WebForm
HTMLを使った
入力ダイアログ

注意:
WebFormはInternet Explorer 11, Edgeモードでないと正常に実行しません。
一度、以下の設定を実行しないと正常に実行しません。
WebForm.setVersion(appname, version)
--- 実行アップリappnameで WebFormが扱うInternetExploreのバージョンをversionに設定する。
現在(2017年8月13日) version>=11000 で実行していないとWebFormは正常動作しない

WebFormの仕様

WebFormはOgOからBrowserを操作するライブラリー。

[機能]
入力DialogをHTMLで実装し
それをOgOのスクリプトで作成・操作し、入力値を取得する。

[メリット]
WebForm以前は、入力GUIはC++で実装し、開発時間がかかった。また、MFCなど複雑でライブラリーの知識が必要であった。

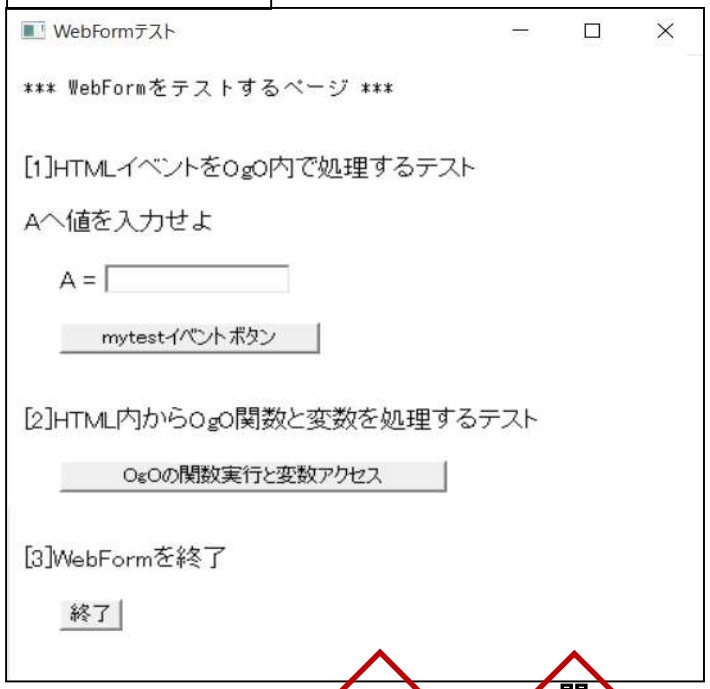
このGUIを、ユーザインタフェースの絶対的かつ将来ずっと標準となるHTMLで実装し、制御をOgOスクリプトで実現することは大きな価値がある。

その価値とは、入力画面が早く簡単に作成でき、テクニックはWebでいくらでも探して利用できる。

それとWeb化へ一歩進める。

Browser＋HTML ＋OgO

Web Browser



＋

== HTMLコード ==

```
<html>
<head>
<title>WebFormテスト</title>
<script type="text/javascript">
 ....
</script>
</head>

<body>
<pre>
*** WebFormをテストするページ ***
</pre>
<br>
<div>
[1]HTMLイベントをOgO内で処理するテスト<br>
    <div id="xyz">Aへ値を入力せよ</div><br>
    A = <input type="text" id="A"><br><br>
    <input type="button" id="mytest" value="mytestイベントボタン" onclick="ogo_sendEvent('mytest')"><br>
</div>
   .....
</body>
</html>
```

作成

関数呼出

イベント

関数呼出

== OgOスクリプト ==

```
var {WebForm} = require('wbf');
var d = new WebForm('file:///D:/work/test.htm‘);

var pos = d.position();
d.position(pos[0], pos[1], pos[0]+500, pos[1]+600);

//イベント関数を登録
d.addEventListener("mytest", function() {
  var self = this.self;
  var script = this.script;
  var document =  this.document;
  var name = this.name;

// html内の変数 abcを操作
  script.abc = 12345;
script.ck_abc();  // HTML内のck_abc()を実行
alert("[ogo->html] abc=" + script.abc); //abc=12345
```

```
 // htmlのdocumentを操作
 alert("[Aへ値を入力せよ]が[Aあいえお]へchangeします!");
 var xyz = document.getElementById("xyz");
 xyz.innerText = “Aあいえお"; //=> <div id="xyz" name="xyz">Aあいえお</div>

  // html textテキスト入力値を取得
  var A = document.getElementById("A");
  alert("[ogo->html document A] A="+A.value);
});
d.addEvent("myclose", function(element) {  d.close(); });

var ans_sum;
function sum(n) {
  var ans = 0;
  for (var i=1; i<=n; i++) ans += i;
  ans_sum = ans;
  return ans;
}
```

WebForm
Browser＋HTMLをOgOで
操作するライブラリ

- WebFormライブラリを使うための準備

```
var  {WebForm} = require(‘wbf’);
```

WebForm作成

- var d = new WebForm(url
  [, parentWnd, is_child]);

  url : HTMLのリソース場所
  parentWnd : 親(または所有者)Wndowハンドル
  -1ならアプリメインWindowハンドル
  省略の場合、親(または所有者) Windowなし
  is_child : trueなら、parentWndの子Windowとする
  省略の場合、falseとなり、ポップアップモードレスWindowとなる。

  var d = new WebForm('file:///webform-sample.html');

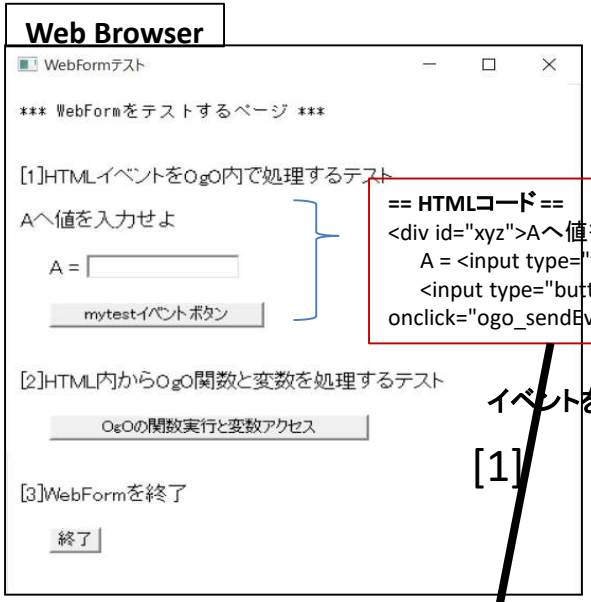
- 位置・サイズ変更

  var pos = d.position(); //(pos[0],pos[1]):左上 (pos[2],pos[3]):右下
  d. position(pos[0],pos[1], pos[0]+W, pos[1] +H); // W:幅、H:高さ

- Windowハンドルの取得

  var hwnd = d.hwnd;

イベント処理とHTMLの操作(in OgO)



== HTMLコード ==

```
<div id="xyz">Aへ値を入力せよ</div><br>
    A = <input type="text" id="A"><br><br>
    <input type="button" id="mytest" value="mytestイベントボタン"
onclick="ogo_sendEvent('mytest')"><br>
```

== HTML内JavaScriptコード ==

```
<script type="text/javascript">
var abc=100;
function add(a,b) { return a + b; }
・・・
function myGetElementById(id) { return document.getElementById(id); }
</script>
```

イベントを転送

[1]

[2]

変数アクセス

v = **script.get**(変数名); // 取得

script.set(変数名, value); // 設定

関数呼び出し

ret = **script.exec**(関数名, 引数);

要素オブジェクト取得

elem = **document**.getElementById(**ID名**);

要素の値

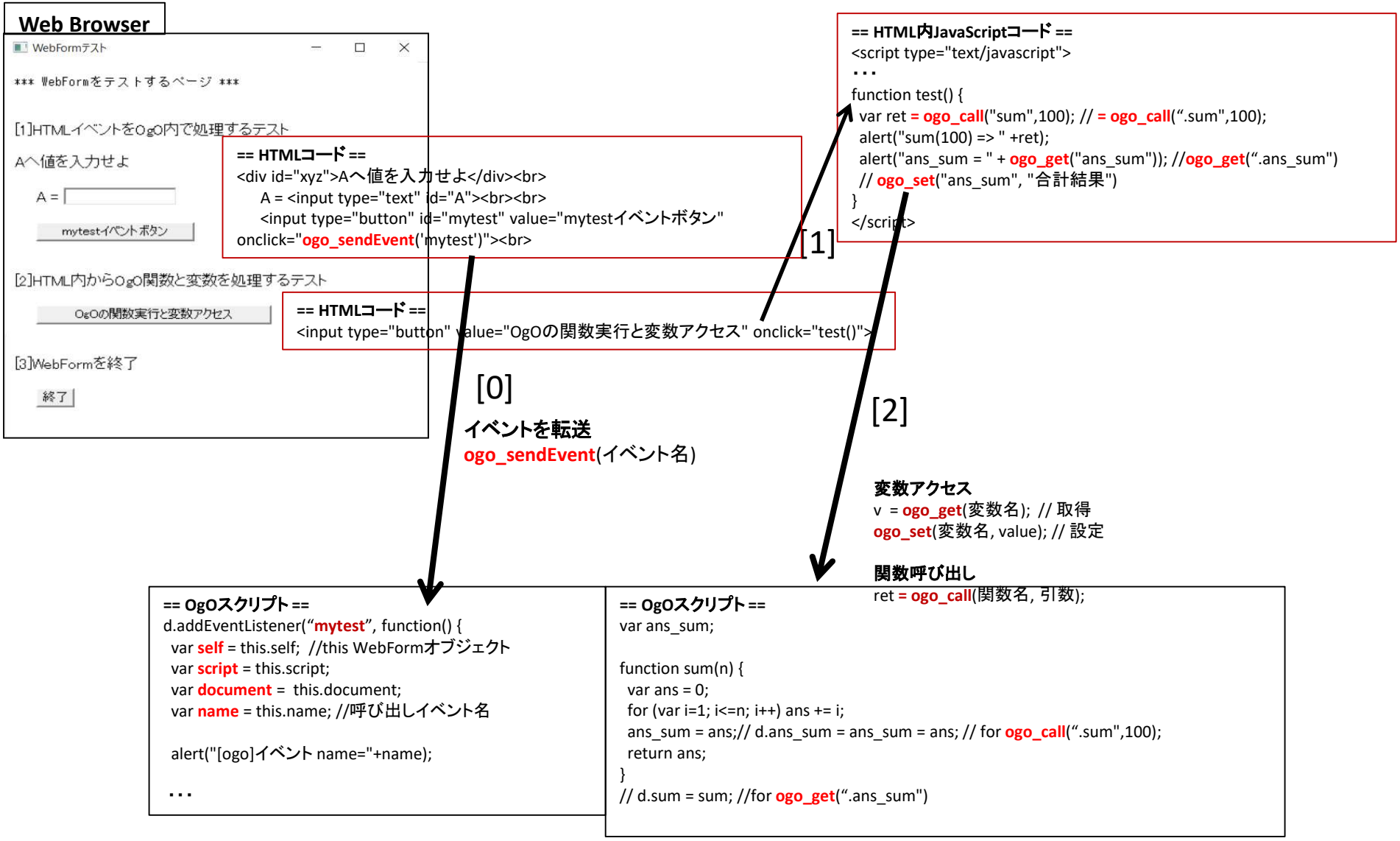
v = elem.value;

elem.value = valu

== OgOスクリプト ==

```
d.addEventListener("mytest", function() {
 var self = this.self;  //this WebFormオブジェクト
 var script = this.script;
 var document =  this.document;
 var name = this.name; //呼び出しイベント名

 alert("[ogo]イベント name="+name);

 // html内の変数 abcを操作
 script.abc = 12345;
 script.ck_abc();  // HTML内のck_abc()を実行
 alert("[ogo->html] abc=" + script.abc); //abc=12345

 // html内の関数をcall
 var ret = script.add(10,20);
 alert("[ogo->html] add(10,20)=>"+ ret);

 // htmlのdocumentを操作
 var xyz = document.getElementById("xyz");//=script.myGetElementById("xyz");
 xyz. innerText = "Aあいえお"; //=> <div id="xyz" name="xyz">Aあいえお</div>

 // html textテキスト入力値を取得
 var A = document.getElementById("A"); //=script.myGetElementById("A");
 alert("[ogo->html document A] A="+A.value);
});
```

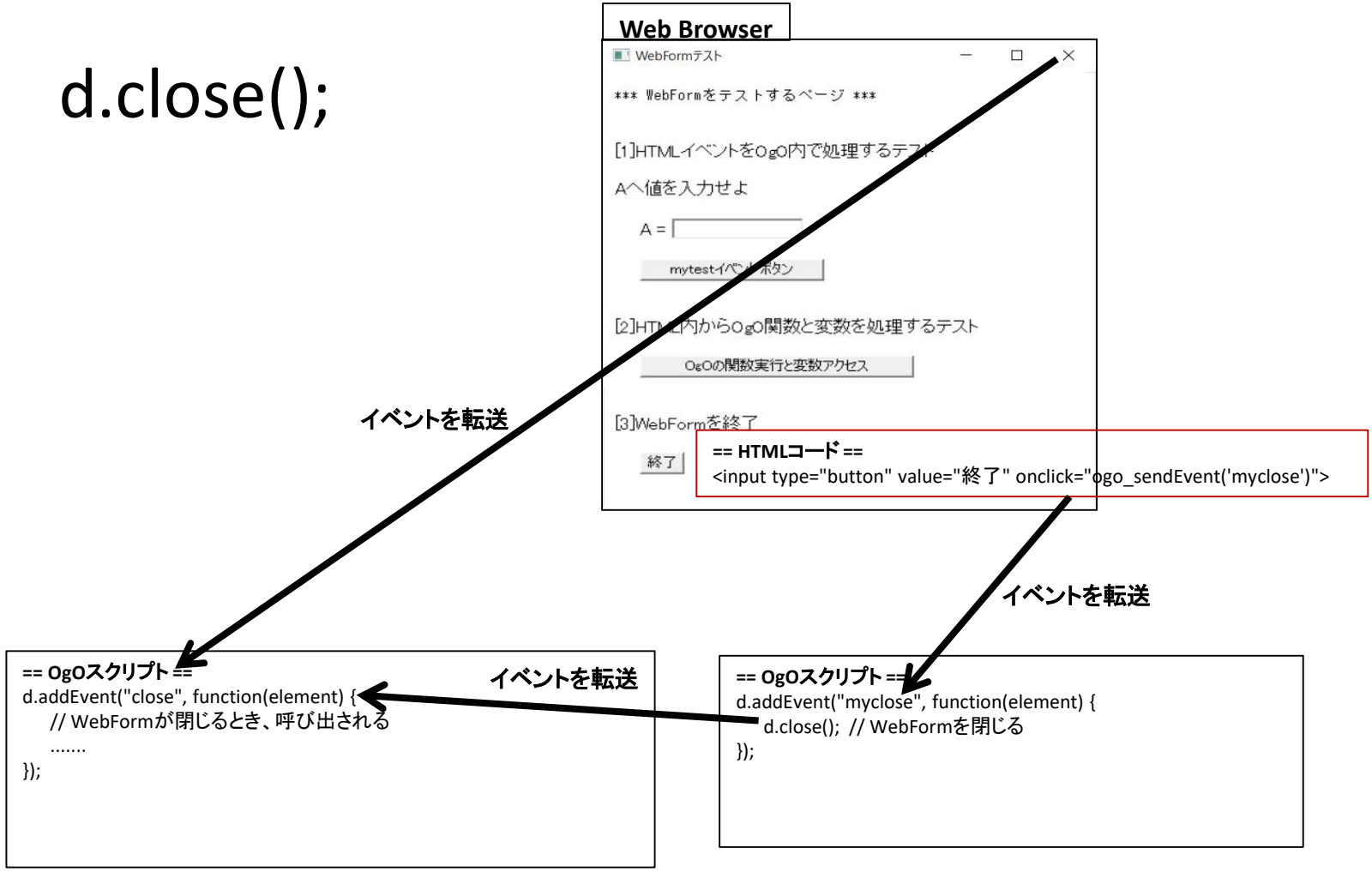
HTML内からOgOを操作(in HTML)



WebFormを終了

- webformオブジェクト.close()

  d.close();



WebForm API

- WebFormオブジェクト

in OgO

var {WebForm} = require('wbf');
WebForm --- コンストラクタ
show, navigate, close, pos, title --- method
browser, script, document --- property
addEventListener, removeEventListener --- event

- OgOからHTML内の関数、変数、DOMを扱う

in OgO

x=script.vname; script.vname=value; --- HTML内のJS変数へのアクセス
script.func(arg) --- HTML内のJS関数呼び出し
document --- HTML内のドキュメントオブジェクト (method,propertyあり)

- HTMLからOgO内の関数、変数を扱う

in HTML

ogo_get, ogo_set --- OgO内の変数へアクセス
ogo_exec, ogo_call --- OgO内の関数呼び出し

WebFormオブジェクト

in OgO

- WebFormライブラリ呼び出し
  var {WebForm} = require('wbf');
- WebFormコンストラクタ
  var webform = new WebForm(url[, parent]); --- 指定url のWebFormオブジェクトを作成し表示する。
  var webform = new WebForm([parent]); --- 空のWebFormオブジェクトを作成する。
  parent : 親Windowハンドル (-1)ならアプリMainWidowのハンドルとする。
  is_child : trueなら、parentの子Windowとする
  省略の場合、falseとなり、ポップアップモードレスWindowとなる。
- webform.navigate(url)
  urlをloadし表示する。
- webform.close()
  WebFormを閉じる。この関数が実行されると、イベント"close" が発生します。
- Webform.isClosed()
  WebFormが閉じられてればtrueを返し、まだ使えればfalseを返します。
- Webform.show(visible)
  visible=trueなら表示、visible=falseなら非表示
  省略ならvisible=trueと同じで表示する。
- Ret=webform.title()
  --- タイトル取得
  webform.title(name)
  ---タイトル設定
  WebFormのWindowタイトル表示を操作
- pos = webform.position()
  --- (pos[0],pos[1]):左上、(pos[0],pos[1]):右下
  webform.position(left,top, right,botom)
  --- 位置とサイズを設定(引数がundefinedなら変更しない)

- webform.browser プロパティ
  WebFormのbrowser機能を扱えるオブジェクト。
  メソッド・プロパティは「InternetExplorer.Application」を参照せよ。
- webform.script プロパティ
  HTML内のJSエンジン操作するオブジェクト。
  「とほほJavaScriptリファレンス」などを参照せよ。
  イベント関数内のほぼthis.scriptと同じ。
- webform.document プロパティ
  HTML内のドキュメントを操作するオブジェクト。
  「とほほJavaScriptリファレンス」などを参照せよ。
  イベント関数内のほぼthis.documentと同じ。
- webform.busy プロパティ
  trueならWebページが読み込み中、falseなら読み込み終了
- webform.readyState プロパティ
  ドキュメントの読み込み状態を示す。
  0:デフォルト値。読み込み開始前, 1:読み込み中, 2:読み込み中で操作可能, 3:読み込み完了
- webform.hwnd, hparent, isChild プロパティ
  ド自身のWindowハンドル、Window親ハンドルまたはOwnerハンドル、子Windowなら化のtrueとなるプロパティ

- webform.addEventListener (name, func)
  イベント(name)とそのを処理関数(func)を登録する。
  HTML内で、
  ogo_sendEvent(name,arg,...);
  を実行すると、ここで登録した関数に処理が移る。

  NameにはWebFormエンジンが出す特殊なイベントBeforeNavigator, DocumebtComplete, closeの3つがある。
  " BeforeNavigator " は、指定URLへの移動しようした際に発生します。
  " DocumentComplete " は、ドキュメントが完全に読み込まれ初期化されると発生します。。
  "close" は、WebFormが閉じらるとき呼び出されます。
  ☒ボタンをクリックするか、webform.close()が実行されるとWebFormが閉じられます。
- webform.removeEventListener (name, func)
  webform.addEventListener(...)で登録したイベントとその関数を削除する。
  引数funcは省略可能で、同じnameで登録した関数すべてを削除する。

OgOからHTML内の関数,変数,DOMを扱う

- OgOでイベント処理関数を登録

```
webform.addEventListener(eventname, function() {
    var script = this.script;
    var document = this.documennt;
    var name = this.name;
    ・・・ // イベント処理
});
```

- HTML内のJS変数へのアクセス

ret = script.varname
--- HTML内JS変数 varnameの値を取得
script.varname = value
--- HTML内JS変数 varnameに値valueを代入

- HTML内のJS関数呼び出し

script.func(arg1,arg2)

- document --- HTML内のドキュメント操作オブジェクト

「とほほJavaScriptリファレンス」のドキュメント(Document)、エレメント(Element)などを参照せよ。
プロパティ取得 x = elem.propname,プロパティ設定 elem.propname = value

```
var elem = document.getElementById(id);
elem.innerText = “あいうえお”;
```

- 注意

webform.script、webform.document
は使うことは薦めません！
WebFormのBrowser状態によって正しく動作」しないことがあるからです。
(開発者から一言、
恐らく正しく動作すると思うがチェックテストしてから使って欲しい)

in OgO

Scriptを使用して、HTML内の関数呼び出しで、Array,Object型の引数を渡しは
配列値は

```
SciptObj.from(script, ogo_array)
/*
var size = ogo_array.length;
var a = new script.Array(size);
for (var i=0; i<size; i++) {
  a[i] = ogo_array[i];
}
*/
```

Object値は

```
SciptObj.from(script, ogo_obj)
/*
var obj = new script.Object();
for (var i in ogo_obj) {
  obj[i] = ogo_obj[i];
}
*/
```

document.getElementById(id)は正しく動作しないことが多い。
対策としてHTML側に

```
<script type="text/javascript">
  function myGetElementById(id) { return document.getElementById(id); }
</script>
```

を記述し、

OgOスクリプトで
document.getElementById(id)
の代わりに
script. myGetElementById(id)
を使用してください。

OgOからHTML要素へのアクセス関数

in OgO

- 要素のvalue値

```
var value = webform.get_value(id);  //取得
webform.set_value(id, value);         //設定
webform.reg_value(id);    //値を残すためcloseイベントで実行
```

- 要素の数値value値

```
var num = webform.get_number(id);  //取得
webform.set_number(id, num);         //設定
webform.reg_number(id);    //値を残すためcloseイベントで実行
```

- 要素のchecke属性値

```
var onoff = webform.get_checked(id);  //取得
webform.set_checked(id, onoff);         //設定
webform.reg_checked(id);    //値を残すためcloseイベントで実行
```

- (nameの)要素群からselect id

```
var id = webform.get_selected(name);  //取得
webform.set_selected(name, id);         //設定
webform.reg_ selected(id);    //値を残すためcloseイベントで実行
```

- 要素の指定attribut値

```
var value = webform.get_attribute(id, attribute);  //取得
webform.set_attribute(id, attribute, value);         //設定
webform.reg_attribute(id, attribute);    //値を残すためcloseイベントで実行
```

webform.close();の実行後や
WebFormのcloseボタンでWebFormを閉じた後でも、WebForm内のHTML要素の値が取得できるようにするためにCloseイベントで
reg_value,reg_number,reg_checked,reg_selected,reg_attributes関数
を実行してください。
そうすれば、WebFormを閉じた後でも
get_value,get_number,get_checked,get_selected,get_attributes関数で
それらの値を取得することができます。

HTMLからOgO内の関数、変数を扱う

- ogo_get(varname)
  --- OgO内JS変数 varnameの値を取得
  varname="objname.prop.propc" もOK
  ogo_set(varname, value)
  --- OgO内JS変数 varnameに値valueを代入
  varname="objname.prop.propc" もOK

in HTML

技術メモ:
ogo_get, ogo_set, ogo_call(ogo_exe)関数の実行で、varname,funcnameの名前が見つからないことがある。

そういう場合、
OgO内のスクリプトに

```
webform.__eval__ = function(expr) {
    return eval(expr);
};
```

を記述すると名前解決をしてくれて名前解決を解消してくれます。

- ret = ogo_call(funcname, 引数)
  ret = ogo_exec(funcname, 引数)
  --- OgO内のJS関数 (名前 funcname)を呼び出し実行する。
  funcname ="objname.prop.func" もOK
  funcname =".prop.func" であった場合、OgOスクリプトのwebform.prop.func(引数)が実行される

- ogo_sendEvent(eventname,引数)
  OgO内で、
  webform.addEventListener(eventname, func);
  で登録されていれば、この関数func(引数)へ処理が移る。

Browserバージョンを設定

- WebForm.setVersion(appname, version)

--- 実行アップリappnameで WebFormが扱うInternetExploreのバージョンをversionに設定する。
　　現在(2017年8月13日) version>=11000 で実行していないとWebFormは正常動作しない
appname=nullなら現在実行プログラム名を与えたことになり、versionを省略したら11001を採用。

例:
WebForm.setVersion("LCDMaster.exe", 11001);　//「Internet Explorer 11, Edgeモード」に設定

バージョン　参照:http://www.ipentec.com/document/document.aspx?page=csharp-change-webbrower-control-internet-explorer-version

値 (10進数)	値 (16進数)	バージョンと意味
11001	0x2AF9	Internet Explorer 11, Edgeモード (最新のバージョンでレンダリング)
11000	0x2AF8	Internet Explorer 11
10001	0x2711	Internet Explorer 10, Standardsモード
10000	0x2710	Internet Explorer 10 (!DOCTYPE で指定がある場合は、Standardsモードになります。)
9999	0x270F	Internet Explorer 9, Standardsモード
9000	0x2710	Internet Explorer 9 (!DOCTYPE で指定がある場合は、Standardsモードになります。)
8888	0x22B8	Internet Explorer 8, Standardsモード
8000	0x1F40	Internet Explorer 8 (!DOCTYPE で指定がある場合は、Standardsモードになります。)
7000	0x1B58	Internet Explorer 7 (!DOCTYPE で指定がある場合は、Standardsモードになります。)

GPU設定

- WebForm.setGpu (appname, on)

  --- 実行アップリappnameで のWebFormのGPUを設定する。

appname=nullなら現在実行プログラム名を与えたことになる。
on：省略または1なら、GPUが有効となる。
on：0なら、GPUが無効となる。
例：
WebForm.setGpu(“LCDMaster.exe”); // LCDMaster.exe でのWebFormはGPUが有効

サンプル・チェックプログラム

- webform-sample.js, webform-sample.html

== HTMLコード == webform-sample.html

```
<html>
<head>
<title>WebFormテスト</title>

<script type="text/javascript">
//==[1]==
var abc=100;
function add(a,b) { return a + b; }

//==[2]==
function test() {
  var ret = ogo_call("sum",100);
  alert("sum(100) => " +ret);
  alert("ans_sum = " + ogo_get("ans_sum"));
  // ogo_set("ans_sum", "合計結果")
}
</script>
</head>
<body>

<pre>
*** WebFormをテストするページ ***
</pre>
<br>
<div>
[1]HTMLイベントをOgO内で処理するテスト<br>
     <div id="xyz">Aへ値を入力せよ</div><br>
     A = <input type="text" id="A"><br><br>
     <input type="button" id="mytest" value="mytestイベントボタン"
onclick="ogo_sendEvent('mytest')"><br>
</div>
<br><br>
<div>
[2]HTML内からOgO関数と変数を処理するテスト<br>
   <br>
     <input type="button" value="OgOの関数実行と変数アクセス" onclick="test()"><br>

</div>
<br><br>
<div>
[3]WebFormを終了<br>
  <br>
     <input type="button" value="終了" onclick="ogo_sendEvent('myclose')"><br>
</div>

</body>
</html>
```

== OgOスクリプト == webform-sample.js

```
var {WebForm} = require('wbf');
var d = new WebForm('d:/work/webform-sample.html');

var pos = d.getPos();
d.setPos(pos[0], pos[1], pos[0]+500, pos[1]+600);

//イベント関数を登録
d.addEvent("mytest", function() { //正式 d.addEventListener("mytest", function() {
  var script = this.script;
  var document =  this.document;
  var name = this.name;

  alert("name="+name);

  // html内の変数 abcを操作
  script.abc = 12345;
  alert("html abc=" + script.abc); //abc=12345

  // html内の関数をcall
  var ret = script.add( 10,20);
  alert("html add(10,20)=>"+ ret);

  // htmlのdocumentを操作
  var xyz = document.getElementById("xyz");
  xyz.innerText = "あいえお"; //=> <div id="xyz" name="xyz">あいえお</div>

  // html textテキスト入力値を取得
  var A = document.getElementById("A");
  alert("A="+A.value);
});

d.addEvent("myclose", function(element) {d.close();});

var ans_sum;

function sum(n) {
  var ans = 0;
  for (var i=1; i<=n; i++) ans += i;
  ans_sum = ans;
  return ans;
}

////////////
d.addEvent("DocumentComplete", function() {
  var script = this.script;
  var document =  this.document;

  var A = document.getElementById("A");
  A.value = 987654;
});
```

OgOからhtml内の配列要素アクセス方法

- webform-sample.js, webform-sample.html

== HTMLコード == webform-sample.html

```
<html>
<head>
<title>WebFormテスト</title>

<script type="text/javascript">
//==[1]==
var   array = [1,2,3,4,5];
var abc=100;
function add(a,b) { return a + b; }

//==[2]==
function test() {
  var ret = ogo_call("sum",100);
  alert("sum(100) => " +ret);
  alert("ans_sum = " + ogo_get("ans_sum"));
  // ogo_set("ans_sum", "合計結果")
}
</script>
</head>
<body>

<pre>
*** WebFormをテストするページ ***
</pre>
<br>
<div>
[1]HTMLイベントをOgO内で処理するテスト<br>
     <div id="xyz">Aへ値を入力せよ</div><br>
     A = <input type="text" id="A"><br><br>
     <input type="button" id="mytest" value="mytestイベントボタン"
onclick="ogo_sendEvent('mytest')"><br>
</div>
<br><br>
<div>
[2]HTML内からOgO関数と変数を処理するテスト<br>
    <br>
     <input type="button" value="OgOの関数実行と変数アクセス" onclick="test()"><br>

</div>
<br><br>
<div>
[3]WebFormを終了<br>
   <br>
     <input type="button" value="終了" onclick="ogo_sendEvent('myclose')"><br>
</div>

</body>
</html>
```

== OgOスクリプト == webform-sample.js

```
var {WebForm} = require('wbf');
var d = new WebForm('d:/work/webform-sample.html');

var pos = d.getPos();
d.setPos(pos[0], pos[1], pos[0]+500, pos[1]+600);

//イベント関数を登録
d.addEvent("mytest", function() { //正式 d.addEventListener("mytest", function() {
  var script = this.script;
  var document =  this.document;
  var name = this.name;

  alert("name="+name);

  // html内の配列アクセス
  var  A = Script.get("array");
  var  N = A. length;
  for (var i=0; i<N; i++) {
     alert("array[" + i + "] ➔ " + A[i] );
}

  // html内の変数 abcを操作
  script.abc = 12345;
  alert("html abc=" + script. abc); //abc=12345

  // html内の関数をcall
  var ret = script.add(10,20);
  alert("html add(10,20)=>"+ ret);

  // htmlのdocumentを操作
  var xyz = document.getElementById("xyz");
  xyz.innerText = "あいえお"; //=> <div id="xyz" name="xyz">あいえお</div>

  // html textテキスト入力値を取得
  var A = document.getElementById("A");
  alert("A="+A.value);
});

d.addEvent("myclose", function(element) {d.close();});

var ans_sum;

function sum(n) {
  var ans = 0;
  for (var i=1; i<=n; i++) ans += i;
  ans_sum = ans;
  return ans;
}

////////////
```

CWebForm作成方法
WebFormの子(Child)Window化

- WebFormは次の３手順(動作)で作成される。
- [1] apOgO->settingCWebForm()関数を使って JSが起動したときに作成されるWebFormのパラメータをsettingする。

```
bool  apOgO->settingCWebForm(name, html, hparent, is_child, initfunc, pdata);  //C++
```

[引数]
name:　作成されるWebFormに名前付け(これを使ってJS側でWebFormオブジェクトが取得できる)
html:　表示するHTMLファイルの名前
hparent: WebFormの親Windowハンドル
is_child:　trueならWebFormをhparentの子Windowとして作成、falseならモードレスダイアログWindowとして作成

initfunc: WebFormが作成された後、callされる関数([2]を参照)
pdata:　データへのポインタ

[返値]
成功したらtrueが返される。失敗したらfalseが返される。

- [2]WebFormが作成された後、callされる関数(settingCWebFormの引数initfunc、引数があればcallされる)

```
int initfunc(hwbf, html, hparent, is_child, pdata);  //C++
```

[引数] hwbf:作成されたWebFormのWindowハンドル、その他はsettingCWebFormの引数
[返値] 初期化に成功したら0を返す、失敗したら0以外(まだ未定だがエラーの種類)を返す。

[3]settingCWebFormで準備してたWebFormをすべて作成し、htmlドキュメントからのイベントを受信する関数を登録する。

```
var webFormS = __newCFebFormS__();  //JS ここでWebFormをすべて作成
if (webFormS.name) {
    var d = webFormS.name;
    d.addEvent("eventName", function() {
        var script = this.script;
        var document =  this.document;
        script.abc = 12345;
        var A = document.getElementById("A");
        alert(A.value);
        ・・・
    );
    ・・・
    ・・・
}
if (webFormS.another) {
    ・・・
    ・・・
}
```

WebPane
C++実装のWindowをHTMLに埋め込む方法

- [1]C++でHTMLに埋め込むWindowの実装(C++)(例MFCで実装)
[クラスビュー]-[MFCクラス]追加-[クラス名](CMyWnd)-[基底クラス](CWndなど)-[完了]

```
// MyWnd.h
class CMyWnd : public CWnd {
   ・・・・
   bool Create(HWND hparent, UINT id=1) {
       return CWnd::Create(nullptr, nullptr, WS_CHILD, CRect(0,0,1,1), CWnd::FromHandle(hparent), id);
    }
    virtual void PostNcDestroy() //CWndから直接継承の場合、Window破棄でオブジェクトがdeleteされないから
    {
        delete this;
    }
    ・・・・
protected:
      DECLARE_MESSAGE_MAP()
  public:
      afx_msg void OnPaint();
      ・・・・
   };

   // MyWnd.cpp
   IMPLEMENT_DYNAMIC(CMyWnd, CWnd
    ・・・・
    BEGIN_MESSAGE_MAP(CMyWnd, CWnd)
        ON_WM_PAINT()
        ・・・・
    END_MESSAGE_MAP()

    void CMyWnd::OnPaint()
    {
        CPaintDC dc(this);
        ・・・・
    }
    ・・・・
    ・・・・
};
```

- [2]HTMLに埋め込むWindowを作成のためにJS側から呼ばれる関数の実装(C++)

```
extern “C”  long __declspec(dllexport)   createMyWnd(long hparent)
   {
       CMyWnd* pmyWnd = new CMyWnd();
       if ( pmyWnd->Create(HWND(hparent)) ) {
           return long(pmyWnd->m_hWnd);
       }else{
           delete pmyWnd;
           return 0;
       }
    }
```

WebPane
C++実装のWindowをHTMLに埋め込む方法（つづき）

- [3]HTMLにC++実装のWindowの埋め込み場所を記述(HTML)

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>CWebForm,WebPane-Test</title>
・・・・
</head>
<body>
・・・・
<object  id="MyWnd"  classid="clsid:F04DFB17-7B74-4B62-90A0-6A38F58C23FE" width="600" height="120"></object>
・・・・
</body>
</html>
```

HTMLに埋め込まれた
WebPane要素

- [4]HTMLにC++Windowを埋め込む（JS)

```
var webFormS = __newCFebFormS__();  //JS ここでWebFormをすべて作成

if (webFormS.name) {
   var d = webFormS.name;
   ・・・・
   d.addEvent("DocumentComplete", function() {
      var document =  this.document;

      setTimeout( function() {
         var wbp = document.getElementById("MyWnd");
         var hpane = wbp.hpane;
         if (!hpane) {
              alert("hpaneはまだ準備できてない!");
              return;
          }

          var {createMyWnd} = DSL.c(`
             long createMyWnd(long hparent);
             #pragma JSEXT dl main
          `);

          var hcwnd = createMyWnd(wbp.hpane) | 0;
          wbp.hcwnd = hcwnd;
       },
       50);
   });
}
```

HTMLに埋め込まれたWebPane要素のプロパティの説明

- WebPane要素だけが備えるプロパティ hpane, hcwnd, dontDestroyCWndについての説明

- [HTML] (準備)
  <object id="MyWnd" classid="clsid:F04DFB17-7B74-4B62-90A0-6A38F58C23FE" width="600" height="120"></object>

- [JS](準備)
  var wbp = document. getElementById("MyWnd");

- [wbp.hpane]プロパティ：
  WebPane要素のWindowハンドル HTMLがdocument内に完全に構成されていないとハンドルOとなってる。構成できていればnonO

- [wbp.hcwnd]プロパティ：
  Windopwハンドルwbp.hpaneのWindowに載せる子Windowハンドル。このプロパティに載せたい子Windowハンドルを代入(set)してください。
  すでにwbp.hcwndにsetしてある場合に新たに上書きでsetしたとき、古いwbp.hcwndのWindowは、wbp. dontDestroyCWndプロパティの値によって処分方法が決まる。

- [wbp. dontDestroyCWnd]プロパティ：
  falseの場合、古いwbp.hcwndのWindowは破棄(destroy)される。
  trueの場合、古いwbp.hcwndのWindowは破棄(destroy)されない。このことによって再利用でき、再度載せることもできる。
  ただし、wbp.hcwndは WebFormが閉じるとき あるいはWebPane要素を削除するとき、自動的に破棄されます。