

An Introduction to JRuby

Rob Di Marco

416 Software, LLC

<http://www.innovationontherun.com>

October 23rd, 2007

Preface

- What We Will Accomplish
 - Introduce JRuby to Java developers
 - Discuss how to develop with JRuby
 - Review situations where using JRuby may be advantageous
- What We Will NOT Do
 - Compare JRuby to other scripting languages
 - Jython, Groovy, Scala, et. al. are nice too!
 - Doing a comparison is a different JUG presentation
 - Look at JRuby as a silver bullet

Agenda

- What is JRuby
- Ruby Basics
- Working with Java and JRuby
- Meta-Programming and DSLs
- JRuby on Rails and J2EE

What is JRuby

- JRuby is a 100% Java implementation of Ruby.
 - Provides a Ruby 1.8.5 compatible interpreter
 - Most built-in Ruby classes provided
- Bean Scripting Framework (BSF) support
 - Can be used anywhere BSF is supported (e.g. Ant scripts)
 - Integrates with Java 6 Scripting API
- Distributed under a tri-license (CPL/GPL/LGPL)
- Strongly supported by Sun
 - A primary developer is employed by Sun
 - Glassfish and NetBeans 6.0 IDE integration

Why JRuby?

- Full implementation of Ruby
 - Vibrant and independent development community
 - Java developers can use the scripting and functional powers of Ruby
- Fully written in Java
 - Runs on any platform with a JVM
 - Easy integration with Java classes
- Active community with support from Sun

About Ruby

- Created by Yukihiro “matz” Matsumoto
- First public release in 1995
- Elements of many other languages
 - Perl, SmallTalk, Eiffel, Lisp

“I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python.”

“Ruby is simple in appearance, but is very complex inside, just like our human body”

Ruby Basics

- Everything is an object
- Dynamic (duck) typing
- Single inheritance
 - All objects descend from the base Object
 - JRuby allows for extending of Java objects and implementing Java interfaces
- Many functional programming concepts
 - *“in Ruby closures, I wanted to respect the Lisp culture”*

Some Ruby/Java Differences

- Variables
 - No type definition
 - Instance variables are defined using @
 - Static variables are defined using @@
 - Global variables are defined using \$
 - Variables starting with a capital letter are constants
- All member variables are private.
 - External access via methods.
 - Methods do not need to be defined, but can be
- It's nil instead of null
- Different object equals semantics
 - Java's equals is Ruby == (equivalence)
 - Java's == is Ruby's equals? (same object)
- Constructor is a method *initialize*

An Example Ruby Class

- Class definition
- Constructor
- Member variable
- Instance Methods
- Static Methods

```
1 class Greeting
2   @@count=0
3   def initialize (msg = "Hello")
4     @msg = msg
5   end
6
7   def say_hello_to_person(person)
8     @@count += 1
9     @msg + " " + person
10  end
11
12  def to_s
13    @msg
14  end
15
16  def self.what_is_count
17    @@count
18  end
19 end
```

Calling JRuby from Java

- JRuby API

```
import org.jruby.*;

public class SimpleJRubyCall {
    public static void main(String[] args) {
        Ruby runtime = Ruby.getDefaultInstance();
        runtime.evalScript("puts 1 + 2");
    }
}
```

- Java 6 Scripting API

```
import javax.script.*;

public class SampleJava6Execution {
    public static void main(String[] args)
        throws Exception {
        ScriptEngine engine
            = new ScriptEngineManager()
                .getEngineByName("jruby");
        engine.getContext()
            .setAttribute("val", 2, ScriptContext.ENGINE_SCOPE);
        engine.eval("puts 1 + $val");
    }
}
```

Using Java in JRuby

- *include Java*
- *require 'pathtojar'*
- Can access Java class by
 - Importing the class using *import*
 - Using the full class name
 - Assigning class name to a constant
- Implement interfaces using *include*

```
include Java

import javax.swing.JFrame
button = javax.swing.JButton.new("Klick Me!")
JOptP = javax.swing.JOptionPane

frame = JFrame.new("Hello Swing")

class ClickAction
  include java.awt.event.ActionListener
  def actionPerformed(evt)
    JOptP.showMessageDialog(nil, <<EOS)
    <html>Hello from <b><u>JRuby</u></b>.<br>
    Button '#{evt.getActionCommand()}' clicked.
    EOS
  end
end
button.add_action_listener(ClickAction.new)

# Add the button to the frame
frame.get_content_pane.add(button)

# Show frame
frame.set_default_close_operation(JFrame::EXIT_ON_CLOSE)
frame.pack
frame.visible = true
```

Java Object in JRuby

- Example: `java.io.File`
- All Java method names are available
- All methods also available with Ruby conventions (`_`, `?`)

```
irb(main):001:0> JFILE=java.io.File
=> Java::JavaIo::File
irb(main):002:0> f=JFILE.new '.'
=> #<Java::JavaIo::File:0xc38157 @java_object=.>
irb(main):003:0> f.isDirectory
=> true
irb(main):004:0> f.is_directory
=> true
irb(main):005:0> f.directory?
=> true
```

Advanced JRuby

Meta Programming and JRuby

More To Know About Ruby

- Method calls actually wrap the send method
 - `java.util.Date.new.to_s` is the same as `java.util.Date.send("new").send("to_s")`
- Implicit `method_missing`
 - Closest thing in Java is `java.lang.reflect.Proxy`
 - Exists on every object
- Code can be managed in modules
 - Modules may be included into objects
 - Creates instance methods in the object
 - Known as “mixins”

Code Blocks as a Variable

- Blocks of Ruby code can be used as variables
- Common use is looping

```
var=[1,2,3,4]

var.each do |i|
  puts i;
end

{:foo=>1, :bar=>2}.each do |k,v|
  puts "Key: #{k} is #{v}"
end

var.map! {|e| e+1}

def wrapper_block(key, &block)
  puts "Starting #{key}"
  yield
  puts "Ending #{key}"
end

wrapper_block 'Test' do
  puts "Hello"
  puts "world"
end
```

Modifying Code At Runtime

- Meta Programming
 - Code that writes code
- Add a method to class
- Add instance methods
- Add to Java classes!

```
class Foo
  def hello_world
    puts "Hello"
  end
end

Foo.new.hello_world

Foo.class_eval do
  def goodbye_cruel_world
    puts "Goodbye"
  end
end

Foo.new.goodbye_cruel_world

f = Foo.new

def f.how_is_this_possible
  puts "It works!"
end

f.how_is_this_possible
```

Modifying the Java Date class

- Add a new method to `java.util.Date`
- *tomorrow* exists for all Date objects
- Same technique can be used on all objects
 - Final objects like String can be modified

```
require 'java'
DATE=java.util.Date
DATE.class_eval do
  def tomorrow
    DATE.new(self.get_time + 24*3600000)
  end
end

d = DATE.new
puts "today: " + d.to_s
puts "tomorrow: " + d.tomorrow.to_s
```

Putting It All Together

- Create a DataRecord class will create a class from a comma delimited file
- Attributes are defined by the names of columns in a comma separated file
- `method_missing` is used to define dynamic method names
- Real JRuby classes are being created and can be used

Refer to [DevSource](#) for more explanation

Domain Specific Language

- DSLs designed to simplify coding
- Meta-programming empowers creating DSLs
- JRuby DSLs can leverage Java libraries
 - Example: Simple JRuby DSL on top of HtmlUnit

```
get_page(url) do |page|
  find_sample(page) do |div|
    find_span_with_class_detail_item(page) do |elements|
      elements.each do |e|
        puts e.as_text
      end
    end
  end
end
```

JRuby On Rails

Using JRuby to Merge Ruby
with J2EE

About JRuby on Rails

- Big driver in the recent popularity of Ruby
- Meta programming functionality critical to:
 - Core Rails API
 - Plugins that modify, enhance, and extend core.
- Almost a DSL for web development
- JRuby 1.0 fully supports Rails
 - Plugins add JDBC functionality
 - JRuby 1.1 focusing on performance enhancements
 - Can be deployed as a WAR
 - Still in development but tremendous potential

JRuby on Rails JUG Application

- About the application
 - Multi-User Link suggestion with voting
 - All users can see a list of links
 - Signed in users can add links and rate links
 - About ~75 lines of custom Ruby and ~100 lines of RHTML
- Using JDBC for Data Access
 - Can configure to use JDBC or JNDI data source
 - If JDBC, need JARs in JRuby classpath.
 - Putting JARs in \$JRUBY_HOME/lib works

Gems and Plugins extend Rails

- ActiveRecord-JDBC
 - Ruby Gem installed in JRuby lib directory
 - Configured to only be used with JRuby
- `acts_as_authenticated`
 - Used for user validation
 - Generates a module that is included in the main application controller
- `acts_as_voteable`
 - Makes an include call on `ActiveRecord::Base` to add its modules
 - Illustrates how Ruby meta programming really empowers Rails plugin developers

WAR Packaging

- Comes as Rails Plugin exists to get all dependent JAR files and bundle into a WAR
- WAR can then be deployed to an application server
- Rails applications can then connect in with other J2EE applications
- Great way to get benefits of Rails in an all Java environment

Resources

- JRuby home page
 - <http://jruby.codehaus.org/>
- JRuby documentation
 - <http://docs.codehaus.org/display/JRUBY/Home>
- Developer IRC
 - <irc://irc.freenode.net/jruby>
- Java Scripting Programmer's Guide
 - <http://java.sun.com/javase/6/docs/technotes/guide>
 - <https://scripting.dev.java.net/>
- JRuby Developer Blogs
 - <http://ola-bini.blogspot.com/> (Ola Bini)
 - <http://headius.blogspot.com/> (Charles Nutter)

Resources (Continued)

- JRuby Extras
 - http://rubyforge.org/tracker/?group_id=2014
- Ruby Documentation
 - <http://www.ruby-doc.org/docs/ProgrammingRuby/>