ななちゃんのIT教室

クリじい探検隊 ドムドム島探検の巻

by nara.yasuhiro@gmail.com

ななちゃんが
DOM の仕組み解明に挑戦するという お話

第 0.6 版　2017 年 5 月 24 日

フリー素材
http://freeillustration.net

いらすとやフリー素材
http://www.irasutoya.com/

もくじ

第1回　探検に出発

クリ：クリじいじゃ。よろしくな。

なな：BSジャパンの「空から日本を見てみよう」に出てくる「くもじい」 のパロディ？

クリ：お前だって、朝日新聞の「ののちゃんのDO科学」のパロディだろうが。

なな：こんどは何を探検するの？

クリ：ドムドム島じゃ。

なな：船にのって出かけるの？

クリ：ブラウザの「コンソール」でも良いんじゃが、ブラウザごとに操作が異なる可能性があるので、ワシは、下記のような簡単なプログラムを使うのが好みなんじゃ。「1+2」と書いてある枠に JavaScript の命令を込んで Enter キーを入力すると、その命令を実行した上で、その下の大きな枠に、その命令実行結果の「評価値」が表示されるんじゃ。

```
<input type=text id=in value="1+2" autofocus size=55><br>
<textarea cols=50 rows=14 id=out></textarea><br>
テスト入力:
<input type=text id=testio value="Hello" size=30><br>
<script>
var inp   = document.getElementById("in");
var outp = document.getElementById("out");
var testp = document.getElementById("testio");
var geval = eval;

document.onkeydown = function (e) {
    if (e.keyCode != 13) return;
    outp.value += "% " + inp.value + "\n";
    try         { outp.value += eval(inp.value) + "\n"; }
    catch(e) { outp.value += "! " + e + "\n"; }
    outp.scrollTop = outp.scrollHeight;
    inp.value = ""; //inp.focus();
}
</script>
```

調査対象となるHTML

なな：ひょうかち？

クリじい：たとえば、「1+2」と入力すれば、「3」が評価値だし、変数 x が 7 という数を記憶している時に「x」と入力すれば、「7」が評価値というわけだ。「x = 5」は、x に 5 を記憶させるという命令（操作）だが、その評価値は 5 になる。「var y = 3」は、y に 3 を記憶するが、評価値は undefined になるので要注意じゃ。

なな：りょうーかい。

第2回　探検の第一歩：名前を確かめる

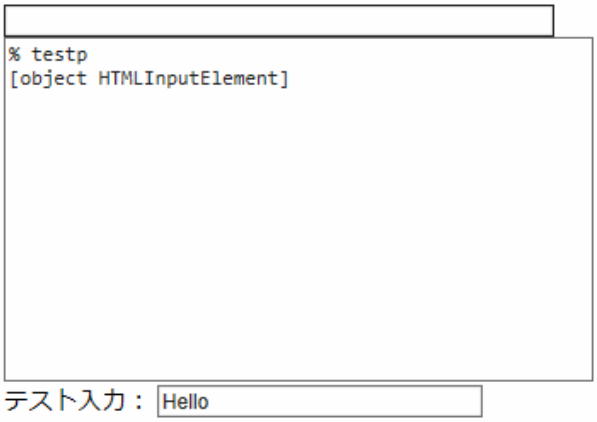
なな：何からはじめるの？

クリじい：新しい島についたら、まず、現地の人と知り合って、名前を聞くんじゃ。言葉が通じるか分からないから、まず、自分を指さして「クリじい」、自分の船員仲間を指さして「ロベルト」、そして相手を指さして首をかしげる。すると、「ドメハメハ」とか言い返してきて、相手の名前が分かるんじゃ。

なな：今回はどうするの？

クリじい：上の枠に「testp」と入力して、「document.getElementById("test")」が何を返してきているのかを調べるんじゃ。ななちゃん、やってみてくれんか。

なな：は～～い。こんな画面になりました。



クリじい：ごくろう。入力枠に入力した「testp」が消えて、下の枠に何か出てきたろう。「% testp」は、「testp」と入力したぞ、という意味。「[object HTMLInputElement]」は、変数 testp が記憶している値は、HTMLInputElement というオブジェクトだということじゃ。 次に、「testp.outerHTML」、それから「testp.value」の値をしらべてくれんか？

なな：は～～い！　こんな画面になったわ。

```
% testp
[object HTMLInputElement]
% testp.outerHTML
<input id="testio" type="text" size="30" value="Hello">
% testp.value
Hello
```

クリじい：outerHTML は、「<input id="testio" type="text" size="30" value="Hello">」という、タグの内容、value は、input 枠に「Hello」という文字列が書かれていた、という意味なんじゃ。

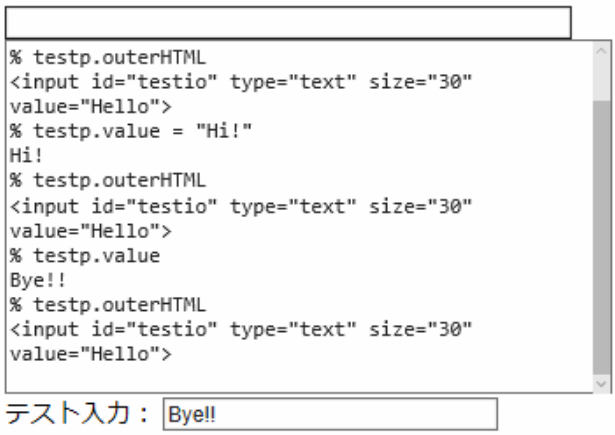
なな：HTMLInputElement という「オブジェクト」が、そのような情報を持っていたということね。

クリじい：そうじゃ。正確に言えば、「HTMLInputElement」は、「日本人」みたいな一般名詞で、「id=”test”」の「test」が、「なな」みたいな固有名詞ということになる。変数 testp が記憶しているオブジェクトは、HTML の input 枠と JavaScript が、通信というか、情報のやりとりをする窓口のような働きをしておるんじゃな。

第3回 クリじい、変です

なな：くりじい、この結果を見てください。何か変です！

```
% testp
[object HTMLInputElement]
% testp.outerHTML
<input id="testio" type="text" size="30" value="Hello">
% testp.value = "Hi!"
Hi!
% testp.outerHTML
<input id="testio" type="text" size="30" value="Hello">
% testp.value
Bye!!
% testp.outerHTML
<input id="testio" type="text" size="30" value="Hello">
```



クリじい：ななちゃん、興奮して、どうしたんじゃ？

なな：「testp.value = "Hi!"」という JavaScript 命令で、input 欄に「Hi!」と表示してから、次に、input 欄に「Bye!!」と書き込んでから、それを「testp.value」の値として JavaScript で読みだしたんだけど、outerHTML で調べると、「value="Hello"」のままなんです！

クリじい：するどいのお。HTMLInputElement というオブジェクトは、JavaScript の本物のオブジェクトではないんじゃ。JavaScript から読み書きができるように、**オブジェクトらしく見えるようにシミュレーション**しているものなんじゃ。「testp.value = "Hi!"」は、オブジェクトの value プロパティの値を書き換えるのではなく、input 枠に「Hi!」と表示するんじゃ。「testp.value」は、オブジェクトの value プロパティの値を調べるのではなく。input 枠に入力されている文字列を調べるんじゃな。

なな：オブジェクトらしく見えるようにすることで、JavaScript で操作しやすくしているのね。

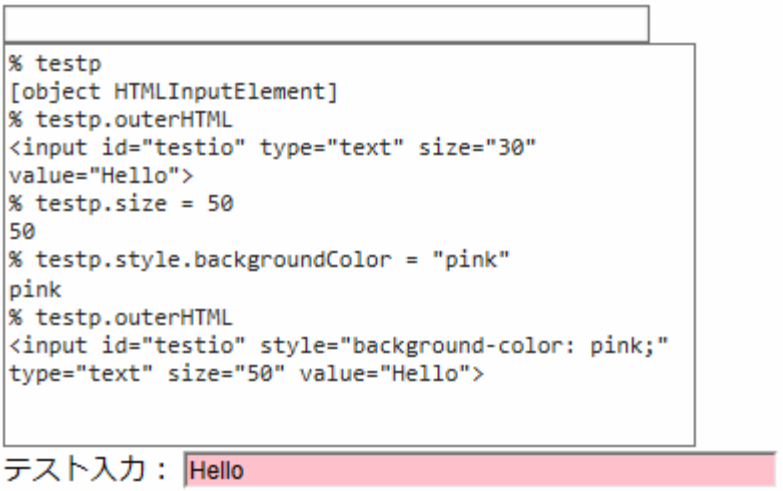
第4回　探検の第二歩：頼み事をする

なな：次に何をするの？

クリじい：新しい島に上陸して、自己紹介の次にやることは、食料や飲料水をもらうことなんじゃ。頼み事をすることで、相手の外来者に対する姿勢を確認したり、こちらが攻撃したり略奪する意図が無いことを伝えたりするんじゃ。ななちゃん、「testp.size = 50」、「testp.style.backgroundColor = "pink"」と入力してみてくらるかのお。

なな：は～～い。

```
% testp
[object HTMLInputElement]
% testp.outerHTML
<input id="testio" type="text" size="30" value="Hello">
% testp.size = 50
50
% testp.style.backgroundColor = "pink"
pink
% testp.outerHTML
<input id="testio" style="background-color: pink;" type="text" size="50"
value="Hello">
```



入力枠の横幅が長くなって、背景がピンクに変わったわ！　outerHTML の結果も変化している。

クリじい：入力枠の内容だけでなく、size などのプロパティの値、background-color などの スタイルを書き換えると、それが瞬時に、表示に反映されるんじゃな。

なな：HTML(CSS) の background-color を書き換えるのに JavaScript では backgroundColor と指定？

クリじい：JavaScript で「 - 」は引き算を意味するので、background-color は「background 引く color」という意味になってしまう。 それでは困るので、書き方を変えることになっているんじゃ。

なな：りょうかい！

先生：testp.setAttribute(“size”,”50) とか、testp.setAttribute(“style”,”color:green;background-color:pink;”) というのも使えるのよ。

なな、クリじい：先生、ドムドム島に来てたの！？

<タグの属性操作>

・element.getAttribute()	タグの属性値を取得
・element.setAttribute()	タグの属性値書き込み
・element.removeAttribute()	タグの属性削除
・document.createAttribute()	属性ノードを生成
・element.attributes	タグの属性を全て取得
・element.hasAttribute()	タグの属性に指定の属性があるかどうか調べる

element.attribute の使い方

```
var attrs=testp.attributes;
var text = "";
for(var i=attrs.length-1; i>=0; i--) {
   text += attrs[i].name + "->"
                + attrs[i].value + ":";
}
testp.value=text;
```

```
value->Hello:size->50:type->text:id->testio:
```

第5回 先祖や、酋長のことを聞く

なな：次にやることは？

クリじい：最初に会った現地の人の、親とか、酋長がいないか確認するんじゃ。若者と交渉成立しても、あとで長老から反対されると交渉やり直しになってしまうからのお。「parentNode」を繰り返し使って、最年長者を調べてくれんか？

なな：は〜い。testp.parentNode.parentNode.parentNode.parentNode.parentNode なんてなると大変なので、途中結果を cwn という変数に記憶するようにしました。 current working node ということで。

```
% var cwn
undefined
% cwn = testp.parentNode
[object HTMLBodyElement]
% cwn = cwn.parentNode
[object HTMLHtmlElement]
% cwn = cwn.parentNode
[object HTMLDocument]
% cwn = cwd.parentNode
null
% document
[object HTMLDocument]
```

HTMLInputElement の親が HTMLBodyElement、その親が HTMLHtmlElement、その親が HTMLDocument、その親が null という結果になったわ。HTMLDocument は、document の値と同じね。

クリじい：null というのは、それ以上親はいないということじゃな。つまり、document が、最長老ということじゃな。 だから「ドムドム島」と言ったんじゃ。

なな：document って、document.getElementById() とか、document.write() の document ？

クリじい：その通りじゃ。

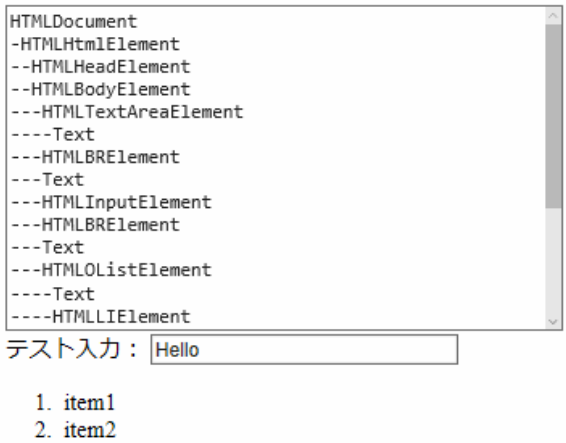
第6回　家系図を把握する

なな：次にやることは？

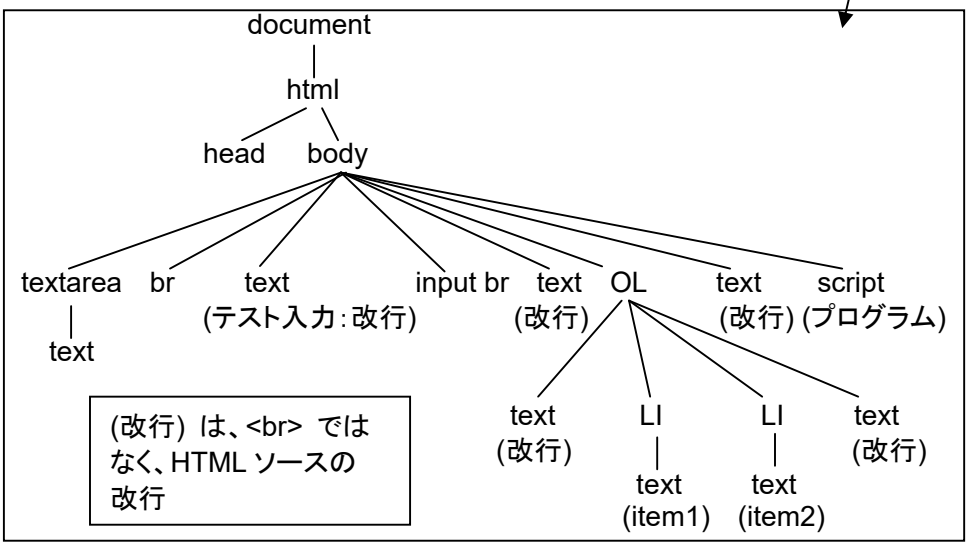
クリじい：家系図を作ることじゃな。島民の全体像をとらえるんじゃ。さっきと逆に、ノード. childNodes で、子供の一覧を得たり、ノード. childNodes.length で子供の人数を調べたり、ノード. childNodes[i] で、i 番目の子供ノード情報を得たりできるんじゃが、それを駆使して、document ファミリの家系図を作ってみた。ノードは、Element のより広い呼称。島民に対する人間みたいな関係と思ってくれ。島民の親は、人間の親、というニュアンスじゃな。調べる対象に、番号付きの箇条書きも入れてみた。

```
<textarea cols=50 rows=14 id=out></textarea><br>
テスト入力:
<input type=text id=testio value="Hello" size=30><br>
<ol>
<li>item1
<li>item2</li>
</ol>
<script>
var inp  = document.getElementById("in");
var outp = document.getElementById("out");
var geval = eval;

tree(0,document);

function tree(b, n) {
    for (var i=0; i<b; i++) outp.value += "-";
    outp.value += (""+n).slice(8,-1) + "\n";
    for (i=0; i<n.childNodes.length; i++) tree(b+1,n.childNodes[i]);
}
</script>
```

```
HTMLDocument
-HTMLHtmlElement
--HTMLHeadElement
--HTMLBodyElement
---HTMLTextAreaElement
----Text
---HTMLBRElement
---Text
---HTMLInputElement
---HTMLBRElement
---Text
---HTMLOListElement
----Text
----HTMLLIElement
-----Text
----HTMLLIElement
-----Text
----Text
---Text
---HTMLScriptElement
----Text
```



これを、分かりやすく書き換えると



第7回　家を作る

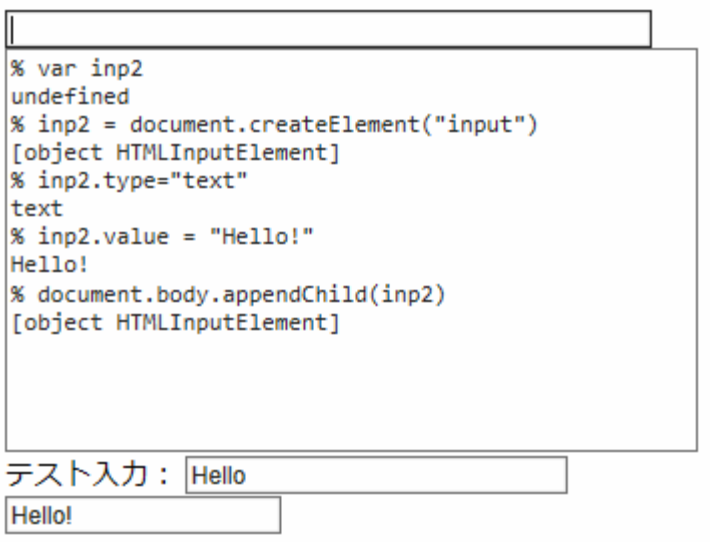
なな：その次にやることは？

クリじい：土地を分けてもらって、自分たちの家を作るんじゃ。

なな：具体的にどうするの？

クリじい：「inp2 = document.createElement("input")」で input タグを作って、それを、「document.body.appendChild(inp2)」で、画面に張り付けるんじゃ。input の、type=text とか、value=”Hello!” とかも設定する。

```
% var inp2
undefined
% inp2 = document.createElement("input")
[object HTMLInputElement]
% inp2.type="text"
text
% inp2.value = "Hello!"
Hello!
% document.body.appendChild(inp2)
[object HTMLInputElement]
```



なな：すごい！　画面の一番下に input 要素が追加されたわ！

第8回　DOM（Document Object Model）

DOM は、JavaScript などのプログラミング言語から HTML を使う仕組み。HTML ドキュメントを、ドキュメントオブジェクトというオブジェクトとして扱っている。

ブラウザや、そのバージョンによっ、てドキュメントツリーの構造が若干異なるという問題がある。HTML 文書内で使っている 「スペース、タブ、改行」を無視するか、テキストノード扱いするかなどで、扱いが異なる。そういうことを意識してプログラムを作らないと、特定のブラウザで誤動作するプログラムになってしまう。

DOM で扱うノードの種類は、HTML に関わるタグ内に記述された文字列を格納しているテキストノード、タグを格納している要素ノード、タグの属性を格納している属性ノード（HTML タグ内にあるプロパティーで、href、src、id など）をはじめ、多数の種類がある。要素ノードは Element クラスのインスタンス。Node クラスは、Element クラスの親クラス要素ノードは、Node クラスの命令や属性を使える。DOM で扱うすべてのオブジェクトは、Node クラスの命令や属性を使える。

<要素の参照>
- element = document.getElementById(“id”)　id 属性値から要素を参照
- elements = document.getElementsByName(“name”)　name 属性値から要素を参照
- elements = document.getElementsByTagName(“tagname”) タグ名から要素を参照
- elements = document.getElementsByClassName(“class”)　クラス属性値から要素を参照

<子ノードの参照>
- nodelist = node.childNodes　そのノードのすべての子ノードを参照
- node = node.firstChild　そのノードの最初の子ノードを参照
- node = node.lastChild　そのノードの最後の子ノードを参照
- bool = node.hasChildNodes()　そのノードが子ノードを持っているかどうか調べる

<親ノードの参照>
- node = node.parentNode　その要素の親にあたるノードを参照
- document = node.ownerDocument　その要素が含まれるトップレベルのドキュメントオブジェクトを参照

<兄弟ノードの参照>
- node = node.previousSibling　その要素の直前のノードを参照
- node = node.nextSibling　その要素の直後のノードを参照。

<ノードの情報参照>
- node.nodeValue　そのノードの値を参照。 element.value が推奨
- node.nodeName　そのノードの名称を参照。例：”INPUT”。element.tagName も可
- node.nodeType　そのノードの種類を参照。数値。例：1=ELEMENT_NODE。

1	ELEMENT_NODE	エレメントノード、要素に相当
2	ATTRIBUTE_NODE	アトリビュートノード、属性に相当
3	TEXT_NODE	テキストノード、タグ以外の文字データに相当
4	CDATA_SECTION_NODE	CDATA セクションノード
5	ENTITY_REFERENCE_NODE	エンティティリファレンスノード
6	ENTITY_NODE	エンティティノード
7	PROCESSING_INSTRUCTION_NODE	処理命令ノード
8	COMMENT_NODE	コメントノード
9	DOCUMENT_NODE	ドキュメントノード、階層のルートに相当
10	DOCUMENT_TYPE_NODE	ドキュメントタイプノード
11	DOCUMENT_FRAGMENT_NODE	ドキュメントフラグメントノード
12	NOTATION_NODE	ノーテーションノード

<ノードの生成、追加、削除等の操作>
- document.createElement(“tagname”)　要素を生成
- document.createTextNode(“textdata”)　テキストノードを生成
- parentnode.appendChild(element)　その要素の一番最後の子ノードとして追加
- parentnode.insertBefore(newnode,refnode)　その要素の指定した子ノードの前にノードを追加
- parentnode.removeChild(element)　その要素の子ノードを削除
- parentnode.replaceChild(newelement,oldelement)　その要素の子ノードを別の新しいノードに置き換える
- node.cloneNode(deep)　要素をコピーする。
  deep: true=子孫ノードも複製、false=node のみ

注意： document.write(“string”) は、DOM を破壊するので、非推奨。

第9回 イベント

先生: DOM の要素ノードに、イベントリスナを登録することで、マウスクリックなどのイベントに応答する JavaScript プログラムを指定することができます。登録方法が 3 種類あります。

なな: ひとつ目は？

先生: myButton.addEventListener('click', function(event) { alert('Hello world'); }, false); の形です。myButton は、ボタンなどの DOM 要素ノードです。イベントの種類は、ここでは click を指定しています。on click ではないので要注意。false の部分は、通常は false と指定してください。true にするのは、前処理をするという、特別、高度な使い方をする場合です。今は、忘れてください。Internet Explorer 8 以前は、この方法の代わりに、.attachEven() を使う必要がありましたが、今は、IE 11 とか、Edge が主流なので気にしないように。

なな: ふたつ目は？

先生: HTML のほうで、<button onclick="alert('Hello world!')"> のように指定する形。 alert('Hello world!') の代わりに、go() のように、自分で作った JavaScript の関数（メソッド）を呼び出すこともできます。

なな: みっつ目は？

先生: myButton.onclick = function(event) { alert('Hello world'); }; の形です。このみっつ目の方法の問題は、ひとつの要素のひとつのイベントにつき、ひとつのハンドラしか定義できないことです。myButton.onclick = の文がふたつ以上あると、最後に実行するものだけが有効になります。myButton.addEventListener で、同じイベントに対する設定がふたつ以上あっても、すべてが実行されることになります。ひとつ目とみっつ目の方法の関数では、event 引数 （event） をとる関数を定義できます。この引数からいろいろな情報を得ることができます。たとえば、

event.target	イベントが最初に送出されたターゲットへの参照。
event.timeStamp	イベントが生成された時刻。
event.type	イベントの名前。(大文字小文字を区別しない)

なな: マウスの場合は？

先生: 下記のような、イベント発生時のマウスカーソルの位置座標を得ることができます。

event.pageX/event.pageY	表示されていない部分を含めて、ページの左上を基点とする座標
event.screenX/event.screenY	ディスプレイの左上を基点とする座標
event.clientX/event.clientY	ブラウザの左上を基点とする座標
event.offsetX/event.offsetY	イベントが発生した要素の左上を基点とする座標

なな: マウスのイベントは、どんな要素に設定できるの？

先生: 個別の要素なら、その要素の部分でのイベントだけ拾えます。document.body や、document に設定した場合は、ブラウザ内のどこで発生したイベントでも拾えます。

なな: キーボードのイベントは？

先生: onkeydown などのイベントを個別の要素、document.body、document に設定できます。event.keyCode で、どのキーが押されたかの情報が得られます。

第10回　スタイルのクラスを動的に変更する

なな：要素ノードのサイズ、文字色、背景色などを同時に変えようとすると、プログラムが何行にもなってしまって大変なんだけど。

先生：そういう場合は、<style> でクラスごとにスタイルを定義しておいて、JavaScript では、要素のクラスを動的に切り替えると便利なのよ。例を使って説明するわね。

```
<style>
  .normal { size:30; background-color: green; }
  .attention { size:50; background-color: pink; }
</style>
テスト入力:
<input type=text id=testio value="Hello" class=normal>
<script>
var testp = document.getElementById("testio");
:
</script>
```

のようにしておいて、<input> の、class を normal から attention に変えれば、サイズと背景色を同時に変更できます。でも、注意点があります。「testp.class="attention"」でクラスを変えても、表示は変化しません。「testp.setAttribute("class","attention")」とか、「testp.className="attention"」にする必要があります。

なな：不思議ね。

先生：これも、要素ノードそのものがオブジェクトなのではなくて、要素ノートが JavaScript から要素らしく見えるようにしているだけだからなのね。こっそりとオブジェクトのアトリビュートを書き換えるのでは表示に反映できないけど、表示システムが気づくような方法でオブジェクトを書き換えると、表示システムがそれを表示に反映してくれるということね。

トリビア： いじわるテスト

クリじい： 今回は、DOM の要素ノードが、本当のオブジェクトでは無いことを暴くんじゃ。

なな： どうやって？

クリじい： DOM 要素ノードの、表示に関係あるような、正規のアトリビュートについては、オブジェクトと表示とが正しく結びつけられている。 JavaScript で、testp.size=50 を実行すれば、testp.setAttribute(“size”,”50) を実行したのと同じように、testp が指し示す要素ノードのサイズが 50 になる。ところが、表示に結び付かないような、その要素が本来持っていないようなアトリビュートを指定しようとすると、本当のオブジェクトでは無いことが見えてくるんじゃ。まあ、実際に役に立たないところまで本当のオブジェクトらしく見せるところまでは付き合っていられない、ということだ。

なな： う～～ん。何を言っているのかわからない。

クリじい： まあ、この実行例を見てくれ。

なな： ようするに、意味のないプロパティ（hoge）だと、オブジェクトと DOM は別々に変化するけど、意味のあるプロパティ（size）だと、オブジェクトと DOM が連動するということね。

先生： オブジェクトと DOM は別物だけど、システムががんばって連動させているということね。

なな： ブラウザさん、お疲れ様です！

トリビア：document.getElementById は長ったらしい？

```
<input type=text id=ipt>
<script>
var iptp = document.getElementById("ipt");
ipt.value = "Hello!";
</script>
```

の、「document.getElementById」 が、長ったらしい、わずらわしいという不満が多い。その対策。

```
<input type=text id=ipt>
<script>
function $(e) { return document.getElementById(e); }

$("ipt").value = "Hello!";
</script>
```

この発想のライブラリとして、prototype.js や jQuery が有名。

```
<form name=fm>
<input type=text name=ipt>
</form>
<script>
document.fm.ipt.value = "Hello!";
</script>
```

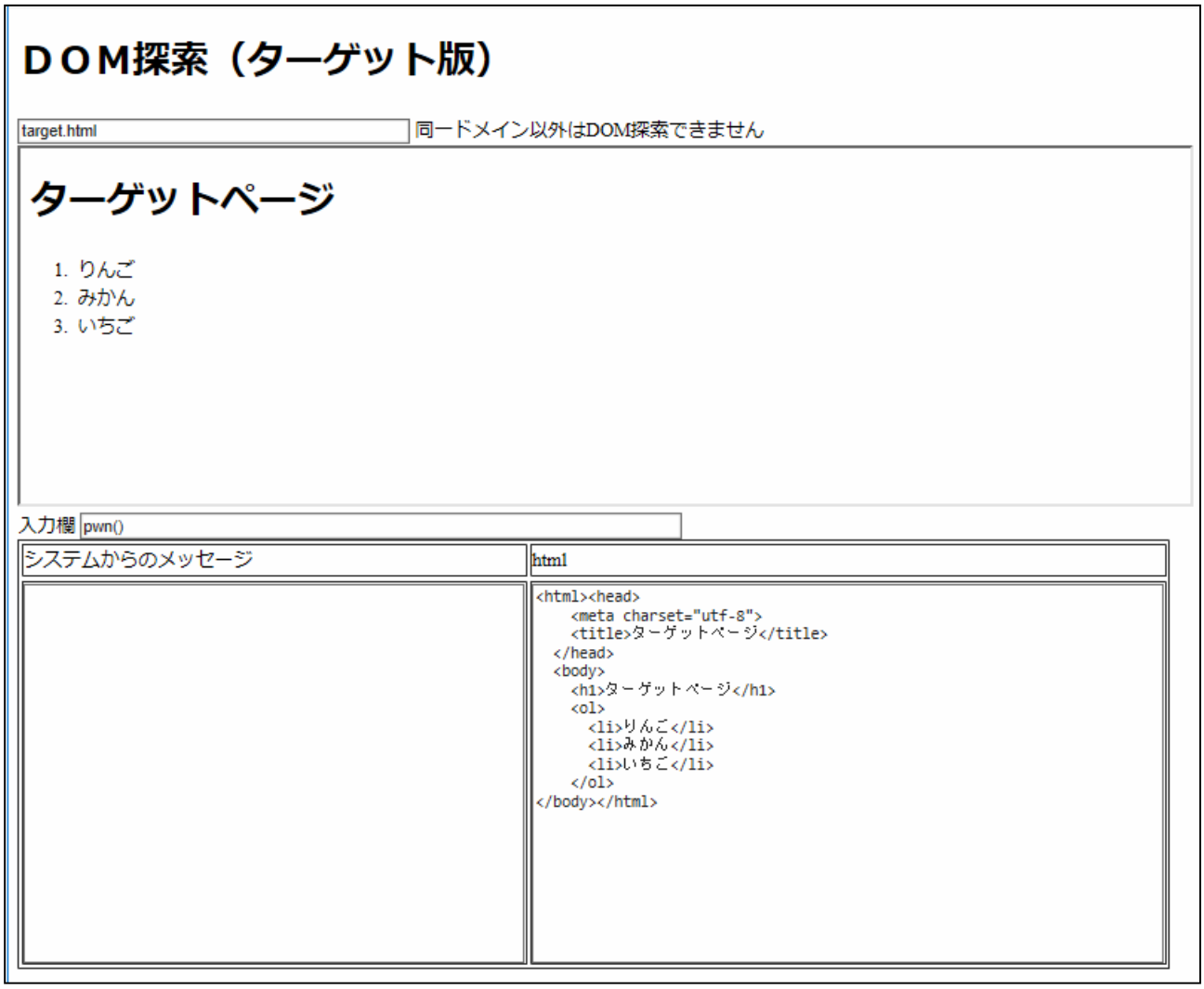
「document.fm.ipt.value」 は、「window.fm.ipt.value」、「fm.ipt.value」 でも OK。ただし、JavaScript から動的に生成したタグには使えない。name 属性を使えるのは、<a>, <applet>, <form>, <frame>, <iframe>, <img>, <input>, <map>,<meta>, <object>, <option>, <param>, <select>, <textarea>.要素だけ。

```
<input type=text id=ipt>
<script>
ipt.value = "Hello!";
</script>
```

なんと、document.getElementById を省ける。こちらも、JavaScript から動的に生成したタグには使えない。
でも、要素（ノード）を動的に画面に張り付ける appendChild() の返値を使うことができる。

これらのテクニックは、参照（読み出し）だけでなく、書き込みにも使える。

おまけ：UNIX のディレクトリ操作感覚で DOM 探索を行えるプログラムです。
　探索対象の HTML ファイルをロードして、探索できます。



target.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>ターゲットページ</title>
  </head>
  <body>
    <h1>ターゲットページ</h1>
    <ol>
      <li>りんご</li>
      <li>みかん</li>
      <li>いちご</li>
    </ol></body></html>
```

pwn = print working node
ls = list
cn = change node

使用例：

```
pwn()
=> <html>
ls()
=> 0:<head>
   1:<body>
cn(1)
=> <body>
ls()
=> 0:<h1>
   1:<ol>
cn(1)
=> <ol>
ls()
=> 0:<li>
   1:<li>
   2:<li>
cn(1)
=> <li>
type()
=>  みかん
```

注目ノード

子ノード確認

ノード移動

innerHTML を表示

使い方

home()	<html> ノードに移動
pwn()	print current working node
ls()	current working node のサブノード一覧表示
cn(n)	change node: n 番目のサブノードに移動
cn("n")	〃 n という名前のサブノードに移動
cn(-1)	〃 親ノードに移動
cn(next)	〃 次の兄弟ノード(nextElementSibling)に移動
cn(prev)	〃 前の兄弟ノード(previousElementSibling)に移動
cnById(id)	change node by id: id(文字列)のノードに移動
type()	current working node 内の文字列を表示
enter("c")	current working node 内の文字列を変更
enterA("an","av")	current working node に attribute 名と値を入力
mkn("t")	current working node に t 子タグ(ノード)を追加
rmn(n)	current working node の n 番目の子ノードを削除
isn("t")	current working node の前に兄弟タグを追加
apn("t")	current working node の後に兄弟タグを追加
target("url")	探索対象ウェブページの変更 (他ドメインのコンテンツは表示できても、DOM アクセス不可) (同一ドメインポリシー)

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>DOM探索（ターゲット版）</title>
    <script src="prog.js"> </script>
 </head>
 <body>
    <h1>DOM探索（ターゲット版）</h1>
    <input type=text id=turl value="target.html" size=50>
同一ドメイン以外は DOM 探索できません<br>
    <iframe id=ifrm height=300 width=100% src=target.html
                                        onload="loaded()"></iframe>
    <br>
    入力欄 <input type=text size=80 id=pg value=pwn()>
    <br>
    <table border=1>
    <tr><td>システムからのメッセージ</td><td>html</td><tr>
    <tr><td><textarea rows="20" cols="55" id=log></textarea></td>
        <td><textarea rows="20" cols="70" id=source></textarea></td></tr>
    </table>
 </body>
</html>
```

prog.js

```
var geval = eval, logp, pgp, sourcep, cwn;

function typeIs(obj) {
      return(Object.prototype.toString.call(obj).slice(8, -1)); }

window.onload = function() {
    logp = document.getElementById("log");
    pgp    = document.getElementById("pg");
    sourcep = document.getElementById("source");
    pgp.focus();
};
function go() {
  pgp.focus();
  try {          logp.value += pgp.value + "\n=> " + geval(pgp.value) + "\n";
                   pgp.value = ""; logp.scrollTop = logp.scrollHeight; }
  catch(e) { logp.value += pgp.value + "\n! " + e + "\n";
                   pgp.value = ""; logp.scrollTop = logp.scrollHeight; }
}
document.onkeydown = function (e){
    if(!e) e = window.event;
    if (e.keyCode == 13) go();
}
function pwn() {
    if (typeIs(cwn) == "HTMLDocument") return "document";
    if (typeIs(cwn) == "Window")     return "window";
    var s = cwn.outerHTML;
    s = s.substring(0,(s.indexOf(">")+1)); return s;
}
function phome() {
    cwn = document.documentElement; //head.parentNode;
    return pwn();
}
function home() {
    cwn = document.getElementById("ifrm").contentWindow.document.documentElement;
    return pwn();
}
function cnById(nm) {
    cwn = document.getElementById(nm); return pwn();
}
function cnAbs(nd) {
    cwn = nd; return pwn();
}
function ls() {
    var ov = "";
    if ((typeIs(cwn) != "HTMLDocument")&&(typeIs(cwn) != "Window")) {
        var nodes = cwn.children;
        for (var j=0; j<nodes.length; j++) {
            ov += j + ":";
            var s = "" + nodes[j].outerHTML;
            ov += s.substring(0,(s.indexOf(">")+1)) + "\n     ";
    }   }
    return ov+list(cwn);
}
```

```
function cn(n) {
    if (typeIs(n) != "Number") { cwn = eval("cwn."+n); }
    else if ((cwn == document)&&(n==-1)) { cwn = window; }
    else if ((typeIs(cwn)=="HTMLDocument")&&(n==-1)) {
          cwn=document.getElementById("ifrm").contentWindow; }
    else if (cwn == document)   { return n+" is illegal"; }
    else if (cwn == window)     { return n+" is illegal"; }
    else if ((n<-1)||(n>=cwn.children.length)) {
                                          return n+" is illegal"; }
    else if (n==-1)                  { cwn = cwn.parentNode; }
    else                               { cwn = cwn.children[n]; }
    return pwn();
}
function type() { return cwn.innerHTML; }

function list(node) {
    var os = "";
    for (var i in node) {
        if (typeIs(eval("node."+i)).search(/^HTML.*Element$/)==0) {
            os += i + ":";

            var s = (eval("node."+i)).outerHTML + "";
            s = s.substring(0,(s.indexOf(">")+1));
            os += s + "\n     ";
  }    }
  if(typeIs(node)=="Window") return "document:\n     "+os;
  return os;
}
function enter(m) { cwn.innerHTML = m; }
function mkn(n) { cwn.appendChild(document.createElement(n)); }
function rmn(n) { cwn.removeChild(cwn.children[n]); }
function isn(n) { cwn.parentNode.insertBefore(document.createElement(n),cwn); }
function apn(n) { cwn.parentNode.insertBefore(document.createElement(n),cwn.nextSibling); }
function enterA(an,av) { cwn.setAttribute(an,av); }

var next = "nextElementSibling";
var prev = "previousElementSibling";
var url="target.html";

function target(nurl) {
    url = nurl;
    document.getElementById("ifrm").src=nurl;
    document.getElementById("turl").value=url;
}
function loaded() {
    home();
    document.getElementById("turl").value=url;
    document.getElementById("source").value = cwn.outerHTML;
}
```

上級者へのプレゼント：XPath（XML Path Language）

XPath という言語による式を用いて、XML や HTML 文書内で複雑な条件を指定してノードを探し出す方法。

＜ Xpath を使う ＞

XPath を使って、ノードを検索するには、document.evaluate メソッドを使う。document.getElementById の代わりに使うというイメージ。5 個の引数を渡して XPath を処理してもらい、結果を XPathResult というオブジェクトで返してもらう。

例：
```
var result = document.evaluate('/html', document, null, XPathResult.FIRST_ORDERED_NODE_TYPE, null);
```

- 第一引数：XPath 文。
- 第二引数：ルートノード。このノードも含め、このノードより下で探索する。
- 第三引数：名前空間解決器。HTML の場合は、名前空間がないので、null を指定する。
- 第四引数：結果のタイプ。たとえば、XPathResult.FIRST_ORDERED_NODE_TYPE。
- 第五引数：再利用器。null の場合、新しい XPathResult が作られて結果として返される。使い終わった XPathResult オブジェクトを渡すと、新しい XPathResult オブジェクトを作らずに、それを書き換えて返す。

第四引数の詳細

どんな形式で検索結果を返すのかを指定する定数。形式の違いにより、XPathResult オブジェクトからノードを得る方法が異なる。ORDERED は、ノード並び順が文書順、UNORDERED の場合、並び順は適当。UNORDERED のほうが検索が速い可能性がある。

<ANY_UNORDERED_NODE_TYPE>、<FIRST_ORDERED_NODE_TYPE>

当てはまるノードの集まり（ノードセット）のうち、1 つだけを取得する。UNORDERED の場合、探す順番が不定なので、複数あった場合どれが出てくるか分からない。ORDERED の場合、文書順で探して最初に見つかったノードとなる。結果の singleNodeValue プロパティにノードが入っている。

```
var result = document.evaluate('/html', document, null,
                                XPathResult.FIRST_ORDERED_NODE_TYPE, null);
console.log(result.singleNodeValue.tagName);
```

<UNORDERED_NODE_SNAPSHOT_TYPE>、<ORDERED_NODE_SNAPSHOT_TYPE>

ノードセットの全結果をまとめて取得する。結果の snapshotLength に、結果のノードの個数が入っている。結果のノードには 0 から番号がついている。 snapshotItem メソッドに数値を渡すことでその番号のノードが返ってくる。

```
for (var i=0;i<result.snapshotLength;i++) console.log(result.snapshotItem(i).tagName);
```

<UNORDERED_NODE_ITERATOR_TYPE>、<ORDERED_NODE_ITERATOR_TYPE>

ノードセットの結果を、ひとつずつ取得する。結果の iterateNext メソッドを呼び出すことでその都度次のノードを探索して返す。もう無い場合は null になる。途中でノードツリーに変化があると結果は無保証。

```
var node;
while (node = result.iterateNext())  console.log(node.tagName);
```

＜ XPath 言語 ＞

```
<div>
  サンプル
  <p class="class1">クラス1</p>
  <p class="class2">クラス2</p>
  <p class="class3">クラス3</p>
</div>
```

```
<input type=text id=inp>
<textarea id=outp></textarea>
<script>
var result = document.evaluate(inp.value, document, null,
    XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null);
for (var i=0;i<result.snapshotLength;i++) {
  var node = result.snapshotItem(i);
  outp.value += node.innerHTML + "\n";
}
</script>
```

```
/html/body/div/p[2]                                  先頭からきちんと書いた場合  -> 2
//p[2]                                               //で途中のパスを省略          -> 2
//p[@class="class2"]                                 属性の値を指定する            -> 2
//p[@class="class1"]/following-sibling::p[@class="class2"]    前隣の要素から指定。[]で条件指定。-> 2
//p[@class="class3"]/preceding-sibling::p[@class="class2"]    後ろ隣の要素から指定。          -> 2
//p[text()="クラス2"]                                完全一致  -> 2
//p[.="クラス2"]                                     完全一致  -> 2
//p[contains(text(), "2")]                           部分一致  -> 2
//p[contains(., "2")]                                部分一致  -> 2
//p[contains(., "ク") and contains(., "2")]          -> 2
```

```
<div>
  サンプル
  <div>
    <p class="class1">クラス1</p>
    <p class="class2">クラス2</p>
    <p class="class3">クラス3</p>
  </div>
  <div id=here>
    <p class="class4">クラス4</p>
    <p class="class5" id=there>
                      クラス5</p>
    <p class="class6">クラス6</p>
  </div>
</div>
```

```
<script>
var here = document.getElementById("here");
var result = document.evaluate(inp.value, here, null,
    XPathResult.ORDERED_NODE_SNAPSHOT_TYPE,
    null);
for (var i=0;i<result.snapshotLength;i++) {
    var node = result.snapshotItem(i);
    outp.value += node.innerHTML + "\n";
}
</script>
```

```
/html/body/div/div/p[2]                              絶対パス  -> 2 5
//p[2]                                               絶対パス  -> 2 5
/descendant::p[2]                                    絶対パス  -> 2
//p[last()]                                          絶対パス  -> 3 6
./p[2]                                               相対パス  -> 5
p[2]                                                 相対パス  -> 5
//p[contains(., "ク")]                               絶対パス 部分一致  -> 1 2 3 4 5 6
.//p[contains(., "ク")]                            相対パス 部分一致  -> 4 5 6
//p[contains(., "2") or contains(., "3")]            -> 2 3
//p[@id]                                             属性存在          -> 5
//*[@class]                                          ワイルドカード  -> 1 2 3 4 5 6
//*[@id="there"]                                     ワイルドカード  -> 5
```

XPath 文を いろいろ 試してみるプログラム

```
<!doctype html>
<html>
  <head>
    <title>XPath test</title>
    <style>div { border:solid 1px; width:676px; }</style>
  </head>
  <body>
    <h2>XPath test: 選択された要素の innerHTML を表示します</h2>
    <input type=text id=in value="/html/body/div/div/p[2]" autofocus size=109><br>
    <textarea cols=90 rows=9 id=out></textarea><br>
    <div>
      サンプル
      <div>
        <p class="class1">クラス１</p>
        <p class="class2">クラス２</p>
        <p class="class3">クラス３</p>
      </div>
      <div id=here>
        <p class="class4">クラス４</p>
        <p class="class5" id=there>クラス５</p>
        <p class="class6">クラス６</p>
      </div>
    </div>
    <textarea cols=90 rows=15 readonly>
      HTML
      <div>
        サンプル
:
      </div>
    </textarea><br>
    <textarea cols=90 rows=15 readonly>
XPath 記述例:
:
    </textarea>      <script>
var inp   = document.getElementById("in");
var outp = document.getElementById("out");
var testp = document.getElementById("testio");
var geval = eval;

document.onkeydown = function (e) {
    if (e.keyCode != 13) return;
    var here = document.getElementById("here");
    outp.value = "\"" + inp.value + "\"\n\n";
    try {
        var result = document.evaluate(inp.value, here, null,
                                       XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null);
        for (var i=0;i<result.snapshotLength;i++) {
            var node = result.snapshotItem(i);
            outp.value += node.innerHTML + "\n";
        }
    } catch(e) { outp.value += "! " + e + "\n"; }
    outp.scrollTop = outp.scrollHeight;
    inp.value = ""; //inp.focus();
}
</script></body></html>
```