Ruby trunk - Bug #7097

Thread locals don't work inside Enumerator

10/02/2012 02:58 AM - tenderlovemaking (Aaron Patterson)

Status:	Closed		
Priority:	Normal		
Assignee:	ko1 (Koichi Sasada)		
Target version:	2.0.0		
ruby -v:	ruby 2.0.0dev (2012-09-25 trunk 37032) [x86_64-darwin12.2.0]	**Backport:**	2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

Description

I set a thread local outside an Enumerator. The Enumerator runs inside the same thread where I set the local. I would expect the thread local to be available since I am in the same thread, but it is not.

Here is a test that shows the problem:

require 'minitest/autorun'

class ThreadLocalBreaks < MiniTest::Unit::TestCase
def test_thread_local_in_enumerator
Thread.current[:foo] = "bar"

```
thread, value = Enumerator.new { |y|
  y << [Thread.current, Thread.current[:foo]]
}.next

assert_equal Thread.current, thread       # passes
assert_equal Thread.current[:foo], value  # fails
```

end
end

History

#1 - 10/02/2012 03:05 AM - kosaki (Motohiro KOSAKI)

- Status changed from Open to Assigned

- Assignee set to ko1 (Koichi Sasada)

I guess it's a side effect to use Fiber in Enumerator internal.

ko1: what do you think?

#2 - 10/02/2012 03:23 AM - Anonymous

- File noname added

On Tue, Oct 02, 2012 at 03:05:17AM +0900, kosaki (Motohiro KOSAKI) wrote:

> Issue #7097 has been updated by kosaki (Motohiro KOSAKI).
>
> Status changed from Open to Assigned
> Assignee set to ko1 (Koichi Sasada)
>
> I guess it's a side effect to use Fiber in Enumerator internal.

Yes, it is. I don't know why Fibers impact thread locals, but I hope
this test demonstrates how it can cause problems.

--
Aaron Patterson
http://tenderlovemaking.com/

#3 - 10/02/2012 03:25 AM - tenderlovemaking (Aaron Patterson)

- File deleted (noname)

#4 - 10/02/2012 03:29 AM - kosaki (Motohiro KOSAKI)

On Mon, Oct 1, 2012 at 2:12 PM, Aaron Patterson
tenderlove@ruby-lang.org wrote:

> On Tue, Oct 02, 2012 at 03:05:17AM +0900, kosaki (Motohiro KOSAKI) wrote:
>
> > Issue #7097 has been updated by kosaki (Motohiro KOSAKI).
> >
> > Status changed from Open to Assigned
> > Assignee set to ko1 (Koichi Sasada)
> >
> > I guess it's a side effect to use Fiber in Enumerator internal.
>
> Yes, it is. I don't know why Fibers impact thread locals, but I hope
> this test demonstrates how it can cause problems.

Thread local variable is pretty misleading name. In fact, ruby only have
a fiber local variable. i.e. Thread.current[:foo] is to write fiber
local variable.

Using 'Thread' class is a just historical reason, IMHO. (and I completely agree
it's pretty misleading)

#5 - 10/02/2012 03:53 AM - kosaki (Motohiro KOSAKI)

On Mon, Oct 1, 2012 at 2:24 PM, KOSAKI Motohiro
kosaki.motohiro@gmail.com wrote:

> On Mon, Oct 1, 2012 at 2:12 PM, Aaron Patterson
> tenderlove@ruby-lang.org wrote:
>
> > On Tue, Oct 02, 2012 at 03:05:17AM +0900, kosaki (Motohiro KOSAKI) wrote:
> >
> > > Issue #7097 has been updated by kosaki (Motohiro KOSAKI).
> > >
> > > Status changed from Open to Assigned
> > > Assignee set to ko1 (Koichi Sasada)
> > >
> > > I guess it's a side effect to use Fiber in Enumerator internal.
> >
> > Yes, it is. I don't know why Fibers impact thread locals, but I hope
> > this test demonstrates how it can cause problems.
>
> Thread local variable is pretty misleading name. In fact, ruby only have
> a fiber local variable. i.e. Thread.current[:foo] is to write fiber
> local variable.
>
> Using 'Thread' class is a just historical reason, IMHO. (and I completely agree
> it's pretty misleading)

see also:

http://bugs.ruby-lang.org/issues/1717
http://bugs.ruby-lang.org/issues/5750

#6 - 10/02/2012 08:23 AM - ko1 (Koichi Sasada)

(2012/10/02 3:12), Aaron Patterson wrote:

> > I guess it's a side effect to use Fiber in Enumerator internal.
> Yes, it is. I don't know why Fibers impact thread locals, but I hope
> this test demonstrates how it can cause problems.

Enumerator uses Fiber to keep a control flow in internal.

Now thread local variables are fiber locals. I agree it is confusing.
But I'm not sure how to solve it.

One idea is:

- Define: Thread#[] -> Thred#current_fiber#[]
- Add: Thread#truly_thread_local_get(key) and Thread#truly_thread_local_set(key, val)

(of course, truly_... is temporal name)

--
// SASADA Koichi at atdot dot net

#7 - 10/02/2012 08:53 AM - ko1 (Koichi Sasada)

(2012/10/02 8:22), SASADA Koichi wrote:

> One idea is:
>
> - Define: Thread#[] -> Thred#current_fiber#[]
> - Add: Thread#truly_thread_local_get(key) and Thread#truly_thread_local_set(key, val)
>
> (of course, truly_... is temporal name)

Another idea:

Add an option to derive Fiber local storage at a Fiber creation.

For example:

```
Thread[:foo] = :bar
Fiber.new(derive_fiber_local_storage: true) do
  Thread[:foo] #=> :bar
end
```

And use Fiber in enumerator with this option true.

--
// SASADA Koichi at atdot dot net

#8 - 10/02/2012 08:53 AM - ko1 (Koichi Sasada)

(2012/10/02 8:32), SASADA Koichi wrote:

> Add an option to derive Fiber local storage at a Fiber creation.

Sorry, derive' is not a correct word. I want to sayshare'.

--
// SASADA Koichi at atdot dot net

#9 - 10/02/2012 09:53 AM - Anonymous

- File noname added

On Tue, Oct 02, 2012 at 08:32:51AM +0900, SASADA Koichi wrote:

> (2012/10/02 8:22), SASADA Koichi wrote:
>
> > One idea is:
> >
> > - Define: Thread#[] -> Thred#current_fiber#[]
> > - Add: Thread#truly_thread_local_get(key) and Thread#truly_thread_local_set(key, val)
> >
> > (of course, truly_... is temporal name)

I prefer this solution. I want a way to set true thread locals.

> Another idea:

> Add an option to derive Fiber local storage at a Fiber creation.
>
> For example:
>
> ```
> Thread[:foo] = :bar
> Fiber.new(derive_fiber_local_storage: true) do
>   Thread[:foo] #=> :bar
> end
> ```
>
> And use Fiber in enumerator with this option true.

I think the "true thread local" solution is better than this because I can use it with existing fiber based code. If a third party library uses Fibers, and I want thread locals, it would not be possible with this solution.

--
Aaron Patterson
http://tenderlovemaking.com/

#10 - 10/02/2012 10:53 AM - ko1 (Koichi Sasada)

(2012/10/02 9:42), Aaron Patterson wrote:

> On Tue, Oct 02, 2012 at 08:32:51AM +0900, SASADA Koichi wrote:
>
> > (2012/10/02 8:22), SASADA Koichi wrote:
> >
> > > One idea is:
> > >
> > > - Define: Thread#[] -> Thred#current_fiber#[]
> > > - Add: Thread#truly_thread_local_get(key) and Thread#truly_thread_local_set(key, val)
> > >
> > > (of course, truly_... is temporal name)
>
> I prefer this solution. I want a way to set true thread locals.
>
> > Another idea:
> >
> > Add an option to derive Fiber local storage at a Fiber creation.
> >
> > For example:
> >
> > ```
> > Thread[:foo] = :bar
> > Fiber.new(derive_fiber_local_storage: true) do
> >   Thread[:foo] #=> :bar
> > end
> > ```
> >
> > And use Fiber in enumerator with this option true.
>
> I think the "true thread local" solution is better than this because I can use it with existing fiber based code. If a third party library uses Fibers, and I want thread locals, it would not be possible with this solution.

I'm afraid that people overuse it rather than Fiber local storage. The reason why Thread local is Fiber local is that most of thread locals should be fiber local.

I think we need to discuss carefully.

Akira-san:
Could you give us your comments?

--
// SASADA Koichi at atdot dot net

#11 - 10/02/2012 02:53 PM - Anonymous

- File noname added

On Tue, Oct 02, 2012 at 10:34:47AM +0900, SASADA Koichi wrote:

> (2012/10/02 9:42), Aaron Patterson wrote:
>
> > On Tue, Oct 02, 2012 at 08:32:51AM +0900, SASADA Koichi wrote:
> >
> > > (2012/10/02 8:22), SASADA Koichi wrote:
> > >
> > > One idea is:
> > >
> > > - Define: Thread#[] -> Thred#current_fiber#[]
> > > - Add: Thread#truly_thread_local_get(key) and Thread#truly_thread_local_set(key, val)
> > >
> > > (of course, truly_... is temporal name)
> >
> > I prefer this solution. I want a way to set true thread locals.
> >
> > Another idea:
> >
> > Add an option to derive Fiber local storage at a Fiber creation.
> >
> > For example:
> >
> > ```
> > Thread[:foo] = :bar
> > Fiber.new(derive_fiber_local_storage: true) do
> >   Thread[:foo] #=> :bar
> > end
> > ```
> >
> > And use Fiber in enumerator with this option true.
>
> I think the "true thread local" solution is better than this because I can use it with existing fiber based code. If a third party library uses Fibers, and I want thread locals, it would not be possible with this solution.

I'm afraid that people overuse it rather than Fiber local storage. The reason why Thread local is Fiber local is that most of thread locals should be fiber local.

I think we need to discuss carefully.

The problem I'm facing is with database connections. Database connections can't be shared among threads (since no parallel access is allowed). I don't want to pass the connection to every function call, so a thread local makes sense for connection storage.

AFAIK, Fibers cannot run in parallel, so it's OK for two Fibers to share a database connection.

Does this make sense? Maybe my assumption is wrong about Fibers.

--
Aaron Patterson
http://tenderlovemaking.com/

#12 - 10/03/2012 02:53 AM - kosaki (Motohiro KOSAKI)

> > For example:
> >
> > ```
> > Thread[:foo] = :bar
> > Fiber.new(derive_fiber_local_storage: true) do
> >   Thread[:foo] #=> :bar
> > end
> > ```
> >
> > And use Fiber in enumerator with this option true.
>
> I think the "true thread local" solution is better than this because I

> can use it with existing fiber based code. If a third party library uses Fibers, and I want thread locals, it would not be possible with this solution.

I'm afraid compatibility breakage. mame-san said:

http://bugs.ruby-lang.org/issues/1717 comment#2

> I can't really remember the rationale, but I think that most of legacy libraries that uses thread-local storage will expect the storage to be also fiber-local.

I'm not surprised old libraries have different assumption with you. share_fiber_local_storage: approach don't break anything.

But real question is, Are such old libraries still living in the wild? Who uses? Who know?

#13 - 10/03/2012 03:45 AM - tenderlovemaking (Aaron Patterson)

- File deleted (noname)

#14 - 10/03/2012 03:46 AM - tenderlovemaking (Aaron Patterson)

- File deleted (noname)

#15 - 10/03/2012 03:47 AM - tenderlovemaking (Aaron Patterson)

- File thread_locals.patch added

Here is a patch to implement thread locals. It just sets a hash on the thread that you can access via Thread#get_local and Thread#set_local. We should probably add a Thread#local? to test for keys.

#16 - 10/03/2012 03:53 AM - Anonymous

- File noname added

On Wed, Oct 03, 2012 at 02:46:02AM +0900, KOSAKI Motohiro wrote:

> > For example:
> >
> > ```
> > Thread[:foo] = :bar
> > Fiber.new(derive_fiber_local_storage: true) do
> >   Thread[:foo] #=> :bar
> > end
> > ```
> >
> > And use Fiber in enumerator with this option true.
>
> I think the "true thread local" solution is better than this because I can use it with existing fiber based code. If a third party library uses Fibers, and I want thread locals, it would not be possible with this solution.
>
> I'm afraid compatibility breakage. mame-san said:

Maybe I wasn't clear. I think we should leave Thread#[] behavior as-is. We can add new methods for accessing thread locals. This will maintain backwards compatibility and give us the new functionality.

I attached a patch to this issue to show what I mean. :-)

--
Aaron Patterson
http://tenderlovemaking.com/

#17 - 10/03/2012 05:59 AM - kosaki (Motohiro KOSAKI)

> I'm afraid compatibility breakage. mame-san said:
>
> > Maybe I wasn't clear. I think we should leave Thread#[] behavior as-is.
> > We can add new methods for accessing thread locals. This will maintain
> > backwards compatibility and give us the new functionality.
>
> Ah, ok. It doesn't have compatibility issue. The last problem is Thread#[] and
> Thread#set_local are very likely confusing. I'd like to wait and see
> other developer's
> opinion. :)
>
> I attached a patch to this issue to show what I mean. :-)

#18 - 10/05/2012 04:53 PM - akr (Akira Tanaka)

2012/10/2 SASADA Koichi ko1@atdot.net:

> Akira-san:
> Could you give us your comments?

I think it is possible to add new methods for thread local storage.
--
Tanaka Akira

#19 - 10/12/2012 03:53 AM - tenderlovemaking (Aaron Patterson)

Akira-san, do you have suggestions for method names?

#20 - 10/12/2012 10:23 AM - akr (Akira Tanaka)

2012/10/12 tenderlovemaking (Aaron Patterson) aaron@tenderlovemaking.com:

> Akira-san, do you have suggestions for method names?

I have no concrete idea.

I guess thread-local variable is used less often than fiber-local variable.
So thread-local methods should be longer than fiber-local methods.
This will be achieved naturally because the fiber-local methods,
Thread#[] and Thread#[]=, are very short.
--
Tanaka Akira

#21 - 10/26/2012 07:19 AM - tenderlovemaking (Aaron Patterson)

- File thread_variables.patch added

I spoke with ko1-san and Usa-san last night, and we thought that thread_variable_(get|set) would be good (similar to instance_variable_(get|set)). I've attached an updated patch to make that change. The documentation includes examples of how the thread local storage and fiber local storage are different.

I added two more methods:

- Thread#thread_variables # => returns a list of the defined variable keys
- Thread#thread_variable? # => returns true if a key is set, otherwise false

Thread#local_variable_(get|set) methods respect the same security and frozen behavior as Thread#[] and Thread#[]=.

I think matz said it's OK to add this. akr (Akira Tanaka), and ko1 (Koichi Sasada) how is this patch? If it's OK, I would like to apply to trunk.

Thanks! :-)

#22 - 10/26/2012 02:53 PM - ko1 (Koichi Sasada)

I don't have any objection.

I recommend that Thread#[] should be reference to
Thread#thread_variable_get, and so on.

Should we add the following new methods to clarify thread local and
fiber local?

Thread#fiber_variable_get <= alias of Thread#[]
Thread#fiber_variable_set <= alias of Thread#[]=
Thread#fiber_variable?
Thread#fiber_variables

Thanks,
Koichi

(2012/10/26 7:20), tenderlovemaking (Aaron Patterson) wrote:

> Issue #7097 has been updated by tenderlovemaking (Aaron Patterson).
>
> File thread_variables.patch added
>
> I spoke with ko1-san and Usa-san last night, and we thought that thread_variable_(get|set) would be good (similar to instance_variable_(get|set)). I've attached an updated patch to make that change. The documentation includes examples of how the thread local storage and fiber local storage are different.
>
> I added two more methods:
>
> - Thread#thread_variables # => returns a list of the defined variable keys
> - Thread#thread_variable? # => returns true if a key is set, otherwise false
>
> Thread#local_variable_(get|set) methods respect the same security and frozen behavior as Thread#[] and Thread#[]=.
>
> I think matz said it's OK to add this. akr (Akira Tanaka), and ko1 (Koichi Sasada) how is this patch? If it's OK, I would like to apply to trunk.
>
> ## Thanks! :-)
>
> Bug #7097: Thread locals don't work inside Enumerator
> https://bugs.ruby-lang.org/issues/7097#change-31586
>
> Author: tenderlovemaking (Aaron Patterson)
> Status: Assigned
> Priority: Normal
> Assignee: ko1 (Koichi Sasada)
> Category:
> Target version:
> ruby -v: ruby 2.0.0dev (2012-09-25 trunk 37032) [x86_64-darwin12.2.0]
>
> I set a thread local outside an Enumerator. The Enumerator runs inside the same thread where I set the local. I would expect the thread local to be available since I am in the same thread, but it is not.
>
> Here is a test that shows the problem:
>
> require 'minitest/autorun'
>
> class ThreadLocalBreaks < MiniTest::Unit::TestCase
> def test_thread_local_in_enumerator
> Thread.current[:foo] = "bar"
>
> ```
> thread, value = Enumerator.new { |y|
>   y << [Thread.current, Thread.current[:foo]]
> }.next
>
> assert_equal Thread.current, thread       # passes
> assert_equal Thread.current[:foo], value  # fails
> ```
>
> end
> end

--
// SASADA Koichi at atdot dot net

#23 - 10/26/2012 03:53 PM - Anonymous

On Fri, Oct 26, 2012 at 02:40:53PM +0900, SASADA Koichi wrote:

> I don't have any objection.

> I recommend that Thread#[] should be reference to
> Thread#thread_variable_get, and so on.

The documentation I added for Thread#thread_variable_get does reference
Thread#[], and each method makes it clear that it's for thread locals
(not fiber locals).

> Should we add the following new methods to clarify thread local and
> fiber local?
>
> Thread#fiber_variable_get <= alias of Thread#[]
> Thread#fiber_variable_set <= alias of Thread#[]=
> Thread#fiber_variable?
> Thread#fiber_variables

I think these aliases would make it clear that we're dealing with Fiber
locals. I don't have a strong opinion though.

--
Aaron Patterson
http://tenderlovemaking.com/

#24 - 10/26/2012 04:53 PM - ko1 (Koichi Sasada)

(2012/10/26 15:41), Aaron Patterson wrote:

> > I recommend that Thread#[] should be reference to
> >
> > Thread#thread_variable_get, and so on.
> > The documentation I added for Thread#thread_variable_get does reference
> > Thread#[], and each method makes it clear that it's for thread locals
> > (not fiber locals).

I wrote wrong sentence. I wanted to write:

The rdoc of Thread# should refer
Thread#thread_variable_get(set).

--
// SASADA Koichi at atdot dot net

#25 - 10/26/2012 11:38 PM - Eregon (Benoit Daloze)

tenderlovemaking (Aaron Patterson) wrote:

> I spoke with ko1-san and Usa-san last night, and we thought that thread_variable_(get|set) would be good (similar to instance_variable_(get|set)). I've attached an updated patch to make that change. The documentation includes examples of how the thread local storage and fiber local storage are different.
>
> I added two more methods:
>
> - Thread#thread_variables # => returns a list of the defined variable keys
> - Thread#thread_variable? # => returns true if a key is set, otherwise false
>
> Thread#local_variable_(get|set) methods respect the same security and frozen behavior as Thread#[] and Thread#[]=.

Sounds great, but I feel the "thread_" prefix is redundant given it is called on a Thread object.
I'm thinking to two alternatives:

- Remove the "thread_" prefix (as usually the thread instance is Thread.current which is very explicit, or the variable name should be clear enough).

  Thread.current.variable_get(:var)
  thread.variable_get(:var)
  worker.variable_get(:var)

versus

Thread.current.thread_variable_get(:var)
thread.thread_variable_get(:var)
worker.thread_variable_get(:var)

Also, I think get/set do not feel very ruby-like, so ...

- Use a API resembling fiber locals:

Thread#locals # => returns an object responding to #[] and #[]= (and maybe #variables and #include?)

```
Thread.current.locals[:var] = some_value
Thread.current.locals[:var]
```

I guess exposing the whole Hash is exposing internal structures and so is unreasonable. But doing so would be intuitive and avoid having 4 new methods for 4 well-known ([],[]=,keys,include?).

#26 - 10/27/2012 02:40 AM - tenderlovemaking (Aaron Patterson)

- File thread_variables.patch added

Updated patch with documentation.

#27 - 10/27/2012 02:53 AM - Anonymous

On Fri, Oct 26, 2012 at 11:38:19PM +0900, Eregon (Benoit Daloze) wrote:

> Issue #7097 has been updated by Eregon (Benoit Daloze).
>
> tenderlovemaking (Aaron Patterson) wrote:
>
> > I spoke with ko1-san and Usa-san last night, and we thought that thread_variable_(get|set) would be good (similar to instance_variable_(get|set)). I've attached an updated patch to make that change. The documentation includes examples of how the thread local storage and fiber local storage are different.
> >
> > I added two more methods:
> >
> > - Thread#thread_variables # => returns a list of the defined variable keys
> > - Thread#thread_variable? # => returns true if a key is set, otherwise false
> >
> > Thread#local_variable_(get|set) methods respect the same security and frozen behavior as Thread#[] and Thread#[]=.
>
> Sounds great, but I feel the "thread_" prefix is redundant given it is called on a Thread object.
> I'm thinking to two alternatives:
>
> - Remove the "thread_" prefix (as usually the thread instance is Thread.current which is very explicit, or the variable name should be clear enough).
>
> Thread.current.variable_get(:var)
> thread.variable_get(:var)
> worker.variable_get(:var)
>
> # versus
>
> Thread.current.thread_variable_get(:var)
> thread.thread_variable_get(:var)
> worker.thread_variable_get(:var)
>
> Also, I think get/set do not feel very ruby-like, so ...

If it becomes cumbersome, we can add aliases later. get/set aren't very
ruby-like, but we have other examples (instance_variable_(get|set)).

> - Use a API resembling fiber locals:
>
> Thread#locals # => returns an object responding to #[] and #[]= (and maybe #variables and #include?)
>
> ```
> Thread.current.locals[:var] = some_value
> Thread.current.locals[:var]
> ```
>
> I guess exposing the whole Hash is exposing internal structures and so is unreasonable. But doing so would be intuitive and avoid having 4 new

> methods for 4 well-known ([],[]=,keys,include?).

I'd rather not expose another object. We can't just expose the hash object because it would have inconsistent behavior with fiber locals:

```
irb(main):001:0> t = Thread.new { }.join
=> #<Thread:0x007fe5f11278c8 dead>
irb(main):002:0> t[:foo] = "bar"
=> "bar"
irb(main):003:0> t["foo"]
=> "bar"
irb(main):004:0>
```

Also, we'd have to figure out what calling freeze on that hash means. Because of these things, we would have to expose a new object that isn't a Hash. Exposing a new object means propagating things like Thread#freeze down to the new object.

If we implement these 4 new method in my patch, our API footprint is only increased by 4 new methods (vs 5 methods + a new object type). If we find later on that exposing an object is a good thing, we can easily implement these 4 methods in terms of the new object and deprecate the 4 methods. Going in the other direction, deprecating an object, is not so easy.

I don't particularly care what the method names are (since we can just alias them later), but I'm firmly against exposing a new object.

--
Aaron Patterson
http://tenderlovemaking.com/

#28 - 10/27/2012 02:53 AM - Anonymous

On Fri, Oct 26, 2012 at 04:41:09PM +0900, SASADA Koichi wrote:

> (2012/10/26 15:41), Aaron Patterson wrote:
>
> > I recommend that Thread#[] should be reference to
> >
> > Thread#thread_variable_get, and so on.
> > The documentation I added for Thread#thread_variable_get does reference
> > Thread#[], and each method makes it clear that it's for thread locals
> > (not fiber locals).
>
> I wrote wrong sentence. I wanted to write:
>
> The rdoc of Thread# should refer
> Thread#thread_variable_get(set).

Sounds good! I'll update the patch.

--
Aaron Patterson
http://tenderlovemaking.com/

#29 - 10/27/2012 05:09 AM - Eregon (Benoit Daloze)

Aaron wrote:

> On Fri, Oct 26, 2012 at 11:38:19PM +0900, Eregon (Benoit Daloze) wrote:
>
> - Remove the "thread_" prefix (as usually the thread instance is Thread.current which is very explicit, or the variable name should be clear enough).
>
> If it becomes cumbersome, we can add aliases later. get/set aren't very
> ruby-like, but we have other examples (instance_variable_(get|set)).

Yeah, I think these examples are not particularly the best, but I see no other good name (well, #[] and #[]= but they are already taken). I'd rather remove the 'thread_' prefix from now, but it makes sense to follow instance_variable_{get,set}.

> I'd rather not expose another object. We can't just expose the hash object because it would have inconsistent behavior with fiber locals:
>
> ```
> irb(main):001:0> t = Thread.new { }.join
> => #<Thread:0x007fe5f11278c8 dead>
> irb(main):002:0> t[:foo] = "bar"
> => "bar"
> irb(main):003:0> t["foo"]
> => "bar"
> irb(main):004:0>
> ```
>
> Also, we'd have to figure out what calling freeze on that hash means. Because of these things, we would have to expose a new object that isn't a Hash. Exposing a new object means propagating things like Thread#freeze down to the new object.

I understand your concern, indeed it's giving too much freedom to the user (and too less to implementers).

> If we implement these 4 new method in my patch, our API footprint is only increased by 4 new methods (vs 5 methods + a new object type). If we find later on that exposing an object is a good thing, we can easily implement these 4 methods in terms of the new object and deprecate the 4 methods. Going in the other direction, deprecating an object, is not so easy.
>
> I don't particularly care what the method names are (since we can just alias them later), but I'm firmly against exposing a new object.

I see, this was a bold suggestion (which I made because I would greatly prefer that API), but it has too much drawbacks and complexity.
Thank you for answering it in details.

#30 - 10/27/2012 05:29 AM - ko1 (Koichi Sasada)

(2012/10/27 5:09), Eregon (Benoit Daloze) wrote:

> Yeah, I think these examples are not particularly the best, but I see no other good name (well, #[] and #[]= but they are already taken). I'd rather remove the 'thread_' prefix from now, but it makes sense to follow instance_variable_{get,set}.

I strongly disagree removing "thread" prefix because this method should not be easy to use (so the name should be long name). Most of case, `Thread#[]' is true selection.

And I think the name "local" is too ambiguous because Ruby has "local variables" (a variable x which is defined by `x = 1', of course you know :)).

For example, Ruby already has Kernel.local_variables', which returns names of local variables in current frame.Thread.local_variables' (or similar naming rule) is confusion (someone can misunderstand that this method returns `local_variables' in thread's current context).

--
// SASADA Koichi at atdot dot net

#31 - 10/27/2012 05:59 AM - Eregon (Benoit Daloze)

On 26 October 2012 22:26, SASADA Koichi ko1@atdot.net wrote:

> (2012/10/27 5:09), Eregon (Benoit Daloze) wrote:
>
> > Yeah, I think these examples are not particularly the best, but I see no other good name (well, #[] and #[]= but they are already taken). I'd rather remove the 'thread_' prefix from now, but it makes sense to follow instance_variable_{get,set}.
>
> I strongly disagree removing "thread" prefix because this method should not be easy to use (so the name should be long name). Most of case, `Thread#[]' is true selection.

In that light, I agree thread_variable* is fine.

> And I think the name "local" is too ambiguous because Ruby has "local variables" (a variable x which is defined by `x = 1', of course you know :)).
>
> For example, Ruby already has Kernel.local_variables', which returns names of local variables in current frame.Thread.local_variables' (or similar naming rule) is confusion (someone can misunderstand that this method returns `local_variables' in thread's current context).

I thought to that too. This might be less of a problem if variables referencing threads have a clear name, but it might be confusing if not.

Sorry for the diversion.

#32 - 10/27/2012 06:23 AM - Anonymous

On Sat, Oct 27, 2012 at 05:09:36AM +0900, Eregon (Benoit Daloze) wrote:

> Issue #7097 has been updated by Eregon (Benoit Daloze).

[snip]

> > If we implement these 4 new method in my patch, our API footprint is only increased by 4 new methods (vs 5 methods + a new object type). If we find later on that exposing an object is a good thing, we can easily implement these 4 methods in terms of the new object and deprecate the 4 methods. Going in the other direction, deprecating an object, is not so easy.
> >
> > I don't particularly care what the method names are (since we can just alias them later), but I'm firmly against exposing a new object.
>
> I see, this was a bold suggestion (which I made because I would greatly prefer that API), but it has too much drawbacks and complexity.
> Thank you for answering it in details.

No problem! Thank you for reviewing my patch. I appreciate the feedback. :-)

--
Aaron Patterson
http://tenderlovemaking.com/

#33 - 10/28/2012 08:26 AM - tenderlovemaking (Aaron Patterson)

- File deleted (noname)

#34 - 10/30/2012 09:16 AM - ko1 (Koichi Sasada)

- Status changed from Assigned to Closed

- Target version set to 2.0.0

This "feature" was introduced.

#35 - 03/27/2017 09:05 PM - ioquatix (Samuel Williams)

I feel like the solution that was reached here is confusing. I feel like the correct outcome would have been

Thread#[] and Thread#[]= are what thread_variable_set and thread_variable_get do.
Fiber#[] and Fiber#[]= are what Thread#[] and Thread#[]= do.

Otherwise, when Fiber is used internally, unexpected behaviour may result. It may even be that certain operations optionally use a Fiber or not, and if that happens, it's even more confusing/unspecified.

Does it make sense to continue this discussion, or are we done, and that's simply how it is?

Files

thread_locals.patch	2.68 KB	10/03/2012	tenderlovemaking (Aaron Patterson)

thread_variables.patch	8.62 KB	10/26/2012	tenderlovemaking (Aaron Patterson)
thread_variables.patch	9.43 KB	10/27/2012	tenderlovemaking (Aaron Patterson)