脱初心者のために
これだけは知っておきたい
JavaScriptネタ

Tsuyoshi Akase

HTML5+α at FUKUOKA

福岡Haxe勉強会
feat.
HTML5+α @福岡
第0x00回

アジェンダ

1. 自己紹介
2. 関数
3. レキシカルスコープ
4. スコープチェイン
5. クロージャ
6. JavaScriptのスコープ
7. Array.length
8. コンストラクタ／プロトタイプ

1. 自己紹介

Tsuyoshi Akase

Groovy Mobile Inc.

We are creating a CMS. (PHP / JavaScript)

akase244 (Twitter / Facebook)

Fukuoka.php / HTML5+α @Fukuoka

みなさん、JavaScriptは毎日さわってますか？

当然さわってますよね。

ところで、10年くらい前の JavaScriptって

```
<script>
   function blink() {
      if (document.getElementById('blink').style.display == 'none') {
         document.getElementById('blink').style.display = 'block';
      } else {
         document.getElementById('blink').style.display = 'none';
      }
      setTimeout('blink()', 500);
   }
</script>
</head>
<body onload="blink();">
<div id="blink" style="color:red;">blink</div>
</body>
```

ところが

Google Map以降

劇的に変わりましたよね？

最近は

Ajax JSON jQuery zepto.js Node.js Backbone.js QUnit Jasmine AngularJS Sencha Titanium PhoneGaP three.js RequireJS Knockout.js enchant.js ...

サーバーからクライアント、 Web、
ゲーム、3D、テストにいたるまで、
正直、もうついていけません。

そんなことも言ってられないですし、
今日は、奥深い JavaScriptの世界に触れて、
初心者という殻を破って行こうじゃありませんか。

さて、本題入りますか

先ほど、たくさんの JavaScriptの
ライブラリや技術などを列挙しましたが、
そもそもその前に、JavaScriptが持つ
様々な特徴についてしっかりと理解されていらっ
しゃいますか？

自分自身、数日前まで全然
理解していませんでした。。。

なので、今日は皆さんと一緒に JavaScriptというヤツを紐解いていってやろうと思います。
とは言え、時間も限られていますので、ここではいくつかピックアップして、説明を進めていきたいと思います。

いい加減、本題に入ります

まずは、関数の話から

2. 関数

JavaScriptの関数は再代入可能（変数として扱える）なオブジェクトで、
関数名は関数オブジェクトを代入するための変数名として利用可能です。

```
function hello() {
    alert("こんにちは");
}
var copyHello = hello;
copyHello(); // こんにちは
```

無名関数（匿名関数）：宣言と代入を同時に行う

```
var hello = function() {
    alert("こんにちは");
}
hello(); // こんにちは
```

2. 関数

高階関数：関数は引数にすることも可能です。

```
function hello(fn) {
    fn();
}
hello(function() {
    alert("こんにちは");
});
```

2. 関数

高階関数：関数は戻り値にすることも可能です。

```
var counter = function() {
    var cnt = 0;
    return function() {
        return cnt++;
    };
}
var callCounter = counter();
alert(callCounter()); // 0
alert(callCounter()); // 1
alert(callCounter()); // 2
```

今、何かおかしくなかった！？

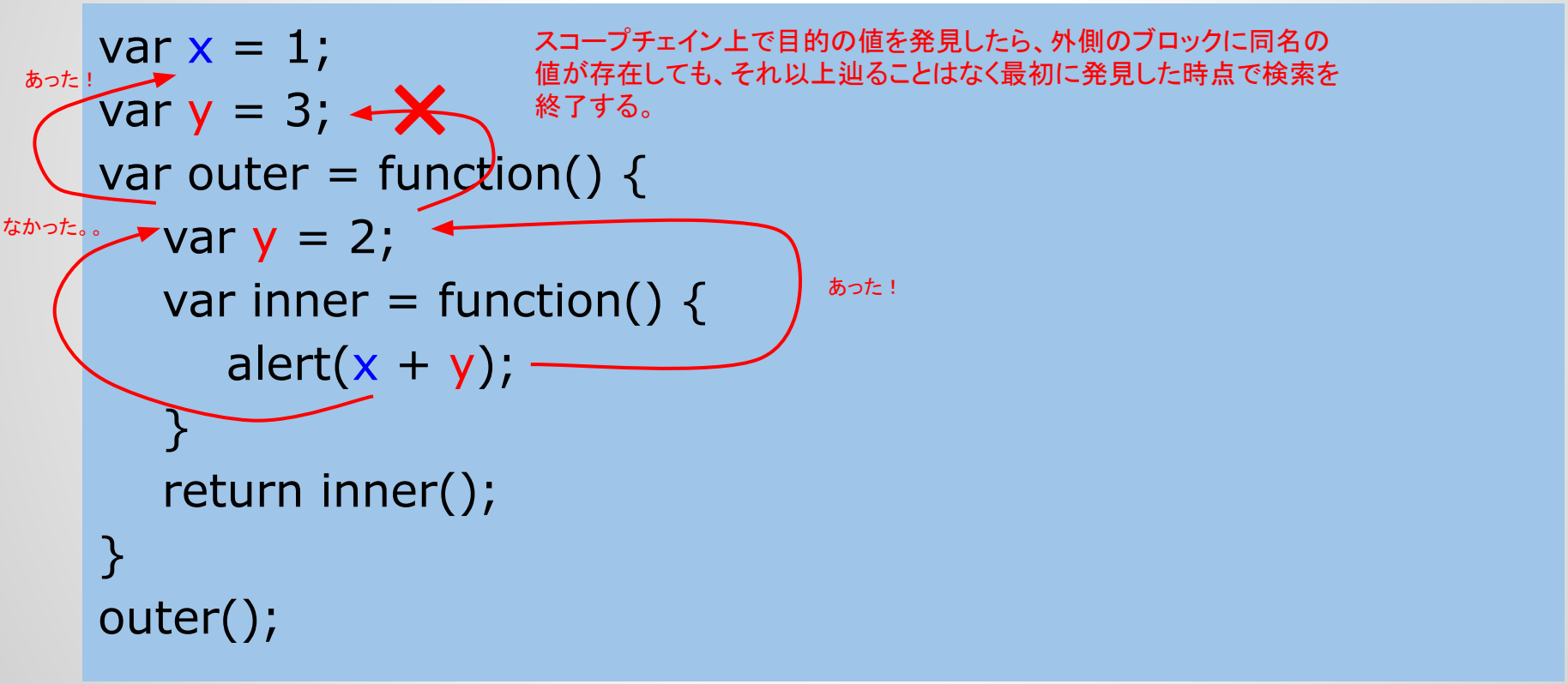
3. レキシカルスコープ

関数ブロック内で宣言した変数は外側のブロックからは参照できませんが、
内側のブロックから外側のブロックの変数は参照できます。

```
var counter = function() {
    var cnt = 0;
    return function() {
        return cnt++;
    };
}
var callCounter = counter();
alert(callCounter()); // 0
alert(callCounter()); // 1
alert(callCounter()); // 2
```

4. スコープチェイン

変数オブジェクトを外側のブロックに向かって探していく仕組みのことを、スコープチェインと呼びます。



いや、だからさっき
何かおかしくなかった！？

5. クロージャ

ローカル変数を参照し続けることができる関数のことをクロージャと呼びます。
クロージャの条件：

- 外側の関数のスコープ内で変数が定義されている。
- 外側の関数のスコープ内に関数を作り、その関数内関数から上記の変数を参照する。

```
var counter = function() {
    var cnt = 0;
    return function() {
        return cnt++;
    };
}
var callCounter = counter();
alert(callCounter()); // 0
alert(callCounter()); // 1 <= 前回の結果に加算されている
alert(callCounter()); // 2 <= 前回の結果に加算されている
```

6. JavaScriptのスコープ

- グローバルスコープ
- ローカルスコープ(関数スコープ)
- evalスコープ

```
var foo = 0; // グローバルスコープ
alert(foo); // 0
var myFunc = function() {
    var foo = 1;  // ローカルスコープ(myFuncのスコープ)
    alert(foo); // 1

    var myFuncNest = function() {
        var foo = 2; // ローカルスコープ(myFuncNestのスコープ)
        alert(foo); // 2
    }();

    var myFuncNest2 = function() {
        alert(foo); // 1（スコープチェイン）
    }();
}();

eval('var foo = 3; alert(foo);'); // evalスコープ
```

6. JavaScriptのスコープ

- **JavaScriptはブロックスコープ（条件文やループ文の範囲）を持たない。**
- **varで定義しなかった場合は、たとえ関数内でもグローバルスコープになる。**

```
var foo = 0; // グローバルスコープ
if (true) {
    foo = 1; // グローバル変数fooを上書き
    for (var i = 2; i < 4; i++) {
        foo = i; // グローバル変数fooを上書き
        alert(foo); // 2, 3
    }
}
```

```
var foo = function() {
    var bar = function() {
        hoge = 4;  // グローバルスコープ（varで定義していない）
    }();
}();

alert(hoge); // 4 (※varで定義すると「Uncaught ReferenceError: hoge is not defined」)
```

関数のまとめ

- 関数は再代入可能なオブジェクト。
- 無名関数は宣言と代入が同時にできる。
- 関数は引数に指定することができる。
- 関数は戻り値に指定することができる。
- ブロックの内側から外側へ向かって、変数を参照することができ（レキシカルスコープ）、かつ、その動きは連鎖する（スコープチェイン）。
- ローカル変数を参照し続けることができる関数のことをクロージャと呼ぶ。
- JavaScriptには3種類のスコープが存在し、ブロックスコープは持たない。

次は、配列の話

7. Array.length

Array.lengthは格納されている個数ではなく、 インデックス最大値＋1 。

```
var arr = new Array(3);
arr[0] = 0;
arr[1] = 1;
arr[100] = 2;
alert(arr.length); // 101
arr["a"] = "a"; // 添え字として無効な値は、arr.lengthに影響を与えない
arr["b"] = "b"; // 添え字として無効な値は、arr.lengthに影響を与えない
alert(arr.length); // 101
arr[101] = 3;
arr[102] = 4;
alert(arr.length); // 103
arr.length = 101; // セットした値の大きさまで切り詰められる(lengthは書き込み可能)
alert(arr[100]) // 2
alert(arr[101]) // undefined
alert(arr["a"]) // "a"はクリアされない
alert(arr["b"]) // "b"はクリアされない
```

問題

次の文字列の 左から3文字目のみを抽出 したい場合、
どういった方法がありますか？

```
var hoge = '1234567890';
```

配列のちょっと変わった使い方

文字列に対して、配列のようなアクセスが可能。

```
var hoge = '1234567890';
alert(hoge.charAt(2)) // 3
alert(hoge.substr(2,1)) // 3
alert(hoge[2]); // 3 ＜＝このような指定が可能
```

小ネタ（console.dir）

**console.logはオブジェクトの構造が 1階層の場合、展開して表示しないが、
console.dirは階層構造で表示する。（※ただし IEは除く）**

```
var arr = [1,2,3];
arr["a"] = "a";
arr["b"] = "b";
console.log(arr);
[1, 2, 3, a: "a", b: "b"]
console.dir(arr);
▼Array[3]
    0: 1
    1: 2
    2: 3
    a: "a"
    b: "b"
    length: 3
    ▼__proto__: Array[0]
```

配列のまとめ

- Array.lengthは格納されている個数とイコールではない。
- 連想配列を定義した場合、lengthプロパティに影響を与えない。
- Array.lengthは書き込みが可能で、指定した値まで切り詰めることができる。
  ※ただし、連想配列の領域は無視される。
- 文字列に対して、配列のようなアクセスが可能。

8. コンストラクタ／プロトタイプ

JavaScriptは、
「プロトタイプベースのオブジェクト指向言語」
と呼ばれています。

「オブジェクト指向言語」と聞くと、プログラムに慣れた人であれば、すぐに疑問が沸くはずです。

先生、JavaScriptで「クラス」を作成することはできますか？

8. コンストラクタ／プロトタイプ

この処理を実行すると。。。

```
var class = "class";
var Class = "Class";
```

「Uncaught SyntaxError: Unexpected reserved word」というエラーが発生します。
つまり、JavaScriptでは「class」、「Class」は予約語として扱われます。

しかし、単語自体は予約語になっているものの、JavaScript自体の機能として、クラスを実装する機能は提供されていません。
では、クラスを作ることはできないのかというと、そうでもなさそうで、ググると「それっぽいもの」が作れるという情報がヒットします。

8. コンストラクタ／プロトタイプ

new演算子を指定することで、コンストラクタ（クラスっぽいもの）が作れます。
コンストラクタは、新規作成されたオブジェクトを初期化するメソッドです。

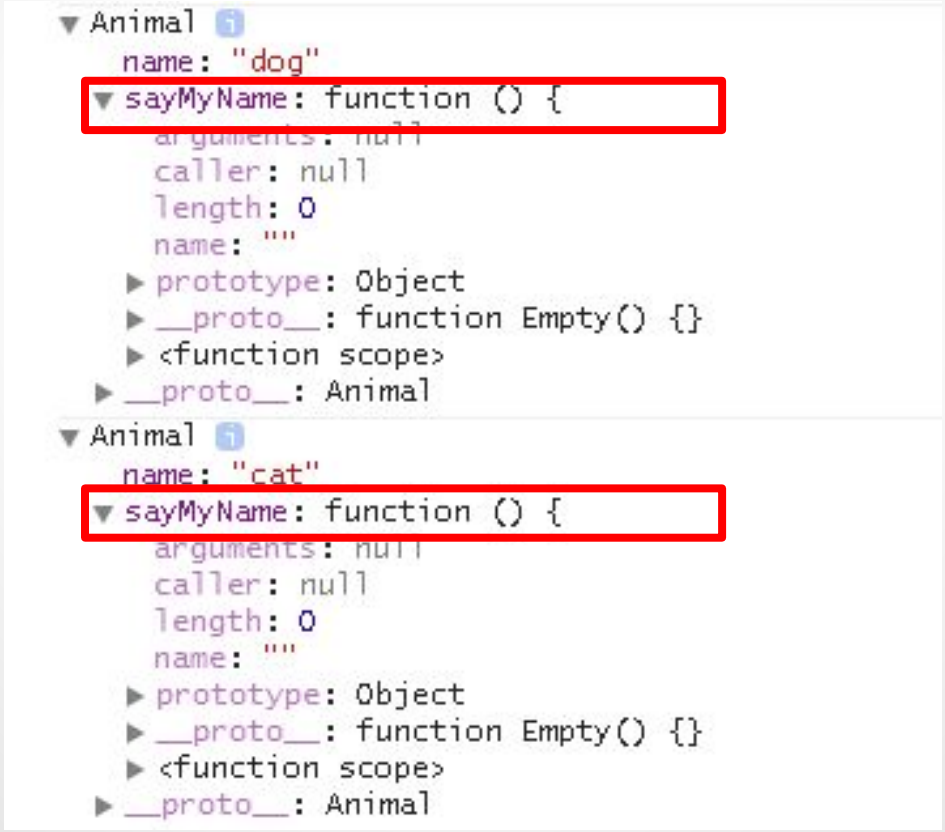
```
function Animal (name) {
    this.name = name;
    this.sayMyName = function() { ←同じ内容なのにメソッドがオブジェクト毎に生成される。
        return 'My name is '+this.name;
    };
}

var dog = new Animal('dog');
alert(dog.constructor.name); // Animal
alert(dog.sayMyName()); // My name is dog

var cat = new Animal('cat');
alert(cat.constructor.name); // Animal
alert(cat.sayMyName()); // My name is cat
```

8. コンストラクタ／プロトタイプ

console.dirで確認すると、オブジェクト毎に sayMyNameが生成されていることがわかります。



それ、プロトタイプで解決できるよ

**sayMyNameメソッドをAnimalクラスのプロトタイプに持たせることで、
newしたオブジェクト毎に無駄な sayMyNameメソッドを生成しなくなります。**

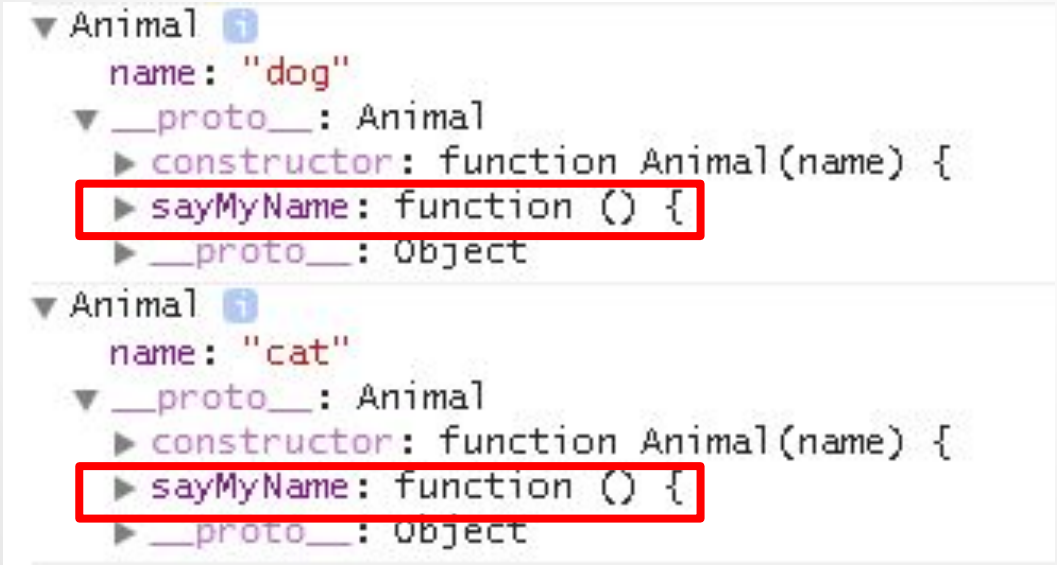
```
function Animal (name) {
    this.name = name;
}
Animal.prototype.sayMyName = function() {
    return 'My name is '+this.name;
}

var dog = new Animal('dog');
alert(dog.sayMyName()); // My name is dog

var cat = new Animal('cat');
alert(cat.sayMyName()); // My name is cat
```

8. コンストラクタ／プロトタイプ

console.dirで確認すると、prototypeとしてsayMyNameが生成されていることがわかります。



静的メンバ（static変数）

コンストラクタ名の後にドットを繋げて変数名を定義すると、 newを必要としない変数、いわゆる、static変数（静的メンバ）として利用可能です。

```
function Animal () {
}
Animal.STATIC_VALUE = 'static value'; // ←static変数のように使える
Animal.prototype.staticValue = Animal.STATIC_VALUE;

var dog = new Animal();
alert(dog.staticValue); // static value

var cat = new Animal();
alert(cat.staticValue); // static value
```

あれ、何かに気づきました！？

プロトタイプチェイン

dog.__proto__.sayMyNameではなく、dog.sayMyNameで呼び出せる理由は、
親のオブジェクトを順に辿って、メソッドやプロパティを探す仕組みがあるため。

```
function Animal (name) {
    this.name = name;
}
Animal.prototype.sayMyName = function() {
    return 'My name is '+this.name;
}
var dog = new Animal('dog');
alert(dog.sayMyName()); // My name is dog
```

←sayMyNameが自オブジェクトにない

←sayMyNameがプロトタイプにあった

※__proto__はプロトタイプチェインを実現するためのオブジェクト。
※チェインは undefinedになるまで、最終的に Objectオブジェクトまで辿ります。
※既に説明したスコープチェインと良く似た考え方であることが分かります。

コンストラクタの見栄えが悪いと感じたら

即時関数内にまとめることで、関数ブロックの外が汚れない感じに。
メソッドとして呼び出したい関数は prototypeに列挙したらよい。

```
var Animal = (function(name) {
   function Animal(name){
      this.name = name;
   }

   function sayMyName() {
      return 'My name is '+this.name;
   }

   Animal.prototype = {
      constructor: Animal,
      sayMyName: sayMyName,
   };

   return Animal;
})();

var dog = new Animal('dog');
alert(dog.constructor.name); // Animal
alert(dog.sayMyName()); // My name is dog
```

コンストラクタ／プロトタイプのまとめ

- コンストラクタはnew演算子で作成可能。
- プロトタイプの実体は、コンストラクタが持っているprototypeプロパティ。
- プロトタイプにメソッドを定義することで、擬似的なクラスが実装可能。
- メソッド、プロパティを呼び出した際は、プロトタイプチェインにより、自オブジェクト→自オブジェクトのプロトタイプ→親オブジェクトの順に定義されているか検索する。

本日、伝えられなかった内容

- 6種類の型
  - Number
  - String
  - Boolean
  - Null
  - Undefined
  - Object
- 4種類のthis
  - メソッド呼び出し
  - 関数呼び出し
  - コンストラクタ呼び出し
  - apply / call呼び出し
- 3種類の関数の書式
  - function文
  - Functionコンストラクタ
  - 関数リテラル（無名関数／匿名関数）
- クラスの書式
  - いろいろ

参考資料

- JavaScript/Object
  - http://ja.wikibooks.org/wiki/JavaScript/Object
- instanceof
  - https://developer.mozilla.org/ja/docs/JavaScript/Reference/Operators/instanceof
- typeof やら instanceof やら toString.apply やら
  - http://kokudori.hatenablog.com/entry/20120915/1347705807
- JavaScriptのイロハ
  - http://builder.japan.zdnet.com/script/sp_javascript-kickstart-2007/
- 猿でもわかるクロージャ超入門　まとめ
  - http://dqn.sakusakutto.jp/2009/01/javascript_5.html
- クロージャの使用
  - https://developer.mozilla.org/ja/docs/Web/JavaScript/Guide/Closures
- JavaScriptを学ぶ上で避けては通れないレキシカルスコープ
  - http://d.hatena.ne.jp/replication/20100309/1269052178
- JavaScript のスコープチェーンとクロージャを理解する
  - http://tacamy.hatenablog.com/entry/2012/12/31/005951
- 【JavaScript】文字列も配列の添字を使ったアクセス方法と同じ方法で各文字にアクセス可能！
  - http://taccuma.com/string-index-access/
- javascriptのオブジェクト指向とかプロトタイプとか
  - http://blog.livedoor.jp/sasata299/archives/51189103.html
- や…やっと理解できた！JavaScriptのプロトタイプチェーン
  - http://d.hatena.ne.jp/maeharin/20130215/javascript_prototype_chain
- proto と prototype
  - https://github.com/toocheap/TechMemos/blob/master/JS__proto__and_prototype.md
- プロトタイプチェーンをもっと理解する
  - http://tacamy.hatenablog.com/entry/2012/12/17/000931
- JavaScriptでのクラス定義
  - http://qiita.com/kakusuke/items/912811cfb87e991176ae
- JavaScriptの「this」は「4種類」？？
  - http://qiita.com/Haru39/items/9935ce476a17d6258e27

ご清聴ありがとうございました

HTML5+α at FUKUOKA