

Ruby on Rails for a job site

part 1 : Peter Vandenabeele

fosdem 23 Feb 2008

- whoami
- patents
- business/architecture choice for Rails
- full text search (Sphinx)
- AJAX + pagination, “reload”, “back”

whoami

- Peter Vandenabeele (vandenabeele.com)
- studied/worked mainly in Leuven
- Mind (embedded Linux 2000-2006)
- sold to Essensium (SoC design)
- Fedict (ODF Belgium 2006-2007)
- summer 2007: find a different job/project in Leuven area => where would I fit ??
- full-time (> 100%) on the site ...

Patents ?

- checked patents before anything else
- Monster.com year report:
 - “... we have obtained one patent and applied for several others ... software ... methods”
 - “... third parties can bring patent claims against us ...”
- business methods not patentable in EU :-)
- if patents in this field would have been present in EU, I would not have started.

The problem

- choosing a job in Leuven
- criterium of completeness:
am I aware of *all* opportunities?
- where do I want to work?
company culture: soft values such as
honesty, responsibility, perfectionism, ...
- how do I know?
- employers do not easily find people ...

The solution (?)

- *all* jobs: allow gratis posting and actively collect job posting for Leuven companies to create critical mass
- anonymous for job seekers (no CV's), will actually attract high-end people
- social networking: people give recommendations for employers. So, the employers that are “open” to the concept will receive tremendous positive vibes.

Rails (or ... ?)

- sorry Java (deployment, complexity)
- sorry C# (deployment)
- sorry C++ (10 years ago, core search algorithm is in C++)
- Python (maybe with Django)
- php (not yet ...)
- Ruby is clean, OO is OK, perfectionist :-)
- Rails seems easy MVC framework

Rails risks: scalability

- Tom Klaasen: “How many *transactions* will you need?” => 0
- scalability becomes a non-issue
- 1 master database, N slaves
- Rails is just 1 interface to the database
- if ever needed, other “read-only” interfaces to slave database(s) can be set-up or caching can resolve most issues

Rails risks: security

- main risk is in our programming style
- Rails motivates clean styles
- e.g. `before_filters` with “:except”
- `before_filter :update_access_required, :except => [:search, ...`
- we found ZERO bugs in the production framework
- 1 difference of opinion on the test framework (`disable_referential_integrity`)

Search: Sphinx

- AJAX “live” search and “star expansion”
- milliseconds count
- stability and speed of Sphinx are praised
- Ultrasphinx actively maintained
- performance of Sphinx is tremendous (pure C++), typically < 0.001 second
- uses some “reverse tables”

Search and Active Record

- Ultrasphinx was fetching all data through ActiveRecord (1 ms for search, 69 ms for 20 separate AR fetches)
- used my own upload of the actual records
- 1 day later: Evan Weaver checked in improvement that fetches data in 1 combined database access:
“Too late, it's merged”
==> call that agility

Search implementation

In ../app/models/job.rb :

```
is_indexed :fields => [  
  'searchable_employer_job',  
  'searchable_location_general',  
  { :field => 'searchable_address', :as => 'job_address' },  
  { :field => 'searchable_region', :as => 'job_region' },  
  { :field => 'sort_date', :sortable => true }  
],  
:conditions => "active IS TRUE"
```

Search implementation (2)

In ../app/controllers/jobs_controller.rb :

```
@search = Ultrasphinx::Search.new(  
  :query => build_query(@search_job, @search_address,  
    @search_region, ""),  
  :per_page => @per_page,  
  :page => params[:page],  
  :sort_by => 'sort_date',  
  :sort_mode => 'descending')
```

Search implementation (3)

- rake ultrasphinx:configure
- rake ultrasphinx:index
- rake ultrasphinx:daemon:start
- cron ==> index

Search implementation (4)

In ../app/views/jobs/_search.rhtml :

```
<%= will_paginate(@search, :params => { :search_job =>
@search_job, ...}) if @search %>
```

Yields pagination links (GET requests):

AJAX and pagination

- AJAX updates the “AjaxWrapper div” with Javascript, no new URL
- Pagination generates classic GET requests (showing a new URL)
- URL bar and search input fields and search results completely out of sync :-(
- solution: lowpro plug-in

pagination behavior (1)

add to application.js:

```
Event.addBehavior.reassignAfterAjax = true;  
Event.addBehavior({  
  'div.pagination a' : Remote.Link  
})
```

Now in request headers (of GET):

```
Accept text/javascript, text/html, application/xml, text/xml, */*  
X-Requested-With XMLHttpRequest  
X-Prototype-Version 1.6.0.1
```

(otherwise: Accept
text/xml,application/xml,application/xhtml+xml,text/html ...)

pagination behavior (2)

```
if request.post?  
  render :partial => 'search', :layout => false  
else  
  respond_to do |format|  
    format.html # search.html.erb  
    format.js do  
      render :update do |page|  
        page.replace_html 'ajaxWrapper', :partial => "search"  
      end  
    end  
  end  
end  
end
```

Reload and AJAX

- Reload normally goes back to original html and forgets all the changes that Javascript has executed:
complete out of sync between input fields and search results ...
- solution: use session to store input fields and pagination info

Reload and AJAX (2)

```
# init session on entry
session[:search_job] ||= ""

# old solution
# @search_job = params[:search_job] ||= "Leuven"

# Get parameters
@search_job = params[:search_job] ||= session[:search_job]

# Save new data into session data
session[:search_job] = @search_job
```

AJAX and the back/forward

- Reload keeps stuff in sync with session data on reload, but back/forward does **not** call the server, so not in sync @#\$%!
- probably there are better ways to do this, but the one way we found that works for now is ...
- we always call the _search partial even on back/forward.
- ==> register partial with onload

AJAX and back/forward

- change order ... of main program
- search view does `_not_` call the `_search` partial
- Javascript onload will call:
 - the `_search` partial (after the main page)
 - start looping of sponsor logo's
 - put the focus on the search box

AJAX and forward/back (2)

```
Event.observe(window, 'load', function ()  
{ update_sponsors(delay) }); // rotate sponsors
```

```
// Forced search on reload
```

```
Event.observe(window, 'load', ajiv_search);
```

```
// set focus to search input box
```

```
Event.observe(window, 'load', function ()  
{ document.search_box.search_job.focus(); });
```

Conclusions

- Rails is great:
 - easy to deploy
 - we have hit no production bugs
 - open source (we can watch/change)
- Learned a lot (and had lots of fun and little sleep)
- User + customer feedback to steer further developments