OSGi - The dynamic Module System for Java : Part 1

Level: Novice

Aneesh Kumar KB(aneesh_kb@cdactvm.in), E-Governance, CDAC

This tutorial enables to understand the basic concepts of OSGi(Open Service Gateway Initiative), the dynamic module system for java and start with OSGi bundle development.

Before you start

About this series

This is a two part tutorial in which the first part covers the basic OSGi technology, architecture and a Hello world OSGi bundle development. The second part of this tutorial covers the Service Oriented Architecture (SOA) in OSGi and Service bundle development .

Objectives

In this tutorial you will

1. learn the OSGi technology and its architecture
2. familiarize with famous OSGi implementations.
3. create a Hello world OSGi bundle.

Prerequisites

You should have a basic knowledge of java programming.

System Requirements

You should have installed and configured the Java Development Kit.

Introduction

Java is famous for its platform Independence but it is not supporting modular components that can be freely plugged in and plugged out of the application. Think of a java class that is part of a java application meant for representing a real world object and doing a functionality associated with that object. Then this class should be part of the application jar file. If you find some serious security bug in the particular class you will have to stop the entire application and then replace the buggy class with the new one. Til the time you correct the bug you have to stop it entirely even though the buggy class is used only for a sub functionality that is rarely used. Here comes the importance of OSGi technology in which the functionalities are independently developed and plugged into an application framework as a service. These services can be dynamically discovered and bind with other parts of the application.Since the other part of the application binds the service dynamically the particular service can be stopped or updated without affecting the other part of the application. Now some of you may be wondering the same modular approach is available in java in the form of EJB and servlet. But they are server side components and their services are available only under the ubrella of an application server. The OSGi specification comes up with a solution for this.

The OSGi, the Open Service Gateway Initiative is a consortium Founded by Ericson, IBM, Oracle and Sun. It is also known as Dynamic Module System for Java. This architecture-specification defines a platform for running modular applications. This platform acts as a container that provides some common services such as life cycle management and security for example. OSGi is basically a framework for containing functional componets. In the OSGi terminology this component is called bundle. The OSGi specification defines the standard that clearly specifies how a bundle should be developed so as to plug into the OSGi platform. Hence these Bundles can be developed independently as per the specification using the standard APIs and can be deployed in the OSGi platform. It is similar to developing a bolt for specific nut. If the hole diameter is known the bolt can be developed independently and later it can be fitted together. These bundles can be started, stopped or updated independently without affecting other components or with out restarting the entire platform.

The OSGi alliance specification like EJB and servlet specification, defines two things.

1. A set of services an OSGi container must implement

2. A contract between the container and the application

The latest OSGi specification is OSGi Service Platform release 4.1. There are number of open source OSGi container implementations such as

- Equinox
- Knoplerfish
- Apache Felix

These implementations allow application developers to break applications into multiple modules and hence the inter dependencies among the application modules can be easily managed. Among the implementation the term Equinox may be familiar to Java developers since the famous IDE eclipse is based on this OSGi container implementation.

Because of its modular development approach it finds wide usage in the area of Enterprise,Mobile/ PDA , Automotive, Smart Home and E-Health application development.

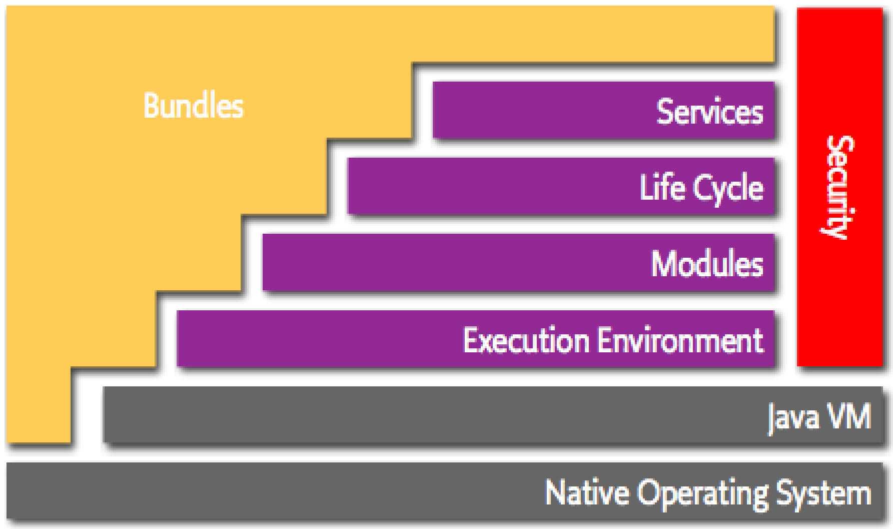
The Advantage of using OSGi

- Reduced complexity: Developing OSGi means developing modules ie bundles. They hide their internals from other bundles and the communication is through well defined interfaces.Hence the later changes can be easily accommodated without affecting other modules.
- Reuse : The OSGi component model makes the integration of third party components very easy.

- Easy deployment- The OSGi specification clearly specifies how the components are installed and managed. It provides APIs so that it can be integrated with external management systems.
- Dynamic updates: Bundles can be installed, started, stopped, updated and uninstalled without bringing down the whole system.
- Adaptive: The OSGi provides a dynamic service registry where bundles can register, get and listen to services.This dynamic service model allows bundle to find out what all services available in the system and can adapt those functionalities.
- Transparency: Certain parts of applications can be shutdown for bug fixing.

The OSGi Architecture

The OSGi has a layered model



courtesy: www.osgi.org

- Bundles : Bundles are the OSGi components made by the developers.This bundle can make use of services provided by different layers of OSGi architecture such as security, service binding, life cycle management etc.
- Services : The services layer connects bundle in a dynamic way by offering a publish-find-bind model for plain old java objects.This layer provides service through which the services bundles can be registered.
- Life-cycle : This layer provides services for starting, stopping, updating, installing and uninstalling bundles in the framework.This layer contains API to install, start, stop, update and uninstall bundles.
- Module : This layer provides the basis class loading functionality. The layer also defines how a bundle can import and export code.
- Security : The layer that handles the security aspects
- Execution Environment : Defines what methods and classes are available in a specific platform.

Getting Started with OSGi Development

Step 1: Download an OSGi implementation. For the demonstration I am using Equinox framework. For other frameworks you have to use the jar file and framework provided by them for development and deployment. How to configure the framework and jar file can be found in their website provided above.

Step 2: Save the downloaded jar file to a location for example C:/lab

Step 3: Create a java file named 'HelloOSGI.java'

```
package test.osgi.helloworld;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
public class Activator implements BundleActivator {

   public void start(BundleContext context) throws Exception {
      System.out.println("Hello World");
   }


   public void stop(BundleContext context) throws Exception {
      System.out.println("Bye Bye");
   }

}
```

This java file is just like a Main class file in java. The start() method is the starting point of the bundle. This is called by the OSGi container.

Step 4: Set your classpath to the downloaded jar file and compile the java file

```
# set classpath=%classpath%;org.eclipse.osgi_3.4.0.v20080605-1900.jar

# javac -d . HelloOSGi.java
```

The above commands are based on the assumption that the working directory is C:\lab and the equinox jar file is located in the same directory.
When you run the above command you will get the class file inside the package-directory hierarchy.

Step 5 : create a Manifest file named MANIFEST.MF with the following content

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: HelloWorld OSGi
Bundle-SymbolicName: test.osgi.HelloWorld
Bundle-Version: 1.0.0
```

```
Bundle-Activator: test.osgi.helloworld.HelloOSGI
Import-Package: org.osgi.framework
```

Bundle-ManifestVersion:The `Bundle-ManifestVersion` header tells the OSGi container that this bundle follows the rules of the OSGi specification. A value of *2* means that the bundle is compliant with OSGi specification Release 4; a value of *1* means that it is compliant with Release 3 or earlier.

Bundle-Name:The `Bundle-Name` header defines a simple name for the bundle.

Bundle-SymbolicName: The `Bundle-SymbolicName` header specifies a unique, non-localizable name for the bundle. This is the name you will use while referring a given bundle from other bundles.

Bundle-Version:The `Bundle-Version` header specifies the version of the bundle.

Bundle-Activator:The `Bundle-Activator` header specifies the name of the optional listener class to be notified of bundle *start* and *stop* events.

Import-Package:The `Import-Package` header defines imported packages for the bundle.

Step 6 : Create the bundle-jar file

```
# jar -cvmf MANIFEST.MF HelloOSGi.jar -C test/*
```

jar command will create HelloOSGi.jar file with the supplied manifest file.

Step 7 : Start the container

```
#java -jar  org.eclipse.osgi_3.4.0.v20080605-1900.jar -console
```

It will start the container in a console with a prompt osgi>

Step 8 : Install the bundle

```
osgi> install file:/C:/lab/HelloOSGi.jar
```

Stet 9: Start the bundle

```
osgi> start file:/C:/lab/HelloOSGi.jar
```

It will print 'Hello World' in the console

Second part of this article covers OSGi Service Bundle development, Using Services offered by other bundles and Service Factory

Command Reference

install	- install and optionally start bundle from the given URL
uninstall	- uninstall the specified bundle(s)
start	- start the specified bundle(s)
stop	- stop the specified bundle(s)

Reference:

http://www.osgi.org
http://www.eclipse.org/equinox
http://felix.apache.org