

Linux 应用编程- 学习总结备查

记 Linux 应用 相关的基本概念和流程，便于 备查 / 复习。p.s Linux 系统性的入门与学习的 路线和途径
推荐去看 [【主线剧情 0.0】Linux 学习资源大综合 - 知乎\(zhihu.com\)](#) ([CSDN 链接](#)) 里面的 [!学习来源!](#)
、[书籍推荐](#) 和 [Linux](#) 应用编程内容补充 这几节！。

注：[在 Github 上的原版文章日后可能会更新](#)，但这里不会跟进。[文章的 Gitee 仓库地址](#)，[Gitee 访问更流畅](#)。

地基

① Linux 开发 基本工具的使用

代码编辑：Vim、gedit（ubuntu 下）。编译工具：gcc，make，cmake（生成 makefile），gdb。项目管理：git。文本编辑：Windows 端的如 source insight、Notepad、VsCode 等；Linux 端的如 gedit（Ubuntu 端）、VsCode 等。

引 [主线剧情02-ARM-Linux基础学习记录 Real-Staok的博客-CSDN博客](#) 里面的 [Linux 下的开发](#) 一节的内容。

② 编译 / 交叉编译 工具的获取

引 [主线剧情02-ARM-Linux基础学习记录 Real-Staok的博客-CSDN博客](#) 里面的 [获取交叉编译工具链](#) 一节的内容。其中 举出三个方法：用开发板厂家提供的 SDK 里的工具链、ARM 官网下载 合适的工具链、使用 Linaro GCC 编译器。

另外，在构建 rootfs 时候使用 buildroot 或 yocto 时候可以构建 交叉编译器。

③ Linux 系统调用（各层调用关系）

- [Linux系统相关的基础问题（空间、内存、库、链接、环境变量）~青萍之末~的博客-CSDN博客](#)。
- [Linux系统调用~青萍之末~的博客-CSDN博客linux系统调用。【第4篇】嵌入式Linux应用开发基础知识哔哩哔哩bilibili](#)，其中 [4-2_文件IO _系统接口](#) 这一个节讲解 系统调用怎么进入内核。
- [Linux内核的五大模块~青萍之末~的博客-CSDN博客linux内核模块](#)，进程调度，进程间通信模块（包括管道、命名管道、消息队列、信号量和共享内存等），内存管理模块，文件系统模块，网络接口模块。
- etc.

④ Linux 一般开发流程

引 [主线剧情02-ARM-Linux基础学习记录 Real-Staok的博客-CSDN博客](#) 里面的 [Linux 一般开发步骤](#) 一节的内容：

Bootloader、Linux 内核、根文件系统、APP 等等软件，需要在 Ubuntu 中编译；但是阅读、修改这些源码时，在 Windows 下会比较方便。

所以日常工作日常开发流程如下：

PC 端，使用 source insight 编、改源码 → 传 → Ubuntu 端，对修改好的源码进行编译、制作 → 下载 → 嵌入式板端，在 Linux 板子上运行、测试。

分步来说就是：

在 Windows 上阅读、研究、修改，修改后，上传（推荐 FileZilla）到 Ubuntu；

在 Ubuntu 上编译、制作（推荐使用 MobaXterm 通过 SSH 远程登陆 Ubuntu）；

把制作好的可执行程序下载到开发板上运行、测试。

u-boot、Linux内核，在 Windows 和 Ubuntu 各存一份。根文件系统使用 buildroot（或 Busybox 或 Yocto）制作，它无需放在 Windows 上。

⑤ 主机端与嵌入式 Linux 板端 相互传文件

引|[主线剧情](#)[02-ARM-Linux基础学习记录](#) [Real-Staok的博客-CSDN博客](#) 里面的 PC 与 嵌入式板 传输文件的方式汇总 一节的内容。这一块内容会只在 [Github](#)/[Gitee](#) 仓库内更新:

- 网络传输: ETH / WiFi。通过 SSH、NFS、TFTP 等。
- USB 传输: ~~U盘拷贝~~, 芯片官方 配套的 USB 传输 / 烧写 工具。
- 串口传输: rz / sz 命令。

⑥ 期待美好发生

这儿的能力是靠 多做项目、多积累经验, 量变出质变。

开发

参考源

通过手册的 查看 API 详细描述参考

引 [【规范】万字集大成的C编写规范 Real-Staok的博客-CSDN博客](#) 中的 `7 C 标准库的使用` 一节内容，即 各种C标准库详解和用例，其中包括了 C/C++ 的常用标准库的 API 使用的详细描述，还包括 Linux_C ([glibc man cn](#)) 、 POSIX-C 等的 API 的完整描述文档（包括中文），便于速查，其仓库在 [额外文档/各种C标准库详解和用例·瞰百/coding-style-and-more - 码云 - 开源中国\(gitee.com\)](#) ([Gituhb 仓库地址](#)) 。

其中包括文件如：C语言标准函数库速查.pdf、C语言函数大全语法着色版.pdf、Linux函数大全.chm、POSIX-C函数速查.chm 等等。

通过命令行的 查看命令、API 描述参考

help、man 和 info 命令。

- help 只能用于查看某个命令的用法。
- man 命令既可以查看命令的用法，还可以查看函数的详细介绍等等。

(引自 百问网)

比如想查看 open() 函数的用法时，可以直接执行 `man open`，发现这不是想要内容时再执行 `man 2 open`。

在 man 命令中可以及时按“h”查看帮助信息了解快捷键。常用的快捷键是：

- `f` 往前翻一页。
 - `b` 往后翻一页。
 - `/<patten>` 往前搜 `patten` 字符。
 - `?<patten>` 往后搜。
- info 命令会显示最全面的信息，一般 `man` 命令就够了，故 `info` 命令介绍 暂略。

Linux 应用开发 / 学习 / 备查 参考源

- [【主线剧情 0.0】Linux 学习资源大综合 - 知乎\(zhihu.com\) \(\[CSDN 链接\]\(#\)\)](#) 里面的 `!学习来源!` 一节！
- [【主线剧情 0.0】Linux 学习资源大综合 - 知乎\(zhihu.com\) \(\[CSDN 链接\]\(#\)\)](#) 里面的 `书籍推荐` 一节。
- [【主线剧情 0.0】Linux 学习资源大综合 - 知乎\(zhihu.com\) \(\[CSDN 链接\]\(#\)\)](#) 里面的 `Linux 应用编程内容补充` 一节。

标准 API 富集

这里就是日常用到的 API 的积累和整理，最详细、全面的 API 参考 见上面 `通过手册的 API 详细描述参考` 一节。

日常用到的 标准 API 富集 单独放在了 `【Linux 通用应用开发】` ([仓库地址 Github](#)、[Gitee](#)) 文件夹里面，包括：

- 文件 IO、字符流收发 和 字符串处理相关的 API 收集积累。
- 一些 Shell 命令 API 收集积累。

- 输入设备框架，读取驱动程序按照 input event 框架上传的信息。
 - 文件 IO 读写的基本四种机制（阻塞、非阻塞、poll/select 和 异步通知（通过信号机制））。
 - Socket TCP / UDP 编程。
 - 进程 和 线程 的设计和编程。包括 进程和线程的分配策略、进程和线程的 API 用法 等：
 - 进程 相关 API (fork()、exec、wait() 等等) 。
 - 进程间通讯（管道、命名管道、信号、信号量、消息队列、共享内存、内存映射 和 套接字）。
 - pthread 线程编程库 相关 POSIX API (pthread_create()、(后面省略 “pthread_”) exit / cancel 和 join / tryjoin_np、self()、attr_setxxx() / attr_getxxx()、) 。
 - 线程间通讯、同步的机制（锁机制（互斥锁、读写锁 和 自旋锁）、信号量机制、条件变量机制 和 信号（与 IPC 的信号使用上有区别））。
 - 串口收发编程、I2C编程。
- 实质上就是对各种设备驱动文件 进行 open/read/write/ioctl/close 来配置和读写，有一些标准接口有标准的信息结构，因此有专门的 库，通过这些库来间接的 配置 和 读写设备驱动文件，比如 tslib、i2ctool 等等。
- 等等其它，用时补充。比如 进程高级编程、内存管理、Linux 高并发编程 epoll 等等。

Linux 应用的基本编程思想

大的思想就是，主要对设备文件使用 read/write/ioctl 和 mmap，以下举例：

- uart (ttyxxx) /i2c/spi设备文件，先用ioctl读出信息和写入配置，然后用read/write来接收和发送（涉及到四种机制：阻塞、非阻塞、poll/select、异步通知）。
- 屏幕，名为fb的设备，通常用帧缓冲，mmap之后，对一块内存用指针直接进行读写，就是读屏和写屏。
- 对于摄像头/图像/视频设备，名为videoxxx的设备，read就是读一帧图像/截图。
- 音频设备，同样是read来录音，write就是播放（如果支持的话）。

对于不同的设备，read 和 write 的使用比较符合直觉，清楚这个编程模式后面看程序，思路就比较清晰。

再有就是各种高级编程，包括进程/线程以及其间的通讯/同步（需要大量实验和经验），网络编程，内存管理，高并发 epoll 等等。