La sécurité dans Java Enterprise Edition

CARA M2 Miage FA-FC

Jean-François Roos - bâtiment M3 extension - bureau 218 - Jean-Francois.Roos@univ-lille1.fr

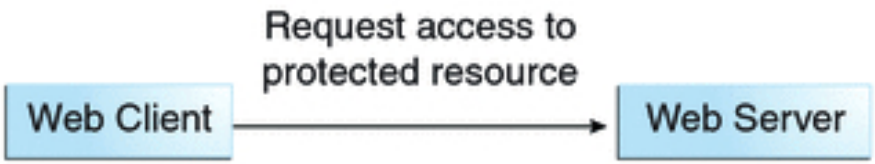
31 janvier 2016

Sommaire

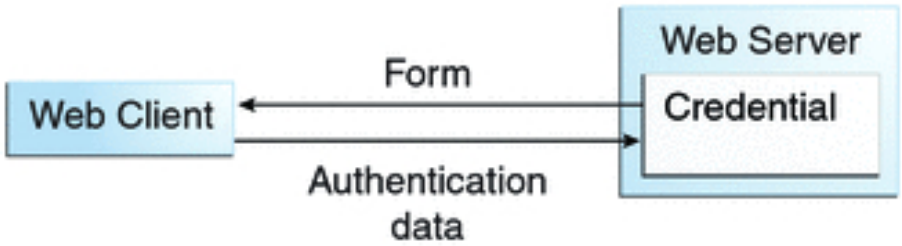
La sécurité en JAVA EE peut être exprimée :

- de manière déclarative : dans le descripteur de l'application ou par des annotations
- de manière programmée : embarqué dans le code de l'application.

Requête vers l'URL de l'application

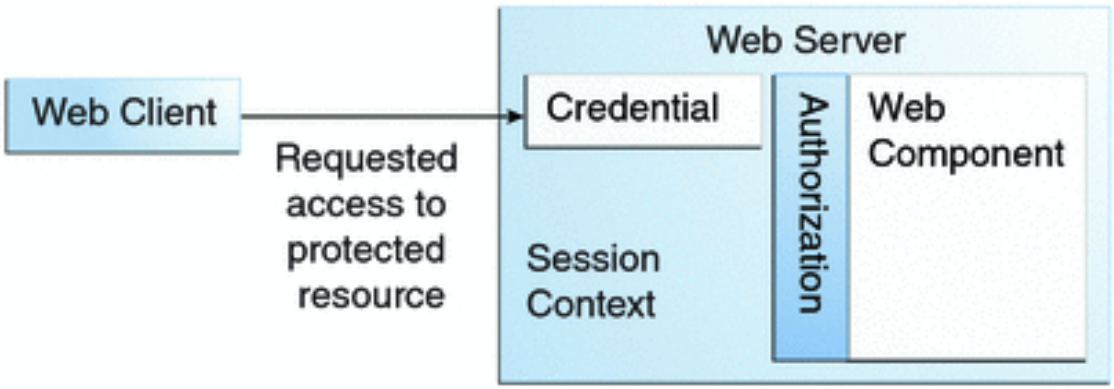


Renvoi d'un formulaire pour l'authentification



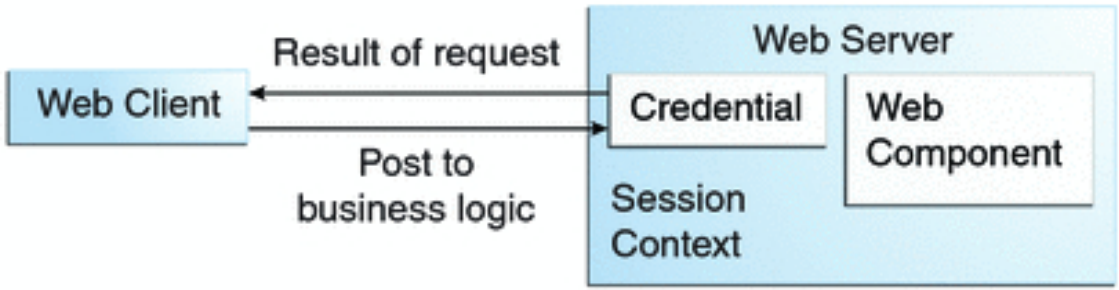
Le credential (accréditation)

utilisé pour déterminer si le client est dans l'un des rôles
définis dans la politique de sécurité autorisés à accéder à la ressource



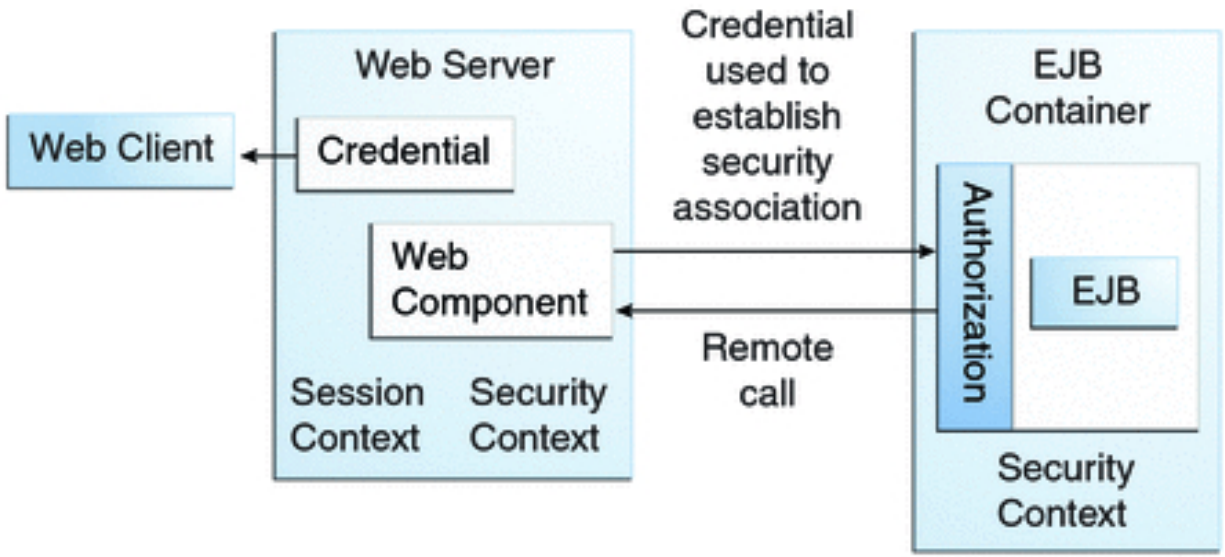
Réponse

le résultat de la requête est envoyé au client



Appel à un EJB

deux contextes de sécurtité, gérés par les containers



Technologies utilisées

- Java Authentication and Authorization Service (JAAS)
  api, framework
- Java Generic Security Services (Java GSS-API)
  échange de messages, api basée sur la possession d'un jeton (kerberos)
- Java Cryptography Extension (JCE)
- Java Secure Sockets Extension (JSSE)
  ssl, tls
- Simple Authentication and Security Layer (SASL) RFC 2222

Sécurité au niveau applicatif

les containers fournissent les mécanismes pour la sécurité au niveau applicatif, facile à implémenter, l'application peut la gérer finement

- avantages :
  - la sécurité est bien adaptée à l'application
  - elle peut être gérée vraiment finement
- inconvénients :
  - l'application est dépendante de paramètres qui ne sont pas transférables entre différentes applications
  - le support de plusieurs protocoles est compliqué

Sécurité au niveau transport

repose sut HTTPS et SSL, sécurité point à point : le client et le serveur s'accordent sur un algorithme, ils échangent une clef, clef publique cryptée et certificat d'autentification, l'échange est crypté de façon symétrique

- avantages :
  - la technologie est simple et bien connue
  - ça s'applique aux messages en entier
- inconvénients :
  - c'est très lié au protocole de transport
  - la protection est temporaire
  - elle protège en tout ou rien

Sécurité au niveau message

les informations de sécurité sont incluses dans les messages SOAP

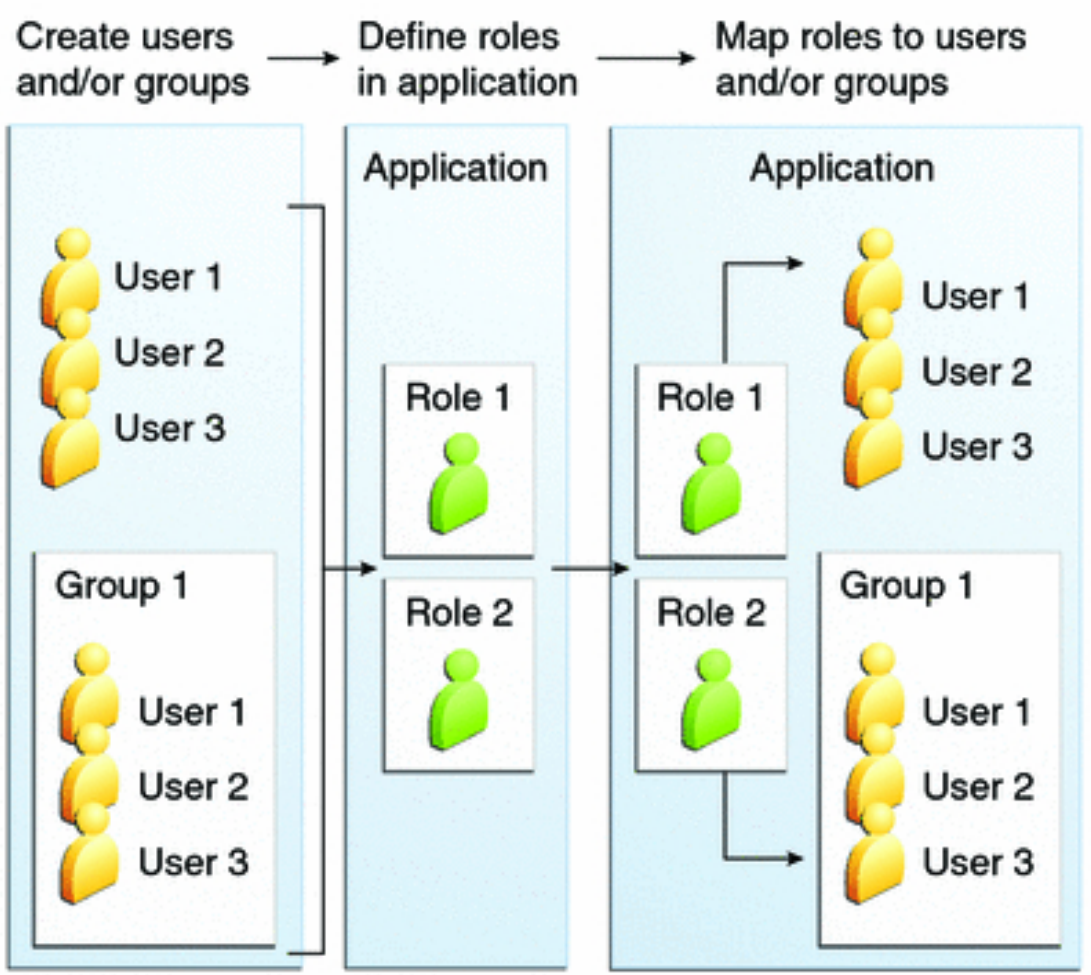
- avantages :
  - la sécurité reste avec le message pour tout son trajet et perdure après la réception
  - elle peut s'appliquer différemment suivant la partie du message
  - indépendante de l'environnement applicatif et du protocole de transport
- inconvénients :
  - c'est assez complexe à utiliser
  - ajoute un overhead conséquent

Les développeurs de l'application

- écrivent le code afin de récupérer le nom d'utilisateur et le mot de passe ;
- définissent comment établir la sécurité en utilisant des annotations ou le descripteur de déploiement.

L'administrateur du serveur d'application déclare les utilisateurs et les groupes dans le serveur d'application.

Le déployeur de l'application projette les rôles de sécurité de l'application sur les utilisateurs et les groupes déclarés dans le serveur.


Create users and/or groups
Define roles in application
Map roles to users and/or groups
User 1
User 2
User 3
Group 1
User 1
User 2
User 3
Application
Role 1
Role 2
Application
Role 1
Role 2
User 1
User 2
User 3
Group 1
User 1
User 2
User 3

Domaine de sécurité (Realm)

- Les serveurs sont partagés en un ensemble de domaines de protection.
- Chaque domaine possède son propre schéma d'authentification, contenant un ensemble d'utilisateurs et de groupes.
- Un realm est une base de données d'utilisateurs et de groupes pour une ou plusieurs applications et contrôlée par la même politique d'authentification.

Domaines préconfigurés dans GlassFish

- `file realm` : le serveur conserve les autorisations dans un fichier nommé `keyfile`. Il peut être modifié dans la console d'administration. Dans ce domaine de protection, le serveur utilise le fichier pour vérifier l'authentification. Il peut être utilisé pour tout sauf les clients web utilisant HTTPS et des certificats.
- `certificate realm` : le serveur enregistre les autorisations des utilisateurs dans une base de données de certificats. Il utilise HTTPS pour authentifier les clients web.
- `admin realm` : c'est aussi un domaine fichier. Le serveur enregistre les autorisations de l'administrateur dans le fichier `admin-keyfile`

Les rôles : composant JEE

- Définir un rôle : annotation `@DeclareRoles`
- Autoriser l'exécution de code par un rôle : annotation `@RolesAllowed`

```
@DeclareRoles({"DEPT-ADMIN", "DIRECTOR"})
@Stateless public class PayrollBean implements Payroll {
  @Resource SessionContext ctx;

  @RolesAllowed("DEPT-ADMIN")
  public void reviewEmployeeInfo(EmplInfo info) {
    oldInfo = ... read from database;
    // ...
  }
  @RolesAllowed("DIRECTOR")
  public void updateEmployeeInfo(EmplInfo info) {
    newInfo = ... update database;
    // ...
  }
  ...
}
```

Les rôles : servlet

on peut utiliser les annotations :

- `@ServletSecurity`
- `@HttpConstraint`

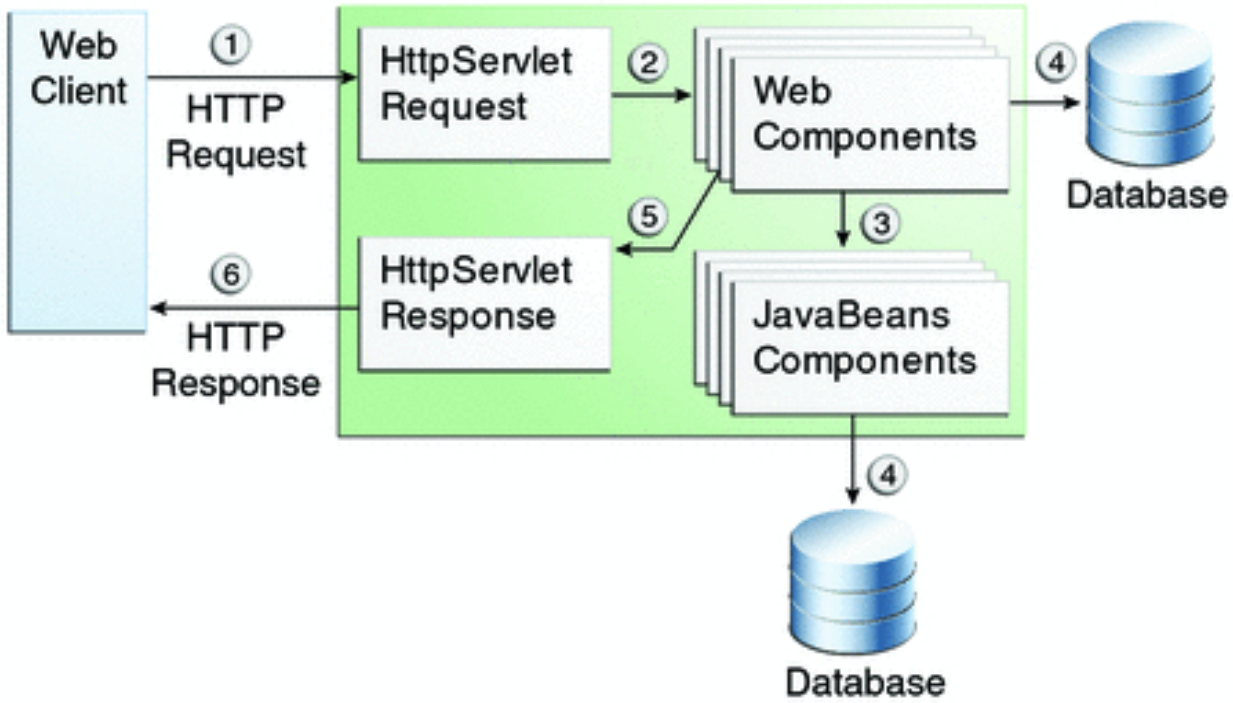
```
WebServlet(name="PayrollServlet", urlPatterns={"/payroll"})
@ServletSecurity(
   @HttpConstraint(transportGuarantee =
                       TransportGuarantee.CONFIDENTIAL,
                   rolesAllowed={"DEPT-ADMIN", "DIRECTOR"}))
public class GreetingServlet extends HttpServlet {
   ...
}
```

Projeter les rôles sur les groupes et utilisateurs

- si le nom des groupes correspond au nom des rôles : on peut activer la projection par défaut `principal-to-role` avec les outils d'administration du serveur.
- sinon il faut déclarer la projection dans le descripteur de déploiement (application, web ou ejb)
  ici descripteur de déploiement `web`, `glassfish-web.xml`

```
<glassfish-web-app>
  ...
  <security-role-mapping>
    <role-name>Mascot</role-name>
    <principal-name>Duke</principal-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>Admin</role-name>
    <group-name>Director</group-name>
  </security-role-mapping>
  ...
</glassfish-web-app>
```

Sommaire


Web Client
1
HTTP Request
HttpServlet Request
2
Web Components
4
Database
5
3
6
HTTP Response
HttpServlet Response
JavaBeans Components
4
Database

Contraintes de sécurité

utilisation de contraintes de sécurité pour définir les droits d'accès à un ensemble de ressources en utilisant leur url

- avec une servlet : on utilise `@HttpConstraint` et `@HttpMethodConstraint` dans une annotation `@ServletSecurity` pour spécifier une contrainte
- sans servlet, il faut définir un élément `security-constraint` dans le descripteur de déploiement. La méthode d'authentification ne pouvant être exprimée par annotation, il faut la définir dans le descripteur.

Les sous-éléments

- `web-resource-collection` : un ensemble de pattern d'url et de méthodes HTTP qui définissent un ensemble de ressources à protéger
  - `web-resource-name` : le nom de la ressource à protéger
  - `url-pattern` : liste les `URI` à protéger (ex /cart/*)
  - `http-method` ou `http-method-omission` : spécifie les méthodes `HTTP` à protéger ou pas
- `auth-constraint` : définit si l'authentification doit être utilisée et les rôles autorisés à utiliser la ressource protégée
  - `role-name`
- `user-data-constraint` : spécifie comment les données sont protégées pendant le transport entre le client et le serveur
  - `transport-guarantee` : peut prendre les valeurs `CONFIDENTIAL`, `INTEGRAL`, ou `NONE`. Avec `CONFIDENTIAL` et `INTEGRAL`, `HTTPS` sera utilisé.

Exemple 1

```
<!-- SECURITY CONSTRAINT #1 -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>wholesale</web-resource-name>
    <url-pattern>/acme/wholesale/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>PARTNER</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Exemple 2

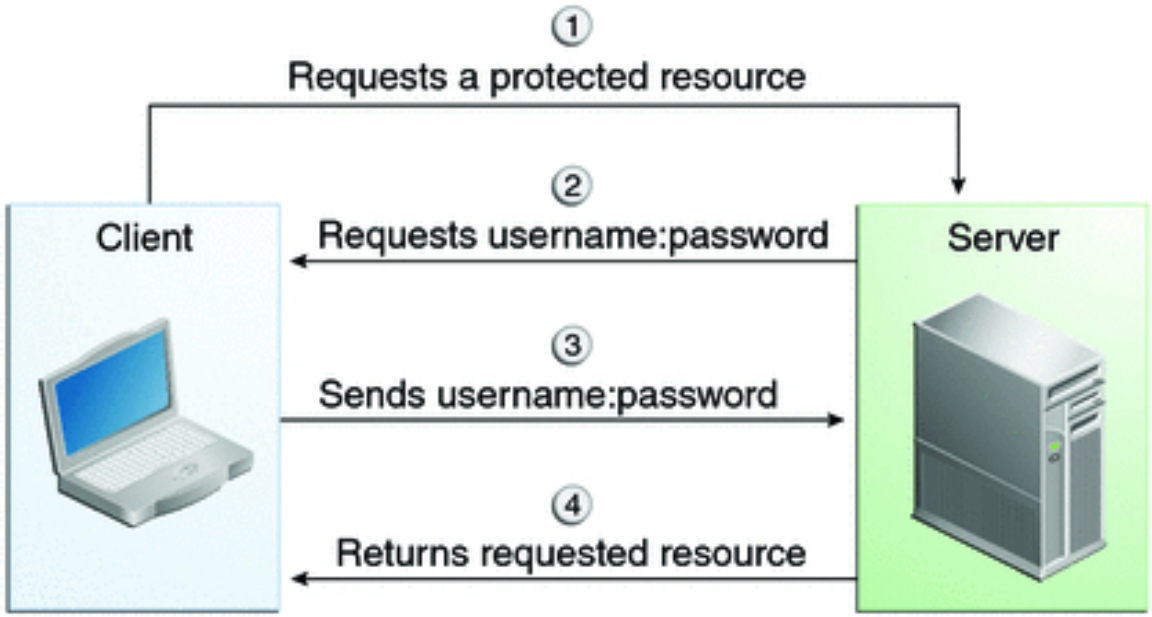
```
<!-- SECURITY CONSTRAINT #2 -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>retail</web-resource-name>
    <url-pattern>/acme/retail/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>CLIENT</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Mécanismes d'authentification

- Basic
- Form-based
- Digest
- Client
- Mutual

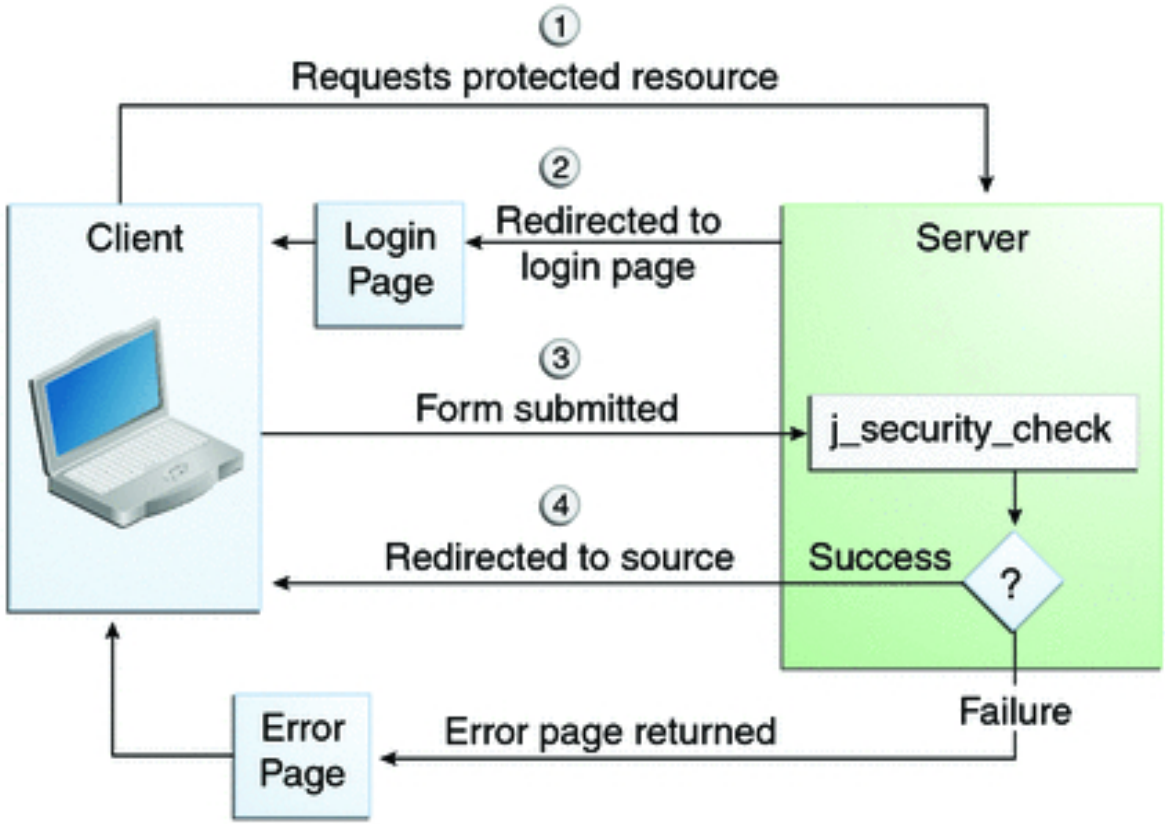
on discutera ici de Basic et Form-based

HTTP basic authentification



Form-based authentification

```
<form method="POST" action="j_security_check">
<input type="text" name="j_username">
<input type="password" name="j_password">
</form>
```



Spécification du mécanisme d'authentification

`login-config` :

- `auth-method` : mécanisme d'authentification (`NONE`,`BASIC`,`FORM`,...)
- `realm-name` : domaine à utiliser pour l'authentification
- `form-login-config` : spécifie la page du formulaire et d'erreur

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>file</realm-name>
  <form-login-config>
    <form-login-page>/login.xhtml</form-login-page>
    <form-error-page>/error.xhtml</form-error-page>
  </form-login-config>
</login-config>
```

Déclarer des rôles

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/security/protected/*</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>manager</role-name>
  </auth-constraint>
</security-constraint>

<!-- Security roles used by this web application -->
<security-role>
  <role-name>manager</role-name>
</security-role>
<security-role>
  <role-name>employee</role-name>
</security-role>
```

Authentifier par programmation

utiliser les méthodes `authenticate`, `login` et `logout` de `HttpServletRequest`

```
@WebServlet(name="ExempleServlet", urlPatterns={"/ExempleServlet"})
public class ExempleServlet extends HttpServlet {
  @EJB
  private ConverterBean converterBean;

  protected void processRequest(HttpServletRequest request,HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      out.println("<html>"); out.println("<head>");
      out.println("<title>Servlet ExempleServlet</title>");
      out.println("</head>"); out.println("<body>");
      request.login("ExemplelUser", "ExempleUser");
      BigDecimal res=converterBean.euroToDollar(new BigDecimal("1.0"));
      out.println("<h1>res= "+res+"</h1>");out.println("</body>");out.println("</html>");
    } catch (Exception e) {
      throw new ServletException(e);
    } finally { request.logout();out.close(); }
  }
}
```

Tester l'identité du client

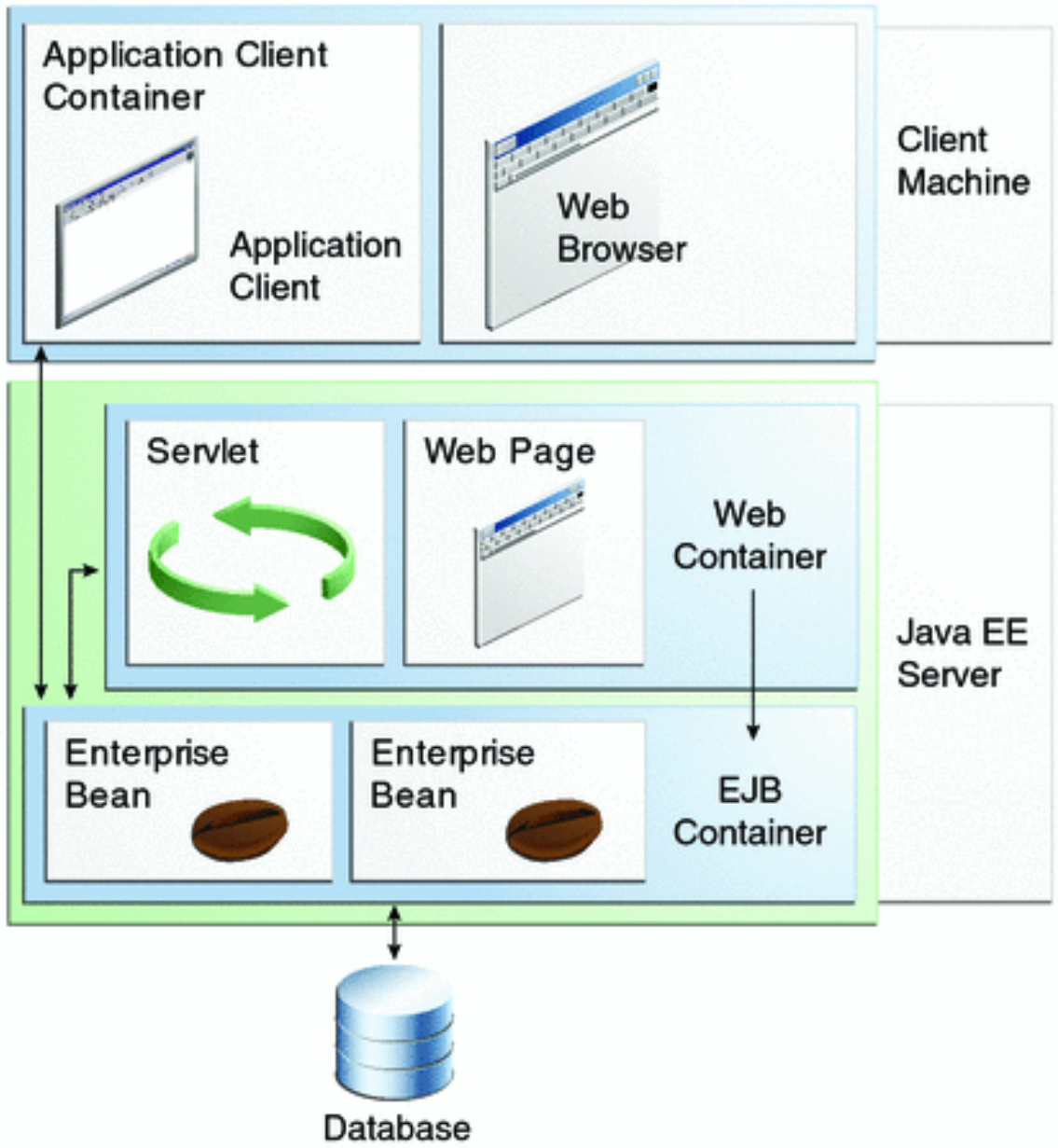
utilisation de méthodes de `HttpServletRequest`

- `getRemoteUser` : renvoie le nom de l'utilisateur associé à la requête par le container, nulll si personne ne s'est authentifié
- `isUserInRole` : permet de tester si l'utilisateur est dans un certain rôle. l'élément `security-role-ref` doit être déclaré dans le descripteur de déploiement avec un sous-élément `role-name`
- `getUserPrincipal` : renvoie un objet `java.security.Principal`

Tester l'identité du client

```
@DeclareRoles("testuser")
public class LoginServlet extends HttpServlet {
  protected void processRequest(HttpServletRequest request,HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      String userName = request.getParameter("txtUserName");
      String password = request.getParameter("txtPassword");
      out.println("Before Login" + "<br><br>");
      out.println("IsUserInRole?.." + request.isUserInRole("testuser")+"<br>");
      out.println("getRemoteUser?.." + request.getRemoteUser()+"<br>");
      out.println("getUserPrincipal?.." + request.getUserPrincipal()+"<br>");
      out.println("getAuthType?.." + request.getAuthType()+"<br><br>");
      try {
        request.login(userName, password);
      } catch(ServletException ex) {
        out.println("Login Failed with a ServletException.." + ex.getMessage()); return;
      }
      out.println("After Login..."+"<br><br>");
      out.println("IsUserInRole?.." + request.isUserInRole("testuser")+"<br>");
      out.println("getRemoteUser?.." + request.getRemoteUser()+"<br>");
      out.println("getUserPrincipal?.." + request.getUserPrincipal()+"<br>");
      out.println("getAuthType?.." + request.getAuthType()+"<br><br>");
      request.logout();
      out.println("After Logout..."+"<br><br>");
      out.println("IsUserInRole?.." + request.isUserInRole("testuser")+"<br>");
      out.println("getRemoteUser?.." + request.getRemoteUser()+"<br>");
      out.println("getUserPrincipal?.." + request.getUserPrincipal()+"<br>");
      out.println("getAuthType?.." + request.getAuthType()+"<br><br>");
...
```

Sommaire


Application Client Container
Application Client
Web Browser
Client Machine
Servlet
Web Page
Web Container
Java EE Server
Enterprise Bean
Enterprise Bean
EJB Container
Database

En général, le développeur et le déployeur de l'application sont différents.

- le développeur doit fournir donc une vue de sécurité dans laquelle il va définir des rôles de sécurité, des permissions sur les méthodes ...
- il faudra que le déployeur mappe les rôles sur les utilisateurs et groupes du serveur d'application

Spécifier les utilisateurs autorisés par déclaration de rôles

utilisation des annotations :

- `@DeclareRoles` : spécifie tous les rôles utilisés dans l'application, même ceux non utilisés dans l'annotation `@RolesAllowed`. L'ensemble des rôles est donc l'union de ceux utilisés dans `RolesAllowed` et `DeclareRoles`.

```
@DeclareRoles("BusinessAdmin")
public class Calculator {
   ...
}

@DeclareRoles({"Administrator", "Manager", "Employee"})
```

- `@RolesAllowed` : spécifie les rôles autorisés à accéder aux méthodes, sur une classe, sur une ou plusieurs méthodes

```
@DeclareRoles({"Administrator", "Manager", "Employee"})
public class Calculator {

  @RolesAllowed("Administrator") public void setNewRate(int rate) {
     ...
  }
}
```

Spécifier les utilisateurs autorisés par déclaration de rôles

- `@PermitAll` : tous les rôles sont autorisés à exécuter cette ou ces méthodes.

```
@RolesAllowed("RestrictedUsers")
public class Calculator {

  @RolesAllowed("Administrator")
  public void setNewRate(int rate) { ... }

  @PermitAll
  public long convertCurrency(long amount) { ... }
}
```

- `@DenyAll` : aucun rôle n'est autorisé à exécuter cette ou ces méthodes, ces méthodes ne peuvent pas s'exécuter dans le container EE.

```
@RolesAllowed("Users")
public class Calculator {


  @RolesAllowed("Administrator")
  public void setNewRate(int rate) { ... }

  @DenyAll
  public long convertCurrency(long amount) { ... }
}
```

Les rôles et l'héritage

```
@RolesAllowed("admin")
public class SomeClass {
    public void aMethod () {...}
    public void bMethod () {...}
    ...
}

@Stateless
public class MyBean extends SomeClass implements A {
    @RolesAllowed("HR")
    public void aMethod () {...}
    public void cMethod () {...}
}
```

- aMethod : @RolesAllowed("HR")
- bMethod : @RolesAllowed("admin")
- cMethod : permissions non spécifiées.

Normalement, la sécurité devrait être gérée de façon transparente par le container.

De rares cas demandent à ce qu'on accède au contexte de sécurité.

Deux méthodes de EJBContext permettent d'accéder à des informations de sécurité sur l'appelant.

- `getCallerPrincipal` : permet d'obtenir le nom de l'appelant
- `isCallerInRole` : permet de programmer la vérification des permissions alors que ce n'est pas possible de façon déclarative

getCallerPrincipal

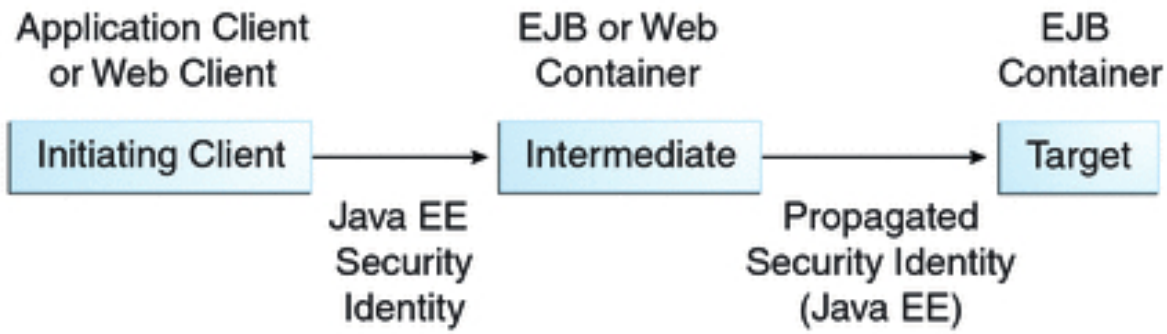
```
@Stateless
public class EmployeeServiceBean implements EmployeeService {
  @Resource SessionContext ctx;
  @PersistenceContext EntityManager em;

  public void changePhoneNumber(...) {
    ...
    // obtain the caller principal.
    callerPrincipal = ctx.getCallerPrincipal();
    // obtain the caller principal's name.
    callerKey = callerPrincipal.getName();
    // use callerKey as primary key to find EmployeeRecord
    EmployeeRecord myEmployeeRecord = em.find(EmployeeRecord.class,
        callerKey);
    // update phone number
    myEmployeeRecord.setPhoneNumber(...);
    ...
  }

  ...

}
```

isCallerInRole

```
@Stateless
public class PayrollBean implements Payroll {
  @Resource SessionContext ctx;

  public void updateEmployeeInfo(EmplInfo info) {
    oldInfo = ... read from database;


    // The salary field can be changed only by callers
    // who have the security role "payroll"
    if (info.salary != oldInfo.salary && !ctx.isCallerInRole("
        payroll")) {
      throw new SecurityException(...);
    }
    ...
  }

  ...

}
```

Propager l'identité

par défaut : l'identité de l'appelant est propagée au container de l'EJB cible



ou une identité précise est propagée à l'EJB cible, si celui ci réclame une identité donnée pour être accédé.
pour cela utiliser l'annotation `@RunAs`

```
@RunAs("Admin")
public class Calculator {
   //....
}
```