テキスト処理'15 ＃4 – Webページの構造とテキスト抽出

久野　靖 *

2015.6.16

1　本日の内容

我々が分析対象のデータを見つけるきっかけとして、Webサイトを見ていて「この内容を分析したい」と思う、というのは比較的多いと思われます。そして、その内容がこれまで学んで来たような形ですぐ取れることもありますが、とくに文章内のデータを抽出したい場合には、たいていはWebページの中から必要な部分だけを取り出して来る必要があります。今回はそのような話題を取り上げましょう。

2　Webページの構造とHTML

2.1　HTMLの形式

Webページは一般に、HTMLと呼ばれるマークアップ言語で記述されています。その構造は大枠では次のような形になっています。先頭の行はDOCTYPE宣言といい、以下がHTMLであることを表します(HTMLのバージョンが色々あるので、それを指定する書き方になっていることもあります)。

```
      <!DOCTYPE html>
+-----<html>
| +---<head>
| |   <meta charset="euc-jp">
| |   <title>ページタイトル</title>
| | +-<style type="text/css">
| | | スタイル記述…
| | +-</style>
| +---</head>
| +---<body>
| |   ページ本体…
| +---</body>
+-----</html>
```

そして2行目以降がHTML記述です。見て分かるように、「<名前>～</名前>」という形のものが多数、入れ子構造になっています。この1つずつを「要素(element)」と呼び、要素の開始部分の「<名前>」を開始タグ、終了部分の「</名前>」を終了タグと言います。開始タグには属性(「属性名=値」という形の指定)がついている場合もあります。

HTMLでは終了タグが省略できる場合がありますが、その場合でも論理的には終了タグが存在して、そこまでが要素の範囲になります。また、終了タグを書かない開始タグだけのものもありますが、これは論理的には「<名前></名前>」のように開始・終了タグがくっついたものです(範囲が空っぽなので「空要素」と呼びます)。

*経営システム科学専攻

2.2 主要な HTML 要素

では、具体的に先に出て来た要素の意味を書きます。

- `<html>〜</html>` — この範囲が HTML 文書全体を表す。
- `<head>〜</head>` — ヘッダ (この文書が何であるかという情報を含める部分) の範囲を表す。
- `<meta charset="文字集合">` — 文書の文字コードを表す (空要素)。
- `<title>〜</title>` — 文書のタイトルを表す。
- `<style>〜</style>` — この文書に使うスタイル記述を記す範囲を表す。`type="text/css"` というのはスタイル記述の形式が CSS (cascading style sheet) であることを表す。
- `<body>〜</body>` — ページ内容の範囲を表す。

つまりここまでは、ページ内容に入れる要素は何も出て来ていません。それについても主要なものを挙げておきます。

- `<h1>〜</h1>` — レベル 1 の見出し (大見出し) を表す。見出しのレベルは 1〜6 までの 6 レベルがある。
- `<div>〜</div>` — ページ内のまとまりを表す。
- `<p>〜</p>` — 1 つの段落を表す。
- `<ul>〜</ul>`、`<ol>〜</ol>` — 番号なし/番号つきの箇条書きを表す。
- `<li>〜</li>` — UL 要素や OL 要素の 1 項目を表す。
- `<dl>〜</dl>` — キーワード型の箇条書きを表す。
- `<dt>〜</dt>`、`<dd>〜</dd>` — DL 要素の中に含めるキーワード部分と説明本体部分を表す。

LI 要素は UL か OL の中、DT 要素や DD 要素は DL の中になければいけない、などの規則がいろいろありますが、こまかいことは省略しています。

上の内容はだいたい「段落レベル」の内容 (ブロック要素) でしたが、段落等の中に改行があったり強調部分があったりする場合、それも要素となります。

- `<a href="`URL`">〜</a>` — リンク。要素の内側の内容がリンクテキストとなり、リンク選択時の行き先は `href` 属性で指定する。
- `<br>` — 改行 (空要素)。
- `<img src="ファイル等">` — 埋め込み画像 (空要素)。
- `<em>〜</em>`、`<strong>〜</strong>`、`<b>〜</b>`、`<i>〜</i>`、`<tt>〜</tt>`、`<span>〜</span>` — 強調、目立つ文字、ボールド、イタリック、タイプライタ書体、一般の文字範囲を表す (SPAN 要素はその見え方は CSS で別途指定するのが普通)。

そのほか、データシートのようなものではしばしば「表」が使われます。

- `<table>〜</table>` — 表をあらわす。
- `<tbody>〜</tbody>` — 表本体を表す。TBODY 要素のタグは (開始・終了とも) 省略されることが多いですが、論理的には TBODY 要素が必ずあってその中に行が入ることに注意。
- `<tr>〜</tr>` — 表の 1 つの行を表す。TBODY の中に入る。
- `<th>〜</th>`、`<td>〜</td>` — 見出しセルと一般のセルを表す。TR 要素の中に入る。

2.3 ページの記述例と構造

ではもう少し具体的なページの例を見ていただきましょう。

```
      <!DOCTYPE html>
+-----<html>
| +---<head>
| |   <meta charset="euc-jp">
| |   <title>Sample Page</title>
| | +-<style type="text/css">
| | | h1 { border: ridge blue 3mm }
| | | ul { padding: 1cm; background-color: rgb(200,255,200) }
| | +-</style>
| +---</head>
| +---<body>
| |   <h1>簡単なページの例</h1>
| | +-<ul>
| | | <li>HTML では文書の構造を記述します。
| | | <li>CSS では文書の表現方法 (見え方) を記述します。
| | | <li>別の文書にリンクを貼れることが HTML の特徴です。
| | |     たとえば、<a href="http://www.nhk.or.jp">NHK<a>とかでも。
| | +-</ul>
| +---</body>
+-----</html>
```

この HTML を Firefox ブラウザで表示したところを図 2 に示します (画面の下の部分については後で説明します)。さて、HTML を扱うときには、その「木構造」を把握しておくことが重要です。たとえば先の HTML の構造を図に描いたものを図 1 に示します。

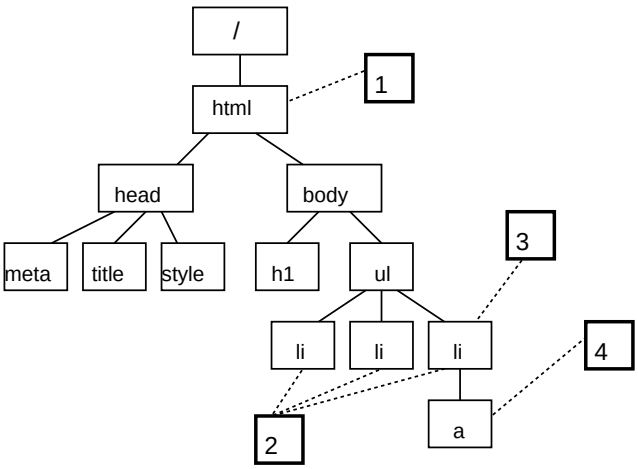

図 1: 例題の HTML の構造

この図では HTML の要素だけを箱で示し、そのつながり関係を線で示しています。HTML は入れ子構造なので、図は木構造になります。本当はこのほかに属性やテキストもあるのですが、それらは省略してあります。このようなものがすぐ描けるようになって頂きたいわけです。

そして、最近のブラウザには開発用にこの構造を表示する機能がついているものがあります。Firefox もその 1 つです。具体的には「Firefox メニュー → Developer → Inspector」を選択すると、窓の下

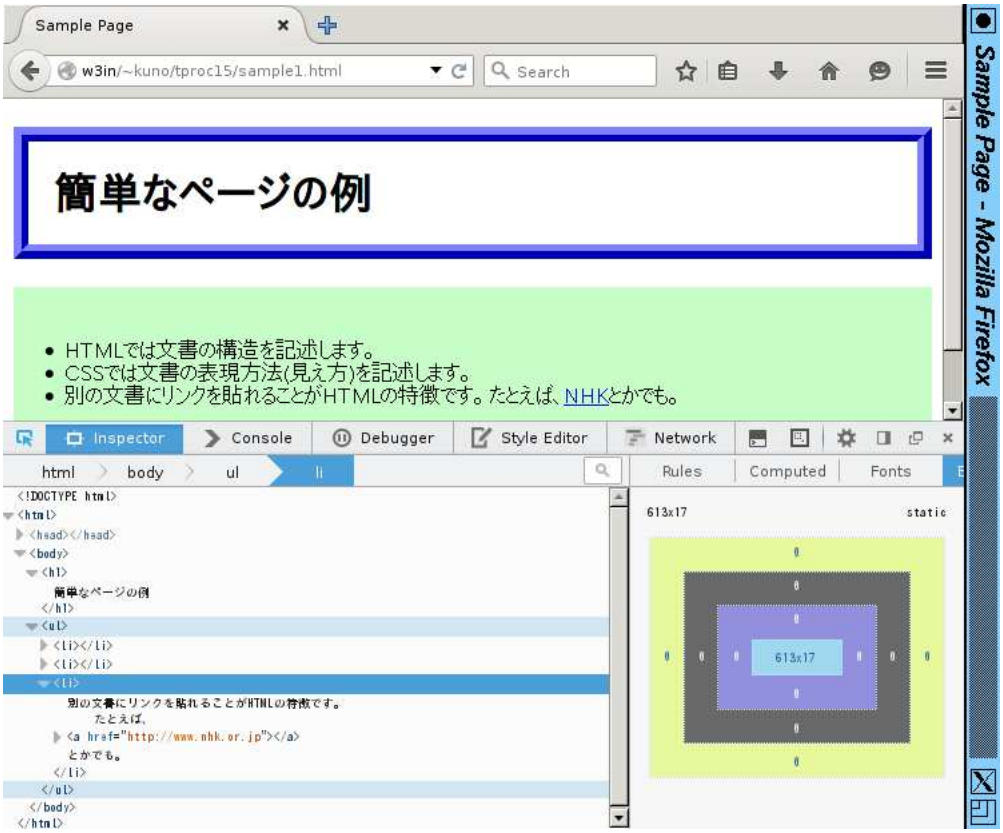

図 2: Firefox での Web ページ表示+構造表示

半分にインスペクタが表示されます。そのうちさらに半分はツリー表示になっていて、必要な箇所の▲を開いていくことで全部の構造を調べることができます (図 2 下)。

3 Web スクレイピング

3.1 Nokogiri と XPath

さて、前述のような HTML の構造から分析に必要な部分を取り出して来るにはどうしたらいいでしょうか。HTML ファイルを取り寄せてくることができたとしても、その中からこれまで学んだパターンマッチングなどを使って必要な箇所を取り出して来るのはひどく大変そうですが…

実は Ruby ユーザの間ではそのためのフレームーワークとして Nokogiri というライブラリが広く使われています。これはどういうものかというと、HTML を読み込んで来た後、先に説明したような木構造を作り出し、その上で必要な箇所を抽出するものです。たとえば HTML の表としてデータが表示されているのなら、その表の構造を用いて「3 番目のカラムと 4 番目のカラムをずらっと取り出す」とか言えばいいわけです。

上で「言えばいい」と書きましたが、そのためには「言い方」が分からないとできませんね。実は Web 標準の中に XPath と呼ばれる記法を定めたものがあり、この記法を使うことで HTML(やその後継である XML) の中から「この部分を取り出したい」ということが簡潔に指定できます。そして、Nokogiri は実は、XPath を受け取って HTML の中から指定された箇所を取り出して来る、というのが主な機能なわけです (他にも色々細かいことはありますが、今回は時間もないのでほとんど扱いません)。

そういうわけでこの先、XPath を学ぶのですが、「おあずけ」があまり長いとやる気が出ないので、まずは先のサンプル HTML から LI 要素を取り出すというサンプルを見てしまいましょう。

```
require 'open-uri'
require 'nokogiri'

$url = 'http://www2.gssm.otsuka.tsukuba.ac.jp/staff/kuno/tproc15/sample1.html'
$proxy = { :proxy => 'http://w3proxy:8080/' }
$html = open($url, $proxy) do |f| f.read end
$page = Nokogiri::HTML.parse($html, nil, 'euc-jp')

def extract
  items = $page.search('//li')
  items.each do |s| puts("[#{s.to_s}]") end
end
extrac
```

この例題の中身を簡単に説明します。まず冒頭の定型部分です。

- 2 つの require は URL からの読み込みと Nokogiri をライブラリとして使うという宣言です。
- 変数$url は取り寄せるページの URL です。実際には HTML を手元に取り寄せて保管してから分析することがおすすめですが、今回は簡単のため取り寄せつつ解析しています。
- 変数$proxy はここの環境内で外部のページを取るにはプロキシ (中継サーバ) を通る必要があるために指定しています。外部に直接つながったところでやる場合は不要です (その場合、次の行の open の引数は$url だけでよいです)。
- 変数$html には、ページを開いて HTML を全部読み込んだ文字列が入ります。
- 変数$page には、Nokogiri で HTML を解析した木構造が入ります。ここで、ページの文字コードは適宜調べて「iso-2022-jp」「euc-jp」「shift-jis」「utf-8」などから適切なものを指定してください。

ここまでで HTML の木構造が用意できましたが、これらはどのページでもだいたい同じです (URL は文字コードは適宜修正するでしょうけれど)。この後のメソッド extract が実際の処理をする関数部分で、最後にそれを呼び出しています。

extract の中では、まず XPath を使って必要な部分を指定します。XPath の説明がまだですが、ここでは簡単に「任意の場所にある LI 要素」を指定しました (「//li」の意味)。そうすると配列のようなオブジェクトが返されるので、その中身を each で順番に取り出し、文字列として出力します。実際にやると次のようになります。

```
% ruby noko1.rb
[<li>HTML では文書の構造を記述します。
</li>
]
[<li>CSS では文書の表現方法 (見え方) を記述します。
</li>
]
[<li>別の文書にリンクを貼れることが HTML の特徴です。
    たとえば、<a href="http://www.nhk.or.jp">NHK</a>とかでも。
</li>]
```

HTML の中の該当部分が取り出されることが分かります。また、ファイルには LI 要素の開始タグしか入っていないのですが、取り出した時は終了タグもつけられています。なお、ここでは要素オブジェクトを.to_s で文字列にしていましたが、この方法だとタグも一緒に含まれた形になります。テキスト部分だけで取り出したい場合は代わりに.text で取り出してください。

3.2 XPath の記法

XPath は WWW コンソーシアム (W3C) が定めている標準の 1 つで、HTML や XML 文書の中から特定要素 (群) を指定する記法を定めるものです。現在の最新版は XPath 3.0 ですが、ここでは最初の版 XPath 1.0 から含まれている内容で、なおかつ今回の内容に関係のある最小限の部分を扱います。また、検索式の記法に省略記法と非省略記法があるのですが、簡単な方がいいので省略記法だけを扱います。

まずいくつか具体例を示し、続いて一般の説明に進みます。先に出て来たように「`//`」で始まると「ドキュメント先頭からの任意位置にある…」になります。

- `//li` — ドキュメントの任意の位置にある LI 要素
- `//ul/li` — ドキュメントの任意の位置にある UL 要素の直下の LI 要素
- `//td[3]` — ドキュメントの任意の位置にある「親要素の中で 3 番目の位置にある」TD 要素
- `//div[@class]` — ドキュメントの任意の位置にある DIV 要素で、ただし class 属性を持っているもの

基本的な XPath の (省略記法の) 式は次のようなパスの形をしています。

`//` ステップ `/` ステップ `/` … `/` ステップ
`.` `/` ステップ `/` … `/` ステップ

「`//`」で始まる場合は文書のルート以降の任意の位置で以下のステップを探すという意味になります。一方、「`.`」は現在位置から探すという意味になります (木の一部を取り出してそこから探す時に使います)。また、「`/`」はその手前までに指定したノードのすぐ下に次のステップで指定するノードがあることを意味しますが、これを「`//`」にしてもよく、その場合は間にいくつでも中間のノードがはさまっていても構いません。

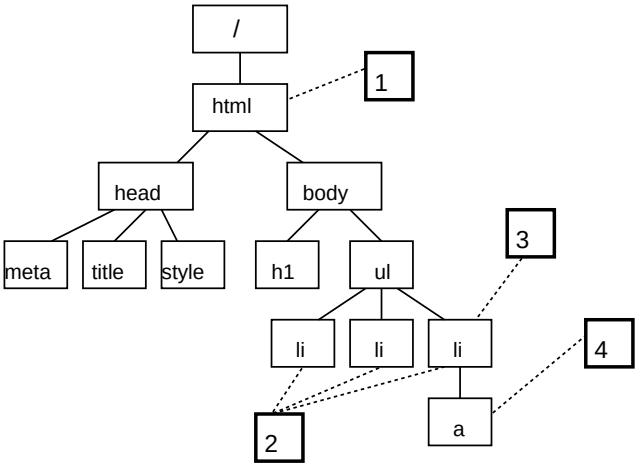

図 3: 例題の HTML の構造 (再掲)

図 3 に先の HTML の構造図を再掲します。これを例に body の下のいくつかのノードを XPath で指定してみましょう。

1 h1 要素 — これは簡単で、「`//h1`」だけでよいですね。1 つしかないので。

2 3 つの li 要素 — これも「`//li`」で大丈夫です。なお、この後ろにも ul 要素がいくつかあって、その最初のものの 3 つの li、とかだと「`//ul[1]/li`」のようにして「どの li」を特定する必要があります。

3 3番目のli要素 — これも「//li[3]」で大丈夫です。後ろにもっとulがある場合は上記と同様にしてulを特定します。

4 a要素 — 「//a」で全部のa要素が取れますが、3番目のli要素の中のa、ということなら「//li[3]/a」となります。

ステップの基本的な形としては次のものがあります。

- 要素名 — 指定した種類の要素を探すことを意味します。代わりに「*」(任意の種類) を指定することもできます。
- 要素名 [条件] — 上記に加えて条件が指定できます。詳細は下で説明します。
- text() — こう指定すると、テキスト部分を取り出すことになります。

条件としては次のものがあります。

- 1、2などの整数でその番号番目の子供ノードを意味します。last()という関数で最後の番号が取れます。計算式が使えるので「last()-1」などとも書けます。
- 一般の条件式も書けます。たとえばそれぞれのノードの番号はposition()という関数で取れるので、「position() <= 3」などとすれば「1〜3番目だけ取る」ことができます。また、「@属性名」「@属性名 = "文字列"」により、どういう属性を持っているか、属性の値が何かを指定することもできます。また、属性でなく「直下にある要素Xの値が指定した文字列に等しい」(X = "文字列"と書けますし、どの要素Xかを番号や条件で指定することもできます。

演習 26 次のような月の一覧データがあるものとします。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="euc-jp">
<title>Sample Page</title>
<style type="text/css">
h1 { border: ridge blue 3mm; padding: 5mm }
table { border-collapse: collapse }
th, td { padding: 2mm; border: solid black 2px }
td.days { background-color: rgb(200,200,255) }
td.old { background-color: rgb(255,200,200) }
</style>
</head>
<body>
<h1>月の一覧</h1>
<table>
<tr><td>番号<td>英語名<td>日本語<td class="old">旧月名<td class="days">日数
<tr><td>1<td>January<td>一月<td class="old">睦月<td class="days">31
<tr><td>2<td>February<td>二月<td class="old">如月<td class="days">28
<tr><td>3<td>March<td>三月<td class="old">弥生<td class="days">31
<tr><td>4<td>April<td>四月<td class="old">卯月<td class="days">30
<tr><td>5<td>May<td>五月<td class="old">皐月<td class="days">31
<tr><td>6<td>June<td>六月<td class="old">水無月<td class="days">30
<tr><td>7<td>July<td>七月<td class="old">文月<td class="days">31
```

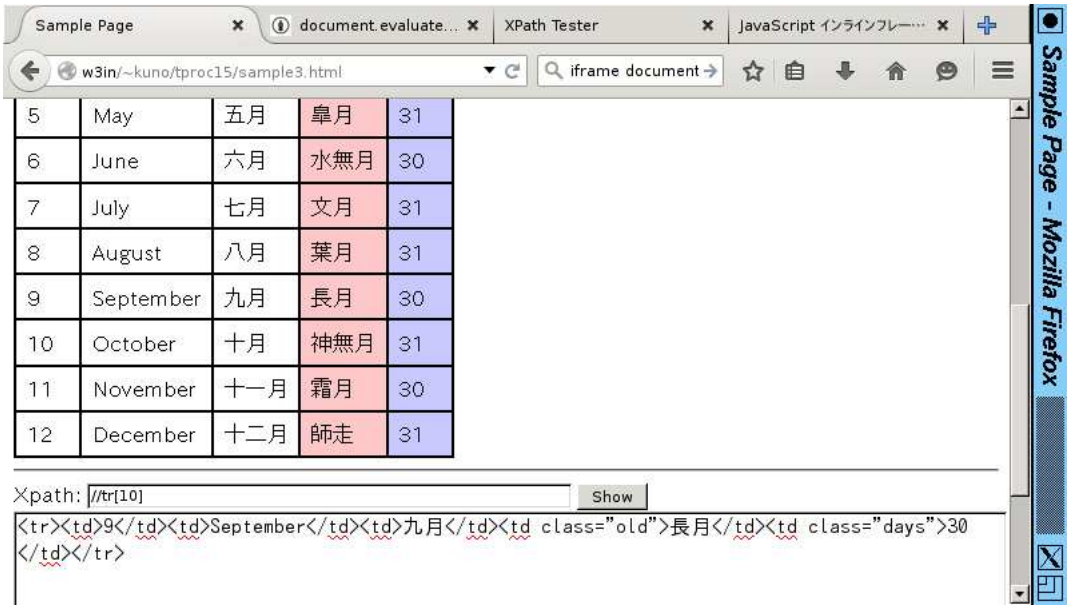

図 4: 月一覧に XPath の練習機能を組み込んだようす

```
<tr><td>8<td>August<td>八月<td class="old">葉月<td class="days">31
<tr><td>9<td>September<td>九月<td class="old">長月<td class="days">30
<tr><td>10<td>October<td>十月<td class="old">神無月<td class="days">31
<tr><td>11<td>November<td>十一月<td class="old">霜月<td class="days">30
<tr><td>12<td>December<td>十二月<td class="old">師走<td class="days">31
</table>
</body>
</html>
```

これについて、木構造を紙に描きなさい (繰り返しのところは省略してよい)。また、この HTML に対して XPath 式を打ち込むと当該要素が表示される機能をくっつけたページを用意したので、それを使ってさまざまな要素を表示してみなさい。

演習 27 この HTML を読み込んで、次のテキストデータを出力する Ruby プログラムを書きなさい。例題から再利用できるところは再利用してよい。

a. 月番号の一覧★
b. 月番号と旧月名の一覧★
c. 月番号とその月の日数の一覧
d. 3 月に対応する全データ
e. 表の全データを CSV 形式で出力

3.3 フレーム機能のあるページ

HTML にはフレーム機能と呼ばれる、別の HTML を取り込んで埋め込む機能があります。今では新たに作られることはあまりないが、古い設計のページには見られるので、説明しておきます。

- `<iframe src="`*URL*`">～</iframe>` — 埋め込みフレーム (四角い領域を区切り、その中に別ページを表示するもの) を表す。

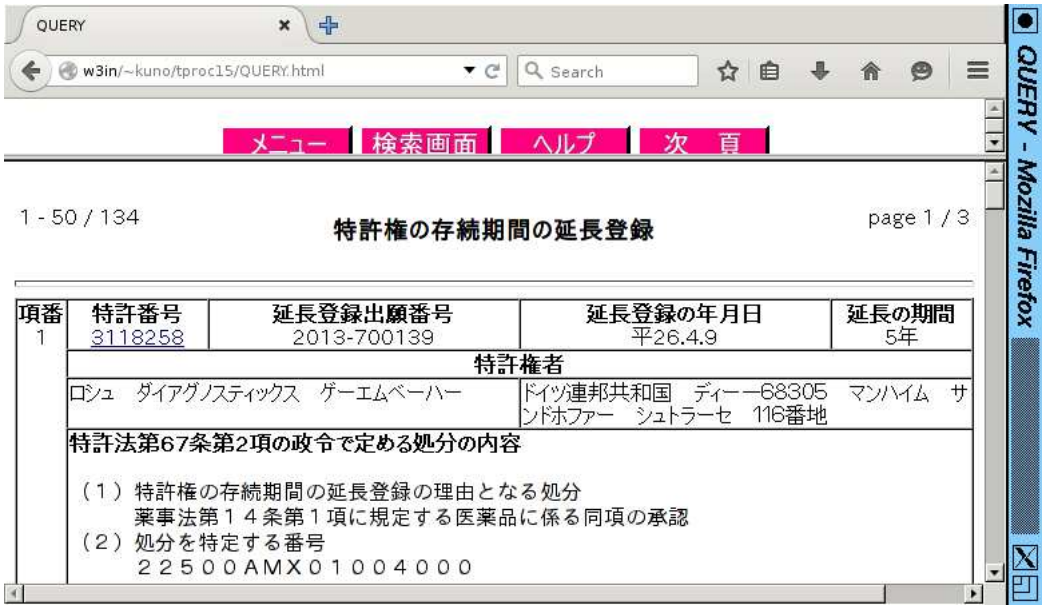

図 5: 「特許権の存続期間の延長登録」ページ

- `<frameset cols="配置">〜</frameset>` — ページ全体を複数の別ページの集まりとして作成する。FRAMESET 要素の内側には FRAME 要素だけが入る。`cols` 属性がある場合は横にいくつかに分割する。代わりに `rows` 属性を指定すると縦にいくつかに分割する。
- `<frame src="`*URL*`" name="名前">` — FRAMESET 要素の中に入れる個々のページを表す。

Web スクレーピングの場合は、フレームを直接扱うことはなく、その中にある個々のページ (`src` 属性で指定されるもの) を扱えばよいのでとくに難しいことはありませんが、このようなものがあることは知っておいた方がよいでしょう。

演習 28 立本先生からいただいた例題データとして「特許権の存続期間の延長登録」ページがあります (図 5)。このページ (FRAMESET を使っている) をまず観察しなさい。続いて、次の情報を取り出してみなさい。

a. 項番と特許権者名の一覧★
b. 項番と特許番号、延長年月日、延長期間の一覧
c. 項番と特許番号と処分の内容の一覧
d. その他、自分ならこの情報を抽出して分析すると思うものを CSV で出力。

演習 29 一般の情報サービスページから情報を抽出してみなさい。たとえば次のようなものが考えられる (詳細は適宜変更してよい)。

a. kakaku.com サイトから、特定メーカーの PC の売れ筋ベスト 40 の品名、型番、最安価格
b. 2 ちゃんねるから、特定スレッドの全記事本文
c. 複数の人の名前を与えて、Wikipedia(日本語でも英語でも) からその人の情報 (たぶんページの先頭部分にあるところ) を取って来る
d. その他、自分で興味を持って集めてみたい Web データ

4 第 4 回レポート課題

2 つ以上の演習のプログラムを選んで「プログラムを実際に動かす」課題をおこない、レポートとして提出しなさい。レポートではプログラムごとに次の内容を含むこと。

- 学籍番号、氏名、提出日付 (これらは最初に 1 回でよい)
- 問題の再掲
- 作成したプログラムとその説明
- 実行結果と考察

とくに考察において、「ロジックのどこが難しいか、どのように工夫したか」をきちんと書くこと。レポート課題は次回授業開始時までに、gssm.k-kiso に投稿すること。ただし★がついている問題 (易しい問題) を含む場合はは本日中。