idtagreplacer -- InDesign tag replacer

概要

■これはなに?

idtagreplacer は、テキストに埋め込んだ編集記号を InDesign タグに変換するためのプログラムです。

たとえば、

```
◆H1◆概要
◆H2◆これはなに？
テキストに埋め込んだ編集記号を InDesign タグに変換するためのプログラムです。
編集記号は、設定ファイルで任意に定義することができます。
```

という原稿ファイルを作成したとしましょう。このファイルでは約束事として「◆H1◆」「◆H2◆」を"編集記号"と定めていて、「◆H1◆」は「大見出し」、「◆H2◆」は「中見出し」、何も指定していない行は「本文」という意味を、それぞれ想定しているとします。

通常、こうした原稿ファイルを DTP に出す場合、DTP オペレータに記号の意味も伝えることになります。そして、オペレータは渡された原稿から編集記号を見付け、対応したスタイルを手作業で適用していきます。

しかしこうした作業は、機械的な単純作業であり、わざわざ人が行う価値のあるものとは言えません。この点 idtagreplacer を使えば、編集記号とスタイルの対応を設定しておくだけで、スタイルの適用された InDesign タグ付きテキストを生成することができ、DTP で発生する単純作業の量を軽減できます。また、こうした自動化は、時間の節約や正確さの保証にもつながることでしょう。

利用する編集記号やスタイルの名前は、設定ファイルで任意に定義できます。

どうやってインストールするの?

プログラム本体を利用したい PC にコピーします。必要なのは、

- idtagreplacer.jar
- tagconf.xml
- lib フォルダ
- log フォルダ

の4点です。

idtagreplacer.jar はプログラムの本体です。

tagconf.xml は編集記号の定義ファイルです。

lib フォルダは、依存ライブラリを入れるフォルダです。ここには js.jar ファイルが必要になります。これ

は JavaScript を起動するためのライブラリです。

log フォルダはログファイルを保存するためのフォルダです。これは空のフォルダで構いません。

また、idtagreplacer は Java のプログラムですから、Java の実行環境が必要です。

基本的に Java SE 5.0 で動くことを確認しています。

■どうやってアンインストールするの?

インストールのときにコピーしたファイルやフォルダー式を削除してください。

■どうやって実行するの?

1. あらかじめ自分用に tagconf.xml を編集しておきます。
2. idtagreplacer.jar をダブルクリックします。するとダイアログが表示されます。
3. 処理したい原稿ファイルを、そのダイアログにドラッグ & ドロップします。
4. すると、原稿ファイルと同じ場所に、処理結果のファイルが作成されます。

なお、このプログラムは InDesign のスタイル自体を作成するわけではありません。ですから、流し込みたい InDesign のフォームには、あらかじめスタイルが設定されていなければなりません。

試してみましょう

idtagprelacer を使った一連の作業の流れを、実際に手を動かしながら簡単になぞってみましょう。

■サンプルファイルの準備と内容確認

まず、InDesign のフォームを用意します。そのフォームにどんなスタイルが設定されてあるかを確認するには、 InDesign で開き、コマンド + F11 を押します。すると、段落スタイルのウィンドウが現れ、設定済みの段落スタイルを確認することができます。続いてコマンド + Shift + F11 を押します。すると、文字スタイルのウィンドウが現れ、設定済みの文字スタイルを確認することができます。idtagreplacer を使う場合、流し込む先となる InDesign のフォームに、このようにあらかじめスタイルが設定されている必要があります。

次に、インストールした idtagreplacer のフォルダにある tagconf.xml の内容を確認しましょう。このファイルには、あらかじめサンプルとなる設定情報が入力されています。テキストエディタで開いてみましょう。

このファイルの中心部分は、

```
<entry key="段落タグ設定"><![CDATA[
...
]]></entry>
```

の部分と、

```
<entry key="文字タグ設定"><![CDATA[
...
]]></entry>
```

の部分です。
それぞれの設定内容を見ると、各行に必ず 1 つ「:」があります。この「:」の右側に書かれているのがスタイル名です。これは、先ほどの InDesign のマスターフォームに設定されているスタイル名と対応しています。
一方「:」の左側に書かれているのが編集記号の設定情報です。つまりこの設定情報は、左側に書かれた編集記号が、右側に書かれたスタイル名に対応するという仕組みになっています。

以上で、あらかじめ用意しておくべきファイルの確認は終わりです。

■原稿ファイルの作成

原稿ファイルを作成します。サンプルの tagconf.xml で設定されているタグをそのまま使って、次のような内容のファイルを作成することにしましょう。

```
■■■はじめに
◆b/◆idtagreplacer◆/b◆とInDesignタグの世界へようこそ！
```

ファイルは (サンプルの tagconf.xml の設定に従って) Shift-JISで保存します。ファイル名は、source.txt としておきましょう。

■InDesignタグ付きテキストへの変換

作成した原稿ファイルをInDesignタグ付きテキストに変換します。手順は次のとおりです。

1. **idtagreplacer を起動します**
   インストールした idtagreplacer.jar をダブルクリックしてください。ダイアログが表示されます。
2. **原稿ファイルをドラッグします**
   表示されたダイアログの下部に「編集記号付きテキストファイルをドラッグしてください。」というメッセージが表示されたら、先ほど作成した原稿ファイル (source.txt) をこのダイアログにドラッグ&ドロップしてください。原稿ファイルのあるフォルダに、InDesignタグ付きテキストに変換されたファイルが作成されるはずです。ファイル名は (サンプルの tagconf.xml の設定に従って) source_id.txt となっているはずです。

■InDesignフォームへの配置

InDesignのマスターフォームを開き、作成したタグ付きテキストを配置します。手順は次のとおりです。

1. **マスターフォームを開きます**
   マスターフォームのファイルをダブルクリックして、開いてください。
2. **「横組みグリッドツール」を選択します**
   InDesignのツールボックス（デフォルトで左端に表示されるフローティングバー）にある「横組みグリッドツール」ボタンをクリックします。
   「横組みグリッドツール」ボタンの場所が分からない場合は、（IM を英字入力に切り替えて）キーボードの「y」をタイプします。これで「横組みグリッドツール」ボタンをクリックしたのと同じことになります。
   「横組みグリッドツール」を選択すると、マウスカーソルが細い十字のようなかたちに変わります。
3. **グリッドフレームを作成します**
   マスターフォームに設定されているガイドに沿って、まず段組みの左段をドラッグします。左段の左上から右下へとマウスをドラッグすると、升目のついたフレームが作成されます。これがグリッドフレームです。
4. **タグ付きテキストを配置します**
   先の手順で作成したグリッドフレームが選択されている状態（白い小さな□が付いている状態）で、メニューの「ファイル」→「配置...」を選択します。「配置」ダイアログが表示されますので、先ほど作成したタグ付きテキストを選択します。ダイアログ下部にある「グリッドフォーマットの適用」にチェックがついている場合はそれを外し、「開く」ボタンをクリックします。
   「配置」ダイアログは、Command+Dでも表示できます。
   なお、グリッドフレームの選択が外れてしまった場合は、ツールボックスから「選択ツール」ボタン（黒い矢印）をクリックして、グリッドフレームをクリックし選択し直します。
5. **（テキストの続きを配置します）**
   今回のサンプルテキストは短いので、上の手順までですべての読み込みが完了しますが、長いテキストの場合、左段だけではテキストが溢れてしまいます。溢れたテキストがある場合、グリッドフレームの右下に赤い四角が現れます。
   溢れたテキストを配置するには、ツールボックスから「選択ツール」ボタン（黒い矢印のボタン）をクリックし、グリッドフレームの右下の赤い四角をクリックします。するとマウスカーソルにテキストの一部が表示されるようになります。
   この状態で、マウスカーソルを右段の左上に持っていき、Shift+クリックします。すると、続きのテキストが自動的にすべて配置されます。

以上の手順で、グリッドにテキストが配置されたはずです。配置されたテキストには、すでにスタイルが適用されていることを確認してください。

tagconf.xml の書き方

■はじめに

tagconf.xml は、とても単純な XML ファイルです。構造としては Java のプロパティーファイルの書式で書かれています。つまり、次のような構造です。

```
<?xml version="1.0" encoding="Shift_JIS"?>

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">

<properties>
```

```
<entry key="key1">value1</entry>

<entry key="key2">value2</entry>

...

</properties>
```

entry要素のkey属性で指定された値がプロパティー名になります。そして、entry要素の値が、そのプロパティー名に対応する値となります。……要するに、上記の例で言えば、key1、key2 がプロパティー名です。プロパティー key1 の値が value1、key2 の値が value2 ということになります。

以下、プロパティー名ごとに、tagconf.xml の設定内容を説明していきます。既存の tagconf.xml を見ながら読んでいただくとよいでしょう。

■エンコード

原稿ファイルの文字コードを指定します。Shift_JIS もしくは UTF-8 のいずれかを指定できます。

■改行コード

出力ファイルの改行コードを指定します (原稿ファイルの改行コードは問いません) 。InDesign に読み込ませる場合は、この値を CR と指定しておきます。

CRLF や LF と指定してもその通りに出力しますが、InDesign はそのファイルを正しく読み込めないかもしれません。出力ファイルを目視で確認したい場合などの例外を除いて、基本的に CR のままにしておきます。

■保存ファイル名

出力ファイルの名前です。出力ファイルは、原稿ファイルと同じディレクトリに作成され、自動的に命名されます。このときの命名ルールをカスタマイズするときに、この設定値を変更します。

設定値の %s には原稿ファイルのファイル名 (拡張子を除く) が入ります。

例：

原稿ファイル：foobar.txt

設定値： %s_id.txt

↓

出力ファイル名は「foobar_id.txt」

■段落タグの既定値

デフォルトの段落スタイルを指定します。デフォルトの段落スタイルとは、無指定の段落を指します。

■段落タグ設定

段落タグの設定を列挙します。段落タグとは、段落スタイルに変換される編集記号のことです。
idtagreplacer では、次の 2 種類の段落タグが利用できます。

ー行段落タグ

行頭に置かれる編集記号で、その行のみにスタイルが適用されます。たとえば、原稿ファイルで下記のように使うタイプの編集記号です。

原稿ファイルでの使用例：

```
■■■はじめに
```

記号「■■■」に、たとえば「大見出し」のスタイルを対応させている場合、これは「はじめに」という大見出しを表現していることになります。ー行段落タグは、このように行頭にあって、その行を修飾するタイプの編集記号です。

○開始 / 終了段落タグ

開始を表す編集記号と、終了を表す編集記号の 1 組がセットになった編集記号です。
開始から終了までのすべての行にスタイルが適用されます。

原稿ファイルでの使用例：

```
◆リスト◆
#include <stdio.h>
int main(int argc, char** argv)
{
   puts("Hello");
   return 0;
}
◆/リスト◆
```

この例では、開始記号「◆リスト◆」と終了記号「◆/リスト◆」をもつ段落タグが使われています。開始記号から終了記号までの間は、この段落タグに対応したスタイルが適用されることになります。
なお、開始記号や終了記号だけが書かれた行は、変換に際して削除されます。

段落スタイルで挟むタグを利用する場合は1行以上にする必要があります。

×正常に変換できない

```
◆list/◆aa◆/list◆
```

○設定ファイルに記述する書式
段落タグの設定は、行単位で解釈されます。書式は次の通りです。

```
一行段落タグの記号 ：スタイル名
```

あるいは

```
開始記号　終了記号：スタイル名
```

開始記号と終了記号の間の空白文字 (スペースやタブ) は、いくつあっても構いません。
記号やスタイル名と「:」の間にある空白は、あってもなくても構いません。また空白を入れた場合、それはいくつ入っていても構いません。

○オプション
原則的に編集記号は、idtagreplacer が変換処理を行う際に削除されます。編集記号を削除したくない場合は、スタイル名の先頭に「!」をつけます。
また、スタイル名の先頭に「!!」をつけると、スタイル名と同じプロパティー名に記述された JavaScript を実行させることができます。

○注意点(1) -- 段落タグの記述順序
プログラムは、原稿ファイルを 1 行読み込むと、該当する段落タグがあるかどうかをチェックします。このときプログラムは、必ず設定ファイルに記述された順序に従って段落タグをチェックしていきます。該当する段落タグが見つかったら、それ以降はチェックせずに、原稿ファイルの次の行を読み込みます。

段落タグの記述順序はしばしば重要な意味を持ちます。
たとえば、「■」という段落タグと「■■」という段落タグを使いたい場合があったとします。このとき、

```
■　　：小見出し
■■　：大見出し
```

などのように書いてしまうと、後者の「大見出し」のスタイルは、一切適用されなくなってしまいます。これは、最初の「■」を見つけた段階でプログラムは「小見出し」と判定してしまい、「■■」をチェックしなくなってしまうからです。こうした場合、次のように順序を逆にして記述します。

```
■■　：大見出し
■　　：小見出し
```

○注意点(2) -- 段落の入れ子と終了記号
idtagreplacer では、段落スタイルは入れ子にして使えるようになっています。たとえば原稿ファイルで、

```
◆コラム◆
◆コラムタイトル◆入れ子の例
◆コラム見出し◆例 1
たとえば、こんな感じに書くこともできます。
◆コラム見出し◆例 2
…
◆/コラム◆
```

という感じの記述を行うこともできます。この例では、全体を「◆コラム◆」と「◆/コラム◆」で囲まれた中

に、さらに段落タグが設定されていることに注意してください。
こうした記述が可能であることは便利ではありますが、一方で、終了記号が記述されていないと見なされた場合、デフォルトの段落スタイルに戻らなくなる、という事態になることがあります。

○注意点(3) -- 開始記号が見つからず、終了記号だけが認識された場合
終了記号だけが認識された場合、プログラムはエラーを起こしてしまいます。
この問題は、しばしば一行段落タグとの関係で起こることがあります。次のような設定を行って、同じ編集記号で、一行段落タグと開始記号とを兼ねようとした場合がそれに当たります。

```
◆箇条◆                ：箇条書き
◆箇条◆　◆/箇条◆　：箇条書き
```

このとき、プログラムは一行段落タグのほうを優先して認識しますから（これは、先に記述してあるほうが優先されるからです）、開始記号は決して認識されることはありません。ですから、これに基づいて原稿ファイルを作成してしまうと、終了記号だけが認識されてエラーになる、という事態が発生します。

■文字タグ設定

文字タグの設定を列挙します。文字タグとは、文字スタイルに変換される編集記号のことです。
文字タグは、必ず開始記号と終了記号を持ちます。原稿ファイル内で、たとえば次のように利用する記号です。

原稿ファイルでの使用例：

```
などのように書いてしまうと、後者の「大見出し」のスタイルは、一切適用◆太字◆されなくなってしまいます◆/太字◆。
```

○設定ファイルに記述する書式
文字タグの設定は、段落タグに準じます。段落タグ同様、設定は行単位で解釈されます。
書式は次の通りです。

```
開始記号　終了記号：スタイル名
```

開始記号と終了記号の間の空白文字（スペースやタブ）は、いくつあっても構いません。

記号やスタイル名と「:」の間にある空白は、あってもなくても構いません。また空白を入れた場合、それはいくつ入っていても構いません。

○オプション

スタイル名の先頭に「!!」をつけると、スタイル名と同じプロパティー名に記述された JavaScript を実行させることができます。

なお、文字タグの編集記号は、idtagreplacer が変換処理を行う際に削除されます。段落タグと異なり、文字タグの場合は、スタイル名の先頭に「!」をつけて編集記号を残すことはできません。

■自由置換設定

自動的に置換したい文字列を列挙します。換言すれば、これは終了記号のない文字タグのようなものとも言えます。書き方は、

```
置換前の文字列：置換後の文字列
```

です。たとえば、

```
インターフェイス：インタフェース
```

と設定しておけば、原稿ファイル中に出てくる「インターフェイス」はすべて「インタフェース」に変換されます。これは単純な文字列置換ですが、置換後の文字列には InDesign タグを直接書くこともできますので、

```
●→●：<27A1>
```

のように、終了記号のない文字タグのような感覚で使うこともできます。

■黒丸数字書式

ここでは、JavaScript によって自由置換設定を追加しています。
「黒丸数字書式」以外にも、「白丸数字書式」や「黒四角数字書式」「アルファベット」がありますが、これらは、設定ファイルを読み込む際に、プログラムから固定的に呼び出されます。このプロパティー名は変更できませんが、もし必要なければ、プロパティーそのものを削除しても問題はないはずです。

これらの一連の JavaScript は、一定のパターンで作成されています。

```
format = "%d●";
code = 0x2776 - 1;
for (var i = 10; i >= 1; i--)
  App.replaceSigns.add(new ReplaceSign(format.replace('%d', i),
                           new ReplaceTag('<' + (code + i).toString(16) +
'>')));
```

最初の青い部分には、黒丸数字に変換したい書式が来ます。「%d」の部分に数字が入ります。

次の青い部分は、コードポイントです。この例は黒丸数字ですから、黒丸数字の 1 に当たる Unicode の値をここに書き込みます。Unicode は一般的に 16 進数で書き表しますので、プログラムでも 0x を付けて 16 進数で表現しています。

次の青い部分は、作成する上限値です。この場合 10 になっていますから、 1 から 10 までの黒丸数字の設定が作成されることになります。

■白丸数字書式

「黒丸数字書式」と同様です。

ここには、自由置換設定に白丸数字の設定を追加する JavaScript が書かれています。設定ファイルを読み込む際に、自動的に実行されます。

プロパティー名を変更すると実行されなくなりますが、このプロパティー自体を削除しても問題はないはずです。

■黒四角数字書式

これも「黒丸数字書式」と同様です。

ここには、自由置換設定に黒四角数字の設定を追加する JavaScript が書かれています。設定ファイルを読み込む際に、自動的に実行されます。

プロパティー名を変更すると実行されなくなりますが、このプロパティー自体を削除しても問題はないはずです。

■アルファベット

これも「黒丸数字書式」とほぼ同様です。

ここには、キーボードのキートップのような字形をもつアルファベットの設定を、自由置換設定に追加する JavaScript が書かれています。設定ファイルを読み込む際に、自動的に実行されます。

プロパティー名を変更すると実行されなくなりますが、このプロパティー自体を削除しても問題はないはずです。

以上で、プロパティーの説明は終了です。

■tagconf.xml 記述の注意点 -- idtagreplacer の変換処理の順序

idtagreplacer は、次のような順序でタグ変換を行います。

1. 「<」「>」を見付けたら、その前に「<005C>」を追加する
2. 段落タグの変換を行う
3. 文字タグの変換を行う
4. 自由置換の変換を行う

注意すべき点は、最初に実行される「<」「>」に対する特殊な変換処理です。

この変換は自由置換や文字タグ変換の前に実行されますので、たとえば自由置換設定や文字タグ・段落タグ設定で「<」や「>」を含む文字列にヒットさせたい場合、単純に「<」や「>」を設定ファイルに記述してしまうと、実行結果が不正になる場合があります。なぜなら、自由置換や文字タグ、段落タグの評価が行われる時点には、すでに「<」や「>」の前に「<005C>」が挿入されているからです。設定ファイルには明示的に「<005C>」を付けて記述する必要があります（※）。

※ 設定ファイルに記述しなくても、プログラムが自動的に設定情報に「<005C>」を付け足すという方法もありますが、いずれにしても動作が不規則であることは免れませんので、そのような実装にはなっていません。その代わり、一度自由置換で変換した InDesign タグを、さらに文字タグや段落タグとして変換する、というような場合には、比較的自然に記述できるようになっています。

■ユニコードとかわからない特殊な文字を追加する

InDesign上に入力されている文字をタグ変換させるものとして設定する（ウムラウトとか）

InDesignで作業

1. [ファイル]→[書き出し]
2. [InDesignタグ付きテキスト]を選択
3. [保存]
4. [略書き]、円コーディング[Shift-Jis]
5. [保存]
6. そのInDesign上で設定されているタグが全部書かれたテキストができる
7. 当てはまる部分をtagconf.xmlに記述
8. ちゃんと変換されるかタグ付テキストを作成してInDesignに流し込む

JavaScript 関係の資料

■プログラム固有のオブジェクト

通常の JavaScript のオブジェクト以外に、Java の標準 API やプログラム固有のオブジェクトを使うことができます。プログラム固有のオブジェクトには、以下のものがあります。

App
App には中心となるオブジェクトがまとめられています。次のものがあります。

App.activeParagraphTag
現在の段落タグが何であるのかを保持するオブジェクトです。実体は Java の LinkedList で、ここには ParagraphTag というオブジェクトが集められています。

App.characterSigns
設定ファイルから読み込んだ文字タグの全設定情報を配列化（リスト化）したものです。実体は Java の LinkedList で、ここには CharacterSign というオブジェクトが集められています。

App.paragraphSigns
設定ファイルから読み込んだ段落タグの全設定情報を配列化（リスト化）したものです。実体は Java の LinkedList で、ここには ParagraphSign というオブジェクトが集められています。

App.replaceSigns
設定ファイルから読み込んだ自由置換設定の全内容を配列化 (リスト化) したものです。実体は Java の LinkedList で、ここには ReplaceSign というオブジェクトが集められています。

App.printController
出力時の操作を管理するオブジェクトです。addCommand メソッドを使って、行うべき操作やその順序を指示します。と言っても、指示できるコマンドは「print」と「remove last para」、その他 (何もしない) の 3 種類だけです。
「print」コマンドは出力を指示することができます。「remove last para」は、段落タグの切り替えを指示することができます。ただし実際にこれが実行されるのは、addCommand メソッドを呼び出したタイミングではなく、現在処理している行の変換処理がすべて完了したタイミングです。

App.out
出力を担当するオブジェクトです。これはアプリケーション固有のオブジェクトで、次のようなメソッドを持っています。

- **print(str)**
  引数で渡された文字列を出力ファイルに出力します。プログラムはこのとき改行コードを加えませんし、行が続いているものと見なして動作します。
- **println(str)**
  引数で渡された文字列に改行コードを加えて、出力ファイルに出力します。プログラムは、行が終了したことを認識し、必要な段落タグの切り替え処理などを行います。
- **getLineFeedCode()**
  設定ファイルに設定された改行コードを返します。返すのは設定値そのままではなく、対応する制御文字のほうです。つまり、たとえば設定値が CR であったら、アスキーコード 13 の改行の制御文字が返されます。

App 以外にも、場所によって個別に利用できるオブジェクトがあります。以下、それを列挙していきます。

まず、「黒丸数字書式」「白丸数字書式」「黒四角数字書式」「アルファベット」など、自由置換設定を行うスクリプトで利用できるオブジェクトです。

ReplaceSign
自由置換設定の情報を 1 つ格納するためのオブジェクトです。自由置換設定を JavaScript で追加したい場合に利用します。

ReplaceTag
自由置換設定の置換後の情報を 1 つ格納するためのオブジェクトです。自由置換設定を JavaScript で追加したい場合に利用します。

次に、文字タグや段落タグから呼び出された場合に利用できるオブジェクトです。

startSign
開始記号の文字列です。実体は Java の String です。

endSign
終了記号の文字列です。実体は Java の String です。一行段落タグの場合は、undefined になります。

targetLine
現在の処理対象の行の文字列です。実体は Java の String です。この文字列の内容は、原稿ファイルの内

容そのままではなく、すでにある程度変換処理がなされたものになっています。

tagName
スタイル名です。実体は Java の String です。段落タグの場合は、先頭の「!!」が取り除かれています。

続いて、文字タグから呼び出された場合に利用できるオブジェクト（関数）です。これは、段落タグからでは呼び出せません。

searchTag()
引数に関数を取ります。
searchTag は、現在の処理対象の行から文字タグを見付け、処理対象の行を次の 3 つの部分に分割します。

- 行の先頭から開始タグの前まで
- 開始タグの後ろから終了タグの前まで
- 終了タグの後ろから行末まで

このそれぞれには、タグ自体の文字列は含まれません。
これらの分割した文字列が、引数として渡された関数の引数になります。引数側の関数は、変換結果の文字列を返さなければなりません。
searchTag は、現在の処理対象の行から、文字タグが見付からなくなるまで繰り返し呼ばれます。
searchTag は、おおよそ次のような書式で実装することができます。

```
searchTag(function(left, middle, right) {
  var result = ...;  // 変換処理を記述
  return result;     // 変換結果を返す
});
```

最後に、段落タグから呼び出された場合に利用できるオブジェクトです。これは文字タグからでは呼び出せません。

ParagraphTag

段落スタイルを格納するためのオブジェクトです。一時的に段落スタイルを適用したい場合などに使います。

■段落タグで実装する関数

段落タグから呼び出される JavaScript では、次の関数を実装しなければなりません。

whenStartSignMuches()
この関数に、開始タグが見つかったときの処理を記述します。出力したい文字列がある場合は、その文字列を return しなければなりません。

whenEndSignMuches()
この関数に、終了タグが見つかったときの処理を記述します。whenStartSignMuches 同様、出力したい文字列がある場合は、その文字列を return しなければなりません。終了タグがない場合は、記述しなくて

も構いません。