EBook Gratis

APRENDIZAJE

ejb

Free unaffiliated eBook created from
Stack Overflow contributors.

#ejb

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: ejb

It is an unofficial and free ejb ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ejb.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con ejb

Observaciones

Esta sección proporciona una descripción general de qué es ejb y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de ejb, y vincular a los temas relacionados. Dado que la Documentación para ejb es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Examples

Configurando EJB con JBoss AS 7.1

1. Información general

En este artículo vamos a discutir cómo comenzar con Enterprise JavaBeans (EJB). Usaremos JBoss AS 7.1.1.Final, pero usted es libre de usar cualquier servidor de su elección.

2. Dependencias de Maven para el frijol

Para usar EJB, asegúrese de agregar la última versión a la sección de dependencias de su archivo pom.xml:

```
<dependency>
    <groupId>org.jboss.spec.javax.ejb</groupId>
    <artifactId>jboss-ejb-api_3.2_spec</artifactId>
    <version>1.0.0.Final</version>
</dependency>
```

Asegúrese de agregar las dependencias de JBoss correctamente, ya que usaremos JBoss como nuestro servidor de aplicaciones en este tutorial. En la parte posterior del tutorial, analizaremos en detalle cómo configurar la compilación de Maven para el proyecto.

3. EJB Remote

Primero creamos la interfaz de Bean llamada HelloWorldRemote.

```
public interface HelloWorldRemote {
    public String getHelloWorld();
}
```

Ahora implementaremos la interfaz anterior y la llamaremos `HelloWorldBean` .

```
@Stateless
public class HelloWorldBean implements HelloWorldRemote {
```

```
    public HelloWorldBean() {

    }

    @Override
    public String getHelloWorld(){
        return ("Hello World");
    }
}
```

Tenga en cuenta la notación `@Stateless` en la parte superior de la declaración de clase. Denota un bean de sesión sin estado.

4. Maven Setup para Remote Bean

En esta sección analizaremos cómo configurar Maven para compilar y ejecutar la aplicación en el servidor.

Veamos los complementos uno por uno.

4.1. Plugin compilador

El compilador-complemento-compilador se usa para compilar las fuentes de nuestro proyecto.

Aquí hemos utilizado la versión 2.3.1 del complemento con el JDK de origen y destino establecido en 1.7 en la configuración.

Hemos definido esta configuración como propiedades dentro de la etiqueta y la remitimos a través de $ {propiedad}.

```
<version.compiler.plugin>2.3.1</version.compiler.plugin>
<!-- maven-compiler-plugin -->
<maven.compiler.target>1.7</maven.compiler.target>
<maven.compiler.source>1.7</maven.compiler.source>
```

4.2 El Plugin EJB

Este complemento genera el archivo Bean, así como el jar cliente asociado.

Hemos especificado la versión ejb como 3.2 y la propiedad generateClient se establece en true, lo que genera el cliente.

4.3 Despliegue en JBoss

El complemento jboss-as-maven se utiliza para implementar, volver a implementar, anular la implementación o ejecutar la aplicación en JBoss AS 7.

En esta configuración, especificamos el nombre del archivo de compilación igual que el nombre de archivo de compilación del proyecto que es, de forma predeterminada, la versión de artefacto en nuestro caso `ejb-remote-1.0-SNAPSHOT` .

4.4 Dependencias Maven requeridas para EJB

jboss-javaee-6.0 define la versión de las API Java EE 6 de JBoss que queremos usar.

JBoss distribuye un conjunto completo de API de Java EE 6, incluida una lista de materiales (BOM).

Una lista de materiales especifica las versiones de una pila (o una colección) de artefactos. Especificamos esto en la etiqueta para que siempre obtengamos las versiones correctas de los artefactos. El tipo de esta dependencia es un pom que contiene las dependencias requeridas.

```
<dependency>
    <groupId>org.jboss.spec</groupId>
    <artifactId>jboss-javaee-6.0</artifactId>
    <version>${version.org.jboss.spec.jboss.javaee.6.0}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```

4.5 anotaciones

Lo siguiente obtendrá la dependencia de anotaciones:

```
<dependency>
    <groupId>org.jboss.spec.javax.annotation</groupId>
    <artifactId>jboss-annotations-api_1.1_spec</artifactId>
    <scope>provided</scope>
</dependency>
```

4.6 EJB versión 3.2

En el siguiente fragmento de código obtenemos la última versión de las especificaciones:

```
<dependency>
    <groupId>org.jboss.spec.javax.ejb</groupId>
    <artifactId>jboss-ejb-api_3.2_spec</artifactId>
    <version>1.0.0.Final</version>
</dependency>
```

Para ejecutar el proyecto anterior en un servidor JBoss, primero debemos ejecutar:

```
mvn clean install
```

Luego debemos implementarlo en un servidor JBoss en ejecución ejecutando el siguiente comando maven:

```
jboss-as:deploy
```

Ahora debería ver el archivo jar desplegado en el servidor jboss.

Alternativamente, puede copiar el jar disponible de la carpeta de destino en el proyecto y pegarlo

en la carpeta de la aplicación web del servidor.

5. Configurando el proyecto del cliente

Después de crear el bean remoto, debemos probar el bean desplegado creando un cliente.

Primero vamos a discutir la configuración de Maven para el proyecto.

5.1 Plugins de Maven utilizados

El complemento compilador-compilador se utiliza para compilar las fuentes de su proyecto.

Especificamos la versión jdk 1.7 para las clases de origen y destino.

Nuestro cliente es un programa de Java, para ejecutarlo usamos el `exec-maven-plugin` que ayuda a ejecutar los programas del sistema y de Java. Necesitamos especificar el ejecutable (es decir, java), classpath y la clase java (com.baeldung.ejb.client.Client).

La ruta de clase se deja vacía porque el complemento incluye los argumentos de ruta de clase necesarios según las dependencias proporcionadas.

5.2 Dependencias de Maven para el cliente EJB3

Para ejecutar el cliente EJB3 necesitamos incluir las siguientes dependencias.

Dependemos de las interfaces comerciales remotas EJB de esta aplicación para ejecutar el cliente. Así que necesitamos especificar la dependencia jar de cliente ejb. La etiqueta con el valor "ejb-client" se utiliza para especificar la dependencia de este proyecto en el contenedor del cliente EJB.

```
<dependency>
    <groupId>com.theopentutorials.ejb3</groupId>
    <artifactId>ejbmavendemo</artifactId>
    <type>ejb-client</type>
    <version>${project.version}</version>
</dependency>
```

Las dependencias `jboss-transaction-api_1.1_spec` , `jboss-ejb-api_3.1_spec` , `jboss-ejb-client` , `xnio-api` , `xnio-nio` , `jboss-remoting` , `jboss-sasl` , `jboss-marshalling-river` tienen alcance como runtime porque estos son tiempo de ejecución requerido y no durante el tiempo de compilación.

Las dependencias **jboss-javaee-6.0** y **jboss-as-ejb-client-bom** bajo dependencyManagement tienen alcance como importación. Esto se utiliza para incluir información de administración de dependencias desde un POM remoto al proyecto actual. JBoss proporciona estos POM remotos que contienen las dependencias necesarias para ejecutar el cliente.

5.3 Propiedades del cliente JBoss EJB

Cree un archivo en "src / main / resources" y asígnele el nombre jboss-ejb-client.properties.

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
```

```
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port = 4447
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

6. Creando la clase de cliente

Primero creamos una clase ClientUtility:

```
public class ClientUtility {
    private static Context initialContext;
    private static final String PKG_INTERFACES = "org.jboss.ejb.client.naming";

    public static Context getInitialContext() throws NamingException {
        if (initialContext == null) {
            Properties properties = new Properties();
            properties.put(Context.URL_PKG_PREFIXES, PKG_INTERFACES);
            initialContext = new InitialContext(properties);
         }
        return initialContext;
    }
}
```

Ahora vamos a crear la clase de Cliente real que consumirá el bean que implementamos en el servidor:

```
public class Client {

    //The lookup method to get the EJB name
    private static HelloWorldRemote doLookup() {
        Context context = null;
        HelloWorldRemote bean = null;
        try {
            // 1. Obtaining Context
            context = ClientUtility.getInitialContext();
            // 2. Generate JNDI Lookup name
            String lookupName = getLookupName();
            // 3. Lookup and cast
            bean = (HelloWorldRemote) context.lookup(lookupName);

        } catch (NamingException e) {
            e.printStackTrace();
        }
        return bean;
    }

    private static String getLookupName() {

         // The app name is the EAR name of the deployed EJB without .ear suffix.
         // Since we haven't deployed the application as a .ear, the app name for
         // us will be an empty string

        String appName = "";


         // The module name is the JAR name of the deployed EJB without the .jar
         // suffix.
        String moduleName = "ejb-remote-0.0.1-SNAPSHOT";
```

```
        // AS7 allows each deployment to have an (optional) distinct name. This
        // can be an empty string if distinct name is not specified.
        String distinctName = "";

        // The EJB bean implementation class name
        String beanName = "HelloWorldBean";

        // Fully qualified remote interface name
        final String interfaceName = "com.baeldung.ejb.tutorial.HelloWorldRemote";

        // Create a look up string name
        String name = "ejb:" + appName + "/" + moduleName + "/" + distinctName
                + "/" + beanName + "!" + interfaceName;

        return name;
    }
}
```

La clase Cliente consume el bean y genera el resultado.

7. Conclusión

Así que hemos creado un servidor EJB y un cliente que consume el servicio. El proyecto se puede ejecutar en cualquier servidor de aplicaciones.

Lea Empezando con ejb en línea: https://riptutorial.com/es/ejb/topic/5704/empezando-con-ejb

Creditos

S. No	Capítulos	Contributors
1	Empezando con ejb	Community, Pritam Banerjee, RamenChef