JavaScript po stronie serwera

Od tropienia mitów do implementacji


THE UNITED STATES OF AMERICA
ONE
IN GOD WE TRUST
ONE
ANNUIT COEPTIS
NOVUS ORDO SECLORUM
ONE DOLLAR

JavaScript po stronie serwera

Od tropienia mitów do implementacji

- Łyk historii
- Okres błędów i wypaczeń
- CommonJS
  - Moduły
  - Pakiety
  - Implementacje
- I o co tyle zamieszania?
- I co dalej?

Łyk historii

1995: Brendan Eich creates JavaScript

1996: Microsoft creates JScript, and Netscape creates the Server-side JavaScript

1997: JavaScript's success result in a new standard: ECMAScript

1998: XML and DOM level 1 become W3C recommendations

1999: Microsoft creates XMLHttpRequest

2000: Macromedia implements a prototype object to its scripting language and calls it ActionScript

2001: Douglas Crockford creates JSON

2002: Douglas Crockford creates JSLint

2003: Adobe extends Acrobat with JavaScript

2004: E4X (ECMAScript for XML) is published by the W3C, and Tim O'Reilly coins the term "Web 2.0"

2005: Sam Stephenson creates the Prototype JavaScript Framework, and Jesse James Garrett coins the term "Ajax"

2006: Joe Hewitt publishes Firebug 1.0, John Resig creates JQuery, and Alex Russell creates the term "Comet"

2007: Creation of the OpenAjax Hub and first InteropFest

2008: Creation of ARIA: Accessible Rich Internet Application, to provide accessible Ajax

2009: ECMAScript Harmony becomes ECMAScript 5

2010: JavaScript 2.0

Źródło: http://en.wikipedia.org/wiki/JavaScript

Okres błędów i wypaczeń

```
1 var Browser =
2 {
3    IE:     !!(window.attachEvent && !window.opera),
4    Opera:  !!window.opera,
5    WebKit: navigator.userAgent.indexOf('AppleWebKit/') > -1,
6    Gecko:  navigator.userAgent.indexOf('Gecko') > -1 &&navigator.userAgent.indexOf('KHTML') == -1,
7    getName: function(){
8       if (this.IE){
9             return 'ie';
10       }
11       if (this.Opera)
12       {
13             return 'opera';
14       }
15       if (this.WebKit)
16       {
17             return 'webkit';
18       }
19       if (this.Gecko)
20       {
21             return 'ff';
22       }
23    }
24 };
```

```
1 #ifndef _WIN32
2 #include <sys/times.h>
3 #include <sys/wait.h>
14 #ifdef _BSD
15 #include <sys/stat.h>
16 #if !defined(__APPLE__) && !defined(__OpenBSD__)
17 #include <libutil.h> 18 #endif
19 #ifdef __OpenBSD__
20 #include <net/if.h>
21 #endif
22 #elif defined (__GLIBC__) && defined (__FreeBSD_kernel__)
23 #include <freebsd/stdlib.h>
24 #else
25 #ifndef __sun__
26 #include <linux/if.h>
27 #include <linux/if_tun.h>
28 #include <pty.h>
29 #include <malloc.h>
30 #include <linux/rtc.h>
```

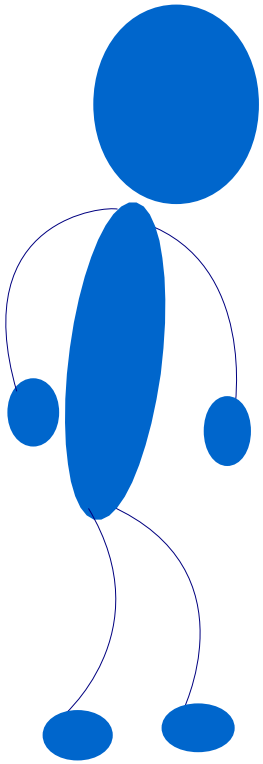

ScriptsResource.com
THE SCRIPTS RESOURCE DIRECTORY


BigResource
Gettin' even bigger

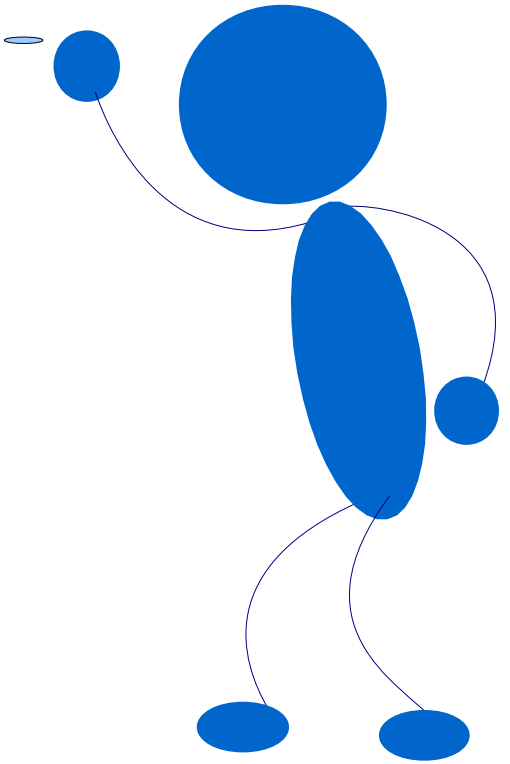

SOLIDSCRIPTS

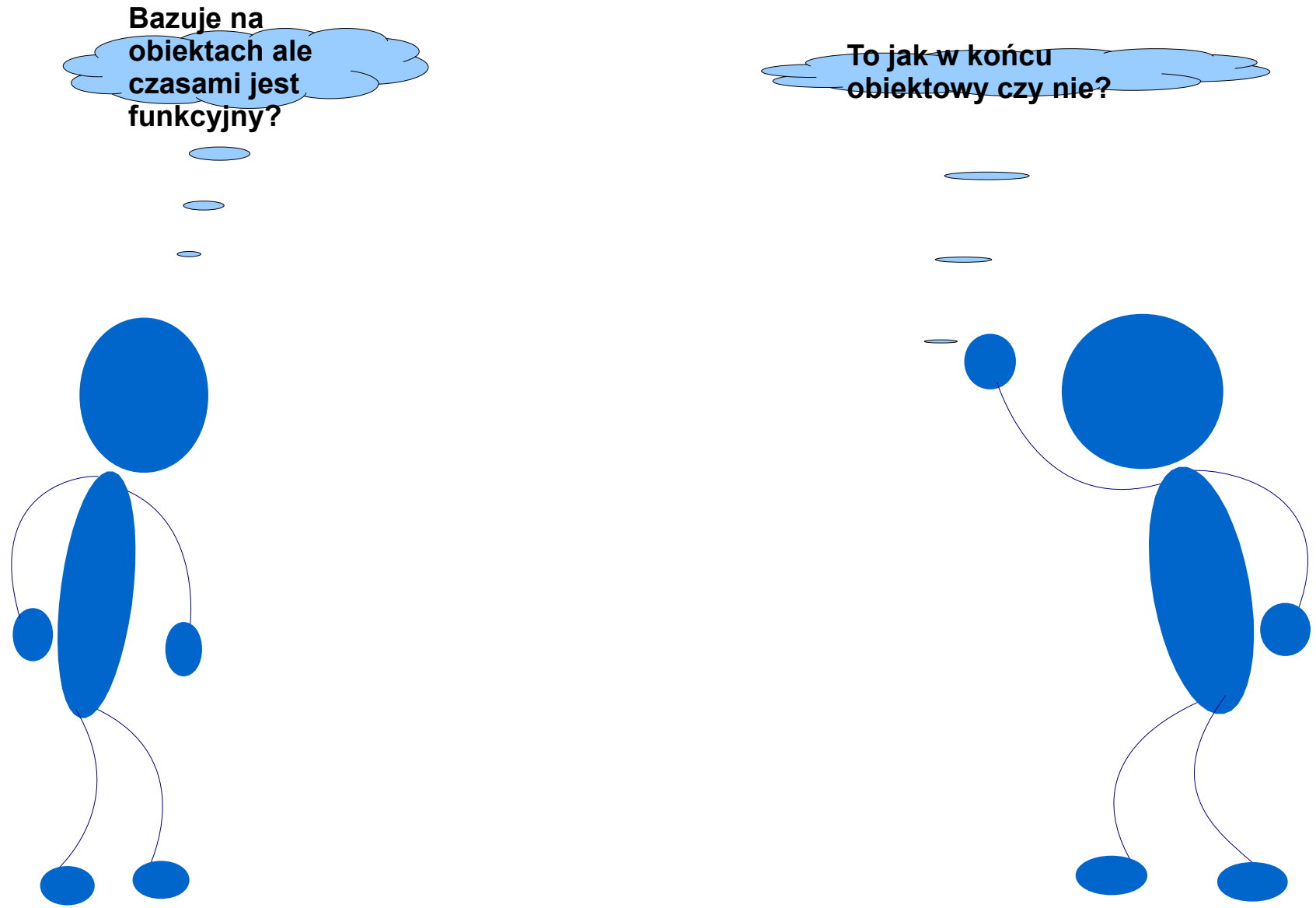

HOTSCRIPTS®
EST. 1998

Jeśli używają go Ci
od HTML'a
to nie może być
poważny język



Java dla
inteligentnych
inaczej
czy dla opornych?




Bazuje na obiektach ale czasami jest funkcyjny?
To jak w końcu obiektowy czy nie?

ECMAScript is an object-oriented programming language for performing computations and manipulating computational objects within a host environment

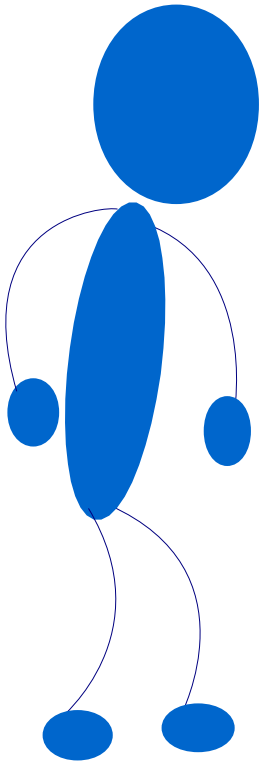
ECMAScript TC39, Page 1

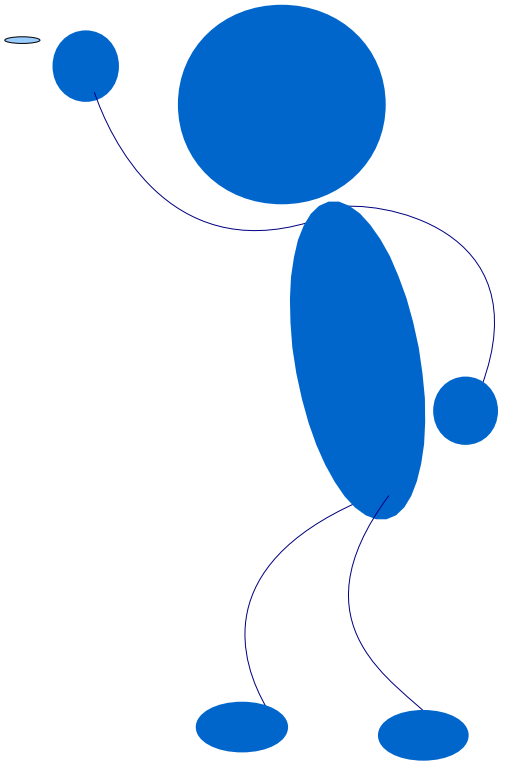
ECMAScript is object-based: basic language and host facilities are provided by objects, and an ECMAScript program is a cluster of communicating objects

ECMAScript TC39, Page 2

I do tego jeszcze te prototypy.

I jak to się w końcu nazywa ECMAScript czy JavaScript?





4 DEVELOPERS

IT Project Management | PHP | .NET & C# | Java

March 26th, 2010 in Poznan, Poland

ECMAScript vs. JavaScript

- ECMAScript (ECMA-262) (http://www.ecmascript.org/) definuje
  - semantykę języka
  - podstawowe typy danych
    - String
    - Boolean
    - Number
    - Object
  - podstawowe obiekty
    - Math
    - Array
- W3C DOM (http://www.w3.org/DOM/)
  - Model dokumentu i zdarzeń
- Wsparcie dla XML
  - ECMAScript for XML, ECMA-357
- AJAX
  - Microsoft XMLHTTP, kontrolka ActiveX wspierana od Internet Explorer 5
  - Mozilla (zaadaptowana przez inne przeglądarki), natywny XMLHttpRequest
  - XMLHttpRequest W3C Working Draft 19 November 2009 (http://www.w3.org/TR/XMLHttpRequest/)
- **A co z pozostałymi zastosowaniami JavaScript?**

CommonJS

- Styczeń 2009:
  Kevin Dangoor na swoim blogu formuje grupę "ServerJS"
- Marzec 2009:
  po długiej dyskusji specyfikacja"securable modules" zostaje wybrana jako podstawa systemu modułów
- Kwiecień 2009:
  Pierwsze implementacje systemu modułów zostały zaprezentowane podczas pierwszej konferencji JSConf w Washington.
- Sierpień 2009:
  Zmiana nazwy na CommonJS aby lepiej oddać planowane szersze zastosowania specyfikacji.

W specyfikacji ECMAScript nie znajdziemy rozdziału o:

- systemie modułów

  Skrypty JavaScript muszą być zawierane w HTML, wstrzykiwane czy ręcznie pobierane. Językowi brakuje natywnych rozwiązań zarządzania modułami

- bibliotece standardowej.

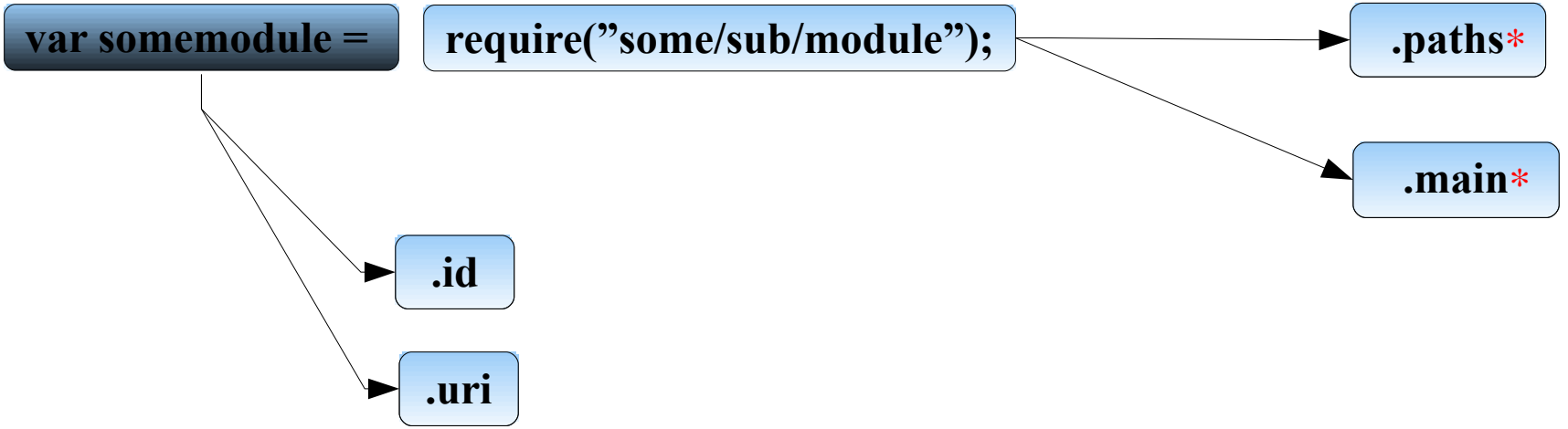
  Mamy co prawda dostępne API dla przeglądarek czy też podstawowe operacje matematyczne,ale brakuje metod dostępu do systemu plików, operacji I/O czy też baz danych

- systemie zarządzania pakietami, który by automatycznie instalował potrzebne zależności

- Moduły
  - Binary Data Type
  - Encodings
  - I/O Streams
  - Filesystem
  - System Interface
  - Unit Testing
  - Sockets
  - Event queue
  - Worker
- Pakiety
- Web Server Gateway Interface

Moduły

```
1 exports.add = function(){
2         var suma = 0;
3         var i;
4         for(i=0;i<arguments.length;i++){
5                 suma = suma+arguments[i];
6         }
7         return suma;
8 };
9
10 exports.silnia = function(n){
11         if(n){
12                 return n * silnia(n-1);
13         } else{
14                 return 1;
15         }
16 };
```


var somemodule =
require(”some/sub/module”);
.paths*
.main*
.id
.uri

```
1 var sys = require("system");
2 var math = require("math");
3
4 sys.print("6!="+math.silnia(6));
```

```
1 var print = require("system").print;
2 var silnia = require("math").silnia;
3
4 print("6!="+silnia(6));
```

Publiczne i prywatne elementy API

```
1 exports.publicMethod = function(i){
2         return privateMethod(i++);
3 }
4
5 function privateMethod(i){
6         return Math.pow(i,2);
7 }
```

Zmienne globalne modułu

Każdy moduł ładowany jest tylko raz, co wymaga od implementacji zapewnienia wsparcia dla aplikacji wielowątkowych

```
1 exports.loadTimestamp = new Date();
```

Nadrzędne i relatywne ścieżki modułów

```
1 var toplevel = require("toplevel");
2 var moduleB = require("../moduleA/moduleB");
3 var moduleC = require("./submoduleC");
```

Brak mechanizmu niejawnego importowania modułów

```
1 var moduleA = require("moduleA");
2 var sys = require("system");
3
4 sys.print(moduleA.moduleB.circleArea(2));
```

Pakiety

- Specyfikacja pakietów definiuje standaryzowany sposób dostarczania modułów CommonJS, podobnie jak:
  - deb dla Debian
  - gem dla Ruby
  - egg dla Python
- Specyfikacja CommonJS/Package/1.0
  - package.json (deskryptor pakietu)
  - struktura pakietu
  - katalog pakietów

```
{
  "name": "mypackage",
  "version": "0.7.0",
  "description": "Sample package for CommonJS. ,
  "repositories": [
    {
      "type": "git",
      "url": "http://hg.example.com/mypackage.git"
    }
  ],
  "dependencies": {
    "webkit": "1.2",
    "ssl": {
      "gnutls": ["1.0", "2.0"],
      "openssl": "0.9.8"
    }
  },
  "implements": ["cjs-module-0.3", "cjs-jsgi-0.1"],
  "os": ["linux", "macos", "win"],
  "cpu": ["x86", "ppc", "x86_64"],
  "engines": ["v8", "ejs", "node", "rhino"],
  "scripts": {
    "install": "install.js",
    "uninstall": "uninstall.js",
    "build": "build.js",
    "test": "test.js"
  }
}

```

Układ plików i katalogów w pakiecie CommonJS

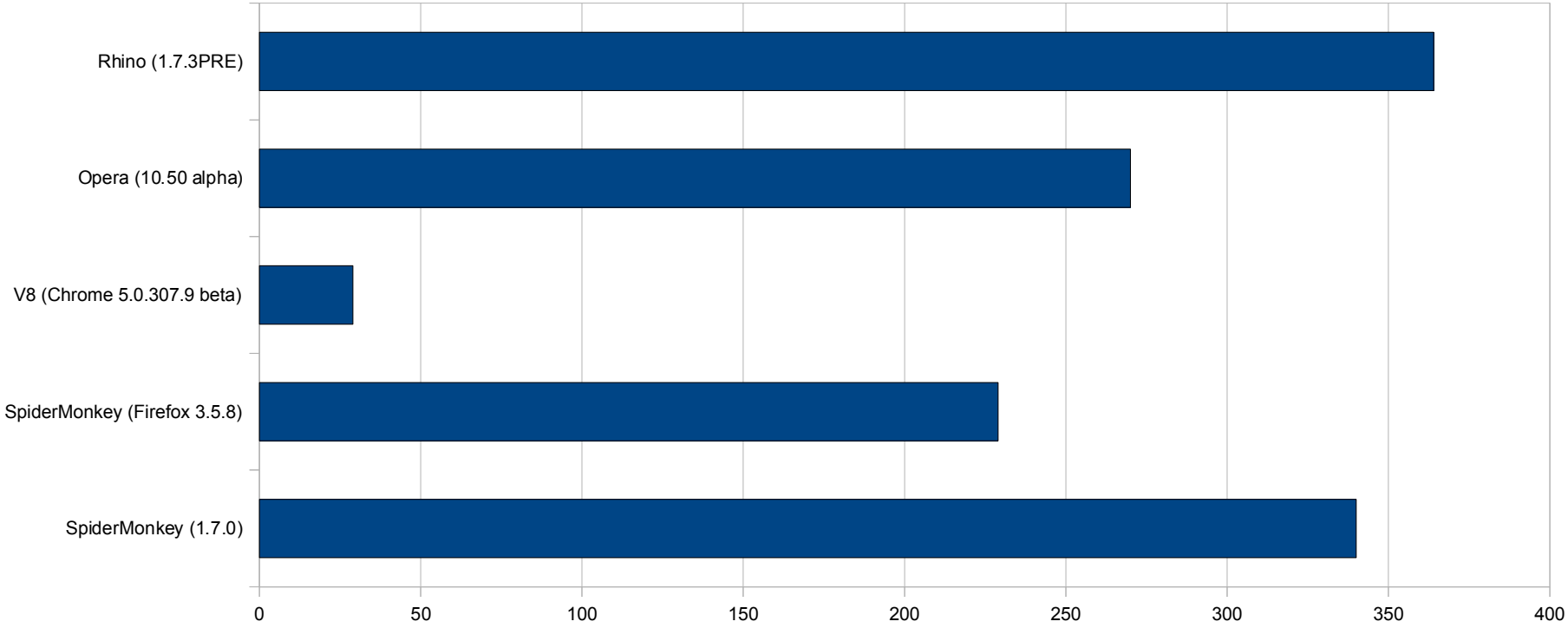
```
|-- README
|-- bin
|   |-- activate ->
activate.bash
|   |-- activate.bash
|   `-- sea
|-- doc
|-- lib
|-- narwhal.conf
|-- package.json
`-- test
```

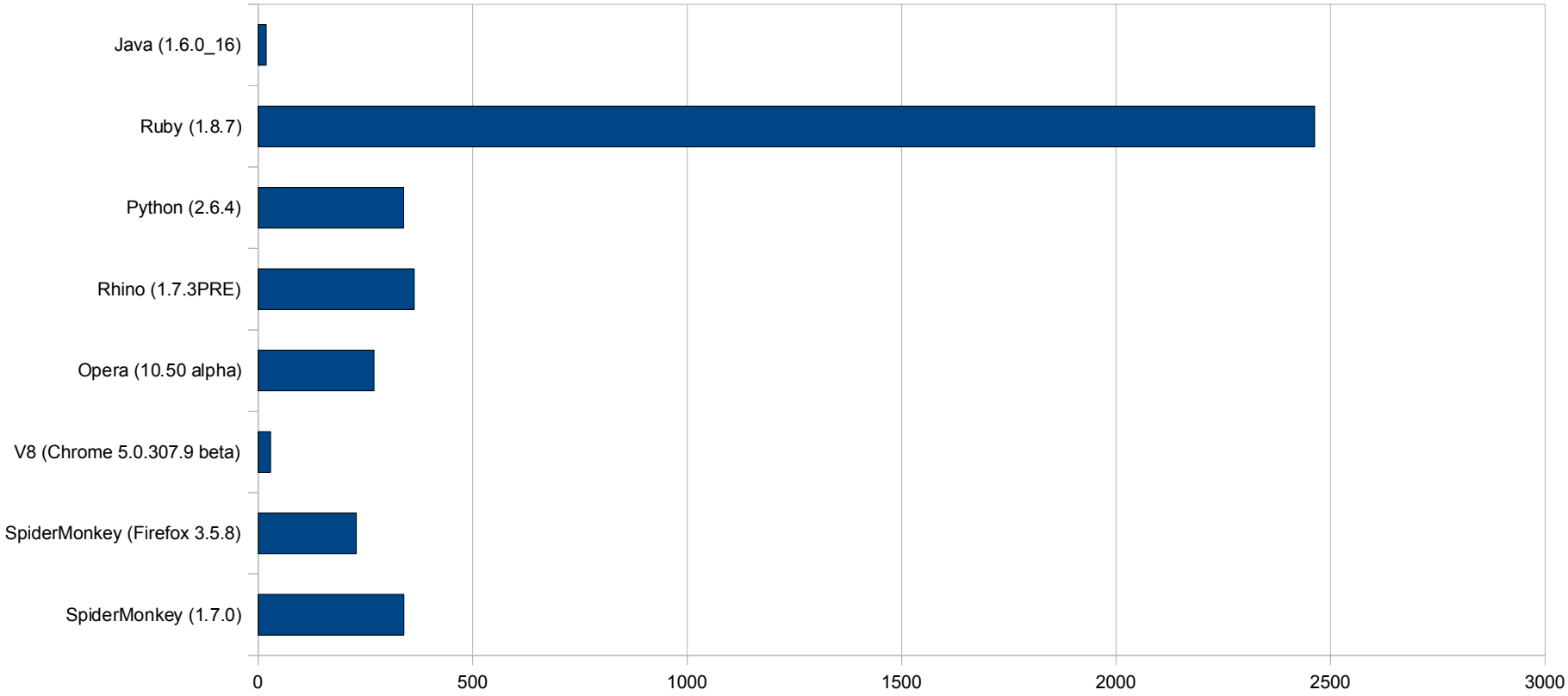
jsgi&jack

narwhal

narwhal & jack

Are you ready?


Rhino (1.7.3PRE)
Opera (10.50 alpha)
V8 (Chrome 5.0.307.9 beta)
SpiderMonkey (Firefox 3.5.8)
SpiderMonkey (1.7.0)
0
50
100
150
200
250
300
350
400


Java (1.6.0_16)
Ruby (1.8.7)
Python (2.6.4)
Rhino (1.7.3PRE)
Opera (10.50 alpha)
V8 (Chrome 5.0.307.9 beta)
SpiderMonkey (Firefox 3.5.8)
SpiderMonkey (1.7.0)
0
500
1000
1500
2000
2500
3000

ECMAScript 5

- Obiekty i własności
  - Zapobieganiu rozszerzaniu obiektów
    - `Object.preventExtensions( obj )`
    - `Object.isExtensible( obj )`
  - Deskryptory własności
    - `Value`
      Zawiera wartość własności
    - `Get`
      Funkcja wywoływana podczas dostępu do wartości
    - `Set`
      Funkcja wywoływana podczas zmiany wartości
    - `Writable`
      Jeśli `false`, wartość nie może być zmieniona
    - `Configurable`
      Jeśli `false`, usunięcie własności lub zmiana atrybutów nie powiedzie się
    - `Enumerable`
      Jeśli `true`, własność będzie widoczna podczas `for (var prop in obj){}`

- W „strict mode” niemożliwe będzie
  - Przypisanie wartości do niezadeklarowanej zmiennej
  - Definiowanie zmiennych wewnątrz eval()
  - Nadpisanie arguments wewnątrz funkcji
  - Użycie `with(){}` (błąd składni)
- JSON
  - `JSON.stringify`
  - `JSON.parse`
- Date
  - Ability to parse and output ISO-formatted dates
- Array
  - `Array.isArray`

Kilka słów od Ojca Prowadzącego na koniec