
JS

JavaScript

Otto Sipe

Otto + JS

- What do I know about JS?
  - Certainly Not Everything
  - Self Taught, Lots of Experience
  - I Have a Few Opinions
- EECS 485 IA
- Michigan Hackers Exec Board
- Feel Free to Interrupt/Ask Questions (really!)

ToDo

Overview
History
Reputation
Stats

Operators
Objects
Functions
Prototypes

Callbacks
Closures
Event Loop
Design

History

- Developed at Netscape
- Brendan Eich
- Originally “Mocha”
- Then “LiveScript”
- Java hot in the 90s...
- Targeted “non-pros”

JS Reputation

- Looked down upon by “pros”
  - Changing with more advanced web apps, AJAX
- Encourages poor style
  - Bad habits w/o constraints like types or classes
- Lots of “Dark Corners”
  - Type coercion, scopes, optional ‘;’, “callback hell”
- Hipsters love JS… (especially Node.js)

Java != JS

- JS stole things from Java (sorry...)
  - parseInt, Date, Math, etc.
  - both have C-like syntax
- JS is prototype based
- JS is dynamically typed
- JS is parsed, Just In Time Compiled (JIT)
- Java is none of these things...

JS Stats

- Dynamic Types
  - var a = 1; a = “string”; a = {}; a = []; // valid
- Object Based
- Functions Are Objects too
- Prototypes *not* Classes
- Objects are passed by reference
- string and number passed by value

ToDo

~~**Overview**~~
~~History~~
~~Reputation~~
~~Stats~~

Operators
Objects
Functions
Prototypes

Callbacks
Closures
Event Loop
Design

Operators

1 == “1”	true
1 === “1”	false
[1] == 1	true
[1] === 1	false
1 == 1	true
1 === 1	true (whew!)

Operators

[] + “”	“”
“” + []	“”
1 + {}	“1[object Object]”
{} + 1	1
[] + 1	“1”
1 + []	“1”

Operators

{} + []	0
[] + {}	“[object Object]”
{} + {}	NaN
[] + []	“”
{} + “”	0
“” + {}	“[object Object]”

Operators

- Yep, it’s nasty.
- Important part:
  - === checks for the same type
  - == will “coerce” the type (just use ===)
  - + and - will attempt to do the same
  - Hence, odd corner cases

Objects

Demo

Objects

- Super convenient
- Serialize to JSON
- Mutable
- Enumerable

```
var obj = {
    name: “Otto”,
    age: 22
};
var arr = [1,2,3,4];
```

Objects

new Object(1)	*Number {}*
new Object(“hi”);	*String {0:"h",1:"i"}*
new Object({x:1})	*Object {x: 1}*
new Object([1,2,3])	[1, 2, 3]

Functions

```
function hello(name) {
    console.log(“hello, ” + name);
}
hello() // hello, undefined
hello(“otto”) // hello, otto
hello({}) // hello, [object Object]
hello([1]) // hello, 1
```

Functions

```
function hello(name) {
    this.greeting = “hello, ” + name
}
hello(“otto”); // call function
> undefined
new hello(“otto”); // like a constructor
> hello { greeting: “hello, otto” } (an object!)
```

Functions

- Nest functions inside functions
- Use “this” to access scope of caller
- JS is Object Oriented
  - create object instances with new obj();
  - delete with “delete”
  - use this.name = function(){}; to assign methods
  - use this.name = 1; to asssign variables

Functions

Demo

Functions

- “this” refers to owner of function executed!

```
function test() {
    console.log(this) // print our owner
};
test() // Window{location: … }
new test(); // test {} (an object!)
```

Prototypes

- Chain-like inheritance
- If obj.prop or obj[“prop”] is not found:
  - Check obj’s prototype, keep moving up until end
- Essentially lazy way to inherit
- Can set obj.prototype explicitly
- Existing object’s prototype can be altered
  - Extend an existing object, like Number, or Array

ToDo

- ~~**Overview**~~
- ~~History~~
- ~~Reputation~~
- ~~Stats~~
- ~~Operators~~
- ~~Objects~~
- ~~Functions~~
- ~~Prototypes~~
- Callbacks
- Closures
- Event Loop
- Design

Callbacks

- Everything in JS is evented, returns are bad!
- Callback: function an argument to a function
- Very important part of JS!

```
function(data, callback) {
    // do something with data
    callback(result); // it's asynchronous!
}
```

Callbacks

Demo

Closures

- Sorta like an on the fly function object
- Returns a function
- Some piece of memory kept on the stack
- Simulate a private method
- Notorious for messing up when inside loops

Closures

Demo

Event Loop

- Concurrency model for JS: no threads!
- When callstack empty, next message runs
- Message processed fully before next
- Event loop does *not* block - a big deal!

```
while(q.waitForMessage()) {
    q.processNextMessage();
}
```

Design

- Use Modular Design
  - Write Objects which model real things
  - Patterns similar to C++
- Think Model View Controller (kinda)
  - Separate data from operation
  - Modules which handles UI
  - Pass events/data to other sub-modules

Design

Demo

What I Didn't Talk About

- The DOM: Document Object Model
- AJAX: Async JS and XML ("Web 2.0")
- JSON: JS Object Notation
- JQuery: Framework NOT a language
- HTML5: WebWorkers, WebSockets, GeoLoc
- Node.js: V8 on the Server-Side
- Bazillion other amazing JS projects!

Thanks!

Happy Hacking