Configuració scope view JSF a Spring 3

A qui va dirigit

Aquest how-to va dirigit a tots aquells desenvolupadors que vulguin utilitzar un scope a nivell vista a JSF 2.0 mitjançant els beans gestionats amb Spring.

Versió de Canigó

Els passos descrits en aquest document apliquen a la darrera versió del Framework Canigó 3.

Introducció

JSF 2.0 proporciona l'scope ViewScope. El cicle de vida d'un bean amb aquest ViewScope acaba quan l'aplicació navega a una altre vista.

Spring no proporciona aquest scope (els scopes proporcionats per Spring són singleton, prototype, request, session i global-session).

Si a la nostra aplicació volem gestionar els beans amb Spring i tenir disponible aquest Scope ho podem aconseguir gràcies al CustomScopeConfigurer de Spring.

Configuració scope view JSF a Spring 3

Exemple d'ús Spring ViewScope

Configuració de Dependències

No cal afegir cap nova dependència.

Creació de ViewScope

Primer de tot hem de crear la implementació del nou scope.

src/main/java/cat/gencat/provascope/utils/**ViewScope.java**

```
package cat.gencat.provascope.utils;

import java.util.Map;

import javax.faces.context.FacesContext;

import org.springframework.beans.factory.ObjectFactory;
import org.springframework.beans.factory.config.Scope;

public class ViewScope implements Scope {

    public Object get(String name, ObjectFactory objectFactory) {
        Map<String,Object> viewMap =
FacesContext.getCurrentInstance().getViewRoot().getViewMap();

        if(viewMap.containsKey(name)) {
            return viewMap.get(name);
        } else {
            Object object = objectFactory.getObject();
            viewMap.put(name, object);

            return object;
        }
    }

    public Object remove(String name) {
        return FacesContext.getCurrentInstance().getViewRoot().getViewMap().remove(name);
    }

    public String getConversationId() {
        return null;
    }

    public void registerDestructionCallback(String name, Runnable callback) {

    }

    public Object resolveContextualObject(String key) {
        return null;
    }
}
```

Aquesta classe implementa el comportament del Scope View de JSF, afegint el bean al ViewMap de JSF quan es crea i eliminant-lo quan perdeix el Scope.

Ara hem d'indicar a Spring la existència d'aquest nou Scope.

Configuració scope view JSF a Spring 3

src/main/resources/spring/**app-custom-beans.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
            http://www.springframework.org/schema/aop
                     http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
                     http://www.springframework.org/schema/context
                     http://www.springframework.org/schema/context/spring-context-
3.0.xsd">

    <context:annotation-config/>
    <context:component-scan base-package="cat.gencat.provascope.bean" />
    <context:component-scan base-package="cat.gencat.provascope.service" />

    <bean class="org.springframework.beans.factory.config.CustomScopeConfigurer">
        <property name="scopes">
            <map>
                <entry key="view">
                    <bean class="cat.gencat.provascope.utils.ViewScope"/>
                </entry>
            </map>
        </property>
    </bean>

</beans>
```

Afegim la part en negreta per a registrar el nou Scope.

Crear els beans

Es creen dos beans amb els que realitzarem les proves. En un configurarem el nou scope (View) i a l'altre el deixarem amb l'scope original (Singleton).

src/main/java/cat/gencat/provascope/bean/**OriginalScopeBean.java**

```
package cat.gencat.provascope.bean;

import org.springframework.stereotype.Component;

@Component("OriginalScopeBean")
public class OriginalScopeBean {

        private int contador = 0;

    public int getContador() {
                return contador;
        }

    public void setContador(int contador) {
                this.contador = contador;
        }

    public void incrementar() {
        contador++;
    }
}
```

Configuració scope view JSF a Spring 3

src/main/java/cat/gencat/provascope/bean/**ViewScopeBean.java**

```
package cat.gencat.provascope.bean;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component("ViewScopeBean")
@Scope("view")
public class ViewScopeBean {

        private int contador = 0;

    public int getContador() {
                return contador;
        }

    public void setContador(int contador) {
                this.contador = contador;
        }

    public void incrementar() {
        contador++;
    }

}
```

Com es pot veure s'han creat dos beans idèntics amb la única diferència que al ViewScopeBean s'ha afegit **@Scope("view")**

Aquests beans tenen un comptador i un mètode que incrementa el seu valor.

Ara creem dues vistes per a provar els dos comptadors.

src/main/webapp/views/**scopeOriginal.xhtml**

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<html   xmlns="http://www.w3.org/1999/xhtml"
                xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:f="http://java.sun.com/jsf/core"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:rich="http://richfaces.org/rich"
                xmlns:c="http://java.sun.com/jstl/core"
                xmlns:a4j="http://richfaces.org/a4j">

        <ui:composition template="layouts/template.jsf">
                <ui:define name="body">
                        <h3>#{msg.menuScopeOriginal}</h3>
                        <br></br>
                        <h:form>
                        Contador: <h:outputText id="contador"
value="#{OriginalScopeBean.contador}" />
                                <br></br>
                                <br></br>
                        <h:commandButton value="Incrementar"
actionListener="#{OriginalScopeBean.incrementar}" update="contador"/>
                        </h:form>

                </ui:define>
        </ui:composition>

</html>
```

Configuració scope view JSF a Spring 3

src/main/webapp/views/**scopeView.xhtml**

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<html   xmlns="http://www.w3.org/1999/xhtml"
                xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:f="http://java.sun.com/jsf/core"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:rich="http://richfaces.org/rich"
                xmlns:c="http://java.sun.com/jstl/core"
                xmlns:a4j="http://richfaces.org/a4j">

        <ui:composition template="layouts/template.jsf">
                <ui:define name="body">
                        <h3>Scope View</h3>
                        <br></br>
                        <h:form>
                        Contador: <h:outputText id="contador"
value="#{ViewScopeBean.contador}" />
                                <br></br>
                                <br></br>
                        <h:commandButton value="Incrementar"
actionListener="#{ViewScopeBean.incrementar}" update="contador"/>
                        </h:form>

                </ui:define>
        </ui:composition>

</html>
```

Les vistes tenen la mateixa funcionalitat, mostren el valor del comptador i amb un botó es pot incrementar.

Resultat

La primera crida a ambdues vistes mostra el Comptador a 0:

Si incrementem Scope Original, canviem de vista i tornem a Scope Original veiem com el comptador manté el valor que tenia abans de canviar de vista:

Configuració scope view JSF a Spring 3

Si incrementem Scope View, canviem de vista i tornem a Scope View veiem que el comptador es reinicia al tornar: