OSGi

(Open Services Gateway initiative)

گرد آورنده:

فاطمه داوری عدالت پناه

زیر نظر:

مهندس امیر سام بهادر

زمستان ۹۰

پیشگفتار:

در این مقاله OSGi بعنوان یک ابزار پیشرو برای ایجاد برنامه های Service Oriented تحت جاوا معرفی می شود. برای آشنایی هر چه بیشتر خوانندگان محترم با OSGi پنج نمونه کد برنامه، از یک برنامه ساده Hello World تا برنامه های پیچیده تر که قادر به Track کردن سرویس ها در OSGi هستند، ارائه شده است.

این نمونه کدها لزوما همان کدهای ارائه شده در مقالات یافت شده در اینترنت که جهت نگارش این مقاله ترجمه شده اند، نیستند. در واقع بدلیل اینکه راهنماییهای ارائه شده در این مقاله بر مبنای استفاده از Intellij بعنوان IDE، بدون استفاده از هیچگونه Wizardی (بجهت درک عمیق OSGi) می باشد، ابتدا از کدهای ارائه شده در مقالات دیگر جهت تولید برنامه ها در محیط Intellij استفاده شد. برخی از این کدها پس از پیاده سازی درست عمل ننموده و برخی نیز دارای exception بودند. پس از رفع مشکلات پیش آمده، کامپایل و اجرای موفقیت آمیز، کدهای تمیز و بدون exception در این مقاله بکار رفتند تا خوانندگان محترم با استفاده از این کدها ۱۰۰٪ با موفقیت بتوانند برنامه ها را پیاده سازی کنند. اما در هر صورت اگر خوانندگان محترم مقاله بهر دلیلی در پیاده سازی موفق نبودند میتوانند جهت دریافت package پیاده سازی شده این برنامه ها با آدرس ایمیل fatemedavari@gmail.com مکاتبه نمایند.

نکته قابل توجه در این مقاله این است که در آن هیچ استفاده ای از eclipse که دارای OSGi plug-in است، نشده است. شایان ذکر است با وجود اینکه در این مقاله مکرراً از Intellij به عنوان IDE برای پیاده سازی برنامه ها نام برده شده است، اما واقع بر این است که این برنامه ها بدون استفاده از IDE و با تایپ sourceکدها در notepad نیز قابل پیاده سازی است.

دلیل این امر این است که از Ant برای اجرای آنها استفاده شده و همانطور که می دانید Ant برنامه نویس را از داشتن IDE بی نیاز می کند . در جستجوهای انجام شده در اینترنت نیز مقاله فارسی OSGi مبتنی بر Ant مشاهده نشد و بنظر میرسد که این مقاله اولین آن باشد.

در پایان از استاد ارجمند جناب آقای مهندس امیر سام بهادر که دانش برنامه نویسی به زبان جاوا را از ایشان آموخته ام و از راهنمایی های ایشان جهت تدوین این مقاله بهره مند شده ام، سپاسگزاری می نمایم.

فاطمه داوری عدالت پناه

OSGi

(Open Services Gateway initiative)

OSGi بعنوان یک معماری برای develop و deploy کردن کتابخانه ها و برنامه های کاربردی ماژولار معرفی می شود. در این مقاله مفاهیم پایه OSGi ارائه شده و همچنین به شما نشان داده خواهد شد که چگونه یک برنامه ساده Hello World را در Intellij IDE نوشته و در بستر OSGi، Knopflerfish اجرا کنید. در پایان نیز طریقه ساختن سرویس های OSGi، ServiceListener و ServiceTracker به عنوان مبحث های تکمیلی ارائه خواهد شد.

همچنین OSGi به عنوان سیستم ماژول دینامیکی برای جاوا شناخته شده است. با استفاده از بسترهای پیاده سازی OSGi نظیر Knopflerfish، Equinox و Apache Felix قادر خواهید بود که application خود را به چندین ماژول بشکنید و با وجود وابستگی های پیچیده آنها، به سادگی آنها را مدیریت کنید. مشابه servlet و EJB، خصوصیت OSGi نیز دو چیز را تعریف میکند: گروهی از سرویس ها که بستر OSGi باید آنها را اجرا کند و رابط بین بستر و application .

Develop کردن یک برنامه بر روی سکوی OSGi (platform) بدین معناست که ابتدا application را با استفاده از OSGi API ها می سازیم و سپس آنرا در یک بستر OSGi، deploy می کنیم. از دیدگاه یک برنامه نویس، OSGi مزایای زیر را ارائه می دهد:

- ✓ شما می توانید ماژول های مختلف برنامه خود را بصورت دینامیک بدون اینکه نیازی به restart آن داشته باشید install، uninstall، start و stop نمایید.
- ✓ برنامه شما می تواند بیش از یک version از یک ماژول خاص را در یک لحظه در حال اجرا داشته باشد.
- ✓ OSGi زیر ساخت های خیلی خوبی را برای توسعه application های service-oriented فراهم می آورد، همان طور که برای برنامه های موبایل یا تحت وب این کار را انجام می دهد.

با توجه به اینکه شما بستر severlet را برای ساختن برنامه های تحت وب و بستر EJB را برای ساختن برنامه های تراکنشی استفاده می کنید، ممکن است از خودتان بپرسید که چرا هنوز نیاز به یک نوع بستر دیگر دارید؟

کوتاه ترین پاسخ این است که بسترهای OSGi بطور خاصی برای توسعه برنامه های پیچیده جاوا که تمایل دارید آنها را به ماژول ها تفکیک شده بشکنید استفاده می شود. در ادامه این پاسخ کوتاه را بسط و گسترش خواهیم داد.

OSGi در Enterprise applicationها

کار بر روی OSGi بوسیله OSGi Alliance در مارس ۱۹۹۹ شروع شد. هدف اصلی آن ایجاد یک بستر باز برای ارائه خدمات مدیریت شبکه ها و device های local بود. ایده اصلی OSGi این است که وقتی یک بار یک platform خدمات OSGi را به device شبکه شده خود اضافه می کنید، شما قادر باشید تا چرخه عمر اجزاء نرم افزار در آن device را از هر مکانی در شبکه

مدیریت کنید. اجزاء نرم افزار میتوانند نصب، به روز رسانی یا حذف شوند بدون اینکه اختلالی در عملکرد device ها بوجود آید. برای سال ها تکنولوژی OSGi در بازار device های شبکه ای و سیستم های embedded رونق یافته است و در حال حاضر OSGi به عنوان یک تکنولوژی ماندگار و ارزشمند در حال ظهور است.

پشتیبانی در حال رشد برای OSGi

در سال ۲۰۰۳، تیم توسعه eclipse شروع به جستجو برای راهکارهایی کرد که eclipse را دینامیک تر و همچنین غنی تر از لحاظ platform های clientی کند و ابزارهای ماژولاریتی را افزایش دهد. در نهایت آنها آن را با استفاده از framework OSGi به عنوان یک runtime component Model حل و فصل کردند.

در حال حاضر تقریبا تمام سرورهای کاربردی enterprise، یا OSGi را پشتیبانی می کنند و یا برنامه پشتیبانی از آن را دارند. Spring framework نیز OSGi را پشتیبانی می کند، این پشتیبانی از طریق ماژول های دینامیک spring برای platformهای خدمات OSGi که یک لایه زیرساخت را برای راحت تر کردن استفاده از OSGi در توسعه برنامه های enterprise جاوای spring-base فراهم می آورد، انجام می شود.

بسترهای کدباز OSGi

از دیدگاه یک برنامه نویس enterprise، بستر OSGi چنان روش ساده ای دارد! که به آسانی می توانید آن را در داخل یک برنامه enterprise جاسازی کنید.

برای مثال در نظر بگیریدکه می خواهید یک برنامه تحت وب پیچیده را develop کنید. بطوریکه برنامه را به ماژول های چندگانه بشکنید. یک ماژول برای لایه view دیگری برای لایه DAO و سومین ماژول برای لایه دسترسی به بانک اطلاعاتی. استفاده از یک بستر embedded OSGi برای مدیریت وابستگی های متقابل این ماژول ها، شما را قادر خواهد ساخت که لایه DAO خود را بدون restart کردن برنامه ، به روز رسانی کنید (از یک DAO با سرعت کم به یک DAO سریع).

از زمانیکه برنامه با خصوصیت OSGi کامپایل می شود، باید این قابلیت را داشته باشد که در هر بستر سازگار OSGi دیگر نیز اجرا شود. در حال حاضر ۳ بستر کد باز OSGi متداول وجود دارد:

- ✓ Equinox: یک modular java runtime در قلب Eclipse IDE است که تمام ویژگی های ضروری و بسیاری از ویژگی های اختیاری OSGi R4 را پیاده سازی میکند.
- ✓ Knopflerfish: یک پیاده سازی کد باز از OSGi R3، OSGi R4 است. نسخه ۲ آن تمام ویژگی های ضروری و بعضی از ویژگی های اختیاری R4 را پیاده سازی میکند.
- ✓ ApacheFelix: یک بستر OSGi کد باز از بنیاد نرم افزاری آپاچه است. از زمان نوشتن این بستر، این بستر بطور کامل با خصوصیات OSGi R4 سازگار نیست.

ابزار لازم جهت پیاده سازی OSGi

در OSGi ، نرم افزار در یک شکلی از bundle توزیع می شود. یک bundle شامل کلاسهای جاوا و دیگر منابعی است که توابع را به deviceها ارائه می دهند و سرویسها و package هایی را برای دیگر bundleها فراهم می آورند. تقریبا در تمام مقالات موجود، از Eclipse IDE جهت تولید برنامه استفاده می کنند تا از Equinox که یک plug-in آماده در این IDE است به عنوان بستر OSGi استفاده شود. Eclipse نه تنها Wizard ی را برای ایجاد bundle های OSGi فراهم می آورد بلکه همچنین دارای یک بستر Equinox OSGi تعبیه شده است تا شما بتوانید از آن برای اجرا و debug کردن plug-inهای OSGi استفاده کنید. توجه داشته باشید که همه plug-in های Eclipse بطور خاصی یک OSGi bundle همراه با بعضی کدهای خاص Eclipse اضافی هستند. Eclipse همچنین به شما این اجازه را می دهد تا OSGi bundle های سازگار و استاندارد را بدون کد خاصی بسازید.

اما در این مقاله جهت شفاف کردن هر چه بیشتر پیاده سازی OSGi، به شکل کاملا جداگانه ابتدا bundleها را در Intellij IDE که هیچگونه wizardی برای فراهم آوردن آنها ندارد، می نوسیم و سپس آنرا در Knopflerfish به عنوان بستر OSGi، که خودمان آنرا دانلود و نصب کرده ایم، اجرا می کنیم!

علت انتخاب Knopflerfish نصب آسان آن و همچنین فراهم آوردن یک GUI تحت Desktop خوب است که به شما کمک خواهد کرد تا اولین bundleهای خود را در OSGi framework، deploy کنید. در ابتدا روش نصب Knopflerfish را اجمالا توضیح می دهیم و سپس به ایجاد اولین OSGi bundle و deploy آن در این framework خواهیم پرداخت. در پایان این مقاله شما درک پایه ای از برنامه نویسی OSGi خواهید داشت.

نصب Knopflerfish OSGi framework

نصب Knopflerfish OSGi واقعا ساده است. ابتدا به آدرس www.knopflerfish.org رفته و سپس با مراجعه به صفحه download، آنرا دانلود کنید. توصیه می کنیم که نسخه کامل آن که حاوی همه مستندات و sourceها می باشد، knopflerfish_osgi_3.2.0.jar، را دانلود کنید.

قبل از هر چیز لطفا از داشتن یک بسته نصب شده توسعه نرم افزار java مطمئن شوید. همچنین فرض می شود که شما Intellij IDE را نیز دارید. قدم بعدی، باز کردن فایل jar دانلود شده است. شما می توانید به آسانی نام فایل را از jar. به zip. تغییر دهید ، knopflerfish_osgi_3.2.0.zip، و آن را با winzip باز کنید.

بعد از extract کردن فایل zip یک شاخه حاوی فایل های مربوط به Knopflerfish OSGi ایجاد خواهد شد. حال این شاخه را در <install_directory> مورد نظر خود کپی کنید.

برای شروع Knopflerfish OSGi framework ، کافیست به سادگی به آدرس زیر رفته:

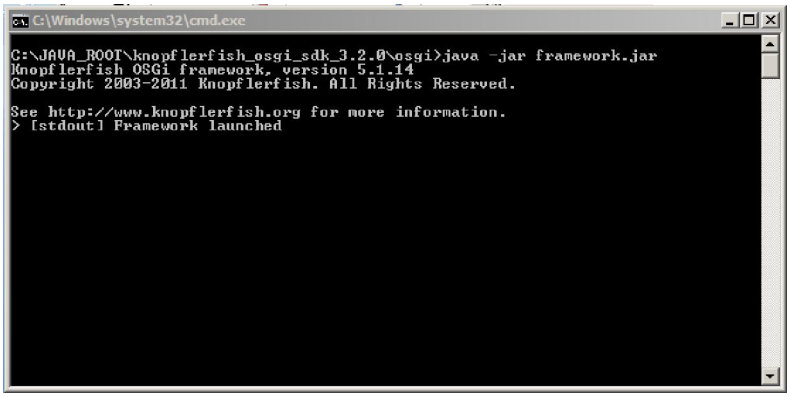
<install-directory>/knopflerfish_osgi_3.2.0/osgi/framework.jar

و یک فایل کوچک بنام startup.bat بسازید، خط زیر را در آن کپی کرده، این فایل را در شاخه فوق الذکر قرار داده و آنرا اجرا کنید.

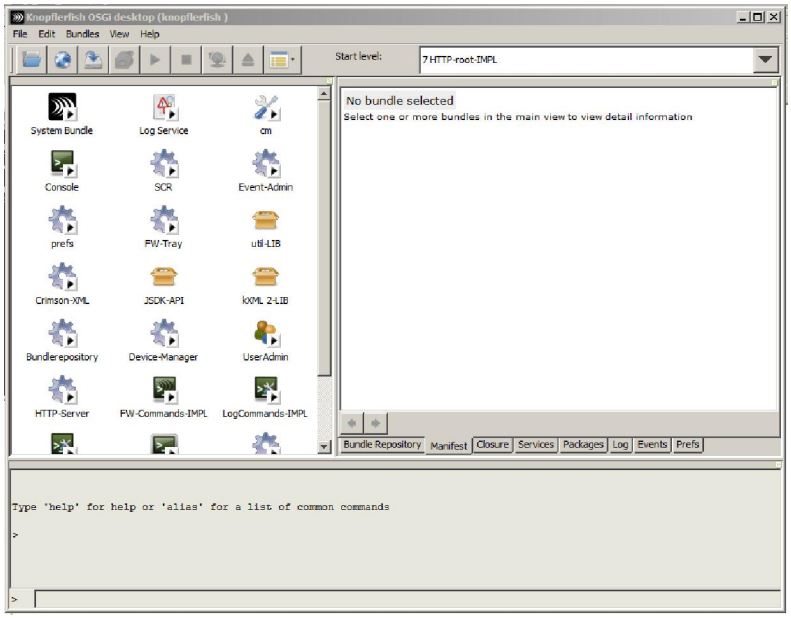
java-jar framework.jar

خواهید دید که پنجره command باز می شود و بلافاصله بعد از آن به اصطلاح Knopflerfish OSGi Desktop بالا خواهد آمد.

تبریک میگویم! شما Knopflerfish را با موفقیت نصب کردید و framework آن را با موفقیت بالا آورید. اگر علاقه مند هستید می توانید اطلاعات بیشتری راجع به option های Knopflerfish و Knopflerfish Desktop در وب سایت KF، http://www.Knopflerfish.org بدست آورید. Knopflerfish OSGi Desktop به شما اجازه خواهد داد تا KF framework های خود را مدیریت کنید. این بخش قابل رویت مدیریت frameworkهای شماست. برای مثال می توانید bundleهای جدید را نصب، start و stop کنید و یا آنها را بروز رسانی و یا uninstall نمایید.



شکل۱: پنجره command پس از کلیک بر روی Startup.bat



شکل۲: Knopflerfish OSGi Desktop

ایجاد اولین Bundle

در این بخش اولین OSGi bundle را ایجاد کرده و آنرا در یک Knopflerfish framework، deploy می کنیم. در برنامه نویسی OSGi، اجزائی که می توانند در framework نصب شوند bundle نامیده می شوند. bundle ها به زبان ساده jar فایل هایی هستند که نوعا شامل فایل های کلاس جاوای service interfaceها، پیاده سازی آنها و چند meta information در یک META-INF/manifedt.mf فایل می باشد. سرویس ها interfaceهای جاوا هستند و وقتی یک bundle یکبار یک سرویس را با OSGi framwork، register کرد، bundleهای دیگر می توانند سرویس منتشر شده را استفاده کنند.

اولین bundle که می خواهیم بسازیم بسادگی یک thread را در background ایجاد می کند که هر ثانیه یکبار عبارت "Hello World!" را چاپ می کند.

ایجاد یک پروژه جدید برای اولین bundle

ابتدا IDE خود را باز کنید (در اینجا Intellij) و یک پروژه جدید جاوا ایجاد نمائید و آنرا SimpleBundle نامگذاری کنید. شاخه های جداگانه برای source و generated classها ایجاد کنید (مثلا ۲ شاخه src و classes). مطمئن شوید که فایل framework.jar را در مسیر build، جاوا، import کرده اید. در غیر این صورت قادر نخواهید بود که به کلاس ها و interfaceهای OSGi تولید شده بوسیله Knopflerfish دسترسی داشته باشید.

ایجاد فایل manifest.mf

در قدم بعدی، یک شاخه META-INF را به پروژه خود اضافه کنید. این شاخه در بر دارنده فایل manifest.mf که bundleها را توصیف می کند، خواهد بود و سپس متن زیر را در فایل manifest.mf کپی کنید:

```
Manifest-Version: 1.0
Bundle-Name: simplebundle
Bundle-SymbolicName: simplebundle
Bundle-Version: 1.0.0
Bundle-Description: Demo Bundle
Bundle-Vendor: Vodafone Pilotentwicklung GmbH
Bundle-Activator: de.vpe.simplebundle.impl.Activator
Bundle-Category: example
Import-Package: org.osgi.framework
```

مهم ترین خصوصیتها در این فایل bundle-Activator و Import-Package هستند. bundle-Activator به framework می گوید که کدام کلاس، کلاس Activator است، این نوعی کلاس برای bundle به منزله main عمل می کند. در مثال پیش روی ما، بعدا یک کلاس به نام de.vpe.simplebundle.impl.Activator خواهیم ساخت و وقتی bundle را deploy و start کنیم، این کلاس بوسیله framwork، lunch خواهد شد.

خصوصیت Import-package به framework می گوید که bundle نیاز دارد تا به همه کلاسهای موجود در package، org.osgi.framwork دسترسی داشته باشد.

ایجاد یک فایل Ant Build

از Ant برای ساخت پروژه استفاده خواهیم کرد. یک فایل build.xml در راس ساختار شاخه ها بسازید و متن زیر را در آن کپی کنید.

```
<?xml version="1.0"?>
<project name="simplebundle" default="all">
    <target name="all" depends="init,compile,jar"/>
    <target name="init">
        <mkdir dir="./classes"/>
        <mkdir dir="./build"/>
    </target>
    <target name="compile">
        <javac destdir="./classes"
               debug="on"
               srcdir="./src"
                >
        </javac>
    </target>
    <target name="jar">
        <jar basedir="./classes"
             jarfile="./build/simplebundle.jar"
             compress="true"
             includes="**/*"
             manifest="./src/META-INF/MANIFEST.MF"
                />
    </target>
    <target name="clean">
        <delete dir="./classes"/>
        <delete dir="./build"/>
    </target>
</project>
```

هم اکنون می توانید فایل build.xml را اجرا کنید. حال که در Intellij کار می کنید صفحه command prompt را باز کرده به شاخه برنامه (محلی که فایل build.xml قرار دارد) رفته و با اجرای دستور Ant این فایل را اجرا کنید. فایل bulid باید با موفقیت کامپایل شود و پس از اجرا یک فایل به نام simplebundle.jar در شاخه bulid ساخته خواهد شد. اگر این طور نشد، ساختار شاخه های خود را چک کنید و جائیکه لازم است را تغییر دهید.

ساخت یک کلاس Activator

بیشتر bundleها یک کلاس Activator دارند که در فایل manifest.mf مشخص شده است. کلاس Activator از BundleActivator ارث می برد و این نیازمند پیاده سازی دو متد به نام های ()start و ()stop است که بوسیله framework برای مدیریت bundle استفاده می شود.

حال یک package بنام de.vpe.simplebundle.impl بسازید. بطور معمول در برنامه نویسی OSGi، سرویس ها را از پیاده سازی آنها جدا می کنیم. در این مثال، بعنوان اولین bundle هیچ سرویسی را ثبت نخواهیم کرد. De.vpe.simplebundle خالی خواهد بود و زیر package، impl کلاس Activator را که bundle را start می کند را در بر خواهد داشت.

حال یک کلاس بنام Activator که از BundleActivator ارث می برد را بسازید و متن زیر را در آن کپی کنید.

```
package de.vpe.simplebundle.impl;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

/**
 * Created by IntelliJ IDEA.
 * User: Fatima, fatemedavari@gmail.com
 * Date: 12/27/11
 * Time: 10:55 AM
 * To change this template use File | Settings | File Templates.
 */

public class Activator implements BundleActivator {

    public void start(BundleContext bc) throws Exception {
        Activator.bc = bc;
    }

    public void stop(BundleContext bc) throws Exception {
        Activator.bc = null;
    }
}
```

توجه داشته باشید که متدهای ()start و ()stop یک شی بنام BundleContest را دریافت می کنند، شما باید همیشه این شی را یکبار وقتی آن را می گیرید ذخیره کنید و وقتی bundle، stop می شود refrence back آنرا null می کنید. در قدم بعدی، یک زیر کلاس thread که هر ۵ ثانیه یکبار عبارت "!Hello World" را چاپ می کند را ایجاد نمائید.

```
package de.vpe.simplebundle.impl;

/**
 * Created by IntelliJ IDEA.
 * User: Fatima, fatemedavari@gmail.com
 * Date: 12/27/11
 * Time: 10:57 AM
 * To change this template use File | Settings | File Templates.
 */

public class HelloWorldThread extends Thread {
    private boolean running = true;

    public HelloWorldThread() {
    }

    public void run() {
        while (running) {
            System.out.println("Hello World!");
            try {
                Thread.sleep(5000);
```

```
            } catch (InterruptedException e) {
                System.out.println("HelloWorldThread ERROR: " + e);
            }
        }
    }

    public void stopThread() {
        this.running = false;
    }
}
```

در نهایت مادامیکه bundle، start می شود باید یک thread جدید ایجاد کنیم و همچنین یکبار وقتیکه bundle، stop می شود، باید thread را stop کنیم. هم چنین چند خط debugging code برای مشاهده اینکه چه موقع bundle؛ start و stop می شود، را اضافه می کنیم.

```
package de.vpe.simplebundle.impl;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

/**
 * Created by IntelliJ IDEA.
 * User: Fatima, fatemedavari@gmail.com
 * Date: 12/27/11
 * Time: 10:55 AM
 * To change this template use File | Settings | File Templates.
 */

public class Activator implements BundleActivator {
    public static BundleContext bc = null;
    private HelloWorldThread thread = null;

    public void start(BundleContext bc) throws Exception {
        System.out.println("SimpleBundle starting...");
        Activator.bc = bc;
        this.thread = new HelloWorldThread();
        this.thread.start();
    }

    public void stop(BundleContext bc) throws Exception {
        System.out.println("SimpleBundle stopping...");
        this.thread.stopThread();
        this.thread.join();
        Activator.bc = null;
    }
}
```

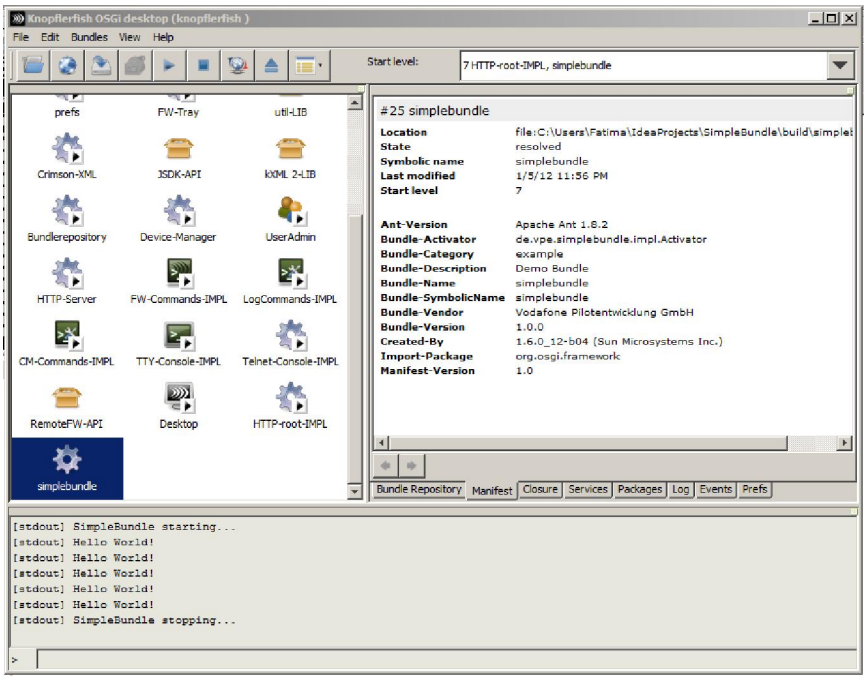
Build و Install کردن اولین bundle

دوباره با استفاده از فایل build.xml پروژه را build کنید. حالا شما باید فایل simplebundle.jar را در شاخه build/. ببینید. Knopflerfish را باز کنید و File>Open Bundle را انتخاب کنید. حال فایل simplebundle.jar را انتخاب و سپس آنرا install کنید. Bundle بطور اتوماتیک active خواهد شد و شما یک icon جدید در پنجره سمت چپ خواهید دید. در این مرحله bundle فقط install شده و حال با تایپ دستور

start simplebundle

باید آنرا start نمایید که باعث می شود هر ۵ ثانیه یکبار عبارت "Hello World!" بر روی console، knopflerfish ظاهر شود.

تبریک میگویم! شما اولین bundle خود را پیاده سازی کردید. به دنیای زیبای OSGi خوش آمدید!



شکل۳: KF Desktop بعد از نصب و start کردن، Hello World Bundle

ایجاد اولین سرویس

این بخش به شما کمک خواهد کرد که اولین service خود را ایجاد کنید. دوباره نیاز به ایجاد یک bundle داریم اما این بار package، interface خالی نخواهد بود و حاویservice interface خواهد بود که یک interface جاوای ساده است.

در ابتدا پروژه SimpleBundle را با نام DateBundle کپی کنید .serviceی که تولید می کنید تاریخ را به قالب مشخصی درمیاورد و تاریخ با آنرا برمیگرداند. قبل از هرچیز در فایل manifest.mf و build.xml نام Bundle را به نام Bundle جدید تغییر دهید.

به روز رسانی فایل manifest.mf

باید یک تغییر کوچک در فایل manifest.mf بدهیم و یک خصوصیت به نام expoer-package به آن اضافه کنیم. در غیر این صورت، بعدا سرویس ها قادر به بازیابی interface سرویس نخواهند بود و بنابراین قادر نخواهند بود که از این سرویس استفاده کنند. مطمئن شوید که فایل manifest.mf شما مشابه متن زیر است:

```
Manifest-Version: 1.0
Bundle-Name: firstservice
Bundle-SymbolicName: firstservice
Bundle-Version: 1.0.0
```

```
Bundle-Description: Demo Bundle
Bundle-Vendor: Vodafone Pilotentwicklung GmbH
Bundle-Activator: de.vpe.firstservice.impl.Activator
Bundle-Category: example
Import-Package: org.osgi.framework
Export-Package: de.vpe.firstservice
```

ایجاد interface سرویس

همچنین توجه داشته باشید که نام package نیز به de.vpe.firstservice تغییر میکند. حال یک interface جاوا بنام First Service ایجاد کنید و متن زیر را در آن کپی نمائید.

```
package de.vpe.firstservice;

import java.util.Date;

/**
 * Created by IntelliJ IDEA.
 * User: Fatima, fatemedavari@gmail.com
 * Date: 12/28/11
 * Time: 9:35 PM
 * To change this template use File | Settings | File Templates.
 */

public interface FirstService {
    public String getFormattedDate(Date date);
}
```

پیاده سازی سرویس

سپس first Service را در زیر package، impl پیاده سازی و متن زیر را در آن کپی کنید.

```
package de.vpe.firstservice.impl;

import java.text.DateFormat;
import java.util.Date;

import de.vpe.firstservice.FirstService;

/**
 * Created by IntelliJ IDEA.
 * User: Fatima, fatemedavari@gmail.com
 * Date: 12/28/11
 * Time: 9:37 PM
 * To change this template use File | Settings | File Templates.
 */

public class FirstServiceImpl implements FirstService {
    public String getFormattedDate(Date date) {
        return DateFormat.getDateInstance(DateFormat.SHORT).format
                (date);
    }
}
```

پیاده سازی interface این سرویس بسیار ساده است اما برای شروع خوب است. این پیاده سازی به سادگی تاریخ را به شکل کوتاهی فرمت شده است را برمی گرداند.

ایجاد یک Activator که سرویس را register میکند

در نهایت ما باید سرویس خود را register نمائیم. این کار در متد ()start از کلاس Activator انجام خواهد شد. ابتدا یک Service Implementation ایجاد می کنیم و سپس این سرویس را تحت نام Service interface ، register می کنیم. همه عملگرهای register کردن از طریق متدهایی در شی BundleContext انجام می شود. این object ارتباط بین bundle و framework را برقرار می کند.

```
package de.vpe.firstservice.impl;

import java.util.Hashtable;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.Constants;
import org.osgi.framework.ServiceRegistration;
import de.vpe.firstservice.FirstService;

/**
 * Created by IntelliJ IDEA.
 * User: Fatima, fatemedavari@gmail.com
 * Date: 12/27/11
 * Time: 10:55 AM
 * To change this template use File | Settings | File Templates.
 */

public class Activator implements BundleActivator {
    public static BundleContext bc = null;

    public void start(BundleContext bc) throws Exception {
        System.out.println(bc.getBundle().getHeaders().get
                (Constants.BUNDLE_NAME) + " starting...");
        Activator.bc = bc;
        FirstService service = new FirstServiceImpl();
        ServiceRegistration registration = bc.registerService(
                FirstService.class.getName(), service, new Hashtable());
        System.out.println("Service registered: FirstService");
    }

    public void stop(BundleContext bc) throws Exception {
        System.out.println(bc.getBundle().getHeaders().get
                (Constants.BUNDLE_NAME) + " stopping...");
        Activator.bc = null;
    }
}
```

متد registerService از BundleContext سه پارامتر را دریافت میکند: اولین پارامتر نام interface سرویس است، دومین آن implementation سرویس است. سومین پارامتر میتواند برای عرضه اطلاعات اضافی درباره سرویس مثل جفت کلید/ مقدار مورد استفاده قرار بگیرد.

ساخت و نصب Service Bundle

دوباره bundle را با استفاده از فایل build.xml بسازید. از تغییر نام jar فایل bundle به اسمی مانند firstservicebundle.jar اطمینان حاصل کنید و با استفاده از KF Desktop آنرا install کنید. حال باید پیام های debug ی که به کد اضافه کرده اید را مشاهده نمایید.

bundle بعدی که خواهید ساخت، سرویسی را که در این bundle فقط register کرده اید را استفاده خواهد کرد.

استفاده از سرویس های دیگر

پروژه FirstService را کپی کرده و به نام FirstServiceUser تغییر نام دهید. فراموش نکنید که نام های فایل و مشخصات را در فایل build.xml ، manifest.mf و همچنین نام های package پروژه جدید را تغییر دهید.

bundleی که خواهید ساخت فقط از سرویس ها استفاده خواهد کرد، بنابراین دوباره package سرویس تهی خواهیم داشت. تنها کلاسی که باید برای این bundle ایجاد کنید یک کلاس Activator جدید است. این Activator به FirstService نگاه خواهد کرد و از آن استفاده خواهد نمود.

به روز رسانی فایل manifest.mf

فایل manifest شما برای bundle کاربر FirstService باید به شکل زیر باشد:

```
Manifest-Version: 1.0
Bundle-Name: firstserviceuser
Bundle-SymbolicName: firstserviceuser
Bundle-Version: 1.0.0
Bundle-Description: Demo Bundle
Bundle-Vendor: Vodafone Pilotentwicklung GmbH
Bundle-Activator: de.vpe.firstserviceuser.impl.Activator
Bundle-Category: example
Import-Package: org.osgi.framework,de.vpe.firstservice
```

توجه داشته باشید که یک کاما و نام یک package جدید به Import-Package اضافه شده است. بدین گونه ما اظهار می کنیم که bundle ما نیاز به داشتن دسترسی به de.vpe.firstservicepackage دارد. framework همواره در دسترس بودن این package را قبل از اینکه Activator، start شود، چک می کند.

بازیابی یک سرویس – روش نامناست

هر زمان که یک سرویس را بازیابی می کنیم، باید این درک را داشته باشیم که OSGi Framework یک محیط کاملا دینامیک است محیطی که سرویس ها در آن ممکن است در دسترس باشند یا نباشند. هر زمان که یک سرویس را دریافت می کنید، این نکته خیلی مهم است که دوباره چک کنید که شما یک سرویس پیاده سازی شده معتبر را دریافت کرده اید و یا یک Null را. بلافاصله بعد

از استفاده از سرویس، همچنین باید آن را unget کنید، این بدان معناست که framework از اینکه شما بیش از این نیاز به استفاده از سرویس را ندارید، مطلع شود.

اولین مثال ما که ساده ترین نیز هست، نامناسب ترین نوع در مورد کیفیت کد، بازیابی FirstService از BundleContext و استفاده از سرویس می باشد. بعدا نشان خواهیم داد که چرا این کد به چندین صورت مشکل ساز خواهد بود.

```
package de.vpe.firstserviceuser.impl;

import java.util.Date;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.Constants;
import org.osgi.framework.ServiceReference;
import de.vpe.firstservice.FirstService;

/**
 * Created by IntelliJ IDEA.
 * User: Fatima, fatemedavari@gmail.com
 * Date: 12/27/11
 * Time: 10:55 AM
 * To change this template use File | Settings | File Templates.
 */

public class Activator implements BundleActivator {
    public static BundleContext bc = null;

    public void start(BundleContext bc) throws Exception {
        System.out.println(bc.getBundle().getHeaders().get(
                Constants.BUNDLE_NAME)
                + " starting...");
        Activator.bc = bc;
        ServiceReference reference = bc.getServiceReference
                (FirstService.class.getName());
        FirstService service = (FirstService) bc.getService(reference);
        System.out.println("Using FirstService: formatting date: " +
                service.getFormattedDate(new Date()));
        bc.ungetService(reference);
    }

    public void stop(BundleContext bc) throws Exception {

        System.out.println(bc.getBundle().getHeaders().get(Constants.BUNDLE_NAME
        ) + " stopping...");
        Activator.bc = null;
    }
}
```

ابتدا یک ServiceReference را از BundleContext بازیابی می کنیم. متد ()getServiceReferenced به سادگی نام interface سرویسی را که مایل به استفاده از آن هستیم را درخواست می کند. هنگامیکه ما یک ServiceReference داریم، از متد ()getService برای بدست آوردن شی سرویس پیاده سازی شده، Cast آن به FirstService و استفاده از آن کمک می گیریم.

حالا قادر هستید که پروژه را تولید و آنرا با استفاده از KF Desktop ، install کنید. اگر FirstService، Start شود، همه چیز روبه راه خواهد بود و شما خروجی debug را خواهید دید.

مشکل اینجاست که هیچ تضمینی برای در دسترس بودن واقعی FirstService وجود ندارد. مراحل زیر را امتحان کنید: هر دو سرویس را stop کنید و سپس ابتدا bundleی را start کنید که FirstService را استفاده می کند. احتمالا شما یک پیغام NullPointerException را دریافت خواهید کرد، به دلیل اینکه متد ()getServiceReference مقدار null را برگردانده است (هیچ سرویسی هنوز register نشده است بنابراین framework نمی تواند آنچه شما درخواست می کنید را به شما بدهد).

یک راه حل بهتر این است که مقدار برگشتی را چک کنیم که null است یا نه:

```
ServiceReference reference = bc.getServiceReference
                (FirstService.class.getName());
        if (reference != null) {
            FirstService service = (FirstService) bc.getService(reference);
            System.out.println("Using FirstService: formatting date: " +
                    service.getFormattedDate(new Date()));
            bc.ungetService(reference);
        } else {
            System.out.println("No Service available!");
        }
```

این راه حل مشکل ظاهر شدن پیغام NullPointerException را حل میکند، اما اگر سرویس ما در start up در دسترس نباشد، ما هرگز نمی توانیم این سرویس را استفاده کنیم! به هر صورت ما باید به طور مرتب چک کنیم که آیا سرویس در دسترس هست یا نه. یا حتی بهتر از آن اینکه framework باید در اسرع وقت به ما اطلاع دهد که سرویس در دسترس هست یا نه. شما می توانید این امکان را با استفاده از ServiceListenerها بدست آورید.

استفاده از ServiceListener برای سرویس هایی که بصورت دینامیک متصل شده

با استفاده از BudleContext این امکان وجود دارد که بتوان یک ServiceListener را با framework ، register کرد. با یک شی فیلتر اختیاری، شما دقیقا می توانید مشخص کنید که برای کدام سرویس ها می خواهید ServiceEventها را دریافت کنید. هر زمانی که یک سرویس جدید register شود، یک سرویس unregister شود و یا خصوصیت آن تغییر کند آنگاه framework، event های سرویس را صادر خواهد کرد.

در مثال بعدی کد متد ()start اصلاح شده را نشان می دهد، ابتدا یک ServiceListeriner با framework، register می شود. رشته فیلتر در سبک LDAP (style) است و به framework می گوید که فقط eventهای سرویس که مربوط به FirstService interface میشود را به ما ارسال کند. توجه داشته باشید که در اینجا ما FirstService را در متد ()start مستقیما بازیابی نمی کنیم. به جای آن ما یک حقه کوچک می زنیم و همه سرویس هایی را که با فیلتر ما match هستند را به دست می آوریم. سپس eventهای ServiceRigestered را برای هر سرویسی که یافت می شود ارسال می کنیم (در این مثال چون فقط یک سرویس وجود دارد پس یک serviceEvent داریم). حال از اضافه کردن ServiceListiner interface به

Activator خود اطمینان حاصل کنید. این کار شما را مجبور خواهد کرد که یک متد serviceChanged را نیز اضافه کنید (متن زیر را نگاه کنید):

```
    public void start(BundleContext bc) throws Exception {
        System.out.println("start " + getClass().getName());
        Activator.bc = bc;
        String filter = "(objectclass=" + FirstService.class.getName() + ")";
        bc.addServiceListener(this, filter);
        ServiceReference references[] = bc.getServiceReferences(null, filter);
        for (int i = 0; references != null && i < references.length; i++) {
            this.serviceChanged(new ServiceEvent(ServiceEvent.REGISTERED,
references[i]));
        }
    }
```

متد serviceChanged همه ServiceEvent ها را برای تغییرات سرویس FirstService دریافت خواهد کرد. متدی که اینجا پیاده سازی شده است، وقتی یک سرویس register میشود شروع به استفاده از آن می کند (یک thread را start می کند که هر ثانیه سرویس را استفاده کند) و هنگامی که سرویس unregister شد استفاده از آن را stop می کند. اگر سرویس تغییر کند (برای مثال خصوصیات سرویس تغییر کنند) استفاده از سرویس را stop، reference جدید برای سرویس بدست آورده و دوباره شروع به استفاده از آن می کند.

```
    public void serviceChanged(ServiceEvent event) {
        switch (event.getType()) {
            case ServiceEvent.REGISTERED:
                log("ServiceEvent.REGISTERED");
                this.service = (FirstService)
Activator.bc.getService(event.getServiceReference());
                this.startUsingService();
                break;
            case ServiceEvent.MODIFIED:
                log("ServiceEvent.MODIFIED received");
                this.stopUsingService();
                this.service = (FirstService)
Activator.bc.getService(event.getServiceReference());
                this.startUsingService();
                break;
            case ServiceEvent.UNREGISTERING:
                log("ServiceEvent.UNREGISTERING");
                this.stopUsingService();
                break;
        }
    }

    private void stopUsingService() {
        this.thread.stopThread();
        try {
            this.thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        this.service = null;
    }

    private void startUsingService() {
        this.thread = new ServiceUserThread(this.service);
        this.thread.start();
```

```
    }

    private void log(String message) {

System.out.println(Activator.bc.getBundle().getHeaders().get(Constants.BUNDLE_NA
ME)+ ": " + message);
    }
```

حال دوباره می توانید پروژه خود را ایجاد و jar فایل bundle جدید (به روز رسانی شده) را install نمائید. خواهید دید که این bundle به محض در دسترس بودن FirstService استفاده از آن را Start می کند. آزمایش کنید و FirstService را هنگامیکه bundle فعال (Active) است stop کنید. خواهید دید که bundle استفاده از سرویس را Stop میکند و منتظر می ماند تا دوباره سرویس در دسترس قرار بگیرد. اگرچه کلاس ServiceUserThread فقط یک thread ساده است اما جهت تکمیل راهنماییهای لازم، متن زیر کد آن میباشد:

```
package de.vpe.firstserviceuser.impl;

import de.vpe.firstservice.FirstService;
import java.util.Date;

/**
 * Created by IntelliJ IDEA.
 * User: Fatima, fatemedavari@gmail.com
 * Date: 12/31/11
 * Time: 12:56 PM
 * To change this template use File | Settings | File Templates.
 */

public class ServiceUserThread extends Thread {
    private FirstService service = null;
    private boolean running = true;

    public ServiceUserThread(FirstService service) {
        this.service = service;
    }

    public void run() {
        Date date = null;
        String formattedDate = null;
        while (running) {
            date = new Date();
            try {
                formattedDate = this.service.getFormattedDate(date);
            } catch (RuntimeException e) {
                System.out.println("RuntimeException occured during service
usage: " + e);
            }
            System.out.println("ServiceUserThread: converted date has value:" +
formattedDate);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println("ServiceUserThread ERROR: " + e);
            }
        }
    }

    public void stopThread() {
        this.running = false;
```

```
    }
}
```

همانطور که مشاهده کردید، کدهای زیادی جهت افزایش توانایی دینامیک برای start و stop کردن استفاده از سرویس های دیگر می توان نوشت. خوشبختانه یک کلاس سودمند در دسترس است که به شما در حل این مشکل کمک می کند. کلاس ServiceTracker جهت مانیتور کردن سرویس ها برای شما مهیاست. در بخش بعدی نشان خواهیم داد که چطور از ServiceTracker استفاده کنید.

استفاده از ServiceTracker برای Track کردن سرویس ها

یک شی ServiceTracker بطور اتوماتیک همه ServiceEventهای یک سرویس خاص را track می کند و به شما این امکان را می دهید که آنرا customize کنید تا وقتی یک سرویس ظاهر می شود و یا از بین می رود چه اتفاقی بیفتد. برای فعال سازی این customization، شما باید یک ServiceTrackerCustomizer interface را پیاده سازی کنید و آنرا برای یک شی ServiceTracker فراهم آورید.

کدی که در ادامه آمده است به روز رسانی شده متد ()start از کلاس Activator است. خواهید دید که حالا به خاطر اینکه از یک ServiceTracker استفاده می کنیم، بخش بیشتری از کد به خارج از Activator منتقل می شوند.

```
package de.vpe.firstserviceuser.impl;

import org.osgi.framework.*;
import de.vpe.firstservice.FirstService;
import org.osgi.util.tracker.ServiceTracker;

/**
 * Created by IntelliJ IDEA.
 * User: Fatima, fatemedavari@gmail.com
 * Date: 12/27/11
 * Time: 10:55 AM
 * To change this template use File | Settings | File Templates.
 */

public class Activator implements BundleActivator {
    public static BundleContext bc = null;
    private FirstService service;
    private ServiceUserThread thread;

    public void start(BundleContext bc) throws Exception {

System.out.println(bc.getBundle().getHeaders().get(Constants.BUNDLE_NAME)+ "
starting...");
        Activator.bc = bc;
        MyServiceTrackerCustomizer customizer = new
MyServiceTrackerCustomizer(bc);
        ServiceTracker tracker = new ServiceTracker(bc,
FirstService.class.getName(), customizer);
        tracker.open();
    }
```

حال نگاهی به کلاس MyServiceTrackerCustomizer می اندازیم که ServiceTrackerCustomizer interface را پیاده سازی می کند.

```
package de.vpe.firstserviceuser.impl;

import org.osgi.framework.*;
import de.vpe.firstservice.FirstService;
import org.osgi.util.tracker.ServiceTrackerCustomizer;

/**
 * Created by IntelliJ IDEA.
 * User: Fatima, fatemedavari@gmail.com
 * Date: 1/2/12
 * Time: 12:16 PM
 * To change this template use File | Settings | File Templates.
 */
public class MyServiceTrackerCustomizer implements ServiceTrackerCustomizer {
    private ServiceUserThread thread = null;
    private BundleContext bc;

    public MyServiceTrackerCustomizer(BundleContext bc) {
        this.bc = bc;
    }

    public Object addingService(ServiceReference reference) {
        FirstService service = (FirstService) bc.getService(reference);
        if (this.thread == null) {
            this.thread = new ServiceUserThread(service);
            this.thread.start();
            return service;
        } else
            return service;
    }

    public void modifiedService(ServiceReference reference, Object
            serviceObject) {
        this.thread.stopThread();
        try {
            this.thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        FirstService service = (FirstService) bc.getService(reference);
        this.thread = new ServiceUserThread(service);
        this.thread.start();
    }

    public void removedService(ServiceReference reference, Object
            serviceObject) {
        this.thread.stopThread();
        try {
            this.thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        this.thread = null;
    }
}
```

متد addingService یک سرویس را می گیرد و اگر threadی موجود نباشد یک thread جدید را start می کند. سپس چک می کنیم که آیا thread، Null است یا نه؟ زیرا در چنین صورتی تمایل داریم که یک thread جدید را start کنیم. می تواند این شرایط نیز پیش بیاید که تعداد زیادی FirstService، register شده باشند، اما حتی در چنین صورتی نیز ما فقط می خواهیم از یک سرویس در یک thread استفاده کنیم.

متد modifiedService به سادگی اجرای thread را stop می کند و آن را با یک سرویس جدید restart می کند. برای ارتقای سودمندی، در واقع ما چک می کنیم آیا سرویسی که تغییر کرده است همان سرویسی است که در حال استفاده از آن هستیم یا نه.

درنهایت متد remoreService به سادگی اجرای thread را stop می کند. بعد از عمل return در این متد، سرویس دیگر استفاده نخواهد شد.

تا این مرحله مفاهیم کلی OSGi جهت آشنایی ارائه شد و اکنون قادر هستید که آنرا پیاده سازی کنید. موفق باشید و از استفاده آن در پیاده سازی applicationهای خود لذت ببرید!

منابع:

- Gravity: Richard S. Hall, OSGi and Gravity Service Binder Tutorial, 2004, http://oscarosgi.
- sourceforge.net/tutorial/
- KF: Erik Wistrand, Develop OSGi Bundles, 2004, http://www.knopflerfish.org/programming.html
- OSGi Intro: OSGi Alliance, OSGi Technology, 2004,
- http://www.osgi.org/osgi_technology/index.asp?section=2
- OSGi Platform: OSGi Initiative, OSGi Service Platform R3, March 2004, http://www.osgi.org