

Scripting Java with JRuby

Charles Oliver Nutter
Thomas Enebo
Sun Microsystems, Inc
www.jruby.org



Except where otherwise noted, the content of this presentation is licensed under the Creative Commons Attribution-Share Alike 3.0 United States License (<http://creativecommons.org/licenses/by-sa/3.0/us/>).

Which language can do all these:

- > Call Java libraries from scripted code
- > Decorate Java classes with new methods
- > Implement Java interfaces
- > Extend Java classes
- > Support annotations and reflection
- > Deploy on Java app servers
- > Run in applets
- > Run on mobile devices
- > Run on Google AppEngine for Java

JRuby!

JRuby: Ruby for the JVM

> Ruby is:

- Dynamically-typed
- Object-oriented
- Rich syntax
- Vibrant community
- www.ruby-lang.org
- Lots of fun!

> JRuby is:

- High performance
- Real native threads
- Access to Java libs
- Run on Java servers
- Embed as a JAR
- www.jruby.org
- Lots of fun!

Getting Started

- > Download JRuby
 - <http://tinyurl.com/getjruby>
 - `jruby-bin-1.3.0.{tar.gz,zip}`
- > Unpack JRuby
- > Run 'jruby' or 'jruby.bat' from bin dir
 - Put in your PATH env var if you like

Getting Started

```
~ → tar xzf jruby-bin-1.3.0RC2.tar.gz
~ → cd jruby-1.3.0RC2/
~/jruby-1.3.0RC2 → bin/jruby -v
jruby 1.3.0RC2 (ruby 1.8.6p287) (2009-05-27 46598e4) (Java HotS
pot(TM) 64-Bit Server VM 1.6.0_07) [x86_64-java]

~/jruby-1.3.0RC2 → bin/jruby -e "puts 'Hello, world'"
Hello, world

~/jruby-1.3.0RC2 →
```

Ruby Tour: Object Oriented

- > Everything is an Object
 - `Circle.new(4)` => instance of Circle
 - `"abc".length` => 3
 - `1.to_s` => "1"
- > All Objects are instances of Classes
 - `1.class` => Fixnum
- > Single Inheritance
- > Object is base class of all classes

Ruby Tour: Literal Values

- Numbers: `1, 1.0, 12345678987654321`
- String: `"one" 'one' %Q[one] %q[one] ...`
- Multiline string ("here doc"):
`x = <<EOS`
 `extend across two`
 `lines`
`EOS`
- Symbol: `:one, %s[one]`
- Regexp: `/^foo\w+.*bar$/, %r[^foo$]`
- Array: `[1, "ein", :ichi]`
- Hash: `{:one => 1, :two => 2}`

Ruby Tour: Duck Typing

- > Dynamic typing
- > “If it waddles like a duck and it quacks like a duck...”

```
def make_it_waddle(animal)
  animal.waddle
end
```

```
make_it_waddle(Duck.new)
make_it_waddle(Penguin.new)
make_it_waddle(Octopus.new)
```

Ruby Tour: Class Definition

```
class Car < Object # '< Object' optional
  # initialize is Ruby's constructor
  def initialize(color)
    @color = color
  end

  def color
    @color
  end
end

# construct a Hello object
my_car = Car.new("Blue")
puts my_car.color      => "Blue"
```

Ruby Tour: Attributes

```
class Car
  attr_reader :color

  # initialize is Ruby's constructor
  def initialize(color)
    @color = color
  end
end

# construct a Hello object
my_car = Car.new("Blue")
puts my_car.color      => "Blue"
```

Ruby Tour: Blocks (Closures)

```
# two formats: braces {} and do..end
File.open('my_file') do |open_file|
  line = open_file.gets
  ...
end
```

```
# methods that accept blocks
def File.open(filename)
  open_file = open(filename)
  begin
    yield open_file
  ensure
    open_file.close
  end
end
```

Ruby Tour: Modules

```
# A collection of objects
class MyProducts
  # Enumerable provides iteration methods
  include Enumerable

  # define an 'each' method that iterates
  def each
    # yield to each element in turn
  end
end

list = MyProducts.new
list.select {|item| item.price > 5.00}
list.sort {|a, b| a.name <=> b.name}
list.max
```

Getting Started Scripting Java

- > Require the 'java' library
 - In code: require 'java'
 - Using a flag: -r java
- > Import classes, interfaces, etc
- > Script them!

Bonuses!

- > Jar files are libraries you can require
 - require 'itext.jar'
- > Add to \$CLASSPATH at runtime
 - \$CLASSPATH << 'itext.jar'
- > Java methods in camel or ruby case
 - currentTimeMillis or current_time_millis
- > Bean-like properties are attributes
 - frame.always_on_top = true

Call Java Libraries

```
import java.util.ArrayList
```

```
list = ArrayList.new
```

```
5.times {|i| list.add i}
```

A Ruby block or closure



```
iter = list.iterator
```

```
puts iter.next while iter.has_next?
```

Call Java Libraries

```
import java.util.ArrayList
```

```
list = ArrayList.new
```

```
5.times {|i| list.add i}
```

```
iter = list.iterator
```

```
puts iter.next while iter.has_next?
```

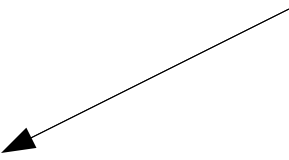
```
~ → jruby -rjava test.rb  
0  
1  
2  
3  
4
```

Decorate Java classes

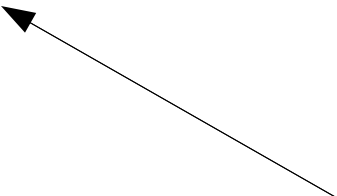
```
import java.lang.System  
import java.util.ArrayList
```

```
list = ArrayList.new  
5.times {|i| list << I}  
list.each {|element| puts element }
```

Using Ruby's <<
operator to append to a
Java collection!



Using Ruby's “each” method
on a Java collection!



Decorate Java classes

```
import java.lang.System
import java.util.ArrayList
```

```
list = ArrayList.new
5.times {|i| list << i}
list.each {|element| puts element }
```

```
~ → jruby -rjava test.rb
0
1
2
3
4
```

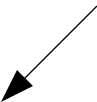
Using Ruby's "each" method
on a Java collection!

Implement Interfaces

```
JThread = java.lang.Thread
```

```
JThread.new { 5.times { puts 'hello' } }.start
```

Using a Ruby closure/block to implement a Java interface!



```
import java.lang.Runnable
```

```
class MyTask
```

```
  include Runnable
```

Include a Java interface to implement it
(good for larger interfaces)



```
  def run; 5.times { puts 'goodbye' }; end
```

```
end
```

```
JThread.new(MyTask.new).start
```

Implement Interfaces

```
JThread = java.lang.Thread
```

```
JThread.new { 5.times {
```

```
import java.lang.Runnable
```

```
class MyTask
```

```
include Runnable
```

```
def run; 5.times { p
```

```
end
```

```
JThread.new(MyTask.new).start
```

```
~ → jruby -rjava test.rb
hello
hello
hello
hello
hello
goodbye
goodbye
goodbye
goodbye
goodbye
```

closure/block to
implement interface!

implement it

Extend Classes

```
import java.util.HashMap
class LoggingMap < HashMap
  def put(a, b)
    puts "adding key: #{a} value: #{b}"
    super
  end
end
my_map = LoggingMap.new
my_map.put(1, 'hello'); puts my_map
```

Extend Classes

```
import java.util.HashMap
class LoggingMap < HashMap
  def put(a, b)
    puts "adding key: #{a} value: #{b}"
    super
  end
end
end

my_map = LoggingMap.new
my_map.put(1, 'hello'); puts my_map
```

```
~ → jruby -rjava test.rb
adding key: 1 value: hello
{1=hello}
```

Annotations and Reflection

- > Ruby2Java gem
 - Input: Ruby source with signatures, annotations
 - Output: Java class
- > Normal Java class
 - Import into Java code
 - Construct with “new”
 - Call or reflect methods
- > Coming soon; prototype works now

Deploy to Java Servers

```
~/my_rails_app $ warble
```

```
...copying files
```

```
mkdir -p tmp/war/WEB-INF
```

```
jar cf my_rails_app.war -C tmp/war .
```

```
~/my_rails_app $ asadmin deploy my_rails_app.war
```

Deploy to Java Servers

- > Use Warbler to package as WAR file
 - JRuby + framework + deps, all in one
 - Deploy on any web container
- > Simple to use
 - `gem install "warbler"`
 - Run `"warble"` inside application dir
 - Deploy resulting WAR file

Applets and Mobile

- > Compiled or interpreted: fewer security hassles
- > Include all or none of Ruby libs for size
- > Check JRuby.org soon for an applet demo
- > JRuby community working on mobile apps

Scripting Java with JRuby

DEMO DEMO DEMO

Learn More!

- > Visit www.jruby.org
- > Email us
 - Charlie: charles.nutter@sun.com
 - Tom: thomas.enebo@sun.com
- > Find us on IRC
 - #jruby on freenode.net
- > Twitter
 - @headius and @tom_enebo