Tecnologie di Sviluppo per il Web

Applicazioni Web J2EE
Framework per il Modello 2
Java Server Faces

versione 1.1

G. Mecca – mecca@unibas.it – Università della Basilicata

Sommario

Introduzione

- Utilizzando framework per il modello 2
  - ⇨lo stile di programmazione per le applicazioni Web diventa molto simile a quello delle applicazioni desktop con architettura MVC
- Con alcuni limiti significativi
  - ⇨che riguardano in particolare lo strato della vista e la tecnologia per la produzione degli schermi

Introduzione

- Nelle applicazioni desktop
  - ⇨tradizionalmente toolkit ricchi
  - ⇨con molti componenti grafici (es: menu, tabelle, alberi, calendari, ecc.)
  - ⇨e un sistema ricco di eventi (es: eventi di selezione, eventi del mouse, eventi relativi al cambiamento dei valori ecc.)
  - ⇨questo consente di sviluppare applicazioni gradevoli graficamente e molto interattive

Introduzione

- Nelle applicazioni Web
  - ⇨la tecnologia per la vista è l'HTML
  - ⇨il sistema di controlli è estremamente povero
  - ⇨in sintesi: campi di testo e password, bottoni, caselle di selezione, pulsanti "radio" e checkbox
  - ⇨inoltre un supporto limitato per liste e tabelle
  - ⇨un sistema di eventi poverissimo: richieste get e richieste post al server

Introduzione

- Negli ultimi anni
  - ⇨il problema delle interfacce utente per le applicazioni Web è stato molto studiato
  - ⇨con l'obiettivo di sviluppare tecnologie per "rich client interfaces"
  - ⇨ovvero interfacce più ricche di quelle HTML ordinarie per rendere più interattiva e appetibile la fruizione delle applicazioni Web
  - ⇨sono state proposte varie soluzioni

Introduzione

- Client Swing
  - ⇨idea: sostituire Swing all'HTML come tecnologia lato client
  - ⇨vantaggio: ricchezza del toolkit; svantaggio: richiede JRE sulla macchina client
- AJAX
  - ⇨idea: utilizzare pesantemente JavaScript sul client per catturare gli eventi, e segnalare gli eventi al server con richieste HTTP

Introduzione

- La risposta ufficiale dello JCP
  - ⇨Java Server Faces
- Java Server Faces (JSF)
  - ⇨un framework per lo sviluppo di interfacce utente ricche in applicazioni J2EE
  - ⇨fornisce un toolkit ricco ed estensibile di componenti basati su HTML
  - ⇨in estrema sintesi: Struts + Swing

Introduzione

- Vantaggi di un toolkit estensibile
  - ⇨consente di rendere maggiormente interattive le applicazioni Web
  - ⇨consente di sviluppare nuovi componenti o personalizzare quelli esistenti
  - ⇨consente lo sviluppo di ambienti RAD e di editor grafici (es: Sun One Studio)
  - ⇨consente uno stile di sviluppo molto simile a quello dei toolkit per applicazioni desktop

Introduzione

- Vantaggi di un toolkit estensibile (cont.)
  - ⇨inoltre, il sistema di componenti è indipendente dalla tecnologia per la produzione degli schermi
  - ⇨consente di utilizzare vari “rendering kit” per visualizzare i componenti grafici
  - ⇨il rendering kit standard è orientato a HTML, ma è possibile pensare a rendering WML, rendering XML+XSL, rendering SVG ecc.

Introduzione

- Svantaggi
  - ⇨complessità molto maggiore del framework
- Infatti
  - ⇨Java Server Faces è un framework molto complesso
  - ⇨che richiede la comprensione di vari meccanismi e di una semantica non banale

Introduzione

- Storia di JSF
  - ⇨standard arrivato alla versione 1.1
  - ⇨una lunghissima gestazione (la versione 1.0 dello standard ha richiesto vari anni)
- Il coordinatore dello standard JSF
  - ⇨Craig McClanahan, l'ideatore di Struts
  - ⇨di conseguenza, JSF eredita molte idee e molte esperienze di Struts
  - ⇨arricchendolo da vari punti di vista

Introduzione

- In generale
  - utilizzanto JSF non c'è più nessuna necessità di utilizzare Struts
  - applicazioni Struts esistenti possono integrare JSF utilizzando la libreria struts-faces
  - è stato varato il progetto Jakarta Shale che integrerà il meglio di JSF e di Struts

Introduzione

- Implementazioni di JSF
  - due implementazioni principali
- Reference Implementation
  - promossa dalla Sun e utilizzata in Sun One Studio, disponibile su java.sun.com
- Apache MyFaces
  - implementazione open-source del progetto Apache, disponibile su myfaces.apache.org

Introduzione

- In sintesi, JSF offre, come Struts
  - ⇨un controllore frontale per il modello 2
  - ⇨un framework di convalida dei dati
- In aggiunta, JSF offre
  - ⇨un ricco insieme di componenti grafici utilizzabili negli schermi delle applicazioni
  - ⇨un insieme ricco di eventi
  - ⇨un framework per il binding
  - ⇨un framework di “dependency injection”

Componenti

- Il toolkit di componenti standard
  - ⇨prevede un numero relativamente piccolo di componenti grafici
  - ⇨implementati come tag personalizzati di due librerie principali
- Le librerie di tag

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

Componenti

>> jsf-components
>> myfaces-examples

- Ma...
  - ⇨il programmatore può sviluppare i propri componenti per estendere il framework
- Alcuni esempi
  - ⇨l'applicazione jsf-components distribuita con la reference implementation
  - ⇨l'applicazione myfaces-examples, che mostra l'utilizzo di vari componenti non standard sviluppati nell'ambito del progetto MyFaces

Componenti

- Un concetto centrale
  - ⇨uno schermo JSF ha due diverse "incarnazioni"
- L'implementazione pratica
  - ⇨una pagina JSP contenente tag personalizzati che invia l'HTML al client
- La rappresentazione interna al framework
  - ⇨un albero di oggetti, ciascuno dei quali rappresenta un componente (contenitore)

Componenti

- Gli oggetti della vista
  - ⇨implementati nel package javax.faces.components
  - ⇨discendono da javax.faces.components.UIComponent
  - ⇨una classe per ciascun tipo di componente
  - ⇨es: UIForm, UIInput, UIGraphic, UIPanel ...
  - ⇨a ciascuno corrisponde un tag di azione

La Filosofia

- La filosofia di programmazione
  - ⇨è ispirata alle applicazioni desktop
- In particolare
  - ⇨l'utente produce gli schermi sotto forma di pagine JSP come alberi di componenti grafici
  - ⇨e associa gestori agli eventi scatenati dai gesti dell'utente
  - ⇨le richieste HTTP servono per gestire gli eventi e aggiornare lo schermo

La Filosofia

- Il senso di una richiesta HTTP
  - ⇨“sulla vista è stato scatenato un evento: gestiscilo”
  - ⇨non corrisponde ad un semplice comando, e quindi è diverso da quello delle applicazioni modello 2 tradizionali
  - ⇨la gestione dell’evento può portare ad aggiornare lo schermo corrente o ad inoltrare l’utente ad un altro schermo

La Filosofia

- Tipica evoluzione di un’applicazione
  - ⇨viene ricevuta la richiesta iniziale; viene stabilita la sessione e prodotto il primo schermo
  - ⇨il framework costruisce lo schermo sotto forma di albero di componenti (oggetti UIComponent)
  - ⇨poi lo rende al client sotto forma di HTML

La Filosofia

- A questo punto
  - ⇨si è prodotta una “separazione” nella vista
  - ⇨il framework ha una sua rappresentazione interna ad oggetti dello schermo corrente
  - ⇨ma l’utente interagisce con lo schermo reso dal client sotto forma di HTML
  - ⇨è necessario ricucire questa separazione in ogni transazione HTTP successiva

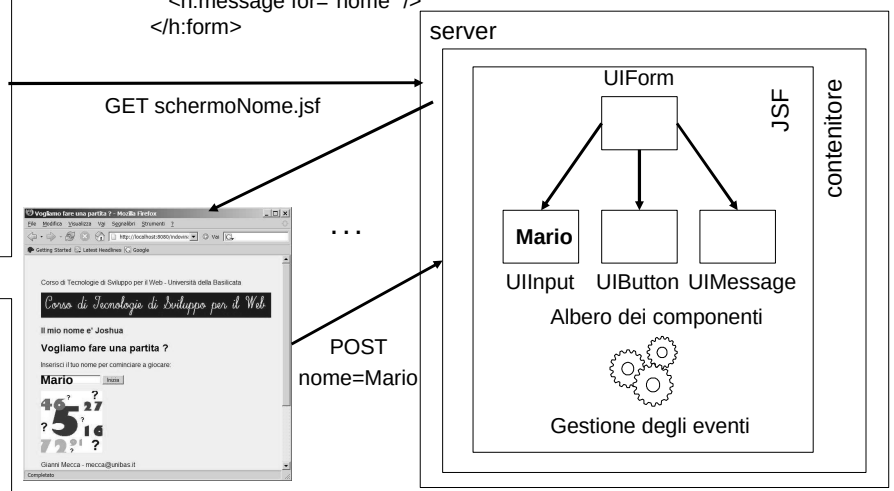
La Filosofia

- Stato della vista
  - ⇨al termine della transazione, il framework salva nella sessione un riferimento allo stato della vista (ovvero all’albero di componenti)
- Le richieste successive
  - ⇨sono originate da eventi scatenati dall’utente interagendo con lo schermo
  - ⇨e richiedono di gestire gli eventi in questione

La Filosofia

- Di conseguenza
  - ⇨per ciascuna richiesta HTTP successiva alla prima, il primo passo consiste nel ripristinare lo stato della vista
  - ⇨ovvero prelevare dalla sessione l'albero di componenti dello schermo corrente
  - ⇨e aggiornarli con i dati forniti dall'utente e prelevati dalla query string
  - ⇨poi è possibile gestire gli eventi


```
<h:form>
  <h:inputText id="nome" />
  <h:commandButton value="Inizia" action="..." />
  <h:message for="nome" />
</h:form>
```



La Filosofia

- Di conseguenza
  - ⇨con JSF ci sono due categorie di richieste
- I categoria
  - ⇨richieste ordinarie, in cui viene esplicitamente richiesto un URI .jsf (es: <a href="esci.jsf">)
- II categoria
  - ⇨richieste per la gestione di eventi

La Filosofia

- Le richieste del secondo tipo
  - ⇨si chiamano richieste di postback
- Postback
  - ⇨fenomeno per cui una pagina client effettua una richiesta (post o get) alla stessa pagina server che l'ha generata (per gestire eventi)
  - ⇨la quale poi può decidere di rendere di nuovo la pagina client aggiornata, o effettuare un inoltro

La Filosofia

- Un problema
  - ⇨le richieste HTTP NON possono essere indirizzate direttamente alle pagine JSP (il contenitore le eseguirebbe immediatamente)
  - ⇨è necessario un componente intermedio di controllo che gestisca la corrispondenza con l'albero di componenti
  - ⇨bisogna alterare in qualche modo gli URI

La Filosofia

- La soluzione di JSF
  - ⇨gli URI degli schermi sono URN costruiti con la seguente convenzione
  - ⇨nome della pagina JSP corrispondente
  - ⇨con un estensione diversa
- Esempio
  - ⇨schermoNome.jsp -> schermoNome.jsf

La Filosofia

- Nel deployment descriptor
  - ⇨tutti gli URI di tipo *.jsf vengono inoltrati al controllore frontale del framework

```
<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>*.jsf</url-pattern>
</servlet-mapping>
```


Componenti

- La pagina indice index.jsp
  - ⇨contiene solo una istruzione di redirezione al primo schermo reale dell'applicazione
  - ⇨in modo da avviare l'applicazione
- Tipico contenuto di index.jsp
  - ⇨<jsp:forward page="primoSchermo.jsf" />

Programmazione MVC con JSF

- Il flusso di controllo
  - ⇨una volta ripristinata la vista, vengono gestiti sul server gli eventi
  - ⇨es: evento associato al click sul bottone di una form
  - ⇨publish&subscribe: attraverso i tag della libreria e l'expression language di JSF è possibile registrare come gestori per gli eventi metodi diversi

Programmazione MVC con JSF

- L'expression language di JSF
  - ⇨praticamente sintatticamente e semanticamente identico a quello di JSP 2.0
  - ⇨prima differenza: le espressioni hanno la sintassi #{<e>} e non ${<e>}
  - ⇨seconda differenza: il linguaggio può essere usato solo nei componenti JSF
  - ⇨normalmente sono presenti entrambi i linguaggi nelle pagine JSP

Programmazione MVC con JSF

- Nota
  - ⇨l'expression language di JSF viene utilizzato essenzialmente per produrre il binding dei bean ai componenti della vista
- Due tipologie di binding
  - ⇨binding dei valori (es: #{partita.nome})
  - ⇨binding dei metodi usato per gli eventi (es: #{azioneIniziaPartita.esegui}

Programmazione MVC con JSF

- Gli eventi gestiti da JSF
  - ⇨"value change events" (il valore di un componente è cambiato)
  - ⇨"action events" (è stato selezionato un link, menu o bottone)
  - ⇨"lifecycle events" (eventi collegati al ciclo di vita della richiesta)

Programmazione MVC con JSF

- Per un action event
  - ⇨è possibile specificare due gestori: una action e un actionListener
  - ⇨differenza: una action realizza azioni applicative, un actionListener aggiorna la vista
- Per eseguire gli eventi
  - ⇨è necessario effettuare una richiesta al server

Programmazione MVC con JSF

- Due possibili modelli di esecuzione
  - ⇨caching ed esecuzione cumulativa (con una richiesta vengono eseguiti vari eventi)
  - ⇨esecuzione singola o “autopostback” (ogni evento scatenato sul client genera una richiesta http)
- In JSF, attualmente
  - ⇨è possibile utilizzare solo l’esecuzione singola
  - ⇨in particolare, per i value change events, devono essere catturati dal client e generare una richiesta http utilizzando JavaScript

Programmazione MVC con JSF

>> schermoNome.jsp

- Nota
  - ⇨i gestori di eventi non devono implementare nessuna interfaccia particolare
  - ⇨purchè siano dichiarati nel file di configuraz. del framework, faces-config.xml nella sezione dei bean gestiti (“managed bean”)
  - ⇨e i metodi utilizzati rispettino i prototipi previsti per gestire gli opportuni eventi
  - ⇨la tecnica è fortemente basata sulla rifless.

Programmazione MVC con JSF

- Esecuzione del gestore di evento
  - ⇨metodo normalm. contenuto all’interno di una azione
  - ⇨accede allo stato del modello (lavorando sui bean in richiesta, sessione, applicazione) e lo modifica
  - ⇨può accedere ai componenti grafici della vista e modificarli
  - ⇨produce un esito che provoca l’inoltro ad un diverso schermo successivo
  - ⇨se l’esito è null, non è necessario effettuare inoltri; basta rendere la versione aggiornata dello schermo corrente

Programmazione MVC con JSF

- Le regole di navigazione
  - ⇨sono specificate nel file faces-config.xml
  - ⇨attraverso elementi di tipo navigation-rule
  - ⇨sintassi molto generale
  - ⇨consente di specificare inoltri del tipo schermo di partenza-esito-schermo successivo
  - ⇨o anche solo stringa-schermo successivo

Programmazione MVC con JSF

>> faces-config.xml

- Nota
  - ⇨le regole di navigazione sono diverse da quelle del modello 2 tradizionale
  - ⇨il punto di partenza è sempre uno schermo
  - ⇨a causa del postback il processo è: ricostruzione dello schermo, gestione degli eventi collegati alla richiesta (normalmente più di uno) e aggiornamento dello schermo, con eventuale inoltro ad altro schermo

Convalida e Binding

- La convalida dei dati
  - ⇨nella vista è possibile specificare vincoli di vario genere sui componenti
  - ⇨questi vengono utilizzati durante il processo di ripristino dei valori
- Due tipologie di vincoli
  - ⇨vincoli sulla conversione dei valori (es: intero, data ecc.) e vincoli di convalida dei valori (es: non nullo, tra min e max, lungh. max ecc.)

Convalida e Binding

- Il processo di gestione della query string
  - ⇨dopo aver ripristinato l'albero dei componenti, i parametri vengono prelevati dalla query string e salvati come "valori locali" nei componenti
  - ⇨successivamente vengono sottoposti automaticamente a conversione e convalida, applicando e verificando i vincoli
  - ⇨in caso di errori, vengono prodotti opportuni messaggi di errore

Convalida e Binding

>> schermoTentativo.jsp

- Per la visualizzazione dei messaggi
  - ⇨è possibile utilizzare gli opportuni componenti di tipo message
  - ⇨ma i messaggi prodotti dal framework di convalida sono in inglese
  - ⇨per produrre versioni italiane, è necessario creare un resource bundle in cui vengono fornite versioni italiane delle chiavi utilizzate

Convalida e Binding

- Naturalmente
  - ⇨nel caso sia necessario un convalidatore non previsto è molto facile svilupparlo
  - ⇨implementando il tag relativo e l'interfaccia Validator
- Di fatto però
  - ⇨l'utilizzo del framework rende inutile lo sviluppo dei formBean

Convalida e Binding

- Infatti
  - ⇨i parametri della query string vengono verificati dal framework usando convertitori e convalidatori
  - ⇨non è necessario salvarli in un componente intermedio
- Inoltre
  - ⇨una volta passata la convalida, posso pensare di salvarli direttamente nelle azioni, o meglio nei bean del modello

Convalida e Binding

- Il sistema di binding
  - ⇨consente, attraverso l'expression language, di associare una proprietà del modello direttamente ad un componente della vista
  - ⇨il componente deve essere un bean gestito
  - ⇨il framework gestisce la sincronizzazione tra vista e modello (chiama il metodo get per produrre lo schermo, e il metodo set quando arriva una richiesta con il valore)

Convalida e Binding

- La terminologia di JSF
  - ogni bean gestito che viene utilizzato per effettuare il binding di proprietà o metodi nei componenti grafici viene detto “backing bean”
- Backing bean
  - bean gestito dal framework che viene gestito come “back end” della vista per la logica applicativa

Convalida e Binding

- Un esempio
  - in schermoNome.jsp posso associare direttamente il valore del nome alla proprietà nome di Partita
  - in schermoTentativo.jsp posso associare direttamente il valore del tentativo (dopo conversione e convalida) con la proprietà tentativo di Partita

Convalida e Binding

- Come conseguenza
  - ⇨il codice dell'applicazione è snellito (non esistono i formBean, le azioni sono ridotte per via della mancata convalida)
  - ⇨il ciclo di vita della richiesta diventa più complesso per via della gestione dell'albero di componenti e ora del processo di convalida e binding

Convalida e Binding

- Nota
  - ⇨oltre che richiedere il binding dei valori dei controlli, posso richiedere anche il binding di un intero componente grafico (UIComponent)
  - ⇨in modo che sia accessibile come proprietà di una azione
  - ⇨esempio: per modificarne lo stato all'interno dell'azione

Il Ciclo di Vita della Richiesta

- Di conseguenza
  - ⇨le richieste successive alla prima hanno un ciclo di vita articolato all'interno del framework
  - ⇨articolato in varie fasi che corrispondono alle varie operazioni effettuate dal framework
  - ⇨tra una fase e l'altra possono essere scatenati e gestiti eventi diversi

Il Ciclo di Vita della Richiesta

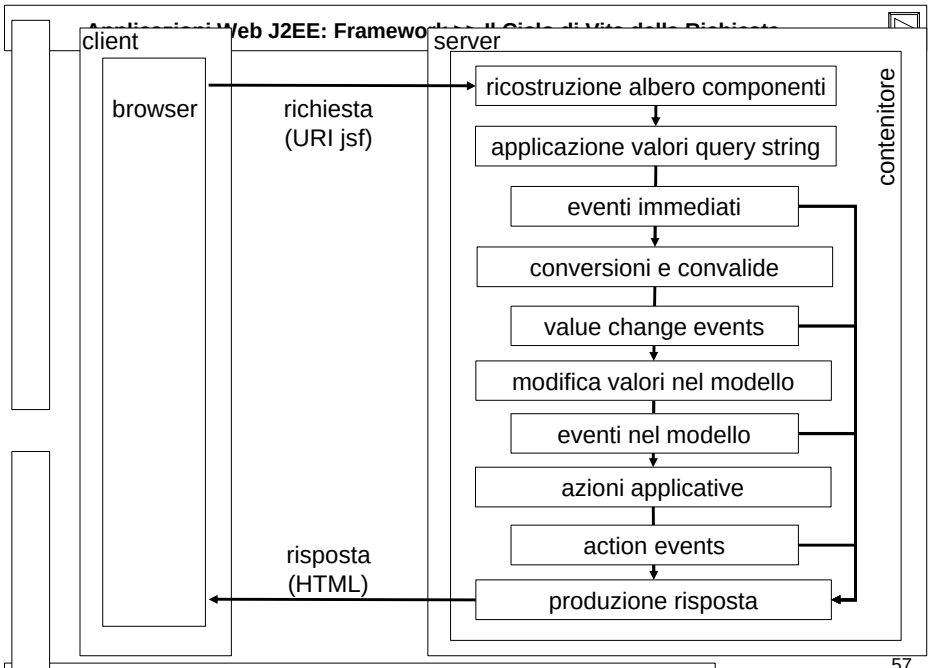
- Il ciclo di vita completo della richiesta
  - ⇨I fase: ricostruzione dell'albero di componenti
  - ⇨II fase: applicazione dei valori della query string
  - ⇨III fase: conversioni e convalide
  - ⇨IV fase: aggiornamento delle proprietà del modello
  - ⇨V fase: esecuzione delle azioni applicative
  - ⇨VI fase: produzione della risposta

Il Ciclo di Vita della Richiesta

- Tra le varie fasi
  - ⇨vengono gestiti gli eventi contenuti nella richiesta
- In particolare
  - ⇨gli eventi di tipo “value changed” vengono gestiti subito dopo la fase di convalida
  - ⇨gli eventi di tipo “action” vengono gestiti durante la fase di esecuzione delle azioni applicative

Il Ciclo di Vita della Richiesta

- Una tipologia di eventi particolari
  - ⇨gli eventi “immediati”, ovvero gli eventi associati ai componenti “immediati”
  - ⇨in sintesi: componenti per la gestione dei quali non è necessario attraversare tutto il ciclo di vita
- Un esempio
  - ⇨il pulsante “Annulla” di una form di registrazione con input obbligatori



Il Ciclo di Vita della Richiesta

- In ogni momento, un gestore di evento
  - ⇨può interrompere il ciclo di vita
  - ⇨eventualmente chiedendo di produrre la risposta
- I caso: interruzione del ciclo di vita
  - ⇨metodo FacesContext.responseComplete()
- II caso: produrre la risposta
  - ⇨metodo FacesContext.renderResponse()

Il Ciclo di Vita della Richiesta

- L'oggetto di tipo FacesContext
  - ⇨package javax.faces.context
  - ⇨consente la comunicazione con il framework durante la gestione della richiesta
  - ⇨accessibile nei managed bean usando:
  - ⇨FacesContext context = FacesContext.getCurrentInstance();

Il Ciclo di Vita della Richiesta

>> Utilita.java

- Utilizzando il FacesContext
  - ⇨è possibile avere accesso al cosiddetto "contesto esterno" dell'applicazione
  - ⇨ovvero agli oggetti richiesta, sessione e servletContext predisposti dal contenitore
  - ⇨può essere utile, nelle azioni, per effettuare operazioni sulle mappe

Dependency Injection

- Il funzionamento di tutto il framework
  - è pesantemente basato sul sistema di manipolazione dei “managed bean”
  - che consente anche forme di “dependency injection”
- Dipendenza tra componenti
  - descrive il fatto che un componente ha bisogno di conoscerne un altro per poter funzionare

Dependency Injection

- Un esempio tipico
  - le azioni hanno normalmente bisogno dei bean del modello per poter compiere il loro lavoro
  - in una tipica azione Web, i riferimenti ai bean vengono acquisiti dalle mappe condivise (richiesta, sessione, applicazione)
  - e poi vengono chiamati i metodi di logica applicativa

Dependency Injection

- Il sistema di gestione dei managed bean
  - ⇨consente di dichiarare una serie di JavaBean nel file faces-config.xml
  - ⇨attribuendogli un id e uno scope
  - ⇨il framework inizializza all'occorrenza il bean e lo salva nella mappa corrispondente allo scope
  - ⇨successivamente, lo recupera dalla mappa se già esiste

Dependency Injection

- Inoltre
  - ⇨consente anche di descrivere e manipolare le proprietà dei bean
  - ⇨es: inizializzare una proprietà
  - ⇨es: utilizzare una proprietà per il binding
- In effetti
  - ⇨consente di specificare che alcune proprietà corrispondono ad associazioni tra bean gestiti

Dependency Injection

>> azioneIniziaPartita
>> azioneGestisciTentativo

- In questo caso
  - ⇨il framework si preoccupa di istanziare correttamente tutti i riferimenti
- Esempio
  - ⇨per soddisfare le dipendenze all'interno di un'azione, posso pensare di dichiarare i bean del modello necessari come proprietà
  - ⇨e lasciare che il framework ne istanzi i valori quando è necessario eseguire l'azione

Dependency Injection

- Un modo per vedere la cosa
  - ⇨il framework sta "iniettando" nell'oggetto di tipo azione i valori necessari a soddisfare le dipendenze
  - ⇨in questo caso chiamando i metodi set corrispondenti
  - ⇨a quel punto le dipendenze sono soddisfatte e l'azione può procedere nella sua esecuzione

Dependency Injection

- Altri tipici casi di dependency injection
  - ⇨servizi di utilizzo frequente come il logger
  - ⇨servizi collegati alla persistenza, come la DataSource oppure i DAO
- In generale
  - ⇨il meccanismo di dependency injection consente di rimuovere molti singleton e molte variabili globali

Dependency Injection

>> Giocatori.java

- Attenzione, però
  - ⇨per utilizzare il meccanismo di dependency injection, i componenti devono essere JavaBeans
  - ⇨di conseguenza, per esempio, non è possibile utilizzare DAO statici
  - ⇨nè classi wrapper, come Integer (a questo scopo nell'applicazione è stato sviluppato un bean apposito per il numero di giocatori)

Dependency Injection

- Di conseguenza
  - ⇨cambia molto lo stile di scrittura delle azioni
- Una azione tipica per il modello 2
  - ⇨acquisisce i valori della query string
  - ⇨effettua le convalide
  - ⇨acquisisce i riferimenti ai bean del modello
  - ⇨aggiorna i valori del modello
  - ⇨chiama i metodi di logica applicativa e decide lo schermo successivo

Dependency Injection

- Con JSF
  - ⇨tutte le prime attività sono gestite dal framework
- Resta solo l'ultimo passo
  - ⇨eseguire i metodi della logica applicativa e decidere lo schermo successivo
  - ⇨quindi – in aggiunta alla mancanza dei formBean – le azioni sono molto più snelle

Dependency Injection

>> indovinam2jsf

- Un esempio
  - ⇨indovinaIlNumero con JSF
- Attenzione alla differenza di filosofia
  - ⇨le azioni sono scritte per la gestione di eventi e non per la gestione esplicita di richieste HTTP
  - ⇨viene utilizzato pesantemente il sistema di binding

Aspetti Sistemistici

- La costruzione dell'applicazione
  - ⇨richiede l'utilizzo di vari jar
- Per la reference implementation
  - ⇨jsf-api.jar
  - ⇨jsf-impl.jar
  - ⇨jstl.jar
  - ⇨standard.jar
  - ⇨commons-beanutils.jar
  - ⇨commons-collections.jar
  - ⇨commons-digester.jar
  - ⇨commons-logging.jar

Aspetti Sistemistici

- Per myFaces
  - ⇨ myfaces.jar
  - ⇨ jstl.jar
  - ⇨ standard.jar
  - ⇨ jakarta-oro.jar
  - ⇨ commons-beanutils-1.6.1.jar
  - ⇨ commons-codec-1.2.jar
  - ⇨ commons-collections-3.0.jar
  - ⇨ commons-digester-1.5.jar
  - ⇨ commons-el.jar
  - ⇨ commons-fileupload-1.0.jar
  - ⇨ commons-logging.jar
  - ⇨ commons-validator.jar

Aspetti Sistemistici

- Nota
  - ⇨nei file di configurazione delle due implementazioni possono esserci varianti
  - ⇨es: parametri di configurazione diversi per il controllore frontale
  - ⇨è opportuno consultare gli esempi forniti a corredo delle distribuzioni per prelevare modelli di file di configurazione corretti (faces-config.xml e web.xml)

Aspetti Sistemistici

>> jsf-template-build.xml

- E' opportuno
  - ⇨utilizzare Ant per la costruzione del codice
- jsf-template-build
  - ⇨un modello di file di Ant per la costruzione di progetti JSF

Aspetti Sistemistici

- Attenzione
  - ⇨in un'applicazione JSF è opportuno impedire l'accesso diretto alle pagine JSP
- Una possibile soluzione
  - ⇨utilizzare le funzionalità di sicurezza integrate nel contenitore
  - ⇨in particolare, l'elemento security constraints di web.xml

Aspetti Sistemistici

- Security constraints
  - ⇨consente di specificare quali utenti sono autorizzati ad accedere ad un gruppo di ris.
  - ⇨autorizzazione gestita dal contenitore

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Pagine JSP</web-resource-name>
        <url-pattern>/primoSchermo.jsp</url-pattern>
        ...
    </web-resource-collection>
    <auth-constraint></auth-constraint>
</security-constraint>
```


Aspetti Sistemistici

- Ma...
  - ⇨questo risolve solo parzialmente il problema della protezione
- Come effettuare la verifica della richiesta?
  - ⇨in un'applicazione modello 2 tutte le richieste generano comandi e sono filtrate dalle azioni
  - ⇨in un'applicazione JSF un'azione viene chiamata solo a seguito di un evento

Aspetti Sistemistici

- Esempio
  - anche proteggendo le pagine JSP, un utente può effettuare una richiesta a schermoIndovinato.jsf
  - in questo caso, al di fuori della sessione, verrebbe considerata come una richiesta iniziale e lo schermo sarebbe reso
  - successivamente sarebbe possibile scatenare eventi

Aspetti Sistemistici

- Soluzioni a questo problema
  - varie, alcune complicate
- Soluzioni complicate
  - catturare l'evento associato alla fase di resa della risposta e impedire accessi scorretti
  - sviluppare un decoratore per il "navigation handler" di JSF che applica le regole di navigazione

Aspetti Sistemistici

>> FiltroSessione.java
>> faces-config.xml

- La soluzione più semplice
  - ⇨sviluppare un filtro per la verifica delle richieste
  - ⇨il filtro verifica che, per ogni richiesta che NON sia indirizzata allo schermo iniziale (schermoNome.jsf), ci sia una sessione in piedi e nella sessione una partita in corso
- Nota
  - ⇨la stessa tecnica sarebbe valida per la verifica dell'autorizzazione in un'applicazione protetta da login

Riassumendo

- Introduzione
- Componenti
- La Filosofia di Programmazione
- Programmazione MVC con JSF
- Convalida e Binding
- Il Ciclo di Vita della Richiesta
- Dependency Injection
- Aspetti Sistemistici

Termini della Licenza