B(一般)

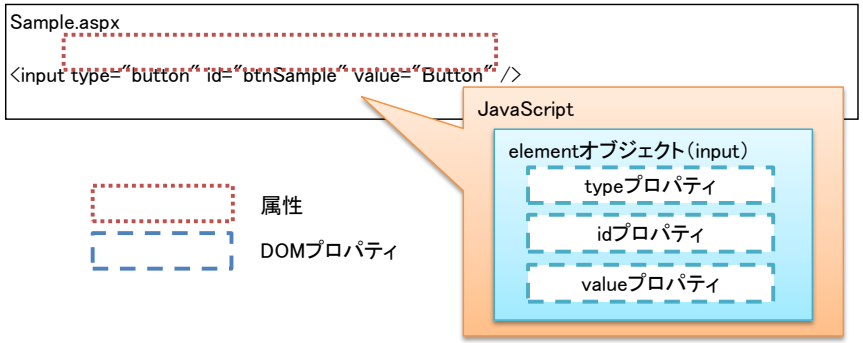
4．Script開発規約

0） 前提事項

1. 用語説明

用語	説明
属性	HTML内に記述されている属性のことを指す。
DOMプロパティ	JavaScript上で扱われるHTMLのオブジェクトはDOMオブジェクトと呼ばれる。 本規約では属性を基に要素のオブジェクトが保持しているプロパティを DOMプロパティ、もしくはDOMオブジェクトのプロパティと表記する。 属性とDOMプロパティの関係については下図参照。

図：属性とDOMプロパティの関係




```
Sample.js

var btn = document.getElementById("btnSample");

var valurAtr = btn.getAttribute("value");

var valueProp = btn.value;
```

2. 特記事項

以下の関数、メソッドを使用する際はFS結果より非互換が確認されているため、本ドキュメントの対象項目を必ず参照すること。

関数／メソッド	本規約対象項目	FS結果サマリ対象項目
getElementById	3. JavaScriptでのコントロールIDの取得方法 6. getElementByIdの使用方法	62. 57.
getAttribute	5. 要素の属性値の取得方法 7. getAttributeの引数記述の厳密化 8. class属性指定のマルチブラウザ対応	51. 52. 88. 87.
setAttribute	8. class属性指定のマルチブラウザ対応	87.
DOMプロパティ	4. JavaScript上の色関連プロパティの形式 5. 要素の属性値の取得方法	49. 54. 51.
createElement	9. createElementを用いた動的な要素の追加方法	50.
document.form	10. form要素の取得方法	56.
childNodes	11. childNodesのマルチブラウザ対応	48.

B(一般)

1） 基本的なコーディングルール

1. **前提**
   クライアントサイドのスクリプト言語はJavaScriptを使うものとする。
   JavaScriptは原則jsファイルとして外部化し、インライン記述や
   aspxのコードビハインドから出力する処理記述については最小限に止める。

2. **変数**
   a. **変数宣言**
      JavaScriptでは変数宣言は必須ではないが、コーディング上の保守性を高めるために
      必ず変数宣言をする。

      記述例）

```
var a = 100;
```

   b. **変数名**
      変数名はcamel方式とする。

      記述例）

```
var objSample = "Sample";
```

3. **関数名**
   関数名はPascal方式とし、原則動詞（Get、Setなど）から始める。

   記述例）

```
function GetValue(){  ...  }
```

4. **コメント**
   jsファイルの内容やHTMLに出力されるJavaScriptは、すべてのソースがクライアント端末に
   ダウンロードされる。
   したがって情報漏洩を未然に防ぐため、コメントの記述は極力行わない。

5. **strictモードについて**
   ECMAScript 5 で採用となったstrictモード(※)については、下記の理由により
   全てのスクリプト処理に対する適用が難しいため導入しない。

   - 2011年7月現在動作確認を行えるブラウザがFireFox（Ver.4以降）しか存在しない。
   - script要素単位で「"use strict";」の宣言が必要なため、既存ASP.NETアプリの修正の場合、
     コードビハインドからスクリプトを出力している箇所にも全て宣言が必要。

   ※補足―ECMAScript 5 とstrictモード
   ECMAScriptはJavaScriptの仕様の基準となる言語で、ECMAScript 5 は現在の最新バージョン。
   strictモードはECMAScript 5 で新たに追加された仕様。
   jsファイルや関数の先頭で「"use strict";」の宣言を行うことで曖昧な記述は実行時にエラーとなる。

6. **フォーマット（体裁）**
   a. **空白行**
      処理ブロックの終了や処理単位ごとに空白行を挿入する。

      記述例）

```
for (i = 0; i < max; i++) {
    .............;        // forループが一つの処理単位
}
                          ← 空白行
while (msg.find("ABC")) {
    .............;        // whileループが一つの処理単位
}
```

   b. **インデント(字下げ)**
      if, while, do, for, switch等の制御文を使用する場合、字下げを行い制御構造の範囲を明確にする。

      記述例）

```
if (条件式) {
    return;
}
```

```
if (条件文) {
    .......;
} else {
    .......;
}
```

```
while (条件文) {
    .......;
    .......;
}
```

```
do {
    .......;
    .........;
} while (条件文);
```

```
for (i = 0; i < max; i++) {
    .......;
    .........;
}
```

```
switch (nVal) {
    case 0:
        .......;
    case 1:
        .........;
    default:
        break;
}
```

2） HTML内でのscript要素の記述方法

■ HTML上にscript要素を配置する場合は、type属性およびlanguage属性の指定は行わないこととする。
・ HTML4.0以降ではlanguage属性は非推奨の扱いとなり、type属性が推奨されている。
また、HTML5標準ではtype属性の既定値が「text/javascript」となる予定である。

記述例）

（1） 外部JavaScriptファイルを読み込む場合

```
<script src="sample.js"></script>
```

（2） インライン記述を行う場合

```
<script>
　（スクリプト記述）
</script>
```

B(一般)

3） JavaScriptでのコントロールIDの取得方法

■ 記述ミスによる障害を防ぐため、JavaScript内にコントロールIDをリテラル（文字列のべた書き）で記述してはならない。
JavaScriptを記述する箇所に応じて以下のどちらかの対応を行う。

a. 外部jsファイルでのコントロールID記述方法

■ コントロールIDが必要な場合は、aspxから関数を呼び出す際にコントロールIDを引数として渡す。
その際は埋め込みコードブロックを使用して、コントロールの「ClientID」プロパティの値を渡すようにする。
・ 上記の方法によってコントロールのIDが変更になった際にjsファイルを修正する必要がなくなり、メンテナンス性が向上する。
また埋め込みコードブロックにaspx上に存在しないコントロールを指定するとコンパイルエラーとなるため、未然にスクリプトエラーを防ぐことができる。

記述例）

Sample.aspx

```
<script src="Sample.js"></script>
<script>
        function SampleRapper() {
                Sample("<%=Me.txtSample.ClientID %>");
                }
</script>
   ...
   <asp:TextBox ID="txtSample" ... ></asp:TextBox>
   <input type="button" id="Button1" value="Button1" onclick="SampleRapper()" />
   ...
   <input type="button" id="Button2" value="Button2" onclick="SampleRapper()" />
```

Sample.js

```
function Sample(arg0) {

        var txt = document.getElementById(arg0);
        ...

}
```

aspx上でも<%= %>タグ（埋め込みコードブロック）を使用することで
コードビハインドと同様にコントロールのプロパティが取得できる。

複数個所から呼び出されることを想定し、<%= %>タグの記述箇所を
少なくするため、script要素の中にラッパー関数を作成する。
各コントロールのからはラッパー関数をイベント属性から呼び出す作りにする。

HTML出力結果

Sample.html

```
<script src="Sample.js"></script>
<script>
        function SampleRapper() {
                Sample("txtSample");
                }
   }
</script>
   ...
   <input name="txtSample" type="text" id="txtSample" />
   <input type="button" id="Button1" value="Button" onclick="SampleRapper()" />
   ...
```

b. コードビハインドから出力するJavaScriptでのコントロールID記述方法

■ コードビハインドからJavaScriptを出力する場合は以下のように記述する。

B(一般)

記述例)

Sample.aspx

```
...
<asp:TextBox ID="txtSample" runat="server"></asp:TextBox>
<input type="button" id="Button1" value="Button" onclick="Sample()" />
...
```

Sample.aspx.vb

```
''' <summary>
''' クライアントにスクリプトを出力
''' </summary>
''' <remarks></remarks>
Private Sub RenderClientScript()

    'スクリプト組み立て
    Dim scr As StringBuilder = New StringBuilder()
    scr.AppendLine("function Sample(){{")
    scr.AppendLine("    var txt = document.getElementById('{0}');")
    ...
    scr.AppendLine("}}")

    'String.Formatメソッドでコントロールの ClientIDを埋め込む
    Dim script As String = String.Format(scr.ToString(), Me.txtSample.ClientID)

    'レンダリング
    ClientScript.RegisterClientScriptBlock(Me.GetType(), "RenderFunction", script, True)

End Sub
```

String.Formatメソッドの引数となる文字列内で中カッコ("{"、"}")を使用する際は、2回続けて記述してエスケープする必要がある。

B(一般)

HTML出力結果

Sample.html

```
<script>
//<![CDATA[
function Sample(){
    var txt = document.getElementById('txtSample');
    ...
}
//]]>
</script>

    ...
    <input name="txtSample" type="text" id="txtSample" />
    <input type="button" id="Button1" value="Button" onclick="Sample()"  />
    ...
```

参考：誤ったコントロールIDの記述例

⇒ id属性の値をリテラルで記述すると、コントロールのid属性の値が誤っていた場合に気付きにくい。
また、コントロールIDが変更になった場合に修正が必要になるためメンテナンス性も低い。

Sample.aspx

```
<script src="Sample.js"></script>
        ...
        <asp:TextBox ID="txtSample" ... ></asp:TextBox>
        <input type="button" id="Button1" value="Button" onclick="Sample()"  />
        ...
</asp:Content>
```

Sample.js

```
function Sample() {

        var txt = document.getElementById("txtSample01");
        ...

}
```

getElementByIdの引数が実際のidと異なっているため
スクリプトを実行しても動作しないが、
デバッグ時にコンパイルエラーなどが出ないため
実行するまで気付きにくい。

B(一般)

4） JavaScript上の色関連プロパティの形式

■ JavaScript上で色関連のCSSプロパティを扱う際は、必ず「rgb(255,255,255)」形式を用いる。
これに伴い、cssファイルで色関連の値を設定する際にも混乱を避けるため原則「rgb(255,255,255)」形式を用いる。
また、カラーネーム形式（red、blueなど）はブラウザごとに実際の表示が異なる場合があるため**原則使用しない。**
・ cssファイルで「#ffffff」形式を用いて設定を行っていても、そのプロパティをJavaScriptで取得すると「rgb(255,255,255)」形式に変換されて取得される。
そのためJavaScriptで現在の色と変更する色の判定などを行う場合には、同形式での判定を行うため「rgb(255,255,255)」の形式に統一する。

記述例）

Sample.js

```
var obj = document.getElementById("tr01");
var oldColor = obj.style.backgroundColor;
var setColor = "rgb(255,255,204)";
if ( setColor == oldColor )          ←Trueと判定される
{
   …
}
```

Sample.aspx

```
<tr  id="tr01" class="cls"… > … </tr>
```

Sample.css

```
.cls {
        background-color:rgb(255,255,204);
}
```

参考1：色関連のプロパティ設定の際は以下を用いる。（現行営業系共通部品に基づく）

no	#rgbｺｰﾄﾞ	rgbｺｰﾄﾞ	ｾｰﾌｶﾗｰ
1	#000000	rgb(0,0,0)	black
2	#004080	rgb(0,64,128)	-
3	#000080	rgb(0,0,128)	-
4	#0000FF	rgb(0,0,255)	blue
5	#0040FF	rgb(0,64,255)	-
6	#2060FF	rgb(32,96,255)	-
7	#4080FF	rgb(64,128,255)	-
8	#6080FF	rgb(96,128,255)	-
9	#00FFFF	rgb(0,255,255)	cyan
10	#80FFFF	rgb(128,255,255)	-
11	#F0F8FF	rgb(240,248,255)	aliceblue
12	#00A000	rgb(0,160,0)	-
13	#339933	rgb(51,153,51)	-
14	#66CC33	rgb(102,204,51)	-
15	#99CC66	rgb(153,204,102)	-
16	#99FF99	rgb(153,255,153)	-
17	#F0F0B0	rgb(240,240,176)	-
18	#FFFFCC	rgb(255,255,204)	-
19	#FFFFE0	rgb(255,255,224)	-
20	#F0F0FF	rgb(240,240,255)	-
21	#C0C0C0	rgb(192,192,192)	silver
22	#E0E0FF	rgb(224,224,255)	-
23	#FFFFFF	rgb(255,255,255)	white
24	#FF0000	rgb(255,0,0)	red

参考2：IE7、IE9でのstyle.backgroundColorプロパティ取得結果検証

	JavaScriptでの取得結果	
	IE7.0標準	IE9.0標準

B(一般)

CSSの設定値	#ffffff	#ffffff	rgb(255,255,255)
	rgb(255,255,255)	rgb(255, 255, 255)	rgb(255, 255, 255)
	色(white,…)	色（white,…）	色（white,…）

[IE7.0標準]

初期設定値	取得結果
[backgroundColor:#ffffff]	#ffffff
[backgroundColor:rgb(150,150,150)]	rgb(150,150,150)
[backgroundColor:cyan]	cyan

[IE9.0標準]

初期設定値	取得結果
[backgroundColor:#ffffff]	rgb(255, 255, 255)
[backgroundColor:rgb(150,150,150)]	rgb(150, 150, 150)
[backgroundColor:cyan]	cyan

5) 要素の属性値の取得方法

■ 要素のオブジェクトから属性の値を取得する場合には、以下に従う。
① HTMLとして出力されている文字列を参照する場合には「getAttribute」を使用する。
② ①以外の場合は「element.(属性名)」形式でDOMオブジェクトのプロパティ(※)から取得する。
スクリプトによって値が動的に変更される可能性がある値を取得する場合は、原則②を使用する。

※補足一属性とDOMオブジェクトのプロパティ
「getAttribute」はHTML要素に設定されている属性を取得する。
DOMプロパティを参照した場合は要素のオブジェクトがプロパティとして保持している値を取得する。
これらは別のものとして保持されており、属性と同名のDOMプロパティを参照した場合でも
取得される値はHTMLに設定された形式とは異なっている場合がある。(下記記述例参照)
そのため本規約では上記①、②の切り分けを行った。
ただし、IE7まではこの二つを同様のものとして扱う不具合があり、
DOMプロパティと「getAttribute」で同じ内容が取得されてしまうことに注意する。

属性とDOMプロパティの関係については
「前提事項 4.用語解説 参照

参考: getAttributeとDOMプロパティの取得結果比較

記述例)disabled、readonlyなどの論理属性

Sample.aspx

```
<input type="text" id="txt01" name="txt01" class="cls01" disabled="disabled" />
```

Sample.js

```
var txt = document.getElementById("txt01");
var disabled = txt.getAttribute("disabled");      //①HTML上の値"disabled"が取得される
var disabled = txt.disabled;                      //②Boolean形式の"true"が取得される
```

記述例)onclick、onblurなどのイベント属性

Sample.aspx

```
<input type="button" id="btn01" onclick="Sample()" value="Sample" ... />
```

Sample.js

```
var btn = document.getElementById("btn01");
var func = btn.getAttribute("onclick");           //①HTML上の値"Sample()"が取得される
var func = btn.onclick;                           //②関数オブジェクト(function Sample(){ ... })が取得される
```

記述例)href、srcなどのURL関連属性
⇒ 「D://sample」がルートディレクトリとする

Sample.aspx

```
<a href="index.html" id="link"  ... >index</a>
```

Sample.js

```
var link = document.getElementById("link");
var url = link.getAttribute("href");              //①HTML上の値"index.html"が取得される
var url = link.href;                              //②絶対パス"file:///D://sample/index.html"が取得される
```

B(一般)

6） getElementByIdの使用方法

a. 引数に設定可能な属性の厳密化

■ getElementByIdで要素を取得する場合の引数は、対象の要素のid属性のみとする。

・ IE7以前はname属性を引数に設定しても取得可能だったが、IE8以降はid属性のみが可能となった。

記述例）

Sample.aspx

```
<a id="targetId" name="targetName"… > … </a>
```

Sample.js

```
var obj = document.getElementById("targetId");
```

参考：IE7、IE9でのgetElementById動作検証（id属性とname属性による取得結果差異）

Sample.aspx

```
<a id="targetId" name="targetName" href="http://www.XXXXXXXXXXX.co.jp/index.html">
```

Sample.js

```
//①
var obj1 = document.getElementById("targetId");
var sHref1Attr = obj1.getAttribute("href");

//②
var obj2 = document.getElementById("targetName");
var sHref2Attr = obj2.getAttribute("Href");
```

結果

[IE7.0標準]

```
target id : http://www.XXXXXXXXXXX.co.jp/index.html      … ①
target name : http://www.XXXXXXXXXXX.co.jp/index.html    … ②
```

[IE9.0標準]

```
target id : http://www.XXXXXXXXXXX.co.jp/index.html      … ①
target name : null                                        … ②
```

B(一般)

b. 引数の記述の厳密化

■ getElementByIdで要素の取得を行う際の引数に設定するid属性は、HTML要素内に記述されているものと大文字小文字まで一致させる。

・ IE7以前はgetElementByIdの引数は大文字小文字を区別していなかったが、IE8以降は区別するように変更された。

記述例)

Sample.aspx

```
<a id="targetId" name="targetName"... > ... </a>
```

Sample.js

```
var obj = document.getElementById("targetId");
```

参考:IE7、IE9でのgetElementById動作検証(大文字小文字の区別による取得結果差異)

Sample.aspx

```
<a id="targetId" name="targetName" href="http://www.XXXXXXXXXXXX.co.jp/index.html">
```

Sample.js

```
var obj = document.getElementById("TargetID");
var sHrefAttr = obj.getAttribute("href");
```

結果

[IE7.0標準]

```
target id : http://www.XXXXXXXXXXXX.co.jp/index.html
```

[IE9.0標準]

```
target id : null
```

B(一般)

7） getAttributeの引数記述の厳密化

■ getAttributeの引数の属性名は小文字で記述する。
・ W3C勧告の記述標準として属性名は小文字で表記するため。
※ IE独自実装の大文字小文字の区別を制御するための第2引数は使用しない。

記述例）

```
Sample.js

var val = obj.getAttribute("value");
```

8） class属性指定のマルチブラウザ対応

■ IE7以前とIE8以降それぞれで同様の結果を得るため、getAttributeおよびsetAttributeによるclass属性の取得および設定時は「class」「className」それぞれを引数とした記述を行い、それらを論理和演算子(||)で結ぶ。
- IE7以前はgetAttributeメソッド等でclass属性を扱う際、属性名を「className」としなければならなかった。
  IE8以降では、属性名を「class」と指定するように変更され、「className」による指定は無効のためnullが戻されてしまう。

記述例）

```
（1） getAttribute
   ⇒var attr = ～.getAttribute("className") || ～.getAttribute("class")

（2） setAttribute
   ⇒eval( ～.setAttribute("className", "xxxx") || ～.setAttribute("class", "xxxx") )
```

setAttributeメソッドの場合は「eval」メソッドを用いる。
これにより各ブラウザで有効な方の記述が実行される。

参考：IE7、IE9でのgetAttribute動作検証

<取得結果一覧>

	IE7以前	IE8以降
getAttribute('className')	○	×
getAttribute('class')	×	○

Sample.aspx

```
<a class="SampleClass" id="link" href="http://www.xxx.co.jp/index.html">
```

Sample.js

```
var obj = document.getElementById("link");

//①
var sClass1Attr = obj.getAttribute("className");
//②
var sClass2Attr = obj.getAttribute("class");
```

B(一般)

結果

[IE7.0標準(getAttribute例)]

```
className value: SampleClass  … ①
class value: null             … ②
```

[IE9.0標準(getAttribute例)]

```
className value: null     … ①
class value: SampleClass  … ②
```

9） createElementを用いた動的な要素の追加方法

■ createElementでは要素の生成のみを行い、setAttributeメソッドを用いて属性の付与を行う。
・ IE9.0標準ﾓｰﾄﾞでは、createElementの引数が要素名のみとなったため。

記述例）

```
var elm = document.createElement("input");
elm.setAttribute("type", "text");
elm.setAttribute("id", "txtSample");
elm.setAttribute("name", "txtSample");
elm.setAttribute("value", "sample");
eval(elm.setAttribute("className", "cls") || elm.setAttribute("class", "cls"));
var obj = document.getElementById("xxxx");
obj.appendChild( elm );
```

class属性を付与する際の記述方法の詳細は
「7. class属性指定のマルチブラウザ対応」参照

B(一般)

10）form要素の取得方法

■ form要素を取得する際は「document.forms[0]」を用いる。

- 「document.formName」形式でname属性を用いる方法については、.NET Framework 4 でname属性が付与されなくなるため使用しない。
  getElementByIdメソッドでid属性を用いる方法については、id属性の値が.NET Frameworkに依存するため将来性を考慮し使用しない。
  form要素を複数配置した場合「forms」のインデックスが増える問題については、
  ASP.NETでは"runat=server"を付与したform要素を2つ以上配置するとコンパイルエラーとなるため通常問題とならない。

記述例）

Sample.aspx

```
<form id="form1" runat"=server>
```

Sample.js

```
var frm = document.forms[0];
```

B(一般)

11）childNodesのマルチブラウザ対応

■ マルチブラウザでの動作を保証するため、childNodesを使用する際には必ず基盤共通関数を併用する。
・ childNodesメソッドはIEのバージョンによって戻り値に空白ノード（改行文字、タブ文字など）が含まれるか否かが異なっているため、
共通関数内で空白ノードを取り除くことでどのブラウザでも同様の挙動が行えるようにする。

共通関数の詳細

関数名	KibanGetFillNodes
JSファイル	MYEFKibanCommonScript.js
引数	NodeList（"element.childNodes"の戻り値）
戻り値	引数のNodeListから空白テキストノードを取り除いたNodeListを返却

記述例）

Sample.js

（1） 配列として戻り値を取得している箇所
```
var children = KibanGetFillNodes(obj.childNodes);
```

（2） 配列のインデックスを直接取得している箇所
```
var child = KibanGetFillNodes(obj.childNodes)[0];
```

共通関数の内容

```
function KibanGetFillNodes(args)        //element.childNodesを引数に指定
{

  //戻り値
   var children = new Array();
   var val = "";
   var max= args.length;

   for(var i = 0;i < max ; i++) {

     //文字列ノードかどうか判定
      if(args[i].nodeName == "#text" && args[i].nodeType == 3) {
          val = args[i].nodeValue;

        //文字列ノードの場合、区切り文字以外の文字（全角スペース含む）がある場合のみ返却対象とする
         if(val.match(/[^¥s]/g) || val.match(/[　]/g)) {
             children.push(args[i]);
         }
      }  else {

        //文字列ノード以外はそのまま返却
         children.push(args[i]);

     }
   }
   return children;  //空白ノードを除去した配列を返却
```

B(一般)

12）日付の年情報の取得方法

■ 日付の年情報を取得する際は、西暦年が戻り値になる「getFullYear」メソッドを用いる。

・「getYear」メソッドの戻り値がIE8以前は西暦年だったが、IE9では1900年からの経過年数に変更されたため、日付の年情報取得には「getYear」メソッドを使用しない。

記述例）

Sample.js

```
var myDate = new Date();
var year = myDate.getFullYear();
```

B(一般)

13）toFixedメソッドを用いた切り捨てした数値の取得方法

■ toFixedメソッドを使用して切り捨てした数値を取得する場合、toFixedメソッド適用前に以下の方法で数値の編集を行う。
① 数値が整数の場合は、四捨五入による影響がないためそのままtoFixedメソッドを適用する。
② 数値が小数を含む場合、四捨五入を行う位置の数値を切り取りtoFixedメソッドを適用する。

・ toFixedメソッドではIE8以前では切り捨てで値が丸められ、IE9では四捨五入で値が丸められるので、
toFixedメソッド使用前に四捨五入による値の切り上げが行われないよう数値の編集を行う必要がある。

記述例）

Sample.js

```
var num = 0.09;
var trg = 1;

var str = num.toString();
var idx = str.indexOf('.');

if(idx !== -1){
        var tmp = str.substring(0, idx + trg + 1);
        num = parseFloat(tmp);
}

var value = num.toFixed(trg);
```

B(一般)

14）配列末尾の空要素の設定方法

■ 配列末尾に空要素を設定する場合は、未定義(undefined)で設定を行う。
- IE8以前では空要素は配列のlengthプロパティの値に含まれていてカウントされていたが、IE9ではlengthプロパティの値に含まれなくなったため、lengthプロパティの値に含まれる未定義(undefined)で設定を行う。

記述例）

Sample.js

```
var arr = [1, 2, undefined];
```

B(一般)

15）正規表現によるマッチング結果の使用方法

■ 正規表現によるマッチング結果を使用する場合は、未定義(undefined)を空白文字列(″″)に置き換えてから使用する。

・ IE8以前では正規表現による文字列検索を行いマッチする文字列がない場合は空文字列（″″）が取得されていたが、IE9では未定義（undefined）が取得されるため、マッチング結果の判定を行い、未定義を空白文字列に置き換える必要がある。

記述例）

Sample.js

```
var arr = /ab|(c)/.exec("ab");
for(i=0; i < arr.length; i++){
        if(arr[i] == undefined){
                a = a + "" +",";
        }else{
                a = a + arr[i] +",";
        }
}
```

B(一般)

16）dontenum属性を持つプロパティのfor inループを用いた列挙方法

■ オブジェクトの継承を行ってdontenum属性を持つプロパティをオーバーロードした場合、「for in」ループを用いたプロパティの列挙をする際は、親クラスでdontenum属性を持っていたプロパティを除外する。

・ IE9ではdontenum属性を持つプロパティをオーバーライドした際に、dontenum属性が継承されない仕様になったため、「for in」ループを使用してプロパティを列挙した場合、親クラスがdontenum属性を持つプロパティも列挙されてしまう。

※ dontenum属性を持つプロパティの数は非常に多いため、対象となるプロパティはIE9とIE8以前での処理結果を比較して洗い出す。

記述例）

Sample.js

```
var obj={valueOf:0, toString:1, foo:2};
var a="";
for (var p in obj){
        if(p=="valueOf" || p=="toString"){
                continue;
        }
        a += p + ",";
}
```

valueOfとtoStringの場合は、dontenum属性を持っているので
continueで処理を抜ける

17）制御文中のセミコロンの記述方法

■ 一つの制御文中ではコードブロック({ })の後ろと予約語の間にセミコロン(;)を記述しない。
- 「if() {} ; else {}」や「do {} ; while()」のように制御文の途中にセミコロンが記述されている場合、IE9ではセミコロンの位置でステートメントが終了されたとみなされ構文エラーとなってしまう。

記述例）

- 構文エラーとなる記述

Sample.js

```
if(boolValue)
{return true};
else{return false};
```

Sample.js

```
var a = 0;
do{a += 1};
while(a < 10);
```

- 正しい記述

Sample.js

```
if(boolValue)
{return true}
else{return false};
```

Sample.js

```
var a = 0;
do{a += 1}
while(a < 10);
```

B（一般）

18）JavaScriptでのユーザーエージェント文字列判定を用いた処理の記述方法

■ navigatorオブジェクトから取得したユーザーエージェント文字列を元に処理を分岐する場合、
IE9に対応するユーザーエージェント文字列の判定を行い、対応する処理を記述する。
・ IE9でユーザーエージェント文字列が追加されたため、IE9上での動作を明記する。

記述例）

Sample.js

```
var ua = navigator.userAgent;
if(ua.indexOf("MSIE 7.0") != -1 ){
        ...
}else if(ua.indexOf("MSIE 8.0") != -1 ){
        ...
}else if(ua.indexOf("MSIE 9.0") != -1 ){
        ...
}
```

※ 正規表現を用いてバージョン番号まで判定を行う場合には、桁数に影響されないように以下のような記述を行うこと。

Sample.js

```
if((navigator.userAgent.match(/MSIE ([¥d.]+);/),RegExp.$1) >= 7.0){
    ...
```

数字とピリオドを組み合わせた
任意の文字数で検索されるので、
バージョンの桁数に左右されずに判定が可