Do It Later with Delayed Job

Simple job processing queue

Pete Campbell

pete@sumirolabs.com
@sumirolabs
github.com/campbell

What Is Delayed Job?

- Database-backed job-processing queue
- Easy to use, observe, manage
- Automatic retries before failure
- Many options for running jobs

Where? Why?

- Created by Shopify for asynchronous background tasks

  - Sending newsletters
  - Image resizing
  - HTTP downloads
  - Updating SOLR
  - Batch imports
  - Spam checks

Why Me?

- Client was new to Rails but had plenty of Linux & MySQL expertise
- Needed simple solution that was easy to learn and support
- Reliable (meaning don’t lose my jobs!)

How Does It Work?

Jobs are stored in the database

- YAML-marshalled Ruby objects
- Pretty much any method can be called at a later time
- Easy to inspect, manipulate
- ActiveRecord, DataMapper, Mongoid, MongoMapper

Persistent workers

- Workers can run on different machines
- Rails is only loaded on startup & not for every job

How Does It Work?

```
create_table :delayed_jobs, :force => true do |table|
  table.integer  :priority, :default => 0
  table.integer  :attempts, :default => 0
  table.text     :handler    # YAML-encoded object
  table.text     :last_error # Reason for last error
  table.datetime :run_at     # When to run (now or future)
  table.datetime :locked_at  # Set when being processed
  table.datetime :failed_at  # Set when all retries failed
  table.string   :locked_by  # Who is working on this
  table.timestamps
end
```

Failure Is An Option

Automatic retries of jobs with errors

- Default of 25 retries, then failure
- Retry every 5+(number of retries)^4 seconds

Jobs automatically deleted after failure (but you can disable this)

Assumed max runtime is 4 hours, after which another queue could start the job

How Do I Use It?

Rails 3.0+

1) Add the gem to your app:

```
gem 'delayed_job'
bundle install
```

2) Set up the database:

```
script/rails generate delayed_job
rake db:migrate
```

How Do I Use It?

Rails 3.0+

3) Schedule jobs:

```
@my_model.delay.do_something(with_this)
```

4) Execute jobs:

```
$ script/delayed_job start
```

Advantages

- Jobs stored in database
  - Less complexity
  - Easy to inspect (failed, pending jobs)
  - Easy to manipulate (delete from jobs where…)
  - Easy to have distributed workers
  - Seems safer
- Performance
  - Persistent workers, less startup cost

Disadvantages

- Monolithic queues
  - Can't send some jobs to specific workers (i.e. sending emails only from an email server)
- No queue monitoring, restarting workers
- Performance
  - Database access slow for large queues

This prompted GitHub to create Resque

- GitHub recommends DJ if background tasks are < 50% of the workload

https://github.com/blog/542-introducing-resque

Options

Specify that some methods should always go through Delayed Job:

```
class Job
  def do_something
    # lotsa stuff
  end
  handle_asynchronously :do_something
end

job = Job.new
job.do_something # => goes into queue
```

Options

Specify when the job should be run & the priority:

```
handle_asynchronously :do_something, :run_at
  => Proc.new{ 5.minutes.from_now }

attr_reader :how_important
handle_asynchronously :do_something,
  :priority => Proc.new{ how_important }
```

Options

Many callback hooks are available, defined as methods in the model:

```
class JobWithHooks < Job
  def enqueue(job); end;
  def perform; end;
  def before(job); end;
  def after(job); end;
  def success(job); end;
  def error(job); end;
  def failure(job); end
end
```

Options

Create custom jobs*:

```
class NewsletterJob < Struct.new(:text, :emails)
  def perform
    emails.each{|e| NewsMailer.send_email(text, e)}
  end
end

Delayed::Job.enqueue NewsletterJob.new('message',
  Customers.find(:all).collect(&:email))
```

* I'd prefer this:

```
n = NewsletterJob.new('message', Customers.find
  (:all).collect(&:email))
n.delay.send_emails  # Assume this method is defined
```

Gotcha!

Restart the queues if you change the model:

- Since workers are persistent, your changes won't be propagated automatically
- New jobs may fail in really strange ways

Remember This Stuff

Delayed Job is:

- Simple to implement, use, observe
- Flexible, many options for setting job priorities, run time, callback hooks
- Good performance due to persistent workers
- Not as good for queues that get backed up
- Can’t assign jobs to specific queues

Do It Later With Delayed Job

Pete Campbell

pete@sumirolabs.com
@sumirolabs
github.com/campbell