A Guide for Upgrading Ruby on Rails

January 13, 2015

This guide provides steps to be followed when you upgrade your applications to a newer version of Ruby on Rails. These steps are also available in individual release guides.

1 General Advice

Before attempting to upgrade an existing application, you should be sure you have a good reason to upgrade. You need to balance several factors: the need for new features, the increasing difficulty of finding support for old code, and your available time and skills, to name a few.

1.1 Test Coverage

The best way to be sure that your application still works after upgrading is to have good test coverage before you start the process. If you don't have automated tests that exercise the bulk of your application, you'll need to spend time manually exercising all the parts that have changed. In the case of a Rails upgrade, that will mean every single piece of functionality in the application. Do yourself a favor and make sure your test coverage is good *before* you start an upgrade.

1.2 Ruby Versions

Rails generally stays close to the latest released Ruby version when it's released:

- Rails 3 and above require Ruby 1.8.7 or higher. Support for all of the previous Ruby versions has been dropped officially. You should upgrade as early as possible.
- Rails 3.2.x is the last branch to support Ruby 1.8.7.
- Rails 4 prefers Ruby 2.0 and requires 1.9.3 or newer.

Ruby 1.8.7 p248 and p249 have marshaling bugs that crash Rails. Ruby Enterprise Edition has these fixed since the release of 1.8.7-2010.02. On the 1.9 front, Ruby 1.9.1 is not usable because it outright segfaults, so if you want to use 1.9.x, jump straight to 1.9.3 for smooth sailing.

1.3 The Rake Task

Rails provides the `rails:update` rake task. After updating the Rails version in the Gemfile, run this rake task. This will help you with the creation of new files and changes of old files in an interactive session.

```
$ rake rails:update
   identical  config/boot.rb
       exist  config
    conflict  config/routes.rb
Overwrite /myapp/config/routes.rb? (enter "h" for help) [Ynaqdh]
       force  config/routes.rb
    conflict  config/application.rb
Overwrite /myapp/config/application.rb? (enter "h" for help) [Ynaqdh]
       force  config/application.rb
    conflict  config/environment.rb
...
```

Don't forget to review the difference, to see if there were any unexpected changes.

2 Upgrading from Rails 4.1 to Rails 4.2

2.1 Web Console

First, add `gem 'web-console', '~> 2.0'` to the `:development` group in your Gemfile and run `bundle install` (it won't have been included when you upgraded Rails). Once it's been installed, you can simply drop a reference to the console helper (i.e., `<%= console %>`) into any view you want to enable it for. A console will also be provided on any error page you view in your development environment.

2.2 Responders

`respond_with` and the class-level `respond_to` methods have been extracted to the `responders` gem. To use them, simply add `gem 'responders', '~> 2.0'` to your Gemfile. Calls to `respond_with` and `respond_to` (again, at the class level) will no longer work without having included the `responders` gem in your dependencies:

```
# app/controllers/users_controller.rb

class UsersController < ApplicationController
  respond_to :html, :json

  def show
    @user = User.find(params[:id])
    respond_with @user
  end
end
```

Instance-level `respond_to` is unaffected and does not require the additional gem:

```
# app/controllers/users_controller.rb

class UsersController < ApplicationController
```

```
  def show
    @user = User.find(params[:id])
    respond_to do |format|
      format.html
      format.json { render json: @user }
    end
  end
end
```

See #16526 for more details.

2.3 Error handling in transaction callbacks

Currently, Active Record suppresses errors raised within `after_rollback` or `after_commit` callbacks and only prints them to the logs. In the next version, these errors will no longer be suppressed. Instead, the errors will propagate normally just like in other Active Record callbacks.

When you define a `after_rollback` or `after_commit` callback, you will receive a deprecation warning about this upcoming change. When you are ready, you can opt into the new behavior and remove the deprecation warning by adding following configuration to your `config/application.rb`:

```
config.active_record.raise_in_transactional_callbacks = true
```

See #14488 and #16537 for more details.

2.4 Ordering of test cases

In Rails 5.0, test cases will be executed in random order by default. In anticipation of this change, Rails 4.2 introduced a new configuration option `active_support.test_order` for explicitly specifying the test ordering. This allows you to either lock down the current behavior by setting the option to `:sorted`, or opt into the future behavior by setting the option to `:random`.

If you do not specify a value for this option, a deprecation warning will be emitted. To avoid this, add the following line to your test environment:

```
# config/environments/test.rb
Rails.application.configure do
  config.active_support.test_order = :sorted # or `:random` if you prefer
end
```

2.5 Serialized attributes

When using a custom coder (e.g. `serialize :metadata, JSON`), assigning `nil` to a serialized attribute will save it to the database as `NULL` instead of passing the `nil` value through the coder (e.g. `"null"` when using the `JSON` coder).

2.6 Production log level

In Rails 5, the default log level for the production environment will be changed to `:debug` (from `:info`). To preserve the current default, add the following line to your `production.rb`:

```
# Set to `:info` to match the current default, or set to `:debug` to opt-into
# the future default.
config.log_level = :info
```

2.7 `after_bundle` in Rails templates

If you have a Rails template that adds all the files in version control, it fails to add the generated binstubs because it gets executed before Bundler:

```
# template.rb
generate(:scaffold, "person name:string")
route "root to: 'people#index'"
rake("db:migrate")

git :init
git add: "."
git commit: %Q{ -m 'Initial commit' }
```

You can now wrap the `git` calls in an `after_bundle` block. It will be run after the binstubs have been generated.

```
# template.rb
generate(:scaffold, "person name:string")
route "root to: 'people#index'"
rake("db:migrate")

after_bundle do
  git :init
  git add: "."
  git commit: %Q{ -m 'Initial commit' }
end
```

2.8 Rails HTML Sanitizer

There's a new choice for sanitizing HTML fragments in your applications. The venerable html-scanner approach is now officially being deprecated in favor of `Rails HTML Sanitizer`.

This means the methods `sanitize`, `sanitize_css`, `strip_tags` and `strip_links` are backed by a new implementation.

This new sanitizer uses Loofah internally. Loofah in turn uses Nokogiri, which wraps XML parsers written in both C and Java, so sanitization should be faster no matter which Ruby version you run.

The new version updates `sanitize`, so it can take a `Loofah::Scrubber` for powerful scrubbing. See some examples of scrubbers here.

Two new scrubbers have also been added: `PermitScrubber` and `TargetScrubber`. Read the gem's readme for more information.

The documentation for `PermitScrubber` and `TargetScrubber` explains how you can gain complete control over when and how elements should be stripped.

If your application needs to use the old sanitizer implementation, include `rails-deprecated_sanitizer` in your Gemfile:

```
gem 'rails-deprecated_sanitizer'
```

2.9 Rails DOM Testing

The `TagAssertions` module (containing methods such as `assert_tag`), has been deprecated in favor of the `assert_select` methods from the `SelectorAssertions` module, which has been extracted into the rails-dom-testing gem.

2.10 Masked Authenticity Tokens

In order to mitigate SSL attacks, `form_authenticity_token` is now masked so that it varies with each request. Thus, tokens are validated by unmasking and then decrypting. As a result, any strategies for verifying requests from non-rails forms that relied on a static session CSRF token have to take this into account.

2.11 Action Mailer

Previously, calling a mailer method on a mailer class will result in the corresponding instance method being executed directly. With the introduction of Active Job and `#deliver_later`, this is no longer true. In Rails 4.2, the invocation of the instance methods are deferred until either `deliver_now` or `deliver_later` is called. For example:

```
class Notifier < ActionMailer::Base
  def notify(user, ...)
    puts "Called"
    mail(to: user.email, ...)
  end
end

mail = Notifier.notify(user, ...) # Notifier#welcome is not yet called at this point
mail = mail.deliver_now           # Prints "Called"
```

This should not result in any noticible differnces for most applications. However, if you need some non-mailer methods to be exectuted synchronously, and you were previously relying on the synchronous proxying behavior, you should define them as class methods on the mailer class directly:

```
class Notifier < ActionMailer::Base
  def self.broadcast_notifications(users, ...)
    users.each { |user| Notifier.notify(user, ...) }
  end
end
```

3 Upgrading from Rails 4.0 to Rails 4.1

3.1 CSRF protection from remote `<script>` tags

Or, "whaaat my tests are failing!!!?"

Cross-site request forgery (CSRF) protection now covers GET requests with JavaScript responses, too. This prevents a third-party site from referencing your JavaScript URL and attempting to run it to extract sensitive data.

This means that your functional and integration tests that use

```
get :index, format: :js
```

will now trigger CSRF protection. Switch to

```
xhr :get, :index, format: :js
```

to explicitly test an `XmlHttpRequest`.

If you really mean to load JavaScript from remote `<script>` tags, skip CSRF protection on that action.

3.2 Spring

If you want to use Spring as your application preloader you need to:

1. Add `gem 'spring', group: :development` to your `Gemfile`.
2. Install spring using `bundle install`.
3. Springify your binstubs with `bundle exec spring binstub --all`.

User defined rake tasks will run in the `development` environment by default. If you want them to run in other environments consult the Spring README.

3.3 `config/secrets.yml`

If you want to use the new `secrets.yml` convention to store your application's secrets, you need to:

1. Create a `secrets.yml` file in your `config` folder with the following content:

   ```
   development:
     secret_key_base:

   test:
     secret_key_base:

   production:
     secret_key_base: <%= ENV["SECRET_KEY_BASE"] %>
   ```

2. Use your existing `secret_key_base` from the `secret_token.rb` initializer to set the SECRET_KEY_BASE environment variable for whichever users running the Rails application in production mode. Alternatively, you can simply copy the existing `secret_key_base` from the `secret_token.rb` initializer to `secrets.yml` under the `production` section, replacing '<%= ENV["SECRET_KEY_BASE"] %>'.

3. Remove the `secret_token.rb` initializer.
4. Use `rake secret` to generate new keys for the `development` and `test` sections.
5. Restart your server.

3.4 Changes to test helper

If your test helper contains a call to `ActiveRecord::Migration.check_pending!` this can be removed. The check is now done automatically when you `require 'rails/test_help'`, although leaving this line in your helper is not harmful in any way.

3.5 Cookies serializer

Applications created before Rails 4.1 uses `Marshal` to serialize cookie values into the signed and encrypted cookie jars. If you want to use the new `JSON`-based format in your application, you can add an initializer file with the following content:

```
Rails.application.config.action_dispatch.cookies_serializer = :hybrid
```

This would transparently migrate your existing `Marshal`-serialized cookies into the new `JSON`-based format.

When using the `:json` or `:hybrid` serializer, you should beware that not all Ruby objects can be serialized as JSON. For example, `Date` and `Time` objects will be serialized as strings, and `Hash`es will have their keys stringified.

```
class CookiesController < ApplicationController
  def set_cookie
    cookies.encrypted[:expiration_date] = Date.tomorrow # => Thu, 20 Mar 2014
    redirect_to action: 'read_cookie'
  end

  def read_cookie
    cookies.encrypted[:expiration_date] # => "2014-03-20"
  end
end
```

It's advisable that you only store simple data (strings and numbers) in cookies. If you have to store complex objects, you would need to handle the conversion manually when reading the values on subsequent requests.

If you use the cookie session store, this would apply to the `session` and `flash` hash as well.

3.6 Flash structure changes

Flash message keys are normalized to strings. They can still be accessed using either symbols or strings. Looping through the flash will always yield string keys:

```
flash["string"] = "a string"
flash[:symbol] = "a symbol"
```

```
# Rails < 4.1
flash.keys # => ["string", :symbol]

# Rails >= 4.1
flash.keys # => ["string", "symbol"]
```

Make sure you are comparing Flash message keys against strings.

3.7 Changes in JSON handling

There are a few major changes related to JSON handling in Rails 4.1.

3.7.1 MultiJSON removal MultiJSON has reached its end-of-life and has been removed from Rails.
If your application currently depend on MultiJSON directly, you have a few options:

1. Add 'multi_json' to your Gemfile. Note that this might cease to work in the future

2. Migrate away from MultiJSON by using `obj.to_json`, and `JSON.parse(str)` instead.

Do not simply replace `MultiJson.dump` and `MultiJson.load` with `JSON.dump` and `JSON.load`. These JSON gem APIs are meant for serializing and deserializing arbitrary Ruby objects and are generally unsafe.

3.7.2 JSON gem compatibility Historically, Rails had some compatibility issues with the JSON gem. Using `JSON.generate` and `JSON.dump` inside a Rails application could produce unexpected errors.

Rails 4.1 fixed these issues by isolating its own encoder from the JSON gem. The JSON gem APIs will function as normal, but they will not have access to any Rails-specific features. For example:

```
class FooBar
  def as_json(options = nil)
    { foo: 'bar' }
  end
end

>> FooBar.new.to_json # => "{\"foo\":\"bar\"}"
>> JSON.generate(FooBar.new, quirks_mode: true) # => "\"#<FooBar:
0x007fa80a481610>\""
```

3.7.3 New JSON encoder The JSON encoder in Rails 4.1 has been rewritten to take advantage of the JSON gem. For most applications, this should be a transparent change. However, as part of the rewrite, the following features have been removed from the encoder:

1. Circular data structure detection
2. Support for the `encode_json` hook
3. Option to encode `BigDecimal` objects as numbers instead of strings

If your application depends on one of these features, you can get them back by adding the `activesupport-json_encoder` gem to your Gemfile.

3.7.4 JSON representation of Time objects

`#as_json` for objects with time component (`Time`, `DateTime`, `ActiveSupport::TimeWithZone`) now returns millisecond precision by default. If you need to keep old behavior with no millisecond precision, set the following in an initializer:

```
ActiveSupport::JSON::Encoding.time_precision = 0
```

3.8 Usage of `return` within inline callback blocks

Previously, Rails allowed inline callback blocks to use `return` this way:

```
class ReadOnlyModel < ActiveRecord::Base
  before_save { return false } # BAD
end
```

This behavior was never intentionally supported. Due to a change in the internals of `ActiveSupport::Callbacks`, this is no longer allowed in Rails 4.1. Using a `return` statement in an inline callback block causes a `LocalJumpError` to be raised when the callback is executed.

Inline callback blocks using `return` can be refactored to evaluate to the returned value:

```
class ReadOnlyModel < ActiveRecord::Base
  before_save { false } # GOOD
end
```

Alternatively, if `return` is preferred it is recommended to explicitly define a method:

```
class ReadOnlyModel < ActiveRecord::Base
  before_save :before_save_callback # GOOD

  private
    def before_save_callback
      return false
    end
end
```

This change applies to most places in Rails where callbacks are used, including Active Record and Active Model callbacks, as well as filters in Action Controller (e.g. `before_action`).

See this pull request for more details.

3.9 Methods defined in Active Record fixtures

Rails 4.1 evaluates each fixture's ERB in a separate context, so helper methods defined in a fixture will not be available in other fixtures.

Helper methods that are used in multiple fixtures should be defined on modules included in the newly introduced `ActiveRecord::FixtureSet.context_class`, in `test_helper.rb`.

```
module FixtureFileHelpers
  def file_sha(path)
    Digest::SHA2.hexdigest(File.read(Rails.root.join('test/fixtures', path)))
  end
end
ActiveRecord::FixtureSet.context_class.send :include, FixtureFileHelpers
```

3.10 I18n enforcing available locales

Rails 4.1 now defaults the I18n option `enforce_available_locales` to `true`. This means that it will make sure that all locales passed to it must be declared in the `available_locales` list.

To disable it (and allow I18n to accept *any* locale option) add the following configuration to your application:

```
config.i18n.enforce_available_locales = false
```

Note that this option was added as a security measure, to ensure user input cannot be used as locale information unless it is previously known. Therefore, it's recommended not to disable this option unless you have a strong reason for doing so.

3.11 Mutator methods called on Relation

`Relation` no longer has mutator methods like `#map!` and `#delete_if`. Convert to an `Array` by calling `#to_a` before using these methods.

It intends to prevent odd bugs and confusion in code that call mutator methods directly on the `Relation`.

```
# Instead of this
Author.where(name: 'Hank Moody').compact!

# Now you have to do this
authors = Author.where(name: 'Hank Moody').to_a
authors.compact!
```

3.12 Changes on Default Scopes

Default scopes are no longer overridden by chained conditions.

In previous versions when you defined a `default_scope` in a model it was overridden by chained conditions in the same field. Now it is merged like any other scope.

Before:

```
class User < ActiveRecord::Base
  default_scope { where state: 'pending' }
  scope :active, -> { where state: 'active' }
  scope :inactive, -> { where state: 'inactive' }
end

User.all
# SELECT "users".* FROM "users" WHERE "users"."state" = 'pending'

User.active
```

```
# SELECT "users".* FROM "users" WHERE "users"."state" = 'active'

User.where(state: 'inactive')
# SELECT "users".* FROM "users" WHERE "users"."state" = 'inactive'
```

After:

```
class User < ActiveRecord::Base
  default_scope { where state: 'pending' }
  scope :active, -> { where state: 'active' }
  scope :inactive, -> { where state: 'inactive' }
end

User.all
# SELECT "users".* FROM "users" WHERE "users"."state" = 'pending'

User.active
# SELECT "users".* FROM "users" WHERE "users"."state" = 'pending' AND "users"."state" =
'active'

User.where(state: 'inactive')
# SELECT "users".* FROM "users" WHERE "users"."state" = 'pending' AND "users"."state" =
'inactive'
```

To get the previous behavior it is needed to explicitly remove the `default_scope` condition using `unscoped`, `unscope`, `rewhere` or `except`.

```
class User < ActiveRecord::Base
  default_scope { where state: 'pending' }
  scope :active, -> { unscope(where: :state).where(state: 'active') }
  scope :inactive, -> { rewhere state: 'inactive' }
end

User.all
# SELECT "users".* FROM "users" WHERE "users"."state" = 'pending'

User.active
# SELECT "users".* FROM "users" WHERE "users"."state" = 'active'

User.inactive
# SELECT "users".* FROM "users" WHERE "users"."state" = 'inactive'
```

3.13 Rendering content from string

Rails 4.1 introduces `:plain`, `:html`, and `:body` options to `render`. Those options are now the preferred way to render string-based content, as it allows you to specify which content type you want the response sent as.

- `render :plain` will set the content type to `text/plain`
- `render :html` will set the content type to `text/html`
- `render :body` will *not* set the content type header.

From the security standpoint, if you don't expect to have any markup in your response body, you should be using `render :plain` as most browsers will escape unsafe content in the response for you.

We will be deprecating the use of `render :text` in a future version. So please start using the more precise `:plain`, `:html`, and `:body` options instead. Using `render :text` may pose a security risk, as the content is sent as `text/html`.

3.14 PostgreSQL json and hstore datatypes

Rails 4.1 will map `json` and `hstore` columns to a string-keyed Ruby `Hash`. In earlier versions, a `HashWithIndifferentAcces` was used. This means that symbol access is no longer supported. This is also the case for `store_accessors` based on top of `json` or `hstore` columns. Make sure to use string keys consistently.

3.15 Explicit block use for `ActiveSupport::Callbacks`

Rails 4.1 now expects an explicit block to be passed when calling `ActiveSupport::Callbacks.set_callback`. This change stems from `ActiveSupport::Callbacks` being largely rewritten for the 4.1 release.

```
# Previously in Rails 4.0
set_callback :save, :around, ->(r, &block) { stuff; result = block.call; stuff }

# Now in Rails 4.1
set_callback :save, :around, ->(r, block) { stuff; result = block.call; stuff }
```

4 Upgrading from Rails 3.2 to Rails 4.0

If your application is currently on any version of Rails older than 3.2.x, you should upgrade to Rails 3.2 before attempting one to Rails 4.0.

The following changes are meant for upgrading your application to Rails 4.0.

4.1 HTTP PATCH

Rails 4 now uses `PATCH` as the primary HTTP verb for updates when a RESTful resource is declared in `config/routes.rb`. The `update` action is still used, and `PUT` requests will continue to be routed to the `update` action as well. So, if you're using only the standard RESTful routes, no changes need to be made:

```
resources :users
```

```
<%= form_for @user do |f| %>
```

```
class UsersController < ApplicationController
  def update
    # No change needed; PATCH will be preferred, and PUT will still work.
  end
end
```

However, you will need to make a change if you are using `form_for` to update a resource in conjunction with a custom route using the `PUT` HTTP method:

```
resources :users, do
  put :update_name, on: :member
end

<%= form_for [ :update_name, @user ] do |f| %>

class UsersController < ApplicationController
  def update_name
    # Change needed; form_for will try to use a non-existent PATCH route.
  end
end
```

If the action is not being used in a public API and you are free to change the HTTP method, you can update your route to use `patch` instead of `put`:

`PUT` requests to `/users/:id` in Rails 4 get routed to `update` as they are today. So, if you have an API that gets real PUT requests it is going to work. The router also routes `PATCH` requests to `/users/:id` to the `update` action.

```
resources :users do
  patch :update_name, on: :member
end
```

If the action is being used in a public API and you can't change to HTTP method being used, you can update your form to use the `PUT` method instead:

```
<%= form_for [ :update_name, @user ], method: :put do |f| %>
```

For more on PATCH and why this change was made, see this post on the Rails blog.

4.1.1 A note about media types

The errata for the `PATCH` verb specifies that a 'diff' media type should be used with `PATCH`. One such format is JSON Patch. While Rails does not support JSON Patch natively, it's easy enough to add support:

```
# in your controller
def update
  respond_to do |format|
    format.json do
      # perform a partial update
      @article.update params[:article]
    end

    format.json_patch do
      # perform sophisticated change
    end
  end
end

# In config/initializers/json_patch.rb:
Mime::Type.register 'application/json-patch+json', :json_patch
```

As JSON Patch was only recently made into an RFC, there aren't a lot of great Ruby libraries yet. Aaron Patterson's hana is one such gem, but doesn't have full support for the last few changes in the specification.

4.2 Gemfile

Rails 4.0 removed the `assets` group from Gemfile. You'd need to remove that line from your Gemfile when upgrading. You should also update your application file (in `config/application.rb`):

```
# Require the gems listed in Gemfile, including any gems
# you've limited to :test, :development, or :production.
Bundler.require(:default, Rails.env)
```

4.3 vendor/plugins

Rails 4.0 no longer supports loading plugins from `vendor/plugins`. You must replace any plugins by extracting them to gems and adding them to your Gemfile. If you choose not to make them gems, you can move them into, say, `lib/my_plugin/*` and add an appropriate initializer in `config/initializers/my_plugin.rb`.

4.4 Active Record

- Rails 4.0 has removed the identity map from Active Record, due to some inconsistencies with associations. If you have manually enabled it in your application, you will have to remove the following config that has no effect anymore: `config.active_record.identity_map`.
- The `delete` method in collection associations can now receive `Fixnum` or `String` arguments as record ids, besides records, pretty much like the `destroy` method does. Previously it raised `ActiveRecord::AssociationTypeMismatch` for such arguments. From Rails 4.0 on `delete` automatically tries to find the records matching the given ids before deleting them.
- In Rails 4.0 when a column or a table is renamed the related indexes are also renamed. If you have migrations which rename the indexes, they are no longer needed.
- Rails 4.0 has changed `serialized_attributes` and `attr_readonly` to class methods only. You shouldn't use instance methods since it's now deprecated. You should change them to use class methods, e.g. `self.serialized_attributes` to `self.class.serialized_attributes`.
- When using the default coder, assigning `nil` to a serialized attribute will save it to the database as `NULL` instead of passing the `nil` value through YAML (`"--- \n...\n"`).
- Rails 4.0 has removed `attr_accessible` and `attr_protected` feature in favor of Strong Parameters. You can use the Protected Attributes gem for a smooth upgrade path.

- If you are not using Protected Attributes, you can remove any options related to this gem such as `whitelist_attributes` or `mass_assignment_sanitizer` options.
- Rails 4.0 requires that scopes use a callable object such as a Proc or lambda:

```
scope :active, where(active: true)

# becomes
scope :active, -> { where active: true }
```

- Rails 4.0 has deprecated `ActiveRecord::Fixtures` in favor of `ActiveRecord::FixtureSet`.
- Rails 4.0 has deprecated `ActiveRecord::TestCase` in favor of `ActiveSupport::TestCase`.
- Rails 4.0 has deprecated the old-style hash based finder API. This means that methods which previously accepted "finder options" no longer do. For example, `Book.find(:all, conditions: { name: '1984' })` has been deprecated in favor of `Book.where(name: '1984')`
- All dynamic methods except for `find_by_...` and `find_by_...!` are deprecated. Here's how you can handle the changes:
  - `find_all_by_...` becomes `where(...)`.
  - `find_last_by_...` becomes `where(...).last`.
  - `scoped_by_...` becomes `where(...)`.
  - `find_or_initialize_by_...` becomes `find_or_initialize_by(...)`.
  - `find_or_create_by_...` becomes `find_or_create_by(...)`.
- Note that `where(...)` returns a relation, not an array like the old finders. If you require an `Array`, use `where(...).to_a`.
- These equivalent methods may not execute the same SQL as the previous implementation.
- To re-enable the old finders, you can use the activerecord-deprecated_finders gem.

4.5 Active Resource

Rails 4.0 extracted Active Resource to its own gem. If you still need the feature you can add the Active Resource gem in your Gemfile.

4.6 Active Model

- Rails 4.0 has changed how errors attach with the `ActiveModel::Validations::ConfirmationValidator`. Now when confirmation validations fail, the error will be attached to `:#{attribute}_confirmation` instead of `attribute`.
- Rails 4.0 has changed `ActiveModel::Serializers::JSON.include_root_in_json` default value to `false`. Now, Active Model Serializers and Active Record objects have the same default behavior. This means that you can comment or remove the following option in the `config/initializers/wrap_parameters.rb` file:

```
# Disable root element in JSON by default.
# ActiveSupport.on_load(:active_record) do
#   self.include_root_in_json = false
# end
```

4.7 Action Pack

- Rails 4.0 introduces `ActiveSupport::KeyGenerator` and uses this as a base from which to generate and verify signed cookies (among other things). Existing signed cookies generated with Rails 3.x will be transparently upgraded if you leave your existing `secret_token` in place and add the new `secret_key_base`.

```
# config/initializers/secret_token.rb
Myapp::Application.config.secret_token = 'existing secret token'
Myapp::Application.config.secret_key_base = 'new secret key base'
```

Please note that you should wait to set `secret_key_base` until you have 100% of your userbase on Rails 4.x and are reasonably sure you will not need to rollback to Rails 3.x. This is because cookies signed based on the new `secret_key_base` in Rails 4.x are not backwards compatible with Rails 3.x. You are free to leave your existing `secret_token` in place, not set the new `secret_key_base`, and ignore the deprecation warnings until you are reasonably sure that your upgrade is otherwise complete.

If you are relying on the ability for external applications or Javascript to be able to read your Rails app's signed session cookies (or signed cookies in general) you should not set `secret_key_base` until you have decoupled these concerns.

- Rails 4.0 encrypts the contents of cookie-based sessions if `secret_key_base` has been set. Rails 3.x signed, but did not encrypt, the contents of cookie-based session. Signed cookies are "secure" in that they are verified to have been generated by your app and are tamper-proof. However, the contents can be viewed by end users, and encrypting the contents eliminates this caveat/concern without a significant performance penalty.

Please read Pull Request #9978 for details on the move to encrypted session cookies.

- Rails 4.0 removed the `ActionController::Base.asset_path` option. Use the assets pipeline feature.
- Rails 4.0 has deprecated `ActionController::Base.page_cache_extension` option. Use `ActionController::Base.default_static_extension` instead.
- Rails 4.0 has removed Action and Page caching from Action Pack. You will need to add the `actionpack-action_caching` gem in order to use `caches_action` and the `actionpack-page_caching` to use `caches_pages` in your controllers.
- Rails 4.0 has removed the XML parameters parser. You will need to add the `actionpack-xml_parser` gem if you require this feature.
- Rails 4.0 changes the default memcached client from `memcache-client` to `dalli`. To upgrade, simply add `gem 'dalli'` to your `Gemfile`.

- Rails 4.0 deprecates the `dom_id` and `dom_class` methods in controllers (they are fine in views). You will need to include the `ActionView::RecordIdentifier` module in controllers requiring this feature.
- Rails 4.0 deprecates the `:confirm` option for the `link_to` helper. You should instead rely on a data attribute (e.g. `data: { confirm: 'Are you sure?' }`). This deprecation also concerns the helpers based on this one (such as `link_to_if` or `link_to_unless`).
- Rails 4.0 changed how `assert_generates`, `assert_recognizes`, and `assert_routing` work. Now all these assertions raise `Assertion` instead of `ActionController::RoutingError`.
- Rails 4.0 raises an `ArgumentError` if clashing named routes are defined. This can be triggered by explicitly defined named routes or by the `resources` method. Here are two examples that clash with routes named `example_path`:

```
get 'one' => 'test#example', as: :example
get 'two' => 'test#example', as: :example

resources :examples
get 'clashing/:id' => 'test#example', as: :example
```

In the first case, you can simply avoid using the same name for multiple routes. In the second, you can use the `only` or `except` options provided by the `resources` method to restrict the routes created as detailed in the Routing Guide.

- Rails 4.0 also changed the way unicode character routes are drawn. Now you can draw unicode character routes directly. If you already draw such routes, you must change them, for example:

```
get Rack::Utils.escape('こんにちは'), controller: 'welcome', action: 'index'
```

becomes

```
get 'こんにちは', controller: 'welcome', action: 'index'
```

- Rails 4.0 requires that routes using `match` must specify the request method. For example:

```
# Rails 3.x
match '/' => 'root#index'

# becomes
match '/' => 'root#index', via: :get

# or
get '/' => 'root#index'
```

- Rails 4.0 has removed `ActionDispatch::BestStandardsSupport` middleware, `<!DOCTYPE html>` already triggers standards mode per http://msdn.microsoft.com/en-us/library/jj676915(v=vs.85).aspx and ChromeFrame header has been moved to `config.action_dispatch.default_headers`.

Remember you must also remove any references to the middleware from your application code, for example:

```
# Raise exception
config.middleware.insert_before(Rack::Lock, ActionDispatch::BestStandardsSupport)
```

Also check your environment settings for `config.action_dispatch.best_standards_support` and remove it if present.

- In Rails 4.0, precompiling assets no longer automatically copies non-JS/CSS assets from `vendor/assets` and `lib/assets`. Rails application and engine developers should put these assets in `app/assets` or configure `config.assets.precompile`.
- In Rails 4.0, `ActionController::UnknownFormat` is raised when the action doesn't handle the request format. By default, the exception is handled by responding with 406 Not Acceptable, but you can override that now. In Rails 3, 406 Not Acceptable was always returned. No overrides.
- In Rails 4.0, a generic `ActionDispatch::ParamsParser::ParseError` exception is raised when `ParamsParser` fails to parse request params. You will want to rescue this exception instead of the low-level `MultiJson::DecodeError`, for example.
- In Rails 4.0, `SCRIPT_NAME` is properly nested when engines are mounted on an app that's served from a URL prefix. You no longer have to set `default_url_options[:script_name]` to work around overwritten URL prefixes.
- Rails 4.0 deprecated `ActionController::Integration` in favor of `ActionDispatch::Integration`.
- Rails 4.0 deprecated `ActionController::IntegrationTest` in favor of `ActionDispatch::IntegrationTest`.
- Rails 4.0 deprecated `ActionController::PerformanceTest` in favor of `ActionDispatch::PerformanceTest`.
- Rails 4.0 deprecated `ActionController::AbstractRequest` in favor of `ActionDispatch::Request`.
- Rails 4.0 deprecated `ActionController::Request` in favor of `ActionDispatch::Request`.
- Rails 4.0 deprecated `ActionController::AbstractResponse` in favor of `ActionDispatch::Response`.
- Rails 4.0 deprecated `ActionController::Response` in favor of `ActionDispatch::Response`.
- Rails 4.0 deprecated `ActionController::Routing` in favor of `ActionDispatch::Routing`.

4.8 Active Support

Rails 4.0 removes the `j` alias for `ERB::Util#json_escape` since `j` is already used for `ActionView::Helpers::JavaScriptHelper#escape_javascript`.

4.9 Helpers Loading Order

The order in which helpers from more than one directory are loaded has changed in Rails 4.0. Previously, they were gathered and then sorted alphabetically. After upgrading to Rails 4.0, helpers will preserve the order of loaded directories and will be sorted alphabetically only within each directory. Unless you explicitly use the `helpers_path` parameter, this change will only impact the way of loading helpers from engines. If you rely on the ordering, you should check if correct methods are available after upgrade. If you would like to change the order in which engines are loaded, you can use `config.railties_order=` method.

4.10 Active Record Observer and Action Controller Sweeper

`ActiveRecord::Observer` and `ActionController::Caching::Sweeper` have been extracted to the `rails-observers` gem. You will need to add the `rails-observers` gem if you require these features.

4.11 sprockets-rails

- `assets:precompile:primary` and `assets:precompile:all` have been removed. Use `assets:precompile` instead.
- The `config.assets.compress` option should be changed to `config.assets.js_compressor` like so for instance:

```
config.assets.js_compressor = :uglifier
```

4.12 sass-rails

- `asset-url` with two arguments is deprecated. For example: `asset-url("rails.png", image)` becomes `asset-url("rails.png")`.

5 Upgrading from Rails 3.1 to Rails 3.2

If your application is currently on any version of Rails older than 3.1.x, you should upgrade to Rails 3.1 before attempting an update to Rails 3.2.

The following changes are meant for upgrading your application to the latest 3.2.x version of Rails.

5.1 Gemfile

Make the following changes to your `Gemfile`.

```
gem 'rails', '3.2.18'

group :assets do
  gem 'sass-rails',   '~> 3.2.6'
  gem 'coffee-rails', '~> 3.2.2'
  gem 'uglifier',     '>= 1.0.3'
end
```

5.2 config/environments/development.rb

There are a couple of new configuration settings that you should add to your development environment:

```
# Raise exception on mass assignment protection for Active Record models
config.active_record.mass_assignment_sanitizer = :strict

# Log the query plan for queries taking more than this (works
# with SQLite, MySQL, and PostgreSQL)
config.active_record.auto_explain_threshold_in_seconds = 0.5
```

5.3 config/environments/test.rb

The `mass_assignment_sanitizer` configuration setting should also be be added to `config/environments/test.rb`:

```
# Raise exception on mass assignment protection for Active Record models
config.active_record.mass_assignment_sanitizer = :strict
```

5.4 vendor/plugins

Rails 3.2 deprecates `vendor/plugins` and Rails 4.0 will remove them completely. While it's not strictly necessary as part of a Rails 3.2 upgrade, you can start replacing any plugins by extracting them to gems and adding them to your Gemfile. If you choose not to make them gems, you can move them into, say, `lib/my_plugin/*` and add an appropriate initializer in `config/initializers/my_plugin.rb`.

5.5 Active Record

Option `:dependent => :restrict` has been removed from `belongs_to`. If you want to prevent deleting the object if there are any associated objects, you can set `:dependent => :destroy` and return `false` after checking for existence of association from any of the associated object's destroy callbacks.

6 Upgrading from Rails 3.0 to Rails 3.1

If your application is currently on any version of Rails older than 3.0.x, you should upgrade to Rails 3.0 before attempting an update to Rails 3.1.

The following changes are meant for upgrading your application to Rails 3.1.12, the last 3.1.x version of Rails.

6.1 Gemfile

Make the following changes to your `Gemfile`.

```
gem 'rails', '3.1.12'
gem 'mysql2'

# Needed for the new asset pipeline
group :assets do
  gem 'sass-rails',   '~> 3.1.7'
  gem 'coffee-rails', '~> 3.1.1'
  gem 'uglifier',     '>= 1.0.3'
end

# jQuery is the default JavaScript library in Rails 3.1
gem 'jquery-rails'
```

6.2 config/application.rb

The asset pipeline requires the following additions:

```
config.assets.enabled = true
config.assets.version = '1.0'
```

If your application is using an "/assets" route for a resource you may want change the prefix used for assets to avoid conflicts:

```
# Defaults to '/assets'
config.assets.prefix = '/asset-files'
```

6.3 config/environments/development.rb

Remove the RJS setting `config.action_view.debug_rjs = true`.

Add these settings if you enable the asset pipeline:

```
# Do not compress assets
config.assets.compress = false

# Expands the lines which load the assets
config.assets.debug = true
```

6.4 config/environments/production.rb

Again, most of the changes below are for the asset pipeline. You can read more about these in the Asset Pipeline guide.

```
# Compress JavaScripts and CSS
config.assets.compress = true

# Don't fallback to assets pipeline if a precompiled asset is missed
config.assets.compile = false

# Generate digests for assets URLs
config.assets.digest = true

# Defaults to Rails.root.join("public/assets")
# config.assets.manifest = YOUR_PATH

# Precompile additional assets (application.js, application.css, and all non-JS/CSS are
already added)
# config.assets.precompile += %w( search.js )

# Force all access to the app over SSL, use Strict-Transport-Security, and use secure
cookies.
# config.force_ssl = true
```

6.5 config/environments/test.rb

You can help test performance with these additions to your test environment:

```
# Configure static asset server for tests with Cache-Control for performance
config.serve_static_assets = true
config.static_cache_control = 'public, max-age=3600'
```

6.6 config/initializers/wrap_parameters.rb

Add this file with the following contents, if you wish to wrap parameters into a nested hash. This is on by default in new applications.

```
# Be sure to restart your server when you modify this file.
# This file contains settings for ActionController::ParamsWrapper which
# is enabled by default.

# Enable parameter wrapping for JSON. You can disable this by setting :format to an empty
array.
ActiveSupport.on_load(:action_controller) do
  wrap_parameters format: [:json]
end

# Disable root element in JSON by default.
ActiveSupport.on_load(:active_record) do
  self.include_root_in_json = false
end
```

6.7 config/initializers/session_store.rb

You need to change your session key to something new, or remove all sessions:

```
# in config/initializers/session_store.rb
AppName::Application.config.session_store :cookie_store, key: 'SOMETHINGNEW'
```

or

```
$ bin/rake db:sessions:clear
```

6.8 Remove :cache and :concat options in asset helpers references in views

- With the Asset Pipeline the :cache and :concat options aren't used anymore, delete these options from your views.

7 Feedback

You're encouraged to help improve the quality of this guide.

Please contribute if you see any typos or factual errors. To get started, you can read our documentation contributions section.

You may also find incomplete content, or stuff that is not up to date. Please do add any missing documentation for master. Make sure to check Edge Guides first to verify if the issues are already fixed or not on the master branch. Check the Ruby on Rails Guides Guidelines for style and conventions.

If for whatever reason you spot something to fix but cannot patch it yourself, please open an issue.

And last but not least, any kind of discussion regarding Ruby on Rails documentation is very welcome in the rubyonrails-docs mailing list.