Ruby - Misc #19772

API naming for YARP compiler

07/17/2023 04:09 PM - jemmai (Jemma Issroff)

Status:	Closed	
Priority:	Normal	
Assignee:		

Description

We are working on the YARP compiler, and have the first PR ready which introduces the YARP compile method. Our only outstanding question before merging it is about naming. How should we expose the public API for YARP's compile method?

Potential suggestions:

1. YARP.compile
2. RubyVM::InstructionSequence.compile(yarp: true)
3. RubyVM::InstructionSequence.compile_yarp
4. Any of the above options, with a name other than yarp (please suggest an alternative)

Regarding option 1, which would mirror YARP.parse, is the top level constant YARP acceptable?

cc @matz (Yukihiro Matsumoto) @ko1 (Koichi Sasada)

History

#1 - 07/17/2023 04:09 PM - jemmai (Jemma Issroff)

- Subject changed from API Naming for YARP compiler to API naming for YARP compiler

#2 - 07/18/2023 10:12 AM - Eregon (Benoit Daloze)

FWIW I don't think renaming YARP at this point makes sense, now that the yarp gem is owned by @kddnewton (Kevin Newton) and after the detailed blog post.
Everyone calls it YARP, renaming now would feel silly.

If/when YARP would become the default then it would just be RubyVM::InstructionSequence.compile.
So I think option 2 is the best and makes it easy to experiment with a different default.

The implementation should be part of compile.c, so IMO it also makes sense the API is under RubyVM and not YARP.

#3 - 07/18/2023 10:19 AM - Eregon (Benoit Daloze)

And also option 1. seems very confusing, because it would imply the YARP module is always defined, but only has the compile method and not parse, etc until require "yarp".

#4 - 07/18/2023 06:47 PM - rubyFeedback (robert heiler)

I believe YARP.compile() is a bit of a strange name since it is so specific to YARP.

If possible I would perhaps recommend something like the other suggestion:

```
RubyVM::InstructionSequence.compile(yarp: true)
```

Or perhaps another name. Residing under the RubyVM namespace may make sense, so at the least I think this is better than YARP as primary name. You could even just pass in a symbol such as:

```
RubyVM::InstructionSequence.compile(:yarp)
```

(I do not know which alternatives may exist, so please keep in mind that I am solely commenting from an API point of view.)

The reason why I come to a similar conclusion as Eregon is mostly because I think this may seem cleaner and may allow for alternative implementations/models too. See the older syck gem, and psych, where ideally the user could just do:

```
require 'yaml'
```

But change the engine if necessary. I used that for a while, before I transitioned into psych completely some years ago. So on the API idea I concur with Eregon; I also think it may be useful if other ruby implementations integrate YARP eventually (jruby, natalie and so forth) or the feature/functionality, however it may be applicable (I am not very familiar with the jruby/truffleruby implementation but they try to stay as compatible as possible with MRI, with a slight implenentation delay normally).

#5 - 07/19/2023 07:12 AM - nobu (Nobuyoshi Nakada)

I don't think "YA-" names nice in general, unless for development code name.

As for YARP.compile, "a parser compiles code" sounds strange a little to me.
It might be better that YARP.parse will return AST and the AST has compile method to return ISeq.

#6 - 08/17/2023 05:21 PM - jemmai (Jemma Issroff)

nobu (Nobuyoshi Nakada) wrote in #note-5:

> I don't think "YA-" names nice in general, unless for development code name.

Agreed with you that "YA-" prefix is overloaded. Is your preference then something like YARP.parse.compile (or a different name)?

> As for YARP.compile, "a parser compiles code" sounds strange a little to me.
> It might be better that YARP.parse will return AST and the AST has compile method to return ISeq.

I have added this topic to the dev meeting agenda and hope we can resolve it there.

#7 - 08/21/2023 04:42 PM - jemmai (Jemma Issroff)

We have reflected on the name YARP and would like to propose Prism as a new name for the external API. The parser is similar to a prism because it takes Ruby code as input and returns it “refracted” into an AST. Also, the existing prism gem has not been updated since 2010 so we think we can ask for the name.

#8 - 08/22/2023 05:06 AM - ko1 (Koichi Sasada)

If the motivation of this API is testing YARP compilation to ISeq, I agree to introduce ISeq.compile_yarp to Ruby 3.3 and remove it from Ruby 3.4.
(Note that the name YARP is renaming one)

Because if YARP is introduced as the MRI parser, we don't need it (compile and compile_yarp are same).

#9 - 08/22/2023 01:38 PM - Eregon (Benoit Daloze)

I think YARP is already too widely used and talked about to be renamed without causing significant confusion and downsides.
In fact the gem already has multiple releases.

> I don't think "YA-" names nice in general, unless for development code name.

I don't see the problem with that.
The obvious names which could be better are parser and ruby_parser but both are already taken.

As a parallel, I think RubyVM should be YARV/CRuby, RubyVM confuses everyone, it looks like something general but it's actually YARV/CRuby-specific (at the very least RubyVM::InstructionSequence).

YARP::CallNode looks better than Prism::CallNode to me.

Many many Rubyists already know this project as YARP, renaming it now would IMO cause confusion and hurt.
The benefits of the rename seem way too small.

(also BTW there is already YJIT/--yjit, and it seems nobody minds that name)

#10 - 08/24/2023 10:52 AM - mame (Yusuke Endoh)

We discussed How to merge YARP to Ruby 3.3 at the dev meeting. Here is what @matz (Yukihiro Matsumoto) approved:

- MRI will use parse.y by default. Using an option (not concretely decided yet), it will use the built-in YARP.
- We introduce ::Ruby module.
  - This namespace is owned and predefined by the Ruby interpreter.
  - Any gem must not define any constants under the namespace.
- Built-in YARP can use used from Ruby in Ruby::Parser.
  - Ruby::Parser is not loaded by default. require 'ruby/parser' will define Ruby::Parser.

    - lib/ruby/parser.rb, etc. will be part of the standard library (it will not be part of default/bundled gems).
- Ruby::Parser should not be defined/modified by any gem (= gem cannot replace Ruby::Parser).
    - yarp/prism gem will use a different namespace than Ruby::Parser.
- We haven't decided whether to bundle the yarp/prism gem.
    - The name of the gem can be either yarp or prism.
    - yarp's implementation for multi-version support is up to yarp maintainers.

(Anyone who was at the meeting, please point out if I'm wrong)

#11 - 08/24/2023 10:54 AM - mame (Yusuke Endoh)

Supplemental information about "mutli-version support":

The parser gem supports multiple versions of Ruby grammars: Parser::Ruby30, Parser::Ruby31, Parser:Ruby32, Parser::CurrentRuby, ...
I guess the same functionality is required for the YARP gem if it aims to replace the parser gem. If so, what would be the API?
Ruby::Parser is supposed to not support multiple versions, but to be a wrapper for the version of YARP linked to the Ruby interpreter.

#12 - 08/24/2023 01:00 PM - Eregon (Benoit Daloze)

Why introduce Ruby::Parser as a different namespace if it does the same as YARP?
It doesn't seem a good idea to have 2 APIs for the same thing (memory overhead, defining every node class twice, etc) and in fact it is problematic on JRuby & TruffleRuby where YARP uses FFI and so 2 APIs would mean 2 librubyparser.so loaded in the same process which would cause symbol conflicts.
The solution to this is simple, single YARP version in a process by being a gem and single API: YARP.

And also Ruby::Parser is extremely confusing with existing RubyParser.

#13 - 08/24/2023 01:41 PM - kddnewton (Kevin Newton)

Thank you all for taking the time to discuss this issue at the meeting! I'm sorry I wasn't able to be there this time.

I have a couple of concerns about the naming, which I'd like to discuss further.

Going forward, the only people that will be using the Ruby API of the parser will be tool developers. (For example, formatters like Syntax Tree, linters like Rubocop, static analysis tools like Steep.) In order to provide a good experience for them, we will need to be able to provide bug fixes and features outside of the Ruby release cycle. As a couple of examples, we recently added IntegerNode#value, FloatNode#value, Location#join, etc. We also fixed some bugs in the Ruby API itself that doesn't impact the C API.

In order for developers to upgrade their version of the parser gem, they will need to be able to install the new version. Currently, when you install the latest version of the yarp gem, it comes earlier in the load path so it gets picked up before the bundled version. This works well.

If we were to name the Ruby API ::Ruby::Parser, but we weren't allowed to define any constants in a gem within ::Ruby, it would mean we would have to convert all of the constant references in the sync script, or define them all twice. I'm not sure how we would accomplish that. Furthermore, if a tool developer were to use ::Ruby::Parser and then wanted to upgrade to get the latest APIs/bug fixes, they would immediately have to switch all of their constant references over to the gem's version. If this ended up being the case, I think I would suggest just never using ::Ruby::Parser for tool developers because inevitably it would require that migration. So my only issue is the consistency in naming, really. If the constants in the built-in API do not match the constants in the gem API, the only way to get bug fixes and features will be to wait for the next version of Ruby.

I have a couple of options that I think would work going forward that I would like to suggest:

1. We could name the gem ruby-parser. The advantages are that the constants would match up and there would be no difficulty in using this API from a tool author's perspective. The disadvantages would be that the gem would be defining constants under ::Ruby and there already is a ruby_parser gem.
2. We could name the gem prism (or some other name). The advantages are the same as ruby-parser, but also that it does not start with the YA- prefix. The disadvantages are that prism is less clear than ruby-parser in what it does. (The original idea was the prism relates to optics, in that we are looking at Ruby source code and reflecting it in different ways through Ruby, C, and serialization APIs.)
3. We could keep the name yarp. The only advantage is that nothing changes. The disadvantage is that no one knows what yarp actually means.

I am fine with any of these options, and am certainly open to more ideas if there are some! In summary the main thing I would like to keep in mind is that the Ruby API will be used by tool developers, and we should make it as easy as possible for them to maintain the tools. Thank you for the consideration!

#14 - 08/24/2023 02:12 PM - Eregon (Benoit Daloze)

> In summary the main thing I would like to keep in mind is that the Ruby API will be used by tool developers, and we should make it as easy as possible for them to maintain the tools.

Exactly. I believe for this reason the best and simplest would be:

- Only copy YARP C files to CRuby (using the sync script), to be used (optionally) as the parser for CRuby. Then it's clear, the part of YARP in CRuby core is just for parsing Ruby code to compile to bytecode, not a Ruby API.
- Add yarp as a bundled gem, so it's already there when installing Ruby. It's just a gem and if tools need a newer version, it just works.

That would avoid any confusion about where the Ruby API of YARP is and the various implications, as discussed extensively recently on CRuby Slack.
It also solves the symbol conflicts issue I mentioned in my previous comment.

Longer-term there is the question of using YARP to implement Ripper.
I think that could just require "yarp" behind the scenes and so use the bundled gem.

#15 - 08/24/2023 02:21 PM - kddnewton (Kevin Newton)

I can also discuss multi-version support.

Starting in December, we will release YARP version 3.3. In this version, the C parser itself will be frozen so that developers can always have access to the same behavior as Ruby version 3.3. The Ruby API, however, will still be able to be changed in order for developers to get new features (like I mentioned in the previous comment).

At this point after the release, we'll begin developing on the 3.4 branch. If consumers of the gem want to parse code with 3.3's behavior, they will be able to use the old version of the gem. We will have the same maintenance cycle as CRuby does.

#16 - 08/24/2023 05:43 PM - k0kubun (Takashi Kokubun)

> Going forward, the only people that will be using the Ruby API of the parser will be tool developers. (For example, formatters like Syntax Tree, linters like Rubocop, static analysis tools like Steep.)

Template engines like Haml that parse Ruby code for optimization purposes, which are executed in production code and probably not considered a "tool". Do you think these gems should keep using Ripper forever?

> At this point after the release, we'll begin developing on the 3.4 branch. If consumers of the gem want to parse code with 3.3's behavior, they will be able to use the old version of the gem. We will have the same maintenance cycle as CRuby does.

I think it's problematic for some use cases that you cannot use a parser for the running Ruby version when you have YARP for an old Ruby version in Gemfile. For example,

- You're parsing Ruby 3.3 code with some tool (e.g. Rubocop) that is running on Ruby 3.4. When you parse Ruby 3.3 code, the parser should be a parser for Ruby 3.3. But when you debug the process with binding.irb or debug.gem, the parser used by them should be a parser for Ruby 3.4 because the code runs on Ruby 3.4.
- Similarly, the Haml template engine should always use a Ruby 3.4 parser for parsing Ruby code for optimizing templates even if Gemfile specifies YARP for an old Ruby version for Rubocop or because you forgot bundle update after upgrading Ruby.

Essentially, it means that you shouldn't use YARP for parsing Ruby code that is going to be eval-ed in the same process because the syntax that yarp.gem parses is not guaranteed to be the same as what eval parses. parser.gem's Parser::CurrentRuby supports this, Ripper supports this, Matz's Ruby::Parser idea supports this, but this yarp.gem design doesn't.

If we ship the current yarp.gem as is and don't ship Ruby::Parser, I will keep using Ripper for those use cases because Ripper is the only Ruby API that is guaranteed to be compatible with the current Ruby version.

I can think of the following options that can support the use cases of parsing eval-ed Ruby code.

1. [current] Keep maintaining Ripper as the only Ruby API that is guaranteed to wrap the parser used by the interpreter.
2. [matz's idea] Provide Ruby::Parser that wraps the parser used by the interpreter with YARP-like API (but with different constant names).
3. yarp.gem is shipped with a parser for a single Ruby version, and required_ruby_version will always allow only that Ruby (minor) version.
4. yarp.gem is shipped with a parser for a single Ruby version that could be old, but also has another class that wraps YARP's C API the interpreter provides, which can be used like parser.gem's Parser::CurrentRuby.
    - C code and Ruby code would be provided by different YARP versions, so I'm not sure if it's practical.
5. yarp.gem is shipped with parsers for multiple Ruby versions, and has a class that selects a parser for the current Ruby version, which works like parser.gem's Parser::CurrentRuby.

#17 - 08/25/2023 12:37 PM - kddnewton (Kevin Newton)

Thank you @k0kubun (Takashi Kokubun), I didn't fully understand that requirement until you wrote it out here.

I would like to suggest we go with a combination of #2 and #4. I don't think we need to duplicate all of the constants. I think we could use the same parse function that the current runtime uses (like matz's idea) but use the same code we use in YARP to build the Ruby objects. However that function is invisible to the YARP C extension, so it would need to be implemented by the runtime (which we can do since all of the runtimes have integrated it).

I would like to do this because it makes it so that we can solve your requirement (that the code is parsed exactly as the current runtime is parsing it) but it also allows us to provide upgrades and bugfixes to the Ruby API of YARP.

I think it's fine that the C and Ruby code would be provided by different YARP versions. We wouldn't want to introduce any breaking changes outside of a major version of Ruby anyway because we would want tool authors to get a consistent experience. (I have been calling everything that calls

Parser.parse a tool, maybe I need a better term...) There are a couple of technical hurdles we will need to overcome if we introduce additional fields between versions, but I believe these will be easily overcome.

#18 - 08/25/2023 01:31 PM - Eregon (Benoit Daloze)

@kddnewton (Kevin Newton)

> There are a couple of technical hurdles we will need to overcome if we introduce additional fields between versions, but I believe these will be easily overcome.

How? That seems near impossible to me, without having 2 versions of the C code (*.h and *.c and api_node.c) & Ruby code too.
If there is a single api_node.c it would use the struct definitions in the gem for the result of yp_parse in interpreter, which would break as soon as there is any field change in any of the node structs.
E.g. if there is a new field in the yarp gem then it would read out of bounds.

And if we think about serialization I think it's more obvious it cannot work without duplicating all files related to serialization & deserialization and their dependencies (so basically every file of YARP), since it would be different YARP::VERSION and the check when deserializing would (correctly) fail.

Here is how I see those 5 solutions:

1. not really a solution because the Ripper API is very inconvenient to use and longer-term Ripper might use YARP, and/or be deprecated.
2. this requires quite a lot of work and magic to adapt the namespace. It feels rather messy, but probably feasible. Obviously there is the overhead of defining every Ruby node class twice, etc.
3. this is the simplest solution. It might be the best because of that.
4. I think this cannot work. It would need to duplicate every file but also if you install an older YARP gem how would it find the {.c,.h,.rb} files for current Ruby? It seems impossible.
5. this is the most generic solution and would allow to fully replace the parser gem, and make adoption much easier for any tool using the parser gem currently. It means more if in the codebase but OTOH a single branch to maintain.

#19 - 08/25/2023 02:01 PM - kddnewton (Kevin Newton)

We already have two copies of the C code. We have the vendored one that is copied into CRuby/JRuby/TruffleRuby, and we have the one that is inside the gem. The symbols are purposefully hidden so that they can be matched up with the correct caller. The struct definitions would be fine, because they would each belong to their respective file sets.

When translating into Ruby or serializing, we could handle it any number of ways. We could introduce new node types whenever we add a field, which would deserialize into the previous node types but with nil values. We could add explicit markers saying "new field here, skip on previous versions". There are plenty of ways around this.

#20 - 08/25/2023 03:30 PM - Eregon (Benoit Daloze)

kddnewton (Kevin Newton) wrote in #note-19:

> We already have two copies of the C code. We have the vendored one that is copied into CRuby/JRuby/TruffleRuby, and we have the one that is inside the gem. The symbols are purposefully hidden so that they can be matched up with the correct caller. The struct definitions would be fine, because they would each belong to their respective file sets.

Right, but then we need to fully use that copy and that also means we need a copy of api_node.c and extension.c (otherwise it's mixing the node structs).
And on JRuby/TruffleRuby the C extension is not used, so then we need a copy of Ruby files too (the Ruby files contain the logic to deserialize).
Even on CRuby we would need a copy of Ruby files because of different node names/fields/visitor methods/etc between versions.
And that implies a different namespace to avoid redefinition warnings.
So basically that's option 2 and 2 full independent copies of YARP files (one in the Ruby impl, one in the gem), with different namespaces (the name of the second namespace could be something else than Ruby::Parser of course).

> When translating into Ruby or serializing, we could handle it any number of ways. We could introduce new node types whenever we add a field, which would deserialize into the previous node types but with nil values. We could add explicit markers saying "new field here, skip on previous versions". There are plenty of ways around this.

I don't think any of these would work well in practice. Mixing files of two YARP versions is a recipe for segfaults and complicated hard-to-debug errors.
And the yarp gem being used might be newer or older, e.g., we cannot assume a node type number is free.
The serialization and deserialization need to match exactly (the version check there is not optional, it crashes without), and the same for C structs and api_node.c.

#21 - 08/29/2023 04:41 AM - mame (Yusuke Endoh)

kddnewton (Kevin Newton) wrote in #note-13:

> Going forward, the only people that will be using the Ruby API of the parser will be tool developers. (For example, formatters like Syntax Tree, linters like Rubocop, static analysis tools like Steep.) In order to provide a good experience for them, we will need to be able to provide bug fixes and features outside of the Ruby release cycle.

What changes are you thinking of providing outside of the Ruby release cycle?

If it is a critical bug fix that involves parsing results, we need to cut a patch release of Ruby in any way. For other fixes and improvements, there is no need to rush, so a patch release (or minor release for new features) is still acceptable, I think.
It is great to provide a good experience for tool developers, but we should keep in mind that it is a trade-off for stability.

I think the current discussion is likely to provide a built-in Ruby API like Ruby::Parser. But any changes to it should be under the Ruby release cycle.
In other words, in principle, bug fixes should be done in patch releases, and improvements and new features should be done in minor releases.
In fact, the ruby_3_3 branch is only writable by the branch maintainers, and changes are made by requesting backports from the branch maintainers.
It would be difficult to change this system for YARP.

I think the following is simplest for Ruby developers (except YARP developers) and Ruby users:

- yarp.gem supports multi-version as the parser gem does:YARP::Ruby33.parse, YARP::Ruby34.parse, YARP::Ruby35.parse, ... and YARP::CurrentRuby.parse.
- Ruby 3.3 links with only the C implementation part of YARP::Ruby33. Ruby itself does not provide a Ruby API like Ruby::Parser.

While this is more work for the YARP developer, it has the advantage of eliminating the need to go through backports to change and improve the Ruby API.

#22 - 08/29/2023 04:29 PM - kddnewton (Kevin Newton)

From this thread and others, this is my understanding of the requirements:

- The YARP name is not good, since we don't want YA- prefixes in production code.
- We could like to be able to improve the Ruby API without having to go through the Ruby release cycle.
- We would like for users to be able to parse Ruby code as various versions of Ruby have previously parsed Ruby code.
- We would like for users to be able to parse Ruby code exactly as the current version of Ruby is parsing Ruby code.

Here are the kinds of changes we would like to be able to make outside of the Ruby release cycle:

- We would like to be able to add fields to the various nodes.
- We would like to be able to add flags to the various nodes.
- We would like to be able to change the Ruby API to provide additional functionality/classes.

Here are the steps I would like to take to address these requirements:

1. I would like to rename the YARP gem to Prism.
2. I would like to change the way we build Ruby objects from both the C API and the serialization API. Currently when we build Ruby objects from these APIs, we call rb_class_new_instance directly with the fields coming back from the parser. Instead, I will change this to call a Ruby method (either XXXNode.new or YARP.build_xxx_node) that will instantiate those objects.
3. When we add fields to nodes, we will patch the factory method to handle the various combinations coming back from the various APIs. Those factory methods will check the target Ruby version and handle the fields appropriately for that version of Ruby.
4. When we add flags to nodes or change the Ruby API to provide additional functionality, nothing should have to change.
5. I would like to add YARP.parse_native which will take advantage of the previous bullets here to specify that the target Ruby version should be exactly the version of Ruby that is currently running.

With these changes in place:

- We will have a better name.
- We will be able to upgrade the Ruby API in non-breaking ways without going through a Ruby release cycle.
- Users will be able to install one version of the gem and be able to parse as previous versions of Ruby did.
- Users will be able to install one version of the gem and be able to parse as the current version of Ruby does.
- We will not be able to make breaking changes to the C parser. (We don't want to do this outside the Ruby release cycle anyway.)

Does this approach sound good to everyone, or are there any other requirements that I have missed?

#23 - 08/29/2023 08:15 PM - Eregon (Benoit Daloze)

> Does this approach sound good to everyone, or are there any other requirements that I have missed?

These may not be requirements but they seem very important considerations for YARP adoption & usability:

- If inside a single application with a Gemfile, one wants to parse with two different non-current Ruby grammar versions, that is not possible with YARP, but it is possible with the parser gem (given enough tooling gems in a Gemfile this does not seem unlikely, suppose RuboCop parsing 3.2 and some lint tool using 3.1 because it doesn't handle new syntax/nodes from 3.2 yet; another example, testing with ruby-head + RuboCop for 3.1 and ruby-lsp/solargraph for 3.2 in Gemfile).
- To make RuboCop fast I think it is well understood we need it to use YARP and that most likely by having the parser gem use YARP. But if YARP can only parse current Ruby + one specific version then it's going to be rather unlikely the YARP version will match the RuboCop TargetRubyVersion. Having to specify gem "yarp", "~> 3.2.0" seems very inconvenient, I would think most Rubyists would not understand this requirement (e.g. why is it suddenly slow after updating yarp in Gemfile.lock?) or find it excessively annoying (if e.g. parser gem warns about it

> and requires to change the Gemfile manually, and yet another place to hardcode the Ruby version, plus it would work poorly for gems supporting multiple Ruby versions).

The solution to both of these is to support multiple Ruby grammar versions in a single gem release, like the parser gem.
Then it is always safe to update YARP, and Ruby interpreters just ship with one version of YARP as a bundled gem (+ the C code for the parser used for the interpreter).
My feeling is this actually not more work than 3 release branches and some way to have both some version + current Ruby version working together.
What are your concerns with supporting multiple Ruby versions inside one YARP version?
And what is your vision for RuboCop & parser gem to use YARP?

Regarding having both some version + current Ruby version working together:

- About adding fields, that may work for Ruby nodes but will not and cannot work for adding/removing/renaming/reordering fields in C structs or serializing more/less fields or differently (e.g. suppose there is some new attribute on DefNode, or some new IntegerNode int32_value field, or some extra location information desired, impossible to add that without changing C structs & serialization, or serialization uses a different more optimized format). As I said I before, I believe it cannot possibly work to have e.g. the yarp 3.5.0 gem installed in CRuby 3.3.0 and hope that the serialization will be compatible between both (and vice versa). It will break at the first field or node change between these 2 versions.
- The only way to support current + another version is two full copies of all files (or IOW the version of yarp "integrated with the interpreter" is just completely separate from the yarp gem version, they would share nothing), and different namespaces, as I wrote in https://bugs.ruby-lang.org/issues/19772#note-20. That feels a bit hacky and less convenient for users but it would work.
  It has the advantage to really use the same parser for the Ruby API as the interpreter uses. But also if there is an issue with the Ruby API for "current version" (e.g. a segfault in the lexer for some weird code, or some incorrect location which leads to poor debugging or REPL experience) it cannot be fixed except with a CRuby patch release (i.e. installing a newer yarp gem has no effect on the current version parser namespace, called Ruby::Parser above). EDIT: And worse than that, it might affect other Ruby implementations too and they have different release cycles so in the worst case Ruby::Parser might be quite difficult to use until all Ruby implementations had a release including the yarp fix, which can take many months.

  > Those factory methods will check the target Ruby version and handle the fields appropriately for that version of Ruby.

That sounds rather expensive and slow, and it will be an overhead on every YARP.parse. Extra method calls and indirections have a non-trivial overhead, at least on CRuby.
Plus having to maintain that manually sounds quite error-prone and verbose, and difficult to test properly.

> The YARP name is not good, since we don't want YA- prefixes in production code.

I like YARP and I think it should stay since it's already established.
I see YARP as just an abbreviation (I pronounce it "YARP" not "Yet Another Ruby Parser", same for YARV, YJIT, etc).
Similarly, whenever I read CSV in code I don't think "Comma-Separated Values", it's just "CSV".
And Yet Another is really a good fit here, we had so many parser projects and attempts, I think this is the good one to unify all.

#24 - 08/29/2023 08:22 PM - Eregon (Benoit Daloze)

Also in some cases the Ruby version to parse might not be known early or be fixed.
Imagine some gems documentation website. It needs to generate docs using rdoc or yard, and that may include snippets of Ruby code, which if there is any syntax incompatibility between Ruby versions should be parsed with a version the gem is compatible with. And that information might be in the gemspec or CI files or based on the published date, etc, it would be quite messy if one has to dynamically generate a Gemfile based on that version and run another process vs just using the latest release of YARP and being able to specify the version dynamically.

#25 - 08/30/2023 08:42 AM - hsbt (Hiroshi SHIBATA)

This issue mixed many of topics. It's not for "YARP compiler".

For Ruby core

- Integration strategy of Ruby eval-ed parser.
- How expose eval-ed parser for users.
- And it's Naming

For standalone gem

- How bundle yarp as default gems or bundled gems.
- Release cycle
- Multi-version support
- etc...

@kddnewton (Kevin Newton) @jemmai (Jemma Issroff) Can you categorize and summarize them to each new issue?

#26 - 09/12/2023 08:37 PM - mame (Yusuke Endoh)

Hi @kddnewton (Kevin Newton)

kddnewton (Kevin Newton) wrote in #note-22:

> From this thread and others, this is my understanding of the requirements:
>
> - The YARP name is not good, since we don't want YA- prefixes in production code.

There may be a communication error. More precisely,

- The YARP name is not good as a built-in name, e.g., built-in constant names (like String).
- The YARP name is acceptable as a gem name.

If the Ruby API of YARP is provided as a gem, there is no need to rename it.

> - We could like to be able to improve the Ruby API without having to go through the Ruby release cycle.

Whose requirement is this? I wasn't sure who was asking for it. This requirement is new to Ruby and makes the discussion complicated. If YARP is a built-in feature, I recommend that you drop this requirement. If YARP is a gem, you are free of course.

> - We would like for users to be able to parse Ruby code as various versions of Ruby have previously parsed Ruby code.
> - We would like for users to be able to parse Ruby code exactly as the current version of Ruby is parsing Ruby code.

As @Eregon (Benoit Daloze) says, I think that two different non-current Ruby grammer versions should be available at the same time. The parser gem allows that. I am concerned that if there is something that the parser gem can do but YARP cannot, then the original goal of making YARP a universal parser might not be achieved.

> Here are the kinds of changes we would like to be able to make outside of the Ruby release cycle:
>
> - We would like to be able to add fields to the various nodes.
> - We would like to be able to add flags to the various nodes.

Sorry, what are "fields" and "flags" here?

> - We would like to be able to change the Ruby API to provide additional functionality/classes.

Note that, if YARP is a built-in feature, all these changes will require Matz's approval. With that in mind, I would never recommend doing it outside of the Ruby release cycle. If YARP is a gem, you can do it freely.

> Here are the steps I would like to take to address these requirements:
>
> 1. I would like to rename the YARP gem to Prism.

You may change it if you wish, but

- As a built-in name, neither YARP nor Prism is allowed. The only name accepted for a built-in name so far is Ruby::Parser.
- As a gem name, both YARP and Prism are okay. (However, Ruby::Parser is not a suitable name for a gem.)

So I think it will solve nothing.

> 1. I would like to change the way we build Ruby objects from both the C API and the serialization API. Currently when we build Ruby objects from these APIs, we call rb_class_new_instance directly with the fields coming back from the parser. Instead, I will change this to call a Ruby method (either XXXNode.new or YARP.build_xxx_node) that will instantiate those objects.
> 2. When we add fields to nodes, we will patch the factory method to handle the various combinations coming back from the various APIs. Those factory methods will check the target Ruby version and handle the fields appropriately for that version of Ruby.
> 3. When we add flags to nodes or change the Ruby API to provide additional functionality, nothing should have to change.
> 4. I would like to add YARP.parse_native which will take advantage of the previous bullets here to specify that the target Ruby version should be exactly the version of Ruby that is currently running.

This seems interesting. Let me confirm. I understood it as follows:

- We introduce, as a built-in feature, only one method to invoke the YARP parser linked into Ruby. Let's call it RubyVM.invoke_yarp(src, handler) as a tentative name.
- RubyVM.invoke_yarp parses the src string and calls handler.build_xxx_node(*child_nodes) when creating a node.

For example, RubyVM.invoke_yarp("1 + x", handler) calls

1. handler.build_integer_node("1") (To be precise, it should pass a code location rather than a code fragment)
2. handler.build_local_var_node("x")
3. handler.build_call_node(:+, (the result of 1), (the result of 2))

and finally returns the result of 3. Everything else (such as node class definitions, helper methods, etc.) is placed in the YARP gem.

YARP.parse_native calls RubyVM.invoke_yarp.

Is my understanding correct? It's similar to Ripper's SAX API. If this is correct, it would be good if you could focus your proposal on the RubyVM.invoke_yarp spec.

#27 - 09/12/2023 08:51 PM - mame (Yusuke Endoh)

I would add that if the yarp gem had straight multi-version support like the parser gem, such a mechanism is not even needed.

#28 - 09/13/2023 09:00 AM - Eregon (Benoit Daloze)

@mame (Yusuke Endoh) Thanks for the clarifications, notably about the name.

My impression is if YARP supports multiple Ruby grammar versions then basically everything discussed in this issue is solved (and no need to rename, and only C files of yarp are copied in CRuby + available as a bundled gem).

Regarding that handler, where would it be defined?
It cannot be vendored in CRuby, otherwise e.g. if IntegerNode has a new field (in a newer yarp gem) it would call the IntegerNode constructor with not enough arguments.
If it is in the yarp gem, and IntegerNode has a new field (in newer yarp gem) then build_integer_node needs e.g. 2 vs 1 arguments. So positional arguments cannot work.
It could be keyword arguments, maybe but then it becomes extremely expensive to call those methods from a C extension.
That overhead seems big enough to hurt YARP adoption, so I think this is not a good way.
I think multi-version support is the best solution here.

> I am concerned that if there is something that the parser gem can do but YARP cannot, then the original goal of making YARP a universal parser might not be achieved.

Exactly.

#29 - 09/21/2023 01:43 AM - mame (Yusuke Endoh)

I had a little conversation with @matz (Yukihiro Matsumoto) about this issue yesterday.

He clearly rejects the name "yarp" not only a built-in constant but also as a bundled gem. He said "prism" is fine for a bundled gem. (Sorry, I misunderstood his opinion of this.)

He came up with an idea to enable a user to use multi-version yarp gems at a time by Namespace on read (#19744) in the future. So he said that the yarp gem doesn't have to support multi-version.

```
ns33 = Namespace.new
ns33.require("yarp", version: "3.3.0")
ns33::YARP.parse(src) # parse it as Ruby 3.3 syntax

ns34 = Namespace.new
ns34.require("yarp", version: "3.4.0")
ns34::YARP.parse(src) # parse it as Ruby 3.4 syntax
```

In light of the above, here is a suggestion.

- How about having the ruby repository mirror only the C implementation of YARP, but not Ruby API? It should be used only when ruby --enable-yarp or something are given.
- How about bundling the "prism" gem as a bundled gem to provide a Ruby API for YARP?

@matz (Yukihiro Matsumoto) Please point out if I misunderstood something.

@kddnewton (Kevin Newton) What do you think?

#30 - 09/21/2023 08:05 AM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote in #note-29:

> He came up with an idea to enable a user to use multi-version yarp gems at a time by Namespace on read (#19744) in the future. So he said that the yarp gem doesn't have to support multi-version.

That's an interesting way to solve it but this seems clearly not for the near future (I think that is several years):

- This feature is not merged yet, it might be difficult to implement reliably (there are so many shenanigans with symbol loaders/linkers/etc and inconsistencies between platforms, if the approach works on one platform it might not on another platform)
- It might be even more difficult to implement on some other Rubies like JRuby and TruffleRuby. Both of these use the FFI gem and not the C extension for yarp, and I'm really not sure it's possible to load 2+ librubyparser in the same process with FFI without everything conflicting or segfaulting (and even if it happens to work for 2 versions on some platform, how to be confident it will be the case for other versions/platforms?). Until that is figured out I think we should assume this approach to be non-viable for other Rubies than CRuby which I think should be a blocker.

- This would require extensive changes in RubyGems and Bundler, which might take a while as well and might be difficult for compatibility reasons. I'm not sure many Rubyists want a Bundler that feels like npm/node_modules with every dependency duplicated N times (+ of course the longer bundle times, etc). We should probably ask the opinion of RubyGems & Bundler maintainers in #19744.

Also, until all the above is done, this doesn't solve that e.g. the parser gem couldn't rely on yarp and add it as a gem dependency then, and the same for all other tools which want to use yarp but don't want to force a given version (and if they do they will then be incompatible with other tools).

So I think this solution is unrealistic for the near future.

OTOH there seems to be a lot of work on yarp to also worry about multi-versions support now.
So maybe multi-versions support in yarp could be added later, e.g. after the CRuby 3.3 release.
Until then YARP can only parse the "Ruby 3.3" dialect which seems an OK limitation for the time being.

> In light of the above, here is a suggestion.
>
> - How about having the ruby repository mirror only the C implementation of YARP, but not Ruby API? It should be used only when ruby --enable-yarp or something are given.
> - How about bundling the "prism" gem as a bundled gem to provide a Ruby API for YARP?

I think this is good and should be done anyway.
It is also easier to support YARP like that in JRuby & TruffleRuby, as the YARP Ruby API is then "just a normal gem" and can be updated with gem install etc.
In fact JRuby & TruffleRuby are both integrating YARP and at least so far they assumed that yarp would be a bundled gem, only C+Java files (so not the Ruby API) are imported in JRuby & TruffleRuby repositories.

#31 - 09/22/2023 04:07 PM - kddnewton (Kevin Newton)

This issue has had a lot of discussion on a lot of different topics. I'm going to close it for now since the original issue that Jemma was talking about has long since been resolved. I will open other tickets for multi-version support and naming going forward.

#32 - 09/22/2023 04:07 PM - kddnewton (Kevin Newton)

- Status changed from Open to Closed