Different Ways to Set Attributes in ActiveRecord (Rails 4)

Last revisited on 17 Apr 2014 , originally published on 29 Jan 2014 . Ruby On Rails

Rails 4 allows the developer to change ActiveRecord attributes in various ways. Each one does it slightly differently with sometimes unique side-effects. It's important you understand which method to use, so here's a cheat sheet with in-depth information below.

This article has been updated for Rails 4. Check out the old Rails 3 version (/5-ways-to-set-attributes-in-activerecord-in-rails-3/) if you're using that version.

Cheat Sheet

Method	Uses Default Accessor	Saved to Database	Validations	Callbacks	Touches `updated_at`	Readonly check
attribute= (http://apidock.com/rails/ActiveRecord/AttributeMethods/Write/attribute%3D)	Yes	No	n/a	n/a	n/a	n/a
write_attribute (http://apidock.com/rails/ActiveRecord/AttributeMethods/Write/write_attribute)	No	No	n/a	n/a	n/a	n/a
update_attribute (http://apidock.com/rails/ActiveRecord/Persistence/update_attribute)	Yes	Yes	No	Yes	Yes	Yes
attributes= (http://apidock.com/rails/ActiveRecord/AttributeAssignment/attributes%3D)	Yes	No	n/a	n/a	n/a	n/a
update (http://apidock.com/rails/ActiveRecord/Persistence/update)	Yes	Yes	Yes	Yes	Yes	Yes
update_column (http://apidock.com/rails/ActiveRecord/Persistence/update_column)	No	Yes	No	No	No	Yes
update_columns (http://apidock.com/rails/ActiveRecord/Persistence/update_columns)	No	Yes	No	No	No	Yes
User::update (http://apidock.com/rails/ActiveRecord/Relation/update)	Yes	Yes	Yes	Yes	Yes	Yes
User::update_all (http://apidock.com/rails/v4.0.2/ActiveRecord/Relation/update_all)	No	Yes	No	No	No	No

In Depth

For the following examples we'll set the `name` attribute on the `user` object.

`user.name = "Rob"`

This regular assignment is the most common and easiest to use. It is the default write accessor generated by Rails. The `name` attribute will be marked as dirty and the change will not be sent to the database yet.

You can undo the change by calling `reload!` or save the change to the database by calling `save` .

`user.write_attribute(:name, "Rob")`

This is the method that is called by the default accessor above. An alias for this method is `user[:name] = "Rob"` . It also has a `read_attribute` counterpart.

Just like above, this method does not yet change the attribute in the database. Use this method anywhere you need to bypass the default write accessor above, for example when you want to write a custom `attribute=` writer:

```
def name=(new_name)
  write_attribute(:name, new_name.upcase)
  # This is equivalent:
  # self[:name] = new_name.upcase
end
```

Automatically uppercase the name when set

`user.update_attribute(:name, "Rob")`

This method will change the attribute in the model and pass it straight to the database, without running any validations.

Two gotchas:

- Any *other* changed attributes are also saved to the database.
- Validations are skipped so you could end up with invalid data.

Because of that last quirk it's a good practice to use `update` instead even though you might only want to update one attribute.

`user.attributes = {name: "Rob"}`

This method will set all the attributes you pass it. The changes are not saved to the database. Any attributes you don't pass will be left unchanged. You can also use `assign_attributes` :

```
user.attributes = {name: "Rob", age: 12}
user.assign_attributes {name: "Rob", age: 12}
```

These are equivalent

`user.update(name: "Rob")`

This method used to be called `update_attributes` in Rails 3. It changes the attributes of the model, checks the validations, and updates the record in the database if it validates.

Note that just like `update_attribute` this method also saves other changed attributes to the database.

`user.update_columns(name: "Rob")`

Much like `User::update_all` this executes a direct SQL UPDATE query and bypasses any validations or callbacks. It does check first if any of the columns are marked as `readonly` and if so, raises an exception.

`user.update_column(:name, "Rob")`

This is equivalent to calling `user.update_columns(name: "Rob")` described above.

`User.update(1, name: "Rob")`

Note: this is a class method

This method finds the object with the specified ID and updates it's attributes with the passed in hash. It uses the `User#update` method to do so, so just like that one it validates and runs callbacks, as well as touch the `updated_at` attribute.

You can also pass in an array of ID's and parameters:

```
User.update(
  [1,2,3],
  [
    {name: "Rob"},
    {name: "David", age: 12},
    {age: 15, location: "London"},
  ]
)
```

Note that the second argument is an array of Hashes

`User.update_all(name: "Rob")`

Note: this is a class method

This method runs an SQL UPDATE query that updates the attributes of all objects without running any validations or callbacks. You can also call this method on a scoped relation:

```
User.where(name: "Robbie").update_all(name: "Rob")
```

Update the name of all people called "Robbie"

More

If you want to understand more about these methods I suggest you check out their source code. Each time it's only a couple of lines and it will really broaden your understanding of how Rails works!

David Verhasselt

Senior full-stack engineer with 5 years of experience building web applications for clients all over the world.

Interested in working together?

Find out what I can do for you (/hire/) or get in touch! (mailto:david@crowdway.com)

Like this? Sign up to get regular updates

Email

First name

Subscribe

31 Comments David Verhasselt 1 Login

Recommend 11 Share Sort by Best

Join the discussion…

feixiong · 3 years ago
thanks, it's very useful.
6 · Reply · Share ›

Leung Ho Kuen · 2 years ago
Notice: I think in rails 4.? #write_attribute is renamed to #raw_write_attribute, haven't tried on 4.2.x yet
1 · Reply · Share ›

Fon · 2 years ago
Nice post! thanks!!!
1 · Reply · Share ›

NoviceCodeNinja · 2 years ago
Really great post. Thanks for sharing. What are your thoughts on including the 'cheat sheet' section in the official rails guide? Also, for greater visibility, try submitting to Ruby Weekly (great channel for leads).
1 · Reply · Share ›

Troy Martin · 3 years ago
Thanks this is a very good post. I couldn't figure out what had changed since Rails 3.
1 · Reply · Share ›

tjchambers · a month ago
Love this sheet.

One thing that may be an addition is the updates that come via collections. For instance a has_many : abc where object.delete(abc) does a direct SQL update (nulls the associated id) and appears to bypass callbacks. That just bit me because the update was (of course) not versioned by PaperTrail gem.
· Reply · Share ›

Francisco Quintero · 2 months ago
Thank you very much!
· Reply · Share ›

Jon Abrams · 3 months ago
Has anything changed in Rails 5?

Amazing blog post, btw, it's and indispensable reference for me.
· Reply · Share ›

br3nt · 4 months ago
This keeps saving me :)
· Reply · Share ›

Mirzalazuardi • 5 months ago
Great article!! Bookmarked
Reply • Share ›

Youcef Dahmani • 9 months ago
Thank you, very good post
Reply • Share ›

Lujia • 9 months ago
very helpful! Thanks
Reply • Share ›

Rafael Laurindo • 10 months ago
Thanks so much.
Reply • Share ›

Joshua Harris • a year ago
Excellent post, Thanks!
Reply • Share ›

Damien Jones • a year ago
just wanted to say thanks, i visit this page a couple times a week
Reply • Share ›

Katy • a year ago
It appears as though `assign_attributes` and `attributes=` actually differ.

I'm seeing `assign_attributes` assign virtual attributes successfully, where `attributes=` does not. Does that seem right?
Reply • Share ›

Katy → Katy • a year ago
Never mind. It's literally an alias in Rails 4; should be identical.
1 • Reply • Share ›

funkdified • 2 years ago
Thanks for putting this together :)
Reply • Share ›

Phil Birnie • 2 years ago
Thank you, thank you for this resource. I'm still in the process of learning Rails and this is a great resource and overview.
Reply • Share ›

Casey Robinson • 2 years ago
great article! very helpful
Reply • Share ›

ferdinandrosario • 2 years ago
thanks for sharing !!!
Reply • Share ›

Viduranga Wijesooriya • 2 years ago
Hi,
Is there anyway to stop any other changed attributes from saving to the database when using update_attribute method?
Reply • Share ›

David Mod → Viduranga Wijesooriya • 2 years ago
Hey Viduranga,

Nope there isn't. If you check out the source code of the update_attribute method, you'll see that eventually the #save method is called, which saves all changed attributes.
Reply • Share ›

Eric Brooke • 2 years ago
Great cheat sheet thanks
Reply • Share ›

goodbedford • 3 years ago
Thanks great post. I will start following you.
Reply • Share ›

Gourav Tiwari • 3 years ago
lovely thanks for cheat sheet :)
Reply • Share ›

Danny • 3 years ago
@David Love the cheat sheet. Thanks a lot!
Reply • Share ›

starrychloe • 3 years ago
assign_attributes also sets the _was fields! If you use "model.assign_attributes username: 'cow'" then "model.username_was" is also set to "cow"! This happens without even saving. What good are the _was fields if they are changed immediately upon setting the attributes?

It only happens with #clone

def update
@user = current_user.clone # use @user for short
puts "*** #{current_user.username} #{current_user.username_was}"
@user.assign_attributes(user_params) # store the form so changes are not lost while editing
puts "*** #{current_user.username} #{current_user.username_was}" # Both are updated!

*** test39 test39
*** test3944 test3944
Reply • Share ›

goodbedford ➦ starrychloe • 3 years ago
Thanks for the add-on. Good addition.
Reply • Share ›

rakesh verma • 3 years ago
awesome points thank you very much
Reply • Share ›

Jorge Díaz • 3 years ago
Nice point of view. Great work.
Reply • Share ›

Subscribe Add Disqus to your site Add Disqus Add Privacy DISQUS

David Verhasselt (/)

Articles (/articles/)
About (/about/)
Resume (/resume/)