Ruby auf der JVM

Eine kurze Einführung zu JRuby

Sebastian Schmeck @ JUG Görlitz, 29.05.2012

Agenda

Einleitung
Ruby trifft Java
Von Ruby zu Java
Von Java zu Ruby
Zusammenfassung

Einleitung
Zu meiner Person

- Sebastian Schmeck
- Saxonia Systems
- Java (> 6 Jahre)
- Ruby (~ 1 Jahr)
- Testgetriebene Entwicklung
- Continuous Integration / Delivery

Einleitung
Ruby

"Ruby ist eine dynamische, freie Programmiersprache, die sich einfach anwenden und produktiv einsetzen lässt. Sie hat eine elegante Syntax, die man leicht lesen und schreiben kann."

"Ruby wirkt simpel, aber ist innen sehr komplex, genau wie der menschliche Körper."

http://www.ruby-lang.org/de/about/

Einleitung
Ruby (2)

```ruby
# Eine Skriptsprache ..
puts "Hello World!"

# mit Klassen ..
class MyHello

  def initialize(yourName)
    @name = yourName
  end

  def say_hello
    puts "Hello #{@name}!"
  end

end

# und Objekten ..
myObject = MyHello.new("JUG")

# und Closures
5.times { myObject.say_hello }
```

- Seit 1993 von Yukihiro "Mats" Matsumoto entwickelt
- Anspielung auf Perl
- Dynamisch typisiert
- Interpretersprache
- Objektorientierte und funktionale Elemente
- Aktuelle Version 2.0.0
- Ruby On Rails

Einleitung
JRuby

- Seit 2001 entwickelt
- Ruby-Interpreter für die JVM
- Echtes Multithreading im Gegensatz zu Standard-Interpreter (MRI/YARV)
- Unterstützt von JSR292 / invokedynamic
- Aktuelle Version 1.7.4
- Herunterladen, installieren und loslegen
  http://jruby.org/download

Einleitung
JRuby (2)

$ jruby --version

```
jruby 1.7.3 (1.9.3p385) 2013-02-21 dac429b on Java HotSpot
(TM) 64-Bit Server VM 1.6.0_30-b12
```

$ jruby -e "puts 'Hallo Welt!'"

```
Hallo Welt!
```

$ jirb # REPL

```
irb(main):001:0> puts ['Hello','World!'].join ' '
Hello World!
=> nil
irb(main):002:0>
```

Java trifft Ruby

Java

- Compiler
- Statisch typisiert
- Unternehmens-anwendungen

Ruby

- Interpreter
- Dynamisch typisiert
- Webanwendungen / Scripting

Java trifft Ruby
Alles ist ein Objekt

```java
// Objekt
Date date =
  new Date()
```

```ruby
# Objekt
date = Date.new
```

```java
// primitiver Typ
String.valueOf(42)
```

```ruby
# Objekt
42.to_s
```

```java
// null
null == null
```

```ruby
# Objekt
nil.nil?
```

Java trifft Ruby
Konventionen

```
// Methoden
startsWith(String prefix)

// Getter
getName()

// Setter
setName(String name)

// Klassenkonstanten
File.separator

// Objekterzeugung
new String("Java")
```

```
# Methoden
start_with?([prefix]+)

# Getter
name

# Setter
name=(name)

# Klassenkonstanten
File::SEPARATOR

# Objekterzeugung
String.new("Ruby")
```

Java trifft Ruby
Blöcke

```
// Closures (Java 8)
List numbers =
  new ArrayList<Integer>
  (Arrays.asList(1,2,3))

Collections.sort(
  list,
  (o1, o2) -> o2 - o1);
```

```
# Closures
[1,2,3].sort {|o1,o2| o2-
o1}

# => [3, 2, 1]
```

```
# map/reduce ;-)
(1..7)
  .map {|i| i%2 }
  .reduce(:+)

# => 4
```

Java trifft Ruby
Offene Klassen

```ruby
# Jeder Klasse kann jederzeit
# Funktionalität hinzugefügt werden
class String
   def palindrome?
      string = self.gsub(/\W/, '').downcase
      string == string.reverse
   end
end

"Dreh mal am Herd!".palindrome? # => true
```

Java trifft Ruby
Zucker

```ruby
# Gezuckert
puts "Hello Java!" # => Hello Java!
5+3 # => 8

# Entzuckert
puts("Hello Java!")
5.+(3)

# Explizites Senden
Kernel.send(:puts, "Hello Java!")
5.send(:+, 3)
```

Von Ruby zu Java

```
$ jirb
irb(main)> puts java.lang.System.get_property 'java.home'
/usr/lib/jvm/java-6-openjdk-amd64/jre
=> nil
```

- Mit JRuby kann man auf Java-Bibliotheken zugreifen
- Der Java-Code wird 'rubifiziert'
- Es ist möglich Java-Klassen mit Ruby-Mitteln zu erweitern
- Zugriff auf alle JVM-Sprachen

Von Ruby zu Java
Classpath

```
# Kommandozeile
$ jruby -J-cp /path/to/library.jar ..

# Im Quellcode
$CLASSPATH << '/path/to/library.jar'

# Im Quellcode im Rubystil
require '/path/to/library.jar'
```

Von Ruby zu Java
Klassen laden

```
# Mit Prefix Java::
irb> Java::java.lang.StringBuffer
=> Java::JavaLang::StringBuffer

# Mit der Funktion java_import
irb> java_import 'java.lang.StringBuffer'
=> [Java::JavaLang::StringBuffer]
```

Von Ruby zu Java
Klassen verwenden

```
# Unterstützung der Ruby-Konventionen

java_import 'java.lang.System'
System.current_time_millis
=> 1361998812513

Java::java.lang.String::CASE_INSENSITIVE_ORDER
=> #<Java::JavaLang::CaseInsensitiveComparator:0x[..]>

date = java.util.Date.new
=> #<Java::JavaUtil::Date:0x2cfb1135>

date.time = 42  # => 42
date.time       # => 42
```

Von Ruby zu Java
Parameterübergabe

```
# Automatische Konvertierung bei Java-Aufruf
java.lang.Integer.parse_int("42").class # => Fixnum

# Kann mit to_java() explizit ausgeführt werden
42.class           # => Fixnum
42.to_java.class   # => Java::JavaLang::Long

# Java Collections sind um Ruby-Funktionen erweitert
java.util.Arrays.as_list(1,2,3).map {|e| e*2 }
=> [2, 4, 6]
```

Von Ruby zu Java
Überladung

```
# Automatische Auswahl der 'richtigen' Implementierung
java.lang.String.value_of 70
# => "70"

# Mit java_send() Implementierung erzwingen
java.lang.String.java_send :valueOf, [Java::char], 70
# => "F"

# Mit java_alias() kann man einen Alternative erstellen
class java.lang.String
  java_alias :value_of_char, :valueOf, [Java::char]
end
java.lang.String.value_of_char 70
# => "F"
```

Von Ruby zu Java
Interfaces implementieren

```
java_import java.lang.Runnable
java_import java.util.concurrent.Executors

# Implementierung aller Methoden
class Foo
  include Runnable
  def run; puts 'foo'; end;
end

Executors.callable(Foo.new).call

# Verwendung eines Blocks (bei Single-Method-Interfaces)
Executors.callable { puts 'foo'} .call
```

Von Java zu Ruby

```
// Eingebetteter Ruby-Container
import org.jruby.embed.ScriptingContainer;

public class JRubyDemo {
 public static void main(String[] args) {
   new ScriptingContainer().runScriptlet("puts 'Moin'");
 }
}
// JRuby muss im Classpath sein
$ java -cp jruby-core.jar JRubyDemo
Moin
```

- Scripting für Java-Anwendungen
- Einsatz von Ruby-Bibliotheken

Von Java zu Ruby
ScriptingContainer

```
// Liefert das Ergebnis des letzten Aufrufs
ScriptingContainer c = new ScriptingContainer();
assertEquals(42L, c.runScriptlet("$answer=42"));

// Merkt sich den Kontext
assertEquals(42L, c.runScriptlet("$answer"));

// Methodenaufruf inkl. Konvertierung
Object rubyMap = c.runScriptlet("{1 => 2, 2 => 4}");
Map<Long, Long> javaMap = new HashMap<Long, Long>();
javaMap.put(1L, 3L);
Map<Long,Long> resultMap = (Map<Long,Long>) c.callMethod
(rubyMap, "merge", javaMap);
assertEquals(Long.valueOf(3L), resultMap.get(1L));
```

Von Java zu Ruby
JSR223 - Scripting for the Java Platform

```
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
// ..
ScriptEngine rubyEngine =
  new ScriptEngineManager().getEngineByName("jruby");
assertEquals(2L, rubyEngine.eval("1+1"));
```

- Alternative zum ScriptingContainer
- Standard zur Unterstützung von Scriptsprachen
- Seit Java 6

Von Java zu Ruby
Just-In-Time-Compiler

- Optimierung von Code während der Ausführung
- Optimiert Code, der oft ausgeführt wird ('hot')
- Standardansatz in JRuby
- Der optimierte Code wird bei mehrfacher Ausführung schneller
- Empfehlungen zur Untersützung des JIT-Compilers
  - Vermeidung von Codegenerierung zur Laufzeit
  - Kleine Methoden schreiben

Von Java zu Ruby
Ahead-Of-Time-Compiler

- Standardansatz für Java
- Kann für Ruby-Code explizit ausgeführt werden
- Erstellt Bytecode

```
# example.rb
class Example
  def hello(name)
    "Hello #{name}!"
  end
end

# Erstellt eine .class-Datei (Nur für JRuby)
$ jrubyc example.rb
```

Von Java zu Ruby
JRuby-AOT-Compiler

```
# Erstellung eines Java-Wrappers (Example.java) mit --
javac
$ jrubyc --javac example.rb
public class Example extends RubyObject  {
    public Object hello(Object name) {
        // ..
    }
}

# Typdefinition mittels java_signature()
java_signature 'String hello(String name)'
def hello(name)
    "Hello #{name}!"
end
```

Von Java zu Ruby
JRuby-AOT-Compiler (2)

```ruby
# Import von Java-Klassen mit java_import()
java_import 'java.util.Collections'

# Definieren eines Package mit java_package()
java_package 'de.saxsys.jruby'

# Implementieren eines Interface mit java_implements()
class Example
  java_implements 'Serializable'
  # ..
end
```

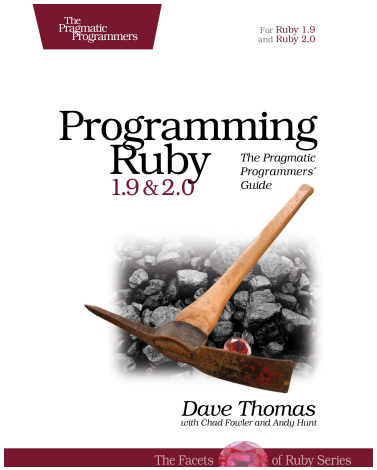
Zusammenfassung

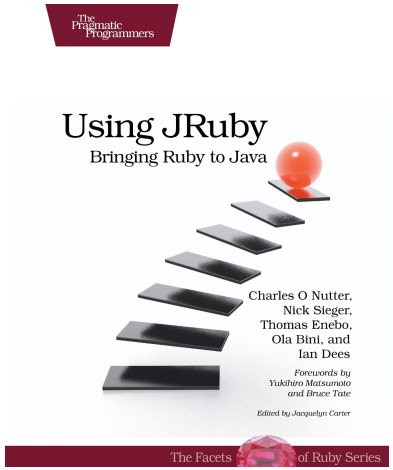
- JRuby ist eine Brücke, um in Java-Projekten eine produktive Skriptsprache einzusetzen (z.B. zum Testen, Monitoring, Reporting, ...)

- JRuby ist eine Brücke, um Ruby-Anwenundungen auf eine reife, performante VM aufzusetzen

Referenzen

- http://jruby.org/
- Programming Ruby 1.9 & 2.0: The Pragmatic Programmers' Guide (Pickaxe)
  2013, Pragmatic Programmers
- Using JRuby: Bringing Ruby to Java
  2011, Pragmatic Programmers





Danke

Fragen?