Active Job – Make work happen later

Active Job is a framework for declaring jobs and making them run on a variety of queuing backends. These jobs can be everything from regularly scheduled clean-ups, to billing charges, to mailings — anything that can be chopped up into small units of work and run in parallel.

It also serves as the backend for Action Mailer's #deliver_later functionality that makes it easy to turn any mailing into a job for running later. That's one of the most common jobs in a modern web application: sending emails outside the request-response cycle, so the user doesn't have to wait on it.

The main point is to ensure that all Rails apps will have a job infrastructure in place, even if it's in the form of an "immediate runner". We can then have framework features and other gems build on top of that, without having to worry about API differences between Delayed Job and Resque. Picking your queuing backend becomes more of an operational concern, then. And you'll be able to switch between them without having to rewrite your jobs.

You can read more about Active Job in the Active Job Basics guide.

Usage

To learn how to use your preferred queuing backend see its adapter documentation at ActiveJob::QueueAdapters.

Declare a job like so:

```ruby
class MyJob < ActiveJob::Base
  queue_as :my_jobs

  def perform(record)
    record.do_work
  end
end
```

Enqueue a job like so:

```ruby
MyJob.perform_later record  # Enqueue a job to be performed as soon as the queuing system is free.
```

```ruby
MyJob.set(wait_until: Date.tomorrow.noon).perform_later(record)  # Enqueue a job to be performed tomorrow at noon.
```

```ruby
MyJob.set(wait: 1.week).perform_later(record) # Enqueue a job to be performed 1 week from now.
```

That's it!

GlobalID support

Active Job supports GlobalID serialization for parameters. This makes it possible to pass live Active Record objects to your job instead of class/id pairs, which you then have to manually deserialize. Before, jobs would look like this:

```ruby
class TrashableCleanupJob
  def perform(trashable_class, trashable_id, depth)
    trashable = trashable_class.constantize.find(trashable_id)
    trashable.cleanup(depth)
  end
end
```

Now you can simply do:

```ruby
class TrashableCleanupJob
  def perform(trashable, depth)
    trashable.cleanup(depth)
  end
end
```

This works with any class that mixes in GlobalID::Identification, which by default has been mixed into Active Record classes.

Supported queuing systems

Active Job has built-in adapters for multiple queuing backends (Sidekiq, Resque, Delayed Job and others). To get an up-to-date list of the adapters see the API Documentation for ActiveJob::QueueAdapters.

Please note: We are not accepting pull requests for new adapters. We encourage library authors to provide an ActiveJob adapter as part of their gem, or as a stand-alone gem. For discussion about this see the following PRs: 23311, 21406, and #32285.

Download and installation

The latest version of Active Job can be installed with RubyGems:

```
$ gem install activejob
```

Source code can be downloaded as part of the Rails project on GitHub:

- https://github.com/rails/rails/tree/main/activejob

License

Active Job is released under the MIT license:

- https://opensource.org/licenses/MIT

Support

API documentation is at:

- https://api.rubyonrails.org

Bug reports for the Ruby on Rails project can be filed here:

- https://github.com/rails/rails/issues

Feature requests should be discussed on the rails-core mailing list here:

- https://discuss.rubyonrails.org/c/rubyonrails-core