Asynchronous Ajax (aka Comet) using the Grizzly Comet Framework

Jeanfrancois Arcand
Senior Staff Engineer
Sun Microsystems

Agenda

- Introduction
- What is Ajax Push (aka Comet)?
- Potential Drawbacks and Pitfalls
- Grizzly Comet Technicals Details
- Demo
  - Ajax Counter
  - DWR Battleship
  - JMaki
  - ICEFaces

What is Comet Request Processing (or Ajax Push)

JGD

Comet is a programming technique that enables web servers to send data to the client without having any need for the client to request for it. It allows creation of event-driven web applications which are hosted in the browser.

What is Ajax Push (aka Comet)?

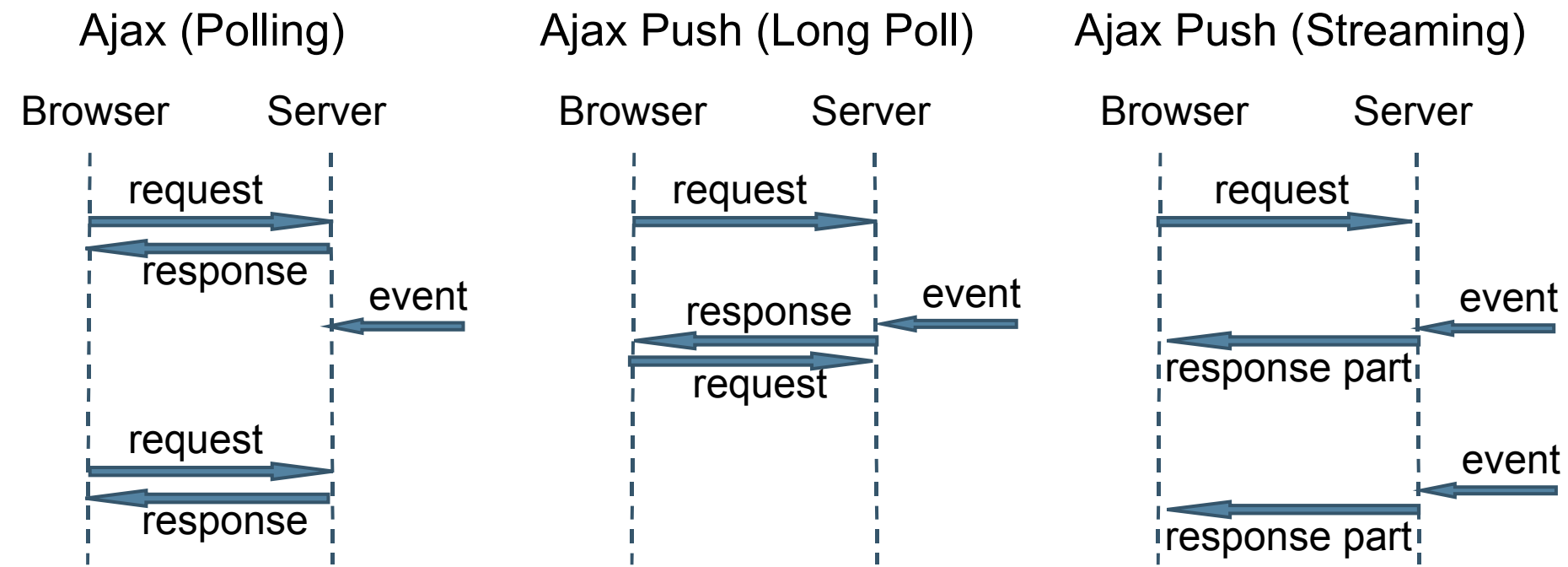
- Use it to create highly responsive, event driven applications in a browser
  - Keep clients up-to-date with data arriving or changing on the server, without frequent polling
- Pros
  - Lower latency, not dependent on polling frequency
  - Server and network do not have to deal with frequent polling requests to check for updates
- Example Applications
  - GMail and GTalk
  - Meebo
  - JotLive
  - KnowNow
  - 4homemedia.com (build on top of GlassFish's Comet)
  - Many more …

How does the “Push” to the browser works

- Deliver data over a previously opened connection
  - Always “keep a connection open”; do not respond to the initiating request until event occurred
  - Streaming is an option by sending response in multiple parts and not closing the connection in between

How does “Push” to the browser work?

Standard Ajax compared to Ajax Push options

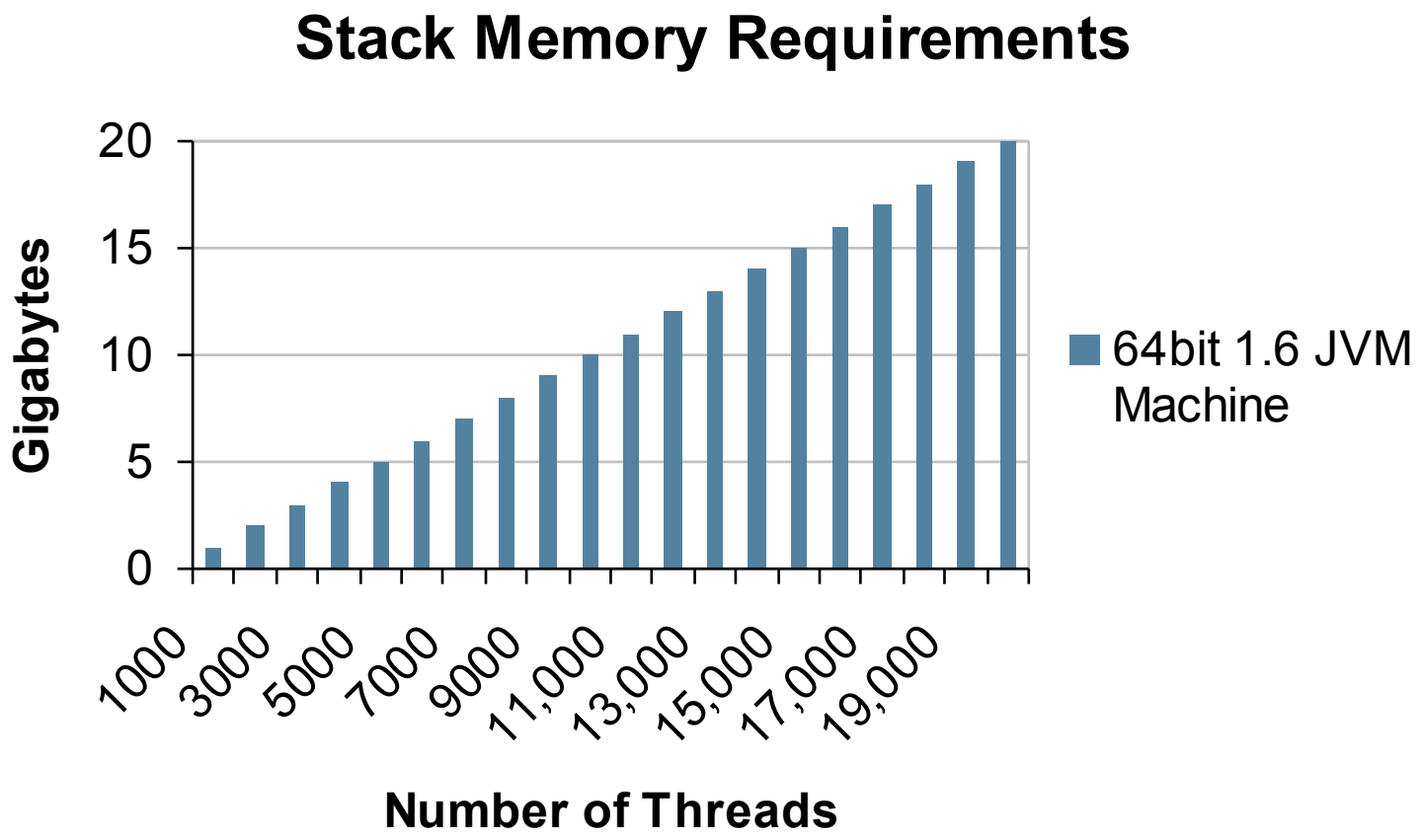


Architecture Challenge

- Using blocking, synchronous technology will result in a blocked thread for each open connection that is “waiting”
  - Every blocked thread will consume memory
  - This lowers scalability and can affect performance
  - To get the Java Virtual Machine (JVM™) to scale to 10,000 threads and up needs specific tuning and is not an efficient way of solving this
- Servlets 2.5 are an example of blocking, synchronous technology

Architecture Challenges

Affect of Blocking threads (default thread stack size)



Technology Solutions

- Use new I/O (NIO) non-blocking sockets to avoid blocking a thread per connection
- Use technology that supports asynchronous request processing
  - Release the original request thread while waiting for an event
  - May process the event/response on another thread than the original request
- Advantages
  - Number of clients is primarily limited by the number of open sockets a platform can support
  - Could have all clients (e.g. 10’000) “waiting” without any threads processing or blocked

Do Comet enabled Servers exist?

- Yes, more and more servers that support Comet request processing are available:
  - GlassFish v2, V3
  - Jetty
  - Ligthttpd
  - Grizzly WebServer
  - ...
- Today we are going to focus on Grizzly's Comet support (standalone and in GlassFish).

What is Grizzly

- Grizzly is a multi protocol (HTTP, UDP, etc.) framework that uses lower level Java NIO primitives, and provides high-performance APIs for socket communications.
- In GlassFish, Grizzly is the HTTP front end.

Comet in Grizzly

- Comet support is build on top of Grizzly Asynchronous Request Processing (ARP), a scalable implementation that doesn't hold one thread per connection, and achieve as closer as possible the performance of synchronous request processing (SRP).

Goal of Grizzly's Comet

JGD

- Hide the complexity of NIO/Asynchronous Request Processing.
- Make it available to various technologies from AJAX based client, JSF, JSP, Servlet, POJO, JavaScript to “traditional” technologies such as JMS, EJB, database, etc.
- Allow complicated scenarios but also support POJO based development.
- **Main Goal: Make it simple to use!**

Clients/Server Considerations

- To handle the “wait” for an event in Ajax Push, choose a Web Server / language that does not have to block
- Some inherently do not block
  - Traditional Web Server (like Apache, Tomcat, and GlassFish v1) are blocking.
- Continuations are another options
  - E.g. JavaScript™ technology “continuation” in the “Rhino” open source implementation

Potential Drawbacks and Pitfall

- Beware of flooding clients with too many events
  - Filters (throttles) on the client or server side? Which events can be discarded, which can't?
- Firewalls may terminate connections after a certain amount of time
  - Solution: Re-establish connection after tear-down or at certain intervals
- The HTTP 1.1 specification suggests a limit of 2 simultaneous connections from a client to the same host
  - Some use a separate host name for the “Push” connection
- In security terms the attack surface is very similar to standard Ajax applications, for denial of service (DoS) the “wait” is a consideration
- Possible lost of data when the connection is closed
  - Real time updates can still occurs when the application goes offline or during re-connection.

Potential Drawbacks and Pitfall (Cont')

- HTTP Streaming is more challenging
  - Portability issue to different browsers and XMLHttpRequest (XHR)
    - IE, for example, does not make data available until connection close
    - Use IFrames instead for portability
  - With streaming data will accumulate, release memory regularly
  - Primitive proxies may buffer data in a way that interferes with streaming
- Possible lost of data when the connection is closed
  - Real time updates can still occurs when the application goes offline.

Comet support in Grizzly: Details

- Implemented on top of the Grizzly Asynchronous Request Processing (heavily used in OpenESB HTTP BC Components).
- Hide the complexity of NIO/Asynchronous Request Processing.
- Support clean and ssl connection.
- Make it available to JSF, JSP, Servlet, POJO, JavaScript (Phobos)
- **Main Goal: Make it simple!**

Grizzly Comet supports:

- Asynchronous Content Handlers
  - Allow handling asynchronous read and write
- Suspendable Requests
  - suspend/resume requests/response
- Container managed server Push
  - push data from one connection to another

Grizzly Comet Framework details

- Grizzly offer three solutions:
  - **Grizzly Comet**: Comet framework for Servlet/JSP deployed in GlassFish or Jetty. Support suspendable request, asynchronous content handler and container managed server push.
  - **Grizzlet**: simple POJO object deployed on top of Grizzly WebServer. Support suspendable request and container managed server push.
  - **Grizzly Continuation**: Simple API for suspending/resuming a connection. Support suspendable request.

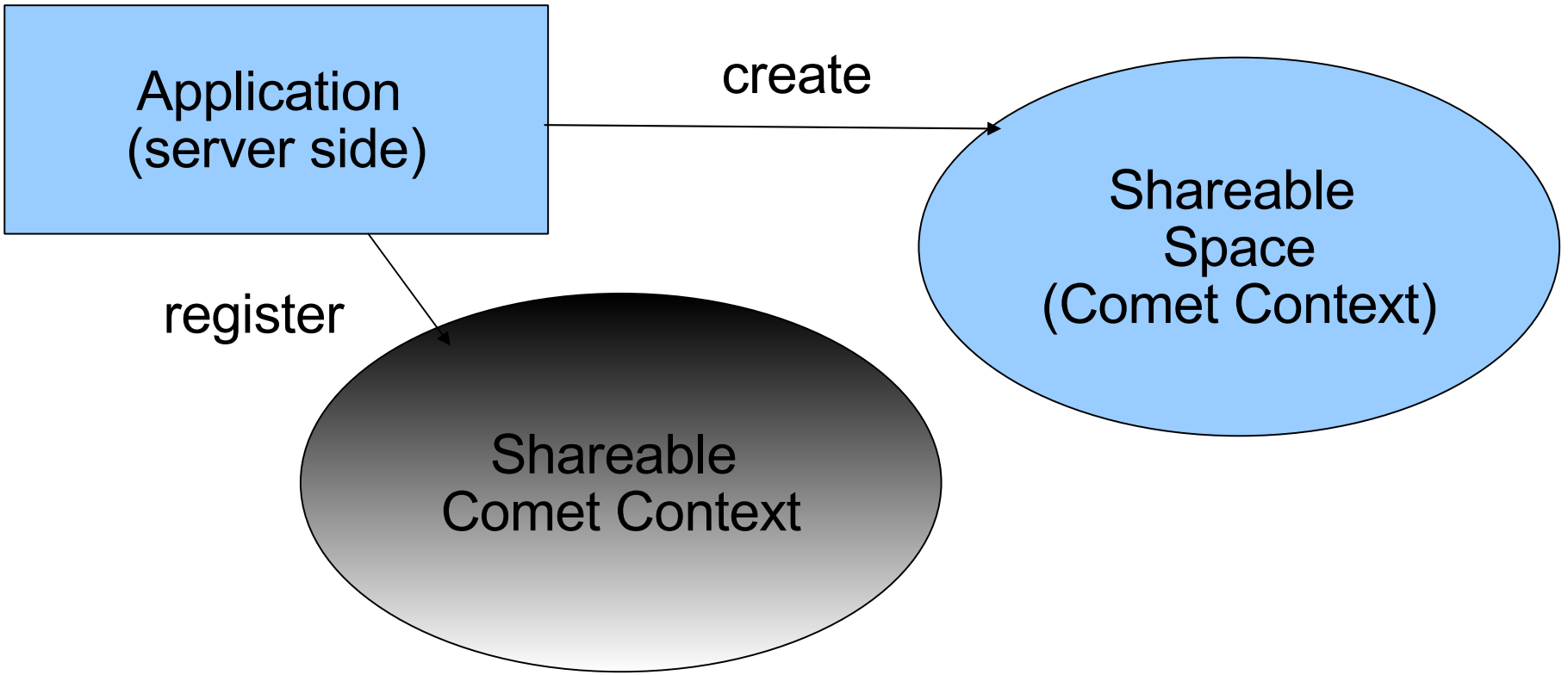
First, let's start with Grizzly Comet Framework

JGD

- CometContext: A shareable “space” to which applications may subscribe and from which they receive updates.
- CometEvent: An object containing the state of the Comet Context (content updated, client connected/disconnected, etc.).
- CometHandler: The interface an application must implement in order to be part of one or several CometContext.

Step 1: Create new Comet Context or register to existing one

JGD



Server side: At Startup

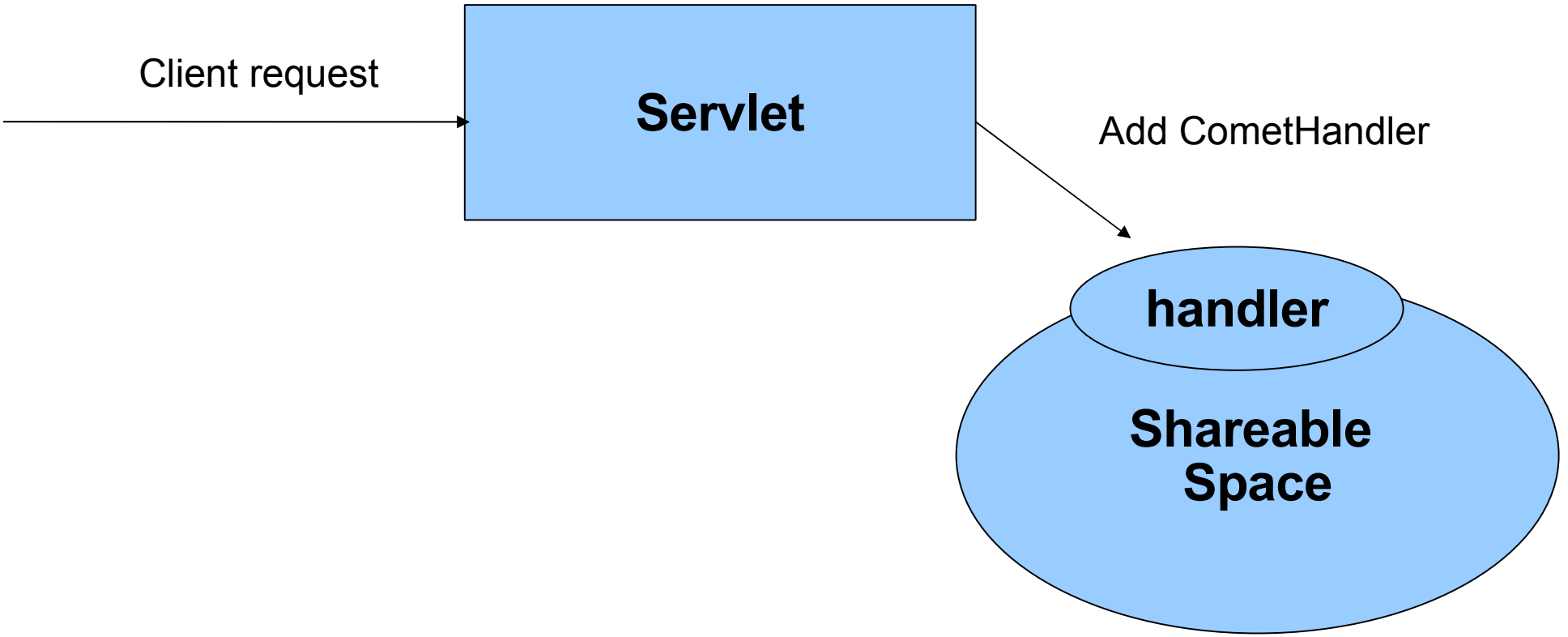
JGD

```
public void init(ServletConfig config){
    ServletContext context =
                    config.getServletContext();
    contextPath = context.getContextPath() + "/comet";

    CometEngine engine = CometEngine.getEngine();
    CometContext cometContext =
    engine.register(contextPath);
```

- Ex: Requests to “ /myApp/Comet” will be considered as Comet request.

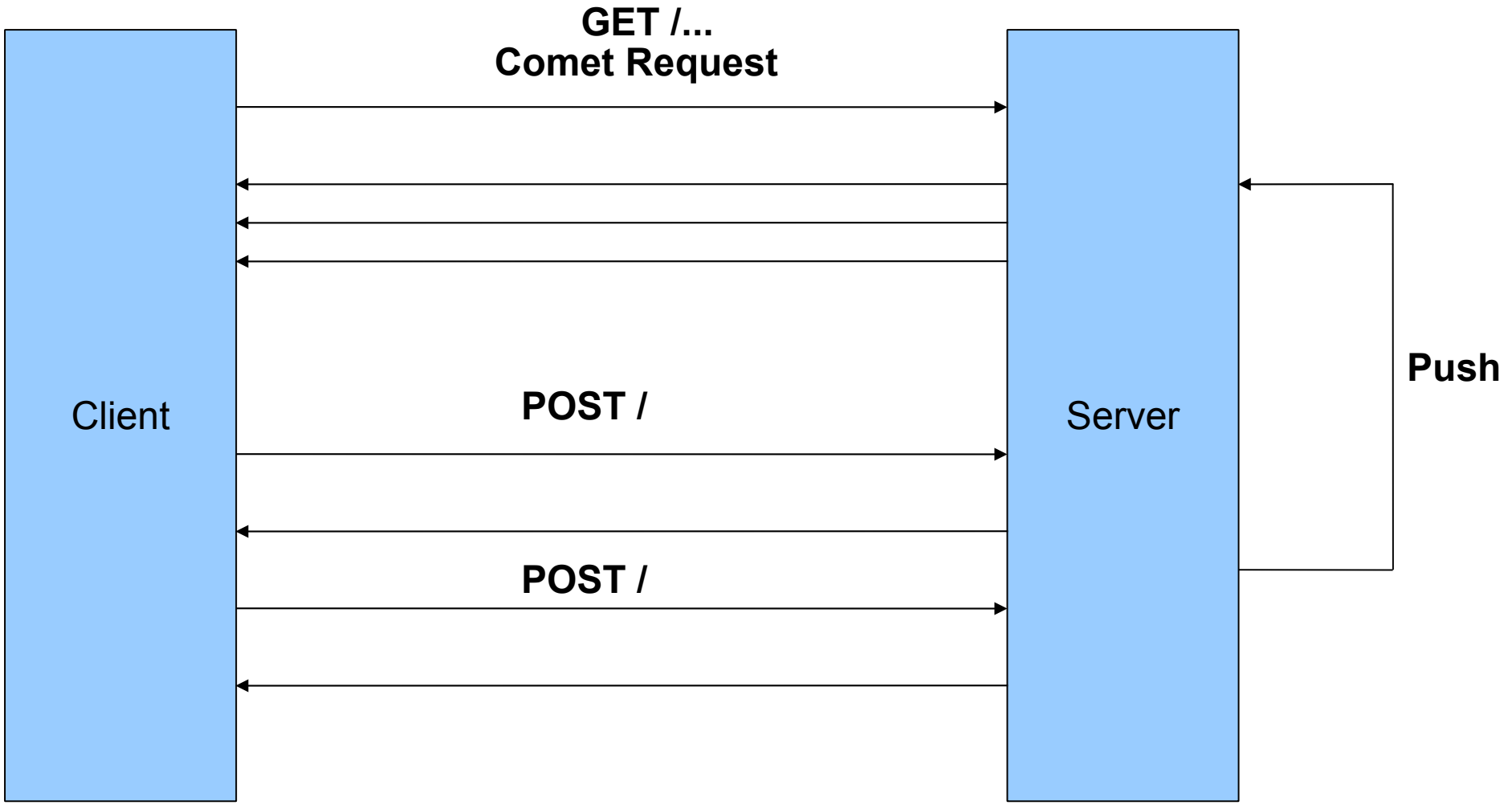
Step 2: Create one Handler per long polled connection (optional)

JGD



Remember you can only open two connections from the Browser.

JGD



Step 3: Open a connection

JGD

```
var counter = {
    'poll' : function() {
        new Ajax.Request('/myApp/comet', {
            method : 'GET',
            onSuccess : counter.update
        });
    },
```

- This connection enables Comet Request Processing (long lived connection).

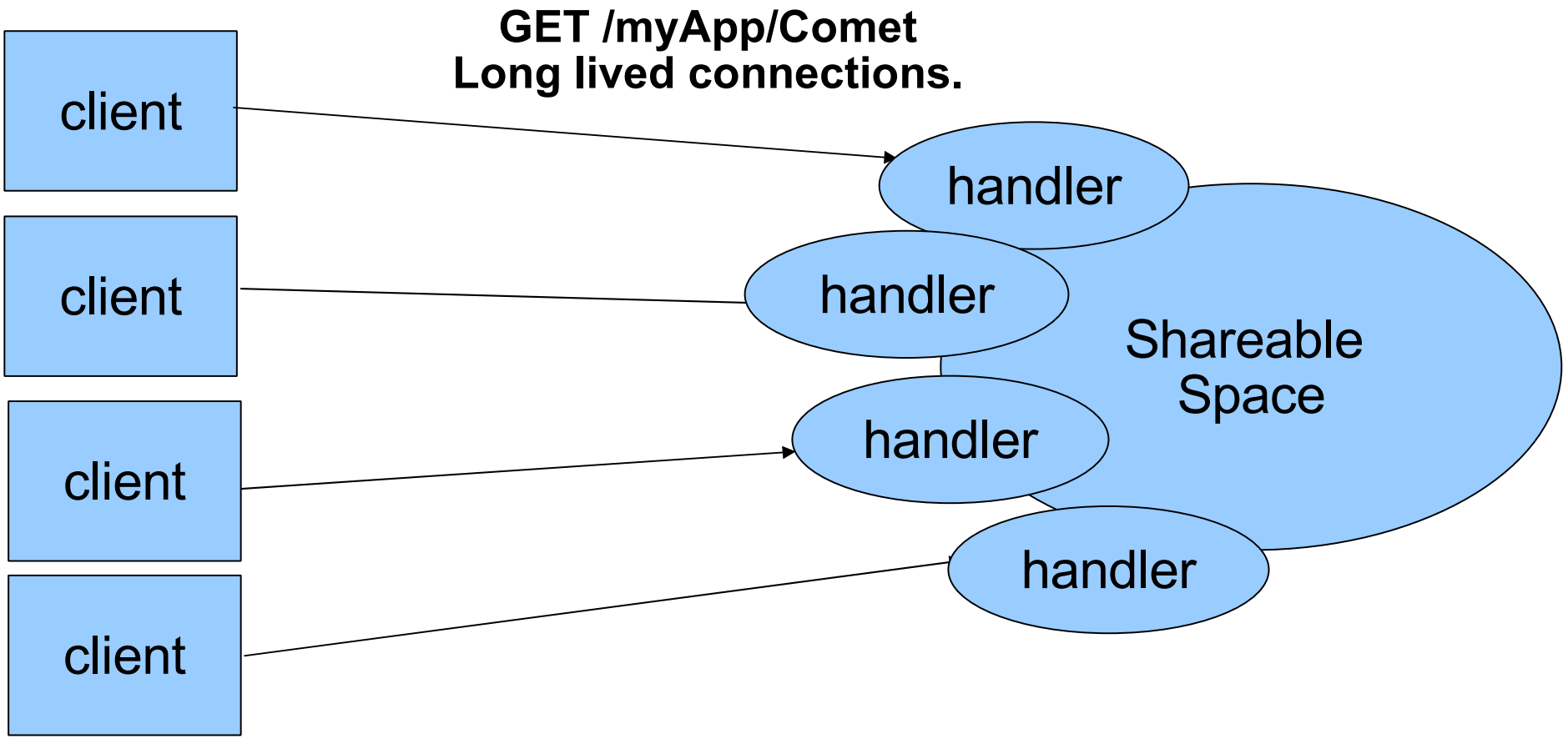
Server side: CometHandler.onEvent()

JGD

```
private class CounterHandler
    implements CometHandler<HttpServletResponse>{

    public void onEvent(CometEvent event)  {
        PrintWriter writer = response.getWriter();
        writer.
          write("<text>Pushing data: success</text>");
        writer.flush();
```

Now all connections are parked.

JGD



Server side: Servlet.doGet(..)

JGD

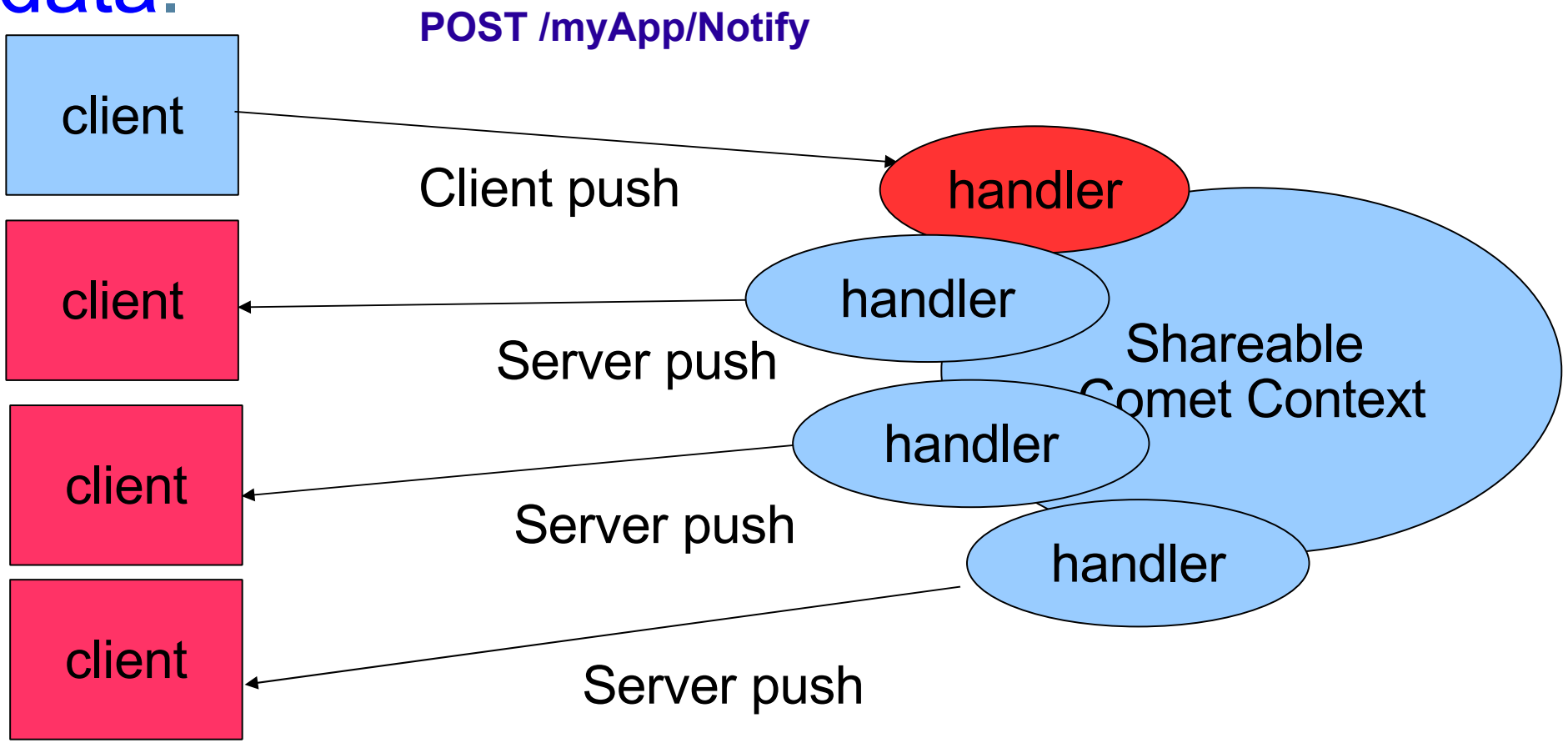
```
protected void doGet(HttpServletRequest req,
        HttpServletResponse res){

    CounterHandler handler = new CounterHandler();
    handler.attach(res);
    CometEngine engine = CometEngine.getEngine();
    CometContext context =
            engine.getCometContext(contextPath);
    context.addCometHandler(handler);
```

Client side: OnClick


```
var counter = {
...
    'increment' : function() {
        new Ajax.Request('/myApp/Notify', {
            method : 'POST'
        });
    },
```

As one client pushes data, all other clients are updated without polling for data.

JGD

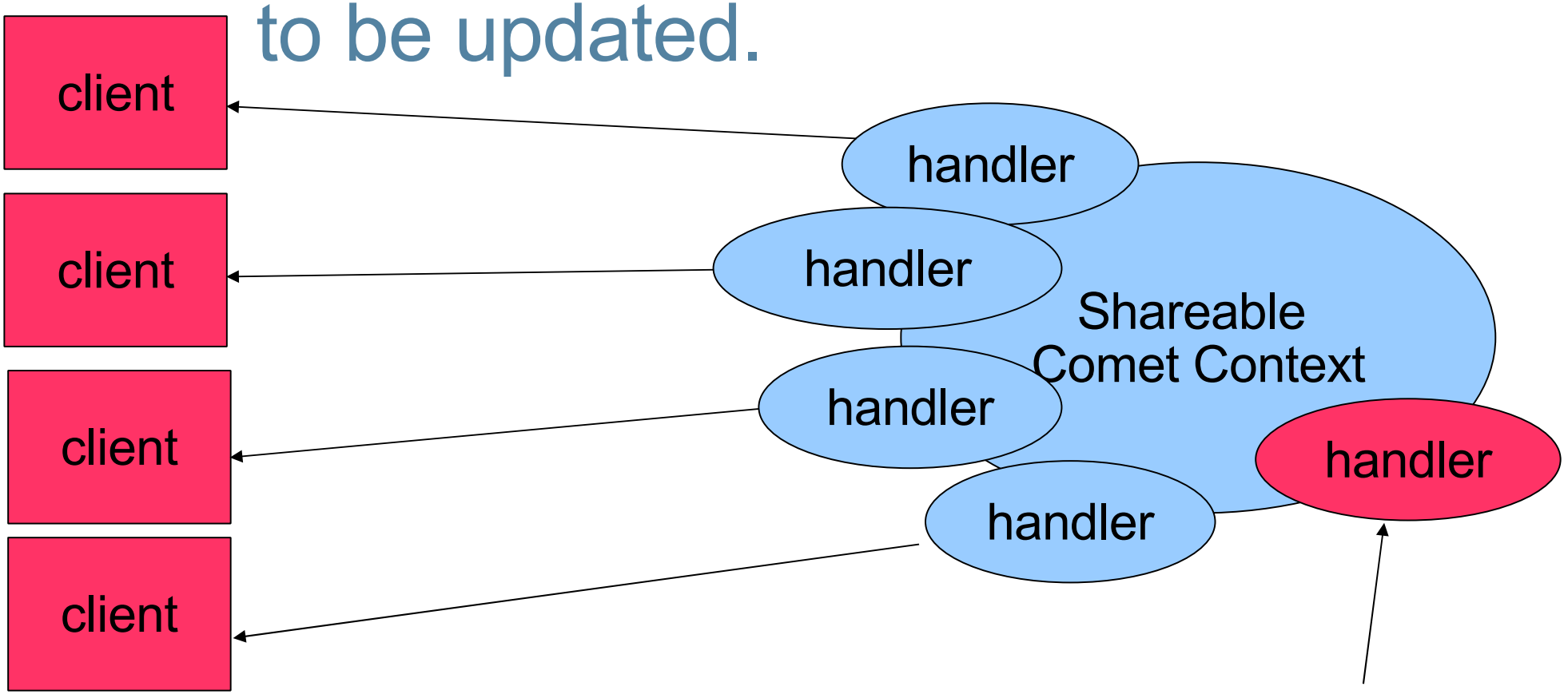


Server side: doPost(..)


```
protected void doPost(HttpServletRequest
        req,HttpServletResponse res) {

    CometEngine engine =
       CometEngine.getEngine();
    CometContext<?> context =

  engine.getCometContext(contextPath);
   context.notify("<text>Hello</text>");
```

The push does not have to come from a TCP client. *Any* change to the CometContext causes the clients to be updated.

JGD



What is a Grizzlet

- Simple interface build on top of the Grizzly Comet Framework:
  - public void onRequest(AsyncConnection asyncConnection)
  - public void onPush(AsyncConnection asyncConnection)
- Implement that interface, deploy your POJO in Grizzly Web Server. Boom it works!

Grizzly Continuation

- Simple API to resume/suspend requests:

```
GrizzlyContinuation continuation =
                    Continuation.getContinuation();
continuation.suspend(timeout);

....

continuation.resume();
```

- Used in Servlet/JSP.

Another simple example!
Four simple steps

- Adding Comet support to an existing application is simple (no big re-factoring).
- The next four slides will describe how to do it.
- More details can be found:

  http://weblogs.java.net/blog/jfarcand/

Step 1 - Registration

- First, register to a new or existing Comet context. The context are usually created using a context path (but not mandatory)

```
CometEngine cometEngine =
                CometEngine.getEngine();
CometContext context =
    cometEngine.register(contextPath);
```

Step 2- Define your Comet handler

- The Comet handler interface is simple:

```
public void attach(E attachment);
public void onEvent(CometEvent event)
public void onInitialize(CometEvent event)
public void onTerminate(CometEvent event)
public void onInterrupt(CometEvent event)
```

Step 3- Adding Comet handler to the context

- Next, instantiate your Comet request handler and add it to the context

```
MyCometHandler handler =
                    new MyCometHandler();

cometContext.addCometHandler(handler)
```

Step 4 – Advertise changes.

- Once the Comet handler has been added to the Context, you can start pushing events. You can notify the context when the client push data or when something external has changed (ex: a database):

```
cometContext.notify("Comet is cool");
```

Comet handler example.

- For a Servlet, you will most probably implement the CometHandler using the HttpServletResponse:

```
public class CometResponseHandler implements
        CometHandler{

    public void attach(HttpServletResponse
                                httpServletResponse){
        this.httpServletResponse =
                    httpServletResponse;
```

Comet handler example.

- And the onEvent method will most probably looks like:

```
public void onEvent(CometEvent event) throws IOException{

    try{
        PrintWriter printWriter = httpServletResponse.getWriter();
        printWriter.println(event.attachment());
        printWriter.flush();
    } catch (Throwable t){
       t.printStackTrace();
    }
```

Demos

- You can download the demo from:

  jMaki:

  http://weblogs.java.net/blog/jfarcand/archive/2007/03/jmaki_comet_orb_1.html

  IceFaces:

  http://weblogs.java.net/blog/jfarcand/archive/2007/10/the_arctic_griz.html

  DWR:

  http://weblogs.java.net/blog/jfarcand/archive/2007/11/grizzly_attack.html

  JavaScripts:

  http://weblogs.java.net/blog/jfarcand/archive/2007/06/new_adventures_3.html

Thanks to!

- Andreas Egloff for his Comet Introduction slides!

Help Improve This Presentation

If you want to contribute, keep attribution and maintain license. We would also appreciate notifying us of the changes.

More related presentations are kept at the Presentations page of the GlassFish Wiki.

http://www.glassfishwiki.org/gfwiki/Wiki.jsp?page=Presentations

jeanfrancois.arcand@sun.com