insin-notes Documentation

Release 1

Jonathan Buchanan

Sep 27, 2017

Contents

CHAPTER 1

Conferences

JavaOne 2012

Tuesday, 2nd October 2012

Meet The Nashorn Development Team BOF

- URL: https://oracleus.activeevents.com/connect/sessionDetail.ww?SESSION_ID=4763

Notes

- No plans to open source Node.jar
- 3 former Rhino developers on the Nashorn team
- ...don't have notes for the questions I asked, so imagine you're reading interesting implementation details here...
- Didn't look at other JavaScript implementations during development to keep it "pure"
- No optimisations are planned for the near future.
- New features
  - Here strings
  - Edit strings
  - Shebangs
- Compatibility script for Rhino code
- Modules
  - Node.jar implements Node's module system
  - No plans for Nashorn itself to commit to a module system yet

- Making *heavy* use of invokeDynamic
- Node implementation is not as fast as native Node, but already in the ballpark and they think they can get there **Akhil sinks into his chair in the audience :)**
- 1 test left to reach 100% conformance with ECMA-262, ~8 bugs
- Not rushing it out the door - "we have time to get it right"
- Will be able to run under one of the smaller Jigsaw runtimes.
- Primary purpose is to enable scripting for Java, not to be some sort of solution for JavaScript in general.

Nashorn, Node.jar and Java Persistence BOF

- URL: https://oracleus.activeevents.com/connect/sessionDetail.ww?SESSION_ID=6661

Node.jar, Akhil Arora

- Nashorn
- Implemented Node's evented APIs
- Using the Grizzly Framework under the hood (GlassFish)
- Can run on any Nashorn platform
  - Including Java SE Embedded
- Node port will always be single-threaded/evented, but Java part can take advantage of threads/multi-core
- Only on Java SE 7 and above
- JavaScript part is single-threaded, but Java portions can spin up as many as you want. Have to post to the event queue to communicate back to Node.
- Access to BigDecimal/BigInteger on JVM (and others etc. etc.)
- Mapping from Node modules to Java APIs used in their implementation (incomplete list)

buffer	Grizzly ByteBuffer (nio ByteBuffer)
child_process	ProcessBuilder
dns	InetAddress etc.
fs	nio.File
http(s)	Grizzly
stream	Grizzly nio Stream

Development

- WIP based on Node v0.8.2
- Reusing Node (JavaScript) source as much as possible
- Redirecting to Java APIs where required in _wrap implementation modules
- Using the Node.js unit tests to test
- Lists of modules which were "Mostly Working", "Partial", "TBD", I only got these ones down:

  TBD crypto, cluster, dns, readline, repl, tty, zlib

Partial fs, http, net, os

Demos

Running on a Raspberry PI

- java -jar node-0.1.jar fortune.js
  - Serving up a `fortune` entry
- dates.js
  - Writing a date to the response every second, using Java Date

Q&A

- Speed?
  - Not getting into it vs Mozilla and Google engines, but faster and smaller than Rhino already.
- Can Node modules retrieved via `npm` be used?
  - Yes, but can't load modules which require native extensions
- Are there any plans to provide wrap modules for popular modules which have native deps, like Socket.IO?
  - Socket.IO support will be an add-on (i.e. later), wrapping Grizzly implementations.
- Why, and why use JavaScript at all?
  - Node API is compact and powerful, with smaller code size
  - JavaScript is dynamic, reactive, fun!
  - Make use of Java APIs
  - Secured with Java security

JPA & Node.jar, Doug Clarke

Implementation details and JS code samples of 4 ways to use JPA persistence with Nashorn/Node.jar.

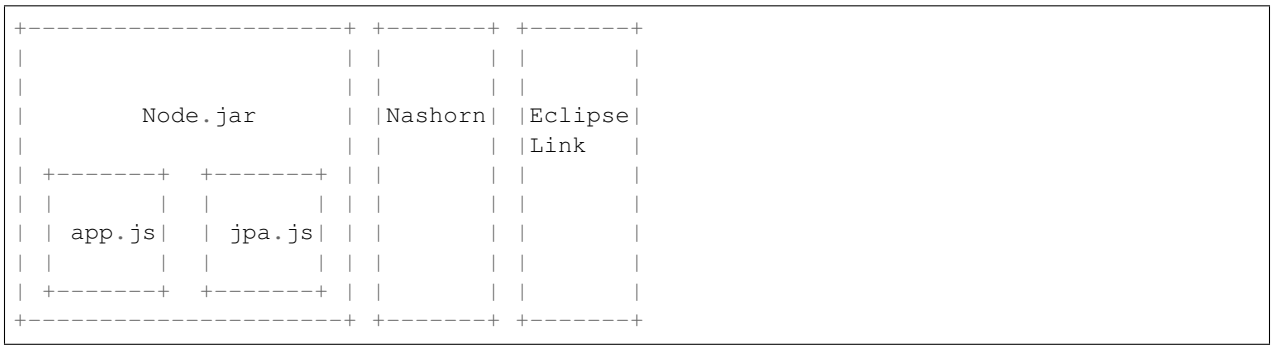
Wrapped JPA, XML + annotated Java

- Access Java EE with SSJS
- Node wrapped JPA
- Annotated Java classes and XML config file
- app.js & jpa.js

Wrapped dynamic JPA

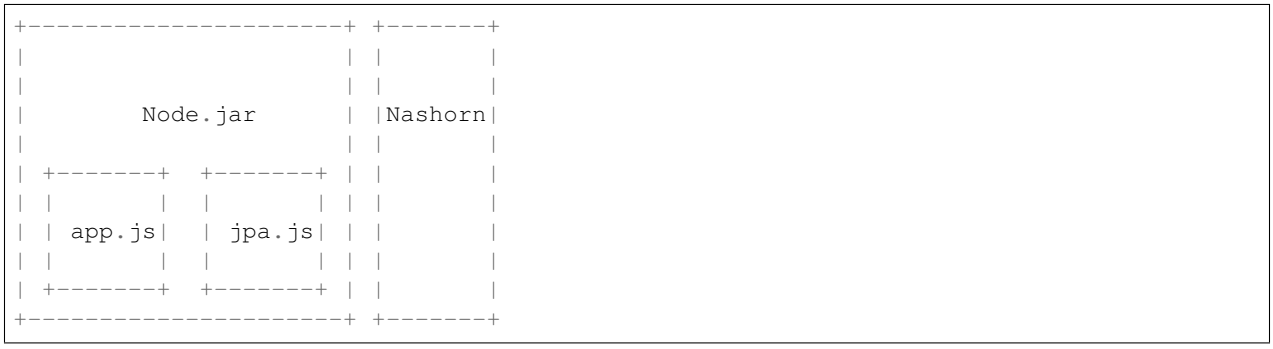
- Access Java EE with SSJS without Java
- No Java source classes

- Persistence XML
- EclipseLink offers dynamic persistence, spinning up classes at runtime (`access="VIRTUAL"`)



JS JPA & JS/JSON Config

- JavaScript object literals equivalent to XML config
- Factories required for creation of new objects, still early days on this front.



JS Database JPA

- Types and mappings from DB
- Types creted based on the schema
- Names mapped to database table names
- Use case for Nashorn/Node could be scripted service chunks leveraging EE capabilities.

Q&A

- (Paraphrasing) Sample JS code is synchronous, WTF?
  - Code put together to demonstrate use, not a JS API
  - EntityManager can transparently be made asynchronous to support writing in the standard Node async way.
- (Adam Bien) Can we do annotations?
  - Not part of the JS language

- After this question there was a bit of discussion back and forth about ways you could annotate, use cases for using EE components in JS, but it's clear that it's possible to use chunks of EE with Node.jar and all the specifics are up for grabs right now.
- It sounds like it's hard to even get a hold of Node.jar if you work for Oracle, and there are no plans to open source it.
- Which is a pity, because more people who know Node.js and use EE could be of use here. I know I'd like to start playing with it yesterday!

CHAPTER 2

Ideas

Twitter App: Substitute

This is now a thing: http://github.com/insin/substitute

A Twitter app for conveniently correcting typos.

In lieu of an edit button, correcting a typo on Twitter involves deleting the original tweet and creating a new one. If you're tweeting via the web interface, this is annoying. If you're tweeting via SMS or via a browser on a standard mobile phone using m.twitter.com, this is more painful (first-world problems definition) than it needs to be, as the typo *must* die.

Specification

The app should monitor your tweet timeline, looking for tweets matching the following format:

    s/this/that/

When a matching tweet is found, the app should:

1. locate the previous tweet in your timeline,
2. verify that it matches the first part of the replacement expression,
3. perform the replacement and tweet it,
4. delete both the original tweet and the tweet specifying the replacement.

CHAPTER 3

Scratch

Scratch area for quick notes and half-formed thoughts.

Dual-Sided JavaScript

Research

- Scaling Isomorphic Javascript Code - describes a Resource-View-Presenter pattern for dual-sided code.
- The client-side templating throwdown: mustache, handlebars, dust.js, and more - reusability is one of the criteria evaluated here.

Frameworks

- Derby (GitHub)
- Flatiron (GitHub)
- Kanso (GitHub)
- Sacrum
- Bones

Modules

Utilities

- isomorph

Models / Data

- resourceful

Routers / URL Mapping

- crossroads.js
- director
- urlresolve

Templating

- DOMBuilder
- Dust
- Jade
- Plates

Validation

- newforms
- revalidator

Shared JavaScript

`module.exports` Detection / Global Namespace Stuffing

This is what I'm currently do in my own shared modules - it's inelegant and I suspect makes it hard to share tests between Node.js and browsers in anything but node-qunit, but I'm yet to confirm that by trying to port any of the modules which use this pattern over to Mocha for testing.

Pros:

- Simple.

Cons:

- Ugly.
- Boilerplated.
- Requires dependents to `require()` into a specifically-named variable.

lib/<modulename>.js:

```
;(function(__global__, server) {

var dep = server ? require('dep') : __global__.dep

var api {}

// Define API
```

```
if (server) {
   module.exports = api
}
else {
   __global__.<modulename> = API
}

})(this, !!(module && typeof module.exports == 'function'))
```

Bundling Dependencies

Haven't actually tried this yet, but it's what I'd planned to do, calling in the context of an Object which will receive exports.

<modulename>.js:

```
;(function() {

var __global__ = {}

;(function() {

// Source for namespace-stuffed dependencies

// Source for module who needs the above dependencies

}).call(__global__)

window.modulename = __global__.modulename

})()
```

Target End State

- Code everything using Node.js-style `require()`.
- For dual-sided code which needs Node.js-only features, detect `process`, or a similar Node.js global.
- Hack in Node shims only as required.
- Export for browsers:
  - Implement `require()` for use in an IIFE, not available publicly - the final API will still be exported as a property of `window`.
  - Wrap code and dependencies with IIFEs which define `module`, `exports` and `require`.
- Dirt simple - *not automated* - should be driven off a file telling it exactly what to do. See Research section below for that. If you find yourself going down or needing to go down that route, use one of those instead.

Simple test case:

/concur.js:

```
var is = require('isomorph/is')

function extend..
```

```
function mixin...
function inheritPrototype...
function inheritFrom...

var Concur = exports.Concur = function...
Concur.extendConstructor = function...
```

/node_modules/isomorph/is.js:

```
function isArray...
function isBoolean...
function isDate...
function isError...
function isFunction...
function isNumber...
function isObject...
function isRegExp...
function isString...
function isEmpty...

module.exports = {
  Array: isArray
, Boolean: isBoolean
, Date: isDate
, Empty: isEmpty
, Error: isError
, Function: isFunction
, NaN: isNaN
, Number: isNumber
, Object: isObject
, RegExp: isRegExp
, String: isString
}
```

Expected output (not tested, rough guesses):

```
;(function() {
  var modules = {}
  // Naive much?
  function require(name) {
    return modules[name]
  }
  // Doesn't handle exports = blah
  function defineModule(name, fn) {
    var module = {}
      , exports = {}
    module.exports = exports
    fn(module, exports, require)
    modules[path] = module.exports
  }

  defineModule('isomorph/is', function(module, exports, require) {
function isArray...
function isBoolean...
function isDate...
function isError...
function isFunction...
function isNumber...
```

```
function isObject...
function isRegExp...
function isString...
function isEmpty...
module.exports = {
  Array: isArray
, Boolean: isBoolean
, Date: isDate
, Empty: isEmpty
, Error: isError
, Function: isFunction
, NaN: isNaN
, Number: isNumber
, Object: isObject
, RegExp: isRegExp
, String: isString
}
  })

  defineModule('concur', function(module, exports, require) {
var is = require('isomorph/is')

function extend..
function mixin...
function inheritPrototype...
function inheritFrom...

var Concur = exports.Concur = function...
Concur.extendConstructor = function...
  })

  window['concur'] = require('concur')
})
```

Research

- https://github.com/substack/node-browserify of course!
- https://github.com/LearnBoost/browserbuild - specify what you need and it handles relative stuff
- https://github.com/hij1nx/codesurgeon can be used to programmatically pick scripts apart to remove bits you don't want in the browser, rather than always writing in a browser-compatible way or shimming.
  - Example: https://github.com/flatiron/broadway/blob/master/bin/build
- https://github.com/visionmedia/mocha/tree/master/support

  Mocha has a build script which provides a browser-side `require()` and scans code being bundled for fixups for the browser, such as pointing at at stubs and shims for Node modules (which are kept in the */browser/* dir) and modifying inheritance to work cross-browser.

  It registers each file being bundled with its custom `require` by filename, wrapping it in a function which provides `module`, `exports` and `require` variables for the module to use.

  Any browser-specific setup is performed after the module is required into a global variable on `window`.

  `require()` is a refactored version of:

  - https://github.com/weepy/brequire/blob/master/browser/require.js

- https://github.com/tobie/modulr-node
- https://github.com/rpflorence/commonjs-rjs
- https://github.com/coolaj86/node-pakmanager - ruh-roh, wants to remove your opt-in rights to strict mode
- https://github.com/azer/onejs - "transform commonjs packages into single, stand-alone javascript files"
- https://github.com/medikoo/modules-webmake
- https://github.com/rolandpoulter/node_modulator
- http://caolanmcmahon.com/posts/writing_for_node_and_the_browser
- http://ender.no.de/

Extending The Sphinx JavaScript Domain

Notes exploring changes which could be made to the Sphinx JavaScript domain to take full account of JavaScript's capabilities and object model.

[TODOs/TBDs are expressed in square brackets]

Current JavaScript Domain

The JavaScript domain (name **js**) currently provides the following directives:

`.. js:function:: name(signature)`
Describes a JavaScript function or method.

`.. js:class:: name`
Describes a constructor that creates an object.

`.. js:attribute:: object.name`
Describes the attribute *name* of *object*.

`.. js:data:: name`
Describes a global variable or constant.

These roles are provided to refer to the described objects:

:js:func:
:js:class:
:js:data:
:js:attr:

Terminology

[Is there an set of single words, or two word phrases at a push, we could agree to use to differentiate between properties of a constructor, a prototype and an instance? The following are commonly used.]

class []

prototype []

variable []

object []

function []

method []

instance []

static []

extends []

inherits []

Necessary Capabilities

What are the full range of properties a JavaScript constructor, instances created from it and regular Objects can have?

```
var data = 0

var object = {
  prop: 42
, func: function() {}
}

function Constructor(arg1) {
  this.instanceProperty1 = arg1
  this.instanceProperty2 = Array.prototype.slice.call(arguments, 1)
  this.instanceFunction = function() {}
}
Constructor.constructorFunction = function() {}
Constructor.constructorProperty = true
Constructor.prototype.prototypeFunction = function() {}
Constructor.prototype.prototypeProperty = 42

var instance = new Constructor('steve', 1, 2, 3)
```

Proposed JavaScript Domain

`.. js:constructor:: name(signature)`
Describes a constructor that creates an object.

The underlying implementation for :js:class:: is already called JSConstructor

- keep js:class as an alias?

`.. js:prototype:: name`
Describes a constructor's prototype.

`.. js:object:: name`
An Object which contains... stuff.

`.. js:function:: name(signature)`
Describes a JavaScript function.

When top level:

name(signature) Top-level function

constructor.name(signature) Constructor function – "static"

constructor.prototype.name(signature) Prototype function – "method"

object.name(signature) Object function – "static"

When nested under:

constructor Prototype function – “method”

prototype Prototype function – “method”

object Object function – “static”

[What about functions attached directly to instances?]

`.. js:property:: name`
Describes a property of an object.

[What does nesting mean?]

[Constructor “static” property vs. prototype property vs. instance property]

[More...]

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

J