JSF Renderer Interceptor (Jerry)

Rudy De Busscher

Version 0.9, ??/??/2017

Table of Contents

Introduction

Origin

On various occasions, I needed the RendererInterceptor concept of Apache MyFaces extensions Validation framework. I didn’t need the great validation features of the framework, so I found it an overhead, in terms of performance and memory consumption.

So I extracted that interface and the required classes and transformed it to use CDI to have the benefits of it. That is how Jerry (JSF Renderer Interceptor, JRI) was started. Jerry is the basis for **Octopus** from version 0.9.5 on but has also some usage scenarios on his own.

Since there are a lot of lookups done into the CDI bean management system, I copied some classes from **Apache DeltaSpike** (the BeanProvider) to make it easier to lookup the beans.

Atbash migration

Mapping between the old and the new maven artifacts.

Rubus artifact	Atbash Artifact
<groupId>be.rubus.web</groupId> <artifactId>jerry</artifactId>	<groupId>be.atbash.ee.jsf</groupId> <artifactId>jerry</artifactId>

Features

1. RendererInterceptor

   With the RendererInterceptor you have a before and after method for every major method in a Renderer which is `decode`, `encodeBegin`, `encodeChildren`, `encodeEnd` and `getConvertedValue`.

2. ComponentInitializer

   With ComponentInitializer it becomes easier to adjust the JSF components. They use a RendererInterceptor in the background and work is done in the `beforeEncodeBegin` phase.

There are some other small CDI features which are useful in any project

1. StartupEvent

   CDI event when the application is ready (JSF subsystem is ready)

2. Injectable Logger

   You can inject SLF4J Logger.

3. BeanProvider from Apache DeltaSpike

   Easier programmatically retrieval of CDI beans in code.

Configure your project

In case you don't use maven, you can just download the jar file and put in the `lib` folder of your project.

1. Open the project `pom.xml` file for editing.
2. Add the JCenter Maven repository where the dependency can be found.

```
<repository>
    <id>Bintray_JCenter</id>
    <url>https://jcenter.bintray.com</url>
</repository>
```

3. Add the Jerry module to the list of dependencies.

```
<dependency>
    <groupId>be.atbash.ee.jsf</groupId>
    <artifactId>jerry</artifactId>
    <version>0.9</version>
</dependency>
```

4. You are ready to use Jerry.

Releases

v0.9 (??/??/2017)

1. Migrate to Atbash namespace (Mvn artefact and package names)

v0.4.1 (25/06/2017)

1. Support for OWB proxies

v0.4 (17/12/2016)

1. Helper for dynamic configuration values (DynamicConfigValueHelper)
2. Warning when for attribute value not found (RequiredMarkerInitializer)

v0.3 (08/03/2016)

1. @ConfigEntry has a member noLogging so that content isn't logged, except when -Djerry.log.all=true

v0.2.3 (15/02/2016)

1. Bug fixing (#8)

v0.2.2 (25/01/2016)

1. Bug fixing (#1)

v0.2.1 (26/11/2015)

1. @DataRange has now the equalsAllowed member to indicate that start date can be equal to end date.
2. Bug fixing (#6)

Usage scenarios

Component Initializer

Jerry can initialize any JSF component just before it will be rendered.

As example, the code is shown for setting the background color of required fields.

ComponentInitializer which makes each PrimeFaces InputText component with a reddish background color when it is required.

```
@ApplicationScoped
public class RequiredInitializer implements ComponentInitializer {
    @Override
    public void configureComponent(FacesContext facesContext, UIComponent uiComponent,
Map<String, Object> metaData) {
        InputText inputText = (InputText) uiComponent;
        if (inputText.isRequired()) {
            String style = inputText.getStyle();
            if (style == null) {
                style = "";
            }
            inputText.setStyle(style + " background-color: #B04A4A;");
        }
    }

    @Override
    public boolean isSupportedComponent(UIComponent uiComponent) {
        return uiComponent instanceof InputText;
    }
}
```

This are the important aspects of the code.

1. Implement the `ComponentInitializer` interface.
2. Annotate the class with `@ApplicationScoped` CDI scope.
3. Define in the `isSupportedComponent` method if this ComponentInitializer should handle the component.

4. Perform the required functionality in the `configureComponent` method.

The metaData parameter is filled up by Valerie, the (Bean) validation companion of Jerry. In the advanced use case scenarios, there is also an example how you can use it using only Jerry features.

Startup Event

You can use the CDI event StartupEvent to perform any initialization when your application is deployed and ready on the server.

Log some message when application is ready

```
    public void onStartup(@Observes StartupEvent startupEvent) {
        System.out.println("Ready to roll"); // Please use logger !
    }
```

You can also using the startup EJB singleton beans to perform some initialization. This is preferred if the initialization does some database actions.

Injectable Logger

Jerry uses SLF4J as logging facade. You can inject such loggers by creating a simple Producer method. That method is available within Jerry and thus injectable loggers can be used.

Usage of injectable logger.

```
@Inject
private Logger logger;

public void doSomething() {
    logger.info("Performed the doSomething");
}
```

The type of logger is `org.slf4j.Logger`.

BeanProvider

The BeanProvider class, copied from Apache DeltaSpike, makes it very easy to retrieve a CDI Bean.

If your project isn't using DeltaSpike, but is using Jerry (or Octopus), you can use the `BeanProvider` which is provided with it.

Examples of BeanProvider.

```
BeanProvider.getContextualReference(DateProvider.class, true);

BeanProvider.getContextualReferences(ComponentInitializer.class, true, false);
```

The first statement retrieve the CDI bean implementing the DateProvider interface. The second parameter indicates that the bean can be optional and thus when no bean is found, null is returned (An exception will be thrown in the case when no bean was found and false was specified as value).

The second statement retrieves all CDI beans, as a list, that implement the ComponentInitializer interface. The true indicate the optionality again (empty list when no beans found). The second boolean is used to indicate if default scoped beans (beans without an annotation) should be included.

Methods work also with a concrete class, not only with interfaces. And it will honour all CDI rules like for instance @Specializes and @Alternative.

Advanced usages

RendererInterceptor

TODO

Override configuration

TODO

Technical aspects

How RendererInterceptor is applied

TODO

Ordering of RendererInterceptors

TODO

ComponentStorage, MetaDataHolder, MetaDataEntry and MetaDataTransformer

TODO