Understanding delegate in Ruby on Rails

モハマド Meraj モラー
Mar 14 · 4 min read

In this article, my focus is to explain Ruby on Rails' `delegate` method — it's what, why, and how with respect to practical example.

https://www.robcarol.com/effectively-delegate/

WHAT?

According to API doc [1]— it "*Provides a delegate class method to easily expose contained objects' public methods as your own.*" And the signature of the method is as below —

```
delegate(*methods, to: nil, prefix: nil, allow_nil: nil) public
```

If used, this delegate class method will expose the set of ***methods** belonging to the target object specified by **to.**

I will explain rest of the arguments as I walk through some examples.

WHY?

Law of Demeter [2] or **principle of least knowledge** is a software design guideline which according to Wikipedia is summarized as —

- Each unit should have only limited knowledge about other units: only units "closely" related to the current unit.
- Each unit should only talk to its friends; don't talk to strangers.
- Only talk to your immediate friends.

delegate methods in Rails help to enforce this law by exposing only necessary methods of containing objects, thereby, making them more easily accessible.

HOW?

Now, for the how part, I will walk through a set of examples. Most of the examples are collected from the book [3] — "**Take My Money: Accepting Payments on the Web**" but of course edited for this article purpose.

Let's assume we have two ActiveRecord models related by has_many association as below —

```
class Event < ActiveRecord::Base
  has_many :performances
end

class Performance < ActiveRecord::Base
  belongs_to :event
end
```

I will launch **rails console** and examine the members of this two models —

> *[3] pry(main)> Event.**column_names***
> *=> ["id", "name", "description", "image_url", "created_at", "updated_at"]*
>
> *[4] pry(main)> Performance.**column_names***
> *=> ["id", "event_id", "start_time", "end_time", "created_at", "updated_at"]*

Now to access an **Event** object's name and description from a **Performance** object, we can normally do as below —

performance = Performance.find(459352183)

*[8] pry(main)> performance.**event.name***
=> "Bleacher Bums"

*[9] pry(main)> performance.**event.description***
=> "A Play About Bleacher Bums"

So, we are navigating through **:event** association here.

But if we use **delegate** method, we can expose **name** and **description** methods more conveniently. Let's modify our **Performance** model to do so.

```
class Performance < ApplicationRecord
  belongs_to :event

  delegate :name, :description, to: :event
end
```

Now, I have exposed **name** and **description** methods of **event** object as if they belong to **performance** object. From rails console, now we can access **event's** name and description as below —

[10] pry(main)> performance.name
=> "Bleacher Bums"

[11] pry(main)> performance.description
=> "A Play About Bleacher Bums"

We no longer have to go via **:event** association.

But what if **Performance** model itself had a **name** method? That's where **prefix** argument comes into play. In that case, I can modify **Performance** model —

```
class Performance < ApplicationRecord
  belongs_to :event

  delegate :name, :description, to: :event, prefix: true

  def name
    "performance"
```

```
  end
end
```

If we specify **prefix: true,** we can access **performance's** name and **event's** name as below —

> *[21] pry(main)> performance.**name***
> *=> "performance"*
>
> *[22] pry(main)> performance.**event_name***
> *=> "Bleacher Bums"*

I can go one step further to specify the prefix as we want as below —

```
class Performance < ApplicationRecord
  belongs_to :event

  delegate :name, :description, to: :event, prefix: :show
end
```

With a prefix specified, we can access **event**'s name and description as below —

> *[25] pry(main)> performance.**show**_name*
> *=> "Bleacher Bums"*
>
> *[26] pry(main)> performance.**show**_description*
> *=> "A Play About Bleacher Bums"*

In case, the object specified by **to** is **nil**, it will raise an exception as noted below —

```
class Performance < ApplicationRecord

  belongs_to :event

  delegate :name, :description, to: :event
end
```

> *[33] pry(main)> Performance.new.event*
> *=> **nil***

> *[34] pry(main)> Performance.new.name*
> ***Module::DelegationError**: Performance#name delegated to event.name, but event is nil: #<Performance id: nil, event_id: nil, start_time: nil, end_time: nil, created_at: nil, updated_at: nil>*

I received a **Module::DelegationError**. To suppress this error and instead receive a **nil** response, I can use the **allow_nil** argument as below —

```
class Performance < ApplicationRecord

  belongs_to :event

  delegate :name, :description, to: :event, allow_nil: true
end
```

> *[36] pry(main)> Performance.new.name*
> *=> **nil***

As a side note, if the receiving object is **not nil** but the method does not exist, we will receive a **NoMethodError** as below —

```
class Performance < ApplicationRecord

  belongs_to :event

  delegate :name, :description, :time, to: :event, allow_nil: true
end
```

[43] pry(main)> performance.**time**
NoMethodError: undefined method `time' for #<Event:0x007fd776780730>

A more practical example

The below excerpt is also from the book [3] —

```
class StripeToken

  attr_accessor :credit_card_number, :expiration_month,
:expiration_year, :cvc

  def initialize(credit_card_number:, expiration_month:,
```

```
  expiration_year:, cvc:)
      @credit_card_number = credit_card_number
      @expiration_month = expiration_month
      @expiration_year = expiration_year
      @cvc = cvc
    end

    def token
      @token ||= Stripe::Token.create(
        card: {
          number: credit_card_number, exp_month: expiration_month,
          exp_year: expiration_year, cvc: cvc})
    end

    delegate :id, to: :token

    def to_s
      "STRIPE TOKEN: #{id}"
    end

    def inspect
      "STRIPE TOKEN #{id}"
    end
  end
```

Here, I am trying to create a Stripe token object. The delegate command transforms **id** to **token.id**. Let's take it for a spin on rails console —

> *[102] pry(main)> StripeToken.new(credit_card_number: "424242424242", expiration_month: 12, expiration_year: 2020, cvc: 111).**id***
> *=> "tok_1GMRoBBN4gTWM2EPi6odYKKA"*

The **id** delegation gives a nice way to access token's id directly. As noted from above code, I can define convenient **to_s** and **inspect** methods using **id** delegation easily. From rails console —

> *[103] pry(main)> StripeToken.new(credit_card_number: "424242424242", expiration_month: 12, expiration_year: 2020, cvc: 111).**to_s***
> *=> "STRIPE TOKEN: tok_1GMRqEBN4gTWM2EPBdehwfuD"*
>
> *[104] pry(main)> StripeToken.new(credit_card_number: "424242424242", expiration_month: 12, expiration_year: 2020, cvc: 111).**inspect***
> *=> "STRIPE TOKEN tok_1GMRqKBN4gTWM2EPUQBcp55o"*

In this article, I tried to clarify Rail's **delegate** method and how readers can put it to real usage. I hope it was enjoyable at least to some people :)

For more elaborate and in depth future technical posts please follow me here or on twitter.

References:

1. https://apidock.com/rails/Module/delegatehttps://apidock.com/rails/Module/delegate
2. http://en.wikipedia.org/wiki/Law_of_Demeter
3. https://www.amazon.com/Take-My-Money-Accepting-Payments/dp/1680501992

Ruby Ruby on Rails Law Of Demeter Programming Software Development