ThoughtWorks

Introduction to JRuby

Neal Ford
ThoughtWorker / Meme Wrangler
www.nealford.com
www.thoughtworks.com
nford@thoughtworks.com
memeagora.blogspot.com

NF

ThoughtWorks

Questions, Slides, & Samples

- Please feel free to ask questions anytime
- Slides and samples at www.nealford.com

NF

ThoughtWorks

What This Session Covers:

- Motivation for JRuby
- What makes Ruby cool?
  - Classes & objects
  - Closures
  - Mixins
  - Idioms
- Ruby + Java

ThoughtWorks

Why Ruby?

- *Purely* object-oriented
- Dynamically typed ("duck typed")
- Compact syntax
- Closures
- Open classes
- Awesome meta-programming support
- Rails

NF

ThoughtWorks

Why JRuby?

- JRuby == Ruby on the JVM
- Allows Java capabilities with Ruby syntax
  - Smart about properties
  - Access to Java libraries
- Ruby libraries from Java
- Supports all Ruby syntax and built-in libraries
  - Easy to port the libraries written in Ruby
  - C libraries ported to Java

ThoughtWorks

Motivations

- Use Java libraries (i.e., Swing) from Ruby code
- Use Ruby libraries (i.e., ActiveRecord) from Java
  - Using Bean Scripting Framework (JDK <= 1.5)
  - Using JSR 223 Scripting API (JDK > 1.5)
- Get an (eventually) faster Ruby
  - Much slower now
  - Lessons learned from IronPython

What JRuby Gives You

- JRuby classes can
  - Inherit from Java classes
  - Implement Java interfaces
  - Open Java classes (visible from JRuby, not Java)
- Running on the JVM provides
  - Native threads
  - Unicode Support
  - Portability
- Makes it possible to "sneak" it into corporate IT

Current Limitations

- JRuby classes can only implement 1 Java interface (fixed soon)
- Java classes can't inherit from JRuby classes
- Most code takes 2 to 3 x longer to run
- No debugger
  - Sun is adding good JRuby support to NetBeans
  - IntelliJ will support Ruby/JRuby in the next IntelliJ

ThoughtWorks

Some Ruby Syntax

- Just the interesting stuff
- Example
  - hr.rb
  - hr_runner.rb
  - Tests
    - test_employee.rb
    - test_manager.rb
    - test_all.rb

NF

ThoughtWorks

Ruby Blocks

- Delineated with either
  - { . . . } # idiomatically used for single line blocks
  - do . . . end # idiomatically used for multi-line blocks
- Both support parameters with |my_param|
- Examples
  - hr_blocks.rb
  - hr_blocks_ruby_way.rb

NF

ThoughtWorks

What Makes a Block a Closure?

- A closure object has
  - Code to run (the executable)
  - State around the code (the environment, including the local variables)
- The closure captures the environment
  - You can reference the local variable inside the closure...
  - ...even if the variable has gone out of scope
- Example
  - hr_closures.rb

ThoughtWorks

Important Closure Characteristics

- Closures combine a block of code with an environment
  - Sets is apart from function pointers, et al
  - Java's anonymous inner classes can access locals (only if they are final)
- Require very little syntax
  - Crucial because they are used all the time
  - Cumbersome to create something like an anonymous inner class all the time

ThoughtWorks

Modules

- Allow you to group classes, method, and constants
- Provide two major benefits
  - Namespaces
  - Mixins
- Mixins are Ruby's alternatives to
  - Multiple inheritance
  - Interfaces

ThoughtWorks

Mixins

- When you `include` a module into a class, the modules methods are mixed into the class
- Methods defined in the module can interact with instance variables of the class
- Don't be fooled by the `include` keyword
  - Not like a C/C++ include
  - Class references the module (i.e., no copying)
- Example: hr_mixin.rb

Using Java Classes in Ruby

- Must contain `require "java"`
- 5 options
  - Provide full class name when using

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

  - Assign full class name to a constant.

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

  - Use include_class

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


Using Java Classes in Ruby

- 5 options (continued)
  - Use `include_class` with an alias
  - Useful when Java class name matches Ruby class name

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

  - Use `include_package` to scope Java classes in a Ruby module namespace (only in a module)

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

Calling Semantics

- Parenthesis aren't required when calling methods
- Java get/set/is methods are invoked like Ruby accessors

```
emp.getName()         => e.name
emp.setName("Homer") => e.name = "Homer"
emp.isManager()       => e.manager?
```

- Camelcase Java names can be called with underscores

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


Building Swing in Ruby

- Much friendlier syntax
  - Dynamic typing cuts way down on useless code
- Uses blocks intelligently
  - Define button click behavior in a block at definition
- Example
  - hello_frame.rb

ThoughtWorks

Ruby Methods Added to Core Java

- JRuby adds lots of methods to core JDK classes
- Decorated all the collections classes to make them "humane"
- Added operators to String and type wrappers
  - <=>, <, <=, =>, >, between?
- Example
  - JRuby Super Console

ThoughtWorks

Proxy Classes

- JRuby builds proxies for Java classes
- Allows open classes with Java
  - Add methods to the class
  - Add methods to the *object instance*
- Example
  - array_list_proxy.rb

JRuby Inheriting from Java Classes

- JRuby classes can inherit from Java classes
- Example
  - Car.java
  - race_car.rb

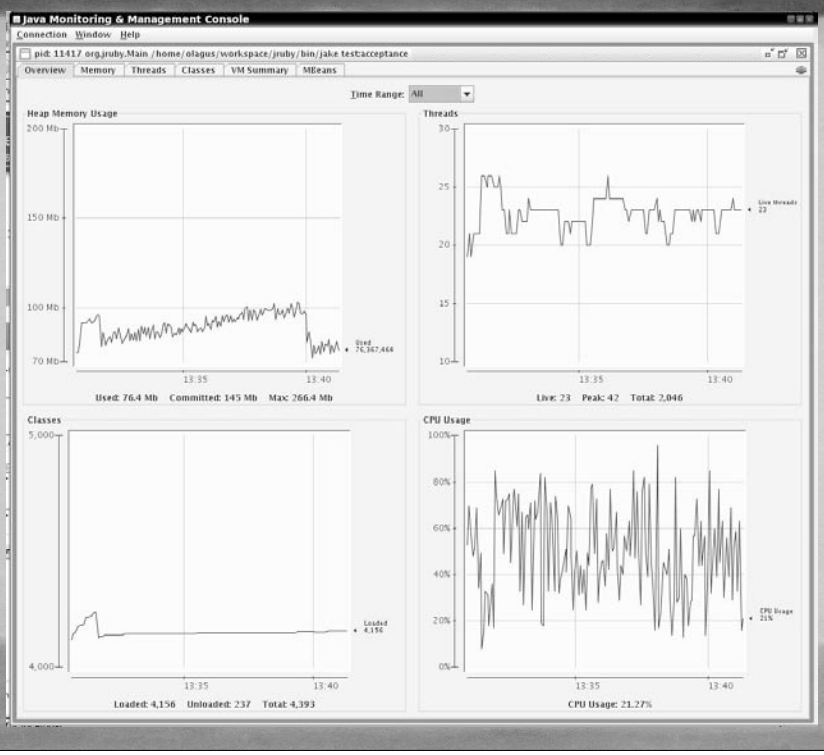
JRuby on Rails

- Yes, Rails now runs on JRuby
- You can build a WAR file from a Rails application and deploy it on a JVM
- Still much slower than Ruby
- Still some manual tweaking for stuff like databases
- But it works!

JRuby Advantages over Ruby

- Ola Bini's recipe:
  - Take one large Rails application with good Selenium test base
  - Convert database configuration to use JDBC
  - Start a selenium proxy with "jake test:selenium_proxy"
  - Start acceptance testing from another window with "jake test:acceptance"
  - In yet another window, write "jconsole"
  - Choose your application

The Result?



Future Directions

- Performance, performance, performance!
- Improving the Rails experience
- 1.0 by JavaOne

Resources

- The JRuby web site
  - http://jruby.codehaus.org/
  - Includes documentation, tutorials, etc.
- Charles Nutter's Blog
  - http://headius.blogspot.com/
- Ola Bini's Blog
  - http://ola-bini.blogspot.com/

ThoughtWorks

Questions?

Please fill out the session evaluations
Samples & slides at www.nealford.com

Neal Ford
www.nealford.com
nford@thoughtworks.com
memeagora.blogspot.com

```
class Employee
  def initialize(name, salary, hire_year)
    @name = name
    @salary = salary
    @hire_year = hire_year
  end

  attr_reader :salary, :hire_year

  def to_s
    "Name is #{@name}, salary is #{@salary}, " +
    "hire year is #{@hire_year}"
  end

  def raise_salary_by(perc)
    @salary += (@salary * (perc * 0.01))
  end
end

class Manager < Employee
  def initialize(name, salary, hire_year, asst)
    super(name, salary, hire_year)
    @asst = asst
  end

  def to_s
    super + ",\tAssistant info: #{@asst}"
  end

  def raise_salary_by(perc)
    perc += 2005 - @hire_year
    super(perc)
  end
end
```

```
require 'hr'

def show(emps)
  emps.each { |e| puts e }
end

employees = Array.new
employees[0] = Employee.new("Homer", 200.0, 1995)
employees[1] = Employee.new("Lenny", 150.0, 2000)
employees[2] = Employee.new("Carl", 250.0, 1999)
employees[3] = Manager.new("Monty", 3000.0, 1950, employees[2])

show(employees)

employees.each { |e| e.raise_salary_by(10) }
puts "\nGive everyone a raise\n\n"
show employees
```

```
require 'test/unit/testcase'
require 'test/unit/autorunner'
require 'hr'

class TestEmployee < Test::Unit::TestCase
  @@Test_Salary = 2500

  def setup
    @emp = Employee.new("Homer", @@Test_Salary, 2003)
  end

  def test_raise_salary
    @emp.raise_salary_by(10)
    expected = (@@Test_Salary * 0.10) + @@Test_Salary
    assert expected == @emp.salary
  end

end
```

```
require 'test/unit/testcase'
require 'test/unit/autorunner'
require 'hr'

class TestManager < Test::Unit::TestCase
  @@Test_Salary = 250000

  def setup
    @manager = Manager.new("Mr. Burns",
      @@Test_Salary, 2003, Employee.new("Smithers", 35000, 1962))
  end

  def test_raise_salary
    @manager.raise_salary_by(10)
    perc = ((2005 - @manager.hire_year) + 10) * 0.01
    expected = (@@Test_Salary * perc) + @@Test_Salary
    assert expected == @manager.salary
  end

end
```

```
require 'test/unit/testsuite'
require 'test/unit/ui/tk/testrunner'
require 'test/unit/ui/console/testrunner'
require 'test_employee'
require 'test_manager'


class TestSuite_AllTests
    def self.suite
        suite = Test::Unit::TestSuite.new("HR Tests")
        suite << TestEmployee.suite
        suite << TestManager.suite
        return suite
    end
end

# Test::Unit::UI::Tk::TestRunner.run(TestSuite_AllTests)
Test::Unit::UI::Console::TestRunner.run(TestSuite_AllTests)
```

```
class Employee
  attr_reader :name, :salary, :hire_year

  def initialize(name, salary, hire_year)
    @name = name
    @salary = salary
    @hire_year = hire_year
  end

  def to_s
    "Name is #{@name}, salary is #{@salary}, " +
    "hire year is #{@hire_year}"
  end

  def raise_salary_by(perc)
    @salary += (@salary * 0.10)
  end

end


class EmployeeList
  def initialize
    @employees = Array.new
  end

  def add(an_employee)
    @employees.push(an_employee)
    self
  end

  def delete_first
    @employees.shift
  end

  def delete_last
    @employees.pop
  end

  def show
    @employees.each { |e|
      puts e
    }
  end
```

```
  def [](key)
    return @employees[key] if key.kind_of?(Integer)
    return @employees.find do |anEmp|
      key == anEmp.name
    end
    return nil
  end
end

list = EmployeeList.new
list.add(Employee.new("Homer", 200.0, 1995)).
  add(Employee.new("Lenny", 150.0, 2000)).
  add(Employee.new("Carl", 250.0, 1999))

list.show
puts "Employee #1 is " + list[0].to_s
puts "Employee named 'Homer' is " + list["Homer"].to_s
```

```
#!/usr/bin/env ruby

class Employee
    def initialize(name, salary)
      @name = name
      @salary = salary
    end

    attr_accessor :name, :salary

    def to_s
      "#{@name} makes #{@salary}"
    end
end
```

```
#!/usr/bin/env ruby
#
#  Created by Neal Ford on 2007-04-20.
#  Copyright (c) 2007. All rights reserved.

require 'hr_closures'

def high_paid(emps)
  threshold = 40000
  return emps.select {|e| e.salary > threshold}
end

def paid_more(amount)
  return Proc.new {|e| e.salary > amount}
end

employees = Array.new
employees << Employee.new("Homer", 15000)
employees << Employee.new("Monty", 100000)
employees << Employee.new("Smithers", 80000)
employees << Employee.new("Carl", 50000)
employees << Employee.new("Lenny", 55000)

puts "Closures that capture local variable scope"
20.times { print '-'}
puts
puts high_paid(employees)

puts
puts "Closures that capture definition scope"
20.times { print '-'}
puts
is_high_paid = paid_more(60000)

puts is_high_paid.call(employees[0])
puts is_high_paid.call(employees[2])
```

```
module Debug
  def who_am_i?
    "#{self.class.name} (\##{self.object_id}): #{self.to_s}"
  end
end

class Employee
  include Debug
  def initialize(name, salary, hire_year)
    @name = name
    @salary = salary
    @hire_year = hire_year
  end

  def to_s
    "Name is #{@name}, salary is #{@salary}, " +
    "hire year is #{@hire_year}"
  end

  def raise_salary_by(perc)
    @salary += (@salary * 0.10)
  end
end

class Manager < Employee
  #include Debug
  def initialize(name, salary, hire_year, asst)
    super(name, salary, hire_year)
    @asst = asst
  end

  def to_s
    super + ",\tAssistant info: #{@asst}"
  end

  def raise_salary_by(perc)
    perc += 2005 - @hire_year
    super(perc)
  end
end



def show(emps)
  emps.each { |e| puts e }
```

```
end

employees = Array.new
employees[0] = Employee.new("Homer", 200.0, 1995)
employees[1] = Employee.new("Lenny", 150.0, 2000)
employees[2] = Employee.new("Carl", 250.0, 1999)
employees[3] = Manager.new("Monty", 3000.0, 1950, employees[2])

show(employees)

puts "\n\nWho are they?"
puts employees[0].who_am_i?
puts employees[3].who_am_i?
```

```
#!/usr/bin/env ruby

require "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

# BlockActionListener is ActionListener whose constructor takes a Ruby block.
# It holds the block and invokes it when actionPerformed is called.
class BlockActionListener < java.awt.event.ActionListener
  # super call is needed for now
  def initialize(&block)
    super
    @block = block
  end

  def actionPerformed(e)
    @block.call(e)
  end
end

# Extend Swing JButton class with a new constructor that takes the button text
# and a Ruby block to be invoked when the button is pressed.
class JButton
  def initialize(name, &block)
    super(name)
    addActionListener(BlockActionListener.new(&block))
  end
end

# Define a class that represents the JFrame implementation of the GUI.
class HelloFrame < JFrame
  def initialize
    super("Hello Swing!")
    populate
    pack
    resizable = false
    defaultCloseOperation = JFrame::EXIT_ON_CLOSE
  end
```

```
  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
    # Note how a block is passed to the JButton constructor.
    greet_button = JButton.new("Greet") do
      name = name_field.text
      # Demonstrate display of HTML in a dialog box.
      msg = %(<html>Hello <span style="color:red">#{name}</span>!</html>)
      JOptionPane.showMessageDialog self, msg
    end
    button_panel.add greet_button
    # Note how a block is passed to the JButton constructor.
    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
```

```
#!/usr/bin/env ruby

require "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
    size == 0 ? nil : get(0)
  end
end
puts "first item is #{list.first}"

# Add "last" method only to the list object ... a singleton method.
def list.last
  size == 0 ? nil : get(size - 1)
end
puts "last item is #{list.last}"
```

```
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;
      }
}
```

```
#!/usr/bin/env ruby
require "java"

include_class "Car"

c = Car.new("Honda", "Accord", 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
```