BEA WebLogic Server R
HTTP Publish-Subscribe Server

Version: 10.3 Tech Preview

Document Date: October 2007

Table of Contents

Overview

This document contains the functional description of the Bayeux based HTTP Publish-Subscribe Server and the usage of JMS destinations for Bayeux channels for handling subscription and publishing of messages.

What is a HTTP Publish-Subscribe Server

A HTTP Publish-Subscribe server is a channels based publish/subscribe mechanism for web based clients to send and receive asynchronous messages over HTTP. The HTTP Publish-Subscribe Server is based on the Bayeux protocol proposed by the cometd project.

The Bayeux protocol defines a contract between the client and the server for communicating with asynchronous messages over HTTP. It allows clients to register, subscribe to channels and publish messages to channel. A Bayeux channel is a named destination and/or source of events. Events are published to channels and subscribers to channels receive the published events.

The HTTP Publish-Subscribe Server can communicate with any clients which can understand Bayeux protocol. It can be used by clients to subscribe to channels (a representation in Bayeux for destinations) and publish messages to the channels. The HTTP Publish-Subscribe Server is responsible for identifying clients, negotiating trust and exchanging Bayeux messages, especially responsible for pushing event messages to subscribed clients.

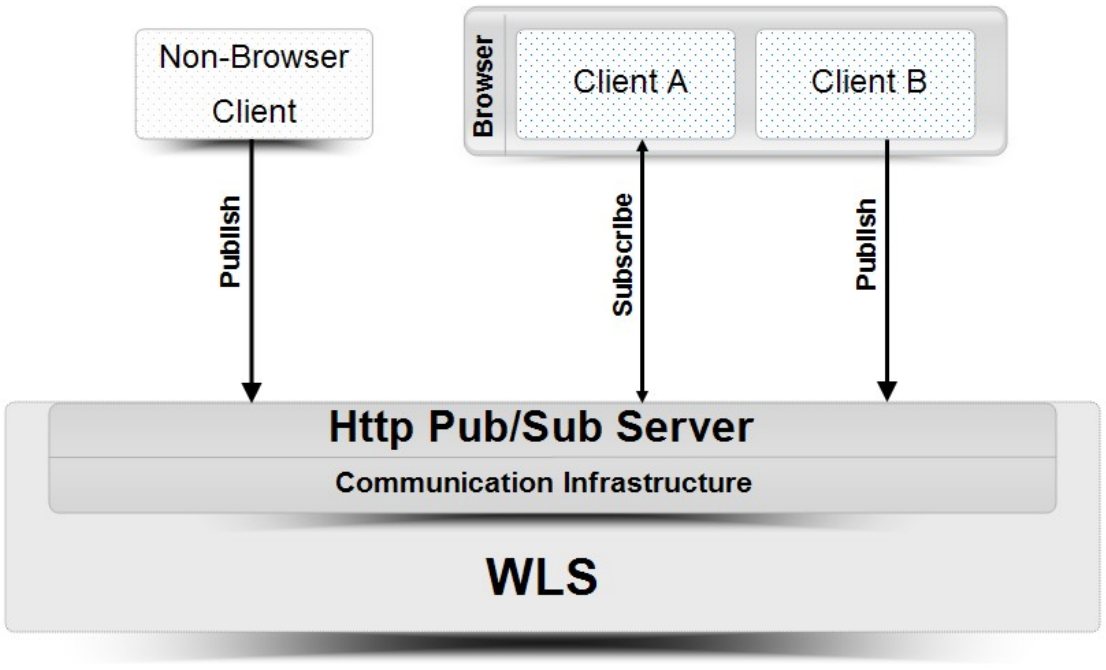

Fig.1 HTTP Publish-Subscribe Server of WebLogic Server

The HTTP Publish-Subscribe Server is used for building event driven or push-based web2.0 rich internet applications that are collaborative and capable of supporting multiple listening and publishing channels with thousands of users.

Understanding HTTP Publish-Subscribe Servers

The following sections briefly reviews the main concepts and features of a HTTP Publish-Subscribe Server.

Implementation

The HTTP Publish-Subscribe Server is implemented atop WebLogic Server Future Response API. The classical HTTP server model is a synchronous request/response model. This model is not suitable for event based scenarios because of scalability issue. With Future Response Servlet feature, servlet responses can be handled with a different thread than the one that handles the incoming request. It allows HTTP Publish-Subscribe Server can push event messages to subscribed clients asynchronously.

Controller Servlet

The Controller Servlet is the entry point for all messages from Bayuex clients and is responsible for routing HTTP requests to HTTP Publish-Subscribe Server. It needs to be configured in the web.xml descriptor with a url-pattern.

The Controller Servlet can be registered in web.xml as follows:

```
<servlet>
   <servlet-name> PubSubServlet </servlet-name>
   <servlet-class>
      com.bea.httppubsub.servlet.ControllerServlet
   </servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name> PubSubServlet </servlet-name>
    <url-pattern> /cometd/* </url-pattern>
</servlet-mapping>
```

The Controller Servlet initialization code is responsible for initializing the HTTP Publish-Subscribe Server. And once the servlet is destroyed, the HTTP Publish-Subscribe Server is shutdown as well.

Channels

Channels are named destinations to which clients subscribe to and publish messages to. Developers can define channels, mapping and security in HTTP Publish-Subscribe Server deployment descriptor file weblogic-pubsub.xml. All channel management is done at a

per web application level from the HTTP Publish-Subscribe Server. Developers can find/create/destroy Channels by invoking methods on HTTP Publish-Subscribe Server. The application designer decides whether clients can create and destroy channels or not. Channel creation or destroy operations will have to be constrained if required so that only some users (or a group of users) can perform these operations. Any attempt to do these operations by an unauthorized client must generate an error message.

When a channel is deleted, all the clients subscribed to that channel and sub-channels of that channel will be automatically unsubscribed. Unsubscribed clients will receive a disconnect response message from the server when this happens so that clients can try to reconnect and resubscribe to the interested channels.

Hierarchical namespaces and channel globbing

The channel namespace is hierarchical. A set of channels can be specified for subscriptions by a channel gobbling pattern with wildcards like '*' and '**'. The client is auto registered with any channels that are created after the client subscribed with a wildcard pattern.

Publish-Subscribe Server

One Publish-Subscribe Server is available per web application. The channel namespace is per web application. In other words, there can be channels with same names configured in two different apps. There is a PubSubServer object which is available per web application so that application code can get a handle to the Publish-Subscribe Server and perform operations on it.

Publish-Subscribe Server handler

The HTTP Publish-Subscribe Server provides an API for handling messages so that users can write their own message handlers. Developers can configure the message handlers in the configuration file.

HTTP Publish-Subscribe server has two internal handlers:

- Default Publish-Subscribe handler
  The HTTP Publish-Subscribe server provides a default in-memory Publish-Subscribe handler (which is not durable, not transactional and not clustered). This handler handles the incoming message and only notify subscribers of delivered messages.
- JMS Publish-Subscribe handler
  The HTTP Publish-Subscribe Server provides a JMS Publish-Subscribe handler which talks to a JMS provider (either external or in process) and delegates message handling to the JMS provider. The provider details should be configurable in the Publish-Subscribe Server's configuration file.

Authenticating and authorizing clients

The latest Bayeux protocol defines that clients can be authenticated with:

- No authentication

- Servlet container supported authentication (BASIC, FORM, CLIENT-CERT)
- Auth-tokens and credentials can be piggybacked on Bayeux messages within the ext fields.

HTTP Publish-Subscribe server supports the standard authentication schemes available in the Servlet container like BASIC, FORM, and CLIENT-CERT.

If an unauthenticated client tries to perform operations on a channel, a Bayeux response is sent back with 'authSuccessful' set to false. Authentication of clients can also happen outside the scope of the Bayeux protocol. If the request is accompanied by a Session cookie, and if the user is authenticated in the Session, then the same user is assumed to be authenticated. Safeguards like auth-cookie are still relevant in the Publish-Subscribe Server.

Developers can specify one role has CREATE/DELETE/PUBLISH/SUBSCRIBE permissions on channels. HTTP Publish-Subscribe server is responsible for authorizing clients. If an unauthorized client tries to perform CREATE/DELETE/PUBLISH/SUBSCRIBE actions on a channel, a Bayeux response is sent back with 'successful' set to false and the 'error' field will indicate that the client has no such permission.

> **Note**: Authorizing feature is not included in this tech-preview release. It will be included in Weblogic Server 10.3 release.

JavaScript library for the client

For Web 2.0 Ajax clients to communicate with the event server, the clients needs a JavaScript library which supports the Bayeux protocol in JavaScript. The client side Bayeux JavaScript framework may be interchangeable, so that in the future when there are alternate implementations, customers can choose the client side library of their choice.

No order of delivery or message delivery guarantee

In the web world, the clients are loosely connected and it is possible that a subscriber is inactive or not connected when a message is published. Any messages that were published when the client is unreachable will not be delivered to the client. When the clients reconnect back, they will continue to receive newly published messages. Order of delivery for messages is not guaranteed between the client and the Publish-Subscribe Server. When using the JMS Publish-Subscribe handler order of delivery, recovery of messages may be possible if the JMS is configured for it.

Runtime MBean support

Runtime MBean support feature is not included in this tech-preview release. It will be included in Weblogic Server 10.3 release.

Logging and debugging

Logging and debugging features are not included in this Tech Preview release. It will be included in Weblogic Server 10.3 release.

Bundling

The Publish-Subscribe Server is available as a web app library. The webapp library has a web.xml which specifies a default mapping so that referencing applications don't need to define the servlet mappings. However, applications can override the defaults provided in the webapp library web.xml descriptor.

> **Note**: For the Tech preview, the Publish-Subscribe server classes are made available in the system classpath.

Understanding clustering and persisting events support of HTTP Publish-Subscribe Server

This section briefly describes how HTTP Publish-Subscribe server supports clustering and persisting events.

Overview

The HTTP Publish-Subscribe server leverages Java Message Service (JMS) to support clustering and persisting event messages. In Weblogic Server cluster environment, developers can define channel to JMS topic mappings in HTTP Publish-Subscribe server deployment descriptor weblogic-pubsub.xml. All channel subscribed clients whichever managed server they're connected will receive events once event messages are published to the channel. Developers can also decide whether to persist the event message or not and specify the duration time of the persisted messages.

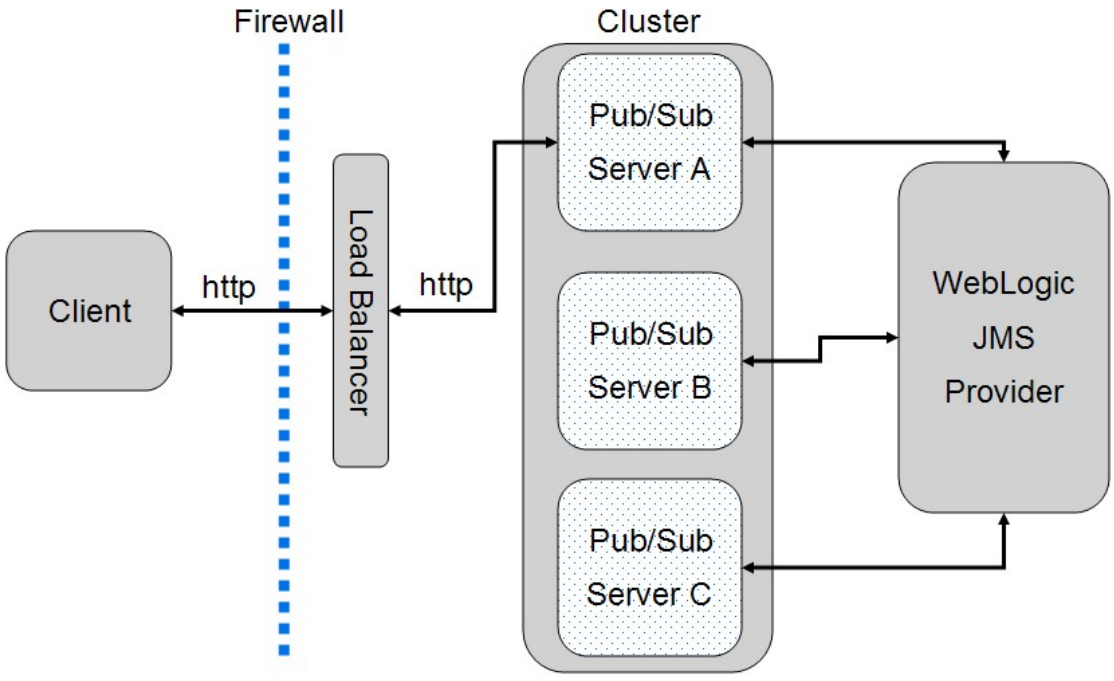

Fig.2 HTTP Publish-Subscribe server interacting with JMS

JMS Publish-Subscribe handler

The internal JMS Publish-Subscribe handler maintains a mapping from channels to JMS topics. It's responsible for dispatching event messages to correct JMS topics according to the Channel to JMS topic mapping.

JMS location transparency

The HTTP Publish-Subscribe server is capable of connecting to a JMS Server irrespective of its location. It takes advantage of performance features available when the JMS Server is run in-process.

Scalability

There can be thousands of web clients and in an application like chat, it is possible to have multiple channels and creating one topic per channel may not scale. The Publish-Subscribe Server takes advantage of indexed subscribers available with WebLogic Server JMS provider to reduce the number of topics which are created in JMS.

Failover

The HTTP Publish-Subscribe server leverages HTTP session state replication feature available in WebLogic Server to support failover. The client state is stored in sessions and is replicated in secondary server. If the primary server is unavailable, the requests from the client are routed to the secondary server. The HTTP Publish-Subscribe server can retrieve the client state from the replicated session and then reinstate the client on the secondary HTTP Publish-Subscribe server. The client does not need to re-authenticate, re-handshake, reconnect and re-subscribe.

> **Note:** Failover guarantees are not provided in the tech preview.

Publishing messages directly to JMS

It is possible for the HTTP Publish-Subscribe Server to deliver messages to the subscribed clients when event messages are published directly (not going through the Publish-Subscribe Server) to JMS. Such 'JMS Clients' have complete access to the topics.

Examples

The following sections provide examples of HTTP Publish-Subscribe application code:

Developing a basic push-based Web2.0 application

Users can develop a basic push-based Web2.0 application without any server side programming. Developer can only focus on client side development. This document will not demonstrate any client side programming since most of this part is tied to JavaScript toolkits. A Bayeux protocol supported JavaScript toolkits must be have to develop the application.

On server side, developers only need to define needed configuration such as channels, securities, and transports in the deployment descriptor file weblogic-pubsub.xml.

The following is a sample weblogic-pubsub.xml file:

```
<weblogic-pubsub>
  <!—configuration parameters for the Publish-Subscribe Server -->
    <transport>
      <type>long-pooling</type>
      <timeout-secs>30</timeout-secs>
    </transport>

   <!-- Define custom message handlers -->
  <message-handler>
    <handler-name>MyMessageHandler</handler-name>
    <handler-class>com.foo.bar.MyMessageHandler</handler-class>
  </message-handler>

  <message-handler>
    <handler-name>AnotherMessageHandler</handler-name>
    <handler-class>com.foo.bar.AnotherMessageHandler</handler-class>
  </message-handler>

  <!-- Map a message handler to a channel pattern -->
  <channel-mapping>
    <channel-pattern>/foo/bar/**</channel-pattern>
    <handler-name>MyMessageHandler</handler-name>
  <channel-mapping>

  <channel-mapping>
    <channel-pattern>/bar/*</channel-pattern>
    <handler-name>AnotherMessageHandler</handler-name>
  <channel-mapping>

  <!-- Map the jms handler to a channel pattern -->
  <channel-mapping>
    <channel-pattern>/baz/*<channel-pattern>
    <jms-handler>
      <jms-provider-url>t3://jms-server-location:7001</jms-provider-
url>
      <jms-topic-name>baz-topic</jms-topic-name>
```

```
    </jms-handler>
  </channel-mapping>

  <!-- This channel pattern be handled by default handler -->
  <channel>
    <channel-pattern>/boo/*<channel-pattern>
  </channel>

  <!-- A channel constraint can be used to constrain channel management
to a
    -- particular user/group. Constraints can also be applied based
    -- on operations
    -->
  <channel-constraint>
    <channel-pattern>/foo/*</channel-pattern>
    <channel-pattern>/bar/*</channel-pattern>
    <operation-constraint>
      <type>CREATE</type>
      <type>DELETE</type>
      <type>SUBSCRIBE</type>
      <type>DELETE</type>
    </operation-constraint>
    <auth-constraint>ADMIN</auth-constraint>
  </channel-constraint>

  <channel-constraint>
    <channel-pattern>/foo/*</channel-pattern>
    <channel-pattern>/bar/*</channel-pattern>
    <operation-constraint> SUBSCRIBE, PUBLISH </operation-constraint>
    <auth-constraint>USERS</auth-constraint>
  </channel-constraint>

</pubsub-Server>
```

Developing event based application with Server side APIs

Users can develop an event based application with server side APIs, which its business logics involve subscribing and publishing messages to Channels. Users can also implement PubSubHandler interface to handle business logic with the inbound payload of event messages and put the outbound payload into the event messages. The HTTP Publish-Subscribe server is responsible for delivering the outbound messages to subscribed clients.

The following illustrates the steps required to develop with server side APIs:

1. Find PubSubServer

```
import com.bea.httppubsub.Channel;
import com.bea.httppubsub.PubSubServer;
import com.bea.httppubsub.PubSubServerFactory;
import com.bea.httppubsub.internal.PubSubServerFactoryImpl;


PubSubServerFactory factory =
PubSubServerFactoryImpl.getInstance();


PubSubServer pubsubServer =
factory.lookupPubSubServer(“demo”);
```

2. Lookup channels from PubSubServer

```
Channel channel = pubsubServer.findOrCreateChannel(“/foo”);
// Subscribe /foo/** channels for the client
channel.subscribe(client,
Channel.SubscriptionType.ALL_SUBCHANNELS);
```

3. Publish event message

```
 // The publish method is asynchronous and returns immediately.
channel.publish(client, message);
```

Sample application of HTTP Publish-Subscribe server

Weblogic Server 10.3 bundled a push-based example application ‘stock quotes’, which is developed on dojo toolkits communicating with HTTP Publish-Subscribe server in Bayeux protocol. This ‘stock quotes’ example demostrates a real world case. User can monitor the real time stocks price on a web page without continuesly refreshing the page.

You can get the example at the directory `${WL_HOME}\samples\server\examples\src\examples\webapp\pubsub\stock`.