JSF utils

Rudy De Busscher

Version 1.0.0, 02/01/2021

Table of Contents

Release notes

1.0.0

- Compiled against JSF 3.0 (Jakarta EE 9 namespace!)

Setup

Add the following artfact to your maven project file for using the programmatic CDI utilities.

```
<dependency>
    <groupId>be.atbash.jakarta.utils</groupId>
    <artifactId>utils-jsf</artifactId>
    <version>1.0.0</version>
</dependency>
```

Requirements

These utilities work only with JSF 3.0 (thus Jakarta EE 9) running on JDK 8 and JDK 11.

Features

General JSF Utilities

```
be.atbash.util.JsfUtils#isRenderResponsePhase
```

verify we are in the render response phase.

```
be.atbash.util.JsfUtils#createMethodExpression(String, Class<?>, Class<?>... )
```

Creates a method expression (like #{bean.doSomething}) programmatically. The second parameter defines the return type, the last parameter are the parameters of the method.

Component utilities

Some utilities related to JSF components.

```
be.atbash.util.ComponentUtils#getValue(UIComponent, FacesContext)
```

Return the value of the **value** property. It first checks if there was a *value* property defined with an

EL expression on the component (*value="#{...}"*). If such a ValueExpression is found, it evaluates it. If this is not found and the UIComponent is a **ValueHolder** instance, it returns the 'hardcoded' value of *value* property.

```
be.atbash.util.ComponentUtils#isRequired(UIComponent, FacesContext)
```

Returns the value of the **required** property or the default *false* when such property is not specified for the component. It first checks if there was a *required* property defined with an EL expression on the component (*required="#{...}"*). If such a ValueExpression is found, it evaluates it. If this is not found and the UIComponent is a **EditableValueHolder** instance, it returns the 'hardcoded' value of *required* property.

```
be.atbash.util.ComponentUtils#getStyle(UIComponent, FacesContext)
```

Return the value of the **style** property. It first checks if there was a *style* property defined with an EL expression on the component (*style="#{...}"*). If such a ValueExpression is found, it evaluates it. If this is not found and the UIComponent is a **HtmlInputText**, **HtmlSelectOneMenu** or **HtmlOutputLabel** instance, it returns the 'hardcoded' value of *style* property.

```
be.atbash.util.ComponentUtils#getStyleClass(UIComponent, FacesContext)
```

Return the value of the **styleClass** property. It first checks if there was a *styleClass* property defined with an EL expression on the component (*styleClass="#{...}"*). If such a ValueExpression is found, it evaluates it. If this is not found and the UIComponent is a **HtmlInputText**, **HtmlSelectOneMenu** or **HtmlOutputLabel** instance, it returns the 'hardcoded' value of *styleClass* property.

```
be.atbash.util.ComponentUtils#getMaxLength(UIComponent, FacesContext)
```

Return the value of the **maxLength** property. It first checks if there was a *maxLength* property defined with an EL expression on the component (*maxLength="#{...}"*). If such a ValueExpression is found, it evaluates it. If this is not found and the UIComponent is a **HtmlInputText** instance, it returns the 'hardcoded' value of *maxLength* property.

```
be.atbash.util.ComponentUtils#getAttributeValue(UIComponent, String, Class<T>)
```

Return the attribute value named by the String parameter for custom a component. It looks for a 'fixed' value and an ValueExpression (#{...}). In the latter case the expression is evaluated. The third parameter defines the type of the attribute and is used as the generic type result.

```
be.atbash.util.ComponentUtils#findTargets(UIComponent, String)
```

Search *UIComponents* with the JSF component tree. It is an adapted algorithm from the default available one from JSF and allows to search multiple targets (by performing the search multiple

times, once for each id)

It is an adapted algorithm in order to find the intended component when there are components with the same id (in different naming containers of course). But be aware that it may result in slower performance if the search starts 'at a not optimal place' within the component tree. This is the algorithm

- When the target is empty, it returns the component (specified as parameter) itself.
- When the target parameter contains multiple ids (separated with ,) the search is performed multiple times, once for each item found in the target parameter
- The target is search by using the *UIComponent.findComponent()* method.
- When no match found, the search is performed again but now with the parent
- This repeat continue until the root of the component tree is reached or a component is found.

```
be.atbash.util.ComponentUtils#processTargets(UIComponent, String, ComponentCallback)
```

Search *UIComponents* with the JSF component tree and for each component the **ComponentCallback** is called. It is an adapted algorithm from the default available one from JSF and allows to search multiple targets (by performing the search multiple times, once for each id)

It is an adapted algorithm in order to find the intended component when there are components with the same id (in different naming containers of course). But be aware that it may result in slower performance if the search starts 'at a not optimal place' within the component tree. This is the algorithm

- When the target is empty, it returns the component (specified as parameter) itself.
- When the target parameter contains multiple ids (separated with ,) the search is performed multiple times, once for each item found in the target parameter
- The target is search by using the *UIComponent.findComponent()* method.
- When no match found, the search is performed again but now with the parent
- This repeat continue until the root of the component tree is reached or a component is found.

The method **handle** has a custom component parameter flag. When the target is not found, the component itself is used as parameter with the custom component flag set.

Testing with FacesContext

In code, we sometime use this snippet to have access to the *FacesContext*

```
   FacesContext.getCurrentInstance()
```

But from within unit tests, this result in a null value since JSF is not activated. In order to make your code work in the test, you can make use of the `FakeFacesContext`.

```
FakeFacesContext.registerFake();
FakeFacesContext.registerFake(ExternalContext);
FakeFacesContext.registerFake(Application);
FakeFacesContext.registerFake(Application, ExternalContext);
```

With the above methods, we can register a FacesContext instance, and at the same time pass a (mock) implementation of ExternalContext, Application or both.

Also other methods of *FacesContext* are implemented, like the ones handling the FacesMessages.

Exceptions

JSF-DEV-01

When using the *ComponentUtils#findTargets* or *ComponentUtils#processTargets* with an invalid search id.

Examples of wrong structures are

- Contains spaces within id like *target id*
- When an intermediate component is specified which is not a *Naming container*. For ex. when *:frm:group:field* the *group* component is not a naming container.