Seguridad

Índice

1. Introducción a la seguridad en EJBs

Los servidores Enterprise JavaBeans pueden soportar tres tipos de seguridad: autentificación, comunicación segura y control de acceso. En el módulo de seguridad se ha hecho hincapié en los dos primeros tipos de servicios usando las APIs que proporciona Java. En este apartado veremos qué mecanismos define la especificación EJB para el control de acceso a los EJBs. Revisemos brevemente cada uno de los tipos de seguridad.

- **Autentificación**

  Dicho sencillamente, la autentificación valida la identidad del usuario. La forma más común de autentificación es una simple ventana de login que pide un nombre de usuario y una contraseña. Una vez que los usuarios han pasado a través del sistema de autentificación, pueden usar el sistema libremente, hasta el nivel que les permita el control de acceso. La autentificación se puede basar también en tarjetas de identificación, certificados y en otros tipos de identificación.

- **Comunicación segura**

  Los canales de comunicación entre un cliente y un servidor son un elemento muy importante en la seguridad del sistema. Un canal de comunicación puede hacerse seguro mediante aislamiento físico (por ejemplo, via una conexión de red dedicada) o por medio de la encriptación de la comunicación entre el cliente y el servidor. El aislamiento físico es caro, limita las posibilidades del sistema y es casi imposible en Internet, por lo que lo más usual es la encriptación. Cuando la comunicación se asegura mediante la encriptación, los mensajes se codifican de forma que no puedean ser leídos ni manipulados por individuos no autorizados. Esto se suele consigue mediante el intercambio de claves criptográficas entre el cliente y el servidor. Las claves permiten al receptor del mensaje decodificarlo y leerlo.

- **Control de acceso**

  El control de acceso (también conocido como autorización) aplica políticas de seguridad que regulan lo que un usuario específico puede y no puede hacer en el sistema. El control de acceso asegura que los usuarios accedan sólo a aquellos recursos a los que se les ha dado permiso. El control de acceso puede restringir el acceso de un usuario a subistemas, datos, y objetos de negocio. Por ejemplo, a algunos usuarios se les puede dar permiso de modificar información, mientras que otros sólo tienen permiso de visualizarla.

La mayoría de los servidores EJB soportan la comunicación segura a través del protocolo SSL (Secure Socket Layer) y proporcionan algún mecanismo de autentificación, pero la especificación Enterprise JavaBeans sólo especifica el control de acceso a los EJBs.

Aunque la autentificación no se especifica en EJB, a menudo se consigue usando el API JNDI. Un cliente que usa JNDI puede proporcionar información de autentificación usando este API para acceder a los recursos del servidor. Esta información se suele pasar cuando el cliente intenta iniciar una conexión JNDI con el servidor EJB. El siguiente código muestra cómo se añaden la contraseña y el nombre de usuario a las propiedades de la conexión que se usan para obtener una conexión JNDI con el servidor EJB:

```
properties.put(Context.SECURITY_PRINCIPAL, userName );
properties.put(Context.SECURITY_CREDENTIALS, userPassword);

javax.naming.Context jndiContext = new
             javax.naming.InitialContext(properties);
Object ref= jndiContext.lookup("AccountEJB");
AccountHome accountHome = (AccountHome)
        PortableRemoteObject.narrow(ref, AccountHomeRemote.class);
```

La arquitectura EJB especifica que todas las apliaciones clientes que acceden a un sistema EJB deben estar asociadas con una identidad de seguridad. La identidad de seguridad representa el cliente o bien como un usuario o bien como un rol. Un usuaro podría ser una persona, una credencial de seguridad, un computador o incluso una tarjeta inteligente. Normalmente, el usuario es una persona a la que se le asigna una identidad cuando entra en el sistema. Un rol especifica una categoría de acceso a la que pueden pertenecer distintos usuarios. Por ejemplo, el rol "`ReadOnly`" definiría una categoría en la que sólo se permite el acceso a operaciones de lectura de datos.

Normalmente el servidor EJB permite definir grupos de usarios a los que se les va a asignar las mismas restricciones de seguridad. ¿Cuál es la diferencia entre roles y grupos? La diferencia fundamental es que los roles son dependientes de la aplicación que se despliega, mientras que los grupos son dependientes de la organización. Al desplegar una aplicación es necesario realizar una asignación de roles a usuarios o grupos de usuarios. Esta separación entre roles y usuarios permite hacer la aplicación portable e independiente de la organización en la que se despliega.

Cuando un cliente remoto se autentifica en el sistema EJB, se le asocia una identidad de seguridad que dura el tiempo que está activa la sesión. Esta identidad se encuentra en una base de datos o directorio específico de la plataforma EJB. Esta base de datos o directorio es responsable de almacenar las indentidades individuales, su pertenencia a grupos y las asignaciones de roles a grupos e individuos.

Una vez que se le ha asociado una identidad de seguridad a un cliente remoto, está listo para usar los beans. El servidor EJB realiza un seguimiento de cada cliente y de su identidad. Cuando un cliente invoca un método de un EJB, el servidor EJB pasa implícitamente la identidad del cliente junto con la invocación al método. Cuando el objeto EJB o el EJB home

recibe la invocación debe chequear la identidad para asegurarse de que ese cliente puede usar ese método.

2. Control de acceso basado en roles

En la arquitectura Enterprise JavaBeans, la identidad de seguridad se representa por un objeto `java.security.Principal`. Este objeto actúa como representante de usuarios, grupos, organizaciones, tarjetas inteligentes, etc. frente a la arquitectura de control de acceso de los EJB.

Los descriptores del despliegue incluyen elementos que declaran a qué roles lógicos se permite el acceso a los métodos de los enterprise beans. Estos roles se consideran lógicos porque no reflejan directamente usuarios, grupos o cualquier otra identidad de seguridad en un entorno operacional específico. Los roles se hacen corresponder con grupos de usuarios o usuarios del mundo real cuando el bean se despliega. Esto permite que el bean sea portable ya que cada vez que el bean se despliega en un nuevo sistema, los roles se asignan a usuarios y grupos específicos de ese entorno operacional.

2.1. Definición de roles

He aquí una porción de un fichero de despliegue de un EJB que define dos roles de seguridad, `ReadOnly` y `Administrator`:

```
<security-role>
    <description>
       A este rol se le permite ejecutar cualquier método del
       bean y leer y escribir cualquier dato del bean.
    </description>
    <role-name>
        Administrator
    </role-name>
</security-role>

<security-role>
    <description>
        A este rol se le permite localizar y leer información del
bean.
        A este rol no se le permite modificar los datos del bean.
    </description>
    <role-name>
        ReadOnly
    </role-name>
</security-role>
```

Los nombres de los roles en este descriptor no son nombres reservados o especiales con significados predefinidos; son simplemente nombres lógicos escogidos por el ensamblador del bean.

Una vez que se declaran los roles, pueden asociarse con métodos en los beans. El elemento `<method-permission>` es en el que definimos a qué métodos puede acceder cada uno de los roles.

```
<method-permission>
    <role-name>Administrator</role-name>
    <method>
        <ejb-name>CustomerEJB</ejb-name>
        <method-name>*</method-name>
    </method>
</method-permission>
<method-permission>
    <role-name>ReadOnly</role-name>
    <method>
        <ejb-name>CustomerEJB</ejb-name>
        <method-name>getName</method-name>
    </method>
    <method>
        <ejb-name>CustomerEJB</ejb-name>
        <method-name>getAddress</method-name>
    </method>
    <method>
        <ejb-name>CustomerEJB</ejb-name>
        <method-name>findByPrimaryKey</method-name>
    </method>
</method-permission>
```

En el primer `method-permission` el rol `Administrator` se asocia con todos los métodos del EJB CustomerEJB. En el segundo, se limita el acceso del rol `ReadOnly` sólo a tres métodos: `getName()`, `getAddress()` y `findByPrimaryKey()`. Cualquier intento de un rol `ReadOnly` de acceder a un método que no sea éstos resultará en una excepción.

2.2. Asignación de usuarios a roles

Hasta ahora hemos comentado que el acceso a los métodos se restringe mediante roles, para no definir usuarios concretos en la aplicación. Pero en algún momento hay que definir los usuarios y grupos (*principals*) asociados a los roles, ya que la autentificación en Java se realiza en base a ellos.

En la arquitectura EJB se promueve una separación de papeles, en la que unas tareas son realizadas por los desarrolladores y otras por los que se encargan de desplegar la aplicación.

En el caso de la seguridad, el desarr desarrollador del bean es el que debe de preocuparse de definir los roles y restringir el acceso a los métodos, mientras que la persona que hace el despliegue se ocupa de asignar los roles a *principals*.

La forma de relacionar *principals* con roles es mediante el descriptor de despliegue propio del servidor de aplicaciones. En el servidor de aplicaciones BEA WebLogic Server esto se declara en el fichero "**weblogic-ejb-jar.xml**", mediante el elemento `security-role-assignment`. Por ejemplo, en el siguiente código se asocian los *principals* "weblogic" y "admin" con el rol "administrador" y los *principals* "dgl", y "admin" al rol "bibliotecario" (un *principal* puede tener más de un rol, como es el caso de "admin"). Hacemos notar también que un principal puede ser un usuario o un grupo, como el segundo caso, en el que se utiliza el nombre de grupo "Bibliotecario". Los *principals* se definen en la consola de administración de WebLogic.

```
<security-role-assignment>
   <role-name>administrador</role-name>
   <principal-name>admin</principal-name>
   <principal-name>weblogic</principal-name>
</security-role-assignment>

<security-role-assignment>
   <role-name>bibliotecario</role-name>
   <principal-name>dgl</principal-name>
   <principal-name>admin</principal-name>
   <principal-name>Bibliotecario</principal-name>
</security-role-assignment>
```

Una vez que el bean se despliega y se pone en marcha, el contenedor se encarga de comprobar que los usuarios acceden únicamente a aquellos métodos a los que tienen permisos. Esto se consigue propagando la identidad de seguridad, el objeto `Principal`, en cada invocación del cliente al bean. Cuando el cliente invoca un método de un bean, el objeto `Principal` del cliente se chequea para comprobar si el rol asociado tiene los privilegios necesarios.

Si un bean intenta acceder a cualquier otro bean mientras que está sirviendo a un cliente, pasará al segundo bean la identidad del cliente para que se realiza un control de acceso en este segundo bean. De esta manera, el `Principal` del cliente se propaga de un bean al siguiente, asegurando que se controla el acceso acceda o no directamente al EJB.

3. Métodos no-chequeados

Es posible definir un conjunto de métodos como `unchecked`, lo que significa que los permisos de seguridad no se chequean. Un método no chequeado puede invocarse por cualquier cliente, sin importancia de el rol que ese cliente esté usando. A continuación hay un

ejemplo de declaración de métodos no-chequeados:

```
<method-permission>
    <unchecked/>
    <method>
        <ejb-name>CabinEJB</ejb-name>
        <method-name>*</method-name>
    </method>
    <method>
        <ejb-name>CustomerEJB</ejb-name>
        <method-name>findByPrimaryKey</method-name>
    </method>
</method-permission>
<method-permission>
    <role-name>administrator</role-name>
    <method>
        <ejb-name>CabinEJB</ejb-name>
        <method-name>*</method-name>
    </method>
</method-permission>
```

Esta declaración nos dice que todos los métodos del EJB `Cabin`, junto con el método `findByPrimaryKey()` del EJB `Customer`, son no-chequeados. Un segundo elemento le da al administrador permiso para acceder a todos los métodos del EJB Cabin. El permiso no-chequeado siempre tiene prioridad sobre cualquier otro permiso.

4. La identidad de seguridad runAs

Mientras que los elementos `method-permission` especifican qué roles tienen acceso a qué métodos del bean, el elemento `security-identity` especifica bajo qué rol se ejecuta el EJB, usando el elemento `runAs`. En otras palabras, el objeto rol que se define en `runAs` se usa como la identidad del enterprise bean cuando éste intenta invocar métodos en otros beans. Esta identidad no tiene por qué ser la misma que la identidad que accede al bean por primera vez.

Por ejemplo, los siguientes elementos del descriptor de despliegue declaran que el método `create()` sólo puede accederse por el rol `JimSmith` y que el EJB `Cabin` siempre se va ejecutar con una identidad `Administrator`.

```
<enterprise-beans>
...
    <entity>
        <ejb-name>CabinEJB</ejb-name>
        ...
        <security-identity>
            <run-as>
               <role-name>Administrator</role-name>
            </run-as>
```

```
        </security-identity>
        ...
    </entity>
...
</enterprise-beans>
<assembly-descriptor>
<security-role>
    <role-name>Administrator</role-name>
</security-role
<security-role>
    <role-name>JimSmith</role-name>
</security-role>
...
<method-permission>
    <role-name>JimSmith</role-name>
    <method>
        <ejb-name>CabinEJB</ejb-name>
        <method-name>create</method-name>
    </method>
</method-permission>
...
</assembly-descriptor>
```

Esta clase de configuración es útil cuando el enterprise bean o los recursos accedidos en el cuerpo del método requieren un rol distinto del que ha sido usado para obtener acceso al método. Por ejemplo, el método `create()` podría llamar a un método en el enterprise bean X que requiera la identidad de seguridad de `Administrador`.

Podemos entender la secuencia de cambio de identidades de la siguiente forma:

1. El cliente invoca un método del bean con una identidad `Id1`.
2. El bean comprueba si la identidad `Id1` tiene permiso para ejecutar el método. La tiene.
3. El bean consulta el elemento `security-identity` y cambia la identidad a la que indica ese elemento. Supongamos que es la identidad `Id2`.
4. El bean realiza las llamadas dentro del método con la identidad `Id2`.

Es obligado resaltar que hay que usar con precaución esta funcionalidad, ya que con ella podemos atribuir cualquier rol a cualquier usuario.

Una limitación de la funcionalidad es que es obligado definir un único rol para todos los métodos.

5. Gestión de seguridad programativa en el EJB

En el código del EJB es posible comprobar qué usuario o grupo ha realizado la llamada al EJB y si tiene un determinado rol asociado. Para ello se usan los siguientes métodos del `SessionContext`:

- `Principal getCallerPrincipal()`: devuelve el objeto `Principal` asociado al usuario o grupo que ha llamado al método.
- `Boolean isCallerInRole(String rol)`: devuelve `true` o `false` dependiendo de si el usuario o grupo que ha llamado al método pertenece al rol que se le pasa como parámetro.

El objeto `SessionContext` se obtiene declarando el método `setSessionContext(SessionContext ctx)` del ciclo de vida del EJB y guardando el contexto en una variable de instancia del EJB. El contenedor EJB llamará a este método en el momento de creación del enterprise bean.

Un ejemplo de código en el que se utilizan estos métodos:

```
public class MiBean implements SessionBean {
   SessionContext ctx = null;

   public void setSessionContext(SessionContext ctx) {
      this.ctx = ctx;
   }

   ...

   public void miMetodo() {
      System.out.println(ctx.getCallerPrincipal().getName());
      if (ctx.isCallerInRole("administrador")) {
         //código ejecutado por administradores
         System.out.println("Me llama un administrador");
      }
      if (ctx.isCallerInRole("bibliotecario")){
         //código ejecutado por bibliotecarios
         System.out.println("Me llama un bibliotecario");
      }
   }
   ...
```

6. Anotaciones EJBGen Weblogic para seguridad

6.1. Definición de roles

El atributo `roles` de las anotaciones `@LocalMethod()` y `@RemoteMethod` permite definir una lista de roles separados por comas que están autorizados a invocar el método.

Por ejemplo, el siguiente método sólo está autorizado para los roles "admin" y "bibliotecario":

```
   @RemoteMethod(
```

```
      roles = "admin, bibliotecario")
   public void addUsuario(String login) {
      // código
  }
```

6.2. Asignación de usuarios a roles

Para definir los usuarios o grupos asociados a los roles hay que utilizar la anotación `@RoleMappings` en la clase del bean que define un array de elementos `@RoleMapping`. Cada `@RoleMapping` define una lista de usuarios o grupos y un nombre de rol. A continuación vemos el mismo ejemplo que comentamos previamente, utilizando anotaciones:

```
@RoleMappings(
      {@RoleMapping(principals = "weblogic, admin", roleName =
"administrador"),
       @RoleMapping(principals = "dgl, admin, Bibliotecario", roleName
= "bibliotecario")})
public class UsuarioBean extends GenericSessionBean implements
SessionBean {
   ...
```