Proposal for Anonymous Symbols and Anonymous Methods

Anonymous Methods (AnonMeths) can be used for complex orthogonal behaviors that dispatch by receiver class without patching core or other sensitive classes in a globally visible manner. AnonMeths are located by Anonymous Symbols (AnonSyms).

AnonSyms do not have parseable names, and can only be referenced by value, limiting namespace problems and promoting encapsulation.

AnonMeths are GCed once the AnonSym bound to them are GCed.

AnonMeths would not appear in Object#methods, thus will not confuse introspection.

Assume:

Symbol.new() => #Symbol:123412 # an AnonSymbol than can never be parsed in ruby code.
anon_sym = Symbol.new() # an AnonSym in a variable that can be closed-over or passed by value.

Optional Supporting Syntax:

a.*anon_sym(args...) # equiv. to a.send(anon_sym, args...)
class A
def *anon_sym(args...); body...; end
end
equiv. to:

class A
define_method(anon_sym) {| args... | body... }
end

AnonSyms are not added directly to a Module's internal symbol-to-method table. Instead, each AnonSym has an internal module-to-method table that is GCed when the AnonSym is GCed.

rcvr.send(anon_sym, ...)

will use anon_sym's module-to-method table to locate a method based on usual the receiver's module lookup chain.

Example Application:

Typical visitor pattern that pollutes Array and Object method namespaces:

class Array; def visit(visitor); each { | elem | elem.visit(visitor); } end; end
class Object; def visit(visitor); visitor.something(self); end; end

Functional alternative using "case ...; when ...":

def visit(obj, visitor)
case obj
  when Array
    obj.each { | elem | visit(elem, visitor) }
  else
    visitor.something(obj)
end
AnonMeth version:

def visit(obj, visitor)
    sel = Symbol.new
    class Array;   def *sel(visitor); each { | elem | elem.*sel(visitor) }; end; end
    class Object;  def *sel(visitor); visitor.something(self); end; end
    obj.*sel(visitor)
end

Imagine that visit() needs dynamic hooks to visit different types:

def visit(obj, visitor)
    sel = Symbol.new
    class Array;   def *sel(visitor); each { | elem | elem.*sel(visitor) }; end; end
    class Object;  def *sel(visitor); visitor.something(self); end; end
    add_visit_methods!(sel)
    obj.*sel(visitor)
end

def add_visit_methods!(sel)
    class Hash;    def *sel(visitor); each { | k, v | v.*sel(visitor) }; end; end
    ...
end

The AnonSym send "rcvr.*sel(...)" dispatches, like a normal method send, directly to the appropriate AnonMeth for **sel**.
visit() can be extended dynamically by adding more AnonMeths bound to **sel**.

The functional "case ...; when..." version is difficult to extend and maintain and is likely to not perform as well as anon messages. This is similar in style to Scheme letrecs, but is object-oriented.

This idea could be extended to Anonymous Ivars to resolve other namespacing and encapsulation issues for mixins that require state.

-- Kurt Stephens

Related issues:
Related to Ruby master - Feature #4288: Allow invoking arbitrary method names... Closed

History

#1 - 10/04/2011 05:29 AM - Anonymous
On Oct 3, 2011, at 3:11 PM, Kurt Stephens wrote:

Optional Supporting Syntax:

    a.*anon_sym(args...) # equiv. to a.send(anon_sym, args...)
    class A
        def *anon_sym(args...); body...; end
    end

I'm not sure about the anonymous symbols but you might want to take a look at this discussion:

http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/34550

regarding using strings/expressions to invoke methods without the use of send().

Gary Wright

#2 - 10/04/2011 10:31 AM - kstephens (Kurt Stephens)
Related: http://redmine.ruby-lang.org/issues/show/4288

#3 - 10/04/2011 07:59 PM - mame (Yusuke Endoh)
Hello,

To all:
This proposal aims to address pollution and collision problem of name
space, as well as refinement. But the approach is different. Though the explanation is not so catchy :-), language designers may want to check it out.

To Kurt:
I recommend you to clarify the advantages and disadvantages between your proposal and the traditional ones (like refinement and classbox).

I have some points. Because class statement changes the scope of local variables, the following code will not work:

```ruby
anon_sym = Symbol.new()
class A
def *anon_sym(args...); body...; end
end
```

And, the syntax "a."anon_sym" collides against the call to method *

```ruby
x, y = 3, 2
p(x.*y) #=> 6
```

The letter is not essential problem, but the former is not negligible.

#4288 aims to provide chemistry with other languages and the discussion focuses on "syntax game", so it is better to distinguish between them.

--
Yusuke Endoh mame@tsg.ne.jp

---

#4 - 10/05/2011 01:53 AM - nobu (Nobuyoshi Nakada)

Hi,

(11/10/04 4:11), Kurt Stephens wrote:

```
Imagine that visit() needs dynamic hooks to visit different types:

    def visit(obj, visitor)
    sel = Symbol.new
    class Array;   def *sel(visitor); each { | elem | elem.*sel(visitor) }; end; end
    class Object;  def *sel(visitor); visitor.something(self); end; end
    add_visit_methods!(sel)
    obj.*sel(visitor)
    end
    class Hash;    def *sel(visitor); each { | k, v | v.*sel(visitor) ; end; end
    ...  
    end
```

The AnonSym send "rcvr.*sel(...)" dispatches, like a normal method send, directly to the appropriate AnonMeth for "*sel".
visit() can be extended dynamically by adding more AnonMeths bound to "*sel".
The functional "case ...; when..." version is difficult to extend and maintain and is likely to not perform as well as anon messages.
This is similar in style to Scheme letrecs, but is object-oriented.

Is this double dispatching system your true goal? If so, anonymous symbol doesn't seem the only way nor the best way, for me.

--

Nobu Nakada

---

#5 - 03/27/2012 03:33 AM - mame (Yusuke Endoh)

- Status changed from Open to Feedback

#6 - 06/27/2012 09:21 PM - alexeymuranov (Alexey Muranov)

I just thought myself about anonymous methods and wanted to post it here with "Joke" status. I thought about a possibility to change at runtime the method name, or switch two method bodies, and not to require that every method have a name. With every object there would be associated a Set of methods, some named, some not, some having multiple names.

#7 - 11/20/2012 11:01 PM - mame (Yusuke Endoh)

- Target version set to 2.6
I refuse to add new syntax described in the proposal. And I think refinement will do the task the OP intended by anon symbols. So I mark this 'rejected'. Feel free to reopen it. But I expect you to persuade me.

Matz.