You are here: Home ▸ Enterprise Java ▸ CometD: Facebook similar chat for your Java web application January 6, 5687 2:20 pm

About Semika Kaluge

CometD: Facebook similar chat for your Java web application

by Semika Kaluge on October 17th, 2012 | Filed in: Enterprise Java Tags: CometD, Facebook

Chatting is easy just like eating a piece of cake or drinking a hot coffee. Have you ever thought about developing a chat program by yourself?. You know that, it is not easy as chatting. But, if you are a developer and if you read to the end of this article, you may put a try to develop a chatting application by your self and allow your users to chat via your web application.I had to implement a chatting application for my web application. As every one does, I started to search on internet. I found IRC. When I read and search more about IRC, I understood that finding a web base client for IRC was difficult. I wanted to have more customizable web client which is working similar to Facebook. At last and luckily, I found CometD.

Finally, I was able to implement chatting application by using CometD and more customizable chat windows opening on the browser which is exactly similar to Facebook. This works almost all the modern browsers. This article explains step by step, How to implement chatting application from the scratch and also How to integrate chatting application to your existing Java base web application. Remember, Your web application should be a Java base one.

You need to download the cometD from their official web site. It has all the dependencies required to implement the chatting application except tow java script libraries. I have written two Javascript libraries, one to create dynamic chat windows like Facebook and other to handle CometD chatting functionality in generic way. If you can manage these stuff by your self, you don't need to use those tow Javascript libraries. Actually, CometD documentation provides good details. But, I go ahead with the tutorial by using those tow libraries. Any way, I recommend first use those tow libraries and then customize it as you need. I hope to share the sample application with you and you can deploy it in your localhost and test, how it works.

1.Adding required jar files.

If you use maven to build your project, add the following dependencies into your pom.xml file

```
<dependencies>
    <dependency>
        <groupId>org.cometd.java</groupId>
        <artifactId>bayeux-api</artifactId>
        <version>2.5.0</version>
    </dependency>
    <dependency>
         <groupId>org.cometd.java</groupId>
         <artifactId>cometd-java-server</artifactId>
         <version>2.5.0</version>
    </dependency>
    <dependency>
         <groupId>org.cometd.java</groupId>
         <artifactId>cometd-websocket-jetty</artifactId>
         <version>2.5.0</version>
         <exclusions>
             <exclusion>
                 <groupId>org.cometd.java</groupId>
                 <artifactId>cometd-java-client</artifactId>
             </exclusion>
         </exclusions>
   </dependency>
   <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.6.6</version>
   </dependency>
   <dependency>
        <groupId>org.cometd.java</groupId>
        <artifactId>cometd-java-annotations</artifactId>
        <version>2.5.0</version>
   </dependency>
</dependencies>
```

If you are not using maven to build your project, just copy the following .jar files into **/WEB-INF/lib** folder from your CometD download bundle. You can find these .jar files from **/cometd-demo/target/cometd-demo-2.5.0.war**file.

- bayeux-api-2.5.0.jar
- cometd-java-annotations-2.5.0.jar

- cometd-java-common-2.5.0.jar
- cometd-java-server-2.5.0.jar
- cometd-websocket-jetty-2.5.0.jar
- javax.inject-1.jar
- jetty-continuation-7.6.7.v20120910.jar
- jetty-http-7.6.7.v20120910.jar
- jetty-io-7.6.7.v20120910.jar
- jetty-jmx-7.6.7.v20120910.jar
- jetty-util-7.6.7.v20120910.jar
- jetty-websocket-7.6.7.v20120910.jar
- jsr250-api-1.0.jar
- slf4j-api-1.6.6.jar
- slf4j-simple-1.6.6.jar

2.Adding required Javascript files.

You need to link the following Javascript files.

- cometd.js
- AckExtension.js
- ReloadExtension.js
- jquery-1.8.2.js
- jquery.cookie.js
- jquery.cometd.js
- jquery.cometd-reload.js
- chat.window.js
- comet.chat.js

The ‘ **chat.window.js**‘ and ‘ **comet.chat.js**‘ are my own tow Javascript libraries which does not come with CometD distribution. If you are totally following this tutorial, you have to link those tow libraries as well. Provided sample application has these tow Javascript libraries.

3.Writing chat service class.

```
/**
 * @author Semika siriwardana
 * CometD chat service.
 */
package com.semika.cometd;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;

import javax.inject.Inject;

import org.cometd.annotation.Configure;
import org.cometd.annotation.Listener;
import org.cometd.annotation.Service;
import org.cometd.annotation.Session;
import org.cometd.bayeux.client.ClientSessionChannel;
import org.cometd.bayeux.server.BayeuxServer;
import org.cometd.bayeux.server.ConfigurableServerChannel;
import org.cometd.bayeux.server.ServerMessage;
import org.cometd.bayeux.server.ServerSession;
import org.cometd.server.authorizer.GrantAuthorizer;
import org.cometd.server.filter.DataFilter;
import org.cometd.server.filter.DataFilterMessageListener;
import org.cometd.server.filter.JSONDataFilter;
import org.cometd.server.filter.NoMarkupFilter;

@Service('chat')
public class ChatService {
    private final ConcurrentMap<String, Map<String, String>> _members = new ConcurrentHashMap<String, Map<String, String>>();
    @Inject
    private BayeuxServer _bayeux;
    @Session
    private ServerSession _session;

    @Configure ({'/chat/**','/members/**'})
    protected void configureChatStarStar(ConfigurableServerChannel channel) {
        DataFilterMessageListener noMarkup = new DataFilterMessageListener(new NoMarkupFilter(),new BadWordFilter());
        channel.addListener(noMarkup);
        channel.addAuthorizer(GrantAuthorizer.GRANT_ALL);
    }

    @Configure ('/service/members')
    protected void configureMembers(ConfigurableServerChannel channel) {
        channel.addAuthorizer(GrantAuthorizer.GRANT_PUBLISH);
        channel.setPersistent(true);
    }

    @Listener('/service/members')
    public void handleMembership(ServerSession client, ServerMessage message) {
        Map<String, Object> data = message.getDataAsMap();
        final String room = ((String)data.get('room')).substring('/chat/'.length());

        Map<String, String> roomMembers = _members.get(room);
        if (roomMembers == null) {
            Map<String, String> new_room = new ConcurrentHashMap<String, String>();
            roomMembers = _members.putIfAbsent(room, new_room);
            if (roomMembers == null) roomMembers = new_room;
        }
        final Map<String, String> members = roomMembers;
        String userName = (String)data.get('user');
        members.put(userName, client.getId());
        client.addListener(new ServerSession.RemoveListener() {
            public void removed(ServerSession session, boolean timeout) {
                members.values().remove(session.getId());
                broadcastMembers(room, members.keySet());
            }
        });

        broadcastMembers(room, members.keySet());
    }

    private void broadcastMembers(String room, Set<String> members) {
        // Broadcast the new members list
        ClientSessionChannel channel = _session.getLocalSession().getChannel('/members/'+room);
        channel.publish(members);
    }

    @Configure ('/service/privatechat')
    protected void configurePrivateChat(ConfigurableServerChannel channel) {
        DataFilterMessageListener noMarkup = new DataFilterMessageListener(new NoMarkupFilter(),new BadWordFilter());
        channel.setPersistent(true);
        channel.addListener(noMarkup);
        channel.addAuthorizer(GrantAuthorizer.GRANT_PUBLISH);
    }

    @Listener('/service/privatechat')
    protected void privateChat(ServerSession client, ServerMessage message) {
        Map<String,Object> data = message.getDataAsMap();

        String room = ((String)data.get('room')).substring('/chat/'.length());
        Map<String, String> membersMap = _members.get(room);
        if (membersMap == null) {
            Map<String,String>new_room=new ConcurrentHashMap<String, String>();
            membersMap=_members.putIfAbsent(room,new_room);
            if (membersMap==null)
                membersMap=new_room;
        }

        String peerName = (String)data.get('peer');
        String peerId = membersMap.get(peerName);

        if (peerId != null) {

         ServerSession peer = _bayeux.getSession(peerId);

            if (peer != null) {
             Map<String, Object> chat = new HashMap<String, Object>();
                String text = (String)data.get('chat');
                chat.put('chat', text);
                chat.put('user', data.get('user'));
                chat.put('scope', 'private');
                chat.put('peer', peerName);
                ServerMessage.Mutable forward = _bayeux.newMessage();
                forward.setChannel('/chat/' + room);
                forward.setId(message.getId());
                forward.setData(chat);

                if (text.lastIndexOf('lazy') > 0) {
                    forward.setLazy(true);
                }
                if (peer != client) {
                    peer.deliver(_session, forward);
                }
                client.deliver(_session, forward);
            }
        }
    }

    class BadWordFilter extends JSONDataFilter {
        @Override
        protected Object filterString(String string) {
            if (string.indexOf('dang') >= 0) {
                throw new DataFilter.Abort();
            }
            return string;
        }
    }
}
```

4.Changing web.xml file.

You should add the following filter into your web.xml file.

```
<filter>
     <filter-name>continuation</filter-name>
     <filter-class>org.eclipse.jetty.continuation.ContinuationFilter</filter-class>
</filter>
<filter-mapping>
     <filter-name>continuation</filter-name>
     <url-pattern>/cometd/*</url-pattern>
</filter-mapping>
```

And also the following servlet.

```
<servlet>
    <servlet-name>cometd</servlet-name>
    <servlet-class>org.cometd.annotation.AnnotationCometdServlet</servlet-class>
    <init-param>
         <param-name>timeout</param-name>
         <param-value>20000</param-value>
    </init-param>
    <init-param>
         <param-name>interval</param-name>
         <param-value>0</param-value>
    </init-param>
    <init-param>
         <param-name>maxInterval</param-name>
         <param-value>10000</param-value>
    </init-param>
    <init-param>
         <param-name>maxLazyTimeout</param-name>
         <param-value>5000</param-value>
    </init-param>
    <init-param>
         <param-name>long-polling.multiSessionInterval</param-name>
         <param-value>2000</param-value>
    </init-param>
    <init-param>
         <param-name>logLevel</param-name>
         <param-value>0</param-value>
    </init-param>
    <init-param>
         <param-name>transports</param-name>
         <param-value>org.cometd.websocket.server.WebSocketTransport</param-value>
    </init-param>
    <init-param>
         <param-name>services</param-name>
         <param-value>com.semika.cometd.ChatService</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>cometd</servlet-name>
    <url-pattern>/cometd/*</url-pattern>
</servlet-mapping>
```

5.Implementing client side functions.

I think this section should be descriptive. If you allows your users to chat with other users, you need to show the list of online users in you web page, just like Facebook shows the online users inside the right side bar. For that, you can place a simple <span> or <div> tag inside your page. I have done it as follows.

```
<div id='members'></div>
```

All the online users will be displayed with in the above container. Once you click on a particular user name, it will open a new chat window similar to Facebook. For each pair of users, it will open a new chat window. To get this behaviour, you should use '**chat.window.js**' which I mentioned before. Chatting in between particular pair of users will continue through a dedicated chat window.
Just after user is logging into your web application as usual way, we should subscribe that user to chat channels. You can do it using the following way.

```
$(document).ready(function(){
    $.cometChat.onLoad({memberListContainerID:'members'});
});
```

Note that, I have passed the 'id' of online user list container as a configuration parameter. Then, user should be joined with channel as follows.You can call the bellow method with the username.

```
function join(userName){
   $.cometChat.join(userName);
}
```

Since for each chat, there is a dedicated chat window just like Facebook, we should maintain global Javascript array to store those created chat window objects. You need to place the following Javascript code inside your page.

```
function getChatWindowByUserPair(loginUserName, peerUserName) {
    var chatWindow;
    for(var i = 0; i < chatWindowArray.length; i++) {
       var windowInfo = chatWindowArray[i];
       if (windowInfo.loginUserName == loginUserName && windowInfo.peerUserName == peerUserName) {
           chatWindow =  windowInfo.windowObj;
       }
    }
    return chatWindow;
}

function createWindow(loginUserName, peerUserName) {
    var chatWindow = getChatWindowByUserPair(loginUserName, peerUserName);
    if (chatWindow == null) { //Not chat window created before for this user pair.
       chatWindow = new ChatWindow(); //Create new chat window.
       chatWindow.initWindow({
             loginUserName:loginUserName,
             peerUserName:peerUserName,
             windowArray:chatWindowArray});

       //collect all chat windows opended so far.
       var chatWindowInfo = { peerUserName:peerUserName,
                              loginUserName:loginUserName,
                              windowObj:chatWindow
                            };

       chatWindowArray.push(chatWindowInfo);
   }
   chatWindow.show();
   return chatWindow;
}
```

As I mentioned above, declare following global Javascript variable.

```
var chatWindowArray = [];
var config = {
    contextPath: '${pageContext.request.contextPath}'
};
```

Since I am using a JSP page, I have to get the context path via ‘ **pageContext**‘ variable. If you are using a HTML page, manage it by your self to declare ‘config’ Javascript global variable. Now, you almost reached to last part of the tutorial.

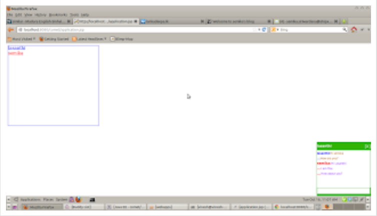
5.How does the sample application works?

You can download the comet.war file and deploy it in your server. Point the browser to following URL.

http://localhost:8080/comet

This will bring you to a page which has a text field and button called ‘Join’. Insert some user name as you wish and click on ‘Join’ button. Then you will be forwarded to a another page which has list of online users. Your name is highlighted in red color. To chat in your local machine, You can open another browser (IE and FF) and join to the chat channel. The peer user displays in blue color in the online users list. Once you click on a peer user, it will open a new chat window so that You can chat with him. This functions very similar to Facebook chatting.

I have tested this chatting application in IE, FF and Crome and works fine. If you want any help of integrating this with your Java base web application, just send me a mail.



Reference: Facebook similar chat for your Java web application. from our JCG partner Semika loku kaluge at the Code Box blog.

25 Responses to "CometD: Facebook similar chat for your Java web application"

sathish

November 10th, 2014 at 12:08 pm

hi,
I need your help.I downloaded comet.war and to deployed into eclipse its deployed successfully and then run the project display the index page but i click the join button error occured.The error mentioned below
101 [main] WARN org.cometd.annotation.AnnotationCometdServlet – Failed to create annotated service com.semika.cometd.ChatService

```
java.lang.ClassNotFoundException: com.semika.cometd.ChatService
at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1387)
at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1233)
at org.eclipse.jetty.util.Loader.loadClass(Loader.java:101)
at org.eclipse.jetty.util.Loader.loadClass(Loader.java:80)
at org.cometd.annotation.AnnotationCometdServlet.newService(AnnotationCometdServlet.java:94)
at org.cometd.annotation.AnnotationCometdServlet.processService(AnnotationCometdServlet.java:80)
at org.cometd.annotation.AnnotationCometdServlet.init(AnnotationCometdServlet.java:64)
at javax.servlet.GenericServlet.init(GenericServlet.java:212)
at org.apache.catalina.core.StandardWrapper.loadServlet(StandardWrapper.java:1173)
at org.apache.catalina.core.StandardWrapper.load(StandardWrapper.java:993)
at org.apache.catalina.core.StandardContext.loadOnStartup(StandardContext.java:4149)
at org.apache.catalina.core.StandardContext.start(StandardContext.java:4458)
at org.apache.catalina.core.ContainerBase.start(ContainerBase.java:1045)
at org.apache.catalina.core.StandardHost.start(StandardHost.java:722)
at org.apache.catalina.core.ContainerBase.start(ContainerBase.java:1045)
at org.apache.catalina.core.StandardEngine.start(StandardEngine.java:443)
at org.apache.catalina.core.StandardService.start(StandardService.java:516)
at org.apache.catalina.core.StandardServer.start(StandardServer.java:710)
at org.apache.catalina.startup.Catalina.start(Catalina.java:583)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
at java.lang.reflect.Method.invoke(Unknown Source)
at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:288)
at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:413)
Nov 10, 2014 3:27:14 PM org.apache.catalina.core.StandardContext loadOnStartup
SEVERE: Servlet /comet threw load() exception
java.lang.ClassNotFoundException: com.semika.cometd.ChatService
at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1387)
at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1233)
at org.eclipse.jetty.util.Loader.loadClass(Loader.java:101)
at org.eclipse.jetty.util.Loader.loadClass(Loader.java:80)
at org.cometd.annotation.AnnotationCometdServlet.newService(AnnotationCometdServlet.java:94)
at org.cometd.annotation.AnnotationCometdServlet.processService(AnnotationCometdServlet.java:80)
at org.cometd.annotation.AnnotationCometdServlet.init(AnnotationCometdServlet.java:64)
at javax.servlet.GenericServlet.init(GenericServlet.java:212)
at org.apache.catalina.core.StandardWrapper.loadServlet(StandardWrapper.java:1173)
at org.apache.catalina.core.StandardWrapper.load(StandardWrapper.java:993)
at org.apache.catalina.core.StandardContext.loadOnStartup(StandardContext.java:4149)
at org.apache.catalina.core.StandardContext.start(StandardContext.java:4458)
at org.apache.catalina.core.ContainerBase.start(ContainerBase.java:1045)
at org.apache.catalina.core.StandardHost.start(StandardHost.java:722)
at org.apache.catalina.core.ContainerBase.start(ContainerBase.java:1045)
at org.apache.catalina.core.StandardEngine.start(StandardEngine.java:443)
at org.apache.catalina.core.StandardService.start(StandardService.java:516)
at org.apache.catalina.core.StandardServer.start(StandardServer.java:710)
at org.apache.catalina.startup.Catalina.start(Catalina.java:583)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
at java.lang.reflect.Method.invoke(Unknown Source)
at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:288)
at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:413)
```

thanks in advance

Reply

Kernel

December 2nd, 2014 at 4:30 pm

Where is the source code for the application?

Reply

« Older Comments

Leave a Reply

Name (Required)

Mail (will not be published) (Required)

Website

7 − two =

□ Notify me of follow up comments via e-mail. You can also subscribe without commenting.

☑ Sign me up for the newsletter!

Submit Comment

Knowledge Base

- Academy
- Examples
- Resources
- Tutorials
- Whitepapers

Partners

- Mkyong

The Code Geeks Network

- Java Code Geeks
- .NET Code Geeks
- Web Code Geeks

Hall Of Fame

- “Android Full Application Tutorial” series
- GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial
- Advantages and Disadvantages of Cloud Computing – Cloud computing pros and cons
- Android Google Maps Tutorial
- Android Location Based Services Application – GPS location
- 11 Online Learning websites that you should check out
- Java Best Practices – Vector vs ArrayList vs HashSet
- Android JSON Parsing with Gson Tutorial
- Android Quick Preferences Tutorial
- Difference between Comparator and Comparable in Java

About Java Code Geeks

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike. JCGs serve the Java, SOA, Agile and Telecom communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.