Pushing real-time data to the browser using cometD and Spring

2013/02/11 1 COMMENT (HTTPS://CHIMPLER.WORDPRESS.COM/2013/02/11/PUSHING-REAL-TIME-DATA-TO-THE-BROWSER-USING-COMETD-AND-SPRING/#COMMENTS)

Comet (http://en.wikipedia.org/wiki/Comet_(programming)) is a set of techniques which allows web applications to push data to the browser. It is also known as Ajax Push, Reverse Ajax and HTTP push server among others.

It is used in web applications to display fluctuating stock prices in real time, in chat applications, collaborative documentation editing and a lot of other applications.

Without comet, the only way the browser can get updates from the server is by periodically issuing AJAX calls to check for updates. This technique is not efficient in term of bandwidth and load on the server but has the advantage of being easily implemented.

Comet can be implemented by using the following techniques:

- long-polling: the browser issues a request to the server. The server waits until there is a message available to be sent to the client or after a suitable timeout. Then the browser immediately sends a new request to the server to repeat this process.
- callback-polling: the web page uses a script tag to load javascript code. The server keeps the connection open and sends a callback of a function with the message content in parameter (e.g. handleMessage({“msg”:”this is the message 1″})). The server can sends multiple messages with the same connection until a timeout occurs. The browser handles those calls as they come in(it does not need to wait for the end of the connection). This method is quite effective as we don’t need to constantly open and close the connection.
- websocket (http://en.wikipedia.org/wiki/WebSocket): it allows a browser to open a full duplex communication to the server using a single TCP connection. All the main browsers have this implemented in their latest version.
- flash socket/java applet socket: it uses a flash applet or a java applet to open a durable connection with the server. When the

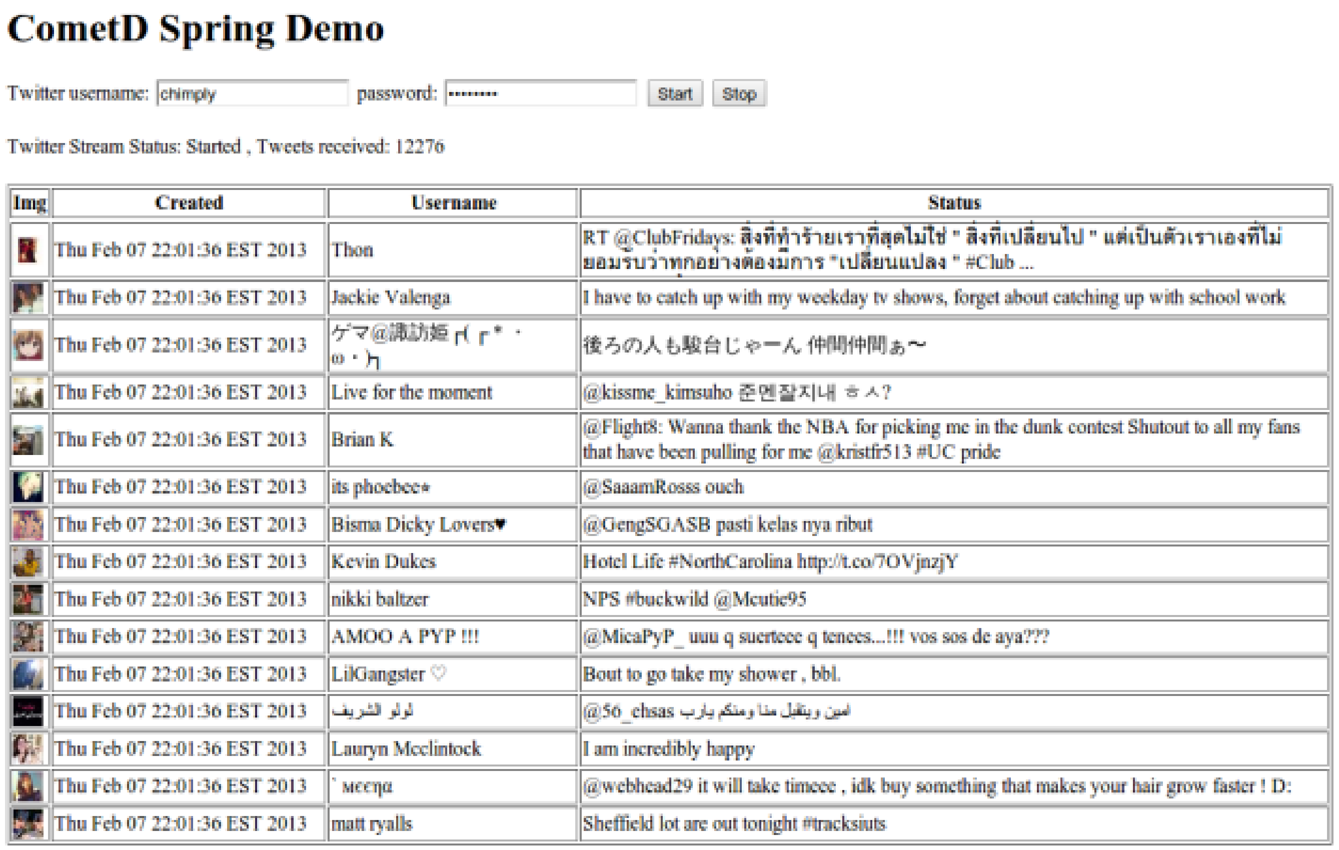
applet receives data from the server, it forwards them to the web page by calling callback javascript method.

For a good description of those techniques, you can look at the push technology (http://en.wikipedia.org/wiki/Push_technology) page on wikipedia.

Depending on the browser, its version and the network settings (proxy, firewall), some of those techniques might work and some might not. So to overcome this issue, comet frameworks usually implement several techniques.

The are a lot of comet frameworks on the market today. We have tried the following in the past 4 years:

- LightStreamer (http://www.lightstreamer.com/). They have an interesting push model which is a mix of a cache model and publisher/subscriber model. The application writes data to a cache. The client(browser, java application, flash application, …) can get the current value of an element in the cache and listen to its updates. If the client asks for an element in the cache which is not there yet, lightstreamer can try to populate it(from the database or another source) and sends it to the client.
- CometD (http://cometd.org/). It's an open source project developed by the Dojo Foundation. It uses the publisher subscriber model to push the data to the clients.
- DWR (http://directwebremoting.org/dwr/index.html). This open source AJAX framework was very popular 4 years ago but the project seems to have not been updated for several years. It allows to define proxy class in javascript to directly call java method and in the last version 3, it provides a reverse ajax implementation.



CometD Spring Demo

Twitter username: chimply password: •••••••• Start Stop

Twitter Stream Status: Started , Tweets received: 12276

Img	Created	Username	Status
	Thu Feb 07 22:01:36 EST 2013	Thon	RT @ClubFridays: สิ่งที่ทำร้ายเราที่สุดไม่ใช่ " สิ่งที่เปลี่ยนไป " แต่เป็นตัวเราเองที่ไม่ยอมรับว่าทุกอย่างต้องมีการ "เปลี่ยนแปลง " #Club ...
	Thu Feb 07 22:01:36 EST 2013	Jackie Valenga	I have to catch up with my weekday tv shows, forget about catching up with school work
	Thu Feb 07 22:01:36 EST 2013	ゲマ@諏訪姫 ┌(┌＊・ω・)┐	後ろの人も駿台じゃーん 仲間仲間ぁ〜
	Thu Feb 07 22:01:36 EST 2013	Live for the moment	@kissme_kimsuho 준멘잘지내 ㅎㅅ?
	Thu Feb 07 22:01:36 EST 2013	Brian K	@Flight8: Wanna thank the NBA for picking me in the dunk contest Shutout to all my fans that have been pulling for me @kristfr513 #UC pride
	Thu Feb 07 22:01:36 EST 2013	its phoebee⋆	@SaaamRosss ouch
	Thu Feb 07 22:01:36 EST 2013	Bisma Dicky Lovers♥	@GengSGASB pasti kelas nya ribut
	Thu Feb 07 22:01:36 EST 2013	Kevin Dukes	Hotel Life #NorthCarolina http://t.co/7OVjnzjY
	Thu Feb 07 22:01:36 EST 2013	nikki baltzer	NPS #buckwild @Mcutie95
	Thu Feb 07 22:01:36 EST 2013	AMOO A PYP !!!	@MicaPyP_ uuu q suerteee q tenees...!!! vos sos de aya???
	Thu Feb 07 22:01:36 EST 2013	LilGangster ♡	Bout to go take my shower , bbl.
	Thu Feb 07 22:01:36 EST 2013	لولو الشريف	@56_chsas امين ويتقبل منا ومنكم يارب
	Thu Feb 07 22:01:36 EST 2013	Lauryn Mcclintock	I am incredibly happy
	Thu Feb 07 22:01:36 EST 2013	` мєєηα	@webhead29 it will take timeee , idk buy something that makes your hair grow faster ! D:
	Thu Feb 07 22:01:36 EST 2013	matt ryalls	Sheffield lot are out tonight #tracksiuts

(https://chimpler.files.wordpress.com/2013/02/cometd-screenshot.png)

In our tutorial, we are going to get a live stream of twitter statuses, and publish them to a web page using cometD. CometD supports multiple techniques to push data to the browser: websocket, long-polling, and callback-polling.

We will use the following tools/libraries:

- Spring Framework 3.0 (http://www.springsource.org/spring-framework)
- cometD 2.5.1 (http://cometd.org/)
- jQuery 1.9.0 (http://jquery.com)
- Maven 3 (http://maven.apache.org/ref/3.0/)
- twitter4j (http://twitter4j.org/en/index.html)
- jetty (http://jetty.codehaus.org/jetty/) or tomcat (http://tomcat.apache.org/)

Requirements

In order to compile and run the web application, you would need Java (http://www.oracle.com/technetwork/java/javase/downloads/index.html) (>=1.6) and Maven 3 (http://maven.apache.org/download.cgi).

All the code shown in this post can be downloaded from github:

```
git clone https://github.com/fredang/cometd-spring-example
```

If you want to run the example you can go to the Deployment section of this post.

Configuring CometD

web.xml

To configure cometd in web.xml, we need to:

- define the context param org.eclipse.jetty.server.context.ManagedAttributes (lines 12-15)
- use the CometdServlet and map it to /cometd/* path and make sure that it's started before the CometdServlet by setting load-on-startup to 1(lines 39-48)
- make sure that the Spring DispatcherServlet is loaded after the CometdServlet by setting a higher number for load-on-startup (line 32)

```
+ expand source
```

Configuring the channels

To define the channel that the clients can listen to to get the status updates, we use annotations:

- annotation on the class

  `@Service("twitter")`

- annotation on a method to configure the channel and set the authorization privileges

  `@Configure ({"/twitter/samples"}`

```
@javax.inject.Named
@javax.inject.Singleton
@Service("twitter")
public class CometTwitterService {

    @Inject
    private BayeuxServer bayeuxServer;
    @Session
    private ServerSession serverSession;

    @Configure ({"/twitter/samples"})
    protected void configureTwitterSamples(ConfigurableServerChannel channel) {
        channel.addAuthorizer(GrantAuthorizer.GRANT_SUBSCRIBE);
    }

    public void publishMessage(Map<String, Object> msg, String id) {
        ServerChannel channel = bayeuxServer.getChannel("/twitter/samples");
        channel.publish(serverSession, msg, id);
    }
}
```

Then we need to process those annotation with the CometConfigurer

```
@Component
@Singleton
```

```java
public class CometConfigurer implements DestructionAwareBeanPostProcessor, ServletContextAware {
    private BayeuxServer bayeuxServer;
    private ServerAnnotationProcessor processor;

    @Inject
    public void setBayeuxServer(BayeuxServer bayeuxServer) {
        this.bayeuxServer = bayeuxServer;
    }

    @PostConstruct
    public void init() {
        this.processor = new ServerAnnotationProcessor(bayeuxServer);
    }

    public Object postProcessBeforeInitialization(Object bean, String name) throws BeansException
        System.out.println("Configuring service " + name);
        processor.processDependencies(bean);
        processor.processConfigurations(bean);
        processor.processCallbacks(bean);
        return bean;
    }

    public Object postProcessAfterInitialization(Object bean, String name) throws BeansException {
        return bean;
    }

    public void postProcessBeforeDestruction(Object bean, String name) throws BeansException {
        processor.deprocessCallbacks(bean);
    }

    @Bean(initMethod = "start", destroyMethod = "stop")
    public BayeuxServer bayeuxServer() {
        BayeuxServerImpl bean = new BayeuxServerImpl();
        bean.setOption(BayeuxServerImpl.LOG_LEVEL, "3");
        return bean;
    }

    public void setServletContext(ServletContext servletContext) {
        servletContext.setAttribute(BayeuxServer.ATTRIBUTE, bayeuxServer);
    }
```

```
}
```

To make the CometConfigurer executed by spring, we add a <component-scan> element in the spring xml file:

```
<context:component-scan base-package="com.chimpler.example" />
```

Getting streaming data from Twitter

To access the Twitter streaming API (https://dev.twitter.com/docs/api/1.1/get/statuses/sample), we are using the twitter4j library.

If you are not familiar with the twitter streaming API, it allows to get a data stream of statuses with a REST API call. To give you a quick idea of what you can get from the streaming API, you can do:

```
$ curl -u <TWITTER USERNAME>:<TWITTER PASSWORD> "https://stream.twitte
```

Press CRTL+C to stop.

```
public class TwitterStatusProducer {
    private final static Logger logger = Logger.getLogger(TwitterStatusProducer.class.getName());

    private TwitterStream twitterStream;
    private CometTwitterService cometTwitterService;

    public void setCometTwitterService(CometTwitterService cometTwitterService) {
        this.cometTwitterService = cometTwitterService;
    }

    public synchronized void startSample(String username, String password) {
        if (twitterStream != null) {
            return;
        }
        TwitterStreamFactory factory = new TwitterStreamFactory(
                new ConfigurationBuilder().setUser(username).setPassword(password)
                .build());
        twitterStream = factory.getInstance();
        twitterStream.addListener(new StatusAdapter() {
            public void onStatus(Status status) {
                Map<String, Object> map = new HashMap<String, Object>();
                map.put("status", "OK");
                map.put("createdAt", status.getCreatedAt().toString());
                map.put("username", status.getUser().getName());
                map.put("profileImageUrl", status.getUser().getMiniProfileImageURL());
                map.put("text", status.getText());
                cometTwitterService.publishMessage(map, Long.toString(status.getId()));
            }

            @Override
            public void onException(Exception ex) {
                Map<String, Object> map = new HashMap<String, Object>();
                map.put("status", "ERR");
                map.put("text", ex.getMessage());
                cometTwitterService.publishMessage(map, "-1");
                stopSample();
            }
        });
        logger.log(Level.INFO, "Starting listening to twitter sample");
            twitterStream.sample();
    }

    public synchronized void stopSample() {
        if (twitterStream == null) {
            return;
        }

        logger.log(Level.INFO, "Stopping listening to twitter sample");
        try {
            twitterStream.shutdown();
        } catch (Exception e) {}
        twitterStream = null;
    }
}
```

In this class, we define a method to start listening to the sample streaming and publish the statuses to the cometd channel and one method to stop listening. They will be called by the ViewController.

ViewController

The ViewController class can handle 3 HTTP requests:

- /index: to show the view file index.jsp
- /startTwitterService: it is called by an ajax call when the user press the start button
- /stopTwitterService: it is called by an ajax call when the user press the stop button

```
@Controller
@Singleton
public class ViewController {
    private TwitterStatusProducer twitterStatusProducer;

    @RequestMapping(value="/index")
    public String index() {
        return "index";
    }

    @RequestMapping(value="/startTwitterService", produces="application/text")
    @ResponseBody
    public String startTwitterService(@RequestParam(value="username") String username,
            @RequestParam(value="password") String password) {
        twitterStatusProducer.startSample(username, password);
        return "OK";
    }

    @RequestMapping(value="/stopTwitterService", produces="application/text")
    @ResponseBody
    public String stopTwitterService() {
        twitterStatusProducer.stopSample();
        return "OK";
    }

    public void setTwitterStatusProducer(TwitterStatusProducer twitterStatusProducer) {
        this.twitterStatusProducer = twitterStatusProducer;
    }
```

Spring xml file

We instantiate the ViewController, CometTwitterService and TwitterStatusProducer in the spring configuration file

+ expand source

Web page

```
function startTwitterService() {
    $('#twitterStreamStatus').text("Starting...");
    $.post('startTwitterService', {
        "username": $('INPUT[name=username]').val(),
        "password": $('INPUT[name=password]').val()
    }, function() {
        $('#twitterStreamStatus').text("Started");
    });
    return false;
}

function stopTwitterService() {
    $('#twitterStreamStatus').text("Stopping...");
    $.post('stopTwitterService', null, function() {
        $('#twitterStreamStatus').text("Stopped");
    });
    return false;
}

function startSubscription() {
    cometd.subscribe('/twitter/samples', function(msg) {
        if (msg.data.status == 'ERR') {
            alert("Error: " + msg.data.text);
            $('#twitterStreamStatus').text("Error");
        } else {
            $('#tweetTable>tbody').prepend($('<tr>'
                + '<td><img src="' + msg.data.profileImageUrl + '" /></td>'
                + '<td>' + msg.data.createdAt + '</td>'
                + '<td>' + msg.data.username + '</td>'
                + '<td>' + msg.data.text + '</td>'
                + '</tr>'));
            $('#tweetTable>tbody>tr:eq(15)').remove();
            tweetCount++;
            $('#tweetCount').text("" + tweetCount);
        }
    });
}

var cometd = $.cometd;
cometd.registerTransport('websocket', new org.cometd.WebSocketTransport());
cometd.registerTransport('long-polling', new org.cometd.LongPollingTransport());
cometd.registerTransport('callback-polling', new org.cometd.CallbackPollingTransport());

var r = document.location.href.match(/^(.*)\/spring/);
cometd.init(r[1] + '/cometd');

var tweetCount = 0;

cometd.addListener('/meta/handshake', function(handshake) {
    if (handshake.successful === true ) {
        cometd.batch(function () {
            startSubscription();
        });
    }
});
```

```

cometd.handshake();

</script>

<h1>CometD Spring Demo</h1>
Twitter username:
<input type="text" name="username" />
password:
<input type="password" name="password" />
<input type="button" value="Start" onClick="startTwitterService()" />
<input type="button" value="Stop" onClick="stopTwitterService()" />
<br /><br />
Twitter Stream Status: <span id="twitterStreamStatus">-</span>
, Tweets received: <span id="tweetCount">-</span>
<br /><br />
<table id="tweetTable" border="1">
    <colgroup>
        <col width="20" />
        <col width="220" />
        <col width="200" />
        <col width="600" />
    </colgroup>
    <thead>
        <th>Img</th>
        <th>Created</th>
        <th>Username</th>
        <th>Status</th>
    </thead>
    <tbody>
    </tbody>
</table>
```

On the web page, you can enter your twitter username and password and press start. You should see a list of status showing on the web page. You can stop receiving the list of statuses by clicking on stop.

Deployment

Deploying on jetty

You can compile and deploy the project to jetty by typing:

```
mvn jetty:run
```

In your browser open the following url: http://localhost:8080/spring/index (http://localhost:8080/spring/index)

Deploying on tomcat

If you want to deploy on tomcat:

```
mvn clean install
cp target/cometd-spring-example-1.0.war <TOMCAT DIR>/webapps/
```

And start tomcat.
Then in your browser open the following URL: http://localhost:8080/cometd-spring-example-1.0/spring/index (http://localhost:8080/cometd-spring-example-1.0/spring/index)

Note that when you deploy on tomcat, cometd cannot use the websocket protocol and will use the long-polling technique instead
You can also open multiple browsers on the same URL and see each of them receiving the same data.

We have shown in this tutorial how to publish data to the browser. We have not shown how to publish data to a particular client. It can be for example to update in real time the balance of a user's account or in a chat application to send private messages. You can look at this tutorial (http://code.google.com/p/cometd/wiki/BayeuxChatExample) to see how to do that.

Troubleshooting

You may come across some issues while integrating cometd to your web applications.

From my logs, it looks like my web apps is deployed two times

It's possible that your spring servlet xml file is deployed by both the DispatcherServlet and the ContextLoaderListener. To make sure that the DispatcherServlet is not trying to deploy it, set the init param to an empty string:

```
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value></param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>
```

By looking at the HTTP request from Firebug/Chrome developer tools, it seems that it's doing a regular polling

If you are using other filters in your web.xml, make sure that you set the value of async-supported to true:

```
<filter>
    <filter-name>UrlRewriteFilter</filter-name>
    <filter-class>org.tuckey.web.filters.urlrewrite.UrlRewriteFilter</filter-class>
    <async-supported>true</async-supported>
</filter>
```

Also make sure that your web.xml is using the version 3.0.

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee (http://java.sun.com/xml/ns/javaee)"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance (http://www.w3.org/2001/XMLSchema-inst
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee (http://java.sun.com/xml/ns/javaee)

http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd (http://java.sun.com/xml/ns/javaee/web-app_3_0.xs

        version="3.0">
```

It seems that my classes are scanned two times by the CometConfigurer class

Make sure that you are not instantiating the class CometConfigurer in your spring servlet xml file.

FILED UNDER COMET, SPRING FRAMEWORK, WEB APPLICATION TAGGED WITH COMETD, SPRING, TOMCAT, TWITTER

About these ads (http://wordpress.com/about-these-ads/)

About chimpler
http://www.chimpler.com

One Response to *Pushing real-time data to the browser using cometD and Spring*

Ayyad says:
2013/10/10 at 8:30 am
i had an error 401
any help

Reply

Follow “Chimpler”