"We should forget about small efficiencies: say about 97% of the time: premature optimization is the root of all evil."

- Don Knuth, December, 1974
In establishing engineering disciplines a 12% improvement, easily obtained, is never considered marginal, and I believe the same viewpoint should prevail in software engineering.

- Don Knuth, December, 1974
"Ruby is designed to make programmers happy."

- Matz, September, 2000
Levels of Optimization

- Design
- Source
- Build
- Compile
- Runtime
## Levels of Optimization

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Architecture and algorithms (e.g. n + 1 queries)</td>
</tr>
<tr>
<td>Source</td>
<td>Writing fast Ruby</td>
</tr>
<tr>
<td>Build</td>
<td>Setting build flags (e.g. ./configure)</td>
</tr>
<tr>
<td>Compile</td>
<td>mrbc, jruby, rbx compile</td>
</tr>
<tr>
<td>Runtime</td>
<td>Thanks Matz &amp; Koichi (e.g. RUBY_GC_MALLOC_LIMIT)</td>
</tr>
</tbody>
</table>
"IT IS OFTEN A MISTAKE TO MAKE
A PRIORI OF JUDGEMENTS ABOUT
WHAT PARTS OF A PROGRAM ARE
REALLY CRITICAL, SINCE THE UNIVERSEAL
EXPERIENCE OF PROGRAMMERS
WHO HAVE BEEN USING MEASUREMENT
TOOLS HAS BEEN THAT THEIR
INTUITIVE GUESSES FAIL."

- DON KNUTH, DECEMBER, 1974
require 'benchmark'
n = 50
Benchmark.bm do |x|
  x.report { n.times { fast } }
  x.report { n.times { slow } }
end
require 'benchmark'
n = 50_000
Benchmark.bm do |x|
  x.report { n.times { fast } }
  x.report { n.times { slow } }
end
require 'benchmark/ips'

Benchmark.ips do |x|
  x.report('fast') { fast }
  x.report('slow') { slow }
end
<table>
<thead>
<tr>
<th>Source</th>
<th>Optimize at the code level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significant</td>
<td>At least 12% improvement</td>
</tr>
<tr>
<td>Easy</td>
<td>Code should be easier to read</td>
</tr>
<tr>
<td>Happy</td>
<td>High quality Ruby</td>
</tr>
</tbody>
</table>
Proc#call versus yield

def slow(&block)
  block.call
end

def fast
  yield
end
Results

slow  950950.6 (±14.0%) i/s
fast  5508226.3 (±15.5%) i/s

Over 5X faster!
def slow
  Proc.new.call
end

def fast
  yield
end
Block versus Symbol#to_proc

```
(1..100).map { |i| i.to_s }
(1..100).map(&:to_s)
```
Results

slow 47524.3 (±7.6%) i/s
fast 56823.2 (±7.2%) i/s

20% faster!
Avoid Symbol.to_proc for performance reasons in Ruby 1.8

```ruby
# active_record.rb
# active_record/associations/has_many_association.rb

before_destroy do |record|
  case @reflection.options[:dependent]
  when :destroy
    records.each(&:destroy)
  when :delete_all
    @reflection.klass.delete(records.map(&:id))
  else
    updates = { @reflection_primary_key_name => nil }
    conditions = { @reflection.association_primary_key => records.map(&:id) };
    @reflection.klass.update_all(updates, conditions)
  end
end
```
Pass symbol as an argument instead of a block #16833

sferik commented 6 days ago

Using Ruby's `Symbol#to_proc` is considerably more concise than using block syntax. It's also about 20% faster (see benchmarks below). `Symbol#to_proc` is already used in many places throughout the Rails codebase, but not everywhere. This patch makes the codebase more consistent and concise. In some cases, it reduces the number of lines of code. For example, in `railties/lib/rails/application/routes_reloader.rb`:

```ruby
def finalize!
  route_sets.each do |routes|
    routes.finalize!
  end
end
```

becomes:

```ruby
def finalize!
  route_sets.each(&:finalize!)
end
```

The net result is 30 fewer lines of code.
Enumerable#map and Array#flatten versus Enumerable#flat_map

```ruby
enum.map do
  # do something
end.flatten(1)

enum.flat_map do
  # do something
end
```
Results

slow 12348.2 (±9.0%) i/s
fast 56647.8 (±7.2%) i/s

Over 4.5X faster!
Replace map.flatten(1) with flat_map #14240

Sferik commented on Mar 1

Using flat_map is equivalent to map.flatten(1) but simpler and an order of magnitude faster.

```
require 'benchmark/ips'

ARRAY = 100.times.map { 0..0.to_a }

def slow
  ARRAY.map { |x| x }.flatten(1)
end

def fast
  ARRAY.flat_map { |x| x }
end

Benchmark.ips do |x|
  x.report('slow') { slow }
  x.report('fast') { fast }
end
```
Optimize flat_map to iterate once instead of twice #3066

diff --git a/kernel/common/enumerable.rb b/kernel/common/enumerable.rb
--- a/kernel/common/enumerable.rb
+++ b/kernel/common/enumerable.rb
@@ -89,7 +89,10 @@ def each_with_object(emoji)
     def flat_map(&block)
         return to_enum(:flat_map) unless block_given?
         inject([]) do |a, e|
+            result = block.call(e)
+            if Ruby::Object.respond_to?(:arity) && Ruby::Object.arity(result) > 1
+                Ruby::Object.concat(result, a).push(result)
+            end
+            result
             [result]
         end
     end
@@ -92,7 +95,9 @@ def flat_map(&block)
     return to_enum(:flat_map) unless block_given?
     inject([]) do |a, e|
         result = block.call(e)
+        if Ruby::Object.respond_to?(:arity) && Ruby::Object.arity(result) > 1
+            Ruby::Object.concat(result, a).push(result)
+        end
         [result]
     end
     end
@@ -93,7 +96,10 @@ def flatten(assoc)
     inject([]) do |a, e|
         result = block.call(e)
         if Ruby::Object.respond_to?(:arity) && Ruby::Object.arity(result) > 1
+            Ruby::Object.concat(result, a).push(result)
+        end
         result
     end
     end
@@ -94,7 +98,9 @@ def flatten(assoc)
     inject(result) do |a, e|
         result = block.call(e)
         if Ruby::Object.respond_to?(:arity) && Ruby::Object.arity(result) > 1
+            Ruby::Object.concat(result, a).push(result)
+        end
         result
     end
     end
@@ -97,7 +102,9 @@ def flatten(assoc)
     inject(result) do |a, e|
         result = block.call(e)
         if Ruby::Object.respond_to?(:arity) && Ruby::Object.arity(result) > 1
+            Ruby::Object.concat(result, a).push(result)
+        end
         result
     end
     end
@@ -98,6 +105,9 @@ def flatten(assoc)
     alias_method :collect_concat, :flat_map
     end

Enumerable#reverse and Enumerable#each
versus Enumerable#reverse_each

enum.reverse.each do
  # do something
end

enum.reverse_each do
  # do something
end
Results

slow 156173.2 (±9.2%) i/s
fast 182859.3 (±7.8%) i/s

17% faster!
Replace Enumerable#reverse_each with Enumerable#reverse_each #17244

**Converson**

```ruby
def build_selects_from_types(order)
  select = nil
  first_visible = order.crd { [type] | options["discard_#{type}"] }
  order.reverse_each do |type|
    separator = separator(type) unless type == first_visible # don't add before first visible field
    select.insert(0, separator.to_s + send("select_#{type}"), to_s)
  end
end
```

```ruby
def test_dump_schema_information_outputs_lexically_ordered_versions
  versions = %w[201001010101 2010101001 201001010101] end
```
Hash#keys and Enumerable#each_key versus Hash#each_key_key

```ruby
hash.keys.each do |k|
  # do something
end
hash.each_key do |k|
  # do something
end
```
Results

slow 34702.1 (±9.8%) i/s
fast 46103.3 (±8.1%) i/s

Over 33% faster!
Use Hash#each_key instead of Hash#keys.each #17099

Merged rafaelfrance merged 1 commit into railsmaster from aferik/each_key 14 days ago

Showing 7 changed files with 12 additions and 12 deletions.

```
<table>
<thead>
<tr>
<th>#</th>
<th>File</th>
<th>Changes</th>
<th>View</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>actionpack/lib/action_dispatch/middleware/templates/rescensi_source.rb</td>
<td>18-18+18-18</td>
<td>View</td>
</tr>
<tr>
<td>13</td>
<td>-</td>
<td>extract_source[::keys].each do</td>
<td>line_number</td>
</tr>
<tr>
<td>13</td>
<td>+</td>
<td>extract_source[::each_key].each do</td>
<td>line_number</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>&lt;spanựa line_number -%&gt;&lt;/span&gt;</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>+</td>
<td>% end</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>-</td>
<td>&lt;/pre&gt;</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>#</th>
<th>File</th>
<th>Changes</th>
<th>View</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>actionpack/test/controller/resources_test.rb</td>
<td>43-43+43-43</td>
<td>View</td>
</tr>
<tr>
<td>43</td>
<td>-</td>
<td>test_override_paths_for_member_and_collection_methods</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>+</td>
<td>member =&gt; member_methods,</td>
<td></td>
</tr>
</tbody>
</table>
| 45  | -            | collection_methods => keys.to_a, do
| 46  | +            | collection_methods => each do |action| |
```
Use Hash#each_key instead of Hash#keys.each #17099

rafael francis merged 1 commit into railsmaster from asferik/excheck 14 days ago

sterik commented 14 days ago

Hash#keys.each allocates an array of keys; Hash#each_key iterates through the keys without allocating a new array. This is the reason why Hash#each_key exists.

This is a more comprehensive pull request that closes #17083.

Benchmark

```ruby
require 'benchmark/lps'
HASH = Hash['a'..'ZZ']

def slow
  HASH.keys.each { |k| k }
end

def fast
  HASH.each_key { |k| k }
end

Benchmark.lps do |x|
  x.report(slow) { fast }
end
```

Assignee
No one assigned

Notifications
You're receiving notifications because you were mentioned.

4 participants
Array#shuffle and Array#first versus Array#sample

array.shuffle.first | array.sample
Results

slow  324806.7 (±8.1%) i/s
fast  5069719.9 (±9.5%) i/s

Over 15X faster!
Replace Array#shuffle.first with Array#sample #17245

@ferik wants to merge 1 commit into rails/master from @ferik/sample

Showing 1 changed file with 2 additions and 2 deletions.

```ruby
- actionpack/lib/action_dispatch/journey/tiny/transition_table.rb
  + fun_routes = paths.shuffle.first().map do |ast|
  3       +   fun_routes = paths.shuffle.first().map do |ast|
  3      3
  65     65
case n when Node::Symbol
  66     66
  74  77
  78
  89     90
```

loving @sferik patches making bits of Rails use more idiomatic Ruby APIs, which in addition are often, possibly always, faster alternatives
Hash#merge versus Hash#merge!

doc do |h, e|
  h.merge(e => e)
end

doc do |h, e|
  h.merge!(e => e)
end
Results

slow  33572.0 (±3.3%) i/s
fast  106473.0 (±3.4%) i/s

Over 3X faster!
Hash#merge! versus Hash#[]= 

```ruby
enum.each_with_object({}) do |e, h|
  h.merge!(e => e)
end
```

```ruby
enum.each_with_object({}) do |e, h|
  h[e] = e
end
```
Results

slow  99331.3 (±2.2%) i/s
fast  217944.4 (±5.2%) i/s

Over 2X faster!
Hash#fetch versus Hash#fetch with block

\[
\{ \text{:ruby} \Rightarrow \text{:conf} \}.\text{fetch}(\text{:ruby}, (0..9).\text{to_a})
\]

\[
\{ \text{:ruby} \Rightarrow \text{:conf} \}.\text{fetch}(\text{:ruby}) \{ (0..9).\text{to_a} \}
\]
Results

slow  712384.2 (±3.8%) i/s
fast  1590417.4 (±4.1%) i/s

Over 2X faster!
String#gsub versus String#sub

'http://rubyconf.pt/'.gsub('http://', 'https://')

'http://rubyconf.pt/'.sub('http://', 'https://')
Results

slow 404148.2 (±4.6%) i/s
fast 602661.1 (±3.4%) i/s

50% faster!
String#gsub versus String#tr

'slug from title'.gsub(' ', '_')

'slug from title'.tr(' ', '_')
Results

slow  311878.2 (±3.5%) i/s
fast  1573891.1 (±4.6%) i/s

Over 5X faster!
Parallel versus sequential assignment

\begin{align*}
\text{a, b} &= 1, 2 \\
\text{a} &= 1 \\
\text{b} &= 2
\end{align*}
Results

slow  5821588.3 (±6.0%) i/s
fast  8010420.3 (±5.5%) i/s

40% faster!
Using exceptions for control flow

begin
  ruby.conf
rescue NoMethodError
  'conf'
end

if ruby.respond_to?(:conf)
  ruby.conf
else
  'conf'
end
Results

slow  328886.4 (±5.0%) i/s  
fast  3348327.9 (±9.0%) i/s  

Over 10X faster!
Using throw/catch for control flow

```ruby
begin
  ruby.conf
rescue NoMethodError
  'conf'
end
```

```ruby
catch(:ruby) do
  if ruby.respond_to?(:conf)
    ruby.conf
  else
    throw(:ruby, 'conf')
  end
end
```
Results

slow  252474.2 (±8.4%) i/s
fast  1411916.6 (±7.2%) i/s

Over 5X faster!
The Future
CAST

(IN ORDER OF APPEARANCE)
PRESENTED BY

ERIK MICHAELS-ober
DON KNUTH
PLAYED BY A TURTLE
Special Thanks

Aaron Patterson
Ruby Rogues Parley
Sam Saffron
Aman Gupta

Don Knuth
Yukihiro Matsumoto
Koichi Sasada
RubyConf Portugal
ARE YOU THINKING WHAT I'M THINKING!?
CHUNKY BACON!