JRuby and Rails

The entirety of this presentation is licensed CC-by-SA

The Future of JRuby and Rails

The Present and Future of JRuby and Rails

The Present and Future of JRuby and the Future of Rails as relates to JRuby

Introductions

Me

- Charles Oliver Nutter
- Java dev since 1997
- Rubyist, JRuby dev since 2004
- Full time JRuby dev since 2006
- headius@headius.com, headius on Twitter
- blog.headius.com

JRuby

- Current version: 1.3.1
- Ruby 1.8.6 compatible (give or take)
- Some Ruby 1.9 support
- Solid performance (≈Ruby 1.9)
- Real native threads
- Runs Ruby/Rails great!

The Present

Performance

- Lies, damned lies
- Startup is very poor (0.5s to several s)
- Execution usually better than 1.9
- Core classes are mixed
  - Apps tend to need fast core classes
- Specific libraries can doom us

Native Threads

- Only "complete" impl with parallel threads
- Just do Thread.new { }
- Real scaling across cores, in-process

FFI

- Foreign Function Interface for C libraries
- Nothing new in C world
- Rubinius put a nice API on it
- JRuby adopted API, released CRuby gem
- Growing popularity

Rails

- Generally, “just works”
- Almost all Rails core tests pass
- ActiveRecord is a challenge...
- Deployment is “pretty good”
- Integration with Java needs work

Demo: Running Typo

- Already done:
  - Unpack (gem has native dependencies)
  - Configure database.yml
  - Prepare database (create, migrate)
- Ready to deploy!

Java Integration

- Generally as easy as calling Ruby
- Massive number of libraries
- No porting, no building
- Makes working with Java actually *fun*

Ruby 1.9 Support

- All in one with --1.9 flag
- Maybe 80% of 1.9.1
  - 1.9.2 adds a bunch more
  - Interested in helping?
- 1.9 support desired?

The Future

What's Missing?

- Finish 1.9 support
- More performance (neverending!)
- Better Java integration
- Adapting Rails to the Java platform
- Rubifying popular Java libraries
  - Hibernate, JPA, Maven, Ant, JAXP, EE, ...

Performance

- JSR 292 “invokedynamic”
  - Fast dynamic calls for the JVM
- New JRuby optimizing compiler
- Continuing optz for core classes
- Faster Java integration

Java Integration
(Mind the gap!)

- Improving existing call layers
  - Perf, memory, GC, coercion, wrapping
- Offline generation of Java classes
  - For "meta" APIs using annotations
- Runtime generation of Java classes
  - Many APIs just want a java.lang.Class obj

ruby2java

- Generates a Java .class
- Uses *runtime* definition of class
- Embeds source directly in .class

“become_java”

(working title)

- Generates a Java class in-memory
- Uses runtime definition of class
- Ruby object is instance of the Java class
- Annotations, signatures, whatever

Rubifying Java Libraries

- Persistence
  - Hibernate obviously; JPA to follow
- Build tooling
  - Ant and Maven really need help
- Services and components
  - Ruby everywhere, even in EE servers(?!?)

Hibernate

- Most extensive ORM package around
- Wide DB support
- Transactions, procedures, prepared stmts
- Caching, pooling, object identity
- DB-agnostic query language
- De-facto standard in Java world

Jibernate!

- Proof-of-concept Hibernate support
- Normal Ruby classes
  - ...turned into POJOs!
- Basically the beginner Hibernate tutorial
- Groundwork for the future
  - I build plumbing, you build porcelain

What's Next?

- The future of Ruby depends on JRuby
  - Java community is gigantic
  - Big businesses *will not* use CRuby
- Java escapees are going back
  - Groovy, Clojure, Scala
  - Libraries, ecosystem

How Can You Help?

- Use JRuby every chance you get
- Help us improve JRuby (1.9 help, please!)
- Start evangelizing at *Java* conferences
- Study Groovy, Scala, and help us compete
- Study Java libraries and help us Rubify

More Info

- www.jruby.org
- wiki.jruby.org
- www.kenai.com/projects/jruby
- blog.headius.com, headius@headius.com
- github.com/headius/railsunder