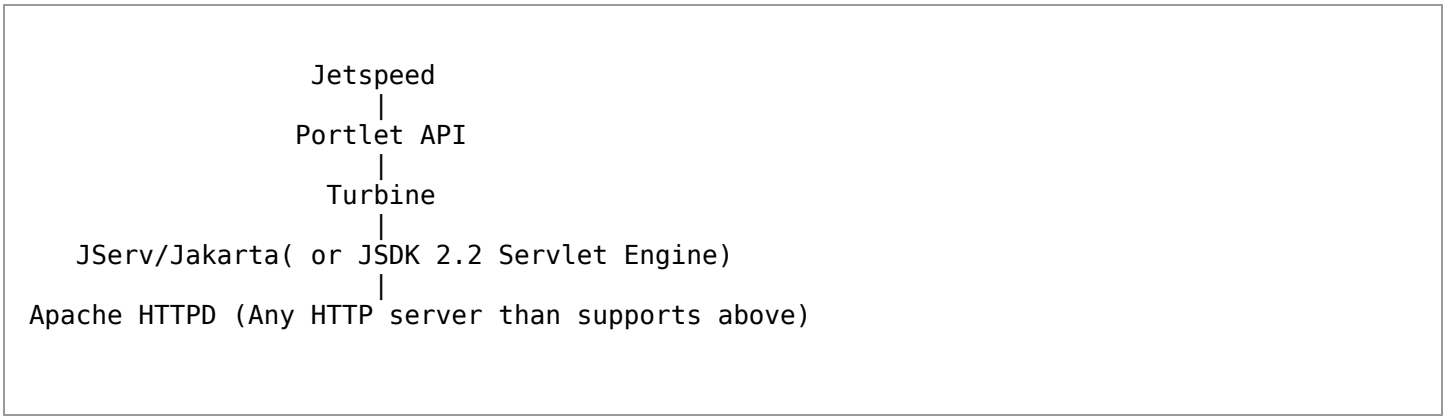
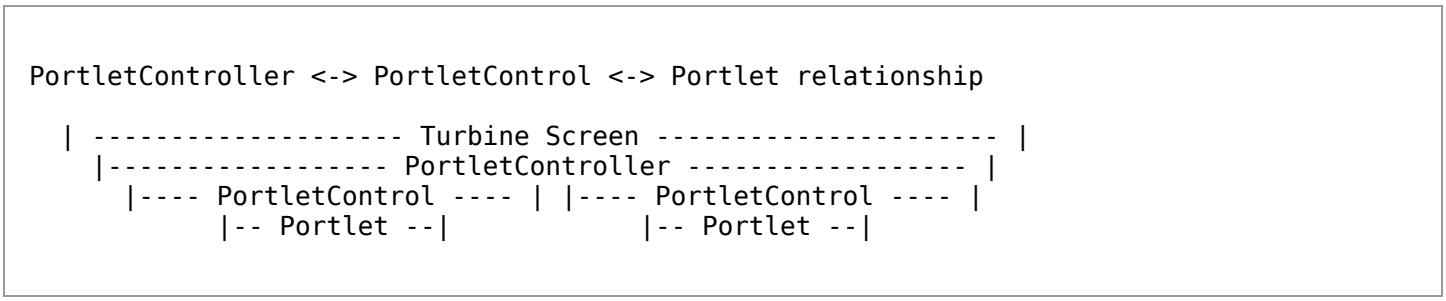
Architecture

Within Jetspeed, Turbine handles user authentication and page layout as well as scheduling. Portlets can interact with Turbine services with the provided RunData object. Pluggable templating systems, including JavaServer Pages, Cocoon, Velocity and Web Macro provide for dynamic web applications through layouts, XML/XSLT, and XSP for Cocoon. Castor handles progmatic XML manipulation. It is a very handy way to think of XML as a set of Java objects instead of dealing with SAX or DOM.



Page layout



Portlets

Early on it became apparent that a standard way of subscribing to content and managing its presentation was needed.

Portlet goals:

- have multiple but small web applications available to the user.
- "skin" these web applications so that users can choose the background color, title bar color, icons, etc.
- maintain performance across multiple Portlets by providing a caching subsystem
- keep track of all available web applications and present them to the user
- Easily choosen by the user so that custom page generation is very easy. They are very similar to Turbine Screens except that you can have muliple Portlets within any given screen.
- Fast with a caching subsystem. This allows even complex Portlets that use databases to render HTML fast.
- Easily developed and doesn't require the developer to know the implementation details of the Engine (Jetspeed) that his Portlet is running under.

- Portlets allow other types of content renderers. A portlet could use the facilities of a database via JDBC or complex XML -> XSL -> HTML rendering via something like Cocoon.
- Portlets allow themselves to be skinned so that the user can change the background color or the width of the Portlet or the color of the title bar.
- Portlets are managed by a PortletController. This controller is implementation specific and can be changed by a developer to provide custom rendering.
- Portlets are rendered and held by a PortletControl. A PortletControl adds the skin to the Portlet and then returns it's content.
- Category based subscription. The XML subscription framework within JetSpeed will allow portlets (via Portlet Markup Language) to be organized into categories. This allows easy management of portlets.
- Portlets are passed a PortletConfig object which includes a URL and a hashtable of parameters. This way a Portlet can adapt its configuration during runtime by observing its PortletConfig.
- Easy to write as an AbstractPortlet defines most of it's behavior in most situations.

A example of portlet could be a small little control that runs within a web page and allows the user to query an LDAP server for a set of phone numbers. This could be developed outside of Jetspeed and snapped into the framework by modifying an XML file to point in its direction. Developing your own Portlets is easy and is really just a matter of implementing the "Portlet" interface, providing custom implementations of the Portlet's methods.

A Portlet may also be setup so that it is a Web Applications. Portlets that are configured as applications (getApplication() == true) only run as full screen. You can't have multiple "Portlications" (Portlet Applications) on screen at once. They can be accessed by the user at any time though via the navigation features of Jetspeed.

Portles are provided with a PortletConfig that can be used from within a Portlet to determine runtime configuration information. The Portlet can get parameters it was given to modify its behavior. All the normal Servlet parameters ServletRequest, ServletResponse, or Turbine RunData.

Jetspeed ships with a few Portlet implementations:

RSSPortlet
- Renders RDF Site Summary format and presents it to the user as HTML.

FileServerPortlet
- Serves up an HTML document to the user from a URL.

CocoonPortlet
- Takes a stylesheet and a url as parameters and transforms it with Cocoon and then returns the content to the user.

PortletViewerPortlet
- Provides a way for the user to obtain additional information about a Portlet including it's configuration options, URL, and properties. It should also be possible to include additional options here such as replication with 3rd party resources (currently Avantgo).

DiskCacheDaemon

Jetspeed can be setup to work with content from all over the Internet. Due to the fact that performance conditions may change on the Internet from minute to minute a high performance cache was needed to keep identical copies of remote documents local. This is the DiskCach. When Jetspeed developers need access to a remote URL they use the DiskCache object to request the document. The DiskCache fetches the URL for them and also stores a copy on the disk. The next time the user triggers a remote document fetch the DiskCache object sees that there is already a copy on the local disk and just returns this version. This is similar in operation to the disk cache that Netscape Navigator/Mozilla and Internet Explorer use.

The only problem with the above scenario is that eventually the content will become out of date. This is where the DiskCacheDaemon comes in. Based on an interval that the Jetspeed Administrator specifies the DiskCacheDaemon(DCD) looks at every URL that is in the cache. If it has been updated remotely the DCD brings down the latest copy so that users always see fresh content.

FeedDaemon

The FeedDaemon is similar to the DiskCacheDaemon. However instead of fetching URLs from the cache it fetches remote copies of Jetspeed document feeds. Usually updated OCS channels.

Jetspeed is setup so that it can receive and publish XML information. XML input feeds are setup to use Open Content Syndication (OCS). OCS allows developers to markup a list of remote XML documents and their format/namespaces. Jetspeed should be able to except OCS feeds from multiple content providers (XMLTree, Moreover, etc). In the future Jetspeed will republish content through OCS so that 3rd parties can replicate data from Jetspeed implementations.

PortletControl and PortletController

On top of a Portlet content is rendered by a PortletControl and a PortletController.

A PortletControl provides a "box" (usually an HTML table) that the portlet runs in. The PortletControl handles rendering the title and body of the the Portlet. It is possible for a PortletControl to allow configuration by the end user

A PortletController combines multiple PortletControls (and thus Portlets) to provide an entire Page of information. The PortletController is configured via XML and has almost every option exposed to the user.

Portlet Caching

Under Jetspeed Portlets are able to take advantage of an advanced caching mechanism. In order to develop more advanced Portlets it is important to understand how this functions.

Portlets are placed within the cache with a handle that is determined via the classname and some of its PortletConfig information including parameters, URL, etc.

More advanced Portlets, ones that are essentially full blown web apps may want to turn off caching. The Portlet interface extends the Cacheable interface. With this you can provide your own isCacheable() method and return false. Jetspeed will not cache your Portlet at this point.

From time to time it may be necessary for a Portlet to request that it be removed from the cache. If this is needed you should provide your own exipire() method that can determine when to expire itself. The org.apache.jetspeed.portal.portlets.RSSPortlet provides a good example of doing this.