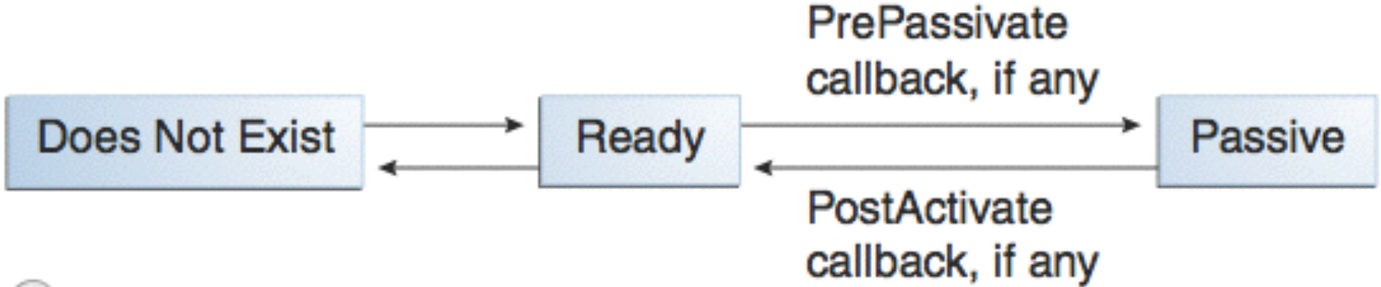
Componentes Enterprise JavaBeans

Sesión 2 - Ciclo de vida, seguridad

Índice

Ciclo de vida de los beans con estado

1. Create
2. Dependency injection, if any
3. PostConstruct callback, if any
4. Init method, or ejbCreate<METHOD>, if any



1. Remove
2. PreDestroy callback, if any

Ciclo de vida de los beans con estado

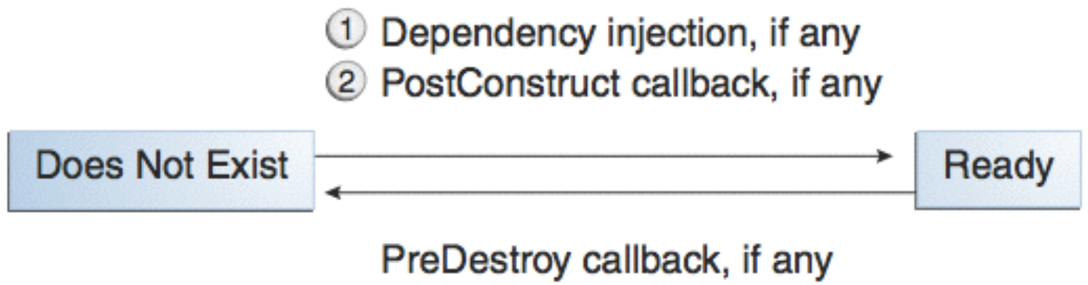
1. El contenedor crea una instancia usando el constructor por defecto cuando comienza una nueva sesión con un cliente.
2. Después de que el constructor se ha completado, el contenedor inyecta los recursos tales como contextos JPA, fuentes de datos y otros beans.
3. La instancia se almacena en memoria, esperando la invocación de alguno de sus métodos.
4. El cliente ejecuta un método de negocio y el contenedor lo invoca en el bean
5. Espera hasta que los siguientes métodos son invocados y los ejecuta.
6. Si el cliente permanece ocioso por un periodo de tiempo, el contenedor pasiva el bean, serializándolo y guardándolo en disco.
7. Si el cliente vuelve a invocar a un bean pasivado, éste se activa (el objeto es leído del disco) y se ejecuta el método
8. Si el cliente no invoca un método sobre el bean durante un cierto periodo de tiempo, el bean es destruido.
9. Si el cliente invoca algún método con el atributo @Remove el bean es destruido.

Anotaciones *callback* del ciclo de vida

- `@PostConstruct`: el método con esta anotación se invoca justo después de que se ha ejecutado el constructor por defecto y de que se han inyectado los recursos.
- `@PrePassivate`: invocado antes de que un bean sea pasivado.
- `@PostActivate`: invocado después de que el bean haya sido traído a memoria por el contenedor y antes de que se ejecute cualquier método de negocio invocado por el cliente
- `@PreDestroy`: invocado después de que haya expirado el tiempo de conexión o el cliente haya invocado a un método anotado con `@Remove`. Despés de invocar al método, la instancia del bean se elimina y se pasa al recolector de basura.

Ciclo de vida de los beans sin estado

- Soportan también las anotaciones `@PostConstruct` y `@PreDestroy`



Aspectos principales en la seguridad

- **Autentificación**: cómo se valida la identidad de los usuarios
- **Autorización**: cómo se evita que determinados usuarios accedan a determinados recursos/ servicios restringidos
- **Comunicación segura**: cómo se evita que los datos no se filtran por la red

Autorización en la capa EJB

- Utilizando EJB es posible definir una autorización a nivel de métodos de negocio
- Podemos restringir el acceso a los métodos de los beans, para que sólo puedan ser ejecutados por ciertos roles
- Ventajas evidentes:
  - Si alguien hackea la aplicación web y llama al método del bean sin haber sido autentificado o sin tener el rol necesario se lanza una excepción
- La aplicación está a prueba de posibles errores en la programación de la capa web

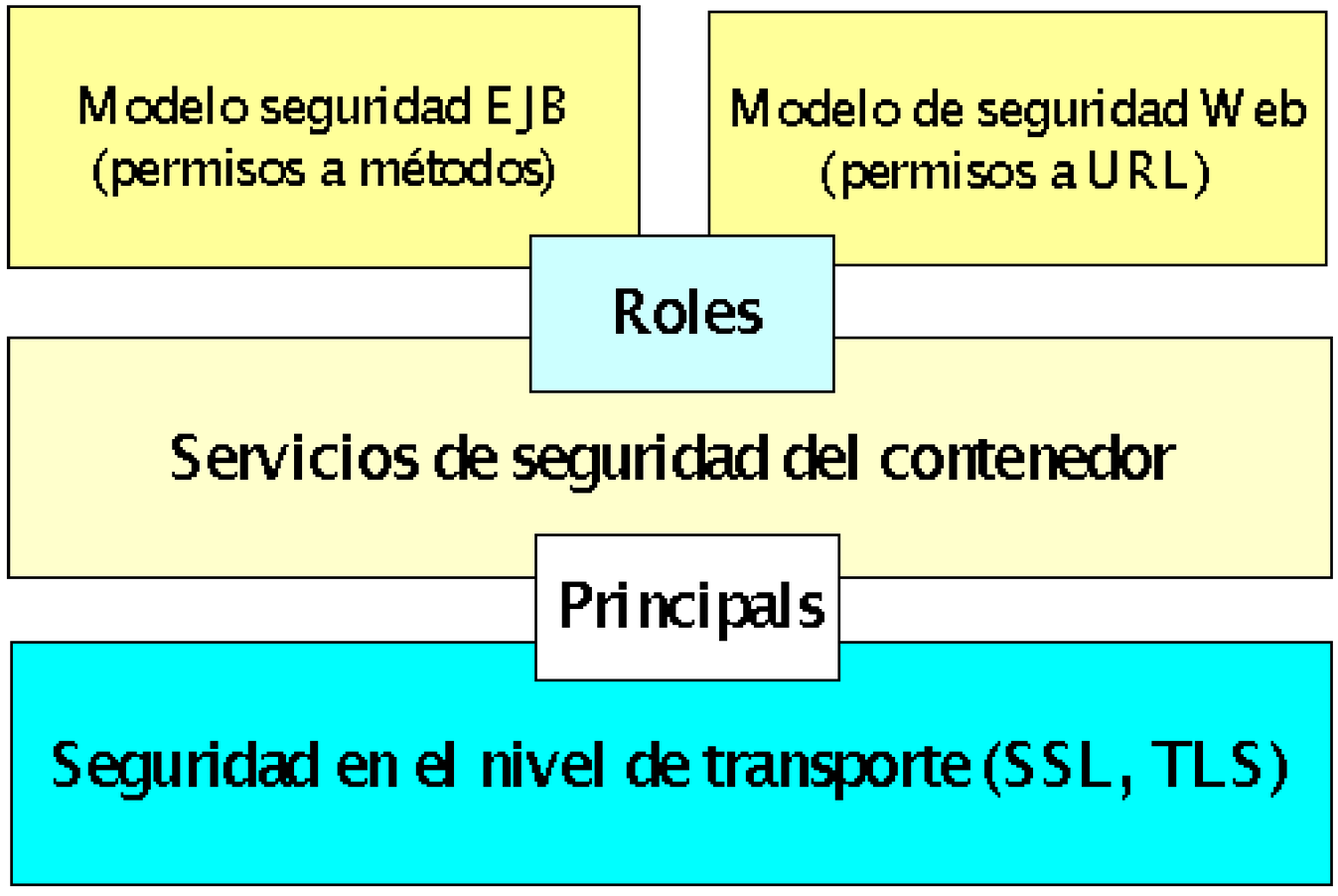
JAAS: Servicio Java de Autentificación y Autorización

- Existen diferencias entre los distintos entornos operativos en los que va a desplegarse una aplicación Java EE
- Distintas representaciones de las credenciales de los usuarios:
  - usuario/contraseña, certificados, kerberos
  - Servidor LDAP, base de datos, Sistema Operativo
- JAAS abstraer todas estas representaciones en una única API
- La autentificación del usuario se hace sólo una vez usando el API y obteniendo el objeto `principal` (usuario registrado)
- El objeto principal se pasa de forma transparente de una capa de la aplicación a otra

Conceptos fundamentales de autentificación

- La seguridad en Java EE se representa mediante:
  - Realms: conjunto de usuarios y roles. El servidor de aplicaciones permite configurar distintos tipos de realms (ficheros de configuración, bases de datos, LDAP, etc.)
  - Principals: usuarios (logins)
  - Roles: un usuario puede pertenecer a varios roles. Los permisos en la aplicación se definen en base a roles, no a usuarios.

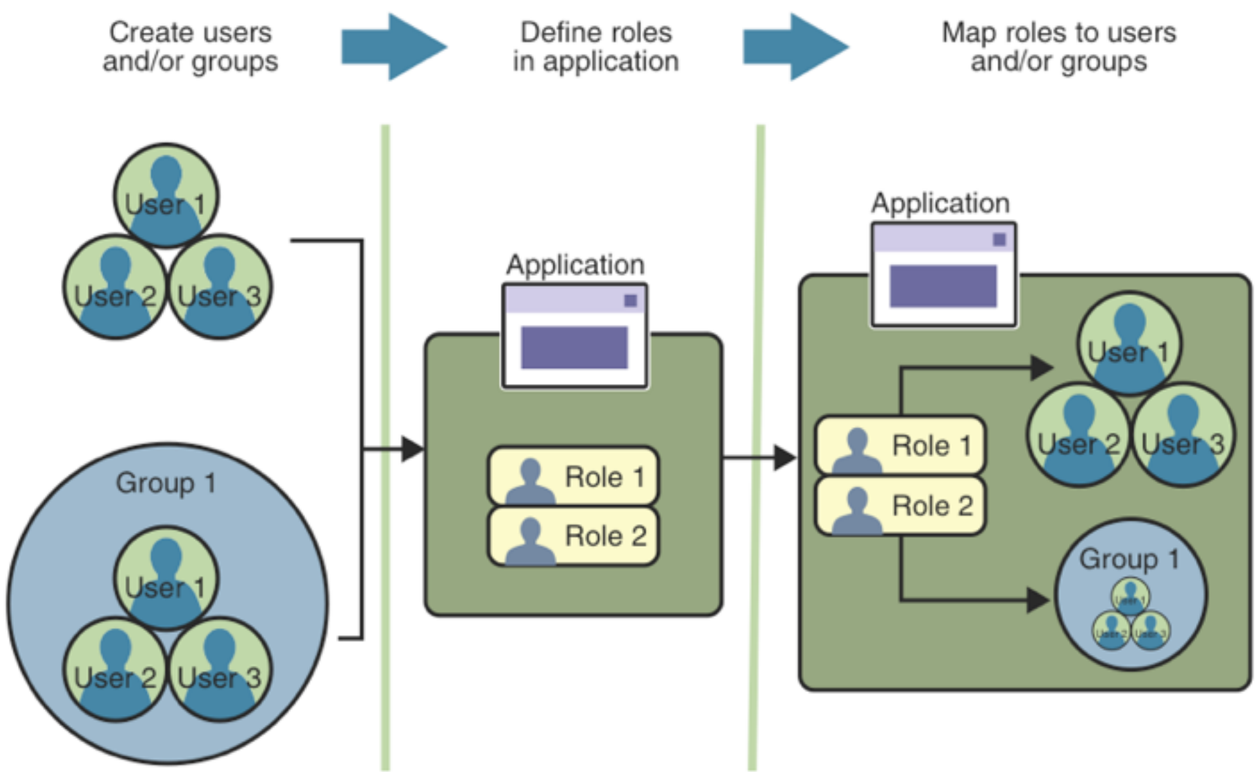
Arquitectura de seguridad en Java EE



Mapeado de roles

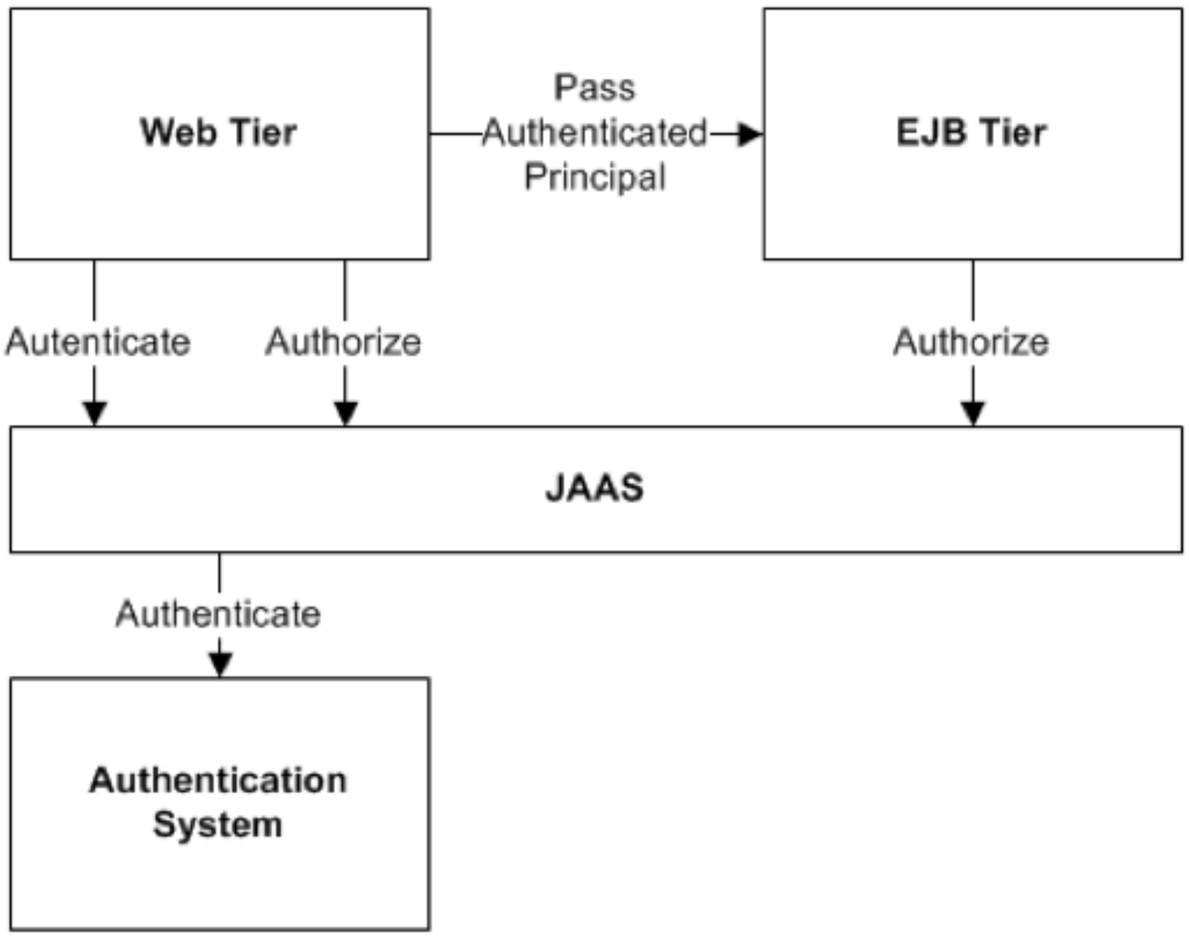
- Se puede diferenciar entre roles y grupos, para hacer más portable la aplicación
- Roles: aplicación
- Grupos: servidor de aplicaciones
- No lo vamos a hacer en los ejemplos
- Creamos los usuarios y los roles con el comando de WildFly

```
$ ./addUser.sh
```



Seguridad en capas web y EJB

- La capa web normalmente es responsable de realizar la autentificación y obtener el `Principal`
- El acceso a los métodos de la capa EJB se restringe a ciertos roles
- Los métodos devuelven una excepción si el `Principal` no tiene los roles necesarios



Ejemplo de autentificación en capa web (web.xml)

URL restringida

Rol permitido

Tipo de autentificación

Listado de todos los roles

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
         http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>holamundorestringido</web-resource-name>
            <url-pattern>/holamundorestringido</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>usuario-saludo</role-name>
        </auth-constraint>
    </security-constraint>
    <login-config>
        <auth-method>BASIC</auth-method>
    </login-config>
    <security-role>
        <role-name>usuario-saludo</role-name>
        <role-name>admin-saludo</role-name>
    </security-role>
</web-app>
```

Autorización en los beans

- Autorización declarativa con anotaciones en los beans
- Al comienzo del bean se deben declarar todos los roles que van a estar autorizados a acceder a algún método del bean
- Se puede restringir el acceso a todos los métodos o a métodos individuales

```
@Stateless
   @DeclareRoles({"Admin","Bibliotecario","Socio"})
   public class OperacionBOBean implements OperacionBOLocal {
   // ...

   @RolesAllowed("Admin")
   public String borraOperacion(String idOperacion) {
   // ...
   }

   @RolesAllowed({"Admin","Bibliotecario"})
   public String realizaReserva(String idUsuario, String idLibro) {
   // ...
   }

   @RolesAllowed({"Admin","Bibliotecario"})
   public String realizaPrestamo(String login, String isbn) {    }

   // ...

   @PermitAll
   public List<OperacionTO> listadoTodosLibros() {    }

}
```

Autorización programativa

- Se comprueba en el programa si el usuario que está ejecutando el código tiene un determinado rol
- El usuario se obtiene llamando al método `getCallerPrincipal` del contexto de la sesión EJB

```
@Stateless
public class MiBean implements SessionBean {

    @Resource
    SessionContext 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");
        }
    }
```

¿Preguntas?