

## Introduzione a JSF: Le basi in un esempio.

Giulio Rossetti

20/09/2007

### Introduzione:

Java Server Faces è un framework, le cui specifiche sono rese disponibili dalla Sun, che permette di realizzare GUI in ambiente Enterprise utilizzando un procedimento molto simile a quello adottato per svolgere il corrispettivo lavoro in ambito standalone.

Nato dalla necessità di fornire un modello standard *de iure* (Struts è stato per anni, ed è tuttora, il framework che si è imposto come standard *de facto*) per la gestione delle interfacce grafiche in ambito enterprise applica sistematicamente il pattern MVC (Model-View-Controller) e introduce un modello programmatico basato sulla gestione degli eventi.

I principali vantaggi nell'uso di questo framework risultano essere:

- Esistenza di componenti predefiniti che avvicinano la programmazione web a quella degli ambienti RAD, consentendo allo sviluppatore di realizzare in breve tempo interfacce web con la stessa semplicità offerta da ambienti come .NET, semplicemente “collegando” elementi di business logic lato server tramite catene di eventi.
- Elementi GUI “intelligenti” in grado di validare in prima persona i dati inseriti dall'utente e di archiviare e caricare ondemand il proprio stato da bean memorizzati lato server denominati “model-object”.
- Definizione di un nuovo paradigma di Event Handling che avvicina la programmazione in ambito web alla tipologia di gestione asincrona degli eventi utilizzata nelle applicazioni client-server.
- Indipendenza dal markup language: ogni modello di interazione lato server viene realizzato lato client tramite Renderer diversificati che producono un'interfaccia utente in grado di soddisfare i requisiti funzionali del server al meglio delle possibilità della piattaforma utilizzata dall'utente.

Vediamo ora di proporre, tramite un esempio, le basi per implementare una pagina contenente un banale form che ci chiede il nome e lo stampa all'interno di un messaggio nella pagina seguente.

### Ambiente di sviluppo:

Ormai buona parte degli ambienti di sviluppo presentano tool per la gestione visuale sia della parte relativa alla composizione delle pagine in Jsf sia per quanto riguarda la gestione dei suoi file di configurazione (in molti tool viene offerta la possibilità di “disegnare” la mappa di navigazione del sito e da tale mappa vengono generate automaticamente le modifiche ai file di configurazione che vedremo in seguito).

Per questo esempio è stato utilizzato Eclipse con installato il plugin MyEclipse (una valida alternativa possono essere i web-tools disponibili sul sito di Eclipse).

Le librerie di Jsf utilizzate sono quelle della implementazione realizzata da Apache.

### Modello di navigazione:

Per fare quello che ci siamo riproposti vediamo come è strutturato il modello di navigazione tra le pagine in Jsf.

Tale modello di navigazione è realizzato in modo da sostituire il concetto di collegamento ipertestuale (link) tra pagine con quello di “azione” di navigazione. Tramite uno specifico file (faces-config.xml), di cui vedremo in seguito la sintassi e gli scopi, è infatti possibile definire delle regole di navigazione tra le pagine della propria web application.

Tali regole prevedono una sintassi di questo tipo:

```
<navigation-rule>
```

```
    <from-view-id>/index.jsp</from-view-id>
```

```

<navigation-case>
    <from-outcome>avanti</from-outcome>
    <to-view-id>/pagina1.jsp</to-view-id>
</navigation-case>

```

```
</navigation-rule>
```

Come è facilmente comprensibile dall'esempio abbiamo definito una regola il cui effetto può essere riassunto nel seguente modo:

- Se mi trovo nella pagina index.jsp e una azione produce una stringa “avanti” allora carico pagina1.jsp -

Le azioni che prendiamo in considerazione non sono altro che metodi, i quali ritornano appunto una String al termine dell'elaborazione, che possiamo definire a nostro piacimento ed assegnare ai componenti di JSF che consentono il submit di una form o, più semplicemente, la navigazione tra pagine.

Questo meccanismo, oltre a rendere omogenea la modalità di navigazione al modello di gestione degli eventi, consente di avere un maggiore controllo sul flusso di caricamento delle pagine e di rendere un'operazione “atomica” la navigazione verso la pagina desiderata con i side-effect che possono essere svolti dal metodo invocato.

Vediamo ora il codice delle nostre prime due pagine.

### La View:

La parte di presentazione è affidata a pagine jsp a cui sono aggiunte le tag libraries di JSF. Tali tag libraries definiscono gli oggetti che possono essere utilizzati all'interno della pagina (in questo articolo prenderemo in considerazione solo le due tag libraries standard; esistono anche progetti che si fanno carico di aggiungere funzionalità e oggetti a Jsf) e devono essere definite all'inizio di ogni singola pagina jsp con questa descrizione:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

A seguito della dichiarazione d'uso sarà quindi possibile utilizzare vari oggetti all'interno della pagina: mostriamo ad esempio il codice della nostra pagina index.jsp (n.b: quanto mostrato in seguito si trova all'interno dei tag <body>) :

```

<f:view>
    <h:form>
        <h:outputText value="Inserisci il tuo nome" align="center"/>
        <h:inputText value="#{utente.nome}"/></h:inputText>
        <h:commandButton value="Cliccami" action="avanti" />
    </h:form>
</f:view>

```

Questo primo esempio propone una pagina con del testo, un campo per inserire del testo e un button che effettua la navigazione.

Come si può notare il corpo della pagina contenente oggetti definiti in JSF deve essere racchiuso tra i tag <f:view> e, condizione necessaria per gli elementi dei form ma fortemente consigliata per qualsiasi altro elemento(per evitare incompatibilità con le varie versioni), i tag <h:form>. Ogni pagina può ospitare più form, non annidati, in tal caso si consiglia di specificarne il nome tramite l'attributo “name” dell'oggetto <h:form>.

La pagina1.jsp avrà quindi una struttura simile. Omettendo adesso i tag view e form mostriamo il contenuto di questa pagina:

```

<h:outputText value="Benvenuto nella tua prima pagina JSP " align="center"/>
<h:outputText value="#{utente.nome}" align="center"/></h:outputText>

```

A seguito della navigazione otterremo quindi un messaggio di benvenuto che ci mostrerà il testo da noi inserito nel campo presente in index.jsp.

### **Java Bean e Binding:**

In alcune proprietà degli oggetti JSF sopra usati abbiamo utilizzato una scrittura del tipo: `value="#{oggetto.metodo}"`.

Questa è la sintassi per specificare a JSF di eseguire il metodo `set/get` (non occorre specificare il tipo del metodo poiché ciascun oggetto conosce il tipo di azione che può eseguire ciascuna sua proprietà per cui sia prevista la possibilità di binding) di un Bean.

I Java Bean sono oggetti il cui costruttore non accetta parametri e le cui variabili di istanza, dichiarate private, sono ricavate e impostate tramite metodi la cui firma, per convenzione, è del tipo: `setVariabile(Object valore)`, `getVariabile()` (o `isVariabile()` nel caso debba ritornare un boolean).

L'uso del binding tra oggetti e valori ricavati/inviati ad una form in JSF esula il programmatore dalla gestione manuale degli oggetti request, response e session utilizzati per salvare lo stato attuale del servizio web a seguito di ciascuna richiesta ricevuta e elaborazione effettuata.

Altro uso comune del binding è quello di associare alla proprietà "action" (presente in molti componenti, tra cui `commandLink` e `commandButton`), al posto di una stringa di testo, un metodo che, eseguite tutte le operazioni volute dal programmatore, ritorni una `String`.

Il nostro bean sarà quindi il seguente:

```
package mioPackage;

public class user{

    private String nome;

    public user(){

    }

    public String getNome(){

        return nome;

    }

    public void setNome(String nome);

        this.nome=nome;

    }

}
```

### **File di configurazione:**

Il file in cui vengono specificate le modalità di navigazione tra le pagine e il binding tra i beans e le entità Jsf è il già citato `faces-config.xml`. Tale file deve essere inserito nella cartella `/WEB-INF/` del progetto ed è fondamentale per la corretta esecuzione del progetto stesso (insieme al file `web.xml`). La nostra prima applicazione presenta una versione minimale di questo file:

```
<faces-config>

    <navigation-rule>

        <from-view-id>/index.jsp</from-view-id>

        <navigation-case>

            <from-outcome>avanti</from-outcome>

            <to-view-id>/pagina1.jsp</to-view-id>

        </navigation-case>

    </navigation-rule>

</faces-config>
```

```

        </navigation-case>
    </navigation-rule>

    <managed-bean>
        <managed-bean-name>utente</managed-bean-name>
        <managed-bean-class>mioPackage.user</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
</faces-config>

```

La prima parte è quella che abbiamo già visto in precedenza per la navigazione, adesso ci interessa capire come vengono gestiti i bean.

Nella pagina index.jsp all'interno del componente inputText si fa riferimento ad un metodo che viene raggiunto tramite il binding a "utente.nome". Nel xml sopra riportato si può notare che "utente" non è altro che una stringa utilizzata da Jsf per recuperare una classe (che deve esser presente in /WEB-INF/classes) avente come nome user e appartenente al package mioPackage. Questa notazione permette al programmatore di rendere disponibili alla View oggetti che fanno parte della logica del programma, di stabilire un ponte tra la presentazione e l'applicativo. Inoltre è l'unico posto in cui può essere specificato il luogo in cui l'oggetto deve essere salvato una volta creato (va specificato infatti se il suo arco di vita è limitato alla request o alla session).

Altro file di configurazione fondamentale per una web application è il file web.xml. Anche esso nella cartella /WEB-INF/, ha lo scopo di definire i filtri e il contesto in cui l'applicazione deve girare. Per utilizzare Jsf sono fondamentali le seguenti aggiunte alla normale configurazione di web.xml:

In questa parte si specifica che vogliamo usare la nostra configurazione del faces-config.xml,

```

<context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/faces-config.xml</param-value>
</context-param>

```

Qui specifichiamo che verrà usato Jsf e che quindi all'avvio dell'applicazione dovrà essere caricata la servlet che consente al framework di interpretare le chiamate ai suoi componenti e di svolgere il suo lavoro,

```

<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

```

Ed infine si stabilisce che le nostre pagine non saranno più raggiungibili come index.jsp e pagina1.jsp ma come index.faces e pagina1.faces (per ricordarci che stiamo utilizzando Jsf, i nomi dei file non vengono modificati è solamente una mappatura effettuata sul web server.)

```

<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
</servlet-mapping>

```

Se abbiamo compilato correttamente i file di configurazione, scritto senza errori le nostre pagine jsp e il nostro bean,

adesso siamo pronti a far girare la nostra prima web application realizzata con Jsf.

### **Cosa è rimasto fuori:**

Quello che è qui presentato non è altro che una minima parte di quanto offre questo framework. Agli oggetti standard possono essere aggiunti elementi complessi che necessitano, una volta compreso il loro funzionamento, di pochi passi per ottenere risultati di buon livello, esiste poi la possibilità di creare con poche righe di codice dei validator personalizzati per i campi delle form.

E' possibile decidere quali campi sono obbligatori semplicemente specificandolo all'interno del componente stesso e lasciare al framework il compito di controllare se ci son stati errori di immissione o omissioni.

Per ottenere modifiche su oggetti salvati in sessione è prevista una modalità di gestione (tramite l'oggetto FacesContext) che consente al programmatore di superare l'astrazione proposta dal framework e di operare manualmente sugli oggetti session, request e response.

In conclusione, Java Server Faces è un ottimo framework che consente al programmatore di scindere con facilità la logica della web application dalla sua parte di presentazione e che rende più facile la gestione di un progetto di dimensioni accettabilmente grandi.