Ruby trunk - Feature #4893

Literal Instantiation breaks Object Model

06/17/2011 12:45 AM - lazaridis.com (Lazaridis Ilias)

Status:	Rejected	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

```
#String2.rb
class String
def initialize(val)
self.replace(val)
puts object_id
end
def my_method_test
'has method '
end
end
```

command line

```
$ irb
irb(main):001:0> original = String.new("original")
=> "original"
irb(main):002:0> load "String2.rb"
=> true
irb(main):003:0> altered = String.new("altered")
21878604
=> "altered"
irb(main):004:0> altered.my_method_test
=> "has method "
irb(main):005:0> literal = "literal"
=> "literal"
irb(main):006:0> literal.my_method_test
=> "has method "
irb(main):007:0>
```

The initialize method is an integral part of the class String.
From the moment that "String2.rb" is loaded, the initialize method of class String has been validly redefined.

(The behaviour of the String class within the "irb session" is altered)

The altered initialize method is now an integral part of the class String.

The altered String object behaves as expected (responds to "my_method_test, initialized via redefined initialize method).

The String(Literal) object responds to "my_method_test", but it is was not initialized with the redefined initialize method.

The "Literal Instantiation" calls the original (core-C-level) String initialize method instead of the redefined one (user-language-level).
This *breaks* the object model.

History

#1 - 06/17/2011 01:13 AM - naruse (Yui NARUSE)

- Status changed from Open to Rejected

It's a limitation of current Ruby, not a bug at least.

#2 - 06/17/2011 01:23 AM - judofyr (Magnus Holm)

How exactly do you expect this to work? If the string literal is supposed to
be created with String.new(), what do we pass to String.new()? If we pass a
simple string (that is, a string which hasn't gone through String.new()),
these two behave different:

"Hello World" # => Calls String.new() with a simple string and returns a
full string
String.new("Hello World") # => Calls String.new() with string literal (a
full string) and returns a full string

Beside, I don't see how this breaks the object model. The object model
doesn't state that all created strings must be created through String.new.
In fact, it's possible to side-step the initialization in pure Ruby
too: String.allocate.replace("Hello").

// Magnus Holm

On Thu, Jun 16, 2011 at 17:46, Lazaridis Ilias ilias@lazaridis.com wrote:

> Issue #4893 has been reported by Lazaridis Ilias.
>
> ---
>
> Bug #4893: Literal Instantiation breaks Object Model
> http://redmine.ruby-lang.org/issues/4893
>
> Author: Lazaridis Ilias
> Status: Open
> Priority: Normal
> Assignee:
> Category:
> Target version:
> ruby -v: 1.9.2
>
> #String2.rb
> class String
> def initialize(val)
> self.replace(val)
> puts object_id
> end
> def my_method_test
> 'has method '
> end
> end
>
> ## command line
>
> $ irb
> irb(main):001:0> original = String.new("original")
> => "original"
> irb(main):002:0> load "String2.rb"
> => true
> irb(main):003:0> altered = String.new("altered")
> 21878604
> => "altered"
> irb(main):004:0> altered.my_method_test
> => "has method "
> irb(main):005:0> literal = "literal"
> => "literal"
> irb(main):006:0> literal.my_method_test
> => "has method "
> irb(main):007:0>
>
> -
>
> The initialize method is an integral part of the class String.
> From the moment that "String2.rb" is loaded, the initialize method of

> class String has been validly redefined.
>
> (The behaviour of the String class within the "irb session" is
> altered)
>
> The altered initialize method is now an integral part of the class
> String.
>
> The altered String object behaves as expected (responds to
> "my_method_test, initialized via redefined initialize method).
>
> The String(Literal) object responds to "my_method_test", but it is was
> not initialized with the redefined initialize method.
>
> -
>
> The "Literal Instantiation" calls the original (core-C-level) String
> initialize method instead of the redefined one (user-language-level).
> This *breaks* the object model.
>
> --
> http://redmine.ruby-lang.org

#3 - 06/17/2011 01:42 AM - lazaridis.com (Lazaridis Ilias)

Magnus Holm wrote:

> How exactly do you expect this to work? If the string literal is supposed to
> [...] - (off context)

You should reread the description, experiment a little, and take some time to respond.

And please, if possible, remove the quoted message from your reply on the issue tracking system.

#4 - 06/17/2011 01:45 AM - lazaridis.com (Lazaridis Ilias)

Yui NARUSE wrote:

> It's a limitation of current Ruby, not a bug at least.

From my point of view, it's a defect/bug, but I wan't argue.

If you think that's an "limitation" (instead of a bug), then there *is* an issue (and thus you should not "reject" this issue).

You could add an issue type "Limitation".

I any way, rejecting the issue does not change that there *is* an issue with the literal-instantiation.

#5 - 06/17/2011 01:53 AM - rkh (Konstantin Haase)

On Jun 16, 2011, at 18:42 , Lazaridis Ilias wrote:

> You should reread the description, experiment a little, and take some time to respond.

To rephrase what Magnus said:
Where does the object passed to initialize come from? How has that object been initialized?

Konstantin

#6 - 06/17/2011 02:00 AM - shyouhei (Shyouhei Urabe)

- Status changed from Rejected to Feedback

Post a patch to fix it.

#7 - 06/17/2011 02:23 AM - judofyr (Magnus Holm)

> You should reread the description, experiment a little, and take some time to respond.

I'm not even sure what you want to accomplish with this issue. You
provide some code that you think should behave differently, but you
don't provide any solutions. I thought you wanted to change the
behavior, so I described one simple solution and the issues with it.
Do you have another way to solve this?

> From my point of view, it's a defect/bug, but I wan't argue.

Yes, but matz is the boss here. From his point of view, this is
expected behavior.

> If you think that's an "limitation" (instead of a bug), then there *is* an issue (and thus you should not "reject" this issue).

The fact that something is a limitation doesn't mean that it's an
issue. Ruby can't spawn magical unicorn (a limitation), but it's not
an issue. "Rejected" in the issue tracker means "We won't try to fix
it". They won't try to change this limitation, therefore they reject
it.

> I any way, rejecting the issue does not change that there *is* an issue with the literal-instantiation.

Issues like these are subjective. *You* have an issue with the
literal-instantiation, while matz does not (see ruby-talk).

#8 - 06/17/2011 02:32 AM - lazaridis.com (Lazaridis Ilias)

Shyouhei Urabe wrote:

> Post a patch to fix it.

Maybe I'll do so.

#9 - 06/17/2011 04:00 AM - lazaridis.com (Lazaridis Ilias)

Konstantin Haase wrote:

> On Jun 16, 2011, at 18:42 , Lazaridis Ilias wrote:
>
> > You should reread the description, experiment a little, and take some time to respond.
>
> To rephrase what Magnus said:
> Where does the object passed to initialize come from? How has that object been initialized?

"How has that 'uninitialized object' been allocated?" would be the question.

String#allocate (either called directly or via String#new)

http://www.ruby-doc.org/core/classes/Class.html#M000178

Those are basics.

The issue is, that after the allocation, the *redefined* initialized should be called.

-

There is a public topic available, with some elaborations:

CORE - Literal Instantiation breaks Object Model
http://groups.google.com/group/comp.lang.ruby/browse_frm/thread/fd00b89dc6d3a7fa#
http://www.ruby-forum.com/topic/1893190#new

#10 - 06/17/2011 07:40 AM - matz (Yukihiro Matsumoto)

- Status changed from Feedback to Rejected

Your request has been too vague for me. Your definition of terms such as "object model" seems different from others. Probably the code will tell what you want. I will reopen this issue when a patch is posted

#11 - 06/18/2011 01:12 AM - lazaridis.com (Lazaridis Ilias)

deleted by myself, duplicate post

#12 - 06/18/2011 01:12 AM - lazaridis.com (Lazaridis Ilias)

Yukihiro Matsumoto wrote:

> Your request has been too vague for me.
>
> Your definition of terms such as "object model" seems different from others.

I don't think so:

"Object Model" as in

- Design and implementation of a programming language's OO behaviour (classes, object, inheritance, mixin, methods, attributes, per-instance-behaviour etc.).

Special part of this language's (ruby) "Object Model" is

- Primitive data types (integer, string, ...) are objects, and have related classes (Integer, String)
- Classes (including those of the primitive data types) can be redefined at runtime.

This is a flexible ability, but it has (in the current implementation) a limitation (essentially a bug):

- Limitation: a redefined initialize method is *not* called during "literal instantiation"

> Probably the code will tell what you want.

Mr. Matsumoto.

Will you honestly insist that you have not understood (after the public thread, and the compact irb demonstration) that "what I want" is:

- The *redefined* initialize method *must* be called during literal instantiation (when instantiating objects from literals)?

> I will reopen this issue when a patch is posted

This means that this issue is on "Standby" or "Feedback" and not "Rejected".

I think you abuse your position here, because you take the freedom to process this issue based on your personal feelings or bias, instead of the existent facts.

You should respect my time more (I've already invested more that a week with this issue on a OO user-level, and I need some confirmations prior to continuing on C-core-level.

I have to use the official ruby interpreter, thus it makes no sense for me to invest time for a patch which has no chance to be applied.

(Btw: This modification/patch would be the foundation to further unify the behaviour of objects within the language, especially those which are speed-critical.)

Possibly we can agree on this:

- You confirm that the issue is a limitation in the current implementation, concrete:
  - Limitation: a redefined initialize method is *not* called during "literal instantiation"
- You confirm that you will consider to apply a patch, under the condition that the patch
  - does not introduces a performance loss (or at least only a minimal one, e.g. < 1%)
  - does not break existent behaviour (compatibility)
- You place the issue back on an "Standby" / "Halted" / "Feedback" state (other than "Rejected")

And then I can go on to work on a solution/patch.

#13 - 06/20/2011 11:23 PM - aprescott (Adam Prescott)

On Fri, Jun 17, 2011 at 5:12 PM, Lazaridis Ilias ilias@lazaridis.com wrote:

> "Object Model" as in
>
> * Design and implementation of a programming language's OO behaviour (classes, object, inheritance, mixin, methods, attributes, per-instance-behaviour etc.).
>
> Special part of this language's (ruby) "Object Model" is

> * Primitive data types (integer, string, ...) are objects, and have related classes (Integer, String)
> * Classes (including those of the primitive data types) can be redefined at runtime.
>
> This is a flexible ability, but it has (in the current implementation) a limitation (essentially a bug):
>
> * Limitation: a redefined initialize method is *not* called during "literal instantiation"

Other than some abstract violation of a model, what problem is this
causing? Do you have actual code in a practical situation for which
this "limitation" is an obstacle? What would this solve, and what
would be its direct, realisable benefit?

#14 - 06/25/2011 08:10 PM - lazaridis.com (Lazaridis Ilias)

Yukihiro Matsumoto wrote:

> Your request has been too vague for me. Your definition of terms such as "object model" seems different from others. Probably the code will tell what you want. I will reopen this issue when a patch is posted

Patch follows within the next day. I've worked against 1.9.2p180 (the version I've in use), hope this is ok.

#15 - 06/26/2011 11:01 AM - lazaridis.com (Lazaridis Ilias)

- File String_call_initialize.diff added

Deleted by myself, duplicate post.

#16 - 06/26/2011 11:01 AM - lazaridis.com (Lazaridis Ilias)

- File String_call_initialize.diff added

Yukihiro Matsumoto wrote:

> Your request has been too vague for me. Your definition of terms such as "object model" seems different from others. Probably the code will tell what you want. I will reopen this issue when a patch is posted

Find attached a draft version of the modification (against branches/ruby_1_9_2) for review. The code should be self-explaining. I've tested on windows 7 with a small test-program (functionality) and with "nmake test" (compatibility when unused).

As you can see, the speed overhead (default, String.call_initialize = false) is near zero (one "if" call against "int").

If a user wants to have his alternate "initialize" method called, he just sets "String.call_initialize = true". The initialize method is called, passing the allocated and semi-initialized string object as a parameter. The user can now process the object as needed, thus it becomes full-initialized and identical to string objects instantiated by String#new().

I can invest some time refined the patch further, if needed. Please just let me know your requirements.

#17 - 06/26/2011 11:39 AM - shyouhei (Shyouhei Urabe)

- Status changed from Rejected to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

OK, Ilias did his homework. It's your move matz. Though I don't like the call_initialize= thing (thread unsafe), the idea of calling initialize at string creation is clearly explained in the patch. I think it's worth considering.

#18 - 06/26/2011 12:43 PM - wishdev (John Higgins)

- File strings.rb added

The patch doesn't work....

The 'strings.rb' file I believe is about as simple as we can get. The 'empty_file' attempted to be required in the file is just that - a completely empty file with nothing in it at all.

ruby strings.rb yields

```
true
strings.rb:10:in require_relative': methodinitialize' called on hidden T_STRING object (0x00000000849f18 flags=0x4c005 klass=0x0)
(NotImplementedError)
```

from strings.rb:10:in `'

require fails - load works but require is a no go.

#19 - 06/26/2011 12:56 PM - wishdev (John Higgins)

Actually I stand slightly corrected - require/load seems to work on the empty file - but require_relative does not.

I apologize for the slightly misleading report.

#20 - 06/26/2011 04:54 PM - lazaridis.com (Lazaridis Ilias)

- File StringInit.rb added

John Higgins wrote:

> The patch doesn't work....

The basic functionality works (see a simple test StringInit.rb), although there's work to do for a final version.

Remember that this is a draft version, mainly to demonstrate how it can work and to collect some requirements for a final implementation.

> The 'strings.rb' file I believe is about as simple as we can get. The 'empty_file' attempted to be required in the file is just that - a completely empty file with nothing in it at all.

Good "catch".

The error occurs when "rb_str_new_frozen" is used to create the string internally. Will look into this.

#21 - 06/26/2011 06:36 PM - lazaridis.com (Lazaridis Ilias)

Lazaridis Ilias wrote:

> John Higgins wrote:
>
> > The patch doesn't work....

Seems you were right, essentially it does not work (only in my small scope test)

> The error occurs when "rb_str_new_frozen" is used to create the string internally. Will look into this.

I've to call "initialize" from "rb_str_new" instead from "str_new".

Will post a corrected patch tomorrow, after testing against the whole test-suite.

#22 - 06/26/2011 08:06 PM - lazaridis.com (Lazaridis Ilias)

- File String_call_initialize_v2.diff added

- File bootstraptest_runner_addition.diff added

Lazaridis Ilias wrote:
[...]

> I've to call "initialize" from "rb_str_new" instead from "str_new".
>
> Will post a corrected patch tomorrow, after testing against the whole test-suite.

The v2 patch works on my site, tested with "nmake test" whilst adding a String running_counter (see file "bootstraptest_runner_addition") to the tests.

All tests pass and the counter behaves as expected.

#23 - 06/27/2011 06:47 PM - lazaridis.com (Lazaridis Ilias)

I'd like to test the modification against the sources which are to become 1.9.3 (and which contain the IBM737 transcoder). Would this be "trunk"?

#24 - 06/27/2011 07:11 PM - shyouhei (Shyouhei Urabe)

Not sure about IBM737 but yes, 1.9.3 branch will be created from trunk.

#25 - 06/28/2011 08:49 PM - lazaridis.com (Lazaridis Ilias)

Shyouhei Urabe wrote:

> Not sure about IBM737 but yes, 1.9.3 branch will be created from trunk.

Ok, I'm working now against trunk. "nmake test" with the applied "bootstraptest_runner_addition.diff" passes "nmake test". Can I do more, or am I at the point where I have to wait?

#26 - 06/29/2011 09:31 AM - shyouhei (Shyouhei Urabe)

Please wait matz. Maybe you can nudge him.

#27 - 06/30/2011 09:00 PM - lazaridis.com (Lazaridis Ilias)

Shyouhei Urabe wrote:

> Please wait matz. Maybe you can nudge him.

Would like to avoid to "nudge" him, he seems to be busy with other tasks. I try today the test-all and test-specs, and then I can try the final implementation (for which I'll need most possibly some help).

- call_initialize is set to true, whenever the redefinition of initialize is detected.
- call_initialize is set to false, whenever the original initialize becomes active again.

This would be the ideal implementation, fully transparent to the user. If the definition of methods happens with *one* low-level function/method, then it should be simple.

Maybe you can help me with some information:

a) who is the maintainer of string.c ?
b) is there a standard-hook to include custom test-code before the execution of the test-all / test-specs suite?

#28 - 07/01/2011 02:13 AM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

Introducing a new global status is a very bad idea. It doesn't work
well with threads. Besides that, it makes program behavior more
complex and less understandable. It's not acceptable for new
addition.

```
                              matz.
```

#29 - 07/01/2011 02:12 PM - lazaridis.com (Lazaridis Ilias)

Yukihiro Matsumoto wrote:

> Introducing a new global status is a very bad idea. It doesn't work well with threads.

All threads share the same (global) instance of the String class object, thus the flag "call_initialize" is naturally global.

> Besides that, it makes program behavior more complex and less understandable.

I've described the final implementation, which makes "call_initialize" an internal implementation detail, completely hidden from the user.

Currently, I wanted a confirmation that the patch works *technically*, thus I can go on to the next step (= incremental design/implementation).

> It's not acceptable for new addition.

Why do you place the issue again on "reject", instead of awaiting the final implementation?

#30 - 07/01/2011 10:59 PM - matz (Yukihiro Matsumoto)

Lazaridis Ilias wrote:

> Yukihiro Matsumoto wrote:

> > Introducing a new global status is a very bad idea. It doesn't work well with threads.
>
> All threads share the same (global) instance of the String class object, thus the flag "call_initialize" is naturally global.

So that means the whole idea of having call_initialize is a very bad idea, from my point of view.

> Why do you place the issue again on "reject", instead of awaiting the final implementation?

I ask you to show us a concrete description of you request. You have shown the basic outline of your idea by a patch, which appeared to be a bad idea. As a natural consequence, I rejected. Show me the outline of your so-called "final implementation", by working code, not by words in vain. Then I will "resurrect" the issue again, I promise.

matz.

#31 - 07/02/2011 06:22 PM - lazaridis.com (Lazaridis Ilias)

Yukihiro Matsumoto wrote:

> Lazaridis Ilias wrote:
>
> > Yukihiro Matsumoto wrote:
> >
> > > Introducing a new global status is a very bad idea. It doesn't work well with threads.
> >
> > All threads share the same (global) instance of the String class object, thus the flag "call_initialize" is naturally global.
>
> So that means the whole idea of having call_initialize is a very bad idea, from my point of view.

This means that it's technically/functionally not a problem (although I don't like the current implementation of flag, too).

> > Why do you place the issue again on "reject", instead of awaiting the final implementation?
>
> I ask you to show us a concrete description of you request. You have shown the basic outline of your idea by a patch, which appeared to be a bad idea. As a natural consequence, I rejected.

I start to understand how you use "rejected". In order to avoid to discuss the issue-tracking-process here, I've opened an new issue:

http://redmine.ruby-lang.org/issues/4963

> Show me the outline of your so-called "final implementation", by working code, not by words in vain. Then I will "resurrect" the issue again, I promise.

I still need to verify that this draft implementation does not break existent behaviour. If assumed that passing all "make test" is not enough, and indeed "make test-all" uncovered some problems.

#32 - 07/03/2011 01:35 AM - lazaridis.com (Lazaridis Ilias)

- File test_runner_String_initialize.diff added

Lazaridis Ilias wrote:
[...]

> I still need to verify that this draft implementation does not break existent behaviour. If assumed that passing all "make test" is not enough, and indeed "make test-all" uncovered some problems.

Find attached the modification for test/runner.rb in order to run "make test-all".

Five tests failed, all with the same message: "Insecure: can't set class variable."

test_safe_04(TestERBCore):
test_safe_04(TestERBCoreWOStrScan):
test_safe4(TestException) [P:/sand/rubyi2/sandbox/ruby193/test/ruby/test_exception.rb:243]:
test_exec_recursive(TestObject):
test_taint(TestRegexp) [P:/sand/rubyi2/sandbox/ruby193/test/ruby/test_regexp.rb:495]:

Seems to have to do with the "tainted" mechanism, not sure if the access to the class-variable should be disallowed (the class object String is *not* tainted, only it's instances). But that's not the issue, as it's a detail of the test code.

-

I'll move tomorrow on, implementing the final (production) version. Can someone inform me please about this:

- is there a standard mechanism to detect "singleton-method created/deleted", which works on the C-level?

[please note that the UI here does not allow me to reopen an issue by myself].

#33 - 07/04/2011 09:02 PM - lazaridis.com (Lazaridis Ilias)

- File String_call_initialize_v3.diff added

Find attached a patch which removes the translation-unit-visible "static int call_initialize" flag, and introduces the low-level flag "FL_USER18" of the String class object.

There was no documentation about those flags. "FL_USER18" seemed unused, please let me know if I should use another one.

Lazaridis Ilias wrote:
[...]

> - is there a standard mechanism to detect "singleton-method created/deleted", which works on the C-level?

I've found the "(singleton_)method_added/removed/undefined group of call-backs, those should be usable to implement the last step (automated activation/deactivation of the flag).

-

#34 - 07/04/2011 09:11 PM - lazaridis.com (Lazaridis Ilias)

- File String_call_initialize_v3b.diff added

Please ignore patch "String_call_initialize_v3", it contains a merge-error.

New version: String_call_initialize_v3b

#35 - 07/05/2011 12:28 AM - lazaridis.com (Lazaridis Ilias)

Lazaridis Ilias wrote:
[...]

> I've found the "(singleton_)method_added/removed/undefined group of call-backs, those should be usable to implement the last step (automated activation/deactivation of the flag).

Although it's possible to automatically activate/deactivate the flag, it seems preferable to require that this is done explicitly by the user.

The explicit setting of "String.call_initialize = true|false" ensures that a users knows what he's doing, and that he is aware about the involved speed loss.

The patch "String_call_initialize_v3b.diff" is the suggested final version.

#36 - 07/05/2011 12:44 AM - matz (Yukihiro Matsumoto)

Perhaps you don't (want to) understand my comment. Introducing new global status is not acceptable as String's new feature. Period.

matz.

#37 - 07/05/2011 02:07 AM - lazaridis.com (Lazaridis Ilias)

Yukihiro Matsumoto wrote:

> Perhaps you don't (want to) understand my comment.

Perhaps you don't (want to) understand a technically perfectly valid solution?

> Introducing new global status is not acceptable as String's new feature. Period.

DEFINITION:

What do you mean with "global status"?

Would it be still a "global status", if the flag would be set automatically and internally, without user intervention?

INFLUENCE:

Why is it not acceptable?

What is your rationale?

Can you backup this rationales with test-code?

-

Maybe *you* should now use *code* to express what do you mean with "global status" etc., because your attitude is *very* counter productive and *very* ungentle against a person which has invested nearly two weeks to produce this *technicallyvalid* solution.

You wrote earlier:

"Show me the outline of your so-called "final implementation", by working code, not by words in vain. "

Now I say (using your wording):

Show me the outline of your so-called "global status" problem, by working test-code, not by words in vain.

#38 - 07/05/2011 02:45 AM - matz (Yukihiro Matsumoto)

|DEFINITION:
|
|What do you mean with "global status"?

That something you implemented by the C global variable in this case.

|Would it be still a "global status", if the flag would be set automatically and internally, without user intervention?

Depends on the automatic set strategy, which you haven't told me.

|INFLUENCE:
|
|Why is it not acceptable?

The global status tends to make program behavior unpredictable,
especially under threading environment.

```
                            matz.
```

#39 - 07/05/2011 10:46 PM - lazaridis.com (Lazaridis Ilias)

Yukihiro Matsumoto wrote:

> |DEFINITION:
> |
> |What do you mean with "global status"?
>
> That something you implemented by the C global variable in this case.

- I do not use a "C global variable", but a class-object variable.
- There is no "global status", but a class-object status.

> |Would it be still a "global status", if the flag would be set automatically and internally, without user intervention?
>
> Depends on the automatic set strategy, which you haven't told me.

I've described the outline.

But it would be nice if you would tell me a strategy, which would make you accept that it's not a "global status".

Possibly this way I can understand your interpretation "global status".

> |INFLUENCE:
> |
> |Why is it not acceptable?

> The global status tends to make program behavior unpredictable,
> especially under threading environment.

I understand that, but this is not the case here:

VARIABLE:

- I don't use *any* global variable (but an internal class-object scope variable)
- I use the same (string-)class-object scope flags that are used *excessively* within the existent ruby source-codes.

STATUS:

- The status is class-object-bound (exactly as the status "redefined initialize method exists" is)
- The test-all passes (with an activated redefined String#initialize)

At this point, if you still have objections, I ask you friendly to demonstrate them with test-code. This issue is far to deep to be handled with words.

[And may I remind you that this issue is still on status "rejected". This is not the way issues are processed: if they exist, they are open/assigned/feedback, until rejected/closed.]

#40 - 07/05/2011 11:23 PM - now (Nikolai Weibull)

On Tue, Jul 5, 2011 at 15:46, Lazaridis Ilias ilias@lazaridis.com wrote:

> - I do not use a "C global variable", but a class-object variable.
> - There is no "global status", but a class-object status.

The point is that it’s still a global.

But that’s really beside the point. Why even have
#call_initialize/#call_initialize=? Simply always invoke #initialize
instead:

diff --git a/string.c b/string.c
index 711918b..8bdf650 100644
--- a/string.c
+++ b/string.c
@@ -409,7 +409,11 @@ str_new(VALUE klass, const char *ptr, long len)
VALUE
rb_str_new(const char *ptr, long len)
{

- return str_new(rb_cString, ptr, len);
- VALUE str = str_new(rb_cString, ptr, len); +
- rb_obj_call_init(str, 1, &str); +
- return str; }

VALUE

I suspect that this modification (in whatever form it eventually ends
up being) should also be done to rb_str_new_with_class().

#41 - 07/06/2011 05:17 PM - matz (Yukihiro Matsumoto)

|VARIABLE:
|* I don't use *any* global variable (but an internal class-object scope variable)
|* I use the same (string-)class-object scope flags that are used *excessively* within the existent ruby source-codes.

OK, you don't use the C global variable any longer. But still uses
global status, that I mean a line of

String.call_initialize=true

in anywhere in the program can change the whole behavior of whole
program so much. Comparing to other usage of global statuses (some of
which I regret), influence of this change is too huge.

|At this point, if you still have objections, I ask you friendly to demonstrate them with test-code. This issue is far to deep to be handled with words.

Did you measured the performance?

```
                                matz.
```

#42 - 07/07/2011 05:26 AM - lazaridis.com (Lazaridis Ilias)

Yukihiro Matsumoto wrote:

> |VARIABLE:
> |* I don't use *any* global variable (but an internal class-object scope variable)
> |* I use the same (string-)class-object scope flags that are used *excessively* within the existent ruby source-codes.
>
> OK, you don't use the C global variable any longer. But still uses
> global status, that I mean a line of
>
> String.call_initialize=true
>
> in anywhere in the program can change the whole behavior of whole
> program so much.

This is really a (class)local status, see this issue subjecting terminology:

http://redmine.ruby-lang.org/issues/4984

> Comparing to other usage of global statuses (some of
> which I regret), influence of this change is too huge.

This change has a trivial influence, as nothing can go wrong. Even if the flag is set, this just means: the method-lookup will be used. The worst thing that can happen is, that you redefine initialize, and forget to set the flag.

See it otherwise:

You had good reasons to bypass the method-lookup by using the internal initialize method directly (resulted in higher execution speed).

Essentially you've introduced an invisible (class)local status, something like "String.speedy_initialize=true".

But if a user says: "I don't care about speed-loss, I want my redefined method to be called", then he must have access to this flag. That's a property of the class String, like any other one. This bit is nothing more than a simplified cache-mechanism, to speed-up the method-lookup. An internal implementation detail, bound to the class object.

> |At this point, if you still have objections, I ask you friendly to demonstrate them with test-code. This issue is far to deep to be handled with words.
>
> Did you measured the performance?

I could not measure any speed difference with the call inactive (call_initialize=false), as it's minimal. Thus unused there's no difference.

With the call active, there is of course speed loss (100% and more), but I've already some ideas to reduce the speed loss. For this I need to wait some time, to be more secure with the internals.

In any way: the user who needs this feature should understand the speed issues.

#43 - 07/07/2011 07:51 AM - matz (Yukihiro Matsumoto)

|This is really a (class)local status, see this issue subjecting terminology:
|
|http://redmine.ruby-lang.org/issues/4984

I am sorry but I don't understand what you mean by issue #4984. Both
global status and local status has String.call_initialize in examples.

Class objects can be accessed from everywhere. A modifiable attribute
of a globally accessible object is global status, in my terminology.

|This change has a trivial influence, as nothing can go wrong. Even if the flag is set, this just means: the method-lookup will be used. The worst thing that can happen is, that you redefine initialize, and forget to set the flag.

I understand you have different criteria from me. At least, I hate
that "worst case".

|> Did you measure the performance?
|
|I could not measure any speed difference with the call inactive (call_initialize=false), as it's minimal. Thus unused there's no difference.
|
|With the call active, there is of course speed loss (100% and more), but I've already some ideas to reduce the speed loss. For this I need to wait some time, to be more secure with the internals.

|
|In any way: the user who needs this feature should understand the speed issues.

Thus adding one thing to worry to the language performance? Once the
feature added, that could be "abused" beyond your expectation. What
if that feature is used deep inside of the third party library?
People would blame not only the author of the library, but the
language and me for decreasing performance. I am not ready nor
enthusiastic to accept that kind of complains for this particular
feature.

But Ilias, you are safe. Since no one cares who proposed the feature.
That is the very reason I recommend you to fork the language off from
Ruby. Don't bother us.

By the way, I have noticed that your patch invokes initialize with the
receiver itself, which is not yet initialized, as an argument. That
makes me feel weird. That might satisfy your particular requirement,
but I don't consider it consistent as you claim.

```
                              matz.
```

#44 - 07/09/2011 06:35 PM - lazaridis.com (Lazaridis Ilias)

Yukihiro Matsumoto wrote:

> |This is really a (class)local status, see this issue subjecting terminology:
> |
> |http://redmine.ruby-lang.org/issues/4984
> [...]

Commented there.

> |This change has a trivial influence, as nothing can go wrong. Even if the flag is set, this just means: the method-lookup will be used. The worst thing that can happen is, that you redefine initialize, and forget to set the flag.
>
> I understand you have different criteria from me. At least, I hate
> that "worst case".

Different criteria, possibly.

- I "hate" that the current 1.9.2 does not call a redefined initialize at all.
- It would be ok if 1.9.x would not call a redefined initialize, because I forgot to set the flag.

Anyway, this is an implementation detail. If you really hate this, the flag could become invisible to the user (automated setting).

> |> Did you measure the performance?
> |
> |I could not measure any speed difference with the call inactive (call_initialize=false), as it's minimal. Thus unused there's no difference.
> |
> |With the call active, there is of course speed loss (100% and more), but I've already some ideas to reduce the speed loss. For this I need to wait some time, to be more secure with the internals.
> |
> |In any way: the user who needs this feature should understand the speed issues.
>
> Thus adding one thing to worry to the language performance? Once the
> feature added, that could be "abused" beyond your expectation. What
> if that feature is used deep inside of the third party library?
> People would blame not only the author of the library, but the
> language and me for decreasing performance. I am not ready nor
> enthusiastic to accept that kind of complains for this particular
> feature.
> But Ilias, you are safe. Since no one cares who proposed the feature.

Based on your reasoning, you should remove the "RUBY_EVENT_*" mechanism, as it can be "used deeply inside a third party library", decreasing the *overall* speed of the interpreter.

The object-model allows Classes to be "reopened", even the classes from the build-in types. This can be "used deeply inside a third party library" to reduce the speed of the overall language. Thus you should drop the whole object-model, too?

Anyway, I understand your reasoning, and I would possibly agree, if this issue would be a "feature request".

It's not. This issue is about to make the interpreter feature-complete, to remove a limitation of the implementation.

The mentioned speed loss for this issue is for a naked literal-instantiation within a loop, not for a natural application. The method-lookup alone costs 35% (if I measured correct).

In other words: the flag would become redundant, if you would say: "Ok, we do the method-lookup, at the speed-cost of 35%, being this way consistent with the object-model". 35% is much speed, an thus, I introduced the flag.

> That is the very reason I recommend you to fork the language off from
> Ruby. Don't bother us.

If I had a goal: increase throughput of web-applications by 100%, then I would make a fork, because that would mean to do an overall refactoring / rework and a partial reimplementation.

But here I really just want to remove a small limitation of the current implementation.

> By the way, I have noticed that your patch invokes initialize with the
> receiver itself, which is not yet initialized, as an argument.

Please look again at the code. It is initialized.

I've not removed the existent low-level initialization. The object is internally initialized, and is passed to the redefined initialized method for further processing/initialization. This way I didn't need to create a temporal internal string object.

An implementation detail, which can be changed (although it seems not necessary).

> That
> makes me feel weird. That might satisfy your particular requirement,
> but I don't consider it consistent as you claim.

It's more consistent than the existent implementation, and it passes the test-all.

And you should know that providing full consistency would need a rework/refactoring of the string.c and other code units (removed duplicated code, clarify structure, ...).

In other words: the current implementation is not consistent. Adding one missing feature (this issue) cannot make it consistent.

.

#45 - 07/10/2011 03:35 PM - naruse (Yui NARUSE)

- Tracker changed from Bug to Feature

- Status changed from Rejected to Assigned

- Priority changed from Normal to 3

Matz, this is long-term issue.
Can you handle other issues before you handle this?

See http://redmine.ruby-lang.org/projects/ruby-19/issues?query_id=72

#46 - 07/13/2011 06:10 AM - lazaridis.com (Lazaridis Ilias)

=begin
related issue #5015

=end

#47 - 07/13/2011 08:18 AM - lazaridis.com (Lazaridis Ilias)

- File String_call_initialize_v4.diff added

Attached an updated patch, where the flag becomes an internal implementation detail, not accessible from outside.

test-all passes, having this code within test/runner.rb active:

class String
@@running_counter = 0
alias :orig_initialize :initialize
def initialize(*args)
orig_initialize(*args)

@@running_counter += 1
end
def self.running_counter
@@running_counter
end
end

#48 - 07/14/2011 02:40 AM - lazaridis.com (Lazaridis Ilias)

Although the latest version (v4) works fine, I dislike to use the callbacks to set/clear the flag.

I have "fuzzily" the perfect implementation in mind, but it will take a few days until it becomes clear. Until then, I would be "glad" if someone detects a problem with the existent solution.

#49 - 07/14/2011 03:37 AM - wishdev (John Higgins)

Lazaridis Ilias wrote:

> Although the latest version (v4) works fine, I dislike to use the callbacks to set/clear the flag.
>
> I have "fuzzily" the perfect implementation in mind, but it will take a few days until it becomes clear. Until then, I would be "glad" if someone detects a problem with the existent solution.

You are hijacking the callbacks - if anyone implemented any of the callbacks your implementation fails because your callbacks won't fire anymore

#50 - 07/14/2011 03:53 AM - judofyr (Magnus Holm)

YARV already has flags to see if methods are redefined (used for fast
numeric operations). Maybe you could use them? See
vm_init_redefined_flag(void) in vm.c.

#51 - 07/17/2011 09:10 AM - lazaridis.com (Lazaridis Ilias)

John Higgins wrote:
[...]

> You are hijacking the callbacks - if anyone implemented any of the callbacks your implementation fails because your callbacks won't fire anymore

You're right, that's one reason I dislike to use the callbacks.

#52 - 07/17/2011 09:17 AM - lazaridis.com (Lazaridis Ilias)

Magnus Holm wrote:

> YARV already has flags to see if methods are redefined (used for fast
> numeric operations). Maybe you could use them? See
> vm_init_redefined_flag(void) in vm.c.

This sounds plausible. But due to the many delays, I've not much energy/focusation left for this issue, and thus I avoid to change the taken path. But I'll take for sure a look later.

#53 - 07/17/2011 11:56 AM - lazaridis.com (Lazaridis Ilias)

- File String_call_initialize_v5.diff added

Not "perfect", but should be stable and adoptable to other classes.

Needs possibly some renaming, and finally a review of Mr. Holms tip subjecting YARV/vm_init_redefined_flag(void). But once more, I need to take distance for some days.

#54 - 07/17/2011 12:01 PM - lazaridis.com (Lazaridis Ilias)

- File String_call_initialize_v5b.diff added

#55 - 07/18/2011 03:06 AM - lazaridis.com (Lazaridis Ilias)

Lazaridis Ilias wrote:

> Not "perfect", but should be stable and adoptable to other classes.
>
> Needs possibly some renaming,

rb_set_call_flags -> rb_set_redefined_flags
rb_obj_call_init_fast -> rb_call_init_if_redefined

This name change "clarifies" that the flag belongs not to the class-object, but to the method-object (method-struct). So it's going "naturally" one step closer to the suggestion below:

> and finally a review of Mr. Holms tip subjecting YARV/vm_init_redefined_flag(void). But once more, I need to take distance for some days.

Now, distance again!

#56 - 07/18/2011 03:19 AM - lazaridis.com (Lazaridis Ilias)

=begin
Note: I've used tab-4 in the latest patch (v5b) by accident. See the issue #5034
=end

#57 - 07/21/2011 02:24 PM - lazaridis.com (Lazaridis Ilias)

- File String_call_initialize_v6.diff added

This one should do the work fine.

Further optimization / normalization (and thus increase of the overall method-handling consistency) depends on some refactoring (and slight redesign) of existent units, especially vm_method.c. I'll possibly file two or three issues to give an overview of the necessary tasks.

#58 - 07/24/2011 02:13 PM - lazaridis.com (Lazaridis Ilias)

=begin
Related issue: #5088 (Refactor and Document vm_method.c / method.h)
=end

#59 - 07/24/2011 10:23 PM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Rejected

Calling a hook from rb_str_new() is insane.

#60 - 07/24/2011 11:55 PM - lazaridis.com (Lazaridis Ilias)

Nobuyoshi Nakada wrote:

> Calling a hook from rb_str_new() is insane.

Mr. Nakada, when the influence of the alcohol goes away, look once more at the code.

Possibly you'll see that I don't call any hooks. It's the existent code, not my modification.

In any way: it's no reason to once more place the issue on "rejected", this starts to become very annoying.

#61 - 08/02/2011 02:06 AM - lazaridis.com (Lazaridis Ilias)

Lazaridis Ilias wrote:

> Nobuyoshi Nakada wrote:
>
>> Calling a hook from rb_str_new() is insane.
>
> Mr. Nakada, when the influence of the alcohol goes away, look once more at the code.
>
> Possibly you'll see that I don't call any hooks. It's the existent code, not my modification.
>
> In any way: it's no reason to once more place the issue on "rejected", this starts to become very annoying.

Can please someone with access reopen the issue?

#62 - 09/21/2011 04:11 PM - lazaridis.com (Lazaridis Ilias)

=begin

related issue: #5346

=end

Files

String_call_initialize.diff	1.73 KB	06/26/2011	lazaridis.com (Lazaridis Ilias)
String_call_initialize.diff	1.73 KB	06/26/2011	lazaridis.com (Lazaridis Ilias)
strings.rb	158 Bytes	06/26/2011	wishdev (John Higgins)
StringInit.rb	851 Bytes	06/26/2011	lazaridis.com (Lazaridis Ilias)
String_call_initialize_v2.diff	1.58 KB	06/26/2011	lazaridis.com (Lazaridis Ilias)
bootstraptest_runner_addition.diff	1.36 KB	06/26/2011	lazaridis.com (Lazaridis Ilias)
test_runner_String_initialize.diff	1.04 KB	07/03/2011	lazaridis.com (Lazaridis Ilias)
String_call_initialize_v3.diff	1.9 KB	07/04/2011	lazaridis.com (Lazaridis Ilias)
String_call_initialize_v3b.diff	1.62 KB	07/04/2011	lazaridis.com (Lazaridis Ilias)
String_call_initialize_v4.diff	2.45 KB	07/13/2011	lazaridis.com (Lazaridis Ilias)
String_call_initialize_v5.diff	2.44 KB	07/17/2011	lazaridis.com (Lazaridis Ilias)
String_call_initialize_v5b.diff	2.44 KB	07/17/2011	lazaridis.com (Lazaridis Ilias)
String_call_initialize_v6.diff	3.3 KB	07/21/2011	lazaridis.com (Lazaridis Ilias)