tutorial-osgi-camel-part2a

Transform existing projects into bundles

To convert a project into a bundle a couple of things/steps must be done :

1) Declare the project in the maven pom.xml file as of type <packaging>bundle</packaging>,
2) Add the maven felix plugin who will generate the MANIFEST.MF file,
3) Identify the packages to be imported/exported and version,
4) Identify and register OSGI services when they are used by another bundle.

Spring team has created the Spring Dynamic Modules project to facilitate the work of the developer working with OSGI specification. It allows to transform existing spring beans (interface + implemented class) into OSGI services reachable by any bundle deployed. In fact the services are declared in a repository under the form of interfaces. This mechanism is known now as blueprint services or RFC 124 and is currently integrated under the OSGI specification R4.2.

Step 1 : reportincident.model

To transform the reportincident.model project, we will execute the steps 1) 2) and 3) because no services must be registered for this project. So, update the pom.xml file created and add/change what is put in the code below in XML comment with word STEP

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>org.apache.camel.example</groupId>
        <artifactId>reportincident.model</artifactId>

        <!-- STEP 1 -->
        <packaging>bundle</packaging>

        <name>Report Incident Model Bundle</name>
        <version>1.0-SNAPSHOT</version>

        <parent>
            <groupId>org.apache.camel.example</groupId>
            <artifactId>reportincident.parent</artifactId>
            <version>1.0-SNAPSHOT</version>
        </parent>

        <dependencies>
                <!-- Camel bindy  -->
                <dependency>
                        <groupId>org.apache.camel</groupId>
                        <artifactId>camel-bindy</artifactId>
                        <version>${camel-version}</version>
                </dependency>
                <dependency>
                        <groupId>commons-lang</groupId>
                        <artifactId>commons-lang</artifactId>
                        <version>${commons-lang}</version>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <!-- to compile with 1.5 -->
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <configuration>
                                        <source>1.5</source>
                                        <target>1.5</target>
                                </configuration>
                        </plugin>

                        <!-- STEP 2 -->
```

```
                        <!-- to generate the MANIFEST-FILE of the bundle -->
                        <plugin>
                                <groupId>org.apache.felix</groupId>
                                <artifactId>maven-bundle-plugin</artifactId>
                                <extensions>true</extensions>
                                <version>${felix-version}</version>
                                <configuration>
                                        <manifestLocation>META-INF</manifestLocation>
                                        <instructions>
                                                <Bundle-SymbolicName>${pom.artifactId}</Bundle-SymbolicName>

                                                <!-- STEP 3 -->
                                                <Export-Package>
                                                    '=META-INF.org.apache.camel.example.reportincident.model',
                                                    org.apache.camel.example.reportincident.model
                                                </Export-Package>

                                                <_failok>true</_failok>
                                        </instructions>
                                </configuration>
                        </plugin>

                </plugins>
        </build>
</project>
```

Remarks :

- The model bundle does not need to import 'specific' packages so the tag <Import-Packages> can be skipped but packages will be added automatically
  by the felix/bnd tool to the MANIFEST.MF file if import statements are defined in the classes or maven dependencies are declared
- The tag <_failok>true</_failok> is added to inform the plugin that it can continue the creation of the MANIFEST.MF file even if an error/warning occurs. This is the case here because files located under META-INF directory must be exported.
- The hibernate file must be exported with the classpath of the bundle. This is why the line '=META-INF.org.apache.camel.example.reportincident.model' has
  been added to export the packagename containing this file

When the pom.xml has been updated, you can execute the following maven command :

```
mvn clean install org.ops4j:maven-pax-plugin:eclipse
```

the goal org.ops4j:maven-pax-plugin:eclipse is added to refresh the MANIFEST.MF file created

This command will generate a jar file containing the classes, hibernate file and MANIFEST.MF file and deploy it in your maven local repository. You can open the jar and check if the the content of MANIFEST.MF is similar to

```
Manifest-Version: 1.0
Export-Package: org.apache.camel.example.reportincident.model;uses:="o
 rg.apache.commons.lang.builder,org.apache.camel.dataformat.bindy.anno
 tation",META-INF.org.apache.camel.example.reportincident.model
Built-By: Charlesm
Build-Jdk: 1.6.0_12
Bundle-Version: 1.0.0.SNAPSHOT
Tool: Bnd-0.0.255
Bundle-Name: Report Incident Model Bundle
Bnd-LastModified: 1240844069542
Created-By: Apache Maven Bundle Plugin
Bundle-ManifestVersion: 2
Bundle-SymbolicName: reportincident.model
Import-Package: META-INF.org.apache.camel.example.reportincident.model
 ,org.apache.camel.dataformat.bindy.annotation;version="2.0.0.M1",org.
 apache.camel.example.reportincident.model,org.apache.commons.lang.bui
 lder;version="2.4"
```

It is time to continue with the persistence project where we will introduce new important concepts

Step 2 : reportincident.persistence

First, you have to replace the pom.xml file created with the file provided in the project attached (see resource). If you open this file, you will see that the <Import-Package> section of the maven-felix-plugin has been enriched with the packages required to work with Hibernate, Spring and JTA classes.

Nevertheless, it is interesting to mention that we have exported the package `org.apache.camel.example.reportincident.dao` and defined `org.apache.camel.example.reportincident.dao.impl` as private. Why, the reason is very simple, we would like to export only the interface to another 'service' bundles and keep internally the implementation.

```
<Private-Package>org.apache.camel.example.reportincident.dao.impl</Private-Package>
    <Export-Package>org.apache.camel.example.reportincident.dao</Export-Package>
```

Discovering all the classes used by a third party library like Hibernate can be cumbersome and takes time. An interesting alternative is to use the command 'DynamicImport-Package' to resolve classloading issue. <DynamicImport-Package> *</DynamicImport-Package>

In order to test the tip, update your pom.xml with the following info :

```
<Import-Package>
        META-INF.org.apache.camel.example.reportincident.model,
        com.mysql.jdbc,
        org.apache.camel.example.reportincident.model,
        org.apache.commons.dbcp,
        *
    </Import-Package>
    <Private-Package>
        org.apache.camel.example.reportincident.dao.impl
    </Private-Package>
    <Export-Package>
        org.apache.camel.example.reportincident.dao
    </Export-Package>
    <DynamicImport-Package>*</DynamicImport-Package>
```

and look at the result generated by the plugin. The Import-Package section is dry.

Now that our pom.xml is configured we will modified our spring.xml files to allow our DAO service to be registered as a OSGI service. Why, because the classes of the bundle reportincident.service uses this DAO class but required also additional functionalities like (Hibernate SessionFactory, Spring Transaction management, ...) who will be instantiated and configured when the persistence bundle/service will be started.

Additional motivations are also provided in the OSGI specification :

> Enterprise application developers working with technologies such as those described in chapter 2 would like to be able to take advantage of the OSGi platform. The core features of enterprise programming models previously described must be retained for enterprise applications deployed in OSGi. The current OSGi specifications are lacking in the following areas with respect to this requirement:
> • There is no defined component model for the internal content of a bundle. Declarative Services only supports the declaration of components that are publicly exposed.
> • The configuration (property injection) and assembly (collaborator injection) support is very basic compared to the functionality offered by frameworks such as Spring.
> • There is no model for declarative specification of services that cut across several components (aspects or aspect-like functionality)
> • Components that interact with the OSGi runtime frequently need to depend on OSGi APIs, meaning that unit testing outside of an OSGi runtime is problematic
> • The set of types and resources visible from the context class loader is unspecified. The context class loader is heavily used in enterprise application libraries
> • Better tolerance of the dynamic aspects of the OSGi platform is required. The programming model should make it easy to deal with services that may come and go, and with collections of such services, via simple abstractions such as an injecting a constant reference to an object implementing a service interface, or to a managed collection of such objects. See the description of osgi:reference in the Spring Dynamic Modules specification for an example of the level of support required here.
> Providing these capabilities on the OSGi platform will facilitate the adoption of OSGi as a deployment platform for enterprise applications. This should be done in a manner that is familiar to enterprise Java developers, taking into account the unique requirements of the OSGi platform. The benefits also extend to other (non-enterprise) OSGi applications that will gain the ability to write simpler, more testable bundles backed by a strong component model.

Create the file `persistence-osgi.xml` in the directory `src/main/resources/META-INF/spring` and add the lines :

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

    <osgi:service ref="incidentDAO" interface="org.apache.camel.example.reportincident.dao.IncidentDAO"/>

</beans>
```

The osgi:service namespace tells to Spring to register the interface `org.apache.camel.example.reportincident.dao.IncidentDAO` as a service in the OSGI registry. This feature proposed by Spring will be part of the next OSGI specification R4.2 under the name of Blueprint - RFC 124.

Remark :

- Notice also that to use this osgi spring namespace, we have imported a new schema `http://www.springframework.org/schema/osgi/spring-osgi.xsd` in the persistence-osgi.xml file
- More than one interface can be part of an OSGI service.

Another feature that I would like to introduce here concerns the Configuration Admin. In its simplest form, the CM can be seen as a configuration source, namely a Dictionary whose keys are always Strings. Spring DM can expose entries in the CM as a Properties object, through the cm-properties element.

So, we can adapt our `spring-datasource-beans.xml` file created in the previous chapter with new xml tags :

```
    <context:property-placeholder properties-ref="preProps" />         (1)
    ...
    <osgix:cm-properties id="preProps" persistent-id="org.apache.camel.example.reportincident.datasource"> (2)
        <prop key="driverClassName">com.mysql.jdbc.Driver</prop>
        <prop key="url">jdbc:mysql:///report</prop>
        <prop key="username"></prop>
        <prop key="password"></prop>
    </osgix:cm-properties>
    ...
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
        <property name="driverClassName" value="${driverClassName}" /> (3)
        <property name="url" value="${url}" />
        <property name="username" value="${username}" />
        <property name="password" value="${password}" />
    </bean>
```

The configuration above, exposes the properties available in the CM under `org.apache.camel.example.reportincident.datasource` entry (2) as a bean named preProps. Moreover, the property declared (3) can be override by creating a file named `org.apache.camel.example.reportincident.datasource.cfg` and containing the parameters :

```
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql:///report
username=root
password=
```

Spring using the `context:property-placeholder` (1) will be able to load it.

Remarks :

- We will see in the chapter 'deployment' where this file must be deployed.
- In our example, we have only defined `properties` for the `driver`, `username` and {{password }} but you can extend the list of values with by example Hibernate parameters like hibernate.show_sql, hibernate.format_sql, ...

Step 3 : reportincident.service

Like for the project reportincident.persistence, we will replace our pom.xml file with the one provided in the zip file. As you can see in the <Import-Package>, we will import here the class required by the service : org.apache.camel.example.reportincident.dao

Adding this line in the Import-Package is not enough to have access to the OSGI service. The file `spring-service-beans-dao.xml` must be modified to have a reference to this interface through the osgi:reference namespace :

```
...
    <property name="incidentDAO">
        <osgi:reference interface="org.apache.camel.example.reportincident.dao.IncidentDAO"/>
    </property>
...
```

This reference will be used to call the osgi service to find the service corresponding to the interface name declared. If a match occurs, then a spring bean reference is created in the bundle reportincident.service.

To expose our service as an OSGI service, we will create the file `service-osgi.xml` in the directory `src/main/resources/META-INF/spring` and add the code.

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

  <osgi:service ref="incidentService" interface="org.apache.camel.example.reportincident.service.
IncidentService"/>

</beans>
```

Remark :

- Transaction Management has been defined in the corresponding files of reportincident.persistence and reportincident.service but we will not discuss them in detail in this tutorial.

Step 4 : reportincident.webservice

This bundle will not be exported as an OSGI service. So, we only need to modify the content of <Export-Package> to export the classes generated by the wsl2java maven plugin and the wsdl file :

```
    <Export-Package>
        org.apache.camel.example.reportincident,
             '=META-INF.wsdl'
    </Export-Package>
```

Conclusion

Now that we have transformed our current project in bundles, it is time to design the routing and web parts of the application. In the next part of the tutorial, we will specify modification to do for the new incoming projects/bundles

Links

- Part 2 : real example, architecture, project setup, database creation
- Part 2a : transform projects in bundles
- Part 2b : add infrastructure and routing
- Part 2c : web and deployment

#Resources

- File Modified

No files shared here yet.