Typed Ruby

October ’23 Ruby ZG meetup @ Opulento

Radan Skorić

How many of you are already using gradual typing in your Ruby projects?

Static typing

Verifying the type safety of a program based on analysis of the source code.

Dynamic typing

Verifying the type safety of a program at runtime.

All languages have types!

Static typing + dynamic checks

- Integer overflow
- Array index out of bounds
- Downcasting
- …

+ gradual static typing

Two main tools:

- Sorbet - originally developed at Stripe
  - Type annotation language and type checker
- RBS - originally developed at Square
  - Type annotating language
  - adopted by core team
  - + Steep or TypeProf - a type checker for RBS

Plain ruby

```ruby
def foo(x)
  x + 1
end

ffoo(0)
foo("not an int")
```

Using Sorbet

```
1   # typed: true
2   extend T::Sig
3
4   sig {params(x: Integer).returns(Integer)}
5   def foo(x)
6     x + 1
7   end
8
9   ffoo(0)
10  foo("not an int")
```

```
editor.rb:9: Method ffoo does not exist on T.class_of(<root>)
https://srb.help/7003
     9 |ffoo(0)
         ^^^^
  Did you mean foo? Use -a to autocorrect
    editor.rb:9: Replace with foo
     9 |ffoo(0)
         ^^^^
    editor.rb:5: Defined here
     5 |def foo(x)
         ^^^^^^^^^^

editor.rb:10: Expected Integer but found String("not an int") for
argument x https://srb.help/7002
    10 |foo("not an int")
              ^^^^^^^^^^^^
  Expected Integer for argument x of method Object#foo:
    editor.rb:4:
     4 |sig {params(x: Integer).void}
                    ^
  Got String("not an int") originating from:
    editor.rb:10:
    10 |foo("not an int")
              ^^^^^^^^^^^^

Errors: 2
```

Using RBS

```ruby
def foo(x)
  x + 1
end

ffoo(0)
foo("not an int")
```

```
class Object
  def foo: (Integer x) → Integer
end
```

Typeprof

```
# Errors
test.rb:5: [error] undefined method: Object#ffoo
test.rb:6: [error] failed to resolve overload: Object#foo
```

Steep

```
lib/test.rb:2:2: [error] Cannot find compatible overloading of method `+` of type `::Integer`
| Method types:
|   def +: (::Integer) -> ::Integer
|        | (::Float) -> ::Float
|        | (::Rational) -> ::Rational
|        | (::Complex) -> ::Complex
|
| Diagnostic ID: Ruby::UnresolvedOverloading
|
└   x + "a"
    ~~~~~~~

lib/test.rb:5:0: [error] Type `::Object` does not have method `ffoo`
| Diagnostic ID: Ruby::NoMethod
|
└ ffoo(0)
  ~~~~

lib/test.rb:6:4: [error] Cannot pass a value of type `::String` as an argument of type `::Integer`
|   ::String <: ::Integer
|     ::Object <: ::Integer
|       ::BasicObject <: ::Integer
|
| Diagnostic ID: Ruby::ArgumentTypeMismatch
|
└ foo("not an int")
      ~~~~~~~~~~~~

Detected 3 problems from 1 file
```

Sorbet, more complex example

```ruby
class Environment
  extend T::Sig
  sig { returns(T.nilable(T.self_type)) }
  attr_reader :enclosing

  # Can't use T.self_type because of a bug in Sorbet: https://github
  sig { params(enclosing: T.nilable(Environment)).void }
  def initialize(enclosing = nil)
    @values = T.let({}, T::Hash[T.untyped, T.untyped])
    @enclosing = enclosing
  end

  sig { params(name: String, value: T.untyped).returns(T.untyped) }
  def define(name, value)
    @values[name] = value
  end
```

RBS, more complex example

```ruby
class Environment
  attr_reader :enclosing

  def initialize(enclosing = nil)
    @values = {}
    @enclosing = enclosing
  end

  def define(name, value)
    @values[name] = value
  end
end
```

```
class Environment
  attr_reader enclosing: Environment

  def initialize: ((nil | Environment) enclosing) -> void

  def define: (String name, Object value) -> Object
```

Observations

- Sorbet is:
  - Better documented
  - More expressive
  - More flexible: Inline or RBI files
- RBS is:
  - Actively developed by core team
  - Most likely to stay the default approach
  - Meant to be an ecosystem

LSPs

```ruby
# typed: true
extend T::Sig

# Documentation strings can use _markdown_
# * That includes *lists*!
#
# Tables also work:
#
# | Column 1 | Column 2 |
# | ----- | ----- |
# | True   | *False*   |
sig {returns(String)}
def my_function
```

```ruby
sig {returns(String)}
private def my_function; end
```

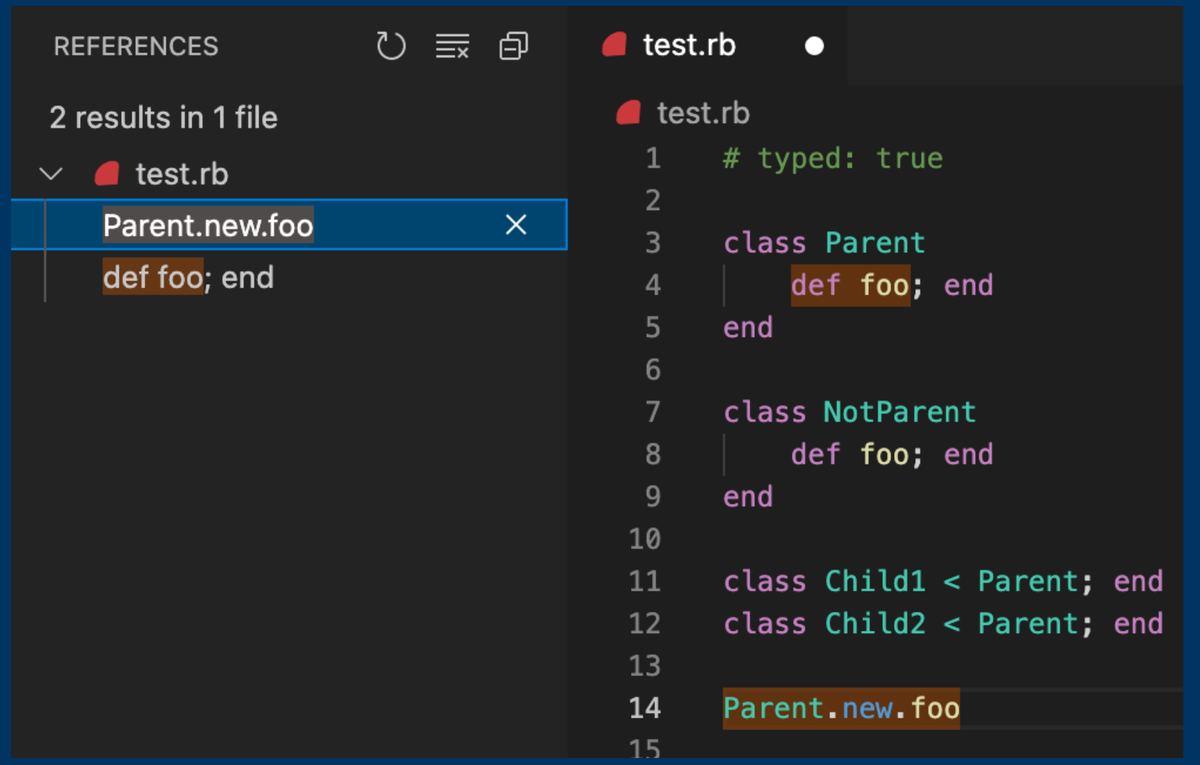
Documentation strings can use *markdown*

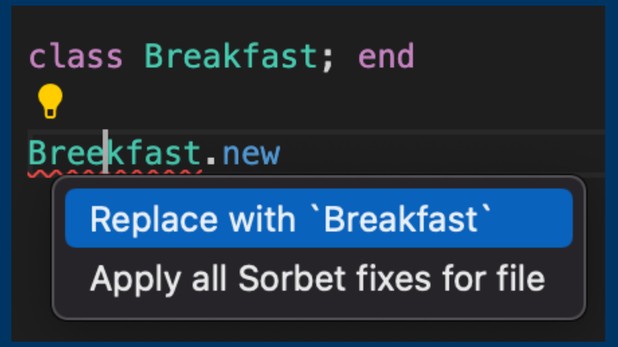
- That includes *lists*!

Tables also work:

Column 1	Column 2
True	*False*

```ruby
my_function
```





Philosophy

Sorbet writes errors as if the code is correct

Steep (RBS) writes errors as if the signature is correct

Type checking tradeoff

Every type checker

will reject

some valid programs

and accept

some invalid programs

Metaprogramming

```ruby
module RubyLox
  module Expressions
    Binary = Struct.new(:left, :operator, :right)
    Call = Struct.new(:callee, :paren, :arguments)
    Get = Struct.new(:object, :name)
    Grouping = Struct.new(:expression)
    Literal = Struct.new(:value)
    Logical = Struct.new(:left, :operator, :right)
    Set = Struct.new(:object, :name, :value)
    Super = Struct.new(:keyword, :method) # rubocop:disable Lint/Struct
    This = Struct.new(:keyword)
    Unary = Struct.new(:operator, :right)
    Variable = Struct.new(:name)
    Assign = Struct.new(:name, :value)

    [Binary, Call, Get, Grouping, Literal, Logical, Set,
     Super, This, Unary, Variable, Assign].each do |expression_class|
      klass_name = expression_class.name.split(":").last
      expression_class.class_eval <<~RUBY
        def accept(visitor)
          visitor.visit#{klass_name}(self)
        end

        def inspect
          AstPrinter.new.visit#{klass_name}(self)
        end
      RUBY
```

Metaprogramming

```ruby
module RubyLox
  module Expressions
    class Binary
    end

    class Call
    end

    class Get
    end

    class Grouping
    end

    class Literal
    end

    class Logical
    end

    class Set
    end

    class Super
    end
```

```ruby
Logical = Struct.new(:left, :operator, :right)
Set = Struct.new(:object, :name, :value)
Super = Struct.new(:keyword, :method) # rubocop:disable Lint/S
This = Struct.new(:keyword)
Unary = Struct.new(:operator, :right)
Variable = Struct.new(:name)
Assign = Struct.new(:name, :value)

# At first I tried to put the list of all classes in a constan
# but sorbet can't parse the splat operator.
SORBET_ANY = T.type_alias {
  T.any(Binary, Call, Get, Grouping, Literal, Logical, Set,
        Super, This, Unary, Variable, Assign)
}

[
  Binary, Call, Get, Grouping, Literal, Logical, Set,
  Super, This, Unary, Variable, Assign
].each do |expression_class|
  klass_name = T.must(expression_class.name).split(":").last
  expression_class.class_eval <<~RUBY
    def accept(visitor)
      visitor.visit#{klass_name}(self)
    end
```

Rails & other gems

- RBS:
  - rbs_rails
  - gem_rbs_collection
- Sorbet:
  - tapioca - generates RBI files

Sorbet tradeoffs

Typed levels:

- ignore - do not even read it
- false - syntax, const resolution and sigs
- true - type errors are reported
- strict - all methods must have sigs
- strong - cannot use untyped

Tradeoffs everywhere

- Testing: what level of coverage?
- Rubocop: how many rules to enable?
- Which tools to adopt: Brakeman, bullet, strict loading …
- Style guides: what style we as a team choose to follow?
- What others can you think of …

Should you adopt it?

It depends:

- On kind of errors you are trying to prevent
- On amount of meta programming
- On your team’s preferences

Questions?