Ruby - Feature #5583

Optionally typing

11/07/2011 12:49 PM - technohippy (Yasushi ANDO)

Status:	Rejected
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	

Description

Although I know all of you dislike static typing it cannot be denied that there are some people aspire for introducing it in ruby. The dartlang solved the problem in unique way called Optionally Typing. In Dart you can declare types for variables but they are ignored by the compiler with default options.

It looks superb idea for me so that I introduced the optionally typing to ruby as a trial. You can write types for variables if you want then the ruby completely ignores these dirts.

Sample code:

```
def method(arg)
  arg
end

def typed_method(arg) : String
  arg
end

def fully_typed_method(arg : String) : String
  arg
end

var = 'var'
puts var

var_with_type : String = 'var_with_type'
puts var_with_type

var_from_method : String = typed_method('var_from_method')
puts var_from_method

@ivar : String = fully_typed_method('@ivar')
puts @ivar
```

Results:

```
$ ./truby -Ilib -I. truby_test.rb
var
var_with_type
var_from_method
@ivar
```

I attached my poor patch. Thanks for your consideration.

Related issues:

Related to Ruby - Feature #6711: Optional typing and method overloading	**Rejected**	**07/09/2012**

History

#1 - 11/07/2011 06:53 PM - dohzya (Etienne Vallette d'Osia)

+1

TL;DR

I love the idea!

I don't know if optional typing is the solution, but the more I use Ruby,
the more I miss the robustness that could provide powerful static analysis tools.

Moreover, I often read Ruby code, and it is *always* clearer when types are provided in doc.
Why writing types in comments should be better than writing them directly in code?

Of course, to be accurate, the typing system should deals with generics (ouch), lambda,
non-yet-defined-classes, respond_to types (but these ones could be seen like implicit module!
Simpler? Oh wait…)

I don't really want to see my favorite language became a Java-like one,
but I'm sure it's possible to handle these cases without adding too much complexity,
at least with adding complexity on the sides, leaving simple code simple :-)

#2 - 11/07/2011 07:17 PM - alexeymuranov (Alexey Muranov)

TypeError exists in Ruby ([1,2]["a"]), why not to have optional static typing to avoid it when possible. I do not think it can hurt.

#3 - 11/14/2011 10:24 AM - technohippy (Yasushi ANDO)

- Status changed from Open to Rejected

No one comments on this issue for a week so that this is rejected. I'm sorry for proposing a dull issue and I'll be back with more fascinating proposal someday.

#4 - 11/14/2011 12:34 PM - trans (Thomas Sawyer)

I don't think it's dull. Actually I think it is obvious future direction. Type information can be very useful for mission critical systems as well as optimization for compilers. While it is not something I'd likely ever use, I've heard other wish they could.

I agree types should be ignored by default, but a "strict" run mode could use/enforce them.

#5 - 11/14/2011 02:23 PM - mrkn (Kenta Murata)

I like this idea, and I cannot think this is a joke.
I believe Ruby should have optional typing system in the future.
Please reopen this issue and continue discussion.

#6 - 11/14/2011 02:42 PM - sorah (Sorah Fukumori)

- Category deleted (Joke)

- Status changed from Rejected to Open

Let's continue this discussion as not a joke.

#7 - 11/14/2011 03:29 PM - matz (Yukihiro Matsumoto)

Hi,

In message "Re: [ruby-core:41006] [ruby-trunk - Feature #5583] Optionally typing"
on Mon, 14 Nov 2011 14:23:14 +0900, Kenta Murata muraken@gmail.com writes:

|I like this idea, and I cannot think this is a joke.
|I believe Ruby should have optional typing system in the future.
|Please reopen this issue and continue discussion.

(a) this syntax conflicts with (already decided) keyword arguments.
(b) in dynamic language like Ruby, optional typing should honor duck
typing, I don't think previous discussion consider this aspect
deeply enough.

```
matz.
```

#8 - 11/15/2011 12:15 AM - trans (Thomas Sawyer)

@matz (Yukihiro Matsumoto)

(a) plenty of other ways to define syntax. maybe best way is to separate it from def since it is optional, e.g.

sig foo(String)

def foo(string)
...
end

(b) not sure how it can honor duck typing. i think the whole idea is to (optionally) roast the duck!

#9 - 11/15/2011 12:30 AM - alexeymuranov (Alexey Muranov)

Yukihiro Matsumoto wrote:

> (b) in dynamic language like Ruby, optional typing should honor duck
> typing, I don't think previous discussion consider this aspect
> deeply enough.

My understanding of how it can possibly work was the following: when foo method that requires a string argument is called, it first of all calls the argument's to_s method (raises an error if there is no such method). I think this is supposed to help catch bugs early and to make the code better documented.
For typed variables, it seems that their class has to be "frozen", and no singleton methods allowed.

#10 - 11/15/2011 12:53 AM - shugo (Shugo Maeda)

Hi,

2011/11/15 Thomas Sawyer transfire@gmail.com:

> (a) plenty of other ways to define syntax. maybe best way is to separate it from def since it is optional, e.g.
>
> Â sig foo(String)
>
> Â def foo(string)
> Â Â ...
> Â end

I prefer method annotations to special syntax.

> (b) not sure how it can honor duck typing. i think the whole idea is to (optionally) roast the duck!

FYI, Scala supports statically checked duck typing. See
http://markthomas.info/blog/?pf.

--
Shugo Maeda

#11 - 11/15/2011 01:20 AM - trans (Thomas Sawyer)

oh, i see what you mean. so in pseudo code:

sig foo(:to_s)

def foo(stringy_thing)

to mean argument must respond to #to_s.

#12 - 11/15/2011 02:29 AM - matz (Yukihiro Matsumoto)

Hi,

In message "Re: [ruby-core:41023] Re: [ruby-trunk - Feature #5583] Optionally typing"
on Tue, 15 Nov 2011 00:36:34 +0900, Shugo Maeda shugo@ruby-lang.org writes:

|FYI, Scala supports statically checked duck typing. See
|http://markthomas.info/blog/?p=66.

It's not concise at all. The beauty of duck typing lives in
its implicitness (I know it's demerit as well).

```
matz.
```

#13 - 11/15/2011 10:53 AM - shugo (Shugo Maeda)

Hi,

2011/11/15 Yukihiro Matsumoto matz@ruby-lang.org:

> |FYI, Scala supports statically checked duck typing. Â See
> |http://markthomas.info/blog/?pf.

> It's not concise at all. Â The beauty of duck typing lives in its implicitness (I know it's demerit as well).

I know that you don't like it, and neither do I. Signatures in GNU C++ (http://docs.freebsd.org/info/gcc/gcc.info.C++_Signatures.html) look better, but it's not enough. If both signatures and argument type inference are available, it would be perfect, but those who like static typing may disagree.

--
Shugo Maeda

#14 - 11/25/2011 09:54 PM - alexeymuranov (Alexey Muranov)

Excuse me if this thread is not a good place to post this question, but i will appreciate any clarification or a link to an answer.

> What is Duck Typing after all, and how is it supposed to work?

According to Wikipedia:

> "duck typing is a style of dynamic typing in which an object's current set of methods and properties determines the valid semantics, rather than its inheritance from a particular class"

But what is the "set of methods"? If it is the set of their *names*, then what if unrelated methods of different objects happen to have the same name? Is there some written requirement about Ruby that, for example, #to_s method has to always convert an object to String? Also, if duck typing is the only typing allowed, then how would you define what String is?

I have been thinking how to check the type of an object, which can be needed, for example, to implement #deep_value (#5531) or other #deep_ method for a hash, and the only nice way i could think of was to use #is_a? method. This would also work if i would want to group classes in some way by their behavior. For example, there are many classes that implement #[] method (Array, Fixnum, Hash, String), so checking if this instance method exists in a class does not always give adequate information about its behavior. So a natural way to group together classes that use this method to store and index objects, like Array and Hash, would be to create an empty module with a unique name, Type::IndexedObjects for example, mix it into Array and Hash and any other class where #[] method behaves similarly, and to use #is_a?(Type::IndexedObjects) to find out the true behavior of an object. But Wikipedia says explicitly that this will not be a Duck Typing, and that in Ruby there shall be only Duck Typing.

-Alexey Muranov.

#15 - 11/27/2011 07:36 PM - amro256 (Belhorma Bendebiche)

I don't think that duck typing is an issue with optional static typing.

1. It doesn't happen at runtime
2. It only happens if you pass a flag to the interpreter
3. is obvious, the whole point of static typing is that your program doesn't blow up at runtime, so raising errors based on it is pointless.
   And with 2), the worst case scenario is you get a warning if you're looking for one. Either way, duck typing works as usual.

This may sound weird but it works out very well in practice (and I have enough practice with AS3 to be able to tell). Even if you strictly type an API, a user can still pass whatever arguments they fancy if they choose to ignore the types, so it's a win-win.

As for syntax:

def foo(bar -> String, baz -> *) -> String

end

Where * or some other token is used to explicitly state dynamic type. It's the same as not providing a type, but in my experience it helps to document it explicitly.

#16 - 11/27/2011 10:47 PM - alexeymuranov (Alexey Muranov)

Belhorma Bendebiche wrote:

> I don't think that duck typing is an issue with optional static typing.
>
> 1. It doesn't happen at runtime

A clash between static and dynamic typing can be observed like this (i think there can be more realistic cases):

```
def foo(bar -> String) -> String
  ...
end

def random_class
  it_is_raining_now? ? String : Array
end

x = random_class.new
foo(x)
```

So it seems to me that static typing has to be checked in runtime, and the benefits will be:

1. better documentation of the code,
2. errors will be raised earlier (at the moment of calling foo in the above example) or will be raised in cases where they wouldn't be raised at all otherwise, and could lead to subtle errors.
3. (Update) possibility to have "method overloading" based on argument types.

Maybe i do not use terminology correctly and it should not be called static typing in such case.

I propose instead of passing a flag to the interpreter, to do

```
require 'types'
```

in the source, which would mix in some predefined Type:: modules into standard classes, and would add hooks for checking for their presence with #is_a?(Type::...) when a type declaration is present.

-Alexey.

Update Declaring method signature dynamically can work like this, for example:

```
method_signature :foo, [Type::String, Type::Integer], Type::String

def foo(s, n)
  s * n
end
```

#17 - 02/25/2012 01:23 PM - ko1 (Koichi Sasada)

Shugo Maeda wrote:

> I prefer method annotations to special syntax.

Me too. Any discussion about it?

#18 - 03/06/2012 12:52 AM - shugo (Shugo Maeda)

Hi,

Koichi Sasada wrote:

> Shugo Maeda wrote:
>
> > I prefer method annotations to special syntax.
>
> Me too. Any discussion about it?

Once Matz rejected a proposal of method annotations because he didn't like that methods had such attributes at runtime. It might be acceptable if method annotations have comment-style syntax and are used only at compilation (or another static analysis) time.

Do you have any idea about method annotations in mind, ko1?
Do we have to file a new ticket for method annotations?

#19 - 03/27/2012 11:55 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

#20 - 03/28/2012 01:43 AM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

Although type notation as a comment a la Dart language is pretty interesting, I have to reject this proposal, since the proposed syntax is conflicting.

Matz.

#21 - 07/31/2012 08:05 PM - alexeymuranov (Alexey Muranov)

Maybe, as a form of typing, Ruby can somehow implement *contracts*, like in http://en.wikipedia.org/wiki/Design_by_contract ?
Some "contract description language" would need to be invented, and classes or methods would then have a possibility to declare contracts they fulfill:

```
method_propery C
this method returns a UTF-8 string
end

class_property D
...
end

class_property E
...
end

class T
satisfy :to_s => C
satisfy D & E

def to_s
...
end
end
```

Then other methods would be able to ask a class if it fulfills a given contract. This would make more sense than simply asking a class if its instances respond to a method with a given name.

Also this could be useful for automatic testing, automatic documentation, and can reduce the amount of necessary comments.

I do not have any clear idea.

P.S. I think it would be inevitable in addition to objects and classes, to have to define *values* and *primitive types*, like string, integer, etc.

Files

optionally-typing.patch	3.84 KB	11/07/2011	technohippy (Yasushi ANDO)