WhatIsComet

What is the Apache Tomcat Comet API

The term Comet was coined by Alex at the Dojo Foundation, it might be a good start to read the introduction on the Comet "http://alex.dojotoolkit.org/?p=545"

Tomcat has a Comet implementation, this page is intended to serve as a quick FAQ to explain the basic question around Comet.

What Is Comet?

- Comet is a client/server interaction handled over one HTTP request and one HTTP response
- Comet is able to push data from the server and from the client, to over come the regular HTTP request polling scalability limitation
- Comet is a way to do asynchronous servlets in Tomcat, ie send a response asynchronously to a client
- Comet is implemented using the Comet Processor/Comet Event interfaces in Tomcat
- Comet is an event based implementation, events are mainly thrown based on underlying IO
- A presentation on Tomcat Comet can be found here "http://people.apache.org/~fhanik/ApacheCon2007-ZeroLatencyHTTP.ppt"

What Comet is Not.

- It is not a socket API.
- It is not a high level poller or poller API, for underlying poll()/select() methods
- It is not a TCP socket, even though the events are fired based on IO

API Proposals

Currently there are two API proposals, one lead by remm(sandbox) and one by fhanik(trunk). Everything in here is up for change and comments. The proposals are not very far from each other, but confusion around Comet has lead to further confusion around the APIs. Here is an outline of differences

1. both intend to implement an event based model
2. sandbox API has events as implicit subscriptions
3. trunk API has explicits subscriptions, one must subscribe/unsubscribe for each event type
4. both APIs have the intention of implementing non blocking read and writes
5. the trunk API intends to allow both non blocking and blocking read/writes, but not both nor mixed for a connection
6. both APIs will have the non blocking implementation utilize the existing servlet/stream/reader/writer APIs, which will make the initial understanding of the APIs somewhat difficult, as the servlet/stream/reader/writer APIs are originally written for blocking APIs.
7. both APIs - A non blocking write will throw an IOException if a previous write has been done but not yet completed, this is as write has a void return signature, and it is impossible to return 0.
8. both APIs - A non blocking read will return 0, if no data exists.
9. both APIs will have helper methods isWriteable and isReadable to help with the state of the connection
10. both APIs will have the ability to suspend any future events, trunk by unregistering current subscriptions, sandbox by calling sleep() (or suggested suspend())

Remember, the API's have a fairly similar idea of what the underlying implementation will look like, the main difference is probably the implicit vs explicit event subscription, here are some examples

Example of implicit vs explicit write subscriptions and non blocking write error

Implicit

```
if ( event.isWriteable() ) { //implicit registration of a one time the WRITE event
  event.getHttpServletResponse().getOutputStream().write(data); //non blocking write
}
```

Explicit

```
if ( event.isWriteable() ) {
  event.getHttpServletResponse().getOutputStream().write(data); //non blocking write
  event.register(OP_WRITE); //explicit registration for the WRITE event
}
```

Inaccurate use of non blocking write API

```
if ( event.isWriteable() ) {
  event.getHttpServletResponse().getOutputStream().write(data); //non blocking write
  event.getHttpServletResponse().getOutputStream().write(data); //will throw an IOException, if previous write
wasn't complete
}
```

Example of implicit vs explicit read unsubscribe/subscribe

Implicit

```
[Thread T1] event.sleep(); //unsubscribe from READ events
...
[Thread T2] event.callback(); //results in a CALLBACK event on Thread Tx, and also implicitly subscribes to
READ events
```

Explicit

```
[Thread T1] event.unregister(event.getRegisteredOps()); //unsubscribe from ALL events OR, see next line
[Thread T1] event.unregister(OP_READ); //unsubscribe from READ events
...
[Thread T2] event.register(OP_CALLBACK); //results in a CALLBACK event on Thread Tx, OR, to subscribe to READ
events
[Thread T2] event.register(OP_READ); //results in a READ event when more data arrives
```

Display

xxx

CategoryFAQ