Topaz Documentation

Release 0.1

Alex Gaynor

Apr 06, 2017

Contents

Topaz is a high performance implementation of the Ruby programming language, written in Python on top of RPython (the toolchain that powers PyPy).

To get started right away, you can download a pre-built binary and run it:

```
$ ./bin/topaz -e "puts 'Hello from Topaz'"
Hello from Topaz
```

CHAPTER 1

Getting Started

Welcome to Topaz! There are two places to get started *Using Topaz* and *Building Topaz*.

Using Topaz

To get started with Topaz, you can download a binary or *build topaz yourself*. Once you've got a `topaz` binary you can run it directly, just like you would any other Ruby:

```
$ ./bin/topaz -e "puts 'hello world'"
hello world
$ echo "puts 'hello world'" >> test.rb
$ ./bin/topaz test.rb
hello world
```

Keep in mind that Topaz is not finished yet, and you may run into bugs or missing features. If you do please *report them*!

Building Topaz

Before you build Topaz, there's a few things you'll need:

- A checkout of the topaz repository: `git clone http://github.com/topazproject/topaz`
- A recent checkout of the PyPy repository: `hg clone https://bitbucket.org/pypy/pypy`
- The libffi development files: e.g. on Debian install with `sudo apt-get install libffi-dev`
- Other dependencies: `pip install -r requirements.txt`

We recommend installing `PyPy` and other dependencies into a virtualenv.

Once everything is setup (make sure `rpython` is on your `PYTHONPATH`), you can compile Topaz:

```
$ python path/to/pypy/rpython/bin/rpython -Ojit targettopaz.py
```

Wait a bit (you'll see fractals printing, and some progress indicators). On a recent machine it'll take about ten minutes. And then you'll have a `topaz` binary in `bin/`.

You can also run Topaz without compiling, on top of Python:

```
$ python -mtopaz -e "puts 'hello world'"
Hello world
```

Note that this is extremely slow, and should never be used for benchmarking, only for testing.

Alternately, you can build `topaz` using `ruby-build`:

```
$ git clone git://github.com/sstephenson/ruby-build.git
$ cd ruby-build
$ ./bin/ruby-build topaz-dev /path/to/install/topaz
```

If you run `homebrew` on OS X, it's even easier:

```
$ brew update && brew install ruby-build
$ ruby-build topaz-dev /path/to/install/topaz
```

You can also install the latest nightly build of `topaz` using `ruby-build` as a plugin to `rbenv`:

```
$ brew update && brew install rbenv ruby-build
$ rbenv install topaz-dev
```

CHAPTER 2

Current Status

Topaz implements Ruby 1.9.3.

At present, Topaz is not considered stable or tested in the real world, and is extremely incomplete. We do not yet consider Topaz to be production ready (but it gets closer every day!). All that said, if Topaz does run your program correctly, it is likely to continue to work.

Topaz is untested on Windows (but we'd love to improve this!), it is tested on OS X and Linux.

The following is a list of known missing features:

- Most of the standard library is missing.
- No threading support.
- No `call/cc`.
- No method visibility (`private`).
- Many builtin methods are missing.
- No flip-flop support.
- No support for `if /regex/`.

Remember, *you can help*!

CHAPTER 3

How to contribute

First, thanks for your interest in contributing to Topaz! Contributors like you are what make it awesome.

There are a number of different ways you can contribute to Topaz.

Testing your software

Right now we know that Topaz doesn't implement all of Ruby, however that's our goal. If you try out your code with Topaz and it doesn't work, let us know. The more people talk to us about a given feature, the more we'll prioritize it.

Filing bugs

If anything doesn't work right with Topaz, whether it's a segfault or a typo in an error message, please let us know. We can't fix the bugs we don't know about! You can file a bug on our Github repository, try to provide all the information someone will need in order to reproduce your bug. When filing a bug, make sure to include the version of Topaz you were testing with (`topaz -v` will show you it).

Benchmarking

We're committed to making Topaz the fastest Ruby implementation. If you benchmark your code and it's slower on Topaz than any other Ruby implementation, let us know. We take performance seriously.

Writing a patch

We welcome patches of all sorts to Topaz, whether it's to the docs or the code. You can send us patches by forking our repository on Github and then sending a pull request.

Getting a copy of the repository

First things first, you'll need to grab a copy of the repository:

```
$ git clone git://github.com/topazproject/topaz.git
```

Running the tests

One thing you should know when writing a patch for Topaz, is that all changes need tests (or a really good reason why they don't). You should first check whether you can find a Rubyspec that previously failed and now passes with your patch. If you do, see below for how to untag it. If there is no Rubyspec that now works, you need to write a test for our test suite. You can run our test suite by installing `py.test` (`pip install -r requirements.txt`):

```
$ py.test
```

This will run all the tests. In general you do not need to compile Topaz when working on a patch, all changes should be testable directly, and the buildbot will verify for every pull request that it compiles and tests pass.

Running Rubyspecs

To run Rubyspecs, you can use the provided `invoke` tasks. To get `invoke` you must have Invoke installed. The rubyspec and mspec repositories have to be checked out next to your topaz repository, the spec tasks will clone them for you if they aren't already there.

To just run all specs that should pass:

```
$ invoke specs.run
```

You can also pass additional options, or run just a subset of the specs:

```
$ invoke specs.run --options="-V --format dotted" --files=../rubyspec/core/array
```

If you encounter failures that you need to tag:

```
$ invoke specs.tag --files=../rubyspec/path/to/failing_spec.rb
```

Not that you cannot tag specs that fail or error during load or setup, to skip those you have to add them to the list of skipped specs in `topaz.mspec`.

If you implemented a new feature, and want to untag the specs that now pass:

```
$ invoke specs.untag --files=../rubyspec/path/to/failing_spec.rb
```

And finally, during development, you may find it useful to run the specs untranslated:

```
$ invoke specs.run --untranslated --files=../rubyspec/core/array/new_spec.rb
```

Adding yourself to the authors file

When you submit your first patch, add your name to the `AUTHORS.rst` file, you've earned it!

CHAPTER 4

Style Guide

Python

Python code should follow PEP 8, lines should be limited to 79 columns.

Ruby

Indentation

Ruby code should be indented with 2 spaces (never hard tabs). Lines should be limited to 79 columns. There should never be trailing white space.

Method Definitions

Ruby method definitions should include parenthesis if there any arguments, and no parenthesis if there aren't arguments.

Good:

```
def f
end

def f(a, b)
end
```

Bad:

```
def f()
end
```

```
def f a, b
end
```

There should be spaces around the = for default arguments.

Good:

```
def f(a = nil)
end
```

Bad:

```
def f(a=nil)
end
```

Operators

Spaces should always be used around operators.

Good:

```
2 ** 31 - 1
```

Bad:

```
2**31-1
```

Blocks

Spaces should be used around the pipes and braces in blocks.

Good:

```
arr.map { |x| x * 2 }
```

Bad:

```
arr.map {|x|x * 2}
```

When testing for a block, prefer explicit `if block` to `block_given?`.

Good:

```
def f(&block)
  if block
  end
end
```

Bad:

```
def f
  if block_given?
  end
end
```

Hashes and Arrays

There should be no spaces around either brackets or braces, spaces should always follow commas and go around hash rockets. Hash rockets should be used in preference to "new-style" hashes.

Good:

```
[1, 2, 3]
{:abc => 45}
```

Bad:

```
[1,2]
{ :abc=>23 }
{abc: 23}
```

Exceptions

Exceptions should be raised using `ExceptionClass.new`, rather than the 2-argument form of `raise`. Error messages should be compatible with CRuby whenever reasonable.

Good:

```
raise ArgumentError.new("A message")
```

Bad:

```
raise ArgumentError, "A message"
```

Statements

Never use `and`, `or`, or `not`, their precedence is confusing, prefer `&&`, `||`, and `!`.

The ternary operator should only be used for selecting a value, never for a side effect.

Good:

```
(a > b) ? a : b
```

Bad:

```
foo ? self.bar! : nil
```

CHAPTER 5

Source Code Map

This document is a map of the source code, it describes what many of the directories/files contain.

ast.py

This contains each of the AST node classes. Each of these is responsible for encoding the structure of the program, and compiling itself to bytecode.

astcompiler.py

This contains utility classes for compiling ASTs to bytecode.

coerce.py

This contains logic for performing type coercion on Ruby objects. It contains implementations for behaviors like, "this Ruby function takes an Integer argument".

executioncontext.py

This contains logic specific to the current thread of execution in Ruby. It maintains things like the Ruby call stack.

frame.py

This contains Ruby frame objects, and associated logic for manipulating them.

gateway.py

This contains logic for exposing RPython functions in Ruby.

interpreter.py

This contains the bytecode interpreter itself.

lexer.py

This contains the hand written Ruby lexer.

main.py

This contains the command line entry-point, including things like argument parsing logic.

module.py

This contains an API for exposing RPython classes in Ruby, it should be merged with `gateway.py`.

objspace.py

This contains the `ObjectSpace` (unrelated to Ruby's `ObjectSpace` module), it is responsible for encoding the behaviors of Ruby, for example it contains methods like `find_const`, `send`, and methods for creating new Ruby objects from RPython ones.

parser.py

This contains the parse rules and actions, built atop `rply`.

modules/

This contains built-in Ruby `Module` objects. There is one `Module` per file.

objects/

This contains built-in Ruby `Class` objects. There is one `Class` per file.

CHAPTER 6

Community

You can find Topaz all over the web:

- Mailing list
- Source code
- Issue tracker
- Documentation
- IRC: `#topaz` on `irc.freenode.net` (logs)
- @topazproject on Twitter

CHAPTER 7

Code of Conduct

Like the technical community as a whole, the Topaz team and community is made up of a mixture of professionals and volunteers from all over the world, working on every aspect of the mission - including mentorship, teaching and connecting people.

Diversity is one of our huge strengths, but it can also lead to communication issues and unhappiness. To that end, we have a few ground rules that we ask people to adhere to when they're participating within this community and project. These rules apply equally to founders, mentors and those seeking help and guidance.

This isn't an exhaustive list of things that you can't do. Rather, take it in the spirit in which it's intended - a guide to make it easier to enrich all of us and the technical communities in which we participate.

This code of conduct applies to all communication: this includes IRC, the mailing list, the issue tracker, and any other forums which the community uses for communication.

- Be welcoming, friendly, and patient.
- Be considerate. Your work will be used by other people, and you in turn will depend on the work of others. Any decision you take will affect users and colleagues, and you should take those consequences into account when making decisions.
- Be respectful. Not all of us will agree all the time, but disagreement is no excuse for poor behaviour and poor manners. We might all experience some frustration now and then, but we cannot allow that frustration to turn into a personal attack. It's important to remember that a community where people feel uncomfortable or threatened is not a productive one. Members of the Topaz community should be respectful when dealing with other members as well as with people outside the Topaz community.
- Be careful in the words that you choose. Remember that sexist, racist, and other exclusionary jokes can be offensive to those around you. Be kind to others. Do not insult or put down other participants. Behave professionally. Remember that harassment and sexist, racist, or exclusionary jokes are not appropriate for the community.
- When we disagree, we try to understand why. Disagreements, both social and technical, happen all the time and Topaz is no exception. It is important that we resolve disagreements and differing views constructively. Remember that we're different. The strength of Topaz comes from its varied community, people from a wide range of backgrounds. Different people have different perspectives on issues. Being unable to understand why someone holds a viewpoint doesn't mean that they're wrong. Don't forget that it is human to err and blaming each other doesn't get us anywhere, rather offer to help resolving issues and to help learn from mistakes.

Original text courtesy of the Speak Up! project.

CHAPTER 8

Blog

Announcing Topaz: A New Ruby

Posted: February 6, 2013

I'm extraordinarily pleased to today announce Topaz, a project I started 10 months ago, to create a brand new implementation of the Ruby programming language (version 1.9.3).

Topaz is written in Python on top of the RPython translation toolchain (the same one that powers PyPy). Its primary goals are simplicity and performance.

Because Topaz builds on RPython, and thus much of the fantastic work of the PyPy developers, it comes out of the box with a high performance garbage collector, and a state of the art JIT (just-in-time) compiler. What does this mean? Out of the box Topaz is extremely fast.

Topaz is far from complete and is missing many builtin methods and classes. However, it does have nearly every element of Ruby, including classes, blocks, many builtin types, all sorts of method calls, and much much more. We don't yet consider it stable, but it's getting closer every day.

If you want to try it out right now, you can grab a nightly build, or *build it yourself*:

- OS X 64-bit
- Linux 32-bit
- Linux 64-bit
- Windows 32-bit

The major goal for the next several months is going to be completeness: adding more features of Ruby, more builtin classes, more standard library modules, and generally getting to a point where real people can run real applications under Topaz (the holy grail, of course, being running Rails). One feature of particular note is `FFI`, once we have this people will begin to be able to run and develop applications that interact with C libraries (such as database bindings).

If you're interested in a high performance Ruby, I'd encourage you to get involved: in testing it out, in writing bug reports, and in helping to build the missing features.

This is just the beginning of Topaz, there's much work to be done. If you'd like to get involved you can find all the source code on Github. The documentation on ReadTheDocs. There's a mailing list and `#topaz` on Freenode IRC to chat. I look forward to seeing you there.

There are innumerable people I'd like to thank for helping out on this project, I'll attempt to enumerate them anyways.

First, Tim Felgentreff. When I started this project 10 months ago I believed it would be the work of a single person for a few months to get it to a release ready state. I could not have been more wrong. Tim has done amazing working to build Topaz, including huge portions of the core object model.

Charles Nutter, Evan Phoenix, and Brian Ford. Each of these individuals are developers of other Ruby implementations (JRuby and Rubinius), and they've provided enormous information and guidance about the Ruby language itself as we've built Topaz.

The PyPy team. Over the last few months the PyPy developers have worked tirelessly to make RPython an even better platform than it already was for building VMs of all sorts, not just for Python. Working with them on this task has been wonderful.

The Travis CI team. They've very kindly donated usage of Private Travis, and it has been phenomenal to use. I can't recommend their product enough.

And no doubt many others. Thanks to everyone I've forgotten who read code over my shoulder, who reviewed and helped clarify documentation, who proofread this blog post, and every other little thing that makes this project possible.

Thank you, I look forward to seeing you around Topaz.

One Year of Topaz

Posted: April 10, 2013

It slipped by so quickly I didn't even notice at the time, but April 7th was the one year anniversary of me working on Topaz. Or at least, of the development work. In reality I'd spent weeks (maybe even months) studying Ruby, learning its quirks, and reading MRI, Rubinius, and JRuby source code to prepare myself.

A year went by very quickly; before long other contributors were helping out, and we were starting to run more and more real Ruby code. Finally we were able to start running useful benchmarks, and then (more importantly) to start running RubySpec, that was a big milestone.

Finally, in February, we open sourced Topaz, and the response from the community has been incredible. So many of you have contributed pull requests, filed bugs, and given your feedback on all aspects of Topaz. It's been amazing to watch.

With your help we now pass (at the time of writing) 3874 examples, and 10356 expectations from RubySpec. And we're passing more every single day.

If you're interested, I'd like to encourage you to *get involved*, it's always a great time.

The last year has been amazing, but I look forward to the next year even more, with your help we're going to make Topaz pass more specs and run more Ruby code, faster.

Don't forget you can always download a nightly build and try Topaz out..

And if you're looking for some fun, grab a copy of the repository and run `git log --reverse -p` – it's fascinating to watch Topaz come to life.

Type Specialized Instance Variables

Posted: July 14, 2013

In Topaz, like most other VMs, all objects are stored in what are called "boxes". Essentially that means when you have something like `x = 5`, `x` is really a pointer to an object which contains `5`, not the value `5` itself. This is often a source of performance problems for VMs, because it generates more garbage for the GC to process and means that to access the value `5` more memory dereferences are needed. Topaz's just-in-time compiler (JIT) is often able to remove these allocations and memory dereferences in individual loops or functions, however it's not able to remove them in structures that stick around in memory, like objects.

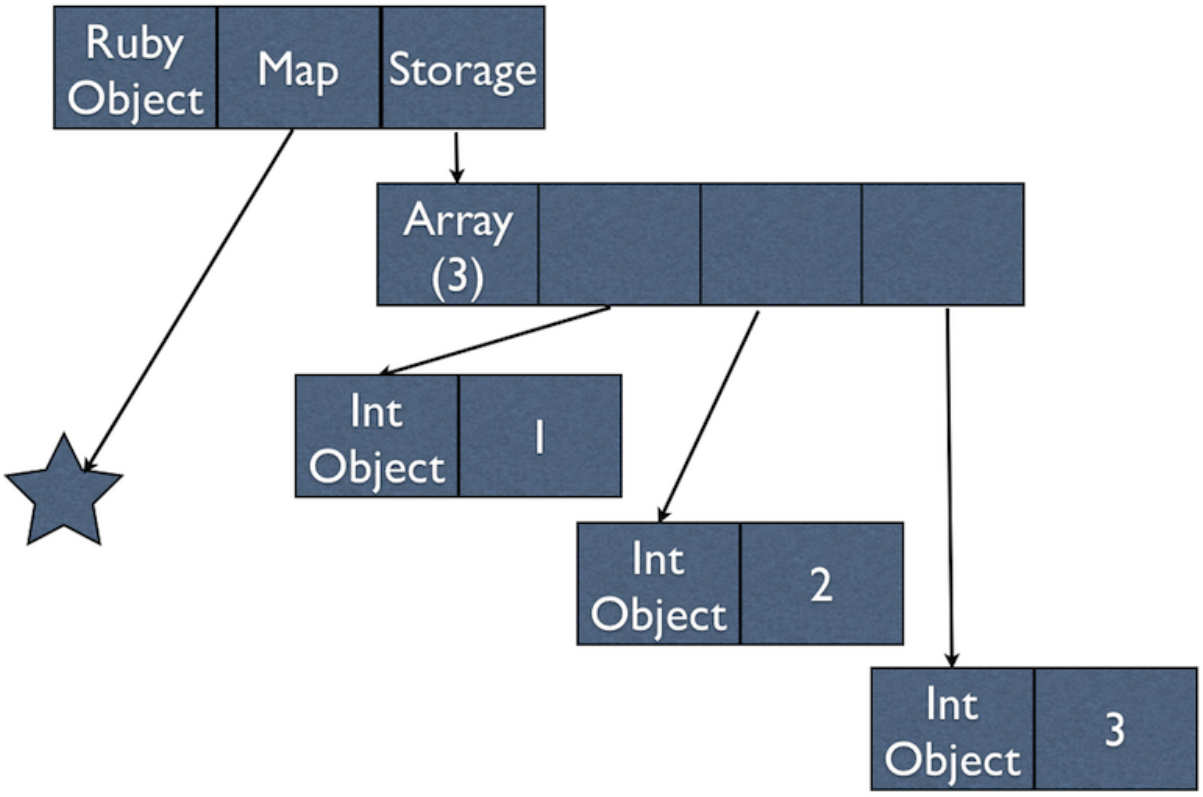
Therefore, over the past week I've been working on an optimization for Topaz called "type specialized instance variables". Basically what that means is that Topaz keeps track of what types instance variables in an object tend to have, and then specializes the storage to remove the indirection for `Fixnum` and `Float` objects.

Let's look at an example:

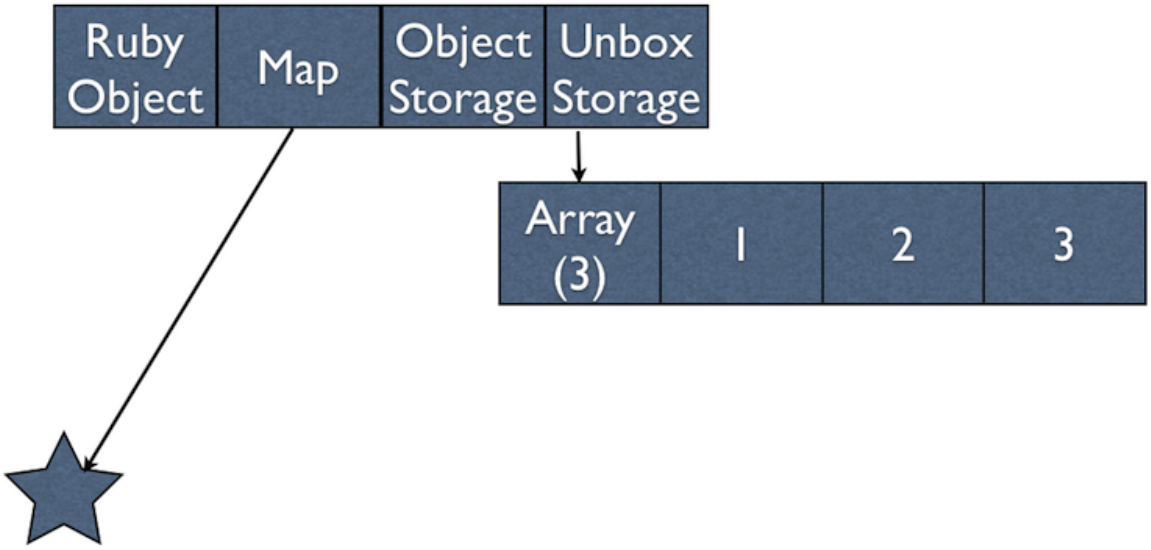
```
class Point
  def initialize(x, y, z)
    @x = x
    @y = y
    @z = z
  end
end

p = Point.new(1, 2, 3)
```

Before this optimization, `p` looked like this in memory. Each box indicates an 8-byte (on 64-bit systems) value, and arrows are pointers:



And after the optimization, it looks like this:



What are these maps? They're a concept out of SELF-88, basically they describe the layout of the object, plus some JIT magic so it's free to find the position and type of a field. Before this patch the map looked like:

```
{
    "x": {"position": 0},
    "y": {"position": 1},
    "z": {"position": 2}
}
```

After this patch it looks like:

```
{
    "x": {"position": 0, "kind": "int"},
    "y": {"position": 1, "kind": "int"},
    "z": {"position": 2, "kind": "int"},
}
```

This is dramatically simplified, if you're interested in the full details of how they work, it's very similar to how they work in PyPy.

With this optimization landed, Topaz will use less memory and be faster for programs that store `Fixnum` and `Float` objects in memory. If you're interested in this type of optimization you can read about a similar one in PyPy for lists that we're in the process of porting to Topaz.

We're looking forward to doing our first release soon, we hope you'll test Topaz out, and give us feedback with the nightly builds until then