Java Server Faces

Giuseppe Sottile

Capitolo 1

Introduzione

Java Server Faces (JFS) é un framework per la realizzazione di faces ovvero facciate - UI per le applicazioni web in java lato server. La caratteristica fondamentale di questo tool messo a disposizione da SUN nella specifica J2EE é la piena compatibilitá con il pattern MVC (Model View Controller), in cui vi é una netta separazione tra interfaccia ed implementazione. Ricordiamo che in Ingegneria del Software il pattern MVC promuove il riuso del software e la buona robustezza in quanto essendo distinti i relativi moduli es: (dati, controllo ed interfaccia) ciascun membro di un eventuale team di sviluppo puó concentrarsi su specifiche aree, rendendo semplice il testing e l'update anche in progetti estesi.

1.1 Struttura di una pagina JSF

Sostanzialmente una pagina jsf include una serie di componenti, ciascuno dei quali arricchisce la pagina stessa con differenti funzionalitá in parte simili alle JSP, in parte diverse. Riassumiamo brevemente di cosa si tratta:

Un'applicazione jsf include:

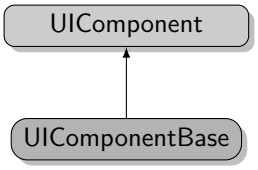
- ◇ Un insieme di elementi standard di interfaccia utente - widgets.
- ◇ Un insieme di library-tag per referenziare i componenti.

Offre inoltre i seguenti servizi:

- ◇ Un modello event-driven.
- ◇ Validazione a run-time dei componenti.
- ◇ Conversioni.
- ◇ Rendering dei componenti.

1.2 UI Components

Diamo, anzitutto uno sguardo ai principali componenti grafici che jsf ci fornisce per facilitare lo sviluppo delle interfacce grafiche (GUI). Le classi che forniscono tali componenti sono essenzialmente due: la classe **javax.faces.component.UIComponent** e la classe **UIComponentBase**



- ◇ UIMessage: stampa un messaggio di errore localizzato.
- ◇ UIMessages: stampa un insieme di messaggi di errore localizzati.
- ◇ UIUIColumn: rappresenta una singola colonna di dati in un componente UIData
- ◇ UICommand: rappresenta un controllo che produce unazione quando é attivato
- ◇ UIData: rappresenta un data binding ad una collezione di dati rappresentata da un'istanza DataModel
- ◇ UIForm: incapsula un gruppo di controlli che sottomettono dati all'applicazione. Questo componente é analogo al tag form in HTML
- ◇ UIGraphic: visualizza un'immagine
- ◇ UIInput: preleva I dati dall'input utente.
- ◇ UIOutcomeTarget: visualizza hyperlink nella forma di link o bottoni
- ◇ UIOutput: visualizza I data di output di una pagina
- ◇ UIPanel: gestisce il layout dei suoi componenti figlio
- ◇ UIParameter: rappresenta parametri
- ◇ UISelectBoolean: consente ad un utente di settare valori booleani in un controllo
- ◇ UISelectItem: rappresenta un singolo item di un insieme
- ◇ UISelectItems: rappresenta un insieme di item
- ◇ UISelectMany: permette ad un utente di selezionare piú item da un gruppo
- ◇ UISelectOne: permette ad un utente di selezionare un item da un gruppo
- ◇ UIViewParameter: rappresenta i parametri di una richiesta
- ◇ UIViewRoot: rappresenta la radice dell'albero dei componenti

1.3 Conversion model

Esiste la possibilitá di agganciare ai componenti di un'interfaccia i componenti bean, lato server del modello dati ed eventualmente prelevarne il contenuto secondo le necessitá .

1.4 Validation model

La validazione dei dati é una prassi comune, sia per chi progetta semplici moduli di raccolta dati in un blog sia per chi progetta grosse web-app come ad esempio il portale web di una banca o di un aeroporto. Java server faces mette a disposizione un meccanismo di validazione locale dei dati.

Vi sono essenzialmente due metodi principali di validazione. Il primo metodo consiste nell'innestare i tag di validazione all'interno dei tag del componente da validare *ad esempio in un componente di input quale una fieldtexx, si puó riportare la lunghezza massima di caratteri ammissibili, oppure in un campo mail, si puó forzare l'inserimento della @ oppure ancora ad esempio in HTML5 si possono definire delle vere e proprie espressioni regolari.* Il metodo globale invece si definisce direttamente nel file di configurazione aggiungendo le seguenti righe di codice:

Listing 1.1: faces-config.xml

```
<faces-config>
   <application>
      <default-validators>
         <validator-id>javax.faces.Bean</validator-id>
      </default-validators>
   </application>
</faces-config>
```

1.5 Modello guidato agli eventi

Esistono due soli tipi di **Application event** e sono:

- **action-event** (eventi scaturiti da azioni).
- **value-changed event** (eventi scaturiti dal cambiamento dello stato di un componente).

Capitolo 2

Navigation model

Il **navigation Model**, ovvero il modello di navigazione, consente di implementare attraverso l'uso di uno o piú file di configurazione xml, la logica di navigazione di una web application. I file di configurazione contengono l'insieme delle regole da elaborare quando si innesca un evento come il click del mouse su un link piuttosto che la pressione di un pulsante da tastiera. JSF mette a disposizione due approcci differenti: la cosiddetta **navigazione statica** che vedremo subito e la **navigazione dinamica**. La differenza tra i due tipi riguarda lo stato dell'input dell'utente, ma di questo ne parleremo successivamente, per ora concentriamoci sulla navigazione statica. Di seguito é riportata la struttura di un file di configurazione xml per una regola di navigazione.

Listing 2.1: rules.xml

```
<navigation-rule>
   <description></description>
   <from-view-id></from-view-id>
   <navigation-case>
      <from-outcome></from-outcome>
      <to-view-id></to-view-id>
   </navigation-case>
</navigation-rule>
```

Il codice contenuto all'interno del tag **navigation-rule**, definisce una regola di navigazione. All'interno della regola vi é la possibilitá di inserire una breve descrizione della regola stessa ed un identificativo della finestra dalla quale si scaturisce un evento che verrá catturato da un modulo chiamato **Navigationhandler**. Nel **navigation-case**, infine bisogna inserire l'id del componente associato alla regola e la pagina da visualizzare.

2.1 Navigazione statica

Consideriamo ora un esempio concreto. Supponiamo di trovarci in una pagina chiamata **home.xhtml** la quale include un pulsante avente id **btn-id** *(vedi Listato 2.3)*. Vogliamo che quando l'utente faccia click sul pulsante, si apra una nuova pagina chiamata **newpage.xhtml**. Il file che definisce la regola é riportato di seguito:

Listing 2.2: rules2.xml

```
<navigation-rule>
   <from-view-id>/home.xhtml</from-view-id>
   <navigation-case>
      <from-outcome>btn-id</from-outcome>
      <to-view-id>/newpage.xhtml</to-view-id>
   </navigation-case>
</navigation-rule>
```

Notate che nella **form-view-id** é stato indicato il nome della pagina principale in cui é presente il link ed il nome della pagina da mostrare alla pressione del link stesso. Nel file **home.xhtml** bisogna impostare l'action del link con lo stesso nome del **from-outcome**.

Listing 2.3: home.xhtml

```
...
<h:commandButton id="submit" action="btn-id" value="Submit" />
...
```

Quando l'utente clicca sul link il NavigationHandler tenta di fare un matching dell'action del link che é stato cliccato con tutte le etichette contenute nei rispettivi from-outcome di tutte le regole definite *(nota: in una stessa regola possono esserci piú navigation-case)*. Se trova una corrispondenza restituisce la pagina indicata nel to-view-id, nel caso in cui non vi sono match il NavigationHandler non ritorna nulla e si rimane sulla pagina corrente. Tutto quí ! Il modello statico é molto semplice ed intuitivo e si applica facilmente anche ad altri tipi di controlli.

2.2 Navigazione dinamica

Nella navigazione di tipo statico noi sappiamo giá a priori la pagina da visualizzare a seguito di un evento generato dall'utente. Il problema e che in una web-app, molto spesso capita di dover visualizzare una determinata pagina a seconda dell'input inviatoci dall'utente *(Si pensi alla compilazione di un modulo di registrazione, o al login utente)* e questo non é possibile con la navigazione statica. Quello di cui abbiamo bisogno é un meccanismo che sia in grado di prendere delle decisioni sulla base dello stato dell'input corrente. Questa caratteristica ci viene fornita dall'approccio di navigazione dinamico, argomento di

questo paragrafo.

Vediamo cosa cambia rispetto all'approccio statico. Anzitutto nell'action del commandButton non bisogna inserire un'etichetta costante-stringa, ma un metodo vero e proprio dipendente dallo stato dei dati inseriti dagli utenti. Questi metodi fanno parte del modello dati e vengono chiamati in gergo **backing-bean functions**.

Listing 2.4: home.xhtml

```
...
<h:commandButton label="check" action="#{utente.checkData}"/>
...
```

Il metodo **checkData** é contenuto in un oggetto **utente**, che contiene ad esempio i dati di tutti gli utenti registrati sul sito e restituisce una stringa di successo o di fallimento a seguito di un controllo - ad esempio se l'utente risulta essere registrato sul sito o meno.

Listing 2.5: Utenti.java

```
// java bean
...
String checkData()
{
   if (userRegister())
      return "success";
   else
      return "failure";
}
...
```

Sulla base del valore restituito si possono realizzare diversi casi in una regola di navigazione come mostrato di seguito:

Listing 2.6: rules.xhtml

```
<navigation-rule>
   <description> login utente </description>
   <from-view-id>/home.xhtml</from-view-id>

   <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/login.jsp</to-view-id>
   </navigation-case>

   <navigation-case>
      <from-outcome>failure</from-outcome>
      <to-view-id>/error.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
```

Capitolo 3

JSF LifeCycle

Prima di mostrare il ciclo di vita (LyfeCycle) JSF, ricordiamo che il modello JSF, si basa, come giá visto sul pattern MVC (Model View Control), il quale ha permesso un vero e proprio disaccoppiamento tra: **Interfaccia (Presentation)**; **logic-business (Control)** e **data-model (DB-Persistence)**. Il ciclo di vita descrive ció che accade nell'intervallo di tempo da quando si fa una richiesta di una pagina (un istante dopo) fino a quando si riceve una risposta dal server (un istante prima). Bisogna distinguere essenzialmente due tipologie di richieste che il framework é in grado di gestire: Le **richieste iniziali** (quando una pagina é richiamata semplicemente senza la sottomissione di dati da parte dell'utente e/o di altri programmi. Le **richieste postback** generate quando si inviano ad esempio i dati con un form. La differenza tra le due é semplice: in una richiesta iniziale si svolgono la prima e l'ultima fase del lifecycle, mentre in una postback dal momento che vendono inviati dei dati al server, bisogna passare per tutte le fasi che che descriveremo di seguito piú approfonditamente.

- ◇ 1. Restore View
  Quando viene richiesta una pagina il nucleo centrale di controllo del framework JSF si fa carico di generare la pagina vera e propria sotto forma di struttura ad albero in memoria. Viene quindi generato l'albero noto anche con il nome tree-view di tutti i componenti della pagina ed inoltre vengono inizializzati i validatori ed i gestori degli eventi. Nel caso in cui la pagina non fosse stata mai realizzata (siamo cioé nel caso di una nuova pagina), JSF si deve occupare di registrarla nel FacesContext, l'istanza che raccoglie tutte le richieste, cui hanno accesso tutti i gestori ed i componenti. Se la pagina era giá stata creata allora jsf si occuperá solo di aggiornare le parti mancanti.

- ◇ 2. Apply Request Values
  Vengono estratti per ogni componente ed inseriti nell'albero i parametri della richiesta sotto forma di stringa mediante il metodo decode(). In caso di errori e/o eccezioni viene fatto un appunto sul file FacesContext.

- ◇ 3. Process Validation
  Quando vengono estratti i parametri dalla richiesta il formato restituito é string, di conseguenza bisogna effettuare una conversione in quanto il

modello dati ha una sua rappresentazione interna differente. Per ogni componente vengono di conseguenza eseguiti i relativi metodi validatori. In caso di errori e/o eccezioni viene fatto un appunto sul file FacesContext e si passa subito alla fase di Render Response in modo tale da visualizzare subito gli errori.

◇ 4. Update Model Values
Se la fase di validazione si conclude correttamente, si passa alla fase di aggiornamento del modello dati. I metodi backing-beans vengono eseguiti, cosicché tutti i java beans corrispondenti vengono aggiornati. In caso di errori e/o eccezioni viene fatto un appunto sul file FacesContext e si passa subito alla fase di Render Response in modo tale da visualizzare subito gli errori.

◇ 5. Invoke Application
In questa fase vengono gestiti tutti gli eventi generati dall'utente - inviati ai vari listener associati.

◇ 6. Render Response
Questa á la fase finale di rendering in cui viene costruita la vista; eventuali errori nelle fasi precedenti, come visto passano il controllo a questa fase, in modo da bloccare il ciclo di vita e mostrare gli errori.