用 fontconfig 治理 Linux 中的字体

catcat.cc/post/2021-03-07

2021年03月07日

在上一篇《Linux fontconfig 的字体匹配机制》当中，我们已经了解过了 fontconfig 的原理。没想到一晃过去了 4 个月之久，别忘了还缺少实践部分呢！现在就一起来实践一把 fontconfig 的配置吧。

混乱的现状

我浏览过一些介绍 fontconfig 配置的文章，要么是配置难于理解和维护，要么是复制粘贴不求甚解。所以我现在的目标是，找出一个最简单的方案，同时还要把各类问题处理好。

配置路径

有一些说法，号称直接修改 `/etc/fonts/` 下的配置文件最为简单。可千万别这么做！**如果改动过，请恢复原始文件。**正常来讲，应该把我们的配置文件加入家目录下的 `~/.config/fontconfig/` ，不要添加在系统端。为了覆盖系统端的配置，可以在用户目录下的 fontconfig 配置中使用 **强绑定** 规则。关于强弱绑定规则对字体配置的影响效果，我在上一篇文章里的小节 字体得分和强弱绑定 当中介绍过。

使用强绑定后，系统端的所有配置不会再干扰我们的配置。系统端的配置的作用是为了能够在我们自己的配置不够用的情况下，依旧能够提供一些有效的字体。或者说，它们的作用是仅作为最后的 fallback 机制来使用。当然，前提是我们一定要在用户配置中使用强绑定规则，切记。

关于合成字体

fontconfig 的字体匹配机制为 Linux 程序提供了字体 fallback 列表，按先后顺序依次渲染字体。这意味着什么？不再需要那些合成字体了。在 Linux 上，你无需使用更纱黑体；为了用上 Nerd Fonts 图标字体，你也不需要给字体打上 Nerd fonts 补丁；当你想让中文字体整合另一个英文字体，不需要把它们合成一个字体。甚至你想让某个字体的某个字形由另一个字体来提供，这些需求统统都可以通过配置 fontconfig 来实现。

一个最简单的思路，本文也基于该思路：不让程序使用某个具体的字体，而是使用通用字体族名 (Generic Font Family)。比如，让程序使用 sans-serif，也就是默认的无衬线字体。然后，使用 fontconfig 配置：

```
<match target="pattern">
  <test name="family">
    <string>sans-serif</string>
  </test>
  <edit name="family" mode="prepend" binding="strong">
    <string>Noto Sans CJK SC</string>
    <string>Noto Sans</string>
    <string>Twemoji</string>
  </edit>
</match>
```

这种 font stack 的方式，即可让程序按照以下顺序渲染字体：

```
Noto Sans CJK SC —> Noto Sans -> Twemoji
```

我在此优先指定中英文使用 Noto Sans CJK SC 来渲染，一些西文使用 Noto Sans 渲染，emoij 使用了字体 Twemoji。那么还有很多没指定的字形怎么办呢，比如腓尼基字母、皇家亚兰字母？没关系，那些都是在系统端的配置中解决的，我只要安装了相应字体，fontconfig 自然能帮我找到它们。同时，即使我系统里有多个中文字体，我也能保证程序使用 Noto Sans CJK SC 渲染字体而不是别的中文字体。

上一句话是骗人的。😕

fontconfig 的解决方案不是完美的，合成字体依旧有它们的价值。这体现在一些对 fontconfig 支持不良的 Linux 程序上。对于 Windows 用户来讲，合成字体是一个很好的解决方案 (因为他们没有 fontconfig)。尽管如此，我也强烈建议 Linux 用户配置好 fontconfig，而不是从一开始就使用合成字体。使用 fontconfig 实现字体的 fallback 机制，相比合成字体的优势是明显的：

自由组合，灵活搭配。

每次升级合成字体，你都需要重新打补丁生成该合成字体。 如果是别人帮你提供好的合成字体那还好说， 从网上下载、从软件仓库安装就完事了。比如说，为了能在西文字体中使用来自 Nerd fonts 的字形，看一看 archlinuxcn 社区仓库打包了多少份 Nerd Fonts：

```
# 显示前 10 个基于 Nerd 的字体
# pacman -Ssq nerd-fonts | head -10
ttf-nerd-fonts-symbols
ttf-nerd-fonts-symbols-mono
nerd-fonts-3270
nerd-fonts-agave
nerd-fonts-anonymous-pro
nerd-fonts-arimo
nerd-fonts-aurulent-sans-mono
nerd-fonts-bigblue-terminal
nerd-fonts-bitstream-vera-sans-mono
nerd-fonts-cascadia-code
# 统计 Nerd 合成字体的总数
# pacman -Sl archlinuxcn | grep nerd-fonts | wc -l
51
```

为了能在字体里用上 Nerd fonts 图标，他们给 51 个字体都打上了来自 Nerd fonts 的补丁！

当我们再想使用某个中文字体时，比如同时搭配思源黑体、anonymous-pro、Nerd font，岂不是要合成 source-han-sans-otc-fonts-nerd-fonts-anonymous-pro？对了，别忘了还可以在合成字体里加入 emoji 字体。

而 fontconfig 的缺点又是什么呢？后文会解释为什么 fontconfig 不能百分百命中想要的字体，特别是在两个浏览器 Chrome 和 Firefox 当中。

关于网页

为了能在浏览器上显示想要的字体，不少人通过浏览器的扩展插件引入第三方的 CSS 样式，强制覆盖原网页的字体样式。这样做可能会破坏网页设计。因为有些网页是需要引入外部字体实现一些特殊效果的，这些字体的效果不依赖系统端的字体。但如果强制修改 CSS font-family 的话，就没办法使用这些字体了。除非完全不想关心网页设计师的想法，否则弊端比好处多。

没必要引入 CSS 强制设置字体。对于网页的字体规则，浏览器也同样遵守 fontconfig 配置！所以我们可以通过 fontconfig 控制网页字体。

现如今大部分网页的 CSS font-family 都是一个长长的串，里面包含了好多字体族名。但它一定会在这个串的最后面，包含一个通用字体族名。我们的 fontconfig 配置，就像上文中的配置那样，是在这个通用字体族名上进行操作的。

解决问题

既然 Linux 有 fontconfig 这么好用的机制，不代表 fontconfig 能很好的解决所有问题。这里有一些历史遗留问题，想解决它们好不是件容易的事情。

全角引号

英文中的单引号有两种，`'` (U+0027) 和 `’` (U+2019)。可能会觉得前一种出现在英文文本中，后一种出现在中文文本中，并且宽度和中文等宽。然而，英语世界中的一些人在英文文章中也是会使用后一种的。所以，字体在显示后一种引号的时候，究竟是和英文字母一样窄，还是该和中文字体等宽呢？如果在英文文章中显示成全宽字符则会显得突兀。

比如，思源黑体其实也包含了拉丁字母的字形，可以完全使用思源黑体去显示西文。此时，后一种单引号的宽度会显示成中文字形的宽度。导致即使在全英文环境中，单引号也会突兀地在文本中过宽。

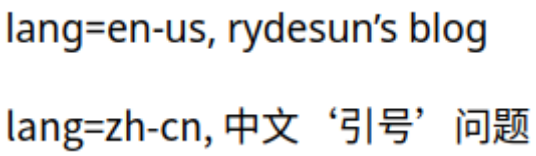
同样地，`‘` (U+2018) `“` (U+201C) `”` (U+201D) 也有类似的情况。

noto-cjk 官方项目在其 issue 上讨论了这个问题：Closed Full width punctuations U+2018, U+2019, U+201C and U+201D in Chinese。在维基百科上也有相关内容：撇号。

我们的目标是在不同语言环境下字形会有所区别，让引号只在中文文本中全宽。用该 HTML 片段测试字体

```
<div style="font-family:sans" >
  <p lang="en-za">lang=en-us, rydesun’s blog</p>
  <p lang="zh-cn">lang=zh-cn, 中文‘引号’问题</p>
</div>
```

用你的浏览器在线测试。效果图：



所以，在没有有效配置的情况下，当使用中文字体去显示英文文本时，会错误地提供全宽引号。更纱黑体提供了一个折衷的办法，分别提供两个字体族名，Sarasa Gothic 使用全角引号，Sarasa UI 使用和字母一个宽度的引号。你可以根据实际需要选择字体族名。来自 Sarasa Gothic 的 README：

```
Style dimension
    Latin/Greek/Cyrillic character set being Inter
        Quotes (“”) are full width —— Gothic
        Quotes (“”) are narrow —— UI
```

由于 fontconfig 是可以根据语言来灵活选择字体的，在后文讲 fontconfig 配置的时候，我们再来处理这个问题。

异体字

我在这里说的异体字，可不是简体和繁体的区别。Noto Sans CJK 中的异体字，是在 **相同的** Unicode 码位下，不同的语言会使用不同的字形。这可不是「卫」和「衛」的区别，也不是诸如中文的「传」和日语汉字的「伝」的区别，因为它们本来就分属于不同的 Unicode 码位。而是说，同一个 Unicode 码位的汉字在字形上有微妙的不同。

没错，这类似于上文说到的引号问题：占据同一个 Unicode 码位，但却可以有不同字形。

用该 HTML 片段测试字体，在不同语言环境下字形会有所区别。

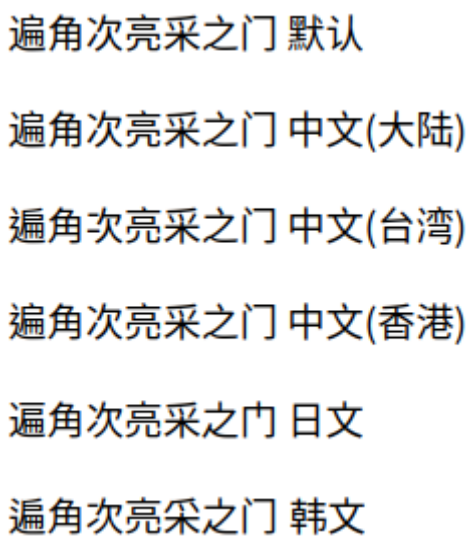
```
<div style="font-family:sans">
  <p>遍角次亮采之门 默认</p>
  <p lang="zh-cn">遍角次亮采之门 中文(大陆) lang=zh-cn</p>
  <p lang="zh-tw">遍角次亮采之门 中文(台湾) lang=zh-tw</p>
  <p lang="zh-hk">遍角次亮采之门 中文(香港) lang=zh-hk</p>
  <p lang="ja">遍角次亮采之门 日文 lang=ja</p>
  <p lang="ko">遍角次亮采之门 韩文 lang=ko</p>
</div>
```

用你的浏览器在线测试。效果图：

如果你的浏览器显示的效果和上面的效果图不一致，并且「默认」那一行 (默认的意思是没有指定语言的情况)不是你想要的默认字形，可能是因为缺少该字体或者没有成功配置 fontconfig。

Arch Linux 用户注意了！如果只选择安装 adobe-source-han-sans-**cn**-fonts (注意是 cn)，是无法正确显示异体字的，因为它是思源黑体的一个子集 (subset)，只有中国大陆字形，去掉了上面说到的异体字。当然，也仅仅是失去了少数这些特殊的异体字罢了，繁体字、日文汉字还是正常包含的。

其实大部分中文字体都没有该功能。你可能好奇思源黑体是怎么做到的。其实这是一个字体特性，字体文件会在内部携带 GSUB 表 (Glyph Substitution Table)，实现从 locl GSUB 中根据特定语言查询到对应的字形。完整版的思源黑体是具备该特性的。所以想要包含上文提到的那些异体字，应该选择安装 adobe-source-han-sans-**otc**-fonts 或者 noto-sans-cjk。



既然如此，为什么还会存在 adobe-source-han-sans-**cn**-fonts (包名带 cn 的那个版本)？那是给完全不关心这些异体字的用户推出的版本，在种语言环境中只显示中国大陆字形。当然大部分繁体字和日文汉字还是正常显示的，毕竟占据相同 Unicode 码位的异体字只占很小一部分。

然而真实情况是，不少人安装 adobe-source-han-sans-**otc**-fonts 或者 noto-sans-cjk 后，抱怨一些字形怎么显示异常呀？然后被告知安装 adobe-source-han-sans-**cn**-fonts 后就正常了。当然会显示正常了，因为把异体字给丢掉了嘛！无论是 Adobe 的 adobe-source-han-sans-**otc**-fonts 还是 Google 的 noto-sans-cjk，默认使用日文异体字。但对于中文用户来说，默认显示日文异体字太奇怪了。

其实完全可以在保留异体字的情况下，让它默认显示中国大陆字形；只在特定语言下显示异体字，比如用浏览器查看一个日文页面。所以需要正确配置 fontconfig，这也是本文的目的之一。在后文的 思源字体 一节，我们再来看看思源黑体的版本。

中文字体的西文部分

如果去观察 Noto Sans CJK 这个中文字体，会发现它的西文部分的字形其实和 Noto Sans 不一样，虽然它们都以 Noto 自称。

除了上面说到的引号问题，中文字体中还存在什么问题呢？那就是中文字体携带的英文字符有可能十分糟糕，特别是 Windows 自带的 SimHei，也就是中易黑体，它的英文相当糟糕。另外，微软雅黑的字重实在是太少了，对于设计师来说很不友好。而各种流行的英文字体支持很多字重。

可以注意到著名的 bootstrap 也遭遇到这个问题，当他们决定在 font-family 中使用 system-ui，后来又不得不去掉 system-ui：Do not let system-ui come before “Segoe UI” in font-family—orremove it from the font stack altogether。因为 system-ui 这个 CSS font-family 值会让浏览器直接调用系统界面所使用的字体去渲染。但对于 Windows 简中用户来说，英文字体渲染这块使用的是微软雅黑，甚至可能是宋体 (中易宋体)，这是网页设计师不愿意看到的结果。它会让简体中文用户看到非预期的、渲染糟糕的英文字符。虽然在 Linux 上我们没这个烦恼，但是想改动它的话当然是能做到的，这对 fontconfig 来说小事一桩，详见后文 第四步：覆盖西文字体。

字体比较与选择

习惯上把中文字体的黑体和无衬线归为一类，把宋体和衬线归为一类。至于等宽字体，是出于排版的需要，规定每个字符是全宽或者半宽，没有规定是否必有衬线。可以选择宋体和衬线西文作为等宽字体，也可以选择黑体和无衬线西文。只要能让西文等宽，并且宽度是中文字符的一半 (HW) 即可。通常使用黑体中文和无衬线西文较为常见。

宋体，日语称为明朝体，Mincho；港台称为明體。所以它们是一个意思。

黑体，在日语中称为 ゴシック 或者 Gothic。注意它不是哥特式风格的意思。

知乎：为什么日本人称『黑体』为『哥特体』？

编程字体

对于经常写程序的人来说，会选择一款合适的等宽字体作为编程字体和虚拟终端下的字体。有太多可供选择了，Source Code Pro，Consolas，Menlo……还有像 Fira Code 这样以连字 (ligature) 特性为主的编程字体。

我比较推崇 Iosevka，它可不是一个普通的等宽字体，而是一个高度可定制化的字体，你可以自己微调字形，再进行编译，从而制作出独特的编程字体，不用再眼馋其他等宽字体的特性了。可以看项目 README 中的 Customized Build 一节，提供了丰富的自定义项。比如我自己的 rydesun/iosevka-custom，在文件 `private-build-plans.toml` 中配置字体，再用 `makepkg` 编译打包成属于我自己的 Arch 包。像我就关闭了连字特性，我不喜欢在代码中看到连字。

另外，你可能还会想使用图标字体。在虚拟终端和一些系统界面 (窗管插件栏) 中使用图标字体是一个很好的想法。有好些个值得推荐的图标字体，比如 Font Awesome，Ionicons。而流行的 Nerd Fonts 则整合了好多个图标字体。对我来说，vim-devicons 和它的搭配是必不可少的，两个项目都来自同一个作者。Arch 用户可以安装 ttf-nerd-fonts-symbols 或者 ttf-nerd-fonts-symbols-mono 就能拥有它。

思源字体

思源字体到底有多少个变体呀？现在让我们来捋一捋 😃

就以 Arch Linux 仓库和 AUR 为例。上游拆分了多少字体，Arch Linux 仓库里就打包了多少个思源黑体：

```
adobe-source-han-sans-cn-fonts
adobe-source-han-sans-hk-fonts
adobe-source-han-sans-jp-fonts
adobe-source-han-sans-kr-fonts
adobe-source-han-sans-tw-fonts
```

好家伙，每种语言来一份。其实它们对应了官方项目 README 中 Region-specific Subset OTFs 这一节所说的版本。

这还只是黑体部分，再来看看到宋体，adobe-source-han-**serif-otc**-fonts 和它的 subset (小提示：思源宋体没有专门制定 HK 版本。)

```
adobe-source-han-serif-cn-fonts
adobe-source-han-serif-jp-fonts
adobe-source-han-serif-kr-fonts
adobe-source-han-serif-tw-fonts
```

等宽字体呢？

其中在 adobe-source-han-**sans** 当中，已经包含了等宽字体家族名，它是以 HW 结尾的，代表西文字符是中文一半的宽度。至于 AUR 和 archlinuxcn 社区仓库打包的 adobe-source-han-**mono-otc**-fonts，以及它的每种语言的 subset (又开始重复了)

```
adobe-source-han-mono-cn-fonts
adobe-source-han-mono-hk-fonts
adobe-source-han-mono-jp-fonts
adobe-source-han-mono-kr-fonts
adobe-source-han-mono-tw-fonts
```

上游来自 adobe-fonts/source-han-**mono**，这是另外一个项目，纯属 Adobe 员工的个人项目 (作者的解释)，它是把西文部分替换成了 Source Code Pro。既然前面提到，adobe-source-han-**sans** 包中是带有 HW 这类等宽变体的，为什么还要有带有 Source Code Pro 这个版本呢？显然是因为喜欢合成字体的人们。在有 fontconfig 的情况下这是不必要的，因为我们可以用 fontconfig 制定替换规则。

如果你不关心异体字，那么偷懒的做法是选择 adobe-source-han-sans-cn-fonts，它来自官方 README 小节 Region-specific Subset OTFs。因为是 subset，所以想要完整版的话，就要选择 adobe-source-han-sans-**otc**-fonts，它来自官方 README 小节 OTCs。其实在没有 fontconfig 的情况下，特别是 Windows 上，应该选择 Language-specific OTFs 一节的版本，它同样包含完整版的异体字，但默认语言不是日文，而是让我们选择。可惜 Arch Linux 社区仓库没有打包它们，所以我们需要在 OTCs 版本上，用 fontconfig 设置默认语言版本。至于那个 Super OTC,它是把多个字重再合并到一个文件中的「便携版」，必要性不大，所以社区仓库也没有打包它。以上说的是思源黑体。思源宋体同理，不再重复阐述。

更纱黑体

更纱黑体是由思源黑体和西文字体 Iosevka 整合而来的，或者说，它把思源黑体的西文部分替换成了 Iosevka 这个字体。更纱黑体的作者就是 Iosevka 的作者；该项目的 README 第一句话就指明了：

```
This is SARASA GOTHIC,
a CJK programming font based on Iosevka and Source Han Sans.
```

由于思源黑体的西文部分不是很全面，所以更纱黑体也算是扩充了一部分字形。另外，更纱黑体在思源黑体的中文字形上做了一些调整。我不是很清楚它具体做了哪方面的调整。但我认为没必要使用更纱黑体，起码在 Linux 上没这个必要，想要整合多个字体靠 fontconfig 就可以做到。

我的选择

- 无衬线：西文 Noto Sans，中文 Noto Sans CJK
- 衬线：西文 Noto Serif，中文 Noto Serif CJK
- 等宽：西文 Iosevka，中文 Noto Sans Mono CJK

西文字体 Noto Sans / Noto Serif 来自于软件包 `noto-fonts` ，中文字体 Noto Sans CJK / Noto Serif CJK / Noto Sans Mono CJK 来自于软件包 `noto-fonts-cjk` 。 Arch Linux 上只装这两个包就行了，就足以覆盖世界上绝大部分的字形了。由于 Noto CJK 是 Google 对思源字体的 rebrand，它们是完全一样的，所以我在这里没有选择以 Adobe 版本为主打包的方式。

关于 emoji，我选择了 Twitter 推出的字体 Twemoji。

    ttf-twemoji

另外，少不了人见人爱的图标字体 Nerd Fonts。

    ttf-nerd-fonts-symbols

基本思路

絮絮叨叨半篇文章，fontconfig 的配置究竟长什么样呀！

现在终于可以进入正式环节了，但我却想把这部分内容压缩得尽可能短，毕竟越简单的配置越容易理解和维护。这里还是建议先看完上一篇文章，因为一些原理，本文不会再重复。如果想深入到 fontconfig 的配置中去，最好是先了解它。

基本思路只有一个，设置默认字体为通用字体族名。

不要在程序设置中指定使用思源黑体或者更纱黑体等等字体，毕竟你一次只能指定一个字体。合理的做法是在设置中指定程序使用通用字体族名：sans-serif，serif，monospace。这样做会让程序使用 fontconfig 提供的 fallback 列表，也就是 font pattern，里面包含我们想要优先使用的所有字体。比如在虚拟终端的配置使用 monospace，而不是某个具体的等宽字体。大部分虚拟终端本来默认就是使用 monospace 的，所以你无需修改它。如果修改过，请复原默认值。

姑且认为这是第零步吧。

最简单的配置文件路径是 `~/.config/fontconfig/fonts.conf` ，我们的配置添加到这个文件中。

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

<!-- 实际内容 -->

</fontconfig>
```

实际配置内容被包裹在其中。我在 gist 上提供了一份 完整的示例配置，作为参考。

第一步：设置默认字体

```
<!-- Default system-ui fonts -->
<match target="pattern">
  <test name="family">
    <string>system-ui</string>
  </test>
  <edit name="family" mode="prepend" binding="strong">
    <string>sans-serif</string>
  </edit>
</match>

<!-- Default sans-serif fonts-->
<match target="pattern">
  <test name="family">
    <string>sans-serif</string>
  </test>
  <edit name="family" mode="prepend" binding="strong">
    <string>Noto Sans CJK SC</string>
    <string>Noto Sans</string>
    <string>Twemoji</string>
  </edit>
</match>

<!-- Default serif fonts-->
<match target="pattern">
  <test name="family">
    <string>serif</string>
  </test>
  <edit name="family" mode="prepend" binding="strong">
    <string>Noto Serif CJK SC</string>
    <string>Noto Serif</string>
    <string>Twemoji</string>
  </edit>
</match>

<!-- Default monospace fonts-->
<match target="pattern">
  <test name="family">
    <string>monospace</string>
  </test>
  <edit name="family" mode="prepend" binding="strong">
    <string>Noto Sans Mono CJK SC</string>
    <string>Symbols Nerd Font</string>
    <string>Twemoji</string>
  </edit>
</match>
```

对 system-ui，sans-serif，serif，monospace 设置优先显示的字体。在这里我让 system-ui 默认为无衬线。注意，system-ui 必须在最前。由于 fontconfig 对 font pattern 的操作是按顺序执行的，所以必须先让 system-ui 能优先以 sans-serif 显示，然后才是对 sans-serif 的操作。

所有的 Noto CJK 字体都以 SC 结尾，因为我希望在没指定语言的默认情况下，以简体中文显示中文字形。

问题来了，为什么我要用属性 `binding="strong"` ？这和 fontconfig 的绑定特性有关。在对 font pattern 的编辑处理过后，fontconfig 会再对 font pattern 进行排序；font pattern 中的强绑定字体会移动到前面，弱绑定字体会移动到后面。也就是说，f1(s) f2(w) f3(w) f4(s) 这种 font pattern 会变成 f1(s) f4(s) f2(w) f3(w)。(s) 代表强绑定字体，(w) 代表弱绑定字体。不仅如此，fontconfig 还要对弱绑定字体再次排序，如果 f3(w) 这个字体支持 font pattern 中传入的语言参数，它将被移动到 f2(w) 前面。所以最终结果是 f1(s) f4(s) f3(w) f2(w)。

这也就是为什么很多人说，明明在 fontconfig 中设置好的字体顺序，结果还是乱了。你期望 fontconfig 编辑过后的字体顺序是 f1(s) f2(w) f3(w) f4(s)，但最后变成了 f1(s) f4(s) f3(w) f2(w)。其实是因为没有注意到 fontconfig 的强弱绑定特性。所以，在我们配置 fontconfig 的时候，应该统一使用强绑定，也就是 `binding="strong"` ，让自己的规则优先于系统端的默认配置。

当网页指定了页面语言，Firefox 和 Chrome 浏览器是怎么处理它的呢？

Firefox 只会对三大通用字体族名 (sans-serif，serif，monospace) 传递页面语言给 fontconfig，其他字体则是传环境变量中的 LANG。另外，Firefox 会把 system-ui 当作普通的字族名传递给 fontconfig，而没有特殊处理它。(TODO 待确认：Firefox 在面对通用字体族名时，只会取配置中的前 3 个字体？)

Chrome 完全不传语言给 fontconfig。与 Firefox 不一样，Chrome 会特殊对待 system-ui，从 GTK 设置那里直接拿来字体。

第二步：按语言选择异体字

显示异体字有两种方式：

- 在 fontconfig 的字体匹配阶段，通过语言选中该语言对应的字体族名。
- 在字体渲染阶段，通过字体文件带有的 GSUB 表根据语言选择字形。

为了能让程序或者 Web 页面能在 zh-HK 语言环境中使用 Noto Sans CJK HK，可以这样做：

```
<match target="pattern">
  <test name="lang">
    <string>zh-HK</string>
  </test>
  <test name="family">
    <string>Noto Sans CJK SC</string>
  </test>
  <edit name="family" binding="strong">
    <string>Noto Sans CJK HK</string>
  </edit>
</match>
<!-- 另外请再重复 zh-TW、ja、ko -->
```

在字体匹配时，将 Noto Sans CJK SC 替换成 Noto Sans CJK HK。除了 zh-HK，你还需要照葫芦画瓢分别指定 zh-TW、ja、ko，以及对宋体和等宽字体进行重复的步骤。在此我就不重复了。

PS. 在用 FC_DEBUG 查看 fontconfig 的调试信息时，程序传递给 fontconfig 的语言参数，显示在 lang 一行而不是 familylang。familylang 是指一个字体它的族名的语言版本，比如简中版思源黑体的字体族名有「思源黑体」(zh-cn)、「Source Han Sans SC」(en)，这两个字体族名指向同一个字体。

第三步：替换任意字体

当系统里已经安装了一些不需要的字体，但又不想删除或者屏蔽它怎么办呢？替换掉 font pattern 就可以了。我发现在安装完 Arch Linux 后，存在依赖 adobe-source-code-pro-fonts，它包含了 Source Code Variable 和 Source Code Pro 这两个字体，但我不想使用它们，所以可以直接替换掉

```
<match target="pattern">
  <test name="family" compare="contains">
    <string>Source Code</string>
  </test>
  <edit name="family" binding="strong">
    <string>Iosevka Term</string>
  </edit>
</match>
```

注意我用了 `compare="contains"` 直接匹配这两个字体，然后改成了我需要的 Iosevka Term。

第四步：覆盖西文字体

让西文字体出现在 font pattern 中的中文字体前面，就能让程序优先使用我们指定的西文字体了，同时也解决了引号问题。在此我们需要继续使用强绑定。

```
<match target="pattern">
  <test name="family" compare="contains">
    <string>Noto Sans CJK</string>
  </test>
  <edit name="family" mode="prepend" binding="strong">
    <string>Noto Sans</string>
  </edit>
</match>
<!-- 糟糕！中文文本的引号不是全角的 -->
```

慢着！这样岂不是在中文环境中使用西文字体，包括英文字符宽度的引号？确实有这个问题。所以，把上面的配置改成：

```
<!-- 比上面的配置好些 -->
<match target="pattern">
  <test name="lang" compare="contains">
    <string>en</string>
  </test>
  <test name="family" compare="contains">
    <string>Noto Sans CJK</string>
  </test>
  <edit name="family" mode="prepend" binding="strong">
    <string>Noto Sans</string>
  </edit>
</match>
```

让西文字体替换规则只在 en 环境中有效，在无语言或者其他语言的情况下，继续使用中文字体中的西文字形和全角引号。

我现在想在等宽字体中优先使用 Iosevka Term，所以可以这样写：

```
<match target="pattern">
  <test name="family" compare="contains">
    <string>Noto Sans Mono CJK</string>
  </test>
  <edit name="family" mode="prepend" binding="strong">
    <string>Iosevka Term</string>
  </edit>
</match>
<!-- 糟糕！Chrome 的中文字体不是我想要的 -->
```

注意！ 有些程序居然只使用 font pattern 结果中的首个字体，比如 Chrome，虽然 Chrome 接受了我们指定的西文字体，但是它忽略了紧接其后的中文字体，即使配置采用了强绑定！然后中文字体又不知道它 fallback 到哪去了，可能会出现你想要的中文字体，也可能不是。所以，可以这样做：

```
<!-- 比上面的配置好些 -->
<match target="pattern">
  <test name="prgname" compare="not_eq">
    <string>chrome</string>
  </test>
  <test name="family" compare="contains">
    <string>Noto Sans Mono CJK</string>
  </test>
  <edit name="family" mode="prepend" binding="strong">
    <string>Iosevka Term</string>
  </edit>
</match>
```

在所有情况下，除了程序名为 Chrome 的情况下，优先使用 Iosevka Term 显示西文，再用 Noto Sans Mono CJK 显示中文。虽然我不能让 Chrome 使用 Iosevka Term，但它一定能用上 Noto Sans Mono CJK 显示中文。由于排除了 Chrome，所以无法解决 Chrome 中的全角引号问题。

那么上面那些字体覆盖规则不都有这个问题吗？没关系，在一份语言标记为 en 的 Web 页面中，出现中文的情况是罕见的，所以我不怎么担心 Chrome 使用了哪个中文字体。实际上，我平时用 Firefox 不用 Chrome，完全没这个烦恼。

不能解决的问题

虽然 fontconfig 本身不负责渲染字体，实际渲染是由程序交给 freetype 来完成的；但是 fontconfig 在处理 font pattern 的时候可以修改一些渲染参数，比如是否开启 hinting 和子像素渲染，是否开启连字特性。同时也可以根据实际情况来选择和过滤字体，比如根据像素尺寸分别指定字体，又或者控制何时使用点阵字体。所以渲染效果如何，不仅仅是 fontconfig 的问题，同时也是桌面程序自身以及 freetype 的问题。你可以用 fontconfig 控制一些渲染特性和参数，指定渲染效果更好的字体，但不能指望 fontconfig 百分百帮你处理所有的渲染问题。

Linux 不强迫程序必须使用特定的依赖，而是程序主动选择了约定俗成的依赖。老话重谈，程序可以自由选择完全遵守 fontconfig，也可以选择部分使用 fontconfig 的配置，或者完全不遵守它。只不过现如今大部分 Linux 程序都能很好地和 fontconfig 协同工作，因为绝大部分 Linux 程序使用了 Qt 或者 GTK 这些部件库来开发界面，而 Qt 和 GTK 本身就依赖 fontconfig，它们参与其中，从而让程序默认就遵守了 fontconfig 的配置。如果程序不采用 Qt 和 GTK 来开发界面，而是自己来绘制窗口，那么也会尽可能地使用 fontconfig，这在一些朴素的虚拟终端模拟器上得到了体现，比如虚拟终端 Alacritty 在没有使用 Qt 或者 GTK 开发界面的情况下，直接调用 fontconfig 和 freetype 来渲染字体。

只能说现如今的 Linux 程序的行为如此，但也有少部分程序使用 Xft，那么就不是 fontconfig 能管得了的。这体现在一些历史悠久的软件上，比如虚拟终端 rxvt，以及一些古朴的游戏。它们是从 Xft 的配置中，读取字体配置的。

也有些程序可以使用自己打包的运行时环境，它自身携带的 fontconfig 库和配置与系统端的 fontconfig 相互独立。比如 Steam， 它的第一个启动框还遵守着我们系统上的 fontconfig 配置和字体文件，因为这个启动框是 Steam 通过使用 zenity 绘制的。在 Steam 进入主界面后，Steam 会使用自己运行时环境中的 fontconfig，并且使用自己的字体文件。所以我们的 fontconfig 配置对 Steam 无效。