pure virtual blog

Some thoughts and ideas of Christoph Petschnig

search

- HOME
- About

- rss feed

2010
04.16

An URL shortener with Ruby on Rails 3 and Redis

Category: Redis, Ruby on Rails / Tag: ActiveModel, Metal, Rails, Redis, Ruby / Add Comment

Note: This tutorial is about introducing Redis and showing some features of Rails 3. In a real project, I suggest using a library like Ohm ॐ.

Redis is an acronym for **RE**mote **DI**ctionary **S**erver and one of the currently so popular NoSQL databases. It is a key-value storage, very much like Memcached, but the data is persistent. All data is kept in memory, which makes it very fast!

Redis Installation

The projects home is http://code.google.com/p/redis/. Download the source:

$> wget http://redis.googlecode.com/files/redis-1.2.6.tar.gz

Extract and enter directory, build it. This should not take longer than 10 seconds, only 10 files will get compiled.

$> tar -zxvf redis-1.2.6.tar.gz
$> cd redis-1.2.6
$> make

Start the server and keep it running. The database file will be `**dump.rdb**` in the current directory, unless you specify something else in `**redis.conf**`.

$> ./redis-server

That's all. For more information about Redis, check out the nice presentation Mathias Meyer from Berlin, Germany once held. Get the slides at http://www.slideshare.net/mattmatt/redis-nsql-berlin-2352325

The Rails application

We don't want to use ActiveRecord in this application, but instead use our own Redis adaptor. Generate the application skeleton:

$> rails mini_urler --skip-activerecord
$> cd mini_urler

We can now take a look at `**config/application.rb**` to verify that ActiveRecord will not be loaded – it is commented out.

```
# Pick the frameworks you want:
# require "active_record/railtie"
require "action_controller/railtie"
require "action_mailer/railtie"
require "active_resource/railtie"
require "rails/test_unit/railtie"
```

Be sure that the redis gem is installed on your system. If in doubt, run

$> sudo gem install redis

Open the `**Gemfile**` and add the line

gem 'redis'

Now we use bundler to satisfy all dependencies for your application.

$> bundle install

Generate some files for our application. The "–orm=…" part is necessary in Rails 3.0.0.beta3 and might be omitted in the final release of Rails 3.0.

$> rails generate scaffold MiniUrl key:string url:string --orm=activerecord

You can now open `**app/views/mini_urls/_form.html.erb**' and remove the key part. Our application will generate the key by itself.

Now create the file `**app/models/mini_url.rb**' and insert the following code. We'll make use of Rubys ducktyping capability.

```
class MiniUrl

  #  take some goodies from ActiveModel
  #  this will help us to seamlessly integrate this class into Rails
  include ActiveModel::Validations

  #  let the user edit the url
  attr_accessor :url

  #  make use of ActiveModel validation
  #  URLs must start with http:// or https://
  validates_format_of :url, :with => /^https?:\/\//

  #  the constructor
  def initialize(params = {})
    @url = params[:url]
  end

  #  at this point, we just supply a stupid version of MiniUrl.all
  def self.all(*args)
    []
  end

  #  Rails form helpers want to know the primary key, if the object
  #  is already stored in the database
  def to_key
    nil
  end

  #  Rails needs the method to determine between an update and create,
  #  i.e. in < %= f.submit %>. For now, we answer with "No, this object is
  #  not yet stored in the database"
  def persisted?
    false
  end

  #  For the beginning, we just validate, but store nothing
  def save
    valid?
  end

end
```

At this point, you can run the server and try to create new urls. There should be no errors, but nothing will be saved. Open the application in your browser at http://localhost:3000/mini_urls.

$> rails server

Now let's add key, that we totally left out so far. We expand our class with the following code.

```
  #  the user should no be able to change the key
  attr_reader :key

  #  the key attribute should not be null
  validates_presence_of :key

  #  add the key attribute to the constructor
  def initialize(params = {})
    @key = params[:key]
    ...
  end

  #  auto-generate a key; for demonstration purposes
  #  we'll be content with a random four letter uppercase string;
  #  a real-word application should make sure,
  #  that the key is not in use yet
  def generate_key
    @key = (rand(26) + 65).chr + (rand(26) + 65).chr +
      (rand(26) + 65).chr + (rand(26) + 65).chr
  end

  #  add key generation to the save method
  def save
    generate_key unless @key
    valid?
  end
```

Again, at this point, the application should not produce errors when creating new entries, but it would not store anything. Therefore, let the fun begin. Let's add Redis. Create the file `**app/models/redis_mod.rb**` and insert the following:

```
#  wrapper around a Redis database connection, that we
#  can use as a singleton object application wide
module RedisMod
  #  a real world application would hand over
  #  a lot of parameters to Redis.new
  @@redis_connection = Redis.new
  def self.conn
    @@redis_connection
  end
end
```

To finish the MiniUrl class, we need to implement database storage and retrieval.

```
  #  the final version of the save method
  def save
    generate_key unless @key
    #  leave method if validation fails
    return false unless valid?
    #  store the key/value pair
    RedisMod.conn[@key] = @url
    # also store key to a list that we will retrieve on MiniUrl.all
    RedisMod.conn.lpush('USER00', @key)
    true
  end

  #  the final version of the MiniUrl.all method
  def self.all(*args)
    #  get an array of all keys
    RedisMod.conn.lrange('USER00', 0, -1).collect do |key|
      #  construct object by only passing the key
      MiniUrl.new(:key => key)
    end
  end

  #  the final version of the constructor
  #  if only the key is given, fetch the value from the database
  def initialize(params = {})
    @key = params[:key]
    @url = params[:url]
    @url = RedisMod.conn[@key] if @url.nil? && @key
  end
```

Let's go to the redirecting part. Now we can take full advantage of the ultra-lightness and speed of Redis. To avoid the whole Rails application overhead when we simply want to make an external redirect, we use Rails Metal:

```
$> rails generate metal Redirector
```

This command has generated the file `**app/metal/redirector.rb**` for us. Open it and change it slightly:

```
class Redirector
  def self.call(env)
    if env["PATH_INFO"] =~ /^\/([A-Z]{4})$/
      url = RedisMod.conn[$1]
      return [301, {"Location" => url}, ['']] if url
    end
    [404, {"Content-Type" => "text/html",
      "X-Cascade" => "pass"}, ["Not Found"]]
  end
end
```

Don't forget to restart the server after you made changes to a piece of metal!

You can find the whole application at GitHub.

Summary

- We build a Rails 3 application without using ActiveRecord
- Still, we made use of the new ActiveModel, so we could use Rails validation methods
- We did not even touch the generated controller and made only very small adjustments to the view
- We made the application serving redirects very fast using Metal
- Not a single SQL statement was written/generated
- We had fun playing around with Rails 3 and Redis (at least I did)

1 comment so far

Add Your Comment

1. 

   Reply
   Aaron Kras said: 2011.06.01 13:31

   Whole Handle Marketing Review- Wonderful piece of particulars which you?ve acquired on this website write-up. Hope I could quite possibly get some a lot a lot more from the stuff on your personal Internet site. I'll arrive once more.

Your Comment

Name
Email
URI :
Submit

- ## Recent Posts
  - URL Rewriting in Rails with Metal
  - An URL shortener with Ruby on Rails 3 and Redis
- ## Recent Comments
  - Aaron Kras: Whole Handle Ma...
- ## Meta
  - Log in