http://www.jugsardegna.org

Portlet Java

II Parte

di
Massimiliano Dessì
massimiliano.dessi@gruppoatlantis.com

Introduzione

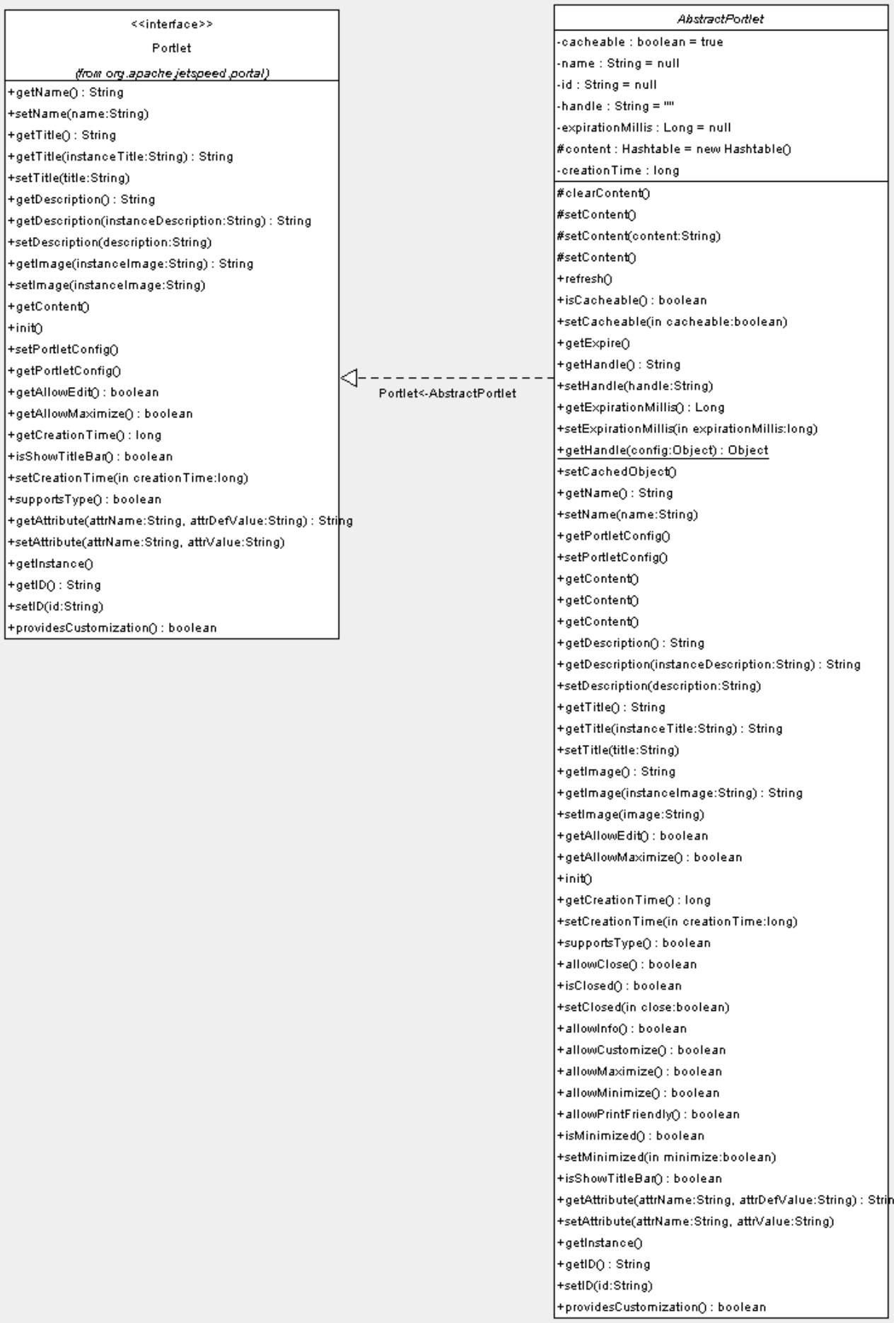
Nel numero precedente si è analizzato in maniera generale le funzionalità offerte dalle portlet e di un portlet Container open-source: Jetspeed.
Si è spiegato come vengano costruite le varie pagine attraverso i template Velocity (o Jsp) e i file psml, che richiamano le singole portlet.
Proseguiamo la spiegazione analizzando la gerarchia delle portlet, quali sono i comportamenti delle classi base, quali usare per i vari compiti, come scriverle e come deployarle.
Si farà riferimento all'ultima release disponibile in questo momento Jetspeed 1.4b4.
Questa versione probabilmente sarà la penultima in versione beta prima del rilascio di Jetspeed2 che sarà pienamente conforme alle specifiche discusse dal java community process per le portlet [1].

Gerarchia delle portlet

Per comprendere appieno alcune scelte progettuali che si possono ritrovare in alcune classi di Jetspeed è necessario risalire al periodo in cui Jetspeed è nato, all'epoca per canale informativo si intendeva la visualizzazione di contenuti statici, le web-application ancora non si erano del tutto affermate.
L'idea delle portlet, era giustificata dalla necessità di centralizzare le informazioni in un unico portale.
Perciò alcuni tipi di portlet usati allora possono risultare oggi di scarso interesse.
Mentre altre di recente introduzione rispecchiano gli standard di programmazione e di architettura attuali, cioè la completa separazione tra la parte che si vede, la parte che contiene i dati, e la parte che decide come e cosa elaborare.
A conferma di questa affermazione vediamo l'interfaccia delle portlet che furono introdotte per prime che troviamo nel package:
`org.apache.jetspeed.portal.portlet`
essa è rimasta la stessa dalla versione 1.3a1 del 2000.

Interfaccia Portlet

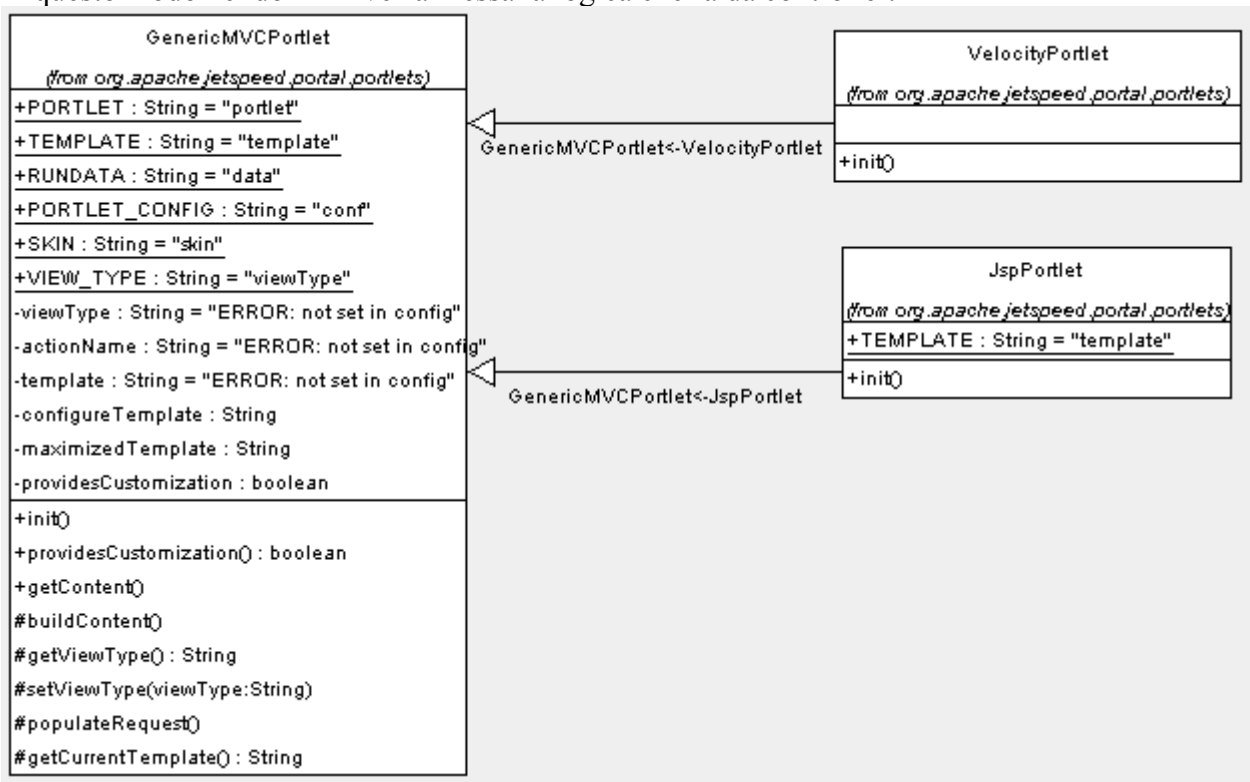
I metodi che essa espone sono quelli necessari a mostrare contenuti, a settare attributi, ma non consentono di effettuare elaborazioni.
Il metodo fondamentale in questa interfaccia è infatti il getContent() che restituisce l'html attraverso un `org.apache.ecs.ConcreteElement`
[2]ECS (element construction set) è un progetto jakarta che permette la compsizione di tag l'html attraverso gli oggetti.
Se si voleva usare una portlet di questo tipo, il file html da renderizzare andava composto riga per riga con degli StringElement dentro il codice rendendo problematica la modifica del lato di presentazione, sempre che non si avesse in un flusso di input un file html, ma la situazione non era certo più rosea perchè le modifiche non erano certo semplici.
Una classe che implementa questa interfaccia è la:
`org.apache.jetspeed.portal.portlets.AbstractPortlet` che è la classe generica contenente le funzionalità generiche delle portlet.


<<interface>>
Portlet
(from org.apache.jetspeed.portal)
+getName() : String
+setName(name:String)
+getTitle() : String
+getTitle(instanceTitle:String) : String
+setTitle(title:String)
+getDescription() : String
+getDescription(instanceDescription:String) : String
+setDescription(description:String)
+getImage(instanceImage:String) : String
+setImage(instanceImage:String)
+getContent()
+init()
+setPortletConfig()
+getPortletConfig()
+getAllowEdit() : boolean
+getAllowMaximize() : boolean
+getCreationTime() : long
+isShowTitleBar() : boolean
+setCreationTime(in creationTime:long)
+supportsType() : boolean
+getAttribute(attrName:String, attrDefValue:String) : String
+setAttribute(attrName:String, attrValue:String)
+getInstance()
+getID() : String
+setID(id:String)
+providesCustomization() : boolean
Portlet<-AbstractPortlet
AbstractPortlet
-cacheable : boolean = true
-name : String = null
-id : String = null
-handle : String = ""
-expirationMillis : Long = null
#content : Hashtable = new Hashtable()
-creationTime : long
#clearContent()
#setContent()
#setContent(content:String)
#setContent()
+refresh()
+isCacheable() : boolean
+setCacheable(in cacheable:boolean)
+getExpire()
+getHandle() : String
+setHandle(handle:String)
+getExpirationMillis() : Long
+setExpirationMillis(in expirationMillis:long)
+getHandle(config:Object) : Object
+setCachedObject()
+getName() : String
+setName(name:String)
+getPortletConfig()
+setPortletConfig()
+getContent()
+getContent()
+getContent()
+getDescription() : String
+getDescription(instanceDescription:String) : String
+setDescription(description:String)
+getTitle() : String
+getTitle(instanceTitle:String) : String
+setTitle(title:String)
+getImage() : String
+getImage(instanceImage:String) : String
+setImage(image:String)
+getAllowEdit() : boolean
+getAllowMaximize() : boolean
+init()
+getCreationTime() : long
+setCreationTime(in creationTime:long)
+supportsType() : boolean
+allowClose() : boolean
+isClosed() : boolean
+setClosed(in close:boolean)
+allowInfo() : boolean
+allowCustomize() : boolean
+allowMaximize() : boolean
+allowMinimize() : boolean
+allowPrintFriendly() : boolean
+isMinimized() : boolean
+setMinimized(in minimize:boolean)
+isShowTitleBar() : boolean
+getAttribute(attrName:String, attrDefValue:String) : Strin
+setAttribute(attrName:String, attrValue:String)
+getInstance()
+getID() : String
+setID(id:String)
+providesCustomization() : boolean

AbstractPortlet

In essa vediamo gli stessi metodi più alcuni campi necessari ad identificare la portlet tramite un id, a settare delle decorazioni e delle scritte per la portlet stessa, ma di elaborazioni vere e proprie a meno di costruire l'output stringa per stringa non ve ne sono.

Con il passare del tempo e la affermazione delle servlet, la maturazione di framework che potessero supportare il paradigma MVC si è arrivati alla situazione attuale.
Nella versione attuale 1.4b4 (la b stà per versione beta mentre le versioni 1.3 erano considerate versioni alfa) troviamo package del tipo:
`org.apache.jetspeed.portal.portlets.GenericMVCPortlet`
che consentono l'iterazione di un utente con una portlet dietro alla quale può stare una applicazione che risponde agli input dell'utente.
Una portlet di questo tipo lavora con un template Velocity o Jsp come view, la action richiamata nella request come controller, e il modello sono i dati messi dalla action nel contesto di velocity che li mostra attraverso il template.
L' uso delle action è permesso principalmente dalla classe `org.apache.turbine.modules.action`, e questo ci permette di scrivere delle action, che verranno eseguite da Turbine quando richieste nell'url.
In questo modo potremmo scrivere una portlet che estende semplicemente una action oppure più correttamente una portlet VelocityPortletAction.
Nella versione corrente le JspAction e le VelocityAction sono classi astratte ed estendono la classe org.apache.jetspeed.modules.actions.GenericMVCAction
in questo modo nel doXXX verrà messa la logica che fa da controller.



GenericMVCPortlet

Vediamo ora una classe di esempio:

```
package org.apache.jetspeed.modules.actions.portlets;

// Turbine stuff
import org.apache.turbine.util.RunData;

// Velocity Stuff
import org.apache.velocity.context.Context;
import org.apache.jetspeed.portal.portlets.VelocityPortlet;

/**
* <p>Title: Portlet Velocity minimale di esempio</p>
* <p>Description: Semplice Velocity Portlet action per Jetspeed1.4b4</p>
* @author <a href="mailto:desmax74@yahoo.it">Massimiliano Dessì</a>
* @version 1.0
*/

public class MaxxAction extends VelocityPortletAction{


      private void init(VelocityPortlet portlet, RunData data) {
      /* prendiamo un parametro dalla request attraverso l'oggetto runData che
      fa da wrapper per le informazioni che sono disponibili attraverso i metodi
      della servlet Turbine*/

            variabileRequest = data.getRequest().getParameter("testo");

      // ottengo il contesto di Velocity dal runData
            contesto = (Context)
            data.getTemplateInfo().getTemplateContext("VelocityPortletContext");
      }

      // viene associato alla chiamata nel vm
      public void doUpdate(RunData data, Context context){

      /* metto nel contesto la variabileRequest chiamandola oggetto , nel vm mi
      riferirò ad essa con lo stesso nome */
            contesto.put("oggetto", variabileRequest);
      }

      protected void buildNormalContext(VelocityPortlet portlet, Context
      context, RunData runData) throws Exception {
      init(portlet, runData);
      }

      private String variabileRequest;
      private Context contesto;
}
```

Una volta compilata e messa in una delle directory canoniche per le web application, classes o impacchettata in un jar dentro lib, è necessario un descrittore come i deployment descriptor per gli ejb.
Anche per le portlet è un file xml con estensione xreg.
In esso dichiariamo il nome (entry name) che vedremo richiamato nel file psml associato all'utente.
Il titolo (title) che verrà visualizzato nelle pagine di scelta delle portlet insieme alla descrizione (description)
Nome del template .vm associato (maxx.vm)
Il nome della classe che contiene la/le action (portlets.MAXXAction)
Il tipo di canale (media-type) su cui verrà visualizzata html

Categoty group serve per filtrare la scelta delle portlet in base al tipo.

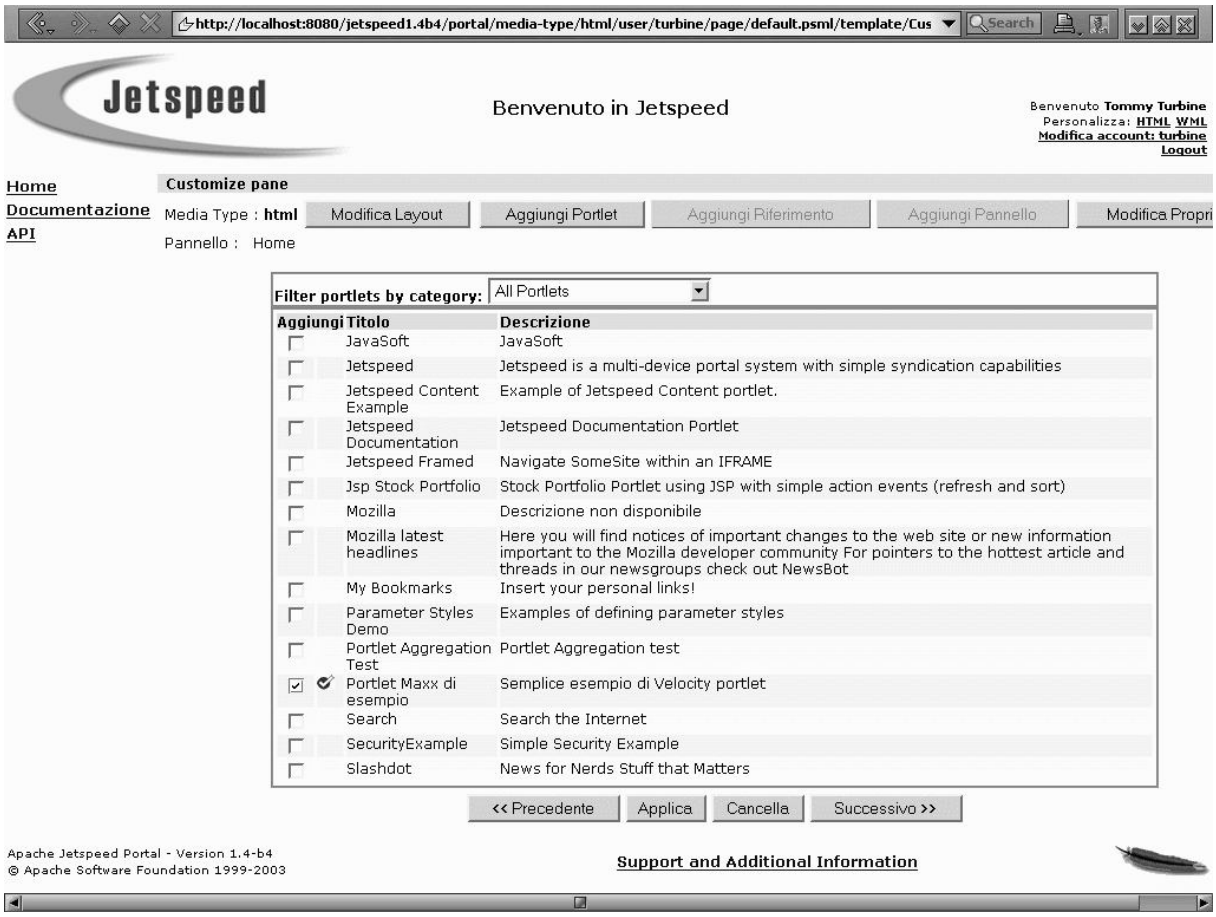
File di configurazione maxx.xreg:

```
<?xml version="1.0" encoding="UTF-8"?>
      <registry>
            <portlet-entry name="Maxx" hidden="false" type="ref"
                  parent="Velocity" application="false">
            <meta-info>
            <title>Portlet Maxx di esempio</title>
            <description>Semplice esempio di Velocityportlet</description>
            </meta-info>
            <parameter name="template" value="maxx" hidden="false"
                  cachedOnName="true" cachedOnValue="true"/>
            <parameter name="action" value="portlets.MaxxAction"
                  hidden="false" cachedOnName="true" cachedOnValue="true"/>
            <media-type ref="html"/>
            <url cachedOnURL="true"/>
            <category group="Jetspeed">velocity</category>
            <category group="Jetspeed">velocity.demo</category>
            </portlet-entry>
      </registry>
```

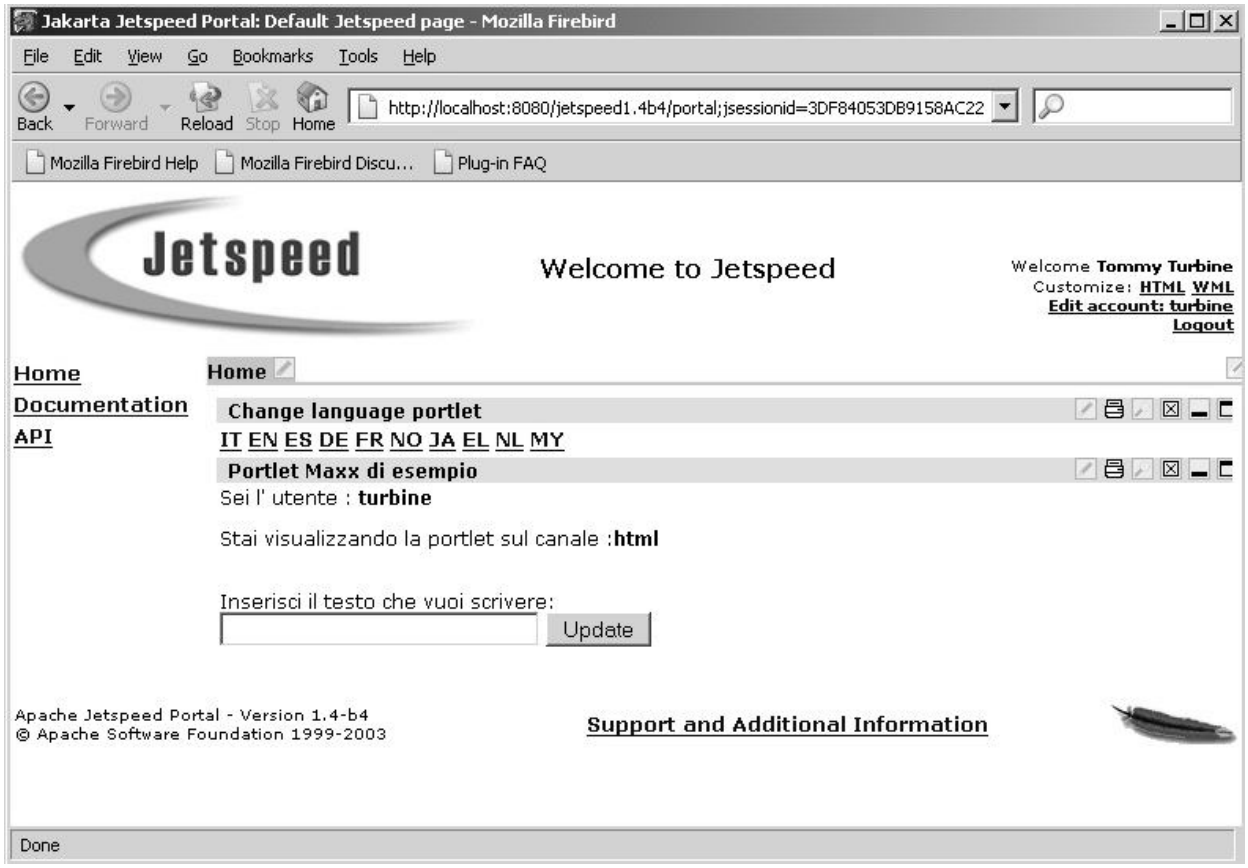
Nel template corrispondente maxx.vm

```
<form action="$jslink.Template" method="post">
      Sei l' utente :
      <b>$jslink.UserName</b>
      <br>
            <p>Stai visualizzando la portlet sul canale
      :<b>$jslink.MediaType</b>
      <br>
      #if($oggetto)
            <br>Avevi scritto :<b>$oggetto</b><br>
      #end
      <br>
      <p>Inserisci il testo che vuoi scrivere:
            <br>
            <input name="testo" value="" size="30">
            <input type="submit" name="eventSubmit_doUpdate" value="Update">
      </p>
</form>
```

avremo accesso all'oggetto messo nel contesto con il metodo put().
Con questo esempio abbiamo visto come con le Portlet Velocity (o Jsp) è possibile realizzare delle applicazioni che eseguano diverse operazioni e diverse action in base ai parametri con cui vengono chiamate, (nell'esempio doUpdate).
Una volta messo il template vm dentro la directory WEB-INF/templates/vm/portlets/html
non ci resta che salvare il nostro file xreg dentro la cartella WEB-INF/conf o copiaredentro uno già presente quello che è presente tra i tag <portlet -entry>(compresi),
un servizio di jetspeed che rilegge i file xml di configurazione ogni arco temporale specificato nei file di properties di Jetspeed provvederà a rileggere anche la nostra aggiunta a caldo (hot-plug) senza dover fermare il server.
L'utente che si autenticherà e vorra modificare le sue pagine troverà nella lista delle portlet disponibili
quella appena aggiunta da noi.



Lista di scelta delle portlet



Risultato finale

Una volta inserita la portlet nelle pagine personalizzate dall'utente, la portlet con un id univoco generato da jetspeed verrà inserita nel file psml, che a sua volta potrà essere o su una base dati oppure come file dentro la directory psml.
L'id univoco è necessario in quanto l'istanza della portlet si comporta esattamente come una servlet che può soddisfare più richieste contemporanee, ma nel caso della portlet deve anche distinguere i vari client sui quali è richiesta perchè potrebbe avere bisogno di configurazioni diverse.
Jetspeed provvederà anche ad effettuare il caching dell'output delle portlet per soddisfare più velocemente le richieste.

Conclusioni

In questo secondo articolo si è dato uno sguardo ai tipi di portlet più usate:
Le abstract portlet usate inizialmente per visualizzare contenuti statici o con poche elaborazioni, e quelle più recenti , che ricadono sotto il tipo di MVC portlet (VelocityAction e JspAction), cioè quelle idonee a eseguire elaborazioni per mezzo di action.
Questi due tipi sono la base per poter integrare applicazioni tramite le portlet.
Nel prossimo articolo vedremo portlet di tipo b2b (business to business) che consentono alle aziende di pubblicizzare i propri prodotti su altri siti e rendere disponibili i propri contenuti su altri tramite l' xml.
Vedremo inoltre le differenze con una le Portlet IBM , usate da Websphere Portal server che è costruito usando Jetspeed come fondamenta.
Daremo anche uno sguardo a ciò che il java Community process ha prodotto sulle portlet.

Bibliografia e riferimenti

Jetspeed: Enterprise Information Portal :http://jakarta.apache.org/jetspeed
Turbine: Web Application framework :http://jakarta.apache.org/turbine
Velocity: Java Template Engine: http://jakarta.apache.org/velocity
[1]jsr168: http://www.jcp.org/en/jsr/detail?id=168
[2] ECS Element construction set :http://jakarta.apache.org/ecs/

Massimiliano Dessì
è raggiungibile a desmax74@yahoo.it oppure
massimiliano.dessi@gruppoatlantis.com
ha iniziato a lavorare presso la Sistemi Informativi S.p.A (IBM)
come programmatore Java.
Da due anni lavora presso Atlantis S.p.A.(http://www.gruppoatlantis.com),
dove utilizzando metodologie agili quali l'eXtreme programming (Xp),
sviluppa applicazioni enterprise Web-based con tecnologia J2EE
quali portali e content management system per la promozione del territorio.
Nel tempo libero contribuisce al progetto open-source jakarta -Jetspeed,
di cui ha realizzato il servizio di localizzazione e la portlet
per la localizzazione nelle varie lingue, cura la versione italiana
di Jetspeed.
Laureando in Ingegneria Elettronica presso l'Università di Cagliari.