Ruby master - Feature #14164

[Suggestion] Type system for ruby 3x to be usable for e. g. rubocop or autogenerating crystal code and so forth

12/09/2017 03:40 PM - shevegen (Robert A. Heiler)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

Hello everyone, especially the ruby core team, nobu and of course matz,

This is an idea for ruby 3.x, but the idea is not ... well, really
finished.

It is more of an idea.

As I am aware that working with "unfinished" ideas is not easy, please
feel free to close this suggestion at any moment in time. It is more
meant a bit to point towards what nobody else may have pointed out
before ... :)

So the situation:

- Matz spoke a few times about a "type system" for ruby (3.x). Now, this can be anything really, but matz also specified his idea a few times.

One idea that I think matz mentioned here or there was that it will be
optional. That is, people can use ruby light-weight as always, without
being forced into mandatory types and having to change the way they
may have written ruby for the past x years.

There is one blog/interview entry here mentioning types as well, at:

https://blog.heroku.com/ruby-3-by-3

I think matz mentioned a type system a few times lateron when he gave
a presentation, and he also commented on a few issues about the
(possible) type system in different suggestions made at the ruby
tracker here.

In particular, I remember that matz wrote something to a suggestion by
another japanese ruby hacker some time ago about ... I think it had to
do with specifying certain behaviour of objects/methods.

I am sorry that I can not describe this in more detail since I forgot
the name of the one who suggested it; I myself do not fully understand
or know what will come with ruby 3.x. The above URL to the blog/interview
may give an indication as to what MIGHT come eventually.

Back then matz said:

"[...] the third major goal of the Ruby 3 is adding some kind of
static typing while keeping the duck typing, so some kind of
structure for soft-typing or something like that. The main goal
of the type system is to detect errors early. So adding this
kind of static type check or type interfaces does not affect
runtime."

"It's just a compile time check. Maybe you can use that kind of
information in IDEs so that the editors can use that data for their
code completion or something like that, but not for performance

improvement."

Ok, so far so good. Matz actually did not necessarily mean it
for performance improvement per se.

But ... here I was thinking ...

I guess many ruby people know the programming language crystal or
have heard about it. Matz even made a considerable donation a
few years ago. :)

Crystal is in many ways similar to ruby; the syntax is somewhat
similar (if we ignore the type system and macros perhaps).

Several ruby people use crystal, too, like jhass and others (I
don't as of yet, largely due to lack of time, inertia and my
own laziness; I even have learning elixir on my todo list and
just did not get around it really).

However had, I somewhat jokingly said in the past that perhaps
we could tune rubocop to generate crystal code. Since the difference
should not be that far apart from ruby code.

It was more meant as a joke though, but ... I think in theory that
could be possible, right? Perhaps not 100% valid crystal, but if
you specify some layout guide for rubocop to conform too, then the
autocorrect option from rubocop could perhaps generate a version
that is very close to the final crystal variant, excluding types
perhaps.

Not long ago, really just a few days ago, I actually remembered that
ruby may have some form of (optional) typing system in the future.

And here I wondered:

- What if this type system could be used, in an optional way, for ruby people to also be able to auto-generate valid crystal code, via an indirection? This indirection could be rubocop, but it could also be part of MRI itself (once the type system is all in place; a bit like the RubyVM).

I do not know how difficult this is but I thought that this
would be pretty cool to have. Write Ruby - have Crystal. :D

(This is meant in addition; I myself obviously prefer to write
ruby, and have the computer autogenerate crystal for me, so
that I can be lazy. :P)

We could write ... ruby ... just as-is, for everyday scripts.

And then at a later time, we may add additional information, e. g.
with any form of type system, IF we need it. And then, ruby
could make use of that *additional* information to not only
consider speeding up ruby code (perhaps; or perhaps not), but
to also ... have it generate valid crystal code! :D

This is perhaps more wishful thinking on my part or for a
future xmas list; and it may also require the crystal team
to keep this in mind as well AND the ruby team.

But anyway, I only wanted to mention this for the time being;
matz once mentioned that ruby 3.x is more aimed towards the
year 2020, so ... +2 years I suppose, so there is still some
time left before 3.x will be released and "finalized".

I do not know how the crystal team thinks, neither how the
ruby team thinks about it - but I like my own idea. :D

Of course this also depends on what kind of type system there

will be, and how useful it may be. In the design stage, naturally
ruby comes first here (this is a bug tracker for ruby after all,
not for other languages), but I wonder about the second step,
where we could translate ruby-code into other languages code.

I should also note that I do not necessarily mean a "100%
correct output where nobody has to make any changes"; I think
any autogenerated output, that can be customized, would be
a "time win" for people in general - both when it comes to
having faster execution time for ruby code, but also lateron
to anyone who may want to use crystal rather than ruby primarily
because of the speed reason. (In my case, speed was never
a reason to use ruby so I have no reason to switch away
from ruby due to speed, but matz also said that nobody minds
if things are fast, so any speed improvement is a good
thing. And if my ruby code were to be almost valid crystal
code, then I'd only have to invest a tiny bit extra time
to get even more speed almost for free.)

Matz mentioned the additional information towards IDEs but
we could perhaps combine this with other tools. For example,
rubocop makes suggestions what to change; an optional type
system could also perhaps make some kind of analysis as to
what could be omitted or improved etc... a bit like the
did-you-mean-gem but working on a more "sophisticated"
level. (With a type system, I assume that a compiler has
an easier time knowing which constraints can be used etc...)

History

#1 - 12/09/2017 05:25 PM - Eregon (Benoit Daloze)

Here are some thoughts:
Although Crystal looks very similar syntactically, the semantics are fairly different: no runtime metaprogromming, no eval, no arbitrary precision integers by default and it is a statically-typed language (with good type inference to hide it).
So I think if we can auto-generate Crystal from Ruby code it will be either not compatible in many situations, or not significantly faster.
It's a bit like the idea to generate C code from Ruby code, but this gains almost nothing if expressed with the same dynamism (with rb_funcall, etc).
Maybe one could help a bit the semantics by being able to specify an argument if of type int64, but this seems to somewhat go against Integer unification.
Personally I would rather advocate a polyglot approach using multiple languages, by using a different language when requirements need it.

#2 - 10/08/2018 01:44 PM - jwmittag (Jörg W Mittag)

I feel that most discussions about "type systems" or "types" for Ruby suffer from a serious lack of unambiguous definitions, this discussion included. Somebody proposes something without precisely defining what *exactly* they are proposing, then somebody else argues for or against this proposal without precisely saying how they interpreted the proposal and so on and so forth.

There also seems to be a serious lack of understanding of type systems and typing in the Ruby community, which doesn't exactly help discussions about type systems and typing.

For example, this is (at least) the second issue on this tracker that mentions optional typing and performance. There are also several blog posts and tons of threads on ruby-lang. The thing is: optional typing can *by definition* not improve performance. In fact, optional typing is *defined* as "typing that does not influence runtime". That is what *makes* it "optional" in the first place. If optional types had an impact on performance, then there could be programs that run fast enough with the types, but too slow without, and for those programs, you would *need* the types, thus they would no longer be optional.

The *only* thing optional typing is allowed to do, is to reject a program as not type-safe. That's it. Adding or removing types can make a program type-check or not type-check, and that is *all* adding or removing types can do.

The powerful thing about optional typing is that, since it is optional, you are not tied to a single type system. You can choose the type system that works best for you. You can change type systems over the course of a project. You can mix multiple type systems within a project. You can even use multiple type systems at the same time for the same chunk of code.

Discussions about typing in Ruby always seem to confuse a lot of things: Type Annotations vs. Type Language vs. Type System vs. Type Checking. Gradual vs. Soft vs. Optional Typing. Explicit and Implicit Typing. Static and Dynamic Typing.

#3 - 08/11/2019 10:00 PM - jwmittag (Jörg W Mittag)

shevegen (Robert A. Heiler) wrote:

> Crystal is in many ways similar to ruby;

It really isn't. I don't know where this myth originated and why it is so widespread and persistent.

> the syntax is somewhat similar

Yes, that is true. However, of the three parts that make up a programming language (Semantics, Type System, Syntax), syntax is the least important part of a programming language, although admittedly the most argued about, possibly since it is the most visible and the most easily understandable part.

It is very easy to change the syntax of a language. See, for example, Vala and Genie which are actually more or less *the same language* just with different syntax. Vala has a C♯-inspired syntax, Genie has a Python-inspired syntax, but they both have the exact same semantics, exact same type system, etc. They are compiled by the same compiler and even have the same Syntax Tree.

On the other hand, C and ECMAScript have very similar syntax but radically different semantics.

> (if we ignore the type system and macros perhaps).

There is a movement in Programming Language Theory to see Type Theory as the Grand Unifying Theory of Programming Language Theory. In other words, there are researchers within PLT who argue that the Type System of a programming language is the *only thing* that matters. So, if we "ignore the type system", then we ignore the only thing that matters, and then you can indeed say that Crystal is very similar to Ruby, but only because you deliberately ignore the *one thing* that makes it completely different.

Personally, I think that is going too far and that Semantics are also important, but again, Crystal's semantics are also very different from Ruby's.

#4 - 08/12/2019 08:15 AM - zverok (Victor Shepelev)

> > Crystal is in many ways similar to ruby;
>
> It really isn't. I don't know where this myth originated and why it is so widespread and persistent.

To the best of my memory/understanding, it is not a myth, but a historical fact: Crystal started as an attempt to just make a "compiled Ruby" (probably with type system, but "invisible" kind of one, so on the surface it still looked totally like Ruby and was type-inferred all the way down), then the authors realized they'll need at least some changes to make it work, and somewhere at the road they understood the language is different enough to diverge whenever they want (for example, rethink parts of standard library, change names and APIs of core classes and so on).

But the main page still states

> Crystal’s syntax is heavily inspired by Ruby’s, so it feels natural to read and easy to write, and has the added benefit of a lower learning curve for experienced Ruby devs.

Obviously, neither of this makes the rest of your comment less true: the languages are quite different by their, so to speak, "spirit" already. I just wanted to point out that the "myth" has its hard justification (and that's why people frequently came with "let's borrow this from Crystal" ideas -- the same, to some extent, happens with Elixir, which also has Ruby-inspired syntax and intersecting communities, so every month somebody proposes to "finally" borrow the |> operator, which just will not make sense in Ruby).