FacesFreeway

Faces Freeway

http://facesfreeway.l3x.net

Mario Ivankovits <imario ~~at~~ apache.org> - 2005-12-03

One thing we can learn from "Ruby on Rails" http://www.rubyonrails.org/ is the speed they have to get up and running a simple database driven application.

I am not a friend of such "designed" applications, but I wont start a "Ruby on Rails" discussion as there are undoubt fields for such applications. However, I would like to speed the development time needed for simple "master data" pages with JSF too.

Code Name: **Faces Freeway** 😉

"Ruby on Rails" archives this speed with two steps:

1. completely generate objects at runtime out of the database metadata and generate the render response out of them
2. generate "static" (whatever static means in the context of Ruby) objects and *.rhtml (simmilar to jsp) pages

So you can change your database and immediately see the result, once your are fine with it you can generate the static.

Faces Freeway wont support solution 1. This requires to use plain JDBC and (for me) this is a NO NO.

However, Faces Freeway should avoid the need to generate static pages as long as possible. My experience in the past showed me that this is really possible - I already created such a framework, but it is heavily integrated - and thus not reuseable.

The "reference implementation" of Faces Freeway will use

1. EJB3 entities to get in touch with the database
2. Hibernate to persist them
3. and annotations to setup the JSF comonents used.
4. however, it should be possible to use the Faces Freeway tags without direct communication with any persistence

- Smart default values will be used to avoid step 3 as long as possible.
- It should be possible to replace EJB3 and Hibernate by other implementations.

What are the steps to get a Faces Freeway application up and running:

1. Given you already have a database: generate POJO from database using hibernate-tools
2. generate the POJO jsf page using the (to be created) Faces Freeway generator (ff-gen ?)

I know, this is not that speedy than "Ruby on Rails", but speedier than doing all by hand.

So whats the result? I think the best will be to show some example code - nocked out of my brain.

Sample with Hibernate

Entity - Blog.java

```
@Entity(access = AccessType.FIELD)
@Table(name = "Blog")
public class Blog
{
        @Id(generate=GeneratorType.AUTO)
        @Column(name = id")
        @NotNull
        @ReadOnly
        private Long id;

        @Column(name = "date")
        @Type(type="org.joda.time.contrib.hibernate.PersistentDateTime")
        @NotNull
        private DateTime date;

        @Column(name = "comment")
        @InputHtml
        @NotNull
        private String comment;

... getters/setters ...
}
```

A simple page to render the input form JSP - Blog.jsp

```
... jsp pre ...
        <ff:beanForm class="Blog value="#{beanForm.bean}">
                <f:facet name="id">
                        <h:outputText />
                </f:faces>
                <f:facet name="comment">
                        <t:inputHtml />
                </f:faces>
        </ff>
        <h:commandLink action="#{beanForm.beanNew}" />
        <h:commandLink action="#{beanForm.beanCopy}" />
        <h:commandLink action="#{beanForm.beanSave}" />
        <h:commandLink action="#{beanForm.beanDelete}" />
... jsp post ...
```

A simple page to render the list of entities JSP - Blogs.jsp

```
... jsp pre ...
        <ff:beanList class="Blog value="#{beanForm.beanList}">
                <f:facet name="comment">
                        <t:inputHtml />
                </f:faces>
                <f:facet name="action">
                        <h:commandLink action="#{beanForm.beanListDelete}" />
                        <h:commandLink action="#{beanForm.beanListCopy}" />
                </f:faces>
        </ff>
... jsp post ...
```

Description:

- Blog.jsp and Blogs.jsp will be generated
- the "facet" stuff is already an example how to customize the output
- notice also the @InputHtml, @NotNull and @ReadOnly annotations in the entity, they will customize the output too. ** for each jsf component we will generate jdk 1.5 annotations - thus InputHtml will force the bean*-tags to use this control instead of the default - inputText in this case ** notNull means "required" ** ReadOnly means outputText

If you change the database and regen your POJOs the bean*-tags will pick them up automatically. Annotations like @OneToMany should result in an selectOne* component.

Plain Bean Sample

Entity - Blog.java

```
public class Blog
{
        @NotNull
        @ReadOnly
        private Long id;

        @NotNull
        private DateTime date;

        @InputHtml
        @NotNull
        private String comment;

... getters/setters ...
}

public class BlogController
{
        public void save(Blog blog)
        {
                ... your code ...
        }

        public void delete(Blog blog)
        {
                ... your code ...
        }
}
```

A simple page to render the input form JSP - Blog.jsp

```
... jsp pre ...
        <ff:beanForm class="Blog value="#{beanForm.bean}" controller="#{blogController}">
                <f:facet name="id">
                        <h:outputText />
                </f:faces>
                <f:facet name="comment">
                        <t:inputHtml />
                </f:faces>
        </ff>
        <h:commandLink action="#{beanForm.beanNew}" />
        <h:commandLink action="#{beanForm.beanCopy}" />
        <h:commandLink action="#{beanForm.beanSave}" />
        <h:commandLink action="#{beanForm.beanDelete}" />
... jsp post ...
```

Features

- Direct use of entities
- Indirect user through controller
- beanForm with 1:n (e.g. blogs -> comments)
- beanList with an way to filter the list

Conclusion

Ok, much has been written now, many details are inexact - at all this is damn much work, but with every database table we save time.

Now lets discuss about it on the myfaces-dev mailing list.

Comments

Anyone interested in this discussion may also want to check out the Grails project (written in Groovy with Hibernate Persistance layer) at http://grails.codehaus.org/ The quickstart example allows you to build an end to end web app in about 10 minutes, including all downloads and setup. Nice! - Peder 🙂