JavaScript on JavaScript

Jim Baker

jim.baker@{python.org, rackspace.com}

Overview

- Basics of syntax, semantics and applying to JavaScript
- Exploration in JS console, using Node 5 - mostly syntax and how to parse
- Unit tests for the (TDD) win!
- Corresponding interpreter implementing JS semantics
- Implement numeric ops, conditionals, assignment, function definition, and recursive call support in 135 LOC

About me

- Architect at Rackspace, focused on overall platform
- Lecturer in CS, CU Boulder teaching CSCI 3155
- Languages I often use: Python, Java, Scala, JavaScript, ...
- But really just a fan of languages!
- Core developer of Jython
- Co-author of *Definitive Guide to Jython* from Apress
- Previous jobs include Canonical (worked on Juju), Sauce Labs

JavaScript on JavaScript

- How to parse JavaScript into. . .
- . . . JSON (but of course!)
- then interpret using a simple evaluation model (big step operational semantics)
- Use Esprima for parsing, Chai for assertions, Mocha for test discovery & running

Or just use eval

Built into JavaScript, since the very beginning (at least 1996):

```
eval('6 * 7')
```

Or just use eval

Does show what we could support:

```
eval('(function (x) { return x + 1 })(8)')
```

Or just use `eval`

- Great way to introduce cross site scripting (XSS) attacks on your site, if you use `eval` with untrusted text!
- But JSON was successful in part because `eval(someJSON)` worked - and worked very efficiently
- Key observation by Douglas Crockford (the "good parts")

Related projects

- Sandboxing with JS in JS
- PyPy - Python on Python - used to be 1000x slower, now it can be 20x faster
- Self-hosting compilers - GNU C, javac (but Hotspot JVM uses C++ at its core), ...

Credits

- Some aspects based on Principles of Programming Languages (CSCI 3155) at CU Boulder
- JavaScript interpreter, written for labs developed by Evan Chang at Univ of Colorado, Boulder
- Judgment forms to define big step AND small step operational semantics
- explore dynamic and lexical scoping
- adding static typing similar to MS TypeScript
- Expressed on Scala
- Modified for this talk to be in terms of unit tests and written in JavaScript itself

ECMAScript 6

Will show ECMAScript 6 (aka ECMAScript 2015, JavaScript 6, ...)

- (Mostly) supported by Chrome 46, Firefox 44, MS Edge, Node 5
- Get to use `let`, `const`, along with other great functionality
- Today's example code may require "transpilation", polyfill, or manual changes to use with your needs
- Use Babel for compilation to older JS 5

What is a programming language?

- PLs are systems in which we can write programs/code/...
- Well-defined (more or less!) what we will get
- And we can collaborate on - we can read code together, make changes, “fork me on GitHub”

Key aspects

- Syntax
- Semantics

Syntax

- **Shape of the code**
- Higher order aspects, such as scoping of names
- e.g. being able to block scope variable assignment (new with `let`, `const`)
- Dealing with ambiguity - think PEDMAS precedence in middle school arithmetic
- JavaScript is often called a Lisp with C syntax. . .
- and Lisp itself doesn't really have a syntax

Semantics

- Functional, imperative, declarative, logical, ...
- Concurrency, mutation, software transactional memory, ... pick your favorite
- As constrained (or not) by typing (static, dynamic, gradual)

Node console

But first, install standard parser package

```
$ npm install esprima
```

Even better

Standard NPM package layout, with doc additions:

```
.
|-- README.md
|-- command.js
|-- examples
|   |-- fact.js
|   `-- six_x_seven.js
|-- lib
|   `-- index.js
|-- package.json
|-- talk.js
`-- talk.md
```

GitHub repo: https://github.com/jimbaker/js-on-js

Dependencies

Relevant `package.json` snippet:

```
"dependencies" : {
  "esprima": "latest",
  "chai": "latest",
  "mocha": "latest"
},
```

Interpreter available from command line

Should be able to support interepreter:

```
$ jsonjs examples/foo.js
```

CLI implementation

Abstract Syntax Trees (ASTs)

Represents a parse of an expression, including up to a program:

- In Esprima (and most likely most JS parsers), ASTs are represented as JSON
- Recursive - expressions (and similar ideas like a program body) are made of expressions
- Verbose
- But not everything is captured - specifically not the concrete syntax representation

ASTs with Esprima

In Node, use `.load` command:

```
> .load ./talk.js
```

Writing an interpreter

- Basics are we transform expressions to expressions
- Simplifying on each step
- Interpreters implement different approaches
- Choose an easy one: **evaluate**, with big step operational semantics
- Other choices covered in CSCI 3155 (!)

Compare to a compiler

- Compilers translate to some other language
- Often called “transpilers” if output is still high-level (but still a compiler)
- May be run by an interpreter such as the JVM or JavaScript runtimes - but generally complemented with just-in-time (JIT) compilation

Unit testing

- Unit testing can capture operational semantics
- Not as complete (in some ways) as mathematical formalisms like judgment forms
- But can capture hard-to-specify details
- Easy to go from specification of what the operation should be to a test

Just code now

- Look at unit tests
- And corresponding implementation
- Loop until complete :)