Programmazione WEB

Lezione del 30 Aprile 2020

Docente: Novella Bartolini

Tag che includono il corpo

- Non richiedono di estendere BodyTagSupport, se il corpo del tag non richiede elaborazione

```
<prefix:tagName>
        JSP Content
</prefix:tagName>
<prefix:tagName att1="val1" ... >
        JSP Content
</prefix:tagName>
```

Tag che includono il corpo: la classe Tag Handler

- **doStartTag**
  - Per includere il corpo restituisce **EVAL_BODY_INCLUDE** invece di **SKIP_BODY**
- **doEndTag**
  - Metodo che definisce azioni da intraprendere dopo l'inclusione del corpo
  - Restituisce **EVAL_PAGE** oppure **SKIP_PAGE** a seconda dei casi

Esempio 2: HeadingTag.java

```
package tags;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;

public class HeadingTag extends TagSupport {
  private String bgColor; // Un attributo obbligatorio
  private String border = null;
  ... //altri attributi

  public void setBgColor(String bgColor) {
    this.bgColor = bgColor;
  }

  public void setBorder(String border) {
    this.border = border;
  }
 ... //altri metodi setter per gli altri attributi
```

Esempio 2: HeadingTag.java (Continua)

```
public int doStartTag() {
    try {
      JspWriter out = pageContext.getOut();
      out.print("<TABLE BGCOLOR=\"" + bgColor + "\"" +
                "  ALIGN=\"" + align + "\"");
      if (border != null) {
        out.print(" BORDER=\"" + border + "\"");
      }
      ...
    } catch(IOException ioe) {
      System.out.println("Error in HeadingTag: " + ioe);
    }
    return(EVAL_BODY_INCLUDE); // Include il corpo del tag
  }
```

Esempio 2: HeadingTag.java (Continua)

```
public int doEndTag() {
    try {
      JspWriter out = pageContext.getOut();
      out.print("</TABLE>");
    } catch(IOException ioe) {
      System.out.println("Error in HeadingTag: " + ioe);
    }
    return(EVAL_PAGE); // Continue with rest of JSP page
  }
```

Tag che includono il corpo: Tag Library Descriptor (TLD)

L'unica novità
(rispetto ai tag che non includono il corpo)
è nell'elemento bodycontent

- Deve essere **JSP** invece di **empty:**

```
<tag>
  <name>…</name>
  <tagclass>…</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>…</info>
</tag>
```

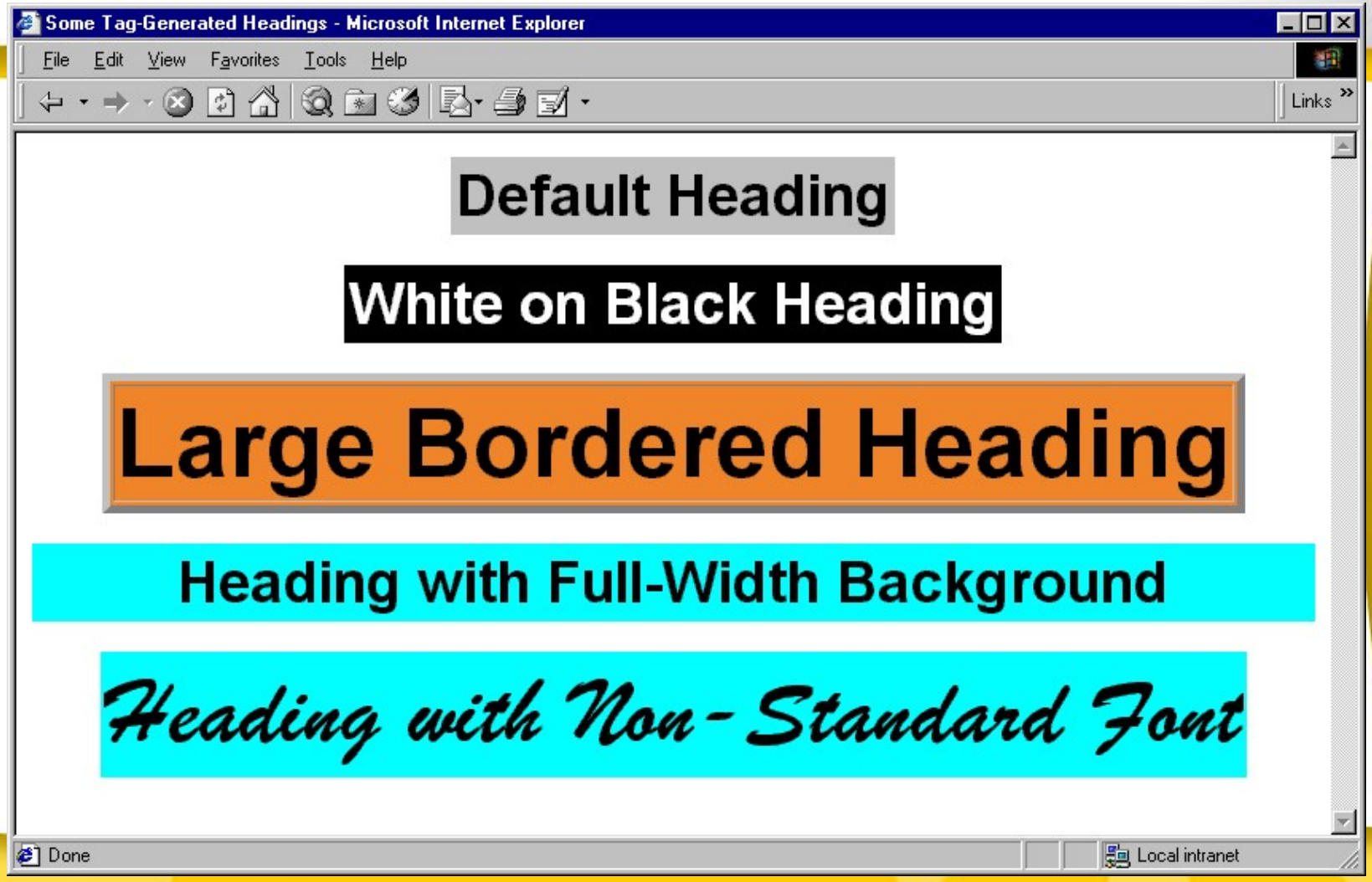
Esempio:
File TLD per il tag della classe HeadingTag

```
...
<taglib>
  <tag>
    <name>heading</name>
    <tagclass>tags.HeadingTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Scrive una tabella di 1 cella per definire un'intestazione</info>
    <attribute>
      <name>bgColor</name>
      <required>true</required> <!-- bgColor obbligatorio -->
    </attribute>
    <attribute>
      <name>border</name>
      <required>false</required> <!-- la dim del bordo della tabella è opzionale -->
    </attribute>
    ...
  </tag>
 </taglib>
```

Uso del tag “heading” in una pagina JSP

```
<%@ taglib uri="libreria.tld" prefix="formato" %>
<formato:heading bgColor="#C0C0C0">
Default Heading
</formato:heading>
<P>
<formato:heading bgColor="BLACK" color="WHITE">
White on Black Heading
</formato:heading>
<P>
<formato:heading bgColor="#EF8429" fontSize="60" border="5">
Large Bordered Heading
</formato:heading>
<P>
<formato:heading bgColor="CYAN" width="100%">
Heading with Full-Width Background
</formato:heading>
...
```

Uso del tag “heading”



Esempio 3: tag per autenticazione

- Definire un bean per una piccola base dati di utenti in cui
  - il singolo utente sia rappresentato da un bean
  - la stessa base dati sia un bean con metodi per l'aggiunta e la ricerca di un utente in un vettore di bean utente
- Definire un meccanismo di sicurezza basato sull'uso di un tag personalizzato

  <security:enforceLogin loginPage="/login.jsp"
  errorPage="/error.jsp">

  il cui funzionamento sia di rimandare alle pagine di login e di errore a seconda dei casi, e che mostri il seguito della pagina solo nel caso in cui l'utente sia stato correttamente autenticato

Rappresentazione del singolo utente
/WEB-INF/classes/beans/User.java

```
package beans;

public class User implements java.io.Serializable {
   private final String userName, password, hint;

   //costruttore1
   public User() {      }
   //costruttore2
   public User(String userName, String password, String hint) {
            this.userName=userName;
            this.password=password;
            this.hint=hint;
            }

//metodi getter
   public String getUserName() {return userName;}
   public String getPassword() {return password;}
   public String getHint() {return hint;}

   //metodo che controlla se il bean utente ha il nome uname e una password pwd
   public boolean equals(String uname, String pwd) {
            return (getUserName().equals(uname) && getPassword().equals(pwd));
   }
}
```

Rappresentazione del singolo utente (cont.)

- Gli utenti hanno tre proprietà: nome, password e suggerimento
- Gli attributi del bean utente non possono essere modificati – le proprietà sono impostate dal costruttore (uso della keyword **final**)

- Se un oggetto non può essere modificato dopo la creazione, non possono verificarsi incoerenze dovute all'accesso concorrente di più thread.

Database di accesso

/WEB-INF/classes/beans/LoginDB.java

```
package beans;

import java.util.Iterator;
import java.util.Vector;

public class LoginDB implements java.io.Serializable {
   private Vector users = new Vector();
   private User[] defaultUsers = {
      new User("Picasso","Pablo","Il mio nome"), };
  //costruttore (aggiunge al vettore users tutti gli utenti di default)
  public LoginDB() {
    for (int i=0;i<defaultUsers.length; i++)
      users.add(defaultUsers[i]);
    }
   //metodo adder (aggiunge al vettore users il bean utente con attributi dati)
   public void addUser(String uname, String pwd, String hint) {
         users.add(new User(uname,pwd,hint));
         }
   . . .
```

Database di accesso (cont.)

/WEB-INF/classes/beans/LoginDB.java

```
. . . //continua def della classe LoginDB

   //metodo di ricerca del bean utente identificato da nome e password
   public User getUser(String uname, String pwd) {
         Iterator it = users.iterator();
         User bean;
         synchronized (users) {
                  while (it.hasNext()) {
                     bean = (User)it.next();
                     if (bean.equals(uname,pwd))
                                                return bean;
                     }
                  }
          return null;
   }

. . .
```

Database di accesso (cont.)

/WEB-INF/classes/beans/LoginDB.java

```
. . . //continua def della classe LoginDB

   //metodo di ricerca del suggerimento di un bean utente identificato da
   // un certo nome

   public String getHint(String uname) {
         Iterator it = users.iterator();
         User bean;
         synchronized (users) {
                  while (it.hasNext()) {
                     bean = (User) it.next();
                     if (bean.getUserName().equals(uname))
                                    return bean.getHint();
                     }
                  }
          return null;
   }

}
```

Una pagina protetta: protectedPage.jsp

```
<html><head><title> Una pagina protetta </title></head>
<%@taglib uri="/WEB-INF/tlds/security.tld" prefix="security" %>
<body>

<security:enforceLogin loginPage="/login.jsp"
                       errorPage="/error.jsp" />

<jsp:useBean id="user" type="beans.User" scope="session" />

Questa è una pagina protetta. Benvenuto <%= user.getUserName() %>
</body>
</html>
```

Tag Library Descriptor: security.tld

```
<taglib><tlibversion>1.0</tlibversion><jspversion>1.1</jspversion>
   <tag>
      <name>enforceLogin</name>
      <tagclass>tags.EnforceLoginTag</tagclass>
      <bodycontent>JSP</bodycontent>
      <attribute>
          <name> loginPage </name>
          <required> true </required>
          <rtexprvalue> true </rtexprvalue>
      </attribute>
      <attribute>
          <name> errorPage </name>
          <required> false </required>
          <rtexprvalue> true </rtexprvalue>
      </attribute>
    </tag>
  <tag> <name>showErrors</name>
          <tagclass>tags.ShowErrorsTag</tagclass>
          <bodycontent>empty</bodycontent>
  </tag>
</taglib>
```

Classe handler del tag `EnforceLogin`

/WEB-INF/classes/tags/EnforceLoginTag.java

```
package tags;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;

public class EnforceLoginTag extends TagSupport {
   private String loginPage, errorPage;
   public void setLoginPage (String loginPage) {
     this.loginPage=loginPage;
     }
   public void setErrorPage (String errorPage) {
     this.errorPage=errorPage;
     }
. . .
```

Classe handler del tag **EnforceLogin**

/WEB-INF/classes/tags/EnforceLoginTag.java

```
. . .//il metodo doEndTag decide se permettere la visualizzazione del resto
     //della pagina
public int doEndTag() throws JspException {
   HttpSession session = pageContext.getSession();
   HttpServletRequest req = (HttpServletRequest)pageContext.getRequest();
   //usa una var protectedPage per memorizzare la pagina richiesta
   //cui fare ritorno dopo l'eventuale redirezione verso la login-page
   String protectedPage = req.getRequestURI();
   if (session.getAttribute("user")==null) {
      session.setAttribute("login-page", loginPage);
      session.setAttribute("error-page", errorPage);
      session.setAttribute("protected-page", protectedPage);
      try {
          pageContext.forward(loginPage);
          return SKIP_PAGE;
          }
      catch (Exception ex) {
            throw new JspException(ex.getMessage());
            }
      }
return EVAL_PAGE; //eseguito se l'attributo user viene trovato nella sessione
}
```

Classe handler del tag **EnforceLogin**

/WEB-INF/classes/tags/EnforceLoginTag.java

```
. . .

        public void release() {
                loginPage=errorPage=null;
                }
}
```

/login.jsp

```
<html><head><title> Login Page </title></head>
<%@taglib uri="/WEB-INF/tlds/security.tld" prefix="security" %>
<body>
  <font size=4 color=red><security:showErrors /> </font>
  <p><font size=5 color=blue">Please login </font> <hr>
  <form action="<%= response.encodeURL("authenticate") %>" method="POST">
  <table>
    <tr>
       <td>Name: </td>
       <td><input type="text" name="userName" /> </td>
    </tr> <tr>
       <td>Password: </td>
       <td><input type="password" name="password" size="8" /> </td>
    </tr> </table>
    <input type="submit" value="login">
   </form> </p>

   Ricorda che un nome valido è: Picasso e password: Pablo
</body></html>
```

/WEB-INF/web.xml

```
<servlet>
       <servlet-name>authenticate</servlet-name>
       <servlet-class> AuthenticateServlet </servlet-class>
</servlet>

<servlet-mapping>
        <servlet-name> authenticate </servlet-name>
        <url-pattern>/authenticate</url-pattern>
</servlet-mapping>
```

/WEB-INF/classes/AuthenticateServlet.java

```
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import beans.LoginDB;
import beans.User;

public class AuthenticateServlet extends HttpServlet {
   private LoginDB loginDB;

   public void init(ServletConfig config) throws ServletException {
      super.init(config);
      loginDB=new LoginDB();
      }
. . .
```

/WEB-INF/classes/AuthenticateServlet.java

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws IOException, SerlvetException {
   HttpSession session=req.getSession(true);
   String uname = req.getParameter("userName");
   String pwd = req.getParameter("password");
   User user = loginDB.getUser(uname,pwd);
//ricerca nella base dati il bean utente con nome e password del form

   if (user != null) { //authorized
     String protectedPage = (String)session.getAttribute("protected-page");
     session.removeAttribute("login-page");
     session.removeAttribute("error-page");
     session.removeAttribute("protected-page");
     session.removeAttribute("login-error");
     //inserisce il bean utente nella sessione
     session.setAttribute("user",user);
     res.sendRedirect(res.encodeURL(protectedPage));
   }
. . .
```

/WEB-INF/classes/AuthenticateServlet.java

```
. . .

//l'utente con i dati digitati nel form non è stato trovato nella base dati
else {//not authorized
   String loginPage = (String) session.getAttribute("login-page");
   String errorPage = (String) session.getAttribute("error-page");
   String forwardTo = errorPage!=null?errorPage:loginPage;
   session.setAttribute("login-error", "Username and pass are not valid");

   //la richiesta viene rediretta alla pagina di errore se è stata
   //configurata, altrimenti alla pagina di login
   getServletContext().getRequestDispatcher(
           res.encodeURL(forwardTo)).forward(req,res);
    }
  }
}
```

/error.jsp

```
<html><head><title> Login Page </title></head>
<%@taglib uri="/WEB-INF/tlds/security.tld" prefix="security" %>
<body>
  <font size=4 color=red>Login Failed because:</font>
   <security:showErrors/> </font>
   Click <a href="login.jsp"> here </a> to retry login.
</body></html>
```

Classe handler del tag **ShowErrors**

/WEB-INF/classes/tags/ShowErrorsTag.java

```
package tags;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;

public class ShowErrorsTag extends TagSupport {
   public int doStartTag() throws JspException {
      String error = (String)pageContext.getSession().
                                    getAttribute("login-error");
      if (error!=null) {
         try {
              pageContext.getOut().print(error);
             }
         catch (java.io.IOException ex) {
            throw new JspException(ex.getMessage());
         }
      }
   return SKIP_BODY;
   }
}
```

Tag che elaborano il proprio contenuto

Interfaccia **BodyTag** (1/6)

- Gli handler di tag con corpo che implementano l'interfaccia **BodyTag** dispongono di due funzionalità mancanti agli altri tag:
  - Possono contenere codice iterativo
  - Possono manipolare il contenuto del loro corpo

- L'interfaccia **BodyTag** estende l'interfaccia **Tag** (estende **IterationTag** che estende **Tag**) definendo i metodi elencati di seguito:

```
1. void setBodyContent()
2. void doInitBody()
3. int doAfterBody()
```

Interfaccia `BodyTag` (2/6)

- Il metodo `doStartTag()` restituisce un valore intero che condiziona l'elaborazione del corpo del tag
  - `SKIP_BODY`: il corpo del tag non deve essere considerato
  - `EVAL_BODY_INCLUDE`: il corpo del tag viene elaborato:
    - **Il corpo viene valutato e passato in output**
    - **Viene invocato il metodo doAfterBody() (eseguito per una o più iterazioni)**
    - **…poi viene invocato doEndTag(), vedi dopo**

Interfaccia **BodyTag** (3/6)

- **EVAL_BODY_BUFFERED**: il corpo del tag viene elaborato e viene creato un oggetto **BodyContent** (sottoclasse di **JspWriter**) utilizzato come oggetto **out**.

- NB: L'oggetto **BodyContent** viene creato esclusivamente se il metodo doStartTag restituisce **EVAL_BODY_BUFFERED**.

Interfaccia **BodyTag** (4/6)

- Il metodo **`setBodyContent()`** configura le proprietà dell'oggetto **`BodyContent`**.
  - Questo metodo non viene invocato per tag vuoti e per i quali il metodo doStartTag() abbia restituito **`SKIP_BODY`** o **`EVAL_BODY_INCLUDE`**.
  - Quando viene invocato, il valore dell'oggetto implicito out viene sostituito nell'oggetto pageContext.
- Il metodo **`doInitBody()`** viene invocato dal container dopo **`setBodyContent`** e prima che il corpo del tag venga valutato per la prima volta.
  - Questo metodo non viene invocato per tag vuoti e per i quali il metodo doStartTag() abbia restituito **`SKIP_BODY`** o **`EVAL_BODY_INCLUDE`**.

Interfaccia `BodyTag` (5/6)

- Il metodo in cui bisogna definire il comportamento per tag che devono modificare/elaborare il corpo è

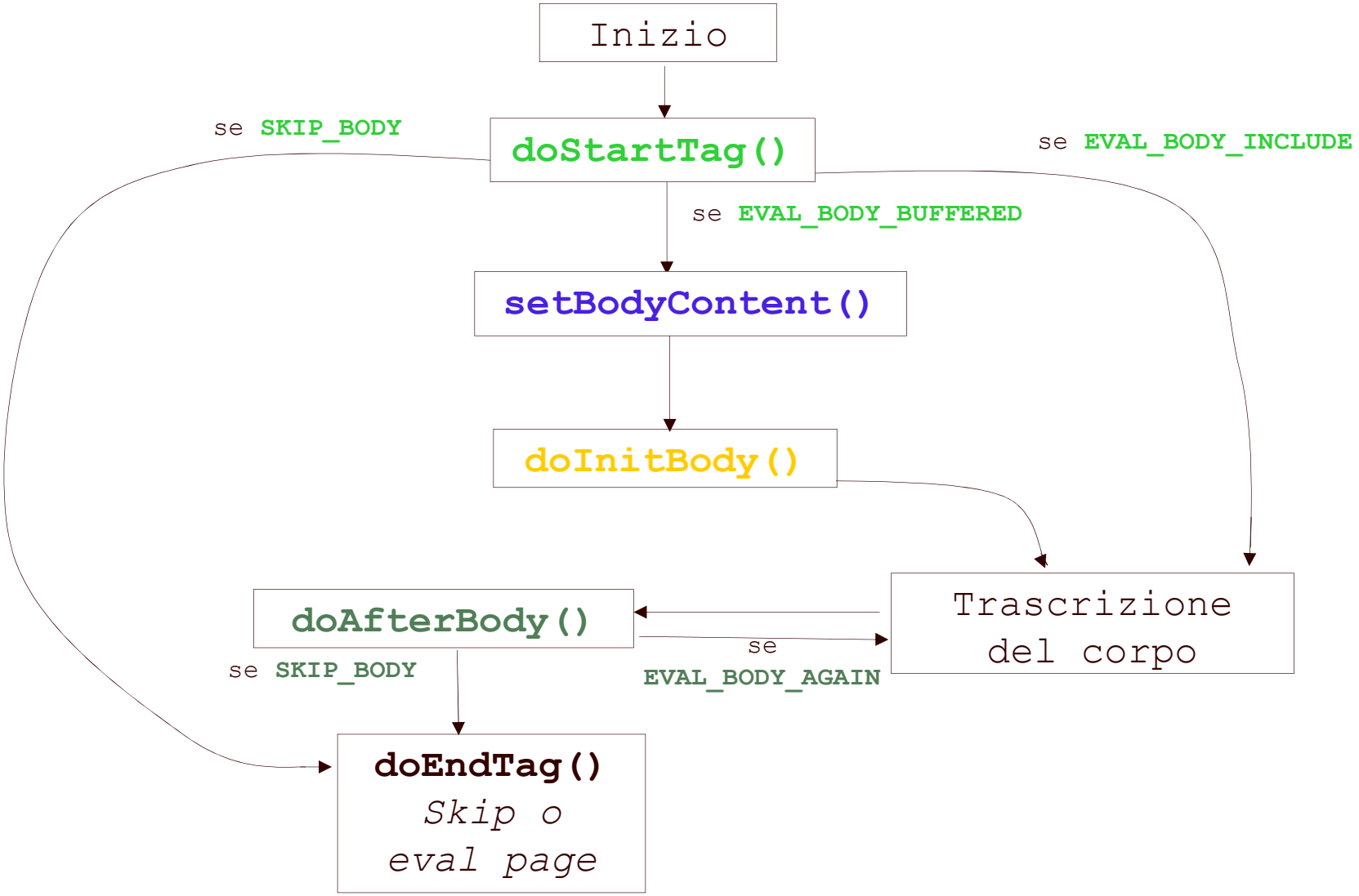
  `doAfterBody()`

  – Tale metodo restituisce un valore intero che condiziona l'elaborazione del tag
    - `SKIP_BODY`: il corpo del tag non deve essere considerato
    - `EVAL_BODY_AGAIN`: il corpo del tag deve essere valutato nuovamente

Interfaccia `BodyTag` (6/6)

- Il metodo `doEndTag()` viene chiamato come per l'interfaccia `Tag` quando il container incontra il tag di chiusura
  - Tale metodo restituisce un valore intero che condiziona l'elaborazione della parte di pagina che segue il tag
    - `SKIP_PAGE`: la parte di pagina oltre il tag di chiusura viene ignorata
    - `EVAL_PAGE`: la parte di pagina oltre il tag di chiusura viene considerata

Ciclo di vita di un tag che implementa l'interfaccia **BodyTag**



Ciclo di vita di un tag che implementa l'interfaccia **BodyTag**

- Il contenitore di servlet richiama i metodi dell'interfaccia **BodyTag** nel seguente modo:

```
if (tag.doStartTag() == EVAL_BODY_BUFFERED) {
    tag.setBodyContent(bodyContent);
    . . .
   tag.doInitBody();}
if ((tag.doStartTag() == EVAL_BODY_INCLUDE)||
    (tag.doStartTag() == EVAL_BODY_BUFFERED))
{
    do {
      // valuta il corpo del tag
   }
   while (tag.doAfterBody() == EVAL_BODY_AGAIN);
}
tag.doEndTag();
```

La classe `BodyTagSupport` (1/2)

- La classe `BodyTagSupport` estende la classe **`TagSupport`** e implementa l'interfaccia **`BodyTag`**
- Questa classe introduce i nuovi metodi
  - **`BodyContent`** `getBodyContent()`

    restituisce il contenuto del corpo di un tag
  - **`JspWriter`** `getPreviousOut()`

    restituisce lo scrittore associato al tag genitore o la variabile implicita **`out`** se il tag è di livello superiore.

La classe `BodyTagSupport` (2/2)

- Per impostazione predefinita le estensioni di **`BodyTagSupport`** valutano il corpo del tag una volta soltanto
- Valori predefiniti restituiti dai metodi **`BodyTagSupport`**
  - **`doStartTag()`:** **`EVAL_BODY_BUFFERED`**
  - **`doAfterBody()`:** **`SKIP_BODY`**
  - **`doEndTag()`:** **`EVAL_PAGE`**

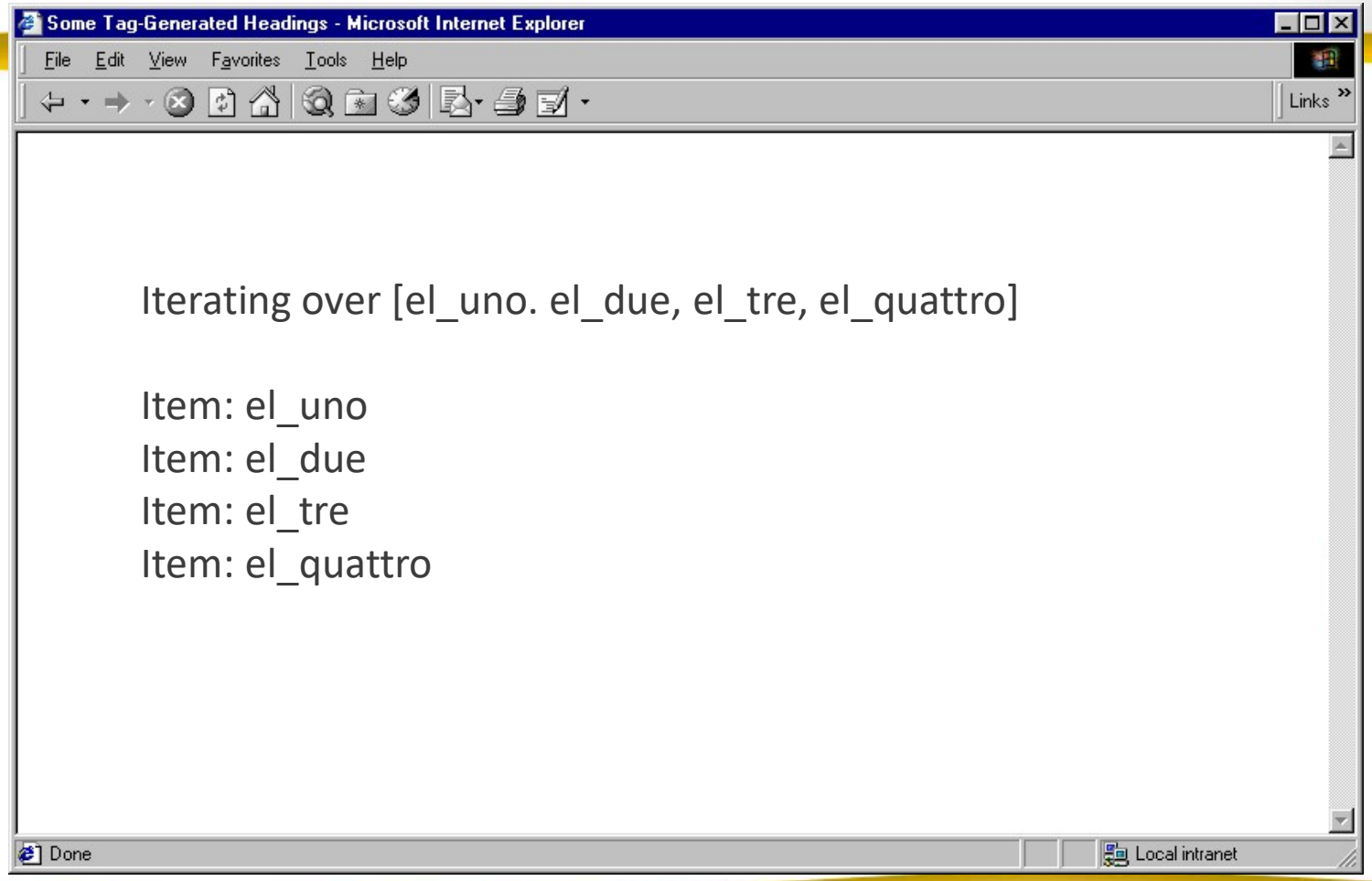
Nota sulla specifica JSP 1.2

- Secondo la nuova specifica è possibile scrivere tag iterativi anche estendendo la classe TagSupport (metodo doAfterBody) ma non viene creato mai l'oggetto BodyContent (si risparmia quando non è strettamente necessario usarlo)
- TagSupport implementa l'interfaccia IterationTag (quest'ultima estende l'interfaccia Tag che invece non consente iterazioni del tag), rendendo inutile l'uso di BodyTag se doStartTag restituisce EVAL_BODY_INCLUDE

Funzionamento del contenuto del corpo

- Il contenuto del corpo è rappresentato dalla classe **`BodyContent`** (scrittore con buffer)
- La classe **`BodyContent`** estende **`JspWriter`** (il tipo della variabile implicita **`out`**)
- I contenitori di servlet mantengono uno stack di oggetti **`BodyContent`** per fare in modo che un tag annidato non sovrascriva il contenuto del corpo di uno dei tag antenati
- Ciascun oggetto **`BodyContent`** conserva un riferimento allo scrittore con buffer del livello inferiore nello stack.
- Tale scrittore è noto come *previous out*, o *scrittore allegato*, ed è disponibile attraverso
  - **`BodyContent.getEnclosingWriter`** oppure
  - **`BodyTagSupport.getPreviousOut`**

Esempio: Iterazione (1/5)



Esempio: Iterazione /test.jsp (2/5)

```
<html><head><title>Un iteratore</title></head>
<%@ taglib uri="/WEB-INF/tlds/iterator.tld" prefix="it" %>
<body>

<% java.util.Vector vector = new java.util.Vector();
   vector.addElement("el_uno"); vector.addElement("el_due");
   vector.addElement("el_tre"); vector.addElement("el_quattro");
%>

Iterating over <%= vector %> ...
<p>
<it:iterate collection="<%= vector %>">
        <jsp:useBean id="item" scope="page" class="java.lang.String"/>
        Item: <%= item %><br>
</it:iterate>
</p>
</body>
</html>
```

Esempio: Iterazione (3/5)

/WEB-INF/classes/tags/IteratorTag.java

```
package tags;

import java.util.Collection;
import java.util.Iterator;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.BodyTagSupport;

public class IteratorTag extends BodyTagSupport{
        private Collection collection;
        private Iterator iterator;

        public void setCollection (Collection collection) {
           this.collection=collection;
        }

        public int doStartTag() throws JspException {
           return collection.size() > 0? EVAL_BODY_BUFFERED : SKIP_BODY;
        }
// . . . segue
```

Esempio: Iterazione (4/5)

IteratorTag.java ... segue

```
public void doInitBody() throws JspException {
  iterator=collection.iterator();
  pageContext.setAttribute("item", iterator.next());
}

public int doAfterBody() throws JspException {
   if (iterator.hasNext()) {
   pageContext.setAttribute("item", iterator.next());
   return EVAL_BODY_AGAIN;
   }
   else {
      try {
         getBodyContent().writeOut(getPreviousOut());
      }
      catch (java.io.IOException e) {
         throw new JspException (e.getMessage());
      }
      return SKIP_BODY;
   }
}
}
```

Esempio: Iterazione (5/5)
/WEB-INF/tlds/iterator.tld

```
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
    <tag>
              <name> iterate </name>
              <tagclass> tags.IteratorTag </tagclass>
              <bodycontent> JSP </bodycontent>
              <attribute>
                       <name> collection </name>
                       <required> true </required>
                       <rtexprvalue> true </rtexprvalue>
              </attribute>
              <info>
              Scrive iterativamente gli elementi di una collezione
              </info>
     </tag>
</taglib>
```

Esercizio

- Scrivere un'applicazione che preveda una pagina voti.jsp contenente un form attraverso il quale l'utente possa inserire una coppia di elementi <nome_esame, voto>.
- La coppia deve essere acquisita all'interno di un vettore di coppie.
- La stessa pagina deve stampare in fondo al form un riassunto di quanto già inserito nel vettore, facendo uso di un tag personalizzato che iterando sugli elementi del vettore li elenchi a video.
- Lo stesso tag personalizzato deve stampare la media degli esami già inseriti.