JavaScript再入門

山田祥寛
yoshihiro@wings.msn.to
http://www.wings.msn.to/

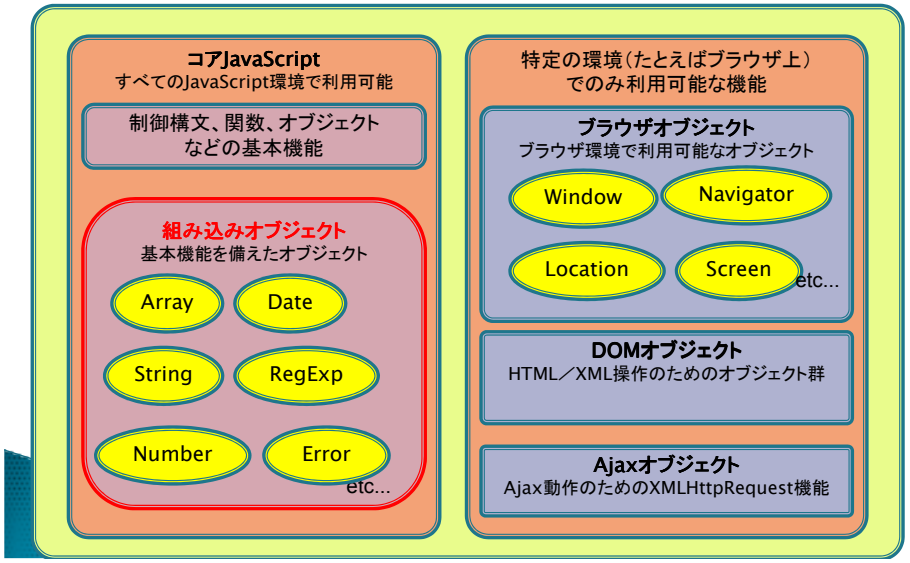
本コースの目的と前提

- Ajaxや、来るべきHTML5時代に向けて、大規模なJavaScript開発にも耐えられる基礎をおさえる
  - JavaScriptをなんらか入門したことのある方を対象
- 汎用的なコアJavaScriptの理解を深めることが目的
- ブラウザオブジェクト、Ajaxなどは扱いません
- なるべく考え方を中心に説明するので、端折るところもありますが、気軽に質問してください

JavaScriptの特徴

- 習得が容易なスクリプト言語
  - 初心者が馴染みやすい
  - オブジェクト指向構文も持ち、本格的な開発にも使える
- トライ＆エラーが簡単なインタプリタ言語
  - 動作の遅さもJavaScriptエンジンの高速化で改善
- さまざまな環境で動作する
  - JavaSE 6、Flash（ActionScript）、WSH、Node.js …

JavaScriptの構造



JavaScriptの記法

- <script>タグで囲む
- src属性で外部化も可能
- <script>タグの記述位置

基本構文

- 文（Statement）のルール
  - セミコロンは任意だが必須
    - 特に次の文が() の場合などは危険！
- コメント
  - 複数行コメントには要注意　/* var reg = /.*/; */
- 変数
  - 変数の宣言は任意だが必須

識別子の命名規則

- 最低限の規則
- 読みやすいコードのための規則
  - ただし、例外もあり（カウンタ変数など）
- 変数名はcamelCase記法、クラス名はPascal記法

データ型

- JavaScriptはデータ型に寛容な言語
  - でも、本格的な開発にはデータ型の理解は重要
- 参照型（array、object、function）、基本型（その他）
- リテラル表現
  - データ型に格納できる値そのもの、またはその表現方法
  - JavaScriptはリテラルでなんでも表現する言語
    → 配列や関数、正規表現、オブジェクトまで

リテラル表現

- 数値リテラル
  - n進数、指数表現など でも見かけ上の違いだけ
- 文字列リテラル
  - シングルクォートでの囲みが基本
- 配列リテラル
- 連想配列リテラル
  - オブジェクトと連想配列は同じもの
  - ドット構文とブラケット構文（厳密には等価でない）

演算子（1）

- 算術演算子
  - 文字列／数値混在の演算に注意
  - 前置演算、後置演算で結果が違う
  - 小数点演算の誤差
- 代入演算子
  - 参照型の代入はアドレスのコピー
- 比較演算子
  - 参照型の比較はアドレスの比較
  - 等価演算子と同値演算子（親切はできるだけ排除）

演算子（2）

- 論理演算子
  - 空文字列、0、NaN、null、undefinedはfalse
    - ‘0’や[]、{}はtrueなので注意
  - ショートカット演算
    - var win = sub_win || window;
- 三項演算子
  - var chkbox = ‘<input type=“checkbox” name=“chk”‘
    + (isChecked ? ' checked' : '') + ' />';
- 優先順位＆結合則
  - 優先順位は明確に →複雑な式ではできるだけ丸カッコを使う
  - 結合則は、直感的に当たり前のことばかり

オブジェクトの基本

- オブジェクトと連想配列は同じものだけど…
  - オブジェクトはモノそのもの
  - 連想配列はモノの集合体
- オブジェクトは、プロパティとメソッドの集合
  - 正確には、JavaScriptにはメソッドは存在しない
- 組み込みオブジェクト
  - あらかじめJavaScriptで用意されたオブジェクト

組み込みオブジェクト(1)

- ラッパーオブジェクト
  - 基本データ型にオブジェクトとしての体裁を与える
    - Object、Array、String、Boolean、Number、Function
- JavaScriptではリテラルがそのままオブジェクトになる
  - ラッパーオブジェクトは使わない(not 使う必要がない)
    - var ary = new Array(10);　×
    - var x = new Boolean(false);
      if (x) { window.alert('true'); }　×
- 文字列連結のテクニック
  - 「+」演算子は使わない

```
var data = [];
for (var i = 0; i < items.length; ++i) {
  data[i] = hogehoge(items[i]);
}
var str = data.join('');
```

組み込みオブジェクト(2)

- Globalオブジェクト
  - グローバル変数、関数を管理する便宜的なオブジェクト
  - たとえば、parseFloat(x) のようにそのままメソッドを呼び出せる

関数の基本

- 関数の定義方法は3つ
  - function命令、Functionコンストラクタ、関数リテラル
- 関数定義／利用の注意点
  - return命令は改行しない
  - 関数はデータ型の一種である
  - functionは静的な構造、Function／関数リテラルは実行時に評価

スコープ

- スコープは2種類(グローバル＆ローカル)
  - スコープの異なる変数は同名でも別もの
- 変数宣言にvarが必須なわけ
- ローカル変数の有効範囲は解りにくい
  - できるだけ関数の先頭での宣言が望ましい
- 参照渡しには要注意
- ブロックスコープは存在しない
- 関数リテラル、Functionオブジェクトではスコープが異なる

argumentsオブジェクト

- JavaScriptは引数の省略は自由
- 引数はすべてargumentsオブジェクトに格納される
- argumentsオブジェクトの役割
  - 引数の個数チェック
  - 可変長引数の関数を定義（※ 引数opt_～など）
  - 再帰呼び出し

関数の高度なテーマ(1)

- 名前付き引数
  - オブジェクトリテラルで引数を渡す
- 高階関数 ＝ 関数を引数、戻り値とする関数
  - 処理の大まかな枠組みだけを提供
  - たとえば…$.eachという命令の例

引数としてインデックス、値を受け取る関数

```
var data = [1, 2, 4, 8, 16];
function showElement(key, value) {
    document.writeln(key + ':' + value);
}

$.each(data, showElement);
```

関数の高度なテーマ(2)

- 匿名関数
  - 使い捨ての関数に名前を付けるのは無駄
  - 高階関数には匿名関数を使う
- スコープチェーン
  - Globalオブジェクト＝グローバル変数、関数を管理
  - Callオブジェクト＝ローカル変数を管理
- クロージャー
  - ローカル変数を参照している関数内関数
  - シンプルなオブジェクトのようなもの
  - 参考）http://jibbering.com/faq/notes/closures/#clSto

オブジェクト指向構文の基礎

- プロトタイプベースのオブジェクト指向
  - プロトタイプ＝オブジェクトの元となるオブジェクト
  - 縛りの弱いクラス
- 関数オブジェクトがコンストラクタを表す
  - Pascal記法で命名（memberではなくMember）
  - 「this.プロパティ名 ＝ 値」でプロパティを初期化
  - 「this.プロパティ名 ＝ 関数」でいわゆるメソッドを表現
- メソッドはインスタンスに追加することもできる（緩い）

prototypeプロパティ

- コンストラクタによるメソッド追加は無駄が多い
  - インスタンスの数だけメソッドをコピー
  - メソッドはprototypeプロパティの配下で宣言する
- プロトタイプを使うメリット
  - メモリの利用量を節減できる
  - メンバの追加や変更をリアルタイムに認識できる
- プロトタイプの注意点
  - プロパティをプロトタイプに設定したら？
  - プロパティを削除したら、すべてのインスタンスに影響？
    - インスタンス単位でプロパティを無効化するにはundefined
  - プロトタイプはオブジェクトリテラルで宣言
  - 組み込みオブジェクトのプロトタイプ拡張はNG（特にObject）

補足：静的メンバ

- Class.method(args,…) の形式で呼び出せる
- クラスに対して直接メンバーを追加する
- 静的メンバーの注意点
  - 静的プロパティは読み取り専用
  - thisキーワードは使わない
- 静的メンバーを定義するわけ
  - 名前の競合を防ぐ（グローバル変数、関数は少なく）
  - 例. jQueryのグローバルメンバは$、jQueryのみ

プロトタイプチェーン

- プロトタイプチェーン＝継承のしくみ
  - プロトタイプを順番に上位に向かって検索
  - チェーンの終端はObject.prototype
- 継承関係は差し替えることもできる
  - ただし、プロトタイプチェーンはインスタンス生成時に固定

名前空間（パッケージ）

- JavaScriptは名前空間の機能はない
- 静的メンバーを定義する要領で名前空間を定義
- 名前空間をインポートするしくみはない
  - オブジェクト名は、常に完全修飾名で表す必要がある

単体テスト

- JavaScriptの世界でもテストは必須
- JsUnit ＝ メソッドや関数の動作をチェック
- テストケースに必要なモノ
  - setUp、tearDownメソッド（初期化と後始末）
  - testXxxxxメソッド（テストメソッド）
- テストスイートでまとめも可

ドキュメンテーションコメント

- クラスやメソッドの仕様を説明したコメント
- あらかじめ決められたルールに則ったコメント
  - スタンドアロンタグ、インラインタグ
- ドキュメント生成ツールJsDoc
  - Perlベースのコマンドラインツール

コードの圧縮

- 余計なコメントや空白を除去し、サイズを小さく
  - Microsoft Ajax Minifierなど
- 難読化ツール
  - ほぼ圧縮と同じ意味
  - 変数名を短いものに、文字列をエンコード処理
  - ただし、小さなスクリプトではサイズが大きくなることも

コーディング規約

- 読みやすいコードは自分のためでもある
- コーディング規約＝読みやすいコードのための指針
- 「安全な」コードを記述するためのノウハウでもある

ご清聴ありがとうございました

- Q&A掲示板 教科書に関する質問はこちらへ
  - http://keijiban.msn.to/top.jsp?id=gr7638