Redmine - Feature #35415

Unified plugin api esp. regarding patches

2021-06-14 08:41 - Ludwig Meysel

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:	Plugin API	**Estimated time:**	0.00 hour
Target version:			
Resolution:			

Description

Many plugins use different ways to include own lib-code. A common topic I came across is applying patches and consuming hooks. Sometimes the files are require'd having something like

```
unless FoobarController.included_modules.include? MyPlugin::Patches::FooControllerPatch
  FooController.send(:include, MyPlugin::Patches::FooControllerPatch)
end
```

in the bottom.

Sometimes the require is wrapped in Reloader's to_prepare-callback (which seems quite useless to me), other plugins have some kind of apply_patch-method implemented, which does the same as the code above in a more generic manner (IMO the most elegant way to go).

While all these techniques basically work, it often invites to mess up with the auto reloader. When doing some tweaks in the codebase (or even develop my own plugin), I either have to disable foreign plugins or painfully restart the dev-server with every change, when autoloading is not properly implemented. Otherwise I face the often reported problem of "Cannot autoload Constant ..."-Error.

My suggestion is to provide a more unified API for applying patches and consuming hooks.

I could imagine a simple approach e.g. of an appyl_patch function, which forwards the params to an array, which then is processed internally by the ActiveSupport::Reloader.to_prepare callback.

Another (more opinionated) approach is to grab all patches from plugins/.../lib/patches/*.rb, after a plugin was registered, which also was applicable to the hooks.

Anyways, I think there should be some mechanism to provide plugins safely, which do not mess up with the autoloader. This could also catch compatibility-stuff I have also often seen like Rails.version < '5.1' ? ActionDispatch::Callbacks : ActiveSupport::Reloader.to_prepare.

History

#1 - 2022-09-19 02:07 - crypto gopher

1. to_prepare() no longer works as expeted in Rails 6, it is enough to put include() directly in *init.rb* (#36245)
2. when including patch it is redundant to check included_modules, https://ruby-doc.org/core-2.7.6/Module.html#method-i-append_features:

   > Ruby's default implementation is to add the constants, methods, and module variables of this module to mod if this module has not already been added to mod or one of its ancestors.

so your example can be shortened to:

```
FooController.include MyPlugin::Patches::FooControllerPatch
```