Add `Bool` module

10/19/2020 12:11 AM - marcandre (Marc-Andre Lafortune)

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

Description

1-line Summary: rbs would benefit from the existence of common ancestor Bool for TrueClass and FalseClass.

Detail:
Matz: I am aware you rejected a similar request, but could we revisit this in light of RBS?

One use case was for an easy way to check for true or false values, instead of simply for truthiness (e.g. for data transfer, strict argument checking, testing, etc.)

I believe there's a new use case: RBS

In RBS, the most used types like String and Integer have types for "string-like" and "integer-like" objects: string and integer (all lowercase).

For example the signature for Integer#>> is:

```ruby
def >>: (int) -> Integer
```

It accepts an Integer or an object responding to to_int (summarized by int) and returns an Integer (and never another class of object responding to to_int or not).

There is a similar idea with boolean values, where a method may accept any object and will use it's truthiness, while returning true | false. For example one of the interface for Enumerable#all? should look like:

```ruby
def all?: () { (Elem) -> bool } -> true | false
```

The user supplied block can return any value, and its truthiness (anything else than nil or false) will be used to determine the result of all?. That result will be true | false, and no other value.

If RBS is to be popular, there will be many signatures for such predicates (in builtin Ruby, stdlib, any gems, applications, etc.). I feel the best option would be Bool, if this would be reflected in Ruby itself.

Proposal: a new global module called Bool, without any method of constant, included in TrueClass and FalseClass.

Following reasons for rejection were given at the time:

many gems and libraries had already introduced Boolean class. I don't want to break them.

I looked and found the bool gem that defines a Bool module. My proposal is compatible. In any case, this gem looks abandoned, the author Aslak Hellesøy doesn't have the code on github, the gem has had 7000 downloads in the past 6 years and has no public reverse dependency. It also fails to install on my machine.

I am not aware of incompatibilities.

true and false are the only representative of true-false values. In Ruby, nil and false are falsy values, and everything else is a true value. There's no meaning for having a superclass of TrueClass and FalseClass as Boolean.

The proposal is exactly to be able to easily write about this duality of Bool as having only true and false as members, and every Ruby object as being implicitly convertible as being truthy or falsy (bool in RBS).

Discussion in RBS:

- [https://github.com/ruby/rbs/issues/133](https://github.com/ruby/rbs/issues/133)
Previous feature requests for Boolean:

- https://bugs.ruby-lang.org/issues/14224
- https://bugs.ruby-lang.org/issues/12515

History

#1 - 10/19/2020 12:15 AM - marcandre (Marc-Andre Lafortune)
- Project changed from CommonRuby to Ruby master

#2 - 10/19/2020 02:34 AM - mame (Yusuke Endoh)
Hi, I'd like to add the background.

Currently, RBS provides bool type as an alias to top type (a union type of all types). The rationale is because any type is actually accepted in the context of condition expression of if and while. Some methods that accept a predicate block, such as Enumerable#any? and #select, also accept any type as the return value of the block.

However, (almost) all methods that end with ? returns true | false. For example, of we write bool as the return type of Array#empty?, it means that the method may return any type, which is a bit less precise. true | false is a bit redundant, so marcandre (Marc-Andre Lafortune) wants to write Bool for it, but in RBS, a capitalized type corresponds to Ruby's class or module. So, to make the story simple, he is proposing adding a Bool module in Ruby side instead of RBS.

Personally, I'm unsure if it is good or not to change Ruby for this. If his proposal is accepted, the type of Enumerable#any? will be:

def any? : { (Elem) -> bool } -> Bool

This looks cryptic to me. I think that the current statement (following) is good enough.

def any? : { (Elem) -> bool } -> true | false

BTW, soutaro (Soutaro Matsumoto) (the original author of RBS) is now thinking the redefinition of bool as an alias to true | false. Based on soutaro's idea, the type will be:

def any? : { (Elem) -> top } -> bool

In terms of documentation, it loses the information that the return value of the block is considered as a condition, but I'm okay for it, too.

#3 - 10/19/2020 03:33 AM - shyouhei (Shyouhei Urabe)
Bool and bool to have different semantics sounds problematic to me. Not against this proposed Bool, but we need to rename bool then.

BTW e.g. class String; include Bool; end shall be prohibited somehow. Maybe a parent class is better?

#4 - 10/19/2020 12:20 PM - marcandre (Marc-Andre Lafortune)
shyouhei (Shyouhei Urabe) wrote in #note-3:

Boo and bool to have different semantics sounds problematic to me.

As I stated, String and string have different semantics. Is that also problematic?

Not against this proposed Bool, but we need to rename bool then.

That would defeat the whole purpose.

BTW e.g. class String; include Bool; end shall be prohibited somehow. Maybe a parent class is better?

A parent class would work too, but one might still write class Foo < Bool...

#5 - 10/19/2020 01:20 PM - Eregon (Benoit Daloze)
So Bool would be the "strict" variant, only true | false, and bool is anything convertible to a boolean, so any object, right?

I don't like the name Bool. If anything, I think it should be properly named Boolean.

I think adding Boolean on RBS' side (and not in Ruby) is not so bad.

#6 - 10/19/2020 11:52 PM - matz (Yukihiro Matsumoto)
Could you clarify the current proposal?

- bool / Bool / Boolean
- RBS side only / Ruby module as well

I am OK if RBS side only, no Ruby module.

Matz.

#7 - 10/22/2020 10:41 AM - shevegen (Robert A. Heiler)

I tried to make a shorter summary of my thoughts here, so just three points - hopefully that makes it easier to read:

(1) Most of the discussion, I think, happened at https://github.com/ruby/rbs/issues/133 - if I then understand the discussion correctly then it would mean that ruby would have to add "module Bool" or "module Boolean".

(2) One slight problem I see with that is that a use case originating (mostly) from RBS, even if it makes sense from the point of view of RBS, would potentially affect change in MRI ruby. I don't have a strong opposing opinion per se, but I think it would be better if the use case would originate from MRI directly, rather than the other way around. See also headius' comment in another issue about other implementations potentially affecting MRI via tests/specifications, without prior discussions. I am not involved in test/specs but I think I understood what headius meant here. This is one reason why I think it should be considered carefully whether change is really necessary in this regard. Keep also in mind that when a "module Bool / Boolean" exists in ruby, new users may ask how this should then be used, and it may be a bit complicated if the explanation is "because RBS uses it", even more so if these users may not need or use RBS (not everyone will use types; some may but others may not).

(3) I know way too little about the internals (admittedly I am not clever enough for high level C, and I am not even kidding here), but if the use case is primarily originating from RBS itself, could there not be another, indirect solution? For example, rather than requiring a change in MRI, could there not be some kind of meta-VM instead, that could be targeted? A bit like rubocop too, perhaps, for RBS? That way people could set it up for their own projects, adjust it as needed, and "simulate" as if a boolean "type" were to exist, without MRI needing to add a module, sort of where you just were to "simulate" that a boolean value exists. Again, I am not sure if this makes any sense what I write, but perhaps it would be better to wait some time, see how RBS shapes up, how the use cases may change, and then re-evaluate in say, two years or so. There are already quite a lot of changes if we look at the ruby-jit, ractor and so forth - it may be more clear how RBS may have to change (or effect change) in a little while.

#8 - 10/22/2020 12:49 PM - Dan0042 (Daniel DeLorme)

mame (Yusuke Endoh) wrote in #note-2:

BTW, soutaro (Soutaro Matsumoto) (the original author of RBS) is now thinking the redefinition of bool as an alias to true | false.

I think that's the better choice. Having bool as an alias to top is quite confusing. If we want to express that a method returns a truthy/falsy value, maybe conditional or cond would be a more meaningful alias for top.

#9 - 10/23/2020 08:53 PM - marcandre (Marc-Andre Lafortune)

matz (Yukihiro Matsumoto) wrote in #note-6:

Could you clarify the current proposal?

- bool / Bool / Boolean
- RBS side only / Ruby module as well

I am OK if RBS side only, no Ruby module.

Matz.

My preference is:

- Boolean, second choice Bool
- Ruby side also, or if deemed not preferable then RBS only (but what if there's user module/class Boolean?)
- If Ruby side, then base Class, second choice then Module.

Ruby side has advantages beyond RBS, especially for communication with other typed systems / data interchange.

I hope that, in RBS, String is strict, str is relax. Same for Boolean (strict true | false) and bool is relax (any object).

#10 - 10/26/2020 07:55 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Feedback

RBS side is up to soutaro (Soutaro Matsumoto) etc. I am still against Ruby side, marcandre (Marc-Andre Lafontune) stated about communication with other typed systems / data interchange as a benefit, but we are not going to make Ruby (type-wise) compatible with statically typed languages. So its benefit is still unclear.

Matz.
I'm planning to make the semantics of bool in RBS (in the next release). It will be an alias of true | false and we will add another type for conditionals, something like type boolish = top.

So, I feel there is no strong reason from RBS side to add new class/module to Ruby.