Programmazione Web

Lezione del 6 Maggio 2019

Docente: Novella Bartolini

Azione standard `<jsp:include>`

Reminder: redirezione attraverso direttiva include
<%@ include file="relativeURLspec"%>
specifica il percorso relativo (URL) di un file che deve essere incluso

- **`<jsp:include …>`**
  - Consente l'inclusione di contenuto **dinamico** in una pagina JSP
  - Più flessibile della direttiva **`include`**
    - Richiede maggiore overhead quando il contenuto della pagina cambia frequentemente
    - La direttiva include il codice al momento della traduzione (l'inclusione corrisponde ad una serie di print), mentre l'azione standard include il codice solo al momento dell'esecuzione

Azione `<jsp:include>`

Attribute	Description
`page`	Specifies the relative URI path of the resource to include. The resource must be part of the same Web application.
`flush`	Specifies whether the response buffer should be flushed before the **`include`** is performed. In JSP 1.1, this attribute is required to be **`true`**.

Azione <jsp:forward>

- **<jsp:forward>**
  - Consente ad una pagina JSP di inoltrare la richiesta ad altre risorse
- L'azione **<jsp:param>** (annidata nel forward) specifica coppie nome/valore di dati da allegare ad altre azioni

JSP e Bean

- Azione Standard **`<jsp:useBean>`**
  - Il codice contenuto negli scriptlet Java può esser inserito in una classe apposita invocata al momento del bisogno
  - Necessaria la conformità agli standard fissati per i JavaBean (architettura a componenti)

```
// classe JellyBean.java

package beans; //NOTARE CHE IL BEAN VIENE MESSO IN UN PACKAGE

public class JellyBean {
  private String color;

  public JellyBean()  {
  }

  public String getColor() {
                return color;
  }

  public void setColor(String newColor) {
                color=newColor;
  }
}
```

JavaBeans

- Ai fini dello sviluppo di servlet e JSP contano i seguenti aspetti della specifica:
  - I JavaBean non devono avere nessuna proprietà pubblica (attributi di classe privati).
  - Tutte le proprietà di un JavaBean che devono essere esposte dovranno avere metodi pubblici **get** e **set** secondo necessità.
    - Tali metodi dovranno avere nome **getVariableName** e **setVariableName**
  - Tutti i JavaBean devono avere un metodo costruttore senza argomenti

Utilizzo di un JavaBean in una JSP

- Affinchè sia visibile alla web application il bean deve essere incluso nella sotto-directory `classes` di `WEB-INF` con tutto il percorso del suo package

- Una volta creato il JavaBean, per poterlo utilizzare in una JSP vengono definiti 3 tag:
  - Per individuare o creare un JavaBean nell'area di visibilità specificata
  - Per definire una proprietà di un JavaBean (set)
  - Per leggere una proprietà di un JavaBean (get)

Caricamento di un JavaBean

- Il tag <jsp:useBean> ha la seguente sintassi:
  - <jsp:useBean id="nome" scope="page|request|session|application" class="nomeClasse" />
    - id è l'etichetta che viene assegnata al bean all'interno dell'applicazione (nome dell'istanza cui si fa riferimento)
    - scope definisce le modalità con cui l'istanza del Bean deve essere ricercata
- Esempio:
  - <jsp:useBean id="jb" scope="application" class="beans.JellyBean" />

Attributi dell'azione <jsp:useBean>

Attribute	Description
id	The name used to manipulate the Java object with actions **<jsp:setProperty>** and **<jsp:getProperty>**. A variable of this name is also declared for use in JSP scripting elements. The name specified here is case sensitive.
scope	The scope in which the Java object is accessible—**page**, **request**, **session** or **application**. The default scope is **page**.
class	The fully qualified class name of the Java object.
type	The type of the JavaBean. This can be the same type as the **class** attribute, a superclass of that type or an interface implemented by that type. The default value is the same as for attribute **class**. A **ClassCastException** occurs if the Java object is not of the type specified with attribute **type**.

Attributes of the **<jsp:useBean>** action.

Impostazione di una proprietà JavaBean

- Una volta creato il Bean è possibile definirne le proprietà con il tag **jsp:setProperty**
  - <jsp:setProperty name="beanName" property="propertyName" value="propertyValue" />

    (assegnazione di un valore)
  - <jsp:setProperty name="beanName" property="propertyName" param="parameterName" />

    (passaggio di un parametro della richiesta)

Si noti che qui "name" è il nome dell'istanza, ovvero l'"id"

Specificato nell'azione standard <jsp:usebean …>

Impostazione di una proprietà JavaBean

- Modo rapido per definire le proprietà del Bean attraverso dati provenienti da un form:
  - <jsp:setProperty name="beanName" property="*">
- Aggiungendo "*" il metodo setProperty fa corrispondere tutti i parametri dell'oggetto richiesta ai metodi set del bean e passa i valori tra di essi.
- Definizione automatica delle proprietà possibile solo se si rispettano le convenzioni sui nomi dei metodi set e get

Attributi dell'azione <jsp:setProperty>

Attribute	Description
`name`	The ID of the JavaBean for which a property (or properties) will be set.
`property`	The name of the property to set. Specifying `"*"` for this attribute causes the JSP to match the request parameters to the properties of the bean. For each request parameter that matches (i.e., the name of the request parameter is identical to the bean's property name), the corresponding property in the bean is set to the value of the parameter. If the value of the request parameter is `""`, the property value in the bean remains unchanged.
`param`	If request parameter names do not match bean property names, this attribute can be used to specify which request parameter should be used to obtain the value for a specific bean property. This attribute is optional. If this attribute is omitted, the request parameter names must match bean property names.
`value`	The value to assign to a bean property. The value typically is the result of a JSP expression. This attribute is particularly useful for setting bean properties that cannot be set using request parameters. This attribute is optional. If this attribute is omitted, the JavaBean property must be of a data type that can be set using request parameters.

Lettura delle proprietà di un bean

- Una volta creato il Bean è possibile leggerne le proprietà con il tag **jsp:getProperty**
  - <jsp:getProperty name="beanName" property="propertyName" />

Esempio: definizione dell’attributo di un bean per l’accesso da una pagina jsp

```
<!– Pagina di scrittura delle proprietà: scrivi.jsp -->

<html>
<head>
  <title> Impostazione del colore di JellyBean </title>
</head>
<%@ page language=“java” %>
<jsp:useBean id=“jb” scope=“session” class=“beans.JellyBean” />
<body>
<form action = “leggi.jsp" method = "get">


        <strong>Scegli il colore del bean:</strong><br />
        <label>Rosso
          <input name = "newColor" type = "radio"
            value = "red" checked = "checked" />
        </label>
```

Esempio: definizione dell'attributo di un bean per l'accesso da una pagina jsp

```
        <label>Verde
          <input name = "newColor" type = "radio"
            value = "green" /></label>

        <label>giallo
          <input name = "newColor" type = "radio"
            value = "yellow" /></label>



          <input type = "submit" value = "Submit" />
</form>
</body>
```

Come ottenere una proprietà di un JavaBean

```
  <!– leggi proprietà: leggi.jsp →

<html>
  <head>
    <title> Ottenere il colore di JellyBean </title>
  </head>
<%@ page language="java" %>
<jsp:useBean id="jb" scope="session" class="JellyBean" />




<body>
   <jsp:setProperty name="jb" property="color" param="newColor" />
   Il colore del bean è stato impostato al valore:</td>
   <jsp:getProperty name="jb" property="color" /></td>
</body>
</html>
```

Esercizio:

- Scrivere semplici esempi di pagine jsp contenenti:
  - La direttiva "include"
  - Le azioni standard
    - include
    - forward
    - useBean (con diversi contesti di visibilità)
    - setProperty (sia il caso in cui viene passato un valore, sia il caso in cui viene preso il valore di un parametro della richiesta)
    - getProperty
  - Descrivere (non codice) come vengono tradotte dal jsp container

    (a tale scopo cercare la cartella "work" dove vengono memorizzate le jsp tradotte nelle servlet corrispondenti)

JSP useBean: contesti di visibilità

- request
  - <jsp:useBean id="..." class="..." scope="request" />
- session
  - <jsp:useBean id="..." class="..." scope="session" />
- application
  - <jsp:useBean id="..." class="..." scope="application" />
- page
  - <jsp:useBean id="..." class="..." scope="page" />
    oppure
    <jsp:useBean id="..." class="..." /> (page è il contesto di default)

Visibilità di un JavaBean

- Si può accedere ad un bean da più pagine JSP, a condizione che il bean sia stato dichiarato con scope idoneo
  - Se si osserva il codice generato per le JSP (work folder) si noterà una chiamata al metodo relativo all'oggetto implicito rappresentante lo scope prescelto
    - ad esempio *session* e sia lo stesso utente ad accedere ad entrambe le pagine
    - ad esempio *application,* allora tutti gli utenti avranno accesso alla stessa istanza del bean
    - ad esempio *request*, se la richiesta viene rediretta a altre pagine

Come configurare il web.xml per pagine jsp

In generale non e` necessario scrivere nulla in una pagina jsp

La pagina One.jsp
puo` essere chiamata direttamente indicando l'URL
in cui si trova nella context root

http://localhost:8080/my_context_root/One.jsp

Tuttavia possiamo sempre usare il deployment descriptor
per definire un alias anche per le pagine jsp

Come configurare il web.xml per pagine jsp

```
<web-app>
        <servlet>
                <servlet-name>Paperino</servlet-name>
                <jsp-file>/One.jsp</jsp-file>
        </servlet>
        <servlet-mapping>
                <servlet-name>Paperino</>
                <url-pattern>/lachiamoinquestomodo</url-pattern>
        </servlet-mapping>
</web-app>
```

http://localhost:8080/my_context_root/One.jsp

oppure

http://localhost:8080/my_context_root/ lachiamoinquestomodo

Welcome file in web.xml

```
<welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
      <welcome-file>index.html</welcome-file>
 </welcome-file-list>
```

Il tag <welcome-file> viene usato per denotare una risorsa di default da invocare quando la richiesta punta direttamente alla context root.

Il container che riceve una richiesta diretta alla context root, prima di dire che la richiesta non punta ad una risorsa corretta, andra` a controllare la presenza di file corrispondenti al tag welcome-file del web.xml

url-pattern in web.xml

```
<web-app>
   <servlet>
      <servlet-name>DonaldDuck</servlet-name>
      <servlet-class>MyServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>DonaldDuck</servlet-name>
      <url-pattern>/cartoon_path/*</url-pattern>
   </servlet-mapping>
</web-app>
```

Esercitazione

- Scrivere due servlet A e B che producono ciascuna una pagina html propria in cui si dica se in concomitanza con l'elaborazione della richiesta corrente si è avuta o no la creazione di una nuova sessione.
- Le due servlet citate accedono allo stesso bean con visibilità di applicazione. Il suddetto bean deve contenere informazioni generiche sull'applicazione e in particolare due contatori: un contatore di accessi alle singole risorse e un contatore di sessioni.
- Scrivere infine una pagina JSP riassuntiva che oltre ad aggiornare come le servlet A e B i dati del bean ne legga il contenuto e lo stampi attraverso opportune azioni standard.

Protezione delle risorse - Autenticazione

- Un utente tenta di accedere ad una risorsa protetta (es. una pagina jsp).
- Se l'utente è già stato autenticato la risorsa viene resa disponibile, altrimenti viene chiesto all'utente di digitare nome utente e password.
- Se nome e password non possono essere autenticati, viene visualizzato un errore e l'utente ha la possibilità di scrivere nuovamente i dati, altrimenti la risorsa viene resa disponibile.

Autenticazione dichiarativa vs. programmata

- L'approccio dichiarativo all'autenticazione è basato interamente sul servlet container che gestisce i nomi degli utenti, le password e i ruoli
- L'approccio programmato implica invece la gestione diretta della sicurezza da parte di servlet e pagine jsp

Autenticazione dichiarativa

- Gli aspetti relativi alla sicurezza sono interamente gestiti dal serlvet container.
- Per prevenire accessi non autorizzati
  - Si utilizza il descrittore della web application (*web.xml*) per dichiarare che certe risorse sono riservate a utenti che rivestono certi ruoli.
  - Si definisce un metodo di autenticazione per identificare gli utenti e rispettivi ruoli.
  - Quando viene richiesta una risorsa protetta, il server richiede all'utente username e password, li confronta con un set predefinito e, automaticamente, tiene traccia degli utenti già autenticati. Questo processo è completamente trasparente alle servlet e alle pagine JSP.
- Per preservare la sicurezza dei dati sulla rete
  - Si usa il descrittore della web application per specificare che a certe risorse si può accedere solo attraverso una connessione https. Se gli utenti provano ad usare una connessione HTTP vengono forzati ad usare HTTPS con meccanismi di redirezione.

Autenticazione programmata

- Le risorse protette (servlet e pagine JSP) sono responsabili della gestione della propria sicurezza.
  - Portabilità del codice. Non ci sono elementi della web application che dipendono dal particolare server utilizzato. Non sono necessarie ulteriori specifiche nel descrittore.
- Per prevenire accessi non autorizzati
  - Ciascuna servlet o pagina JSP deve autenticare l'utente o verificare che sia già stato autenticato.
- Per preservare la sicurezza dei dati sulla rete
  - Ogni servlet o pagina JSP deve controllare il protocollo usato
  - Se gli utenti provano ad usare una connessione HTTP la servlet o la pagina JSP devono redirigere le richieste sul protocollo HTTPS.

Tipi di autenticazione dichiarativa

1. Autenticazione BASIC
2. Autenticazione basata su form
3. Autenticazione DIGEST

- La scelta del meccanismo viene specificata nel file web.xml in corrispondenza dei tag
  - `<login-config>` e `<auth-method>`

Principali e ruoli

- L'utente è un *principal*
- I *principal* sono entità denominate, che in genere rappresentano singoli individui o società.
- I principal possono ricoprire uno o più *ruoli (roles)*
  - Es: un cliente può essere nello stesso tempo un dipendente
- Le restrizioni di sicurezza della specifica delle servlet, associano i ruoli con le risorse protette

Restrizioni di sicurezza: /WEB-INF/web.xml

```
<web-app>
   <display-name>Web App x test meccanismi di sicurezza </display-name>
   <description> Test dei meccanismi di sicurezza </description>
   <login-config> … vedremo tra poco …  </login-config>
   <security-constraint>
   <web-resource-collection> <!-- risorse protette -->
      <web-resource-name>Una pagina protetta </web-resource-name>
      <url-pattern>/jsps/pagina-protetta.jsp</url-pattern>
   </web-resource-collection>
   <auth-constraint><!-- ruoli associati alle risorse -->
      <role-name>tomcat</role-name>
      <role-name>role1</role-name>
   </auth-constraint>
   </security-constraint>
</web-app>
```

Elementi di <security-constraint>

Elemento	Tipo*	Descrizione
web-resource-collection	**+**	Un sottoinsieme delle risorse di un'applicazione web al quale vengono applicate le restrizioni di sicurezza
auth-constraint	**?**	Le restrizioni di autorizzazione collocate su una o più raccolte di risorse web (conterrà i tag che identificano i ruoli)
user-data-constraint	**?**	Una specifica sul meccanismo di trasporto dei dati inviati tra un client e un servlet container

* Tipo: + = uno o più

? = uno, facoltativo

Elementi <web-resource-collection>

Elemento	Tipo*	Descrizione
web-resource-name	**1**	Il nome di una risorsa web
description	**?**	La descrizione di una risorsa web
url-pattern	*****	Uno schema url associato ad una risorsa web
http-method	**?**	Un metodo http associato ad una risorsa web

* Tipo: 1= uno, obbligatorio

?= zero o più

*=uno o più

Elementi <auth-constraint>

Elemento	Tipo*	Descrizione
description	**?**	Una descrizione della restrizione di un'autorizzazione
role-name	*****	Il ruolo al quale si applica la restrizione

* Tipo: 1= uno, obbligatorio

?= zero o più

*=uno o più

Elementi <user-data-constraint>

Elemento	Tipo*	Descrizione
description	**?**	Una descrizione della restrizione dei dati utente
transport-guarantee	**1**	NONE: non ci sono restrizioni sul trasporto dei dati INTEGRAL: il contenitore della servlet deve garantire che i dati non potranno essere manomessi durante il trasporto (di solito SSL) CONFIDENTIAL: i dati in transito non devono essere letti (di solito SSL)

* Tipo: 1= uno, obbligatorio

?= zero o più

* =uno o più

Risorse – Ruoli - Principali

- Le restrizioni specificate nel file web.xml associano i ruoli alle risorse.
- L'associazione dei ruoli ai principali è compito del servlet container
  - (file <catalina-home>/conf/tomcat-users.xml).

File di configurazione
<CATALINA-HOME>/conf/tomcat-users.xml

```
<tomcat-users>
 . . .
                <role rolename="cliente"/>
                <user name="rossi" password="tomcat" roles="cliente", "altro" />
 . . .
</tomcat-users>
```

Metodi di **HttpServletRequest**

- Principal getUserPrincipal()
  - Restituisce un riferimento a java.security.Principal
- Boolean isUserInRole**(String)**
  - Stabilisce se un utente ricopre un ruolo specificato da una stringa
- String getRemoteUser()
  - Restituisce il nome utente utilizzato per il login

Metodi di **HttpServletRequest** (cont.)

- Le API della servlet non forniscono i metodi "setter" corrispondenti ai metodi "getter" getUserPrincipal() e getRemoteUser()

→ Le applicazioni non possono impostare principali e ruoli; questi possono essere impostati solo dai contenitori di servlet

Metodi di **HttpServletRequest** (cont.)

- String getAuthType()
  - Restituisce il tipo di autenticazione BASIC, FORM, DIGEST.
- Boolean isSecure()
  - Restituisce true se la connessione è http o https
- String getScheme()
  - Fornisce lo schema che rappresenta il meccanismo di trasporto: http, https, ftp o altro.

/WEB-INF/web.xml (aut. dichiarativa)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
. . .
<web-app>
. . .
   <security-constraint>
   . . .
   </security-constraint>

      <login-config>
            <auth-method> BASIC </auth-method>
            <realm-name>Esempio di autenticazione Basic </realm-name>
      </login-config>
</web-app>
```

Il metodo **HttpServletRequest.getAuthType()** fornisce il metodo di autenticazione utilizzato

Autenticazione BASIC

- Quando un client tenta di accedere ad una risorsa protetta, il servlet container invia automaticamente una finestra di richiesta di nome utente e password
- L'invio della finestra di richiesta avviene in modo trasparente alle pagine JSP e alle servlet
- Questo metodo manca completamente di sicurezza. Le password vengono trasmesse con la codifica base64

Codifica base64

- La password viene trasformata in una sequenza di caratteri appartenenti ad un sottogruppo del set di caratteri ASCII. In questo modo, ogni carattere codificato può essere rappresentato con sei bit.
- Ogni gruppo di 24 bit viene diviso in quattro gruppi di sei bit, ad ognuno dei quali si associa il corrispondente carattere ASCII appartenente al sottogruppo specificato.
- Le password sono estremamente vulnerabili

Esempio sull’autenticazione BASIC

Web Application di test (aut. BASIC)

/JSPS/pagina-protetta.jsp

```
<html><head><title>Una pagina protetta </title></head>
<body>
<%@ include file="show-security.jsp" %>

<%  if (request.isUserInRole("tomcat")){ %>
      Appartieni al ruolo <i>tomcat</i> <br>
<%      } else { %>
      Non appartieni al ruolo <i>tomcat</i><br>
<%      }  %>

<% if (request.isUserInRole("role1")) { %>
     Appartieni al ruolo <i>role1</i> <br>
<%     } else { %>
     Non appartieni al ruolo <i>role1</i><br>
<%     }  %>

</body></html>
```

Web Application di test (aut. BASIC)

/JSPS/show-security.jsp

```
<font size="4" color="blue">
  Informazioni sulla sicurezza
</font> <br>
User principal: <%= request.getUserPrincipal().getName() %>. <br>
Request authenticated with: <%= request.getAuthType() %>. <br>

<% if (request.isSecure()) { %>
    This connection is secure. <br>
<% } else { %>
    This connection is NOT secure. <br>
<% } %>

Server Address: <%= request.getServerName() %> <br>
Remote Host: <%= request.getRemoteHost() %> <br>
Remote Addr; <%= request.getRemoteAddr() %>
```

Web Application di test (aut. BASIC)
/WEB-INF/web.xml

```
<web-app>
   <display-name>Web App x test meccanismi di sicurezza </display-name>
   <description>
   Test dei meccanismi di sicurezza
   </description>

   <security-constraint>
   <!-- risorse protette -->
   <web-resource-collection>
      <web-resource-name>Una pagina protetta </web-resource-name>
      <url-pattern>/jsps/pagina-protetta.jsp</url-pattern>
   </web-resource-collection>
. . .
```

Web Application di test (aut. BASIC) /WEB-INF/web.xml (cont.)

```
. . .
    <auth-constraint>
        <!-- ruoli associati alle risorse indicate sopra -->
        <role-name>tomcat</role-name>
        <role-name>role1</role-name>
    </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>Basic Authentication Example</realm-name>
    </login-config>
</web-app>
```

<CATALINA-HOME>/conf/tomcat-users.xml

```
<tomcat-users>
    <user name="tomcat" password="tomcat" roles="tomcat" />
    <user name="role1" password="tomcat" roles="role1" />
    <user name="both" password="tomcat" roles="tomcat","role1" />
</tomcat-users>
```

Il file tomcat-users.xml appartiene al servlet container e realizza l'associazione tra il principal e il ruolo da lui ricoperto

Autenticazione basata su form

- Permette di controllare l'aspetto e il comportamento della pagina di login
- Il metodo basato su form funziona come il metodo basic ma viene visualizzata la pagina di login anzichè una finestra di dialogo
- Viene inoltre specificata una pagina di errore per il caso di mancata autenticazione
- La password viene trasmessa con la codifica base64

Autenticazione basata su form (cont.)

- Implementare una pagina di login
- Implementare una pagina di errore, che verrà visualizzata in caso di mancata autenticazione
- Nel descrittore della web application specificare che si adotta l'autenticazione basata su form e il nome delle pagine di login e di errore

Web Application di test (aut. FORM)
/JSPS/login.jsp

```
<html><head><title>Pagina di login</title></head>
<body>
<font size="4" color="blue">
   Per favore inserisci i tuoi dati:<br></font>

<form action="j_security_check" method="POST">
<table>
<tr><td>Name:</td>
   <td><input type="text" name="j_username"></td></tr>
<tr><td>Password:</td>
   <td><input type="password" name="j_password" size="8"></td>
</tr>
</table>
<br>
  <input type="submit" value="login">
</form>
</body>
</html>
```

Web Application di test (aut. FORM)
/JSPS/login.jsp

- **`j_username`**, **`j_password`**, **`j_security_check`** sono nomi dettati dalla servlet specification
- `j_username`: il nome del campo username
- `j_password`: il nome del campo password
- `j_security_check`: l'azione del modulo di login

- La pagina di login così definita viene automaticamente richiamata dal servlet container quando si tenta di accedere alla risorsa protetta specificata nel descrittore.

Web Application di test (aut. FORM)
/JSPS/error.jsp

```
<html><head><title>ERRORE !!!</title></head>

<body>

<font size="6" color="red">
   I dati inseriti non sono validi!<br>
</font>

Clicca <a href=' <%=response.encodeURL("login.jsp") %>'>qui</a> per riprovare!

</body>
</html>
```

Web Application di test (aut. FORM)
/WEB-INF/web.xml

```
<web-app>
   <display-name>test di autenticazione mediante form</display-name>
   <description>Test </description>
   <security-constraint>
   <!-- risorse protette -->
   <web-resource-collection>
      <web-resource-name>Una pagina protetta </web-resource-name>
      <url-pattern>/jsps/pagina-protetta.jsp</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <!-- ruoli associati alle risorse indicate sopra -->
      <role-name>tomcat</role-name>
      <role-name>role1</role-name>
   </auth-constraint>
   </security-constraint> . . .
```

Web Application di test (aut. FORM)
/WEB-INF/web.xml (cont.)

```
. . .
<login-config>
        <auth-method>FORM</auth-method>
        <form-login-config>
            <form-login-page>/jsps/login.jsp</form-login-page>
            <form-error-page>/jsps/error.jsp</form-error-page>
        </form-login-config>
    </login-config>
</web-app>
```

/WEB-INF/web.xml

```
. . .
<web-app>
. . .
   <security-constraint>
   <web-resource-collection> . . .
      <url-pattern>/jsps/risorsaA.jsp</url-pattern>
   </web-resource-collection>
   <auth-constraint> . . .
      <role-name>ruolo A</role-name>
   </auth-constraint>
   </security-constraint>
   <security-constraint>
   <web-resource-collection> . . .
      <url-pattern>/jsps/risorsaB.jsp</url-pattern>
   </web-resource-collection>
   <auth-constraint> . . .
      <role-name>ruolo B</role-name>
   </auth-constraint>
 /security-constraint> <
</web-app>
```

Meccanismi di autenticazione dichiarativa

- BASIC (equiv. FORM) e DIGEST
- Sono definiti nel documento RFC 2617, http://ftp.isi.edu/in-notes/rfc2617.txt
- BASIC e FORM: la password viene trasmessa in chiaro
- DIGEST: la password non viene trasmessa, ma viene trasmesso un valore hash.

Autenticazione DIGEST

- Quando un client tenta di accedere ad una risorsa protetta, il servlet container invia automaticamente una finestra di richiesta di nome utente e password
- L'invio della finestra di richiesta avviene in modo trasparente alle pagine JSP e alle servlet
- La password viene codificata con metodi di hashing

Codifica con il metodo DIGEST

- Lo schema Digest è basato su un meccanismo di domanda-risposta.
- L'entità responsabile dell'autenticazione (il server) propone all'utente un valore *nonce* (ogni volta diverso e valido solo per la richiesta corrente)
- Affinchè l'utente possa essere autenticato, deve rispondere con un valore di checksum calcolato in base a
  - Username e Password
  - Il valore nonce proposto
  - Il metodo HTTP
  - URI

/WEB-INF/web.xml (aut. dichiarativa)

Per realizzare l'autenticazione secondo lo schema Digest ci si comporta esattamente come per il metodo basic, ma si specifica il metodo **DIGEST** nel file /WEB-INF/web.xml

```
<web-app>
. . .
   <security-constraint>
   . . .
   </security-constraint>

     <login-config>
            <auth-method> DIGEST </auth-method>
            <realm-name>Esempio di autenticazione Digest </realm-name>
     </login-config>
</web-app>
```

Autenticazione dichiarativa vs. programmata

- L'approccio dichiarativo all'autenticazione è basato interamente sul servlet container che gestisce i nomi degli utente, le password e i ruoli
- L'approccio programmato implica invece la gestione diretta della sicurezza da parte di servlet e pagine jsp

Problemi con meccanismi dichiarativi

- L'accesso è tutto-o-niente
  - Gli utenti possono accedere alle risorse protette oppure essere bloccati, senza alternative ulteriori
  - Non è possibile proporre agli utenti contenuti diversi a seconda dei ruoli ricoperti.
- L'accesso, basato esclusivamente su password è completamente controllato dal server.

Problemi con meccanismi dichiarativi

- La soluzione non è sempre portabile
- I server devono supportare *qualche* metodo per la definizione di utenti, password e ruoli, ma server diversi lo fanno in modo diverso.
- E' necessario modificare il file web.xml

Autenticazione programmata

- Le risorse protette (servlets e pagine JSP) sono responsabili della gestione della propria sicurezza.
  - Protabilità del codice. Non ci sono elementi della web application che dipendono dal particolare server utilizzato. Non sono necessarie ulteriori specifiche nel descrittore.
- Per prevenire accessi non autorizzati
  - Ciascuna servlet o pagina JSP deve autenticare l'utente o verificare che sia già stato autenticato.
- Per preservare la sicurezza dei dati sulla rete
  - Ogni servlet o pagina JSP deve controllare il protocollo usato
  - Se gli utenti provano ad usare una connessione HTTP la servlet o la pagina JSP devono redirigere le richieste sul protocollo HTTPS.

Approccio combinato: dichiarativo + programmatico

- Si fa affidamento sul server per la gestione di nomi, password e ruoli attraverso metodi dichiarativi
  - Autenticazione BASIC, basata su form oppure DIGEST
- Si gestisce l'accesso alle risorse in modo esplicito dalle servlet o dalle pagine JSP
  - Esempio: si può cambiare il contenuto della pagina secondo l'identità del richiedente (impossibile con metodi puramente dichiarativi).

Approccio combinato: dichiarativo + programmatico (cont.)

- Si utilizzano i seguenti metodi di **HttpServletRequest**
  - **boolean isUserInRole(String)**
  - **String getRemoteUser()**
  - **Principal getUserPrincipal()**

  per definire il contenuto della risorsa protetta sulla base dell'identità del richiedente.

Approccio combinato: /WEB-INF/web.xml (cont.)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
. . .
<web-app>
   <display-name>Web App x test meccanismi di sicurezza </display-name>
   <description> Test dei meccanismi di sicurezza </description>

   <security-constraint>
   <web-resource-collection> <!-- risorse protette -->
      <web-resource-name>Una pagina protetta </web-resource-name>
      <url-pattern>/jsps/pagina-protetta.jsp</url-pattern>
   </web-resource-collection>
   <auth-constraint><!-- ruoli associati alle risorse -->
      <role-name>pagante</role-name>
      <role-name>nonpagante</role-name>
   </auth-constraint>
   </security-constraint>
</web-app>
```

Approccio combinato: /WEB-INF/pagina-protetta.jsp (cont.)

- Notare che il file web.xml consente l'accesso alla risorsa protetta sia agli utenti paganti che a quelli non paganti

```
<html><head><title>Una pagina protetta </title></head>
<body>
<%  if (request.isUserInRole("pagante")){ %>
      Informazioni riservate agli utenti abbonati
<%     } else if (request.isUserInRole("nonpagante")) {  %>
      Informazioni per chi non si è ancora abbonato
<%     }  %>
</body></html>
```

Approccio combinato: /WEB-INF/pagina-protetta.jsp (cont.)

- Oppure se gli utenti non paganti cercano di accedere ad una risorsa riservata agli abbonati si può decidere di deviare la richiesta verso una pagina di registrazione (o di login):

```
<html><head><title>Una pagina protetta </title></head>
<body>
<%  if (request.isUserInRole("pagante")){ %>
      Informazioni riservate agli utenti abbonati
<%      } else if (request.isUserInRole("nonpagante")) {
      response.sendRedirect("PaginaDiLogin.jsp");
      }  %>
</body></html>
```