Ruby master - Bug #7158

require is slow in its bookkeeping; can make Rails startup 2.2x faster

10/14/2012 01:56 PM - gregprice (Greg Price)

Status:	Closed		
Priority:	Normal		
Assignee:	h.shirosaki (Hiroshi Shirosaki)		
Target version:	2.0.0		
ruby -v:	ruby 1.9.3p194 (2012-04-20 revision 35409) [i686-linux]	**Backport:**	

Description

=begin
Starting a large application in Ruby is slow. Most of the startup time is not spent in the actual work of loading files and running Ruby code, but in bookkeeping in the 'require' implementation. I've attached a patch series which makes that bookkeeping much faster. These patches speed up a large Rails application's startup by 2.2x, and a pure-'require' benchmark by 3.4x.

These patches fix two ways in which 'require' is slow. Both problems have been discussed before, but these patches solve the problems with less code and stricter compatibility than previous patches I've seen.

- Currently we iterate through $LOADED_FEATURES to see if anything matches the newly required feature. Further, each iteration iterates in turn through $LOAD_PATH. Xavier Shay spotted this problem last year and a series of patches were discussed (in Issue #3924) to add a Hash index alongside $LOADED_FEATURES, but for 1.9.3 none were merged; Masaya Tarui committed Revision r31875, which mitigated the problem. This series adds a Hash index, and keeps it up to date even if the user modifies $LOADED_FEATURES. This is worth a 40% speedup on one large Rails application, and 2.3x on a pure-'require' benchmark.

- Currently each 'require' call runs through $LOAD_PATH and calls rb_file_expand_path() on each element. Yura Sokolov (funny_falcon) proposed caching this last December in Issue #5767, but it wasn't merged. This series also caches $LOAD_PATH, and keeps the cache up to date with a different, less invasive technique. The cache takes 34 lines of code, and is worth an additional 57% speedup in starting a Rails app and a 46% speedup in pure 'require'.

== Staying Compatible

With both the $LOADED_FEATURES index and the $LOAD_PATH cache,

- we exactly preserve the semantics of the user modifying $LOAD_PATH or $LOADED_FEATURES;

- both $LOAD_PATH and $LOADED_FEATURES remain ordinary Arrays, with no singleton methods;

- we make just one semantic change: each element of $LOAD_PATH and $LOADED_FEATURES is made into a frozen string. This doesn't limit the flexibility Ruby offers to the programmer in any way; to alter an element of either array, one simply reassigns it to the new value. Further, normal path-munging code which only adds and removes elements shouldn't have to change at all.

These patches use the following technique to keep the cache and the

index up to date without modifying the methods of $LOADED_FEATURES or
$LOAD_PATH: we take advantage of the sharing mechanism in the Array
implementation to detect, in O(1) time, whether either array has been
mutated. We cause $LOADED_FEATURES to be shared with an Array we keep
privately in load.c; if anything modifies it, it will break the
sharing and we will know to rebuild the index. Similarly for
$LOAD_PATH.

== Benchmarks

First, on my company's Rails application, where $LOAD_PATH.size is 207
and $LOADED_FEATURES.size is 2126. I measured the time taken by
'bundle exec rails runner "p 1"'.

```
.               Rails startup time,
version              best of 5       speedup
v1_9_3_194            12.197s
v1_9_3_194+index        8.688s          1.40x
v1_9_3_194+index+cache  5.538s          2.20x
```

And now isolating the performance of 'require', by requiring
16000 empty files.

```
version            time, best of 5     speedup
trunk (at r36920)      10.115s
trunk+index             4.363s          2.32x
trunk+index+cache       2.984s          3.39x
```

(The timings for the Rails application are based on the latest release
rather than trunk because a number of gems failed to compile against
trunk for me.)

== The Patches

I've attached four patches:

(1) Patch 1 changes no behavior at all. It adds comments and
simplifies a bit of code to help in understanding why patch 3 is
correct. 42 lines, most of them comments.

(2) Patch 2 adds a function to array.c which will help us tell when
$LOAD_PATH or $LOADED_FEATURES has been modified. 17 lines.

(3) Patch 3 adds the $LOADED_FEATURES index. 150 lines.

(4) Patch 4 adds the $LOAD_PATH cache. 34 lines.

Reviews and comments welcome -- I'm sure there's something I could do
to make these patches better. I hope we can get some form of them
into trunk before the next release. My life has been happier since I
switched to this version because commands in my Rails application all
run faster now, and I want every Ruby programmer to be happier in the
same way with 2.0 and ideally with 1.9.4.

=end

Related issues:

Related to Ruby master - Bug #3924: Performance bug (in require?)	**Closed**	**10/10/2010**

Associated revisions

Revision 4d414c9f - 11/05/2012 03:24 PM - shirosaki

Clarify and explain loaded_feature_path and rb_feature_p

- load.c (loaded_feature_path): clarify and briefly comment
  function. These clarifications have no effect on the behavior
  of the function.

- load.c (rb_feature_p): explain the search loop. Especially
  useful because the logic is complicated as described in the
  second paragraph.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@37477 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 37477 - 11/05/2012 03:24 PM - shirosaki

Clarify and explain loaded_feature_path and rb_feature_p

- load.c (loaded_feature_path): clarify and briefly comment
  function. These clarifications have no effect on the behavior
  of the function.

- load.c (rb_feature_p): explain the search loop. Especially
  useful because the logic is complicated as described in the
  second paragraph.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37477 - 11/05/2012 03:24 PM - shirosaki

Clarify and explain loaded_feature_path and rb_feature_p

- load.c (loaded_feature_path): clarify and briefly comment
  function. These clarifications have no effect on the behavior
  of the function.

- load.c (rb_feature_p): explain the search loop. Especially
  useful because the logic is complicated as described in the
  second paragraph.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37477 - 11/05/2012 03:24 PM - shirosaki

Clarify and explain loaded_feature_path and rb_feature_p

- load.c (loaded_feature_path): clarify and briefly comment
  function. These clarifications have no effect on the behavior
  of the function.

- load.c (rb_feature_p): explain the search loop. Especially
  useful because the logic is complicated as described in the
  second paragraph.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37477 - 11/05/2012 03:24 PM - shirosaki

Clarify and explain loaded_feature_path and rb_feature_p

- load.c (loaded_feature_path): clarify and briefly comment
  function. These clarifications have no effect on the behavior
  of the function.

- load.c (rb_feature_p): explain the search loop. Especially
  useful because the logic is complicated as described in the
  second paragraph.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37477 - 11/05/2012 03:24 PM - shirosaki

Clarify and explain loaded_feature_path and rb_feature_p

- load.c (loaded_feature_path): clarify and briefly comment function. These clarifications have no effect on the behavior of the function.

- load.c (rb_feature_p): explain the search loop. Especially useful because the logic is complicated as described in the second paragraph.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37477 - 11/05/2012 03:24 PM - shirosaki

Clarify and explain loaded_feature_path and rb_feature_p

- load.c (loaded_feature_path): clarify and briefly comment function. These clarifications have no effect on the behavior of the function.

- load.c (rb_feature_p): explain the search loop. Especially useful because the logic is complicated as described in the second paragraph.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision e575070f - 11/05/2012 03:24 PM - shirosaki

Expose whether two arrays are shared

- array.c (rb_ary_shared_with_p): new function.
  Expose whether two arrays are shared (read-only, C only).

- include/ruby/intern.h (rb_ary_shared_with_p): declare.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@37478 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 37478 - 11/05/2012 03:24 PM - shirosaki

Expose whether two arrays are shared

- array.c (rb_ary_shared_with_p): new function.
  Expose whether two arrays are shared (read-only, C only).

- include/ruby/intern.h (rb_ary_shared_with_p): declare.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37478 - 11/05/2012 03:24 PM - shirosaki

Expose whether two arrays are shared

- array.c (rb_ary_shared_with_p): new function.
  Expose whether two arrays are shared (read-only, C only).

- include/ruby/intern.h (rb_ary_shared_with_p): declare.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37478 - 11/05/2012 03:24 PM - shirosaki

Expose whether two arrays are shared

- array.c (rb_ary_shared_with_p): new function.

Expose whether two arrays are shared (read-only, C only).

- include/ruby/intern.h (rb_ary_shared_with_p): declare.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37478 - 11/05/2012 03:24 PM - shirosaki

Expose whether two arrays are shared

- array.c (rb_ary_shared_with_p): new function.
  Expose whether two arrays are shared (read-only, C only).

- include/ruby/intern.h (rb_ary_shared_with_p): declare.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37478 - 11/05/2012 03:24 PM - shirosaki

Expose whether two arrays are shared

- array.c (rb_ary_shared_with_p): new function.
  Expose whether two arrays are shared (read-only, C only).

- include/ruby/intern.h (rb_ary_shared_with_p): declare.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37478 - 11/05/2012 03:24 PM - shirosaki

Expose whether two arrays are shared

- array.c (rb_ary_shared_with_p): new function.
  Expose whether two arrays are shared (read-only, C only).

- include/ruby/intern.h (rb_ary_shared_with_p): declare.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision b56a2afc - 11/05/2012 03:27 PM - shirosaki

Index $LOADED_FEATURES so that require isn't so slow

- load.c (rb_feature_p, rb_provide_feature): index $LOADED_FEATURES
  so that require isn't so slow.

- load.c (rb_provide_feature, get_loaded_features_index): ensure
  that $LOADED_FEATURES entries are frozen strings. The user
  must mutate $LOADED_FEATURES itself rather than its individual
  entries.

- load.c (reset_loaded_features_snapshot): add a new function to reset
  vm->loaded_features_snapshot.

- load.c (get_loaded_features_index_raw): add a new function to get
  the loaded-features index.

- load.c (features_index_add_single): add a new function to add to the
  loaded-features index a single feature.

- load.c (features_index_add): add a new function to add to the
  loaded-features index all the required entries for feature.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- include/ruby/intern.h (rb_hash_clear): declare function.
- hash.c (rb_hash_clear): make function non-static.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@37480 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 37480 - 11/05/2012 03:27 PM - shirosaki

Index $LOADED_FEATURES so that require isn't so slow

- load.c (rb_feature_p, rb_provide_feature): index $LOADED_FEATURES
  so that require isn't so slow.
- load.c (rb_provide_feature, get_loaded_features_index): ensure
  that $LOADED_FEATURES entries are frozen strings. The user
  must mutate $LOADED_FEATURES itself rather than its individual
  entries.
- load.c (reset_loaded_features_snapshot): add a new function to reset
  vm->loaded_features_snapshot.
- load.c (get_loaded_features_index_raw): add a new function to get
  the loaded-features index.
- load.c (features_index_add_single): add a new function to add to the
  loaded-features index a single feature.
- load.c (features_index_add): add a new function to add to the
  loaded-features index all the required entries for feature.
- vm_core.h (rb_vm_struct): add fields.
- vm.c (rb_vm_mark): mark new fields.
- include/ruby/intern.h (rb_hash_clear): declare function.
- hash.c (rb_hash_clear): make function non-static.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37480 - 11/05/2012 03:27 PM - shirosaki

Index $LOADED_FEATURES so that require isn't so slow

- load.c (rb_feature_p, rb_provide_feature): index $LOADED_FEATURES
  so that require isn't so slow.
- load.c (rb_provide_feature, get_loaded_features_index): ensure
  that $LOADED_FEATURES entries are frozen strings. The user
  must mutate $LOADED_FEATURES itself rather than its individual
  entries.
- load.c (reset_loaded_features_snapshot): add a new function to reset
  vm->loaded_features_snapshot.
- load.c (get_loaded_features_index_raw): add a new function to get
  the loaded-features index.
- load.c (features_index_add_single): add a new function to add to the
  loaded-features index a single feature.
- load.c (features_index_add): add a new function to add to the
  loaded-features index all the required entries for feature.
- vm_core.h (rb_vm_struct): add fields.
- vm.c (rb_vm_mark): mark new fields.
- include/ruby/intern.h (rb_hash_clear): declare function.

- hash.c (rb_hash_clear): make function non-static.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37480 - 11/05/2012 03:27 PM - shirosaki

Index $LOADED_FEATURES so that require isn't so slow

- load.c (rb_feature_p, rb_provide_feature): index $LOADED_FEATURES
  so that require isn't so slow.

- load.c (rb_provide_feature, get_loaded_features_index): ensure
  that $LOADED_FEATURES entries are frozen strings. The user
  must mutate $LOADED_FEATURES itself rather than its individual
  entries.

- load.c (reset_loaded_features_snapshot): add a new function to reset
  vm->loaded_features_snapshot.

- load.c (get_loaded_features_index_raw): add a new function to get
  the loaded-features index.

- load.c (features_index_add_single): add a new function to add to the
  loaded-features index a single feature.

- load.c (features_index_add): add a new function to add to the
  loaded-features index all the required entries for feature.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- include/ruby/intern.h (rb_hash_clear): declare function.

- hash.c (rb_hash_clear): make function non-static.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37480 - 11/05/2012 03:27 PM - shirosaki

Index $LOADED_FEATURES so that require isn't so slow

- load.c (rb_feature_p, rb_provide_feature): index $LOADED_FEATURES
  so that require isn't so slow.

- load.c (rb_provide_feature, get_loaded_features_index): ensure
  that $LOADED_FEATURES entries are frozen strings. The user
  must mutate $LOADED_FEATURES itself rather than its individual
  entries.

- load.c (reset_loaded_features_snapshot): add a new function to reset
  vm->loaded_features_snapshot.

- load.c (get_loaded_features_index_raw): add a new function to get
  the loaded-features index.

- load.c (features_index_add_single): add a new function to add to the
  loaded-features index a single feature.

- load.c (features_index_add): add a new function to add to the
  loaded-features index all the required entries for feature.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- include/ruby/intern.h (rb_hash_clear): declare function.

- hash.c (rb_hash_clear): make function non-static.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37480 - 11/05/2012 03:27 PM - shirosaki

Index $LOADED_FEATURES so that require isn't so slow

- load.c (rb_feature_p, rb_provide_feature): index $LOADED_FEATURES
  so that require isn't so slow.

- load.c (rb_provide_feature, get_loaded_features_index): ensure
  that $LOADED_FEATURES entries are frozen strings. The user
  must mutate $LOADED_FEATURES itself rather than its individual
  entries.

- load.c (reset_loaded_features_snapshot): add a new function to reset
  vm->loaded_features_snapshot.

- load.c (get_loaded_features_index_raw): add a new function to get
  the loaded-features index.

- load.c (features_index_add_single): add a new function to add to the
  loaded-features index a single feature.

- load.c (features_index_add): add a new function to add to the
  loaded-features index all the required entries for feature.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- include/ruby/intern.h (rb_hash_clear): declare function.

- hash.c (rb_hash_clear): make function non-static.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37480 - 11/05/2012 03:27 PM - shirosaki

Index $LOADED_FEATURES so that require isn't so slow

- load.c (rb_feature_p, rb_provide_feature): index $LOADED_FEATURES
  so that require isn't so slow.

- load.c (rb_provide_feature, get_loaded_features_index): ensure
  that $LOADED_FEATURES entries are frozen strings. The user
  must mutate $LOADED_FEATURES itself rather than its individual
  entries.

- load.c (reset_loaded_features_snapshot): add a new function to reset
  vm->loaded_features_snapshot.

- load.c (get_loaded_features_index_raw): add a new function to get
  the loaded-features index.

- load.c (features_index_add_single): add a new function to add to the
  loaded-features index a single feature.

- load.c (features_index_add): add a new function to add to the
  loaded-features index all the required entries for feature.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- include/ruby/intern.h (rb_hash_clear): declare function.

- hash.c (rb_hash_clear): make function non-static.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 9823c461 - 11/05/2012 03:27 PM - shirosaki

Cache the expanded load path

- load.c (rb_get_expanded_load_path): cache the expanded load
  path. This saves 4KB of allocation and some stats for every
  element of the load path (so nearly a MB in my Rails app)
  on every require.

- load.c (rb_construct_expanded_load_path): ensure that $LOAD_PATH
  entries are frozen strings. The user must mutate $LOAD_PATH
  itself rather than its individual entries.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- ruby.c (process_options): modify $LOAD_PATH directly rather than
  its elements.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@37481 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 37481 - 11/05/2012 03:27 PM - shirosaki

Cache the expanded load path

- load.c (rb_get_expanded_load_path): cache the expanded load
  path. This saves 4KB of allocation and some stats for every
  element of the load path (so nearly a MB in my Rails app)
  on every require.

- load.c (rb_construct_expanded_load_path): ensure that $LOAD_PATH
  entries are frozen strings. The user must mutate $LOAD_PATH
  itself rather than its individual entries.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- ruby.c (process_options): modify $LOAD_PATH directly rather than
  its elements.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37481 - 11/05/2012 03:27 PM - shirosaki

Cache the expanded load path

- load.c (rb_get_expanded_load_path): cache the expanded load
  path. This saves 4KB of allocation and some stats for every
  element of the load path (so nearly a MB in my Rails app)
  on every require.

- load.c (rb_construct_expanded_load_path): ensure that $LOAD_PATH
  entries are frozen strings. The user must mutate $LOAD_PATH
  itself rather than its individual entries.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- ruby.c (process_options): modify $LOAD_PATH directly rather than
  its elements.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37481 - 11/05/2012 03:27 PM - shirosaki

Cache the expanded load path

- load.c (rb_get_expanded_load_path): cache the expanded load path. This saves 4KB of allocation and some stats for every element of the load path (so nearly a MB in my Rails app) on every require.

- load.c (rb_construct_expanded_load_path): ensure that $LOAD_PATH entries are frozen strings. The user must mutate $LOAD_PATH itself rather than its individual entries.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- ruby.c (process_options): modify $LOAD_PATH directly rather than its elements.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37481 - 11/05/2012 03:27 PM - shirosaki

Cache the expanded load path

- load.c (rb_get_expanded_load_path): cache the expanded load path. This saves 4KB of allocation and some stats for every element of the load path (so nearly a MB in my Rails app) on every require.

- load.c (rb_construct_expanded_load_path): ensure that $LOAD_PATH entries are frozen strings. The user must mutate $LOAD_PATH itself rather than its individual entries.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- ruby.c (process_options): modify $LOAD_PATH directly rather than its elements.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37481 - 11/05/2012 03:27 PM - shirosaki

Cache the expanded load path

- load.c (rb_get_expanded_load_path): cache the expanded load path. This saves 4KB of allocation and some stats for every element of the load path (so nearly a MB in my Rails app) on every require.

- load.c (rb_construct_expanded_load_path): ensure that $LOAD_PATH entries are frozen strings. The user must mutate $LOAD_PATH itself rather than its individual entries.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- ruby.c (process_options): modify $LOAD_PATH directly rather than its elements.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision 37481 - 11/05/2012 03:27 PM - shirosaki

Cache the expanded load path

- load.c (rb_get_expanded_load_path): cache the expanded load path. This saves 4KB of allocation and some stats for every element of the load path (so nearly a MB in my Rails app) on every require.

- load.c (rb_construct_expanded_load_path): ensure that $LOAD_PATH
  entries are frozen strings. The user must mutate $LOAD_PATH
  itself rather than its individual entries.

- vm_core.h (rb_vm_struct): add fields.

- vm.c (rb_vm_mark): mark new fields.

- ruby.c (process_options): modify $LOAD_PATH directly rather than
  its elements.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

Revision d33b9a8a - 11/05/2012 03:27 PM - shirosaki

Fix compatibility of cached expanded load path

- file.c (rb_get_path_check_to_string): extract from
  rb_get_path_check(). We change the spec not to call to_path of
  String object.

- file.c (rb_get_path_check_convert): extract from rb_get_path_check().

- file.c (rb_get_path_check): follow the above change.

- file.c (rb_file_expand_path_fast): remove check_expand_path_args().
  Instead we call it in load.c.

- file.c (rb_find_file_ext_safe): use rb_get_expanded_load_path() to
  reduce expand cost.

- file.c (rb_find_file_safe): ditto.

- internal.h (rb_get_expanded_load_path): add a declaration.

- internal.h (rb_get_path_check_to_string, rb_get_path_check_convert):
  add declarations.

- load.c (rb_construct_expanded_load_path): fix for compatibility.
  Same checks in rb_get_path_check() are added. We don't replace
  $LOAD_PATH and ensure that String object of $LOAD_PATH are frozen.
  We don't freeze non String object and expand it every times. We add
  arguments for expanding load path partially and checking if load path
  have relative paths or non String objects.

- load.c (load_path_getcwd): get current working directory for checking
  if it's changed when getting load path.

- load.c (rb_get_expanded_load_path): fix for rebuilding cache properly.
  We check if current working directory is changed and rebuild expanded
  load path cache. We expand paths which start with ~ (User HOME) and
  non String objects every times for compatibility. We make this
  accessible from other source files.

- load.c (rb_feature_provided): call rb_get_path() since we changed
  rb_file_expand_path_fast() not to call it.

- load.c (Init_load): initialize vm->load_path_check_cache.

- vm.c (rb_vm_mark): mark vm->load_path_check_cache for GC.

- vm_core.h (rb_vm_struct): add vm->load_path_check_cache to store data
  to check load path cache validity.

- test/ruby/test_require.rb (TestRequire): add tests for require
  compatibility related to cached expanded load path.
  [ruby-core:47970] [Bug #7158]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@37482 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 37482 - 11/05/2012 03:27 PM - shirosaki

Fix compatibility of cached expanded load path

- file.c (rb_get_path_check_to_string): extract from rb_get_path_check(). We change the spec not to call to_path of String object.
- file.c (rb_get_path_check_convert): extract from rb_get_path_check().
- file.c (rb_get_path_check): follow the above change.
- file.c (rb_file_expand_path_fast): remove check_expand_path_args(). Instead we call it in load.c.
- file.c (rb_find_file_ext_safe): use rb_get_expanded_load_path() to reduce expand cost.
- file.c (rb_find_file_safe): ditto.
- internal.h (rb_get_expanded_load_path): add a declaration.
- internal.h (rb_get_path_check_to_string, rb_get_path_check_convert): add declarations.
- load.c (rb_construct_expanded_load_path): fix for compatibility. Same checks in rb_get_path_check() are added. We don't replace $LOAD_PATH and ensure that String object of $LOAD_PATH are frozen. We don't freeze non String object and expand it every times. We add arguments for expanding load path partially and checking if load path have relative paths or non String objects.
- load.c (load_path_getcwd): get current working directory for checking if it's changed when getting load path.
- load.c (rb_get_expanded_load_path): fix for rebuilding cache properly. We check if current working directory is changed and rebuild expanded load path cache. We expand paths which start with ~ (User HOME) and non String objects every times for compatibility. We make this accessible from other source files.
- load.c (rb_feature_provided): call rb_get_path() since we changed rb_file_expand_path_fast() not to call it.
- load.c (Init_load): initialize vm->load_path_check_cache.
- vm.c (rb_vm_mark): mark vm->load_path_check_cache for GC.
- vm_core.h (rb_vm_struct): add vm->load_path_check_cache to store data to check load path cache validity.
- test/ruby/test_require.rb (TestRequire): add tests for require compatibility related to cached expanded load path. [ruby-core:47970] [Bug #7158]

Revision 37482 - 11/05/2012 03:27 PM - shirosaki

Fix compatibility of cached expanded load path

- file.c (rb_get_path_check_to_string): extract from rb_get_path_check(). We change the spec not to call to_path of String object.
- file.c (rb_get_path_check_convert): extract from rb_get_path_check().
- file.c (rb_get_path_check): follow the above change.
- file.c (rb_file_expand_path_fast): remove check_expand_path_args(). Instead we call it in load.c.
- file.c (rb_find_file_ext_safe): use rb_get_expanded_load_path() to reduce expand cost.

- file.c (rb_find_file_safe): ditto.

- internal.h (rb_get_expanded_load_path): add a declaration.

- internal.h (rb_get_path_check_to_string, rb_get_path_check_convert): add declarations.

- load.c (rb_construct_expanded_load_path): fix for compatibility. Same checks in rb_get_path_check() are added. We don't replace $LOAD_PATH and ensure that String object of $LOAD_PATH are frozen. We don't freeze non String object and expand it every times. We add arguments for expanding load path partially and checking if load path have relative paths or non String objects.

- load.c (load_path_getcwd): get current working directory for checking if it's changed when getting load path.

- load.c (rb_get_expanded_load_path): fix for rebuilding cache properly. We check if current working directory is changed and rebuild expanded load path cache. We expand paths which start with ~ (User HOME) and non String objects every times for compatibility. We make this accessible from other source files.

- load.c (rb_feature_provided): call rb_get_path() since we changed rb_file_expand_path_fast() not to call it.

- load.c (Init_load): initialize vm->load_path_check_cache.

- vm.c (rb_vm_mark): mark vm->load_path_check_cache for GC.

- vm_core.h (rb_vm_struct): add vm->load_path_check_cache to store data to check load path cache validity.

- test/ruby/test_require.rb (TestRequire): add tests for require compatibility related to cached expanded load path. [ruby-core:47970] [Bug #7158]

Revision 37482 - 11/05/2012 03:27 PM - shirosaki

Fix compatibility of cached expanded load path

- file.c (rb_get_path_check_to_string): extract from rb_get_path_check(). We change the spec not to call to_path of String object.

- file.c (rb_get_path_check_convert): extract from rb_get_path_check().

- file.c (rb_get_path_check): follow the above change.

- file.c (rb_file_expand_path_fast): remove check_expand_path_args(). Instead we call it in load.c.

- file.c (rb_find_file_ext_safe): use rb_get_expanded_load_path() to reduce expand cost.

- file.c (rb_find_file_safe): ditto.

- internal.h (rb_get_expanded_load_path): add a declaration.

- internal.h (rb_get_path_check_to_string, rb_get_path_check_convert): add declarations.

- load.c (rb_construct_expanded_load_path): fix for compatibility. Same checks in rb_get_path_check() are added. We don't replace $LOAD_PATH and ensure that String object of $LOAD_PATH are frozen. We don't freeze non String object and expand it every times. We add arguments for expanding load path partially and checking if load path have relative paths or non String objects.

- load.c (load_path_getcwd): get current working directory for checking if it's changed when getting load path.

- load.c (rb_get_expanded_load_path): fix for rebuilding cache properly.

We check if current working directory is changed and rebuild expanded load path cache. We expand paths which start with ~ (User HOME) and non String objects every times for compatibility. We make this accessible from other source files.

- load.c (rb_feature_provided): call rb_get_path() since we changed rb_file_expand_path_fast() not to call it.

- load.c (Init_load): initialize vm->load_path_check_cache.

- vm.c (rb_vm_mark): mark vm->load_path_check_cache for GC.

- vm_core.h (rb_vm_struct): add vm->load_path_check_cache to store data to check load path cache validity.

- test/ruby/test_require.rb (TestRequire): add tests for require compatibility related to cached expanded load path.
[ruby-core:47970] [Bug #7158]

Revision 37482 - 11/05/2012 03:27 PM - shirosaki

Fix compatibility of cached expanded load path

- file.c (rb_get_path_check_to_string): extract from rb_get_path_check(). We change the spec not to call to_path of String object.

- file.c (rb_get_path_check_convert): extract from rb_get_path_check().

- file.c (rb_get_path_check): follow the above change.

- file.c (rb_file_expand_path_fast): remove check_expand_path_args(). Instead we call it in load.c.

- file.c (rb_find_file_ext_safe): use rb_get_expanded_load_path() to reduce expand cost.

- file.c (rb_find_file_safe): ditto.

- internal.h (rb_get_expanded_load_path): add a declaration.

- internal.h (rb_get_path_check_to_string, rb_get_path_check_convert): add declarations.

- load.c (rb_construct_expanded_load_path): fix for compatibility. Same checks in rb_get_path_check() are added. We don't replace $LOAD_PATH and ensure that String object of $LOAD_PATH are frozen. We don't freeze non String object and expand it every times. We add arguments for expanding load path partially and checking if load path have relative paths or non String objects.

- load.c (load_path_getcwd): get current working directory for checking if it's changed when getting load path.

- load.c (rb_get_expanded_load_path): fix for rebuilding cache properly. We check if current working directory is changed and rebuild expanded load path cache. We expand paths which start with ~ (User HOME) and non String objects every times for compatibility. We make this accessible from other source files.

- load.c (rb_feature_provided): call rb_get_path() since we changed rb_file_expand_path_fast() not to call it.

- load.c (Init_load): initialize vm->load_path_check_cache.

- vm.c (rb_vm_mark): mark vm->load_path_check_cache for GC.

- vm_core.h (rb_vm_struct): add vm->load_path_check_cache to store data to check load path cache validity.

- test/ruby/test_require.rb (TestRequire): add tests for require compatibility related to cached expanded load path.
[ruby-core:47970] [Bug #7158]

Revision 37482 - 11/05/2012 03:27 PM - shirosaki

Fix compatibility of cached expanded load path

- file.c (rb_get_path_check_to_string): extract from
  rb_get_path_check(). We change the spec not to call to_path of
  String object.

- file.c (rb_get_path_check_convert): extract from rb_get_path_check().

- file.c (rb_get_path_check): follow the above change.

- file.c (rb_file_expand_path_fast): remove check_expand_path_args().
  Instead we call it in load.c.

- file.c (rb_find_file_ext_safe): use rb_get_expanded_load_path() to
  reduce expand cost.

- file.c (rb_find_file_safe): ditto.

- internal.h (rb_get_expanded_load_path): add a declaration.

- internal.h (rb_get_path_check_to_string, rb_get_path_check_convert):
  add declarations.

- load.c (rb_construct_expanded_load_path): fix for compatibility.
  Same checks in rb_get_path_check() are added. We don't replace
  $LOAD_PATH and ensure that String object of $LOAD_PATH are frozen.
  We don't freeze non String object and expand it every times. We add
  arguments for expanding load path partially and checking if load path
  have relative paths or non String objects.

- load.c (load_path_getcwd): get current working directory for checking
  if it's changed when getting load path.

- load.c (rb_get_expanded_load_path): fix for rebuilding cache properly.
  We check if current working directory is changed and rebuild expanded
  load path cache. We expand paths which start with ~ (User HOME) and
  non String objects every times for compatibility. We make this
  accessible from other source files.

- load.c (rb_feature_provided): call rb_get_path() since we changed
  rb_file_expand_path_fast() not to call it.

- load.c (Init_load): initialize vm->load_path_check_cache.

- vm.c (rb_vm_mark): mark vm->load_path_check_cache for GC.

- vm_core.h (rb_vm_struct): add vm->load_path_check_cache to store data
  to check load path cache validity.

- test/ruby/test_require.rb (TestRequire): add tests for require
  compatibility related to cached expanded load path.
  [ruby-core:47970] [Bug #7158]

Revision 37482 - 11/05/2012 03:27 PM - shirosaki

Fix compatibility of cached expanded load path

- file.c (rb_get_path_check_to_string): extract from
  rb_get_path_check(). We change the spec not to call to_path of
  String object.

- file.c (rb_get_path_check_convert): extract from rb_get_path_check().

- file.c (rb_get_path_check): follow the above change.

- file.c (rb_file_expand_path_fast): remove check_expand_path_args().
  Instead we call it in load.c.

- file.c (rb_find_file_ext_safe): use rb_get_expanded_load_path() to

reduce expand cost.

- file.c (rb_find_file_safe): ditto.
- internal.h (rb_get_expanded_load_path): add a declaration.
- internal.h (rb_get_path_check_to_string, rb_get_path_check_convert): add declarations.
- load.c (rb_construct_expanded_load_path): fix for compatibility. Same checks in rb_get_path_check() are added. We don't replace $LOAD_PATH and ensure that String object of $LOAD_PATH are frozen. We don't freeze non String object and expand it every times. We add arguments for expanding load path partially and checking if load path have relative paths or non String objects.
- load.c (load_path_getcwd): get current working directory for checking if it's changed when getting load path.
- load.c (rb_get_expanded_load_path): fix for rebuilding cache properly. We check if current working directory is changed and rebuild expanded load path cache. We expand paths which start with ~ (User HOME) and non String objects every times for compatibility. We make this accessible from other source files.
- load.c (rb_feature_provided): call rb_get_path() since we changed rb_file_expand_path_fast() not to call it.
- load.c (Init_load): initialize vm->load_path_check_cache.
- vm.c (rb_vm_mark): mark vm->load_path_check_cache for GC.
- vm_core.h (rb_vm_struct): add vm->load_path_check_cache to store data to check load path cache validity.
- test/ruby/test_require.rb (TestRequire): add tests for require compatibility related to cached expanded load path.
  [ruby-core:47970] [Bug #7158]

History

#1 - 10/14/2012 02:41 PM - gregprice (Greg Price)

=begin
I've also made these patches available at https://github.com/gnprice/ruby , in the "fast-require" branch against a recent trunk and in "fast-require-1.9.3p194" against the latest release.

If you use (()) and its (()) plugin, one convenient way to try out the patched version is to create a file "fast-require" containing the one line

install_git 1.9.3-p194-price-fast-require https://github.com/gnprice/ruby fast-require-1.9.3p194 autoconf standard

and run "rbenv install /path/to/fast-require". Then "rbenv shell fast-require" will select the new version in the current shell, and "rbenv global fast-require" will select it globally.

=end

#2 - 10/14/2012 10:52 PM - trans (Thomas Sawyer)

I believe a great deal of additional speed could be gained by optimizing #require_relative (and making use of it, of course). From what I understand, #require_relative ends up calling ordinary #require code, which is inefficient since #require_relative already knows which path to find the script, so why have require search the $LOAD_PATH for it?

In all these efforts to speed up load time it would be nice to see someone tackle this issue as well.

#3 - 10/16/2012 01:21 PM - gregprice (Greg Price)

trans (Thomas Sawyer) wrote:

> I believe a great deal of additional speed could be gained by optimizing #require_relative (and making use of it, of course).

I'd like to keep this thread focused on speeding up existing code which uses #require. If you're interested in making changes to #require_relative (which is a function that is not involved in the startup time of most existing libraries or applications), a separate issue would be the best place to discuss that.

> From what I understand, #require_relative ends up calling ordinary
> #require code, which is inefficient since #require_relative already
> knows which path to find the script, so why have require search
> the $LOAD_PATH for it?

Note that most of the time #require spends is not in searching $LOAD_PATH -- it's in deciding whether the requested library needs to be loaded at all, or refers to a file that has already been loaded. That's what this patch series addresses. (Even the expanded $LOAD_PATH, which this Patch 4 caches, is used in making that decision before it is used to search to find the script.)

Greg

#4 - 10/16/2012 10:29 PM - wycats (Yehuda Katz)

Yehuda Katz
(ph) 718.877.1325

On Tue, Oct 16, 2012 at 12:21 AM, gregprice (Greg Price) price@mit.eduwrote:

> Issue #7158 has been updated by gregprice (Greg Price).
>
> trans (Thomas Sawyer) wrote:
>
> > I believe a great deal of additional speed could be gained by optimizing
> > #require_relative (and making use of it, of course).
>
> I'd like to keep this thread focused on speeding up existing code which
> uses #require. If you're interested in making changes to #require_relative
> (which is a function that is not involved in the startup time of most
> existing libraries or applications), a separate issue would be the best
> place to discuss that.

I would agree. Also, I consider require_relative an antipattern, so I hope
people don't start insisting that libraries use it in order to speed things
up instead of just speeding up require.

> > From what I understand, #require_relative ends up calling ordinary
> > #require code, which is inefficient since #require_relative already
> > knows which path to find the script, so why have require search
> > the $LOAD_PATH for it?
>
> Note that most of the time #require spends is not in searching $LOAD_PATH
> -- it's in deciding whether the requested library needs to be loaded at
> all, or refers to a file that has already been loaded. That's what this
> patch series addresses. (Even the expanded $LOAD_PATH, which this Patch 4
> caches, is used in making that decision before it is used to search to find
> the script.)
>
> Greg
>
> ---
>
> Bug #7158: require is slow in its bookkeeping; can make Rails startup 2.2x
> faster
> https://bugs.ruby-lang.org/issues/7158#change-30820
>
> Author: gregprice (Greg Price)
> Status: Open
> Priority: Normal
> Assignee:
> Category: core
> Target version:
> ruby -v: ruby 1.9.3p194 (2012-04-20 revision 35409) [i686-linux]
>
> =begin
> Starting a large application in Ruby is slow. Most of the startup
> time is not spent in the actual work of loading files and running Ruby
> code, but in bookkeeping in the 'require' implementation. I've
> attached a patch series which makes that bookkeeping much faster.
> These patches speed up a large Rails application's startup by 2.2x,
> and a pure-'require' benchmark by 3.4x.

These patches fix two ways in which 'require' is slow. Both problems
have been discussed before, but these patches solve the problems with
less code and stricter compatibility than previous patches I've seen.

- Currently we iterate through $LOADED_FEATURES to see if anything
  matches the newly required feature. Further, each iteration
  iterates in turn through $LOAD_PATH. Xavier Shay spotted this
  problem last year and a series of patches were discussed
  (in Issue #3924) to add a Hash index alongside $LOADED_FEATURES,
  but for 1.9.3 none were merged; Masaya Tarui committed Revision r31875,
  which mitigated the problem. This series adds a Hash index,
  and keeps it up to date even if the user modifies $LOADED_FEATURES.
  This is worth a 40% speedup on one large Rails application,
  and 2.3x on a pure-'require' benchmark.

- Currently each 'require' call runs through $LOAD_PATH and calls
  rb_file_expand_path() on each element. Yura Sokolov (funny_falcon)
  proposed caching this last December in Issue #5767, but it wasn't
  merged. This series also caches $LOAD_PATH, and keeps the cache up
  to date with a different, less invasive technique. The cache takes
  34 lines of code, and is worth an additional 57% speedup in
  starting a Rails app and a 46% speedup in pure 'require'.

== Staying Compatible

With both the $LOADED_FEATURES index and the $LOAD_PATH cache,

- we exactly preserve the semantics of the user modifying $LOAD_PATH
  or $LOADED_FEATURES;

- both $LOAD_PATH and $LOADED_FEATURES remain ordinary Arrays, with
  no singleton methods;

- we make just one semantic change: each element of $LOAD_PATH and
  $LOADED_FEATURES is made into a frozen string. This doesn't limit
  the flexibility Ruby offers to the programmer in any way; to alter
  an element of either array, one simply reassigns it to the new
  value. Further, normal path-munging code which only adds and
  removes elements shouldn't have to change at all.

These patches use the following technique to keep the cache and the
index up to date without modifying the methods of $LOADED_FEATURES or
$LOAD_PATH: we take advantage of the sharing mechanism in the Array
implementation to detect, in O(1) time, whether either array has been
mutated. We cause $LOADED_FEATURES to be shared with an Array we keep
privately in load.c; if anything modifies it, it will break the
sharing and we will know to rebuild the index. Similarly for
$LOAD_PATH.

== Benchmarks

First, on my company's Rails application, where $LOAD_PATH.size is 207
and $LOADED_FEATURES.size is 2126. I measured the time taken by
'bundle exec rails runner "p 1"'.

.	Rails startup time,	
version	best of 5	speedup
v1_9_3_194	12.197s	
v1_9_3_194+index	8.688s	1.40x
v1_9_3_194+index+cache	5.538s	2.20x

And now isolating the performance of 'require', by requiring
16000 empty files.

version	time, best of 5	speedup
trunk (at r36920)	10.115s	
trunk+index	4.363s	2.32x
trunk+index+cache	2.984s	3.39x

(The timings for the Rails application are based on the latest release
rather than trunk because a number of gems failed to compile against
trunk for me.)

== The Patches

I've attached four patches:

(1) Patch 1 changes no behavior at all. It adds comments and
simplifies a bit of code to help in understanding why patch 3 is
correct. 42 lines, most of them comments.

(2) Patch 2 adds a function to array.c which will help us tell when
$LOAD_PATH or $LOADED_FEATURES has been modified. 17 lines.

(3) Patch 3 adds the $LOADED_FEATURES index. 150 lines.

(4) Patch 4 adds the $LOAD_PATH cache. 34 lines.

Reviews and comments welcome -- I'm sure there's something I could do
to make these patches better. I hope we can get some form of them
into trunk before the next release. My life has been happier since I
switched to this version because commands in my Rails application all
run faster now, and I want every Ruby programmer to be happier in the
same way with 2.0 and ideally with 1.9.4.

=end

--
http://bugs.ruby-lang.org/

#5 - 10/24/2012 09:00 PM - usa (Usaku NAKAMURA)

- Status changed from Open to Assigned

- Assignee set to ko1 (Koichi Sasada)

#6 - 10/26/2012 08:10 PM - ko1 (Koichi Sasada)

I heard that taru-san's analysis of for your patch (1) to (4).

(1) to (3) are good (no compatibility issue). "freezing" strings are acceptable (no impact, I think). I want to introduce them.

However, (4) has compatibility issue because the patch assume that the current working directory does not changed. Please correct me if our understanding is wrong.

#7 - 10/27/2012 12:43 AM - funny_falcon (Yura Sokolov)

Checking for "shares same array" is a great catch :) I like it.

But my patch #5767 for $LOAD_PATH so "invasive" cause there is markable benefits from overriding "$:.push", so that whole $LOAD_PATH will not be re-scaped after each $: << gem_path .
Also, which way do you handle changing of current directory? Cache should be invalidated on Dir.chdir .
Also, which way do you handle changing filesystem encoding? One test from 'make check' fails, if you do not invalidate cache on changing filesystem encoding.

load_features_index will be recalculated after each require for new feature also? It is very strange, that you patch for $LOADED_FEATURES gains so much speedup. My patch #5427 gains only 6%, not 40%, and I doubt that hash lookup on ~2000 elements so much faster than crafted binary search (As well as you, I compare only "basename" of a feature). And I have no additional objects: no hashes, no arrays, no substrings, only $LOADED_FEATURES itself sorted in a rignt way.
... but maybe I'm mistacken.

About numbers: speedup 1.40x - is 29% speedup, and 2.20x is 55% speedup. My patches combined gains same 55% speedup as yours.

Can you test with "cache" patch but without "index" patch also?

#8 - 10/27/2012 11:40 PM - h.shirosaki (Hiroshi Shirosaki)

Indeed in Greg's patch (4) load path cache remains when current working directory was changed. But so far I cannot find any test cases which fail with Greg's patch.
In rb_find_file_safe() in file.c load path is expanded without cache, so cached load path in rb_feature_p() might be recovered by rb_find_file_safe().

I tried to create patches for current working directory change using same approach as Yura Sokolov's #5767.

https://gist.github.com/3964679

These patches are applied on the Greg's patches.

Patch 1 expands relative load path always. (If load path has many relative path, require slows down a lot.)
Patch 2 caches expanded relative load path and invalidate cache if current working directory is changed.
Patch 3 uses cached expanded load path in rb_find_file_ext_safe() and rb_find_file_safe().

From my benchmark, performance is almost same as Greg's. Benchmark is in the gist.
make test, test-all, test-rubyspec seem fine.

#9 - 10/28/2012 04:29 AM - ko1 (Koichi Sasada)

Hi,

(2012/10/27 23:40), h.shirosaki (Hiroshi Shirosaki) wrote:

> Indeed in Greg's patch (4) load path cache remains when current working directory was changed. But so far I cannot find any test cases which fail with Greg's patch.
> In rb_find_file_safe() in file.c load path is expanded without cache, so cached load path in rb_feature_p() might be recovered by rb_find_file_safe().
>
> I tried to create patches for current working directory change using same approach as Yura Sokolov's #5767.
>
> https://gist.github.com/3964679
>
> These patches are applied on the Greg's patches.
>
> Patch 1 expands relative load path always. (If load path has many relative path, require slows down a lot.)
> Patch 2 caches expanded relative load path and invalidate cache if current working directory is changed.
> Patch 3 uses cached expanded load path in rb_find_file_ext_safe() and rb_find_file_safe().
>
> From my benchmark, performance is almost same as Greg's. Benchmark is in the gist.
> make test, test-all, test-rubyspec seem fine.

Okay, there are patch of Gregs (1) to (4) and yours (1) to (3). Let's
say (G1) to (G4) and (S1) to (S3) respectively.

Could you give us the explanation of compatibility impact?
I doubt that current test cases have enough tests.

BTW, on your benchmark results, I can't find any performance improvement
with (S1) to (S3). What's the purpose?

--
// SASADA Koichi at atdot dot net

#10 - 10/28/2012 07:10 AM - h.shirosaki (Hiroshi Shirosaki)

ko1 (Koichi Sasada) wrote:

> Could you give us the explanation of compatibility impact?
> I doubt that current test cases have enough tests.
>
> BTW, on your benchmark results, I can't find any performance improvement
> with (S1) to (S3). What's the purpose?

Hi,

I think (S1) is more compatible trunk. It expands only relative load path always. Trunk expands all load path always.
If load path has many relative load path, (G1) to (G4) + (S1) performance is more close to (G1) to (G3) since relative paths are no cache.
I didn't show benchmark result of that.

(S2) is similar approach as Yura's #5767. It caches also relative load path. If current working directory was changed, it invalidates load path cache when getting expanded load path.
I can't find test cases which fail so far. Yura's patch is widely used (I'm also using it with 1.9.3) without issues. I think it's enough compatible.

I thought (S3) would give performance improvement in theory since it reduces expand costs to use expanded path. But my benchmark shows very small improvement. Yura's #5767 also uses this. Another benchmark might show more meaningful improvement result. If (S2) is enough compatible, (S3) would be also enough compatible because using same expanded load path cache.

#11 - 10/28/2012 07:53 AM - ko1 (Koichi Sasada)

(2012/10/28 7:10), h.shirosaki (Hiroshi Shirosaki) wrote:

> I think (S1) is more compatible trunk. It expands only relative load path always. Trunk expands all load path always.
> If load path has many relative load path, (G1) to (G4) + (S1) performance is more close to (G1) to (G3) since relative paths are no cache.

> I didn't show benchmark result of that.
>
> (S2) is similar approach as Yura's #5767. It caches also relative load path. If current working directory was changed, it invalidates load path cache when getting expanded load path.
> I can't find test cases which fail so far. Yura's patch is widely used (I'm also using it with 1.9.3) without issues. I think it's enough compatible.
>
> I thought (S3) would give performance improvement in theory since it reduces expand costs to use expanded path. But my benchmark shows very small improvement. Yura's #5767 also uses this. Another benchmark might show more meaningful improvement result. If (S2) is enough compatible, (S3) would be also enough compatible because using same expanded load path cache.

Thanks a lot for your explanation.

I'm very happy if you describe the "changed behavior/assumption".
Your explanation is "how to do it".
But we need to know "what changed".
Sorry if I misunderstood your explanation.

Or no behavior changed except "freeze $LOAD_PATH" by (G3)?
(Your words "enough compatible" meant that?)

Thanks,
Koichi

--
// SASADA Koichi at atdot dot net

#12 - 10/28/2012 08:23 AM - h.shirosaki (Hiroshi Shirosaki)

On Sun, Oct 28, 2012 at 7:36 AM, SASADA Koichi ko1@atdot.net wrote:

> I'm very happy if you describe the "changed behavior/assumption".
> Your explanation is "how to do it".
> But we need to know "what changed".
> Sorry if I misunderstood your explanation.
>
> Or no behavior changed except "freeze $LOAD_PATH" by (G3)?
> (Your words "enough compatible" meant that?)

Yes. I think no behavior changed except "freeze $LOAD_PATH" by (G3)
if I didn't overlook anything.

--
Hiroshi Shirosaki

#13 - 10/28/2012 08:53 AM - ko1 (Koichi Sasada)

(2012/10/28 8:03), Hiroshi Shirosaki wrote:

> Yes. I think no behavior changed except "freeze $LOAD_PATH" by (G3)
> if I didn't overlook anything.

Thank you for your clarification.

Maybe Tarui-san will review them.
Please wait for it.

Thanks,
Koichi

--
// SASADA Koichi at atdot dot net

#14 - 10/28/2012 10:29 AM - gregprice (Greg Price)

Sasada-san, thank you for the review. You're right, patch (4) should (and doesn't) invalidate the cache when the working directory changes. I believe Yura is right that it should also invalidate the cache when the filesystem encoding changes.

Yura, thanks for your existing patches and your detailed comments. At first look, I suspect that the reason a sorted-list approach to $LOADED_FEATURES was not a major speedup is that insertion is slow -- it takes time O(n), on the same order as the lookup takes in trunk. So it should still be helpful when one does many 'require' calls on the same files, but not as much as if insertion is also fast. With the hash-table-based index in (3), insertion takes time O(p), where p ~= 10 is the number of path components in the library filenames, and lookup is O(1) except in pathological cases.

I'm not too worried about recomputing the $LOAD_PATH cache from scratch after it's modified. For one thing, we do so only in an actual 'require' call, and it's only as expensive as the computation we do now in trunk on every 'require' call, so there's no case in which it should cause a regression. I suspect it's also the case that in common usage (with e.g. Bundler) most $LOAD_PATH modifications happen all at once with no intervening 'require' calls, and in that case we will only recompute it once for all of those changes. (But I haven't checked that that's the case.) So I think the trade-off of recomputing from scratch with less code complexity is a good one.

Shirosaki-san, thanks for your additional patches. I'm reading them now.

#15 - 10/29/2012 01:59 PM - funny_falcon (Yura Sokolov)

Which way insertion into sorted array takes same O(n) time? Do you mean
cost of memcpy after finding position for insert? Despite memcpy (which I
be leave cheap on this amounts) cost of insertion is O(log n).
Can you show result with caching LOAD_PATH and without indexing
LOADED_FEATURES? I believe that indexing will add small improvement after
caching.

28.10.2012 5:30 пользователь "gregprice (Greg Price)" price@mit.edu
написал:

> Issue #7158 has been updated by gregprice (Greg Price).
>
> Sasada-san, thank you for the review. You're right, patch (4) should
> (and doesn't) invalidate the cache when the working directory changes. I
> believe Yura is right that it should also invalidate the cache when the
> filesystem encoding changes.
>
> Yura, thanks for your existing patches and your detailed comments. At
> first look, I suspect that the reason a sorted-list approach to
> $LOADED_FEATURES was not a major speedup is that insertion is slow -- it
> takes time O(n), on the same order as the lookup takes in trunk. So it
> should still be helpful when one does many 'require' calls on the same
> files, but not as much as if insertion is also fast. With the
> hash-table-based index in (3), insertion takes time O(p), where p ~= 10 is
> the number of path components in the library filenames, and lookup is O(1)
> except in pathological cases.
>
> I'm not too worried about recomputing the $LOAD_PATH cache from scratch
> after it's modified. For one thing, we do so only in an actual 'require'
> call, and it's only as expensive as the computation we do now in trunk on
> every 'require' call, so there's no case in which it should cause a
> regression. I suspect it's also the case that in common usage (with e.g.
> Bundler) most $LOAD_PATH modifications happen all at once with no
> intervening 'require' calls, and in that case we will only recompute it
> once for all of those changes. (But I haven't checked that that's the
> case.) So I think the trade-off of recomputing from scratch with less code
> complexity is a good one.
>
> Shirosaki-san, thanks for your additional patches. I'm reading them now.
>
> ---
>
> Bug #7158: require is slow in its bookkeeping; can make Rails startup
> 2.2x faster
> https://bugs.ruby-lang.org/issues/7158#change-31844
>
> Author: gregprice (Greg Price)
> Status: Assigned
> Priority: Normal
> Assignee: ko1 (Koichi Sasada)
> Category: core
> Target version:
> ruby -v: ruby 1.9.3p194 (2012-04-20 revision 35409) [i686-linux]
>
> =begin
> Starting a large application in Ruby is slow. Most of the startup
> time is not spent in the actual work of loading files and running Ruby
> code, but in bookkeeping in the 'require' implementation. I've
> attached a patch series which makes that bookkeeping much faster.
> These patches speed up a large Rails application's startup by 2.2x,
> and a pure-'require' benchmark by 3.4x.
>
> These patches fix two ways in which 'require' is slow. Both problems
> have been discussed before, but these patches solve the problems with
> less code and stricter compatibility than previous patches I've seen.

- Currently we iterate through $LOADED_FEATURES to see if anything matches the newly required feature. Further, each iteration iterates in turn through $LOAD_PATH. Xavier Shay spotted this problem last year and a series of patches were discussed (in Issue #3924) to add a Hash index alongside $LOADED_FEATURES, but for 1.9.3 none were merged; Masaya Tarui committed Revision r31875, which mitigated the problem. This series adds a Hash index, and keeps it up to date even if the user modifies $LOADED_FEATURES. This is worth a 40% speedup on one large Rails application, and 2.3x on a pure-'require' benchmark.

- Currently each 'require' call runs through $LOAD_PATH and calls rb_file_expand_path() on each element. Yura Sokolov (funny_falcon) proposed caching this last December in Issue #5767, but it wasn't merged. This series also caches $LOAD_PATH, and keeps the cache up to date with a different, less invasive technique. The cache takes 34 lines of code, and is worth an additional 57% speedup in starting a Rails app and a 46% speedup in pure 'require'.

== Staying Compatible

With both the $LOADED_FEATURES index and the $LOAD_PATH cache,

- we exactly preserve the semantics of the user modifying $LOAD_PATH or $LOADED_FEATURES;

- both $LOAD_PATH and $LOADED_FEATURES remain ordinary Arrays, with no singleton methods;

- we make just one semantic change: each element of $LOAD_PATH and $LOADED_FEATURES is made into a frozen string. This doesn't limit the flexibility Ruby offers to the programmer in any way; to alter an element of either array, one simply reassigns it to the new value. Further, normal path-munging code which only adds and removes elements shouldn't have to change at all.

These patches use the following technique to keep the cache and the index up to date without modifying the methods of $LOADED_FEATURES or $LOAD_PATH: we take advantage of the sharing mechanism in the Array implementation to detect, in O(1) time, whether either array has been mutated. We cause $LOADED_FEATURES to be shared with an Array we keep privately in load.c; if anything modifies it, it will break the sharing and we will know to rebuild the index. Similarly for $LOAD_PATH.

== Benchmarks

First, on my company's Rails application, where $LOAD_PATH.size is 207 and $LOADED_FEATURES.size is 2126. I measured the time taken by 'bundle exec rails runner "p 1"'.

```
.               Rails startup time,
version              best of 5       speedup
v1_9_3_194            12.197s
v1_9_3_194+index       8.688s         1.40x
v1_9_3_194+index+cache  5.538s         2.20x
```

And now isolating the performance of 'require', by requiring 16000 empty files.

```
version            time, best of 5     speedup
trunk (at r36920)      10.115s
trunk+index            4.363s         2.32x
trunk+index+cache       2.984s         3.39x
```

(The timings for the Rails application are based on the latest release rather than trunk because a number of gems failed to compile against trunk for me.)

== The Patches

I've attached four patches:

> (1) Patch 1 changes no behavior at all. It adds comments and
> simplifies a bit of code to help in understanding why patch 3 is
> correct. 42 lines, most of them comments.
>
> (2) Patch 2 adds a function to array.c which will help us tell when
> $LOAD_PATH or $LOADED_FEATURES has been modified. 17 lines.
>
> (3) Patch 3 adds the $LOADED_FEATURES index. 150 lines.
>
> (4) Patch 4 adds the $LOAD_PATH cache. 34 lines.
>
> Reviews and comments welcome -- I'm sure there's something I could do
> to make these patches better. I hope we can get some form of them
> into trunk before the next release. My life has been happier since I
> switched to this version because commands in my Rails application all
> run faster now, and I want every Ruby programmer to be happier in the
> same way with 2.0 and ideally with 1.9.4.
>
> =end
>
> --
> http://bugs.ruby-lang.org/

#16 - 10/29/2012 09:57 PM - tarui (Masaya Tarui)

Hi,

I re-checked behavior, and found some compatibility issues.

first,
rb_file_expand_path_fast depends on not only current working directory but also home directories too.
it uses ENV["HOME"] and getpwnam (normally).

second,
if non String Object is included in LOAD_PATH, it will be replaced to result of it's to_str.

we have some choice.
1) support both ENV and getpwnam.
2) support only ENV.
3) no support home dirs change.
4) accept (G1)~(G3) and reject others.
5) reject this ticket :-(

what do you think about this issue?
in my mind, 4 > 1 = 2 >>>>>>> 5 > 3

#17 - 10/30/2012 01:39 AM - h.shirosaki (Hiroshi Shirosaki)

Thank you for review.

> 1) support both ENV and getpwnam.

My intention is 1) since I would like see (G4) merged.

Additional patch.
https://gist.github.com/3964679#file_0004_fix_compatibility_of_require.patch

No cache for '~' or '~xxxx' and not replace load path.
If $LOAD_PATH contains relative path or '~' or '~user', require would slow down.

I tried another benchmark. It uses many relative load path instead of expanded load path.
In this benchmark, (G1) to (G4) slows down a lot. But I think usually load path contains absolute path.
https://gist.github.com/3964679#file_0000_bench_requrie_empty_relative.rb

relative load path

Greg's patch (G1) to (G4)

ruby 2.0.0dev (2012-10-27 trunk 37341) [x86_64-darwin12.2.0]
user   system   total   real
0.810000   1.350000   2.160000 (2.171822)

```
LOAD_PATH SIZE = 209
LOADED_FEATURES SIZE = 1010
```

Greg's patch (G1) to (G4) + my patch (S1) to (S4)

```
ruby 2.0.0dev (2012-10-27 trunk 37341) [x86_64-darwin12.2.0]
user     system      total        real
0.400000   0.360000   0.760000 (  0.755793)
```

#18 - 10/30/2012 09:17 AM - ko1 (Koichi Sasada)

- Assignee changed from ko1 (Koichi Sasada) to tarui (Masaya Tarui)

- Priority changed from Normal to 5

- Target version set to 2.0.0

#19 - 10/30/2012 11:42 PM - tarui (Masaya Tarui)

- Assignee changed from tarui (Masaya Tarui) to h.shirosaki (Hiroshi Shirosaki)

shirosaki-san,
Thank you for fast patch.

it seems almost good.
one point, freezed NonStringObject can return different result by redefining to_str method in class scope.
if expand it everytime as same as home dirs, It is not necessary to freeze it.
And rb_ary_shared_with_p mechanism will continue operating well about StringObject.

Would you make the above-mentioned change?
In either case, please commit all patches.

Best regards.

#20 - 10/31/2012 10:19 AM - h.shirosaki (Hiroshi Shirosaki)

tarui (Masaya Tarui) wrote:

> it seems almost good.
> one point, freezed NonStringObject can return different result by redefining to_str method in class scope.
> if expand it everytime as same as home dirs, It is not necessary to freeze it.
> And rb_ary_shared_with_p mechanism will continue operating well about StringObject.
>
> Would you make the above-mentioned change?
> In either case, please commit all patches.

Thank you. I agreed.
I will change to expand NonStringObject(TYPE(v) != T_STRING) every time and not freeze it.

#21 - 11/01/2012 01:09 PM - h.shirosaki (Hiroshi Shirosaki)

I noticed one compatibility issue.

rb_file_expand_path_fast calls to_path method before to_str, but to_path was ignored.

I fixed to_path is called before to_str. If StringObject has to_path, the object are expended every time because frozen StringObject's to_path can return different result.

NonStringObject is not frozen and is expended every times.

I extracted functions from rb_get_path_check() and call it in load.c for to_path and to_str.

tarui-san, what do you think?

I rebased against latest trunk and pushed to github.

https://github.com/shirosaki/ruby/commits/optimize_require

latest patch:
https://github.com/shirosaki/ruby/commit/480fcd43ce844fef456b958352c9b8e605a6ff0e

diff againt trunk:
https://github.com/shirosaki/ruby/compare/trunk...optimize_require

#22 - 11/02/2012 09:10 AM - tenderlovemaking (Aaron Patterson)

Just wanted to update with some test results. I tried Hiroshi's patch along with my DTrace patches to time require searches against a small Rails application. Without GC running, before Hiroshi's patch, require searching took about 30% of the app require time. After applying the patch, I saw it was only 16%.

Here is the application I used:

https://github.com/tenderlove/lolwut

This is the test I ran:

$ ruby -Ilib:. -rnotrace -rconfig/environment -e 0

"notrace.rb" just contains "GC.disable". This test loads bundler and the rails environment, then exits.

Here is the DTrace script I used:

https://gist.github.com/3997732#file_find_require.d

CSV of the times (left column is the current time, center column is the time it took for searching, right column is the file name),

trunk: https://gist.github.com/3997732#file_trunk_req.csv
with patch: https://gist.github.com/3997732#file_fast_req.csv

Here are graphs of the data. X is the file number required (1 is the first file, 2 is the second, and so on), Y is the time it took to search in nanoseconds. Black is trunk, green is with this patch:

http://i.imgur.com/22hf4.png

Another graph, but the times are sorted. Black is trunk, red is with the patch:

http://i.imgur.com/yvAWn.png

#23 - 11/02/2012 12:23 PM - tarui (Masaya Tarui)

Thank you for the careful work.

Although I saw patch, compatibility is maintained now or it cannot
have confidence.
ex) OK even if String#to_path is defined after constructing expand_path?

I think reasonable to change specification to stop calling to_path
when it is T_STRING.

shirosaki-san, sasada-san, endoh-san, what do you think?

Best regards,

#24 - 11/02/2012 02:53 PM - ko1 (Koichi Sasada)

(2012/11/01 21:10), Masaya TARUI wrote:

> shirosaki-san, sasada-san, endoh-san, what do you think?

I don't have any idea on it.

Matz:
Please decide it.

--
// SASADA Koichi at atdot dot net

#25 - 11/02/2012 03:53 PM - Anonymous

Hi,

In message "Re: [ruby-core:48738] Re: [ruby-trunk - Bug #7158] require is slow in its bookkeeping; can make Rails startup 2.2x faster"
on Fri, 2 Nov 2012 14:42:16 +0900, SASADA Koichi ko1@atdot.net writes:
|
|(2012/11/01 21:10), Masaya TARUI wrote:
|> shirosaki-san, sasada-san, endoh-san, what do you think?
|
|I don't have any idea on it.
|

|Matz:
|Please decide it.

Then, go ahead and see if we will face any problem.

```
                              matz.
```

#26 - 11/02/2012 06:34 PM - h.shirosaki (Hiroshi Shirosaki)

tarui (Masaya Tarui) wrote:

> I think reasonable to change specification to stop calling to_path
> when it is T_STRING.

I agree with that. to_path of T_STRING may be less-useful.
Then I'm going to stop calling to_path of T_STRING in rb_get_path_check() for consistency.

Here is the change.
https://github.com/shirosaki/ruby/commit/bc7652840d6ad03db4fb713c4142e1464a695e0c

test, test-all, rubyspec results look fine. So I'll commit if there are no other opinions.

#27 - 11/06/2012 12:24 AM - Anonymous

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r37477.
Greg, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

Clarify and explain loaded_feature_path and rb_feature_p

- load.c (loaded_feature_path): clarify and briefly comment
  function. These clarifications have no effect on the behavior
  of the function.

- load.c (rb_feature_p): explain the search loop. Especially
  useful because the logic is complicated as described in the
  second paragraph.
  Patch by Greg Price.
  [ruby-core:47970] [Bug #7158]

#28 - 02/04/2013 04:24 AM - lzap (Lukas Zapletal)

Hello,

I am trying Ruby 2.0 RC1 which should contain this fix, but I think I still see many stat and open syscalls with ENOENT result. The number is slightly lower.

$ ruby -v
ruby 1.9.3p327 (2012-11-10 revision 37606) [x86_64-linux]
$ strace irb < /dev/null 2>&1 | grep ENOENT | wc -l
5195

$ ruby -v
ruby 2.0.0dev (2013-01-07 trunk 38733) [x86_64-linux]
$ strace irb < /dev/null 2>&1 | grep ENOENT | wc -l
5112

Here is the full strace output with all these ENOENT lines for 2.0 RC1. I guess something is wrong with locale trying to find all rb/so files with my cs_CZ and cs locales first, which essentialy triple amount of requires. I can confirm the same behavior for 1.9.3 ruby.

Edit: Forgot the add the link - http://sprunge.us/gffS

#29 - 02/04/2013 04:31 AM - lzap (Lukas Zapletal)

Ok the above looks like bug in lib/irb/locale.rb:

```
LC_ALL=C strace irb < /dev/null 2>&1 | grep ENOENT | wc -l
293
```

It not only triples amount of stat/open calls, it is like 17.5x faster. I am filling new bugreport for this.

Files

0001-Clarify-and-explain-loaded_feature_path-and-rb_featu.patch	4.54 KB	10/14/2012	gregprice (Greg Price)
0002-Expose-whether-two-arrays-are-shared-read-only-C-onl.patch	2.15 KB	10/14/2012	gregprice (Greg Price)
0003-Index-LOADED_FEATURES-so-require-isn-t-so-slow.patch	11.2 KB	10/14/2012	gregprice (Greg Price)
0004-Cache-the-expanded-load-path.patch	4.33 KB	10/14/2012	gregprice (Greg Price)