

Ruby - Feature #462

autoload with a block

08/21/2008 11:54 AM - nobu (Nobuyoshi Nakada)

Status:Rejected Priority:Normal Assignee:matz (Yukihiro Matsumoto) Target version:	
Description =begin Hi, At Mon, 18 Aug 2008 10:35:01 +0900, Nobuyoshi Nakada wrote in [ruby-talk:311603]: At Fri, 15 Aug 2008 15:57:21 +0900, David Masover wrote in [ruby-talk:311365]: autoload :Foo do require 'foo' Foo.autoload :Bar ... end In that simplest form, it would at least centralize all my loading stuff in one place. In a more meta form, I could run through some search paths I'm likely to use (in my own app, for instance), and generate an autoload scheme based on that. I don't think it makes things simpler, but it sounds interesting. You can file it in the ITS, if you want. http://redmine.ruby-lang.org/projects/ruby/issues?set_filter=1&tracker_id=2 Although the OP doesn't seem to intend to request this feature, I move it to ruby-core since it feels interesting. Index: load.c --- load.c (revision 18653) +++ load.c (working copy) @@ -618,10 +618,29 @@ ruby_init_ext(const char *name, void (*i static VALUE -rb_mod_autoload(VALUE mod, VALUE sym, VALUE file) +rb_mod_autoload(int argc, VALUE *argv, VALUE mod) { • ID id = rb_to_id(sym); • Check_SafeStr(file); • rb_autoload(mod, id, RSTRING_PTR(file)); • extern VALUE rb_autoload_proc(VALUE, ID, VALUE, VALUE); • ID id; • VALUE block = rb_block_given_p() ? rb_block_proc() : Qnil; • VALUE arg = Qnil; • switch (argc) { •	

- arg = argv[1];
- break;

- XXXXXXXXXX

- if (!NIL_P(block)) break;

- XXXXXXXXXX

- rb_raise(rb_eArgError, "wrong number of arguments (%d for 2)", argc);
- }
- id = rb_to_id(argv[0]);
- if (NIL_P(block)) {
- VALUE file = argv[1];
- Check_SafeStr(file);
- rb_autoload(mod, id, RSTRING_PTR(file));
- }
- else {
- rb_autoload_proc(mod, id, block, arg);
- }
- return Qnil;
- }

@@ -649,5 +668,5 @@ rb_mod_autoload_p(VALUE mod, VALUE sym)

static VALUE

```
-rb_f_autoload(VALUE obj, VALUE sym, VALUE file)
+rb_f_autoload(int argc, VALUE *argv, VALUE obj)
{
  VALUE klass = rb_vm_cbase();
  @@ -655,5 +674,5 @@ rb_f_autoload(VALUE obj, VALUE sym, VALU
  rb_raise(rb_eTypeError, "Can not set autoload on singleton class");
}

  • return rb_mod_autoload(klass, sym, file);

  • return rb_mod_autoload(argc, argv, klass);
}
```

```
@@ -693,7 +712,7 @@ Init_load()
rb_define_global_function("load", rb_f_load, -1);
rb_define_global_function("require", rb_f_require, 1);

  • rb_define_method(rb_cModule, "autoload", rb_mod_autoload, 2);

  • rb_define_method(rb_cModule, "autoload", rb_mod_autoload, -1);
    rb_define_method(rb_cModule, "autoload?", rb_mod_autoload_p, 1);

  • rb_define_global_function("autoload", rb_f_autoload, 2);

  • rb_define_global_function("autoload", rb_f_autoload, -1);
    rb_define_global_function("autoload?", rb_f_autoload_p, 1);
```

Index: variable.c

```
--- variable.c (revision 18653)
+++ variable.c (working copy)
@@ -1322,16 +1322,12 @@ check_autoload_table(VALUE av)
}

-void
-rb_autoload(VALUE mod, ID id, const char *file)
+typedef VALUE rb_autoload_func(VALUE, ID, VALUE, VALUE);
+
+static void
```

```

+autoload_callback(VALUE mod, ID id, rb_autoload_func *func, VALUE arg1, VALUE arg2)
{
    • VALUE av, fn;

    • VALUE av;
      struct st_table *tbl;

    • if (!rb_is_const_id(id)) {
    • rb_raise(rb_eNameError, "autoload must be constant name: %s", rb_id2name(id));
    • }
    • if (!file || !*file) {
    • rb_raise(rb_eArgError, "empty file name");
    • }

    • NODE *n;

    if ((tbl = RCLASS_IV_TBL(mod)) && st_lookup(tbl, id, &av) && av != Qundef)
        @@ -1348,8 +1344,61 @@ rb_autoload(VALUE mod, ID id, const char
        DATA_PTR(av) = tbl = st_init_numtable();
    }

    • n = rb_node_newnode(NODE_MEMO, (VALUE)func, arg1, arg2);

    • st_insert(tbl, id, (st_data_t)n);
    +}

+static VALUE
+autoload_load(VALUE klass, ID id, VALUE file, VALUE safe)
+{
    • return rb_require_safe(file, (int)safe);
    +}

+static void
+check_autoload_id(ID id)
+{
    • if (!rb_is_const_id(id)) {
    • rb_raise(rb_eNameError, "autoload must be constant name: %s", rb_id2name(id));
    • }
    +}

+void
+rb_autoload(VALUE mod, ID id, const char *file)
+{
    • VALUE fn;

    • check_autoload_id(id);
    • if (!file || !*file) {
    • rb_raise(rb_eArgError, "empty file name");
    • }
    • fn = rb_str_new2(file);
      FL_UNSET(fn, FL_TAINT);
      OBJ_FREEZE(fn);

    • st_insert(tbl, id, (st_data_t)rb_node_newnode(NODE_MEMO, fn, rb_safe_level(), 0));

    • autoload_callback(mod, id, autoload_load, fn, (VALUE)rb_safe_level());
    +}

+void

```

```
+rb_autoload_callback(VALUE mod, ID id, rb_autoload_func *func, VALUE arg1, VALUE arg2)
```

```
+{  
    • check_autoload_id(id);  
    • autoload_callback(mod, id, func, arg1, arg2);  
    +}
```

```
+static VALUE
```

```
+autoload_proc_call(VALUE mod, ID id, VALUE proc, VALUE arg)
```

```
+{  
    • VALUE args[3];  
    • args[0] = mod;  
    • args[1] = ID2SYM(id);  
    • args[2] = arg;  
    • return rb_proc_call_with_block(proc, 3, args, Qnil);  
    +}
```

```
+void
```

```
+rb_autoload_proc(VALUE mod, ID id, VALUE proc, VALUE arg)
```

```
+{  
    • rb_autoload_callback(mod, id, autoload_proc_call, proc, arg);  
    }
```

```
@@ -1382,25 +1431,34 @@ VALUE
```

```
rb_autoload_load(VALUE klass, ID id)
```

```
{  
    • VALUE file;  
      NODE *load = autoload_delete(klass, id);  
  
    • if (!load || !(file = load->nd_lit)) {  
  
    • if (!load) {  
        return Qfalse;  
    }  
  
    • return rb_require_safe(file, load->nd_nth);  
  
    • return load->u1.cfunc(klass, id, load->u2.value, load->u3.value);  
    }
```

```
static VALUE
```

```
-autoload_file(VALUE mod, ID id)
```

```
+autoload_p(VALUE mod, ID id)
```

```
{  
    VALUE val, file;  
    struct st_table *tbl;  
  
    • st_data_t load;  
  
    • st_data_t data;  
    • NODE *load;  
  
    • if (!st_lookup(RCLASS_IV_TBL(mod), autoload, &val) ||  
        !(tbl = check_autoload_table(val)) || !st_lookup(tbl, id, &load)) {  
  
    • if (!st_lookup(RCLASS_IV_TBL(mod), autoload, &data) ||  
        !(tbl = check_autoload_table(val = (VALUE)data)) ||  
        !st_lookup(tbl, id, &data)) {  
        return Qnil;  
    }  
}
```

- file = ((NODE *)load)->nd_lit;
- load = (NODE *)data;
- if (load->u1.cfunc == autoload_proc_call) {
- /* rb_obj_is_proc() */
- return load->u2.value;
- }
- if (load->u1.cfunc != autoload_load) {
- return Qtrue;
- }
- file = load->u2.value;
- Check_Type(file, T_STRING);
- if (!RSTRING_PTR(file) || !*RSTRING_PTR(file)) {
- @@ -1433,5 +1491,5 @@ rb_autoload_p(VALUE mod, ID id)
- return Qnil;
- }
- return autoload_file(mod, id);
- return autoload_p(mod, id);
- }

```
@@ -1630,5 +1688,5 @@ rb_const_defined_0(VALUE klass, ID id, i
while (tmp) {
if (RCLASS_IV_TBL(tmp) && st_lookup(RCLASS_IV_TBL(tmp), id, &value)) {
```

- [REDACTED]
- [REDACTED]

```
return Qfalse;
return Qtrue;
```

```
--
Nobu Nakada
=end
```

History

#1 - 09/03/2008 06:51 PM - ko1 (Koichi Sasada)

- Assignee set to matz (Yukihiro Matsumoto)

```
=begin
```

```
=end
```

#2 - 09/23/2008 11:27 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

```
=begin
```

the code in the block may not be executed (if the constant is defined anywhere else), so it is not stable.

```
=end
```