**Description**

- [PATCH 1/2] variable.c: extract common functions for generic ivar
  http://80x24.org/spew/m/4e9df8a150a121c894fe142bde5e5f15d43e5e94.txt
- [PATCH 2/2] variable.c: use indices for generic ivars
  http://80x24.org/spew/m/aabb09c886a23ea496722b13f2b39da8606b8180.txt

This reduces memory overhead of ivars for common types such as T_DATA the same way T_OBJECT does it.

For 9992 accepted clients on an OpenSSL server, this reduces RSS memory from 77160K to 69248K with the attached ossl.rb script.

Connecting client process was reduced from 246312K to 230724K RSS.

OpenSSL 1.0.1e-2+deb7u16 on Debian 7

**Associated revisions**

Revision 9d9aea7f - 05/29/2015 11:42 PM - normal
variable.c: use indices for generic ivars

This reduces memory overhead of ivars for common types such as T_DATA the same way T_OBJECT does it.

For 9992 accepted clients on an OpenSSL server, this reduces memory from 77160K to 69248K with the script in https://bugs.ruby-lang.org/issues/11170

- variable.c (static int special_generic_ivar): move
  (rb_generic_ivar_table): rewrite for compatibility
  (gen_ivtbl_bytes): new function
  (generic_ivar_get): update to use ivar index
  (generic_ivar_update): ditto
  (generic_ivar_set): ditto
  (generic_ivar_defined): ditto
  (generic_ivar_remove): ditto
  (rb_mark_generic_ivar): ditto
  (givar_i): ditto
  (rb_free_generic_ivar): ditto
  (rb_mark_generic_ivar_tbl): ditto
  (rb_generic_ivar_memsizes): ditto
  (rb_copy_generic_ivar): ditto
  (rb_ivar_set): ditto
  (rb_ivar_foreach): ditto
  (rb_ivar_count): ditto
  (givar_mark_i): remove
  (gen_ivtbl_mark): new function
  (gen_ivar_each): ditto
  (iv_index_tbl_extend): update for struct ivar_update
  (iv_index_tbl_newsize): ditto
  [ruby-core:69323] [Feature #11170]

Revision 50678 - 05/29/2015 11:42 PM - normalperson (Eric Wong)
variable.c: use indices for generic ivars

This reduces memory overhead of ivars for common types such as
T_DATA the same way T_OBJECT does it.

For 9992 accepted clients on an OpenSSL server, this reduces memory from 77160K to 69248K with the script in
https://bugs.ruby-lang.org/issues/11170

- variable.c (static int special_generic_ivar): move
  (rb_generic_ivar_table): rewrite for compatibility
  (gen_ivtbl_bytes): new function
  (generic_ivar_get): update to use ivar index
  (generic_ivar_update): ditto
  (generic_ivar_set): ditto
  (generic_ivar_defined): ditto
  (generic_ivar_remove): ditto
  (rb_mark_generic_ivar): ditto
  (givar_i): ditto
  (rb_free_generic_ivar): ditto
  (rb_mark_generic_ivar_tbl): ditto
  (rb_generic_ivar_memsize): ditto
  (rb_copy_generic_ivar): ditto
  (rb_ivar_set): ditto
  (rb_ivar_foreach): ditto
  (rb_count): ditto
  (givar_mark_i): remove
  (gen_ivtbl_mark): new function
  (gen_ivar_each): ditto
  (ivar_index_tbl_extend): update for struct ivar_update
  (ivar_index_tbl_newsize): ditto

[ruby-core:69323] [Feature #11170]

Revision 50678 - 05/29/2015 11:42 PM - normal
variable.c: use indices for generic ivars

This reduces memory overhead of ivars for common types such as
T_DATA the same way T_OBJECT does it.

For 9992 accepted clients on an OpenSSL server, this reduces memory from 77160K to 69248K with the script in
https://bugs.ruby-lang.org/issues/11170

- variable.c (static int special_generic_ivar): move
  (rb_generic_ivar_table): rewrite for compatibility
  (gen_ivtbl_bytes): new function
  (generic_ivar_get): update to use ivar index
  (generic_ivar_update): ditto
  (generic_ivar_set): ditto
  (generic_ivar_defined): ditto
  (generic_ivar_remove): ditto
  (rb_mark_generic_ivar): ditto
  (givar_i): ditto
  (rb_free_generic_ivar): ditto
  (rb_mark_generic_ivar_tbl): ditto
  (rb_generic_ivar_memsize): ditto
  (rb_copy_generic_ivar): ditto
  (rb_ivar_set): ditto
  (rb_ivar_foreach): ditto
  (rb_count): ditto
  (givar_mark_i): remove
  (gen_ivtbl_mark): new function
  (gen_ivar_each): ditto
  (ivar_index_tbl_extend): update for struct ivar_update
  (ivar_index_tbl_newsize): ditto

[ruby-core:69323] [Feature #11170]

Revision 50678 - 05/29/2015 11:42 PM - normal
variable.c: use indices for generic ivars

This reduces memory overhead of ivars for common types such as
T_DATA the same way T_OBJECT does it.

For 9992 accepted clients on an OpenSSL server, this reduces memory from 77160K to 69248K with the script in
https://bugs.ruby-lang.org/issues/11170
variable.c (static int special_generic_ivar): move
(rb_generic_ivar_table): rewrite for compatibility
(gen_ivtbl_bytes): new function
(generic_ivar_get): update to use ivar index
(generic_ivar_update): ditto
(generic_ivar_set): ditto
(generic_ivar_defined): ditto
(generic_ivar_remove): ditto
(rb_mark_generic_ivar): ditto
givar_i: ditto
(rb_free_generic_ivar): ditto
(rb_mark_generic_ivar_tbl): ditto
(rb_generic_ivar_memsize): ditto
(rb_copy_generic_ivar): ditto
(rb_ivar_set): ditto
(rb_ivar_foreach): ditto
(rb_ivar_count): ditto
givar_mark_i: remove
(gen_ivtbl_mark): new function
(gen_ivar_each): ditto
(iv_index_tbl_extend): update for struct ivar_update
(iv_index_tbl_newsize): ditto

[ruby-core:69323] [Feature #11170]

Revision 50678 - 05/29/2015 11:42 PM - normal

variable.c: use indices for generic ivars

This reduces memory overhead of ivars for common types such as
T_DATA the same way T_OBJECT does it.

For 9992 accepted clients on an OpenSSL server, this reduces
memory from 77160K to 69248K with the script in
https://bugs.ruby-lang.org/issues/11170

variable.c (static int special_generic_ivar): move
(rb_generic_ivar_table): rewrite for compatibility
(gen_ivtbl_bytes): new function
(generic_ivar_get): update to use ivar index
(generic_ivar_update): ditto
(generic_ivar_set): ditto
(generic_ivar_defined): ditto
(generic_ivar_remove): ditto
(rb_mark_generic_ivar): ditto
givar_i: ditto
(rb_free_generic_ivar): ditto
(rb_mark_generic_ivar_tbl): ditto
(rb_generic_ivar_memsize): ditto
(rb_copy_generic_ivar): ditto
(rb_ivar_set): ditto
(rb_ivar_foreach): ditto
(rb_ivar_count): ditto
givar_mark_i: remove
(gen_ivtbl_mark): new function
(gen_ivar_each): ditto
(iv_index_tbl_extend): update for struct ivar_update
(iv_index_tbl_newsize): ditto

[ruby-core:69323] [Feature #11170]

Revision 50678 - 05/29/2015 11:42 PM - normal

variable.c: use indices for generic ivars

This reduces memory overhead of ivars for common types such as
T_DATA the same way T_OBJECT does it.

For 9992 accepted clients on an OpenSSL server, this reduces
memory from 77160K to 69248K with the script in
https://bugs.ruby-lang.org/issues/11170
Revision f6cd5825 - 05/30/2015 12:20 AM - normal

variable.c: avoid compatibility table with generic ivars

This recovers and improves performance of Marshal.dump/load on
Time objects compared to when we implemented generic ivars
entirely using st_table.

This also recovers some performance on other generic ivar objects,
but does not bring bring Marshal.dump/load performance up to
previous speeds.

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)
name    trunk   geniv   after
marshal_dump_flo        0.343   0.334   0.335
marshal_dump_load_geniv 0.487   0.527   0.495
marshal_dump_load_time  1.262   1.401   1.257

Speedup ratio: compare with the result of 'trunk' (greater is better)
name    geniv   after
marshal_dump_flo        1.026   1.023
marshal_dump_load_geniv 0.925   0.985
marshal_dump_load_time  0.901   1.004

- include/ruby/intern.h (rb_generic_ivar_table): deprecate
- internal.h (rb_attr_delete): declare
- marshal.c (has_ivars): use rb_iivar_foreach
  (w_iivar): ditto
  (w_object): update for new interface
- time.c (time_mload): use rb_attr_delete
- variable.c (generic_ivar_delete): implement
  (rb_iivar_delete): ditto
  [ruby-core:69323] [Feature #11170]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@50680 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 50680 - 05/30/2015 12:20 AM - normalperson (Eric Wong)

variable.c: avoid compatibility table with generic ivars

This recovers and improves performance of Marshal.dump/load on
Time objects compared to when we implemented generic ivars
entirely using st_table.

This also recovers some performance on other generic ivar objects,
but does not bring bring Marshal.dump/load performance up to
previous speeds.

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)
name    trunk   geniv   after
marshal_dump_flo        0.343   0.334   0.335
marshaldump_load_geniv 0.487  0.527  0.495
marshaldump_load_time  1.262  1.401  1.257

Speedup ratio: compare with the result of `trunk' (greater is better)
name geniv after
marshaldump_flo 1.026  1.023
marshaldump_load_geniv 0.925  0.985
marshaldump_load_time  0.901  1.004

- include/ruby/intern.h (rb_generic_ivar_table): deprecate
- internal.h (rb_attr_delete): declare
- marshal.c (has_ivars): use rb_ivar_foreach
  (w_ivar): ditto
  (w_object): update for new interface
- time.c (time_mload): use rb_attr_delete
- variable.c (generic_ivar_delete): implement
  (rb_ivar_delete): ditto
  (rb_attr_delete): ditto
[ruby-core:69323] [Feature #11170]

Revision 50680 - 05/30/2015 12:20 AM - normal
variable.c: avoid compatibility table with generic ivars

This recovers and improves performance of Marshal.dump/load on
Time objects compared to when we implemented generic ivars
entirely using st_table.

This also recovers some performance on other generic ivar objects,
but does not bring Marshal.dump/load performance up to
previous speeds.

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)
name trunk geniv after
marshaldump_flo  0.343  0.334  0.335
marshaldump_load_geniv 0.487  0.527  0.495
marshaldump_load_time  1.262  1.401  1.257

Speedup ratio: compare with the result of `trunk' (greater is better)
name geniv after
marshaldump_flo 1.026  1.023
marshaldump_load_geniv 0.925  0.985
marshaldump_load_time  0.901  1.004

- include/ruby/intern.h (rb_generic_ivar_table): deprecate
- internal.h (rb_attr_delete): declare
- marshal.c (has_ivars): use rb_ivar_foreach
  (w_ivar): ditto
  (w_object): update for new interface
- time.c (time_mload): use rb_attr_delete
- variable.c (generic_ivar_delete): implement
  (rb_ivar_delete): ditto
  (rb_attr_delete): ditto
[ruby-core:69323] [Feature #11170]

Revision 50680 - 05/30/2015 12:20 AM - normal
variable.c: avoid compatibility table with generic ivars

This recovers and improves performance of Marshal.dump/load on
Time objects compared to when we implemented generic ivars
entirely using st_table.

This also recovers some performance on other generic ivar objects,
but does not bring Marshal.dump/load performance up to
previous speeds.

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)
name trunk geniv after
marshaldump_flo  0.343  0.334  0.335
marshaldump_load_geniv 0.487  0.527  0.495
Revision 50680 - 05/30/2015 12:20 AM - normal

variable.c: avoid compatibility table with generic ivars

This recovers and improves performance of Marshal.dump/load on Time objects compared to when we implemented generic ivars entirely using st_table.

This also recovers some performance on other generic ivar objects, but does not bring bring Marshal.dump/load performance up to previous speeds.

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)
name trunk geniv after
marshal_dump_flo 0.343 0.334 0.335
marshal_dump_load_geniv 0.487 0.527 0.495
marshal_dump_load_time 1.262 1.401 1.257

Revision 50680 - 05/30/2015 12:20 AM - normal

variable.c: avoid compatibility table with generic ivars

This recovers and improves performance of Marshal.dump/load on Time objects compared to when we implemented generic ivars entirely using st_table.

This also recovers some performance on other generic ivar objects, but does not bring bring Marshal.dump/load performance up to previous speeds.

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)
name trunk geniv after
marshal_dump_flo 0.343 0.334 0.335
marshal_dump_load_geniv 0.487 0.527 0.495
marshal_dump_load_time 1.262 1.401 1.257
Speedup ratio: compare with the result of ‘trunk’ (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>geniv</th>
<th>after</th>
</tr>
</thead>
<tbody>
<tr>
<td>marshal_dump_flo</td>
<td>1.026</td>
<td>1.023</td>
</tr>
<tr>
<td>marshal_dump_load_geniv</td>
<td>0.925</td>
<td>0.985</td>
</tr>
<tr>
<td>marshal_dump_load_time</td>
<td>0.901</td>
<td>1.004</td>
</tr>
</tbody>
</table>

- include/ruby/intern.h (rb_generic_ivar_table): deprecate
- internal.h (rb_attr_delete): declare
- marshal.c (has_ivars): use rb_ivar_foreach
- (w_ivar); ditto
- (w_object); update for new interface
- time.c (time_mload): use rb_attr_delete
- variable.c (generic_ivar_delete): implement
- (rb_ivar_delete): ditto
- (rb_attr_delete): ditto
- [ruby-core:69323] [Feature #11170]

History

#1 - 05/23/2015 01:35 AM - normalperson (Eric Wong)
- File ossl_11170.rb added

Attached standalone test script, increase "ulimit -n" as necessary.

#2 - 05/23/2015 02:19 AM - ko1 (Koichi Sasada)
+1.

T_CLASS/T_MODULE can use same technique, but it seems not so many use-cases.

#3 - 05/29/2015 12:58 AM - normalperson (Eric Wong)
After the original patch, rb_generic_ivar_table() is much more expensive
but kept for compatibility reasons. I propose deprecating it, I'm not
sure if any 3rd party C-exts use it.

http://80x24.org/spew/m/1432859944-14374-1-git-send-email-e@80x24.org.txt

[PATCH 3/2] avoid compatibility table with generic ivars

This recovers and improves performance of Marshal.dump/load on
Time objects compared to when we implemented generic ivars
entirely using st_table.

This also recovers some performance on other generic ivar objects,
but does not bring bring Marshal.dump/load performance up to
previous speeds.

benchmark results:
minimum results in each 10 measurements.
Execution time (sec)

<table>
<thead>
<tr>
<th>name</th>
<th>trunk</th>
<th>geniv</th>
<th>after</th>
</tr>
</thead>
<tbody>
<tr>
<td>marshal_dump_flo</td>
<td>0.343</td>
<td>0.334</td>
<td>0.335</td>
</tr>
<tr>
<td>marshal_dump_load_geniv</td>
<td>0.487</td>
<td>0.527</td>
<td>0.495</td>
</tr>
<tr>
<td>marshal_dump_load_time</td>
<td>1.262</td>
<td>1.401</td>
<td>1.257</td>
</tr>
</tbody>
</table>

Speedup ratio: compare with the result of ‘trunk’ (greater is better)

<table>
<thead>
<tr>
<th>name</th>
<th>geniv</th>
<th>after</th>
</tr>
</thead>
<tbody>
<tr>
<td>marshal_dump_flo</td>
<td>1.026</td>
<td>1.023</td>
</tr>
<tr>
<td>marshal_dump_load_geniv</td>
<td>0.925</td>
<td>0.985</td>
</tr>
<tr>
<td>marshal_dump_load_time</td>
<td>0.901</td>
<td>1.004</td>
</tr>
</tbody>
</table>

#4 - 05/29/2015 11:43 PM - Anonymous
- Status changed from Open to Closed

Applied in changeset r50678.

variable.c: use indices for generic ivars

This reduces memory overhead of ivars for common types such as
T_DATA the same way T_OBJECT does it.
For 9992 accepted clients on an OpenSSL server, this reduces memory from 77160K to 69248K with the script in https://bugs.ruby-lang.org/issues/11170

- variable.c (static int special_generic_ivar): move
  (rb_generic_ivar_table): rewrite for compatibility
  (gen_ivtbl_bytes): new function
  (generic_ivar_get): update to use ivar index
  (generic_ivar_update): ditto
  (generic_ivar_set): ditto
  (generic_ivar_defined): ditto
  (generic_ivar_remove): ditto
  (rb_mark_generic_ivar): ditto
  (givar_i): ditto
  (rb_free_generic_ivar): ditto
  (rb_mark_generic_ivar_tbl): ditto
  (rb_generic_ivar_memsize): ditto
  (rb_copy_generic_ivar): ditto
  (rb_ivar_set): ditto
  (rb_ivar_foreach): ditto
  (rb_ivar_count): ditto
  (givar_mark_i): remove
  (gen_ivtbl_mark): new function
  (gen_ivar_each): ditto
  (iv_index_tbl_extend): update for struct ivar_update
  (iv_index_tbl_newsize): ditto
[ruby-core:69323] [Feature #11170] [Feature #11170]

Files
ivar-reduce-combined.patch 17.2 KB 05/23/2015  normalperson (Eric Wong)
ossl_11170.rb 1.74 KB 05/23/2015  normalperson (Eric Wong)