Spring with Adobe AEM/CQ5

Overview

The Spring Framework is arguably the largest, most popular java framework available. The reasoning behind this is that the Spring framework provides a wide range of open source tools to ease all aspects of development. To enable Spring inside an OSGi container, SpringSource has partnered with the Eclipse Gemini project to establish a common blueprint for implementation.

Environment Setup

Since CQ5 uses Apache Felix as its OSGi container you can install the required jars directly into the container. For Spring to work you need the following jars:
com.springsource.org.aopalliance
gemini-blueprint-core
gemini-blueprint-extender
gemini-blueprint-io
org.springframework.aop
org.springframework.asm
org.springframework.beans
org.springframework.context
org.springframework.core
org.springframework.expression

You can download the most recent version of these jars from Spring's enterprise bundle repository: http://ebr.springsource.com/repository/app/

Project Configuration

To enable your application to use Spring you will need an applcationContext.xml file just like with a standard Spring application. The only difference with OSGi is that you need to include the xml file in the META-INF/spring folder of your bundle. When you bundle is installed the Spring will register the beans and services defined in the configuration file.

OSGi Services

A key feature inside OSGi is the ability to register services within a bundle. A registered service can easily be accessed inside the OSGi container using the bundle API or by annotations inside your application code. This allows easily sharing domain specific services and for consuming services exposed by CQ5.

Registering a Service

In the configuration snippet below you will see a Spring bean bookService defined to the class BookServiceImpl. The class is an implementation of the BookService interface. Below the bean definition is an OSGi service definition that references the bookService bean and maps to the BookService interface.

```
<!-- Sample Spring Bean config -->
<bean id="bookService" class="com.perficient.cq.examples.service.impl.BookServiceImpl">
        <property name="bookDao" ref="bookDao"/>
</bean>

<!-- Registered OSGi Service -->
<osgi:service id="bookServiceOsgi" ref="bookService" interface="com.perficient.cq.examples.service.BookService"/>
```

The above configuration is all that is needed to make your Spring beans referenceable inside of your OSGi bundle code and accessible as an OSGi service.

Referencing a Service

There are multiple OSGi services exposed within the standard installation of CQ5. To consume a service inside a Spring bean you will need to reference the the service inside your Spring configuration file. Once it is referenced in spring, you can then inject the service into your Spring beans.

```
<!-- Referencing an OSGi Service -->
<osgi:reference id="messageService" interface="com.xyz.MessageService"/>

<!-- Sample Spring Bean config using the OSGi service -->
<bean id="bookService" class="com.perficient.cq.examples.service.impl.BookServiceImpl">
        <property name="bookDao" ref="bookDao"/>
        <property name="messageService" ref="messageService"/>
</bean>
```

Note on Annotations

The examples provided above are all using standard XML configuration but annotations could have been used for the Spring bean examples. I have not seen annotations that can replace the OSGi specific configuration but you could have easily autowired in the messageService or bookDao and configured bookService via annotations.