java - Prijedlozi #15654

xwiki, grails, sitemesh

26.10.2008 04:23 - Ernad Husremović

Status:	Odbačeno	**Početak:**	26.10.2008
Prioritet:	Normalan	**Završetak:**	
Odgovorna osoba:	Ernad Husremović	**% završeno:**	30%
Kategorija:		**Procjena vremena:**	0.00 sat
Ciljna verzija:			

Opis

http://www.xwiki.org/xwiki/bin/view/Main/WebHome

juče sam naletio članak gdje se pominje kako se sa glassfish v3 može podesiti xwiki ... pa rekoh hadje da pogledam šta je xwiki.

http://blogs.sun.com/alexismp/tags/xwiki

Putting the pieces together

....
GlassFish Embedded does not use OSGi but rather a more traditional class-loader hierarchy so I had to work around a "classical" JAR precedence issue:
• removed the log4j jar from the XWiki application and make it available as a library (appropriate Class-Path: entry in the manifest of the overall JAR).
• used <class-loader delegate="false"/> in sun-web.xml to avoid further library conflicts "(documented here)"http://docs.sun.com/app/docs/doc/819-3672/beagb?a=view

Using the GlassFish Embedded API is pretty straightforward :

```
import org.glassfish.embed.App;
import org.glassfish.embed.AppServer;

public class XWikiOnGlassFish {
    public static void main(String[] args) throws IOException {

        AppServer glassfish = new AppServer(8080);
        App app = glassfish.deploy( new File("/path/to/xwiki-enterprise-web-1.5.2.war") );
    }
}
```

Historija

#1 - 26.10.2008 04:25 - Ernad Husremović

http://grails.org/Grails+vs+Rails+Benchmark

Overall, as expected, given its base on solid Java technologies, Grails out performs Rails (100% Ruby) in pretty much every test. As I said at the start I'm no Rails performance tuning wizard, so if someone can suggest ways to get Rails to perform better please say so. We are conducting these tests purely to inform concerned Grails users who are asking for figures, and hope to review them as Grails progresses.

Interestingly, Grails, at least on this machine, sees performance increases when lowering the thread pool to 10, whilst Rails suffers when 10 mongrels are configured with Pound.

Grails hasn't undergone much optimisation at all yet and there are certainly areas we can improve on, GSP rendering being one. GSP is of course a much more complex view rendering technology in comparison to RHTML due to its support for dynamic tag libraries, plus Sitemesh decoration. However, there are definitely areas in GSP, such as the way it uses exceptions, that can be improved upon.
....

#2 - 26.10.2008 04:45 - Ernad Husremović

http://auth.opensolaris.org/restructuring.html

Consistent opensolaris.org 'look and feel'

Although we are moving towards a model where opensolaris.org functionality is provided by a set of loosely-connected web applications, we still want to preserve a common look and feel across all the components. At present all the 'chrome' associated with the site (menus, icons, stylesheets etc.)

has to be manually replicated across each component webapp. The proposal is to use **SiteMesh** to provide a common "Look and Feel" mechanism for opensolaris.org. One advantage of SiteMesh is that although it is J2EE based, it can also be used to 'decorate' webapps that are written in other languages.

It is possible to configure SiteMesh for cross-context use withing a single web container, but external sites will need to have local copies of the SiteMesh configuration. This will unavoidably require coordination between the administrators of the various webapps making up opensolaris.org if changes to the SiteMesh configuration are needed, e.g. for bug fixing or site rebranding.
Content authoring

At present the content on opensolaris.org is entered using either raw HTML or with a opensolaris-only markup language called TML. It is proposed that this be replaced with a WYSIWYG editor such as TinyMCE, configured to accept only XHTML. This would make it easier to enforce consistent styling across the site. A further suggestion is to replace the current custom content management with a wiki-based system, based on a customised version of an existing wiki package such as XWiki.

An important constraint is that any chosen solution must have good localization facilities. These are needed both to support translation of the site content, and to allow user groups in particular geographies to provide their own language-specific areas of the site. For more details of the requirements and the work that has been done to date, see the OpenSolaris portals project.

Further work is needed in this area to determine the best solution.

#3 - 26.10.2008 04:47 - Ernad Husremović

na ovom dokumentu imamo interesantan dio i za site search:

Site search

At present the site search facility for opensolaris.org is embedded inside the webapp which provides the site content. When a page is edited, it is immediately reindexed by the webapp, which uses **Lucene** to implement the search functionality. This has an unfortunate side effect of massively inflating the footprint of the webapp - the current opensolaris.org webapp requires over 2Gb of RSS to run. This centralised mechanism is also obviously not suitable for an architecture consisting of distributed webapps.

The suggested architecture for the new search mechanism is to provide a **central webapp** that can provide search management facilities for **all the components of opensolaris.org**. This would use the **Solr** open source search server, which is based on Lucene. Solr provides an **XMLRPC interface that allows federated search management** - partitipating applications can add, modify and delete content from the central search server using the XMLRPC interface. In addition it also provides facilities for load balancing and index distribution.

#4 - 26.10.2008 04:51 - Ernad Husremović

- Naslov promijenjeno iz xwiki, grails u xwiki, grails, sitemesh

SiteMesh

You can also start by reading Will Iverson's Introduction to SiteMesh over at java.net or skip all this and download SiteMesh right away.

What Is It?

- SiteMesh is a web-page layout and decoration framework and web- application integration framework to aid in creating large sites consisting of many pages for which a consistent look/feel, navigation and layout scheme is required.
- SiteMesh intercepts requests to any static or dynamically generated HTML page requested through the web-server, parses the page, obtains properties and data from the content and generates an appropriate final page with modifications to the original. This is based upon the well-known GangOfFour Decorator design pattern.
- SiteMesh can also include entire HTML pages as a **Panel within another page**. This is similar to a Server-Side Include, except that the HTML document will be modified to create a visual window (using the document's Meta-data as an aid) within a page. Using this feature, Portal type web sites can be built very quickly and effectively. This is based upon the well-known GangOfFour Composite design pattern.
- SiteMesh is built using Java 2 with Servlet, JSP and XML technologies. This makes it ideal for use with J2EE applications, however it can be integrated with server-side web architectures that are not Java based such as CGI (Perl/Python/C/C++/etc), PHP, Cold Fusion, etc...
- SiteMesh is very extensible and is designed in a way in which it is easy to extend for custom needs.

#5 - 26.10.2008 04:56 - Ernad Husremović

SiteMesh - Building SiteMesh Decorators

Once SiteMesh has been installed and configured, you can begin writing **decorators for your web application**.

Introduction

Decorators are the pages that "decorate" the original, requested page **(the page that is handed to the SiteMesh filter from the web container)**. Most (HTML) decorators are a combination of:

- meta tags (keywords, description, author)
- stylesheet (CSS)
- header
- navigation
- footer

- copyright notice

First, define what different navigation/layout schemes you need. For example: Do I need a default decorator (a standard one for all pages)? Do I have a special layout for the index page? Is the header needed for my documentation files? Do I need printable version of my website?
Web Application Structure

Here is an example structure of a web application. This is not needed for SiteMesh to work.
/decorators

Directory containing all decorator files (e.g. main.jsp, printable.jsp).
/includes

Directory containing all files to be included into other files (e.g. header.jsp, footer.jsp, copyright.jsp).
/images

Directory containing all images (e.g. background.gif, logo.gif).
/styles

Directory containing all .CSS styles (e.g. ie4.css, ns4.css).
/scripts

Directory containing all scripts (JavaScript files).

Good practices:

- Define a stylesheet to use in the entire application and include it using this script.
- Use includes in your decorators (e.g. includes/navigation.jsp, includes/style.jsp).
- Try not to refer to the absolute root ("/") path. Use <%=request.getContextPath()%>/ instead. This will make life easier when moving your web application under another context path.
- Making your decorators compatible with multiple browsers (Mozilla, IE, Opera, ...) will (probably) make your entire application (all decorated pages) compatible.
- Be careful when using frames, because decorators may NOT be applied to frames (FrameSetDecoratorMapper).

My First Decorator

Basically, all you need to know is what decorator tags you can use. The title, head and body tags are most used.
Here is an example of a decorator (save it as /decorators/main.jsp):

```
1: <%--
2: % This is the main decorator for all SOMECOMPANY INTRANET pages.
3: % It includes standard caching, style sheet, header, footer and copyright notice.
4: --%>
5: <%@ taglib uri="http://www.opensymphony.com/sitemesh/decorator" prefix="decorator" %>
6: <%@ include file="/includes/cache.jsp" %>
7: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
8: <html>
9: <head>
10: <title><decorator:title default="INTRANET" /></title>
11: <decorator:head />
12: <%@ include file="/includes/style.jsp" %>
13: </head>
14: <body bgcolor="#FFFFFF" background="<%=request.getContextPath()%>/images/bg.gif">
15: <script type="text/javascript">window.status = "Loading: <decorator:title default="INTRANET" />...";</scri
pt>
16: <%@ include file="/includes/header.jsp"%>
17: <table width="100%" border="0" cellspacing="0" cellpadding="0">
18: <tr>
19: <td height="20" nowrap> </td>
20: </tr>
21: <tr>
22: <td width="1%" nowrap> </td>
23: <td width="16%" valign="top" nowrap>
23: <script type="text/javascript">window.status = "Loading: Navigation...";</script>
24: <%@ include file="/includes/navigation.jsp" %>
25: </td>
26: <td width="2%" nowrap> </td>
27: <td valign="top">
28: <br>
29: <script type="text/javascript">window.status = "Loading: Document body...";</script>
30: <div class="docBody"><decorator:body /></div>
31: </td>
32: <td width="1%" nowrap> </td>
33: </tr>
34: </table>
35: <br>
```

```
36: <%@ include file="/includes/footer.jsp" %>
37: <%@ include file="/includes/copyright.jsp" %>
38: <script type="text/javascript">window.status = "Done";</script>
39: </body>
40: </html>
```

- Line 1-4: An explanation of the decorator. This way different people working on the decorator are quickly up to speed.
- Line 5: This is needed for the decorator: tags to work (also needed on all pages that work with inline decorators (page:applyDecorator).
- Line 6: Sets the necessary response headers to let the browser cache the page. Omit this line if your application is real dynamic (changing data).
- Line 10: If the requested page doesn't have a title, the default title is used ("INTRANET").
- Line 15: The status bar gets a message when the page is loading.
- Line 30: The entire body of the requested page has the docBody class. This way the navigation and body do not have to have the same font.

Now open WEB-INF/decorators.xml with your favorite editor and let SiteMesh know there is a decorator (with a mapping):

```
<decorators defaultdir="/decorators">
    <decorator name="main" page="main.jsp">
          <pattern>/*</pattern>
    </decorator>
</decorators>
```

Now deploy the web application, go to the welcome page, and the main decorator will be applied.

#6 - 26.10.2008 04:58 - Ernad Husremović

http://www.opensymphony.com/sitemesh/flow.html

#7 - 26.10.2008 05:02 - Ernad Husremović

http://kasparov.skife.org/blog/src/java/rails-for-strutters-2.html

#8 - 26.10.2008 05:05 - Ernad Husremović

Rails for Struts-ters, Part 2: The Views

We looked at the basics of controllers for Ruby on Rails. Now lets take a look at rendering stuff. The default view technology for most Struts apps is JSP, which works reasonably well. The default renderer for Rails is ERB.

An ERB template would be named foo.rhtml and might look something like:

```
<h1>RSVP</h1>
<p>Just to make sure we've got it right, you are?</p>
<table>
<% for invite in @invitations %>
  <tr>
    <td>
      <b><%= link_to invite.name , :action => 'verify_invitation', :id => invite.id %></b>
    </td>
  </tr>
<% end %>
</table>
```

Which nicely highlights a big potential beef -- there are scriptlets there! Argh, woe, agony. Well, maybe. You can form your own opinion, but lets take a look at it. Let's look at the scriptlet used here:

```
<% for invite in @invitations %>
  ...
<% end %>
```

This loops through invites in the instance var @invitations. The instance var is defined in the controller (I am trying to follow DHH's advice here, I would prefer to expose it via a helper, personally). The equivalent jsp would be:

```
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
...
<logic:iterate name="invite" property="invitation">
  ...
</logic:iterate>
```

Disturbingly similar for a very similar thing. The big difference being that the erb template uses ruby, whereas the jsp defines a domain specific language via tag libraries. The JSP lets you drop into Java, but that is feared-and-not-to-be-spoken-of, mostly because people have a bad habit of putting application logic in JSP's, which was the ASP and PHP model (from which JSP was copied, go figure).

Now, RoR supports the equivalent of tag libraries, however, just to make things more fun, look at the lines inside the loop:

```
<tr>
  <td>
    <b><%= link_to invite.name , :action => 'verify_invitation', :id => invite.id %></b>
  </td>
</tr>
```

The equivalent JSP/Struts would be:

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
...
<tr>
  <td>
    <b>
      <html:link action="/VerifyInvitation" paramId="id" paramName="invite" paramProperty="id">
        $invite.name
      </html:link>
    </b>
  </td>
</tr>
```

The Struts HTML taglib uses a jsp taglib DSL again to define links within the application, in this case Rails uses a Helper named link_to which is built in to the rails framework (Actionpack specifically). Helpers can be defined in several places. For helpers which are to be globally available, defining them on the application_helper is natural, such as:

```
# The methods added to this helper will be available to all templates in the application.
module ApplicationHelper

  # roles == list of roles required to see text
  def secure(text, roles = [])
    user = @session['user']
    return '' unless user
    if roles.all? { |role| user.roles.map {|r| r.name }.include? role  }
      return text
    else
      return ''
    end
  end

end
```

Methods defined on the ApplicationHelper are exposed as helpers to all views in the application. This example defines a single helper, secure which will return (and hence render) the text passed as the first argument if and only if the logge din user is in all of the roles passed in the array for the second argument, so it would be used like:

```
<%= secure link_to('Accepted Guests', :controller=>'rsvp', :action=>'accepted'), ['admin'] %>
```

Which will render a link to an action which lists the guests who have accepted invitations if the logged in user is in the role admin.

In addition to defining helpers ~globally, you can define per-controller helpers in a [ControllerName]Helper module, which will be automatically mixed into the controller it is executing for, so it will have access to that controllers scope. Helper modules can be explicitely mixed into a controller view the helper function, in addition to the implicit helper module. Finally, you can expose methods on the controller itself as helpers via the helper_method and helper_attr functions. Basically, you can define helpers in ways that are convenient for you. You can also declare that you need helpers, using the helper function directly in the template, which is analogous to declaring that you need a tag library in JSP.

That is a basic overview, now for more fun stuff, like form handling. I won't go into explaining Struts ActionForms, you presumably know all about them already. Rails places form parameters into an @params instance var on the controller. It supports structured form parameters, like the EL properties in Struts, but via more-natural-for-ruby hashes, take for example the following form:

```
<form action="<%= url_for :action => 'register', :controller => 'rsvp' %>" >
  <table>
    <% @invite.number_of_seats.times do |slot| %>
    <tr>
      <td>Guest Attending:</td>
      <td>
          <input  type="text"
                  name="invite_names[<%= slot %>]"
                  length="30"
                  <%= "value='#{@invite.split_names[slot]}'" %>/>
      </td>
      <td>
        <input type="radio" name="dinner[<%= slot %>]"
               checked="true" value="chicken" /> <b>Chicken</b> or
        <input type="radio" name="dinner[<%= slot %>]"
               value="seafood" /> <b>Salmon</b>
      </td>
```

```
    </tr>
    <% end %>
    <tr>
      <td> </td>
      <td><input type="submit" value="Continue"/>
      <td> </td>
    </tr>
  </table>
</form>
```

An invitation has a specified number of seats attached to it (ie, if you invite a family of four, it is four, a couple it is two). We loop this number of times:

```
<% @invite.number_of_seats.times do |slot| %>
  ...
<% end %>
```

Assigning the index of the current seat to slot. Now, with this we build up the input names, such as:

```
<input type="radio" name="dinner[<%= slot %>]"
       checked="true" value="chicken" /> <b>Chicken</b> or
<input type="radio" name="dinner[<%= slot %>]"
       value="seafood" /> <b>Salmon</b>
```

where the name of the parameter is dinner[<%= slot %>], so we'll have things like dinner[1] and dinner[2]. On the controller side

```
def register
  names = @params['invite_names']
  names.each do |key, name|
    if name and name != ''
      u = User.new('login' => name.downcase.strip, 'name' => name.strip)
      u.attending = true
      u.meal_preference = @params['dinner'][key]
      u.save
      @session['user'] = u unless @session['user']
    end
  end
  invite = Invitation.find(@session['invite_id'])
  invite.accepted = true
  invite.save
  redirect_to :action => 'thank_you' and return
end
```

we'll pull those out the indexed dinners (indexed the same as the user names ;-)

```
names.each do |key, name|
  ...
  u.meal_preference = @params['dinner'][key]
  ...
end
```

Note that names is a hash, so names.each will include the key and the value. In this case the key is an number, but it is not in fact ordered, it is like a Map. Despite iterating over names (to pull out the other info), we are going back to @params for the dinner hash and looking up values in it. I picked this example as it is somewhat complicated, most often you don't need to do much this fancy.

Now, JSP 2.0 introduced JSP 2.0 Tag Libraries, which are basically global macros. It seems that the Struts mail reader example app doesn't use them, but we'll assume you know about them. They are a nice way of thinking about Rails partials. Partials are basically little included snippets for things that are likely reused, or complicated, etc. Take for example a snippet for rending a table of things in a gift registry:

```
<% gifts = gift_table %>
<table>
<% for gift in gifts %>
  <tr>
    <td><%= link_to gift.name, :action => 'show', :id => gift.id %></td>
    <% unless gift.being_given? %>
      <td>
        <b><%= link_to 'Give this Gift', :action => 'give', :id => gift.id %></b>
      </td>
    <% end %>
  </tr>
<% end %>
</table>
```

which is used here:

```
<h2>Gifts Not Yet Selected</h2>
```

```
<%= render_partial 'gift_table', @gifts.find_all {|g| not g.being_given? }%>
<h2>Gifts Being Given</h2>
<%= render_partial 'gift_table', @gifts.find_all {|g| g.being_given? }%>
```

The partial takes a single argument named gift_table which is a collection of gifts to be rendered in a table. There is actually a convenience function for this exact use case (passing a collection in), but I didn't see it when I wrote this, so didn't use it, oops. The render_partial helper in the template passes in the collection to be assigned to the name 'render_partial'. Works nicely.

In JSP 2 tag file format, this might look something like (table.tag, totally untested and unvalidated, and just as pseudo-template ;-)

```
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<%@ attribute name="gifts" required="true" %>

<table>
<logic:iterate name="gift" property="gifts">
  <tr>
    ...
  </tr>
</logic:iterate>
</table>
```

and be called via

```
<%@ taglib prefix="gifts" tagdir="WEB-INF/tags/gifts/" %>
...
<gifts:table gifts="unselectedGifts" />
<gifts:table gifts="selectedGifts" />
```

Final thing to look at in views is **layouts**. The quasi-official tool for this in Struts is Tiles, but I prefer **Sitemesh**, so will use a sitemesh example. Rails has Layouts, which are the same conceptually as Sitemesh decorators. Here is one

```
<html>
<head>
  <title>Brian and Joy's Wedding</title>
  <link href="/stylesheets/scaffold.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <%= render_menu %>
  <%= @content_for_layout %>
</body>
</html>
```

The layout gets wrapped around any rendered output, with the output going at the <%= @content_for_layout %> point. The easiest way to use a layout for all views is to just have a layout named application.html, and it will be used implicitely. You can explicitely use a layout by declaring it in the controller, a la

```
class GiftController < ApplicationController
  layout 'scaffold'
  before_filter :require_logged_in
```

which will have it look for scaffold.rhtml to use as a layout. Sitemesh uses an external XML file to configure these, under typical usage, but it is conceptually the same.

#9 - 26.10.2008 05:08 - Ernad Husremović

http://platform.xwiki.org/xwiki/bin/view/AdminGuide/InstallationGlassFish

#10 - 26.10.2008 06:06 - Ernad Husremović

xwiki testovi

instalirao 1.7 developer milestone 1 zip kod sebe (sa hsqldb-om):

hernad@nmraka-1:/data/home_1/hernad/java/xwiki-enterprise-hsqldb-1.7-milestone-1$

podesio xwiki.cfg
hernad@nmraka-1:/data/home_1/hernad/java/xwiki-enterprise-hsqldb-1.7-milestone-1/webapps/xwiki/WEB-INF$ cat xwiki.cfg | grep super

```
# Enable to allow superadmin. It is disabled by default as this could be a security breach if
xwiki.superadminpassword=sxxxxxn
```

#11 - 26.10.2008 06:08 - Ernad Husremović

- % završeno promijenjeno iz 0 u 30

xwiki je vidi se pravo zreo proizvod

ima richedit tekst editor sjajan.

Posebno je intersantna "insert macro" opcija sa kojom ubacuješ snippet-e kao što je "mindmap", "jabber" itd

#12 - 27.10.2008 04:43 - Ernad Husremović

dodavanje novog plugin-a, aplikacije

hernad@nmraka-1:~/java/xwiki-enterprise-hsqldb-1.7-milestone-1/webapps/xwiki/WEB-INF$ cp
/home/hernad/Desktop/xwiki-plugin-activitystream-1.1.jar lib/
hernad@nmraka-1:~/java/xwiki-enterprise-hsqldb-1.7-milestone-1/webapps/xwiki/WEB-INF$ vi xwiki.cfg
hernad@nmraka-1:~/java/xwiki-enterprise-hsqldb-1.7-milestone-1/webapps/xwiki/WEB-INF$ vi hibernate.cfg.xml

hernad@nmraka-1:~/java/xwiki-enterprise-hsqldb-1.7-milestone-1/webapps/xwiki/WEB-INF$ cat xwiki.cfg | grep activitystream

```
                    com.xpn.xwiki.plugin.activitystream.plugin.ActivityStreamPlugin
```

-INF$ cat hibernate.cfg.xml | grep activitystream

```
    <mapping resource="activitystream.hbm.xml"/>
```

#13 - 28.10.2008 01:09 - Ernad Husremović

http://obsessivecollaborator.com/2008/01/xwiki.html

.. Xwiki

I went back to the Wikimatrix in search of another good open source WYSIWYG wiki and found Xwiki.

Xwiki had an easy to use self-contained version that I was able to quickly and easily install on my server. It turns out that a few days later that version was updated, and updating that version is not straightforward. I had to take the new zip version and copy out the necessary directories to avoid writing over my previous data. But after that I was updated. Then I went and added the start script to my startup scripts, added some Apache rules, and all was well.

Xwiki is a good looking Wiki with useful WYSIWYG editing functions and very access, group and user control.

I found the access control particular straightforward and well implemented.

#14 - 26.05.2010 15:43 - Ernad Husremović

- Status promijenjeno iz Dodijeljeno u Odbačeno