Write a Ruby interpreter in Ruby (& C) for Ruby 3

Koichi Sasada
Cookpad Inc.
<ko1@cookpad.com>
This talk is about

• New proposal to write built-in method definitions in Ruby and C
  • MRI built-in libraries
  • Extension libraries

• Not about how to write VM/GC/⋯ in Ruby
Today’s talk

• Current problems with C-methods
• Proposal: Writing builtin methods in Ruby with C
• Performance hacks
  • Runtime-performance: New FFI instruction
  • Startup-time: New compiled binary features
Koichi Sasada
http://atdot.net/~ko1/

• A programmer
  • 2006-2012 Faculty
  • 2012-2017 Heroku, Inc.
  • 2017- Cookpad Inc.

• Job: MRI development
  • Core parts
    • VM, Threads, GC, etc
• Daily Ruby Puzzles
  • You can get a paper at our booth.
  • Complete “Hello world” program by adding minimum letters

```ruby
# example
def foo
  "Hello world" if false
end
puts foo
```
Background
MRI built-in method definitions

• Well-known fact: MRI is written in C
• Most of built-in methods are written and defined in C
• A few methods written in Ruby, “prelude.rb” introduced from Ruby 1.9.0.
  • Unlike libraries, it is loaded by default and bundled with a MRI binary (you can’t edit it)
Built-in methods definitions in C

// quote from string.c Init_String()

rb_cString = rb_define_class("String", rb_cObject);

...  
rb_define_method(rb_cString, "<=>", rb_str_cmp_m, 1);
rb_define_method(rb_cString, "==", rb_str_equal, 1);
rb_define_method(rb_cString, "===", rb_str_equal, 1);

...  
rb_define_method(rb_cString, "length", rb_str_length, 0);

...  
When the method called, corresponding C function will be called.
Built-in methods definitions in C

Method body function

```c
# String#length impl.
VALUE
rb_str_length(VALUE str)
{
    return LONG2NUM(
        str_strlen(str, NULL));
}
```
How many classes/methods?

```ruby
# ruby --disable-gems

require 'objcspace'
p ObjectSpace.count_objects[:T_CLASS] +
    ObjectSpace.count_objects[:T_MODULE]
  #=> 526 # How many classes/modules?
p ObjectSpace.count_imemo_objects[:imemo_ment]
  #=> 2363 # How many method entries?
```
Problems

1. Annotation (meta data)
2. Performance
3. Productivity
4. API change for “Context”
Problem 1
Annotation (meta data)

• Lack of meta-data compare with methods defined in Ruby (Ruby methods)
  • For example: “Method#parameters”

```ruby
def hello(msg) puts "Hello #{msg}"; end
p method(:hello).parameters
  #=> [[:req, :msg]]

p method(:is_a?).parameters
  #=> [[:req]]  # no parameter name
```

Ruby’s functionality issue

Other meta data
• Backtrace (stack-prof)
• Source (if exists)
• ...
Problem 1
Annotation (meta data)

• Need more information for further optimizations
• We can not know behaviors of C-methods unless analyzing “C-code” and it is feasible.
  • Can throw exceptions?
  • Has side-effect?
• If we know a method is “pure”, we can apply more aggressive optimizations, such as passing “frozen” string instead of making new Strings.
  • Ex: `str.gsub(“goodby”, “hello”)`
  • In this example, two strings can be frozen singleton objects (don’t need “frozen-string-literal” pragma).
Problem 1
Annotation (meta data)

• We can not know how many methods are defined before running.
  • We can not allocate method table at once.
  • Now we grow a method table on demand.
Problem 2
Performance

• Known fact: “C” is faster than Ruby
  • Most of case, it is true
  • Sometimes it is false

• Keyword parameters and exception handling are typical examples
Problem 2
Performance: Keyword parameters in C

```c
static VALUE
tdummy_func_kw(int argc, VALUE *argv, VALUE self)
{
    VALUE h;
    ID ids[2] = {rb_intern("k1"), rb_intern("k2")};
    VALUE vals[2];

    rb_scan_args(argc, argv, "0:", &h);
    rb_get_kwargs(h, ids, 0, 2, vals);
    return tdummy_func2(self,
                          vals[0] == Qundef ? INT2FIX(1) : vals[0],
                          vals[1] == Qundef ? INT2FIX(2) : vals[1]);
}
```

# Ruby
def dummy_func_kw(k1: 1, k2: 2)
    dummy_func2(k1, k2)
end
Performance problem on Keyword parameters in C

C is faster because of calling overhead of dummy_func2()

Ruby is faster because Ruby version has special optimization for keyword parameters (don’t make Hash objects)
Problem 2
Performance: Exception in C

```c
static VALUE
dummy_body(VALUE self)
{
    return Qnil;
}
static VALUE
dummy_rescue(VALUE self)
{
    return Qnil;
}
static VALUE
tdummy_func_rescue(VALUE self)
{
    return rb_rescue(dummy_body, self,
                      dummy_rescue, self);
}
```

# in Ruby
def em_dummy_func_rescue
  rescue
end
Problem 2
Performance: Exception in C
Problem 3
Productivity

- Ruby C-API
  - It is **easy** to make simple methods such as `String#length` in C
  - It is **difficult** to make methods using **complex features**.
    - Exception handling
    - Keyword parameters
    - Iterators

```c
# String#length impl.
VALUE
rb_str_length(VALUE str)
{
    return LONG2NUM(
        str_strlen(str, NULL));
}
```
Problem 3
Productivity: Keyword parameters in C

```c
static VALUE
tdummy_func_kw(int argc, VALUE *argv, VALUE self)
{
    VALUE h;
    ID ids[2] = {rb_intern("k1"), rb_intern("k2")};
    VALUE vals[2];

    rb_scan_args(argc, argv, "0:", &h);
    rb_get_kwargs(h, ids, 0, 2, vals);
    return tdummy_func2(self,
                        vals[0] == Qundef ? INT2FIX(1) : vals[0],
                        vals[1] == Qundef ? INT2FIX(2) : vals[1]);
}
```

```ruby
# Ruby
def dummy_func_kw(k1: 1, k2: 2)
dummy_func2(k1, k2)
end
```
Problem 3
Productivity: Exception in C

```c
static VALUE
dummy_body(VALUE self)
{
    return Qnil;
}
static VALUE
dummy_rescue(VALUE self)
{
    return Qnil;
}
static VALUE
tdummy_func_rescue(VALUE self)
{
    return rb_rescue(dummy_body, self,
                     dummy_rescue, self);
}
```

# in Ruby
def dummy_func_rescue
  nil
  rescue
    nil
end
Problem 3
Productivity

- **Written in C is reasonable**
  - if it is performance required (frequently used)
  - if it is easy to implement
  - if we cannot implement it in Ruby

- **Written in Ruby is reasonable**
  - if it is not frequently used, non-performance required features (such as TracePoint, and so on)
  - if it should try with proto-type
Problem 4  
API changes for “Context”

• C API needs update to accept “Context”
  • To implement Guild system (or other parallel execution mechanism), we need to access “context” object.
  • Current API doesn’t accept a context object

```c
# String#length impl.
VALUE
rb_str_length(VALUE str)
{
    return LONG2NUM(
        str_strlen(str, NULL));
}
```

```c
# mrb_value
mrb_str_size(mrb_state *mrb, mrb_value self)
{
    mrb_int len = RSTRING_CHAR_LEN(self);
    return mrb_fixnum_value(len);
}
```

mruby accepts “mrb_state” data as context!
Problem 4
API changes for “Context”

• Getting “Context” from Thread-local-storage (TLS) is one idea
  • Good: We can keep current API. It is hard to bring new specification without carrots (飴).
  • Bad: Very slow to access TLS, especially from .so (.dll)

For details:
Problems and requests

1. Annotation (meta data)
   • We need **DSL** instead of C code

2. Performance
   • Sometimes **Ruby is faster**

3. Productivity
   • Sometimes **Ruby is enough** to implement

4. API changes for “Context”
   • We need **brand new** definition APIs
Problems and requests

1. Annotation (meta data)
   - We need **DSL** instead of C code

2. Performance
   - Sometimes **Ruby is faster**

3. Productivity
   - Sometimes **Ruby is enough** to implement

4. API changes for "Context"
   - We need **brand new** definition APIs

**We should know good DSL supporting language!!**
Solution

Write method definitions in Ruby with C.
Solution
Writing definitions (declarations) in Ruby

• Write a definitions or declarations in Ruby
  • We can analyze Ruby code in advance
  • We can embed meta-data (method attribute) in Ruby DSL

• Call C functions if needed
  • We only need to call already implemented C functions (w/ small modifications)

• We don’t need to replace all of defs
  • We can move new defs gradually.
Solution
Writing definitions (declarations) in Ruby

1. Annotation (meta data)
   - Write a code in Ruby and analyze it
   - Add additional annotation method

2. Performance
   - Introduce **new FFI feature** to call C function
   - Sometimes pure-Ruby is enough

3. Productivity
   - Some cases (kwparams, and so on) are definitely easy to write in Ruby

4. API changes for “Context”
   - New FFI passes “Context” parameter “ec”
Where should we write definitions?

Current definitions

```c
// string.c
str_length() {...}

Init_String(){
  ...
  rb_define_method(...);
  rb_define_method(...);
  rb_define_method(...);
  ...
}
```

Proposed definitions

```ruby
# string.rb
class String
  def ...; end
  def length
    ...
  end
end
```

```c
// string.c
str_length() {
  ...
}
```

compile

Ruby binary

analysis and combine

Ruby binary
New FFI feature to call C functions

# string.rb
class String
  ...
  def length
    __ATTR__.pure
    __C__.str_length
  end
end

# String#length impl. with new FFI
static VALUE
str_length(rb_ec_t *ec, VALUE str)
{
  return LONG2NUM(str_strlen(str, NULL));
}

NOTE
Keywords are not fixed yet
These keywords are enabled on special compile mode (not a syntax proposal for ordinal Ruby)
Programming with new FFI

• You can use Ruby features as you want 😊
  • Parameter analyzing (opt, kw, …)
  • Complex feature like exception handling, iterators, …

• You can write C code 😊
  • Reuse existing C implementation
  • Write fast code with C

• You need to care a bit more 😒
  • GVL release (interrupt) timing, GC timing…
Questions:
Is it slow than C methods?

• Runtime overhead concerns
  • FFI can be runtime overhead 😞
    • Primitive methods like String#length can affect this kind of overhead

• Startup time concerns
  • Compile time can increase a loading time 😞
Today’s technical achievements

• Runtime overhead
  → Fast FFI implementation by new instructions

• Loading time
  → Improve compiled binary format
Fast FFI implementation by new instructions
FFI (Foreign Function call) instruction “invokecfunc”

• Introduce a new instruction “invokecfunc” in the virtual machine

```ruby
# string.rb
class String
  def length
    ___C___.str_length
  end
end
```

```
== disasm: #<ISeq:length@string.rb:10>
0000 invokecfunc
0002 leave
```
Optimize “invokecfunc”

• In fact, “invokecfunc” is slow compare with C methods
  • Additional overhead for VM stack manipulation
  • Additional overhead for VM frame manipulation

• Policy: DO EVERYTHING I CAN DO
Optimize “invokecfunc”

• Most of cases, methods become delegator type definitions. Passing same parameters.

```python
def dummy_func2 a, b
    __C__.dummy_func2(a, b)
end
```

```assembly
0000  getlocal  a@0,  0
0003  getlocal  b@1,  0
0006  invokecfunc <dummy_func2/2>
0008  leave
```

a, b: Same as parameters
Optimize “invokecfunc”

• Most of cases, methods become delegator type definitions. Passing same parameters.
→ Special instruction: invokecfuncwparam

```python
def dummy_func2 a, b
    __C__.dummy_func2(a, b)
end
```

```plaintext
0000 invokecfuncwparam<dummy_func2/2>
0002 leave
```
Optimize “invokecfunc”

- Most of cases, methods become delegator type definitions. “call&return”

```python
def dummy_func2 a, b
   __C__.dummy_func2(a, b)
end
```

```assembly
0000 invokecfuncwparam<dummy_func2/2>
0002 leave
```
Optimize “invokecfunc”

• Most of cases, methods become delegator type definitions. “call&return”

→ Special instruction: invokecfuncwparamandleave

```python
def dummy_func2 a, b
    __C__.dummy_func2(a, b)
end
```

```assembly
0000 invokecfuncwparamandleave ...
0002 leave
```

NOTE: To support TracePoint, “leave” is required yet.
Evaluation

• Prepare empty methods and compare them
  • Traditional C methods
    • rb_define_method(…)
  • Ruby methods calls empty C function
    • def dummy_func; __C__.dummy_func(); end

• Apply optimizations
  • baseline: invokefunc
  • w/param: invokefuncwparm
  • w/param&leave: invokefuncwparamandleave
Evaluation
With positional arguments

With optimize, faster than C methods! 😊
Evaluation
With optional arguments

Ruby’s “opt params”
is slower 😞
Evaluation
With keyword arguments

```python
def dummy_func_kw k1:1, k2:2
    __C__.dummy_func2(k1, k2)
end
```

Ruby’s “nokw” is slower 😞

Ruby’s “kwparms” is 4~5 faster! 😊
Evaluation
Rescue

Ruby’s “rescue” is 1.5 faster! 😊

def dummy_func_rescue
    __C__.dummy_func0
rescue
    __C__.dummy_func0
end

static VALUE
dummy_func_rescue(VALUE s)
{
    return
    rb_rescue(dummy_func0, self,
              dummy_func0, self);
}
FFI instruction “invokecfunc”

Summary

• **Good performance ☺**
  • Many cases, “__C__.func” new FFI calls are faster than C methods
  • Some cases, significant improvements (keyword parameters / exception handling)
  • Optional parameters are slow 😞
    • Built-in methods have many optional arguments so it is important problem

• **Enjoy Ruby’s productivity ☺**
FFI instruction “invokecfunc”

Future work

• Introduce arity overloading for slow opt params

# example syntax
overload def foo(a)
  __C__.foo1(a)
end
overload def foo(a, b)
  __C__.foo2(a, b)
end

At method dispatch, we can find appropriate method body and we can cache it in inline cache!
Related work

Interestingly, C code can refer Ruby variables 😊

```
# Writing C in Ruby code

def open_fd(path)
    fd = __C__(%q{ // passing string literals to __C__ methods
        /* C implementation */
        return INT2FIX(open(RSTRING_PTR(path), O_RDONLY));
    })
    raise 'open error' if fd == -1
    yield fd
ensure
    raise 'close error' if -1 == __C__(%q{
        /* C implementation */
        return INT2FIX(close(FIX2INT(fd)));
    })
end
```

Koichi Sasada: Ricsin: A System for “C Mix-in to Ruby” (Ricsin: RubyにCを埋め込むシステム) (2009.3)
Improve compiled binary format
Compiled binary format?

- **Bytecode dumped binary**
  - `bin = RubyVM::InstructionSequence#to_binary`
  - `RubyVM::InstructionSequence.load_from_binary(bin)`
  - Introduced from Ruby 2.3 (by me 😊)

- **AOT compiling feature**
  - `bootsnap (maybe) use it`
Compiled binary: creation and usage

Faster loading because we don’t need to parse/compile.
Embed compiled binary into MRI

• For short startup time, we can bundle compiled binary with MRI binary

```
const char compiled_binary[] = {
  0x59, 0x41, 0x52, 0x42, 0x02, 0x00, 0x00, 0x00, 0x00, 0x07, 0x00, 0x00, 0x00, 0xbc, 0x56, 0xf8,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0xf0, 0x87, 0x00, 0x00, 0xce, 0x0b,
  0x00, 0x00, 0x8d, 0x9f, 0x00, 0x00, 0xf8, 0x6a, 0xe1, 0x00, 0x38, 0x4b, 0xdf, 0x00, 0x00,
  0x6b, 0xe1, 0x00, 0x88, 0xd8, 0xf5, 0x00, 0x78, 0x38, 0x36, 0x5f, 0x36, 0x34, 0x2d, 0x6c
};
```
Further optimizations for startup-time

• Lazy loading
  • RubyKaigi 2015, but not enabled yet on MRI
• Multiple file supported binary
Technique
Lazy loading

Idea: Load only really used ISeq
Most of methods are not used in a process

(1) Load and make an empty toplevel ISeq

Ruby script

```ruby
class C1
  def m1; end
  def m2; end
end
C1.new.m2
class C2; end
```

Compiled binary
toplevel C1,C2, m1,m2

Toplevel (empty)
Technique
Lazy loading

Idea: Load only really used ISeq
Most of methods are not used in a process

Ruby script
```ruby
class C1
def m1; end
def m2; end
end
class C2; end
```

(2) Load toplevel ISeq and make empty
C1, C2 empty

```ruby
class C1 (empty)
class C2 (empty)
```
**Technique**

**Lazy loading**

**Idea:** Load only really used ISeq
Most of methods are not used in a process

(3) Load C1 and evaluate C1
Define m1 and m2 with empty ISeqs

Ruby script

```ruby
class C1
  def m1; end
  def m2; end
end
C1.new.m2

C2; end
```

Diagram:

- **Compiled binary**
  - `toplevel`
    - `C1, C2, m1, m2`

- **Toplevel**
  - `class C1`
    - `def m1 (empty)`
    - `def m2 (empty)`

- **Class C2 (empty)**
Technique: Lazy loading

Idea: Load only really used ISeq. Most of methods are not used in a process.

(4) Load m2 and invoke m2
Technique
Lazy loading

Idea: Load only really used ISeq
Most of methods are not used in a process

(4) Load C2 and evaluate C2

Ruby script
class C1
def m1; end
def m2; end
class C2; end
C1.new.m2

class C1
def m1; end
def m2; end

class C2 (empty)

def m1 (empty)
def m2
Technique
Multiple file supported binary

• Current compiled binary can contain 1 file
• Expand it to support multiple files
  • We can share resources more.
Evaluation

• Create 3000 classes, they have 1~20 methods
  • One file
    • def.rb: 582KB (3000 classes definitions)
    • compiled binary: 16MB
    • translated C code: 79MB
  • Separate files
    • .rb: 3000 files
    • compiled binary in 1 file: 17MB
    • translated C code: 86MB
• Corresponding C code: 4.2MB
  • using rb_define_method()

```ruby
class C0
  def m0; m; end
  def m1; m; end
  def m2; m; end
  def m3; m; end
  def m4; m; end
  def m5; m; end
  def m6; m; end
  def m7; m; end
  def m8; m; end
  def m9; m; end
  def m10; m; end
  def m11; m; end
  def m12; m; end
end
class C1
  ...
```
## Evaluation

### Startup time w/ additional 3000 classes

<table>
<thead>
<tr>
<th></th>
<th>Startup time (ms)</th>
<th>Compare w/ C inits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Miniruby (w/o 3k classes)</strong></td>
<td>6.4</td>
<td></td>
</tr>
<tr>
<td>C inits</td>
<td><strong>27.5</strong></td>
<td></td>
</tr>
<tr>
<td>binary/one</td>
<td>87.1</td>
<td>x3.2</td>
</tr>
<tr>
<td>binary/sep</td>
<td>82.1</td>
<td>x3.0</td>
</tr>
<tr>
<td>binary/one/lazy</td>
<td>47.1</td>
<td>x1.7</td>
</tr>
<tr>
<td>binary/sep/lazy</td>
<td>51.1</td>
<td>x1.9</td>
</tr>
<tr>
<td>require/one</td>
<td>161.3</td>
<td>x5.9</td>
</tr>
<tr>
<td>require/sep</td>
<td>425.2</td>
<td>x15.5</td>
</tr>
</tbody>
</table>
Future work

• Pre-allocated method table
  • We can know which methods are defined at startup, so we can pre-allocate method table in TEXT area (reduce making table overhead)

• Bulk-define instruction
  • Multiple definitions can be done at once with special bulk definition instruction

• Make compact binary format
  • Now it consume huge bytes. Can anyone try?
MRI build Bootstrap

MRI C code

FFI name table

miniruby

C code of compiled binary

ruby (libruby)

Build

load&dump

gen

build

Ruby script

Ruby script

array.rb

...
Today’s talk

• Current problems with C-methods
• Proposal: Writing builtin methods in Ruby with C
• Performance hacks
  • Runtime-performance: New FFI instruction
  • Startup-time: New compiled binary features
Thank you for your attention

Write a Ruby interpreter in Ruby for Ruby 3

“Ask the speaker” at Cookpad booth (3F) next break

Koichi Sasada
Cookpad Inc.
<ko1@cookpad.com>