SVILUPPO CON LA TECNOLOGIA JAVA SERVER PAGES

Tecnologia JSP come meccanismo di presentazione

- Le pagine JSP sono documenti basati su testo che descrivono come elaborare una richiesta e creare una risposta.
  - Gli sviluppatori di pagine possono utilizzare la tecnologia JSP per creare un documento per la generazione di contenuto dinamico.
  - Elementi JSP:
    - Abilitare l'accesso agli oggetti esterni
    - Aggiungere funzionalità di programmazione create in precedenza
  - I file di origine per le pagine JSP generalmente terminano con l'estensione `.jsp`.

Tecnologia JSP come meccanismo di presentazione

- Tecnologia JSP:
  - Utilizza i bean per interagire con gli oggetti sul lato server
  - Utilizza le librerie di tag per sviluppare ed estendere le funzionalità create in precedenza fornite dalle azioni
  - Consente un elevato livello di separazione tra il contenuto statico e quello dinamico in una pagina JSP
  - Offre (laddove necessario) un linguaggio di script avanzato per le pagine JSP
  - È parte integrante della piattaforma Java EE, pertanto offre accesso front-end ai componenti EJB

Confronto tra la presentazione che utilizza pagine JSP e quella che utilizza i servlet

- Le pagine JSP sono componenti Web che si basano sul modello di servlet e che vengono eseguiti come servlet:

Caratteristica	JSP	Servlet
Modello di richiesta e risposta	Stesso modello	
Utilizzo del linguaggio di markup	Sì, simile al linguaggio HTML	No, utilizza le istruzioni `println`
Runtime	Presentano vantaggi di runtime simili rispetto ai tool di script, come l'interfaccia CGI	

Confronto tra la presentazione che utilizza pagine JSP e quella che utilizza i servlet

Caratteristica	JSP	Servlet
Funzionalità	Analoghe ad altre tecnologie a contenuto dinamico	La risposta HTML generata è testo statico creato dalle istruzioni `println`.
Ricompilazione automatica	Sì	No
Competenze degli sviluppatori	Le pagine JSP possono essere create da sviluppatori non software.	I servlet vengono creati da sviluppatori software.
Debug	Test di layout e presentazione più semplice e veloce Debug più complesso	Test dei layout più complesso Debug più semplice dei problemi della fase di compilazione e delle eccezioni di runtime

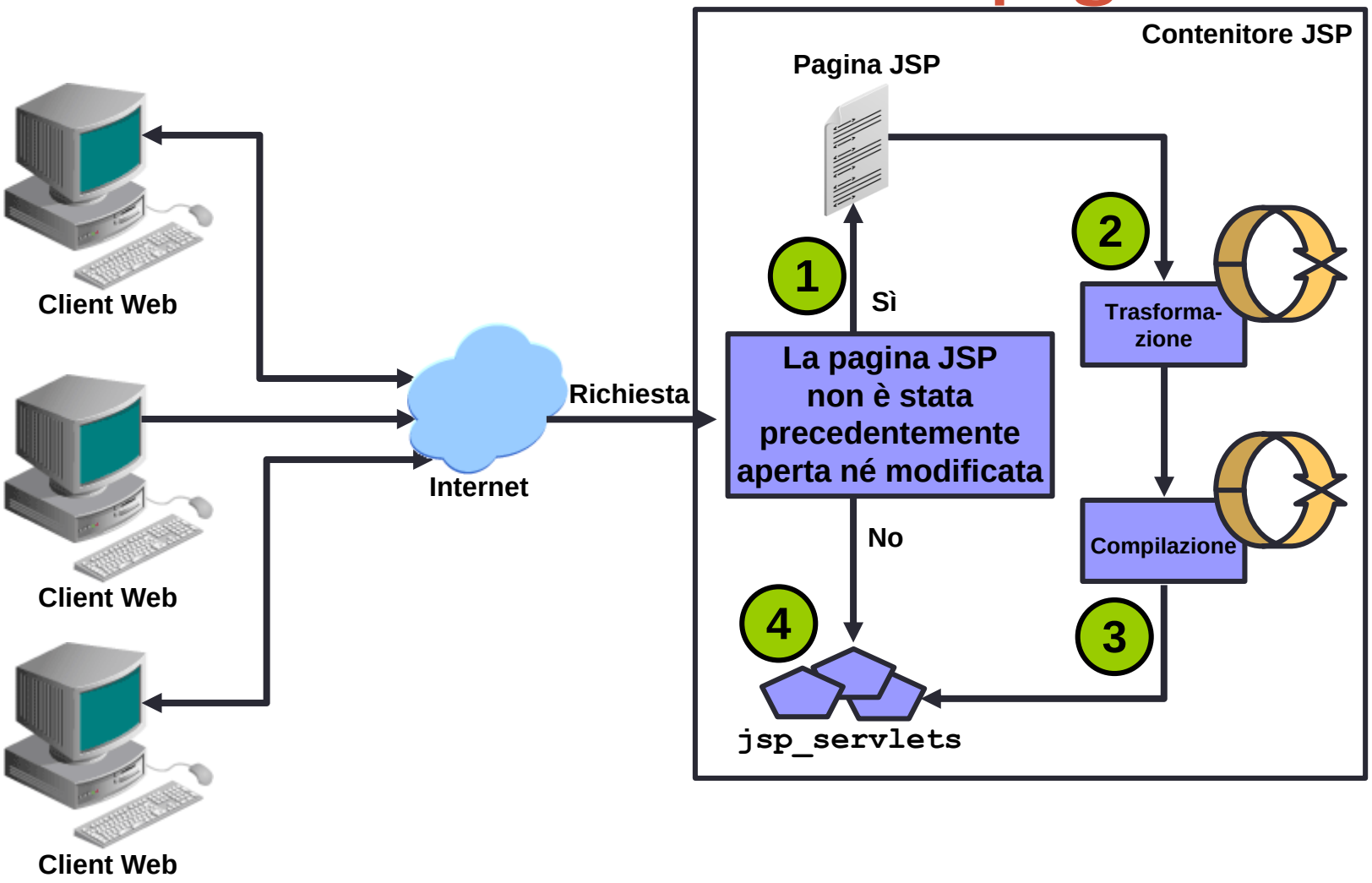
Bean di lavoro, JSTL e tag personalizzati

- È possibile separare la funzionalità di programmazione dalla presentazione nei componenti JSP in due modi:
  - Incorporare le classi con il tag `<jsp:useBean>` e la libreria JSTL.
    - Utile per il trasporto dei dati nel componente JSP
    - JSTL consente il comportamento programmatico senza scriptlet
  - Utilizzare librerie di tag personalizzate.
    - Utile soprattutto quando le librerie di tag sono generiche e riutilizzabili
    - Meno utile per la logica specifica delle pagine, ad esempio per l'elaborazione dei form univoci

Meccanismo di distribuzione delle pagine JSP

- Pagine JSP:
  - Vengono convertite in servlet su richiesta
  - Possono essere distribuite con le stesse modalità previste per una pagina HTML, ovvero copiando il file nel server

Procedura di conversione delle pagine JSP



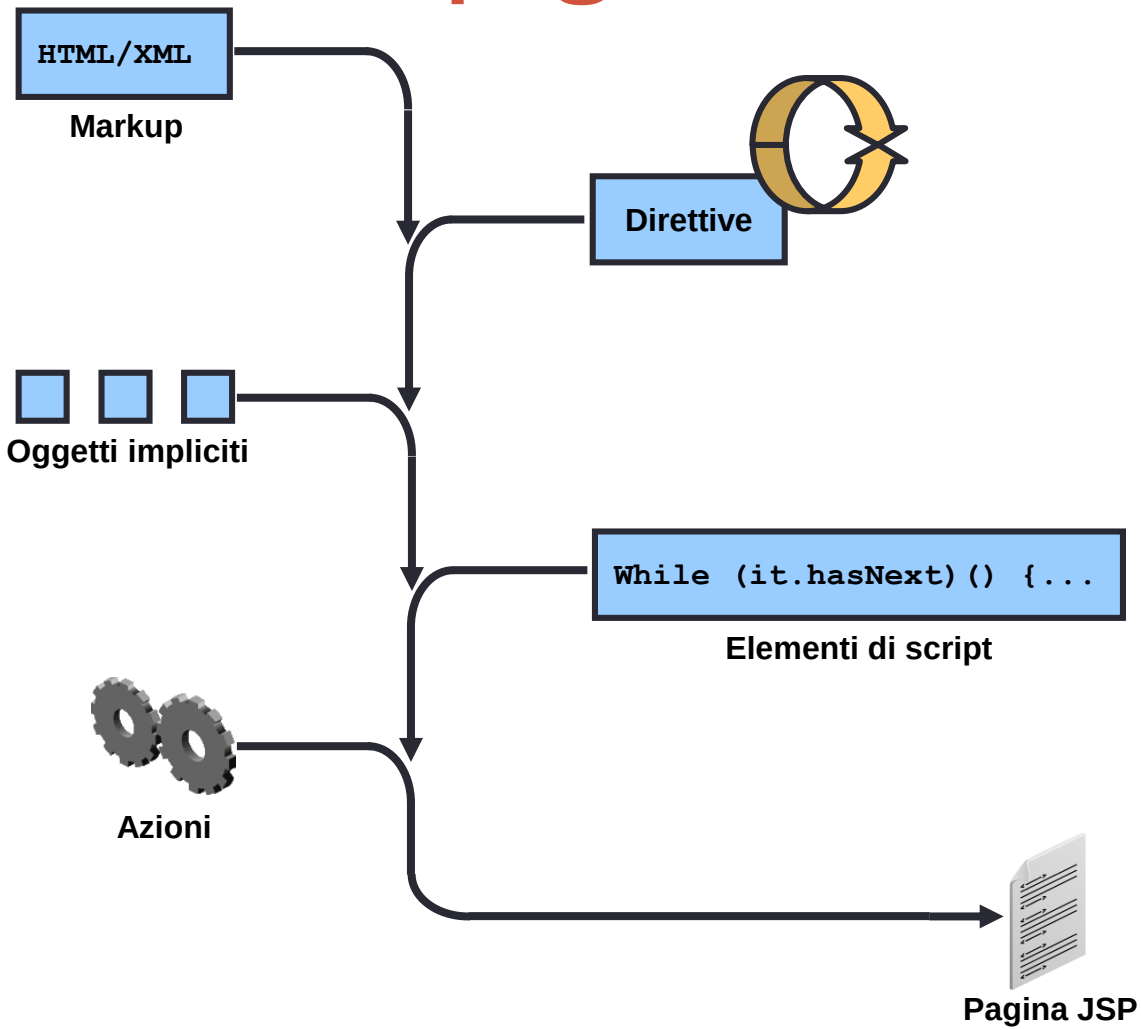
Codice Java incorporato nelle pagine JSP

- È preferibile che una pagina JSP venga utilizzata solo per la logica di presentazione. Per l'elaborazione della logica e per il controllo del flusso il servlet costituisce un'alternativa più valida. L'utilizzo del codice di script per l'elaborazione della logica e il controllo del flusso in una pagina JSP comporta problemi, tra cui quelli riportati di seguito.
  - Un autore di pagine JSP:
    - Deve creare codice corretto nel linguaggio di script
    - Potrebbe richiedere una conoscenza più approfondita del dominio business
  - Nella pagina JSP è più difficile visualizzare le informazioni di presentazione.
  - Le operazioni di debug risultano più difficili a causa di una maggiore complessità e di una minore chiarezza.

Creazione di pagine JSP

- Una pagina JSP contiene:
  - Tag di markup standard, ad esempio HTML o XML
  - Dati di testo associati
  - Un'ampia gamma di elementi definiti dalla specifica JSP

Componenti della pagina JSP



Forme sintattiche dei tag JSP

- Le forme sintattiche dei tag basati su JSP possono essere rappresentate in due modi diversi:

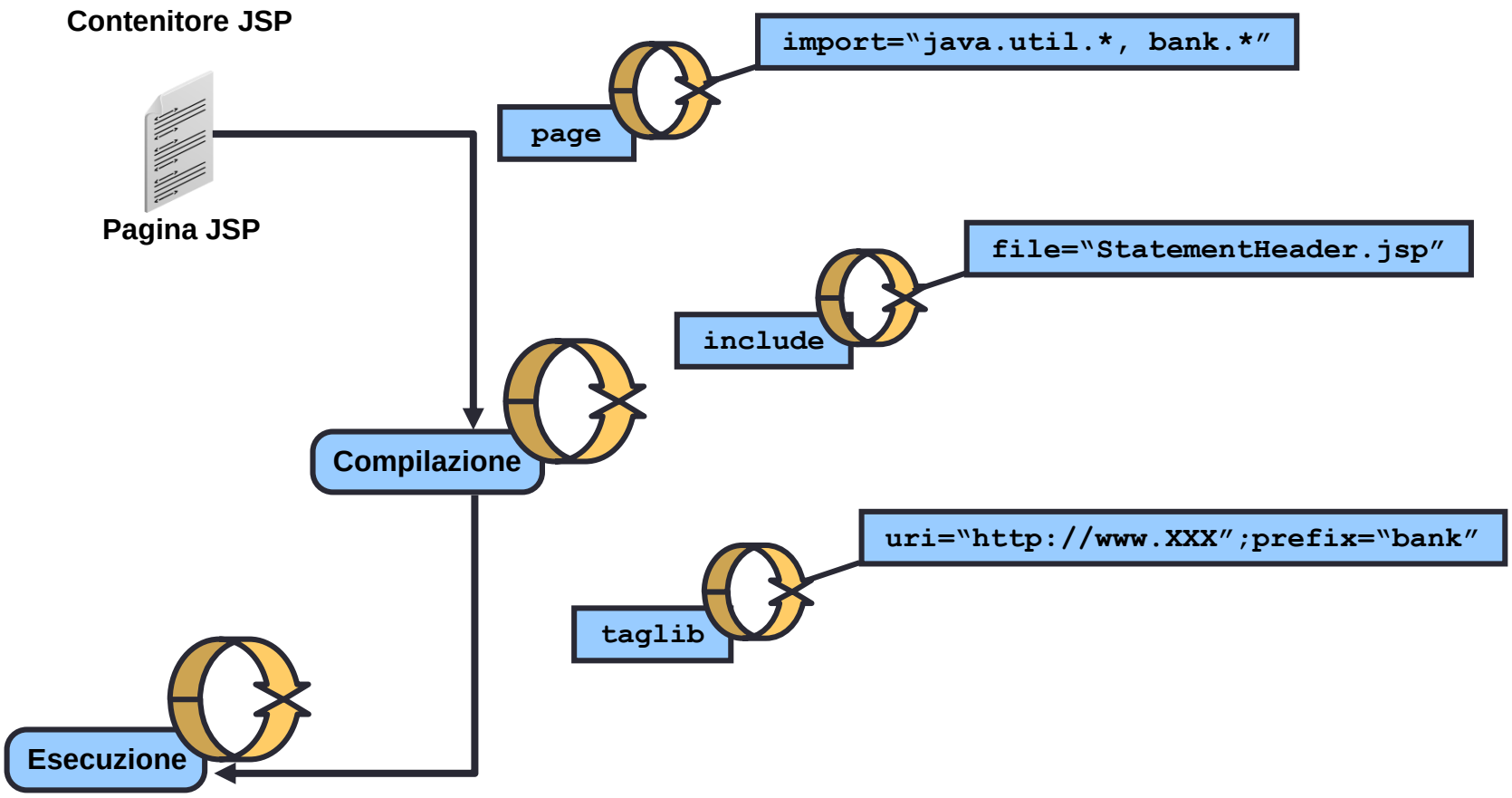
Sintassi precedente: analoga ad altre tecnologie di presentazione dinamica basate su tag	Sintassi XML: con tag iniziali e finali
`<%! ... %>`	`<jsp:declaration> ...` `</jsp:declaration>`
`<%= ... %>`	`<jsp:expression> ...` `</jsp:expression>`
`<% ... %>`	`<jsp:scriptlet> ...` `</jsp:scriptlet>`
`<%@ ... %>`	`<jsp:directive.type ... />`

Direttive della tecnologia JSP

- Includono informazioni che consentono a un contenitore JSP di configurare ed eseguire una pagina JSP
- Sono associate al servlet compilato che viene creato dalla pagina JSP
- Non producono output
- Presentano la sintassi generica seguente:

```
<%@ directive attribute="value" ... %>
```

Direttive JSP



La direttiva `page`

- La direttiva `page` definisce attributi dipendenti dalla pagina:
  - Una coppia attributo-valore non può essere ridefinita all'interno di un'unità di conversione. Fa eccezione la direttiva `include page`.
  - Se si prova a ridefinire una direttiva `page`, si da origine a un errore irreversibile, a meno che la nuova definizione non sia identica a quella precedente.

La direttiva `page`

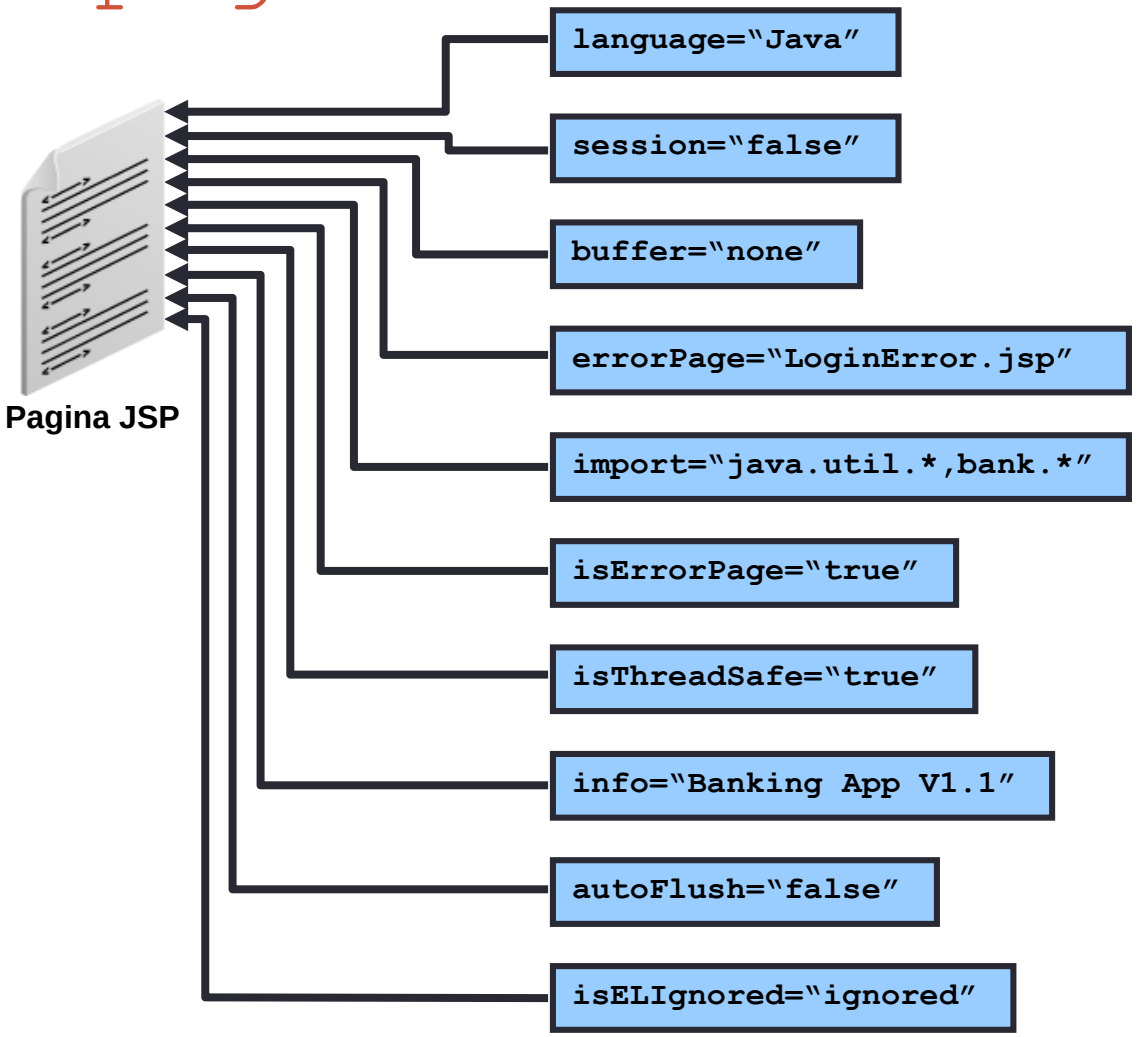
- Esempi di utilizzo di entrambi gli stili di sintassi:

```
<%@ page import="java.util.*, java.lang.*" %>

<%@ page buffer="5kb" autoFlush="false" %>

<jsp:directive.page errorPage="error.jsp" />
```

Direttive page JSP



La direttiva `include`

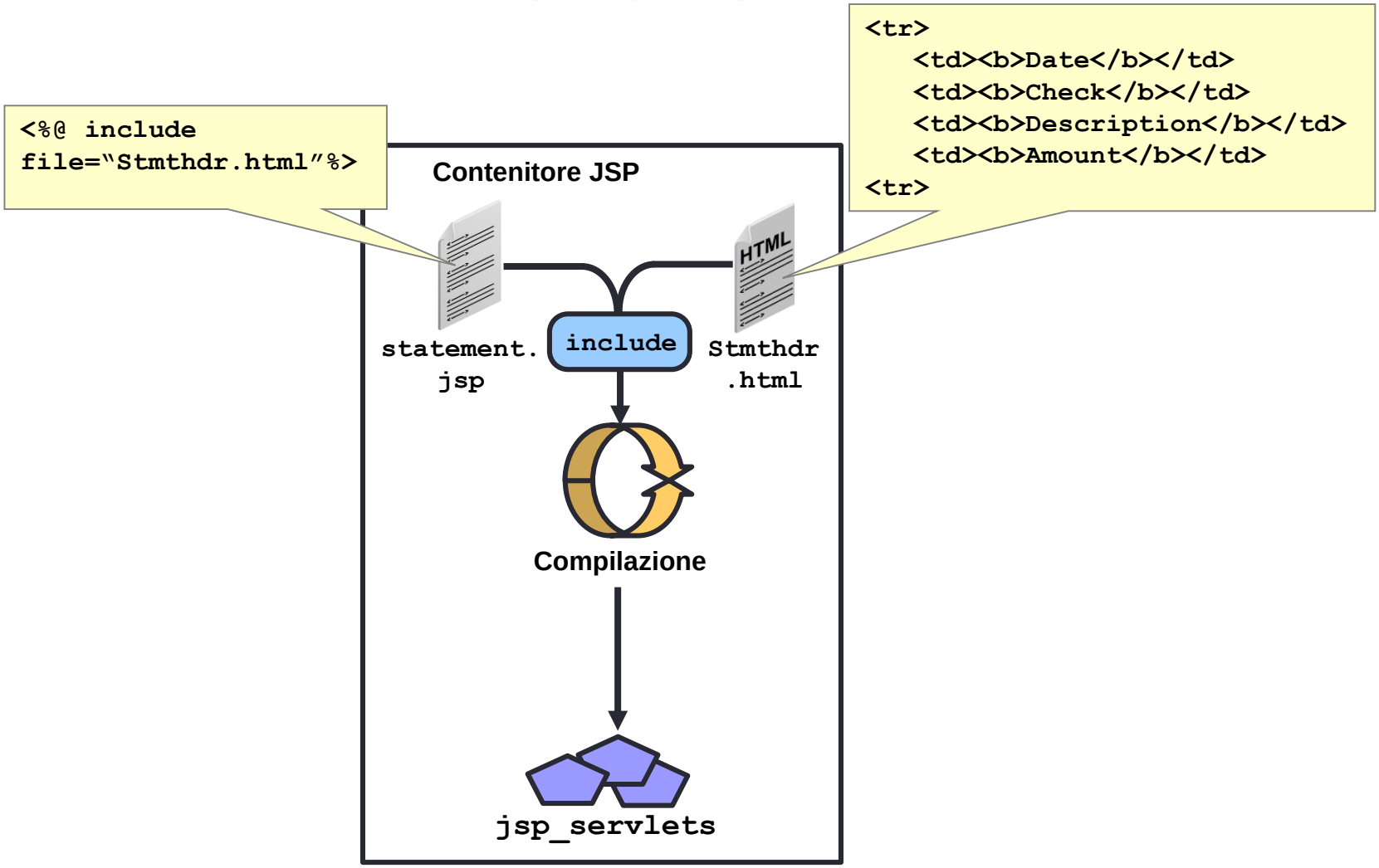
- La direttiva `include`:
  - Inserisce il testo della risorsa specificata nel file `.jsp` in fase di conversione della pagina
  - Tratta le risorse come oggetti statici
  - Può essere costituita da altri file HTML o da altre pagine JSP contenenti testo, codice o entrambi
- Esempi della direttiva `include`:

```
<%@ include file="relativeURL" %>
```

- Oppure

```
<jsp:directive.include file="relativeURL" />
```

La direttiva `include`



Dichiarazioni, espressioni e scriptlet

- Gli elementi di script consentono agli sviluppatori di pagine di offrire funzionalità di programmazione avanzate. Gli elementi di script includono:
  - Dichiarazioni
  - Espressioni
  - Scriptlet

Dichiarazioni, espressioni e scriptlet

	Dichiarazioni	Espressioni	Scriptlet
Utilizzo	Dichiarare metodi e variabili con ambito di istanza	Recuperare i valori delle variabili di pagina, dei metodi o dei campi dei bean	Incorporare snippet di codice di script
Requisiti della sintassi	Codice che deve: • Essere conforme alla sintassi del linguaggio di script • Formare un'istruzione dichiarativa	Qualsiasi espressione legale nel linguaggio di script	Blocchi raw di codice di programma che utilizzano Java come linguaggio di script predefinito
Descrizione	Non producono output nel flusso `out` corrente	Vengono valutati dal contenitore JSP in fase di esecuzione e i risultati vengono convertiti in un oggetto `String` nella pagina	• Vengono inseriti nel servlet generato senza modifiche • Utilizzare con moderazione

Elementi di script dichiarazioni

Sintassi	Sintassi alternativa
<%! declaration(s) %>	<jsp:declaration> *declaration(s)* </jsp:declaration>
Esempio	

<%! final String SHOWDETAILS_URL = "/showdetails.jsp";
boolean hasAccounts(Customer c) {return
!c.getAccounts().isEmpty();} %>
<jsp:declaration>
// This instance variable is assigned at initialization
time
protected BankMgr bankMgr = null;
</jsp:declaration>	

Elementi di script espressioni

Sintassi	Sintassi alternativa

`<%= expression %>`	`<jsp:expression>`
	`expression`
	`</jsp:expression>`

Esempio	

`<td> <%=acct.getBalance()%> </td>`
 	
`Oppure`
 	
`<td>`
`<jsp:expression>`
`acct.getBalance()`
`</jsp:expression>`
`</td>`	

Elementi di script scriptlet

Sintassi	Sintassi alternativa
<% *code_segment* %>	<jsp:scriptlet> *code_segment* </jsp:scriptlet>

Esempio

```
1  <%
2  if(isAllowedTransaction() == false){url=ScreenMgr.BANK_ERRORPAGE; }
3    else {
4      Vector checkList = account.getCheckByAmount(amt);
5      Iterator it = checkList.iterator();
6      double totalCheckAmount = 0.00;
7      while (it.hasNext()) {
8          Check chk = (Check) it.next();
9          totalCheckAmount += chk.amount();
10       } // end while
11     } // end if/else
12   %>
```

Elementi di script scriptlet

Esempio

```
1  <%
2     Check chk;
3     while (it.hasNext()) {
4        chk = (Check)it.next();
5        // end of first code fragment
6  %>
7        <%-- output check amount using HTML --%>
8        <br> Check Amount: <%=chk.getAmount()%> </br>
9  <%
10    } // closing bracket for while loop
11       // end of second code fragment
12 %>
```

Viene convertito in:

```
1          Check chk;
2          while (it.hasNext()) {
3                    chk = (Check)it.next();
4                    // end of first code fragment
5                    out.write("\t\t<br> Check Amount: ");
6                    out.print(chk.getAmount());
7                    out.write(" </br>\r\n");
8                    out.write("");
9          } // closing bracket for while loop
10         // end of second code fragment
```

Esempio JSP con scriptlet

```
<%! private static final String DEFAULT_NAME = "world";%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" href="res/styles.css" type="text/css"/>
    <title>helloJspScriptlet</title>
  </head>
  <body>
    <%
      String name = request.getParameter("name");
      if (name == null) {
        name = DEFAULT_NAME;
      }
    %>
    <h1>Hello <%=name%>!</h1>
  </body>
</html>
```

Implicazioni per la sicurezza thread

- Le dichiarazioni si verificano a livello di istanza del servlet generato. Pertanto:
  - Tutte le richieste alla pagina JSP condividono queste variabili e questi metodi
  - Con questa tecnica possono verificarsi problemi di sicurezza thread.
- Tutte le precauzioni che si applicano per i servlet e per la sicurezza thread vengono applicate anche per le dichiarazioni delle pagine JSP.

Elaborazione dei dati dai servlet

- La specifica JSP definisce un insieme di tipi di *azione* standard che devono essere implementati da tutti i contenitori JSP, tra cui:
  - Creare o utilizzare bean
  - Impostare e ottenere proprietà dei bean
  - Includere risorse statiche e dinamiche nel contesto della pagina corrente
- È possibile definire tipi di azione aggiuntivi utilizzando le librerie di tag personalizzate.

L'azione `jsp:useBean`

- Crea o individua un bean esistente che soddisfa i criteri del tag
- Associa l'istanza del bean a un ID di ambito e azione
- Rende l'ID accessibile per gli elementi di script e per i tag personalizzati

L'azione `jsp:useBean`

- Sintassi di `jsp:useBean`:

```
<jsp:useBean id="name" scope="scope" typeSpec />
```

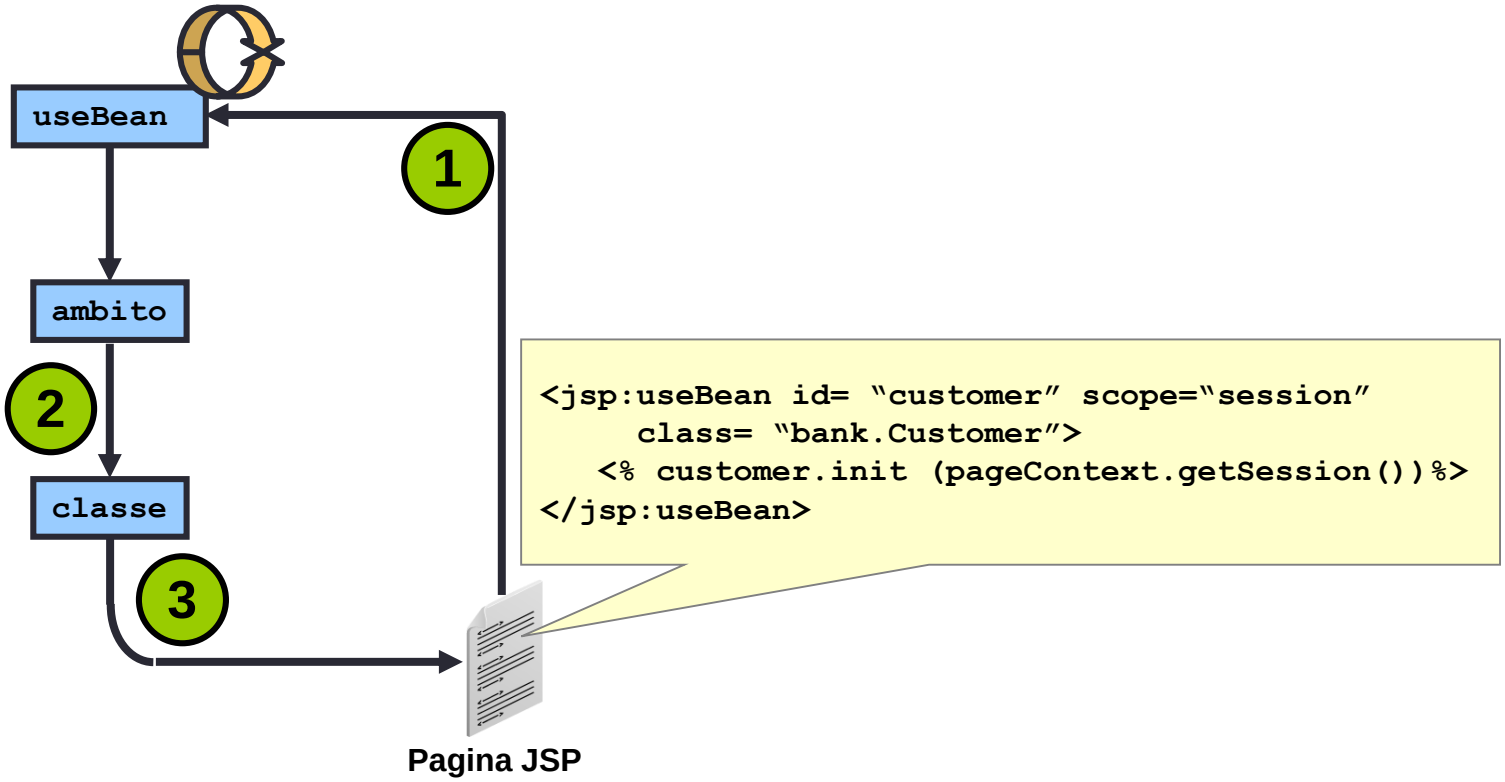
- Sintassi alternativa con codice di inizializzazione:

```
<jsp:useBean id="name" scope="scope" typeSpec >
      <% ...initialization code... %>
</jsp:useBean>
```

- `typeSpec` può essere uno dei seguenti:

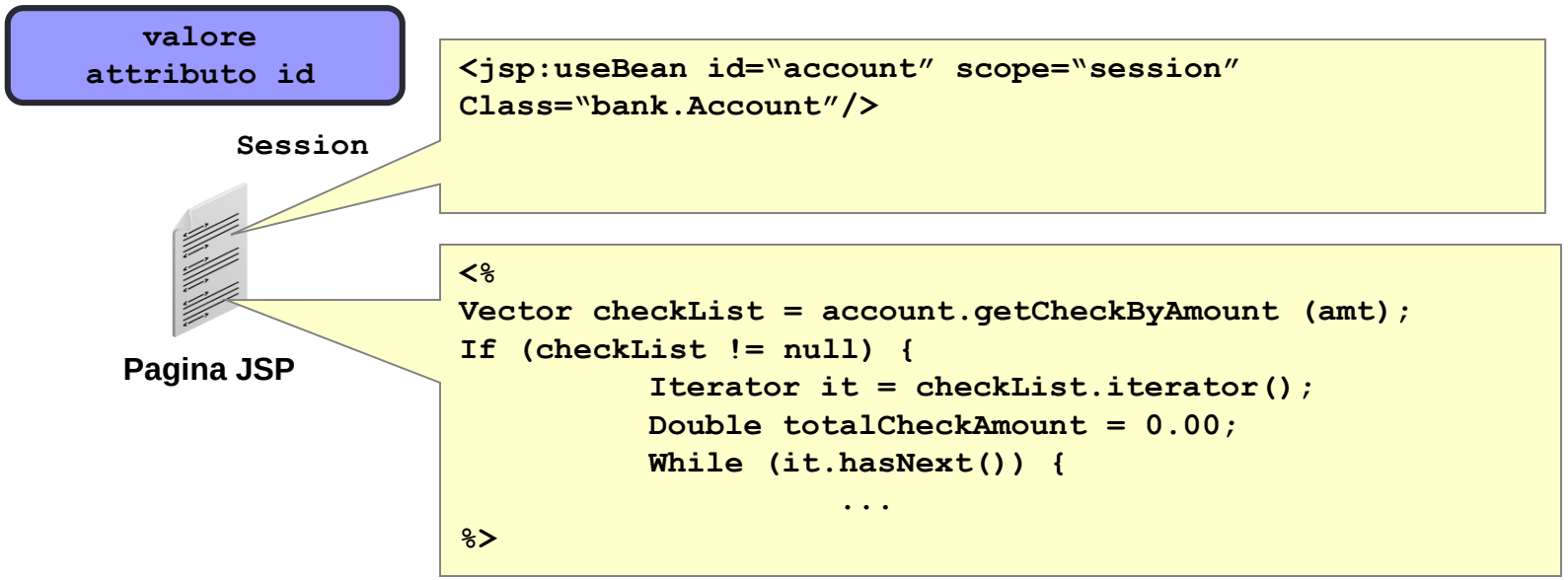
```
class="className"
class="className" type="typeName"
beanName="beanName" type=" typeName"
type="typeName"
```

L'azione `jsp:useBean`



L'azione `jsp:useBean`

- Questa figura mostra l'attributo `id`.



L'azione `jsp:useBean`

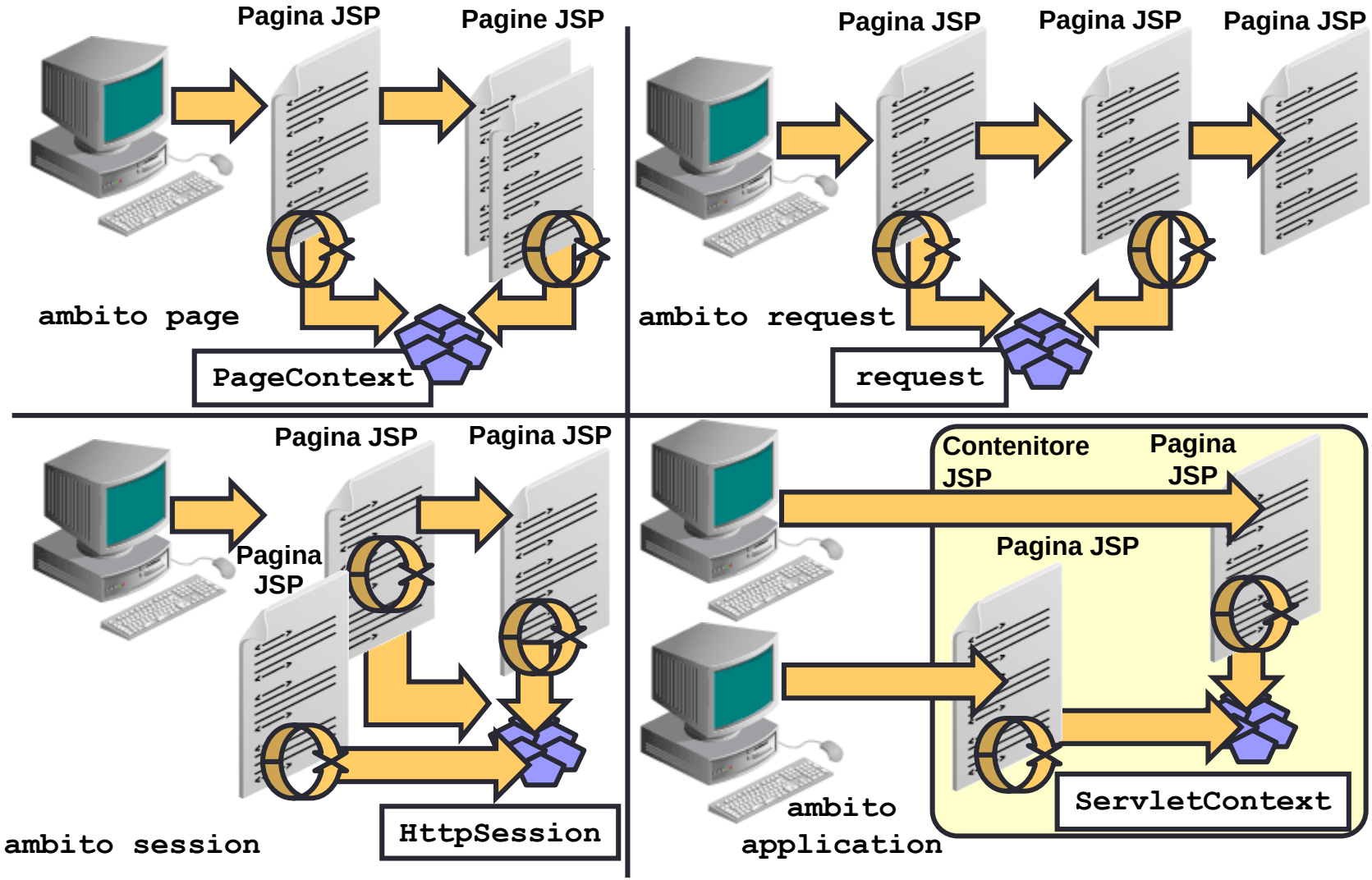
- Esempi di `jsp:useBean`:
  - Esempio di utilizzo per l'attributo `id`:

```
<jsp:useBean id="account" class="bank.Account"/>
```

  - Per recuperare il saldo conto, utilizzare l'espressione:

```
<%=account.getBalance() %>
```

Gli ambiti `jsp:useBean`



L'interfaccia `RequestDispatcher`

- Esistono due metodi `getRequestDispatcher`("URI") che restituiscono un'implementazione `RequestDispatcher`.
- Il metodo `ServletRequest`, che accetta percorsi relativi o percorsi che iniziano con il simbolo a "/"

```
RequestDispatcher requestDispatcher =
request.getRequestDispatcher("relativeURI");
```

- Il metodo `ServletContext`, che deve iniziare con il simbolo "/"

```
RequestDispatcher requestDispatcher
=   getServletContext().getRequestDispatcher
("/ServletName");
```

La destinazione `RequestDispatcher` e la radice di contesto

- L'argomento di `getRequestDispatcher` è un URI, tuttavia viene interpretato dal contenitore Web con riferimento al contesto dell'applicazione corrente. L'URI:
  - Deve iniziare con una barra (/) o essere relativo rispetto alla pagina corrente
  - *Non deve contenere una radice di contesto o essere un URI completo*
- Nell'applicazione di esempio bank il servlet ottiene il componente JSP al quale trasferirà il controllo utilizzando l'istruzione seguente:

```
getRequestDispatcher
("/showCustomerDetails.jsp");
```

JavaBean

- Classi POJO (Plain Old Java Object)
- Costruttore senza argomenti (default constructor)
- Attributi non visibili
- Accesso a proprietà mediante getter e setter (accessor e mutator)
- Es. `String getFirstName()` definisce una proprietà di tipo stringa chiamata `firstName` e che può essere letta.
- Es. `setFirstName(String s)` defininisce una proprietà di tipo stringa chiamata `firstName` e che può essere modificata.

Bean con ambito request e raccolta dei dati dai servlet

- L'azione `jsp:useBean` viene comunemente utilizzata per condividere i dati tra i servlet e le pagine JSP.
Di seguito viene illustrata una sequenza tipica degli eventi che interessano `jsp:useBean`:
  - Un servlet esegue l'elaborazione front-end.
  - Il servlet imposta un attributo sull'oggetto request.
  - Il servlet invia il controllo a una pagina JSP per visualizzare i dati dinamici.
  - JSP utilizza `jsp:useBean` con l'attributo con ambito request per la raccolta dei dati.

Esempio JSP:useBean

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" href="res/styles.css" type="text/css"/>
    <title>Hello JSP tags</title>
  </head>
  <body>
    <jsp:useBean scope="request" id="nameBean"
class="it.prova.bean.NameBean/>
    <jsp:setProperty name="nameBean" param="name" property="name"/>
    <h1>
      Hello
      <jsp:getProperty name="nameBean" property="name"/>
    </h1>
  </body>
</html>
```

Esempio JSP:useBean : classe Java

```
package it.prova.bean;

public class NameBean {

  private String name;

  public String getName() {
    return name == null ? "world" : name;
  }

  public void setName(String name) {
    this.name = name;
  }
}
```

Bean con ambito request e raccolta dei dati dai servlet

- Lo snippet di codice che segue crea un nuovo oggetto `Customer` e lo salva in un attributo `request` denominato `customer`:

```
1 public void doPost(HttpServletRequest request, HttpServletResponse response) {
2                     ...
3         try {
4                     Customer cust = new Customer(firstName, lastName);
5                     request.setAttribute("customer", cust);
6
7                     // use a request dispatcher to forward to a JSP page
8                     RequestDispatcher disp =
9                     getServletContext().getRequestDispatcher("/jsp/example.jsp");
10                    disp.forward(request, response);
11        } catch (Exception ex) {
12                  . . .
13        } // end catch
14 }
```

Bean con ambito request e raccolta dei dati dai servlet

- La pagina JSP `example.jsp` illustrata nello snippet di codice che segue può quindi accedere all'oggetto `customer` ed elaborarlo nel modo seguente:

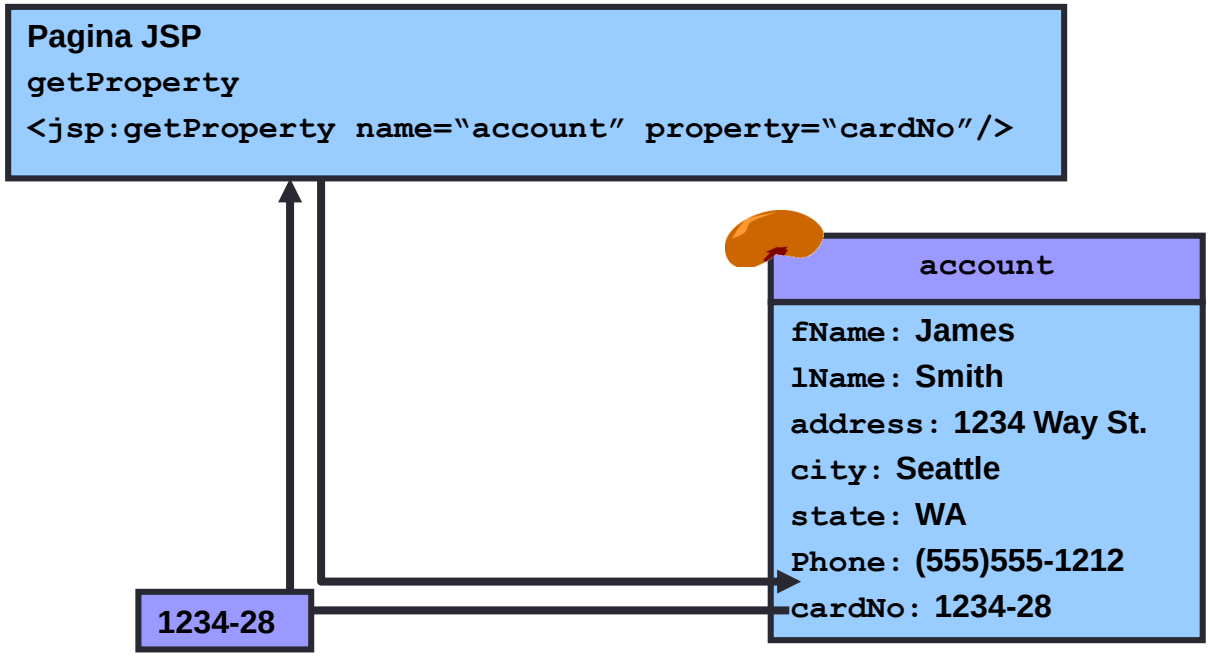
```
1 <jsp:useBean id="customer" class="bank.Customer" scope="request"/>
2                ...
3 <jsp:getProperty name="customer" property="firstName" />
4 <jsp:getProperty name="customer" property="lastName" />
5                ...
```

Bean con ambito request e raccolta dei dati dai servlet

- Come è definita la classe Customer?

. . .

L'azione `jsp:getProperty`



Linguaggio di espressione (EL)

- Il linguaggio di espressione è un linguaggio di facile utilizzo che può essere incorporato nelle pagine JSP in sostituzione degli scriptlet (quando viene utilizzato con le librerie di tag). La sintassi è simile a JavaScript ed è facile da utilizzare anche per utenti non programmatori.

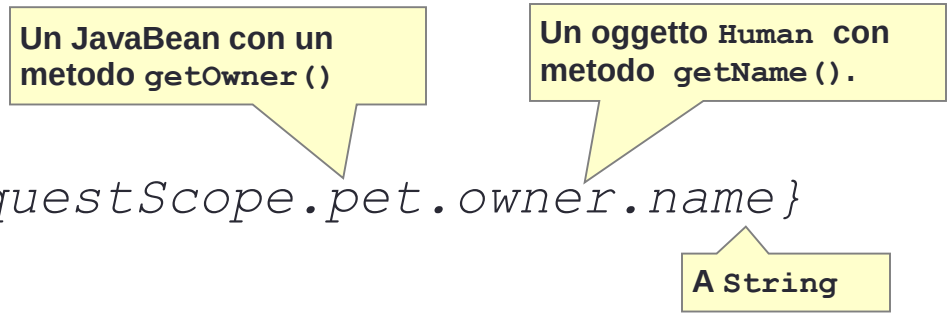
```
${ 3 + 2 }
${ param.address }
${ requestScope.customer.name }
${ not empty sessionScope.message }
```

EL Implicit Objects

Implicit Object	Description
`pageContext`	The `PageContext` object
`pageScope`	A map containing page-scoped attributes and their values
`requestScope`	A map containing request-scoped attributes and their values
`sessionScope`	A map containing session-scoped attributes and their values
`applicationScope`	A map containing application-scoped attributes and their values
`param`	A map containing request parameters and single string values
`paramValues`	A map containing request parameters and their corresponding string arrays
`header`	A map containing header names and single string values
`headerValues`	A map containing header names and their corresponding string arrays
`cookie`	A map containing cookie names and their values
`initParam`	A map of the servlet's init parameters

Dot Operator in EL

- L'operatore . (dot operator) permette di accedere ad una proprietà di un bean o all'elemento di una mappa.
- Example:



- Le proprietà o le chiavi non devono avere nel nome il carattere "."
  - In questo caso si usa la notazione vettoriale

```
${requestScope.pet.owner["name"]}
```

Libreria di tag di base JSTL

- Java EE 6 offre diversi tag personalizzati già scritti noti come JavaServer Pages Standard Tag Library o JSTL.
- Tali librerie sono raggruppate per funzionalità.
- La libreria più utilizzata è la libreria di base.

Area funzionale	URI	Prefisso
Di base	`http://java.sun.com/jsp/jstl/core`	`c`
Elaborazione XML	`http://java.sun.com/jsp/jstl/xml`	`x`
Formattazione i18N	`http://java.sun.com/jsp/jstl/fmt`	`fmt`
Accesso al database relazionale (SQL)	`http://java.sun.com/jsp/jstl/sql`	`sql`
Funzioni	`http://java.sun.com/jsp/jstl/functions`	`fn`

Core Tag Library

Tag	Purpose
`c:out`	Evaluates an expression and outputs the result to the current `JspWriter`
`c:set`	Sets the value of a scoped variable or property
`c:remove`	Removes a scoped variable
`c:catch`	Catches a `java.lang.Throwable` that occurs in the body of the tag
`c:if`	Evaluates the body of the tag if the expression specified by the test attribute is `true`
`c:choose`	Provides a mutually exclusive conditional
`c:when`	Provides an alternative within a `c:choose` element
`c:otherwise`	Provides the last alternative within a `c:choose` element
`c:forEach`	Iterates over a collection of objects or for a fixed number of cycles
`c:forTokens`	Splits a string into tokens and iterates over those tokens
`c:import`	Imports the content of a URL resource
`c:url`	Rewrites relative URLs
`c:redirect`	Sends an HTTP redirect to the client
`c:param`	Adds parameters to the request (used within `c:import`, `c:url`, and `c:redirect`)

`url` Tag

- Riscrive la URL:

```
<c:url value="value"
   [var="varName"]
   [scope="{page|request|session|application}"] />
```

- Optional: Usa `var` e `scope` per memorizzare la URL.
- Esempio:

```
<c:url value="/register/enter_player.do" var="enterPlayerUrl"/>
<form action="${enterPlayerUrl}"  method='POST'> …
</form>
```

Esempio JSP senza scriptlet

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" href="res/styles.css" type="text/css"/>
    <title>JSP Page</title>
  </head>
  <body>
    <c:set var="name" value="world" />
    <c:if test="${not empty param.name}">
      <c:set var="name" value="${param.name}" />
    </c:if>
    <h1>Hello ${name}!</h1>
    <a href="index.html">Go home</a>
  </body>
</html>
```

Esempio JSP:include

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<jsp:include page="/templates/head.jsp">
<jsp:param name="title" value="Situazione" />
</jsp:include>
</head>
<body>
<jsp:include page="/templates/header.jsp">
<jsp:param name="title" value="Pagina principale" />
</jsp:include>
Qualche dato utile
    <<jsp:include page="/templates/footer.jsp" />
</body>
```

Esempio JSP:include

- **/templates/head.jsp**

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<c:url value="/res/styles.css" var="URLstyles"/>

<title>${param.title}</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link rel="stylesheet" href="${URLstyles}" type="text/css">
```

- **/templates/header.jsp**

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
Applicazione web Kmakers
<h1>${param.title}</h1>
```

Esempio JSP:include

- **/templates/footer.jsp**

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
  <p>Copyright 2014: KMakers</p>
  <p>Contatti: <a href="mailto:info@kmakers.com">info@kmakers.com</a>.</p>
```

Esempi JSTL

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<c:forEach var="item" items="${requestScope.list}">
        <tr><td>${item.var1}</td><td>${item.var2}</td></tr>
</c:forEach>

<c:if test="${x < 3}" >
</c:if>

<c:choose>
        <c:when test="${requestScope.message == null}">
        </c:when>
        <c:otherwise>
        </c:otherwise>
</c:choose>
```

- **Per saperne di più : http://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/**

JSP

- *JSP - Expression Language (EL)*
- Sintassi molto semplice
- Viene incorporato all'interno delle JSP in sostituzione degli scriptlet
- Utilizzabile anche da utenti non programmatori
- EL invocato attraverso il costrutto ${expr}
- Java EE offre diversi tag personalizzati già scritti noti come JavaServer Pages Standard Tag Library o JSTL.
- Tali librerie sono raggruppate per funzionalità.

JSP

Area funzionale	URI	Prefisso
Di base	http://java.sun.com/jsp/jstl/core	c
Elaborazione XML	http://java.sun.com/jsp/jstl/xml	x
Formattazione 118N	http://java.sun.com/jsp/jstl/fmt	fmt
Accesso al database relazionale (SQL)	http://java.sun.com/jsp/jstl/sql	sql
Funzioni	http://java.sun.com/jsp/jstl/functions	fn

- La libreria core è quella più utilizzata : consente di raggiungere i medesimi risultati degli scriptlet
- Vediamo alcuni esempi

JSP

```
<h3>Esempio di EL</h3>
Il numero 20 è maggiore di 30 ? :
${20>30} <br>
La somma di 20 e 30 è : ${20+30}
```

Esempio di EL

Il numero 20 è maggiore di 30 ? :false
La somma di 20 e 30 è : 50

JSP

- *Controllo del flusso c:if*
- Nella JSP

```
<%@taglib
uri="http://java.sun.com/jsp/jstl/core"
prefix="c"%>

..........
<h3>Esempio c:if in  EL</h3>
  <c:set var="num" value="100"/>
  <c:if test="${num > 0}">
       Num = <c:out value="${num}"/>
  </c:if>
```

Esempio c:if in EL

Num = 100

JSP

- *c:choose, c:when e c:otherwise*

```
<%@ taglib
uri="http://java.sun.com/jsp/jstl/core"
prefix="c" %>
<h3>Test di choose, when e otherwise</h3>
<c:set var="val" value="10"/>
<c:choose>
    <c:when test="${val <= 10}">
        Numero minore o uguale a 10.
    </c:when>
    <c:otherwise>
        Numero maggiore di 10.
    </c:otherwise>
```

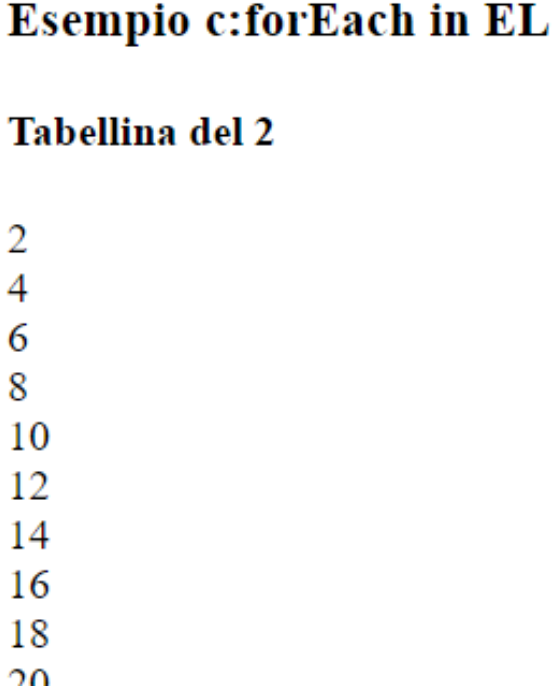
Test di choose, when e otherwise

Numero maggiore di 10.

JSP

- *c:forEach (esempio 1)*

```
<h3>Esempio c:forEach in  EL</h3>
<h4>Tabellina del 2</h4>
      <c:forEach var="i" begin="1" end="10">
            <c:out value="${i*2}"/><br>
      </c:forEach>
```



JSP

- *c:forEach (esempio 2)*

```
<h4>Esempio c:forEach in  EL</h4>
<h4>Array di stringhe</h4>
<c:set var="names" scope='request'>Pippo, Pluto, Paperino,    Qui, Quo, Qua</c:set>
<c:forEach var="nome" items="${names}">
     Nome: <c:out value="${nome}" /><br />
</c:forEach>
```

Esempio c:forEach in EL

Array di stringhe

Nome: Pippo
Nome: Pluto
Nome: Paperino
Nome: Qui
Nome: Quo
Nome: Qua

JSP

- *c:forTokens*

```
<h3>Test di c:forTokens</h3>
<c:forTokens items="Pippo, Pluto, Paperino"
delims=", " var="name">
<c:out value="${name}"/><p>
</c:forTokens>
```

Test di c:forTokens

Pippo

Pluto

Paperino

JSP

- *c:url con c:param*
- Il tag <c:url> forma un URL in una stringa e lo memorizza in una variabile. L'attributo var specifica la variabile che contiene l'URL formattato.
- c:param consente di veicolare un parametro con coppia nome-valore
- Esempio :

Su *Prima.jsp*

```
<c:url value="/login.jsp" var="helloUrl">
<c:param name="userName" value="Hello TIM !!!"></c:param>
</c:url> <h4><a href="${helloUrl}">Click here</a></h4>
```

JSP

Su *login.jsp*

```
<c:out value="${param.userName}"></c:out>
```

---OUTPUT :

JSP

- Direttiva taglib e librerie tag personalizzate
  - Estende l'insieme di tag che un contenitore JSP può interpretare
  - Associa un prefisso di tag a una libreria di tag
  - Formato della direttiva

```
<%@ taglib uri="iterator_tags" prefix="iterator" %>
```

  - Utilizzo :

```
<iterator:iterate>
        <%-- perform repetitive task --%>
        ...
</iterator:iterate>
```

  - Utilizzo di una libreria di tag che implementa un iteratore generico. Gli sviluppatori di pagine JSP utilizzano
    il tag iterator per ripetere un'azione su tutti gli elementi che implementano l'interfaccia

JSP

- Le librerie di tag standard sono già presenti in qualsiasi sistema Java EE 5 come file JAR. Nei file JAR JSTL sono incorporati i file TLD in modo tale da renderne superfluo l'inserimento in un elenco nel file `web.xml`.
- Associazione URI - JAR

Grazie