Simone (/)'s Blog (/blog)

HOME (/) ABOUT (/ABOUT) BLOG (/BLOG) SPEAKING (/SPEAKING) PROJECTS (/PROJECTS) QUOTES (/QUOTES)

Understanding Ruby and Rails: Delegate

15 December 2009 — 1 Comment

This article targets Rails 2.3 Rails 3

The information contained in this page might not apply to different versions.

This is article is part of my series Understanding Ruby and Rails. Please see the table of contents (/blog/2009/09/inside-ruby-on-rails-reading-source-code/) for the series to view the list of all posts.

Delegation is a quite common practice in Ruby projects, if you consider proxies, mixins and composition as the ingredient of the **Delegation Pattern**.

The Delegation Design Pattern is a technique where an object exposes certain behavior but it actually delegates responsibility for implementing that behavior to an associated object.

A really common technique is to use method_missing to intercepts the calls to undefined method, then forward the call to the right handler. However, this isn't always a good idea (http://twitter.com/jamis/status/6014188374). There are better ways to implement the delegation pattern in Ruby.

The Ruby standard library contains a Delegate module (http://ruby-doc.org/stdlib/libdoc/delegate/rdoc/index.html) that aims to provide support for the Delegation pattern. Sadly, I found it to be way more complex than the traditional approach and I never really used it.

ActiveSupport Delegate module

If your project includes ActiveSupport, and every Rails project does, you have a more clean and easy way to implement the delegation pattern: the Module#delegate extension. It provides a Module#delegate module you can use in your class or in your modules to delegate a specific method to an associate object.

For instance, consider a standard Post model which belongs to a User.

```
class Post
  belongs_to :user
end

class User
  has_many :posts
end
```

You might want a call to post.name to return the name of the user associated to the given post. Normally,
you would create a new name method as follows

```
class Post
  belongs_to :user

def name
    # let's use try to bypass nil-check
    user.try(:name)
  end
end
```

The same code expressed using the delegate method.

```
class Post
  belongs_to :user
  delegate :name, :to => :user, :allow_nil => true
end
```

The <u>delegate</u> method can be used in any context, it's not limited to ActiveRecord models. For example, your custom queue wrapper can delegate to the internal queue implementation some specific methods.

class QueueManager

```
attr_accessor :queue

# Delegates some methods to the internal queue
delegate :size, :clear, :to => :queue

def initialize
    self.queue = []
end

end

m = QueueManager.new
m.size
# => 0
m.clear
# => []
```

Methods can be delegated to instance variables, class variables, or constants by providing them as a symbol. At least one method and the <code>:to</code> option are required.

Options

The delegate method understand some additional options, useful to customize the behavior.

This is my favorite option. The <u>:prefix</u> can be set to <u>true</u> to prefix the delegate method with the name of the object being delegated to. You can also provide a custom prefix.

class Post belongs_to :user delegate :name, :to => :user, :prefix => true # post.user_name delegate :name, :to => :user, :prefix => "author" # post.author_name end

The <u>_:allow_nil</u> option allows the class to delegate the method to an object that might be <u>nil</u>. In this case, a call to the delegated method will return <u>nil</u>. The default behavior is to raise a <code>NoMethodError</code>.

```
class Post
  belongs_to :user
  delegate :name, :to => :user, :prefix => true
end

Post.new.user_name
# raise NoMethodError

class Post
  belongs_to :user
  delegate :name, :to => :user, :prefix => true, :allow_nil => true
end

Post.new.user_name
# => nil
```

The :to it a non-option, because it's mandatory.

Documentation

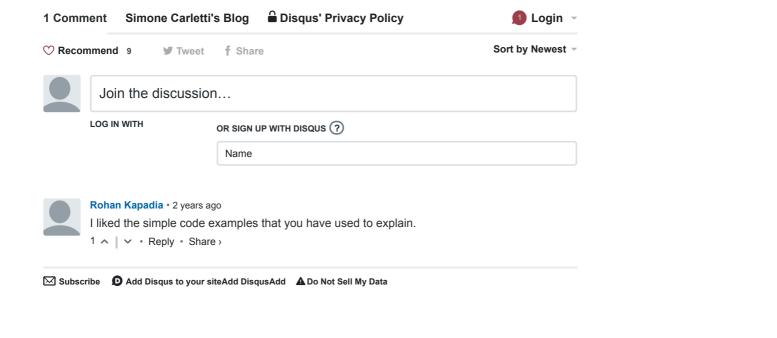
The ActiveSupport <u>delegate</u> extension has a detailed documentation but, unfortunately, it doesn't show up in the main Rails documentation nor you can find it in the ActiveSupport documentation. I suggest you to jump directly into the source code

(https://github.com/rails/rails/blob/master/activesupport/lib/active_support/core_ext/module/delegation.rb), it is worth the effort.

Related Posts

```
07 Jan 2018 The Art of Invisibility book (/blog/2018/01/book-art-of-invisibility/)
27 Dec 2016 How I use StackOverflow (/blog/2016/12/how-i-use-stackoverflow/)
08 Nov 2016 9 years of 1Password (/blog/2016/11/1password-9years/)
```

Filed under: </> Programming (/blog/categories/programming/) — About: Ruby Ruby on Rails ActiveSupport delegation pattern



Copyright © 1985-2020 Simone Carletti