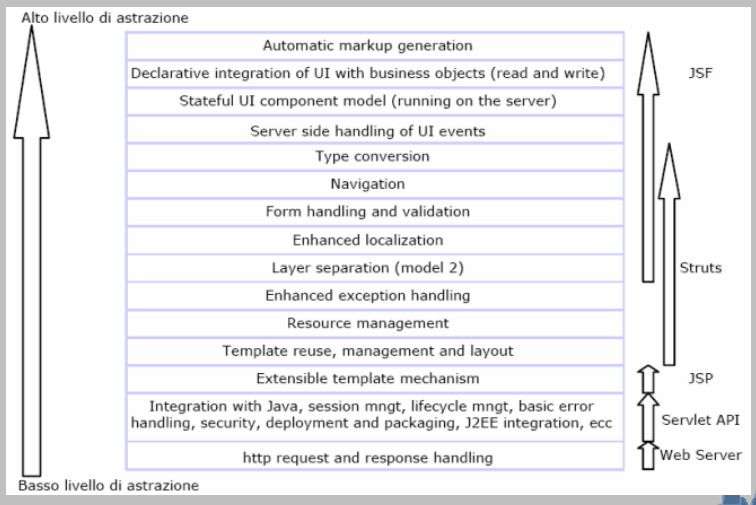
Web Application Engineering
Java Server Faces

cristian lucchesi
IIT-CNR

Pescara, 15-16 Maggio 2007
Alei – Ud'A

Web application: evoluzione



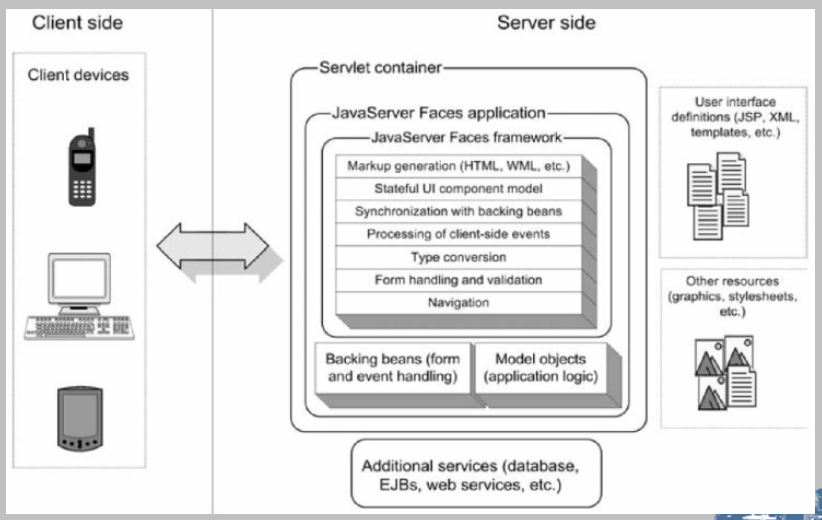
Java Server Faces (JSF)

- la tecnologia JSF è una framework a componenti lo sviluppo di interfacce utente
- i componenti principali della tecnologia JSF sono:
  - una API per rappresentare i componenti della UI
  - un sistema di gestione gli eventi, validazione server-side, e conversione dei dati
  - un sistema di definizione della navigazione
  - supporto all'internazionalizzazione
  - metodi per legare i componenti della UI a oggetti server-side
- la prima versione delle specifiche è del 2003, la versione attuale è del 2006 (JSF 1.2 JSR-252)

JSF: tecnologie pre-esistenti

- **Servlet**
  - sono il fondamento delle web application nell'architettura J2EE.
  - l'approccio a basso livello per la produzione del markup (lo sviluppo di una pagina web richiede, sostanzialmente, allo sviluppatore di codificare un programma che produca come output i tag della pagina html
  - Le servlet API forniscono un insieme di funzionalità di base (session tracking, security, logging, filtering, lifecycle events, ecc) di cui si giova il framework JSF
- **JSP**
  - rappresentano il meccanismo di template standard definito nell'architettura J2EE.
  - le pagine JSP sono orientate ai tag: utilizzano, cioè, oltre ai consueti marcatori HTML i cosiddetti custom tag per introdurre comportamento dinamico nella pagina. Di più, consentono di inserire nella pagina codice Java embedded alternato ai tag descritti.

JSF Overview



Esempio con JSF

- costruiremo una applicazione di tipo "Hello, world"
- predisporremo due pagine:
  - inputname.jsp: chiede all'utente di inserire il proprio nome
  - greeting.jsp: mostra un saluto all'utente

Struttura tipica di un progetto JSF

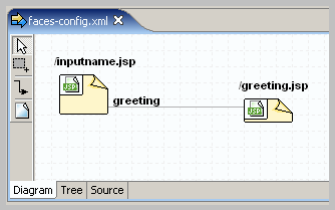
```
rootFolder
   /ant
      build.xml              ->  ant build script

   /src                      -> contiene le classi Java ed i file properties

   /WebContent
      /WEB-INF
         /classes            -> ospiterà classi Java compilate e file properties
            /lib
               jsf-impl.jar
               jsf-api.jar
         faces-config.xml    -> lista dei bean gestiti e regole di navigazione
         web.xml             -> descrittore servlet e di altri componenti

      /pages                 -> folder che ospiterà le pagine jsf
```


Navigation



- la navigazione è il cuore delle applicazioni JSF
- la navigazione è definita nel file faces-config.xml

```
<navigation-rule>
  <from-view-id>
     /pages/inputname.jsp
  </from-view-id>
  <navigation-case>
     <from-outcome>greeting</from-outcome>
     <to-view-id>/pages/greeting.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

- la regola dice che se l' "outcome" eseguito da inputname.jsp è greeting si passa dalla view (pagina) inputname.jsp alla view (pagina) greeting.jsp

Creare i Managed Bean

```
package examples.jsf;

public class PersonBean {

    String personName;

    public String getPersonName() {
        return personName;
    }

    public void setPersonName(String name) {
        personName = name;
    }
    ...
}
```

- la classe è un semplice Java Bean con un attributo e i metodi setter/getter
- il bean cattura il nome inserito dall'utente dopo l'invio della form
- il bean fornisce un ponte tra la pagina JSP e la logica dell'applicazione
- il nome della proprietà deve corrispondere al nome del campo nella JSP

Dichiarare i bean in faces-config.xml

```
<managed-bean>

    <managed-bean-name>
        personBean
    </managed-bean-name>

    <managed-bean-class>
        examples.jsf.PersonBean
    </managed-bean-class>

    <managed-bean-scope>
        request
    </managed-bean-scope>

</managed-bean>
```

- in faces-config.xml si descrivono i Java bean che utilizzeremo nell'applicazione JSF
- i bean sono identificati con un nome tramite elemento **managed-bean-name**
- deve essere specificato il nome qualificato della classe tramite elemento **managed-bean-class**
- deve essere specificato lo scopo del bean (request, session) tramite elemento **managed-bean-scope**

faces-config.xml

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config
1.1//EN" "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
   <navigation-rule>
      <from-view-id>/pages/inputname.jsp</from-view-id>
      <navigation-case>
         <from-outcome>greeting</from-outcome>
         <to-view-id>/pages/greeting.jsp</to-view-id>
      </navigation-case>
   </navigation-rule>
   <managed-bean>
      <managed-bean-name>personBean</managed-bean-name>
      <managed-bean-class>jsfks.PersonBean</managed-bean-class>
      <managed-bean-scope>request</managed-bean-scope>
   </managed-bean>
</faces-config>
```


file properties

messages.properties

```
inputname_header=JSF KickStart
prompt=Tell us your name: greeting_
text=Welcome to JSF button_
text=Say Hello sign=!
```

- un file properties è un file composto da coppie ***parametro=valore***
- contiene i messaggi (le stringhe di testo utilizzate) delle nostre pagine JSP
- mantenere i messaggi separati dalla pagina JSP permette una rapida modifica dei messaggi senza modificare la pagina JSP
- tipicamente si chiamano ***messages.properties***, sono messi nella directory dei sorgenti e ricopiati dal build script nella directory **WEB-INF/classes**

inputname.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<f:loadBundle basename="examples.jsf.messages" var="msg"/>
   <html>
      <head> <title>enter your name page</title> </head>
      <body>
         <f:view>
            <h1> <h:outputText value="#{msg.inputname_header}"/> </h1>
            <h:form id="helloForm">
               <h:outputText value="#{msg.prompt}"/>
               <h:inputText value="#{personBean.personName}" />
               <h:commandButton action="greeting" value="#{msg.button_text}" />
            </h:form>
         </f:view>
      </body>
   </html>
```


inputname.jsp: spiegazioni

```
<%@ taglib uri="http://java.sun.com/jsf/html"
           prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core"
           prefix="f" %>
<f:loadBundle
     basename="jsfks.bundle.messages"
     var="msg"/>
```

- la prima riga indica dove trovare la definizione dei tag JSF che definiscono gli elementi HTML
- la seconda riga indica dove trovare i tag JSF che definiscono gli elementi base (il core) delle JSF
- la terza riga carica il file messages.properties nella variabile *msg*

```
<h:outputText
     value="#{msg.inputname_header}"/>
```

- questo tag indica di guardare nell'oggetto msg che abbiamo definito sopra, cercare il valore per *inputname_header* e di stamparlo nell'html

inputname.jsp: spiegazioni

cont.

```
1 <h:form id="helloForm">

2   <h:outputText
        value="#{msg.prompt}"/>

3   <h:inputText id="name"
        value="#{personBean.personName}" />

4   <h:commandButton
        action="greeting“
        value="#{msg.button_text}" />

5 </h:form>
```

- Riga 1
  - crea una form HTML utilizzando i tag JSF
- Riga 2
  - stampa un messaggio prelevando dal file properties il valore di *prompt*
- Riga 3.
  - crea un campo input HTML. L'attributo *value* viene legato (bind) all'attributo *personName* del Managed Bean che si chiama *personBean*
- Riga 4
  - tag JSF per il botton HTML per la submit
  - il valore del bottone è prelevato dal file properties
  - l'attributo *action* è impostato a *greeting* che corrisponde al *navigation-outcome* nel file faces-config.xml
  - il valore dell'action è utilizzato dalla jsf per sapere la prossima pagina da mostrare

greeting.jsp

```
<%@ taglib uri=http://java.sun.com/jsf/html
            prefix="h" %>
<%@ taglib uri=http://java.sun.com/jsf/core
            prefix="f" %>
<f:loadBundle basename="jsfks.bundle.messages“
              var="msg"/>
<html>
 <head> <title>greeting page</title> </head>
 <body>
  <f:view>
   <h3>
    <h:outputText
        value="#{msg.greeting_text}" />,
    <h:outputText
        value="#{personBean.personName}" />
    <h:outputText
        value="#{msg.sign}" />
   </h3>
  </f:view>
 </body>
</html>
```

- le prime tre righe sono identiche a quelle di inputname.jsp
- l'elemento

```
<h:outputText
  value="#{personBean.personName}" />
```

accede all'attributo *personName* del bean *personBean*, attributo precedentemente impostato, e lo stampa

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
   xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
            http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsp</url-pattern>
  </servlet-mapping>
</web-app>
```

- essendo l'applicazione JSF una web application J2EE, è necessario configurare l'applicazione mediante web.xml

Utilizzare i validatori standard

```
...
<f:view>
  <h1>
    <h:outputText value="#{msg.inputname_header}"/>
  </h1>
  <h:form id="helloForm">
    <h:outputText value="#{msg.prompt}"/>
    <h:inputText value="#{personBean.personName}"
                 required="true" />
    <h:commandButton action="greeting"
                     value="#{msg.button_text}" />
  </h:form>
</f:view>
...
```

- per rendere il nome della persona obbligatorio è sufficiente specificare l'attributo **required="true"**

Verificare la lunghezza di un campo

```
...
<h:form id="helloForm">
  <h:outputText value="#{msg.prompt}"/>
  <h:inputText value="#{personBean.personName}"
               required="true">
     <f:validateLength minimum="2"
                       maximum="10" />
   </h:inputText>
   <h:commandButton action="greeting"
                    value="#{msg.button_text}" />
</h:form>
...
```

- **Validatori standard**
  - DoubleRangeValidator verifica l'intervallo di un double (in virgola mobile)
  - LengthValidator verifica la lunghezza di una stringa
  - LongRangeValidator verifica l'intervallo di un long

Custom validator

è possibile definire propri validatori:

- implementando l'interfaccia

  javax.faces.validator.Validator, in particolare
  public void validate(FacesContext cxt, UIComponent comp, Object value);

- definendo il validatore in faces-config.xml

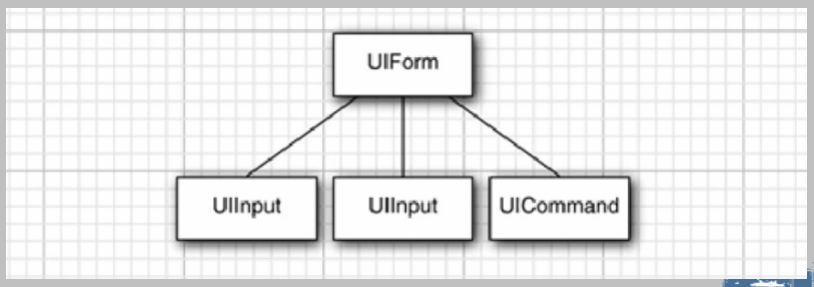
```
<validator>
   <validator-id>MyValidator<validator-id>
   <validator-class>validator.MyValidator<validator-class>
<validator>
```

- utilizzando nella jsf l'elemento

```
<f:validate validatorId="myValidator" />
```

JSF: component tree

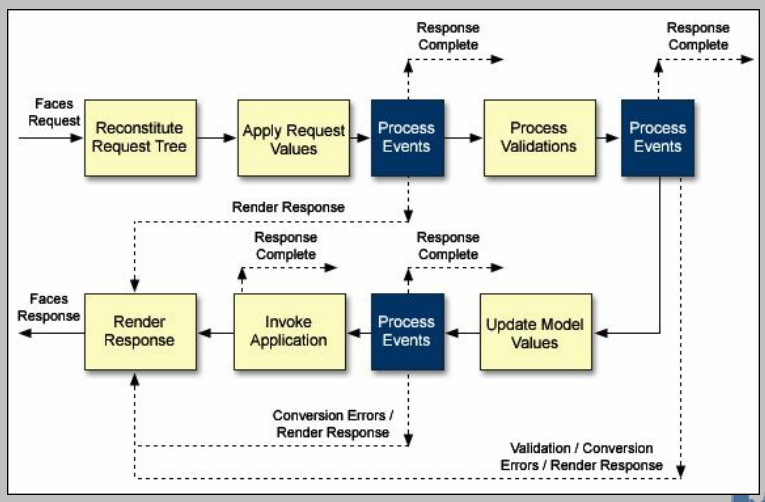
- quando la pagina jsp (nel nostro caso inputname.jsp) viene invocata, i tag handler associati ai tag presenti nella pagina vengono eseguiti
- i vari tag handler collaborano per realizzare il cosiddetto component tree
- il component tree è una struttura dati che contiene oggetti Java per tutti gli elementi UI di una pagina JSF



JSF: rendering

- successivamente la pagina HTML viene renderizzata
- tutto ciò che non è un tag JSF viene scritto sul output
- i tag riconosciuti (perché associati ad un definito namespace) vengono invece processati producendo omologhi tag HTML
- Il renderer associato all'oggetto UIInput invoca il framework JSF per ottenere il valore corrente dell'espressione **personBean.personName** (con cui aggiorna il componente inputText per mantenere la sincronizzazione con il backing bean **personBean**)
- la pagina html generata viene restituita al browser

JSF life-cycle



JSF life-cycle: le fasi iniziali

fase di Restore View

- viene ricercato il component tree della pagina richiesta, se si tratta della prima richiesta (initial request) il component tree viene creato.
- se la request non presenta dati in POST, l'implementazione JSF porta alla fase *Render Response* (accade per esempio quando la pagina viene richiesta la prima volta)

fase di Apply Request Values

- JSF itera sui componenti affinché ciascuno di essi memorizzi i dati di pertinenza (submitted values).

JSF life-cycle: Process Validation

fase di Process Validation

- se sono definiti validatori su uno o più componenti viene attivato il processo di validazione
- se ci sono errori di conversione o validazione, viene invocata direttamente la fase Render Response che porta al ri-display della pagina per garantire all'utente di correggere i dati
- se non ci sono errori si aggiorna il modello dei dati. Durante la fase Update Model i "local values" sono utilizzati per aggiornare le proprietà dei bean associati ai componenti

JSF life-cycle: fasi finali

fase di Invoke Application

- il metodo indicato nell'attributo action viene eseguito
- questo metodo tipicamente implementa la logica di business associata all'azione dell'utente
- il metodo ritorna un outcome string che viene passata al navigation handler il quale provvede al look up della pagina successiva

fase di Render Response

- effettua l'encoding della risposta e la spedisce al browser
- quando l'utente genera una nuova richiesta il ciclo ricomincia

Riferimenti

- JavaServer Faces (JSF) Tutorial:
  http://www.exadel.com/tutorial/jsf/jsftutorial-kickstart.html
- The Java EE 5 Tutorial:
  http://java.sun.com/javaee/5/docs/tutorial/doc/index.html

grazie per l'attenzione

cristian.lucchesi@iit.cnr.it