初めての JavaScript(関数)

[初めての JavaScript]

・関数を作成するには、主に 3 つの方法がある。
1. 宣言型 (静的、static) 関数
2. 無名関数
3. リテラル

■宣言型の関数

・最もよく使われる関数。
・ロード時に解析され、呼び出されるたびに使用される

```
function 関数名 ( 引数 1, 引数 2, ... ){
}
```

・基本データ型 (string、boolean、number) は値渡し
・オブジェクトは参照渡し

■無名関数

・関数はオブジェクト。
・コンストラクタを使用して生成したり、変数に代入したりできる。
・以下は無名関数または匿名関数と呼ばれる
・動的に生成され、そのたびに解析される

```
var showMsg = new Function("msg", "alert(msg)");
showMsg("hello");
```

・コンストラクタを使用する構文は以下

```
var variable = new Function(" 引数 1", " 引数 2", ・・・ , " 関数本体 ");
```

■関数リテラル

・宣言的な関数と無名関数の両方の特徴を持つ
・他のオブジェクトと同様に関数を作成したり、変数に代入したりするためにコンストラクタを使用する必要はなく、関数リテラルを使用できる。

```
var func = function( 引数 ){
}
```

・式の一部として生成されるため、関数式とも呼ばれる
・構文解析が行われるのは 1 回のみ
・無名でないものもある

```
var func = function funcName() {
};
```

・ただし、ここでつけた名前は、関数の内部からしかアクセスできない。

■コールバック関数

・Array オブジェクトには、以下のようなコールバック関数がある
・element,index,array の引数を持ち、元の配列には影響を及ぼさない

コールバック関数	内容
filter	ある値を満たさない限りは要素を加えない
forEach	指定した関数を各要素に適用
every	どれかひとつが false を返すまで関数を実行
map	全要素に対して関数を実行し結果から新しい配列を作成
some	every の逆。どれかひとつが true を返すまで関数を実行

filter

```
function isNum(element, index, array) {
  return ((new String(element).match(/[0-9]+/)) != null);
}

var elm = ['a','1','b','2','c','3'];
var num = elm.filter(isNum);
var s="";
for (var i=0; i<num.length; i++) {
    s += ((i > 0)?",":"") + num[i]
}
alert(s); // "1,2,3"
```

forEach

```
var msg = "";
function concat(element, index, array) {
    msg += element;
}

['a','b','c'].forEach(concat);
alert(msg); // "abc"
```

every

```
function isNum(element, index, array) {
  return ((new String(element).match(/[0-9]+/)) != null);
}

alert(['1','2','3'].every(isNum)); // true
alert(['1','a','3'].every(isNum)); // false
```

map

```
function toUpper(element, index, array) {
    return (new String(element).toUpperCase());
}

var uc = ['a','b','c'].map(toUpper);
var s = "";
for (var i=0; i<uc.length; i++) {
    s += uc[i];
}
alert(s); // "ABC"
```

■再起関数

・関数リテラルで、関数に名前をつけておくと再起呼び出しが可能になる。

■クロージャ

・入れ子になった内側の関数リテラル

```
function outer( 引数 ) {
  function inner( 引数 ) {
  }
}
```

・上記の場合、inner 関数は、outer という名前の関数のコンテキストでのみ使用される。
・inner から、outer のすべての変数にアクセスできるが、outer からはできない。
・他の関すが inner を呼び出すこともできない。

```
function outer() {
    var outer_msg = "outer";
    function inner() {
        var inner_msg = "inner";
        alert("call outer from inner : " + outer_msg);
    }
    inner();
    alert("call inner from outer : " + inner_msg); // error
}

outer();
inner(); // error
```

・ただし、内側の関数がリテラルとして作成され、呼び出し側に戻り値として返されれば呼び出される可能性がある。

```
function calc(exper) {
    function exec(x, y) {
        return eval(new String(x) + exper + new String(y));
    }
    return exec;
}

var add = calc("+");
var sub = calc("-");

alert(add(1,2)); // 3
alert(sub(1,2)); // -1
```

■オブジェクトとしての関数

・コンストラクタを使ってできるものなら何でも、プロパティをメソッドを持てる。関数も同様

```
function func(x,y,z) {
    for(var i=0; i<func.length; i++) {
        alert(func.arguments[i]);
    }
}

func(3,2,1);
```
