November 19th, 2008

Posted by Daniel

Delayed Gratification with Rails

I realized when I started taking suggestions that I would not be able to do them all justice, so I asked a few of my friends to be guest authors. Daniel Morrison, of Collective Idea, is the first and will be showing a few ways he has used delayed job to offload tasks to the background. Without any further ado, here is Dan.

At Collective Idea, we started using delayed_job a few months ago, and have fallen in love with its simplicity. In fact, my first implementation of it was done and tested on a quick train ride to Chicago, with time to spare.

So it's easy?

Yep, you can add delayed_job in 10 minutes or less.

Getting Started

Install the plugin, which is available on GitHub.

Then build & run a migration to add the delayed_jobs table:

```
create_table :delayed_jobs, :force => true do |table|
  table.integer  :priority, :default => 0
  table.integer  :attempts, :default => 0
  table.text     :handler
  table.string   :last_error
  table.datetime :run_at
  table.datetime :locked_at
  table.datetime :failed_at
  table.string   :locked_by
  table.timestamps
end
```

(In the future there will be a generator for this step. Tobi, please merge some of the forks!)

Run your jobs

The rest of the article will focus on creating jobs, but when you want to run them, you can simply run `rake jobs:work`

The job runner will grab a few jobs and run them one at a time. It locks them so that multiple runners won't conflict, and it will retry jobs a number of times if it fails for some reason. If it does fail, it stores the most recent error message. Play around in script/console with the `Delayed::Job` model to see how it works.

There are some other ways to run jobs in production in some of the forks on github. Collective Idea's for example, adds `script/delayed_job`. The rake task will work for now though.

Example 1: Delay Something

Now the fun part: pick something to delay. A great place for delay is email. I've seen places where apps have broken due to email not being able to send. Maybe the client changed their email server and didn't tell the programmer, or maybe the mail server was temporarily down for maintenance. Either way, it generally shouldn't stop our app.

Here's a common controller pattern for a contact form:

```
def create
  @contact_form = ContactForm.new(params[:contact_form])

  if @contact_form.save
    flash[:notice] = 'Your feedback has been sent. Thanks for contacting us!'
    ContactMailer.deliver_contact_request(@contact_form)
    redirect_to @contact_form
  else
    render :action => "new"
  end
end
```

The problem is that if the mailer fails due to outside circumstances, we're throwing an error. We could rescue from that, but since there's nothing the user can do, we shouldn't involve them.

Instead, let's send email as a delayed_job, which will retry on failure and also keep track of the last error it sees.

Here's the refactored action:

```
def create
  @contact_form = ContactForm.new(params[:contact_form])

  if @contact_form.save
    flash[:notice] = 'Your feedback has been sent. Thanks for contacting us!'
    ContactMailer.send_later(:deliver_contact_request, @contact_form)
    redirect_to @contact_form
  else
    render :action => "new"
  end
end
```

That's it! `send_later` works just like `send`, but magically turns it into a delayed_job. Now our user can keep clicking through the app, and the email will send in a few seconds.

Example 2: A more complex example.

My first use of delayed_job was a large import process that could take 5 minutes or more. What's going on here is the user uploads a large CSV file that we then processed the crap out of, adding hundreds or thousands of rows to different tables.

In my controller, I had something along these lines:

```
def update
  @item = Item.find(params[:id])
  if @item.update_attributes(params[:item])
    @item.import_file.import!
    redirect_to @item
  else
    render :action => 'edit'
  end
end
```

The problem here is the browser has to sit and wait until the `import!` method finishes. Not good. There's no need for the user to wait for the import. We can give them a message in the interface that the import is still in-progress.

So change the controller method above to this:

```
def update
  @item = Item.find(params[:id])
  if @item.update_attributes(params[:item])
    Delayed::Job.enqueue ImportJob.new(@item.import_file)
    redirect_to @item
  else
    render :action => 'edit'
  end
end
```

`ImportJob` is a simple class I've defined and tossed in the lib/ directory.

```
class ImportJob
  attr_accessor :import_file_id

  def initialize(import_file)
    self.import_file_id = import_file.id
  end

  def perform
    import_file = ImportFile.find(import_file_id)
    import_file.import!
    import_file.complete!
  end
end
```

Again, that's it! Our `ImportJob`, holding our ImportFile (which is an ActiveRecord object, using

attachment_fu) is added to the queue. When we pop it off the queue later, the `perform` method is called, which does our import. My `complete!` method sets a `completed_at` flag so I can tell the user that we're done.

My real code is a bit more complex, but hopefully you can see how this style can by used for doing multi-step jobs.

Example 3: Tiny but useful.

The import job above adds a lot (hundreds) of locations to this app that will show up on a map eventually. I'm using acts_as_geocodable (because it's awesome) to geocode the addresses via Google & Yahoo, but I don't need that info right away, and I don't need it holding up the imports.

acts_as_geocodable does its work by adding an `after_filter :attach_geocode` automatically to the model you specify.

So for my app, I changed it from an `after_filter` to a delayed_job.

Here's all the code it took:

```
class Location < ActiveRecord::Base
  acts_as_geocodable

  # some code removed for clarity

  def attach_geocode_with_delay
    self.send_later(:attach_geocode_without_delay)
  end
  alias_method_chain :attach_geocode, :delay

end
```

Very fun.

Your turn

There's really not much more to delayed_job than that. Its simplicity is what makes it great. So go and delay something already!

Tags: background jobs, collectiveidea, delayed_job, performance, plugins

Related Articles

- Oct 17, 2008 Who Done What? a.k.a. User Stamping
- Oct 13, 2008 How To Generate PDFs in Rails With Prawn
- Sep 17, 2008 Rails App Performance Monitoring
- Feb 28, 2008 KYTCR Part IV: Resource Reporting
- Feb 27, 2007 Open Up Your ID With Rails