bxluafacade パッケージ（v0.2c）

八登崇之 （Takayuki YATO; aka. “ZR”）

2012/06/05

1 概要

本パッケージは以下の機能を提供する。

- LaTeX 上の多少複雑な処理を Lua プログラムを用いて解決する際に、必要となる機能を用意する。
- 主に、LaTeX と Lua の間のインタフェースに関わる機能である。
- 従って、本パッケージは LaTeX と Lua の知識さえあれば、TeX の知識がなくとも使えるようにすることを目指す。

■前提環境

- TeX フォーマット： LuaLaTeX

■依存パッケージ

- etoolbox パッケージ
- luatexbase パッケージ

2 パッケージの読込

\usepackage で読み込む。オプションはない。

```
\usepackage{bxluafacade}
```

3 基本的な機能（LaTeX 側）

本節の説明中の「例」においては以下に示す定義が行われていることを前提とする。

```
\newcommand*{\macroA}{\macroB}
\newcommand*{\macroB}{I'd}
\newcommand*{\Answer}{42}
\newcounter{answer} \setcounter{answer}{42}
\newlength{\sample} \setlength{\sample}{1ptplus1pt}
```

- `\luaescape{`⟨テキスト⟩`}` ： `\luastring` と同様に、引数（完全展開する）を文字列化したものに Lua 文字列リテラルのためのエスケープを施すが、`" "` では囲わない。
  例： `s = "\luaescape{\macroA}"` →（展開：`s = "I\'d"`）→ `s` は “I’d”
- `\luaescapeO{`⟨テキスト⟩`}` ： `\luastringO` と同様に、引数（一回展開する）を文字列化したものに Lua 文字列リテラルのためのエスケープを施すが、`" "` では囲わない。
  例： `s = "\luaescapeO{\macroA}"` →（展開：`s = "\\macroB "`）→ `s` は “\macroB ”
- `\luaescapeN{`⟨テキスト⟩`}` ： `\luastringN` と同様に、引数（展開しない）を文字列化したものに Lua 文字列リテラルのためのエスケープを施すが、`" "` では囲わない。
  例： `s = "\luaescapeN{\macroA}"` →（展開：`s = "\\macroA "`）→ `s` は “\macroA ”
- `\luanumber{`⟨数値⟩`}` ： 引数（完全展開される）を Lua で数値として扱う式に変換する。引数が直接数字列で書かれた数値でない場合は nil と解釈される。
  例： `n = \luanumber{\Answer}` →（展開：`n = tonumber("42")`）→ `n` は 42
  例： `n = \luanumber{6*9}` →（展開：`n = tonumber("6*9")`）→ `n` は nil
  例： `n = \luanumber{\value{answer}}` →（展開：`n = tonumber("\c@answer ")`）→ `n` は nil
- `\luanumberN{`⟨数値⟩`}` ： 引数（展開しない）を Lua で数値として扱う式に展開する。引数が直接数字列で書かれた数値でない場合は nil と解釈される。
  例： `n = \luanumberN{\Answer}` →（展開：`n = tonumber("\Answer ")`）→ `n` は nil
  例： `n = \luanumberN{3.14159}` →（展開：`n = tonumber("3.14159")`）→ `n` は 3.14159
- `\luacounterval{`⟨カウンタ名⟩`}` ： LaTeX カウンタの現在の値を Lua で数値として扱う式に展開する。
  例： `n = \luacounterval{answer}` →（展開：`n = (42)`）→ `n` は 42
- `\lualengthval{\`長さ変数名`}` ： 長さ変数の現在の値を Lua でグル—値（gluespec）として扱う式に展開する。
  例： `g = \lualengthval{\sample}` →（展開：`g = bxlt.to_skip("1pt plus 1pt")`）→ `g` は「1pt plus 1pt」を表す gluespec

4 基本的な機能（Lua 側）

- `bxlt.print(`[*number* *c*`,`] *string* *s*`, ...)` ： 引数の各文字列を改行文字（`'\n'`）毎に分割して `tex.print(`[*c*`,`] `...)` に渡す（文字列の末尾の改行は無視する）。結果的に、文字列の中の各行が TeX の 1 つの入力行と扱われる。
  例： 例えば `tex.print("A%B\nC")` は「A」しか出力しないが、`bxlt.print("A%B\nC")` は `tex.print("A%B", "C")` と同じで「A C」と出力する。
- `bxlt.printf(`[*number* *c*`,`] *string* *f*`, ...)` ： 書式付で TeX にテキスト出力（print 版）。`bxlt.print(`[*c*`,`] `string.format(`*f*`, ...)` と同じ。
  例： `bxlt.printf("%02X", 42)` は「2A」を TeX に書き出す。
- `bxlt.writef(`[*number* *c*`,`] *string* *f*`, ...)` ： 書式付で TeX に文字列出力（write 版）。`tex.write(`[*c*`,`] `string.format(`*f*`, ...)` と同じ。
  例： `bxlt.writef("[%s]", "C#")` は verbatim に（カテゴリコード 12 で）`[C#]` を TeX に書き出す。

- `bxlt.sprintf(`[*number c*`,]` *string f*`, ...)` ： 書式付で TEX にテキスト出力（sprint 版）。[*1] `bxlt.sprint(`[*c*`,] string.format(`*f*`, ...)` と同じ。
- (1) *number sp* = `bxlt.to_dimen(nil)`
  (2) *number sp* = `bxlt.to_dimen(`*number d*`)`
  (3) *number sp* = `bxlt.to_dimen(`*string d*`)` ： 長さの表現を、sp 単位の整数値に変換する。(1) は 0 を返す。(2) は *d* をそのまま返す。(3) は *d* を TEX での長さ表記（単位付き数値）として解釈して sp 単位の整数値を返す。
  例： `sp = bxlt.to_dimen(26214)` → `sp` は 26214。
  例： `sp = bxlt.to_dimen("1cm")` → `sp` は 1864679。
- (1) *gluespec gs* = `bxlt.to_skip(nil)`
  (2) *gluespec gs* = `bxlt.to_skip(`*number s*`)`
  (3) *gluespec gs* = `bxlt.to_skip(`*gluespec s*`)`
  (4) *gluespec gs* = `bxlt.to_skip(`*string s*`)` ： グル—値（伸縮する長さ）の表現を、gluespec に変換する。(1) は「0pt」の gluespec を返す。(2) は「*s* sp」の gluespec をそのまま返す。(3) は *s* 自身を返す。(4) は *s* を TEX でのグル—値表記として解釈して結果の gluespec を返す。
  例： `sp = bxlt.to_skip(16384)` → `sp` は「0.25pt」の gluespec。
  例： `sp = bxlt.to_dimen("1pt minus .1pt")` → `sp` は「1pt minus 0.1pt」の gluespec。

5 Quick-escape

LATEX コード中で `\directlua` で Lua コードを記述する場合、LATEX の特殊文字（\ や %）はその特殊な意味を保つので、そのままでは記述することができない。その場合はできるだけ別ファイル（Lua ソースファイル）に Lua コードを移動するか、あるいは、luacode パッケージが提供する `luacode*` 関数を利用することが通常の方法である。

しかし、場合によっては `\directlua` の中に特殊文字を書けた方が便利であると考える人もいるであろう。Quick-escape はそのような要求を満たすための機能である。

Quick-escape を有効にすると、Backtick 記号〈‘〉とその後に続く 1 つのトークンについて、以下の変換が行われる。（a は任意の ASCII 英字、* は任意の ASCII 非英字を表す。）

- ‘* は * に展開される。よって、任意の LATEX 特殊文字（全て ASCII 非英字である）について、前に ‘\ を前置することで Lua コードに含めることができる。
  例： `("[‘\%3d]‘\\n", ‘\#str)` → `("[%3d]\n", #str)`
- ‘a は ‘a に展開される。
  例： `p("‘Yes!’")` → `p("‘Yes!’")`
- ‘* の組み合わせのうち一部は以下のように変換される。

+	&	|	-	(	)	<	>
↓	↓	↓	↓	↓	↓	↓	↓
#	%	\	~	{	}	^	\\

[*1] C 言語の `sprintf()` に相当するのは `string.format()`。

例： `p("‘>def‘-‘+1‘(‘)")` → `p("\\def~#1{}")`

- **以上に挙げたもの以外の組み合わせについては未定義である。**

Quick-escape **は既定では無効になっている。以下の命令を用いて有効／無効を切り替える。**

- `\usequickescape` **：** Quick-escape **を有効にする。**
  ※‘ のカテゴリコードを（ローカルに）13 **にする。初回の実行では以下の初期化処理がグローバルに行われる：アクティブな‘ を** `\qesc` **と等価にし、‘ のカテゴリコードを** 12 **にする。**
- `\nousequickescape` **：** Quick-escape **を無効にする。**
  ※‘ のカテゴリコードを（ローカルに）12 **にする。**
- `\finishquickescape` **：** Quick-escape **の使用を終了。**
  ※アクティブな‘ の意味と‘ のカテゴリコードを初期化処理の直前の状態にグローバルに戻す。これを実行してしまうと、それまで quick-escape **を用いて定義していた命令（マクロ）が正しく動作しなくなることに注意せよ。**
- `\qesc` **：** Quick-escape **の処理の実体。**Quick-escape **が無効の場合でも、‘ の代わりに** `\qesc` **を用いて** quick-escape **が実行できる。**