Transacciones y Seguridad en J2EE

Pablo Basanta Val
Florina Almenares Mendoza
Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid
{pbasanta,florina}@it.uc3m.es

Agradecimientos a Cohen (IBM Haifa)

Objetivos didácticos

- Comprender cómo se gestionan las transacciones en J2EE
  - Distinguir el modelo programático del declarativo
  - Conocer los interfaces tanto declarativas como programáticas y como éstas se integran en el modelo de interfaces
- Comprender cómo funciona la seguridad en aplicaciones J2EE
  - Entender cuál es el modelo que soporta J2EE
    - Autentificación en la web, en los clientes IIOP y en los EIS
  - Configuración declarativa y programática de aplicaciones

Índice (1/2)

Índice (2/2)

Índice (2/2)

Transacciones

- Una transacción es:
  - una unidad de trabajo “ACID”, o
  - un conjunto de tareas que deben ser realizadas juntas y de forma atómica
    - si una o más tarea **falla**: “***roll-back***”
    - si todas las tareas son **exitosas**: “***commit***”

- La arquitectura EJB proporciona dos clases de transacciones
  - **Gestionadas por el contenedor** ≡ Transacciones declarativas
    - CMT, *Container-Managed Transaction*
  - **Gestionadas explícitamente** ≡ Transacciones prográmaticas
    - BMT, *Bean-Managed Transaction*
    - Iniciadas por el cliente

Características de las transacciones (ACID)

- A: Atomicidad
  - Que se realice o por completo o que no se dé ningún paso.
- C: Consistencia
  - Que la transacción deje la base de datos en un estado consistente, aunque falle la transacción.
- I : Aislamiento
  - Que los cambios realizados por una transacción no se vean entorpecidos por otra transacción.
  - Normalmente, hay varios niveles de aislamiento.
- D: Durabilidad
  - Que tras la finalización de una transacción los datos se encuentren almacenados adecuadamente en la base de datos.

Transacciones Gestionadas por el contenedor

- Las transacciones se gestionan automáticamente
  - usando ***Java Transaction Service*** **(JTS)** API
- **Atributos transaccionales** se declaran en el descriptor de despliegue
  - son asociados con cada método del EJB
- Fácil de usar
- El comportamiento transaccional es independiente de la lógica de negocio
- El contenedor maneja
  - el comienzo y el fin de la transacción
  - la interacción con la base de datos
  - la creación y propagación del contexto durante la vida de la transacción

Transacciones Gestionadas explícitamente

- El desarrollador gestiona las transacciones en el código de la aplicación (cliente o EJB)
  - explícitamente comienza la transacción
  - completa o anula la transacción
  - más control
- Uso explícito de ***Java Transaction API*** (JTA)
- Uso de JDBC (p. ej. bean de sesión "*wrapping*" código legado)
- Más difícil de usar que CMT

APIs de Java

- ***Java Transaction Service*** (JTS)
  - conjunto de APIs de bajo nivel
  - es usado por los **desarrolladores de gestores de transacciones**, servidores de aplicación, contenedores EJB, etc.
  - no usado por los desarrolladores de aplicaciones
  - el contenedor garantiza integridad transaccional a las aplicaciones
- ***Java Transaction API*** (JTA)
  - usado por los **desarrolladores de aplicaciones**
  - especifica la interfaz entre el gestor de transacción y todos los objetos involucrados
  - Interfaz principal: `UserTransaction`

Transacción gestionada por contenedor

```
<assembly-descriptor>
 ...
 <container-transaction>
   <method>
     <ejb-name>TravelAgentEJB</ejb-name>
     <method-name> * </method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
 </container-transaction>
 <container-transaction>
   <method>
      <ejb-name>TravelAgentEJB</ejb-name>
      <method-name>listAvailableCabins</method-name>
   </method>
   <trans-attribute>Supports</trans-attribute>
 </container-transaction>
...
</assembly-descriptor>
```

Fuente:
Enterprise JavaBeans, Fourth Edition
By Richard Monson-Haefel (Author), Bill Burke (Author), Sacha Labourey (Author) **Publisher:** O'Reilly

Atributos transaccionales

- Cuando se usa CMT, los beans (o métodos específicos) tienen un atributo transaccional
  - métodos `create` y `remove` de
    - un **bean de entidad** tienen atributos transaccionales
    - un **bean de sesión** NO tienen atributos transaccionales

- Seis posibles valores:

```
1.Required
2.RequiresNew
3.NotSupported
4.Supports
5.Mandatory
6.Never
```

Atributos : `Required` y `RequiresNew`

- **`Required`** (valor por defecto):
  - si el invocador (cliente, o un método diferente) tiene un contexto transaccional, éste es propagado al bean
  - si no, el contenedor crea un nuevo contexto transaccional
  - el método siempre se ejecuta dentro de un contexto transaccional,
  - **uso:** métodos que actualizan bases de datos u otros gestores de recursos que soportan transacciones
- **`RequiresNew`**:
  - si el invocador tiene un contexto transaccional, éste es suspendido durante la ejecución del método
  - en cualquier caso, se crea un **nuevo** contexto transaccional
  - **uso:** el método debe ejecutarse dentro de una transacción
    - pero es "independiente" de la salida de la transacción del invocador
    - p. ej., una aplicación de seguimiento

Atributos: `Supports` y `NotSupported`

- **`Supports`**:
  - si el invocador tiene un contexto transaccional, es propagado al bean
  - si no, ningún contexto transaccional es usado
  - **uso:** el método no requiere totalmente una transacción
    - p. ej., métodos que ejecutan una sola operación de actualización
- **`NotSupported`**:
  - si el invocador tiene un contexto transaccional, éste es suspendido durante la ejecución del método
  - si no, ningún contexto transaccional es usado
  - en cualquier caso, el método se ejecuta sin contexto transaccional
  - **uso:** cuando se usan gestores de recursos que no propagan la transacción

Atributos: `Mandatory` y `Never`

- **`Mandatory`**:
  - si el invocador tiene un contexto transaccional, éste es propagado al bean
  - si no, una excepción es lanzada
    `TransactionRequiredException`
    `TransactionRequiredLocalException`
  - **uso:** cuando el invocador debe proporcionar la transacción
    - no necesariamente implica BMT o transacciones gestionadas por el cliente: el invocador puede ser un método diferente en el mismo EJB
- **`Never`**:
  - si el invocador tiene un contexto transaccional, una excepción es lanzada
    - `RemoteException` o `EJBException`
  - si no, el método se sigue normalmente, se ejecuta sin un contexto
  - **uso:** recursos no transaccionales

Asignación de transacciones

Cliente	Atributo transaccional	Asociado al método de negocio
Ninguno Recibido con la petición	`Required`	Creado por el contenedor Recibido con la petición
Ninguno Recibido con la petición	`RequiresNew`	Creado por el contenedor Creado por el contenedor
Ninguno Recibido con la petición	`Supports`	Ninguno Recibido con la petición
Ninguno Recibido con la petición	`NotSupported`	Ninguno Ninguno
Ninguno Recibido con la petición	`Mandatory`	Error Recibido con la petición
Ninguno Recibido con la petición	`Never`	Ninguno Error

Recomendaciones EJB 2.0

- Se recomienda que los beans de entidad únicamente utilicen
  - Transacciones gestionadas por contenedor (CMT)
  - `Required`, `RequiresNew`, o `Mandatory`
- Beans dirigidos a mensajes pueden declarar
  - `NotSupported` o `Required`
- `Supports`, `NotSupported` y `Never` no son implementados por todos los contenedores
- Suspender una transacción
  - si un método de negocio desea suspender una transacción
    - debe invocar `setRollBackOnly` sobre su objeto de contexto EJB
  - se suele hacer antes de lanza una excepción de aplicación
- Utilice `getRollBackOnly` para descubrir si se está "running on empty"

Transacciones gestionadas explícitamente

- Esta característica está disponible para EJBs de **sesión** y **message-driven** únicamente
- Un método de negocio puede usar JTA
  - para crear un nuevo objeto `UserTransaction` (inicia una transacción), y la usa como su propio contexto transaccional
    - `begin`, `commit`, `rollback`, `setRollbackOnly`
- Este método será propagado cuando se invocan métodos sobre otros EJBs
  - dependiendo de las configuraciones de los métodos invocados
- El bean que crea la transacción es responsable para completarla/anularla
  - en los beans de sesión sin estado, el método que inicia la transacción debe terminarla
- Una instancia que comienza una transacción debe completarla antes de comenzar una nueva

Interfaz para transacciones programáticas

javax.transaction.UserTransaction	
void	**begin()** //Crea una nueva transacción
void	**commit()** //Finaliza una transacción
int	**getStatus()**
void	**rollback()** //Cancela una transacción
void	**setRollbackOnly()** //Marca una transacción para ser cancelada
void	**setTransactionTimeout(int seconds)** //Fija el tiempo máximo para que se complete

API de transacciones:
http://java.sun.com/products/jta/jta-1_0_1B-doc/

Ejemplo de una transacción programática iniciada por un cliente

```
[...]
InitialContext cxt = new InitialContext();
userTx = (javax.transaction.UserTransaction)
          cxt.lookup("java:comp/UserTransaction");
try{
    userTx.begin();

    beanA.setX(1);
    beanA.setName("uno");
    beanB.setY(2);
    beanB.setName("dos");

    userTx.commit();
} catch(Exception e){
   try{
     System.out.println(e.getMessage());
     userTx.rollback();
   } catch(Exception ex){}
}
[...]
```


Fuente:
Enterprise JavaBeans, Fourth Edition
By Richard Monson-Haefel (Author), Bill Burke (Author), Sacha Labourey (Author) **Publisher:** O'Reilly

Interfaz `SessionSynchronization`

- EJBs de sesión con estado que usan CMTs (únicamente) pueden recibir notificaciones de eventos de transacción
  - permite al bean mantener una caché sincronizada con la BD
- Para hacer esto, el bean debe simplemente implementar `SessionSynchronization`
- Esta interfaz tiene tres métodos:
  - `afterBegin`
  - `beforeCompletion`
    - la oportunidad para suspender la transacción llamando `setRollbackOnly` sobre el contexto de sesión
  - `afterCompletion(boolean committed)`
    - el parámetro indica “commit” (`true`) o “rollback”

Excepciones y transacciones

- Las transacciones son automáticamente anuladas si una excepción de sistema es lanzada
  - excepción de sistema: `RuntimeException`, `EJBException`, …
  - el contenedor registrará el error y marcará la transacción como "rollback only"
  - cualquier intento para seguir con la transacción sería inútil
- Una **excepción de aplicación no suspende** automáticamente la transacción
  - si el cliente invoca m_1, m_1 inicia la transacción e invoca m_2, y m_2 lanza una excepción de aplicación, entonces m_1 puede intentar continuar la transacción
  - si m_1 lanza (o no captura) una excepción de aplicación, la transacción será anulada

Seguridad

Visión general
Seguridad en componentes Web
Seguridad en clientes de consola
Seguridad en EJBS
Seguridad EIS

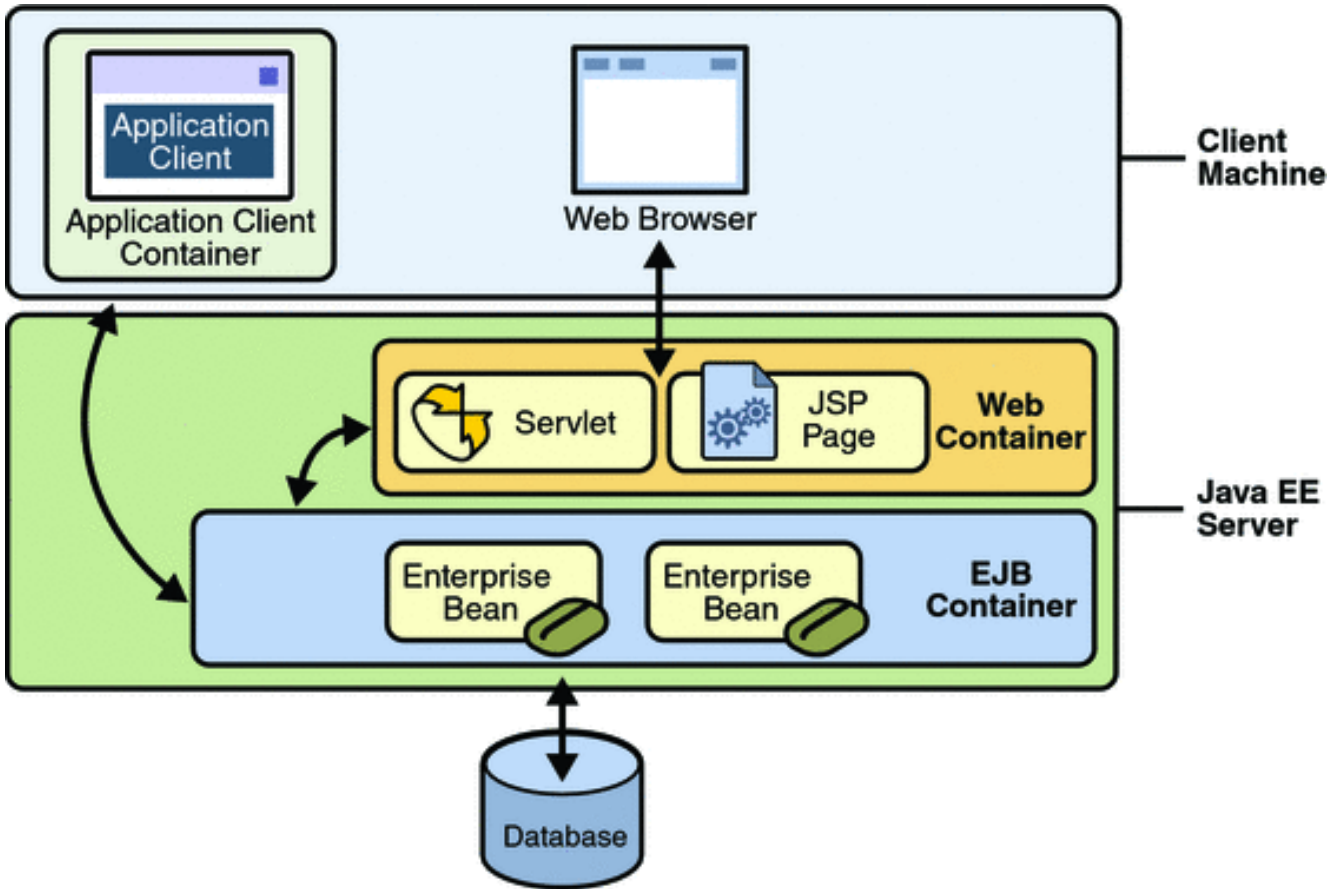
Pablo Basanta Val

pbasanta@it.uc3m.es

La seguridad dentro de J2EE

- Muchas aplicaciones requieren proveer la identidad de los usuarios que están accediéndolas
  - Por ejemplo un banco, aula global, …
- J2EE y los EJBs proveen un modelo de seguridad híbrido (tanto programático como declarativo) basado en:
  - Autentificación (verificación de credenciales)
  - Autorización (con lo que puede interactuar un usuario)
  - Confidencialidad y integridad (soporte contra ataques, como por ejemplo SSL)
- En la arquitectura de J2EE se reparte la responsabilidad de la seguridad entre sus diferentes roles.
  - Administrador del sistema, proveedor del contenedor, el desarrollador de aplicaciones y el encargado de despliegue

Arquitectura J2EE



Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Autenticación

- Es el proceso de verificar si alguien es quien dice ser.
  - Típicamente mediante login y password
  - La clave puede estar en muchos formatos
  - Una vez autenticado, al usuario se le asigna un rol
- Los EJBs no especifican cómo se hace este proceso, es responsabilidad del servidor de aplicaciones
  - Se puede hacer con JNDI (por ejemplo contra LDAP de la universidad) o por ejemplo en Jboss se usa JAAS (Java Autentication and Autorización Service) o realms

Autorización

- Una vez autenticado el componente, se procede por un mecanismo de autorización, basado en roles
- En J2EE las políticas se aplican sobre roles y no sobre usuarios

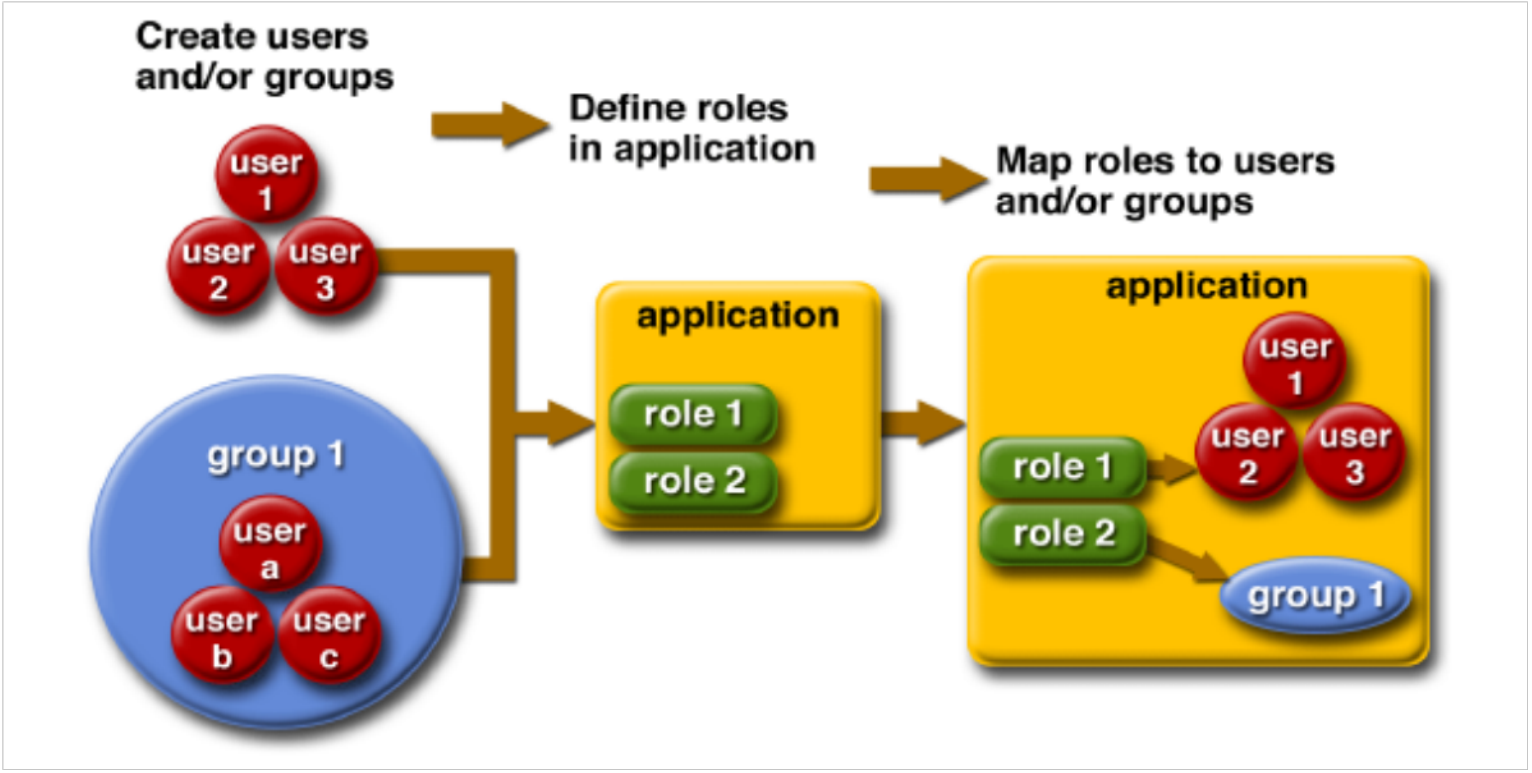
Reinos (Realms)

- También conocidos como *security policy domains* or security *domains*.
- Son ámbitos sobre los cuales se definen entidades de seguridad similares.
- El AppServer de Sun soporta:
  - File, Ldap, Certificate y Solaris
- Su configuración es dependiente de la plataforma
  - En el AppServer se puede utilizar la interfaz web a través del puerto 4848

Usuario, grupo y rol

- Usuario: identidad de un usuario
  - Ejemplo: Pablo Basanta
- Grupo: entidad colectiva de un grupo de usuarios
  - Ejemplo: Profesor de la asignatura de J2EE
- Rol: Desde el punto de vista lógico de la aplicación es un perfil
  - Ejemplo: profesor

Mapeo de roles a usuarios y/o grupos



Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Asignación de usuarios a grupos

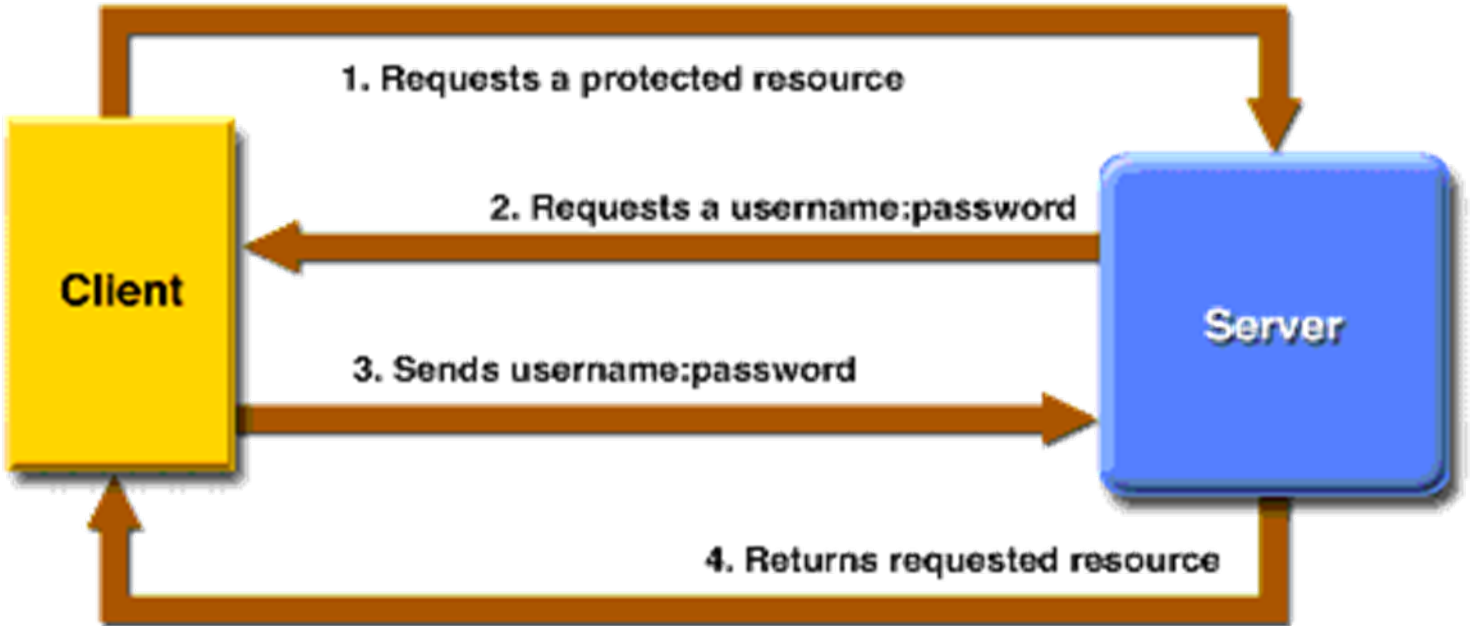
```
<sun-ejb-jar>
. . .
  <security-role-mapping>
    <role-name> AUTHORIZED_MERCHANT </role-name>
    <principal-name>JuanPalomo</principal-name>
    <group-name>TravelAgent</group-name>
  </security-role-mapping>
. . .
</sun-ejb-jar>
```

Fuente:
Enterprise JavaBeans, Fourth Edition
By Richard Monson-Haefel (Author), Bill Burke (Author), Sacha Labourey (Author) **Publisher:** O'Reilly

Mecanismos de autenticación

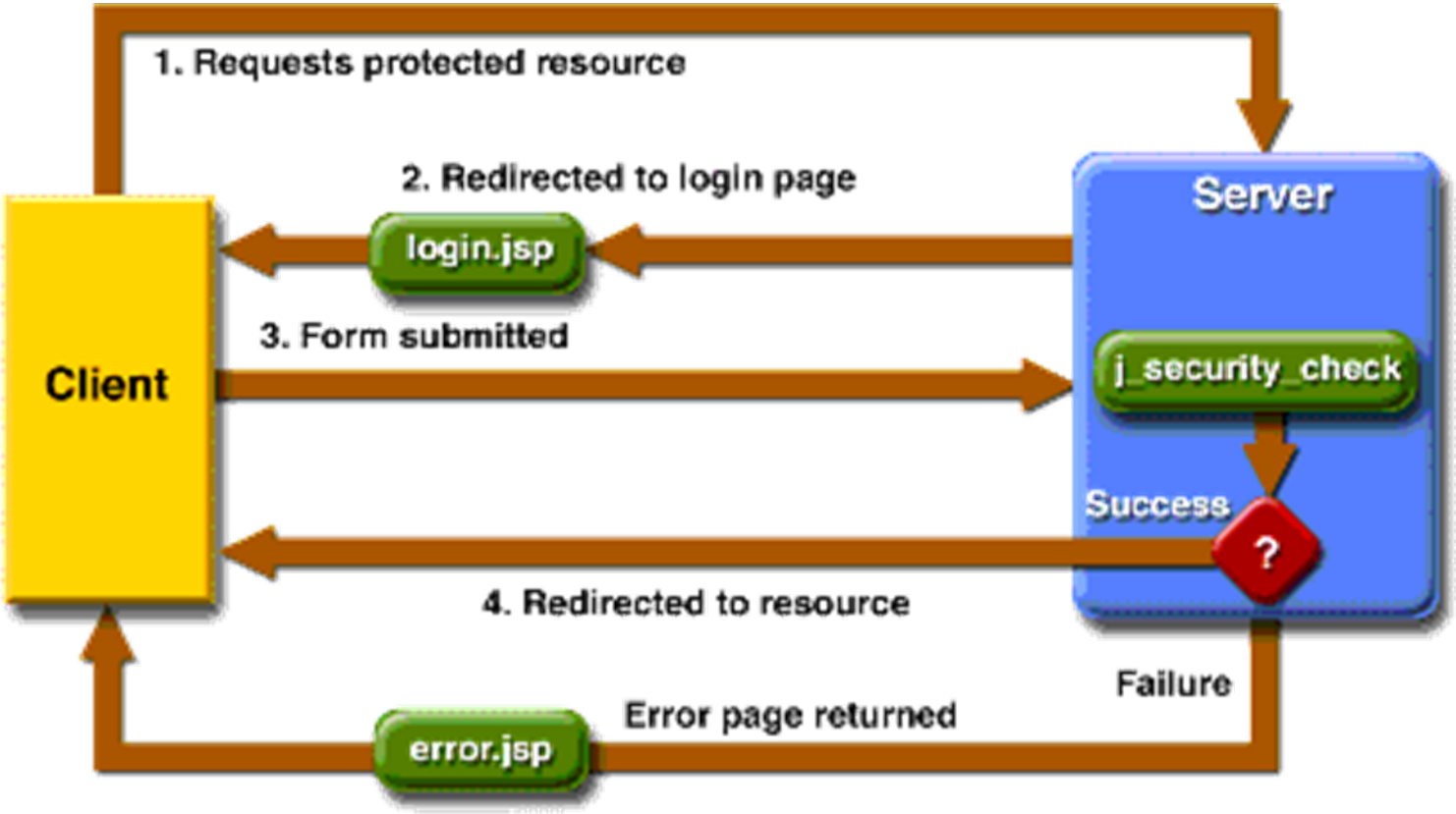
- Los clientes (web o appclients) se autentican antes de poder entrar
- En aplicaciones web se soportan varios tipos de autenticaciones
  - **HTTP basic authentication**
  - **Form-based login**
  - **Client certificate**
  - **Mutual authentication**
  - **Digest authentication**
- Mientras que para clientes de consola se suelen preferir otros mecanismos
  - **JAAS** (*Java Autorization and Autentication System*)

HTTP Basic



Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Autenticación basada en formulario

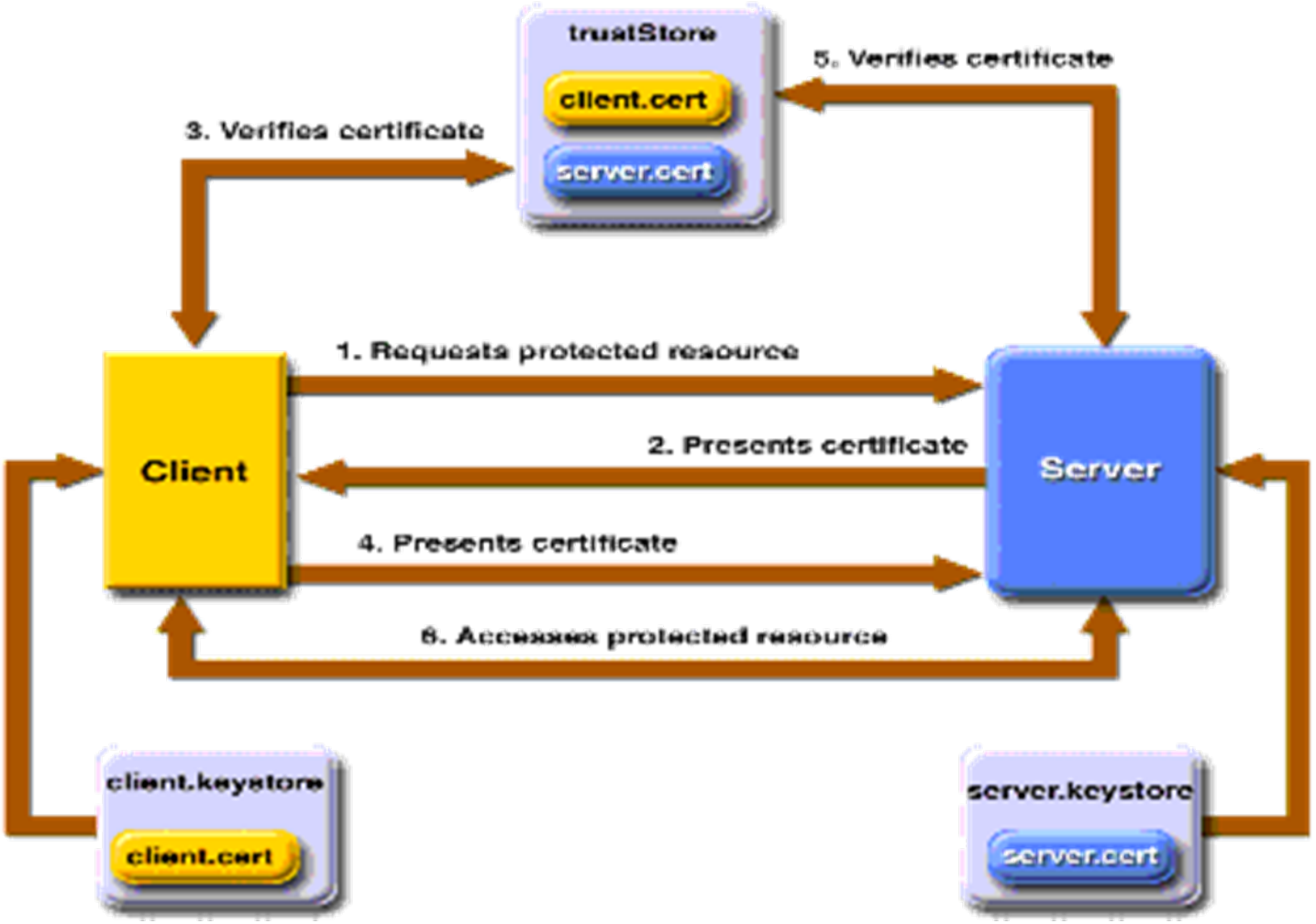


Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Basada en certificado de cliente

- Utiliza HTTP sobre SSL (Secure Server Layer), por lo que es bastante segura
- SSL ofrece tanto encriptación, como autenticación del servidor, como integridad de mensaje como identificación opcional del cliente (opcional) para conexiones TCP/IP
- En caso de hacerse la autenticación del cliente se utiliza un certificado X.509
- Antes de usar esta configuración se ha de habilitar dicho tipo de seguridad en el servidor

Autentificación mutua



Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Digest

- Es bastante parecida a la http basic pero más segura.
  - HTTP-basic utiliza 64-base encoding
- Pero sin embargo no es muy utilizad, aunque J2EE la mantiene por universalidad

Ejemplo de autenticación con formulario (1/2)

```
<body>
  <form action="j_security_check">
  Enter your user name: <input type="text" name="j_username"/>

  Enter your password: <input type="text" name="j_password"/>
 </form>
```

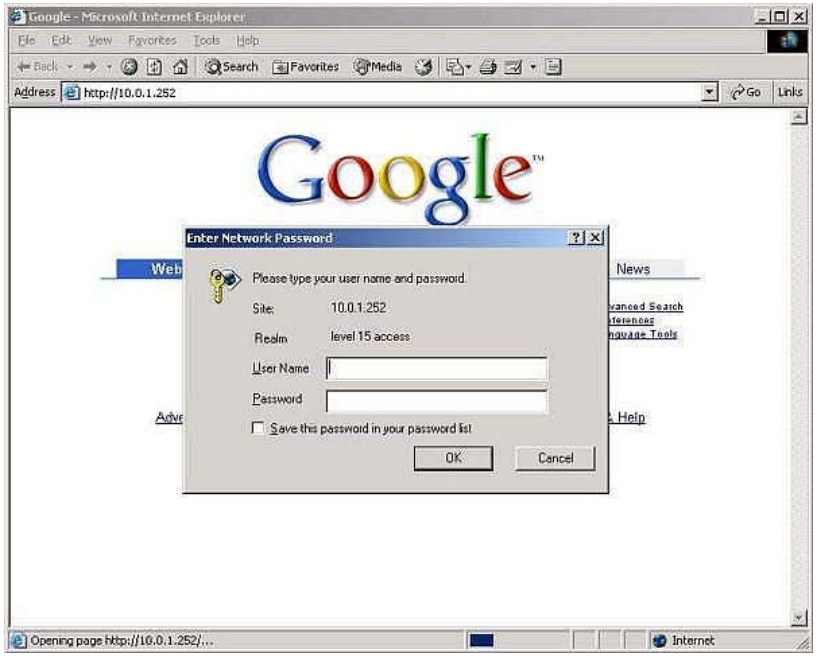
Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Ejemplo de autenticación con formulario (2/2)

```
<login-config>
  <auth-method>FORM</auth-method>
    <form-login-config>
       <form-login-page>login.jsp</form-login-page>
       <form-error-page>login-invalid.jsp</form-error-page>
  </form-login-config>
</login-config>
```

Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Ejemplo de configuración HTTP-basic (1/2)



Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Ejemplo de configuración HTTP-basic (1/2)

```
<login-config>
   <auth-method>BASIC</auth-method>
   <realm-name>MyRealm</realm-name>
</login-config>
```

Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Asignación de restricciones a componentes web

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/private <url-pattern>
    <http-method>PUT</http-method>
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>role1</role-name>
    <role-name>employee</role-name>
 </auth-constraint>
</security-constraint>
```

Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Seguridad (extra) en la autenticación

- Tanto la autenticación básica http como la basada en formulario carecen de seguridad y las claves enviadas pueden ser interceptadas por otros clientes
- Se puede hacer que esta autentificaciones se cifren con SSL haciendo que sean confidenciales o integrales
  - CONFIDENCIAL= nadie leer la comunicación
  - INTEGRAL= nadie puede alterarlo

Autenticación de clientes

- Una primera alternativa basada en JAAS
  - El cliente implementa un clase especial javax.security.auth.callback.CallbackHandler
- Una segunda alternativa basada en login programático
  - Solo para clientes EJB en Appserv de Sun (uso restringido) com.sun.appserv.security.Programmatic

Fragmento de autenticación de cliente con JAAS

```
LoginContext ctx = null;
//Creación de contexto
try {
ctx = new LoginContext("WeatherLogin",
                       new   MyCallbackHandler());
} catch(LoginException le) { System.exit(-1);
} catch(SecurityException se) { System.err.println(“Error en
el contexto "}
//Login
try {
ctx.login(); } catch(LoginException le) {
    System.out.println("Authentication failed. "}
System.out.println(“Ya estás autenticado");
```

Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Seguridad (extra) en los clientes

- Se puede configurar a IIOP para que utilice características avanzadas (seguridad, confidencialidad, confianza), relacionadas con las seguridad de los clientes

```
<ior-security-config>
<transport-config>
     <integrity>NONE</integrity>
     <confidentiality>NONE</confidentiality>
     <establish-trust-in-target> NONE </establish-trust-in-target>
     <establish-trust-in-client> NONE </establish-trust-in-client>
</transport-config>
</ior-security-config>
```

Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Seguridad programática (dentro del componente)

- En el API consta de dos métodos:
  - `getCallerPrincipal`, el cual permite obtener la identidad del invocante
  - `isCallerInRole`, que permite asociar una sesión a un cliente

Method Summary	
java.security.Principal	**getCallerPrincipal()** Obtain the java.security.Principal that identifies the caller.
boolean	**isCallerInRole(java.lang.String roleName)** Test if the caller has a given security role.

http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/ejb/EJBContext.html

Ejemplo de seguridad programática

```
[…]
InitialContext ctx =new InitialContext();
Principal caller= ctx.getCallerPrincipal();
String travelAget =caller.getName();

if ctx.isCallerInRole(“JUNIOR_TRAVEL_AGENT”)
   throw new IllegalRoleh();

[…]
```

Fuente:
Java EE Tutorial 1.4 , Fourth Edition

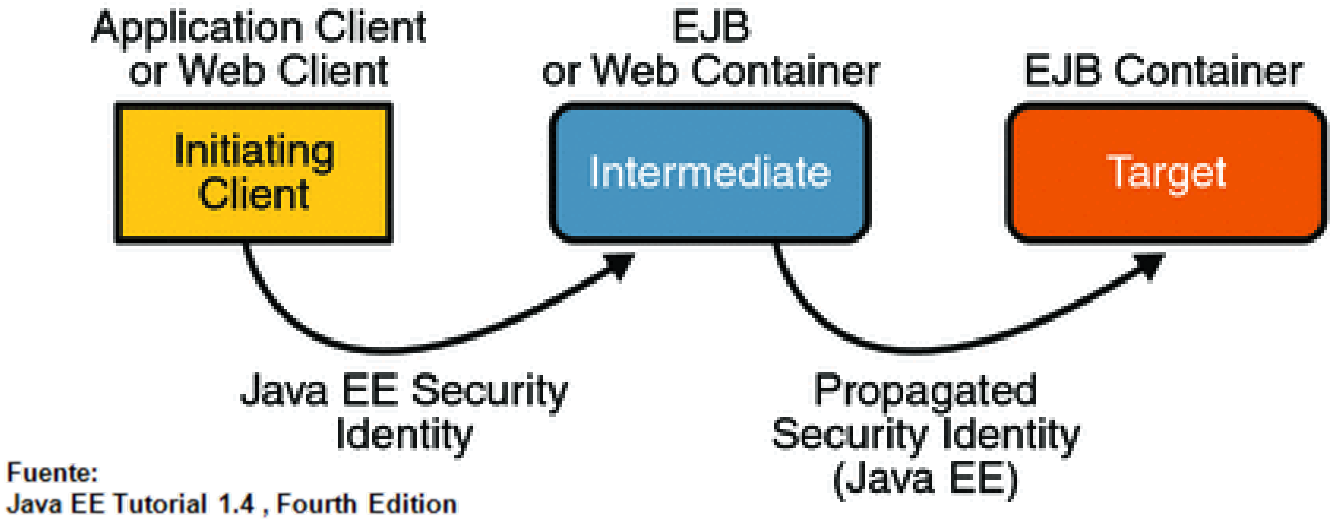
Seguridad declarativa en EJBs

- En los descriptores xml se pueden definir restricciones en los accesos (en función de los roles)

```
<ejb-jar version=2.0>
 <assembly-descriptor>
  <security-role>
   <description/>
   <role-name>AUTHORIZED_MERCHANT</role-name>
  <security-role>
 </method-permission>
   <role-name>AUTHORIZED_MERCHANT</role-name>
   <method>    <ejb-name>ProcessPaymetnBean</ejb-name>
               <method-name>byCredit</method-name>
   </method>
  <method-permission>
. . .
```

Fuente:
Java EE Tutorial 1.4 , Fourth Edition

Propagación de roles



Fuente:
Java EE Tutorial 1.4 , Fourth Edition

- Por defecto, la identidad se propaga dentro de entornos seguros (por ejemplo de la aplicación web al contenedor de EJBs). Esto sucede cuando hay confianza en el otro extremo.
- Se puede cambiar esta identidad con el elemento `<run-as>`.

Seguridad EIS

- La capa EIS puede requerir una conexión (con seguridad) a un elemento
  - Por ejemplo a una base de datos
- J2EE ofrece dos alternativas:
  - **Container-Managed Sign-On**
    - **Donde el programador es responsable de enviar datos al servidor sobre la atentificación**
  - **Component-Managed Sign-On**
    - **Donde el programador es responsable de enviar datos al servidor sobre la atentificación**

Cuestiones para reflexionar

Bloque I y II: Transacciones

- ¿Porqué cree que J2EE ofrece un doble soporte, basado en un modelo declarativo y otro programático para trabajar con transacciones?
- ¿Describa un caso en el llamar a un método transaccional etiquetado como **requiresNew** surta el mismo efecto que llamar a uno **required**?
- ¿Qué son los atributos transaccionales de J2EE?¿Cuantos hay? Enumérelos
- ¿Cuál es la diferencia existente entre JTS y JTA?

Bloque III, IV: Seguridad

- Discuta sobre cual puede haber sido el motivo que ha empujado a los diseñadores de J2EE a incluir dentro de sus modelo soporte directo para la seguridad.
- ¿Qué es un *Realm*?
- ¿En qué se diferencian autorización y autenticación?
- ¿Cómo se llama el principal mecanismo de autenticación utilizado para clientes de consola?
- ¿Qué significa JAAS?