Different Ways to Set Attributes in ActiveRecord

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

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 if you're using that version.

Cheat Sheet

Method	Uses Default Accessor	Saved to Database	Validations	Callbacks	Touches `updated_at`	Readonly check
attribute=	Yes	No	n/a	n/a	n/a	n/a
write_attribute	No	No	n/a	n/a	n/a	n/a
update_attribute	Yes	Yes	No	Yes	Yes	Yes
attributes=	Yes	No	n/a	n/a	n/a	n/a
update	Yes	Yes	Yes	Yes	Yes	Yes
update_column	No	Yes	No	No	No	Yes
update_columns	No	Yes	No	No	No	Yes
User::update	Yes	Yes	Yes	Yes	Yes	Yes
User::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

I’m a consultant & entrepreneur. I build webapps and optimize Minimal Viable Products for clients all over the world. If you'd like to chat, hit me up on david@crowdway.com.

Like this? Sign up to get regular updates

Email

First name

Subscribe