The fontspec package
Font selection for XeLaTeX and LuaLaTeX

Will Robertson
With contributions by Khaled Hosny,
Philipp Gesang, Joseph Wright, and others.
http://wspr.io/fontspec/

2022/01/15 v2.8a

Contents

Part I

Getting started
开始

1 History
历史

This package began life as a LaTeX interface to select system-installed macOS fonts in Jonathan Kew's XeTeX, the first widely-used Unicode extension to TeX. Over time, XeTeX was extended to support OpenType fonts and then was ported into a cross-platform program to run also on Windows and Linux.

这个宏包最初是作为一个 LaTeX 接口，用于在 Jonathan Kew 的 XeTeX 中选择安装在系统中的 macOS 字体，这是第一个被广泛使用的 Unicode 扩展 TeX。随着时间的推移，XeTeX 得到了扩展，支持了 OpenType 字体，然后被移植成一个跨平台程序，可以在 Windows 和 Linux 上运行。

More recently, LuaTeX is fast becoming the TeX engine of the day; it supports Unicode encodings and OpenType fonts and opens up the internals of TeX via the Lua programming language. Hans Hagen's ConTeXt Mk. IV is a re-write of his powerful typesetting system, taking full advantage of LuaTeX's features including font support; a kernel of his work in this area has been extracted to be useful for other TeX macro systems as well, and this has enabled fontspec to be adapted for LaTeX when run with the LuaTeX engine.

最近，LuaTeX 快速成为当今的 TeX 引擎；它支持 Unicode 编码和 OpenType 字体，并通过 Lua 编程语言打开了 TeX 的内部。Hans Hagen 的 ConTeXt Mk.,IV 是他强大的排版系统的重写，充分利用了 LuaTeX 的特性，包括字体支持；他在这个领域的一些工作被提取出来，可以用于其他 TeX 宏系统，并且这使得 fontspec 在使用 LuaTeX 引擎时适用于 LaTeX。

2 Introduction
介绍

The fontspec package allows users of either XeTeX or LuaTeX to load OpenType fonts in a LaTeX document. No font installation is necessary, and font features can be selected and used as desired throughout the document.

fontspec 包允许 LaTeX 文档中使用 XeTeX 或LuaTeX 加载 OpenType 字体。无需安装字体，文档中可随意选择和使用字体特性。

Without fontspec, it is necessary to write cumbersome font definition files for LaTeX, since LaTeX's font selection scheme (known as the 'NFSS') has a lot going on behind the scenes to allow easy commands like `\emph` or `\bfseries`. With an uncountable number of fonts now available for use, however, it becomes less desirable to have to write these font definition (`.fd`) files for every font one wishes to use.

如果没有使用 fontspec，则需要为 LaTeX 写冗长的字体定义文件，因为 LaTeX 的字体选择方案（称为“NFSS”）在幕后进行了许多处理，以允许使用简单的命令，如 `\emph`或 `\bfseries`。然而，由于现在有无数可用的字体，因此不希望为希望使用的每个字体编写这些字体定义（`.fd`）文件。

Because fontspec is designed to work in a variety of modes, this user documentation is split into separate sections that are designed to be relatively independent. Nonetheless, the basic functionality all behaves in the same way, so previous users of fontspec under XeTeX should have little or no difficulty switching over to LuaTeX.

由于 fontspec 设计为在各种模式下工作，因此本用户文档分为单独的部分，这些部分被设计为相对独立。尽管如此，所有基本功能的行为方式都相同，因此之前在 XeTeX 下使用 fontspec 的用户应该很容易切换到 LuaTeX。

This manual can get rather in-depth, as there are a lot of details to cover. See the documents `fontspec-example.tex` for a complete minimal example to get started quickly.

由于本手册包含了许多详细信息，因此可能会深入到细节。要想获得完整的最小示例，以便快速入门。可以查看文档 `fontspec-example.tex`。

2.1 Acknowledgements
致谢

This package could not have been possible without the early and continued support the author of XeTeX, Jonathan Kew. When I started this package, he steered me many times in the right direction.

感谢 XeTeX 的作者 Jonathan Kew 在早期和持续支持中的作用，否则本宏包无法存在。当我开始编写此软件包时，他在许多方面给予我指导。

I've had great feedback over the years on feature requests, documentation queries, bug reports, font suggestions, and so on from lots of people all around the world. Many thanks to you all.

多年来，我从世界各地的许多人那里得到了很好的反馈，例如功能请求、文档查询、错误报告、字体建议等等。感谢你们所有人。

Thanks to David Perry and Markus Böhning for numerous documentation improvements and David Perry again for contributing the text for one of the sections of this manual.

感谢 David Perry 和 Markus Böhning 为文档做出的大量改进，David Perry 再次为本手册的一部分贡献了文本。

Special thanks to Khaled Hosny, who was the driving force behind the support for LuaLaTeX, ultimately leading to version 2.0 of the package.

特别感谢 Khaled Hosny，他是支持 LuaLaTeX的驱动力，最终导致了该软件包的 2.0 版本。

3 Package loading and options
包加载和选项

For basic use, no package options are required:

对于基本用法，不需要任何包选项：

```
\usepackage{fontspec}
```

Package options will be introduced below; some preliminary details are discussed first.

包选项将在下面介绍；首先讨论一些预备细节。

3.1 Font encodings
字体编码

The (default) `tuenc` package option switches the NFSS font encoding to `TU`. `TU` is a new Unicode font encoding, intended for both XeTeX and LuaTeX engines, and automatically contains support for symbols covered by LaTeX's traditional `T1` and `TS1` font encodings (for example, `\%`, `\textbullet`, `\"u`, and so on). Some additional features are provided by fontspec to customise some encoding details; see Part V on page 68 for further details.

（默认的）`tuenc` 宏包选项将 NFSS 字体编码切换为 `TU`。`TU` 是一种新的 Unicode 字体编码，旨在为 XeTeX 和LuaTeX引擎提供支持，并自动包含 LaTeX 传统的 `T1` 和 `TS1` 字体编码所涵盖的符号的支持（例如`%`，`\textbullet`，`"u` 等）。一些附加功能由 fontspec 提供以自定义一些编码细节；更多详细信息请参见第 V on page 68 部分。

Pre-2017 behaviour can be achieved with the `euenc` package option. This selects the `EU1` or `EU2` encoding (XeTeX/LuaTeX, resp.) and loads the xunicode package for symbol support. Package authors and users who have referred explicitly to the encoding names `EU1` or `EU2` should update their code or documents. (See internal variable names described in Section 5 on page 89 for how to do this properly.)

可以通过使用 `euenc` 宏包选项来实现 2017 年之前的行为。这将选择 `EU1` 或 `EU2` 编码（分别为 XeTeX/LuaTeX），并加载 xunicode 宏包以支持符号。已明确引用编码名称 `EU1` 或 `EU2` 的宏包作者和用户应更新其代码或文档。（有关如何正确执行此操作的方法，请参见第Section 5 on page 89节中描述的内部变量名称。）

3.2 Maths fonts adjustments
数学字体调整

By default, fontspec adjusts LaTeX's default maths setup in order to maintain the correct Computer Modern symbols when the roman font changes. However, it will attempt to avoid doing this if another maths font package is loaded (such as mathpazo or the unicode-math package).

默认情况下，fontspec 会调整 LaTeX 的默认数学设置，以保持正确的 Computer Modern 符号，当罗马字体发生更改时。但是，如果加载了另一个数学字体宏包（例如 mathpazo 或 unicode-math 宏包），则它将尝试避免这样做。

If you find that fontspec is incorrectly changing the maths font when it shouldn't be, apply the `no-math` package option to manually suppress its behaviour here.

如果您发现 fontspec 在不应该更改数学字体的情况下错误地更改了它，请在此处应用`no-math` 宏包选项以手动抑制其行为。

3.3 Configuration
配置

If you wish to customise any part of the fontspec interface, this should be done by creating your own `fontspec.cfg` file, which will be automatically loaded if it is found by XeTeX or LuaTeX. A `fontspec.cfg` file is distributed with fontspec with a small number of defaults set up within it.

如果您希望自定义 fontspec 接口的任何部分，应创建自己的 `fontspec.cfg` 文件，如果XeTeX 或LuaTeX找到它，它将自动加载。fontspec 附带了一个`fontspec.cfg` 文件，其中设置了一些默认值。

To customise fontspec to your liking, use the standard `.cfg` file as a starting point or write your own from scratch, then either place it in the same folder as the main document for isolated cases, or in a location that XeTeX or LuaTeX searches by default; *e.g.* in MacTeX: `~/Library/texmf/tex/latex/`.

要将 fontspec 自定义为所需的样子，请使用标准的`cfg` 文件作为起点或从头开始编写自己的文件，然后将其放在与主文档相同的文件夹中，以进行隔离的情况，或者将其放在XeTeX 或LuaTeX默认搜索的位置中；例如，在 MacTeX 中：`~/Library/texmf/tex/latex/`。

The package option `no-config` will suppress the loading of the `fontspec.cfg` file under all circumstances.

包选项`no-config` 将在任何情况下都禁止加载`fontspec.cfg` 文件。

3.4 Warnings
警告

This package can give some warnings that can be harmless if you know what you're doing. Use the `quiet` package option to write these warnings to the transcript (`.log`) file instead.

这个包可能会产生一些警告，如果你知道你在做什么，这些警告是无害的。你可以使用 `quiet` 包选项将这些警告写入转录 (`.log`) 文件中。

Use the `silent` package option to completely suppress these warnings if you don't even want the `.log` file cluttered up.

如果你甚至不想让 `.log` 文件变得混乱，你可以使用 `silent` 包选项来完全抑制这些警告。

4 Interaction with LaTeX 2ε and other packages
与LaTeX 2ε和其他软件包的交互

This section documents some areas of adjustment that fontspec makes to improve default behaviour with LaTeX 2ε and third-party packages.

这一节介绍了 fontspec 对 LaTeX 2ε 和第三方包的默认行为进行了一些调整，以改善其效果。

4.1 Commands for old-style and lining numbers
旧式数字和直立数字的命令

\oldstylenums
\liningnums

LaTeX's definition of \oldstylenums relies on strange font encodings. We provide a fontspec-compatible alternative and while we're at it also throw in the reverse option as well. Use \oldstylenums{⟨*text*⟩} to explicitly use old-style (or lowercase) numbers in ⟨*text*⟩, and the reverse for \liningnums{⟨*text*⟩}.

LaTeX 的 \oldstylenums 定义依赖于奇怪的字体编码。我们提供了一个与 fontspec 兼容的替代方案，顺便也加入了反向选项。使用 \oldstylenums{⟨*text*⟩} 来显式地在 ⟨*text*⟩ 中使用旧式（或小写）数字，反之亦然。

4.2 Italic small caps
斜体小型大写字母

Support now provided by LaTeX 2ε in 2020.

在 2020 年由LaTeX 2ε支持。

4.3 Emphasis and nested emphasis
强调和嵌套强调

Support now provided by LaTeX 2ε in 2020.

在 2020 年由LaTeX 2ε支持。

4.4 Strong emphasis
强烈强调

\strong
\strongenv

The \strong macro is used analogously to \emph but produces variations in weight. If you need it in environment form, use \begin{strongenv}...\end{strongenv}.

\strong宏类似于\emph，但产生字体粗细的变化。如果您需要使用环境形式，请使用\begin{strongenv}...\end{strongenv}。

As with emphasis, this font-switching command is intended to move through a range of font weights. For example, if the fonts are set up correctly it allows usage such as \strong{...\strong{...}} in which each nested \strong macro increases the weight of the font.

与强调一样，该字体切换命令旨在移动到一系列字体重量中。例如，如果正确设置了字体，则允许使用如下用法：`\strong{...\strong{...}}`，其中每个嵌套的 `\strong` 宏都增加了字体的重量。

`\strongfontdeclare`

Currently this feature set is somewhat experimental and there is no syntactic sugar to easily define a range of font weights using fontspec commands. Use, say, the following to define first bold and then black (`k`) font faces for `\strong`:

目前，该功能集有点实验性，没有使用 fontspec 命令轻松定义一系列字体权重的简洁语法。例如，使用以下内容定义 `\strong` 的第一个粗体，然后是黑体 (`k`) 字体面:

```
\strongfontdeclare{\bfseries,\fontseries{k}\selectfont}
```

`\strongreset`

If too many levels of `\strong` are reached, `\strongreset` is inserted. By default this is a no-op and the font will simply remain the same. Use `\renewcommand\strongreset{\mdseries}` to start again from the beginning if desired.

如果达到太多级别的`\strong`，则插入`\strongreset`。默认情况下，这是一个无操作，并且字体将保持不变。如果需要，使用`\renewcommand\strongreset{\mdseries}` 从头开始再次启动。

An example for setting up a font family for use with `\strong` is discussed in 4.3.1 on page 23.

有关为使用`\strong`设置字体系列的示例，请参见4.3.1。

Part II

General font selection
通用字体选择

1 Main commands
主要命令

This section concerns the variety of commands that can be used to select fonts.

本节介绍可用于选择字体的各种命令。

```
\setmainfont{⟨font⟩}[⟨font features⟩]
\setsansfont{⟨font⟩}[⟨font features⟩]
\setmonofont{⟨font⟩}[⟨font features⟩]
```

These are the main font-selecting commands of this package which select the standard fonts used in a document, as shown in Example 1. Here, the scales of the fonts have been chosen to equalise their lowercase letter heights. The `Scale` font feature will be discussed further in Section 6 on page 34, including methods for automatic scaling. Note that further options may need to be added to select appropriate bold/italic fonts, but this shows the main idea.

这些是本包中用于选择文档中使用的标准字体的主要字体选择命令，如 Example 1所示。这里，字体的比例已被选择为平衡其小写字母的高度。字体特性 `Scale` 将在Section 6 on page 34中进一步讨论，包括自动缩放的方法。请注意，可能需要添加其他选项以选择适当的粗体/斜体字体，但这显示了主要思想。

Note that while these commands all look and behave largely identically, the default setup for font loading automatically adds the `Ligatures=TeX` feature for the `\setmainfont` and `\setsansfont` commands. These defaults (and further customisations possible) are discussed in Section 1 on page 27.

请注意，虽然这些命令看起来并且行为大部分相同，但默认设置的字体加载会自动为`\setmainfont`和`\setsansfont`命令添加`Ligatures=TeX`特性。这些默认设置（以及更多可定制的选项）在Section 1 on page 27中进行了讨论。

```
\newfontfamily⟨cmd⟩{⟨font⟩}[⟨font features⟩]
\setfontfamily⟨cmd⟩{⟨font⟩}[⟨font features⟩]
\renewfontfamily⟨cmd⟩{⟨font⟩}[⟨font features⟩]
\providefontfamily⟨cmd⟩{⟨font⟩}[⟨font features⟩]
```

These commands define new font family commands (like `\rmfamily`). The `new` command checks if ⟨*cmd*⟩ has been defined, and issues an error if so. The `renew` command checks if ⟨*cmd*⟩ has been defined, and issues an error if not. The `provide` command checks

if ⟨*cmd*⟩ has been defined, and silently aborts if so. The `set` command never checks; use at your own risk.

这些命令定义了新的字体系列命令（例如`\rmfamily`）。`new` 命令检查是否已定义了⟨*cmd*⟩，如果已定义则发出错误。`renew` 命令检查是否已定义了⟨*cmd*⟩，如果未定义则发出错误。`provide` 命令检查是否已定义了⟨*cmd*⟩，如果已定义则悄悄退出。`set` 命令永远不会检查；使用时自担风险。

`\fontspec{`⟨*font*⟩`}[`⟨*font features*⟩`]`

The plain `\fontspec` command is not generally recommended for document use. It is an ad hoc command best suited for testing and loading fonts on a one-off basis.

`\fontspec` 命令一般不推荐用于文档中。它是一个临时的命令，最适合用于测试和加载一次性的字体。

All of the commands listed above accept comma-separated ⟨*font feature*⟩=⟨*option*⟩ lists; these are described later:

上面列出的所有命令都接受逗号分隔的 ⟨*font feature*⟩=⟨*option*⟩ 列表；这些将在后面介绍：

- For general font features, see Section 6 on page 34
  有关通用字体特性，请参见 Section 6 on page 34
- For OpenType fonts, see Part IV on page 43
  有关 OpenType 字体，请参见第 IV on page 43 部分
- For XeTeX-only general font features, see Part VII on page 77
  有关 XeTeX-only 通用字体特性，请参见第 VII on page 77 部分
- For LuaTeX-only general font features, see Part VI on page 74
  有关 LuaTeX-only 通用字体特性，请参见第 VI on page 74 部分
- For features for AAT fonts in XeTeX, see Section 3 on page 80
  有关 XeTeX 中的 AAT 字体特性，请参见 Section 3 on page 80

Example 1: Loading the default, sans serif, and monospaced fonts.
加载默认、无衬线和等宽字体。

```
\setmainfont{texgyrebonum-regular.otf}
\setsansfont{lmsans10-regular.otf}[Scale=MatchLowercase]
\setmonofont{Inconsolatazi4-Regular.otf}[Scale=MatchLowercase]

\rmfamily Pack my box with five dozen liquor jugs\par
\sffamily Pack my box with five dozen liquor jugs\par
\ttfamily Pack my box with five dozen liquor jugs
```

—Example graphic not generated—

2 Font selection
体选择

In both LuaTEX and XETEX, fonts can be selected (using the ⟨*font*⟩ argument in Section 1) either by 'font name' or by 'file name', but there are some differences in how each engine finds and selects fonts — don't be too surprised if a font invocation in one engine needs correction to work in the other.

在 LuaTEX和 XETEX 中，可以通过字体名'或文件名'来选择字体（使用第 Section 1 节中的 ⟨*font*⟩ 参数），但是每个引擎在查找和选择字体时有一些差异—如果一个引擎中的字体调用在另一个引擎中需要修正才能工作，不要太惊讶。

2.1 By font name
按字体名

Fonts known to LuaTEX or XETEX may be loaded by their standard names as you'd speak them out loud, such as *Times New Roman* or *Adobe Garamond*. 'Known to' in this case generally means 'exists in a standard fonts location' such as `~/Library/Fonts` on macOS, or `C:\Windows\Fonts` on Windows. In LuaTEX, fonts found in the TEXMF tree can also be loaded by name. In XETEX, fonts found in the TEXMF tree can be loaded in Windows and Linux, but not on macOS.

对于 LuaTeX 或 XeTeX 所知道的字体，可以通过它们的标准名称来加载，就像你口头说出来的那样，比如说 Times New Roman 或 Adobe Garamond。所知道'在这里通常意味着存在于标准字体位置'比如说 `~/Library/Fonts` 在 MacOSX 上，或者 `C:\Windows\Fonts` 在 Windows 上。在 LuaTeX 中，在 texmf 树中找到的字体也可以通过名称加载。在 XeTeX 中，在 texmf 树中找到的字体可以在 Windows 和 Linux 上加载，但不能在 MacOSX 上加载。

The simplest example might be something like

最简单的例子可能是像这样的：

```
\setmainfont{Cambria}[ ... ]
```

in which the bold and italic fonts will be found automatically (if they exist) and are immediately accessible with the usual `\textit` and `\textbf` commands.

其中，粗体和斜体字体将自动被找到（如果存在），并可以立即通过常规的 `\textit` 和 `\textbf` 命令进行访问。

The 'font name' can be found in various ways, such as by looking in the name listed in a application like *Font Book* on Mac OS X. Alternatively, TEXLive contains the `otfinfo` command line program, which can query this information; for example:

"字体名称" 可以通过多种方式找到，例如通过在 Mac OS X 上的 *Font Book* 中列出的名称进行查找。或者，TEXLive 包含 `otfinfo` 命令行程序，可以查询此信息；例如：

```
otfinfo -i `kpsewhich lmroman10-regular.otf`
```

results in a line that reads:

会输出以下一行：

```
Preferred family:    Latin Modern Roman
```

(The 'preferred family' name is usually better than the 'family' name.)

("首选族"名称通常比"族"名称更好。)

LuaTEX users only
LuaTEX 用户专用 In order to load fonts by their name rather than by their filename (*e.g.*, 'Latin Modern Roman' instead of 'ec-lmr10'), you may need to run the script `luaotfload-tool`, which is distributed with the luaotfload package. Note that if you do not execute this script beforehand, the first time you attempt to typeset the process will pause for (up to) several minutes. (But only the first time.)

为了通过名称而不是文件名加载字体(例如,"Latin Modern Roman"而不是"ec-lmr10"),您可能需要运行随 luaotfload 包一起分发的脚本 `luaotfload-tool`。请注意,如果您没有预先执行此脚本,则在您尝试排版过程时,该过程将暂停数分钟(最多)。(但只有第一次。)

Please see the luaotfload documentation for more information.

有关更多信息,请参见 luaotfload 文档。

2.2 By file name
通过文件名加载

XeTEX and LuaTEX also allow fonts to be loaded by file name instead of font name. When you have a very large collection of fonts, you will sometimes not wish to have them all installed in your system's font directories. In this case, it is more convenient to load them from a different location on your disk. This technique is also necessary in XeTEX when loading OpenType fonts that are present within your TEX distribution, such as `/usr/local/texlive/2013/texmf-dist/fonts/opentype/public`. Fonts in such locations are visible to XeTEX but cannot be loaded by font name, only file name; LuaTEX does not have this restriction.

XeTEX 和 LuaTEX 还允许通过文件名而不是字体名称加载字体。当您有大量字体集合时,有时可能不希望将它们全部安装在系统的字体目录中。在这种情况下,从磁盘上的不同位置加载它们会更方便。在加载存在于您的 TEX 发行版中的 OpenType 字体(例如 `/usr/local/texlive/2013/texmf-dist/fonts/opentype/public`)时,这种技术在 XeTEX 中也是必要的。这种位置上的字体对 XeTEX 可见,但只能通过文件名而不是字体名称加载;LuaTEX 没有这个限制。

When selecting fonts by file name, any font that can be found in the default search paths may be used directly (including in the current directory) without having to explicitly define the location of the font file on disk.

当通过文件名选择字体时,可以直接使用默认搜索路径中找到的任何字体(包括当前目录),无需显式定义磁盘上字体文件的位置。

Fonts selected by filename must include bold and italic variants explicitly, unless a `.fontspec` file is supplied for the font family (see Section 2.3). We'll give some first examples specifying everything explicitly:

通过文件名选择的字体必须显式包括粗体和斜体变体，除非为字体系列提供了 .fontspec 文件（请参见 Section 2.3 ）。我们将给出一些第一次显式指定所有内容的示例：

```
\setmainfont{texgyrepagella-regular.otf}[
     BoldFont       = texgyrepagella-bold.otf ,
     ItalicFont     = texgyrepagella-italic.otf ,
     BoldItalicFont = texgyrepagella-bolditalic.otf ]
```

fontspec knows that the font is to be selected by file name by the presence of the '.otf' extension. An alternative is to specify the extension separately, as shown following:

fontspec 可以通过文件名来选择字体，方法是在文件名中包含“.otf”扩展名。另一种选择是分别指定扩展名，如下所示：

```
\setmainfont{texgyrepagella-regular}[
     Extension      = .otf ,
     BoldFont       = texgyrepagella-bold ,
     ... ]
```

If desired, an abbreviation can be applied to the font names based on the mandatory 'font name' argument:

如果需要，可以根据必需的“字体名称”参数对字体名称进行缩写：

```
\setmainfont{texgyrepagella}[
     Extension      = .otf ,
     UprightFont    = *-regular ,
     BoldFont       = *-bold ,
     ... ]
```

In this case 'texgyrepagella' is no longer the name of an actual font, but is used to construct the font names for each shape; the * is replaced by 'texgyrepagella'. Note in this case that UprightFont is required for constructing the font name of the normal font to use.

在这种情况下，“texgyrepagella”不再是实际字体的名称，而是用于构建每种形状的字体名称；星号“*”被替换为“texgyrepagella”。请注意，在此情况下，需要使用 UprightFont 来构建要使用的普通字体的字体名称。

To load a font that is not in one of the default search paths, its location in the filesystem must be specified with the Path feature:

要加载不在默认搜索路径之一的字体，必须使用 Path 功能指定其在文件系统中的位置：

```
\setmainfont{texgyrepagella}[
     Path           = /Users/will/Fonts/ ,
     UprightFont    = *-regular ,
     BoldFont       = *-bold ,
     ... ]
```

Note that XeTeX and LuaTeX are able to load the font without giving an extension, but fontspec must know to search for the file; this can be indicated by using the Path feature without an argument:

请注意，XeTeX 和 LuaTeX可以在不提供扩展名的情况下加载字体，但是 fontspec 必须知道要搜索该文件；可以通过使用没有参数的 Path 功能来指示这一点：

```
\setmainfont{texgyrepagella-regular}[
     Path, BoldFont = texgyrepagella-bold,
     ... ]
```

My preference is to always be explicit and include the extension; this also allows fontspec to automatically identify that the font should be loaded by filename.

我的偏好是始终明确地包含扩展名；这也使得 fontspec 能够自动识别应该通过文件名加载字体。

In previous versions of the package, the Path feature was also provided under the alias ExternalLocation, but this latter name is now deprecated and should not be used for new documents.

在包的早期版本中，Path 功能也在别名 ExternalLocation 下提供，但是后者的名称现已弃用，不应在新文档中使用。

2.3 By custom file name using a .fontspec file
通过使用 .fontspec 文件的自定义文件名

When fontspec is first asked to load a font, a font settings file is searched for with the name '⟨*fontname*⟩.fontspec'.[1] If you want to *disable* this feature on a per-font basis, use the IgnoreFontspecFile font option.

当 fontspec 第一次被要求加载一个字体时，会搜索一个名为 '⟨*fontname*⟩.fontspec' 的字体设置文件。[2] 若你想在每个字体的基础上禁用这个功能，可以使用 IgnoreFontspecFile 字体选项。

The contents of this file can be used to specify font shapes and font features without having to have this information present within each document. Therefore, it can be more flexible than the alternatives listed above.

这个文件的内容可以用来指定字体形状和字体特性，而不必在每个文档中都有这些信息。因此，它比上面列出的替代方案更灵活。

When searching for this .fontspec file, ⟨*fontname*⟩ is stripped of spaces and file extensions are omitted. For example, given \setmainfont{TeX Gyre Adventor}, the .fontspec file would be called TeXGyreAdventor.fontspec. If you wanted to transparently load options for \setmainfont{texgyreadventor-regular.otf}, the configuration file would be texgyreadventor-regular.fontspec.

在搜索这个 .fontspec 文件时，⟨*fontname*⟩ 会去掉空格，并省略文件扩展名。例如，给定 \setmainfont {TeX Gyre Adventor}，.fontspec 文件将被称为 TeXGyreAdventor.fontspec. 如果你想透明地加载 \setmainfont {texgyreadventor-regular.otf} 的选项，配置文件将是 texgyreadventor-regular.fontspec.

N.B. that while spaces are stripped, the lettercase of the names should match.

[1] Located in the current folder or within a standard texmf location.
[2] 位于当前文件夹或标准的 texmf 位置。

注意，虽然空格被去掉了，但名称的大小写应该匹配。

This mechanism can be used to define custom names or aliases for your font collections. The syntax within this file follows from the `\defaultfontfeatures`, defined in more detail later but mirroring the standard fontspec font loading syntax. As an example, suppose we're defining a font family to be loaded with `\setmainfont{My Charis}`. The corresponding `MyCharis.fontspec` file would contain, say,

该机制可用于为您的字体集合定义自定义名称或别名。该文件中的语法遵循 `\defaultfontfeatures`，稍后将更详细地定义，但是镜像标准 fontspec 字体加载语法。例如，假设我们要定义一个将通过 `\setmainfont{My Charis}` 加载的字体系列。相应的 `MyCharis.fontspec` 文件将包含如下内容：

```
\defaultfontfeatures[My Charis]
  {
    Extension = .ttf ,
    UprightFont    = CharisSILR,
    BoldFont       = CharisSILB,
    ItalicFont     = CharisSILI,
    BoldItalicFont = CharisSILBI,
    % <any other desired options>
  }
```

The optional argument to `\defaultfontfeatures` must exactly match that requested by the font loading command (`\setmainfont`, etc.) — in particular note that spaces are significant here, so `\setmainfont{MyCharis}` will not 'see' the default font feature setting within the `.fontspec` file.

`\defaultfontfeatures` 的可选参数必须与字体加载命令 (`\setmainfont` 等) 请求的完全匹配—特别注意这里的空格是有意义的，所以 `\setmainfont{MyCharis}` 不会 '看到' `.fontspec` 文件中的默认字体特性设置。

Finally, note that options for individual font faces can also be defined in this way. To continue the example above, here we colour the different faces:

最后，请注意，个别字体面的选项也可以通过这种方式定义。接着上面的例子，这里我们给不同的面设置颜色：

```
\defaultfontfeatures[CharisSILR]{Color=blue}
\defaultfontfeatures[CharisSILB]{Color=red}
```

Such configuration lines could be stored either inline inside `My Charis.fontspec` or within their own `.fontspec` files; in this way, fontspec is designed to handle 'nested' configuration options.

这样的配置行可以存储在 `My Charis.fontspec` 内联中，也可以存储在它们自己的 `.fontspec` 文件中；这样，fontspec 被设计为处理“嵌套”配置选项。

Where `\defaultfontfeatures` is being used to specify font faces by a custom name, the `Font` feature is used to set the filename of the font face. For example:

当 `\defaultfontfeatures` 被用于通过自定义名称指定字体面时，`Font` 特征被用于设置字体面的文件名。例如：

```
\defaultfontfeatures[charis]
  {
    UprightFont = charis-regular,
    % <other desired options for all font faces in the family>
  }

\defaultfontfeatures[charis-regular]
  {
    Font = CharisSILR
    % <other desired options just for the `upright' font>
  }
```

The fontspec interface here is designed to be flexible to accomodate a variety of use cases; there is more than one way to achieve the same outcome when font faces are collected together into a larger font family.

在这里，fontspec 接口被设计为灵活适应各种用例；当将字体面收集到更大的字体系列中时，有多种方法可以实现相同的结果。

2.4 Querying whether a font 'exists'
查询字体是否“存在”

`\IfFontExistsTF{`⟨*font name*⟩`}{`⟨*true branch*⟩`}{`⟨*false branch*⟩`}`

The conditional `\IfFontExistsTF` is provided to test whether the ⟨*font name*⟩ exists or is loadable. If it is, the ⟨*true branch*⟩ code is executed; otherwise, the ⟨*false branch*⟩ code is.

提供了条件语句 `\IfFontExistsTF` 来测试 ⟨字体名称⟩ 是否存在或可加载。如果存在，则执行 ⟨真分支⟩ 代码；否则，执行 ⟨假分支⟩ 代码。

This command can be slow since the engine may resort to scanning the filesystem for a missing font. Nonetheless, it has been a popular request for users who wish to define 'fallback fonts' for their documents for greater portability.

该命令可能会很慢，因为引擎可能会扫描文件系统以查找缺失的字体。尽管如此，对于希望为其文档定义“回退字体”以实现更大的可移植性的用户来说，它已经成为一个受欢迎的请求。

In this command, the syntax for the ⟨*font name*⟩ is a restricted/simplified version of the font loading syntax used for `\fontspec` and so on. Fonts to be loaded by filename are detected by the presence of an appropriate extension (`.otf`, etc.), and paths should be included inline. E.g.:

在这个命令中，⟨字体名称⟩的语法是用于`\fontspec`等的字体加载语法的受限/简化版本。通过文件名加载的字体是通过存在适当的扩展名（`.otf` 等）来检测的，并且路径应该被包含在行内。例如：

```
\IfFontExistsTF{cmr10}{T}{F}
\IfFontExistsTF{Times New Roman}{T}{F}
\IfFontExistsTF{texgyrepagella-regular.otf}{T}{F}
\IfFontExistsTF{/Users/will/Library/Fonts/CODE2000.TTF}{T}{F}
```

The \IfFontExistsTF command is a synonym for the programming interface function \fontspec_font_if_exist:nTF (Section 5 on page 89).

\IfFontExistsTF 命令是编程接口函数 \fontspec_font_if_exist:nTF（Section 5 on page 89）的同义词。

3 Commands to select font families
选择字体系列的命令

For cases when a specific font with a specific feature set is going to be re-used many times in a document, it is inefficient to keep calling \fontspec for every use. While the \fontspec command does not define a new font instance after the first call, the feature options must still be parsed and processed.

在文档中多次重复使用具有特定特征集的特定字体时，每次使用都调用\fontspec是低效的。虽然第一次调用\fontspec命令不定义新的字体实例，但特性选项仍必须被解析和处理。

For this reason, new commands can be created for loading a particular font family with the \newfontfamily command and variants, outlined in Section 1 on page 8 and demonstrated in Example 2. This macro should be used to create commands that would be used in the same way as \rmfamily, for example. If you would like to create a command that only changes the font inside its argument (i.e., the same behaviour as \emph) define it using regular LaTeX commands:

因此，可以使用\newfontfamily命令和其变体（在Section 1 on page 8中概述，示例在 Example 2中演示）创建新命令来加载特定字体系列。该宏应用于创建像\rmfamily这样的命令。如果您想创建一个只在其参数内更改字体的命令（即与\emph的行为相同），请使用常规的 LaTeX 命令定义它：

```
\newcommand\textnote[1]{{\notefont #1}}
\textnote{This is a note.}
```

Note that the double braces are intentional; the inner pair is used to delimit the scope of the font change.

请注意，双花括号是有意的；内部一对用于界定字体更改的范围。

Comment for advanced users: The commands defined by \newfontfamily (and \newfontface; see next section) include their encoding information, so even if the document is set to use a legacy TeX encoding, such commands will still work correctly. For example,

高级用户注：由\newfontfamily（和\newfontface；请参阅下一节）定义的命令包括其编码信息，因此，即使文档设置为使用传统的 TeX 编码，这些命令也将正常工作。例如：

Example 2: Defining new font families.

—Example graphic not generated—

```
\newfontfamily\notefont{Kurier}
\notefont This is a \emph{note}.
```

```
\documentclass{article}
\usepackage{fontspec}
\newfontfamily\unicodefont{Lucida Grande}
\usepackage{mathpazo}
\usepackage[T1]{fontenc}
\begin{document}
A legacy \TeX\ font. {\unicodefont A unicode font.}
\end{document}
```

4 Commands to select single font faces
选择单个字体面的命令

```
\newfontface⟨cmd⟩{⟨font⟩}[⟨font features⟩]
\setfontface⟨cmd⟩{⟨font⟩}[⟨font features⟩]
\renewfontface⟨cmd⟩{⟨font⟩}[⟨font features⟩]
\providefontface⟨cmd⟩{⟨font⟩}[⟨font features⟩]
```

Sometimes only a specific font face is desired, without accompanying italic or bold variants being automatically selected. This is common when selecting a fancy italic font, say, that has swash features unavailable in the upright forms. `\newfontface` is used for this purpose, shown in Example 3, which is repeated in Section 3.4 on page 81.

有时只需要选择特定的字体，而不会自动选择斜体或粗体变体。这种情况很常见，例如选择一种花体斜体字体，但直立形式中不可用的花边特性。可以使用`\newfontface` 来实现这一目的，如 Example 3所示，并在Section 3.4中重复说明。

4.1 More control over font shape selection
更多字体形状选择控制

```
BoldFont = ⟨font name⟩
ItalicFont = ⟨font name⟩
BoldItalicFont = ⟨font name⟩
SlantedFont = ⟨font name⟩
BoldSlantedFont = ⟨font name⟩
SwashFont = ⟨font name⟩
BoldSwashFont = ⟨font name⟩
SmallCapsFont = ⟨font name⟩
UprightFont = ⟨font name⟩
```

Example 3: Defining a single font face.

—Example graphic not generated—

```
\newfontface\fancy{Hoefler Text Italic}%
    [Contextuals={WordInitial,WordFinal}]
\fancy where is all the vegemite
% \emph, \textbf, etc., all don't work
```

The automatic bold, italic, and bold italic font selections will not be adequate for the needs of every font: while some fonts mayn't even have bold or italic shapes, in which case a skilled (or lucky) designer may be able to chose well-matching accompanying shapes from a different font altogether, others can have a range of bold and italic fonts to chose among. The `BoldFont` and `ItalicFont` features are provided for these situations. If only one of these is used, the bold italic font is requested as the default from the *new* font. See Example 4.

自动选择粗体、斜体和粗斜体字体可能无法满足每种字体的需求：有些字体甚至可能没有粗体或斜体形式，这种情况下熟练的（或幸运的）设计师可能会从另一种字体中选择匹配度较高的伴随形式，而其他字体可能有多种粗体和斜体字体可供选择。为这些情况提供了 `BoldFont` 和 `ItalicFont` 特性。如果只使用其中之一，则默认从新字体请求粗斜体字体。参见 Example 4。

If a bold italic shape is not defined, or you want to specify *both* custom bold and italic shapes, the `BoldItalicFont` feature is provided.

如果未定义粗斜体形式，或者要指定两种自定义粗体和斜体形式，则提供了 `BoldItalicFont` 特性。

4.1.1 Small caps shapes
小型大写字母形式

For modern OpenType fonts, small caps glyphs are included within a fontface and fontspec will automatically detect them for use with the `\textsc` and `\scshape` commands. Pre-OpenType, it was common for font families to be distributed with small caps glyphs in separate fonts, due to the limitations on the number of glyphs allowed in the PostScript Type 1 format. Such fonts may be used by declaring the `SmallCapsFont` for each font of the family you are specifying:

对于现代 OpenType 字体，小型大写字母字形已包含在字体中，fontspec 会自动检测它们以供`\textsc`和`\scshape`命令使用。在 OpenType 之前，由于 PostScript Type 1 格式中允许的字形数量有限制，字体系列中小型大写字母字形通常在单独的字体中分发。可以通过为所指定字体系列的每个字体声明 `SmallCapsFont` 来使用这些字体：

```
\setmainfont{ <upright> }[
  UprightFeatures    = { SmallCapsFont={ <sc> } } ,
  BoldFeatures       = { SmallCapsFont={ <bf sc> } } ,
  ItalicFeatures     = { SmallCapsFont={ <it sc> } } ,
```

Example 4: Explicit selection of the bold font.

—*Example graphic not generated*—

```
\fontspec{Helvetica Neue UltraLight}%
         [BoldFont={Helvetica Neue}]
              Helvetica Neue UltraLight         \\
{\itshape     Helvetica Neue UltraLight Italic} \\
{\bfseries                Helvetica Neue       } \\
{\bfseries\itshape        Helvetica Neue Italic} \\
```

```
  BoldItalicFeatures = { SmallCapsFont={ <bf it sc> } } ,
]
Roman 123 \\ \textsc{Small caps 456}
```

For most modern fonts that have small caps as a font feature, this level of control isn't generally necessary.

对于大多数具有小型大写字体作为字体特征的现代字体，通常不需要这种级别的控制。

All of the bold, italic, and small caps fonts can be loaded with different font features from the main font. See Section 3 for details. When an OpenType font is selected for `SmallCapsFont`, the small caps font feature is *not* automatically enabled. In this case, users should write instead, if necessary,

所有粗体、斜体和小型大写字体都可以使用不同于主字体的不同字体特征加载。有关详细信息，请参见 Section 3。当选择一个 OpenType 字体作为 `SmallCapsFont` 时，小型大写字体特征不会自动启用。在这种情况下，如果需要，用户应该写成：

```
\setmainfont{...}[
  SmallCapsFont={...},
  SmallCapsFeatures={Letters=SmallCaps},
]
```

4.1.2 Slanted font shapes
斜体字形

When a font family has both slanted *and* italic shapes, these may be specified separately using the analogous features `SlantedFont` and `BoldSlantedFont`. Without these, however, the LaTeX font switches for slanted (`\textsl`, `\slshape`) will default to the italic shape.

当一个字体族既有斜体又有倾斜形状时，可以使用类似的特征 `SlantedFont` 和 `BoldSlantedFont` 分别指定这些形状。但如果没有这些特征，则 LaTeX 的斜体字体切换命令（`\textsl`, `\slshape`）将默认使用 italic 形状。

4.1.3 Swash font shapes
花体字形

Swash font shapes in a family is supported by LaTeX's commands `\textsw` and `\swshape`. These commands assume that swash shapes are in a sense 'parallel' to italic shapes — for instance, writing both `\swshape` and `\itshape` would not result in an italic swash shape (you would get whichever was declared last). The fontspec package adopts this approach, while recognising that OpenType fonts in theory could have any crazy combination of shapes such as 'italic swash small caps'. Attempting to support arbitrarily complex situations makes setup (and the code) more difficult with let's say infrequent benefit — fontspec's alternate feature selection mechanisms (such as verb`\addfontfeature{Style=Swash}`) can be used in such situations.

在一个字体族中支持花体字形的是 LaTeX 的命令 `\textsw` 和 `\swshape`。这些命令假定花体形状在某种意义上与 italic 形状“平行”——例如，同时写入 `\swshape` 和 `\itshape` 将不会产生斜体的花体形状（您会得到最后声明的任何一个）。fontspec 包采用这种方法，

同时认识到 OpenType 字体在理论上可以具有任何疯狂的形状组合，比如“斜体花体小型大写字体”。试图支持任意复杂的情况会使设置（以及代码）更加困难，收益较少——fontspec 的备选特征选择机制（如 verb\addfontfeature{Style=Swash}）可以在这种情况下使用。

Therefore, setup is quite simple:

因此，设置非常简单：

```
\setmainfont{...}[
      SwashFont = {...} ,
  BoldSwashFont = {...} ,
]
```

No assumptions are made about the +swsh OpenType feature availability, and if desired the 'Swash' feature needs to be explicitly requested as in:

不对 +swsh OpenType 特征的可用性做任何假设，如果需要，则需要显式请求“Swash”特征，如：

```
\setmainfont{...}[
      SwashFont = {...} ,
      SwashFeatures = {Style=Swash} ,
      ...
]
```

This may become more automatic in the future.

在将来，这可能会变得更加自动化。

4.2 Specifically choosing the NFSS family
具体选择 NFSS 字体族

In LaTeX's NFSS, font families are defined with names such as 'ppl' (Palatino), 'lmr' (Latin Modern Roman), and so on, which are selected with the \fontfamily command:

在 LaTeX 的 NFSS 中，字体系列的名称使用诸如“ppl”（Palatino）、“lmr”（Latin Modern Roman）等名称定义，这些名称可以使用\fontfamily命令进行选择：

```
\fontfamily{ppl}\selectfont
```

In fontspec, the family names are auto-generated based on the fontname of the font; for example, writing \fontspec{Times New Roman} for the first time would generate an internal font family name of 'TimesNewRoman(1)'. Please note that you should not rely on the name that is generated.

在 fontspec 宏包中，系列名称是根据字体的字体名称自动生成的；例如，第一次写入\fontspec{Times New Roman}将生成内部字体系列名称为“TimesNewRoman(1)”。请注意，您不应依赖所生成的名称。

In certain cases it is desirable to be able to choose this internal font family name so it can be re-used elsewhere for interacting with other packages that use the LaTeX's font selection interface; an example might be

在某些情况下，希望能够选择此内部字体系列名称，以便在其他使用 LaTeX 字体选择接口的包中重新使用；例如：

```
\usepackage{fancyvrb}
\fvset{fontfamily=myverbatimfont}
```

To select a font for use in this way in fontspec use the `NFSSFamily` feature:[3]

要在 fontspec 宏包中选择要以此方式使用的字体，请使用 `NFSSFamily` 功能：[4]

```
\newfontfamily\verbatimfont{Inconsolata}[NFSSFamily=myverbatimfont]
```

It is then possible to write commands such as:

然后可以编写以下命令：

```
\fontfamily{myverbatimfont}\selectfont
```

which is essentially the same as writing `\verbatimfont`, or to go back to the orginal example:

这本质上与编写`\verbatimfont` 相同，或者返回到最初的示例：

```
\fvset{fontfamily=myverbatimfont}
```

Only use this feature when necessary; the in-built font switching commands that fontspec generates (such as `\verbatimfont` in the example above) are recommended in all other cases.

仅在必要时使用此功能；对于所有其他情况，建议使用 fontspec 生成的内置字体切换命令（例如上面的`\verbatimfont`）。

If you don't wish to explicitly set the NFSS family but you would like to know what it is, an alternative mechanism for package writers is introduced as part of the fontspec programming interface; see the function `\fontspec_set_family:Nnn` for details (Section 5 on page 89).

如果您不希望显式设置 NFSS 系列，但您想知道它是什么，请为包作者引入的另一种机制作为 fontspec 编程接口的一部分；有关详细信息，请参见`\fontspec_set_family:Nnn`函数（Section 5 on page 89）。

4.3 Choosing additional NFSS font faces
选择其他 NFSS 字体面

LaTeX's font selection scheme (NFSS) is more flexible than the fontspec interface discussed up until this point. It assigns to each font face a *family* (discussed above), a *series* such as bold or light or condensed, and a *shape* such as italic or slanted or small caps. The fontspec features such as `BoldFont` and so on all assign faces for the default series and shapes of the NFSS, but it's not uncommon to have font families that have multiple weights and shapes and so on.

LaTeX 的字体选择方案（NFSS）比到目前为止讨论的 fontspec 界面更灵活。它为每个字体面分配了一个系列（如粗体或轻体或压缩），一个字形（如斜体或倾斜或小型大写）。像

[3] Thanks to Luca Fascione for the example and motivation for finally implementing this feature.
[4] 感谢 Luca Fascione 提供的示例和激励，终于实现了此功能。

BoldFont 之类的 fontspec 功能都为 NFSS 的默认系列和字形分配了面，但常见的字体系列具有多个重量和字形等。

If you set up a regular font family with the 'standard four' (upright, bold, italic, and bold italic) shapes and then want to use, say, a light font for a certain document element, many users will be perfectly happy to use \newfontface\⟨*switch*⟩ and use the resulting font \⟨*switch*⟩. In other cases, however, it is more convenient or even necessary to load additional fonts using additional NFSS specifiers.

如果你使用标准四种字体（正体，粗体，斜体和粗斜体）设置了一个常规字体系列，然后想在某个文档元素中使用轻型字体，很多用户将会非常高兴使用\newfontface\⟨*switch*⟩并使用由\⟨*switch*⟩产生的字体。然而，在其他情况下，使用额外的 NFSS 说明符加载其他字体更加方便甚至是必要的。

```
FontFace = {⟨series⟩}{⟨shape⟩} { Font = ⟨font name⟩ , ⟨features⟩ }
FontFace = {⟨series⟩}{⟨shape⟩}{⟨font name⟩}
```

The font thus specified will inherit the font features of the main font, with optional additional ⟨*features*⟩ as requested. (Note that the optional {⟨*features*⟩} argument is still surrounded with curly braces.) Multiple FontFace commands may be used in a single declaration to specify multiple fonts. As an example:

因此指定的字体将继承主字体的字体特性，并可按请求选择可选的额外⟨*features*⟩。(注意可选的 {⟨*features*⟩} 参数仍然用花括号括起来。) 可以在单个声明中使用多个 FontFace 命令来指定多个字体。如下例所示：

```
\setmainfont{font1.otf}[
   FontFace = {c}{\shapedefault}{ font2.otf } ,
   FontFace = {c}{m}{ Font = font3.otf , Color = red }
  ]
```

Writing \fontseries{c}\selectfont will result in font2 being selected, which then followed by \fontshape{m}\selectfont will result in font3 being selected (in red). A font face that is defined in terms of a different series but an upright shape (\shapedefault, as shown above) will attempt to find a matching small caps feature and define that face as well. Conversely, a font face defined in terms of a non-standard font shape will not.

写上\fontseries{c}\selectfont 将会选择font2，然后再写上\fontshape{m}\selectfont 将会选择font3（以红色显示）。以不同系列定义但直立形状（如上所示的\shapedefault）的字体将尝试查找匹配的小型大写字母特性，并将该字体也定义为小型大写字体。相反，以非标准字体形状定义的字体面将不会匹配小型大写字体。

There are some standards for choosing shape and series codes; the LaTeX 2_ε font selection guide[5] has a comprehensive listing.

选择形状和系列代码的标准有一些规定；LaTeX 2_ε 字体选择指南[6]中有全面的列表。

The FontFace command also interacts properly with the SizeFeatures command as follows: (nonsense set of font selection choices)

FontFace 命令还与 SizeFeatures 命令相互作用，如下所示：(无意义的字体选择集合)

[5] texdoc fntguide
[6] texdoc fntguide

```
  FontFace = {c}{n}{
    Font = Times ,
    SizeFeatures = {
      { Size =   -10 , Font = Georgia } ,
      { Size = 10-15}                 , % default "Font = Times"
      { Size = 15-   , Font = Cochin  } ,
    },
  },
```

Note that if the first `Font` feature is omitted then each size needs its own inner `Font` declaration.

请注意，如果省略第一个 `Font` 特性，则每个大小需要其自己的内部 `Font` 声明。

4.3.1 An example for `\strong`
关于`\strong`的示例

If you wanted to set up a font family to allow nesting of the `\strong` to easily access increasing font weights, you might use a declaration along the following lines:

如果您想设置一个字体系列，以允许嵌套`\strong`以便轻松访问不断增加的字重，那么您可以使用以下声明：

```
\setmonofont{SourceCodePro}[
  Extension = .otf ,
  UprightFont = *-Light ,
  BoldFont = *-Regular ,
  FontFace = {k}{n}{*-Black} ,
]
\strongfontdeclare{\bfseries,\fontseries{k}\selectfont}
```

Further 'syntactic sugar' is planned to make this process somewhat easier.

计划进一步提供“语法糖”，以使这个过程更加容易。

4.4 Math(s) fonts
数学字体

When `\setmainfont`, `\setsansfont` and `\setmonofont` are used in the preamble, they also define the fonts to be used in maths mode inside the `\mathrm`-type commands. This only occurs in the preamble because LaTeX freezes the maths fonts after this stage of the processing. The fontspec package must also be loaded after any maths font packages (*e.g.*, euler) to be successful. (Actually, it is *only* euler that is the problem.[7])

当在导言区使用`\setmainfont`、`\setsansfont`和`\setmonofont`时，它们也定义了用于在`\mathrm`型命令内部使用的数学模式中使用的字体。这只发生在导言区，因为 LaTeX 在处理后会冻结数学字体。要成功，fontspec 包也必须在任何数学字体包（例如，euler）之

[7] Speaking of euler, if you want to use its `[mathbf]` option, it won't work, and you'll need to put this after fontspec is loaded instead: `\AtBeginDocument{\DeclareMathAlphabet\mathbf{U}{eur}{b}{n}}`

后加载。(实际上，只有 euler 才是问题。[8])

Note that fontspec will not change the font for general mathematics; only the upright and bold shapes will be affected. To change the font used for the mathematical symbols, see either the mathspec package or the unicode-math package.

请注意，fontspec 不会更改一般数学的字体；只有正体和粗体形状会受到影响。要更改用于数学符号的字体，请参见 mathspec 包或 unicode-math 包。

Note that you may find that loading some maths packages won't be as smooth as you expect since fontspec (and XeTeX in general) breaks many of the assumptions of TeX as to where maths characters and accents can be found. Contact me if you have troubles, but I can't guarantee to be able to fix any incompatibilities. The Lucida and Euler maths fonts should be fine; for all others keep an eye out for problems.

请注意，您可能会发现加载某些数学包不如您所期望的那样顺利，因为 fontspec（以及 XeTeX 总体）打破了 TeX 对数学字符和重音的位置的许多假设。如果您遇到问题，请联系我，但我不能保证能够解决任何不兼容性。Lucida 和 Euler 数学字体应该没问题；对于所有其他字体，请留意问题。

```
\setmathrm{⟨font name⟩}[⟨font features⟩]
\setmathsf{⟨font name⟩}[⟨font features⟩]
\setmathtt{⟨font name⟩}[⟨font features⟩]
\setboldmathrm{⟨font name⟩}[⟨font features⟩]
```

However, the default text fonts may not necessarily be the ones you wish to use when typesetting maths (especially with the use of fancy ligatures and so on). For this reason, you may optionally use the commands above (in the same way as our other \fontspec-like commands) to explicitly state which fonts to use inside such commands as \mathrm. Additionally, the \setboldmathrm command allows you define the font used for \mathrm when in bold maths mode (which is activated with, among others, \boldmath).

然而，默认的文本字体不一定是您在排版数学时希望使用的字体（尤其是在使用花式连字等特殊效果时）。因此，您可以选择使用上述命令（与我们的其他类似于 \fontspec命令的方式相同）来明确指定在 \mathrm等命令中使用哪些字体。另外，\setboldmathrm命令允许您定义在粗体数学模式下使用的 \mathrm字体（该模式通过诸如 \boldmath等命令激活）。

For example, if you were using Optima with the Euler maths font, you might have this in your preamble:

例如，如果您正在使用 Optima 字体和 Euler 数学字体，则可以在导言区中使用以下代码：

```
\usepackage{mathpazo}
\usepackage{fontspec}
\setmainfont{Optima}
\setmathrm{Optima}
\setboldmathrm[BoldFont={Optima ExtraBlack}]{Optima Bold}
```

[8]谈到 euler, 如果您想使用它的 [mathbf] 选项，它将不起作用，您需要在加载 fontspec 之后将其放在以下位置: \AtBeginDocument{\DeclareMathAlphabet\mathbf{U}{eur}{b}{n}}

These commands are compatible with the unicode-math package. Having said that, unicode-math also defines a more general way of defining fonts to use in maths mode, so you can ignore this subsection if you're already using that package.

这些命令与 unicode-math 宏包兼容。话虽如此，unicode-math 也定义了一种更通用的定义数学模式下要使用的字体的方法，因此如果您已经在使用该宏包，则可以忽略本小节。

5 Miscellaneous font selecting details
其他字体选择细节

The optional argument — from v2.4
可选参数——自 **v2.4** 起 For the first decade of fontspec's life, optional font features were selected with a bracketed argument before the font name, as in:

在 fontspec 的前十年中，可选字体特性是通过字体名称之前的带方括号的参数来选择的，例如：

```
\setmainfont[
  lots and lots ,
  and more and more ,
  an excessive number really ,
  of font features could go here
]{myfont.otf}
```

This always looked like ugly syntax to me, because the most important detail — the name of the font — was tucked away at the end. The order of these arguments has now been reversed:

我一直认为这看起来很丑陋，因为最重要的细节——字体名称——被藏在最后面。现在这些参数的顺序已经被颠倒过来了：

```
\setmainfont{myfont.otf}[
  lots and lots ,
  and more and more ,
  an excessive number really ,
  of font features could go here
]
```

I hope this doesn't cause any problems.

希望这不会引起任何问题。

1. Backwards compatibility has been preserved, so either input method works.
   向后兼容性已经得到保留，因此任何一种输入方式都可以。
2. In fact, you can write
   实际上，如果您真的想这样做，可以编写以下代码：

   ```
   \fontspec[Ligatures=Rare]{myfont.otf}[Color=red]
   ```

   if you really felt like it and both sets of features would be applied.

   那么两组特性都会被应用。

Spaces
空格 `\fontspec` and `\addfontfeatures` ignore trailing spaces as if it were a 'naked' control sequence; *e.g.*, '`M. \fontspec{...} N`' and '`M. \fontspec{...}N`' are the same.

`\fontspec`和`\addfontfeatures`像一个裸控制序列一样忽略尾随空格;例如,`M. \fontspec{...} N`和`M. \fontspec{...}N`是相同的。

Part III

Selecting font features
选择字体特性

The commands discussed so far such as `\fontspec` each take an optional argument for accessing the font features of the requested font. Commands are provided to set default features to be applied for all fonts, and even to change the features that a font is presently loaded with. Different font shapes can be loaded with separate features, and different features can even be selected for different sizes that the font appears in. This part discusses these options.

到目前为止，我们已经介绍了一些命令，例如`\fontspec`，每个命令都有一个可选参数，用于访问请求字体的字体特性。提供了一些命令来设置应用于所有字体的默认特性，甚至可以更改当前已加载的字体的特性。不同的字体形状可以使用不同的特性加载，甚至可以为字体出现的不同大小选择不同的特性。本部分将讨论这些选项。

1 Default settings
默认设置

`\defaultfontfeatures{`⟨*font features*⟩`}`

It is sometimes useful to define font features that are applied to every subsequent font selection command. This may be defined with the `\defaultfontfeatures` command, shown in Example 5. New calls of `\defaultfontfeatures` overwrite previous ones, and defaults can be reset by calling the command with an empty argument.

有时定义应用于后续所有字体选择命令的字体特性很有用。如 Example 5所示，可以使用`\defaultfontfeatures`命令定义它。新调用`\defaultfontfeatures`将覆盖先前的调用，可以通过使用空参数调用该命令来重置默认值。

`\defaultfontfeatures[`⟨*font name*⟩`]{`⟨*font features*⟩`}`

Example 5: A demonstration of the `\defaultfontfeatures` command.
`\defaultfontfeatures`命令的演示。

—Example graphic not generated—

```
\fontspec{texgyreadventor-regular.otf}
Some default text 0123456789 \\
\defaultfontfeatures{
   Numbers=OldStyle, Color=888888
}
\fontspec{texgyreadventor-regular.otf}
Now grey, with old-style figures:
0123456789
```

Default font features can be specified on a per-font and per-face basis by using the optional argument to `\defaultfontfeatures` as shown.

可以使用`\defaultfontfeatures`的可选参数来指定每个字体和每个面的默认字体特性，如下所示：

```
\defaultfontfeatures[texgyreadventor-regular.otf]{Color=blue}
\setmainfont{texgyreadventor-regular.otf}% will be blue
```

Multiple fonts may be affected by using a comma separated list of font names.

可以使用逗号分隔的字体名称列表来影响多个字体。

`\defaultfontfeatures[⟨\font-switch⟩]{⟨font features⟩}`

New in v2.4. Defaults can also be applied to symbolic families such as those created with the `\newfontfamily` command and for `\rmfamily`, `\sffamily`, and `\ttfamily`:

自 **v2.4** 起。还可以将默认值应用于符号家族，例如使用`\newfontfamily` 命令创建的家族，以及`\rmfamily`，`\sffamily` 和`\ttfamily`:

```
\defaultfontfeatures[\rmfamily,\sffamily]{Ligatures=TeX}
\setmainfont{texgyreadventor-regular.otf}% will use standard TeX ligatures
```

The line above to set TEX-like ligatures is now activated by *default* in `fontspec.cfg`. To reset default font features, simply call the command with an empty argument:

以上一行代码用于设置 TEX 连字号，现在默认情况下在 `fontspec.cfg` 中已激活。要重置默认字体特性，请使用空参数调用该命令：

```
\defaultfontfeatures[\rmfamily,\sffamily]{}
\setmainfont{texgyreadventor-regular.otf}% will no longer use standard TeX ligatures
```

`\defaultfontfeatures+{⟨font features⟩}`
`\defaultfontfeatures+[⟨font name⟩]{⟨font features⟩}`

New in v2.4. Using the + form of the command appends the ⟨*font features*⟩ to any already-selected defaults.

使用 + 形式的命令将 ⟨字体特性⟩ 添加到已选中的默认字体特性中。

2 Working with the currently selected features
使用当前所选功能

`\IfFontFeatureActiveTF{⟨font feature⟩}{⟨true code⟩}{⟨false code⟩}`

This command queries the currently selected font face and executes the appropriate branch based on whether the ⟨*font feature*⟩ as specified by fontspec is currently active.

该命令查询当前选择的字体并根据 fontspec 所指定的⟨字体特征⟩是否当前活动来执行相应的分支。

For example, the following will print 'True':

例如，以下代码将打印“True”：

```
\setmainfont{texgyrepagella-regular.otf}[Numbers=OldStyle]
\IfFontFeatureActiveTF{Numbers=OldStyle}{True}{False}
```

Note that there is no way for fontspec to know what the default features of a font will be. For example, by default the `texgyrepagella` fonts use lining numbers. But in the following example, querying for lining numbers returns false since they have not been explicitly requested:

请注意，fontspec 无法知道字体的默认功能。例如，默认情况下，`texgyrepagella` 字体使用的是 lining 数字。但是，在以下示例中，查询 lining 数字会返回 false，因为它们没有被明确请求：

```
\setmainfont{texgyrepagella-regular.otf}
\IfFontFeatureActiveTF{Numbers=Lining}{True}{False}
```

Please note: At time of writing this function only supports OpenType fonts; AAT/Graphite fonts under the XeTeX engine are not supported.

请注意：在编写本文时，此函数仅支持 OpenType 字体。不支持在 XeTeX 引擎下使用的 AAT/Graphite 字体。

`\addfontfeatures{`⟨*font features*⟩`}`

This command allows font features in an entire font family to be changed without knowing what features are currently selected or even what font family is being used. A good example of this could be to add a hook to all tabular material to use monospaced numbers, as shown in Example 6. If you attempt to *change* an already-selected feature, fontspec will try to de-activate any features that clash with the new ones. *E.g.*, the following two invocations are mutually exclusive:

该命令允许更改整个字体系列中的字体特征，而无需知道当前选择了哪些特征，甚至不知道正在使用哪个字体系列。这样做的一个好例子是为所有制表符材料添加钩子以使用等宽数字，如 Example 6所示。如果您尝试“更改”已选择的功能，则 fontspec 将尝试停用与新功能冲突的任何功能。例如，以下两个调用是相互排斥的：

```
\addfontfeature{Numbers=OldStyle}...
\addfontfeature{Numbers=Lining}...
123
```

Since `Numbers=Lining` comes last, it takes precedence and deactivates the call `Numbers=OldStyle`.

因为 `Numbers=Lining` 在后面，它具有优先权并停用了 `Numbers=OldStyle` 调用。

If you wish to apply the change to only one of the fonts of a family (say, italics only) you can write

如果您只希望将更改应用于字体系列的一个字体（例如，仅斜体字体），则可以编写：

```
\addfontfeature{ItalicFeatures={Numbers=Lowercase}}
```

`\addfontfeature` This command may also be executed under the alias `\addfontfeature`.

该命令也可以通过别名`\addfontfeature`来执行。

Example 6: A demonstration of the `\addfontfeatures` command.
`\addfontfeatures` 命令的演示。

```
\fontspec{texgyreadventor-regular.otf}%
         [Numbers={Proportional,OldStyle}]
`In 1842, 999 people sailed 97 miles in
 13 boats. In 1923, 111 people sailed 54
 miles in 56 boats.'            \bigskip

{\addfontfeatures{Numbers={Monospaced,Lining}}
\begin{tabular}{@{} cccc @{}}
          Year & People & Miles & Boats \\
  \hline  1842 &  999   &  75   &  13   \\
          1923 &  111   &  54   &  56
\end{tabular}}
```

—Example graphic not generated—

2.1 Priority of feature selection
功能选择的优先级

Features defined with `\addfontfeatures` override features specified by `\fontspec`, which in turn override features specified by `\defaultfontfeatures`. If in doubt, whenever a new font is chosen for the first time, an entry is made in the transcript (`.log`) file displaying the font name and the features requested.

使用`\addfontfeatures`定义的功能将覆盖由`\fontspec`指定的功能，而`\fontspec`又将覆盖`\defaultfontfeatures`指定的功能。如果有疑问，每当首次选择新字体时，都会在转录（`.log`）文件中显示字体名称和请求的功能。

3 Different features for different font shapes
不同字形的不同功能

```
BoldFeatures={⟨features⟩}
ItalicFeatures={⟨features⟩}
BoldItalicFeatures={⟨features⟩}
SlantedFeatures={⟨features⟩}
BoldSlantedFeatures={⟨features⟩}
SwashFeatures={⟨features⟩}
BoldSwashFeatures={⟨features⟩}
SmallCapsFeatures={⟨features⟩}
UprightFeatures={⟨features⟩}
```

It is entirely possible that separate fonts in a family will require separate options; *e.g.*, Hoefler Text Italic contains various swash feature options that are completely unavailable in the upright shapes.

同一字体系列中的不同字体可能需要不同的选项；例如，Hoefler Text Italic 包含在正体中完全无法使用的各种草书功能选项。

The font features defined at the top level of the optional `\fontspec` argument are applied to *all* shapes of the family. Using the `xxFeatures` options shown above, separate font features may be defined to their respective shapes *in addition* to, and with precedence over, the 'global' font features. See Example 7.

在可选的`\fontspec` 参数的顶层定义的字体功能将应用于该系列的所有字形。使用上面显示的 `xxFeatures` 选项，可以分别为它们各自的字形另外定义字体功能，并具有优先权，超过“全局”的字体功能。请参阅 Example 7。

Note that because most fonts include their small caps glyphs within the main font, features specified with `SmallCapsFeatures` are applied *in addition* to any other shape-specific features as defined above, and hence `SmallCapsFeatures` can be nested within `ItalicFeatures` and friends. Every combination of upright, italic, bold, (etc.), and small caps can thus be assigned individual features, as shown in the somewhat ludicrous Example 8.

请注意，由于大多数字体都在主字体中包含了其小型大写字母，因此使用 `SmallCapsFeatures` 指定的特性将会在上面定义的任何其他形状特性之外被应用，因此 `SmallCapsFeatures` 可以嵌套在 `ItalicFeatures` 等特性之内。因此，每种直立、斜体、粗体等与小型大写字母相结合的组合都可以分配个别特性，如在有些荒谬的示例 Example 8 中所示。

4 Selecting fonts from TrueType Collections (TTC files) 从 TrueType 集合（TTC 文件）中选择字体

TrueType Collections are multiple fonts contained within a single file. Each font within a collection must be explicitly chosen using the `FontIndex` command. Since TrueType Collections are often used to contain the italic/bold shapes in a family, fontspec automatically selects the italic, bold, and bold italic fontfaces from the same file. For example, to load the macOS system font Optima:

TrueType Collections 是包含在单个文件中的多个字体。必须使用 `FontIndex` 命令显式选择集合中的每个字体。由于 TrueType 集合通常用于包含字体系列中的斜体/粗体形状，fontspec 会自动从同一文件中选择斜体、粗体和粗斜体字体。例如，要加载 macOS 系统字体 Optima：

```
\setmainfont{Optima.ttc}[
  Path = /System/Library/Fonts/ ,
  UprightFeatures    = {FontIndex=0} ,
  BoldFeatures       = {FontIndex=1} ,
  ItalicFeatures     = {FontIndex=2} ,
```

Example 7: Features for, say, just italics.
例如，只针对斜体字的功能。

—Example graphic not generated—

```
\fontspec{EBGaramond-Regular.otf}%
  [ItalicFont=EBGaramond-Italic.otf]
\itshape Don't Ask Victoria! \\
\addfontfeature{ItalicFeatures={Style=Swash}}
Don't Ask Victoria! \\
```

Example 8: An example of setting the `SmallCapsFeatures` separately for each font shape.
为每个字体形状分别设置 `SmallCapsFeatures` 的示例。

```
\fontspec{texgyretermes}[
    Extension = {.otf},
    UprightFont = {*-regular}, ItalicFont = {*-italic},
    BoldFont = {*-bold}, BoldItalicFont = {*-bolditalic},
    UprightFeatures={Color = 2200022,
         SmallCapsFeatures = {Color=115511}},
     ItalicFeatures={Color = 2244FF,
         SmallCapsFeatures = {Color=112299}},
       BoldFeatures={Color = FF4422,
         SmallCapsFeatures = {Color=992211}},
 BoldItalicFeatures={Color = 888844,
         SmallCapsFeatures = {Color=444422}},
         ]
Upright {\scshape Small Caps}\\
\itshape Italic {\scshape Italic Small Caps}\\
\upshape\bfseries Bold {\scshape Bold Small Caps}\\
\itshape Bold Italic {\scshape Bold Italic Small Caps}
```

—Example graphic not generated—

```
  BoldItalicFeatures = {FontIndex=3} ,
]
```

Support for TrueType Collections has only been tested in XeTeX, but should also work with an up-to-date version of LuaTeX and the luaotfload package.

对 TrueType 集合的支持仅在 XeTeX 中进行了测试，但应该也可以在最新版本的 LuaTeX和 luaotfload 包中正常工作。

5 Different features for different font sizes
针对不同字体大小的不同特性

```
SizeFeatures = {
  ...
  { Size = ⟨size range⟩, ⟨font features⟩ },
  { Size = ⟨size range⟩, Font = ⟨font name⟩, ⟨font features⟩ },
  ...
}
```

The `SizeFeature` feature is a little more complicated than the previous features discussed. It allows different fonts and different font features to be selected for a given font family as the point size varies.

`SizeFeature` 特性比前面讨论过的特性要稍微复杂一些。它允许在字体系列的给定字体大小变化时选择不同的字体和不同的字体特性。

It takes a comma separated list of braced, comma separated lists of features for each size

range. Each sub-list must contain the `Size` option to declare the size range, and optionally `Font` to change the font based on size. Other (regular) fontspec features that are added are used on top of the font features that would be used anyway. A demonstration to clarify these details is shown in Example 9. A less trivial example is shown in the context of optical font sizes in Section 6.6 on page 39.

它接受逗号分隔的分组列表作为输入，每个尺寸范围的分组列表都包含 `Size` 选项来声明尺寸范围，可选的 `Font` 选项来根据尺寸更改字体。其他（常规）fontspec 特性添加在即将使用的字体特性的基础上。为了澄清这些细节，演示如下（详见 Example 9）。在光学字体尺寸的上下文中，显示了一个不太琐碎的示例（详见Section 6.6 on page 39）。

To be precise, the `Size` sub-feature accepts arguments in the form shown in Table 1 on the following page. Braces around the size range are optional. For an exact font size (`Size=X`) font sizes chosen near that size will 'snap'. For example, for size definitions at exactly 11pt and 14pt, if a 12pt font is requested *actually* the 11pt font will be selected. This is a remnant of the past when fonts were designed in metal (at obviously rigid sizes) and later when bitmap fonts were similarly designed for fixed sizes.

具体来说，`Size` 子特性接受如Table 1 on the next page所示形式的参数。尺寸范围周围的花括号是可选的。对于精确的字体大小（`Size=X`），选择接近该大小的字体大小将会“捕捉”。例如，对于恰好为 11pt 和 14pt 的尺寸定义，如果请求 12pt 字体，则实际上会选择 11pt 字体。这是过去的遗物，当时字体是在金属中设计的（显然是刚性尺寸），后来在设计位图字体时也是如此。

If additional features are only required for a single size, the other sizes must still be specified. As in:

如果仅需要单个尺寸的其他特性，则仍必须指定其他尺寸。例如：

```
SizeFeatures={
   {Size=-10,Numbers=Uppercase},
   {Size=10-}}
```

Otherwise, the font sizes greater than 10 won't be defined at all!

否则，大于 10 的字体大小将不会被定义！

Example 9: An example of specifying different font features for different sizes of font with `SizeFeatures`.
使用 `SizeFeatures` 为不同字体尺寸指定不同的字体特性的示例

```
\fontspec{texgyrechorus-mediumitalic.otf}[
  SizeFeatures={
    {Size={-8}, Font=texgyrebonum-italic.otf, Color=AA0000},
    {Size={8-14}, Color=00AA00},
    {Size={14-}, Color=0000AA}} ]
```

—Example graphic not generated—

```
{\scriptsize Small\par} Normal size\par {\Large Large\par}
```

Table 1: Syntax for specifying the size to apply custom font features.
指定应用自定义字体特性的尺寸的语法。

Input	Font size, s
`Size = X-`	$s \geq$ `X`
`Size = -Y`	$s <$ `Y`
`Size = X-Y`	`X` $\leq s <$ `Y`
`Size = X`	$s =$ `X`

Interaction with other features
与其他特性的交互　For `SizeFeatures` to work with `ItalicFeatures`, `BoldFeatures`, etc., and `SmallCapsFeatures`, a strict heirarchy is required:

为了使 `SizeFeatures` 与 `ItalicFeatures`、`BoldFeatures` 等以及 `SmallCapsFeatures` 配合使用，需要严格的层次结构：

```
UprightFeatures =
 {
  SizeFeatures =
   {
    {
     Size = -10,
     Font = ..., % if necessary
     SmallCapsFeatures = {...},
     ... % other features for this size range
    },
    ... % other size ranges
   }
 }
```

Suggestions on simplifying this interface welcome.

欢迎提出简化此接口的建议。

6 Font independent options
与字体无关的选项

Features introduced in this section may be used with any font.

本节介绍的功能可与任何字体配合使用。

6.1 Colour
颜色

`Color` (or `Colour`) uses font specifications to set the colour of the text. You should think of this as the literal glyphs of the font being coloured in a certain way. Notably, this mechanism is different to that of the color/xcolor/hyperref/etc. packages, and in fact us-

ing fontspec commands to set colour will prevent your text from changing colour using those packages at all! (For example, if you set the colour in a `\setmainfont` command, `\color{...}` and related commands, including hyperlink colouring, will no longer have any effect on text in this font.) Therefore, fontspec's colour commands are best used to set explicit colours in specific situations, and the xcolor package is recommended for more general colour functionality.

`Color`（或 `Colour`）使用字体规范来设置文本的颜色。您应该将其视为以某种方式着色的字体的文字。值得注意的是，此机制不同于 color / xcolor / hyperref / 等包的机制，实际上使用 fontspec 命令设置颜色将完全防止您的文本使用这些包之一改变颜色！（例如，如果您在`\setmainfont` 命令中设置颜色，则`\color{...}` 和相关命令（包括超链接颜色）将不再对此字体中的文本产生任何影响。）因此，fontspec 的颜色命令最好用于特定情况下设置明确的颜色，建议使用 xcolor 包来获得更通用的颜色功能。

The colour is defined as a triplet of two-digit Hex RGB values, with optionally another value for the transparency (where `00` is completely transparent and `FF` is opaque.)

颜色定义为一个三位十六进制 RGB 值的三元组，其中可选地包括另一个值表示透明度（其中`00` 为完全透明，`FF` 为不透明）。

Transparency is supported by LuaLaTeX; XeLaTeX with the `xdvipdfmx` driver does not support this feature.

透明度由LuaLaTeX支持；带有 `xdvipdfmx` 驱动程序的 XeLaTeX 不支持此功能。

If you load the xcolor package, you may use any named colour instead of writing the colours in hexadecimal.

如果加载了 xcolor 包，则可以使用任何命名颜色，而无需将颜色写成十六进制。例如：

```
\usepackage{xcolor}
...
\fontspec[Color=red]{Verdana} ...
\definecolor{Foo}{rgb}{0.3,0.4,0.5}
\fontspec[Color=Foo]{Verdana} ...
```

The color package is *not* supported; use xcolor instead.

color 包不被支持，应使用 xcolor 包代替。

You may specify the transparency with a named colour using the `Opacity` feature which takes an decimal from zero to one corresponding to transparent to opaque respectively:

Example 10: Selecting colour with transparency.
选择带透明度的颜色。

—Example graphic not generated—

```
\fontsize{48}{48}
\fontspec{texgyrebonum-bold.otf}
{\addfontfeature{Color=FF0000}W}\kern-0.4ex
{\addfontfeature{Color=0000FF99}S}\kern-0.4ex
{\addfontfeature{Color=DDBB2299}P}\kern-0.5ex
{\addfontfeature{Color=00BB3399}R}
```

您可以使用 Opacity 特性通过指定具有命名颜色的透明度来设置透明度，其取值为从零到一的十进制数，分别对应于透明到不透明：

```
\fontspec[Color=red,Opacity=0.7]{Verdana} ...
```

It is still possible to specify a colour in six-char hexadecimal form while defining opacity in this way, if you like.

如果您愿意，仍然可以以六个十六进制字符的形式指定颜色，并以这种方式定义不透明度。

6.2 Scale
缩放

Scale = ⟨*number*⟩
Scale = MatchLowercase
Scale = MatchUppercase

In its explicit form, Scale takes a single numeric argument for linearly scaling the font, as demonstrated in Example 1.

在显式形式中，Scale 接受一个数字参数，用于线性缩放字体，如 Example 1 所示。

As well as a numerical argument, the Scale feature also accepts options MatchLowercase and MatchUppercase, which will scale the font being selected to match the current default roman font to either the height of the lowercase or uppercase letters, respectively; these features are shown in Example 11. The amount of scaling used in each instance is reported in the .log file.

除了数字参数之外，Scale 特性还接受选项 MatchLowercase 和 MatchUppercase，它们将缩放被选择的字体以匹配当前默认罗马字体的小写或大写字母的高度，分别在 Example 11 中展示。在每种情况下所使用的缩放量会在 .log 文件中报告。

Additional calls to the Scale feature overwrite the settings of the former. If you want to accumulate scale factors (useful perhaps to fine-tune the settings of MatchLowercase), the ScaleAgain feature can be used as many times as necessary. For example:

对 Scale 特性的额外调用将覆盖先前的设置。如果您想累加缩放因子（也许对微调 MatchLowercase 的设置有用），则可以使用 ScaleAgain 特性，其可以使用多次。例如：

```
[ Scale = 1.1 , Scale = 1.2 ]      % -> scale of 1.2
[ Scale = 1.1 , ScaleAgain = 1.2 ] % -> scale of 1.32
```

Example 11: Automatically calculated scale values.
自动计算的缩放值。

—Example graphic not generated—

```
\setmainfont{Georgia}
\newfontfamily\lc[Scale=MatchLowercase]{Verdana}
 The perfect match {\lc is hard to find.}\\
\newfontfamily\uc[Scale=MatchUppercase]{Arial}
 L O G O \uc F O N T
```

Note that when `Scale=MatchLowercase` is used with `\setmainfont`, the new 'main' font of the document will be scaled to match the old default. If you wish to automatically scale all fonts except have the main font use 'natural' scaling, you may write

请注意，当使用 `Scale=MatchLowercase` 与 `\setmainfont` 结合使用时，文档的新“主”字体将缩放以匹配旧的默认字体。如果您希望自动缩放所有字体，除了主字体使用“自然”缩放之外，可以编写以下内容：

```
\defaultfontfeatures{ Scale = MatchLowercase }
\defaultfontfeatures[\rmfamily]{ Scale = 1}
```

One or both of these lines may be placed into a local `fontspec.cfg` file (see Section 3.3 on page 5) for this behaviour to be effected in your own documents automatically. (Also see Section 1 on page 27 for more information on setting font defaults.)

这两行代码中的一行或者两行可以被放置在本地的`fontspec.cfg`文件中（参见Section 3.3 on page 5），以便这种行为能够自动地在你自己的文档中实现。（关于设置字体默认值的更多信息，请参见Section 1 on page 27。）

6.3 Interword space
单词间距

While the space between words can be varied with the TEX primitive `\spaceskip` command, fontspec also supports changing the interword spacing when a given font is loaded.

虽然单词间的间距可以通过 TEX 原始命令`\spaceskip`进行变化，但是在加载特定字体时，fontspec 还支持更改单词间的间距。

The space in between words in a paragraph will be chosen automatically, and generally will not need to be adjusted. For those times when the precise details are important, the `WordSpace` feature is provided, which takes either a single scaling factor to scale the default value, or a triplet of comma-separated values to scale the nominal value, the stretch, and the shrink of the interword space by, respectively. (`WordSpace={`x`}` is the same as `WordSpace={`x`,`x`,`x`}`.)

段落中单词之间的间距将会自动选择，并且通常不需要进行调整。对于那些重视细节的时候，提供了 `WordSpace` 功能，该功能可以采用单个缩放因子来缩放默认值，或采用逗号分隔的三元组来分别缩放单词间隔的名义值、拉伸值和收缩值。（`WordSpace={`x`}` 等同于`WordSpace={`x`,`x`,`x`}`。）

Note that TEX's optimisations in how it loads fonts means that you cannot use this feature in `\addfontfeatures`.

请注意，TEX 在加载字体时的优化意味着您不能在`\addfontfeatures`中使用此功能。

6.4 Post-punctuation space
标点符号后的间距

If `\frenchspacing` is *not* in effect (which is the default), TEX will allow extra space after some punctuation in its goal of justifying the lines of text.

Example 12: Scaling the default interword space. An exaggerated value has been chosen to emphasise the effects here.

```
\fontspec{texgyretermes-regular.otf}
Some text for our example to take
up some space, and to demonstrate
the default interword space.
\bigskip

\fontspec{texgyretermes-regular.otf}%
  [WordSpace = 0.3]
Some text for our example to take
up some space, and to demonstrate
the default interword space.
```

—Example graphic not generated—

如果\frenchspacing未生效（这是默认设置），TEX 会在对齐文本行的目标中，在一些标点符号后面允许额外的间距。

The PunctuationSpace feature takes a scaling factor by which to adjust the nominal value chosen for the font; this is demonstrated in Example 13. Note that PunctuationSpace=0 is *not* equivalent to \frenchspacing, although the difference will only be apparent when a line of text is under-full.

PunctuationSpace 功能采用缩放因子来调整所选择的字体的名义值；这在 Example 13中得到了说明。请注意，PunctuationSpace=0 与\frenchspacing不同，尽管差异只有在文本行未填满时才会显现出来。

Note that TEX's optimisations in how it loads fonts means that you cannot use this feature in \addfontfeatures.

请注意，TEX 在加载字体时的优化意味着您不能在\addfontfeatures中使用此功能。

6.5 The hyphenation character
连字字符

The letter used for hyphenation may be chosen with the HyphenChar feature. With one exception (HyphenChar = None), this is a XHTEX-only feature since LuaTEX cannot set the

Example 13: Scaling the default post-punctuation space.

```
\nonfrenchspacing
\fontspec{texgyreschola-regular.otf}
 Letters, Words. Sentences.          \par
\fontspec{texgyreschola-regular.otf}[PunctuationSpace=2]
 Letters, Words. Sentences.          \par
\fontspec{texgyreschola-regular.otf}[PunctuationSpace=0]
 Letters, Words. Sentences.
```

—Example graphic not generated—

hyphenation character on a per-font basis; see its \prehyphenchar primitive for further details.

可以使用 HyphenChar 功能选择用于连字的字母。除了 HyphenChar = None 之外，这是仅限于 XeTeX 的功能，因为LuaTeX不能根据字体分别设置连字符；请参见其\prehyphenchar原语以获取更多信息。

HyphenChar takes three types of input, which are chosen according to some simple rules. If the input is the string None, then hyphenation is suppressed for this font.

HyphenChar 接受三种类型的输入，这些输入根据一些简单的规则进行选择。如果输入为字符串 None，则会禁止使用此字体的连字。

As part of fontspec.cfg, the default monospaced family (e.g., \ttfamily) is set up to automatically set HyphenChar = None.

作为 fontspec.cfg 的一部分，设置默认的等宽字体族（例如，\ttfamily）以自动设置 HyphenChar = None。

If the input is a single character, then this character is used. Finally, if the input is longer than a single character it must be the UTF-8 slot number of the hyphen character you desire.

如果输入是单个字符，则使用该字符。最后，如果输入不止一个字符，则必须是您想要的连字符的 UTF-8 槽编号。

Note that TeX's optimisations in how it loads fonts means that you cannot use this feature in \addfontfeatures.

请注意，TeX 在加载字体方面的优化意味着您无法在\addfontfeatures中使用此功能。

6.6 Optical font sizes
光学字体大小

Optically scaled fonts thicken out as the font size decreases in order to make the glyph shapes more robust (less prone to losing detail), which improves legibility. Conversely, at large optical sizes the serifs and other small details may be more delicately rendered.

随着字体大小的减小，光学缩放的字体会变得更厚，以使字形更加健壮（不易丢失细节），从而提高可读性。相反，在大型光学大小中，衬线和其他小细节可能会更加精细。

Example 14: Explicitly choosing the hyphenation character.
显式选择连字字符。

—Example graphic not generated—

```
\def\text{\fbox{\parbox{1.55cm}{%
  EXAMPLE HYPHENATION%
}}\qquad\qquad\null\par\bigskip}

\fontspec{LinLibertine_R.otf}[HyphenChar=None]
\text
\fontspec{LinLibertine_R.otf}[HyphenChar={+}]
\text
```

OpenType fonts with optical scaling can exist in several discrete sizes (in separate font files). When loading fonts by name, XeTeX and LuaTeX engines will attempt to *automatically* load the appropriate font as determined by the current font size. An example of this behaviour is shown in Example 15, in which some larger text is mechanically scaled down to compare the difference for equivalent font sizes.

具有光学缩放的 OpenType 字体可以存在于多个离散的大小（在单独的字体文件中）。在按名称加载字体时，XeTeX 和 LuaTeX 引擎将尝试根据当前字体大小自动加载适当的字体。此行为的示例显示在 Example 15中，其中一些较大的文本被机械缩小以比较相同字体大小的差异。

The `OpticalSize` feature may be used to specify a different optical size. With `OpticalSize` set (Example 16) to zero, no optical size font substitution is performed.

`OpticalSize` 功能可用于指定不同的光学大小。将 `OpticalSize` 设置为零（Example 16）时，不执行光学大小字体替换。

The `SizeFeatures` feature (Section 5 on page 32) can be used to specify exactly which optical sizes will be used for ranges of font size. For example, something like:

`SizeFeatures` 功能（Section 5 on page 32）可用于指定哪些光学大小将用于字体大小范围。例如，类似以下内容的内容：

```
\fontspec{Latin Modern Roman}[
  UprightFeatures = { SizeFeatures = {
    {Size=-10,     OpticalSize=8 },
    {Size= 10-14,  OpticalSize=10},
    {Size= 14-18,  OpticalSize=14},
    {Size=    18-, OpticalSize=18}}}
        ]
```

6.7 Font transformations
字体变换

In rare situations users may want to mechanically distort the shapes of the glyphs in the current font such as shown in Example 17. Please don't overuse these features; they are *not* a good alternative to having the real shapes.

在极少数情况下，用户可能希望机械地扭曲当前字体字形的形状，例如在 Example 17 中所示。请不要过度使用这些功能；它们不是拥有真实形状的良好替代方案。

If values are omitted, their defaults are as shown above.

如果省略了值，则其默认值如上所示。

Example 15: A demonstration of automatic optical size selection.
演示自动光学大小选择。

—Example graphic not generated—

```
\fontspec{Latin Modern Roman}
 Automatic optical size                  \\
\scalebox{0.4}{\Huge
 Automatic optical size}
```

Example 16: Explicit optical size substitution for the Latin Modern Roman family.
Latin Modern Roman 家族的显式光学大小替换。

```
\fontspec{Latin Modern Roman}[OpticalSize=5]
 Latin Modern optical sizes                \\
\fontspec{Latin Modern Roman}[OpticalSize=8]
 Latin Modern optical sizes                \\
\fontspec{Latin Modern Roman}[OpticalSize=12]
 Latin Modern optical sizes                \\
\fontspec{Latin Modern Roman}[OpticalSize=17]
 Latin Modern optical sizes
```

—Example graphic not generated—

Example 17: Articifial font transformations.
人工字体变换。

```
\fontspec{Quattrocento-Regular.otf} \emph{ABCxyz} \quad
\fontspec{Quattrocento-Regular.otf}[FakeSlant=0.2] ABCxyz

\fontspec{Quattrocento-Regular.otf}  ABCxyz \quad
\fontspec{Quattrocento-Regular.otf}[FakeStretch=1.2] ABCxyz

\fontspec{Quattrocento-Regular.otf} \textbf{ABCxyz} \quad
\fontspec{Quattrocento-Regular.otf}[FakeBold=1.5] ABCxyz
```

—Example graphic not generated—

If you want the bold shape to be faked automatically, or the italic shape to be slanted automatically, use the `AutoFakeBold` and `AutoFakeSlant` features. For example, the following two invocations are equivalent:

如果您希望自动伪造粗体形状，或自动倾斜斜体形状，请使用 `AutoFakeBold` 和 `AutoFakeSlant` 功能。例如，以下两个调用是等效的：

```
\fontspec[AutoFakeBold=1.5]{Charis SIL}
\fontspec[BoldFeatures={FakeBold=1.5}]{Charis SIL}
```

If both of the `AutoFake...` features are used, then the bold italic font will also be faked.

如果同时使用 `AutoFake...` 功能，则粗斜体字体也将被伪造。

6.8 Letter spacing
字母间距

Letter spacing, or tracking, is the term given to adding (or subtracting) a small amount of horizontal space in between adjacent characters. It is specified with the `LetterSpace`, which takes a numeric argument, shown in Example 18.

字母间距或字距是指在相邻字符之间添加（或减去）一小段水平间距的术语。它由 `LetterSpace` 指定，它带有一个数值参数，如 Example 18所示。

The letter spacing parameter is a normalised additive factor (not a scaling factor); it is defined as a percentage of the font size. That is, for a 10 pt font, a letter spacing parameter

of '1.0' will add 0.1 pt between each letter.

字母间距参数是一个归一化的加法因子（而不是缩放因子）；它被定义为字体大小的百分比。也就是说，对于一个 10,pt 字体，字母间距参数为 '1.0' 将在每个字母之间添加 0.1,pt。

This functionality is not generally used for lowercase text in modern typesetting but does have historic precedent in a variety of situations. In particular, small amounts of letter spacing can be very useful, when setting small caps or all caps titles. Also see the OpenType `Uppercase` option of the `Letters` feature (3.1.7 on page 56).

这个功能通常不用于现代排版中的小写文本，但在各种情况下具有历史悠久的先例。特别是，当设置小型大写字母或全大写标题时，小量字母间距非常有用。另请参见 `Letters` 功能的 `Uppercase` 选项（3.1.7 on page 56）。

Example 18: The `LetterSpace` feature.

—Example graphic not generated—

```
\fontspec{Didot}
\addfontfeature{LetterSpace=0.0}
USE TRACKING FOR DISPLAY CAPS TEXT \\
\addfontfeature{LetterSpace=2.0}
USE TRACKING FOR DISPLAY CAPS TEXT
```

Part IV

OpenType

1 Introduction
介绍

OpenType fonts (and other 'smart' font technologies such as AAT and Graphite) can change the appearance of text in many different ways. These changes are referred to as font features. When the user applies a feature — for example, small capitals — to a run of text, the code inside the font makes appropriate substitutions and small capitals appear in place of lowercase letters. However, the use of such features does not affect the underlying text. In our small caps example, the lowercase letters are still stored in the document; only the appearance has been changed by the OpenType feature. This makes it possible to search and copy text without difficulty. If the user selected a different font that does not support small caps, the 'plain' lowercase letters would appear instead.

OpenType 字体（和其他“智能”字体技术，如 AAT 和 Graphite）可以以许多不同的方式改变文本的外观。这些更改称为字体特征。当用户将特征（例如小型大写字母）应用于一段文本时，字体内部的代码会进行适当的替换，并在小型大写字母取代小写字母。但是，使用这些特征不会影响底层文本。在我们的小型大写字母示例中，小写字母仍存储在文档中；只有通过 OpenType 特性更改了外观。这使得可以轻松搜索和复制文本。如果用户选择不支持小型大写字母的不同字体，则“普通”的小写字母将出现。

Some OpenType features are required to support particular scripts, and these features are often applied automatically. The Indic scripts, for example, often require that characters be reshaped and reordered after they are typed by the user, in order to display them in the traditional ways that readers expect. Other features can be applied to support a particular language. The Junicode font for medievalists uses by default the Old English shape of the letter thorn, while in modern Icelandic thorn has a more rounded shape. If a user tags some text as being in Icelandic, Junicode will automatically change to the Icelandic shape through an OpenType feature that localises the shapes of letters.

一些 OpenType 特性是必需的，以支持特定的脚本，并且这些特性通常会自动应用。例如，印度脚本经常要求用户在输入字符后进行重塑和重新排序，以便以读者期望的传统方式显示它们。其他特性可用于支持特定语言。中世纪学者使用的 Junicode 字体默认使用字母 thorn 的古英语形状，而现代冰岛语的 thorn 具有更圆润的形状。如果用户将一些文本标记为冰岛语，则 Junicode 将通过本地化字母形状的 OpenType 特性自动更改为冰岛语形状。

There are a large group of OpenType features, designed to support high quality typography a multitude of languages and writing scripts. Examples of some font features have already been shown in previous sections; the complete set of OpenType font features supported by fontspec is described below in Section 3.

有一个大的 OpenType 特性组，旨在支持众多语言和书写脚本的高质量排版。一些字体特性的示例已在以前的章节中显示；fontspec 支持的完整 OpenType 字体特性集在 Section 3 中进行了描述。

The OpenType specification provides four-letter codes (e.g., `smcp` for small capitals) for each feature. The four-letter codes are given below along with the fontspec names for various features, for the benefit of people who are already familiar with OpenType. You can ignore the codes if they don't mean anything to you.

OpenType 规范为每个特性提供四个字母的代码（例如 `smcp` 表示小型大写字母）。这些四字代码以及各种特性的 fontspec 名称如下所示，以方便熟悉 OpenType 的人使用。如果这些代码对您没有意义，则可以忽略它们。

1.1 How to select font features
如何选择字体特性

Font features are selected by a series of ⟨*feature*⟩=⟨*option*⟩ selections. Features are (usually) grouped logically; for example, all font features relating to ligatures are accessed by writing `Ligatures={...}` with the appropriate argument(s), which could be `TeX`, `Rare`, etc., as shown below in 3.1.8.

字体特征是通过一系列⟨*feature*⟩=⟨*option*⟩选择来选择的。这些特征通常按逻辑分组；例如，所有与连字有关的字体特征都可以通过编写`Ligatures={...}`来访问相应的参数（如`TeX`、`Rare`等），如3.1.8中所示。

Multiple options may be given to any feature that accepts non-numerical input, although doing so will not always work. Some options will override others in generally obvious ways; `Numbers={OldStyle,Lining}` doesn't make much sense because the two options are mutually exclusive, and XeTeX will simply use the last option that is specified (in this case using `Lining` over `OldStyle`).

可以给任何接受非数值输入的特征提供多个选项，尽管这样做并不总是有效的。某些选项会以通常很明显的方式覆盖其他选项；`Numbers={OldStyle,Lining}`并不合理，因为这两个选项是互斥的，XeTeX 将仅使用指定的最后一个选项（在本例中使用 `Lining` 而非 `OldStyle`）。

If a feature or an option is requested that the font does not have, a warning is given in the console output. As mentioned in Section 3.4 on page 5 these warnings can be suppressed by selecting the `[quiet]` package option.

如果请求字体没有的特征或选项，则会在控制台输出中给出警告。如Section 3.4 on page 5所述，可以通过选择 `[quiet]` 包选项来抑制这些警告。

1.2 How do I know what font features are supported by my fonts?
我如何知道我的字体支持哪些字体特征？

Although I've long desired to have a feature within fontspec to display the OpenType features within a font, it's never been high on my priority list. One reason for that is the existence of the document `opentype-info.tex`, which is available on CTAN or typing `kpsewhich opentype-info.tex` in a Terminal window. Make a copy of this file and place it somewhere convenient. Then open it in your regular TeX editor and change the font name to the font you'd like to query; after running through plain XeTeX, the output PDF will look something like this:

虽然我长期以来一直希望在 fontspec 中有一个显示字体的 OpenType 特征的功能，但这从未成为我的优先事项之一。其中一个原因是存在名为`opentype-info.tex` 的文档，可以在 CTAN 上获取或在终端窗口中键入`kpsewhich opentype-info.tex`。复制该文件并将其放在方便的位置。然后在常规的 TEX 编辑器中打开它，并将字体名称更改为您要查询的字体；经过普通的 XETEX 运行后，输出的 PDF 将类似于以下内容：

OpenType Layout features found in '[Asana-Math.otf]'

script = 'DFLT'

language = ⟨default⟩

features = 'onum' 'salt' 'kern'

script = 'cher'

language = ⟨default⟩

features = 'onum' 'salt' 'kern'

script = 'grek'

language = ⟨default⟩

features = 'onum' 'salt' 'ssty' 'kern'

script = 'latn'

language = ⟨default⟩

features = 'dtls' 'onum' 'salt' 'ssty' 'kern'

script = 'math'

language = ⟨default⟩

features = 'dtls' 'onum' 'salt' 'ssty' 'kern'

I intentionally picked a font above that by design contains few font features; 'regular' text fonts such as Latin Modern Roman contain many more, and I didn't want to clutter up the document too much. After finding the scripts, languages, and features contained within the font, you'll then need to cross-check the OpenType tags with the 'logical' names used by fontspec.

我故意选择了一个设计上包含很少字体特征的字体；例如，拉丁现代罗马体包含更多字体特征，而我不想使文档太混乱。在找到脚本、语言和字体中包含的特性之后，您需要使用 fontspec 使用的“逻辑”名称交叉检查 OpenType 标记。

otfinfo Alternatively, and more simply, you can use the command line tool `otfinfo`, which is distributed with TEXLive. Simply type in a Terminal window, say:

另外，更简单的方法是使用命令行工具`otfinfo`，它与 TEXLive 一起分发。只需在终端窗口中输入：

```
otfinfo -f `kpsewhich lmromandunh10-oblique.otf`
```

which results in:

```
aalt        Access All Alternates
cpsp        Capital Spacing
dlig        Discretionary Ligatures
frac        Fractions
kern        Kerning
liga        Standard Ligatures
lnum        Lining Figures
onum        Oldstyle Figures
pnum        Proportional Figures
size        Optical Size
tnum        Tabular Figures
zero        Slashed Zero
```

2 OpenType scripts and languages OpenType 脚本和语言

Fonts that include glyphs for various scripts and languages may contain different font features for the different character sets and languages they support, and different font features may behave differently depending on the script or language chosen. When multilingual fonts are used, it is important to select which language they are being used for, and more importantly what script is being used.

包含各种脚本和语言的字形的字体可能针对支持的不同字符集和语言包含不同的字体特征，并且不同的字体特征可能根据所选择的脚本或语言而表现不同。当使用多语言字体时，重要的是要选择使用它们的语言，更重要的是要选择使用哪种脚本。

The 'script' refers to the alphabet in use; for example, both English and French use the Latin script. Similarly, the Arabic script can be used to write in both the Arabic and Persian languages.

"脚本"指使用的字母表；例如，英语和法语都使用拉丁字母表。同样，阿拉伯字母表可以用于写阿拉伯语和波斯语。

The `Script` and `Language` features are used to designate this information. The possible options are tabulated in Table 2 on page 48 and Table 3 on page 49, respectively. When a script or language is requested that is not supported by the current font, a warning is printed in the console output. See Section 2 on page 86 for methods to create new `Script` or `Language` options if required.

`Script` 和 `Language` 特征用于指定此信息。可用选项列在 Table 2 on page 48 和 Table 3 on page 49 中。当请求不受当前字体支持的脚本或语言时，会在控制台输出中打印警告。如果需要，可以参见 Section 2 on page 86 中的方法创建新的 `Script` 或 `Language` 选项。

Because these font features can change which features are able to be selected for the font, the `Script` and `Language` settings are automatically selected by fontspec before all others, and, if XeTeX is being used, will specifically select the `OpenType` renderer for this font, as described in Section 1.2 on page 77.

由于这些字体特征可以改变可以选择的字体特征，因此在所有其他特征之前，fontspec 会自动选择 `Script` 和 `Language` 设置，并且如果使用 XeTeX，则会专门选择此字体的 `OpenType` 渲染器，如 Section 1.2 on page 77 中所述。

OpenType fonts can make available different font features depending on the Script and Language chosen. In addition, these settings can also set up their own font behaviour and glyph selection (one example is differences in style between some of the letters in the alphabet used for Bulgarian, Serbian, and Russian). The fontspec feature `LocalForms = Off` will disable some of these substitutions if desired for some reason. It is important to note that `LocalForms = On` is a default not of fontspec but of the underlying font shaping engines in both XeTeX and LuaTeX/otfload.

OpenType 字体可以根据所选择的脚本和语言提供不同的字体特征。此外，这些设置还可以设置自己的字体行为和字形选择（一个示例是用于保加利亚语、塞尔维亚语和俄语的字母表中某些字母之间的样式差异）。如果需要，fontspec 功能 `LocalForms = Off` 将禁用其中一些替换。重要的是要注意，`LocalForms = On` 不是 fontspec 的默认设置，而是 XeTeX 和 LuaTeX/otfload 中的基础字体成形引擎的默认设置。

2.1 `Script` and `Language` examples
`Script` 和 `Language` 示例

In the examples shown in Example 19, the Code2000 font[9] is used to typeset various input texts with and without the OpenType Script applied for various alphabets. The text is only rendered correctly in the second case; many examples of incorrect diacritic spacing as well as a lack of contextual ligatures and rearrangement can be seen. Thanks to Jonathan Kew, Yves Codet and Gildas Hamel for their contributions towards these examples.

在 Example 19所示的例子中，使用 Code2000 字体[10]对各种字母表的输入文本进行排版，同时应用或不应用 OpenType 脚本。只有在第二种情况下文本才能正确呈现；可以看到许多错误的重音间距以及缺乏上下文连字和重新排列的例子。感谢 Jonathan Kew，Yves Codet 和 Gildas Hamel 对这些示例的贡献。

[9] http://www.code2000.net/
[10] http://www.code2000.net/

Example 19: An example of various Scripts and Languages.
各种脚本和语言的例子。

```
\testfeature{Script=Arabic}{\arabictext}
\testfeature{Script=Devanagari}{\devanagaritext}
\testfeature{Script=Bengali}{\bengalitext}
\testfeature{Script=Gujarati}{\gujaratitext}
\testfeature{Script=Malayalam}{\malayalamtext}
\testfeature{Script=Gurmukhi}{\gurmukhitext}
\testfeature{Script=Tamil}{\tamiltext}
\testfeature{Script=Hebrew}{\hebrewtext}
\def\examplefont{DoulosSILR.ttf}
\testfeature{Language=Vietnamese}{\vietnamesetext}
```

—Example graphic not generated—

Table 2: Defined `Scripts` for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrows (¶).
定义了 OpenType 字体的 `Script`。重命名的名称在相邻的位置用红色段落符号（¶）标出。

Adlam	Glagolitic	Marchen	Rejang
Ahom	Gothic	¶Math	Runic
Anatolian Hieroglyphs	Grantha	¶Maths	Samaritan
Arabic	Greek	Meitei Mayek	Saurashtra
Armenian	Gujarati	Mende Kikakui	Sharada
Avestan	Gurmukhi	Meroitic Cursive	Shavian
Balinese	Hangul Jamo	Meroitic Hieroglyphs	Siddham
Bamum	Hangul	Miao	Sign Writing
Bassa Vah	Hanunoo	Modi	Sinhala
Batak	Hatran	Mongolian	Sora Sompeng
Bengali	Hebrew	Mro	Sumero-Akkadian
Bhaiksuki	¶Hiragana and Katakana	Multani	Cuneiform
Bopomofo	¶Kana	Musical Symbols	Sundanese
Brahmi	Imperial Aramaic	Myanmar	Syloti Nagri
Braille	Inscriptional Pahlavi	¶N'Ko	Syriac
Buginese	Inscriptional Parthian	¶N'ko	Tagalog
Buhid	Javanese	Nabataean	Tagbanwa
Byzantine Music	Kaithi	Newa	Tai Le
Canadian Syllabics	Kannada	Ogham	Tai Lu
Carian	Kayah Li	Ol Chiki	Tai Tham
Caucasian Albanian	Kharosthi	Old Italic	Tai Viet
Chakma	Khmer	Old Hungarian	Takri
Cham	Khojki	Old North Arabian	Tamil
Cherokee	Khudawadi	Old Permic	Tangut
¶CJK	Lao	Old Persian Cuneiform	Telugu
¶CJK Ideographic	Latin	Old South Arabian	Thaana
Coptic	Lepcha	Old Turkic	Thai
Cypriot Syllabary	Limbu	¶Oriya	Tibetan
Cyrillic	Linear A	¶Odia	Tifinagh
Default	Linear B	Osage	Tirhuta
Deseret	Lisu	Osmanya	Ugaritic Cuneiform
Devanagari	Lycian	Pahawh Hmong	Vai
Duployan	Lydian	Palmyrene	Warang Citi
Egyptian Hieroglyphs	Mahajani	Pau Cin Hau	Yi
Elbasan	Malayalam	Phags-pa	
Ethiopic	Mandaic	Phoenician	
Georgian	Manichaean	Psalter Pahlavi	

Table 3: Defined Languages for OpenType fonts. Aliased names are shown in adjacent positions marked with red pilcrows (¶).
定义了 OpenType 字体的 Language 选项。已定义的别名显示在相邻位置，并用红色段落符号 (¶) 标出。

Abaza	Default	Igbo	Koryak	Norway House Cree	Serer
Abkhazian	Dogri	Ijo	Ladin	Nisi	South Slavey
Adyghe	Divehi	Ilokano	Lahuli	Niuean	Southern Sami
Afrikaans	Djerma	Indonesian	Lak	Nkole	Suri
Afar	Dangme	Ingush	Lambani	N'ko	Svan
Agaw	Dinka	Inuktitut	Lao	Dutch	Swedish
Altai	Dungan	Irish	Latin	Nogai	Swadaya Aramaic
Amharic	Dzongkha	Irish Traditional	Laz	Norwegian	Swahili
Arabic	Ebira	Icelandic	L-Cree	Northern Sami	Swazi
Aari	Eastern Cree	Inari Sami	Ladakhi	Northern Tai	Sutu
Arakanese	Edo	Italian	Lezgi	Esperanto	Syriac
Assamese	Efik	Hebrew	Lingala	Nynorsk	Tabasaran
Athapaskan	Greek	Javanese	Low Mari	Oji-Cree	Tajiki
Avar	English	Yiddish	Limbu	Ojibway	Tamil
Awadhi	Erzya	Japanese	Lomwe	Oriya	Tatar
Aymara	Spanish	Judezmo	Lower Sorbian	Oromo	TH-Cree
Azeri	Estonian	Jula	Lule Sami	Ossetian	Telugu
Badaga	Basque	Kabardian	Lithuanian	Palestinian Aramaic	Tongan
Baghelkhandi	Evenki	Kachchi	Luba	Pali	Tigre
Balkar	Even	Kalenjin	Luganda	Punjabi	Tigrinya
Baule	Ewe	Kannada	Luhya	Palpa	Thai
Berber	French Antillean	Karachay	Luo	Pashto	Tahitian
Bench	¶Farsi	Georgian	Latvian	Polytonic Greek	Tibetan
Bible Cree	¶Parsi	Kazakh	Majang	Pilipino	Turkmen
Belarussian	¶Persian	Kebena	Makua	Palaung	Temne
Bemba	Finnish	Khutsuri Georgian	Malayalam	Polish	Tswana
Bengali	Fijian	Khakass	Traditional	Provencal	Tundra Nenets
Bulgarian	Flemish	Khanty-Kazim	Mansi	Portuguese	Tonga
Bhili	Forest Nenets	Khmer	Marathi	Chin	Todo
Bhojpuri	Fon	Khanty-Shurishkar	Marwari	Rajasthani	Turkish
Bikol	Faroese	Khanty-Vakhi	Mbundu	R-Cree	Tsonga
Bilen	French	Khowar	Manchu	Russian Buriat	Turoyo Aramaic
Blackfoot	Frisian	Kikuyu	Moose Cree	Riang	Tulu
Balochi	Friulian	Kirghiz	Mende	Rhaeto-Romanic	Tuvin
Balante	Futa	Kisii	Me'en	Romanian	Twi
Balti	Fulani	Kokni	Mizo	Romany	Udmurt
Bambara	Ga	Kalmyk	Macedonian	Rusyn	Ukrainian
Bamileke	Gaelic	Kamba	Male	Ruanda	Urdu
Breton	Gagauz	Kumaoni	Malagasy	Russian	Upper Sorbian
Brahui	Galician	Komo	Malinke	Sadri	Uyghur
Braj Bhasha	Garshuni	Komso	Malayalam	Sanskrit	Uzbek
Burmese	Garhwali	Kanuri	Reformed	Santali	Venda
Bashkir	Ge'ez	Kodagu	Malay	Sayisi	Vietnamese
Beti	Gilyak	Korean Old Hangul	Mandinka	Sekota	Wa
Catalan	Gumuz	Konkani	Mongolian	Selkup	Wagdi
Cebuano	Gondi	Kikongo	Manipuri	Sango	West-Cree
Chechen	Greenlandic	Komi-Permyak	Maninka	Shan	Welsh
Chaha Gurage	Garo	Korean	Manx Gaelic	Sibe	Wolof
Chattisgarhi	Guarani	Komi-Zyrian	Moksha	Sidamo	Tai Lue
Chichewa	Gujarati	Kpelle	Moldavian	Silte Gurage	Xhosa
Chukchi	Haitian	Krio	Mon	Skolt Sami	Yakut
Chipewyan	Halam	Karakalpak	Moroccan	Slovak	Yoruba
Cherokee	Harauti	Karelian	Maori	Slavey	Y-Cree
Chuvash	Hausa	Karaim	Maithili	Slovenian	Yi Classic
Comorian	Hawaiin	Karen	Maltese	Somali	Yi Modern
Coptic	Hammer-Banna	Koorete	Mundari	Samoan	Chinese Hong Kong
Cree	Hiligaynon	Kashmiri	Naga-Assamese	Sena	Chinese Phonetic
Carrier	Hindi	Khasi	Nanai	Sindhi	Chinese Simplified
Crimean Tatar	High Mari	Kildin Sami	Naskapi	Sinhalese	Chinese Traditional
Church Slavonic	Hindko	Kui	N-Cree	Soninke	Zande
Czech	Ho	Kulvi	Ndebele	Sodo Gurage	Zulu
Danish	Harari	Kumyk	Ndonga	Sotho	
Dargwa	Croatian	Kurdish	Nepali	Albanian	
Woods Cree	Hungarian	Kurukh	Newari	Serbian	
German	Armenian	Kuy	Nagari	Saraiki	

3 OpenType font features
OpenType 字体特征

There are a finite set of OpenType font features, and fontspec provides an interface to around half of them. Full documentation will be presented in the following sections, including how to enable and disable individual features, and how they interact.

有限的一组 OpenType 字体特征，fontspec 提供了其中约一半的接口。下面的章节将提供完整的文档，包括如何启用和禁用单个特征以及它们之间的交互。

A brief reference is provided (Table 4 on the following page) but note that this is an incomplete listing — only the 'enable' keys are shown, and where alternative interfaces are provided for convenience only the first is shown. (E.g., `Numbers=OldStyle` is the same as `Numbers=Lowercase`.)

提供了简要参考信息（Table 4 on the next page），但请注意这是不完整的列表——仅显示了“启用”键，并且在提供方便的替代界面的情况下，仅显示了第一个界面。（例如，`Numbers=OldStyle` 与 `Numbers=Lowercase` 相同。）

For completeness, the complete list of OpenType features *not* provided with a fontspec interface is shown in Table 5 on page 52. Features omitted are partially by design and partially by oversight; for example, the `aalt` feature is largely useless in TEX since it is designed for providing a GUI interface for selecting 'all alternates' of a glyph. Others, such as optical bounds for example, simply haven't yet been considered due to a lack of fonts available for testing. Suggestions welcome for how/where to add these missing features to the package.

为了完整起见，未提供 fontspec 接口的完整的 OpenType 特征列表在 Table 5 on page 52 中显示。省略的功能部分是出于设计，部分是由于疏忽；例如，`aalt` 特征在 TEX 中大多无用，因为它旨在为选择字形的“所有替代”提供 GUI 接口。其他例如光学边界的特征之所以未考虑是由于缺乏可用于测试的字体。欢迎提供建议，以便将这些缺失的特征添加到该软件包中的何处。

3.1 Tag-based features
基于标记的特征

3.1.1 Alternates — `salt`
替代形式 — `salt`

The `Alternate` feature, alias `StylisticAlternates`, is used to access alternate font glyphs when variations exist in the font, such as in Example 20. It uses a numerical selection, starting from zero, that will be different for each font. Note that the `Style=Alternate` option is equivalent to `Alternate=0` to access the default case.

当字体存在多种变体时（如在例 Example 20 中），`Alternate` 特征（别名 `StylisticAlternates`）用于访问替代字形。它使用数字选择，从零开始，并对每种字体都不同。请注意，选项 `Style=Alternate` 等效于 `Alternate=0`，用于访问默认情况。

Note that the indexing starts from zero. With the LuaTEX engine, `Alternate=Random` selects a random alternate.

Table 4: Summary of OpenType features in fontspec, alphabetic by feature tag.
按功能标签字母顺序列出的 fontspec 中 OpenType 功能的摘要。

abvm	Diacritics = AboveBase	*Above-base Mark Positioning*
afrc	Fractions = Alternate	*Alternative Fractions*
blwm	Diacritics = BelowBase	*Below-base Mark Positioning*
calt	Contextuals = Alternate	*Contextual Alternates*
case	Letters = Uppercase	*Case-Sensitive Forms*
clig	Ligatures = Contextual	*Contextual Ligatures*
cpsp	Kerning = Uppercase	*Capital Spacing*
cswh	Contextuals = Swash	*Contextual Swash*
cvNN	CharacterVariant = $N:M$	*Character Variant N*
c2pc	Letters = UppercasePetiteCaps	*Petite Capitals From Capitals*
c2sc	Letters = UppercaseSmallCaps	*Small Capitals From Capitals*
dlig	Ligatures = Rare	*Discretionary Ligatures*
dnom	VerticalPosition = Denominator	*Denominators*
expt	CJKShape = Expert	*Expert Forms*
falt	Contextuals = LineFinal	*Final Glyph on Line Alternates*
fina	Contextuals = WordFinal	*Terminal Forms*
frac	Fractions = On	*Fractions*
fwid	CharacterWidth = Full	*Full Widths*
halt	CharacterWidth = AlternateHalf	*Alternate Half Widths*
hist	Style = Historic	*Historical Forms*
hkna	Style = HorizontalKana	*Horizontal Kana Alternates*
hlig	Ligatures = Historic	*Historical Ligatures*
hwid	CharacterWidth = Half	*Half Widths*
init	Contextuals = WordInitial	*Initial Forms*
ital	Style = Italic	*Italics*
jp78	CJKShape = JIS1978	*JIS78 Forms*
jp83	CJKShape = JIS1983	*JIS83 Forms*
jp90	CJKShape = JIS1990	*JIS90 Forms*
jp04	CJKShape = JIS2004	*JIS2004 Forms*
kern	Kerning = On	*Kerning*
liga	Ligatures = Common	*Standard Ligatures*
lnum	Numbers = Uppercase	*Lining Figures*
locl	LocalForms = On	*Localized Forms*
mark	Diacritics = MarkToBase	*Mark Positioning*
medi	Contextuals = Inner	*Medial Forms*
mkmk	Diacritics = MarkToMark	*Mark to Mark Positioning*
nalt	Annotation = N	*Alternate Annotation Forms*
nlck	CJKShape = NLC	*NLC Kanji Forms*
numr	VerticalPosition = Numerator	*Numerators*
onum	Numbers = Lowercase	*Oldstyle Figures*
ordn	VerticalPosition = Ordinal	*Ordinals*
ornm	Ornament = N	*Ornaments*
palt	CharacterWidth = AlternateProportional	*Proportional Alternate Widths*
pcap	Letters = PetiteCaps	*Petite Capitals*
pkna	Style = ProportionalKana	*Proportional Kana*
pnum	Numbers = Proportional	*Proportional Figures*
pwid	CharacterWidth = Proportional	*Proportional Widths*
qwid	CharacterWidth = Quarter	*Quarter Widths*
rand	Letters = Random	*Randomize*
rlig	Ligatures = Required	*Required Ligatures*
ruby	Style = Ruby	*Ruby Notation Forms*
salt	Alternate = N	*Stylistic Alternates*
sinf	VerticalPosition = ScientificInferior	*Scientific Inferiors*
smcp	Letters = SmallCaps	*Small Capitals*
smpl	CJKShape = Simplified	*Simplified Forms*
ssNN	StylisticSet = N	*Stylistic Set N*
ssty	Style = MathScript	*Math script style alternates*
subs	VerticalPosition = Inferior	*Subscript*
sups	VerticalPosition = Superior	*Superscript*
swsh	Style = Swash	*Swash*
titl	Style = Titling	*Titling*
tnum	Numbers = Monospaced	*Tabular Figures*
trad	CJKShape = Traditional	*Traditional Forms*
twid	CharacterWidth = Third	*Third Widths*
unic	Letters = Unicase	*Unicase*
valt	Vertical = AlternateMetrics	*Alternate Vertical Metrics*
vert	Vertical = Alternates	*Vertical Writing*
vhal	Vertical = HalfMetrics	*Alternate Vertical Half Metrics*
vkna	Style = VerticalKana	*Vertical Kana Alternates*
vkrn	Vertical = Kerning	*Vertical Kerning*
vpal	Vertical = ProportionalMetrics	*Proportional Alternate Vertical Metrics*
vrt2	Vertical = RotatedGlyphs	*Vertical Alternates and Rotation*
vrtr	Vertical = AlternatesForRotation	*Vertical Alternates for Rotation*
zero	Numbers = SlashedZero	*Slashed Zero*

Table 5: List of *unsupported* OpenType features.

AALT	*Access All Alternates*	HNGL	*Hangul*	RCLT	*Required Contextual Alternates*
ABVF	*Above-base Forms*	HOJO	*Hojo Kanji Forms*	RKRF	*Rakar Forms*
ABVS	*Above-base Substitutions*	ISOL	*Isolated Forms*	RPHF	*Reph Forms*
AKHN	*Akhands*	JALT	*Justification Alternates*	RTBD	*Right Bounds*
BLWF	*Below-base Forms*	LFBD	*Left Bounds*	RTLA	*Right-to-left alternates*
BLWS	*Below-base Substitutions*	LJMO	*Leading Jamo Forms*	RTLM	*Right-to-left mirrored forms*
CCMP	*Glyph Composition / Decomposition*	LTRA	*Left-to-right alternates*	RVRN	*Required Variation Alternates*
CFAR	*Conjunct Form After Ro*	LTRM	*Left-to-right mirrored forms*	SIZE	*Optical size*
CJCT	*Conjunct Forms*	MED2	*Medial Forms #2*	STCH	*Stretching Glyph Decomposition*
CPCT	*Centered CJK Punctuation*	MGRK	*Mathematical Greek*	TJMO	*Trailing Jamo Forms*
CURS	*Cursive Positioning*	MSET	*Mark Positioning via Substitution*	TNAM	*Traditional Name Forms*
DIST	*Distances*	NUKT	*Nukta Forms*	VATU	*Vattu Variants*
DTLS	*Dotless Forms*	OPBD	*Optical Bounds*	VJMO	*Vowel Jamo Forms*
FIN2	*Terminal Forms #2*	PREF	*Pre-Base Forms*		
FIN3	*Terminal Forms #3*	PRES	*Pre-base Substitutions*		
FLAC	*Flattened accent forms*	PSTF	*Post-base Forms*		
HALF	*Half Forms*	PSTS	*Post-base Substitutions*		
HALN	*Halant Forms*				

Example 20: The `Alternate` feature.
`Alternate` 特征。

—Example graphic not generated—

```
\fontspec{LinLibertine_R.otf}
\textsc{a} \& h \\
\addfontfeature{Alternate=0}
\textsc{a} \& h
```

请注意，索引从零开始。在 LuaTEX 引擎中，`Alternate=Random` 会选择随机替代。

See Section 1 on page 85 for a way to assign names to alternates if desired.

如需按名称为替代字形分配名称，请参阅 Section 1 on page 85。

3.1.2 Character Variants — `cvNN`
字符变体 — `cvNN`

'Character Variations' are selected numerically to adjust the output of (usually) a single character for the particular font. These correspond to the OpenType features `cv01` to `cv99`.

“字符变体”用数字选择来调整特定字体中（通常）单个字符的输出。这些对应于 OpenType 特征 `cv01` 到 `cv99`。

For each character that can be varied, it is possible to select among possible options for that particular glyph. For example, in the hypothetical example below, variants are chosen for glyphs '4' and '5', and the trailing :⟨*n*⟩ corresponds to which variety to choose.

对于每个可以变化的字符，可以在该特定字形的可能选项中进行选择。例如，在以下假设的示例中，为字形“4”和“5”选择了变体，并且后面的 :⟨*n*⟩ 对应于要选择的种类。

```
\fontspec{CV Font}[CharacterVariant={4,5:2}] \& violet
```

The numbering is entirely font-specific. Glyph '5' might be the character 'v', for example. Character variants are specifically designed not to conflict with each other, so you can enable them individually per character. (Unlike stylistic alternates, say.) Note that the indexing starts from zero.

编号完全取决于字体。例如，字形“5”可能是字符“v”。字符变体专门设计为彼此不冲突，因此您可以针对每个字符单独启用它们。请注意，索引从零开始。

3.1.3 Contextuals
上下文相关替换

This feature refers to substitutions of glyphs that vary 'contextually' by their relative position in a word or string of characters; features such as contextual swashes are accessed via the options shown in Table 6.

该特征指的是字形在单词或一串字符中的相对位置而变化的“上下文”替换；可以通过 Table 6 中显示的选项访问上下文替换，如上下文花边等。

Historic forms are accessed in OpenType fonts via the feature `Style=Historic`; this is generally *not* contextual in OpenType, which is why it is not included in this feature.

历史形式可以通过 OpenType 字体中的 `Style=Historic` 特征访问；这通常在 OpenType 中不是上下文相关的，因此未包括在此特征中。

3.1.4 Diacritics
变音符号

Specifies how combining diacritics should be placed. These will usually be controlled automatically according to the Script setting.

Table 6: Options for the OpenType font feature 'Contextuals'.

Feature	Option	Tag
Contextuals =	Swash	`cswh` †
	Alternate	`calt` †
	WordInitial	`init` †
	WordFinal	`fina` †
	LineFinal	`falt` †
	Inner	`medi` †
	ResetAll	

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

指定如何放置组合变音符号。通常会根据脚本设置自动控制。

3.1.5 Fractions — `frac`
分数 — `frac`

Activates the construction of 'vulgar' fractions using precomposed glyphs and/or subscript and superscript characters from within the font. Coverage will vary by font; see Example 21. Some (Asian fonts predominantly) also provide for the `Alternate` option.

激活使用预先组成的字形和/或字体内的上标和下标字符构造“真分数”的功能。不同字体的覆盖范围会有所不同；请参见 Example 21。有些字体（主要是亚洲字体）还提供了 `Alternate` 选项。

3.1.6 Kerning — `kern`
字距 — `kern`

Specifies how inter-glyph spacing should behave. Well-made fonts include information for how differing amounts of space should be inserted between separate character pairs. This kerning space is inserted automatically but in rare circumstances you may wish to turn it off.

指定字形之间的间距应该如何表现。制作精良的字体包括不同的字形对之间应该插入多

Table 7: Options for the OpenType font feature 'Diacritics'.

Feature	Option	Tag
Diacritics =	MarkToBase	`mark` †
	MarkToMark	`mkmk` †
	AboveBase	`abvm` †
	BelowBase	`blwm` †
	ResetAll	

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Table 8: Options for the OpenType font feature 'Fractions'.

Feature		Option	Tag
Fractions	=	On	`+frac`
		Off	`-frac`
		Reset	
		Alternate	`afrc` †
		ResetAll	

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Example 21: The `Fractions` feature.

—Example graphic not generated—

```
\setsansfont{Lato}[Fractions=On]
\setmonofont{IBM Plex Mono}[Fractions=On]

\sffamily 1/2 47/11 1/1000 \par
\ttfamily 1/2 47/11
```

少空格的信息。这种字距空间会自动插入，但在极少数情况下，您可能希望关闭它。

As briefly mentioned previously at the end of 3.1.7 on page 56, the `Uppercase` option will add a small amount of tracking between uppercase letters, seen in Example 22, which uses the Romande fonts[11] (thanks to Clea F. Rees for the suggestion). The `Uppercase` option acts separately to the regular kerning controlled by the `On/Off` options.

如3.1.7节末尾简要提到的，`Uppercase` 选项会在大写字母之间添加一小段跟踪，如 Example 22所示，使用了 Romande 字体（感谢 Clea F. Rees 提供建议）。`Uppercase` 选项与由 `On/Off` 选项控制的常规字距相分离。

[11] http://arkandis.tuxfamily.org/adffonts.html

Table 9: Options for the OpenType font feature 'Kerning'.

Feature		Option	Tag
Kerning	=	On	`+kern`
		Off	`-kern`
		Reset	
		Uppercase	`cpsp` †
		ResetAll	

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Example 22: Adding extra kerning for uppercase letters. (The difference is usually very small.)
为大写字母添加额外的字距（差异通常非常小）。

—Example graphic not generated—

```
\fontspec{RomandeADFStd-DemiBold.otf}
 UPPERCASE EXAMPLE \\
\addfontfeature{Kerning=Uppercase}
 UPPERCASE EXAMPLE
```

3.1.7 Letters
字母

The `Letters` feature specifies how the letters in the current font will look. OpenType fonts may contain the following options: `SmallCaps`, `PetiteCaps`, `UppercaseSmallCaps`, `UppercasePetiteCaps`, and `Unicase`.

`Letters` 特性规定了当前字体中字母的外观。OpenType 字体可以包含以下选项：`SmallCaps`、`PetiteCaps`、`UppercaseSmallCaps`、`UppercasePetiteCaps` 和 `Unicase`。下面的代码展示了这些选项，你可以在你的文档中使用它们：

Petite caps are smaller than small caps. `SmallCaps` and `PetiteCaps` turn lowercase letters into the smaller caps letters, whereas the `Uppercase...` options turn the *capital* letters into the smaller caps (good, *e.g.*, for applying to already uppercase acronyms like 'NASA'). This difference is shown in Example 23. 'Unicase' is a weird hybrid of upper and lower case letters.

小小型大写字母比小型大写字母更小。选项 `SmallCaps` 和 `PetiteCaps` 将小写字母转换为更小的大写字母，而选项 `Uppercase...` 将大写字母转换为更小的大写字母（例如，适用于已经大写的缩写词，如“NASA”）。这种差异在 Example 23 中展示。选项 `Unicase` 是大写和小写字母的奇怪混合体。

3.1.8 Ligatures
连字

`Ligatures` refer to the replacement of two separate characters with a specially drawn glyph for functional or æsthetic reasons. The list of options, of which multiple may be

Table 10: Options for the OpenType font feature 'Letters'.

Feature	Option	Tag
Letters =	SmallCaps	`smcp` †
	PetiteCaps	`pcap` †
	UppercaseSmallCaps	`c2sc` †
	UppercasePetiteCaps	`c2pc` †
	Unicase	`unic` †
	ResetAll	

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Example 23: Small caps from lowercase or uppercase letters.
小型大写字母和小型小写字母

```
\fontspec{texgyreadventor-regular.otf}[Letters=SmallCaps]
 THIS SENTENCE no verb                 \\
\fontspec{texgyreadventor-regular.otf}[Letters=UppercaseSmallCaps]
 THIS SENTENCE no verb
```

—*Example graphic not generated*—

selected at one time, is shown in Table 11. A demonstration with the Linux Libertine fonts[12] is shown in Example 24.

`Ligatures`（连字）指的是为了实现某些功能或美观效果而将两个分开的字符替换为特殊绘制的字形。可以同时选择多个选项，选项列表如 Table 11 所示。使用 Linux Libertine 字体的演示见 Example 24。

Note the additional features accessed with `Ligatures=TeX`. These are not actually real OpenType features, but additions provided by luaotfload (i.e., LuaTEX only) to emulate TEX's behaviour for ASCII input of curly quotes and punctuation. In XETEX this is achieved with the `Mapping` feature (see Section 1.1 on page 77) but for consistency `Ligatures=TeX` will perform the same function as `Mapping=tex-text`.

请注意，可以通过 `Ligatures=TeX` 访问其他特性。这些实际上不是真正的 OpenType 特性，而是由 luaotfload 提供的补充功能（仅适用于 LuaTEX），以模拟 ASCII 输入的花式引号和标点符号在 TEX 中的行为。在 XETEX 中，可以使用 `Mapping` 特性（见 Section 1.1 on page 77）来实现此功能，但为了保持一致性，`Ligatures=TeX` 将执行与 `Mapping=tex-text` 相同的功能。

3.1.9 Localised Forms — `locl`
本地化表单 — `locl`

This feature enables and disables glyph substitutions, etc., that are specific to the `Language` selected in the font. This feature is automatically activated by default when

[12] http://www.linuxlibertine.org/

Table 11: Options for the OpenType font feature 'Ligatures'.

Feature	Option	Tag
Ligatures =	Required	`rlig` †
	Common	`liga` †
	Contextual	`clig` †
	Rare/Discretionary	`dlig` †
	Historic	`hlig` †
	TeX	`tlig` †
	ResetAll	

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Example 24: An example of the Ligatures feature.
连字 特性的例子。

—Example graphic not generated—

```
\def\test#1#2{%
  #2 $\to$ {\addfontfeature{#1} #2}\\}
\fontspec{LinLibertine_R.otf}
\test{Ligatures=Historic}{strict}
\test{Ligatures=Rare}{wurtzite}
\test{Ligatures=CommonOff}{firefly}
```

Table 12: Options for the OpenType font feature 'LocalForms'.

Feature	Option	Tag
LocalForms =	On	+locl
	Off	-locl
	Reset	

† These feature options can be disabled with ..Off variants, and reset to default state (neither explicitly on nor off) with ..Reset.

present, so it should not be generally necessary to use LocalForms = On. In certain scenarios it may be important to turn it Off (although nothing specifically springs to mind).

此特性启用和禁用特定于字体中所选 语言 的字形替换等。该特性在存在时默认自动激活，因此通常不需要使用 本地化表单 = 打开。在某些情况下，关闭它可能很重要（尽管没有什么具体想法）。

3.1.10 Numbers
数字

The Numbers feature defines how numbers will look in the selected font, accepting options shown in Table 13.

数字 特性定义所选字体中数字的外观，接受 Table 13 中显示的选项。

The synonyms Uppercase and Lowercase are equivalent to Lining and OldStyle, respectively. The differences have been shown previously in Section 2 on page 28. The Monospaced option is useful for tabular material when digits need to be vertically aligned.

同义词 大写 和 小写 分别等效于 标准 和 古老。差异在 Section 2 on page 28 中已经显示过了。等宽 选项在数字需要垂直对齐的表格材料中很有用。

The SlashedZero option replaces the default zero with a slashed version to prevent confusion with an uppercase 'O', shown in Example 25.

零带斜线 选项将默认零替换为带斜线的版本，以防与大写字母 'O' 混淆，如 Example 25 所示。

The Arabic option (with tag anum) maps regular numerals to their Arabic script or Persian equivalents based on the current Language setting (see Section 2 on page 46). This option is based on a LuaTEX feature of the luaotfload package, not an OpenType feature.

Table 13: Options for the OpenType font feature 'Numbers'.

Feature	Option	Tag
Numbers =	Uppercase	`lnum` †
	Lowercase	`onum` †
	Lining	`lnum` †
	OldStyle	`onum` †
	Proportional	`pnum` †
	Monospaced	`tnum` †
	SlashedZero	`zero` †
	Arabic	`anum` †
	ResetAll	

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Example 25: The effect of the `SlashedZero` option.
零带斜线 选项的效果。

```
\fontspec[Numbers=Lining]{texgyrebonum-regular.otf}
0123456789
\fontspec[Numbers=SlashedZero]{texgyrebonum-regular.otf}
0123456789
```

—Example graphic not generated—

(Thus, this feature is unavailable in XeTeX.) This feature should be considered deprecated; while there are no plans to remove it from this package, if its support is dropped from the font loader it could disappear from fontspec with little notice.

`Arabic` 选项（标签为`anum`）基于当前的 `Language` 设置（见Section 2 on page 46），将常规数字映射到阿拉伯文或波斯文数字的等效符号。此选项基于 luaotfload 软件包的LuaTeX 功能，而不是 OpenType 功能。（因此，在 XeTeX 中无法使用此功能。）此功能应被视为已弃用；虽然没有计划从该软件包中删除它，但如果从字体加载程序中删除它的支持，则可能很快从 fontspec 中消失。

3.1.11 Ornament — `ornm`
装饰性 — `ornm`

Ornaments are selected with the `Ornament` feature (OpenType feature `ornm`), selected numerically such as for the `Annotation` feature.

使用 `Ornament` 特性（OpenType 特性`ornm`）来选择装饰，可以使用数字选择，例如 `Annotation` 特性。

3.1.12 Style
风格

'Ruby' refers to a small optical size, used in Japanese typography for annotations. For fonts with multiple `salt` OpenType features, use the fontspec `Alternate` feature instead.

Table 14: Options for the OpenType font feature 'Style'.

Feature	Option	Tag
Style =	Alternate	salt †
	Cursive	curs †
	Historic	hist †
	Italic	ital †
	Ruby	ruby †
	Swash	swsh †
	Titling	titl †
	Uppercase	case †
	HorizontalKana	hkna †
	VerticalKana	vkna †
	ResetAll	

† These feature options can be disabled with ..Off variants, and reset to default state (neither explicitly on nor off) with ..Reset.

'Ruby' 指的是日本排版中用于注释的小字体。对于带有多个salt OpenType 特性的字体，使用 fontspec 的 Alternate 特性代替。

Example 26 shows an example of a font feature that involves glyph substitution for particular letters within an alphabet. Other options in these categories operate in similar ways, with the choice of how particular substitutions are organised with which feature largely up to the font designer.

示例**??**展示了涉及字母表中特定字母的字形替换的字体特性的示例。这些类别中的其他选项以类似的方式运作，特定替换的选择大多由字体设计师决定。

The Uppercase option is designed to select various uppercase forms for glyphs such as accents and dashes, such as shown in Example 27; note the raised position of the hyphen to better match the surrounding letters. It will (probably) not actually map letters to uppercase.[13] This option used to be selected under the Letters feature, but moved here as it generally does not actually affect the letters themselves. The Kerning feature also contains an Uppercase option, which adds a small amount of spacing in between letters (see 3.1.6 on page 54).

Uppercase 选项旨在选择大写字母的各种形式，例如用于重音符号和破折号的字形，如示例**??**所示；请注意连字符的上升位置，以更好地匹配周围的字母。它（可能）不会实

[13] If you want automatic uppercase letters, look to LaTeX's \MakeUppercase command.

Example 26: Example of the Alternate option of the Style feature.
Style 特性的 Alternate 选项的示例

—Example graphic not generated—

```
\fontspec{Quattrocento-Regular.otf}
 M Q W                          \\
\addfontfeature{Style=Alternate}
 M Q W
```

际映射字母到大写。如果你想要自动大写字母，请使用 LaTeX 的 \MakeUppercase命令。该选项曾在 Letters 特性下选择，但移至此处，因为它通常实际上并不影响字母本身。Kerning 特性还包含一个 Uppercase 选项，它在字母之间添加了一小段间距（参见3.1.6 on page 54 ）。

In other features, larger breadths of changes can be seen, covering the style of an entire alphabet. See Example 28; here, the Italic option affects the Latin text and the Ruby option the Japanese.

在其他功能中，可以看到更大的变化幅度，涵盖整个字母表的风格。参见 Example 28；在这里，Italic 选项影响拉丁文本，Ruby 选项影响日语。

Note the difference here between the default and the horizontal style kana in Example 29: the horizontal style is slightly wider.

请注意，在 Example 29 中默认样式和水平样式假名之间的差异：水平样式略微宽。

3.1.13 Stylistic Set variations — ssNN
风格集变体 — ssNN

This feature selects a 'Stylistic Set' variation, which usually corresponds to an alternate glyph style for a range of characters (usually an alphabet or subset thereof). This feature is specified numerically. These correspond to OpenType features ss01, ss02, etc.

该功能选择“风格集”变体，通常对应于一系列字符的备用字形样式（通常是一个字母表或其子集）。该功能以数字形式指定。这些对应于 OpenType 功能 ss01、ss02 等。

Two demonstrations from the Junicode font[14] are shown in Example 30 and Example 31; thanks to Adam Buchbinder for the suggestion.

在 Junicode 字体[15] 中展示了两个示例，感谢 Adam Buchbinder 的建议，分别在 Example 30 和 Example 31 中。

Multiple stylistic sets may be selected simultaneously by writing, e.g., StylisticSet={1,2,3}.

多个风格集可以同时选择，例如，写成 StylisticSet={1,2,3}。

The StylisticSet feature is a synonym of the Variant feature for AAT fonts. See Section 1 on page 85 for a way to assign names to stylistic sets, which should be done on a per-font basis.

[14] http://junicode.sf.net
[15] http://junicode.sf.net

Example 27: An example of the Uppercase option of the Style feature.
Style 特性的 Uppercase 选项的示例

—Example graphic not generated—

```
\fontspec{LinLibertine_R.otf}
 UPPER-CASE example \\
\addfontfeature{Style=Uppercase}
 UPPER-CASE example
```

Example 28: Example of the `Italic` and `Ruby` options of the `Style` feature.

—Example graphic not generated—

```
\fontspec{Hiragino Mincho Pro}
 Latin \kana        \\
\addfontfeature{Style={Italic, Ruby}}
 Latin \kana
```

Example 29: Example of the `HorizontalKana` and `VerticalKana` options of the `Style` feature.
`Style` 功能的 `HorizontalKana` 和 `VerticalKana` 选项的示例。

—Example graphic not generated—

```
\fontspec{Hiragino Mincho Pro}
  \kana   \\
{\addfontfeature{Style=HorizontalKana}
  \kana } \\
{\addfontfeature{Style=VerticalKana}
  \kana }
```

Example 30: Insular letterforms, as used in medieval Northern Europe, for the Junicode font accessed with the `StylisticSet` feature.
Junicode 字体（使用 `StylisticSet` 功能）的 Insular 字形，用于中世纪北欧。

—Example graphic not generated—

```
\fontspec{Junicode}
 Insular forms. \\
\addfontfeature{StylisticSet=2}
 Insular forms. \\
```

Example 31: Enlarged minuscules (capital letters remain unchanged) for the Junicode font, accessed with the `StylisticSet` feature.
Junicode 字体的放大小写字母（大写字母保持不变），使用 `StylisticSet` 功能访问。

—Example graphic not generated—

```
\fontspec{Junicode}
 ENLARGED Minuscules. \\
\addfontfeature{StylisticSet=6}
 ENLARGED Minuscules. \\
```

StylisticSet 特性是 AAT 字体中 Variant 特性的同义词。关于如何为字体样式集分配名称，请参见 Section 1 on page 85，这应该在每个字体的基础上完成。

3.1.14 Vertical Position
垂直位置

The VerticalPosition feature is used to access things like subscript (Inferior) and superscript (Superior) numbers and letters (and a small amount of punctuation, sometimes). The Ordinal option will only raise characters that are used in some languages directly after a number. The ScientificInferior feature will move glyphs further below the baseline than the Inferior feature. These are shown in Example 32

VerticalPosition 特性用于访问下标 (Inferior) 和上标 (Superior) 数字、字母 (以及一些标点符号)，Ordinal 选项仅会提升一些语言中直接在数字后面使用的字符。ScientificInferior 特性会使字形相比 Inferior 特性进一步下降基线。这些内容显示在 Example 32 中。

Numerator and Denominator should only be used for creating arbitrary fractions (see next section).

Numerator 和 Denominator 应仅用于创建任意分数（见下一节）。

The realscripts package (which is also loaded by xltxtra for XeTeX) redefines the \textsubscript and \textsuperscript commands to use the above font features automatically, including for use in footnote labels. If this is the only feature of xltxtra you wish to use, consider loading realscripts on its own instead.

realscripts 宏包（也被 XeTeX 中的 xltxtra 加载）会自动重定义 \textsubscript 和 \textsuperscript 命令，使用以上的字体特性，包括用于脚注标签。如果这是您想要使用 xltxtra 的唯一功能，请考虑单独加载 realscripts。

Table 15: Options for the OpenType font feature 'VerticalPosition'.

Feature	Option	Tag
VerticalPosition =	Superior	sups †
	Inferior	subs †
	Numerator	numr †
	Denominator	dnom †
	ScientificInferior	sinf †
	Ordinal	ordn †
	ResetAll	

† These feature options can be disabled with ..Off variants, and reset to default state (neither explicitly on nor off) with ..Reset.

Example 32: The VerticalPosition feature.
VerticalPosition 特性。

```
\fontspec{LibreCaslonText-Regular.otf}[VerticalPosition=Superior]
 Superior: 123456789Q                                          \\
\fontspec{LibreCaslonText-Regular.otf}[VerticalPosition=Numerator]
 Numerator: 12345                                              \\
\fontspec{LibreCaslonText-Regular.otf}[VerticalPosition=Denominator]
 Denominator: 12345                                            \\
\fontspec{LibreCaslonText-Regular.otf}[VerticalPosition=ScientificInferior]
 Scientific Inferior: 12345
```

—Example graphic not generated—

3.2 CJK features
CJK 特性

This section summarises the features which are largely intending for Chinese, Korean, and Japanese typesetting.

本节总结了主要面向中文、韩文和日文排版的特性。

3.2.1 Annotation — nalt
注释 — nalt

Some fonts are equipped with an extensive range of numbers and numerals in different forms. These are accessed with the Annotation feature (OpenType feature nalt), selected numerically as shown in Example 33. Note that the indexing starts from zero.

一些字体配备了不同形式的广泛数字和数字字符。可以使用 Annotation 特性（OpenType 特性 nalt）来访问它们，如 Example 33 中所示进行数字选择。请注意，索引从零开始。

3.2.2 Character width
字符宽度

Many Asian fonts are equipped with variously spaced characters for shoe-horning into their generally monospaced text. These are accessed through the CharacterWidth feature.

许多亚洲字体配备了各种间隔不同的字符以适应它们通常的等宽文本排版。这些特征可以通过 CharacterWidth 功能来实现。

Example 33: Annotation forms for OpenType fonts.
OpenType 字体的注释形式。

```
\fontspec{Hiragino Maru Gothic Pro}
 1 2 3 4 5 6 7 8 9
\def\x#1{\\{\addfontfeature{Annotation=#1}
          1 2 3 4 5 6 7 8 9 }}
\x0\x1\x2\x3\x4\x5\x6\x7\x7\x8\x9
```

—Example graphic not generated—

Table 16: Options for the OpenType font feature 'CharacterWidth'.

Feature	Option	Tag
CharacterWidth =	Proportional	`pwid` †
	Full	`fwid` †
	Half	`hwid` †
	Third	`twid` †
	Quarter	`qwid` †
	AlternateProportional	`palt` †
	AlternateHalf	`halt` †
	ResetAll	

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Japanese alphabetic glyphs (in Hiragana or Katakana) may be typeset proportionally, to better fit horizontal measures, or monospaced, to fit into the rigid grid imposed by ideographic typesetting. In this latter case, there are also half-width forms for squeezing more kana glyphs (which are less complex than the kanji they are amongst) into a given block of space. The same features are given to roman letters in Japanese fonts, for typesetting foreign words in the same style as the surrounding text.

日文假名（平假名或片假名）可以按比例排版以更好地适应横向度量，也可以等宽排版以适应象形字排版所强加的刚性网格。在后一种情况下，还有半角形式，以将更多假名字母（它们比它们之间的汉字更简单）挤入给定的空间块中。日文字体中的罗马字母也具有相同的特征，可用于以与周围文本相同的风格排版外来词汇。

The same situation occurs with numbers, which are provided in increasingly illegible compressed forms seen in Example 35.

同样的情况也出现在数字中，这些数字以越来越难以辨认的压缩形式呈现，如 Example 35 所示。

Example 34: Proportional or fixed width forms.
比例或固定宽度形式。

```
\def\test{\makebox[2cm][l]{\texta}%
          \makebox[2.5cm][l]{\textb}%
          \makebox[2.5cm][l]{abcdef}}
\fontspec{Hiragino Mincho Pro}
{\addfontfeature{CharacterWidth=Proportional}\test}\\
{\addfontfeature{CharacterWidth=Full}\test}\\
{\addfontfeature{CharacterWidth=Half}\test}
```

—Example graphic not generated—

Example 35: Numbers can be compressed significantly.
数字可以显著压缩。

```
\fontspec[Renderer=AAT]{Hiragino Mincho Pro}
{\addfontfeature{CharacterWidth=Full}
 ---12321---}\\
{\addfontfeature{CharacterWidth=Half}
 ---1234554321---}\\
{\addfontfeature{CharacterWidth=Third}
 ---123456787654321---}\\
{\addfontfeature{CharacterWidth=Quarter}
 ---12345678900987654321---}
```

—Example graphic not generated—

Table 17: Options for the OpenType font feature 'CJKShape'.

Feature	Option	Tag
CJKShape =	Traditional	`trad`
	Simplified	`smpl`
	JIS1978	`jp78`
	JIS1983	`jp83`
	JIS1990	`jp90`
	Expert	`expt`
	NLC	`nlck`

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

3.2.3 CJK shape
CJK 形状

There have been many standards for how CJK ideographic glyphs are 'supposed' to look. Some fonts will contain many alternate glyphs available in order to be able to display these gylphs correctly in whichever form is appropriate. Both AAT and OpenType fonts support the following `CJKShape` options: `Traditional`, `Simplified`, `JIS1978`, `JIS1983`, `JIS1990`, and `Expert`. OpenType also supports the `NLC` option.

有许多标准规定 CJK 表意字应该长成什么样子。一些字体包含了许多备用字形，以便能够以适当的形式正确显示这些字形。AAT 和 OpenType 字体都支持以下 CJKShape 选项：Traditional、Simplified、JIS1978、JIS1983、JIS1990 和 Expert。OpenType 还支持 NLC 选项。

3.2.4 Vertical typesetting
竖排版

OpenType provides a plethora of features for accommodating the varieties of possibilities needed for vertical typesetting (CJK and others). No capabilities for achieving such vertical typesetting are provided by fontspec, however; please get in touch if there are improvements that could be made.

OpenType 提供了大量功能来适应竖排版（CJK 和其他语言）的各种可能性。然而，fontspec 并没有提供实现这种竖排版的功能；如果有改进的空间，请联系我们。

Example 36: Different standards for CJK ideograph presentation.
不同的 CJK 表意字符表示标准。

—Example graphic not generated—

```
\fontspec{Hiragino Mincho Pro}
{\addfontfeature{CJKShape=Traditional}
\text }                              \\
{\addfontfeature{CJKShape=NLC}
\text }                              \\
{\addfontfeature{CJKShape=Expert}
\text }
```

Table 18: Options for the OpenType font feature 'Vertical'.

Feature	Option	Tag
Vertical =	RotatedGlyphs	`vrt2` †
	AlternatesForRotation	`vrtr` †
	Alternates	`vert` †
	KanaAlternates	`vkna` †
	Kerning	`vkrn` †
	AlternateMetrics	`valt` †
	HalfMetrics	`vhal` †
	ProportionalMetrics	`vpal` †
	ResetAll	

† These feature options can be disabled with `..Off` variants, and reset to default state (neither explicitly on nor off) with `..Reset`.

Part V

Commands for accents and symbols ('encodings')
重音和符号命令（“编码”）

The functionality described in this section is experimental.
本节所描述的功能是实验性的。

In the pre-Unicode era, significant work was required by LaTeX to ensure that input characters in the source could be interpreted correctly depending on file encoding, and that glyphs in the output were selected correctly depending on the font encoding. With Unicode, we have the luxury of a single file and font encoding that is used for both input and output.

在 Unicode 时代之前，LaTeX 需要做大量工作来确保源代码中的输入字符能够根据文件编码正确解释，并且输出中的字形能够根据字体编码正确选择。而在 Unicode 时代，我们有了一个单一的文件和字体编码，用于输入和输出。

While this may provide some illusion that we could get away simply with typing Unicode text and receive correct output, this is not always the case. For a start, hyphenation in particular is language-specific, so tags should be used when switch between languages in a document. The babel and polyglossia packages both provide features for this.

虽然这可能会给我们一些幻觉，认为我们可以简单地输入 Unicode 文本并获得正确的输出，但并非总是如此。首先，特别是连字号化是语言特定的，因此在文档中切换语言时应使用标记。babel 和 polyglossia 包都提供了此类功能。

Multilingual documents will often use different fonts for different languages, not just for style, but for the more pragmatic reason that fonts do not all contain the same glyphs. (In fact, only test fonts such as Code2000 provide anywhere near the full Unicode coverage.) Indeed, certain fonts may be perfect for a certain application but miss a handful of necessary diacritics or accented letters. In these cases, fontspec can leverage the font encoding technology built into LaTeX2 to provide on a per-font basis either provide fallback options or error messages when a desired accent or symbol is not available. However, at present these features can only be provided for input using LaTeX commands rather than Unicode input; for example, typing `\`e` instead of è or `\textcopyright` instead of © in the source file.

多语言文档通常会为不同的语言使用不同的字体，这不仅仅是为了风格，更是出于更实际的原因，即字体不包含相同的字形。（实际上，只有 Code2000 等测试字体提供了几乎完整的 Unicode 覆盖。）确实，某些字体可能非常适合某些应用程序，但缺少少量必要的变音符或重音字母。在这种情况下，fontspec 可以利用内置在 LaTeX2 中的字体编码技术，为每个字体提供后备选项或在所需的重音符或符号不可用时提供错误消息。但是，目前这些功能仅适用于使用 LaTeX 命令输入的输入，而不是 Unicode 输入；例如，在源文件中输入 `` `e `` 而不是 è 或 `\textcopyright` 而不是 ©。

The most widely-used encoding in LaTeX 2ε was T1 with companion 'TS1' symbols provided by the textcomp package. These encodings provided glyphs to typeset text in a variety of western European languages. As with most legacy LaTeX 2ε input methods, accents and symbols were input using encoding-dependent commands such as \`e as described above.

在 LaTeX 2ε 中，最广泛使用的编码是带有伴随 TS1' 符号的 T1。这些编码提供了用于排版各种西欧语言文本的字形。与大多数旧版 LaTeX 2ε 输入方法一样，重音符和符号是使用依赖于编码的命令输入的，如上所述的 \e。

As of 2017, in LaTeX 2ε on XeTeX and LuaTeX, the default encoding is TU, which uses Unicode for input and output. The TU encoding provides appropriate encoding-dependent definitions for input commands to match the coverage of the T1+TS1 encodings. Wider coverage is not provided by default since (a) each font will provide different glyph coverage, and (b) it is expected that most users will be writing with direct Unicode input.

截至 2017 年，在 XeTeX 和 LuaTeX中，LaTeX 2ε的默认编码为TU，这使用 Unicode 作为输入和输出。TU 编码为输入命令提供了适当的编码依赖定义，以匹配T1 + TS1 编码的覆盖范围。默认情况下不提供更广泛的覆盖范围，因为 (a) 每个字体提供不同的字形覆盖范围，和 (b) 大多数用户将使用直接的 Unicode 输入进行编写。

For those users who do need finer-grained control, fontspec provides an interface for a more extensible system.

对于那些需要更细粒度控制的用户，fontspec 提供了一个更可扩展的系统的接口。

1 A new Unicode-based encoding from scratch 从头开始创建一个基于 Unicode 的新编码

Let's say you need to provide support for a document originally written with fonts in the OT2 encoding, which contains encoding-dependent commands for Cyrillic letters. An example from the OT2 encoding definition file (ot2enc.def) reads:

假设您需要提供对最初使用OT2 编码的字体编写的文档的支持，该编码包含依赖于 Cyrillic 字母的编码依赖命令。OT2 编码定义文件 (ot2enc.def) 中的一个示例如下：

```
\DeclareTextSymbol{\CYRIE}{OT2}{5}
\DeclareTextSymbol{\CYRDJE}{OT2}{6}
\DeclareTextSymbol{\CYRTSHE}{OT2}{7}
\DeclareTextSymbol{\cyrnje}{OT2}{8}
\DeclareTextSymbol{\cyrlje}{OT2}{9}
\DeclareTextSymbol{\cyrdzhe}{OT2}{10}
```

To recreate this encoding in a form suitable for fontspec, create a new file named, say, fontrange-cyr.def and populate it with

要在适合 fontspec 的形式中重新创建此编码，请创建一个名为，例如，fontrange-cyr.def 的新文件，并填充为：

```
...
\DeclareTextSymbol{\CYRIE}  {\LastDeclaredEncoding}{"0404}
```

```
\DeclareTextSymbol{\CYRDJE} {\LastDeclaredEncoding}{"0402}
\DeclareTextSymbol{\CYRTSHE}{\LastDeclaredEncoding}{"040B}
\DeclareTextSymbol{\cyrnje} {\LastDeclaredEncoding}{"045A}
\DeclareTextSymbol{\cyrlje} {\LastDeclaredEncoding}{"0459}
\DeclareTextSymbol{\cyrdzhe}{\LastDeclaredEncoding}{"045F}
...
```

The numbers "0404, "0402, ..., are the Unicode slots (in hexadecimal) of each glyph respectively. The fontspec package provides a number of shorthands to simplify this style of input; in this case, you could also write

数字"0404、"0402 等是各个字形的 Unicode 插槽（用十六进制表示）。fontspec 软件包提供了许多简写方式来简化此输入样式；在这种情况下，您也可以编写如下代码：

```
\EncodingSymbol{\CYRIE}{"0404}
...
```

To use this encoding in a fontspec font, you would first add this to your preamble:

要在 fontspec 字体中使用此编码，您首先需要将其添加到导言区：

```
\DeclareUnicodeEncoding{unicyr}{
  \input{fontrange-cyr.def}
}
```

Then follow it up with a font loading call such as

然后跟随一个字体加载调用，例如：

```
\setmainfont{...}[NFSSEncoding=unicyr]
```

The first argument `unicyr` is the name of the 'encoding' to use in the font family. (There's nothing special about the name chosen but it must be unique.) The second argument to `\DeclareUnicodeEncoding` also allows adjustments to be made for per-font changes. We'll cover this use case in the next section.

第一个参数 `unicyr` 是字体族中要使用的“编码”的名称。（选择的名称没有什么特别之处，但它必须是唯一的。）`\DeclareUnicodeEncoding` 的第二个参数也允许进行针对每个字体的调整。我们将在下一节中讨论这种用例。

2 Adjusting a pre-existing encoding 调整预设编码

There are three reasons to adjust a pre-existing encoding: to add, to remove, and to redefine some symbols, letters, and/or accents.

有三个原因需要调整预设编码：添加、删除和重新定义一些符号、字母和/或重音符号。

When adding symbols, etc., simply write

当添加符号等时，只需编写以下内容：

```
\DeclareUnicodeEncoding{unicyr}{
  \input{tuenc.def}
```

```
  \input{fontrange-cyr.def}
  \EncodingSymbol{\textruble}{"20BD}
}
```

Of course if you consistently add a number of symbols to an encoding it would be a good idea to create a new `fontrange-XX.def` file to suit your needs.

当然,如果您要向编码中添加一些符号,建议为您的需求创建一个新的 `fontrange-XX.def` 文件。

When removing symbols, use the `\UndeclareSymbol{`⟨*cmd*⟩`}` command. For example, if you a loading a font that you know is missing, say, the interrobang (not that unusual a situation), you might write:

当删除符号时，请使用 `\UndeclareSymbol{`⟨*cmd*⟩`}` 命令。例如，如果您正在加载一种字体，您知道它缺少叹问号感叹号符号（这种情况并不罕见），您可以编写：

```
\DeclareUnicodeEncoding{nobang}{
  \input{tuenc.def}
  \UndeclareSymbol\textinterrobang
}
```

Provided that you use the command `\textinterrobang` to typeset this symbol, it will appear in fonts with the default encoding, while in any font loaded with the `nobang` encoding an attempt to access the symbol will either use the default fallback definition or return an error, depending on the symbol being undeclared.

只要您使用命令 `\textinterrobang` 来排版此符号，它将出现在具有默认编码的字体中，而在任何加载了 `nobang` 编码的字体中，尝试访问该符号将使用默认回退定义或返回错误，具体取决于符号是否未声明。

The third use case is to redefine a symbol or accent. The most common use case in this scenario is to adjust a specific accent command to either fine-tune its placement or to 'fake' it entirely. For example, the underdot diacritic is used in typeset Sanskrit, but it is not necessarily included as an accent symbol is all fonts. By default the underdot is defined in `TU` as:

第三种用例是重新定义符号或重音符号。在这种情况下，最常见的用例是将特定重音命令调整为微调其位置或完全“伪造”它。例如，下划线变音符用于排版梵文，但并不一定在所有字体中都包括为一个重音符号。默认情况下，`TU` 中的下划线被定义为：

```
\EncodingAccent{\d}{"0323}
```

For fonts with a missing (or poorly-spaced) `"0323` accent glyph, the 'traditional' TEX fake accent construction could be used instead:

对于缺少（或间距不好）的 `"0323` 重音符号的字体，可以使用“传统”的 TEX 伪造重音构造代替：

```
\DeclareUnicodeEncoding{fakeacc}{
  \input{tuenc.def}
  \EncodingCommand{\d}[1]{%
    \hmode@bgroup
      \o@lign{\relax#1\crcr\hidewidth\ltx@sh@ft{-1ex}.\hidewidth}%
```

```
    \egroup
  }
}
```

This would be set up in a document as such:

这将在文档中设置如下：

```
\newfontfamily\sanskitfont{CharisSIL}
\newfontfamily\titlefont{Posterama}[NFSSEncoding=fakeacc]
```

Then later in the document, no additional work is needed:

那么在文档的后面，就不需要再做额外的工作了。

```
...{\titlefont   kalita\d m}...  % <- uses fake accent
...{\sanskitfont kalita\d m}...  % <- uses real accent
```

To reiterate from above, typing this input with Unicode text ('kalitaṃ') will *bypass* this encoding mechanism and you will receive only what is contained literally within the font.

重申一下，使用 Unicode 文本 ('kalitaṃ') 输入时，会绕过编码机制，你将只会得到在字体中包含的内容。

3 Summary of commands
命令概述

The LaTeX 2_ε kernel provides the following font encoding commands suitable for Unicode encodings:

LaTeX 2_ε 核心提供了适用于 Unicode 编码的以下字体编码命令：

`\DeclareTextCommand{`⟨*command*⟩`}{`⟨*encoding*⟩`}[`⟨*num*⟩`][`⟨*default*⟩`]{`⟨*code*⟩`}`

`\DeclareUnicodeAccent{`⟨*command*⟩`}{`⟨*encoding*⟩`}{`⟨*slot*⟩`}`

`\DeclareTextSymbol{`⟨*command*⟩`}{`⟨*encoding*⟩`}{`⟨*slot*⟩`}`

`\DeclareTextComposite{`⟨*command*⟩`}{`⟨*encoding*⟩`}{`⟨*letter*⟩`}{`⟨*slot*⟩`}`

`\DeclareTextCompositeCommand{`⟨*command*⟩`}{`⟨*encoding*⟩`}{`⟨*letter*⟩`}{`⟨*code*⟩`}`

`\UndeclareTextCommand{`⟨*command*⟩`}{`⟨*encoding*⟩`}`

See `fntguide.pdf` for full documentation of these. As shown above, the following shorthands are provided by fontspec to simplify the process of defining Unicode font range encodings:

详细文档请参见 `fntguide.pdf`。如上所示，下列快捷方式由 fontspec 提供，以简化定义 Unicode 字体范围编码的过程：

`\EncodingCommand{`⟨*command*⟩`}[`⟨*num*⟩`][`⟨*default*⟩`]{`⟨*code*⟩`}`

`\EncodingAccent{`⟨*command*⟩`}{`⟨*code*⟩`}`

`\EncodingSymbol{`⟨*command*⟩`}{`⟨*code*⟩`}`

`\EncodingComposite{`⟨*command*⟩`}{`⟨*letter*⟩`}{`⟨*slot*⟩`}`

`\EncodingCompositeCommand{`⟨*command*⟩`}{`⟨*letter*⟩`}{`⟨*code*⟩`}`

`\UndeclareSymbol{`⟨*command*⟩`}`

`\UndeclareAccent{`⟨*command*⟩`}`

`\UndeclareCommand{`⟨*command*⟩`}`

`\UndeclareComposite{`⟨*command*⟩`}{`⟨*letter*⟩`}`

Part VI

LuaTEX-only font features
LuaTEX-专有字体特性

1 Different font technologies and shapers
不同的字体技术和渲染器

LuaTEX does not directly support any font rendering technologies out of the box, it requires additional functionality to be added to properly support and control technologies such as OpenType.

LuaTEX 并不直接支持任何字体渲染技术，需要添加额外的功能以适当支持和控制诸如 OpenType 等技术。

Using the `Renderer` feature, there are a number of options that fontspec can pass to the engine to control which font technology is being used. Pre-2019, there were two options provided by luaotfload that generally did not require user intervention.

使用 `Renderer` 特性，fontspec 可以向引擎传递一些选项以控制正在使用的字体技术。2019 年之前，luaotfload 提供了两个选项，通常不需要用户干预。

- `Renderer = Node` : the default 'mode' for typesetting OpenType fonts.
  `Renderer = Node`：排版 OpenType 字体的默认“模式”。
- `Renderer = Base` : a simplified mode useful only in a limited number of situations such as mathematics typesetting.
  `Renderer = Base`：仅在少数情况下有用的简化模式，例如数学排版。

From 2019 the possibility of using the Harfbuzz text shaping engine within LuaTEX has been developed by Khaled Hosny. When running a suitable LuaTEX engine with Harfbuzz support, fontspec provides the following options:

自 2019 年以来，Khaled Hosny 开发了在 LuaTEX 中使用 Harfbuzz 文本整形引擎的可能性。在运行带有 Harfbuzz 支持的适当 LuaTEX 引擎时，fontspec 提供以下选项：

- `Renderer = HarfBuzz` : use the Harfbuzz engine without an explicit 'shaper' (the old `Harfbuzz` name is kept for compatibility).
  `Renderer = HarfBuzz`：使用 Harfbuzz 引擎而无需显式“整形器”（旧的 `Harfbuzz` 名称仍保留以保持兼容性）。
- `Renderer = OpenType` : use the Harfbuzz engine with the OpenType shaper.
  `Renderer = OpenType`：使用 OpenType 整形器的 Harfbuzz 引擎。
- `Renderer = AAT` : use the Harfbuzz engine with the AAT shaper.
  `Renderer = AAT`：使用 AAT 整形器的 Harfbuzz 引擎。
- `Renderer = Graphite` : use the Harfbuzz engine with the Graphite shaper.
  `Renderer = Graphite`：使用 Graphite 整形器的 Harfbuzz 引擎。

- `Renderer` = ⟨*foo*⟩ : use the Harfbuzz engine with the ⟨*foo*⟩ shaper.
  `Renderer` = ⟨*foo*⟩：使用 ⟨*foo*⟩ 整形器的 Harfbuzz 引擎。

Support for the Harfbuzz renderer is preliminary and may be improved over time. Please treat the interface for Harfbuzz fonts as subject to change.

对于 Harfbuzz 渲染器的支持是初步的，可能会随着时间的推移得到改进。请将 Harfbuzz 字体的界面视为可能会改变的内容。

2 Custom font features
自定义字体特性

LuaTEX, via the luaotfload package, allows the definition and re-definition of custom OpenType features for a selected font. This facility is particularly useful to implement custom substitutions or to disable unwanted but not all ligatures.

通过 luaotfload 包，LuaTEX 允许为选定的字体定义和重新定义自定义 OpenType 特性。此功能特别有用，可用于实现自定义替换或禁用不需要但不是所有连字符。

Figure 1 shows an minimal example of this type of functionality. This example creates a new OpenType feature, `oneb`, which substitutes the glyph when typesetting '1' for the named glyph `one.ss01`. The glyph names are font specific and can be interrogated with third-party software such as *FontForge*.

图 1展示了此类型功能的一个最简示例。该示例创建了一个新的 OpenType 特性 `oneb`，当排版 '1' 时，将其替换为名为 `one.ss01` 的字形。字形名称是字体特定的，可以使用第三方软件（如 *FontForge*）来查询。

A third-party collection of additional examples are maintained in the repository '`fonts-in-luatex`'[16]. These examples are intended to correct or adjust font features in a range of commercial fonts and provide a good introduction to some of the possibilities that LuaTEX affords.

第三方的额外示例集合存储在 '`fonts-in-luatex`'[17]中。这些示例旨在纠正或调整商业字体的特性，并为LuaTEX 提供了一些可能性的良好介绍。

Please refer to the LuaTEX/luaotfload documentation for more details.

有关更多详细信息，请参阅LuaTEX/luaotfload 文档。

[16] https://github.com/mewtant/fonts-in-luatex
[17] https://github.com/mewtant/fonts-in-luatex

Figure 1: An example of custom font features.
自定义字体特性的示例。

```
\documentclass{article}
\usepackage{fontspec}
\directlua{
    fonts.handlers.otf.addfeature {
        name = "oneb",
        type = "substitution",
        data = {
                ["1"] = "one.ss01",
        }
    }
}
\setmainfont{Vollkorn-Regular.otf}[RawFeature=+oneb]
\begin{document}
123456789\
\end{document}
```

Part VII

Fonts and features with XeTeX
使用 XeTeX 的字体和特性

1 XeTeX-only font features
XeTeX 专有的字体特性

The features described here are available for any font selected by fontspec.

此处描述的特性适用于任何由 fontspec 选定的字体。

1.1 Mapping
映射

The `Mapping` feature enables a XeTeX text-mapping scheme, with an example shown in Example 37.

`Mapping` 特性启用了 XeTeX 的文本映射方案，如 Example 37 中所示。

Only one mapping can be active at a time and a second call to `Mapping` will override the first. Using the `tex-text` mapping is also equivalent to writing `Ligatures=TeX`. The use of the latter syntax is recommended for better compatibility with LuaTeX documents.

一次只能激活一个映射，第二次调用 `Mapping` 将覆盖第一次调用。使用 `tex-text` 映射等价于写 `Ligatures=TeX`。建议使用后者的语法，以更好地兼容 LuaTeX文档。

1.2 Different font technologies: AAT, OpenType, and Graphite
不同的字体技术：AAT、OpenType 和 Graphite

Note that from 2020 it appears that XeTeX can no longer support AAT fonts in macOS.
请注意，从 2020 年开始，在 macOS 上似乎已不再支持 AAT 字体。

XeTeX supports three rendering technologies for typesetting, selected with the `Renderer` font feature. The first, `AAT`, is that provided only by macOS. The second, `OpenType`, is an open source OpenType interpreter. It provides greater support for OpenType features, notably contextual arrangement, over `AAT`. The third is `Graphite`, which is an alternative to OpenType with particular features for less-common languages and the capability for more powerful font options. Features for `OpenType` have already been discussed in IV

Example 37: XeTeX's `Mapping` feature.
XeTeX 的 `Mapping` 特性示例。

—Example graphic not generated—

```
\fontspec{texgyrepagella-regular.otf}[Mapping=tex-text]
``!`A small amount of---text!''
```

on page 43; Graphite and AAT features are discussed later in Section 2 on the following page and Section 3 on page 80.

XeTeX 支持三种排版渲染技术，可通过 Renderer 字体特性选择。第一种是 AAT，仅由 macOS 提供。第二种是开源的 OpenType 解释器 OpenType。它提供了更好的 OpenType 特性支持，尤其是上下文排版功能，超过了 AAT。第三种是 Graphite，是 OpenType 的替代品，具有特定的支持较少见语言的特性，以及更强大的字体选项功能。OpenType 特性已在 IV on page 43 中讨论；Graphite 和 AAT 特性将在 Section 2 on the next page 和 Section 3 on page 80 中讨论。

Unless you have a particular need, the Renderer feature is rarely explicitly required: for OpenType fonts, the OpenType renderer is used automatically, and for AAT fonts, AAT is chosen by default. Some fonts, however, will contain font tables for multiple rendering technologies, such as the Hiragino Japanese fonts distributed with macOS, and in these cases one over the other may be preferred.

除非您有特定需求，否则通常不需要显式指定 Renderer 特性：对于 OpenType 字体，将自动使用 OpenType 渲染器，而对于 AAT 字体，默认选择 AAT。然而，一些字体将包含多个渲染技术的字体表，例如随 macOS 发行的 Hiragino 日文字体，在这些情况下，可能会更喜欢其中之一。

Among some other font features only available through a specific renderer, OpenType provides for the Script and Language features, which allow different font behaviour for different alphabets and languages; see Section 2 on page 46 for the description of these features. *Because these font features can change which features are able to be selected for the font instance, they are selected by fontspec before all others and will automatically and without warning select the OpenType renderer.*

在某些其他字体特性中，只有通过特定渲染器才能使用，OpenType 提供了 Script 和 Language 特性，允许在不同的字母表和语言中使用不同的字体行为；详见Section 2 on page 46 中对这些特性的描述。因为这些字体特性可能会改变可以选择的字体实例的特性，所以它们会在所有其他特性之前由 *fontspec* 选择，并且将自动且没有警告地选择 *OpenType* 渲染器。

1.3 Optical font sizes
光学字体大小

Multiple Master fonts are parameterised over orthogonal font axes, allowing continuous selection along such features as weight, width, and optical size. Whereas an OpenType font will have only a few separate optical sizes, a Multiple Master font's optical size can be specified over a continuous range. Unfortunately, this flexibility makes it harder to create an automatic interface through LaTeX, and the optical size for a Multiple Master font must always be specified explicitly.

多主字体是在正交字体轴上参数化的，允许沿着这些特性，例如重量、宽度和光学大小进行连续选择。而 OpenType 字体只有几个独立的光学大小，多主字体的光学大小可以在连续的范围内指定。不幸的是，这种灵活性使得通过 LaTeX 自动创建接口变得更加困难，并且必须始终明确指定多主字体的光学大小。

```
\fontspec{Minion MM Roman}[OpticalSize=11]
 MM optical size test                    \\
```

```
\fontspec{Minion MM Roman}[OpticalSize=47]
 MM optical size test                    \\
\fontspec{Minion MM Roman}[OpticalSize=71]
 MM optical size test                    \\
```

1.4 Vertical typesetting
纵向排版

XeTeX provides for vertical typesetting simply with the ability to rotate the individual glyphs as a font is used for typesetting, as shown in Example 38.

XeTeX 提供了纵向排版的简单方法，只需在用于排版的字体中旋转单个字形即可，如 Example 38 所示。

No actual provision is made for typesetting top-to-bottom languages; for an example of how to do this, see the vertical Chinese example provided in the XeTeX documentation.

实际上没有提供排版自上而下的语言的功能；有关如何实现这一点的示例，请参见 XeTeX 文档中提供的纵向中文示例。

2 The Graphite renderer
Graphite 渲染器

Since the Graphite renderer is designed for less common scripts and languages, usually with specific or unique requirements, Graphite features are not standard across fonts.

由于 Graphite 渲染器设计用于不太常见的书写和语言，通常具有特定或独特的要求，因此字体的 Graphite 特性在各种字体之间是不标准的。

Currently fontspec does not support a convenient interface to select Graphite font features and all selection must be done via 'raw' font feature selection.

目前，fontspec 不支持方便的界面选择 Graphite 字体特性，所有选择都必须通过“原始”字体特性选择完成。

Here's an example:

以下是一个例子

```
\fontspec{Charis SIL}[
  Renderer=Graphite,
```

Example 38: Vertical typesetting.
纵向排版。

```
\fontspec{Hiragino Mincho Pro}
\verttext

\fontspec{Hiragino Mincho Pro}[Renderer=AAT,Vertical=RotatedGlyphs]
\rotatebox{-90}{\verttext}% requires the graphicx package
```

—Example graphic not generated—

```
    RawFeature={Uppercase Eng alternates=Large eng on baseline}]
  D
```

Here's another:

以下是另一个例子：

```
\fontspec{AwamiNastaliq-Regular.ttf}[Renderer=Graphite] ^^^^0b5
\addfontfeature{RawFeature={Lam with V=V over bowl}}    ^^^^0b5
```

3 macOS's AAT fonts
macOS 的 AAT 字体

> ***Warning!***
> 警告！ *XeTeX's implementation on macOS is currently in a state of flux and the information contained below may well be wrong from 2013 onwards. There is a good chance that the features described in this section will not be available any more as XeTeX's completes its transition to a cross-platform–only application. All examples in this section have now been removed.*
>
> *XeTeX 在 macOS 上的实现目前处于变化状态，下面的信息从 2013 年开始可能已经不正确。随着 XeTeX 完成其过渡到跨平台应用程序，这一节描述的功能很可能不再可用。本节中的所有示例现已被删除。*

macOS's font technology began life before the ubiquitous-OpenType era and revolved around the Apple-invented 'AAT' font format. This format had some advantages (and other disadvantages) but it never became widely popular in the font world.

macOS 的字体技术在普及的 OpenType 时代之前诞生，并且围绕着由苹果发明的“AAT”字体格式。这种格式有一些优点（以及其他缺点），但在字体世界中并没有广泛流行。

Nonetheless, this is the font format that was first supported by XeTeX (due to its pedigree on macOS in the first place) and was the first font format supported by fontspec. A number of fonts distributed with macOS are still in the AAT format, such as 'Skia'.

尽管如此，这是 XeTeX 最早支持的字体格式（由于它在 macOS 上的渊源），也是 fontspec 支持的第一个字体格式。一些随 macOS 分发的字体仍然是 AAT 格式，例如“Skia”。

3.1 Ligatures
连字

`Ligatures` refer to the replacement of two separate characters with a specially drawn glyph for functional or æsthetic reasons. For AAT fonts, you may choose from any combination of `Required`, `Common`, `Rare` (or `Discretionary`), `Logos`, `Rebus`, `Diphthong`, `Squared`, `AbbrevSquared`, and `Icelandic`.

连字是指出于功能或美观原因替换两个独立字符为特殊绘制的字形。对于 AAT 字体，您可以选择任何组合的 `Required`、`Common`、`Rare`（或 `Discretionary`）、`Logos`、`Rebus`、`Diphthong`、`Squared`、`AbbrevSquared` 和 `Icelandic`。

Some other Apple AAT fonts have those 'Rare' ligatures contained in the `Icelandic` feature. Notice also that the old TEX trick of splitting up a ligature with an empty brace pair does not work in XETEX; you must use a 0 pt kern or `\hbox` (*e.g.*, `\null`) to split the characters up if you do not want a ligature to be performed (the usual examples for when this might be desired are words like 'shelffull').

一些其他的 Apple AAT 字体包含在 `Icelandic` 功能中的这些“Rare”连字。请注意，旧版的 TEX 技巧使用空的花括号对来分解连字在 XETEX 中不起作用；如果您不希望执行连字，则必须使用 0,pt kern 或 `\hbox`（例如 `\null`）来分解字符（通常需要执行这种操作的例子是像“shelffull”这样的单词）。

3.2 Letters
字母

The `Letters` feature specifies how the letters in the current font will look. For AAT fonts, you may choose from `Normal`, `Uppercase`, `Lowercase`, `SmallCaps`, and `InitialCaps`.

`Letters` 功能指定当前字体中字母的外观。对于 AAT 字体，您可以选择 `Normal`、`Uppercase`、`Lowercase`、`SmallCaps` 和 `InitialCaps`。

3.3 Numbers
数字

The `Numbers` feature defines how numbers will look in the selected font. For AAT fonts, they may be a combination of `Lining` or `OldStyle` and `Proportional` or `Monospaced` (the latter is good for tabular material). The synonyms `Uppercase` and `Lowercase` are equivalent to `Lining` and `OldStyle`, respectively. The differences have been shown previously in Section 2 on page 28.

数字特性定义了在所选字体中数字的外观。对于 AAT 字体，它们可以是 `Lining` 或 `OldStyle` 以及 `Proportional` 或 `Monospaced` 的组合（后者适用于表格材料）。同义词 `Uppercase` 和 `Lowercase` 分别等同于 `Lining` 和 `OldStyle`。这些差异在Section 2 on page 28中已经展示过了。

3.4 Contextuals
上下文相关

This feature refers to glyph substitution that vary by their position; things like contextual swashes are implemented here. The options for AAT fonts are `WordInitial`, `WordFinal` (Example **??**), `LineInitial`, `LineFinal`, and `Inner` (Example **??**, also called 'non-final' sometimes). As non-exclusive selectors, like the ligatures, you can turn them off by prefixing their name with `No`.

该特性涉及按位置变化的字形替换；例如实现上下文花押的就在这里。AAT 字体的选项包括 `WordInitial`、`WordFinal`（Example **??**）、`LineInitial`、`LineFinal` 和 `Inner`（Example **??**，有时也称为“非终止”）。像连字一样，它们也可以通过在它们的名称前缀中添加 `No` 来关闭。

3.5 Vertical position
垂直位置

The VerticalPosition feature is used to access things like subscript (Inferior) and superscript (Superior) numbers and letters (and a small amount of punctuation, sometimes). The Ordinal option is (supposed to be) contextually sensitive to only raise characters that appear directly after a number.

VerticalPosition 特性用于访问上下标（Inferior 和 Superior）数字、字母（有时也包括少量标点符号）。Ordinal 选项是（应该是）仅在数字之后直接出现的字符上下文敏感的。

The realscripts package redefines the \textsubscript and \textsuperscript commands to use the above font features, including for use in footnote labels.

realscripts 宏包重新定义了\textsubscript 和 \textsuperscript 命令，使用上述字体特性，包括用于脚注标签。

3.6 Fractions
分数

Many fonts come with the capability to typeset various forms of fractional material. This is accessed in fontspec with the Fractions feature, which may be turned On or Off in both AAT and OpenType fonts.

许多字体都具有排版各种形式的分数材料的能力。这可以通过在 fontspec 中使用 Fractions 特性来访问，可以在 AAT 和 OpenType 字体中将其打开（On）或关闭（Off）。

In AAT fonts, the 'fraction slash' or solidus character, is to be used to create fractions. When Fractions are turned On, then only pre-drawn fractions will be used.

在 AAT 字体中，将使用“分数斜杠”或实线符号来创建分数。当打开 Fractions 时，只使用预绘制的分数。

Using the Diagonal option (AAT only), the font will attempt to create the fraction from superscript and subscript characters.

使用 Diagonal 选项（仅限 AAT），字体将尝试从上下标字符中创建分数。

Some (Asian fonts predominantly) also provide for the Alternate feature.

一些（主要是亚洲字体）还提供了 Alternate 特性。

3.7 Variants
变体

The Variant feature takes a single numerical input for choosing different alphabetic shapes. See Section 1 on page 85 for a way to assign names to variants, which should be done on a per-font basis.

Variant 特性采用单个数字输入来选择不同的字母形状。有关将变体命名的方法，请参见Section 1 on page 85，这应该根据字体而定。

3.8 Alternates
替代

Selection of `Alternates` again must be done numerically. See Section 1 on page 85 for a way to assign names to alternates, which should be done on a per-font basis.

选择 `Alternate` 必须再次通过数字进行。有关将备用项命名的方法，请参见Section 1 on page 85，这应该根据字体而定。

3.9 Style
样式

The options of the `Style` feature are defined in AAT as one of the following: `Display`, `Engraved`, `IlluminatedCaps`, `Italic`, `Ruby`,[18] `TallCaps`, or `Titling`.

`Style` 功能的选项在 AAT 中被定义为以下之一：`Display`、`Engraved`、`IlluminatedCaps`、`Italic`、`Ruby`、`TallCaps` 或 `Titling`。

Typical examples for these features are shown in 3.1.12.

这些功能的典型示例显示在 3.1.12 中。

3.10 CJK shape
CJK 形状

There have been many standards for how CJK ideographic glyphs are 'supposed' to look. Some fonts will contain many alternate glyphs in order to be able to display these gylphs correctly in whichever form is appropriate. Both AAT and OpenType fonts support the following `CJKShape` options: `Traditional`, `Simplified`, `JIS1978`, `JIS1983`, `JIS1990`, and `Expert`. OpenType also supports the `NLC` option.

有许多标准规定了 CJK 表意文字的“应该”样式。一些字体包含许多替代字形，以便能够以适当的形式正确显示这些字形。AAT 和 OpenType 字体都支持以下 `CJKShape` 选项：`Traditional`、`Simplified`、`JIS1978`、`JIS1983`、`JIS1990` 和 `Expert`。OpenType 还支持 `NLC` 选项。

3.11 Character width
字符宽度

See 3.2.2 on page 64 for relevant examples; the features are the same between OpenType and AAT fonts. AAT also allows `CharacterWidth=Default` to return to the original font settings.

相关示例请参见 3.2.2 on page 64；OpenType 和 AAT 字体之间的特性是相同的。AAT 还允许 `CharacterWidth=Default` 返回到原始字体设置。

[18] 'Ruby' refers to a small optical size, used in Japanese typography for annotations.

3.12 Diacritics
变音符号

Diacritics are marks, such as the acute accent or the tilde, applied to letters; they usually indicate a change in pronunciation. In Arabic scripts, diacritics are used to indicate vowels. You may either choose to `Show`, `Hide` or `Decompose` them in AAT fonts. The `Hide` option is for scripts such as Arabic which may be displayed either with or without vowel markings. E.g., `\fontspec[Diacritics=Hide]{...}`

变音符号是应用于字母的标记，例如锐音符号或波浪符号，它们通常表示发音的变化。在阿拉伯语书写中，变音符号用于表示元音。在 AAT 字体中，您可以选择 `Show`、`Hide` 或 `Decompose`。`Hide` 选项适用于像阿拉伯语这样可以显示带或不带元音标记的书写系统。例如，`\fontspec[Diacritics=Hide]{...}`。

Some older fonts distributed with macOS included '`O/`' *etc.* as shorthand for writing 'Ø' under the label of the `Diacritics` feature. If you come across such fonts, you'll want to turn this feature off (imagine typing `hello/goodbye` and getting 'helløgoodbye' instead!) by decomposing the two characters in the diacritic into the ones you actually want. I recommend using the proper LaTeX input conventions for obtaining such characters instead.

某些旧版的配有 macOS 的字体使用 '`O/`' 等作为书写 `Diacritics` 功能标签下 'Ø' 的速记符号。如果您遇到这样的字体，您需要将此功能关闭（想象一下输入 `hello/goodbye` 却得到 'helløgoodbye' 的结果！），通过将变音符号中的两个字符分解成您实际需要的字符。我建议使用适当的 LaTeX 输入约定来获取这样的字符。

3.13 Annotation
注释

Various Asian fonts are equipped with a more extensive range of numbers and numerals in different forms. These are accessed through the `Annotation` feature with the following options: `Off`, `Box`, `RoundedBox`, `Circle`, `BlackCircle`, `Parenthesis`, `Period`, `RomanNumerals`, `Diamond`, `BlackSquare`, `BlackRoundSquare`, and `DoubleCircle`.

各种亚洲字体配备了更广泛的不同形式的数字和数字符号。这些是通过 `Annotation` 功能访问的，具有以下选项：`Off`、`Box`、`RoundedBox`、`Circle`、`BlackCircle`、`Parenthesis`、`Period`、`RomanNumerals`、`Diamond`、`BlackSquare`、`BlackRoundSquare` 和 `DoubleCircle`。

Part VIII

Customisation and programming interface
自定义和编程接口

This chapter describes the current interfaces and hooks that use fontspec for various macro programming purposes.

本章介绍使用 fontspec 的当前接口和钩子，以用于各种宏编程目的。

1 Defining new features
定义新特性

This package cannot hope to contain every possible font feature. Three commands are provided for selecting font features that are not provided for out of the box. If you are using them a lot, chances are I've left something out, so please let me know.

该软件包无法包含所有可能的字体特性。提供了三个命令来选择不提供的字体特性。如果您经常使用它们，很可能我遗漏了某些功能，请告诉我。

`\newAATfeature` New AAT features may be created with this command:

`\newAATfeature{`⟨*feature*⟩`}{`⟨*option*⟩`}{`⟨*feature code*⟩`}{`⟨*selector code*⟩`}`

Use the XeTeX file `AAT-info.tex` to obtain the code numbers. See Example 39.

使用此命令可以创建新的 AAT 特性：

`\newAATfeature{`⟨*feature*⟩`}{`⟨*option*⟩`}{`⟨*feature code*⟩`}{`⟨*selector code*⟩`}`

使用 XeTeX 文件 `AAT-info.tex` 获取代码编号。参见 Example 39。

`\newopentypefeature` New OpenType features may be created with this command:

使用此命令可以创建新的 OpenType 特性：

`\newopentypefeature{`⟨*feature*⟩`}{`⟨*option*⟩`}{`⟨*feature tag*⟩`}`

The synonym `\newICUfeature` is deprecated.

Example 39: Assigning new AAT features.
分配新的 AAT 特性。

```
\newAATfeature{Alternate}{HoeflerSwash}{17}{1}
\fontspec{Hoefler Text Italic}[Alternate=HoeflerSwash]
 This is XeTeX by Jonathan Kew.
```

—Example graphic not generated—

同义词 \newICUfeature 已弃用。

Here's what it would look like in practise:

在实践中，它看起来像这样：

```
\newopentypefeature{Style}{NoLocalForms}{-locl}
```

\newfontfeature

In case the above commands do not accommodate the desired font feature (perhaps a new XeTeX feature that fontspec hasn't been updated to support), a command is provided to pass arbitrary input into the font selection string:

如果上述命令不能满足所需的字体特性（也许是 XeTeX 的新特性，fontspec 还没有更新以支持），则提供了一个命令将任意输入传递到字体选择字符串中：

\newfontfeature{⟨*name*⟩}{⟨*input string*⟩}

For example, Zapfino used to contain an AAT feature 'Avoid d-collisions'. To access it with this package, you could do some like the following:

例如，Zapfino 曾经包含一个 AAT 特性“避免 d-collisions”。为了使用此软件包访问它，可以执行以下操作：

```
\newfontfeature{AvoidD}  {Special= Avoid d-collisions}
\newfontfeature{NoAvoidD}{Special=!Avoid d-collisions}
\fontspec{Zapfino}[AvoidD,Variant=1]
 sockdolager rubdown                \\
\fontspec{Zapfino}[NoAvoidD,Variant=1]
 sockdolager rubdown
```

The advantage to using the \newAATfeature and \newopentypefeature commands instead of \newfontfeature is that they check if the selected font actually contains the desired font feature at load time. By contrast, \newfontfeature will not give a warning for improper input.

使用 \newAATfeature 和 \newopentypefeature 命令的优点是它们在加载时检查所选字体是否实际包含所需的字体特性。相比之下，\newfontfeature 不会对不正确的输入发出警告。

2 Defining new scripts and languages
定义新的脚本和语言

\newfontscript
\newfontlanguage

While the scripts and languages listed in Table 2 and Table 3 are intended to be comprehensive, there may be some missing; alternatively, you might wish to use different names to access scripts/languages that are already listed. Adding scripts and languages can be performed with the \newfontscript and \newfontlanguage commands. For example,

尽管 Table 2 和 Table 3 中列出的脚本和语言是全面的，但可能有一些遗漏；或者，您可能希望使用不同的名称来访问已经列出的脚本/语言。可以使用 \newfontscript 和 \newfontlanguage 命令添加脚本和语言。例如：

```
\newfontscript{Arabic}{arab}
\newfontlanguage{Zulu}{ZUL}
```

The first argument is the fontspec name, the second the OpenType tag. The advantage to using these commands rather than `\newfontfeature` (see Section 1 on page 85) is the error-checking that is performed when the script or language is requested.

第一个参数是 fontspec 的名称，第二个参数是 OpenType 标签。使用这些命令而不是 `\newfontfeature`（请参见 Section 1 on page 85）的优点在于，在请求脚本或语言时执行的错误检查。

Both commands accept a comma-separated list of OpenType tags in order of preference. This permits, for example, supporting both new and old versions of a language tag with a common user interface:

这两个命令都接受以首选顺序的逗号分隔的 OpenType 标签列表。例如，这允许支持具有公共用户界面的语言标签的新旧版本：

```
\newfontlanguage{Turkish}{TRK,TUR}
```

Here, a font that is requested with `Script=Turkish` will first be checked for the OpenType language tag `TRK`, which will be selected if available. If not available, the `TUR` tag will be queried and used if possible as a fallback.

在这里，使用 `Script=Turkish` 请求的字体将首先检查 OpenType 语言标签 `TRK`，如果可用，将选择它。如果不可用，则查询并使用 `TUR` 标签作为备用。

3 Going behind fontspec's back 绕过 fontspec

Expert users may wish not to use fontspec's feature handling at all, while still taking advantage of its LaTeX font selection conveniences. The `RawFeature` font feature allows font feature selection using a literal feature selection string if you happen to have the OpenType feature tag memorised. More importantly, this can be used to enable features for which fontspec does not yet have a user interface to.

专业用户可能希望根本不使用 fontspec 的特性处理，同时仍然利用其 LaTeX 字体选择便利。如果您记住了 OpenType 特性标签，则可以使用 `RawFeature` 字体特性使用文本特性选择字符串进行字体特性选择。更重要的是，这可以用于启用 fontspec 尚未具有用户界面的特性。

Multiple features can either be included in a single declaration: 多个特性可以包含在单个声明中：

```
[RawFeature=+smcp;+onum]
```

or with multiple declarations:

Example 40: Using raw font features directly.
直接使用原始字体特性。

```
\fontspec{texgyrepagella-regular.otf}[RawFeature=+smcp]
Pagella small caps
```

—Example graphic not generated—

或通过多个声明：

```
[RawFeature=+smcp, RawFeature=+onum]
```

Note that there is no error-checking when using `RawFeature`. Where a fontspec interface exists to a feature it is generally better to use it. If the font lacks the feature or if it would clash with another feature, fontspec will attemmpt to warn and/or resolve the issues.

请注意，在使用 `RawFeature` 时没有错误检查。如果存在到特性的 fontspec 接口，通常最好使用它。如果字体缺少特性或者会与另一个特性冲突，fontspec 将尝试警告和/或解决问题。

4 Renaming existing features & options
重命名现有特性和选项

`\aliasfontfeature`

If you don't like the name of a particular font feature, it may be aliased to another with the `\aliasfontfeature{`⟨*existing name*⟩`}{`⟨*new name*⟩`}` command, such as shown in Example 41.

如果您不喜欢特定字体功能的名称，则可以使用`\aliasfontfeature{`⟨*existing name*⟩`}{`⟨*new name*⟩`}` 命令将其重命名为其他功能，如 Example 41中所示。

Spaces in feature (and option names, see below) *are* allowed. (You may have noticed this already in the lists of OpenType scripts and languages).

功能（和选项名称，见下文）中的空格是允许的（您可能已经在 OpenType 脚本和语言列表中注意到了这一点）。

`\aliasfontfeatureoption`

If you wish to change the name of a font feature option, it can be aliased to another with the command `\aliasfontfeatureoption{`⟨*font feature*⟩`}{`⟨*existing name*⟩`}{`⟨*new name*⟩`}`, such as shown in Example 42.

如果要更改字体功能选项的名称，则可以使用命令`\aliasfontfeatureoption{`⟨*font feature*⟩`}{`⟨*existing name*⟩`}{`⟨*new name*⟩`}` 将其重命名为其他选项，如 Example 42中所示。

This example demonstrates an important point: when aliasing the feature options, the *original* feature name must be used when declaring to which feature the option belongs.

此示例演示了一个重要的观点：在别名功能选项时，必须使用原始功能名称来声明选项属于哪个功能。

Only feature options that exist as sets of fixed strings may be altered in this way. That is, `Proportional` can be aliased to `Prop` in the `Letters` feature, but `55␀␀99BB` cannot be substituted for `Purple` in a `Color` specification. For this type of thing, the `\newfontfeature`

Example 41: Renaming font features.
重命名字体功能。

—Example graphic not generated—

```
\aliasfontfeature{ItalicFeatures}{IF}
\fontspec{Hoefler Text}[IF = {Alternate=1}]
Roman Letters \itshape And Swash
```

Example 42: Renaming font feature options.
重命名字体功能选项。

```
\aliasfontfeature{VerticalPosition}{Vert Pos}
\aliasfontfeatureoption{VerticalPosition}{ScientificInferior}{Sci Inf}
\fontspec{LinLibertine_R.otf}[Vert Pos=Sci Inf]
 Scientific Inferior: 12345
```

—Example graphic not generated—

command should be used to declare a new, *e.g.*, PurpleColor feature:

只有存在一组固定字符串的功能选项才能以这种方式更改。也就是说，在 Letters 功能中，Proportional 可以被别名为 Prop，但在 Color 规范中，无法用 55QQ99BB 替换 Purple。对于这种类型的事情，应使用\newfontfeature 命令来声明一个新的功能，例如 PurpleColor：

```
\newfontfeature{PurpleColor}{color=55QQ99BB}
```

Except that this example was written before support for named colours was implemented. But you get the idea.

除了这个示例是在支持命名颜色之前编写的。但您可以了解到其中的思路。

5 Programming interface
编程接口

5.1 Variables
变量

\l_fontspec_family_tl
\l_fontspec_font

In some cases, it is useful to know what the LaTeX font family of a specific fontspec font is. After a \fontspec-like command, this is stored inside the \l_fontspec_family_tl macro. Otherwise, LaTeX's own \f@family macro can be useful here, too. The raw TeX font that is defined from the 'base' font in the family is stored in \l_fontspec_font.

在某些情况下，知道特定 fontspec 字体的 LaTeX 字体系列名称很有用。在\fontspec-like 命令之后，这将存储在\l_fontspec_family_tl 宏中。否则，LaTeX 自己的\f@family 宏在这里也很有用。定义从系列中的“基础”字体定义的原始 TeX 字体存储在\l_fontspec_font中。

\g_fontspec_encoding_tl

Package authors who need to load fonts with legacy LaTeX NFSS commands may also need to know what the default font encoding is. Since this has changed from EU1/EU2 to TU, it is best to use the variable \g_fontspec_encoding_tl instead.

需要使用传统的 LaTeX NFSS 命令加载字体的包作者，可能也需要知道默认的字体编码是什么。由于这已经从 EU1/EU2 更改为 TU，因此最好使用变量 \g_fontspec_encoding_tl。

5.2 Functions for loading new fonts and families
加载新字体和族的函数

\fontspec_gset_family:Nnn
\fontspec_set_family:Nnn

#1 : LaTeX family

`#2` : fontspec features

`#3` : font name

Defines a new NFSS family from given ⟨*features*⟩ and ⟨*font*⟩, and stores the family name in the variable ⟨*family*⟩. This font family can then be selected with standard LaTeX commands `\fontfamily{`⟨*family*⟩`}\selectfont`. See the standard fontspec user commands for applications of this function.

使用给定的⟨功能⟩和⟨字体⟩定义一个新的 NFSS 字族，并将字族名称存储在变量⟨*family*⟩中。然后可以使用标准的 LaTeX 命令 `\fontfamily{`⟨*family*⟩`}\selectfont` 选择这个字体族。有关此函数的应用，请参见标准 fontspec 用户命令。

(*End definition for* `\fontspec_gset_family:Nnn` *and* `\fontspec_set_family:Nnn`*. These functions are documented on page* **??**.)

`\fontspec_gset_fontface:NNnn`
`\fontspec_set_fontface:NNnn`

`#1` : primitive font

`#2` : LaTeX family

`#3` : fontspec features

`#4` : font name

Variant of the above in which the primitive TeX font command is stored in the variable ⟨*primitive font*⟩. If a family is loaded (with bold and italic shapes) the primitive font command will only select the regular face. This feature is designed for LaTeX programmers who need to perform subsequent font-related tests on the ⟨*primitive font*⟩.

与上述函数的变体，其中原始的 TeX 字体命令存储在变量⟨*primitive font*⟩中。如果加载了一个族（具有粗体和斜体形状），原始字体命令将仅选择常规字形。此功能旨在为需要执行后续与字体相关的测试的 LaTeX 程序员提供服务。

(*End definition for* `\fontspec_gset_fontface:NNnn` *and* `\fontspec_set_fontface:NNnn`*. These functions are documented on page* **??**.)

5.3 Conditionals
条件语句

The following functions in expl3 syntax may be used for writing code that interfaces with fontspec-loaded fonts. The following conditionals are all provided in `TF`, `T`, and `F` forms.

可以使用以下 expl3 语法的函数编写与加载 fontspec 字体进行交互的代码。所有以下条件语句都以 `TF`、`T` 和 `F` 形式提供。

5.3.1 Querying font families
查询字体族

`\fontspec_font_if_exist:nTF`

Test whether the 'font name' (`#1`) exists or is loadable. The syntax of `#1` is a restricted/simplified version of fontspec's usual font loading syntax; fonts to be loaded by filename are detected by the presence of an appropriate extension (`.otf`, etc.), and paths should be included inline. E.g.:

测试“字体名称”（#1）是否存在或可加载。#1 的语法是 fontspec 常规字体加载语法的一种受限/简化版本；将要通过文件名加载的字体由适当的扩展名（.otf 等）检测到，路径应内联包含。例如：

```
\fontspec_font_if_exist:nTF {cmr10}{T}{F}
\fontspec_font_if_exist:nTF {Times~ New~ Roman}{T}{F}
\fontspec_font_if_exist:nTF {texgyrepagella-regular.otf}{T}{F}
\fontspec_font_if_exist:nTF {/Users/will/Library/Fonts/CODE2000.TTF}{T}{F}
```

(*End definition for* `\fontspec_font_if_exist:nTF`. *This function is documented on page* **??**.)

The synonym `\IfFontExistsTF` is provided for 'document authors'.

“文档作者”提供了 `\IfFontExistsTF` 的同义词。

`\fontspec_if_fontspec_font:TF` Test whether the currently selected font has been loaded by fontspec.

测试当前选择的字体是否已被 fontspec 加载。

(*End definition for* `\fontspec_if_fontspec_font:TF`. *This function is documented on page* **??**.)

`\fontspec_if_opentype:TF` Test whether the currently selected font is an OpenType font. Always true for LuaTeX fonts.

测试当前选择的字体是否为 OpenType 字体。对于LuaTeX字体始终为真。

(*End definition for* `\fontspec_if_opentype:TF`. *This function is documented on page* **??**.)

`\fontspec_if_small_caps:TF` Test whether the currently selected font has a 'small caps' face to be selected with `\scshape` or similar. Note that testing whether the font has the `Letters=SmallCaps` font feature is sufficient but not necessary for this command to return true, since small caps can also be loaded from separate font files. The logic of this command is complicated by the fact that fontspec will merge shapes together (for italic small caps, etc.).

测试当前选择的字体是否具有“小型大写字母”字形，可用`\scshape` 或类似命令进行选择。请注意，测试字体是否具有`Letters=SmallCaps` 字体特征是充分但不必要的，因为小型大写字母也可以从单独的字体文件中加载。此命令的逻辑由于 fontspec 会将形状合并在一起（如斜体小型大写字母等）而变得复杂。

(*End definition for* `\fontspec_if_small_caps:TF`. *This function is documented on page* **??**.)

5.3.2 Availability of features
功能的可用性

`\fontspec_if_aat_feature:nnTF` Test whether the currently selected font contains the AAT feature (#1,#2).

测试当前选择的字体是否包含 AAT 功能（#1，#2）。

(*End definition for* `\fontspec_if_aat_feature:nnTF`. *This function is documented on page* **??**.)

`\fontspec_if_feature:nTF` Test whether the currently selected font contains the raw OpenType feature #1. E.g.:

测试当前选择的字体是否包含原始 OpenType 功能#1。例如：

`\fontspec_if_feature:nTF {pnum} {True} {False}`.

Returns false if the font is not loaded by fontspec or is not an OpenType font.

如果字体未由 fontspec 加载或不是 OpenType 字体，则返回 false。

(*End definition for* `\fontspec_if_feature:nTF`. *This function is documented on page* **??**.)

`\fontspec_if_feature:nnnTF` Test whether the currently selected font with raw OpenType script tag `#1` and raw OpenType language tag `#2` contains the raw OpenType feature tag `#3`. E.g.:

测试当前选择的字体，是否包含原始 OpenType 脚本标签`#1` 和原始 OpenType 语言标签`#2` 的原始 OpenType 功能标签`#3`。例如：

`\fontspec_if_feature:nnnTF {latn} {ROM} {pnum} {True} {False}`.

Returns false if the font is not loaded by fontspec or is not an OpenType font.

如果字体未由 fontspec 加载或不是 OpenType 字体，则返回 false。

(*End definition for* `\fontspec_if_feature:nnnTF`. *This function is documented on page* **??**.)

`\fontspec_if_script:nTF` Test whether the currently selected font contains the raw OpenType script `#1`. E.g.:

测试当前选择的字体是否包含原始 OpenType 脚本`#1`。例如：

`\fontspec_if_script:nTF {latn} {True} {False}`.

Returns false if the font is not loaded by fontspec or is not an OpenType font.

如果字体未由 fontspec 加载或不是 OpenType 字体，则返回 false。

(*End definition for* `\fontspec_if_script:nTF`. *This function is documented on page* **??**.)

`\fontspec_if_language:nTF` Test whether the currently selected font contains the raw OpenType language tag `#1`. E.g.:

测试当前选择的字体是否包含原始 OpenType 语言标签`#1`。例如：

`\fontspec_if_language:nTF {ROM} {True} {False}`.

Returns false if the font is not loaded by fontspec or is not an OpenType font.

如果字体未由 fontspec 加载或不是 OpenType 字体，则返回 false。

(*End definition for* `\fontspec_if_language:nTF`. *This function is documented on page* **??**.)

`\fontspec_if_language:nnTF` Test whether the currently selected font contains the raw OpenType language tag `#2` in script `#1`. E.g.:

测试当前选择的字体是否包含脚本`#1` 中的原始 OpenType 语言标签`#2`。例如：

`\fontspec_if_language:nnTF {cyrl} {SRB} {True} {False}`.

Returns false if the font is not loaded by fontspec or is not an OpenType font.

如果字体未由 fontspec 加载或不是 OpenType 字体，则返回 false。

(*End definition for* `\fontspec_if_language:nnTF`. *This function is documented on page* **??**.)

5.3.3 Currently selected features
当前选定的功能

\fontspec_if_current_feature:nTF

Test whether the currently loaded font is using the specified raw OpenType feature tag `#1`. The tag string `#1` should be prefixed with `+` to query an active feature, and with a `-` (hyphen) to query a disabled feature.

如果当前加载的字体使用了指定的原始 OpenType 功能标记`#1`，则进行测试。标记字符串`#1` 应以`+` 作为前缀来查询活动功能，并以`-`（连字符）作为前缀来查询禁用功能。

(*End definition for* `\fontspec_if_current_feature:nTF`*. This function is documented on page* **??**.)

\fontspec_if_current_script:nTF

Test whether the currently loaded font is using the specified raw OpenType script tag `#1`.

如果当前加载的字体使用了指定的原始 OpenType 脚本标记`#1`，则进行测试。

(*End definition for* `\fontspec_if_current_script:nTF`*. This function is documented on page* **??**.)

\fontspec_if_current_language:nTF

Test whether the currently loaded font is using the specified raw OpenType language tag `#1`.

如果当前加载的字体使用了指定的原始 OpenType 语言标记`#1`，则进行测试。

(*End definition for* `\fontspec_if_current_language:nTF`*. This function is documented on page* **??**.)