(http://baeldung.com)

Guía de integración para Spring y EJB

Última modificación: 31 de enero de 2018

por baeldung (http://www.baeldung.com/author/baeldung/)
(http://www.baeldung.com/author/baeldung/)

Java EE (http://www.baeldung.com/category/java-ee/)
Primavera (http://www.baeldung.com/category/spring/) +

1. Información general

En este artículo, mostraremos cómo **integrar Spring y Remote Enterprise Java Beans (EJB)** .

Para hacer esto, crearemos algunos EJB y las interfaces remotas necesarias, y luego los ejecutaremos dentro de un contenedor JEE. Después de eso, comenzaremos nuestra aplicación Spring y, utilizando las interfaces remotas, crearemos instancias de nuestros beans para que puedan ejecutar llamadas remotas.

Si hay alguna duda sobre qué son los EJB o cómo funcionan, ya hemos publicado un artículo introductorio sobre el tema aquí (http://www.baeldung.com/ejb-intro) .

2. Configuración de EJB

Tendremos que crear nuestras interfaces remotas y nuestras implementaciones EJB. Para que sean utilizables, también necesitaremos un contenedor para contener y administrar los frijoles.

2.1. Interfaces remotas EJB

Comencemos definiendo dos beans muy simples: uno sin estado y otro con estado.

Comenzaremos con sus interfaces:

```
@Remote
public interface HelloStatefulWorld {
    int howManyTimes();
    String getHelloWorld();
}
```

```
@Remote
public interface HelloStatelessWorld {
    String getHelloWorld();
}
```

2.2. Implementación de EJB

Ahora, implementemos nuestras interfaces EJB remotas:

```
@Stateful(name = "HelloStatefulWorld")
public class HelloStatefulWorldBean implements HelloStatefulWorld {

    private int howManyTimes = 0;

    public int howManyTimes() {
        return howManyTimes;
    }

    public String getHelloWorld() {
        howManyTimes++;
        return "Hello Stateful World";
    }
}
```

```
@Stateless(name = "HelloStatelessWorld")
public class HelloStatelessWorldBean implements HelloStatelessWorld {

    public String getHelloWorld() {
        return "Hello Stateless World!";
    }
}
```

Si los beans con estado y sin estado suenan desconocidos, este artículo de introducción (http://www.baeldung.com/ejb-intro) puede ser útil.

2.3. Contenedor EJB

Podemos ejecutar nuestro código en cualquier contenedor JEE, pero para fines prácticos, usaremos Wildfly y el plugin de *carga* Maven para hacer el trabajo pesado por nosotros:

```
<plugin>
    <groupId>org.codehaus.cargo</groupId>
    <artifactId>cargo-maven2-plugin</artifactId>
    <version>1.6.1</version>
    <configuration>
        <container>
            <containerId>wildfly10x</containerId>
            <zipUrlInstaller>
                <url>
                  http://download.jboss.org/wildfly/10.1.0.Final/wildfly-10.1.0.Final.zip (http
                </url>
            </zipUrlInstaller>
        </container>
        <configuration>
            <properties>
                <cargo.hostname>127.0.0.1</cargo.hostname>
                <cargo.jboss.configuration>standalone-full</cargo.jboss.configuration>
                <cargo.jboss.management-http.port>9990</cargo.jboss.management-http.port>
                <cargo.servlet.users>testUser:admin1234!</cargo.servlet.users>
            </properties>
        </configuration>
    </configuration>
</plugin>
```

2.4. Ejecutando los EJB

Con estos configurados, podemos ejecutar el contenedor directamente desde la línea de comandos de Maven:

```
mvn clean package cargo:run -Pwildfly-standalone
```

Ahora tenemos una instancia activa de Wildfly alojando nuestros beans. Podemos confirmar esto por las líneas de registro:

```
java:global/ejb-remote-for-spring/HelloStatefulWorld!com.baeldung.ejb.tutorial.HelloStatefulWor
java:app/ejb-remote-for-spring/HelloStatefulWorld!com.baeldung.ejb.tutorial.HelloStatefulWorld
java:module/HelloStatefulWorld!com.baeldung.ejb.tutorial.HelloStatefulWorld
java:jboss/exported/ejb-remote-for-spring/HelloStatefulWorld!com.baeldung.ejb.tutorial.HelloSta
java:global/ejb-remote-for-spring/HelloStatefulWorld
java:app/ejb-remote-for-spring/HelloStatefulWorld
java:module/HelloStatefulWorld
```

```
java:global/ejb-remote-for-spring/HelloStatelessWorld!com.baeldung.ejb.tutorial.HelloStatelessW
java:app/ejb-remote-for-spring/HelloStatelessWorld!com.baeldung.ejb.tutorial.HelloStatelessWorl
java:module/HelloStatelessWorld!com.baeldung.ejb.tutorial.HelloStatelessWorld
java:jboss/exported/ejb-remote-for-spring/HelloStatelessWorld!com.baeldung.ejb.tutorial.HelloSt
java:global/ejb-remote-for-spring/HelloStatelessWorld
java:app/ejb-remote-for-spring/HelloStatelessWorld
java:module/HelloStatelessWorld
```

3. Configuración de primavera

Ahora que tenemos nuestro contenedor JEE en funcionamiento y nuestros EJB implementados, podemos comenzar nuestra aplicación Spring. Usaremos *spring-boot-web* para que resulte más fácil probarlo manualmente, pero no es obligatorio para la llamada remota.

3.1. Dependencias Maven

Para poder conectarnos a los EJB remotos, necesitaremos la biblioteca de *Wildfly EJB Client* y nuestra interfaz remota:

```
<dependency>
    <groupId>org.wildfly</groupId>
    <artifactId>wildfly-ejb-client-bom</artifactId>
    <version>10.1.0.Final</version>
    <type>pom</type>
</dependency>
<dependency>
    <groupId>com.baeldung.spring.ejb</groupId>
    <artifactId>ejb-remote-for-spring</artifactId>
    <version>1.0.1</version>
    <type>ejb</type>
</dependency>
```

La última versión de *wildfly-ejb-client-bom* se puede encontrar aquí (http://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.wildfly%22%20AND%20a%3A%22wildfly-ejb-client-bom%22) .

3.2. Denominación del contexto de la estrategia

Con estas dependencias en classpath, podemos **instanciar un *javax.naming.Context* para hacer la búsqueda de nuestros beans remotos** . Crearemos esto como un Spring Bean para que podamos conectarlo automáticamente cuando lo necesitemos:

```
@Bean
public Context context() throws NamingException {
    Properties jndiProps = new Properties();
    jndiProps.put("java.naming.factory.initial",
      "org.jboss.naming.remote.client.InitialContextFactory");
    jndiProps.put("jboss.naming.client.ejb.context", true);
    jndiProps.put("java.naming.provider.url",
      "http-remoting://localhost:8080 (remoting://localhost:8080)");
    return new InitialContext(jndiProps);
}
```

Las propiedades son necesarias para **informar tanto la *URL* remota como el contexto de la estrategia de nombres** .

3.3. Patrón JNDI

Antes de que podamos conectar nuestros beans remotos dentro del contenedor Spring, necesitaremos saber cómo llegar a ellos. Para esto, usaremos sus enlaces JNDI. Veamos el patrón estándar para estas vinculaciones:

```
${appName}/${moduleName}/${distinctName}/${beanName}!${viewClassName}
```

Tenga en cuenta que, **dado que implementamos un simple *jar en* lugar de un *ear* y no configuramos explícitamente un nombre, no tenemos un *appName* y un *distinctName*** . Hay más detalles en nuestro artículo de introducción de EJB (http://www.baeldung.com/ejb-intro) en caso de que algo parezca extraño.

Usaremos este patrón para unir nuestros beans remotos a los de Spring.

3.4. Construyendo nuestros frijoles primavera

Para llegar a nuestros EJB, usaremos el JNDI antes mencionado. ¿Recuerda las líneas de registro que usamos para verificar si nuestros enterprise beans fueron implementados?

Veremos esa información en uso ahora:

```
@Bean
public HelloStatelessWorld helloStatelessWorld(Context context)
  throws NamingException {

    return (HelloStatelessWorld)
      context.lookup(this.getFullName(HelloStatelessWorld.class));
}
```

```
@Bean
public HelloStatefulWorld helloStatefulWorld(Context context)
  throws NamingException {

    return (HelloStatefulWorld)
      context.lookup(this.getFullName(HelloStatefulWorld.class));
}
```

```
private String getFullName(Class classType) {
    String moduleName = "ejb-remote-for-spring/";
    String beanName = classType.getSimpleName();
    String viewClassName = classType.getName();
    return moduleName + beanName + "!" + viewClassName;
}
```

Debemos ser muy cuidadosos sobre la correcta vinculación JNDI completa , o el contexto no podrá llegar al EJB remoto y crear la infraestructura subyacente necesaria.

Tenga en cuenta que la *búsqueda de* métodos de *Context* arrojará una *NamingException* en caso de que no encuentre el bean que está requiriendo.

4. Integración

Con todo en su lugar, podemos **inyectar nuestros granos en un controlador** , por lo que podemos probar si el cableado es correcto:

```
@RestController
public class HomeEndpoint {

    // ...

    @GetMapping("/stateless")
    public String getStateless() {
        return helloStatelessWorld.getHelloWorld();
    }

    @GetMapping("/stateful")
    public String getStateful() {
        return helloStatefulWorld.getHelloWorld()
          + " called " + helloStatefulWorld.howManyTimes() + " times";
    }
}
```

Comencemos nuestro servidor de Spring y revisemos algunos registros. Veremos la siguiente línea, que indica que todo está bien:

```
EJBCLIENT000013: Successful version handshake completed
```

Ahora, probemos nuestro frijol sin estado. Podemos probar algunos comandos *curl* para verificar que están funcionando como se espera:

```
curl http://localhost:8081/stateless (http://localhost:8081/stateless)
Hello Stateless World!
```

Y revisemos nuestro estado:

```
curl http://localhost:8081/stateful (http://localhost:8081/stateful)
Hello Stateful World called 1 times

curl http://localhost:8081/stateful (http://localhost:8081/stateful)
Hello Stateful World called 2 times
```

5. Conclusión

En este artículo, aprendimos cómo integrar Spring to EJB y hacer llamadas remotas al contenedor JEE. Creamos dos interfaces EJB remotas, y pudimos llamar a aquellos que usan Spring Beans de forma transparente.

Aunque Spring es ampliamente adoptado, los EJB siguen siendo populares en entornos empresariales, y en este ejemplo rápido, hemos demostrado que es posible utilizar tanto las ganancias distribuidas de JavaEE como la facilidad de uso de las aplicaciones de Spring.

Como siempre, el código se puede encontrar en GitHub (https://github.com/eugenp/tutorials/tree/master/spring-ejb) .

>> VERIFIQUE LAS LECCIONES (/rest-with-spring-course#new-modules)

CATEGORÍAS

PRIMAVERA (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
DESCANSO (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)
HTTPCLIENT (HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

TUTORIAL "VOLVER A LO BÁSICO" DE JAVA (HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (HTTP://WWW.BAELDUNG.COM/JACKSON)
TUTORIAL DE HTTPCLIENT 4 (HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE)
REST CON SPRING TUTORIAL (HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/)
TUTORIAL DE SPRING PERSISTENCE (HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/)
SEGURIDAD CON SPRING (HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING)

ACERCA DE

ACERCA DE BAELDUNG (HTTP://WWW.BAELDUNG.COM/ABOUT/)
LOS CURSOS (HTTP://COURSES.BAELDUNG.COM)
TRABAJO DE CONSULTORÍA (HTTP://WWW.BAELDUNG.COM/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
EL ARCHIVO COMPLETO (HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE)
ESCRIBIR PARA BAELDUNG (HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES)
CONTACTO (HTTP://WWW.BAELDUNG.COM/CONTACT)
INFORMACIÓN DE LA COMPAÑÍA (HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
TÉRMINOS DE SERVICIO (HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY)
EDITORES (HTTP://WWW.BAELDUNG.COM/EDITORS)
KIT DE MEDIOS (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF)