多様な携帯端末に適応可能なコンテンツ中間記述

田枝 覚[1]、渡部 聡彦[1]、中川 裕志[1]

携帯端末からのインターネット利用が急増している。コンテンツ提供者にとって、既にある PC 向けのコンテンツを携帯端末向けに変換できるれば経済的・時間的に大変メリットがある。本研究では、PC 向けに作成されたコンテンツを、それ自体に手を加えることなく変換するアーキテクチャを提案する。このアーキテクチャでは、元のコンテンツから２種類のテンプレートを作成する。またマークアップ言語変換のためのスクリプトを用意し、これらからコンテンツ変換を動的に行う。

An intermediate contents description to be applicable various kinds of PDA

Satoru TAEDA[1], Toshihiko WATANABE[1], Hiroshi NAKAGAWA[1]

Utilization of internet from PDA is increased rapidly. If there is a system that can converts the web documents for PC to the web documents for PDA, then there is a merit in the contents contributor. In this study, we propose the system that can do that kind of conversion. This system uses templates of two types. They are made on the basis of original web documents. And we prepare script files. then the system can convert web documents for PC to other PDA's document.

1　はじめに

1.1 背景

PC 向けの Web ページは 、インターネットの常時接続の普及、通信速度の急激な発展でこれまで以上に活用され、コンテンツを提供する側にとっても大きな市場になっている。一方でインターネット接続できる携帯端末も PC 市場の成長以上に拡大している。携帯端末へのコンテンツ提供はビジネス上、大きな位置を占めるようになった。しかし、携帯端末は、表示面積、受け取るデータ量の制約などから、PC 向けコンテンツとは別に製作されている。携帯端末向けにしか発信しないコンテンツであれば問題ないものの、例えばニュースなどのように、PC、携帯端末両者に用意しなければならないコンテンツであっても現状では別々に製作されており、開発コストにおいて不経済である。したがってコンテンツ変換の需要は非常に大きくなっている。

これまでにも、コンテンツ変換について IBM の Dharma [1]、長尾ら[2] がある。
IMB の Dharma は元コンテンツから携帯向けのコンテンツを Servlet で生成している。元コンテンツの内容分析を行い、適当な分割によってハイパーテキスト化もされる。
長尾らの研究では、元になるコンテンツ記述で工夫をし、コンテンツにタグを付け、端末の機能や利用者の要求に応じた表示を可能にしている。
製品としてでているものにフレックス・ファーム社の「x-Servlet」（クロスサーブレット）、ユーリードシステムズの「Web Chameleon」などがある。

1.2 変換における問題点

(1). マークアップ言語の機能の違い

ＰＣ向けのＨＴＭＬではテーブルタグがあるが、携帯端末、例えばｉモード対応ＨＴＭＬでは

[1] 東京大学 情報基盤センター
Infomation Technology Center, The University of Tokyo.

それがない。このとき、単にテーブルタグをはずし、列間はタブまたはスペース、行は改行に変換するといった、よく行われる変換をすると、携帯端末で見たとき、データ内容が理解し難くなる。

(2). 表示項目を指定すること

ＨＴＭＬで書かれているので、ページの構造から文書の構造を解析することが困難。そのため、表示させたいデータを絞ることができない。

例：

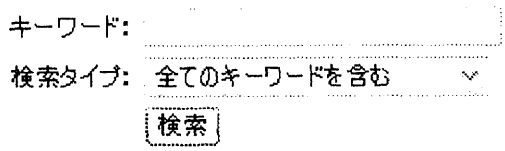

図 1　ＰＣ画面

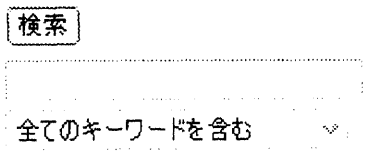

図 2　携帯端末画面

ＰＣ向けの検索サイトで「図 1　ＰＣ画面」のように表示されているものを携帯端末のような表示領域が狭いものでは、「図 2　携帯端末画面」のようにタイトルを省きたいとする。この場合、例えば“キーワード”という文字列がその隣のテキストボックスのタイトルであることを知っておかなければならない。しかしHTML文書では、そういった部品の意味的なつながりについての情報はない。

(3). 表示順序に関すること

例：

「図 1　ＰＣ画面」のように表示されているものを表示領域が狭い携帯端末において「図 2　携帯端末画面」のように、“検索”ボタンを上部に持ってきたいと考えたとする。このように、同じ機能であっても、ＰＣ画面と携帯端末ではその配置を変えることがある。システムにある判断をさせて表示位置を与えたとしても、それが意図した表示入れ替えを行わない場合、すみやかに対応できる方法を用意しておかなければならない。

(4). ページ分割

１ページのものを変換した結果、端末が一度に受け入れられるデータサイズを超えてしまい数ページに分割しなければならない場合がある。実際ｉモードでは１画面は５ｋバイト未満で、２ｋバイト以内を推奨している。

ページが分割された場合は、元のページで、

```
<a href="#place1" > Go to place1 </a>
...
<a name="place1">place1 here</a>
...
```

のように内部でジャンプする記述があり、これがファイル分割により別々になった場合、href の記述を書き換えなければならない。

1.3 変換システムの提案

以上のような問題点に対処するために本稿で提案するシステムの全体像は「図 3　システムの全体」のようになる。

ページを扱う場合、取得したいデータの位置を判定、また出力位置の指示、そしてマークアップ言語の変換法則自体をどのように扱うかが、まず大きな問題となる。渡部のシステムではこれらをプログラム内で操作している。[5]

この場合、コンテンツ構造の変更や変換法則の変更をシステムのプログラム変更で対応することになる。これは煩雑な作業で、またシステム内にバグをはらむ危険性が大きくなる。このことを回避するために、データ関連、変換法則をシステムから切り離すことにした。

システム構成はデータ取得部分、携帯端末画面へのデータ再配置部分、マークアップ言語変換部分となる。データ取得、データ再配置及びマークアップ言語変換は、その指示を外部ファイルでユーザが用意する。その外部ファイルはこのシステムで用意されている言語に基づいて記述される。システムはその記述の指示を翻訳・実行する仕組みを持つ。なお、図の“ページ取得”操作の前にある“正規化処理”とは、終了タグが省略できるタグにおいて、省略されているものには終了タグを補う処理である。したがって“ページ取得”時には、終了タグ省略可のものにも必ず終了タグがついた形でページが取得される。

2 変換用テンプレートファイル

Web ページを扱うときはまず、利用したいデ

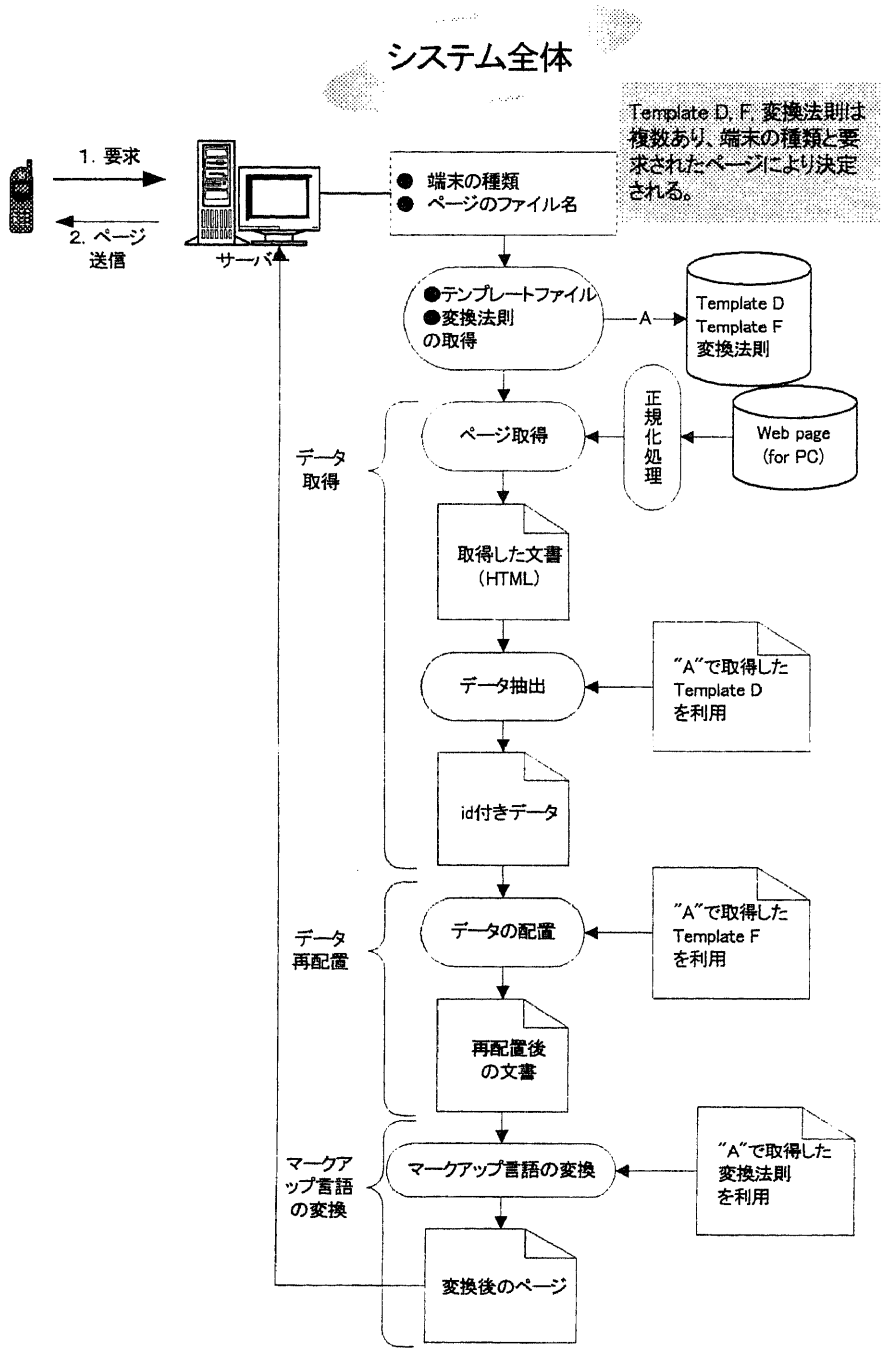

図 3　システムの全体

ータはどれで、どのように使いたいのかを特定しなければならない。そして先にも説明したように、そのような情報は外部に持ちたい。これを実現するために、そのページを元にテンプレートを作ることにした。このテンプレートで利用したいデータに id を振り、後の作業でのデータ操作を行いやすくする。

テンプレートは２種類用意する。一つは取得したいデータの指定のためのもので、もう一つはデータ配置用のテンプレートである。データ指定用のテンプレートを Template D、携帯端末画面上での配置を指定するテンプレートは Template F と呼ぶことにする。

Template D も Template F も HTML（XML でも同じだが）のコメントを利用して記述する。具体的には、システムはこれらテンプレートにおいて「<!--tuf:」という形のコメントを認識するのである。このコメント形式で記述された様々な指示を、システムの“テンプレート記述言語エンジン”が認識するのである。

2.1 Template D

Template D はあらかじめ取得した Web ページを元に作成する。利用したいデータにコメント形式で目印をつけることが目的である。コメント形式なので、一般に流通している HTML エディタで画面を見ながらこのコメントが挿入でき、また表示が乱れない利点がある。

Template D の例は下記のようになる。

Template D 例（「図 4　Template D」の①の部分）

```
<tr>
  <td nowrap><FONT SIZE="2">キーワード：
```

```
</FONT>
</td>
  <td><!-- tuf:data start id="inputKeyword"
--><input type="text" name="keyword" value=""
style="width:180px" size="12"><!-- tuf:data end
id="inputKeyword" -->
  </td>
</tr>
```

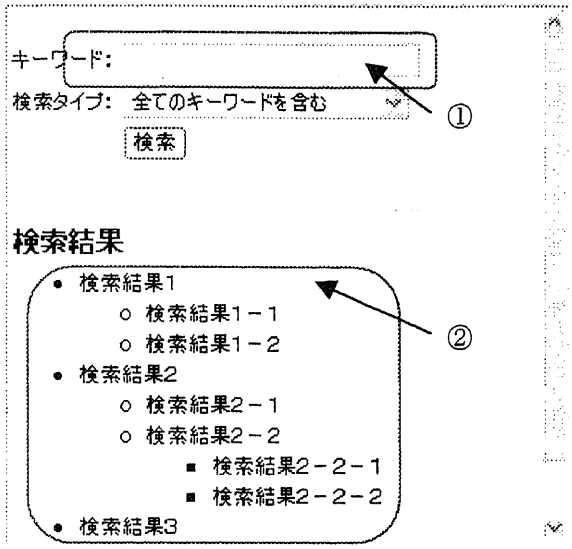

図 4 Template D

□ データ指定

● データ指定（基本）

これは、後に述べる繰り返し部分のないデータを指定したいときに用いる書式である。

書式：

<!-- tuf:data start id="*idname*" -->

取得したい部分

<!-- tuf:data end -->

意味：

これにより、start と end ではさまれた部分が取得部分と認識される。データは id の *idname* で区別される。

● データ指定（繰り返しデータの場合）

検索結果画面のように、決まった形式のデータが出力される場合を考える。（「図 4 Template D」の２の部分）

--- code ---

```
...
<ul>
  <li>検索結果１
    <ul>
      <li>検索結果１－１</li>
      <li>検索結果１－２</li>
    </ul>
  </li>
  <li>検索結果２
    <ul>
      <li>検索結果２－１</li>
      <li>検索結果２－２
        <ul>
          <li>検索結果２－２－１</li>
          <li>検索結果２－２－２</li>
        </ul>
      </li>
    </ul>
  </li>
  <li>検索結果３</li>
</ul>
  ...
```

この検索結果ページは、“検索結果１”, “検索結果２”・・・のように検索結果の件数分出力される。各検索結果は、内容により構造が変わっている。例えば“検索結果３”はそれのみだが、その他は付随データが出力されている。その出力も“検索結果１”と“検索結果２”のように構造の深さが変わっている。しかし、検索結果は外側の“
”と“”で囲まれた部分が繰り返されるデータとなっている。このようなデータを取得する場合の指定方法は次のようになる。

--- Template D での指定---

```
  ....
<ul>
<!-- tuf:repeat start id="search_result"
repeat_tag="li/li" -->
  <li>検索結果１
    <ul>
      <li>検索結果１－１</li>
      <li>検索結果１－２</li>
    </ul>
  </li>
<!-- tuf:repeat end id="search_result" -->
</ul>
  ....
```

このように、最初の定型パターンだけ取り出し、

<!-- tuf:repeat start id="*idname*"

repeat_tag="*tagnames*" -->
データ
<!-- tuf:repeat end id="*idname*" -->
により指定する。上記例では“li”タグで囲まれているので“li/li”のように書く。こうすることで、
の内側に
にて展開される検索結果パターンは取得できるようになる。なお、最初の定型パターンだが、「Template D での指定」では“検索結果１”の内部も使っているようにみえるが、システムは“
”と“”で囲まれた部分としてのみ認識するので、内側の“
...”は無視する。Template D を作る際、元のデータの最初の１件目をそのまま使っているということで、内部を残しているだけである。したがって検索結果パターンとして“検索結果３”の方を使ってかまわない。
書式をまとめると次のようになる。
書式：
<!-- tuf:repeat start id="*idname*"
repeat_tag="*tagnames*" -->
取得したい繰り返しデータ
<!-- tuf:repeat end id="*idname*" -->
意味：
start と end ではさまれた部分で、タグ“*tagname*”の繰り返しデータを取得する。

□　Template D　解析
解析はシステムで事前に行っておく。その内容は、ファイルの先頭から、最初の「<!-- tuf:」までのタグを列挙する。その後は順次、「<!-- tuf:」を含めタグを列挙する、という形をとる。
解析結果を同ファイルの「<body>」の次に記録する。
記録は
<!-- tuf:analyticResult start -->
<!-- tuf:"*tagname1*","*tagname2*",... -->
...
<!-- tuf:analyticResult end -->
のように記述される。
この記録では、タグ名は始まりだけでなく、終了タグも区別して記録される。

例：
---解析前----------
<BODY>
<!-- tuf:data start id="formStart" -->
...

<!-- tuf:repeat start id="search_result"
repeat_tag="li/li" -->

      検索結果１－１
      検索結果１－２

<!-- tuf:repeat end id="search_result" -->

...

--- Template D 解析後 ----------
<BODY>
<!-- tuf:**analyticResult start** -->
<!-- tuf:data start id="formStart" -->
...
<!-- tuf:"BODY", ... -->
...
<!-- tuf:"ul" -->
<!-- tuf:repeat start id="search_result"
repeat_tag="li/li" -->
<!-- tuf:repeat end id="search_result" -->
<!-- tuf:"/ul" -->
...
<!-- tuf:**analyticResult end** -->

□　システムの動作
「図３　システムの全体」における変換操作は次のように行われる。携帯端末から要求のあったページを取得し、このページを、「図３　システムの全体」の“A”で決定した Template D と比較する。比較は、上部からタグの出現順序をたどることで行われる。これを「Template D 解析後」の解析部分と比較することでデータ位置を特定し取得する。繰り返し指定の
<!-- tuf:repeat start ... -->
<!-- tuf:repeat end ... -->
では、その間のデータを取得し、「<!-- repeat end ... -->」の後から再びタグの比較を始める。

2.2 Template F
Template F は出力時のデータの位置関係を指定する役割を持つ。
Template F 作成例を見てみよう。「図５ Template F の①の部分」は次のようになる。
Template D と同じく HTML エディタで編集できる。

...

<!-- tuf:data start id="formStart" -->

<form name="mspress"

action="JPN_ViewMsPressList.asp" method="get">

<!-- tuf:data end id="formStart" -->

<!-- tuf:data start id="inputSubmit" -->

<INPUT type="submit" value=検索 name=Find >

<!-- tuf:data end id="inputSubmit" -->

<!-- tuf:data start id="inputKeyword" -->

<input type="text" name="keyword" value=""

style="width:180px" size="12">

<!-- tuf:data end id="inputKeyword" -->

...

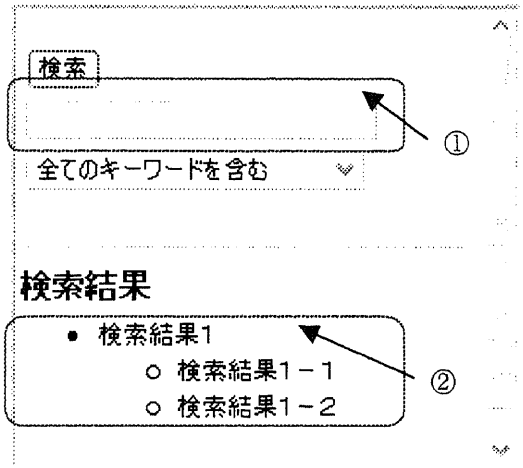

図 5　Template F

このように画面デザインを作成する。見て分かるように、記述は Template D で与えた id の部品を当てはめる形をとる。また上記のコードでは省略しているが、「図 5　Template F の②の部分」の繰り返しデータもやはり Template D の記述と同じものとなる。このように画面をデザインするが、実際の画面はこのテンプレートどおりにならない場合がある。例えば、ＨＤＭＬへの変換の際は、入力用の部品で画面が切り替わるというように、大きくレイアウトが変わる。ここでの作業は、表示内容の重み付けを、表示順序という視覚的作業で与えているのである。

Template F は作成後、システムに解析させておく。この解析作業は、

<!-- tuf:data start id="*idname*" -->

と

<!-- tuf:data end id="*idname*" -->

のペアとそれにはさまれた部分を

<!-- tuf:data temp id="*idname*" -->

に変換する作業となる。(Template F の解析結果)

Template F の解析結果

<!-- tuf:data temp id="formStart" -->

<!-- tuf:data temp id="inputKeyword" -->

<!-- tuf:data temp id="inputSubmit" -->

**
**

□　システムの動作

「図 3　システムの全体」における“データの再配置”は次のように行われる。

Template F を元にページを作成していくが、Template F 内の

<!-- tuf:data temp id="*idname*" -->

に、その idname に相当するデータを当てはめていく。

3　マークアップ言語の変換規則

「図 3　システムの全体」での“マークアップ言語の変換”について述べる。

マークアップ言語の変換規則を、このシステム独自のスクリプト言語で記述する。変換規則を外付けにすることで、ユーザ独自の変換規則に変更することが可能になる。マークアップ言語の変換規則は、マークアップ言語のタグ定義部分と変換規則の部分からなる。これらをユーザが用意する。システムは取得したページを、これらの変換規則に基づき、タグ変換し、ファイル分割が必要になる場合ファイル分割をし、それに伴う必要な調整を行う。なお、ファイル分割は最初はデータ内部で行い、必要な調整が終わった後物理的なファイル分割をし出力する。

3.1　マークアップ言語のタグ定義

タグ定義部分は、使用するマークアップ言語のタグ名とその属性を記述する。(例：スクリプト記述例（タグ定義))

スクリプト記述例（タグ定義）

□　標準 HTML

<language="html">

....

```
  <tag name="a">
    <attribute>href</attribute>
    <attribute>name</attribute>
    <attribute>accesskey</attribute>
  </tag>
....
</language>
```

□　c-html

```
<language="chtml">
....
  <tag name="a">
    <attribute>href</attribute>
    <attribute>name</attribute>
    <attribute>accesskey</attribute>
  </tag>
....
</language>
```

「<language="html">」で、言語名を宣言する。この場合、コンテンツ記述言語“html”という名称にしている。そして「<languate=..>」と「</language>」の中にタグ名を上げていくのである。

「<tag name="a">」では、タグの名前が“a”であることを示す。そして「<tag...>」と「</tag>」の間にそのタグの属性を列挙する。属性名は「<attribute>」と「</attribute>」の間に書く。

3.2 マークアップ言語変換規則

マークアップ言語変換規則の記述例を挙げる。

スクリプト記述例（タグ変換）

--- script 1 ---

```
na = ml("html","a");
ca = ml("chtml","a");

attr(ca, "href") = attr(na, "href");
attr(ca, "name") = attr(na, "name");
attr(ca, "accesskey") = f(attr(na, "accesskey") );
```

ここで、

ml(言語名,　タグ名)

でどの言語のタグの操作をしたいか指定する。ml("html",a)で"html"のタグ“a”と指定している。

attr(タグを表す変数, 属性名)

でそのタグの属性値を扱う。

上記例では attr(na, href) で、na の属性の一つ“href”を指定している。

f はユーザ定義関数である。このスクリプトはユーザが関数を定義することができる。

これでファイル全体でタグ“a”の入れ替えがおこなわれる。この「script 1」により、以下の様にコンテンツ記述言語“html”は“chtml”に変換される。

--- html 変換前 ---

```
<a href="test.html" name=test" accesskey="t">go to the test page.</a>
```

--- chtml 変換後 ---

```
<a href="test.html" name=test" accesskey="8">go to the test page.</a>
```

(accesskey はユーザ定義関数によって“t”は“8”に変えられている)

コンテンツ記述言語変換の際、いろいろな書き換えが必要になるが、ここではそのうちの一つ、内部リンクの書き換えについて述べる。

□　ページ分割に伴う内部リンクの書き換え

次のような内部リンクのあるものを考える。

--- chtml(file1.html) ---

```
<body>some text<br>
<a href="#name1">innner link(jump to name1)</a><br>
some text
<a name="name1">Here is name1.</a>
some text</body>
```

変換した結果、ファイル容量が大きくなり、file1.html と file1-2.html に分割することになった。

--- chtml(file1.html) ---

```
<body>some text<br>
<a href="file1-2.html#name1">innner link(jump to name1)</a><br></body>
```

--- chtml(file1-2.html) ---

```
<body>some text<br>
<a name="#name1">Here if name1.</a>
some text</body>
```

それにより、内部リンクが

<a href="#name1">

から

<a href="file1-2.html#name1">

に変わる。この操作を行うスクリプトの抜粋を挙げる。左側の数字は説明に用いるための行番号である。スクリプトに現れている変数 contents は body タグの内部を表していた content がファイル分割により配列 contents に分けられたものである。

--- script 2 ----------

```
01:for(i=0; i<dim(contents); i++)
02:{
03:    condition = cond(Left(attr(a, href), 1)='#');
04:    Set(contents[i], condition2, sa2[]);
05:    condition2 = cond(attr(a, name) !="");
06:    Set(contents[i], condition2, sa3[]);
07:}
08:for(i; i<dim(sa2); i++)
09:{
10:    for(j; j<dim(sa3); j++)
11:    {
12:        if(attr(sa2[i], href) == ('#' + attr(sa3[j[,
name]]
13:        {
14:            if(filename(sa2[i]) != filename(sa3[j])
15:            {
16:                attr(sa2[i],href) = filename(sa3[j]) + '#'
+ attr(sa3[j], name);
17:            }
18:        }
19:    }
20:}
```

---

01 行で分割されて格納された contents 全体を見る準備をしている。

03 行で条件を設定している。これは a タグが内部リンクかどうかの判定条件である。

<a href="#name1">

のように、a タグのアトリビュート "name" の値が "#" で始まっているもの、という条件を設定している。

04 行で条件にあったものを配列 sa2 に格納している。

05 行ではページ内で名前付けをしている a タグかどうかを判定している。

06 行で 05 行で設定した条件にあったものを配列 sa3 に格納している。

12 行では内部リンクの sa2[i]の行き先を sa3 内から探している。

14 行で内部リンク sa2[i]の行き先 sa3[j]が、sa2[i]と同じファイルではないかどうかを調べている。そしてファイル分割の結果違うファイルになっていれば、16 行目のようにリンク先を書き換えている。filename(sa2[i])というのは sa2[i]が属するファイル名ということ。システムは物理的なファイル分割を最終段階で行うので、このスクリプトを実行しているときの filename は論理的なファイル名である。

4 展望

これまで述べたような標準的なタグ変換規則はシステム提供時に用意しておいて、適宜その変換規則に書かれたスクリプトを書き換えるという運用を考えている。例えば html から chtml への変換で、テーブルを標準的には"td"をスペース、"tr" を改行に変換するスクリプトで提供しておく。これをリスト表現にしたい場合は、その部分を書き換える、という具合である。

今回のシステムでは、同一の Web ページの内部構造がダイナミックに変化する場合、テンプレートが機能しなくなるという弱点がある。この弱点は、ページ内のデータ位置をタグの出現パターンで認識するためである。ただしその変化が検索結果ページのようにパターンのあるものならこれまで見てきたように対応している。

5 謝辞

本研究は、東京大学・研究支援推進費、TAO 受託研究「モバイル環境における自然言語処理に関する研究」 による援助を受けて行われた。

6 参考文献

[1] IBM 東京基礎研 (2002), 多種端末向け Web アプリケーション構築技術: Dharma.
http://www.trl.ibm.com/projects/dharma/

[2] Nagao K, Shirai Y, Squire K, "Semantic annotation and transcoding: making Web content more accessible", IEEE Multimedia, April-June 2001 pp.69 –81(2001)

[3] 中川裕志, "携帯端末向けコンテンツ記述" 言語処理学会第 8 回年次大会ワークショップ「社会情報基盤のための言語・メディア処理」論文集,pp. 33-40, 2002

[4] 渡部聡彦, 武井純孝, 中川裕志: "携帯端末へのカタログ的コンテンツ表示のための問題点と対策", 第１５回人工知能学会全国大会予稿集, 1C1-04 (2001)

[5] 渡部聡彦, 中川裕志 (2002) "多種の表示デバイスへ適応可能なコンテンツ中間表現形式の提案", 情報処理学会研究会報告, 2002-FI-66-9, 2002.