LuaTEX-ja パッケージ

LuaTEX-ja プロジェクトチーム

2015 年 4 月 20 日

目次

第 I 部　ユーザーズマニュアル　4

1　はじめに　4

1.1 背景 4
1.2 pTEX からの主な変更点 4
1.3 用語と記法 5
1.4 プロジェクトについて 6

2　使い方　7

2.1 インストール 7
2.2 注意点 8
2.3 plain TEX で使う 8
2.4 LATEX で使う 9

3　フォントの変更　10

3.1 plain TEX and LATEX 2_ε 10
3.2 fontspec 11
3.3 プリセット設定 12
3.4 `\CID`, `\UTF` と `otf` パッケージのマクロ 15
3.5 標準和文フォントの変更 16

4　パラメータの変更　16

4.1 **JAchar** の範囲の設定 17
4.2 kanjiskip と xkanjiskip 18
4.3 xkanjiskip の挿入設定 19
4.4 ベースラインの移動 20
4.5 禁則処理関連パラメータと font feature 21

第 II 部　リファレンス　22

5　LuaTEX-ja における `\catcode`　22

5.1 予備知識：pTEX と upTEX における `\kcatcode` 22
5.2 LuaTEX-ja の場合 22
5.3 制御綴中に使用出来る JIS 非漢字の違い 23

6　縦組　23

6.1 サポートする組方向 24
6.2 異方向のボックス 24
6.3 組方向の取得 26
6.4 プリミティブの再定義 27

7　フォントメトリックと和文フォント　27

7.1 `\jfont` 命令 27

7.2 \tfont 命令 30
7.3 psft プリフィックス 30
7.4 JFM ファイルの構造 31
7.5 数式フォントファミリ 35
7.6 コールバック 36

8 パラメータ 38
8.1 \ltjsetparameter 38
8.2 \ltjgetparameter 40
8.3 \ltjsetparameter の代替 41

9 plain でも LaTeX でも利用可能なその他の命令 43
9.1 pTeX 互換用命令 43
9.2 \inhibitglue 43
9.3 \ltjdeclarealtfont 43

10 LaTeX 2_ε 用の命令 44
10.1 NFSS2 へのパッチ 44

11 拡張 46
11.1 luatexja-fontspec.sty 47
11.2 luatexja-otf.sty 48
11.3 luatexja-adjust.sty 50
11.4 luatexja-ruby.sty 51
11.5 lltjext.sty 51

第 III 部 実装 53

12 パラメータの保持 53
12.1 LuaTeX-ja で用いられるレジスタと whatsit ノード 53
12.2 LuaTeX-ja のスタックシステム 55
12.3 スタックシステムで使用される関数 56
12.4 パラメータの拡張 57

13 和文文字直後の改行 57
13.1 参考：pTeX の動作 57
13.2 LuaTeX-ja の動作 58

14 JFM グルーの挿入，kanjiskip と xkanjiskip 59
14.1 概要 59
14.2 「クラスタ」の定義 60
14.3 段落／ hbox の先頭や末尾 62
14.4 概観と典型例：2 つの「和文 A」の場合 62
14.5 その他の場合 64

15 ベースライン補正の方法 68
15.1 yoffset フィールド 68

15.2 **ALchar** の補正 68

16 listings パッケージへの対応 68

16.1 注意 69

16.2 文字種 70

17 和文の行長補正方法 71

17.1 行末文字の位置調整 72

17.2 グルーの調整 72

18 IVS 対応 73

19 複数フォントの「合成」（未完） 74

20 LuaTEX-ja におけるキャッシュ 74

20.1 キャッシュの使用箇所 74

20.2 内部命令 75

21 縦組の実装 76

21.1 direction whatsit 76

21.2 *dir_box* 77

参考文献 80

本ドキュメントはまだまだ未完成です．

第 I 部
ユーザーズマニュアル

1　はじめに

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

1.1　背景

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

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

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

1.2　pTEX からの主な変更点

LuaTEX-ja は，pTEX に多大な影響を受けている．初期の開発目標は，pTEX の機能を Lua コードにより実装することであった．しかし，（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={`ぁ,100}}
\ifnum\ltjgetparemeter{direction}=4 ... \fi
```

特に注意してほしいのは，pTEX で追加された zw と zh という単位を LuaTEX-ja で扱うには，\zw, \zh と制御綴の形にしないといけないという点である．

■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 13 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　本版からは，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．LuaTEX のコールバックや実装により，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．

1.3　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.

- サンセリフ体で書かれた語（例：prebreakpenalty）は日本語組版用のパラメータを表し，これらは `\ltjsetparameter` 命令のキーとして用いられる.
- 下線付きタイプライタ体の語（例：`fontspec`）は LaTeX のパッケージやクラスを表す.
- 本ドキュメントでは，自然数は 0 から始まる．自然数全体の集合は ω 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.

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

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

■開発メンバー

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

2 使い方

2.1 インストール

- LuaTEX beta-0.80.0 (or later)
- luaotfload v2.5 (or later)
- adobemapping (Adobe cmap and pdfmapping files)
- everysel (if you want to use LuaTEX-ja with LaTEX 2_ε)
- fontspec v2.4
- **IPAex フォント (http://ipafont.ipa.go.jp/)**

要約すると，本バージョンの LuaTEX-ja は TEX Live 2014 以前では動作しない[*1].

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)
- W32TEX (luatexja.tar.xz)

これらのディストリビューションは IPAex フォントも収録している．W32TEX においては IPAex フォントは luatexja.tar.xz 内にある．

■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 リポジトリをクローンする：

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

   - master ブランチのスナップショット（tar.gz 形式）をダウンロードする．
     http://git.sourceforge.jp/view?p=luatex-ja/luatexja.git;a=snapshot;h=HEAD;sf=tgz.

   master ブランチ（従って，CTAN 内のアーカイブも）はたまにしか更新されないことに注意．主な開発は master の外で行われ，比較的まとまってきたらそれを master に反映させることにしている．
2. 「Git リポジトリをクローン」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や標準の禁則処理用パラメータを格納した ltj-kinsoku.lua を生成するために，以下を実行する必要がある：

   ```
   $ cd src
   $ lualatex ltjclasses.ins
   $ lualatex ltjsclasses.ins
   $ lualatex ltjltxdoc.ins
   $ luatex   ltj-kinsoku_make.tex
   ```

   最後の ltj-kinsoku_make.tex の実行を忘れないように注意．ここで使用した *.{dtx,ins} と ltj-kinsoku_make.tex は通常の使用にあたっては必要ない．

[*1] もっとも，自分で LuaTEX のバイナリを Subversion リポジトリからビルドしていれば話は別である．

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．LuaJITTEX を用いると LuaTEX のだいたい 1.3 倍の速度で動くようであるが，IPA mj 明朝などの大きいフォントを用いた場合には LuaTEX よりも遅くなることもある．
- LuaTEX-ja が動作するためには，**導入・更新後の初回起動時に** `UniJIS2004-UTF32-{H,V}`, `Adobe-Japan1-UCS2` という 3 つの CMap が LuaTEX によって見つけられることが必要である．しかしやや古い MiKTEX ではそのようになっていないので，次のエラーが発生するだろう：

  ```
  ! LuaTeX error ...iles (x86)/MiKTeX 2.9/tex/luatex/luatexja/ltj-rmlgbm.lua
  bad argument #1 to 'open' (string expected, got nil)
  ```

  そのような場合には，プロジェクト Wiki 英語版トップページ中に書かれているバッチファイルを実行して欲しい．このバッチファイルは，作業用ディレクトリに CMap 達をコピーし，その中で LuaTEX-ja の初回起動を行い，作業用ディレクトリを消す作業をしている．

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：

組方向	字体	フォント名	“10 pt”	“7 pt”	“5 pt”
横組	明朝体	IPAex 明朝	`\tenmin`	`\sevenmin`	`\fivemin`
	ゴシック体	IPAex ゴシック	`\tengt`	`\sevengt`	`\fivegt`
縦組	明朝体	IPAex 明朝	`\tentmin`	`\seventmin`	`\fivetmin`
	ゴシック体	IPAex ゴシック	`\tentgt`	`\seventgt`	`\fivetgt`

  - `luatexja.cfg` を用いることによって，標準和文フォントを IPAex フォントから別のフォントに置き換えることができる．3.5 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 2_ε を用いる場合も基本的には同じである．日本語組版のための最低限の環境を設定するためには，`luatexja.sty` を読み込むだけでよい：

```
\usepackage{luatexja}
```

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

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

- 標準では，次のフォントファミリが用いられる：

字体	ファミリ	`\mdseries`	`\bfseries`	スケール
明朝体	`mc`	IPAex 明朝	IPAex ゴシック	0.962216
ゴシック体	`gt`	IPAex ゴシック	IPAex ゴシック	0.962216

  どちらのファミリにおいても，その bold シリーズで使われるフォントはゴシック体の medium シリーズで使われるフォントと同じであることに注意．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．
- 数式モード中の和文文字は明朝体 (`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}
```

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_ε 新ドキュメントクラス」）に対応するものとして，`ltjclasses`[*2]，`ltjsclasses`[*3]がそれぞれ用意されている．

■脚注とボトムフロートの出力順序　オリジナルの LaTeX では脚注がボトムフロートの上に来るようになっており，pLaTeX では脚注がボトムフロートの下に来るように変更されている．

LuaTeX-ja では「欧文クラスの中にちょっとだけ日本語を入れる」という利用も考慮し，脚注とボトムフロートの順序は LaTeX 通りとした．もし pLaTeX の出力順序が好みならば，`stfloats` パッケージを利用して

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

のようにすればよい．`footmisc` パッケージを `bottom` オプションを指定して読み込むという方法もあるが，それだとボトムフロートと脚注の間が開いてしまう．

[*2] 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．

[*3] `ltjsarticle.cls`, `ltjsbook.cls`, `ltjskiyou.cls`.

■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．

3　フォントの変更

3.1　plain TeX and LaTeX 2_ε

■plain TeX　plain TeX で和文フォントを変更するためには，pTeX のように \jfont 命令や \tfont 命令を直接用いる．7.1 節を参照．

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

- 和文フォントの属性を変更するには，\fontfamily, \fontseries, \fontshape, そしてそれらを反映させるために \selectfont を用いればよい．

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

  ここで，\fontencoding{⟨*encoding*⟩} は，引数により和文側か欧文側かのどちらかのエンコーディングを変更する．例えば，\fontencoding{JY3} は和文フォントのエンコーディングを JY3 に変更し，\fontencoding{T1} は欧文フォント側を T1 へと変更する．\fontfamily も引数により和文側，欧文側，**あるいは両方**のフォントファミリを変更する．詳細は 10.1 節を参照すること．
- 和文フォントファミリの定義には \DeclareFontFamily の代わりに \DeclareKanjiFamily を用いる．しかし，現在の実装では \DeclareFontFamily を用いても問題は生じない．
- 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}{bx}{n}{<-> s*KozMinPr6N-Bold:jfm=ujis;-kern}{}
    % Kozuka Mincho Pr6N 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 9.3 節の \ltjdeclarealtfont と，その LaTeX 版の \DeclareAlternateKanjiFont（10.1 節）を参照せよ．

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

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

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

$$y=(x-1)^2+2 \quad よって \quad y>0$$

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

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，

- 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：

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

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

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

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

また LuaTEX-ja プロジェクトでは，和文文字が識別子として用いられることはほとんどないと考えており，したがってこの節では数式モード中の和文フォントを変更する方法については記述しない．この方法については 7.5 節を参照のこと．

3.2 fontspec

fontspec パッケージと同様の機能を和文フォントに対しても用いる場合，luatexja-fontspec パッケージを読み込めばよい．

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

このパッケージは必要ならば自動で luatexja パッケージと fontspec パッケージを読み込む．

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

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

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_ε 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．**なお，\setmonojfont はこの match オプションが指定された時のみ定義される．**この命令は標準の「タイプライタ体に対応する和文フォント」を指定する．

pass=⟨*opts*⟩

fontspec パッケージに渡すオプション ⟨*opts*⟩ を指定する．

標準で \setmonojfont コマンドが定義されないのは，和文フォントではほぼ全ての和文文字のグリフが等幅であるのが伝統的であったことによる．また，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標準では使用されない，言い換えれば kern feature は標準では無効化となっている．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（7.1 節を参照）．

以下に\jfontspec の使用例を示す．

```
\jfontspec[CJKShape=NLC]{KozMinPr6N-Regular}
JIS~X~0213:2004→辻\par
\jfontspec[CJKShape=JIS1990]{KozMinPr6N-Regular}
JIS~X~0208:1990→辻
```

JIS X 0213:2004 →辻

JIS X 0208:1990 →辻

3.3 プリセット設定

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

■一般的なオプション

fontspec

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．**このオプションは標準で有効になっている．**もし fontspec パッケージに何らかのオプションを渡す必要がある[*4]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（明朝），ltjpg（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（丸ゴシック）という 3 つの和文フォントファミリを定義し，これらを用いる．

欧文フォントの指定で fontspec パッケージを読み込んでいる場合でも

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

のようにこのオプションを指定することは可能である．一方，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．

nodeluxe

LaTeX 2_ε 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．**このオプションは標準で有効になっている．**

deluxe

明朝体 2 ウェイト・ゴシック体 3 ウェイトと，丸ゴシック体 (\mgfamily, \textmg{...}) を使

[*4] 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．

用可能とする．ゴシック体は中字・太字・極太の 3 ウェイトがあるが，極太ゴシック体を使う場合，

- `\gtebfamily`, `\textgteb{...}`
- `\ebseries`（周囲がゴシック体のとき，`nfssonly` オプション指定時のみ）

のいずれかを用いる．標準で `\ebseries` が準備されていないのは，バージョンが古い `fontspec` では中字 (`\mdseries`) と太字 (`\bfseries`) しか扱えなかった名残である．

`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[*5]．

`bold`

「明朝の太字」をゴシック体の太字によって代替する．

`90jis`

出来る限り 90JIS の字形を使う．

`jis2004`

出来る限り JIS2004 の字形を使う．

`jis`

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

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

■多ウェイト用プリセットの一覧 `morisawa-pro`, `morisawa-pr6n` 以外はフォントの指定は（ファイル名でなく）フォント名で行われる．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	medium	KozMinPro-Regular	KozMinProVI-Regular	KozMinPr6N-Regular
	bold	KozMinPro-Bold	KozMinProVI-Bold	KozMinPr6N-Bold
ゴシック	medium	KozGoPro-Regular*	KozGoProVI-Regular*	KozGoPr6N-Regular*
		KozGoPro-Medium	KozGoProVI-Medium	KozGoPr6N-Medium
	bold	KozGoPro-Bold	KozGoProVI-Bold	KozGoPr6N-Bold
	heavy	KozGoPro-Heavy	KozGoProVI-Heavy	KozGoPr6N-Heavy
丸ゴシック		KozGoPro-Heavy	KozGoProVI-Heavy	KozGoPr6N-Heavy

[*5] `\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 font feature の追加，`nfssonly` 指定時にはシェイプを `rb` に切り替え）．

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

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

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

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

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

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

family	series	morisawa-pro	morisawa-pr6n
明朝	medium	A-OTF-RyuminPro-Light.otf	A-OTF-RyuminPr6N-Light.otf
	bold	A-OTF-FutoMinA101Pro-Bold.otf	A-OTF-FutoMinA101Pr6N-Bold.otf
ゴシック	medium	A-OTF-GothicBBBPro-Medium.otf	A-OTF-GothicBBBPr6N-Medium.otf
	bold	A-OTF-FutoGoB101Pro-Bold.otf	A-OTF-FutoGoB101Pr6N-Bold.otf
	heavy	A-OTF-MidashiGoPro-MB31.otf	A-OTF-MidashiGoPr6N-MB31.otf
丸ゴシック		A-OTF-Jun101Pro-Light.otf	A-OTF-ShinMGoPr6N-Light.otf

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

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

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

`moga-mobo` MogaMincho, MogaGothic, and MoboGothic. 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
		Moga90Gothic	MogaGothic
	bold	Moga90Gothic Bold	MogaGothic Bold
	heavy	Moga90Gothic Bold	MogaGothic Bold
丸ゴシック		Mobo90Gothic	MoboGothic

■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
明朝体	Ryumin-Light（非埋込）	IPA 明朝	IPAex 明朝	MS 明朝
ゴシック体	GothicBBB-Medium（非埋込）	IPA ゴシック	IPAex ゴシック	MS ゴシック

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

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

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

標準 フォント名（HGMinchoE など）
90jis **指定時** ファイル名 (hgrme.ttc, hgrge.ttc, hgrsgu.ttc, hgrsmp.ttf)
jis2004 **指定時** ファイル名 (hgrme04.ttc, hgrge04.ttc, hgrsgu04.ttc, hgrsmp04.ttf)

として指定を行っているので注意すること．

3.4　\CID, \UTF と otf パッケージのマクロ

pLaTeX では，JIS X 0208 にない Adobe-Japan1-6 の文字を出力するために，齋藤修三郎氏による otf パッケージが用いられていた．このパッケージは広く用いられているため，LuaTeX-ja においても otf パッケージの機能の一部を（luatexja-otf という別のパッケージとして）実装した．

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

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

\aj半角{はんかくカタカナ}
```

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

otf パッケージでは，それぞれ次のようなオプションが存在した：

deluxe
: 明朝体・ゴシック体各 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，10.1 節 (NFSS2) や 3.2 節 (fontspec) の方法で手動で指定する必要がある．

3.5 標準和文フォントの変更

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．このファイルを用いると plain TEX, LaTEX 2_ε における標準和文フォントを IPAex 明朝・IPAex ゴシックから変更することができる．しかし，基本的には**文章中で用いるフォントは（例えば luatexja-preset などで）文書ソース内で指定するべき**であり，この luatexja.cfg は，「IPAex フォントがインストールできない」など，IPAex フォントが使用できない場合にのみ応急処置的に用いるべきである．

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．

なお，20140906.0 以前のバージョンのように，Ryumin-Light, GothicBBB-Medium という名前の非埋込フォントを用いる場合は

```
\def\ltj@stdmcfont{psft:Ryumin-Light}
\def\ltj@stdgtfont{psft:GothicBBB-Medium}
```

と記述すればよい．

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 \ltjsetparameter と \ltjgetparameter を用いる．

4.1 JAchar の範囲の設定

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 は全て LuaTEX-ja のデフォルトでは 4 番の文字範囲に属しているが，上記の指定を行えば SIP は 100 番に属すようになり，4 番からは除かれる．

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．例えば，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 番の文字範囲に属する文字は **ALchar**.
- 2 番，3 番，6 番，7 番，8 番の文字範囲に属する文字は **JAchar**.

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

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．

■初期設定　LuaTEX-ja では 8 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 6.0 のブロック．
- Adobe-Japan1-6 の CID と Unicode の間の対応表 Adobe-Japan1-UCS2.
- 八登崇之氏による upTEX 用の PXbase バンドル．

以下ではこれら 8 つの文字範囲について記述する．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．なお，U+0080 以降でこれら 8 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^{J} ISO 8859-1 の上位領域（ラテン 1 補助）と JIS X 0208 の共通部分．この文字範囲は以下の文字で構成される：

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

範囲 1^{A} ラテン文字のうち，Adobe-Japan1-6 との共通部分があるもの．この範囲は以下の 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：

- 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:

表 1. 文字範囲 3 に指定されている Unicode ブロック.

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

表 2. 文字範囲 6 に指定されている Unicode ブロック.

U+2460–U+24FF	Enclosed Alphanumerics	U+2E80–U+2EFF	CJK Radicals Supplement
U+3000–U+303F	CJK Symbols and Punctuation	U+3040–U+309F	Hiragana
U+30A0–U+30FF	Katakana	U+3190–U+319F	Kanbun
U+31F0–U+31FF	Katakana Phonetic Extensions	U+3200–U+32FF	Enclosed CJK Letters and Months
U+3300–U+33FF	CJK Compatibility	U+3400–U+4DBF	CJK Unified Ideographs Extension A
U+4E00–U+9FFF	CJK Unified Ideographs	U+F900–U+FAFF	CJK Compatibility Ideographs
U+FE10–U+FE1F	Vertical Forms	U+FE30–U+FE4F	CJK Compatibility Forms
U+FE50–U+FE6F	Small Form Variants	U+20000–U+2FFFF	(Supplementary Ideographic Plane)
U+E0100–U+E01EF	Variation Selectors Supplement		

Combining Diacritical Marks　　Latin Extended Additional

- U+1E00–U+1EFF:

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）はこれらの文字を持つ．

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

範囲 3[J] 句読点と記号類．ブロックのリストは表 1 に示してある．

範囲 4[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-"A82F, "A840-"ABFF, "FB00-"FE0F,
  "FE20-"FE2F, "FE70-"FEFF, "10000-"1FFFF, "E000-"F8FF} % non-Japanese
```

範囲 5[A] 代用符号と補助私用領域.

範囲 6[J] 日本語で用いられる文字．ブロックのリストは表 2 に示す．

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

4.2 kanjiskip と xkanjiskip

JAglue は以下の 3 つのカテゴリに分類される：

表 3. 文字範囲 7 に指定されている Unicode ブロック.

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

- 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 0.92491pt minus 0.09242pt,
xkanjiskip: 2.5pt plus 1.49994pt minus 0.59998pt

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（8.1 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．

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

abc あいう abc あいう

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

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

xyz 漢字 XYZ ひらがな abc かな

なお，以下の場合には 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) がどちらも非零である．

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 パラメータはほぼ pTEX における \ybaselineshift 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．例えば，表 4 のように，数式中に明示的に現れた \hbox は，

- pTEX では，ボックス全体が \ybaselineshift だとシフトされるので，表 4 中の “い” のように，ボックス中の和文文字は \ybaselineshift だけシフトされ，一方，“for all” のように，ボックス内の欧文文字は 2 重にシフトされることになる．
- 一方，LuaTEX-ja ではそのようなことはおこらず，数式中に明示的に現れた \hbox はシフトしない．そのため，表 4 中の “い” も “for all” も，それぞれ本文中に書かれたときと同じ上下位置に組

表 4. 数式関係のベースライン補正 (yalbaselineshift = 10 pt)

入力	数式abc: `$あa\hbox{い}$, $\int_0^x t\,dt=x^2\!/2$, $\Phi\vdash F(x)\ \hbox{for all}\ x\in A$`
pTEX	数式 あ abc: aい, $\int_0^x t\,dt = x^2/2$, $\Phi \vdash F(x)$ for all $x \in A$
LuaTEX-ja	数式 あ い abc: a , $\int_0^x t\,dt = x^2/2$, $\Phi \vdash F(x)$ for all $x \in A$

まれる．

4.5 禁則処理関連パラメータと font feature

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 feature 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 feature の適用前の文字コードによって判定される．**例えば，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．

```
1 \ltjsetparameter{postbreakpenalty={`ア, 10}}
2 \ltjsetparameter{postbreakpenalty={`ｱ, 20}}
3
4 \newcommand\showpostpena[1]{%
5   \leavevmode\setbox0=\hbox{#1\hbox{}}%
6   \unhbox0\setbox0=\lastbox\the\lastpenalty}
7
8 \showpostpena{ア},
9 \showpostpena{ｱ},
10 {\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 5 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[*6]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．

ネイティブに Unicode 全部の文字を扱える XeTEX や LuaTEX では，文字が制御綴内で使用できるかは通常の欧文文字と同じく \catcode で指定することとなる．plain XeTEX における \catcode の初期設定は unicode-letters.tex 中に記述されており，plain LuaTEX ではそれを元にした luatex-unicode-letters.tex を用いている．LaTEX では \catcode の設定はカーネルに unicode-letters.def として統合され．このファイルを XeLaTEX, LuaLaTEX の両方が用いている．

だが，XeTEX における \catcode の初期設定と LuaTEX におけるそれは一致していない：

- luatex-unicode-letters.tex の元になった unicode-letters.tex が古い
- unicode-letters.tex 後半部や unicode-letters.def 後半部では\XeTeXcharclass の設定を行なっており，それによって漢字や仮名の \catcode が 11 に設定されている．
  しかし，luatex-unicode-letters.tex ではこの「後半部」がまるごと省略されており，また LuaLaTEX でも unicode-letters.def 後半部は実行されない．従って漢字や仮名の \catcode は 12 のままになっている．

言い換えると，LuaTEX の初期状態では漢字や仮名を制御綴内に使用することはできない．

これでは pTEX で使用できた \西暦 などが使えないこととなり，LuaTEX-ja への移行で手間が生じる．そのため，LuaTEX-ja では unicode-letters.tex の後半部にあたる内容を自前でパッチし，結果として **XeTEX における初期設定と同じになるようにしている．**

[*6] 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．

表 5. \kcatcode in upTEX

\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 6. 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
ｾ (U+FF3E)	1	16	N	N	Y
￣ (U+FFE3)	1	17	N	N	Y
＿ (U+FF3F)	1	18	N	N	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
／ (U+FF0F)	1	31	N	N	Y
＼ (U+FF3C)	1	32	N	N	Y

	区	点	pTEX	upTEX	LuaTEX-ja
｜ (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

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 6 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．

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[*7]．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 7. 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*	銀は、Ag	銀は、Ag	銀は、Ag	銀は、Ag

* 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 7 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．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．

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：

```
ここは横組%      yoko
\hbox{\tate %    tate
  \hbox{縦組}%   tate
  の中に
  \hbox{\yoko 横組の内容}% yoko
  を挿入する
}
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

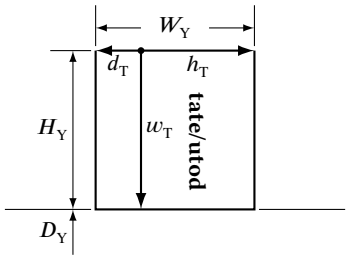
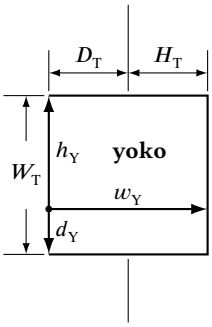
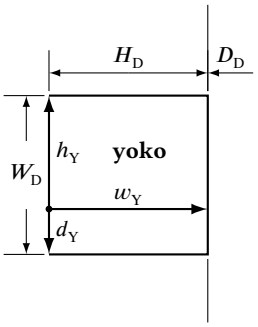
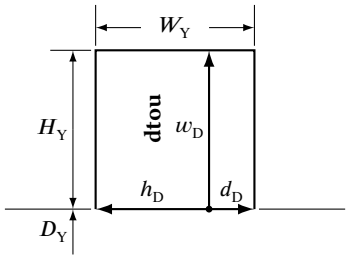
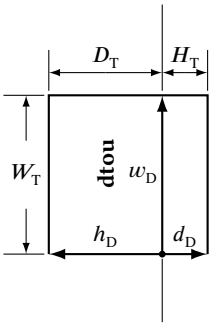
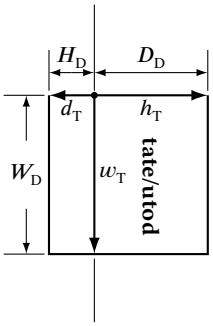
異なる組方向のボックスを配置した場合にどう組まれるかの仕様も，pTEX を踏襲している．表 8 に示す．

■\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

*7 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 8. 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
		
$W_Y = h_T + d_T$, $H_Y = w_T$, $D_Y = 0\,\mathrm{pt}$	$W_T = h_Y + d_Y$, $H_T = w_Y/2$, $D_T = w_Y/2$	$W_D = h_Y + d_Y$, $H_D = w_Y$, $D_D = 0\,\mathrm{pt}$
		
$W_Y = h_D + d_D$, $H_Y = w_D$, $D_Y = 0\,pt$	$W_T = h_D + d_D$, $H_T = d_D$, $D_T = h_D$	$W_D = w_T$, $H_D = d_T$, $D_D = h_T$

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, 20.0pt 100.0pt, 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．

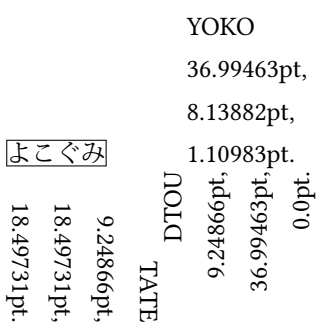
\ltjgetwd⟨*num*⟩, \ltjgetht⟨*num*⟩, \ltjgetdp⟨*num*⟩

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」「縦組及び \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

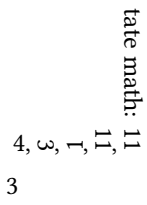
「現在の組方向」や「⟨*num*⟩ 番のボックスの組方向」は，pTEX では \ifydir や \ifybox⟨*num*⟩ といった条件判断文を使って判断することができた．しかし，LuaTEX-ja はあくまでも TEX マクロと Lua コードで記述されており，それでは新たな条件判断命令を作るのは難しい．

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}
```



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 プリミティブの再定義

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 LuaTEX-ja による前処理もしくは後処理が行われるように `\protected\def` により再定義してある．

`\unhbox`⟨*num*⟩, `\unvbox`⟨*num*⟩, `\unhcopy`⟨*num*⟩, `\unvcopy`⟨*num*⟩

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 と異なり，エラーを無視して無理矢理 `\unhbox`, `\unvbox` 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*⟩ etc., `\vcenter`

一方，こちらでは必要に応じて *dir_box* を作成する前処理を追加している．

7 フォントメトリックと和文フォント

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

當／體／醫／區

なお，`\jfont` で定義された制御綴（上の例だと `\tradgt`）は *font_def* トークンではなくマクロである．従って，`\fontname\tradgt` 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．

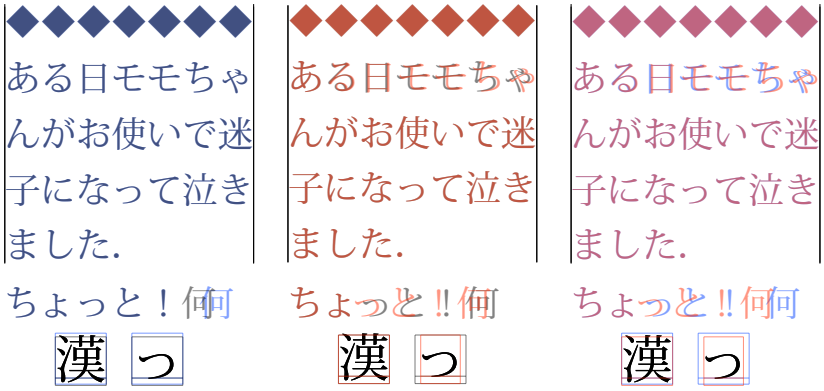
■JFM JFM は文字と和文組版で自動的に挿入されるグルー／カーンの寸法情報を持っている．JFM の構造は次の節で述べる．`\jfont` 命令の呼び出しの際には，どの JFM を用いるのかを以下のキーで指定する必要がある：

`jfm=`⟨*name*⟩

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

`jfm-ujis.lua` LuaTEX-ja の標準 JFM である．この JFM は upTEX で用いられる UTF/OTF パッケージ用の和文用 TFM である `upnmlminr-h.tfm` を元にしている．`luatexja-otf` パッケー

表 9. 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`)

```
\ltjsetparameter{differentjfm=both}
\jfont\F=file:KozMinPr6N-Regular.otf:jfm=ujis
\jfont\G=file:KozGoPr6N-Medium.otf:jfm=ujis
\jfont\H=file:KozGoPr6N-Medium.otf:jfm=ujis;jfmvar=hoge
\F ）{\G 【】}（ % halfwidth space
   ）{\H 『』}（ % fullwidth space

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 1. Example of `jfmvar` key

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．

これら 3 つの JFM の違いは表 9 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．

`jfmvar=`⟨*string*⟩

標準では，JFM とサイズが同じで，実フォントだけが異なる 2 つの和文フォントは「区別されない」．例えば図 1 において，最初の「）」と「【」の実フォントは異なるが，JFM もサイズも同じなので，普通に「）【」と入力した時と同じように半角空きとなる．

しかし，JFM とサイズが同じであっても，`jfmvar` キーの異なる 2 つの和文フォント，例えば図 1 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．

■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，本バージョン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

|ダイナミックダイクマ|
|ダイナミックダイクマ|
|ダイナミックダイクマ|
|ダイナミックダイクマ|

|ダイナミックダイクマ|
|ダイナミックダイクマ|
|ダイナミックダイクマ|
|ダイナミックダイクマ|

```
\newcommand\test{\vrule ダイナミックダイクマ\vrule\\}
\jfont\KMFW = KozMinPr6N-Regular:jfm=prop;-kern at 17pt
\jfont\KMFK = KozMinPr6N-Regular:jfm=prop at 17pt % kern is activated
\jfont\KMPW = KozMinPr6N-Regular:jfm=prop;script=dflt;+pwid;-kern at 17pt
\jfont\KMPK = KozMinPr6N-Regular:jfm=prop;script=dflt;+pwid;+kern at 17pt
\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 2. Kerning information and kanjiskip

空白と JFM 由来のグルー・カーンは同時に入りうる．図 2 を参照．

- **`\jfont` や，NFSS2 用の命令（3.1 節，10.1 節）における指定ではカーニング情報は標準で使用する**ことになっているようである．言い換えれば，カーニング情報を使用しない設定にするには，面倒でも

  ```
  \jfont\hoge=KozMinPr6N-Regular:jfm=ujis;-kern at 3.5mm
  \DeclareFontShape{JY3}{fuga}{m}{n} {<-> s*KozMinPr6N-Regular:jfm=ujis;-kern}{}
  ```

  のように，`-kern` という指定を自分で追加しなければいけない．
- 一方，**`luatexja-fontspec` の提供する `\setmainjfont` などの命令の標準設定ではカーニング情報は使用しない**（`Kerning=Off`）ことになっている．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．

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

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

の 2 つがある．`extend` や `slant` を指定した場合は，それに応じた JFM を指定すべきである[*8]．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：

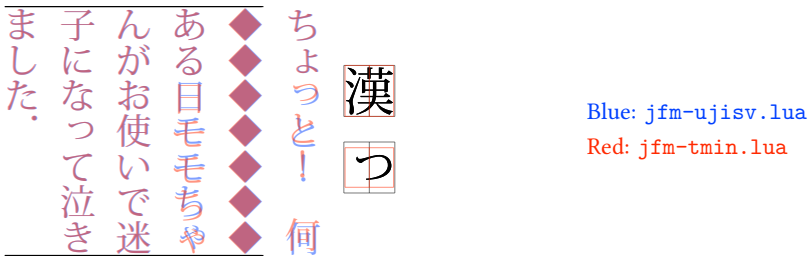
```
\jfont\E=KozMinPr6N-Regular:extend=1.5;jfm=ujis;-kern
\E あいうえお

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

あいうえお
*あいう*ABC

[*8] 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．

表 10. LuaTEX-ja に同梱されている縦組用 JFM の違い



7.2 \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 の文法は \jfont と同じである．\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：

- 自動的に vert, vrt2 の両 OpenType feature が有効化される．但し，以下の例の 3 行目に示すように，明示的に vert, vrt2（のいずれか）の有効・無効を指定した場合は別である．

```
\tfont\S=file:KozMinPr6N-Regular.otf:jfm=ujisv
    % vert and vrt2 are automatically activated
\tfont\T=file:KozMinPr6N-Regular.otf:jfm=ujisv;-vert % vert and vrt2 are not activated
```

- 7.5 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*⟩ の部分には縦組用 JFM を指定する．以下の縦組用 JFM が LuaTEX-ja には同梱されており，違いを表 10 に示した．

  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．

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．

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

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

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

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

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

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

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

のようにそれら以外の値を指定すると，エラーが発生する：

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

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

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

7.4 JFM ファイルの構造

JFM ファイルはただ一つの関数呼び出しを含む Lua スクリプトである：

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

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

`dir=`⟨*direction*⟩ （必須）

JFM の書字方向．`'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）．

⟨*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．

例外として，**`width` フィールドには数値以外に `'prop'` が指定可能である**．この場合，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 3（横組用和文フォント），図 4（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，文字クラスが開き括弧類であるときに実際必要である．

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

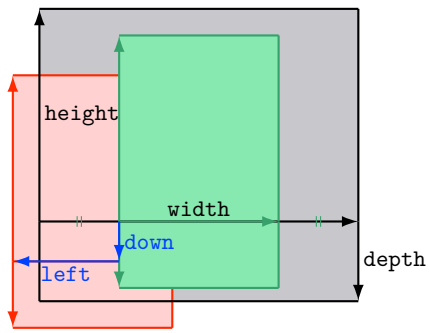
`glue={[`j`]={`⟨*width*⟩`,` ⟨*stretch*⟩`,` ⟨*shrink*⟩`, [`⟨*priority*⟩`], [`⟨*ratio*⟩`]}, ...}`

文字クラス i の文字と j の文字の間に挿入される kern や glue の量を指定する．

⟨*priority*⟩ は `luatexja-adjust` による優先順位付き行長調整（11.3 節）が有効なときのみ意味を持つ．このフィールドは省略可能であり，行調整処理におけるこの glue の優先度を −2 から +2 の間の整数で指定する．大きい値ほど「伸びやすく，縮みやすい」ことを意味する．省略時の値は 0 であり，範囲外の値が指定されたときの動作は未定義である．

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

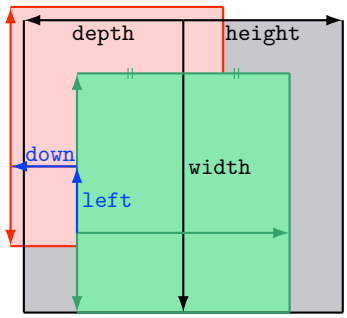
- −1 はこのグルーが「前の文字」由来であることを示す．
- +1 はこのグルーが「後の文字」由来であることを示す．
- それ以外の値は，「前の文字」由来のグルーと「後の文字」由来のグルーが混合されているこ



`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 3. 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 4. 縦組和文フォントにおける「実際の」グリフの位置

とを示す．

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

例えば，[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*⟩ には次の値を指定する．

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

`end_stretch=`⟨*kern*⟩，`end_shrink=`⟨*kern*⟩ （任意）

優先順位付き行長調整が有効であり，かつ現在の文字クラスの文字が行末に来た時に，行長を詰める調整・伸ばす調整のためにこの文字と行末の間に挿入可能なカーンの大きさを指定する．

■文字クラスの決定　文字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=file:KozMinPr6N-Regular.otf: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 font feature 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.

```
\jfont\a=file:KozMinPr6N-Regular.otf: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.

■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.

'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」（欧文文字，glue，kern 等）との境界.

−1 行中数式と地の文との境界.

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

- 実際に出力される和文フォントのサイズが design size となる．このため，例えば 1 zw が design size の 0.962216 倍である JIS フォントメトリック等を移植する場合は，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．
- 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．

```
\jfont\g=KozMinPr6N-Regular:jfm=test \g
\parindent1\zw\noindent{}◆◆◆◆◆
\par 「◆◆←二分下がり
\par 【◆◆←全角下がり
\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[*9]．

7.5 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[*10]で管理し，それぞれのファミリは 3 つのフォントを持っている：`\textfont`, `\scriptfont` そして `\scriptscriptfont` である．

LuaTEX-ja の数式中での和文フォントの扱いも同様である．表 11 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．

[*9] `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．

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

表 11. 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*⟩

7.6 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，

$$\begin{cases} 0 & \texttt{'left'} \text{ や省略時} \\ 0.5 & \texttt{'middle'} \\ 1 & \texttt{'right'} \end{cases}$$

  のいずれかの値をとる．
- $j \in \omega$ に対して，`[j]` は文字クラス i の文字と j の文字の間に挿入される kern や glue を格納している．間に入るものが kern であれば，このフィールドの値は `[j]={false,` ⟨*kern_node*⟩, ⟨*ratio*⟩`}` である．⟨*kern_node*⟩ は kern を表すノードそのものである[*11]．glue であれば，`[j]={false,` ⟨*spec_node*⟩, ⟨*ratio*⟩, ⟨*icflag*⟩`}` である．⟨*spec_node*⟩ は glue の長さを表すノードそのものであり，⟨*icflag*⟩ = *from_jfm* + ⟨*priority*⟩ である．

`ascent`, `descent`

……

`chars_cbcache`

……

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

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

`luatexja.set_width` コールバック

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

[*11] 本バージョンではノードのアクセス手法に direct access model を用いている．そのため，例えば LuaTEX beta-0.78.2 では，単なる自然数のようにしか見えないことに注意．

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

引数 `shift_info` と戻り値の `new_shift_info` は `down` と `left` のフィールドを持ち，これらの値は文字の下／左へのシフト量（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，以下の量だけ上にぜらされることとなる：

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

8 パラメータ

8.1 `\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．12 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*⟩ のリストを引数としてとる．両者の違いはスコープであり，`\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 ={⟨*chr_code*⟩,⟨*natural number*⟩}*

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

prebreakpenalty ={⟨*chr_code*⟩,⟨*penalty*⟩}* [`\prebreakpenalty`]

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

例えば閉じ括弧「〗」は絶対に行頭にきてはならないので，

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

と，最大値の 10000 が標準で指定されている．他にも，小書きのカナなど，絶対禁止というわけ

ではないができれば行頭にはきて欲しくない場合に，0 と 10000 の間の値を指定するのも有用であろう．

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[*12]
- 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 ={⟨*chr_code*⟩,⟨*penalty*⟩}* [\postbreakpenalty]

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

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 ={⟨*chr_code*⟩,⟨*mode*⟩}*

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

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

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

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

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

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

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

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

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

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

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

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

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

autospacing =⟨*bool*⟩ [\autospacing]

autoxspacing =⟨*bool*⟩ [\autoxspacing]

kanjiskip =⟨*skip*⟩* [\kanjiskip]

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

xkanjiskip =⟨*skip*⟩* [\xkanjiskip]

[*12] 後から指定した方で上書きされる．

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 14.4 節の「『右空白』の算出」を参照してほしい．

jacharrange =⟨*ranges*⟩

kansujichar ={⟨*digit*⟩，⟨*chr_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．

8.2 `\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 0.92491pt minus 0.09242pt, 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 ⟨*reg_num*⟩ に対して，`\ltjgetparameter{boxdim}{`⟨*reg_num*⟩`}` は，`\box`⟨*reg_num*⟩

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}{`⟨*chr_code*⟩`}` によって ⟨*chr_code*⟩ の属する文字範囲の番号を知ることができる．

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

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

  ```
  \ltjgetparameter{jacharrange}{\ltjgetparameter{chartorange}{⟨chr_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
  ```

8.3 `\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 2_ε 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．

■ベースライン補正量の設定　pLaTeX の ascmac パッケージでは，縦組の欧文ベースライン補正量の一時待避・復帰処理に \@saveybaselineshift という寸法レジスタを用い

```
\@savetbaselineshift\tbaselineshift\tbaselineshift\z@
...
\tbaselineshift\@savetbaselineshift
```

という処理を行っている．

これを LuaTeX-ja 用に \ltjsetparameter を使って書き直すと，

```
\@savetbaselineshift\ltjgetparameter{talbaselineshift}
\ltjsetparameter{talbaselineshift=\z@}
...
\ltjsetparameter{talbaselineshift=\@savetbaselineshift}
```

となる．

さて，縦組の欧文ベースライン補正量 talbaselineshift は，実際には \ltj@tablshift という属性レジスタに格納されている（12 節参照）．属性レジスタは長さではなく整数値を格納する[*13]ものであり，\ltj@tablshift は補正量を sp 単位で保持することから，上記のコードと同じ内容をより速い以下のコードで実現することができる．

```
\@savetbaselineshift\ltj@tablshift sp%
\ltj@tablshift\z@
...
\ltj@tablshift\@savetbaselineshift
```

この手法は ascmac パッケージの LuaTeX-ja 対応パッチ lltjp-tascmac.sty で実際に用いられている．lltjp-tascmac.sty は自動的に読み込まれるので，ユーザは何も気にせず普通に ascmac パッケージを \usepackage で読みこめば良い．

[*13] 従って，\@savetbaselineshift=\ltj@tablshift のように記述することはできない．属性レジスタを \tbaselineshift という名称にしなかったのはそのためである．

9 plain でも LaTeX でも利用可能なその他の命令

9.1 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`, `\jis`, `\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, い
一七〇一

9.2 `\inhibitglue`

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

```
\jfont\g=file:KozMinPr6N-Regular.otf: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\hrule\noindent あoff\inhibitglue ice
```

あ　ウあウ
あ　1
あ　2
あ　3
あ　office

この例を援用して，`\inhibitglue` の仕様について述べる．

- **`\inhibitglue` の垂直モード中での呼び出しは意味を持たない**[*14]．4 行目の入力で有効にならないのは，`\inhibitglue` の時点では垂直モードであり，`\noindent` の時点で水平モードになるからである．
- `\inhibitglue` の（制限された）水平モード中での呼び出しはその場でのみ有効であり，段落の境界を乗り越えない．さらに，`\inhibitglue` は上の例の最終行のように（欧文における）リガチャとカーニングを打ち消す．これは，`\inhibitglue` が内部的には「現在のリスト中に whatsit ノードを追加する」ことを行なっているからである．
- `\inhibitglue` を数式モード中で呼び出した場合はただ無視される．
- LaTeX で LuaTeX-ja を使用する場合は，`\inhibitglue` の代わりとして `\<` を使うことができる．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．

9.3 `\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．

[*14] 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．

\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，

```
\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 LaTeX 2_ε 用の命令

10.1 NFSS2 へのパッチ

LuaTEX-ja の NFSS2 への日本語パッチは pLaTeX 2_ε 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_ε 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「あ」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

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．\fontfamily のように，有効にするためには \selectfont が必要である．

\adjustbaseline

pLaTeX 2_ε では，\adjustbaseline は縦組時に「M」と「あ」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た：

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

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\adjustbaseline は同様に talbaselineshift パラメータの調整処理を行っている．

同時に，これも pLaTeX 2_ε の\adjustbaseline で行われていたが，「漢」の寸法を元に，（本節の最初に述べた，小文字で始まる）\cht, \cwd といった長さ変数を設定する．

\fontfamily{⟨*family*⟩}

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

- 現在の和文フォントに対するエンコーディングが ⟨*ja-enc*⟩ であるとしよう．現在の和文フォントファミリは，以下の 2 つの条件のうちの 1 つが満たされているときに ⟨*family*⟩ に変更される：
  - エンコーディング ⟨*ja-enc*⟩ におけるファミリ ⟨*family*⟩ が既に \DeclareKanjiFamily によって定義されている．
  - フォント定義ファイル ⟨*ja-enc*⟩⟨*family*⟩.fd（ファイル名は全て小文字）が存在する．

```
\DeclareKanjiFamily{JY3}{edm}{}
\DeclareFontShape{JY3}{edm}{m}{n}     {<-> s*KozMinPr6N-Regular:jfm=ujis;}{}
\DeclareFontShape{JY3}{edm}{m}{green}{<-> s*KozMinPr6N-Regular:jfm=ujis;color=007F00}{}
\DeclareFontShape{JY3}{edm}{m}{blue} {<-> s*KozMinPr6N-Regular:jfm=ujis;color=0000FF}{}
\DeclareAlternateKanjiFont{JY3}{edm}{m}{n}{JY3}{edm}{m}{green}{"4E00-"67FF,{-2}-{-2}}
\DeclareAlternateKanjiFont{JY3}{edm}{m}{n}{JY3}{edm}{m}{blue}{ "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 5. \DeclareAlternateKanjiFont の使用例

- 現在の欧文フォントに対するエンコーディングを ⟨*al-enc*⟩ とする．欧文フォントファミリに対しても，上記の基準が用いられる．
- 上記のいずれもが適用されない，つまり ⟨*family*⟩ が ⟨*ja-enc*⟩ と ⟨*al-enc*⟩ のどちらでも定義されないような場合がある．この場合，代替フォントに用いられるデフォルトのフォントファミリが欧文フォントと和文フォントに用いられる．LaTeX のオリジナルの実装とは異なり，現在のエンコーディングは ⟨*family*⟩ には設定されないことに注意する．

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

9.3 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 5 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）であったが，\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．

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

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

あいう abc

11 拡張

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．

```
\jfontspec[
  YokoFeatures={Color=007F00}, TateFeatures={Color=00007F},
  TateFont=KozGoPr6N-Regular
]{KozMinPr6N-Regular}
\hbox{\yoko 横組のテスト}\hbox{\tate 縦組のテスト}
\addjfontfeatures{Color=FF0000}
\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 6. TateFeatures 等の使用例

11.1 luatexja-fontspec.sty

3.2 節で述べたように，この追加パッケージは fontspec パッケージで定義されているコマンドに対応する和文フォント用のコマンドを提供する．

fontspec パッケージで指定可能な各種 font feature に加えて，和文版のコマンドには以下の “font feature” を指定することができる：

CID=⟨*name*⟩, JFM=⟨*name*⟩, JFM-var=⟨*name*⟩

これら 3 つのキーはそれぞれ \jfont, \tfont に対する cid, jfm, jfmvar キーとそれぞれ対応する．cid, jfm, jfmvar キーの詳細は 7.1 節と 7.3 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．7.3 節を参照．

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 feature 達を指定する．使用例は図 6 参照．

YokoFeatures={⟨*features*⟩}

同様に，横組においてのみ適用される feature 達を指定する．使用例は図 6 参照．

AltFont

9.3 節の \ltjdeclarealtfont や，10.1 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 font feature 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 7 に示した．

```
\jfontspec[
  AltFont={
    {Range="4E00-"67FF, Color=007F00},
    {Range="6800-"9EFF, Color=0000FF},
    {Range="3040-"306F, Font=KozGoPr6N-Regular},
  }
]{KozMinPr6N-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 7. AltFont の使用例

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=007F00} }
}
```

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=007F00 } },
}
```

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

11.2　luatexja-otf.sty

この追加パッケージは Adobe-Japan1（フォント自身が持っていれば，別の CID 文字セットでも可）の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．

\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[*15] 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．
- OpenType feature（例えばグリフ置換やカーニング）をサポートするための luaotfload パッケージのコードはこれらの文字には働かない．

■JFM への記法の追加　luatexja-otf 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．

■IVS サポート　最近の OpenType フォントや TrueType フォントには，U+E0100–U+E01EF の範囲の「文字」（漢字用異体字セレクタ）を後置することによって字形を指定する仕組み (IVS) が含まれている．執筆時点の 2013 年 12 月では，luaotfload や fontspec パッケージ類は IVS に対応してはいないようである．これらのパッケージで対応してくれるのが理想的だが，それまでのつなぎとして，luatexja-otf パッケージ内に IVS 対応を仕込んでおいた．

IVS 対応は試験的なものである．有効にするには，luatexja-otf パッケージを読み込んだ上で以下の命令を実行する[*16]：

```
\directlua{luatexja.otf.enable_ivs()}
```

すると，上の命令を実行した箇所以降では，以下のように IVS による字形指定が有効となる．

```
1 \Large
2 \jfontspec{KozMinPr6N-Regular}
3 奈良県葛E0100城市と，東京都葛E0101飾区．\\
4 こんにちは，渡
5 邉E0100邉E0101邉E0102邉E0103邉E0104
6 邉E0105邉E0106邉E0107邉E0108邉E0109
7 邉E010A邉E010B邉E010C邉E010D邉E010E
8 さん．
```

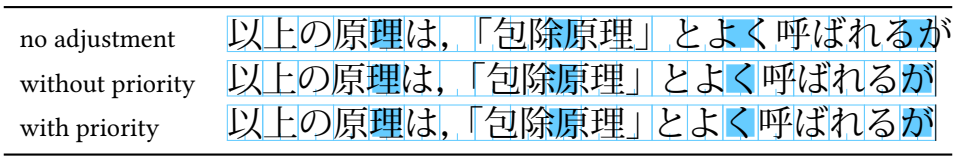
奈良県葛城市と，東京都葛飾区．
こんにちは，渡邉邉邉邉邉邉邉邉
邉邉邉邉邉邉邉さん．

左上側の入力においては，漢字用異体字セレクタを明示するため，例えば Variation Selector 18 (U+E0101) を葛E0101のように表記している．

また，IVS による字形指定は，font feature によるそれに優先されることとした．下の例において，jp78, jp90 指定で字形が変化した文字は異体字セレクタが続いていない「葛西」中の「葛」のみである．

[*15] 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．

[*16] この命令を 2 回以上実行しても意味がない．



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

図 8. 行長調整

```
\def\TEST#1{%
  {\jfontspec[#1]{KozMinPr6N-Regular}%
  葛城市，葛飾区，葛西}\\}
指定なし：\TEST{}
\texttt{jp78}：\TEST{CJKShape=JIS1978}
\texttt{jp90}：\TEST{CJKShape=JIS1990}
```

指定なし：葛城市，葛飾区，葛西
jp78：葛城市，葛飾区，葛西
jp90：葛城市，葛飾区，葛西

現状では，TEX 側のインターフェースとなる luatexja-otf.sty は一切変更していないので，ZR さんによる PXipamjm パッケージ[*17]にあるような気の利いた命令はまだない．異体字の一覧表示を行いたい場合は，git リポジトリ内の test/test19-ivs.tex 中にある Lua・TEX コードが参考になるだろう．

11.3 luatexja-adjust.sty

pTEX では，行長調整において優先度の概念が存在しなかったため，図 8 上段における半角分の半端は，図 8 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 8 下段）ことになっている．この追加パッケージは [5] や [7] における規定のような，優先順位付きの行長調整を提供する．詳細な仕様については 17 章を参照してほしい．

- 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．
  また，\hbox to ... {...} のような「幅が指定された hbox」では無効である．
- 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．

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

\ltjdisableadjust
: 優先順位付きの行長調整を無効化する．

\ltjenableadjust
: 優先順位付きの行長調整を有効化する．

adjust =⟨*bool*⟩
: \ltjsetparameter で指定可能な追加パラメータであり，⟨*bool*⟩ が *true* なら \ltjenableadjust を，そうでなければ \ltjdisableadjust を実行する．

[*17] https://github.com/zr-tex8r/PXipamjm．説明は彼のブログ記事「pxipamjm パッケージの説明書のような何か (http://d.hatena.ne.jp/zrbabbler/20131221)」にある．

11.4 luatexja-ruby.sty

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……
```

東西線の妙典(みようでん)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}
}
```

京急蒲田(けいきゆうかまた)
田(た) 京急(けいきゆう)
蒲田(かまた) 京(けい)
急蒲田(きゆうかまた)

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
る ……を承(うけたまわ)る
……を
承(うけたまわ) る

11.5 lltjext.sty

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追加パッケージ lltjext である．

従来の plext パッケージとの違いは，

- 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のに対し，<u> にはそのような制限がないことである．
- plext パッケージでは，表組（tabular 環境，align 環境等）や minipage 環境，\parbox 命令において，垂直位置指定 [t], [b] の挙動が非読み込み時と微妙に変わることがあった．
  lltjext パッケージでは，垂直位置指定が LaTeX 2_ε と同様の挙動（以下に示す）になるように修正した．
  - [t] オプション指定時は，ボックスのベースラインが中身の 1 行目のベースライン（1 行目の上に罫線などが来た時は，ボックスの上端）に一致するように配置する．
  - [b] オプション指定時は，ボックスのベースラインが中身の最終行のベースライン（中身の最後が罫線などの時は，ボックスの下端）に一致するように配置する．
  - それ以外のときは，ボックスの中央が「数式の軸」に一致するように配置する．
- 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

参照番号

第 III 部
実装

12　パラメータの保持

12.1　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 は写植で用いられた 1 Q = 0.25 mm（「級」とも書かれる）に等しい．したがって，この寸法レジスタの値を変更してはならない．

\jH (dimension) 同じく写植で用いられていた単位として「歯」があり，これも 0.25 mm と等しい．この \jH は \jQ と同じ寸法レジスタを指す．

\ltj@zw (dimension) 現在の和文フォントの「全角幅」を保持する一時レジスタ．\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@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) 現在の横組用和文フォントのフォント番号．

\ltj@curtfnt (attribute) 現在の縦組用和文フォントのフォント番号．

\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．

\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)

\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．

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

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

***italic* (1)** イタリック補正 (\/) によるカーン，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*** − 2)–(***from_jfm*** + 2) **(4–8)** JFM 由来のグルー／カーン．

***kanji_skip* (9), *kanji_skip_jfm* (10)** 和文間空白 kanjiskip を表すグルー．

***xkanji_skip* (11), *xkanji_skip_jfm* (12)** 和欧文間空白 xkanjiskip を表すグルー．

***processed* (13)** LuaTEX-ja の内部処理によって既に処理されたノード．

***ic_processed* (14)** 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* (15)** hbox か段落の最初か最後に挿入されたグルー／カーン．

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* (128) 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（21 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．

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` のコールバック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` や IVS 対応処理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．

12.2 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：

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

ふがふが. ほ げ ほ げ. ぴよぴよ

8.1 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：

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

`unsave()` が `filtered_hpack()`（これは `hpack_filter` コールバックが実行されるところである）の**前に**実行されていることに注意する．したがって，上記ソース中で 5 pt は `unsave()` のところで捨てられ，`hpack_filter` 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[*18]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：

[*18] `[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 9. パラメータ設定命令の定義

- 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．

12.3 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．

12.4 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 9 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，

$$\texttt{luatexja.isglobal} = \begin{cases} \texttt{'global'} & \text{パラメータ設定はグローバル} \\ \texttt{''} & \text{パラメータ設定はローカル} \end{cases} \tag{1}$$

として自動的にセットされる[*19]．

■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

```
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，12.2 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}
```

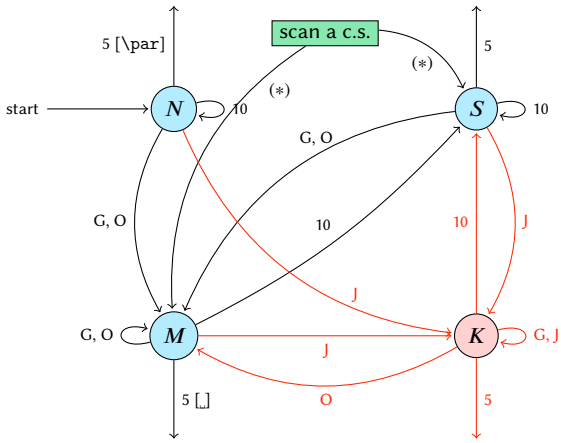
を実行し，TEX インターフェース側に「\ltjgetparameter{fuga} は追加引数が必要」ということを通知する必要がある．

13 和文文字直後の改行

13.1 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．

[*19] 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．



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 10. pTEX の入力処理部の状態遷移

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 10 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．

13.2 LuaTEX-ja の動作

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

すると，我々のとれる道は，`process_input_buffer` を用いて LuaTEX の入力処理部に引き渡さ

れる前に入力文字列を編集するというものしかない．以上を踏まえ，LuaTEX-ja における「和文文字直後の改行抑制」の処理は，次のようになっている：

> 各入力行に対し，**その入力行が読まれる前の内部状態で**以下の 3 条件が満たされている場合，LuaTEX-ja は U+FFFFF の文字[*20]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[*21]のカテゴリーコードが 5 (*end-of-line*) である．
> 2. U+FFFFF のカテゴリーコードが 14 (*comment*) である．
> 3. 入力行は次の「正規表現」にマッチしている：
>
> $$(\text{any char})^*(\textbf{JAchar})\big(\{\text{catcode} = 1\} \cup \{\text{catcode} = 2\}\big)^*$$

この仕様は，前節で述べた pTEX の仕様にできるだけ近づけたものとなっている．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          xyz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，コメント文字 U+FFFFF 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．

14 JFM グルーの挿入，kanjiskip と xkanjiskip

14.1 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

[*20] 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．
[*21] 普通は，改行文字（文字コード 13 番）である．

2 つの ⟨*char_node*⟩ の間には kanjiskip がある」ものとみなされる．

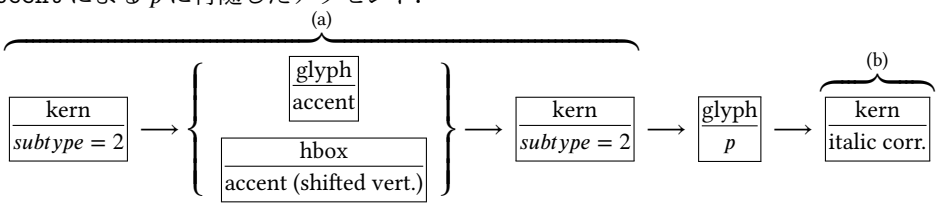
しかし，LuaTEX-ja では，hbox へのパッケージングや行分割前に全ての **JAglue**，即ち JFM グルー・xkanjiskip・kanjiskip の 3 種類を一度に挿入することになっている．これは，LuaTEX において欧文の合字・カーニング処理がノードベースになったことに対応する変更である．

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．

14.2 「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．



   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．
- *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．

14.3 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.

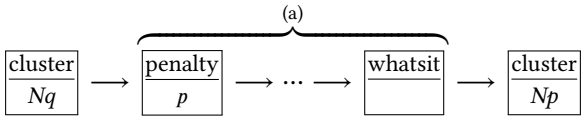
■末尾の処理 末尾の処理は，問題のリストが段落のものか hbox のものかによって異なる．後者の場合は容易い：最後のクラスタを *Nq* とおくと，*Nq* と「文字コード `'boxbdd'` の文字」の間に入るグルー／カーンを，*Nq* の直後に挿入するのみである．

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[*22]．

14.4 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，



[*22] 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．

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：

$$\boxed{\begin{array}{c}\text{cluster}\\ \hline Nq\end{array}} \longrightarrow \overbrace{\boxed{\begin{array}{c}\text{penalty}\\ \hline p+x\end{array}} \longrightarrow \cdots \longrightarrow \boxed{\begin{array}{c}\text{whatsit}\\ \hline \end{array}}}^{\text{(a)}} \longrightarrow \boxed{\begin{array}{c}\text{glue or kern}\\ \hline \text{右空白}\end{array}} \longrightarrow \boxed{\begin{array}{c}\text{cluster}\\ \hline Np\end{array}}$$

以後，**典型的な例として，クラスタ *Nq* と *Np* が共に和文 A である場合を見ていこう**，この場合が全ての場合の基本となる．

■「右空白」の算出　まず，「右空白」にあたる量を算出する．通常はこれが，隣り合った 2 つの **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 $\langle ratio \rangle$ の値を d_b, d_a とする．

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

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

     ここで．$f(x, y)$ は

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

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

     $$f(gb, ga)$$

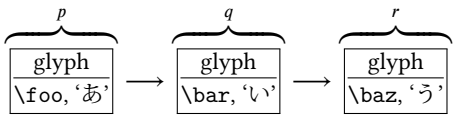
     ここで．$f(x, y)$ は

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

例えば，

```
\jfont\foo=psft:Ryumin-Light:jfm=ujis;-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．

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 パラメタが両方とも false だった場合は，長さ 0 の glue とする．
2. ユーザ側から見た kanjiskip パラメタの自然長が `\maxdimen` $= (2^{30}-1)\,sp$ でなければ，kanjiskip パラメタの値を持つ glue を採用する．
3. 2. でない場合は，*Nq*, *Np* で使われている JFM に指定されている kanjiskip の値を用いる．どちらか片方のクラスタだけが **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，

$$a := (Nq^{*23}\text{の文字に対する postbreakpenalty の値}) + (Np^{*24}\text{の文字に対する prebreakpenalty の値})$$

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

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

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

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

- 「右空白」がカーンであるとき，それは「*Nq* と *Np* の間で改行は許されない」ことを意図している．そのため，この場合は $a \neq 0$ であってもペナルティの挿入はしない．
- 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．

14.5　その他の場合

本節の内容は表 12 にまとめてある．

*24 厳密にはそれぞれ *Nq.tail*，*Np.head*．

表 12. JFM グルーの概要

Np ↓	和文 A	和文 B	欧文	箱	glue	kern
和文 A	$\frac{M \to K}{PN}$	$\frac{O_A \to K}{PN}$	$\frac{O_A \to X}{PN}$	$\frac{O_A}{PA}$	$\frac{O_A}{PN}$	$\frac{O_A}{PS}$
和文 B	$\frac{O_B \to K}{PA}$	$\frac{K}{PS}$	$\frac{X}{PS}$			
欧文	$\frac{O_B \to X}{PA}$	$\frac{X}{PS}$				
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．

■和文 A と欧文の間　*Nq* が**和文** A で，*Np* が**欧文**の場合，JFM グルー挿入処理は次のようにして行われる．

- 「右空白」については，まず以下に述べる Boundary-B [O_B] により空白を決定しようと試みる．それが失敗した場合は，xkanjiskip [X] によって定める．
- 禁則用ペナルティも，以前述べた P-normal [PN] と同じである．

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

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

xkanjiskip [X] この段階では，kanjiskip [K] のときと同じように，以下で定めた量を「右空白」として採用する．`\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 パラメタが共に false である．
   - *Nq* の中身の文字コードについて，「直後への xkanjiskip の挿入」が禁止されている（つまり，jaxspmode (or alxspmode) パラメタが 2 以上）．
   - *Np* の中身の文字コードについて，「直前への xkanjiskip の挿入」が禁止されている（つまり，jaxspmode (or alxspmode) パラメタが偶数）．
2. ユーザ側から見た xkanjiskip パラメタの自然長が `\maxdimen` $= (2^{30} - 1)\,sp$ でなければ，xkanjiskip パラメタの値を持つ glue を採用する．

3. 2. でない場合は，*Nq*, *Np*（**和文** A/**和文** B なのは片方だけ）で使われている JFM に指定されている xkanjiskip の値を用いる．

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

- 「右空白」については，まず以下に述べる Boundary-A [O_A] により空白を決定しようと試みる．それが失敗した場合は，xkanjiskip [X] によって定める．
- 禁則用ペナルティは，以前述べた P-normal [PN] と同じである．

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

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

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

- 「右空白」については，既に述べた Boundary-B [O_B] により空白を決定しようと試みる．それが失敗した場合は，「右空白」は挿入されない．
- 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．

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，既に述べた Boundary-A [O_A] により空白を決定しようと試みる．それが失敗した場合は，「右空白」は挿入されない．
- 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．

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

表 13. yoffset and imaginary body

yoffset	10 pt	5 pt	0	−5 pt	−10 pt
仮想ボディ	y H	y H	y H	y H	y H

15 ベースライン補正の方法

15.1 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 ≥ 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 13 参照）．

15.2 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，……

16 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 U+FFFFF 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．

16.1 注意

■LaTeX へのエスケープ 日本語対応を行うために process_input_buffer を使用したことで，texcl, escapeinside といった「**LaTeX へのエスケープ」中では，JAchar を名称の一部に含む制御綴は使用不可能**である．例えば次のような入力を考えよう：

```
\begin{lstlisting}[escapechar=\#]
#\ほげ␣x ぴよ#
\end{lstlisting}
```

ここで，2 行目は process_input_buffer の作用により，

#\[OFF]ほ[OFF]げ␣x[OFF]ぴ[OFF]よ#

と変換されてから，実際の処理に回される．「#」で挟まれた「LaTeX へのエスケープ」中では U+FFFFF のカテゴリーコードは 9 (*ignored*) となるので，結局「\ほげ」の代わりに「\[OFF]」という control symbol が実行されることになる．

■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．もしも IVS サポート（11.2 節）が有効になっていた場合は，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[OEO100]城市，葛[OEO101]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[OEO100]城市，葛[OEO101]飾区，葛西
\end{lstlisting}
```

```
葛[VS17]城市，葛[VS18]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{%
  \textcolor{blue}{\raisebox{3.5pt}{\tt%
    \fboxsep=0.5pt\fbox{\tiny \oalign{0#1#2\crcr#3#4#5\crcr}}}}%
}
{\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   :
```

16.2 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：

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

上の表において，

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

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．

17　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述べる．大まかに述べると，次のようになる．

- 通常の TEX の行分割方法に従って，段落を行分割する．この段階では，行長に半端が出た場合，その半端分は **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．
  `luatexja-adjust` の作用は，この処理を行う callback を追加するだけであり，この章の残りでは callback での処理について解説する．

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

よって，以降，問題にしている行の行長調整は伸縮度が有限長のグルーを用いて行われているとして良い．さらに，簡単のため，この行はグルーが広げられている（自然長で組むと望ましい行長よりの短い）場合しか扱わない．

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[*25]別にまとめられる）
- 和欧文間空白 (xkanjiskip)
- 和文間空白 (kanjiskip)

の $1+1+5+1=8$ つに類別し，それぞれの種別ごとに許容されている伸び量（`stretch` の値）の合計を計算する．また，行長と自然長との差を *total* とおく．

17.1 行末文字の位置調整

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

例えば，行末文字が句点「。」であり，そこで用いられている JFM 中に

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

という指定があった場合，この行末の句点は

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

となる．

行末文字を移動した場合，その分だけ *total* の値を引いておく．

17.2 グルーの調整

total の分だけが，行中のグルーの伸び量に応じて負担されることになる．負担するグルーの優先度は以下の順であり，できるだけ kanjiskip を自然長のままにすることを試みている．

(A) **JAglue** 以外のグルー
(B) 優先度 2 の JFM グルー
(C) 優先度 1 の JFM グルー
(D) 優先度 0 の JFM グルー
(E) 優先度 −1 の JFM グルー
(F) 優先度 −2 の JFM グルー
(G) xkanjiskip
(H) kanjiskip

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

[*25] 7.4 節にあるように，各 JFM グルーには −2 から 2 までの優先度がついている．

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．

3. *total* が「(A) の伸び量の合計」以上ならば，(A)–(H) 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，

$$total = ((\mathrm{A})\text{–}(\mathrm{B})\ \text{の伸び量の合計}) + p \cdot ((\mathrm{C})\ \text{の伸び量の合計}), \qquad 0 \le p < 1$$

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)–(H) 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)–(H) 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)–(H) 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．

18 IVS 対応

`luatexja.otf.enable_ivs()` を実行し，IVS 対応を有効にした状態では，`pre_linebreak_filter` や `hpack_filter` コールバックには次の 4 つが順に実行される状態となっている：

`ltj.do_ivs` *glyph_node p* の直後に，異体字セレクタ（を表す *glyph_node*）が連続した場合に，*p* のフォントに対応したが持つ「異体字情報」に従って出力するグリフを変える．
しかし，単に *p.char* を変更するだけでは，後から font feature の適用（すぐ下）により置換される可能性がある．そのため，`\CID` や `\UTF` と同じように，*glyph_node p* の代わりに `user_id` が *char_by_cid* であるような user-defined whatsit を用いている．

（`luaotfload` による **font feature の適用**）

`ltj.otf` `user_id` が *char_by_cid* であるような user-defined whatsit をきちんと *glyph_node* に変換する．この処理は，`\CID`, `\UTF` や IVS による置換が，font feature の適用で上書きされてしまうのを防止するためである．

`ltj.main_process` **JAglue** の挿入処理（14 章）と，JFM の指定に従って各 **JAchar** の「寸法を補正」することを行う．

問題は各フォントの持っている IVS 情報をどのように取得するか，である．`luaotfload` はフォント番号 ⟨*font_number*⟩ の情報を `fonts.hashes.identifiers[`⟨*font_number*⟩`]` 以下に格納している．しかし，OpenType フォントの IVS 情報は格納されていないようである[*26]．

一方，LuaTEX 内部の `fontloader` の返すテーブルには OpenType フォントでも TrueType フォントでも IVS 情報が格納されている．具体的には……

[*26] TrueType フォントに関しては，

`fonts.hashes.idenfiers[`⟨*font_number*⟩`].resources.variants[`⟨*selector*⟩`][`⟨*base_char*⟩`]`

に，⟨*base_char*⟩ 番の文字の後に異体字セレクタ ⟨*selector*⟩ が続いた場合に出力すべきグリフが書かれてある．

表 14. cid key and corresponding files

cid key	name of the cache	used CMaps	
Adobe-Japan1-*	ltj-cid-auto-adobe-japan1.lua	UniJIS2004-UTF32-*	Adobe-Japan1-UCS2
Adobe-Korea1-*	ltj-cid-auto-adobe-korea1.lua	UniKS-UTF32-*	Adobe-Korea1-UCS2
Adobe-GB1-*	ltj-cid-auto-adobe-gb1.lua	UniGB-UTF32-*	Adobe-GB1-UCS2
Adobe-CNS1-*	ltj-cid-auto-adobe-cns1.lua	UniCNS-UTF32-*	Adobe-CNS1-UCS2

そのため，LuaTEX-ja の IVS 対応においては，LuaTEX 内部の fontloader を直接用いることで，フォントの IVS 情報を取得している．20140114.0 以降でキャッシュを用いるようにした要因はここにあり，fontloader の呼び出しでかなり時間を消費することから，IVS 情報をキャッシュに保存することで 2 回目以降の実行時間を節約している．

19 複数フォントの「合成」（未完）

20 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）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．
  - LuaTEX と LuaJITTEX ではバイトコードの形式が異なるため，バイナリ形式のキャッシュは共有できない．LuaTEX 用のバイナリキャッシュは .luc，LuaJITTEX 用のは .lub と拡張子を変えることで対応している．
  - 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．

20.1 キャッシュの使用箇所

LuaTEX-ja では以下の 3 種類のキャッシュを使用している：

ltj-cid-auto-adobe-japan1.lua
: 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．
  30 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 14 を参照して欲しい．

extra_***.lua
: フォント “***” における異体字セレクタの情報，縦組用字形への変換テーブル，そして縦組時における幅を格納している．構造は以下の通り：

```
return {
 {
  [10955]={    -- U+2ACB "Subset Of Above Not Equal To"
   [65024]=983879, -- <2ACB FE00>
   ["vwidth"]=0.98, -- vertical width
  },
  [37001]={    -- U+9089 "邉"
   [0]=37001,  --   <9089 E0100>
   991049,     --   <9089 E0101>
   ...
   ["vert"]=995025, -- vertical variant
  },
  ...
 },
 ["chksum"]="FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF", -- checksum of the fontfile
 ["version"]=2, -- version of the cache
}
```

`ltj-jisx0208.{luc|lub}`

LuaTEX-ja 配布中の `ltj-jisx0208.lua` をバイナリ化したものである．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．

20.2 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 3 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` のみならず，そのバイナリ版も作成/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．

`load_cache` は，まずバイナリキャッシュ ⟨*filename*⟩`.{luc|lub}`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」ならばテキスト版 ⟨*filename*⟩`.lua` を読み込み，その値を返す．

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．

21 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．

21.1 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[*27]．
- `\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[*28]．

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` 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．

[*27] 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．

[*28] 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．

21.2 *dir_box*

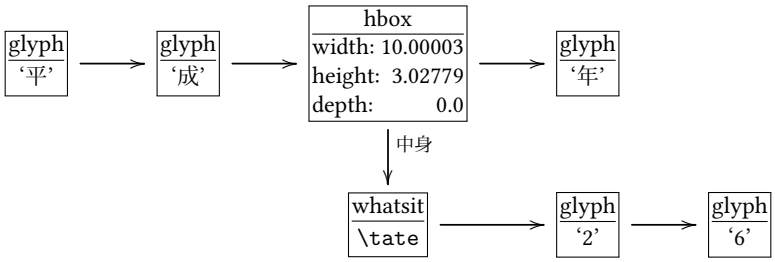
縦中横など異方向のボックスを配置する場合に，周囲の組方向と大きさを整合させるため，LuaTEX-ja では `\ltj@dir` が 128 以降の *hlist_node*, *vlist_node* を用いる．これらは pTEX における *dir_node* の役割と同じ果たしており，この文章中では *dir_box* と呼称する．

21.2.1 異方向のボックスの整合

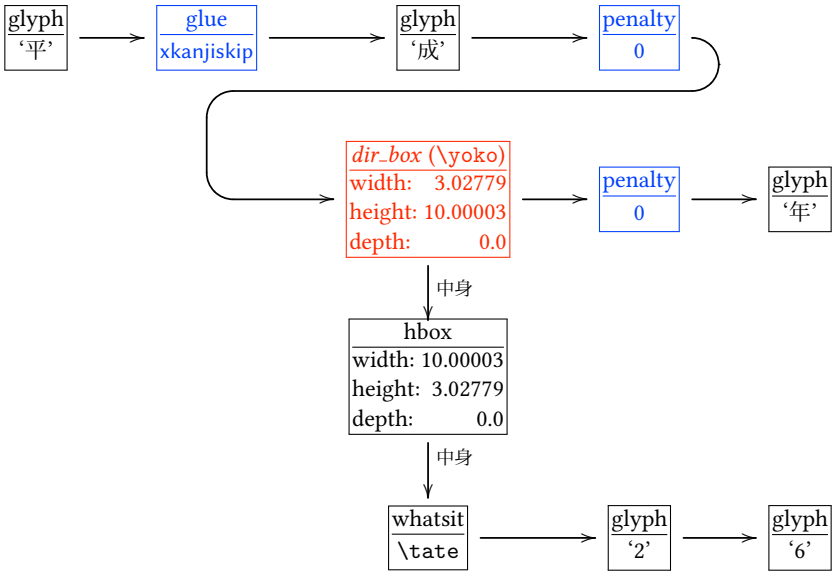
dir_box の第一の使用目的は，異方向のボックスの大きさを整合させることである．例えば，

```
% 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．

21.2.2 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．

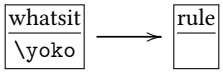
例えば

```
1 \setbox0=\hbox{\vrule width 10pt height 5pt depth 2pt}
2 \setbox1=\hbox{\tate\ltjsetwd0=20pt}
3 \wd0=9pt
4 \setbox1=\hbox{\dtou\ltjsetwd0=20pt}
5 \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，

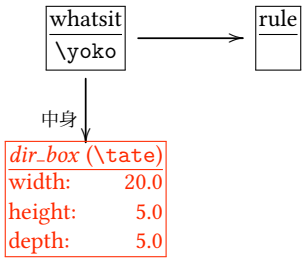
- 縦組下では \box0 は幅 7 pt，高さ・深さ 5 pt のボックスとして扱われる．
- \dtou 下では \box0 は幅 7 pt，高さ 10 pt，深さ 0 pt のボックスとして扱われる．

このとき，\box0 の**中身**は



である．

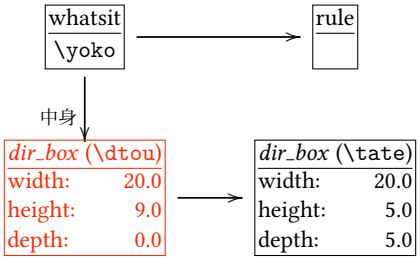
さて，2 行目で縦組時の \box0 の幅が 20 pt に設定される．この情報が direction whatsit 内部のノードリストに，*dir_box* として格納される：



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

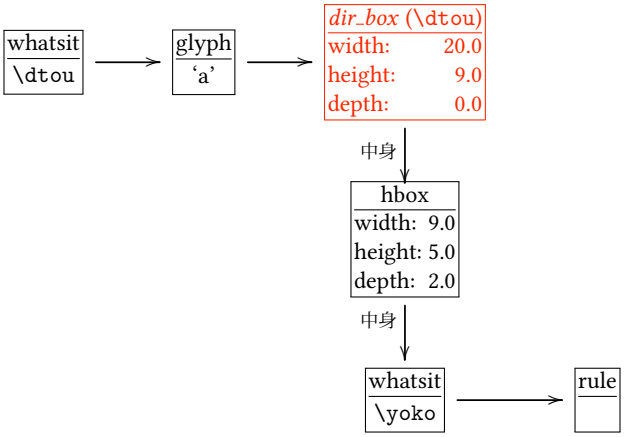
- 縦組下では寸法代入が既に行われているので，2 行目で作成された *dir_box* の通りに幅 20 pt，高さ・深さ 5 pt のボックスとして扱われる．
- \dtou 下ではまだ寸法代入が行われていないので，\box0 の寸法変更に追従し，幅 7 pt，高さ 9 pt，深さ 0 pt のボックスとして扱われる．

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．

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.sourceforge.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