Ubuntu 8.10字体美化原理初步探索

前言

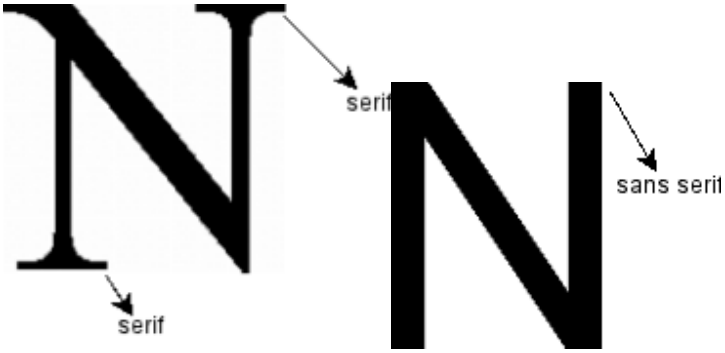
本人（Gary）对操作系统的美观要求颇高，因为没有一个看上去舒服的系统就很难激发学习使用的热情。而Linux跟Windows相比在字体显示的美观度方面（特指汉字）差距颇大。一来微软投入了大量的资源和财力开发出了一些特定的商业字库供Windows使用，而来Windows本身也是以易用性强著称，所以微软在背后也做了大量的工作让使用者安装好系统就能近乎完美的浏览各种文档和页面。即使默认没有安装某种字体或者字库，想要增加也仅仅靠点击几下鼠标就能达到目的，不可谓不是已经做到极致。老实说至少在这个方面不管哪个Linux的发行版都是比不上。

虽然如此，但是由于Linux自身功能的强大性和自由性，我坚信可以定制修改出一个让人满意的字体显示效果，但是要达到这个目的可不是点点鼠标就可以的，背后隐藏着很多的知识。对于像我这样未入门的人来说门槛有些高。虽然网上有很多美化的方法和步骤，可是在我尝试过后都不是那么令人满意。主要有这样几个方面的问题：1. 每个人的系统安装情况不同，你有安装过的包我不一定装了。你方便找到或者安装上的包对于我来说却可能非常困难；2. 命令繁杂且都没有任何说明，让人难以理解，特别是对于初学者；3.美化以后绝大部分总会留有美中不足，比如大部分都好看了，可是某某某应用程序还是不行之类的事情经常发生；4.即使幸运的美化好了让你重新帮别人再弄一次估计都很难重现，因为根本不懂原理或者理解很浅薄容易遗忘。我就是因为上面4点原因因此决心好好研究一下Linux系统下的字体系统，以我目前新装上的Ubuntu 8.10为蓝本在网上搜索了大量的相关文章撰写本文，只是来源琐碎，有一些来源链接丢失了，实在是不好意思。不过本文也融入了大量的个人理解，并且修正了很多的错误。大家可以随意转载，没有任何限制，也不用通知本人，只需要连同前言部分也一起转载就可以了。如有错误欢迎批评指正，以便及时修正，免得误人子弟。 如有问题需要探讨请发送邮件到<ghgary@sina.com>

预备知识

1.有衬线字体（Serif）、无衬线字体（Sans Serif)和等宽字体（Monospace）

初看到“衫线”这个词时候以为是衬衫，心想字体还分什么有衬衫无衬衫？仔细一看才发现看错了，在字典里面查了一下发现原来“衫线”是一个术语起源于荷兰语，指字母的拐角或端点位置的修饰线。（如下图所示）



我们平时所用的Times New Roman、Georgia等就属于有衫线字体(Serif)，而Arial、Tahoma、Verdana等则属于无衫线字体（Sans Serif）。对中文而言，同样存在这两大种类，很明显，宋体、细明体（繁体中常用）等就属于Serif，而黑体、幼圆等则属于Sans Serif。总结一下个人觉得有衫线字体（Serif）就是有棱角而且粗细不一的，无衫线字体（sans serif）就是相对比较圆滑而且粗细均匀的字体。正是由于这个特性使得无衬衫字体（Sans Serif）比较醒目，而且字体较小的时候依然容易辨认，而有衫线字体（Serif）由于强调笔画始末，因此前后字母看上去连续性更强，适合阅读，可是字号小了以后有些细的地方就变得更细显得模糊或者发虚，但是大的时候却更有型。因此各有利弊，不同场合应该选择不同的字体来显示，这样才更加美观，也使得本文更有意义。

上面这两种字体类别又叫比例字体（Proportional Font），另外还有一种字体叫做等宽字体（Monospace Font）。顾名思义就是字符宽度相等。在等宽字体中，字母 i,j 显得两侧餘白较多，

而字母 w,m 等的笔画显得相当拥挤。汉字因为都是方块字，基本上都是作为等宽字体在处理，但是有些字体同时还涵盖了西文的半角文字字符，造成一个字体中两种类型混杂的局面。

2.字体的DPI/PPI

我们的屏幕是由一个个小方格组成，我们称这些方格为像素，LCD 中有个术语叫屏幕的物理解析度（native resolution），指的的就是屏幕所包含的像素数量，比如我的 TFT LCD 14″物理解析度为1024x768，即水平由1024个像素组成，垂直由768个像素组成。通常让我们的 LCD 工作在物理解析度成像效果最好。DPI(Dot Per Inch)的意思就是每英寸的点阵数，通常适合于CRT显示器，而PPI（Pixel Per Inch）是每英寸的像素数针对LCD。

换算比例: 1 cm = 0.39 inch, 1 inch = 2.54 cm （以1024*768的LCD为例）

水平 dpi = 水平 resolution * 2.54 / width = 1024 * 2.54 / 28.7 = 90.6

垂直 dpi = 垂直 resolution * 2.54 / height = 768 * 2.54 / 21.5 = 90.7

我们知道显示器分辨率越高说明点阵越密集，从而相同DPI的字在高分辨率上会显得较小，这也是为什么我们有时候需要修改DPI的原因。由于跟分辨率相关所以DPI无法反映出汉字的实际大小，这里我们引入另外一个重要单位磅（point）。

1 point = 1/72 inch

比如一个9磅的汉字，9 * 1/72 = 1/8 inch = 1/8 * 2.54 cm = 3.175 mm。这是一个绝对值，所以在比较汉字大小的时候需要用磅而不是像素。

3.X11核心字体子系统、xft字体子系统

请注意这里说的是字体子系统而不是字体，因为这里要说的是一种渲染字体的流程模式。X里面有两种字体系统，一种是核心X11字体子系统，由X客户端发起向X服务器提出请求，然后X服务器利用内置的渲染引擎（默认是freetype）渲染并显示。另一种是xft字体子系统，它允许应用程序直接使用字体文件，自己渲染（通过X Font Server协调控制）然后交由X服务器来显示。X11核心字体子系统对于现在的Linux发行版来说并不常用，只是被一些老的GTK1.x程序或者一些终端程序沿用着。不过随着时间的推移和更为强大的功能，xft将会取代核心X11字体子系统。

X11核心字体系统的字体配置信息是被嵌入在/etc/X11/xorg.conf中的，而xft字体子系统来说，由于它使用fontconfig库来完成匹配的动作，所以对应的字体配置信息是在fontconfig的配置文件中。因此，当我们添加一个字体的时候需要同时修改两个配置文件才能让使用不同字体子系统的应用程序都能享用到。否则就会出现有的程序字体选择列表里有这个字体而有的则找不到。

这里有必要把xfs、xft以及fontconfig的关系进一步明确一下。实际上fontconfig和xft是两个不同的库，其中fontconfig负责找到某种字体，而xft负责在X系统中把fontconfig库找到的字体显示出来。当客户端像X服务器请求字体的时候，X服务器会将请求转发给xfs（X Font Server），然后xfs会利用xft库控制字体的匹配（通过fontconfig）、渲染（通过X协议调用渲染引擎）然后交还给X显示出来。

因此我们美化字体的时候对于xft子系统只需要跟fontconfig相关的配置文件打交道就可以了。

Fontconfig会自动搜索字体路径，发现新拷贝的字体，当需显示的字体不存在时，会自动选择其它字体代替显示。因此有时候我们需要调整该配置文件里面字体的排列顺序达到英文和汉字使用不同字体显示的目的。

目前，使用Qt3或GTK2工具包（包括KDE和GNOME应用程序）的应用程序使用xft字体子系统；其它部分程序(如 xmms 、gimp) 还在使用核心X字体。注意以上列出的字体子系统用法有两个例外，它们是 OpenOffice.org（使用它自己的字体绘制技术）和 Mozilla（使用 fontconfig，但不是 GTK 2）。正是上面不同的字体子系统以及各种不同类型的应用程序使得我们在对其美化的时候需要区别对待，这也是美化步骤繁杂且难以一次完美的主要原因。

4.Freetype

FreeType不是一个字体的格式，而是开源字体渲染引擎,它并不只为X设计。其功能就是读取Truetype字体信息, 如大小、分辨率、编码等, 然后渲染成所需的位图数据输出。Freetype 现在的版本是 2.x, 与1.0 相比, 最大的差别就是加入了抗锯齿功能。

5.点阵字体和矢量字体

点阵字库是把每一个汉字都分成16×16或24×24个点，然后用每个点的虚实来表示汉字的轮廓，常用来作为显示字库使用，这类点阵字库汉字最大的缺点是不能放大，一旦放大后就会发现文字边缘的锯齿。矢量字库保存的是对每一个汉字的描述信息，比如一个笔划的起始、终止坐标，半径、弧度等等。在显示、打印这一类字库时，要经过一系列的数学运算才能输出结果，但是这一类字库保存的汉字理论上可以被无限地放大，笔划轮廓仍然能保持圆滑，打印时使用的字库均为此类字库。不过要注意一点，当我们具体谈到某一个字体文件的时候，里面可能既有矢量字体信息又同时内置了对应的点阵字体信息。

6.TrueType、Type1和OpenType

这三种都是矢量字体，其中Type1字体由Adobe公司开发，从postscript演变而来。苹果和微软为了不受制于Adobe,自己开发了TrueType字体。这两者原理上没什么大区别,但是相互不兼容。后来adobe又开发了opentype字体，兼容前两种。因为Windows用的是TrueType字体，所以这种字体的资源丰富一些。把文件从windows平台直接拷贝过来就可以用了。我们一般美化矢量字体都是选择的TrueType，当然除了矢量字体有时候我们也需要点阵字体，原因是有些情况下点阵字体的效果也会比矢量字体来得好，这个需要具体原因具体分析，同时也需要不断尝试。

7.扩展名为ttf和ttc的字体文件

上面两种字体文件都是属于常见的TrueType字体类型文件，ttf叫做TrueType Font而ttc叫做TrueType Collection。顾名思义ttf才是TrueType字体的最小单位而ttc则是ttf的集合，就是说一个ttc文件里面可以包含多个ttf的集合。举个例子Windows上面的simsun.ttc文件实际上就包含了宋体和新宋体两个ttf，利用字体相关工具甚至可以将ttc里面的ttf分别提取出来或者将多个ttf合并成一个ttc。

8.反锯齿（Anti-aliasing）和微调（Hinting）

这两个术语是讲字体渲染（就是把矢量字体显示到屏幕上）时候用的一些技术。

在使用矢量字体的时候，我们会需要对汉字进行缩放。虽然理论上汉字的笔画是圆滑的，但是由于视觉的关系会产生锯齿感。Antialiasing是将字体在后台先以数倍的大小来绘画，然后再缩成想要的大小从而使字体的边角变得圆滑。Anti-aliasing会让人一种朦胧的感觉，习惯了Windows XP下清晰，锐利的字体显示时，Antialiasing会让人不太适应，会让用户觉得Linux下的字体显示不如Windows。但其实Antialiasing是一种很先进的显示技术，当长时间显示器上阅读的时候，Windows下的锐利的字体显示风格，会让眼睛更加容易疲劳。使用Antialiasing技术能够使得字体显示更加柔和，更加适合长时间的上阅读，减少眼睛的疲劳。 微软从Vista操作系统开始也采用具有类似渲染效果的ClearType字体来取代传统的黑白、锐利的英文字体显示和点阵汉字显示。

由于屏幕的像素有限，矢量字型的缩放需要有更多的考虑，例如当一条线位在两个像素格子中间时，该取左边的格子还是右边的格子? 如果这方面的控制没有做好，就常常会出现字型的衬线没有对齐，或是小字歪七扭八的情况。Hinting是内嵌在字体文件中的额外信息，它告诉渲染引擎该如何处理这些细节的部份，使得矢量字在小字的时候也能够好看。

因为中文的笔画繁多，在字号更小的情况下根本无法显示全部的笔画，这时候还需要设计师在不影响整体的情况下，对笔画进行取舍，去掉一些不影响识别的笔画，否则这个文字就会因糊成一团无法识别。 Hinting调整的范围需要涵盖各级小字号，一般最少要包括 9px — 18px 这个常用的字号区间。这种Hinting，即使是对于非常有经验的设计师，也是非常高难度而且费时费力的工作。

我们知道英文只有 26 个字母，但是对于中文的汉字情况就复杂的多了，仅仅是最常用的汉字就有6000 个，然后为了在简繁体混排时候能完美的显示，就必须同时包含繁体和简体两套字符，再加上众多的不常用但是会在古籍文献中非常重要的生僻字，一套比较完整的大字符集字库所包含的字符数目将接近 3 万个。仅仅是这矢量造字的工作就是非常浩大的。

这还不算，作为一套功能完整的正文字体，还需要考虑到斜体和粗体的显示。所有的斜体状态，也同样必须由设计师对不同的字号指定不同的Hinting，否则就会有显示问题，因此要开发一套优秀的中文大型字库，耗费的人力物力是惊人的。

安装字体

前言当中我们已经讨论过X中使用的两种字体子系统，那么如何将我们的新字体分别加入到里面去呢？注意，在这里我们先只讨论安装的问题，就是让我们的新字体能够被识别并加入到这两个字体子系统中可供应用程序选择使用。具体如何让不同类型的应用程序使用到并且尽可能为了完美而做的一些参数调整工作将是下节的目标。

为X11的核心字体子系统安装新字体

安装的步骤分为两步，第一个步当然是先拿到字体文件并且将其放入到我们的系统目录中。理论上说你把这些文件放在哪里都行，但是最好还是将新加入的字体文件跟原始的X11核心字体集中放置比较好。字体文件本身可以网上下载也可以从你的windows系统中间直接拷贝过来使用。

1.安装点阵字体

X同时支持跨平台的点阵字体格式BDF和效率更为高效一些的PCF格式。一般来说点阵字体是以BDF格式发布的，所以你可以用bdftopcf命令将其转换成PCF后再使用，当然你也可以不必这么做。当转换完成以后，你也可以用gzip来压缩一下使其变成.pcf.gz，然后再拷贝到任意的一个目录中，比如/usr/local/share/fonts/bitmap/，然后切换到该目录下用mkfontdir命令生成fonts.dir文件。这个命令会扫描当前目录下的所有文件，自动识别出里面的字体并且在分析之后给每一个字体都取一个XLFD字体名（这个名字很长包含了各种相关信息，比如制造商、字体字号、解析度、字间距、字符集等）和其它相关信息或者参数（比如所属的文件名、是否倾斜、是否加粗等等）这个fonts.dir就像该目录下所有字体文件的索引数据库一样，通过它能够获得该目录下所有字体的信息并且定位到对应的文件。

准备好这些以后下一步就是要让X知道这个目录以及这个索引文件的存在了。有两种方式一种是临时性的，以Session为单位，利用xset命令。不是很常用，所以就不赘述。另外一种是永久性的，将对应的路径信息加入到/etc/X11/xorg.conf的Section “Files”中即可。类似如下这样：

```
Section “Files”
        FontPath "/usr/local/fonts/Type1"
        ...
        FontPath "/usr/local/fonts/bitmap"
End Section
```

不过在Ubuntu8.10中似乎已经找不到对应的Section ”Files”了，我查阅了相关的资料，似乎Ubuntu 8.10减少了对xorg.conf的依赖,对于硬件设备这一块尽可能都自动探测和设置了，有兴趣的可以去研究一下HAL，Ubuntu8.10是利用fdi配置文件而非xorg.conf了。对于FontPath，原来的那些核心字体路径都从xorg.conf中移除了，但是X11仍然是支持的（从log文件可以看出来）而且我手工在xorg.conf做过实验，从log文件可以看出来把字体加入到X11核心字体子系统依然是通过上面说的这个方法。

2.安装矢量字体

矢量字体的安装跟点阵字体的安装非常相似。主要步骤也是先准备好字体文件，然后拷贝到某个目录中然后生成索引，最后把该目录路径加入到xorg.conf中去就可以了。但是要注意一点区别，在生成索引的时候，针对点阵字体我们是用的mkfontdir生成了fonts.dir。但是对于矢量字体我们必须先用mkfontscale（不需要任何参数，直接切换到包含字体的目录打入运行）生成fonts.scale，然后再运行mkfontdir生成fonts.dir索引文件。之所以需要这样是因为mkfontdir不能直接从矢量字体文件生成对应的索引文件，而mkfontscale可以。Mkfontdir运行以后会把mkfontscale生成的过渡性文件fonts.scale变成fonts.dir。如果该目录里面只有矢量字体，那么fonts.dir跟fonts.scale是完全一样的，你也可通过改名将fonts.scale变成fonts.dir，但是如果矢量点阵同时存在，我想这两个文件一定是有区别的，估计fonts.scale里面只有矢量字体的信息，fonts.dir是包括点阵和矢量字体的。

为X11的xft字体子系统安装新字体

相较于前面而言，为xft字体子系统安装新字体要容易一些。只需要把对应的字体文件拷贝到指定字体目录中并且让系统重建一下字体列表就可以了。我们前面说过xft字体子系统是通过fontconfig来搜索匹配字体的。Fontconfg会去从/usr/X11R6/lib/X11/lib/fonts/以及~/.fonts目录下搜索新的字体文件。其实如果你要重新随便再建一个路径也是可以的，那你就要去修改fontconfig的配置文件。不过个人觉得没有这个必要，字体文件放在一块比较好。因此把对应的文件拷贝到上面任何一个目录中就可以了。Fontconfig会在下一个合适的时机发现这些新字体并且更新它自己的字体列表索引。不过你也可以用命令fc-cache来强制fontconfig现在就做这个动作，这样立马所有利用xft字体子系统的软件都能够在字体选择中找到这些新加入的字体格式。另外你可以利用fc-list来查看fontconfig的字体列表。

应用和调整字体

在我们把点阵或者矢量字体争正确安装之后只是完成了第一部分。通过安装，系统以及应用软件都已经能够有能力来使用这些字体了，可是有能力不等于已经施展了该能力。同时字体文件本身并不是十分完善，比如某些字体显示英文好看，某些字体显示中文好看，或者还有其它的一些个性需求，所以需要在字体配置文件中做一些手工的调整达到我们的目的。

关于字体配置文件

核心X11字体子系统通常都是gtk1.x的应用程序在使用，对应的字体配置信息是在/etc/gtk/gtkrc中设置和调整的。对于gtk2.x的应用程序则是在/etc/gtk2.0/gtkrc中。（请注意具体的文件跟你所处的环境有关，比如如果你的locale是zh_CN，那么对应的文件可能是gtkrc.zh_CN）另外gtk2.0的应用程序如果要使用xft需设置环境变量GDK_USE_XFT=1，否则也是使用的核心X11字体。不过要注意的是如果找不到以上路径可能该发行版已经不再支持或者不推荐使用了。毕竟是已经过时的了，比如在我的Ubuntu8.10里面就已经找不到/etc/gtk这个目录了。对于gtk2.2以上的Gnome应用程序或者基于qt的KDE应用程序都是默认使用fontconfig来配置字体的。fontconfig有三个配置文件分别是/etc/fonts/fonts.conf, /etc/fonts/local.conf 以及~/.fonts/.fonts.conf。要注意最好针对你的系统man fontconfig具体查看一下，一般来说/etc/fonts/fonts.conf是全局的配置文件，~/.fonts/.fonts.conf是本地用户的。至于那个local.conf有的发行版已经去掉了，但是如果你添加上仍然支持。所以请一定man一下。我个人的建议是尽可能把对字体配置的修改在~/.fonts/.fonts.conf中进行。该文件是被全局的/etc/fonts/fonts.conf文件include在内的，所以效果是一样的。如果你有多个配置文件都要被使用，你可以在~/.fonts/.fonts.conf里面再分别include进来。因为在全局配置文件中修改有可能在升级之后该文件就被替换掉了，所以请不要像有些美化教程说的那样去修改局配置文件/etc/fonts/fonts.conf。

正是因为应用程序类别有分这么多种，而且对应调整字体的配置文件又有好几个，所以在某某应用程序中没法找到某个自以为已经成功安装了的字体，或者已经修改了配置文件但是没有效果的情况经常被初学者碰到。请在碰到类似情况的时候先确认该应用程序是基于什么开发的，是Gtk1.x还是Gtk2.0或者Gtk2.2以上的，还是Qt？字体文件是否针对以上类型正确安装到了对应的字体子系统中？（核心X11？xft？）并且还要确保系统已经识别并刷新过了。最后要确认所修改的字体配置文件是否跟该应用程序的类型所匹配。不要在网上搜索出来某某大侠修改了fonts.conf使得宋体可用了，你就依葫芦画瓢想用同样的方式让你的xmms也能使用宋体。这是不可能的，因为xmms是gtk1.x的，它是使用核心字体的，宋体添加到xft中是没办法让xmms找到的更别说使用了，而且对应的配置信息应该是/etc/gtk/gtkrc。

如何读懂并能自己修改配置文件

任何 fontconfig 配置文件，都被包含在 XML 结构之中：（不要跟我说不知道什么是XML，如果真不知道那就是普通的文本文件，但是具有某种特殊逻辑格式）

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<fontconfig>
...
```

</fontconfig>

我们之前提到过，应用程序在需要字体的时候会向xft来请求，xft会让fontconfig来匹配寻找到系统里面最接近该需求的字体，然后才由渲染模块完成相应动作后交由X去显示出来。也就是说fontconfig将会有一个输入，而且不管找到与否都会有一个输出。在这里我们可以简单的理解这个输入其实就是一个字体属性列表，而这个输出就是系统能提供的最接近的字体属性列表。既然fontconfig有输入，所以为了达到我们的特殊目的，使得匹配能够按照我们的需求来进行就需要对这个输入先做一些修改然后再继续。再举个例子，比如系统在匹配过程中可能需要和系统内部的字体列表逐一去比较，一旦找到符合要求的就返回不再继续了，但是有可能这个匹配从人的视觉角度来说效果并不好，这个时候我们就需要让效果好的那个字体先被匹配并且命中返回，因此可能需要对匹配的顺序做一些调成。刚才我们说了fontconfig的输出也是一个字体属性，为了使显示效果好，我们对这个输出也需要修改，比如在某些特定尺寸关闭字体的反锯齿使用点阵（矢量字体文件里面有些也内嵌某些字号的点阵）等等，所有这些动作都可以在配置文件中做到。

我们不妨用windows的显示效果做为我们的目标，毕竟微软耗费了大量的人力物力才使得界面字体如此美观，而且也适应了，因此先看看windows里面是怎么做到的。

字体名称	版本	反锯齿（AA）	微调（Hinting）	AA + Hinting
Andale Mono	2.00	0-6	7-20	21+
Arial	2.82	0-6	7-13	14+
Arial Black	2.35	0-6	7-12	13+
Arial Bold	2.82	0-6	7-8	9+
Arial Bold Italic	2.82	0-6	7-8	9+
Arial Italic	2.82	0-6	7-13	14+
Comic Sans MS	2.10	0-6	7-11	12+
Comic Sans MS Bold	2.10	0-6	7-8	9+
Courier New	2.82	0-6	7-27	28+
Courier New Bold	2.82	0-6	7-11	12+
Courier New Bold Italic	2.82	0-6	7-12	13+
Courier New Italic	2.82	0-6	7-27	28+
Georgia	2.05	0-6	7-12	13+
Georgia Bold	2.05	0-6	7-12	13+
Georgia Bold Italic	2.05	0-6	7-12	13+
Georgia Italic	2.05	0-6	7-12	13+
Impact	2.35	0-6	7-16	17+
Times New Roman	2.82	0-6	7-13	14+
Times New Roman Bold	2.82	0-6	7-10	11+
Times New Roman Bold Italic	2.82	0-6	7-13	14+
Times New Roman Italic	2.82	0-6	7-15	16+
Trebuchet MS	1.22	0-6	7-12	13+
Trebuchet MS Bold	1.22	0-6	7-8	9+
Trebuchet MS Bold Italic	1.22	0-6	7-8	9+
Trebuchet MS Italic	1.22	0-6	7-12	13+
Verdana	2.35	0-6	7-12	13+
Verdana Bold	2.35	0-6	7-12	13+
Verdana Bold Italic	2.35	0-6	7-12	13+
Verdana Italic	2.35	0-6	7-12	13+
Webdings	1.03	0-6	7-21	22-1536 1537+
Tahoma	3.09	0-6	7-12	13+
Tahoma Bold	3.09	0-6	7-12	13+

SimSun	3.03	0-6		7-18			19+	
MingLiU	5.03	0-6		7-36			37+	
Arial Unicode MS	1.01	0-6		7-18			19+	
MS 雅黑	0.72	0-6		7-17			18+	
MS 雅黑 Bold	0.72	0-6	7-13			14+		
Segoe UI	0.98	0-6		7-14			15+	
Segoe UI Bold	0.98	0-6		7-14			15+	
Segoe UI Italic	0.96	0-6		7-14			15+	
Segoe UI Bold Italic	0.95	0-6		7-14			15+	

这里你或许要问难道对于每个不同的字体（黑体，斜体，黑斜体都算是不同字体）都要一个个去定制，对每个不同的字号大小分别告诉系统该不该打开AA，该不该微调？对！就是如此。而且还不仅如此，我之前说过了，某些情况下我们使用矢量字体时还需要使用内嵌的点阵字体而不是去缩放。如此大量的配置信息这要写到什么时候去？而且还要会写才行啊。

原理基本都讲完了，剩下的就是如何操作了，你可以去详细阅读一下fontconfig的man page。不过为了节省时间可以参考quanliking兄的一篇帖子，他帮我们把这些配置都写好了，实在是感激。请大家参阅该链接，弄懂了的部分可以节省时间直接跳过了，个人推荐从：“三、字体配置”开始往下继续，别忘了表示支持一下顶一下本帖哦，当然还有quanliking兄的，呵呵：
参考链接：http://www.linuxsir.org/bbs/showthread.php?t=266659