Doppio:
Java meets Coffee in the Browser

CJ Carey, Jonny Leahey, Jez Ng

What?

- A Java Virtual Machine, in the browser
- Implemented in Coffeescript
- Tested against Java 1.6
  - runs many headless Java programs, including javac4 and Rhino (JS in Java)

Why?

- Atwood's Law
    - "any application that can be written in JavaScript, will eventually be written in JavaScript"

Why?

- Java in the browser is dead
  - Flash and HTML5 killed it
  - OSes don't ship Java, browser vendors don't promote it
  - iOS and Android browsers don't support it
  - numerous 0-day exploits
- Applets are still pervasive
  - especially in math, science, and education
  - demos of native-app functionality
  - games

Structure

- Two front ends:
  - console, via Node.JS
  - browser, which emulates part of the Node API
- Two execution modes:
  - disassembler (emulating `javap`)
  - runner (emulating `java`)

Structure

- Depends on the Java Class Library
  - almost all JRE methods are run directly
  - class files loaded on-demand
- Native methods are implemented in Coffeescript

Demo

Memory Model

- JVM bytecodes do not rely on memory layout
- No pointer arithmetic required, pointers only used as object IDs
- Allows Java objects to be mapped directly to JS objects with an '`$id`' field

Memory Model

- Javascript handles all heap management
  - free GC!
- We explicitly store a call stack for each thread
- Filesystem operations are mocked in the browser using LocalStorage

Threading

- Basic support is in place now
- Only one-at-a-time execution
    - may use WebWorkers eventually

Interesting Challenges

- Loading the JRE in the browser
- Javascript blocks the UI rendering thread
- Working with asynchronous JS functions

Interesting Challenges

- Extracting Java from the traditional OS-centered model
- Debugging support for Java exceptions
  - as well as CS/JS errors
- Handling different JRE distributions

Thoughts on Coffeescript

- Concise, but readable
  - if a consistent style is enforced
- Sometimes painful to debug
  - source maps in future implementations will help

Thoughts on Coffeescript

- Promotes using the Good Parts of JS
- Viable for large projects, but maybe not large teams

Thoughts on Node.JS

- Invaluable for automated testing
- Served as a model for interfacing with traditionally native resources
  - we wrote a browser equivalent of the node 'fs' API

Future Work

- Simple solution for running applets without a Java plug-in
- Profile, examine, and tune Doppio to specific JS implementations
- Java graphics emulation with Canvas

Future Work

- Compile `.class` → `.js`
- Use a cloud storage service in place of a local file system
- Stepping debugger

echo $?

fork us at https://github.com/int3/doppio