

MARCEL POELKER

Validation Context Behaviour Cheat Sheet (Rails 4)

December 12, 2017 in Tech

Hello world, this is our first tech blog post 😊 We recently encountered some unexpected behaviour with validations and found that most material out there is either shallow or outdated, so we would like to share what we have learned (using Rails v4.2.4 and Ruby v2.3.3.).

Validation contexts in Rails allow you to pick and choose when validations are triggered. But what is actually happening when you call `.valid?(:my_context)`? **It doesn't behave the way you expect...!** Here is a cheat sheet, followed by in-depth examples.

Validation triggered?	Validation			
Method	(without context)	on: :create	on: :update	on: :admin_area
<code>.valid?</code>	Yes	Yes	Yes	Yes
<code>.valid?(:create)</code>	Yes	Yes	No	No
<code>.valid?(:update)</code>	Yes	No	Yes	No
<code>.valid?(:admin_area)</code>	Yes	No	No	Yes
<code>.valid? (:undefined_context)</code>	Yes	No	No	No

In-depth

You can pick and choose validations with validation contexts. For example, you might only want to run a validation when the object is created:

```
class Appointment < ActiveRecord::Base
  validates :starts_in_the_future, on: :create
end
```

For an overview of validation contexts, see this Arkency post

<http://blog.arkency.com/2014/04/mastering-rails-validations-contexts>

You already use validation contexts in every model. Each time an object is created or updated, Rails secretly validates with the built-in create or update contexts: `.valid?(:create)` or `.valid?(:update)`.

Let's write a custom validation context. A validation statement requiring uniqueness : true triggers an extra query, when running the validation. We wanted to reduce the number of queries, so we wanted to only check for uniqueness where on create and update.

Example validation for uniqueness:

```
class Company < ActiveRecord::Base
  validates :name, presence: true, uniqueness: true, on: [:create, :update]
  validates :name, presence: true, on: :admin_area
  validates :address, presence: true
  validates :country_of_incorporation, presence: true
end
```

Use `my_company.valid?(:admin_area)` to trigger the admin area validation context. Sounds simple, but we managed to find a few gotchas! Let's see what is actually happening with a little quiz.

Quiz: Which validations will be triggered by `my_company.valid?(:admin_area)`

Not surprisingly they will trigger validations with `on: :admin_area`:

```
validates :name, presence: true, on: :admin_area
```

But that's not all there is, **it will also** run validations without a specified context!

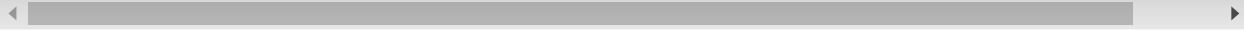
```
validates :address, presence: true
validates :country_of_incorporation, presence: true
```

Gotcha alert! Be careful with custom validation contexts, as you might accidentally trigger unwanted validations.

As I mentioned earlier, the `:create` and `:update` validation contexts are built into Rails. What exactly is happening when we create a new Company?

Quiz: Which validations will be triggered by `new_company.valid?(:create)`

It will trigger validations with `on: :create`.




And... same here, it will also run validations without a specified context.

```
validates :address, presence: true
validates :country_of_incorporation, presence: true
```

If you know that `:create` and `:update` are the default contexts, the following validations are equivalent:

```
validates :address, presence: true, on: [:create, :update]
validates :country_of_incorporation, presence: true, on: [:create, :up
```



However, without this knowledge, it might surprise you. Hence, *gotcha alert!*

Let's come to the plain case of `.valid?`. What is actually happening here?

Quiz: Which validations will be triggered when we do not specify a validation context `my_company.valid?`

You might think that a validation that's clearly labeled as `on: :admin_area` will only be executed under that context. *Gotcha alert!* A plain `valid?` will trigger **all validations**, using the usual Rails validation prioritisation.

One more for the gotcha catchers. What happens if you accidentally mistype your context? What if you ask Rails to use a validation context that is not defined?

Bonus round: Which validations will be triggered by `valid?(:qwertyuiop)` or `valid?(:undefined_context)`

Gotcha alert! **Rails will not raise an error and behave strangely.**

Rails is blind. It will only trigger validations **without a specified context**.

And Rails ignores all validations with a context!

Validation contexts are a powerful tool, but it doesn't always behave as you might expect. If you found this useful to solve your validation puzzle, or if you have feedback, please drop me a line in the comment section.

Written by Donna Zhou



Marcel Poelker

SHARE THIS POST

