JACL: JavaScript Assisted Common Lisp

Alan Dipert

European Lisp Symposium

28 April 2020

Agenda

1. Setting the Scene
    a. My personal road to JACL
    b. JavaScript platform capabilities, limitations
2. Technical overview
    a. Interoperation
    b. Compiler
    c. Residential development
        i. Asynchronous reader
3. Demonstration
    a. Interacting in the browser
    b. Devtools Protocol REPL

My Road to JACL

- ~1994: First heard about Lisp from my father
- 1995-2008: Visual Basic, PHP, Java
- 2009: Clojure
- 2012: ClojureScript
  - As a mostly static subset of Clojure, Web development in ClojureScript required a lot of JVM-based tooling in the host environment.
  - Used Google Closure Compiler to compile generated JavaScript into reasonably small and efficient executables.
- 2016: Common Lisp tinkering
- 2019: JACL

The JACL Ideas

- Started on a new Lisp dialect, *Betalisp* around 2018
  - Test vehicle for a Lisp with macros that received arguments macro-expanded by default.
  - Didn't get far but stumbled on the utility of an asynchronous reader.
- Picked up a copy of CLtL2, perused COMMON-LISP@SU-AI.ARPA
- Implementing a spec is wonderfully "gamified"
  - Able to work intermittently and in short spurts and still make continuous progress.
- JACL could be a great prototyping substrate for the language-level ideas I have for improving web development.
- Opportunity to build out tooling in the browser via asynchronous reader, perform *residential development*
- Can't hurt to have a handy CL I can use for work stuff.

The JavaScript Platform

- Brings real programming to the browser environment.
- Makes applications feel more responsive to end users.
- Garbage collection, tracing JIT.
- Document Object Model (DOM) for interface construction and output.
- Heavily restricted for security reasons
  - Sandboxed.
  - No access to the host filesystem.
- Single process with *event loop*-based input model.

Technical Overview

- Interoperate directly and efficiently with JavaScript.
  - `JACL:%CALL` to call JavaScript functions
  - `JACL:%DOT` to access JavaScript object fields
  - `JACL:%JS` special form to generate arbitrary JavaScript code
- Generate reasonably small and efficient code.
  - Ideally I'll be able to use it for work some day.
  - Leaning heavily on existing JavaScript runtime features saves greatly on code size.
  - Analyzing compiler with a few optimizations.
- Support residential development.
  - Run more development tools in the browser and fewer on the host.
  - Anything that works in the browser during development will be available in production.
  - A browser-based IDE is too much for me to take on, but should never be impossible.
    - Emacs and inferior-lisp in the meantime.

Asynchronous Reader

- Lisp readers typically recursive.
  - Runtime stack used to store intermediate object structure.
  - Reading is a blocking operation.
- Restrictive JavaScript input model inhibits classical approach.
  - All input data must be sent to JavaScript through callbacks.
  - Any Intermediate object structures must be stored on the heap.
  - A blocking reader deadlocks the browser.
- Asynchronous reader.
  - Stateful object with a method to “push” characters from callbacks.
  - Incrementally parses characters and invokes a callback when a Lisp datum has been read.
  - General facility for incrementally parsing characters in JavaScript from any source without locking the entire page.

Demonstration

- Interacting in the browser
  - Reading Lisp objects
  - Compiling Lisp code
  - Inspecting Lisp objects and packages
- Developing in Emacs
  - Introduce the `jacl` REPL client
  - Evaluating Lisp expressions

Resources and Future Work

- https://tailrecursion.com/JACL/
  - JavaScript (runtime and compiler)
  - R (for the REPL client)
  - Lisp (for the nascent `CL` and `JACL` packages)
- Coming up
  - Building out the CL package
  - Toward the LOOP macro
  - Dumping images & tree-shaking
  - ABCL REPL client