Lezione del 24 Maggio 2010

Docente: Novella Bartolini

Tag Personalizzati

- Sono la caratteristica più potente delle pagine jsp
  - Incapsulano funzionalità complesse
  - Permettono agli sviluppatori di software e agli autori di pagine di lavorare in modo autonomo

- Le funzionalità dei tag vengono definite all'interno di classi java che implementano l'interfaccia Tag
  - Package javax.servlet.jsp.tagext
  - (includere il file jsp-api.jar nel CLASSPATH per poter compilare le classi che definiscono i tag)

- In alcuni casi l’azione <jsp:useBean> può realizzare gli stessi obiettivi di un tag personalizzato ma…
  - I bean non possono manipolare il contenuto JSP

    (es: tag di “sicurezza”

    <jsp:reserved usertype=“amministratore”>

    pannello amministratore …

    </jsp:reserved>

    può consentire la visualizzazione del pannello al solo utente amministratore)
  - Operazioni molto complesse possono essere espresse in una forma molto più semplice utilizzando tag personalizzati

Definire ed utilizzare tag personalizzati

1. Definire, per ogni tag, una **classe handler** del tag che ne implementi le funzionalità
2. Definire un **descrittore della libreria** di tag (TLD)
3. Scrivere le pagine JSP che fanno uso della libreria dei tag personalizzati
   - Invocare la direttiva per la localizzazione del descrittore della libreria dei tag
   - Utilizzare i tag all'interno della pagina

Tag personalizzati - come si usano

1. Scrivere la classe handler del tag personalizzato
2. Scrivere il descrittore di libreria
3. Scrivere la pagina JSP che utilizza il tag personalizzato

- E’ la classe che contiene la logica del tag
- E’ una classe Java che implementa l’**interfaccia Tag**
- Quando il container incontra un tag personalizzato
  - Crea l’oggetto tag handler
  - Invoca i metodi dell’interfaccia Tag necessari ad attivare la logica del tag.

- Le funzionalità dei tag personalizzati sono definite all’interno di classi java che implementano l’interfaccia Tag
  - In genere si estendono le classi TagSupport oppure BodyTagSupport che implementano l’interfaccia Tag
    - **TagSupport viene utilizzata per tag che non elaborano il contenuto del proprio body**
    - **BodyTagSupport viene utilizzata per tag che elaborano il contenuto del proprio body**
- L’interfaccia Tag fornisce i metodi che vengono invocati dal container durante l’elaborazione del tag

E’ definita nel package

- javax.servlet.jsp.tagext

Definisce sei metodi

1. void **setPageContext**(PageContext)*
2. void **setParent**(Tag)
3. int **doStartTag**() throws JspException
4. int **doEndTag**() throws JspException
5. void **release**()
6. Tag **getParent**()

> Vedremo tra poco questi metodi
>
> nel dettaglio,
>
> per ora procediamo per esempi

I metodi 1-5 vengono invocati dal container nell’ordine indicato quando viene elaborato un tag all’interno di una pagina jsp

* Dalla documentazione: “A PageContext instance provides access to all the namespaces associated with a JSP page, provides access to several page attributes, as well as a layer above the implementation details. Implicit objects are added to the pageContext automatically”.

Come definire un tag handler

- I tag handler devono obbligatoriamente essere definiti all'interno di package
- Si definisce l'implementazione dei metodi dell'interfaccia Tag che si ritengono utili all'elaborazione dei tag personalizzati
- Tutte le eccezioni che possono verificarsi durante l'elaborazione di un tag devono essere catturate e deve essere lanciata un'eccezione **JspException**

```java
1    // WelcomeTagHandler.java
2    // Custom tag handler that handles a simple tag.
3    package miei_tag;
4
5    // Java core packages
6    import java.io.*;
7
8    // Java extension packages
9    import javax.servlet.jsp.*;
10   import javax.servlet.jsp.tagext.*;
11
12   public class WelcomeTagHandler extends TagSupport {
13
14      // Method called to begin tag processing
15      public int doStartTag() throws JspException
16      {
17         // attempt tag processing
18         try {
19            // obtain JspWriter to output content
20            JspWriter out = pageContext.getOut();
21
22            // output content
23            out.print( "Messaggio proveniente dal tag" );
24         }
25
26         // rethrow IOException to JSP container as JspException
27         catch( IOException ioException ) {
28            throw new JspException( ioException.getMessage() );
29         }
30
31         return SKIP_BODY;  // ignore the tag's body (alternativa a EVAL_BODY_INCLUDE)
32      }
33   }
```

Class **WelcomeTagHandler** implementa l’interfaccia **Tag** estendendo la classe **TagSupport**

Il JSP container invoca il metodo **doStartTag** quando incontra l’apertura di un tag personalizzato

Viene utilizzato l’oggetto **pageContext** ereditato da TagSupport, per ottenere l’oggetto **JspWriter** necessario per scrivere il testo di output

Tag personalizzati - come si usano

1. Scrivere la classe handler del tag personalizzato
2. Scrivere il descrittore di libreria
3. Scrivere la pagina JSP che utilizza il tag personalizzato

Direttiva taglib

- Questa direttiva è necessaria per poter utilizzare tag personalizzati
- Individua il percorso di un **descrittore della libreria** e un **prefisso** che verrà utilizzato per accedere agli elementi della libreria

Attribute	Description
`uri`	Specifica il percorso relativo o assoluto del tag library descriptor (tld).
`prefix`	Specifica il prefisso richiesto per distinguere i tag personalizzati dai tag built-in. I prefissi **`jsp`**, **`jspx`**, **`java`**, **`javax`**, **`servlet`**, **`sun`** e **`sunw`** sono riservati.

<%@ taglib uri="/WEB-INF/tlds/mialib.tld" prefix="util" %>

- Un descrittore di libreria di tag (TLD) è un documento XML che definisce una libreria di tag e i tag in essa contenuti

```
<taglib>
   <tlibversion>1.0</tlibversion>
   <jspversion>1.1</jspversion>
   <shortname>Libreria Personale</shortname>
   <info> Una semplice libreria di tag di generica utilità</info>

   <tag>
      <name>welcome</name>
      <tagclass> miei_tag.WelcomeTagHandler </tagclass>
      <bodycontent>empty</bodycontent>
      <info> Inserisce un testo di benvenuto </info>
   </tag>

</taglib>
```

```
<taglib>
   <tlibversion>1.0</tlibversion>
   <jspversion>1.1</jspversion>
   <shortname>Libreria Personale</shortname>
   <info> Una semplice libreria di tag di generica utilità</info>

  <tag>
     <name>welcome</name>
     <tagclass> miei_tag.WelcomeTagHandler </tagclass>
     <bodycontent>empty</bodycontent>
     <info> Scrive un messaggio di benvenuto</info>
  </tag>

</taglib>
```

- **<tlibversion>** versione della libreria
- **<jspversion>** versione della specifica JSP
- **<shortname>** e **<info>** descrizioni

File TLD: descrittore … (cont.)

- I tag sono definiti attraverso l’elemento **<tag>** che ha a sua volta due elementi obbligatori:
  - **<name>:** Il nome del tag così come viene utilizzato nella pagina JSP
  - **<tagclass>**: La classe Java che implementa la funzionalità del tag (tag handler)
- L’ulteriore elemento **<body-content>** specifica il tipo di contenuto del tag, può essere empty, tagdependent oppure JSP.
- Deve essere presente un elemento <tag> per ogni tag personalizzato della libreria

Tag personalizzati - come si usano

1. Scrivere la classe handler del tag personalizzato
2. Scrivere il descrittore di libreria
3. Scrivere la pagina JSP che utilizza il tag personalizzato

```
<!– tagdibenvenuto.jsp →
<html>
  <head>
  <title> Pagina che visualizza un messaggio di benvenuto</title>
  </head>

  <body>

  <%@ taglib uri="/WEB-INF/tlds/mialib.tld" prefix="util" %>

  Questo messaggio: <b><util:welcome/></b> <br>
  è stato prodotto da un tag personalizzato.

  </body>
</html>
```

- Copiare la classe handler del tag con tutto il suo package nella directory

  \WEB-INF\classes della web application

- Mettere il file TLD nel percorso indicato con l'attributo **uri** della direttiva **taglib**

- Mettere il file tagdibenvenuto.jsp in una qualsiasi cartella della web application

Custom tag con attributi

- I tag personalizzati possono avere un qualunque numero di attributi, obbligatori o facoltativi specificati come *attributo*=*valore*
  - Es: <util:iterate times="4">

1. Scrivere il tag con il relativo attributo nel file JSP
2. Aggiungere un tag di attributo al TLD
3. Implementare il metodo setAttributo nell'handler del tag

- Quando il container incontra il tag crea l’oggetto tag handler e invoca i metodi setter necessari per impostare i valori degli attributi

- E’ prassi comune implementare anche un metodo getter nell’handler del tag per permettere ai tag annidati di accedere alle proprietà degli altri.

- Il file TLD deve contenere per ogni tag l'elenco degli attributi ad esso relativi, ciascuno specificato attraverso un elemento **<attribute>**

- L'elemento <attribute> contiene a sua volta gli elementi:
  - <name> *(nome dell'attributo)*
  - <required> *(indica se l'attributo è necessario o opzionale)*
  - <rtexprvalue> *(indica se l'attributo deve essere specificato come stringa o se è permessa l'elaborazione a tempo di esecuzione)*

```
1    <?xml version = "1.0"?>
2    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5    <!-- customTagAttribute.jsp             -->
6    <!-- JSP that uses a custom tag to output content. -->
7
8    <%-- taglib directive --%>
9    <%@ taglib uri = "mialib.tld" prefix = "util" %>
10
11   <html>
12
13      <head>
14         <title>Utilizzo di tag personalizzati con attributi</title>
15      </head>
16
17      <body>
18         <p>Uso di un attributo dichiarato come stringa</p>
19         <h1>
20            <util:welcome2 firstName = "Paul" />
21         </h1>
22
23         <p>Valutazione del valore di un attributo a tempo di esecuzione</p>
24         <h1>
25            <%-- scriptlet to obtain "name" request parameter --%>
26            <%
27               String name = request.getParameter( "name" );
28            %>
29
30            <util:welcome2 firstName = "<%= name %>" />
31         </h1>
32      </body>
33
34   </html>
```

Pagina JSP che usa un tag con un attributo firstName

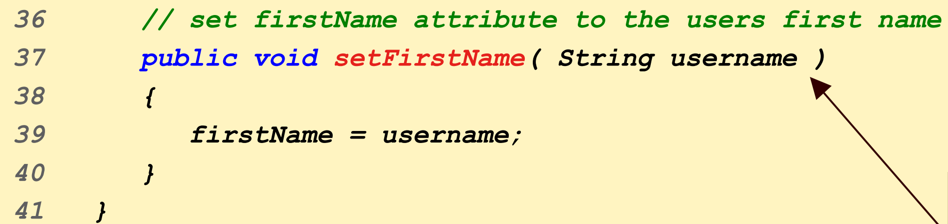
Viene usato il tag personalizzato **welcome2** per inserire un testo nella JSP concatenando il valore dell'attributo **firstName**

```java
1     // Welcome2TagHandler.java
2     // Classe tag handler che gestisce un tag con un attributo
3     package miei_tag;
4
5     // Java core packages
6     import java.io.*;
7
8     // Java extension packages
9     import javax.servlet.jsp.*;
10    import javax.servlet.jsp.tagext.*;
11
12    public class Welcome2TagHandler extends TagSupport {
13       private String firstName = "";
14
15       // Method called to begin tag processing
16       public int doStartTag() throws JspException
17       {
18          // attempt tag processing
19          try {
20             // obtain JspWriter to output content
21             JspWriter out = pageContext.getOut();
22
23             // output content
24             out.print( "Hello " + firstName +
25                ", <br />Welcome to JSP Tag Libraries!" );
26          }
27
28          // rethrow IOException to JSP container as JspException
29          catch( IOException ioException ) {
30             throw new JspException( ioException.getMessage() );
31          }
32
33          return SKIP_BODY;  // ignore the tag's body
34       }
35
```

Classe handler di un tag con un attributo
firstName

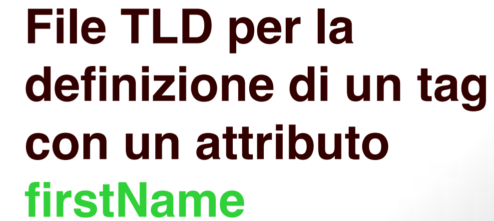
Definizione dell'attributo **firstName**

Uso dell'attributo **firstName** per produrre un messaggio di output attraverso il tag personalizzato



```
36     // set firstName attribute to the users first name
37     public void setFirstName( String username )
38     {
39        firstName = username;
40     }
41  }
```

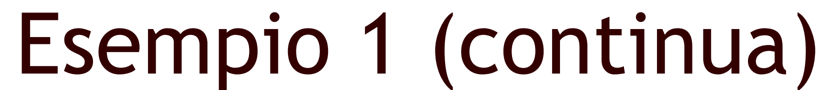
Metodo *setter* per l’attributo **`firstName`**




```
1  <!-- A tag with an attribute -->
2     <tag>
3        <name>welcome2</name>
4
5        <tagclass>
6           miei_tag.Welcome2TagHandler
7        </tagclass>
8
9        <bodycontent>empty</bodycontent>
10
11       <info>
12          Inserisce un messaggio di benvenuto
13          usando l'attributo "name" per inserire il nome dell'utente
14       </info>
15
16       <attribute>
17          <name>firstName</name>
18          <required>true</required>
19          <rtexprvalue>true</rtexprvalue>
20       </attribute>
21    </tag>
```

Si introduce l'elemento **`attribute`** per definire le modalità di uso dell'attributo **`firstName`**

- Form contenente diversi campi di testo.
- Se il form non è stato compilato correttamente viene riproposto all'utente e i campi del form già compilati vengono rivisualizzati in modo che l'utente debba immettere solo quelli mancanti.



- Un primo tentativo potrebbe essere:

```
<input type="text"
size=15
nome="firstName"
value="<%=request.getParameter('firstName')%>">
```

- Inconveniente:
  - se nessun parametro di richiesta corrisponde al nome dei campi (es. prima visualizzazione del form) viene visualizzato il valore null
- Soluzione: implementare un tag personalizzato che
  - resituisce il valore del parametro della richiesta, se esiste, e una stringa vuota in caso contrario

Esempio 1: /register.jsp

```
<%@ taglib uri="WEB-INF/tlds/html.tld" prefix="form_util" %>
. . .
<table>
   <tr>
      <td> Nome: </td>
      <td> <input type="text" size=15 name=firstName"
            value="<form_util:requestParameter property='firstName'/>">
      </td>
   </tr>
   <tr>
      <td> Cognome: </td>
      <td> <input type="text" size=15 name=lastName"
            value="<form_util:requestParameter property='lastName'/>">
      </td>
   </tr>
   <tr>
      <td> Email: </td>
      <td> <input type="text" size=25 name=emailAddress"
            value="<form_util:requestParameter property='emailAddress'/>">
      </td>
   </tr>
</table>
. . .
```

```
. . .
<taglib>
   . . .
   <tag>
      <name>requestParameter</name>
      <tagclass>miei_tag.GetRequestParameterTag</tagclass>
      <bodycontent>empty</bodycontent>

      <attribute>
          <name>property</name>
          <required>true</required>
          <rtexprvalue>true</rtexprvalue>
      </attribute>
   </tag>
</taglib>
```

Esempio 1: /WEB-INF/classes/miei_tag/GetRequestParameterTag.java

```
package miei_tag;
import javax.servlet.ServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
public class GetRequestParameterTag extends TagSupport {
  private String property;
  public void setProperty(String valore) {
  this.property=valore;
  }
  public int doStartTag() throws JspException {
    ServletRequest req=pageContext.getRequest();
    String value= req.getParameter(property);
    try {
       pageContext.getOut().print(value==null? "":value);
    }
    catch (java.io.IOException ex) {
       throw new JspException(ex.getMessage());
    }
    return SKIP_BODY;
  }
}
```

Documentazione sui tag personalizzati

- Sul vostro pc all'indirizzo:

http://127.0.0.1:8080/tomcat-docs/jspapi/index.html

Ricapitolando:
per definire una libreria di "**custom tag**" (tag personalizzati)

- classe Handler di un Tag
  - Codice Java che stabilisce come tradurre il tag in codice
  - Implementa l'interfaccia javax.servlet.jsp.tagext.Tag oppure javax.servlet.jsp.tagext.BodyTag
  - Estende TagSupport or BodyTagSupport
  - Va messa in …/WEB-INF/classes insieme a servlet e beans (in un opportuno package)
- file TLD (= Tag Library Descriptor)
  - File XML che descrive il nome del tag, i suoi attributi, e la classe handler che lo implementa
  - Può essere messo insieme al file JSP che usa la libreria oppure ad un altro URL
- file JSP
  - Importa una libreria di tag (specificando l'URL di un file TLD in una direttiva taglib)
  - Definisce il prefisso da associare ai tag della libreria (specificando il prefisso nella direttiva taglib)
  - Utilizza i tag

Tag personalizzati e contesto della pagina (PageContext)

- I tag hanno accesso alle informazioni della pagina tramite l'oggetto **pageContext** (istanza di **PageContext**)
- La classe **PageContext** prevede un insieme di metodi per l'accesso agli oggetti impliciti nell'ambito di una pagina (come richiesta, session, scrittore per produrre l'output sulla pagina ecc.)

1. **void setPageContext(PageContext)**
2. **void setParent(Tag)**
3. **int doStartTag() throws JspException**
4. **int doEndTag()**
5. **void release()**

6. **Tag getParent()**

- I metodi del primo gruppo (1-5) vengono chiamati nell’ordine dal contenitore di servlet (per questo si parla di *ciclo di vita di un tag*)

Ciclo di vita di un tag (interf. Tag) (1/3)

1. Il metodo **setPageContext(PageContext)** configura il contesto di pagina associato al tag
2. Il metodo **setParent(Tag)** associa un genitore al tag:
   - Tutti i tag hanno un genitore, che è **null** per i tag di livello superiore ed è il tag contenitore per i tag annidati

```
<esempio:tag_esterno>
        <esempio:tag_intermedio>
              <esempio:tag_interno>
              …
              </esempio:tag_interno>
        </esempio:tag_intermedio>
</esempio:tag_esterno>
```

> tag_intermedio è genitore di tag_interno
>
> tag_esterno è genitore di tag_intermedio
>
> Il genitore di tag_esterno è null

- Entrambi questi metodi sono rivolti a coloro che implementano il contenitore delle servlet e non agli sviluppatori JSP

Ciclo di vita di un tag (interf. Tag) (2/3)

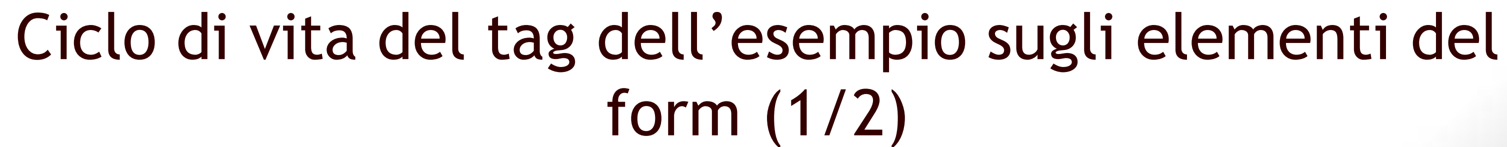
3. Se il tag prevede degli attributi, vengono invocati tutti i metodi necessari a configurarne i valori (set)
4. Il metodo **doStartTag()** viene invocato subito dopo i primi due (**setPageContext** e **setParent**) e gli eventuali metodi set
   - Tale metodo restituisce un valore intero che condiziona l’elaborazione del tag
     - **SKIP_BODY**: il corpo del tag non viene considerato
     - **EVAL_BODY_INCLUDE**: il corpo del tag deve essere trascritto invariato

4. Il metodo **doEndTag()** viene chiamato in corrispondenza del 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
5. Il metodo **release()** rilascia del risorse dell’handler del tag

- La classe ***tagSupport*** implementa l'interfaccia Tag e aggiunge alcuni attributi e metodi,

fare riferimento alla documentazione

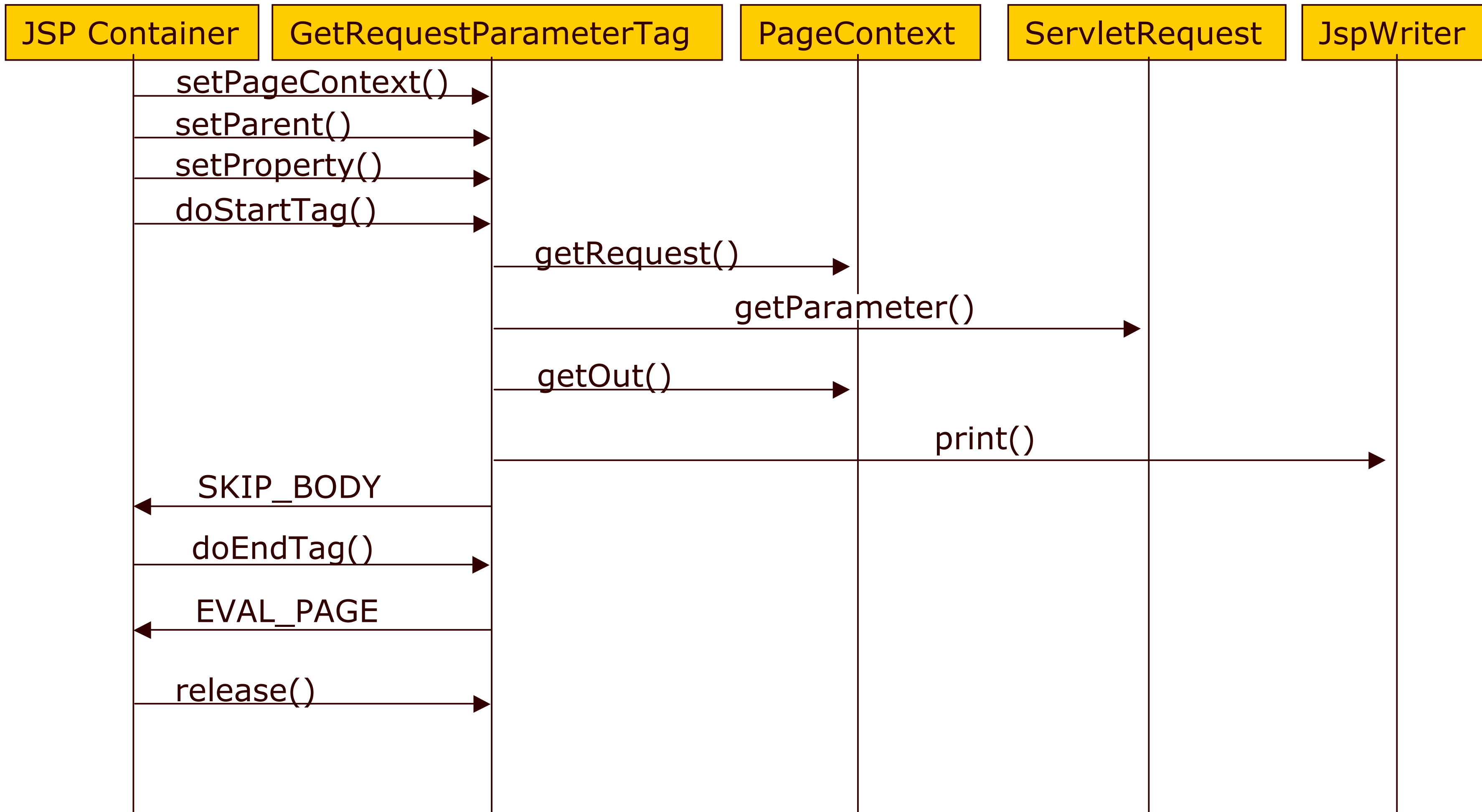
```
protected String id
protected PageContext pageContext

Object getValue(String key)
void  setValue(String key, Object value)
void removeValue(String key)
Enumeration getValues()

String getID()
void setID()
```



Ciclo di vita del tag dell’esempio sugli elementi del form (1/2)

```
package miei_tag;
import javax.servlet.ServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
public class GetRequestParameterTag extends TagSupport {
  private String property;
  public void setProperty(String valore) {
  this.property=valore;
  }
  public int doStartTag() throws JspException {
    ServletRequest req=pageContext.getRequest();
    String value= req.getParameter(property);
    try {
       pageContext.getOut().print(value==null? “”:value);
    }
    catch (java.io.IOException ex) {
       throw new JspException(ex.getMessage());
    }
    return SKIP_BODY;
  }
}
```

Ciclo di vita del tag dell'esempio sugli elementi del form (2/2)



- 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

```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 BORDER=" + border +
                " BGCOLOR=\"" + bgColor + "\"" +
                " ALIGN=\"" + align + "\"");
      if (width != null) {
        out.print(" WIDTH=\"" + width + "\"");
      }
      ...
    } 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
  }
```

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>
...
```

Some Tag-Generated Headings - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Links

Default Heading

White on Black Heading

Large Bordered Heading

Heading with Full-Width Background

Heading with Non-Standard Font

Done

Local intranet

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

```java
package beans;

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

   //costruttore
   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>
```

```
<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>
```

/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>
```

```
<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();
      }
. . .
```

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws IOException, SerlvetException {
   HttpSession session=req.getSession();
   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);
    }
  }
}
```

```
<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

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

Interfaccia BodyTag (2/6)

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

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**.

- 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**.

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

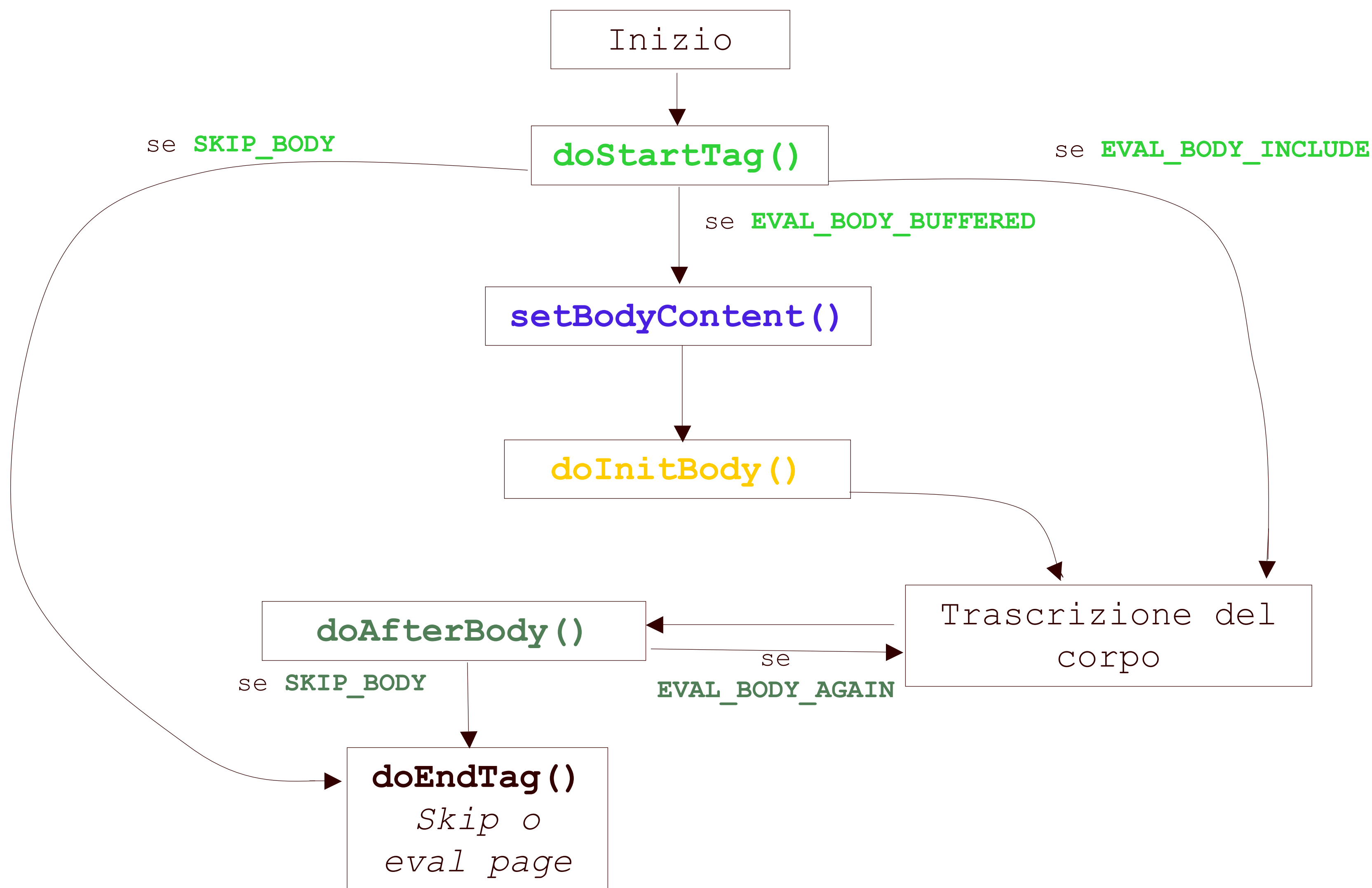
  `doAfterBody()`

  - 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)

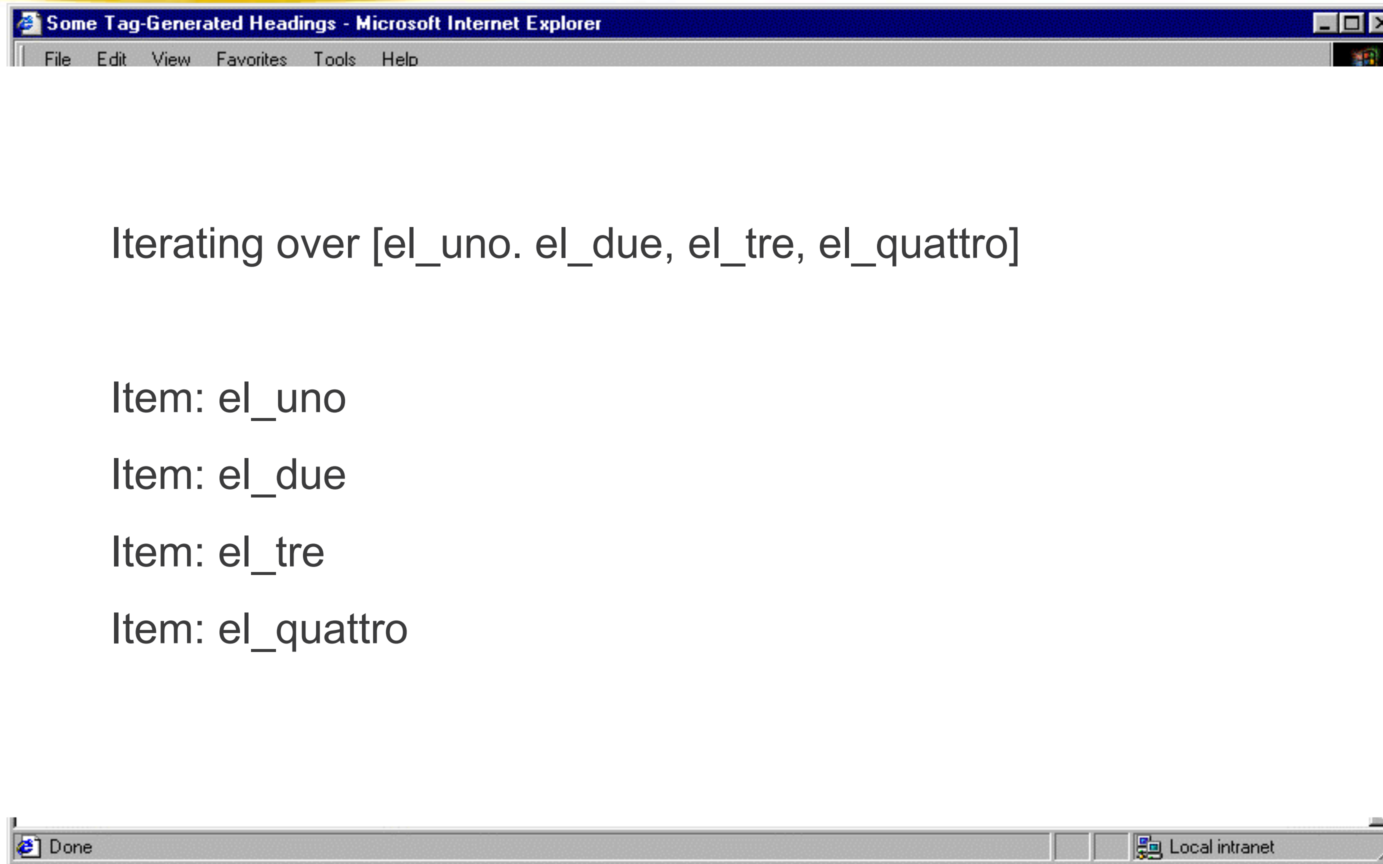
- Per impostazione predefinita le estensioni di **BodyTagSupport** valutano il corpo del tag una volta soltanto
- Valori predefiniti restituiti dai metodi **BodyTagSupport**
  - **doStartTag():** **EVAL_BODY_BUFFERED**
  - **doAfterBody():** **SKIP_BODY**
  - **doEndTag():** **EVAL_PAGE**

- 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)



```
<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)

```
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)

```
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>
```