Java J2EE

Il framework Apache *Struts*

Relatore
Dott.ssa Lucrezia Macchia
Ph.D Student in Computer Science

Dipartimento DI
INFORMATICA

Outiline

- Architettura Model-View-Controller
- Le componenti Struts
- Le Action
- Le Taglibs
- Il framework Validator
- Internazionalizzazione

Struts e MVC

- Apache Struts è un progetto open source sponsorizzato dalla Apache Software Foundation ed è un'implementazione Java server-side del design pattern MVC (**Model View Controller**).
- L'MVC è un pattern architetturale diffuso nello sviluppo di interfacce grafiche di sistemi software object-oriented.
- Il pattern è stato esplicitamente o implicitamente sposato da numerose tecnologie moderne, come framework basati su PHP, su Ruby, su Python, su Java (Swing, JSF e Struts), su Objective C o su .NET.

Struts e MVC

- Il progetto Struts è nato con l'intenzione di implementare un framework open-source per la creazione di applicazioni Web che permettesse la separazione del *livello di presentazione* e che fosse, allo stesso tempo, astratto dai vari livelli di dati e dalle transazioni.
- Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:
- il **model** fornisce i metodi per accedere ai dati utili all'applicazione;
- il **view** visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
- il **controller** riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti
- Questo schema, fra l'altro, implica anche la tradizionale separazione fra la logica applicativa (in questo contesto spesso chiamata "logica di business"), a carico del *controller* e del *model*, e l'interfaccia utente a carico del *view*.

- Model

- implementa la logica applicativa

- insieme di classi java (Tipicamente Bean)

- presenza di uno o più packages responsabili

- View

- Insieme di Pagine JSP costruite mediante l'ausilio di particolari tag offerte da Struts2

- Controller

- classi Actions

Le classi Actions rispecchiano una struttura flessibile e possono estendere o implementare elementi messi a servizio dal framework

- sulla base delle richieste dell'utente, decide quale action eseguire per interagire con il modello.

Queste proprietà sono elencate e gestibili dal file Struts.xml

- sulla base del risultato ritornato dalla Action decide a quale pagina JSP affidare la gestione della risposta, funzionalità anche questa offerta e personalizzabile dal file Struts.xml

Perché Struts?

L'utilizzo di Struts supporta vantaggi significativi in termini del progetto:

- **Modularità e Riusabilità**: i diversi ruoli dell'applicazione sono affidati a diversi componenti. Ciò consente di sviluppare codice modulare e più facilmente riutilizzabile
- **Manutenibilità**: l'applicazione è costituita da livelli logici ben distinti. Una modifica in uno dei livelli non comporta modifiche negli altri
- **Rapidità di sviluppo**: è possibile sviluppare in parallelo le varie parti dell'applicazione, logica di business e view

- **Progettazione semplificata** - La maggior parte delle classi Struts sono basate su interfacce. Le classi di Struts 2 sono indipendenti e sono semplificate
- **Azioni semplificate** - Tutte le classi Java con metodo execute () possono essere utilizzate come una classe ACTION.
- **Predefinite intelligenti** - elementi di configurazione hanno un valore predefinito che può essere impostato in base alla necessità. È possibile configurarli nel file XML di configurazione e possono essere sovrascritti a seconda delle necessità.
- **Risultati migliori** - A differenza ActionForwards, le risposte Struts 2 forniscono la flessibilità per creare risposte multiple

- **Caratteristiche Tag Better** - i tag di Struts 2 consentono di aggiungere fogli di stile, in modo da poter creare pagine coerenti con meno codice. I tag possono essere modificati modificando il foglio di stile di base.
- **Annotazioni:** sono state introdotte le annotazioni Java 5 come alternativa a XML per la configurazione delle proprietà. Le annotazioni servono per minimizzare l'uso di xml.
- **QuickStart** - Molti cambiamenti possono essere fatte al volo senza riavviare un contenitore web.
- **personalizzazione controller** - Struts 1 permette di personalizzare la richiesta processata per ogni modulo, Struts 2 consente di personalizzare la gestione delle richieste per azione, se lo si desidera.
- **Facile integrazione di Spring**
- **Plugin Facile** - estensioni di Struts 2 possono essere aggiunte inserendo il Jar. Nessuna configurazione manuale è necessaria!
- **Supporto AJAX**

Il design pattern MVC

Componente	Descrizione
Model	Rappresenta i dati, tipicamente persistenti su database, attraverso oggetti, questa rappresentazione ci permette di manipolare e il modello dei dati in modo semplificato
View	È lo strato più esterno, quello di presentazione. È qui che definiamo l'interfaccia utente, forniamo una rappresentazione del modello dei dati e riceviamo richieste dall'esterno.
Controller	È il componente che contiene la logica di business. Qui gestiamo le interazioni con l'interfaccia, istradiamo le richieste, preleviamo i dati dal Model e stabiliamo quale View dovrà rappresentarli e come.

Una richiesta client viene intercettata dal **Controller**, attraverso il **Model** vengono forniti tutti i metodi per accedere ai dati dell'applicazione e quindi per elaborare la risposta e visualizzarla attraverso i componenti **View**.

File di Configurazione

Per un corretto funzionamento del framework vi è la necessità di configurare correttamente due file xml :

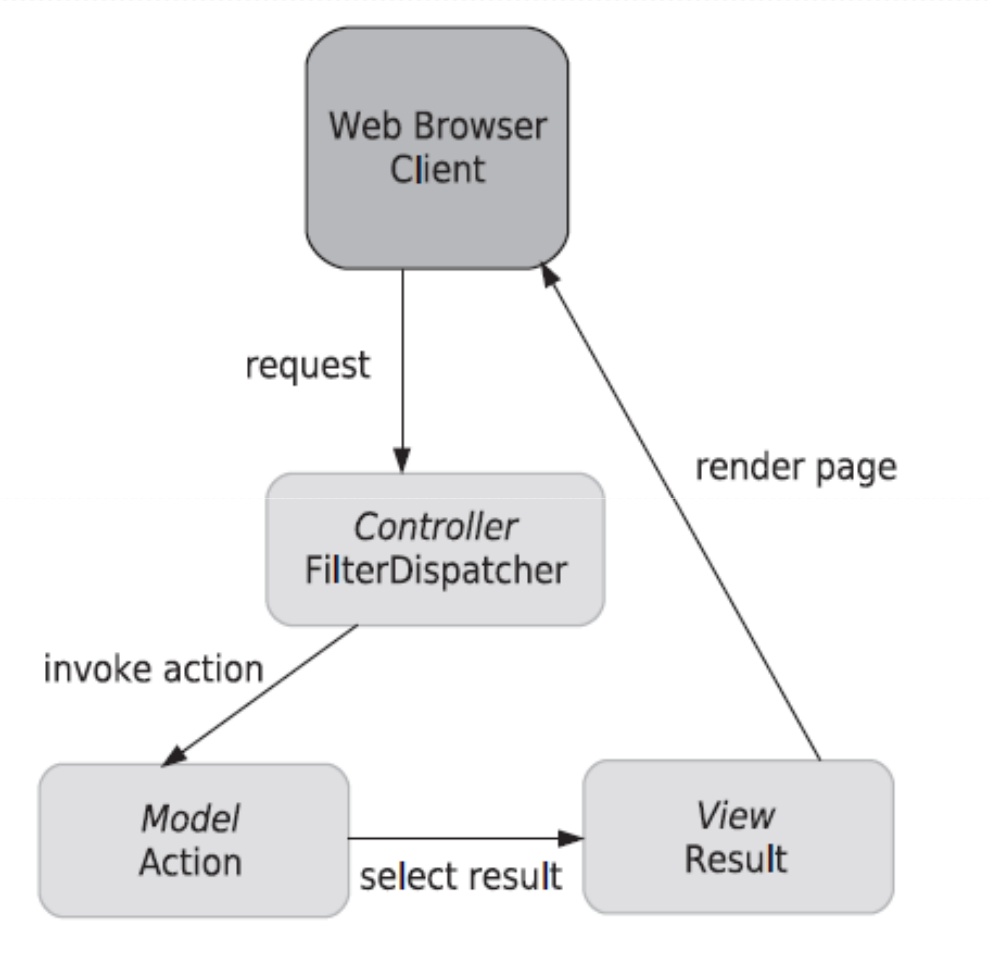
- struts.xml (file di configurazione proprio di Struts2 e da inserire nel source folder del proprio codice sorgente)
- web.xml (file di configurazione fondamentale per ogni applicazione web,che dovrà essere configurato correttamente affinchè il contenitore web specifico permetta il corretto funzionamento del framework)

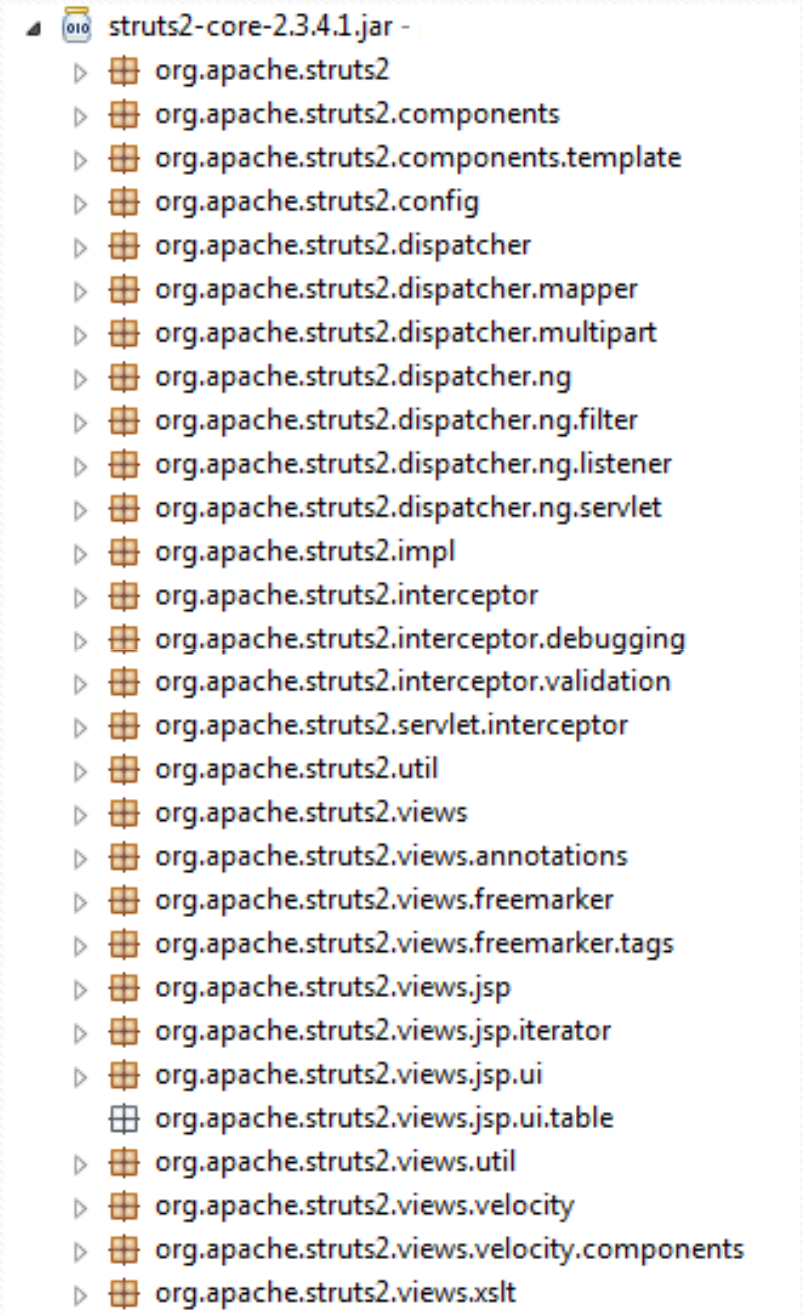
Web.xml

All'interno di questo file è opportuno impostare come filtro per la gestione delle richieste Struts2.

- Il framework possiede infatti una classe che si comporterà da controller primario : Il FilterDispatcher
- Una volta creato il filtro sarà doveroso associarlo ad ogni url che il file web.xml dovrà gestire.

- # Codice di esempio :
- <web-app id="WebApp_9" version="2.4" xmlns=http://java.sun.com/xml/ns/j2ee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd“>
- <display-name>Esempio</display-name>
- <filter>
  - <filter-name>struts2</filter-name>
  - <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
- </filter>
- <filter-mapping>
  - <filter-name>struts2</filter-name>
  - <url-pattern>/*</url-pattern>
- </filter-mapping>
- <welcome-file-list>
  - <welcome-file>index.jsp</welcome-file>
- </welcome-file-list>
- </web-app>


Web Browser
Client
request
render page
Controller
FilterDispatcher
invoke action
Model
Action
select result
View
Result


struts2-core-2.3.4.1.jar -
org.apache.struts2
org.apache.struts2.components
org.apache.struts2.components.template
org.apache.struts2.config
org.apache.struts2.dispatcher
org.apache.struts2.dispatcher.mapper
org.apache.struts2.dispatcher.multipart
org.apache.struts2.dispatcher.ng
org.apache.struts2.dispatcher.ng.filter
org.apache.struts2.dispatcher.ng.listener
org.apache.struts2.dispatcher.ng.servlet
org.apache.struts2.impl
org.apache.struts2.interceptor
org.apache.struts2.interceptor.debugging
org.apache.struts2.interceptor.validation
org.apache.struts2.servlet.interceptor
org.apache.struts2.util
org.apache.struts2.views
org.apache.struts2.views.annotations
org.apache.struts2.views.freemarker
org.apache.struts2.views.freemarker.tags
org.apache.struts2.views.jsp
org.apache.struts2.views.jsp.iterator
org.apache.struts2.views.jsp.ui
org.apache.struts2.views.jsp.ui.table
org.apache.struts2.views.util
org.apache.struts2.views.velocity
org.apache.struts2.views.velocity.components
org.apache.struts2.views.xslt

Struttura

- Struts, come ogni application framework, è un insieme di classi e interfacce che costituiscono lo scheletro per costruire Web application.

Componente	Descrizione
ActionServlet	È la servlet di controllo che gestisce tutte le richieste dell'applicazione. Come tutte le servlet estende la classe javax.servlet.http.HttppServlet e quindi implementa tutti i metodi di lifecycle, incluso init(), doGet(), doPost() ed il destroy
struts.xml	È il cuore di tutta l'applicazione. In questo file XML possiamo definire i vari elementi dell'applicazione e le loro associazioni. Viene letto in fase di start-up dell'applicazione dalla ActionServlet
Action	Le Action sono le classi alle quali le ActionServlet delega l'elaborazione della richiesta
Custom-tags	Sono tag particolari forniti dal framework Struts per assolvere a molti dei più comuni compiti delle pagine JSP

Il ciclo di vita delle richieste

- Il controller ha la responsabilità di ricevere l'input da un client, invocare le operazioni necessarie alla logica applicativa e coordinare la vista da restituire al client. In altre parole contiene tutte la logica di business.
- Il componente Model fornisce gli oggetti necessari alla logica di business per astrarre la persistenza dei dati.
- Infine le view rappresentano il modo di interagire dell'applicazione con l'utente sia in fase di richiesta che in fase di risposta.

Il Model-View-Controller modello in Struts2 è realizzato con i seguenti cinque componenti fondamentali:

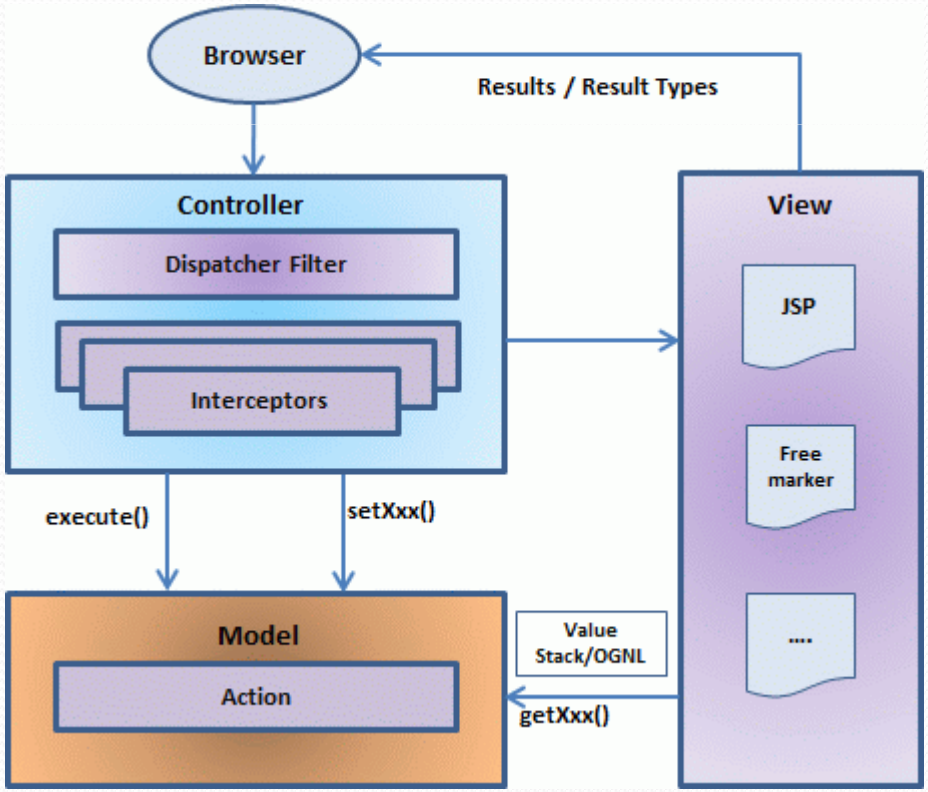
•Azioni

•intercettori

•Valore Stack

•Risultati

•tecnologie



- In ciclo di vita con Struts 2:

- Utente invia una richiesta al server per la richiesta di una risorsa (pagine).
- Il FilterDispatcher esamina la richiesta e determina l'azione appropriata.
- Si eseguono ulteriori funzionalità come intercettori come: convalida, caricamento ecc.
- L'azione selezionata viene eseguita per l'operazione richiesta.
- Anche in questo caso, gli intercettori configurati sono applicati a qualsiasi post-elaborazione, se necessario.
- Infine il risultato viene preparato alla visualizzazione e si restituisce il risultato all'utente.

Il file di configurazione «struts.xml»

- Per caricare e creare all'avvio tutti i componenti necessari all'applicazione, Struts fa riferimento ad un file di configurazione chiamato **struts.xml**. Questo file ci permette di specificare in maniera dichiarativa il comportamento dei componenti del framework evitando che le informazioni e il comportamento siano inseriti rigidamente nel codice delle applicazioni.

- Questo fornisce agli sviluppatori la flessibilità necessaria ad inserire le proprie estensioni che il framework può utilizzare in maniera dinamica.

- Lo struts si basa sul formato XML e può essere validato con il DTD di Struts

Struttura del file «struts.xml»

- Il tag root dello struts-config è <struts> ed è composto da cinque sezioni:
  - **global-forwards**
  - **form-beans**
  - **action-mappings**
  - **message-resources**
  - **plug-in**

global-forwards

- In questa sezione possiamo creare delle associazioni tra particolari nomi (che specificano azioni del controller) e i relativi percorsi (che specificano delle viste), stabilendo dei forward validi a livello "**globale**" nell'applicazione (ad esempio la gestione degli error)

```
<global-forwards>
  <forward name="error" path="/error.jsp"/>
</global-forwards>
```

- Di seguito l'elenco degli attributi più importanti

Attributo	**Descrizione**
className	Serve a dichiarare la classe che estende il bean di configurazione e che manterrà tutte le informazioni di forward. Se non specificata, la classe predefinita sarà org.apache.struts.action.ForwardConfig
Name	È il nome (unico) che servirà a riferirsi a questo forward nell'applicazione. Questo attributo è *obbligatorio*
path	È l'URI verso cui dovrebbe avvenire il forward. È un attributo *obbligatorio* e deve cominciare con il carattere "/"

form-beans

- La seconda parte serve a **definire i form bean**, particolari classi che contengono i dati inseriti in un form all'interno di una Jsp. Si dichiara uno o più form bean nel seguente modo:

```
<form-beans>
  <form-bean name="..." type="..." />
</form-beans>
```

- Di seguito l'elenco degli attributi più importanti

Attributo	**Descrizione**
className	Quando non si vuole utilizzare il bean di configurazione standard di Struts, bisogna specificare in questo attributo la classe creata nell'applicazione che la sostituisce
Name	È il nome (unico) che servirà a riferirsi a questo form bean in tutta l'applicazione. Questo attributo è *obbligatorio*
Type	Il nome di una classe Java che estende la classe ActionForm di Struts

action-mappings

- In questa sezione definiamo le action. Per ogni azione inseriamo un elemento <action> e ne specifichiamo sia le caratteristiche (grazie alle proprietà come path, name, parameters, etc.) sia i forward dell'azione stessa (grazie ad elementi <forward>)

```
<action-mappings>
  <action path="/..." name="..." scope="..." type="..."
          parameter="..." validate="...">

    <forward name="..." path="/..." />
  </action>
</action-mappings>
```

- Ecco un **esempio di action mapping**, in cui associamo il path "/azione" alla classe it.html.struts.MyAction, indichiamo che utilizzeremo il form myForm e che per gli errori ci serviamo della vista dataerror.jsp:

```
<action path = "/azione"
        type = "it.html.struts.MyAction"
        name = "myForm"
        input = "/WEB-INF/jsp/dataError.jsp">

  <forward name="OK" path="/WEB-INF/jsp/viewResult.jsp"/>
  <forward name="ERROR" path="/WEB-INF/jsp/busError.jsp"/>
</action>
```

message-resources

- I message-resources sono classi utili a gestire i messaggi in modo unificato. Sono particolarmente utili per applicazioni *multilingua*.

```
<message-resources parameter="MessageResources" />
```

- Nelle view si farà riferimento a delle chiavi (key) che saranno poi associate ai messaggi corrispondenti.

plug-in

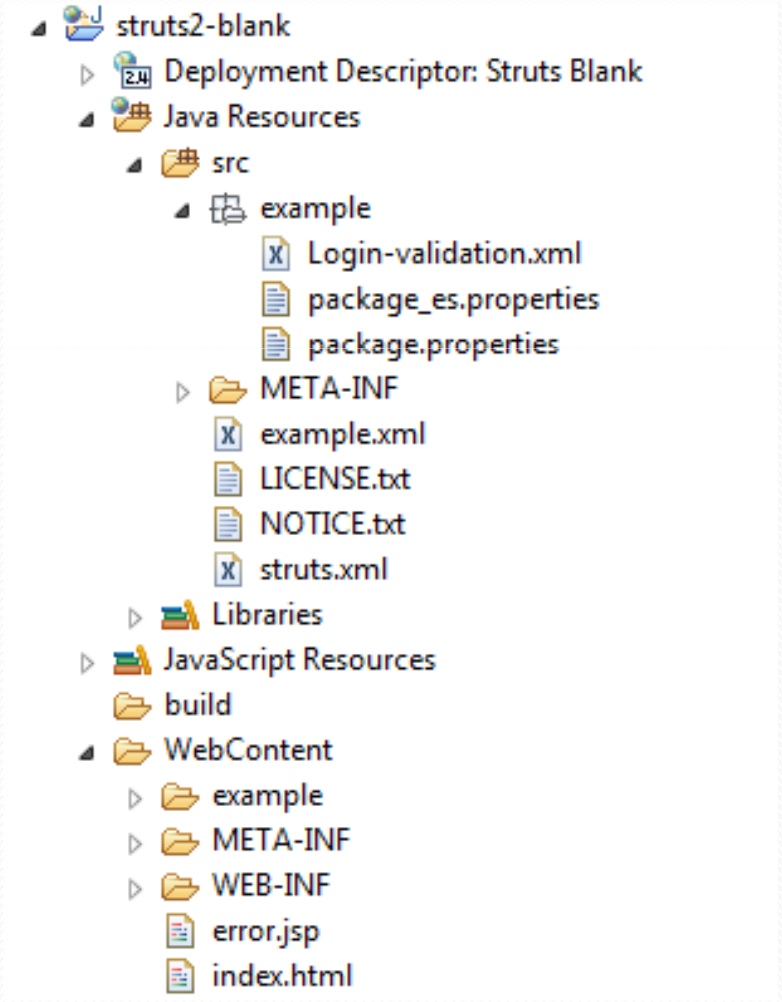
- Infine abbiamo la parte dedicate alla dichiarazione dei plugin usati nell'applicazione.
- Osserviamo come aggiungere all' applicazione uno dei plugin più utilizzati del framework, il **validator** che permette di validare i parametri inseriti in un form:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">

  <set-property property="pathnames"
                value="/org/apache/struts/validator/validator-rules.xml,
                       /WEB-INF/validation.xml"/>
</plug-in>
```

File proprieties

- Tutte le proprietà devono essere impostate nel file .properties!!!!



Le action

- Le Action sono gli strumenti grazie ai quali il Controller di Struts gestisce le attività.
- Ogni Action rappresenta una funzione dell'applicazione, quindi è qui che scriviamo la logica applicativa dei nostri progetti.
- La classe Action disaccoppia le richieste del client dall'applicazione.
- Ogni Action deve essere dichiarata e configurata nel file struts.xml e non nel web.xml.

Le action

Le Action servono ad esempio a ricevere le richieste dai form, ad elaborare i dati e a lanciare le View per la visualizzazione delle informazioni.

Per **realizzare le Action**, è necessario:

- **Creare una classe** che estenda com.opensymphony.xwork2.ActionSupport
- **Implementare il metodo execute()** aggiungendo la logica di business della nostra applicazione
- **Aggiungere un elemento <action> al file struts** all'applicazione che descrive la nuova azione

Creare una Action

- Per dichiarare una nuova Action è sufficiente implementare la classe base ed importare tutti i namespace necessari

```
public interface Action
{
	public static final String SUCCESS = "success";
	public static final String NONE = "none";
	public static final String ERROR = "error";
	public static final String INPUT = "input";
	public static final String LOGIN = "login";

	public String execute() throws Exception;
}
```

- Oppure si può estende una sua specializzazione con l'uso della classe ActionSupport

```
import com.opensymphony.xwork2.ActionSupport;
public class HelloWorldAction extends ActionSupport
{
    private String name;

    public String execute() throws Exception {
        if ("SECRET".equals(name))
            { return SUCCESS; }
        else{ return ERROR; }
    }

    public String getName() { return name; }

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

Il metodo execute()

Con la classe base Action possiamo effettuare l'override del metodo execute() riscrivendo le operazioni che vogliamo far compiere alla nostra azione. E' qui che inseriamo la "logica di business".

- Il metodo execute() viene invocato dal controller quando viene ricevuta una richiesta. Inoltre **una classe Action viene istanziata una sola volta, all'avvio dell'applicazione**, quindi occorre garantire che tutte le Action operino correttamente in un ambiente multithread, proprio come si fa quando si sviluppa una *servlet*.

Le funzioni principali di execute() sono:

- compiere la logica dell'applicazione
- instradare la richiesta indicando al Framework il passo successivo da eseguire

Configurazione della classe Action

- **Le classi Action si configurano nel file struts.xml**, poiché si tratta di oggetti specifici di Struts.
- L'elemento che viene utilizzato per descrivere un'azione Struts è **<action>**. e la classe che definisce gli attributi dell'elemento <action> è **org.apache.struts.action.ActionMapping**.

```
<struts>
   <constant name="struts.devMode" value="true" />
   <package name="helloworld" extends="struts-default">
         <action name="hello" class="struts2.HelloWorldAction" method="execute">
            <result name="SUCCESS">/HelloWorld.jsp</result>
            <result name="ERROE">/AccessDenied.jsp</result>
         </action>
   </package>
</struts>
```

Con Struts 2

- È possibile creare un insieme di azioni semplicemente dichiarando un package all'interno del file xml

- <package name="html" extends="struts-default">
  <action name="HtmlAction" class="action.HtmlAction“ method=“execute”>
    - <result name="giorno">/jsp/giorno.jsp</result>
    - <result name="notte">/jsp/notte.jsp</result>

  </action>
  <action name="HtmlAction2" class="action.HtmlAction“ method=“aggiorna”>
    - <result name=“sera">/jsp/sera.jsp</result>
    - <result name=“indefinito">/jsp/error.jsp</result>

  </action>
  ….

  </package>

Creare una semplice web

web.xml Servlet Filter

```
<filter>
  <filter-name>struts2</filter-name>
  <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>struts2</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

struts.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

        <constant name="struts.devMode" value="true" />

        <package name="basicstruts2" extends="struts-default">

                <action name="index">
                        <result>/index.jsp</result>
                </action>

        </package>

</struts>
```

- Quando si fa clic su un collegamento ipertestuale o si invia un modulo HTML in una applicazione web Struts 2, l'ingresso non viene inviato a un'altra pagina, ma a una classe Java. Queste classi sono chiamate Azioni.

- Dopo viene selezionata una risorsa per inviare la risposta. La risorsa è generalmente una pagina del server, ma può anche essere un file PDF, un foglio di calcolo di Excel, o una finestra di applet Java.

- Si supponga di voler creare un semplice esempio "Ciao Mondo" che visualizza un messaggio di benvenuto.

- Dopo la creazione di un progetto vuoto, per creare un'applicazione "Ciao Mondo", è necessario fare quattro cose:
  - Creare una classe per memorizzare il messaggio di benvenuto (il modello)
  - Creare una pagina server per presentare il messaggio (la vista)
  - Creare una classe Action per controllare l'interazione tra l'utente, il modello, e la vista (controllore)
  - Creare una mappatura (struts.xml) per associare l'azione di classe e vista

```
package org.apache.struts.helloworld.model;

public class MessageStore {

        private String message;

        public MessageStore() {

                setMessage("Hello Struts User");
        }

        public String getMessage() {

                return message;
        }

        public void setMessage(String message) {

                this.message = message;
        }

}
```

- Abbiamo bisogno di una classe Action che agisca come controller. La classe Action risponde a un'azione dell'utente. Uno o più dei metodi della classe d'azione vengono eseguiti e viene restituita una stringa. In base al valore del risultato, è resa una “vista”(pagina) specifica

```
package org.apache.struts.helloworld.action;

import org.apache.struts.helloworld.model.MessageStore;
import com.opensymphony.xwork2.ActionSupport;

public class HelloWorldAction extends ActionSupport {

        private static final long serialVersionUID = 1L;

        private MessageStore messageStore;

        public String execute() throws Exception {

                messageStore = new MessageStore() ;
                return SUCCESS;
        }

        public MessageStore getMessageStore() {
                return messageStore;
        }

        public void setMessageStore(MessageStore messageStore) {
                this.messageStore = messageStore;
        }

}
```

- Verrà creato un oggetto della classe HelloWorldAction e chiamato il metodo execute() in risposta all'azione di un utente (cliccando su un collegamento ipertestuale che invia un URL specifico per il Servlet Container).

- In questo esempio, il metodo execute() crea un oggetto di classe MessageStore e quindi restituisce la String SUCCESS

- Dal momento che vogliamo rendere l'oggetto MessageStore disponibile alla pagina di visualizzazione (HelloWorld.jsp) abbiamo bisogno di fornire metodi get e set.

HelloWorld.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hello World!</title>
</head>
<body>
    <h2><s:property value="messageStore.message" /></h2>
</body>
</html>
```

- La direttiva taglib dice al Servlet Container che questa pagina verrà utilizzato i tag Struts e che questi tag sanno preceduti da s.
- La tag s:proprerty visualizza il valore restituito dalla chiamata al metodo getMessageStore della classe controller HelloWorldAction. Questo metodo restituisce un oggetto MessageStore. Il metodo getMessage della classe MessageStore restituisce una stringa.

- Occorre un mapping per legare l'URL, la classe HelloWorldAction (controller), e il HelloWorld.jsp (la vista) insieme.
- Il mapping indica al framework Struts 2 che classe risponderà alle azioni dell'utente (l'URL), il metodo di quella classe verrà eseguito, e cosa si vedrà in base al risultato che restituisce il metodo.
- Modificare il file struts.xml per aggiungere il mapping dell'Action.
- struts.xml completo dovrebbe essere simile a:

struts.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

  <constant name="struts.devMode" value="true" />

  <package name="basicstruts2" extends="struts-default">

  <action name="index">
    <result>/index.jsp</result>
  </action>

  <action name="hello" class="org.apache.struts.helloworld.action.HelloWorldAction" method="execute">
    <result name="success">/HelloWorld.jsp</result>
  </action>

</package>

</struts>
```

- In index.jsp si aggiunge un URL Action che ci permette di eseguire il metodo execute() della classe HelloWorldAction e visualizzarà la HelloWorld.jsp.

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Basic Struts 2 Application - Welcome</title>
</head>
<body>
<h1>Welcome To Struts 2!</h1>
<p><a href="<s:url action='hello'/>">Hello World</a></p>
</body>
</html>
```

- Codifica in Struts 2 di una Action coinvolge diverse parti:

- 1. Mapping di un ricorso a una classe
- 2. Mapping di un risultato a una vista
- 3. Scrivere la logica di controllo nella classe Action

Action Mapping

```
<action name="hello" class="org.apache.struts.helloworld.action.HelloWorldAction" method="execute">
        <result name="success">/HelloWorld.jsp</result>
</action>
```

Il mapping di Action sopra specifica che se il metodo execute della classe HelloWorldAction restituisce success allora HelloWorld.jsp verrà restituito al browser.

- Classi Action agiscono come controller del pattern MVC. Classi Action rispondono a un'azione dell'utente, eseguono la logica di business (o chiamano altre classi per farlo), e poi restituiscono un risultato
- Uno dei compiti più comuni della classe Action è quello di elaborare l'input dell'utente in un modulo e poi visualizzare il risultato della elaborazione nella pagina di visualizzazione.
- Sulla pagina di visualizzazione, HelloWorld.jsp, si vuole visualizzare ad esempio "Ciao Struts utente Bruce.“
- Nel Utilizzando Struts 2 i Tag è stato aggiunto un form per Struts 2 in index.jsp.

```
<s:form action="hello">
        <s:textfield name="userName" label="Your name" />
        <s:submit value="Submit" />
</s:form>
```

- Quando l'utente fa clic sul pulsante di invio per il modulo di cui sopra, verrà eseguita l'azione (hello.action).
- I valori del campo sarà inviato alla classe Action (HelloWorldAction).

```
        private String userName;

        public String getUserName() {
                return userName;
        }

        public void setUserName(String userName) {
                this.userName = userName;
        }
```

- Per personalizzare il messaggio MessageStore (ricordiamo che la classe MessageStore memorizza il messaggio da visualizzare)

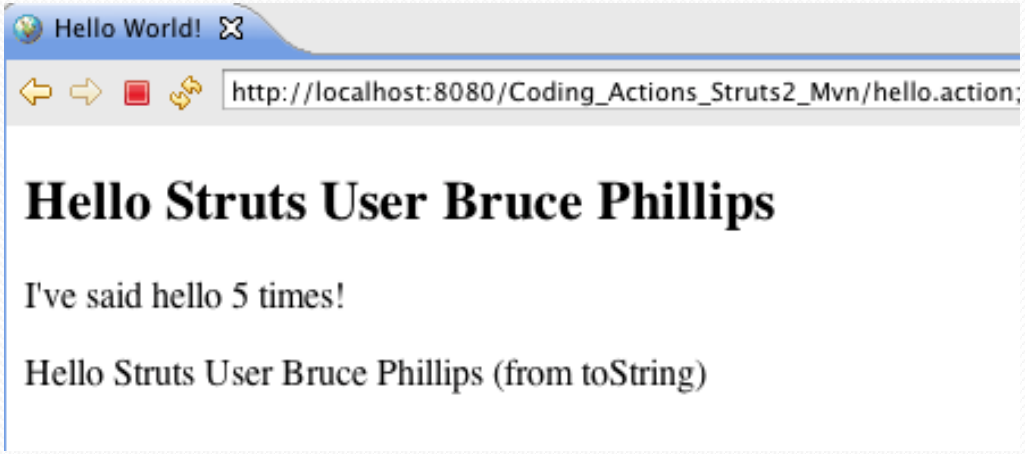
```
                                                                Add userName value to message

if (userName != null) {

        messageStore.setMessage( messageStore.getMessage() + " " + userName);

}
```


Hello World!
http://localhost:8080/Coding_Actions_Struts2_Mvn/hello.action
Hello Struts User Bruce
I've said hello 4 times!
Hello Struts User Bruce (from toString)


Hello World!
http://localhost:8080/Coding_Actions_Struts2_Mvn/hello.action
Hello Struts User Bruce Phillips
I've said hello 5 times!
Hello Struts User Bruce Phillips (from toString)

Struts 2 url Tag

- Un caso d'uso molto comune nelle applicazioni web è il collegamento ad altre pagine.
- In Ciao Mondo abbiamo aggiunto alla index.jsp un link al hello.action utilizzando il tag url Struts.

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Basic Struts 2 Application - Welcome</title>
</head>
<body>
<h1>Welcome To Struts 2!</h1>
<p><a href="<s:url action='hello'/>">Hello World</a></p>
</body>
</html>
```

- Quando si esegue Ciao Mondo nel servlet container e poi click sul collegamento ipertestuale Ciao Mondo creato dal tag URL Struts, l'URL creato è hello.action (relativo alla cartella principale del contesto web).

struts.xml

```
...
<action name="hello" class="org.apache.struts.helloworld.action.HelloWorldAction" method="execute">
  <result name="success">/HelloWorld.jsp</result>
</action>
...
```

Se il metodo execute restituisce success, HelloWorld.jsp (nella cartella root di contesto web) verrà restituito all'utente.

- Un caso d'uso comune è che l'URL deve anche includere un valore per un parametro di stringa di query come nome utente. Per aggiungere un parametro di stringa di query e il suo valore d'uso nel tag param di Struts, nidificato all'interno del tag url.

```
<s:url action="hello" var="helloLink">
  <s:param name="userName">Bruce Phillips</s:param>
</s:url>

<p><a href="${helloLink}">Hello Bruce Phillips</a></p>
```

Si noti l'uso dell'attributo var. Il valore dell'attributo var è un riferimento che possiamo usare più avanti nel codice per indicare l'URL creato.

Si noti il valore dell'attributo href è ${helloLink}. La pagina di visualizzazione sostituirà il collegamento ipertestuale che abbiamo creato utilizzando il tag url per il riferimento ${helloLink}.

Struts 2 Form Tags

- La maggior parte delle applicazioni utilizzeranno diversi moduli di immissione dati. Le tag di Struts rendono la facile la creazione di form.
- Ognuna delle tag form Struts ha numerosi attributi per imitare i normali attributi del tag HTML form.
- Per creare il contenuto esterno del form,si usa il modulo del tag Struts. L'attributo action imposta il nome dell'azione

Struts

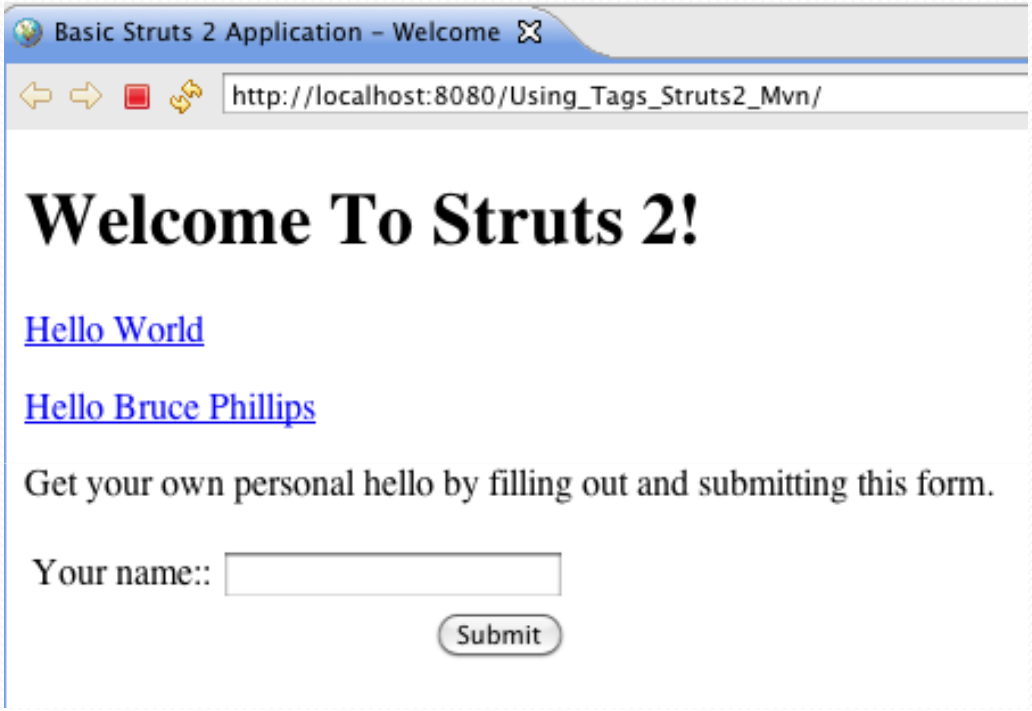
```
<p>Get your own personal hello by filling out and submitting this form.</p>

<s:form action="hello">

  <s:textfield name="userName" label="Your name" />

   <s:submit value="Submit" />

</s:form>
```


Basic Struts 2 Application - Welcome
http://localhost:8080/Using_Tags_Struts2_Mvn/
Welcome To Struts 2!
Hello World
Hello Bruce Phillips
Get your own personal hello by filling out and submitting this form.
Your name::
Submit

```
<s:select key="personBean.sport" list="sports" />
```

```
<s:radio key="personBean.gender" list="genders" />
```

```
<s:select key="personBean.residency" list="states" listKey="stateAbbr" listValue="stateName" />
```

```
<s:checkbox key="personBean.over21" />
```

```
<s:checkboxlist key="personBean.carModels" list="carModelsAvailable" />
```

Message Resource Files

- Struts 2 permette la configurazione di message resource file.

- Questi messaggi forniscono un modo semplice per mettere il testo in una pagina di visualizzazione (il view) che è lo stesso durante tutto l'applicazione,
- per creare le etichette dei campi del modulo, e per modificare il testo in un linguaggio specifico in base alle impostazioni internazionali dell'utente (i18n).

- In una web application Struts 2 è possibile associare una message resources file con ogni classe Action con la creazione di un file di proprieties con lo stesso nome della classe Action e con l'estensione .proprieties .
- Questo file di proprieties deve andare nello stesso pacchetto della classe Action.

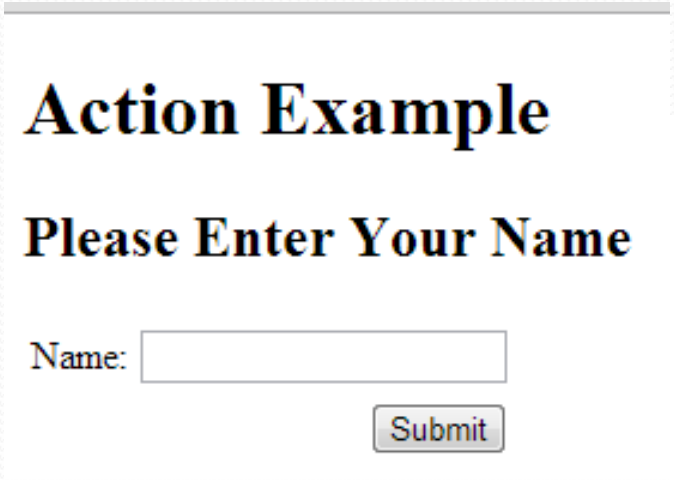
```
personBean.firstName=First name
personBean.lastName=Last name
personBean.age=Age
personBean.email=Email
thankyou=Thank you for registering ${personBean.firstName}.
```

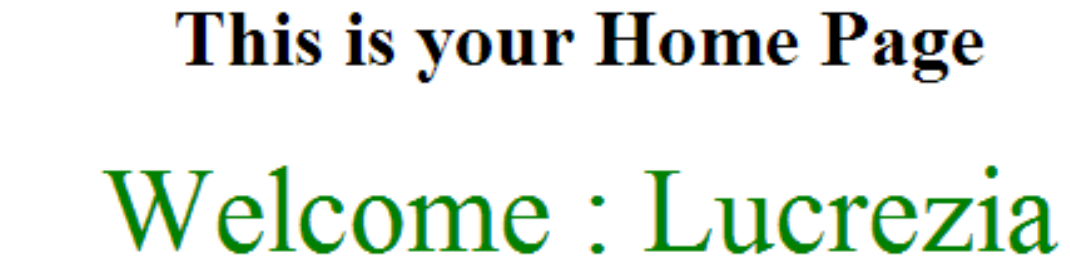
```
<s:textfield name="personBean.firstName" label="First name" />
```

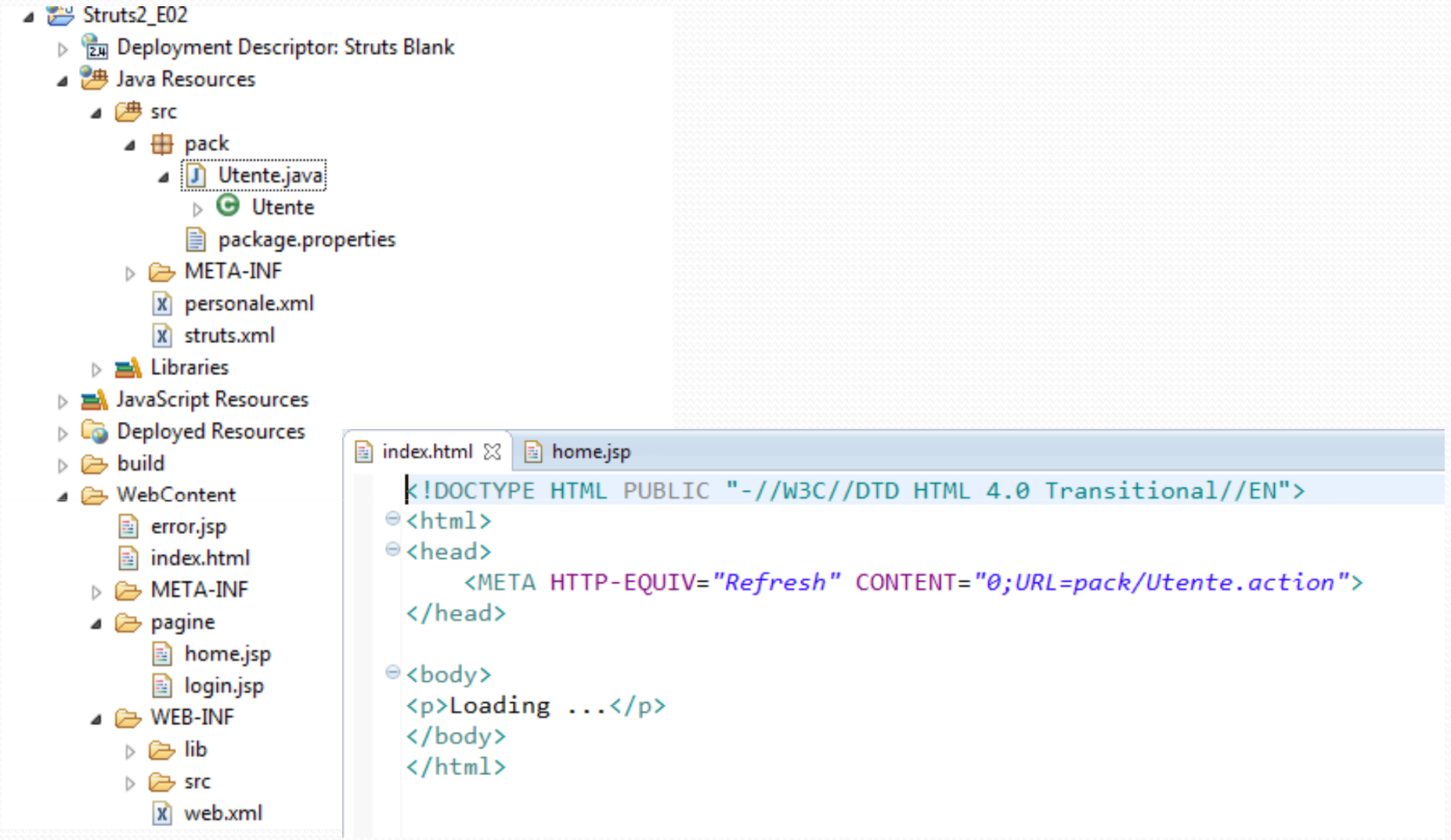
```
<s:textfield key="personBean.firstName"  />
```

- Creare a pagina dove è possibile inserire il nome di un utente

- Dopo l'inserimento del nome l'utente viene reindirizzato ad un'altra pagina di risposta che si occuperà di visualizzare un messaggio:






Struts2_E02
Deployment Descriptor: Struts Blank
Java Resources
src
pack
Utente.java
Utente
package.properties
META-INF
personale.xml
struts.xml
Libraries
JavaScript Resources
Deployed Resources
build
WebContent
error.jsp
index.html
META-INF
pagine
home.jsp
login.jsp
WEB-INF
lib
src
web.xml
index.html
home.jsp
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<META HTTP-EQUIV="Refresh" CONTENT="0;URL=pack/Utente.action">
</head>
<body>
Loading ...
</body>
</html>

Utente.java | package.properties | personale.xml | struts.xml | login.jsp | web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_9" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi

    <display-name>Struts Blank</display-name>

    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteF
    </filter>

    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>

</web-app>
```

Utente.java | package.properties | personale.xml | struts.xml | login.jsp

```
        //Apache Software Foundation//DTD Struts Configuration 2.3//EN
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

    <constant name="struts.enable.DynamicMethodInvocation" value="false" />
    <constant name="struts.devMode" value="false" />

 <package name="default" namespace="/" extends="struts-default">
        <default-action-ref name="index" />
        <global-results>
            <result name="error">/error.jsp</result>
        </global-results>
        <global-exception-mappings>
            <exception-mapping exception="java.lang.Exception" result="error"/>
        </global-exception-mappings>
<action name="index">
     <result type="redirectAction">
     <param name="actionName">utente</param>
     <param name="namespace">/pagine</param>
    </result>
</action>
</package>

<include file="personale.xml"/>

    <!-- Add packages here -->

</struts>
```

Utente.java | package.properties | *personale.xml | login.jsp

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

    <constant name="struts.enable.DynamicMethodInvocation" value="false" />
    <constant name="struts.devMode" value="false" />

<package name="pack" namespace="/pagine" extends="struts-default">

<action name="utente" class="pack.Utente" >
    <result name="ERROR">/pagine/login.jsp</result>
    <result name="SUCCESS">/pagine/home.jsp</result>
</action>

</package>



    <!-- Add packages here -->

</struts>
```

Utente.java | package.properties | login.jsp

```java
package pack;
import com.opensymphony.xwork2.Action;
public class Utente implements Action{
   String control = LOGIN;
   String name="";

   public String getName() {
         return name;
   }

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

    @Override
    public String execute() throws Exception
    {
       // TODO Auto-generated method stub
      if (this.name.length() > 0 ) {
          return "SUCCESS";
      } else {
          return "ERROR";
      }

    }
}
```

package.properties

```
label.name= Name
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO
<title>Insert title here</title>
</head>
<%@ taglib prefix="s" uri="/struts-tags" %>
<body>
<h1>Action Example</h1>

<h2>Please Enter Your Name</h2>
<s:form action="utente" >
<s:textfield name="name" label="Name" key="label.name" />
<s:submit />
</s:form>

</body>

</html>
```

- **Package Level Properties**
- Struts 2 ha la possibilità di utilizzare i file di proprietà multiple fornite nel file proprieties situato nella cartella resources presente sotto la cartella src
- Inserire il tutto in un file chiamato package.properties

- **Global Properties**
- Struts 2 ha la possibilità di utilizzare i file di proprietà multiple fornite nel file proprieties situato nella cartella resources presente sotto la cartella src
- Inserire il tutto in un file chiamato global.properties

Validation Form

- Per le Action Struts2 per convalidare l'input di un utente in un form, è necessario definire un metodo validate nella Action. Ad esempio si potrebbe verificare:
  1. L'utente deve fornire un nome
  2. L'utente deve fornire un indirizzo e-mail
  3. Utente più giovane di 18 anni non è possibile registrare
- Nel metodo validate si può fare riferimento per ottenere i valori dei campi esempio personBean utilizzando i metodi appropriati get.
- Aggiungere il seguente metodo Validate

validate method

```
public void validate(){
        if ( personBean.getFirstName().length() == 0 ){
                addFieldError( "personBean.firstName", "First name is required." );
        }

        if ( personBean.getEmail().length() == 0 ){
                addFieldError( "personBean.email", "Email is required." );
        }
        if ( personBean.getAge() < 18 ){
                addFieldError( "personBean.age", "Age is required and must be 18 or older" );
        }

}
```

- Quando l'utente preme il pulsante di invio del modulo di registrazione, Struts2 trasferirà l'input dell'utente ai campi del personBean. Poi Struts2 eseguirà automaticamente il metodo validate.

XML Validator

```
<!DOCTYPE validators PUBLIC
"-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
 <validator type="requiredstring">
        <param name="fieldname">personBean.firstName</param>
        <message>First name is required.</message>
 </validator>
</validators>
```

- Per convalidare le voci di campo di un utente del modulo è possibile utilizzare un file XML separato che contiene le regole di convalida.
- Il file XML che contiene le regole di convalida devono essere chiamati come ActionClassName-validation.xml. In applicazione di esempio, il file di convalida XML è denominato EditAction-validation.xml
- Formato XML Validator

```
<validator type="requiredstring">
       <param name="fieldname">personBean.email</param>
       <message>Email address is required.</message>
</validator>
<validator type="email">
       <param name="fieldname">personBean.email</param>
       <message>Email address not valid.</message>
</validator>
```

```
<validator type="requiredstring">
        <param name="fieldname">personBean.phoneNumber</param>
        <message>Phone number is required.</message>
 </validator>
<validator type="regex">
        <param name="fieldname">personBean.phoneNumber</param>
        <param name="expression"><![CDATA[\d{3}-\d{3}-\d{4}]]></param>
        <message>Phone number must be entered as 999-999-9999.</message>
</validator>
```

Interceptors

- Gli **Interceptor**, sono classi stateless (non mantengono uno stato tra invocazioni successive) che possono essere invocate automaticamente prima e dopo una Action.
- Di default, Struts 2, prevede un gruppo di interceptor, che vengono richiamati prima di invocare una qualsiasi action. Il cosiddetto stack di default, prevede ben 17 interceptor che lavorano dietro le quinte per offrire vari servizi. I principali sono i seguenti:
- **Exception**: permette di mappare una particolare eccezione ad una vista;
- **Prepare**: permette di richiamare un metodo di inizializzazione della Action nel caso questa implementi una determinata interfaccia;

- **I18n**: gestisce la memorizzazione del locale per l'utente corrente;
- **Debugging**: permette di attivare il debug delle viste;
- **FileUpload**: permette di gestire l'upload dei file;
- **Validation**: permette di eseguire la validazione dei dati forniti nella form, congruentemente al contenuto dei relativi file xml di definizione dei controlli

- Per configurare un nuovo Interceptor:

```
<interceptors>
    …
    <interceptor name="autowiring"
        class="interceptor.ActionAutowiringInterceptor"/>
</interceptors>
```

- Per collegare l'interceptor all'Action:

```
<action name="my" class="com.fdar.infoq.MyAction" >
    <result>view.jsp</result>
    <interceptor-ref name="autowiring"/>
</action>
```

- Il vantaggio di utilizzare Struts2 è che è possibile concentrarsi sulla logica del controller (il Struts 2 ActionSupport classe), sul livello di servizio, sul livello di accesso ai dati, sui modelli di dominio, ecc
- Le attività svolte dal framework, prima e dopo che un'azione venga eseguita, sono fatte dagli intercepton.
- intercepton sono classi Java, che vengono eseguite in un ordine specifico.

```
<action name="register" class="org.apache.struts.register.action.Register" method="execute">
        <interceptor-ref name="timer" />
        <interceptor-ref name="logger" />
        <interceptor-ref name="defaultStack">
                <param name="exception.logEnabled">true</param>
                <param name="exception.logLevel">ERROR</param>
        </interceptor-ref>
        <result name="success">thankyou.jsp</result>
        <result name="input">register.jsp</result>
</action>
```

Internazionalizzazione

- I componenti **i18n** sono packaged con lo Struts Framework
- Il primo di questi componenti, che viene gestito dall'application *Controller*, è una classe Message che referenzia un resource boundle contenente stringhe locale-dependent;
- il secondo è un custom tag JSP <bean:message /> che viene utilizzato nel View layer per presentare le stringhe gestite dal Controller

Internazionalizzazione: i Boundles di risorsa

- Il **resource boundle** è un file che contiene le coppie chiave/valore per il linguaggio dell'applicazione. Il formato del nome per questo file è *ResourceBundleName_language_COUNTRY.properties* (es. ApplicationResources_en_US.properties – tutte le risorse richieste da un client negli Stati Uniti che parla la lingua Inglese utilizzeranno questo file per effettuare un retrieve specifico).
- Quando si sviluppa un'applicazione i18n, è necessario definire un file di proprietà per ciascun linguaggio che l'applicazione utilizzerà.

Internazionalizzazione: deployment dei Boundles di risorsa

- Una volta definiti tutti i file di proprietà dell'applicazione è necessario informare Struts.
- Il tutto utilizzando il tag <init-parameter> di ActionServlet

```
..//
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  <init-param>
    <param-name>application</param-name>
    <param-value>ApplicationResources</param-value>
  </init-param>
..//
```

- Tutto quello che dobbiamo fare è copiare tutti i file di risorse nella directory classes e riavviare Tomcat.

Internazionalizzazione di un'applicazione

I passi per effettuare l'internalizzazione di una applicazione sono:

1. Creazione del resource boundles che conterrà una coppia chiave/valore utilizzata nell'applicazione.

   ApplicationResources_it_IT.properties
   app.symbol=Simbolo
   app.price=Prezzo Corrente

   ApplicationResources_en_US.properties
   app.symbol=Symbol
   app.price=current Price

2. Copia di questi file nella cartella classes dell'applicazione
3. Aggiunta del sotto-elemento <init-param> di ActionServlet, chiamato ApplicationResources

```
<init-param>
<param-name>application</param-name>
<param-value>ApplicationResources</param-value>
</init-param>
```

Internazionalizzazione di un'applicazione

4. Aggiunta di un <taglib> descrivente la bean tag library nel file web.xml

```
< taglib>
<taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
< /taglib>
```

5. Produrre la pagina JSP

```
< %@ page language="java" %>
< %@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
< %@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<html>
< head>
<title><bean:message key="app.title" /></title>
< /head>
< body>
< html:form action="Lookup">
<table width="45%" border="0">
< tr>
< td><bean:message key="app.symbol" /></td>
< td><html:text property="symbol" /></td>
< /tr>
< tr>
< td colspan="2" align="center"><html:submit /></td>
< /tr>
< /table>
< /html:form>
< /body>
< /html>
```