redmineorg-copy202205 - Vote #69767

Setting model should use Rails.cache instead of class variable

2022/05/09 15:33 - Admin Redmine

ステータス:	Needs feedback	**開始日:**	2011/04/25
優先度:	通常	**期日:**	
担当者:		**進捗率:**	0%
カテゴリ:	Rails support_44	**予定工数:**	0.00時間
対象バージョン:	Candidate for next minor release_33	**作業時間:**	0.00時間
Redmineorg_URL:	https://www.redmine.org/issues/8222	**status_id:**	10
category_id:	44	**tracker_id:**	3
version_id:	33	**plus1:**	0
issue_org_id:	8222	**affected_version:**	
author_id:	347	**closed_on:**	
assigned_to_id:	0	**affected_version_id:**	
comments:	15		

説明

The @Setting@ model uses two local class variables called @@cached_settings@ and @@cached_cleared_on@ to prevent reloading the settings from database and/or @settings.yml@ at every request.

Rails has been supporting different cache implementations for some time now, why not let the @Setting@ model use them as well? Besides the fact that this will help with tenant switching which we've implemented for Planio, it should also improve Redmine's performance when running it with more than processes and a custom cache mechanism, like memcached.

A patch is attached.

journals

(spam)

?? :-)

how were you even able to update this ticket?

This is patch series from https://www.chiliproject.org/issues/350

Chiliproject looks to add a lot of code complexity and we should not need to change tests, Jan's patch, on the other side, is pretty clean.

Should settings be cached separately or could the full Hash of Settings be cached instead ?

There's at list one possible gotcha with this: @Rails.cache@ defaults to a file backend which leads to potential problems if permissions are not set correctly, or, say, the production server disk is full (which never happens hey?).

In those cases you'll have this behaviour:

```
Rails.cache.write("foo", "bar")
Errno::EACCES: Permission denied - /Users/jbbarth/Projects/redmine/redmine-test/tmp/cache/145
...
```

... which will break the application completely, whereas the current code would work.

Also it seems even Rails will issue a "read" operation on the file each time we call @Rails.cache.read/fetch@. Maybe not a good idea on instances with lots of users, this will lead to unecessary IOs.

Otoh I admit it would be nice to rely on @Rails.cache@ if we configure a memory store or a smart database (e.g. memcached or redis).

Ideally, I'd like to make this optional. What do you think ?

Jean-Baptiste Barth wrote:

> There's at list one possible gotcha with this: @Rails.cache@ defaults to a file backend which leads to potential problems if permissions are not set correctly, or, say, the production server disk is full (which never happens hey?).
>
> In those cases you'll have this behaviour: [...]
> ... which will break the application completely, whereas the current code would work.
>
> Also it seems even Rails will issue a "read" operation on the file each time we call @Rails.cache.read/fetch@. Maybe not a good idea on instances with lots of users, this will lead to unecessary IOs.

Oh, I was under the impression that the default store would be MemoryStore, but I can confirm it's FileStore in a vanilla Redmine - you're absolutely right. However, Redmine already uses Rails Caching since r10844. So, I would say, it wouldn't hurt to make more use of it.

> Otoh I admit it would be nice to rely on @Rails.cache@ if we configure a memory store or a smart database (e.g. memcached or redis).
>
> Ideally, I'd like to make this optional. What do you think ?

Hmmmm, personally I would not make this optional. I think the current caching mechanism of the Setting model might be a relic from times where Rails Caching wasn't available, and that should be replaced. To make it more obvious that caching is used there and less confusing for cases where in-memory caching is not the best option, I would really like to move this to Rails.cache.

I agree that a default to MemoryStore for Redmine in general may be a good idea. But then again, it may also be nice just to follow Rails' defaults. I think users who deploy Redmine in production will have their own opinion on Caching anyways and will want to define their favorite cache store. (And if they do, I think they would be confused by the fact that the current Setting cache does not follow their configuration for Rails.cache but uses its own in-memory mechanism...)

I just stumbled upon this issue and had to rebase the change onto the lastest trunk anyway. So I thought, I'd give this a bump.

Attached you may find an updated version of Jan's patch based on r15494.

Addendum: The code currently used at Planio also dups the value fetched from the cache using the following code:

```
def  self.[](name)
  v  =  Rails.cache.fetch("redmine-settings/#{name}")  {  find_or_default(name).value  }
  v.dup  rescue  v
end
```

Talking to coworkers and inspecting the history of the file suggest, that this was done to avoid getting frozen objects out of the cache. Something similar was discussed in the matching "ChiliProject issue":https://www.chiliproject.org/issues/350. I was trying to verify that, but I wasn't able not reproduce that behavior. FileStore, MemoryStore and MemCacheStore all seem to return the objects unfrozen. (FileStore and MemCacheStore return new instances for each each call, MemoryStore returns the same object - similar to the ad-hoc cache currently in place.) But since this patch is already 5 years old, it might as well be the case, that implementations have changed since then. Nevertheless, I wanted to make sure, that everybody is aware of that.
-- --
-- The Rails default :file_store seems to be OK for most use cases but I agree with Jean-Baptiste and would definitely not use it to cache settings that are heavily used (some requests generate > 1000 setting reads). And the problem with the proposed patch is that it doesn't work if the cache is not shared between Redmine processes (eg. if using :memory_store): @redmine-settings-cleared-on@ is not initialized and @Setting.check_cache@ never clears the cache

(@redmine-settings-cleared-on@ would only be set in @Setting.clear_cache@).
-- From a performance point of view: With patch applied:

```
irb(main):004:0> Benchmark.ms {1000.times {Setting.app_title}}
=> 218.401
```

With patch applied and Rails cache store set to :memory_store

```
irb(main):004:0> Benchmark.ms {1000.times {Setting.app_title}}
=> 31.2
```

With current implementation:

```
irb(main):004:0> Benchmark.ms {1000.times {Setting.app_title}}
=> 0.0
```

Thanks a lot for reviewing this discussion and the patch.

Jean-Philippe Lang wrote:

> And the problem with the proposed patch is that it doesn't work if the cache is not shared between Redmine processes (eg. if using :memory_store):
> @redmine-settings-cleared-on@ is not initialized and @Setting.check_cache@ never clears the cache
> (@redmine-settings-cleared-on@ would only be set in @Setting.clear_cache@).

Attached you may find an updated patch, which addresses the bug you described.

Jean-Philippe Lang wrote:

> From a performance point of view: ...

Cannot argue with that. 0.0ms is a very convincing argument ;-) In that case, using @Rails.cache@ would maybe only be beneficial if it's in fact being shared between several processes or app servers. In that case, maybe coming back to Jean-Baptiste's proposal to make this optional could be a way forward.

Of course, if you feel that this doesn't really benefit Redmine, we can also just close this issue...

履歴

#1 - 2022/05/10 17:19 - Admin Redmine

- カテゴリ を Rails support_44 にセット

- 対象バージョン を Candidate for next minor release_33 にセット