Ruby master - Bug #11279
remove rb_control_frame_t::klass
06/18/2015 11:35 AM - ko1 (Koichi Sasada)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>ko1 (Koichi Sasada)</td>
</tr>
<tr>
<td>Target version:</td>
<td>ruby -v: 2.3dev</td>
</tr>
<tr>
<td>Backport:</td>
<td>2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN</td>
</tr>
</tbody>
</table>

**Description**

Please check [https://bugs.ruby-lang.org/issues/11278](https://bugs.ruby-lang.org/issues/11278)

I sent it to ruby-dev because of my operation miss.

This is big change of method data structure.

**Related issues:**

Related to Ruby master - Bug #11278: remove rb_control_frame_t::klass
Closed

**Associated revisions**

Revision 5e8a1474 - 07/03/2015 11:24 AM - ko1 (Koichi Sasada)

- method.h: introduce rb_callable_method_entry_t to remove rb_control_frame_t::klass. [Bug #11278], [Bug #11279] rb_method_entry_t data belong to modules/classes. rb_method_entry_t::owner points defined module or class. module M def foo; end end In this case, owner is M. rb_callable_method_entry_t data belong to only classes. For modules, MRI creates corresponding T_ICLASS internally. rb_method_entry_t can also belong to T_ICLASS. rb_callable_method_entry_t::defined_class points T_CLASS or T_ICLASS. rb_method_entry_t data for classes (not for modules) are also rb_callable_method_entry_t data because it is completely same data. In this case, rb_method_entry_t::owner == rb_callable_method_entry_t::defined_class. For example, there are classes C and D, and includes M, class C; include M; end then, two T_ICLASS objects for C's super class and D's super class will be created. When C.new.foo is called, then Mfoo is searched and rb_callable_method_t data is used by VM to invoke Mfoo. rb_method_entry_t data is only one for Mfoo. However, rb_callable_method_entry_t data are two (and can be more). It is proportional to the number of including (and prepending) classes (the number of T_ICLASS which point to the module). Now, created rb_callable_method_entry_t are collected when the original module M was modified. We can think it is a cache. We need to select what kind of method entry data is needed. To operate definition, then you need to use rb_method_entry_t. You can access them by the following functions.

  - rb_method_entry(VALUE klass, ID id);
  - rb_method_entry_with_refinements(VALUE klass, ID id);
  - rb_method_entry_without_refinements(VALUE klass, ID id);
  - rb_resolve_refined_method(VALUE refinements, const rb_method_entry_t *me); To invoke methods, then you need to use rb_callable_method_entry_t which you can get by the following APIs corresponding to the above listed functions.

  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry_with_refinements(VALUE klass, ID id);
  - rb_callable_method_entry_without_refinements(VALUE klass, ID id);
  - rb_resolve_refined_method Callable(VALUE refinements, const rb_callable_method_entry_t *me); VM pushes rb_callable_method_entry_t, so that rb_vm_frame_method_entry() returns rb_callable_method_entry_t. You can check a super class of current method by rb_callable_method_entry_t::defined_class.

- method.h: renamed from rb_method_entry_t::klass to rb_method_entry_t::owner.

- internal.h: add rb_classext struct::callable_m_tbl to cache rb_callable_method_entry_t data. We need to consider aboue this field again because it is only active for T_ICLASS.

- class.c (method_entry_i): ditto.

- class.c (rb_define_atrr): rb_method_entry() does not takes defiend_class_ptr.

- gc.c (mark_method_entry): mark RCLASS_CALLABLE_M_TBL for T_ICLASS.

- cont.c (fiber_init): rb_control_frame_t::klass is removed.

- proc.c: fix `struct METHOD' data structure because rb_callable_method_t has all information.

- vm_core.h: remove several fields.

  - rb_control_frame_t::klass.

  - rb_block_t::klass. And catch up changes.

- eval.c: catch up changes.

- gc.c: ditto.

- insns.def: ditto.

- vm.c: ditto.

- vm_args.c: ditto.

- vm_backtrace.c: ditto.

- vm_dump.c: ditto.

- vm_eval.c: ditto.

- vm_insnhelper.c: ditto.

- vm_method.c: ditto.
Revision 51126 - 07/03/2015 11:24 AM - ko1 (Koichi Sasada)

- method.h: introduce rb_callable_method_entry_t to remove rb_control_frame_t::class. [Bug #11278], [Bug #11279] rb_method_entry_t data belong to modules/classes. rb_method_entry_t::owner points defined module or class. module M def foo; end end In this case, owner is M. rb_callable_method_entry_t data belong to only classes. For modules, MRI creates corresponding T_ICLASS internally. rb_callable_method_entry_t can also belong to T_ICLASS. rb_callable_method_entry_t::defined_class points T_CLASS or T_ICLASS. rb_method_entry_t data for classes (not for modules) are also rb_callable_method_entry_t data because it is completely same data. In this case, rb_method_entry_t::owner == rb_method_entry_t::defined_class. For example, there are classes C and D, and includes M, class C; include M; end class D; include M; end then, two T_ICLASS objects for C's super class and D's super class will be created. When C.new.foo is called, then M#foo is searched and rb_callable_method_t data is used by VM to invoke M#foo. rb_method_entry_t data is only one for M#foo. However, rb_callable_method_entry_t data are two (and can be more). It is proportional to the number of including (and prepending) classes (the number of T_ICLASS which point to the module). Now, created rb_callable_method_entry_t are collected when the original module M was modified. We can think it is a cache. We need to select what kind of method entry data is needed. To operate definition, then you need to use rb_method_entry_t. You can access them by the following functions.
  - rb_method_entry(VALUE klass, ID id);
  - rb_method_entry(VALUE klass, ID id);
  - rb_method_entry(VALUE klass, ID id);
  - rb_resolve_refined_method(VALUE refinements, const rb_method_entry_t *me); To invoke methods, then you need to use rb_callable_method_entry_t which you can get by the following APIs corresponding to the above listed functions.
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);

- internals.h: add rb_bitclass_struct::callable_m_tbl to cache rb_callable_method_entry_t data. We need to consider about this field again because it is only active for T_ICLASS.
  - class.c (method_entry_i): ditto.
  - class.c (rb_define_attr): rb_method_entry() does not takes defiend_class_ptr.
  - gc.c (mark_method_entry): mark RCLASS_CALLABLE_M_TBL() for T_ICLASS.
  - cont.c (fiber_init): rb_control_frame_t::klass is removed.
  - proc.c: fix 'struct METHOD' data structure because rb_callable_method_t has all information.
  - vm_core.h: remove several fields.
    - rb_control_frame_t::klass.
    - rb_block_t::klass. And catch up changes.
  - eval.c: catch up changes.
  - gc.c: ditto.
  - insns.def: ditto.
  - vm.c: ditto.
  - vm_args.c: ditto.
  - vm_backtrace.c: ditto.
  - vm_dump.c: ditto.
  - vm_eval.c: ditto.
  - vm_irshhelper.c: ditto.
  - vm_method.c: ditto.

Revision 51126 - 07/03/2015 11:24 AM - ko1 (Koichi Sasada)

- method.h: introduce rb_callable_method_entry_t to remove rb_control_frame_t::class. [Bug #11278], [Bug #11279] rb_method_entry_t data belong to modules/classes. rb_method_entry_t::owner points defined module or class. module M def foo; end end In this case, owner is M. rb_callable_method_entry_t data belong to only classes. For modules, MRI creates corresponding T_ICLASS internally. rb_callable_method_entry_t can also belong to T_ICLASS. rb_callable_method_entry_t::defined_class points T_CLASS or T_ICLASS. rb_method_entry_t data for classes (not for modules) are also rb_callable_method_entry_t data because it is completely same data. In this case, rb_method_entry_t::owner == rb_method_entry_t::defined_class. For example, there are classes C and D, and includes M, class C; include M; end class D; include M; end then, two T_ICLASS objects for C's super class and D's super class will be created. When C.new.foo is called, then M#foo is searched and rb_callable_method_t data is used by VM to invoke M#foo. rb_method_entry_t data is only one for M#foo. However, rb_callable_method_entry_t data are two (and can be more). It is proportional to the number of including (and prepending) classes (the number of T_ICLASS which point to the module). Now, created rb_callable_method_entry_t are collected when the original module M was modified. We can think it is a cache. We need to select what kind of method entry data is needed. To operate definition, then you need to use rb_method_entry_t. You can access them by the following functions.
  - rb_method_entry(VALUE klass, ID id);
  - rb_method_entry(VALUE klass, ID id);
  - rb_method_entry_with_refinements(VALUE klass, ID id);
  - rb_method_entry_without_refinements(VALUE klass, ID id);
  - rb_resolve_refined_method(VALUE refinements, const rb_method_entry_t *me); To invoke methods, then you need to use rb_callable_method_entry_t which you can get by the following APIs corresponding to the above listed functions.
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);
  - rb_callable_method_entry(VALUE klass, ID id);

03/28/2021
rb_callable_method_entry_t::defined_class.

- method.h: renamed from rb_method_entry_t::klass to rb_method_entry_t::owner.
- internal.h: add rb_classext_struct::callable_m_tbl to cache rb_callable_method_entry_t data. We need to consider about this field again because it is only active for T_ICLASS.
- class.c (method_entry_i): ditto.
- class.c (rb_define_attr): rb_method_entry() does not takes defiend_class_ptr.
- gc.c (mark_method_entry): mark RCLASS_CALLABLE_M_TBL() for T_ICLASS.
- cont.c (fiber_init): rb_control_frame_t::klass is removed.
- proc.c: fix 'struct METHOD' data structure because rb_callable_method_t has all information.
- vm_core.h: remove several fields.
  * rb_control_frame_t::klass.
  * rb_block_t::klass. And catch up changes.
- eval.c: catch up changes.
- gc.c: ditto.
- insns.def: ditto.
- vm.c: ditto.
- vm_args.c: ditto.
- vm_backtrace.c: ditto.
- vm_dump.c: ditto.
- vm_eval.c: ditto.
- vm_inshelper.c: ditto.
- vm_method.c: ditto.

Revision 51126 - 07/03/2015 11:24 AM - ko1 (Koichi Sasada)

- method.h: introduce rb_callable_method_entry_t to remove rb_control_frame_t::klass. [Bug #11278], [Bug #11279] rb_method_entry_t data belong to modules/classes. rb_method_entry_t::owner points defined module or class. module M def foo; end end In this case, owner is M. rb_callable_method_entry_t data belong to only classes. For modules, MRI creates corresponding T_ICLASS internally. rb_callable_method_entry_t can also belong to T_ICLASS. rb_callable_method_entry_t::defined_class points T_CLASS or T_ICLASS. rb_method_entry_t data for classes (not for modules) are also rb_callable_method_entry_t data because it is completely same data. In this case, rb_method_entry_t::owner == rb_method_entry_t::defined_class. For example, there are classes C and D, and includes M, class C; include M; end then, two T_ICLASS objects for C's super class and D's super class will be created. When C.new.foo is called, then M#foo is searched and rb_callable_method_t data is used by VM to invoke M#foo. rb_method_entry_t data is only one for M#foo. However, rb_callable_method_entry_t data are two (and can be more). It is proportional to the number of including (and prepending) classes (the number of T_ICLASS which point to the module). Now, created rb_callable_method_entry_t are collected when the original module M was modified. We can think it is a cache. We need to select what kind of method entry data is needed. To operate definition, then you need to use rb_method_entry_t. You can access them by the following functions.
  * rb_method_entry(VALUE klass, ID id);
  * rb_method_entry_with_refinements(VALUE klass, ID id);
  * rb_method_entry_without_refinements(VALUE klass, ID id);
  * rb_resolve_refined_method(VALUE refinements, const rb_method_entry_t *me); To invoke methods, then you need to use rb_callable_method_entry_t which you can get by the following APIs corresponding to the above listed functions.
  * rb_callable_method_entry(VALUE klass, ID id);
  * rb_callable_method_entry_with_refinements(VALUE klass, ID id);
  * rb_callable_method_entry_without_refinements(VALUE klass, ID id);
  * rb_resolve_refined_method_callable(VALUE refinements, const rb_callable_method_entry_t *me); VM pushes rb_callable_method_entry_t, so that rb_vm_frame_method_entry() returns rb_callable_method_entry_t. You can check a super class of current method by rb_callable_method_entry_t::defined_class.

- method.h: renamed from rb_method_entry_t::klass to rb_method_entry_t::owner.
- internal.h: add rb_classext_struct::callable_m_tbl to cache rb_callable_method_entry_t data. We need to consider about this field again because it is only active for T_ICLASS.
- class.c (method_entry_i): ditto.
- class.c (rb_define_attr): rb_method_entry() does not takes defiend_class_ptr.
- gc.c (mark_method_entry): mark RCLASS_CALLABLE_M_TBL() for T_ICLASS.
- cont.c (fiber_init): rb_control_frame_t::klass is removed.
- proc.c: fix 'struct METHOD' data structure because rb_callable_method_t has all information.
- vm_core.h: remove several fields.
  * rb_control_frame_t::klass.
  * rb_block_t::klass. And catch up changes.
- eval.c: catch up changes.
- gc.c: ditto.
- insns.def: ditto.
- vm.c: ditto.
- vm_args.c: ditto.
- vm_backtrace.c: ditto.
- vm_dump.c: ditto.
- vm_eval.c: ditto.
- vm_inshelper.c: ditto.
- vm_method.c: ditto.

Revision 51126 - 07/03/2015 11:24 AM - ko1 (Koichi Sasada)
belong to modules/classes. rb_method_entry_t::owner points defined module or class. module M def foo; end end In this case, owner is M.
rb_callable_method_entry_t data belong to only classes. For modules, MRI creates corresponding T_ICLASS internally.
rb_callable_method_entry_t can also belong to T_ICLASS. rb_callable_method_entry_t::defined_class points T_CLASS or T_ICLASS.
rb_method_entry_t data for classes (not for modules) are also rb_callable_method_entry_t data because it is completely same data. In this case, rb_method_entry_t::owner == rb_method_entry_t::defined_class. For example, there are classes C and D, and includes M, class C; include M; end class D; include M; end then, two T_ICLASS objects for C's super class and D's super class will be created. When C.new.foo is called, then M#foo is searched and rb_callable_method_t data is used by VM to invoke M#foo. rb_method_entry_t data is only one for M#foo. However, rb_callable_method_entry_t data are two (and can be more). It is proportional to the number of including (and prepending) classes (the number of T_ICLASS which point to the module). Now, created rb_callable_method_entry_t are collected when the original module M was modified. We can think it is a cache. We need to select what kind of method entry data is needed. To operate definition, then you need to use rb_method_entry_t. You can access them by the following functions.
- rb_method_entry(VALUE klass, ID id);
- rb_method_entry_with_refinements(VALUE klass, ID id);
- rb_method_entry_without_refinements(VALUE klass, ID id);
- rb_resolve_refined_method(VALUE refinements, const rb_method_entry_t *me); To invoke methods, then you need to use
rb_callable_method_entry_t which you can get by the following APIs corresponding to the above listed functions.
- rb_callable_method_entry(VALUE klass, ID id);
- rb_callable_method_entry_with_refinements(VALUE klass, ID id);
- rb_callable_method_entry_without_refinements(VALUE klass, ID id);
- rb_resolve_refined_methodCallable(VALUE refinements, const rb_callable_method_entry_t *me); VM pushes rb_callable_method_entry_t, so that rb_vm_frame_method_entry() returns rb_callable_method_entry_t. You can check a super class of current method by
rb_callable_method_entry_t::defined_class.

- method.h: renamed from rb_method_entry_t::klass to rb_method_entry_t::owner.
- internal.h: add rb_classext_struct::callable_m_tbl to cache rb_callable_method_entry_t data. We need to consider about this field again because it is only active for T_ICLASS.
class.c (method_entry_i): ditto.
class.c (rb_define_attr): rb_method_entry() does not take defined_class_ptr.
gc.c (mark_method_entry): mark RCLASS_CALLABLE_M_TBL() for T_ICLASS.
vm_core.h: remove several fields.
- rb_control_frame_t::klass.
- rb_block_t::klass. And catch up changes.
eval.c: catch up changes.
gc.c: ditto.
isns.def: ditto.
vm.c: ditto.
vm_args.c: ditto.
vm_backtrace.c: ditto.
vm_dump.c: ditto.
vm_eval.c: ditto.
vm_insnhelper.c: ditto.
vm_method.c: ditto.

Revision 51126 - 07/03/2015 11:24 AM - ko1 (Koichi Sasada)

- method.h: introduce rb_callable_method_entry_t to remove rb_control_frame_t::klass. [Bug #11278], [Bug #11279] rb_method_entry_t data belong to modules/classes. rb_method_entry_t::owner points defined module or class. module M def foo; end end In this case, owner is M.
rb_callable_method_entry_t data belong to only classes. For modules, MRI creates corresponding T_ICLASS internally.
rb_callable_method_entry_t can also belong to T_ICLASS. rb_callable_method_entry_t::defined_class points T_CLASS or T_ICLASS.
rb_method_entry_t data for classes (not for modules) are also rb_callable_method_entry_t data because it is completely same data. In this case, rb_method_entry_t::owner == rb_method_entry_t::defined_class. For example, there are classes C and D, and includes M, class C; include M; end class D; include M; end then, two T_ICLASS objects for C's super class and D's super class will be created. When C.new.foo is called, then M#foo is searched and rb_callable_method_t data is used by VM to invoke M#foo. rb_method_entry_t data is only one for M#foo. However, rb_callable_method_entry_t data are two (and can be more). It is proportional to the number of including (and prepending) classes (the number of T_ICLASS which point to the module). Now, created rb_callable_method_entry_t are collected when the original module M was modified. We can think it is a cache. We need to select what kind of method entry data is needed. To operate definition, then you need to use rb_method_entry_t. You can access them by the following functions.
- rb_method_entry(VALUE klass, ID id);
- rb_method_entry_with_refinements(VALUE klass, ID id);
- rb_method_entry_without_refinements(VALUE klass, ID id);
- rb_resolve_refined_method(VALUE refinements, const rb_method_entry_t *me); To invoke methods, then you need to use
rb_callable_method_entry_t which you can get by the following APIs corresponding to the above listed functions.
- rb_callable_method_entry(VALUE klass, ID id);
- rb_callable_method_entry_with_refinements(VALUE klass, ID id);
- rb_callable_method_entry_without_refinements(VALUE klass, ID id);
- rb_resolve_refined_methodCallable(VALUE refinements, const rb_callable_method_entry_t *me); VM pushes rb_callable_method_entry_t, so that rb_vm_frame_method_entry() returns rb_callable_method_entry_t. You can check a super class of current method by
rb_callable_method_entry_t::defined_class.

- method.h: renamed from rb_method_entry_t::klass to rb_method_entry_t::owner.
- internal.h: add rb_classext_struct::callable_m_tbl to cache rb_callable_method_entry_t data. We need to consider about this field again because it is only active for T_ICLASS.
class.c (method_entry_i): ditto.
class.c (rb_define_attr): rb_method_entry() does not take defined_class_ptr.
gc.c (mark_method_entry): mark RCLASS_CALLABLE_M_TBL() for T_ICLASS.
cont.c (fiber_init): rb_control_frame_t::klass is removed.
proc.c: fix 'struct METHOD' data structure because rb_callable_method_t has all information.
vm_core.h: remove several fields.
  * rb_control_frame_t::klass.
  * rb_block_t::klass. And catch up changes.
eval.c: catch up changes.
 gc.c: ditto.
 insns.def: ditto.
 vm.c: ditto.
 vm_args.c: ditto.
 vm_backtrace.c: ditto.
 vm_dump.c: ditto.
 vm_eval.c: ditto.
 vm_inshelper.c: ditto.
 vm_method.c: ditto.

History

#1 - 06/18/2015 11:36 AM - ko1 (Koichi Sasada)
- Related to Bug #11278: remove rb_control_frame_t::klass added

#2 - 07/03/2015 11:25 AM - ko1 (Koichi Sasada)
- Status changed from Open to Closed

Applied in changeset r51126.

- method.h: introduce rb_callable_method_entry_t to remove rb_control_frame_t::klass. [Bug #11278], [Bug #11279] rb_method_entry_t data belong to modules/classes. rb_method_entry_t::owner points defined module or class. module M def foo; end end In this case, owner is M.

  * rb_callable_method_entry_t data belong to only classes. For modules, MRI creates corresponding T_ICLASS internally.
  * rb_callable_method_entry_t can also belong to T_ICLASS. rb_callable_method_entry_t::defind_class points T_CLASS or T_ICLASS.
  * rb_method_entry_t data for classes (not for modules) are also rb_callable_method_entry_t data because it is completely same data. In this case, rb_method_entry_t::owner == rb_method_entry_t::defined_class. For example, there are classes C and D, and includes M, class C; include M; end class D; include M; end then, two T_ICLASS objects for C's super class and D's super class will be created. When C.new.foo is called, then M#foo is searched and rb_callable_method_t data is used by VM to invoke M#foo. rb_method_entry_t data is only one for M#foo. However, rb_callable_method_entry_t data are two (and can be more). It is proportional to the number of including (and prepending) classes (the number of T_ICLASS which point to the module). Now, created rb_callable_method_entry_t are collected when the original module M was modified. We can think it is a cache. We need to select what kind of method entry data is needed. To operate definition, then you need to use rb_method_entry_t. You can access them by the following functions.
  * rb_method_entry(VALUE klass, ID id);
  * rb_method_entry(VALUE klass, ID id);
  * rb_method_entry(VALUE klass, ID id);
  * rb_method_entry(VALUE klass, ID id);
  * rb_method_entry(VALUE klass, ID id);
  * rb_method_entry(VALUE klass, ID id);
  * rb_method_entry(VALUE klass, ID id);
  * rb_method_entry(VALUE klass, ID id);
  * rb_method_entry(VALUE klass, ID id);

- method.h: renamed from rb_method_entry_t::klass to rb_method_entry_t::owner.

- internal.h: add rb_classext_struct::callable_m_tbl to cache rb_callable_method_entry_t data. We need to consider about this field again because it is only active for T_ICLASS.

- class.c (method_entry_i): ditto.
- class.c (rb_defind_attr): rb_method_entry() does not takes defind_class_ptr.
- gc.c (mark_method_entry): mark RCLASS_CALLABLE_M_TBL() for T_ICLASS.
- cont.c (fiber_init): rb_control_frame_t::klass is removed.
- proc.c: fix 'struct METHOD' data structure because rb_callable_method_t has all information.

- History

  #1 - 06/18/2015 11:36 AM - ko1 (Koichi Sasada)
  - Related to Bug #11278: remove rb_control_frame_t::klass added

  #2 - 07/03/2015 11:25 AM - ko1 (Koichi Sasada)
  - Status changed from Open to Closed

  Applied in changeset r51126.

- History

  #1 - 06/18/2015 11:36 AM - ko1 (Koichi Sasada)
  - Related to Bug #11278: remove rb_control_frame_t::klass added

  #2 - 07/03/2015 11:25 AM - ko1 (Koichi Sasada)
  - Status changed from Open to Closed

  Applied in changeset r51126.

- History

  #1 - 06/18/2015 11:36 AM - ko1 (Koichi Sasada)
  - Related to Bug #11278: remove rb_control_frame_t::klass added

  #2 - 07/03/2015 11:25 AM - ko1 (Koichi Sasada)
  - Status changed from Open to Closed

  Applied in changeset r51126.