JRuby
Java and Ruby together under one VM

Cliff Moon
Chariot Solutions

CRUSH THE PYTHON!

PRODUCED BY THE PEOPLE'S COMMITTEE FOR REVOLUTIONARY RE-EDUCATION

What is JRuby?

- A 100% Java implementation of the Ruby 1.8.5 runtime.
- An embeddable scripting engine compatible with the BSF and the JSR 223 (Java 6) scripting frameworks.

JRuby is not...

- intended to displace MRI, YARV, or Rubinius.
- interested in changing the syntax of the ruby language.

How does JRuby differ from MRI?

- Threading
  - MRI uses green threads (due to change in ruby 2)
  - JRuby uses Java threads
- Continuations - Kernel.callcc
  - MRI supports them (possibly to change in ruby 2)
  - JRuby does not
- fork()
  - MRI supports this on posix platforms
  - JRuby does not

How Do Ruby and Java Get Along?

The Nuts and Bolts of Integration

- Java calling into Ruby
  - Ruby objects are just AST's
  - The JVM does not recognize Ruby classes
  - Proxies enable invocation
- Ruby calling into Java
  - Load Java objects via scripting framework
  - JRuby converts camelCase to under_score
  - Getters and setters are converted to accessors
  - Require can load JAR's

Loading Jars

```
if RUBY_PLATFORM =~ /java/
  require 'find'

  Find.find("/my/lib/folder/") do |jar|
    if File.basename(jar) =~ /(.*\.jar)$/
      require jar
    end
  end
end
```

Arrays and Collections

- Java arrays are distinct from Ruby arrays
  - Conversion is not automatic
  - Convenience methods are provided
- Java collections are distinct from Ruby arrays
  - Conversion are automatic
  - Convenience methods are provided, though mostly unnecessary

Array Conversions

```
irb(main):004:0> java = [10, 22, 33].to_java(:int)
=> #<#<Class:01x171cdc>:0x2c065 @java_object=[I@a531a9>
irb(main):005:0> java[2]
=> 33
irb(main):006:0> java.to_a
=> [10, 22, 33]
```

Collection Conversions

```
irb(main):009:0> list = ArrayList.new([1, 2, 3, 4])
=> #<Java::JavaUtil::ArrayList:0x1d3edd @java_object=[1,
  2, 3, 4]>
irb(main):010:0> list.to_a
=> [1, 2, 3, 4]
irb(main):011:0> list.add_all([5, 6, 7])
=> true
irb(main):012:0> list.to_a
=> [1, 2, 3, 4, 5, 6, 7]
```

Of Bytes and Strings

- Ruby has no concept of byte arrays
- Strings are used instead of byte arrays
- Java API's often take and return byte[]
- Convenience methods for conversion
  - String::from_java_bytes
  - String#to_java_bytes

Byte[] -> RString

```
irb(main):026:0> String.from_java_bytes [83, 119, 101,
  101, 101, 101, 116, 46].to_java(:byte)
=> "Sweeeet."
```

RString -> Byte[]

```
irb(main):027:0> "Frankenstein".to_java_bytes
=> #<#<Class:01xf8297b>:0x60efe7 @java_object=[B@df7713>
irb(main):028:0> "Frankenstein".to_java_bytes.to_a
=> [70, 114, 97, 110, 107, 101, 110, 115, 116, 101, 105,
  110]
```

Hibernate

- Cannot Persist Ruby Classes (yet)
  - JRuby classes are just AST's
  - Hibernate has lots of hooks
  - Possible, but worth it?
- Already have a Java domain model?
  - Load as a Jar
  - Call hibernate api from ruby
  - Does not play nicely with AR

Hibernate Conn Mgmt

```
class ApplicationController < ActionController::Base
  def hibernate_session
    unless @hibernate_session
      @@session_factory ||= AnnotationConfiguration.new().configure().build_session_factory()
      jdbc_connection = ActiveRecord::Base.connection.raw_connection.instance_eval('@connection')
      @hibernate_session = @@session_factory.open_session(jdbc_connection)
    end
    @hibernate_session
  end

  def cleanup_hibernate_session
    return unless @hibernate_session
    @hibernate_session.flush
    @hibernate_session.close
    @hibernate_session = nil
  end

 after_filter :cleanup_hibernate_session
end
```

Rails

- The vast majority of rails runs perfectly
- ActiveRecord-JDBC project
- Integration through the Goldspike project
- Multiple deployment options
  - jruby script/server - Webrick
  - Command line Jetty
  - Mongrel with the mongrel-support project
  - WAR for deployment in app server

Installing Rails Integration

```
jem install activerecord-jdbc
jem install rails
rails jruby_test
cd jruby_test
./script/plugin install svn://rubyforge.org/var/svn/jruby-
   extras/trunk/rails-integration/plugins/goldspike
jake war:standalone:create
jake war:standalone:run
```

Gotchas

- Path issues: gem, rails, jake, mongrel, etc.
  - ln -s gem jem ...
- ./script/* will invoke MRI
  - jruby ./script/server
- JDBC driver loading
  - JAVA_OPTS="-Xbootclasspath/p:/path/to/driver.jar" jruby script/server

Graphing

- Scruffy and Gruff are great unless...
  - the canned graphs don't suit your needs
  - you have no access to imagemagick

Adding JFreeChart

In config/war.rb:

```
maven_library('jfree', 'jcommon', '1.0.8')
maven_library('jfree', 'jfreechart', '1.0.4')
```

Graphs Controller

```
include Java

import org.jfree.data.xy.DefaultOHLCDataset
import org.jfree.data.xy.OHLCDataItem
import org.jfree.chart.ChartFactory
import org.jfree.chart.ChartUtilities

class GraphsController < ApplicationController

  def show
    @ds = DefaultOHLCDataset.new("Price", random_gook)
    @chart = ChartFactory.create_high_low_chart("RB", "Date", "Price", @ds, false)
    @image = @chart.create_buffered_image(params[:width].to_i, params[:height].to_i)
    @bytes = ChartUtilities.encodeAsPNG @image
    @rstring = String.from_java_bytes @bytes

    send_data @rstring, :type => 'image/png', :disposition => 'inline'
  end

  private

  def random_gook
    (1..100).collect do |i|
      OHLCDataItem.new(i.days.ago, rand(100), rand(100), rand(100), rand(100), rand(1000))
    end.to_java(OHLCDataItem)
  end
end
```

Questions?

Links

- JRuby
  - http://jruby.codehaus.org
- JRuby Extras
  - http://rubyforge.org/projects/jruby-extras