Enterprise Client CCEE Spring Integration

Table of contents

Download and install

Part of CaptainCasa installation

The CaptainCasa installation includes a file:

```
<installDir>
    resources
        addons
            eclnt_ccee.zip
            ...
        ...
    ...
```

This file contains the following library files that need to be added to your project:

```
eclnt_ccee.jar
eclnt_ccee_spring.jar
```

Add both files to your project, e.g. if using the CaptainCasa project structure add both files to the webcontent directory:

```
<project>
    webcontent
        WEB-INF
            lib
                …
                eclnt_ccee.jar
                eclnt_ccee_spring.jar
                …
```

Add the libraries required for Spring to WEB-INF/lib as well.

Maven

Use the CaptainCasa-Spring project-archetype

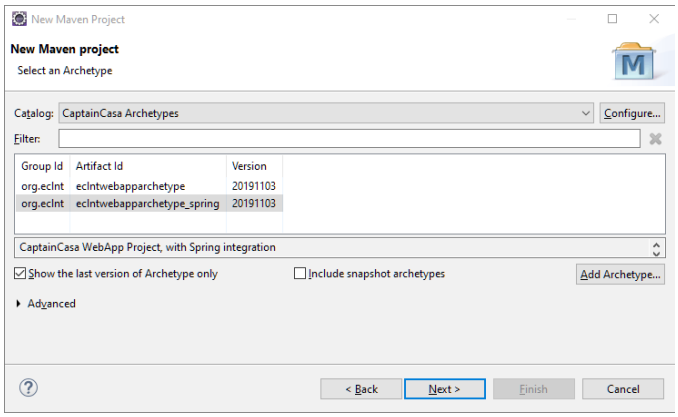
There is a project-archetype than contains the configuration of Spring that is explained in the next chapters. We recommend to use this archetype for creating your projects.

The archetype is available within the following remote catalog:

```
http://www.captaincasademo.com/mavenrepository/archetypecatalog.xml
```

In Eclipse the project is created in the following way:

- Select “File > New > Project...” from the menu.
- Select “Maven > Maven Project” in the popup dialog
- You then may select an archetype from a catalog of archetypes. If not yet done: create a catalog by selecting “Add catalog...” and defining the catalog-URL (http://www.captaincasademo.com/mavenrepository/archetypecatalog.xml).



Select the archetype “eclntwebapparchetype_spring”

- Do the other configurations in just normal way.

Similar steps need to be done if using e.g. NetBeans, or IntelliJ IDEA as development environment.

Do it yourself!

(Refer to the documentation “Setting up a Maven Project” if you require information about how to in general create a CaptainCasa Maven project.)

Add the dependencies that as follows.

```
<repositories>
    ...
    <repository>
         <id>org.eclnt</id>
         <url>https://www.captaincasademo.com/mavenrepository</url>
    </repository>
    ...
</repositories>

<properties>
    <cc.version>20191102</cc.version>
</properties>

...

<dependencies>
    ...
    <dependency>
        <groupId>org.eclnt</groupId>
        <artifactId>eclntccee</artifactId>
        <version>${cc.version}</version>
    </dependency>
    <dependency>
        <groupId>org.eclnt</groupId>
        <artifactId>eclntccee_spring</artifactId>
        <version>${cc.version}</version>
    </dependency>
    ...
</dependencies>
```

Source Code

Please note that all source code is available as part of the download. Sometimes textual explanations leave the impression of something very complex going on and indeed things are quite simple if you take a look inside – especially if you are already experiences with Spring.

Take a look into the classes – there are only a few ones, and inside there are only few lines of code!

Overview

Dispatcher

With CaptainCasa a page (.jsp) refers via expressions to its page bean. By default each page is managed by exactly one bean-class.

Example:

```
demo.jsp

...
<t:row id="g_1">
    <t:field id="g_2" width="100" text="#{d.DemoUI.firstName}"/>
</t:row>
...
```

The bean class is referenced through a so called dispatcher object which is represented by the leading "d" within all the expressions of the page.

The "d" is resolved within the file "/WEB-INF/faces-config.xml" to a Dispatcher-instance.

```
faces-config.xml (example)

...
<managed-bean>
    <managed-bean-name>d</managed-bean-name>
    <managed-bean-class>managedbeans.Dispatcher</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
...
```

And this Dispatcher-instance is the one to be responsible for transferring the name "DemoUI" into a corresponding Java object. - The Dispatcher itself is just an implementation of the java.util.Map interface, in which the get(...)-method is the one which is overridden correspondingly.

Dispatcher object resolution

The Dispatcher is the one that is affected by the integration to Spring: instead of following own rules how to resolve a name ("DemoUI") into an object instance – it requests the Spring-context to do the resolution.

Example, in the Spring-context there is a definition:

```
<beans ...>
    ...
    <bean id="DemoUI" class="test.DemoUI" scope="prototype">
        <property name="..." ref="..."/>
    </bean>
    ...
</beans>
```

Class DispatcherBySpringAccess

CaptainCasa provides within the eclnt_ccee_spring.jar-library a Dispatcher-implementation that does the object resolution via Spring. The following text will tell you details how to use this class.

Implementing Spring Access

There are so many ways to configure and to work with Spring. We will concentrate on the XML-way in this documentation, but of course all other ways (e.g. annotation-based) are supported as well.

Activate Spring in your web application

The spring framework needs to be integrated into the web.xml of your project:

```
...
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath:spring_context_webapplication.xml</param-value>
</context-param>
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
...
```

The web-application-context

Part of this definition is the name of the XML file that sets up the bean definitions for the web-application-context of Spring. In the example above we define the location to be “classpath:spring_context_webapplication.xml”.

This XML file is the normal Spring-XML for setting up bean instances.

```
File: <project>/<Java-resources>/spring_context_webapplication.xml


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

    <bean .../>
    </bean>

    <bean .../>
    </bean>

</beans>
```

This XML file already could be the end of the Spring integration story by telling you: define your beans here – and they are picked up by the Dispatcher at runtime. - But: we made things bit more structured...

The dialog-session-context

The class “DispatcherBySpringAccess” is an implementation of the “Dispatcher” and opens up one Spring-context for each dialog-session. This dialog-session-context is arranged as child to the web-application-context, so that all bean-definitions of the web-application-context are also available within the dialog-session-context.

At runtime there is one dialog session for each browser instance. If the user opens up the browser three times (either individual browsers or tabs inside one browser), then there are three dialog sessions on server side – and there are three instances of dialog-session-contexts as well.

(Please do not mix “dialog-session” with “http-session”. When using cookie-based session tracking there is one http-session which may span multiple browsers, but still there is one dialog-session per browser.)

There are two implementations of this dialog-session-context that come with eclnt_ccee_spring.jar:

```
DialogSessionXMLApplicationContext
    (extending Spring's ClassPathXmlApplicationContext)

DialogSessionAnnotationApplicationContext
    (extending Spring's AnnotationConfigApplicationContext)
```

Both classes provide configuration methods to set-up there internal processing. In case of the XML based approach the central method is...

```
DialogSessionXMLApplicationContext.setConfigLocation(...nameOfXmlFile...)
```

...in which the name of the XML-file is passed that is configuring the beans on dialog-session-context level.

Advantages of defining a dialog-sesison-context

In simple scenarios the dispatcher-name-resolution could directly access the web-application-context definitions. But, there are some advantages when using some explicit context below the web-application-context:

- The dialog-session-context represents on dialog session. So you are able to keep all objects which should be reach-able within one session-context – while still being able to access definitions of the web-application-context, which is set up as parent of the web-application-context.
- The dialog-session-context can be hot deployed easily. The definition of the corresponding bean-XML file can be exchanged without deeply reloading the web-application.
- ...and: it's just some nice structuring to not have all beans defined in one big context – but have a separation of UI-related page beans and logic-oriented web-application beans.

Whole setup

The Dispatcher-implementation “DispatcherBySpringAccess” is the one to create a dialog-session-context-instance for each dialog session. Its inner implementation does not directly create the corresponding context-instance, but requests the instance from the web-application-contenxt using the id “DialogSessionApplicationContext”.

This means: in the web-appplication-context you set up the prototype-definition for the dialog-context-instances:

```
File: <project>/<Java-resources>/spring_context_webapplication.xml


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

    ...
    ...
    <bean id="DialogSessionApplicationContext"
          class="org.eclnt.ccee.spring.context.DialogSessionXMLApplicationContext"
          scope="prototype">
```

```
        <property name="configLocation" value="spring_context_dialogsession.xml"/>
    </bean>
    ...
    ...
</beans>
```

In the definition of “DialogSessionApplicationContext” you configure the corresponding context-instances, in case of using the XML based dialog-session-context this is the property “configLocation”.

In the example the configLocation “spring_context_dialogsession.xml” is used – so this is the Spring-XML-definition one for the dialog-session-context, which now finally holds the beans that are resolved by the Dispatcher (you remember the “#{d.DemoUI.firstName}”...?).

```
File: <project>/<Java-resources>/spring_context_dialogsession.xml

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

    ...
    <bean id="DemoUI" class="test.DemoUI" scope="prototype">
        <property name="..." ref="..."/>
    </bean>
    ...

</beans>
```

Your Dispatcher-implementation

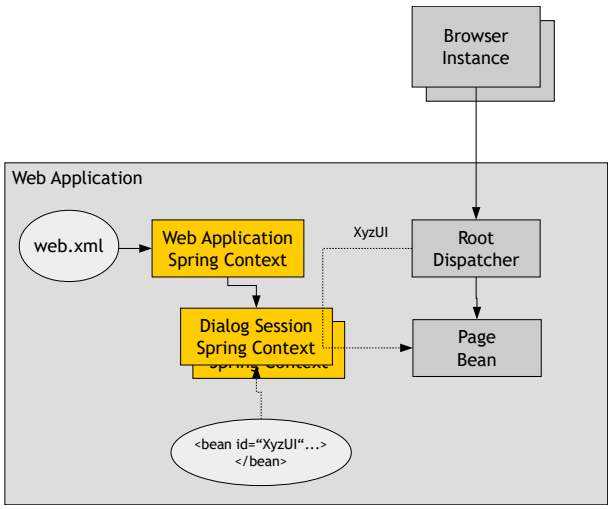
Each project comes with an own Dispatcher-class – within a new project it is located in package “managedbeans”. This dispatcher needs to extend the “DispatcherBySpringAccess”:

```
package managedbeans;

import org.eclnt.ccee.spring.context.DispatcherBySpringAccess;
import org.eclnt.ccee.spring.context.SpringDispatcherInfo;
import org.eclnt.workplace.IWorkpageContainer;

public class Dispatcher extends DispatcherBySpringAccess
{
    public static DispatcherInfo getStaticDispatcherInfo()
    {
        return new  SpringDispatcherInfo(Dispatcher.class,
                                         "spring_context_dialogsession.xml");
    }

    public Dispatcher()
    {
    }

    public Dispatcher(IWorkpageContainer workpageContainer)
    {
        super(workpageContainer);
    }
}
```

By providing the optional method “getStaticDispatcherInfo” this class delivers information to the CaptainCasa tools, so that the beans that are defined in the dialog-session-context are directly visible within the classes of the bean-browser-tool.

Summary

The total view on the scenario is:



- The wep-application-context of Spring is created and defined inside the web.xml
- The dispatcher creates one dialog-session-context per dialog session. The definition of this context is part of the web-application-context.
- Names of the dispatcher are resolved using the dialog-session-context.

Other issues

Accessing the dialog-session-context at runtime

You can access the dialog-sesion-context at runtime by using the API:

```
DialogSessionApplicationContextFactory.instance()
DialogSessionApplicationContextFactory.instance(ISessionAbstraction dialogSession)
```

Both methods return a Spring-AbstractApplicationContext-instance.

Hot Deployment

The implementations within eclnt_ccee_spring.jar are aware of dealing with hot deployment within the CaptainCasa toolset. Hot deployment splits up the application processing into two classloaders: the normal web-application classloader on <webapp>/WEB-INF/lib-level and a child-classloader on <webapp>/eclnthotdeploy/classes level.

The web-application part of the Spring integration is running within the web-application classloader because this is the runtime starting point of Spring.

The dialog-session part of the Spring integration is managed by CaptainCasa, so here the hot deployment is taken into consideration.

This means: when using hot deployment (which we always recommend!), then...

- The configuration of the web-application-context (in the example: “spring_context_webapplication.xml”) must NOT be hot deployed. Changes to this file require a reload of the whole web-application.
- The configuration of the dialog-session-context (in the example: “spring_context_dialogsession.xml”) shoud be hot deployed. Changes to this file then only require a quick hot deployment.