redmineorg-copy202205 - Vote #64473

Plugin hooks should have access to the "request" variable

2022/05/09 13:48 - Admin Redmine

ステータス:	Closed	**開始日:**	2009/01/19
優先度:	通常	**期日:**	
担当者:		**進捗率:**	100%
カテゴリ:	Plugin API_20	**予定工数:**	0.00時間
対象バージョン:	0.8.2_8	**作業時間:**	0.00時間
Redmineorg_URL:	https://www.redmine.org/issues/2542	**status_id:**	5
category_id:	20	**tracker_id:**	2
version_id:	8	**plus1:**	0
issue_org_id:	2542	**affected_version:**	
author_id:	3539	**closed_on:**	
assigned_to_id:	5	**affected_version_id:**	
comments:	21		

説明

Plugin Hooks do not have access to the "request" variable; this makes linking from them difficult.

In my environment, we have to contend with the different ways that people access the site:

- IP
- redmine
- redmine.myCompany.com

Referencing the Setting class works if the instance of Redmine is accessed only from one URL.

From Eric:

request would be best. Problem is, request is setup in a Rails Controller but plugin hooks are not Controllers.

Redmine version: 0.8

journals

It would be sweet if something like this could be done so that we don't need (as plugin developers) to access the "request" variable.

```
include ActionController::UrlWriter

default_url_options[:host] = request.env[ 'SERVER_NAME' ]
default_url_options[:port] = request.env[ 'SERVER_PORT' ]
```

Douglas Manley wrote:

> It would be sweet if something like this could be done so that we don't need (as plugin developers) to access the "request" variable.

That would solve the @host@ and @port@ issue (would also need @protocol@) but @request@ has a lot of other useful information. For example, a plugin might want to sniff the User Agent and resent a mobile view.

I'd also love to be able to use @render@ in a hook but I know it's not trivial. Putting "view" code in a class as a string feels so dirty to me.

This is how I would change @hook.rb@:

```
module Helper
```

```
    def call_hook(hook, context={})
        Redmine::Hook.call_hook(hook, {:project => @project, :request => request}.merge(context))
    end
end
```

And Redmine::Hook.call_hook:

```
# Calls a hook.
# Returns the listeners response.
def call_hook(hook, context={})
    returning "" do |response|
        hls = hook_listeners(hook)
        if hls.any?
            request = context[:request]
            if request
                default_url_options[:host] = request.env["SERVER_NAME"]
                default_url_options[:port] = request.env["SERVER_PORT"]
            end
            hls.each do |listener|
                response << listener.send(hook, context).to_s
            end
        end
    end
end
```

This works for me at least. I can use "link_to" in a hook listener and I can query context[:response].

What do you think?

Eric Davis wrote:

> I'd also love to be able to use @render@ in a hook but I know it's not trivial. Putting "view" code in a class as a string feels so dirty to me.

If the @call_hook@ method is extended like this:

```
module Helper
    def call_hook(hook, context={})
        ctx_init = {:project => @project, :controller => controller, :request => request}
        Redmine::Hook.call_hook(hook, ctx_init.merge(context))
    end
end
```

you can use @render_to_string@ in your hook listener method:

```
def view_issues_show_details_bottom(context)
    controller = context[:controller]
    issue = context[:issue]
    controller.send(:render_to_string, :partial => "show_more_data", :locals => {:issue => issue})
end
```

It works, but I wonder if this is the correct way to do it. At least the HTML is in a (partial) view again.

Actually the above method has to be:

```
module Helper
    def call_hook(hook, context={})
        ctx_init = {:project => @project, :request => request}
        ctx_init[:controller] = self.is_a?(ActionController::Base) ? self : controller
        Redmine::Hook.call_hook(hook, ctx_init.merge(context))
    end
end
```

Then it will work for controller and view hooks.

Thomas thanks for the suggestions, I'll see what I can do.

This is how we can make calling @render_to_string@ more straightforward:

In hook.rb

```
class ViewListener < Listener
    [...]
    def self.render_on(hook, options={})
        define_method hook do |context|
            context[:controller].send(:render_to_string, {:locals => context}.merge(options))
        end
    end
end
```

Then we can define a hook that renders its result using a partial view like this:

```
class MyHook < Redmine::Hook::ViewListener
    render_on :view_issues_show_details_bottom, :partial => "show_more_data"
end
```

The partial view will have access to @issue@ because it is given as a parameter in the @call_hook@ call.

This is the complete patch.

I split the "call_hook" helper into two methods. One for controllers and one for views. The controller method returns an array of results (which is more appropriate if I want to handle the result inside a controller). The view method uses "join" to convert the array into a string.

As I just saw it was a bad idea to split the "call_hook" helper. It doesn't work if a controller includes the ApplicationHelper.

So here's a revised patch.

Thomas Löber wrote:

> As I just saw it was a bad idea to split the "call_hook" helper. It doesn't work if a controller includes the ApplicationHelper.
>
> So here's a revised patch.

I'd like to apply this but I'm having a hard time testing it. Could you provide some tests for the different paths of:

- @Redmine::Hook.call_hook@
- @Redmine::Hook::Helper.call_hook@

* @Redmine::Hook::ViewListener.render_on@

I found a way to test a few of the methods and committed your patch with some modifications in r2429 and r2430. I'd still like to add some tests for @Redmine::Hook::Helper.call_hook@ and @Redmine::Hook::ViewListener.render_on@ before closing this issue. You can find the tests at the bottom of @test/unit/lib/redmine/hook_test.rb@.

h3. Commit details

Added request and controller objects to the hooks by default.

The request and controller objects are now added to all hook contexts by default. This will also make url_for work better in hooks by setting up the default_url_options :host, :port, and :protocol.

Finally a new helper method @render_or@ has been added to ViewListener. This will let a hook easily render a partial without a full method definition.

See #2754

This should be finished now. Instead of using the request and setting host, port, and protocol directly I made the @default_url_options@ use @:only_path@. This was a much cleaner implementation and should work for the majority of the plugin cases.

Any non standard case (like Douglas Manley) can set the host, port, and protocol in their plugin itself using the @request@ context that's passed in:

```
context[:request].env["SERVER_NAME"]
context[:request].env["SERVER_PORT"]
```

A big thanks goes to "Peter Suschlik":
http://github.com/edavis10/redmine/commit/5b7a5c39a7da667b1a2718d166a80f1f0ae4d434#comments for the idea.

Implemented in r2489 through r2491

Adding to 0.8.2, since I couldn't reproduce #2754 and I don't think #2754 would be valid anymore after my refactoring.

Great, thank you!

This should work just fine; thanks!

default_url_options[:only_path] ||= true

This will make the default_url_options[:only_path] to be true permanently. So the link in the notification mail will lost it's protocal, host and port.

Chaoqun Zou wrote:

> default_url_options[:only_path] ||= true
>
> This will make the default_url_options[:only_path] to be true permanently. So the link in the notification mail will lost it's protocal, host and port.

Can you provide a test case showing the incorrect behavior? The email urls were being generated correctly for me (see #2754).

Here is a test (in hook_test unit test) for :only_path :

```
    fixtures :issues

    def test_call_hook_will_break_issue_link_in_mail
        issue = Issue.find(1)

        ActionMailer::Base.deliveries.clear
        Mailer.deliver_issue_add(issue)
        mail = ActionMailer::Base.deliveries.last
        puts mail.body

        @hook_module.add_listener(TestLinkToHook)
        @hook_helper.call_hook(:view_layouts_base_html_head)

        ActionMailer::Base.deliveries.clear
        Mailer.deliver_issue_add(issue)
        mail2 = ActionMailer::Base.deliveries.last
        puts mail2.body
```

```
        assert_equal  mail.body,  mail2.body
    end
```

This test will fail before the line 11-12 be commented out.

A diff patch file is attatched.

I believe I fixed the Plugin and Mailer default_url_options in r2522. It turned out to be a tricky bug that was caused by using @default_url_options@ incorrectly.

Both the plugin hooks and Mailer were setting @default_url_options@ incorrectly and causing @ActionContoller::UrlWritter@ to cache the settings on the module (@mattr_accessor@) causing several url generators to fail in either the plugin hooks or the Mailer. I found a good description of why this was happening on "Stackoverflow": http://stackoverflow.com/questions/185573/what-is-mattraccessor-in-a-rails-module/188915#188915.

Thanks for reporting the bug and providing the test case, it helped debugging.

Merged in 0.8-stable in r2558.

related_issues

relates,Closed,2754,Setting up the default_url_options :port in hook will duplicate with the host parameter(which already contains port param) in settings
duplicates,Closed,2649,Need @controller context in plugin hook(view_layouts_base_html_head and view_layouts_base_body_bottom)

履歴

#1 - 2022/05/10 17:26 - Admin Redmine

- カテゴリ を Plugin API_20 にセット

- 対象バージョン を 0.8.2_8 にセット