Ruby QuickStart

Michael Neumann (mneumann@ntecs.de)

Ntecs – Neumann Technologies

Was ist Ruby?

- Ruby = Smalltalk – Unfamiliar syntax
  + Perl's scripting power
  + Python's exception etc.
  + CLU's iterator
  + a lot more good things
- Ruby > (Smalltalk + Perl) / 2

Features

- Interpretiert (AST), strong dynamic typing
- Alles ist ein Objekt
- GC (mark & sweep, generational)
- Exceptions
- Lightweight Threads, Continuations (auch unter DOS :-)
- Closures, Code-Blocks
- Eingebaute Regexps, Arbitrary precise Integers

1. Beispiel

```
1.upto(10) { |i| print i }
```

```
class Integer
  def upto(to)
    for j in self..to
      yield j
    end
  end
end
```

Offene Klassen

Java:

- Klasse nur an einer Stelle definiert, dann nicht mehr veränderbar/erweiterbar
- Vorhandene Programme kann man nicht ändern, (kann auch positiv sein; Bsp: Addieren von Integern)
- Klassen nur durch Vererbung erweiterbar

2. Beispiel

```
a = ['welcome', "at", "GPN", 2]
a.each { |ele| print ele }
```

```
class Array
  def each
    for i in 0..(self.size-1)
      yield self[i]
    end
  end
end
```

Keine Klammern, kein **self**

[] ist Methodenaufruf

3. Beispiel (“Finish what you Start”)

```
f = File.open('/tmp/hw', 'w+')
f.write 'Hello World\n'
f.close

File.open('/tmp/hw', 'w+') {|f|
  f.write 'Hello World\n'
}
```

```
f = File.open('/tmp/hw', 'w+')
begin
  f.write 'Hello World\n'
ensure
  f.close
end
```

Ensure == Finally in Java

Blocks: Sicherstellen Resourcen-Freigabe

dadurch: weniger Code, sicherer (wer prüft denn Exceptions etc.)

Beispiel: Networking

```
require 'socket'

IPSocket.getaddress('www.entropia.de')
   # => "212.227.3.124"

TCPSocket.open('localhost', 'daytime').read
   # => Sat May 24 18:51:12 CEST 2003
```

```
require 'net/http'
Net::HTTP.start( 'www.ntecs.de', 80 ) {|http|
   response, body = http.get('/blog/index.rss')
   print body
}
   # => <rss version='0.91'>...</rss>
```

Weitere Beispiele

- Word-Frequency-Count

```
freq = Hash.new(0)
open("wordfile").read.scan(/\w+/){|w| freq[w] += 1}
freq.keys.sort.each {|w| puts "#{ w }: #{ freq[w] }" }
```

- Fibonacci

```
def fib(n)
  if (0..1).include? n
    1
  else
    fib(n-2) + fib(n-1)
  end
end
```

```
fib(10_000)
```

Wirklich?

ca. 10 ^ 2086 mal schneller:

```
require "memoize"
memoize :fib
fib(10_000)
```

O(n) vs. O(Fib(n))

Elemination von Rekursion durch Result-Caching.

Was haben wir gelernt?

- Offene Klassen
- Code-Blöcke
  - Iteratoren
  - Resourcen-Freigabe
  - Callbacks

Offene Klassen: Klassen an beliebiger Stelle im Code zu jeder Zeit erweiterbar.

Wichtige Datentypen

Float, Integer (Fixnum, Bignum)

```
3.1415  -45  10_000  10**1000
```

String

```
"a string"  'other'  %{again a string}
```

Array

```
[]  [1,'test', nil]  %w(ruby python perl)

Array.new(23)
```

Fixnums: 31-bit signed, BOXED

Automatic conversion from Fixnum to Bignum!

WICHIG: alles nur Syntax um Ob jekte zu erzeugen.

Wichtige Datentypen (2)

Hash (Python: Dictionary)

```
{}   {'Mike' => 23, 'John' => 34}
Hash.new
```

Symbol

```
:black  :red  :On  :Off
```

Range

```
1..10   1...10   'a'..'b'   objA..objB
Range.new(1,10)
```

Symbols: wie in Lisp, Smalltalk (#xxx)

WICHIG: alles nur Syntax um Ob jekte zu erzeugen.

Keine spezielle eingebaute Semantik!

Wichtig(st)e Datentypen (3)

Regexp

```
/^[a-zA-Z_]\w*$/

Regexp.new("^[a-zA-Z_]\w*$")
```

Statt (Python):

```
import re
r = re.compile("^[a-zA-Z_]\w*$")
r.match("AnIdentifier")
```

Einfach (Ruby):

```
"AnIdentifier" =~ /^[a-zA-Z_]\w*$/
```

Regexp's sind vollständig objekt-orientiert, d.h. man kann auch Python-style benutzen!

Regexp sind in vielen (Ruby-) Programmen essentiell, daher macht spezielle Syntax Sinn.

Wichtige Datentypen (4)

Proc (Closure)

```
proc { ... }                lambda { ...}
Proc.new { ... }
```

```
def adder(n)
  return proc {|a| a+n }
end

add1 = adder(1)
add1.call(5)       # => 6
add1[6]            # => 7
```

Bzw. dasselbe mit do ... end.

In Python: lambda

Wird oft in funktionalen Sprachen verwendet.

Spezielle Werte

nil, true, false

Objekte der Singleton-Klassen *NilClass*, *TrueClass* und *FalseClass*.

Kontrollstrukturen

```
if condition
  ...
elsif condition
  ...
else
  ...
end
```

```
if cond then expr1 else expr2 end
```

```
cond ? expr1 : expr2
```

```
expr if cond
```

z.B.: `exit if i > a.size`

Genauso: `unless` == `if not`

Modifier

Kontrollstrukturen (2)

```
case anObj
when v1
  ...
when v2, v3
  ...
else
  ...
end
```

<=>

```
if anObj == v1
  ...
elsif anObj == v2 or
      anObj == v3
  ...
else
  ...
end
```

“switch” in C

Case wie wie switch in C, jedoch ohne break.

Kontrollstrukturen (3)

```
{ ... }
```
<=>
```
do ... end
```

```
for var in enum
  ...
end
```
<=>
```
enum.each do |var|
  ...
end
```

```
while cond
  ...
end
```

Ruby	C/C++
`break`	*`break`*
`next`	*`continue`*

Exceptions

```
begin
  ...
  raise RuntimeError
  ...
rescue RuntimeError
  ...
end
```

Exceptions sind Objekte!

`raise` ist eine Methode!

Exceptions sind Objekte

Beliebig viele rescue-Klauseln (auch ohne angabe von Klasse)

ensure

redo, raise

Variablen, Konstanten

Lokale Variablen (beginnen klein)

```
i   j1   name   aXb   _abc      ~~Klasse~~   ~~1abc~~
```

Globale Variablen (beginnen mit $)

```
$a   $DEBUG   $Debug   $$   $@   $1
```

Instanz Variablen (beginnen mit @)

```
@a   @name   @Value           ~~@1~~
```

Konstanten (beginnen gross)

```
String Array ABC            ~~aBC~~
```

Konventionen (OO)

- Klassen = Substantive Gross!

```
Animal Car Dog Cat File
```

- Methoden = Verben Klein!

```
bark miau read new open
```

Konventionen (2)

- `aMethod?` Rückgabewert ist Boolean, Frage

```
anArray.empty?
File.directory? '/tmp'
```

- `aMethod!` Destruktives Update (Objekt wird verändert)

```
a = '   test'
a.strip     # => 'test'
a           # => '   test'
a.strip!    # => 'test'
a           # => 'test'
```

- `attrib=` Setter-Methode (Java: setXXX)

```
a.value = 4  <-->  a.value=(4)
```

OO Ruby

- Alles ist ein Objekt (wie in Smalltalk)
- Prozeduren sind Methoden
- Einfachvererbung + Mixins (= Mehrfachvererbung)
- Meta-Klassen, Klassen selbst sind Objekte
- Singleton Methoden

Prozeduren = Methoden bedeutet, daß jede Methode einen impliziten self Parameter bekommt, und zu einer Klasse gehört (Methode ohne Klasse gibts nicht).

Klassen: Ein Beispiel

```
class Vehicle
  def initialize(wheels)
    @wheels = wheels
  end
end

# erzeuge Fahrzeug mit zwei Räder
aVehicle = Vehicle.new(2)
```

`new` erzeugt Objekt und ruft dann `initialize` auf um das neue Objekt zu initialisieren.

Klassen: Ein Beispiel (2)

Wollen auf die Anzahl der Räder zugreifen.
Lösung: Klasse um Methode erweitern, die Anzahl Räder zurück liefert.

```
class Vehicle
  def initialize(wheels)
    @wheels = wheels
  end
  def wheels
    @wheels
  end
end
aVehicle = Vehicle.new(2)
aVehicle.wheels     # liefert: 2
```

WICHTIG: Es lassen sich nur Methoden aufrufen; keine Variablen direkt zuweisen, wie z.B. in Python.

Wir hätten auch def initialize weglassen können, da wir die Klasse erweitern.

Return ist optional. Letzter Wert ist Rückgabewert.

Klassen: Ein Beispiel (3)

Wollen nun auch Anzahl der Räder ändern.

```
class Vehicle
  def wheels=(newWheels)
    @wheels = newWheels
  end
end

aVehicle = Vehicle.new(2)
aVehicle.wheels     # liefert: 2
aVehicle.wheels = 4
aVehicle.wheels     # liefert: 4
```

Avehicle.wheels= ist eine Methode!

Offene Klasse, wir erweitern die Klasse aus den vorangegangenen Beispielen.

WICHTIG: Status eines Objektes lässt sich von aussen nur durch Methodenaufrufe verändern (wie Smalltalk).

Vererbung

```
class Car < Vehicle
  attr_accessor :ps    # erzeuge getter/setter

  def initialize(ps)
    super(4)           # 4 Räder
    @ps = ps
  end
end

aCar = Car.new(1001)
aCar.ps                # liefert: 1001
aCar.ps = 50
```

attr_accessor erzeugt zwei Methoden ps und ps=

gibt auch attr_reader, attr_writer

Klassen-Methoden

```
class Car
  def Car.car_with_1001_ps
    Car.new(1001)
  end
end

aCar = Car.car_with_1001_ps
aCar.ps                  # liefert: 1001
```

Anstatt `def Car.car_with_1001_ps` hätte man auch einfach `def self.car_with_1001_ps` schreiben können.

Car.new ebenso durch self.new bzw einfach durch new ersetzbar.

Man hätte auch den Klassen-Rumpf weglassen können.

Singleton-Methoden

Definiere Methode nur für ganz bestimmtes Objekt.

```
a = [1,2,3]

def a.count
  self.size
end

a.count         # => 3
[1,2,3].count   # => undefined method `count'
```

Äquivalent zu

```
def a.count
  self.size
end
```

ist

```
class << a
  def count
    self.size
  end
end
```

Mixins, Module, Mehrfachvererbung

```
class A
  def a
  end
end

module B_Impl
  def b
  end
end

class B
  include B_Impl
end

class C < A
  include B_Impl
end
```

Module = Namensräume
(oder “Behälter”)

C erbt von A und B (bzw. B_Impl).

Module sind einfach “Behälter”

Es lassen sich beliebig viele Module in eine Klasse einbinden.

Beispiel: Mixins

```
class Sequence
  include Enumerable   # definiert collect, select, reject...

  def initialize(from=1, to=nil)
    @from, @to = from, to
  end

  def each
    i = @from
    loop {
      yield i
      break unless @to.nil? or i <= @to
      i += 1
    }
  end
end

p Sequence.new(1,10).select{|i| i%2 == 0}.collect{|i| i**2}
# => [4, 16, 36, 64, 100]
# Quadrate aller geraden Zahlen zwischen 1 und 10
```

Mehrfach (Parallel) Zuweisung (Pattern-matching)

Methoden in Enumerable rufen each auf.

Zugriffskontrolle

```
class A

  public   # Öffentliche Methoden folgen

  def a
  end

  private  # Private Methoden folgen

  def b
  end
end

x = A.new
x.a        # OK
x.b        # NameError: private method `b' called
```

Private Methoden lassen sich nur ohne “.” Notation aufrufen!

D.h. innerhalb einer Klasse.

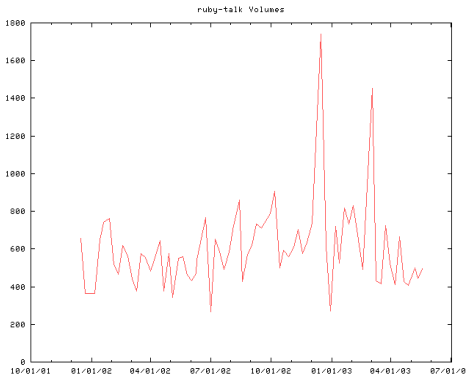
Statistics

Small is beautiful: 277 Objekte “on startup”, 42 Klassen in Basis-Library

```
i=0; ObjectSpace.each_object {i += 1}; p i
```

Array, Bignum, Binding, Class, Comparable, Continuation, Dir, Enumerable, Errno, Exception, FalseClass, File, File::Stat, FileTest, Fixnum, Float, GC, Hash, IO, Integer, Kernel, Marshal, MatchData, Math, Method, Module, NilClass, Numeric, Object, ObjectSpace, Proc, Process, Range, Regexp, String, Struct, Struct::Tms, Symbol, Thread, ThreadGroup, Time, TrueClass

[ruby-talk]: ~70,000 Messages seit Dez. 1998 = 1500/Monat



Application Archive: 897 Projekte

German books:
- 2x Programmieren mit Ruby (Stefan,Armin,Wyss/Dave&Andy)
- Ruby: Das Einsteigerseminar

English:
Programming Ruby, RDG, The Ruby Way
Ruby in a Nutshell (matz), Making Use of Ruby
Teach Yourself Ruby in 21 days

http://www.andsoforth.com/galleries/seattle2002/tn

Number of Perl ports (FreeBSD): 992
Debian unstable:
- Ruby 170
- Python 417

Ruby's Popularity

- Artikel in iX und c't
- Regelmäßig Artikel in Linux-Enterprise
- 3 Deutsche, 6 Englische, 23+ Japanische Bücher
- FreeBSD:

```
find /usr/ports -name 'ruby-*' | wc -l
```

=> 249

```
find /usr/ports -name 'py*' | wc -l
```

=> 225

Resourcen

www.ruby-lang.org - Ruby Homepage

raa.ruby-lang.org - Ruby Application Archive

www.rubycentral.com - Online Book und Library Reference

www.rubygarden.org - Wiki, Diskussions Forum

www.rubytalk.com - ML Archive

www.hypermetrics.com/ruby37.html - 37 Reasons I Love Ruby

www.ruby-lang.org/~slagell - Gutes Tutorial

Werbung...


First European Ruby Conference
EuRuKo 03
APPROXIMITY
we manage risk
When?
21. and 22. June 2003 (10:00 - open end)
Where?
Germany, University of Karlsruhe, Am Fasanengarten 5
Building 50.34, Informatik-Multimedia-Hörsaal, Room -102
Topics?
Talks, Discussions, Programming, Fun
Cost?
Conference fee: 20 €
More info?
EuRuKo-Wiki at http://www.approximity.com/euruko03
LinuX
Enterprise
Q
Software
Solutions
dpunkt.verlag

Fragen ?

GUIs: Gtk


```
require 'gtk2'

Gtk.init

window = Gtk::Window.new
button = Gtk::Button.new("Hello World")
button.signal_connect("clicked") { exit }
button.show
window.add(button)
window.show

Gtk.main
```

GUIs: Tk


```
require 'tk'

root = TkRoot.new

TkButton.new(root) {
  text "Hello World"
  command { exit }
}.pack

Tk.mainloop
```