http://www.jugsardegna.org

Portlet Java

III Parte

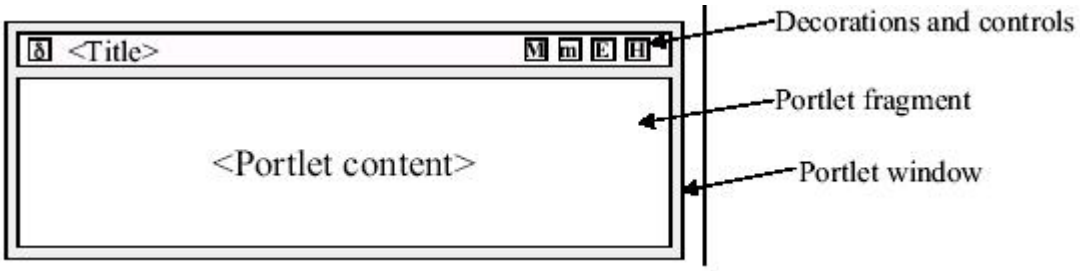
di
Massimiliano Dessì
massimiliano.dessi@gruppoatlantis.com

Introduzione

Dopo i precedenti due articoli dove veniva illustrato cosa è possibile fare con le portlet e come scrivere una semplice portlet Mvc, daremo uno sguardo a cosa il jcp ha prodotto e rilasciato sulle portlet, per espressa dichiarazione si è riusato il lavoro prodotto dalle specifiche delle Servlet2.3, infatti molti metodi sono identici.
Su queste specifiche viene costruito in questi mesi Jetspeed 2 che grazie alle portlet standard (che soddisfano i requisiti J2EE) diventerà un EIS (Enterprise information Portal) J2EE in quanto adotterà molto strettamente le tecnologie e gli standard J2EE .
Con il termine portlet application costruita usando le portlet viene ora indicata una estensione della web-application.

Portlet Container e Portal server
Una distinzione resasi ora necessaria è la distinzione tra la parte di aggregazione delle portlet e la gestione del loro funzionamento.
Abbiamo visto nel precedente articolo come deployare una portlet e visualizzarla, però non era presente una netta separazione tra chi doveva istanziare la portlet e chi la doveva visualizzare nelle pagine.
Nelle specifiche questa separazione a livello logico funzionale viene fatta, ma a livello pratico il container delle portlet e chi le aggrega potrebbe essere lo stesso framework, oppure due ben distinti.
Il Portlet Container fornisce alle portlet il necessario ambiente di esecuzione, gestisce il loro ciclo di vita e il comportamento in base al file di deploy con estensione xml (in jetspeed1.x con estensione xreg).
Il Container istanzia le portlet prima che vengano richieste, lancia le eccezioni se qualcosa va male.
Esso è una estensione di un Servlet Container e può essere costruito sopra di esso.
Il suo nome in codice è Pluto e verrà fornito dalla IBM.
Il Portal server invece è il responsabile della aggregazione delle portlet fornite dal Portlet Container.

Preferences object al servizio delle portlet.
I file di deploy che caratterizzano il funzionamento della portlet e i parametri necessari al suo funzionamento vengono associati a ciascuna portlet mediante un oggetto chiamato preferences object, l'associazione portlet-preferences-object è gestita dal container, questa loro occorrenza è chiamata portlet window (o più brevemente window).



Portlet window

Il client chiama chi risponde ?
Una volta che il container ha istanziato correttamente le portlet , esse possono essere invocate.
L' interfaccia Portlet definisce due metodi per gestire le richieste , il processAction e il render .
Il metodo processAction esegue una richiesta di elaborazione, il metodo render esegue la visualizzazione delle portlet.
Una volta che il client ha effettuato la sua richiesta , il container esegue il metodo processAction (di una sola portlet) e solo dopo che questa azione è finita vengono chiamati i metodi render delle altre portlet presenti nella pagina (in qualsiasi ordine, anche in parallelo se possibile).

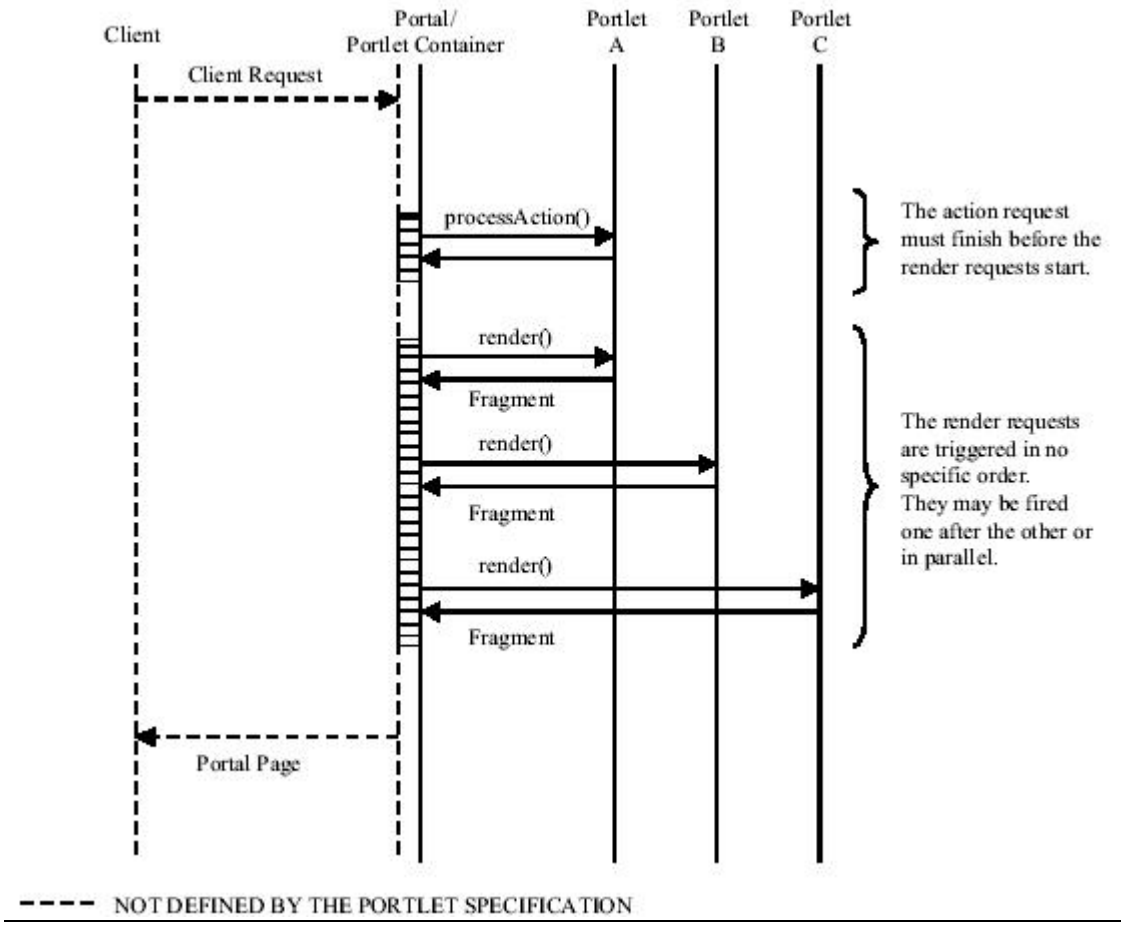


Figura 2

ProcessAction : dietro le quinte
Il metodo processAction riceve due parametri, la ActionRequest (che estende la PortletRequest) e la ActionResponse (che estende la PortletResponse).
L' ActionRequest fornisce i parametri della request , e le informazioni sulla portlet, sulle preferenze e sulla sessione, in questa punto si può anche eseguire una ridirezione se necessaria alla portlet.
Nella ActionResponse invece si può cambiare la modalità di funzionamento della portlet o il suo stato che diverrà effettivo nella successiva fase di render.

Render Request :mostrami il tuo viso
In questa operazione viene generato il contenuto, anche qui vengono passati due parametri RenderRequest e la RenderResponse, il comportamento è il medesimo della processAction solamente che ora vengono passati parametri per la visualizzazione

GenericPortlet :
Essa è una classe astratta e fornisce le funzionalità basilari e i metodi per modificare la render request.
Il metodo render della generic portlet setta il titolo usando quello presente nel deployment descriptor della portlet e invoca il metodo doDispatch.
Il metodo doDispatch viene usato a seconda dello stato corrente della portlet per il reindirizzamento ad altri metodi che sono, il doView per la view request, il doEdit per l' edit request e il doHelp per l' help request.
Il getPortletConfig restituisce l'oggetto che contiene le informazioni presenti nel deployment descriptor
Il getPortletContext che funziona allo stesso modo delle Servlet.
I metodi getPortletName, getPortletTitle che non hanno bisogno di spiegazioni in quanto il loro nome chiarisce la loro funzione , il getResourceBundle che serve per la localizzazione nelle varie lingue.
Sono inoltre presenti altri metodi di utilità.

Container e multithread delle portlet
Il Container gestisce le richieste concorrenti che arrivano alla portlet (la stessa portlet può essere visualizzata in vari punti in più pagine) da esecuzioni concorrenti di gestione richiesta su differenti thread.
Lo sviluppatore deve scrivere le portlet per gestire l'esecuzione concorrente del processAction e del render

Eccezioni
Durante la richiesta può essere lanciata una eccezione di tipo:

- PortletException , quando si è verificato un errore e il container deve pulire la request , ma se si verifica nella processAction il container mostra le altre portlet visibili nella pagina ignorando tutte le operazioni nella actionResponse della portlet che ha lanciato l' eccezione.Vengono trattate come eccezioni di questo tipo anche le RuntimeException
- PortletSecurityException , quando l' utente non ha sufficienti diritti per eseguire una operazione e il container gestirà questa evenienza nella maniera più appropriata (magari ridirezionando ad una pagina di servizio)
- UnavailableException quando la portlet è temporaneamente o permanentemente non disponibile, se è il secondo caso, il container rimuove il servizio della portlet chiamando il metodo destroy.

Fine del servizio di una portlet: metodo destroy
Quando per un qualsiasi motivo valido il container debba interrompere il servizio di una portlet chiama il metodo destroy presente nella interfaccia Portlet , la portlet rilascerà le risorse che usava e salverà in maniera persistente i dati che ne avessero bisogno.

Configurazione : PortletConfig object
Attraverso il PortletConfig la portlet accede al suo deployment descriptor che contiene tutti i dati che le sono necessari durante il funzionamento, anche parti del resource bundle se necessario (viene usato dal metodo render della GenericPortlet)

Come ti chiamo direttamente la portlet : PortletUrl
Se fosse necessario chiamare una portlet direttamente e dirgli in quale stato di visualizzazione si deve porre, lo si può fare attraverso l'oggetto PortletUrl, lo si crea chiamando il createActionURL e il createRenderURL della interfaccia RenderResponse (è una specializzazione della action url).
Va tenuto presente che questi url delle portlet non vanno usati con l' http get che è usato dal portale per codificare stati interni di funzionamento , e va fatta attenzione che i parametri aggiuntivi eventuali (aggiunti con addParameter) della portlet non abbiano nomi uguali ad altri già presenti.

Conversiamo con le Servlet e condividiamo informazioni :PortletContext
Per ciascuna portlet esiste una istanza della interfaccia (che sarà naturalmente implementata) PortletContext associata a ciascuna portlet application deployata nel container.
Nel caso il container non sia solo uno ma la computazione sia distribuita, l'istanza del PortletContext sarà una per virtual machine.
Attraverso il PortletContext è possibile accedere ai parametri di inizializzazione della portlet application , immagazzinare dati , ottenere risorse statiche e ottenere il request dispatcher per includere Servlet e Jsp.
Poichè gli attributi settati nel PortletContext devono essere messi nel ServletContext una diretta conseguenza di questo è che il ServletContext delle Servlet e delle Jsp è accessibile alle portlet e il PortletContext è accessibile dalle Servlet e dalle Jsp.

Sessioni, visibilità e attributi e relazioni con la Web-App e la sessione HTTP
Il container deve assicurare che ciascuna richiesta alle portlet generata come richieste dal portale in seguito ad una richiesta di un client , abbiano tutte la stessa sessione.
Se con queste richieste alle portlet , più di una portlet creasse una sessione, l'oggetto sessione deve essere lo stesso per tutte le portlet della stessa portlet application.
La visibilità della sessione è a livello di contesto della portlet application.
Naturalmente poichè una portlet è usata da più utenti in più modi, ciascuna portlet ha il suo unico oggetto PortletSession per identificare la sessione di ciascun utente.
Il container ha la responsabilità di non condividere il PortletSession tra portlet application o tra diversi utenti
Una portlet può mascherare attributi nella PortletSession.
Questi attributi possono avere visibilità a livello di applicazione APPLICATION_SCOPE (visibile a tutte le portlet che fanno parte della portlet application)o a livello di portlet PORTLET_SCOPE, ma in quest' ultimo caso non è protetto dagli altri componenti web, perciò bisogna prestare attenzione ai nomi che si danno.
Per conoscere la visibilità di una variabile nella sessione http (usata dalla PortletSession) va usata la PortletSessionUtil.

Essendo una portlet application una estensione di una web - application , essa può contenere Servlet e Jsp oltre alle portlet.
La PortletSession immagazzina gli attributi nella sessione HTTP perciò i dati messi nella http-session dalle Servlet e dalle Jsp sono accessibili alle portlet attraverso la PortletSession, e viceversa, quelli messi dalle portlet nella PortletSession sono accessibili alle Servlet e Jsp attraverso la HttpSession.
Se la sessione http deve essere invalidata, il container invalida anche la PortletSession associata e viceversa.
I metodi per agire sulla sessione e sui suoi parametri hanno gli stessi nomi e le stesse funzioni di quelli usati con le Servlet.

PortletRequestDispatcher
Una portlet può delegare la creazione del contenuto a una Servlet o a una Jsp, per farlo viene usato il PortletRequestDispatcher.
La portlet lo ottiene dal PortletContext, ma può usarlo solo durante l' esecuzione del metodo render.
Una volta chiamata la Servlet o la Jsp con i parametri eventuali, viene chiamato l' include sul PortletRequestDispatcher e vengono passati anche la renderRequest e la renderResponse, è compito del container assicurare che le Servlet o le Jsp chiamate dall' include del PortletRequestDispatcher, vengano chiamate nello stesso thread di questo.
Se una risorsa invocata con l'include dovesse lanciare una eccezione , questa deve essere incapsulata in una PortletException, in questo modo è il chiamante che gestisce i malfunzionamenti.

Caching
Viene definito un meccanismo di caching per portlet per singolo utente, nel deployment descriptor della portlet viene dichiarato per quanto tempo deve essere tenuto in cache il contenuto della portlet. In questo modo tutte le portlet che compongono una pagina ma che non vengono modificate da elaborazioni non devono essere rigenerate, naturalmente la portlet può sempre agire anche settando l' expiration-cache nell' oggetto RenderResponse, però deve essere necessariamente scritto nel deployment descriptor che la portlet richiede il caching altrimenti il container non esegue il caching. Non è richiesto ai container che abbiano un sistema di caching, ma se viene implementato può anche essere disabilitato parzialmente o totalmente in qualsiasi momento ci sia bisogno di liberare memoria.

Sicurezza
Una portlet application contiene risorse che saranno accedute da diversi utenti, è perciò necessario che gli utenti abbiano diversi gradi di permessi.
I ruoli definiti sono gli stessi definiti nelle specifiche 2.3 delle Servlet, per conoscere l' utente l' interfaccia Request espone tre metodi

- getRemoteUser
- isUserInRole
- getUserPrincipal

Il getRemoteUser restituisce il nome usato dal client per l' autenticazione. Il isUserInRole se l' utente ha uno specifico ruolo di sicurezza e il getUserPrincipal determina il principal name dell' utente corrente e restituisce un oggetto java.security.Principal
Usando questi tre metodi nel codice è possibile capire se l' utente è autorizzato ad eseguire determinate operazioni.
I permessi di ciò che l' utente può fare sono scritti nel deployment descriptor della portlet, se il container ne avesse bisogno e non trova i permessi dichiarati in questo file , li cerca nel web.xml della portlet application.

Ci sono anche degli specifici vincoli di sicurezza che possono essere assegnati a:

- portlet collection
- user data constraint

La prima è una collezione con i nomi delle portlet che descrivono la lista delle risorse da proteggere, in questo modo le richieste che arrivano a una portlet della collezione sono soggette a questi vincoli.
Invece lo user data constraint sono i requisiti per lo strato di trasporto delle portlet collection, questi requisiti possono essere sulla integrità del contenuto o per evitare che possano essere letti durante il trasporto (Confidential), in questo caso il container deve usare SSL se per rispondere alla request se le risorse nel deployment descriptor della portlet sono segnate con livello di trasporto CONFIDENTIAL.

Propagazione della sicurezza a un EJB container
Quando una portlet esegue chiamate a degli EJB il Portlet container propaga l' identita dell' utente all' EJB container.
Poichè i Portlet Container sono parte della piattaforma J2EE è richiesto che utenti non conosciuti dal Portlet container possano fare chiamate all EJB container, in questo caso, la Portlet application può specificare l' elemento run-as nel web.xml.
In questo caso, l' identità dichiarata come run-as viene propagata allo strato EJB con uno dei ruoli definiti nel web.xml.
Se questo non dovesse avvenire la Portlet-application è la sola che si deve occupare della autenticazione nel container EJB.

PortletTagLibrary
Attraverso le tag library le Jsp possono accedere alla RenderRequest e alla RenderResponse e al PortletConfig delle portlet, che forniscono i tag actionURL e renderURL.

TCK Test Compatibility KIT
Per assicurare la compatibilità e l' aderenza allo standard dei portlet Container, l' implementazione di quest' ultimo deve soddisfare i requisiti del TCK.
Il TCK è composto da dei test case e da 165 asserzioni che devono essere verificate affinchè il container sia certificato come aderente allo standard.

Conclusioni:
Con l' ufficializzazione dello standard sarà possibile portare le portlet da un Portlet container ad un altro, altresì, si apre ufficialmente anche il mercato dei portlet container commerciali (Sun, IBM e Oracle) e quello open source con Jetspeed 2 che aderirà in maniera molto stretta alle specifiche J2EE.
Insieme alle portlet vi è un altro importante sviluppo che cammina parallelo ma che non è ancora ufficiale, quello dei WSRP cioè i web-services for remote portlet di cui l' Apache Software Foundation insieme all' OASIS con altre società stanno definendo lo standard.
Jetspeed2 e i WSRP saranno argomento dei prossimi articoli.

Bibliografia e riferimenti
jsr168: <http://www.jcp.org/en/jsr/detail?id=168>
Documentazione ufficiale completa:http://jcp.org/aboutJava/communityprocess/review/jsr168/
Esempi portlet SUN:
http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/index.html

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.