Linux 桌面应用

王欣

Linux 桌面应用

王欣

draft r0.1, 2011-04-10

身体的健康和灵魂的平静是幸福的极致。

—伊壁鸠鲁

目录

前言

计算机作为信息化时代的重要工具，渗透到了生活的方方面面，甚至它已经超越一般意义上“工具”的概念，不再单单为我们所用，更是在悄然影响并引导着我们的生活方式。然而对于它，我们只是和对待其它普通工具一样，被动地接受工具的设计者和制造者所提供的功能和使用方式，没有意识到它对生活的种种影响，也没有主动地思考我们的生活需要什么。我们不再是工具的支配者，反而是被工具所支配。这是 Douglas Rushkoff 在 Program or be Programmed 一书中给予我们的警示。

就像随着交通、通信等技术的发展，我们虽然享受到了便利，但面对因为这些技术而不断加快的生活节奏却无能为力。所以发明技术的初衷或许是希望让人类生活更幸福，但技术的发展却不是个人所能把控的，所以必须对计算机和信息技术有深入的了解，才不至于处于被动。或许说得夸张了点，但至少，生活在这个时代，不管你是不是从事于 IT 行业，都需要对计算机技术有相当的了解。在我们日常生活中，很难找到像计算机这样功能如此复杂而又如此普及的工具。所以，了解它熟悉它不仅仅是可以提高工作、生活的效率，更是可以让自己从工具中解放出来，不再受制于它。

然而，现在的应用软件大多被一层层地包装，已经完全无法让用户全然理解，更多的只能是让用户望而生畏。对于一个软件，我们往往只希望用最短的时间学会必需的功能，而对于其它的功能，对于它的基本原理，不可能再去关心。我们只能被迫全然信任软件的设计者和他们所制定的使用手册。

计算机技术真的就只能这么复杂？软件将被设计得越来越傻瓜化，这是否是信息技术发展的必然趋势？计算机技术与其它技术相比的一个重要特点是它的门槛更低，不是任何人都可以造出汽车，但大多数人却都是可以制作出软件的。所以是否有一种途径可以让更多的人参与到信息革命中来？

商业的系统永远只会以利益为导向，不可能顾及这些。而开源运动由于其本身的开放性，是否可以带来一丝的希望？ Linux 作为开源世界的重要成员，这些年有了长足的发展，然而在桌面应用市场却迟迟未见扩张，其中的原因值得我们思考。是历史的兼容性问题？还是用户使用体验问题？还是有其它什么原因？ Linux 桌面应用该往怎样的方向发展？极力

地去模仿 Windows 和 Mac 的形式和操作习惯是否可行？还是说能有一条更为理想的路？Linux 有其自身的诸多特点，是否可以加以利用？

虽然现有的技术还不足以让每一个用户设计自己的软件，但用户可以根据自己的需求对不同组件进行定制和整合。本文就是以上述问题为引线，从 Linux 的高可定制性以及它的 KISS[1] 设计哲学出发，尝试搭建一个日常使用的桌面环境。桌面应用只是计算机应用的一小方面，计算机技术对生活有着更深层次的影响，但我们可以从桌面应用出发，思考我们对于计算机的需求。

另外，由于 Linux 在很多功能上都有较多的程序可供选择，所以这里的桌面环境必然是非常个人化的，希望读者更多的是从中得到一些思路，而不只是实际的方法。

1. 内容组织

本文的大部分内容是自己这些年在对 Linux 桌面断断续续的使用中所积累的一些笔记和体会，一开始只是作为对自己知识的一个疏理。最终考虑用更为严肃的形式加以整理，这样可以和其他 Linux 使用者分享。如果有任何人能够从本书中找到一点两点对其有用的内容，那么我也就非常欣慰了。

对于个人笔记的整理，曾经尝试过很多的方式：博客、个人 WIKI 、思维导图等，但都不是非常理想，最终还是觉得以正式的文档甚至是书的形式来组织最能引起自己的认真对待。很多时候，形式是重要的，只有在正式的形式下，才能引导自己进行严肃的思考。

在文中会提出一些问题，大多也会给出我的理解。在思考这些问题时，我尽量不参考其它的资料和别人的观点，因为那些资料读者通过搜索引擎自然可以找到，这里更多的是我对自己的理解的直接表述，尽量不受其他观点的影响。但这样的一个后果是，可能很多想法会很不成熟，甚至有些是根本错误的，还望读者能够多多包涵。

2. 面向的读者

本文主要面向的是对 Linux 系统以及 Linux 桌面应用已经有所了解的读者，比如对于发行版的概念，桌面环境与窗口管理器的关系等应该都已经有所了解。本文不会再对这些常用概念做额外的介绍。

由于 Linux 的普及程序依然不够，如果没有对 Linux 有一定了解可能在阅读本文时会有一定困难。但由于自身能力和经验的限制，无法对各类概念做简明的定义和说明，所以不能够面向更为广泛的读者。不过我会尽量在一些概念首次出现时添加外部链接，这样可以方便读者查找。

[1] KISS 是 Keep It Simple and Stupid 的缩写，详细说明可以查看维基百科： http://en.wikipedia.org/wiki/KISS_principle 。

3. 环境说明

本文的环境是在 Windows 下的 VirtualBox 搭建的，不过文中大部分的内容与是否运行于虚拟机下是无关的，所以全文不会提及与 VirtualBox 相关的内容，只在附录中专门对它的设置进行的介绍。

另外，这里使用的 Linux 发行版是 CRUX ，虽然不同发行版在某些方面相差较大，但对于桌面应用影响不大，所以在正文中也不会提及与 CRUX 配置相关的内容。

4. 配置和代码

Linux 桌面环境下必然会涉及到较多的配置，另外文中也提及了我自己写的一些小工具，所有这些相关的配置和代码都存放在 https://github.com/dram/configs 。可以通过 git 工具进行同步，对于 git 的使用可以查看 配置文件管理 一节。

5. 更新

本文档会不定期进行更新，可以从 https://github.com/dram/docs 中找到相关更新的细节，请注意查看。

6. 意见反馈

由于经验有限，本文必然有很多不足，读者如有发现错误或者有待探讨的地方，还望指出，我的邮箱为 dram.wang@gmail.com 。

第 1 章 操作系统

操作系统，作为对底层硬件的封装和抽象，为上层应用程序提供访问硬件资源的接口，这是内核的基本功能，从这一层意义来说，如果将计算机作为桌面应用的工具，用户基本上不需要关心操作系统层面的东西。

就比如线程，它究竟是在用户层还是在内核实现，这对于用户来说没有区别，甚至于应用程序是利用多线程还是多进程机制实现，用户也可以不关心。就像 Google 在宣传 Chrome 浏览器时，强调的一点是它的每一个标签页都是独用进程，所以其中任何一个页面崩溃都不会影响其它标签页。但这更多的只是对 Firefox 不稳定性的一个攻击，而不能说是一个优点，并且理想的程序就不应该让“不合法的页面内容导致页面崩溃”这一现象出现。所以对用户来说，如果不出错，两者的体验是完全相同的。但用户需要知道“多任务”这一概念，“单任务”和“多任务”将促成全然不同的事务组织方式 [1]。

另外，由于操作系统功能相当明确，所以理想情况下在系统稳定后是无需再做大幅更新的， Windows XP 就是一个很好的例子，从 2001 年发布到现在将近 10 年，依然在大范围的使用。然而硬件和软件的更新换代是相互推动的，商业公司必须促使用户更新产品，以此才能获取利润。所以即使对于一个操作系统来说， Windows XP 已经够用， Microsoft 还是需要推出 Windows 7 ，对于 XP 的支持最终还是要停止。这样的后果是，在越来越多的应用软件只对 Windows 7 支持后，用户不得不对操作系统进行更新，而这一更新，不是因为操作系统本身的新功能，而只是为了能够支持新的应用软件。所以微软在更新操作系统时，并在是仅仅更新内核，更是要更新桌面体验，以些来吸引用户。

这就扩大了操作系统的概念，除了内核外，还包含软件包管理系统、桌面环境和大量的基础程序等。这也是我们平时说的操作系统的概念，比如 Windows 和 Mac 。所以用户还是需要关心一些与操作系统相关的东西的，这样可以了解计算机软件的运行基础。这样的操作系统概念在 Linux 中对应的就是发行版[2]的概念。桌面环境将在下一章再作介绍，这里主要就 Linux 发行版以及软件包管理机制作一些讨论。

[1]当然，至于单任务和多任务，到底人脑更适合哪一种方式，这又是另一层面的问题了，至少，“多任务”在大部分情况下确实提高了工作效率。

[2]关于发行版的定义，可以查看这里： http://en.wikipedia.org/wiki/Linux_distribution 。

1. 发行版

对于发行版，首先有必要考虑一个问题：为什么 Linux 有如此多的发行版？这或许正的 Linux 自由的体现，而同时也让我们看到了自由的代价。

用户的需求是多样的，这是必然。对于商业系统，会根据多种因素对用户进行归并、取舍，最终也只是维护少数几个版本，比如微软在 Windows 系统分为 Desktop, Server, Mobile 等版本。而在 Linux 中，用户除了有选择发行版的自由外，还有制作发行版的自由，而且制作的难度不是很高 [3]，这可能是导致发行版本泛滥的一个主要原因。

虽然，很多时候，矛盾是不可调和的，就比如上面提到的桌面用户的需求和服务器用户的需求在很多方面必然是不同的。但人往往太容易陷入细节，太容易追求局部的、片面的完美。为了一些细小的分歧而将一个项目分支为两个项目。虽然对于各自项目来说，都实现了各自的需求，都达到某种层度上的完美，但从全局看，固定的资源，原本可以在同一个项目上发力，而现在却要分为两拨，其中必然有大量的重复劳动，是很大的损失，从这点上说又是完全不能称之为完美的。

那么对于普通用户来说，对于这个发行版，他最需要的是什么？这里我们只就桌面应用的用户进行讨论。如果除去不同发行版在不同软件上的选择，那么发行版需要提供的最重要的功能就是方便的包管理机制以及对应用软件的支持，因为用户需要使用的不是操作系统，而是运行于操作系统之上的应用软件。

可以说这两点也是各发行版之间主要的不同。在包管理方面，有 dpkg, apt-get 组合，有 rpm, yum 组合，还有 pacman, AUR 组合，还有 emerge, portage 组合等等，它们在功能上很多是相似的，都是对程序文件的组织，对依赖关系的处理。但在设计理念和使用方式上，却又有着很大的不同。附录中对 Fedora 的 RPM 及 YUM 工具作了简单介绍，可以作为参考。

包管理是为支持应用软件而服务的，最终是要提供用户方便的安装和更新软件的方式。所以对于用户来说，选择发行版本，最为重要的是看它对软件的管理方式。当然，在选择时还需要考虑其它一些因素，比如发行版的用户定位，用户群的大小等。下面主要就软件管理机制方面进行讨论，以此选择合适的发行版。

2. 软件管理

软件管理，主要就是将程序文件组织在文件系统中，并对其进行统一管理，方便安装、卸载和升级。对于用户来说，只要需要使用的软件能够装上，能够使用，以后能够升级，对系统以及其它软件没有影响就可以了。但就是这么简单的需求，要实现起来也并不容易。特别是对于 Linux 来说，而很大一个原因还是因为自由。

[3]从某种意义上说，这可以用帕金森的鸡毛蒜皮定律或芝麻绿豆定律(Parkinson ' s Law of Triviality)来解释。越是简单的，大家都了解的事，不同的意见也就越多。这个时候，自由有时就并非好事，自由之下意见不和可能就要导致各自为政，从而浪费了大量的资源。

以 Windows 为例，由于 Windows 的所有底层接口都由微软一家设计和维护，所以只要微软想让接口保持稳定，可以比较容易实现，甚至在 XP 和 Win7 跨度那么大的两个版本，很多二进制程序依然可以同时在两个系统中安装运行。而在 Linux 中，像操作系统接口、 C 语言库、 X Window 、图形控件等等，不同组件由不同组织开发，各自发行版本，再经过发行版选择组装。有些发行版以稳定性为中心，使用较老版本，另一些软件追求更多、更新的功能，使用较新版本。这就导致一个发行版的二进制程序很可能无法在另一个版本中运行。 Android 就是因为各版本间接口的不稳定而遭至骂声一片。

这还只是发行版层面的兼容性问题。更为麻烦的是，不同的应用程序开发者，对于基础库选择的趋向也是不同的。比如软件 A 和 B 都依赖于基础程序库 L ， A 开发较早，依赖于 α 版本的 L 库， B 依赖于 β 版本。由于程序库从 α 到 β 版本有重大更新， L 库的开发者针对 β 版本放弃了向下兼容。那么，如果用户同时需要使用软件 A 和 B ，此时应该怎么处理呢?

Windows 的处理方法是，软件 A 和 B 都将程序库 L 包含在自身的目录中，这样就避免了冲突。在 Windows 下这是可行的，因为绝不部分底层库由微软控制，已经保证了它的兼容性，而其它的一些第三方库只占少数。

而在 Linux 中，底层库也存在兼容性问题，而对它的处理就不能像 Windows 那样让各应用程序自己维护了，而只能是让不同应用程序共享底层程序库。因为底层库被使用的频率太高，很多库的体积又很大，如果各自维护，势必程序体积的增大，这样就会增加磁盘占用，降低程序加载速度，增大内存的使用。所以 Linux 的处理很可能是放弃软件 A ，或者等 A 更新后再加入软件 B 的支持。对虽然满足了同时需要软件 A 和 B 的用户的需求，但却迫使用户对软件进行升级。对于这一点， BSD 相对来说好一点，它每一次发布的版本中包含内核以及应用层的一些基础库，这样在一定程度上保证的接口的稳定性。但这并没有包含开发桌面应用所需要的接口，所以对于桌面应用还是不够。

以上是基础库 API 及 ABI 接口层面导致的兼容性问题，从而让用户的应用程序版本选择上失去了一定的自由。而在各一方面，各个发行版自身对于底层接口稳定性的忽视也是导致这一问题的重要原因。

由于 Linux 上的应用程序一般为开源程序，可以非常方便的进行重新编译打包，所以二进制包一般由各发行版各自维护，而不是由软件设计者维护。再是在 Linux 中，大部分包管理工具都包含有网络下载更新功能，可以方便地对整个系统进行更新，这也致使大部分 Linux 发行版都会对软件包进行频繁更新，比如现在很多的都是六月一个新版本。这在一定程度上给用户造成了困扰。因为升级是需要花费人力成本的，而且还要考虑兼容性问题。并非所有用户都希望使用最新的软件。可能他只是需要对其中一两个应用软件更新，但由于这些应用软件的二进制包由发行版维护，而它们依赖于新的底层库，所以用户必须对整个系统进行更新才能安装使用这些软件的新版本。当然还是可以自行编译代码的，但这对于一个普通桌面用户来说，是要求太高了。给予用户选择是否对软件进行更新的自由，是对稳定性的一个重要保证，但在现有 Linux 发行版本中，很难找到比较理想的选择。用户只能寄希望于系统的更新不会带来什么麻烦，但这往往是不现实的。

当然，追求稳定的发行版是有的，比如 RHEL, Debian stable 等，但它们的稳定策略是针对整体系统而言，并没有对程序库和应用程序作区分，本质上它们是针对于服务器用户的。如果需要在这些系统中使用较新的应用程序也是复杂的。所以个人觉得现在 Linux 发行

版的一个重要问题是，对库程序和应用程序在升级处理上没有做很好的区分。对于一个桌面应用为主的系统来说，库应该是相对稳定的，而应用程序应该是包含多个可用版本，供用户选择，一样可以在一定程度上避免了不同应用程序之间的冲突。

比较接近这一思路的是 MEPIS 发行版，它底层基于 Debian stable ，这样保证了系统接口的相对稳定性，而桌面应用程序版本则比较新。但对于需要较新的应用程序的界定也是困难的，不同的用户会有不同的需求。在满足不同用户需求上， Gentoo 更为理想，它给予用户极大的自由，用户可以根据自身需要为不同软件选择不同版本。 Gentoo 将软件分为稳定(stable)和测试(testing)两个分支，与其它发行版不同的是，由于 Gentoo 的包管理机制是基于源码的，所以完全可以在 stable 分支的系统中针对个别软件使用 testing 分支的版本，这就允许用户在大部分包相对稳定的基础上，依然可以让少部分包保持较新，通过 stable 分支构建基本系统，而应用程序更为的是使用 testing 分支。而在二进制发行版本中，往往没有选择应用程序版本的自由。只是对于一个桌面系统来说， Gentoo 门槛太高了。

对于一个系统来说，一个用户的需求是基本确定的，但要满足 1 万， 1 百万用户的需求，就不是那么容易了。不同发行版通过事先设想用户的需求来选择软件以及版本，而 Gentoo 通过给予用户选择的自由，让用户自己来搭建满足自己需求的系统。

希望在 Linux 各主要的底层库和桌面应用程序趋于稳定之后，各发行版的区别将逐渐淡化，这样在软件管理方式上也可能出现统一。 Linux Standard Base(http://www.linuxbase.org)标准正是在尝试对 Linux 应用的底层库接口制定标准。而 PackageKit(http://www.packagekit.org/ 则是希望对各发行版的包管理机制在图形界面上进行统一。

接下来，进入实战阶段，主要说明 Linux 发行版的安装以及基础功能配置的问题。

3. 安装

对于一个操作系统的安装，并不需要每个用户都能够掌握，但也不能因此而忽略了它的用户体验，就比如 Gentoo ，虽然在它的安装过程中给予了用户极大的自由，同时也让用户对于 Linux 操作系统的组织有了一个较深的认识，但还是会因为没有较为简便的安装方式而将很多用户拒之门外。而相对来说，其它很多发行版，比如 Fedora 等，由于采用了图形化的安装界面，有较为丰富的提示和说明，减小了初次使用 Linux 系统的用户的障碍。

传统上，操作系统一般都是安装在硬盘中，但现在已经出现了越来越多的形式，比如 LiveCD ， U 盘等。对于像网络设备等功能相对固定的系统更多的采用 CF 卡等只读媒介来存放系统，而对于桌面应用来说，虽然网络操作系统已经出现，将所有数据存储在网络上已经成为可能[4]，但大多还是以硬盘方式存储。

虽然不同发行版在安装上有不同处理，但最终完成的任务是相似的：磁盘分区、设置时间时区和语言、拷贝文件、安装系统引导程序、创建用户、系统自启动服务选择、网络设置等。下面对其中一些操作作简要说明，方便用户了解 Linux 系统的一些基础功能。

[4]Google 的 Chrome OS 就是这样一种尝试，整个操作系统的唯一桌面应用软件就只有浏览器，所有其它应用都用网络服务代替。

3.1. 磁盘分区

针对不同的应用，会需要不同的磁盘分区方式，比如服务器可能就需要对日志存放进行单独分区，避免因为日志过大而影响服务的正常运行。而对于桌面应用来说，可能需要将数据文件存放进行单独分区，这样可以方便以后系统升级。

而分区的策略也会受到文件系统组织的影响。相对于 Windows 分盘形式来说， Linux 系统的整体树型结构更为自由，可以对任何一个结点进行单独分区，不同分区可以针对存放的内容采用不同的文件系统。当然，这些更多的是出于性能上的考虑，而对于桌面应用影响不大。

在命令行下，分区一般通过 fdisk 或 cfdisk 进行，现在也出现了一些图形化的分区工具，比如 GParted(http://gparted.sourceforge.net/ 等。

在分区之后就需要考虑对各分区选择不同的文件系统了。比如传统的 ext2, ext3 ，以及新近的 ext4, btrfs 等，对于这些文件系统的说明，可以查看这篇文章： http://linuxtweaks.wordpress.com/2010/04/23/which-linux-filesystem-to-choose/ 。

3.2. 创建用户

Linux 基于 UNIX 设计，所以从一开始对多用户就有较好的支持。不过对于桌面系统来说，多用户并非必需，但通过区分超级用户和普通用户，还是可以增强系统的稳定性的。所以在 Linux 桌面系统中，一般不建议使用 root 作为日常使用的用户。

在桌面应用中，用户概念会同时结合用户组概念在文件权限，资源访问权限、操作权限等方面有所体现。比如如果需要使用声卡，则应该将用户加入 `audio` 组，可以通过 `gpasswd -a username audio` 命令或者直接修改 `/etc/group` 文件进行添加。

通过 `useradd -m username` 命令可以创建普通用户， `passwd username` 可以修改该用户的密码。

另外，如果觉得每次在需要 root 权限时都要调用 `su` 并输密码比较繁锁，可以使用 `sudo` 命令。先调用 `visudo` ，在 sudo 配置文件中添加下面的内容，以后只要在需要管理员权限的命令前加上 `sudo` 就能以管理员权限运行该命令。 `sudo su -` 可以不必输入密码以 root 权限运行 SHELL 。

```
username    ALL=(ALL)       NOPASSWD: ALL
```

3.3. 系统服务

在 Linux 系统中，有一个 runlevel[5]概念，用于指定用户希望以什么形式运行系统，对于桌面应用来说，一般会选择 3 或 5 ，分别指多用户终端环境和多用户桌面环境。而实际

[5]关于 runlevel 的说明，可以查看这里： http://en.wikipedia.org/wiki/Runlevel

两者的区别不大，只是初始启动的服务有所不同。这些都在 init 系统中指定。对于 init 系统的维护，不同发行版有不同的工具， Feodra 中使用的是 `chkconfig` ， Gentoo 中使用的是 `eselect rc` ，但最终实现的动作是相似的。关于这一方面的介绍，可以查看 wikipedia 的说明： http://en.wikipedia.org/wiki/Init 。

3.4. 网络

各发行版本在网络启动脚本上的设置都有所不同，但最终调用的命令都是相似的。通过 `ifconfig` 或者 `ip link` 命令设置静态网卡地址，而对于 DHCP 的支持则通过 `dhcpcd` 命令实现。 DNS 的设置存放在 `/etc/resolv.conf` 文件中。 `/etc/hosts` 文件中存放静态 IP 与域名绑定。

3.5. 时间设置

对于时间的调整主要包括时区和时间的设置。时区的配置文件是 `/etc/localtime` ，从 `/usr/share/zoneinfo/` 中拷贝相应时区的文件到 `/etc/localtime` 即可完成对时区的修改。

设置系统时间可以通过 `date` 命令完成，格式为： `date MMDDhhmmCCYY` 。 M 为月， D 为日， h 为时， m 为分， CCYY 为年。当然也可以使用 ntp 进行同步。

通过 `date` 命令设置的其实是系统的时间，并没有同步到物理时钟，通过 `/sbin/hwclock --systohc` 进行同步，但各发行版一般都会在系统关机脚本中加入物理时钟自动同步功能。

4. 优化配置

以下再罗列一些对系统的配置和优化，供读者参考。

4.1. 系统启动

对于系统启动速度的优化，主要需要从两方面着手。一是内核定制，删去不必要的驱动，将必需的驱动直接整合到内核文件而不是以模块方式编译。再是调整系统启动服务，删去不必要的服务。

4.2. 虚拟磁盘

对于现在的计算机来说，磁盘 IO 依然是一个很大的瓶颈 [6]，所以对一些频繁读写而又不需要保存的文件，如果将它们转移到内存中，应该可以极大地提高系统性能。以下就是对 `/tmp` 目录以及 Firefox 的 cache 目录进行的处理。

`/tmp/` 目录

在 `/etc/fstab` 中添加 `tmpfs /tmp tmpfs size=300m 0 0`，可以将 `/tmp/` 目录挂载为 tmpfs 文件系统，也就是使用内存存储文件。其中 size 是可选的，表示最大限值。

Firefox Cache

挂载 Firefox 缓存到共享内存目录[7]。方法是在 `about:config` 配置页面添加 `browser.cache.disk.parent_directory` 参数，值设置为 `/dev/shm/firefox-cache`，再通过 `install -dm700 /dev/shm/firefox-cache` 新建该目录。

[6]希望固态硬盘(SSD)的普及可以对这一现状有所改观。

[7]这一优化对 Firefox 的响应速度会有较大提升，在 http://wiki.archlinux.org/index.php/Speed-up_Firefox_using_tmpfs 中有详细的介绍。

第 2 章 桌面环境

桌面环境是桌面应用中不可或缺的一部分。按功能划分，主要包括：窗口管理器(window manager)、任务栏(taskbar)、系统托盘(system tray)、桌面(desktop)等。

Linux 下可以直接使用 GNOME, KDE 或 XFCE 等系统直接得到桌面环境的所有功能。而另一种方式是分别对各个组件进行选择，拼装成满足自身需求的桌面环境，这是 KISS 设计哲学很好的体现。其实对于 GNOME, KDE 和 XFCE 来说，它们对于这些功能也是通过不同程序实现的，只是这些程序的界面和操作的一致性较高，耦合性较大。

这些不同功能的程序能够很好的协调工作，主要得益于 X Window 标准化的接口。比如 EWMH [1]标准规定了窗口管理器的一些接口，这样不同窗口管理器虽然可能有全然不同的窗口组织策略，但处理机制是相同的，这就允许其它程序独立于窗口管理器直接对窗口进行处理。其它标准还有 System Tray Protocol Specification, XEmbed Protocol Specification 等等，可以在 http://standards.freedesktop.org/查看更多的 Linux 下桌面应用相关标准。

下面主要介绍如何通过整合 Openbox, tint2, xchainkeys 等程序搭建一个轻量级的桌面环境。相关的配置文件和脚本先在这里列出，读者可以作为参考：

X Window 启动脚本
: https://github.com/dram/configs/blob/master/.xinitrc

Openbox 启动脚本
: https://github.com/dram/configs/blob/master/.config/openbox/autostart.sh

Openbox 样式
: https://github.com/dram/configs/tree/master/.themes

xchainkeys 快捷键配置
: https://github.com/dram/configs/blob/master/.config/xchainkeys/xchainkeys.conf

[1]EWMH 标准由 X Desktop Group 制定，详细信息可以查看： http://standards.freedesktop.org/wm-spec/wm-spec-latest.html 。

窗口控制辅助脚本
https://github.com/dram/configs/blob/master/bin/wm-assist.py

tint2 任务栏程序配置
https://github.com/dram/configs/blob/master/.config/tint2/tint2rc

fontconfig 配置
https://github.com/dram/dram-configs/blob/master/.fonts.conf

1. 窗口管理器

窗口管理器的主要作用是管理窗口的摆放，显示标题栏以及对窗口边框的修饰，还有对常用窗口操作的支持，如：最大化、最小化、窗口移动、改变窗口大小等。另外为方便键盘操作，窗口管理器一般还会提供快捷键设置的功能。

关于窗口的放置策略，有两种比较流行的方式，一种是层叠，另一种是平铺[2]。平铺可能对于大屏幕比较合适，这里还是选择传统的层叠式的窗口管理器。

另外还需要考虑窗口操作驱动方式的问题，可选的是计算机主要的两个输入接口：键盘和鼠标。两种方式各有利弊，鼠标操作非常直观，但双手更多的是停放在键盘上，这样会导致手在鼠标和键盘间频繁切换；而使用键盘控制窗口虽然比较快捷，但需要有一个适应的过程，需要在一段时间之后，才能将按键序列变成下意识的动作。这里主要是以介绍键盘操作为主。

对于用户来说，由于会对窗口进行大量的操作，所以窗口管理器的行为必须是直观的，同时为适应不同用户的需求，需要具备一定的可定制性。这里选择使用 Openbox ，它对于上述功能有比较好的支持，同时本身体积比较小巧，适合与其它程序整合使用。

Openbox 配置文件是 XML 格式，可以直接修改，也可以使用图形化配置工具 obconf 修改配置，文件为 `$HOME/.config/openbox/rc.xml` 。下面主要来介绍与窗口管理器相关的一些配置。

1.1. 虚拟桌面

关于虚拟桌面的概念，详细的可以查看： http://en.wikipedia.org/wiki/Virtual_desktop 。简单说来就是可以将开启的窗口分发到不同桌面从而更便于管理。

在 Openbox 中可以通过 `Ctrl+Alt+方向键` 或者 `Alt+鼠标滚轮` 进行虚拟桌面切换。移动窗口到桌面边缘可以将窗口移入另一个桌面， `Shift+Alt+方向键` 也能完成同样的功能。

[2]相对来说，层叠的放置策略比较常见，对于平铺窗口放罟策略的介绍，可以查看： http://en.wikipedia.org/wiki/Tiling_window_manager 。

1.2. 快捷键

上面已经提及，这里希望尽可能使用键盘完成常用的窗口操作以及其它一些桌面应用功能，虽然 Openbox 自带有快捷键设置功能，但并不支持 chained key bindings 功能[3]，所以这里使用 xchainkeys 程序完成主要功能的快捷键设置，而对于一些只能由 Openbox 提供的功能，则依然使用 Openbox 的快捷键设置。

关于 xchainkeys 的使用，可以参考这里的 配置 。在配置中可能需要用到各个键盘按键的键名，可以通过下面的命令查看：

```
xev | grep keycode
```

另外，如果需要对特定按键进行交换或设置，可以使用 xmodmap 命令。比如下面的命令将替换菜单键为 Win 键，其中 135 是菜单键的 keycode ，不同键盘可能不同，具体可以通过上面的 xev 命令查看。

```
xmodmap -e "keycode 135 = Super_R"
```

1.3. 窗口操作

虽然 Openbox 自身已经支持大量的窗口操作，并可以将这些动作设置为相应快捷键或鼠标动作。但其扩展性不好，不能支持一些更为复杂的窗口动作。所以这里基于 EWMH 标准编写了一个脚本程序对窗口进行控制，再利用 xchainkeys 捕捉键盘事件绑定到相应动作。通过外部脚本控制的另一个好处是，该脚本不受 Openbox 限制，也可以在其它窗口管理器中使用。完整的脚本可以查看： https://github.com/dram/configs/blob/master/bin/wm-assist.py 。

`wm-assist.py` 脚本基于 python-xlib 实现，参考了 wmctrl 程序，其中除了实现基本的窗口操作外，还实现了其它一些功能，会在后面再作介绍。该脚本是一 daemon 程序，通过 pipe 接口接受外部命令，在 xchainkeys 的 配置文件 中可以查看到具体的命令格式。以下对在 wm-assist.py 中与窗口操作相关的几个动作加以说明：

`MOVE`

相对于当前位置对窗口进行移动。

`RESIZE`

改变当前窗口大小。

`CENTER`

将当前窗口移至屏幕正中。

[3]chained key bindings 是指可以将多个键盘点击动作做为一个事作，将动作绑定到这一按键序列中，这样可以方便的快捷键进行分类设置，同时有效避免了因为首字母相同而引起的冲突。更为详细的说明可以查看 xchainkeys 的项目网站： http://code.google.com/p/xchainkeys/ 。

`CLOSE`
关闭当前窗口。

`MAXIMIZE`
最大化当前窗口。

`JOE`
提升指定窗口或启用程序，关于这个动作的详细说明将在“程序启动”一节再作介绍。

`ALL`
将浏览器、终端、编辑器依次排列，再次调用会将这些窗口恢复回原先的位置，主要是在需要同时查看多个窗口内容时使用。

另外，虽然这里以键盘操作为主，但对于一些操作来说，鼠标可能更为方便，下面再介绍几个键盘与鼠标相结合的快捷操作方式:

`Alt + 左键拖动`
移动窗口。

`Alt + 右键拖动`
更改窗口大小。

`Alt + 滚轮`
切换虚拟桌面，这个前面已经提到。

1.4. 样式

大多数窗口管理器在处理窗口的修饰时，为了方便用户自定义，都采用样式的形式处理，在 http://box-look.org 中可以找到大量的 Openbox 的样式。 Openbox 样式是一个文本描述文件，可以很方便地进行调整。这里使用的样式是基于 Moka 和 1977 Openbox 这两个样式调整的。

Openbox 样式只是对窗口修饰的调整，程序自身的样式并没有改变。如果是 GTK 程序，可以通过 `gtk-chtheme` 程序进行调整。 `gtk2-engines` 软件包中包含很多 GTK 样式，更多样式可以在 http://gnome-look.org 中找到。

2. 桌面

在 Windows 中，桌面往往会放置常用程序的快捷方式以及最近需要访问的文件，而 Linux 的桌面环境常常会看到只显示一张背景图片，或者显示一些系统当前的资源占用状态。而且，多数窗口管理器都不带有桌面背景设置功能，需要借由其它程序实现。

Openbox 也不例外，这里通过 `feh`命令对桌面背景进行设置。利用 feh 的 `--bg-tile` , `--bg-center` , `--bg-scale` 或 `--bg-seamless` 参数设置背景之后， feh 会生成 `$HOME/.fehbg` 文件，下次如果需要设置同样的背景，只需要运行 `source $HOME/.fehbg`

即可，随后将看到，可以把这一命令放入 Openbox 自启动程序脚本中，从而在每次启动 X Window 时对桌面背景进行自动设置。

这里没有在桌面添加常用程序的快捷启动，而是通过其它形式加以代替，将在下面介绍。

3. 程序启动

3.1. 手工启动

对于安装在系统中的程序，需要有一个入口可以让用户启动这些程序，在 Windows 中一般是使用开始菜单或者桌面的快捷方式。菜单是比较详细的罗列，而桌面快捷方式则是便于启动常用的程序。

下面简要说明这里在处理程序启动问题时使用的几种方式。

首先是采用快捷键的方式，上面已经提到了 xchainkeys 工具，可以通过它来设置快捷键来直接启动相应程序。另外，在 `wm-assist.py` 还实现了 `jump-or-exec` 功能[4]，该功能可以在指定程序没有启动时启动该程序，而在程序已经被开启时将该程序窗口提到最上层。

再一种方式是使用“运行”窗口，类似于 Windows 中按下 Win+R 键所弹出的窗口，在 Linux 中对应的程序是 gmrun ，它可以利用 Tab 键对命令进行补全。另外也可以使用跨平台工具 Launchy ，可以获得同样的功能。

最后再来介绍菜单方式，与配置文件一样， Openbox 的菜单也是 XML 格式的，可以直接编辑，也可以通过 menumaker 程序自动生成， menumaker 会自动检测当前系统所安装的程序，比如它会对 `/usr/share/applications` 目录下的 `.desktop`[5]文件进行分析。

3.2. 自启动

对于 Openbox 来说，有两处地方可以设置在启动 X Window 时需要自启动的程序。分别是自身的 `$HOME/.config/openbox/autostart.sh` 和 `/etc/xdg/openbox/autostart.sh` 这两个脚本，以及 XDG 标准规定的 `~/.config/autostart/` 和 `/etc/xdg/autostart/` 目录。

HOME 目录中的 `autostart.sh` 优先级高于 `/etc/xdg/openbox/autostart.sh` 。`$HOME/.config/autostart` 目录下的文件优先级高于 `/etc/xdg/autostart/` 下的同名文件 。

[4]jump-or-exec 功能最初源自于 Sawfish 的 jump-or-exec 插件，相对于传统的 Alt-Tab 来说，这个功能更加方便，更为详细的介绍可以查看 pluskid 关于它的说明： http://lifegoo.pluskid.org/wiki/JumpOrExec.html 。

[5]desktop 文件内容格式遵循 Desktop Entry Specification 标准，该标准的内容可以查看： http://standards.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html 。

XDG 标准的规定并不是强制性的，可以由窗口管理器决定是否需要运行该目录下的程序。在 Openbox 中是在 `autostart.sh` 脚本中进行的处理。

更为详细的说明可以查看 http://openbox.org/wiki/Help:Autostart 还有 http://specifications.freedesktop.org/autostart-spec/autostart-spec-latest.html 。

4. 任务栏

与 Windows 一样， Linux 中的任务栏程序一般会实现两个功能：提供窗口列表的显示和切换，以及 systray 显示功能。

Linux 中可供选择的任务栏程序非常多，这里选择使用 tint2 ，它包含了上述两项功能，相关设置可以参考这里的 配置文件 。另外，如果需要单独的 systray 功能，可以使用 stalonetray 程序。

5. 其它设置

5.1. 复制粘贴

在 X11 窗口系统中，一般有两种复制粘贴方式，一种与 Windows 相似，通过各个程序菜单中的复制粘贴项或是使用与其相关连的快捷键。而另一种方式是通过鼠标选中文本进行复制，再利用鼠标中键或者 SHIFT+INSERT 进行粘贴。

第二种方式 SELECTION 方式，需要注意的是，鼠标的点击动作会清空 SELECTION ，所以如果希望将一个窗口的内容复制到另一窗口，必须保证在选中文本之后不能再点击鼠标左键。

但两种方式是不能互操作的，比如使用“复制”菜单项进行复制的内容不能通过鼠标中键进行粘贴。可以通过 autocutsel 工具对它们进行同步，将下面程序加入到自启动脚本中即可：

```
autocutsel &
autocutsel -selection PRIMARY &
```

5.2. 字体

X Window 的字体主要是通过 libXft 处理，而它又基于 fontconfig 和 freetype ，fontconfig 可以在 `$HOME/.fonts.conf` 文件中对其进行配置，比如字体选择、 AA 开关等。对于 libXft 可以在 `$HOME/.Xdefaults` 进行配置，示例如下：

```
Xft.dpi:       96
Xft.antialias: true
Xft.rgba:      rgb
Xft.hinting:   true
Xft.hintstyle: hintslight
Xft.lcdfilter: lcddefault
```

现在大部分 GTK 程序都使用 cairo 渲染字体， cairo 在 1.10.0 版本之后，加入了对 lcdfilter 的支持，这样在使用 LCD 显示屏时开启 lcdfilter 可以得到更为理想的字体显示效果。

以上只是对字体渲染的设置，对于中文用户来说，还需要进行默认中文字体的设置，这个设置由 fontconfig 处理，在 `$HOME/.fonts.conf` 配置文件中。可以在该文件中对 sans, serif, monospace 设置默认中文字体，详细方式参考 配置文件 。

关于 fontconfig 配置的更多细节，可以查看官方文档： http://www.freedesktop.org/software/fontconfig/fontconfig-user.html ，也可以查看 ArchWiki 里的相关说明： https://wiki.archlinux.org/index.php/Font_Configuration 。

第 3 章 资源管理

计算机通过文件系统组织和管理资源，可以把文件系统理解为一个抽屉，而其中存放着许多的文件夹或文件，而在文件夹里可以再存放文件或文件夹。只不过计算机中的文件夹与现实的一点不同是它可以无限制的嵌套。

而对于用户来说，作为一个资源管理工具，必须要提供的基本操作包括：打开文件、移动、重命名、删除文件等。而进行这些操作的一个前提是可以快捷地定位文件。

在 Linux 中，由于命令行程序的广泛使用，资源管理工具并不是必需的，所有相关功能都可以在命令行下完成。但对于普通的桌面用户来说，增加了不必要的学习负担，所以这里介绍使用 lfm[1]来进行资源管理。

lfm 本身是一个字符界面的程序，但这并不影响它在桌面环境下的使用[2]。比如通过下面的命令可以利用 sakura 虚拟终端打开 lfm ，再将该命令利用上面介绍的 xchainkeys 工具定制快捷键，就可以用类似 Windows 中的 Win+E 的方式对系统资源进行管理。

```
sakura --title 'File Manager' -e lfm
```

另外，除了的对普通文件进行管理外，像多媒体文件，比如音频文件，现在很多音频播放器都提供媒体库功能，可以通过它们实现对音频文件的分类管理。

接下来先简单介绍 lfm 的使用，再来讨论与资源管理相关的其它一些话题。

[1]lfm 是 Last File Manager 的缩写，详细信息可以查看官方网站： http://www.terra.es/personal7/inigoserna/lfm/ 。

[2]在 Windows 中，字符界面的程序几乎不再使用，而 Linux 中则依然有大量的使用，不仅仅是一些开发时间较早的程序，就是一些新近设计开发的程序而有一些选择使用字符界面，这可以是出于程序的稳健以及开发维护复杂度的考虑。

1. lfm 介绍

1.1. 文件定位

要实现文件的快速定位，需要从以下几个方面考虑:

方便的目录切换
: lfm 中有多种方式进行目录切换，通过 `g` 键可以直接输入想要打开的目录。也可以在列表中通过左右方向键进行上下层目录的切换。

文件搜索
: 对于文件的搜索， lfm 中可以通过 `Ctrl+S` 在当前列表中搜索。也可以通过 `/` 命令在当前目录下进行递归搜索。

排序
: lfm 的 `s` 命令可以对文件进行各种方式的排序。由于系统为 UTF-8 编码环境，中文无法以拼音字母顺序进行排列。

文件名定位
: 在 Windows 的资源管理器中，可以通过文件名的首字母对文件进行快速定位。而在 lfm 中可以通过 `Ctrl+S` 搜索命令取代，与排序同样的一个问题是，对中文文件的定位并不方便，因为需要输入中文。

快捷方式
: Windows 的快捷方式对应于 Linux 的符号链接， lfm 中可以通过 `l` 命令创建。

1.2. 打开文件

对于打开文件的操作，主要是需要对不同的文件类型进行判断。在 lfm 中，需要通过配置文件进行指定。配置文件存放在 `$HOME/.lfmrc` ，通过`[Programs]`和`[File types]`这两段配置可以指定，例如以下配置指定使用 acroread 打开 pdf 文件。

```
[Programs]
...
pdf: acroread
...

[File Types]
...
pdf: pdf
...
```

另外 lfm 可以通过 F3 查看文件，通过 F4 编辑文件。编辑和查看所使用的程序可以在`[Programs]`中的`editor`和`pager`指定。

更多类型的设置可以参考这里的配置： https://github.com/dram/configs/tree/master/.lfmrc 。

1.3. 其它操作

在 lfm 中，删除使用 `F8` ，重命名使用 `F2` ，复制使用 `F5` ，移动使用 `F6` 。

通过 `INSERT` 可以同时选中多个文件进行批量操作。

lfm 还有其它很多实用的功能，具体可以查看 http://www.terra.es/personal7/inigoserna/lfm/#keys 。

1.4. lfm 的不足

总的来说， lfm 作为文件管理工具，基本功能已经比较完善，主要的不足有：

1. 对中文的支持并不理想；
2. 按键上没有 vifm 方便；
3. 没有 `Trash`(回收站)功能；
4. 没有整合 locate 工具。

2. 文件权限

Linux 从最初就是作为多用户操作系统进行设计，所以它的文件权限管理非常完备。Linux 中每一个文件或文件夹都可以对三类用户分别设置权限：文件拥有者、文件所属用户组以及其它用户。每类用户都可以有读、写、执行三类权限。可以通过 umask 命令对创建文件时的默认权限进行设置。

但对于普通桌面用户来说，一般为个人使用，所以不需要特别关心。

3. 数据备份

在计算机中，数据对于用户来说往往是最为重要的。所以对于重要数据必须进行必要的备份。可以考虑像 Dropbox 这类网络硬盘方式，它对 Linux 也有较好的支持。如果文件不大，也可以考虑使用邮箱进行数据备份，利用 cron 定时将数据打包以附件形式发送给邮箱。

而对于一些不涉及版权问题并希望与其他人分享的数据，也可以在 github, googlecode 等免费代码管理服务中进行备份。下面就以本文涉及到的配罟文件为例，说明基本的使用方式。

3.1. 配置文件管理

Linux 程序和 Windows 程序在设计理念上的一个重要不同是， Linux 程序大量采用可以直接编辑的文本配置文件。其中的一个好处是，可以通过版本控制程序来管理这些配置文件。

配置文件经由版本控制管理后，可以追溯以前的修改，同时如果将其放在网络上的公共版本管理服务中，还可以与其他用户分享。

版本管理工具有很多，比如 Subversion, Git, Mercurial 等，下面主要介绍如何通过 git 工具下载本文中的配置。关于 git 的基本使用方法介绍，可以从这里查看： http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html 。

通过以下命令，可以下载本文涉及的所有配置文件:

```
git clone git://github.com/dram/configs.git
```

但使用上面的命令下载的配置文件并没有直接存放到 HOME 目录中。可以通过在 HOME 目录执行下面的命令，将这些文件导入到 HOME 目录:

```
git init
git remote add github git://github.com/dram/configs.git
git fetch github
git checkout master
```

第 4 章 输入法

在 PC 中输入文字，一般有这样几种方式：键盘输入、手写识别和语音识别。而现在主要的输入方式还是依靠键盘完成。

对于英文，所有单词都是由 26 个字母组成，所以可以通过键盘直接输入需要的单词。而对于中文来说，由于字符太多，不可能将每一个汉字与键盘上的按键一一对应，所以只能通过将键盘上的几个字母的组合对应为一个汉字或词组，以此实现中文的输入，对于这一对应关系的处理就是输入法的主要功能之一。

输入法是非常重要的基础性应用程序，所以不管在 Windows 还是 Linux 下，都已经有比较完善的支持。在这里，为了让输入法与之后将要介绍的 Vim 编辑器较好的协同工作，选用了 uim 这一输入法框架，并在它原有郑码输入法的基础上对其加以完善。这里主要参考了 Windows 中极点郑码的设计，因为对于输入法来说，非常重要的一点是与其它系统保持一致。另外在设计中，尽量保证每次字词输入按键的一致性，以便于提高输入速度。

相关的代码在： https://github.com/dram/configs/blob/master/.uim.d/plugin/zhengma.scm 。另外 uim 的配置文件可以参考这里： https://github.com/dram/configs/blob/master/.uim 。

下面就来介绍 uim 输入法的一些细节。

1. 安装

在 Linux 中，输入法是通过动态库的方式整合到应用程序中的，所以在安装时需要指定需要加载的输入法模块。在 GTK 中，可以通过 gtk-query-immodules-2.0 命令查找当前安装的所有输入法，在其中找到于 uim 相关行，将其加入到 `/etc/gtk-2.0/gtk.immodules` 文件中。再通过设置下面环境变量即可完成 uim 的基本安装设置。

```
export GTK_IM_MODULE=uim
```

```
export XMODIFIERS=@im=uim
export QT_IM_MODULE=uim
```

这里对 uim 郑码输入的完善是通过 uim 模块的方式实现的，安装的方法是将上面提及的文件存放在 `$HOME/.uim.d/plugin/` 目录，再通过下面的命令对其进行加载：

```
uim-module-manager --register zhengma --path $HOME/.uim.d/plugin/
```

另处还需要安装郑码的码表文件，这里对 uim 原先的郑码码表进行了一些调整，相关脚本在这里可以找到： https://github.com/dram/configs/tree/master/tools/uim-table/ 。

2. 功能完善

接下来简要说明这个郑码模块对原有 uim 中文输入的优化。

2.1. 自动上屏

在郑码中，最大码长了 4 码，所以在极点郑码中，如果已经输入 4 码，并且没有重码，那么会将唯一的候选词直接上屏。 uim 中虽然有自动上屏功能，但它会将不到 4 码的字也直接上屏，并不符合一般的中文输入习惯，这里进行了相应的调整。

2.2. 第二候选词

由于对于郑码这类形码输入法来说，重码出现的几率不高，并且在单字输入时，即使有重码，大多也只是两重的，所以需要一个便捷的方法来选取第二候选词。 uim 默认不具有这一功能，这里与极点保持一致，使用 **;** 作为第二候选词的按键。

2.3. 英文上屏

在编辑一些文档时，可能需要同时输入中英文，这样就需要输入法支持的开启状态下输入少量中文。这里与 FCITX 一致，使用回车键对英文进行直接上屏。

2.4. 状态提示

在中英文混合输入时，由于经常在中英文输入之间进行切换，所以输入法应该能够直接地提示当前处于什么状态。这里使用候选框作为状态的提示，中文状态下候选框始终显示，而英文状态下不显示。

2.5. 单字

在大多中文输入法中，一般都提供是否只输入单字的选项，而这里直接在码表中进行处理，删除了码表中的所有词组。

2.6. 筛选简体中文

与单字输入的处理相似，这里通过在码表中删除所有繁体中文的方式实现只输入简体中文的功能，这样可以减少重码数。

3. Vim 协同工作

对于让输入法与 Vim 模式切换协同工作的实现方式，主要可以分为在编辑器实现和在输入法中实现。以下对这两者分别予以说明。

3.1. 编辑器实现

编辑器中的实现可以查看 GVim 的 `imdisable` 和 `noimdisable` 参数，具体可以如下设置:

```
autocmd! InsertLeave *  set imdisable
autocmd! InsertEnter *  set noimdisable
```

通过上面的配置可以让 GVim 在离开和进入输入模式时对自身窗口的 XIM 进行同步切换。具体的动作可以描述为:

1. 从 INSERT 转为 NORMAL 模式时，输入法被禁用。

2. 从 NORMAL 模式转到 INSERT 时，如果之前输入法是开启的，将依然保持状态。

但这一方式只适用于 GVim ，对于虚拟终端下的 Vim 则不能起作用，因为 Vim 无法控制虚拟终端的 XIM 。并且不同输入法在 GVim 中 imdisable 和 noimdisable 切换时的行为也不同，所以实际效果并不理想。

3.2. 输入法实现

而另一种实现方式是在输入法中实现。比如在 SCIM 中可以通过将关闭输入法快捷键设置为 `ESC + KeyRelease` ，也就是将 ESC 的 KeyPress 的动作传递给 Vim ，对 KeyRelease 动作进行捕捉。

但这只实现了上述两点中的第 1 点。在重新回到 INSERT 模式时，因为 SCIM 无从知道 Vim 是否已经回到了 INSERT 模式，所以无法实现输入法的自动重新开启。

3.3. 另一种选择

在这里，通过在输入法中增加临时停用及恢复的功能，再在 Vim 中通过外部程序模似按键的方式实现了上述两点功能。主要思路为在输入法中监测 `ESC` 和 `Ctrl+C` 按键，在有这两个按键时，临时关闭输入法。而在 Vim 中，在 InsertEnter 事件时调用外部程序模拟键盘事件，触发输入法临时停用的恢复功能。模拟键盘事件的程序可以在这里查看： https://github.com/dram/configs/tree/master/tools/fake-key/ 。

第 5 章 网络

从最开始的静态网页、电子邮件以及 Telnet BBS，到现在的博客、网上购物、社交、微博，网络的发展不可谓不快。网络的发展，不仅仅技术的革新，更是在技术应用上的发展，而这些都切切实实地改变着我们的生活。

网络给我们带来了更为迅速的信息传播，更为便捷的通信方式。但网络也必有其弊端，那么我们又该警惕什么呢？比如信息爆炸，比如永久在线，还有更多层出不穷地网络新概念，这些都是值得我们思考的。网络已经不仅仅是一个技术问题，它更是一个社会问题。

下面简单介绍 Linux 中的浏览器以及通信工具。

1. 浏览器

现在占主导地位的浏览器主要有 Mozilla 的 Firefox， Google 的 Chrome， Apple 的 Safari 以及 Microsoft 的 IE 。作为网络的重要入口，各浏览器之间的竞争自然也是热闹非凡。也可以从浏览器之间的比较中看出对于一个软件来说什么是最为重要的。

对于浏览器来说，非常关键的几个参数是：

稳定性
: 稳定性是对于绝大部分软件的基本要求，而对于浏览器来说也非常重要，因为 WEB 中存在大量不符合标准的网站，浏览器必须能保证不会因为这些网站而导致自身崩溃。

兼容性
: 由于 WEB 标准的滞后，以及不同浏览器这间的兼容性问题，一个网站在设计时很可能只考虑在某一个或几个特定的浏览器中正常使用。而浏览器的设计应该在保证遵循标准的同时，尽可能地与其它浏览器保持兼容，从而减轻网页开发者和使用者的负担。

安全性
: WEB 应用的安全主要需要从两方面考虑，一是同站自身的安全，再是浏览器的安全。

性能

由于现在的网络应用越来越复杂，对于网络性能上的要求也就越来越高。网络整体的性能受多方面的影响，比较网络带宽，网络接口，协议设计以及实现等。而浏览器也是非常重要的一环，比如渲染引擎的性能，包括对 HTML, CSS, Javascript 等的解析，再比如缓存机制、预加载等。

功能

而在功能性上，由于不同用户对于 WEB 应用有着全然不同的需求，要同时满足这些用户，需要有一个方便的扩展接口。

这里选择使用 Firefox ，主要是因为它较为丰富的插件，以及它较大的市场占有率。下面对 Firefox 做一些简单的介绍。

1.1. 插件

下面例举一些在 Firefox 中常用的插件:

Adblock Plus

屏蔽广告。可以定阅不断更新的 URL 列表，能够屏蔽大部分中英文广告。

DownThemAll!

多线程下载工具。

Flashblock

屏蔽页面中所有的 Flash 。 Flash 对浏览器整体的性能有较大影响，使用 Flashblock 插件可以让页面中的所有 Flash 默认不显示，如有需要显示的可以手动开启。

Gmail Watcher

Gmail 邮件提醒。于 Gmail 邮件提醒相关的 Firefox 插件有很多，但总体来说 Gmail Watcher 最为完善。

1.2. 其它

下面再来介绍一些 Firefox 的快捷键。如果需要完全使用键盘控制 Firefox ，可以考虑使用 vimperator 插件，具体可以看这里： http://vimperator.org/。

Ctrl+L

转到地址栏

Ctrl+K

搜索栏

Ctrl+Tab

标签页切换

Ctrl+T
: 新建标签页

Ctrl+W
: 关闭标签页

Ctrl+R ， F5
: 刷新页面，另外 Ctrl+F5 可以让浏览器忽略自身缓存

Alt+Left Alt+Right
: 后退、前进

Ctrl+左键
: 在新标签页打开键接

2. 通信工具

PC 的通信工具基本上有 Email 、 IRC 、即时通信工具等。 Linux 中有相应工具对其支持。比如 Pidgin 、 Thunderbird 等。由于使用方法比较直观，这里不再做介绍。

这些技术虽然给我们带来了便捷，但同时带来了一些问题，比如安全问题、隐私问题、真实性问题等等。

第 6 章 文本编辑

对于一个桌面应用系统来说，文本编辑器其实并不是必需的。就比如在 Windows 中，对一般用户来说，如果有 Word 等文字处理工具，那么记事本被使用到的频率并不高。但在 Linux 中有所不同，在很多时候都需要用到文本编辑器。一方面是因为 Linux 中大量使用文本配置文件，而另一方面 Windows 中 Word 的文字处理功能，在 Linux 中可以通过文本编辑器实现，这将在下一章再作介绍。

如果只是需要文本编辑器的基本的、核心的功能，那么像记事本这样的程序就可以满足要求，但如果需要更为便捷、更为复杂的功能，那就要使用更为强大的编辑器了。 Linux 中的编辑器多得甚至于无法记数，最为流行的有 Vim 和 Emacs ，下面将要介绍的就是其中的 Vim 。

Vim 的基本操作就不在这里作介绍了，不太熟悉的读者可以通过 `vimtutor` 命令学习，下面介绍的主要是一些零散的技巧。

相关的详细配置可以在 https://github.com/dram/configs/tree/master/.vimrc 和 https://github.com/dram/configs/tree/master/.vim/ 中查看。

1. 中文处理

有必要先就 Vim 的中文支持作一说明。 Vim 本身对于多语言已经有较好的支持。只是在中文句点判断、输入法切换上还不太方便，下面介绍对这些问题的处理。

1.1. 文件编码

在 Vim 中针对文件编码的选项主要有三个： `fileencoding` 、 `fileencodings` 和 `encoding` 。其中 `encoding` 指 Vim 自身在存储信息时的编码，而 `fileencoding` 对文件写入起作用，用于指定 Vim 使用什么编码写入文件。 `fileencodings` 作用于读取文件时，

Vim 尝试使用 `fileencodings` 里所列编码读取文件，如果尝试成功，则将 `fileencoding` 设置为该编码。

读取文件时，除了参考 `fileencodings` 之外，也可以用 `:edit ++enc=gbk` 的形式直接指定编码。

基于上面的说明，如果需要更改文件编码，首先通过设置 `fileencodings` 或直接指定 `++enc` 参数使 Vim 正常显示文本，再设置 `fileencoding` 并保存即可。

1.2. 中文句点

Vim 是基于英文的词语和句子来定义移动的，这样在中文的行内移动就会比较麻烦。对于词语，没有什么好的处理办法，因为这涉及到中文分词问题。而中文句子间的移动还是可以通过配置来实现的。

英文句子间移动的命令是（ 和 ），下面配置对这两个命令进行重新定义，使其支持对中文句点的判断。

```
nmap <silent> ( :call search('\n\\|。\\|！\\|？\\|\.\s\\|!\s\\|?\s', "bw")<CR>
nmap <silent> ) :call search('\n\\|。\\|！\\|？\\|\.\s\\|!\s\\|?\s', "w")<CR>
```

1.3. 输入法切换

在 GVim 中可以通过下面的设置实现在切换 INSERT 和 NORMAL 模式时关闭或重新开启输入法。

```
autocmd! InsertLeave *  set imdisable
autocmd! InsertEnter *  set noimdisable
```

但上面的命令只对 GVim 有效，终端下的 Vim 是无效的，因为实际上 `imdisable` 作用的是当前的图形窗口程序的输入法开关，而并非是输入法程序的开关。

至于终端下的 Vim ，也可以使用在 输入法 一章所介绍的方法。

2. 普通编辑

接下来介绍在使用 Vim 编辑普通文件时的一些小的技巧。

2.1. 移动

`*` 和 `#` 是两个非常有用的快捷键，可以正向或反向查找光标所在单词。

Vim 虽然在显示较长文本行时，会对其进行折叠，然而普通的 `j` 和 `k` 命令依然只是在真实的行之间进行移动，但可以使用 `gj` 和 `gk` 命令在行内的折叠行间移动。

行内的移动还可以通过 `f` 和 `F` 命令处理，这两个命令用于进行行内搜索。 `;` 用于重复 `f` 或 `F` 的动作。

更为复杂的移动操作可以通过 mark 实现， `m[a-z]` 用于标记光标当前位置，通过 `` `[a-z] `` 回到相应 mark 所在位置，而 `'[a-z]` 则是回到 mark 所在行。

2.2. 换行符

在 Winodows 中以 `\r\n` 表示换行， Unix 中以 `\n` 表示， Mac 中以 `\r` 表示，因为这个原因，在跨平台工作时，还是会带来不少麻烦的，好在 Vim 中可以方便地对换行符进行识别和转化。

Vim 中换行符的处理与上面说明的文件编码处理是类似的， `fileformats` 作用于读取， `fileformat` 作用于写入。而 `++ff` 对应于 `++enc` 。

2.3. 复制粘贴

在桌面环境一章中已经提到， X11 窗口系统中有两套复制粘贴的方式，但可以通过 `autocutsel` 命令将其同步。而在 Vim 中，又有其自身的复制粘贴的方式。具体说来， Vim 中的 `"*` register 相对于 X11 的 selection ，而 `"+` 相对于 X11 的 clipboard 。

2.4. 文件切换

在编辑文件时，往往不是对单个文件进行处理，而是同时对多个文件进行编辑。这样在文件之间方便地切换就显得比较重要了，下面介绍常用的几种方式。

打开文件所在目录

编辑的多个文件常常在同一目录中，这时可以通过 `:cd %:h` 命令让 Vim 先跳转到该目录，再结合 `:edit` 命令以及 `Ctrl-D` 补全打开需要的文件。

也可以直接使用 `:edit %:h` 命令，通过 Vim 的 netrw 组件打开该目录。

另外还可以在 Vim 配置文件中加入下面的配置，这样在输入 `:edit %/` 之后， Vim 会将 `%/` 替换为当前文件所在目录的路径。

```
cmap %/ <C-R>=expand("%:p:h")."/"<cr>
```

MRU 插件

MRU[1]插件用于记录最近打开的文件。通过 :MRU 命令打开 MRU 窗口。由于经常使用，可以对其设置快捷键:

```
let mapleader = ','
nmap <silent> <leader>m         :MRU<CR>
```

buffer

Vim 对于打开过的文件，会以 buffer 的形式加以管理。通过 :buffers 或 :ls 命令可以查看当前打开的所有文件。 :bn 和 :bp 命令用于在 buffer 间切换，也可以使用 :b num 直接跳转到指定 buffer 。 :bd 用于删除 buffer 。另外如果不想关闭 Vim ，只是想关闭对该文件的编辑，可以使用 :bd 代替 :q 。

最近编辑

在编写程序时，经常需要在两个文件之间进行切换，比如 .h 和 .c 文件间，或者实现与调用间，或者实现与测试间，或者编辑和帮助间的切换， CTRL-6 , CTRL-^ 或 :e # 可以用来在两个最近编辑的文件间切换。

具体说明可以查看 :h CTRL-6 。

2.5. INSERT 模式

Vim 注重 NORMAL 模式下的便捷处理，在 INSERT 模式下一般主要还是进行简单的文本输入操作。但 Vim 也支持在 INSERT 模式下的一些快捷操作。

比如 CTRL-W 可以删除一个单词，而 CTRL-U 可以删除一整行。另外 CTRL-N 和 CTRL-P 这两个快捷键可以进行简单的文本补全。

通过 :h ins-special-keys 可以查看 INSERT 模式下所有快捷方式的详细说明。

3. 代码编写

Vim 更多的是用于代码编写，它对于大多数语言都有很好的支持。以下介绍一些使用 Vim 进行代码编写的技巧。

[1]MRU 插件可以从 http://www.vim.org/scripts/script.php?script_id=521 下载，存放到 $HOME/.vim/plugin/ 目录下即可。

3.1. 代码风格

不同类型的程序往往会有不同的代码风格，比如 C 语言一般以 Tab 缩进，而 Python 则更多的以 4 个空格进行缩进。

在 Vim 中可以在 `$HOME/.vim/ftplugin/` 目录中创建以类型名启始命名的文件对该类型进行定制。比如可以在 `$HOME/.vim/ftplugin/python_own.vim` 文件中加入：

```
setl shiftwidth=4
setl expandtab
```

甚至可以为不同的文件类型创建不同的快捷键，同样是在该文件中加入下面的配置后，可以通过 `<leader>r` 快捷键运行该 Python 脚本文件。

```
nmap <buffer> <leader>r          :w<cr>:!python %<cr>
```

另外，通过 colorcolumn 可以在代码行超出指定长度时给予提示，从而保持良好的代码风格，具体配置如下：

```
set colorcolumn=81
hi ColorColumn ctermbg=darkgrey
```

3.2. text objects

Vim 有一 text object 的概念，这在编写程序时非常有用，可以对 `()` , `""` , `<></>` 等整体进行处理，比如 di(可以删除括号内的文本， ci" 可以修改引号内的文件。详细说明可以查看 `:h text-objects` 。

3.3. 简单的模板

很多的程序文件特别是脚本语言中每个文件都有部分固定的格式，在 Vim 中可以通过 `autocmd` 命令实现简单的模板功能，以避免重复劳动。下面以 Python 脚本为例作简单说明。

首先需要一个模板文件，比如 `~/.vim/templates/template.py` ：

```
#!/bin/env python
# vim: set fileencoding=utf-8
```

```
if __name__ == '__main__':
    pass
```

再是在 .vimrc 中通过 autocmd 命令设置在新建 .py 文件时自动读取模板中的内容：

```
au BufNewFile *.py :0r ~/.vim/templates/template.py
```

4. 其它技巧

4.1. 查看 man

Vim 下的 :Man 可以用来查看 man 文档。这在编写 C 程序或 SHELL 脚本时都非常有用。

4.2. Caps Lock 替换 ESC

很多 Emacs 的用户会将 Caps Lock 键替换为 Ctrl，而在 Vim 也可以借鉴这一方式，将 Caps Lock 替换为 Esc 键。

首先创建 $HOME/.Xmodmap 文件，内容为：

```
! Swap caps lock and escape
remove Lock = Caps_Lock
keysym Escape = Caps_Lock
keysym Caps_Lock = Escape
add Lock = Caps_Lock
```

然后在 $HOME/.xinitrc 中加入：

```
if [ -f ~/.Xmodmap ]; then
    xmodmap ~/.Xmodmap
fi
```

另外，在 Windows 中也可以通过注册表设置，用以下内容创建 reg 文件并运行即可。

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Keyboard Layout]
"Scancode Map"=hex:00,00,00,00,00,00,00,00,02,00,00,00,01,00,3a,00,00,00,00,00
```

5. 扩展

Vim 的一个重要特点是可以通过多种语言对其进行扩展，同时它也有自身的脚本语言。在下面的多媒体一章将使用 Vim 的可扩展性实现一个简单的音频播放器。

第 7 章 文字处理

在 Windows 中，较为常用的文字处理软件是 WPS 或者 Word 等所见即所得(WYSIWYG[1])的编辑器。 Linux 也有类似的工具，比如 OpenOffice.org 、 KOffice 等。

但除了所见即所得编辑器之外，还可以使用 TeX 等标记型语言来处理文档的排版，相对于所见即所得的编辑器来说，使用标记型语言编写的文档维护更加方便、样式更为统一，很多时候最终生成的文档也更为美观。

但这类标记型语言有一个的缺点是标记比较复杂，有一定的入门门槛，并且在编写文档时经常需用输入大量的标记。不过通过对编辑器进行合理的配置可以部分解决这一问题。下面主要介绍编写本文所使用的 DocBook 标记语言，它以 XML 格式来组织文本。

读者也可以选择一些轻量级标记语言，比 asciidoc, txt2tags, markdown, pandoc 等，相对来说，这些轻量级的标记语言标记更为简洁。它们不是直接生成最终文档，而是先转化为 Tex 、 Docbook 等标记语言，再最终转化为所需要的文档。

这里使用的是 DocBook 5.0 版，详细的文档可以查看 http://www.docbook.org/tdg5/en/html/docbook.html 和 http://www.sagehill.net/docbookxsl/index.html 。

1. 工具

由于 DocBook 本身是 XML 文档，所以可以借助一些通用的 XML 处理工具对 DocBook 文档进行处理。这里主要用到了 Jing[2]和 xsltproc 这两个工具，分别用于语法检查和格式转化。

[1]WYSIWYG 是 What You See Is What You Get 的缩写。关于它的详细说明，可以看这里： http://en.wikipedia.org/wiki/WYSIWYG 。

[2]另一个常用的 XML 格式检查工具是 xmllint ，相对来说， Jing 对 RELAX NG 有更好的支持，这在编写 DocBook 文档时非常重要，借助于它可以更容易发现可能的格式错误。

先从 DocBook 官网下载 DocBook5.0 Schemas 的代码(http://www.docbook.org/xml/5.0/docbook-5.0.zip)，可以通过其中的 `rng/docbook.rng` 文件结合 Jing 工具对 DocBook 文档进行语法上的检查，具体方式如下：

```
java -jar /path/to/jing.jar /path/to/rng/docbook.rng file.xml
```

而 DocBook 到其它格式的转化则需要借助与 docbook-xsl-ns(http://sourceforge.net/projects/docbook/files/)和 xsltproc 工具，下面的命令就是通过它们将 DocBook 文档转化为 fo 文档，之后再交由 FOP 工具将 fo 转化为 pdf 文档：

```
xsltproc -o output.fo /path/to/xsl-ns-stylesheets/fo/docbook.xsl input.xml
```

关于 FOP 的设置随后会详细介绍，接下来先介绍对编辑器的配置。

2. 编辑器

这里选择使用 Vim 作为 DocBook 文档的编辑器， Vim 自身对 XML 编辑上的支持并不多，需要先从 http://www.vim.org/scripts/script.php?script_id=1397 下载一个插件，这个插件可以对 XML tag 进行自动补全，并提供 tag 跳转等功能。

2.1. 语法

由于 DocBook 文档一般是较大，并且有较多的长行， Vim 在对这类 XML 文本进行语法高亮时性能较差，直接影响编辑的效率，所以有必要对它进行一些调整。 https://github.com/dram/configs/blob/master/.vim/syntax/docbk.vim 这份 Vim 语法文件是针对 DocBook 进行的精减，同时支持对章节的代码折叠，在 DocBook 文档的最后加入下面 modeline 的设置即可加载该文件并启用折叠：

```
<!-- vim: set ft=docbk fdm=syntax : -->
```

2.2. 缩进

Vim 默认开启 XML 文档的缩进，如果需要关闭的话，可以在 `$HOME/.vim/indent/` 目录新建 `xml.vim` 文件，加入下面的内容：

```
let b:did_indent = 1
```

2.3. 折叠

在语法一节中已经提到如何开启代码折叠，另外还可以在 ftplugin 中设置 `foldlevel` 以及 `foldnestmax` 这两个参数设置默认展开几级以及最多折叠几级。

2.4. 移动

在安装了 xml.vim 插件之后，可以通过 `[[` , `]]` , `[]` , `][` 这几个按键在 XML 标签之间移动。但感觉默认的设置不是很习惯，可以在 xml.vim 中对其进行重新配置，可以参考 https://github.com/dram/configs/tree/master/.vim/ftplugin/xml.vim ，重新设罟后的按键含义为：

`[[`
: 上一个 open tag

`[]`
: 上一个 close tag

`][`
: 下一个 open tag

`]]`
: 下一个 close tag

3. 脚本

从以上的介绍中可以看到，将 DocBook 转化为 PDF 文档需要结合使用几个工具，可以通过 Makefile 对其进行管理，生成本文档所用的 Makefile 可以到这里查看： https://github.com/dram/docs/blob/master/Makefile 。

4. FOP 中文支持

FOP 本身是支持多语言的，这里唯一需要处理的是字体设置的问题。

本文档使用的字体主要有：文鼎 PL 简报宋、文鼎 PL 简中楷、 Computer Modern Unicode 以及 Droid Sans Fallback 。

`文鼎 PL 简报宋`
: http://ftp.debian.org/debian/pool/main/t/ttf-arphic-gbsn00lp/ttf-arphic-gbsn00lp_2.11.orig.tar.gz

文鼎 PL 简中楷
: http://ftp.debian.org/debian/pool/main/t/ttf-arphic-gkai00mp/ttf-arphic-gkai00mp_2.11.orig.tar.gz

Computer Modern Unicode
: http://canopus.iacp.dvo.ru/~panov/cm-unicode/

Droid Sans Fallback
: http://android.git.kernel.org/?p=platform/frameworks/base.git;a=tree;f=data/fonts

FOP 不能直接读取其中的 Droid Sans Fallback 字体，需要使用 fontforge 打开该字体再重新导出为 TTF 字体。

设置 FOP 的字体分两步进行。先是让 FOP 识别这些字体，这个通过 FOP 的配置文件完成，再是让 PDF 文档使用这些字体，这个通过 DocBook 的 XSL 文件进行设置。详细的配置可以查看下面两份文件:

fop.xconf
: https://github.com/dram/docs/blob/master/fop.xconf

fo.xsl
: https://github.com/dram/docs/blob/master/fo.xsl

最终在调用 fop 生成 PDF 文件，命令示例如下:

```
fop -c fop.xconf input.fo output.pdf
```

5. PDF 优化

单是设置了字体后 FOP 生成的 PDF 效果依然不是很理想。还需要进行一些微调，配置还是写在上面提及的 `fo.xsl` 文件中。另外中英文混排的问题通过脚本进行优化。

5.1. 中英文混排

在 TeX 中，有专门的 xeCJK 包提供对中英文混排的优化，而在 FOP 中并没有对其做优化。这样看起来中文和英文会有些拥挤，这里通过一个脚本分析 DocBook 的 XML 文件，自动在中文和英文间加入空格，以此实现与 xeCJK 相似的功能。

详细代码可以从这里查看： https://github.com/dram/docs/blob/master/optimize-char-spacing.py 。

5.2. 排版

在 FOP 中可以对很多排版细节进行详细的设置，比如首行缩进两格可以通过下面的方式配置:

```
<xsl:attribute-set name="normal.para.spacing">
  <xsl:attribute name="text-indent">2em</xsl:attribute>
</xsl:attribute-set>
```

行距设置:

```
<xsl:param name="line-height">1.6</xsl:param>
```

还有比如正文字体大小、标题高度等等，都可以设置，这里就不再一一列举，读者可以从 `fo.xsl` 文件中找到相应的设置。关于这些参数的说明可以到这里查看： http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html 。

5.3. 封面调整

在 FOP 中，如果需要对封面进行调整，需要先根据 docbook-xsl 目录中的 `fo/titlepage.templates.xml` 进行调整，再利用 xsltproc 通过 `template/titlepage.xsl` 将新的模板转化为 XSL 文件，之后就可以通过这个 XSL 文件对 DocBook 文本进行转化了。

第 8 章 多媒体

多媒体应用主要包括音频和视频，这里主要介绍 Linux 下音频播放相关内容。

1. 音频播放器

虽然在线视听已经有多年的发展，网络收音机、音乐盒等概念也在近两年逐渐流行，但音乐播放器作为传统的桌面应用程序，依然在桌面应用中占有着非常重要的地位。所以在 Linux 中有着大量的音乐播放器可供选择。

从用户角度来说，音乐播放器中最为重要的功能包括：解码播放、资源管理、播放列表、播放控制。

由于没有找到合适的软件，这里通过 mpg123 和 Vim 实现了一个简易的播放器。mpg123 有一个非常方便的开发接口[1]，通过它可以方便地对其进行包装。完整的代码如下：

vim-mpg123.py
: https://github.com/dram/configs/tree/master/bin/vim-mpg123.py

mpg123.vim
: https://github.com/dram/configs/blob/master/.vim/plugin/mpg123.vim

xchainkeys 配置文件
: https://github.com/dram/configs/blob/master/.config/xchainkeys/xchainkeys.conf

下面分别就上述四个方面对这一播放器的实现作简要说明。

[1]通过 mpg123 的`--remote`参数可以利用 mpg123 进程在标准输入输出对 mpg123 进行控制。

2. 解码播放

这里的解码播放功能通过 mpg123 实现，它只支持 MP3 格式文件，现在虽然有 OGG, WMA 等有损压缩格式以及 FLAC, APE 等无损压缩格式的出现，但 MP3 依然是最为通用和流行的一种格式。如果需要对多种格式进行支持，可以考虑 GStreamer 等解码库[2]。

另外，为了能通过多种途径控制播放，这里使用 C/S 模式组织程序，以 FIFO 文件为接口。

3. 资源管理

现在很多播放器以媒体库的形式组织音乐，但就个人而言，简单的目录结构组织已经足够，所以这里的资源管理和浏览实际上就是在 Vim 编辑器中显示目录内容。同时定义了一些快捷键用于向播放器服务端发送指令。

4. 播放列表

很多播放器在显示播放列表时会读取音频文件的 ID3 信息，而这里做了简化处理，直接显示文件名。文件名事先经过整理，包含了 track number 以及 title 信息。

5. 播放控制

由于采用了 C/S 模式，可以实现多种播放控制方式，这里除了可以在 Vim 中控制外，还可以通过键盘直接控制播放，详细信息请查看 xchainkeys 的配置文件。

另外，音量控制一般会在 systray 下实现，像 Gnome ， KDE 都会在 systray 中显示音量控制图标，显示当前音量，同时通过它也可以对音量进行调节。而其它窗口管理器也可以使用 gvolwheel, pyvolwheel 等程序得到同样的功能。

而这里则是使用 ossmix 、 xchainkeys 和 libnotify[3]提供了完全的键盘控制，因为对于音量调节不外乎这么几个功能：增大音量、减小音量，还有静音切换。这些功能完全可以通过键盘控制。具体实现可以查看下面的代码以及 xchainkeys 的配置。

volume-control

https://github.com/dram/configs/blob/master/bin/volume-control

volume-control-alsa

https://github.com/dram/configs/blob/master/bin/volume-control-alsa

[2]GStreamer 提供多种语言接口， https://github.com/dram/configs/blob/master/bin/vim-gst-srv.py 这里是一个 Python 接口的示例。

[3]libnotify 基于 dbus 通过 C/S 模式实现，现在的 daemon 端依赖于一些 Gnome 库，但也可以选用 XFCE 的 daemon 实现。

第 9 章 SHELL

本章主要来讲讲 SHELL ，对桌面用户来说，理想情况下并不需要了解 SHELL ，但了解 SHELL 有助于了解 Linux 的运作机制。同时有些与操作系统直接相关的操作也需要通过 SHELL 来执行。

接下来对 SHELL 作一些零散的介绍。

1. SHELL 实现

Linux 中有很多的 SHELL 实现，其中主要包括 sh 及 csh 两个分支。在 Linux 中比较普及的是 bash 。这里选择使用 mksh ，相对于 bash 来说，它更为轻量级，也更接近于 POSIX 标准。如果喜欢功能强大的 SHELL ，也可以考虑 zsh 。

通过 `chsh` 命令可以更换 shell 。

关于它的配置，可以参考这里： https://github.com/dram/dram-configs/blob/master/.mkshrc 。

这里在设置 `PATH` 时，将 `$HOME/bin/` 目录置于系统的命令路径之前，这样可以方便地对系统原有程序进行覆盖。通过 `which` 命令可以确认当前调用的是哪一个命令。

2. 常用命令

对于桌面应用来说，有几个常用命令需要掌握： top, ps, kill 。在 GNOME 或者 KDE 环境下也有对应的图形化工具。

第 10 章 其它

这一章主要介绍 Linux 桌面应用中的一些零碎技巧。

1. 待办事项

用于日程管理的软件有很多，这里参考 todo.txt(http://todotxt.com/)在 Vim 中实现了一个简易的基于文本的待办事项整理工具。

基本的想法是基于代码高亮进行筛选，基于代码折叠管理已经完成的任务。代码高亮可以在 `$HOME/.vimrc` 文件中加入下面的配置定义一个 :Hi 命令：

```
command -nargs=1 Hi :syn clear Search | syn match Search ".*<args>.*"
```

代码折叠可以直接通过在文本中结合使用 modeline 和标记完成。具体可以参考这个模板： https://github.com/dram/docs/blob/master/data/todo.txt 。

2. gtk icon theme

GTK 的 Icon 样式存放在 `/usr/share/icons/` 或 `$HOME/.icons/` 目录下。如果是 SVG 格式的，注意需要安装 librsvg 库，并执行下面的语句：

```
gdk-pixbuf-query-loaders > /usr/etc/gtk-2.0/gdk-pixbuf.loaders
```

可以通过 gtk-update-icon-cache 做一定优化。

3. 内核定制

在定制内核时首先需要考虑引导问题，其中包括磁盘 IO 相关驱动，文件系统相关驱动。这些驱动必须编译到内核中，而不是以模块的方式。不过现在有一个 initrd 文件系统，利用 mkinitrd 命令可以将需要的模块整合作为二次引导。

比如在 VirtualBox 中， Device Drivers->SCSI device support->SCSI (disk|CDROM|generic) 都应该编入内核。而 File systems 一块则可以根据需要选择。

3.1. 声卡驱动

在 Linux 中，声卡驱动主要有 OSS 和 ALSA 两种选择。而更为上层的应用框架则又有 Jack, PluseAudio 等。这里使用 OSS 作为声卡的驱动。

附录 A. Fedora 软件包管理

包管理对于一个发行版来说是非常重要的， Fedora 以 RPM 包的形式组织软件，主要通过 rpm 和 yum 这两个工具。其中 rpm 是包管理的基础程序， yum 是对 rpm 的包装，增加了包下载及依赖的自动处理等功能。

关于 Fedora 软件管理的详细介绍，可以查看 Feodra Software Management Guide [http://docs.fedoraproject.org] 。

Fedora 从 RedHat 发展而来，软件支持还是比较丰富的，除了官方的源之外，还可以到 rpmfusion [http://rpmfusion.org/] 安装官方没有收录的包。

在 这里 [http://pkgs.fedoraproject.org/gitweb/] 可以查看 Fedora 包的 spec 文件，不过一般不需要了解 RPM 包的描述方式。

Fedora 已带有图形化的工具，但下面还是介绍 rpm 和 yum 这两个命令行工具的使用，以此对 Fedora 的包管理机制有一个较深入的理解。

1. YUM 命令

`yum search pattern`
查找软件包

`yum install pkg-name`
从 YUM 源安装包

`yum downgrade pkg-name`
对包进行降级，如果更新后有问题时有用

`yum info pkg-name`
查看包信息

`yum update [pkg-name]`
更新软件包，如果不指定名称，更新系统所有包

`yum clean`
清空缓存

`yum list installed ['pattern']`
显示所有安装的包，类似于 `rpm -qa`

`yum provides file`
显示文件所属的包，类似于 `rpm -qf`

`yum list extras`
显示没有库中不包含的软件包

`yum deplist pkg-name`
显示依赖关系

ArchLinux 中，在利用 pacman 删除包时，可以通过 `-Rs` 参数同时删除自动安装的依赖包。而 yum 没有这个功能，可以在 `yum-utils` 包中找到 `package-cleanup` 命令，通过它的 `--leaves` 参数可以查找到那些已经不再需要的 lib 包。

在利用 yum update 系统时，可以通过 `-x` 参数指定不升级特定包。也可以添加到 `yum.conf` 的 `exclude` 中。

另外， yum 中还有 group 的概念，可以方便安装由官方分类好的一组包。包括以下几个命令：

```
yum groupinstall
yum groupupdate
yum grouplist
yum groupremove
yum groupinfo
```

2. YUM 配置

对于 YUM 的配置，其中软件源的配置是非常重要的一部分。 YUM 关于源的配置在 `/etc/yum.repos.d/` 目录中。该目录下的每一个文件针对一个软件库，像镜像等的设置都存放在对应软件库文件中。

如果需要禁用某个软件库，有两种方式，一种是删除对应的库配置文件或重命名为其它扩展名。也可以将其中的 `enabled` 配置项设为 `0` 。通过 `enabled` 设置的一个好处是，可以通过 yum 的 `--eanblerepo` 参数临时重新启用。 `yum repolist enabled` 可以显示所有启用的软件库。

针对每一个软件库， YUM 默认会从 http://mirrors.fedoraproject.org 下载一份镜像列表，然后随机地选择从哪一个镜像下载。这样有时效率并不高，因为各个源的访问速率可能会相差很大。可以通过对源进行相应的配置，让 YUM 优先选择一些源。

比如将 fedora 和 fedora-updates 这两个软件库的镜像设置为优先使用网易的镜像，可以在 `/etc/yum.repos.d/fedora.repo` 和 `fedora-updates.repo` 文件中添加 `failovermethod` 及 `baseurl` 设置:

```
failovermethod=priority
baseurl=http://mirrors.163.com/fedora/updates/$releasever/$basearch/
```

另外，也可以安装 `yum-plugin-fastestmirror` 包，让 yum 自动选择最快的镜像。

3. RPM

通过对 yum 的介绍，可以看到 yum 已经基本上可以完成软件管理的功能，一般可以不再直接使用 rpm 命令了，这里只列出几个 yum 无法完成的功能:

`rpm -ql pkg-name`
显示软件包中包含的所有文件

`rpm -ql -p path/to/rpm`
列出一个 rpm 包中的所有文件

实际上， yum-utils 中的 repoquery 可以用来完成这一功能。