

Ruby Basics - „Appetizer“

Ein hoffentlich appetitanregender subjektiver Überblick der Programmiersprache Ruby.

9. Januar 2008 - „Rails Hock“
Ruby on Rails Switzerland User Group

Menü



- Warum Ruby?
- Arbeiten mit Arrays und Hashes
- Object Handling, Mixins
- Reflection
- Ruby Standard Library
- Literatur, Fragen, Kontakt

Warum ein Vortrag über Ruby hier?



- Wer mit Rails arbeitet, sollte doch mit Ruby vertraut sein, oder?
- Rails ist so „magic“ gebaut, dass wir uns nicht in der Tiefe mit Ruby auseinandersetzen müssen.
- Viele Ruby Features werden vorweggenommen („Convention over Configuration“ dank Reflection).
- Es ist nicht zwingend notwendig, Ruby in der Tiefe zu beherrschen, aber natürlich hilfreich, Ruby etwas genauer zu kennen
- Background EK: von C++, Perl über Ruby zu Rails!
Meine Intention: Euch Ruby schmackhaft zu machen und Euch den ein oder anderen Kniff weiterzugeben.

Warum also Ruby?



- Mächtige Programmiersprache
- produktiv
- gut lesbar = wartbar
- Fazit EK: Macht Spass

Warum Ruby? - Ruby ist lesbar und intuitiv



- ▶ `3.times { print 'Hallo! ' }` → `Hallo! Hallo! Hallo!`
- ▶ `3.upto(6) { | i | print i, " " }` → `3 4 5 6`
- ▶ `"ab".upto("ad") { | x | print x, "; " }` → `ab; ac; ad`
- ▶ `[1, 2, 3, "a"].each { | x | print x }` → `123a`

Warum Ruby? - Idiome ? !



- ▶ `an_object#method?` - Prädikat, das true/false liefert

- ▶ `[1,3,6].include?(5)` → false
 `"rails".nil?` → false
 `nil.nil?` → true

- ▶ `an_object#method!` - Methode, die `an_object` verändert, im Gegensatz zu
 `an_object#method` - die eine (modifizierte) Kopie von `an_object` liefert

- ▶ `ruby = "ruby"`
 `ruby.upcase` → "RUBY"
 `ruby` → "ruby"

- `ruby.upcase!` → "RUBY"
 `ruby` → "RUBY"

Warum Ruby - Blocks und Iteratoren I



- Iterator = Methode, die einen Code Block aufruft
- Block = Stück Code, das innerhalb eines Iterator aufgerufen wird. Der Aufruf des Blocks wird durch `yield` ausgelöst

```
▶ def two_times  
  yield  
  yield  
end
```

```
two_times { print 'Hello!' } → Hello!Hello
```

Warum Ruby - Blocks und Iteratoren II



- Beispielhafte Implementation `Array#find`;

liefert das erste Element aus einem Array, das die Bedingung gegebenen Block erfüllt

```
▶ class Array
  def find
    for i in 0..size
      value = self[i]
      return value if yield(value)
    end
    return nil
  end
end
```

```
▶ [1, 3, 5, 7, 9].find { |v| v > 5 } → 7
```

Arbeiten mit Arrays



- Zugriff auf die Elemente des Arrays - first, last, []
- Traversieren eines Arrays - each, collect
- Traversieren und modifizieren eines Arrays - collect!
- Kumulieren der Elemente eines Arrays - inject

Arbeiten mit Arrays - Element Zugriff



- ▶ `a = [1,2,3]`
 `a.include?(3)` → `true`

- ▶ `a.first` → `1`
 `a.last` → `3`

- ▶ `a[2]` → `3`
 `a[-1]` → `3` # anstelle: `a[a.length - 1]`

- ▶ `a.pop` → `3`
 `a` → `[1, 2]`

- ▶ `a.push(5, 6, 7)` → `[1, 2, 5, 6, 7]`
 `a << 17` → `[1, 2, 5, 6, 7, 17]`

- `a.select {|x| x > 5}` → `[6, 7, 17]`

Arbeiten mit Arrays - Traversieren #each



- Traversieren über alle Elemente eines Arrays ausführen des Codes im gegebenen Block, ohne ein Element zu verändern.

▶ `a = [1,2,3]`

▶ `a.each {|x| x*2}` → `[1,2,3]` # macht keinen Sinn

▶ `a.each {|x| print x}` → `[1,2,3]`
→ `123` #out

- `Array#each`

Arbeiten mit Arrays - Traversieren #collect



- Traversieren über alle Elemente eines Arrays, ohne ein Element zu verändern
- Liefert einen **neuen** Array, der die Ergebnisse aus dem Block für jedes Element enthält

▶ `a = [1,2,3]`

▶ `a.collect{|x| x*2}` → `[2, 4, 6]`
 `a` → `[1, 2, 3]`

- `Enumerable#collect`

Arbeiten mit Arrays - Traversieren #collect!



- Traversieren über alle Elemente eines Arrays. Jedes Element des Arrays wird durch das Ergebnis des Blocks **ersetzt**.

▶ `a = [1,2,3]`

▶ `a.collect!{|x| x*2}` → `[2, 4, 6]`
 `a` → `[2, 4, 6]`

- `Array#collect!`

Arbeiten mit Arrays - Kumulieren #inject



- Traversieren über alle Elemente eines Arrays, ohne ein Element zu verändern
- Kumulieren der Elemente im Array aus dem Ergebnis des Blocks, liefert das kumulierte Ergebnis

▶ `a = [1,2,3,4]`

▶ `a.inject(0){|sum, x| sum + x}` → 10

▶ `a.inject(1){|sum, x| sum * x}` → 24

▶ `a.inject(1){|sum, x| sum * x if x <4}` → 6

▶ `Enumerable#inject`

Arbeiten mit Hashes



- Elementzugriff
- Traversieren über Key-Value-Paare eines Hashes - `each`
- Traversieren über Keys oder Values eines Hashes - `each_key`, `each_value`

Arbeiten mit Hashes I



```
▶ h = { :a => 1, :b => 2, :c => 3 }
```

- Elementzugriff

```
▶ h.include?(:a)           → true
  h.has_key?(:a)          → true # Synonym
  h.has_value(3)          → true
  h.has_value(1000)       → false
```

```
▶ h[:c]                   → 3
  h[:another_symbol]     → nil
```

- Traversieren über die Key-Value Paare eines Hashes

```
▶ h.each { |k,v| print "#{k}=# {v} " } → a=1 b=2 c=3
```

Arbeiten mit Hashes II



▶ `h = { :a => 1, :b => 2, :c => 3 }`

- Traversieren über die Keys oder Values eines Hashes

▶ `h.each_key { |k| print "#{k}; " }` → `a; b; c;`

▶ `h.each_value { |v| print "#{v}; " }` → `1; 2; 3;`

Object Handling - Klassendefinition



- ▶ # sale.rb

```
class Sale < ActiveRecord::Base      # Klasse definieren
end
```

- ▶ # somewhere.rb

```
class Sale < ActiveRecord::Base      # Klasse redefinieren
  include Comparable
  # ...
end
```

- ▶ # somehow.rb

```
class Object                          # Rootklasse redefinieren
  include ModelGuard
end
```

Mixins



- Hinzufügen von Funktionalität von Modulen
Beispiel: Module Comparable

```
▶ class BusinessPeriod
  include Comparable

  def <=>(other)
    ...
  end
end
```

- Einbinden des Moduls, Implementation Methode <=>:
- Vorteil: Die Methode unterstützt nun alle Vergleichsoperatoren <, <=, >, >=, == plus die Methode between?

Mixins - Beispiel Comparable



```
▶ class A
  include Comparable

  attr :a

  def initialize(a)
    @a = a
  end

  def <=>(other)
    @a <=> other.a
  end
end
```

```
▶ a1 = A.new(3)
  a2 = A.new(5)
  a3 = A.new(7)

▶ # Comparable methods
  a1 < a2           → true
  a1 == a2          → false
  a1 >= a3          → false

▶ a2.between?(a1, a2) → true

▶ # Enumerable methods (Array)
  [a3, a1, a2].sort → [3, 5, 7]

▶ [a3, a1, a2].min  → 3
  [a3, a1, a2].max  → 7
```

Reflection - Methoden



- Methoden ermitteln

- ▶ `r = 1..10` # Range object

- ▶ `list = r.methods` → [...]

- ▶ `list.length` → 118

- ▶ `list[0..3]` → ["map", "display", "any?", "=="]

- ▶ `r.respond_to?("frozen?")` → true
 - `r.respond_to?("has_key?")` → false

Reflection - Klassen



- Klasseninformationen ermitteln

▶ `num = 1`

```
num.object_id      → 118  
num.class         → Fixnum
```

```
num.kind_of? Fixnum → true  
num.kind_of? Numeric → true
```

```
num.instance_of? Fixnum → true  
num.instance_of? Numeric → false
```

Reflection - Dynamische Methodenaufrufe



- Methoden können dynamisch aufgerufen werden
- Rails: Scaffolding (`model_object.send`)

▶ `num = 100`

▶ `num.send("to_s")` → `"100"`

▶ `eval("num.to_s")` → `"100"`

Reflection - Beispiel



```
▶ class Account < ActiveRecord::Base
  end
```

```
class ModelFactory
  include Singleton
```

```
  def model_class
    eval("#{self.class.to_s.sub(/Factory/, '')}")
  end
end
```

```
class AccountFactory < ModelFactory
  end
```

```
▶ AccountFactory.instance().model_class → Account
```

Ruby Standard Library (kleine Auswahl)



- BigDecimal / Complex / Rational - Zahlenformate
- CSV - Handling von Files/Strings im CSV-Format
- Date / DateTime
- GetOptLong / OptionParser - Command Line Parsing
- Set, REXML, Test::Unit, YAML,
- Networking
- ...

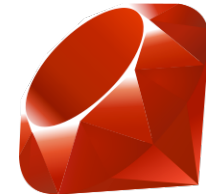
Literatur, Links



- Dave Thomas: Programming Ruby: The Pragmatic Programmers Guide
„The Pickaxe“
- <http://www.ruby-doc.org/core/>
API Ruby Core Library
- <http://stdlib.rubyonrails.org/>
API Ruby Standard Library
- <http://del.icio.us/ewaldkoenig/ruby>
Ruby-tagged Bookmarks EK

Fragen?





Kontakt

Ewald König
Leiter Software Entwicklung
Profics GmbH, Winterthur

ek@profics.ch

office +41 (52) 218 10 17

mobil +41 (79) 616 10 17

profics 

The Profics logo consists of the word "profics" in a lowercase, sans-serif font, followed by a square icon containing a white circuit board pattern on a blue background.