

## 使用 HTTP Publish-Subscribe Server

### 新的挑戰

有開發傳統的 Web 應用程式的開發人員，對 Web 程式的體認大多為，使用者要進行某個行為，例如點按一個瀏覽器畫面上的按鈕，將一個隨身碟加入到購物車中，實際上，該行為是觸發瀏覽器發送一個 Http Request 到 Web Server，而 Web Server 會因為你的 Request 回報特定的內容給你；意思就是有索取，才有回饋，而這也是 Http Protocol 所定義的標準行為。

隨著 Web 應用程式在應用面上的擴展，瀏覽器畫面的內容由平板單調的靜態內容，逐漸變成了豐富多彩的動態內容；在這幾年開始，使用者們的胃口被養大了，除了瀏覽器內容的豐富動態性外，更要求要達到有如原本 Windows 應用程式般與 Server 快速而及時的互動，至此，AJAX 的應用因應而生，而 AJAX 開發所引用 API 的根本，就是 javascript api 中的 XMLHttpRequest 物件了，利用它來讓瀏覽器網頁在進行 Request 動作時，可以不用整個頁面更新，讓 Request 在背景執行，完成後再更新，讓使用者有更好的使用體驗。

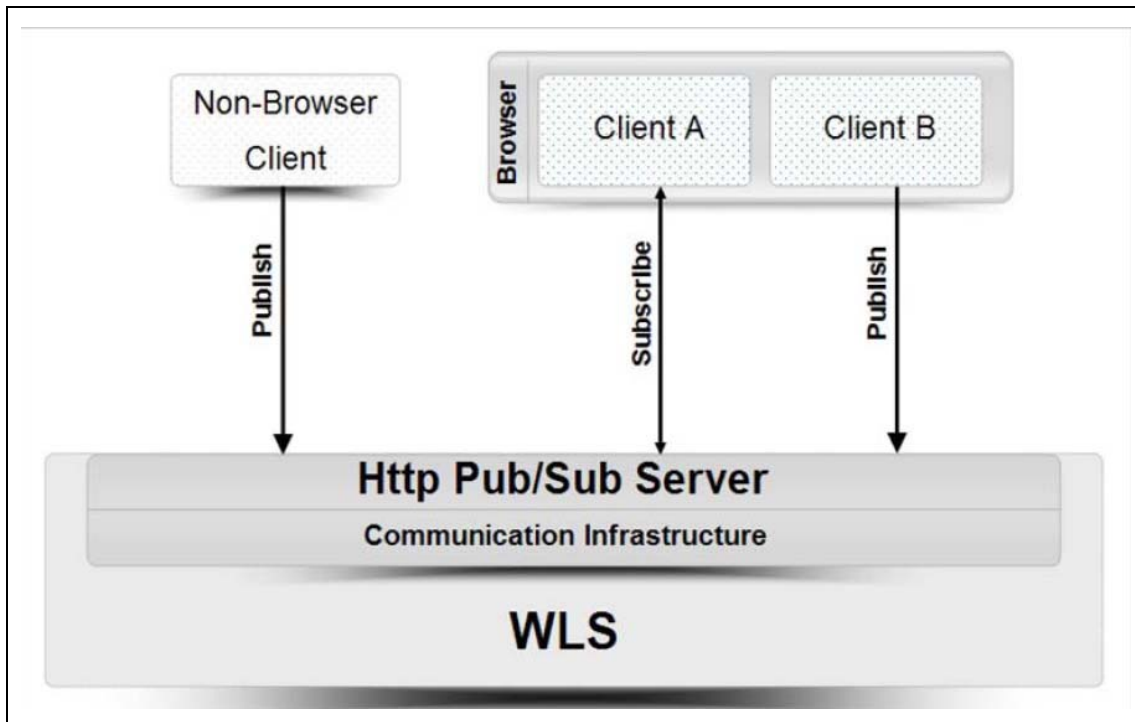
然則，Web 應用程式的開發進展至此，並非沒有缺陷的，此類 AJAX 程式雖不會令使用者感覺到整個瀏覽器畫面有被 refresh 的感覺，但實際上，對 Web Server 而言，使用者 Request 數，不但沒有減少，反而可能更增加了；此話怎講，舉個例子來說，就一個 workflow 的平台來說，使用者平時在觀看 Work List 時，舊有的習慣會對瀏覽器進行 refresh 的動作，可能久久才一次，而現在有了 AJAX，程式開發人員可能會引用 AJAX 定期在瀏覽器的背景中不斷 Request Web Server，要求內容的更新，無形之中對 Web Server 的壓力反而增加了，這時管理 Server 的人員便疑惑了，「怎麼新技術的引用，反而造成了 Server 的負擔」。

### WebLogic 的回應

WebLogic 的開發團隊已經聽到了管理者的疑問了，並且在 10gr3 這個版本做出了回應，新的解決方案，HTTP Publish-Subscribe Server 的架構應用因應而生。各位可以將此新的架構想像成 JavaEE 中 JMS 服務的 Publish and Subscribe 型態的 Topic 一般，只要使用者訂閱了訊息，Server 即會依訊息提供者所發布的內容 Push 資訊到使用者身上，而且基本上，訂閱訊息的使用者會與 Server 建立一個 Persistent 的 Http Channel，我們不需因為要更新資訊就不斷的重新建立新的 Http 連線，對 Server 在 Network 層級的負擔就大為減輕了。就用我們上個章節引用的 workflow 系統的 Work List 功能這個案例而言，使用者只要註冊了自己的帳號一次，Persistent 的 Http Channel 即會建立，Server 也會依實際上 Work List 的變動，主動 push 新的內容到使用者端。

## 具體的作法與範例

以下是 Http Publish-Subscribe Server 的架構圖



由圖可知，瀏覽器的使用者提出的一個 Persistent 的 Http Request 向 Http Pub/Sub Server 訂閱資訊，而 Http Pub/Sub Server 的資訊哪來的？就是由資訊的提供者主動發佈提供的，資訊的提供者可以是瀏覽器，也可以是一個非瀏覽器的 Client(如，Java Client 應用程式)，以下我們會開始帶各位讀者進入實作的情境中。

註：行前準備，請先將<WL\_HOME>/common/deployable-libraries/pubsub-1.0.war 佈署成爲一個 weblogic 的 sharelib。

一個 Web 應用程式模組(War)僅可以設定一個 Http Pub/Sub Server，但可以定義任意多的 Channel，這個 Channel 就有如 JMS 服務中的 Topic 扮演的腳色。那麼怎麼定義 Http Pub/Sub Server，以及定義在哪呢？

爲了設定這個奇特的 Server 的 Channel，我們必須在 Web 模組下的 WEB-INF 目錄中放上一個叫做 weblogic-pubsub.xml 的 xml 定義檔，範例內容如下：

```

<?xml version="1.0" encoding="UTF-8"?>
<wpls:weblogic-pubsub
  xmlns:wpls="http://www.bea.com/ns/weblogic/weblogic-pubsub">
  <!--channel definition, no handler specified, default handler will be used-->
  <wpls:channel>
    <wpls:channel-pattern>/stock/**</wpls:channel-pattern>
  </wpls:channel>

```

```
<wps:channel-constraint>
  <wps:channel-resource-collection>
    <wps:channel-resource-name>publish</wps:channel-resource-name>
    <wps:description>publish channel constraint</wps:description>
    <wps:channel-pattern>/stock/*</wps:channel-pattern>
    <wps:channel-pattern>/management/publisher</wps:channel-pattern>
    <wps:channel-operation>publish</wps:channel-operation>
    <wps:channel-operation>create</wps:channel-operation>
  </wps:channel-resource-collection>
  <wps:auth-constraint>
    <!--allow only user with "publisher" role to publish messages to above channels-->
    <wps:role-name>publisher</wps:role-name>
  </wps:auth-constraint>
</wps:channel-constraint>
<wps:channel-constraint>
  <wps:channel-resource-collection>
    <wps:channel-resource-name>subscribe</wps:channel-resource-name>
    <wps:description>subscribe channel constraint</wps:description>
    <wps:channel-pattern>/stock/*</wps:channel-pattern>
    <wps:channel-operation>subscribe</wps:channel-operation>
    <wps:channel-operation>create</wps:channel-operation>
  </wps:channel-resource-collection>
  <wps:auth-constraint>
    <!--allow only user with "subscriber" role to subscrib to above channels-->
    <wps:role-name>subscriber</wps:role-name>
  </wps:auth-constraint>
</wps:channel-constraint>
</wps:weblogic-pubsub>
```

此檔定義了一個叫 stock 的 channel，並且定義使用 channel 的角色限制。

有了 Http Pub/Sub Server 後，再來就是訊息的發佈者與訂閱者了，一般的訂閱者多為瀏覽器這個類型的 Client，那麼，要與 Http Pub/Sub Server 的 Channel 溝通，只能靠 javascript 了，WebLogic 利用 Open Source 的 javascript 套件"Dojo"內的 API 來用來建立一個 Persistent Channel 與 Http Pub/Sub Server 溝通，首先，我們必須在 html 頁面或者 jsp 內加上對 Dojo 的宣告

```
<script type="text/javascript" src="dojo/dojo.js"></script>
```

接下來，進行環境初始化動作

```
dojo.io.cometd.init({}, "/stock/cometd");
```

之後當然是進行訂閱註冊了

```
function onUpdate(message) {
    if (!message.data) {
        alert("bad message format "+message);
        return;
    }
    var data = message.data;
}
dojo.io.cometd.subscribe("/a/channel", null, "onUpdate");
```

還有一個要角就是資料的發佈者了，這個腳色最長由 Java 應用程式來扮演，它可以是個 Standard alone 的 Java Application，也可以是個在 Web 端的 Server Side 應用，重點是它需要引用 weblogic 提供的 API，如下：

```
import com.bea.httppubsub.FactoryFinder;
import com.bea.httppubsub.LocalClient;
import com.bea.httppubsub.PubSubSecurityException;
import com.bea.httppubsub.PubSubServer;
import com.bea.httppubsub.PubSubServerException;
import com.bea.httppubsub.PubSubServerFactory;
import org.json.JSONObject;
public class ApiBasedClient implements Client {
    private PubSubServer pubSubServer;
    public ApiBasedClient(String serverName) throws PubSubServerException {
        PubSubServerFactory pubSubServerFactory =
            (PubSubServerFactory)FactoryFinder.getFactory(FactoryFinder.PUBSUBSERVER_FACTORY);
        pubSubServer = pubSubServerFactory.lookupPubSubServer(serverName);
    }
    public void publish(String channel, JSONObject data) throws IOException {
        try {
            pubSubServer.publishToChannel(localClient, channel, data.toString());
        } catch (PubSubSecurityException e) {
            throw new IOException(e);
        }
    }
}
```

```
.....  
}
```

範例程式片段中，我們取得了抽象的 PubServer 介面後，即可對 Http Pub/Sub Server 發布訊息給使用者了。

最後，請在 Web 應用程式目錄 WEB-INF 目錄下加上修改最重要的 web.xml 檔，因為到此為止，我們仍未 config 真正 Http Pub/Sub Server 的實作，weblogic 有個 Default 的 servlet 讓我們註冊，範例如下：

```
<web-app>  
  <servlet>  
    <servlet-name>PubSubServlet</servlet-name>  
    <servlet-class>com.bea.httpsubsub.servlet.ControllerServlet</servlet-class>  
    <load-on-startup>1</load-on-startup>  
  </servlet>  
  <servlet-mapping>  
    <servlet-name>PubSubServlet</servlet-name>  
    <url-pattern>/stock/*</url-pattern>  
  </servlet-mapping>  
</web-app>
```

## 以一個股票看板程式作結

進行股票資訊的更新，發佈到 Http Pub/Sub Server

```
E:\bea_install\bea103\wlserver_10.3\samples\server\examples\src\examples\webapp\
Buildfile: build.xml

publisher.run:
    [java] CNCT, 63.19, 63.06, 8749
    [java] FCCY, 69.49, 69.28, 6475
    [java] TCHC, 56.73, 56.87, 4853
    [java] COMS, 64.27, 64.29, 5642
    [java] TDSC, 16.17, 16.12, 1265
    [java] EGHT, 47.84, 47.61, 9963
    [java] ABIX, 77.34, 77.03, 9733
    [java] ABER, 36.78, 36.86, 6638
    [java] ACAD, 74.31, 74.26, 849
    [java] ANCX, 57.64, 57.27, 5458
    [java] LEND, 42.29, 42.12, 6483
    [java] ARAY, 32.38, 32.29, 1146
    [java] ACEC, 42.68, 42.66, 8134
    [java] ACET, 57.36, 57.34, 3995
    [java] ACTL, 14.99, 15.04, 249
    [java] ACTI, 66.52, 66.37, 1484
    [java] ACTU, 17.22, 17.18, 2703
    [java] ACXM, 20.66, 20.54, 5014
    [java] ADBE, 10.68, 10.68, 5255
    [java] ABCO, 33.91, 33.84, 9984
```

訂閱股票資訊，享受無負擔的及時更新吧。

**ORACLE** WEB2.0 Sample

The data displayed in the following table is pushed by the server, click the checkbox to show/hide the stock's change line on the chart.

Chart	Symbol	Time	Open	Price	Change	Quantity	Delete
<input checked="" type="checkbox"/>	CNCT	01:46:03	63.06	63.89	1.32%	1563	<input type="checkbox"/>
<input checked="" type="checkbox"/>	COMS	01:46:05	64.29	64.56	0.42%	4426	<input type="checkbox"/>

COMS   Select a stock and subscribe, the server will start to push the stock's trade information to your browser.

