
740             return 1;
741         } else {
742             return 0;
743         }
744     } else {
745         return 0;
746     }
747 }
748
749 /**
750 * @Description: 求二叉树的完全性
751 * @Param: BiTree T 结构体指针
752 * @Return: 无
753 * @Author: Carlos
754 */
755 int Complete(BiTree T){
756     if (T) {
757         int lcomplete = Complete(T->lchild);
758         int rcomplete = Complete(T->rchild);
759         if (lcomplete == 1 && rcomplete == 1) {
760             return 1;
761         } else {
762             return 0;
763         }
764     } else {
765         return 0;
766     }
767 }
768
769 /**
770 * @Description: 求二叉树的深度
771 * @Param: BiTree T 结构体指针
772 * @Return: 无
773 * @Author: Carlos
774 */
775 int Depth(BiTree T){
776     if (T) {
777         int ldepth = Depth(T->lchild);
778         int rdepth = Depth(T->rchild);
779         if (ldepth > rdepth) {
780             return ldepth + 1;
781         } else {
782             return rdepth + 1;
783         }
784     } else {
785         return 0;
786     }
787 }
788
789 /**
790 * @Description: 求二叉树的节点数
791 * @Param: BiTree T 结构体指针
792 * @Return: 无
793 * @Author: Carlos
794 */
795 int Count(BiTree T){
796     if (T) {
797         int lcount = Count(T->lchild);
798         int rcount = Count(T->rchild);
799         return lcount + rcount + 1;
800     } else {
801         return 0;
802     }
803 }
804
805 /**
806 * @Description: 求二叉树的叶子数
807 * @Param: BiTree T 结构体指针
808 * @Return: 无
809 * @Author: Carlos
810 */
811 int Leaf(BiTree T){
812     if (T) {
813         int lcount = Leaf(T->lchild);
814         int rcount = Leaf(T->rchild);
815         if (T->lchild == NULL && T->rchild == NULL) {
816             return 1;
817         } else {
818             return lcount + rcount;
819         }
820     } else {
821         return 0;
822     }
823 }
824
825 /**
826 * @Description: 求二叉树的分支数
827 * @Param: BiTree T 结构体指针
828 * @Return: 无
829 * @Author: Carlos
830 */
831 int Branch(BiTree T){
832     if (T) {
833         int lcount = Branch(T->lchild);
834         int rcount = Branch(T->rchild);
835         if (T->lchild != NULL && T->rchild != NULL) {
836             return lcount + rcount + 1;
837         } else {
838             return lcount + rcount;
839         }
840     } else {
841         return 0;
842     }
843 }
844
845 /**
846 * @Description: 求二叉树的满度数
847 * @Param: BiTree T 结构体指针
848 * @Return: 无
849 * @Author: Carlos
850 */
851 int Full(BiTree T){
852     if (T) {
853         int ldegree = Full(T->lchild);
854         int rdegree = Full(T->rchild);
855         if (T->lchild == NULL && T->rchild == NULL) {
856             return 0;
857         } else {
858             return ldegree + rdegree + 1;
859         }
860     } else {
861         return 0;
862     }
863 }
864
865 /**
866 * @Description: 求二叉树的空度数
867 * @Param: BiTree T 结构体指针
868 * @Return: 无
869 * @Author: Carlos
870 */
871 int Empty(BiTree T){
872     if (T) {
873         int ldegree = Empty(T->lchild);
874         int rdegree = Empty(T->rchild);
875         if (T->lchild != NULL && T->rchild != NULL) {
876             return 0;
877         } else {
878             return ldegree + rdegree + 1;
879         }
880     } else {
881         return 0;
882     }
883 }
884
885 /**
886 * @Description: 求二叉树的平衡度数
887 * @Param: BiTree T 结构体指针
888 * @Return: 无
889 * @Author: Carlos
890 */
891 int BalanceDegree(BiTree T){
892     if (T) {
893         int ldegree = BalanceDegree(T->lchild);
894         int rdegree = BalanceDegree(T->rchild);
895         if (ldegree - rdegree == 1 || rdegree - ldegree == 1) {
896             return 1;
897         } else {
898             return 0;
899         }
900     } else {
901         return 0;
902     }
903 }
904
905 /**
906 * @Description: 求二叉树的平衡性
907 * @Param: BiTree T 结构体指针
908 * @Return: 无
909 * @Author: Carlos
910 */
911 int Balance(BiTree T){
912     if (T) {
913         int lbalance = BalanceDegree(T->lchild);
914         int rbalance = BalanceDegree(T->rchild);
915         if (lbalance == 1 && rbalance == 1) {
916             return 1;
917         } else {
918             return 0;
919         }
920     } else {
921         return 0;
922     }
923 }
924
925 /**
926 * @Description: 求二叉树的对称性
927 * @Param: BiTree T 结构体指针
928 * @Return: 无
929 * @Author: Carlos
930 */
931 int Symmetric(BiTree T){
932     if (T) {
933         int lsymmetric = Symmetric(T->lchild);
934         int rsymmetric = Symmetric(T->rchild);
935         if (lsymmetric == 1 && rsymmetric == 1) {
936             return 1;
937         } else {
938             return 0;
939         }
940     } else {
941         return 0;
942     }
943 }
944
945 /**
946 * @Description: 求二叉树的完全性
947 * @Param: BiTree T 结构体指针
948 * @Return: 无
949 * @Author: Carlos
950 */
951 int Complete(BiTree T){
952     if (T) {
953         int lcomplete = Complete(T->lchild);
954         int rcomplete = Complete(T->rchild);
955         if (lcomplete == 1 && rcomplete == 1) {
956             return 1;
957         } else {
958             return 0;
959         }
960     } else {
961         return 0;
962     }
963 }
964
965 /**
966 * @Description: 求二叉树的深度
967 * @Param: BiTree T 结构体指针
968 * @Return: 无
969 * @Author: Carlos
970 */
971 int Depth(BiTree T){
972     if (T) {
973         int ldepth = Depth(T->lchild);
974         int rdepth = Depth(T->rchild);
975         if (ldepth > rdepth) {
976             return ldepth + 1;
977         } else {
978             return rdepth + 1;
979         }
980     } else {
981         return 0;
982     }
983 }
984
985 /**
986 * @Description: 求二叉树的节点数
987 * @Param: BiTree T 结构体指针
988 * @Return: 无
989 * @Author: Carlos
990 */
991 int Count(BiTree T){
992     if (T) {
993         int lcount = Count(T->lchild);
994         int rcount = Count(T->rchild);
995         return lcount + rcount + 1;
996     } else {
997         return 0;
998     }
999 }
1000
1001 /**
1002 * @Description: 求二叉树的叶子数
1003 * @Param: BiTree T 结构体指针
1004 * @Return: 无
1005 * @Author: Carlos
1006 */
1007 int Leaf(BiTree T){
1008     if (T) {
1009         int lcount = Leaf(T->lchild);
1010         int rcount = Leaf(T->rchild);
1011         if (T->lchild == NULL && T->rchild == NULL) {
1012             return 1;
1013         } else {
1014             return lcount + rcount;
1015         }
1016     } else {
1017         return 0;
1018     }
1019 }
1020
1021 /**
1022 * @Description: 求二叉树的分支数
1023 * @Param: BiTree T 结构体指针
1024 * @Return: 无
1025 * @Author: Carlos
1026 */
1027 int Branch(BiTree T){
1028     if (T) {
1029         int lcount = Branch(T->lchild);
1030         int rcount = Branch(T->rchild);
1031         if (T->lchild != NULL && T->rchild != NULL) {
1032             return lcount + rcount + 1;
1033         } else {
1034             return lcount + rcount;
1035         }
1036     } else {
1037         return 0;
1038     }
1039 }
1040
1041 /**
1042 * @Description: 求二叉树的满度数
1043 * @Param: BiTree T 结构体指针
1044 * @Return: 无
1045 * @Author: Carlos
1046 */
1047 int Full(BiTree T){
1048     if (T) {
1049         int ldegree = Full(T->lchild);
1050         int rdegree = Full(T->rchild);
1051         if (T->lchild == NULL && T->rchild == NULL) {
1052             return 0;
1053         } else {
1054             return ldegree + rdegree + 1;
1055         }
1056     } else {
1057         return 0;
1058     }
1059 }
1060
1061 /**
1062 * @Description: 求二叉树的空度数
1063 * @Param: BiTree T 结构体指针
1064 * @Return: 无
1065 * @Author: Carlos
1066 */
1067 int Empty(BiTree T){
1068     if (T) {
1069         int ldegree = Empty(T->lchild);
1070         int rdegree = Empty(T->rchild);
1071         if (T->lchild != NULL && T->rchild != NULL) {
1072             return 0;
1073         } else {
1074             return ldegree + rdegree + 1;
1075         }
1076     } else {
1077         return 0;
1078     }
1079 }
1080
1081 /**
1082 * @Description: 求二叉树的平衡度数
1083 * @Param: BiTree T 结构体指针
1084 * @Return: 无
1085 * @Author: Carlos
1086 */
1087 int BalanceDegree(BiTree T){
1088     if (T) {
1089         int ldegree = BalanceDegree(T->lchild);
1090         int rdegree = BalanceDegree(T->rchild);
1091         if (ldegree - rdegree == 1 || rdegree - ldegree == 1) {
1092             return 1;
1093         } else {
1094             return 0;
1095         }
1096     } else {
1097         return 0;
1098     }
1099 }
1100
1101 /**
1102 * @Description: 求二叉树的平衡性
1103 * @Param: BiTree T 结构体指针
1104 * @Return: 无
1105 * @Author: Carlos
1106 */
1107 int Balance(BiTree T){
1108     if (T) {
1109         int lbalance = BalanceDegree(T->lchild);
1110         int rbalance = BalanceDegree(T->rchild);
1111         if (lbalance == 1 && rbalance == 1) {
1112             return 1;
1113         } else {
1114             return 0;
1115         }
1116     } else {
1117         return 0;
1118     }
1119 }
1120
1121 /**
1122 * @Description: 求二叉树的对称性
1123 * @Param: BiTree T 结构体指针
1124 * @Return: 无
1125 * @Author: Carlos
1126 */
1127 int Symmetric(BiTree T){
1128     if (T) {
1129         int lsymmetric = Symmetric(T->lchild);
1
```

```

79 }
80 //如果结点为空，返回上一层
81 return;
82 }
83 int main() {
84 BiTree Tree;
85 CreateBiTree(&Tree);
86 printf("先序遍历: \n");
87 PreOrderTraverse(Tree);
88 }

```

### 先序遍历代码（非递归）

因为要在遍历完某个树的根节点的左子树后接着遍历节点的右子树，为了能找到该树的根节点，需要使用栈来进行暂存。中序和后序也都涉及到回溯，所以都需要用到栈。

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #define TElmType int
5 //构造结点的结构体
6 typedef struct BiTNode{
7     TElmType data;//数据域
8     struct BiTNode *lchild,*rchild;//左右孩子指针
9 }BiTNode,*BiTree;
10 int top = -1;
11 //定义一个顺序栈
12 BiTree a[20];
13 /**
14 * @Description: 初始化树
15 * @Param: BiTree *T 指针的指针
16 * @Return: 无
17 * @Author: Carlos
18 */
19 void CreateBiTree(BiTree *T){
20     *T=(BiTree)malloc(sizeof(BiTNode));
21     //根节点
22     (*T)->data=1;
23     (*T)->lchild=(BiTree)malloc(sizeof(BiTNode));
24     (*T)->rchild=(BiTree)malloc(sizeof(BiTNode));
25     //1节点的左孩子2
26     (*T)->lchild->data=2;
27     (*T)->lchild->lchild=(BiTree)malloc(sizeof(BiTNode));
28     (*T)->lchild->rchild=(BiTree)malloc(sizeof(BiTNode));
29     //2节点的右孩子5
30     (*T)->lchild->rchild->data=5;
31     (*T)->lchild->rchild->lchild=NULL;
32     (*T)->lchild->rchild->rchild=NULL;
33     //1节点的右孩子3
34     (*T)->rchild->data=3;
35     (*T)->rchild->lchild=(BiTree)malloc(sizeof(BiTNode));
36     //3节点的左孩子6
37     (*T)->rchild->lchild->data=6;
38     (*T)->rchild->lchild->lchild=NULL;
39     (*T)->rchild->lchild->rchild=NULL;
40     (*T)->rchild->rchild=(BiTree)malloc(sizeof(BiTNode));
41     //3节点的右孩子7

```

```

42     (*T->rchild->rchild->data=7;
43     (*T->rchild->rchild->lchild=NULL;
44     (*T->rchild->rchild->rchild=NULL;
45     //2节点的左孩子4
46     (*T->lchild->lchild->data=4;
47     (*T->lchild->lchild->lchild=NULL;
48     (*T->lchild->lchild->rchild=NULL;
49 }
50 /**
51 * @Description: 打印二叉树
52 * @Param: BiTree elem 指针的指针
53 * @Return: 无
54 * @Author: Carlos
55 */
56 void PrintBiT(BiTree elem){
57     printf("%d ",elem->data);
58 }
59 /**
60 * @Description: 二叉树压栈函数
61 * @Param: BiTree* a 结构体指针的指针（也可以理解为指针数组） BiTree elem 结构体指
62 * @Return: 无
63 * @Author: Carlos
64 */
65 void Push(BiTree* a,BiTree elem)
66 {
67     a[++top]=elem;
68 }
69 /**
70 * @Description: 二叉树弹栈函数
71 * @Param: 无
72 * @Return: 无
73 * @Author: Carlos
74 */
75 void Pop()
76 {
77     if (top== -1) {
78         return ;
79     }
80     top--;
81 }
82 /**
83 * @Description: 获取栈顶元素
84 * @Param: BiTree*a 结构体指针数组
85 * @Return: 结构体指针
86 * @Author: Carlos
87 */
88 BiTree GetTop(BiTree*a){
89     return a[top];
90 }
91 /**
92 * @Description: 先序遍历
93 * @Param: BiTree Tree 结构体指针
94 * @Return: 无
95 * @Author: Carlos
96 */
97 void PreOrderTraverse(BiTree Tree)
98 {

```

```

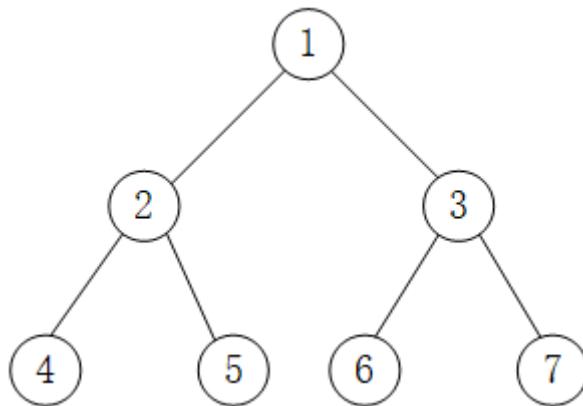
99 //临时指针
100 BiTree p;
101 //根结点进栈
102 Push(a, Tree);
103 while (top!=-1) {
104     //取栈顶元素
105     p=GetTop(a);
106     //弹栈
107     Pop();
108     while (p) {
109         //调用结点的操作函数
110         PrintBiT(p);
111         //如果该结点有右孩子，右孩子进栈
112         if (p->rchild) {
113             Push(a,p->rchild);
114         }
115         p=p->lchild;//一直指向根结点最后一个左孩子
116     }
117 }
118 }
119 int main() {
120     BiTree Tree;
121     CreateBiTree(&Tree);
122     printf("先序遍历: \n");
123     PreOrderTraverse(Tree);
124 }
```

## 二叉树中序遍历（递归和非递归）

### 中序遍历规则

二叉树中序遍历的实现思想是：1.访问当前节点的左子树；2.访问根节点；3.访问当前节点的右子树。即考察到一个节点后，将其暂存，遍历完左子树后，再输出该节点的值，然后遍历右子树。（左根右）

### 中序遍历举例



以上图为例，采用中序遍历的思想遍历该二叉树的过程为：

- 1.访问该二叉树的根节点，找到 1；
- 2.遍历节点 1 的左子树，找到节点 2；
- 3.遍历节点 2 的左子树，找到节点 4；
- 4.由于节点 4 无左孩子，因此找到节点 4，并遍历节点 4 的右子树；
- 5.遍历节点 2 的右子树，找到节点 5；

- 6.由于节点 5 无左子树，因此访问节点 5，又因为节点 5 没有右子树，因此节点 1 的左子树遍历完成，访问节点 1，并遍历节点 1 的右子树，找到节点 3；
- 7.遍历节点 3 的左子树，找到节点 6；
- 8.由于节点 6 无左子树，因此访问节点 6，又因为该节点无右子树，因此节点 3 的左子树遍历完成，开始访问节点 3，并遍历节点 3 的右子树，找到节点 7；
- 9.由于节点 7 无左子树，因此访问节点 7，又因为该节点无右子树，因此节点 1 的右子树遍历完成，即整棵树遍历完成；

因此，上图中二叉树采用中序遍历得到的序列为：4 2 5 1 6 3 7

### 中序遍历代码（递归）

```

1  /*
2   * @Description: 递归实现的中序遍历
3   * @Version: V1.0
4   * @Autor: Carlos
5   * @Date: 2020-05-18 14:53:29
6   * @LastEditors: Carlos
7   * @LastEditTime: 2020-05-30 17:21:06
8   */
9 #include <stdio.h>
10 #include <string.h>
11 #include <stdlib.h>
12 #define TElmType int
13 //构造结点的结构体
14 typedef struct BiTNode{
15     //数据域
16     TElmType data;
17     //左右孩子指针
18     struct BiTree* lchild,*rchild;
19 }BiTNode,*BiTree;
20 /**
21  * @Description: 初始化树
22  * @Param: BiTree *T 结构体指针的指针（指针数组）
23  * @Return: 无
24  * @Author: Carlos
25  */
26 void CreateBiTree(BiTTree *T){
27     *T=(BiTree)malloc(sizeof(BiTNode));
28     (*T)->data=1;
29     (*T)->lchild=(BiTree)malloc(sizeof(BiTNode));
30     (*T)->rchild=(BiTree)malloc(sizeof(BiTNode));
31
32     (*T)->lchild->data=2;
33     (*T)->lchild->lchild=(BiTree)malloc(sizeof(BiTNode));
34     (*T)->lchild->rchild=(BiTree)malloc(sizeof(BiTNode));
35     (*T)->lchild->rchild->data=5;
36     (*T)->lchild->rchild->lchild=NULL;
37     (*T)->lchild->rchild->rchild=NULL;
38     (*T)->rchild->data=3;
39     (*T)->rchild->lchild=(BiTree)malloc(sizeof(BiTNode));
40     (*T)->rchild->lchild->data=6;
41     (*T)->rchild->lchild->lchild=NULL;
42     (*T)->rchild->lchild->rchild=NULL;
43     (*T)->rchild->rchild=(BiTree)malloc(sizeof(BiTNode));

```

```

44     (*T->rchild->rchild->data=7;
45     (*T->rchild->rchild->lchild=NULL;
46     (*T->rchild->rchild->rchild=NULL;
47     (*T->lchild->lchild->lchild=4;
48     (*T->lchild->lchild->lchild=NULL;
49     (*T->lchild->lchild->rchild=NULL;
50 }
51 /**
52 * @Description: 显示函数
53 * @Param: BiTree elem 结构体指针
54 * @Return: 无
55 * @Author: Carlos
56 */
57 void PrintBiT(BiTree elem){
58     printf("%d ",elem->data);
59 }
60 /**
61 * @Description: 中序遍历
62 * @Param: BiTree T 结构体指针
63 * @Return: 无
64 * @Author: Carlos
65 */
66 void INOrderTraverse(BiTree T){
67     if (T) {
68         INOrderTraverse(T->lchild); //遍历左孩子
69         PrintBiT(T); //调用操作结点数据的函数方法
70         INOrderTraverse(T->rchild); //遍历右孩子
71     }
72     //如果结点为空，返回上一层
73     return;
74 }
75
76 int main() {
77     BiTree Tree;
78     CreateBiTree(&Tree);
79     printf("中序遍历算法: \n");
80     INOrderTraverse(Tree);
81 }
```

## 中序遍历代码（非递归）

和非递归先序遍历类似，唯一区别是考查到当前节点时，并不直接输出该节点。而是当考查节点为空时，从栈中弹出的时候再进行输出(永远先考虑左子树，直到左子树为空才访问根节点)。

```

1 /*
2 * @Description: 二叉树的先序遍历（非递归）
3 * @Version: v1.0
4 * @Author: Carlos
5 * @Date: 2020-05-17 16:35:27
6 * @LastEditors: Carlos
7 * @LastEditTime: 2020-05-18 14:51:01
8 */
9 #include <stdio.h>
10 #include <string.h>
11 #include <stdlib.h>
12 #define DBG_PRINTF(fmt, args...) \
13 do\
```

```

14     {\ \
15         printf("<<File:%s  Line:%d  Function:%s>> ", __FILE__, __LINE__, \
16             __FUNCTION__); \
17         printf(fmt, ##args); \
18     }while(0)
19 #define TElmType int
20 int top=-1;//top变量时刻表示栈顶元素所在位置
21 //构造结点的结构体
22 typedef struct BiTNode{
23     //数据域
24     TElmType data;
25     //左右孩子指针
26     struct BiTNode *lchild,*rchild;
27 }BiTNode,*BiTree;
28 /**
29 * @Description: 初始化树
30 * @Param: BiTree *T 结构体指针的指针
31 * @Return: 无
32 * @Author: Carlos
33 */
34 void CreateBiTree(BiTree *T){
35     *T=(BiTree)malloc(sizeof(BiTNode));
36     (*T)->data=1;
37     (*T)->lchild=(BiTree)malloc(sizeof(BiTNode));
38     (*T)->rchild=(BiTree)malloc(sizeof(BiTNode));
39     (*T)->lchild->data=2;
40     (*T)->lchild->lchild=(BiTree)malloc(sizeof(BiTNode));
41     (*T)->lchild->rchild=(BiTree)malloc(sizeof(BiTNode));
42     (*T)->lchild->rchild->data=5;
43     (*T)->lchild->rchild->lchild=NULL;
44     (*T)->lchild->rchild->rchild=NULL;
45     (*T)->rchild->data=3;
46     (*T)->rchild->lchild=(BiTree)malloc(sizeof(BiTNode));
47     (*T)->rchild->lchild->data=6;
48     (*T)->rchild->lchild->lchild=NULL;
49     (*T)->rchild->lchild->rchild=NULL;
50     (*T)->rchild->rchild=(BiTree)malloc(sizeof(BiTNode));
51     (*T)->rchild->rchild->data=7;
52     (*T)->rchild->rchild->lchild=NULL;
53     (*T)->rchild->rchild->rchild=NULL;
54     (*T)->lchild->lchild->data=4;
55     (*T)->lchild->lchild->lchild=NULL;
56     (*T)->lchild->lchild->rchild=NULL;
57 }
58 /**
59 * @Description: 中序遍历使用的进栈函数
60 * @Param: BiTree* a 指向树的指针数组 BiTree elem 进栈的元素
61 * @Return: 无
62 * @Author: Carlos
63 */
64 void Push(BiTree* a,BiTree elem){
65     //指针进栈
66     a[++top]=elem;
67 }
68 /**
69 * @Description: 前序遍历使用的弹栈函数
70 * @Param: 无
71 * @Return: 无

```

```

71 * @Author: Carlos
72 */
73 void Pop( ){
74     if (top== -1) {
75         return;
76     }
77     top--;
78 }
79 /**
80 * @Description: 显示函数
81 * @Param: BiTree elem 指向树的指针
82 * @Return: 无
83 * @Author: Carlos
84 */
85 void PrintBiT(BiTree elem){
86     printf("%d ",elem->data);
87 }
88 /**
89 * @Description: 拿到栈顶元素
90 * @Param: BiTree*a 指针数组
91 * @Return: 栈顶元素的地址
92 * @Author: Carlos
93 */
94 BiTree GetTop(BiTree*a){
95     return a[top];
96 }
97 /**
98 * @Description: 中序遍历非递归算法，先左，然后回退，然后右。从根结点开始，遍历左孩子同时压栈，当遍历结束，说明当前遍历的结点没有左孩子，  

99 * 从栈中取出来调用操作函数，然后访问该结点的右孩子，继续以上重复性的操作
100 * @Return: 栈顶元素的地址
101 * @Author: Carlos
102 */
103 void InOrderTraverse1(BiTree Tree){
104     //定义一个顺序栈
105     BiTree a[20];
106     //临时指针
107     BiTree p;
108     //根结点进栈
109     Push(a, Tree);
110     //top!= -1说明栈内不为空，程序继续运行
111     while (top!= -1) {
112         //一直取栈顶元素，且不能为NULL
113         while ((p=GetTop(a)) &&p){
114             //将该结点的左孩子进栈，如果没有左孩子，NULL进栈
115             Push(a, p->lchild);
116         }
117         //跳出循环，栈顶元素肯定为NULL，将NULL弹栈。 打印的第一个元素没有右孩子，所以也会Pop掉，再取栈顶元素就是第一个元素的父节点
118         Pop();
119         if (top!= -1) {
120             //取栈顶元素
121             p=GetTop(a);
122             //栈顶元素弹栈
123             Pop();
124             //遍历完所有左孩子之后，打印栈顶的元素。
125             PrintBiT(p);
126             //将p指向的结点的右孩子进栈

```

```

127         Push(a, p->rchild);
128     }
129 }
130 */
131 /**
132 * @Description: 中序遍历非递归算法。中序遍历过程中，只需将每个结点的左子树压栈即可，右
133 * 当结点的左子树遍历完成后，只需要以栈顶结点的右孩子为根结点，继续循环遍历即可
134 * @Param: 无
135 * @Return: 栈顶元素的地址
136 * @Author: Carlos
137 */
138 void InorderTraverse2(BiTTree Tree){
139     //定义一个顺序栈
140     BiTTree a[20];
141     //临时指针
142     BiTTree p;
143     p=Tree;
144     //当p为NULL或者栈为空时，表明树遍历完成
145     while (p || top!= -1) {
146         //如果p不为NULL，将其压栈并遍历其左子树
147         if (p) {
148             Push(a, p);
149             p=p->lchild;
150         }
151         //如果p==NULL，表明左子树遍历完成，需要遍历上一层结点的右子树 弹出时顺便访问右
152         //子树
153         else{
154             p=GetTop(a);
155             Pop();
156             PrintBiT(p);
157             p=p->rchild;
158         }
159     }
160 int main(){
161     BiTTree Tree;
162     CreateBiTree(&Tree);
163     printf("中序遍历: \r\n");
164     InorderTraverse2(Tree);
165     DBG_PRINTF("123456\r\n");
166     return 0;
167 }

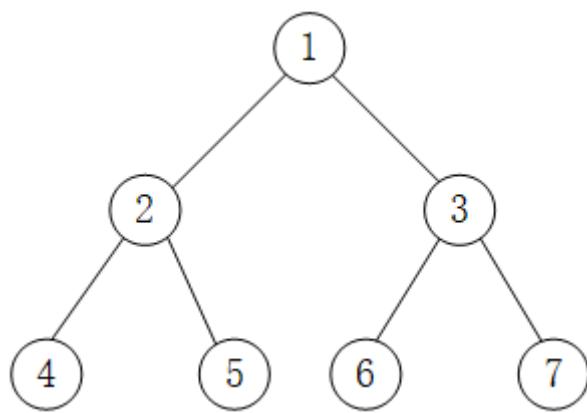
```

## 二叉树后序遍历（递归和非递归）

### 后序遍历规则

二叉树后序遍历的实现思想是：1.访问左子树；2.访问右子树；3.完成该节点的左右子树的访问后，再访问该节点。即考察到一个节点后，将其暂存，遍历完左右子树后，再输出该节点的值。（左右根）

### 后序遍历举例



如上图中，对此二叉树进行后序遍历的操作过程为：

从根节点 1 开始，遍历该节点的左子树（以节点 2 为根节点）；

1. 遍历节点 2 的左子树（以节点 4 为根节点）；

2. 由于节点 4 既没有左子树，也没有右子树，此时访问该节点中的元素 4，并回退到节点 2，遍历节点 2 的右子树（以 5 为根节点）；

3. 由于节点 5 无左右子树，因此可以访问节点 5，并且此时节点 2 的左右子树也遍历完成，因此也可以访问节点 2；

4. 此时回退到节点 1，开始遍历节点 1 的右子树（以节点 3 为根节点）；

5. 遍历节点 3 的左子树（以节点 6 为根节点）；

6. 由于节点 6 无左右子树，因此访问节点 6，并回退到节点 3，开始遍历节点 3 的右子树（以节点 7 为根节点）；

7. 由于节点 7 无左右子树，因此访问节点 7，并且节点 3 的左右子树也遍历完成，可以访问节点 3；节点 1 的左右子树也遍历完成，可以访问节点 1；

由此，对上图 中二叉树进行后序遍历的结果为：4 5 2 6 7 3 1

### 后序遍历代码（递归）

```

1  /*
2   * @Description: 二叉树的后序遍历（递归）
3   * @Version: v1.0
4   * @Autor: Carlos
5   * @Date: 2020-05-18 16:23:57
6   * @LastEditors: Carlos
7   * @LastEditTime: 2020-05-30 17:29:38
8   */
9  #include <stdio.h>
10 #include <string.h>
11 #include <stdlib.h>
12 #define TElmType int
13 //构造结点的结构体
14 typedef struct BiTNode{
15     //数据域
16     TElmType data;
17     //左右孩子指针
18     struct BiTNode *lchild,*rchild;
19 }BiTNode,*BiTree;
20 /**
21 * @Description: 初始化树
  
```

```

22 * @Param: BiTree *T 结构体指针
23 * @Return: 无
24 * @Author: Carlos
25 */
26 void CreateBiTree(BiTree *T){
27     *T=(BiTree)malloc(sizeof(BiTNode));
28     (*T)->data=1;
29     (*T)->lchild=(BiTree)malloc(sizeof(BiTNode));
30     (*T)->rchild=(BiTree)malloc(sizeof(BiTNode));
31
32     (*T)->lchild->data=2;
33     (*T)->lchild->lchild=(BiTree)malloc(sizeof(BiTNode));
34     (*T)->lchild->rchild=(BiTree)malloc(sizeof(BiTNode));
35
36     (*T)->lchild->rchild->data=5;
37     (*T)->lchild->rchild->lchild=NULL;
38     (*T)->lchild->rchild->rchild=NULL;
39
40     (*T)->rchild->data=3;
41     (*T)->rchild->lchild=(BiTree)malloc(sizeof(BiTNode));
42     (*T)->rchild->lchild->data=6;
43     (*T)->rchild->lchild->lchild=NULL;
44     (*T)->rchild->lchild->rchild=NULL;
45
46     (*T)->rchild->rchild=(BiTree)malloc(sizeof(BiTNode));
47     (*T)->rchild->rchild->data=7;
48     (*T)->rchild->rchild->lchild=NULL;
49     (*T)->rchild->rchild->rchild=NULL;
50
51     (*T)->lchild->lchild->data=4;
52     (*T)->lchild->lchild->lchild=NULL;
53     (*T)->lchild->lchild->rchild=NULL;
54 }
55
56 /**
57 * @Description: 显示函数
58 * @Param: BiTree elem 指向树的结构体指针
59 * @Return: 无
60 * @Author: Carlos
61 */
62 void PrintBiT(BiTree elem){
63     printf("%d ", elem->data);
64 }
65 /**
66 * @Description: 先序遍历
67 * @Param: BiTree T 指针数组, 存放各个节点的指针
68 * @Return: 无
69 * @Author: Carlos
70 */
71 void PreOrderTraverse(BiTree T){
72     if (T) {
73         PreOrderTraverse(T->lchild); //访问该结点的左孩子
74         PreOrderTraverse(T->rchild); //访问该结点的右孩子
75         PrintBiT(T); //调用操作结点数据的函数方法
76     }
77     //如果结点为空, 返回上一层
78     return;
79 }

```

```

80 int main() {
81     BiTree Tree;
82     CreateBiTree(&Tree);
83     printf("后序遍历: \n");
84     PreOrderTraverse(Tree);
85 }

```

## 后序遍历代码（非递归）

```

1 /*
2 * @Description: 二叉树的后序遍历（非递归）
3 * @Version: v1.0
4 * @Author: Carlos
5 * @Date: 2020-05-18 16:23:57
6 * @LastEditors: Carlos
7 * @LastEditTime: 2020-05-18 16:24:29
8 */
9 #include <stdio.h>
10 #include <string.h>
11 #include <stdlib.h>
12 #define TElemType int
13 //top变量时刻表示栈顶元素所在位置
14 int top=-1;
15 //构造结点的结构体
16 typedef struct BiTNode{
17     //数据域
18     TElemType data;
19     //左右孩子指针
20     struct BiTNode *lchild,*rchild;
21 }BiTNode,*BiTree;
22 /**
23 * @Description: 初始化树
24 * @Param: BiTree *T 结构体指针数组
25 * @Return: 无
26 * @Author: Carlos
27 */
28 void CreateBiTree(BiTree *T){
29     *T=(BiTNode*)malloc(sizeof(BiTNode));
30     (*T)->data=1;
31     (*T)->lchild=(BiTNode*)malloc(sizeof(BiTNode));
32     (*T)->rchild=(BiTNode*)malloc(sizeof(BiTNode));
33     (*T)->lchild->data=2;
34     (*T)->lchild->lchild=(BiTNode*)malloc(sizeof(BiTNode));
35     (*T)->lchild->rchild=(BiTNode*)malloc(sizeof(BiTNode));
36     (*T)->lchild->rchild->data=5;
37     (*T)->lchild->rchild->lchild=NULL;
38     (*T)->lchild->rchild->rchild=NULL;
39     (*T)->rchild->data=3;
40     (*T)->rchild->lchild=(BiTNode*)malloc(sizeof(BiTNode));
41     (*T)->rchild->lchild->data=6;
42     (*T)->rchild->lchild->lchild=NULL;
43     (*T)->rchild->lchild->rchild=NULL;
44     (*T)->rchild->rchild=(BiTNode*)malloc(sizeof(BiTNode));
45     (*T)->rchild->rchild->data=7;
46     (*T)->rchild->rchild->lchild=NULL;
47     (*T)->rchild->rchild->rchild=NULL;
48     (*T)->lchild->lchild->data=4;

```

```

49     (*T)->lchild->lchild->lchild=NULL;
50     (*T)->lchild->lchild->rchild=NULL;
51 }
52 /**
53 * @Description: 后序遍历使用的弹栈函数
54 * @Param: 无
55 * @Return: 无
56 * @Author: Carlos
57 */
58 void Pop( ){
59     if (top== -1) {
60         return ;
61     }
62     top--;
63 }
64 /**
65 * @Description: 显示函数
66 * @Param: 无
67 * @Return: 无
68 * @Author: Carlos
69 */
70 void PrintBiT(BiTTree elem){
71     printf("%d ",elem->data);
72 }
73
74 //增加左右子树的访问标志
75 typedef struct SNode{
76     BiTTree p;
77     int tag;
78 }SNode;
79 /**
80 * @Description: 后序遍历使用的进栈函数
81 * @Param: SNode *a 指向树和标志位的结构体的指针 BiTTree sdata 进栈的元素
82 * @Return: 无
83 * @Author: Carlos
84 */
85 void Push(SNode *a,SNode sdata){
86     a[++top]=sdata;
87 }
88 /**
89 * @Description: 后序遍历非递归算法。后序遍历是在遍历完当前结点的左右孩子之后，才调用操作函数，所以需要在操作结点进栈时，为每个结点配备一个标志位。
90 * 当遍历该结点的左孩子时，设置当前结点的标志位为 0，进栈；当要遍历该结点的右孩子时，设置当前结点的标志位为 1，进栈。这样，当遍历完成，该结点弹栈时，
91 * 查看该结点的标志位的值：如果是 0，表示该结点的右孩子还没有遍历；反之如果是 1，说明该结点的左右孩子都遍历完成，可以调用操作函数。
92 * @Param: 结构体指针数组
93 * @Return: 无
94 * @Author: Carlos
95 */
96 void PostOrderTraverse(BiTTree Tree){
97     //定义一个顺序栈
98     SNode a[20];
99     //临时指针
100    BiTTree p;
101    int tag;
102    SNode sdata;
103    p=Tree;

```

```

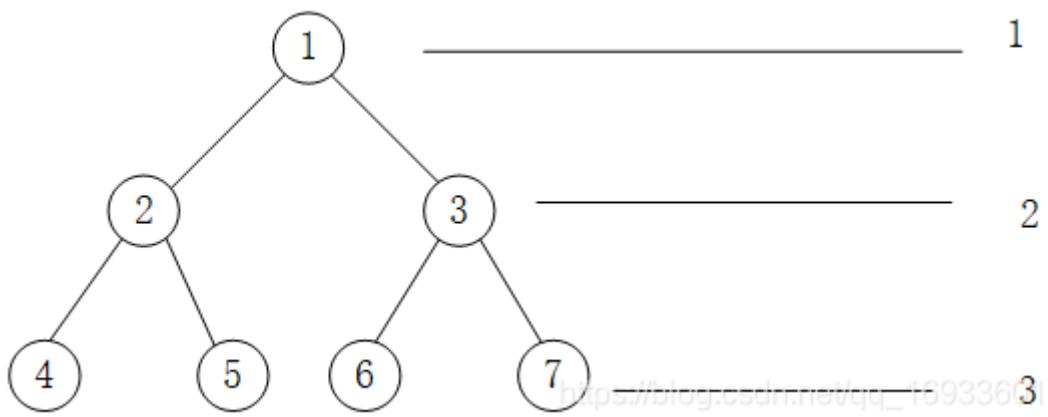
104     while (p||top!=-1) {
105         //左孩子进栈
106         while (p) {
107             //为该结点入栈做准备
108             sdata.p=p;
109             //由于遍历是左孩子，设置标志位为0
110             sdata.tag=0;
111             //压栈
112             Push(a, sdata);
113             //以该结点为根结点，遍历左孩子
114             p=p->lchild;
115         }
116         //取栈顶元素 取左孩子的父节点
117         sdata=a[top];
118         //栈顶元素弹栈
119         Pop();
120         p=sdata.p;
121         tag=sdata.tag;
122         //右孩子进栈
123         //如果tag==0，说明该结点还没有遍历它的右孩子
124         if (tag==0) {
125             sdata.p=p;
126             sdata.tag=1;
127             //更改该结点的标志位，重新压栈
128             Push(a, sdata);
129             //以该结点的右孩子为根结点，重复循环
130             p=p->rchild;
131         }
132         //如果取出来的栈顶元素的tag==1，说明此结点左右子树都遍历完了，可以调用操作函数
133     }  
    else{
134         PrintBiT(p);
135         p=NULL;
136     }
137 }
138 }
139 int main(){
140     BiTree Tree;
141     CreateBiTree(&Tree);
142     printf("后序遍历: \n");
143     PostOrderTraverse(Tree);
144 }
```

## 层次遍历

### 层次遍历规则

按照二叉树中的层次从左到右依次遍历每层中的结点。通过使用队列的数据结构，从树的根结点开始，依次将其左孩子和右孩子入队。而后每次队列中一个结点出队，都将其左孩子和右孩子入队，直到树中所有结点都出队，出队结点的先后顺序就是层次遍历的最终结果。

### 层次遍历举例



例如，层次遍历如上图中的二叉树：

- 1.根结点 1 入队；
- 2.根结点 1 出队，出队的同时，将左孩子 2 和右孩子 3 分别入队；
- 3.队头结点 2 出队，出队的同时，将结点 2 的左孩子 4 和右孩子 5 依次入队；
- 4.队头结点 3 出队，出队的同时，将结点 3 的左孩子 6 和右孩子 7 依次入队；
- 5.不断地循环，直至队列内为空。

### 层次遍历代码

```

1  /*
2   * @Description: 二叉树的层次遍历
3   * @Version: v1.0
4   * @Author: Carlos
5   * @Date: 2020-05-20 14:52:38
6   * @LastEditors: Carlos
7   * @LastEditTime: 2020-05-30 17:41:48
8   */
9  #include <stdio.h>
10 #include <stdlib.h>
11 #define TElmType int
12 //初始化队头和队尾指针开始时都为0
13 int front=0,rear=0;
14 typedef struct BiTNode{
15     //数据域
16     TElmType data;
17     //左右孩子指针
18     struct BiTNode *lchild,*rchild;
19 }BiTNode,*BiTree;
20 //采用顺序队列，初始化创建队列数组
21 BiTree a[20];
22 /**
23  * @Description: 初始化二叉树
24  * @Param: BiTree *T 二叉树的结构体指针数组
25  * @Return: 无
26  * @Author: Carlos
27  */
28 void CreateBiTree(BiTree *T){
29     *T=(BiTNode*)malloc(sizeof(BiTNode));
30     (*T)->data=1;
31     (*T)->lchild=(BiTNode*)malloc(sizeof(BiTNode));
```

```

32     (*T)->rchild=(BiTNode*)malloc(sizeof(BiTNode));
33
34     (*T)->lchild->data=2;
35     (*T)->lchild->lchild=(BiTNode*)malloc(sizeof(BiTNode));
36     (*T)->lchild->rchild=(BiTNode*)malloc(sizeof(BiTNode));
37     (*T)->lchild->rchild->data=5;
38     (*T)->lchild->rchild->lchild=NULL;
39     (*T)->lchild->rchild->rchild=NULL;
40
41     (*T)->rchild->data=3;
42     (*T)->rchild->lchild=(BiTNode*)malloc(sizeof(BiTNode));
43     (*T)->rchild->lchild->data=6;
44     (*T)->rchild->lchild->lchild=NULL;
45     (*T)->rchild->lchild->rchild=NULL;
46
47     (*T)->rchild->rchild=(BiTNode*)malloc(sizeof(BiTNode));
48     (*T)->rchild->rchild->data=7;
49     (*T)->rchild->rchild->lchild=NULL;
50     (*T)->rchild->rchild->rchild=NULL;
51
52     (*T)->lchild->lchild->data=4;
53     (*T)->lchild->lchild->lchild=NULL;
54     (*T)->lchild->lchild->rchild=NULL;
55 }
56 /**
57 * @Description: 入队
58 * @Param: BiTree *a 二叉树结构体指针 BiTree node 入队的节点
59 * @Return: 无
60 * @Author: Carlos
61 */
62 void EnQueue(BiTree *a,BiTree node){
63     a[rear++]=node;
64 }
65 /**
66 * @Description: 出队
67 * @Param: BiTree *node 二叉树结构体指针数组
68 * @Return: 结构体指针
69 * @Author: Carlos
70 */
71 BiTree DeQueue(BiTree *node){
72     return a[front++];
73 }
74 /**
75 * @Description: 二叉树输出函数
76 * @Param: BiTree node 输出的节点
77 * @Return: 无
78 * @Author: Carlos
79 */
80 void displayNode(BiTree node){
81     printf("%d ",node->data);
82 }
83 int main(){
84     BiTree tree;
85     //初始化二叉树
86     CreateBiTree(&tree);
87     BiTree p;
88
89     //根结点入队

```

```

90     EnQueue(a, tree);
91     //当队头和队尾相等时，表示队列为空
92     while(front<rear) {
93         //队头结点出队
94         p=DeQueue(a);
95         displayNode(p);
96         //将队头结点的左右孩子依次入队
97         if (p->lchild!=NULL) {
98             EnQueue(a, p->lchild);
99         }
100        if (p->rchild!=NULL) {
101            EnQueue(a, p->rchild);
102        }
103    }
104    return 0;
105 }
```

## 求二叉树的深度

### 题目描述

[力扣链接](#)

输入一棵二叉树的根节点，求该树的深度。从根节点到叶节点依次经过的节点（含根、叶节点）形成树的一条路径，最长路径的长度为树的深度。

例如：

给定二叉树 [3,9,20,null,null,15,7]，

```

1      3
2      / \
3      9   20
4      /   \
5      15   7
```

返回它的最大深度 3。

提示：节点总数  $\leq 10000$

### 解题思路

考虑以下几种情况：

如果二叉树为空，深度为 0；

如果二叉树只有根节点，深度为 1；

如果二叉树的根节点只有左子树，深度为左子树的深度加 1；

如果二叉树的根节点只有右子树，深度为右子树的深度加 1；

如果二叉树的根节点既有左子树又有右子树，深度为左右子树深度的最大者再加 1。

### 代码

```

1 int maxDepth(struct TreeNode* root){
2     if (root == NULL) {
3         return 0;
4     }
5
6     /* 左子树的深度 */
7     int lenLeft = maxDepth(root->left);
8     /* 右子树的深度 */
9     int lenRight = maxDepth(root->right);
10
11    /* 二叉树的深度等于左右子树的较大者加 1 (根节点深度) */
12    return lenLeft > lenRight ? lenLeft + 1 : lenRight + 1;
13 }
```

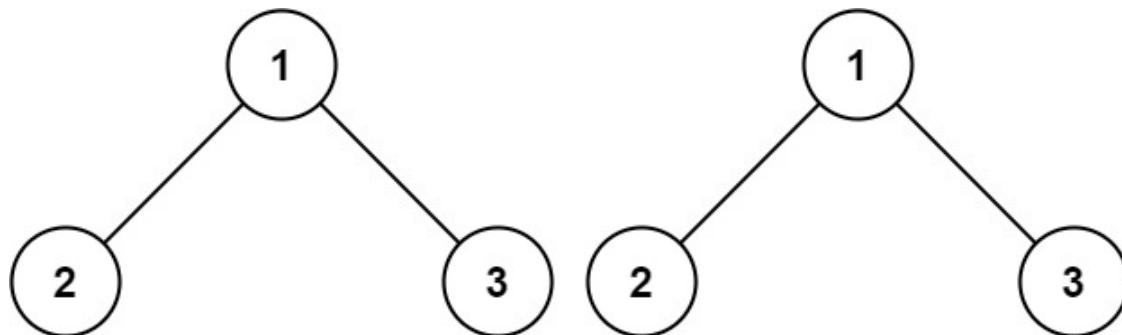
## 如何判断二叉树是否相等

### 题目描述

[力扣链接](#)

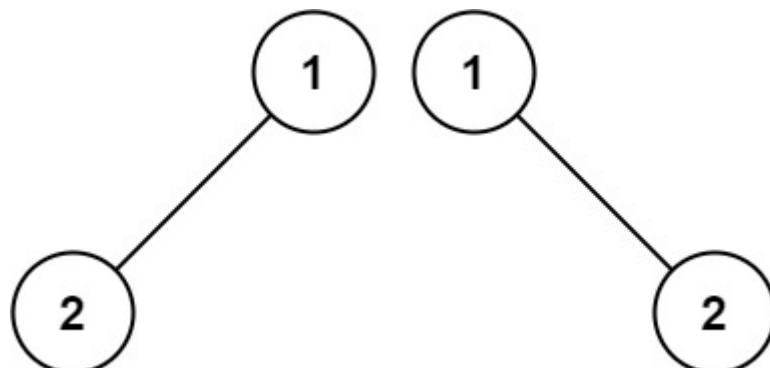
给你两棵二叉树的根节点  $p$  和  $q$ ，编写一个函数来检验这两棵树是否相同。如果两个树在结构上相同，并且节点具有相同的值，则认为它们是相同的。

示例 1

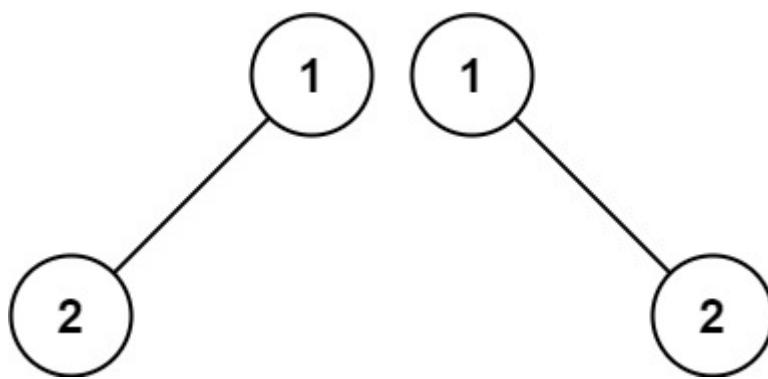


1 | 输入:  $p = [1, 2, 3]$ ,  $q = [1, 2, 3]$   
2 | 输出: true

示例 2



1 | 输入:  $p = [1, 2]$ ,  $q = [1, null, 2]$   
2 | 输出: false



1 | 输入: p = [1,2,1], q = [1,1,2]  
 2 | 输出: false

提示:

两棵树上的节点数目都在范围 [0, 100] 内

-104 <= Node.val <= 104

### 解题思路

采用前序遍历方法遍历二叉树，并把遍历的节点值存储在数组中，并且也把每个节点的空子树也储存下来（用0表示某个节点的空子树）。

### 代码

```

1  /*
2   * Definition for binary tree
3   * struct TreeNode {
4   *     int val;
5   *     struct TreeNode *left;
6   *     struct TreeNode *right;
7   * };
8   */
9 void PreOrder(struct TreeNode *root, int *arr, int *i){
10    (*i)++;
11    if(root == NULL){
12        *(arr + *i) = 0;//用0存储每个节点的空子树值
13        return;
14    }
15    *(arr + *i) = root->val;
16    PreOrder(root->left, arr, i);
17    PreOrder(root->right, arr, i);
18}
19 bool isSameTree(struct TreeNode *p, struct TreeNode *q) {
20    int i,j;
21    int arr1[1000],arr2[1000];//初始化一个数组用于存储节点值
22    memset(arr1, 0, 1000*sizeof(int));
23    memset(arr2, 0, 1000*sizeof(int));
24    if(p == NULL && q == NULL){
25        return 1;
26    }
27    if(p == NULL || q == NULL){
28        return 0;
29    }
  
```

```

30     j = i = 0;
31     PreOrder(p, arr1, &i);
32     PreOrder(q, arr2, &j);
33     for(i = 0; i < 1000; i++){
34         if(arr1[i] != arr2[i]){
35             return 0;
36         }
37     }
38     return 1;
39 }
```

## 如何判断二叉树是否是平衡二叉树

### 题目描述

#### [力扣链接](#)

输入一棵二叉树的根节点，判断该树是不是平衡二叉树。如果某二叉树中任意节点的左右子树的深度相差不超过1，那么它就是一棵平衡二叉树。

示例 1:

给定二叉树 [3,9,20,null,null,15,7]

1	3
2	/ \
3	9 20
4	/ \
5	15 7

返回 true。

示例 2:

给定二叉树 [1,2,2,3,3,null,null,4,4]

1	1
2	/ \
3	2 2
4	/ \
5	3 3
6	/ \
7	4 4

返回 false。

限制：

0 <= 树的结点个数 <= 10000

### 解题思路

- 1.理解平衡树是什么？平衡树是一棵树的左右子树的高度差距小于等于1
- 2.使用递归的算法可以求出二叉树的高度
- 3.先求根的左右的高度差距是不是满足，然后递归查询，用与的（&&）的方法，如果其中有一个false，那么整体就是false；

## 代码

```
1  /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     struct TreeNode *left;
6  *     struct TreeNode *right;
7  * };
8  */
9
10 int deptMax(struct TreeNode* root)
11 {
12     if (root == NULL)
13     {
14         return 0;
15     }
16     int leftDept = deptMax(root->left);
17     int rightDept = deptMax(root->right);
18     return leftDept>rightDept?leftDept+1:rightDept+1;
19 }
20
21 bool isBalanced(struct TreeNode* root)
22 {
23     if (root== NULL)
24     {
25         return true;
26     }
27     int leftDept = deptMax(root->left);
28     int rightDept = deptMax(root->right);
29
30     if (abs(leftDept-rightDept)<2 == false)
31     {
32         return false;
33     }
34
35     return abs(leftDept-rightDept)<2&&isBalanced(root-
36 >left)&&isBalanced(root->right);
```



微信搜一搜

嵌入式与Linux那些事

# 数组

## 最大子序和

### 题目描述

[力扣链接](#)

给定一个整数数组 `nums`，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

示例:

输入: [-2,1,-3,4,-1,2,1,-5,4]

输出: 6

解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。

进阶:

如果你已经实现复杂度为  $O(n)$  的解法，尝试使用更为精妙的分治法求解。

### 解题思路

#### 方法一：暴力

遍历两次，记下max的值，在内层循环和累加值比较。

#### 方法二：分治

1. 结束条件，当`numsSize==1`时直接返回当前值
2. 处理左支 求 `Max_Left`
3. 处理右支 求 `Max_Right`
4. 处理本身，向左求最大和`Max_L` 然后 向右求最大和`Max_R`
5. 最终结果为 `MAX(Max_Left, Max_Right, Max_R)`

### 代码

```

1 int maxSubArray(int* nums, int numssize){
2
3     int i = 0;
4     int j = 0;
5     int Max = nums[0];
6     int Tmp = 0;
7
8     for (i = 0; i < numssize; i++)
9     {
10         Tmp = 0;
11         for (j = i; j < numssize; j++)
12         {
13             Tmp += nums[j];
14             if (Tmp > Max)
15             {
16                 Max = Tmp;
17             }
18         }
19     }
20     return Max;
21 }
```

方法二：

```

1 #define MAX(a, b, c) ((a) > ((b) > (c) ? (b) : (c)) ? (a) : ((b) >
2 (c) ? (b) : (c)))
3 int maxSubArray(int* nums, int numssize){
4     int i = 0;
5     int iTmp = 0;
6     int Max_Left = 0;
7     int Max_Right = 0;
8     int Max_l = 0;
9     int Max_r = 0;
10
11    //1,结束条件
12    if ((0 == numssize) || (1 == numssize))
13    {
14        return nums[0];
15    }
16    else
17    {
18        //2,求左支
19        Max_Left = maxSubArray(&nums[0], (numssize - 1) / 2);
20        //3,求右支
21        Max_Right = maxSubArray(&nums[(numssize + 1) / 2], numssize / 2);
22    }
23
24    //4,求中间
25    iTmp = 0;
26    Max_l = nums[(numssize - 1) / 2];
27    for (i = (numssize - 1) / 2; i >= 0; i--)
28    {
29        iTmp += nums[i];
30        if (iTmp > Max_l)
31        {
32            Max_l = iTmp;
33        }
34    }
35    iTmp = Max_l;
36    Max_r = Max_l;
37    for (i = (numssize + 1) / 2; i < numssize; i++)
38    {
39        iTmp += nums[i];
40        if (iTmp > Max_r)
41        {
42            Max_r = iTmp;
43        }
44    }
45    //5,返回最大值
46    return MAX(Max_Left, Max_Right, Max_r);
47 }

```

## 原地移除元素

### 题目描述

[力扣链接](#)

给你一个数组 `nums` 和一个值 `val`，你需要原地移除所有数值等于 `val` 的元素，并返回移除后数组的新长度。

不要使用额外的数组空间，你必须仅使用 O(1) 额外空间并原地修改输入数组。

元素的顺序可以改变。你不需要考虑数组中超出新长度后面的元素。

示例 1：

给定 nums = [3,2,2,3], val = 3,

函数应该返回新的长度 2，并且 nums 中的前两个元素均为 2。

你不需要考虑数组中超出新长度后面的元素。示例 2：

给定 nums = [0,1,2,2,3,0,4,2], val = 2,

函数应该返回新的长度 5，并且 nums 中的前五个元素为 0, 1, 3, 0, 4。

注意这五个元素可为任意顺序。

你不需要考虑数组中超出新长度后面的元素。

## 解题思路

一个（慢）num指针为数组本身进行复写，一个（快）p指针不断移动并判断是否等于val的值。一旦快指针所指向的值不等于val，则进行复写，将快指针所指向的值复写慢指针所指向的值，且计数器加一。

（注意顺序：先`*nums=p[i]`，后`nums++`；因为慢指针的最后一项总是待复写的位置，所以若快指针所指向的值不等于val时，先将快指针的值复写慢指针的值，再将慢指针往后挪一位）

## 代码

```

1 int removeElement(int* nums, int numssize, int val){
2
3     int count=0;
4     int *p=nums;
5     for(int i=0;i<numssize;i++){
6         if(p[i]!=val){
7             *nums=p[i];
8             nums++;
9             count++;
10        }
11    }
12    return count;
13 }
```

## 合并两个有序数组

### 题目描述

#### [力扣链接](#)

给你两个有序整数数组 nums1 和 nums2，请你将 nums2 合并到 nums1 中，使 nums1 成为一个有序数组。

说明：

初始化 nums1 和 nums2 的元素数量分别为 m 和 n。你可以假设 nums1 有足够的空间（空间大小大于或等于 m +n）来保存 nums2 中的元素。

示例：

输入: nums1 = [1,2,3,0,0,0], m = 3 nums2 = [2,5,6], n = 3

输出: [1,2,2,3,5,6]

## 解题思路

方法一：分别遍历两个数组nums1和nums2，申请nums3的空间存放二者排序的结果。

方法二：其实也可以不用申请额外空间，从后往前遍历两个数组，统一合并到nums1中。

## 代码

方法一：

```

1 void merge(int* nums1, int nums1Size, int m, int* nums2, int nums2Size, int n){
2     int *nums3 = (int *)malloc(sizeof(int) * (m + n));
3     int i = 0, j = 0, k = 0;
4     //遍历两个数组，只要有一个遍历完成时就退出
5     while(i < m && j < n)
6     {
7         if(nums1[i] <= nums2[j])
8         {
9             nums3[k] = nums1[i];
10            i++;
11        }
12        else
13        {
14            nums3[k] = nums2[j];
15            j++;
16        }
17        k++;
18    }
19    //当nums1遍历完成，nums2还有元素时，直接把nums2的放在nums3后面
20    if(i == m && j < n)
21    {
22        while(j < n)
23        {
24            nums3[k] = nums2[j];
25            j++;
26            k++;
27        }
28    }
29    //当nums2遍历完成，nums1还有元素时，直接把nums1的放在nums3后面
30    if(j == n && i < m)
31    {
32        while(i < m)
33        {
34            nums3[k] = nums1[i];
35            i++;
36            k++;
37        }
38    }
39    //把nums3的元素赋值给nums1
40    for(i = 0; i < nums1Size; i++)
41    {
42        nums1[i] = nums3[i];
43    }
44 }
```

方法二：

```

1 void merge(int* nums1, int nums1Size, int m, int* nums2, int nums2Size, int
n){
2     int M = m - 1;
3     int N = n - 1;
4     int cur = m + n - 1;
5     // 从后往前处理，nums2都加到num1上去
6     while (M >= 0 && N >= 0)
7     {
8         if (nums1[M] >= nums2[N])
9         {
10             nums1[cur--] = nums1[M--];
11         }
12         else if (nums1[M] < nums2[N])
13         {
14             nums1[cur--] = nums2[N--];
15         }
16     }
17     // 要是nums2有剩余的，处理一下
18     while (N >= 0)
19     {
20         nums1[cur--] = nums2[N--];
21     }
22 }
```

## 查找常用字符

### 题目描述

#### [力扣链接](#)

给定仅有小写字母组成的字符串数组 A，返回列表中的每个字符串中都显示的全部字符（包括重复字符）组成的列表。例如，如果一个字符在每个字符串中出现 3 次，但不是 4 次，则需要在最终答案中包含该字符 3 次。

你可以按任意顺序返回答案。

示例 1：

输入：["bella", "label", "roller"]

输出：["e", "l", "l"]

示例 2：

输入：["cool", "lock", "cook"]

输出：["c", "o"]

提示：

1 <= A.length <= 100 1 <= A[i].length <= 100 A[i][j] 是小写字母

### 解题思路

用一个二维数组存储每个单词出现的a-z的个数，然后再按照列遍历，找到每个字符在所有单词中出现的最小个数，该个数就是结果中需要添加的个数；

## 代码

```

1  /**
2   * Note: The returned array must be malloced, assume caller calls free().
3   */
4  char ** commonChars(char ** A, int ASize, int* returnSize){
5      if (A == NULL || ASize == 0) {
6          (*returnSize) = 0;
7          return NULL;
8      }
9      int res[100][26] = {0}; // 每个单词占一行
10     char **rslt = malloc(sizeof(char*) * 100);
11
12     int i, j;
13     for (i = 0; i < 100; i++) {
14         rslt[i] = malloc(sizeof(char) * 2);
15     }
16
17     for (i = 0; i < ASize; i++) {
18         for (j = 0; j < strlen(A[i]); j++) {
19             res[i][A[i][j] - 'a']++;
20             // printf("i = %d, %d\n", i, res[i][A[i][j] - 'a']);
21         }
22     }
23
24     int idx = 0;
25     //依次比较每个单词中j字符出现的次数
26     for (i = 0; i < 26; i++) {
27         int min = res[0][i];
28         for (j = 1; j < ASize; j++) {
29             if (res[j][i] < min) {
30                 min = res[j][i];
31             }
32         }
33         // 找到最小的公共的
34         while (min > 0) {
35             rslt[idx][0] = i + 'a'; // 存储重复的字母
36             rslt[idx][1] = '\0';
37             idx++;
38             min--;
39         }
40     }
41     (*returnSize) = idx;
42     return rslt;
43 }
44

```

## 寻找数组的中心索引

### 题目描述

[力扣链接](#)

给定一个整数类型的数组 `nums`, 请编写一个能够返回数组“中心索引”的方法。  
我们是这样定义数组 中心索引 的：数组中心索引的左侧所有元素相加的和等于右侧所有元素相加的和。  
如果数组不存在中心索引，那么我们应该返回 -1。如果数组有多个中心索引，那么我们应该返回最靠近左边的那个。

示例 1：

输入: `nums = [1, 7, 3, 6, 5, 6]`

输出: 3

解释: 索引 3 (`nums[3] = 6`) 的左侧数之和 ( $1 + 7 + 3 = 11$ ), 与右侧数之和 ( $5 + 6 = 11$ ) 相等。同时, 3 也是第一个符合要求的中心索引。

示例 2：

输入: `nums = [1, 2, 3]`

输出: -1

解释: 数组中不存在满足此条件的中心索引。

说明:

`nums` 的长度范围为  $[0, 10000]$ 。任何一个 `nums[i]` 将会是一个范围在  $[-1000, 1000]$  的整数。

## 解题思路

方法一： $S$  是数组的和, 当索引  $i$  是中心索引时, 位于  $i$  左边数组元素的和  $leftsum$  满足  $S - nums[i] - leftsum$ 。我们只需要判断当前索引  $i$  是否满足  $leftsum == S - nums[i] - leftsum$  并动态计算  $leftsum$  的值。

方法二：滑动窗口，先设定一个初始位置0（坑啊，题目出的有问题，应该从1开始才对，没办法，题目认为0左边没元素也符合。。。）已其为中心分别计算左、右的和，然后比较不相等就右移，更新左、右和即可，代码简单易懂。

## 代码

### 方法一

```

1 int pivotIndex(int* nums, int numsSize){
2
3     if (numsSize < 3)
4     {
5         return -1;
6     }
7     int sum = 0, right = 0, left = 0;
8     for(int i = 0;i < numssize;i++)
9     {
10         sum+=nums[i];
11     }
12     for(int i = 0;i < numssize;i++)
13     {
14         //计算右边的和
15         right = sum-nums[i]-left;
16         //左右相等则返回
17         if(left == right)
18             return i;
19         //左边求和
20         left += nums[i];
21     }
22     return -1;
23 }
```

## 方法二

```

1  /*
2   * 1. 从位置0开始右移寻找，先计算left和right，然后每右移一位判断是否符合
3   */
4
5  int pivotIndex(int* nums, int numssize)
6  {
7      if (numssize < 3) {
8          return -1;
9      }
10     int left = 0;
11     int right = 0;
12     for (int i = 1; i < numssize; ++i) {
13         right += nums[i];
14     }
15
16     int pos = 0;
17     while (left != right) {
18         if (pos + 1 == numssize) {
19             return -1;
20         }
21         left += nums[pos];
22         right -= nums[pos + 1];
23         pos++;
24     }
25     return pos;
26 }
```

## 数组中数字出现的次数

### 题目描述

#### [力扣链接](#)

一个整型数组 nums 里除两个数字之外，其他数字都出现了两次。请写程序找出这两个只出现一次的数字。要求时间复杂度是O(n)，空间复杂度是O(1)。

示例 1：

输入: nums = [4,1,4,6]

输出: [1,6] 或 [6,1]

示例 2: 输入: nums = [1,2,10,4,1,4,3,3]

输出: [2,10] 或 [10,2]

限制:

2 <= nums.length <= 10000

### 解题思路

利用异或运算^的特点

0和任何数异或等于该数

任何数和自身异或等于0

异或运算满足交换律

当将一个数组的数全部按位异或后，可以将成对的数放在前面先运算，而成对的数经过上述1,2会得到0，也就达到了消去成对的数的目的。

此时数组中剩下两个不同的数异或，得到的结果mask是这两个数不同的位上值为1，相同的为0。

但这样没办法把两个数分离出来。既然这两个不同的数必然有位是不同的，并且异或的结果也告诉我们哪些位不同。我们不妨以mask的最低的值为1的位（设为a0位）来区分这两个数，其中一个数a0位等于1，另一个等于0。

其中通过位运算 $x \& (-x)$ 得到 $a_0 = 1$ ，其他位为0的结果。

同样我们将数组中其他的数，也分成两组： $a_0 = 0$ 组和 $a_0 = 1$ 组，这样保证了两个不同的数分在两组，也保证了成对的数必在其中一组而不会一对数分在两组。

再分别对两组的数进行异或运算，最后得到的两个数即所求。

## 代码

```

1 /**
2  * Note: The returned array must be malloced, assume caller calls free().
3 */
4 int* singleNumbers(int* nums, int numssize, int* returnSize){
5     int mask = 0;
6     int i;
7     for(i = 0; i < numssize; i++)
8     {
9         mask ^= nums[i];
10    }
11    mask = mask & (-1*mask);
12    int m1 = 0, m2 = 0;
13    for(i = 0; i < numssize; i++)
14    {
15        if((nums[i] & mask) == 0)//位运算优先级低于关系运算
16        {
17            m1 ^= nums[i];
18        }
19        else
20        {
21            m2 ^= nums[i];
22        }
23    }
24    *returnSize = 2;
25    int *a = (int *)malloc(2*sizeof(int));
26    a[0] = m1;
27    a[1] = m2;
28    return a;
29 }
```

## 数组中数字出现的次数 II

### 题目描述

#### [力扣链接](#)

在一个数组`nums`中除一个数字只出现一次之外，其他数字都出现了三次。请找出那个只出现一次的数字。

#### 示例 1

输入：`nums = [3,4,3,3]`

输出：4

### 示例 2

输入: nums = [9,1,7,9,7,9,7]

输出: 1

### 解题思路

理论基础: 某一数字出现3次, 则他们每一位的和分别可以被3整除

如数字7: 0111, 第0位为1, 3个1加起来为11, 可以被三整除。第二第三位同理。

解题思路:

1. 分别给数组中所有数字不同位分别求和, 结果记录为temp, int型要分别求32次。
2. 若第i位的求和结果temp能被3整除, 说明只出现了1次的那个数字该位为0, 否则为1.
3. 根据第二部结果, 给result的第i位置1或0。 (置0可以不做处理)。

### 代码

```

1 int singleNumber(int* nums, int numssize){
2     int result = 0;
3     for(int i = 0; i < 32; i++){//遍历32位
4         int temp = 0;//记录所有数字第i位的和
5         for(int j = 0; j < numssize; j++){
6             temp += (nums[j] >> i) & 1;//求第i位之和
7         }
8         if(temp % 3){
9             result += 1 << i;//置1
10        }
11    }
12    return result;
13 }
```

## 数组中缺失的元素

### 题目描述

#### [力扣链接](#)

给定一个包含 0, 1, 2, ..., n 中 n 个数的序列, 找出 0 ... n 中没有出现在序列中的那个数。

### 示例 1:

输入: [3,0,1]

输出: 2

### 示例 2:

输入: [9,6,4,2,3,5,7,0,1]

输出: 8

说明: 你的算法应具有线性时间复杂度。你能否仅使用额外常数空间来实现?

### 解题思路

#### 方法一: 哈希数组

方法二: 异或。好像和以前的一道题 (只出现一次的数字) 有异曲同工之处。看了大家的题解, 异或操作 (^) 是一种很好的方式, 不用考虑sum越界问题。

举个例子:

```

1 | 0 ^ 4 = 4
2 | 4 ^ 4 = 0
3 | 12

```

那么，就可以不用求和，直接使用异或运算 $\wedge$ 进行抵消，剩下的数字就是缺失的了。  
再举个例子：

```

1 | 1^1^2^2^3 = 3
2 | 1

```

方法三：最简单的方法应该是求和，然后和1到n项的数列和对比，相差的数就是缺的这个数

## 代码

### 方法一

```

1 | int missingNumber(int* nums, int numssize){
2 |
3 |     int hash[50000] = {0};
4 |     for(int i = 0; i < numssize; i++)
5 |     {
6 |         hash[nums[i]]++;
7 |     }
8 |     for(int i = 0; i < 50000; i++)
9 |     {
10 |         if(hash[i]==0)
11 |         {
12 |             return i;
13 |         }
14 |     }
15 |     return -1;
16 |
17 |

```

### 方法二

```

1 | int missingNumber(int* nums, int numssize){
2 |
3 |     int res = numssize;
4 |     for (int i = 0; i < numssize; ++i){
5 |         res ^= nums[i];
6 |         res ^= i;
7 |     }
8 |     return res;
9 |
10 |

```

### 方法三

```

1 int missingNumber(int* nums, int numssize){
2
3     int sum = 0;
4     for(int i = 0; i < numssize; i++){
5         sum += nums[i];
6     }
7     return numssize * (numssize + 1) / 2 - sum;
8
9 }
```

## 按奇偶排序数组

### 题目描述

[力扣链接](#)

给定一个非负整数数组 A，返回一个数组，在该数组中，A 的所有偶数元素之后跟着所有奇数元素。你可以返回满足此条件的任何数组作为答案。

示例：

输入：[3,1,2,4]

输出：[2,4,3,1]

输出 [4,2,3,1], [2,4,1,3] 和 [4,2,1,3] 也会被接受。

提示：

$1 \leq A.length \leq 5000$   $0 \leq A[i] \leq 5000$

### 解题思路

定义额外数组和两个头尾下标，遍历原数组，将偶数从前往后放，奇数从后往前放。

### 代码

```

1 /**
2  * Note: The returned array must be malloced, assume caller calls free().
3 */
4 int* sortArrayByParity(int* A, int ASize, int* returnSize){
5
6     *returnSize = ASize;
7     if (ASize < 2)
8         return A;
9
10    int *arr = (int *)malloc(sizeof(int)*ASize);
11    int head = 0,tail = ASize-1;
12    for(int i = 0;i<ASize;i++)
13    {
14        if(A[i]%2==0)
15            arr[head++] = A[i];
16        else
17            arr[tail--] = A[i];
18    }
19
20    return arr;
21
22 }
```

## 数组是否存在重复元素

### 题目描述

#### 力扣链接

给定一个整数数组和一个整数  $k$ , 判断数组中是否存在两个不同的索引  $i$  和  $j$ , 使得  $\text{nums}[i] = \text{nums}[j]$ , 并且  $i$  和  $j$  的差的 绝对值 至多为  $k$ 。

示例 1:

输入:  $\text{nums} = [1,2,3,1]$ ,  $k = 3$  输出: true

示例 2:

输入:  $\text{nums} = [1,0,1,1]$ ,  $k = 1$  输出: true

示例 3:

输入:  $\text{nums} = [1,2,3,1,2,3]$ ,  $k = 2$  输出: false

### 解题思路

方法一：结构体数组排序再遍历

方法二：C有uthash.h头文件，再也不用担心C被哈希虐了。本题就是一直找，再遇到两数相等且  $|j-i| \leq k$  即可马上返回

### 代码

#### 方法一

```

1  typedef struct node_t{
2      int value;
3      int index;
4  } node;
5
6  int cmp(void *a, void *b) {
7      if (((node*)a)->value < ((node*)b)->value)) return -1;
8      else if (((node*)a)->value == ((node*)b)->value)) return 0;
9      else return 1;
10
11 }
12
13
14 bool containsNearbyDuplicate(int* nums, int numssize, int k){
15     node array[100000];
16     int i,j;
17     for(i=0;i<numssize;i++){
18         array[i].value = nums[i];
19         array[i].index = i;
20     }
21     qsort(array, numssize, sizeof(node), cmp);
22     for(i=0;i<numssize; i++){
23         for(j=i+1;j<numssize;j++){
24             if(array[i].value == array[j].value){
25                 if(array[j].index - array[i].index <= k){
26                     return true;
27                 }
28             }
29         else{
30             break;

```

```

31     }
32 }
33 }
34     return false;
35 }
```

## 方法二

```

1  typedef struct hash{
2      int key; // 键
3      int index; // 索引值
4      UT_hash_handle hh; // 让结构体哈希柄
5  } *hash_ptr;
6
7  bool containsNearbyDuplicate(int* nums, int numsSize, int k){
8      hash_ptr p=NULL, tables=NULL;
9      for(int i=0;i<numsSize;i++){
10         if(tables) HASH_FIND_INT(tables, &(nums[i]), p);
11         //如果哈希表中已经存在这个元素，判断当前正在放入的和已经放入的索引值的差值
12         if(p&&(i-p->index)<=k) return true;
13         p=(hash_ptr)malloc(sizeof(*p));
14         p->key=nums[i];
15         p->index=i;
16         HASH_ADD_INT(tables, key, p);
17     }
18     return false;
19 }
20
```

## 有序数组出现次数超过25%的元素

### 题目描述

#### 力扣链接

给你一个非递减的 有序 整数数组，已知这个数组中恰好有一个整数，它的出现次数超过数组元素总数的 25%。

请你找到并返回这个整数

示例：

输入： arr = [1,2,2,6,6,6,7,10]

输出： 6

提示：

$1 \leq \text{arr.length} \leq 10^4$   $0 \leq \text{arr}[i] \leq 10^5$

### 解题思路

方法一：将每个元素放进哈希表，遍历哈希表求次数相等的元素。

方法二：数组有序，且某元素出现次数超过25%。那么对于此元素第1次出现位置，加25%数组长度，必定仍为它自身

## 代码

### 方法一

```

1 int findspecialInteger(int* arr, int arrSize){
2
3     if (arrSize < 1) return -1;
4     int flag[100000] = {0};
5     for(int i = 0;i < arrSize;i++)
6     {
7         flag[arr[i]]++;
8     }
9     int cnt = arrSize*0.25;
10    for(int i = 0;i < 100000;i++)
11    {
12        if(cnt < flag[i])
13            return i;
14    }
15    return -1;
16
17 }
```

### 方法二

```

1 int findspecialInteger(int* arr, int arrSize){
2
3     for (int *p = arr, *q = arr + arrSize / 4; ; p++, q++)
4         if (*p == *q) return *p;
5     return 0;
6 }
```

## 有效的山脉数组

### 题目描述

[力扣链接](#)

给定一个整数数组 A, 如果它是有效的山脉数组就返回 true, 否则返回 false。

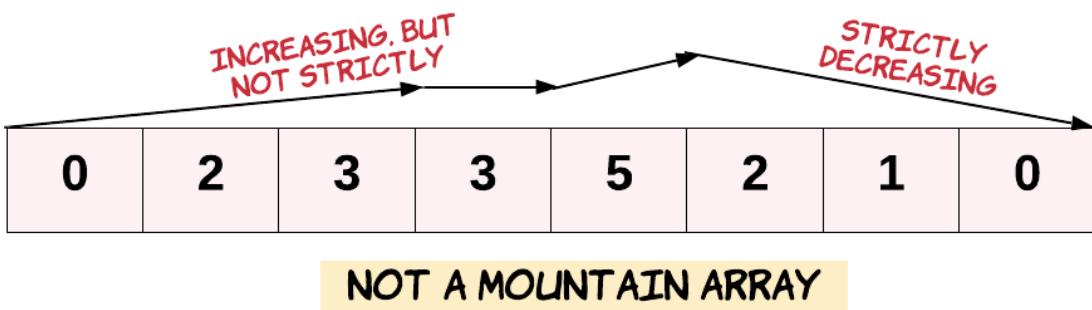
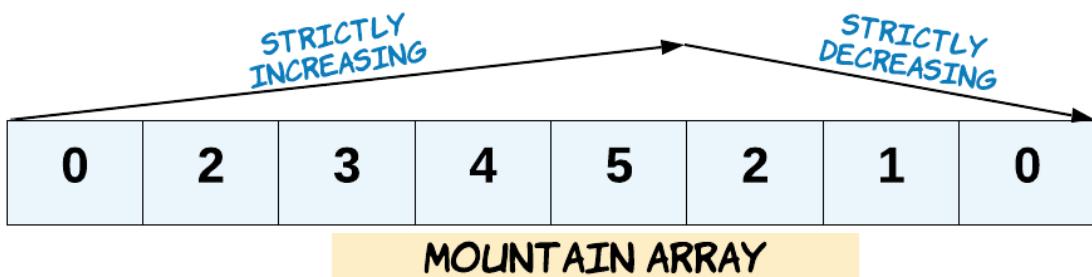
让我们回顾一下, 如果 A 满足下述条件, 那么它是一个山脉数组:

A.length >= 3

在  $0 < i < A.length - 1$  条件下, 存在 i 使得:

$A[0] < A[1] < \dots < A[i-1] < A[i]$

$A[i] > A[i+1] > \dots > A[A.length - 1]$



示例 1：

输入：[2,1] 输出：false

示例 2：

输入：[3,5,5] 输出：false

示例 3：

输入：[0,3,2,1] 输出：true

提示：

$0 \leq A.length \leq 10000$

$0 \leq A[i] \leq 10000$

## 解题思路

方法一：先编写函数找出数组中最大的数的下标，这个下标之前的数应该是严格递增的，这个下标之后的数应该是严格递减的。

方法二：上山步数和下山步数相加等于总长度-1；

## 代码

### 方法一

```

1 int max(int* A, int ASize)//找出数组中最大的数的下标
2 {
3     int i=0, index=0, max=A[0];
4     for(i=0; i<ASize; i++)
5     {
6         if(A[i]>max)
7         {
8             max=A[i];
9             index=i;
10        }
11    }
12    return index;

```

```

13 }
14 bool validMountainArray(int* A, int ASize){
15     if(ASize<3)//数组长度小于3为假
16     {
17         return 0;
18     }
19     int i=0,re=0,ans=1;
20     re=max(A,ASize);
21     if(re==0||re==ASize-1)//这两种情况，整个数组都是递增或递减的，为假
22     {
23         return 0;
24     }
25     for(i=0;i<re;i++)//数组从零直到最大元素都应该是递增的
26     {
27         if(A[i]>=A[i+1])
28         {
29             ans=0;
30             break;
31         }
32     }
33     for(i=re;i<ASize-1;i++)//数组从最大元素直到结尾都应该是递减的
34     {
35         if(A[i]<=A[i+1])
36         {
37             ans=0;
38             break;
39         }
40     }
41     return ans;
42 }
```

## 方法二

```

1 bool validMountainArray(int* A, int ASize){
2
3     if (ASize < 3)
4     {
5         return false;
6     }
7     int inc = 0;
8     int dec = 0;
9     for (int i = 0; i < ASize - 1; i++)
10    {
11        if (A[i] < A[i + 1] && dec == 0)
12        {
13            inc++;
14        } else if (A[i] > A[i + 1] && inc > 0)
15        {
16            dec++;
17        }
18    }
19    if ((inc + dec == ASize - 1) && (inc > 0 && inc < ASize - 1) && (dec > 0
20 && dec < ASize - 1))
21    {
22        return true;
23    }
24    return false;
25 }
```

24 | }

## 最长连续递增序列

### 题目描述

#### [力扣链接](#)

给定一个未经排序的整数数组，找到最长且连续的的递增序列，并返回该序列的长度。

示例 1:

输入: [1,3,5,4,7]

输出: 3

解释: 最长连续递增序列是 [1,3,5]，长度为3。尽管 [1,3,5,7] 也是升序的子序列，但它不是连续的，因为5和7在原数组里被4隔开。

示例 2:

输入: [2,2,2,2,2]

输出: 1

解释: 最长连续递增序列是 [2]，长度为1。

注意: 数组长度不会超过10000。

### 解题思路

首先列出可能出现的数组类型，大致可以分为以下五种：

1. []
2. [1]
3. [1, 1, 1]
4. [1, 2, 3, 4, 5]
5. [7, 8, 9, 1, 2, 3, 4]
6. 12345

对于情况1和2，可以单独列出来做一个判断，并返回对应数组长度作为结果。

对于情况3, 4, 5，进行for循环，并判断后一项是否大于前一项，若是，则计数+1，若不是，计数重置为1。

在for循环中每进行一次if判断都取ans和j中的最大值，可以保证循环结束时ans的值是数组中最长的连续递增序列的长度。

### 方法二：滑动窗口

#### 代码

#### 方法一

```

1 int findLengthofLCIS(int* nums, int numssize){
2     if(numssize == 0)
3         return 0;
4     if(numssize == 1)
5         return 1;
6     int length = 1;
7     int max = 1;
8     for(int i = 0; i < numssize - 1; i++){
9         if(nums[i] < nums[i+1])
10             length++;
11         else

```

```

12     length = 1;
13     if(length > max)
14         max = length;
15     }
16     return max;
17 }
```

## 方法二

```

1 #define MAX(a, b) ((a)>(b)?(a):(b))
2
3 int findLengthofLCIS(int* nums, int numssize)
4 {
5     int start, end;
6     int max_len;
7
8     if (nums == NULL || numssize == 0)
9         return 0;
10
11    start = 0;
12    end = 1;
13    max_len = 1;
14    while (end < numssize) {
15        if (nums[end-1] >= nums[end]) {
16            start = end;
17        }
18        max_len = MAX(max_len, end-start+1);
19        end++;
20    }
21
22    return max_len;
23 }
```

# 字符串

## 有效的括号

### 题目描述

[力扣链接](#)

给定一个只包括 '(', ')', '{', '}', '[', ']' 的字符串，判断字符串是否有效。

有效字符串需满足：

左括号必须用相同类型的右括号闭合。

左括号必须以正确的顺序闭合。

注意空字符串可被认为是有效字符串。

示例 1:

输入: "()" 输出: true

示例 2:

输入: "()[]{}" 输出: true

示例 3:

输入: "[]" 输出: false

示例 4:

输入: "([])" 输出: false

示例 5:

输入: "{}" 输出: true

## 解题思路

这是一道很简单的题目，用栈可以很轻松的实现 当左括号出现的时候入栈，当右括号出现的出栈，如果匹配就继续，不匹配就错误 当字符串遍历完成之后，栈内仍有字符串就错误 用一个数组进行和一个记录栈顶值的int进行了栈的模拟，代码很简单，很好理解（虽说题目也很简单就是了）

## 代码

```

1  /*这是一道很简单的题目，用栈可以很轻松的实现
2  当左括号出现的时候入栈，当右括号出现的出栈，如果匹配就继续，不匹配就错误
3  当字符串遍历完成之后，栈内仍有字符串就错误
4  用一个数组进行和一个记录栈顶值的int进行了栈的模拟，代码很简单，很好理解（虽说题目也很简单就是了）
5 */
6  bool isValid(char * s){
7
8      int len = strlen(s);
9      char stack[3500];
10     int top = -1;
11     int i = 0;
12     for( i=0;i<len;i++)
13     {
14         if((s[i] == '(') || (s[i] == '{') || (s[i] == '['))
15             stack[++top] = s[i];
16         else
17         {
18             if(top<0)//出现了右括号，但数组为空，即没有左括号与之匹配
19                 return false;
20             if((s[i] == ')'))
21             {
22                 if(stack[top]!='(') return false;
23                 else top--;
24             }
25
26
27             if((s[i] == '}'))
28             {
29                 if(stack[top]!='{' return false;
30                 else top--;
31             }
32
33             if((s[i] == ']'))
34             {
35                 if(stack[top]!='[') return false;
36                 else top--;
37             }
38
39         }
40     }
41     if(top>=0) //数组内仍有左括号，没有右括号与之匹配
42         return false;\n        //这里一定要有返回值
43         return true;
44
45
46 }
```

## 字符串数字相加

### 题目描述

[力扣链接](#)

给定两个字符串形式的非负整数 num1 和num2，计算它们的和。

注意：

num1 和num2 的长度都小于 5100.

num1 和num2 都只包含数字 0-9.

num1 和num2 都不包含任何前导零。

你不能使用任何内建 BigInteger 库，也不能直接将输入的字符串转换为整数形式。

### 解题思路

两数之和链表相加一样，其实可以不用移位，非要让数组第一个开头，可以直接返回指针，只要保证字符串结尾有 '\0' 就不会报错。

全局开res[5103]。因为最大可能就是两个都是5100个9，进位最多产生2个位，因此加上末尾的'\0'那么够多了。

注意：不能开函数体内，因为里面res[5103]栈内存，虽然这个够小完全可以开，但是返回返回结束就小时，再次访问可能是NULL或者报错。解决办法在函数体内用关键字static char res[5103]，这跟下面代码放在全局是一样的。

### 代码

```

1  char res[5103] = {'\0'};
2  char * addStrings(char * num1, char * num2){
3
4      short s1 = strlen(num1), s2 = strlen(num2), len = 5101, carry = 0;
5      for (s1--, s2--; s1 >= 0 || s2 >= 0 || carry;)
6      {
7          //长度为0时，数值取0
8          carry += (s1 >= 0 ? num1[s1--] - 48 : 0) + (s2 >= 0 ? num2[s2--] - 48
9          : 0); // 最长遍历处理
10         res[len--] = carry % 10 + 48;
11         carry /= 10;
12     }
13     return &res[len+1]; // 返回指针，取地址
14 }
```

## 二进制求和

### 题目描述

[力扣链接](#)

给你两个二进制字符串，返回它们的和（用二进制表示）。输入为 非空 字符串且只包含数字 1 和 0。

示例 1:

输入: a = "11", b = "1" 输出: "100" 示例 2:

输入: a = "1010", b = "1011" 输出: "10101"

提示:

每个字符串仅由字符 '0' 或 '1' 组成。  $1 \leq a.length, b.length \leq 10^4$  字符串如果不是 "0"，就都不含前导零。

## 解题思路

这种不管是十进制的加减还是二进制的加减本质都是一样的，定义一个carry变量足够了，只要注意下数组的边界条件和结束条件，写对没问题。

## 代码

```

1  char * addBinary(char * a, char * b){
2
3      int carry = 0; //进位
4
5      int length = (strlen(a)>strlen(b)? strlen(a)+2:strlen(b)+2);
6
7      char* result = (char*)malloc(sizeof(char)*length); //开辟空间
8      result[length-1] = '\0';
9      //多开辟两个空间 一个存储进位，一个存储结束符
10     for(int i = strlen(a)-1,j = strlen(b)-1,k = length -2; (i >=0) || (j >=
11         0); i--,j--,k--)
12     {
13         int sum = carry;
14         sum += (i >= 0? a[i]-'0':0);
15         sum += (j >= 0? b[j]-'0':0);
16
17         carry = sum /2;
18         result[k] = '0'+ sum % 2;
19     }
20
21     if(carry == 0) //最后无进位，直接返回
22     //为什么result+1可以 result[0]+1会报错
23     return result+1;
24     else
25     result[0] = '1'; //有进位，补一个最高位
26     return result;
}

```

## 简洁代码

```

1  char * addBinary(char * a, char * b){
2
3      int len1 = strlen(a)-1;
4      int len2 = strlen(b)-1;
5      int carry = 0;
6      //多开辟了两个空间存储进位和结束符
7      char*res = (char*)malloc(sizeof(char)*10002);
8      //倒着开始存储
9      int cnt = 10000;
10     res[10001] = '\0';
11     while(len1>=0 || len2>=0 || carry)
12     {
13         //注意判断数组是否为空
14         carry +=(len1<0?0:a[len1]-'0')+(len2<0?0:b[len2]-'0');
15         res[cnt--] = carry%2+48;
16         carry = carry/2;
17         //注意判断结束条件len1 len2
}

```

```

18     len1 = (len1>=0?len1-1:-1);
19     len2 = (len2>=0?len2-1:-1);
20 }
21
22     return &res[cnt+1];
23 }
```

## 反转字符串

### 题目描述

[力扣链接](#)

编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 char[] 的形式给出。

不要给另外的数组分配额外的空间，你必须原地修改输入数组、使用 O(1) 的额外空间解决这一问题。

你可以假设数组中的所有字符都是 ASCII 码表中的可打印字符。

示例 1：

输入：["h","e","l","l","o"] 输出：["o","l","l","e","h"]

示例 2：

输入：["H","a","n","n","a","h"] 输出：["h","a","n","n","a","H"]

### 解题思路

常规题目，定义两个变量，同时从开头和结尾遍历，交换字符即可

### 代码

```

1 void swap(char *s,int start,int end)
2 {
3     while(start<end)
4     {
5         char temp;
6         temp = s[start];
7         s[start] = s[end];
8         s[end] = temp;
9         start++;
10        end--;
11    }
12 }
13
14 void reversestring(char* s, int ssize){
15
16     int start = 0;
17     int end = ssize-1;
18     swap(s,start,end);
19 }
```

## 反转字符串中的单词

### 题目描述

[力扣链接](#)

给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。

示例 1: 输入: "Let's take LeetCode contest"

输出: "s'teL ekat edoCteeL tsetnoc"

注意: 在字符串中, 每个单词由单个空格分隔, 并且字符串中不会有额外的空格。

## 解题思路

题目要求反转字符串中每个单词的字符顺序, 同时仍保留空格和单词的初始顺序, 只需要找到字符串中每个单词的开始位置跟结束位置, 然后交换这两个位置的字符同时开始位置右移动结束位置左移动即可。

## 代码

```

1 void swap(char *str,int start,int end)
2 {
3
4     while(start<end)
5     {
6         char temp = str[start];
7         str[start] = str[end];
8         str[end] = temp;
9         start++;
10        end--;
11    }
12
13}
14 char * reversewords(char * s){
15
16     int i = 0;
17     int start = 0;
18     int end = 0;
19     int len = strlen(s);
20
21     while(s[end++]!='\0')
22     {
23         //注意边界条件
24         if(s[end] == ' ' || s[end] == '\0')
25         {
26             swap(s,start,end-1);
27             start = end+1;
28         }
29     }
30
31
32     return s;
33 }
```

## 反转字符串中的元音字母

### 题目描述

#### [力扣链接](#)

编写一个函数, 以字符串作为输入, 反转该字符串中的元音字母。

示例 1:

输入: "hello" 输出: "holle" 示例 2:

输入: "leetcode" 输出: "leotcede" 说明: 元音字母不包含字母"y"。

## 解题思路

和之前的反转字符串差不多，只不过在反转以前要判断下是否是元音字母，注意大写字母也要判断。

## 代码

```

1 void swap(char *str,int start,int end)
2 {
3
4     char temp = str[start];
5     str[start] = str[end];
6     str[end] = temp;
7 }
8
9 int vowel(char c)
10 {
11     if(c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||
12         c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U')
13         return 1;
14     else
15         return 0;
16 }
17
18 char * reverseVowels(char * s){
19     int i = 0
20     int j = strlen(s) - 1;
21     while(i < j)
22     {
23         //注意不要越界
24         while(i < j && !vowel(s[i]))
25             i++;
26         while(i < j && !vowel(s[j]))
27             j--;
28         swap(s,i,j);
29         i++;
30         j--;
31     }
32     return s;
33 }
```

## 验证回文串

### 题目描述

[力扣链接](#)

给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。

说明：本题中，我们将空字符串定义为有效的回文串。

示例 1:

输入: "A man, a plan, a canal: Panama" 输出: true

示例 2:

输入: "race a car" 输出: false

## 解题思路

1. 回文串判断，首尾比较。忽略大小写，可以都转换为同一个基准，比如都是小写
2. 只考虑数字和字母，可以使用isalpha,isdigit (数字,字母返回非0, 不是返回0) 来过滤
3. 考虑空指针或空串的场景

## 代码

```

1  bool isPalindrome(char * s){
2
3      int len = strlen(s);
4      //空字符串定义为有效的回文串
5      if (len == 0) return true;
6      //单个字符定义为有效的回文串
7      if (len == 1) return true;
8      //两个指针，一个指向开头，一个指向结尾
9      char *p = s;
10     char *q = s+len-1;
11     //双指针同时遍历字符串
12     while(p<q)
13     {
14         //跳过非字符和非字母，注意每次判断是否两个指针相遇
15         while((!(isalpha(*p))&&!(isdigit(*p)))||(p==q))
16         {
17             p++;
18         }
19
20         while((!(isalpha(*q))&&!(isdigit(*q)))||(p==q))
21         {
22             q--;
23         }
24         //只要有一个不相等就返回false
25         if((tolower(*p)!=tolower(*q)))
26             return false;
27         //下一个字符的比较
28         p++;
29         q--;
30     }
31     return true;
32 }
```

## 验证回文字符串 II

### 题目描述

#### 力扣链接

给定一个非空字符串 s，最多删除一个字符。判断是否能成为回文字符串。

示例 1:

输入: "aba" 输出: True

示例 2:

输入: "abca" 输出: True 解释: 你可以删除c字符。 注意:

字符串只包含从 a-z 的小写字母。字符串的最大长度是50000。

## 解题思路

检查到前后不一样的字符，必须要删掉一个且只能删除一次（要么开头，要么结尾）。删掉之后检查剩下的字符串是否为回文。

以"abdda"这个串为例，此时i指向'b'，j指向'd'，发现不对了。但是有一次删除的机会，我们自己写几个case其实就能发现，此时子串范围为(i+1, j)或(i, j-1)的俩子串只要有任意一个是回文串，则结果就是回文串，否则就不是。

## 代码

```

1  bool isPalindrome (char* start, char* end)    //检查s中一段是否为回文数
2  {
3      while(start <= end) {
4          if(*start != *end) {
5              return false;
6          }
7          start++;
8          end--;
9      }
10     return true;
11 }
12
13 bool validPalindrome(char * s)
14 {
15     int flag = 0;      // 一次删除机会
16     int len = strlen(s);
17     if (len <= 2) {   //单独一字符或者两字符肯定能变成回文
18         return true;
19     }
20     char* start = s;
21     char* end = s + len -1;
22     while (start <= end) {
23         if (*start != *end) {
24             //删除左边
25             if (isPalindrome(start+1, end)) {
26                 return true;
27             }
28             //删除右边
29             if (isPalindrome(start, end - 1)) {
30                 return true;
31             }
32             return false;
33         }
34         start++;
35         end--;
36     }
37     return true;
38 }
39 }
```

## 根据字符串出现频率排序

### 题目描述

#### [力扣链接](#)

给定一个字符串，请将字符串里的字符按照出现的频率降序排列。

示例 1:

输入: "tree"

输出: "eert"

解释: 'e'出现两次，'r'和't'都只出现一次。因此'e'必须出现在'r'和't'之前。此外，"eetr"也是一个有效的答案。示例

2:

输入: "cccaaa"

输出: "cccaaa"

解释: 'c'和'a'都出现三次。此外，"aaaccc"也是有效的答案。注意"cacaca"是不正确的，因为相同的字母必须放在一起。示例

3:

输入: "Aabb"

输出: "bbAa"

解释: 此外，"bbaA"也是一个有效的答案，但"Aabb"是不正确的。注意'A'和'a'被认为是两种不同的字符。

### 解题思路

字符范围为：0-128，利用数组去构建哈希表：

1.首先对字符串进行遍历，取得每个字符出现的次数保存在数组count中

2.循环遍历count数组，每次找出最大值所对应的索引，将其值赋为0（这次下次就不会重复找到它），然后将索引所对应的字符赋值到字符串s中。

### 代码

```

1
2 char * frequencySort(char * s){
3     int flag[128] = {0};
4     int len = strlen(s);
5     if (len <= 2)
6         return s;
7     for(int i = 0;i<len;i++)
8     {
9         printf("%d ",s[i]);
10        flag[s[i]]++;
11    }
12    int i ,j = 0;
13    while(i < len){
14        int maxValue = 0;
15        int maxIndex = 0;
16        //每次都找到最大值 记录下最大值和下标
17        for( j = 0; j < 128; j++){
18            if (flag[j] > maxValue){

```

```

19         maxValue = flag[j];
20         maxIndex = j;
21     }
22 }
23
24 //把最大值对应的内容置0 表示已经打印了
25 if (maxValue != 0){
26     flag[maxIndex] = 0;
27 }
28
29 //将当前的字符串中出现的最大频率的字母放在s中
30 for(j = 0; j < maxValue; j++){
31     s[i++] = maxIndex;
32 }
33 }
34 return s;
35 }

```

## 字符串中的单词

### 题目描述

#### 力扣链接

统计字符串中的单词个数，这里的单词指的是连续的不是空格的字符。

请注意，你可以假定字符串里不包括任何不可打印的字符。

示例:

输入: "Hello, my name is John"

输出: 5

解释: 这里的单词是指连续的不是空格的字符，所以 "Hello," 算作 1 个单词。

### 解题思路

统计前一个为空格且自身不是空格的数量即可。(注意s[0])

### 代码

```

1 int countSegments(char * s){
2
3     if(s[0] == '\0') return 0;
4     int cnt=0;
5     int i = 0;
6     int len = strlen(s);
7     for(i =0; i < len; i++){
8         //注意第一个单词的判断 i == 0
9         if ((i == 0 || s[i-1] == ' ') && s[i] != ' ')
10             cnt++;
11
12     }
13     return cnt;
14
15 }

```

## 前K个高频单词

### 题目描述

#### [力扣链接](#)

给一非空的单词列表，返回前 k 个出现次数最多的单词。

返回的答案应该按单词出现频率由高到低排序。如果不同的单词有相同出现频率，按字母顺序排序。

示例 1：

输入: ["i", "love", "leetcode", "i", "love", "coding"], k = 2 输出: ["i",

"love"] 解析: "i" 和 "love" 为出现次数最多的两个单词，均为2次。

注意，按字母顺序 "i" 在 "love" 之前。

示例 2：

输入: ["the", "day", "is", "sunny", "the", "the", "sunny", "is",

"is"], k = 4 输出: ["the", "is", "sunny", "day"] 解析: "the", "is",

"sunny" 和 "day" 是出现次数最多的四个单词，

出现次数依次为 4, 3, 2 和 1 次。

注意：

假定 k 总为有效值， $1 \leq k \leq$  集合元素数。 输入的单词均由小写字母组成。

扩展练习：

尝试以  $O(n \log k)$  时间复杂度和  $O(n)$  空间复杂度解决。

### 解题思路

qsort的结构体的排序，注意字典序是如何排序的。

### 代码

```

1  /**
2   * Note: The returned array must be malloced, assume caller calls free().
3   */
4  typedef struct node{
5      char word[20];
6      int num;
7  }NODE;
8
9  int cmp1(const NODE* a, const NODE* b){
10     return (b->num - a->num);
11 }
12 int cmp2(const NODE* a, const NODE* b){
13     return strcmp(a->word, b->word);
14 }
15
16 char ** topKFrequent(char ** words, int wordsSize, int k, int* returnSize){
17     NODE* node = calloc(wordsSize, sizeof(NODE));
18     char** ret = calloc(wordsSize, sizeof(char*));
19     int node_len = 0;
20     for (int i = 0; i < wordsSize; i++) { // 建立结构体数组，存储出现次数
21         bool flag = false;
22         // 若前面出现过这个单词，直接累加次数
23         for (int j = 0; j < node_len; j++) {
24             // printf("check : \"%s\" ", node[j].word);
```

```

25         // printf("compare with \"%s\"\n", words[i]);
26         if (strcmp(words[i], node[j].word) == 0) {
27             node[j].num++;
28             flag = true;
29             // printf("node[%d].num++\n", j);
30             break;
31         }
32     }
33     if (!flag) { // 若前面没出现过，新增一个节点
34         strcpy(node[node_len].word, words[i]);
35         node[node_len].num = 1;
36         // printf("add node : \"%s\" %d\n", node[node_len].word,
37         node[node_len].num);
38         node_len++;
39         // puts("----");
40     }
41
42     qsort(node, node_len, sizeof(NODE), cmp2); // 排序字符串
43     qsort(node, node_len, sizeof(NODE), cmp1); // 排序出现次数
44     for (int i = 0; i < k; i++) {
45         // printf("print node : \"%s\" %d\n", node[i].word, node[i].num);
46         ret[i] = node[i].word;
47     }
48
49     *returnsize = k;
50     return ret;
51 }
```

## 检测大写字母

### 题目描述

[力扣链接](#)

给定一个单词，你需要判断单词的大写使用是否正确。

我们定义，在以下情况时，单词的大写用法是正确的：

全部字母都是大写，比如"USA"。

单词中所有字母都不是大写，比如"leetcode"。

如果单词不只含有一个字母，只有首字母大写，比如 "Google"。

否则，我们定义这个单词没有正确使用大写字母。

示例 1:

输入: "USA" 输出: True

示例 2:

输入: "FlaG" 输出: False 注意: 输入是由大写和小写拉丁字母组成的非空单词。

### 解题思路

检测大写字母的个数即可，

1.当字符串全为大写字母或者没有一个大写字母，返回true。

2.当字符串第一个为大写并且后面的不为大写字母是，返回true。

3.其余情况返回false

## 代码

```

1  bool detectCapitalUse(char * word){
2
3      int len = strlen(word);
4      int cnt = 0;
5      for(int i = 0;i<len;i++)
6      {
7          if((word[i] >='A' )&&(word[i] <='Z'))
8              cnt++;
9      }
10     //对应情况1和情况3
11     if((cnt == len)|| (cnt == 0))
12         return true;
13     //对应情况2 要考虑到首字母
14     else if(((word[0]>='A')&&(word[0]<='Z')&&(cnt==1)))
15         return true;
16     else
17         return false;

```

## 最长的公共前缀

### 题目描述

[力扣链接](#)

编写一个函数来查找字符串数组中的最长公共前缀。

如果不存在公共前缀，返回空字符串 ""。

示例 1:

输入: ["flower","flow","flight"] 输出: "fl" 示例 2:

输入: ["dog","racecar","car"] 输出: "" 解释: 输入不存在公共前缀。说明:

所有输入只包含小写字母 a-z 。

### 解题思路

以第一个字符串为基准进行竖向扫描，将每个字符串的前`col`个字符进行依次比较。

## 代码

```

1  char * longestCommonPrefix(char ** strs, int strssize){
2      if(0==strssize) return "";
3      char *s0= strs[0]; //首个字符串作为基准
4      for(int col=0; s0[col]!='\0' ;++col){ //s0的第col个字符作为基准
5          for(int row=1; row<strssize; ++row){//扫描其他字符串，仅当所有字符串的第
6              col个字符都同基准，才会正常结束循环
7                  if(s0[col]!=strs[row][col]){//不会越界！因为即将越界会有 '\0'停止
8                     符作为哨兵，此时马上跳出。
9                      s0[col]='\0'; //用'\0'表示结束，改短s0
10                     return s0;
11                 }
12             } //而且C语言即便越界，只要越的界没有超出整个进程代码段的地址空间，都不会报错。
13         } //其他字符串不存在或都包含s0的前缀
14     }

```

## 最长特殊序列 I

### 题目描述

#### [力扣链接](#)

给你两个字符串，请你从这两个字符串中找出最长的特殊序列。

「最长特殊序列」定义如下：该序列为某字符串独有的最长子序列（即不能是其他字符串的子序列）。子序列可以通过删去字符串中的某些字符实现，但不能改变剩余字符的相对顺序。空序列为所有字符串的子序列，任何字符串为其自身的子序列。

输入为两个字符串，输出最长特殊序列的长度。如果不存在，则返回 -1。

示例 1：

输入：“aba”，“cdc” 输出：3 解释：最长特殊序列可为 “aba”（或 “cdc”），两者均为自身的子序列且不是对方的子序列。示例 2：

输入：a = “aaa”，b = “bbb” 输出：3 示例 3：

输入：a = “aaa”，b = “aaa” 输出：-1

提示：

两个字符串长度均处于区间 [1 - 100]。字符串中的字符仅含有 ‘a’~‘z’。

### 解题思路

标签：题意理解，本题题意难于理解。独有指的是只有自己有，另一个字符串没有  
举例说明，设两个字符串变量名分别为 a 和 b

a = ‘c’，b = ‘cd’，‘cd’ 是 b 独有的，所以最长子序列为 ‘cd’，长度为 2

a = ‘cd’，b = ‘cd’，‘cd’，‘c’，‘d’ 在两个字符串中都有，所以不存在独有的最长子序列，返回 -1

通过举例分析，得出以下结论：

如果两个字符串长度不一样，则较长的字符串本身不可能是短字符串的子序列，直接返回其长度即可

如果两个字符串内容相等，那么他们独有的最长子序列不存在，返回 -1

### 代码

```

1 int findLUSlength(char * a, char * b){
2     int lenA=strlen(a),lenB=strlen(b);
3     if (strcmp(a,b)==0) return -1;
4     return lenA>lenB?lenA:lenB;
5 }
```

## 最长特殊序列 II

### 题目描述

#### [力扣链接](#)

给定字符串列表，你需要从它们中找出最长的特殊序列。最长特殊序列定义如下：该序列为某字符串独有的最长子序列（即不能是其他字符串的子序列）。

子序列可以通过删去字符串中的某些字符实现，但不能改变剩余字符的相对顺序。空序列为所有字符串的子序列，任何字符串为其自身的子序列。

输入将是一个字符串列表，输出是最长特殊序列的长度。如果最长特殊序列不存在，返回 -1。

示例：

输入：“aba”, “cdc”, “eae” 输出: 3

提示：

所有给定的字符串长度不会超过 10 。 给定字符串列表的长度将在 [2, 50 ] 之间。

## 解题思路

思路参考最长特殊序列 I

由上一题我们知道判断两个字符串的最长特殊序列只要比较两个字符串是否相等、长度即可，但是在此道题并不能这样判断，因为长度大的字符串可能存在重复（重复了一定不能作为整个字符串数组的最长特殊序列），长度短的字符串可能是长度大的字符串的子序列。（比如“ab”是“acbd”的子序列）。所以我们的重新定义两个字符串是否是子序列的问题。

算法如下：

第一步：将整个字符串数组按照长度降序排序

第二步：统计各个字符串出现的次数

第三步：寻找可以作为整个字符串数组的最长序列

## 代码

```

1 int cmp1(const void *a, const void *b)
2 {
3     return strlen(*(char **)b) - strlen(*(char **)a);
4 }
5
6 int cmp2(const void *a, const void *b)
7 {
8     return strcmp(*(char **)a, *(char **)b);
9 }
10 //a是不是b的子序列 a为当前序列 b为当前序列之前的序列
11 bool isSequence(char *a, char *b)
12 {
13     int len1 = strlen(a);
14     int len2 = strlen(b);
15     printf("%d,%d\r\n", len1, len2);
16     int count = 0;
17     int j = 0;
18     for (int i = 0; i < len1; i++) {
19         while (j < len2) {
20             if (a[i] == b[j++]) {
21                 count++;
22                 break;
23             }
24             // j++;
25         }
26     }
27     if (count == len1)
28         return true;
29     return false;
30 }
31
32 int findLUSlength(char ** strs, int strssize){
33     bool flag = false;
34     qsort(strs, strssize, sizeof(strs[0]), cmp2);

```

```

35     qsort(strs, strssize, sizeof(strs[0]), cmp1);
36     // 向前查询是否为子串
37     for (int i = 0; i < strssize; i++) {
38         bool f1 = true, f2 = true;
39         //从第二个序列开始，往前查询 之前的序列是否为当前序列的子序列
40         if (i >= 1) {
41             for (int j = 0; j < i; j++) {
42                 //第二个序列strs[i] 是不是第一个序列strs[j]的子序列
43                 if (isSequence(strs[i], strs[j])) {
44                     f1 = false;
45                     break;
46                 }
47             }
48         }
49         // 向后查询有无重复字符串
50         for (int k = i + 1; k < strssize; k++) {
51             //长度不同 内容肯定不同 跳出循环，继续往后比较
52             if (strlen(strs[k]) != strlen(strs[i]))
53                 break;
54             //长度相同时，比较内容是否相同
55             if (strcmp(strs[k], strs[i]) == 0) {
56                 f2 = false;
57                 //内容不同时，将从此序列开始比较
58                 i = k;
59             }
60         }
61         if (f1 && f2)
62             return strlen(strs[i]);
63     }
64
65     return -1;
66 }
```

## 计数二进制子串

### 题目描述

[力扣链接](#)

给定一个字符串 s，计算具有相同数量0和1的非空(连续)子字符串的数量，并且这些子字符串中的所有0和所有1都是组合在一起的。

重复出现的子串要计算它们出现的次数。

示例 1：

输入: "00110011" 输出: 6 解释:

有6个子串具有相同数量的连续1和0: "0011", "01", "1100", "10", "0011" 和 "01"。

请注意，一些重复出现的子串要计算它们出现的次数。

另外，“00110011”不是有效的子串，因为所有的0（和1）没有组合在一起。

示例 2：

输入: "10101" 输出: 4 解释: 有4个子串: "10", "01", "10", "01"，它们具有相同数量的连续1和0。注意:

s.length 在1到50,000之间。s 只包含“0”或“1”字符。

## 解题思路

假设最简单的情况 000111, 先遍历, 前面的 0 的数量为 curr = 3, 遍历到 1 时, pre = curr 赋值为 3, 然后 curr = 1 表示现在 1 的个数, 只要 curr <= pre, 比如 curr = 1, 那么可以组成 01; curr = 2, 可以组成 0011, curr = 3, 组成 000111, 直到出现下一次 0, 然后那么 prev 就是 1 的个数, curr 表示 0001110。这个 1 后面 0 的个数, 不断重复迭代下去。

## 代码

```

1 int countBinarySubstrings(char * s){
2     int n = 0, pre = 0, curr = 1, len = strlen(s)-1;
3     for (int i = 0; i < len; ++i) {
4         if (s[i] == s[i+1]) ++curr;
5         else {pre = curr; curr = 1;}
6         if (pre >= curr) ++n;
7     }
8     return n;
9 }
```

## 实现字符串的库函数

### strcpy

```

1 void mycpy(char *s1, char *s2)
2 {
3     while(*s1++ = *s2++);
4 }
```

### strlen

```

1 int mylen(char *s1)
2 {
3     char *p = s1;
4     while(*p++);
5     return p - s -1;
6 }
```

### strcat

```

1 void mycat(char *s1, char *s2)
2 {
3     while(*s1++);
4     s1--;
5     while(*s1++ = *s2++);
6 }
```

### atoi

```

1 int myatoi (char* str)
2 {
3     if(str == NULL)
4     {
5         printf("Invalid Input");
6         return -1;
7     }
}
```

```

8  while(*str == ' ')//去掉字符串开始的空格
9  {
10     str++;
11 }
12 int nSign = (*str=='-')?-1:1;//确定符号位
13 if(*str == '+' || *str == '-')
14 {
15     str++;
16 }
17 int nResult = 0;
18 while(*str>='0'&&*str<='9')
19 {
20     nResult = nResult*10+(*str-'0');
21     str++;
22 }
23 while(*str == ' ')
24 {
25     str++;
26 }
27 if(*str!='\0')
28 {
29     printf("Invalid Input");
30     return -1;
31 }
32 return nResult*nSign;
33 }
34
35 int main()
36 {
37     printf("%d\r\n",myatoi("123465"));
38     return 0;
39 }
40

```

## itoa

```

1 char *myitoa(int num,char *str,int radix)
2 {
3     char *ptr = str;
4     int temp;
5     int sign = num,i = 0,j = 0;
6     if(sign<0)
7     {
8         num =-num;
9     }
10    while(num)
11    {
12        //先存储低位
13        *ptr++ = num%radix+'0';
14        num/=radix;
15        //小于进制数直接放进去
16        if(num<radix)
17        {
18            *ptr++=num+'0';
19            break;
20        }
21    }

```

```

22 //最高位存储符号位
23 if(sign<0)
24 {
25     *ptr++='-' ;
26 }
27 *ptr = '\0';
28 //存储的是反过来的 我们需要调换位置
29 j = ptr-str-1;
30 for(i=0;i<j;i++)
31 {
32     int temp = str[i];
33     str[i] = str[j];
34     str[j--]=temp;
35 }
36 return str;
37 }
38 int main()
39 {
40     char str[6];
41     myitoa(-12345,str,10);
42     printf("%s\n",str);
43     itoa(12345,str,10);
44     printf("%s\n",str);
45     return 0;
46 }
```

## memmove

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4
5 void * my_memmove( void * const dest, const char * const src, size_t n )
6 {
7     // check parameters
8     if( 0 == n )
9     {
10         return NULL;
11     }
12     if( NULL == dest || NULL == src )
13     {
14         return NULL;
15     }
16
17     char * psrc = (char *)src;
18     char * pdest = (char *)dest;
19     if( pdest <= psrc || pdest > psrc + n )
20     {
21         std::cout << "forward overlapping" << std::endl;
22         // copy forward direction
23         for( size_t i = 0; i < n; i++ )
24         {
25             *pdest = *psrc;
26             pdest++;
27             psrc++;
28         }
29     }
30 }
```

```

30     else
31     {
32         std::cout << "backward overlapping" << std::endl;
33         // copy backward direction
34         pdest = pdest + n;
35         psrc = psrc + n;
36         for( size_t i = 0; i < n; i++ )
37         {
38             *pdest = *psrc;
39             pdest--;
40             psrc--;
41         }
42     }
43     return dest;
44 }
45
46 int main( int argc, char ** argv )
47 {
48     char *src = new char[100];
49     sprintf( src, "%s", "hello world!" );
50     char * dest = new char[100];
51     memset( dest, 0, 100*sizeof(char) );
52     std::cout << src << std::endl;
53     char * result = (char*)my_memmove( dest, src, strlen(src) );
54     std::cout << result << std::endl;
55     delete src;
56     delete dest;
57     return 0;
58 }
```



## 排序算法及其改进方法

### 快速排序

#### 基本思想

我们从数组中选择一个元素，我们把这个元素称之为中轴元素吧，然后把数组中所有小于中轴元素的元素放在其左边，所有大于或等于中轴元素的元素放在其右边，显然，此时中轴元素所处的位置的是有序的。也就是说，我们无需再移动中轴元素的位置。

从中轴元素那里开始把大的数组切割成两个小的数组(两个数组都不包含中轴元素)，接着我们通过递归的方式，让中轴元素左边的数组和右边的数组也重复同样的操作，直到数组的大小为1，此时每个元素都处于有序的位置。

#### 具体步骤：

- 1.从数列中挑出一个元素，称为“基准”(pivot)；

2.重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区(partition)操作；

3.递归地(recursive)把小于基准值元素的子数列和大于基准值元素的子数列排序；

**最好时间复杂度：** $O(N \log N)$

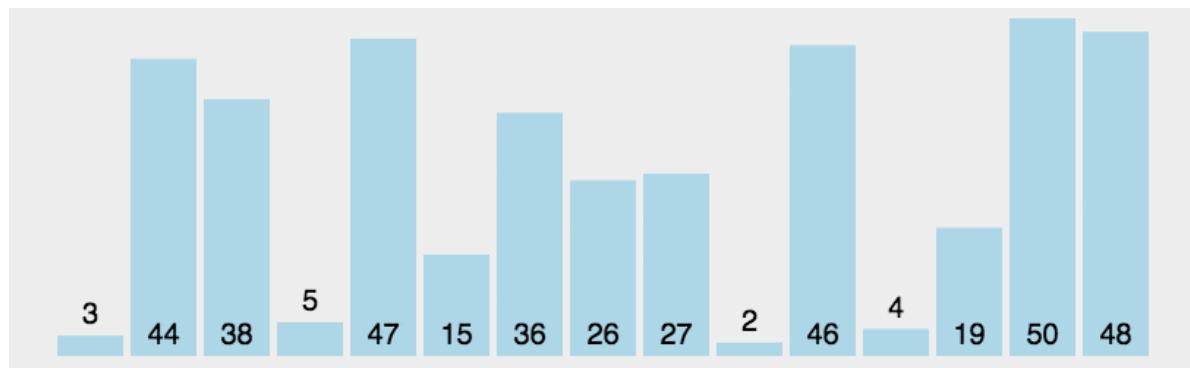
**最坏时间复杂度：** $O(N^2)$

**最坏时间复杂度：** $O(N \lg N)$

**空间复杂度：** $O(\log N)$

**稳定排序：**否

**原地排序：**是



## 代码

```

1 int *Quicksort(Node* pBegin, Node* pEnd)
2 {
3     if(pBegin == NULL || pEnd == NULL || pBegin == pEnd)
4         return 0;
5
6     //定义两个指针
7     Node* p1 = pBegin;
8     Node* p2 = pBegin->next;
9     int pivot = pBegin->data;
10
11    //每次只比较小的，把小的放在前面。经过一轮比较后，被分成左右两部分。其中p1指向中值处，pbegin为pivot。
12    while(p2 != NULL)/* && p2 != pEnd->next */
13    {
14        if(p2->data < pivot)
15        {
16            p1 = p1->next;
17            if(p1 != p2)
18            {
19                SwapData(&p1->data, &p2->data);
20            }
21        }
22        p2 = p2->next;
23    }
24    /*此时pivot并不在中间，我们要把他放到中间，以他为基准，把数据分为左右两边*/
25    SwapData(&p1->data, &pBegin->data);
26    //此时p1是中值节点
27    //if(p1->data >pBegin->data)
28    Quicksort(pBegin, p1);

```

```

29 //if(p1->data < pEnd->data)
30 QuickSort(p1->next, pEnd);
31
32 }

```

## 改进方案

### 1. 选取随机数作为枢轴。

但是随机数的生成本身是一种代价，根本减少不了算法其余部分的平均运行时间。

### 2. 使用左端，右端和中心的中值做为枢轴元。

经验得知，选取左端，右端，中心元素的中值会减少了快排大约\*14%的比较。

### 3. 每次选取数据集中的中位数做枢轴。

选取中位数的可以在  $O(n)$  时间内完成。（证明见《算法导论（第二版）》） P111 第九章：在平均情况下，任何顺序统计量（特别是中位数）都可以在线性时间内得到。

### 4. 快速排序在处理小规模数据时的表现不好。这个时候可以改用插入排序。

### 5. 对于一个每个元素都完全相同的一个序列来讲，快速排序也会退化到 $O(n^2)$

要将这种情况避免到，可以这样做：

在分区的时候，将序列分为 3 堆，一堆小于中轴元素，一堆等于中轴元素，一堆大于中轴元素，下次递归调用快速排序的时候，只需对小于和大于中轴元素的两堆数据进行排序，中间等于中轴元素的一堆已经放好。

## 冒泡排序

### 基本思想：

把第一个元素与第二个元素比较，如果第一个比第二个大，则交换他们的位置。接着继续比较第二个与第三个元素，如果第二个比第三个大，则交换他们的位置。

我们对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对，这样一趟比较交换下来之后，排在最右的元素就会是最大的数。除去最右的元素，我们对剩余的元素做同样的工作，如此重复下去，直到排序完成。

### 具体步骤：

1. 比较相邻的元素。如果第一个比第二个大，就交换他们两个。

2. 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。

3. 针对所有的元素重复以上的步骤，除了最后一个。

4. 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

**最好时间复杂度：**  $O(N)$

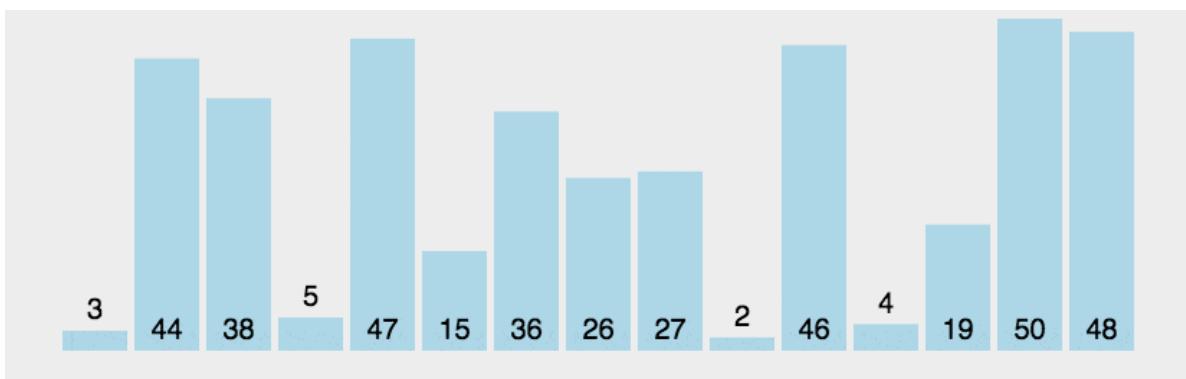
**最坏时间复杂度：**  $O(N^2)$

**平均时间复杂度：**  $O(N^2)$

**空间复杂度：**  $O(1)$

**稳定排序:** 是

**原地排序:** 是



## 代码

```

1 void bubblesort(int *arr,int n)
2 {
3     int m,i,j;
4     for(i=0;i<n-1;i++)
5         for(j=0;j<n-1-i;j++)
6             if(arr[j]>arr[j+1])
7             {
8                 m=arr[j];
9                 arr[j]=arr[j+1];
10                arr[j+1]=m;
11            }
12 }
```

## 改进

1.一般的冒泡排序方法有可能会在已经排好序的情况下继续比较，改进的冒泡排序，设置了一个哨兵flag，如果一次for循环没有进行交换，则元素已经排好序，由哨兵控制退出循环。

2.有序区的长度和排序的轮数是相等的。比如第一轮排序过后的有序区长度是1，第二轮排序过后的有序区长度是2。实际上，数列真正的有序区可能会大于这个长度，比如例子中仅仅第二轮，后面5个元素实际都已经属于有序区。因此后面的许多次元素比较是没有意义的。如何避免这种情况呢？我们可以在每一轮排序的最后，记录下最后一次元素交换的位置，那个位置也就是无序数列的边界，再往后就是有序区了。

## 堆排序

### 基本思想

将待排序的序列构造成一个大顶堆。此时，整个序列的最大值就是堆顶的根节点。将它移走（其实就是将其与堆数组的末尾元素交换，此时末尾元素就是最大值），然后将剩余的  $n-1$  个序列重新构造成一个堆，这样就会得到  $n$  个元素中的次大值。如此反复执行，便能得到一个有序序列了。

### 具体步骤

1. 将待排序序列构造成一个大顶堆。
2. 此时，整个序列的最大值就是堆顶的根节点。
3. 将其与末尾元素进行交换，此时末尾就为最大值。
4. 然后将剩余  $n-1$  个元素重新构造成一个堆，这样会得到  $n$  个元素的次小值。如此反复执行，便能得到一个有序序列了。

可以看到在构建大顶堆的过程中，元素的个数逐渐减少，最后就得到一个有序序列了。

**最好时间复杂度：**  $O(N \log N)$

**最坏时间复杂度：**  $O(N \log N)$

**平均时间复杂度:**  $O(N \log N)$

**空间复杂度:**  $O(1)$

**稳定排序:**否

**原地排序:**是

[动图点击这里](#)

**代码**

```

1 void heapify(Mytype tree[], int n, int i)
2 {
3     if (i >= n)
4     {
5         return;
6     }
7     //当前父节点的两个子节点
8     int c1 = 2 * i + 1;
9     int c2 = 2 * i + 2;
10    //假设根节点为最大值
11    int max = i;
12    //找出子树中三个节点中的最大的节点
13    // if (c1 < n )
14    // {
15    //     max = (tree[c1] > tree[max]) ? c1:c2;
16    // }
17    if (c1 < n && tree[c1] > tree[max])
18    {
19        max = c1;
20    }
21    if (c2 < n && tree[c2] > tree[max])
22    {
23        max = c2;
24    }
25    //将最大节点和当前节点交换位置
26    if (max != i)
27    {
28        swap(tree, max, i);
29        //继续进行下一个节点的 构建堆
30        heapify(tree, n, max);
31    }
32 }
33
34 int build_heap (Mytype tree[],int n)
35 {
36     int last_node = n-1;
37     //最后一个父节点
38     int parent = (last_node-1)/2;
39     int i;
40     //从最后一个父节点开始，由下到上遍历每一个父节点依次调整构建堆
41     for(i = parent;i>=0;i--)
42     {
43         heapify(tree,n,i);
44     }
45 }
46 /**

```

```

47 * @Description: 堆排序时间复杂度  $O(n \log n)$  思想: 构建大根堆/小根堆, 将堆顶元素与最后一个叶子节点交换, 切断叶子节点 (B站正月点灯笼)
48 * @Param: Mytype tree[] 数组 int n 数组中元素的个数
49 * @Return: 无
50 * @Author: Carlos
51 */
52 int HeapSort (Mytype tree[], int n)
53 {
54     //构建堆 第一个节点为堆中最大的数
55     build_heap(tree,n);
56     int i;
57     //每次都换出来一个最大值
58     for(i = n-1;i>=0;i--)
59     {
60         //交换最后一个节点和第一个节点
61         swap(tree,i,0);
62         //每次都截断最后一个元素, 我们可以默认最后一个元素已经被截断了, i即为剩下的元素,
63         //每次传入i的值即可继续构建0--i范围内的元素
64         heapify(tree,i,0);
65     }
}

```

## 改进

## 插入排序

**基本思想:**每一步将一个待排序的记录，插入到前面已经排好序的有序序列中去，直到插完所有元素为止。

**具体步骤:**

- 1.将待排序序列第一个元素看做一个有序序列，把第二个元素到最后一个元素当成是未排序序列；
- 2.取出下一个元素，在已经排序的元素序列中从后向前扫描；
- 3.如果该元素（已排序）大于新元素，将该元素移到下一位置；
- 4.重复步骤3，直到找到已排序的元素小于或者等于新元素的位置；
- 5.将新元素插入到该位置后；
- 6.重复步骤2~5。

**最好时间复杂度:**  $O(N)$

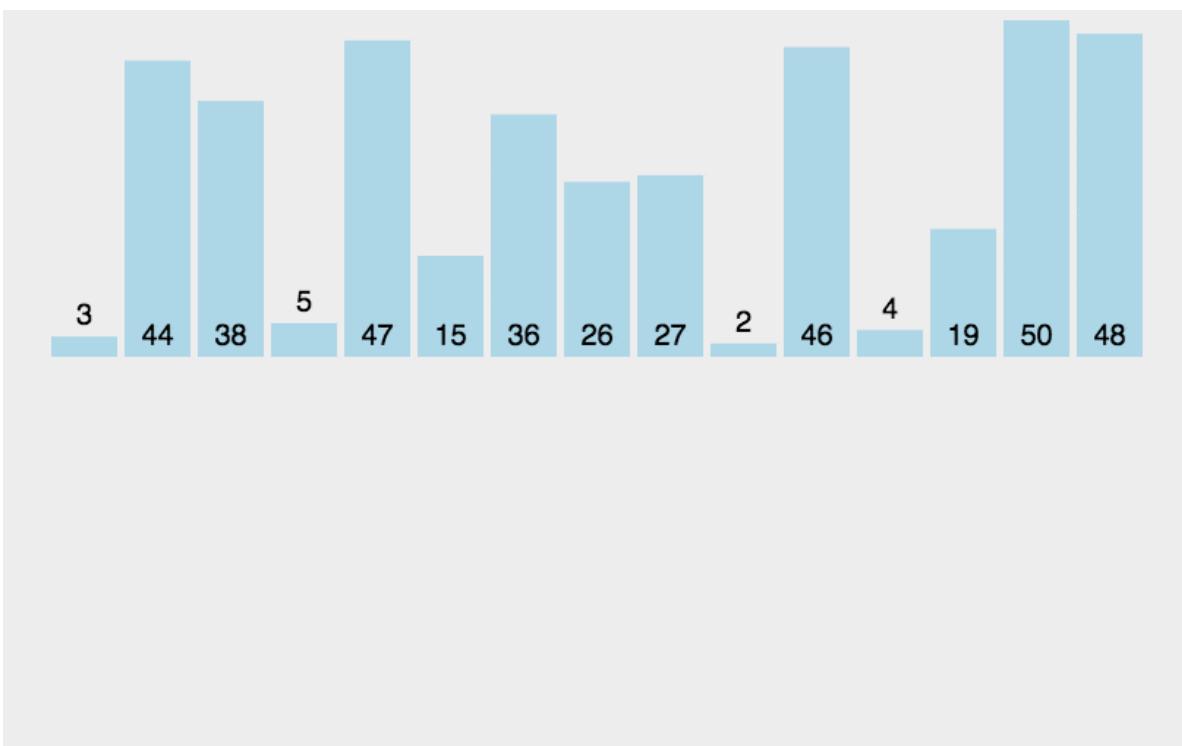
**最坏时间复杂度:**  $O(N^2)$

**平均时间复杂度:**  $O(N^2)$

**空间复杂度:**  $O(1)$

**稳定排序:** 是

**原地排序:** 是



## 代码

```

1 //插入排序
2 //直接插入排序函数 升序
3 void InsertSort(Mytype a[], int n)
4 {
5     //注意从第二个元素开始
6     for(int i= 1; i<n; i++)
7     {
8         //如果a[i-1]本来就是小于a[i]的，就不需要做任何操作。当a[i-1]是较大的元素，需要
9         //从i向前找，找到合适的位置，插在a[i]前面
10        if(a[i] < a[i-1])
11            {//若第 i 个元素大于 i-1 元素则直接插入；反之，需要找到适当的插入位置后在插入。
12                int j= i-1;
13                int x = a[i];
14                //在0到j-1的中找到 a[i-1]<a[i]的位置
15                //从第j个元素开始，即i的前一个。找一个比i位置元素小的位置
16                while(j>-1 && x < a[j])
17                    { //采用顺序查找方式找到插入的位置，在查找的同时，将数组中的元素进行后移操
18                     //作，给插入元素腾出空间
19                         a[j+1] = a[j];
20                         j--;
21                     }
22                 //跳出循环时，找到了x > a[j] 的恰当位置
23                 a[j+1] = x;           //插入到正确位置
24             }
25     }
26 }
```

## 改进

在插入某个元素之前需要先确定该元素在有序数组中的位置，原始的做法是对有序数组中的元素逐个扫描，当数据量比较大的时候，这是一个很耗时间的过程，可以采用二分查找法改进，这种排序也被称为二分插入排序。

## 选择排序

### 基本思想

首先，找到数组中最小的那个元素，其次，将它和数组的第一个元素交换位置(如果第一个元素就是最小元素那么它就和自己交换)。其次，在剩下的元素中找到最小的元素，将它与数组的第二个元素交换位置。如此往复，直到将整个数组排序。这种方法我们称之为选择排序。

### 具体步骤

- 1.首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置。
- 2.再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。
- 3.重复第二步，直到所有元素均排序完毕。

**最好时间复杂度：**  $O(N^2)$

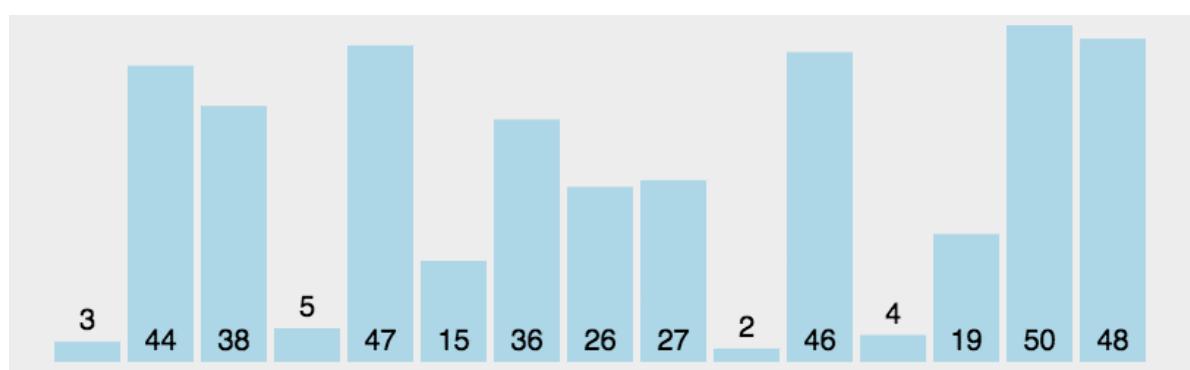
**最坏时间复杂度：**  $O(N^2)$

**平均时间复杂度：**  $O(N^2)$

**空间复杂度：**  $O(1)$

**稳定排序：** 否

**原地排序：** 是



```

1 /**
2  * @Description: 从i的位置开始，寻找最大值
3  * @Param: int arr[] 数组名 int i 查找最大值开始的位置, int n 数组长度
4  * @Return: 最大值的位置
5  * @Author: Carlos
6 */
7 int FindMaxPos1(Mytype arr[], int i, int n)
8 {
9     //假设第i 的位置为最大值
10    int max=i;
11    //从i+1开始比较
12    while (i+1<n) {
13        if (arr[max]<arr[i+1])
14        {
15            max=i+1;
16        }
17        i++;
18    }
19    return max;
20 }
21 /**
22 * @Description: 从i的位置开始，寻找最小值

```

```

23 * @Param: int arr[] 数组名 int i 查找最小值开始的位置, int n 数组长度
24 * @Return: 最小值的位置
25 * @Author: Carlos
26 */
27 int FindMinPos1(Mytype arr[],int i,int n)
28 {
29     //假设第i 的位置为最小值
30     int min=i;
31     //从i+1开始比较
32     while (i+1<n) {
33         if (arr[min]>arr[i+1])
34         {
35             min=i+1;
36         }
37         i++;
38     }
39     return min;
40 }
41 /**
42 * @Description: 选择排序, 对于具有 n 个记录的无序表遍历 n-1 次
43 * 首先在未排序序列中找到最小(大)元素, 存放到排序序列的起始位置。
44     再从剩余未排序元素中继续寻找最小(大)元素, 然后放到已排序序列的末尾。
45     重复第二步, 直到所有元素均排序完毕。
46 * O(n2)
47 * @Param: arr[]数组 n数组大小
48 * @Return: 无
49 * @Author: Carlos
50 */
51 void selectsort1(Mytype arr[],int n)
52 {
53     //从第二个数开始
54     for(int i = 0;i<n;i++)
55     {
56         //每次从i开始, 在i到n中找一个最小值出来
57         int pos = FindMinPos1(arr,i,n);
58         if(pos!=i)
59             swap(arr,pos,i);
60     }
61 }
```

## 改进

简单选择排序，每趟循环只能确定一个元素排序后的定位。我们可以考虑改进为每趟循环确定两个元素（当前趟最大和最小记录）的位置，从而减少排序所需的循环次数。改进后对n个数据进行排序，最多只需进行[n/2]趟循环即可。

## 归并排序

### 基本思想

归并排序是利用**归并**的思想实现的排序方法，该算法采用经典的**分治**（divide-and-conquer）策略（分治法将问题**分**(divide)成一些小的问题然后递归求解，而**治**(conquer)的阶段则将分的阶段得到的各答案“修补”在一起，即分而治之）。实际上就是将数据序列划分为越来越小的半子表，再对半子表排序，最后再用递归方法将排好序的半子表合并成越来越大的有序序列。

### 具体步骤

1. 申请空间，使其大小为两个已经排序序列之和，该空间用来存放合并后的序列；
2. 设定两个指针，最初位置分别为两个已经排序序列的起始位置；
3. 比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置；
4. 重复步骤 3 直到某一指针达到序列尾；
5. 将另一序列剩下的所有元素直接复制到合并序列尾。

**最好时间复杂度：**  $O(N \log N)$

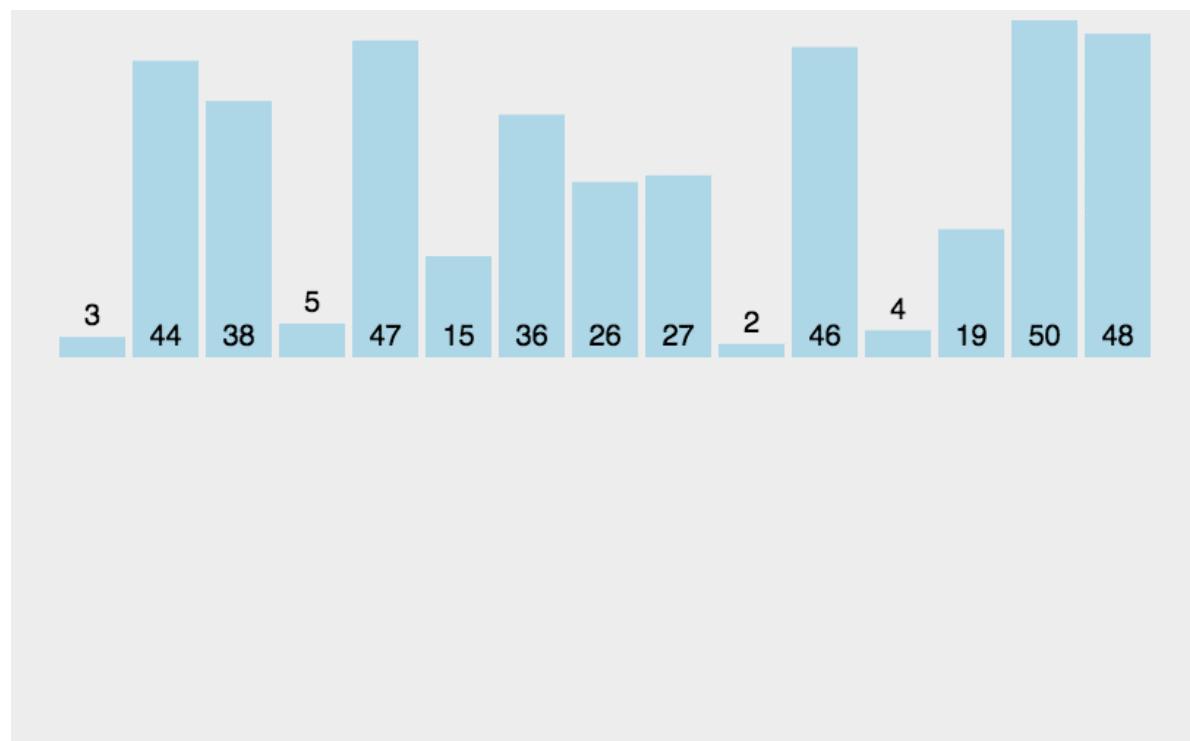
**最坏时间复杂度：**  $O(N \log N)$

**平均时间复杂度：**  $O(N \log N)$

**空间复杂度：**  $O(N)$

**稳定排序:** 是

**原地排序:** 是



## 代码

```

1 #include <iostream>
2 #define MAXSIZE 10
3
4 using namespace std;
5
6 // 实现归并，并把最后的结果存放到list1里
7 void merging(int *list1, int list1_size, int *list2, int list2_size)
8 {
9     int i, j, k, m;
10    int temp[MAXSIZE];
11
12    i = j = k = 0;
13
14    while (i < list1_size && j < list2_size)
15    {
16        if (list1[i] < list2[j])
17        {
18            temp[k++] = list1[i++];

```

```

19         }
20     else
21     {
22         temp[k++] = list2[j++];
23     }
24 }
25
26 while (i < list1_size)
27 {
28     temp[k++] = list1[i++];
29 }
30
31 while (j < list2_size)
32 {
33     temp[k++] = list2[j++];
34 }
35
36 for (m = 0; m < (list1_size + list2_size); m++)
37 {
38     list1[m] = temp[m];
39 }
40 }

41
42 void MergeSort(int k[], int n)
43 {
44     if (n > 1)
45     {
46         int *list1 = k;
47         int list1_size = n / 2;
48         int *list2 = k + n / 2;
49         int list2_size = n - list1_size;
50
51         MergeSort(list1, list1_size);
52         MergeSort(list2, list2_size);
53
54         merging(list1, list1_size, list2, list2_size);
55     }
56 }
57
58 int main()
59 {
60     int i, a[10] = { 5, 2, 6, 0, 3, 9, 1, 7, 4, 8 };
61
62     cout << "排序前的数组是: ";
63     for (i = 0; i < 10; i++)
64     {
65         cout << a[i];
66     }
67     cout << endl;
68
69     MergeSort(a, 10);
70
71     cout << "排序后的数组是: ";
72     for (i = 0; i < 10; i++)
73     {
74         cout << a[i];
75     }
76     cout << endl;

```

```

77     return 0;
78 }
79 }
```

## 非递归

```

1 #include <iostream>
2 #include <malloc.h>
3 #define MAXSIZE 10
4
5 using namespace std;
6
7 void MergeSort(int k[], int n)
8 {
9     int i, next, left_min, left_max, right_min, right_max;
10    //开辟一个与原来数组一样大小的空间用来存储用
11    int *temp = (int *)malloc(n * sizeof(int));
12    //逐级上升，第一次比较2个，第二次比较4个，第三次比较8个。。
13    for(i=1; i<n; i*=2)
14    {
15        //每次都从0开始，数组的头元素开始
16        for(left_min=0; left_min<n-i; left_min = right_max)
17        {
18            right_min = left_max = left_min + i;
19            right_max = left_max + i;
20            //右边的下标最大值只能为n
21            if(right_max>n)
22            {
23                right_max = n;
24            }
25            //next是用来标志temp数组下标的，由于每次数据都有返回到K,
26            //故每次开始得重新置零
27            next = 0;
28            //如果左边的数据还没达到分割线且右边的数组没到达分割线，开始循环
29            while(left_min<left_max&&right_min<right_max)
30            {
31                if(k[left_min] < k[right_min])
32                {
33                    temp[next++] = k[left_min++];
34
35                }
36                else
37                {
38                    temp[next++] = k[right_min++];
39                }
40            }
41            //上面循环结束的条件有两个，如果是左边的游标尚未到达，那么需要把
42            //数组接回去，可能会有疑问，那如果右边的没到达呢，其实模拟一下就可以
43            //知道，如果右边没到达，那么说明右边的数据比较大，这时也就不用移动位置了
44
45            while(left_min < left_max)
46            {
47                //如果left_min小于left_max，说明现在左边的数据比较大
48                //直接把它们接到数组的min之前就行
49                k[--right_min] = k[--left_max];
50            }
51            while(next>0)
```

```

52     {
53         //把排好序的那部分数组返回该k
54         k[--right_min] = temp[--next];
55     }
56 }
57 }
58 }
59
60 int main()
61 {
62     int i, a[10] = { 5, 2, 6, 0, 3, 9, 1, 7, 4, 8 };
63
64     cout << "排序前的数组是: ";
65     for (i = 0; i < 10; i++)
66     {
67         cout << a[i];
68     }
69     cout << endl;
70
71     MergeSort(a, 10);
72
73     cout << "排序后的数组是: ";
74     for (i = 0; i < 10; i++)
75     {
76         cout << a[i];
77     }
78     cout << endl;
79
80     return 0;
81 }
```

## 改进方法

归并排序的时间复杂度为 $O(n\lg n)$ ，一般来讲，基于从单个记录开始两两归并的排序并不是特别提倡，一种比较常用的改进就是结合插入排序，即先利用插入排序获得较长的有序子序列，然后再两两归并（改进后的归并亦是稳定的，因为插入排序是稳定的）。

原因：尽管插入排序的最坏情况是 $O(n^2)$ ，看起来大于归并的最坏情况 $O(n\lg n)$ ，但通常情况下，由于插入排序的常数因子使得它在n比较小的情况下能运行的更快一些，因此，归并时，当子问题足够小时，采用插入排序是比较合适的。

## 希尔排序

### 基本思想

希尔排序，也称递减增量排序算法，是插入排序的一种更高效的改进版本。但希尔排序是非稳定排序算法。

希尔排序是基于插入排序的以下两点性质而提出改进方法的：

- 插入排序在对几乎已经排好序的数据操作时，效率高，即可以达到线性排序的效率；
- 但插入排序一般来说是低效的，因为插入排序每次只能将数据移动一位；

希尔排序的基本思想是：先将整个待排序的记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对全体记录进行依次直接插入排序。

### 具体步骤

选择一个增量序列  $t_1, t_2, \dots, t_k$ , 其中  $t_i > t_j, t_k = 1$ ;

按增量序列个数  $k$ , 对序列进行  $k$  趟排序;

每趟排序, 根据对应的增量  $t_i$ , 将待排序列分割成若干长度为  $m$  的子序列, 分别对各子表进行直接插入排序。仅增量因子为 1 时, 整个序列作为一个表来处理, 表长度即为整个序列的长度。

**最好时间复杂度:**  $O(N \log^2 N)$

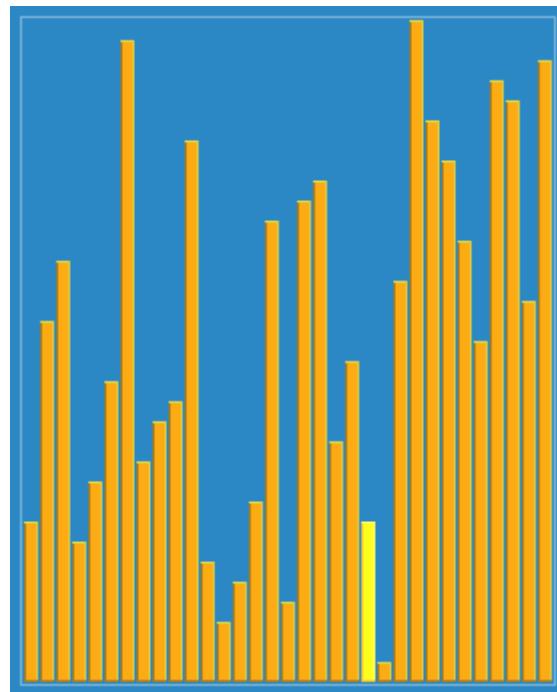
**最坏时间复杂度:**  $O(N \log^2 N)$

**平均时间复杂度:**  $O(N \log N)$

**空间复杂度:**  $O(1)$

**稳定排序:** 否

**原地排序:** 是



## 代码

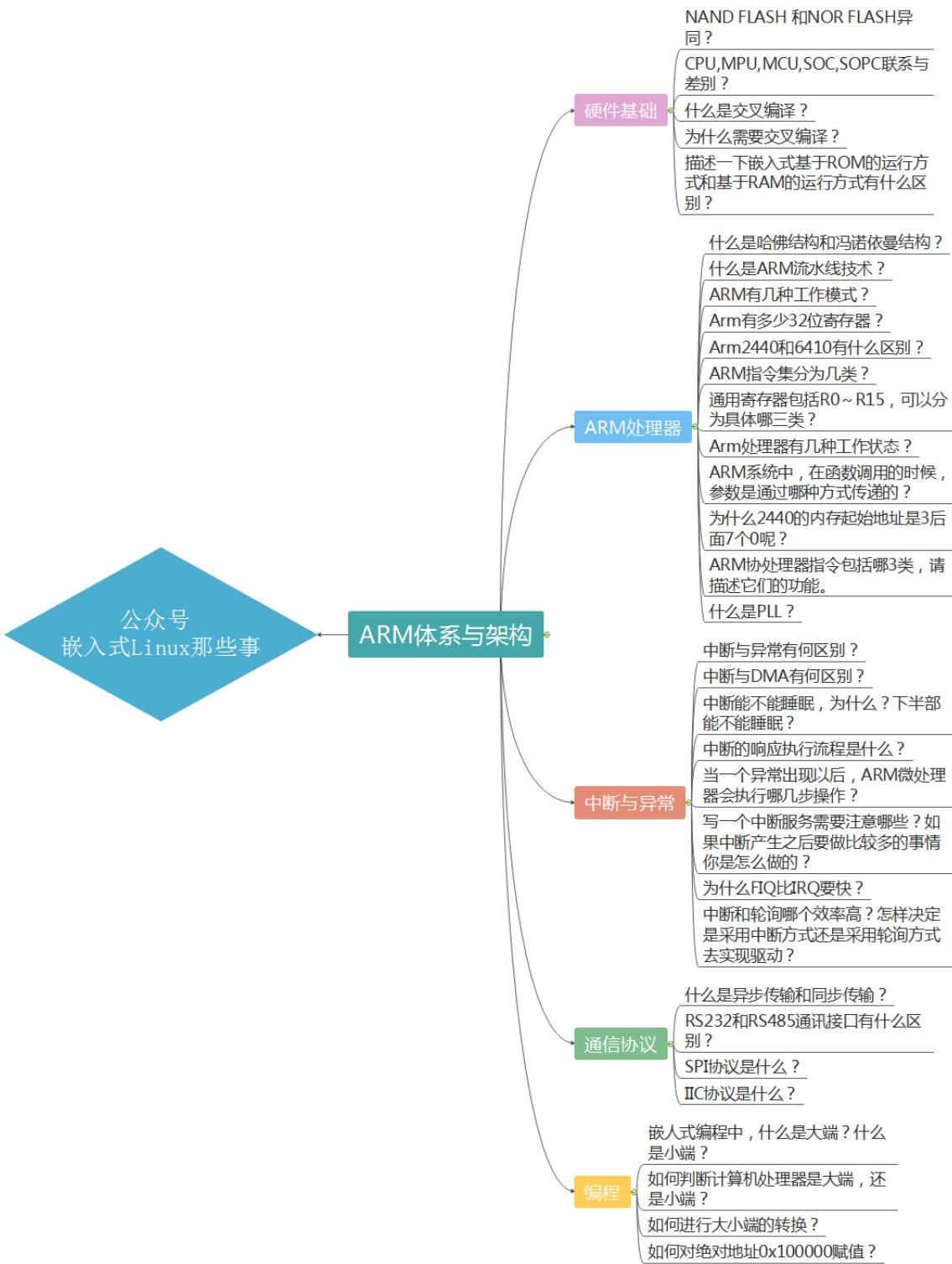
```

1 void shell_sort(int arr[], int len) {
2     int gap, i, j;
3     int temp;
4     for (gap = len >> 1; gap > 0; gap >>= 1)
5         for (i = gap; i < len; i++) {
6             temp = arr[i];
7             for (j = i - gap; j >= 0 && arr[j] > temp; j -= gap)
8                 arr[j + gap] = arr[j];
9             arr[j + gap] = temp;
10        }
11 }
```

## 改进

在扫描分组数据使不需要每次都交换, 找到最终的位置直接插入待处理元素即可

# ARM体系与架构



## 硬件基础

**NAND FLASH 和 NOR FLASH 异同?**

**不同点**

类别	NOR	NAND
读	快 像访问SRAM一样，可以随机访问任意地址的数据；如： unsigned short *pwAddr = (unsigned short *)0x02; unsigned short wVal; wVal = *pwAddr	快,有严格的时序要求，需要通过一个函数才能读取数据，先发送读命令->发送地址->判断nandflash是否就绪->读取一页数据读命令、发送地址、判断状态、读数据都是通过操作寄存器实现的,如数据寄存器NFDATA
写	慢，写之前需要擦除，因为写只能是1->0,擦除可以使0->1	快，写之前需要擦除，因为写只能是1->0,擦除可以使0->1
擦除	非常慢 (5S)	快 (3ms)
XIP	代码可以直接在NOR FLASH上运行	NO
可靠性	比较高，位反转的比例小于NAND FLASH的10%	比较低，位反转比较常见，必须有校验措施
接口	与RAM接口相同，地址和数据总线分开	I/O接口
可擦除次数	10000~100000	100000~1000000
容量	小，1MB~32MB	大，16MB~512MB
主要用途	常用于保存代码和关键数据	用于保存数据
价格	高	低

**注意：nandflash和norflash的0地址是不冲突的，norflash占用BANK地址，而nandflash不占用BANK地址，它的0地址是内部的。**

### 相同点

1	写之前都要先擦除，因为写操作只能使1->0，而擦除动作是为了把所有位都变1
2	擦除单元都以块为单位

## CPU, MPU, MCU, SOC, SOPC联系与差别?

1.CPU (Central Processing Unit) ,是一台计算机的**运算核心和控制核心**。CPU由运算器、控制器和寄存器及实现它们之间联系的数据、控制及状态的总线构成。差不多所有的CPU的运作原理可分为四个阶段：提取（Fetch）、解码（Decode）、执行（Execute）和写回（Writeback）。CPU从存储器或高速缓冲存储器中取出指令，放入指令寄存器，并对指令译码，并执行指令。所谓的计算机的可编程性主要是指对CPU的编程。

2.MPU (Micro Processor Unit), 叫**微处理器**（不是微控制器），通常代表一个**功能强大的CPU**（暂且理解为**增强版的CPU**吧），但不是为任何已有的特定计算目的而设计的芯片。这种芯片往往是个人计算机和高端工作站的核心CPU。最常见的微处理器是Motorola的68K系列和Intel的X86系列。

3.MCU(Micro Control Unit), 叫**微控制器**，是指随着大规模集成电路的出现及其发展，**将计算机的CPU、RAM、ROM、定时计数器和多种I/O接口集成在一片芯片上**，形成芯片级的芯片，**比如51, avr这些芯片，内部除了CPU外还有RAM,ROM,可以直接加简单的外围器件（电阻，电容）就可以运行代码了**，而MPU如x86, arm这些就不能直接放代码了，它只不过是增强版的CPU，所以得**添加RAM,ROM**。

MCU MPU 最主要的区别就睡能否直接运行代码。MCU有内部的RAM ROM，而MPU是增强版的CPU，需要添加外部RAM ROM才可以运行代码。

4.SOC (System on Chip) ，指的是片上系统，MCU只是**芯片级**的芯片，而**SOC是系统级的芯片**，它既MCU (51, avr) 那样有内置RAM,ROM同时又像MPU (arm) 那样强大的，不单单是放简单的代码，可以放系统级的代码，也就是说**可以运行操作系统**（将就认为是MCU集成化与MPU强处理力各优点二合一）。

5.SOPC (System On a Programmable Chip) 可编程片上系统 (FPGA就是其中一种)，上面4点的硬件配置是固化的，就是说51单片机就是51单片机，不能变为avr，而avr就是avr不是51单片机，他们的硬件是一次性掩膜成型的，能改的就是软件配置，说白点就是改代码，本来是跑流水灯的，改下代码，变成数码管，而SOPC则是**硬件配置，软件配置都可以修改**，软件配置跟上面一样，没什么好说的，至于硬件，是可以自己构建的也就是说这个芯片是自己构造出来的，**这颗芯片我们叫“白片”**，什么芯片都不是，把**硬件配置信息下载进去了，他就是相应的芯片了**，可以让他变成51，也可以是avr，甚至arm，同时SOPC是在SOC基础上来的，所以他也是系统级的芯片，所以记得当他变成arm时还得加外围ROM, RAM之类的，不然就是MPU了。

## 什么是交叉编译?

在一种计算机环境中运行的编译程序，能编译出在**另外一种环境下**运行的代码，我们就称这种编译器**支持交叉编译**。这个**编译过程**就叫交叉编译。简单地说，就是在**一个平台上生成另一个平台上的可执行代码**。

这里需要注意的是所谓平台，实际上包含两个概念：体系结构 (Architecture) 、操作系统 (OperatingSystem)。同一个体系结构可以运行不同的操作系统；同样，同一个操作系统也可以在不同的体系结构上运行。举例来说，我们常说的x86 Linux平台实际上是Intel x86体系结构和Linux for x86操作系统的统称；而x86 WinNT平台实际上是Intel x86体系结构和Windows NT for x86操作系统的简称。

## 为什么需要交叉编译?

有时是因为目的平台上不允许或不能够安装我们所需要的编译器，而我们又需要这个编译器的某些特征；有时是因为目的平台上的资源贫乏，无法运行我们所需要编译器；有时又是因为目的平台还没有建立，连操作系统都没有，根本谈不上运行什么编译器。

## 描述一下嵌入式基于ROM的运行方式和基于RAM的运行方式有什么区别？

### 基于RAM

1. 需要把硬盘和其他介质的代码先加载到ram中，加载过程中一般有重定位的操作。
2. 速度比基于ROM的快，可用RAM比基于ROM的少，因为所有的代码，数据都必须存放在RAM中。

### 基于ROM

1. 速度较基于RAM的慢，因为会有一个把变量，部分代码等从存储器（硬盘，flash）搬到RAM的过程。
2. 可用RAM资源比基于RAM的多；



## ARM处理器

### 什么是哈佛结构和冯诺依曼结构？

#### 定义

冯诺依曼结构采用指令和数据统一编址，使用同条总线传输，CPU读取指令和数据的操作无法重叠。

哈佛结构采用指令和数据独立编址，使用两条独立的总线传输，CPU读取指令和数据的操作可以重叠。

#### 利弊

冯诺依曼结构主要用于通用计算机领域，需要对存储器中的代码和数据频繁的进行修改，统一编址有利于节约资源。

哈佛结构主要用于嵌入式计算机，程序固化在硬件中，有较高的可靠性、运算速度和较大的吞吐。

### 什么是ARM流水线技术？

流水线技术通过多个功能部件并行工作来缩短程序执行时间，提高处理器核的效率和吞吐率，从而成为微处理器设计中最为重要的技术之一。ARM7处理器核使用了典型三级流水线的冯·诺伊曼结构，ARM9系列则采用了基于五级流水线的哈佛结构。通过增加流水线级数简化了流水线各级的逻辑，进一步提高了处理器的性能。

PC代表程序计数器，流水线使用三个阶段，因此指令分为三个阶段执行：1.取指（从存储器装载一条指令）；2.译码（识别将要被执行的指令）；3.执行（处理指令并将结果写回寄存器）。而R15 (PC) 总是指向“正在取指”的指令，而不是指向“正在执行”的指令或正在“译码”的指令。一般来说，人们习惯性约定将“正在执行的指令作为参考点”，称之为当前第一条指令，因此PC总是指向第三条指令。当ARM状态时，每条指令为4字节长，所以PC始终指向该指令地址加8字节的地址，即：**PC值=当前程序执行位置+8**；

ARM指令是三级流水线，取指，译指，执行，同时执行的，现在**PC指向的是正在取指的地址（下一条指令）**，那么cpu正在译指的指令地址是PC-4（假设在ARM状态下，一个指令占4个字节），**cpu正在执行的指令地址是PC-8**，也就是说PC所指向的地址和现在所执行的指令地址相差8。

当突然发生中断的时候，保存的是PC的地址（ $PC-8+4 = PC-4$  下一条指令的地址）

这样你就知道了，如果返回的时候返回PC，那么中间就有一个指令没有执行，所以用 `SUB pc lr-irq #4.`

Cycle	1	2	3	4	5	6	7	8	9
Operation									
<b>ADD</b>	F	D	E						
<b>SUB</b>		F	D	E					
<b>ORR</b>			F	D	E				
<b>AND</b>				F	D	E			
<b>ORR</b>					F	D	E		
<b>EOR</b>						F	D	E	

**F – 取指 D – 解码 E – 执行**

## ARM有几种工作模式？

### 1. 用户模式(USR)

用户模式是用户程序的工作模式，它运行在操作系统的用户态，它没有权限去操作其它硬件资源，只能执行处理自己的数据，**也不能切换到其它模式下**，要想访问硬件资源或切换到其它模式只能通过软中断或产生异常。

### 2. 系统模式(SYS)

系统模式是特权模式，不受用户模式的限制。用户模式和系统模式共用一套寄存器，操作系统在该模式下可以方便的访问用户模式的寄存器，而且操作系统的  
一些特权任务可以使用这个模式访问一些受控的资源。

说明：用户模式与系统模式两者使用相同的寄存器，都没有SPSR (Saved Program Statement Register，已保存程序状态寄存器)，但系统模式比用户模式有更高的权限，可以访问所有系统资源。

### 3. 一般中断模式(IRQ)

一般中断模式也叫普通中断模式，用于处理一般的中断请求，通常在硬件产生中断信号之后自动进入该模式，该模式为特权模式，可以自由访问系统硬件资源。

### 4. 快速中断模式(FIQ)

快速中断模式是相对一般中断模式而言的，它是用来处理对时间要求比较紧急的中断请求，主要用于高速数据传输及通道处理中。（快中断有许多（R8~R14）自己的专用寄存器，发生中断时，使用自己的寄存器就避免了保存和恢复某些寄存器。如果异常中断处理程序中使用它自己的物理寄存器之外的其他寄存器，异常中断处理程序必须保存和恢复这些寄存器）

### 5. 管理模式 (SVC)

管理模式是**CPU上电后默认模式**，因此，在该模式下主要用来做**系统的初始化**，软中断处理也在该模式下。当用户模式下的用户程序请求使用硬件资源时，通过软件中断进入该模式。

说明：系统复位或开机、软中断时进入到SVC模式下。

### 6. 终止模式(ABT):

中止模式用于支持虚拟内存或存储器保护，当用户程序访问**非法地址，没有权限读取的内存地址**时，会进入该模式，linux下编程时经常出现的segment fault通常都是在该模式下抛出返回的。

## 7. 未定义模式(UND):

未定义模式用于支持硬件协处理器的软件仿真，CPU在指令的译码阶段不能识别该指令操作时，会进入未定义模式。

1. 除了用户模式外，其它6种模式称为特权模式。所谓特权模式，即具有如下权利：

- a. MRS (把状态寄存器的内容放到通用寄存器)；
- b. MSR (把通用寄存器的内容放到状态寄存器中)。

由于状态寄存器中的内容不能够改变，因此，要先把内容复制到通用寄存器中，然后修改通用寄存器中的内容，再把通用寄存器中的内容复制给状态寄存器中，即可完成“修改状态寄存器”的任务。

2. 剩下的六种模式中除去系统模式外，统称为异常模式。

## Arm有多少32位寄存器？

ARM处理器共有37个寄存器。它包含31个通用寄存器和6个状态寄存器。

## Arm2440和6410有什么区别？

1. 主频不同。2440是400M的。6410是533/667M的；
2. 处理器版本不一样：2440是arm920T内核，6410是arm1176ZJF内核；
3. 6410在视频处理方面比2440要强很多。内部视频解码器，包括MPEG4等视频格式；
4. 6410支持WMV9、xvid、mpeg4、h264等格式的硬解码和编码；
5. 6410多了很多扩展接口比如：tv-out、CF卡和S-Video输出等；
6. spi、串口、sd接口也比那两个要丰富；
7. 6410采用的是DDR内存控制器；2440采用的是SDRam内存控制器；
8. 6410为双总线架构，一路用于内存总线、一路用于Flash总线；
9. 6410的启动方式更加灵活：主要包括SD、Nand Flash、Nor Flash和OneFlash等设备启动；
10. 6410的Nand Flash支持SLC和MLC两种架构，从而大大扩大存储空间；
11. 6410为双总线架构，一路用于内存总线、一路用于Flash总线；
12. 6410具备8路DMA通道，包括LCD、UART、Camera等专用DMA通道；
13. 6410还支持2D和3D的图形加速；

## ARM指令集分为几类？

2类，分别为Thumb指令集，ARM指令集。ARM指令长度为32位，Thumb指令长度为16位。这种特点使得ARM既能执行16位指令，又能执行32位指令，从而增强了ARM内核的功能。

## 通用寄存器包括R0 ~ R15，可以分为具体哪三类？

通用寄存器包括R0-R15，可以分为3类：

### 1. 未分组寄存器R0-R7

在所有运行模式下，未分组寄存器都指向同一个物理寄存器，他们未被系统用作特殊的用途。因此在中断或异常处理进行异常模式转换时，由于不同的处理器运行模式均使用相同的物理寄存器，所以可能造成寄存器中数据的破坏。

### 2. 分组寄存器R8-R14

对于分组寄存器，他们每次所访问的物理寄存器都与当前的处理器运行模式相关。

R13常用作存放堆栈指针，用户也可以使用其他寄存器存放堆栈指针，但在Thumb指令集下，某些指令强制要求使用R13存放堆栈指针。

R14称为链接寄存器（LR，Link Register），当执行子程序时，R14可得到R15（PC）的备份，执行完子程序后，又将R14的值复制回PC，即使用R14保存返回地址。

### 3. 程序计数器PC (R15)

寄存器R15用作程序计数器 (PC) , 在ARM状态下, 位[1:0]为0, 位[31:2]用于保存PC; 在Thumb状态下, 位[0]为0, 位[31:1]用于保存PC。

## Arm处理器有几种工作状态?

从编程的角度来看, ARM微处理器的工作状态一般ARM和Thumb有两种, 并可在两种状态之间切换。

1. ARM状态: 此时处理器执行32位的字对齐ARM指令, 绝大部分工作在此状态。
2. Thumb状态: 此时处理器执行16位的半字对齐的Thumb指令。

## ARM系统中, 在函数调用的时候, 参数是通过哪种方式传递的?

当参数小于等于4的时候是通过r0-r3寄存器来进行传递的, 当参数大于4的时候是通过压栈的方式进行传递。

## 为什么2440的内存起始地址是0x30000000?

S3C2440处理器有八个固定的内存块, 只有两个是可以作为ROM,SRAM和SDRAM等存储器bank。具体如下图所示。

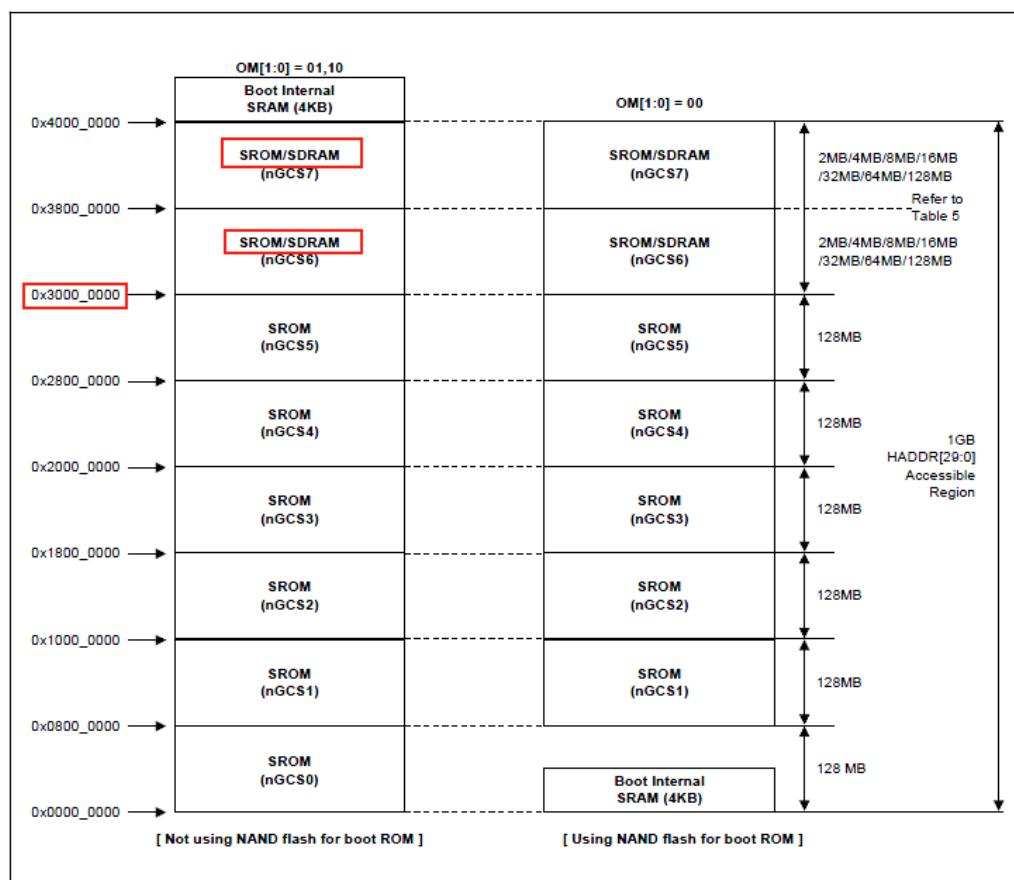


Figure 5-1. S3C2440A Memory Map after Reset

#### NOTE

SROM means ROM or SRAM type memory

[https://blog.csdn.net/qq\\_16933601](https://blog.csdn.net/qq_16933601)

## ARM协处理器指令包括哪3类, 请描述它们的功能。

ARM协处理器指令包括以下3类:

1. 用于ARM处理器初始化ARM协处理器的数据处理操作。
2. 用于ARM处理器的寄存器和ARM协处理器的寄存器间的数据传送操作。
3. 用于在ARM协处理器的寄存器和内存单元之间传送数据。

## 什么是PLL (锁相环) ?

简单来说，输入时钟的存在是作为“参考源”。锁相环不是为了单纯产生同频同相信号，而是一般集成进某种“频率综合电路”，产生一个不同频，但锁相的信号。

有点绕，打个比方：某参考晶振10Mhz，频率综合器A使用该参考源产生了900Mhz时钟，而频率综合器B产生了1Ghz时钟。虽然两路频率不同，但**由于使用的同一个参考源，他们俩仍然是同源信号**。相反，如果不同源，那么即便同频他们也不可能一致，因为世界上没有两个钟能做到完全一样，总有微弱的频差，导致相位飘移。在很多现实应用中有要求同源时钟的场合，所以，锁相环被广泛应用。锁相环的另外一项衍生应用是相干解调，可以自己查查相关资料。



## 中断与异常

### 中断与异常有何区别？

中断是指**外部硬件**产生的一个电信号从CPU的中断引脚进入，打断CPU的运行。

异常是指软件运行过程中发生了一些**必须作出处理**的事件，CPU自动产生一个陷入来打断CPU的运行。异常在处理的时候必须考虑与处理器的**时钟同步**，实际上异常也称为**同步中断**，在处理器执行到因编译错误而导致的错误指令时，或者在执行期间出现特殊错误，必须靠内核处理的时候，处理器就会产生一个异常。

### 中断与DMA有何区别？

**DMA**：是一种**无须CPU的参与**，就可以让**外设与系统内存**之间进行**双向数据传输**的硬件机制，使用DMA可以使系统CPU从实际的I/O数据传输过程中摆脱出来，从而大大提高系统的吞吐率。

**中断**：是指CPU在执行程序的过程中，出现了某些**突发事件**时，CPU必须**暂停执行**当前的程序，转去**处理突发事件**，处理完毕后CPU又**返回**源程序被中断的位置并继续执行。

所以中断和DMA的区别就是：**DMA不需CPU参与，而中断是需要CPU参与的。**

### 中断能不能睡眠，为什么？下半部能不能睡眠？

1. 中断处理的时候，不应该发生进程切换。因为在中断上下文中，**唯一能打断当前中断handler的只有更高优先级的中断，它不会被进程打断**。如果在中断上下文中休眠，则**没有办法唤醒它**，因为所有的**wake\_up\_xxx都是针对某个进程而言的**，而在中断上下文中，没有进程的概念，没有一个**task\_struct**（这点对于softirq和tasklet一样）。因此真的休眠了，比如调用了会导致阻塞的例程，内核几乎肯定会死。
2. schedule()在切换进程时，**保存当前的进程上下文**（CPU寄存器的值、进程的状态以及堆栈中的内容），以便以后恢复此进程运行。中断发生后，内核会**先保存当前被中断的进程上下文**（在调用中断处理程序后恢复）。
 

但在中断处理程序里，CPU寄存器的值肯定已经变化了（最重要的程序计数器PC、堆栈SP等）。如果此时因为睡眠或阻塞操作调用了schedule()，则保存的进程上下文就不是当前的进程上下文了。所以，不可以在中断处理程序中调用schedule()。
3. 2.4内核中schedule()函数本身在进来的时候**判断是否处于中断上下文**：

```

1 | if(unlikely(in_interrupt()))
2 | BUG();

```

因此，强行调用schedule()的结果就是内核BUG，但看2.6.18的内核schedule()的实现却没有这句，改掉了。

4. 中断handler会使用被中断的进程内核堆栈，但不会对它有任何影响，因为handler使用完后会完全清除它使用的那部分堆栈，恢复被中断前的原貌。
5. 处于中断上下文时候，**内核是不可抢占的**。因此，如果休眠，则内核一定挂起。

## 中断的响应执行流程是什么？

中断的响应流程：cpu接受中断->保存中断上下文跳转到中断处理历程->执行中断上半部->执行中断下半部->恢复中断上下文。

## 当一个异常出现以后，ARM微处理器会执行哪几步操作？

1. 将下一条指令的地址存入相应连接寄存器LR，以便程序在处理异常返回时能从正确的位置重新开始执行。若异常是从**ARM状态**进入，则LR寄存器中保存的是**下一条指令的地址**（当前PC + 4或PC + 8，与异常的类型有关）；若异常是从**Thumb状态**进入，则在LR寄存器中保存**当前PC的偏移量**，这样，异常处理程序就不需要确定异常是从何种状态进入的。例如：在软件中断异常SWI，指令MOV PC, R14\_svc总是返回到下一条指令，不管SWI是在ARM状态执行，还是在Thumb状态执行。
2. 将CPSR复制到相应的SPSR中。
3. 根据异常类型，强制设置CPSR的运行模式位。
4. 强制PC从相关的异常向量地址取下一条指令执行，从而跳转到相应的异常处理程序处。

## 写一个中断服务需要注意哪些？如果中断产生之后要做比较多的事情你是怎么做的？

1. 写一个中断服务程序要注意**快进快出**，在中断服务程序里面尽量**快速采集信息**，包括硬件信息，然后退出中断，要做其它事情可以使用**工作队列**或者**tasklet**方式。也就是中断上半部和下半部。
2. 中断服务程序中**不能有阻塞操作**。应为中断期间是完全占用CPU的（即不存在内核调度），中断被阻塞住，其他进程将无法操作。
3. 中断服务程序**注意返回值**，要用操作系统定义的宏做为返回值，而不是自己定义的。
4. 如果要做的事情较多，应将这些任务放在后半段(tasklet，等待队列等)处理。

## 为什么FIQ比IRQ要快？

1. ARM的FIQ模式提供了**更多的banked寄存器**，r8到r14还有SPSR，而IRQ模式就没有那么多，R8,R9,R10,R11,R12对应的banked的寄存器就没有，这就意味着在ARM的IRQ模式下，中断处理程序**自己要保存R8到R12这几个寄存器**，然后退出中断处理时程序要恢复这几个寄存器，而FIQ模式由于这几个寄存器都有banked寄存器，模式切换时CPU自动保存这些值到banked寄存器，退出FIQ模式时自动恢复，所以这个过程FIQ比IRQ快。不要小看这几个寄存器，ARM在编译的时候，如果你FIQ中断处理程序足够用这几个独立的寄存器来运作，它**就不会进行通用寄存器的压栈**，这样也省了一些时间。
2. FIQ比IRQ有**更高优先级**，如果FIQ和IRQ同时产生，那么FIQ先处理。
3. 在symbian系统里，当CPU处于FIQ模式处理FIQ中断的过程中，预取指令异常，未定义指令异常，软件中断全被禁止，所有的中断被屏蔽。所以FIQ就会很快执行，不会被其他异常或者中断打断，所以它又比IRQ快了。而IRQ不一样，当ARM处理IRQ模式处理IRQ中断时，如果来了一个FIQ中断请求，那正在执行的IRQ中断处理程序会被抢断，ARM切换到FIQ模式去执行这个FIQ，所以FIQ比IRQ快多了。

4. 另外FIQ的入口地址是0x1c,IRQ的入口地址是0x18。写过完整汇编系统的都比较明白这点的差别，18只能放一条指令，为了不与1C处的FIQ冲突，**这个地方只能跳转**，而FIQ不一样，1C以后没有任何中断向量表了，这样**可以直接在1C处放FIQ的中断处理程序**，由于跳转的范围限制，至少少了一条跳转指令。

## 中断和轮询哪个效率高？怎样决定是采用中断方式还是采用轮询方式去实现驱动？

中断是CPU处于**被动状态**下来接受设备的信号，而轮询是**CPU主动去查询该设备是否有请求**。

凡事都是两面性，所以，看效率不能简单的说那个效率高。如果是请求设备是一个**频繁请求cpu的设备**，或者有**大量数据请求**的网络设备，那么**轮询的效率是比中断高**。如果是一般设备，并且该设备请求cpu的**频率比较低**，则用**中断效率要高一些**。主要是看请求频率。



## 通信协议

### 什么是异步传输和同步传输？

**异步传输：**是一种典型的基于字节的输入输出，数据按每次一个字节进行传输，其传输速度低。

**同步传输：**需要外界的时钟信号进行通信，是把数据字节组合起来一起发送，这种组合称之为帧，其传输速度比异步传输快。

### RS232和RS485通讯接口有什么区别？

1. **传输方式不同。** RS232采取不平衡传输方式，即所谓**单端通讯**。而RS485则采用平衡传输，即**差分传输方式**。
2. **传输距离不同。** RS232适合本地设备之间的通信，传输距离一般**不超过20m**。而RS485的传输距离为**几十米到上千米**。
3. **设备数量。** RS232 只允许**一对一通信**，而RS485 接口在总线上是允许连接**多达128个收发器**。
4. **连接方式。** RS232，规定**用电平表示数据**，因此线路就是**单线路的**，用**两根线**才能达到**全双工**的目的；而RS485，使用**差分电平表示数据**，因此，必须用**两根线**才能达到传输数据的基本要求，要实现**全双工**，必需用**4根线**。

总结：从某种意义上，可以说，线路上存在的仅仅是电流，**RS232/RS485规定了这些电流在什么样的线路上流动和流动的样式**。

## SPI协议

### SPI的应用

SPI(Serial Peripheral Interface)协议是由摩托罗拉公司提出的通讯协议，即**串行外围设备接口**，是一种**高速全双工**的通信总线。SPI总线系统是一种同步串行外设接口，它可以使MCU与各种外围设备以串行方式进行通信以交换信息。SPI总线可直接与各个厂家生产的多种标准外围器件相连，包括FLASH、RAM、网络控制器、LCD显示驱动器、A/D转换器和MCU等。

## 接口

1. **MOSI (Master Output, Slave Input)**

主设备输出/从设备输入引脚。主机的数据从这条信号线输出，从机由这条信号线读入主机发送的数据，即这条线上数据的方向为主机到从机。

## 2. MISO(Master Input., Slave Output)

主设备输入/从设备输出引脚。主机从这条信号线读入数据，从机的数据由这条信号线输出到主机，即在这条线上数据的方向为从机到主机。

## 3. SCLK (Serial Clock)

时钟信号线，用于通讯数据同步。它由通讯主机产生，决定了通讯的速率，不同的设备支持的最高时钟频率不一样，如 STM32 的 SPI 时钟频率最大为 $f_{pclk}/2$ ，两个设备之间通讯时，通讯速率受限于低速设备。

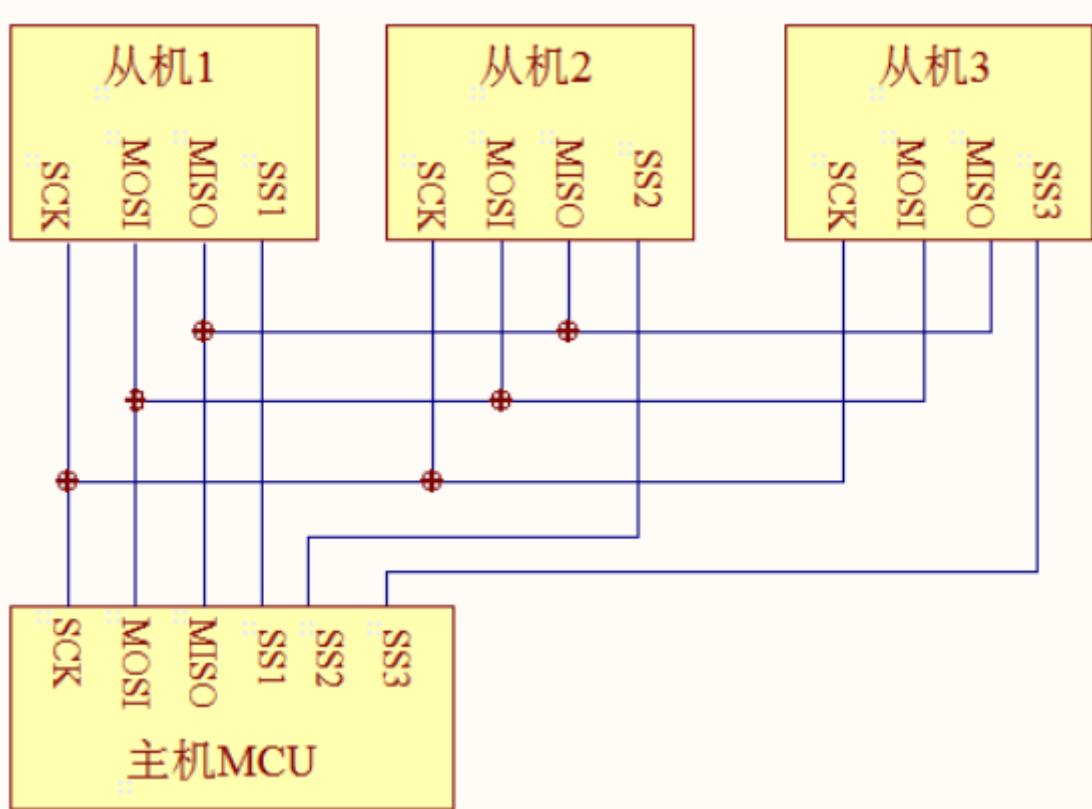
## 4. SS( Slave Select)

从设备选择信号线，常称为片选信号线，也称为 NSS、CS，以下用 NSS 表示。当有多个 SPI 从设备与 SPI 主机相连时，设备的其它信号线 SCK、MOSI 及 MISO 同时并联到相同的 SPI 总线上，即无论有多少个从设备，都共同只使用这 3 条总线；而每个从设备都有独立的这一条 NSS 信号线，本信号线独占主机的一个引脚，即有多少个从设备，就有多少条片选信号线。

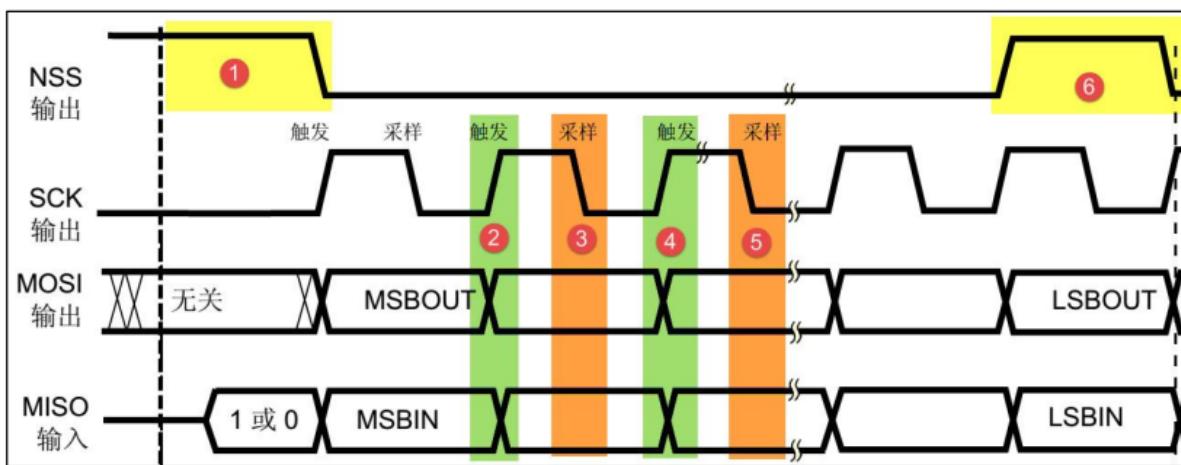
I2C 协议中通过设备地址来寻址、选中总线上的某个设备并与其进行通讯；而 SPI 协议中没有设备地址，它使用 NSS 信号线来寻址，当主机要选择从设备时，把该从设备的 NSS 信号线设置为低电平，该从设备即被选中，即片选有效，接着主机开始与被选中的从设备进行 SPI 通讯。所以 SPI 通讯以 NSS 线置低电平为开始信号，以 NSS 线被拉高作为结束信号。

## 协议层

SPI 通讯设备之间的常用连接方式见下图：



SPI 通讯的通讯时序，见下图：



### 1. 通讯的起始和停止信号

在图中的标号①处，NSS 信号线由高变低，是 SPI 通讯的起始信号。 NSS 是每个从机各自独占的信号线，当从机检在自己的 NSS 线检测到起始信号后，就知道自己被主机选中了，开始准备与主机通讯。在图中的标号⑥处， NSS 信号由低变高，是 SPI 通讯的停止信号，表示本次通讯结束，从机的选中状态被取消。

### 2. 数据有效性

SPI 使用 MOSI 及 MISO 信号线来传输数据，使用 SCK 信号线进行数据同步。 MOSI 及 MISO 数据线在 SCK 的每个时钟周期传输一位数据，且数据输入输出是同时进行的。数据传输时， MSB 先行（高位先行）或 LSB （低位先行）先行并没有作硬性规定，但要保证两个 SPI 通讯设备之间使用同样的协定，一般都会采用上图中的 MSB 先行（高位先行）模式。

观察图中的2345标号处， MOSI 及 MISO 的数据在 SCK 的上升沿期间变化输出，在 SCK 的下降沿时被采样。即在 SCK 的下降沿时刻， MOSI 及 MISO 的数据有效，高电平时表示数据“1”，为低电平时表示数据“0”。在其它时刻，数据无效， MOSI 及 MISO 为下一次表示数据做准备。

SPI 每次数据传输可以 8 位或 16 位为单位，每次传输的单位数不受限制。

### 3. CPOL (时钟极性) /CPHA (时钟相位) 及通讯模式

上面讲述的图中的时序只是 SPI 中的其中一种通讯模式， SPI 一共有四种通讯模式，它们的主要区别是：总线空闲时 SCK 的时钟状态以及数据采样时刻。为方便说明，在此引入“时钟极性CPOL”和“时钟相位CPHA”的概念。

时钟极性 CPOL 是指 SPI 通讯设备处于空闲状态时， SCK 信号线的电平信号(即 SPI 通讯开始前、 NSS 线为高电平时 SCK 的状态)。 CPOL=0 时， SCK 在空闲状态时为低电平， CPOL=1 时，则相反。

时钟相位 CPHA 是指数据的采样的时刻，当 CPHA=0 时， MOSI 或 MISO 数据线上的信号将会在 SCK 时钟线的“奇数边沿”被采样。当 CPHA=1 时，数据线在 SCK 的“偶数边沿”采样。

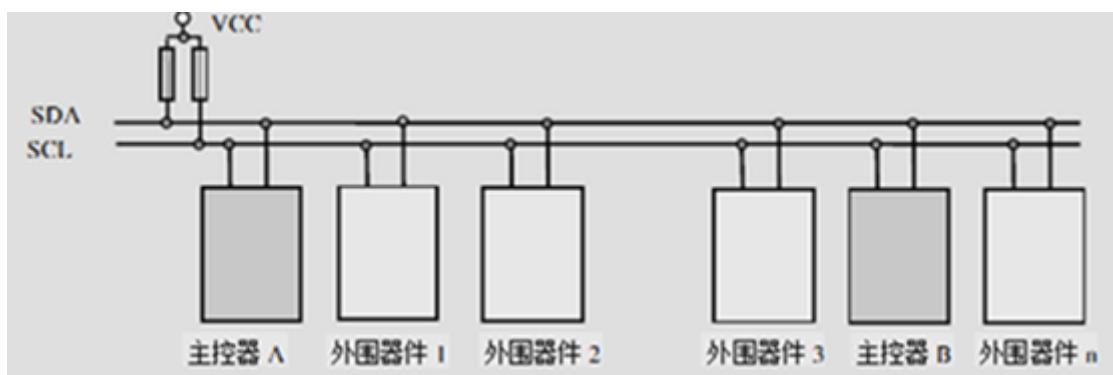
## IIC协议

### 简介

IIC协议是由数据线SDA和时钟SCL构成的串行总线，可发送和接收数据,是一个多主机的半双工通信方式

每个挂接在总线上的器件都有个唯一的地址。位速在标准模式下可达 100kbit/s,在快速模式下可达 400kbit/s，在高速模式下可待3.4Mbit/s。

I2C总线系统结构,如下所示:



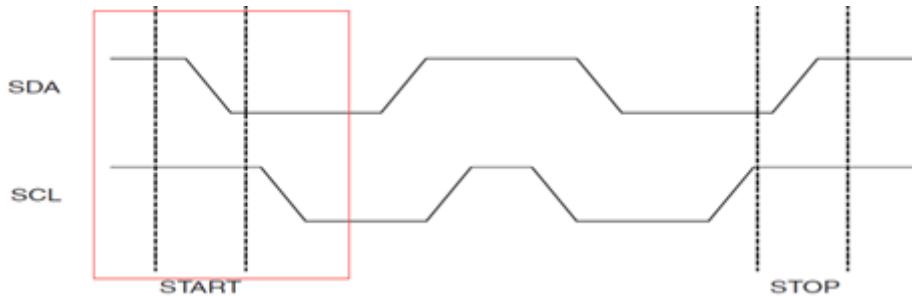
## I2C时序介绍

### 1. 空闲状态

当总线上的SDA和SCL两条信号线同时处于高电平,便是**空闲状态**,如上面的硬件图所示,当我们不传输数据时,SDA和SCL被上拉电阻拉高,即进入空闲状态

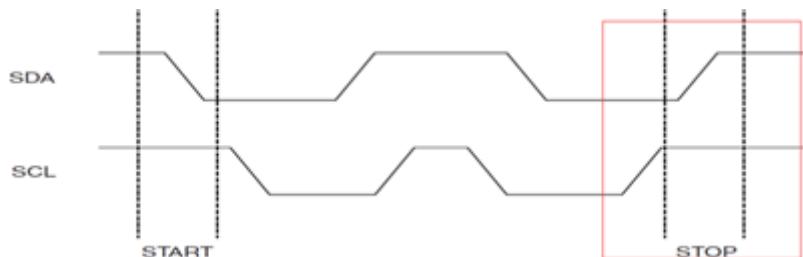
### 2. 起始信号

当SCL为高期间, SDA由高到低的跳变;便是总线的**启动信号**,只能由主机发起,且在空闲状态下才能启动该信号,如下图所示:



### 3. 停止信号

当SCL为高期间, SDA由低到高的跳变;便是总线的**停止信号**,表示数据已传输完成,如下图所示:



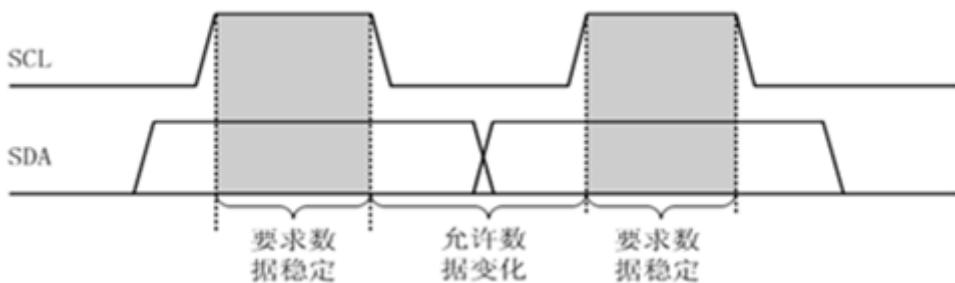
### 4. 传输数据格式

当发了起始信号后,就开始传输数据,传输的数据格式如下图所示:

当SCL为高电平时,便会获取SDA数据值,其中**SDA数据必须是稳定的**(若SDA不稳定就会变成起始/停止信号)。

当SCL为低电平时,便是**SDA的电平变化状态**。

若主从机在传输数据期间,需要完成其它功能(例如一个中断),可以主动**拉低SCL**,使I2C进入**等待状态**,直到处理结束再释放SCL,数据传输会继续

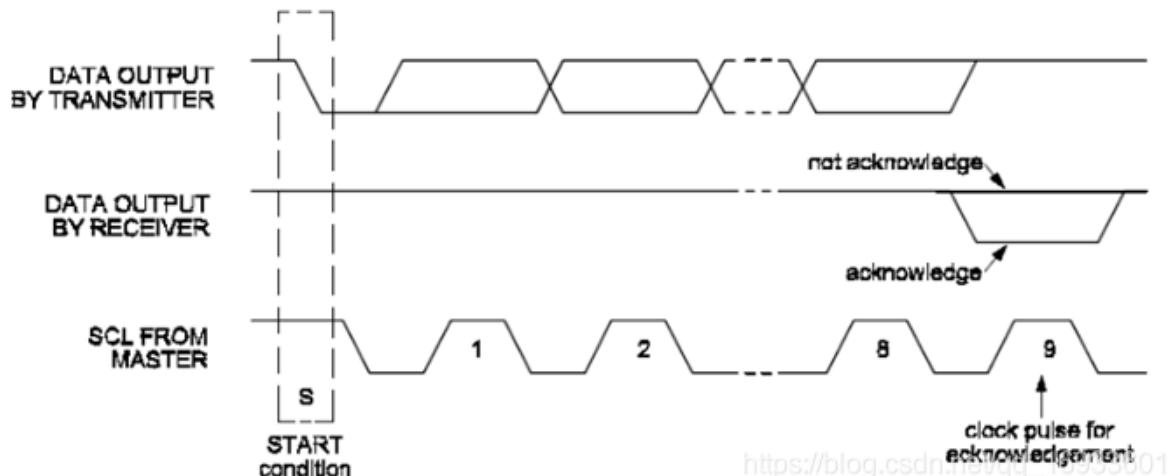


## 5. 应答信号ACK

I2C总线上的数据都是以8位数据(字节)进行的，当发送了8个数据后，发送方会在第9个时钟脉冲期间释放SDA数据，当接收方接收该字节成功，便会输出一个ACK应答信号，当SDA为高电平，表示为非应答信号NACK，当SDA为低电平，表示为有效应答信号ACK

PS:当主机为接收方时,收到最后一个字节后,主机可以不发送ACK,直接发送停止信号来结束传输。

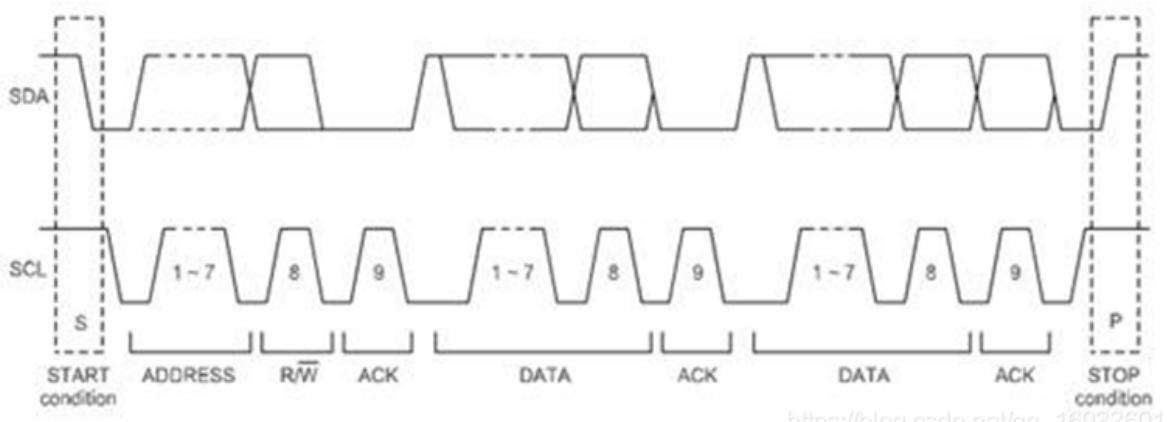
当从机为接收方时，没有发送ACK，则表示从机可能在忙其它事、或者不匹配地址信号和不支持多主机发送，主机可以发送停止信号，再次发送起始信号启动新的传输。



## 6. 完整的数据传输

如下图所示,发送起始信号后,便发送一个8位的设备地址,其中第8位是对设备的读写标志,后面紧跟着的就是数据了,直到发送停止信号终止。

PS:当我们第一次是读操作,然后想换成写操作时,可以再次发送一个起始信号,然后发送读的设备地址,不需要停止信号便能实现不同的地址转换。

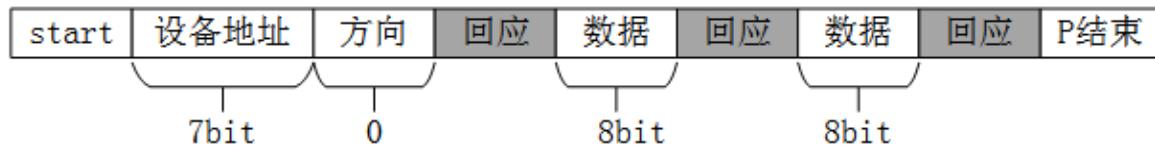


## IIC传输数据的格式

### 1.写操作

刚开始主芯片要发出一个**start信号**，然后发出一个(用来确定是往哪一个芯片写数据)，**方向**(读/写，0表示写，1表示读)。**回应**(用来确定这个设备是否存在)，然后就可以**传输数据**，传输数据之后，要有一个**回应信号** (确定数据是否接受完成)，然后再传输**下一个数据**。每传输一个数据，接受方都会有一个回应信号，**数据发送完**之后，主芯片就会发送一个**停止信号**。

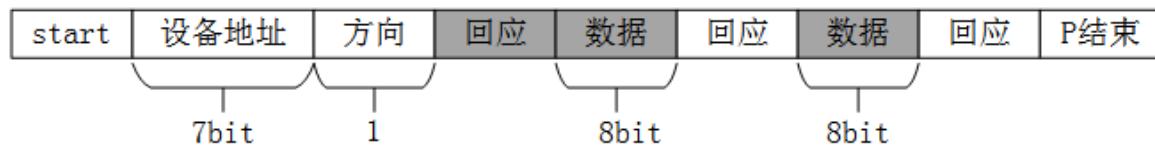
白色背景：主→从。灰色背景：从→主。



## 2. 读操作

刚开始主芯片要发出一个**start信号**，然后发出一个**设备地址**(用来确定是从哪一个芯片读取数据)，**方向**(读/写，0表示写，1表示读)。**回应**(用来确定这个设备是否存在)，然后就可以**传输数据**，传输数据之后，要有一个**回应信号** (确定数据是否接受完成)，然后在传输**下一个数据**。每传输一个数据，接受方都会有一个**回应信号**，数据发送完之后，主芯片就会发送一个**停止信号**。

白色背景：主→从。灰色背景：从→主



编程

嵌入式编程中，什么是大端？什么是小端？

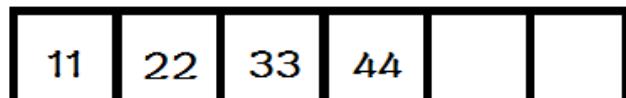
大端模式：低位字节存在高地址上，高位字节存在低地址上。

小端模式：高位字节存在高地址上，低位字节存在低地址上。

11 22 33 44

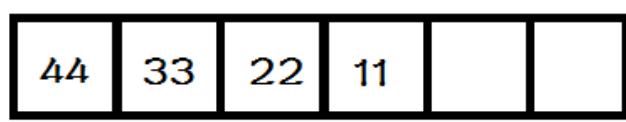
尾端 44

高尾端（大端）



低地址 -----&gt; 高地址

低尾端（小端）



低地址 -----&gt; 高地址

[https://www.csdn.net/qq\\_16933601](https://www.csdn.net/qq_16933601)

STM32属于小端模式，简单的说，比如u32 temp=0X12345678；假设temp地址在0X2000 0010。那么在内存里面存放就变成了：

1	地址		HEX	
2	0X2000 0010		78 56 43 12	

因为是16进制的，一个数为0.5字节，所以 12 代表一个字节 34 代表一个字节。

采用小端模式的CPU对操作数的存放方式是从低字节到高字节，而大端模式对操作数的存放方式是从高字节到低字节。例如，16位宽的数0x1234在小端模式CPU内存中的存放方式（假设从地址0x4000开始存放）见表1，而在大端模式CPU内存中的存放方式见表2。

表1 0x1234在小端CPU内存中的存放方式

内存地址	存放内容
0x4000	0x34
0x4001	0x12

表2 0x1234在大端CPU内存中的存放方式

内存地址	存放内容
0x4000	0x12
0x4001	0x34

32位宽的数0x12345678在小端模式CPU内存中的存放方式（假设从地址0x4000开始存放）见表3，而在大端模式CPU内存中的存放方式见表4。

表3 0x12345678在小端CPU内存中的存放方式

内存地址	存放内容
0x4000	0x78
0x4001	0x56
0x4002	0x34
0x4003	0x12

表4 0x12345678在大端CPU内存中的存放方式

内存地址	存放内容
0x4000	0x12
0x4001	0x34
0x4002	0x56
0x4003	0x78

以下程序为例：

```

1 #include <stdio.h>
2 struct mybitfields
3 {
4     unsigned short a:4;
5     unsigned short b:5;
6     unsigned short c:7;
7 }test;
8 int main()
9 {
10    int i;
11    test.a = 2;
12    test.b = 3;
13    test.c = 0;
14    i =*((short*)&test);
15    printf("%d\n",i);
16    return 0;
17 }
```

程序的输出结果为 50。

上例中，`sizeof ( test ) =2`，上例的声明方式是把一个 `short`（也就是一块16位内存）分成3部分，各部分的大小分别是4位、5位、7位，赋值语句 `i =*((short*)&test)` 就是把上面的16位内存转换成 `short` 类型进行解释。

变量a的二进制表示为0000000000000010，取其低四位是0010.变量b的二进制表示为0000000000000011，取其低五位是00011。变量c的二进制表示为0000000000000000，取其低七位是0000000。

**80x86机是小端（修改分区表时要注意）模式，单片机一般为大端模式。**小端一般是低位字节在高位字节的前面，也就是低位在内存地址低的一端，可以这样记（小端→低位→在前→与正常逻辑顺序相反），所以合成分后得到0000000000110010，即十进制的50。

下面给出另外一个例子

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 int main()
5 {
6     unsigned int uival_1 = 0x12345678;
7     unsigned int uival_2 = 0;
8     unsigned char aucVal[4] = {0x12,0x34,0x56,0x78};
9     unsigned short usval_1 = 0;
10    unsigned short usval_2 = 0;
11    memcpy(&uival_2,aucVal,sizeof(uival_2));
12    usval_1 = (unsigned short)uival_1;//在这里截断，都取得的是低位
13    usval_2 = (unsigned short)uival_2;//在这里截断
14    printf("usval_1:%x\n",usval_1);//在这里又转化回来
15    printf("usval_2:%x\n",usval_2);//在这里又转化回来
16    return 0;
17
18 }

```

小端模式是低地址存放低字节，高地址存放高字节，结构如下所示

```

1 78//低地址
2 56
3 34
4 12//高地址

```

在内存里面测试机是小端，地址由小到大。

```

1 val1:78563412
2 riva12:12345678

```

结果如下：

```

1 5678
2 3412

```

## 如何判断计算机处理器是大端，还是小端？

```

1 #include <stdio.h>
2 int checkCPU()
3 {
4     {
5         union w
6         {
7             int a;
8             char b;
9         }c;
10        c.a =1;
11        return(c.b == 1);
12    }
13 }
14 int main()
15 {
16     if(checkCPU())

```

```

17     printf("小端\n");
18 else
19     printf("大端\n");
20 return 0;
21 }
```

编者的处理器为Intel处理器，因为Intel处理器一般都是小端模式，所以此时程序的输出结果为：小端

上述代码中，如果处理器是大端，则返回0；如果处理器是小端，则返回1。联合体union的存放顺序是所有成员都从低地址开始存放，如果能够通过改代码知道CPU对内存是采用小端模式读写，还是采用大端模式读写，一定会令面试官刮目相看。

还可以通过指针地址来判断，由于在32位计算机系统中，short占两个字节，char占1个字节，所以可以采用如下做法实现该判断。

```

1 #include <stdio.h>
2 int checkCPU()
3 {
4     unsigned short usData = 0x1122;
5     unsigned char*pucData = (unsigned char*)&usData;
6     return (*pucData == 0x22);
7 }
8 int main()
9 {
10    if(checkCPU())
11        printf("小端\n");
12    else
13        printf("大端\n");
14    return 0;
15 }
```

程序输出的结果为：小端

## 如何进行大小端的转换？

```

1 int swapInt32(int intValue){
2
3     int temp = 0;
4
5     temp = ((intValue & 0x000000FF) <<24) |
6
7         ((intValue & 0x0000FF00) <<8) |
8
9         ((intValue & 0x00FF0000) >>8) |
10
11         ((intValue & 0xFF000000) >>24);
12     return temp;
13 }
14 /*short型：*/
15 unsigned short swapshort16(unsigned short shortvalue){
16
17
18     return ((shortvalue & 0x00FF ) <<8) | ((shortvalue & 0xFF00)>>8);
19
20 }
21 /*float型：*/
```

```

23 float swapFloat32(float floatValue){
24
25     typedef union SWAP_UNION{
26
27         float unionFloat;
28
29         int unionInt;
30
31     }SWAP_UNION;
32
33     SWAP_UNION swapUnion;
34     swapUnion.unionFloat = floatValue;
35     swapUnion.unionInt = swapInt32( swapUnion.unionInt);
36
37     return swapUnion.unionFloat;
38 }
39 /*double型换一种写法，用一下指针，不然移位移死了.....*/
40 void swapDouble64(unsigned char *pIn, unsigned char *pOut){
41
42     for( int i=0;i<8;i++)
43
44         pOut[7-i] = pIn[i];
45
46     }
47
48     int main()
49     {
50         int x = 0x12345678;
51         int y = swapInt32(x);
52         printf("%x\r\n",y);
53         return 0;
54     }

```

## 如何对绝对地址0x100000赋值？

```
1 | *(unsigned int*)0x100000 = 1234;
```

那么要是想让程序跳转到绝对地址是0x100000去执行，应该怎么做？

```
1 | *((void (*)())0x100000)();
```

首先要将0x100000强制转换成函数指针，即：

```
1 | (void (*)())0x100000
```

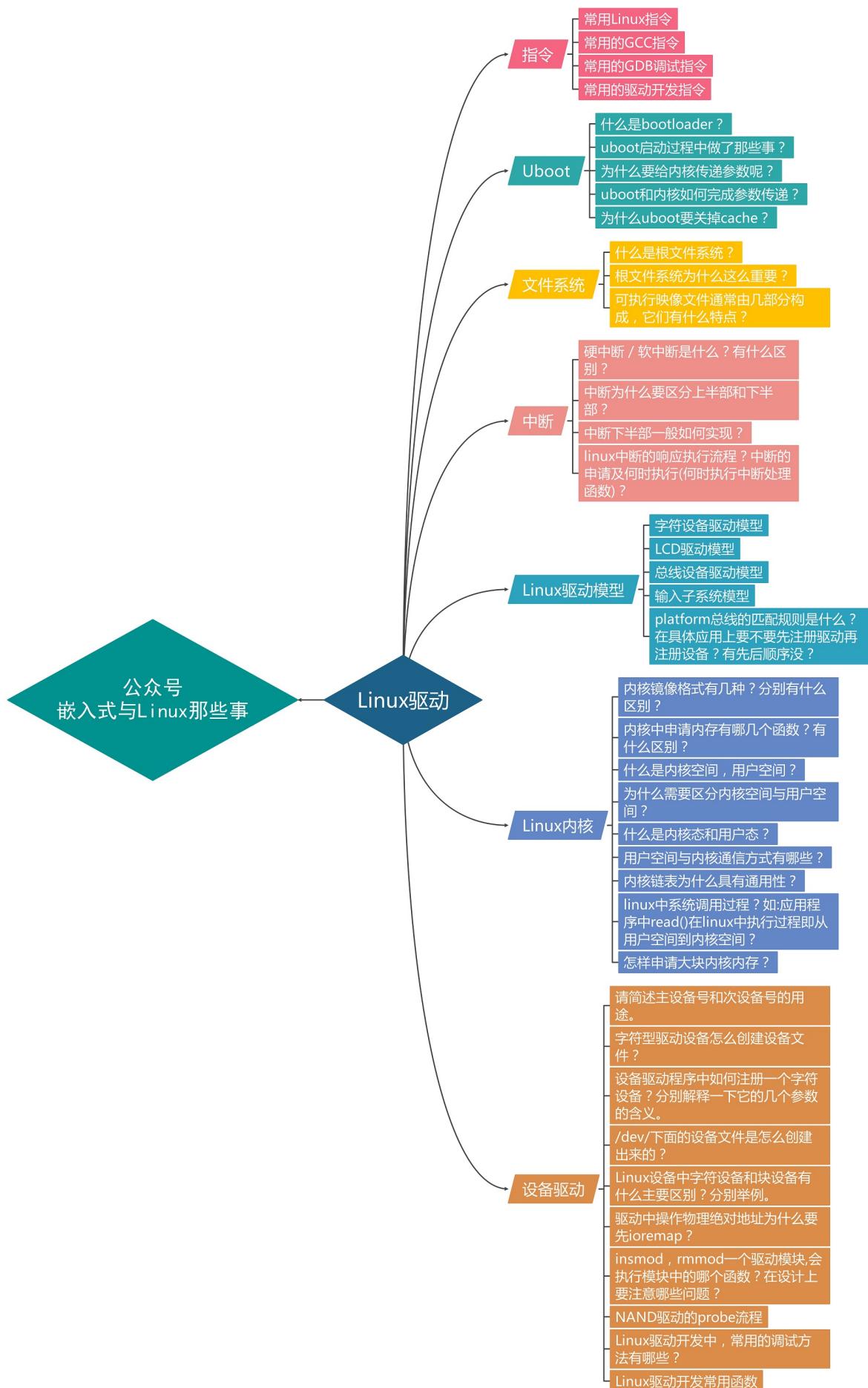
然后再调用它：

```
1 | *((void (*)())0x100000)();
```

用typedef可以看得更直观些：

```
1 | typedef void(*)() voidFuncPtr;
2 | *((voidFuncPtr)0x100000)();
```

# Linux驱动



# 指令

## 常用的Linux指令

**怎么查看当前进程？怎么执行退出？怎么查看当前路径？**

查看当前进程： ps

执行退出： exit

查看当前路径： pwd

**ls 命令执行什么功能？可以带哪些参数？**

### 功能

列出指定目录中的目录，以及文件

### 参数

- -a 显示所有文件及目录 (. 开头的隐藏文件也会列出)
- -l 除文件名称外，亦将文件型态、权限、拥有者、文件大小等资讯详细列出
- -r 将文件以相反次序显示(原定依英文字母次序)
- -t 将文件依建立时间之先后次序列出
- -A 同 -a，但不列出 "." (目前目录) 及 ".." (父目录)
- -F 在列出的文件名称后加一符号；例如可执行档则加 "\*", 目录则加 "/"
- -R 若目录下有文件，则以下之文件亦皆依序列出

**创建目录用什么命令？**

```
1 | mkdir runoob      #在工作目录下，建立一个名为 runoob 的子目录
2 | mkdir -p runoob2/test #在工作目录下的 runoob2 目录中，建立一个名为 test 的子目录。
若 runoob2 目录原本不存在，则建立一个。（注：本例若不加 -p 参数，且原本 runoob2 目录不存在，则产生错误。）
```

**创建文件用什么命令？**

### vi或vim

```
1 | vi file1.txt #直接创建并打开一个文件file1.txt
```

### touch

```
1 | touch file2.txt #创建新的空文件file2.txt
```

### echo

```
1 | echo "this is a new file" > file3.txt #创建文件file3.txt并将this is a new
file写入（说明：使用>指令覆盖文件原内容并重新输入内容，若文件不存在则创建文件。）
2 | echo "add contents" >>file3.txt #在已存在的文件补充写入新内容add contents（说
明：使用>>指令向文件追加内容，原内容将保存。）
```

### less、more、cat

三者都是将文件内容输出到标准输出，其中less和more可以分页显示，cat是显示全部。

三者可以根据已经存在的文件创建新的文件。假设已经存在文件1.txt。

```

1 | cat 1.txt > 2.txt
2 | less 1.txt > 3.txt
3 | more 1.txt > 4.txt

```

此时创建的文件内容都和1.txt中文件内容相同。

**cd**

```

1 | cd > file3.txt #创建新的空文件file3.txt
2 | cd >> file4.txt #创建新的空文件file4.txt

```

cd最主要的作用是切换目录，在cd后面跟>或>>再加上文件名就可以创建一个内容为空的文件。它和echo的区别之处在于echo可写文件内容，而cd并不能。

**复制文件用什么命令？**

```

1 | cp -r test/ newtest      #将当前目录 test/ 下的所有文件复制到新目录 newtest 下

```

**查看文件内容有哪些命令可以使用？**

```

1 | vi 文件名 #编辑方式查看，可修改
2 | cat 文件名 #显示全部文件内容
3 | more 文件名 #分页显示文件内容
4 | less 文件名 #与 more 相似，更好的是可以往前翻页
5 | tail 文件名 #仅查看尾部，还可以指定行数
6 | head 文件名 #仅查看头部，还可以指定行数

```

**怎么向屏幕输出带空格的字符串，比如“hello world”？**

```

1 | echo hello world

```

**移动文件用哪个命令？改名用哪个命令？**

```

1 | mv source_file(文件) dest_file(文件) #将源文件名 source_file 改为目标文件名 dest_file
2 | mv source_file(文件) dest_directory(目录) #将文件 source_file 移动到目标目录 dest_directory 中

```

**删除文件用哪个命令？如果需要连目录及目录下文件一块删除呢？删除空文件夹用什么命令？**

```

1 | rm -rf file/directory      #删除当前目录下的所有文件及目录，并且是直接删除，无需逐一确认
2 | rm -rf directory/         #删除目录 directory，不管该目录下是否有子目录或文件

```

**查找文件内容用哪个命令？**

```

1 | grep test *file #在当前目录中，查找后缀有 file 字样的文件中包含 test 字符串的文件，并打印出该字符串的行
2 | grep -r update /etc/acpi #查找指定目录/etc/acpi 及其子目录（如果存在子目录的话）下所有文件中包含字符串"update"的文件
3 | grep -v test *test*      #查找文件名中包含 test 的文件中不包含test 的行

```

## 查找文件用哪个命令？

```
1 | find . -name "*.c" #将当前目录及其子目录下所有文件后缀为 .c 的文件列出来
2 | find . -ctime -20 #将当前目录及其子目录下所有最近 20 天内更新过的文件列出
```

## cat命令

```
1 | cat -n myfile1 #把 myfile1 的文档内容加上行号后输入到屏幕
2 | cat -n myfile1 > myfile2 #把 myfile1 的文档内容加上行号后输入 myfile2 这个文档里
3 | cat -b myfile1 myfile2 >> myfile3 #把 myfile1 和 myfile2 的文档内容加上行号(空白行不加)之后将内容附加到 myfile3 文档里
4 | cat /dev/null > /etc/test.txt #清空 /etc/test.txt 文档内容
```

## 常用的GCC指令

### 预处理

```
1 | gcc -E test.c -o test.i #把预处理的结果导出到test.i文件
```

### 编译为汇编代码

```
1 | gcc -S test.i -o test.s #编译器将test.i翻译成汇编语言，并将结果存储在test.s文件中。
```

### 汇编

```
1 | gcc -c test.s -o test.o #将汇编代码编译为目标文件 (.o) 但不链接
```

### 链接

```
1 | gcc test.o -o test #将生成的目标文件test.o生成最终的可执行文件test
```

### 一步到位编译

```
1 | gcc test.c -o test #将源文件test.c编译链接为可执行文件test
```

### 多文件编译

```
1 | gcc test1.c test2.c -o test
```

### 警告处理

```
1 | gcc -w test.c -o test #忽略编译时的警告
2 | gcc -Wall test.c -o test #编译后显示所有警告
3 | gcc -Werror test.c -o test #在产生警告的地方停止编译
```

## 常用的GDB调试指令

```
1 | gcc -g test.c -o test #编译时生成debug有关的程序信息
2 | gdb test #启动调试
3 | help #查看命令帮助，具体命令查询在gdb中输入help + 命令，简写h
4 | run #重新开始运行文件 (run-text: 加载文本文件, run-bin: 加载二进制文件)，简写r
```

```

5  start #单步执行，运行程序，停在第一执行语句
6  list #查看原代码（list-n,从第n行开始查看代码。list+ 函数名：查看具体函数），简写l
7  set #设置变量的值
8  next #单步调试（逐过程，函数直接执行），简写n
9  step #单步调试（逐语句：跳入自定义函数内部执行），简写s
10 backtrace #查看函数的调用的栈帧和层级关系，简写bt
11 frame #切换函数的栈帧，简写f
12 info #查看函数内部局部变量的数值，简写i
13 finish #结束当前函数，返回到函数调用点
14 continue #继续运行，简写c
15 print #打印值及地址，简写p
16 quit #退出gdb，简写q
17 break+num #在第num行设置断点，简写b
18 info breakpoints #查看当前设置的所有断点
19 delete breakpoints num #删除第num个断点，简写d
20 display #追踪查看具体变量值
21 undisplay #取消追踪观察变量
22 watch #被设置观察点的变量发生修改时，打印显示
23 i watch #显示观察点
24 enable breakpoints #启用断点
25 disable breakpoints #禁用断点
26 x #查看内存x/20xw 显示20个单元，16进制，4字节每单元
27 run argv[1] argv[2] #调试时命令行传参
28 set follow-fork-mode child #Makefile项目管理：选择跟踪父子进程(fork())

```

## 常用的驱动开发指令

### 加载/卸载驱动

```

1 | insmod/modprobe #加载驱动
2 | rmmod   #卸载驱动

```

### Linux驱动如何查看驱动模块中打印信息？

```
1 | dmesg
```

### 如何查看内核中已有的字符设备的信息？

lsmod 和modprobe, lsmod可以查看模块的依赖关系，modprobe在加载模块时会加载其他依赖的模块。

### 如何查看正在使用的有哪些中断号？

```
1 | cat /proc/interrupt
```



微信搜一搜

嵌入式与Linux那些事

## uboot

### 什么是bootloader?

Linux系统要启动就必须需要一个 bootloader程序，也就说芯片上电以后先运行一段bootloader程序。这段 bootloader程序会先**初始化时钟，看门狗，中断，SDRAM，等外设，然后将 Linux内核从 flash (NAND, NOR FLASH, SD, MMC等) 拷贝到SDRAM中，最后启动Linux内核。**当然了， bootloader的实际工作要复杂的多，但是它最主要的工作就是启动 Linux内核。

bootloader和 Linux内核的关系就跟PC上的BIOS和 Windows的关系一样， bootloader就相当于 BIOS。总得来说， **Bootloader就是一小段程序，它在系统上电时开始执行，初始化硬件设备、准备好软件环境，最后调用操作系统内核。**

### uboot启动过程中做了那些事？

#### 第一阶段

初始化时钟，关闭看门狗，关中断，启动ICACHE，关闭DCACHE和TLB，关闭MMU，初始化SDRAM，初始化NAND FLASH，重定位。

#### 第二阶段

初始化一个串口，检测系统内存映射，将内核映象和根文件系统映象从 Flash上读到SDRAM空间中，为内核设置启动参数，调用内核。

#### [详细版解释](#)

### uboot和内核如何完成参数传递？

uboot启动后已经完成了基本的硬件初始化（如：内存、串口等），接下来，它的主要任务就是加载 Linux内核到开发板的内存，然后跳转到Linux内核所在的地址运行。

**PS:只要问到uboot，面试官必问uboot和内核的参数传递，所以一定要知道！**

具体是如何跳转呢？做法很简单，**直接修改PC寄存器的值为Linux内核所在的地址**，这样CPU就会从 Linux内核所在的地址去取指令，从而执行内核代码。

在前面我们已经知道，在跳转到内核以前， uboot需要做好以下三件事情：

#### (1) CPU寄存器的设置

R0=0

R1=机器类型ID；对于ARM结构的CPU，其机器类型ID可以参见 `linux/arch/arm/tools/mach-types`

R2=启动参数标记列表在RAM中起始基址

#### (2) CPU工作模式

必须禁止中断 (IRQs和FIQs)

CPU必须为SVC模式

#### (3) Cache和MMU的设置

MMU必须关闭

指令 Cache可以打开也可以关闭

数据 Cache必须关闭

其中上面第一步CPU寄存器的设置中，就是通过R0,R1,R2三个参数给内核传递参数的。 ([ATPCS规则可以参考](#))

## 为什么要给内核传递参数呢？

在此之前，uboot已经完成了硬件的初始化，可以说已经“适应了”这块开发板。然而，内核并不是对于所有的开发板都能完美适配的（如果适配了，可想而知这个内核有多庞大，又或者有新技术发明了，可以完美的适配各种开发板），此时，对于开发板的环境一无所知。所以，要想启动Linux内核，uboot必须给内核传递一些必要的信息来告诉内核当前所处的环境。

## 如何给内核传递参数？

uboot把机器ID通过R1传递给内核，Linux内核运行的时候，首先就从R1中读取机器ID来判断是否支持当前机器。这个机器ID实际上就是开发板CPU的ID，每个厂家生产出一款CPU的时候都会给它指定一个唯一的ID，大家可以到uboot源码的arch\arm\include\asm\mach-type.h文件中去查看。

```
#define MACH_TYPE_IKDP2801          300
#define MACH_TYPE_IQ31244             327
#define MACH_TYPE_BAST                331
#define MACH_TYPE_H1940               347
#define MACH_TYPE_ENP2611              356
#define MACH_TYPE_S3C2440             362
#define MACH_TYPE_GUMSTIX             373
#define MACH_TYPE_OMAP_H2              382
#define MACH_TYPE_E740                 384
```

R2存放的是块内存的基地址，这块内存中存放的是uboot给Linux内核的其他参数。这些参数有内存的起始地址、内存大小、Linux内核启动后挂载文件系统的方式等信息。很明显，参数有多个，不同的参数有不同的内容，为了让Linux内核能精确的解析出这些参数，双方在传递参数的时候要求参数在存放的时候需要按照双方规定的格式存放。

除了约定好参数存放的地址外，还要规定参数的结构。Linux2.4.x以后的内核都期望以标记列表

(tagged\_list) 的形式来传递启动参数。标记，就是一种数据结构；标记列表，就是挨着存放的多个标记。标记列表以标记 ATAG\_CORE 开始，以标记 ATAG\_NONE 结束。

标记的数据结构为tag，它由一个tag\_header结构和一个联合 (union) 组成。tag\_header结构表示标记的类型及长度，比如是表示内存还是表示命令行参数等。对于不同类型的标记使用不同的联合 (union)，比如表示内存时使用tag\_mem32，表示命令行时使用 tag\_cmdline。具体代码见 arch\arm\include\asm\setup.h。

```
struct tag {
    struct tag_header hdr;           // ATAG_CORE 标识tag参数的开始
    union {                         // ATAG_MEM 标识内存信息
        struct tag_core core;        // ATAG_CMDLINE 标识命令行参数
        struct tag_mem32 mem;         // ATAG_NONE 标识 tag 参数的结束
        struct tag_videotext videotext;
        struct tag_ramdisk ramdisk;
        struct tag_initrd initrd;
        struct tag_serialnr serialnr;
        struct tag_revision revision;
        struct tag_videofb videofb;
        struct tag_cmdline cmdline;   // 内存信息。如内存的基本地址，内存的大小。
    };                                // 描述命令参数信息，如 Linux 内核，挂载文件系统方式等
    /* * Acorn specific */
    /* */
    struct tag_acorn acorn;
    /* * DC21285 specific */
    /* */
    struct tag_memclk memclk;
} ? end {anonu} ? u;
} ? end tag ?;
```

[https://blog.csdn.net/qq\\_16933601](https://blog.csdn.net/qq_16933601)

从上面可以看出，struct\_tag结构体由structtag\_header+联合体union构成，结构体struct tag\_header用来描述每个tag的头部信息，如tag的类型，tag大小。联合体union用来描述每个传递给Linux内核的参数信息。

## 为什么uboot要关掉caches?

caches是cpu内部的一个2级缓存，它的作用是将常用的数据和指令放在cpu内部。caches是通过CP15管理的，刚上电的时候，cpu还不能管理caches。上电的时候指令cache可关闭，也可不关闭，但数据cache一定要关闭，否则可能导致刚开始的代码里面，去取数据的时候，从cache里面取，而这时候RAM中数据还没有caches过来，导致数据预取异常。



## 文件系统

### 什么是根文件系统？

根文件系统首先是一种文件系统，该文件系统不仅具有普通文件系统的存储数据文件的功能，但是相对于普通的文件系统，它的特殊之处在于，它是内核启动时所挂载（mount）的第一个文件系统，内核代码的映像文件保存在根文件系统中，系统引导启动程序会在根文件系统挂载之后从中把一些初始化脚本（如rcS,inittab）和服务加载到内存中去运行，里面包含了Linux系统能够运行所必需的应用程序、库等，比如可以给用户提供操作Linux的控制界面的shell程序、动态连接的程序运行时需要的glibc库等。

我们要明白文件系统和内核是完全独立的两个部分。在嵌入式中移植的内核下载到开发板上，是没有办法真正的启动Linux操作系统的，会出现无法加载文件系统的错误。

### 根文件系统为什么这么重要？

根文件系统之所以在前面加一个“根”，说明它是加载其它文件系统的“根”，那么如果没有这个根，其它的文件系统也就没有办法进行加载的。根文件系统包含系统启动时所必须的目录和关键性的文件，以及其他文件系统得以挂载（mount）所必要的文件。例如：

- init进程的应用程序必须运行在根文件系统上。
- 根文件系统提供了根目录“/”。
- linux挂载分区时所依赖的信息存放于根文件系统/etc/fstab这个文件中。
- shell命令程序必须运行在根文件系统上，譬如ls、cd等命令。

总之：一套linux体系，只有内核本身是不能工作的，必须要rootfs（上的etc目录下的配置文件、/bin /sbin等目录下的shell命令，还有/lib目录下的库文件等）相配合才能工作。

### 可执行映像文件通常由几部分构成，它们有什么特点？

可执行映像文件通常由以下几部分构成。

- 一个或多个代码段，代码段的属性为只读。
- 零个或多个包含初始化数据的数据段，数据段的属性为可读写。
- 零个或多个不包含初始化数据的数据段，数据段的属性为可读写。



## 中断

### 硬中断 / 软中断是什么？有什么区别？

#### 硬中断

1. 硬中断是由**硬件产生的**，比如，像磁盘，网卡，键盘，时钟等。**每个设备或设备集都有它自己的IRQ（中断请求）**。基于IRQ，CPU可以将相应的请求分发到对应的硬件驱动上（注：硬件驱动通常是内核中的一个子程序，而不是一个独立的进程）。
2. 处理中断的驱动是需要运行在CPU上的，因此，当中断产生的时候，CPU会中断当前正在运行的任务，来处理中断。在多核心的系统上，一个中断通常只能中断一颗CPU（也有一种特殊的情况，就是在大型主机上是有硬件通道的，它可以在没有主CPU的支持下，可以同时处理多个中断。）。
3. **硬中断可以直接中断CPU**。它会引起内核中相关的代码被触发。对于那些需要花费一些时间去处理的进程，中断代码本身也可以被其他的硬中断中断。
4. 对于时钟中断，内核调度代码会将当前正在运行的进程挂起，从而让其他的进程来运行。它的存在是为了让调度代码（或称为调度器）可以调度多任务。

#### 软中断

1. 软中断的处理非常像硬中断。然而，它们仅仅是由**当前正在运行的进程**所产生的。
2. 通常，软中断是一些对I/O的请求。这些请求会调用内核中可以调度I/O发生的程序。**对于某些设备，I/O请求需要被立即处理，而磁盘I/O请求通常可以排队并且可以稍后处理**。根据I/O模型的不同，进程或许会被挂起直到I/O完成，此时内核调度器就会选择另一个进程去运行。I/O可以在进程之间产生。并且调度过程通常和磁盘I/O的方式是相同。
3. 软中断仅与内核相联系。而内核主要负责对需要运行的任何其他的进程进行调度。一些内核允许设备驱动的一些部分存在于用户空间，并且当需要的时候内核也会调度这个进程去运行。
4. **软中断并不会直接中断CPU。也只有当前正在运行的代码（或进程）才会产生软中断**。这种中断是一种需要内核为正在运行的进程去做一些事情（通常为I/O）的请求。有一个特殊的软中断是Yield调用，它的作用是请求内核调度器去查看是否有一些其他的进程可以运行。

#### 区别

1. 软中断是执行中断指令产生的，而硬中断是由外设引发的。
2. 硬中断的中断号是由中断控制器提供的，软中断的中断号由指令直接指出，无需使用中断控制器。
3. 硬中断是可屏蔽的，软中断不可屏蔽。
4. 硬中断处理程序要确保它能快速地完成任务，这样程序执行时才不会等待较长时间，称为上半部。
5. 软中断处理硬中断未完成的工作，是一种推后执行的机制，属于下半部。

### 中断为什么要区分上半部和下半部？

Linux中断分为硬件中断和内部中断（异常），调用过程：**外部中断产生->发送中断信号到中断控制器->通知处理器产生中断的中断号，让其进一步处理。**

对于中断上半部和下半部的产生，为了中断处理过程中被新的中断打断，将中断处理一分为二，上半部登记新的中断，快速处理简单的任务，剩余复杂耗时的处理留给下半部处理，下半部处理过程中可以被中断，上半部处理时不可被中断。

## 中断下半部一般如何实现？

软中断、tasklet、工作队列。

[详细版解释](#)

## linux中断的响应执行流程？中断的申请及何时执行(何时执行中断处理函数)？

中断的响应流程：**cpu接受中断->保存中断上下文跳转到中断处理历程->执行中断上半部->执行中断下半部->恢复中断上下文。**

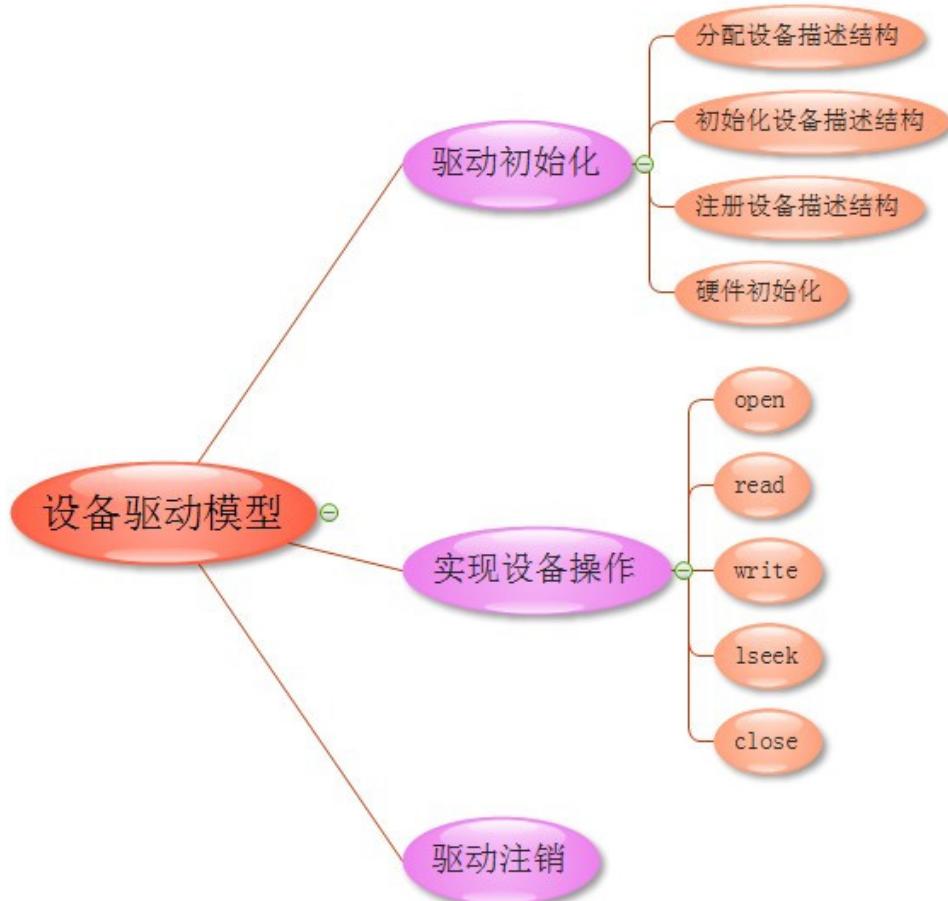
中断的申请request\_irq的正确位置：应该是在第一次打开、硬件被告知终端之前。



## Linux驱动模型

本节内容，重在理解。面试时，面试官很大可能会让你挑一个你熟悉的驱动讲讲，如何编写的？能回答出大概的驱动框架就可以。

### 字符设备驱动模型



驱动初始化中涉及到一个设备描述结构的概念。在任何一种驱动模型中，设备都会用内核中的一种结构来描述，这种结构成为设备描述结构。字符设备在内核中使用struct cdev这种结构来描述。

```

1 struct cdev
2 {
3     struct kobject kobj;
4     struct module *owner;
5     const struct file_operations *ops; //设备操作集
6     struct list_head list;
7     dev_t dev; //设备号
8     unsigned int count; //设备数
9 };

```

count表明该类型设备的数目，如有两个串口，则count的值为2。

dev是设备号，包含有主设备号和次设备号的信息。主设备号用于区分设备的类型，次设备号用于标记相同类型的设备的不同个体。如串口1和串口2使用同一驱动程序，则其主设备号相同，但次设备号不同。Linux内核中使用 dev\_t 类型来定义设备号， dev\_t 这种类型其实质为32位的 unsigned int，其中高12位为主设备号，低20位为次设备号。

1. 知道主设备号与次设备号，可通过 dev\_t dev = MKDEV(主设备号, 次设备号) 获得设备号；
2. 从设备号分解出主设备号：主设备号 = MAJOR(dev\_t dev)
3. 从设备号分解出次设备号：次设备号=MINOR(dev\_t dev)

主设备号是一个重要的资源，可以通过静态申请和动态分配为设备分配一个主设备号：

1. 静态申请：开发者自己选择一个数字作为主设备号，然后通过函数register\_chrdev\_region向内核申请使用。这种方法的缺点是如果申请使用的设备号已经被内核中的其它驱动使用了，则申请失败。
2. 动态分配：使用 alloc\_chrdev\_region 由内核分配一个可用的主设备号。因为内核知道哪些号已经被使用了，所以不会导致分配到已经被使用的号。

既然设备号是一种资源，则设备驱动在退出后都应该释放该资源。使用unregister\_chrdev\_region函数释放这些设备号。

ops是操作函数集。 file\_operations 是一个很重要的结构，该结构的成员基本都是函数指针，并且是一些文件操作的函数的指针。

```

1 struct file_operations {
2     struct module *owner;
3     loff_t(*lseek) (struct file *, loff_t, int);
4     ssize_t(*read) (struct file *, char __user *, size_t, loff_t *);
5     ssize_t(*aio_read) (struct kiocb *, char __user *, size_t, loff_t *);
6     ssize_t(*write) (struct file *, const char __user *, size_t, loff_t *);
7     ssize_t(*aio_write) (struct kiocb *, const char __user *, size_t, loff_t *);
8     int (*readdir) (struct file *, void *, filldir_t);
9     unsigned int (*poll) (struct file *, struct poll_table_struct *);
10    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
11    int (*mmap) (struct file *, struct vm_area_struct *);
12    int (*open) (struct inode *, struct file *);
13    int (*flush) (struct file *);
14    int (*release) (struct inode *, struct file *);
15    int (*fsync) (struct file *, struct dentry *, int datasync);
16    int (*aio_fsync) (struct kiocb *, int datasync);
17    int (*fasync) (int, struct file *, int);
18    int (*lock) (struct file *, int, struct file_lock *);

```

```

19 ssize_t(*readv) (struct file *, const struct iovec *, unsigned long,
20 loff_t *);
21 ssize_t(*writev) (struct file *, const struct iovec *, unsigned long,
loff_t *);
22 ssize_t(*sendfile) (struct file *, loff_t *, size_t, read_actor_t, void
__user *);
23 ssize_t(*sendpage) (struct file *, struct page *, int, size_t, loff_t *,
int);
24 unsigned long (*get_unmapped_area) (struct file *, unsigned long,unsigned
long, unsigned long,unsigned long);
25 };

```

`struct file_operations` 是一个函数指针的集合，定义能在设备上进行的操作。结构中的函数指针指向驱动中的函数，这些函数实现一个针对设备的操作，对于不支持的操作则设置函数指针为 NULL。例如：

```

1 struct file_operations dev_fops = {
2     .llseek = NULL,
3     .read = dev_read,
4     .write = dev_write,
5     .ioctl = dev_ioctl,
6     .open = dev_open,
7     .release = dev_release,
8 };

```

该结构体表示应用程序能够对设备文件使用函数 `read()`, `write()`, 等，但不能使用函数 `llseek()`。当执行到 `read()` 函数时，内核根据该结构体转移到驱动程序中的 `dev_read` 函数去执行。

驱动初始化有四大步骤：

### 1. 分配

`cdev` 变量的定义可以采用静态和动态两种办法：

静态分配：`struct cdev mdev;`

动态分配：`struct cdev *pdev = cdev_alloc();`

### 2. 初始化

`struct cdev` 的初始化使用 `cdev_init` 函数来完成。

原型：`cdev_init(struct cdev *cdev, const struct file_operations *fops)`

参数：

`cdev`: 待初始化的 `cdev` 结构

`fops`: 设备对应的操作函数集

### 3. 注册

字符设备的注册使用 `cdev_add` 函数来完成。

原型：`cdev_add(struct cdev *p, dev_t dev, unsigned count)`

参数：

`p`: 待添加到内核的字符设备结构

`dev`: 设备号

`count`: 该类设备的设备个数

### 3. 硬件初始化

根据相应硬件的数据手册完成初始化。

## 实现设备操作

由 `struct file_operations` 可以看出，要实现的操作并不少，这里只介绍一些重要的操作。

```

1 int (*open)(struct inode *, struct file *) //打开设备，响应open系统调用
2 int (*release)(struct inode *, struct file *); //关闭设备，响应close系统调用
3 loff_t (*lseek)(struct file *, loff_t, int) //重定位读写指针，响应lseek系统调用
4 ssize_t (*read)(struct file *, char __user *, size_t, loff_t *) //从设备读取数据，响应read系统调用
5 ssize_t (*write)(struct file *, const char __user *, size_t, loff_t *) //向设备写入数据，响应write系统调用

```

以上几个函数涉及到了 `struct inode` 和 `struct file` 这两种结构体。

在Linux系统中，每一个打开的文件，在内核中都会关联一个 `struct file` 结构体，它由内核在打开文件时创建，在文件关闭后释放。该结构体的重要成员有：

```

1 loff_t f_pos /*文件读写指针*/
2 struct file_operations *f_op /*该文件所对应的操作*/

```

每一个存在于文件系统里面的文件都会关联一个 `inode` 结构，该结构主要用来记录文件物理上的信息。因此，它和代表打开文件的 `file` 结构是不同的。一个文件没有被打开时不会关联 `file` 结构，但是却会关联一个 `inode` 结构。该结构体重要的成员有：

```
1 dev_t i_rdev /*设备号*/
```

一个设备支持的函数操作又称为设备方法。

`open` 设备方法是驱动程序用来为以后的操作完成初始化准备工作的。在大部分驱动程序中，`open` 完成如下工作：标明次设备号，启动设备。

`release` 设备方法的作用与 `open` 相反，这个设备方法有时也称为 `close`，它完成的工作是关闭设备。

`read` 设备方法通常完成两件事情：从设备中读取数据（属于硬件访问类操作），将读取到的数据返回给应用程序。

```
1 ssize_t (*read)(struct file *filp, char __user *buff, size_t count, loff_t
*offp)111
```

`filp`：与字符设备文件关联的 `file` 结构指针，由内核创建。

`buff`：从设备读取到的数据，需要保存到的位置。由 `read` 系统调用提供该参数。

`count`：请求传输的数据量，由 `read` 系统调用提供该参数。

`offp`：文件的读写位置，由内核从 `file` 结构中取出后，传递进来。

要注意的是，`buff` 参数是来源于用户空间的指针，这类指针都不能被内核代码直接引用，必须使用专门的函数：

```

1 int copy_to_user(void __user *to, const void *from, int n)
2 int copy_from_user(void *to, const void __user *from, int n)121212

```

其中 `copy_to_user()` 用于将内核数据传送给用户空间； `copy_from_user()` 用于将用户空间的数据传送给内核空间。

`write`设备方法通常完成两件事情：从应用程序提供的地址中取出数据，将数据写入设备(属于硬件访问类操作)

函数原型：`ssize_t (*write)(struct file *, const char __user *, size_t, loff_t *)`

## 驱动注销

驱动注销：当我们从内核中卸载驱动程序的时候，需要使用 `cdev_del` 函数来完成字符设备的注销。

一个驱动程序范例：

```

1 #include <linux/module.h>
2 #include <linux/types.h>
3 #include <linux/fs.h>
4 #include <linux/errno.h>
5 #include <linux/init.h>
6 #include <linux/cdev.h>
7 #include <asm/uaccess.h>
8 #include <linux/slab.h>
9
10 int dev1_registers[5];
11 int dev2_registers[5];
12
13 struct cdev cdev;
14 dev_t devno;
15
16 /*文件打开函数*/
17 int mem_open(struct inode *inode, struct file *filp)
18 {
19
20     /*获取次设备号*/
21     int num = MINOR(inode->i_rdev);
22
23     if (num==0)
24         filp->private_data = dev1_registers;
25     else if(num == 1)
26         filp->private_data = dev2_registers;
27     else
28         return -ENODEV; //无效的次设备号
29
30     return 0;
31 }
32
33 /*文件释放函数*/
34 int mem_release(struct inode *inode, struct file *filp)
35 {
36     return 0;
37 }
38
39 /*读函数*/
40 static ssize_t mem_read(struct file *filp, char __user *buf, size_t size,
41 loff_t *ppos)
42 {
43     unsigned long p = *ppos;
44     unsigned int count = size;

```

```

44     int ret = 0;
45     int *register_addr = filp->private_data; /*获取设备的寄存器地址*/
46
47     /*判断读位置是否有效*/
48     if (p >= 5*sizeof(int))
49         return 0;
50     if (count > 5*sizeof(int) - p)
51         count = 5*sizeof(int) - p;
52
53     /*读数据到用户空间*/
54     if (copy_to_user(buf, register_addr+p, count))
55     {
56         ret = -EFAULT;
57     }
58     else
59     {
60         *ppos += count;
61         ret = count;
62     }
63
64     return ret;
65 }
66
67 /*写函数*/
68 static ssize_t mem_write(struct file *filp, const char __user *buf, size_t
size, loff_t *ppos)
69 {
70     unsigned long p = *ppos;
71     unsigned int count = size;
72     int ret = 0;
73     int *register_addr = filp->private_data; /*获取设备的寄存器地址*/
74
75     /*分析和获取有效的写长度*/
76     if (p >= 5*sizeof(int))
77         return 0;
78     if (count > 5*sizeof(int) - p)
79         count = 5*sizeof(int) - p;
80
81     /*从用户空间写入数据*/
82     if (copy_from_user(register_addr + p, buf, count))
83         ret = -EFAULT;
84     else
85     {
86         *ppos += count;
87         ret = count;
88     }
89
90     return ret;
91 }
92
93 /* seek文件定位函数 */
94 static loff_t mem_llseek(struct file *filp, loff_t offset, int whence)
95 {
96     loff_t newpos;
97
98     switch(whence) {
99         case SEEK_SET:
100             newpos = offset;

```

```

101     break;
102
103     case SEEK_CUR:
104         newpos = filp->f_pos + offset;
105         break;
106
107     case SEEK_END:
108         newpos = 5*sizeof(int)-1 + offset;
109         break;
110
111     default:
112         return -EINVAL;
113     }
114     if ((newpos<0) || (newpos>5*sizeof(int)))
115         return -EINVAL;
116
117     filp->f_pos = newpos;
118     return newpos;
119
120 }
121
122 /*文件操作结构体*/
123 static const struct file_operations mem_fops =
124 {
125     .llseek = mem_llseek,
126     .read = mem_read,
127     .write = mem_write,
128     .open = mem_open,
129     .release = mem_release,
130 };
131
132 /*设备驱动模块加载函数*/
133 static int memdev_init(void)
134 {
135     /*初始化cdev结构*/
136     cdev_init(&cdev, &mem_fops);
137
138     /* 注册字符设备 */
139     alloc_chrdev_region(&devno, 0, 2, "memdev");
140     cdev_add(&cdev, devno, 2);
141 }
142
143 /*模块卸载函数*/
144 static void memdev_exit(void)
145 {
146     cdev_del(&cdev);    /*注销设备*/
147     unregister_chrdev_region(devno, 2); /*释放设备号*/
148 }
149
150 MODULE_LICENSE("GPL");
151
152 module_init(memdev_init);
153 module_exit(memdev_exit);

```

## LCD驱动模型

写个LCD驱动入口函数,需要以下4步:

1. 分配一个fb\_info结构体: `framebuffer_alloc()`;
2. 设置 `fb_info`
3. 设置硬件相关的操作
4. 使能LCD,并注册 `fb_info`: `register_framebuffer()`

需要用到的函数:

```

1 void *dma_alloc_writecombine(struct device *dev, size_t size, dma_addr_t
2 *handle, gfp_t gfp); //分配DMA缓存区给显存
3 //返回值为:申请到的DMA缓冲区的虚拟地址,若为NULL,表示分配失败,则需要使用
4 dma_free_writecombine()释放内存,避免内存泄漏//参数如下:
5
6 /*dev:指针,这里填0,表示这个申请的缓冲区里没有内容
7
8 /*handle:申请到的物理起始地址
9
10 //gfp:分配出来的内存参数,标志定义在<linux/gfp.h>,常用标志如下:
11 //GFP_ATOMIC    用来从中断处理和进程上下文之外的其他代码中分配内存. 从不睡眠.
12 //GFP_KERNEL    内核内存的正常分配. 可能睡眠.
13 //GFP_USER      用来为用户空间页来分配内存; 它可能睡眠.

```

分配一段DMA缓存区,分配出来的内存会禁止cache缓存(因为DMA传输不需要CPU)。它和 `dma_alloc_coherent()` 函数相似,不过 `dma_alloc_coherent()` 函数是分配出来的内存会禁止 cache缓存以及禁止写入缓冲区。

```

1 dma_free_writecombine(dev,size,cpu_addr,handle); //释放缓存,cpu_addr:虚拟地址,
  handle:物理地址

```

释放DMA缓冲区, dev和size参数和上面的一样。

```

1 struct fb_info *framebuffer_alloc(size_t size, struct device *dev); //申请一个
  fb_info结构体,size:额外的内存,*dev:指针, 这里填0,表示这个申请的结构体里没有内容
2 int register_framebuffer(struct fb_info *fb_info); //向内核中注册fb_info结构体,
  若内存不够,注册失败会返回负数
3 int unregister_framebuffer(struct fb_info *fb_info); //注销内核中fb_info结构体

```

需要用到的结构体:

`fb_info` 结构体如下:

```

1 struct fb_info {
2     ...
3     struct fb_var_screeninfo var;           //可变的参数
4     struct fb_fix_screeninfo fix;           //固定的参数
5     ...
6     struct fb_ops *fbops;                  //操作函数
7     ...
8     char __iomem *screen_base;             //显存虚拟起始地址
9     unsigned long screen_size;            //显存虚拟地址长度
10    void *pseudo_palette;
11 //假的16色调色板,里面存放了16色的数据,可以通过8bpp数据来找到调色板里面的16色颜色索引值,模拟出16色颜色来,节省内存,不需要的话就指向一个不用的数组即可
12    ...
13 };

```

其中操作函数 `fb_info-> fbops` 结构体写法如下:

```

1 static struct fb_ops s3c_lcdfb_ops = {
2     .owner          = THIS_MODULE,
3     .fb_setcolreg   = my_lcdfb_setcolreg, //设置调色板fb_info->
4     pseudo_palette,自己构造该函数
5
6     .fb_fillrect    = cfb_fillrect,        //填充矩形,用/drivers/video/
7     cfbfillrect.c里的函数即可
8
9     .fb_copyarea    = cfb_copyarea,      //复制数据,
10    用/drivers/video/cfbcopyarea.c里的函数即可
11
12     .fb_imageblit   = cfb_imageblit,    //绘画图形,
13     用/drivers/video/imageblit.c里的函数即可
14 };

```

固定的参数 `fb_info-> fix` 结构体如下:

```

1 struct fb_fix_screeninfo {
2     char id[16];                      //id名字
3     unsigned long smem_start;          //framebuffer物理起始地址
4
5     __u32 smem_len;                   //framebuffer长度,字节为单位
6     __u32 type;                      //lcd类型,默认值0即可
7     __u32 type_aux;                 //附加类型,为0
8     __u32 visual;                   //画面设置,常用参数如下
9 // FB_VISUAL_MONO1                0 单色,0:白色,1:黑色
10 // FB_VISUAL_MONO10               1 单色,1:白色,0:黑色
11 // FB_VISUAL_TRUECOLOR           2 真彩(TFT:真彩)
12 // FB_VISUAL_PSEUDOCOLOR         3 伪彩
13 // FB_VISUAL_DIRECTCOLOR         4 直彩
14
15     __u16 xpanstep;                /*如果没有硬件panning就赋值为0 */
16     __u16 ypanstep;                /*如果没有硬件panning就赋值为0 */
17     __u16 ywrapstep;               /*如果没有硬件ywrap就赋值为0 */
18
19     __u32 line_length;              /*一行的字节数 ,例:(RGB565)240*320,那
么这里就等于240*16/8 */
20     /*以下成员都可以不需要*/

```

```

20     unsigned long mmio_start;           /*内存映射IO的起始地址,用于应用层直接访问寄
存器,可以不需要*/
21     __u32 mmio_len;                  /* 内存映射IO的长度,可以不需要*/
22     __u32 accel;
23     __u16 reserved[3];
24
25 };

```

可变的参数 `fb_info-> var` 结构体如下:

```

1 structfb_var_screeninfo{
2     __u32xres;                      /*可见屏幕一行有多少个像素点*/
3     __u32yres;                      /*可见屏幕一列有多少个像素点*/
4     __u32xres_virtual;             /*虚拟屏幕一行有多少个像素点 */
5     __u32yres_virtual;             /*虚拟屏幕一列有多少个像素点*/
6     __u32xoffset;                  /*虚拟到可见屏幕之间的行偏移,若可见和虚拟的分辨率
一样,就直接设为0*/
7     __u32yoffset;                  /*虚拟到可见屏幕之间的列偏移*/
8     __u32bits_per_pixel;           /*每个像素的位数即BPP,比如:RGB565则填入16*/
9     __u32grayscale;               /*非0时,指的是灰度,真彩直接填0即可*/
10
11     struct fb_bitfield red;        //fb缓存的R位域, fb_bitfield结构体成员如下:
12     __u32 offset;                 /*区域偏移值,比如RGB565中的R,就在第11位
13     __u32 length;                /*区域长度,比如RGB565的R,共有5位
14     __u32 msb_right;             msb_right ==0,表示数据左边最大, msb_right!=0,表示数据右边最大
15
16     struct fb_bitfield green;    /*fb缓存的G位域*/
17     struct fb_bitfield blue;      /*fb缓存的B位域*/      /*以下参数都可以不填,默认
为0*/
18     struct fb_bitfield transp;   /*透明度,不需要填0即可*/
19     __u32nonstd;                 /* != 0表示非标准像素格式*/
20     __u32 activate;              /*设为0即可*/
21     __u32height;                 /*外设高度(单位mm),一般不需要填*/
22     __u32width;                  /*外设宽度(单位mm),一般不需要填*/
23     __u32accel_flags;            /*过时的参数,不需要填*/
24
25     /* 除了pixclock本身外, 其他的都以像素时钟为 单位*/
26     __u32pixclock;               /*像素时钟(皮秒)*/
27     __u32left_margin;            /*行切换,从同步到绘图之间的延迟*/
28     __u32right_margin;           /*行切换,从绘图到同步之间的延迟*/
29     __u32upper_margin;           /*帧切换,从同步到绘图之间的延迟*/
30     __u32lower_margin;           /*帧切换,从绘图到同步之间的延迟*/
31     __u32hsync_len;              /*水平同步的长度*/
32     __u32vsync_len;              /*垂直同步的长度*/
33     __u32sync;
34     __u32vmode;
35     __u32rotate;
36     __u32reserved[5];            /*保留*/
37
38 }

```

在驱动init入口函数中:

1. 分配一个 `fb_info` 结构体
2. 设置 `fb_info`
  - 2.1 设置固定的参数 `fb_info-> fix`。id,lcd的名字

2.2 设置可变的参数 `fb_info-> var`。可见屏幕一行有多少个像素点，虚拟屏幕一行有多少个像素点，每个像素的位数即BPP,比如:RGB565则填入16

2.3 设置操作函数 `fb_info-> fbops`

2.4 设置 `fb_info` 其它的成员，`my_lcdfb_setcolreg`调色板，`cfb_copyarea`复制数据

### 3. 设置硬件相关的操作

3.1 配置LCD引脚 `GPBcon = ioremap(0x56000010, 8); GPBdat = GPBcon+1;`

3.2 根据LCD手册设置LCD控制器 VSYNC, HSYNC 等参数

3.3 分配显存(framebuffer),把地址告诉LCD控制器和 `fb_info`

### 4. 开启LCD,并注册 `fb_info`: `register_framebuffer()`

4.1 直接在init函数中开启LCD(后面讲到电源管理,再来优化)，控制LCDCON5允许PWREN信号,然后控制LCDCON1输出PWREN信号,输出GPB0高电平来开背光,

4.2 注册 `fb_info`

在驱动exit出口函数中:

1. 卸载内核中的 `fb_info`
2. 控制LCDCON1关闭PWREN信号,关背光,iounmap注销地址
3. 释放DMA缓存地址 `dma_free_writecombine()`
4. 释放注册的 `fb_info`

LCD驱动完整代码如下所示:

```

1 #include <linux/module.h>
2 #include <linux/kernel.h>
3 #include <linux/errno.h>
4 #include <linux/string.h>
5 #include <linux/mm.h>
6 #include <linux/slab.h>
7 #include <linux/delay.h>
8 #include <linux/fb.h>
9 #include <linux/init.h>
10 #include <linux/dma-mapping.h>
11 #include <linux/interrupt.h>
12 #include <linux/workqueue.h>
13 #include <linux/wait.h>
14 #include <linux/platform_device.h>
15 #include <linux/clk.h>
16 #include <asm/io.h>
17 #include <asm/uaccess.h>
18 #include <asm/div64.h>
19 #include <asm/mach/map.h>
20 #include <asm/arch/regs-lcd.h>
21 #include <asm/arch/regs-gpio.h>
22 #include <asm/arch/fb.h>
23
24 /*LCD : 480*272      */
25 #define LCD_xres    480          //LCD 行分辨率
26 #define LCD_yres    272          //LCD列分辨率
27
28
29 /* GPIO prot   */
30 static unsigned long *GPBcon;
31 static unsigned long *GPCcon;

```

```

32 static unsigned long *GPDcon;
33 static unsigned long *GPGcon; //GPG4:控制LCD信号
34 static unsigned long *GPBdat; //GPB0: 控制背光
35
36 /* LCD control */
37 struct lcd_reg{
38     unsigned long lcdcon1;
39     unsigned long lcdcon2;
40     unsigned long lcdcon3;
41     unsigned long lcdcon4;
42     unsigned long lcdcon5;
43     unsigned long lcdsaddr1;
44     unsigned long lcdsaddr2;
45     unsigned long lcdsaddr3 ;
46     unsigned long redlut;
47     unsigned long greenlut;
48     unsigned long bluelut;
49     unsigned long reserved[9];
50     unsigned long dithmode;
51     unsigned long tpal ;
52     unsigned long lcdintpnd;
53     unsigned long lcdsrcpnd;
54     unsigned long lcdintmsk;
55     unsigned long tconsel;
56 };
57
58 static struct lcd_reg *lcd_reg;
59
60 static struct fb_info *my_lcd; //定义一个全局变量
61 static u32 pseudo_palette[16]; //调色板数组,被fb_info->pseudo_palette调用
62
63 static inline unsigned int chan_to_field(unsigned int chan, struct
fb_bitfield *bf)
64 {
65 /*内核中的单色都是16位,默认从左到右排列,比如G颜色[0x1f],那么chan就等于0XF800*/
66     chan &= 0xffff;
67     chan >>= 16 - bf->length; //右移,将数据靠到位0上
68     return chan << bf->offset; //左移一定偏移值,放入16色数据中对应的位置
69 }
70
71 static int my_lcdfb_setcolreg(unsigned int regno, unsigned int red,unsigned
int green, unsigned int blue,unsigned int transp, struct fb_info *info)
//设置调色板函数,供内核调用
72 {
73     unsigned int val;
74     if (regno >=16) //调色板数组不能大于15
75         return 1;
76
77     /* 用red,green,blue三个颜色值构造出16色数据val */
78     val = chan_to_field(red, &info->var.red);
79     val |= chan_to_field(green, &info->var.green);
80     val |= chan_to_field(blue, &info->var.blue);
81
82     ((u32 *) (info->pseudo_palette))[regno] = val; //放到调色板数组中
83     return 0;
84 }
85
86

```

```

87 static struct fb_ops my_lcdfb_ops = {
88     .owner          = THIS_MODULE,
89     .fb_setcolreg  = my_lcdfb_setcolreg,//调用my_lcdfb_setcolreg()函数,来设
90     //置调色板fb_info-> pseudo_palette
91     .fb_fillrect   = cfb_fillrect,      //填充矩形
92     .fb_copyarea   = cfb_copyarea,      //复制数据
93     .fb_imageblit  = cfb_imageblit,    //绘画图形,
94 };
95
96 static int lcd_init(void)
97 {
98     /*1.申请一个fb_info结构体*/
99     my_lcd= framebuffer_alloc(0,0);
100
101     /*2.设置fb_info*/
102
103     /* 2.1设置固定的参数fb_info-> fix */
104     /*my_lcd->fix.smem_start    物理地址后面注册MDA缓存区设置*/
105     strcpy(my_lcd->fix.id, "mylcd");                                //名字
106     my_lcd->fix.smem_len =LCD_xres*LCD_yres*2;                      //地址长
107     my_lcd->fix.type       =FB_TYPE_PACKED_PIXELS;                  // TFT STN
108     my_lcd->fix.visual     =FB_VISUAL_TRUECOLOR;                    //真彩色
109     my_lcd->fix.line_length =LCD_xres*2;                            //LCD 一行的字节
110
111     /* 2.2 设置可变的参数fb_info-> var */
112     my_lcd->var.xres        =LCD_xres;                                //可见屏X 分辨
113     率
114     my_lcd->var.yres        =LCD_yres;                                //可见屏y 分辨
115     率
116     my_lcd->var.xres_virtual =LCD_xres;                              //虚拟屏x分辨率
117     my_lcd->var.yres_virtual =LCD_yres;                              //虚拟屏y分辨率
118     my_lcd->var.xoffset     = 0;                                     //虚拟到可见屏幕之间
119     的行偏移
120     my_lcd->var.yoffset     =0;                                     //虚拟到可见屏幕之间
121     的行偏移
122
123     my_lcd->var.bits_per_pixel=16;                                 //像素为16BPP
124     my_lcd->var.grayscale   = 0;                                    //灰色比例
125
126     my_lcd->var.red.offset  = 11;
127     my_lcd->var.red.length  = 5;
128     my_lcd->var.green.offset = 5;
129     my_lcd->var.green.length = 6;
130     my_lcd->var.blue.offset = 0;
131     my_lcd->var.blue.length = 5;
132
133     /* 2.3 设置操作函数fb_info-> fbops */
134     my_lcd->fbops           = &my_lcdfb_ops;
135
136     /* 2.4 设置fb_info 其它的成员 */
137     /*my_lcd->screen_base    虚拟地址在后面注册MDA缓存区设置*/
138     my_lcd->pseudo_palette  =pseudo_palette;                         //保存调色板数组
139     my_lcd->screen_size      =LCD_xres * LCD_yres *2;                //虚拟地址长
140
141     /*3    设置硬件相关的操作*/
142     /*3.1 配置LCD引脚*/
143     GPBcon                   = ioremap(0x56000010, 8);

```

```

140 GPBdat           = GPBcon+1;
141 GPCcon          = ioremap(0x56000020, 4);
142 GPDcon          = ioremap(0x56000030, 4);
143 GPGcon          = ioremap(0x56000060, 4);
144
145 *GPBcon          &=~(0x03<<(0*2));
146 *GPBcon          |= (0x01<<(0*2));      //PGB0背光
147 *GPBdat          &=~(0x1<<0);          //关背光
148 *GPCcon          =0aaaaaaaaaa;
149 *GPDcon          =0aaaaaaaaaa;
150 *GPGcon          |=(0x03<<(4*2));     //GPG4:LCD信号
151
152 /*3.2 根据LCD手册设置LCD控制器,参考之前的裸机驱动*/
153 lcd_reg=ioremap(0x4d000000, sizeof( lcd_reg) );
154 /*HCLK:100Mhz */
155 lcd_reg->lcdcon1 = (4<<8) | (0x3<<5) | (0x0c<<1) ;
156 lcd_reg->lcdcon2 = ((3)<<24) | (271<<14) | ((1)<<6) |((0)<<0) ;
157 lcd_reg->lcdcon3 = ((16)<<19) | (479<<8) | ((10));
158 lcd_reg->lcdcon4 = (4);
159 lcd_reg->lcdcon5 = (1<<11) | (1<<9) | (1<<8) |(1<<0);
160
161 lcd_reg->lcdcon1 &=~(1<<0);          // 关闭PWREN信号输出
162 lcd_reg->lcdcon5 &=~(1<<3);          //禁止PWREN信号
163
164 /* 3.3 分配显存(framebuffer),把地址告诉LCD控制器和fb_info*/
165 my_lcd->screen_base=dma_alloc_writecombine(0,my_lcd->fix.smem_len,
&my_lcd->fix.smem_start, GFP_KERNEL);
166
167 /*lcd控制器的地址必须是物理地址*/
168 lcd_reg->lcdsaddr1 =(my_lcd->fix.smem_start>>1)&0X3FFFFFF; //保存缓冲起
始地址A[30:1]
169 lcd_reg->lcdsaddr2 =((my_lcd->fix.smem_start+my_lcd-
>screen_size)>>1)&0X1FFFFF; //保存存缓冲结束地址A[21:1]
170 lcd_reg->lcdsaddr3 =LCD_xres& 0x3ff;           //OFFSIZE[21:1]:保存
LCD上一行结尾和下一行开头的地址之间的差
//PAGEWIDTH [10:0]:保存LCD一行占的宽度(半字数为单位)
171
172 /*4开启LCD,并注册fb_info: register_framebuffer()*/
173 /*4.1 直接在init函数中开启LCD(后面讲到电源管理,再来优化)*/
174 lcd_reg->lcdcon1 |=1<<0;          //输出PWREN信号
175 lcd_reg->lcdcon5 |=1<<3;          //允许PWREN信号
176 *GPBdat           |=(0x1<<0);        //开背光
177
178 /*4.2 注册fb_info*/
179 register_framebuffer(my_lcd);
180 return 0;
181 }
182 static int lcd_exit(void)
183 {
184 /* 1卸载内核中的fb_info*/
185 unregister_framebuffer(my_lcd);
186 /*2 控制LCDCON1关闭PWREN信号,关背光,iounmap注销地址*/
187 lcd_reg->lcdcon1 &=~(1<<0);          // 关闭PWREN信号输出
188 lcd_reg->lcdcon5 &=~(1<<3);          //禁止PWREN信号
189 *GPBdat           &=~(0x1<<4);        //关背光
190 iounmap(GPBcon);
191 iounmap(GPCcon);
192 iounmap(GPDcon);

```

```

193     iounmap(GPGcon);
194
195     /*3.释放DMA缓存地址dma_free_writecombine()*/
196     dma_free_writecombine(0,my_lcd->screen_size,my_lcd->screen_base,my_lcd-
197     >fix.smem_start);
198
199     /*4.释放注册的fb_info*/
200     framebuffer_release(my_lcd);
201
202     return 0;
203 }
204
205 module_init(lcd_init);
206 module_exit(lcd_exit);
207 MODULE_LICENSE("GPL");

```

## 总线设备驱动模型

自内核2.6版本开始，需要关注的是总线、设备和驱动这3个实体，总线将设备和驱动绑定。在Linux内核系统中注册一个设备的时候，会寻找与之对应驱动进行匹配；相反地，系统中注册一个驱动的时候，会去寻找一个对应的设备进行匹配。匹配的工作由总线来完成。

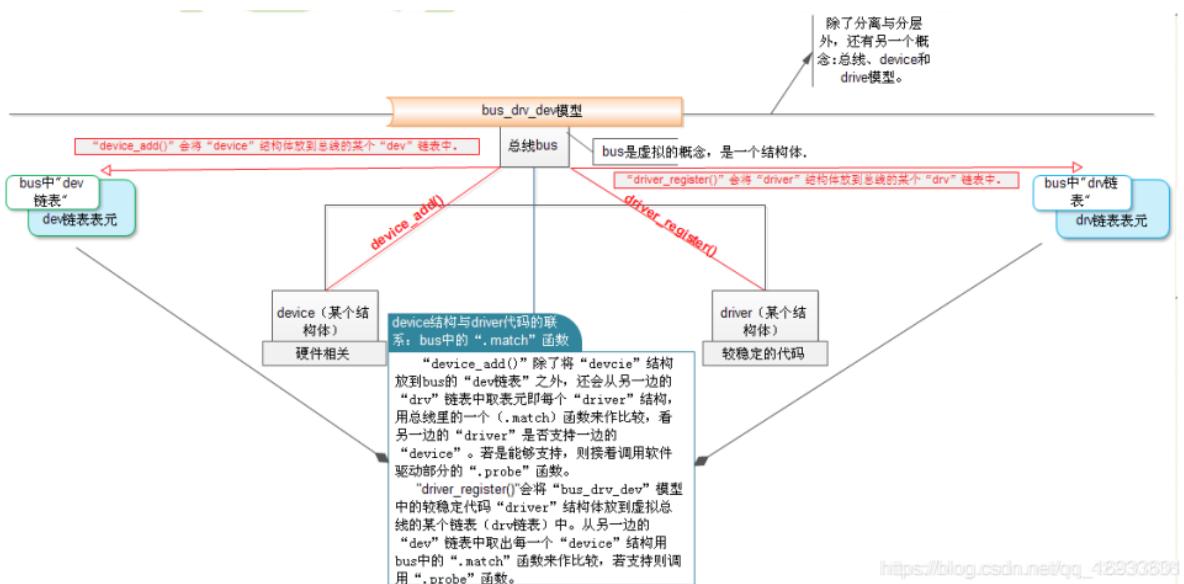
**在Linux设备中有的是没有对应的物理总线的，但为了适配Linux的总线模型，内核针对这种没有物理总线的设备开发了一种虚拟总线——platform总线。**

将设备和驱动独立开，驱动尽可能写的通用，当来了一个类似的设备后也可以使用这个驱动，让驱动程序可以重用。这体现了Linux驱动的软件架构设计的思想。

按照这个思路，Linux中的设备和驱动都需要挂接在一种总线上，比如i2c总线上的eeprom，**eeprom作为设备，eeprom的驱动都挂接在i2c驱动上**。但是在嵌入式系统中，soc系统一般都会集成独立的i2c控制器，控制器也是需要驱动的，但是再按照设备-总线-驱动模型进行设计，就会发现无法找到一个合适总线去挂接控制器设备和控制器驱动了（i2c控制器是挂接在CPU内部的总线上，而不是i2c总线），所以Linux发明了一种虚拟总线，称为**platform总线**，相应的设备称为**platform\_device**（控制器设备），对应的驱动为**platform\_driver**（控制器驱动），用**platform总线**来承载这些相对特殊的系统。

**注意：****所谓的platform\_device并不是与字符设备、块设备和网络设备并列的概念，而是Linux系统提供的一种附加手段**，例如，在S3C6410处理器中，把内部集成的I2C、RTC、SPI、LCD、看门狗等**控制器**都归纳为**platform\_device**，而它们本身就是字符设备。我们要记住，platform驱动只是在**字符设备驱动外套一层platform\_driver的外壳**。引入platform模型符合Linux设备模型——总线、设备、驱动，设备模型中配套的sysfs节点都可以用，方便我们的开发；**当然你也可以选择不用，不过就失去了一些platform带来的便利**；

设备驱动中引入platform概念，隔离BSP和驱动。在BSP中定义platform设备和设备使用的资源、设备的具体匹配信息，而在驱动中，只需要通过API去获取资源和数据，做到了板相关代码和驱动代码的分离，使得驱动具有更好的可扩展性和跨平台性。



[https://blog.csdn.net/qq\\_42933603](https://blog.csdn.net/qq_42933603)

## 下面分析下总线设备驱动模型的匹配过程

一边的“device”结构体和另一边的“较稳定的 drivice 代码”的联系：“device\_add()”除了将“devcie”结构放到 bus 的“dev 链表”之外，还会从另一边的“drv”链表中取表示元即某个“driver”结构，用总线里的一个 (.match) 函数来作比较，看另一边的“driver”是否支持一边的“device”。若是能够支持，则接着调用软件驱动部分的“.probe”函数。

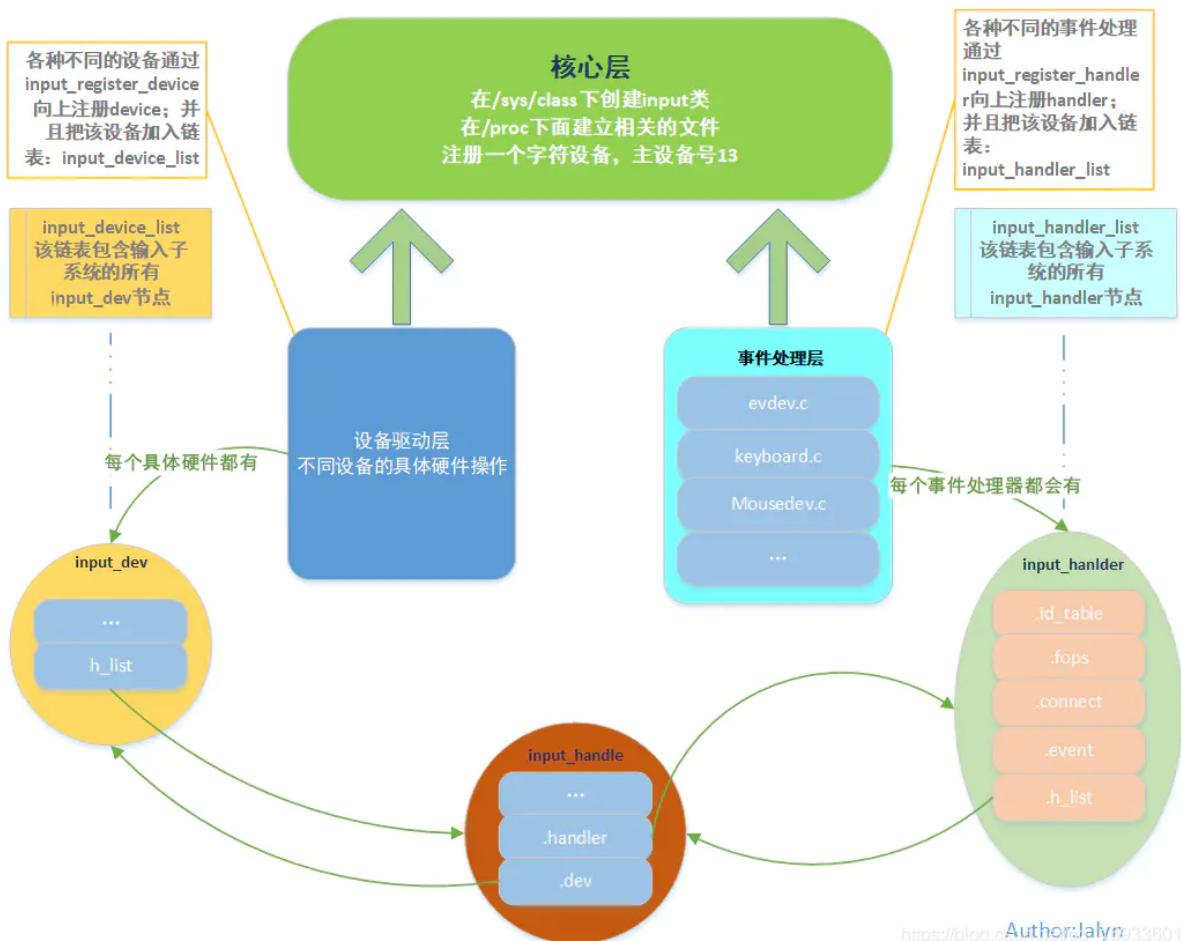
“driver\_register()”会将“bus\_drv\_dev”模型中的较稳定代码“driver”结构体放到虚拟总线的某个链表 (drv 链表) 中。从另一边的“dev”链表中取出每一个“device”结构用 bus 中的“.match”函数来作比较，若支持则调用“.probe”函数。左右两个注册就建立起来的一种机制。在“.probe”函数中做的事件由自己决定，打印一句话，或注册一个字符设备，再或注册一个“input\_dev”结构体等等都是由自己决定。强制的把一个驱动程序分为左右两边这种机制而已，可以把这套东西放在任何地方，这里的“driver”只是个结构体不要被这个名字迷惑，“device”也只是个结构体，里面放什么内容都是由自己决定的。

## 输入子系统模型

每个硬件都有一个input\_dev结构体，每个软件都有一个input\_handler结构体。input\_dev和input\_handler分别通过input\_register\_device(),input\_register\_handler()向核心层注册硬件和软件。

```

1 int input_register_device(struct input_dev *dev) /*dev:要注册的驱动设备
2 {
3     ...
4         list_add_tail(&dev->node, &input_dev_list); // (1)放入链表中
5     ...
6         list_for_each_entry(handler, &input_handler_list, node) // (2)
7             input_attach_handler(dev, handler);
8     ...
9 }
```



<https://blog.cAuthor:Jalyr933601>

从input\_dev方向分析：input设备在增加到input\_dev\_list链表上之后，会查找 input\_handler\_list事件处理链表上的handler进行匹配，这里的匹配方式与总线设备驱动模型的device和driver匹配过程很相似，所有的input\_device都挂在input\_dev\_list上，所有类型的事件都挂在input\_handler\_list 上，进行“匹配相亲”。如果匹配上了，就调用input\_handler的connect函数进行连接。设备就是在此时注册的

从input\_handler方向分析：将handler挂到链表input\_handler\_list下，然后遍历input\_dev\_list链表,查找并匹配输入设备对应的事件处理层，如果匹配上了，就调用connect函数进行连接，并创建input\_handle结构。

所以，不管新添加input\_dev还是input\_handler,都会进入input\_attach\_handler()判断两者id是否有支持,若两者支持便进行连接。

## platform总线的匹配规则是什么？在具体应用上要不要先注册驱动再注册设备？有先后顺序没？

总线，设备，驱动。匹配规则就是当有一个新的设备挂起时，总线被唤醒，match函数被调用，用device名字去跟本总线下的所有驱动名字去比较。相反就是用驱动的名字去device链表中和所有device的名字比较。如果匹配上，才会调用驱动中的probe函数，否则不调用。至于先后顺序，鉴于个人理解，不会有影响，不管谁先谁后，bus都会完成匹配工作。谈谈对Linux设备驱动模型的认识：设备驱动模型的出现主要有三个好处，设备与驱动分离，驱动可移植性增强；设备驱动抽象结构以总线结构表示看起来更加清晰明了，谁是属于哪一条bus的；最后，就是大家最熟悉的热插拔了，设备与驱动分离，很好的奠定了热插拔机制。



## Linux内核

### 内核镜像格式有几种？分别有什么区别？

1. uboot经过编译直接生成的elf格式的可执行程序是u-boot，这个程序类似于windows下的exe格式，在操作系统下是**可以直接执行的**。但是这种格式**不能用来烧录下载**。我们用来烧录下载的是u-boot.bin，这个东西是由u-boot使用arm-linux-objcopy工具进行加工（主要目的是去掉一些无用的）得到的。这个u-boot.bin就叫镜像（image），镜像就是用来烧录到iNand中执行的。
2. linux内核经过编译后也会生成一个elf格式的可执行程序，叫vmlinuz或vmlinux，这个就是**原始的未经任何处理加工的原版内核elf文件**；嵌入式系统部署时烧录的一般不是这个vmlinuz/vmlinux，而是要用objcopy工具去制作成烧录镜像格式（就是u-boot.bin这种，但是内核没有.bin后缀），经过制作加工成烧录镜像的文件就叫Image（制作把78M大的精简成了7.5M，因此这个制作烧录镜像主要目的就是**缩减大小，节省磁盘**）。
3. 原则上Image就可以直接被烧录到Flash上进行启动执行（类似于u-boot.bin），但是实际上并不是这么简单。实际上linux的作者们觉得Image还是太大了所以对Image进行了压缩，并且在image压缩后的文件的**前端附加了一部分解压缩代码**。构成了一个压缩格式的镜像就叫zImage。（因为当年Image大小刚好比一张软盘（软盘有2种，1.2M的和1.44MB两种）大，为了节省1张软盘的钱于是乎设计了这种压缩Image成zImage的技术）。
4. uboot为了启动linux内核，还发明了一种内核格式叫ulimage。ulimage是由zImage加工得到的，uboot中有一个工具，可以将zImage加工生成ulimage。注意：ulimage不关心linux内核的事，linux内核只管生成zImage即可，然后uboot中的mkimage工具再去由zImage加工生成ulimage来给uboot启动。**这个加工过程其实就是在zImage前面加上64字节的ulimage的头信息即可**。
5. 原则上uboot启动时应该给他ulimage格式的内核镜像，但是实际上uboot中也可以支持zImage，是否支持就看x210\_sd.h中是否定义了LINUX\_ZIMAGE\_MAGIC这个宏。所以大家可以看出：有些uboot是支持zImage启动的，有些则不支持。但是**所有的uboot肯定都支持ulimage启动**。
6. 如果直接在kernel底下去make ulimage会提供mkimage command not found。解决方案是去uboot/tools下cp mkimage /usr/local/bin/，复制mkimage工具到系统目录下。再去make ulimage即可。

通过上面的介绍我们了解了内核镜像的各种格式，如果通过uboot启动内核，**Linux必须为ulimage格式**。

### 内核中申请内存有哪几个函数？有什么区别？

#### kmalloc

```
1 | void *kmalloc(size_t size, gfp_t flags)
```

kmalloc是**内核中最常用的一种内存分配方式**，它通过调用kmem\_cache\_alloc函数来实现。kmalloc一次最多能申请的内存大小由include/linux/Kmalloc\_size.h的内容来决定，在默认的2.6.18内核版本中，kmalloc一次最多能申请大小为131702B也就是**128KB字节**的连续物理内存。测试结果表明，如果试图用kmalloc函数分配大于128KB的内存，编译不能通过。

#### vmalloc

```
1 | void *vmalloc(unsigned long size)
```

前面几种内存分配方式都是物理连续的，能保证较低的平均访问时间。但是在某些场合中，对内存区的请求不是很频繁，较高的内存访问时间也可以接受，这是就可以分配一段线性连续，物理不连续的地址，带来的好处是一次可以**分配较大块的内存**。图3-1表示的是vmalloc分配的内存使用的地址范围。vmalloc对一次能分配的内存大小没有明确限制。出于性能考虑，应谨慎使用vmalloc函数。在测试过程中，**最大能一次分配1GB的空间**。

### dma\_alloc\_coherent

```
1 | void *dma_alloc_coherent(struct device *dev, size_t size, ma_addr_t
 *dma_handle, gfp_t gfp)
```

DMA是一种硬件机制，允许外围设备和主存之间直接传输IO数据，而不需要CPU的参与，使用DMA机制能大幅提高与设备通信的吞吐量。DMA操作中，涉及到CPU高速缓存和对应的内存数据一致性的问题，必须保证两者的数据一致，在x86\_64体系结构中，硬件已经很好的解决了这个问题，**dma\_alloc\_coherent**和**get\_free\_pages**函数实现差别不大，前者实际是调用**alloc\_pages**函数来分配内存，因此一次分配内存的大小限制和后者一样。**\_get\_free\_pages**分配的内存同样可以用于DMA操作。测试结果证明，**dma\_alloc\_coherent**函数一次能分配的最大内存也为4M。

### ioremap

```
1 | void * ioremap (unsigned long offset, unsigned long size)
```

**ioremap**是一种更直接的内存“分配”方式，使用时直接指定物理起始地址和需要分配内存的大小，然后将该段物理地址映射到内核地址空间。**ioremap**用到的物理地址空间都是事先确定的，和上面的几种内存分配方式并不太一样，并不是分配一段新的物理内存。**ioremap**多用于设备驱动，可以让CPU直接访问外部设备的IO空间。**ioremap**能映射的内存由原有的物理内存空间决定，所以没有进行测试。

## 什么是内核空间，用户空间？

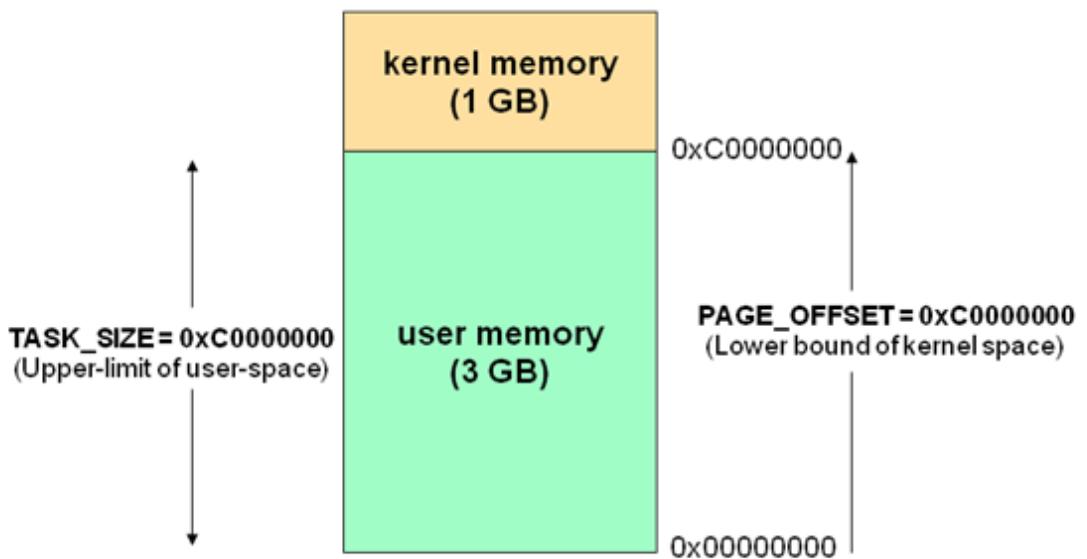
对32位操作系统而言，它的寻址空间（虚拟地址空间，或叫线性地址空间）为4G（2的32次方）。也就是说一个进程的最大地址空间为4G。

操作系统的内核是核心(kernel)，它独立于普通的应用程序，可以访问受保护的内存空间，也有访问底层硬件设备的所有权限。为了保证内核的安全，现在的操作系统一般都强制**用户进程不能直接操作内核**。具体的实现方式基本都是由**操作系统将虚拟地址空间划分为两部分**，一部分为内核空间，另一部分为用户空间。针对Linux操作系统而言，**最高的1G字节**（从虚拟地址0xC0000000到0xFFFFFFFF）由内核使用，称为**内核空间**。而**较低的3G字节**（从虚拟地址0x00000000到0xBFFFFFFF）由各个进程使用，称为**用户空间**。

对上面这段内容我们也可以这样理解：

每个进程的4G地址空间中，最高1G都是一样的，即内核空间。只有剩余的3G才归进程自己使用。换句话说就是，**最高1G的内核空间是被所有进程共享的！**

下图描述了每个进程4G地址空间的分配情况：



[https://blog.csdn.net/qq\\_16933601](https://blog.csdn.net/qq_16933601)

## 为什么需要区分内核空间与用户空间？

在 CPU 的所有指令中，有些指令是非常危险的，如果错用，将导致系统崩溃，比如清内存、设置时钟等。如果允许所有的程序都可以使用这些指令，那么系统崩溃的概率将大大增加。

所以，CPU 将指令分为特权指令和非特权指令，对于那些危险的指令，只允许操作系统及其相关模块使用，普通应用程序只能使用那些不会造成灾难的指令。比如 Intel 的 CPU 将特权等级分为 4 个级别：Ring0~Ring3。

其实 Linux 系统只使用了 Ring0 和 Ring3 两个运行级别(Windows 系统也是一样的)。当进程运行在 Ring3 级别时被称为运行在用户态，而运行在 Ring0 级别时被称为运行在内核态。

## 什么是内核态和用户态？

当进程运行在内核空间时就处于内核态，而进程运行在用户空间时则处于用户态。

在内核态下，进程运行在内核地址空间中，此时 CPU 可以执行任何指令。运行的代码也不受任何的限制，可以自由地访问任何有效地址，也可以直接进行端口的访问。

在用户态下，进程运行在用户地址空间中，被执行的代码要受到 CPU 的诸多检查，它们只能访问映射其地址空间的页表项中规定的在用户态下可访问页面的虚拟地址，且只能对任务状态段(TSS)中 I/O 许可位图(I/O Permission Bitmap)中规定的可访问端口进行直接访问。

对于以前的 DOS 操作系统来说，是没有内核空间、用户空间以及内核态、用户态这些概念的。可以认为所有的代码都是运行在内核态的，因而，用户编写的应用程序代码可以很容易的让操作系统崩溃掉。

对于 Linux 来说，通过区分内核空间和用户空间的设计，**隔离了操作系统代码**(操作系统的代码要比应用程序的代码健壮很多)与**应用程序代码**。即便是单个应用程序出现错误，也不会影响到操作系统的稳定性，这样其它的程序还可以正常的运行(Linux 可是个多任务系统啊！)。所以，**区分内核空间和用户空间本质上是要提高操作系统的稳定性及可用性**。

## 用户空间与内核通信方式有哪些？

### 1. 使用API

- ```

1 | get_user(x, ptr) //在内核中被调用，获取用户空间指定地址的数值并保存到内核变量x中。
2 | put_user(x, ptr) //在内核中被调用，将内核空间的变量x的数值保存到到用户空间指定地址处。
3 | Copy_from_user() / copy_to_user() //主要应用于设备驱动读写函数中，通过系统调用触发。

```

## 2. 使用proc文件系统。

和sysfs文件系统类似，也可以作为内核空间和用户空间交互的手段。/proc文件系统是一种虚拟文件系统，通过它可以作为一种linux内核空间和用户空间的。与普通文件不同，这里的虚拟文件的内容都是动态创建的。使用/proc文件系统的方式很简单。调用create\_proc\_entry，返回一个proc\_dir\_entry指针，然后去填充这个指针指向的结构就好了。

## 3. 使用sysfs文件系统+kobject

每个在内核中注册的kobject都对应着sysfs系统中的一个目录。可以通过读取根目录下的sys目录中的文件来获得相应的信息。除了sysfs文件系统和proc文件系统之外，一些其他的虚拟文件系统也能同样达到这个效果。

## 4. netlink

netlink socket提供了一组类似于BSD风格的API，用于用户态和内核态的IPC。相比于其他的用户态和内核态IPC机制，netlink有几个好处：1.使用自定义一种协议完成数据交换，不需要添加一个文件等。2.可以支持多点传送。3.支持内核先发起会话。4.异步通信，支持缓存机制。

## 5. 文件

应该说这是一种比较笨拙的做法，不过确实可以这样用。当处于内核空间的时候，直接操作文件，将想要传递的信息写入文件，然后用户空间可以读取这个文件便可以得到想要的数据了。下面是一个简单的测试程序，在内核态中，程序会向“/home/melody/str\_from\_kernel”文件中写入一条字符串，然后我们在用户态读取这个文件，就可以得到内核态传输过来的数据了。

## 6. 使用mmap系统调用

可以将内核空间的地址映射到用户空间。在以前做嵌入式的时候用到几次。一方面可以在driver中修改Struct file\_operations结构中的mmap函数指针来重新实现一个文件对应的映射操作。另一方面，也可以直接打开/dev/mem文件，把物理内存中的某一页映射到进程空间中的地址上。

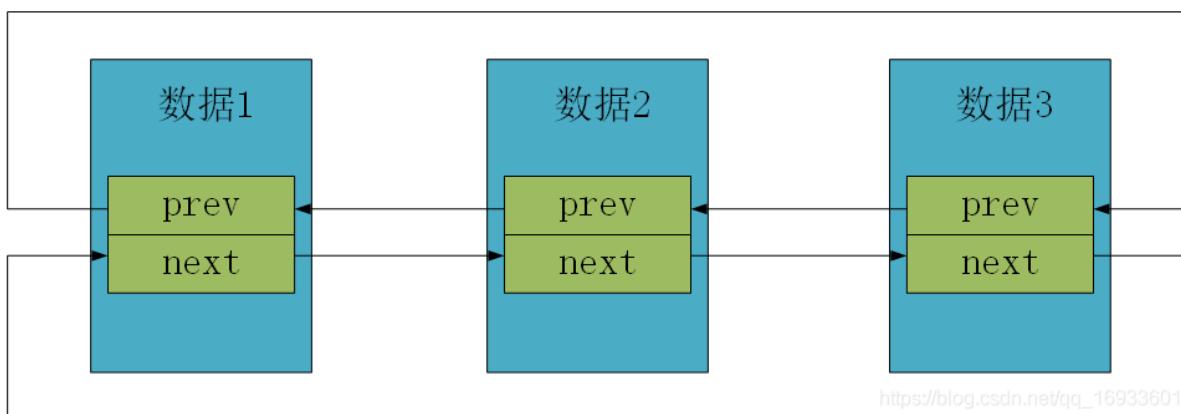
其实，除了重写Struct file\_operations中mmap函数，我们还可以重写其他的方法如ioctl等，来达到驱动内核空间和用户空间通信的方式。

## 7. 信号

从内核空间向进程发送信号。这个倒是经常遇到，用户程序出现重大错误，内核发送信号杀死相应进程。

## 内核链表为什么具有通用性？

内核中由于要管理大量的设备，但是各种设备各不相同，必须将他们统一起来管理，于是内核设计者就想到了使用通用链表来处理，通用链表看似神秘，实际上就是**双向循环链表**，这个链表的每个节点都是只有指针域，没有任何数据域。



使用通用链表的好处是：

1. 通用链表中每个节点中**没有数据域**，也就是说无论数据结构有多复杂在链表中**只有前后级指针**。
2. 如果一个数据结构（即是描述设备的设备结构体）想要用通用链表管理，只需要在**结构体中包含节点的字段即可**。

3. 双向链表可以从任意一个节点的前后遍历整个链表，遍历非常方便。
4. 使用循环链表使得可以不断地循环遍历管理节点，像进程的调度：操作系统会把就绪的进程放在一个管理进程的就绪队列的通用链表中管理起来，循环不断地，为他们分配时间片，获得cpu进行周而复始的进程调度。

## 应用程序中open()在linux中执行过程中是如何从用户空间到内核空间？

1. 应用层调用open函数，在VFS层中找到**struct inode结构体**，判断是字符设备还是块设备，根据设备号，可以找到对应的驱动程序。
2. 在驱动层中，每个字符设备都有一个**struct cdev结构体**，这个结构体通过struct inode结构体中的*cdev*把连接起VFS层和驱动层，struct cdev结构体描述了字符设备所有信息，其中最重要的一项就是字符设备的**操作函数接口**
3. struct cdev结构体中的**struct file结构体**记录了操作字符设备的一些函数，比如open read write函数等。  
struct file结构体其实是在VFS层的，通过struct file结构体指针指向驱动层的struct file结构体将驱动层函数和VFS层链接起来
4. 任务完成，VFS层会给应用返回一个**文件描述符 (fd)**。这个fd是和**struct file结构体**对应的。

## 怎样申请大块内核内存？

vmalloc



## 设备驱动

### 请简述主设备号和次设备号的用途。

**主设备号：**主设备号标识设备对应的特定的驱动程序。虽然现代的linux内核允许多个驱动程序共享主设备号，但我们看待的大多数设备仍然按照“一个主设备对应一个驱动程序”的原则组织。

**次设备号：**次设备号由内核使用，用于确定由主设备号对应驱动程序中的各个设备。依赖于驱动程序的编写方式，我们可以通过次设备号获得一个指向内核设备的直接指针，也可将此设备号当作设备本地数组的索引。

### 字符型驱动设备怎么创建设备文件？

1. 手动创建

```
mknod /dev/led c 250 0 , 其中dev/led 为设备节点,c 代表字符设备, 250代表主设备号, 0代表次设备号。
```

2. 自动创建

UDEV/MDEV是运行在用户态的程序，可以动态管理设备文件，包括创建和删除设备文件，运行在用户态意味着系统要运行之后，在 /etc/init.d/rcS 脚本文件中会执行 mdev -s 自动创建设备节点。

## 设备驱动程序中如何注册一个字符设备？分别解释一下它的几个参数的含义。

注册一个字符设备驱动有两种方法：

1. `void cdev_init(struct cdev *cdev, struct file_operations *fops)`

该注册函数可以将cdev结构嵌入到自己的设备特定的结构中。cdev是一个指向结构体cdev的指针，而fops是指向一个类似于file\_operations结构（可以是file\_operations结构，但不限于该结构）的指针。

2. `int register_chrdev(unsigned int major, const char *name, struct file_operations *fopen);`

该注册函数是早期的注册函数，major是设备的主设备号，name是驱动程序的名称，而fops是默认的file\_operations结构（这是只限于file\_operations结构）。对于register\_chrdev的调用将为给定的主设备号注册0 - 255作为次设备号，并为每个设备建立一个对应的默认cdev结构。

## /dev/下面的设备文件是怎么创建出来的？

普遍说法有三种方式，devfs机制，udev机制，再有一个就是手动创建设备节点。谈谈个人见解：

1. devfs机制从来没用过，应该是2.6以前的内核使用的；
2. udev，其实就是现在常用的device\_create()、class\_create()这一套接口，所谓udev是上层用户空间程序，是基于驱动中创建使用了这两个接口而起作用的，但是udev在日常开发中几乎接触不到，我们只需在驱动中调用创建节点的这两个API就ok了，剩下的工作就交给udev去做了，有想深究它具体实现原理的那就自己去研究吧，我觉得会用就行了；
3. mknod，新手最常用的一种创建设备节点方法，但并非入门后就再没有用途，在某些情境下，或许有人不想使用udev机制，于是把节点创建工作写在脚本里，这样也是无可厚非的。

## Linux设备中字符设备和块设备有什么主要区别？分别举例。

Linux中I/O设备分为两类：块设备和字符设备。两种设备本身没有严格限制，但是，基于不同的功能进行了分类。

**字符设备：**提供**连续**的数据流，应用程序可以**顺序**读取，通常不支持随机存取。相反，此类设备支持**按字节/字符**来读写数据。字符终端、串口、鼠标、键盘、摄像头、声卡和显卡等就是典型的字符设备。

**块设备：**应用程序可以**随机访问**设备数据，程序可自行确定读取数据的位置。硬盘是典型的块设备，应用程序可以寻址磁盘上的**任何位置**，并由此读取数据。此外，数据的读写只能以块(通常是512B)的倍数进行。与字符设备不同，块设备并不支持基于字符的寻址。如：u盘，SD卡，磁盘等。

## 驱动中操作物理绝对地址为什么要先ioremap？

ioremap是内核中用来将外设寄存器物理地址映射到主存上去的接口，即将io地址空间映射到虚拟地址空间上去，便于操作。为什么非要映射呢，因为保护模式下的cpu只认虚拟地址，不认物理地址，给它物理地址它并不帮你做事，所以你要操作外设上的寄存器必须先映射到虚拟内存空间，拿着虚拟地址去跟cpu对接，从而操作寄存器。

## insmod，rmmmod一个驱动模块，会执行模块中的哪个函数？在设计上要注意哪些问题？

分别会执行 `module_init()` 和 `module_exit()` 指定的init函数和exit函数。要注意的就是，尽量使在init函数中出现的资源申请及使用，都要有对应的释放操作在exit中，即init申请，exit释放。

## NAND驱动的probe流程

probe 函数就会与NAND 芯片进行，主要做的事情主要包括这几个方面：读取NAND 芯片的ID，然后查表得到这片NAND 芯片的如厂商，page size，erase size 以及chip size 等信息，接着，根据struct nand\_chip 中options 的值的不同，或者在NAND 芯片中的特定位置查找bad block table，或者scan 整个NAND 芯片，并在内存中建立bad block table。说起来复杂，但其实所有的这些动作，都可以在MTD 提供的一个叫做nand\_scan 的函数中完成。

## Linux驱动开发中，常用的调试方法有哪些？

利用printk，查看OOP消息，利用strace，利用内核内置的hacking选项，利用ioctl方法，利用/proc 文件系统，使用kgdb。

建议大家，亲自动手调试下。面试中，很大可能会问你，在写驱动过程中遇到了什么问题的，如何解决的？

如果你能讲出以上几种调试方法中的一两种，一定会让面试官刮目相看！



## Linux驱动开发常用函数

### ioremap

#### 简介

```
1 void * __ioremap(unsigned long phys_addr, unsigned long size, unsigned long flags)
2 void *ioremap(unsigned long phys_addr, unsigned long size)
```

入口：phys\_addr：要映射的起始的IO地址；

size：要映射的空间的大小；

flags：要映射的IO空间的和权限有关的标志；

phys\_addr：是要映射的物理地址

size：是要映射的长度，单位是字节

头文件：io.h

#### 主要功能

将一个IO地址空间映射到内核的虚拟地址空间上去，便于访问。

ioremap是内核提供的用来映射外设寄存器到主存的函数，我们要映射的地址已经从pci\_dev中读了出来（上一步），这样就水到渠成的成功映射了而不会和其他地址有冲突。映射完了有什么效果呢？我举个例子，比如某个网卡有100个寄存器，他们都是连在一块的，位置是固定的，假如每个寄存器占4个字节那么一共400个字节的空间被映射到内存成功后，ioaddr就是这段地址的开头（注意ioaddr是虚拟地址，而mmio\_start是物理地址，它是BIOS得到的，肯定是物理地址，而保护模式下CPU不认物理地址，

只认虚拟地址），ioaddr+0就是第一个寄存器的地址，ioaddr+4就是第二个寄存器地址（每个寄存器占4个字节），以此类推，我们就能够在内存中访问到所有的寄存器进而操控他们了。

## open

### 函数定义

```
1 | int open( const char * pathname, int flags);
2 | int open( const char * pathname, int flags, mode_t mode);
```

### 参数说明

**pathname**：文件的名称，可以包含（绝对和相对）路径

**flags**：文件打开模式

**mode**: 用来规定对该文件的所有者，文件的用户组及系统中其他用户的访问权限，则文件权限为：  
mode&(~umask)

## read

### 函数定义

```
1 | ssize_t read(int fd, void * buf, size_t count);
```

### 函数说明

read()会把参数fd所指的文件传送count个字节到buf指针所指的内存中。

### 返回值

返回值为实际读取到的字节数，如果返回0，表示已到达文件尾或是无可读取的数据。若参数count为0，则read()不会有作用并返回0。

### 注意

read时fd中的数据如果小于要读取的数据，就会引起阻塞。

read的用法比write较为简单，此处不叙述过多。由于笔者水平也有限，如果文中有谬误之处还恳请诸位指出，以免误导大家。

## write

### 函数定义

```
1 | ssize_t write (int fd, const void * buf, size_t count);
```

### 函数说明

write()会把参数buf所指的内存写入count个字节到参数放到所指的文件内。

### 返回值

如果顺利write()会返回**实际写入的字节数**。当有错误发生时则返回-1，错误代码存入errno中。

### 说明

1. write()函数返回值一般无0，只有当如下情况发生时才会返回0： write(fp, p1+len, (strlen(p1)-len) 中第三参数为0，此时write()什么也不做，只返回0。

2. write()函数从buf写数据到fd中时，若buf中数据无法一次性读完，那么第二次读buf中数据时，其读位置指针（也就是第二个参数buf）不会自动移动，需要程序员编程控制。而不是简单的将buf首地址填入第二参数即可。如可按如下格式实现读位置移动：write(fp, p1+len, (strlen(p1)-len)。这样write第二次循环时变会从p1+len处写数据到fp, 之后的也由此类推，直至(strlen(p1)-len变为0。
3. 在write一次可以写的最大数据范围内（貌似是BUFSIZ ,8192），第三参数count大小最好为buf中数据的大小，以免出现错误。（经过笔者再次试验，write一次能够写入的并不只有8192这么多，作者尝试一次写入81920000，结果也是可以，看来其一次最大写入数据并不是8192，但内核中确实有BUFSIZ这个参数，具体指什么还有待研究）

## copy\_to\_user

### 函数定义

```
1 | unsigned long copy_to_user(void *to, const void *from, unsigned long n)
```

### 参数说明

to: 目标地址（用户空间）

from: 源地址（内核空间）

n: 将要拷贝数据的字节数

### 函数说明

从内核空间中读取数据到用户空间。

### 返回值

成功返回0，失败返回没有拷贝成功的数据字节数。

## copy\_from\_user

### 函数定义

```
1 | unsigned long copy_from_user(void *to, const void *from, unsigned long n);
```

### 参数说明

to: 目标地址（内核空间）

from: 源地址（用户空间）

n: 将要拷贝数据的字节数

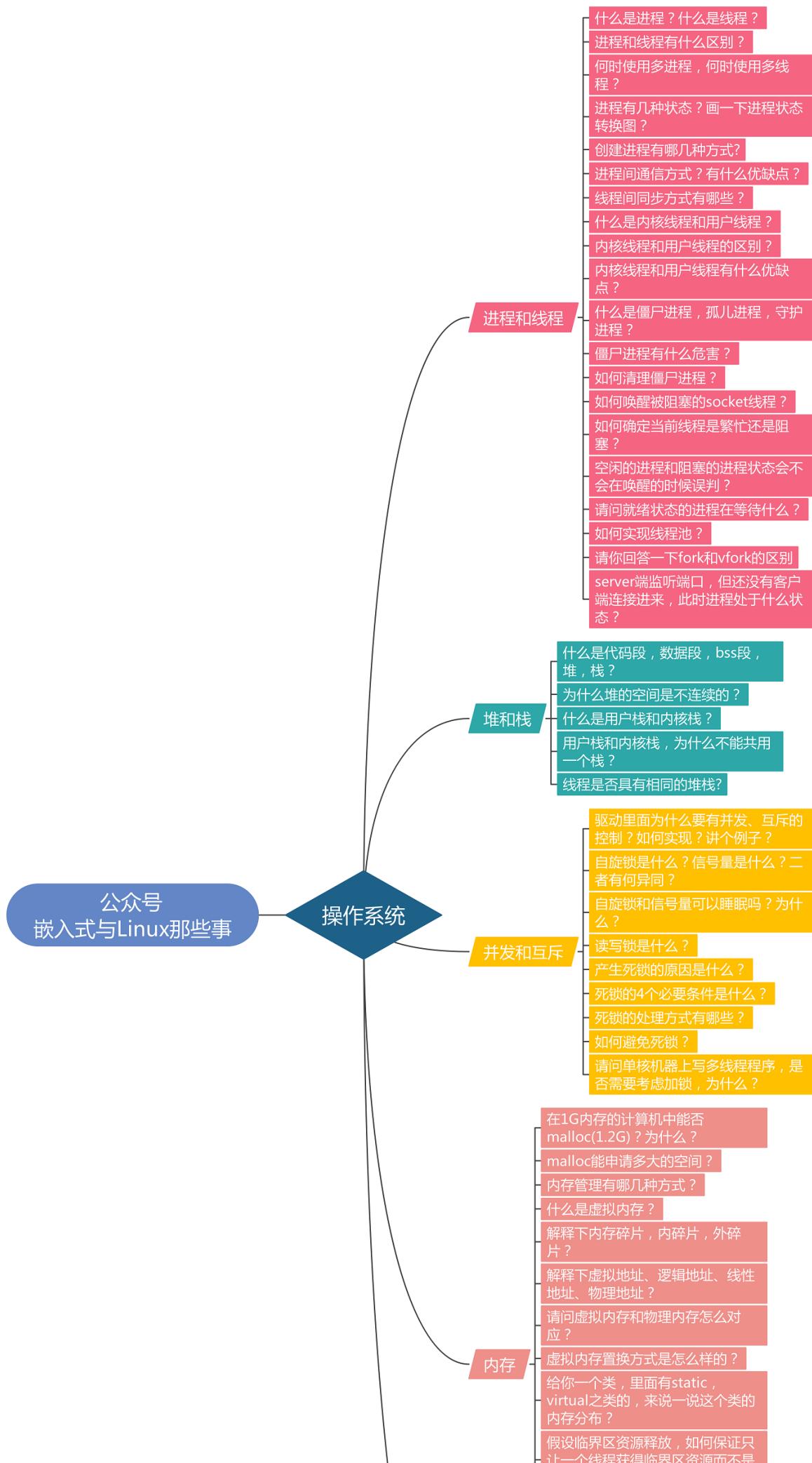
### 函数说明

从用户空间中读取数据到内核空间。

### 返回值：

成功返回0，失败返回没有拷贝成功的数据字节数。

## 操作系统



在一些线性映射的翻译中页表条目是  
都获得？  
操作系统中的缺页中断是什么？  
OS缺页置换算法如何实现的？  
系统调用是什么，你用过哪些系统  
调用，和库函数有什么区别？  
为什么要有page cache，操作系统  
怎么设计的page cache？

上下文  
上下文有哪些？怎么理解？  
为什么会有上下文这种概念？  
什么情况下进行用户态到内核态的  
切换？  
中断上下文代码中有哪些注意事  
项？  
请问线程需要保存哪些上下文，  
SP、PC、EAX这些寄存器是干嘛用  
的？

## 进程和线程

### 什么是进程？什么是线程？

进程是资源分配的基本单位，它是程序执行时的一个实例，在程序运行时创建。

线程是程序执行的最小单位，是进程的一个执行流，一个进程由多个线程组成的。

### 进程和线程有什么区别？

1. 进程是资源分配的最小单位。
2. 线程是程序执行的最小单位，也是处理器调度的基本单位，但进程不是，两者均可并发执行。
3. 进程有自己的独立地址空间，每启动一个进程，系统就会为它分配地址空间，建立数据表来维护代码段、堆栈段和数据段，这种操作非常昂贵。而线程是共享进程中的数据，使用相同的地址空间，因此，CPU切换一个线程的花费远比进程小很多，同时创建一个线程的开销也比进程小很多。
4. 线程之间的通信更方便，同一进程下的线程共享全局变量、静态变量等数据，而进程之间的通信需要以通信的方式（IPC）进行。不过如何处理好同步与互斥是编写多线程程序的难点。但是多进程程序更健壮，**多线程程序只要有一个线程死掉，整个进程也跟着死掉了，而一个进程死掉并不会对另外一个进程造成影响，因为进程都有自己独立的地址空间。**
5. 进程切换时，消耗的资源大，效率低。所以涉及到频繁的切换时，使用线程要好于进程。同样如果要求同时进行并且又要共享某些变量的并发操作，只能用线程不能用进程。
6. 执行过程：**每个独立的进程有一个程序运行的入口、顺序执行序列和程序入口。但是线程不能独立执行**，必须依存在应用程序中，由应用程序提供多个线程执行控制。
7. 线程执行开销小，但是不利于资源的管理和保护。线程适合在SMP机器（双CPU系统）上运行。进程执行开销大，但是能够很好的进行资源管理和保护，可以跨机器迁移。

### 何时使用多进程，何时使用多线程？

对资源的管理和保护要求高，不限制开销和效率时，使用多进程。

要求效率高，频繁切换时，资源的保护管理要求不是很高时，使用多线程。

### 进程有几种状态？画一下进程状态转换图？

进程可以分为五个状态，分别是：

1. 创建状态
2. 就绪状态
3. 运行状态
4. 阻塞状态
5. 终止状态

## 创建状态

一个应用程序从系统上启动，首先就是进入创建状态，需要获取系统资源创建进程管理块（PCB：Process Control Block）完成资源分配。

## 就绪状态

在创建状态完成之后，进程已经准备好，但是还未获得处理器资源，无法运行。

## 运行状态

获取处理器资源，被系统调度，开始进入运行状态。如果进程的时间片用完了就进入就绪状态。

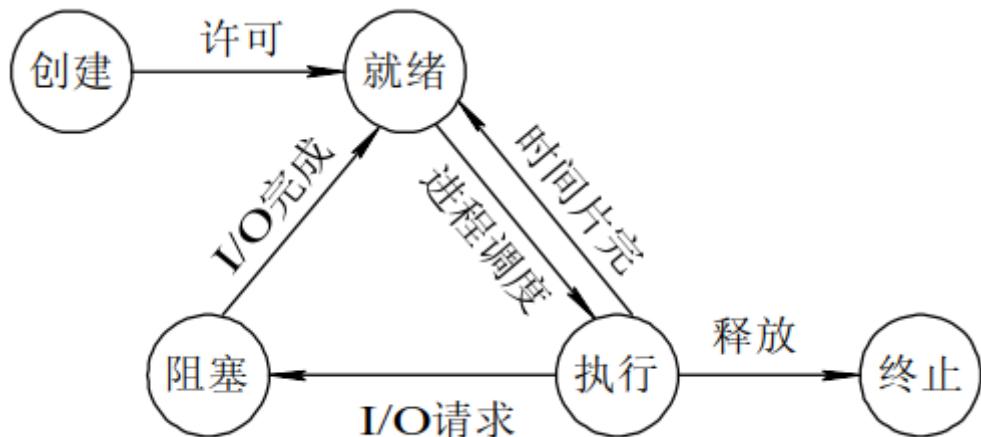
## 阻塞状态

在运行状态期间，如果进行了阻塞的操作，如耗时的I/O操作，此时进程暂时无法操作就进入了阻塞状态，在这些操作完成后就进入就绪状态。

## 终止状态

进程结束或者被系统终止，进入终止状态

## 进程的状态转换图



## 创建进程有哪几种方式？

创建进程的多种方式但凡是硬件，都需要有操作系统去管理，只要有操作系统，就有进程的概念，就需要有创建进程的方式，一些操作系统只为个应用程序设计，比如扫地机器人，一旦启动，所有的进程都已经存在。

而对于通用系统（跑很多应用程序），需要有系统运行过程中创建或撤销进程的能力，主要分为**4种形式**创建新的进程：

1. **系统初始化**（查看进程 linux中用ps命令， windows中用任务管理器，前台进程负责与用户交互，后台运行的进程与用户无关，运行在后台并且只在需要时才唤醒的进程，称为守护进程，如电子邮件、web页面、新闻、打印）
2. **一个进程在运行过程中开启了子进程**（如 nginx开启多进程， os. fork, subprocess Popen等）
3. **用户的交互式请求，而创建一个新进程**（如用户用鼠标双击任意款软件图片：q微信暴风影音等）
4. **一个批处理作业的初始化**（只在大型机的批处理系统中应用）无论哪种，新进程的创建都是由一个已经存在的进程执行了一个用于创建进程的系统调用而创建的。

## 进程间通信方式有哪些？有什么优缺点？

### 管道(pipe)

管道这种通讯方式有两种限制，**一是半双工的通信，数据只能单向流动，二是只能在具有亲缘关系的进程间使用。**进程的亲缘关系通常是指**父子进程关系**。

**流管道**s\_pipe: 去除了第一种限制，可以**双向传输**（全双工）。

管道可用于具有亲缘关系进程间的通信。**命名管道**: name\_pipe克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许**无亲缘关系**进程间的通信（命名管道也交FIFO）。

### 信号量(semaphore)

信号量是一个**计数器**，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，**防止某进程正在访问共享资源时，其他进程也访问该资源**。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

### 消息队列(message queue)

消息队列是由消息组成的链表，存放在内核中并由消息队列标识符标识。

消息队列是消息的链接表，包括Posix消息队列system V消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。

### 信号(signal)

信号是一种比较复杂的通信方式，用于**通知接收进程某个事件已经发生**。主要作为进程间以及同一进程不同线程之间的同步手段。

### 共享内存(shared memory)

共享内存就是映射一段能被其他进程所访问的内存，**这段共享内存由一个进程创建，但多个进程都可以访问**。**共享内存是最快的 IPC 方式**，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。

### 套接字(socket)

套接字也是一种进程间通信机制，与其他通信机制不同的是，它可用于**不同机器间的进程通信**。

### 各通信方式的比较和优缺点：

#### 管道

速度慢，容量有限，只有父子进程能通讯。

#### FIFO

任何进程间都能通讯，但速度慢。

#### 消息队列

容量受到系统限制，且要注意第一次读的时候，要考虑上一次没有读完数据的问题，消息队列可以不再局限于父子进程，而允许任意进程通过共享消息队列来实现进程间通信，并由系统调用函数来实现消息发送和接收之间的同步，从而使得用户在使用消息缓冲进行通信时不再需要考虑同步问题，使用方便，但是信息的复制需要额外消耗CPU的时间，**不适用于信息量大或操作频繁的场合**。此种方法不太常用。

#### 信号量

不能用来传递复杂消息，**只能用来同步**。

#### 共享内存

利用内存缓冲区直接交换信息，无须复制，快捷、信息量大是其优点。共享内存块提供了在任意数量的进程之间进行高效双向通信的机制。每个使用者都可以读取写入数据，但是所有程序之间必须达成并遵守一定的协议，以防止诸如在读取信息之前覆写内存空间等竞争状态的出现。

### 进程间通信方式的选择

PIPE和FIFO(有名管道)用来实现进程间相互发送非常短小的、频率很高的消息，这两种方式通常适用于两个进程间的通信。

共享内存用来实现进程间共享的、非常庞大的、读写操作频率很高的数据；这种方法适用于多进程间的通信。

其他考虑用socket。主要应用在分布式开发中。

### 线程间同步方法有哪些？

现在流行的进程线程同步互斥的控制机制，其实是由最原始、最基本的4种方法（临界区、互斥量、信号量和事件）实现的。

1. **临界区**：通过对多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问。在任意时刻只允许一个线程访问共享资源，如果有多个线程试图访问共享资源，那么当有一个线程进入后，其他试图访问共享资源的线程将被挂起，并一直等到进入临界区的线程离开，临界区被释放后，其他线程才可以抢占。
2. **互斥量**：为协调对一个共享资源的单独访问而设计，只有拥有互斥量的线程，才有权限去访问系统的公共资源，因为互斥量只有一个，所以能够保证资源不会同时被多个线程访问。互斥不仅能实现同一应用程序的公共资源安全共享，还能实现不同应用程序的公共资源安全共享。
3. **信号量**：为控制一个具有有限数量的用户资源而设计。它允许多个线程在同一个时刻去访问同一个资源，但一般需要限制同一时刻访问此资源的最大线程数目
4. **事件**：用来通知线程有一些事件已发生，从而启动后续任务的开始。

### 什么是内核线程和用户线程？

**用户线程**指不需要内核支持而在用户程序中实现的线程，其不依赖于操作系统核心，应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程。不需要用户态/核心态切换，速度快，操作系统内核不知道多线程的存在，因此一个线程阻塞将使得整个进程（包括它的所有线程）阻塞。由于这里的处理器时间片分配是以进程为基本单位，所以每个线程执行的时间相对减少。

**内核线程**：由操作系统内核创建和撤销。内核维护进程及线程的上下文信息以及线程切换。一个内核线程由于I/O操作而阻塞，不会影响其它线程的运行。

### 内核线程和用户线程的区别？

1. 内核支持线程是OS内核可感知的，而用户级线程是OS内核不可感知的。
2. 用户级线程的创建、撤销和调度不需要OS内核的支持，是在语言（如java）这一级处理的；而内核支持线程的创建、撤销和调度都需OS内核提供支持，而且与进程的创建、撤销和调度大体是相同的。
3. 用户级线程执行系统调用指令时将导致其所属进程被中断，而内核支持线程执行系统调用指令时，只导致该线程被中断。
4. 在只有用户级线程的系统内，CPU调度还是以进程为单位，处于运行状态的进程中的多个线程，由用户程序控制线程的轮换运行；在有内核支持线程的系统内，CPU调度则以线程为单位，由OS的线程调度程序负责线程的调度。
5. 用户级线程的程序实体是运行在用户态下的程序，而内核支持线程的程序实体则是可以运行在任何状态下的程序。

## 内核线程和用户线程有什么优缺点？

### 内核线程的优点：

1. 当有多个处理机时，一个进程的多个线程可以同时执行。

### 缺点：

1. 由内核进行调度。

### 用户线程的优点：

1. 线程的调度不需要内核直接参与，控制简单。
2. 可以在不支持线程的操作系统中实现。
3. 创建和销毁线程、线程切换代价等线程管理的代价比内核线程少得多。
4. 允许每个进程定制自己的调度算法，线程管理比较灵活。这就是必须自己写管理程序，与内核线程的区别
5. 线程能够利用的表空间和堆栈空间比内核级线程多。
6. 同一进程中只能同时有一个线程在运行，如果有一个线程使用了系统调用而阻塞，那么整个进程都会被挂起。另外，页面失效也会产生同样的问题。

### 缺点：

4. 资源调度按照进程进行，多个处理机下，同一个进程中的线程只能在同一个处理机下分时复用

## 什么是僵尸进程，孤儿进程，守护进程？

**僵尸进程**是一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵死进程。

**孤儿进程**是因为父进程异常结束了，然后被1号进程init收养。

**守护进程**是创建守护进程时有意把父进程结束，然后被1号进程init收养

**区分：**一个正常运行的子进程，如果此刻子进程退出，父进程没有及时调用wait或waitpid收回子进程的系统资源，该进程就是僵尸进程，如果系统收回了，就是正常退出，如果一个正常运行的子进程，父进程退出了但是子进程还在，该进程此刻是孤儿进程，被init收养，如果父进程是故意被杀掉，子进程做相应处理后就是守护进程

## 僵尸进程有什么危害？

在进程退出的时候，内核释放该进程所有的资源，包括打开的文件，占用的内存等。但是仍然为其保留一定的信息(包括进程号 PID，退出状态 the termination status of the process，运行时间 the amount of CPU time taken by the process 等)。直到父进程通过 wait / waitpid 来取时才释放。

如果进程不调用 wait / waitpid 的话，那么保留的那段信息就不会释放，其进程号就会一直被占用，但是系统所能使用的进程号是有限的，如果大量的产生僵死进程，将因为没有可用的进程号而导致系统不能产生新的进程。

## 如何清理僵尸进程？

僵尸进程的产生是因为父进程没有 wait() 子进程。所以如果我们自己写程序的话一定要在父进程中通过 wait() 来避免僵尸进程的产生。

当系统中出现了僵尸进程时，我们是无法通过 kill 命令把它清除掉的。但是我们可以杀死它的父进程，让它变成孤儿进程，并进一步被系统中管理孤儿进程的进程收养并清理。

## 如何唤醒被阻塞的socket线程？

### 同步阻塞

等待锁的释放

### 等待阻塞

1. 使用Thread.sleep造成的阻塞:时间结束后自动进入RUNNABLE状态
2. 使用Thread.wait造成的阻塞:使用Thread.notify或者Thread.notifyAll唤醒
3. 使用Thread.join造成的阻塞:等待上一个线程执行完后自动进入RUNNABLE状态
4. 使用Thread.suspend造成的阻塞:使用Thread.resume唤醒
5. 使用LockSupport.park造成的阻塞:使用LockSupport.unpark唤醒
6. 使用LockSupport.parkNanos造成的阻塞:指定时间结束后, 自动唤醒
7. 使用LockSupport.parkUntil造成的阻塞:到达指定的时间, 自动唤醒

## 如何确定当前线程是繁忙还是阻塞？

使用ps命令查看

### 空闲的进程和阻塞的进程状态会不会在唤醒的时候误判？

不会。每个进程有个进程控制块PCB，两种状态的进程分别处于两种队列。唤醒应该是找阻塞队列的进程。

### 请问就绪状态的进程在等待什么？

被调度使用cpu的运行权

## 如何实现线程池？

1. 设置一个生产者消费者队列, 作为临界资源
2. 初始化n个线程, 并让其运行起来, 加锁去队列取任务运行
3. 当任务队列为空的时候, 所有线程阻塞
4. 当生产者队列来了一个任务后, 先对队列加锁, 把任务挂在到队列上, 然后使用条件变量去通知阻塞中的一个线程

## 请你回答一下fork和vfork的区别？

**fork的基础知识：** fork:创建一个和当前进程映像一样的进程可以通过fork( )系统调用:

```

1 #include <sys/types.h>
2 #include <unistd.h>
3 pid_t fork(void);

```

成功调用fork( )会创建一个新的进程, 它几乎与调用fork( )的进程一模一样, 这两个进程都会继续运行。在子进程中, 成功的fork( )调用会返回0。在父进程中fork( )返回子进程的pid。如果出现错误, fork( )返回一个负值。最常见的fork( )用法是创建一个新的进程, 然后使用exec( )载入二进制映像, 替换当前进程的映像。这种情况下, 派生(fork)了新的进程, 而这个子进程会执行一个新的二进制可执行文件的映像。这种“派生加执行”的方式是很常见的。在早期的Unix系统中, 创建进程比较原始。当调用fork时, 内核会把所有的内部数据结构复制一份, 复制进程的页表项, 然后把父进程的地址空间中的内容逐页的复制到子进程的地址空间中。但从内核角度来说, 逐页的复制方式是十分耗时的。现代的Unix系统采取了更多的优化, 例如Linux, 采用了写时复制的方法, 而不是对父进程空间进程整体复制。

**vfork的基础知识：**在实现写时复制之前, Unix的设计者们就一直很关注在fork后立刻执行exec所造成的地址空间的浪费。BSD的开发者们在3.0的BSD系统中引入了vfork( )系统调用。

```

1 #include <sys/types.h>
2 #include <unistd.h>
3 pid_t vfork(void);

```

除了子进程必须要立刻执行一次对exec的系统调用，或者调用\_exit( )退出，对vfork( )的成功调用所产生的结果和fork( )是一样的。vfork( )会挂起父进程直到子进程终止或者运行了一个新的可执行文件的映像。通过这样的方式，vfork( )避免了地址空间的按页复制。在这个过程中，父进程和子进程共享相同的地址空间和页表项。实际上vfork( )只完成了一件事：复制内部的内核数据结构。因此，子进程也就不能修改地址空间中的任何内存。vfork( )是一个历史遗留产物，Linux本不应该实现它。需要注意的是，即使增加了写时复制，vfork( )也要比fork( )快，因为它没有进行页表项的复制。然而，写时复制的出现减少了对于替换fork( )争论。实际上，直到2.2.0内核，vfork( )只是一个封装过的fork( )。因为对vfork( )的需求要小于fork( )，所以vfork( )的这种实现方式是可行的。

**写时复制：** Linux采用了写时复制的方法，以减少fork时对父进程空间整体复制带来的开销。写时复制是一种采取了惰性优化方法来避免复制时的系统开销。它的前提很简单：如果有多个进程要读取它们自己的那部分资源的副本，那么复制是不必要的。每个进程只要保存一个指向这个资源的指针就可以了。只要没有进程要去修改自己的“副本”，就存在着这样的幻觉：每个进程好像独占那个资源。从而就避免了复制带来的负担。如果一个进程要修改自己的那份资源“副本”，那么就会复制那份资源，并把复制的那份提供给进程。不过其中的复制对进程来说是透明的。这个进程就可以修改复制后的资源了，同时其他的进程仍然共享那份没有修改过的资源。所以这就是名称的由来：在写入时进行复制。写时复制的主要好处在于：如果进程从来就不需要修改资源，则不需要进行复制。惰性算法的好处就在于它们尽量推迟代价高昂的操作，直到必要的时刻才会去执行。在使用虚拟内存的情况下，写时复制(Copy-On-Write)是以页为基础进行的。所以，只要进程不修改它全部的地址空间，那么就不必复制整个地址空间。在fork( )调用结束后，父进程和子进程都相信它们有一个自己的地址空间，但实际上它们共享父进程的原始页，接下来这些页又可以被其他的父进程或子进程共享。写时复制在内核中的实现非常简单。与内核页相关的数据结构可以被标记为只读和写时复制。如果有进程试图修改一个页，就会产生一个缺页中断。内核处理缺页中断的方式就是对该页进行一次透明复制。这时会清除页面的COW属性，表示着它不再被共享。现代的计算机系统结构中都在内存管理单元(MMU)提供了硬件级别的写时复制支持，所以实现是很容易的。在调用fork( )时，写时复制是有很大优势的。因为大量的fork之后都会跟着执行exec，那么复制整个父进程地址空间中的内容到子进程的地址空间完全是在浪费时间：如果子进程立刻执行一个新的二进制可执行文件的映像，它先前的地址空间就会被交换出去。写时复制可以对这种情况进行优化。

### fork和vfork的区别：

1. fork( )的子进程拷贝父进程的数据段和代码段；vfork( )的子进程与父进程共享数据段
2. fork( )的父子进程的执行次序不确定；vfork( )保证子进程先运行，在调用exec或exit之前与父进程数据是共享的，在它调用exec或exit之后父进程才可能被调度运行。
3. vfork( )保证子进程先运行，在它调用exec或exit之后父进程才可能被调度运行。如果在调用这两个函数之前子进程依赖于父进程的进一步动作，则会导致死锁。
4. 当需要改变共享数据段中变量的值，则拷贝父进程。

### server端监听端口，但还没有客户端连接进来，此时进程处于什么状态？

最普通的Server模型，则处于阻塞状态；如果使用IO复用中epoll、select等，则处于运行状态。



## 堆和栈

### 什么是代码段，数据段，bss段，堆，栈？

**代码段**：代码段通常用来存放程序**执行代码**的一块区域。这部分区域的大小在程序运行前就已经确定了，通常这块内存区域属于**只读**，有些架构也允许可写，在代码段中也有可能包含以下只读的常数变量，例如字符串常量等。程序段为程序代码在内存中映射一个程序可以在内存中有多个副本。

**数据段**：数据段通常用来存放程序中**已初始化的全局变量和已初始化为非0的静态变量**的一块内存区域，属于静态内存分配。直观理解就是C语言程序中的全局变量（注意：**全局变量才算是程序的数据，局部变量不算程序的数据，只能算是函数的数据**）

**BSS段**：bss段通常是指用来存放程序中**未初始化的全局变量和未初始化的静态变量或者初始化为0的静态变量**一块区域。bss英文Block started by symbol，bss属于静态内存分配。bss段的**特点就是被初始化为0**，**bss段本质上也是属于数据段**，bss段就是被初始化为0的数据段。注意区分。

**数据段 (.data) 和bss段的区别和联系**：二者本来没有本质区别，都是用来存放C程序中的全局变量的。区别在于把显示初始化为非零的全局变量存在.data段中，而把**显式初始化为0或者并未显式初始化 (C语言规定未显式初始化的全局变量值默认为0)**的全局变量存在bss段。

**堆**：堆是用来存放进程中被动态分配的内存段，它的大小并不固定，可动态扩张或缩减。当进程调用malloc等函数分配内存时，新分配的内存就被动态分配到堆上，当利用free等函数释放内存时，被释放的内存从堆中被剔除。

**栈**：栈又称堆栈，是用户存放程序临时创建的变量，也就是我们函数{}中定义的变量，但不包括static声明的变量，static意味着在数据段中存放变量。除此之外，在函数被调用时，其参数也会被压入发起调用的进程栈中，并且待到调用结束后，函数的返回值也会被存放回栈中，由于栈的先进后出特点，所以栈特别方便用来保存、恢复调用现场。从这个意义上讲，我们可以把堆栈看成一个寄存，交换临时数据的内存区。

```

1 #include <stdio.h>
2 int a = 0; // 数据段
3 char *p1; // BSS段
4
5 int main()
6 {
7
8     int b; // 栈
9     char s[] = "abc"; // 栈
10    char *p2; // 栈
11    char *p3 = "123456"; // 123456\0在常量区，p3在栈上。
12    static int c = 0; // BSS段
13    class c1 = new class(); //new出的对象就在堆区
14    strcpy(p1, "123456"); //123456\0放在常量区，编译器可能会将它与p3所指向的"123456"优化成一个地方。
15    return 0;
16 }
```

## 为什么堆的空间是不连续的？

堆包含一个链表来维护已用和空闲的内存块。在堆上新分配（用 new 或者 malloc）内存是从空闲的内存块中找到一些满足要求的合适块。所以可能让人觉得只要有很多不连续的零散的小区域，只要总数达到申请的内存块，就可以分配。

但事实上是不行的，这又让人觉得是不是零散的内存块不能连接成一个大的空间，而必须要一整块连续的内存空间才能申请成功呢。

申请和释放许多小的块可能会产生如下状态：在已用块之间存在很多小的空闲块。进而申请大块内存失败，虽然空闲块的总和足够，但是**空闲的小块是零散的，不连续的**，不能满足申请的大小，这叫做“堆碎片”。

当旁边有空闲块的已用块被释放时，新的空闲块会与相连的空闲块合并成一个大的空闲块，这样就可以有效的减少“堆碎片”的产生。

堆分配的空间在**逻辑地址（虚拟地址）上是连续的，但在物理地址上是不连续的**（因为采用了页式内存管理，windows下有段机制、分页机制），如果逻辑地址空间上已经没有一段连续且足够大的空间，则分配内存失败。

## 什么是用户栈和内核栈？

### 内核栈

内存中属于操作系统空间的一块区域。

### 作用

1. 保存中断现场，对于嵌套中断，被中断程序的现场信息一次压入系统栈，中断返回时逆序弹出。
2. 保存操作系统程序相互调用的参数，返回值，以及函数的局部变量。

### 用户栈

用户进程空间的一块区域，用于保存用户空间子程序间调用的参数，返回值以及局部变量。

## 用户栈和内核栈，为什么不能共用一个栈？

1. 如果只用系统栈，系统栈一般大小有限，用户程序调用次数可能很多。如果有16个优先级，那么系统栈一般大小为15（只需保存15个低优先级中断，另一个高优先级中断在运行）用户程序调用次数很多，那样15次子程序调用以后的子程序的参数，返回值，局部变量就不能保存，用户程序也就不能正常运行。
2. 如果只用用户栈，系统程序需要在某种保护下运行，而用户栈在用户空间不能提供相应的保护措施。

## 线程是否具有相同的堆栈？

真正的程序执行都是线程来完成的，程序启动的时候操作系统就帮你创建了一个主线程。

每个线程有自己的堆栈。



## 并发和互斥

### 驱动里面为什么要有并发、互斥的控制？如何实现？讲个例子？

并发，指的是多个执行单元同时、并行被执行，而并发的执行单元对共享资源（硬件资源和软件上的全局变量、静态变量等）的访问则很容易导致竞态。

解决竞态问题的途径是保证对共享资源的互斥访问，所谓互斥访问就是指一个执行单元在访问共享资源的时候，其他的执行单元都被禁止访问。

访问共享资源的代码区域被称为临界区，临界区需要以某种互斥机制加以保护。中断屏蔽，原子操作，自旋锁，和信号量都是linux设备驱动中可采用的互斥途径。

### 自旋锁是什么？信号量是什么？二者有何异同？

#### 自旋锁

自旋锁，顾名思义，我们可以把他理解成厕所门上的一把锁。这个厕所门只有一把钥匙，当我们进去时，把钥匙取下来，进去后反锁。那么当第二个人想进来，必须等我们出去后才可以。当第二个人在外面等待时，可能会一直等待在门口转圈。

我们的自旋锁也是这样，自旋锁只有锁定和解锁两个状态。当我们进入拿上钥匙进入厕所，这就相当于自旋锁锁定的状态，期间谁也不可以进来。当第二个人想要进来，这相当于线程B想要访问这个共享资源，但是目前不能访问，所以线程B就一直在原地等待，一直查询是否可以访问这个共享资源。当我们从厕所出来后，这个时候就“解锁”了，只有再这个时候线程B才能访问。

假如，在厕所的人待的时间太长怎么办？外面的人一直等待吗？如果换做是我们，肯定不会这样，简直浪费时间，可能我们会寻找其他方法解决问题。自旋锁也是这样的，如果线程A持有**自旋锁时间过长，显然会浪费处理器的时间，降低了系统性能**。我们知道CPU最伟大的发明就在于多线程操作，这个时候让线程B在这里傻傻的不知道还要等待多久，显然是不合理的。因此，**如果自旋锁只适合短期持有**，如果遇到需要长时间持有的情况，我们就要换一种方式了（互斥体）。

#### 信号量

信号量和自旋锁有些相似，不同的是信号量会发出一个信号告诉你还需要等多久。因此，不会出现傻傻等待的情况。比如，有100个停车位的停车场，门口电子显示屏上实时更新的停车数量就是一个信号量。这个停车的数量就是一个信号量，他告诉我们是否可以停车进去。当有车开进去，信号量加一，当有车开出来，信号量减一。

比如，厕所一次只能让一个人进去，当A在里面的时候，B想进去，如果是自旋锁，那么B就会一直在门口傻傻等待。如果是信号量，A就会给B一个信号，你先回去吧，我出来了叫你。这就是一个信号量的例子，B听到A发出的信号后，可以先回去睡觉，等待A出来。

因此，信号量显然可以提高系统的执行效率，避免了许多无用功。

#### 区别

1. 由于争用信号量的进程在等待锁重新变为可用时会睡眠，所以信号量适用于锁会被长时间持有的情况。
2. 相反，锁被短时间持有时，使用信号量就不太适宜了，因为睡眠引起的耗时可能比锁被占用的全部时间还要长。
3. 由于执行线程在锁被争用时会睡眠，所以只能在进程上下文中才能获取信号量锁，因为在中断上下文中（使用自旋锁）是不能进行调度的。
4. 你可以在持有信号量时去睡眠（当然你也可能并不需要睡眠），因为当其它进程试图获得同一信号量时不会因此而死锁，（因为该进程也只是去睡眠而已，而你最终会继续执行的）。
5. 在你占用信号量的同时不能占用自旋锁，因为在你等待信号量时可能会睡眠，而在持有自旋锁时是不允许睡眠的。

6. 信号量锁保护的临界区可包含可能引起阻塞的代码，而自旋锁则绝对要避免用来保护包含这样代码的临界区，因为阻塞意味着要进行进程的切换，如果进程被切换出去后，另一进程企图获取本自旋锁，死锁就会发生。
7. 信号量不同于自旋锁，它不会禁止内核抢占（自旋锁被持有时，内核不能被抢占），所以持有信号量的代码可以被抢占，这意味着信号量不会对调度的等待时间带来负面影响。

## 自旋锁和信号量可以睡眠吗？为什么？

自旋锁不能睡眠，信号量可以。

### 原因

自旋锁自旋锁禁止处理器抢占；而信号量不禁止处理器抢占。

基于这个原因，如果自旋锁在锁住以后进入睡眠，由于不能进行处理器抢占，其他系统进程将都不能获得CPU而运行，因此不能唤醒睡眠的自旋锁，因此系统将不响应任何操作（除了中断或多核的情况，下面会讨论）。而信号量在临界区睡眠后，其他进程可以用抢占的方式继续运行，从而可以实现内存拷贝等功能而使得睡眠的信号量程序由于获得了等待的资源而被唤醒，从而恢复了正常的代码运行。

当然，自旋锁的睡眠的情况包含考虑多核CPU和中断的因素。自旋锁睡眠时，只是当前CPU的睡眠以及当前CPU的禁止处理器抢占，所以，如果存在多个CPU，那么其他活动的CPU可以继续运行使操作系统功能正常，并有可能完成相应工作而唤醒睡眠了的自旋锁，从而没有造成系统死机；自旋锁睡眠时，如果允许中断处理，那么中断的代码是可以正常运行的，但是中断通常不会唤醒睡眠的自旋锁，因此系统仍然运行不正常。

## 自旋锁和信号量可以用于中断中吗？

信号量不能用于中断中，因为信号量会引起休眠，**中断不能休眠**。

自旋锁可以用于中断。在获取锁之前一定要**先禁止本地中断**（也就是**本CPU中断**，对于多核SOC来说会有多个CPU核），否则可能导致锁死现象的发生。

## 读写锁是什么？

当临界区的一个文件可以被**同时读取**，但是**并不能被同时读和写**。如果一个线程在读，另一个线程在写，那么很可能会读取到错误的**不完整的数据**。读写自旋锁是可以**允许对临界区的共享资源进行并发读操作的**。但是**并不允许多个线程并发读写操作**。

## 产生死锁的原因是什么？

**多个并发进程因争夺系统资源而产生相互等待的现象**。即：一组进程中的每个进程都在等待某个事件发生，而只有这组进程中的其他进程才能触发该事件，这就称这组进程发生了死锁。

### 产生死锁的本质原因为：

1. 系统资源有限。
2. 进程推进顺序不合理。

## 死锁的4个必要条件是什么？

1. **互斥**：某种资源一次只允许一个进程访问，即该资源一旦分配给某个进程，其他进程就不能再访问，直到该进程访问结束。
2. **占有且等待**：一个进程本身占有资源（一种或多种），同时还有资源未得到满足，正在等待其他进程释放该资源。
3. **不可抢占**：别人已经占有了某项资源，你不能因为自己也需要该资源，就去把别人的资源抢过来。
4. **循环等待**：存在一个进程链，使得每个进程都占有下一个进程所需的至少一种资源。

当以上四个条件均满足，必然会造成死锁，发生死锁的进程无法进行下去，它们所持有的资源也无法释放。这样会导致CPU的吞吐量下降。所以死锁情况是会浪费系统资源和影响计算机的使用性能的。那么，解决死锁问题就是相当有必要的了。

## 死锁的处理方式有哪些？

死锁的处理方式主要从**预防死锁、避免死锁、检测与解除死锁**这四个方面来进行处理。

### 预防死锁

1. 资源一次性分配：（破坏请求和保持条件）
2. 可剥夺资源：即当某进程新的资源未满足时，释放已占有的资源（破坏不可剥夺条件）
3. 资源有序分配法：系统给每类资源赋予一个编号，每一个进程按编号递增的顺序请求资源，释放则相反（破坏环路等待条件）

### 避免死锁

预防死锁的几种策略，会严重地损害系统性能。因此在避免死锁时，要施加较弱的限制，从而获得较满意的系统性能。由于在避免死锁的策略中，允许进程动态地申请资源。因而，系统在进行资源分配之前预先计算资源分配的安全性。若此次分配不会导致系统进入不安全状态，则将资源分配给进程；否则，进程等待。其中最具有代表性的避免死锁算法是银行家算法。

### 检测死锁

首先为每个进程和每个资源指定一个唯一的号码；然后建立资源分配表和进程等待表

### 解除死锁：

当发现有进程死锁后，便应立即把它从死锁状态中解脱出来，常采用的方法有：

1. 剥夺资源：从其它进程剥夺足够数量的资源给死锁进程，以解除死锁状态；
2. 撤消进程：可以直接撤消死锁进程或撤消代价最小的进程，直至有足够的资源可用，死锁状态消除为止；所谓代价是指优先级、运行代价、进程的重要性和价值等。

## 如何避免死锁？

在有些情况下死锁是可以避免的。三种用于避免死锁的技术：

### 加锁顺序（线程按照一定的顺序加锁）

**加锁时限**（线程尝试获取锁的时候加上一定的时限，超过时限则放弃对该锁的请求，并释放自己占有的锁）

### 死锁检测

### 加锁顺序

当多个线程需要相同的一些锁，但是按照不同的顺序加锁，死锁就很容易发生。

如果能确保所有的线程都是按照相同的顺序获得锁，那么死锁就不会发生。看下面这个例子：

```

1 Thread 1:
2   lock A
3   lock B
4
5 Thread 2:
6   wait for A
7   lock C (when A locked)
8
9 Thread 3:
10  wait for A
11  wait for B
12  wait for C

```

如果一个线程（比如线程3）需要一些锁，那么它必须按照确定的顺序获取锁。它只有获得了从顺序上排在前面的锁之后，才能获取后面的锁。

例如，线程2和线程3只有在获取了锁A之后才能尝试获取锁C(译者注：获取锁A是获取锁C的必要条件)。因为线程1已经拥有了锁A，所以线程2和3需要一直等到锁A被释放。然后在它们尝试对B或C加锁之前，必须成功地对A加了锁。

按照顺序加锁是一种有效的死锁预防机制。但是，这种方式需要你事先知道所有可能会用到的锁(译者注：并对这些锁做适当的排序)，但总有些时候是无法预知的。

### 加锁时限

另外一个可以避免死锁的方法是在尝试获取锁的时候加一个超时时间，这也就意味着在尝试获取锁的过程中若超过了这个时限该线程则放弃对该锁请求。若一个线程没有在给定的时限内成功获得所有需要的锁，则会进行回退并释放所有已经获得的锁，然后等待一段随机的时间再重试。这段随机的等待时间让其它线程有机会尝试获取相同的这些锁，并且让该应用在没有获得锁的时候可以继续运行(译者注：加锁超时后可以先继续运行干点其它事情，再回头来重复之前加锁的逻辑)。

以下是一个例子，展示了两个线程以不同的顺序尝试获取相同的两个锁，在发生超时后回退并重试的场景：

```

1 Thread 1 locks A
2 Thread 2 locks B
3
4 Thread 1 attempts to lock B but is blocked
5 Thread 2 attempts to lock A but is blocked
6
7 Thread 1's lock attempt on B times out
8 Thread 1 backs up and releases A as well
9 Thread 1 waits randomly (e.g. 257 millis) before retrying.
10
11 Thread 2's lock attempt on A times out
12 Thread 2 backs up and releases B as well
13 Thread 2 waits randomly (e.g. 43 millis) before retrying.

```

在上面的例子中，线程2比线程1早200毫秒进行重试加锁，因此它可以先成功地获取到两个锁。这时，线程1尝试获取锁A并且处于等待状态。当线程2结束时，线程1也可以顺利的获得这两个锁（除非线程2或者其它线程在线程1成功获得两个锁之前又获得其中的一些锁）。

需要注意的是，由于存在锁的超时，所以我们不能认为这种场景就一定是出现了死锁。也可能是因为获得了锁的线程（导致其它线程超时）需要很长的时间去完成它的任务。

此外，如果有非常多的线程同一时间去竞争同一批资源，就算有超时和回退机制，还是可能会导致这些线程重复地尝试但却始终得不到锁。如果只有两个线程，并且重试的超时时间设定为0到500毫秒之间，这种现象可能不会发生，但是如果是10个或20个线程情况就不同了。因为这些线程等待相等的重试时间的概率就高的多（或者非常接近以至于会出现问题）。

这种机制存在一个问题，在Java中不能对synchronized同步块设置超时时间。你需要创建一个自定义锁，或使用Java5中java.util.concurrent包下的工具。写一个自定义锁类不复杂，但超出了本文的内容。后续的Java并发系列会涵盖自定义锁的内容。

## 死锁检测

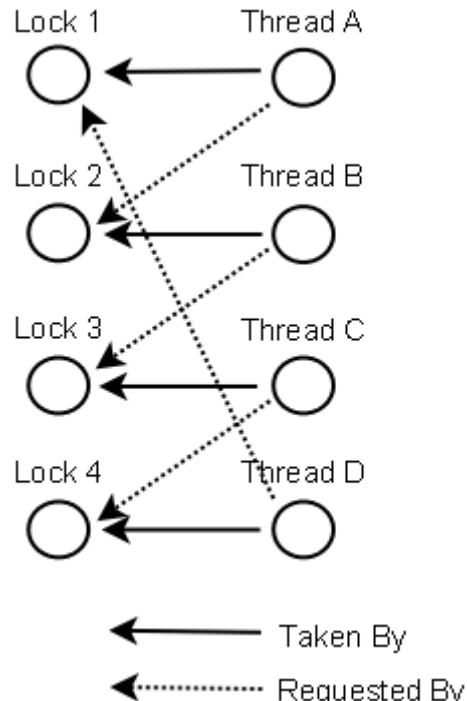
死锁检测是一个更好的死锁预防机制，它主要是针对那些不可能实现按序加锁并且锁超时也不可行的场景。

每当一个线程获得了锁，会在线程和锁相关的数据结构中（map、graph等等）将其记下。除此之外，每当有线程请求锁，也需要记录在这个数据结构中。

当一个线程请求锁失败时，这个线程可以遍历锁的关系图看看是否有死锁发生。例如，线程A请求锁7，但是锁7这个时候被线程B持有，这时线程A就可以检查一下线程B是否已经请求了线程A当前所持有的锁。如果线程B确实有这样的请求，那么就是发生了死锁（线程A拥有锁1，请求锁7；线程B拥有锁7，请求锁1）。

当然，死锁一般要比两个线程互相持有对方的锁这种情况要复杂的多。线程A等待线程B，线程B等待线程C，线程C等待线程D，线程D又在等待线程A。线程A为了检测死锁，它需要递进地检测所有被B请求的锁。从线程B所请求的锁开始，线程A找到了线程C，然后又找到了线程D，发现线程D请求的锁被线程A自己持有着。这是它就知道发生了死锁。

下面是一幅关于四个线程（A,B,C和D）之间锁占有和请求的关系图。像这样的数据结构就可以被用来检测死锁。



那么当检测出死锁时，这些线程该做些什么呢？

一个可行的做法是释放所有锁，回退，并且等待一段随机的时间后重试。这个和简单的加锁超时类似，不一样的是只有死锁已经发生了才回退，而不会是因为加锁的请求超时了。虽然有回退和等待，但是如果大量的线程竞争同一批锁，它们还是会重复地死锁（编者注：原因同超时类似，不能从根本上减轻竞争）。

一个更好的方案是给这些线程设置优先级，让一个（或几个）线程回退，剩下的线程就像没发生死锁一样继续保持着它们需要的锁。如果赋予这些线程的优先级是固定不变的，同一批线程总是会拥有更高的优先级。为避免这个问题，可以在死锁发生的时候设置随机的优先级。

## 请问单核机器上写多线程程序，是否需要考虑加锁，为什么？

在单核机器上写多线程程序，仍然需要线程锁。因为线程锁通常用来实现线程的同步和通信。在单核机器上的多线程程序，仍然存在线程同步的问题。因为在抢占式操作系统中，通常为每个线程分配一个时间片，当某个线程时间片耗尽时，操作系统会将其挂起，然后运行另一个线程。如果这两个线程共享某些数据，不使用线程锁的前提下，可能会导致共享数据修改引起冲突。



## 内存

### 在1G内存的计算机中能否malloc(1.2G)? 为什么?

malloc能够申请的空间大小与物理内存的大小没有直接关系，仅与程序的虚拟地址空间相关。

程序运行时，堆空间只是程序向操作系统申请划出来的一大块虚拟地址空间。应用程序通过malloc申请空间，得到的是在虚拟地址空间中的地址，之后程序运行所提供的物理内存是由操作系统完成的。

本题要申请空间的大小为  $1.2\text{G} = 2^{30} \times 1.2 \text{ Byte}$ ， $1.2\text{G} = (1024 * 1024 * 1024) * 1.2\text{Byte}$  转换为十六进制约为 4CCC CCC，这个数值已经超过了 int 类型的表示范围，但还在 unsigned 的表示范围。幸运的是 malloc 函数要求的参数为 unsigned。见下面的示例代码。

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     char*p;
6     constunsigned k= 1024*1024*1024*1.2;
7     printf("%x\n",k);
8     p= (char *)malloc( k );
9     if( p!=NULL )
10         printf("OK");
11     else
12         printf("error");
13     return0;
14 }
```

### malloc能申请多大的空间？

malloc能够申请的空间到底能达到多大，还真是一个比较复杂的问题。想知道在一台机器上malloc能够申请的最大空间到底是多少，可以使用下面的程序进行测试。

```

1 #include <stdio.h>
2 #include <stdlib.h>
```

```

3  unsigned maximum = 1024*1024*1024;
4  int main(int argc, char *argv[])
5  {
6      unsignedblocksize[] = {1024*1024, 1024, 1};
7      inti, count;
8      void* block;
9      for(i=0; i<sizeof(blocksize)/sizeof(unsigned); i++)
10     {   for( count = 1; ;count++ )
11         {   block = malloc( maximum +blocksize[i]*count );
12             if( block!=NULL ) {
13                 maximum= maximum + blocksize[i]*count;
14                 free(block );
15             }else {
16                 break;
17             }
18         }
19     }
20     printf("maximumalloc size = %u bytes\n", maximum);
21     return0;
22 }

```

在当前正在使用的Windows环境中，可申请的最大空间超过1.9G。实际上，具体的数值会受到操作系统版本、程序本身的大小、用到的动态/共享库数量、大小、程序栈数量、大小等的影响，甚至每次运行的结果都可能存在差异，因为有些操作系统使用了一种叫做随机地址分布的技术，使得进程的堆空间变小。感兴趣的读者可以去研究操作系统中的相关内容。

## 内存管理有哪几种方式？

常见的内存管理方式有**块式管理**、**页式管理**、**段式管理和段页式管理**。最常用的是段页式管理。

### 块式管理

把主存分为一大块一大块的，当所需的程序片断不在主存时就分配一块主存空间，把程序片断载入主存，就算所需的程序片段只有几个字节，也只能把这一块分配给它。这样会造成**很大的浪费**，平均浪费了50%的内存空间，但是易于管理。

### 页式管理

用户程序的地址空间被划分成若干个固定大小的区域，这个区域被称为“页”，相应地，内存空间也被划分为若干个物理块，页和块的大小相等。可将用户程序的任一页放在内存的任一块中，从而实现了离散分配。这种方式的**优点是页的大小是固定的，因此便于管理；缺点是页长与程序的逻辑大小没有任何关系**。这就导致在某个时刻一个程序可能只有一部分在主存中，而另一部分则在辅存中。这不利于编程时的独立性，并给换入换出处理、存储保护和存储共享等操作造成麻烦。

### 段式管理

**段是按照程序的自然分界划分的并且长度可以动态改变的区域**。使用这种方式，程序员可以把子程序、操作数和不同类型的数据和函数划分到不同的段中。这种方式将用户程序地址空间**分成若干个大小不等的段**，每段可以定义一组**相对完整的逻辑信息**。存储分配时，以段为单位，段与段在内存中可以**不相邻接**，也实现了**离散分配**。

分页对程序员而言是不可见的，而分段通常对程序员而言是可见的，因而分段为组织程序和数据提供了方便，但是对程序员的要求也比较高。

分段存储主要有如下优点：

1. 段的逻辑独立性不仅使其易于编译、管理、修改和保护，也便于多道程序共享。
2. 段长可以根据需要动态改变，允许自由调度，以便有效利用主存空间。
3. 方便分段共享、分段保护、动态链接、动态增长。

分段存储的缺点如下：

1. 由于段的大小不固定，因此**存储管理比较麻烦**。
2. 会生成**段内碎片**，这会造成**存储空间利用率降低**。而且段式存储管理比页式存储管理方式需要更多的硬件支持。

正是由于页式管理和段式管理都有各种各样的缺点，因此，为了把这两种存储方式的优点结合起来，新引入了段页式管理。

## 段页式管理

段页式存储组织是分段式和分页式结合的存储组织方法，这样可充分利用分段管理和分页管理的优点。

1. **用分段方法来分配和管理虚拟存储器**。程序的地址空间按逻辑单位分成基本独立的段，而每一段有自己的段名，再把每段分成固定大小的若干页
2. **用分页方法来分配和管理内存**，即把整个主存分成与上述页大小相等的存储块，可装入作业的任何一页。程序对内存的调入或调出是按页进行的，但它又可按段实现共享和保护

## 什么是虚拟内存？

虚拟内存简称虚存，是计算机系统内存管理的一种技术。它是相对于物理内存而言的，可以理解为“假的”内存。它使得**应用程序认为它拥有连续可用的内存**（一个连续完整的地址空间），**允许程序员编写并运行比实际系统拥有的内存大得多的程序**，这使得许多大型软件项目能够在具有有限内存资源的系统上实现。而实际上，它通常被分割成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。

相比实存，虚存有以下好处：

1. **扩大了地址空间**。无论是段式虚存，还是页式虚存，或是段页式虚存，**寻址空间都比实存大**。
2. **内存保护**。每个进程运行在各自的虚拟内存地址空间，互相不能干扰对方。另外，虚存还对特定的内存地址提供写保护，可以防止代码或数据被恶意篡改。
3. **公平分配内存**。采用了虚存之后，每个进程都相当于有同样大小的虚存空间。
4. 当进程需要通信时，可采用虚存共享的方式实现不过，使用虚存也是有代价的，主要表现在以下几个方面。
  - (1) 虚存的管理需要建立很多数据结构，这些数据结构要占用额外的内存。
  - (2) 虚拟地址到物理地址的转换，增加了指令的执行时间。
  - (3) 页面的换入换出需要磁盘IO，这是很耗时间的
  - (4) 如果一页中只有一部分数据，会浪费内存。

## 解释下内存碎片，内碎片，外碎片？

内存碎片是由于多次进行内存分配造成的，当进行内存分配时，内存格式一般为：（用户使用段）（空白段）（用户使用段），当空白段很小的时候，可能不能提供给用户足够多的空间，如夹在中间的空白段的大小为5，而用户需要的内存大小为6，这样会产生很多的间隙，造成使用效率下降，这些很小的空隙叫碎片。

**内碎片**：分配给程序的存储空间没有用完，有一部分是程序不使用，但其他程序也没法用的空间。内碎片是处于区域内部或页面内部的存储块，占有这些区域或页面的进程并不使用这个存储块，而在进程占有这块存储块时，系统无法利用它，直到进程释放它，或进程结束时，系统才有可能利用这个存储块。

**外碎片**：空间太小，小到无法给任何程序分配（不属于任何进程）的存储空间。外部碎片是出于任何已分配区域或页面外部的空闲存储块，这些存储块的总和可以满足当前申请的长度要求，但是它们的地址不连续或其他原因，使得系统无法满足当前申请。

内碎片和外碎片是一对矛盾体，一种特定的内存分配算法，很难同时解决好内碎片和外碎片的问题，只能根据应用特点进行取舍

## 解释下虚拟地址、逻辑地址、线性地址、物理地址？

虚拟地址是指由程序产生的由**段选择符**和**段内偏移地址**组成的地址。这两部分组成的地址**并没有直接访问物理内存**，而是通过分段地址的变换处理后才会对应到相应的物理内存地址。

逻辑地址指由程序产生的段内偏移地址。有时直接把逻辑地址当成虚拟地址，两者并没有明确的界限。

线性地址是指**虚拟地址到物理地址变换之间的中间层**，是处理器可寻址的内存空间（称为线性地址空间）中的地址。程序代码会产生逻辑地址，或者说是段中的偏移地址，加上相应段基址就生成了一个线性地址。如果启用了分页机制，那么线性地址可以再经过变换产生物理地址。若没有采用分页机制，那么线性地址就是物理地址。

物理地址是指现在**CPU外部地址总线上的寻址物理内存的地址信号**，是地址变换的最终结果。虚拟地址到物理地址的转化方法是与体系结构相关的，一般有分段与分页两种方式。以x86CPU为例，分段、分页都是支持的。内存管理单元负责从虚拟地址到物理地址的转化。**逻辑地址是段标识+段内偏移量的形式**，MMU通过查询段表，可以把逻辑地址转化为线性地址。如果CPU没有开启分页功能，那么线性地址就是物理地址；如果CPU开启了分页功能MMU还需要查询页表来将线性地址转化为物理地址：逻辑地址（段表）→线性地址（页表）→物理地址。

映射是一种多对一的关系，即不同的逻辑地址可以映射到同一个线性地址上；不同的线性地址也可以映射到同一个物理地址上。而且，同一个线性地址在发生换页以后，也可能被重新装载到另外一个物理地址上，所以这种多对一的映射关系也会随时间发生变化。

## 虚拟内存置换方式是怎么样的？

四种页面置换算法：

最佳 (OPT, Optional)

最近最少使用 (LRU, Least Recently Used)

先进先出 (FIFO, First In First Out)

时钟 (Clock)

### 最佳置换算法

OPT 策略选择置换下次访问距当前时间最长的那些页，可以看出该算法能导致最少的缺页中断，但是由于它要求操作系统必须知道将来的事件，显然这是不可能实现的。但它仍然能作为一种标准来衡量其他算法的性能。

### 最近最少使用算法

LRU 策略置换内存中上次使用距当前最远的页。根据局部性原理，这也是最不可能访问的页。实际上，LRU 策略的性能接近于 OPT 策略。该方法的问题在于比较难以实现。一种实现方法是给每一页添加一个最后访问的时间戳，并且必须每次访问内存时，都更新这个时间戳。即使有这种方案的硬件，开销仍然是非常大的。另外一种可选的方法是维护一个关于访问页的栈，但开销同样很大。

### 先进先出算法

FIFO 策略把分配给进程的页框视为一个循环缓冲区，按循环方式移动页。它所需的只是一个指针，这个指针在该进程的页框中循环。因此这是一种最简单的页面置换策略。除了它的简单性，这种选择方法所隐含的逻辑是置换驻留在内存中最长时间的页：一个很久以前取入内存的页，到现在可能已经不会再用了。这个推断是错误的，因为经常出现一部分程序或数据在整个程序的生命周期中使用频率很高的情况，如果使用 FIFO 算法，则这些页会被反复的换入换出，增加了系统开销。

### 时钟

时钟是 LRU 的近似实现。最简单的时钟策略需要给每一页框管理一个附加位，称为使用位。当某一页首次装入内存时，则将该页框的使用位置为 1；当该页随后被访问到时（在访问产生缺页中断后），他的使用位也会被置为 1。对于页面置换算法用于置换的候选页框集合被视为一个循环缓冲区，并且有一个指针与之相关联。当一页被置换时，该指针被设置成指向缓冲区中的下一个页框。当需要置换一页时，操作系统扫描缓冲区，以查找使用位被置为 0 的一个页框。每当遇到一个使用位为 1 的页框时，操作系统就将该为重新置为 0；如果在这个过程开始时，缓冲区所有页框的使用位均为 0，则选择遇到的第一个页框置换；如果所有页框的使用位均为 1，则指针在缓冲区中完整地循环一周，把所有使用位都置为 0，并且停留在最初的位置上，置换该页框中的页。

## 给你一个类，里面有static, virtual之类的，来说一说这个类的内存分布？

### 1. static修饰符

#### (1) static修饰成员变量

对于非静态数据成员，每个类对象都有自己的拷贝。而静态数据成员被当做是类的成员，无论这个类被定义了多少个，静态数据成员都只有一份拷贝，为该类型的所有对象所共享（包括其派生类）。所以，静态数据成员的值对每个对象都是一样的，它的值可以更新。

因为静态数据成员在全局数据区分配内存，属于本类的所有对象共享，所以它不属于特定的类对象，在没有产生类对象前就可以使用。

#### (2) static修饰成员函数

与普通的成员函数相比，静态成员函数由于不是与任何的对象相联系，因此它不具有this指针。从这个意义上来说，它无法访问属于类对象的非静态数据成员，也无法访问非静态成员函数，只能调用其他的静态成员函数。static修饰的成员函数，在代码区分配内存。

### 2. C++继承和虚函数

C++多态分为静态多态和动态多态。静态多态是通过重载和模板技术实现，在编译的时候确定。动态多态通过虚函数和继承关系来实现，执行动态绑定，在运行的时候确定。

动态多态实现有几个条件：

- (1) 虚函数；
- (2) 一个基类的指针或引用指向派生类的对象；

基类指针在调用成员函数(虚函数)时，就会去查找该对象的虚函数表。虚函数表的地址在每个对象的首地址。查找该虚函数表中该函数的指针进行调用。

每个对象中保存的只是一个虚函数表的指针，C++内部为每一个类维持一个虚函数表，该类的对象的都指向这同一个虚函数表。

虚函数表中为什么就能准确查找相应的函数指针呢？因为在类设计的时候，虚函数表直接从基类也继承过来，如果覆盖了其中的某个虚函数，那么虚函数表的指针就会被替换，因此可以根据指针准确找到该调用哪个函数。

### 3. virtual修饰符

如果一个类是局部变量则该类数据存储在栈区，如果一个类是通过new/malloc动态申请的，则该类数据存储在堆区。

如果该类是virutal继承而来的子类，则该类的虚函数表指针和该类其他成员一起存储。虚函数表指针指向只读数据段中的类虚函数表，虚函数表中存放着一个个函数指针，函数指针指向代码段中的具体函数。

如果类中成员是virtual属性，会隐藏父类对应的属性。

## 假设临界区资源释放，如何保证只让一个线程获得临界区资源而不是都获得？

给临界区资源加互斥锁，可以保证临界区资源释放时，只有一个线程获得临界区资源。

## 操作系统中的缺页中断是什么？

malloc()和mmap()等内存分配函数，在分配时只是建立了进程虚拟地址空间，并没有分配虚拟内存对应的物理内存。当进程访问这些没有建立映射关系的虚拟内存时，处理器自动触发一个缺页异常。

缺页中断：在请求分页系统中，可以通过查询页表中的状态位来确定所要访问的页面是否存在于内存中。每当所要访问的页面不在内存时，会产生一次缺页中断，此时操作系统会根据页表中的外存地址在外存中找到所缺的一页，将其调入内存。

缺页本身是一种中断，与一般的中断一样，需要经过4个处理步骤：

1. 保护CPU现场
2. 分析中断原因
- 3. 转入缺页中断处理程序进行处理**
4. 恢复CPU现场，继续执行

但是缺页中断是由于所要访问的页面不存在于内存时，由硬件所产生的一种特殊的中断，因此，与一般的中断存在区别：

1. 在指令执行期间产生和处理缺页中断信号
2. 一条指令在执行期间，可能产生多次缺页中断
3. 缺页中断返回是，执行产生中断的一条指令，而一般的中断返回是，执行下一条指令。

## OS缺页置换算法如何实现的？

当访问一个内存中不存在的页，并且内存已满，则需要从内存中调出一个页或将数据送至磁盘对换区，替换一个页，这种现象叫做缺页置换。当前操作系统最常采用的缺页置换算法如下：

**先进先出(FIFO)算法**：置换最先调入内存的页面，即置换在内存中驻留时间最久的页面。按照进入内存的先后次序排列成队列，从队尾进入，从队首删除。

**最近最少使用 (LRU) 算法**：置换最近一段时间以来最长时间未访问过的页面。根据程序局部性原理，刚被访问的页面，可能马上又要被访问；而较长时间内没有被访问的页面，可能最近不会被访问。当前最常采用的就是LRU算法。

## 系统调用是什么，你用过哪些系统调用，和库函数有什么区别？

### 系统调用

系统调用是通向操作系统本身的接口，是面向底层硬件的。通过系统调用，可以使得用户态运行的进程与硬件设备(如CPU、磁盘、打印机等)进行交互，是操作系统留给应用程序的一个接口。下面为适用于访问设备驱动程序的系统调用：

- |   |                      |
|---|----------------------|
| 1 | open：打开文件或设备         |
| 2 | read：从打开的文件或设备中读取数据  |
| 3 | write：向打开的文件或设备中写入数据 |
| 4 | close：关闭文件或设备        |
| 5 | ioctl：把控制信息传递给设备驱动文件 |

### 库函数

库函数（Library function）是把函数放到库里，供别人使用的一种方式。方法是把一些常用到的函数编完放到一个文件里，供不同的人进行调用。一般放在.lib文件中。库函数调用则是面向应用开发的，库函数可分为两类，一类是C语言标准规定的库函数，一类是编译器特定的库函数。

**系统调用是为了方便使用操作系统的接口，而库函数则是为了人们编程的方便。**

库函数调用与系统无关，不同的系统，调用库函数，库函数会调用不同的底层函数实现，因此可移植性好。

由于库函数是基于c库的，因此不能用于内核对于底层驱动设备的操作。

## 区别

1. 库函数是语言或应用程序的一部分，而系统调用是内核提供给应用程序的接口，属于系统的一部分
2. 库函数在用户地址空间执行，系统调用是在内核地址空间执行，库函数运行时间属于用户时间，系统调用属于系统时间，库函数开销较小，系统调用开销较大
3. 库函数是有缓冲的，系统调用是无缓冲的
4. 系统调用依赖于平台，库函数并不依赖

| 库函数                             | 系统调用                         |
|---------------------------------|------------------------------|
| 在所有的ANSI C编译器版本中，C库函数是相同的       | 各个操作系统的系统调用是不同的              |
| 它调用函数库中的一段程序                    | 它调用系统内核的服务                   |
| 与用户程序相联系                        | 是操作系统的一个入口点                  |
| 在用户地址空间执行                       | 在内核地址空间执行                    |
| 它的运行时间属于“用户时间”                  | 它的运行时间属于“系统时间”               |
| 属于过程调用，调用开销小                    | 需要在用户控件和内核上下文环境间切换，开销较大      |
| 在C函数库libc中有大约300个函数             | 在UNIX中大约有90个系统调用             |
| 典型的C函数库调用：system fprintf malloc | 典型的系统调用：chdir fork write brk |

## 为什么要有page cache，操作系统怎么设计的page cache？

加快从磁盘读取文件的速率。page cache中有一部分磁盘文件的缓存，因为从磁盘中读取文件比较慢，所以读取文件先去page cache中去查找，如果命中，则不需要去磁盘中读取，大大加快读取速度。

在Linux 内核中，文件的每个数据块最多只能对应一个 Page Cache 项，它通过两个数据结构来管理这些 Cache项，一个是radix tree，另一个是双向链表。Radix tree 是一种搜索树，Linux内核利用这个数据结构来通过文件内偏移快速定位Cache 项



## 上下文

### 上下文有哪些？怎么理解？

上下文简单说来就是一个环境。

用户空间的应用程序，通过系统调用，进入内核空间。这个时候用户空间的进程要传递很多变量、参数的值给内核，内核态运行的时候也要保存用户进程的一些寄存器值、变量等。**所谓的“进程上下文”，可以看作是用户进程传递给内核的这些参数以及内核要保存的那一整套的变量和寄存器值和当时的环境等。**

相对于进程而言，就是进程执行时的环境。具体来说就是各个变量和数据，包括所有的寄存器变量、进程打开的文件、内存信息等。

一个进程的上下文可以分为三个部分：**用户级上下文、寄存器上下文以及系统级上下文。**

1. 用户级上下文：正文、数据、用户堆栈以及共享存储区；
2. 寄存器上下文：通用寄存器、程序寄存器(IP)、处理器状态寄存器(EFLAGS)、栈指针(ESP)；
3. 系统级上下文：进程控制块task\_struct、内存管理信息(mm\_struct、vm\_area\_struct、pgd、pte)、内核栈。

**当发生进程调度时，进行进程切换就是上下文切换(context switch)。操作系统必须对上面提到的全部信息进行切换，新调度的进程才能运行。而系统调用进行的模式切换(mode switch)。模式切换与进程切换比较起来，容易很多，而且节省时间，因为模式切换最主要的任务只是切换进程寄存器上下文的切换。**

硬件通过触发信号，导致内核调用中断处理程序，进入内核空间。这个过程中，硬件的一些变量和参数也要传递给内核，内核通过这些参数进行中断处理。**所谓的“中断上下文”，其实也可以看作就是硬件传递过来的这些参数和内核需要保存的一些其他环境（主要是当前被打断执行的进程环境）。**中断时，内核不代表任何进程运行，它一般只访问系统空间，而不会访问进程空间，内核在中断上下文中执行时一般不会阻塞。

### 为什么会有上下文这种概念？

内核空间和用户空间是现代操作系统的两种工作模式，内核模块运行在内核空间，而用户态应用程序运行在用户空间。它们代表不同的级别，而对系统资源具有不同的访问权限。内核模块运行在最高级别（内核态），这个级下所有的操作都受系统信任，而应用程序运行在较低级别（用户态）。在这个级别，处理器控制着对硬件的直接访问以及对内存的非授权访问。内核态和用户态有自己的内存映射，即自己的地址空间。

其中，处理器总处于以下状态中的一种：

内核态，运行于进程上下文，内核代表进程运行于内核空间；

内核态，运行于中断上下文，内核代表硬件运行于内核空间；

用户态，运行于用户空间。

系统的两种不同运行状态，才有了上下文的概念。用户空间的应用程序，如果想请求系统服务，比如操作某个物理设备，映射设备的地址到用户空间，必须通过**系统调用**来实现。（系统调用是操作系统提供给用户空间的**接口函数**）。

通过系统调用，用户空间的应用程序就会进入内核空间，由内核代表该进程运行于内核空间，这就涉及到上下文的切换，用户空间和内核空间具有不同的地址映射，通用或专用的寄存器组，而**用户空间的进程要传递很多变量、参数给内核，内核也要保存用户进程的一些寄存器、变量等，以便系统调用结束后回到用户空间继续执行**，

## 什么情况下进行用户态到内核态的切换？

1. 进程上下文主要是异常处理程序和内核线程。内核之所以进入进程上下文是因为进程自身的一些工作需要在内核中做。例如，系统调用是为当前进程服务的，异常通常是处理进程导致的错误状态等。
2. 中断上下文是由于硬件发生中断时会触发中断信号请求，请求系统处理中断，执行中断服务子程序。

## 中断上下文代码中有哪些注意事项？

运行于进程上下文的内核代码是可抢占的，但中断上下文则会一直运行至结束，不会被抢占。所以中断处理程序代码要受到一些限制，在中断代码中不能出现实现下面功能的代码：

1. **睡眠或者放弃CPU**  
因为内核在进入中断之前会关闭进程调度，一旦睡眠或者放弃CPU，这时内核无法调度别的进程来执行，系统就会死掉。牢记：中断服务子程序一定不能睡眠（或者阻塞）。
2. **尝试获得信号量**  
如果得不到信号量，代码就会睡眠，导致（1）中的结果。
3. **执行耗时的任务**  
中断处理应该尽可能快，因为如果一个处理程序是IRQF\_DISABLED类型，他执行的时候会禁止所有本地中断线，而内核要响应大量服务和请求，中断上下文占用CPU时间太长会严重影响系统功能。  
中断处理程序的任务尽可能放在中断下半部执行。
4. **访问用户空间的虚拟地址**  
因为中断运行在内核空间。

## 请问线程需要保存哪些上下文，SP、PC、EAX这些寄存器是干嘛用的？

线程在切换的过程中需要保存当前线程id、线程状态、堆栈、寄存器状态等信息。其中寄存器主要包括SP、PC、EAX等寄存器，其主要功能如下：

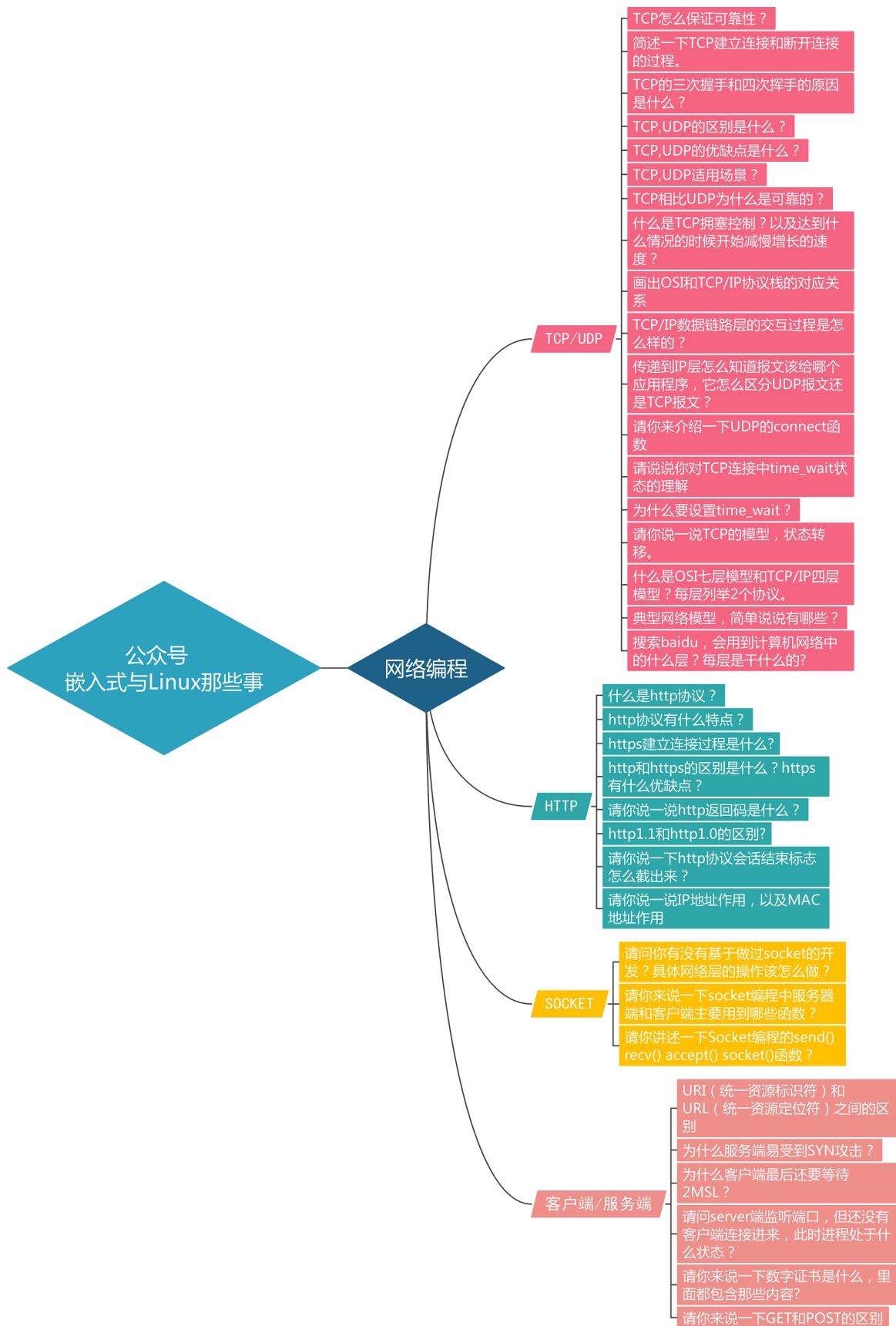
SP:堆栈指针，指向当前栈的栈顶地址

PC:程序计数器，存储下一条将要执行的指令

EAX:累加寄存器，用于加法乘法的缺省寄存器

## 网络编程

---



## TCP/ UDP

### TCP怎么保证可靠性？

#### TCP保证可靠性

1. 序列号、确认应答、超时重传

数据到达接收方，接收方需要发出一个确认应答，表示已经收到该数据段，并且确认序号会说明了它下一次需要接收的数据序列号。如果发送方迟迟未收到确认应答，那么可能是发送的数据丢失，也可能是确认应答丢失，这时发送方在等待一定时间后会进行重传。这个时间一般是 $2 \times RTT$ (报文段往返时间) +一个偏差值。

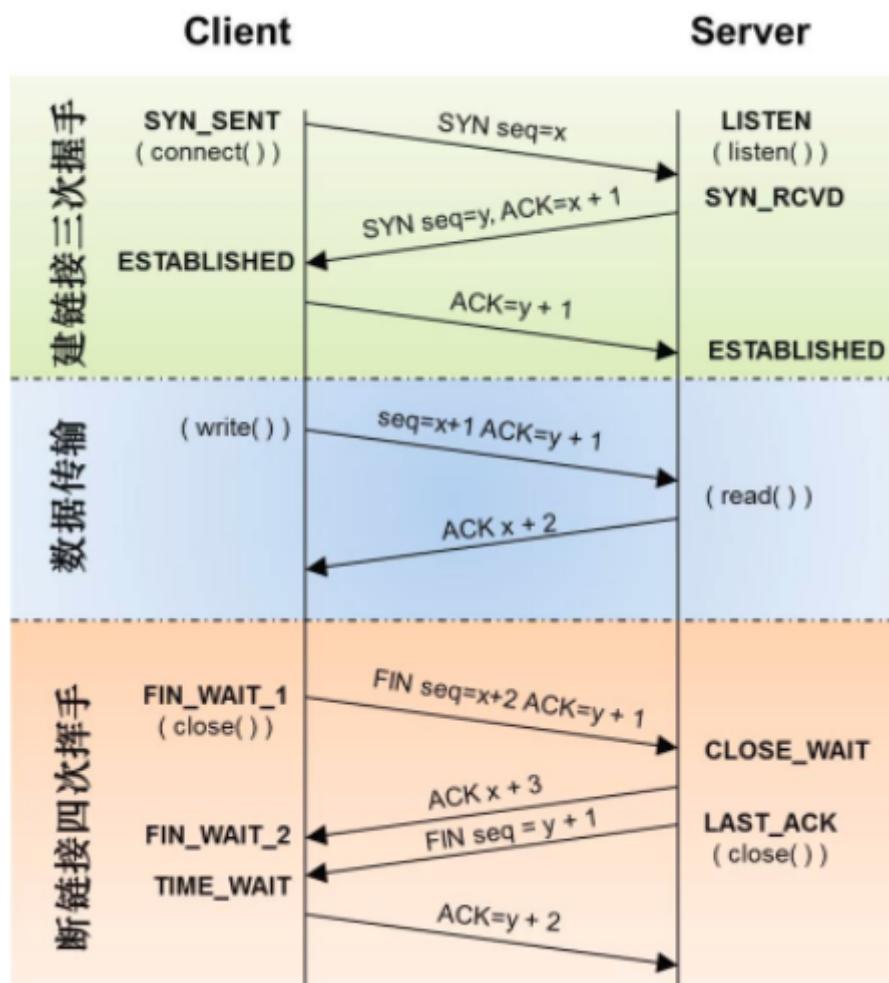
## 2. 窗口控制与高速重发控制/快速重传 (重复确认应答)

TCP会利用窗口控制来提高传输速度，意思是在一个窗口大小内，不用一定要等到应答才能发送下一段数据，窗口大小就是无需等待确认而可以继续发送数据的最大值。**如果不使用窗口控制，每一个没收到确认应答的数据都要重发。**

使用窗口控制，如果数据段1001-2000丢失，后面数据每次传输，确认应答都会不停地发送序号为1001的应答，表示我要接收1001开始的数据，发送端如果收到3次相同应答，就会立刻进行重发；但还有种情况有可能是数据都收到了，但是有的应答丢失了，这种情况不会进行重发，因为发送端知道，如果是数据段丢失，接收端不会放过它的，会疯狂向它提醒...

**简述一下TCP建立连接和断开连接的过程。**

**TCP建立连接和断开连接的过程**



### 三次握手

1. Client将标志位SYN置为1，随机产生一个值seq=j，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。
2. Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=j 1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。
3. Client收到确认后，检查ack是否为j 1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K 1，并将该数据包发送给Server，Server检查ack是否为K 1，ACK是否为1，如果正确则连接建立成

功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。

## 四次挥手

由于TCP连接时全双工的，因此，每个方向都必须要单独进行关闭，这一原则是当一方完成数据发送任务后，发送一个FIN来终止这一方向的连接，收到一个FIN只是意味着这一方向上没有数据流动了，即不会再收到数据了，但是在这个TCP连接上仍然能够发送数据，直到这一方向也发送了FIN。首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭。

1. 数据传输结束后，客户端的应用进程发出连接释放报文段，并停止发送数据，客户端进入FIN\_WAIT\_1状态，此时客户端依然可以接收服务器发送来的数据。
2. 服务器接收到FIN后，发送一个ACK给客户端，确认序号为收到的序号 1，服务器进入CLOSE\_WAIT状态。客户端收到后进入FIN\_WAIT\_2状态。
3. 当服务器没有数据要发送时，服务器发送一个FIN报文，此时服务器进入LAST\_ACK状态，等待客户端的确认
4. 客户端收到服务器的FIN报文后，给服务器发送一个ACK报文，确认序列号为收到的序号 1。此时客户端进入TIME\_WAIT状态，等待2MSL (MSL: 报文段最大生存时间)，然后关闭连接。

## TCP的三次握手和四次挥手的原因是什么？

### 为什么是三次握手？

1. 为了防止已失效的连接请求报文段突然有送到了B，因而产生错误
2. 假设两次握手时，A发出的第一个请求连接报文段在某一网络节点长时间滞留，以致延误到连接释放后才到达B。B收到失效的连接请求报文段后，认为是A又发出一次新的连接请求。于是向A发送确认报文段，同意建立连接，此时在假定两次握手的前提下，连接建立成功。这样会导致B的资源白白浪费
3. 假设两次握手时，A发出一个请求报文段，但发送过后A就因为问题而导致下线。之后B收到了A发来的请求连接报文段，给A发送确认报文段，同意建立连接，此时在假定两次握手的前提下，连接建立成功。这样会导致B的资源白白浪费

### 为什么是四次挥手？

1. TCP协议是全双工通信，这意味着客户端和服务器端都可以向彼此发送数据，所以关闭连接是双方都需要确认的共同行为
2. 假设是三次挥手时，首先释放了A到B方向的连接，此时TCP连接处于半关闭 (Half-Close) 状态，这时A不能向B发送数据，而B还是可以向A发送数据。如果此时A收到了B的确认报文段后，就立即发送一个确认报文段，这会导致B向A还在发送数据时连接就被关闭。这样会导致A没有完整收到B所发的报文段

## TCP， UDP的区别是什么？

1. TCP 是面向连接的，UDP 是面向无连接的
2. UDP程序结构较简单
3. TCP 是面向字节流的，UDP 是基于数据报的
4. TCP 保证数据正确性，UDP 可能丢包
5. TCP 保证数据顺序，UDP 不保证

## TCP， UDP的优缺点是什么？

### TCP的优点

可靠，稳定 TCP的可靠体现在TCP在传递数据之前，会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完后，还会断开连接用来节约系统资源。

### TCP的缺点

慢，效率低，占用系统资源高，易被攻击 TCP在传递数据之前，要先建连接，这会消耗时间，而且在数据传递时，确认机制、重传机制、拥塞控制机制等都会消耗大量的时间，而且要在每台设备上维护所有的传输连接，事实上，每个连接都会占用系统的CPU、内存等硬件资源。而且，因为TCP有确认机制、三次握手机制，这些也导致TCP容易被人利用，实现DOS、DDOS、CC等攻击。

### UDP的优点

快，比TCP稍安全 UDP没有TCP的握手、确认、窗口、重传、拥塞控制等机制，UDP是一个无状态的传输协议，所以它在传递数据时非常快。没有TCP的这些机制，UDP较TCP被攻击者利用的漏洞就要少一些。但UDP也是无法避免攻击的，比如：UDP Flood攻击。

### UDP的缺点

不可靠，不稳定 因为UDP没有TCP那些可靠的机制，在数据传递时，如果网络质量不好，就会很容易丢包。

## TCP， UDP适用场景？

### TCP应用场景

效率要求相对低，但对准确性要求相对高的场景。因为传输中需要对数据确认、重发、排序等操作，相比之下效率没有UDP高。举几个例子：文件传输（准确高要求高、但是速度可以相对慢）、接受邮件、远程登录。

### UDP应用场景

效率要求相对高，对准确性要求相对低的场景。举几个例子：QQ聊天、在线视频、网络语音电话（即时通讯，速度要求高，但是出现偶尔断续不是太大问题，并且此处完全不可以使用重发机制）、广播通信（广播、多播）。

## TCP相比UDP为什么是可靠的？

### 1. 确认和重传机制

建立连接时三次握手同步双方的“序列号 + 确认号 + 窗口大小信息”，是确认重传、流控的基础。传输过程中，如果Checksum校验失败、丢包或延时，发送端重传。

### 2. 数据排序

TCP有专门的序列号SN字段，可提供数据re-order

### 3. 流量控制

窗口和计时器的使用。TCP窗口中会指明双方能够发送接收的最大数据量

### 4. 拥塞控制

TCP的拥塞控制由4个核心算法组成。“慢启动”（Slow Start）、“拥塞避免”（Congestion avoidance）、“快速重传”（Fast Retransmit），“快速恢复”（Fast Recovery）

以上就是TCP比UDP传输更可靠的原因。

## 什么是TCP拥塞控制？以及达到什么情况的时候开始减慢增长的速度？

### 拥塞控制

如果把窗口定的很大，发送端连续发送大量的数据，可能会造成网络的拥堵（大家都在用网，你在这狂发，吞吐量就那么大，当然会堵），甚至造成网络的瘫痪。所以TCP在为了防止这种情况而进行了拥塞控制。

**慢启动：定义拥塞窗口，一开始将该窗口大小设为1，之后每次收到确认应答（经过一个rtt），将拥塞窗口大小\*2。**

**拥塞避免：**设置慢启动阈值，一般开始都设为65536。拥塞避免是指当拥塞窗口大小达到这个阈值，拥塞窗口的值不再指数上升，而是加法增加（每次确认应答/每个rtt，拥塞窗口大小+1），以此来避免拥塞。

将报文段的超时重传看做拥塞，则一旦发生超时重传，我们需要先将阈值设为当前窗口大小的一半，并且将窗口大小设为初值1，然后重新进入慢启动过程。

**快速重传：**在遇到3次重复确认应答（高速重发控制）时，代表收到了3个报文段，但是这之前的1个段丢失了，便对它进行立即重传。

然后，先将阈值设为当前窗口大小的一半，然后将拥塞窗口大小设为慢启动阈值+3的大小。

这样可以达到：在TCP通信时，网络吞吐量呈现逐渐的上升，并且随着拥堵来降低吞吐量，再进入慢慢上升的过程，网络不会轻易的发生瘫痪。

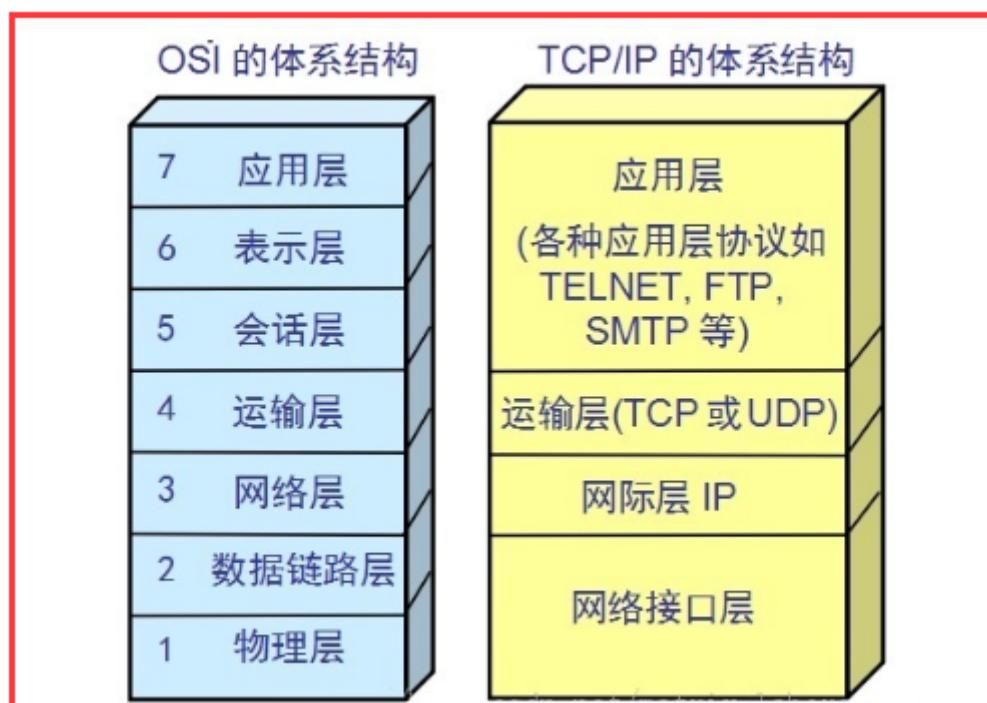
### 采用慢开始和拥塞避免算法的时候

1. 一旦 $cwnd >$ 慢开始门限，就采用拥塞避免算法，减慢增长速度
2. 一旦出现丢包的情况，就重新进行慢开始，减慢增长速度

### 采用快恢复和快重传算法的时候

1. 一旦 $cwnd >$ 慢开始门限，就采用拥塞避免算法，减慢增长速度
2. 一旦发送方连续收到了三个重复确认，就采用拥塞避免算法，减慢增长速度

### 画出OSI和TCP/IP协议栈的对应关系



OSI 参考模型共有 7 层，从下到上分别为：物理层、数据链路层、网络层、运输层、会话层、表示层、应用层

TCP/IP 模型大体分为四层：网络接口层、网际层、运输层、应用层

两者的对应关系为：

物理层+数据链路层 => 网络接口层

网络层 => 网际层

运输层 => 运输层

会话层 + 表示层 + 应用层 => 应用层

## TCP/IP数据链路层的交互过程是怎么样的？

网络层等在数据链路层用MAC地址作为通信目标，数据包到达网络层等往数据链路层发送的时候，首先回去ARP缓存表去查找ip对应的MAC地址，如果查到了，就将此ip对应的MAC地址封装到链路层数据包的包头。如果缓存中没有找到，则会发起一个广播，who is ip xxx tell ip xxxx，所有收到广播的机器看到这个ip是不是自己的，如果是自己的，则以单播的形式将自己的mac地址回复给请求机器。

## 传递到IP层怎么知道报文该给哪个应用程序，它怎么区分UDP报文还是TCP报文？

根据端口继续区分需接受的程序；

根据ip协议头中标识字段： UDP 17 、 TCP 6

## 请你来介绍一下UDP的connect函数

除非套接字已连接，否则异步错误是不会反悔到UDP套接字的。我们确实可以给UDP套接字调用connect，然而这样做的结果却与TCP连接不同的是没有三路握手过程。内核只是检查是否存在立即可知的错误，记录对端的IP地址和端口号，然后立即返回调用进程。

对于已连接UDP套接字，与默认的未连接UDP套接字相比，发生了三个变化。

其实一旦UDP套接字调用了connect系统调用，那么这个UDP上的连接就变成一对一的连接，但是通过这个UDP连接传输数据的性质还是不变的，仍然是不可靠的UDP连接。一旦变成一对一的连接，在调用系统调用发送和接受数据时也就可以使用TCP那一套系统调用了。

1、我们再也不能给输出操作指定目的IP地址和端口号。也就是说，我们不使用sendto，而改用write或send。写到已连接UDP套接字上的任何内容都自动发送到由connect指定的协议地址。可以给已连接的UDP套接字调用sendto，但是不能指定目的地址。sendto的第五个参数必须为空指针，第六个参数应该为0。

2、不必使用recvfrom以获悉数据报的发送者，而改用read、recv或recvmsg。在一个已连接UDP套接字上，由内核为输入操作返回的数据报只有那些来自connect指定协议地址的数据报。这样就限制一个已连接UDP套接字能且仅能与一个对端交换数据报。

3、由已连接UDP套接字引发的异步错误会返回给它们所在的进程，而未连接的UDP套接字不接收任何异步错误。

来自任何其他IP地址或断开的数据报不投递给这个已连接套接字，因为它们要么源IP地址要么源UDP端口不与该套接字connect到的协议地址相匹配。

UDP客户进程或服务器进程只在使用自己的UDP套接字与确定的唯一对端进行通信时，才可以调用connect。调用connect的通常是UDP客户，不过有些网络应用中的UDP服务器会与单个客户长时间通信TFTP，这种情况下，客户和服务器都可能调用connect。

## 请说说你对TCP连接中time\_wait状态的理解

先上TCP的状态变迁图

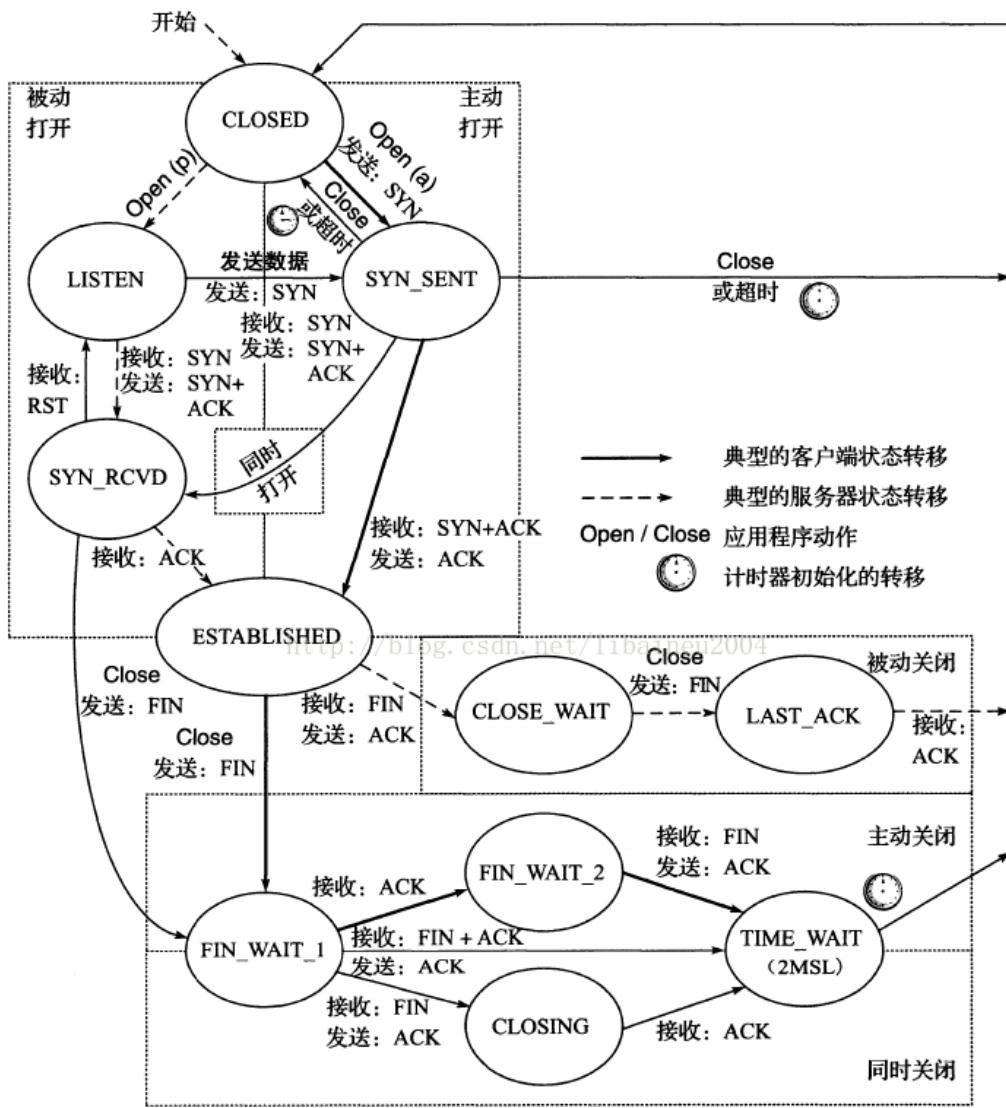


图 13-8 TCP 状态转换图（也称作有限状态机）。箭头表示因报文段传输、接收以及计时器超时而引发的状态转换。粗箭头表示典型的客户端的行为，虚线箭头表示典型的服务器行为。粗体指令（例如 open、close）是应用程序执行的操作

### time\_wait状态如何产生？

由上面的变迁图，首先调用close()发起主动关闭的一方，在发送最后一个ACK之后会进入time\_wait的状态，也就是说该发送方会保持2MSL时间之后才会回到初始状态。MSL值得是数据包在网络中的最大生存时间。产生这种结果使得这个TCP连接在2MSL连接等待期间，定义这个连接的四元组（客户端IP地址和端口，服务端IP地址和端口号）不能被使用。

### time\_wait状态产生的原因

#### 1. 为实现TCP全双工连接的可靠释放

由TCP状态变迁图可知，假设发起主动关闭的一方（client）最后发送的ACK在网络中丢失，由于TCP协议的重传机制，执行被动关闭的一方（server）将会重发其FIN，在该FIN到达client之前，client必须维护这条连接状态，也就是说这条TCP连接所对应的资源（client方的local\_ip,local\_port）不能被立即释放或重新分配，直到另一方重发的FIN到达之后，client重发ACK后，经过2MSL时间周期没有再收到另一方的FIN之后，该TCP连接才能恢复初始的CLOSED状态。如果主动关闭一方不维护这样一个TIME\_WAIT状态，那么当被动关闭一方重发的FIN到达时，主动关闭一方的TCP传输层会用RST包响应对方，这会被对方认为是有错误发生，然而这事实上只是正常的关闭连接过程，并非异常。

#### 2. 为使旧的数据包在网络因过期而消失

为说明这个问题，我们先假设TCP协议中不存在TIME\_WAIT状态的限制，再假设当前有一条TCP连接：(local\_ip, local\_port, remote\_ip, remote\_port)，因某些原因，我们先关闭，接着很快以相同的四元组建立一条新连接。本文前面介绍过，TCP连接由四元组唯一标识，因此，在我们假设的情况下，TCP协议栈是无法区分前后两条TCP连接的不同的，在它看来，这根本就是同一条连接，中间先释放再建立的过程对其来说是“感知”不到的。这样就可能发生这样的情况：前一条TCP连接由local peer发送的数据到达remote peer后，会被该remote peer的TCP传输层当做当前TCP连接的正常数据接收并向上传递至应用层（而事实上，在我们假设的场景下，这些旧数据到达remote peer前，旧连接已断开且一条由相同四元组构成的新TCP连接已建立，因此，这些旧数据是不应该被向上传递至应用层的），从而引起数据错乱进而导致各种无法预知的诡异现象。作为一种可靠的传输协议，TCP必须在协议层面考虑并避免这种情况的发生，这正是TIME\_WAIT状态存在的第2个原因。

### 3. 总结

具体而言，local peer主动调用close后，此时的TCP连接进入TIME\_WAIT状态，处于该状态下的TCP连接不能立即以同样的四元组建立新连接，即发起active close的那一方占用的local port在TIME\_WAIT期间不能再被重新分配。由于TIME\_WAIT状态持续时间为2MSL，这样保证了旧TCP连接双工链路中的旧数据包均因过期（超过MSL）而消失，此后，就可以用相同的四元组建立一条新连接而不会发生前后两次连接数据错乱的情况。

### time\_wait状态如何避免

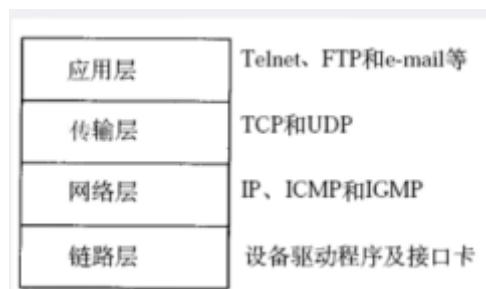
首先服务器可以设置SO\_REUSEADDR套接字选项来通知内核，如果端口忙，但TCP连接位于TIME\_WAIT状态时可以重用端口。在一个非常有用的场景就是，如果你的服务器程序停止后想立即重启，而新的套接字依旧希望使用同一端口，此时SO\_REUSEADDR选项就可以避免TIME\_WAIT状态。

### 为什么要设置time\_wait？

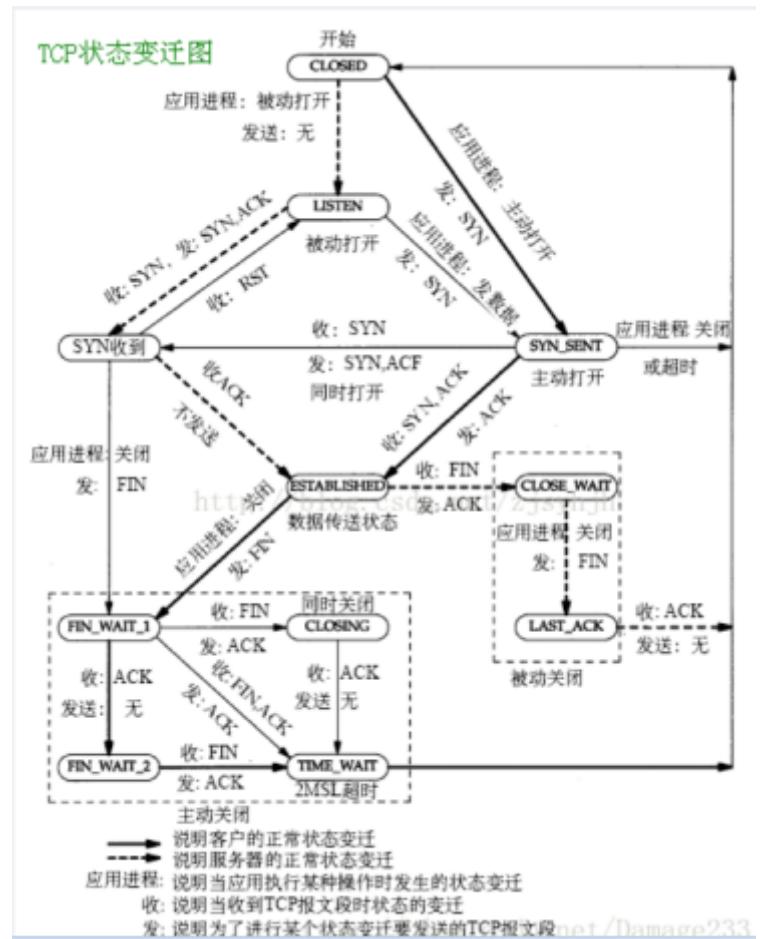
1. 保证A发送的最后一个ACK报文能够到达B。因为这个ACK报文有可能丢失，B收不到这个对FIN+ACK报文的确认，所以B就会超时重传这个FIN+ACK报文段，这样A就能在2MSL时间内收到重传的FIN+ACK，接着A再重传一次确认报文段，重新启动时间等待计时器
2. A在发送完最后一个ACK报文段后，经过2MSL后，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失

### 请你说一说TCP的模型，状态转移。

四层TCP/IP模型如下，包括应用层、网络层、数据链路层、物理层。



其状态转移图如下：



## 服务端的状态转移

服务端打开后处于listen，等待客户端的连接请求。当服务端接收到客户端发来的连接请求syn后，服务端进入synreceive状态，并回复syn和ack表示接受连接后，当服务端再次收到ack后。服务端认为连接已建立并进入establish状态。当服务端接收到fin=1后，表明客户端要关闭连接，这时服务端进入close\_wait状态，此时，服务端不在接收数据，但是可以继续发送数据，当服务端数据发送完后，服务端会发送fin，并进入last\_ack状态，在last\_ack状态下，服务端收到客户端发来的fin和ack后，服务端关闭连接，处于close状态。

## 客户端的状态转移

客户端发送syn = 1 后，会处于synsneed状态，当收到服务端发来的syn和ack后，客户端会返回一个确认发送ack，发完之后会进入establish状态。客户端发送fin=1主动关闭连接，会处于fin\_wait\_1状态。

此状态下：

1. 服务端会回复一个ack表示服务端还能发送数据，此时客户端会进入fin\_wait\_2状态，表明等待服务端关闭连接。
2. 收到服务端发来的fin =1，此时客户端发送ack，进入closing状态。
3. 收到服务端发来额fin+ack，此时客户端会发送ack进入time\_wait状态，等到2MSL后，就会彻底关闭。

## 什么是OSI七层模型和TCP/IP四层模型？每层列举2个协议。

### OSI七层模型及其包含的协议如下

1. 物理层: 通过媒介传输比特,确定机械及电气规范,传输单位为bit, 主要包括的协议为： IEEE802.3  
CLOCK RJ45
2. 数据链路层: 将比特组装成帧和点到点的传递,传输单位为帧,主要包括的协议为 MAC VLAN PPP
3. 网络层: 负责数据包从源到宿的传递和网际互连, 传输单位为包,主要包括的协议为 IP ARP ICMP
4. 传输层: 提供端到端的可靠报文传递和错误恢复, 传输单位为报文,主要包括的协议为 TCP UDP

5. 会话层：建立、管理和终止会话，传输单位为SPDU，主要包括的协议为RPC NFS
6. 表示层：对数据进行翻译、加密和压缩，传输单位为PPDU，主要包括的协议为JPEG ASCII
7. 应用层：允许访问OSI环境的手段，传输单位为APDU，主要包括的协议为FTP HTTP DNS

## 典型网络模型，简单说说有哪些？

网络模型一般是指OSI七层参考模型和TCP/IP四层参考模型。

**OSI分层（7层）**：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

**TCP/IP分层（4层）**：网络接口层、网际层、运输层、应用层。

**网络层**：IP协议、ICMP协议、ARP协议、RARP协议。

**传输层**：UDP协议、TCP协议。

**应用层**：FTP（文件传送协议）、Telnet（远程登录协议）、DNS（域名解析协议）、SMTP（邮件传送协议），POP3协议（邮局协议），HTTP协议。

## 搜索baidu，会用到计算机网络中的什么层？每层是干什么的？

浏览器中输入URL

浏览器要将URL解析为IP地址，解析域名就要用到DNS协议，首先主机会查询DNS的缓存，如果没有就给本地DNS发送查询请求。DNS查询分为两种方式，一种是递归查询，一种是迭代查询。如果是迭代查询，本地的DNS服务器，向根域名服务器发送查询请求，根域名服务器告知该域名的一级域名服务器，然后本地服务器给该一级域名服务器发送查询请求，然后依次类推直到查询到该域名的IP地址。**DNS服务器是基于UDP的，因此会用到UDP协议。**

得到IP地址后，浏览器就要与服务器建立一个http连接。因此要用到http协议，http协议报文格式上面已经提到。http生成一个get请求报文，将该报文传给TCP层处理，所以还会用到TCP协议。如果采用https还会使用https协议先对http数据进行加密。TCP层如果有需要先将HTTP数据包分片，分片依据路径MTU和MSS。TCP的数据包然后会发送给IP层，用到IP协议。IP层通过路由选路，一跳一跳发送到目的地址。**当然在一个网段内的寻址是通过以太网协议实现**(也可以是其他物理层协议，比如PPP, SLIP)，以太网协议需要知道目的IP地址的物理地址，有需要ARP协议。

其中：

### 1、DNS协议，http协议，https协议属于应用层

应用层是体系结构中的最高层。应用层确定进程之间通信的性质以满足用户的需要。这里的进程就是指正在运行的程序。应用层不仅要提供应用进程所需要的信息交换和远地操作，而且还要作为互相作用的应用进程的用户代理，来完成一些为进行语义上有意义的信息交换所必须的功能。应用层直接为用户的应用进程提供服务。

### 2、TCP/UDP属于传输层

传输层的任务就是负责主机中两个进程之间的通信。因特网的传输层可使用两种不同协议：即面向连接的传输控制协议TCP，和无连接的用户数据报协议UDP。面向连接的服务能够提供可靠的交付，但无连接服务则不保证提供可靠的交付，它只是“尽最大努力交付”。这两种服务方式都很有用，各有其优缺点。在分组交换网内的各个交换结点机都没有传输层。

### 3、IP协议，ARP协议属于网络层

网络层负责为分组交换网上的不同主机提供通信。在发送数据时，网络层将运输层产生的报文段或用户数据报封装成分组或包进行传送。在TCP/IP体系中，分组也叫作IP数据报，或简称为数据报。网络层的另一个任务就是要选择合适的路由，使源主机运输层所传下来的分组能够交付到目的主机。

### 4、数据链路层

当发送数据时，数据链路层的任务是将在网络层交下来的IP数据报组装成帧，在两个相邻结点间的链路上传送以帧为单位的数据。每一帧包括数据和必要的控制信息（如同步信息、地址信息、差错控制、以及流量控制信息等）。控制信息使接收端能够知道一个帧从哪个比特开始和到哪个比特结束。控制信息还使接收端能够检测到所收到的帧中有无差错。

## 5. 物理层

物理层的任务就是**透明地传送比特流**。在物理层上所传数据的单位是比特。传递信息所利用的一些物理媒体，如双绞线、同轴电缆、光缆等，并不在物理层之内而是在物理层的下面。因此也有人把物理媒体当做第0层。



## HTTP/IP

### 什么是http协议？

HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写，是用于从万维网（WWW:World Wide Web）服务器传输超文本到本地浏览器的传送协议。

HTTP是一个基于TCP/IP通信协议来传递数据（HTML文件，图片文件，查询结果等）。

HTTP是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。它于1990年提出，经过几年的使用与发展，得到不断地完善和扩展。目前在WWW中使用的是HTTP/1.0的第六版，HTTP/1.1的规范化工作正在进行之中，而且HTTP-NG（Next Generation of HTTP）的建议已经提出。

HTTP协议工作于客户端-服务端架构之上。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。Web服务器根据接收到的请求后，向客户端发送响应信息。

### http协议有什么特点？

#### 1. 简单快速

客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。

#### 2. 灵活

HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。

#### 3. 无连接

无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

#### 4. 无状态

HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

#### 5. 支持B/S及C/S模式。

6. 默认端口80

7. 基于TCP协议

## https建立连接过程是什么？

HTTP协议定义Web客户端如何从Web服务器请求Web页面，以及服务器如何把Web页面传送给客户端。HTTP协议采用了请求/响应模型。客户端向服务器发送一个请求报文，请求报文包含请求的方法、URL、协议版本、请求头部和请求数据。服务器以一个状态行作为响应，响应的内容包括协议的版本、成功或者错误代码、服务器信息、响应头部和响应数据。

HTTP 请求/响应的步骤如下：

1. 客户端连接到Web服务器

一个HTTP客户端，通常是浏览器，与Web服务器的HTTP端口（默认为80）建立一个TCP套接字连接。例如，<http://www.baidu.com>。

2. 发送HTTP请求

通过TCP套接字，客户端向Web服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据4部分组成。

3. 服务器接受请求并返回HTTP响应

Web服务器解析请求，定位请求资源。服务器将资源副本写到TCP套接字，由客户端读取。一个响应由状态行、响应头部、空行和响应数据4部分组成。

4. 释放连接TCP连接

若connection 模式为close，则服务器主动关闭TCP连接，客户端被动关闭连接，释放TCP连接；若connection 模式为keepalive，则该连接会保持一段时间，在该时间内可以继续接收请求；

5. 客户端浏览器解析HTML内容

客户端浏览器首先解析状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的HTML文档和文档的字符集。客户端浏览器读取响应数据HTML，根据HTML的语法对其进行格式化，并在浏览器窗口中显示。

## http和https的区别是什么？https有什么优缺点？

### 区别

1. HTTP协议是以明文的方式在网络中传输数据，而HTTPS协议传输的数据则是经过TLS加密后的，HTTPS具有更高的安全性
2. HTTPS在TCP三次握手阶段之后，还需要进行SSL 的handshake，协商加密使用的对称加密密钥
3. HTTPS协议需要服务端申请证书，浏览器端安装对应的根证书
4. HTTP协议端口是80，HTTPS协议端口是443

### HTTPS优点

HTTPS传输数据过程中使用密钥进行加密，所以安全性更高

HTTPS协议可以认证用户和服务器，确保数据发送到正确的用户和服务器

### HTTPS缺点

HTTPS握手阶段延时较高：由于在进行HTTP会话之前还需要进行SSL握手，因此HTTPS协议握手阶段延时增加

HTTPS部署成本高：一方面HTTPS协议需要使用证书来验证自身的安全性，所以需要购买CA证书；另一方面由于采用HTTPS协议需要进行加解密的计算，占用CPU资源较多，需要的服务器配置或数目高

## 请你说一说http返回码是什么？

HTTP协议的响应报文由状态行、响应头部和响应包体组成，其响应状态码总体描述如下：

1xx：指示信息--表示请求已接收，继续处理。

2xx：成功--表示请求已被成功接收、理解、接受。

3xx：重定向--要完成请求必须进行更进一步的操作。

4xx：客户端错误--请求有语法错误或请求无法实现。

5xx：服务器端错误--服务器未能实现合法的请求。

常见状态代码、状态描述的详细说明如下。

200 OK：客户端请求成功。

206 partial content 服务器已经正确处理部分GET请求，实现断点续传或同时分片下载，该请求必须包含Range请求头来指示客户端期望得到的范围

300 multiple choices (可选重定向) :被请求的资源有一系列可供选择的反馈信息，由浏览器/用户自行选择其中一个。

301 moved permanently (永久重定向) : 该资源已被永久移动到新位置，将来任何对该资源的访问都要使用本响应返回的若干个URI之一。

302 move temporarily(临时重定向)：请求的资源现在临时从不同的URI中获得，

304：not modified :如果客户端发送一个待条件的GET请求并且该请求以经被允许，而文档内容未被改变，则返回304,该响应不包含包体（即可直接使用缓存）。

403 Forbidden：服务器收到请求，但是拒绝提供服务。

404 Not Found：请求资源不存在，举个例子：输入了错误的URL。

## http1.1和http1.0的区别？

HTTP1.0最早在网页中使用是在1996年，那个时候只是使用一些较为简单的网页上和网络请求上，而HTTP1.1则在1999年才开始广泛应用于现在的各大浏览器网络请求中，同时HTTP1.1也是当前使用最为广泛的HTTP协议。主要区别主要体现在：

1. **缓存处理**，在HTTP1.0中主要使用header里的If-Modified-Since,Expires来做为缓存判断的标准，HTTP1.1则引入了更多的缓存控制策略例如Entity tag, If-Unmodified-Since, If-Match, If-None-Match等更多可供选择的缓存头来控制缓存策略。
2. **带宽优化及网络连接的使用**，HTTP1.0中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，HTTP1.1则在请求头引入了range头域，它允许只请求资源的某个部分，即返回码是206 (Partial Content)，这样就方便了开发者自由的选择以便于充分利用带宽和连接。
3. **错误通知的管理**，在HTTP1.1中新增了24个错误状态响应码，如409 (Conflict) 表示请求的资源与资源的当前状态发生冲突；410 (Gone) 表示服务器上的某个资源被永久性的删除。
4. **Host头处理**，在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名(hostname)。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机(Multi-homed Web Servers)，并且它们共享一个IP地址。HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误(400 Bad Request)。
5. **长连接**，HTTP 1.1支持长连接(PersistentConnection) 和请求的流水线(Pipelining) 处理，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟，在HTTP1.1中默认开启Connection: keep-alive，一定程度上弥补了HTTP1.0每次请求都要创建连接的缺点。

## 请你说一下http协议会话结束标志怎么截出来？

看tcp连接是否有断开的四部挥手阶段。

## 请你说一说IP地址作用，以及MAC地址作用

MAC地址是一个硬件地址，用来定义网络设备的位置，主要由数据链路层负责。而IP地址是IP协议提供的一种统一的地址格式，为互联网上的每一个网络和每一台主机分配一个逻辑地址，以此来屏蔽物理地址的差异。



## SOCKET

### 请问你有没有基于做过socket的开发？具体网络层的操作该怎么做？

服务端：socket-bind-listen-accept

客户端：socket-connect

### 请你来说一下socket编程中服务器端和客户端主要用到哪些函数？

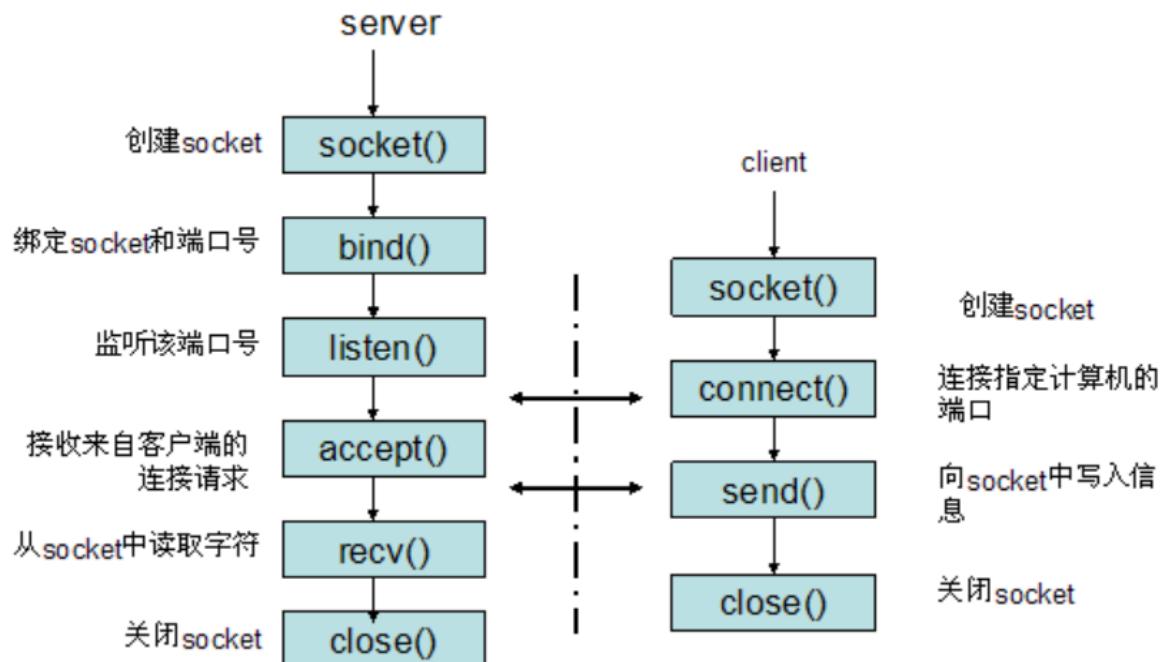
#### 基于TCP的socket

##### 1. 服务器端程序

- (1) 、创建一个socket，用函数socket()
- (2) 、绑定IP地址、端口等信息到socket上，用函数bind()
- (3) 、设置允许的最大连接数，用函数listen()
- (4) 、接收客户端上来的连接，用函数accept()
- (5) 、收发数据，用函数send()和recv()，或者read()和write()
- (6) 、关闭网络连接

##### 2. 客户端程序：

- (1) 、创建一个socket，用函数socket()
- (2) 、设置要连接的对方的IP地址和端口等属性
- (3) 、连接服务器，用函数connect()
- (4) 、收发数据，用函数send()和recv()，或read()和write()
- (5) 、关闭网络连接



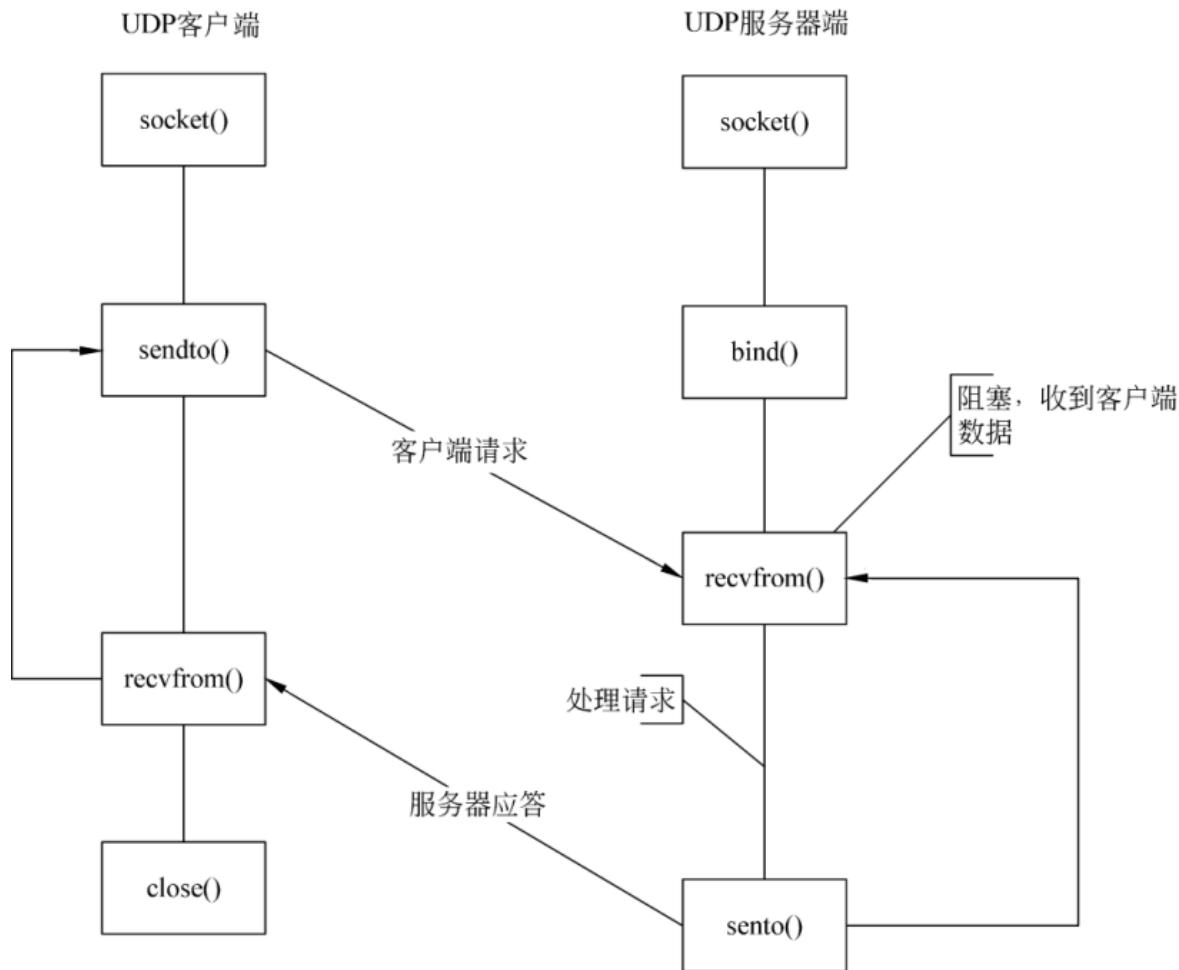
## 基于UDP的socket

### 1. 服务器端流程

- (1)、建立套接字文件描述符，使用函数socket()，生成套接字文件描述符。
- (2)、设置服务器地址和侦听端口，初始化要绑定的网络地址结构。
- (3)、绑定侦听端口，使用bind()函数，将套接字文件描述符和一个地址类型变量进行绑定。
- (4)、接收客户端的数据，使用recvfrom()函数接收客户端的网络数据。
- (5)、向客户端发送数据，使用sendto()函数向服务器主机发送数据。
- (6)、关闭套接字，使用close()函数释放资源。 UDP协议的客户端流程

### 2. 客户端流程

- (1)、建立套接字文件描述符，socket()。
- (2)、设置服务器地址和端口，struct sockaddr。
- (3)、向服务器发送数据，sendto()。
- (4)、接收服务器的数据，recvfrom()。
- (5)、关闭套接字，close()。



**请你讲述一下Socket编程的send() recv() accept() socket()函数？**

send函数将要发送的数据从用户空间拷贝到TCP关联的内核发送缓冲区

recv函数从TCP关联的内核接收缓冲区中取出数据到程序定义的缓冲区

accept函数从连接队列里边儿取出已经完成三次握手的连接

socket函数返回一个用于网络通讯的文件描述符



## 客户端/服务端

**URI（统一资源标识符）和URL（统一资源定位符）之间的区别**

### URL

**URL** 统一资源定位符（Uniform Resource Locator），其实就是我们访问web页面时需要输入的“网页地址”“网址”，比如：<https://www.google.com/> 就是URL。

完整定义如下：

协议类型：// 登录信息（认证） @ 服务器地址：端口号 / 带层次的文件路径 ? 查询字符串 # 片段标识符  
http://user:pass@[www.example.jp](http://www.example.jp):80/dir/index.html?uid=1#ch

## URI

**URI 统一资源标识符 (Uniform Resource Identifier)**，就是某个网络协议方案表示的资源的定位标识符，比如：<https://www.google.com/> 也同样可以说是，在https网络协议下的一个URI。

如果想要了解 URI统一资源标识符，那么我们必须理清它和 URL统一资源定位符之间的关系。

第一，URL统一资源定位符是一个具体的概念，而URI统一资源标识符是一个抽象的概念。

第二，URL统一资源定位符是URI统一资源标识符的子集。

**它是 URL，而 URL 属于 URI。：**

http://user:pass@[www.example.jp](http://www.example.jp):80/dir/index.html?uid=1#ch

**它是 URI：**

联系地址：// 访客:央视记者 @ 央视：直播 / 中国/湖北/武汉/红十字会的仓库？物品=捐款物资

## 为什么服务端易受到SYN攻击？

SYN-Flood攻击是当前网络上最为常见的DDoS攻击，也是最为经典的拒绝服务攻击，它就是利用了TCP协议实现上的一个缺陷，通过向网络服务所在端口发送大量的**伪造源地址的攻击报文**，就可能造成目标服务器中的**半开连接队列**被占满，从而阻止其他合法用户进行访问。这种攻击早在1996年就被发现，但至今仍然显示出强大的生命力。它的数据包特征通常是，源发送了大量的SYN包，并且缺少三次握手的最后一步握手ACK回复。

原理：**攻击者首先伪造地址对服务器发起SYN请求，服务器回应(SYN+ACK)包，而真实的IP会认为，我没有发送请求，不作回应。**服务器没有收到回应，这样的话，服务器不知道(SYN+ACK)是否发送成功，默认情况下会重试5次。这样的话，对于服务器的内存，带宽都有很大的消耗。攻击者如果处于公网，可以伪造IP的话，对于服务器就很难根据IP来判断攻击者，给防护带来很大的困难。

### SYN Flood 防护措施

#### 1. 无效连接监视释放

这种方法不停的监视系统中半开连接和不活动连接，当达到一定阈值时拆除这些连接，释放系统资源。这种绝对公平的方法往往也会将正常的连接的请求也会被释放掉，“伤敌一千，自损八百”。

#### 2. 延缓TCB分配方法

SYN Flood关键是利用了，SYN数据报文一到，系统立即分配TCB资源，从而占用了系统资源，因此有两种技术来解决这一问题。

**Syn Cache技术**，这种技术在收到SYN时不急着去分配TCB，而是先回应一个ACK报文，并在一个专用的HASH表中（Cache）中保存这种半开连接，直到收到正确的ACK报文再去分配TCB。

**Syn Cookie技术**，Syn Cookie技术则完全不使用任何存储资源，它使用一种特殊的算法生成Sequence Number，这种算法考虑到了对方的IP、端口、己方IP、端口的固定信息，以及对方无法知道而己方比较固定的一些信息，如MSS、时间等，在收到对方的ACK报文后，重新计算一遍，看其是否与对方回应报文中的（Sequence Number-1）相同，从而决定是否分配TCB资源。

#### 3. 使用SYN Proxy防火墙

原理：对试图穿越的SYN请求进行验证之后才放行

## 为什么客户端最后还要等待2MSL？

MSL (Maximum Segment Lifetime) , TCP允许不同的实现可以设置不同的MSL值。

保证客户端发送的最后一个ACK报文能够到达服务器，因为这个ACK报文可能丢失，站在服务器的角度看来，我已经发送了FIN+ACK报文请求断开了，客户端还没有给我回应，应该是我发送的请求断开报文它没有收到，于是服务器又会重新发送一次，而客户端就能在这个2MSL时间段内收到这个重传的报文，接着给出回应报文，并且会重启2MSL计时器。

防止类似与“三次握手”中提到了的“已经失效的连接请求报文段”出现在本连接中。客户端发送完最后一个确认报文后，在这个2MSL时间中，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失。这样新的连接中不会出现旧连接的请求报文。

建立连接的时候，服务器在LISTEN状态下，收到建立连接请求的SYN报文后，把ACK和SYN放在一个报文里发送给客户端。而关闭连接时，服务器收到对方的FIN报文时，仅仅表示对方不再发送数据了但是还能接收数据，而自己也未必全部数据都发送给对方了，所以己方可以立即关闭，也可以发送一些数据给对方后，再发送FIN报文给对方来表示同意现在关闭连接，因此，己方ACK和FIN一般都会分开发送，从而导致多了一次。

TCP还设有一个保活计时器，显然，客户端如果出现故障，服务器不能一直等下去，白白浪费资源。服务器每收到一次客户端的请求后都会重新复位这个计时器，时间通常是设置为2小时，若两小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔75秒发送一次。若一连发送10个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。

## 请问server端监听端口，但还没有客户端连接进来，此时进程处于什么状态？

这个需要看服务端的编程模型，如果如上一个问题的回答描述的这样，则处于阻塞状态。

如果使用了epoll,select等这样的io复用情况下，处于运行状态。

## 请你来说一下数字证书是什么，里面都包含那些内容？

### 1. 概念

数字证书是数字证书在一个身份和该身份的持有者所拥有的公/私钥对之间建立了一种联系，由认证中心 (CA) 或者认证中心的下级认证中心颁发的。根证书是认证中心与用户建立信任关系的基础。在用户使用数字证书之前必须首先下载和安装。

认证中心是一家能向用户签发数字证书以确认用户身份的管理机构。为了防止数字凭证的伪造，认证中心的公共密钥必须是可靠的，认证中心必须公布其公共密钥或由更高级别的认证中心提供一个电子凭证来证明其公共密钥的有效性，后一种方法导致了多级别认证中心的出现。

### 2. 数字证书颁发过程

数字证书颁发过程如下：用户产生了自己的密钥对，并将公共密钥及部分个人信息信息传送给一家认证中心。认证中心在核实身份后，将执行一些必要的步骤，以确信请求确实由用户发送而来，然后，认证中心将发给用户一个数字证书，该证书内附了用户和他的密钥等信息，同时还附有对认证中心公共密钥加以确认的数字证书。当用户想证明其公开密钥的合法性时，就可以提供这一数字证书。

### 3. 内容：

数字证书的格式普遍采用的是X.509V3国际标准，一个标准的X.509数字证书包含以下一些内容：

- (1) 证书的版本信息；
- (2) 证书的序列号，每个证书都有一个唯一的证书序列号；
- (3) 证书所使用的签名算法；
- (4) 证书的发行机构名称，命名规则一般采用X.500格式；
- (5) 证书的有效期，通用的证书一般采用UTC时间格式；

- (6) 证书所有人的名称，命名规则一般采用X.500格式；
- (7) 证书所有人的公开密钥；
- (8) 证书发行者对证书的签名。

## 请你来说一下GET和POST的区别

1. get参数通过url传递，post放在request body中。
2. get请求在url中传递的参数是有长度限制的，而post没有。
3. get比post更不安全，因为参数直接暴露在url中，所以不能用来传递敏感信息。
4. get请求只能进行url编码，而post支持多种编码方式。
5. get请求会浏览器主动cache，而post支持多种编码方式。
6. get请求参数会被完整保留在浏览器历史记录里，而post中的参数不会被保留。
7. GET和POST本质上就是TCP链接，并无差别。但是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。
8. GET产生一个TCP数据包；POST产生两个TCP数据包。

# 面经总结

## 我的嵌入式软件工程师秋招之路

秋招是每个在校学生都要经历的一个阶段。本篇文章记录了自己的秋招历程。秋招投递公司23家，简历被刷1家。笔试/测评挂掉3家。至今无消息的8家。获得Offer的公司有小米，兆易创新，全志科技，浙江大华，海格通信，京信通信，景嘉微电子，广州朗国电子，北京华大电子，中国长城某子公司。已签约浙江大华。最后收获了一个满意的Offer。前事不忘，后事之师。希望自己总结的这些内容能对后面准备秋招的同学有所帮助！

### 1. 自我介绍

本硕双非，本科电子信息工程，硕士电子与通信工程。导师申请的项目中有一部分需要用Stm32实现，所以自己在硕士期间接触Stm32比较多。当时也考虑到，如果只会Stm32，找工作可能会比较吃力。而自己对嵌入式底层的内容也比较感兴趣。所以，在研二的时候每天花一点时间来学习下驱动开发，以后找工作打算从事底层驱动开发相关的内容。

### 2. 秋招准备

#### 2.1 Linux驱动

在2019年12月的时候，基本就把韦东山老师的第二期课程学习了一遍了，虽然在学习过程中有很多不明白的，但也坚持看了一遍。把有疑问的地方记录了下来，打算后面再慢慢的去深入研究。

韦东山老师讲的课程确实很好，但是对于基础不太好的可能会比较吃力，很容易劝退。当时思考了下，**自己为什么听不懂呢，哪里有欠缺？**我们对自己应该有一个清晰的认识，我从Stm32转驱动开发，优势就是我对于基本的硬件原理都比较熟悉，欠缺的是对于Arm架构的深入了解，操作系统和计算机组成原理的基本知识。所以，这个时候发现自己听不懂的情况下，有些问题搞不明白，先不要深究，后面可以慢慢补。

今年疫情在家，在3月份的时候，对照自己之前的学习笔记和遗留的疑问，把之前学过的知识又看了下，当看第二遍的时候，对于很多问题也可以想清楚了。

考虑到驱动这块没有做过具体的项目，我就对照着韦东山老师的移植Uboot的视频，自己移植了一遍。 ([S3C2440移植uboot之编译烧写uboot](#)) 自己对于Uboot的启动流程也就很熟悉了。[超详细分析Bootloader \(Uboot\) 到内核的启动流程 \(万字长文！\)](#) 既然打算把这个写在简历上，就要把这个启动流程搞得特别明白，有些基本的源码也是要知道的（比如，如何初始化NAND FLASH，CLK如何配置的）。而且，写在简历上的面试官一定会问的。

## 2.2 Arm体系与架构

关于Arm的体系架构这部分是一定要看的。推荐一本书，杜春雷老师写的《**ARM体系结构与编程**》，这本书其实就是ARMV7开发手册的中文版，很多内容都是手册里面的。书的话不一定都要看，学习过程中，哪里不了解再去查书，效果可能会更好。

## 2.3 数据结构与算法分析

数据结构与算法的内容时秋招中的重中之重，**笔试必考，面试必考**。所以这部分要好好准备。

数据结构的话本科也没学过。4月份的时候开始在家，从基本的链表，二叉树，堆，队列，字符串，排序算法，查找算法等基础内容学起。当时参考的资料是在网上找的一本PDF。里面包含了基本数据结构的算法的实现。对着PDF资料边看边理解，自己动手去实现了一遍。刚开始学习数据结构的内容，一定要多画图，像链表的内容，不画图有时候可能不太好理解指针是如何指向的。总之要找到适合自己的学习方式。

图论的内容比较难，如果不打比赛，可以直接跳过（只针对嵌入式开发的同学）。还有字符串的KMP算法也比较难理解（其实刷题Leetcode题目之后就会发现，字符串匹配的题目解法很多，不一定要KMP）。也可以先不看。

关于我写的一些数据结构的内容，可以在数据结构与算法分析专栏中看到。大家可以参考下。

此外我还整理了下数据结构中比较重要的内容，在面试中一定要达到可以在白纸上写的水平。具体如下图所示。

- ▽ 面试中需要能达到手写水平的代码
  - ▽ 链表
    - 实现链表的逆置
    - 判断单链表中是否存在环
    - 单链表相交，如何求交点
    - 求有环链表第一个入环节点
    - 写出链表的删除一个节点的程序
    - 用普通算法实现两个有序链表的合并
    - 用递归算法实现两个有序列表的合并
  - ▽ 二叉树
    - 先序遍历
    - 中序遍历
    - 后序遍历（重点）
  - ▽ 排序/查找算法及其改进
    - 快速排序
    - 冒泡排序
    - 堆排序
    - 插入排序
    - 选择排序
    - 二分查找法

在学习完基本的数据结构与算法的内容之后，接下来就是刷题了。我自己是在Leetcode上刷的题目。由于时间有限，我刷题一开始的策略是按照**题目类型**去刷题（参考了知乎各位大佬的刷题策略），主要内容集中在了链表，二叉树，字符串，数组这四个部分。把这几个部分出现频率高的题目都总结了出来。各个部分频率总结链接：

[【leetcode】高频题目整理 所有题目汇总篇\(High Frequency Problems, All Problems.\)](https://blog.csdn.net/qq_16933601)

#### [面试leetcode题型总结](#)

刷题过程中会遇到一些特定算法的题目，比如滑动窗口，双指针，动态规划等。遇到这种题目，能解决的先解决掉，不能解决的后面可以按照**算法类型**统一解决。

最后，在笔试的时候，很多公司的大题的笔试系统其实并不是像Leetcode一样只写个子函数就行了，而是和ACM竞赛的类型一样，需要自己处理输入输出。**这部分一定要提前练习**。如果不熟悉系统，笔试的时候虽然有思路，但是也写不出来。建议提前熟悉下输入输出。

[OJ在线编程常见输入输出练习场](#)

## 2.4 C语言

C语言这部分的话，可以上网搜索一些面试中经常问到的内容。这里也推荐一本书，何昊老师写的《程序员面试笔试宝典第三版》（不要找错了）。这本书是针对C/C++程序员的，主要介绍了面试过程中经常问到的问题，整体的内容偏向底层，问题解释的也比较清楚。但是有些地方有错误，看的时候要注意。可以加书后面的QQ群，向作者反馈错误。

C++语言，平常用得少，我这部分就没准备，给不了大家意见。在面试中好几次问到我C++的指针引用之类的基础知识，只能和面试官说C++内容用得少，不熟悉。不过，这并不会成为面试官最后是否要你的决定性因素（只针对嵌入式底层来说）。

## 2.5 操作系统&计组

这块是我的软肋，好多概念不太懂。不过在学习驱动的过程中，我把这些概念也都整理了一些。具体文章可以看下面的内容。

[你真的懂Linux内核中的阻塞和异步通知机制吗？（花了五天整理，墙裂推荐！）](#)

[面试官让你讲讲Linux内核的竞争与并发，你该如何回答？](#)

[S3C2410 MMU（存储器管理单元）详述](#)

[Linux内核中断顶半部和底半部的理解](#)

[谈谈进程上下文、中断上下文及原子上下文的一些概念](#)

关于操作系统的一些知识，同样是整理了网上常见的一些面试题目。这份嵌入式软件开发知识点总结一共有13W字，涵盖了Linux，C语言，Arm体系与架构，操作系统，计算机组成原理等方方面面的知识。我在秋招过程中问到的问题，在里面基本都可以找到。这份资料给了我很大帮助。资料放在了公众号【嵌入式与Linux那些事】中，大家可以关注后回复“秋招大礼包”免费自取。

- └ 线程是否具有相同的堆栈?
- └ 线程同步方法
- └ 内核线程和用户线程的区别
- └ 读写锁
- └ 产生死锁的原因是什么 ?
- └ 死锁的4个必要条件
- └ 死锁的处理方式有哪些 ?
- └ 如何避免死锁
  - └ 加锁顺序
  - └ 加锁时限
  - └ 死锁检测
- └ 用户栈和内核栈有什么区别
- └ 内存泄漏
- └ 内存管理有哪几种方式
- └ 什么是虚拟内存
- └ 内存碎片 内碎片 外碎片
- └ 虚拟地址、逻辑地址、线性地址、物理地址
- └ CPU对外部设备的控制方式按CPU的介入程度，从小到大依次为通道方式，DMA方式，中断方式，程序控制方式。
- └ 进程的几种状态
- └ 异常处理
- └ S3C2410 MMU (存储器管理单元) 详述 [\[1693360\]](#)

## 2.6 项目经验

如果自己做过一些项目的话，建议写两到三个自己做过的项目，主要从以下几个方面介绍：

**项目名称：**基于XXX的XXX

**个人角色：**项目负责人/模块负责人

**起止时间：**2020.3~2020.6

**项目描述：**主要介绍项目主要内容，4句话即可。

**编程语言和环境：**gcc3.4.2, Ubuntu16.04, S3C2440开发板

**负责事宜：**写自己做了哪些内容，第一，XXXX；第二，XXXX；第三，XXX；

如果没有做过项目，可以去找一些开源的项目做一下（韦东山老师第三期的视频中有相关项目的介绍），比如移植Uboot，数码相框等（**韦老师最新的IMX6ULL开发板已经有了更加高大上的项目，也可以写在简历中**）。但是不要找那种烂大街的，比如XXX管理系统，五子棋，贪吃蛇。说实话，这些东西再怎么做，一听这个名字面试官就会没兴趣了。

我们也可以适当“包装”自己的项目，我这个项目是导师申请的国家自然科学基金项目（或者XX省科技重大专项）的一部分，这个项目主要是完成XX技术的攻关。我这个项目是和XX部队（XX高校）合作的项目，我们负责XX部分。这么一写的话“档次”是不是就上去了呢？**但是这里不是让大家在简历中去造假，在简历中造假是不能接受的，这是一个人基本的道德底线！**

自己写上去的项目一定要搞清楚，弄得明明白白。有好几次面试中，面试官让我讲你做的这个东西原理是什么。由于原理比较复杂，所以面试中，我都是边画图边讲，涉及到的公式还要进行推导，为什么这样做，都得给面试官讲清楚。**如果他没有听懂，那就是我们没有讲清楚。**

说了这么多，其实主要就是弄清楚这几个问题：**为什么做这个项目？一共几个人做？你是什么角色？你做了哪些部分？你认为最难得地方是什么？如何解决的？最后收获是什么？**

事实上，我们所做的东西是什么，面试官并不在意，毕竟公司做的东西比我们要复杂的多，对比公司的产品来说，我们的简直太low了，我们所做的可能只是个“玩具”而已。面试官主要考察的就是项目的真实性和我们的表达能力。

也可以参考下其他人整理的[如何在面试中介绍自己的项目经验](#)

## 2.7 其他

至于其他内容，主要有以下几个方面吧。

### 1.不要放过任何一次和面试官聊天的机会（发哥和我说的，受益匪浅）。

今年疫情在家的时候，请教了发哥关于校招找一份什么样的工作，嵌入式软件工程师的职业规划等问题，非常感谢发哥在晚上十点多手打了1000多字解答我的疑惑。特别强调了，在校生要抓住校招的机会，提前准备，多和面试官聊聊天。和发哥聊完后自己对于整个行业有了更清楚的认识，谢谢发哥！

**多面试！多面试！多面试！重要的话说三遍！**看到有合适的公司要尽早投递，不要到后面没有HC了。

比如我投递华为是在9.10号。笔试在9.14。虽然笔试过了，测评也过了。但是这个时候其实有点晚了（也可能和投递的部门有关系）。部门的HR和我说，他们第一批的面试已经结束了，领导要求这段时间要形成闭环。第二批还会有一些HC，但是在10月之后了，而且名额也不会太多了。相反，投递其他部门的同学，在9.19-9.24这周都收到了面试通知。

其刚开始前几次面试肯定会有些紧张，不知所措。但是当你面了三五场之后，你就会发现，面试官问的东西都是差不多的。我在9月中下旬，最多的一天面了4场。有句话说得好，**吹牛X吹多了，后面也就很自然了。**

### 2.回答问题一定要有逻辑性！先抛出结论，再分要点回答。

比如，面试官问你平常你是如何学习专业知识的？以什么样的方式？

答：**学习的形式主要分为两种：一种是在学习的过程中解决疑问，以解决问题为导向。**比如，在移植Uboot的过程中，我会想到为什么要关闭I-Cache,D-Cache等，关闭中断等（这里可以抛出这些问题，以防后面会问），遇到这些问题我就会记录下来，然后上网去查资料。在解决这些问题的过程中，可能会遇到其他的问题，继续查找相关资料，直到最后都搞清了。**第二种就是系统的学习**，如果有比较充足的时间，我会去拿着像Arm体系和架构，深入理解计算机系统这些书去一点一点的读。去理解这些内容。**以上就是我日常的一个学习方式。**

### 3.自我介绍。

自我介绍是面试中必不可少的部分。面试官一般会利用自我介绍的时间来浏览下你的简历。建议准备一个一分钟的自我介绍和三分钟的自我介绍。我在面试小米的时候，面试官直接打断我，说自我介绍简单一点，要不后面没有时间写代码了。

我的自我介绍主要是三部分，**第一部分**是开场白，名字，学校，专业，应聘岗位，应聘该岗位的原因。**第二部分**是项目经验的简单介绍，这里一定要简洁，主要说下自己做了那些东西。**第三部分**是学习能力的介绍，这部分是为了抛出自己写博客的情况，有技术博客并且博客的内容比较充实的话，在面试中确实是个加分项。

#### 4.简历填写

投递简历时常常需要在各个网站填写自己的个人信息。建议使用如下工具，可以方便的在各个网站自动识别填充所需内容。当然，也有许多识别不了的。这种就没办法了。当然，如果大家有其他工具也可以推荐！

[生客简历助手](#)

#### 5.最后要知道一些知识

链接如下[Offer, 三方, 两方, 毅约 这些你需要知道的事](#)

### 3. 书籍推荐

#### C语言

**C Primer Plus**, C缺陷和陷阱, **C和指针**, C专家编程

#### 数据结构

**大话数据结构**, 数据结构与算法描述-C语言描述。

#### 硬件原理

**Arm体系结构与编程**, Armv7/Armv9数据手册。

#### Linux驱动

**嵌入式Linux应用开发完全手册**, Linux设备驱动开发详解, Linux设备驱动程序

#### Linux应用编程

**Unix环境高级编程**, Unix环境网络编程

#### 计算机基础

**深入理解计算机系统, 现代操作系统**, 计算机组成与设计：硬件软件接口，计算机体系结构：量化研究方法

#### Linux内核

**Linux内核完全注释**（麻雀虽小五脏俱全），Linux内核设计与实现，**Linux内核源代码情景分析**

以上书籍加粗的为重点推荐。如果时间不充裕，可以当作工具书来查询。当然，有时间还是建议认真读下。

需要电子书的可以在公众号【嵌入式与Linux那些事】回复“电子书”领取，或者点击右下角加QQ群，群里也有整理的相关资料。

## 4. 未来展望

1.接下来的时间准备下大论文的内容，之前投的第二篇小论文有了审稿意见，只是没改而已，今年过年之前投出去吧。保证自己顺利毕业！

2.补充下操作系统和计算机组成原理的知识。主要看下《深入理解计算机系统》《现代操作系统》这两本书吧，每周把看过的东西总结下，照常输出一篇博客。这部分是最主要的内容，把基础打扎实了。

3.操作系统和计算机组成原理的知识形成博客专栏，坚持！

4.坚持每周的跑步，打球。

5.期待入职大华，即将开启一段新的旅程，充满了未知和挑战，要继续加油鸭！

## 5. 总结

秋招结束了，今年感觉好难，好多公司缩招。在九月初有段时间，心情很低落，一直在怀疑自己。特别是简历被刷掉以后，无比难受！心里想的，怎么也得给个笔试面试的机会吧。心情不好的时候，我就会去操场跑步，听着音乐，特别喜欢大汗淋漓的感觉。跑完之后心情会好许多。调整好心情，再次备战秋招，看面经，刷题，总结基本知识！

## 6. 致谢

刚开始拿到Offer后不知道该如何抉择，父母那边倒是没有太大的问题，说哪里工作合适就去哪里。很感谢我的父母可以支持我。当时咨询了发哥，波哥，平哥，朱老师，豆豆姐等，还有在投递简历前也请教了下肖遥哥。非常感谢这些前辈能给我建议！

2020年疫情在家期间，自己对于找一份什么样的工作并没有明确的规划。于是咨询了发哥关于校招，职业发展的问题。发哥大晚上的手打了1000多字回复我，特别强调要重视校招，**多和面试官聊天**，把握机会。在职业发展规划上，发哥向我介绍了底层驱动是做什么的，在不同的公司扮演什么角色，以及未来的晋升通道等。应聘的时候，要**多些自信**，努力把握好人生的选择点，在没有做选择前，**不要把话说死，把方向定死**。很多校招企业更看重的是**可培养性**，进入企业后都是按照一张白纸来培养的。

发哥：公众号【嵌入式与Linux】。嵌入式Linux公众号号主发哥，平时不仅分享技术文章，还会给嵌入式入门的同学们做职业分享和讨论，当然了，也有篮球和晒娃。发哥技术也比较猛，开始的STC89C51, AVR, STM32, 到现在的ARM7, ARM9, ARM11, 从裸奔到嵌入式Linux和安卓(Android)系统。之前在一家大厂工作，后面辞掉高薪，出来创业。除了技术比较牛之外，还喜欢打篮球，有机会我要去深圳找他单挑。

投简历前，咨询了下肖遥哥在校招中是选择岗位还是选择公司的问题。肖遥哥说，**岗位重要**，再好的公司如果做的不是你喜欢的擅长的，那么你可能在这家公司也不会长久，不管做那个方向，做那个行业，还是在一个方面要**专注，坚持**。

肖遥哥：公众号【技术让梦想更伟大】力争原创，内容涵盖嵌入式Linux、C/C++/Qt、算法、数据结构、职场感悟等方向。这里有学习路线、经验心得、面试宝典、源码解析、技术精选及经典资料等。号主李肖遥，一个认真做技术的职场老鸟，孵化编程，乐于分享。讲原理，抠细节，究根源。用心写好每一篇文章，专注每一个细节，期待与您一起成长。

在选择Offer的时候也咨询了下波哥，待遇差不多的情况下，考虑公司发展前景以及地域，但要优先考虑**公司行业以及前景**。也要考虑以后**定居**等问题，比如是否能落户，这个和以后孩子的教育也有些关系。但是要找到**兴趣，待遇，地点**等各方面都能满足的工作还是有点难度的。

波哥：公众号【嵌入式客栈】，号主逸珺，高级嵌入式软件工程师，从事嵌入式软硬件开发多年，主要分享Linux系统构建、Linux驱动开发、实战信号处理算法（数字滤波器、谱分析等）、单片机技术、AIOT学习笔记等相关技术内容。

小平哥说，要注意下你应聘的职位和你进去从事的工作内容是否是一样的。在公司如果从事的方向正好是公司的主要业务，你会学到更多的东西。考虑定居的问题，是干几年回老家，还是找个宜居的城市呢？一线城市的压力是会大一点，但是机会也会多一点。如果回老家或者周边，公司的规模和待遇等就会比一线城市差一点。

小平哥：公众号：txp玩Linux.从事过linux应用和c++服务器开发；如果您不知如何准备面试、以及学习路线的迷茫等问题，都可以来找我，一一帮您解答，欢迎大家来“骚扰”我！

朱老师说，你担心的地域问题和年龄问题都不是事，完全可以干几年再跳走。而且，**你要相信个人机缘就是最好的安排**。这种问题没有标准答案，紫光存在不确定性，对于目前的来说，还是先**落袋为安**吧。杭州也是个不错的城市，工作机会也比较多，完全可以支持你未来三到五年的发展。芯片原厂也不一定是最好的，第一份工作只要在**质量上没有问题**，后面还是容易跳槽的。

豆豆姐，第一份工作，跟对人很重要。大公司有成熟的培养体系。刚开始工作，左右对比也很正常。工作本身，**无论哪个岗位都能历练自己。不是赚到，就是学到，关乎己心。工作无非也是借境修心**。自己私下仔细一想，不管是工作还是生活，确实是这么个道理，很赞！



## 2020秋招联发科小米等面经分享

秋招投递公司23家，简历被刷1家。笔试/测评挂掉3家。至今无消息的8家。获得Offer的公司有小米，兆易创新，全志科技，浙江大华，海格通信，京信通信，景嘉微电子，广州朗国电子，北京华大电子，中国长科技集团。已签约浙江大华。

友情提示：公司名字后面的日期代表投递日期，面试批次后面的时间代表面试时长和面试日期。

### 有面试

#### 联发科北京（7.16）

20200805接到通知，0806早上九点半面试。邮件中写的是用Webex Meet，之前都没听过的一个软件，网上找了半天才找到，而且软件没有简体，只好调成繁体了。邮件中写的是等待通知后再连入，大概9.40的时候接到了电话，要我加入会议中。

面试官是个女的，首先让我自我介绍下，然后开始看我的简历。介绍完了直接问项目。

#### 一面（35min, 8.5）

你自己做了那部分？是不是在师兄师姐基础上做的？

不是，师兄师姐之前主要做的是理论研究。我本人所做的是硬件的设计和软件代码的移植。

移植的开源代码，做了那些修改？如何修改的？

主要修改的是硬件的管脚，时钟的配置，SPI总线的调试，芯片通信过程的调试。

上位机部分你说用了卡尔曼滤波，有没有调研过其他的滤波方式？

没有考虑，当时请教了也做这个方向的一些人，他们给的建议就是用卡尔曼就可以。没有考虑其他方式。（其实最主要的是解决问题，能解决问题就可以）

复盘：当时想到的第一个就是卡尔曼，因为卡尔曼在实际工程中应用比较广泛且成熟，效果也不错。当时就拿来试了下，定位精度得到了很好的提升。（定位漂移和抖动40cm左右。漂移和抖动的主要原因就是每次接收到的不止是多个信号叠加的结果，卡尔曼滤波主要是滤除了首径信号以外的其他信号）

做的东西效果怎么样？和其他人做的对标了吗？

定位效果还可以，每秒钟可以定位64个标签。

复盘：业界的评判标准主要有几个方面。

整个工程文件有多少行代码？

具体多少行不清楚，最后编译的hex文件为112k

代码移植过程中遇到什么问题，如何解决的？

巴拉巴拉，通信过程有点复杂，估计面试官没听懂，就没往下问了（其实应该边画图边讲的）。

复盘：解决的整个过程应该描述的再详细一些，重点突出关键部分，这个问题的三个部分都要讲清楚！

项目中实际写的代码量有多少

没多少，主要是硬件的设计和调试，软件的移植，解决问题，修改。

复盘：显然面试官嫌弃代码写少了，这个时候可以说微信小程序的代码自己写了很多，70%以上。从0到1.

项目代码中多线程，多进程是如何运行的

没有用到多线程，多进程。

汇编，C++掌握怎么样？

汇编自学过，可以看懂。C++基本没用过。

复盘：汇编是自学的，C++和C的语法差不多，都可以看懂。

重写strcpy函数？

写完了给面试官解释了下。写对了

将一个寄存器的第三位的值从0改成1

写完了给解释下。写对了

你有什么想问我的？

如果我有幸能进入贵公司，驱动主要负责那部分？

主要还是看你分到那个部门，camera，音视频，IO驱动都有在做的。

什么时候能给到面试结果的答复？

不确定，要先把面试过程的记录交给HR。

## 总结

- 1.我项目上做的是软件+硬件的一个实现，面试官全程在问软件，硬件一点没问。
- 2.在简历中写了自己在写博客，放了一个链接，不知道面试官看没看。
- 3.面试的岗位是Linux驱动开发，全程没有问一点像bootloader，Linux内核的输入子系统，总线设备驱动模型等偏底层的东西。
- 4.女面试官可能都不太懂硬件？全程都是软件，而且自己的项目中写的代码不是太多，主要是修改。面试官还是侧重实际的写代码能力吧。感觉凉了。
- 5.全程35分钟吧。
- 6.总结下，项目考虑再深化下，如何讲解？

## 广州朗国电子科技（8.24）

### 一面（60min, 9.14）

无领导小组讨论。没有标准答案，上网搜索下无领导小组讨论的注意事项，想好自己要扮演什么角色。但是一定不要不说话，要有逻辑的表达自己的观点。

### 二面（25min, 9.16）

HR面，主要问了家庭情况，有没有女朋友，工作地点的问题，能不能接受加班，HR也很坦白的说，公司处在上升期。我们是标准的996。

### 三面（40min, 9.18）

#### 项目

主要针对简历上写的内容来问，项目画原理图，流程图讲清楚，并进行公式推导。

#### 什么是内核空间？什么是用户空间

#### 内核空间和用户空间通信方式

#### 为什么需要uboot？不用行不行？

用uboot的目的是引导内核启动。

我理解的，理论是可以的。把uboot中所做的一些工作写进内核里，板子也能启动。但是很少有人这么做，毕竟内核很庞大，大面积修改难度比较大。

#### volatile关键字

## 总结

9.25号发来邮件，要先签两份协议。这家公司做Smart TV之类的显示设备的，安卓驱动和Linux驱动都有，也有嵌入式应用层的。零食甜点下午茶，10点以后打车报销，每个季度有奖金（0.5-1个月月薪），年终还有年终奖（据说可以拿到18薪），就是加班太多（据说996是标配，忙的时候9107），怕受不了。最后还是拒绝了。

## 浙江大华股份 (9.3)

### 一面 (30min, 9.10)

2020.9.8号做完笔试，9.10下午突然打电话来问是否方便，做个电话面试。

自我介绍

笔试题的建议

笔试题好多关于C++的部分，个人是做嵌入式软件部分的（偏底层）。做起来C++部分有些吃力。希望笔试题可以分嵌入式上层和底层的部分。

项目

问了好久，面试官对我做的项目很感兴趣。

static关键字

修饰变量的话，这个变量的作用域只是本函数，而且如果多次调用函数的话，这个变量只会被初始化一次。修饰函数的话，函数的作用域只是在本文件内。

Arm有几个寄存器？什么是CPSR，SPSR？什么时候用到？

37个寄存器。CPSR是当前程序状态寄存器，存储的是当前程序的状态，比如上下文的一些寄存器内容，程序运行的话就要用到CPSR。SPSR为备份的程序状态寄存器，主要是中断发生时用来存储CPSR的值的。

字符设备有哪些？和块设备有什么区别？如何写一个字符设备驱动？

字符设备有键盘，鼠标等。字符设备和块设备的区别主要是访问方式不同，访问字符设备是以字符流的方式访问的，访问块设备是以块为单位，并且可以随机访问。

以一个LED驱动为例，先定义一个file\_operations结构体，接着编写init函数，在init函数中完成对管脚的映射，register\_chrdev字符设备的注册，class\_create类的注册，class\_device\_create在类下面注册一个设备。exit函数中完成字符设备的卸载，类的卸载，内存空间的释放。在open函数中完成硬件管脚的初始化，在write函数中完成点灯操作。

Uboot启动过程说下？

没有难度。

堆和栈的区别？

1.申请方式，栈的空间由操作系统自动分配，释放，堆上的空间手动分配，释放。2.申请大小，堆的可用空间比较大，栈的可用空间比较小，一般是2M。3.申请效率，栈申请速度比较慢，堆的申请速度比较快。

为什么栈的空间不连续

不知道。

通用学科，你喜欢那个，学得好的。

数学，英语。

数学的那个分支比较感兴趣

矩阵理论。因为在许多问题的深入研究中，基本上50%以上的问题都会转化成矩阵来解决。所以这部分看的比较多。

除了课本学的数学之外，自己私下有没有看其他的关于数学的内容

没有，自己看的比较多的是专业方面的书籍。

专业课中，那个课学的比较好

C语言，操作系统，计算机组成原理，Arm体系和架构

除了课堂上学之外，某个领域有没有深耕，自己研究过，私下看过

Linux内核的源码，操作系统，计算机组成原理，私下都会去花时间去了解，学习

如何学习的？以什么样的方式

我在学习Linux驱动的过程中，会想到一些问题，比如UBOOT的启动过程中为什么会关闭中断，关闭DCACHE，关闭MMU，关闭TLC等。遇到这些疑问我就会去查，解决问题的过程中会想到其他的一些问题，把这些问题记录下来，一一解决。

像Linux内核的话我最近再看一本书，赵炯老师写的Linux0.12源码剖析，这个书以Linux0.12内核为基础，详细介绍了内核的各个部分，虽然看起来比较吃力，但是我也在坚持阅读。

复盘：有条理更好。**学习的形式主要分为两种：**一种是在学习的过程中解决疑问，以解决问题为导向。比如，在移植UBOOT的过程中，我会想到为什么要关闭ICACHE,DCACHE等，关闭中断等（这里可以抛出这些问题，以防后面会问），遇到这些问题我就会记录下来，然后上网去查资料。在解决这些问题的过程中，可能会遇到其他的问题，继续查找相关资料，直到最后都搞清了。**第二**就是系统的学习，如果有比较充足的时间，我会去拿着像Arm体系和架构，操作系统，Linux源码剖析这些书去一点一点的读。去理解这些内容。**以上就是我平时的一个学习方式。**

C/C++那个更熟悉？做过开发吗？

C更熟悉，做过开发，3000行代码的经验。

Linux操作系统熟悉吗？

熟悉，常用的使用都是没问题的。

平时有空了做什么

我会去学习一些新的知识，研究一些底层的东西，比如操作系统，计算机组成原理等。我到现在一直坚持的一个事情就是写博客。每周的话我都会把这周的疑问，这周学习的新的知识去做一个总结，每周都会要求自己去发布一篇博客，对本周的内容做一个总结。

复盘：有空的话我更多的时间还是投入到对于技术的学习中，在学习的过程中我会通过写博客的方式来输出自己的想法。每周我都会去坚持写一篇博客，博客的主要内容就是这周的学习的新知识以及遗留的疑问的解决。

反问

这个算第一次面试吗？什么时候会得到这次面试的结果？下次面试是否会提前通知？

算第一次面试。结果的话这最近两三天会给到。因为人比较多，面试不会提前通知。

## 总结

面试时间35min吧，整个面试过程还是很顺利的，问的问题基本都答了上来，唯一一个关于栈的空间不连续的问题，确实是自己的知识盲区了，后面也补上了。

回想起来。有些问题应该想好再说，注意条理性，问你什么答什么，不要有废话。

## 二面（35min,9.12）

自我介绍

进程和线程的区别

1.进程是系统进行资源分配和调度的一个基本单位，线程是CPU调度和分配的基本单位。2.进程有自己的独立地址空间，线程是共享进程的内存空间的。3.进程切换的开销大，线程切换开销小。4.多线程程序只要有一个线程死掉，整个进程也跟着死掉了，多进程程序中的一个进程死掉并不会对另外一个进程造成影响。

| 死循环有几种方式来写

for(1;1){}, while (1) {}, do {} while(1);

| 看你写的熟悉内核的总线设备驱动模型，讲解下。总线设备驱动模型和字符设备有什么区别？

总线设备驱动模型和字符设备驱动并不是一个平行的概念，总线设备驱动模型是在字符设备驱动模型的基础上套一个外壳，其实内部的驱动编写方式仍然和常规的字符设备驱动基本是一样的，这样做的目的为了隔离BSP和驱动，使得驱动具有更好的可扩展性。

| Uboot如何引导内核启动的？

uboot引导内核启动主要向内核传递三个参数R0, R1,R2, 第一个参数R0，默认为0。第二个参数，R1，CPU ID，告诉内核板载CPU的型号。第三个参数R2，告诉内核映像文件存在什么地方，板子还剩多少内存空间。这些参数的传递都是以tag\_list的方式传递的。

| 主要擅长的开发语言

C语言最擅长

| 左值和右值

左值可写，右值可读。通常，左值可以作为右值，但是右值不一定是左值。

| 数组名和指针区别

数组名对应的是一块内存的地址，指针是指向一块内存地址。数组名对应的内存地址不可以修改，指针指向的内存地址可以修改，更加灵活。数组存放的是数据内容，指针存储的是地址。

| 平常像C++, python这种语言有涉及吗

C++能看懂，会改。自己独立写一个大程序的话不太行。汇编的话是自己学过的，能看懂，会改。python语言没有涉及到。JS/HTML/CSS这些前端的语言是自己在项目中实际用过的，使用没问题。

复盘：先抛出结论，C++ 汇编 JS/HTML/CSS这是我会的语言。然后再描述。

| 之前做的项目都是偏底层的实现，对吧。

不是。能称得上是底层的就是第二个项目吧，移植uboot2012到2440的开发板。第一个项目的话是硬件软件的一个设计开发，没有涉及到底层的东西。

复盘：如何清晰的描述第一个项目。第一个项目主要做的就是硬件的设计，软件代码的移植，是偏上层的，没有和底层相关的技术。

| 你自己考虑的话以后自己是偏向底层的开发对吧

对的

| 是偏向系统呢，驱动呢，内核呢？

我目前考虑的是做驱动开发

| 为什么是驱动开发呢

我觉得做底层这一块比较有意思吧，像做驱动开发的话，我们知道像安卓的camera驱动，音视频驱动这些，都是独立的一块，每一块拿出来都值得研究，我个人也比较倾向于从事有挑战性的工作。目前考虑的是先做Linux驱动，以后如果有可能的话会去做安卓的驱动，再慢慢的到camera驱动，音视频驱动这些。这也算是我的一个职业规划吧

复盘：回答的有点跑偏了，不过不要紧，还是向面试官表达出了自己做这个行业的一个规划。

下次这样回答：因为我觉得做底层的话，可以更清楚的知道我们的程序是如何运行的，程序编译完后是如何在内存里面排布的，我个人对于技术好奇心很重，经常会想一些问题，比如在uboot启动过程中可以把dcache, icache都关掉吗，Linux内核是如何知道我现在所处的环境（运行于那个cpu上，其实也就是uboot和内核参数传递方式，故意说的不清楚，让面试官问你）等等。对于这些问题，我喜欢刨根问底都搞明白。因此我喜欢做驱动开发。

除了上课外，你会看一些什么样的内容呢？

我最近在看的两本书是Linux内核源代码情景分析，赵炯老师的Linux0.12内核完全注释。昨晚我在看的是Linux内核的源代码情景分析中的数据结构部分，包括链表，队列，二叉树等是如何实现的。（很巧了，正好昨晚看了这部分）。此外，每周的话，我都会去学习一些新的知识，把自己的感悟和其他人对于这个问题的看法记录在博客里面。大概就在些把，做的最多的就是写博客，从输入到输出的一个反馈吧。

复盘：先抛出结论。先说，做的最多的就是写博客了，从输入到输出的反馈，巴拉巴拉。

我每周做的最多的事情主要有两个。第一个是学习新的知识，学习一些自己感兴趣的内容，比如我最近在看的书。第二就是写博客，每周我都会写一篇博客，这篇博客的内容主要就是本周学习的知识的总结，或者是之前遗留的疑问的解决。

除了知识学习外，平常还有什么爱好

每周都会跑步3次左右，每周最少打一次篮球。

看你写的博客，11个月，104篇，相当于每个月差不多10篇左右吧。

这104篇中有好多是我在自学Linux驱动开发过程中的一些笔记，随笔。在学习的过程中就顺手记录下来了。其实到后面慢慢发现，一个月内想要产出一篇不错的文章的话，还是要花一点时间的。所以说，我现在对于我自己的一个要求就是每周一篇，保证质量。不像之前那样，以一种记笔记的形式。

这些文章中哪些是访问量比较高的

单链表的增删改查反转等操作，单链表的冒泡，快排，归并等排序，线索二叉树等。

数据结构也是自学的吗？对树熟悉吗？说下那些树，有什么特点

满二叉树，所有根节点都会有两个子节点。平衡二叉树，根节点的左孩子比根节点的值要小，右孩子比根节点的值要大

复盘：平衡二叉树都回答错了。

做驱动过程中，有没有针对某一个点的优化和改善？

犹豫了半天，没有想起来。尴尬！

复盘：最基本的按键驱动啊，由查询方式改为中断方式。Uboot的启动过程中，将重定位的程序靠前存放（链接脚本），保证在4K以内的代码能完成后面程序的复制。

反问

我什么时候能得到二面的结果呢？

一周之内。后面会有HR联系你的。

接下来还有几轮面试

应该还有一轮面试。HR面完了之后会综合评估，给出offer。

## 总结

总体还行吧。70分。面试问的技术问题都是自己在资料中总结到的，二叉树的说错了，不知道面试官发觉没有。面试中很大一部分时间都在聊博客的事情，也算自己擅长的方向。

最后一个问题是回答的不好，实在没想起来，现在想想，举一个差不多的例子就可以了。直接回答不会有太大问题。

## 三面（15min, 9.14）

面试完了有什么收获

有些问题本可以回答的很好，但是由于没什么经验，答得不太好。

目前投了哪些公司

就投了两家，一家是XX，另一家就是大华，大华是面试进度最快的。（其实已经投了几家了）

家庭成员的情况。对工作地点有没有要求？

如实回答就可以。

期望的工资

我说，公司应该都有个统一的标准，按照标准来就好了。HR说你最好还是说一个吧，我说了XXX。

目前有那些OFFER

当时怕压价，就说了有了XX和XXX的offer（实际上都没有啊）。其实这里说漏嘴了，之前说只投递了两家，哪里来的offer？不知道HR有没有意识到。（如果大华的同事看到了，不要打我啊，哈哈。）

反问

什么时候有结果？

一周之内。

## 总结

今天已经是9.21了。仍然没有结果。明天问下吧。

最终的offer发出是在9.30号，看了下也比较满意！

## 兆易创新（9.3）

### 一面（50min, 9.21）

项目问了很久

30min

字符设备如何写，框架

资料总结的驱动框架里面有。很容易答了上来。

LCD驱动框架

资料总结的驱动框架里面有。很容易答了上来。

uboot启动流程

资料总结的驱动框架里面有。很容易答了上来。

SDRAM接口地址

具体地址忘了，但是我知道他是接在bank6的

| 你是北方人，对工作地点有要求吗。

我：没有。我主要看重我在公司做什么。

| 向你介绍下我们部门吧。巴拉巴拉说了很多作品内容后（当面试官说这个的时候，我觉得就有戏了），主要是做指纹芯片的，作品内容大多集中在Android底层和hal层，给客户解决问题要占50%的时间。

其实个人不太喜欢这个工作，主要是有点FAE的感觉。但是当时没说出来。

## 二面 (15min, 9.25)

| 如何了解到兆易创新的

很多比赛都是兆易创新赞助或者以兆易创新的名字命名的，我也参加了兆易创新今年举办的研电赛，我们在研电赛中获得了全国三等奖。

| 目前手上的offer？给你开了多少？

大华，全志。还没谈。（当时也是为了怕压价，说了手上有两个offer）

| 何时发三方？先拉进群

10月底，11月初。

| 工作地点去深圳还是上海？

再考虑下。

| 父母对于工作地点的要求

没有。

| 打算在上海，深圳落户吗

没有。

直接拉进offer群里了，200+人，群里好多电科大，西电的，西交，西工大的，我在群里都不敢发言。

## 总结

个人还是不太喜欢这个工作，主要是做Android底层和Hal层，还给客户解决问题，占用了50%的时间。其实30%的时间是可以接受的，50%有点多了。感谢兆易创新对我的认可，不好意思！

## 景嘉微 (9.13)

景嘉微的面试还是专业的。一天之内三面。每次面试都是两个面试官，二面还是一轮压力面。

### 一面 (35min, 9.23)

| 项目

问的不详细，基本都是原理之类的，解释清楚就好。

| 看你学过数据结构，自学的吗？刷了多少题？问个简单的吧

自己学的。具体多少没算过吗，大概有200道左右吧。

如何判断一个数是不是质数？真不会，忘了质数的概念，想了半天，面试官说，估计你对质数概念不了解，算了。

| 自旋锁和信号量说下

答对了。

二叉树什么时候会退化？什么是平衡二叉树？

不知道什么时候会退化。平衡二叉树就是左右子树度的差值小于1.

uboot启动流程

说了很多遍了。

Cache一致性

不知道这个概念。但是我具体说了下读写Cache的一些注意问题，比如初始化的时候一定要清空Cache之类的。

如何写一个字符设备驱动

init函数, exit函数, file\_operation结构体之类的东西

uboot启动为什么要关闭中断, MMU, DCACHE之类的

非必须关闭吧。uboot的目的就是引导内核启动，而且uboot启动的话只是把需要的打开了，其他非必须的都可以关闭。当然也可以打开。DCACHE之类的必须要关闭，因为SDRAM没有初始化，可能会取到错误的数据。

bin文件和elf文件区别

不清楚。后来在自己总结的面试知识点中增加了这个内容。

什么时候用哈希？哈希冲突如何解决？

对时间要求比较高，对占用内存空间大小要求不高。开放地址法，再哈希法。

## 二面 (45min, 9.23)

项目原理的介绍

写博客的目的

之前也考虑过这个问题，当时说了4点。1.随笔的形式，方便自己回顾。2.好的学习习惯 3.认识了很多业内前辈。4.习惯之后，坚持每周输出。

看你写了很多奖项，好多是校级的，有其他的吗？

有一个国家级的，研电赛全国三等奖。

项目难点

巴拉巴拉。

还有各种针对本人的问题，总之就是压力面，一直否定你这个人

你简历上写了这么多奖学金，你觉得你是你们周围最优秀的吗？为什么？你和别人差在哪？为什么本科毕业不直接参加工作？为什么考研？本科期间都做了哪些事情了？有什么收获？你觉得你是最努力的吗？以前是不是没有努力？为什么在看C primer plus？是基础不好吗？等等这类的问题。（这种问题沉着冷静的如实回答就可以。不要让面试官觉得你人品有问题）

## 三面 (30min, 9.23)

印象最深刻的人

自我介绍

介绍项目

最成功的项目

项目难点

如何克服和学习项目的难点

如果一笔订单需要你的上级审核，但是上级很忙，你会怎么办？

如果这个客户是和我们第一次交易并且时间又很紧急，那么可以考虑找上一级领导。如果这个客户已经和我们有过多次交易并且上级领导实在没时间，那么我也可以审核。

复盘：反思了下，这里其实不该回答“自己也可以审核”这些话的。一般来讲，公司肯定有应对措施的。按照公司的流程走就好了。

## 总结

9.25发来Offer，国庆节后给答复。公司是在长沙，做显卡的，主要客户是部队。不是很想去，国庆节后上班第一天就给了景嘉微答复说不去。

## 全志科技（9.15）

### 一面（20min, 9.23）

项目简单介绍

进程和线程区别

问了很多遍了。

编程题：指针函数，函数的参数为 int，返回值为字符指针

```
1 | char *(*p)(int)
```

宏定义求最大数

```
1 | #define MAX(a,b) (a)>(b)?a:b
```

uboot启动流程

问了很多遍了。

### 二面（40min, 9.26）

一个酒店，和研发部总经理聊了聊受益匪浅。

为什么写博客？

之前也考虑过这个问题，当时说了4点。1.随笔的形式，方便自己回顾。2.好的学习习惯。3.认识了很多业内前辈。4.习惯之后，坚持每周输出。

项目原理，五分钟给我讲明白

边画图边讲。

为什么学Linux？用的什么开发板？啥时候买的？多少钱买的？自学的吗？有人引导吗？学了多久？

周围搞Linux的多不多？

操作系统会吗？计算机组成原理了解多少？

了解一点，没有时间去完整的看。

为什么没有时间完整的看？

白天忙导师安排的事情，还要写论文，改论文，写专利等等，晚上才有时间学习一些内容。当时紧接着又说虽然没时间完整的看，但是操作系统说基本的概念还是都理解的。

说下MMU。什么是MMU？为什么需要MMU？来龙去脉讲清楚

大概讲的是这个里面的一些内容：

#### S3C2410 MMU（存储器管理单元）详述

物理地址到虚拟地址的映射，为了跑大型程序，操作更多的地址

是虚拟地址到物理地址的映射，你搞反了。

紧接着又说，我最看重的是操作系统和计算机组成原理的掌握程度，这些都是嵌入式的基础中的基础。

这些都是放在我计划之中，不过我目前在看源码的框架。

先不要看源码，不懂操作系统和计算机组成原理，看源码会累死。我们之前一个项目要修改内核中的关于调度的程序，我研究进程调度这部分，研究了两周多才修改完。所以，有时间还是补下计算机基础。源码的内容别急着看。

面试官确实很厉害。也和自己说了很多。所以接下来调整下战略思路，**先看操作系统和计算机组成原理**

在校期间，就要把基础打牢，好好看书。

之前面试其他公司，面试官问的关于操作系统的一些内容能答上来，这里我有点膨胀了，居然和面试官说操作系统的基本概念都理解。一下就被面试官问倒了。所以，如果不是特别熟悉，不要和面试官说我精通XXX之类的话，否则，肯定会被面试官问到不会为止。**吸取教训！**

### HR面（20min 9.26）

最后去另一个房间和HR聊了聊待遇和薪资，待遇一般，而且工资组成里面还有20%的绩效，上下浮动。不打算去。而且，因为之前全志也闹过裁员风波，所以试探性问了下HR试用期会裁多少人？HR回答的是我们并不规定具体的指标。我接着又补充说到，那么我是否可以理解为没有上限也没有下限呢？HR说是的。所以，直接拒了。

和HR聊完了，顺便问了下面试官的名字，HR说这是他们XX部门的研发总监。和这个面试官聊天，确实学到了很多，受益匪浅！

## 小米（9.15）

### 一面（40min, 9.21）

自我介绍

首先是自我介绍，本来准备的是三分钟的自我介绍，但是中途被面试官打断了，说面试时间有限，简短一点，要不后面没有时间写代码了。就介绍了下自己写博客的事情。接着面试官也很直接，上来就基础知识开始问。

进程和线程的区别

答对了。

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，它是系统进行资源分配和调度的一个独立单位。例如，用户运行自己的程序，系统就创建一个进程，并为它分配资源，包括各种表格、内存空间、磁盘空间、IO设备等，然后该进程被放入到进程的就绪队列，进程调度程序选中它，为它分配CPU及其他相关资源，该进程就被运行起来。

线程是进程的一个实体，是CPU调度和分配的基本单位，线程自己基本上不拥有系统资源，只拥有一些在运行中必不可少的资源（如程序计数器、一组寄存器和栈），但是，它可以与同属一个进程的其他的线程共享进程所拥有的全部资源。

在没有实现线程的操作系统中，进程既是资源分配的基本单位，又是调度的基本单位，它是系统中并发执行的单元。而在实现了线程的操作系统中，进程是资源分配的基本单位而线程是调度的基本单位，是系统中并发执行的单元。

不全。面试官又提醒了。进程间通信方式有那些，也回答上了。管道，FIFO，信号，信号量，消息队列，共享内存（最快），套接字。

| 僵尸进程听过吗

没有。

| static 和 volatile

答对了。

static主要是改变函数和变量的作用域。volatile防止对寄存器进行优化，使得每条指令都要按照我们写的进行运行

| 两个Linux操作系统之间使用什么命令进行文件的传递？

我不知道是什么命令，但是我可以说下我的想法，我觉得利用HTTPS协议可以进行传输。

| 不是命令，这是协议。

| 数据结构学过吗？说下你知道的排序算法？

在校没有学过，但是我自己私下学过。排序算法：快排，选择排序，冒泡排序，插入排序，堆排序。

| 说下快排的过程，快排的时间复杂度

巴拉巴拉，也答对了。

| 手撕代码

反转字符串中的单词 I am a teacher -> rehcaet a ma I

很基础的问题，A了出来。

| 最后反问

什么时候有结果？

| 不确定，一周以后吧。

## 总结

有好多基础知识没打上来。主要集中在操作系统相关的概念上。确实不知道，接下来要好好补充了！

## 二面 (50min,9.25)

项目问的很细

20min,原理和流程图

typedef和 define有什么区别

1.typedef在编译时处理，具有类型检查的功能；define在预编译时展开，不会进行错误的检查，只是字符的替换。2.define没有作用域的限制，typedef有自己的作用域。3.typedef定义指针的别名时，别名可以连续定义两个指针变量。define定义指针的别名时，使用这个别名连续定义两个指针变量会报错。

数组下标可以为负数吗

没见过，应该可以吧。

不能用 sizeof () 函数，如何判断操作系统是16位，还是32位

16位系统中，int变量的范围-32768到+32767,32767+1变为-32768。可以利用这个特性来判断。

IIC如何发送一个数据？IIC时序图画下。IIC芯片有哪些？

回答的还可以。

用户栈和内核栈是同一个区域吗？有什么区别？

不是。用户栈和内核栈是两个独立的区域。内核栈保存的是内核态程序运行的时候相关寄存器信息，用户栈保存的是用户态的内容。

用户空间和内核空间的通信方式？

1.API函数，Copy\_from\_user, get\_user等。2.proc文件系统 3.mmap系统调用 4.使用文件

中断的响应执行流程？听过顶半部和底半部吗？讲讲

cpu接受中断->保存中断上下文跳转到中断处理历程->执行中断上半部->执行中断下半部->恢复中断上下文。

顶半部执行一般都是比较紧急的任务，比如清中断。下半部执行的是一些不太紧急的任务，可以节省中断处理的时间。

写过哪些驱动？讲下LCD驱动如何编写？

巴拉巴拉，问了很多遍了。

手撕代码

给定一个数组，找出和为s的数字。二分查找，A了。

反问

如果有幸进公司，主要负责那些方面？

IO驱动，音视频驱动，内核的优化移植都有。看个人兴趣和过往经历的匹配程度。

多久出结果呢？

月底前应该会给

### 三面 (10min, 10.15)

| 家庭情况

| 期望薪资

| 有没有女朋友？工作地点是北京还是深圳？

北京。

### 总结

发下offer已经是10.20号了，太晚了，三方早已经寄走了，综合考虑，感觉性价比不是很高。如果冲着小米的平台去，确实还不错。其实心里有点小后悔。不过也就这样了，注定与小米无缘。

## 中国长城科技集团 (9.15)

| 项目

15min。

| 你知道的Linux指令有那些

ls, ps, rm, cat, mv.

| busybox是什么？

缩小版的Unix系统常用命令工具箱。主要包含了一些常用的Linux指令，环境等。

| 什么是根文件系统

根文件系统上是内核启动时所挂载的第一个文件系统，内核代码映像文件保存在根文件系统中。

| 为什么写博客？

之前回答过了，四点。

| 反问

这个岗位主要负责什么？

| 你的岗位是Linux系统工程师，主要是负责操作系统的优化和移植。

### 总结

9.25号微信告诉我面试通过了，问我是否签约，综合考虑了下，拒绝了，抱歉！公司其实还不错，在长沙，一年保底16薪，包食宿，长沙人去蛮适合的。

## CEC子公司-北京华大电子 (9.15)

面试官是两个人，一个HR，一个40多岁的主管。没问很深的技术问题，主要问了项目和写博客的事情。他们主要做WIFI芯片和SOC的，更多的可能是应用层的开发任务。待遇给的还可以，但是觉得平台小了，拒了，抱歉！

## 京信通信 (9.16)

### 一面 (10min, 9.21)

具体问了什么忘记了。但是都不难，很基础的。（有史以来最短面试，10分钟就完了）

## 二面 (15min, 10.19)

早上九点半进入腾讯会议，发现还不止一个人。几个面试者就随便聊了几句。大家都不知道接下来是技术面还是群面，还是HR面？提前也没有通知面试的内容。后来进来一个女的说，大家可以耐心等待，有兴趣可以一起听下，我们挨个面。每个人大概20min左右。

第一个人面试完，HR让第二个人来，我看大家都没反应，我就第二个了。

| 自我介绍

| 你做决定的过程是怎么样的？

| offer情况？期望薪资？工作地点？

| 反问

大概多久会发offer？多久之后会谈薪？

| 一周之内

## 总结

感觉面试不是很正规，电话面试官迟到20min，而且也没问几个技术问题，10min就完了；一面和二面战线拉的太长了，不知道其他人有没有，反正我是这样。也可能是太菜了，后面才被捞起来的。10.23HR打电话确认是否接受offer，拒绝了，抱歉！

## 海格通信 (9.27)

### 一面 (20min, 9.27)

群面，五人一组围一圈，每个人做自我介绍，HR会单独提问，大概内容就是关于哪里人，家庭，父母工作，对海格了解有多少等等。

### 二面 (25min, 9.27)

| 项目

| 你知道的常用Linux命令

mv, ls, cat, ps, mkdir, touch, find。

| IO多路复用

不知道，但是我知道IO管脚的复用，巴拉巴拉解释了下。

| 在Linux上编写过那些程序

基本的外设驱动都写过，led，按键，lcd，Nand Flash等。

| 程序的编译过程分为几部分

预处理，编译，汇编，链接

| 什么是4字节对齐？为什么需要对齐？

资料总结的有。

| 如何求一个结构体成员变量的地址

正好前几天写了一个博客。

[内核中container\\_of宏的详细解释](#)

| 函数指针和指针函数

资料总结的有。

| 什么是野指针？如何避免？

资料总结的有。

| sizeof和strlen区别？

sizeof是运算符，在程序编译时就已经确定了；strlen是函数，程序运行时才能计算。

| int a[5] = {1,2,3,4,5},sizeof(a) = ?

20。

| 快速排序的思想？时间复杂度？

理解为打扑克整理牌，O (nlogn) 。

| 哈希表是什么？如何使用？

答对了。

| 反问

如果有幸进入贵公司，主要负责哪方面？

| 负责调试和维护基本的外设驱动，配合硬件工程师进行新平台的开发。

什么时候可以得到这轮面试的结果？

| 明天。

### 三面 (10min, 9.28)

党委副书记面试，全程很放松，就是普通的聊天，自我介绍，哪里人？为什么选择来南方读书？对于海格了解多少？如何看待加班？能接受加班吗？职业规划等？

### 四面 (15min, 9.28)

大boss面试，好像是个总经理，自我介绍，介绍下项目，项目几个人？你负责那部分？除了这个项目研究生期间还有那些团队合作的项目？你是扮演什么角色？你导师研究那个方向的？为什么你和导师的研究方向不一样？你觉得研究生期间导师对你的帮助大不大？女朋友哪里的？做什么工作的？有考虑过在广州定居吗？为什么？等等吧，还有些想不起来了。

### 总结

整个面试感觉浓浓的国企风味，很看重人的综合素质，对技术要求感觉不是很高。9.29约去酒店谈薪，与其说是谈薪，不如说是直接告诉你。没有argue的余地。统一打包价。拒绝了，抱歉！

## 简历被刷

### oppo (8.23)

不得不说，oppo的简历卡的是真的严格。全是人工筛选。不止要看你的学校，还要看你的项目经历是否匹配。我这被刷了也很正常。

## 笔试/测评挂

### 海康威视 (9.1)

测评居然挂了，很可惜。

### 乐鑫 (8.18)

之前乐鑫的HR说，提前批投递不影响秋招。所以很早就投递了乐鑫科技。乐鑫是我第一家笔试的公司。当时笔试题目是三道编程题。巨难！比华为的笔试题难度都大。而且，第一次用牛客的笔试系统，最基本的如何读取输入输出都不会，于是笔试直接挂了。

9.8号找HR确认说，提前批挂掉的会自动推到正式批。但是我等到9.19号也没有收到笔试。后来HR让我去官网看自己的简历的状态，结果显示人才池！正式批都没有笔试就直接人才池了？什么操作啊。。说不影响有点假，其实还是有影响的。可能提前批笔试挂掉的是不会有办法参加正式批的。（自己猜测）

### CVTE (9.16)

C厂的笔试挂的就很玄学了。当时对C厂还是抱有很大期望的。笔试题目中规中矩，大题也都A了出来。选择填空做的正确率应该有80%。但是最后笔试莫名其妙挂了。我同学投研发岗的无一例外笔试也都挂了。难道100分的题目，90分及格线？以后做C厂笔试题，大家要格外小心，尽量还是要高正确率！

12.9号，接到CVTE HR打来的电话，说在补录，约个时间面试。手上没有三方了，拒绝了。

## 没消息

### 寒武纪 (9.3)

9.16号笔试之后就没消息了。三个大题，A了1.5个。(其实这个公司也不太了解，看到了有岗位就投了，据说是AI四小龙的老大)

### 华为 (9.10)

8月份华为在线上做专场宣讲，当时的宣讲会每个部门都做了介绍，给了微信群，我也加了几个群。加到群里后HR会主动加你好友，直接打语音电话解决你关于投递岗位的疑问，很是热情。每个部门的HR都会说我们这里有很多HC，建议投递我们部门。

在选择部门时，主要考虑以下几个方面。鉴于今年的情况，华为的消费BG是不考虑了，毕竟芯片断供，手机业务也大打折扣，很大可能是缩招的。无线部门，也不考虑了。神终端，圣无线的名号不是吹的。而且，华为的无线部门成绩要求也比较高，10%左右？（别的部门的HR说的）。最后考虑智能车BU是新成立的，而且智能车BU的HR也极力推荐我们投递。于是就投递了智能车BU。

投递之后，在网上搜索关于这个部门的情况，看到了一条消息，大概意思就是说，这个部门的人数不会太多，要小而精。瞬间感觉到有点被车BU的HR坑了。

9.14笔试，大题A了第一道。后面两个没做。9.16发的测评链接。9.19问所投递部门的负责人说，第一批面试暂时截止了，后面还可能会有HC，名额不会太多，但是到了十月以后了。十月份再给HR发消息，HR已经不回复消息了。相反，投递其他部门的同学，在9.19-9.24这周都安排了面试。

怎么说呢，怪自己投递晚了吧，而且加上华为今年形势紧张。当时主要考虑，还没有准备好，而且华为基本是一天结束三面。没有准备好过去当炮灰也不值。所以投递有点晚。现在准备好了，但是已经错过了时间节点，有点可惜。

[华为技术面试的准备和经验分享【完全攻略，已签约】精](#)

[各位大佬，求华为面试手撕代码](#)

[华为这波操作以后，中兴可能成了最大赢家](#)

[大半夜的睡不着谈谈华为秋招的看法。。](#)

更多关于华为招聘的内容，可以去牛客网搜索下。

11.30号，12.3号下午，分别接到了华为Cloud&AI和消费者BG的HR打来的电话，问了下我的基本情况，让我继续投递简历，考虑到已经没有三方了，就拒绝了。而且华为的人才池海了去了，这个时间节点还在拉人面试，猜测可能是HR的KPI还没完成吧。

### BOE (8.25 & 9.12)

很奇怪，提前批投了没消息，正式批也不给笔试。太难了。

### 恩智浦 (9.15)

听说只招211/985？

### 瑞芯微 (9.17)

挺想去这个公司的，但是也没消息。

## 紫光展锐 (9.18)

主要是投递晚了。而且，很搞笑，只发了笔试短信通知笔试，在笔试当天却没有收到笔试链接。后来问了HR说：不好意思，这是我们第一次全网招聘，系统BUG了，后面我们会统一处理的。后面又发邮件问了几次HR什么时候安排笔试，回复也是很官方。太难了，错过了九月的最后一批笔试。

11.1号晚上12点发短信让11.2号下午直接参加面试，考虑到已经没有三方协议了，拒绝了。

## 联发科成都 (9.18)

投递晚了啊，联发科成都那边可能是不缺人了，一直没消息。

## 小马智行 (9.18)

随便投的，后来才知道这个公司基本只招985。

## 总结

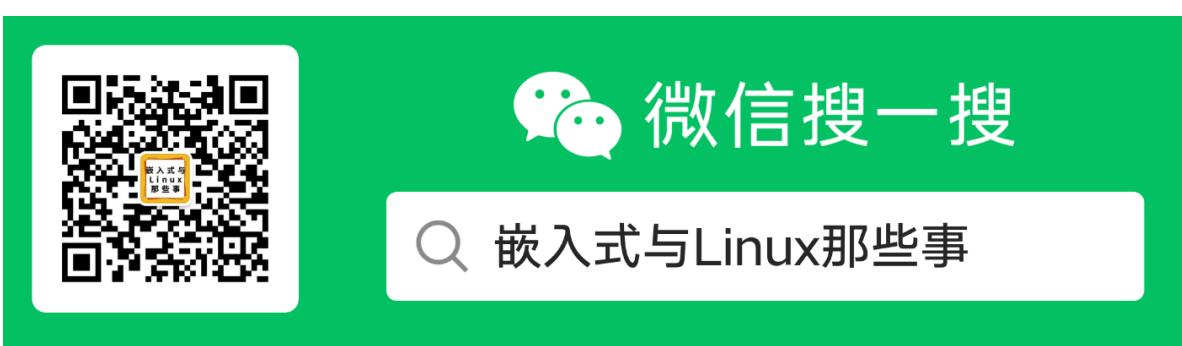
整个秋招还算是比较顺利的，虽然九月初的时候有段时间心态有点崩，但是及时调整过来了。感觉秋招也很戏剧性，六分运气，四分实力。运气好，和面试官聊的顺利，说不定还能拿到sp, ssp。但是，也不能一味的去靠运气。机会总是留给有准备的人，机会没有到来时，要认真准备。当机会来临，我们要好好把握。

最后祝所有看到这篇文章的小伙伴，在秋招中都可以拿到心仪的offer。

有任何问题，欢迎加我微信，一起学习交流。



养成习惯，先赞后看！如果觉得写的不错，欢迎关注，点赞，在看，转发，谢谢！



## oppo和海康嵌入式软件工程师面经分享

哈喽，大家好。分享下春招面试的两家公司。海康和oppo是三月初的时候投递的，虽然，秋招已经签了。但是，我还是想充分利用下应届生的身份，搏一搏更大的公司和更高的待遇。

海康缺口比较大，一直在招人。oppo春招不像秋招那样，卡简历卡的那么严格，普通学校的学生也有了很大的机会。

两家公司的面试都不太难，问题都是提前准备到的。所以，笔试面试的一些八股文，该背还是要背的。毕竟，面试的时候，答不上来等于不会。

八股文的内容，可以看下这篇文章[嵌入式软件工程师笔试面试指南-C/C++](#)。所有关于笔试面试的内容，我都会同步在github (<https://github.com/ZhongYi-LinuxDriverDev/EmbeddedSoftwareEngineerInterview>) 上，**大家可以star下**，以免错过更新。大家如果想提前准备的，可以加我好友，我会把旧版本先给你。

### 海康

一面 (3.23,35min)

自我介绍

## 项目介绍

你做的这个项目遇到了那些问题，如何解决的？

移植uboot，只做了移植吗？

用的那个文件系统？

移植过程中，网卡驱动做了那些工作？

写过那些驱动，讲一个你熟悉的？

写驱动过程中，遇到过什么问题，如何解决的？

对网络设备驱动有了解吗？

你有什么想问我的？

## HR面 (3.30, 25min)

### 自我介绍

作为北方人，你对南方的饮食，气候习惯吗？

为什么写博客？

读研也挺忙的，写博客不会影响你日常工作吗？

团队协作能力怎么样？举个例子？

周围的人是如何评价你的？

三个词概括你自己

## 签约答疑 (4.9,30min)

4.9号下午2点，腾讯会议，HR主要介绍海康的主营业务，福利待遇，晋升通道，公司文化等各个方面。

当天下午六点，另一个负责薪资的HR打电话告知了薪资。待遇不满意，拒了。

## OPPO

### 一面 (4.2,20min)

#### 自我介绍

三个项目，问的很详细

SPI是什么？有几条线？几种模式？

使用IO模拟过SPI吗？

堆和栈有什么区别？

调用函数时，有那些内容需要压栈？

有什么要问我的？

4.4号打来电话，告知一面通过，清明节后安排二面。

## 二面 (4.9,40min)

自我介绍

项目

uboot启动流程

uboot启动前还需要做那些事情？

uboot启动时使用的是物理地址还是虚拟地址？MMU要开启吗？

x86汇编和Arm汇编有什么区别？

介绍一个你熟悉的驱动程序

操作系统学过吗？自旋锁和信号量有什么区别？

Linux系统的启动流程

学过那些专业课，哪些学的比较好？

你有什么想问我的？

## 三面 (4.12,15min)

自我介绍

HR介绍部门

和那个公司签约了？

对工作地有什么要求？

你有什么想问我的？

## Offer

4.15收到录用排序的通知，4.20发下正式offer。影像部门，在深圳，C/C++岗位，工作内容主要是相机驱动，图像调测等，进去后随机分岗。

oppo软工和影像加班是真的多，11点后下班是常态，每周六天，每年强制淘汰5%。薪资待遇给的白菜价，而且，没有argue的余地。综合各个方面考虑，拒绝了。

## 总结

春招到四月底了，基本算结束了。我应该也不会再面了。本来，还挺想去紫光展锐天津的。但是，由于秋招的时候拒绝了紫光展锐，春招时，紫光展锐直接不给机会了。

对于笔试面试，做个简单总结吧。更详细的内容，可以到github查看笔试面试的一系列内容。（[点击阅读原文可以直达](#)）

1. **扎实基础，准备好八股文。** 面试的重点，主要是**基础知识和综合素质**的考察。
2. **简历重点突出。** 简历上的内容，一定把你想要面试官问的内容放在一眼可以看到的地方。要对写在简历上的每一个字负责。可以适当吹牛，但是，你要保证可以自圆其说。
3. **心态。** 无论是秋招，还是春招，心态一定要调整好。像我本硕都是双非的，简历投出去没消息，被刷掉很正常。被大公司刷了KPI也很正常。
4. **多投简历，充分利用等待期。** 不要在一棵树上吊死。投完简历，简历并不一定会被及时处理，中间还有很长的等待期，可以利用这段时间面试其他的公司，让自己保持在一个很好的面试状态。
5. **及时跟进，及时反馈。** 一般来讲，面试结果一周内都会给到。如果在面试完一周后没有任何消息，可以尝试给负责的HR打电话询问具体情况。

# 读者秋招总结

## 秋招总拿SP是什么样的体验

### 个人情况简介

本科双非，机电学院某小众专业，要学的课程有机械/计算机/管理等，比较杂。后面看本专业实在不感兴趣，就决定考研计算机。读研在某985计算机学院，做的嵌入式方向。

本科的时候先是学习了C语言，然后接触到单片机，觉得自己的兴趣点在这里。

大二时加入了学院的机器人团队，开始学习STM32，参加过一些机器人比赛。大三退出团队以后，比较迷茫，不知何去何从，思考很久以后决定考研计算机。

读研期间，研一期间是用MSP430做导师的横向项目，研二下开始学习FPGA，使用FPGA辅助单片机，帮助师兄搭建论文原型。

我对底层比较感兴趣，用FPGA复现了一个MIPS流水线CPU。总体来说，自己的知识偏向底层硬件开发，对于Linux/RTOS这些，自己一直想学，没有项目做，学得比较一般，也成为了我面试过程中的短板。

### 秋招投递及offer情况

今年嵌入式的秋招7月份就已经如火如荼进行了。不要觉得时间太早还没准备好，要在实战中提升自己，正式批一般都会更难，而且提前批更容易拿到sp。

我投的基本是嵌入式软件开发岗，有一些是驱动开发。拜美国制裁中国所赐，今年IC岗随便白菜价就是40w年薪，嵌入式软件开发近水楼台先得月，很多同学都能拿到35w以上。这在以往是不可想象的。

### 投递过的企业

大华，小米，vivo，oppo，乐鑫，联发科技，汇顶，大疆，蔚来，海思，全志，壁仞，海康，锐石，芯动，联影微电子，紫光展锐，星辰，瑞星微，荣耀，禾赛，诺瓦，cvte，燧原，寒武纪，地平线，科大讯飞，德州仪器，中科芯，中兴，阿里云，美团，arm，360，美的，哈罗出行，英伟达，恩智浦，小鹏，中国电信，中国移动。

### 拿到的offer

我拿到的基本都是sp或以上级别，具体如下(城市为深圳杭州东莞广州)：

联发科技，30w-，sp，签字费9w多。

大华，30w-，大白菜。

小米，30w~35w，sp。

oppo，30w~35w，应该是白菜。

星辰，30w~35w，sp。

汇顶，40w+，sp。

大疆，45w+，sp。

荣耀，40w+，15级。

小马智行，hr说面试通过，当前还未开奖。

华为海思，严格来说还没拿到，只是面试通过，面评还可以，泡池子很久了。

cvte, 通过所有面试, 但是一定要去实习一个星期才能拿到offer, 折腾谁呢。

## 秋招准备

### 项目

#### 本科阶段

在本科的机器人团队里面, 学习了STM32, 并且用它来做过中国机器人大赛。比赛内容是武术擂台机器人, 我做的部分主要是编写陀螺仪的驱动程序, 检测机器人的姿态, 在跌倒时可以站立起来。

本科的这个项目面试的时候问得不多, 问到的话, 都是一些非技术性问题, 比如主要负责什么工作, 从中学到了什么。

#### 研究生阶段

研一的时候做了两个MSP430相关的横向项目。项目大同小异, 业务功能也不复杂, 主要就是通过传感器采集环境中的某些数据, 然后通过4G模块/RF模块将数据传出去, 有一定的低功耗要求。

这两个项目本身功能不复杂(直接裸机while(1)循环搞定), 所以我第一次面试vivo把它们写到简历里去就直接翻车了。

估计面试官问完我的项目以后, 内心想法是"就这? 你也好意思拿来面试? "

所以后面我调整策略, 就没有把MSP430的项目写入简历里去了, 简历里主推我的FPGA项目。

但是这两个MSP430项目虽说功能简单, 也是我呕心沥血做出来的, 不能白白浪费了精力。后来策略就变成面试的自我介绍环节口头叙述这两个项目。

在本科那会, 因为STM32市面已经很多现成的资料了, 单片机的初始化代码/传感器的驱动代码大多都已经有别人做好了, 我们只需要拿来用就好, 这样导致我一直没有掌握从0开始写代码的能力。

在研一的时候, 我想好好利用这些项目锻炼自己的能力, 实验室用到的MSP430型号比较新, 我就自己去啃数据手册/用户手册, 故意不去参考官方的例程(实在写不出来再参考), 把单片机的代码从0写了出来。

对于传感器部分的代码大致也是这样的编写过程, 自己去死磕数据手册, 过程很累(有些数据手册提供的信息有误), 但是感觉是真正锻炼自己的地方。

印象深刻的一次是为了方便自己调试, 决定要在MSP430里使用简化版printf()函数, 然后去找TI官方的编译器文档, 了解MSP430架构的汇编指令集和C语言到汇编的调用规则, 最后把printf()实现出来也是挺有成就感的。 (现在其实忘得差不多了哈哈)

所以我觉得我研一其实做了挺多事情, 但是无奈项目的功能实在简单, 面试吃亏。

我的策略是, 在自我介绍时这样说: "研一做的这两个单片机项目功能比较简单, 它们的功能是balabala, 所以我没有写到简历里去, 但是整个项目基本是我一个人完成的, 这款单片机的资料也比较少, 我就锻炼自己阅读官方第一手资料, 从0开始写代码的能力, 也锻炼了我查找信息的能力."

大概就是这样, 说完以后就会切入到我的FPGA项目.

到了研二下学期, 也就是2020年年初那会, 全世界疫情闹得正凶, 我也只能呆在家。还没开学, 师兄就叫我学一下FPGA, 后面要做项目。

当初学FPGA也比较痛苦, 我看原子的教程, 跟着他写代码, 好像代码会写了, 但是面对师兄的项目完全没有感觉, 不知从何开始。

关于FPGA的教程, 我比较推荐野火的, 无奈野火的在那时还要等大半年才出来, 它们的教程一出来我就下载了电子版, 读起来确实好舒服。

我做的FPGA项目(用单片机来辅助调试了)一个是关于热插拔的, 一个是关于无线信道带宽利用率的。这两个项目就成为了面试中的主打内容。复盘这两个项目也用了好久时间, 和师兄讨论有些地方为什么要这么做, 项目的意义(从工程意义上和从科研意义上)何在, 总之是想了好多面试官可能会问到的问题。

因为这两个项目整个工程都是我一个人完成的(包括协议设计, 所有代码的编写, 仿真测试, 下板子做实验), 所以和面试官也比较有得聊. 不过有的面试官很赞赏, 有的觉得我的项目一般般, 有的觉得其实我没干什么。

其实说句心里话, 我觉得这两个项目确实锻炼到我, 但是从实际意义上来看, 我觉得这两个项目意义是不大的, 它们最大的意义是帮助我拿到了offer。所以说服面试官相信你自己也不太相信的事情, 也算一个挑战。

话说回来, 当如让我学FPGA做项目我是心里不太愿意的, 我心里真正想学的是嵌入式Linux, 软件的带操作系统的东西!

可是师兄哄着我做, 我想着我不做他也不好毕业(他是博士师兄), 算了吧就这么做下来了。

学完FPGA后, 想着别自学, 又自己去做了一款流水线CPU(自己动手写CPU.雷思磊著), 放到FPGA上跑。哎, 没想到误打误撞, 师兄还帮助我不少, 让我秋招有实在的项目, 过程也比较顺利。其实我们实验室其他同学都没什么拿得出手的项目, 导师的项目都是一些简单的curd, 最后它们拿的是本科的项目包装了一下, 实验室其他同学找的是互联网工作, 秋招过程并不顺利。不管怎样, 还是要感谢我的师兄吧!

这里截图放一下我的简历吧。理工科的简历不用太花哨, 把你会什么, 做过什么写上去就好。

我做简历也很简单, 先在word文档搞一个表格, 然后在里面填内容, 调格式, 在表格的限定下格式不会乱, 做好以后就把表格设定成隐藏线段。

**专业技能**

- → 掌握基本的 LINUX 驱动及应用开发技能。◦
- → 熟悉使用 STM32、MSP430 开发单片机程序。◦
- → 具有一定的硬件知识，能看懂原理图、用户手册及数据手册并编写底层硬件驱动代码。◦
- → 对嵌入式操作系统 RT-Thread 较为熟悉。◦
- → 熟悉 ARM 与 MIPS 体系结构，掌握流水线 CPU 模型。◦
- → 有较为丰富的 FPGA 开发经验。◦
- → 熟悉常用的数据结构及算法(树，链表，经典动态规划，排序等)。◦

**项目经验**

|                        |                                                                                                                                                                                                                       |               |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| <b>2020.10~2021.3◦</b> | <b>高带宽利用率的无处理器终端研究◦</b>                                                                                                                                                                                               | <b>项目开发者◦</b> |
| <b>项目描述◦</b>           | 以通信为核心的物联网边端体系中(突出特色为终端无 MCU)，对通信信道资源存在大量的冗余消耗，本研究从带宽利用率优化及传感器原始数据处理两方面进行优化，设计并实现一套网关与终端的高效通信协议。◦                                                                                                                     |               |
| <b>负责事宜◦</b>           | <ul style="list-style-type: none"> <li>• → 参与了协议的设计，该协议主要规定了对终端传感器的调度以及数据的压缩传输。◦</li> <li>• → 独立负责整套协议实现，使用 FPGA 实现协议在物联网终端的部分，包括对协议帧的解析、与传感器的交互以及数据的压缩传输，使用 STM32 模拟协议在网关的部分，包括生成并传输协议帧，解析并恢复终端回传的传感器数据。◦</li> </ul> |               |
| <b>2020.4~2020.9◦</b>  | <b>实现易重构、即插即用的物联网终端框架◦</b>                                                                                                                                                                                            | <b>项目开发者◦</b> |
| <b>项目描述◦</b>           | 传统的物联网终端架构与 MCU 高度耦合，一旦终端设备有所更换，就不可避免设计软硬件的修改，并且 MCU 会带来极大的能耗。提出一种全新的物联网终端设计框架，本框架去除了控制核心 MCU，将对终端的控制上浮到网关，并设计一套协议，以实现终端传感器的快速更新与即插即用。◦                                                                               |               |
| <b>负责事宜◦</b>           | <ul style="list-style-type: none"> <li>• → 设计一套网关与终端传感器交互的协议，协议实现用户免编程、即插即用使用传感器。◦</li> <li>• → 独立负责整套协议实现，使用 FPGA 实现协议在物联网终端的部分，包括对协议帧的解析、与传感器的交互以及对热插拔事件的处理，使用 MSP430 搭接协议在网关上的功能进行验证。◦</li> </ul>                  |               |

## 专业知识

我主要准备了：

1. 单片机，主要看野火的STM32教程。
2. RTOS，跟着野火的RT-Thread那本电子教程写了一遍RTT的内核代码。
3. 嵌入式Linux，主要B站学习韦东山的嵌入式Linux快速入门课程，他的裸机视频(imx6ull)我也看了，校招的话，其实这些知识储备基本够用。
4. 学了FPGA，还准备了CPU流水线的知识，在面试ARM的时候用到了。

这里跑个题，说一下韦东山和野火的教程吧

感觉韦老师的教程只适合快速上手，很难深入学习，他的快速入门教程我反反复复把驱动部分看了4遍，面试过程中很多问题也能回答上来，但总感觉自己还没有学到原理性的东西。

后面尝试去看驱动大全，感觉有点跟不上。我觉得韦老师Linux教程最大优点也是最大的缺点，快速入门。

比如说，写第一个hello驱动，他会告诉你，申请，设置，注册，创建几步曲，让你很快看到现象结果，先有成就感，但是他会告诉你class\_create()不重要，写下来就好(这种教学观点一直贯穿全程)，这个过程下来，我确实很快写出了第一个驱动，第一个平台驱动，第一个设备树，但感觉就是亦步亦趋，我跟着他摸石头过河，他没带我摸过的石头，我就不知道该怎么办了。

最近秋招结束，我空下来去翻了一下野火的Linux教程，野火的教程风格就是，一步步来，从上到下，带你认识内核的生态，我觉得你去学一门新的技术，生态感真的挺重要，这能让你置身这个世界中，然后你就不会去疑惑这一步为什么这样做，而是觉得自然而然就要这样做。

这一部分是我觉得野火教程远远好于百问网教程的地方，最近我也跟着野火的教程重新学习驱动开发，感觉收获了很多。但是缺点是，野火的视频教程真的在劝退初学者啊，基本是“深度优先”的方式去讲解知识，就算我前面学习过韦东山的，看起来也头晕，后面我就直接不看视频了，直接看他家的pdf，发现写得还不错，有些小瑕疵，凭着前面的基础，很快能克服（但对于初学者来说，可能是一道坎）。

但是无论如何，还是要感谢韦老师能做这么多免费视频，帮助我秋招找到了工作，后面也想购买他的驱动大全来学习一下。

学习的过程中，要多想，我不否认韦老师和野火的教程都很优秀，但是教程里有些地方其实没有解释清楚的，有的地方是错误的，如果不搞清楚，往往就会成为面试中的一个坑。

比如我在看韦东山的驱动大全的时候，讲到arm内核如何实现原子操作，基本原理是使用ldrex和strex指令。

韦东山的教程是这样描述的：

在 ARMv6 及以上的架构中，有 ldrex、strex 指令，ex 表示 exclude，意为独占地。这 2 条指令要配合使用，举例如下：

- ① 读出：ldrex r0, [r1]  
读取 r1 所指内存的数据，存入 r0；并且标记 r1 所指内存为“独占访问”。  
如果有其他程序再次执行“ldrex r0, [r1]”，一样会成功，一样会标记 r1 所指内存为“独占访问”。
- ② 修改 r0 的值
- ③ 写入：strex r2, r0, [r1]  
如果 r1 的“独占访问”标记还存在，则把 r0 的新值写入 r1 所指内存，并且清除“独占访问”的标记，把 r2 设为 0 表示成功。  
如果 r1 的“独占访问”标记不存在了，就不会更新内存，并且把 r2 设为 1 表示失败。

假设这样的抢占场景：

- ① 程序 A 在读出、修改某个变量时，被程序 B 抢占了；
- ② 程序 B 先完成了操作，程序 B 的 strex 操作会清除“独占访问”的标记；
- ③ 轮到程序 A 执行剩下的写入操作时，它发现独占访问”标记不存在了，于是取消写入操作。  
这就避免了这样的事情发生：程序 A、B 同时修改这个变量，并且都自认为成功了。

我一开始也没想太多，觉得内核这样实现原子操作很精妙，可是后来和同学讨论的过程中，发现这种解释是行不通的，我们设计了一种运行场景，拿到百问网上提问（我百度搜，竟然没人觉得ldrex和strex有任何令人疑惑的地方），最后韦老师也思考了好久，才说自己的教程搞错了。具体是什么我截图下来：

关于ldrex, strex的一点思考和疑惑.

ldrex r0, [r1]

将r1所指内存的数据存入r0, 并且标记r1所指内存为"独占访问".

那么可以简记ldrex为一个原子操作: r0=[r1], memFlag=1.

strex r2, r0, [r1]

如果memFlag值还是为1, 则把r0写入内存[r1]中, 并且令memFlag=0, 然后令r2=0.

那么可以简记strex为一个原子操作: if(memflag) [r1]=r0, memFlag=0, r2=0.

教程中举的一个成功原子操作的例子:

- (1) A在读出, 修改某个变量时被B打断了.
- (2) B完成读改写的操作, memFlag清零.
- (3) A准备完成写入操作, 发现memFlag为0, 于是取消写入.

这就避免了程序A和B同时修改变量, 并且都自认为成功.

#### 但是, 考虑这样一个操作序列:

有三个线程A B C, 对初始值为0的变量a执行++操作, 我们再简化一点, ldrex简记为set(memFlag), strex简记为clr(memFlag).

- (1) 线程A: set(memFlag), 然后执行加1操作. 然后被线程B打断.
- (2) 线程B: set(memFlag), 然后执行加1操作.
- (3) 线程B: 发现memFlag为1, 可以写回内存, 然后clr(memFlag). 此时内存中a为1. 然后被线程C打断.
- (4) 线程C: set(memFlag), 然后被线程A打断.
- (5) 线程A: 发现memFlag为1, 可以写回内存, 然后clr(memFlag). 此时内存中a仍为1, 但是我们的预期效果是2.

那么其实线程A的操作被线程B打断了, 后面线程A发现memFlag为1其实是线程C的操作结果, 线程A也将值写回内存, 第(5)操作失败后, a的值为1, 但是预期值应该为2, 并没有实现原子操作.

下面是韦老师的答案:

### 🏆 最佳答案

2021-08-17 11:14

终于搞清楚了, 任务切换、中断发生时, 会清除"独占访问"标记。

- (1) 线程A: set(memFlag), 然后执行加1操作. 然后被线程B打断: 切换线程B时, meFlag被清除
- (2) 线程B: set(memFlag), 然后执行加1操作.
- (3) 线程B: 发现memFlag为1, 可以写回内存, 然后clr(memFlag). 此时内存中a为1. 然后被线程C打断, meFlag被清除
- (4) 线程C: set(memFlag). 然后被线程A打断, meFlag被清除
- (5) 线程A: 发现memFlag为0, 不可以写回内存, 再次循环后成功, 此时a为2
- (6) 切换到C, meFlag被清除, 当它写内存时失败, 再次循环后成功。

嵌入式与Linux那些事

还有一个例子, 我同学面试过程中遇到的。为什么快排最坏的情况是O(n<sup>2</sup>)。我当初学的是王道那本算法, 只是说元素基本有序的时候, 每次划分都不均匀, 所以是O(n<sup>2</sup>), 我也就接收了这种说法。

但是面试同学的这个面试官, 一直不满意, 不是这么糊弄过去的。最后百度了一会, 才知道, 为什么是O(n<sup>2</sup>)是要用**主定理**来推导的。

所以自己学习的过程中要多思考。很多时候看起来很合理的解释，往往隐藏一些谬误，然后就会成为面试中的坑。

## 刷算法题

很多搞嵌入式的同学，都觉得算法不太重要，刷题的话，链表/字符串这些搞定就好了。但是如果你没什么好的项目或者说想拿更好一点的offer的话，多刷算法题还是很必要的。

我想起了4月份那会，参加海思的实习面试，面试官给我出了一道easy题，我又菜又紧张，写了近二十分钟，没写出来，最后面试官叫我秋招好好准备。那时候真的好尴尬，也没办法侥幸认为算法题不重要了。然后开启了我的leetcode刷题旅程。

一开始刷起来没什么章法，有些题目，现在会了，后面不会，看了官方解答，好像懂了，其实没懂，只是记住了而已。一开始刷的第一个多月，确实有点灰心丧气的，感觉花费在刷题上的功夫不少，还是没什么长进。

后来，我慢慢形成了自己的刷题方法，分类刷，自己写题解，用word做笔记，记录下来每一次自己的思路，写下来的代码也要记录在word文档中，并且用一个文档记录刷的每道题的日期。

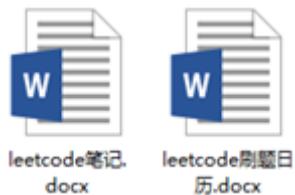
重点是题目要反复刷，在反复刷的过程中，你会发现经常有这些情况：

- 这道题以前(某月某日)一下子就想出思路，隔了n个星期这次堵了好久没想出来；
- 这道题以前代码写得好简洁啊，我现在写的什么78玩意；
- 之前写代码的时候，这一行没想太多就这么写了，但是现在发现好像又不是那么理所当然，然后又发现了自己一个思维中的弱点。

当然，惊喜时刻还是有的，比如发现这次写的代码比之前简洁好看多了，这次的解题思路比官方的比之前的都要更自然更好理解了。这样用word文档记录，就发现自己在打怪升级，将自己的螺旋式成长清晰可见得记录下来，就很踏实，很有成就感。

我刷题的题目是跟着"leetcode101"这本电子书上的题号来刷的，这本书github上能找到。它上面题目的归类很不错，但是题解真的很烂！我刷过的专题有：双指针，贪心，二分，排序，搜索，动态规划，数学，位运算，栈/队列，字符串，链表/树/图。

两个word文档，一个用来记录思路和代码，一个用来记录哪天刷了什么题目，并且可以简单记录下刷这道题的情况。



刷题：

第一，重要的是重复。

第二，是要学会死磕，准备拿一大块时间，告诉自己，我就是要拿下背包/n皇后/二叉树的各种遍历方法。

很多时候，leetcode的题解并不符合我自己的思维习惯，学习起来真的死活不懂那种痛苦啊，然后我就撒手不看，自己对着代码发呆，手动模拟，脑海想想数组/节点在跳动变换，然后就一瞬间，我明白了算法是怎么回事，代码要怎么写，跟着别人的教程自己很难懂。我就是这样搞懂了背包动态规划，空间压缩，二叉树的moris遍历等以前怎么都不明白的问题。

第三，尝试让自己喜欢上刷题吧，不然算法这玩意，带着强烈的认为到了秋招才不得不刷的情绪，很难学的，很容易急躁，有些思想，想得懂想不懂就是那一瞬间，急躁的话基本想不懂。

第四，如果实在没时间，就刷链表链表链表！反转链表，合并链表，找交点等等，总之，把力扣上链表的经典题目刷烂。第二重要就是二叉树了。嵌入式基本是链表和二叉树。

下面截几张图吧.

这是我leetcode刷题的整体记录. 从3月中旬开始的. 我刷了240道题, 但是提交了1500次, 重复了很多遍.



下面截几张word文档里面的内容.

|                        |                                                |                                                                                            |   |
|------------------------|------------------------------------------------|--------------------------------------------------------------------------------------------|---|
| 1143 最长公共子序列(中等)◦      | 2021 年 4 月 27 日◦<br>熟练◦                        | 2021 年 6 月 21 日◦<br>2021 年 8 月 1 日◦                                                        | ◦ |
| 474 一和零(中等)◦           | 2021 年 4 月 27 日◦<br>想到 0-1 背包 但是不会做 答案有点不确定◦   | 2021 年 6 月 21 日◦<br>先写三维数组版本◦<br>然后改成压缩空间版本，就可以理解官方答案了◦<br>2021 年 8 月 17 日◦<br><b>我好菜◦</b> | ◦ |
| 322 零钱兑换(中等)◦          | 2021 年 4 月 27 日◦<br>熟练◦                        | 2021 年 6 月 22 日◦<br>2021 年 8 月 1 日◦<br>2021 年 8 月 18 日◦                                    | ◦ |
| 650 只有两个按键的键盘(中等)◦     | 2021 年 4 月 28 日◦<br>不会◦                        | 2021 年 6 月 22 日◦<br>2021 年 8 月 18 日星期三◦<br>解锁新方法◦                                          | ◦ |
| 121 买卖股票的最佳时机(简单)◦     | 2021 年 4 月 29 日◦<br>会◦                         | 2021 年 6 月 22 日◦<br>弄明白了为什么是动态规划◦                                                          | ◦ |
| 188 买卖股票的最佳时机 IV(困难)◦  | 2021 年 4 月 29 日◦<br>不会 大佬教会了◦                  | 2021 年 6 月 22 日◦<br>又不会了◦                                                                  | ◦ |
| 309 买卖股票的最佳时机含冷冻期(中等)◦ | 2021 年 4 月 29 日◦<br>会◦                         | 2021 年 6 月 22 日◦<br>会◦<br>2021 年 8 月 18 日◦<br>其实不是很熟◦                                      | ◦ |
| 213 打家劫舍 III◦          | 2021 年 4 月 30 日◦<br>会做 但是官方思路更好◦               | 2021 年 6 月 23 日 会◦<br>2021 年 8 月 1 日◦<br>2021 年 8 月 18 日◦                                  | ◦ |
| 53 最大子序和(简单)◦          | 2021 年 4 月 30 日◦                               | 2021 年 6 月 23 日◦                                                                           | ◦ |
| <hr/>                  |                                                |                                                                                            |   |
|                        | 会做◦                                            | 不会◦<br><b>2021 年 8 月 16 日星期一◦</b><br>想到一种前缀和的方法◦                                           | ◦ |
| 343 整数拆分(中等)◦          | 2021 年 4 月 30 日◦<br>会做◦                        | 2021 年 6 月 23 日◦<br>会 (还有数学方法)◦                                                            | ◦ |
| 583 两个字符串的删除操作(中等)◦    | 2021 年 4 月 30 日◦<br>会做◦                        | 2021 年 6 月 23 日◦<br>2021 年 8 月 18 日◦                                                       | ◦ |
| 646 最长数对链(中等)◦         | 2021 年 4 月 30 日◦<br>会做◦                        | 2021 年 6 月 23 日◦<br>2021 年 8 月 18 日◦                                                       | ◦ |
| 376 摆动序列(中等)◦          | 2021 年 4 月 30 日◦<br>只想到 $O(n^2)$ 官方 $O(n)$ 值得学 | 陷入死循环了 ◦<br>2021 年 6 月 23 日 ◦<br>想到比官方更好理解的                                                | ◦ |

**•542 01 矩阵**

给定一个由 0 和 1 组成的矩阵，找出每个元素到最近的 0 的距离。  
 两个相邻元素间的距离为 1。  
 一个元素  
 (1) 如果它是 0，则最近距离为 0。  
 (2) 如果它是 1，则最近距离为  $\min(\text{上}, \text{下}, \text{左}, \text{右}) + 1$ 。  
 如果用  $dp[i][j]$  如何保证  $(i, j)$  的上下左右都是被计算过的？  
 如果只用一次循环，是无论如何都做不到某个坐标  $(i, j)$  它的上下左右都被计算过。  
 要找出一个位置从四个方向到达 0 的最小距离，可以为四个区域来做。  
 从上方和左方达到的最小距离。  
 从左方和下方达到的最小距离。  
 从下方和右方达到的最小距离。  
 从右方和上方达到的最小距离。  
 取四个距离的最小者，就能得到答案。相当于第 64 题做 4 次。

```
+  
class Solution {  
public:  
    vector<vector<int>> updateMatrix(vector<vector<int>>& mat) {  
        int m = mat.size();  
        int n = mat[0].size();  
        vector<vector<int>> dp(mat.size(), vector<int>(n, INT_MAX - 4));  
        for(int i = 0; i < m; i++) {  
            for(int j = 0; j < n; j++) {  
                if(mat[i][j] == 0) {  
                    dp[i][j] = 0;  
                }  
            }  
        }  
  
        // 下右  
        for(int i = 0; i < m; i++) {  
            for(int j = 0; j < n; j++) {  
                // dp[i][j] = min(dp[i][j], dp[i - 1][j] + 1, dp[i][j - 1] + 1);  
                if(i != 0)  
                    dp[i][j] = min(dp[i][j], dp[i - 1][j] + 1);  
                if(j != 0)  
                    dp[i][j] = min(dp[i][j], dp[i][j - 1] + 1);  
            }  
        }  
    }  
};
```

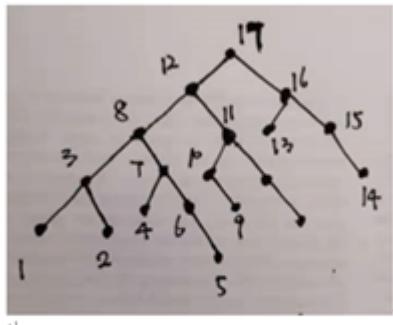
(将自己犯了每一次粗心大意记录下来，确保以后不会再犯)

```

TreeNode *pre = NULL;
TreeNode *head = root;
while(head) {
    if(head->left) {
        pre = head->left;
        while(pre->right && pre->right != head) {
            pre = pre->right;
        }
        if(!pre->right) {
            pre->right = head;
            head = head->left;
        } else {
            head = pre->right;
            pre->right = NULL;
            pushAndReverse(rsbt, head->left);
            head = head->right;
        }
    } else {
        head = head->right;
    }
}
pushAndReverse(rsbt, root);
return rsbt;
}

```

### ·后序遍历的另一种视角



对上面这颗二叉树后续遍历的结果就是{1, 2, ..., 16, 17}这个序列。

所以下续遍历就是沿着这个方向，从左往右串联每个节点。

明白这一点对于写清楚 Morris 的后续遍历非常重要。

Morris 的后续遍历中，有一个反转操作，其实质是先遍历，再反过来就是遍历了。

(自己学习Morris遍历树, 看官方题解死活不懂, 自己死磕了好久, 才发现可以这样看待后序遍历, 瞬间清晰Morris的后序遍历该如何写)

如果你的目标是sp或以上, 最好要多刷各种题。可以预料的是, 今年嵌入式薪资涨了一波, 明年会卷起来, 更卷的情况下, 只能多考核算法题了。

## 秋招实况记录

### vivo 2021年06月13日一面(挂)

和面试官聊的不开心。

先是自我介绍, 这是我秋招第一场面试, 整个自我介绍磕磕碰碰, 有一会紧张得有点说不上话来。

介绍完以后, 面试官挑了我简历上写得msp430项目来问, 当时就感觉不太对劲, msp430的项目只是我用来凑数的, 项目功能也很简单, 很多细节也忘记了。

后面他问我为什么单片机里面会有ADC呢? 我反复问了几遍为什么没有? 他也反复问了几遍为什么有。最后拗不过我, 就接受了单片机有ADC这个设定。我也是晕了, 难道是故意给我压力的吗? 还是他真的不懂? 现在市面上随便哪个单片机不内置一个ADC模块进去? 反正这么来回, 我就感受到我的面试要挂了. 最后他然我写一个shell程序, 我说我不不会, 面试结束。

这场面试完以后, 确定无疑的是我遇到了一个sb面试官。后面确实挂了。

反思一下自己, 简历做的不够好, 对以前的项目不够熟悉, msp430的项目后面我就不再放了, 主推我的FPGA项目, 然后对两个FGPA项目也复盘了好久.

自我介绍这一块, 在修改好简历以后, 经常会在脑海里排练, 后面试的自我介绍没有崩过了。

### 乐鑫 2021年06月17日11:00一面(70min 挂)

1. 自我介绍
2. 聊项目, 两个FPGA的项目
3. 问tcp udp区别(昨晚才看)
4. 问如何tcp保证可靠性
5. 问如何tcp超时重传时间如何设置(越问越深, 不会)
6. 单核操作系统如何实现宏观并行?
7. 操作系统的时基设置成多大比较合适?
8. 手撕代码:设计一个软件定时器(主要考核单链表操作 写了有半个多小时)

写这个题目的时候, 一开始没理解题目的意思, 折腾了好久, 估计是挂载这里的

整个面试过程70分钟, 累!

自我感觉良好 然后挂。

### 大华 2021年07月19日9:00电话面试一面(15min)

1. 自我介绍
2. C语言的编译过程是怎样的?

| 嵌入式面试最热门问题之一, 就是预处理, 编译, 汇编, 链接, 加载这些, 自己对中间过程要熟悉.

3. 全局变量可以和局部变量同名吗?

| 答: 可以. 程序会优先使用局部变量.

4. while和do while区别.

5. 算法的时间复杂度是怎么得出来的?

| (答不上来, 后来想了一下主要是说怎样去掉中间项的过程?)

6. 写中断程序的注意事项?

| 这个也是高频问题, 一般的我的回答是快进快出, 最好不要嵌套, 不能休眠.

7. 字符设备和块设备的区别?

"LINUX设备驱动程序(第三版)"这本书的第14页有介绍。字符设备能够像字节流一样被访问, 大多数字符设备只能顺序访问。块设备进程IO操作时, 每次只能传输一个或多个完整的块.

8. verilog有多少关键字?

(主要是我说有FPGA经验, 他才问, 答曰几十个吧)

8. 是否有奖学金和项目比赛经验?

中间扯了一下项目. 问主要负责什么.

9. 说一下个人优势和劣势(感觉说得不太好, 之前没仔细想过)

### 大华 2021年07月23日电话面试二面(10min)

1. 自我介绍.

2. Linux驱动学了哪些知识?

我按着韦东山教程的知识结构给回答了过去.

3. 如何实现一个双向链表?

答: 一个节点, 定义一个prev指针, 一个next指针, 初始化为NULL.

4. 常用哪些数据结构?

5. 讲一下Makefile的规则.

6. 如何判断一个网络数据的字节序?

7. 两个线程都执行a++100次, a的值是多少?

我义无反顾地回答200!(真是错得离谱). 答案其实是1~200之间都可能.

8. Linux如何修改IP地址.

答: ifconfig

9. 会不会设计模式?

答: 不会.

### 大华 2021年07月27日电话面试(hr面10min)

聊天.

问个人的兴趣爱好, 生活习惯, 未来规划, 是否愿意去杭州发展等.

### 联发科技 2021年07月16日一面10:45(60min)

一直在聊项目, 没有问技术问题, 后面聊部门主要工作, 联发科的发展状况, 薪资意愿.

对自己的项目要很熟悉才行, 我直接给他上手画框架图讲解协议包, 对面也比较满意.

### 联发科技 2021年07月19日16:45二面 35min

综合面 技术问题, 团队合作, 人际沟通, 加班问题都有涉及.

### 联发科技 2021年8月22日18:00谈薪

总包29w, 签字费9.6w, sp.

你永远可以相信发哥不加班.

拒了.

### 汇顶 2021年07月17日9:45一面(50min)

主要聊项目 后面聊了很多低功耗话题 问GPIO的工作模式 NVIC等等

低功耗这一块, 结合我后面的面试来看, 有不少做嵌入式产品的确很在乎功耗, 所以如果自己在简历里面写了低功耗追求, 最好去了解以下功耗有多低, 具体如何实现的.

这次面试我也是运气好, 前一段时间, 师弟有一个项目是STM32低功耗采集数据的, 然后我特意看了一下野火STM32电源管理这一部分, 看得不算仔细, 没想到全用上了.

GPIO的工作模式, NVIC等也是看野火的PDF的, 野火的资料太好用了.

### 汇顶 2021年07月20日15:00二面(15min)

综合面, 如何保证产品交付的可靠性, 还有对汇顶的问题提问.

后面拿到了汇顶的SP, 给到了40w总包, 第一次知道嵌入式可以这么顶的.

能提前批还是提前批啊, 后面很多参加汇顶正式批的, 听说难度增加了不少.

### 汇顶谈薪 2021年08月12日16:40

月薪24k, 总包40w多一点.

hr跟我说月薪24k的时候我是没什么概念(第一次谈薪), 然后跟我说了好长一串, 来了一句总包大概40吧, 我先是一愣, 反复问了好几次, 你说多少? 然后才反应过来, 我去, 40w!

我没想过嵌入式总包可以去到35w以上.

先签了. (毁约金24k, 后面果然毁了).

## 星辰科技 2021年07月13日一面 电话面试(36min)

1. 自我介绍.
2. 讲一下Linux对中断的处理.
3. 讲一下自旋锁与互斥锁.
4. 自旋锁和互斥锁哪个可以在中断里使用? 为什么?

答: 自旋锁可以在中断里面使用, 互斥锁会引起阻塞.

5. 讲一下读写锁. 读的时候可不可以写, 写的时候可不可以读?

答: 对读写锁不是很熟悉, 面试官并不是很满意我的回答.

6. 熟悉内存吗? 讲一下物理内存和虚拟内存.

7. kmalloc()和vmalloc()的区别.

8. 讲一下用户调用malloc函数是怎样一步步执行到底层的内存分配函数的?

答: 不清楚.

9. 系统调用有哪些? 执行过程是怎样的?

常用的有open, read, write. 对应驱动层有open, read, write函数, 执行到低下会触发一个软件中断, 从而使系统进入内核态, 完成后续读写工作.

10. 内核空间的数据如何传到用户空间?

答: copy\_to\_user(), mmap().

11. 一块物理内存可以被映射两次吗? 具体来说可以被用户空间映射以后再一次被内核空间映射吗?

瞎扯了一会表示这题也不会.

12. 平时是怎么学习Linux的? 有没有买开发板.

我: 我用的是IMX6ULL这款开发板, 因为最近比较流行.

他: IMX6ULL不是挺旧的一款芯片了吗?

我: 主要是大家都推荐用这款芯片的板子来学, 资料也比较多, 我就选了这一款.

13. 有没有使用git? 怎样在github上克隆一个仓库.

14. 聊了一下团队该如何分功合作. 如果团队有一个人好吃懒做怎么办?

首先我会尝试和这个人沟通, 如果沟通无效, 就团结那些愿意干活的人.

15. 问我的职业规划.

(这点自己平时其实没有多想, 只有模糊的大概的规划, 后面要补上.)

16. 你是如何看待芯片公司与产品公司的?

17. 有了解我们公司吗?(联发科, 汇顶, 大华也问了, 以后面试前还是要多做好预习)

我说主要是做音视频芯片的. 然后他说了一堆它们公司的规模, 业绩和业务(大概就是公司处于增长期, 有联发科的投资, tp-link, 大华, 华为都是他们的重要客户). 期间有说到深圳今年大概只招十来个人, 我心里想怎么这么少.

18. 问我还有什么问题? 我问深圳那边的主要业务是什么?

主要做Linux内核的驱动.

结束对话前他说了一小段话. 首先我笔试做得不错, 在对话过程中感觉我的知识大部分都只停留在书本层面, 感觉实践比较少, 技术还需要多打磨. 我说是的, 平时实验室事情比较多, 也在努力学习, 后面要多加学习与实践.

他继续说深圳那边有比较多我本科学校毕业的同学, 然后又是平常工作怎balabala.

整个面试过程下来, 好些技术问题答得不好, 但从后续的对话来看, 面试官应该会放我过, 不然和我介绍他们公司的平常工作干嘛? 个人感觉这家公司处于初创期, 也确实是在走上坡路, 公司规模不算大, 但在扩大. 在offershow上查了一下, 不高不低, 也还可以吧. 福利方面听说是双休+12天年假, 一年两次调薪机会, 年底双薪(这个感觉就一般), 有体检之类的.

后来才知道双薪不是1个月年终奖的意思.

而是12月份发两个月的工资, 年终奖另计。

### 星辰科技 2021年07月17日20min HR沟通

13薪 965双休 12天带薪年假

虽是965, 但是五天有三四天都要晚上8点多下班.

一年两次绩效考核, 一次晋升机会.

新人进去有一个半月的培训, 大概一年会接触实际的项目.

公司经常组织员工学习。

她说如果我有80%的意愿去星辰, 就给我发意向书.

我说, 我愿意。

## 星辰谈薪2021年9月27日16:50

总包34w, 公积金5.6%, 加班有点多。

拒了。

## 联影微 2021年08月9日15:00一面(60min)

自我介绍, 然后技术问题轰炸

1. 如何使用define返回2100年的秒数

```
#define second_2100 31536000UL
```

2. static关键字的用法

修饰函数定义

修饰函数内部变量

修饰全局变量

3. inline的用法? inline里面可以使用switch吗?

直接回答不会

4. 如何定义一个数组指针, 指针指向整型指针?

答: int \*\*p[5];

5. 如何定义一个指向函数的指针, 该函数有一个整型参数, 返回值类型为整型. 如何定义长度为5的这样一个数组?

int (\*p)(int);

int (\*p[5])(int);

6. const int a;和int const a;有何区别?

答: 没有区别

7. const int \* a; int \* const a; 有什么不同?

8. volatile用法?

9. volatile 和 const可以同时用啦修饰一个变量吗?

答: 可以. 详细解释网上一堆.

10. 如何对整型数a的第10bit置1和清零?

11. 如何不使用第三个变量交换两个变量的值

答: a ^= b; b ^= a; a ^= b; 还没完, 我接着说, 我们经常要在排序算法里用到交换, 但是用这种方法直接交换两个元素会有bug. 比如要交换A[i]和A[j], 如果i==j, 那么最终会是的A[i]变为0.

12. 如何让绝对地址0x2001存储整型数3?

答: `*((int *)0x2001) = 3;`

扩展一下, 如何跳转到绝对地址0x00010000去执行?

```
*((void (*)(void))0x00010000)();
```

也可以

```
typedef void(*)(void) voidPointer;*
```

```
((voidPointer)0x00010000)();
```

13. `int a = 100; unsigned int b = -99;` 请问`a + b`的值是多少?

答: 是一个很大的无符号数.

14. `sizeof`和`strlen`的区别?

答: 一个是编译时行为, 一个运行时行为. `sizeof`会计算'\0', 而`strlen`不会.

15. `0x12345678`在小端模式存储方式是?

答: 78 56 34 12

16. `float a = 2, b = 3, c = 4;`请问`(a + b) * c / 2`的值会是一个怎样的数?

答: 不会.

17. 请问`1 / 2 * (a + b) * c`的值会是一个怎样的数?

答: 不会.

18. 给定一个整型变量a, 如何判断它的无符号还是有符号?

答: 条件一 `a>=0`, 条件二 `(~a)>=0`

19. 一个结构体有`short char float int`, 怎样安排占用内存最少, 是多少? 怎样安排占用内存最多? 是多少?

20. C语言的编译过程?

21. C语言的内存布局(文本段 数据段 BSS段)

22. `char s[] = {"abcd"}; char *p = "abcd";` 有何区别?

23. 内存碎片是什么? 如何产生的?

24. 中断和异常的区别?

25. 有没有用过SPI IIC USB这些协议?

26. Linux启动过程大概是怎么样的?

28. 什么样的CPU才支持Linux系统? stm32为什么不支持?

答: 有MMU才行. MMU的作用是虚拟空间和权限保护. Linux是有线程和进程的概念的, MMU为每个进程提供了虚拟的4G内存空间, 使得每个进程互相隔离. stm32没有MMU部件, 只能运行RTOS, 里面没有进程和线程的概念, 统一称为任务, 任务之间是共享地址空间的, 如果一个任务奔溃了, 会使得别的任务跟着奔溃。

今年有一本新出版的书, 叫做"嵌入式程序员的自我修养", 我是从上面得到的知识.

29. 在linux中, 如何使用命令获取文本文件的最后20行?

我回答用sed命令, 其实是用tail命令

tail -n 20 filename

30. 优先级反转的危害? 有什么应对措施?

野火rt-thread那本书关于优先级反转介绍得很清楚. 嵌入式面试涉及到RTOS也喜欢问这个问题.

31. 有没有使用过低功耗协议? 没有

后面面试官让我等一会, 找经理过来和我聊了一会, 经理问我还有什么问题. 我问了他们组主要工作, 工作地点有无深圳, 企业文化等? 答无深圳岗. 感觉要凉.

这个公司是C语言狂魔吗, 问那么多C语言问题, 一点项目都没有问.

后面约我二面, 考虑到是上海的, 拒了.

**紫光展锐 2021年08月6日10:00一面(30min)**

自我介绍

然后聊FPGA项目, 刚开始沟通出现了误解, 看到面试官老是露出疑惑的表情, 感觉他不是很满意. 聊到后面才发现一开始我对某个概念的理解和他的理解出现偏差, 导致跨服聊天. 后面纠正过来, 又把项目的思路表述了一遍, 应该还是比较满意. 后面抽了三个简单的C语言问题, 还有ARM的中断类型. 感觉面试官有点想问什么又不知道该问什么了.

**海康威视 2021年08月7日11:00(突然来电30min)**

问的还是比较多不会的，主要是Linux相关和ARM架构相关的。不过挑一些会的来讲，应该能过。  
ARM Cortex系列处理器的分类。

Cortex-M3内部框图

MSP430与STM32的区别

Linux系统由哪几个模块组成。

Linux对中断的处理。

后面通过了约我二面，成都的岗位不想折腾，就拒了。

**紫光展锐 2021年08月9日16:50二面(16min)**

首先自我介绍。

和面试官聊项目，互动了几个问题感觉他不是很感兴趣。

问我有没有做过竞赛，我说本科作过。

问我对职业发展的规划。

问我还有没有什么要问的。我尬吹了一波紫光。

整个过程不是很顺利，面试官表现得无聊又疲倦，打了几个哈欠。

估计会挂。

确实挂了，不过展锐真的给得好低。

**诺瓦科技 2021年08月12日10:40(嵌入式一面50min)**

聊项目比较多，问了一些简单的C语言问题。

**诺瓦科技 2021年08月12日15:00(FPGA一面30min)**

这个应该会挂，FPGA不是我专长

**诺瓦科技 2021年08月19日09:50二面(嵌入式方向)**

聊天，问一些项目相关的，以及个人情况，薪资意愿等等。

聊得还挺开心。

好久没消息，2021年8月27日查了一下二面挂了

**CVTE 2021年08月12日16:00(一面15min)**

面试官就没有提前准备，面试的时候才临时看我的简历。

问了一些Linux驱动相关的问题，没有答好。

最后问题爬楼梯(力扣原题)的算法思路。

结束。

### CVTE 2021年08月18日21:00 (HR面25min)

聊天, 关于性格, 职业方向, 期望薪资, 连最喜欢的小说人物都要问.

### 阿里云 2021年08月17日20:00电话面试(30min)

突然约我电话面试.

没有自我介绍, 上来就问我的项目.

然后针对项目聊了有二十分钟. 感觉他对项目要解决的问题以及应用场景不是很认可. 还会涉及我在项目中的主要工作. 后面想问我TCP的知识, 我简略答了几句, 直言我对TCP研究不多. 又瞎扯了一会. 他哪边感觉可能我的技术和它们研究的不是很符合, 暗示我的简历可能会被推到其他部门里面去. 算是挂了.

阿里在杭州, 没打算去, 投这个也是想体验一下互联网大厂的面试.

后面了解到阿里云的价格是30\*16, 签字费有5w有8w.

### 小米 2021年08月24日14:00(一面)

自我介绍.

聊项目, 面试官对我的项目还是比较感兴趣的.

然后问了几个简单的技术问题.

1. 我看你项目简历写到对算法有一定的了解, 你能说说除了冒泡以外还有什么排序算法吗?

答: 插入, 选择, 归并, 快排, 堆排.(到这里我以为他要我手撕一个排序了, 结果没有)

2. 讲一讲归并排序的思路.

答: 归并排序其实是一种基于分治和递归思想的算法, 先把数组的左半部分归并排序好, 然后把数组的右半部分归并排序好, 然后合并左半部分和右半部分. 算法的时间复杂度是 $O(n\log n)$ .

3. 信号量是用来干嘛的?

答: 用来同步和互斥. 如果初始值为0, 则用于同步, 如果初始值为非零, 则用于互斥.

4. 信号量和互斥量有什么区别?

答: 互斥量首先可以认为是初始值为1的信号量. 除此之外, 根据我阅读RT-Thread的源码了解到, 互斥量拥有所有权, 递归访问和防止优先级反转等特性.

注: 这部分其实是从野火的RT-Thread内核与实战指南了解到的.

5. 互斥与死锁的关系?

答: 互斥是死锁的必要条件之一.

在面试前一天, 从一位投小米C++岗的同学口中了解到, 小米要手撕代码, 难度是简单和中等. 然而面试官并没有让我手撕, 可能是嵌入式对算法要求不高.

### **小米 2021年08月26日15:00(二面)**

自我介绍.

他说他们部门做RF开发的, 比较偏通信, 看我的简历, 似乎对Linux比较感兴趣, 问我能不能接收入职以后做通信这一块.

然后给我介绍了他们通信技术组的三个方向: 协议方向, 驱动方向, RF方向.

我说, 没问题. 不过我坦白说我在计算机学院, 平时对通信了解不算很多, 不知道技术上能不能达到你们要求. 他表示, 你们这一届, 简历都比较好看, 动手能力都挺强, 我们看重你们的学习能力, 可以进来以后再学.

接下来问我有什么兴趣爱好.

我说平时锻炼一下身体, 周末去找吃的.

到此为止不到10分钟, 他就想要结束面试了.

我心里想, 这也太快了吧.

他好像也意识到太快了. 让我问一些问题.

我说, 你刚才说的几个方向, 不是特别理解, 可以再详细说说吗?

然后他就详细说了一遍.

整体下来感觉就是, 小米就是组装厂.

最后他让我说说做项目的过程中遇到的体会深刻的经验.

然后我就说了一个解决bug的过程.

终于把时间拉长到接近半个小时了.

结束.

### **小米 2021年9月15日19:30(谈薪)**

总包32w, sp, 公积金12%.

小米不加班还是比较可靠的.

最后还是拒了.

### **荣耀 2021年08月25日09:00~11:00(一面)**

自我介绍.

介绍项目, 面试官就项目简单问了几个问题, 然后就完了.

感觉被刷kpi了, 面试官对我的项目不是很感兴趣, 对话过程中好几次感受到他注意力不在我这里.

想吐槽一下荣耀对面试的组织, 先是提前一天发邮件告诉我第二天早上9点开始面试. 然后早上九点, 根据指引进去一个线上会议室, 发现里面是hr在主持的多人会议室, hr说面试官在看同学们的简历, 等会会叫号. 我等了一个小时才轮到我面试. 体验不太好, 也可能是荣耀第一年校招, 没什么经验.

### **荣耀 2021年08月28日09:30~10:10(二面)**

(没想到过了)

自我介绍.

你对职业发展的想法.

在项目中遇到的问题, 具体是怎么解决的? 要偏技术细节一点.

讲讲你对算法的理解.

反问: 你们部门是做什么的?

他: 媒体业务, 具体是音视频图像处理.

这次是综合面试, 技术问题和非技术问题交叉着问, 主题偏非技术.

整个过程还是比较顺畅.

### 大疆 2021年08月27日19:45~20:30(一面)

自我介绍.

1. 有阅读过Linux内核的源代码吗?

答: 没有专门地阅读过. 之前学习到Linux内核驱动, 遇到相关知识会深入一点看源代码.

2. 熟悉Linux的内存管理吗?

答: 了解得不是很多, 大概知道是有一个伙伴系统. 不过Linux的虚拟内存有一定了解, 是之前学习MMU模块的时候学的.

3. 能讲讲伙伴系统吗? 还是只是知道这个名字?

答: 伙伴系统就是内存块大小为 $2, 4, 8, \dots$ , 这样的 $2$ 的幂的内存, 大小一样的内存块称为伙伴. 分配的时候从小到大查找适合的内存. 释放的时候找到同级别的内存块, 合并成更大的伙伴.

4. Linux的信号量和互斥量了解吗?

答: 其实不太了解, 不过之前阅读过RT-Thread的源代码, 对这个操作系统的信号量和互斥量比较熟悉.

5. 那你说说RT-Thread的信号量和互斥量.

答: 信号量的取值可以为 $0, 1, 2, 3$ 等等, 而互斥量可以取值为 $1$ . 他们具体的区别可以从控制块定义看出来, 信号量只有信号量的值, 以及阻塞在这个信号量上的线程的链表, 而互斥量还有持有者, 优先级等域. 归纳来说, 互斥量对比信号量有3点不同: 1. 所有权; 2. 递归访问; 3. 防止优先级反转.

6. 你可以说说优先级反转吗?

答: 我以一个场景来说明可以吗? (可以). 然后说了野火RT-Thread教程上关于优先级反转的一个例子.

7. 怎么解决优先级反转?

答: 可以通过优先级继承的方法. 然后又通过场景来解释.

8. 还有其它方法吗?

答: 我没有了解过其它了. 不过应该可以通过关调度器或者关中断的方式解决.

他: 你可以了解一下优先级天花板.

9. 我看你的简历有说做了一个MIPS的CPU, 可以说说具体是怎么做的吗?

答: 找了一本CPU相关的书, 然后跟着敲代码, 不仅仅是仿真运行, 还下到板子真正跑起来了.

10. MIPS和ARM以FPGA有什么关联或者区别?

答: MIPS和ARM都是精简指令集的, 都使用流水线技术. 不过据我了解, MIPS使用的是五级流水线, ARM的话比如说STM32, 用的是3级流水线. MIPS和FPGA不是一个平级的概念, 具体来说, 要研发一款芯片, 需要先在FPGA上实现软核, 验证通过后再流片量化生产. FPGA更加偏底层, 它是原始的逻辑阵列, 逻辑门之间怎么连线是由芯片设计者来决定的.

11. 问: 你有说过项目中的低功耗实现. 能说说功耗有多低吗, 和市面上的单片机相比如何?

答: 在通信速率为300k/bit下, 可去到20uW的功率, 而像低功耗做得比较好的MSP430, 至少都上上百uW.

12. 市面上的低功耗单片机有了解过吗?

答: 据我所知, 前几年的话, 德州仪器的MSP430系列低功耗是做得最好的, 而近年来, 意法半导体STM32的L系列功耗方面也做得很出色, 比MSP430系列要好, 而且STM32的生态要好很多, 以后应该是ST能拿下市场.

13. 问: 最近有看什么书籍吗?

答: 最近找工作, 看的都是和计算机相关的书籍. 有RT-Thread相关的, 还有经典书籍比如深入理解计算机系统, 看的主要是链接以及系统IO方面的内容. 最近也有看算法方面的书, 了解基本的一些算法思想.

14. 你们学计算机的, 算法不应该是基础课程吗?

答: 以前确实有选这么课, 但是学得不是很好, 最近才补回来.

15. 两个有交点的单链表, 如何找到它们的交点, 给我说一下思路?

答: (这题力扣有)我思考了一分钟左右(其实我早就会了, 战略性思考), 就按着力扣的思路给他讲了一遍, 得到认可.

16. 有玩过大疆的无人机吗?

答: 没有. 身边一个师兄, 喜欢旅游看风景, 有时会跟我说一下大疆无人机拍照多好看.

17. 你觉得大疆的无人机产品有什么缺点?

答: 没用过, 不清楚.

结束.

## 大疆 2021年09月4日10:00(二面)

自我介绍.

深挖项目.

面试过程中, 对面一直反驳我项目中的一些做法和思路, 我也反驳回去.

当时为什么敢反驳, 因为我根本没想着去大疆, 有最好, 没有就拜拜的心态.

没想到可能是我敢反敢驳, 最后给了我sp.

最后问我关于Linux的lcd驱动是怎么写的.

全程面试官不置可否, 感觉自己表现一般.

## 大疆 2021年09月13日15:30(三面)20min

首先自我介绍, 因为是非技术面, 自我介绍的时候技术方面的就略过.

1. 在做项目过程中遇到的印象深刻的困难?

答: 疫情期间在家自学FPGA, 从零开始, 要把整个项目独立做出来.

2. 最大收获?

答: 自学能力, 以及独立完成项目的工程能力?

3. 认为自己在做项目过程中的优缺点是什么

答: 我认为优点是我能够静下心来去学习全新的技术, 这点对以后工作是有帮助的, 因为在工作中可能需要经常学习新的东西. 缺点是当初在做项目的过程中和师兄沟通方面不是很充分, 导致自己完成了任务, 并不是所需要的.

4. 那你认为应该怎么进行沟通呢?

答: 我举了一个请师兄帮我调试bug, 怎样与他进行沟通的案例.

复盘一下, 觉得上面两条答得不是很好, 缺点方面我可以说, 平常都是一个人负责项目, 没有尝试过团队合作, 可能以后要团队合作的话, 会缺少一些经验. 后面问怎么沟通, 也可以以实验室日常工作举例子来回答.

5. 最后项目有什么产出吗?

答: 论文原型的实现, 不仅仅是仿真运行, 还有下板子做实验, 为论文采集实验数据.

6. 在工作学习之余还会做什么感兴趣的事情吗?

答: 之前学习过FPGA, 虽然说以后不会往FGPA方面发展, 但是因为我对底层逻辑还是比较感兴趣的, 利用学习过的FPGA知识, 在空闲时间, 去复现一款MIPS的32位CPU出来, 最后下到板子运行, 另外还用FPGA编写了一个程序存储器, 将CPU要运行的代码下载到Nor Flash上面, 这个过程需要去阅读Nor Flash的芯片手册, 并用FPGA实现它的通信时序.

7. 为什么要投大疆?

大家都说大疆是国内做嵌入式的天花板, 要做一款无人机产品出来, 要用到的不仅仅是IT领域的知识, 基本上整个工科领域都要精通才可以, 如果我能加入大疆, 对我的技术视野的提升是非常有帮助的. 我个人也比较喜欢在硬件上捣鼓软件, 写代码让硬件跑起来, 就感觉在注入灵魂, 是一件有趣的事情.

#### 8. 对大疆有什么认识?

答: 无人机在消费, 工业, 农业, 教育都有涉及, 市场占有率达到80以上.

后面可能感觉我平时不怎么玩无人机, 就问我是不是无人机接触比较少. 我说是的, 毕竟现在没什么钱, 等以后工作了, 会买几台来玩玩. 然后他说公司里面无人机随便玩.

下面就到反问环节了.

我问工作产品线是进去以后再分吗? 答: 是的.

如果有幸加入大疆, 在毕业前我还想学习进步, 你有什么建议吗?

答: 建议多看看实时操作系统.

#### 大疆 2021年9月28日(谈薪)

45w以上, 公积金8%, 双休, 提供南山区半价宿舍.

签了.

#### ARM 2021年09月13日14:00(一面)20min

我投的是CPU设计岗.

自我介绍完以后, 面试官怕我投错岗位了, 解释说他们是做CPU的模型编程的, 具体就是用C或C++模拟一个CPU的行为. 我说, 我就是偏向做软件的, 岗位描述也是看清楚了才投的.

#### 1. 你做的MSP430单片机项目是哪家公司的, 有接触硬件设计吗?

答: 德州仪器, 没有设计硬件, 但是板子是我们实验室做的, 我也要懂硬件设计才能编程.

#### 2. 你做的项目里面有CPU的实现, 可以说说流水线设计的这种思想吗?

答: 流水线思想是RISC类型的CPU专有的技术, 其思想就是将多种操作拆解成共同的操作单元, 我所做的MIPS为五级流水线: 取值, 译码, 执行, 访存, 写回, 然后这些操作单元的搭配, 就可以完成各种各样的操作. 只需要较少的硬件电路就可以实现这些操作单元, 而不像CISC类型CPU要很多复杂的硬件结构, 这样大大降低了CPU的设计难度.

#### 3. 你是怎样查看CPU的运行结果的?

答: 首先我会在仿真软件上看过波形没有问题, 然后就会烧写到FPGA板子上, 使用逻辑分析仪来看信号是否正确.

#### 4. CPU的性能有没有做过评估?

答: 当时做这个项目是出于兴趣, 主要还是想弄清楚原理, 对性能方面的事情不太关注.

5. 一个具体技术问题, 如何解决LOAD相关, 比如说load指令后面的add指令要用到刚刚读取出来的数据?

答: 我思考一会(战略性思考)...因为add指令直接跟在load后面, 所以需要暂停流水线, 然后直接将数据送到执行阶段的寄存器.

6. 我再问具体一点,暂停多久?

答: 一个周期.

7. 你写汇编代码怎么验证执行结果的正确性的?

答: 写的汇编代码, 会先手算出预期的工作结果, 然后仿真运行的时候再去对比.

8. 追问: 可是如果代码量太大了, 手算很慢怎么办?

答: 可能你想说使用一些自动化的方法, 比如说使用python来计算出测试集的结果, 不过这个项目是我出于兴趣在做的, 汇编的代码量也不是很大, 也就没有使用这些脚本去编程.

9. 你在做项目过程中遇到的挑战?

答: (答过很多遍了).

### ARM 2021年09月18日17:00(二面)

对面有三个面试官, 其中一个还是一面面过我的.

自我介绍完后, 一直深怼我的MIPS32 CPU项目, 忿内核, CPU流水线, 我确实研究得不深.

其间好次, 对面问, 我答, 沉默几十秒.

表现不是很好, 感觉没戏.

2021年9月30日更新

问过hr, 说第一批offer名单没有我.

问过参加了线下二面的同学, 说问了比较简单基础.

可能是我没有参加线下的, 故意给我提高难度.

arm开得还挺高, base年薪三十多, 然后加上绩效/股票/签字费, 去到了四十多.

### 科大讯飞 2021年09月13日15:30~16:00(一面)

自送介绍.

狂怼项目.

问: 你在做项目过程中遇到的挑战?

答: (答过很多遍了).

讯飞挂了, 估计是深圳没什么岗位需求.

### 360 2021年09月6日14:00(一面)

本来预计1个小时的面试时间, 最后只进行了不到30分钟.

对面没有开摄像头.

自我介绍, 还没介绍完就被打断了, 问我简历上的第二个项目. 我简历上的第二个项目是关于无线通信信道利用率的提升的. 他们360做的是偏向网络通信方面的, 所以打断我就问.

整个面试过程下来, 那边比较希望我有一些网络相关的实战, 或者是Linux, RTOS方面的实战, 但不巧, 我都没有, 我的项目是偏向FPGA和MCU开发的, Linux和RTOS知识都是理论层面的.

面试过程穿插着算法部分和专业技能部分.

算法部分包括判断大小端, 字符串拷贝, 反转单向链表, 树的四种遍历方法, 排序算法, 平时都做得很熟了.

专业技能部分, 他想问我有没RTOS, Linux实战项目, 我都回答没有.

又问: 对HTTP熟悉吗?

答: 不熟.

问: 对Web开发熟悉吗?

答: 不会.

问: 数据库呢?

答: 完全不会.(回答到这, 我已经理直气壮不会了)

最后问我, 你觉得你还有什么亮点?

我问: 除简历上的吗?

他说: 你简历上说对MIPS架构比较了解, 是怎么回事?

答: 我学完FPGA以后, 自己写了一个CPU, 然后大概介绍了一下.

没有反问环节, 面试结束.

整场下来感觉就是, 360毕竟还不是专门搞嵌入式的, 他们可能更想找一些项目经验对口的, 但是对面试官又想捞一下, 所以多问我一些算法方面的知识.

一面通过了, 没有参加二面, 因为刚好没空.

### 小马智行 2021年09月11日16:15(一面)

大约1小时

自我介绍, 然后狂怼项目.

技术问题:

1. Linux与RTOS的区别?

答: 1. RTOS从名字上来看, 实时性是其关键特性(有很多O(1)算法), Linux更加看重高性能会用一些复杂的数据结构和算法来提升整体的性能, 但是不保证实时性; 2. Linux有线程和进程的区别, 这是由内存管理单元支持的特性, RTOS没有内存管理单元, 执行单元叫做任务, 所有任务共享一个运行环境.

2. RT-Thread是怎样进行任务调度的?

答: 优先级+时间片轮转.

3. 如何设计任务的优先级?

答: 答得不好.

他说: 可以分为IO型任务和计算型任务去设计.

4. volatile关键字的用法?

答: 1. 防止优化. 2. 强迫CPU读取内存而不是cache.

5. 如何查看Linux的IP地址?

答: ifconfig.

6. 同一个局域网的Linux系统, 如何互传文件?

答: 我说用mount, 他说用ssh. (后面查了一下, 可以用scp工具)

编程题: 去除链表升序的重复元素, 重复的只保留一个.

2021年9月26日更新

我早早就面完小马了, 而且感觉自己表现还不错.

后来有的同学在我后面面试的, 都已经安排面试了.

然后中秋节没联系上hr, 搞得中秋节都过的不安心.

后面hr才说面试官忘记反馈我的成绩了, 今天终于说要安排我二面了.

**小马智行 2021年09月26日14:00(二面)60min**

自我介绍.

针对我的项目聊了挺多.

然后问一些Linux和RTOS相关的.

(一般聊到Linux和RTOS都会问一下他们之间的区别, 具有什么部件的芯片(MMU)可以跑Linux)

Linux方面问了设备树, 设备驱动模型.

RTOS方面问了系统通信方式.

还有一些其它问题:

栈设置的大小为多少合适?

什么情况下会栈溢出. 我回答嵌套太深, 递归太深, 中断不断发生嵌套.

还有编译过程.

编译过程其实就是那一套：预处理，编译，汇编，链接，装载运行。

如果要展开的话，会问预处理做了什么，链接完以后的文件怎样组成。可能还会问一下Makefile的写法。

编译涉及到的知识可以参考深入理解计算机系统这本书。

顺着编译过程，问可执行文件的段组成。

然后问BSS段是怎样生成，怎样设置的。

BSS这一块，我一开始答错了，我说是装载器根据可执行文件里的段信息，装载的时候，在内存中清理出一块空间作为零值空间，然后在内存中就形成了bss段空间。

然后他问我，可执行文件是bin文件，全是机器码，没有段信息，怎么知道如何形成bss段。

我说，可以自己写重定位代码，在代码里获取链接脚本的段信息，然后通过代码设置清零，这段代码最后在可执行文件里。

这部分知识可以看韦东山的免费教程

接下来是代码环节。

让我写一个基本计算器，带括号的。

我懵了，前两天问了两个同学说手撕memcpy和strncpy。

基本计算器的写法我只会中缀表达式转后缀表达式的方式，转换的规则还有点繁琐，挺久以前写的，实现起来也要有上百行代码，而且涉及到带括号，有一些细节要处理好，现在实在写不出来了。想了几分钟，我坦白说没有思路。

然后面试官给换了一道简单题，完美树(力扣有)。

用O(根号n)的方法做出来了。

完了以后给面试官阐述了以下第一道题的基本计算器的思路。

然后面试结束。

### **小马智行 2021年10月12日20:00(三面)**

小马是我留在广州唯一的希望。

之前听杭电的同学说，小马三面竟然还是问项目，整得我有点慌。

等我自我介绍完以后，他问我说，你对底层比较感兴趣，了解也比较多，简历上写了做过一个流水线CPU，那么你能说说流水线太长有什么坏处吗？

我没回答上来，心里想流水想过长，会导致跳转/中断等效率低下，但是组织不出系统性的语言。

这全场唯一的一个技术问题没回答上来，有点小慌。

简历上做过的项目，有些细节难免会忘记或者没有注意到，在面试之前最好尽量把自己的盲点找出来。

后面就是聊一些非技术性的话题了：

问：在做项目中遇到印象深刻困难，如何克服？

问：在团队协作中，人际交往方面的感悟。

问：为什么要选择小马智行？有没有了解过其他同类型的公司。

问：你在完成一件事的过程中，在什么情况下会求助别人？

问：我说在之前做第一个FPGA项目的过程中，遇到了一个很难的bug，自己调了好久，看书找理论，参考别人的代码，将自己的代码推倒重来，花费了好多时间，最后还是没有调出来。后面我求助了一位师兄，他很热心帮我调了一天，问题才得以解决。但是我后来反省自己，虽说这种自己与困难斗争到底的精神值得鼓励，但是在尝试过以后，要尽快找他人求助，不然会耽误项目的交付时间。后面我也学会遇到困难先自己调一下，但不会死磕，实在搞不定就要尽快求助他人。

后面的一些问题不太记得了，主要还是一些综合问题，对面好像在看一个清单，上面是综合面试时应该提的问题。（因为我每次回答完他就盯一下屏幕，感觉是在挑一个合适的来问）

到了反问环节，我了解到广州南沙他们团队现有做嵌入式方向的有5个人，今年秋招完以后会扩展到30个左右。平时工作用到的技术包括Linux/RTOS/ARM等。

后面hr也告诉我面试通过了，不过要十月底才开奖。

### 华为 2021年09月13日15:00(一面)

面试官迟到了15分钟，估计上一个延误了。

自我介绍。然后问简历上的第一个项目，问得不是很深入。觉得面试官面无表情，语气平淡，搞得我不自信了。

技术问题有：

1. Linux和RTOS有什么区别？
2. RTOS的优先级反转以及解决方法？
3. Linux系统挑一个熟悉的讲讲。（我就挑了中断系统，这一块回答得不是很好，凭着学习韦东山视频的记忆去讲的）。
4. 软中断是什么？我就顺着讲了tasklet和工作队列，线程化中断。
5. 工作队列的执行环境是用户空间还是内核空间。（这里我竟然回答了用户空间，脑袋抽了）。
6. 然后又问了一些自旋锁和一般的互斥锁的问题。
7. 如何解除死锁？就回答了教科书八股文。
8. 用过什么算法？
9. 贪心算法和动态规划算法比较一下？

手撕代码：

1. 使用数组实现一个队列。

要实现入队，出队，判空，判满操作。

判空，判满这里我用( $r == w$ )和 $((w + 1) \% \text{SIZE} == R)$ 来实现。他说会浪费一个存储空间，怎样不浪费，我想了一会，说加一个变量记录丢弃的长度吧。他说也行吧，但是还有其它方法，让我后面再想想。

写完后问我多个线程怎样防止访问队列出错，我回答说用锁，或者按照线程ID有序访问。

2. 反转一个单向链表。

反转链表真的是考过好多次了。

我一下子就写出一个用假头节点的方法，他不满意，让我别使用假头节点。

然后我又改成了没有假头节点的，并且在程序入口加了特判（如果链表为空或者只有一个节点就直接返回），结果他不满意，让我把特判删除了，只能在while有判断，我实在想不出来怎么优化了，他就说算了，面试就结束了。

上次小马面试，我不加特判，面试官不高兴，这次我加了特判，面试官不高兴，无语了。

面试完以后，觉得自己15级无望了。当时在市区的一家自习室面试，准备收拾包裹走人了，刚走到电梯，就收到华为马上要二面的通知。我也是服了华为了。

### 华为 2021年09月13日(紧接着二面)

这次就不用自我介绍了. 一开局, 对面对跟我说, 一面面试官对我评价很不错. 果然是压力面.

然后面试官主要挖项目, 挖得挺深.

后面是算法题, 要设计一个类, 对字符串出现的次数进行统计, 添加字符串, 删除字符串等等操作.

我心想不是一个map就搞定了吗? 想起一面, 简单题各种优化, 我心里还是有小慌. 不管那么多, 先暴力解决一下, 然后我说我想想怎么优化吧, 那边直接说, 你把代码复制一下, 提交过来吧. (不会就这么简单吧?)

结尾问我有什么优缺点.

后面很快就收到通过短信了, 等主管面.

### 华为 2021年09月17日15:00(主管面)

主管面还是聊些综合兴问题. 聊的还挺好.

坑的是, 我反问环节才知道, 我的简历被推去深圳了.

之前有一个东莞海思的短距通信部门约我面试, 我就一直以为是东莞这个部门, 还好我多问了一句.

后面折腾了好久, 终于联系到hr说把我的简历推回去短距部门了, 不需要重新面试, 等结果就好了.

十月下旬了, 身边同学都有华为意向书了, 我估计是无了, 估计是东莞海思没有hc了.

### 美团 2021年09月19日10:45(一面)

不太在意这场面试, 问的什么忘记了.

一面通过了.

### 美团 2021年09月25日16:00 (二面)

自我介绍, 然后全程聊项目. 20min.

挂了.

### vivo 2021年09月28日14:30~15:00(一面)

第二次面vivo了, 招工作第一次就是vivo, 然后凉得很快.

这次整个过程还比较好.

自我介绍完以后, 问了一下项目, 然后是Linux内核方面的知识.

有些地方回答得不是很好.

vivo这次能不能过只能说一半一半.

vivo挂了.

### OPPO 2021年09月27日16:00(一面)

面试官迟到, 约了4点, 过了五分钟还没人, 我想等到10分就不等了, 结果那边刚好9分打过来.

信号还不好, 折腾了好久, 最后换成电话面试.

自我介绍, 然后问项目.

在项目上问力几个问题, 突然就说面试结束了.

估计对我的项目不是很感兴趣.

### OPPO 2021年09月29日19:00(二面)

我以为一面凉了.

这次没有自我介绍.

面试官现场看了我的简历, 然后顺着简历问了我的项目.

主要是项目的原理, 目的, 遇到的问题, 如何解决.

最后还聊了一下天, 问兴趣爱好, 我说平常在成都找吃的, 他让我推荐馆子.

我叫他去玉林路.

整过过程比较轻松.

面试官留着络腮胡子, 有点像张学友.

### OPPO 2021年10月11日15:20~16:00(三面)

hr面.

整体感觉不是很顺畅.

在交流的过程中好几次不知道该如何措辞, 讲完这句不知道下句讲啥.

感觉对面hr很会控场.

问的几个问题我一下子不知道该如何措辞回答(或者说她问的比较空?)

比如说这些问题:

你觉得自己的表达能力如何?

你做完项目以后, 有没有归纳总结?

你的项目能够代表你的水平吗?

后面打听我投了哪些公司, 拿到了哪些offer, 有没有签约, 三方下来没有.

后来了解到oppo的驱动开发岗只有深圳, 为了不让面试尬住, 我说深圳也可以去.

2021年10月13日更新

拿到了offer

22 \* 15 公积金5% 深圳房补1200/月

拒了

## 总结

6月份开始秋招,一开始对嵌入式的薪资没什么期待(35w是想象力的天花板),我读研更多是出于兴趣, 嵌入式属于计算机这一块, 虽然没有互联网这么好, 但是肯定饿不死的。 我是在决定考研计算机几个月以后, 才慢慢发现计算机原来这么火爆, 薪资这么高。

今年秋招, 嵌入式的薪资水平我个人感觉已经和互联网不相上下, 甚至互联网还有点比不上的感觉。 汇顶第一个给我开奖, 从来没想过年薪可以去到40。

后来, 在嵌入式秋招群里, 不乏一堆兄弟年包40以上的, 还有一个双非本拿到接近50的大疆总包。 然后慢慢接受了嵌入式薪资跟着IC水涨船高这个事实, 最后大疆的整体薪资也压倒了一片头部互联网公司。

我觉得今年在嵌入式这块找工作是幸运的, 互联网依然内卷, 笔试刷题, 好几轮技术面, 每轮都要手撕算法, 还要项目好。 而嵌入式这边, 算法是加分项, 很多次面试, 面试官基本只问我项目和专业知识, 让手撕代码的屈指可数。

最后互联网开出来的薪资其实比起嵌入式高不了太多了, 甚至在头部公司嵌入式要高一点。 但是还是劝嵌入式的同学, 多刷题, 鬼知道嵌入式会不会卷起来。

最后推荐一下学习资料.

1. 单片机和RTOS, 野火的指南者和rt-thread的电子书我是看得最多的.
2. 嵌入式Linux, 韦东山的免费教程(B站, 百问网), 野火的也很不错(最近在补).
3. 开发板用的是韦老师的.
4. 算法方面, 胡凡的算法笔记, 力扣, 还有leetcode101的题目列表.
5. 秋招面经这块, 感谢@仲一群主提供的"嵌入式软件工程师笔试面试指南", 一开始秋招真的很焦虑, 觉得随便一个专业问题都回答不上来, 最后是看了这本面经心态才稳定下来. [嵌入式与Linux那些事]3群yyds!!!
6. FPGA的话, 也是野火的教程比较好(嵌入式软件FPGA不需要, 有是加分项).
7. 像什么CSAPP, 程序员的自我修养这些, 是好书, 但是就不多推荐了, 秋招过程中可以针对某些问题翻一翻, 不适合一大块时间拿来啃, 对秋招的直接帮助其实不算太大。 如果真的想学, 先按前几点学习, 秋招完以后抽时间好好看这些计算机的经典书籍, 对职业发展肯定有帮助. 我列举我看过的几本书籍: 深入理解计算机系统, UNIX环境高级编程, LINUX设备驱动程序, C和指针, 算法笔记, 嵌入式程序员的自我修养, RT-Thread内核实现与应用开发实战指南-基于STM32, STM32库开发实战指南-基于STM32F103。

## 双非本科进大疆

哈喽, 大家好, 我是仲一。 今天和大家分享的是一位优秀双非本科生上岸大疆的经历 (羡慕哭了。。。)。

今年4月底的时候, 这位学弟和我分享了他拿下oppo, 京东, 联发科实习offer的经历, 当时我还发了朋友圈, 为这位学弟感到开心。 这也是第一位粉丝向我报喜, 我映像很深刻。

晚上9:55

16.4K/s ⌂ ⌂ HD HD WiFi 48%

2021年4月25日 11:50



...



...

上午10:53



大佬您好，我是 [REDACTED] 想加入资料群 ~

以上是打招呼的内容

你已添加了 [REDACTED] 现在可以开始聊天了。



前辈您好呀，我是知乎公众号  
CSDN 老粉丝了 😊，今年大三找实  
习也是参考了您的面试经历 知道  
了面试的大致套路 最后签了  
oppo 的实习



感谢!!

上午11:03

恭喜，恭喜！



我可以截图发朋友圈吗，名字打码



可以的！

我还有京东的 offer 联发科刚面完  
估计也快了

恭喜，恭喜！这样的喜讯请多来一点！哈哈

❤ 赞

评论

❤ 34

评论 12

公众号：嵌入式与Linux那些事

晚上9:55

2.7K/s ⌂ ⌂ HD HD WiFi 48%

2021年4月25日 11:50



...



...

看来总结的内容，对大家还是有用的。你是第一个向我反馈的 😊



有的呀 我发现其实嵌入式能问的问题就是那几类

你总觉得很好了 尤其是 c/c++ 那篇

确实经常被问到

还有笔试题也是必考这些内容 😊

是的。不刻意去注意，问到了可能都不知道。我当初就是在网络上搜集各种笔试面试题目，拿来看。



面了这么多场下来，基本都在里面了



对！系统看面经复习 提高的成功率不是一星半点

是的，技术好，在校做项目厉害。  
并不代表，面试一定能表现好。



恭喜，恭喜！这样的喜讯请多来一点！哈哈



♥ 赞

评论

♥ 34 □ 12

公众号：嵌入式与Linux那些事

晚上9:55

2.7K/s ⌂ ⌂ HD HD WiFi 48%

2021年4月25日 11:50



...



...

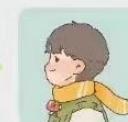
笔试面试的内容，针对性的准备，  
可以事半功倍。



实习，好好表现



建议，写文档。



嗯嗯 我现在有意识的培养这习惯了

把遇到的问题，如何解决的，都记录下来。秋招的时候，讲给其他面试官，也能回忆起来。



学的知识点都会总结

还是得有个引路人哈哈 当初牛客上看到了你的面经真的好 我就对着你面经的题目大致的开始准备答案

嗯嗯。关于资料，有任何建议和补



充的内容，都可以向我反馈。我尽量都照顾到各个方向的同学。

没问题！

恭喜，恭喜！这样的喜讯请多来一点！哈哈

♥ 赞

评论

♥ 34 □ 12

公众号：嵌入式与Linux那些事

找工作其实也没你想的那么难的，找不到合适的工作，可能是方法不对。

其实，很多大厂对于应届生的要求不会太高，最重要的，还是考察基础和综合素质。进入公司后，公司会有完善的培养体系，会把应届生当做白纸来培养。

面试也不要太难。在面试的时候，问题答上来了，面试就可以通过。而这些问题，是我们可以把控的，可以提前准备。

笔试面试的常考知识点我都整理好了，大家可以参考这个专栏。

[笔试面试文章汇总](#)

不多说了，下面就分享下这位学弟的秋招经历。

## 个人背景

大家好，我的秋招已经告一段落了，写下一些总结，记录下这三个月的面试经历，希望能帮助到后面找工作的伙伴们。

**背景：**双非一本 本科。

**大三暑期实习：**拿到了OPPO、MTK、京东等offer，最后去了本分厂实习了三个多月，在其中有输出，也有成长。

**秋招提前批、正式批：**因为只想在广深工作，所以秋招只投递公司共20家，目前拿到了其中10家公司的offer，还有2家等结果。

**被挂简历的企业：**（华为、英特尔、tplink、autox）。

**投递后没有消息的：**（小马、tcl、寒武纪、vivo）。终面后在等待结果的有：（美团、紫光展锐）。

**获得公司offer的有：**大疆（sp 签约~）、Arm china、小米、荣耀、联发科、全志科技、星宸科技、科大讯飞、CVTE、诺瓦科技。

## 学习经历

我在大一的时候加入了一个老师的实验室，在里面开始玩玩stm32单片机，实验室是做飞控项目的。在大一到大二的时候基本都在这里面度过，也学到了很多嵌入式的相关知识。

大二疫情期间开始学习RTOS、QT，也陆续自己做过一些小项目。加入了一个的学生创业团队，开始打一些比赛，挑战杯、互联网+、ican等。

大三开始基本就不打比赛了，继续学习嵌入式相关的内容，跟着韦东山老大哥的课程学习了linux的应用以及驱动方面的内容，也根据这些知识做了一个电子量产工具的项目。

## 实习经历

春招实习投了几个厂：OPPO、联发科、京东、阿里、网易、CVTE等。

其中拿到了OPPO、联发科、京东的offer。

最后在oppo的穿戴软件部门实习了三个月的时间，在其中的工作内容主要是跟OPPO WATCH2智能手表相关，上班状态基本是双休，工作日偶尔加班（因为我是实习生）。用到的技能有RTOS、LVGL、C、python等。

## 面试经历

接下来就是面试的记录总结啦，由于当初比较懒，基本上都是以记录题目为主，所以大部分题目没有写下自己的答案。

### 联发科面试 oc

#### 联发科提前批一面 30分钟 8.2

1. 为什么不留在oppo?
2. 说下中断和轮询
3. 平时学嵌入式有看什么书
4. 介绍电视线 balabala 两种工作的情况
5. 问我 更倾向于哪种
6. 音频 视频 外设的驱动
7. 框架性回答一个嵌入式系统启动到结束的过程
8. 反问

#### 联发科提前批二面 30分钟 8.9

1. 说说实习三个月的感受
2. switch\_context的底层原理 (freertos)

讲了下切换的场景，pendsv函数的实现，switch\_context里面关于pcb\_current指针的逻辑等。

3. 实习做的偏应用层，为什么要学习freertos呢
4. 面试官讲讲电视线的情况，业务做得好可能要两年，成为专家要四年。
5. 有学过linux/安卓吗 学过linux 写过一些驱动
6. 这些驱动是在开发板上写的还是在pc上模拟的？
7. 有什么想问我

## 总结

联发科提前批的面试给我的感觉是比较简单，面试氛围比较轻松。二面的时候面试官问了一个freertos的问题后，我回答得较好，然后直接就说后面不想怎么问了，后面也是第一个谈薪的offer。

### 星辰科技 oc

#### 星辰科技技术面 一小时 8.12

1. C语言基础 sizeof 指针、数组的大小
2. 局部变量存在哪里，malloc的变量在哪里
3. Linux怎么搜索所有.so文件
4. 平时有没有用github
5. 双向链表和单链表
6. Cpp的map、list、vector的底层数据结构

7. 多态的实现
8. 静态链接和动态链接
9. 动态链接的代码在数据段和代码段的分配
10. 项目管理的场景，你需要一个接口，需要别的部门的人来提供，他以很忙为由推脱，你这边也很急，要怎么处理？角色调换，又该怎么处理？
11. 介绍一个最成功的项目
12. Gdb的指令
13. Linux驱动的ioctl
14. 内核向应用层获取数据用哪个接口
15. 讲下i2c
16. 有什么想问我

### 星辰科技HR 半小时 8.17

1. 为什么不想留在oppo
2. 能接受几点下班？
3. 介绍下我们公司
4. 怎么看待互联网和半导体行业
5. 介绍最成功的一个项目，遇到了什么困难，领导不喜欢怎么办
6. 介绍最成功的一个比赛，遇到了什么困难
7. 在建模比赛中怎么分工，谁作为领导的角色
8. 期望的工作强度是怎样的
9. 家在哪里
10. 期望薪资
11. 有什么想问我？

### 总结

hr面后过两天就发意向书了，九月份谈薪的时候开的跟mtk差不多，拒掉后hr再提薪了一次，后再拒。不过面试的体验还是很好的！

### 大疆oc+签约

#### 大疆一面 一小时 8.19

1. 自我介绍
2. Linux创建线程有没有设置过调度相关的参数

主要考察第二个参数：

属性对象主要包括是否绑定、是否分离、堆栈地址、堆栈大小、优先级。默认的属性为非绑定、非分离、缺省的堆栈、与父进程同样级别的优先级。

3. linux 线程调度的几种方法

Linux系统的三种调度策略：

1. SCHED\_OTHER：分时调度策略（Linux线程默认的调度策略）。
2. SCHED\_FIFO：实时调度策略，先到先服务。该策略简单的说就是一旦线程占用CPU则一直运行，一直运行直到有更高优先级任务到达或自己放弃。
3. SCHED\_RR：实时调度策略，时间片轮转。给每个线程增加了一个时间片限制，当时间片用完后，系统将把该线程置于队列末尾。放在队列尾保证了所有具有相同优先级的RR任务的调度公平。
4. Linux中的条件变量怎么用？以及虚假唤醒的总结

5. 接上面条件变量 Cond wait后还持锁么
6. 硬件中断触发的全过程、堆栈的保存

进入异常步骤：

1. 处理器在当前堆栈上把xPSR、PC、LR、r12、r3~r0八个寄存器自动依次入栈。
2. 读取向量表（如果是复位中断，更新SP值）
3. 根据向量表更新PC值
4. 加载新PC处的指令（2、3、4步与1步同时进行）
5. 更新LR为EXC\_RETURN（EXC\_RETURN表示退出异常后返回的模式及使用的堆栈）。

退出异常步骤：

1. 根据EXC\_RETURN指示的堆栈，弹出进入中断时被压栈的8个寄存器。
2. 从刚出栈的IPSR寄存器[8:0]位检测恢复到那个异常（此时为嵌套中断中），若为0则恢复到线程模式。
3. 根据EXC\_RETURN选择使用相应SP。

7. 讲下MMU的相关知识
8. 多线程、多进程的方式
9. 共享内存的底层原理
10. 线程和进程的理解
11. 调用一个函数后，会返回到哪里？

这里应该是想问栈帧的变化

12. 程序在内存中的分配
13. 未初始化的变量打印出来是什么值  
全局的初始化为0 局部变量未知
14. 编译和链接的过程
15. 平时有没有用GDB
16. 讲下为什么断点调试可以停在那里

软件断点在X86系统中就是指令INT 3，它的二进制代码opcode是0xCC。当程序执行到INT 3指令时，会引发软件中断。操作系统的INT 3中断处理器会寻找注册在该进程上的调试处理程序。从而像Windbg和VS等等调试器就有了上下其手的机会。

17. Mmap的使用 了解页表么
18. TCP创建sever的过程
19. I2c、SPI讲下，i2c主从能互换不
20. 为什么不考研
21. 有什么想问我

## 大疆二面 半小时 8.26

1. 自我介绍
2. Oppo实习的内容
3. 四轴飞行器有几个自由度 6
4. 控制往左偏航 四个电机需要怎么操作
5. 遥控用的什么协议
6. 讲下另一个项目
7. 为什么项目用udp不用tcp

8. 讲下I2c和spi
9. Ic是怎么读写数据的
10. I2c速率有哪些 跟什么有关

### 大疆三面 半小时 9.3

1. 自我介绍
2. 介绍最有挑战性的一个项目 问了相关问题
3. 你的优势跟劣势
4. 最想从事哪方面的工作
5. 有了解大疆的产品吗
6. 跟自己做的飞行器有什么不同
7. 有什么想问我

### 总结

dji的一面问得比较多比较深入，后面两面就感觉比较水了，三面类似于hr面。面试官级别很高，是一个dji机器学习团队的leader，压力随之而来，所以面起来有点磕磕碰碰的。10.23hr电话谈薪sp，思考了两天后决定拒掉arm中国，签约大疆，去追逐自己的梦想hhh，总而言之，希望自己能够在dji学有所成！

### 荣耀 oc

#### 荣耀一面 半小时 8.21

1. 问笔试的题目，第一题还能怎样优化
2. 说下实习时候测试的心得
3. 毕业设计想做什么
4. 打算怎么开展
5. 从获得的奖项中选一个去讲
6. 有什么想问我

#### 荣耀二面 半小时 8.22

1. 在oppo实习负责的内容
2. 实习开发过程中遇到了什么困难
3. 工作地点的意向？
4. 更希望做上层应用还是底层？
5. 你在学校做的项目是在实验室做的还是自学的？
6. 有什么想问我

### 荣耀综合面 9.7

1. 对加班的看法
2. 华为和荣耀 oppo的比较
3. 大学怎么克服困难
4. 给自己的大学生涯打几分？为什么
5. 有独自旅游的经历吗
6. 对996的看法
7. 工作部门的意向
8. 有什么问我
9. 紫光一面 40分钟对加班的看法
10. 华为和荣耀 oppo的比较
11. 大学怎么克服困难
12. 给自己的大学生涯打几分？为什么
13. 有独自旅游的经历吗

14. 对996的看法
15. 工作部门的意向
16. 有什么问我

## 总结

荣耀的面试体验总体来说一般，技术类问题也没问很多，综合面的女面试官在面试的时候还在嚼口香糖，有点不是很礼貌，另外公司加班氛围特别重，所以对荣耀并没有太多的兴趣。

## 紫光 面试通过泡池子

### 紫光一面 40分钟 8.24

1. 技术面第一次见女面试官~
2. 四轴飞行器项目中负责哪些内容
3. 讲一下电子量产工具项目
4. 这个项目的需求是什么？市面上已经有了吗
5. 实习过程中做了什么
6. 对工作的意向，有没有哪些内容是比较想做的
7. 有什么问我

### 紫光二面 40分钟 8.29

1. 自我介绍
2. 介绍第一个项目~
3. 气压计的精度 5cm
4. 介绍第二个项目
5. 介绍实习内容
6. 面试官对实习做的产品很感兴趣 balabala蛮久
7. 说一下 Const的作用
8. 说一下 const int \*p和 int const \*p
9. 有什么问我
10. 面完就告知通过了

## 总结

面试体验还不错，两轮面试的面试官都比较有耐心，二面试官对实习做的东西比较感兴趣以及认可，但因为紫光是交叉面试，最后都是扔到池子里等人捞，所以最后没有适合的部门捞就没后续了。

## CVTE oc

### CVTE提前批技术面 1个多小时 8.25

1. 对c厂有什么了解
2. 介绍自己
3. 实习负责的内容
4. 有什么成长？
5. 找工程师review代码有什么心得？
6. GPIO能配置成什么功能
7. IO输入输出有哪几种
8. Cpp指针和引用的区别
9. Malloc和new的区别
10. Sizeof和strlen的区别
11. 了解什么总线？ I2c和SPI
12. I2c一次最多能挂载多少设备
13. 中断能传参吗？
14. 手撕代码 排序+二分查找

15. Linux由哪几个部分组成
16. Linux有哪几种设备
17. Linux查看内存状态的命令
18. 讲下网络设备？
19. 哪些是字符设备，他们有什么共同点
20. 内核态和用户态的区别
21. 有什么想问我

### CVTE提前批 HR面 9.2

1. 现在手上有几个offer
2. 期望的工作时间和薪资
3. 理想是什么？
4. 为什么会有这个理想
5. 平时获取知识的来源
6. 家庭情况
7. 对你影响最大的一个人
8. 假如给你30k、40k的月薪 你会怎么做
9. 有什么想问我

### 总结

cvte面试通过后，还需要去实习七天才发offer，我没去，所以没有拿到有具体薪资的offer。也算是一次证明自己的过程吧，毕竟是实习的时候第一家面试的公司，当初被拒绝，现在也回拒一次，扯平！

### 科大讯飞 oc

#### 科大讯飞一面 半小时 8.25

1. Linux ./ 到main函数的过程
2. 科大讯飞的语音识别你用过，怎么用的
3. Oppo实习负责的内容
4. 一个.c文件到运行的四个过程
5. 堆和栈的区别
6. 讲下linux的虚拟地址和物理地址
7. 动态库和静态库的区别

#### 科大讯飞二面 20分钟 8.28

1. 项目中的代码量
2. 实习的代码量
3. 大一时候为什么选择嵌入式呢
4. 面试官介绍智能家居的部门
5. 还有什么问题问我

#### 科大讯飞Hr 20分钟 9.1

1. 能不能接受先到合肥工作（我报的深圳base）
2. 在oppo工作的感受
3. 工作中遇到过什么困难
4. 为什么不考虑留下？
5. 现在还有哪几个offer
6. 还有什么问我？

## 总结

科大讯飞的面试给我的感觉，就是面试官时间比较紧张，不能跟我进行过多的交流，所以问的问题比较少，但是面试过程中也能感受到被尊重，给面试体验打个合格分数吧，最后也是果断的拒了。

### Arm china oc

#### ARM一面一个小时 9.8

1. 自我介绍
2. 介绍项目
3. Linux的启动过程
4. 怎么看.ko文件的信息

`lsmod` 查看已经安装好的模块，也可以查看`/proc/modules`文件的内容。

实际上,`lsmod`读命令就是通过查看`/proc/modules`的内容来显示模块信息的。

`modinfo` 显示模块信息

`modprobe`不需要指定路径，它会到默认路径下寻找模块。

`rmmmod` 卸载模块，但是内核会认为卸载模块不安全，可以添加命令强制卸载。

`depmod` 检查系统中模块之间的依赖关系，并把依赖关系信息存于`/lib/modules/2.6.18-1.2798/modules.dep`中。

`insmod` 加载模块，需要指定完整的路径和模块名字。

5. 你觉得有什么品质是这个岗位需要的
6. 熟悉什么数据结构
7. 单链表和双链表的区别
8. 什么时候需要用结构体 如何定义
9. 指针在32、64位操作系统占多少字节 为什么？

我们一般需要64个0或1的组合就可以找到内存中所有的地址，而64个0或1的组合，就是64个位，也就是8个字节的大小，因此，我们只需要8个字节就可以找到所有的数据。所以，在64位的计算机中，指针占8个字节。同理，在32位的计算机中，指针占4个字节。

10. 说说Static、extern
11. 变量存在什么区域
12. 数组越界访问会有什么后果
13. 函数调用栈的变化过程
14. 堆和栈上变量的生存周期
15. 检查一个32位整形变量的bit10是否为1的几种方法
16. 了解arm的 trustzone架构吗
17. Cotex-m的中断过程
18. 说说Cache
19. Write back和writer through的区别
20. 说说cache line 不一致的问题  
当时讲的跟这里差不多[https://blog.csdn.net/jasonchen\\_gbd/article/details/79462064](https://blog.csdn.net/jasonchen_gbd/article/details/79462064)
21. 线程和进程的区别
22. 线程的同步方法

### 23. 有什么问我

#### ARM 二面 9.24 一小时

1. 对加减密算法有没有了解
2. 怎么定义一个常量字符串
3. 堆和栈的区别
4. 说说四轴飞行器项目
5. 说说pid
6. 操作堆空间有哪些函数？
7. Malloc和calloc的区别

答：共同点就是：都为了分配存储空间，它们返回的是 void \* 类型，也就是说如果我们要为int或者其他类型的数据分配空间必须显式强制转换；不同点是： malloc一个形参，因此如果是数组，必须由我们计算需要的字节总数作为形参传递用malloc只分配空间不初始化，也就是依然保留着这段内存里的数据，calloc 2个形参，因此如果是数组，需要传递个数和数据类型而calloc则进行了初始化，calloc分配的空间全部初始化为0，这样就避免了可能的一些数据错误。

8. Strcpy和memocpy区别
9. Do while (0) 的好处

答：这样，宏被展开后，上面的调用语句才会保留初始的语义。do能确保大括号里的逻辑能被执行，而while(0)能确保该逻辑只被执行一次，就像没有循环语句一样。  
总结：在Linux和其他代码库里的，很多宏实现都使用do/while(0)来包裹他们的逻辑，这样不管在调用代码中怎么使用分号和大括号，而该宏总能确保其行为是一致的。

10. Static的作用
11. 栈溢出和堆溢出
12. 如何求一个结构体数组的大小
13. 在头文件分别定义static变量和普通变量会怎样？

#### ARM 三面 9.24 一小时

1. alice介绍部门、工作、员工发展
2. 有没有了解过测试理论
3. 白盒和黑盒了解吗
4. 汇编指令：smc和hvc了解吗
5. 了解arm架构中异常的同步中断和异步中断吗
6. Fiq和普通中断有什么区别
7. 讲一下栈帧

过程跟这里差不多

<https://blog.csdn.net/ylyuanlu/article/details/18947951>

8. 实习过程中有什么收获？学到了什么

#### ARM 四面 9.26 45分钟

1. 自我介绍 上海的面试官
2. 职业规划？
3. 应聘这个岗位的优缺点是什么？
4. 大学期间的代码量
5. 说说四轴飞行器项目的难点 遇到最大的问题
6. 说说pid控制算法
7. 说说多线程多进程
8. 说下互斥锁和条件变量

9. 条件变量的使用时：如果生产者唤醒消费者的时候 消费者未准备好？
10. 熟悉arm架构吗 熟悉m还是a系列多点
11. M系列和a系列中断的处理过程？
12. 函数调用一般用哪几个通用寄存器？
13. 函数返回时用到哪个寄存器？
14. 函数的返回值在什么时候入栈？
15. 有什么问题问我

## 总结

四轮面试官体验都非常不错。唯一的遗憾就是openday当天没有去深圳现场看看，面试过程中感受到arm对技术的要求还是比较高的，对arm架构的问题问得也比较深入，薪资也很有竞争力，在所有offer中是第二高的。是一轮体验相当好的面试，虽然最后没去，但在这里也祝arm china越来越好！

## 诺瓦科技 oc

### 诺瓦 一面 40分钟 9.11

1. 自我介绍
2. 介绍一个单片机项目
3. cortex m3和m4区别
4. I2c的时序 空闲时scl的电平
5. I2c可以接多少个设备
6. 讲下交叉编译
7. 讲下makefile由哪几个部分组成
8. GPIO有哪几种模式
9. 程序编译后存放在哪几个区域 堆和栈的区别
10. 给出一个结构体 求结构体的字节数
11. 说下反转链表的思路
12. 如何给地址0x67A9上的值赋为0xAA66
13. 讲下内存泄漏
14. 讲下链表和数组的区别
15. 平时有哪些调试手段
16. 有没有意愿留在实习单位
17. 有什么问我

### 诺瓦 二面 40分钟 9.14

1. 自我介绍
2. 介绍四轴飞行器项目
3. 介绍写程序期间遇到过的困难
4. 为什么要选择pid双环 单环不可以吗
5. 介绍实习的工作
6. 遇到了什么困难
7. 解决这个困难对你有什么提升？
8. 对诺瓦有什么了解
9. 想做什么方向的工作，假如让你做mcu裸机开发愿不愿意？
10. 有什么问我

## 总结

诺瓦深圳的面试难度偏向简单，深圳base的规模较小，而且只有12薪，开的base还算可以，但是各方面还是比大厂差了一些，面试体验还是不错的！

## 全志科技 oc

### 全志科技一面 半小时 9.16

1. 自我介绍
2. 如何实现一个队列
3. Insmod会调用驱动的哪个函数
4. 网络协议了解吗 tcp在第几层
5. 数据结构是自学的还是有相关课程
6. 栈和队列的特点
7. 用的内核的什么版本
8. 用设备树的话驱动如何编写
9. 写过哪些驱动
10. Linux项目给你带来的提升是什么
11. 最成功的一件事
12. Freertos怎么学的 任务切换的原理?
13. 除了任务切换对freertos其他底层了解吗
14. 有什么想问我

### 全志科技hr 半小时 10.8

1. 自我介绍
2. 大学最成功的经历
3. 大学最挫败的经历
4. 大学最有成长的比赛
5. 大学关系最好的三个人
6. 舍友怎么评价我
7. 怎么选offer 薪资 公司 地域进行排序
8. 现在有哪些offer?
9. 全志如果发offer, 开多少才考虑不去arm?
10. 有什么问我

## 总结

对于全志的感觉就是，这是一家不错的公司，网上被骂的声音较少，然后就是校招流程有些慢，也没有提前批。开的薪资也算中规中矩。

## 小米 oc

### 小米一面 9.24 35分钟

1. 介绍实习负责内容
2. 分别介绍项目
3. 问了pid相关的
4. 介绍i2c的读写时序
5. 数组和指针的联系
6. 堆和栈的区别
7. 线程和进程的区别
8. 介绍他们部门 手机部门做安卓驱动部分
9. 反问

## 小米二面 9.26 40分钟

1. 介绍下实习做的内容
2. 说下程序的内存分布
3. 说下堆和栈的区别
4. 说下栈溢出
5. 说下多进程和多线程
6. 共享内存的使用注意事项
7. 手写swap和strcpy
8. 说下malloc的底层原理

参考链接：<https://www.cnblogs.com/zpcoding/p/10808969.html>

9. 说下ioctl的实现原理 内核态和用户态怎么交互
10. 说说copy to user的底层原理
11. 说下缺页中断
12. 有什么问我

## 总结

对小米的印象就是，提前批投得慢，后面到正式批才捞我，最后也是顺利通过，开的价格也比较有诚意，sp最高档，但是相比其他手机厂和arm、大疆就低了不少，毕竟小米工作强度也低一些。面试体验以及流程的推进上还是体验不错的。

## 美团 待开奖

### 美团一面 80分钟 9.30

1. 自我介绍
2. 面试官介绍无人配送产品线
3. 花了很长时间依次介绍实习经历、三个项目经历。
4. 分别补充项目其中的创新点
5. 讲讲pid控制算法
6. 讲讲freertos任务调度的原理
7. 讲讲如何写一个字符设备驱动
8. 讲讲static的作用
9. 如果在类里面定义static变量 和在函数内定义的一样吗
10. 讲讲堆栈区别
11. 讲讲volatile

### 美团二面 50分钟 10.8

1. 自我介绍
2. 介绍实习的输出
3. 介绍实习项目的架构、freertos的应用
4. 介绍实习遇到的问题及解决办法
5. 介绍linux的电子量产工具项目
6. 介绍项目的用途及目标
7. 介绍实习过程中技术上最大的收获
8. 为什么做的项目都是c语言的，c++部分少？
9. 为什么不考虑来北京呢
10. 反问环节

## 总结

面试过程中能够感受到美团的面试官对技术是有一定要求的，两轮面试时间也花费了不少时间，可惜美团的加班氛围让我心生敬畏，并且无人配送的base在北京，所以表示了强烈的留在深圳的意向。估计大概率不发offer了。

## 个人总结

以上就是我个人的成长经历的介绍，以及秋招的面试经历，希望能对大家有所帮助！  
给后来的小伙伴的建议就是：

1. 秋招开始得越早越好，多投递自己喜欢的企业，提前批尽量不要错过。
2. 大三的小伙伴最好在大三暑期能够有一份不错的大厂实习经历，这样就算不考虑转正，也能在秋招中增加了很多argue的筹码。
3. 不要让自己卡在笔试上，leetcode刷给两百道题足以应付大部分笔面试中的算法题了，这里推荐先刷剑指offer的经典题，再刷些别的。
4. 嵌入式的学习方面，时间充裕的话，理想的路线我觉得是从mcu->RTOS->linux。



## 双非本科拿下Oppo sp

哈喽，大家好，我是仲一。今天分享的是一位双非本科生拿下oppo sp的秋招经验。当时，这位粉丝咨询我offer选择的时候，看到年薪31W这个数字，我以为他是研究生。后来，再三确认了，他确实是本科生。

大佬您好！我是本科双非应届生，拿了OPPO的驱动开发岗offer，然后base是■■■加一些其他的补助和年终奖，总包31w左右，不知道这个待遇怎么样

然后我还拿了海康威视的嵌入式软件工程师岗offer，但待遇还不清楚，请问您觉得这两个岗位性价比怎么样呢？哪个岗位的发展前景好一点呢？



单纯比较岗位oppo好点吧



像其他地点，工作内容，待遇，你应该也关心吧



你撤回了一条消息

嗯嗯，对的



看着以上这些因素哪个是你最关心的



做个简单排序，也很好选择



本科生能拿到这个薪资，真的已经超越了很多很多人。和这位粉丝交流下来，给我最直观的感受是，他很有礼貌，而且，很谦虚。能拿下oppo sp 和他平时的积累是分不开的。我就不多说了，下面的内容就是这位粉丝的秋招经验分享。

我真的是比较幸运的菜鸡了，走狗屎运了啦😂

海康昨晚开了14k

越努力，越幸运



我真的还是比较幸运的菜鸡了，走狗屎运了啦😂

和你平时的努力也是分不开的😊



体面



海康昨晚开了14k

不过，本科里面也算高的了



谢谢大佬的认可🎁





## 个人背景

学校：本科双非

专业：自动化

实际项目经验：参赛经验不多，只参加过学校的一个创新创业比赛（团队），拿了个奖；然后自己也会利用课余时间在网上找些例程进行学习，主要是单片机方面；

## 学习经历

目前大四上学期，我将我的大学经历主要分为两个阶段：

第一阶段：

大一~大二：这一阶段由于刚从高中来到大学，自己没能很好的对大学生活以及学习进行规划，也由于这两年做家教的影响，自己没能把重心放在学习上，所以这两年在学习上表现的很平凡，也没参加过什么比赛。

第二阶段：

大三~现在：大三开始把精力放在了学习上，除了学习专业课外，我也会利用课余时间学一下单片机，如stm32,esp8266等，但其实都是跟着教程了解一下，也并没有全部都学完，也是这一年由于巧合的机会我同学把我介绍给一个缺乏stm32知识的团队，然后我跟着团队参加了我的第一次竞赛，最后也拿到了校赛级的一个奖项。然后下学期在考研与找工作之间经过考虑最后选择了后者，4月份决定找工作，当时因为对找工作没有方向，然后看到年级群里面有阿里的师兄在宣传暑期实习生的招聘信息，所以就懵懵懂懂的报了C++岗位，然后在接下来的一个多月时间内断断续续的学习C++以及数据结构，结合视频将这些知识大概过了一遍，但是最后也没能过笔试（因为学的不扎实，也没去某网站上刷题，因为当时并不知道有这些网站），最后去了一个和学校有合作的小公司实习

## 实习经历

实习公司：一家做车载/终端产品的小公司

实习岗位：嵌入式驱动开发工程师助理

实习时长：暑期一个月

在7月份在一家小公司开启了我为期一个月的实习，在整个实习过程中，其实导师并没有给我们多少的帮助，最开始的时候给我们布置了一个关于stm8单片机的综合性任务，我用了半个月左右就完成，其他实习生晚一点，完成后导师让我们改进代码，后来也没怎么给任务或者安排给我们做了。然后在接下来的时间段了，我都在学习数据结构并且去leetcode刷数据结构专题（虽然进度比较慢）。然后又看了Linux相关视频与资料重新温习了一下linux的基础命令和操作。中后期的时候导师找我谈过两次话，想让我继续留下来实习，但我没有接受邀请。不过这段实习经历也让我对自己产生了肯定以及对这个岗位有了初步的了解。

## 面试经历

### 1. 面试情况

面试经历按照简历投递时间排序，由于忘记了具体的投递时间，所以有些只能写大概的时间，投递的岗位都是嵌入式/驱动软件开发工程师

| 公司名称             | 投递/笔试时间 | 笔试/面试情况         | offer            |
|------------------|---------|-----------------|------------------|
| 联发科 (MTK) (第一次投) | 7月25日   | 笔试挂 (没准备，裸考)    | 无                |
| 视源股份 (CVTE)      | 8月中旬    | 通过终面            | 需体验实习通过后才给offer  |
| OPPO             | 9月中旬    | 通过终面            | 拿到offer(应该是SP)   |
| TCL(C++嵌入式)      | 9月初     | HR面挂(惨)         | 无                |
| 海康威视             | 9月初     | 通过终面            | 拿到offer (SP)     |
| 小米               | 9月下旬    | 笔试挂             | 无                |
| 中兴               | 9月下旬    | 通过终面            | 收到offer call(没签) |
| 华为               | 9月下旬    | 笔试挂             | 无                |
| 荣耀               | 9月下旬    | 一面挂             | 无                |
| VIVO             | 9月下旬    | 笔试挂             | 无                |
| 联发科(MTK) (第一次投)  | 9月下旬    | 笔试情况未通知         | 无                |
| TCL(华星光电)        | 10月初    | 笔试通过直接跳过一面发二面通知 | 拒了面试             |

除此之外，我还投递过几家中小厂，但都是投着来找面试经验的，不完全统计在秋招中投递的公司数量在20家左右，大部分简历都通过了筛选，上面列表中的是自己心仪的公司的笔试/面试情况。也很幸运在10月中旬陆陆续续收到了OPPO、海康威视、中兴、CVTE的offer，最终选择了OPPO。

### 2. 面试问到的问题

以下面试问题都是自己面试完根据回忆想起来的，可能会有一些遗漏，因为自己太懒了，没有时间整理当时面试时候的回答情况，然后需要大家自己去查找相关问题的答案啦。在此也提醒大家以后面试可以录音或者录屏哦，不要被发现就行，这样更有利于自己面试后进行总结。

## 1) OPPO篇

### 一面面试题

1. 自我介绍
2. ++i与i++的区别?
3. Switch() { case} switch里面可以是什么类型的数据? 可以是float吗? 字符串呢?
4. 指针的高效性和灵活性? 如何体现?
5. 你选修的这些课程的原因是什么? 你是怎么学习的?

### 二面试题

1. 自我介绍
2. 问项目情况: (问的是实习的项目)
  - a) 这个项目是团队的还是个人的?
  - b) 在这个项目中你遇到什么问题? 怎么解决的?
  - c) 在这个项目的过程中最让你满意的是什么?
3. 后面的忘了, 哈哈

## 2) 视源股份(CVTE)篇

### CVTE一面面试题

1. 自我介绍
2. 问项目: 介绍项目 (介绍的时候还可以再精炼)
  - (1) 项目里用到了什么技术, 负责的是什么部分?
  - (2) 遇到什么问题?
3. ++i和i++的区别?

答: i++: 先赋值再++, 效率低;

++i: 先++, 再赋值, 效率比++i高

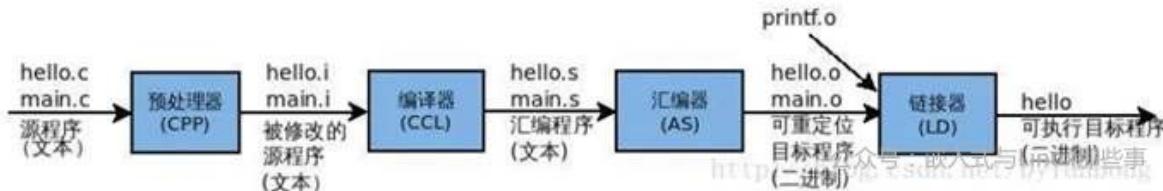
为什么++i比i++效率高?

i++: int temp=i; i=i+1; return temp;(i++需要开辟临时变量, 效率低)

++i: i=i+1; return i; (++i不需要开辟临时变量, 效率高)

4. Static关键字
  - (1) static+局部变量:
  - (2) static+全局变量:
  - (3) static+函数:
  - (4) static+类成员变量:
  - (5) static+类成员函数:
5. 指针和引用的区别
6. 拷贝构造函数里面为什么用的是引用? 有什么作用?
7. Linux命令: cut(怎么把CVTE\_STN中的\_识别并且将CVTE STN分开?), Sed, AWK?
8. 文件系统有哪些类型?
9. Bootloader
10. 汇编语言: DDR

11. 无论数据多少，复杂度最低，效率最高的排序算法是什么？
12. IIC最多能有多少个从机？
13. IIC软件中断和硬件中断的区别？
14. SPI协议原理
15. 为什么说TCP是安全的连接方式？
16. 程序编译的四个部分：预处理->编译->汇编->链接（各阶段分别生成的是什么文件类型.c/.o/.s?）



17. 中断函数有什么特点？如果在中断函数里面弄个10ms延时会怎么样？
18. 如果有一个递归函数，没有初始条件，它会一直执行吗？会怎么样？
19. 进程间的通信方式，最常用的通信方式是什么？
20. 如果一个进程堵塞了，你怎么让这个进程运行？
21. 你是怎么理解同步和竞争的？异常？（同步、异步、竞争）
22. GPIO的模式？方式？（上升沿/下降沿/双边沿之类的）
23. 会JAVA/Python吗？
24. 代码题

### 3. 无重复字符的最长子串

难度 中等    5979               

给定一个字符串 `s`，请你找出其中不含有重复字符的 最长子串 的长度。

#### 3) TCL篇

**一面：**主要深挖项目，根据项目做假设，问你可以做什么改进，并且很多都已假设形式提问，并且根据项目用到的知识点进行深挖，所以需要对写在简历上的项目要很熟悉，没有做过的不要写，不要给自己挖坑。

#### 4) 海康威视篇

##### 一面面试问题

1. 配置pwm需要设置什么寄存器？
2. 怎么用单片机测量pwm波的周期和占空比？（描述思路，用代码编程的方法）
3. 怎么将32位的int型数如`int a=12345678`转化为字符型string输出？（`printf("%s")`）
4. Main函数里面分几个区？（数据区，代码区，bass区，栈区，堆区），`string a="hello world"`存储在哪个区？加上`static`后在哪个区？
5. 截至目前你坚持的最长的事是什么？学习上或则生活上。
6. 成绩怎么样？偏科吗？
7. 面试官说我应变能力不错，面试不错，其他问题忘了，哈哈。

## 5) 中兴篇

### 一面面试题

1. C和C++有什么区别？各自有什么特点？
2. 链表和栈有什么区别？
3. 对中断有了解吗？中断发生时，CPU是如何处理中断的，如果程序在执行中断之后终中断返回异常，你该怎么判断呢？怎么处理？
4. 232和485协议有什么区别？
5. 使用的晶振是多少？你是怎么判断计数是一定精准的？
6. 平衡二叉树是什么？
7. 有用到浮点数吗？浮点数在内存中是如何存储的？

## 6) 荣耀篇

### 一面面试题

主要问项目以及实习情况，抓着问，很难受。没有问基础问题（八股文）。

### 3. 关于HR面

一般来说，HR面问的问题套路基本一样，但是不要以为到了HR面就可以放松警惕，就算你前面的技术面表现得多么好，如果HR面没准备好的话那也会功亏一篑，经过多个HR面，我发现HR大部分都会问一下几个问题：

1. 你为什么投递我们公司/这个岗位，你对我们公司有了解吗？
2. 你的家庭情况，学习情况
3. 你有什么爱好吗？有什么让你坚持了很久的事吗？
4. 你有什么优缺点吗？（说缺点的时候也不要说很明显的缺点，如很懒、没责任心啥的。可以说过于追求完美、不够果断等，因为这些对公司没啥影响）
5. 你对加班怎么看？
6. 你的期待薪资/范围是多少？
7. 还有其他的一些常见问题可以上牛客或者网上看一下其他面经哈哈

## 个人总结

### 在投递时间上

越早投递越好，大厂提前批一般在6、7月份就开始了，要抓好秋招提前批这个时间点，因为这个时候大厂的hc（岗位）是最多的，而且竞争可能没那么激烈，最重要的是有一些大厂提前批没有笔试，这对于那些没怎么刷代码提的同学来说是很有利的。

### 在公司投递选择上

不要全部都投大公司或者自己的心仪公司，也要适当的投递一些中小企业或者自己不那么喜欢的公司，如果你没有什么面试经验的话建议你先面小公司/不是自己心仪的公司先，这样可以为自己增加面试经验，为面试大厂做准备；同时，投递的岗位应该是你喜欢的，不要广撒渔网海投，这样不仅会浪费自己的经历而且还会让自己对面试产生厌倦，投递公司数量适可而止，并且投递的岗位最好专一点。

### 在面试准备上

在面了这么多面试中，首先，我发现技术面中面试官100%会问项目，所以一定要对你的项目要熟悉；其次，面试官会问你的实习情况，如做了什么，有什么收获（如果有相关经验的话）；然后是一些基础知识，如C/C++语言基础、数据结构、Linux基本指令、算法/手撕代码(Leetcode简单/中等题，不过我遇到的不多)。

在面试前，建议上牛客网上面查找相关面经，我在每次面试之前都会看一下对应的面经，然后根据面经预测面试官可能会问的问题，然后记录并且把答案搜索出来，从面试结果来看，有些问题确实被我预测到了，因为面试官问的问题基本都是那几个范围。

面试过程中，一定要注意一下自己的着装以及形象以及周围的环境，我在每次面试时都会洗一下脸以及穿着得体。然后在面试中最好保持自信，就算问到不会的问题也要思考一下然后讲出自己的想法，直接说不会是大忌。对了，还有就是一定要礼貌，在最后面试官问你还有没有问题问他/她的时候，要表现出你好学的态度，一般问2~3问题就好。

## 在心态上

还是要保持积极的心态吧，一般来说整个笔试/面试流程下来可能需要1个月甚至更长的时间，所以也不要干等待，在这段空窗期可以投递以及面试其他公司，自己协调好时间即可。

本人在秋招的过程中心态还是受到了很大的影响的，经常会失眠到凌晨3~4点，因为自己本来就睡眠不好，加上秋招一直没受到好结果，同时又断断续续听到别的同学拿到了多少多少W的offer，心理压力就更大了，因此也经常因为这是而失眠，那段时间真的很煎熬，没睡过几天好觉，但是很庆幸自己熬了过来，在10月份陆陆续续收到了几个大厂的offer，其中好几个都是我之前面试完之后以为挂了而且难受了很久的，现在想起来真的没必要。

## 最后寄语及感谢

不要和别人比，要和自己比！要认真对待每一次面试，即使这个面试的公司不是你的心仪公司，因为你的每次面试表现都会影响着你最后的薪资评级，所以还是要好好准备每一次面试。早得到的不一定是好的，晚得到的不一定是不好的！祝大家早日拿到自己满意的offer！！！

对啦！最后还是要感谢一下在牛客网上面找到的大佬的笔试面试总结《嵌入式软件开发笔试面试指南》，里面涵盖有简历建议，笔试面试题等相关内容，真的很全面。公众号是《嵌入式与Linux那些事》，里面干货挺多的，这次的面试我也是以这个为路线的，大家可以参考一下啦！！！



## 优秀的学弟

### offer

总投递了大约30多家公司，最终拿到海康、大华、中兴、HPE、小米、荣耀的offer，小米给的SP，最后去了小米。

## 基本介绍

本科双非自动化，硕士北京地区普通211院校，控制工程，主要的技术栈C/C++、单片机开发、freeRTOS、linux应用、计算机网络、操作系统。身边没有找嵌入式工作的人，可参考的指导的人比较少，导师课题和项目事情也比较多，准备时间大概从5月开始陆续学习一些C++的知识并刷题，到9月份才看到群主的面经，之前的学习都比较迷茫，后来也开始面试才慢慢知道学习到一些知识，9月份才开始大规模投递简历，其实比较晚了，但也收到了一些offer。感觉只要认准方向，加以努力，最后的结果一定是好的。

## 本科

参加了挺多比赛的，电子设计大赛、工程训练大赛、嵌入式大赛，最好的成绩是嵌入式大赛的国二，还参加了一些大学生创新创业项目（这个在简历上没提），大多做的是单片机的项目，常用的STM系列、MSP系列、瑞萨的单片机都接触过，对通信协议很熟悉IIC、SPI、UART等通信协议，电赛飞控组的常驻选手，本身就是控制专业出身，对控制算法很熟悉。

## 硕士

其实主要做nlp有一些机器学习，神经网络的背景，但没找这方面的工作，投的期刊论文和SCI文章都没什么用。硕士期间和企业合作做过一个嵌入式机器人的项目，主要负责底层的单片机程序的设计和与linux的ROS通信，这里花费了很多功夫，单片机上从C转到C++做了一个ROS底盘，这对找机器人相关的工作还是比较合适的，海康和大华都是机器人相关的嵌入式开发，面试也比较容易。

## 实习

实习是一家清华背景初创公司，因为实习时间比较自由选择了这家公司，做嵌入式软件的相关工作，关于物联网的感知层和传输层的搭建，主要负责linux应用开发，学习到了许多计算机网络TCP/UDP，以及MQTT、protobuf相关的知识，持续时间大概两个月，但是正好是8月到9月，其实写在简历上的内容也不太多，只能说有实习经历，并没有主要讲这部分的工作。但是其中做项目中遇到的问题和解决方法在面试中都用到了，还是发挥了比较大的作用，感觉实习最重要的还是从事具体有一定要求的项目的开发，要对自己工作做一个量化，如果只是简单的学习和普通任务的实现，就意义不大。

## 面经

因为准备的比较晚，主要想去的公司是大疆，但是大疆开始的比较早，所以大疆也是第一个投递的公司，大疆的一面技术面问的很深，当时也没背面经，全靠平时的知识，有些问题答上来也不是很好，最后终面被刷，很遗憾，感觉要是准备充分的话还是有机会的。其实面试了很多公司这里只列举几个有代表性的。

| 公司   | 流程                |
|------|-------------------|
| 大疆   | 笔试，一面，二面，三面（淘汰）   |
| 图森未来 | 笔试，一面（挂）          |
| 蔚来   | 笔试，一面，二面（淘汰）      |
| 寒武纪  | 笔试，一面，二面（淘汰）      |
| HPE  | 笔试，一面，二面，三面，offer |
| 小米   | 笔试，一面，二面，offer    |

感觉面试前不管多有把握还是要看一下面经，专门准备了相关的内容，如STM单片机从上单到运行程序的步骤、中断机制、IIC的总线冲突问题、SPI的四种模式等一些面试中常考的问题，也获得了不错的结果，后面基本上按照简历问都能通过面试。

## 大疆

### 一面 (9.5) (30min)

1. 串口DMA有什么缺点，为什么要换成ROSLib；
2. 你是怎么移植ROSLib的，ros的通讯方式是怎么保证数据的可靠性的；
3. 串口字节组成，串口的波特率，串口的传输位数，串口能传输多个有效数据；
4. DMA重复访问出错是为啥，cache (DMA的底层实现)； (没回答上来)
5. SPI的四种模式；
6. static关键字；
7. 字符串指针与字符数组有什么区别，存储在什么区域；
8. CPU大小端 (没回答上来)
9. 结构体对齐
10. 死锁，死锁怎么解决
11. 进程间怎么通信

### 二面 (9.8) (30min)

1. 项目
2. 问串口通信遇到的问题，是由什么原因导致的，有没有其他的解决办法
3. 使用过大疆的产品么，有什么问题
4. 来大疆想做什么

### 三面 (9.19) (30min)

1. 竞赛中获得遇到了什么问题？(为什么电赛成绩一直很一般)  
回答一开始强化技术，没有考虑不同环境的适应性。这个回答被问炸了，感觉面试官不太满意。
2. 是干一行爱一行还是爱一行干一行？
3. 如果技术认可你，让你去带一个小团队，你会愿意去么？
4. 工作规划？

### 总结

最后终面被淘汰，也是开奖时收到了感谢信，主要原因是一面试时的技术部分问题没回答上来，终面回答的也不好，简历可能是刚开始的简历，比较糟糕，最想去的还是不能放在一个面试，主要还是因为我投递的比较晚。

### 图森

#### 一面 (60min) (淘汰)

1. 介绍你熟悉的编程语言 (C/C++)
2. C++的特性，功能 (封装，继承，多态，STL)
3. 与C的不同
4. 构造函数的种类和功能
5. 虚函数的作用
6. 继承发生时构造函数与析构函数的顺序？有成员类时父类成员类与父类构造的顺序  
父类先于子类，成员类先于父类  
有成员类时父类成员类先于父类 (这个当时回答错了)
7. 移动构造 (不会)
8. 智能指针，回答了一个weak\_ptr
9. 数组与链表的区别
10. 左值与右值的区别，左值引用与右值引用

11. 结构体的字节对齐
12. 怎么去定义错误，和调试程序。
13. 类的类型转换，除了强制转换（也没回答上来）
14. 线程的通信和进程的通信，有什么不同  
    都可以使用共享内存，有什么区别
15. 懂不懂cmake，cmake中包含什么内容，cmake中怎么找到库的地址
16. 信号量是怎么实现同步的
17. 算法题：链表反转和链表排序的结合（大致说了并写了思路，最终没有跑）

### 总结：

图森问的C++知识感觉很深入，我只是背了简单的面经，肯定是不够的，这个也是反映了只背面经在真正的纯C++的面试中不够，后续可以多准备C++。

## 寒武纪

### 一面 (30min)

1. 项目相关
2. IIC的总线竞争
3. IIC如果主机复位，会有什么异常，怎么重新建立连接
4. RTOS的优先级反转
5. RTOS的死锁
6. 自旋锁于互斥锁的差别
7. 中断函数的写的时候要注意什么

### 二面 (10min)

1. 项目相关
2. 论文相关

### 总结：

寒武纪的一面技术面感觉还是很不错的面试，问的问题很深入，但是我回答的很好，面试官也是给予了肯定。不过这里碰到了一个新的问题，IIC主机复位的问题，这里虽然我没有准备过相关的内容但是因为我对IIC的通信协议比较了解，也推断出了出现的问题和解决的方法，这里就是如果面试中遇到你没有准备过的问题不要先忙着说不会，有的时候经过自己的思考是可以做出一些回答的，有些面试官看中的是解决问题的能力，和思考的过程。

但是寒武纪的二面不太清楚没有问什么相关的问题，面试很短，面完之后二个小时内就发感谢信了，感觉可能被刷KPI了。应该是比较晚了，岗位招满了。有的时候面试成功还靠的是运气。

## 蔚来

### 一面10.15 (20min)

1. 询问项目相关，主要内容
2. 给字符让快速排序
3. 给数组让二分查找
4. 二叉树前序，中序，后续遍历

## 二面10.20 (30min) (淘汰)

1. 询问项目相关
2. 进程与线程的区别
3. IIC的总线时钟频率计算 (没回答上来，不太满意)
4. IIC的协议构成，SPI的四种模式
5. TCP与UDP的区别
6. RTOS与linux有什么不同，为什么说linux不实时
7. volatile关键字
8. 大小端的判定，用程序实现

### 总结：

面试官对于我没有回答上IIC的频率计算很不满，有些回答认为我只是背诵理解的不深，感觉自己擅长的部分得了解每一部分的细节，对于自己简历上设计的相关只是要比面经了解的更深一些，最好有自己的亦一些思考。

## HPE

大多数人可能没听过这个公司，惠与中国，和惠普有关，我面试的是其下的一个子公司，师兄推荐的，工作比较轻松，这是我面试最难的一家公司，还是在线下面，不过给的不多，最后没有接offer

## 笔试+一面 (120min)

笔试直接线下做，懂得都懂现在做笔试也太难了，我是因为操作系统学的还不错，那一块的部分做的还行，最后综合看下来做的还可以。

一面是4对一的面试，会让去白板上画项目的结构图，程序的流程图，比较考验功底，还会让写代码的思路和实现，不过只写核心的部分，所以还不算要求很高，技术面试大多也是八股文上的常见问题和项目相关的问题，回答的还是不错。

## 二面 (90min)

二面是经理面，以为没有技术了，还是四个人对一的面试，问了一个多小时，

从结构上一步步拆分项目，感觉有点压力面试的感觉，每个问题都问道你不会为止，问的问题太多了，这里我也想不起来，没办法一一罗列了，最终还问了一个井盖为什么是圆的问题，我回答是因为受力均匀，不容易掉下去。但是答案被否定了，我最后还是坚持了我的回答。面试官答案不重要，重要的是思考的过程，我觉得我回答的还行，面试官给了个及格分，最后也是通过了二面。

## 三面 (60min)

感觉线下的面试会更真诚一些，与主管聊的也比较多，其实有时候没必要和面试官博弈撒谎什么的，感觉就自己什么情况说明白就行了，主管也是为了争取了一下薪资，但是和手上拿到的差的有点多，就给拒绝了，但感觉确实是一个很好的去除，不为钱的话可能就去这里了。

## 小米

## 一面9.24 (40min)

1. volatile关键字
2. 常用的排序算法（冒泡和快排），快排的实现思路
3. RTOS的message\_queue
4. IIC的通信协议的详细组成，如果地址为0x00则是什么结果？（广播）
5. SPI的四种模式
6. 项目中遇到的问题，怎么解决的
7. 串口怎么发送和接受的，帧头和帧尾怎么定义的，有没有校验位，校验位怎么定义
8. 代码题：C++打印循环

9. 代码题：deque删除第n个元素（删除元素后迭代器失效，需要重新赋值）

## 二面 (9.30) (30min)

1. C++的三种特性，继承，多态，封装（详细介绍一下多态）
2. 虚函数，纯虚函数
3. 通信协议：IIC和SPI，详细介绍其中的协议组成
4. freeRTOS的进程间通信
5. 按照项目去问（每个项目都会问）
6. IIC使用的时候有没有遇到什么问题（IIC使用中的总线竞争的问题）

### 总结：

小米面的比较晚，正好我已经经历了很多的面试，项目准备与面经准备的比较充分，基本上所有的问题都回答上来了，有些会自己引导问比较擅长的问题，所以最后拿到了小米的SP。

## 最后

总结下来自己的几个问题：

1. 投递的时间较晚。感觉最迟应该从8月开始投递，尽早准备很多公司都是8月可以投递，等到9月份笔试，10月份才进行面试（提前批每个公司都不一样，这里只说正式批），我虽然也是8月底开始投递，但是投递的时候简历还没有经历过面试的捶打，后面也是及时的更新了简历上的问题，但是导致一开始投递的很多公司按照初版的简历进行的面试，感觉很吃亏；
2. 项目做的不深，是做的时候做的不深，后面包装也是有限的，很多代码最好自己写，弄懂为什么要这样写，有可能的话要看详细的api的实现，比如RTOS中的通信问题，面试中被人问到使用消息队列为什么不能做数组等大容量数据的通信，一开始没注意到这点，后来看api的时候发现消息队列发送的消息是uint32，发送数组时是发送的地址，原数据容易被改写，这里应该用邮箱。这个点在后续面试的时候也经常被问道，或者主动引出，会显得对底层有一定的了解。
3. 基础知识不深入，如果要找直接对应的岗位，只看这部分的面经是不够的，比如图森的linux的C++面试，是第一次只抓着编程语言深入的询问问题，很多问题都会比面经上问的深一些，比如线程的共享内存的使用和进程的共享内存的使用有什么不同，会遇到什么问题，比如析构和构造有成员函数时顺序，当时也是有很多没回答上来。

几点建议：

1. 虽然大家都会看面经和八股，但是最好是有自己很擅长的知识，比如我操作系统学的比较好，再结合RTOS可以讲很多东西；
2. 及时的更新简历，简历和自我介绍要尽可能的突出自己的优势，我一开始都是自我介绍时介绍，就会显得自己的自我介绍很多，有时会被面试官打断，后面就把一些准备的一些问题或者面试官很多都会询问的问题（比如有什么科目）放在简历上，后面感觉面试会轻松一些，也被面试官评价简历上写的东西都比较擅长；
3. 面试公司前要做对应的准备，比如我面试的HPE是主要做WIFI，我之前准备了WIFI的基础知识和计算机网络的一些，面试的时候就很有用，提前预习公司的知识很有用，比如中兴面试问到了嵌入式岗位懂不懂什么是OTN，我当时肯定回答的是不太懂，但是如果有的话会有一些帮助；
4. 自己的项目要充分掌握，如果自己擅长的东西被问倒之后是很被动的，自己也要突出自己在项目中的主要工作，比如我的项目最多可以讲10分钟左右，但实际不会讲这么多；
5. 要有自己思考和解决问题的能力，如果面试中问道的问题没有准备，可以根据有关的知识做对应的思考，不要急于否定自己的不会，思考的过程也同样重要；

# 一位音视频方向读者的秋招总结

## 作者简介

2021应届本二，目前年薪20，在某行业头部大厂从事嵌入式多媒体开发（音视频应用方向）。

由于没有耀眼的学历，学习的资料和企业实际应用有些偏差，2020年大三时秋招处处碰壁，当时最高只有10k，于是抛开薪资选择了一个自认为不错的方向进行实习。

**努力终有回报，实习半年，毕业三个月后，拿到了当前行业几乎所有头部公司的offer。**其中包括海康、大华、宇视、新华三、紫光、萤石、华晨等等。

## 面试前准备

### 资料推荐

#### 嵌入式软件工程师笔试面试指南简介

嵌入式软件工程师笔试面试指南，详细分成了笔试面试准备，笔试面试八股文总结，面经总结，名企笔试真题解析等四个部分。

其中，八股文又分成了C/C++，数据结构与算法分析，Arm体系与架构，Linux驱动开发，操作系统，网络编程等六个部分。

**资料未经作者授权严禁各类培训机构私下传阅，如有发现，必将追究责任！**

公众号：嵌入式与Linux那些事 CSDN：嵌入式与Linux那些事 来源网络，个人整理，转载声明

这份面试指南涵盖了95%以上应届生面试可能遇到的问题，如果真的想拿高薪，那么建议你好好研究。你可以去csdn上付费，也可以联系到他免费给你（别问我）。

## 要不要刷笔试题、力扣

**摘要：要刷笔试题，力扣刷不刷看你要面的公司考不考。**

笔试题因人而异，基本上应届生没有漂亮的项目经验，所以应届生的面试中几乎所有公司都会让你做题，由于我有实习经历，而且项目我做了一大半，简历丰富，所以我没有刷题。

还有一点就是收集信息，比如某公司一共几轮面试，有几轮笔试面试，笔试题难度。这个需要你找到一些相应的社群去了解，比如下面的图



昨天  
江波龙笔试没做就面试了[捂脸]

昨天

江波龙找笔试完一个月才有动静



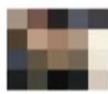
## 我为什么不好好做英伟达的笔试[苦涩]

星期三



母鸡呀，笔试被刷了[捂脸]

星期二



我没收到笔试题，直接面试了

星期二



## 笔试考到低通滤波器

2021/10/18



## 你们不会笔试题

2021/10/16



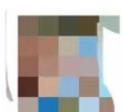
现在还有人做笔试？

2021/10/16



## 笔试不都是搜？

2021/10/16



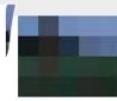
笔试?

2021/10/16

搜索框：笔试



取消



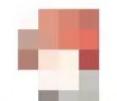
大华笔试题。

2021/10/10



现在才笔试有点太晚

2021/10/9



大华笔试有点悬咋办，影响大不

2021/10/9



我没有笔试的还能改吗

2021/10/9



没有笔试吗

2021/10/9



我全志投了有两个月了 笔试做完之后直接没...

2021/10/8



大佬拿秋招当笔试练手的

2021/10/8



我笔试都没咋写

2021/10/8



笔试做好久了，才来通知

2021/10/7



2021/10/7

我笔试完没消息了

这个群是我找资料的时候加的，都是应届面试的同学，要加的话找那个资料的主人。

如果你的学校比较好，有计划冲击华为、微软那些公司，那么你就需要提早准备起来。

由于没有好好准备笔试，电话面试问到项目问媒体层知识乱杀，人家邀请我到线下给我做了套22年的笔试题，我有挺多做错的，尴尬的很。不过笔试做不出来也没啥大问题，offer该给还是给，薪资压点

## 企业资讯获取

- 1、学校资源  
学校、学院就业网站，宣讲和招聘信息，招聘会等
- 2、企业官网  
大中型企业招聘官网、公众号。
- 3、招聘APP  
我用的boss，其实都可以，但建议用当下比较火的
- 4、社群资源



2021/10/14

小米是北京 oppo是南京



2021/10/14

南京做兄弟



2021/10/14

我投的南京



p

2021/10/13

...痴绝处 正式员工有没有加班? **南京**那边做...



2021/10/13

我在**南京**实习过



2021/10/13

我上次**南京**华为要了20w, 我都没敢多要



2021/10/12

**南京**奕斯伟开了吗



2021/10/12

...金5% 涨薪评价1k一年**南京**奕斯伟20\*14...



2021/10/11

**南京**还有地平线?

在这种应届生面试的社群里, 你可以知道某地的嵌入式类型公司哪些比较好, 有什么坑, 有没有笔试, 薪资标准等等资讯, 我非常推荐大家找找看相关社群

**简历制作**

很多细节在我推荐的那个面经里面有

|             |    |
|-------------|----|
| 2. 书写简历注意问题 | 15 |
| 2.1 个人信息    | 15 |
| 2.2 邮箱      | 15 |
| 2.3 教育经历    | 15 |
| 2.4 专业技能    | 15 |
| 2.5 实习经历    | 15 |
| 2.6 项目经验    | 16 |
| 2.7 荣誉及奖项   | 16 |
| 2.8 个人博客    | 16 |
| 2.9 自我评价    | 16 |
| 2.10 其他注意事项 | 16 |

这里我讲讲我的心得，如何针对性的去写个人技能

首先要确定自己的方向，比如

- 做路由相关，那么你就要着重写上网络相关的知识
- 做应用相关，就着重写应用相关的知识
- 做驱动相关，那么就着重写驱动相关

道理大家都懂，首先写上自己会的，然后再找到你想要的方向的龙头企业，以它作为参考去写你的简历  
以下面这个为例

#### 职位描述

---

DSP开发工程师

- 1、本科以上学历，信息、通信、自动化、计算机等相关专业；
- 具有DSP或其它嵌入式系统上的至少2年开发经验；
- 2、熟练使用C/C++语言，熟练阅读英文技术文档；
- 3、具有DSP或其它嵌入式系统上的开发经验，对视频、音频、图像处理、码流封装（PS/rtp/ts/rtmp）具有一定的基础；
- 4、熟悉DSP/BIOS实时操作系统，了解Linux等嵌入式操作系统，具有多个OS下开发经验者优先；
- 5、熟悉主流DSP，具有Ti DSP（C6000、DaVinci）、海思SOC开发经验者优先。

你的简历应该如下

- 熟练掌握C/C++，具有良好的编程规范
- 熟练掌握linux应用层编程，文件IO、标准IO、多线程、多进程、SOCKET网络编程
- 熟悉H264、ACW、PCM等常用音视频编码标准
- 熟悉RTP、RTSP传输协议
- 了解RTP、PS封装
- 具有XX平台XX年开发经验

这么样的简历完全匹配，能够匹配行业龙头企业的要求，那么行业中游企业的要求也可以满足

再举一个例子

### 职位描述

---

#### 1、工作内容

负责数据中心交换机驱动软件开发工作。

#### 2、任职要求：

- 1) 计算机等相关专业本科及以上学历；
- 2) 扎实的计算机基础知识；
- 3) 熟练掌握嵌入式Linux环境下C语言开发；
- 4) 对计算机软件开发及设计有强烈的兴趣；
- 5) 熟悉Linux kernel优先考虑；
- 6) 熟悉x86汇编，mips汇编以及arm汇编等优先考虑；
- 7) 熟悉Makefile/shell等脚本优先考虑。

简历如下

- 熟练掌握C/C++，具有良好的编程规范
- 熟悉Makefile/shell脚本编写
- 熟练掌握linux应用层编程，文件IO、标准IO、多线程、多进程、SOCKET网络编程
- 熟悉XXX版本内核
- 了解xx汇编、xx汇编，能够通过反汇编快速定位并解决问题
- 具有XX平台XX年开发经验

两个例子写得很粗糙，大家懂意思就行，根据自己的方向写上相关知识点

### 不会怎么办？

总归有不会的，学就是了。简历匹配可以让你有一个面试的机会，能学到什么程度面试聊到什么程度就可以了，面试官知道你不会就不会再为难你。

简历上的东西你多少得说出来，如果简历写上了你一点答不出来就GG。

### 先有面试机会再说

### 去不去外包

网上讨论很多大家自行参考，我直接说我的结论

**如果实在没有选择了，可以去外包。缺钱的话，可以第二第三份公司选择外包，不缺钱，有选择的余地就不去**

**建议在最开始面试可以先投外包进行面试练习**

跟华为搭边的百分之九十都是外包

## 面试

### 自我介绍

一开始肯定让你自我介绍开场

- 1、如果学校好就介绍一下自己哪个学校毕业的；学校不好就说自己是xx级毕业生
  - 2、在学校做的项目，做的比赛/项目
- 这一点尤其重要，**介绍你项目的同时要给出面试官问你的空间**。比如说，“在项目中我解决xxx问题/xxxbug定位/学会xxxx知识”，如果面试官顺着你说的问你，你将这些准备好的问题正确的回答上，算一个好开头
- 3、阐述你对他们公司这个方向很感兴趣，想在这个方向进一步去钻研

### 技术面试

面试官对你项目的了解肯定不如你，所以主动权在你手上

如果一开始介绍项目的时候，面试官没有顺着你给出的点来问，那么一般情况如下

- 1、介绍一下项目框架、逻辑、项目做了多久
  - 2、你在项目中的职责，贡献的代码量
  - 3、具体问项目中某一个模块的实现
  - 4、再具体到某一个技术点
  - 5、有的面试可能不会限于你简历上的，会循环34问到你不会为止  
还可能有，面试官遇到啥问题搞不定，要做什么需求，然后问你的思路
  - 6、你在项目中解决的最难的bug
- 重要**，体现你的能力。在别人说加打印的时候，你聊栈溢出；别人说项目逻辑的时候，你聊整体框架问题
- 7、和同组成员如何合作(代码管理)
  - 8、代码风格
  - 9、零散的问一点基础知识，如C相关内容
  - 10、反问。建议问工作内容，**要不要经常出差**

上面列出的12涉及到的细节一定不能被问倒，最开始的几个问题都是来判断这个项目是不是你做的，一旦被问倒，说明你没做过

回答的注意点如下

- 1、不要结结巴巴，这样会让面试官觉得你沟通、表达能力不行
- 2、不要问什么答什么，比如问到你线程detach，你聊完了可以聊聊关于线程属性、遇到的bug，线程和进程对比。  
问什么答什么会让面试官感觉很累，如果说一个技术点，你把这个技术点比较重要的内容说出来，他对你的好感就会增加
- 3、不要说的太流利，太过于流利会让人感觉你在背书，适当停顿假装思考
- 4、答不上就说不会，不要硬答
- 5、答不上的内容你可以引申一下，比如面试官问我项目中有没有用到cache，我说没有，然后讲了一下cache是什么，有什么用，可能可以使用的场景，相关的还有一个violate关键字  
这样就算没答出来还是可以化被动为主动
- 6、不经意间体现你的优点，比如介绍项目框架的时候你是怎么思考的，怎么样可以增强项目的扩展性；你发现了哪里的代码设计的不好，为什么；如何团队合作；如何定位bug；项目的不足

总的来说，面试的时候要主动一点，面试官没义务去发掘你的亮点，在面试的过程中尽可能的展示自己

**面试的过程也是你考察公司的过程，包括面试官的风度、问题的专业程度等**

苏州科达的面试官是给我感觉最好的，问题有深度，你不会了还给你台阶，全程友善。

行业某龙头给我压力面试，每次问道我不会的，挖个坑我踩了，就搁那笑，气死我了！

具体技术细节在我推荐的面试资料里，大家自行学习，我会在后面的目录列出我被问到比较多的问题

面经中的部分内容如下

|          |    |                    |    |                   |     |                    |     |
|----------|----|--------------------|----|-------------------|-----|--------------------|-----|
| ◀ C/C++  | 25 | ◀ 数据结构与算法          | 66 | ◀ 树               | 79  | ◀ 数组               | 102 |
| ▶ 关键字    | 28 | ▶ 链表               | 68 | ▶ 二叉树先序遍历（递归和非递归） | 79  | ▶ 最大子序和            | 102 |
| ▶ 内存     | 34 | ▶ 删除单链表的重复节点       | 68 | ▶ 二叉树中序遍历（递归和非递归） | 84  | ▶ 原地移除元素           | 103 |
| ▶ 指针     | 36 | ▶ 如何找出链表的倒数第K个元素？  | 69 | ▶ 二叉树后序遍历（递归和非递归） | 89  | ▶ 合并两个有序数组         | 104 |
| ▶ 预处理    | 42 | ▶ 如何找出链表的中间节点      | 70 | ▶ 后序遍历代码（非递归）     | 92  | ▶ 代码               | 105 |
| ▶ 变量     | 46 | ▶ 反转链表             | 71 | ▶ 层次遍历            | 94  | ▶ 查找常用字符           | 106 |
| ▶ 函数     | 46 | ▶ 环形链表             | 72 | ▶ 求二叉树的深度         | 97  | ▶ 寻找数组的中心索引        | 107 |
| ▶ 数组     | 50 | ▶ 单链表相交，如何求交点？     | 74 | ▶ 如何判断二叉树是否相等     | 98  | ▶ 数组中数字出现的次数       | 109 |
| ▶ 位操作    | 51 | ▶ 回文链表             | 75 | ▶ 如何判断二叉树是否是平衡二叉树 | 100 | ▶ 数组中数字出现的次数 II    | 110 |
| ▶ 容器和算法  | 53 | ▶ 移除重复节点           | 76 |                   |     | ▶ 数组中缺失的元素         | 111 |
| ▶ 类和数据抽象 | 56 | ▶ 用普通算法实现两个有序链表的合并 | 78 |                   |     | ▶ 按奇偶排序数组          | 113 |
| ▶ 面向对象   | 57 |                    |    |                   |     | ▶ 数组是否存在重复元素       | 114 |
| ▶ 虚函数    | 64 |                    |    |                   |     | ▶ 有序数组出现次数超过25%的元素 | 115 |
|          |    |                    |    |                   |     | ▶ 有效的山脉数组          | 116 |
|          |    |                    |    |                   |     | ▶ 最长连续递增序列         | 119 |

## HR面

有的公司HR面会刷人的，面试前可以问问看群里的人，不要犯病一般都没问题

之前一个哥们，被问未来规划，他来一句要去创业，然后挂在HR面

HR面的可能问题如下

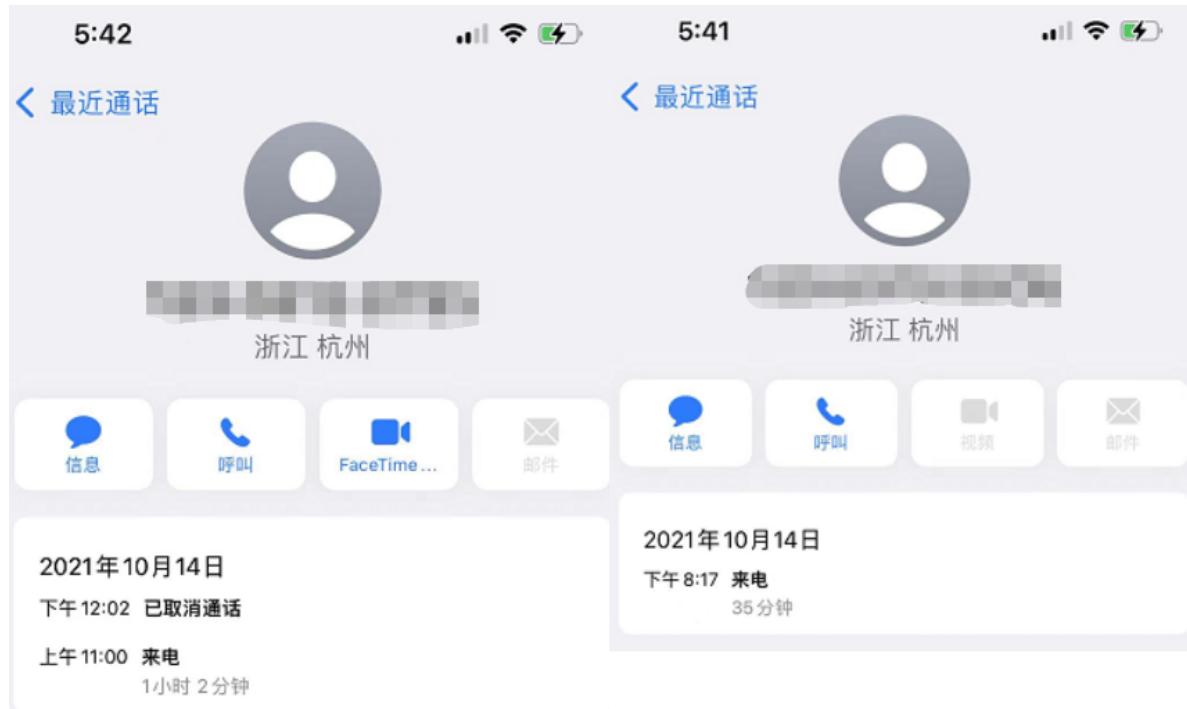
- 1、什么时候入职  
我一般回答女朋友在这（不管有没有都可以说）
- 2、籍贯，异地的话，为什么决定来这个城市发展  
我一般回答在当前方向深耕两三年，先成为项目中的骨干成员（别犯病）
- 3、未来规划  
我一般回答在当前方向深耕两三年，先成为项目中的骨干成员（别犯病）
- 4、如果和领导观点冲突怎么办  
一般要回答，视野不同，领导在行业更久看的一定更远，保留意见，比较领导和自己看待问题的方法（一般公司想要服从的人，但是也有例外，大疆就喜欢激进的）
- 5、决定在公司待多久  
我一般回答，贵公司在行业内属于领导地位，我希望在贵公司长远发展，我也喜欢这座城市，女朋友在这，家里条件不错，有计划结婚后在公司附近买房
- 6、父母是做什么的  
就是问家庭条件，是不是会在这座城市长久发展，我一般都是往家庭条件不错的方向回答
- 7、为什么从上一家公司离职  
别说上家公司坏话，我回答是项目完结没有新项目，现在闲了，人员流动较大，项目组就只有我一个人
- 8、能不能接受加班  
不管它要不要求加班你都说能接受，明确说不能接受会让人觉得你没有抗压能力（决定养老除外）  
目标公司是否加班可以通过网络获取，我也传了一份[2021年公司加班情况表](#)，可以免费下载
- 9、反问

五险一金，住房、餐补、通信、交通等补贴，补贴是否包含在工资里

大概问题就是上面列的，**总结就是往积极的方向回答，不要说负面内容**

## 我的面试经历

由于我确定了自己的方向，投的细分行业公司问的都差不多，所以直接合起来讲了



我感觉一面有点压力面试的意思，每一个点追着我问，问到我不会为止，一般面试是半小时，超过半小时就没什么问题了。比如这家公司一面问了很久，二面就问了20多分钟然后开始类似hr面试的问题，三面HR，之后发offer。

下面总结一下这段时间被问到的问题，从简介绍

### 基础

由于我简历的重点是项目，所以基础问的少，项目经历不够丰富的应届生自行通过我推荐的面经进行学习

#### 负数是怎么存在内存里的

在计算机中，数据都是以正数的补码的形式存在的。正数的补码是其本身，负数则是以其本身的正数的补码的形式存在的。

#### 有没有遇到过bus error

没遇到过，但是我知道他的起因和排查方法

进程的虚拟内存空间实际上是对物理地址的一个映射，通过mmu实现，页表管理，操作系统和编译器会用内存对齐来做优化，通常就是4字节对齐，所以int,float这种类型的起始地址都是4的倍数，而short的起始地址是2的倍数，double的起始地址是8的倍数，假如此时对一个不是4倍数的地址a进行解引用 (int \*) a，就可能会出现总线错误，这个出现的情况还要具体看是哪一种操作系统。总线错误一般不会出现，出现的情况多半是使用了指针的强制转换

#### 栈溢出怎么查

gdb、hexdump看反汇编，然后我介绍arm寄存器，怎么看堆栈信息

#### 线程detach

自动回收资源，然后我介绍线程其他属性，讲了线程栈溢出

#### 面试官追问，你们的项目线程栈大小多少

可以使用ulimit -s查看，我们使用的是8M

## 为什么是8M

用的是默认大小，为什么默认是8M我不知道

## 口述一个函数指针

```

1  char *strcpy(char *dest, const char *src);
2  {
3      xxxxxxxxx
4  }
5
6  int main()
7  {
8      char *(*pFunc)(char *dest, const char *src);
9      pFunc = strcpy
10 }

```

项目中的使用是不同层之间函数传递。比如媒体层的算法触发上报函数就是通过将应用层给的函数绑定回调实现

## typedef 与#define宏的区别

如果是都定义char \*

然后对 a, b操作

typedef会将两个变量都定义为char \*

define只会定义a, b是char型

## 大小端，如何判断大小端

经典面试题，然后我补充了要区分大小端的使用场景，比如要考虑通信双方字节序不同，socket网络字节序默认大端等等

## 文件IO和标准IO区别，哪个带缓冲

## 线程、进程区别，使用场景

## 媒体层知识

由于我是做音视频媒体层的，所以这部分知识问的比较多，相关知识点我整理在了我的博客，大家可以根据链接学习

## YUV格式，422p转420p怎么转

### [详解YUV数据格式](#)

此外还和面试官讨论了RGB转YUV的效率问题

## sensor和主控的连接方式

控制信息用I2C，数据信息通过MIPI传递。提了一嘴，这个东西是做驱动关注的，我只知道大概，细节不懂。

## ISP模块和缩放模块的数据交互

在缩放模块的配置中，有一个video buffer，我们设置的是2，通过类似队列的方式进行交互。

## ISP、缩放、算法、编码模块里面有什么可以配置的属性

## 每个算法分别通过什么指标进行成功的判断

## 编码出来的视频、音频结构体中分有哪些参数

## 音频帧的概念

## codec编出来的音频断续怎么办

驱动相关，不会

## 264、265区别，264知道哪些nalu type及应用场景

### OSD的实现

项目中使用工具将图片转成二进制实现，内存拷贝方式，我还介绍了操作字库的方式实现

### 视频编码的原理

[消除视频冗余的方法及原理](#)

### 码率控制相关内容

[音视频中的码率控制](#)

QP、bitrate调高调低的作用

给面试官讲了我们码率调优方面遇到的问题，主码流次码流分别用了多少码率

### RTP中的时间戳

我是用的RTSP是从live555移植的，RTP封包中的时间戳是根据h264的频率/码率来计算的  
其他的时间戳一般是底层的驱动返上来的，媒体层也可以做

### PS、RTP封包

[PS流详解](#)

[RTSP详解](#)

### 画面花屏，出现拖影、快效应原因及解决方法

### 项目

#### 项目做了多久

#### 你贡献了多少行代码

我说的七千左右

#### 选择的芯片，芯片的资源

比如ddr多大，系统分区怎么分，flash多大  
芯片是否提供dsp用于编码，内置还是外置的codec

#### 每个模块的细节实现

#### 你在项目中做过哪些模块，哪些代码是你写的

#### 你的项目一共开了几个线程

这个问题被我朋友预测到了，我假装惊讶然后一条一条捋一遍，接着他就问每个线程的细节实现

#### 电机人形追踪，将电机移动步长和画面像素点进行匹配

设置区间，多次测试，给出经验值

#### 代码怎么管理

git，然后我介绍了常用的git指令，

#### 权限是都开放的吗

我说我们只有两个人，把大伙都整乐了

#### 我回答出所有问题之后，他们暂时没得问了，我就讲了一下对这个项目的思考

如，每个模块可能存在的问题，我们项目没有做到位的地方，可能产生延迟的地方，和面试官讨论整个项目框架，如何搭建方便扩展功能

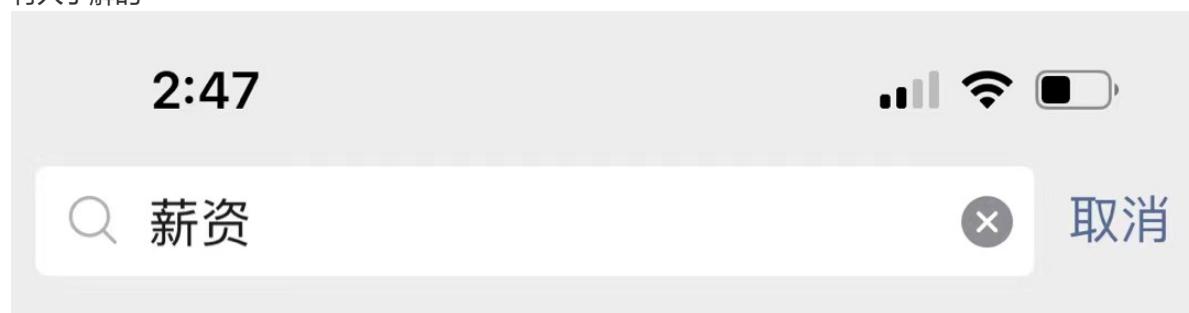
#### 代码风格问题

参考华为的规范。

函数命名，结构体加ST，枚举EN，全局g，函数传参传地址不传值等等，还有海康用的do while(0)，不用goto，用函数封全局来跨文件交互而不用extern等等

## 谈薪资

需要横向对比，知道目标公司的薪资标准。可以通过公司里认识的人了解，或者问就业相关社群里有没有人了解的



星期日

辜负了给的薪资

星期六

今年普遍都很高呀

星期六

大佬们，能说说双飞本的普遍薪资嘛？

星期六

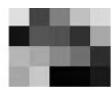
[聊天记录]...离职，走的时候告诉我他薪资 2...

星期六

他问我期望薪资，我就说 18w

星期五

基本薪资 11k\*16



...什么，但是要维持二三十年的高薪资还是...



小三

星期三

兄弟们，问个问题，a薪资的时候会不会把of...



一木

星期三

感觉确实，应该好不少，加上这个薪资

在我看来

- 双非本正常水平年12~15  
即月8~10
- 双飞本比较优秀年18~20  
即月10~15
- 双非硕比较优秀年20~25  
即月15~18
- 211、985硕年25以上  
即月20以上

今年(2021)的时代背景是，国家宏观调控，互联网受到打压(蓝色、黄色logo企业上市受阻)，美国制裁，中国大力发展芯片行业。资金流入芯片行业，所以嵌入式人才会被高薪挖走，行业人才出现缺口，于是其他的厂不得不跟着提高应届薪资标准来抢人。

对于薪资大家还是要理性看待，除了学历、能力，市场也是一方面。

## 最后

志晖君在一个采访里说过，他并非天才，只是提前做规划，然后全部完成了而已。

作为一个本二，本身不占什么优势，我是大四秋招实习了半年多，在第二年的秋招才拿到一份年20的大厂offer。这可能只是别人上一年的起点。我想说的就是，大家不要因为自身条件不够好就妄自菲薄，抓住每一个你能抓住的机会，提前规划，然后实现。

祝大家能够进入自己理想的公司，拿到自己想要的offer



微信搜一搜

嵌入式与Linux那些事

## 一位通信工程本科生的秋招经历

哈喽，大家好，我是仲一。秋招基本接近尾声了，不知道大家找工作怎么样了。

我建立了一个秋招群，每天虽然上班很忙，但是也会时不时关注群里的消息，回答下大家的问题。

根据大家在秋招群里聊天的情况看，很多人都拿到了不错的offer。而且，今年嵌入式的薪资，整体比去年又高了一个档次。

晚上10:07

... 1.0K/s ⏺ ⏻ HD HD WiFi 48%

【嵌入式与Linux那些事】3群(499) 🔍 ...

小米我二面答得不好

arm 老哥开了多少 ‘’

██████████ 七瓜 大佬，你  
目前手里过 30w 的有几个呀



出身黑龙江，盲猜应该就是哈工大  
的硕士了吧 😂

这群里面人均年薪 30w??

这群里都是软硬通吃

这群里面人均年薪 30w??

30w 不多吧

这群里面人均年薪 30w??

拖群友的后腿了 😊

这群里面人均年薪 30w??

大佬，小米给你开多少@ ━ █



公众号：嵌入式与Linux那些事

不过，秋招没有拿到满意的offer也不要紧。春招会有很多人毁约的，各大厂名额也不少。

大家记得关注下春招的时间段：**2月中旬 ~ 4月底**。趁着这段时间，可以好好的查漏补缺。关于秋招春招有任何问题，欢迎大家找我学习交流。

接下来，我邀请了几位公众号的粉丝，分享下自己的面试经验，也欢迎大家向我投稿你的秋招之路。

今天分享的是一位拿下中兴，联发科，海康等大厂 offer 的优秀本科生的秋招经历。

## 个人介绍

大家好，我是lambda，就读于一所功课211院校通信工程专业的一名本科生，在2021年9月底结束我的秋招工作，拿到理想的offer，受仲一的邀请，写下我的秋招总结。

感谢仲一的文档，总结归纳了许多关于嵌入式软件岗的面试相关的知识点，还有所有给予过我帮助的大佬们。

相信许多大三的学生都会面临升学或就业的选择问题。尽管我的学校有超过10%的保研率，但是依照我的成绩依旧不足获得保研名额。所以我就买了几本考研的参考书，莫名其妙地跟风走上了这条考研路。

不过因为个人原因，我放弃得相对较早，大概在三月份左右就放弃了考研，全力准备电赛，同时也为秋招做准备。

尽管2021年的全国电赛因为疫情原因被推迟，但是我依旧从前期的准备工作和校赛中学习到了宝贵项目经验，这宝贵的电赛经验也是我秋招相对顺利的重要原因。

根据我个人和周边参加电赛同学的经历，我非常建议电子类专业的学生参加电赛，不管是为了求职，保研，都有非常巨大的加分作用，甚至是直接的录取门票。

当然除了2021年的全国电赛，我在其他时间中也有参加过智能车竞赛，数学建模之类的比赛。

当然不管是含金量还是学习价值，智能车竞赛远不如电子设计大赛，可能是因为有学长和指导老师们的祖传代码与祖传电路的原因。至于数学建模比赛，虽然他在我们校内加分方面的含金量充足，但是为了求职技术岗，其学习价值也没那么高。

下面是部分面试企业的面经。第一，在我拿到满意的offer后，剩下不少企业的面试都拒了。第二，当时我投的企业也不算多，所以也就只分享了以下几个企业的面经。

## 海康

### 技术面

1. 简单自我介绍
2. 问了C语言的基础知识，如静态变量，预处理之类的
3. spi, iic, uart这些通信协议中选一个讲
4. 深挖项目的细节
5. 反问

大概是这些，海康的技术面试相对来讲还是比较容易的，只要项目是自己做的，并且项目质量过关都没啥问题

### HR面

1. 个人情况
2. 在校期间的学习情况
3. 项目中遇到的困难
4. 如何看待996
5. 期望薪资
6. 对海康的看法
7. 反问

## 大疆

### 性格测评

大疆在笔试之前会有一份性格评测，据说性格不符合大疆的企业文化的人会直接刷掉，有大疆员工表示如果这个评测过不了那就是真的不适合。个人猜测大疆的企业文化有激情和活力，以效率和工作至上的员工，所以我抱着这么一份心态也通过了这次的评测。

### 笔试

毫不夸张的讲，大疆的笔试是我在所有嵌入式岗位中难度最大，范围最宽，深度最深的笔试，详细的笔试内容，在各大论坛中搜索大疆嵌入式笔试就能搜到。从arm, 语言, 网络, Linux, 操作系统都有考察到。但应该他们也知道这份试题的难度，即便我在有题目不会的情况下也能通过。

### 大疆一面（挂）

1. 问了项目中的细节
2. iic协议，这一块问了许多，应答信号，读写操作之类的都问了，如果iic协议没有熟到能手画时序图的熟练度就有可能被问倒
3. 问了操作系统方面的知识，但是个人仅了解一点rtos，再加上我当时并没有系统学过操作系统这门课程，这里应该是导致一面挂的直接原因，后面对操作系统大概学习了一点
4. 询问我对大疆的了解，之前有了解过他们公司的机甲大师，以及玩过固定翼航模，就在一片欢声笑语中结束面试

## OPPO

1. 这里我投了硬件岗，因为在他们嵌入式岗位中写到此岗位竞争激烈，请谨慎投递。当时我挺希望进入这家公司，所以我投递了相对竞争较小，base于东莞的硬件岗
2. 由于oppo校招相对较晚，并且要求线下面试，我并没有参加他们的面试
3. 硬件岗的笔试非常简单，仅仅是性格评测，甚至连一些其他企业硬件岗笔试中的常出现的dc-dc 电路，模电，运放等问题都没有。
4. 个人猜测，这家企业位于东莞的硬件岗要求并不是那么高。

## 中科创达

这是我唯一一次的线下面试经历，这家企业来到我们校进行了宣传和面试。第一天是宣讲会，大概介绍了他们公司的规模和业务，并收取简历，尽管他们做的是外包业务，在宣讲中他们会进行美化和包装，并不会直接提到外包

### 面试

1. 细聊项目，这里因为面试官是做软件开发方向，对硬件并不了解，还需要我额外地去解释硬件方面的细节
2. 询问了主修课程，会几门编程语言，并谈到了rtos，这时已经进入秋招的尾声，加上前面折戟大疆的经验，这里还聊了不少。
3. 考察了C语言的编程能力，问了static变量，预处理等。最后手撕了一个关于斐波那契数列的代码，都是一些非常常规的问题。
4. HR面：问了些个人职业规划，对就业城市的期望等问题，好好回答就行

最后开的薪资不高，再加上主要为外包业务，遂拒。

## 中兴通讯

尽管这家企业的风评很一般，在投递岗位之前我对他还是有一定好感度的，经过他们的招聘之后好感全无。

我投递的岗位是嵌入式软件岗以及硬件开发岗。但是在面试之前，HR给我打电话希望给我调到其他岗位，并问了我的个人意向，在我明确表明除了硬件岗以外不去其他岗位。最后还是被面莫名其妙面了无线网络开发岗。

### 一面

1. 简单聊了聊我的项目
2. 编程语言的掌握
3. 对出差的看法
4. 大致介绍了他们部门的工作方向
5. 了解我的家庭情况，有无女朋友之类的

### 二面

1. 在校期间的学习情况
2. 问了英文水平，并简单使用英文做了一段介绍
3. 个人发展意向，因为我项目中软硬件都有所涉及，问了我希望做软件还是硬件
4. 由于这个岗位需要出差，所以也问了我对出差的看法
5. 反问环节

## 联发科

### 一面

1. 项目介绍。
2. 项目中涉及到设备驱动的开发，问了几个通信协议
3. 项目中遇到的困难
4. 关于编程语言的掌握能力，主要还是问C
5. 问了关于团队协作的看法
6. 家庭情况，还有有无女朋友的问题
7. 期望薪资

### 二面：

1. 这次的二面和一面比较接近，基本等于一面的审核

## 总结

对于电子方向本科生来说，我非常建议目标为工作的同学去参加电子设计大赛，这个比赛能给普通本科生带来非常大的一个提升，拿到什么奖项不重要，学到了多少干货才是最重要的。感谢在比赛中与我一同拼搏的队友，感谢仲一的文档总结，也感谢所有帮助过我的贵人，谢谢！

## 机械专业转行嵌入式成功上岸

### 个人背景

本211，硕985，专业机械电子工程，

项目经验1：stm32+改进PID控制+组网通信

项目经验2：一个信号采集测试系统，FPGA+QT+以太网通信逻辑设计+电路设计

### 学习经历

我本科是机械设计制造及其自动化的，学的除了数学物理基础课程，还有就是机械的专业课，跟嵌入式相关的也就是数电，模电，单片机，机电一体化设计这几个，数据结构、操作系统本科没有接触过。本科期间做的有意义的事情就是参加了很多比赛，拿过全国大学生数学竞赛一等，还有一些创新创业的比赛，挑战杯，互联网+。本科期间过的比较随意，没有太多规划，课程成绩比较好，就保研了。硕士的方向是机械电子，期间做了两个项目，然后重新学的数电，模电，因为有单片机基础，然后就用stm32做了项目。因为课题项目需要，又学了FPGA，对着正点原子买的开发板学的。这也是我面试过程中，面试官主要问的内容，说得详细点一般够说个10分钟左右。

读研期间，主要还是接触硬件的东西多一些，画板子调试，软件编程方面，就是c比较熟悉，但是没有接触过操作系统。操作系统也是在秋招的时候才准备的。

### 秋招准备

因为机械出身，行业不太景气，师兄师姐都推荐转行，但是好转一点的就是嵌入式的方向了。互联网的纯软开，算法转的不多，难度也较大吧，基于这些原因，我也是目标岗位方向是嵌入式软开。为了给简历增加点东西，4月-6月找了个学校附近的公司实习，总共就实习了两个月，中间还有请假，我算了一下，实习天数就21天，写点python脚本和java的ADB调试。实习比较水，所以在那边就经常学习点自己

的东西。

实习辞职之后，暑期就没有找实习了，主要是我当时数据结构基础不行，我觉得得自己好好准备一下，实习给我带来的体验不好，没学到东西，所以7月份我就安心准备秋招的知识点，也就是这个时候看到知乎上群主发的pdf，顺着群主的目录大纲一点点学习。到了8月20号我才开始投递秋招的简历（所以没赶上大疆）。那个时候的水平大概是，掌握C/C++的语法知识点，算法题就是一般难度的可以做做。后来发现笔试题难度也都不是很大，面试就完全按照pdf来的，然后结合自己了解到的，又补充记录了一些知识点。操作系统的知识仅限于一些简单的八股文，开始投简历的时候，还跟着群主推荐的韦东山的视频看了看，了解了一下开发的流程。

所以总的来说，我的学习周期不是很长，而且暑期中间还出去玩过，到了秋招的时候，也是比较慌的，觉得自己基础知识不扎实。

## 面试经验

秋招投了也挺多的，没有记录，20多家吧。因为住在上海，所以都是投的上海的岗位。基本都进面试了。说几个印象比较深刻的。

### 上海瀚讯（通过）

这家公司不是我自己投的，岗位是驱动开发，因为是猎头推荐的，所以一面就直接跳过进的二面，二面是现场主管面，过去的时候在会议室两个人面试你，轮番提问。都是问项目经历，挖项目细节，30分钟，第一次这么正式，很紧张，头上冒汗。不过两天后，猎头通知我通过了。

### 小米（通过）

小米一面：

小米的岗位是无线通信软开，然后面试都比较水吧，项目细节基本没怎么问。一面面试官人很好，就问我四个问题：

- 1.堆和栈的区别
- 2.栈溢出会产生什么问题
- 3.怎么检测到栈溢出，如何规避
- 4.进程和线程的区别

后面就是聊家常了，聊一些业务相关的，还有个人职业规划。最后说完直接就摆明了说我把你的简历送到下一个面试官里面，这是唯一一个当场说通过了的。

小米二面：

二面是个年轻一点的人，上来我自我介绍完就问我你对计算机网络熟悉么，我说学过一些，然后就问了两个很简单的C语言问题，然后问我对无线知识了解多少，我说没有学过，他就没问题了，我以为凉了，后来小米HR微信联系说通过了

### 乐鑫科技（通过）：

乐鑫总共有三面，第一轮是技术面，一个小时，问的很多，简历的东西全部问一遍。大概的话。问到一个小时就结束了。第二轮是hr面，聊家常。第三轮是主管面，问一问项目细节和自己擅长什么。不会深挖。后来谈薪的时候加面了一个H2芯片的研发主管，谷歌回来的，面了一个小时，后半个小时是手撕了一道软硬件timer的题目，不会做，跟着面试官提示，思路走出来了。面试官对我表示了肯定。后来想要我过去，但是还是拒了。

### Tplink (一面挂)

很不好的面试体验，看起来嫌弃我专业不对口，项目直接没问，也没有手撕，问的第一个问题是全局变量和局部变量同名，编译器是怎么处理调用的，我没答出来，问我有没有学过编译原理，我说了解一点，然后匆匆结束了，总共20分钟，就是流水线的那种面试，20分钟一个人。

### 广联达 (一面挂)

C++岗位，基本上从C++的语法问到了设计模式，问题有指针和引用的区别，多态，排序算法，时间复杂度，稳定性，UML，设计模式，项目代码规范性，我大概回答出了70%-80%，以为过，但是挂了，可能因为没有深入的C++项目

### Marvell (通过)

这是家外企的半导体公司，然后岗位是做车载以太网芯片的系统工程师，两轮面试，都是线上视频，每一轮一个小时，都是三个人在会议室。第一轮先问项目，半个小时，没深挖，简历过一遍。后半小时，面试官打开一个pdf的试卷，在线完成，题目范围很广，C/C++基础，arm架构的，信号处理的，一道深搜的岛屿问题，最后一道岗位相关的，状态变化的流程，全是英文，没见过，面试官提示我做出来了。整套卷子难度比较大。不过我是口述，一道一道做，面试官也给了肯定。第二轮换了三个人，深挖项目细节，十分痛苦。过了一周hr联系问期望薪资。外企没有加班，较为轻松，但是我由于报了太多，后来就没联系我了。

### 华为2012实验室 (通过)

岗位是通用软开的。机试不是很好做。一面一个小时，半小时聊项目，半小时手撕，二面也是半小时项目和经历，半小时手撕，三面主管面40分钟，聊的项目经历，然后主管跟我介绍了业务项目的重要性和保密性。面试流程很快。

## 个人总结

我秋招准备不是很充分，但是也收获了比较满意的offer，整个秋招下来我认为最重要的几点如下：

1.简历要打磨好，突出自己的优势，我的竞赛比较多，然后我就单独做了个荣誉奖项的模块，**项目经历是面试流程中最重要的一环**，项目最好要有两个，其中一个必须很熟悉，大概就是项目背景，研发流程，技术的学习方法，有没有团队工作，对项目中的技术要十分了解，一般面试官通过你的简历可能不能透彻掌握你的项目内容，你要有一套完整的说辞将你的这个项目完全表述出来，前期可能借助讲稿，面多了就记住了。面试过程中，针对岗位，突出项目中涉及技术，给面试官往这个方向提问的暗示。就比如我的项目是用FPGA做的，但是面的岗位是C语言的，那软件部分还有通信设计就要多强调一些。

2.手撕和笔试的难度不在一个等级上，手撕题就参考那个pdf就行了，自己再补充一些。笔试的话，就刷题就行了。我没做多少道题，看的都是专题性的，除了基本的一些数据结构题，还有比如动规，回溯，深搜等，hard的题没时间可以不刷。前期会比较痛苦，做多了就好了，注意积累和形成方法论，像回溯，动规这些都是有基本的代码框架的。

3.保持平常心，找工作不容易，企业找人也不容易，如果基础不是很扎实，多强调自己的学习能力，特别是项目和岗位不太匹配，或者面试官对项目不感兴趣的时候，学习能力是企业招人的标准。

4.给转行朋友的建议：个人觉得转行不难，像中兴还有国企的一些企业就比较简单，当然工资也不是很高，想去工资高一些，就自己提前打算，做好自己的学习路径，学好基础知识，从GitHub或者一些渠道找一个完整的项目坐下来，现在就有很多智能家居的项目，但是也逐渐在大众化，最好还是做一个软硬件都有，比较全面的项目，哪个企业不想要一个代码基础好，又有硬件知识的嵌入式工程师呢？



## 个人随谈（附谈薪话术）

秋招又要开始了，针对大家最近问的多的问题，说点什么吧。

### 笔试准备

准备时间前两天有人在知乎咨询我说，7号大疆要笔试了，有没有什么办法突击下（提问的时间是5号）。

咨询服务：图文咨询

7号大疆嵌入式笔试，但是我是学电气的平时接触到的嵌入式相关的很少，有没有短期恶补的方法

2022-08-05

拒绝回答

立即回答 嵌入式与Linux那些事

我能想到的办法就是背背八股文而已，不过，感觉用处也不大。毕竟，嵌入式的八股文太广，太杂了。一时间不可能都理解，消化掉的。只能拼运气，看能不能背到原题。

问了下群里的读者，大疆笔试题型为单选+多选+填空+判断+简单+编程2道，一共1小时。一小时时间确实不多，2个大题基本没有太多思考的时间。

考前突击只对笔试考察填空，选择占比大的用处大点。而最后的大题最少要A出一道，才可能有面试机会。

最后，我把《嵌入式软件工程师笔试面试指南》给了这个读者一份，希望对他有帮助把。

晚上9:28

...0.1K/s ⏺ HD HD WiFi 31%

## 【嵌入式与Linux那些事】3群(500) 🔔 ...

晚上6:03

xdm, 大疆笔试有填空, 选择吗,  
还是只有大题?



我拍了拍" - chan"

大学课本课后作业 plus 版



我确实想不出来了

晚上6:08

群主要跳槽?

晚上6:17

都有, 还有简答, 编程

杭州-嵌入式-仲一: xdm, 大疆笔试有填  
空, 选择吗, 还是只有大题?

晚上6:22

单选 + 多选 + 填空 + 判断 + 简答 + 2 编程，时间 1 小时

杭州-嵌入式-仲一：xdm，大疆笔试有填空，选择吗，还是只有大题？

一个小时太少了



单选 + 多选 + 填空 + 判断



嵌入式与Linux那些事

## 刷题

看过《嵌入式软件工程师笔试面试指南》的，肯定知道，我在里面多次强调要重视算法题，并详细介绍了如何一步一步准备算法题。

像华为，小米，大疆，oppo除了笔试会考察算法题，面试也会手撕代码的。算法题的重要性不言而喻。

和互联网一些岗位相比，嵌入式考察的算法题没那么深，像图论这类的没必要深挖。付出的时间和最后的收益不成正比。因为很少有公司在笔试面试会考到图论的内容。

作为基础一般的同学，扎实地把中等难度以下的题目都做会了，已经很不错了。就像数学考试，最后的压轴题是给考150分的同学留的。我们能做的就是把前面的简单题都做对。可是，又有多少人能做到这一点呢？

扯远了，回归正题。剑指offer是标配，有时间我建议大家刷两遍。很多大厂面试考察的都是剑指offer的原题。就算不是原题，也能在剑指offer中找到类型差不多的题目。

仲哥，现在嵌入式算法好卷

我这两天做了科大讯飞和虹软科技的笔试

已经怀疑人生了



7月24日 晚上22:02

怎么又去搞算法了



前两天有个本科学弟，拿了图森未来的 offer

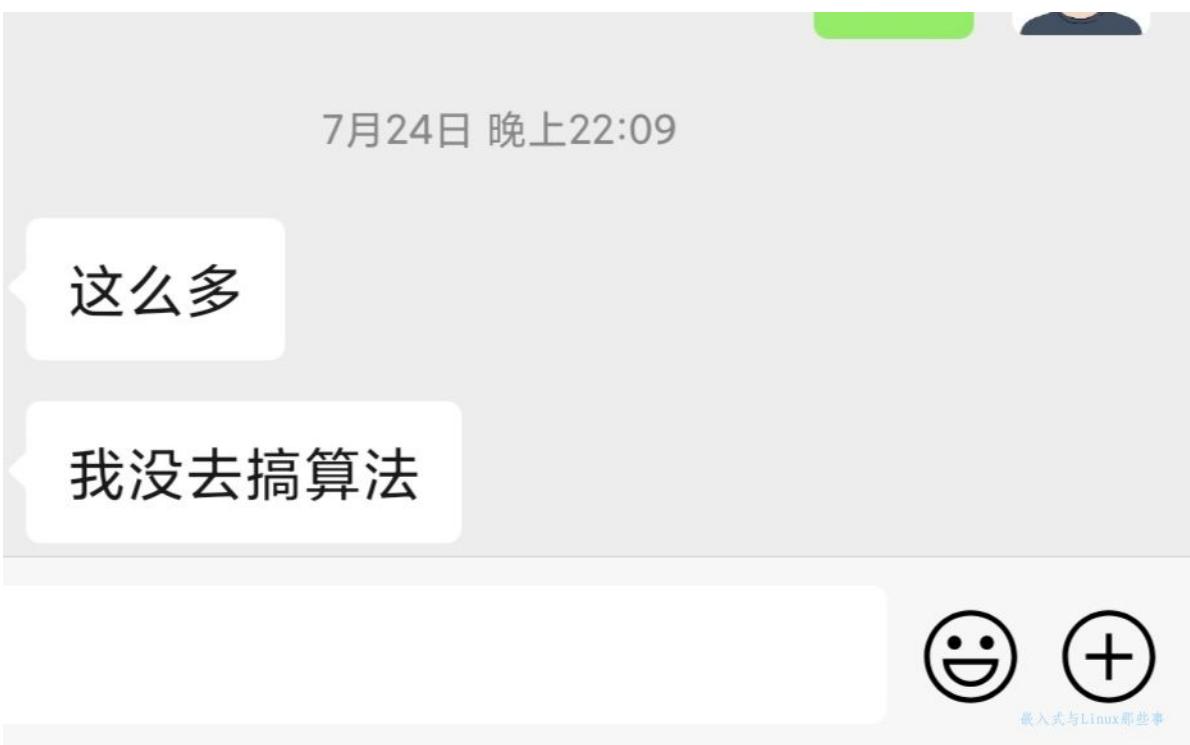


估计有 30k 了



嵌入式





我没去搞算法

是嵌入式岗位笔试题

算法题

有点难



你不会，别人也不会



别慌



7月24日 晚上22:10

今天上来一个前缀树 有思路但没实现出来

我的思维还停留在反转链表这种基础题



从明天开始继续搞算法题吧 😊

剑指 offer 刷起来啊



没刷吗



应该有前缀树的题目



有的 我也不知道我为什么没做 应该是版本不一样

再拾起来🤔

剑指 offer, 两遍起步



嗯 牢记了

太累了...还要做实验

一米八的汉子都要顶不住了

哈哈



划水



做实验，不用那么认真👉



还早大庄缩我时间了 而且八日右

迦南人小组的叨叨叨

## 中期

该做的...逃不掉

中期敢说吧



嵌入式与Linux那些事

我哲库应该最近二面

上海的

做 gpu 的，arm 板上下链

可以



哲库看能开多少钱



往 30k 冲



往年好像挺多的

我努力先准备二面

嗯 他们岗位可我无敌匹配 一面面试官说

可以



不过我一面一紧张，合并链表没做

出来 有个bug 巨尴尬



就说了一下思路



嵌入式与Linux那些事

这就是你的锅了



社死了

默写十遍



难过了好久

第一次面试 真是...尴尬 😞

绝无下次 md

哈哈



二面好好准备



肯定会撕一道题的



嗯

必

tmd

加油老哥

希望早点看到你年薪百w的一天



嵌入式与Linux那些事

如果有更多的时间，可以准备下leetcode hot100，这题目也是常考的。最后，不要忘了去牛客练习下输入输出。很多公司的笔试系统是acm模式的，需要自己处理输入输出。如果能做到上面我说的那些，算法题就没什么大问题了。

## 复盘

及时复盘很重要！

面试的时候，大家可以准备个录音笔或者手机，把面试时和面试官交流的过程记录下来。面完之后，把这些问题整理下，自己再过一遍。

仔细想想面试官为什么会这么问？面试官想得到什么样的回答？如何能把这些问题回答的更完美？自己当时的回答的差在哪里？经历过这样的几次复盘之后，你的下次面试会变得更加淡定和从容。

当然，大家整理的这些问题和答案后，也可以加我微信LinuxDriverDev，投稿给我，整合到《嵌入式软件工程师笔试面试指南》，让这份资料可以帮助到更多的人。我也会给你稿费作为奖励。大家可以参考去年几个很棒的学长整理的面经：

[优秀的学弟](#)

[进大疆了](#)

[双非本科拿下oppo sp](#)

[校招总拿SP是什么样的体验？](#)

## 谈薪

谈薪谈薪这方面我没什么经验，不过可以就我了解到的和大家大概说下。

一般大家在面试前对所面试的公司的薪资的上下限都有大概了解（不了解的可以去offershow微信小程序查）。如果你觉得你面试表现不错，可以报一个高一点的薪资；如果觉得面试一般，报一个上下限中间的薪资。这样就可以避免报价过高被刷掉。

中大型公司都有完善的薪资定价流程，一般在面试前HR就对你的薪资有了大概了解，很多时候，面试问你就是走个流程而已。

除非你特别优秀，面试表现不错+其他大厂offer (+国奖+论文)。这时候你就可以有足够的资本和HR谈薪了。关于谈薪的一些话术，我在公众号和《嵌入式软件工程师笔试面试指南》都写过，大家可以去看下。

总结一句话：你要有资本并且值得公司为你报出更高的价格。

当然，我也见过有些人忽悠HR成功的。某同学A厂和B厂的offer还没拿到，和A厂谈薪时就说已经有了B厂的offer，待遇是XXX，希望A厂给的薪资能比B高。和B厂谈薪时，说已经拿了A厂的offer，待遇是XXX，希望B厂给的薪资能比A高。

不得不说，这位同学很机智，但是这样做的风险很大，有可能因为要价过高，A和B的offer都丢掉。不建议大家这样做，如果你这么做了，你也要能承担相应的风险。

## 谈薪话术

首先，非常感谢XX能给我Offer。

我心目中理想的薪酬价位是XX元，之所以提出这一标准有以下几个原因：

第一，我现在手头拥有XX的offer，薪酬范围大概XXX。因为XX在行业内是首屈一指的，所以想争取一下，进入XXX。在大公司，自己的能力和价值会有更大的展现舞台。而且，XX这种一线大厂都会有成熟的薪酬体制，这种薪酬福利体系一定能够让员工有归属感。我也希望HR您可以满足我的薪酬要求。

第二，岗位匹配度。在和面试官聊天的过程中比较顺利，像面试官提到的XXX等这些工作内容都是我比较熟悉的，在熟悉了公司的工作内容后，我觉得，我可以很快上手项目，为公司创造价值。

第三，个人实力和上升空间。我在硕士期间，拿了三次一等奖学金，在华为杯，兆易创新杯等比赛获得过全国二等奖，在CSDN写作一年半，累计发表原创文章145篇，累计访问量45W+。以上这些事实都可以证明我的技术能力和可塑性还是比较好的。

第四，最后我相信，通过我的努力和表现，我会在很短的时期内，也许就在试用期结束后，我展现出来的价值甚至超越我的期望薪资。

## 最后

临近秋招，有很多同学咨询我秋招怎么准备，让我帮忙修改简历等等。我平时上班也很忙，看手机的时间比较少。大家有问题可以直接问，看到了我都会回复的。有时候没回复可能是因为忙，如果过了两天没回复，可以再戳我下。

最后，祝愿大家都能拿到心仪的offer！秋招，冲冲冲！

## 名企笔试真题解析

### 小米嵌入式软件工程师笔试题目解析

哈喽，大家好。我又来分享笔试题目了。今天分享的是某大厂的嵌入式软件开发工程师的笔试题目。这份题目很奇怪，操作系统，数据结构，网络基础，Java，C++，数据库，正则表达式，Linux都考到了。当时做题的时候，我都怀疑发错卷子了。。。还好最后两道大题都做了出来，否则，笔试很容易就挂了。面试这个公司的时候，一共面了两轮技术面，一轮HR面。最后也收获了Offer。但是，已经是十月中旬，手上没有三方协议了，很可惜，错过了。面经可以参考下这篇文章[2020秋招联发科小米等面经分享](#)

## 选择题

1. 已经获得除CPU以外的所有所需资源的进程处于（）状态

- A 就绪状态
- B 阻塞状态
- C 运行状态
- D 活动状态

A

进程的五状态模型：

运行态：该进程正在执行。

就绪态：进程已经做好了准备，只要有机会就开始执行。

阻塞态（等待态）：进程在某些事情发生前不能执行，等待阻塞进程的事件完成。

新建态：刚刚创建的进程，操作系统还没有把它加入到可执行进程组中，通常是进程控制块已经创建但是还没有加载到内存中的进程。

退出态：操作系统从可执行进程组中释放出的进程，或由于自身或某种原因停止运行。

2. 某二叉树的中序遍历序列为32145，后序遍历序列为32145，则前序遍历序列为

- A 54123
- B 32154
- C 32541
- D 54321

A

二叉树的中序遍历序列为 32145，后序遍历序列为32145，可知该树只有左子树结点，没有右子树结点，5 为根结点。

中序遍历序列与后序遍历序列相同，说明该树只有左子树没有右子树，因此该树有 5 层，从顶向下依次为54123。

具体分析过程也可以参考下[北京联发科嵌入式软件工程师笔试题目解析](#)

3. 若已知一个栈的入栈顺序是1,2,3...,n,其输出序列为P1,P2,P3,...,Pn,若P1是n,则Pi=（）？

- A i
- B n-i+1
- C 不确定
- D n-i

B

栈的排列遵循先进后（即后进先出）出的原则

因为P1是n，是出栈的第一个数字，说明在n之前进栈的数字都没有出栈。所以这个顺序是确定的。

还可以知道，最后出栈的一定是数字1，也就是Pn。代入这个式子，是正确的。

#### 4 (多选题) .下面协议中属于应用层协议的是 ()

- A ICMP、ARP
- B FTP、TELNET
- C HTTP、SNMP
- D SMTP、POP3

BCD

1、物理层：以太网、调制解调器、电力线通信(PLC)、SONET/SDH、G.709、光导纤维、同轴电缆、双绞线等。

2、数据链路层：Wi-Fi(IEEE 802.11)、WiMAX(IEEE 802.16)、ATM、DTM、令牌环、以太网、FDDI、帧中继、GPRS、EVDO、HSPA、HDLC、PPP、L2TP、PPTP、ISDN·STP、CSMA/CD等。

3、网络层协议：IP IPv4、IPv6、ICMP、ICMPv6·IGMP、IS-IS、IPsec、ARP、RARP、RIP等。

4、传输层协议：TCP、UDP、TLS、DCCP、SCTP、RSVP、OSPF等。

5、应用层协议：DHCP、DNS、FTP、Gopher、HTTP、IMAP4、IRC、NNTP、XMPP、POP3、SIP、SMTP、SNMP、SSH、TELNET、RPC、RTCP、RTP、RTSP、SDP、SOAP、GTP、STUN、NTP、SSDP、BGP等。

#### 5.下列程序段的时间复杂度是 ()

```

1 int fact(int n){
2     if(n<=1){
3         return 1;
4     }
5     return n*fact(n-1);
6 }
```

A O(log2n)

B O(nlog2n)

C O(n)

D O(n\*n)

C

当n<=1时执行return 1这一个语句，每次返回上一层都执行n\*fact(n-1)这一个语句，共执行n-1次。因此共执行基本语句n次，时间复杂度为O(n)

#### 6.下列排序算法中最好情况和最坏情况的时间复杂度相同的是？ ()

A 堆排序

B 快速排序

C 冒泡排序

D 归并排序

A D

堆排序在最好和最坏情况下的时间复杂度均为 $O(n\log n)$

快速排序最好和最坏情况下的时间复杂度分别为 $O(n\log n)$ 和 $O(n^2)$

冒泡排序排序在最好和最坏情况下的时间复杂度均为 $O(n\log n)$

归并排序在最好和最坏情况下的时间复杂度均为 $O(n\log n)$

**7.将两个各有n个元素的有序表归并成一个有序表，最少的比较次数是？（）**

A n

B 2n

C n-1

D 2n-1

A

归并排序是将两个或两个以上的有序子表合并成一个新的有序表。在归并排序中，核心步骤是将相邻的两个有序序列归并为一个有序序列。

题目中告诉我们，有两个各有n个元素的有序序列，要将这两个序列归并成一个有序序列，其方法是依次从小到大取每个序列中的元素进行比较，将较小的放进一个新的序列中，直到取完一个有序序列中的所有元素。再把另一个序列中剩下的元素放进新序列的后面即可。

最好的情况是一个有序序列中的最小元素大于另一个有序序列中的所有元素，这样只需要比较n次。

**8.将递归算法转换为非递归算法通常需要使用（）**

A 栈

B 队列

C 队列

D 广义表

A

**9.在MySQL中， productname regexp '[1-3]xiaomi'的含义是（）**

A productname 匹配“xiaomi重复1次或5次”的字符串

B productname 匹配“xiaomi字符串前一个字符为1或3”的字符串

C productname 匹配“xiaomi重复1到3次”的字符串

D productname 匹配“xiaomi字符串前一个字符为1到3”的字符串

D

**10.同个进程的不同线程以下不能被共享的是？（）**

A 全局变量

B 堆

C 文件句柄

## D 栈

B

线程共享的进程环境包括：

进程代码段、进程的公有资源（如全局变量，利用这些共享的数据，线程很容易的实现相互之间的通信）、进程打开的文件描述符、消息队列、信号的处理器、进程的当前目录、进程用户ID、进程组ID

线程独占资源：

线程ID、寄存器组的值、用户栈、内核栈（在一个进程的线程共享堆区（heap））、错误返回码、线程的信号屏蔽码、线程的优先级

## 专项选择题

### 1.下列Java函数的执行结果是什么 ()

```

1 static boolean foo(char c)
2 {
3     System.out.print(c);
4     return true;
5 }
6 public static void main(string[] args){
7     int i = 0;
8     for(foo('B');foo('A')&&(i<2);foo('C'))
9     {
10         i++;
11         foo('D');
12     }
13 }
```

A BADCBDCB

B BACDBACD

C BADCADCA

D 运行时抛出异常

C

- 1.其实foo('B')就是初始化条件，只会执行一次，所以第一个打印的肯定是B。
  - 2.因为i=0;循环条件是i<2（由此可知，循环i等于2的时候就会停止循环），所以0<2满足条件，接着会输出A。然后执行i++;i就变成1了，在输出D，在最后输出C。一次循环后的结果是：BADC。
  - 3.第二次循环的开始是foo('B');是初始条件，所以不会执行。直接从foo('A')开始，输出A，然后i为1，且小于2，此时循环体内再次执行i++；i的值为2了，再次输出D，最后输出C。第二次循环输出：ADC。
  - 4.然后循环再次执行for(foo('B');foo('A')&&(i<2);foo('C'))，直接输出A。i的值在第二轮循环后的值变成了2，2<2不成立，终止循环，输出A。
- 故输出结果是：BADCADCA。

### 2.下列有关软链接表述正确的是？ ()

A 不可以对不存在的文件创建软链接

B 不能对目录创建软链接

C 和普通文件没有什么不同，inode都指向同一个文件在硬盘中的区块

D 保存了其代表的文件的绝对路径是另一种文件。在硬盘上有独立的区块，访问时替代自身路径

C

A: 错。后半句说的是硬链接。硬链接是共同拥有同一个inode,不过是每个链接名不同，暂时理解成不同的文件名却指向同一文件。一个文件每加一个硬链接linkcount加1。

B: 错。可以对目录创建软链接。如下图所示。

```
[root@server ~]# mkdir -p ~/tmp/test/mydir
[root@server ~]# ls -a tmp/test/
. ..
mydir
[root@server ~]# ln -s ~/tmp/test/mydir ~/softlink_mydir
[root@server ~]# ll
total 4
lrwxrwxrwx 1 root root 20 Feb 21 11:53 softlink_mydir -> /root/tmp/test/mydir
drwxr-xr-x 3 root root 4096 Feb 21 11:52 tmp
[root@server ~]#
```

D:错。可以对不存在的文件创建软链接，如下图所示。

```
[root@server ~]# ln -s ~/tmp/test/myfile ~/softlinke myfile
[root@server ~]# ll
total 4
lrwxrwxrwx 1 root root 21 Feb 21 12:22 softlinke myfile -> /root/tmp/test/myfile
lrwxrwxrwx 1 root root 20 Feb 21 11:53 softlink_mydir -> /root/tmp/test/mydir
drwxr-xr-x 3 root root 4096 Feb 21 11:52 tmp
[root@server ~]# ll tmp/test/
total 4
drwxr-xr-x 2 root root 4096 Feb 21 11:52 mydir
[root@server ~]# echo "khkh">>tmp/test/myfile
[root@server ~]# ll
total 4
lrwxrwxrwx 1 root root 21 Feb 21 12:22 softlinke myfile -> /root/tmp/test/myfile
lrwxrwxrwx 1 root root 20 Feb 21 11:53 softlink_mydir -> /root/tmp/test/mydir
drwxr-xr-x 3 root root 4096 Feb 21 11:52 tmp
```

### 3.选项中那一行代码可以替换//add code here 而不产生编译错误 ()

```
1 public abstract class MyClass{
2     public int testInt = 5;
3     //addcode here
4     public void method(){
5     }
6 }
```

A public abstract void another Method(){}

B testInt = testInt \* 5

C public int method();

D public abstract void another Method(int a)

D

A:该项方法有abstract修饰，所以是抽象方法，由于抽象方法不能有方法体，所以A项错误

B:类体中只能定义变量和方法，不能有其他语句，所以B项错误

C:选项中的方法和类中的方法重复，所以会发生编译异常，所以C项错误

**4.有关Java静态初始化块说法不正确的是？（）**

- A 用户可以控制何时执行静态初始化块
- B 无法直接调用静态初始化块
- C 在创建第一个实例前，将自动调用静态初始化块来初始化
- D 静态初始化块没有访问修饰符和参数

A

JAVA的初始化顺序：

父类的静态成员初始化>父类的静态代码块>子类的静态成员初始化>子类的静态代码块>父类的代码块>父类的构造方法>子类的代码块>子类的构造方法

**5(多选题)以下分别对变量a给出定义，正确的有（）**

- A 一个有10个指针的数组，该指针指向同一个整型数:int \*a[10];
- B 一个指向10个整型数组的指针:int (\*a)[10];
- C 一个指向函数的指针，该函数有一个整型数并返回一个整型数:int \*a(int);
- D 一个有10个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数：int (\*a[10])(int);

ABD

C改为：int (\*a)(int)

指针数组：首先是一个数组，数组里面的元素都是指针；（存储指针的数组）

数组指针：首先是一个指针，指针指向一个一维数组；（指向数组的指针）

函数指针：一定要理解，回调中经常使用函数指针；

指针函数：就是一个普通函数，只是返回值是指针形式；

**6(多选题)下列叙述正确的是（）**

- A 指针可以为空，引用不能为空。
- B 不存在指向空值的引用，但是存在指向空值的指针
- C 引用必须被初始化，但是指针不必
- D 指针初始化后不能被改变，引用可以改变所指对象

ABC

D：引用初始化以后不能被改变，指针可以改变所指的对象

**7.下列关于C++容器描述错误的是？（）**

- A list类型支持双向顺序访问，在list中任何位置插入删除都很快
- B deque类型支持快速顺序访问，在头尾插入 / 删除速度很快
- C C++标准库map的底层实现为红黑树
- D vector类型在每次调用 pushback时都在栈上开辟新内存

B

deque：双端队列。支持快速随机访问。在头尾位置插入/删除速度很快。

**8 (多选题) C++中，下列数据类型的转换，哪个可能会发生信息丢失？**

- A int -> double
- B int -> long
- C long -> float
- D int -> char

CD

精度丢失只会发生在从大范围到小范围的转换

32位编译器：

|      |       |     |      |       |        |    |
|------|-------|-----|------|-------|--------|----|
| char | short | int | long | float | double | 指针 |
| 1    | 2     | 4   | 4    | 4     | 8      | 4  |

64位编译器：

|      |       |     |      |       |        |    |
|------|-------|-----|------|-------|--------|----|
| char | short | int | long | float | double | 指针 |
| 1    | 2     | 4   | 8    | 4     | 8      | 8  |

**9.在Java中，要使某个类能被同一个包中的其他类访问，但不能被这个包以外的类访问，可以（）**

- A 使用 private关键字
- B 让该类不使用任何关键字
- C 使用public关键字
- D 使用protected关键字

B

default和protected的区别是：

前者只要是外部包，就不允许访问。

后者只要是子类就允许访问，即使子类位于外部包。

总结：default拒绝一切包外访问；protected接受包外的子类访问

**10 (多选题) list和vector的区别有哪些（）**

A vector拥有一段连续的内存空间，因此支持随机存取，如果需要高效的随即存取，而不在乎插入和删除的效率，使用 vector.

B list拥有一段不连续的内存空间，因此支持随机存取，如果需要大量的插入和删除，而不关心随即存取，则应使用list

C 已知需要存储的元素时，使用list较好

D 如果需要任意位置插入元素，使用 vector较好

AB

C:已知需要存储的元素时, vector要好

D:如果需要任意位置插入元素， list要好

## 编程题1（字符串筛选）

给定一个字符串，需要去除所有之前曾经出现过的字符，只保留第一次出现的字符。

样例输入:hello,welcome to xiaomi

样例输出:heIo,wcmtxiao

### 思路

1. 首先需要定义两个数组，分别为“输入的字符串数组”old[ ]以及“输出的字符串数组” new[ ]；
2. 取old数组中的第一个字符去和new数组中的每一个字符串相比较是否相同，若出现相同，则取old数组的下一个字符再次与new中每一个字符相比较，若都不相同则存入new的数组中；
3. 最后输出数组new；

### 代码

```

1 #include <stdio.h>
2 void killsame(char *o, char *n)
3 {
4     int i=0, j, k=0;
5     int label;
6
7     while(o[i] != '\0')
8     {
9         label = 1;
10        for(j=0; j<i; j++)
11        {
12            if (o[i] == n[j])
13                label = 0; //一旦相同标志位置0
14        }
15        if(label) // 不相等
16            n[k++]=o[i];
17        i++;
18    }
19    n[k]='\0'; //结尾给\0
20    puts(n); //输出
21 }
22
23 int main(void)
24 {
25     printf("Please input a string you want:\n");
26     char old[126];
27     char new[126];
28     scanf("%s",old);
29     killsame(old, new); //去重
30     return 0;
31 }
```

## 编程题2（字符串有效判断）

给定一个只包括"(),{}[],'的字符串，判断字符串是否有效

有效字符串需满足：

1. 左括号必须使用相同类型的右括号闭合

## 2.左括号必须以正确的顺序闭合

注意空字符串可被认为是有效字符串

输入描述：待判断的字符串，多个字符串需换行输入

输出描述：每个字符串的判断结果，多个结果需换行输出

样例输入：

0 [] {}

([])

{[]}

样例输出：

true

false

true

## 思路

栈先入后出特点恰好与本题括号排序特点一致，所以用栈来实现。

当左括号出现的时候入栈，当右括号出现的出栈，如果匹配就继续，不匹配就错误。

当字符串遍历完成之后，栈内仍有字符串就错误。

用一个数组进行和一个记录栈顶值的int进行了栈的模拟，代码很简单，很好理解。

## 代码

```

1 #include <stdio.h>
2 bool isValid(char * s){
3
4     int len = strlen(s);
5     char stack[3500];
6     int top = -1;
7     int i = 0;
8     for( i=0;i<len;i++)
9     {
10         if((s[i] == '(') || (s[i] == '{') || (s[i] == '['))
11             stack[++top] = s[i];
12         else
13         {
14             if(top<0)//出现了右括号，但数组为空，即没有左括号与之匹配
15                 return false;
16             if((s[i] == ')'))
17             {
18                 if(stack[top]!='(') return false;
19                 else top--;
20             }
21
22             if((s[i] == '}'))
23             {
24                 if(stack[top]!='{' ) return false;
25                 else top--;
26             }
}

```

```

27         if((s[i] == '['))
28     {
29         if(stack[top]!='[') return false;
30         else top--;
31     }
32     }
33 }
34 if(top>=0) //数组内仍有左括号，没有右括号与之匹配
35     return false;\n
36 //这里一定要有返回值
37     return true;
38 }
39 int main(void)
40 {
41     printf("Please enter a bunch of brackets:\n");
42     char brackets[100];
43     scanf("%s",brackets);
44     printf(isValid(brackets));
45     return 0;
46 }
```

今天的题目就分享到这里，下一篇章，将会分享大厂的笔试题目解析。



## 北京联发科嵌入式软件工程师笔试题目解析

### 逻辑题

1. 参加新型冠状病毒疫苗开发研讨会的70名学者中，亚裔学者39人，博士33人，非亚裔学者中无博士学位的4人，根据以上陈述，参加此次研讨会的亚裔博士有几人？

- A 1
- B 2
- C 6
- D 7
- E 8
- C

亚裔学者39人，博士33人，非亚裔学者中无博士学位4人，这三者加起来是76人，但实际总人数只有70人。亚裔学者和博士两个概念之间为交叉关系，这两个概念和非亚裔学者中无博士学位者之间都是全异关系。这说明，既是亚裔学者又是博士即亚裔博士有6人。

也可以通过运用计算法来求解。设亚裔博士有x人，则可列方程： $39+33-x+4=70$ ，解这个方程，可得： $x=6$ 。

## 2. 某省妇女儿童占全省总人口的 $\frac{2}{3}$ 。如果妇女是指所有女性人口，儿童是指所有非成年人口，并且对任一年龄段，该省男女人口的数量持平，则上述断定能推出以下哪项结论？

- A 该省男性成年人口和儿童人口持平。
- B 该省男性成年人口大于儿童人口。
- C 该省男性成年人口小于儿童人口。
- D 该省女性成年人口和男性儿童人口持平
- E 该省男性成年人口和女性儿童人口持平。

A

由题干，可以给出以下表

|    | 女性            | 男性            |               |
|----|---------------|---------------|---------------|
| 成年 | $\frac{1}{3}$ | $\frac{1}{3}$ | $\frac{2}{3}$ |
| 儿童 | $\frac{1}{6}$ | $\frac{1}{6}$ | $\frac{1}{3}$ |
|    | $\frac{1}{2}$ | $\frac{1}{2}$ |               |

由任一年龄段，该省男女人口的数量持平，可得总人口男女持平。

由妇女儿童占全省总人口的三分之二，可得成年男性占三分之一。

由成年男性占三分之一，得男童占六分之一（因为男性占二分之一）。

由男童占六分之一，得女童占六分之一。因此，该省男性成年人口和儿童人口持平。

## 3. 某次讨论会共有25名与会者，已知(1)至少有7名青年教师是男性，(2)至少有8名男教师已过中年(3)至少有10名男青年是教师；如上述3句话两真一假，则关于与会人员可以得出以下哪项

- A 青年教师至少有10名
- B 男教师至多有15名
- C 男青年都是教师
- D 男青年至少有7名

D

[1]: 大于等于7名青男 [2]: 大于等于8名中男[3]: 大于等于10名青男。

[1]与[3]数量上有重叠部分，如果[3]为真，则[1]一定为真；

如果[1]为假，则[3]一定为假。此时就会两个为假，与题干条件的两真一假相矛盾，故[1]必真。而如果[1]为真，既青年男教师大于等于7人，那么男青年至少有7名。

4. 某中药配方有如下要求(1)如果有甲药材,那么没有乙药材(2)如果没有丙药材,那么必须有丁药材(3)人参和天麻不能都有(4)如果没有甲药材而有丙药材,则需要有人参。如果还有天麻,则关于该配方的断定哪项为真?D

- A 含有甲药材
- B 含有丙药材
- C 没有丙药材
- D 含有乙药材或不含丁药材

无正确答案

由“含有天麻”和（3）可以推出，不含有参；进而由（4），否定后件就能否定前件，可以推出，有甲药材或者没有丙药材。

如果有甲药材，由（1）可以推出，**无乙药材**；如果没有丙药材，由（2）可以推出，**有丁药材**；故**无乙药材或有丁药材**。

因此，选项中没有正确答案。

5. 某国拟在甲、乙、丙、丁、戊己6种农作物中进口几种,用于该国庞大的动物饲料产业,考虑到些农作物可能有违禁成分,以及它们之间存在的互补或可替代因素,该国对进口这些农作物有如下要求(1)它们当中不含违禁成分的都进口。2)如果甲或乙含有违禁成分,就进口丙和丁。3)如果戊含有违禁成分,那么己就不进口了;如果进口丙,就进口乙和己。(4)如果不进口己,就进口戊;如果进口戊,就不进口己。根据上述要求,以下哪项所列的农作物是该国可以进口的;

- A 甲、乙、丙
- B 乙、丙、丁
- C 甲、乙、戊
- D 甲、乙、己
- E 丙、戊、己

C

- A选项与（2）矛盾
- B选项与（2）矛盾
- C选项与（3）矛盾
- D选项与（4）矛盾

## 不定向选择

1. **int i =1;const int j =2;**以下说法不正确的是

- A const int \*p1 = &i;
- B const int \*p2 = &j;
- C int \*const p3 = &i;
- D int \*const p4 = &j;

D

int \*const p4 ,p4为指针常量， p4指向的内存位置不能改变，但是， p4所指内存存放的值是可以改变的。j表示常量，其数值不能被改变。

将j的地址赋给p4后， **p4可以执行其他操作**（如`*p4=4;`）， 将j的值改变，因此， `int *const p4 = &j;`是错误的。

## 2. 以下关于内存的说法正确的是

- A RAM是随机存储器，在断电时将丢失其存储内容，ROM是只读存储器，断电时不会丢失存储内容
- B 内存的数据带宽与内存的数据传输频率、内存数据总线位数以及内存大小有关
- C 用户进程通常情况只能访问用户空间的虚拟地址,不能访问内核空间虚拟地址
- D Linux中使用 buddy system算法可以管理页外内存碎片,使用slub算法可以管理页内内存碎片

ACD

B:内存的数据带宽的计算公式是：数据带宽=内存的数据传输频率×内存数据总线位数/8

## 3. 以下哪些事件会导致进程的创建

- A 系统初始化
- B fork系统调用
- C pthread\_create函数调用
- D 一个批处理作业的初始化

ABD

创建进程的多种方式但凡是硬件，都需要有操作系统去管理，只要有操作系统，就有进程的概念，就需要有创建进程的方式，一些操作系统只为一个应用程序设计，比如扫地机器人，一旦启动，所有的进程都已经存在。

而对于通用系统（跑很多应用程序），需要有系统运行过程中创建或撤销进程的能力，主要分为4中形式创建新的进程

- 1.系统初始化（查看进程 linux中用ps命令， windows中用任务管理器，前台进程负责与用户交互，后台运行的进程与用户无关，运行在后台并且只在需要时才唤醒的进程，称为守护进程，如电子邮件、web页面、新闻、打印）
- 2.一个进程在运行过程中开启了子进程（如 nginx开启多进程， os.fork等）
- 3.用户的交互式请求，而创建一个新进程（如用户用鼠标双击任意一款软件， qq， 微信等）
- 4.一个批处理作业的初始化（只在大型机的批处理系统中应用）

无论哪一种，新进程的创建都是由一个已经存在的进程执行了一个用于创建进程的系统调用而创建的。

## 4. 下列说法正确的有

- A 计算机体体系结构是一门研究计算机系统软件结构的学科。
- B 现代计算机处理器结构按照存储方式划分,可分为复杂指令集计算机和精简指令集计算机
- C RISC技术对比CISC最大的区别就是对CPI的精简

D 单指令流单数据流计算机的每个机器周期最多执行一条指令

CD

- A.计算机体系结构主要研究软件、硬件功能分配和对软件、硬件界面的确定
- B.现代计算机处理器结构按照**指令系统**方式划分,可分为复杂指令集计算机和精简指令集计算机

## 5. 32位系统中，该程序的输出为

```

1 //参数传递 退化为指针
2 void Func(char str_arg[100])
3 {
4     printf("%d\n", sizeof(str_arg));
5 }
6 int main()
7 {
8     char str[] = "Hello";
9     printf("%d\n", sizeof(str));
10    printf("%d\n", strlen(str));
11    char *p = str;
12    printf("%d\n", sizeof(p));
13    Func(str);
14    return 0;
15 }
```

A 5 5 4 4

B 6 5 4 4

C 6 5 6 4

D 5 5 5 100

B 6 5 4 4

使用函数**strlen()**求某个字符串的长度时是**不包括结尾标志符'\0'**的，但当你用**sizeof()**求某个字符串占用的内存空间时，**结尾字符'\0'是被包括在里面的**。

**strlen**用来计算字符串的长度（在C/C++中，字符串是以"\0"作为结束符的），它从内存的某个位置（可以是字符串开头，中间某个位置，甚至是某个不确定的内存区域）开始扫描直到碰到第一个字符串结束符\0为止，然后返回计数器值。

**sizeof**是C语言的关键字，它以**字节的形式**给出了其操作数的**存储大小**，操作数可以是一个表达式或括在括号内的类型名，操作数的存储大小由操作数的类型决定。

## 6. 有以下程序,求输出结果

```

1 #include<stdio.h>
2 int fun(int i)
3 {
4     int cnt = 0;
5     while(i)
6     {
7         cnt++;
8         i=i&(i-1);
9     }
10    return cnt;
11 }
```

```

12 int main()
13 {
14     printf("%d\n\r", fun(2021));
15     return 0;
16 }
17

```

8

&是按位与，对应位都为1时该位得1，否则得0。所以  $i \& (i - 1)$  的作用：将i的二进制表示中的最右边的1置为0。

在本题中即数出2021转换成二进制有几个1就会走几次循环(不断除2)。2021对应的二进制是：  
1010011111，一共8个1，故走8次。

扩展： $(n > 0 \&& ((n \& (n - 1)) == 0))$  是判断n是不是2的次幂

## 7. 若 int x = 5&6,那么x的值为 ()

- A 3
- B 4
- C 5
- D 6

```

1 B
2 5: 0101
3 6: 0110
4 x: 0100

```

## 8. 以下错误的表达式为

```

1 struct {
2     int a;
3     char b;
4 }Q, *p=&Q;

```

- A Q.a
- B (\*p).b
- C p->a
- D \*p.b

D

$*p = \&Q$ ，把Q的地址赋值给了指针p，对p解引用其实就是Q。

A 选项肯定是对的，结构体的正常访问方法。

B 选项  $(*p).b$  等价于  $Q.b$ 。

C  $p \rightarrow a$  p为指针访问结构体用 $\rightarrow$ 没问题。

D  $*p.b$  优先级问题， $.$ 的优先级高于 $*$ ，所以  $*p.b == * (p.b)$ ，p为指针，访问结构体成员要用 $\rightarrow$ 。

| 优先级 | 运算符                                            | 结合性  |
|-----|------------------------------------------------|------|
| 1   | 0 [] .                                         | 从左到右 |
| 2   | ! + (正) - (负) ~ ++ --                          | 从右向左 |
| 3   | * / %                                          | 从左向右 |
| 4   | + (加) - (减)                                    | 从左向右 |
| 5   | << >> >>>                                      | 从左向右 |
| 6   | < <= > >= instanceof                           | 从左向右 |
| 7   | == !=                                          | 从左向右 |
| 8   | & (按位与)                                        | 从左向右 |
| 9   | ^                                              | 从左向右 |
| 10  |                                                | 从左向右 |
| 11  | &&                                             | 从左向右 |
| 12  |                                                | 从左向右 |
| 13  | ? :                                            | 从右向左 |
| 14  | = += -= *= /= %=<br>= &=  = ^= ~= <<= >>= >>>= | 从右向左 |

扩展：结构体中.和->两种访问区别

定义结构体指针，访问成员时就用->

定义结构体变量，访问成员时就用.

```

1
2 struct A {
3     int a;
4     char b;
5 };
6 struct A q; //访问成员就用: q.a;
7 struct A *p; //访问成员就用: p->a;

```

## 9. 关于对象的this指针,以下叙述不正确的有

A 必须显示地在类中定义声明this数据成员才能使用this指针

B 一旦生成一个对象,该对象的this指针就指向该对象本身

C 一个类的所有对象的this指针的值都是相同的

D 不能通过对对象的this指针访问对象的数据成员和成员函数

A

this指针的特点：

(1) 每个当前对象都含有一个指向该对象的this指针。this指针只能在类的成员函数中使用，在全局函数、静态成员函数中都不能使用 this。

(2) this 指针是在成员函数的开始前构造，并在成员函数的结束后清除。

(3) this 指针会因编译器不同而有不同的存储位置，可能是寄存器或全局变量。

(4) this 是类的指针。

(5) 因为 this 指针只有在成员函数中才有定义，所以获得一个对象后，不能通过对对象使用 this 指针，所以也就无法知道一个对象的 this 指针的位置。不过，可以在成员函数中指定this 指针的位置。

(6) 普通的类函数（不论是静态成员函数，还是非静态成员函数）都不会创建一个函数表来保存函数指针，只有虚函数才会被放到函数表中。

## 10. 若某线性表中最常用的操作是在最后一个元素之后插入一个元素和删除第一个元素，则最节省运算时间的存储方式是

- A 单链表
- B 仅有头指针的单循环链表
- C 双链表
- D 仅有尾指针的单循环链表

D

单链表只能单向遍历，即只能由链表头向链表尾遍历。

单循环链表也只能单向遍历：链表头->链表尾->链表头；

对于A, B, C要想在尾端插入结点，需要遍历整个链表。

对于D，要插入结点，只要改变一下指针即可，要删除头结点，只要删除指针.next的元素即可。

如果只要知道尾指针p，则通过计算一次p->next就能获得头指针；插入和删除算法复杂度 $O(1)+O(1)$

而如果只知道头指针，则需要遍历整个链表来获得尾指针的位置；插入和删除算法复杂度 $O(1)+O(N)$

所以D仅有尾指针的单循环链表存储方式最节省运算时间

## 填空题

1. F和Q分别是指向单链表两个元素的指针，那么，F所指元素是Q所指元素后继的条件是 ( $Q->next == F$ )

2. 设有一个空栈，现有输入序列为1, 2, 3, 4, 5，经过 push, push, pop, push  
pop, push, pop, push后，输出序列是 (2,3,4)

2,3,4

push进栈，栈中是1

push进栈，栈中是1, 2

pop出栈，栈中是1， 输出2

push进栈，栈中是1, 3

pop出栈，栈中是1， 输出3

push进栈，栈中是1, 4

pop出栈，栈中是1， 输出4

push进栈，栈中是1, 5

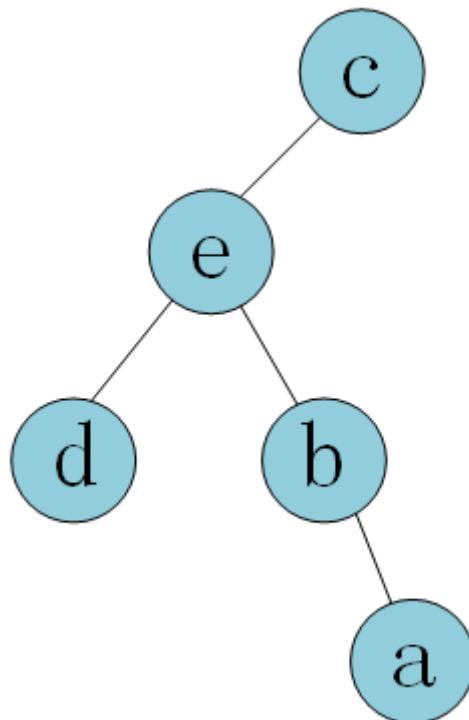
3.H是一个带有头结点的单链表,那么在第一个元素之后插入一个结点 (设P为要插入节点的指针) , 其操作是 ( p->next=H->next; ) ( H->next = p;)

4.已知二叉树后序遍历序列为dabec, 中序遍历序列为debac, 它的前序遍历序列为

cedba

- 1、由后续遍历可知c是根结点。
- 2、由中序遍历可知deba在c的左孩子树上。
- 3、由后序遍历知e是c的左孩子树的根结点。
- 4、由中序遍历可知d是e的左孩子, ba在e的右孩子树上。
- 5、由后序遍历, 可以得出b是e的右孩子, a是e的左孩子, 而第4步中确定了a不在e的左孩子数上, 因此, a只能在b上。
- 6、由中序遍历, 可知, a是b的右孩子。

所以前序序列为cedba, 具体如下图所示。



5.以下程序的输出结果为

```

1 #include <stdio.h>
2 int main()
3 {
4     int a;
5     a=(int)((double)(3/2)+0.5);
6     printf("a = %d",a);
7     return 0;
8 }
```

1

3和2是整形常量，所以 $3/2=1$ ；前面 (double)  $1 = 1.000000$ ； $1.000000 + 0.5 = 1.500000$ ；double转int会直接去掉小数部分。所以答案为1。

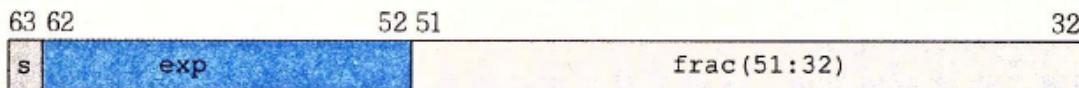
下面简单分析下double转int为什么会舍去小数部分

根据国际标准 IEEE 754，任意一个二进制浮点数 V 可以表示成下面的形式：

$$V = (-1)^S \times M \times 2^E$$

- (1)  $(-1)^s$  表示符号位, 当  $s=0$ ,  $V$  为正数; 当  $s=1$ ,  $V$  为负数。
  - (2)  $M$  表示有效数字, 大于等于 1, 小于 2, 但整数部分的 1 不变, 因此可以省略。 $M$  由 frac 编码。
  - (3)  $2^E$  表示指数位。 $E$  由 exp 编码。

双精度



对于 64 位的双精度数来说，从低位到高位，尾数 M 用 52 位来表示，阶码用 11 位来表示，而符号位用最高位 1 位来表示，0 表示正，1 表示负。

将1.5转换为双精度浮点数的过程如下：

1. 将十进制数1.5转换成二进制为1.1。
  2. 1.1用二进制的科学计数法表示为 $1.1 * 2^0$
  3. 按照上面浮点数的存储结构，得出符号位为：0，表示正数；阶码（指数）E为1023；小数部分M为1。
  4. 双精度的二进制位：

int类型为32位，double转换为int只能截取低32位为00000000000000000000000000000001。

所以最终的输出结果为1。

## 6.这段程序会存在什么问题

```
1 #include <stdio.h>
2 void my_alloc(char **p)
3 {
4     *p = (char *)malloc(100);
5 }
6 int main()
7 {
8     char *otr = NULL;
9     char *ptr = NULL;
10    my_alloc(&ptr);
11    strcpy(ptr, "hello world");
12    printf("%s\n", ptr);
13    return 0;
14 }
```

内存泄漏，分配给其他变量的内存就会减小。我们在删除一个指针之后，编译器只会释放该指针所指向的内存空间，而不会删除这个指针本身。此时`p`也就成为一个野指针

**扩展：**

### free()到底释放了什么？

free()释放的是指针指向的内存！注意，释放的是内存，不是指针。

**指针是一个变量，只有程序结束时才被销毁。**释放了内存空间后，原来指向这块空间的指针还是存在，只不过现在指针指向的内容是垃圾，是未定义的，所以说是垃圾。

因此，释放内存后把指针指向NULL，防止指针在后面不小心又被解引用了。当然，具体情况要具体分析以及具体解决。比如说，你定义了一个指针，在一个函数里申请了一块内存，然后通过函数返回传递给这个指针，那么也许释放这块内存这项工作就应该留给其他函数了。

### malloc()到底从哪里得到了内存空间？

从堆里面获得空间。也就是说函数返回的指针是指向堆里面的一块内存。操作系统中有一个记录空闲内存地址的链表。当操作系统收到程序的申请时，就会遍历该链表，然后就寻找第一个空间大于所申请空间的堆结点，然后就将该结点从空闲结点链表中删除，并将该结点的空间分配给程序。

## 7. 改正程序中的错误

下列给定程序将数组元素循环右移，数组大小和元素以及移动位数由键盘输入指定，例如数组 {1,2,3,4,5,6}，循环右移三位，得到结果为 {4,5,6,1,2,3}

请改正程序中的错误，使它能得出正确的结果。注意：不能更改程序的结构（共有四处错误）

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 void shift_func(int *array, int len, int k)
4 {
5     int i = 0, j = 0;
6     int temp = 0;
7     if (array == NULL)
8         return;
9     k %= len;
10    for (i = 0; i < k; i++)
11    {
12        temp = array[len - 1];
13        for (j = len; j > 0; j--)
14        {
15            array[j] = array[j - 1];
16        }
17        array[0] = temp;
18    }
19 }
20 int main()
21 {
22     int len = 0, k = 0, i = 0;
23     int *array = NULL;
24
25     scanf("%d%d", &len, &k);
26     array = (int*)malloc(len * sizeof(int));
27     if (array == NULL)
28         return -1;
29     printf("input the array\n");
30     for (i = 0; i < len; i++)
31         scanf("%d", array[i]);

```

```

32     shift_func(array, len, k);
33
34     printf("after shift the array is:\n");
35     for (i = 0; i < len; i++)
36         printf("%d", array[i]);
37     printf("\n");
38
39     return 0;
40 }

```

修改后的程序如下所示

```

1 void shift_func(int *array, int len, int k)
2 {
3     int i = 0, j = 0;
4     int temp = 0;
5     if (array == NULL)
6         return;
7     k %= len;
8     for (i = 0; i < k; i++)
9     {
10         temp = array[len - 1];
11         for (j = len; j > 0; j--)
12         {
13             array[j] = array[j - 1];
14         }
15         array[0] = temp;
16     }
17 }
18 int main()
19 {
20     int len = 0, k = 0, i = 0;
21     int *array = NULL;
22
23     scanf("%d%d", &len, &k);
24     array = (int*)malloc(len * sizeof(int)); //array = (int*)malloc(len *
25     sizeof(int));
26     if (array == NULL)
27         return -1;
28     printf("input the array\n");
29     for (i = 0; i < len; i++)
30         scanf("%d", &array[i]); //scanf("%d", &array[i]);
31
32     shift_func(array, len, k);
33
34     printf("after shift the array is:\n");
35     for (i = 0; i < len; i++)
36         printf("%d", array[i]);
37     printf("\n");
38
39     // free(array);
40     // array = NULL;
41     return 0;
42 }

```

## 编程题

输入年,月,日,计算这一天是该年的第几天。年份符合以下两种条件的任意一种即为闰年, 闰年里2月会有29天:

1. 年份是4的倍数但不是100的倍数。

2. 年份是400的倍数

输入描述:

输入表示日期的格式: Year, Month, Day包含三个整数:年(1<=Year<=3000), 月(1<= Month<=12), 日(1<=Day<=31)

**思路:** 比较容易想到的一种方法是查表。将一年中每个月份的天数放进数组中, 数组下标索引即代表月份。

这里要注意闰年的处理。为了方便, 我们定义两个数组, 分别对应闰年的天数和非闰年的天数。再定义一个变量flag来判断是否为闰年即可。具体代码如下所示。

```

1  /*
2   * @Description: 北京联发科嵌入式软件工程师笔试题目
3   * @Version:
4   * @Author: 嵌入式与Linux那些事
5   * @Date: 2021-3-15 22:24:12
6   * @LastEditors: 嵌入式与Linux那些事
7   * @LastEditTime: 2021-3-15 22:39:41
8   */
9 #include<stdio.h>
10 //两个数组, 分别存放闰年和非闰年每个月的天数, 对应 1~ 12月
11 int b1[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
12 int b2[12] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
13
14 /**
15  * @Description: 累加天数
16  * @Param: months, 月份。flag, 闰年判断标志位
17  * @Return: 天数
18  * @Author: 公众号【嵌入式与Linux那些事】
19  */
20 int add(int months, int flag)
21 {
22     int i, j = 0;
23     if(flag)
24     {
25         for(i=0; i<months; i++)
26         {
27             j+=b1[i];
28         }
29     }
30     else
31     {
32         for(i=0; i<months; i++)
33         {
34             j+=b2[i];
35         }
36     }
37     return j;
38 }
```

```

39
40 int main()
41 {
42     int years,months,days;
43     //设置标志位判断闰年
44     int flag = 0;
45     scanf("%d,%d,%d",&years,&months,&days);
46
47     if(years>=1 && years<=3000 && months>=1 && months<=12 && days>=1 &&
48     days<=31)
49     {
50         if((years%4==0&&years%100!=0))
51         {
52             flag = 1;
53         }
54         printf("result is %d",add(months,flag));
55     }
56     else
57     {
58         printf("invalid parameter");
59     }
60     return 0;
}

```

今天的题目就分享到这里，下一篇文章，将会分享小米的笔试题目和答案。



## 兆易创新嵌入式软件工程师笔试题目解析

哈喽，大家好。今天分享的是兆易创新的嵌入式软件开发工程师的笔试题目。这份题目中等难度，考察基础知识的偏多，最后的编程题只考了一个结构体数组的初始化。所以，在准备校招时，将重点还是要放在基础知识上。下面看下这份题目你可以答几分？

### 单选题

1. Linux中使用 mkdir命令创建新的目录时，在其父目录不存在时先创建父目录的选项是（）

- A -m
  - B -d
  - C -f
  - D -p
- mkdir [选项] [目录]

- m --mode=模式，建立目录的时候同时设置目录的权限。
- p --parents 若所建立的上层目录目前尚未建立，则会一并建立上层目录。
- v --verbose 每次创建新目录都显示信息。
- h --help 帮助信息。

## 2.下面代码创建了多少个进程（不包含main进程本身）（）

```

1 int main(int argc, char* argv[])
2 {
3     fork();
4     fork() && fork() || fork();
5     fork();
6 }
```

- A 19
- B 30
- C 24
- D 29

A

这道题主要考了两个知识点，一是逻辑运算符运行的特点；二是对fork的理解。

如果有一个这样的表达式：cond1 && cond2 | cond3 这句代码会怎样执行呢？

- 1、cond1为假，那就不判断cond2了，接着判断cond3。
- 2、cond1为真，这又要分为两种情况：

```

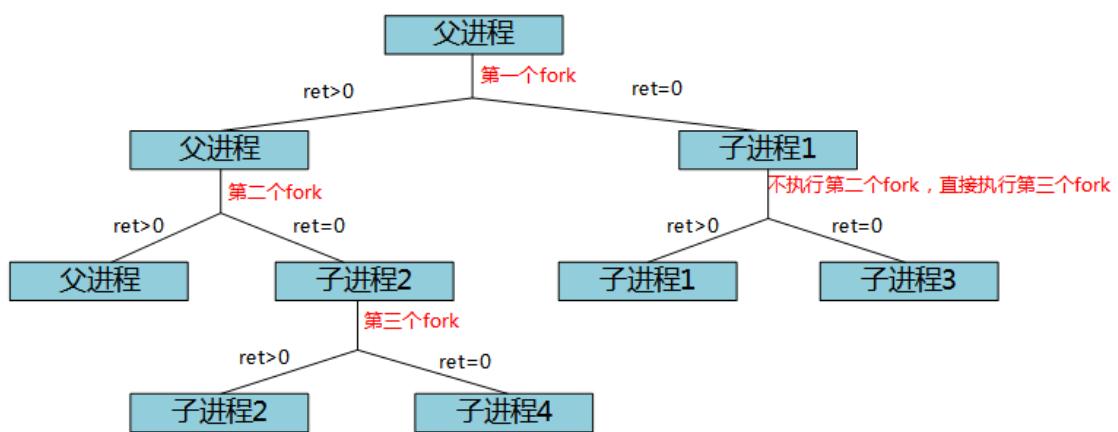
1 2.1 cond2为真，这就不需要判断cond3了。
2
3 2.2 cond2为假，那还得判断cond3。
```

fork调用的一个奇妙之处在于它仅仅被调用一次，却能够返回两次，它可能有三种不同的返回值：

- 1、在父进程中，fork返回新创建子进程的进程ID。
- 2、在子进程中，fork返回0。
- 3、如果出现错误，fork返回一个负值（题干中说明了不用考虑这种情况）。

在fork函数执行完毕后，如果创建新进程成功，则出现两个进程，一个是子进程，一个是父进程。在子进程中，fork函数返回0，在父进程中，fork返回新创建子进程的进程ID。我们可以通过fork返回的值来判断当前进程是子进程还是父进程。

有了上面的知识之后，下面我们来分析fork() && fork() || fork()会创建几个新进程。



很明显fork() && fork() || fork()创建了4个新进程。

总结：

第一行fork生成1个新进程。

第二行的三个fork生成 $4+4=8$ 个新进程。

第三行的fork会生成10个新进程(这是因为前面总共有10个进程,调用一次fork生成10个新进程)。

所以一共会生成 $1+8+10=19$ 个新进程。

### 3.如果下列公式成立： $3A*124=446C$ 。则采用的是（）进制

- A 11
- B 12
- C 14
- D 16

C

看个位。最后的结果446C个位为C，因此，A, B可以排除。

假设为14进制， $(A * 4) \% 14 = 12$ ，结果正好为C。因此，答案为14进制。

### 4.下面关于字符数组的初始化，那个是正确的？（）

- A `char b[2][3] = {"d", "e", "f"};`
- B `char b[2][3] = {"d", "e"};`
- C `char b[2][3] = {{"d", "e", "f"}, {"a", "b", "c"}};`
- D `char b[2] = {"d", "e"};`

B

通常情况下，二维数组的每一行分别使用一个字符串进行初始化。例如：

```
char c[3][8] = {"apple", "orange", "banana"};
```

等价于：

```
char c[3][8] = {"apple", "orange", "banana"};
```

A:应改为 `char b[3][2] = {"d", "e", "f"};`

C:应改为 `char b[2][3][2] = {{"d", "e", "f"}, {"a", "b", "c"}};`

D:应改为 `char b[2][2]={"d","e"};`

## 5.在32位系统中，下列类型占用8个字节的为（）

- A int
- B unsigned long long
- C char
- D short int

B

32位操作系统

int: 4字节

unsigned long long: 8字节

char : 1字节

short int: 2字节

注意和64位操作系统的区别：64位系统中，指针变量和long以及unsigned long 都占8个字节，其他的和32位系统一样

## 简答

### 1.

```
1 | int a[6] = {1,2,3,4,5,6};
2 | printf("%d\n",*((int*)(&a+1)-1));
```

那么打印结果是什么？

6。

将 `*((int*)(&a+1)-1)` 化简为：`*(p-1)` 和 `p=(int*)(&a+1)`

`&a` 是一个指向 `int(*)[6]` 的指针。由于 `&a` 是一个指针，那么在32位机器上，`sizeof(&a)=4`，但是 `&a+1` 的值取决于 `&a` 指向的类型，由于 `&a` 指向

`int(*)[6]`，所以 `&a+1 = &a + sizeof(int(*)[6])=&a+24`。`&a` 是数组的首地址，由此 `&a+1` 表示 `a` 向后移动了 6 个 `int` 从而指向 `a[6]`（越界），所以 `p` 指向的是 `a[6]`。

由于 `p` 是 `int *` 类型，那么 `p-1` 就指向 `a[5]`，所以 `*(p-1)=a[5]=6`。

### 2.请写出常量指针和指针常量的代码形式，并简述他们的区别

`int const *p1;` `const` 在前，定义为常量指针

`int *const p2;` `*` 在前，定义为指针常量

常量指针 `p1`：指向的地址可以变，但内容不可以重新赋值，内容的改变只能通过修改地址指向后变换。

指针常量 `p2`：指向的地址不可以重新赋值，但内容可以改变，必须初始化，地址跟随一生。

### 3.如何避免头文件被重复包含

1. 条件编译：

```

1 #ifndef _HEADERNAME_H
2 #define _HEADERNAME_H
3
4
5 ...//(头文件内容)
6
7
8 #endif

```

## 2. #pragma once

指定当前文件在构建时只被包含(或打开)一次，这样就可以减少构建的时间，因为加入#pragma once后，编译器在打开或读取第一个#include模块后，就不再打开或读取随后出现的相同#include模块。

## 4. 运行char name[] = "/dev/spdev"后，系统会分配几块内存，这些内存共占多少个字节？

11字节。

字符串最后以\0结尾，共占据11字节。

## 5. 如下代码，请设计宏定义STR(x)，将USART\_RATE转换成字符串并打印出来

```

1 #define USART_RATE 115200
2 #define STR(x)____?
3 printf("uart rate = %s\n", STR(USART_RATE));

```

#define STR(x) #x

#：会把参数转换为字符串

## 6. 已知结构体成员d的地址为p1，请获取成员变量b的地址。

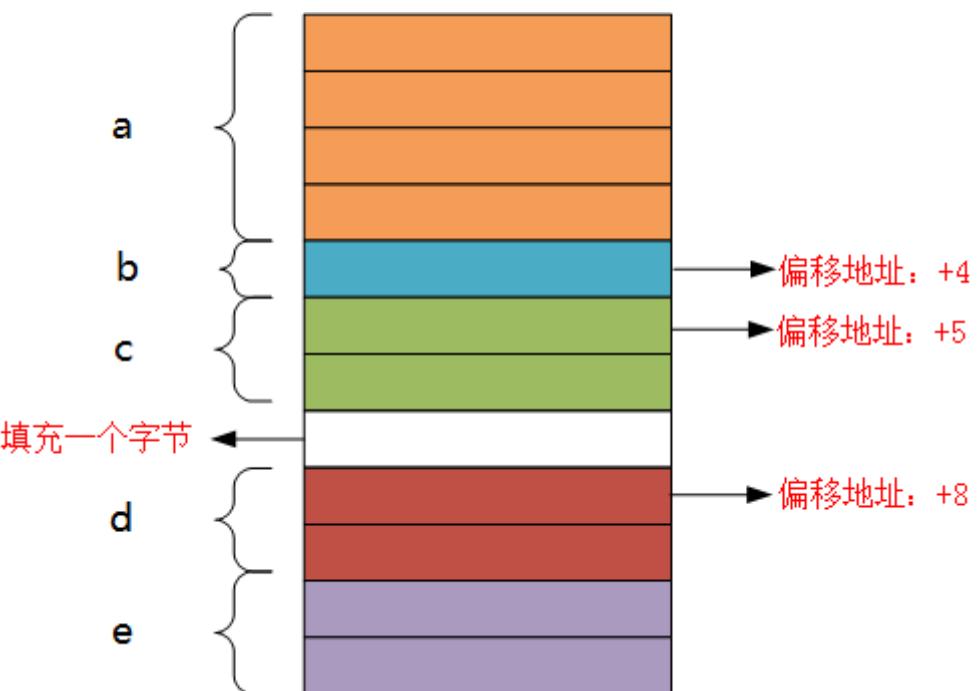
```

1 struct data{
2     int a;
3     char b;
4     short c;
5     short d;
6     int e;
7 };

```

p1 - 4 \* sizeof(p1)

结构体中成员变量在内存中存储的其实是偏移地址。也就是说结构体的首地址+成员变量的偏移地址 = 结构体成员变量的起始地址。具体如下图所示。



因此，将指针p1向上移动4个单位即可。即 `p1+4 * sizeof(p1)`

### 7.请写出下列代码的输出结果

```

1 int main(int argc, char *argv[])
2 {
3     char *buff[] = {"char", "int", "double"};
4     printf("%c\n", *(buff+1)[1]);
5     return 0;
6 }
```

d

buff是指针数组，一个3个元素的数组，数组里面是个字符串指针，这里执行buff+1时，则buff指向下一个数组元素，即int。

因此，`*(buff+1)[0]`指向int的地址，`*(buff+1)[1]`指向double的地址，而最后是输出一个字符。所以，输出d。

### 8.下面的代码输出什么？为什么？

```

1 void foo(void)
2 {
3     unsigned int a = 6;
4     int b = -20;
5     (a+b>6)?puts(">6"):puts("<=6");
6 }
```

>6

C中运算有规定，如果整型变量间进行数据运算，只要有一个变量是无符号的，结果就按无符号数据输出，因此 $a+b > a$

结果会输出 >6

## 编程题

表一：人员信息表： (info\_table)

| 序号 (num) | 姓名 (name)  | 性别 (gender) | 年龄 (age) |
|----------|------------|-------------|----------|
| 0        | 张三 (Bob)   | 男 (man)     | 60       |
| 1        | 李四 (Chris) | 男 (man)     | 30       |
| 2        | 小红 (Colin) | 女 (woman)   | 56       |

表二：人员职业表： (work\_table)

| 序号 (num) | 姓名 (name)  | 职业 (work)     | 等级 (level) |
|----------|------------|---------------|------------|
| 0        | 张三 (Bob)   | 司机 (driver)   | 9          |
| 1        | 李四 (Chris) | 厨师 (chief)    | 3          |
| 2        | 小红 (Colin) | 幼师 (teachers) | 6          |

表三：技能成绩表： (grade\_table)

| 序号 (num) | 姓名 (name)  | 技能 (skill)    | 成绩 (grade) |
|----------|------------|---------------|------------|
| 0        | 张三 (Bob)   | 开车 (drive)    | 50         |
| 1        | 李四 (Chris) | 烹饪 (cook)     | 64         |
| 2        | 小红 (Colin) | 教学 (teaching) | 55         |

如上所示有人员-职业-成绩的三个关系表

请尝试

1. 使用结构体表示三个表格。

2. 设计一个函数来依次录入人员信息

如add\_personnel(info\_table,work\_table,grade\_table);

```

1  /*
2  * @Description: 兆易创新编程题
3  * @Version:
4  * @Author: 公众号【嵌入式与Linux那些事】
5  * @Date: 2021-04-03 21:46:16
6  * @LastEditors: 公众号【嵌入式与Linux那些事】
7  * @LastEditTime: 2021-04-03 22:03:38
8  */
9 #include <stdio.h>
10 //人员信息表
11 typedef struct info_table{
12     char name[6];
13     char gender[5];
14     int age;
15 }INFO;

```

```

16 //人员职业表
17 typedef struct work_table{
18     char name[6];
19     char work[10];
20     int level;
21 }WORK;
22 //技能成绩表
23 typedef struct grade_table{
24     char name[6];
25     char skill[10];
26     int grade;
27 }GRADE;
28 /**
29 * @Description: 依次录入人员信息
30 * @param {INFO} *pinfo
31 * @param {WORK} *pwork
32 * @param {GRADE} *pgrade
33 * @Return: 无
34 * @Author: 公众号【嵌入式与Linux那些事】
35 */
36 void add_personnel(INFO *pinfo,WORK *pwork,GRADE *pgrade){
37     int i;
38     for(i = 0;i < 3;i++){
39         printf("请依次输入第%d个人的信息: 姓名, 性别, 年龄\n",i+1);
40         scanf("%s%s%d",pinfo[i].name,pinfo[i].gender,&pinfo[i].age);
41         printf("%s,%s,%d\n",pinfo[i].name,pinfo[i].gender,pinfo[i].age);
42     }
43     for(i = 0;i < 3;i++){
44         printf("请依次输入第%d个人的信息: 姓名, 职业, 等级\n",i+1);
45         scanf("%s%s%d",pwork[i].name,pwork[i].work,&pwork[i].level);
46         printf("%s,%s,%d\n",pwork[i].name,pwork[i].work,pwork[i].level);
47     }
48     for(i = 0;i < 3;i++){
49         printf("请依次输入第%d个人的信息: 姓名, 技能, 成绩\n",i+1);
50         scanf("%s%s%d",pgrade[i].name,pgrade[i].skill,&pgrade[i].grade);
51         printf("%s,%s,%d\n",pgrade[i].name,pgrade[i].skill,pgrade[i].grade);
52     }
53 }
54 int main()
55 {
56     INFO info_array[3];
57     WORK work_array[3];
58     GRADE grade_array[3];
59     add_personnel(info_array,work_array,grade_array);
60     return 0;
61 }
```

今天的题目就分享到这里，关于题目，有任何疑问都可以私信我。下一篇文章，将会分享大厂的笔试题目解析。



微信搜一搜

搜索框：嵌入式与Linux那些事

## 联系作者

### 微信

扫描下方二维码加我微信。微信号:LinuxDriverDev。



### 公众号

扫描下方二维码，关注我的公众号。

公众号有近**2000G**的嵌入式学习视频和**10G**的电子书，均可以免费获取！



## 赞赏

创作不易，如果觉得这些资料对你有帮助，那就可以赞赏给我，金额不重要，**我想看到你的头像出现在我列表里。**

最后，祝愿各位老板，钱多多，offer多多，跳槽薪资double！





支付宝

## 文档更新记录

建议大家加入资料群中，文档每一次更新我都会通知大家的。

| 更改人 | 日期        | 文档更新记录                         |
|-----|-----------|--------------------------------|
| 仲一  | 2021/6/3  | 整合所有笔试面试内容，修正部分错误              |
| 仲一  | 2022/8/20 | 修正部分错误，更新读者面经，新增谈薪话术，HR面试常见问题。 |