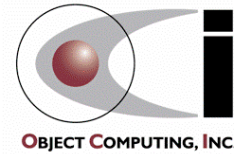


Ruby Plays Well With Others - Part 1



“You got Ruby in my Java!”
“You got Java on my Ruby!”
“Two great tastes that taste great together.”

Mark Volkmann
mark@ociweb.com



Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

Ruby Overview

- **Features**
 - object-oriented
 - dynamically-typed
 - compact, yet easy to read syntax
 - blocks that are closures
 - open classes and objects
 - language of Rails - web app. framework with DSL features
- **Current state**
 - supported by an interpreter implemented in C
 - no compiler
 - minimal optimization of parsed code
 - libraries
 - some implemented in Ruby and others in C
 - no formal language specification
 - small library of tests
 - somewhat slower than Python

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

2

Ruby Overview (Cont'd)

- **Future**
 - **Yet Another Ruby VM (YARV)**
 - new VM focused on performance
 - targeted for Ruby 2.0
 - implemented in C by Sasada Koichi
 - **Rubinius**
 - another Ruby VM focused on performance
 - patterned after Smalltalk VMs
 - implemented in C by Evan Phoenix
- **Resources**
 - **main web site:** www.ruby-lang.org
 - **books**
 - “Programming Ruby: The Pragmatic Programmers’ Guide, Second Edition” - referred to as “the pickaxe”
 - “Agile Web Development with Rails”
 - many more

JRuby Overview

- **Ruby on JVM**
 - Ruby interpreter written entirely in Java
 - can use Java capabilities from Ruby
 - can use Ruby capabilities from Java
- **Current state**
 - supports all Ruby syntax and built-in libraries and supports most standard libraries
 - retained Ruby libraries implemented in Ruby
 - many Ruby libraries that are implemented in C have been reimplemented in Java
 - these and other Ruby tools/libraries work with JRuby
 - Active Record (with JDBC), DRb, Rake (Ruby’s answer to Java’s Ant), Rails, RSpec (behavior-driven development), RubyGems
 - currently slower than C-based interpreter

Many languages are implemented on the JVM:

BeanShell
Bex (BeanShell variant)
Groovy
Jaskell (Haskell)
Jawk (AWK)
JudoScript
Jython (Python)
JRuby (Ruby)
Pnuts
Quercus (PHP)
Rhino (JavaScript) ...
SISC (Scheme)
Sleep (Perl/Objective-C)
Jacl (TCL)
and more

See list at
<http://scripting.dev.java.net/>

JRuby Overview (Cont'd)

- **Current focus**
 - creating a formal specification for the language and libraries
 - see headius.com/rubyspec
 - creating a larger library of tests
 - improving compatibility with standard Ruby
 - improving performance of interpreter
 - getting Rails to run under JRuby
 - working on passing existing Rails unit tests
 - writing a Ruby to Java bytecode compiler
 - initial results are about twice as fast as C-based Ruby interpreter
- **Future**
 - 1.0 release is expected in May 2007
 - in time for announcement at JavaOne, May 8-11
 - continue improving interpreter and compiler
 - a mixed mode is expected where some code is compiled and some is interpreted

.NET Ruby Implementations

- **RubyCLR**
 - a Common Language Runtime (CLR) bridge
 - from John Lam, hired by Microsoft 1/2007
 - see www.iunknown.com/articles/2006/10/20/dynamic-languages-microsoft-and-me
 - www.rubyclr.com
- **Gardens Point Ruby.NET Compiler**
 - a compiler, not an interpreter, implemented in C#
 - from Queensland University of Technology in Brisbane Australia
 - funded by Microsoft
 - www.plas.fit.qut.edu.au/rubynet
- **IronRuby**
 - a Ruby interpreter, similar to IronPython, implemented in C#
 - started by Wilco Bauwer, a Microsoft intern until 12/2006
 - reimplemented by Jim Hugunin?
 - announcement on 4/30/07 said will be available in a few months
 - http://www.iunknown.com/2007/04/introducing_iro.html

JRuby History

- **Stephen Mattias Aust**
 - ported the grammar from C-based Ruby to Jay, a Java-based parser
 - Jay is still used by JRuby
- **Jan Arne Petersen**
 - started JRuby project in 2001
 - built on work by Aust
- **Thomas Enebo**
 - began work in late 2002
 - became project lead in late 2003
 - moved JRuby from a 1.6 to a 1.8 implementation
- **Charles Nutter**
 - began work in 2004

JRuby History (Cont'd)

- **Sun Microsystems**
 - hired Nutter and Enebo to develop JRuby full-time in 9/2006
 - Tim Bray at Sun is a major advocate of dynamic languages
 - will remain open source
 - will provide more Ruby development tools
 - such as support in NetBeans
- **Other contributors**
 - Ola Bini became a committer on 10/3/2006
 - enabled high-performance YAML support in JRuby
 - implemented `Enumerable` in Java
 - Nick Sieger became a committer on 1/1/2007
 - original author of ActiveRecord-JDBC connector
 - over 35 developers are currently credited for contributing

Reasons To Use JRuby

- To use Java libraries from code written in Ruby syntax
 - for example, [Swing](#)
- To use Ruby libraries from code written in Java syntax
 - for example, [ActiveRecord](#)
 - two ways
 - Bean Scripting Framework (BSF)
 - “Scripting for the Java Platform” (JSR 223) in Java 6
- To get a faster implementation of Ruby
 - not yet faster than the current C-based Ruby, but likely will be soon
 - “Work is proceeding on the [JRuby compiler](#) and initial benchmarks are impressive. Although it’s not ready for prime time yet, and a lot of Ruby code can’t be compiled directly to Java bytecode yet, benchmarks that have been done show compiled JRuby code to be up to twice as fast as plain Ruby code running under Ruby C.”
 - from www.javalobby.org/java/forums/t89729.html

What JRuby Offers That Ruby Doesn’t

- Integration with Java libraries
 - JRuby classes can
 - extend Java classes
 - implement Java interfaces
 - add methods to existing Java classes
 - visible from JRuby, but not Java
- Native threads
 - JRuby uses native threads, Ruby uses green threads
 - can result in different behaviors between JRuby and Ruby
- Portability
 - JRuby runs on any machine with a JVM
- Unicode support
 - JRuby uses the unicode support in Java
 - Ruby has some support for unicode, but it’s not built-in

Can “sneak” JRuby into environments where installing the Ruby interpreter wouldn’t be allowed. JRuby can run using an already installed JRE and only requires an additional JAR file.

The Rails Wiki has a page on using Unicode strings (wiki.rubyonrails.com/rails/pages/HowToUseUnicodeStrings). See the `unicode` gem and `unicode_hacks` Rails plugin. Better support is coming soon. See redhanded.hobix.com/cult/yayMatzIsOnTheCuspOfUnveilingRubyUnicodeSupport.html.

Current Limitations

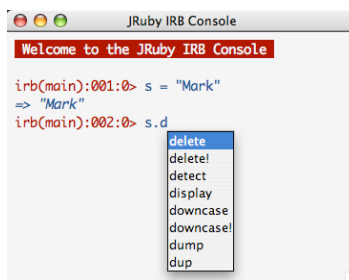
- JRuby classes can't ...
 - implement more than one Java interface this limitation will be removed soon
- Java classes can't ...
 - inherit from a JRuby class
- Performance
 - most code takes 2 to 3 times as long to run with JRuby as it does with C-based Ruby
- No debugger for JRuby code
 - Tor Norbye and Martin Krauskopf of Sun are working on adding integrated Java/JRuby debugging to NetBeans

Tool Support

- IDEs
 - many IDEs support Ruby including these
 - Eclipse
 - RDT plugin supports Ruby development
 - RadRails IDE (based on Eclipse) supports Rails development
 - IntelliJ IDEA 6.0
 - NetBeans - in work
- Editors
 - many editors offer Ruby support such as syntax highlighting
 - examples include emacs, jEdit, TextMate and Vim
- Spring 2
 - an IOC framework and more
 - supports beans implemented in Java, JRuby, Groovy and BeanShell

SuperConsole - Graphical IRB Console

- Similar to Ruby's Interactive Ruby (IRB)
- Available in several forms
 - executable JAR, Mac OS X application, Java Web Start
- Supports class and method name completion
 - activate with tab key
 - if more than one match is available, select from a popup list
- To download
 - browse www.jruby.org and select "JRuby Console" link



Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

13

Using JRuby From Command-Line

- Steps to install
 - download a binary release from www.jruby.org
 - unzip/untar the downloaded archive
 - set JRUBY_HOME environment variable to point to resulting directory
 - add \$JRUBY_HOME/bin to PATH
- Steps to use
 - `jruby {script-name}`
 - suggested file suffix is `.jrb` when using JRuby extensions; `.rb` otherwise
 - runs the class `org.jruby.Main` in `$JRUBY_HOME/lib/jruby.jar`
- Example
 - `hello.rb`

```
name = ARGV[0] || "you"
puts "Hello #{name}!"
```
 - run with `"jruby hello.rb"`; outputs `"Hello you!"`
 - run with `"jruby hello.rb Mark"`; outputs `"Hello Mark!"`

To checkout from the trunk of the Subversion repository,
svn co <http://svn.codehaus.org/svn/jruby/trunk/jruby>
To build, simply run "ant".
Of course Java and Ant must be installed.

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

14

Using Java Classes in JRuby

- **Must include Java**
- Provide full names of Java classes to be used

option #1 - provide full name when using

```
frame = javax.swing.JFrame.new("My Title")
```

option #2 - assign full class name to a constant

```
JFrame = javax.swing.JFrame  
frame = JFrame.new("My Title")
```

option #3 - use `include_class`

```
include_class "javax.swing.JFrame"  
frame = JFrame.new("My Title")
```

option #4 - use `include_class` with an alias

```
include_class("java.lang.String") do |pkg_name, class_name|  
  "J#{class_name}"  
end  
msg = JString.new("My Message")
```

option #5 - use `include_package`

```
module Swing  
  include_package "javax.swing"  
end  
frame = Swing::JFrame.new("My Title")
```

options 1 & 2 only work with classes in packages that begin with java, javax, com and org

useful when the name of a Java class matches the name of a Ruby class

`include_package` can only be used inside a module and has performance issues

Proxy Classes

- JRuby creates proxy classes for Java classes
 - allows methods to be added just like in Ruby
- Example

```
include Java  
include_class "java.util.ArrayList"  
list = ArrayList.new  
%w(Red Green Blue).each { |color| list.add(color) }  
  
# Add "first" method to proxy of Java ArrayList class.  
class ArrayList  
  def first  
    self.size == 0 ? nil : self.get(0)  
  end  
end  
puts "first item is #{list.first}"  
  
# Add "last" method only to the list object ... a singleton method.  
def list.last  
  self.size == 0 ? nil : self.get(self.size - 1)  
end  
puts "last item is #{list.last}"
```

Output

```
first item is Red  
last item is Blue
```

Method Calling Details

- Parens aren't required when calling methods
 - `foo.bar()` is the same as `foo.bar`
 - `foo.bar(baz)` is the same as `foo.bar baz`
- Can invoke Java get/set/is methods like Ruby accessors
 - `value = foo.getBar()` is the same as `value = foo.bar`
 - `foo.setBar(value)` is the same as `foo.bar = value`
 - `foo.isBar()` is the same as `foo.bar?`
 - but when invoking a method without a receiver ...
 - `bar` is interpreted as a reference to a local variable
 - to make it be interpreted as a method call, use `self.bar`
- Method naming conventions
 - can invoke camel-cased Java methods with those names
 - or use Ruby underscore convention

```
include Java
url = java.net.URL.new("http://www.ociweb.com")
puts url.to_external_form # method name is toExternalForm
puts url.to_uri # method name is toURI
```

underscore versions of methods
get added to JRuby proxy class

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

17

Automatic Conversions

- Listed below are some of the conversions between Ruby and Java types that happen automatically

Ruby types

- Boolean
- String
- Fixnum

- Float

- Array
- Hash

Java types

- boolean, java.lang.Boolean
- char, java.lang.String
- byte, java.lang.Byte,
short, java.lang.Short,
int, java.lang.Integer,
long, java.lang.Long
- float, java.lang.Float,
double, java.lang.Double
- java.util.List
- java.util.Map

- for more see code in the following files
 - `src/builtin/javasupport.rb`
 - `src/builtin/java/*.rb`
 - `src/org/jruby/javasupport/Java.java`
 - `src/org/jruby/javasupport/JavaUtil.java`

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

18

Ruby Methods Added to Core Java Classes

- `java.lang String` and type wrapper classes

- `<=>`

- mixes in Ruby's `Comparable` module which adds many methods defined in terms of `<=>`

`<, <=, ==, ==, >, >`
between?

- `java.util` collection classes

- `Collection` (base interface of `List` and `Set`)

- `each` (iterator)
- `<<` (append)
- `+` (adds another collection)
- `-` (removes another collection)
- mixes in Ruby's `Enumerable` module which adds many methods

- `List`

- `[], []=, sort` and `sort!`

- `Map`

- `each, []` and `[]=`

`all?, any?, collect, each_with_index, find, find_all, grep, max, min, sort, sort_by, etc.`

see `src/builtin/java/collections.rb`

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

19

ArrayList Example

```
include Java
list = java.util.ArrayList.new

puts list.kind_of? java.util.List
puts list.instance_of? java.util.ArrayList
puts list.class.ancestors
puts "list is a #{list.class.name}"
puts "list is a #{list.java_class}"

list << "Mark"
# Since everything in Ruby is an object,
# numbers and booleans can be added to Java collections.
list << 19
list << true
list.each { |element| puts element } # "each" method added to ArrayList

# The following line invokes the Java ArrayList#toString method
# instead of the Ruby Array#to_s method.
puts list
```

Output
false (a bug)
true
#<Class:01x3d511e> (a bug)
Enumerable
ConcreteJavaProxy
JavaProxy
Object
Kernel
list is a (a bug)
list is a java.util.ArrayList

↑ should have a name

Output
[Mark, 19, true]

Output
Mark
19
true

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

20

JRuby Inheriting From Java Classes

```
package com.ocicweb.demo;

public class Car {
  private String make;
  private String model;
  private int year;

  public Car() {}

  public Car(String make, String model, int year) {
    this.make = make;
    this.model = model;
    this.year = year;
  }

  public String getMake() { return make; }
  public String getModel() { return model; }
  public int getYear() { return year; }

  public void setMake(String make) { this.make = make; }
  public void setModel(String model) { this.model = model; }
  public void setYear(int year) { this.year = year; }

  public String toString() {
    return year + " " + make + " " + model;
  }
}
```

Java

Output

```
1997 Honda Prelude
2005 Ferrari F430 can go 196 MPH
1971 Porche 917 can go 248 MPH
```

```
include Java

Car = com.ocicweb.demo.Car

c = Car.new("Honda", "Prelude", 1997)
puts c

class RaceCar < Car
  attr_accessor :top_speed

  def initialize(
    make=nil, model=nil, year=0, top_speed=0)
    super(make, model, year)
    @top_speed = top_speed
  end

  def to_s
    "#{super} can go #{@top_speed} MPH"
  end
end

c = RaceCar.new("Ferrari", "F430", 2005, 196)
puts c

c = RaceCar.new("Porche", "917")
c.year = 1971
c.top_speed = 248
puts c
```

Ruby



Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

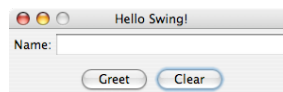
JRuby

21

Swing Demo

```
include Java

BorderLayout = java.awt.BorderLayout
JButton = javax.swing.JButton
JFrame = javax.swing.JFrame
JLabel = javax.swing.JLabel
JOptionPane = javax.swing.JOptionPane
JPanel = javax.swing.JPanel
JTextField = javax.swing.JTextField
```



Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

22

Swing Demo (Cont'd)

```
class BlockActionListener < java.awt.event.ActionListener
  # super call is needed for now - see JRUBY-66 in JIRA
  def initialize(&block); super; @block = block; end
  def actionPerformed(e); @block.call(e); end
end

class JButton
  def initialize(name, &block)
    super(name)
    addActionListener(BlockActionListener.new(&block))
  end
end

class HelloFrame < JFrame
  def initialize
    super("Hello Swing!")
    populate
    pack
    self.resizable = false
    self.defaultCloseOperation = JFrame::EXIT_ON_CLOSE
  end
end
```

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

23

Swing Demo (Cont'd)

```
def populate
  name_panel = JPanel.new
  name_panel.add JLabel.new("Name:")
  name_field = JTextField.new(20)
  name_panel.add name_field

  button_panel = JPanel.new
  greet_button = JButton.new("Greet") do
    name = name_field.text
    msg = %(<html>Hello <span style="color:red">#{name}</span>!</html>)
    JOptionPane.showMessageDialog self, msg
  end
  button_panel.add greet_button
  clear_button = JButton.new("Clear") { name_field.text = "" }
  button_panel.add clear_button

  contentPane.add name_panel, BorderLayout::CENTER
  contentPane.add button_panel, BorderLayout::SOUTH
end
end # of HelloFrame class

HelloFrame.new.visible = true
```

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

24

Installing Gems Under JRuby

- Gems are ...
 - the preferred mechanism for packaging, distributing and installing Ruby libraries and applications
- Use scripts in JRuby bin directory to work with them
 - for example, to install a gem
`$JRUBY_HOME/bin/gem install activerecord -y`
`-y` is equivalent to `--include-dependencies`
 - currently, generating rdoc and ri documentation from JRuby is slow
 - to avoid this when installing gems, include `--no-rdoc --no-ri`

Using Gems in JRuby

- Required setup
 - set the following system properties
 - `jruby.base=$JRUBY_HOME` for the ActiveRecord example, this is done in the Ant `build.xml`
 - `jruby.home=$JRUBY_HOME`
 - `jruby.lib=$JRUBY_HOME/lib`
 - `jruby.script={jruby for Unix variants, jruby.bat for Windows}`
 - `jruby.shell={/bin/sh for Unix variants, cmd.exe for Windows}`
 - set load path
 - append the following directories to the global array `$` :
 - `$JRUBY_HOME/lib`
 - `$JRUBY_HOME/lib/ruby/site_ruby/1.8` for the ActiveRecord example, done in my `JRubyHelper.java`
 - `$JRUBY_HOME/lib/ruby/site_ruby/1.8/java`
 - `$JRUBY_HOME/lib/ruby/site_ruby`
 - `$JRUBY_HOME/lib/ruby/1.8`
 - `$JRUBY_HOME/lib/ruby/1.8/java`
 - `lib/ruby/1.8`
- See ActiveRecord example later

Bean Scripting Framework (BSF)

- A Java library that
 - allows Java code to
 - evaluate code written in various scripting languages
 - allows scripting language code to
 - access Java objects
 - invoke Java methods
- “Scripting languages” with BSF engines include
 - Groovy, Javascript (Rhino), Python (Jython), Ruby (JRuby), JudoScript, NetRexx, ooRexx, ObjectScript, PROLOG (JLog), Tcl (Jacl), XSLT (Xalan and Xerces)
- Key methods in **BSFManager** class
 - **registerScriptingEngine**
 - **exec** - executes script code
 - **eval** - executes script code and returns its value
 - **declareBean** - creates an object in the context of a scripting language (global variable) that won't be retrieved with **lookupBean**
 - **registerBean** - adds an object to the object registry
 - **lookupBean** - gets an object from the object registry

JRuby From Java 5 - BSF

- Consider increasing maximum memory
 - `-Xmx256m`
- Setup from Java
 - classpath must contain these JARs from JRuby lib directory
 - bsf.jar
 - jruby.jar
 - jvyaml.jar - a Java YAML parser and emitter
 - register the JRuby engine

```
String language = "ruby";
String engineName = "org.jruby.javasupport.bsf.JRubyEngine";
String[] extensions = {"rb"};
BSFManager.registerScriptingEngine(
    language, engineName, extensions);
```
 - create a BSFManager

```
BSFManager manager = new BSFManager();
```

JRuby From Java 5 - BSF (Cont'd)

- **BSF beans - option #1**
 - in Java, [declare](#) Java objects as BSF beans
`manager.declareBean("frame", aFrame, JFrame.class);`
 - in Ruby, access beans through [global variables](#)
 - in this case, `$frame`
- **BSF beans - option #2**
 - in Java, [register](#) Java objects as BSF beans
`manager.registerBean("frame", aFrame);`
 - in Ruby, [lookup](#) registered beans via [\\$bsf](#) object
`$bsf.lookupBean("frame")`

JRuby From Java 5 - BSF (Cont'd)

- **Running Ruby code from Java**
 - to execute Ruby code with no return value
`manager.exec("ruby", "java", 1, 1,
"$frame.title = 'My Title'");`
 - to evaluate Ruby code, returning a value
`String sourceLang = "java";
String scriptLang = "ruby";
String scriptCode = "(1..5).collect {|e| e**2 }";
List<Long> squares = (List<Long>)
manager.eval(scriptLang, sourceLang, 1, 1, scriptCode);
for (long square : squares) {
 System.out.println(square);
}`

The "1, 1" parameters are line and column numbers that provide context info.

Output
1
4
9
16
25

Java Using ActiveRecord

- ActiveRecord is ...
 - a Ruby library for accessing relational databases
- Can be used from Java through JRuby
 - install the ActiveRecord gem as shown earlier
 - under Java 5 and earlier
 - use Bean Scripting Framework (BSF)
 - classpath must contain
bsf.jar, jruby.jar, jvyaml.jar, commons-logging-1.1.jar
 - under Java 6 and later
 - use JSR 223 Scripting API
 - we'll focus on BSF here
 - the classes **BSFHelper** and **JRubyHelper** were written to make this easier
 - get them from links on
<http://www.ocieweb.com/mark/programming/ActiveRecord.html>

models.rb

```
require "rubygems"
require "active_record"

ActiveRecord::Base::establish_connection(
  :adapter=>"mysql",
  :host=>"localhost",
  :database=>"music",
  :user=>"root", :password=>"" )

class Artist < ActiveRecord::Base
  has_many :recording
  # Sort based on artist name.
  def <=>(other); name <=> other.name; end
end

class Recording < ActiveRecord::Base
  belongs_to :artist
  # Sort based on recording name.
  def <=>(other); name <=> other.name; end
end
```

2003recordings.jrb

```
require "models"

# $recordings is a Java ArrayList that is created in Query.java
# and declared as a BSF bean.
# This code populates it with Ruby Recoding objects.

Recording.find_all_by_year(2003).sort.each do |r|
  $recordings << r
end
```

Query.java

```
package com.ocweb.activerecord;

import com.ocweb.bsf.BSFHelper;
import com.ocweb.jruby.JRubyHelper;
import java.util.*;
import org.apache.bsf.BSFException;
import org.jruby.*;

public class Query {

    private BSFHelper bsf = new BSFHelper();
    private JRubyHelper helper = new JRubyHelper();

    public static void main(String[] args) throws Exception {
        new Query();
    }

    private Query() throws BSFException, java.io.IOException {
        // Pass a Java object into Ruby code which will populate it.
        System.out.println("\n2003 Recordings");
        List recordings = new ArrayList();
        bsf.declareBean("recordings", recordings);
        bsf.evalFile("2003recordings.jrb");
    }
}
```

Query.java (Cont'd)

```
// Retrieve data that Ruby populated into recordings.
Iterator iter = recordings.iterator();
while (iter.hasNext()) {
    RubyObject recording = (RubyObject) iter.next();

    // The attributes of Recording are id, name, year and artist_id.
    String recordingName =
        (String) helper.getAttribute(recording, "name");

    // Get the Artist object associated with this Recording object.
    // What is the intermediate object here?
    RubyObject artist = helper.callMethod(recording, "artist");
    artist = (RubyObject) artist.getInstanceVariable("@target");

    String artistName = (String) helper.getAttribute(artist, "name");

    System.out.println(" " + recordingName + " by " + artistName);
}
}
```

```
Output
2003 Recordings
Soviet Kitch by Regina Spektor
Transatlanticism by Deathcab For Cutie
```

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

35

JRuby From Java 6 - JSR 223 Scripting API

- Setup
 - [install Java 6](#)
 - [download JRuby](#)
 - from www.jruby.org
 - works with 0.9.1, but not 0.9.2, what about 0.9.8?
 - [download scripting engines](#)
 - from <http://scripting.dev.java.net>
 - click “Documents & files” link
 - download `jsr223-engines` zip or tar
 - unzip or untar it
 - [add to classpath](#)
 - `jruby.jar` from JRuby download
 - `jruby-engine.jar` from scripting engines download

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

36

JSR 223 - Evaluating Scripts

- **ScriptEngine eval** method
 - takes a **String** of Ruby code or a **Reader**
 - to read from a file in the file system
`new BufferedReader(new FileReader(path));`
 - to read from a file in the classpath
`new InputStreamReader(
 ClassLoader.getSystemResourceAsStream(path));`
 - returns the return value of the script, if any

```
import javax.script.*;

public class JSR223Demo {

    public static void main(String[] args) throws ScriptException {
        ScriptEngineManager manager = new ScriptEngineManager();
        ScriptEngine engine = manager.getEngineByName("jruby");
        engine.eval("puts 'Hello World!'");
    }
}
```

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

37

JSR 223 - Invoking Functions and Methods

- **Steps**
 - load the script that defines the functions to be invoked using the **ScriptEngine eval** method
 - cast **ScriptEngine** to **Invocable**
`Invocable invocable = (Invocable) scriptEngine;`
 - optionally specify data to be made available to the scripting language through global variables
`invocable.put(name, value);`
 - reference in Ruby code with `$name`
 - doesn't work in some versions of JRuby
 - invoke a Ruby function or method

```
Object returnValue =
    invocable.invokeFunction(functionName [, params]);
Object returnValue =
    invocable.invokeMethod(object, functionName [, params]);
```

parameters can be any type

a script object returned by a previous invocation

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

38

JSR-223 Example - demo.rb

```
class Calculator
  def average_of_3(n1, n2, n3)
    (n1 + n2 + n3) / 3.0
  end

  def average(*array)
    sum = 0
    array.each { |n| sum += n }
    sum.to_f / array.size
  end
end

def getCalculator
  Calculator.new
end
```

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

39

JSR 223 Example - Demo.java

```
import java.io.*;
import javax.script.*;

public class Demo {

  public static void main(String[] args)
  throws IOException, NoSuchMethodException, ScriptException {
    ScriptEngineManager manager = new ScriptEngineManager();
    ScriptEngine engine = manager.getEngineByName("jruby");
    engine.eval(new BufferedReader(new FileReader("src/demo.rb")));

    Invocable invocable = (Invocable) engine;
    Object calculator = invocable.invokeFunction("getCalculator");
    double average = (Double) invocable.invokeMethod(
      calculator, "average_of_3", 1, 4, 5);
    System.out.println("average = " + average);

    average = (Double) invocable.invokeMethod(
      calculator, "average", 1, 4, 5);
    System.out.println("average = " + average);
  }
}
```

```
Output
average = 3.3333333333333335
average = 3.3333333333333335
```

Copyright © 2007 by Object Computing, Inc. (OCI).
All rights reserved.

JRuby

40

More Information

- **Web pages**
 - [project homepage - www.jruby.org](http://www.jruby.org)
- **Wikis**
 - www.headius.com/jrubywiki
- **Mailing lists**
 - see “Mailing Lists” link at www.jruby.org
- **Blogs**
 - Nutter’s - headius.blogspot.com
 - Enebo’s - www.bloglines.com/blog/ThomasEEnebo
 - Bini’s - ola-bini.blogspot.com
- **Podcasts**
 - Java Posse interview with Nutter and Enebo on 1/17/07
 - javaposse.com/index.php?post_id=171709
- **Books**
 - Nutter and Enebo plan to write a book on JRuby soon