# LuaTEX-ja パッケージ

LuaTEX-ja プロジェクトチーム

20261122.0（2026 年 11 月 22 日）

# 目次


## 第 I 部　ユーザーズマニュアル　5

**1　はじめに　5**

1.1　背景 . . . . . 5
1.2　pTEX からの主な変更点 . . . . . 5
1.3　用語と記法 . . . . . 7
1.4　プロジェクトについて . . . . . 8

**2　使い方　9**

2.1　インストール . . . . . 9
2.2　注意点 . . . . . 10
2.3　plain TEX で使う . . . . . 10
2.4　LATEX で使う . . . . . 11

**3　フォントの変更　14**

3.1　plain TEX and LATEX $2_\varepsilon$ . . . . . 14
3.2　luatexja-fontspec パッケージ . . . . . 15
3.3　和文フォントのプリセット設定 . . . . . 16
3.4　\CID, \UTF と otf パッケージのマクロ . . . . . 17

**4　パラメータの変更　17**

4.1　**JAchar** の範囲 . . . . . 17
4.2　kanjiskip と xkanjiskip . . . . . 21
4.3　xkanjiskip の挿入設定 . . . . . 21
4.4　ベースラインの移動 . . . . . 22
4.5　禁則処理関連パラメータと OpenType 機能 . . . . . 23

## 第 II 部　リファレンス　25

**5　LuaTEX-ja における \catcode　25**

5.1　予備知識：pTEX と upTEX における \kcatcode . . . . . 25
5.2　LuaTEX-ja の場合 . . . . . 25
5.3　制御綴中に使用出来る JIS 非漢字の違い . . . . . 25

**6　縦組　26**

6.1　サポートする組方向 . . . . . 27
6.2　異方向のボックス . . . . . 27
6.3　組方向の取得 . . . . . 29
6.4　実装の比較 . . . . . 30

**7 プリミティブの再定義 30**
7.1 再定義の抑制 . . . 32

**8 フォントメトリックと和文フォント 33**
8.1 \jfont 命令 . . . 33
8.2 \tfont 命令 . . . 38
8.3 標準和文フォント・JFM の変更 . . . 40
8.4 psft プリフィックス . . . 40
8.5 JFM ファイルの構造 . . . 41
8.6 数式フォントファミリ . . . 47
8.7 コールバック . . . 47

**9 パラメータ 50**
9.1 \ltjsetparameter . . . 50
9.2 \ltjgetparameter . . . 52
9.3 \ltjsetparameter の代替 . . . 53

**10 plain でも LaTeX でも利用可能なその他の命令 54**
10.1 pTeX 互換用命令 . . . 54
10.2 \inhibitglue, \disinhibitglue . . . 55
10.3 \ltjfakeboxbdd, \ltjfakeparbegin . . . 55
10.4 \insertxkanjiskip, \insertkanjiskip . . . 56
10.5 \ltjdeclarealtfont . . . 57
10.6 \ltjalchar と \ltjjachar . . . 58

**11 LaTeX 2ε 用の命令 58**
11.1 LaTeX 2ε 下での和文フォントの読み込み . . . 58
11.2 NFSS2 へのパッチ . . . 59
11.3 \fontfamily コマンドの詳細 . . . 62
11.4 \DeclareTextSymbol 使用時の注意 . . . 63
11.5 \strutbox . . . 63
11.6 \verb と xkanjiskip . . . 64

**12 expl3 形式の命令 64**

**13 拡張パッケージ 65**
13.1 luatexja-fontspec . . . 65
13.2 luatexja-otf . . . 67
13.3 luatexja-adjust . . . 68
13.4 luatexja-ruby . . . 72
13.5 lltjext . . . 73
13.6 luatexja-preset . . . 76

**第 III 部　実装** **83**

**14　パラメータの保持** **83**
14.1 LuaTEX-ja で用いられるレジスタと whatsit ノード . . . . . 83
14.2 LuaTEX-ja のスタックシステム . . . . . 85
14.3 スタックシステムで使用される関数 . . . . . 86
14.4 パラメータの拡張 . . . . . 87

**15　和文文字直後の改行** **88**
15.1 参考：pTEX の動作 . . . . . 88
15.2 LuaTEX-ja の動作 . . . . . 89
15.3 濁点・半濁点付き仮名の正規化→ luaotfload v3.19 以降ではそちらで . . . . . 90

**16　JFM グルーの挿入，kanjiskip と xkanjiskip** **91**
16.1 概要 . . . . . 91
16.2 「クラスタ」の定義 . . . . . 91
16.3 段落／ hbox の先頭や末尾 . . . . . 93
16.4 概観と典型例：2 つの「和文 A」の場合 . . . . . 94
16.5 その他の場合 . . . . . 98

**17　ベースライン補正の方法** **101**
17.1 `yoffset` フィールド . . . . . 101
17.2 **ALchar** の補正 . . . . . 102

**18　listings パッケージへの対応** **102**
18.1 注意 . . . . . 102
18.2 文字種 . . . . . 103

**19　和文の行長補正方法** **105**
19.1 行末文字の位置調整（行分割後の場合） . . . . . 106
19.2 行末文字の位置調整（行分割での考慮） . . . . . 106
19.3 グルーの調整 . . . . . 107

**20　複数フォントの「合成」（未完）** **108**

**21　LuaTEX-ja におけるキャッシュ** **108**
21.1 キャッシュの使用箇所 . . . . . 108
21.2 内部命令 . . . . . 109

**22　縦組の実装** **110**
22.1 direction whatsit . . . . . 110
22.2 *dir_box* . . . . . 111
22.3 縦組用字形の取得 . . . . . 114

参考文献 116

# 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 拡張の主要部分（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，（pTEX はエンジン拡張であったのに対し）LuaTEX-ja は Lua コードと TEX マクロを用いて全てを実装していなければならないため，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．より詳細については第 III 部など本文書の残りを参照．

**■命令の名称**　例えば pTEX で追加された次のようなプリミティブ

```
\kanjiskip=10pt  \dimen0=kanjiskip
\tbaselineshift=0.1zw
\dimen0=\tbaselineshift
```

```
\prebreakpenalty`あ=100
\ifydir ... \fi
```

は LuaTEX-ja には存在しない．LuaTEX-ja では以下のように記述することになる．

```
\ltjsetparameter{kanjiskip=10pt}  \dimen0=\ltjgetparameter{kanjiskip}
\ltjsetparameter{talbaselineshift=0.1\zw}
\dimen0=\ltjgetparameter{talbaselineshift}
\ltjsetparameter{prebreakpenalty={`あ,100}}
\ifnum\ltjgetparameter{direction}=4 ... \fi
```

特に注意してほしいのは，pTEX で追加された zw と zh という単位は LuaTEX-ja では使用できず，**\zw, \zh と制御綴の形にしないといけない**という点である[*1]．

**■和文文字直後の改行**　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．詳しくは 15 章を参照．

**■和文関連の空白**　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．

- LuaTEX の内部での合字の扱いは「ノード」を単位として行われるようになっている（例えば，of{}fice で合字は抑制されない）．それに合わせ，**JAglue** の挿入処理もノード単位で実行される．
- 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：**

  ちょ{}っと　　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 (\hbox) を間に挟めばよい：

  ちょ\hbox{}っと

- 処理中では，2 つの和文フォントは，実フォントが異なるだけの場合には同一視される．

**■組方向**　バージョン 20150420.0 からは，不安定ながらも LuaTEX-ja における縦組みをサポートしている．なお，LuaTEX 本体も Ω 流の組方向をサポートしているが，それとは全くの別物であることに注意してほしい．特に，異なった組方向のボックスを扱う場合には \wd, \ht, \dp 等の仕様が pTEX と異なるので注意．詳細は第 6 章を参照．

**■\discretionary**　\discretionary 内に直接和文文字を記述することは，pTEX においても想定されていなかった感があるが．LuaTEX-ja においても想定していない．和文文字をどうしても使いたい場合は \hbox で括ること．

[*1] 別のパッケージやユーザが \zw, \zh を書き換えてしまうことに対応するため，LuaTEX-ja 20200127.0 以降では，\ltj@zw, \ltj@zh がそれぞれ \zw, \zh のコピーとして定義されている．

**■ギリシャ文字・キリル文字と ISO 8859-1 の記号**　標準では，LuaTEX-ja はギリシャ文字やキリル文字を和文フォントを使って組む．ギリシャ語などを本格的に組むなどこの状況が望ましくない場合，プリアンブルに

```
\ltjsetparameter{jacharrange={-2,-3}}
```

を入れると上記種類の文字は欧文フォントを用いて組まれるようになる．詳しい説明は 4.1 節を参照してほしい．

また，¶, § といった ISO 8859-1 の上位領域と JIS X 0208 の共通部分の文字はバージョン 20150906.0 から標準で欧文扱いとなった．LaTeX $2_\varepsilon$ 2017/01/01 以降では標準で TU エンコーディングの Latin Modern フォントが使われるので，特に何もせずソース中にそのまま記述してもこれらの文字が出力される．和文扱いで出力するには `\ltjjachar`§` のように `\ltjjachar` 命令を使えばよい．

**■\verb や verbatim 環境中の和欧文間空白**　バージョン 20260420.0 以降では，`\verb` や `verbatim` 環境[*2]内部では和欧文間空白を入れないようにした．これは，「入力通りに組まれる」状況で，和欧文間空白と明示的な空白を混同してしまうことを避けるためである．ただ，以下に示すように，`\verb` の前後や，単なる `\texttt{}`の内部には和欧文間空白は入る．

```
1\texttt{漢字12 34漢 字5}漢\\
1\verb|漢字12 34漢 字5|漢\\
1\verb*|漢字12 34漢 字5|漢
\begin{verbatim}
漢字12 34漢 字
\end{verbatim}
```

1 漢字 12  34 漢  字 5 漢
1 漢字12  34漢  字5 漢
1 漢字12␣34漢␣字5 漢

漢字12  34漢  字

バージョン 20260420.0 より前の挙動に戻したければ，LuaTEX-ja 読込み後にプリアンブルに次のように記述すれば良い：

```
\makeatletter\let\ltj@verb@fontsetup\relax\makeatother
```

## 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：

- 文字は次の 2 種類に分けられる．この類別は固定されているものではなく，ユーザが後から変更可能である（4.1 節を参照）．
  - **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**: ラテン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．
- 下線つきローマン体で書かれた語（例：prebreakpenalty）は日本語組版用のパラメータを表し，これらは `\ltjsetparameter` 命令のキーとして用いられる．
- 下線なしサンセリフ体の語（例：fontspec）は LaTeX のパッケージやクラスを表す．

[*2] `.dtx` 内の `macrocode` 環境や，fancyvrb パッケージの提供する `Verbatim` 環境も含む．実装方法については 11.6 節を参照．

- 本ドキュメントでは，自然数は 0 から始まる．（TEX で扱える）自然数全体の集合は $\omega$ と表記する．

## 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．

- https://github.com/luatexja/luatexja/wiki（日本語）
- https://github.com/luatexja/luatexja/wiki/Home(en)（英語）
- https://github.com/luatexja/luatexja/wiki/Home(zh)（中国語）

本プロジェクトは GitHub 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 パッケージの動作には次のパッケージ類が必要である.

- LuaTEX 1.10.0 (or later)（DVI 出力 (`\outputmode=0`) は対応していない.）
- recent luaotfload (v3.1 or later recommended)
- adobemapping (Adobe cmap and pdfmapping files)
- etoolbox（LATEX $2_\varepsilon$ 下で LuaTEX-ja を使う場合）
- ltxcmds, pdftexcmds
- fontspec v2.9e (or later)
- **原ノ味フォント** (https://github.com/trueroad/HaranoAjiFonts)
  LuaTEX-ja の最低限の動作には原ノ味明朝 Regular (HaranoAjiMincho-Regular) と原ノ味角ゴシック Medium (HaranoAjiGothic-Medium) があれば十分である.

現在，LuaTEX-ja は CTAN (`macros/luatex/generic/luatexja`) に収録されている他，以下のディストリビューションにも収録されている：

- MiKTEX (`luatexja.tar.lzma`)
- TEX Live (`texmf-dist/tex/luatex/luatexja`)

これらのディストリビューションは原ノ味フォントも収録している（TEX Live, MiKTEX では `haranoaji`）.

**■HarfBuzz と LuaTEX-ja**　現時点では，HarfBuzz の機能を用いたときの LuaTEX-ja の使用は十分にテストされていない．エラーが発生せずにタイプセットできるかもしれないが，特に縦組時や `\CID` などで意図しない結果となりうる可能性が大きい．特に，**`Renderer=Harfbuzz` 等（fontspec）や `mode=harf` 指定（それ以外）を通じて和文フォントに対して HarfBuzz を用いることは推奨しない**.

**■手動インストール方法**

1. ソース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 リポジトリを次のコマンドでクローンする：

     ```
     $ git clone https://github.com/luatexja/luatexja.git
     ```

   - `master` ブランチのスナップショット（`zip` 形式）をダウンロードする：
     https://github.com/luatexja/luatexja/archive/refs/heads/master.zip.

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 リポジトリをクローン」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
```

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

pTEX からの変更点として，1.2 節も熟読するのが望ましい．ここでは一般的な注意点を述べる．

- 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．コードを調整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．また，和文フォントを読み込むために多量のメモリを消費することにも注意が必要である．
- 現在 RTT と Identity-V CMap を用いた縦組の実装を試験中である（6.4 小節を参照）．この新たな実装は「縦組時でもあっても pdf からのテキスト抽出がまともになる」という利点があるが，荒削りであるのでまだ標準では有効になっていない．有効にするには，LuaTEX-ja **読み込み前**[*3]に，次を実行する：

```
\directlua{luatexja_cmapidv = true}
```

## 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 12 個の和文フォントが定義される：

[*3] LATEX で使う場合には，\documentclass より前に記述するのが安全である．

| 組方向 | 字体 | フォント名 | “10 pt” | “7 pt” | “5 pt” |
|---|---|---|---|---|---|
| 横組 | 明朝体 | 原ノ味明朝 Regular | \tenmin | \sevenmin | \fivemin |
| | **ゴシック体** | 原ノ味角ゴシック Medium | \tengt | \sevengt | \fivegt |
| 縦組 | 明朝体 | 原ノ味明朝 Regular | \tentmin | \seventmin | \fivetmin |
| | **ゴシック体** | 原ノ味角ゴシック Medium | \tentgt | \seventgt | \fivetgt |

  - 標準和文フォントや JFM を原ノ味フォントから別のものに置き換えるには，\ltj@stdmcfont 等を luatexja.sty 読み込み**前**に定義すればよい．8.3 節を参照．
  - 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 2ε 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．

- 和文フォントのエンコーディングとしては，横組用には JY3，縦組用には JT3 が用いられる．
- 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 | 命令 | | ファミリ名 |
|---|---|---|---|
| 明朝体 | \textmc{...} | {\mcfamily ...} | \mcdefault |
| **ゴシック体** | \textgt{...} | {\gtfamily ...} | \gtdefault |
| （タイプライタ体と合わせる和文） | — | | \jttdefault |

  \jttdefault は \verb や verbatim 環境中の和文文字に使われる和文フォントファミリであり，標準値は \mcdefault，つまり明朝体である[*4]．和文フォントファミリ（のみ）を \jttdefault に切り替える命令は準備していない．
- 標準では，次のフォントが用いられる：

| 字体 | ファミリ | \mdseries | \bfseries | スケール |
|---|---|---|---|---|
| 明朝体 | mc | 原ノ味明朝 Regular | 原ノ味角ゴシック Medium | 0.962216 |
| **ゴシック体** | gt | 原ノ味角ゴシック Medium | 原ノ味角ゴシック Medium | 0.962216 |

  明朝・ゴシックどちらのファミリにおいても，太字 (\bfseries) のフォントはゴシック体中字

[*4] ltjsclasses を使用したり，また luatexja-fontspec や luatexja-preset パッケージを match オプションを指定して読み込んだときは，単なる \ttfamily によっても和文フォントが \jttdefault に変更される．また，これらのクラスファイルやパッケージは \jttdefault を \gtdefault（ゴシック体）に再定義する．

(\gtfamily\mdseries) で使われるフォントと同じであることに注意．また，どちらのファミリでもイタリック体・スラント体は定義されない．

- 和文の太字を表すシリーズ名は，（元々の Computer Modern が太字に bx を用いていたことから）伝統的に bx (Bold Extended) が使われてきた．しかし，太字にシリーズ b を使うフォントも増えてきたため，バージョン 20180616.0 以降では和文の太字として bx, b の両方を扱えるようにした．
- バージョン 20181102.0 以降では，disablejfam オプションを LuaTEX-ja 読み込み時に指定できるようになった．このオプションは，数式モード中に直に和文文字を書けるようにするための LATEX へのパッチを読み込まない．
  disablejfam のない状況では，以前と同様に和文文字を数式モード中に直に書くことができる（但し 14 ページの記述も参照）．その際には明朝体 (mc) で出力される．
- beamer クラスを既定のフォント設定で使う場合，既定欧文フォントがサンセリフなので，既定和文フォントもゴシック体にしたいと思うかもしれない．その場合はプリアンブルに次を書けばよい：

  ```
  \renewcommand{\kanjifamilydefault}{\gtdefault}
  ```

- pLATEX と同様に，mc, gt 両ファミリには「従属欧文」書体が定義されている．これらは \userelfont を \selectfont（や，その他の「実際に」フォントを変更する命令）の前で実行することにより使うことができる．
  pLATEX では標準の欧文フォントは OT1 エンコーディングの Computer Modern Roman (cmr) であったが，2017 年以降の LuaLATEX では TU エンコーディングの Latin Modern Roman (lmr) に変更されている．そのため，前段落で述べた「従属欧文」も，Latin Modern Roman に設定している．

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「pLATEX $2_\varepsilon$ 新ドキュメントクラス」）に対応するものとして，ltjclasses[*5]，ltjsclasses[*6]がそれぞれ LuaTEX-ja 標準で用意されている．

元々の jsclasses では本文のフォントサイズを設定するのに \mag プリミティブが使われていたが，LuaTEX では PDF 出力時の \mag のサポートが廃止された．そのため，ltjsclasses では nomag*オプション[*7]が標準で有効になっており，これを使って本文フォントサイズの設定を行っている．しかし，この nomag*オプションでは（バージョン 20180121.0 より前で unicode-math パッケージ使用時に起きたように）予想外の意図しない現象に遭遇する危険がある．そのような場合は \documentclass において nomag オプションを指定してほしい．

**■脚注とボトムフロートの出力順序**　オリジナルの LATEX では脚注はボトムフロートの上に出力され，また \raggedbottom 命令でページの高さが不揃いであることを許した場合には脚注の下端の垂直位置もページに応じて変わるようになっている．一方，日本語の組版では脚注はボトムフロートの下に来るのが一般的であるので，pLATEX ではそのように変更されており，さらに \raggedbottom 命令を

[*5] 横組用は ltjarticle.cls, ltjbook.cls, ltjreport.cls であり，縦組用は ltjtarticle.cls, ltjtbook.cls, ltjtreport.cls である．
[*6] ltjsarticle.cls, ltjsbook.cls, ltjsreport.cls, ltjskiyou.cls.
[*7] jsclasses や，八登崇之氏による BXjscls クラスにおける同名のオプションと同じ．上記クラスは TEX コードのみで実装しているが，ltjsclasses では Lua コードも用いている．

実行した後でも脚注は常にページの下端に固定されるようになっている．

**脚注とボトムフロートの順序，及び \raggedbottom 時の脚注の垂直位置は，LuaTeX-ja パッケージはとくに変更しない**[*8]が，これら 2 点については，以下のような制御手段がある：

**LaTeX 2ε 2024-12-01 以前** footmisc パッケージか stfloats パッケージを利用する．例えば stfloats パッケージを利用して脚注をボトムフロートの下に組む場合は，次のようにする：

```
\usepackage{stfloats}\fnbelowfloat
```

また，\raggedbottom 時の脚注の垂直位置は，\iffnfixbottom という真偽値で制御する．

**偽 (\fnfixbottomfalse) の場合** LaTeX 標準と同じく，本文と脚注の間の空白は \skip\footins のみ．従って脚注の垂直位置はページにより変動する．

**真 (\fnfixbottomtrue) の場合** pLaTeX や footmisc パッケージを bottom オプションで読み込んだ場合のように，脚注は常にページの下端に固定される．

**LaTeX 2ε 2025-06-01 以降** LaTeX 2ε 本体の提供する build/column/outputbox ソケットを利用する．このソケットには 1 つのプラグを選択して挿すことができるが，「脚注を下端に」という目的で使えるプラグは以下の通りである：

- space-floats-footnotes プラグ：余分の空白→ボトムフロート→脚注の順に出力する．\usepackage[bottomfloats,belowfloats]{footmisc}の場合の組み方．
- floats-space-footnotes プラグ：ボトムフロート→余分の空白→脚注の順に出力する．\usepackage[bottom]{footmisc}の場合の組み方．
- floats-footnotes-platex プラグ：pLaTeX の挙動を再現したもの．

例えば floats-footnotes-platex プラグを挿す場合は，次のようにする：

```
\AssignSocketPlug{build/column/outputbox}{floats-footnotes-platex}
```

**「互換クラス」ltjclasses, ltjsclasses の挙動** pLaTeX と合わせるために，以下のようにしている：

**LaTeX 2ε 2024-12-01 以前** \iffnfixbottom を真にし，かつ \fnfixbottomtrue\usepackage{stfloats}\fnbelowfloatを実行する．

**LaTeX 2ε 2025-06-01 以降** build/column/outputbox ソケットに floats-footnotes-platex プラグを挿す．

**■縦組での geometry パッケージ** pLaTeX の縦組用標準クラスファイルでは geometry パッケージを利用することは出来ず，

```
! Incompatible direction list can't be unboxed.
\@begindvi ->\unvbox \@begindvibox
                                \global \let \@begindvi \@empty
```

というようなエラーが発生することが知られている．LuaTeX-ja では，ltjtarticle.cls といった縦組クラスの下でも geometry パッケージが利用できるようにパッチ lltjp-geometry パッケージを自動的に当てている．

なお，lltjp-geometry パッケージは pTeX 系列でも明示的に読み込むことによって使用可能である．詳細や注意事項は lltjp-geometry.pdf を参照のこと．

[*8] LuaTeX-ja を「欧文の中にちょっとだけ日本語を入れる」ため使うことも考慮したためである

# 3　フォントの変更

## 3.1　plain TEX and LATEX 2ε

**■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` 命令や `\tfont` 命令を直接用いる．8.1 節を参照．

**■LATEX 2ε (NFSS2)**　LATEX で用いる際には，pLATEX 2ε (`plfonts.dtx`) 用のフォント選択機構の大部分を流用している．

| | エンコーディング | ファミリ | シリーズ | シェイプ | 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` |

- `\fontfamily`, `\fontseries`, `\fontshape` は欧文・和文フォント両方の属性を変更しようとする．もちろん，それらを実際に反映させるには手動で `\selectfont` を実行する必要がある．
  なお，`\fontshape{`⟨*shape*⟩`}`は常に欧文フォントのシェイプを設定するが，もしも現在の和文フォントファミリ・シリーズで要求されたシェイプが利用不能だった場合には，和文フォントのシェイプは変更しない．詳細は 11.2 節を参照すること．
- 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 11.2 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．以前の実装では `\DeclareFontFamily` を用いても問題は生じなかったが，現在の実装ではそうはいかない．
- 和文フォントのシェイプを定義するには，通常の `\DeclareFontShape` を使えば良い：

```
\DeclareFontShape{JY3}{mc}{b}{n}{<-> s*HaranoAjiMincho--Bold:jfm=ujis;-kern}{}
    % Harano Aji Mincho Bold
```

  仮名書体を使う場合など，複数の和文フォントを組み合わせて使いたい場合は 10.5 節の `\ltjdeclarealtfont` と，その LATEX 版の `\DeclareAlternateKanjiFont`（11.2 節）を参照せよ．

**■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}\,\}$.
```

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

$$y=(x-1)^2+2 \quad よって \quad y>0$$

$5\in 素:=\{\,p\in\mathbb{N}:\text{$p$ 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，

表 1. luatexja-fontspec で定義される命令

| | | | | |
|---|---|---|---|---|
| **和文** | \jfontspec | \setmainjfont | \setsansjfont | \setmonojfont |
| **欧文** | \fontspec | \setmainfont | \setsansfont | \setmonofont |
| **和文** | \newjfontfamily | \renewjfontfamily | \setjfontfamily | \providejfontfamily |
| **欧文** | \newfontfamily | \renewfontfamily | \setfontfamily | \providefontfamily |
| **和文** | \newjfontface | \renewjfontface | \setjfontface | \providejfontface |
| **欧文** | \newfontface | \renewfontface | \setfontface | \providefontface |
| **和文** | \defaultjfontfeatures | \addjfontfeatures | | |
| **欧文** | \defaultfontfeatures | \addfontfeatures | | |

- 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」が単なる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 \[ y=(x-1)^2+2\quad
4   \mathrel{\mbox{よって}}\quad y>0 \]
5 $5\in 素:=\{\,p\in\mathbb N:\text{$p$ is a
    prime}\,\}$.
```

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

$$y = (x-1)^2 + 2 \quad \text{よって} \quad y > 0$$

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

なお 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 8.6 節を参照のこと．

**既に記述した通り，disablejfam オプションを指定して LuaTEX-ja を読み込んだ場合は，$素$ のように直接和文文字を数式モード中に記述することはできなくなる．\mbox，あるいは amsmath パッケージの提供する \text 命令などを使うことになる．**

## 3.2 luatexja-fontspec パッケージ

fontspec パッケージは，LuaTEX・XETEX において TrueType・OpenType フォントを容易に扱うためのパッケージであり，このパッケージを読み込んでおけば Unicode による各種記号の直接入力もできるようになる．LuaTEX-ja では和文と欧文を区別しているため，fontspec パッケージの機能は欧文フォントに対してのみ有効なものとなっている．

LuaTEX-ja 上において，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場合は luatexja-fontspec パッケージを読み込む：

```
\usepackage[⟨options⟩]{luatexja-fontspec}
```

このパッケージは自動で 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，表 1 の「和文」行に示した命令を fontspec パッケージの元のコマンド（「欧文」行）に対応するものとして定義している：

luatexja-fontspec パッケージのオプションは以下の通りである：

`match`
: このオプションが指定されると，「pLaTeX $2_\varepsilon$ 新ドキュメントクラス」のように `\rmfamily`, `\textrm{...}`, `\sffamily` 等が欧文フォントだけでなく和文フォントも変更するようになる．

`pass=`⟨*options*⟩
: fontspec パッケージに渡すオプション ⟨*options*⟩ を指定する．本オプションは時代遅れである．

`scale=`⟨*float*⟩
: 欧文に対する和文の比率を手動で上書きするときに使用する．標準では
  - `\Cjascale` が定義されている場合[*9]は，それを用いる．
  - `\Cjascale` が未定義の場合は，luatexja-fontspec 読み込み時の和文フォントから自動計算される．

上記にないオプションは全て fontspec パッケージに渡される．例えば，下の 2 行は同じ意味になる：

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

これらの和文用のコマンドではフォント内のペアカーニング情報は既定では使用されない，言い換えれば kern feature は既定では無効となっている．これは以前のバージョンの LuaTeX-ja との互換性のためである（8.1 節を参照）．

以下に `\jfontspec` の使用例を示す．

```
\jfontspec[CJKShape=NLC]{HaranoAjiMincho-Regular}
JIS~X~0213:2004→辻鯵\par
\jfontspec[CJKShape=JIS1990]{HaranoAjiMincho-Regular}
JIS~X~0208-1990→辻鯵\par
\jfontspec[CJKShape=JIS1978]{HaranoAjiMincho-Regular}
JIS~C~6226-1978→辻鯵
```

JIS X 0213:2004 →辻鯵
JIS X 0208-1990 →辻鯵
JIS C 6226-1978 →辻鯵

## 3.3 和文フォントのプリセット設定

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．オプションや各プリセットの詳細については 13.6 節を参照して欲しい．現時点では以下のプリセットが定義されている：

`haranoaji`, `hiragino-pro`, `hiragino-pron`, `ipa`, `ipa-hg`, `ipaex`, `ipaex-hg`, `kozuka-pr6`, `kozuka-pr6n`, `kozuka-pro`, `moga-mobo`, `moga-mobo-ex`, `bizud`, `morisawa-pr6n`, `morisawa-pro`, `ms`, `ms-hg`, `noembed`, `noto-otc`, `noto-otf`, `noto`, `noto-jp`, `sourcehan`, `sourcehan-jp`, `ume`, `yu-osx`, `yu-win`, `yu-win10`

例えば，本ドキュメントでは luatexja-preset パッケージを

```
\usepackage[haranoaji]{luatexja-preset}
```

として読み込み，原ノ味フォントを使うことを指定している．

[*9] LuaTeX-ja が用意しているクラスファイル (ltjclasses, ltjsclasses) を使う場合はこちらに当てはまる．

### 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（luatexja-otf という別のパッケージとして）実装した．

```
森\UTF{9DD7}外と\CID{13966}田百\UTF{9592}とが
\UTF{9AD9}島屋に\\
\CID{7652}飾区の\CID{13706}野家，
\CID{1481}城市，葛西駅，\\
高崎と\CID{8705}\UTF{FA11}，濱と\ajMayuHama\\
\aj半角{カタカナ}\ajKakko3\ajMaruYobi{2}%
\ajLig{令和}\ajLig{〇問}\ajJIS
```

森鷗外と内田百閒とが髙島屋に
葛飾区の吉野家，葛城市，葛西駅，
高崎と髙﨑，濱と濵
ｶﾀｶﾅ⑶㈪㈱㈹㋀

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 3 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 `\rubyfamily` によって扱えるようになる．

`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，11.2 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．これらのパラメータを設定・取得するためにはそれぞれ `\ltjsetparameter` と `\ltjgetparameter` を用いる．

### 4.1 JAchar の範囲

LuaTeX-ja は，Unicode の `U+0080–U+10FFFF` の空間を 1 番から 217 番までの**文字範囲**に分割している．区分けは `\ltjdefcharrange` を用いることで（グローバルに）変更することができ，例えば，次は追加漢字面 (SIP) にある全ての文字と「漢」を「100 番の文字範囲」に追加する．

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

各文字はただ一つの文字範囲に所属することができる．例えば，SIP 内の文字は全て LuaTeX-ja の

表 2. 文字範囲 8 に指定されている文字.

| | | | |
|---|---|---|---|
| § (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 |

表 3. 文字範囲 1 に指定されている Unicode ブロック.

| | | | |
|---|---|---|---|
| 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 | | |

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 内の文字は 100 番に属すようになり，4 番からは除かれる．

**ALchar** と **JAchar** の区別は文字範囲ごとに行われる．これは jacharrange パラメータによって編集できる．以下は LuaTEX-ja の初期設定であり，次の内容を設定している：

- 1 番，4 番，5 番，8 番の文字範囲に属する文字は **ALchar**.
- 2 番，3 番，6 番，7 番，9 番の文字範囲に属する文字は **JAchar**.

```
\ltjsetparameter{jacharrange={-1, +2, +3, -4, -5, +6, +7, -8, +9}}
```

jacharrange パラメータの引数は非零の整数のリストである．リスト中の負の整数 $-n$ は「文字範囲 $n$ に属する文字は **ALchar** として扱う」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．

なお，U+0000–U+007F は常に **ALchar** として扱われる（利用者が変更することは出来ない）.

**■文字範囲の初期値**　LuaTEX-ja では 9 つの文字範囲を予め定義しており，これらは以下のデータに基づいて決定している．

- Unicode 12.0 のブロック.
- Adobe-Japan1-7 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 9 つの文字範囲について記述する．添字のアルファベット「J」「A」は，その文字範囲内の文字が **JAchar** か **ALchar** かを表している．これらの初期設定は PXbase バンドルで定義されている prefercjk と類似のものであるが，8 ビットフォント使用時のトラブルを防ぐために U+0080–U+00FF の文字は全部 **ALchar** としている．なお，U+0080 以降でこれら 9 つの文字範囲に属さない文字は，217 番の文字範囲に属することになっている．

**範囲 8$^{A}$** ISO 8859-1 の上位領域（ラテン 1 補助）と JIS X 0208 の共通部分．文字のリストは表 2 を参照.

**範囲 1$^{A}$** ラテン文字のうち，Adobe-Japan1-7 との共通部分があるもの．この範囲は表 3 で示した Unicode のブロックのうち**範囲 8 を除いた**部分で構成されている.

**範囲 2$^{J}$** 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）には，これらの文字の一部が含まれている.

表 4. 文字範囲 3 に指定されている Unicode ブロック.

| | | | |
|---|---|---|---|
| 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. Math Symbols-B | U+2B00–U+2BFF | Misc. Symbols and Arrows |

表 5. 文字範囲 9 に指定されている文字.

| | | | |
|---|---|---|---|
| (U+2002) | En space | (U+2010) | Hyphen |
| - (U+2011) | Non-breaking hyphen | – (U+2013) | En dash |
| — (U+2014) | Em dash | ― (U+2015) | Horizontal bar |
| ‖ (U+2016) | Double vertical line | ‘ (U+2018) | Left single quotation mark |
| ’ (U+2019) | Right single quotation mark | ‚ (U+201A) | Single low-9 quotation mark |
| “ (U+201C) | Left double quotation mark | ” (U+201D) | Right double quotation mark |
| „ (U+201E) | Double low-9 quotation mark | † (U+2020) | Dagger |
| ‡ (U+2021) | Double dagger | • (U+2022) | Bullet |
| ‥ (U+2025) | Two dot leader | … (U+2026) | Horizontal ellipsis |
| ‰ (U+2030) | Per mille sign | ′ (U+2032) | Prime |
| ″ (U+2033) | Double prime | ‹ (U+2039) | Single left-pointing angle quot. |
| › (U+203A) | Single right-pointing angle quot. | ※ (U+203B) | Reference mark |
| ‼ (U+203C) | Double exclamation mark | ‾ (U+203E) | Overline |
| ‿ (U+203F) | Undertie | ⁂ (U+2042) | Asterism |
| ⁄ (U+2044) | Fraction slash | ⁇ (U+2047) | Double question mark |
| ⁈ (U+2048) | Question exclamation mark | ⁉ (U+2049) | Exclamation question mark |
| ⁑ (U+2051) | Two asterisks aligned vertically | | |

- U+0370–U+03FF: Greek and Coptic
- U+0400–U+04FF: Cyrillic
- U+1F00–U+1FFF: Greek Extended

**範囲 3**[J] 記号類．ブロックのリストは表 4 に示してある．

**範囲 9**[J] Unicode の「一般句読点」ブロック (U+2000–U+206F) と Adobe-Japan1-7 の共通部分．この文字範囲は表 5 に示した文字で構成される．

**範囲 4**[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,
  "2C00-"2E7F, "4DC0-"4DFF, "A4D0-"A95F, "A980-"ABFF, "E000-"F8FF,
  "FB00-"FE0F, "FE20-"FE2F, "FE70-"FEFF, "10000-"1AFFF, "1B170-"1F0FF,
  "1F300-"1FFFF, ... (and characters in U+2000–U+206F which are not in range 9)
} % 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 6 に示す．

**範囲 7**[J] CJK 言語で用いられる文字のうち，Adobe-Japan1-7 に含まれていないもの．ブロックのリス

表 6. 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 Ext-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+FF00–U+FFEF | Halfwidth and Fullwidth Forms |
| U+1B000–U+1B0FF | Kana Supplement | U+1B100–U+1B12F | Kana Extended-A |
| U+1F100–U+1F1FF | Enclosed Alphanumeric Supp. | U+1F200–U+1F2FF | Enclosed Ideographic Supp. |
| U+20000–U+2FFFF | (Supp. Ideographic Plane) | U+30000–U+3FFFF | (Tert. Ideographic Plane) |
| U+E0100–U+E01EF | Variation Selectors Supp. | | |

表 7. 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+A960–U+A97F | Hangul Jamo Extended-A |
| U+AC00–U+D7AF | Hangul Syllables | U+D7B0–U+D7FF | Hangul Jamo Extended-B |

トは表 7 に示す.

**■U+0080–U+00FF についての注意**　LuaTEX-ja で，marvosym パッケージ等，Unicode フォントでなく伝統的な 8 ビットフォントを用いる場合には注意が必要である.

例えば，marvosym パッケージの提供する `\Frowny` も，符号位置は 167，つまり Unicode における § (U+00A7) と同じ符号位置にある．即ち，以前のバージョンのように，「前節の文字範囲 8 内の文字は **JAchar**」という設定であったとすると，上記の `\Frowny` は和文フォントで「§」を出力することになる.

このような事態を避けるために，バージョン 20150906.0 からは U+0080–U+00FF の範囲の文字は全て **ALchar** となるように初期設定を変更している.

なお，文字範囲の設定に関わらず 1 つの文字を **ALchar**, **JAchar** で出力したい場合には，以下の例のようにそれぞれ `\ltjalchar`, `\ltjjachar` に該当文字の文字コードを渡せばよい.

```
1 \gtfamily\large % default, ALchar, JAchar
2 ¶, \ltjalchar`¶, \ltjjachar`¶\\   % default: ALchar
3 α, \ltjalchar`α, \ltjjachar`α   % default: JAchar
```

¶, ¶, ¶
α, α, α

**■絵文字を利用する場合の注意**　（luaotfload による）OpenType 機能や合字等の処理は，LuaTEX-ja が段落・水平ボックスの中身全体に対して「この文字は **JAchar** だから和文フォントで組む」とフォントを置き換えた後に適用される．そのため，絵文字など複数のコードポイントの列で表現される文字を組む場合には，列全体で **JAchar** の範囲か **ALchar** の範囲かが統一されていないといけない.

標準では絵文字として使われる可能性が大きい一部の文字が **JAchar** となっている[*10]．絵文字を用いる場合にはこの点に留意する必要がある.

[*10] 同じ Unicode ブロック内に Adobe-Japan1-6 の文字があったため.

```
\ltjsetparameter{jacharrange={+3}}
\font\nce=NotoColorEmoji.ttf:mode=harf\nce
🏻‍♀ ☒ 💋 🏻‍♂ % U+2764: JAchar
\ltjsetparameter{jacharrange={-3}}
🏻‍♀ ☒ 💋 🏻‍♂ % U+2764: ALchar
```

👩 ☒ 💋 👨　　👩‍❤‍👨

**■異体字セレクタの扱い**　バージョン 20260517.0 以降，異体字セレクタ（SVS 用の U+FE00–FE0F と IVS 用の U+E0100–U+E01EF）が **JAchar** か **ALchar** かは直前の文字（すなわち，基底文字）によって判断されることになった．歴史的経緯により U+FE00–FE0F は **ALchar** の文字範囲に，U+E0100–U+E01EF は **JAchar** の文字範囲に属することになっているが，この設定は実際には効力を持たない．

実際，下の例のように異体字セレクタを **ALchar** の文字範囲 (4) に移動させても IVS は機能する：

```
\ltjdefcharrange{4}{"E0100-"E01EF}
葛🏻‍♀城市，葛🏻‍♂飾区，葛西
```

葛城市，葛飾区，葛西

## 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 **JAchar** 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}}
```

ここで，\zw は現在の和文フォントの全角幅を表す長さであり，pTEX における長さ単位 zw と同じように使用できる．

これらのパラメータの値は以下のように取得できる．戻り値は内部値ではなく**文字列**である（\the は前置できない）ことに注意してほしい：

```
kanjiskip: \ltjgetparameter{kanjiskip},\\
xkanjiskip: \ltjgetparameter{xkanjiskip}
```

kanjiskip: 0.0pt plus 0.99597pt minus 0.09953pt,
xkanjiskip: 2.69249pt plus 1.61542pt minus 0.64616pt

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が，\ltjgetparameter によって取得することはできないので注意が必要である．

## 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（9.1 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[*11]．

## 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（または \tbaselineshift）を設定することでなされていた（**ALchar** のベースラインがその分だけ下がる）．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と和文フォントのベースラインのシフト量を独立に設定できるようになっている．

| | 横組など | 縦組 |
|---|---|---|
| **欧文フォントのシフト量** | yalbaselineshift parameter | talbaselineshift parameter |
| **和文フォントのシフト量** | yjabaselineshift parameter | tjabaselineshift parameter |

下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．

```
\vrule width 150pt height 0.2pt depth 0.2pt
    \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

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）：

[*11] バージョン 20261122.0 以降では，各ページのヘッダやフッタの組版時には（つまり，output routine の内部では）autoxspacing の値が \begin{document}実行時の値にリセットされる．これはバージョン 20260420.0 で導入された 11.6 節の措置における副作用への対応である．
もし「文書全体で，ヘッダやフッタも含め xkanjiskip の挿入を無効化したい」場合は \begin{document}**より前**に設定を行なっておく必要がある．

```
\vrule width 150pt height4.417pt depth-4.217pt%
\kern-150pt
\large xyz漢字
{\scriptsize
  \ltjsetparameter{yjabaselineshift=-1.757pt,
    yalbaselineshift=-1.757pt}
  漢字xyzあいう
}あいうabc
```

xyz 漢字漢字 xyz あいうあいう abc

なお，以下がどちらも成り立つ場合には 1 文字の **ALchar** からなる「音節」の深さは増加しないことに注意．

- yalbaselineshift, talbaselineshift パラメータが正になっている．
- 「音節」を構成する唯一の文字 $p$ の左余白への突出量 (\lpcode)，右余白への突出量 (\rpcode) がどちらも零ではない．

**JAchar** は必要に応じて 1 文字ずつボックスにカプセル化されるため，yjabaselineshift, tjabaselineshift パラメータについてはこのような問題は起こらない．

**■数式における挙動：pTEX との違い**　**ALchar** のベースラインを補正する yalbaselineshift, talbaselineshift パラメータはほぼ pTEX における \ybaselineshift, \tbaselineshift に対応しているものであるが，数式中の挙動は異なっているので注意が必要である（表 8 参照）．

- バージョン 20221002.0 以降では，pTEX 4.0.0 と同様に数式が yalbaselineshift だけシフトされる．
  しかしそれでは数式中に直に書かれた \hbox, \vbox 中の欧文には yalbaselineshift が二重に適用されることになるので，数式中に直に書かれた \hbox, \vbox には yalbaselineshift を打ち消す補正をしている．
  なお，\vcenter によるボックスにはこの「打ち消す補正」は行われないので注意．
- pTEX では数式のスタイルごとに「打ち消す補正」の割合を \textbaselineshiftfactor, \scriptbaselineshiftfactor, \scriptscriptbaselineshiftfactor で指定できるようにしたが，LuaTEX-ja では 2 番の数式ファミリ (\fam2) に使われているフォントの大きさから自動で計算する．
- 数式中に直に書かれた和文文字（表 8 中の “あ”）については pTEX と LuaTEX-ja で違いが見られる．pTEX では，欧文文字と変わらず欧文ベースライン補正 (\ybaselineshift) がかかり，また周囲に和欧文間空白 (\xkanjiskip) が入りうる．その一方，LuaTEX-ja では「和文ベースライン補正 (yjabaselineshift) がかかる」見た目になり，また周囲に和欧文間空白は入らない．

## 4.5　禁則処理関連パラメータと OpenType 機能

禁則処理や kanjiskip, xkanjiskip の挿入に関連したパラメータのうち

jaxspmode, alxspmode, prebreakpenalty, postbreakpenalty, kcatcode

は，文字コードごとに設定する量である．

fontspec パッケージを使う（3.2 節）場合など，各種の OpenType 機能を適用することもあると思うが，前段落に述べたパラメータ類は，**OpenType 機能の適用前の文字コードによって適用される．**例

表 8. 数式関係のベースライン補正 (yalbaselineshift = 10 pt)

| | |
|---|---|
| **入力** | 数式abc: $あa\hbox{い}$, $\int_0^x t\,dt=x^2/2$,<br>$\Phi\vdash F(x)\ \hbox{for all}\ x\in A$,<br>$\sqrt{A}-\underline{X}+\frac{あ3}{2あ}-\vcenter{\hbox{aお}}$ |
| **pTEX (p4.0.0)** | 数式 い<br>abc: あ*latea* , $\int_0^x t\,dt = x^2/2, \Phi \vdash F(x)$ for all $x \in A$ $\sqrt{A}-\underline{X}+\frac{あ late3}{2late あ}-$ a お |
| **LuaTEX-ja** | 数式 あ い<br>abc: *a* , $\int_0^x t\,dt = x^2/2, \Phi \vdash F(x)$ for all $x \in A$, $\sqrt{A}-\underline{X}+\frac{あ}{\frac{3}{あ}2}-$ a お |

えば，以下の例において 10 行目の「ア」は，hwid feature の適用により半角カタカナの「ｱ」に置き換わる．しかし，その直後に挿入される postbreakpenalty は，置換前の「ア」に対する値 10 である．

```
\ltjsetparameter{postbreakpenalty={`ア, 10}}
\ltjsetparameter{postbreakpenalty={`ｱ, 20}}

\newcommand\showpostpena[1]{%
  \leavevmode\setbox0=\hbox{#1\hbox{}}%
  \unhbox0\setbox0=\lastbox\the\lastpenalty}

\showpostpena{ア},
\showpostpena{ｱ},
{\addjfontfeatures{CharacterWidth=Half}\showpostpena{ア}}
```

ア 10,ｱ 20, ア 10

# 第 II 部
# リファレンス

## 5 LuaTEX-ja における \catcode

### 5.1 予備知識：pTEX と upTEX における \kcatcode

pTEX，upTEX においては，和文文字が制御綴内で利用できるかどうかは \kcatcode の値によって決定されるのであった．詳細は表 9 を参照されたい．

pTEX では \kcatcode は JIS X 0208 の区単位，upTEX では概ね Unicode ブロック単位[*12]で設定可能になっている．そのため，pTEX と upTEX の初期状態では制御綴内で使用可能な文字が微妙に異なっている．

### 5.2 LuaTEX-ja の場合

LuaTEX-ja では，従来の pTEX・upTEX における \kcatcode の役割を分割している：

**欧文/和文の区別 (upTEX)** \ltjdefcharrange と jacharrange パラメータ（4.1 節）
**制御綴中に使用可か** LuaTEX 自身の \catcode でよい
**jcharwidowpenalty が挿入可か** kcatcode パラメータの最下位ビット
**直後の改行** 日本語しか想定していないので，**JAchar** 直後の改行で半角スペースが挿入されることはない．

最近の（2015-10-01 以降の）LuaLATEX では漢字や仮名を制御綴内に使用することが可能であるが，全角英数字は使用できない．これでは pTEX で使用できた \１年目西暦[*13]などが使えないため，LuaTEX-ja への移行で手間が生じることになる．

そのため，LuaTEX-ja では全角英数字など一部の文字[*14]の \catcode を 11 に変更し，これらの文字を制御綴中で使用可能にしている．

### 5.3 制御綴中に使用出来る JIS 非漢字の違い

エンジンが異なるので，pTEX, upTEX, LuaTEX-ja において制御綴中に使用可能な JIS X 0208 の文字は異なる．異なっているところだけを載せると，表 10 のようになる．「・」「゛」「゜」を除けば，LuaTEX-ja では upTEX より多くの文字が制御綴に使用可能になっている．

JIS X 0213 の範囲に広げると，差異はさらに大きくなる．詳細については例えば https://github.com/h-kitagawa/kct 中の kct-out.pdf などを参照すること．

[*12] U+FF00–U+FFEF (Halfwidth and Fullwidth Forms) は「全角英数字」「半角カナ」「その他」と 3 つに分割されており，それぞれ別々に \kcatcode が指定できるようになっている．
[*13] 科研費 LATEX で使用されているそうです．
[*14] 正確には，Unicode の行分割アルゴリズム (UAX #14) で “ID” (Ideographic) と指定されている文字．

表 9. \kcatcode in upTEX u1.30

| \kcatcode | 意図 | 制御綴中に使用 | 文字ウィドウ処理 | 直後での改行 |
|---|---|---|---|---|
| 15 | non-cjk | | (treated as usual LaTeX) | |
| 16 | kanji | Y | Y | ignored |
| 17 | kana | Y | Y | ignored |
| 18 | other | N | N | ignored |
| 19 | hangul | Y | Y | space |

文字ウィドウ処理：「漢字が一文字だけ次の行に行くのを防ぐ」\jcharwidowpenalty が，その文字の直前に挿入されうるか否か，を示す．

表 10. 制御綴中に使用出来る JIS X 0208 非漢字の違い

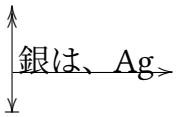
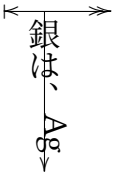
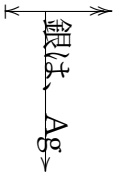
| | 区 | 点 | pTEX | upTEX | LuaTEX-ja |
|---|---|---|---|---|---|
| ・(U+30FB) | 1 | 6 | N | Y | N |
| ゛(U+309B) | 1 | 11 | N | Y | N |
| ゜(U+309C) | 1 | 12 | N | Y | N |
| ｀(U+FF40) | 1 | 14 | N | N | Y |
| ＾(U+FF3E) | 1 | 16 | N | N | Y |
| ￣(U+FFE3) | 1 | 17 | N | N | Y |
| ＿(U+FF3F) | 1 | 18 | N | N | Y |
| ヽ(U+30FD) | 1 | 19 | N | Y | Y |
| ヾ(U+30FE) | 1 | 20 | N | Y | Y |
| ゝ(U+309D) | 1 | 21 | N | Y | Y |
| ゞ(U+309E) | 1 | 22 | N | Y | Y |
| 〃(U+3003) | 1 | 23 | N | N | Y |
| 仝(U+4EDD) | 1 | 24 | N | Y | Y |
| 々(U+3005) | 1 | 25 | N | N | Y |
| 〆(U+3006) | 1 | 26 | N | N | Y |
| 〇(U+3007) | 1 | 27 | N | N | Y |
| ー(U+30FC) | 1 | 28 | N | Y | Y |

| | 区 | 点 | pTEX | upTEX | LuaTEX-ja |
|---|---|---|---|---|---|
| ／(U+FF0F) | 1 | 31 | N | N | Y |
| ＼(U+FF3C) | 1 | 32 | N | N | Y |
| ～(U+FF5E) | 1 | 33 | N | N | Y |
| ｜(U+FF5C) | 1 | 35 | N | N | Y |
| ＋(U+FF0B) | 1 | 60 | N | N | Y |
| ＝(U+FF1D) | 1 | 65 | N | N | Y |
| ＜(U+FF1C) | 1 | 67 | N | N | Y |
| ＞(U+FF1E) | 1 | 68 | N | N | Y |
| ＃(U+FF03) | 1 | 84 | N | N | Y |
| ＆(U+FF06) | 1 | 85 | N | N | Y |
| ＊(U+FF0A) | 1 | 86 | N | N | Y |
| ＠(U+FF20) | 1 | 87 | N | N | Y |
| 〒(U+3012) | 2 | 9 | N | N | Y |
| 〓(U+3013) | 2 | 14 | N | N | Y |
| ￢(U+FFE2) | 2 | 44 | N | N | Y |
| Å(U+212B) | 2 | 82 | N | N | Y |
| ギリシャ文字（6 区） | | | Y | N | Y |
| キリル文字（7 区） | | | N | N | Y |

## 6 縦組

LuaTEX 本体でも，Ω・ℵ 由来の機能として，複数の組方向をサポートしている．しかし，LuaTEX がサポートするのは TLT, TRT, RTT, LTL のみであり，日本語の縦組に使うのは望ましくない[*15]．そのため，LuaTEX-ja では横組 (TLT) で組んだボックスを回転させる方式で縦組を実装した．

表 11. LuaTEX-ja のサポートする組方向

| | 横組 | 縦組 | 「dtou 方向」 | 「utod 方向」 |
|---|---|---|---|---|
| **命令** | \yoko | \tate | \dtou | \utod |
| **字送り方向** | 水平右向き（→） | 垂直下向き（↓） | 垂直上向き（↑） | 垂直下向き（↓） |
| **行送り方向** | 垂直下向き（↓） | 水平左向き（←） | 水平右向き（→） | 水平左向き（←） |
| **使用する和文フォント** | 横組用 (\jfont) | 縦組用 (\tfont) | 横組用 (\jfont) の 90° 回転 | |
| **組版例**[*] |  |  |  |  |



[*] 幅 (width)，高さ (height)，深さ (depth) の増加方向を，それぞれ「⟶」，「↠」，「⟶」で表している．

## 6.1 サポートする組方向

LuaTEX-ja がサポートする組方向は表 11 に示す 4 つである．4 列目の \dtou は聞き慣れない命令だと思うが，実は pTEX に同名の命令が（ドキュメントには書かれていないが）存在する．Down-TO-Up の意味なのだろう．\dtou を使用する機会はないだろうが，LuaTEX-ja ではデバッグ用に実装している．5 列目の \utod は，pTEX で言う「縦数式ディレクション」に相当するものである．

組方向は，\yoko, \tate, \dtou, \utod をそれぞれ使用することで，現在作成中のリストやボックスが空の時にのみ変更可能である．ただし，現在のモードが非制限水平モードや（文中，別行立て問わず）数式モードであるときには組方向を変更することは出来ない．また，縦組中の数式内のボックスは pTEX と同じように組方向が \utod となる．

なお，LaTEX の下で LuaTEX-ja を使用する場合，組方向変更命令には「新たな組方向下での和文フォントを必要なら読み込み（・選択する）」という処理が付け加えられている（11.1 節参照）．

## 6.2 異方向のボックス

縦組の中に「42」などの 2 桁以上の算用数字を横組で組むなど，異なる組方向を混在させることがしばしば行われる．組方向の混在も pTEX と同じようにできる：

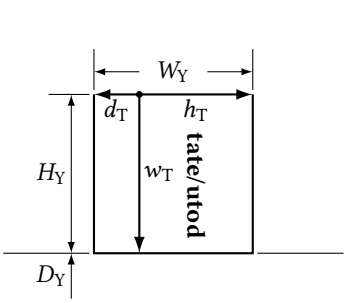
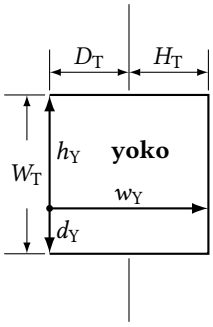
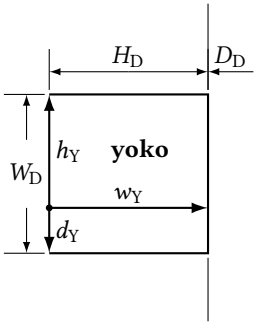
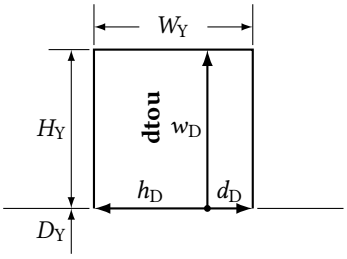
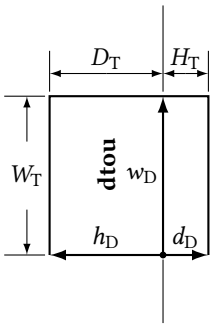
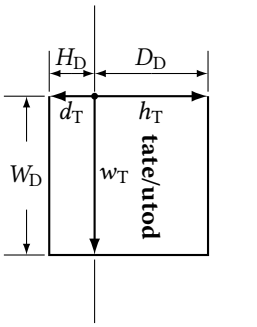
```
1 ここは横組%      yoko
2 \hbox{\tate %    tate
3   \hbox{縦組}%   tate
4   の中に
5   \hbox{\yoko 横組の内容}% yoko
6   を挿入する
7 }
8 また横組に戻る% yoko
```

ここは横組 縦組の中に横組の内容を挿入する また横組に戻る

異なる組方向のボックスを配置した場合にどう組まれるかの仕様も，pTEX を踏襲している．表 12 に示す．

[15] 和文文字だけならば RTT を使えばなんとかなると思うが，欧文文字が入ってきた場合はうまくいかず，RTR という組方向が必要になる．

表 12. 異方向のボックスの配置

| 横組中に配置 | 縦組中に配置 | 組方向 \dtou 中に配置 |
|---|---|---|
|  |  |  |
| $W_Y = h_T + d_T$,<br>$H_Y = w_T$,<br>$D_Y = 0\,\mathrm{pt}$ | $W_T = h_Y + d_Y$,<br>$H_T = w_Y/2$,<br>$D_T = w_Y/2$ | $W_D = h_Y + d_Y$,<br>$H_D = w_Y$,<br>$D_D = 0\,\mathrm{pt}$ |
|  |  |  |
| $W_Y = h_D + d_D$,<br>$H_Y = w_D$,<br>$D_Y = 0\,\mathrm{pt}$ | $W_T = h_D + d_D$,<br>$H_T = d_D$,<br>$D_T = h_D$ | $W_D = w_T$,<br>$H_D = d_T$,<br>$D_D = h_T$ |

**■\wd 達と組方向**　ボックスレジスタ \box⟨*num*⟩ にセットされているボックスの幅・高さ・深さの取得や変更にはそれぞれ \wd, \ht, \dp プリミティブを用いるのであった．pTEX ではこれらのプリミティブは，「現在の組方向におけるボックスの寸法」を指すもので，同じボックスに対しても現在の組方向によって返る値は異なるものであった．

LuaTEX-ja においては状況が異なり，\wd, \ht, \dp が返す値は現在の組方向には依存しない．下の例のように，横組のボックスが格納されていれば \wd 等は常に「横組におけるボックスの寸法」を意味する．

```
\setbox0=\hbox to 20pt{foo}
\the\wd0,~\hbox{\tate\vrule\the\wd0}
\wd0=100pt
\the\wd0,~\hbox{\tate \the\wd0}
```

20.0pt, 100.0pt,

pTEX のように現在の組方向に応じたボックスの寸法の取得・設定を行うには，代わりに次の命令を使用する．

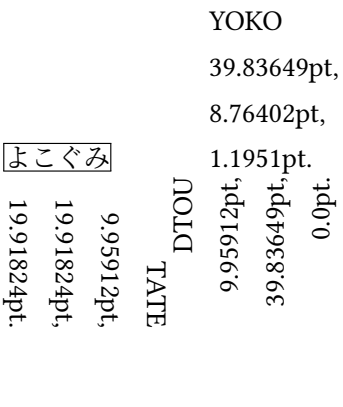
\ltjgetwd⟨*num*⟩, \ltjgetht⟨*num*⟩, \ltjgetdp⟨*num*⟩

現在の組方向に応じたボックスの寸法の取得を行う．結果は内部長さであるため，

```
\dimexpr 2\ltjgetwd42-3pt\relax, \the\ltjgetwd1701
```

のように \wd⟨*num*⟩ の代わりとして扱うことができる．使用例は以下の通りである．

```
\parindent0pt
\setbox32767=\hbox{\yoko よこぐみ}
\fboxsep=0mm\fbox{\copy32767}
\vbox{\hsize=20mm
\yoko YOKO \the\ltjgetwd32767, \\
  \the\ltjgetht32767, \\ \the\ltjgetdp32767.}
\vbox{\hsize=20mm\raggedleft
\tate TATE \the\ltjgetwd32767, \\
  \the\ltjgetht32767, \\ \the\ltjgetdp32767.}
\vbox{\hsize=20mm\raggedleft
\dtou DTOU \the\ltjgetwd32767, \\
  \the\ltjgetht32767, \\ \the\ltjgetdp32767.}
```




\ltjsetwd⟨*num*⟩=⟨*dimen*⟩, \ltjsetht⟨*num*⟩=⟨*dimen*⟩, \ltjsetdp⟨*num*⟩=⟨*dimen*⟩

現在の組方向に応じたボックスの寸法の設定を行う．\afterassignment を 2 回利用して実装しているので，次の 4 通りは全て同じ意味である．

```
\ltjsetwd42 20pt,  \ltjsetwd42=20pt,  \ltjsetwd=42 20pt,  \ltjsetwd=42=20pt
```

設定値は「横組」「縦組または utod 方向」「dtou 方向」の 3 種ごとに独立して記録される．参考として，Git リポジトリ内の test/test55-boxdim_diffdir.{tex,pdf}を挙げておく．

## 6.3 組方向の取得

「現在の組方向」や「⟨*num*⟩ 番のボックスの組方向」は，pTEX では \ifydir や \ifybox⟨*num*⟩ といった条件判断文を使って判断することができた．しかし，LuaTEX-ja はあくまでも TEX マクロと Lua コードで記述されており，それでは新たな条件判断命令を作るのは難しい．

LuaTEX-ja では，direction パラメータで現在の組方向を，boxdir パラメータ（と追加の引数 ⟨*num*⟩）によって \box⟨*num*⟩ の組方向をそれぞれ取得できるようにした．戻り値は文字列である：

| 組方向 | 横組 | tate 縦組 | dtou 方向 | utod 方向 | (未割り当て) |
|---|---|---|---|---|---|
| 戻り値 | 4 | 3 | 1 | 11 | 0 |

```
\leavevmode\def\DIR{\ltjgetparameter{direction}}
\hbox{\yoko\DIR}, \hbox{\tate\DIR},
\hbox{\dtou\DIR}, \hbox{\utod\DIR},
\hbox{\tate$\hbox{tate math: \DIR}$}

\setbox2=\hbox{\tate}\ltjgetparameter{boxdir}{2}
```

4, 3, 1, 11, tate math: 11

3

これらを用いれば，例えば pTEX の \ifydir, \ifybox200 と同等の条件判断を

```
\ifnum\ltjgetparameter{direction}=4
\ifnum\ltjgetparameter{boxdir}{200}=4
```

のように行うことができる．\iftdir は少々面倒であるが，8 で割った余りが 3 であるか否かを判断すれば良いから

```
\ifnum\numexpr
  \ltjgetparameter{direction}-(\ltjgetparameter{direction}/8)*8=3
```

とすればよい．

### 6.4　実装の比較

現バージョンの LuaTEX-ja では，縦組みの実装方法について従来からの「一文字ずつ回転」方式（以下，**方式 A**）と試験中の「RTT と Identity-V CMap を用いた」方式（以下，**方式 B**）の 2 つから選べるようになっている．2.2 小節で述べたように，今のところ標準は方式 A であり，方式 B を試すには

```
\directlua{luatexja_cmapidv = true}
```

を LuaTEX-ja 読み込み前[*16]に記述する必要がある．

どちらの方式でも，「横組 (TLT) でボックスを組み，最後にボックス全体を時計回りに 90 度回転させる」ことと使うグリフは同じだが，それ以外の点では異なる：

**方式 A**　縦組用和文フォントであっても，PDF 中では Identity-H エンコーディング（横組用）を用いる．ボックス内の **JAchar** は一文字ずつ反時計周りに回転される．

LuaTEX-ja の縦組みとして 10 年以上用いられており安定した方法であるが，PDF では一文字ずつ別々の TJ オペレータで描画されるという性質上，この方式で組まれた縦組からはまともにテキスト抽出ができないというのが難点である．

**方式 B**　縦組用和文フォントは，PDF 中で Identity-V エンコーディング（縦組用）を用いる．同じ「水平位置」で出力可能な **JAchar** たちの並びは，Ω でいう組方向 RTT のボックスとしてまとめられる[*17]．PDF では，個々の RTT ボックスごとに 1 つの TJ オペレータでまとめて描画されることになる．

方式 A と比べるとテキスト抽出がまともに行えるが，未だ不安が残るのは確かである：

- Identity-V CMap や，フォントの書字方向 (WMode) として縦書きを用いることは LuaTEX の undocumented な機能である．
- 組方向 RTT はほとんど使用されておらず，LuaTEX でもほとんどテストされていない．現状ではグリフ出力位置の計算がどこかおかしいようである[*18]．

両者の比較として，31 ページに載せた入力例と，その下にある PDF の一部も参照してほしい．

## 7　プリミティブの再定義

LuaTEX-ja では和文組版や異なる組方向に対応するために，以下に挙げるプリミティブは \protected\def により再定義を行っている．

\/　和文フォントに対するイタリック補正のサポートが追加されている．

\unhbox⟨*number*⟩, \unvbox⟨*number*⟩, \unhcopy⟨*number*⟩, \unvcopy⟨*number*⟩

ボックスの組方向が現在のリストと異なる場合は事前にエラーメッセージを出力する．pTEX と

[*16] LaTEX で使う場合には，\documentclass より前に記述するのが安全である．

[*17] まとめる作業は \shipout の直前に行われるため，ユーザが意識することは通常はない．

[*18] LuaTEX-ja では適宜カーンで補正しているが，その補正量の根拠は「色々な場合で試してみてとりあえずこれでうまくいっているようだ」以上にはない．

```
\documentclass{ltjarticle}
\pagestyle{empty}\pdfcompresslevel=0
\begin{document}
\hbox{\tate あいう\ltjjachar`α漢字}% 「α」は90度寝た形で出力される
\end{document}
```

図 1. 縦組みの 2 方式の比較用入力

```
<< /Length 937 >>
stream
1 0 0 1 133.768 655.585 cm
q
0 -1 1 0 0 0 cm
1 0 0 1 -57.517 4.793 cm
q
0 1 -1 0 0 0 cm
1 0 0 1 -76.251 -660.378 cm
BT
/F18 9.58624 Tf
1 0 0 1 71.458 651.942 Tm [<034B>]TJ
ET
1 0 0 1 76.251 660.378 cm
Q
1 0 0 1 9.586 0 cm
q
0 1 -1 0 0 0 cm
1 0 0 1 -85.837 -660.378 cm
BT
/F18 9.58624 Tf
1 0 0 1 81.044 651.942 Tm [<034D>]TJ
ET
1 0 0 1 85.837 660.378 cm
Q
1 0 0 1 9.586 0 cm
q
0 1 -1 0 0 0 cm
1 0 0 1 -95.423 -660.378 cm
BT
/F18 9.58624 Tf
1 0 0 1 90.63 651.942 Tm [<034F>]TJ
ET
1 0 0 1 95.423 660.378 cm
Q
1 0 0 1 -95.423 -660.378 cm
BT
/F18 9.58624 Tf
1 0 0 1 105.01 656.736 Tm [<040B>]TJ
ET
1 0 0 1 114.596 660.378 cm
q
0 1 -1 0 0 0 cm
1 0 0 1 -114.596 -660.378 cm
BT
/F18 9.58624 Tf
1 0 0 1 109.803 651.942 Tm [<05FD>]TJ
ET
1 0 0 1 114.596 660.378 cm
Q
1 0 0 1 9.586 0 cm
q
0 1 -1 0 0 0 cm
1 0 0 1 -124.182 -660.378 cm
BT
/F18 9.58624 Tf
1 0 0 1 119.389 651.942 Tm [<08C8>]TJ
ET
1 0 0 1 124.182 660.378 cm
Q
1 0 0 1 9.586 -4.793 cm
Q

endstream
```

図 2. 方式 A による出力の一部

```
<< /Length 493 >>
stream
1 0 0 1 133.768 655.585 cm
q
0 -1 1 0 0 0 cm
1 0 0 1 -57.517 4.793 cm
q
0 1 -1 0 0 0 cm
1 0 0 1 -76.251 -660.378 cm
BT
/F18 9.58624 Tf
1 0 0 1 76.251 660.378 Tm [<034B034D034F>]TJ
ET
1 0 0 1 76.251 660.378 cm
Q
1 0 0 1 -76.251 -660.378 cm
BT
/F18 9.58624 Tf
1 0 0 1 109.803 665.172 Tm [<040B>]TJ
ET
1 0 0 1 114.596 660.378 cm
q
0 1 -1 0 0 0 cm
1 0 0 1 -114.596 -660.378 cm
BT
/F18 9.58624 Tf
1 0 0 1 114.596 660.378 Tm [<05FD08C8>]TJ
ET
1 0 0 1 114.596 660.378 cm
Q
1 0 0 1 19.172 -4.793 cm
Q

endstream
```

図 3. 方式 B による出力の一部

```
\makeatletter\scriptsize\ttfamily
\meaning\vadjust        \\ % current          luacall 50
\meaning\ltj@@vadjust   \\ % LuaTeX-ja        luacall 50
\meaning\ltj@@orig@vadjust % original         \vadjust
```

図 4. Redefining \vadjust primitive by LuaTEX-ja

異なり，エラーを無視して無理矢理 \unhbox, \unvbox 等を続行させることもできるが，その場合の組版結果は保証しない．

\vadjust{⟨*material*⟩}
: 一旦プリミティブ本来の挙動を行う．その後，⟨*material*⟩ の組方向が周囲の垂直リストの組方向と一致しない場合にエラーを出力し，該当の \vadjust を無効にする．

\insert⟨*number*⟩{⟨*material*⟩}
: 一旦プリミティブ本来の挙動を行い，その後 ⟨*material*⟩ 内の各ボックス・罫線の直前に組方向を示す direction whatsit を挿入する．

\lastbox
: ボックスの「中身」を現在の組方向に合わせるためのノード（*dir_box* という）を必要ならば除去し，正しく「中身」のボックスが返されるように前処理をする．

\raise⟨*dimen*⟩⟨*box*⟩, \lower⟨*dimen*⟩⟨*box*⟩, \moveleft⟨*dimen*⟩⟨*box*⟩, \moveright⟨*dimen*⟩⟨*box*⟩, \split⟨*number*⟩to⟨*dimen*⟩, \vcenter{⟨*material*⟩}
: これらのプリミティブについては必要に応じて *dir_box* を作成する前処理を追加している．

上記の一覧中にあるプリミティブ \⟨*primitive*⟩ については，LuaTEX-ja 読み込み前の意味が \ltj@@orig@⟨*primitive*⟩ に，そして LuaTEX-ja による再定義後の意味が \ltj@@⟨*primitive*⟩ に保存される．例えば，\vadjust については図 4 のようになっている．

## 7.1 再定義の抑制

場合によっては LuaTEX-ja によるプリミティブの再定義が不都合を起こすこともある．例えば，breqn パッケージ（少なくとも v0.98k, 2020-09-24））は読み込み時に \vadjust, \insert がプリミティブのままであることを要請するので，このままでは LuaTEX-ja の後で読み込むことはできない．

この状況に対応するため，バージョン 20210517.0 以降では

- 制御綴 \ltj@stop@overwrite@primitive 内に並べられたプリミティブは，LuaTEX-ja 読み込み直前時の意味のままとなる．
- LuaTEX-ja 読み込み後に \ltj@overwrite@primitive に引数として与えたプリミティブを与えることで，それらを「LuaTEX-ja によって再定義する」際の意味に再定義する

機能を導入した．使用例については図 5 を参照．

```
\makeatletter
\def\ltj@stop@overwrite@primitive{\insert\vadjust\/\unhbox\vcenter\fontseries}
\makeatother
  %% Keep the meaning of \insert, \vadjust, \/, \unhbox and \vcenter.
  %% \fontseries will still be redefined by \LuaTeX-ja, because it is not primitive.
\usepackage{luatexja}
...
\usepackage{breqn}
...
\makeatletter
\ltj@overwrite@primitive\expandafter{\insert\vadjust\/\unhbox\vcenter}
\makeatother
  %% Redefine \insert, \vadjust, \/, \unhbox and \vcenter.
```

図 5. \ltj@stop@overwrite@primitive and \ltj@overwrite@primitive

# 8　フォントメトリックと和文フォント

## 8.1　\jfont 命令

フォントを（横組用）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\tradmc={IPAexMincho:script=latn;%
  +trad;-kern;jfm=ujis} at 14pt
\tradmc 当／体／医／区
```

當／體／醫／區

\jfont 命令の実行ごとにどの（横組用）JFM を用いるのかを指定する必要がある．JFM は文字の寸法情報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 Lua スクリプトで，その構造は次の節で述べる．

なお，\jfont で定義された制御綴（上の例だと \tradmc）は *font_def* トークンではなくマクロである．従って，\fontname\tradmc のような入力はエラーとなる．以下では \jfont で定義された制御綴を ⟨*jfont_cs*⟩ で表す．

**■JFM の指定**　JFM の一般的な指定は次のようになっている：

\jfont⟨*jfont_cs*⟩=...;jfm=⟨*JFM name*⟩[/{⟨*JFM features*⟩}];...;[jfmvar=⟨*identifier*⟩];...

⟨*JFM name*⟩（横組用）JFM の名称．LuaTEX-ja は jfm-⟨*JFM name*⟩.lua というファイルを探索して読み込む[*19]．

⟨*JFM features*⟩ 省略可能なコンマ区切りリスト．全体を囲む {} は省略可能であるが，囲ったからと

[*19] LuaTEX-ja 20230409.0 以降では，LATEX 2ε 下で読み込まれた場合には \input@path で指定された箇所も加えて JFM を探索する．

```
\ltjsetparameter{differentjfm=both}
\jfont\F=HaranoAjiMincho-Regular:jfm=ujis
\jfont\G=HaranoAjiGothic-Medium:jfm=ujis
\jfont\H=HaranoAjiGothic-Medium:jfm=ujis;jfmvar=hoge
\F ) {\G 【】} ( % halfwidth space
   ) {\H 『』} ( % fullwidth space

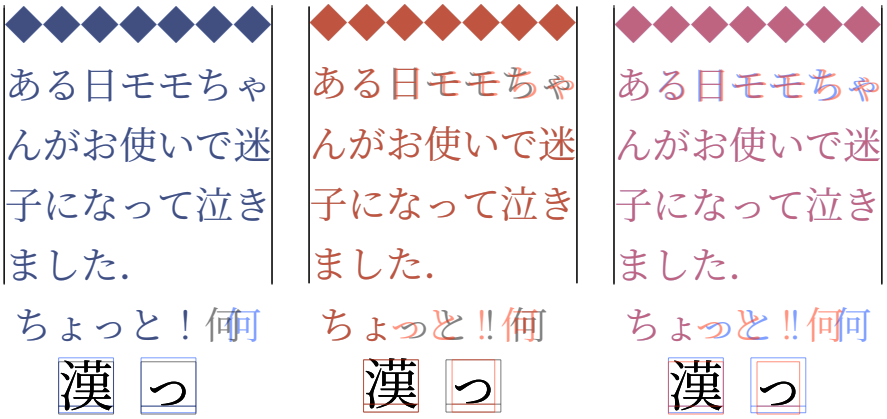
ほげ，{\G 「ほげ」}（ほげ）\par
ほげ，{\H 「ほげ」}（ほげ）% pTeX-like

\ltjsetparameter{differentjfm=paverage}
```

)【】( ) 『』 (
ほげ，**「ほげ」**（ほげ）
ほげ，　**「ほげ」**　（ほげ）

図 6. Example of `jfmvar` key

表 13. LuaTEX-ja に同梱されている横組用 JFM の違い




(Blue: `jfm-ujis.lua`, Black: `jfm-jis.lua`, Red: `jfm-min.lua`)

いって ⟨*JFM features*⟩ の中で使用可能な文字が増えるわけではない．⟨*JFM features*⟩ で指定された内容は，テーブル `luatexja.jfont.jfm_feature` として JFM 読み込み時に（JFM から）アクセス可能である．図 7 に使用例を載せた．

なお，LuaTEX-ja が標準で提供する JFM ではこの機能は用いられていない．

⟨*identifier*⟩ 省略可能な文字列．

LuaTEX-ja 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」．ここで「JFM が同じ」とは，両フォントの ⟨*JFM name*⟩, ⟨*JFM features*⟩, ⟨*identifier*⟩ が全て一致することである．

例えば図 6 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組まれる，つまり両文字の間は半角空きとなる．

しかし，JFM とサイズが同じであっても，`jfmvar` キーの値 ⟨*identifier*⟩ の異なる 2 つの和文フォント，例えば図 6 で言う `\F` と `\H`，は「区別される」．異なる和文フォントに異なる `jfmvar` キーを割り当て，かつ differentjfm パラメータを `both` に設定すれば，pTEX と似た状況で組版されることになる．

**■横組用 JFM**　以下の横組用 JFM が LuaTEX-ja には同梱されている：

**`jfm-ujis.lua`** LuaTEX-ja の標準 JFM ファイルであり，この JFM は upTEX で用いられる UTF/OTF

\A: (nil)
\B: [ps] = false, [kana] = true, [kern] = “0.5”,
\C: [down] = “0.2”, [kern] = “0.5”,
\D: [down] = “0.2”, [kern] = “0.5”,

| | \A | \B | \C | \D |
|---|---|---|---|---|
| \A | あ漢イ字 | あ漢 **イ 字** | あ漢 イ 字 | あ漢 **イ 字** |
| \B | **あ 漢** イ字 | **あ 漢イ 字** | **あ 漢** イ 字 | **あ 漢 イ 字** |
| \C | あ 漢 イ字 | あ 漢 **イ 字** | あ 漢イ 字 | あ 漢**イ 字** |
| \D | **あ 漢** イ字 | **あ 漢 イ 字** | **あ 漢**イ 字 | **あ 漢イ 字** |

```
\small\ltjsetparameter{differentjfm=both}\tabcolsep=.5\zw
  % \printjfmfeat is defined in the source of this document
\jfont\A=HaranoAjiMincho-Regular:jfm=testf                  at 9pt \printjfmfeat\A
\jfont\B=HaranoAjiMincho-Bold:jfm=testf/kern=0.5,-ps,+kana  at 9pt \printjfmfeat\B
\jfont\C=HaranoAjiGothic-Regular:jfm=testf/kern=0.5,down=0.2 at 9pt \printjfmfeat\C
\jfont\D=HaranoAjiGothic-Bold:jfm=testf/down=0.2,kern=0.5   at 9pt \printjfmfeat\D
\def\TEST#1{\string#1&{#1あ漢}{\A イ字}&{#1あ漢}{\B イ字}&{#1あ漢}{\C イ字}&{#1あ漢}{\D イ字}}
\vspace{-4\baselineskip}\hfill\ttfamily
\begin{tabular}{lllll}
  &\string\A&\string\B&\string\C&\string\D\\\TEST\A\\\TEST\B\\\TEST\C\\\TEST\D\\
\end{tabular}
  % No space between ``漢'' and ``イ'' iff two Japanese fonts uses same JFM
\ltjsetparameter{differentjfm=paverage}
```

図 7. Example of JFM features

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ことと，jfm-ujis.lua では「？」「！」の直後に半角空白が挿入されることである．

**jfm-min.lua** pTEX に同梱されているデフォルトの和文用 TFM (min10.tfm) に相当し，行末で文字が揃うようにするために「っ」など一部の文字幅が変わっている．min10.tfm については [6] が詳しい．

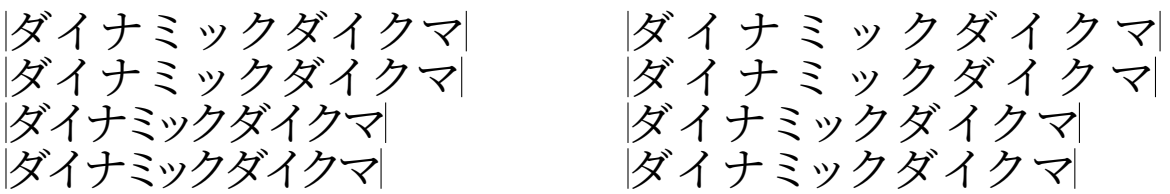
**jfm-prop.lua** プロポーショナル組用の JFM．文字幅・高さ・深さの情報も自動挿入されるグルー・カーンの情報は持たない（つまりグリフの文字幅をそのまま採用する）．

**jfm-propw.lua** プロポーショナル組用のさらなる JFM．jfm-prop.lua と異なり，高さ・深さの情報は持っている．

jfm-ujis.lua, jfm-jis.lua, jfm-min.lua の違いは表 13 に示した．表中の文例の一部には，[6] の図 3, 4 のものを用いた．

**■ペアカーニング情報の使用**　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．このカーニング情報は以前の LuaTEX-ja とはあまり相性が良くなかったが，バージョン 20140324.0 以降ではカーニングによる空白はイタリック補正と同様に扱うことになっている．つまり，カーニング由来の空白と JFM 由来のグルー・カーンは同時に入ることがある．図 8 を参照．

- **\jfont や，NFSS2 用の命令（3.1 節，11.2 節）では，カーニング情報を使用する設定（OpenType**




```
\newcommand\test{\vrule ダイナミックダイクマ\vrule\\}
\jfont\KMFW = HaranoAjiMincho-Regular:jfm=prop;-kern at 17.28pt
\jfont\KMFK = HaranoAjiMincho-Regular:jfm=prop at 17.28pt % kern is activated
\jfont\KMPW = HaranoAjiMincho-Regular:jfm=prop;script=dflt;+palt;-kern at 17.28pt
\jfont\KMPK = HaranoAjiMincho-Regular:jfm=prop;script=dflt;+palt;+kern at 17.28pt
\begin{multicols}{2}
\ltjsetparameter{kanjiskip=0pt}
{\KMFW\test \KMFK\test \KMPW\test \KMPK\test}

\ltjsetparameter{kanjiskip=3pt}
{\KMFW\test \KMFK\test \KMPW\test \KMPK\test}
\end{multicols}
```

図 8. Kerning information and kanjiskip

**機能 kern）はとくに指定しなくても有効になる**．すなわち，以下の 2 行目と 3 行目，5 行目と 6 行目はそれぞれ等価である：

```
\jfont\hoge=hogem:jfm=ujis;-kern at 3.5mm % ==> kern 無効（明示）
\jfont\hoge=hogem:jfm=ujis at 3.5mm       % ==> kern 有効（暗黙）
\jfont\hoge=hogem:jfm=ujis;+kern at 3.5mm % ==> kern 有効（明示）
\DeclareFontShape{JY3}{fuga}{m}{n}{<-> s*hogem:jfm=ujis;-kern}{} % ==> kern 無効（明示）
\DeclareFontShape{JY3}{fuga}{m}{n}{<-> s*hogem:jfm=ujis}{}       % ==> kern 有効（暗黙）
\DeclareFontShape{JY3}{fuga}{m}{n}{<-> s*hogem:jfm=ujis;+kern}{} % ==> kern 有効（明示）
```

- バージョン 20220411.0 以降では，LuaTEX-ja 読み込み時や，ltjclasses, ltjsclasses において和文フォントを

  ```
  \jfont\tenmin=\ltj@stdmcfont:-kern;jfm=\ltj@stdyokojfm\space at 9.62216pt
  \DeclareFontShape{JY3}{mc}{m}{n}{<-> s*[\ifdefined\Cjascale\Cjascale\else 0.962216\fi]
    \ltj@stdmcfont:-kern;jfm=\ltj@stdyokojfm}{}
  ```

  と OpenType 機能 kern を明示的に無効化した状態で定義する．これは標準 JFM (jfm-ujis.lua, jfm-ujisv.lua) がフォント由来のカーニングが入ることを期待していないためである．
- 一方，**luatexja-fontspec の提供する \setmainjfont などの命令では，既定ではカーニング情報は使用しない** (Kerning=Off)．すなわち，次の 2 行は等価である：

  ```
  \setmainjfont{HaranoAjiMincho-Regular}
  \setmainjfont[Kerning=Off]{HaranoAjiMincho-Regular}
  ```

  これは前項目の理由の他に，以前のバージョンの LuaTEX-ja との互換性のためもある．

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

```
\fboxsep=0mm\fboxrule=0.125mm
\jfont\Js=HaranoAjiMincho-Regular:jfm=ujis;-kern
\jfont\Jd=HaranoAjiMincho-Regular:down=0.5;jfm=ujis;-kern
\jfont\Jl=HaranoAjiMincho-Regular:left=0.25;jfm=ujis;-kern
\fbox{\Js 漢字「A」あ}\ \fbox{\Jd 漢字「B」あ}\\[10pt]
\fbox{\Jl 漢字「C」あ}\
\fbox{\Js\ltjsetparameter{yjabaselineshift=.5\zh}漢字「D」あ}
```

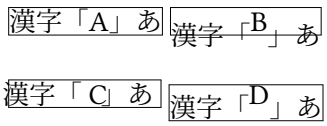



図 9. down and left

**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[*20]．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=HaranoAjiMincho-Regular:extend=1.5;jfm=ujis;-kern
\jfont\S=HaranoAjiMincho-Regular:slant=1;jfm=ujis;-kern
\E あいうえお \S あいう\/ABC
```

あいうえおあいうABC

**■down と left**　他の OpenType 機能と同じような形式で down=⟨*real*⟩ を指定した場合，これらは「JFM 中のすべての文字クラスで down の値を ⟨*real*⟩ だけ増やす」のと同じ効力を発揮する．⟨*real*⟩ の単位は design size である．言い換えれば，図 9 に示すように，JFM で指定された文字の幅・高さ・深さは**変えず**に，実際のグリフは ⟨*real*⟩ だけ下に下がって組まれることになる．その意味で，この down=⟨*real*⟩ 指定は yjabaselineshift の代替ではない．

down と同様に，「JFM のすべての文字クラスで left の値を ⟨*real*⟩ だけ増やす」のと同じ効力を発揮する left=⟨*real*⟩ も使用可能である．

**■ltjksp「機能」**　LuaTEX-ja 標準では，JFM 中における kanjiskip_natural, kanjiskip_stretch, kanjiskip_shrink キー（44 ページ）の使用によって，「JFM 由来のグルーの他に，kanjiskip の自然長/伸び量/縮み量の一部が同じ場所に挿入される」という状況が起こりうる．この機能を無効化し，バージョン 20150922.0 以前と同じような組版を得るためには，他の OpenType 機能と同じように -ltjksp 指定を行えば良い（図 10 参照）．なお，

```
\jfont\G=HaranoAjiMincho-Regular:jfm=ujis;-ltjksp;+ltjksp at \zw
```

のように +ltjksp 指定を行った場合は，kanjiskip_natural など 3 キーは再び有効化される．-ltjksp, +ltjksp を複数回指定した場合は，最後に指定したものが有効となる．

**■ltjpci「機能」**　luaotfload v3.19 以降では，標準で Unicode（文字から作られるノードたち）が NFC に正規化されるようになっている．これにより，ソース中で例えば「か」と合成用濁点 (U+3099) を続けて入力した場合，両者それぞれからノードが生成されるが，結果的には「が」を表す 1 ノードになるわけである．

しかし，NFC に正規化することで，例えば「神」(U+FA19) が「神」(U+795E) にというふうに，CJK

[*20] 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．

```
\leavevmode
\ltjsetparameter{kanjiskip=0pt plus 3\zw}
\vrule\hbox to 15\zw{あ「い」う，えお}\vrule\\
\jfont\G=HaranoAjiMincho-Regular%
  :jfm=ujis;-ltjksp at \zw
\G\leavevmode%
\vrule\hbox to 15\zw{あ「い」う，えお}\vrule
```

|あ　「い」　う，　え　お|
|---|
|あ「い」う，え　　お|

図 10. `ltjksp` “feature”

```
\def\TEST{\leavevmode\char"FA10\char"FA12\char"FA15
  \char"FA19.か\char"3099.は\char"309A.\par}
\jfont\A=HaranoAjiMincho-Regular:jfm=ujis; at 15pt
\A\TEST % default
\jfont\G=HaranoAjiMincho-Regular:jfm=ujis;-ltjpci at 15pt
\G\TEST % ltjpci off
\jfont\H=HaranoAjiMincho-Regular:jfm=ujis;-normalize at 15pt
\H\TEST % normalization off
```

塚晴凜神. が. ぱ.
塚晴凜神. が. ぱ.
塚晴凜神. か. ゚は.

図 11. `ltjpci` “feature”

互換漢字が CJK 統合漢字に変換されてしまうという問題がある．異体字セレクタを用いればこのようなことは起きないが，古くからあるフォントでは異体字セレクタをサポートしていない．

以上の事情に対応するため，LuaTEX-ja では，標準で **CJK 互換漢字・CJK 互換漢字補助の文字には luaotfload パッケージによる処理は働かない**ようにしている．この機能を無効化するには，他の OpenType 機能と同じように `-ltjpci` 指定を行えば良い（図 11 参照）．`ltjksp` と同様に，`-ltjpci`, `+ltjpci` を複数回指定した場合は，最後に指定したものが有効となる．

## 8.2 `\tfont` 命令

`\tfont` はフォントを縦組用の和文フォントとして読み込む命令であり，`\tfont` の構文は `\jfont` と同様である．`\tfont` で定義された縦組用和文フォントは，以下の点が `\jfont` による横組用和文フォントとは異なる：

- 明示的に OpenType 機能 `vert`, `vrt2`（のいずれか）の有効・無効を指定した場合を除き，自動的に OpenType 機能 `vrt2` の有効化が指定されたものとみなされる[*21].

  ```
  \tfont\S=HaranoAjiMincho-Regular:jfm=ujisv % vrt2 is automatically activated
  \tfont\T=HaranoAjiMincho-Regular:jfm=ujisv;-vert % vert and vrt2 are not activated
  \tfont\U=file:ipaexm.ttf:jfm=ujisv
      % vert is automatically activated, since this font does not have vrt2
  ```

- `vert`, `vrt2` の少なくとも一つの有効を指定した場合にも関わらず，script tag と language system identifier の値の組み合わせによって実際には有効にならなかった場合，LuaTEX-ja は

[*21] もしフォントが `vrt2` を定義していなかった場合，代わりに `vert` を用いる．

```
\jfont\X=[HaranoAjiMincho-Regular.otf]:jfm=ujis
\tfont\U=[HaranoAjiMincho-Regular.otf]:jfm=ujisv
\tfont\V=[HaranoAjiMincho-Regular.otf]:jfm=ujisv;jpotf
\def\TEST#1#2{\leavevmode
  \hbox{#1#2\string#2 “引用，と句読点．”}}
\ttfamily\centering
\TEST\yoko\X\quad \TEST\tate\U\quad
\TEST\tate\V
```

\X “引用，と句読点．”　\U “引用，と句読点．”　\V “引用、と句読点。”

図 12. jpotf “feature”

表 14. jpotf が指定された際に行われる追加の縦組形への置換

| | |
|---|---|
| ， (U+FF0C) ⟼ 、 (U+3001) | ． (U+FF0E) ⟼ 。 (U+3002) |
| ﹔ (U+201C) ⟼ ″ (U+301D) | ﹕ (U+201D) ⟼ ‶ (U+301F) |

どれかの script, language で定義されている vert による（単一グリフから単一グリフへの）置換を全部適用する

という挙動を取る[*22].

- さらに，置換前と置換後のグリフがどちらも「UAX #50 で “r” もしくは “Tr” と指定されている」ものは 90 度自動回転させる．
- 8.6 節で述べる，数式中の和文フォントには縦組用和文フォントは指定できない．
- ⟨*JFM name*⟩ には縦組用 JFM を指定する．以下の縦組用 JFM が LuaTEX-ja には同梱されている．

  **jfm-ujisv.lua** LuaTEX-ja の標準縦組用 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-v.tfm を元にしている．

  **jfm-tmin.lua** pTEX に同梱されているデフォルトの和文用縦組 TFM である tmin10.tfm に相当し，min10.tfm と同様に「っ」など一部の文字幅が狭められている．
- vert, vrt2 の少なくとも片方が（明示的・自動的を問わず）有効になっていた場合，さらに jpotf を指定することで「通常では行わない縦組用字形への置換」を行うことができる．
  標準では，表 14 に示した置換が登録されている[*23]．実行例は図 12 を参照．
  ユーザ側で「置換」をカスタマイズしたい場合，luatexja.jfont.register_vert_replace 関数に変更内容を記したテーブルを渡す．例えば置換 $i_1 \longmapsto v_1, i_2 \longmapsto v_2, \ldots$ を登録する場合は

  ```
  \directlua{luatexja.jfont.register_vert_replace{[i1]=v1, [i2]=v2, ...}}
  ```

  を実行する．luatexja.jfont.register_vert_replace による変更はこの関数の**実行後に定義**されるフォントについてのみ有効である．

なお，pTEX では，\font, \jfont, \tfont のどれでも欧文フォント・横組用和文フォント・縦組用和文フォントの定義が可能であったが，LuaTEX-ja ではそうでないので注意．

[*22] 例えば，Windows 7 に付属している SimHei では，vert は Script が hani，Language が CHN という状況でのみ定義されている．しかし，luaotfload ではこの script, language の組み合わせを指定することはできないので，luaotfload そのままでは vert を適用させることはできない．

[*23] jpotf という名前にしたのは，OTF パッケージの縦組用和文 TFM でほぼ同じの処理（そちらではさらに一重引用符を「シングルミニュート」に置換する機能もあった）を行っていたことに由来する．

## 8.3 標準和文フォント・JFM の変更

LuaTEX-ja が読み込まれる前に以下の命令が定義されていた場合は，それらが標準和文フォントやそれらに用いる JFM として使われる．

`\ltj@stdmcfont` 明朝体として用いるフォント．
`\ltj@stdgtfont` ゴシック体として用いるフォント．
`\ltj@stdyokojfm` 標準で用いる横組用 JFM.
`\ltj@stdtatejfm` 標準で用いる縦組用 JFM.

例えば

```
\def\ltj@stdmcfont{IPAMincho}
\def\ltj@stdgtfont{IPAGothic}
```

と記述しておけば，標準和文フォントが IPA 明朝・IPA ゴシックへと変更される．

この機能は，特別の JFM を用いるクラス[*24]などでの使用を意図しており，命令名に`@`が含まれることから通常の TEX/LATEX 文書での使用は意図していない．通常の LATEX 文書では luatexja-preset や luatexja-fontspec などで使用フォントを選択することを推奨する．

旧バージョンとの互換性のため，LuaTEX から見える位置に `luatexja.cfg` があれば，LuaTEX-ja はそれを読み込む．しかし，`luatexja.cfg` 内で `\ltj@stdmcfont` 等が定義されていた場合はそちらが優先されるので，もはや `luatexja.cfg` は使わないほうが良い．

## 8.4 `psft` プリフィックス

luaotfload で使用可能になった `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．なお，現行の LuaTEX で非埋め込みフォントを作成すると PDF 内でのエンコーディングが Identity-H となり，PDF の標準規格 ISO32000-1:2008 ([10]) に非準拠になってしまうので注意してほしい．

**`psft` プリフィックスの下では `+jp90` などの OpenType 機能の効力はない．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-Japan1-7 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：

```
1 \jfont\testJ={psft:Ryumin-Light:cid=Adobe-Japan1-7;jfm=jis}     % Japanese
2 \jfont\testD={psft:Ryumin-Light:jfm=jis}                       % default: Adobe-Japan1-7
3 \jfont\testC={psft:AdobeMingStd-Light:cid=Adobe-CNS1-7;jfm=jis}% Traditional Chinese
4 \jfont\testG={psft:SimSun:cid=Adobe-GB1-6;jfm=jis}             % Simplified Chinese
```

[*24] 例えば阿部紀行氏による jlreq がそれにあたる．

```
\jfont\testK={psft:Batang:cid=Adobe-Korea1-2;jfm=jis}          % Korean
\jfont\testKR={psft:SourceHanSerifAKR9:cid=Adobe-KR-9;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 5 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.
?
```

## 8.5 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．

`version`=⟨*version*⟩（任意，既定値は 1）

JFM のバージョン．1, 2, 3 がサポートされる．

`dir`=⟨*direction*⟩（必須）

JFM の書字方向．`'yoko'`（横組）と `'tate'`（縦組）がサポートされる．

`zw`=⟨*length*⟩（必須）

「全角幅」の長さ．この量が `\zw` の長さとなる．pTEX では「全角幅」`1zw` は「文字クラス 0 の文字」の幅と決められていたが，LuaTEX-ja ではここで指定する．

`zh`=⟨*length*⟩（必須）

「全角高さ」(height + depth) の長さ．通常は全角幅と同じ長さになるだろう．pTEX では「全角高さ」`1zh` は「文字クラス 0 の文字」の高さと深さの和と決められていたが，LuaTEX-ja ではここで指定する．

`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（指定なしは 0 pt として扱われる）．

| JFM 書字方向 | **'yoko'（横組）** | **'tate'（縦組）** |
|---|---|---|
| width | 「実際のグリフ」の幅 | |
| height | 「実際のグリフ」の高さ | 0.0 |
| depth | 「実際のグリフ」の深さ | 0.0 |
| italic | 0.0 | |

表 15. width フィールド等の標準値

⟨*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）

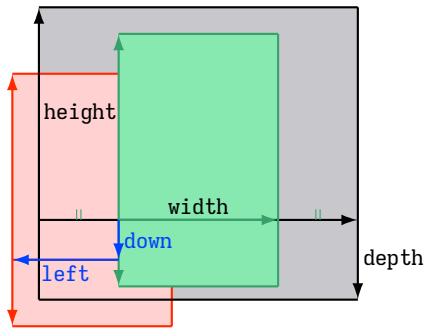
width=⟨*length*⟩, height=⟨*length*⟩, depth=⟨*length*⟩, italic=⟨*length*⟩（必須）

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．省略時や，数でない値を指定した時には表 15 に示されている値を用いる．例えば，横組用 JFM で width フィールドには数値以外の値を指定した場合，文字の幅はその「実際の」グリフの幅となる．OpenType の prop feature と併用すれば，これによってプロポーショナル組を行うことができる．

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 13（横組用和文フォント），図 14（縦組用和文フォント）で説明する．

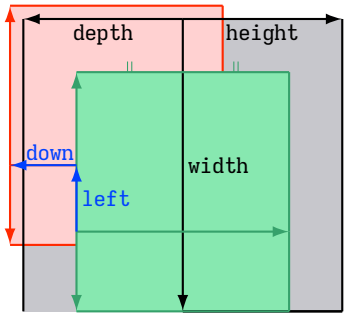
多くの場合，left と down は 0 である一方，align フィールドが 'middle' や 'right' であることは珍しいことではない．例えば，align フィールドを 'right' に指定することは，文字クラスが




`align` フィールドの値が `'middle'` であるような文字クラスに属する和文文字ノードを考えよう．

- 黒色の長方形はノードの枠であり，その幅，高さ，深さは JFM によって指定されている．
- `align` フィールドは `'middle'` なので，実際のグリフの位置はまず水平方向に中央揃えしたものとなる（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 13. 横組和文フォントにおける「実際の」グリフの位置




`align` フィールドの値が `'right'` であるような文字クラスに属する和文文字を考えよう．

- 実際のグリフの「垂直位置」は，まずベースラインが文字の物理的な左右方向の中央を通る位置となる．
- また，この場合 `align` フィールドは `'right'` なので，「水平位置」は字送り方向に「右寄せ」したものとなる（緑色の長方形）．
- その後さらに `left` と `down` の値に従ってシフトされるのは横組用和文フォントと変わらない．

図 14. 縦組和文フォントにおける「実際の」グリフの位置

開き括弧類であるときに実際必要である．

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

`glue={[`*j*`]={`⟨*width*⟩`, `⟨*stretch*⟩`, `⟨*shrink*⟩`, [ratio=`⟨*ratio*⟩`, ...]}, ...}`

文字クラス *i* の文字と *j* の文字の間に挿入されるカーンやグルーの量を指定する．

⟨*ratio*⟩ は，グルーの自然長のうちどれだけの割合が「後の文字」由来かを示す量で，0 から +1 の実数値をとる．省略時の値は 0.5 である．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，[7] 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 $0.25/(0.5 + 0.25) = 1/3$ を指定する．

グルーの指定においては，上記に加えて各 [*j*] の各サブテーブル内に次のキーを指定できる，

`priority=`⟨*priority*⟩ luatexja-adjust による優先順位付き行長調整（13.3 節）において，このグルーの優先度を指定する．許される値は以下の通り：

**バージョン 1** −4 から +3 の間の整数

**バージョン 2 以降** −4 から +3 の間の整数の 2 つ組 {⟨*stretch*⟩，⟨*shrink*⟩} か，または −4 から +3 の間の整数．⟨*stretch*⟩, ⟨*shrink*⟩ はそれぞれこのグルーが伸びるときの優先度，縮むときの優先度であり，単に整数 *i* が指定された場合は {*i*，*i*} であると解釈される．

ここで指定する値は，大きい値ほど「先に伸ばされる」「先に縮ませる」ことを意味しており，省略時の値は 0 である．範囲外の値が指定されたときの動作は未定義である．

`kanjiskip_natural=`⟨*num*⟩, `kanjiskip_stretch=`⟨*num*⟩, `kanjiskip_shrink=`⟨*num*⟩

JFM によって本来挿入されるグルーの他に kanjiskip 分の空白を自然長 (`kanjiskip_natural`)，伸び量 (`kanjiskip_stretch`)，縮み量 (`kanjiskip_shrink`) ごとに挿入するための指定である．いずれも省略された場合のデフォルト値は 0（追加しない）である．

例えば，LuaTEX-ja の横組標準 JFM の `jfm-ujis.lua` では，

- 通常の文字「あ」と開き括弧類の間に入るグルーは，自然長・縮み量半角，伸び量 0 のグルーとなっているが，さらに kanjiskip の伸び量に `kanjiskip_stretch`（ここでは 1）を掛けた分だけ伸びることが許される．
- 同様に，閉じ括弧類（全角コンマ「,」も含む）と和文文字「う」「え」，閉じ括弧類と「f」の間も自然長・縮み量半角，伸び量 0 のグルーとなっているが，さらに kanjiskip の伸び量に `kanjiskip_stretch`（ここでは 1）を掛けた分だけ伸びることが許される．
- 一方，開き括弧類と通常の文字の間，また通常の文字と閉じ括弧類の間は自然長・縮み量・伸び量 0 のグルーだが，kanjiskip の縮み量に `kanjiskip_shrink`（ここでは 1）を掛けた分だけ縮むことが許される．

となっている．従って，以下のような組版結果を得る．

```
1 \leavevmode\let\V=\vrule
2 \ltjsetparameter{kanjiskip=0pt plus 5\zw}
3 \ltjsetparameter{xkanjiskip=0pt plus 0.5\zw}
4 \V\hbox spread 7\zw{aあ「い」う，えお」f}\V
5
6 \vrule\hbox{ああ「い」う，えお」f}\V\par
7 \ltjsetparameter{kanjiskip=0pt minus \zw}
8 \V\hbox spread-2.5\zw{aあ「い」う，えお」f}\V
```

|aあ　「い」　う，　え　お」　f|
|ああ「い」う，えお」f|
|aあ「い」う，えお」f|

`end_stretch=`⟨*kern*⟩，`end_shrink=`⟨*kern*⟩（任意，バージョン 1 のみ）

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．

`end_adjust={`⟨*kern*⟩，⟨*kern*⟩，`...}`（任意，バージョン 2 以降）

行末文字の位置調整が有効であり，かつ現在の文字クラスの文字が行末に来た時に，この文字と行末の間には指定された値のいずれかの大きさのカーンが挿入される（13.3 節参照）．

バージョン 1 における

```
end_stretch = a, end_shrink = b
```

という指定は，バージョン 2 以降では次の指定と同じになる．

```
end_adjust = {-b, 0.0, a}
```

もし真ん中の `0.0` がない場合は，*a* か −*b* かいずれかのカーンが常に行末に追加される．

`round_threshold=`⟨*float*⟩（任意，バージョン 3 以降，文字クラス 0 のみ）

「実際のグリフの幅に合わせて文字幅を整数倍する」際のしきい値を指定する．より正確に述べ

ると，次のようになる．このフィールドに正の数 $r$ が指定されていたとし，JFM 中で「文字クラス 0 の文字幅」と指定されている値が $w$，文字クラス 0 に属する文字のあるグリフの実際の幅が $w'$ であったとする．$n = \mathrm{nint}(w'/w)$ とした[*25]とき，もし $w' > w$ かつ $|w'/w - n| < r$ であれば，JFM で文字幅 $nw$ が指定されたものとして扱う．
この機能は，ほとんど源ノ明朝・源ノ角ゴシックにおける 2 倍角・3 倍角ダッシュの合字のために実装されたと言ってもよい（この場合 $w' = 2, 3$ である）．これらのグリフは LuaTEX 内部では Unicode の私用領域に割り当てられるので，JFM 側で番号を指定することができない．

**■文字クラスの決定**　文字からその文字の属する文字クラスを算出する過程について，次の内容を含んだ `jfm-test.lua` を用いて説明する．

```
[0] = {
   chars = { '漢' },
   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,
},
```

ここで，次のような入力とその実行結果を考える：

```
\jfont\a=IPAexMincho:jfm=test;+hwid
\setbox0\hbox{\a ヒ漢}\the\wd0
```

15.0pt

上記の出力結果が，15 pt となっているのは理由によるものである：

1. hwid feature によって「ヒ」が半角幅のグリフ「ﾋ」と置き換わる（luaotfload による処理）．
2. JFM によれば，この「ﾋ」のグリフの文字クラスは 2000 である．
3. 以上により文字クラス 2000 とみなされるため，結果として「ﾋ」の幅は半角だと認識される．

この例は，**文字クラスの決定は OpenType 機能の適用によるグリフ置換の結果に基づく**ことを示している．

但し，JFM によって決まる置換後のグリフの文字クラスが 0 である場合は，置換前の文字クラスを採用する．次の例を参照のこと．

```
\jfont\a=HaranoAjiMincho-Regular:jfm=test;+vert
\a 漢。\inhibitglue 漢
```

漢 漢

ここで，句点「。」(U+3002) の文字クラスは，以下のようにして決まる．

1. luaotfload によって縦組用句点のグリフに置き換わる．
2. 置換後のグリフは U+FE12 であり，JFM に従えば文字クラスは **0** と判定される．
3. この場合，置換前の横組用句点のグリフによって文字クラスを判定する．
4. 結果として，上の出力例中の句点の文字クラスは 2000 となる．

[*25] ここで，$\mathrm{nint}(a) = \lfloor a + 0.5 \rfloor$ は $a$ に「もっとも近い整数」を表す．

**■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'`
: hbox 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．この「文字」との間に設定したグルー・カーンがボックス $b$ の先頭（もしくは末尾）に来た場合，そのボックス $b$ の直前（もしくは直後）には和文処理グルーは入らない．

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

`'jcharbdd'`
: **JAchar** と「その他のもの」との境界．
  バージョン 2 以前では **ALchar**，箱，罫線 (rule)，glue, kern などいろいろなものと **JAchar** の境界に対して本特殊文字が用いられていたが，バージョン 3 以降では `'jcharbdd'`, `'alchar'`, `'nox_alchar'`, `'glue'` と細分化され，`'jcharbdd'` は主に **JAchar** とボックスや罫線 (rule) との境界に使われるようになった．

`'alchar'`, `'nox_alchar'`
: （バージョン 3 以降）**JAchar** と **ALchar** との境界．**JAchar** と **ALchar** の間に xkanjiskip が入ることが可能な場合は `'alchar'` が，そうでない場合は `'nox_alchar'` が用いられる（この区別は **ALchar** 側の alxspmode の値によってのみ行われる）．

`'glue'`
: （バージョン 3 以降）**JAchar** と glue，kern 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，次のようにするべきである：
  - 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，hbox 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．

表 16. 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 ∈ [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*⟩ |

- 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=HaranoAjiMincho-Regular: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[*26]．

## 8.6 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[*27]で管理し，それぞれのファミリは \textfont, \scriptfont, \scriptscriptfont という 3 つのフォントを持っている．

LuaTEX-ja の数式中での和文フォントの扱いも同様である．表 16 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．jatextfont 他の第 2 引数 ⟨*jfont_cs*⟩ は，\jfont で定義された**横組用**和文フォントである．\tfont で定義された**縦組用**和文フォントを指定することは想定していない．

## 8.7 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スする

[*26] 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．

[*27] 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．

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_jfont 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 は最低限以下のフィールドを持つが，これらを書き換えてはならない：

size
: 実際に使われるフォントサイズ（sp 単位）．$1\,\mathrm{sp} = 2^{-16}\,\mathrm{pt}$.

zw, zh, kanjiskip, xkanjiskip
: JFM ファイルで指定されているそれぞれの値をフォントサイズに合わせてスケーリングしたものを sp 単位で格納している．

jfm
: 利用されている JFM を識別するための番号．

var
: \jfont, \tfont で指定された jfmvar キーの値（未指定のときは空文字列）．

chars
: 文字コードから文字クラスへの対応が記述されたテーブル．

  JFM 内の [i].chars={⟨*character*⟩, ...} という指定は chars={[⟨*character*⟩]=$i$, ...} という形式に変換されている．

char_type
: $i \in \omega$ に対して，char_type[$i$] は文字クラス $i$ の文字の寸法を格納しており，以下のフィールドを持つ．

  - width, height, depth, italic, down, left は JFM で指定されているそれぞれの値をスケーリングしたものである．

- `align` は JFM で指定されている値によって，

$$\begin{cases} 1 & (\texttt{'right'} \text{ in JFM}), \\ 0.5 & (\texttt{'middle'} \text{ in JFM}), \\ 0 & (\text{otherwise}) \end{cases}$$

のいずれかの値をとる．

$i, j \in \omega$ に対して，`char_type[`$i$`][`$j$`]` は文字クラス $i$ の文字と $j$ の文字の間に挿入されるグルーやカーンの情報を格納している．

間に入るものがカーンであれば，この値は `[`$j$`]={`⟨*kern*⟩`, ratio=`⟨*ratio*⟩`}` であり，⟨*kern*⟩ はカーンの値を sp 単位で表したものである．

一方，間に入るものがグルーであれば，この値は以下のキーを持つテーブルである．

**[1], [2], [3]** グルーのそれぞれ自然長，伸び量，縮み量を sp 単位で表したもの．

**priority**（バージョン 2 以降の）JFM での指定 {⟨*stretch*⟩,⟨*shrink*⟩} を

$$(\langle stretch \rangle + 4) \cdot 8 + \langle shrink \rangle + 4$$

として 0–63 の整数にパックしたもの．

**ratio, kanjiskip_natural, kanjiskip_stretch, kanjiskip_shrink** JFM 中の同名のフィールドの値がそのまま使われている．

`chars_cbcache`

文字クラス決定の処理で．キャッシュとして使われる．

戻り値の `new_jfont_info` テーブルも上に述べたフィールドをそのまま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> char_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以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, <table> char_type)
  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（sp 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，以下の量だけ上にずらされることとなる：

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

# 9 パラメータ

## 9.1 \ltjsetparameter

先に述べたように，LuaTEX-ja の内部パラメータにアクセスするには \ltjsetparameter（または \ltjglobalsetparameter）と \ltjgetparameter を用いる．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．14 章を参照．

\ltjsetparameter と \ltjglobalsetparameter はパラメータを指定するための命令で，key-value リストを引数としてとる．両者の違いはスコープであり，標準では \ltjsetparameter はローカルな設定を行うのに対し，\ltjglobalsetparameter はグローバルな設定を行う．また，他のパラメータ指定と同様に \globaldefs の値にも従う．

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の記号には次の意味がある：

- “∗”：段落や hbox の終端での値がその段落／ hbox 全体で用いられる．
- “†”：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 ={⟨*char_code*⟩,⟨*natural number*⟩}*

文字コードが ⟨*char_code*⟩ の文字が持つ付加的な属性値．バージョン 20120506.0 以降では，⟨*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 ={⟨*char_code*⟩,⟨*penalty*⟩}* [\prebreakpenalty]

文字コード ⟨*char_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．

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い[*28]
- 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い

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．

postbreakpenalty ={⟨*char_code*⟩,⟨*penalty*⟩}* [`\postbreakpenalty`]

文字コード ⟨*char_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．

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`]

tjabaselineshift =⟨*dimen*⟩

talbaselineshift =⟨*dimen*⟩ [`\tbaselineshift`]

jaxspmode ={⟨*char_code*⟩,⟨*mode*⟩}*

文字コードが ⟨*char_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 ={⟨*char_code*⟩,⟨*mode*⟩}* [`\xspcode`]

文字コードが ⟨*char_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別名となっていることに注意する．

autospacing =⟨*bool*⟩ [`\autospacing`]

[*28] 後から指定した方で上書きされる．

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

デフォルト値は paverage である．各々の値による差異の詳細は 16.4 節の「『右空白』の算出」を参照してほしい．

jacharrange =⟨*ranges*⟩

kansujichar =⟨*digit*⟩, ⟨*char_code*⟩* [\kansujichar]

direction =⟨*dir*⟩ (always local)

組方向を変更する \yoko (if ⟨*dir*⟩ = 4), \tate (if ⟨*dir*⟩ = 3), \dtou (if ⟨*dir*⟩ = 1), \utod (if ⟨*dir*⟩ = 11) と同じ役割を持つ．利用可能な状況もこれら 4 命令と同一である．引数 ⟨*dir*⟩ が 4, 3, 1, 11 のいずれでも無いときの動作は未定義である．

## 9.2 \ltjgetparameter

\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．

```
\ltjgetparameter{differentjfm},
\ltjgetparameter{autospacing},
\ltjgetparameter{kanjiskip},
\ltjgetparameter{prebreakpenalty}{`) }.
```

paverage, 1, 0.0pt plus 0.99597pt minus 0.09953pt, 10000.

**\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，空白「 」(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，空白のカテゴリーコードは 10 (space) である．

- 第 1 引数が次のいずれかの場合には，追加の引数は必要ない．

  jcharwidowpenalty, yjabaselineshift, yalbaselineshift, autospacing, autoxspacing, kanjiskip, xkanjiskip, differentjfm, direction

  \ltjgetparameter{autospacing} と \ltjgetparameter{autoxspacing} は，true や false を返すの**ではなく**，1 か 0 のいずれかを返すことに注意，
- 第 1 引数が次のいずれかの場合には，さらに文字コードを第 2 引数としてとる．

  kcatcode, prebreakpenalty, postbreakpenalty, jaxspmode, alxspmode

`\ltjgetparameter{jaxspmode}{...}` や `\ltjgetparameter{alxspmode}{...}` は，preonly などといった文字列ではなく，0 から 3 までの値を返す．

- `\ltjgetparameter{jacharrange}{`⟨*range*⟩`}` は，⟨*range*⟩ が **JAchar** 達の範囲ならば 0 を，そうでなければ 1 を返す．「−1 番の文字範囲」は存在しないが，⟨*range*⟩ に −1 を指定してもエラーは発生しない（1 を返す）．
- 0–9 の数 ⟨*digit*⟩ に対して，`\ltjgetparameter{kansujichar}{`⟨*digit*⟩`}` は，`\kansuji`⟨*digit*⟩ で出力される文字の文字コードを返す．
- `\ltjgetparameter{adjustdir}` は，周囲の vbox の組方向（言い換えれば，`\vadjust` で用いられる組方向）を表す数値を返す．direction と同様に，1 は `\dtou` 方向を，3 は縦組みを，4 は横組みを表す．
- 0–65535 の数 ⟨*register*⟩ に対して，`\ltjgetparameter{boxdir}{`⟨*register*⟩`}` は，`\box`⟨*register*⟩ に格納されているボックスの組方向を表す．もしこのレジスタが空の場合は，0 が返される．
- 次のパラメータ名を `\ltjgetparameter` に指定することはできない．

  jatextfont, jascriptfont, jascriptscriptfont, jacharrange
- `\ltjgetparameter{chartorange}{`⟨*char_code*⟩`}` によって ⟨*char_code*⟩ の属する文字範囲の番号を知ることができる．

  ⟨*char_code*⟩ に 0–127 の値を指定した場合（このとき，⟨*char_code*⟩ が属する文字範囲は存在しない）は −1 が返る．

  そのため，⟨*char_code*⟩ が **JAchar** か **ALchar** かは次で知ることができる：

  ```
  \ltjgetparameter{jacharrange}{\ltjgetparameter{chartorange}{⟨char_code⟩}}
                                                  % 0 if JAchar, 1 if ALchar
  ```
- 返り値が文字列であることから，kanjiskip や xkanjiskip を直接 `\ifdim` を使って比較することは望ましくない．伸び量や縮み量を持っている時には，次はエラーを発生させる：

  ```
  \ifdim\ltjgetparameter{kanjiskip}>\z@ ... \fi
  \ifdim\ltjgetparameter{xkanjiskip}>\z@ ... \fi
  ```

  レジスタに一旦代入するのが良い：

  ```
  \@tempskipa=\ltjgetparameter{kanjiskip} \ifdim\@tempskipa>\z@ ... \fi
  \@tempskipa=\ltjgetparameter{xkanjiskip}\ifdim\@tempskipa>\z@ ... \fi
  ```

## 9.3 `\ltjsetparameter` の代替

原則として各種内部パラメータの設定には `\ltjsetparameter` もしくは `\ltjglobalsetparameter` を用いることになるが，`\ltjsetparameter` の実行には時間がかかるという難点があり，LuaTEX-ja の内部ではより高速に実行できる別の形式を用いている．**本節は一般利用者むけの内容ではない．**

**■kanjiskip, xkanjiskip の設定**　pLaTeX 2ε 新ドキュメントクラスでは，

```
\def\@setfontsize#1#2#3{%
  ...
  \kanjiskip=0zw plus .1zw minus .01zw
  \ifdim\xkanjiskip>\z@
    \if@slide \xkanjiskip=0.1em \else
```

```
    \xkanjiskip=0.25em plus 0.15em minus 0.06em
  \fi
\fi}
```

と，フォントサイズを変更するごとに \kanjiskip, \xkanjiskip を変更している．この \@setfontsize は文書の中で多数回実行されるので．LuaTEX-ja 用に素直に書き換えた

```
\ltjsetparameter{kanjiskip=0\zw plus .1\zw minus .01\zw}
\@tempskipa=\ltjgetparameter{xkanjiskip}
\ifdim\@tempskipa>\z@
  \if@slide
    \ltjsetparameter{xkanjiskip=0.1em}
  \else
    \ltjsetparameter{xkanjiskip=0.25em plus 0.15em minus 0.06em}
  \fi
\fi
```

としたのではタイプセットが遅くなってしまう．そこで，\ltjsetparameter の中で

- \globaldefs の値を読み取る \ltj@setpar@global
- kanjiskip の設定を行う \ltjsetkanjiskip
- xkanjiskip の設定を行う \ltjsetxkanjiskip

を独立させ，ltjsclasses では，

```
\ltj@setpar@global
\ltjsetkanjiskip{\z@ plus .1\zw minus .01\zw}
\@tempskipa=\ltjgetparameter{xkanjiskip}
\ifdim\@tempskipa>\z@
  \if@slide
    \ltjsetxkanjiskip.1em
  \else
    \ltjsetxkanjiskip.25em plus .15em minus .06em
  \fi
\fi
```

としている．\ltj@setpar@global を直前に実行せず，単独で \ltjsetkanjiskip, \ltjsetxkanjiskip を実行することは想定されていないので注意．

# 10　plain でも LaTeX でも利用可能なその他の命令

## 10.1　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

これら 6 命令は内部整数を引数とするが，実行結果は**文字列**であることに注意．

```
\newcount\hoge
\hoge="2423 %"
\the\hoge, \kansuji\hoge\\
\jis\hoge, \char\jis\hoge\\
\kansuji1701
```

9251, 九二五一
12355, ゝ
一七〇一

## 10.2 \inhibitglue, \disinhibitglue

\inhibitglue は発行箇所での JFM 由来グルー／カーンの挿入を抑制する．以下は，ボックスの始めと「あ」の間，「あ」「ウ」の間にグルーが入る特別な JFM を用いた例である．

```
\jfont\g=HaranoAjiMincho-Regular:jfm=test \g
\fbox{\hbox{あウあ\inhibitglue ウ}}
\inhibitglue\par\noindent あ1
\par\inhibitglue\noindent あ2
\par\noindent\inhibitglue あ3
\par ) 4) \inhibitglue 5
\par\hrule\noindent あoff\inhibitglue ice
```

あ　ウあウ
あ　1
あ　2
あ　3
) 4) 5
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**[*29]．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 は kanjiskip, xkanjiskip の挿入は抑制しない．例えば上の例の 6 行目では，「)」と「5」の間には本来は JFM 由来の半角空きが挿入されるはずだが，それが \inhibitglue で無効になったため，xkanjiskip が代わりに挿入されている．
- \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．既に \<が定義されていた場合は，LuaTeX-ja の読み込みで強制的に上書きされるので注意すること．

\disinhibitglue は \inhibitglue の効果を無効化する．言い換えれば，(\inhibitglue で抑制されたはずの))JFM 由来グルー／カーンの挿入を許可する．同じ箇所に \inhibitglue と \disinhibitglue が両方ある場合は，後ろの指定が有効になる．この命令はバージョン 20201224.0 で追加された．

なお，\disinhibitglue もリガチャやカーニングを打ち消すことに注意されたい．これは（\inhibitglue と同様に）whatsit ノードを使って実装されていることによる．

## 10.3 \ltjfakeboxbdd, \ltjfakeparbegin

リスト環境内での \item で始まる各項目などでは，「段落最初の鍵括弧が余計に半角字下げされる」など，JFM にある 'parbdd', 'boxbdd' の指定が見かけ上破綻していることがある．

[*29] この点は TeX Live 2014 での pTeX における \inhibitglue の仕様変更と同じである．

これは TEX が \everypar を用いて段落開始時に記号類や空白などを挿入してしまっているため，段落最初の鍵括弧が実際には段落最初のノードではないことに起因する．以下に例を示した．

```
\parindent1\zw
\noindent あああああああああ\par % for comparison
「あああああああ \par            % normal paragraph

\everypar{\null}
「あああああああ \par            % ???
```

あああああああああ
「あああああああ
「あああああああ

\ltjfakeboxbdd, \ltjfakeparbegin はこの状況を改善する命令である．実際には \everypar の末尾にこれらを追加するという使い方がほとんどになるだろう．

- \ltjfakeparbegin は，実行された箇所が「インデントあり段落の開始」であると LuaTEX-ja の和文処理グルー挿入処理に認識させる．この命令の直前に **JAchar** があった場合，この文字の後ろに入るグルー等の処理については未定義である[*30]．
- \ltjfakeboxbdd は，実行された箇所が「ボックスの先頭と末尾」であると LuaTEX-ja の和文処理グルー挿入処理に認識させる．

例えば，先ほどの例に対して適用すると，次のようになる．

```
\parindent1\zw
\noindent あああああああああ\par % for comparison
「あああああああ \par            % normal paragraph

\everypar{\null\ltjfakeparbegin}
「あああああああ \par
```

あああああああああ
「あああああああ
「あああああああ

## 10.4 \insertxkanjiskip, \insertkanjiskip

TEX で日本語の文章を作成していると，しばしば「手動で和欧文間空白 xkanjiskip を挿入したい」という状況に遭遇する．このような場合，\hskip\ltjgetparameter{xkanjiskip} とするのがよくある対応であったが，これらには次のような難点がある：

- \xkanjiskip は「段落や hbox での終端での値がその段落／ hbox 全体で用いられる」となっているため，\hskip\ltjgetparameter{xkanjiskip} 以降に xkanjiskip の値が変わる場合に対応できない．
- LuaTEX-ja では，\xkanjiskip の自然長が \maxdimen = $(2^{30} - 1)$ sp であった場合，JFM で指定された値を実際に利用することになっているが，それに対応できていない．
- luatexja-adjust（13.3 節）による優先度行長調整では，\hskip\ltjgetparameter{xkanjiskip} は手動で挿入したグルーであるから，自動で挿入された xkanjiskip とは伸縮の優先順序が異なってしまう．

これらの難点に対処した，xkanjiskip をグルーとして手動挿入する命令が \insertxkanjiskip である．これはバージョン 20201224.0 で追加された．以下の実行例に示すように，

[*30] この命令と同等の内容は，\dirrctlua の形で ltjsclasses 内で以前から使われていた．一般ユーザでも利用しやすくするため，バージョン 20170505.0 で新たに命令として定義した．

- 単独で \insertxkanjiskip とした場合は，その時点での xkanjiskip の値を使用する
- 「\insertxkanjiskip late」と late キーワードを後置した場合は，段落／ hbox 終了時にそのときの xkanjiskip の値に自動設定される（段落／ hbox 途中での値は未定義）
- どちらであっても，実行箇所に本来なら自動挿入されるはずの JFM 由来グルー／カーンは挿入されない

となっている．

```
\ltjsetparameter{xkanjiskip=0.25\zw}
あ（%                       0.5\zw  (from JFM)
あ\insertxkanjiskip （%     0.25\zw (xkanjiskip at here)
あ\insertxkanjiskip late （% 0.25\zw (xkanjiskip at EOP)
あa%                        1.25\zw (xkanjiskip at EOP)
\\%
\ltjsetparameter{xkanjiskip=1.25\zw}
あ\insertxkanjiskip （%     1.25\zw (xkanjiskip at here)
あa%                        1.25\zw (xkanjiskip at EOP)
%% At the end of the paragraph (EOP), xkanjiskip is 1.25\zw.
```

```
あ（あ（あ　（あ　a
あ　（あ　a
```

\insertxkanjiskip（または late つき）の短縮形[*31]は LuaTEX-ja では定義していない．短縮形を使いたい人は，面倒でも各自で

```
\protected\def\+{\insertxkanjiskip late}
```

などと定義してほしい．

最後になるが，以上の説明の xkanjiskip をすべて標準の和文間空白 kanjiskip に置き換えた \insertkanjiskip 命令も準備されている．

## 10.5 \ltjdeclarealtfont

\jfont の書式を見ればわかるように，基本的には LuaTEX-ja における 1 つの和文フォントに使用出来る「実際のフォント」は 1 つである．しかし，\ltjdeclarealtfont を用いると，この原則から外れることができる．

\ltjdeclarealtfont は以下の書式で使用する：

\ltjdeclarealtfont⟨*base_font_cs*⟩⟨*alt_font_cs*⟩{⟨*range*⟩}

これは「現在の和文フォント」が ⟨*base_font_cs*⟩ であるとき，⟨*range*⟩ に属する文字は ⟨*alt_font_cs*⟩ を用いて組版される，という意味である．

- ⟨*base_font_cs*⟩, ⟨*alt_font_cs*⟩ は \jfont によって定義された和文フォントである．
- ⟨*range*⟩ は文字コードの範囲を表すコンマ区切りのリストであるが，例外として負数 $-n$ は「⟨*base_font_cs*⟩ の JFM の文字クラス $n$ に属する全ての文字」を意味する．
  ⟨*range*⟩ 中に ⟨*alt_font_cs*⟩ 中に実際には存在しない文字が指定された場合は，その文字に対する設定は無視される．

例えば，\hoge の JFM が LuaTEX-ja 標準の jfm-ujis.lua であった場合，

[*31] ちょうど \inhibitglue の短縮形 \< に対応するもの．

```
\ltjdeclarealtfont\hoge\piyo{"3000-"30FF, {-1}-{-1}}
```

は「\hoge を利用しているとき，U+3000–U+30FF と文字クラス 1（開き括弧類）中の文字だけは \piyo を用いる」ことを設定する．{-1}-{-1} という変わった指定の仕方をしているのは，普通に -1 と指定したのでは正しく −1 と読み取られないというマクロの都合による．

### 10.6 \ltjalchar と \ltjjachar

文字コードが ⟨*char_code*⟩ (≥ 128 = 0x80) の文字を \char プリミティブを使い \char⟨*char_code*⟩ として出力させると，その文字の属する文字範囲（4.1 節参照）によって **ALchar** か **JAchar** か，つまり欧文フォントで出力されるか和文フォントで出力されるかが決まる．

文字範囲の設定を無視し，文字コードが ⟨*char_code*⟩ の文字を強制的に **ALchar**, **JAchar** で出力する命令がそれぞれ \ltjalchar, \ltjjachar である．使用方法は \char と同じく \ltjalchar⟨*char_code*⟩, \ltjjachar⟨*char_code*⟩ とすればよい．LuaTEX-ja 20190926.0 から，⟨*char_code*⟩ が 127 以下の場合でも \ltjjachar⟨*char_code*⟩ が **JAchar** として出力されるようになっている．

以下は 4.1 節に載せた例に，\char の動作などを追加したものである．

```
\gtfamily\large
¶,\char`¶,\ltjalchar`¶,\ltjjachar`¶\\      % default: ALchar
α,\char`α,\ltjalchar`α,\ltjjachar`α\\ % default: JAchar
g,\char`g,\ltjalchar`g,\ltjjachar`g       % ALchar unless \ltjjachar
```

¶,¶,¶, ¶
α, α,α, α
g,g,g, g

## 11 LaTeX 2ε 用の命令

### 11.1 LaTeX 2ε 下での和文フォントの読み込み

バージョン 20190107 以降では，LaTeX 2ε の下で LuaTEX-ja を使用した際に，**横組用和文フォントと縦組み用和文フォントを両方一度に読み込み・選択せずに，実際にそれぞれを使う組方向になったときに行う**という方針にした．これは実際に読み込むフォント数を削減することで，タイプセットにかかる時間と（主に Lua の）メモリ消費を削減するためである ([11]).

- \selectfont は横組用・縦組用和文フォントのうち，現在の組方向で使う方を実際に読み込み（・選択し），そうでない方は「フォントサイズと JFM のみ LuaTEX-ja が把握している状態」（以下，**JFM 把握状態**）とする．
- 組方向変更命令 \yoko, \tate, \dtou, \utod には
  新たな組方向での和文フォントが読み込まれていない（JFM 把握状態）ならば，現在のエンコーディング・ファミリ・シリーズ・シェイプから改めて読み込む（または選択する）
  処理が付け加えられている．もとの「組方向を変更するだけ」の命令は \ltj@@orig@yoko のように ltj@@orig@が前についた命令に保存されている．
- \jfont, \tfont, \DeclareFixedFont で定義された和文フォントはその時点で実際にフォントが読み込まれる．すなわち，以下のコードにおいて，\box0 中の **JAchar** は \HOGE でタイプセットされる．

```
% in horizontal direction (\yoko)
\DeclareFixedFont\HOGE{JT3}{gt}{m}{n}{12} % JT3: for vertical direction
\HOGE
```

```
\setbox0=\hbox{\tate あいう}
```

## 11.2 NFSS2 へのパッチ

LuaTEX-ja の NFSS2 への日本語パッチは pLATEX $2_\varepsilon$ で同様の役割を果たす `plfonts.dtx` をベースに，和文エンコーディングの管理等を Lua で書きなおしたものである．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．

### 追加の長さ変数達

pLATEX $2_\varepsilon$ と同様に，LuaTEX-ja は「現在の和文フォントの情報」を格納する長さ変数

`\cht` (height), `\cdp` (depth), `\cHT` (sum of former two),

`\cwd` (width), `\cvs` (lineskip), `\chs` (equals to `\cwd`)

と，その `\normalsize` 版である

`\Cht` (height), `\Cdp` (depth), `\Cwd` (width),

`\Cvs` (equals to `\baselineskip`), `\Chs` (equals to `\cwd`)

を定義している．なお，`\cwd` と `\zw`，また `\cHT` と `\zh` は一致しない可能性がある．なぜなら，`\cwd`, `\cHT` は文字クラス 0 の和文文字の寸法から決定されるのに対し，`\zw` と `\zh` は JFM に指定された値に過ぎないからである．

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

`\DeclareTateKanjiEncoding{`⟨*encoding*⟩`}{`⟨*text-settings*⟩`}{`⟨*math-settings*⟩`}`

LuaTEX-ja の NFSS2 においては，欧文フォントと和文フォントはそのエンコーディングによってのみ区別される．例えば，OT1 と T1 のエンコーディングは欧文フォントのエンコーディングであり，和文フォントはこれらのエンコーディングを持つことはできない．これらコマンドは横組用・縦組用和文フォントのための新しいエンコーディングをそれぞれ定義する．

`\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 NFSS2 の `\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*⟩ の 1 段展開結果){⟨*ja-cmd*⟩ の 1 段展開結果){⟨*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` の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に対して，そのフォントに対応する「従属欧文」のフォントを後半の 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`

次回（のみ）の `\selectfont` の実行時に，現在の欧文フォントのエンコーディング／ファミリ／…… を，`\DeclareRelationFont` か `\SetRelationFont` で指定された現在の和文フォントに対応する「従属欧文」フォントに変更する．

以下に `\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 Latin Modern Sans Serif (TU/lmss/m/n) に変わっていることがわかる．

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

あいう abc

`\adjustbaseline`

pLaTeX 2ε では，`\adjustbaseline` は縦組時に「M」と「漢」の中心線を一致させるために，`\tbaselineshift` を設定する役割を持っている：

$$\texttt{\textbackslash tbaselineshift} \leftarrow \frac{(h_{\mathrm{M}} + d_{\mathrm{M}}) - (h_{\text{漢}} + d_{\text{漢}})}{2} + d_{\text{漢}} - d_{\mathrm{M}},$$

ここで，$h_a$, $d_a$ はそれぞれ「$a$」の高さ・深さを表す．LuaTeX-ja においても，同じように `\adjustbaseline` は talbaselineshift パラメータの調整処理を行っている（但し「漢」でなく「文字クラス 0 の和文文字」を用いる）．

pLaTeX 2ε では，`\adjustbaseline` で（本節の最初に述べた，小文字で始まる）`\cht`, `\cwd` 設定処理も行っていたが，LuaTeX-ja でも全く同様である．

`\fontfamily{`⟨*family*⟩`}`

元々の LaTeX 2ε 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．詳細は 11.3 節を参照すること．

`\fontshape{`⟨*shape*⟩`}`, `\fontshapeforce{`⟨*shape*⟩`}`

元々の LaTeX 2ε におけるものと同様に，このコマンドは現在の欧文フォントシェイプを `\DeclareFontShapeChangeRule` によるシェイプ更新規則によって変更する．

伝統的には，`\fontshape` は無条件に和文フォントシェイプも変更した．しかし，例えば多くの和文フォントはシェイプが “n” しか持たないことと `\itshape` が `\fontshape` を呼び出すことから，

```
Font shape `JY3/mc/m/it' undefined
using `JY3/mc/m/n' instead on ....
```

といった警告をもたらしてしまっていた．

一方，LuaTeX-ja 20200323.0 以降では，`\fontshape{`⟨*shape*⟩`}`, `\fontshapeforce{`⟨*shape*⟩`}`が和文

```
\DeclareKanjiFamily{JY3}{edm}{}
\DeclareFontShape{JY3}{edm}{m}{n}   {<-> s*HaranoAjiMincho-Regular:jfm=ujis}{}
\DeclareFontShape{JY3}{edm}{m}{fb}  {<-> s*HaranoAjiGothic-Regular:jfm=ujis;color=003FFF;down
    =0.1}{}
\DeclareFontShape{JY3}{edm}{m}{fb2} {<-> s*HaranoAjiGothic-Regular:jfm=ujis;color=FF1900}{}
\DeclareAlternateKanjiFont{JY3}{edm}{m}{n}{JY3}{edm}{m}{fb}{ "4E00-"67FF,{-2}-{-2}}
\DeclareAlternateKanjiFont{JY3}{edm}{m}{n}{JY3}{edm}{m}{fb2}{"6800-"9FFF}
{\kanjifamily{edm}\selectfont
日本国民は、正当に選挙された国会における代表者を通じて行動し、……}
```

日本国民は、正当に選挙された国会における代表者を通じて行動し、……

図 15. \DeclareAlternateKanjiFont の使用例

フォントシェイプを更新するのは，シェイプ更新規則に基づいた値や ⟨*shape*⟩ の少なくとも一つが現在の和文フォントファミリ・シリーズで利用可能なときに限られる．どちらでもなく，和文フォントシェイプが変更されなかった場合には

```
Kanji font shape JY3/mc/m/it' undefined
No change on ...
```

という info（警告でなく）を出力する．

\kanjishape{⟨*shape*⟩}, \kanjishapeforce{⟨*shape*⟩}

…

\DeclareAlternateKanjiFont

{⟨*base-encoding*⟩}{⟨*base-family*⟩}{⟨*base-series*⟩}{⟨*base-shape*⟩}

{⟨*alt-encoding*⟩}{⟨*alt-family*⟩}{⟨*alt-series*⟩}{⟨*alt-shape*⟩}{⟨*range*⟩}

10.5 節の \ltjdeclarealtfont と同様に，前半の 4 引数の和文フォント（基底フォント）のうち ⟨*range*⟩ 中の文字を第 5 から第 8 引数の和文フォントを使って組むように指示する．使用例を図 15 に載せた．

- \ltjdeclarealtfont では基底フォント・置き換え先和文フォントはあらかじめ定義されていないといけない（その代わり即時発効）たが，\DeclareAlternateKanjiFont の設定が実際に効力が発揮するのは，書体変更やサイズ変更を行った時，あるいは（これらを含むが）\selectfont が実行された時である．
- 段落や hbox の最後での設定値が段落／ hbox 全体にわたって通用する点や，⟨*range*⟩ に負数 $-n$ を指定した場合，それが「基底フォントの文字クラス $n$ に属する文字全体」と解釈されるのは \ltjdeclarealtfont と同じである．

この他にも，標準では \DeclareSymbolFont, \SetSymbolFont などの命令で（NFSS2 の枠組みで）数式フォントとして日本語フォントを使えるようにするためのパッチを当てている．

一方，disablejfam オプション指定時には，これらのパッチを当てないので

```
\DeclareSymbolFont{mincho}{JY3}{mc}{m}{n}
\DeclareSymbolFontAlphabet{\mathmc}{mincho}
```

のように設定しても，数式モード中に直に日本語を記述することはできない．$\mathmc{あ}$ のよう

に \mathmc で囲んでもできない．

## 11.3 \fontfamily コマンドの詳細

本節では，\fontfamily⟨*family*⟩ がいつ和文/欧文フォントファミリを変更するかについて解説する．基本的には，⟨*family*⟩ が和文フォントファミリだと認識されれば和文側が，欧文フォントファミリだと認識されれば欧文側が変更される．どちらとも認識されれば和文・欧文の両方が変わることになるし，当然どちらとも認識されないこともある．

**■和文フォントファミリとしての認識**　まず，⟨*family*⟩ が和文フォントファミリとして認識されるかは以下の順序で決定される．これは pLaTeX $2_\varepsilon$ の \fontfamily にとても似ているが，ここでは Lua によって実装している．補助的に「和文フォントファミリではないと認識された」ファミリを格納したリスト $N_{\mathrm{J}}$ を用いる．

1. ファミリ ⟨*family*⟩ が既に \DeclareKanjiFamily によって定義されている場合，⟨*family*⟩ は和文フォントファミリであると認識される．ここで，⟨*family*⟩ は現在の和文フォントエンコーディングで定義されていなくてもよい．
2. ファミリ ⟨*family*⟩ がリスト $N_{\mathrm{J}}$ に既に含まれていれば，それは ⟨*family*⟩ が和文フォントファミリではないことを意味する．
3. もし luatexja-fontspec パッケージが読み込まれていれば，ここで終了であり，⟨*family*⟩ は和文フォントファミリとして認識されないことになる．
   もし luatexja-fontspec パッケージが読み込まれていなければ，和文エンコーディング ⟨*enc*⟩ でフォント定義ファイル ⟨*enc*⟩⟨*family*⟩.fd（ファイル名は全て小文字）が存在するようなものがあるかどうかを調べる．存在すれば，⟨*family*⟩ は和文フォントファミリと認識される（フォント定義ファイルは読み込まれない）．存在しなければ，⟨*family*⟩ は和文フォントファミリでないと認識され，リスト $N_{\mathrm{J}}$ に ⟨*family*⟩ を追加することでそれを記憶する．

**■欧文フォントファミリとしての認識**　同様に，⟨*family*⟩ が和文フォントファミリとして認識されるかは以下の順序で決定される．補助的に「欧文フォントファミリと既に認識された」ファミリのリスト $F_{\mathrm{A}}$ と，「欧文フォントファミリではないと認識された」ファミリを格納したリスト $N_{\mathrm{A}}$ を用いる．

1. ファミリ ⟨*family*⟩ がリスト $F_{\mathrm{A}}$ に既に含まれていれば，⟨*family*⟩ は欧文フォントファミリと認識される．
2. ファミリ ⟨*family*⟩ がリスト $N_{\mathrm{A}}$ に既に含まれていれば，それは ⟨*family*⟩ が欧文フォントファミリではないことを意味する．
3. ある欧文フォントエンコーディング下でファミリ ⟨*family*⟩ が定義されていれば，⟨*family*⟩ は欧文フォントファミリと認識され，リスト $F_{\mathrm{A}}$ に ⟨*family*⟩ を追加することでこのことを記憶する．
4. 最終段階では，欧文エンコーディング ⟨*enc*⟩ でフォント定義ファイル ⟨*enc*⟩⟨*family*⟩.fd（ファイル名は全て小文字）が存在するようなものがあるかどうかを調べる．存在すれば，⟨*family*⟩ は欧文フォントファミリと認識される（フォント定義ファイルは読み込まれない）．存在しなければ，⟨*family*⟩ は欧文フォントファミリと認識されないので，リスト $N_{\mathrm{A}}$ に ⟨*family*⟩ を追加してそのことを記憶する．

表 17. strut

| box | direction | width | height | depth | user command |
|---|---|---|---|---|---|
| \ystrutbox | yoko | 0 | 0.7\baselineskip | 0.3\baselineskip | \ystrut |
| \tstrutbox | tate, utod | 0 | 0.5\baselineskip | 0.5\baselineskip | \tstrut |
| \dstrutbox | dtou | 0 | 0.7\baselineskip | 0.3\baselineskip | \dstrut |
| \zstrutbox | — | 0 | 0.7\baselineskip | 0.3\baselineskip | \zstrut |

また，\DeclareFontFamily が LuaTEX-ja の読み込み後に実行された場合は，第 2 引数（ファミリ名）が自動的に $F_{\mathrm{A}}$ に追加される．

以上の方針は pLATEX $2_\varepsilon$ における \fontfamily にやはり類似しているが，3. が加わり若干複雑になっている．それは pLATEX $2_\varepsilon$ がフォーマットであるのに対し LuaTEX-ja はそうでないため，LuaTEX-ja は自身が読み込まれる前にどういう \DeclareFontFamily の呼び出しがあったか把握できないからである．

**■注意**　さて，引数によっては，「和文フォントファミリとも欧文フォントファミリとも認識されなかった」という事態もあり得る．この場合，引数 ⟨*family*⟩ は不正だった，ということになるので，和文・欧文の両方のフォントファミリを ⟨*family*⟩ に設定し，代用フォントが使われるに任せることにする．

## 11.4　\DeclareTextSymbol 使用時の注意

LATEX (2017/01/01) 以降では，TU エンコーディングが標準となり，特に何もしなくても T1, TS1 エンコーディングで定義されていた記号類が使えるようになった．LuaTEX-ja ではこれらの命令によって記号が欧文フォントで出力されるようにするため，\DeclareTextSymbol 命令を改変し，そして TU エンコーディングの定義である tuenc.def を再読込している．

従来は \DeclareTextSymbol で内部的に定義された \T1\textquotedblleft といった命令は *chardef* トークンであった．しかし前段落で述べた改変によりもはやそうではなくなっており，例えば \TU\textquotedblleft は \ltjalchar8220␣ という定義になっている．

## 11.5　\strutbox

pLATEX 2017/04/08 以降と同じように，\strutbox は現在の組方向によって \ystrutbox, \tstrutbox, \dstrutbox のいずれかに展開される**マクロ**となっている（これらについては表 17 参照）．同様に \strut もこの 3 ボックスのいずれかを組方向によって使い分けるようになっている．

\zstrutbox は utod 方向（pTEX でいう縦数式ディレクション）で使われる支柱ボックスであるが，実際に使われるのは \zstrut が明示的に発行された時，そして lltjext パッケージで追加される組方向指定で <u> を指定した時，および周囲が縦組の状況で <z> を指定した時に限られている．

### 11.6 \verb と xkanjiskip

7 ページで述べたように，バージョン 20260420.0 以降では，\verb や verbatim 環境の内部では xkanjiskip を入れないようにした．これは

- \verb や verbatim 環境で用いられるフォントを決める \verbatim@font 定義の末尾に \ltj@verb@fontsetup を追加
- .dtx 内の macrocode 環境で使われるフォントを決める \MacroFont, \AltMacroFont の末尾にも \ltj@verb@fontsetup を追加
- \ltj@verb@fontsetup を，\ltjsetparameter{autoxspacing=false}と定義

することで行っている．そのため

- 以前の挙動に戻したければ，7 ページで述べたように \ltj@verb@fontsetup に \relax を \let すればよい．
- \verbatim@font が手動/別パッケージにより再定義された際には，例えば次のように末尾に \ltj@verb@fontsetup を追加する必要がある．

```
\makeatletter\apptocmd\verbatim@font{\ltj@verb@fontsetup}{}{}\makeatother
```

## 12 expl3 形式の命令

expl3 の文法に沿った組方向変更命令や組方向による条件判断文である．これらは pLaTeX との互換性の為に用意されているので，platex モジュールとして定義されている．なお，“†” がついている命令は LuaTeX-ja 独自のものである．

\platex_direction_yoko:, \platex_direction_tate:, \platex_direction_dtou:
　それぞれ \yoko, \tate, \dtou と同義.

\platex_if_direction_yoko_p:

\platex_if_direction_yoko:*TF* {⟨*true code*⟩}{⟨*false code*⟩}
　現在の組方向が横組であるか否かをテストする.

\platex_if_direction_tate_nomath_p:†

\platex_if_direction_tate_nomath:*TF*† {⟨*true code*⟩}{⟨*false code*⟩}
　現在の組方向が縦組であるか否かをテストする.

\platex_if_direction_tate_math_p:†

\platex_if_direction_tate_math:*TF*† {⟨*true code*⟩}{⟨*false code*⟩}
　現在の組方向が utod 方向（pTeX でいう「縦数式ディレクション」）であるか否かをテストする.

\platex_if_direction_tate_p:

\platex_if_direction_tate:*TF* {⟨*true code*⟩}{⟨*false code*⟩}
　現在の組方向が縦組または utod 方向であるか否かをテストする.

\platex_if_direction_dtou_p:

\platex_if_direction_dtou:*TF* {⟨*true code*⟩}{⟨*false code*⟩}

現在の組方向が dtou 方向であるか否かをテストする．

\platex_if_box_yoko_p:N ⟨*box*⟩

\platex_if_box_yoko:N*TF* ⟨*box*⟩ {⟨*true code*⟩}{⟨*false code*⟩}

ボックス ⟨*box*⟩ の組方向が横組であるか否かをテストする．

\platex_if_box_tate_nomath_p:N† ⟨*box*⟩

\platex_if_box_tate_nomath:N*TF*† ⟨*box*⟩ {⟨*true code*⟩}{⟨*false code*⟩}

ボックス ⟨*box*⟩ の組方向が縦組であるか否かをテストする．

\platex_if_box_tate_math_p:N† ⟨*box*⟩

\platex_if_box_tate_math:N*TF*† ⟨*box*⟩ {⟨*true code*⟩}{⟨*false code*⟩}

ボックス ⟨*box*⟩ の組方向が utod 方向であるか否かをテストする．

\platex_if_box_tate_p:N ⟨*box*⟩

\platex_if_box_tate:N*TF* ⟨*box*⟩ {⟨*true code*⟩}{⟨*false code*⟩}

ボックス ⟨*box*⟩ の組方向が縦組または utod 方向であるか否かをテストする．

\platex_if_box_dtou_p:N ⟨*box*⟩

\platex_if_box_dtou:N*TF* ⟨*box*⟩ {⟨*true code*⟩}{⟨*false code*⟩}

ボックス ⟨*box*⟩ の組方向が dtou 方向であるか否かをテストする．

# 13 拡張パッケージ

LuaTEX-ja には（動作には必須ではないが）自由に読み込める拡張が付属している．これらは LATEX のパッケージとして制作しているが，luatexja-otf と luatexja-adjust については plain LuaTEX でも \input で読み込み可能である．

## 13.1 luatexja-fontspec

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．

fontspec パッケージで指定可能な各種 OpenType 機能に加えて，和文版のコマンドには以下の「フォント機能」を指定することができる：

CID=⟨*name*⟩, JFM=⟨*name*⟩, JFM-var=⟨*name*⟩, Down=⟨*real*⟩, Left=⟨*real*⟩

これら 5 つのキーはそれぞれ \jfont, \tfont に対する cid, jfm, jfmvar, down, left キーとそれぞれ対応する．この 5 つのキーの詳細は 8.1 節と 8.4 節を参照．

CID キーは下の NoEmbed と合わせて用いられたときのみ有効である．また，横組用 JFM と縦組用 JFM は共用できないため，実際に JFM キーを用いる際は後に述べる YokoFeatures キーや TateFeatures の中で用いることになる．

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．8.4 節を参照．

```
\jfontspec[
  YokoFeatures={Color=FF1900}, TateFeatures={Color=003FFF},
  TateFont=HaranoAjiGothic-Regular
]{HaranoAjiMincho-Regular}
\hbox{\yoko 横組のテスト}\hbox{\tate 縦組のテスト}
\addjfontfeatures{Color=00AF00}
\hbox{\yoko 横組}\hbox{\tate 縦組}
```

横組のテスト
縦組のテスト

横組
縦組

図 16. TateFeatures 等の使用例

```
\jfontspec[
  AltFont={
    {Range="4E00-"67FF, Font=HaranoAjiGothic-Regular, Color=003FFF, Down=0.1},
    {Range="6800-"9EFF, Color=FF1900},
    {Range="3040-"306F, Font=HaranoAjiGothic-Regular, Color=35A16B},
  }
]{HaranoAjiMincho-Regular}
日本国民は、正当に選挙された国会における代表者を通じて行動し、われらとわれらの子孫のために、
諸国民との協和による成果と、わが国全土にわたつて自由のもたらす恵沢を確保し、……
```

日本国民は、正当に選挙された国会における代表者を通じて行動し、われらとわれらの子孫のために、諸国民との協和による成果と、わが国全土にわたつて自由のもたらす恵沢を確保し、……

図 17. AltFont の使用例

Kanjiskip=⟨*bool*⟩

37 ページで説明した \jfont 中での ltjksp 指定と同一の効力を持ち，JFM 中における kanjiskip_natural, kanjiskip_stretch, kanjiskip_shrink キー（44 ページ）の有効/無効を切り替える．標準値は true である．

TateFeatures={⟨*features*⟩}, TateFont=⟨*font*⟩

縦組において使用されるフォントや，縦組においてのみ適用されるフォント機能達を指定する．使用例は図 16 参照．

YokoFeatures={⟨*features*⟩}

同様に，横組においてのみ適用されるフォント機能達を指定する．使用例は図 16 参照．

AltFont

10.5 節の \ltjdeclarealtfont や，11.2 節の \DeclareAlternateKanjiFont と同様に，このキーを用いると一部の文字を異なったフォントや機能たちを使って組むことができる．AltFont キーに指定する値は，次のように二重のコンマ区切りリストである：

```
AltFont = {
  ...
  { Range=⟨range⟩, ⟨features⟩},
  { Range=⟨range⟩, Font=⟨font name⟩, ⟨features⟩ },
  { Range=⟨range⟩, Font=⟨font name⟩> },
  ...
```

}

各部分リストには Range キーが必須である（含まれない部分リストは単純に無視される）．指定例は図 17 に示した．

なお，luatexja-fontspec 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．

**■AltFont, YokoFeatures, TateFeatures 等の制限**　AltFont, YokoFeatures, TateFeatures の各キーはシェイプ別に指定されるべきものであり，内部では BoldFeatures などのシェイプ別の指定は行うことが出来ない．例えば．

```
AltFont = {
  { Font=HogeraMin-Light, BoldFont=HogeraMin-Bold,
    Range="3000-"30FF, BoldFeatures={Color=FF1900} }
}
```

のように指定することは出来ず，

```
UprightFeatures = {
  AltFont = { { Font=HogeraMin-Light, Range="3000-"30FF, } },
},
BoldFeatures = {
  AltFont =  { { Font=HogeraMin-Bold, Range="3000-"30FF, Color=FF1900 } },
}
```

のように指定しなければならない．

一方，AltFont キー内の各リストでは YokoFeatures, TateFeatures 及び TateFont キーを指定することは可能であり．また YokoFeatures, TateFeatures キーの中身に AltFont を指定することができる．

また，図 16 後半部では 6 行目の色指定が効かず，2 行目で指定した YokoFeatures, TateFeatures による色指定が有効になったままである．これは **YokoFeatures, TateFeatures による OpenType 機能指定は組方向に依存しない OpenType 機能の指定より後に解釈される**からである．

## 13.2　luatexja-otf

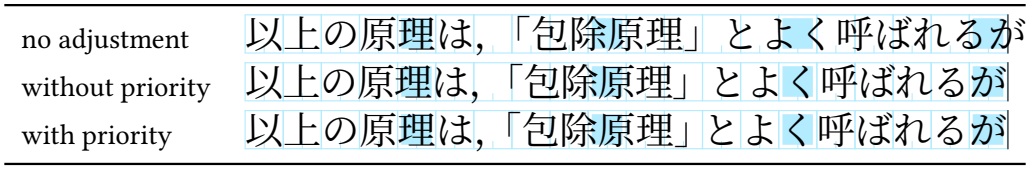
この追加パッケージは CID 番号による文字の出力をサポートする．luatexja-otf は以下の 2 つの低レベルコマンドを提供する：

\CID{⟨*number*⟩}

CID 番号が ⟨*number*⟩ の文字を出力する．もし現在の和文フォントが Adobe-Japan1, Adobe-GB1, Adobe-CNS1, Adobe-Korea1, Adobe-KR のいずれの CID-keyed font でもない場合，⟨*number*⟩ は Adobe-Japan1 の CID 番号であると解釈し「適切なグリフ」[*32]を出力する．

なお，現在の和文フォントが HarfBuzz を用いて読み込まれた場合には，\CID は正しく動作しない．

[*32] 特に縦組用グリフの CID 番号を指定した場合は（LuaTeX-ja 20190504.0 以降では若干改良されているが）意図しない結果になる可能性が高い．なお，バージョン 20190708.0 以降では，CID からグリフへの選択にグリフ名の情報を使用していない．また，フォントに Adobe-Japan1 の IVS が含まれていれば，その情報を用いてグリフを選択する．




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 18. 行長調整

`\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，下の注意を参照すること．

このパッケージは，マクロ集 `luatexja-ajmacros.sty`[*33] も自動的に読み込む．`luatexja-ajmacros.sty` は，そのため，luatexja-otf を読みこめば `ajmacros.sty` マクロ集にある `\aj半角` などのマクロもそのまま使うことができる．

**■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．
- 縦組時には，現在の縦組用和文フォントで `vert`/`vrt2` 機能が有効か無効かを問わず，`\UTF` で出力される文字にはこれらの OpenType 機能が働いた字形になる．
- その他の OpenType 機能（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 パッケージによる処理はこれらの文字には働かない．

**■JFM への記法の追加** luatexja-otf パッケージを読み込むと，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．

この拡張記法は，標準 JFM `jfm-ujis.lua` で，半角ひらがなのグリフ（CID 516–598）を正しく半角幅で組むために利用されている．

## 13.3 luatexja-adjust

この追加パッケージは以下の機能を提供する．詳細な仕様については 19 章を参照してほしい．

**行末文字の位置調整** pTEX では，（是非はともかく）「行末の読点はぶら下げか二分取りか全角取りのいずれかに」のように行末文字と実際の行末の位置関係を 2 通り以上にすることは面倒であった．和文フォントメトリックだけでは「常に行末の読点はぶら下げ」といったことしかできず，前の文に書いたことを実現するには

```
\def\。{%
  \penalty10000 % 禁則ペナルティ
  \hbox to0pt{。\hss}\penalty0 % ぶら下げの場合
  \kern.5\zw\penalty0 % 二分取りの場合
```

[*33] otf パッケージ付属の井上浩一氏によるマクロ集 `ajmacros.sty` に対して漢字コードを UTF-8 にしたり，plain LuaTEX でも利用可能にするという修正を加えたものである．

……だから，①より $\dfrac{a^2}{b} = \dfrac{1+\sqrt{5}}{2}$.
よって $b = \dfrac{1-\sqrt{5}}{2}$ である．
これを②式に代入すると……

(a)

……だから，①より $\dfrac{a^2}{b} = \dfrac{1+\sqrt{5}}{2}$.
よって $b = \dfrac{1-\sqrt{5}}{2}$ である．
これを②式に代入すると……

(b)

図 19. 高い行が連続したときの状況

```
  \kern.5\zw\penalty0 % 全角取りの場合
}
```

のような命令を定義し，文中の全ての句点を \。で書くことが必要だった．

luatexja-adjust パッケージは，上で述べた行末文字と実際の行末との位置関係を 2 通り以上から自動的に選択する機能を提供する．pdfTEX と同じように，「TEX による行分割の後で行末文字の位置を補正する」方法と「行分割の過程で行末文字の位置を考慮に入れる」方法を選べるようにした（luatexja-adjust パッケージの既定では前者）．

**優先順位付きの行長調整** 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 18 上段における半角分の半端は，図 18 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 [7] 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 18 下段）ことになっている．luatexja-adjust パッケージの提供する第 2 の機能は，[5] や [7] における規定のような，優先順位付きの行長調整である．

- 優先度付き行長調整は，段落を行分割した後に個々の行について行われるものである．そのため，行分割の位置は変化することはない．
  \hbox{...} といった「途中で改行できない水平ボックス」では（たとえ幅が指定されていても）無効である．
- 優先度付き行長調整を行うと，和文処理グルーの自然長は変化しないが，伸び量や縮み量は一般に変化する．そのため，既に組まれた段落を \unhbox などを利用して組み直す処理を行う場合には注意が必要である．

**「中身までみた」行送り計算** 複数行に渡る文章を組版するときには行間に空きが入ることが普通である．TEX では各行が一つずつの水平ボックスをなしていることを思い出すと，隣り合った 2 つの行（つまり水平ボックス）の間の空きは次のようにして決まるのだった：

- 「通常に組んだときの行間」$d$ を，\baselineskip から「前の行」の深さと「次の行」の高さを加えたものを引いた値とする．
- $d \geq$ \lineskiplimit の場合，標準の行送り \baselineskip で組んでも十分な間隔があると判断され，2 行の間には長さ $d$ の空白が挿入される．つまり行送りは \baselineskip.
- $d <$ \lineskiplimit の場合，2 行の間には長さ \lineskip の空白が挿入される．そのため（設定値によるが，多くの場合）行送りは \baselineskip より広がる．

ここで，TEX は行送りの決定で「高さ・深さを取っているものが行のどの水平位置にあるか」は一切考慮しないことに注意してほしい．そのため，図 19 (a) のように「必要以上に行間が空いて見える」状況が起こることがある．

luatexja-adjust パッケージでは，「通常に組んだときの行間」$d$ を各行の中身の文字・グルー・ボッ

……だから，①より $\frac{a^2}{b} = \frac{1+\sqrt{5}}{2}$.
よって $b = \frac{1-\sqrt{5}}{2}$ である．
これを②式に代入すると……

(a): 無効

……だから，①より $\frac{a^2}{b} = \frac{1+\sqrt{5}}{2}$.
よって $b = \frac{1-\sqrt{5}}{2}$ である．
これを②式に代入すると……

(b): 0.25\baselineskip 刻み

……だから，①より $\frac{a^2}{b} = \frac{1+\sqrt{5}}{2}$.
よって $b = \frac{1-\sqrt{5}}{2}$ である．
これを②式に代入すると……

(c): 0.5\baselineskip 刻み

……だから，①より $\frac{a^2}{b} = \frac{1+\sqrt{5}}{2}$.
よって $b = \frac{1-\sqrt{5}}{2}$ である．
これを②式に代入すると……

(d): \baselineskip 刻み

図 20. 段階的な行送り増加

クスの寸法を勘案して計算するという方法を利用できるようにした．この機能を使うと，図 19 (b) のように行間の空きが必要以上に大きくなることを避けることができる．

- 段落中の隣り合った二行だけでなく，行間の空きは新たに水平ボックス $h$ を（内部・外部問わず）垂直モードで追加した時にも自動で挿入される．その場合には，前段落で述べた「中身までみる」処理は
  - 現在のリストにおける最後のノード[*34]が水平ボックス $h'$ であり，かつ
  - \prevdepth の値とその $h'$ の深さの値が一致している

  場合にのみ発動するようにしている．
- 行の中身に水平ボックス $h''$ が入ってくることもあるが，その場合は $h''$ の中身の高さ・深さまでは参照せず，あくまでも $h''$ 自身の高さ・深さのみを参照する．参照するようにしてしまうと，\smash など手動で行った高さ・深さ調整の意味がなくなってしまうからである．

なお，現在の実装では，「中身までみる行間調整」は，外部垂直モードにおける前の段落の最終行と次の段落の先頭行との間では満足に動作しないことがある．これについては今後の課題である．

**段階的な行送り調整** 既に述べたように，「通常に組んだときの行間」$d$ が \lineskiplimit より小さい場合，TEX 標準では行間は \lineskip となるのだった．このとき行送りは「前の行の深さ」，「次の行の高さ」，\lineskip の 3 つの和になるわけだが，場合によっては行送りを「\baselineskip の整数倍」などと切りのいい値に揃えたいという状況が考えられなくもない．

luatexja-adjust パッケージでは，$d <$ \lineskiplimit のときに行送りを \baselineskip の linestep_factor 倍ずつ増減させて

行間が \lineskip 以上となるような，最小の $(1 + k \cdot$ linestep_factor$)$\baselineskip（$k$ は整数）の値

とする機能を利用できるようにした．図 20 の (a) がこの機能を無効にした状況で，(b), (c), (d) がそれぞれ linestep_factor を 0.25, 0.5, 1 とした状況である．

[*34] 最後のノードが \parskip によるグルーであった場合のみさらに一つ前のノードを参照する．

なお，この機能は表組時 (\halign, \valign) には無効である．LaTeX における表組環境（tabular, array など）では，\baselineskip, \lineskip はどちらも 0 に設定されているので（代わりに各行に \@arstrut という支柱が入る）ために意味がないことと，数式を内部で表組を使って組む align 環境などではかえって行間が不揃いになってしまうからである．

luatexja-adjust パッケージは，上記で述べた 4 機能を有効化/無効化するための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．

\ltjenableadjust[...]
: ... に指定した key-value リストに従い，「行末文字の位置調整」「優先順位付きの行長調整」「『中身までみた』行送り計算」「段階的な行送り調整」を有効化/無効化する．指定できるキーは以下の通り．

  **lineend=[false,true,extended]** 行末文字の位置調整の機能を無効化 (false)，「行分割後に調整」の形で有効化 (true)，「行分割の過程で考慮」の形で有効化 (extended) する．

  **priority=[false,true]** 優先順位付きの行長調整を無効化 (false)，または有効化 (true).

  **profile=[false,true]**「中身までみた」行送り計算を無効化 (false)，または有効化 (true).

  **linestep=[false,true]** 段階的な行送り調整を無効化 (false)，または有効化 (true).

  どのキーともキー名のみを指定した場合は値として true が指定されたものと扱われる．

  互換性の為，オプション無しでただ \ltjenableadjust が呼び出された場合は，

  ```
  \ltjenableadjust[lineend=true,priority=true]
  ```

  と扱われる．

\ltjdisableadjust
: luatexja-adjust パッケージの機能を無効化する．

  ```
  \ltjenableadjust[lineend=false,priority=false,profile=false,linestep=false]
  ```

  と同義．

また，次のパラメータが \ltjsetparameter 内で追加される．いずれもグローバルに効力を発揮する．

stretch_priority ={⟨*list*⟩} kanjiskip, xkanjiskip，および「**JAglue** 以外のグルー」を，「行を自然長より伸ばす」場合の調整に用いる優先度を指定する．

: 指定方法は，⟨*list*⟩ の中に key-value list の形で

  ```
  stretch_priority={kanjiskip=-35,xkanjiskip=-25,others=50}
  ```

  のようにして行う．キー名 kanjiskip, xkanjiskip についてはそのままの意味であり，others キーが「**JAglue** 以外のグルー」を表す．各キーの値は，JFM グルーにおける「優先度 $i$」を $10i$ に対応させた整数値であり，大きい方が先に伸ばされることを意味している．初期値は

  ```
  {kanjiskip=-35,xkanjiskip=-25,others=50}
  ```

  であり，「優先度 $-4$」と指定されている JFM グルーが最も伸びにくいようになっている．

shrink_priority ={⟨*list*⟩} 同様に，「行を自然長より縮める」場合の調整に用いる優先度を指定する．それ以外は stretch_priority と指定の形式は変わらない（初期値も変わらない）．

linestep_factor =⟨*float*⟩ 段階的な行送り調整の際，\baselineskip の自然長の何倍単位で行送りを変えるかを指定する．0 を指定すると無効になるのと変わらない．また負数を指定すると，その絶対値が指定されたかのように扱われる．初期値は 0.5（つまり半行単位）である．

profile_hgap_factor =⟨*float*⟩「中身まで見た」行送り計算の際，前の行にある深さが大きいものと次の行にある高さが大きいものが水平方向にどれだけ離れていないといけないかを「\lineskip の自然長の何倍か」で指定する．負数を指定すると，その絶対値が指定されたかのように扱われる．初期値は 1（つまり \lineskip（の自然長））である．

さらに，バージョン 20220211.0 以降では次の命令が提供される．

\ltjghostbeforejachar

LuaTEX-ja 本体が提供している \ltjfakeparbegin, \ltjfakeboxbdd と類似の命令である．実行された箇所が（限定・非限定を問わず）水平モードであった場合に，実行された箇所は「文字クラス 0 の **JAchar**」の直前であると，LuaTEX-ja の和文処理グルー挿入処理に認識させる．以下の実行例を参照．

```
\ltjsetparameter{kanjiskip=14pt,xkanjiskip=50pt}
\let\LG=\ltjghostbeforejachar
A\LG  B   \par% ==> ALchar--(xkanjiskip)--\LG
A\LG  字  \par% ==> ALchar--(xkanjiskip)--\LG
漢\LG B   \par% ==> JAchar--(kanjiskip)--\LG
漢\LG 字  \par% ==> JAchar--(kanjiskip)--\LG
```

A B
A 字
漢 B
漢 字

\ltjghostafterjachar

\ltjghostbeforejachar と対を成す命令で，実行された箇所は「文字クラス 0 の **JAchar**」の直後であると，LuaTEX-ja の和文処理グルー挿入処理に認識させる．以下の実行例を参照．

```
\ltjsetparameter{kanjiskip=14pt,xkanjiskip=50pt}
\let\LG=\ltjghostafterjachar
A\LG  B   \par% ==> \LG--(xkanjiskip)--ALchar
漢\LG B   \par% ==> \LG--(xkanjiskip)--ALchar
A\LG  字  \par% ==> \LG--(kanjiskip)--JAchar
漢\LG 字  \par% ==> \LG--(kanjiskip)--JAchar
```

A B
漢 B
A 字
漢 字

なお，バージョン 20220207.0 で追加された \ltjghostjachar は実装にバグがあったのと「両側」という点が扱いづらかったので，将来は削除する予定である．

両命令の主な仕様用途は和文ゴーストでの使用である．BXghost パッケージ ([12]) などでは伝統的に全角空白 (U+3000) と \kern-1\zw を組み合わせた方法が使われてきたが，LuaTEX-ja では全角空白を使っただけではうまくいかない可能性があるため，新たに命令が用意された．

## 13.4 luatexja-ruby

この追加パッケージは，LuaTEX-ja の機能を利用したルビ（振り仮名）の組版機能を提供する．前後の文字種に応じた前後への自動進入や，行頭形・行中形・行末形の自動的な使い分けが特徴である．

ルビ組版に設定可能な項目や注意事項が多いため，本追加パッケージの詳細な説明は使用例と共に luatexja-ruby.pdf という別ファイルに載せている．この節では簡単な使用方法のみ述べる．

**グループルビ** 標準ではグループルビの形で組まれる．第 1 引数に親文字，第 2 引数にルビを記述する．

```
東西線\ruby{妙典}{みようでん}駅は……\\
東西線の\ruby{妙典}{みようでん}駅は……\\
東西線の\ruby{妙典}{みようでん}という駅……\\
東西線\ruby{葛西}{かさい}駅は……
```

東西線妙典(みようでん)駅は……
東西線の妙典(みようでん)駅は……
東西線の妙典(みようでん)という駅……
東西線葛西(かさい)駅は……

この例のように，標準では前後の平仮名にルビ全角までかかるようになっている．

**モノルビ** 親文字を 1 文字にするとモノルビとなる．2 文字以上の熟語をモノルビの形で組みたい場合は，面倒でもその数だけ \ruby を書く必要がある．

```
東西線の\ruby{妙}{みよう}\ruby{典}{でん}駅は……
```

東西線の妙(みよう)典(でん)駅は……

**熟語ルビ** 引数内の縦棒 | はグループの区切りを表し，複数グループのルビは熟語ルビとして組まれる．[7] にあるように，どのグループでも「親文字」が対応するルビ以上の長さの場合は各グループごとに，そうでないときは全体をまとめて 1 つのグループルビとして組まれる．[5] で規定されている組み方とは異なるので注意.

```
\ruby{妙|典}{みよう|でん}\
\ruby{葛|西}{か|さい}\
\ruby{神楽|坂}{かぐら|ざか}
```

妙典(みようでん) 葛西(かさい) 神楽坂(かぐらざか)

複数ルビではグループとグループの間で改行が可能である．

```
\vbox{\hsize=6\zw\noindent
  \hbox to 2.5\zw{}\ruby{京|急|蒲|田}{けい|きゆう|かま|た}
  \hbox to 2.5\zw{}\ruby{京|急|蒲|田}{けい|きゆう|かま|た}
  \hbox to   3\zw{}\ruby{京|急|蒲|田}{けい|きゆう|かま|た}
}
```

京急(けいきゅう)
蒲田(かまた)
京急蒲田(けいきゅうかまた)
京急(けいきゅう)
蒲田(かまた)

また，ルビ文字のほうが親文字よりも長い場合は，自動的に行頭形・行中形・行末形のいずれか適切なものを選択する．

```
\vbox{\hsize=8\zw\noindent
\null\kern3\zw ……を\ruby{承}{うけたまわ}る
    \kern1\zw ……を\ruby{承}{うけたまわ}る\\
\null\kern5\zw ……を\ruby{承}{うけたまわ}る
}
```

……を承(うけたまわ)る
……を承(うけたまわ)る
……を承(うけたまわ)る

## 13.5 lltjext

pLaTeX では縦組用の拡張として plext パッケージが用意されていたが，それを LuaTeX-ja 用に書きなおしたものが本 lltjext パッケージである．

従来の plext パッケージとの違いは，

- シンプル化のため，array パッケージを常に読み込むことにした．
- 組方向オプション <y>（横組），<t>（縦組），<z> の他に <d>（dtou 方向），<u>（utod 方向）を追加した．<z> と <u> の違いは，<z> が（plext パッケージと同様に）周囲の組方向が縦組のときに

しか意味を持たない[*35]のに対し，<u> にはそのような制限がないことである．

- 連数字用命令 \rensuji における位置合わせオプション [l], [c], [r] の挙動を若干変更した．

念の為，本 lltjext パッケージで追加・変更している命令の一覧を載せておく．

tabular, array, minipage 環境

これらの環境は，

```
\begin{tabular}<dir>[pos]{table spec} ... \end{tabular}
\begin{array}<dir>[pos]{table spec}   ... \end{array}
\begin{minipage}<dir>[pos]{width}     ... \end{minipage}
```

のように，組方向オプション <dir> が拡張されている．既に述べたように，組方向オプションに指定できる値は以下の 5 つであり，それ以外を指定した時や無指定時は周囲の組方向と同じ組方向になる．

**y** 横組 (\yoko)

**t** 縦組 (\tate)

**z** 周囲が縦組の時は utod 方向，それ以外はそのまま

**d** dtou 方向

**u** utod 方向

\parbox<⟨*dir*⟩>[⟨*pos*⟩]{⟨*width*⟩}{⟨*contents*⟩}

\parbox 命令も同様に，組方向の指定ができるように拡張されている．

\pbox<⟨*dir*⟩>[⟨*width*⟩][⟨*pos*⟩]{⟨*contents*⟩}

組方向 ⟨*dir*⟩ で ⟨*contents*⟩ の中身を LR モードで組む命令である．⟨*width*⟩ が正の値であるときは，ボックス全体の幅がその値となる．その際，中身は ⟨*pos*⟩ の値に従い，左寄せ (l)，右揃え (r)，中央揃え（それ以外）される．

picture 環境

図表作成に用いる picture 環境も，

```
\begin{picture}<dir>(x_size, y_size)(x_offset,y_offset)
  ...
\end{picture}
```

と組方向が指定できるように拡張されている．$x$ 成分の増加方向は字送り方向，$y$ 成分の増加方向は行送り方向の**反対方向**となる．plext パッケージと同様に内部ではベースライン補正（yalbaselineshift パラメータなど）の影響を受けないように，\put, \line, \vector, \dashbox, \oval, \circle もベースライン補正を受けないように再定義されている．

\rensuji[⟨*pos*⟩]{⟨*contents*⟩}, \rensujiskip

\Kanji{⟨*counter_name*⟩}

\kasen{⟨*contents*⟩}, \bou{⟨*contents*⟩}, \boutenchar

[*35] 周囲の組方向が縦組以外のときは，<z> を指定しても中身の組方向は周囲の組方向と変わらない．

表 18. lltjext パッケージにおける表組・\parbox 命令他の揃え位置

| ↓中身＼周囲→ | \yoko | \tate | \utod | \dtou |
|---|---|---|---|---|
| \yoko | A | B | B | B |
| \tate | B | A | D | C |
| \utod | B | D | A | C |
| \dtou | B | C | C | A |

参照番号

**■表組他の揃え位置**　表組（array, tabular 環境），\parbox 命令，\minipage 環境の揃え位置については表 18 を参照．pLaTeX 2017-07-29 とできるだけ同じ挙動になるようにしている．表 18 中の A–D の意味は次の通り．

**A** 周囲の組方向と中身の組方向が同じ場合．
- [t] 指定のとき：中身の先頭行のベースラインが周囲のベースラインと一致する．表組で先頭行の上に罫線があった場合は，それがベースラインの位置[*36]となる．
- [c] 指定のとき：中身の上下の中心が周囲の数式の軸を通る．
- [b] 指定のとき：中身の最終行のベースラインが周囲のベースラインと一致する．表組で最終行の下に罫線があった場合は，それがベースラインの位置となる．

**B** 周囲の組方向と中身の組方向が 90 度ずれている場合．
- [t] 指定のとき：表組においては，上端が周囲のベースラインと一致する．\parbox や \minipage 環境においては，上端が周囲の和文文字の上端と一致する．
- [c] 指定のとき：中身の上下の中心が周囲の数式の軸を通る．
- [b] 指定のとき：表組においては，下端が周囲のベースラインと一致する．\parbox や \minipage 環境においては，下端が周囲の和文文字の下端と一致する．

**C** 周囲の組方向と中身の組方向が 180 度ずれている場合．\parbox や \minipage 環境においては，上の B の場合と同じ挙動である．表組においては，A で [t] と [b] を入れ替えた
- [t] 指定のとき：中身の最終行のベースラインが周囲のベースラインと一致する．最終行の下に罫線があった場合は，それがベースラインの位置となる．
- [c] 指定のとき：中身の上下の中心が周囲の数式の軸を通る．
- [b] 指定のとき：中身の先頭行のベースラインが周囲のベースラインと一致する．表組で先頭行の上に罫線があった場合は，それがベースラインの位置となる．

**D** 通常の縦組 (\tate) と「縦数式ディレクション」に相当する \utod 方向が絡んだ場合．\parbox や \minipage 環境においては，上の B の場合と同じ挙動である．表組においては，
- [t] 指定のとき：中身の先頭行の欧文ベースラインが周囲の欧文ベースラインと一致する．
- [c] 指定のとき：中身の上下の中心が周囲の数式の軸を通る．
- [b] 指定のとき：中身の最終行の欧文ベースラインが周囲の欧文ベースラインと一致する．

[*36] LuaTeX-ja では和文側のベースラインの位置も上下移動できることに注意．そのため「和文ベースライン」の位置に来るとは限らない．

## 13.6 luatexja-preset

3.3 節で述べたように，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（多書体化）と八登崇之氏による PXchfon パッケージの一部（プリセット指定）とを合わせたような格好をしている．

パッケージ読み込み時に渡されたオプションのうち，本節にないものを指定した場合，それらはそのまま luatexja-fontspec パッケージに渡される[*37]．例えば，下の 1–3 行目は 5 行目のように一行にまとめることができる．

```
\usepackage[no-math]{fontspec}
\usepackage[match]{luatexja-fontspec}
\usepackage[kozuka-pr6n]{luatexja-preset}
%%--------
\usepackage[no-math,match,kozuka-pr6n]{luatexja-preset}
```

### 13.6.1 一般的なオプション

**fontspec（既定）**

luatexja-fontspec パッケージの機能を用いて和文フォントを選択する．これは，fontspec パッケージが自動で読み込まれることを意味する．

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[*38]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}
```

**nfssonly**

LaTeX 標準のフォント選択機構 (NFSS2) を用いて `ltjpm`*n*（明朝），`ltjpg`*n*（ゴシック），それに後に述べる `deluxe` オプションが指定された場合には `ltjpmg`*n*（丸ゴシック）という和文フォントファミリを定義[*39]し，これらを用いる．

本オプション指定時には fontspec・luatexja-fontspec パッケージは自動では読み込まれない，しかし，

```
\usepackage{fontspec}
\usepackage[hiragino-pron,nfssonly]{luatexja-preset}
```

のようにすれば，このオプションを指定すれば欧文フォントを fontspec パッケージの機能を使って指定することができる．

一方，luatexja-preset パッケージ読み込み時に既に luatexja-fontspec パッケージが読み込まれている場合は `nfssonly` オプションは無視される．

**match**

このオプションが指定されると，「pLaTeX 2$\varepsilon$ 新ドキュメントクラス」のように `\rmfamily`,

[*37] `nfssonly` オプションが指定されていた場合は，luatexja-fontspec パッケージは読み込まれないので単純に無視される．

[*38] 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．

[*39] *n* は自然数であり，`\ltjapplypreset` の実行（この命令は luatexja-preset パッケージ読み込み時に自動的に実行される）ごとに増加していく．

\textrm{...}, \sffamily 等が欧文フォントだけでなく和文フォントも変更するようになる．fontspec オプションが有効になっている場合は，このオプションは luatexja-fontspec パッケージへと渡される．

nodeluxe（既定）

deluxe オプションの否定．LaTeX 2ε 環境下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．

deluxe

明朝体・ゴシック体各 3 ウェイトと，丸ゴシック体 (\mgfamily, \textmg{...}) を利用可能にする．明朝体は細字・中字・太字の 3 ウェイトがあり，明朝体の細字は \mcfamily\ltseries で利用できる．また，ゴシック体は中字・太字・極太の 3 ウェイトがあり，ゴシック体の極太は \gtfamily\ebseries で利用できる[*40]．

- プリセット設定によっては明朝体細字が用意されていないものもある．その場合は明朝体中字が代用される．
- 明朝体細字，ゴシック体極太，丸ゴシック体の 3 フォントについては実際にフォントをロードする前に存在するかチェックを行う．存在しなかったものについては警告を発し，それぞれ明朝体中字．ゴシック体太字，ゴシック体太字で代用する．

expert

横組・縦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[*41]．

bold

nodeluxe オプション指定時には，「明朝の太字」をゴシック体と同じフォントにする．deluxe オプション指定時には，「明朝の太字」「ゴシック体の中字」をゴシック体の太字と同じフォントにする．

jis90, 90jis

出来る限り JIS X 0208:1990 の字形を使う．

jis2004, 2004jis

出来る限り JIS X 0213:2004 の字形を使う．

jfm_yoko=⟨*jfm*⟩

横組用和文フォントで用いる JFM を jfm-⟨*jfm*⟩.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．

jfm_tate=⟨*jfm*⟩

縦用和文フォントで用いる JFM を jfm-⟨*jfm*⟩.lua にする．このオプションがない時は LuaTeX-ja 標準の jfm-ujisv.lua が用いられる．

jis

jfm_yoko=jis と同じ．ここで用いる JFM jfm-jis.lua は JIS フォントメトリックを元にしたものである．

jis90, 90jis, jis2004, 2004jis 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

[*40] 過去との互換性のため，\gtebfamily, \textgteb{...} も依然として利用可能である．

[*41] \rubyfamily とはいいつつ，実際にはフォントファミリを切り替えるのではない（通常では OpenType 機能の有効化であり，nfssonly 指定時にはシェイプを rb に切り替える）．

シック体）にのみ有効である．これら 4 オプションのうち複数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．

### 13.6.2 多ウェイト用プリセットの一覧

bizud, haranoaji, 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．以下の表において，*つきのフォント (e.g., KozGo...-Regular) は，**deluxe オプション指定時に**ゴシック体中字として用いられるものを示している．

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 | light | KozMinPro-Light | KozMinProVI-Light | KozMinPr6N-Light |
| | 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 |
| | extra bold | KozGoPro-Heavy | KozGoProVI-Heavy | KozGoPr6N-Heavy |
| 丸ゴシック | | KozGoPro-Heavy | KozGoProVI-Heavy | KozGoPr6N-Heavy |

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

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

極太ゴシック体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．

なお，明朝体細字として用いるヒラギノ明朝体 W2 は OS X にはバンドルされておらず，別途購入する必要がある．

| family | series | hiragino-pro | hiragino-pron |
|---|---|---|---|
| 明朝 | light | Hiragino Mincho Pro W2 | Hiragino Mincho ProN W2 |
| | 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 |
| | extra bold | Hiragino Kaku Gothic Std W8 | Hiragino Kaku Gothic StdN W8 |
| 丸ゴシック | | Hiragino Maru Gothic Pro W4 | Hiragino Maru Gothic ProN W4 |

bizud BIZ UD fonts (by Morisawa Inc.) bundled with Windows 10 October 2018 Update.

| family | series | |
|---|---|---|
| 明朝 | | BIZ-UDMinchoM.ttc |
| ゴシック | medium | BIZ-UDGothicR.ttc |
| | bold | BIZ-UDGothicB.ttc |
| | extra bold | BIZ-UDGothicB.ttc |
| 丸ゴシック | | BIZ-UDGothicB.ttc |

`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 |
| | extra bold | A-OTF-MidashiGoPro-MB31.otf | A-OTF-MidashiGoPr6N-MB31.otf |
| 丸ゴシック | | A-OTF-Jun101Pro-Light.otf | A-OTF-ShinMGoPr6N-Light.otf |

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

`yu-win10` Yu fonts bundled with Windows 10.

`yu-osx` Yu fonts bundled with OSX Mavericks.

| family | series | yu-win | yu-win10 | yu-osx |
|---|---|---|---|---|
| 明朝 | light | YuMincho-Light | YuMincho-Light | (YuMincho Medium) |
| | medium | YuMincho-Regular | YuMincho-Regular | YuMincho Medium |
| | bold | YuMincho-Demibold | YuMincho-Demibold | YuMincho Demibold |
| ゴシック | medium | YuGothic-Regular* | YuGothic-Regular* | YuGothic Medium* |
| | | YuGothic-Regular | YuGothic-Medium | YuGothic Medium |
| | bold | YuGothic-Bold | YuGothic-Bold | YuGothic Bold |
| | extra bold | YuGothic-Bold | YuGothic-Bold | YuGothic Bold |
| 丸ゴシック | | YuGothic-Bold | YuGothic-Bold | YuGothic Bold |

`moga-mobo` MogaMincho, MogaGothic, and MoboGothic.

`moga-mobo-ex` MogaExMincho, MogaExGothic, and MoboExGothic. これらのフォントは http://yozvox.web.fc2.com/ からダウンロードできる.

| family | series | default, `90jis` option | `jis2004` option |
|---|---|---|---|
| 明朝 | medium | Moga90Mincho | MogaMincho |
| | bold | Moga90Mincho Bold | MogaMincho Bold |
| ゴシック | medium | Moga90Gothic | MogaGothic |
| | bold | Moga90Gothic Bold | MogaGothic Bold |
| | extra bold | Moga90Gothic Bold | MogaGothic Bold |
| 丸ゴシック | | Mobo90Gothic | MoboGothic |

`moga-mobo-ex` オプション指定時には MogaEx90Mincho などの Ex が名前についたフォントが使われる.

`ume` Ume Mincho and Ume Gothic. これらのフォントは https://ja.osdn.net/projects/ume-font/wiki/FrontPage からダウンロードできたが，現在ではリンク切れである．

| family | series | default |
|---|---|---|
| 明朝 | medium | Ume Mincho |
| | bold | Ume Mincho |
| ゴシック | medium | Ume Gothic* |
| | | Ume Gothic O5 |
| | bold | Ume Gothic O5 |
| | extra bold | Ume Gothic O5 |
| 丸ゴシック | | Ume Gothic O5 |

`sourcehan` Source Han Serif and Source Han Sans fonts (Language-specific OTF *or* OTC)

`sourcehan-jp` Source Han Serif JP and Source Han Sans JP fonts (Region-specific Subset OTF)

| family | series | `sourcehan` | `sourcehan-jp` |
|---|---|---|---|
| 明朝 | light | Source Han Serif Light | Source Han Serif JP Light |
| | medium | Source Han Serif Regular | Source Han Serif JP Regular |
| | bold | Source Han Serif Bold | Source Han Serif JP Bold |
| ゴシック | medium | Source Han Sans Regular* | Source Han Sans JP Regular* |
| | | Source Han Sans Medium | Source Han Sans JP Medium |
| | bold | Source Han Sans Bold | Source Han Sans JP Bold |
| | extra bold | Source Han Sans Heavy | Source Han Sans JP Heavy |
| 丸ゴシック | | Source Han Sans Medium | Source Han Sans JP Medium |

`noto-otc` Noto Serif CJK and Noto Sans CJK fonts (OTC)

`noto-otf`, `noto` Noto Serif CJK and Noto Sans CJK fonts (Language-specific OTF)

`noto-jp` Noto Serif CJK and Noto Sans CJK fonts (Region-specific subset OTF)

| family | series | `noto-otc` | `noto-otf`, `noto` | `noto-jp` |
|---|---|---|---|---|
| 明朝 | light | Noto Serif CJK Light | Noto Serif CJK JP Light | Noto Serif JP Light |
| | medium | Noto Serif CJK Regular | Noto Serif CJK JP Regular | Noto Serif JP Regular |
| | bold | Noto Serif CJK Bold | Noto Serif CJK JP Bold | Noto Serif JP Bold |
| ゴシック | medium | Noto Sans CJK Regular* | Noto Sans CJK JP Regular* | Noto Sans JP Regular* |
| | | Noto Sans CJK Medium | Noto Sans CJK JP Medium | Noto Sans JP Medium |
| | bold | Noto Sans CJK Bold | Noto Sans CJK JP Bold | Noto Sans JP Bold |
| | extra bold | Noto Sans CJK Black | Noto Sans CJK JP Black | Noto Sans JP Black |
| 丸ゴシック | | Noto Sans CJK Medium | Noto Sans CJK JP Medium | Noto Sans JP Medium |

`haranoaji` Harano Aji Fonts. これらのフォントは https://github.com/trueroad/HaranoAjiFonts からダウンロードできる．「原ノ味丸ゴシック」は存在しないので，便宜的に原ノ味角ゴシック Heavy によって代用している．

| family | series | haranoaji |
|---|---|---|
| 明朝 | light | HaranoAjiMincho-Light.otf |
| | medium | HaranoAjiMincho-Regular.otf |
| | bold | HaranoAjiMincho-Bold.otf |
| ゴシック | medium | HaranoAjiGothic-Regular.otf* |
| | | HaranoAjiGothic-Medium.otf |
| | bold | HaranoAjiGothic-Bold.otf |
| | extra bold | HaranoAjiGothic-Heavy.otf |
| 丸ゴシック | | HaranoAjiGothic-Medium.otf |

### 13.6.3 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 設定では明朝体太字・丸ゴシック体はゴシック体と同じフォントが用いられる．

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

### 13.6.4 HG フォントを使うプリセット

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．以下の表では，*付きのフォント（例：IPA ゴシック*）は jis2004 と nodeluxe のいずれかのオプションが有効になっているときに使われる．

| family | series | ipa-hg | ipaex-hg | ms-hg |
|---|---|---|---|---|
| 明朝 | medium | IPA 明朝 | IPAex 明朝 | MS 明朝 |
| | bold | HG 明朝 E | HG 明朝 E | HG 明朝 E |
| ゴシック | medium | IPA ゴシック* | IPAex ゴシック* | MS ゴシック* |
| | | HG ゴシック M | HG ゴシック M | HG ゴシック M |
| | bold | HG ゴシック E | HG ゴシック E | HG ゴシック E |
| | extra bold | HG 創英角ゴシック UB | HG 創英角ゴシック UB | HG 創英角ゴシック UB |
| 丸ゴシック | | HG 丸ゴシック M-PRO | HG 丸ゴシック M-PRO | HG 丸ゴシック M-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）
**jis90, 90jis 指定時** ファイル名 (hgrme.ttc, hgrge.ttc, hgrsgu.ttc, hgrsmp.ttf)
**jis2004, 2004jis 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．

### 13.6.5 新たなプリセットの定義

バージョン 20170904.0 以降では，自分で新たなプリセットを定義することが出来るようになった．以下に説明する 2 命令はプリアンブルでしか実行できない．

\ltjnewpreset{⟨*name*⟩}{⟨*specification*⟩}

新たに ⟨*name*⟩ という名称のプリセットを定義する．この名称は，すでに定義されているプリセット名や，13.6.1 で定義されているオプション，さらに次の 13 個と重複してはならない．

```
mc mc-l mc-m mc-b mc-bx gt gt-u gt-d gt-m gt-b gt-bx gt-eb mg-m
```

⟨*specification*⟩ は，プリセット名や以下のキー達のコンマ区切りリストを指定する：

**mc-l=**⟨*font*⟩ 明朝体細字 (\mcfamily\ltseries)

**mc-m=**⟨*font*⟩ 明朝体中字 (\mcfamily\mdseries)

**mc-b=**⟨*font*⟩ 明朝体太字 (\mcfamily\bfseries)

**mc-bx=**⟨*font*⟩ mc-b=⟨*font*⟩ と同義．

**gt-u=**⟨*font*⟩ deluxe オプション未指定時のゴシック体 (\gtfamily)・明朝体太字

**gt-d=**⟨*font*⟩ deluxe オプション指定時のゴシック体中字 (\gtfamily\mdseries)

**gt-m=**⟨*font*⟩ deluxe オプションの指定の有無に関係なくゴシック体中字 (\gtfamily\mdseries) を指定する．「gt-u=⟨*font*⟩，gt-d⟨*font*⟩」と同義．

**gt-b=**⟨*font*⟩ ゴシック体太字 (\gtfamily\bfseries)

なお，パッケージ読み込み時に bold オプションが指定された場合は，mc-b=⟨*font*⟩ を指定したことにもなる．

**gt-bx=**⟨*font*⟩ gt-b=⟨*font*⟩ と同義．

**gt-eb=**⟨*font*⟩ ゴシック体太字 (\gtfamily\ebseries)

**mg-m=**⟨*font*⟩ 丸ゴシック体 (\mgfamily)

**mc=**⟨*font*⟩ 明朝体の細字・中字・太字全部を設定．以下を指定したことと同じである：

mc-l=⟨*font*⟩，mc-m=⟨*font*⟩，mc-b=⟨*font*⟩

**gt=**⟨*font*⟩ ゴシック体の中字・太字・極太全部を設定．以下を指定したことと同じである：

gt-u=⟨*font*⟩，gt-d=⟨*font*⟩，gt-b=⟨*font*⟩，gt-eb=⟨*font*⟩

\ltjnewpreset*{⟨*name*⟩}{⟨*specification*⟩}

\ltjnewpreset とほぼ同じであるが，こちらはすでに定義されているプリセット名を ⟨*name*⟩ に指定した場合にはエラーを出さずに定義を置き換える．

\ltjapplypreset{⟨*name*⟩}

⟨*name*⟩ で指定されたプリセットを使って和文フォントを設定する．

なお，\ltjnewpreset の第二引数 ⟨*specification*⟩ に含まれるプリセット名は \ltjnewpreset の時点で定義されている必要はなく，\ltjapplypreset で実際に使うときに定義されていれば良い．そのため，次のような記述も可能である：

```
\ltjnewpreset{hoge}{piyo,mc-b=HiraMinProN-W6}
\ltjnewpreset{piyo}{mg-m=HiraMaruProN-W4}
\ltjapplypreset{hoge}
```

**■注意** \ltjnewpreset で定義したプリセットには以下の制限がある．

- 非埋め込みのフォントを指定することはできない．
- ipa-hg などのいくつかのプリセットでは「90jis, jis2004 が指定されているか否かでフォントの指定を変える」処理が行われていたが，\ltjnewpreset で定義したプリセットではこの処理は働かない．HG フォントやモガ明朝などを使うプリセットを定義する場合には注意すること．

# 第 III 部
# 実装

## 14　パラメータの保持

### 14.1　LuaTEX-ja で用いられるレジスタと 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` は写真植字で用いられた 1 Q = 0.25 mm（「級」とも書かれる）を格納している．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．
なお，TEX では長さはスケールド・ポイント ($2^{-16}$ pt) を最小単位としており，実際の値は 46616 sp ≃ 0.249994662mm である[*42]．そのため，次のように若干の誤差が出ることは気をつけてほしい．

```
1 \dimen0=1000\jQ \the\dimen0, % ==> 46616000 sp
2 \dimen0=250mm   \the\dimen0  % ==> 46616995 sp
```

711.30371pt, 711.3189pt

`\jH` (dimension) 同じく写真植字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．この `\jH` は `\jQ` と同じ寸法レジスタを指す．

`\ltj@dimen@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．`\zw` 命令は，このレジスタを適切な値に設定した後，「このレジスタ自体を返す」．

`\ltj@dimen@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．`\zh` 命令は，このレジスタを適切な値に設定した後，「このレジスタ自体を返す」．

`\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) 基本的には現在の横組用和文フォントのフォント番号を格納しているが，LATEX 下で使用する場合は（−2 以下の）負数となることがある．負数の場合は「横組用和文フォントは実際には読み込まれておらず，そのフォントサイズと JFM だけが LuaTEX-ja が把握している」状態を表す．

`\ltj@curtfnt` (attribute) 縦組用和文フォントに関する `\ltj@curjfnt` と同様の値．

`\ltj@charclass` (attribute) **JAchar** の文字クラス．**JAchar** が格納された *glyph_node* でのみ使われる．

`\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．この属性が「未設定」(`-"7FFFFFFF`) のときは 0 であるとみなされる．`\ltj@ykblshift` 他も同様．

`\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@tablshift` (attribute)

[*42] 0.25 mm ≃ 46616.99527 sp なので 46617 sp ≃ 0.250000025 mm の方が近いが，TEX で「`\dimen0=0.25mm`」とすると，`\dimen0` の値は 46616 sp となる．

`\ltj@tkblshift` (attribute)

`\ltj@autospc` (attribute) そのノードで kanjiskip の自動挿入が許されるかどうか．0 は「許可しない」，0 以外の値（「未設定」も含む）は「許可する」．

`\ltj@autoxspc` (attribute) そのノードで xkanjiskip の自動挿入が許されるかどうか．0 は「許可しない」，0 以外の値（「未設定」も含む）は「許可する」．

`\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 (\/) によるカーン，または luaotfload によって挿入されたフォントのカーニング情報由来のカーン．これらのカーンは通常の `\kern` とは異なり，**JAglue** の挿入処理においては透過する．

***packed* (2)**

***kinsoku* (3)** 禁則処理のために挿入されたペナルティ．

***from_jfm*–(*from_jfm* + 63) (4–67)** JFM 由来のグルー／カーン．

***kanji_skip* (68), *kanji_skip_jfm* (69)** 和文間空白 kanjiskip を表すグルー．

***xkanji_skip* (70), *xkanji_skip_jfm* (71)** 和欧文間空白 xkanjiskip を表すグルー．

***processed* (73)** LuaTEX-ja の内部処理によって既に処理されたノード．

***ic_processed* (74)** イタリック補正に由来するグルーであって，既に **JAglue** 挿入処理にかかったもの．

***boxbdd* (75)** hbox か段落の最初か最後に挿入されたグルー／カーン．

***special_jaglue* (76)** `\insert[x]kanjiskip` 由来のグルー．

また，挿入処理の結果であるリストの最初のノードでは，`\ltj@icflag` の値に *processed_begin_flag* (4096) が追加される．これによって，`\unhbox` が連続した場合でも「ボックスの境界」が識別できるようになっている．

`\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．

`\ltj@dir` (attribute) *direction* whatsit（後述）において組方向を示すために，あるいは *dir_box* の組方向を用いる．*direction* whatsit においては値は

*dir_dtou* (1), *dir_tate* (3), *dir_yoko* (4), *dir_utod* (11)

のいずれかであり，*dir_box* ではこれらに次を加えた値をとる（22 章参照）．

***dir_node_auto* (128)** 異なる組方向に配置するために自動的に作られたボックス．

***dir_node_manual* (256)** `\ltjsetwd` によって「ボックスの本来の組方向とは異なる組方向での寸法」を設定したときに，それを記録するためのボックス．

TEX 側から見える値，つまり `\the\ltj@dir` の値は常に 0 である．

`\ltjlineendcomment` (counter) LuaTEX-ja は **JAchar** で入力行が終了した場合，その直後にコメント文字をおくことで余計な空白が挿入されることを防いでいる．`\ltjlineendcomment` はその際のコメント文字の Unicode における符号位置を指定する（詳細は 15.2 節を参照）．

LuaTEX-ja における既定値は `"FFFFF` = 1048575 であり，ユーザは内部動作を熟知していない限りこのカウンタの値を変更してはならない．`\ltjlineendcomment` の値が Unicode の範囲外（負や，`"10FFFF` = 1114111 を超えた場合）にくることは想定されていない．

さらに，LuaTEX-ja はいくつかの user-defined whatsit node を内部処理に用いる．*direction* whatsit

はノードリストを格納するが，それ以外の whatsit ノードの type は 100 であり，ノードは自然数を格納している．user-defined whatsit を識別するための user_id は luatexbase.newuserwhatsitid により確保されており，下の見出しは単なる識別用でしかない．

***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ネストレベルを表す．

***char_by_cid*** luaotfload による処理が適用されない **JAchar** のためのノードで，value フィールドに文字コードが格納されている．この種類の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．\CID, \UTF でこの種類のノードが利用されている．

***replace_vs*** 上の *char_by_cid* と同様に，これらのノードは luaotfload のコールバックによる処理が適用されない **ALchar** のためものである．

***begin_par***「段落の開始」を意味するノード．list 環境，itemize 環境などにおいて，\item で始まる各項目は……

***direction***

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

## 14.2 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{%
3   \ltjsetparameter{kanjiskip=5pt}ほげほげ}
4 \box0.ぴよぴよ\par
```

ふがふが. ほ げ ほ げ. ぴよぴよ

9.1 節で述べたように，ある hbox 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 の実装を観察すると，この 5 pt という長さはどのコールバックからも知ることはできないことがわかる．LuaTEX のソースファイルの 1 つ tex/packaging.w の中に，以下のコードがある：

```
1226 void package(int c)
1227 {
1228     scaled h;                   /* height of box */
1229     halfword p;                 /* first node in a box */
1230     scaled d;                   /* max depth */
1231     int grp;
1232     grp = cur_group;
1233     d = box_max_depth;
1234     unsave();
1235     save_ptr -= 4;
1236     if (cur_list.mode_field == -hmode) {
1237         cur_box = filtered_hpack(cur_list.head_field,
```

```
1238                                    cur_list.tail_field, saved_value(1),
1239                                    saved_level(1), grp, saved_level(2));
1240        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 コールバック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[*43]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 にスタックレベル，\ltj@@group@level に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 では次の手法を用いる：スタックレベルが増加するするとき，type, subtype, value がそれぞれ 44 (*user_defined*), *stack_marker*，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 hbox 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 hbox group の直後の TEX のグループレベルを $t$ とすると：

- もしその hbox の中身を表すリストの中に *stack_flag* ノードがなければ，hbox 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 hbox 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 hbox の中で代入が起こったことになる．したがって，hbox 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，hbox 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．

## 14.3　スタックシステムで使用される関数

本節では，ユーザが LuaTEX-ja のスタックシステムを使用して，TEX のグルーピングに従うような独自のデータを取り扱う方法を述べる．

スタックに値を設定するには，以下の Lua 関数を呼び出せば良い：

```
luatexja.stack.set_stack_table(<any> index, <any> data)
```

直感的には，スタックテーブル中のインデックス *index* の値を *data* にする，という意味である．*index* の値としては nil と NaN 以外の任意の値を使えるが，自然数は LuaTEX-ja が使用する（将来の拡張用

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

```
\protected\def\ltj@setpar@global{%
  \relax\ifnum\globaldefs>0\directlua{luatexja.isglobal='global'}%
    \else\directlua{luatexja.isglobal=''}\fi
}
\protected\def\ltjsetparameter#1{%
  \ltj@setpar@global\setkeys[ltj]{japaram}{#1}\ignorespaces}
\protected\def\ltjglobalsetparameter#1{%
  \relax\ifnum\globaldefs<0\directlua{luatexja.isglobal=''}%
    \else\directlua{luatexja.isglobal='global'}\fi%
  \setkeys[ltj]{japaram}{#1}\ignorespaces}
```

図 21. パラメータ設定命令の定義

も含む）ので，ユーザが使用する場合は負の整数値か文字列の値にすることが望ましい．また，ローカルに設定されるかグローバルに設定されるかは，`luatexja.isglobal` の値に依存する（グローバルに設定されるのは，`luatexja.isglobal == 'global'` であるちょうどその時）．

スタックの値は，

```
luatexja.stack.get_stack_table(<any> index, <any> default, <number> level)
```

の戻り値で取得できる．*level* はスタックレベルであり，通常は `\ltj@@stack` の値を指定することになるだろう．*default* はレベル *level* のスタックに値が設定されていなかった場合に返すデフォルト値である．

## 14.4 パラメータの拡張

ここでは，luatexja-adjust で行なっているように，`\ltjsetparameter`, `\ltjgetparameter` に指定可能なキーを追加する方法を述べる．

**■パラメータの設定** `\ltjsetparameter` と，`\ltjglobalsetparameter` の定義は図 21 ののようになっている．本質的なのは最後の `\setkeys` で，これは xkeyval パッケージの提供する命令である．

このため，`\ltjsetparameter` に指定可能なパラメータを追加するには，⟨*prefix*⟩ を `ltj`，⟨*family*⟩ を `japaram` としたキーを

```
\define@key[ltj]{japaram}{...}{...}
```

のように定義すれば良いだけである．なお，パラメータ指定がグローバルかローカルかどうかを示す `luatexja.isglobal` が，

$$\texttt{luatexja.isglobal} = \begin{cases} \texttt{'global'} & \text{（パラメータ設定はグローバル），} \\ \texttt{''} & \text{（パラメータ設定はローカル）．} \end{cases}$$

として自動的にセットされる[*44]．

**■パラメータの取得** 一方，`\ltjgetparameter` は Lua スクリプトによって実装されている．値を取得するのに追加引数の要らないパラメータについては，`luatexja.unary_pars` 内に処理内容を記述した関数を定義すれば良い．例えば，Lua スクリプトで

[*44] 命令が `\ltjglobalsetparameter` かどうかだけではなく，実行時の `\globaldefs` の値にも依存して定まる．

```
function luatexja.unary_pars.hoge (t)
  return 42
end
```

を実行すると，`\ltjgetparameter{hoge}` は 42 という**文字列**を返す．関数 `luatexja.unary_pars.hoge` の引数 $t$ は，14.2 節で述べた LuaTEX-ja のスタックシステムにおけるスタックレベルである．戻り値はいかなる値であっても，最終的には文字列として出力されることに注意．

一方，追加引数（**数値しか許容しない**）が必要なパラメータについては，まず Lua スクリプトで処理内容の本体を記述しておく：

```
function luatexja.binary_pars.fuga (c, t)
  return tostring(c) .. ', ' .. tostring(42)
end
```

引数 $t$ は，先に述べた通りのスタックレベルである．一方，引数 $c$ は `\ltjgetparameter` の第 2 引数を表す数値である．しかしこれだけでは駄目で，

```
\ltj@@decl@array@param{fuga}
```

を実行し，TEX インターフェース側に「`\ltjgetparameter{fuga}` は追加引数が必要」ということを通知する必要がある．

# 15　和文文字直後の改行

## 15.1　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．

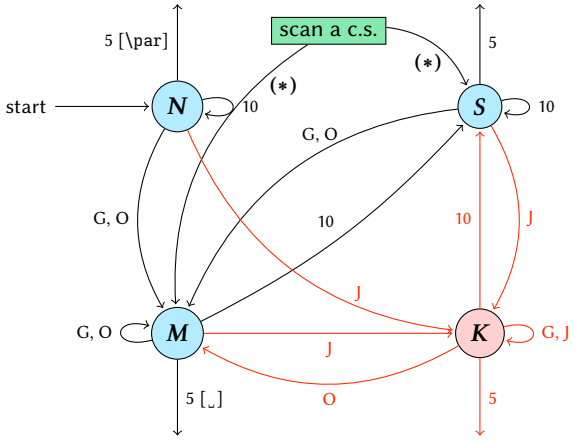
- State $N$: 行の開始.
- State $S$: 空白読み飛ばし.
- State $M$: 行中.
- State $K$: 行中（和文文字の後）.

また，状態遷移は，図 22 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．




**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 22. pTEX の入力処理部の状態遷移

## 15.2 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 `\ltjlineendcomment` 番の文字[*45]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[*46]のカテゴリーコードが 5 (*end-of-line*) である．
> 2. `\ltjlineendcomment` 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：
>
> $$(\text{any char})^*\Big(\mathbf{JAchar} \cap \big(\{\text{catcode} = 11\} \cup \{\text{catcode} = 12\}\big)\Big)\big(\{\text{catcode} = 1\} \cup \{\text{catcode} = 2\}\big)^*$$

[*45] `\ltjlineendcomment` の既定値は"FFFFF であるので，既定では U+FFFFF が使われることになる．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．
[*46] 普通は，改行文字（文字コード 13 番）である．

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．条件 1. は，lstlisting 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，pTEX と完全に同じ挙動が実現できたわけではない．次のように，**JAchar** の範囲を変更したちょうどその行においては挙動が異なる：

```
1 \fontspec[Ligatures=TeX]{TeX Gyre Termes}
2 \ltjsetparameter{autoxspacing=false}
3 \ltjsetparameter{jacharrange={-6}}xあ              x yzい u
4 y\ltjsetparameter{jacharrange={+6}}zい
5 u
```

上ソース中の「あ」は **ALchar**（欧文扱い）であり．ここで使用している欧文フォント TEX Gyre Termes は「あ」を含まない．よって，出力に「あ」は現れないことは不思議ではない．それでも，pTEX とまったく同じ挙動を示すならば，出力は「x yzいu」となるはずである．しかし，実際には上のように異なる挙動となっているが，それは以下の理由による：

- 3 行目を process_input_buffer で処理する時点では，「あ」は **JAchar**（和文扱い）である．よって 3 行目は **JAchar** で終わることになり，\ltjlineendcomment 番のコメント文字が追加される．よって，直後の改行文字は無視されることになり，空白は入らない．
- 4 行目を process_input_buffer で処理する時点では，「い」は **ALchar** である．よって 4 行目は **ALchar** で終わることになり，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，**JAchar** 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．

## 15.3 濁点・半濁点付き仮名の正規化→ luaotfload v3.19 以降ではそちらで

TEX Live 2016 以降の (u)pTEX では，合成用濁点 (U+3099)・合成用半濁点 (U+309A) を用いて表現された平仮名・片仮名を合成済み文字に変換するという処理を行っている．この処理を行っている要因としては，

- 無用なトラブルを避けるため．濁点・半濁点付きの仮名文字が「合成用濁点・半濁点を使って入力されているか」「最初から合成済み文字で入力されているか」を見た目から判別することは難しい．
- pTEX との互換性のため．pTEX は内部コードが JIS X 0208 の範囲に限られるため，合成用濁点・半濁点は利用できない．そのため上記の変換処理はさらに前から行われていた．

LuaTEX(-ja) では入力の変換は基本的に行わず，文字の合成は使用しているフォントの OpenType 機能に委ねるという立場であったが，luaotfload v3.19 以降では，標準で NFC への Unicode 正規化を行っている．そのため，バージョン 20230409.0 以降では，LuaTEX-ja による自前の変換[*47]は行わないようにしている．

[*47] バージョン 20220103.0 で実装した．

# 16　JFM グルーの挿入，kanjiskip と xkanjiskip

## 16.1　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，hbox 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．

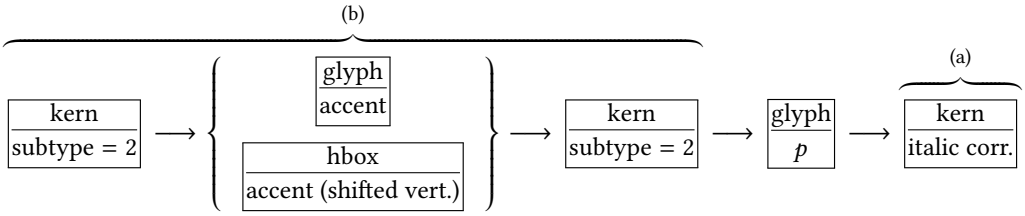
しかし，LuaTEX-ja では，hbox へのパッケージングや行分割前に全ての **JAglue**，即ち JFM グルー・xkanjiskip・kanjiskip の 3 種類を一度に挿入することになっている．これは，LuaTEX において欧文の合字・カーニング処理がノードベースになったことに対応する変更である．

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．

## 16.2　「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 hbox から `\unhbox` でアンパックされたものである．この場合，クラスタの *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．この場合，クラスタの *id* は *id_math* である．
3. **JAchar** を表す *glyph_node* $p$ とそれに関係するノード：
   (a) $p$ のイタリック補正のためのカーン．
   (b) `\accent` による $p$ に付随したアクセント．
   (c) OpenType の `palt` 機能などに由来する，$p$ の位置補正を行うための $p$ の直前/直後に配置されたカーン．これらの subtype は 0.




この場合の *id* は *id_jglyph* である．
4. **ALchar** を表す *glyph_node*，`\accent` によるアクセント位置補正用のカーン (subtype が 2)，そし

てイタリック補正・カーニングによって挿入されたカーン達が連続したもの．この場合の *id* は *id_glyph* である．

5. 水平ボックス (hbox)，垂直ボックス，罫線 (\vrule)，そして *unset_node*．クラスタの *id* は垂直に移動していない hbox ならば *id_hlist*，そうでなければ *id_box_like* となる．
6. グルー，subtype が 2 (*accent*) ではないカーン，そして discretionary break．その *id* of the cluster 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*** **JAchar**（和文文字）．

*Np.head*, *Np.tail* は，その **JAchar** を表している *glyph_node* そのものである．

***id_glyph*** **JAchar**（和文文字）以外のものを表す *glyph_node p*．

多くの場合，*p* は **ALchar**（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 hbox．

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 hbox 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** hbox 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 hbox，`\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 **JAchar**．*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 hbox の中身の先頭として出現した **JAchar**．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／ hbox の中身として出現している「**JAchar** 以外の文字」．次の 3 つの場合が該当：

- *id* が *id_glyph* である．
- *id* が *id_math* である（つまりこのクラスタは 1 つの文中数式をなす）．
- *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．

## 16.3　段落／ hbox の先頭や末尾

**■先頭部の処理**　まず，段落／ hbox の一番最初にあるクラスタ *Np* を探索する．hbox 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 hbox(subtype = 3)
- *subtype* が 44 (*user_defined*) でないような whatsit

これは，`\parindent` 由来の hbox 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 hbox 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 hbox）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.

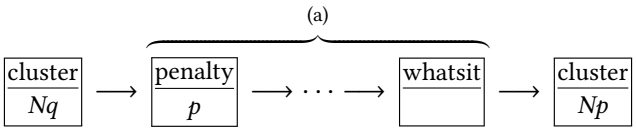
**■末尾の処理**　末尾の処理は，問題のリストが段落のものか hbox のものかによって異なる．後者の場合は容易い：最後のクラスタを *Nq* とおくと，*Nq* と「文字コード `'boxbdd'` の文字」の間に入るグルー／カーンを，*Nq* の直後に挿入するのみである．

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．段落の最後の「通常の **JAchar** ＋　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．

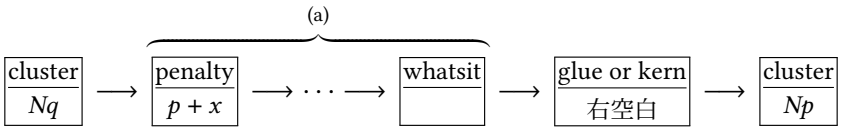
ペナルティ量を増やす場所は，*head* が **JAchar** であり，かつその文字の kcatcode が偶数であるような最後のクラスタの直前にあるものたちである[*48]．

## 16.4　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．

**■カーニングの算出（横組）**　クラスタ *Np* の先頭が subtype 0 のカーン *k* である場合，*k* は次の 2 つを合計した量である：

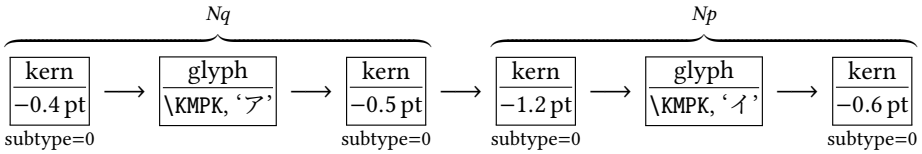
- *Nq.last*, *Np.head* という 2 つの **JAchar** の間に入るカーニング
- OpenType の `palt` 機能などによる，*Np.head* の位置補正

例えば

```
\jfont\KMPK = KozMinPr6N-Regular.otf:jfm=prop;+palt;+kern at 10pt
\KMPK アイ
```

[*48] 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．

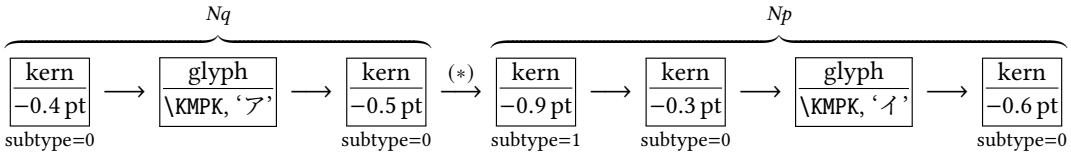
からは，luaotfload により次のノード列が得られる：




(1)

ここで，subtype の値が 0 であるカーンは行分割でも除去されない．また 3 つ目のカーン −1.2 pt は，カーニング −0.9 pt と「イ」の位置補正 −0.3 pt の和である．「ア」「イ」間で行分割が起こる場合，前者 −0.9 pt 分はなくなってほしいが，後者 −0.3 pt 分は残ることが望ましい．

そのため，LuaTEX-ja では (1) というノード列を

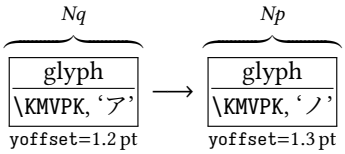



と変換し，(∗) の箇所に JFM グルーを挿入することになる．

**■カーニングの算出（縦組）** luaotfload パッケージは，vpal, vkrn などの縦組時に用いられる OpenType 機能を *glyph_node* の yoffset フィールド（17.1 小節を参照）の値を増減することで「実装」している．これも，例えば

```
\tate
\tfont\KMVPK = KozMinPr6N-Regular.otf:jfm=propv;+vpal;+vkrn at 10pt
\KMVPK アノ
```

からは，次のノード列が得られる：




(2)

各 *glyph_node* の yoffset の値は

(i) OpenType の vpal, vhal 機能によるグリフ高さの補正（狭める方向が正）
(ii) OpenType の vkrn, vapk 機能によって入るカーニング（詰める方向が正）
(iii) その他

の 3 種類に分かれる．例えば，(2) における 1 つ目のノード「ア」の yoffset の値は

(i) 0.6 pt　　(ii) 0.7 pt（「ア」「イ」間のカーニング）　　(iii) 上に 0.1 pt ずらす（vpal 機能由来）

の和である．LuaTEX-ja は (i), (ii) の補正量を認識し，正しい位置になるように調整する．

**■「右空白」の算出**　次に，「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 **JAchar** 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，JFM・`jfmvar`・サイズの 3 つ組は *Nq* と *Np* で異なる．この場合，まず

   *gb* := (*Nq* と「使用フォントが *Nq* のそれと同じで，
   文字コードが *Np* のそれの文字」との間に入るグルー／カーン)
   *ga* := (「使用フォントが *Np* のそれと同じで，
   文字コードが *Nq* のそれの文字」と *Np* との間に入るグルー／カーン)

   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 ⟨*ratio*⟩ 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，⟨*ratio*⟩ 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，⟨*ratio*⟩ 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：

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

     ここで．$f(x,y)$ は

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

   - differentjfm がそれ以外の値の時は，⟨*ratio*⟩ 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：

     $$f(gb,ga)$$

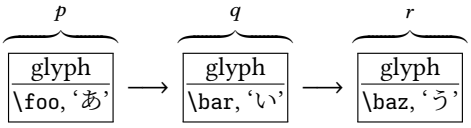
     ここで．$f(x,y)$ は

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

例えば，

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

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．

なお，JFM で `kanjiskip_natural`, `kanjiskip_stretch`, `kanjiskip_shrink` キーが指定されていた場合は，……

**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 **JAchar** 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 パラメータが両方とも 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 パラメータの自然長が `\maxdimen` $= (2^{30} - 1)$ sp でなければ，kanjiskip パラメータの値を持つ 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 **JAchar**（**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．

**■禁則用ペナルティの挿入**　まず，

$a :=$ (*Nq*[*49] の文字に対する postbreakpenalty の値) + (*Np*[*50] の文字に対する prebreakpenalty の値)

とおく．ペナルティは通常 $[-10000, 10000]$ の整数値をとり，また $\pm 10000$ 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（$\pm 10000$ 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．
- そうでないないときは，$a \neq 0$ ならば *penalty_node* を作って挿入する．

[*50] 厳密にはそれぞれ *Nq.tail*, *Np.head*.

表 19. JFM グルーの概要

| *Np* ↓ | 和文 A | 和文 B | 欧文 | 箱 | glue | kern |
|---|---|---|---|---|---|---|
| 和文 A | $\frac{M \to K}{PN}$ | $\frac{O_A \to K}{PN}$ | $\frac{N_A \to X}{PN}$ | $\frac{O_A}{PA}$ | $\frac{O_A}{PN}$ | $\frac{O_A}{PS}$ |
| 和文 B | $\frac{O_B \to K}{PA}$ | $\frac{K}{PS}$ | $\frac{X}{PS}$ | | | |
| 欧文 | $\frac{N_B \to X}{PA}$ | $\frac{X}{PS}$ | | | | |
| 箱 | $\frac{O_B}{PA}$ | | | | | |
| glue | $\frac{O_B}{PN}$ | | | | | |
| kern | $\frac{O_B}{PS}$ | | | | | |

上の表において，$\frac{M \to K}{PN}$ 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. 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設定するために「P-normal [PN]」の方法を採用する．

## 16.5 その他の場合

本節の内容は表 19 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**欧文境界** B [$N_B$] 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．
- 禁則用ペナルティも，以前述べた P-normal [PN] と同じである．

**欧文境界 B [$N_B$]** 以下で求めた量を「右空白」として採用する．この処理は JFM-origin [M] の変種と考えて良く，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），未定義．
2. そうでなければ，*Nq* と「文字コードが $x$ の文字」との間に入るグルー／カーンとする．この $x$ は次の場合分けによる：
   - *Np.id* が *id_math* のとき（つまりクラスタ *Np* が文中数式を表す）ときは，$x = -1$.
   - *Np* の中身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) パラメータが偶数）ときは，$x$ = `'nox_alchar'`.

- 以上のいずれでもないときは，$x$ = 'alchar'.

**xkanjiskip [X]** この段階では，xkanjiskip [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．\inhibitglue は効力を持たない．

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 パラメータが共に 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) パラメータが 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) パラメータが偶数）．
2. ユーザ側から見た xkanjiskip パラメータの自然長が \maxdimen = $(2^{30} - 1)$ sp でなければ，xkanjiskip パラメータの値を持つ 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，**欧文境界** B [$N_B$] の部分が変わるだけである．

- 「右空白」については，まず以下に述べる**欧文境界** 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．
- 禁則用ペナルティは，以前述べた P-normal [PN] と同じである．

**欧文境界 A [$N_A$]** これは**欧文境界** B [$N_B$] で *Np* と *Nq* の役割が交換されたものと思えば良い．この処理で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），未定義．
2. そうでなければ，「文字コードが $x$ の文字」と *Np* との間に入るグルー／カーンと定める．$x$ は *Nq* から**欧文境界** B [$N_B$] におけるそれと同じ方法で定めるが，'nox_alchar' か 'alchar' は
   *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) パラメータが 2 以上）．
   か否かで判断する．

**■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，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，「右空白」は挿入されない．
- 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，

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

**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．

**Boundary-B [$O_B$]** この処理は**欧文境界** B [$N_B$] と同様であり，$x$ が次によって決まることのみが異なる：

- *Np* がグルーやカーンのときは，$x$ = `'glue'`.
- そうでない（*Np* が**箱**）ときは，$x$ = `'jcharbdd'`.

**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，以下のいずれかの場合に $a$ 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．
- $a \neq 0$ のときは，「右空白」がグルーであっても *penalty_node* を作る．

**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．

**■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）場合と同じである．

- 「右空白」については，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，「右空白」は挿入されない．
- 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，

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

**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．

**Boundary-A [$O_A$]** この処理は**欧文境界** A [$N_A$] と同様であり，$x$ が次によって決まることのみが異なる：

- *Nq* がグルーやカーンのときは，$x$ = `'glue'`.
- そうでない（*Nq* が**箱**）ときは，$x$ = `'jcharbdd'`.

**■和文 A と和文 B の違い**　先に述べたように，**和文** B は hbox の中身の先頭（or 末尾）として出現している **JAchar** である．リスト内に直接ノードとして現れている **JAchar**（**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 **由来** [M]，Boundary-A [$O_A$] など）の挿入は行われない．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**欧文境界** B [$N_B$] 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（hbox の中身の末尾として登場しているから）ので，そもそも**欧文境界** B [$N_B$] 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**欧文境界** B [$N_B$] 由来の半角アキが入ることになる．

# 17　ベースライン補正の方法

## 17.1　`yoffset` フィールド

yalbaselineshift 等のベースライン補正は，基本的には対象となっている *glyph_node* の `yoffset` フィールドの値を増減することによって実装されている．なお，`yoffset` の値は上方向への移動量であるのに対し，yalbaselineshift などは下方向への移動量である．

さて，`yoffset` の増減によって見かけのグリフ位置は上下に移動するが，仮想ボディの高さ $h$，深さ $d$ については

**`yoffset`** $\geq 0$ **のとき**　$h = \max(\texttt{height} + \texttt{yoffset}, 0), d = \max(\texttt{depth} - \texttt{yoffset}, 0),$
**`yoffset`** $< 0$ **のとき**　$h = \max(\texttt{height} + \texttt{yoffset}, 0), d = \texttt{depth}.$

という仕様になっている．つまり，`yoffset` が負（グリフを下げる）の場合に深さは増加しない（表 20 参照）．

表 20. yoffset and imaginary body

| **yoffset** | 10 pt | 5 pt | 0 | −5 pt | −10 pt |
|---|---|---|---|---|---|
| 仮想ボディ |  |  |  |  |  |

### 17.2　ALchar の補正

上記の問題について，**ALchar** のベースライン補正では「正しい深さ」を持った罫線 (rule) を補うという対応策をとった．この罫線による補正は，*id* が *id_glyph* であるクラスタ単位，大雑把に言えば音節単位で行われる．文字列 “Typeset” を

- フォントは Latin Modern Roman (`lmroman10-regular.otf`) 10 pt
- yalbaselineshift は 5 pt

という状況で組んだ場合を例にとって説明しよう．

LuaTEX・luaotfload によるカーニング・ハイフネーションが終わった段階では，……

## 18　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 ([4]) 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．出力命令としては，アクティブ文字化した `\ltjlineendcomment` を用いている．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 は，listings 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．

### 18.1　注意

**■異体字セレクタの扱い**　`lstlisting` 環境などの内部にある異体字セレクタを扱うため，lltjp-listings では `vsraw` と `vscmd` という 2 つのキーを追加した．しかし，lltjp-listings が実際に読み込まれるのは `\begin{document}` のところであるので，プリアンブル内ではこれらの追加キーは使用できない．

`vsraw` は，ブール値の値をとるキーであり，標準では false である．

- true の場合は，異体字セレクタは「直前の文字に続けて」出力されるため，例えば以下の例（左側は入力，右側はその出力）のようになる．

```
\begin{lstlisting}[vsraw=true]
葛城市，葛飾区，葛西
\end{lstlisting}
```

```
葛城市，葛飾区，葛西
```

- false の場合は，異体字セレクタは適当な命令によって「見える形で」出力される．どのような形で出力されるかを規定するのが vscmd キーであり，lltjp-listings の標準設定では以下の例の右側のように出力される．

```
\begin{lstlisting}[vsraw=false,
   vscmd=\ltjlistingsvsstdcmd]
葛城市，葛飾区，葛西
\end{lstlisting}
```

```
葛城市，葛飾区，葛西
```

ちなみに，本ドキュメントでは次のようにしている：

```
\def\IVSA#1#2#3#4#5{%
  \hbox to1em{\hss\textcolor{blue}{\raisebox{3.5pt}{\normalfont\ttfamily%
    \fboxsep=0.5pt\fbox{\hbox to0.75em{\hss\tiny \oalign{0#1#2\crcr#3#4#5\crcr}\hss}}}}\hss}%
}
{\catcode`\%=11
  \gdef\IVSB#1{\expandafter\IVSA\directlua{
    local cat_str = luatexbase.catcodetables['string']
    tex.sprint(cat_str, string.format('%X', 0xE00EF+#1))
}}}
\lstset{vscmd=\IVSB}
```

既定の出力命令を復活させたい場合は vscmd=\ltjlistingsvsstdcmd とすれば良い．

**■doubleletterspace キー**　listings パッケージで列揃えが [c]fixed となっている場合でも，場合によっては文字が縦に揃わない場合もある．例を以下に示そう．これは強調するために basewidth=2em を設定している．

```
:   H   :
:    H  H  H  H    :
```

1 行目と 2 行目の「H」の位置が揃っていないが，これは出力単位ごとに，先頭・末尾・各文字間に同じ量の空白を挿入することによる．

lltjp-listing では，このような症状を改善させるために doubleletterspace キーを追加した（標準では互換性のために無効になっている）．このキーを有効にすると，出力単位中の各文字間の空白を 2 倍にすることで文字を揃いやすくしている．上と同じものを doubleletterspace キーを有効にして組んだものが以下であり，きちんと「H」の位置が揃っていることが分かる．

```
:   H   :
:   H   H   H   H   :
```

## 18.2　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 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：

| | | **後側文字種** | | | | |
|---|---|---|---|---|---|---|
| | | 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．

`U+0080` 以降の**異体字セレクタ以外の**各文字が Letter, Other, Kanji, Open, Close のどれに属するかは次によって決まる：

- （`U+0080` 以降の）**ALchar** は，すべて Letter 扱いである．
- **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 3 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．

# 19　和文の行長補正方法

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

- （`lineend=extended` の場合）**JAglue** の挿入処理のところで，……
- 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 **JAglue**（xkanjiskip，kanjiskip，JFM グルー）とそれ以外のグルーの全てで（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（`lineend=true` の場合）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．その処理においては，グルーの自然長と **JAglue** 以外のグルーの伸び量・縮み量は変更せず，必要に応じて **JAglue** の伸び量・縮み量のみを変更する設計とした．

この章の残りでは各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）がある．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行では，「行末文字の位置調整」のみ行い，「グルーの調整」は行わない．**

まず，段落中の行中のグルーを

- **JAglue** ではないグルー
- JFM グルー（優先度[*51]別にまとめられる）
- 和欧文間空白 (xkanjiskip)
- 和文間空白 (kanjiskip)

の 1 + 1 + 8 + 1 = 10 つに類別する．そして許容されている伸び量（`stretch` の値）の合計を無限のレベルごとに

$$T_l^+ := \sum_{\mathtt{stretch_order}(p) = l} \mathtt{stretch}(p), \qquad l \in \{(\text{finite}), \mathtt{fi}, \mathtt{fil}, \mathtt{fill}, \mathtt{filll}\}$$

と計算する．さらに，

$$T^+ := T^+_{L^+}, \qquad L^+ = \max\{l \in \{(\text{finite}), \mathtt{fi}, \mathtt{fil}, \mathtt{fill}, \mathtt{filll}\} : T_l^+ \neq 0\}$$

とおく．有限の伸び量については，上記の 8 種類の類別ごとにも合計を計算する．さらに縮み量（`shrink` の値）についても同様の処理を行い，$T^-$ を計算する．

また，行長から自然長を引いた値を *total* とおく．

[*51] 8.5 節にあるように，各 JFM グルーには −4 から 3 までの優先度がついている．場合によっては伸びと縮みで異なる優先度が付いているかもしれない．

## 19.1　行末文字の位置調整（行分割後の場合）

行末が **JAchar** であり，この文字の属する文字クラスでは

$$\texttt{end_adjust} = \{a_1, a_2, \ldots, a_n\}$$

であったとする．このとき，以下の条件を満たした場合，この文字クラスに対する `end_adjust` の値のいずれかだけこの文字の位置を移動させる．

**最終行以外**　行長調整に無限大の伸縮度が用いられていない．すなわち，$total > 0$ ならば $L^+ = (\text{finite})$ であり，$total > 0$ ならば $L^- = (\text{finite})$ である．

**最終行**　行長調整に無限大に伸び縮みするグルーが用いられたなら，それは `\parfillskip` のみであり，かつ，次の不等式が成立する：

$$\min\{0, a_1\}\texttt{\zw} \le (\texttt{\parfillskip} \text{ の実際の長さ}) \le \max\{0, a_n\}\texttt{\zw}$$

各 $1 \le i \le n$ に対して，「行末に $a_i$ 全角だけのカーンを追加した時の，*glue_set* の値」を $b_i$ とおく．式で書くと，

$$b_i = \begin{cases} \dfrac{|total - a_i\texttt{\zw}|}{T^+} & (total - a_i\texttt{\zw} \ge 0), \\ \dfrac{|total - a_i\texttt{\zw}|}{T^-} & (total - a_i\texttt{\zw} < 0). \end{cases}$$

$b_i$ 達の最小値を与えるような $i$ を $j$ としたとき[*52]，行末に大きさ $a_j$ のカーンを追加する．*total* から $a_j$ 全角の大きさだけ引いておく．

## 19.2　行末文字の位置調整（行分割での考慮）

`lineend=extended` が指定されている場合，TEX による行分割が行われる前に各 **JAchar** の直後に，その文字が行末に来たときの位置補正用のノードを挿入していく．

16 章の用語を使って述べる．前側のクラスタ *Nq* が「和文 A」「和文 B」であり，JFM によって `end_adjust` の値が

$$\texttt{end_adjust} = \{a_1, a_2, \ldots, a_n\}$$

であったとする．このとき，次のクラスタ *Np* の直前に以下のノード列を挿入する．**JAglue** の挿入過程で禁則処理のために「*Nq* と *Np* の間のペナルティ値を増やす」ことが行われることがあるが，以下で述べられている $(n+1)$ 個のペナルティはみなその処理対象になっている．

$$\boxed{\begin{matrix}\text{kern}\\ a_1\texttt{\zw}\end{matrix}} \longrightarrow \boxed{\begin{matrix}\text{penalty}\\ 0\end{matrix}} \longrightarrow \boxed{\begin{matrix}\text{kern}\\ (a_2 - a_1)\texttt{\zw}\end{matrix}} \longrightarrow \boxed{\begin{matrix}\text{penalty}\\ 0\end{matrix}} \longrightarrow \boxed{\begin{matrix}\text{kern}\\ (a_3 - a_2)\texttt{\zw}\end{matrix}}$$
$$\longrightarrow \cdots \longrightarrow \boxed{\begin{matrix}\text{penalty}\\ 0\end{matrix}} \longrightarrow \boxed{\begin{matrix}\text{kern}\\ (a_n - a_{n-1})\texttt{\zw}\end{matrix}} \longrightarrow \boxed{\begin{matrix}\text{penalty}\\ 0\end{matrix}} \longrightarrow \boxed{\begin{matrix}\text{kern}\\ -a_n\texttt{\zw}\end{matrix}} \longrightarrow \boxed{\begin{matrix}\text{penalty}\\ 10000\end{matrix}}$$

$n$ 個あるペナルティの箇所が改行可能箇所である．いずれかで改行された場合は，その前にあるカーン（$n$ 箇所のうちどこで改行しても，合計の長さは $a_i$ の形）は行末に残るが，後ろのペナルティ・

[*52] そのような $i$ が 2 つ以上あるときは，$|total - a_i \cdot \texttt{\zw}|, |a_i|, a_i$ の順で比較して一番小さくなるものが選ばれる．

カーンは除去される．なお，$a_1 = 0$ のときは最初の幅が $a_1$\zw のカーンは不要なので挿入されず，さらにかつ $n = 1$ であった場合は後ろのペナルティも挿入されない．

なお，段落の末尾には \penalty10000 と \parfillskip 由来のグルーが自動的に入るが，これらとの兼ね合いのため**最後のクラスタについては上記のノード挿入処理は行われない**．段落最終行の行末文字の位置調整は，すでに述べた「行分割後の場合」における最終行の処理を流用しているが，そのままでは「段落末尾をぶら下げ組 ($a_1 = -0.5$) にする」ことができない[*53]ため，

- 段落末尾の \penalty10000\parfillskip の直前に，$a_1$\zw のカーンを挿入する
- 行分割後，行末文字の位置調整を行う前に，そのカーンを削除する

という前処理を追加している．

## 19.3 グルーの調整

$|total|$ の分だけが，行中のグルーの伸び量，あるいは縮み量に応じて負担されることになる．以下，$total \geq 0$ であると仮定して話を進めるが，負のときも同様である．luatexja-adjust の初期値では以下の順に伸び量を負担するようになっており，（優先度 $-4$ の JFM グルーは例外として）できるだけ kanjiskip を自然長のままにすることを試みている．この順番は stretch_priority（縮み量については shrink_priority）パラメータで変更可能である．

(A) **JAglue** 以外のグルー
(B) 優先度 3 の JFM グルー
(C) 優先度 2 の JFM グルー
(D) 優先度 1 の JFM グルー
(E) 優先度 0 の JFM グルー
(F) 優先度 $-1$ の JFM グルー
(G) 優先度 $-2$ の JFM グルー
(H) xkanjiskip
(I) 優先度 $-3$ の JFM グルー
(J) kanjiskip
(K) 優先度 $-4$ の JFM グルー

1. 行末の **JAchar** を移動したことで $total = 0$ となれば，調整の必要はなく，行が格納されている hbox の glue_set, glue_sign, glue_order を再計算すればよい．以降，$total \neq 0$ と仮定する．
2. $total$ が「**JAglue** 以外のグルーの伸び量の合計」（以下，(A) の伸び量の合計，と称す）よりも小さければ，それらのグルーに $total$ を負担させ，**JAglue** 達自身は自然長で組むことができる．よって，以下の処理を行う：
   (1) 各 **JAglue** の伸び量を 0 とする．
   (2) 行が格納されている hbox の glue_set, glue_sign, glue_order を再計算する．これによって，$total$ は **JAglue** 以外のグルーによって負担される．

[*53] 通常時は \parfillskip の内容は 0pt plus 1fil であるため，負の長さになることはない．これに伴って，「段落末尾はぶら下げ組が望ましい」状況であっても，実際には末尾の句点（とその前の数文字）がまとめて次の行に追い出されてしまう．

3. *total* が「(A) の伸び量の合計」以上ならば，(A)–(K) のどこまで負担すれば *total* 以上になるかを計算する．例えば，

$$\textit{total} = \text{((A)–(B) の伸び量の合計)} + p \cdot \text{((C) の伸び量の合計)}, \qquad 0 \leq p < 1$$

であった場合，各グルーは次のように組まれる：
   - (A), (B) に属するグルーは各グルーで許された伸び量まで伸ばす．
   - (C) に属するグルーはそれぞれ $p \times$ (伸び量) だけ伸びる．
   - (D)–(K) に属するグルーは自然長のまま．

   実際には，前に述べた「設計」に従い，次のように処理している：
   (1) (C) に属するグルーの伸び量を $p$ 倍する．
   (2) (D)–(K) に属するグルーの伸び量を 0 とする．
   (3) 行が格納されている hbox の `glue_set`, `glue_sign`, `glue_order` を再計算する．これによって，*total* は **JAglue** 以外のグルーによって負担される．
4. *total* が (A)–(K) の伸び量の合計よりも大きい場合，どうしようもないので`^^;` 何もしない．

# 20　複数フォントの「合成」（未完）

# 21　LuaTEX-ja におけるキャッシュ

luaotfload パッケージが，各 TrueType・OpenType フォントの情報をキャッシュとして保存しているのと同様の方法で，LuaTEX-ja もいくつかのキャッシュファイルを作成するようになった．

- 通常，キャッシュは `$TEXMFVAR/luatexja/` 以下に保存され，そこから読み込みが行われる．
- 「通常の」テキスト形式のキャッシュ（拡張子は `.lua.gz`，gzip 圧縮されているため）以外にも，それをバイナリ形式（バイトコード）に変換したものもサポートしている．
  - キャッシュを読み込む時，同名のバイナリキャッシュがあれば，テキスト形式のものよりそちらを優先して読み込む．
  - テキスト形式のキャッシュが更新/作成される際は，そのバイナリ版も同時に更新される．また，（バイナリ版が見つからず）テキスト形式のキャッシュ側が読み込まれたときは，LuaTEX-ja はバイナリキャッシュを作成する．未圧縮のテキスト形式のキャッシュ (`hoge.lua`) は 20200802.0 以降では利用しない．

## 21.1　キャッシュの使用箇所

LuaTEX-ja では以下のキャッシュを使用している：

`ltj-cid-auto-adobe-japan1.{lua.gz,luc}`
: Ryumin-Light のような非埋め込みフォントの情報を格納しており，（それらが LuaTEX-ja の標準和文フォントなので）LuaTEX-ja の読み込み時に自動で読まれる．生成には `UniJIS2004-UTF32-{H,V}`, `Adobe-Japan1-UCS2` という 3 つの CMap が必要である．
  40 ページで述べたように，`cid` キーを使って非埋め込みの中国語・韓国語フォントを定義する場合，同様のキャッシュが生成される．キャッシュの名称，必要となる CMap については表 21 を

表 21. cid key and corresponding files

| cid key | name of the cache | used CMaps | |
|---|---|---|---|
| Adobe-Japan1-* | ltj-cid-auto-adobe-japan1.{lua.gz,luc} | UniJIS2004-UTF32-* | Adobe-Japan1-UCS2 |
| Adobe-Korea1-* | ltj-cid-auto-adobe-korea1.{lua.gz,luc} | UniKS-UTF32-* | Adobe-Korea1-UCS2 |
| Adobe-KR-* | ltj-cid-auto-adobe-kr.{lua.gz,luc} | UniAKR-UTF32-* | Adobe-KR-UCS2 |
| Adobe-GB1-* | ltj-cid-auto-adobe-gb1.{lua.gz,luc} | UniGB-UTF32-* | Adobe-GB1-UCS2 |
| Adobe-CNS1-* | ltj-cid-auto-adobe-cns1.{lua.gz,luc} | UniCNS-UTF32-* | Adobe-CNS1-UCS2 |

参照して欲しい．

`ltj-kinsoku_default.{lua.gz,luc}`

禁則処理，kansujichar などの標準設定が格納されたファイルである．

`ltj-jisx0208.luc`

LuaTEX-ja 配布中の `ltj-jisx0208.lua` をバイトコード化したものである．これは JIS X 0208 と Unicode との変換テーブルであり，pTEX との互換目的の文字コード変換命令で用いられる．

`ltj-ivd_aj1.luc`

LuaTEX-ja 配布中の `ltj-ivd_aj1.lua` をバイトコード化したものである．これは Unicode の漢字異体字データベースの Adobe-Japan1 コレクションの内容を格納したテーブルであり，luatexja-otf パッケージの `\CID` 命令で使われることがある．

`extra_***.{lua.gz,luc}`

フォント “***” における，グリフ番号から Unicode 値への変換テーブル，縦組時のグリフ回転の有無を格納したテーブル，及び縦組時におけるグリフの原点位置・高さのテーブルを格納している．

## 21.2 内部命令

LuaTEX-ja におけるキャッシュ管理は，`luatexja.base` (`ltj-base.lua`) に実装しており，以下の関数が公開されている．ここで，⟨*filename*⟩ は保存するキャッシュのファイル名を**拡張子なしで**指定する．

`save_cache(`⟨*filename*⟩`,` ⟨*data*⟩`)`

nil でない ⟨*data*⟩ をキャッシュ ⟨*filename*⟩ に保存する．テキスト形式の ⟨*filename*⟩`.lua.gz`[*54] のみならず，そのバイナリ形式も作成・更新される．

`save_cache_luc(`⟨*filename*⟩`,` ⟨*data*⟩`[,` ⟨*serialized_data*⟩`])`

`save_cache` と同様だが，バイナリキャッシュのみが更新される．第 3 引数 ⟨*serialized_data*⟩ が与えられた場合，それを ⟨*data*⟩ の文字列化表現として使用する．そのため，⟨*serialized_data*⟩ は普通は指定しないことになるだろう．

`load_cache(`⟨*filename*⟩`,` ⟨*outdate*⟩`)`

キャッシュ ⟨*filename*⟩ を読み込む．⟨*outdate*⟩ は 1 引数（キャッシュの中身）をとる関数であり，その戻り値は「キャッシュの更新が必要」かどうかを示すブール値でないといけない．

[*54] 拡張子からわかる通り，実際には gzip 圧縮される．

load_cache は，まずバイナリキャッシュ ⟨*filename*⟩.luc を読みこむ．もしその内容が「新しい」，つまり ⟨*outdate*⟩ の評価結果が false なら load_cache はこのバイナリキャッシュの中身を返す．もしバイナリキャッシュが見つからなかったか，「古すぎる」ならば（gzip 圧縮された）テキスト形式の ⟨*filename*⟩.lua.gz を読み込み，⟨*outdate*⟩ で再度評価する．

以上より，load_cache 自体が nil でない値を返すのは，ちょうど「新しい」キャッシュが見つかった場合である．

remove_cache(⟨*filename*⟩)

キャッシュ ⟨*filename*⟩ を削除する．テキスト形式（gzip 圧縮されているか否かを問わず）もバイナリ形式もまとめて削除する．

# 22　縦組の実装

6 章の最初でも述べたように，LuaTEX-ja は横組 (TLT) で組んだボックスを回転させる方式で縦組を実装している．

LuaTEX-ja における縦組の実装は pTEX における実装 ([8, 9]) をベースにしている．

## 22.1　direction whatsit

*direction* whatsit とは，*direction* という特定の user_id を持つ whatsit のことであり，以下のタイミングで作られる．

- 組方向を \tate 等で変更したとき．
- \hbox, \vbox, \vtop による明示的なボックスの開始時．
  \hbox{}, \vbox{}といった，
  - \tate 等によりボックス内部の組方向を変更していない
  - ボックスの中身のリストが空である

  場合は，LuaTEX の hpack_filter, vpack_filter といった callback に処理が回らない．そこで，LuaTEX-ja では，\everyhbox, \everyvbox を利用することで各ボックスの先頭に確実に追加するようにしている[*55]．
- \vsplit によって vbox を分割した時の「残り」の先頭．
- LuaTEX-ja 読み込み前に作成したボックスの寸法を \ltjsetwd 等によって変更した時．
- \insert による insertion では，中身の先頭に *direction* whatsit は作られず，その代わりに中身の各ボックス・罫線の直前に作られる[*56]．

なお，\vtop{...} の場合は，先頭に direction whatsit を置くとボックスの高さが常に 0 pt になるという問題が発生する．そのため，この場合に限っては vpack 時に direction whatsit をリストの 2 番目に移動させている．

direction whatsit はあくまでも組方向処理のための補助的なノードであるので，\unhbox, \unhcopy

[*55] 問題は \hbox to 25pt{} という状況である．実際のこのボックスの中身は空でない（少なくとも direction whatsit がある）ため，何も対策をしなければ hpack 時に Underfill 警告が発生してしまうことになる．LuaTEX-ja ではそうならないように「\hbadness, \vbadness を一時的に 10000 に変更し，hpack, vpack 後に元の値に戻す」処理を行っている．

[*56] これは，ページ分割の過程で insertion が分割される時，「現在のページで出力される部分」が空となることがあることによる．先頭に whatsit を置くと，最悪でも「現在のページに whatsit が残る」ことになってしまう．

によってボックスの中身が展開される時には展開直前に削除される．これは

```
 % yoko direction
\setbox0=\hbox{\tate B}
\noindent % 水平モードに入る．この時点でのリストの中身は空
\unhbox0 A
```

といった場合に，段落が縦組で組まれたり，あるいは

```
\setbox0=\hbox{}
\leavevmode \hbox{A}\unhbox0
\setbox1=\lastbox % \box1 はどうなる？
```

で \box1 が \hbox{A} でなく空になってしまうことを防ぐためである．

## 22.2 *dir_box*

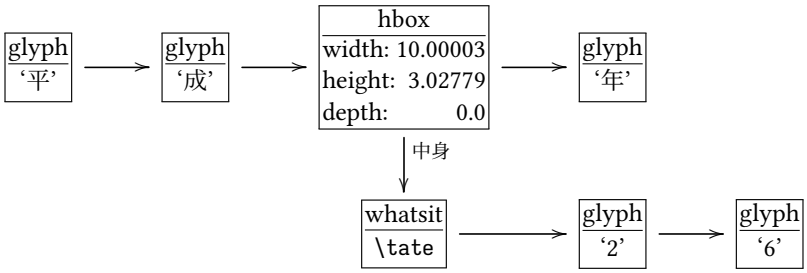
縦中横など異方向のボックスを配置する場合に，周囲の組方向と大きさを整合させるため，LuaTEX-ja では \ltj@dir が 128 以降の *hlist_node*, *vlist_node* を用いる．これらは pTEX における *dir_node* の役割と同じ果たしており，この文章中では *dir_box* と呼称する．

### 22.2.1 異方向のボックスの整合

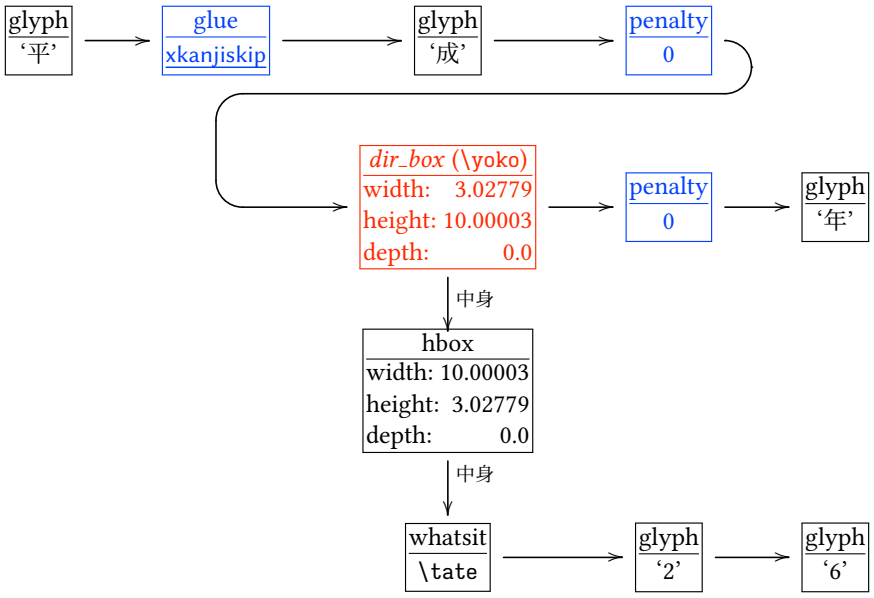
*dir_box* の第一の使用目的は，異方向のボックスの大きさを整合させることである．例えば，

```
 % yoko direction
平成\hbox{\tate 26}年
```

は段落中で



というリストを作る．その後，この段落が終了したときに，LuaTEX-ja の **JAglue** 挿入処理が行われ




のようになる（青字は **JAglue**，赤字が整合処理のための *dir_box* である）．TEX の \showbox 形式で書けば

```
.\tenmin 平
.\glue 0.0 plus 0.4 minus 0.4
.\tenmin 成
.\penalty 0
.\hbox(10.00003+0.0)x3.02779, direction TLT
..\hbox(3.02779+0.0)x10.00003, direction TLT
...\whatsit4=[]
...\tenrm 2
...\tenrm 6
.\penalty 0
.\tenmin 年
```

である．

なお，\raise, \lower, \moveleft, \moveright といったボックス移動命令では．移動を正しく表現するために段落やボックスの途中でも異方向のボックスは *dir_box* にカプセル化している．例えば

```
% yoko direction
平成\raise1pt\hbox{\tate 26}年\showlists
```

は以下のような結果を得る．

```
(前略)
\tenrm 平
\tenrm 成
\hbox(10.00003+0.0)x3.02779, shifted -1.0, direction TLT
.\hbox(3.02779+0.0)x10.00003, direction TLT
..\whatsit4=[]
..\tenrm 2
..\tenrm 6
\tenrm 年
```

また，メインの垂直リストに異方向のボックスが追加される場合にも同様に即座に *dir_box* にカプ

セル化している．ページ分割のタイミングを正しく TEX が判断するためである．\lastbox によるボックスの取得では，*dir_box* は削除される．

### 22.2.2　異方向のボックス寸法の格納

第二の使用目的は，現在の組方向がボックス本来の組方向とは異なる状況で，\ltjsetwd によってボックス寸法を設定されたことを記録することである．

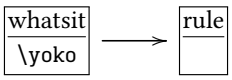
例えば

```
\setbox0=\hbox{\vrule width 10pt height 5pt depth 2pt}
\setbox1=\hbox{\tate\ltjsetwd0=20pt}
\wd0=9pt
\setbox1=\hbox{\dtou\ltjsetwd0=20pt}
\setbox0=\hbox{\dtou a\box0}
```

というコードを考える．1 行目で \box0 には横組の幅 10 pt，高さ 5 pt，深さ 2 pt のボックスが代入される．よって，

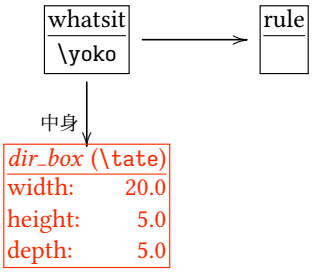
- 縦組下では \box0 は幅 7 pt，高さ・深さ 5 pt のボックスとして扱われる．
- \dtou 下では \box0 は幅 7 pt，高さ 10 pt，深さ 0 pt のボックスとして扱われる．

このとき，\box0 の**中身**は




である．

さて，2 行目で縦組時の \box0 の幅が 20 pt に設定される．この情報が direction whatsit 内部のノードリストに，*dir_box* として格納される：



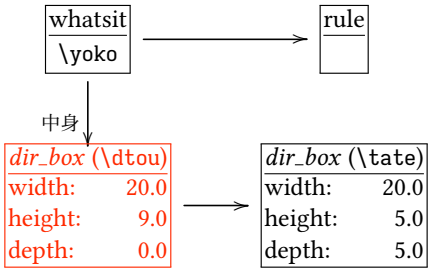

次に，3 行目では横組時の，つまり \box0 本来の組方向での深さが 9 pt に変更される．このとき，\box0 は

- 縦組下では寸法代入が既に行われているので，2 行目で作成された *dir_box* の通りに幅 20 pt，高さ・深さ 5 pt のボックスとして扱われる．
- \dtou 下ではまだ寸法代入が行われていないので，\box0 の寸法変更に追従し，幅 7 pt，高さ 9 pt，深さ 0 pt のボックスとして扱われる．

表 22. LuaTEX-ja 標準で行われる縦組形への置換

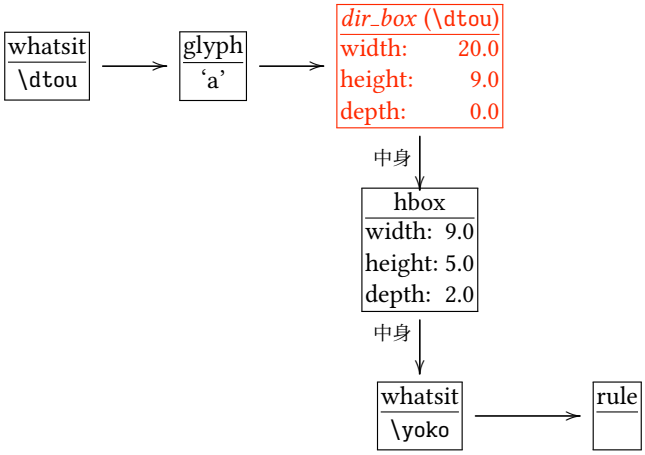
| | | |
|---|---|---|
| 、 (U+3001) ⟼ ﹑ (U+FE11) | 。 (U+3002) ⟼ ﹒ (U+FE12) | 〖 (U+3016) ⟼ ﹗ (U+FE17) |
| 〗 (U+3017) ⟼ ﹘ (U+FE18) | … (U+2026) ⟼ ⋮ (U+FE19) | ‥ (U+2025) ⟼ ： (U+FE30) |
| — (U+2014) ⟼ ｜ (U+FE31) | – (U+2013) ⟼ ﹐ (U+FE32) | ＿ (U+FF3F) ⟼ ﹍ (U+FE33) |
| （ (U+FF08) ⟼ ﹙ (U+FE35) | ） (U+FF09) ⟼ ﹚ (U+FE36) | ｛ (U+FF5B) ⟼ ﹛ (U+FE37) |
| ｝ (U+FF5D) ⟼ ﹜ (U+FE38) | 〔 (U+3014) ⟼ ﹝ (U+FE39) | 〕 (U+3015) ⟼ ﹞ (U+FE3A) |
| 【 (U+3010) ⟼ ﹟ (U+FE3B) | 】 (U+3011) ⟼ ﹠ (U+FE3C) | 《 (U+300A) ⟼ ﹡ (U+FE3D) |
| 》 (U+300B) ⟼ ﹢ (U+FE3E) | 〈 (U+3008) ⟼ ﹤ (U+FE3F) | 〉 (U+3009) ⟼ ﹥ (U+FE40) |
| 「 (U+300C) ⟼ ﹁ (U+FE41) | 」 (U+300D) ⟼ ﹂ (U+FE42) | 『 (U+300E) ⟼ ﹃ (U+FE43) |
| 』 (U+300F) ⟼ ﹄ (U+FE44) | ［ (U+FF3B) ⟼ ﹇ (U+FE47) | ］ (U+FF3D) ⟼ ﹈ (U+FE48) |

4 行目では \dtou 下での \box0 の幅が 20 pt に設定されるので，2 行目と同じように




と *dir_box* が作成される．

このように寸法代入によってつくられた *dir_box* は，前節の整合過程のときに再利用される．上記の例でいえば，5 行目を実行した後の \box0 の内容は




のようになる．

## 22.3 縦組用字形の取得

縦組時には，「、」(U+3001) から「﹑」(U+FE11) のように縦組用字形への置き換えに関係する処理は，以下のようになっている．

- 各縦組用和文フォントは読み込み時に以下の属性が設定される：

***vert_activated*** 真となるのは，明示的に `-vert` も `-vrt2` のいずれも指定されていないちょうどその時．

***auto_enable_vrt2*** 真となるのは，`vert`, `vrt2` のいずれについても有効・無効が指定されていないちょうどその時．

*vert_activated* については `luatexja.define_jfont` コールバックで渡される引数 `jfont_info` から取得可能である．

- *auto_enable_vrt2* が真の場合は，現在の script tag と language system identifier の値で `vrt2` 機能が利用可能か調べる．利用可能ならば `vrt2` を，そうでなければ `vert` を有効化する．
- また，各和文フォント読み込み時には，「OpenType 機能による置換以前に行う縦組形への置換」を格納したテーブル *vform* も作成する．
  1. 表 22 に示した各置換 $i \longmapsto v$ に対し，置換先 $v$ がフォント内に存在する文字コードであるならば，$i \longmapsto v$ を *vform* に登録する．
  2. 8.2 節にある `jpotf` が指定された場合，LuaTEX-ja 内部の別のテーブル *vert_jpotf_table* に登録されている各置換 $i \longmapsto v$ に対して置換先 $v$ がフォント内に存在する文字コードであるならば，$i \longmapsto v$ を *vform* に登録する．
  3. もし `vert` も `vrt2` も現在の script, language では有効にできない場合，どこかの script, language における `vert` で定義されている置換 $i \longmapsto v$ をすべて *vform* に登録する．

  あとで説明するように，*vform* は *vert_activated* が真であるような縦組用和文フォントでしか利用されない．
- 「現在の水平リスト」内の **JAchar** を（欧文フォントから）和文フォントへ置き換える処理において，その時点での組方向が縦組であり，かつ処理対象の各ノードの縦組用フォントで *vert_activated* が真である場合，*vform* に従いグリフが置き換えられる．

  luaotfload が行う，OpenType 機能に沿ったグリフ置換はこの後の処理となる．

# 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] Takuji Tanaka. upTeX—Unicode version of pTeX with CJK extensions, TUG 2013, October 2013. http://tug.org/tug2013/slides/TUG2013_upTeX.pdf

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

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

[7] 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.

[8] 濱野尚人，田村明史，倉沢良一.「TEX の出版への応用—縦組み機能の組み込み—」. .../texmf-dist/doc/ptex/base/ptexdoc.pdf

[9] Hisato Hamano. *Vertical Typesetting with TEX*, TUGBoat **11**(3), 346–352, 1990.

[10] International Organization for Standardization. ISO 32000-1:2008, *Document management – Portable document format – Part 1: PDF 1.7*, 2008. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51502

[11] 北川弘典.「LuaTEX-ja の近況」, TEXConf 2018. https://raw.githubusercontent.com/h-kitagawa/presentations/main/tc18ltja.pdf

[12] Takuto ASAKURA. The BXghost Package. https://github.com/wtsnjp/BXghost