**Flex+java+bluzeds反向推送**

**作者：于海明**

反向推送的基本思想：消息服务允许Flex客户端可以发布(Publish)和订阅(Subscribe)消息，相应的，客户端的被称为生产者(Producer)和消费者(Consumer)，其实也就是发送和接受消息了。  
当一个消息被发到已订阅的目的地(destination)时，就会触发一个客户端的message事件。发布的任何消息会发给所有订阅的人，即使是生产者自己。如 果某一订阅方只想收到特定的消息，可以在订阅时使用selecor属性来指定过滤条件，只有符合条件的消息才会被服务器转发。如果想要服务器产生消息，可 以产生一个flex.messaging.messages.AsyncMessage类的实例，用 flex.messaging.MessageBroker发送出去。

示例：客户端可以发布和 订阅消息，在messaging-config.xml配置消息目的地，在程序中用“destination”映射关系来访问该类的示例，如示例中的 “chat”。注意，通道及其端点(endpoint)真正的定义是在service-config.xml中，messaging- config.xml中仅仅是引用它们。

可以看到，messaging包括两大类不同的消息通道。分别是streaming channel和polling channel，每一类又有一些扩展类型。采用polling channel，可以配置为每隔一段时间就轮询一次，也可以配置让服务器在无数据时等待，直到有数据时才响应（长轮询）。轮询响应完成后后就结束连接。

采 用streaming channel时，服务器的响应一直保持连接状态，这样服务器一次连接后，可以持续下发数据到客户端。由于HTTP连接不是双工的，所以一条AMF或 HTTP通道实际上需要2条浏览器连接，分别用于上行和下行数据。这第二条连接仅在需要向服务器发送数据时才建立，之后立即释放。这种通道特别适合实时性 高、客户端刷新的应用。比起轮询，这种模式有效减少了反复建立连接的开销。  
IE和Firefox浏览器在每个session最大连接数上有些不同。如果因此导致streaming channel建立失败，则BlazeDS会自动使用messaging-config.xml配置的下一个连接。

**本例的主要思路：**

1. 在messaging-config.xml中注册要进行推送的信息。
2. 在service-config.xml中配置相关的服务信息。
3. 用java的代码编写要推送的机制和数据。
4. 在flex页面中进行显示。
5. 运行相应的服务器地址，查看相应的显示信息

**主要代码：**

1. 在messaging-config.xml中的配置信息：

<destination id="tick-data-feed">  
         <properties>  
             <server>  
                 <allow-subtopics>true</allow-subtopics>  
                 <subtopic-separator>.</subtopic-separator>  
             </server>  
         </properties>  
         <channels>  
             <channel ref="my-polling-amf" />  
             <channel ref="my-streaming-amf" />  
         </channels>  
     </destination>

1. 在service-config.xml中配置相关的服务信息：

<channel-definition

id="my-streaming-amf" class="mx.messaging.channels.StreamingAMFChannel">  
   <endpoint url="http://{server.name}:{server.port}/{context.root}/messagebroker/streamingamf" class="flex.messaging.endpoints.StreamingAMFEndpoint"/>  
             <properties>  
                 <idle-timeout-minutes>0</idle-timeout-minutes>  
                 <max-streaming-clients>10</max-streaming-clients>  
                 <server-to-client-heartbeat-millis>5000</server-to-client-heartbeat-millis>  
                 <user-agent-settings>  
                     <user-agent match-on="MSIE" kickstart-bytes="2048" max-streaming-connections-per-session="1"/>  
                     <user-agent match-on="Firefox" kickstart-bytes="2048" max-streaming-connections-per-session="1"/>  
                 </user-agent-settings>  
             </properties>  
         </channel-definition>

1. 在java类中编写相关的代码：(可以为servlet或jsp)当访问servlet或jsp时会自动的推送相关的数据到服务器中，flex中编写相关的获取代码，在此页面中运行了相关的代码时，会接收java的服务器中推送的数据。java发送消息的代码为：

AsyncMessage msg = new AsyncMessage();  
                 msg.setDestination("tick-data-feed");  
                 msg.setHeader("DSSubtopic", "tick");  
                 msg.setClientId(clientID);  
                 msg.setMessageId(UUIDUtils.createUUID());  
                 msg.setTimestamp(System.currentTimeMillis());  
                 msg.setBody(“数据”);  
                 msgBroker.routeMessageToService(msg, null);

1. 在flex中编写相关的接收代码：

**var** consumer:Consumer = **new** Consumer();

consumer.destination = **"tick-data-feed"**;

consumer.subtopic = **"tick"**;

consumer.channelSet = **new** ChannelSet([ **"my-streaming-amf"** ]);

consumer.addEventListener(MessageEvent.MESSAGE, messageHandler);

consumer.subscribe();

并在监听函数中写获取数据:

**private** **function** messageHandler(event:MessageEvent):**void**

{

**var** tick:String = event.message.body **as** String;

**if** (tick.toString() != **null**)

{

Alert.show(**""** + tick.toString());

txtTick.text = tick.toString();

}

}

1. 运行Flex：[http://localhost:8080/blazeds/HelloWorld-debug/HelloWorld.html](http://localhost:8080/blazeds/HelloWorld-debug/HelloWorld.html?debug=true" \t "_blank)

运行Java中的jsp：[http://localhost:8080/blazeds/](http://localhost:8080/blazeds/jsp)tick.Jsp

**查看相关的页面信息，恭喜。成功了。**