HPI Hasso Plattner Institut

Metaprogrammierung

extend

HPI Hasso Plattner Institut

Erweitert Objekt selbst

- Einzelnes Objekt um neue Methoden ergänzen

  z.B. für Data Context Interaction (DCI)

```ruby
module Extension
  def my_method
    self
  end
end

obj = Object.new
obj.extend Extension
obj.my_method # => #<Object:0x00007ff07a20ba08>
```

extend

Erweitert Objekt selbst

- Klassenobjekt selbst erweitern
  z.B. mit Metaprogrammiermethoden

```
module Extension
  def my_method
    self
  end
end

class A
  extend Extension
  self.respond_to?(:my_method) # => true
  my_method # => A
end
```

extend

Beispiel

- Einfaches #attr_writer mit #define_method implementieren

```ruby
module AttributeWriter
  def attribute_writer(attribute_name)
    define_method("#{attribute_name}=") do |new_value|
      instance_variable_set "@#{attribute_name}", new_value
    end
  end
end

class MyClass
  extend AttributeWriter

  attribute_writer :my_attribute
end

MyClass.new.my_attribute = 5
# => 5
```

Struct

Container für strukturierte Daten

- Erzeugt Klasse mit Getter, Setter und Konstruktor für angegebene Attributnamen

```ruby
Struct.new(:name, :age)
# => #<Class:0x00007ff079550070>

Person = Struct.new(:name, :age)

john = Person.new("John", 18)
john.name # => “John”
john.age  # => 18

john.age = 20
john.age  # => 20
```

#send

HPI Hasso Plattner Institut

Message Passing

- Detail: Ruby führt nicht die Methode aus, sondern schickt eine Nachricht an das Objekt

```ruby
string = "Hello Ruby"

string.send :downcase
# => "hello ruby"
```

#method_missing

HPI Hasso Plattner Institut

Auf unbekannte Methoden antworten

- Aufruf von `#method_missing` für jede unbekannte “Nachricht”

```ruby
class A
  def method_missing(mth, *args, &block)
    puts "Die Methode #{mth} gibt’s nicht..."
  end
end

A.new.does_this_work?
# Die Methode does_this_work? gibt’s nicht…
# => nil
```

def object.name

Methode auf Objekt definieren

- Ähnlich zu `#extend`

```ruby
obj = Object.new

def obj.my_method
  "Hello!"
end

obj.my_method
# => "Hello!"
```

def object.name

Methode auf Objekt definieren

- Grundlage für Klassenmethoden, da `self` innerhalb von `class...end` das Klassenobjekt referenziert

```ruby
class MyClass
  self # => MyClass

  def self.my_class_method
    "Hello!"
  end
end

# Äquivalent zu:
def MyClass.my_class_method
  "Hello!"
end
```

Unendliche Möglichkeiten

`Module#prepend`

- Wie `#include`, aber vor der eigenen Klasse im Lookup

`Forwardable`

- Methodenaufrufe weiterleiten via Metaprogrammierung
- Dekorator-Muster z.B. mit `Delegator`

`#instance_eval`

- Blöcke in einem anderen Kontext ausführen, z.B. innerhalb eines Objektes
- Grundlage für Domain-Specific Languages (DSL)

Und vieles mehr…

HPI Hasso Plattner Institut

Metaprogrammierung