JavaScriptの歴史をざっくりまとめたよ!

JavaScript ポエム altjs

第一部 JavaScript、その起源

- 1995年 NetScape社のブレダン・アイク氏が開発
  - ポケモン赤・緑とタメ
  - 最初はLiveScriptって名前だった

「Javaっていう言語が超ナウいらしいぞ」

- 1995年にJavaによるブラウザWebRunnerが、HotJavaという名で世界に公開
- NetscapeがJavaのサポートを表明

今、Javaが熱い!!

「せや!名前もらったろ!」

改名 LiveScript -> JavaScript

第二部 スクリプト戦国時代

ライセンスの乱

- 1996年　MicrosoftがIE3.0を発表「JavaScript使わせてやー」
    - NetScape社「だが断る」
    - Microsoft「ほな勝手に作るわ」 => JScript誕生
    - 各ブラウザベンダが独自言語を実装する戦国時代へ
    - Webエンジニアは血の涙を流す

想像図

「JavaScriptってどうよ?」

- インターネットの普及により個人サイト増加
    - 無駄にアニメーション
    - ブラウザのセキュリティへの考慮の不足

    - 「セキュリティ上まずいからJavaScript無効にしてね」という風潮

-> 「JavaScriptいらね」

- (今は亡き)cgiとかFlashとかが幅を利かせる

暗黒時代

- 1997年　戦乱に疲弊した各社が和平(仕様の標準化)に動き始める

NetscapeがECMAに標準仕様の策定を依頼
-> 記念すべきECMA-262初版が策定

「やっと平和が訪れるのか?」

ところがどっこい 夢でした

ECMA-262 第4版戦争

- 標準化仕様を巡って各社の対立
- 第4版推進派: Adobe Mozilla(旧Netscape社)
- 否定派: Microsoft Yahoo

標準化が暗礁に乗り上げる -> 暗黒時代に突入

第三部 ルネッサンス そして標準化へ...

JavaScript復活

- 2005年 Google Map が世に発表される
  - 非同期通信(Ajax)をふんだんに利用
  - 業界に衝撃が走る
- prototype.jsやjQueryの登場
  - Webでの表現がよりリッチに
- 2008年 Google Chrome(V8エンジン搭載)誕生
  - 「JavaScriptは遅い」なんて言わせない! Cとかに比べたら遅いけど

-> 「JavaScriptってできる子じゃん！」

まさに「ルネッサーンス!」

ついでにとん挫していた標準化も進み始める

第四部　新時代の幕開け

Node.jsの登場

- 2009年 サーバサイドのJavaScriptが爆誕
  - ポケモンHG・SSとタメ
  - 付随するツール(npm とか)も発展

みんな喜べ! JavaScriptをモジュールにできるぞ!

-> 「ブラウザのJavaScriptもモジュールにしてぇなぁ」

ビルドツール

browserify

webpack

webpack
MODULE BUNDLER

ビルドを手動でやるとかだりぃわ

Gulp vs Grunt

今ではGruntはだいぶマイナー

Gulpも開発が滞り気味

「そんなん使わんでもコマンド組み合わせたらいけるっしょ」の精神

第五部　もはやJavaScriptではない

ECMAScript2015(通称ES6)の策定

- 新しい仕様が盛り込まれ、だいぶ開発者にやさしくなる
- しかし、ブラウザ側の対応が追い付かない(主にIE)

		Compilers/polyfills					Desktop browsers																
	97%	56%	71%	48%	59%	18%	5%	11%	83%	93%	95%	86%	94%	94%	97%	97%	97%	97%	97%	54%	100%	100%	100%	100%
Feature name	Current browser	Traceur	Babel + core-js[2]	Closure	TypeScript + core-js	es6-shim	KQ 4.14[3]	IE 11	Edge 13[4]	Edge 14[4]	Edge 15[4]	FF 45 ESR	FF 51	FF 52 Beta	FF 53 Aurora	FF 54 Nightly	CH 56, OP 43[1]	CH 57, OP 44[1]	CH 58, OP 45[1]	SF 9	SF 10	SF 10.1	SF TP	WK
Optimisation																								
proper tail calls (tail call optimisation)	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2
Syntax																								
default function parameters	7/7	4/7	4/7	4/7	5/7	0/7	0/7	0/7	0/7	7/7	7/7	4/7	6/7	6/7	7/7	7/7	7/7	7/7	7/7	0/7	7/7	7/7	7/7	7/7
rest parameters	5/5	4/5	3/5	2/5	4/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	5/5	5/5	5/5	5/5
spread (...) operator	15/15	15/15	13/15	12/15	4/15	0/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	9/15	15/15	15/15	15/15	15/15
object literal extensions	6/6	6/6	6/6	4/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	5/6	6/6	6/6	6/6	6/6
for..of loops	9/9	9/9	9/9	6/9	3/9	0/9	0/9	0/9	7/9	7/9	9/9	7/9	7/9	7/9	9/9	9/9	9/9	9/9	9/9	8/9	9/9	9/9	9/9	9/9
octal and binary literals	4/4	2/4	4/4	4/4	4/4	2/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
template literals	5/5	4/5	4/5	3/5	3/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
RegExp "y" and "u" flags	5/5	3/5	3/5	0/5	0/5	0/5	0/5	0/5	5/5	5/5	5/5	2/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	5/5	5/5	5/5	5/5
destructuring, declarations	22/22	20/22	21/22	19/22	15/22	0/22	0/22	0/22	0/22	21/22	22/22	19/22	21/22	21/22	22/22	22/22	22/22	22/22	22/22	19/22	22/22	22/22	22/22	22/22
destructuring, assignment	24/24	23/24	24/24	17/24	19/24	0/24	0/24	0/24	0/24	23/24	24/24	21/24	23/24	23/24	24/24	24/24	24/24	24/24	24/24	21/24	24/24	24/24	24/24	24/24
destructuring, parameters	23/23	19/23	20/23	18/23	15/23	0/23	0/23	0/23	0/23	22/23	23/23	18/23	20/23	20/23	23/23	23/23	23/23	23/23	23/23	18/23	23/23	23/23	23/23	23/23
Unicode code point escapes	2/2	1/2	1/2	1/2	1/2	0/2	0/2	0/2	2/2	2/2	2/2	1/2	1/2	1/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
new.target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	2/2	2/2	2/2	2/2
Bindings																								
const	16/16	14/16	14/16	14/16	14/16	0/16	2/16	12/16	12/16	16/16	16/16	12/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	1/16	16/16	16/16	16/16	16/16
let	12/12	10/12	10/12	10/12	10/12	0/12	0/12	10/12	10/12	12/12	12/12	10/12	12/12	12/12	12/12	12/12	12/12	12/12	12/12	0/12	12/12	12/12	12/12	12/12
block-level function declaration[12]	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Functions																								
arrow functions	13/13	11/13	9/13	10/13	9/13	0/13	0/13	0/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	0/13	13/13	13/13	13/13	13/13
class	24/24	17/24	19/24	13/24	19/24	0/24	0/24	0/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	16/24	24/24	24/24	24/24	24/24
super	8/8	7/8	4/8	5/8	7/8	0/8	0/8	0/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	6/8	8/8	8/8	8/8	8/8

「ブラウザはES5でしか動かないけど、ES6で書きたいよ～」

「力(ES6)が欲しいか?」

「力が欲しいなら」

「くれてやる!」

Babelの衝撃

- ES6で書いたコードをES5に変換
- 書くのはあくまでJavaScript

- メリット: ブラウザの実装が追い付けば変換せずにそのまま使える
- デメリット: 所詮JavaScript 文法的にはガバガバ

「いや、もっといい感じに書きたいんだわ」

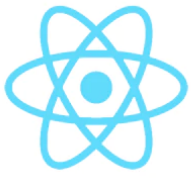
AltJSの登場

AltJS = 実行前にJavaScriptに変換する言語 基本的にこのままでは実行できない

- TypeScript
  - Microsoftが開発
  - 静的型宣言が特徴　バグを早期発見できる
  - 中規模以上、メンテナンスとかを考慮するときに強い
  - 現状ではAltJSの最有力
- CoffeeScript
  - とにかくコードの量を減らしてスッキリさせたい

    - 小規模向き
    - Ruby on Railsはデフォで対応
    - ES6に仕様が逆輸入されてから下火
- Dart
    - Googleが開発　見た目がGo言語チック
    - 「各ブラウザにDartVM(Dartの実行環境)搭載や!」 -> 「ダメでした...」
    - ちょっと時代が追い付いてない印象

フレームワークも続々登場



React

Vue.js

aurelia

人気があるのはAngular, Vue.jsあたり

おわりに

今後のJavaScript

- 黎明期ほど手軽ではない
  - 簡単なJavaScriptはすぐにブラウザで実行できるけど、真面目にやりだすと環境の準備が怠い
  - webpackなどのビルドツールも使えるようにする必要がある
  - (AltJS使うなら)実質2言語学習が必要
  - (フレームワークを使うなら)それに加えてさらにフレームワークの作法も習得が必要
  - そもそもJavaScriptに求められることがめっちゃ増えてる
- とはいえ、現状ブラウザで実行できるのはこいつだけなのでやらざるを得ない
- 技術の移り変わりがとくに激しい
  - 「イケるやん」と思った技術もすぐに廃れる 1年前は絶好調だったGulpがががが...
  - 勉強のしがいはあるかもね

参考資料

- Javascriptの歴史(Qiita)
- 言い訳とJavaScriptの歴史
- JavaScriptの歴史とルーツを辿る｜背景を知ってJSを深く理解しよう

おまけ1 CoffeeScriptのサンプル

sample.coffee

```
class Animal
  constructor: (@name) ->

  move: (meters) ->
    alert @name + " moved #{meters}m."

class Snake extends Animal
  move: ->
    alert "Slithering..."
    super 5

class Horse extends Animal
  move: ->
    alert "Galloping..."
```

```
    super 45

sam = new Snake "Sammy the Python"
tom = new Horse "Tommy the Palomino"

sam.move()
tom.move()
```

おまけ2 TypeScriptで書いてみた

sample.ts

```
abstract class Animal {

  private name: string;

  constructor(name: string) {
      this.name = name;
  }

  move(meters: number): void {
      alert(this.name + " moved " + String(meters));
  }
}

class Snake extends Animal {
```

```
    move(): void {
        alert("Slithering...");
        super.move(5);
    }
}

class Horse extends Animal {
    move(): void {
        alert("Galloping...");
        super.move(45);
    }
}

const sam = new Snake("Sammy the Python");
const tom = new Horse("Tommy the Palomino");

sam.move();
tom.move();
```

おまけ3 これがES5に変換されると

ファッ!?

sample.js

```
var __extends = (this && this.__extends) || function (d, b) {
    for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
    function __() { this.constructor = d; }
    d.prototype = b === null ? Object.create(b) : (__.prototyp
};
var Animal = (function () {
    function Animal(name) {
        this.name = name;
    }
    Animal.prototype.move = function (meters) {
        alert(this.name + " moved " + String(meters));
    };
    return Animal;
}());
var Snake = (function (_super) {
    __extends(Snake, _super);
    function Snake() {
        return _super.apply(this, arguments) || this;
    }
    Snake.prototype.move = function () {
        alert("Slithering...");
        _super.prototype.move.call(this, 5);
    };
    return Snake;
}(Animal));
var Horse = (function (_super) {
    __extends(Horse, _super);
    function Horse() {
        return _super.apply(this, arguments) || this;
```

```
    }
    Horse.prototype.move = function () {
        alert("Galloping...");
        _super.prototype.move.call(this, 45);
    };
    return Horse;
}(Animal));
var sam = new Snake("Sammy the Python");
var tom = new Horse("Tommy the Palomino");
sam.move();
tom.move();
```