Three Ruby usages

Kouhei Sutou

ClearCode Inc.

RubyKaigi 2014
2014/09/20
Silver sponsor

SILVER SPONSORS

ClearCode Inc.

ClearCode runs free software business and contributes to free software.
Goal

✓ You know three Ruby usages
✓ High-level interface
✓ Glue
✓ Embed
✓ You can remember them later
Targets

✓ High-level interface
  ✓ Pure Rubyists

✓ Glue
  ✓ Rubyists who can write C/C++

✓ Embed
  ✓ Rubyists who also write C/C++
Case study

"Implement distributed full-text search engine in Ruby"

Abbreviation: DFTSE = Distributed Full-Text Search Engine
DFTSE?

1: Full-text search

2: Distribute sub requests

3: Merge responses
Why do we use DFTSE?

I'm developing Droonga

(A DFTSE implementation in Ruby)

😃
High-level interface

Three Ruby usages

✓ High-level interface

✓ Target: Pure Rubyists

✓ Glue

✓ Embed
High-level interface

✓ Provides
   lower layer feature to higher layer

✓ With simpler/convenience API
High-level interface

Higher layer users

High-level interface

Feature

Application/Library

Three Ruby usages
Example

Vagrant

Developers

Vagrantfile

Build development environment

Vagrant

Active Record

Developers

Object based API

Access data in RDBMS

Active Record
Droonga: High-level IF

- DFTSE components
- ✓ Full-text search engine
- ✓ Messaging system
- ✓ Cluster management
- ✓ Process management
**Messaging system**

1: Full-text search

2: Distribute sub requests

3: Merge responses

FTSE

DTFSE

Worker process

Messaging system
Messaging system

✓ Provides distributed search feature

✓ Plan how to search

✓ Distribute requests

✓ Merge responses

✓ Users don't know details
Characteristic

✓ Plan how to search
  ✓ May speed up/down over 100 times

✓ Distribute requests
  ✓ Network bound operation

✓ Merge responses
  ✓ CPU and network bound operation
Point

✓ Algorithm is important
✓ Need to find new/existing better algorithm
✓ "Rapid prototype and measure" feedback loop is helpful
✓ Ruby is good at rapid dev.
Glue

Three Ruby usages

✓ High-level interface

✓ Glue

✓ Target:
  Rubyists who can write C/C++

✓ Embed
Glue

Export a feature

Ruby

Other Language

Combine features

Feature

Glue
Example

Active Record

mysql2 gem

Access to MySQL
libmysqlclient.so

Vagrant

VM
(VirtualBox)

Provision
(Chief)

Feature

Glue
Why do we glue?

✓ Reuse existing features
How to glue

✓ Use external library
  ✓ Implement bindings (mysql2 gem)

✓ Use external command
  ✓ Spawn command (Vagrant)

✓ Use external service
  ✓ Implement client
Glue in Droonga

✓ **Rroonga**: Groonga bindings
   ✓ **Groonga**: FTSE C library (and server)

✓ **Cool.io**: libev bindings
   ✓ **libev**: Event loop C library
     (Based on I/O multiplexing and non-blocking I/O)

✓ **Serf**: Clustering tool (in Droonga)
Rroonga in Droonga

1: Full-text search

2: Distribute sub requests

3: Merge responses

FTSE

Worker process

Messaging system

Three Ruby usages

Powered by Rabbit 2.1.4
FTSE in Droonga

✓ Must be fast!
✓ CPU bound processing
For fast Rroonga

✓ Do heavy processing in C
  ✓ Nice to have Ruby-ish API
✓ Less memory allocation
  ✓ Cache internal buffer
✓ Multiprocessing
  ✓ Groonga supports multiprocessing
Search

```ruby
Groonga::Database.open(ARGV[0])
entries = Groonga["Entries"]

entries.select do |record|
    record.description =~ "Ruby"
end
```
Search - Pure Ruby (ref)

Groonga::Database.open(ARGV[0])
entries = Groonga["Entries"]

entries.find_all do |record|
  # This block is evaluated for each record
  /Ruby/ =~ record.description
end
Search impl.

# (2) Evaluate expression in C
entries.select do |record|
  # (1) Build expression in Ruby
  # This block is evaluated only once
  record.description =~ "Ruby"
end
Search impl. - Fig.

```ruby
entries.select do |record|
  record.description =~ "Ruby"
end
```

Build request → Search

Evaluate expression → Result set

C

Groonga by Groonga project
Search - Benchmark

✓ Ruby  (It's already showed)
✓ C
grn_obj *expr;
grn_obj *variable;
const gchar *filter = "description @ "Ruby";"
grn_obj *result;

GRN_EXPR_CREATE_FOR_QUERY(&ctx, table, expr, variable);
grn_expr_parse(&ctx, expr,
    filter, strlen(filter), NULL,
    GRN_OP_MATCH, GRN_OP_AND,
    GRN_EXPR_SYNTAX_SCRIPT);
result = grn_table_select(&ctx, table, expr, NULL, GRN_OP_OR);
grn_obj_unlink(&ctx, expr);
grn_obj_unlink(&ctx, result);
# Search - Benchmark

Ruby impl. is fast enough 😊

<table>
<thead>
<tr>
<th>Impl.</th>
<th>Elapsed time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0.6ms</td>
</tr>
<tr>
<td>Ruby</td>
<td>0.8ms</td>
</tr>
</tbody>
</table>

(Full-text search with "Ruby" against 72632 records)
Embed

Three Ruby usages
✓ High-level interface
✓ Glue
✓ Embed
✓ Target:
  Rubyists who also write C/C++
Embed

Internal engine

C/C++ application
C/C++ library

Implement some features in Ruby

Interface

Plugin API
Configuration

C/C++ application
C/C++ library

Three Ruby usages
Powered by Rabbit 2.1.4
Examples

Internal engine

Implement query optimizer in Ruby

Interface

vim-ruby

VIM

by Yukihiro Matsumoto
Embed in Droonga
CRuby vs. mruby

✓ CRuby
  ✓ Full featured!
  ✓ Signal handler isn't needed 😞

✓ mruby
  ✓ Multi-interpreters in a process!
  ✓ You may miss some features 😞
mruby in Groonga

✓ Query optimizer

✓ Command interface (plan)
  ✓ Interface and also high-level interface!

✓ Plugin API (plan)
  ✓ Interface!
Query optimizer

Query -> Optimize (Query Optimizer) -> Optimized query -> Full-text search (Evaluator) -> Result set

Three Ruby usages Powered by Rabbit 2.1.4
Query optimizer

✓ Plan how to search
  ✓ It's a bother 😞

✓ Light operation than FTS

✓ Depends on data
  (Choose effective index, use table scan and so on)
Example

```
rank < 200 && rank > 100
```
Simple impl.

```
rank  1 2 ··· 100 101 ··· 199 200 ··· ··· 10000

rank < 200

&&

101 ··· 199

rank < 200 && rank > 100
```
Simple impl.

✓ Slow against many out of range data
Optimized impl.

<table>
<thead>
<tr>
<th>rank</th>
<th>1 2 ··· 100 101 ··· 199 200 ··· ··· 10000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100 &lt; rank &lt; 200</td>
</tr>
<tr>
<td></td>
<td>101 ··· 199</td>
</tr>
<tr>
<td></td>
<td>rank &lt; 200 &amp;&amp; rank &gt; 100</td>
</tr>
</tbody>
</table>
Is embedding reasonable?

Measure
Measure

✓ mruby overhead
✓ Speed-up by optimization
# Overhead

Small overhead: Reasonable 😊

<table>
<thead>
<tr>
<th># conds</th>
<th>mruby</th>
<th>Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>🌟</td>
<td>0.24ms</td>
</tr>
<tr>
<td>1</td>
<td>🚔</td>
<td>0.16ms</td>
</tr>
<tr>
<td>4</td>
<td>🌟</td>
<td>0.45ms</td>
</tr>
<tr>
<td>4</td>
<td>🚔</td>
<td>0.19ms</td>
</tr>
</tbody>
</table>
## Speed-up

Fast for many data: Reasonable 😊

<table>
<thead>
<tr>
<th># records</th>
<th>mruby</th>
<th>no mruby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.29ms</td>
<td>0.31ms</td>
</tr>
<tr>
<td>10000</td>
<td>0.31ms</td>
<td>2.3ms</td>
</tr>
<tr>
<td>100000</td>
<td>0.26ms</td>
<td>21.1ms</td>
</tr>
<tr>
<td>1000000</td>
<td>0.26ms</td>
<td>210.2ms</td>
</tr>
</tbody>
</table>
Note

✓ Embedding needs many works
  ✓ Write bindings, import mruby your build system and ...

✓ How to test your mruby part?
  ✓ And how to debug?
Conclusion 1

✓ Describe three Ruby usages

✓ High-level interface

✓ Glue

✓ Embed
Conclusion 2

✓ High-level interface
✓ Target: Pure Rubyists
✓ Provides lower layer feature to higher layer w/ usable interface
✓ Ruby's flexibility is useful
Glue

Target: Rubyists who can write C/C++

Why: Reuse existing feature

To be fast, do the process in C
Conclusion 4

✓ Embed

✓ Target:
  Rubyists who also write C/C++

✓ Why:
  Avoid bother programming by Ruby
Conclusion 5

✓ Embed

✓ Is it reasonable for your case?
✓ You need many works
✓ Very powerful
  if your case is reasonable 😊
Announcement

✓ ClearCode Inc.
✓ A silver sponsor
✓ Is recruiting
✓ Will do readable code workshop

✓ The next Groonga conference
✓ It's held at 11/29