# Ruby - Feature #6711

## Optional typing and method overloading

07/09/2012 10:15 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

| | |
|---|---|
| **Status:** | Rejected |
| **Priority:** | Normal |
| **Assignee:** | matz (Yukihiro Matsumoto) |
| **Target version:** | 3.0 |

**Description**

Today I woke up with some ideas to overcome some of the features I miss in Ruby. I've searched for similar issues and I found this #5583, which was rejected.

But my idea is a bit different and also concerns about method overloading. So I decided to open a new issue. I'll be also creating another feature request soon for introducing super! keyword for overrides of methods by reopening classes.

#5583 wants the checks to be made at compile time, while I think it should be a dynamic check instead.

For example:

```
class A
def some_method(a)
"a"
end

def some_method(a # Numeric)
"a is a number"
end

def some_method(a, b)
"untyped a and b"
end

def some_method(a, b # String)
"typed b"
end

def some_method(a # String, b = 2)
"typed a"
end
end

class B < A
def! some_method(*args) # overrides all overloaded methods - *args is not required for this
if args[0].is_a? Number
super(args[0])
else
'do something else'
end
end
end
```

The rule to decide over multiple alternatives should be by order:

1 - max number of better matched argument types # Float is a better match for 1.2 than Numeric
2 - max number of matched argument types
3 - max number of consecutive matched argument types

Example:

```
a = A.new
b = B.new

b.some_method == 'do something else'
```

b.some_method(1.3) == 'a is a number'
b.some_method('s') == 'do something else'
a.some_method('s') == 'typed a'
a.some_method('s', 's') == 'typed a'
a.some_method(:s, 's') == 'typed b'
a.some_method(:s) == 'a'
a.some_method(1, :s) == 'untyped a and b'

Current method and send are not affected. Argument-matching checks would happen at runtime

The goal is to make programmers happier, not be perform better.

m = a.method :some_method
m.call # raise arguments mismatch exception
m.call(:a) == 'a'

To get a specific method one could ask for it like:

m = a.method :some_method, nil, String # some_method(a, b # String)

We should also introduce "methods" for getting a list of all named methods defined:

# HTTP server dispatcher class example:

```
def handle_http_request(params)
methods = controller.methods :some_method
raise "Overloaded actions are not supported" unless methods.size == 1
action = methods.first
method_arguments = []
action.arguments.each do |name, type, default_value = nil|
hk = params.has_key? name
(method_arguments << hk ? params[name] : default_value; next) unless hk && !type.nil?
method_arguments << method(:bind_value, type).call params[name], type
end
action.call *method_arguments
end

def bind_value(value, type)
value
end

def bind_value(value, type # Integer)
value.to_i
end

def bind_value(value, type # Float)
value.to_f
end

def bind_value(value, type # Numeric) # BigDecimal and Numeric would match for instance
BigDecimal.new value
end

def bind_value(value, type # Date)
Date.parse value
end

#...
```

**Related issues:**

| | | |
|---|---|---|
| Related to Ruby - Feature #5583: Optionally typing | **Rejected** | **11/07/2011** |

**History**

**#1 - 07/09/2012 10:26 PM - rosenfeld (Rodrigo Rosenfeld Rosas)**

Actually the precedence should be:

1 - max number of matched argument types
2 - max number of better matched argument types # Float is a better match for 1.2 than Numeric
3 - max number of consecutive matched argument types

## #2 - 07/10/2012 02:30 AM - matz (Yukihiro Matsumoto)

*- Status changed from Open to Rejected*

Good try, but:

- You cannot use # sign as type specifier, since it's already used as a comment sign.
- You didn't defined how type match should be done, but simple class hierarchy match would hinder duck typing.

Until above concerns are addressed, I have to reject the proposal.

Matz.

## #3 - 07/10/2012 05:25 AM - jballanc (Joshua Ballanco)

Just a humble suggestion, but it seems like what you want is not so much method overloading as it is multidispatch. Clojure has a feature like this, and there is a gem that simulates the same: https://github.com/psantacl/ruby-multimethods . If you'd like to re-work your proposal, you might want to think about how multidispatch could be better supported by the Ruby language.

## #4 - 07/10/2012 11:18 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

matz (Yukihiro Matsumoto) wrote:

> Good try, but:
>
> - You cannot use # sign as type specifier, since it's already used as a comment sign.

Wow, how did I completely forget about this?! :P

Well, I need to spend some time thinking in some valid syntax that won't conflict with named arguments.

> - You didn't defined how type match should be done, but simple class hierarchy match would hinder duck typing.

I've defined how type match works. It is just a matter that you don't agree with it :)

The motivation about simple class hierarchy (and not caring about duck typing) is for the usual cases I have in mind like a generic Numeric and specifics Float, Integer and BigDecimal.

Duck typing is simply ortogonal to typed arguments in my opinion.

Could you please elaborate on your opinions about typed arguments and duck typing? I couldn't understand your comment concerning this in #5583 either.

But I agree that I wasn't specific about the rules for included modules ("multiple inheritance"). The problem is that I don't have a real use case for this so this is a bit hard for me to define those rules at this point. I'll have to think more about this.

## #5 - 07/10/2012 11:26 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

jballanc (Joshua Ballanco) wrote:

> Just a humble suggestion, but it seems like what you want is not so much method overloading as it is multidispatch. Clojure has a feature like this, and there is a gem that simulates the same: https://github.com/psantacl/ruby-multimethods . If you'd like to re-work your proposal, you might want to think about how multidispatch could be better supported by the Ruby language.

Perl 6 also calls it "multi dispatcher":

http://perlgeek.de/en/article/5-to-6#post_21

But Java and C++ call them overloaded methods/functions, so I'm not sure if there is any conceptual difference among them...

I certainly want to re-work this proposal, but I'll wait the inspiration to come after another great night of sleep instead of forcing it to come :)