

OSGi and Spring: Part 2: Build and deploy OSGi as Spring bundles using Felix

A step-by-step guide for the developers to build Java components and package it as OSGi based Spring bundles using Apache Felix, an open source OSGi container

Skill Level: Intermediate

[Rajeev J. Hathi \(rajeevhathi@gmail.com\)](mailto:rajeevhathi@gmail.com)
Senior Software Consultant

[Naveen Balani \(naveenbalani@rediffmail.com\)](mailto:naveenbalani@rediffmail.com)
Enterprise Architect
IBM India Software Labs (ISL)

31 Mar 2009

Build and package Java classes as OSGi bundles using the Spring DM framework in a Felix container. This article, Part 2 of this series, shows you how to create bundles using the Spring framework and then deploy them in a Felix runtime environment. You will see how the core OSGi framework dependency is removed through a simple Spring-based configuration.

Introduction

In this article, you will revisit the order application developed in part 1 of the series. The application will now use Spring DM to build and package the bundles. The application client will invoke the service component to process the order and the server component will print the order ID. The article will help you understand the concept of Spring DM and its use with the Felix-based OSGi container.

System requirements

To run the examples in this article, make sure the following software is installed and set up on your machine:

- Java 5 or higher
- Ant build tool
- Felix binary distribution 1.0.4
- Spring DM bundles

After the above distribution is installed, set up the following environment variables: (by way of example set `ANT_HOME=C:\apache-ant-1.7.0`).

- `JAVA_HOME` (for Java)
- `ANT_HOME` (for Ant)

Next add the following to the `PATH` environment variable:

- `JAVA_HOME\bin`
- `ANT_HOME\bin`

Spring DM

Spring DM includes JARs or bundles that help deploy Spring applications in an OSGi environment. Spring DM-based applications can make use of services offered by the OSGi environment. These types of applications provide ease and convenience in the development of OSGi-based applications. Spring DM offers the following benefits in the OSGi environment:

- Modularizes the application into dynamic bundles and provide runtime services to these bundles.
- Provides versioning capability to the modules.
- Modules are bundled as services that can be dynamically discovered and consumed.
- Modules can be installed, updated and uninstalled dynamically
- Takes advantage of the Spring framework in configuring the application as OSGi modules
- Provides separation of business logic and configuration thereby making development easy and convenient

Order Application Revisited

We will revisit the order application developed in part 1 of this series. As you can see, the classes, currently, are strongly coupled with the OSGi framework. Now you will remove this strong coupling and make the classes as simple POJOs with the use of Spring DM. Let's look at the revised OrderClient.java, below.

Listing 1. The client component OrderClient

```
package order.client;

import order.OrderService;

public class OrderClient {

    private OrderService orderService;

    public void setOrderService(OrderService orderService) {
        this.orderService = orderService;
    }

    public void removeService() {
        this.orderService = null;
    }

    public void start() {
        orderService.processOrder();
    }

    public void stop() {
        System.out.println("Bundle stopped");
    }
}
```

The OSGi framework dependency is completely removed. The class is a plain POJO with the start() method simply processing the order. There is no use of ServiceTracker class.

Listing 2. OrderService implementation

```
package order.impl;

import order.OrderService;

public class OrderServiceImpl implements OrderService {

    public void start() {
        System.out.println("Order Service registered");
    }

    public void stop() {
        System.out.println("Order Service stopped");
    }

    public void processOrder() {
        System.out.println("Order id: ORD123") ;
    }
}
```

```
}
```

Similarly, the above `OrderServiceImpl` too a simple POJO with the `processOrder()` method. There is no association with the OSGi core components. Also neither client nor service classes implement the `BundleActivator`. The bundle lifecycle here is managed by Spring DM.

So, how do these simple POJOs work as OSGi components? The magic lies in the Spring configuration files. It is here where you define the OSGi part.

But before you define XML files, there are some changes to be made to the manifest files. The manifest will include import packages that will contain Spring DM files. This will ensure that Spring will now take over the management and lifecycle of OSGi bundles. The code snippets below illustrates the service and client manifest files.

Listing 3. Service Manifest

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Order Service
Bundle-SymbolicName: orderservice
Bundle-Version: 1.0.0
Import-Package: org.springframework.beans.factory.xml,
org.springframework.aop, org.springframework.aop.framework,
org.aopalliance.aop, org.xml.sax, org.osgi.framework,
org.springframework.osgi.service.importer.support,
org.springframework.beans.propertyeditors,
org.springframework.osgi.service.exporter.support,
org.springframework.osgi.service.exporter
Export-Package: order
```

Listing 4. Client Manifest file

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Order Service Client
Bundle-SymbolicName: orderclient
Bundle-Version: 1.0.0
Import-Package: org.springframework.beans.factory.xml,
org.springframework.aop, org.springframework.aop.framework,
org.aopalliance.aop, org.xml.sax, org.osgi.framework,
org.springframework.osgi.service.importer.support,
org.springframework.beans.propertyeditors,
org.springframework.osgi.service.importer,
org.springframework.osgi.service.exporter.support,
order
```

You will create two XML files each for service and client components. For the service, the `orderservice.xml` XML file will define the order service implementation bean and the `orderservice-osi.xml` XML file will define the order service interface and refer to its implementation. Similarly for the client, XML files will define the order client bean and refer to order service components. The service and client XML files

reside under their respective META-INF/spring folder. The below code shows the service XML file

Listing 5. Service XML file orderservice.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
  http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean name="orderService" class="order.impl.OrderServiceImpl"/>

</beans>
```

Listing 6. Service OSGi XML file orderservice.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
  http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean name="orderService" class="order.impl.OrderServiceImpl"/>

</beans>
```

You could actually provide both the bean and OSGi definition in one XML file. It is not necessary to create two separate files. Here we are making two separate files so that we distinguish between the interface definition and configuration. Managing and maintaining these files becomes easier. The beauty of a Spring-based configuration is that it will allow you to test your order service bean outside the OSGi container.

To be able to actually do this, you will have to install the relevant Spring DM jars in Felix.

Listing 7. Portion of Felix configuration

```
felix.auto.start.1= \
file:/felix-1.0.4/bundle/org.apache.felix.shell-1.0.1.jar \
file:/felix-1.0.4/bundle/org.apache.felix.shell.tui-1.0.1.jar \
file:/felix-1.0.4/bundle/org.apache.felix.bundlerepository-1.0.3.jar \
file:/osgi_spring/order/springlib/aopalliance.osgi-1.0-SNAPSHOT.jar \
file:/osgi_spring/order/springlib/jcl104-over-slf4j-1.4.3.jar \
file:/osgi_spring/order/springlib/log4j.osgi-1.2.15-SNAPSHOT.jar \
file:/osgi_spring/order/springlib/org.apache.felix.main-1.0.1.jar \
file:/osgi_spring/order/springlib/slf4j-api-1.4.3.jar \
file:/osgi_spring/order/springlib/slf4j-log4j12-1.4.3.jar \
file:/osgi_spring/order/springlib/spring-aop-2.5.1.jar \
file:/osgi_spring/order/springlib/spring-beans-2.5.1.jar \
file:/osgi_spring/order/springlib/spring-context-2.5.1.jar \
file:/osgi_spring/order/springlib/spring-core-2.5.1.jar \
file:/osgi_spring/order/springlib/spring-osgi-core-1.0.2.jar \
file:/osgi_spring/order/springlib/spring-osgi-extender-1.0.2.jar \
file:/osgi_spring/order/springlib/spring-osgi-io-1.0.2.jar
```

As you can see from what we've demonstrated above, the Felix configuration file will have the Spring DM bundles defined. You can download the Spring DM bundle and put in the folder of your choice. Once the jars are put in the appropriate folder, make the relevant entries in the Felix config file specifying these jars to be installed at the Felix startup.

You can then proceed with the steps of installing the client and service bundles in the Felix runtime environment. Once installed, you should be able to start and stop the bundles and see the same results. You will notice that unlike a Felix-based OSGi, this time it is Spring DM that is managing the bundle lifecycle.

Conclusion

The power of the Spring framework has made development of OSGi applications simpler and more effective. OSGi itself also has revolutionized the way Java applications are bundled. OSGi, in association with Spring, has provided a solid foundation to the development of enterprise-wide applications.

Downloads

Description	Name	Size	Download method
OSGi Spring Order Application	osgispring_orderapp.zip	11KB	HTTP

[Information about download methods](#)

Resources

- [Participate in the discussion forum for this content.](#)
- [Design and develop RESTful Web service using Apache CXF](#) (developerWorks, Jul 2008) is a step-by-step guide for developers to create a RESTful Web service using CXF, an open source Web service framework.
- [Design and develop Web service using Apache CXF](#) (developerWorks, Jul 2008) is a step-by-step guide to develop Web service using CXF and Spring.
- [Design and develop JAX-WS 2.0 Web services](#) (developerWorks, Sep 2007) is a step-by-step guide to develop Web service using JAX-WS technology.
- Browse the [technology bookstore](#) for books on these and other technical topics.

About the authors

Rajeev J. Hathi

Rajeev Hathi works as a software consultant for the J2EE platform. His interests are architecting and designing J2EE-based applications. Rajeev attained SUN certifications in Java and J2EE technologies (Web, EJB, Architect). He has contributed to IBM developerWorks by writing articles about Ajax, Web services, DB2, and XML. In addition, he was involved in writing a book about applying Java 6 and Hands to Web services. He is based in Mumbai, India. His hobbies are watching sports and listening to rock music.

Naveen Balani

Naveen Balani works as an architect with IBM India Software Labs (ISL). He leads the design and development activities for WebSphere Business Services Fabric out of ISL. He likes to research new technologies and is a regular contributor to IBM developerWorks, having written about such topics as Web services, ESB, JMS, SOA, architectures, open source frameworks, semantic Web, J2ME, pervasive computing, Spring, Ajax, and various IBM products. He's also a coauthor of Beginning Spring Framework 2 and Getting Started with IBM WebSphere Business Services Fabric V6.1.

