LuaTEX-ja パッケージ

LuaTEX-ja プロジェクトチーム

2013 年 11 月 2 日

目次

本ドキュメントはまだまだ未完成です．

第 I 部
ユーザーズマニュアル

1　はじめに

LuaTEX-ja パッケージは，次世代標準 TEX である LuaTEX の上で，pTEX と同等／それ以上の品質の日本語組版を実現させようとするマクロパッケージである．

1.1　背景

従来，「TEX を用いて日本語組版を行う」といったとき，エンジンとしては ASCII pTEX やそれの拡張物が用いられることが一般的であった．pTEX は TEX のエンジン拡張であり，（少々仕様上不便な点はあるものの）商業印刷の分野にも用いられるほどの高品質な日本語組版を可能としている．だが，それは弱点にもなってしまった：pTEX という（組版的に）満足なものがあったため，海外で行われている数々の TEX の拡張——例えば ε-TEX や pdfTEX——や，TrueType，OpenType，Unicode といった計算機で日本語を扱う際の状況の変化に追従することを怠ってしまったのだ．

ここ数年，若干状況は改善されてきた．現在手に入る大半の pTEX バイナリでは外部 UTF-8 入力が利用可能となり，さらに Unicode 化を推進し，pTEX の内部処理まで Unicode 化した upTEX も開発されている．また，pTEX に ε-TEX 拡張をマージした ε-pTEX も登場し，TEX Live 2011 では pLATEX が ε-pTEX の上で動作するようになった．だが，pdfTEX 拡張（PDF 直接出力や micro-typesetting）を pTEX に対応させようという動きはなく，海外との gap は未だにあるのが現状である．

しかし，LuaTEX の登場で，状況は大きく変わることになった．Lua コードで ‘callback’ を書くことにより，LuaTEX の内部処理に割り込みをかけることが可能となった．これは，エンジン拡張という真似をしなくても，Lua コードとそれに関する TEX マクロを書けば，エンジン拡張とほぼ同程度のことができるようになったということを意味する．LuaTEX-ja は，このアプローチによって Lua コード・TEX マクロによって日本語組版を LuaTEX の上で実現させようという目的で開発が始まったパッケージである．

1.2　pTEX からの主な変更点

LuaTEX-ja は，pTEX に多大な影響を受けている．初期の開発目標は，pTEX の機能を Lua コードにより実装することであった．しかし，開発が進むにつれ，pTEX の完全な移植は不可能であり，また pTEX における実装がいささか不可解になっているような状況も発見された．そのため，**LuaTEX-ja は，もはや pTEX の完全な移植は目標とはしない．pTEX における不自然な仕様・挙動があれば，そこは積極的に改める．**

以下は pTEX からの主な変更点である．

- 和文フォントは（小塚明朝，IPA 明朝などの）実際のフォント，和文フォントメトリック（JFM と呼ぶ[*1]）の組である．
- 日本語の文書中では改行はほとんどどこでも許されるので，pTEX では和文文字直後の改行は無視される（スペースが入らない）ようになっていた．しかし，LuaTEX-ja では LuaTEX の仕様のためにこの機能は完全には実装されていない．
- 2 つの和文文字の間や，和文文字と欧文文字の間に入るグルー／カーン（両者をあわせて **JAglue** と呼ぶ）の挿入処理が 0 から書き直されている．

[*1] 混乱を防ぐため，pTEX の意味での JFM (`min10.tfm`) などは本ドキュメントでは**和文用 TFM** とよぶことにする．

  - LuaTeX の内部での文字の扱いが「ノードベース」になっているように（例えば，`of{}fice` で合字は抑制されない），**JAglue** の挿入処理も「ノードベース」である．
  - さらに，2 つの文字の間にある行末では効果を持たないノード（例えば `\special` ノード）や，イタリック補正に伴い挿入されるカーンは挿入処理中では無視される．
  - **注意：上の 2 つの変更により，従来 JAglue の挿入処理を分断するのに使われていたいくつかの方法は用いることができない．具体的には，次の方法はもはや無効である：**
    `\hskip2\zw ちょ{}っと\hskip2\zw ちょ\/っと`
    もし同じことをやりたければ，空の水平ボックスを間に挟めばよい：
    `\hskip2\zw ちょ\hbox{}っと`
  - 処理中では，2 つの和文フォントは，「実際の」フォントが異なるだけの場合には同一視される．
- LuaTeX-ja では，pTeX と同様に漢字・仮名を制御綴内に用いることができ，**\西暦** などが正しく動作するようにしている．但し，制御綴中に使える和文文字が pTeX・upTeX と全く同じではないことに注意すること．
- 現時点では，縦書きは LuaTeX-ja ではサポートされていない．

詳細については第 III 部を参照．

1.3 用語と記法

本ドキュメントでは，以下の用語と記法を用いる：

- 文字は 2 種類に分けられる：
  - **JAchar**: ひらがな，カタカナ，漢字，和文用の約物といった日本語組版に使われる文字のことを指す．
  - **ALchar**: アルファベットを始めとする，その他全ての文字を指す．

  そして，**ALchar** の出力に用いられるフォントを「欧文フォント」と呼び，**JAchar** の出力に用いられるフォントを「和文フォント」と呼ぶ．
- サンセリフ体で書かれた語（例：prebreakpenalty）は日本語組版用のパラメータを表し，これらは `\ltjsetparameter` コマンドのキーとして用いられる．
- 下線付きタイプライタ体の語（例：fontspec）は LaTeX のパッケージやクラスを表す．
- 本ドキュメントでは，自然数は 0 から始まる．

1.4 プロジェクトについて

■プロジェクト Wiki　プロジェクト Wiki は構築中である．

- http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage（日本語）
- http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage%28en%29（英語）
- http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage%28zh%29（中国語）

本プロジェクトは SourceForge.JP のサービスを用いて運営されている．

■開発メンバー

- 北川 弘典
- 黒木 裕介
- 本田 知亮
- 前田 一貴
- 阿部 紀行
- 齋藤 修三郎
- 八登 崇之
- 山本 宗宏
- 馬 起園

2 使い方

2.1 インストール

LuaTEX-ja パッケージのインストールには，次のものが必要である．

- LuaTEX beta-0.74.0 (or later)
- luaotfload v2.2
- luatexbase v0.6 (2013/05/04)
- xunicode v0.981 (2011/09/09)
- adobemapping (Adobe cmap and pdfmapping files)

本バージョン以降の LuaTEX-ja は TEX Live 2012 以前では動作しない．これは，LuaTEX と luaotfload が TEX Live 2013 において更新されたことによる．逆に，20130318.1 以前の LuaTEX-ja は TEX Live 2013 では動作しない．

現在，LuaTEX-ja は以下のアーカイブ，およびディストリビューションに収録されている：

- CTAN (`macros/luatex/generic/luatexja`)
- MiKTEX (`luatexja.tar.lzma`)
- TEX Live (`texmf-dist/tex/luatex/luatexja`)
- W32TEX (`luatexja.tar.xz`)

例えば TEX Live 2013 を利用しているなら，LuaTEX-ja は TEX Live manager (`tlmgr`) を使ってインストールすることができる．

```
$ tlmgr install luatexja
```

手動でインストールする場合の方法は以下のようになる：

1. ソースアーカイブを以下のいずれかの方法で取得する．現在公開されているのはあくまでも開発版であって，安定版でないことに注意．
   - Git リポジトリの内容をコピーする：
     `$ git clone git://git.sourceforge.jp/gitroot/luatex-ja/luatexja.git`
   - `master` ブランチのスナップショット（`tar.gz` 形式）をダウンロードする．
     http://git.sourceforge.jp/view?p=luatex-ja/luatexja.git;a=snapshot;h=HEAD;sf=tgz.

   `master` ブランチ（従って，CTAN 内のアーカイブも）はたまにしか更新されないことに注意．主な開発は `master` の外で行われ，比較的まとまってきたらそれを `master` に反映させることにしている．
2. 「Git リポジトリをコピー」以外の方法でアーカイブを取得したならば，それを展開する．`src/` をはじめとしたいくつかのディレクトリができるが，動作には `src/`以下の内容だけで十分．
3. もし CTAN から本パッケージを取得したのであれば，日本語用クラスファイルや標準の禁則処理用パラメータを格納した `ltj-kinsoku.lua` を生成するために，以下を実行する必要がある：

   ```
   $ cd src
   $ lualatex ltjclasses.ins
   $ lualatex ltjsclasses.ins
   $ lualatex ltjltxdoc.ins
   ```

```
$ luatex   ltj-kinsoku_make.tex
```

ここで使用した `*.{dtx,ins}` と `ltj-kinsoku_make.tex` は通常の使用にあたっては必要ない.

4. `src` の中身を自分の `TEXMF` ツリーにコピーする. 場所の例としては, 例えば `TEXMF/tex/luatex/luatexja/` がある．シンボリックリンクが利用できる環境で，かつリポジトリを直接取得したのであれば，（更新を容易にするために）コピーではなくリンクを貼ることを勧める.
5. 必要があれば，`mktexlsr` を実行する.

2.2 注意点

- 原稿のソースファイルの文字コードは UTF-8 固定である．従来日本語の文字コードとして用いられてきた EUC-JP や Shift-JIS は使用できない.
- LuaTEX-ja は動作が pTEX に比べて非常に遅い．コードを変更して徐々に速くしているが，まだ満足できる速度ではない．LuaJITTEX を用いると LuaTEX のだいたい 1.3 倍の速度で動くようである.
- **MiKTEX 利用者への注意：**LuaTEX-ja が動作するためには，`UniJIS2004-UTF32-H`, `Adobe-Japan1-UCS2` という 2 つの CMap が Kpathsearch によって見つけられることが必要である．TEX Live や W32TEX ユーザは普通にインストールすればそのようになっているはずである.
  確認するには，以下のように `kpsewhich` コマンドを実行すればよい：

```
$ kpsewhich -format=cmap  UniJIS2004-UTF32-H
/opt/texlive/2013/texmf-dist/fonts/cmap/adobemapping/aj16/CMap/UniJIS2004-UTF32-H
$ kpsewhich -format=cmap  Adobe-Japan1-UCS2
/opt/texlive/2013/texmf-dist/fonts/cmap/adobemapping/ToUnicode/Adobe-Japan1-UCS2
```

2.3 plain TEX で使う

LuaTEX-ja を plain TEX で使うためには，単に次の行をソースファイルの冒頭に追加すればよい:

```
\input luatexja.sty
```

これで（`ptex.tex` のように）日本語組版のための最低限の設定がなされる：

- 以下の 6 つの和文フォントが定義される：

字体	フォント名	‘10 pt’	‘7 pt’	‘5 pt’
明朝体	Ryumin-Light	`\tenmin`	`\sevenmin`	`\fivemin`
ゴシック体	GothicBBB-Medium	`\tengt`	`\sevengt`	`\fivegt`

  – ‘Ryumin-Light’ と ‘GothicBBB-Medium’ は PDF ファイルに埋め込まずに名前参照のみで用いることが広く受け入れられており，この場合 PDF リーダーが適切な外部フォントで代用する（例えば，Adobe Reader では Ryumin-Light は小塚明朝で代替される）．そこで，これらを引き続きデフォルトのフォントとして採用する.
  – 欧文フォントの文字は和文フォントの文字よりも，同じ文字サイズでも一般に小さくデザインされている．そこで，標準ではこれらの和文フォントの実際のサイズは指定された値よりも小さくなるように設定されており，具体的には指定の 0.962216 倍にスケールされる．この

0.962216 という数値も，pTEX におけるスケーリングを踏襲した値である．

- **JAchar** と **ALchar** の間に入るグルー (xkanjiskip) の量は次のように設定されている：

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

2.4 LaTeX で使う

■LaTeX 2_ε LaTeX 2_ε を用いる場合も基本的には同じである．日本語組版のための最低限の環境を設定するためには，`luatexja.sty` を読み込むだけでよい：

```
\usepackage{luatexja}
```

これで pLaTeX の `plfonts.dtx` と `pldefs.ltx` に相当する最低限の設定がなされる：

- JY3 は和文フォント用のフォントエンコーディングである（横書き用）．
  将来的に，LuaTeX-ja で縦書きがサポートされる際には，JT3 を縦書き用として用いる予定である．
- pLaTeX と同様に，標準では「明朝体」「ゴシック体」の 2 種類を用いる：

字体			ファミリ名
明朝体	`\textmc{...}`	`{\mcfamily ...}`	`\mcdefault`
ゴシック体	`\textgt{...}`	`{\gtfamily ...}`	`\gtdefault`

- 標準では，次のフォントファミリが用いられる：

字体	ファミリ	`\mdseries`	`\bfseries`	スケール
明朝体	`mc`	Ryumin-Light	GothicBBB-Medium	0.962216
ゴシック体	`gt`	GothicBBB-Medium	GothicBBB-Medium	0.962216

  どちらのファミリにおいても，その bold シリーズで使われるフォントはゴシック体の medium シリーズで使われるフォントと同じあることに注意．これは初期の DTP において和文フォントが 2 つ（それがちょうど Ryumin-Light，GothicBBB-Medium だった）しか利用できなかった時の名残であり，pLaTeX での標準設定とも同じである．
- 数式モード中の和文文字は明朝体 (`mc`) で出力される．

しかしながら，上記の設定は日本語の文書にとって十分とは言えない．日本語文書を組版するためには，`article.cls`，`book.cls` といった欧文用のクラスファイルではなく，和文用のクラスファイルを用いた方がよい．現時点では，`jclasses`（pLaTeX の標準クラス）と `jsclasses`（奥村晴彦氏によるクラスファイル）に対応するものとして，`ltjclasses`，`ltjsclasses` がそれぞれ用意されている．

3 フォントの変更

3.1 plain TeX and LaTeX 2_ε

■plain TeX plain TeX で和文フォントを変更するためには，pTEX のように `\jfont` 命令を直接用いる．5.1 節を参照．

■LaTeX 2_ε (NFSS2) LaTeX 2_ε については，LuaTeX-ja ではフォント選択システムを pLaTeX 2_ε (`plfonts.dtx`) の大部分をそのまま採用している．

- \fontfamily, \fontseries, \fontshape, そして\selectfont が和文フォントの属性を変更するために使用できる．

	エンコーディング	ファミリ	シリーズ	シェープ	選択
欧文	\romanencoding	\romanfamily	\romanseries	\romanshape	\useroman
和文	\kanjiencoding	\kanjifamily	\kanjiseries	\kanjishape	\usekanji
両方	—	–	\fontseries	\fontshape	—
自動選択	\fontencoding	\fontfamily	—	—	\usefont

ここで，\fontencoding{<encoding>} は，引数により和文側か欧文側かのどちらかのエンコーディングを変更する．例えば，\fontencoding{JY3} は和文フォントのエンコーディングを JY3 に変更し，\fontencoding{T1} は欧文フォント側を T1 へと変更する．\fontfamily も引数により和文側，欧文側，**あるいは両方**のフォントファミリを変更する．詳細は 8.1 節を参照すること．

- 和文フォントファミリの定義には \DeclareFontFamily の代わりに \DeclareKanjiFamily を用いる．しかし，現在の実装では \DeclareFontFamily を用いても問題は生じない．

■注意：数式モード中の和文文字　pTEX では，特に何もしないでも数式中に和文文字を記述することができた．そのため，以下のようなソースが見られた：

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

$f_{高温}$ ($f_{\text{high temperature}}$).

$$y=(x-1)^2+2 \quad よって \quad y>0$$

$5\in 素:=\{\,p\in\mathbb{N}:p \text{ is a prime}\,\}$.

LuaTEX-ja プロジェクトでは，数式モード中での和文文字はそれらが識別子として用いられるときのみ許されると考えている．この観点から，

- 上記数式のうち 1, 2 行目は正しくない．なぜならば ‘高温’ が意味のあるラベルとして，‘よって’ が接続詞として用いられているからである．
- しかしながら，3 行目は ‘素’ が識別子として用いられているので正しい．

したがって，LuaTEX-ja プロジェクトの意見としては，上記の入力は次のように直されるべきである：

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

$f_{高温}$ ($f_{\text{high temperature}}$).

$$y=(x-1)^2+2 \quad よって \quad y>0$$

$5\in 素:=\{\,p\in\mathbb{N}:p \text{ is a prime}\,\}$.

また LuaTEX-ja プロジェクトでは，和文文字が識別子として用いられることはほとんどないと考えており，したがってこの節では数式モード中の和文フォントを変更する方法については記述しない．この方法については 5.4 節を参照のこと．

3.2　fontspec

fontspec パッケージと同様の機能を和文フォントに対しても用いるためには，luatexja-fontspec パッケージをプリアンブルで読み込む必要がある．このパッケージは必要ならば自動で luatexja パッケージと fontspec パッケージを読み込む．

luatexja-fontspec パッケージでは，以下の 7 つのコマンドを fontspec パッケージの元のコマンドに対応するものとして定義している：

和文	\jfontspec	\setmainjfont	\setsansjfont
欧文	\fontspec	\setmainfont	\setsansfont
和文	\newjfontfamily	\newjfontface	\defaultjfontfeatures
欧文	\newfontfamily	\newfontface	\defaultfontfeatures
和文	\addjfontfeatures		
欧文	\addfontfeatures		

```
\fontspec[Numbers=OldStyle]{LMSans10-Regular}
\jfontspec{IPAexMincho}
JIS~X~0213:2004→辻
                                        JIS X 0213:2004 →辻
                                        JIS X 0208:1990 →辻
\jfontspec[CJKShape=JIS1990]{IPAexMincho}
JIS~X~0208:1990→辻
```

和文フォントについては全ての和文文字のグリフがほぼ等幅であるのが普通であるため，\setmonojfont コマンドは存在しないことに注意．また，これらの和文用の 7 つのコマンドでは Kerning feature はデフォルトでは off となっている．これはこの feature が **JAglue** と衝突するためである（5.1 節を参照）．

3.3 プリセット設定

よく使われている和文フォント設定を一行で指定できるようにしたのが luatexja-preset パッケージである．このパッケージは，otf パッケージの一部機能と八登崇之氏による PXchfon パッケージの一部機能とを合わせたような格好をしており，内部で luatexja-fontspec，従って fontspec を読み込んでいる．

もし fontspec パッケージに何らかのオプションを渡す必要がある[*2]場合は，次のように luatexja-preset の前に fontspec を手動で読みこめば良い：

```
\usepackage[no-math]{fontspec}
\usepackage[...]{luatexja-preset}
```

■一般的なオプション

`nodeluxe` LaTeX 2_ε 環境下での標準設定のように，明朝体・ゴシック体を各 1 ウェイトで使用する．より具体的に言うと，この設定の下では\mcfamily\bfseries，\gtfamily\bfseries，\gtfamily\mdseries はみな同じフォントとなる．このオプションは標準で有効になっている．

`deluxe` 明朝体 2 ウェイト・ゴシック体 3 ウェイトと，丸ゴシック体 (\mgfamily, \textmg{...}) を使用可能とする．ゴシック体は細字・太字・極太の 3 ウェイトがあるが，極太ゴシック体はファミリの切り替え (\gtebfamily, \textgteb{...}) で実現している．fontspec では通常 (\mdseries) と太字 (\bfseries) しか扱えないためにこのような中途半端な実装になっている．

`expert` 横組専用仮名を用いる．また，\rubyfamily でルビ用仮名が使用可能となる．

`bold` 「明朝の太字」をゴシック体の太字によって代替する．

[*2] 例えば，数式フォントまで置換されてしまい，\mathit によってギリシャ文字の斜体大文字が出なくなる，など．

`90jis` 出来る限り 90JIS の字形を使う．

`jis2004` 出来る限り JIS2004 の字形を使う．

`jis` 用いる JFM を（JIS フォントメトリック類似の）`jfm-jis.lua` にする．このオプションがない時は LuaTEX-ja 標準の `jfm-ujis.lua` が用いられる．

`90jis` と `jis2004` については本パッケージで定義された明朝体・ゴシック体（・丸ゴシック体）にのみ有効である．両オプションが同時に指定された場合の動作については全く考慮していない．

■多ウェイト用プリセットの一覧 `morisawa-pro`, `morisawa-pr6n` 以外はフォントの指定は（ファイル名でなく）フォント名で行われる．

`kozuka-pro` Kozuka Pro (Adobe-Japan1-4) fonts.

`kozuka-pr6` Kozuka Pr6 (Adobe-Japan1-6) fonts.

`kozuka-pr6n` Kozuka Pr6N (Adobe-Japan1-6, JIS04-savvy) fonts.

小塚 Pro 書体・Pr6N 書体は Adobe InDesign 等の Adobe 製品にバンドルされている．「小塚丸ゴシック」は存在しないので，便宜的に小塚ゴシック H によって代用している．

family	series	`kozuka-pro`	`kozuka-pr6`	`kozuka-pr6n`
明朝	medium	KozMinPro-Regular	KozMinProVI-Regular	KozMinPr6N-Regular
	bold	KozMinPro-Bold	KozMinProVI-Bold	KozMinPr6N-Bold
ゴシック	medium	KozGoPro-Regular*	KozGoProVI-Regular*	KozGoPr6N-Regular*
		KozGoPro-Medium	KozGoProVI-Medium	KozGoPr6N-Medium
	bold	KozGoPro-Bold	KozGoProVI-Bold	KozGoPr6N-Bold
	heavy	KozGoPro-Heavy	KozGoProVI-Heavy	KozGoPr6N-Heavy
丸ゴシック		KozGoPro-Heavy	KozGoProVI-Heavy	KozGoPr6N-Heavy

上の表において，*つきのフォント (KozGo…-Regular) は，*`deluxe`* オプション非指定時にゴシック体細字として用いられる．

`hiragino-pro` Hiragino Pro (Adobe-Japan1-5) fonts.

`hiragino-pron` Hiragino ProN (Adobe-Japan1-5, JIS04-savvy) fonts.

ヒラギノフォントは，Mac OS X 以外にも，一太郎 2012 の上位エディションにもバンドルされている．極太ゴシックとして用いるヒラギノ角ゴ W8 は，Adobe-Japan1-3 の範囲しかカバーしていない Std/StdN フォントであり，その他は Adobe-Japan1-5 対応である．

family	series	`hiragino-pro`	`hiragino-pron`
明朝	medium	Hiragino Mincho Pro W3	Hiragino Mincho ProN W3
	bold	Hiragino Mincho Pro W6	Hiragino Mincho ProN W6
ゴシック	medium	Hiragino Kaku Gothic Pro W3*	Hiragino Kaku Gothic ProN W3*
		Hiragino Kaku Gothic Pro W6	Hiragino Kaku Gothic ProN W6
	bold	Hiragino Kaku Gothic Pro W6	Hiragino Kaku Gothic ProN W6
	heavy	Hiragino Kaku Gothic Std W8	Hiragino Kaku Gothic StdN W8
丸ゴシック		Hiragino Maru Gothic ProN W4	Hiragino Maru Gothic ProN W4

`morisawa-pro` Morisawa Pro (Adobe-Japan1-4) fonts.

`morisawa-pr6n` Morisawa Pr6N (Adobe-Japan1-6, JIS04-savvy) fonts.

family	series	morisawa-pro	morisawa-pr6n
明朝	medium	A-OTF-RyuminPro-Light.otf	A-OTF-RyuminPr6N-Light.otf
	bold	A-OTF-FutoMinA101Pro-Bold.otf	A-OTF-FutoMinA101Pr6N-Bold.otf
ゴシック	medium	A-OTF-GothicBBBPro-Medium.otf	A-OTF-GothicBBBPr6N-Medium.otf
	bold	A-OTF-FutoGoB101Pro-Bold.otf	A-OTF-FutoGoB101Pr6N-Bold.otf
	heavy	A-OTF-MidashiGoPro-MB31.otf	A-OTF-MidashiGoPr6N-MB31.otf
丸ゴシック		A-OTF-Jun101Pro-Light.otf	A-OTF-Jun101Pr6N-Light.otf

`yu-win` Yu fonts bundled with Windows 8.1.

`yu-osx` Yu fonts bundled with OSX Mavericks. 上の `yu-win` とはフォント名が微妙に異なることに注意.

family	series	yu-win	yu-osx
明朝	medium	YuMincho-Regular	YuMincho Medium
	bold	YuMincho-Demibold	YuMincho Demibold
ゴシック	medium	YuGothic-Regular*	YuGothic Medium*
		YuGothic-Bold	YuGothic Bold
	bold	YuGothic-Bold	YuGothic Bold
	heavy	YuGothic-Bold	YuGothic Bold
丸ゴシック		YuGothic-Bold	YuGothic Bold

■単ウェイト用プリセット一覧　次に，単ウェイト用の設定を述べる．この 4 設定では「細字」「太字」の区別はない．また，丸ゴシック体はゴシック体と同じフォントを用いる．

	noembed	ipa	ipaex	ms
明朝体	Ryumin-Light（非埋込）	IPA 明朝	IPAex 明朝	MS 明朝
ゴシック体	GothicBBB-Medium（非埋込）	IPA ゴシック	IPAex ゴシック	MS ゴシック

■HG フォントの利用　すぐ前に書いた単ウェイト用設定を，Microsoft Office 等に付属する HG フォントを使って多ウェイト化した設定もある．

	ipa-hg	ipaex-hg	ms-hg
明朝体細字	IPA 明朝	IPAex 明朝	MS 明朝
明朝体太字	HG 明朝 E		
ゴシック体細字			
単ウェイト時	IPA ゴシック	IPAex ゴシック	MS ゴシック
`jis2004` 指定時	IPA ゴシック	IPAex ゴシック	MS ゴシック
それ以外の時	HG ゴシック M		
ゴシック体太字	HG ゴシック E		
ゴシック体極太	HG 創英角ゴシック UB		
丸ゴシック体	HG 丸ゴシック体 PRO		

なお，HG 明朝 E・HG ゴシック E・HG 創英角ゴシック UB・HG 丸ゴシック体 PRO の 4 つについては，内部で

標準　フォント名（`HGMinchoE` など）

`90jis` **指定時**　ファイル名 (`hgrme.ttc`, `hgrge.ttc`, `hgrsgu.ttc`, `hgrsmp.ttf`)

`jis2004` **指定時**　ファイル名 (`hgrme04.ttc`, `hgrge04.ttc`, `hgrsgu04.ttc`, `hgrsmp04.ttf`)

として指定を行っているので注意すること.

3.4 \CID, \UTF と otf パッケージのマクロ

pLaTeX では，JIS X 0208 にない Adobe-Japan1-6 の文字を出力するために，齋藤修三郎氏による otf パッケージが用いられていた．このパッケージは広く用いられているため，LuaTeX-ja においても otf パッケージの機能の一部をサポートしている．これらの機能を用いるためには luatexja-otf パッケージを読み込めばよい．

```
\jfontspec{KozMinPr6N-Regular.otf}
森\UTF{9DD7}外と内田百\UTF{9592}とが\UTF{9
    AD9}島屋に行く。

\CID{7652}飾区の\CID{13706}野家，
\CID{1481}城市，葛西駅，
高崎と\CID{8705}\UTF{FA11}

\aj半角{はんかくカタカナ}
```

森鷗外と内田百閒とが髙島屋に行く。
葛飾区の吉野家，葛城市，葛西駅，高崎と髙﨑
ﾊﾝｶｸｶﾀｶﾅ

otf パッケージでは，それぞれ次のようなオプションが存在した：

deluxe　明朝体・ゴシック体各 2 ウェイトと，丸ゴシック体を扱えるようになる．
expert　仮名が横組・縦組専用のものに切り替わり，ルビ用仮名も扱えるようになる．
bold　ゴシック体を標準で太いウェイトのものに設定する．

しかしこれらのオプションは luatexja-otf パッケージには存在しない．otf パッケージが文書中で使用する和文用 TFM を自前の物に置き換えていたのに対し，luatexja-otf パッケージでは，そのようなことは行わないからである．

これら 3 オプションについては，luatexja-preset パッケージにプリセットを使う時に一緒に指定するか，あるいは対応する内容を 3.1 節 (NFSS2) や 3.2 節 (fontspec) の方法で手動で指定する必要がある．

4 パラメータの変更

LuaTeX-ja には多くのパラメータが存在する．そして LuaTeX の仕様のために，その多くは TeX のレジスタにではなく，LuaTeX-ja 独自の方法で保持されている．そのため，これらのパラメータを設定・取得するためには \ltjsetparameter と \ltjgetparameter を用いる必要がある．

4.1 JAchar の範囲の設定

JAchar の範囲を設定するためには，まず各文字に 0 より大きく 217 より小さい index を割り当てる必要がある．これには \ltjdefcharrange を用いる．例えば，次のように書くことで追加漢字面 (SIP) にある全ての文字と ‘漢’ が「100 番の文字範囲」に属するように設定される．

```
\ltjdefcharrange{100}{"20000-"2FFFF,`漢}
```

この文字範囲の割り当ては常にグローバルであり，したがって文書の途中でこの操作をするべきではない．

もし指定されたある文字がある非零番号の範囲に属していたならば，これは新しい設定で上書きされる．例えば，SIP は全て LuaTeX-ja のデフォルトでは 4 番の文字範囲に属しているが，上記の指定を行えば SIP は 100 番に属すようになり，4 番からは除かれる．

文字範囲に番号を割り当てた後は，jacharrange パラメータが **JAchar** として扱われる文字の範囲を設定するために用いられる．例えば，以下は LuaTEX-ja の初期設定である：

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

jacharrange パラメータには整数のリストを与える．リスト中の負の整数 $-n$ は「文字範囲 n に属する文字は **ALchar** として扱われる」ことを意味し，正の整数 $+n$ は **JAchar** として扱うことを意味する．

■初期設定　LuaTEX-ja では 8 つの文字範囲を設定している．これらは以下のデータに基づいて決定している．

- Unicode 6.0 のブロック．
- Adobe-Japan1-6 の CID と Unicode の間の対応表 `Adobe-Japan1-UCS2`.
- 八登崇之氏による upTEX 用の `PXbase` バンドル．

以下ではこれら 8 つの文字範囲について記述する．番号のあとのアルファベット ‘J’ と ‘A’ はデフォルトで **JAchar** か **ALchar** かを表している．これらの設定は `PXbase` バンドルで定義されている `prefercjk` と類似のものである．

範囲 8$^{\mathsf{J}}$　ISO 8859-1 の上位領域（ラテン 1 補助）と JIS X 0208 の共通部分にある記号．この文字範囲は以下の文字で構成される：

- § (`U+00A7`, Section Sign)
- ¨ (`U+00A8`, Diaeresis)
- ° (`U+00B0`, Degree sign)
- ± (`U+00B1`, Plus-minus sign)
- ´ (`U+00B4`, Spacing acute)
- ¶ (`U+00B6`, Paragraph sign)
- × (`U+00D7`, Multiplication sign)
- ÷ (`U+00F7`, Division Sign)

範囲 1$^{\mathsf{A}}$　ラテン文字．一部は Adobe-Japan1-6 にも含まれている．この範囲は以下の Unicode のブロックから構成されている．**ただし，範囲 8 は除く．**

- `U+0080`–`U+00FF`: Latin-1 Supplement
- `U+0100`–`U+017F`: Latin Extended-A
- `U+0180`–`U+024F`: Latin Extended-B
- `U+0250`–`U+02AF`: IPA Extensions
- `U+02B0`–`U+02FF`: Spacing Modifier Letters
- `U+0300`–`U+036F`: Combining Diacritical Marks
- `U+1E00`–`U+1EFF`: Latin Extended Additional

範囲 2$^{\mathsf{J}}$　ギリシャ文字とキリル文字．JIS X 0208（したがってほとんどの和文フォント）はこれらの文字を持つ．

- `U+0370`–`U+03FF`: ギリシア文字・コプト文字
- `U+0400`–`U+04FF`: キリル文字
- `U+1F00`–`U+1FFF`: キリル文字補助

範囲 3$^{\mathsf{J}}$　句読点と記号類．ブロックのリストは表 1 に示してある．

範囲 4$^{\mathsf{A}}$　通常和文フォントには含まれていない文字．この範囲は他の範囲にないほとんど全ての Unicode ブロックで構成されている．したがって，ブロックのリストを示す代わりに，範囲の定義そのものを示す：

```
\ltjdefcharrange{4}{%
  "500-"10FF, "1200-"1DFF, "2440-"245F, "27C0-"28FF, "2A00-"2AFF,
```

表 1. 文字範囲 3 に指定されている Unicode ブロック.

U+2000–U+206F	General Punctuation	U+2070–U+209F	Superscripts and Subscripts
U+20A0–U+20CF	Currency Symbols	U+20D0–U+20FF	Comb. Diacritical Marks for Symbols
U+2100–U+214F	Letterlike Symbols	U+2150–U+218F	Number Forms
U+2190–U+21FF	Arrows	U+2200–U+22FF	Mathematical Operators
U+2300–U+23FF	Miscellaneous Technical	U+2400–U+243F	Control Pictures
U+2500–U+257F	Box Drawing	U+2580–U+259F	Block Elements
U+25A0–U+25FF	Geometric Shapes	U+2600–U+26FF	Miscellaneous Symbols
U+2700–U+27BF	Dingbats	U+2900–U+297F	Supplemental Arrows-B
U+2980–U+29FF	Misc. Mathematical Symbols-B	U+2B00–U+2BFF	Miscellaneous Symbols and Arrows

表 2. 文字範囲 6 に指定されている Unicode ブロック.

U+2460–U+24FF	Enclosed Alphanumerics	U+2E80–U+2EFF	CJK Radicals Supplement
U+3000–U+303F	CJK Symbols and Punctuation	U+3040–U+309F	Hiragana
U+30A0–U+30FF	Katakana	U+3190–U+319F	Kanbun
U+31F0–U+31FF	Katakana Phonetic Extensions	U+3200–U+32FF	Enclosed CJK Letters and Months
U+3300–U+33FF	CJK Compatibility	U+3400–U+4DBF	CJK Unified Ideographs Extension A
U+4E00–U+9FFF	CJK Unified Ideographs	U+F900–U+FAFF	CJK Compatibility Ideographs
U+FE10–U+FE1F	Vertical Forms	U+FE30–U+FE4F	CJK Compatibility Forms
U+FE50–U+FE6F	Small Form Variants	U+20000–U+2FFFF	(Supplementary Ideographic Plane)

表 3. 文字範囲 7 に指定されている Unicode ブロック.

U+1100–U+11FF	Hangul Jamo	U+2F00–U+2FDF	Kangxi Radicals
U+2FF0–U+2FFF	Ideographic Description Characters	U+3100–U+312F	Bopomofo
U+3130–U+318F	Hangul Compatibility Jamo	U+31A0–U+31BF	Bopomofo Extended
U+31C0–U+31EF	CJK Strokes	U+A000–U+A48F	Yi Syllables
U+A490–U+A4CF	Yi Radicals	U+A830–U+A83F	Common Indic Number Forms
U+AC00–U+D7AF	Hangul Syllables	U+D7B0–U+D7FF	Hangul Jamo Extended-B

```
"2C00-"2E7F, "4DC0-"4DFF, "A4D0-"A82F, "A840-"ABFF, "FB00-"FE0F,
"FE20-"FE2F, "FE70-"FEFF, "10000-"1FFFF, "E000-"F8FF} % non-Japanese
```

範囲 5^{A}　代用符号と補助私用領域.

範囲 6^{J}　日本語で用いられる文字．ブロックのリストは表 2 に示す.

範囲 7^{J}　CJK 言語で用いられる文字のうち，Adobe-Japan1-6 に含まれていないもの．ブロックのリストは表 3 に示す.

4.2　kanjiskip と xkanjiskip

JAglue は以下の 3 つのカテゴリに分類される：

- JFM で指定されたグルー／カーン．もし `\inhibitglue` が和文文字の周りで発行されていれば，このグルーは挿入されない.
- デフォルトで 2 つの **JAchar** の間に挿入されるグルー (kanjiskip).
- デフォルトで **JAchar** と **ALchar** の間に挿入されるグルー (xkanjiskip).

kanjiskip や xkanjiskip の値は以下のようにして変更可能である.

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

JFM は「望ましい kanjiskip の値」や「望ましい xkanjiskip の値」を持っていることがある．これらのデータを使うためには，kanjiskip や xkanjiskip の値を \maxdimen の値に設定すればよい．

4.3 xkanjiskip の挿入設定

xkanjiskip がすべての **JAchar** と **ALchar** の境界に挿入されるのは望ましいことではない．例えば，xkanjiskip は開き括弧の後には挿入されるべきではない（‘(あ’ と ‘(あ’ を比べてみよ）．LuaTEX-ja では xkanjiskip をある文字の前／後に挿入するかどうかを，**JAchar** に対しては jaxspmode を，**ALchar** に対しては alxspmode をそれぞれ変えることで制御することができる．

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

p あq い! う

2 つ目の引数の preonly は「xkanjiskip の挿入はこの文字の前でのみ許され，後では許さない」ことを意味する．他に指定可能な値は postonly, allow, inhibit である．

なお，現行の仕様では，jaxspmode，alxspmode はテーブルを共有しており，上のコードの 1 行目を次のように変えても同じことになる：

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

また，これら 2 パラメータには数値で値を指定することもできる（6.2 節を参照）．

もし全ての kanjiskip と xkanjiskip の挿入を有効化／無効化したければ，それぞれ autospacing と autoxspacing を true/false に設定すればよい．

4.4 ベースラインの移動

和文フォントと欧文フォントを合わせるためには，時々どちらかのベースラインの移動が必要になる．pTEX ではこれは \ybaselineshift を非零の長さに設定することでなされていた（欧文フォントのベースラインが下がる）．しかし，日本語が主ではない文書に対しては，欧文フォントではなく和文フォントのベースラインを移動した方がよい．このため，LuaTEX-ja では欧文フォントのベースラインのシフト量（yalbaselineshift パラメータ）と和文フォントのベースラインのシフト量（yjabaselineshift パラメータ）を独立に設定できるようになっている．

```
\vrule width 150pt height 0.4pt depth 0pt\
    hskip-120pt
\ltjsetparameter{yjabaselineshift=0pt,
    yalbaselineshift=0pt}abcあいう
\ltjsetparameter{yjabaselineshift=5pt,
    yalbaselineshift=2pt}abcあいう
```

abc あいう abc あいう

上の例において引かれている水平線がベースラインである．

この機能には面白い使い方がある：2 つのパラメータを適切に設定することで，サイズの異なる文字を中心線に揃えることができる．以下は一つの例である（値はあまり調整されていないことに注意）：

```
xyz漢字
{\scriptsize
  \ltjsetparameter{yjabaselineshift=-1pt,
    yalbaselineshift=-1pt}
  XYZひらがな
}abcかな
```

xyz 漢字 XYZ ひらがな abc かな

第 II 部
リファレンス

5　フォントメトリックと和文フォント

5.1　\jfont 命令

フォントを和文フォントとして読み込むためには，\jfont を\font プリミティブの代わりに用いる．\jfont の文法は \font と同じである．LuaTEX-ja は luaotfload パッケージを自動的に読み込むので，TrueType/OpenType フォントに feature を指定したものを和文フォントとして用いることができる：

```
\jfont\tradgt={file:KozMinPr6N-Regular.otf:script=latn;%
  +trad;-kern;jfm=ujis} at 14pt
\tradgt 当／体／医／区
```

當／體／醫／區

なお，\jfont で定義されたコントロールシーケンス（上の例だと \tradgt）は *font_def* トークンではないので，\fontname\tradgt のような入力はエラーとなることに注意する．以下では \jfont で定義されたコントロールシーケンスを ⟨*jfont_cs*⟩ で表す．

■JFM 「はじめに」の節で述べたように，JFM は文字と和文組版で自動的に挿入されるグルー／カーンの寸法情報を持っている．JFM の構造は次の小節で述べる．\jfont 命令の呼び出しの際には，どの JFM を用いるのかを以下のキーで指定する必要がある：

jfm=⟨*name*⟩ JFM の名前を指定する．もし以前に指定された JFM が読み込まれていなければ，jfm-⟨*name*⟩.lua を読み込む．以下の JFM が LuaTEX-ja には同梱されている：

jfm-ujis.lua　LuaTEX-ja の標準 JFM である．この JFM は upTEX で用いられる UTF/OTF パッケージ用の和文用 TFM である upnmlminr-h.tfm を元にしている．luatexja-otf パッケージを使うときはこの JFM を指定するべきである．

jfm-jis.lua　pTEX で広く用いられている「JIS フォントメトリック」jis.tfm に相当する JFM である．jfm-ujis.lua とこの jfm-jis.lua の主な違いは，jfm-ujis.lua ではほとんどの文字が正方形状であるのに対し，jfm-jis.lua では横長の長方形状である．

jfm-min.lua　pTEX に同梱されているデフォルトの和文用 TFM である min10.tfm に相当する JFM である．この JFM と他の 2 つの JFM の間には表 4 に示すような特筆すべき違いがある．

jfmvar=⟨*string*⟩ 標準では，JFM とサイズが同じで，実フォントだけが異なる 2 つの和文フォントは「区別されない」．例えば下の例において，最初の「）」と「【」の実フォントは異なるが，JFM もサイズも同じなので，普通に「）【」と入力した時と同じように半角空きとなる．
しかし，時には……

表 4. LuaTEX-ja に同梱されている JFM の違い

	jfm-ujis.lua	jfm-jis.lua	jfm-min.lua
例 1[4]	◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました. ちょっと！何	◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました. ちょっと！何	◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました. ちょっと！何
例 2	漢	漢	漢
Bounding Box			

```
\ltjsetparameter{differentjfm=both}
\jfont\F=file:KozMinPr6N-Regular.otf:jfm=ujis
\jfont\G=file:KozGoPr6N-Medium.otf:jfm=ujis
\jfont\H=file:KozGoPr6N-Medium.otf:jfm=ujis;jfmvar=
    hoge

\F ） {\G 【】} （ % halfwidth space
   ） {\H 『』} （ % fullwidth space

\ltjsetparameter{differentjfm=paverage}
```

）【】（ ）『』（

■注意：kern feature　いくつかのフォントはグリフ間のスペースについての情報を持っている．しかし，この情報は LuaTEX-ja とはあまり相性がよくない．具体的には，この情報に基づいて挿入されるカーニングスペースは **JAglue** の挿入過程の**前に**挿入され，JFM に基づくグルー／カーンも挿入される場合には 2 文字間の意図しないスペースの原因となる．

- `script=...` といった feature を使いたい場合には，`\jfont` に`-kern` を指定するべきである．
- もしプロポーショナル幅の和文フォントをそのフォントの情報に基づいて使いたいならば，`jfm-prop.lua` を JFM として指定し，……TODO: kanjiskip?

■`extend` と `slant`　OpenType font feature と見かけ上同じような形式で指定できるものに，

`extend=`⟨*extend*⟩　横方向に ⟨*extend*⟩ 倍拡大する．
`slant=`⟨*slant*⟩　⟨*slant*⟩ に指定された割合だけ傾ける．

の 2 つがある．`extend` や `slant` を指定した場合は，それに応じた JFM を指定すべきである[*3]．例えば，次の例では無理やり通常の JFM を使っているために，文字間隔やイタリック補正量が正しくない：

```
\jfont\E=file:KozMinPr6N-Regular.otf:extend=1.5;jfm=ujis
\E あいうえお

\jfont\S=file:KozMinPr6N-Regular.otf:slant=1;jfm=ujis
\S あいう\/ABC
```

あいうえお
*あいう*ABC

[*3] LuaTEX-ja では，これらに対する JFM を特に提供することはしない予定である．

5.2 psft プリフィックス

file: と name: のプリフィックスに加えて，\jfont（と \font プリミティブ）では psft: プリフィックスを用いることができる．このプリフィックスを用いることで，PDF には埋め込まれない「名前だけの」和文フォントを指定することができる．「標準的な」和文フォント，つまり ‘Ryumin-Light’ と ‘GothicBBB-Medium’ の指定でこのプリフィックスが使われる．

psft プリフィックスの下では +jp90 などの OpenType font feature の効力はない．非埋込フォントを PDF に使用すると，実際にどのようなフォントが表示に用いられるか予測できないからである． extend と slant 指定は単なる変形のため psft プリフィックスでも使用可能である．

■cid キー 標準で psft:プリフィックスで定義されるフォントは日本語用のものであり，Adobe-Japan1-6 の CID に対応したものとなる．しかし，LuaTEX-ja は中国語の組版にも威力を発揮することが分かり，日本語フォントでない非埋込フォントの対応も必要となった．そのために追加されたのが cid キーである．

cid キーに値を指定すると，その CID を持った非埋込フォントを定義することができる：

```
\jfont\testJ={psft:Ryumin-Light:cid=Adobe-Japan1-6;jfm=jis}     % Japanese
\jfont\testD={psft:Ryumin-Light:jfm=jis}                        % default value is Adobe-
    Japan1-6
\jfont\testC={psft:AdobeMingStd-Light:cid=Adobe-CNS1-6;jfm=jis} % Traditional Chinese
\jfont\testG={psft:SimSun:cid=Adobe-GB1-5;jfm=jis}              % Simplified Chinese
\jfont\testK={psft:Batang:cid=Adobe-Korea1-2;jfm=jis}           % Korean
```

上のコードでは中国語・韓国語用フォントに対しても JFM に日本語用の jfm-jis.lua を指定しているので注意されたい．

今のところ，LuaTEX-ja は上のサンプルコード中に書いた 4 つの値しかサポートしていない．

```
\jfont\test={psft:Ryumin-Light:cid=Adobe-Japan2;jfm=jis}
```

のようにそれら以外の値を指定すると，エラーが発生する：

```
! Package luatexja Error: bad cid key `Adobe-Japan2'.

See the luatexja package documentation for explanation.
Type  H <return>  for immediate help.
<to be read again>
                   \par
l.78

? h
I couldn't find any non-embedded font information for the CID
`Adobe-Japan2'. For now, I'll use `Adobe-Japan1-6'.
Please contact the LuaTeX-ja project team.
?
```

5.3 JFM ファイルの構造

JFM ファイルはただ一つの関数呼び出しを含む Lua スクリプトである：

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

実際のデータは上で `{ ... }` で示されたテーブルの中に格納されている．以下ではこのテーブルの構造について記す．なお，JFM ファイル中の長さは全て design-size を単位とする浮動小数点数であることに注意する．

`dir=`⟨*direction*⟩（必須）
: JFM の書字方向．現時点では横書き (`'yoko'`) のみがサポートされる．

`zw=`⟨*length*⟩（必須）
: 「全角幅」の長さ．

`zh=`⟨*length*⟩（必須）
: 「全角高さ」(height + depth) の長さ．

`kanjiskip={`⟨*natural*⟩`,` ⟨*stretch*⟩`,` ⟨*shrink*⟩`}`（任意）
: 「理想的な」kanjiskip の量を指定する．4.2 節で述べたように，もし kanjiskip が `\maxdimen` の値ならば，このフィールドで指定された値が実際には用いられる（もしこのフィールドが JFM で指定されていなければ，0 pt であるものとして扱われる）．⟨*stretch*⟩ と ⟨*shrink*⟩ のフィールドも design-size が単位であることに注意せよ．

`xkanjiskip={`⟨*natural*⟩`,` ⟨*stretch*⟩`,` ⟨*shrink*⟩`}`（任意）
: `kanjiskip` フィールドと同様に，xkanjiskip の「理想的な」量を指定する．

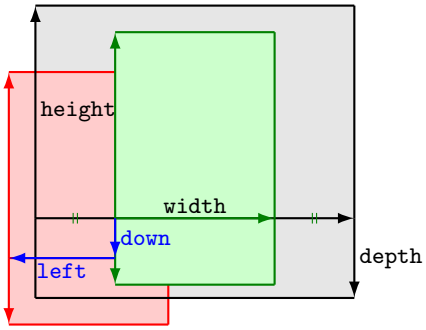
■文字クラス　上記のフィールドに加えて，JFM ファイルはそのインデックスが自然数であるいくつかのサブテーブルを持つ．インデックスが $i \in \omega$ であるテーブルは「文字クラス」i の情報を格納する．少なくとも，文字クラス 0 は常に存在するので，JFM ファイルはインデックスが `[0]` のサブテーブルを持たなければならない．それぞれのサブテーブル（そのインデックスを i で表わす）は以下のフィールドを持つ：

`chars={`⟨*character*⟩`, ...}`（文字クラス 0 を除いて必須）
: このフィールドは文字クラス i に属する文字のリストである．このフィールドは $i = 0$ の場合には任意である（文字クラス 0 には，0 以外の文字クラスに属するものを除いた全ての **JAchar** が属するから）．このリスト中で文字を指定するには，以下の方法がある：
  - Unicode におけるコード番号
  - 「`'あ'`」のような，文字それ自体
  - 「`'あ*'`」のような，文字それ自体の後にアスタリスクをつけたもの
  - いくつかの「仮想的な文字」（後に説明する）

`width=`⟨*length*⟩`, height=`⟨*length*⟩`, depth=`⟨*length*⟩`, italic=`⟨*length*⟩（必須）
: 文字クラス i に属する文字の幅，高さ，深さ，イタリック補正の量を指定する．文字クラス i に属する全ての文字は，その幅，高さ，深さがこのフィールドで指定した値であるものとして扱われる．しかし，例外が一つある：もし `'prop'` が `width` フィールドに指定された場合，文字の幅はその「実際の」グリフの幅となる．

`left=`⟨*length*⟩`, down=`⟨*length*⟩`, align=`⟨*align*⟩
: これらのフィールドは「実際の」グリフの位置を調整するためにある．`align` フィールドに指定できる値は`'left'`，`'middle'`，`'right'` のいずれかである．もしこれら 3 つのフィールドのうちの 1 つが省かれた場合，`left` と `down` は 0，`align` フィールドは`'left'` であるものとして扱われる．これら 3 つのフィールドの意味については図 1 で説明する．
  多くの場合，`left` と `down` は 0 である一方，`align` フィールドが`'middle'` や`'right'` であることは珍しいことではない．例えば，`align` フィールドを`'right'` に指定することは，文字クラスが開き括弧類であるときに実際必要である．

`kern={[`j`]=`⟨*kern*⟩`, [`j'`]={`⟨*kern*⟩`, [`⟨*ratio*⟩`]}, ...}`



align フィールドの値が'middle' である和文文字を含むノードを考えよう．

- 黒色の長方形はノードの枠である．その幅，高さ，深さは JFM によって指定される．
- align フィールドは middle なので，「実際の」グリフは水平方向の中心に配置される（緑色の長方形）．
- さらに，グリフは left と down の値に従ってシフトされる．最終的な実際のグリフの位置は赤色の長方形で示された位置になる．

図 1. 「実際の」グリフの位置．

glue={[j]={⟨*width*⟩, ⟨*stretch*⟩, ⟨*shrink*⟩, [⟨*priority*⟩], [⟨*ratio*⟩]}, ...} 文字クラス i の文字と j の文字の間に挿入される kern や glue の量を指定する．

⟨*priority*⟩ は luatexja-adjust.sty による優先順位付き行長調整（9.3 節）が有効なときのみ意味を持つ．このフィールドは省略可能であり，行調整処理におけるこの glue の優先度を -2 から $+2$ の間の整数で指定する．⟨*priority*⟩ の省略時の値は 0 であり，範囲外の値が指定されたときの動作は未定義である）．

⟨*ratio*⟩ も省略可能フィールドであり，-1 から $+1$ の実数値をとる．省略時の値は 0 である．

- -1 はこのグルーが「前の文字」由来であることを示す．
- $+1$ はこのグルーが「後の文字」由来であることを示す．
- それ以外の値は，「前の文字」由来のグルーと「後の文字」由来のグルーが混合されていることを示す．

なお，このフィールドの値は differentjfm の値が pleft, pright, paverage の値のときのみ実際に用いられる．

例えば，[6] では，句点と中点の間には，句点由来の二分空きと中点由来の四分空きが挿入されるが，この場合には

- ⟨*width*⟩ には $0.5 + 0.25 = 0.75$ を指定する．
- ⟨*ratio*⟩ には次の値を指定する．

$$-1 \cdot \frac{0.5}{0.5+0.25} + 1 \cdot \frac{0.25}{0.5+0.25} = -\frac{1}{3}$$

end_stretch=⟨*kern*⟩

end_shrink=⟨*kern*⟩ これらのフィールドは省略可能である．優先順位付き行長調整が有効であり，かつ現在の文字クラスの文字が行末に来た時に，行長を詰める調整・伸ばす調整のためにこの文字と行末の間に挿入可能なカーンの大きさを指定する．

■文字クラスの決定　文字クラスの決定は少々複雑である．ここでは例を用いて説明しよう．

たとえば，次の内容を一部に含んだ jfm-test.lua を考えよう：

```
[0] = {
   chars = { '漢', 'ヒ*' },
   align = 'left', left = 0.0, down = 0.0,
   width = 1.0, height = 0.88, depth = 0.12, italic=0.0,
},
[2000] = {
   chars = { '。', '、*', 'ヒ' },
```

```
    align = 'left', left = 0.0, down = 0.0,
    width = 0.5, height = 0.88, depth = 0.12, italic=0.0,
  },
```

句点「。」の幅は二分であるので

```
\jfont\a=file:KozMinPr6N-Regular.otf:jfm=test;+vert
\setbox0\hbox{\a 。\inhibitglue 漢}
\the\wd0
```

20.0pt

では，全角二分 (15.0 pt) とならなければおかしいが，上の実行結果では 20 pt となっている．それは以下の事情によるものである：

1. `vert` feature によって句点が縦書き用のグリフと置き換わる（`luaotfload` による処理）．
2. しかしこのグリフは「文字コード」U+F0000 以降とみなされている（実際にいくらになるかは，フォントによって異なる）．
3. よって，文字クラス 0 とみなされるため，結果として「。」の幅は全角だと認識されてしまう．

一方，「`'、*'`」のようにアスタリスクつきの指定があると，状況は異なってくる．

```
\jfont\a=file:KozMinPr6N-Regular.otf:jfm=test;+vert
\a 漢、\inhibitglue 漢
```

漢 漢

ここで，読点「、」の文字クラスは，以下のようにして決まる．

1. とりあえず句点の時と同じように，`luaotfload` によって縦書き用読点のグリフに置き換わる．
2. 置換後のグリフの「文字コード」は U+F0000 以降であり，そのままでは文字クラスは 0 と判定される．
3. ところが，JFM には「`'、*'`」指定があるので，置換前の横書き用読点のグリフ「、」（文字コードは U+3001）によって文字クラスを判定する．
4. 結果として，上の出力例中の読点の文字クラスは 2000 となる．

なお，置換後のグリフで判定した文字クラスの値が 0 でなければ，そちらをそのまま作用する．

```
\jfont\a=file:KozMinPr6N-Regular.otf:jfm=test;+hwid
\a 漢ﾋひ
```

漢ﾋひ

上の例では，`hwid` feature により，「ヒ」が半角の「ﾋ」に置き換わるが，文字クラスは「ヒ」の属する 0 **ではなく**，「ﾋ」の属する 2000 となる．

■仮想的な文字　上で説明した通り，`chars` フィールド中にはいくつかの「特殊文字」も指定可能である．これらは，大半が pTEX の JFM グルーの挿入処理ではみな「文字クラス 0 の文字」として扱われていた文字であり，その結果として pTEX より細かい組版調整ができるようになっている．以下でその一覧を述べる：

`'boxbdd'` 水平ボックスの先頭と末尾，及びインデントされていない（`\noindent` で開始された）段落の先頭を表す．

`'parbdd'` 通常の（`\noindent` で開始されていない）段落の先頭．

`'jcharbdd'` 和文文字と「その他のもの」（欧文文字，glue，kern 等）との境界．

−1　行中数式と地の文との境界．

■pTEX 用和文用 TFM の移植　以下に，pTEX 用に作られた和文用 TFM を LuaTEX-ja 用に移植する場合の注意点を挙げておく．

- 実際に出力される和文フォントのサイズが design size となる．このため，例えば 1 zw が design size の 0.962216 倍である JIS フォントメトリック等を移植する場合は，
  - JFM 中の全ての数値を 1/0.962216 倍しておく．
  - TEX ソース中で使用するところで，サイズ指定を 0.962216 倍にする．LATEX でのフォント宣言なら，例えば次のように：
    `\DeclareFontShape{JY3}{mc}{m}{n}{<-> s*[0.962216] psft:Ryumin-Light:jfm=jis}{}`
- 上に述べた特殊文字は，`'boxbdd'` を除き文字クラスを全部 0 とする（JFM 中に単に書かなければよい）．
- `'boxbdd'` については，それのみで一つの文字クラスを形成し，その文字クラスに関してはグルー／カーンの設定はしない．
  これは，pTEX では，水平ボックスの先頭・末尾とインデントされていない（`\noindent` で開始された）段落の先頭には JFM グルーは入らないという仕様を実現させるためである．
- pTEX の組版を再現させようというのが目的であれば以上の注意を守れば十分である．
  ところで，pTEX では通常の段落の先頭に JFM グルーが残るという仕様があるので，段落先頭の開き括弧は全角二分下がりになる．全角下がりを実現させるには，段落の最初に手動で `\inhibitglue` を追加するか，あるいは `\everypar` のハックを行い，それを自動化させるしかなかった．
  一方，LuaTEX-ja では，`'parbdd'` によって，それが JFM 側で調整できるようになった．例えば，LuaTEX-ja 同梱の JFM のように，`'boxbdd'` と同じ文字クラスに`'parbdd'` を入れれば全角下がりとなる．

```
1 \jfont\g=file:KozMinPr6N-Regular.otf:jfm
      =test \g
2 \parindent1\zw\noindent{}◆◆◆◆◆
3 \par 「◆◆←二分下がり
4 \par 【◆◆←全角下がり
5 \par 〔◆◆←全角二分下がり
```

◆◆◆◆◆
「◆◆←二分下がり
【◆◆←全角下がり
〔◆◆←全角二分下がり

但し，`\everypar` を利用している場合にはこの仕組みは正しく動かない．そのような例としては箇条書き中の `\item` で始まる段落があり，ltjsclasses では人工的に「`'parbdd'` の意味を持つ」whatsit ノードを作ることによって対処している[*4]

5.4 数式フォントファミリ

TEX は数式フォントを 16 のファミリ[*5]で管理し，それぞれのファミリは 3 つのフォントを持っている：`\textfont`, `\scriptfont` そして `\scriptscriptfont` である．

LuaTEX-ja の数式中での和文フォントの扱いも同様である．表 5 は数式フォントファミリに対する TEX のプリミティブと対応するものを示している．`\fam` と `\jfam` の値の間には関係はなく，適切な設定の下では `\fam` と `\jfam` の両方に同じ値を設定することができる．

5.5 コールバック

LuaTEX 自体のものに加えて，LuaTEX-ja もコールバックを持っている．これらのコールバックには，他のコールバックと同様に `luatexbase.add_to_callback` 関数などを用いることでアクセ

[*4] `no_runtime/ltjsclasses.dtx` を参照されたい．JFM 側で一部の対処ができることにより，jsclasses のように if 文の判定はしていない．

[*5] Omega, Aleph, LuaTEX, そして ε-(u)pTEX では 256 の数式ファミリを扱うことができるが，これをサポートするために plain TEX と LATEX では外部パッケージを読み込む必要がある．

表 5. 和文数式フォントに対する命令

和文フォント	欧文フォント
\jfam $\in [0, 256)$	\fam
jatextfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}	\textfont⟨*fam*⟩=⟨*font_cs*⟩
jascriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}	\scriptfont⟨*fam*⟩=⟨*font_cs*⟩
jascriptscriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}	\scriptscriptfont⟨*fam*⟩=⟨*font_cs*⟩

スすることができる．

luatexja.load_jfm コールバック このコールバックを用いることで JFM を上書きすることができる．このコールバックは新しい JFM が読み込まれるときに呼び出される．

```
function (<table> jfm_info, <string> jfm_name)
  return <table> new_jfm_info
end
```

引数 jfm_info は JFM ファイルのテーブルと似たものが格納されるが，クラス 0 を除いた文字のコードを含んだ chars フィールドを持つ点が異なる．

このコールバックの使用例は ltjarticle クラスにあり，jfm-min.lua 中の'parbdd' を強制的にクラス 0 に割り当てている．

luatexja.define_font コールバック このコールバックと次のコールバックは組をなしており，Unicode 中に固定された文字コード番号を持たない文字を非零の文字クラスに割り当てることができる．このコールバックは新しい和文フォントが読み込まれたときに呼び出される．

```
function (<table> jfont_info, <number> font_number)
  return <table> new_jfont_info
end
```

jfont_info は以下の 2 フィールドを持つ：

size_cache　使用されている JFM の情報が格納されているテーブルで，**このテーブルを書き換えてはならない**．中身はほぼ JFM ファイルに書かれている唯一のテーブルであるが，次のように若干変わっている：

- 各文字クラス i に属する文字達のテーブル [i].chars={⟨*character*⟩，...}は，トップレベルにまとめられ，chars={[⟨*character*⟩]=i，...}という形になっている．
- zw, zh, kanjiskip, xkanjiskip の各フィールドの値は，実際に使われるフォントサイズに合わせた sp ($1\,\mathrm{sp} = 2^{-6}\,\mathrm{pt}$) 単位の長さに変わっている．
- 各文字クラス i の情報を格納したテーブルも，char_type フィールドの下にまとめられている．例えば，文字クラス 1 に属する文字の高さは char_type[1].height で参照できる．
- dir フィールドはこのテーブルにはない．

var　\jfont の呼び出しの際に jfmvar=... で指定された値．

戻り値の new_jfont_info テーブルもこれら 2 つのフィールドを含まなければならないが，それ以外にユーザが勝手にフィールドを付け加えることは自由である．font_number はフォント番号である．

これと次のコールバックの良い使用例は luatexja-otf パッケージであり，JFM 中で Adobe-Japan1 CID の文字を "AJ1-xxx" の形で指定するために用いられている．

luatexja.find_char_class コールバック このコールバックは LuaTEX-ja が chr_code の文字がどの文字クラスに属するかを決定しようとする際に呼び出される．このコールバックで呼び出される関数は次の形をしていなければならない：

```
function (<number> char_class, <table> jfont_info, <number> chr_code)
  if char_class~=0 then return char_class
  else
    ....
    return (<number> new_char_class or 0)
  end
end
```

引数 char_class は LuaTEX-ja のデフォルトルーチンか，このコールバックの直前の関数呼び出しの結果を含んでおり，したがってこの値は 0 ではないかもしれない．さらに，戻り値の new_char_class は char_class が非零のときには char_class の値と同じであるべきで，そうでないときは LuaTEX-ja のデフォルトルーチンを書き換えることになる．

luatexja.set_width コールバック このコールバックは LuaTEX-ja が **JAchar** の寸法と位置を調節するためにその *glyph_node* をカプセル化しようとする際に呼び出される．

```
function (<table> shift_info, <table> jfont_info, <number> char_class)
  return <table> new_shift_info
end
```

引数 shift_info と戻り値の new_shift_info は down と left のフィールドを持ち，これらの値は文字の下／左へのシフト量（スケールド・ポイント単位）である．

良い例が test/valign.lua である．このファイルが読み込まれた状態では，JFM 内で規定された文字クラス 0 の文字における (高さ) : (深さ) の比になるように，実際のフォントの出力上下位置が自動調整される．例えば，

- JFM 側の設定：(高さ) $= 88x$, (深さ) $= 12x$（和文 OpenType フォントの標準値）
- 実フォント側の数値：(高さ) $= 28y$, (深さ) $= 5y$（和文 TrueType フォントの標準値）

となっていたとする．すると，実際の文字の出力位置は，以下の量だけ上にぜらされることとなる：

$$\frac{88x}{88x+12x}(28y+5y)-28y=\frac{26}{25}y=1.04y.$$

6 パラメータ

6.1 \ltjsetparameter 命令

先に述べたように，\ltjsetparameter と \ltjgetparameter は LuaTEX-ja のほとんどのパラメータにアクセスするための命令である．LuaTEX-ja が pTEX のような文法（例えば，\prebreakpenalty`)=10000）を採用しない理由の一つは，LuaTEX のソースにおける hpack_filter コールバックの位置にある．10 節を参照.

\ltjsetparameter と \ltjglobalsetparameter はパラメータを指定するための命令である．これらは ⟨*key*⟩=⟨*value*⟩ のリストを引数としてとる．許されるキーは次の節に記述する．\ltjsetparameter と \ltjglobalsetparameter の違いはスコープの違いのみである．\ltjsetparameter はローカルな指定，\ltjglobalsetparameter はグローバルな指定を行う．これらは他のパラメータ指定と同様に \globaldefs の値に従う．

\ltjgetparameter はパラメータの値を取得するための命令であり，常にパラメータの名前を第一引数にとる．そして，いくつかの場合には加えてさらに引数（例えば文字コード）をとる．

```
\ltjgetparameter{differentjfm},
\ltjgetparameter{autospacing},                paverage, 1, 10000.
\ltjgetparameter{prebreakpenalty}{`) }.
```

\ltjgetparameter の戻り値は常に文字列である．これは tex.write() によって出力しているためで，スペース‘ ’(U+0020) を除いた文字のカテゴリーコードは全て 12 (other) となる．一方，スペースのカテゴリーコードは 10 (space) である．

6.2 パラメーター一覧

以下は \ltjsetparameter に指定することができるパラメータの一覧である．[\cs] は pTEX における対応物を示す．また，それぞれのパラメータの右上にある記号には次の意味がある：

- 記号なし：段落や水平ボックスの終端での値がその段落／水平ボックス全体で用いられる．
- ‘∗’：ローカルなパラメータであり，段落／水平ボックス内のどこででも値を変えることができる．
- ‘†’：指定は常にグローバルになる．

jcharwidowpenalty =⟨*penalty*⟩ [\jcharwidowpenalty] パラグラフの最後の字が孤立して改行されるのを防ぐためのペナルティの値．このペナルティは（日本語の）句読点として扱われない最後の **JAchar** の直後に挿入される．

kcatcode ={⟨*chr_code*⟩,⟨*natural number*⟩} 文字コードが ⟨*chr_code*⟩ の文字が持つ付加的な属性値 (attribute)．現在のバージョンでは，⟨*natural number*⟩ の最下位ビットが，その文字が句読点とみなされるかどうかを表している（上の jcharwidowpenalty の記述を参照）．

prebreakpenalty ={⟨*chr_code*⟩,⟨*penalty*⟩} [\prebreakpenalty] 文字コード ⟨*chr_code*⟩ の **JAchar** が行頭にくることを抑止するために，この文字の前に挿入/追加されるペナルティの量を指定する．

例えば閉じ括弧「〗」は絶対に行頭にきてはならないので，

```
\ltjsetparameter{prebreakpenalty={`〗,10000}}
```

と，最大値の 10000 が標準で指定されている．他にも，小書きのカナなど，絶対禁止というわけではないができれば行頭にはきて欲しくない場合に，0 と 10000 の間の値を指定するのも有用であろう．

postbreakpenalty ={⟨*chr_code*⟩,⟨*penalty*⟩} [\postbreakpenalty] 文字コード ⟨*chr_code*⟩ の **JAchar** が行末にくることを抑止するために，この文字の後に挿入/追加されるペナルティの量を指定する．

pTEX では，\prebreakpenalty, \postbreakpenalty において，

- 一つの文字に対して，pre, post どちらか一つしか指定することができなかった（後から指定した方で上書きされる）．
- pre, post 合わせて 256 文字分の情報を格納することしかできなかった．

という制限があったが，LuaTEX-ja ではこれらの制限は解消されている．

jatextfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩} [TEX の \textfont]

jascriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩} [TEX の \scriptfont]

jascriptscriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩} [TEX の \scriptscriptfont]

yjabaselineshift =⟨*dimen*⟩*

yalbaselineshift =⟨*dimen*⟩* [\ybaselineshift]

jaxspmode ={⟨*chr_code*⟩,⟨*mode*⟩} 文字コードが ⟨*chr_code*⟩ の **JAchar** の前／後ろに xkanjiskip の挿入を許すかどうかの設定．以下の ⟨*mode*⟩ が許される：

0, inhibit xkanjiskip の挿入は文字の前／後ろのいずれでも禁止される．

1, preonly xkanjiskip の挿入は文字の前では許されるが，後ろでは許されない．

2, postonly xkanjiskip の挿入は文字の後ろでは許されるが，前では許されない．

3, allow　xkanjiskip の挿入は文字の前／後ろのいずれでも許される．これがデフォルトの値である．

このパラメータは pTEX の \inhibitxspcode プリミティブと似ているが，互換性はない．

alxspmode={⟨*chr_code*⟩,⟨*mode*⟩} [\xspcode]

文字コードが ⟨*chr_code*⟩ の **ALchar** の前／後ろに xkanjiskip の挿入を許すかどうかの設定．以下の ⟨*mode*⟩ が許される：

0, inhibit　xkanjiskip の挿入は文字の前／後ろのいずれでも禁止される．

1, preonly　xkanjiskip の挿入は文字の前では許されるが，後ろでは許されない．

2, postonly　xkanjiskip の挿入は文字の後ろでは許されるが，前では許されない．

3, allow　xkanjiskip の挿入は文字の前／後ろのいずれでも許される．これがデフォルトの値である．

jaxspmode と alxspmode は共通のテーブルを用いているため，これら 2 つのパラメータは互いの異名となっていることに注意する．

autospacing=⟨*bool*⟩* [\autospacing]

autoxspacing=⟨*bool*⟩* [\autoxspacing]

kanjiskip=⟨*skip*⟩ [\kanjiskip] デフォルトで 2 つの **JAchar** の間に挿入されるグルーである．通常では，pTEX と同じようにフォントサイズに比例して変わることはない．しかし，自然長が \maxdimen の場合は，例外的に和文フォントの JFM 側で指定されている値を採用（こちらはフォントサイズに比例）することになっている．

xkanjiskip=⟨*skip*⟩ [\xkanjiskip] デフォルトで **JAchar** と **ALchar** の間に挿入されるグルーである．kanjiskip と同じように，通常ではフォントサイズに比例して変わることはないが，自然長が \maxdimen の場合が例外である．

differentjfm=⟨*mode*⟩† JFM（もしくはサイズ）が異なる 2 つの **JAchar** の間にグルー／カーンをどのように入れるかを指定する．許される値は以下の通り：

average

both

large

small

pleft

pright

paverage

jacharrange=⟨*ranges*⟩*

kansujichar={⟨*digit*⟩, ⟨*chr_code*⟩} [\kansujichar]

7　その他の命令

7.1　pTEX 互換用命令

以下の命令は pTEX との互換性のために実装されている．そのため，JIS X 0213 には対応せず，pTEX と同じように JIS X 0208 の範囲しかサポートしていない．

\kuten

\jis

\euc

\sjis

\ucs
\kansuji

7.2 \inhibitglue

\inhibitglue は **JAglue** の挿入を抑制する．以下は，ボックスの始めと ‘あ’ の間，‘あ’ と ‘ウ’ の間にグルーが入る特別な JFM を用いた例である．

```
1 \jfont\g=file:KozMinPr6N-Regular.otf:jfm=
    test \g
2 \fbox{\hbox{あウあ\inhibitglue ウ}}
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　ウあウ
あ　1
あ　2
あ　3
あ　office

この例を援用して，\inhibitglue の仕様について述べる．

- \inhibitglue の垂直モード中での呼び出しは意味を持たない．4 行目の入力で有効にならないのは，\inhibitglue の時点では垂直モードであり，\noindent の時点で水平モードになるからである．
- \inhibitglue の（制限された）水平モード中での呼び出しはその場でのみ有効であり，段落の境界を乗り越えない．さらに，\inhibitglue は上の例の最終行のように（欧文における）リガチャとカーニングを打ち消す．これは，\inhibitglue が内部的には「現在のリスト中に whatsit ノードを追加する」ことを行なっているからである．
- \inhibitglue を数式モード中で呼び出した場合はただ無視される．
- LaTeX で LuaTeX-ja を使用する場合は，\inhibitglue の代わりとして \< を使うことができる．

8 LaTeX 2ε 用の命令

8.1 NFSS2 へのパッチ

2.4 節で述べたように，LuaTeX-ja は NFSS2 への日本語パッチである pLaTeX 2ε の plfonts.dtx を単純に取り入れている．便宜のため，ここでは 3.1 節で述べていなかった命令について記述しておく．

\DeclareYokoKanjiEncoding{⟨*encoding*⟩}{⟨*text-settings*⟩}{⟨*math-settings*⟩}

LuaTeX-ja の NFSS2 においては，欧文フォントファミリと和文フォントファミリはそのエンコーディングからのみ作られる．例えば，OT1 と T1 のエンコーディングは欧文フォントファミリに対するものであり，和文フォントファミリはこれらのエンコーディングを持つことはできない．このコマンドは和文フォントファミリ（横書き用）のための新しいエンコーディングを定義する．

\DeclareKanjiEncodingDefaults{⟨*text-settings*⟩}{⟨*math-settings*⟩}

\DeclareKanjiSubstitution{⟨*encoding*⟩}{⟨*family*⟩}{⟨*series*⟩}{⟨*shape*⟩}

\DeclareErrorKanjiFont{⟨*encoding*⟩}{⟨*family*⟩}{⟨*series*⟩}{⟨*shape*⟩}{⟨*size*⟩}

上記 3 つのコマンドはちょうど DeclareFontEncodingDefaults などに対応するものである．

\reDeclareMathAlphabet{⟨*unified-cmd*⟩}{⟨*al-cmd*⟩}{⟨*ja-cmd*⟩}

和文・欧文の数式用フォントファミリを一度に変更する命令を作成する．具体的には，欧文数式用フォントファミリ変更の命令 ⟨*al-cmd*⟩（`\mathrm` 等）と，和文数式用フォントファミリ変更の命令 ⟨*ja-cmd*⟩（`\mathmc` 等）の 2 つを同時に行う命令として ⟨*unified-cmd*⟩ を（再）定義する．実際の使用では ⟨*unified-cmd*⟩ と ⟨*al-cmd*⟩ に同じものを指定する，すなわち，⟨*al-cmd*⟩ で和文側も変更させるようにするのが一般的と思われる．
本命令は

⟨*unified-cmd*⟩`{`⟨*arg*⟩`}` ⟶ （⟨*al-cmd*⟩ **を 1 段展開したもの**）`{`⟨*ja-cmd*⟩ **を 1 段展開したもの**）`{`⟨*arg*⟩`}}`

と定義を行うので，使用には注意が必要である：

- ⟨*al-cmd*⟩, ⟨*ja-cmd*⟩ は既に定義されていなければならない．`\reDeclareMathAlphabet` 後に両命令の内容を再定義しても，⟨*unified-cmd*⟩ の内容にそれは反映されない．
- ⟨*al-cmd*⟩, ⟨*ja-cmd*⟩ に`\@mathrm` などと`@`をつけた命令を指定した時の動作は保証できない．

`\DeclareRelationFont{`⟨*ja-encoding*⟩`}{`⟨*ja-family*⟩`}{`⟨*ja-series*⟩`}{`⟨*ja-shape*⟩`}`
`{`⟨*al-encoding*⟩`}{`⟨*al-family*⟩`}{`⟨*al-series*⟩`}{`⟨*al-shape*⟩`}`

いわゆる「従属欧文」を設定するための命令である．前半の 4 引数で表される和文フォントファミリに対して，そのフォントに対応する「従属欧文」フォントファミリを後半の 4 引数により与える．

`\SetRelationFont`

このコマンドは `\DeclareRelationFont` とローカルな指定であることを除いてほとんど同じである（`\DeclareRelationFont` はグローバル）．

`\userelfont`

現在の欧文フォントエンコーディング／ファミリ／…… を，`\DeclareRelationFont` か `\SetRelationFont` で指定された現在の和文フォントファミリに対応する「従属欧文」フォントファミリに変更する．`\fontfamily` のように，有効にするためには `\selectfont` が必要である．

`\adjustbaseline`

…

`\fontfamily{`⟨*family*⟩`}`

元々の LaTeX 2$_\varepsilon$ におけるものと同様に，このコマンドは現在のフォントファミリ（欧文，和文，**もしくは両方**）を ⟨*family*⟩ に変更する．どのファミリが変更されるかは以下のようにして決定される：

- 現在の和文フォントに対するエンコーディングが ⟨*ja-enc*⟩ であるとしよう．現在の和文フォントファミリは，以下の 2 つの条件のうちの 1 つが満たされているときに ⟨*family*⟩ に変更される：
  - エンコーディング ⟨*ja-enc*⟩ におけるファミリ ⟨*family*⟩ が既に `\DeclareKanjiFamily` によって定義されている．
  - フォント定義ファイル ⟨*ja-enc*⟩⟨*family*⟩`.fd`（ファイル名は全て小文字）が存在する．
- 現在の欧文フォントに対するエンコーディングを ⟨*al-enc*⟩ とする．欧文フォントファミリに対しても，上記の基準が用いられる．
- 上記のいずれもが適用されない，つまり ⟨*family*⟩ が ⟨*ja-enc*⟩ と ⟨*al-enc*⟩ のどちらでも定義されないような場合がある．この場合，代替フォントに用いられるデフォルトのフォントファミリが欧文フォントと和文フォントに用いられる．LaTeX のオリジナルの実装とは異なり，現在のエン

コーディングは ⟨*family*⟩ には設定されないことに注意する．

この節の終わりに，`\SetRelationFont` と `\userelfont` の例を紹介しておこう．`\userelfont` の使用によって，「abc」の部分のフォントが Avant Garde (OT1/pag/m/n) に変わっていることがわかる．

```
\makeatletter
\SetRelationFont{JY3}{\k@family}{m}{n}{OT1}{pag}{m}{n}
  % \k@family: current Japanese font family
\userelfont\selectfont あいうabc
```

あいう abc

9 拡張

9.1 luatexja-fontspec.sty

3.2 節で述べたように，この追加パッケージは `fontspec` パッケージで定義されているコマンドに対応する和文フォント用のコマンドを提供する．オリジナルの `fontspec` での ‘font feature’ に加えて，和文版のコマンドには以下の ‘font feature’ を指定することができる：

`CID=`⟨*name*⟩
`JFM=`⟨*name*⟩
`JFM-var=`⟨*name*⟩
: これら 3 つのキーはそれぞれ `\jfont` に対する `cid`, `jfm`, `jfmvar` キーとそれぞれ対応する．`CID` は下の `NoEmbed` と合わせて用いられたときのみ有効である．`\jfont` プリミティブに対する `cid`, `jfm`, `jfmvar` キーの詳細は 5.1 節と 5.2 節を参照．

`NoEmbed` これを指定することで，PDF に埋め込まれない「名前だけ」のフォントを指定することができる．5.2 節を参照．

なお，`luatexja-fontspec.sty` 読み込み時には和文フォント定義ファイル ⟨*ja-enc*⟩⟨*family*⟩`.fd` は全く参照されなくなる．

9.2 luatexja-otf.sty

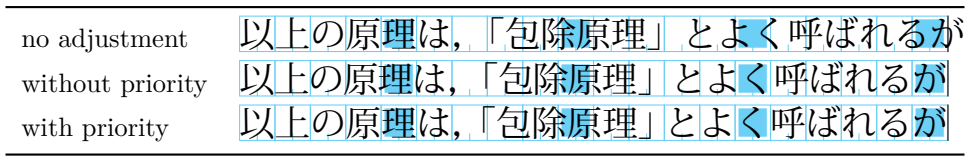
この追加パッケージは Adobe-Japan1 の文字の出力をサポートする．`luatexja-otf.sty` は以下の 2 つの低レベルコマンドを提供する：

`\CID{`⟨*number*⟩`}` CID 番号が ⟨*number*⟩ の文字を出力する．
`\UTF{`⟨*hex_number*⟩`}` 文字コードが（16 進で）⟨*hex_number*⟩ の文字を出力する．このコマンドは `\char"`⟨*hex_number*⟩ と似ているが，下の記述に注意すること．

■注意 `\CID` と `\UTF` コマンドによって出力される文字は以下の点で通常の文字と異なる：

- 常に **JAchar** として扱われる．
- OpenType feature（例えばグリフ置換やカーニング）をサポートするための `luaotfload` パッケージのコードはこれらの文字には働かない．

■JFM への記法の追加 `luatexja-otf.sty` は JFM の記法を拡張する．JFM の `chars` テーブルのエントリとして `'AJ1-xxx'` の形の文字列が使えるようになる．これは Adobe-Japan1 における CID 番号が `xxx` の文字を表す．



Note: the value of kanjiskip is $0\,\mathrm{pt}^{+1/5\,\mathrm{em}}_{-1/5\,\mathrm{em}}$ in this figure, for making the difference obvious.

図 2. 行長調整

9.3 luatexja-adjust.sty

pTEX では，行長調整において優先度の概念が存在しなかったため，図 2 上段における半角分の半端は，図 2 中段のように，鍵括弧周辺の空白と和文間空白 (kanjiskip) の両方によって負担される．しかし，「日本語組版処理の要件」[5] や JIS X 4051 [6] においては，このような状況では半端は鍵括弧周辺の空白のみで負担し，その他の和文文字はベタ組で組まれる（図 2 下段）ことになっている．この追加パッケージは [5] や [6] における規定のような，優先順位付きの行長調整を提供する．詳細な仕様については 14 を参照.

luatexja-adjust.sty は，以下の命令を提供する．これらはすべてグローバルに効力を発揮する．

\ltjdisableadjust 優先順位付きの行長調整を無効化する．
\ltjenableadjust 優先順位付きの行長調整を有効化する．
優先度設定……

第 III 部
実装

10 パラメータの保持

10.1 LuaTEX-ja で用いられる寸法レジスタ，属性レジスタ，whatsit ノード

以下は LuaTEX-ja で用いられる寸法レジスタ (dimension)，属性レジスタ (attribute) のリストである．

\jQ (dimension) \jQ は 1 Q = 0.25 mm と等しい．ここで，‘Q’（もしくは「級」）は日本の写植で用いられる単位である．したがって，この寸法レジスタの値を変更してはならない．
\jH (dimension) 同じく写植で用いられていた単位として「歯」があり，これも 0.25 mm と等しい．\jH は \jQ の別名である．
\ltj@zw (dimension) 現在の和文フォントの「全角幅」を保持する一時レジスタ．
\ltj@zh (dimension) 現在の和文フォントの「全角高さ」(通常，高さと深さの和）を保持する一時レジスタ．
\jfam (attribute) 数式用の和文フォントファミリの現在の番号．
\ltj@curjfnt (attribute) 現在の和文フォントのフォント番号．
\ltj@charclass (attribute) 和文文字の *glyph_node* の文字クラス．
\ltj@yablshift (attribute) スケールド・ポイント (2^{-16} pt) を単位とした欧文フォントのベースラインの移動量．
\ltj@ykblshift (attribute) スケールド・ポイント (2^{-16} pt) を単位とした和文フォントのベース

ラインの移動量.

`\ltj@autospc` (attribute) そのノードで kanjiskip の自動挿入が許されるかどうか.

`\ltj@autoxspc` (attribute) そのノードで xkanjiskip の自動挿入が許されるかどうか.

`\ltj@icflag` (attribute) ノードの「種類」を区別するための属性. 以下のうちのひとつが値として割り当てられる：

italic (1) イタリック補正 (\/) によるグルー. このグルーの由来の区別（`\kern` か`\/` か）は xkanjiskip の挿入過程において必要になる.

packed (2)

kinsoku (3) 和文文字のワードラップ過程において挿入されたペナルティ (*kinsoku*).

from_jfm (6) JFM 由来のグルー／カーン.

kanji_skip (9) kanjiskip のグルー.

xkanji_skip (10) xkanjiskip のグルー.

processed (11) LuaTEX-ja の内部処理によって既に処理されたノード.

ic_processed (12) イタリック補正に由来するグルーであるが，まだ処理されていないもの.

boxbdd (15) ある水平ボックスか段落の最初か最後に挿入されたグルー／カーン.

`\ltj@kcat`i (attribute) i は 7 より小さい自然数. これら 7 つの属性レジスタは，どの文字ブロックが **JAchar** のブロックとして扱われるかを示すビットベクトルを格納する.

さらに，LuaTEX-ja はいくつかの「ユーザ定義の」whatsit ノードを内部処理に用いる. これらの全てのノードは自然数を格納している（したがってノードの `type` は 100 である）.

`inhibitglue` `\inhibitglue` が指定されたことを示すノード. これらのノードの `value` フィールドは意味を持たない.

`stack_marker` LuaTEX-ja のスタックシステム（次の節を参照）のためのノード. これらのノードの `value` フィールドは現在のグループを表す.

`char_by_cid` luaotfload のコールバックによる処理が適用されない和文文字のためのノードで，`value` フィールドにその文字のコードが格納されている. この `user_id` を持つノードはそれぞれが luaotfload のコールバックの処理の**後で** 'glyph_node' に変換される. この `user_id` は luatexja-otf パッケージでのみ使用される.

`begin_par` 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 ...

これらの whatsit ノードは **JAglue** の挿入処理の間に取り除かれる.

10.2 LuaTEX-ja のスタックシステム

■背景 LuaTEX-ja は独自のスタックシステムを持ち，LuaTEX-ja のほとんどのパラメータはこれを用いて保持されている. その理由を明らかにするために，kanjiskip パラメータがスキップレジスタで保持されているとし，以下のコードを考えてみよう：

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

ふがふが. ほ げ ほ げ. ぴよぴよ

6.2 節で述べたように，ある水平ボックスの中で効力を持つ kanjiskip の値は最後に現れた値のみであり，したがってボックス全体に適用される kanjiskip は 5 pt であるべきである. しかし，LuaTEX

の実装のために，この‘5 pt’はどのコールバックからも知ることはできない．tex/packaging.w（これは LuaTEX のソースファイルである）の中に，以下のコードがある：

```
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;
```

unsave が filtered_hpack（これは hpack_filter コールバックが実行されるところである）の**前に**実行されていることに注意する．したがって，上記ソース中で‘5 pt’は unsave のところで捨てられ，hpack_filter からはアクセスすることができない．

■解決法　スタックシステムのコードは Dev-luatex メーリングリストのある投稿[*6]をベースにしている．

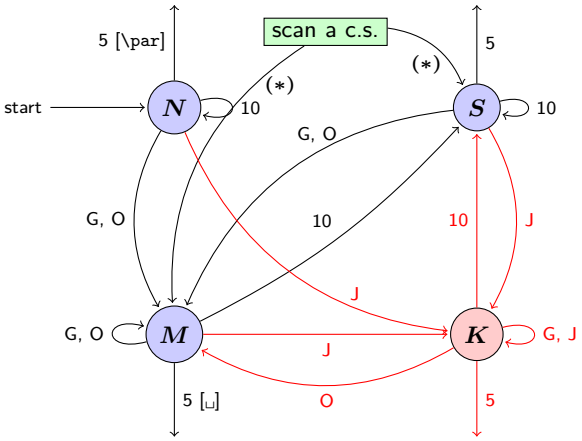
情報を保持するために，2 つの TEX の整数レジスタを用いている：\ltj@@stack でスタックレベル，\ltj@@group@level で最後の代入がなされた時点での TEX のグループレベルを保持している．パラメータは charprop_stack_table という名前のひとつの大きなテーブルに格納される．ここで，charprop_stack_table[i] はスタックレベル i のデータを格納している．もし新しいスタックレベルが \ltjsetparameter によって生成されたら，前のレベルの全てのデータがコピーされる．

上の「背景」で述べた問題を解決するために，LuaTEX-ja ではもう一つの手法を導入する：新しいスタックレベルが生成されようとするとき，type, subtype, value がそれぞれ 44 (*user_defined*), 30112，そして現在のグループレベルである whatsit ノードを現在のリストに付け加える（このノードを *stack_flag* とする）．これにより，ある水平ボックスの中で代入がなされたかどうかを知ることが可能となる．スタックレベルを s，その水平ボックスグループの直後の TEX のグループレベルを t とすると：

- もしその水平ボックスのリストの中に *stack_flag* ノードがなければ，水平ボックスの中では代入は起こらなかったということになる．したがって，その水平ボックスの終わりにおけるパラメータの値はスタックレベル s に格納されている．
- もし値が $t+1$ の *stack_flag* ノードがあれば，その水平ボックスグループの中で代入が起こったことになる．したがって，水平ボックスの終わりにおけるパラメータの値はスタックレベル $s+1$ に格納されている．
- もし *stack_flag* ノードがあるがそれらの値が全て $t+1$ より大きい場合，そのボックスの中で代入が起こったが，それは「より内部の」グループで起こったということになる．したがって，水平ボックスの終わりでのパラメータの値はスタックレベル s に格納されている．

このトリックを正しく働かせるためには，\ltj@@stack と \ltj@@group@level への代入

[*6] [Dev-luatex] tex.currentgrouplevel: Jonathan Sauer による 2008/8/19 の投稿.



G	Beginning of group (usually {) and ending of group (usually }).
J	Japanese characters.
5	*end-of-line* (usually ^^J).
10	space (usually ␣).
O	other characters, whose category code is in $\{3, 4, 6, 7, 8, 11, 12, 13\}$.
[␣], [\par]	emits a space, or \par.

- We omitted about category codes 9 (*ignored*), 14 (*comment*) and 15 (*invalid*) from the above diagram. We also ignored the input like '^^A' or '^^df'.
- When a character whose category code is 0 (*escape character*) is seen by TEX, the input processor scans a control sequence (scan a c.s.). These paths are not shown in the above diagram.
  After that, the state is changed to State S (skipping blanks) in most cases, but to State M (middle of line) sometimes.

図 3. pTEX の入力処理部の状態遷移.

は \globaldefs の値によらず常にローカルでなければならないことに注意する．この問題は \directlua{tex.globaldefs=0}（この代入は常にローカル）を用いることで解決している．

11 和文文字直後の改行

11.1 参考：pTEX の動作

欧文では文章の改行は単語間でしか行わない．そのため，TEX では，（文字の直後の）改行は空白文字と同じ扱いとして扱われる．一方，和文ではほとんどどこでも改行が可能なため，pTEX では和文文字の直後の改行は単純に無視されるようになっている．

このような動作は，pTEX が TEX からエンジンとして拡張されたことによって可能になったことである．pTEX の入力処理部は，TEX におけるそれと同じように，有限オートマトンとして記述することができ，以下に述べるような 4 状態を持っている．

- State N: 行の開始.
- State S: 空白読み飛ばし.
- State M: 行中.
- State K: 行中（和文文字の後）.

また，状態遷移は，図 3 のようになっており，図中の数字はカテゴリーコードを表している．最初の 3 状態は TEX の入力処理部と同じであり，図中から状態 K と「j」と書かれた矢印を取り除けば，TEX の入力処理部と同じものになる．

この図から分かることは，

行が和文文字（とグループ境界文字）で終わっていれば，改行は無視される

ということである．

11.2 LuaTEX-ja の動作

LuaTEX の入力処理部は TEX のそれと全く同じであり，コールバックによりユーザがカスタマイズすることはできない．このため，改行抑制の目的でユーザが利用できそうなコールバックとしては，`process_input_buffer` や `token_filter` に限られてしまう．しかし，TEX の入力処理部をよく見ると，後者も役には経たないことが分かる：改行文字は，入力処理部によってトークン化される時に，カテゴリーコード 10 の 32 番文字へと置き換えられてしまうため，`token_filter` で非標準なトークン読み出しを行おうとしても，空白文字由来のトークンと，改行文字由来のトークンは区別できないのだ．

すると，我々のとれる道は，`process_input_buffer` を用いて LuaTEX の入力処理部に引き渡される前に入力文字列を編集するというものしかない．以上を踏まえ，LuaTEX-ja における「和文文字直後の改行抑制」の処理は，次のようになっている：

> 各入力行に対し，**その入力行が読まれる前の内部状態で**以下の 3 条件が満たされている場合，LuaTEX-ja は U+FFFFF 番の文字[*7]を末尾に追加する．よって，その場合に改行は空白とは見做されないこととなる．
>
> 1. `\endlinechar` の文字[*8]のカテゴリーコードが 5 (end-of-line) である．
> 2. U+FFFFF のカテゴリーコードが 14 (comment) である．
> 3. 入力行は次の「正規表現」にマッチしている：
>
> $$(\text{any char})^*(\textbf{JAchar})\big(\{\text{catcode} = 1\} \cup \{\text{catcode} = 2\}\big)^*$$

この仕様は，前節で述べた pTEX の仕様にできるだけ近づけたものとなっている．最初の条件は，`verbatim` 系環境などの日本語対応マクロを書かなくてすませるためのものである．しかしながら，完全に同じ挙動が実現できたわけではない．差異は，次の例が示すように，和文文字の範囲を変更した行の改行において見られる：

```
1 \ltjsetparameter{autoxspacing=false}
2 \ltjsetparameter{jacharrange={-6}}xあ
3 y\ltjsetparameter{jacharrange={+6}}zあ
4 u
```

xyzあ u

もし pTEX とまったく同じ挙動を示すならば，出力は「x yzあu」となるべきである．しかし，実際には上のように異なる挙動となっている．

- 2 行目は「あ」という和文文字で終わる（2 行目を処理する前の時点では，「あ」は和文文字扱いである）ため，直後の改行文字は無視される．
- 3 行目は「あ」という欧文文字で終わる（2 行目を処理する前の時点では，「あ」は欧文文字扱いである）ため，直後の改行文字は空白に置き換わる．

このため，トラブルを避けるために，和文文字の範囲を`\ltjsetparameter` で編集した場合，その行はそこで改行するようにした方がいいだろう．

[*7] この文字はコメント文字として扱われるように LuaTEX-ja 内部で設定をしている．

[*8] 普通は，改行文字（文字コード 13 番）である．

12　JFM グルーの挿入，kanjiskip と xkanjiskip

12.1　概要

LuaTEX-ja における **JAglue** 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．

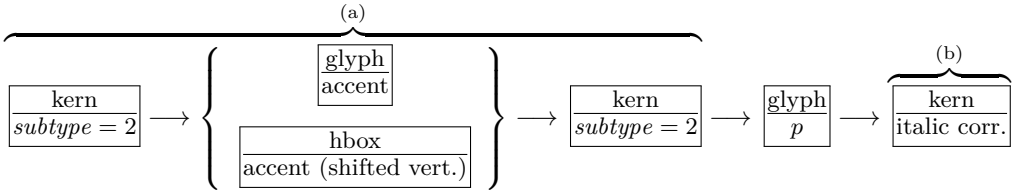
しかし，LuaTEX-ja では，水平ボックスへのパッケージングや行分割前に全ての **JAglue**，即ち JFM グルー・xkanjiskip・kanjiskip の 3 種類を一度に挿入することになっている．これは，LuaTEX において欧文の合字・カーニング処理がノードベースになったことに対応する変更である．

LuaTEX-ja における **JAglue** 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．

12.2　「クラスタ」の定義

定義 1．クラスタは以下の形のうちのどれかひとつをとる連続的なノードのリストである：

1. その `\ltj@icflag` の値が $[3, 15)$ に入るノードのリスト．これらのノードはある既にパッケージングされた水平ボックスから `\unhbox` でアンパックされたものである．その *id* は *id_pbox* である．
2. インライン数式でその境界に 2 つの *math_node* を含むもの．その *id* は *id_math* である．
3. *glpyh_node p* とそれに関係するノード：
   (1) *p* のイタリック補正のためのカーン．
   (2) `\accent` による *p* に付随したアクセント．



   id は *glyph_node* が和文文字を表すかどうかによって *id_jglyph*，もしくは *id_glyph* となる．
4. ボックス様のノード，つまり水平ボックス，垂直ボックス，罫線 (`\vrule`)，そして *unset_node*．その *id* は垂直に移動していない水平ボックスならば *id_hlist*，そうでなければ *id_box_like* となる．
5. グルー，subtype が 2 (*accent*) ではないカーン，そして任意改行．その *id* はそれぞれ *id_glue*, *id_kern*，そして *id_disc* である．

以下では *Np*, *Nq*, *Nr* でクラスタを表す．

■*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，*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，'ffi' 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てたどり着いた *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，TEX 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ので，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　以上にない *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 **JAchar** 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 **JAchar** 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 **ALchar**.

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．

12.3　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 *Np* 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，*Np* 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 *Np* 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 *Np* 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 *Np* 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 *Np* 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
- *Np* 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 *Nq* とおくと，*Nq* と「文字コード`'boxbdd'` の文字」の間に入るグルー／カーンを，*Nq* の直後に挿入するのみである．

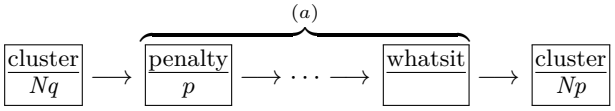
一方．前者（段落）の場合は，リストの末尾は常に`\penalty10000` と，`\parfillskip` 由来のグルーが存在する．よって，最後のクラスタ *Np* はこの`\parfillskip` 由来のグルーとなり，実質的な中身の最後はその 1 つ前のクラスタ *Nq* となる．

1. まず *Nq* 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 **JAchar** であり，かつその文字の kcatcode が偶数である

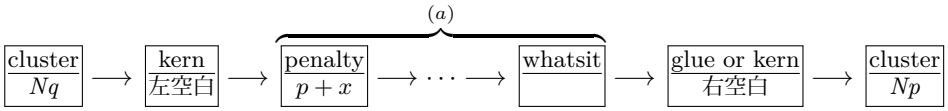
ような最後のクラスタの直前にあるものたちである[*9].

12.4 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タ，Nq と Np 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 クラスタ間は次のようになる：



以後，**典型的な例として，クラスタ Nq と Np 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），代わりに kanjiskip が挿入されることとなる．次へ．
2. Nq と Np 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，決まっていればそれを採用．
3. 1. でも 2. でもない場合は，Nq と Np が違う JFM/jfmvar/サイズである．この場合，まず

$$\begin{aligned} gb &:= (Nq \text{ と「使用フォントが } Nq \text{ のそれと同じで，} \\ &\qquad \text{文字コードが } Np \text{ のそれの文字」との間に入るグルー／カーン}) \\ ga &:= (\text{「使用フォントが } Np \text{ のそれと同じで，} \\ &\qquad \text{文字コードが } Nq \text{ のそれの文字」と } Np \text{ との間に入るグルー／カーン}) \end{aligned}$$

として，前側の文字の JFM を使った時の空白（グルー／カーン）と，後側の文字の JFM を使った時のそれを求める．

gb, ga それぞれに対する $\langle ratio \rangle$ の値を d_b, d_a とする．

- ga と gb の両方が未定義であるならば，JFM 由来のグルーは挿入されず，kanjiskip を採用することとなる．どちらか片方のみが未定義であるならば，次のステップでその未定義の方は長さ 0 の kern で，$\langle ratio \rangle$ の値は 0 であるかのように扱われる．
- diffrentjfm の値が pleft, pright, paverage のとき，$\langle ratio \rangle$ の指定に従って比例配分を行う．JFM 由来のグルー／カーンは以下の値となる：

$$f\left(\frac{1-d_b}{2}gb+\frac{1+d_b}{2}ga, \frac{1-d_a}{2}gb+\frac{1+d_a}{2}ga\right)$$

[*9] 大雑把に言えば，kcatcode が奇数であるような **JAchar** 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．

ここで．$f(x,y)$ は

$$f(x,y)=\begin{cases} x & \text{if diffrentjfm} = \texttt{pleft};\\ y & \text{if diffrentjfm} = \texttt{pright};\\ (x+y)/2 & \text{if diffrentjfm} = \texttt{paverage}; \end{cases}.$$

- differentmet がそれ以外の値の時は，$\langle ratio\rangle$ の値は無視され，JFM 由来のグルー／カーンは以下の値となる：

$$f(gb,ga)$$

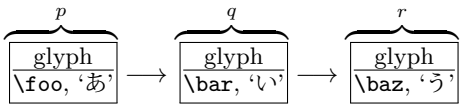
ここで．$f(x,y)$ は

$$f(x,y)=\begin{cases} \min(x,y) & \text{if diffrentjfm} = \texttt{small};\\ \max(x,y) & \text{if diffrentjfm} = \texttt{large};\\ (x+y)/2 & \text{if diffrentjfm} = \texttt{average};\\ x+y & \text{if diffrentjfm} = \texttt{both}; \end{cases}.$$

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タをなす）．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．

kanjiskip [K]　上の [M] において空白が定まらなかった場合，以下で定めた量「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 \maxdimen $=(2^{30}-1)\,\mathrm{sp}$ 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　「左空白」は過去のバージョンでは定義していたが，このバージョンでは挿入は一切行われない（機能自体削除している）．しかし，仕様は流動的であり，将来復活する可能性もあるため，マニュアル中の記述は今のところ極力変更しない．

■禁則用ペナルティの挿入　まず，

$a :=$ (Nq^{*10}の文字に対する postbreakpenalty の値) + (Np^{*11}の文字に対する prebreakpenalty の値)

表 6. JFM グルーの概要.

Np ↓	和文 A	和文 B	欧文	箱	glue	kern
和文 A	$\frac{\mathsf{E}\quad \mathsf{M}\to\mathsf{K}}{\mathsf{PN}}$	$\frac{—\ \mathsf{O_A}\to\mathsf{K}}{\mathsf{PN}}$	$\frac{—\ \mathsf{O_A}\to\mathsf{X}}{\mathsf{PN}}$	$\frac{—\qquad \mathsf{O_A}}{\mathsf{PA}}$	$\frac{—\qquad \mathsf{O_A}}{\mathsf{PN}}$	$\frac{—\qquad \mathsf{O_A}}{\mathsf{PS}}$
和文 B	$\frac{\mathsf{E}\quad \mathsf{O_B}\to\mathsf{K}}{\mathsf{PA}}$	$\frac{—\qquad \mathsf{K}}{\mathsf{PS}}$	$\frac{—\qquad \mathsf{X}}{\mathsf{PS}}$			
欧文	$\frac{\mathsf{E}\quad \mathsf{O_B}\to\mathsf{X}}{\mathsf{PA}}$	$\frac{—\qquad \mathsf{X}}{\mathsf{PS}}$				
箱	$\frac{\mathsf{E}\qquad \mathsf{O_B}}{\mathsf{PA}}$					
glue	$\frac{\mathsf{E}\qquad \mathsf{O_B}}{\mathsf{PN}}$					
kern	$\frac{\mathsf{E}\qquad \mathsf{O_B}}{\mathsf{PS}}$					

ここで $\frac{\mathsf{E}\quad \mathsf{M}\to\mathsf{K}}{\mathsf{PN}}$ は次の意味である：

1.「右空白」を決めるために，LuaTEX-ja はまず「JFM 由来 [M]」の方法を試みる．これが失敗したら，LuaTEX-ja は「kanjiskip [K]」の方法を試みる．
2. Nq と Np の間の「左空白」は「line-end [E]」の方法で決定される．
3. LuaTEX-ja は 2 つのクラスタの間の禁則処理用のペナルティを調節するために「P-normal [PN]」の方法を採用する．

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：そ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．

12.5　その他の場合

本節の内容は表 6 にまとめてある．

[*11] 厳密にはそれぞれ $Nq.tail$，$Np.head$.

■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，以下で定めた量を「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 `\maxdimen` $= (2^{30}-1)\,\mathrm{sp}$ 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，*Nq*, *Np*（**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　*Nq* が**欧文**で，*Np* 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．
- *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 *Np* 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　Nq が**和文 A** で，Np 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 Np の種類によって異なる．なお，$Np.head$ は無意味であるから，「$Np.head$ に対する prebreakpenalty の値」は 0 とみなされる．言い換えれば，

$$a := (Nq^{*12}\text{の文字に対する postbreakpenalty の値}).$$

箱　Np 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　Np 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　Np 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，Nq と Np 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]　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．

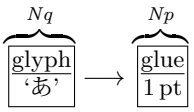
(a) 部分にペナルティが存在していない場合，LuaTEX-ja は Nq と Np の間の行分割を可能にしようとする．そのために，以下の場合に a をもつ *penalty_node* を作って「右空白」の（もし未定義なら Np の）直前に挿入する：

- 「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」がカーンとしてきっちり定義されている時．

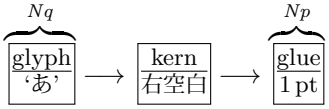
P-suppress [PS]　Nq と Np の間の (a) 部分にペナルティがあれば，P-normal [PN] と同様に，それらの各ノードにおいてペナルティ値を a だけ増加させる．

(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

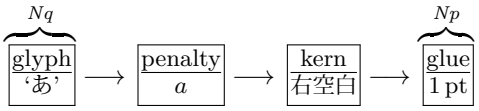


のような状況を考える．このとき，a，即ち「あ」の postbreakpenalty がいかなる値であっても，この 2 クラスタ間は最終的に



(1)

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[*13]．

■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，

$$a := (Np\text{[*14]の文字に対する prebreakpenalty の値}).$$

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．

■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\\
2 \hbox{あ．}A\\
3 あ．A
```

あ．A
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

[*13] 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．

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．

13 listings パッケージへの対応

listings パッケージが，そのままでは日本語をまともに出力できないことはよく知られている．きちんと整形して出力するために，listings パッケージは内部で「ほとんどの文字」をアクティブにし，各文字に対してその文字の出力命令を割り当てている [2]．しかし，そこでアクティブにする文字の中に，和文文字がないためである．pTEX 系列では，和文文字をアクティブにする手法がなく，jlisting.sty というパッチ [3] を用いることで無理やり解決していた．

LuaTEX-ja では，process_input_buffer コールバックを利用することで，「各行に出現する U+0080 以降の文字に対して，それらの出力命令を前置する」という方法をとっている．これにより，（入力には使用されていないかもしれない）和文文字をもすべてアクティブ化する手間もなく，見通しが良い実装になっている．

LuaTEX-ja で利用される listings パッケージへのパッチ lltjp-listings.sty は，listings.sty と LuaTEX-ja を読み込んでおけば，\begin{document} の箇所において自動的に読み込まれるので，通常はあまり意識する必要はない．

■文字種 listings パッケージの内部では，大雑把に言うと

1. 識別子として使える文字 (“letter”, “digit”) たちを集める．
2. letter でも digit でもない文字が現れた時に，収集した文字列を（必要なら修飾して）出力する．
3. 今度は逆に，letter でない文字たちを letter が現れるまで集める．
4. letter が出現したら集めた文字列を出力する．
5. 1. に戻る．

という処理が行われている．これにより，識別子の途中では行分割が行われないようになっている．直前の文字が識別子として使えるか否かは \lst@ifletter というフラグに格納されている．

さて，日本語の処理である．殆どの和文文字の前後では行分割が可能であるが，その一方で括弧類や音引きなどでは禁則処理が必要なことから，lltjp-listings.sty では，直前が和文文字であるかを示すフラグ \lst@ifkanji を新たに導入した．以降，説明のために以下のように文字を分類する：

	Letter	Other	Kanji	Open	Close
\lst@ifletter	T	F	T	F	T
\lst@ifkanji	F	F	T	T	F
意図	識別子中の文字	その他欧文文字	殆どの和文文字	開き括弧類	閉じ括弧類

なお，本来の listings パッケージでの分類 “digit” は，出現状況によって，上の表の Letter と Other のどちらにもなりうる．また，Kanji と Close は \lst@ifletter と \lst@ifkanji の値が一致しているが，これは間違いではない．

例えば，Letter の直後に Open が来た場合を考える．文字種 Open は和文開き括弧類を想定しているので，Letter の直後では行分割が可能であることが望ましい．そのため，この場合では，すでに収集されている文字列を出力することで行分割を許容するようにした．

同じように，$5 \times 5 = 25$ 通り全てについて書くと，次のようになる：

		後ろ側の文字				
		Letter	Other	Kanji	Open	Close
直	Letter	収集		出力		収集
前	Other	出力	収集		出力	収集
文	Kanji		出力			収集
字	Open			収集		
種	Close		出力			収集

上の表において，

- 「出力」は，それまでに集めた文字列を出力（≒ここで行分割可能）を意味する．
- 「収集」は，後側の文字を，現在収集された文字列に追加（行分割不可）を意味する．

■和文文字扱いとなる文字　`listings` パッケージにおいて和文文字と扱われる（前に述べた Kanji, Open，あるいは「閉じ括弧類」分類）か否かは，通常の **JAchar**/**Alchar** の範囲の設定（jacharrange パラメータ，4.1 節を参照）に従って行われる：

- （U+0080 以降の）**ALchar** は，すべて Letter 扱いである．
- （U+0080 以降の）**JAchar** については，以下の順序に従って文字種を決める：
  1. prebreakpenalty が 0 以上の文字は Open 扱いである．
  2. postbreakpenalty が 0 以上の文字は Close 扱いである．
  3. 上の 2 条件のどちらにも当てはまらなかった文字は，Kanji 扱いである．

なお，半角カナ (U+FF61–U+FF9F) 以外の **JAchar** は欧文文字 2 文字分の幅をとるものとみなされる．半角カナは欧文文字 1 文字分の幅となる．

これらの文字種決定は，実際に `lstlisting` 環境などの内部で文字が出てくるたびに行われる．

14　和文の行長補正方法

`luatexja-adjust.sty` で提供される優先順位付きの行長調整の詳細を述べる．大まかに述べると，次のようになる．

- 通常の TEX の行分割方法に従って，段落を行分割する．この段階では，行長に半端が出た場合，その半端分は xkanjiskip，kanjiskip，JFM グルーの全てで（優先順位なく）負担される．
- その後，`post_linebreak_filter` callback を使い，**段落中の各行ごとに**，行末文字の位置を調整したり，優先度付きの行長調整を実現するためにグルーの伸縮度を調整する．`luatexja-adjust.sty` の作用は，この callback を追加するだけであり，この章の残りでは callback での処理について解説する．

■準備：合計伸縮量の計算　グルーの伸縮度（`plus` や `minus` で指定されている値）には，有限値の他に，`fi`，`fil`，`fill`，`filll` という 4 つの無限大レベル（後ろの方ほど大きい）があり，行の調整に `fi` などの**無限大レベルの伸縮度が用いられている場合は，その行に対しての処理を中止**する．

よって，以降，問題にしている行の行長調整は伸縮度が有限長のグルーを用いて行われているとして良い．まず，段落中の行中のグルーを

- 下のどれにも該当しないグルー
- JFM グルー（優先度別にまとめられる）
- 和欧文間空白 (xkanjiskip)

- 和文間空白 (kanjiskip)

の $1+1+5+1=8$ つに類別し，それぞれの種別ごとに許容されている伸縮度の合計を計算する．また，行長と自然長との差の絶対値を計算し，それを *total* とおく．

14.1 行末文字の位置調整

まず，行末が文字クラス n の **JAchar** であった場合，それを動かすことによって，**JAglue** が負担する調整量を少なくしようとする．この行末文字の左右の移動可能量は，JFM 中にある文字クラス n の定義の `end_stretch`, `end_shrink` フィールドに全角単位の値として記述されている．

例えば，行末文字が句点「。」であり，そこで用いられている JFM 中に

```
[2] = {
  chars = { '。', ... }, width = 0.5, ...,
  end_stretch = 0.5, end_shrink = 0.5,
},
```

という指定があった場合，この行末の句点は

- 通常の TEX の行分割処理で「半角以上の詰め」が行われていた場合，この分の行中の **JAglue** の負担を軽減するため，行末の句点を半角だけ右に移動する（ぶら下げ組を行う）．
- 通常の TEX の行分割処理で「半角以上の空き」が行われていた場合，逆に行末句点を半角左に移動させる（見た目的に全角取りとなる）．
- 以上のどちらでもない場合，行末句点の位置調整は行わない．

となる．

行末文字を移動した場合，その分だけ *total* の値を引いておく．

14.2 グルーの調整

total の分だけが，行中のグルーの伸縮度に応じて負担されることになる．……

参考文献

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

[2] C. Heinz, B. Moses. The Listings Package.

[3] Thor Watanabe. Listings - MyTeXpert. http://mytexpert.sourceforge.jp/index.php?Listings

[4] 乙部厳己, min10 フォントについて. http://argent.shinshu-u.ac.jp/~otobe/tex/files/min10.pdf

[5] W3C Japanese Layout Task Force (ed), Requirements for Japanese Text Layout (W3C Working Group Note), 2011, 2012. http://www.w3.org/TR/jlreq/
日本語訳の書籍版：W3C 日本語組版タスクフォース（編），『W3C 技術ノート 日本語組版処理の要件』，東京電機大学出版局，2012.

[6] 日本工業規格 (Japanese Industrial Standard) JIS X 4051, 日本語文書の組版方法 (Formatting rules for Japanese documents), 1993, 1995, 2004.

付録 A　Package versions used in this document

This document was typeset using the following packages:

```
geometry.sty              2010/09/12 v5.6 Page Geometry
ifvtex.sty                2010/03/01 v1.5 Detect VTeX and its facilities (HO)
ifxetex.sty               2010/09/12 v0.6 Provides ifxetex conditional
luatexja-adjust.sty       2013/05/14
expl3.sty                 2013/10/13 v4597 L3 Experimental code bundle wrapper
l3names.sty               2012/12/07 v4346 L3 Namespace for primitives
l3bootstrap.sty           2013/07/28 v4581 L3 Experimental bootstrap code
l3basics.sty              2013/07/28 v4581 L3 Basic definitions
l3expan.sty               2013/08/17 v4584 L3 Argument expansion
l3tl.sty                  2013/09/16 v4592 L3 Token lists
l3seq.sty                 2013/07/28 v4581 L3 Sequences and stacks
l3int.sty                 2013/08/02 v4583 L3 Integers
l3quark.sty               2013/07/21 v4564 L3 Quarks
l3prg.sty                 2013/08/25 v4587 L3 Control structures
l3clist.sty               2013/07/28 v4581 L3 Comma separated lists
l3token.sty               2013/08/25 v4587 L3 Experimental token manipulation
l3prop.sty                2013/07/28 v4581 L3 Property lists
l3msg.sty                 2013/07/28 v4581 L3 Messages
l3file.sty                2013/10/13 v4596 L3 File and I/O operations
l3skip.sty                2013/07/28 v4581 L3 Dimensions and skips
l3keys.sty                2013/07/28 v4581 L3 Experimental key-value interfaces
l3fp.sty                  2013/07/09 v4521 L3 Floating points
l3box.sty                 2013/07/28 v4581 L3 Experimental boxes
l3coffins.sty             2012/09/09 v4212 L3 Coffin code layer
l3color.sty               2012/08/29 v4156 L3 Experimental color support
l3luatex.sty              2013/07/28 v4581 L3 Experimental LuaTeX-specific functions
l3candidates.sty          2013/07/24 v4576 L3 Experimental additions to l3kernel
amsmath.sty               2013/01/14 v2.14 AMS math features
amstext.sty               2000/06/29 v2.01
amsgen.sty                1999/11/30 v2.0
amsbsy.sty                1999/11/29 v1.2d
amsopn.sty                1999/12/14 v2.01 operator names
array.sty                 2008/09/09 v2.4c Tabular extension package (FMi)
tikz.sty                  2010/10/13 v2.10 (rcs-revision 1.76)
pgf.sty                   2008/01/15 v2.10 (rcs-revision 1.12)
pgfrcs.sty                2010/10/25 v2.10 (rcs-revision 1.24)
everyshi.sty              2001/05/15 v3.00 EveryShipout Package (MS)
pgfcore.sty               2010/04/11 v2.10 (rcs-revision 1.7)
graphicx.sty              1999/02/16 v1.0f Enhanced LaTeX Graphics (DPC,SPQR)
graphics.sty              2009/02/05 v1.0o Standard LaTeX Graphics (DPC,SPQR)
trig.sty                  1999/03/16 v1.09 sin cos tan (DPC)
pgfsys.sty                2010/06/30 v2.10 (rcs-revision 1.37)
xcolor.sty                2007/01/21 v2.11 LaTeX color extensions (UK)
pgfcomp-version-0-65.sty 2007/07/03 v2.10 (rcs-revision 1.7)
pgfcomp-version-1-18.sty 2007/07/23 v2.10 (rcs-revision 1.1)
pgffor.sty                2010/03/23 v2.10 (rcs-revision 1.18)
pgfkeys.sty
pict2e.sty                2011/04/05 v0.2y Improved picture commands (HjG,RN,JT)
multienum.sty
float.sty                 2001/11/08 v1.3d Float enhancements (AL)
booktabs.sty              2005/04/14 v1.61803 publication quality tables
multicol.sty              2011/06/27 v1.7a multicolumn formatting (FMi)
listings.sty              2013/08/26 1.5b (Carsten Heinz)
lstmisc.sty               2013/08/26 1.5b (Carsten Heinz)
showexpl.sty              2013/03/21 v0.3k Typesetting example code (RN)
calc.sty                  2007/08/22 v4.3 Infix arithmetic (KKT,FJ)
ifthen.sty                2001/05/26 v1.1c Standard LaTeX ifthen package (DPC)
varwidth.sty              2009/03/30 ver 0.92;  Variable-width minipages
hyperref.sty              2012/11/06 v6.83m Hypertext links for LaTeX
```

```
hobsub-hyperref.sty     2012/05/28 v1.13 Bundle oberdiek, subset hyperref (HO)
hobsub-generic.sty      2012/05/28 v1.13 Bundle oberdiek, subset generic (HO)
hobsub.sty              2012/05/28 v1.13 Construct package bundles (HO)
intcalc.sty             2007/09/27 v1.1 Expandable calculations with integers (HO)
etexcmds.sty            2011/02/16 v1.5 Avoid name clashes with e-TeX commands (HO)
kvsetkeys.sty           2012/04/25 v1.16 Key value parser (HO)
kvdefinekeys.sty        2011/04/07 v1.3 Define keys (HO)
pdfescape.sty           2011/11/25 v1.13 Implements pdfTeX's escape features (HO)
bigintcalc.sty          2012/04/08 v1.3 Expandable calculations on big integers (HO)
bitset.sty              2011/01/30 v1.1 Handle bit-vector datatype (HO)
uniquecounter.sty       2011/01/30 v1.2 Provide unlimited unique counter (HO)
letltxmacro.sty         2010/09/02 v1.4 Let assignment for LaTeX macros (HO)
hopatch.sty             2012/05/28 v1.2 Wrapper for package hooks (HO)
xcolor-patch.sty        2011/01/30 xcolor patch
atveryend.sty           2011/06/30 v1.8 Hooks at the very end of document (HO)
atbegshi.sty            2011/10/05 v1.16 At begin shipout hook (HO)
refcount.sty            2011/10/16 v3.4 Data extraction from label references (HO)
hycolor.sty             2011/01/30 v1.7 Color options for hyperref/bookmark (HO)
auxhook.sty             2011/03/04 v1.3 Hooks for auxiliary files (HO)
kvoptions.sty           2011/06/30 v3.11 Key value format for package options (HO)
url.sty                 2006/04/12  ver 3.3  Verb mode for urls, etc.
rerunfilecheck.sty      2011/04/15 v1.7 Rerun checks for auxiliary files (HO)
amsthm.sty              2004/08/06 v2.20
luatexja-otf.sty        2013/05/14
luatexja-ajmacros.sty   2013/05/14
luatexja-preset.sty     2013/10/28 Japanese font presets
luatexja-fontspec.sty   2013/08/17 fontspec support of LuaTeX-ja
fontspec.sty            2013/05/20 v2.3c Font selection for XeLaTeX and LuaLaTeX
xparse.sty              2013/10/13 v4597 L3 Experimental document command parser
fontspec-patches.sty    2013/05/20 v2.3c Font selection for XeLaTeX and LuaLaTeX
fixltx2e.sty            2006/09/13 v1.1m fixes to LaTeX
fontspec-luatex.sty     2013/05/20 v2.3c Font selection for XeLaTeX and LuaLaTeX
fontenc.sty
xunicode.sty            2011/09/09 v0.981 provides access to latin accents and many other
                        characters in Unicode lower plane
amssymb.sty             2013/01/14 v3.01 AMS font symbols
amsfonts.sty            2013/01/14 v3.01 Basic AMSFonts support
metalogo.sty            2010/05/29 v0.12 Extended TeX logo macros
lltjp-fontspec.sty      2013/05/14 Patch to fontspec for LuaTeX-ja
lltjp-xunicode.sty      2013/05/14 Patch to xunicode for LuaTeX-ja
lltjp-listings.sty      2013/05/14 Patch to listings for LuaTeX-ja
epstopdf-base.sty       2010/02/09 v2.5 Base part for package epstopdf
grfext.sty              2010/08/19 v1.1 Manage graphics extensions (HO)
nameref.sty             2012/10/27 v2.43 Cross-referencing by name of section
gettitlestring.sty      2010/12/03 v1.4 Cleanup title references (HO)
```