

LuaTeX-ja 宏包

LuaTeX-ja 项目团队

2012 年 5 月 5 日

目次

第 I 編 用户手册	3
1 引言	3
2 背景	3
2.1 与 p _T E _X 的差异所在	3
2.2 一些约定	4
2.3 关于本项目	4
3 使用	5
3.1 安装	5
3.2 注意	5
3.3 plain T _E X 下使用	5
3.4 L ^A T _E X 下使用	6
3.5 字体更改	7
3.6 fontspec	8
4 变量更改	8
4.1 J _A char 范围设定	8
4.2 kanjiskip 和 xkanjiskip	10
4.3 xkanjiskip 插入设定	11
4.4 基线浮动	11
4.5 裁剪框标记	12
第 II 編 参考指南	12
4.6 \jfont 基本语句	12
4.7 psft 前缀	13
4.8 JFM 结构	13
4.9 luatexja-fontspec.sty	16
4.10 luatexja-otf.sty	16
4.11 段落/水平ボックスの先頭や末尾	20
4.12 概観と典型例: 2つの「和文 A」の場合	21
4.13 その他の場合	23
5 psft	26

本文档尚未完成。

第 I 编

用户手册

1 引言

LuaTeX-j_a 宏包是应用于下一代标准 TeX 引擎亦即 LuaTeX 引擎上的高质量日语文档排版宏包。

2 背景

一般情况下，TeX 下的日语文档输出，是 ASCII pTeX (TeX 的一个扩展) 及其衍生软件来完成的。pTeX 作为 TeX 的一个扩展引擎，在生成高质量的日语文档时，规避了繁杂的宏编写。但是在和同时期的引擎相比之下，pTeX 的处境未免有些尴尬：pTeX 已经远远落后于 ε-TeX 和 pdfTeX，此外也没有跟上计算机上对日文处理的演进（比如，UTF-8 编码，TrueType 字体，OpenType 字体）。

最近开发的 pTeX 扩展，即 upTeX (Unicode 下的 pTeX 实现) 和 ε-pTeX (pTeX 和 ε-TeX 的融合版本)，虽然在部分情况上弥补了上述的差距，但是差距依然存在。

不过，LuaTeX 的出现改变了整个状况。用户可以通过使用 Lua 语言的“callback”来调整 LuaTeX 的内部处理机制。所以，没有必要去通过修改引擎的源代码来支持日文排版，相反，我们需要做的仅仅是编写其当处理 callback 的 Lua 脚本。

2.1 与 pTeX 的差异所在

LuaTeX-j_a 宏包在设计上，受 pTeX 影响很大。最初开发的主要议题是实现 pTeX 的特性。不过，LuaTeX-j_a 不是简简单单的移植 pTeX，很多不自然的特征和现象都被移出了。

下面列举出了一些和 pTeX 的差异：

- 一个日文字体是由三部分构成的元组：实际的字体（如小塚明朝，IPA 明朝），日文字体测度 (JFM) 和变体字串。
- pTeX 中，日文字符之后的断行并不允许（也不产生空格），其他在源码中的断行是可以随处允许的。不过，因为 LuaTeX 的特殊关系，LuaTeX-j_a 并没有这个功能。
- 插在日文字符和其他字符之间的胶/出格（我们将此称为 **JAgglue**）是重新实现的。
 - 在 LuaTeX 中，内部的字符处理是“基于 node 的”（例如：`of{}fice` 不会避免合字），**JAgglue** 的插入处理，现在也是“基于 node 的”。
 - 此外，两个字符之间的 node 在断行时不起作用的（例如，`\specialnode`），还有意大利体校正带来的出格在插入处理中也是被忽略的。
 - **警告：**鉴于以上两点，在 pTeX 中分割 **JAgglue** 处理的多种方法不再生效。明确地说，下列两种方法不再生效：

```
    ちょ{}っと    ちょ\Vっと
```

如果想得到此种结果，请使用空盒子替代：

```
    ちょ\hbox{}っと
```
 - 处理过程中，两个在“真实”字体上具区别的日文字体可以被识别出来。
- 当下，LuaTeX-j_a 并不支持直行排版。

详细的描述，请参见第 4.10 编。

2.2 一些约定

在本文档中，有下面一些约定：

- 字符被分为两种类型：
 - **JAchar**：表示日文字符，如平假名，片假名，汉字，日文标点。
 - **ALchar**：代表其他字母字符。我们将用于 **ALchar** 的字体称为“字母字体”，用于 **JAchar** 的字体称为“日文字体”。
- 用无衬线字体表示的词（如：`prebreakpenalty`）表示日文排版中的内部便利 `iang`，并用做 `\ltjsetparameter` 命令一个键。
- 用下划线表示的词（如：`fontspec`）表示 L^AT_EX 的宏包或者文档类。
- “primitive”，该词在本文档中不仅表示 Lua_TE_X 的基本控制命令，也包括 Lua_TE_X-ja 的相关的基本控制命令
- 所有的自然数从 0 开始

2.3 关于本项目

■项目 wiki 本项目 wiki 正在编写中。

- <http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage> (日语)
- <http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage%28en%29> (英语)
- <http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage%28zh%29> (汉语)

本项目由 SourceForge.JP 托管。

■开发者

- 北川 弘典
- 黒木 裕介
- 本田 知亮
- 前田 一贵
- 阿部 纪行
- 斋藤 修三郎
- 八登 崇之
- 山本 宗宏
- 马 起园

3 使用

3.1 安装

安装 LuaTeX-ja 之前，需要如下：

- LuaTeX (版本号为大于 0.65) 和相关支持宏包。
如果用户使用的是 T_EXLive2011 以及最新版本的 W32T_EX，可不考虑此项。
- LuaTeX-ja 的源码 (当然喽:)。 xunicode 宏包，当前版本必须为 *v0.981(2011/09/09)*。
如果你使用 fontspec 宏包，xunicode 就必须存在。但是请注意该包的版本，其他版本可能不会正常工作。

安装方法如下：

1. 按照如下方法下载源码归档。现在，LuaTeX-ja 没有稳定版本。

- 复制 Git 仓库：

```
$ git clone git://git.sourceforge.jp/gitroot/luatex-ja/luatexja.git
```

- 下载 master HEAD 版本的 tar.gz 归档：

```
http://git.sourceforge.jp/view?p=luatex-ja/luatexja.git;a=snapshot;h=HEAD;sf=tgz.
```

- 现在 LuaTeX-ja 可以在下列仓库和发行版中获取：：

- CTAN (`macros/luatex/generic/luatexja`)
- MiKTeX (`luatexja.tar.lzma`)
- T_EX Live (`texmf-dist/tex/luatex/luatexja`)
- W32T_EX (`luatexja.tar.xz`)

这些版本都基于 master 分支。

注意 master 分支和 CTAN 仓库中的版本，升级并不频繁。前段开发并未在 master 分支。

2. 解压归档。你会看到 `src/` 和其他相关文件夹。但是只有 `src/` 文件夹下的相关文件是 LuaTeX-ja 运行所必须的。
3. 复制 `src/` 文件夹下内容至 TEXMF 数下。TEXMF/`tex/luatex/luatexja/` 为例。如果你复制了整个 Git 仓库，为 `src/` 制作软链接来替代复制也是可以的。
4. 如有必要，执行 `mktexlsr`。

3.2 注意

- 源文档编码必须是 UTF-8。其他的编码，如 EUC-JP 和 Shift-JIS 都不被支持。

3.3 plain T_EX 下使用

在 plain T_EX 下使用 LuaTeX-ja 相当简易，在文档开头放置一行：

```
\input luatexja.sty
```

这里做出了做小的日文文档排版设定 (如 `ptex.tex`)：

- 提前加载了六种日文字体，如下：

字体	字体名	‘10 pt’	‘7 pt’	‘5 pt’
明朝体	Ryumin-Light	<code>\tenmin</code>	<code>\sevenmin</code>	<code>\fivemin</code>
哥特体	GothicBBB-Medium	<code>\tengt</code>	<code>\sevengt</code>	<code>\fivegt</code>

- “Q (级)” 是日本照排中使用的尺寸单位, $1Q = 0.25\text{mm}$ 。该长度保存在长度`\jq`中。
- 广为接受的“Ryumin-Light”和“GothicBBB-Medium”字体不嵌入 PDF 文件, 而 PDF 阅读器则会使用外部日文字体替代 (例如, 在 Adobe Reader 中使用 Kozuka Mincho 字体替代 Ryumin-Light)。我们使用默认设定。
- 一般情况下, 相同大小日文字体比西文字体要大一下。所以实际的日文字体尺寸需哟小于西文字体, 即使用一个缩放率: 0.962216。
- 在 **J**Achar 和 **AL**char 之间插入的胶 (`xkanjiskip` 参数) 大小为:

$$(0.25 \cdot 0.962216 \cdot 10\text{pt})_{-1\text{pt}}^{+1\text{pt}} = 2.40554\text{pt}_{-1\text{pt}}^{+1\text{pt}}$$

3.4 L^AT_EX 下使用

■L^AT_EX 2_ε 在 L^AT_EX 2_ε 下使用基本相同。设定日文的最小环境, 你只需加载 `luatexja.sty`:

```
\usepackage{luatexja}
```

这些做了最小的设定 (作用相当于 pL^AT_EX 中的 `plfonts.dtx` 和 `pldefs.ltx`):

- JY3 是日文字体编码 (在水平方向)。
在将来 LuaT_EX-ja 要支持直行排版的时候, JT3 会用于直行字体。
- 定义了两个字体族: `mc` 和 `gt`:

字体	字体族	<code>\mdseries</code>	<code>\bfseries</code>	缩放率
<i>mincho</i>	<code>mc</code>	Ryumin-Light	GothicBBB-Medium	0.962216
<i>gothic</i>	<code>gt</code>	GothicBBB-Medium	GothicBBB-Medium	0.962216

注意的是两个字体族的粗体系列同为中等系列的哥特族。这 pL^AT_EX 中的规定。在近些年中的 DTP 实务中仅有使用 2 个字体的趋向 (是为 Ryumin-Light 和 GothicBBB-Medium)。

- 在数学模式下, 日文字符使用 `mc` 字体族来排印。

不过, 上述设定并不能满足排版基于日文的文档。为了排印基于日文的文档, 你最好不要使用 `article.cls`, `book.cls` 等文档类文件。现在, 我们有相当于 `jclasses` (pL^AT_EX 标准文档类) 和 `jsclasses` (奥村晴彦) 的文档类, 即 `ltjclasses` 和 `ltjsclasses`。

■OTF 包中的 `\CID`, `\UTF` 及其他宏 pL^AT_EX 下, `otf` 宏包 (斋藤修三郎开发) 是用来排印存在于 Adobe-Japan1-6 但不存在于 JIS X 0208 中的字符。该包已经广泛使用, LuaT_EX-ja 支持部分 `otf` 包中的部分功能。如果你想使用这些功能, 加载 `luatexja-otf` 宏包。

1 森`\UTF{9DD7}`外と内田百`\UTF{9592}`とが`\UTF{9`

`AD9}`島屋に行く。

森鷗外と内田百間とが高島屋に行く。

2

3 `\CID{7652}`飾区の`\CID{13706}`野家,

葛飾区の吉野家, 葛飾区の吉野家

4 葛飾区の吉野家

3.5 字体更改

■注记：数学模式下的日文字符 pTeX 支持在数学模式下的日文字符，如以下源码：

```

1 $f_{高温}$~($f_{\text{high temperature}}$)$.          f_{高温} (f_{high temperature}).
   }$).
2 [\ y=(x-1)^2+2\quad よって\quad y>0 \]              y = (x-1)^2 + 2   よって   y > 0
3 $5\in \text{素} := \{p \in \mathbb{N} : \text{素} p \text{ is a}   5 \in \text{素} := \{p \in \mathbb{N} : p \text{ is a prime}\}.
   \text{prime}\}, \}$$.

```

我们 (LuaTeX-ja 项目成员) 认为在数学模式下使用日文字符，只有在这些字符充当标识符时才是正确的。在这点上：

- 第 1 行和第 2 行是不正确的，因为“高温”的作用为文本标签，“よって”用作为连词。
- 不过，第 3 行是正确的，因为“素”是作为标识符的。

那么，根据我们的观点，上述输入应当校正为：

```

1 $f_{\text{高温}}$~%
2 ($f_{\text{high temperature}}$)$.          f_{高温} (f_{high temperature}).
3 [\ y=(x-1)^2+2\quad
4 \mathrel{\text{よって}}\quad y>0 \]          y = (x-1)^2 + 2   よって   y > 0
5 $5\in \text{素} := \{p \in \mathbb{N} : \text{素} p \text{ is a}   5 \in \text{素} := \{p \in \mathbb{N} : p \text{ is a prime}\}.
   \text{prime}\}, \}$$.

```

我们也认为使用日文字符作为标识符的情况极为少见，所以我们不在此章节描述如何在数学模式下改变日文字体。关于此方法，请参见4.8。

■plain TeX 在 plain TeX 下改变日文字体，你必须使用基本语句\jfont。请参见4.6。

■NFSS2 对于 L^AT_EX 2_ε，LuaTeX-ja 采用了 pL^AT_EX 2_ε 中（即 plfonts.dtx）大部分字体选择系统。

- \mcdefault 和\gtdefault 控制语句用来分别控制默认的 mincho 和 gothic 字体族。默认值：mc 用于\mcdefault，gt 用于\gtdefault。
- 命令\fontfamily, \fontseries, \fontshape 个\selectfont 用来改变日文字体属性。

	编码	族	系列	形状	选择
西文字体	\romanencoding	\romanfamily	\romanseries	\romanshape	\useroman
日文字体	\kanjiencoding	\kanjifamily	\kanjiseris	\kanjishape	\usekanji
两者	—	—	\fontseries	\fontshape	—
自动选择	\fontencoding	\fontfamily	—	—	\usefont

\fontencoding{<encoding>}依赖于参数以改变西文字体或者日文字体。例如，\fontencoding{JY3} 改变当前日文字体至 JY3，\fontencoding{T1} 改变西文字体至 T1。fontfamily 也会改变日文字体或西文字体的族，抑或二者。细节详见4.8。

- 对于定义日文字体族，使用\DeclareKanjiFamily 代替\DeclareFontFamily。不过，在现在的实现中，使用\DeclareFontFamily 不会引起任何问题。

3.6 fontspec

为与 `fontspec` 宏包共存，需要在导言区中使用 `luatexja-fontspec` 宏包。这个附加宏包会自动加载 `luatexja` 和 `fontspec`。

在 `luatexja-fontspec` 中，定义了如下七条命令，这些命令和 `fontspec` 的相关命令对比如下：

日文字体	<code>\jfontspec</code>	<code>\setmainjfont</code>	<code>\setsansjfont</code>	<code>\newfontfamily</code>
西文字体	<code>\fontspec</code>	<code>\setmainfont</code>	<code>\setsansfont</code>	<code>\newfontfamily</code>
日文字体	<code>\newfontface</code>	<code>\defaultjfontfeatures</code>	<code>\addfontfeatures</code>	
西文字体	<code>\newfontface</code>	<code>\defaultfontfeatures</code>	<code>\addfontfeatures</code>	

```
1 \fontspec[Numbers=OldStyle]{TeX Gyre
   Termes}
2 \jfontspec{IPAexMincho}
3 JIS-X-0213:2004→辻
4 JIS X 0208:1990→辻
5 \addfontfeatures{CJKShape=JIS1990}
6 JIS-X-0208:1990→辻
```

请注意并没有 `\setmonofont` 命令，因为流行的日文字体几乎全部是等宽的。另注意，出格特性在这 7 个命令中默认关闭，因为此特性会与 `JAgglue` 冲突（参见 4.6）。

4 变量更改

LuaTeX-ja 包含大量的参数，以控制排版细节。设定这些参数需要使用命令：`\ltjsetparameter` 和 `\ltjgetparameter` 命令。

4.1 JAchar 范围设定

在设定 `JAchar` 之前，需要分配一个小于 217 的自然数。这个可以由 `\ltjdefcharrange` 基本语句来完成。例如，下面就分配了整个表意文字补充平面和汉字“漢”为 100。

```
\ltjdefcharrange{100}{"10000-"1FFFF,`漢}
```

范围数的分配是全局的，故你不可在文档中使用。

如果某些字符被改变为新的非零数范围，将会被新设定重写。例如，整个 SIP 在 LuaTeX-ja 默认设定中属于范围 4，如果你使用如上设定，SIP 将会被设定为范围 100，且从范围 4 种移除。

分配了范围数之后，`jacharrange` 参数将用于设定字符范围为 `JAchar`，如下（为 LuaTeX-ja 默认设定）：

```
\ltjsetparameter{jacharrange={-1, +2, +3, -4, -5, +6, +7, +8}}
```

`jacharrange` 参数的变量未整数数组。负数 $-n$ 在数组中表示“字符范围 n 中的字符被视作 `ALchar`”，正数 $+n$ 则表示“字符范围 n 被视作 `JAchar`”。

■默认设定 LuaTeX-ja 默认设定了 8 个字符范围。如下设定：

- Unicode 6.0 区块

表 1. 字符范围 3 定义的 Unicode 范围

U+2000–U+206F	一般标点符号	U+2070–U+209F	上标及下标
U+20A0–U+20CF	货币符号	U+20D0–U+20FF	符号用组合附加符号
U+2100–U+214F	类字母符号	U+2150–U+218F	数字形式
U+2190–U+21FF	箭头符号	U+2200–U+22FF	数学运算符号
U+2300–U+23FF	杂项技术符号	U+2400–U+243F	控制图像
U+2500–U+257F	制表符	U+2580–U+259F	区块元素
U+25A0–U+25FF	几何形状	U+2600–U+26FF	杂项符号
U+2700–U+27BF	什锦符号	U+2900–U+297F	补充性箭头-B
U+2980–U+29FF	混合数学符号-B	U+2B00–U+2BFF	杂项符号和箭头符号
U+E000–U+F8FF	私有区域		

- 在 CID Adobe-Japan1-6 和 Unicode 之间的映射 Adobe-Japan1-UCS2。
- 八登崇之的 P_Xbase 宏包 (up_TE_X 下使用)。

现在我们描述 8 个字符范围。在数字后的“J”和“A”表明代表 **J**Achar 或者未跟随默认设定。这些设定类似于 P_Xbase 中的 `prefercjk` 设定。

范围 8^J ISO 8859-1 (Latin-1 补充) 的上半部和 JIS X 0208 (日文基本字符集) 的重叠部分, 包含下列字符:

- § (U+00A7, 分节符)
- ¨ (U+00A8, 分音符)
- ° (U+00B0, 温度符号)
- ± (U+00B1, 加减符号)
- (U+00B4, 置位尖音)
- (U+00B6, 段落符号)
- × (U+00D7, 乘号)
- ÷ (U+00F7, 除号)

范围 1^A 包含于 Adobe-Japan1-6 中的拉丁字符, 此范围包含下列 Unicode 区域, 但不包括上述提到过的范围 8:

- U+0080–U+00FF: 拉丁字母补充-1
- U+0100–U+017F: 拉丁字母扩充-A
- U+0180–U+024F: 拉丁字母扩充-B
- U+0250–U+02AF: 国际音标扩充
- U+02B0–U+02FF: 进格修饰符元
- U+0300–U+036F: 组合音标附加符号
- U+1E00–U+1EFF: 拉丁字母扩充附加

范围 2^J 希腊文和西里尔字母, 使用 JIS X 0208 的大部分日文字体包含这些字符:

- U+0370–U+03FF: 希腊字母
- U+0400–U+04FF: 西里尔字母
- U+1F00–U+1FFF: 希腊文扩充

范围 3^J 标点以及杂项符号, 参见表1。

范围 4^A 通常情况下不包含于日文字体的部分。本范围包含有其他范围尚未涵盖部分。故, 我们直接给出定义:

```
\ltjdefcharrange{4}{%
    "500-"10FF, "1200-"1DFF, "2440-"245F, "27C0-"28FF, "2A00-"2AFF,
    "2C00-"2E7F, "4DC0-"4DFF, "A4D0-"A82F, "A840-"ABFF, "FB50-"FE0F,
    "FE20-"FE2F, "FE70-"FEFF, "FB00-"FB4F, "10000-"1FFFF} % non-Japanese
```

范围 5^A 代替以及补充私有使用区域。

表 2. 字符范围 6 定义的 Unicode 范围

U+2460–U+24FF	圈状字母数字	U+2E80–U+2EFF	CJK 部首补充
U+3000–U+303F	CJK 标点符号	U+3040–U+309F	平假名
U+30A0–U+30FF	片假名	U+3190–U+319F	汉文标注号
U+31F0–U+31FF	片假名音标补充	U+3200–U+32FF	圈状 CJK 字母及月份
U+3300–U+33FF	CJK 兼容	U+3400–U+4DBF	CJK 统一表意文字扩充 A
U+4E00–U+9FFF	CJK 统一表意文字	U+F900–U+FAFF	CJK 兼容表意文字
U+FE10–U+FE1F	直行标点	U+FE30–U+FE4F	CJK 兼容形式
U+FE50–U+FE6F	小写变体	U+20000–U+2FFFF	(补充字符)

表 3. 字符范围 7 定义的 Unicode 范围

U+1100–U+11FF	谚文字母	U+2F00–U+2FDF	康熙字典部首
U+2FF0–U+2FFF	汉字结构描述字符	U+3100–U+312F	注音字母
U+3130–U+318F	谚文兼容字母	U+31A0–U+31BF	注音字母扩充
U+31C0–U+31EF	CJK 笔划	U+A000–U+A48F	彝文音节
U+A490–U+A4CF	彝文字母	U+A830–U+A83F	一般印度数字
U+AC00–U+D7AF	谚文音节	U+D7B0–U+D7FF	谚文字母扩充-B

范围 6^J 日文字符。

范围 7^J 不包含于 Adobe-Japan1-6 的 CJK 字符，参见表 3。

4.2 kanjiskip 和 xkanjiskip

JAglue 分为以下三个范畴：

- JFM 设定的胶或出格值。如果在一个日文字符附近使用 `\inhibitglue`，则胶便不会插入。
- 两个 **JA**char 之间默认插入的胶 (`kanjiskip`)。
- **JA**char 和 **AL**char 之间默认插入的胶 (`xkanjiskip`)。

`kanjiskip` 和 `xkanjiskip` 的设定如下所示：

```
\ltjsetparameter{kanjiskip={0pt plus 0.4pt minus 0.4pt},
                 xkanjiskip={0.25\zw plus 1pt minus 1pt}}
```

当 JFM 包含“`kanjiskip` 理想宽度”和/或“`xkanjiskip` 理想宽度”数据时，上述设定产生作用。如果想用 JFM 中的数据，请设定 `kanjiskip` 或 `xkanjiskip` 为 `\maxdimen`。

4.3 xkanjiskip 插入设定

并不是在所有的 **JA**char 和 **AL**char 周围插入 `xkanjiskip` 都是合适的。比如，在开标点之后插入 `xkanjiskip` 并不合适 [如，比较“(あ”和“(あ”]。Lua_TE_X-ja 可以通过设定 **JA**char 的 `jaxspmode` 以及 **AL**char 的 `alxspmode` 来控制 `xkanjiskip` 在字符前后的插入。

```
1\ltjsetparameter{jaxspmode={`あ,preonly},
                 alxspmode={`\!,postonly}}
2pあ q い!う
```

第二个参数 `preonly` 表示的含义为“允许在该字符前插入 `xkanjiskip`，但不允许在该字符之后插入”。其他参数还有 `postonly`，`allow` 和 `inhibit`。

当前版本的 `jaxspmode` 和 `alxspmode` 使用相同的的表保存参数。因此，上一行可被写作：

```
\ltjsetparameter{alxspmode={`あ,preonly}, jaxspmode={`\!,postonly}}
```

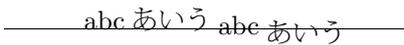
你也可以使用数字来定义两个参数（参见4.8）。

如果你想要启用/屏蔽所有的 `kanjiskip` 和 `xkanjiskip` 插入，设定 `autospacing` 和 `autoxspcng` 为 `ture/false` 即可。

4.4 基线浮动

为了确保日文字体和西文字体能够对其，有时需要浮动其中一者的基线。在 `pTeX` 中，此项设定由设定 `\yabaseshift` 为非零长度（西文字体基线应向下浮动）。不过，如果文档的中主要语言不是日文，那么最好上浮日文字体的基线，西文字体不变。如上所述，`LuaTeX-ja` 可以独立设定西文字体的基线（`yabaseshift` 参数）和日文字体的基线（`yjabaseshift` 参数）。

```
1 \vrule width 150pt height 0.4pt depth 0pt\
   hskip-120pt
2 \ltjsetparameter{yjabaseshift=0pt,
   yabaseshift=0pt}abcあいう
3 \ltjsetparameter{yjabaseshift=5pt,
   yabaseshift=2pt}abcあいう
```



上述水平线为此行基线。

这里还有一个有趣的副作用：不同大小的字符可以通过适当调整这两个参数而在一行中垂直居中。下面是一个例子（注意，参数值并没有刻意调整）：

```
1 xyz漢字
2 {\scriptsize
3 \ltjsetparameter{yjabaseshift=-1pt,
4 yabaseshift=-1pt} xyz 漢字 XYZ ひらがな abc かな
5 XYZひらがな
6 }abcかな
```

4.5 裁剪框标记

裁剪框标记是在一页的四角和水平/垂直中央放置的标记。在日文中，裁剪框被称为“トンボ”。`pLaTeX` 和 `LuaTeX-ja` 均在底层支持裁剪框标记。需要下列步骤来实现：

1. 首先，首先定义页面左上角将会出现的注记。这由向 `@bannertoken` 分配一个 `token` 列完成。例如，下列所示将会设定注记为“filename (YYYY-MM-DD hh:mm)”：

```
\makeatletter

\hour\time \divide\hour by 60 \@tempcnta\hour \multiply\@tempcnta 60\relax
\minute\time \advance\minute-\@tempcnta
\@bannertoken{%
```

```
\jobname\space(\number\year-\two@digits\month-\two@digits\day
\space\two@digits\hour:\two@digits\minute)}%
```

2. ...

第 II 编

参考指南

4.6 \jfont 基本语句

为了加载日文字体，需要使用 `\jfont` 基本语句替代 `\font`，前者支持后者所有相同句法。LuaTeX-ja 自动加载 `luaotfload` 宏包，故 TrueType/OpenType 字体的特性可以使用于日文字体：

```
1 \jfont\tradgt={file:ipaexg.ttf:script=latn
   ;%
2 +trad;-kern;jfm=ujis} at 14pt
3 \tradgt{}当/体/医/区
```

當 / 體 / 醫 / 區

注意定义的控制序列（上例中的 `\tradgt`）使用的 `\jfont` 并不是一个 `font_def` 标记，故类似 `\fontname\tradgt` 输入会引起错误。我们将定义 `\jfont` 采用 `\jfont_cs`。

■JFM 在引言中已提及此项，所谓 JFM 是字符亮度和日文排版中自动插入的胶/出格。JFM 的结构将在下节进行描述。在使用 `\jfont` 基本语句时，必须设定 JFM 如下两个键：

`jfm=<name>` 设定 JFM 名称。设定的 JFM 如未加载，LuaTeX-ja 会搜寻并加载一个命名为 `jfm-<name>.lua` 的文件。

LuaTeX-ja 提供如下 JFM：

`jfm-ujis.lua` LuaTeX-ja 标准 JFM。次 JFM 基于 upTeX 使用的 UTF/OTF 宏包的 `upnmlminr-h.tfm`。如果你使用 `luatexja-otf` 宏包，你将会用到此 JFM。

`jfm-jis.lua` 相当于 pTeX 使用的 `jis.tfm`（“JIS font metric”）。`jfm-ujis.lua` 和 `jfm-jis.lua` 主要区别是：`jfm-ujis.lua` 下的大部分字符是方形，`jfm-jis.lua` 下则为长方形。

`jfm-min.lua` 相当于 pTeX 中默认的 `min10.tfm`。这个 JFM 与其他 2 个 JFM 的区别如表 4 所示。

`jfmvar=<string>` Sometimes there is a need that ...

■注意：kern feature 一些字体具有内部字形间距信息。但是，这些信息在 LuaTeX-ja 下并不良好兼容。仔细些说，出格间距是在 **JAgLue** 插入之前而先行插入的，这就造成了字体数据中和 JFM 中的胶/出格在两个字符间插入出错。

- 当你想使用其他字体特性如 `script=...` 的时候，可以在 `jfont` 基本语句中设置 `-kern`
- 如果你想使用比例宽度的日文字体，并且使用此字体信息，使用 `jfm-prop.lua` 为其 JFM，……
TODO: kanjiskip?

*1 from: 乙部巖己, min10 フォントについて. <http://argent.shinshu-u.ac.jp/~otobe/tex/files/min10.pdf>.

表 4. LuaTeX-ja 下不同 JFM 表现

	jfm-ujis.lua	jfm-jis.lua	jfm-min.lua
例 1*1	ある日モモちゃんがお使いで迷子になって泣きました。	ある日モモちゃんがお使いで迷子になって泣きました。	ある日モモちゃんがお使いで迷子になって泣きました。
例 2	ちょっと！何	ちょっと！何	ちょっと！何
Bounding Box			

4.7 psft 前缀

除使用 `file:` 和 `name:` 外, 我们还可以在 `\jfont` (以及 `\font`) 中使用 `psft:` 来设定一个“名义上”的并不嵌入 PDF 中的日文字体。此前缀的典型使用是定义“标准”日文字体, 即“`Ryumin-Light`”和“`GothicBBB-Medium`”。

■`cid` 键 默认使用 `psft:` 前缀定义的字体是为 Adobe-Japan1-6 CID 字体。也可以使用 `cid` 键来使用其他的 CID 字体, 如中文和韩文。

■`cid key` `cid key, ...`

4.8 JFM 结构

JFM 文件为下列函数调用的 Lua 脚本:

```
luatexja.jfont.define_jfm { ... }
```

实际的数据保存在表中, 即如上的 `{ ... }`。以下部分描述表结构。请注意, 在 JFM 中的所有长度都是按照以 `design-size` 为单位的浮点数。

`dir`=*(direction)* (必须)

JFM 的方向, 现在只支持 `yoko` (水平)。

`zw`=*(length)* (必须)

“全角”长度。

`zh`=*(length)* (必须)

“全角高度” (`height + depth`) 长度。

`kanjiskip`={*(natural)*, *(stretch)*, *(shrink)*} (可选)

这部分为“理想长度” `kanjiskip`。4.2 节有详述, 如果参数 `kanjiskip` 为 `\maxdimen`, 则值设定将会被使用 (若再 JFM 中未设定, 则被视为 0 pt)。请注意, *(stretch)* 和 *(shrink)* 的长度均为 `design-size` 单位。

`xkanjiskip`={*(natural)*, *(stretch)*, *(shrink)*} (可选)

和 `kanjiskip` 类似, 此部分设定 `xkanjiskip` 的“理想长度”。

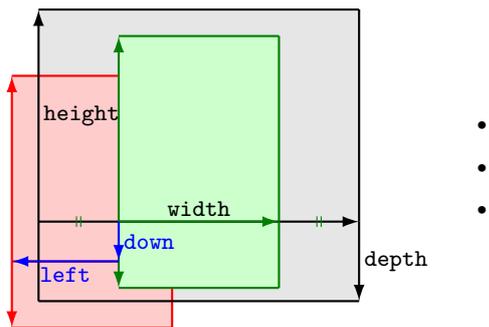


图 1.

font family	<code>\jfam ∈ [0,256)</code>	<code>\fam</code>
text size	<code>jatextfont = {⟨jfam⟩, ⟨jfont_cs⟩}</code>	<code>\textfont⟨fam⟩ = ⟨font_cs⟩</code>
script size	<code>jascriptfont = {⟨jfam⟩, ⟨jfont_cs⟩}</code>	<code>\scriptfont⟨fam⟩ = ⟨font_cs⟩</code>
scriptscript size	<code>jascriptscriptfont = {⟨jfam⟩, ⟨jfont_cs⟩}</code>	<code>\scriptscriptfont⟨fam⟩ = ⟨font_cs⟩</code>

除了上面涉及到的内容，JFM 文件中还有几个以自然数进行声明的次级表。这些表依靠 $i \in \omega$

```

chars={⟨character⟩, ...}
width=⟨length⟩, height=⟨length⟩, depth=⟨length⟩, italic=⟨length⟩
left=⟨length⟩, down=⟨length⟩, align=⟨align⟩
kern=[⟨j⟩=⟨kern⟩, ...}
glue=[⟨j⟩={⟨width⟩, ⟨stretch⟩, ⟨shrink⟩}, ...}

'lineend'
'diffmet'
'boxbdd'
'parbdd'
'jcharbdd'
-1

1 function (⟨table⟩ jfm_info, ⟨string⟩ jfm_name)
2   return ⟨table⟩ new_jfm_info
3 end

1 function (⟨table⟩ jfont_info, ⟨number⟩ font_number)
2   return ⟨table⟩ new_jfont_info
3 end

jfm
size
var

1 function (⟨number⟩ char_class, ⟨table⟩ jfont_info, ⟨number⟩ chr_code)
2   if char_class~=0 then return char_class
3   else
4     ....
5   return (⟨number⟩ new_char_class or 0)

```

```

6   end
7   end

1  function (<table> shift_info, <table> jfont_info, <number> char_class)
2    return <table> new_shift_info
3  end

```

```

1 \ltjgetparameter{differentjfm},
2 \ltjgetparameter{autospacing},           average, 1, 10000.
3 \ltjgetparameter{prebreakpenalty}{`)}.

```

```

jcharwidowpenalty=<penalty> [\jcharwidowpenalty]
kcatcode={<chr_code>,<natural number>}
prebreakpenalty={<chr_code>,<penalty>} [\prebreakpenalty]
postbreakpenalty={<chr_code>,<penalty>} [\postbreakpenalty]
jatextfont={<jfam>,<jfont_cs>}
jascriptfont={<jfam>,<jfont_cs>}
jascriptscriptfont={<jfam>,<jfont_cs>}
yjabaselineshift=<dimen>*
yalbaselineshift=<dimen>* [\ybaselineshift]
jaxspmode={<chr_code>,<mode>}
alxspmode={<chr_code>,<mode>} [\xspcode]
autospacing=<bool>* [\autospacing]
autoxspacing=<bool>* [\autoxspacing]
kanjiskip=<skip> [\kanjiskip]
xkanjiskip=<skip> [\xkanjiskip]
differentjfm=<mode>† average
    both
    large
    small
jacharrange=<ranges>*
kansujichar={<digit>,<chr_code>} [\kansujichar]

```

```

\kuten
\jis
\euc
\sjis
\ucsc
\kansuji

```

```

1 \jfont\g=psft:Ryumin-Light:jfm=test \g
2 \fbox{\hbox{あうあ\inhibitglue う}}
3 \inhibitglue\par\noindent あ1
4 \par\inhibitglue\noindent あ2
5 \par\noindent\inhibitglue あ3
6 \par\hrule\noindent あ off\inhibitglue ice

```

あ	うあう
---	-----

あ 1
あ 2
あ 3

あ office

```

\DeclareYokoKanjiEncoding{<encoding>}{<text-settings>}{<math-settings>}
\DeclareKanjiEncodingDefaults{<text-settings>}{<math-settings>}
\DeclareKanjiSubstitution{<encoding>}{<family>}{<series>}{<shape>}
\DeclareErrorKanjiFont{<encoding>}{<family>}{<series>}{<shape>}{<size>}
\reDeclareMathAlphabet{<unified-cmd>}{<al-cmd>}{<ja-cmd>}
\DeclareRelationFont{<ja-encoding>}{<ja-family>}{<ja-series>}{<ja-shape>}
  {<al-encoding>}{<al-family>}{<al-series>}{<al-shape>}
\SetRelationFont
\userelfont
\adjustbaseline
...
\fontfamily{<family>}
1 \gtfamily{ } あいう abc
2 \SetRelationFont{JY3}{gt}{m}{n}{OT1}{pag}{
  m}{n} あいう abc あいう abc
3 \userelfont\selectfont{ } あいう abc

```

4.9 luatexja-fontspec.sty

```

JFM=<name>
JFM-var=<name>
NoEmbed
CID=<name>

```

4.10 luatexja-otf.sty

```

\CID{<number>}
\UTF{<hex_number>}

```

```

\jQ (dimension)
\jH (dimension)
\ltj@zw (dimension)
\ltj@zh (dimension)
\jfam (attribute)
\ltj@curjfnt (attribute)
\ltj@charclass (attribute)
\ltj@yablshift (attribute)
\ltj@ykblshift (attribute)
\ltj@autospc (attribute)
\ltj@autoxspc (attribute)
\ltj@icflag (attribute) italic (1)

```

packed (2)
kinsoku (3)
from_jfm (4)
line_end (5)
kanji_skip (6)
xkanji_skip (7)
processed (8)
ic_processed (9)
boxbdd (15)
\ltj@kcati (attribute)

30111

30112

30113

30114 Nodes for indicating beginning of a paragraph. A paragraph which is started by `\item` in list-like environments has a horizontal box for its label before the actual contents. So ...

```

1 \ltjsetparameter{kanjiskip=0pt}ふがふが.%
2 \setbox0=\hbox{\ltjsetparameter{kanjiskip
   =5pt}{ほげほげ}          ふがふが.ほげほげ.ぴよぴよ
3 \box0.ぴよぴよ\par

```

void package(int c)

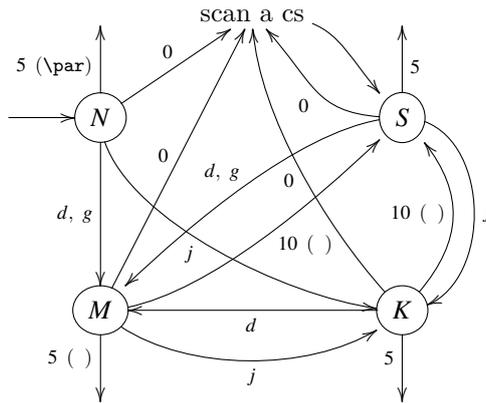
```

{
    scaled h;          /* height of box */
    halfword p;        /* first node in a box */
    scaled d;          /* max depth */
    int grp;
    grp = cur_group;
    d = box_max_depth;
    unsave();
    save_ptr -= 4;
    if (cur_list.mode_field == -hmode) {
        cur_box = filtered_hpack(cur_list.head_field,
                                cur_list.tail_field, saved_value(1),
                                saved_level(1), grp, saved_level(2));
        subtype(cur_box) = HLIST_SUBTYPE_HBOX;
    }
}

```

LuaTeX-ja における和文処理グルーの挿入方法は、pTeX のそれとは全く異なる。pTeX では次のような仕様であった:

- JFM グルーの挿入は、和文文字を表すトークンを元に水平リストに (文字を表す) `(char_node)` を追加する過程で行われる。
- `xkanjiskip` の挿入は、水平ボックスへのパッケージングや行分割前に行われる。
- `kanjiskip` はノードとしては挿入されない。パッケージングや行分割の計算時に「和文文字を表す 2 つの `(char_node)` の間には `kanjiskip` がある」ものとみなされる。

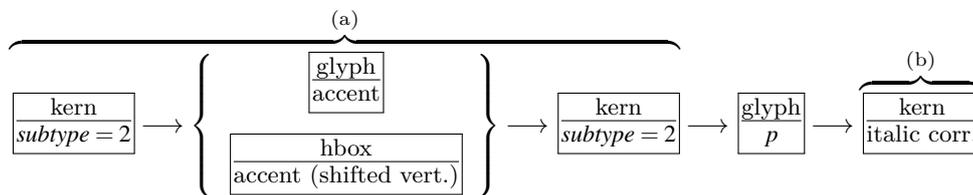


$d := \{3,4,6,7,8,11,12,13\}$, $g := \{1,2\}$, $j := (\text{Japanese characters})$

しかし、LuaTeX-ja では、水平ボックスへのパッケージングや行分割前に全ての **JAgglue**、即ち JFM グルー `xkanjiskip` `kanjiskip` の 3 種類を一度に挿入することになっている。これは、LuaTeX において欧文の合字 カーニング処理がノードベースになったことに対応する変更である。

LuaTeX-ja における **JAgglue** 挿入処理では、次節で定義する「クラスタ」を単位にして行われる。大雑把にいうと、「クラスタ」は文字とそれに付随するノード達（アクセント位置補正用のカーンや、イタリック補正）をまとめたものであり、2つのクラスタの間には、ペナルティ、`\vadjust`、`whatsit` など、行組版には関係しないものがある。

定義 1.



■ *id* の意味 $Np.id$ の意味を述べるとともに、「先頭の文字」を表す $glyph_node\ Np.head$ と、「最後の文字」を表す $glyph_node\ Np.tail$ を次のように定義する。直感的に言うとも、 Np は $Np.head$ で始まり $Np.tail$ で終わるような単語、と見做すことができる。これら $Np.head$, $Np.tail$ は説明用に準備した概念であって、実際の Lua コード中にそのように書かれているわけではないことに注意。

id_jglyph 和文文字。

$Np.head$, $Np.tail$ は、その和文文字を表している $glyph_node$ そのものである。

id_glyph 和文文字を表していない $glyph_node\ p$ 。

多くの場合、 p は欧文文字を格納しているが、「冪」などの合字によって作られた $glyph_node$ である可能性もある。前者の場合、 $Np.head$, $Np.tail = p$ である。一方、後者の場合、

- $Np.head$ は、合字の構成要素の先頭→ (その $glyph_node$ における) 合字の構成要素の先頭→……と再帰的に検索していったどり着いた $glyph_node$ である。
- $Np.last$ は、同様に末尾→末尾→と検索してたどり着いた $glyph_node$ である。

id_math インライン数式。

便宜的に、 $Np.head$, $Np.tail$ とともに「文字コード -1 の欧文文字」とおく。

id_hlist 縦方向にシフトされていない水平ボックス。

この場合、*Np.head*, *Np.tail* はそれぞれ *p* の内容を表すリストの、先頭 末尾のノードである。

- 状況によっては、 $\text{T}_{\text{E}}\text{X}$ ソースで言うと

```
\hbox{\hbox{abc}...\hbox{\lower1pt\hbox{xyz}}}
```

のように、*p* の内容が別の水平ボックスで開始 終了している可能性も十分あり得る。そのような場合、*Np.head*, *Np.tail* の算出は、垂直方向にシフトされていない水平ボックスの場合だけ内部を再帰的に探索する。例えば上の例では、*Np.head* は文字「a」を表すノードであり、一方 *Np.tail* は垂直方向にシフトされた水平ボックス、`\lower1pt\hbox{xyz}` に対応するノードである。

- また、先頭にアクセント付きの文字がきたり、末尾にイタリック補正用のカーンが来ることもあり得る。この場合は、クラスタの定義のところにもあったように、それらは無視して算出を行う。
- 最初 最後のノードが合字によって作られた *glyph_node* のときは、それぞれに対して *id_glyph* と同様に再帰的に構成要素をたどっていく。

id_pbox 「既に処理された」ノードのリストであり、これらのノードが二度処理を受けないためにまとめて1つのクラスタとして取り扱うだけである。*id_hlist* と同じ方法で *Np.head*, *Np.tail* を算出する、

id_disc discretionary break (`\discretionary{pre}{post}{nobreak}`).

id_hlist と同じ方法で *Np.head*, *Np.tail* を算出するが、第3引数の *nobreak* (行分割が行われない時の内容) を使う。言い換えれば、ここで行分割が発生した時の状況は全く考慮に入れない。

id_box_like *id_hlist* とならない box や、rule。

この場合は、*Np.head*, *Np.tail* のデータは利用されないで、2つの算出は無意味である。敢えて明示するならば、*Np.head*, *Np.tail* は共に nil 値である。

他 以上がない *id* に対しても、*Np.head*, *Np.tail* の算出は無意味。

■**クラスタの別の分類** さらに、JFM グルー挿入処理の実際の説明により便利なように、*id* とは別のクラスタの分類を行っておく。挿入処理では2つの隣り合ったクラスタの間に空白等の実際の挿入を行うことは前に書いたが、ここでの説明では、問題にしているクラスタ *Np* は「後ろ側」のクラスタであるとする。「前側」のクラスタについては、以下の説明で *head* が *last* に置き換わることに注意すること。

和文 A リスト中に直接出現している和文文字。*id* が *id_jglyph* であるか、

id が *id_pbox* であって *Np.head* が **J**Achar であるとき。

和文 B リスト中の水平ボックスの中身の先頭として出現した和文文字。和文 A との違いは、これの前に JFM グルーの挿入が行われない (`xkanjiskip`, `kanjiskip` は入り得る) ことである。

id が *id_hlist* か *id_disc* であって *Np.head* が **J**Achar であるとき。

欧文 リスト中に直接/水平ボックスの中身として出現している欧文文字。次の3つの場合が該当:

- *id* が *id_glyph* である。
- *id* が *id_math* である。
- *id* が *id_pbox* か *id_hlist* か *id_disc* であって、*Np.head* が **A**Lchar。

箱 box, またはそれに類似するもの。次の2つが該当:

- *id* が *id_pbox* か *id_hlist* か *id_disc* であって、*Np.head* が *glyph_node* でない。
- *id* が *id_box_like* である。

4.11 段落 / 水平ボックスの先頭や末尾

■**先頭部の処理** まず、段落／水平ボックスの一番最初にあるクラスタ N_p を探索する．水平ボックスの場合は何の問題もないが、段落の場合では以下のノード達を事前に読み飛ばしておく：

`\parindent` 由来の水平ボックス ($subtype = 3$)、及び $subtype$ が 44 ($user_defined$) でないような `whatsit`．

これは、`\parindent` 由来の水平ボックスがクラスタを構成しないようにするためである．

次に、 N_p の直前に空白 g を必要なら挿入する：

1. この処理が働くような N_p は和文 A である．
2. 問題のリストが字下げありの段落 (`\parindent` 由来の水平ボックスあり) の場合は、この空白 g は「文字コード '`parbdd`' の文字」と N_p の間に入るグルー／カーンである．
3. そうでないとき (`noindent` で開始された段落や水平ボックス) は、 g は「文字コード '`boxbdd`' の文字」と N_p の間に入るグルー／カーンである．

ただし、もし g が `glue` であった場合、この挿入によって N_p による行分割が新たに可能になるべきではない．そこで、以下の場合には、 g の直前に `\penalty10000` を挿入する：

- 問題にしているリストが段落であり、かつ
- N_p の前には予めペナルティがなく、 g は `glue`．

■**末尾の処理** 末尾の処理は、問題のリストが段落のものか水平ボックスのものかによって異なる．後者の場合は容易い：最後のクラスタを N_q とおくと、 N_q と「文字コード '`boxbdd`' の文字」の間に入るグルー／カーンを、 N_q の直後に挿入するのみである．

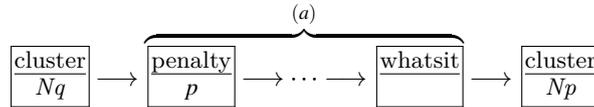
一方、前者（段落）の場合は、リストの末尾は常に `\penalty10000` と、`\parfillskip` 由来のグルーが存在する．よって、最後のクラスタ N_p はこの `\parfillskip` 由来のグルーとなり、実質的な中身の最後はその1つ前のクラスタ N_q となる．

1. まず N_q の直後に（後に述べる）`line-end [E]` によって定まる空白を挿入する．
2. 次に、段落の最後の「通常の和文文字 + 句点」が独立した行となるのを防ぐために、`jcharwidowpenalty` の値の分だけ適切な場所のペナルティを増やす．
ペナルティ量を増やす場所は、`head` が **J**Achar であり、かつその文字の `kcatcode` が偶数であるような最後のクラスタの直前にあるものたちである*2．

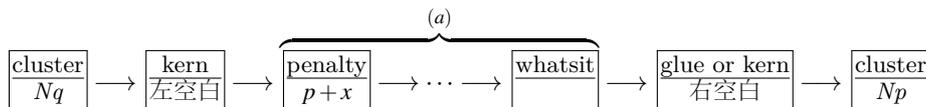
*2 大雑把に言えば、`kcatcode` が奇数であるような **J**Achar を約物として考えていることになる．`kcatcode` の最下位ビットはこの `jcharwidowpenalty` 用にのみ利用される．

4.12 概観と典型例：2つの「和文 A」の場合

先に述べたように、2つの隣り合ったクラス、 N_q と N_p の間には、ペナルティ、`\vadjust`、`whatsit` など、行組版には関係しないものがある。模式的に表すと、



のようになっている。間の (a) に相当する部分には、何のノードもない場合ももちろんあり得る。そうして、JFM グルー挿入後には、この2クラス間には次のようになる：



以後、典型的な例として、クラス N_q と N_p が共に和文 A である場合を見ていこう、この場合が全ての場合の基本となる。

■「右空白」の算出 まず、「右空白」にあたる量を算出する。通常はこれが、隣り合った2つの和文文字間に入る空白量となる。

JFM 由来 [M] JFM の文字クラス指定によって入る空白を以下によって求める。この段階で空白量が未定義（未指定）だった場合、デフォルト値 `kanjiskip` を採用することとなるので、次へ。

1. もし両クラスの間で `\inhibitglue` が実行されていた場合（証として `whatsit` ノードが自動挿入される）、代わりに `kanjiskip` が挿入されることとなる。次へ。
2. N_q と N_p が同じ JFM 同じ `jfmvar` キー 同じサイズの和文フォントであったならば、共通に使っている JFM 内で挿入される空白（グルーかカーン）が決まっているか調べる。
3. 1. でも 2. でもない場合は、 N_q と N_p が違う JFM/`jfmvar`/サイズである。この場合、まず

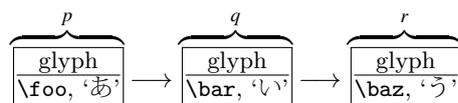
$$\begin{aligned} gb &:= (\text{ } N_q \text{ と「文字コードが } \textit{diffmet} \textit{」の文字} \text{ との間に入るグルー／カーン}) \\ ga &:= (\text{ 「文字コードが } \textit{diffmet} \textit{」の文字} \text{ と } N_p \text{ との間に入るグルー／カーン}) \end{aligned}$$

として、左側由来 右側由来の空白（グルー／カーン）を（それぞれの JFM から）求める。 ga と gb のどちらか片方が未定義であるならば、定義されている側の値をそのまま採用する。もし ga と gb が両方決まっているならば、両者の値を平均^{*3}した値を採用する。

例えば、

```
\jfont\foo=psft:Ryumin-Light:jfm=ujis
\jfont\bar=psft:GothicBBB-Medium:jfm=ujis
\jfont\baz=psft:GothicBBB-Medium:jfm=ujis;jfmvar=piyo
```

という3フォントを考え、



という3ノードを考える（それぞれ単独でクラスをなす）。この場合、 p と q の間は、実フォントが異なるにもかかわらず (2) の状況となる一方で、 q と r の間は（実フォントが同じなのに）`jfmvar` キーの内容が異なるので (3) の状況となる。

*3 `differentjfm` パラメタの値によって、「大きい方」「小さい方」「合計」に変えることができる。

kanjiskip [K] 上の [M] において空白が定まらなかった場合、kanjiskip の値を以下で定め、それを「右空白」として採用する。この段階においては、\inhibitglue は効力を持たないため、結果として、2つの和文文字間には常に何らかのグルー／カーンが挿入されることとなる。

1. 両クラスタ（厳密には $Nq.tail$, $Np.head$ ）の中身の文字コードに対する autospacing パラメタが両方とも false だった場合は、長さ 0 の glue とする。
2. ユーザ側から見た kanjiskip パラメタの自然長が $\backslash maxdimen = (2^{30} - 1)sp$ でなければ、kanjiskip パラメタの値を持つ glue を採用する。
3. 2. でない場合は、 Nq , Np で使われている JFM に指定されている kanjiskip の値を用いる。どちらか片方のクラスタだけが和文文字（和文 A 和文 B）のときは、そちらのクラスタで使われている JFM 由来の値だけを用いる。もし両者で使われている JFM が異なった場合は、上の [M] 3. と同様の方法を用いて調整する。

■「左空白」の算出とそれに伴う補正 次に、「左空白」にあたる量を算出する：

line-end [E] Nq と Np の間で行分割が起きたときに、 Nq と行末の間に入る空白である。ぶら下げ組の組版などに用いられることを期待している。

1. 既に算出した「右空白」がカーンである場合は、「左空白」は挿入されない。
2. 「右空白」が glue か未定義（長さ 0 の glue とみなす）の場合は、「左空白」は Nq と「文字コード 'lineend' の文字」との間に入るカーンとして、JFM から決定される。
3. 2. で決まった「左空白」の長さが 0 でなければ、その分だけ先ほど算出した「右空白」の自然長を引く。

■禁則用ペナルティの挿入 まず、

$$a := (Nq^{*4} \text{の文字に対する } postbreakpenalty \text{ の値}) + (Np^{*5} \text{の文字に対する } prebreakpenalty \text{ の値})$$

とおく。ペナルティは通常 $[-10000, 10000]$ の整数値をとり、また ± 10000 は正負の無限大を意味することになっているが、この a の算出では単純な整数の加減算を行う。

a は禁則処理用に Nq と Np の間に加えられるべきペナルティ量である。

P-normal [PN] Nq と Np の間の (a) 部分にペナルティ ($penalty_node$) があれば処理は簡単である：これらの各ノードにおいて、ペナルティ値を (± 10000 を無限大として扱いつつ) a だけ増加させればよい。また、 $10000 + (-10000) = 0$ としている。

少々困るのは、(a) 部分にペナルティが存在していない場合である。直感的に、補正すべき量 a が 0 でないとき、その値をもつ $penalty_node$ を作って「右空白」の（もし未定義なら Np の）直前に挿入……ということになるが、実際には僅かにこれより複雑である。

- 「右空白」がカーンであるとき、それは「 Nq と Np の間で改行は許されない」ことを意図している。そのため、この場合は $a \neq 0$ であってもペナルティの挿入はしない。
- 「左空白」がカーンとしてきっちり定義されている時（このとき、「右空白」はカーンでない）、この「左空白」の直後での行分割を許容しないといけないので、 $a = 0$ であっても $penalty_node$ を作って挿入する。
- 以上のどれでもないときは、 $a \neq 0$ ならば $penalty_node$ を作って挿入する。

*5 厳密にはそれぞれ $Nq.tail$, $Np.head$ 。

$Np \downarrow$	和文 A	和文 B	欧文	箱	glue	kern
和文 A	$\frac{E \quad M \rightarrow K}{PN}$	$\frac{O_A \rightarrow K}{PN}$	$\frac{O_A \rightarrow X}{PN}$	$\frac{O_A}{PA}$	$\frac{O_A}{PN}$	$\frac{O_A}{PS}$
和文 B	$\frac{E \quad O_B \rightarrow K}{PA}$	$\frac{K}{PS}$	$\frac{X}{PS}$			
欧文	$\frac{E \quad O_B \rightarrow X}{PA}$	$\frac{X}{PS}$				
箱	$\frac{E \quad O_B}{PA}$					
glue	$\frac{E \quad O_B}{PN}$					
kern	$\frac{E \quad O_B}{PS}$					

4.13 その他の場合

本節の内容は表 4.12 にまとめてある。

■和文 A と欧文の間 Nq が和文 A で、 Np が欧文の場合、JFM グルー挿入処理は次のようにして行われる。

- 「右空白」については、まず以下に述べる Boundary-B [O_B] により空白を決定しようと試みる。それが失敗した場合は、`xkanjiskip [X]` によって定める。
- 「左空白」については、既に述べた line-end [E] をそのまま採用する。それに伴う「右空白」の補正も同じ。
- 禁則用ペナルティも、以前述べた P-normal [PN] と同じである。

Boundary-B [O_B] 和文文字と「和文でないもの」との間に入る空白を以下によって求め、未定義でなければそれを「右空白」として採用する。JFM-origin [M] の変種と考えて良い。これによって定まる空白の典型例は、和文の閉じ括弧と欧文文字の間に入る半角アキである。

1. もし両クラスタの間で`\inhibitglue`が実行されていた場合（証として`whatsit` ノードが自動挿入される）、次へ。
2. そうでなければ、 Nq と「文字コードが'`jcharbdd`' の文字」との間に入るグルー／カーンとして定まる。

`xkanjiskip [X]` この段階では、`kanjiskip [K]` のときと同じように、`xkanjiskip` の値を以下で定め、それを「右空白」として採用する。この段階で`\inhibitglue` は効力を持たないのも同じである。

1. 以下のいずれかの場合は、`xkanjiskip` の挿入は抑止される。しかし、実際には行分割を許容するために、長さ 0 の glue を採用する：
 - 両クラスタにおいて、それらの中身の文字コードに対する `autoxspacing` パラメタが共に `false` である。
 - Nq の中身の文字コードについて、「直後への `xkanjiskip` の挿入」が禁止されている（つまり、`jaxspmode` (or `alxspmode`) パラメタが 2 以上）。
 - Np の中身の文字コードについて、「直前への `xkanjiskip` の挿入」が禁止されている（つまり、`jaxspmode` (or `alxspmode`) パラメタが偶数）。

2. ユーザ側から見た `xkanjiskip` パラメタの自然長が $\backslash\maxdimen = (2^{30} - 1)\text{sp}$ でなければ、`xkanjiskip` パラメタの値を持つ `glue` を採用する。
3. 2. でない場合は、 N_q , N_p (和文 A/和文 B なのは片方だけ) で使われている JFM に指定されている `xkanjiskip` の値を用いる。

■**欧文と和文 A の間** N_q が欧文で、 N_p が和文 A の場合、JFM グルー挿入処理は上の場合とほぼ同じである。和文 A のクラスタが逆になるので、Boundary-A [O_A] の部分が変わるだけ。

- 「右空白」については、まず以下に述べる Boundary-A [O_A] により空白を決定しようと試みる。それが失敗した場合は、`xkanjiskip [X]` によって定める。
- N_q が和文でないので、「左空白」は算出されない。
- 禁則用ペナルティは、以前述べた P-normal [PN] と同じである。

Boundary-A [O_A] 「和文でないもの」と和文文字との間に入る空白を以下によって求め、未定義でなければそれを「右空白」として採用する。JFM-origin [M] の変種と考えて良い。これによって定まる空白の典型例は、欧文文字と和文の開き括弧との間に入る半角アキである。

1. もし両クラスタの間で `\inhibitglue` が実行されていた場合 (証として `whatsit` ノードが自動挿入される)、次へ。
2. そうでなければ、「文字コードが `'jcharbdd'` の文字」と N_p との間に入るグルー／カーンとして定まる。

■**和文 A と箱 グルー カーンの間** N_q が和文 A で、 N_p が箱 グルー カーンのいずれかであった場合、両者の間に挿入される JFM グルーについては同じ処理である。しかし、そこでの行分割に対する仕様が異なるので、ペナルティの挿入処理は若干異なったものとなっている。

- 「右空白」については、既に述べた Boundary-B [O_B] により空白を決定しようと試みる。それが失敗した場合は、「右空白」は挿入されない。
- 「左空白」については、既に述べた line-end [E] の算出方法をそのまま採用する。それに伴う「右空白」の補正も同じ。
- 禁則用ペナルティの処理は、後ろのクラスタ N_p の種類によって異なる。なお、 $N_p.head$ は無意味であるから、「 $N_p.head$ に対する `prebreakpenalty` の値」は 0 とみなされる。言い換えれば、

$$a := (N_q^{*6} \text{の文字に対する } \text{postbreakpenalty} \text{ の値}).$$

箱 N_p が箱であった場合は、両クラスタの間での行分割は (明示的に両クラスタの間に `\penalty10000` があった場合を除き) いつも許容される。そのため、ペナルティ処理は、後に述べる P-allow [PA] が P-normal [PN] の代わりに用いられる。

グルー N_p がグルーの場合、ペナルティ処理は P-normal [PN] を用いる。

カーン N_p がカーンであった場合は、両クラスタの間での行分割は (明示的に両クラスタの間にペナルティがあった場合を除き) 許容されない。ペナルティ処理は、後に述べる P-suppress [PS] を使う。

これらの P-normal [PN], P-allow [PA], P-suppress [PS] の違いは、 N_q と N_p の間 (以前の図だと (a) の部分) にペナルティが存在しない場合にのみ存在する。

P-allow [PA] N_q と N_p の間の (a) 部分にペナルティがあれば、P-normal [PN] と同様に、それらの各ノードにおいてペナルティ値を a だけ増加させる。

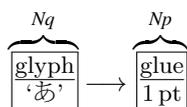
(a) 部分にペナルティが存在していない場合, LuaTeX-ja は Nq と Np の間の行分割を可能にしようとする. そのために, 以下の場合に a をもつ `penalty_node` を作って「右空白」の (もし未定義なら Np の) 直前に挿入する:

- 「右空白」がグルーでない (カーンか未定義) であるとき.
- 「左空白」がカーンとしてきちり定義されている時.

P-suppress [PS] Nq と Np の間の (a) 部分にペナルティがあれば, **P-normal [PN]** と同様に, それらの各ノードにおいてペナルティ値を a だけ増加させる.

(a) 部分にペナルティが存在していない場合, Nq と Np の間の行分割は元々不可能のはずだったのであるが, LuaTeX-ja はそれをわざわざ行分割可能にはしない. そのため, 「右空白」が glue であれば, その直前に `\penalty10000` を挿入する.

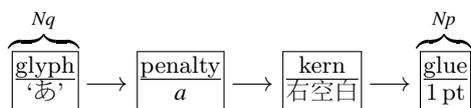
なお, 「右空白」はカーン, 「左空白」は未定義の



のような状況を考える. このとき, a , 即ち「あ」の `postbreakpenalty` がいかなる値であっても, この 2 クラスタ間は最終的に



となり, a 分のペナルティは挿入されないことに注意して欲しい. `postbreakpenalty` は (a は) 殆どの場合が非負の値と考えられ, そのような場合では (1) と



との間に差異は生じない*7.

■箱 グルー カーンと和文 A の間 Np が箱 グルー カーンのいずれかで, Np が和文 A であった場合は, すぐ上の (Nq と Np の順序が逆になっている) 場合とほぼ同じであるが, 「左空白」がなくなることにのみ注意.

- 「右空白」については, 既に述べた **Boundary-A [O_A]** により空白を決定しようと試みる. それが失敗した場合は, 「右空白」は挿入されない.
- Nq が和文でないので, 「左空白」は算出されない.
- 禁則用ペナルティの処理は, Nq の種類によって異なる. $Nq.tail$ は無意味なので,

$$a := (Np^{*8} \text{の文字に対する } \text{prebreakpenalty} \text{ の値}).$$

箱 Nq が箱の場合は, **P-allow [PA]** を用いる.

グルー Nq がグルーの場合は, **P-normal [PN]** を用いる.

カーン Nq がカーンの場合は, **P-suppress [PS]** を用いる.

*7 `kern→glue` が 1 つの行分割可能点 (行分割に伴うペナルティは 0) であるため, たとえ $a = 10000$ であっても, Nq と Np の間で行分割を禁止することはできない.

■和文 A と和文 B の違い 先に述べたように、和文 B は水平ボックスの中身の先頭 (or 末尾) として出現している和文文字である。リスト内に直接ノードとして現れている和文文字 (和文 A) との違いは、

- 和文 B に対しては、JFM の文字クラス指定から定まる空白 JFM-origin [M], Boundary-A [O_A], Boundary-B [O_B] の挿入は行われない。「左空白」の算出も行われない。例えば、
 - 片方が和文 A, もう片方が和文 B のクラスタの場合, Boundary-A [O_A] または Boundary-B [O_B] の挿入を試み, それがダメなら kanjiskip [K] の挿入を行う。
 - 和文 B の 2 つのクラスタの間には, kanjiskip [K] が自動的に入る。
- 和文 B と箱 グルー カーンが隣接したとき (どちらが前かは関係ない), 間に JFM グルー ペナルティの挿入は一切しない。
- 和文 B と和文 B, また和文 B と欧文とが隣接した時は, 禁則用ペナルティ挿入処理は P-suppress [PS] が用いられる。
- 和文 B の文字に対する prebreakpenalty, postbreakpenalty の値は使われず, 0 として計算される。

次が具体例である:

1 あ . \inhibitglue A\\	あ . A
2 \hbox{あ . }A\\	あ . A
3 あ . A	あ . A

- 1 行目の \inhibitglue は Boundary-B [O_B] の処理のみを抑止するので, ピリオドと「A」の間には xkanjiskip (四分アキ) が入ることに注意。
- 2 行目のピリオドと「A」の間においては, 前者が和文 B となる (水平ボックスの中身の末尾として登場しているから) ので, そもそも Boundary-B [O_B] の処理は行われない。よって, xkanjiskip が入ることとなる。
- 3 行目では, ピリオドの属するクラスタは和文 A である。これによって, ピリオドと「A」の間には Boundary-B [O_B] 由来の半角アキが入ることになる。

5 psft

参考文献

- [1] Victor Eijkhout, *TEX by Topic, A T_EXnician's Reference*, Addison-Wesley, 1992.