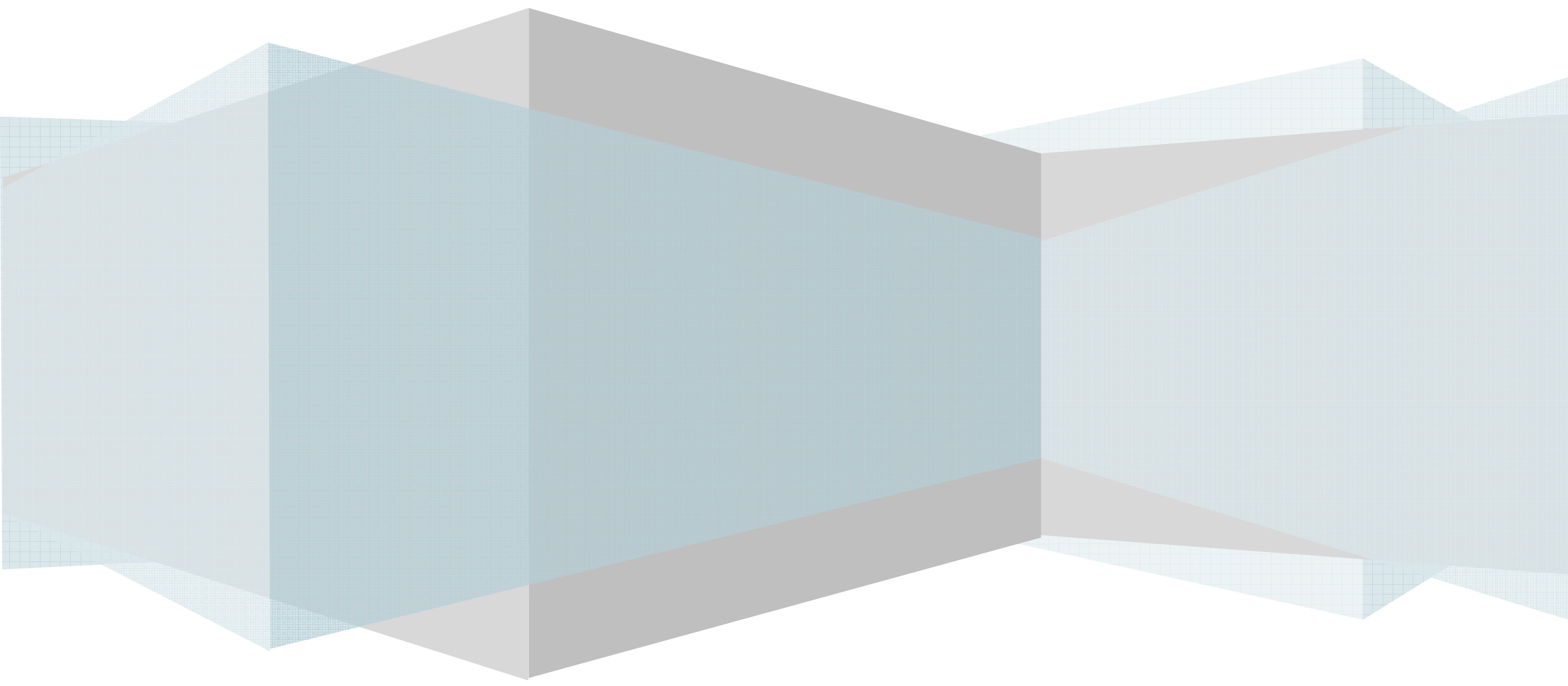
rjax.net

RJAX Documentation

Version 1.0 - May 2007

Arnaud RETHORE (arnaud@rjax.net)



Project overview

RJAX (Reverse aJAX) is a solution for pushing web server data to a browser without having any need for the client to request it.

The aim is to avoid having the browser continuously polling data from the server. This is what you would do if you were using standard AJAX techniques.

The solution maintains an open connection between the browser and the server using a dedicated applet in the client page. This allows the server to execute JavaScript commands directly on the client.

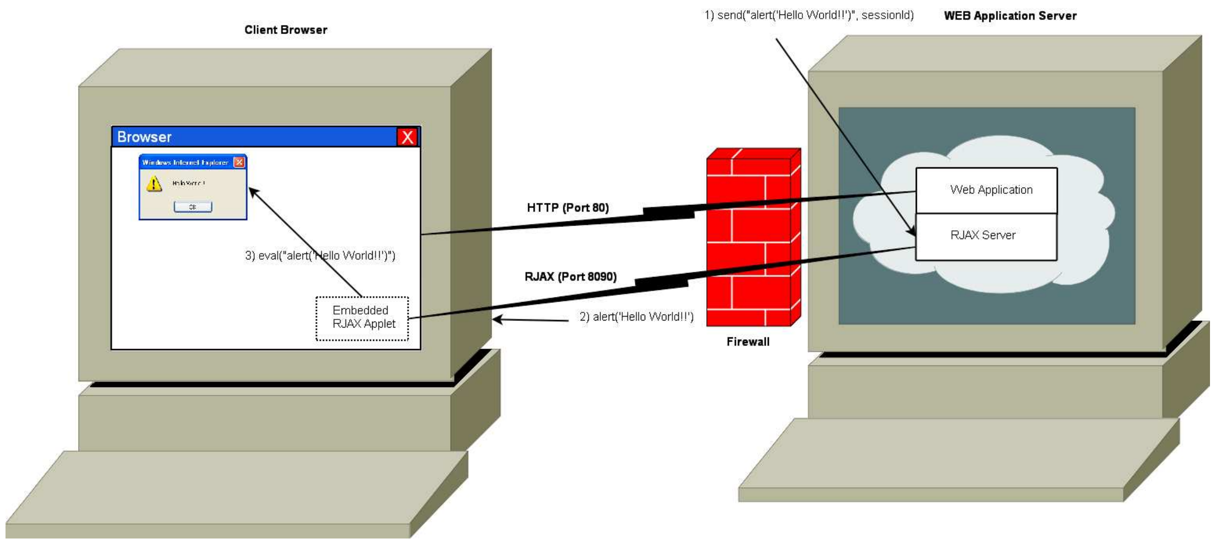
A possible downside of such a solution is that some firewalls are typically configured to drop connections that have been open for too long. RJAX will address this issue by implementing a mechanism for dropping and recreating connections on a (configurable) regular basis.

NOTE: This mechanism for dropping and recreating connections is not implemented the version 1.0

The project consists in two pieces of code: The RJAX client applet and the RJAX server that can be embedded in a Java web application (war file).

The RJAX server is started when the web application gets deployed and stopped when the web application gets un-deployed. The server listens to a port different from the web server. A client is identified by a unique ID (Typically its J2EE session ID). To send a JavaScript command to a client, using the RJAX server, you just need to know its ID.

The RJAX client applet is hidden in the page. It maintains the connection with the server and signals the server when the applet is loaded or unloaded. Its role is to execute JavaScript commands received from the server. To do so, it relies on the use of **netscape.javascript.JSObject** which works with most JavaScript enabled browser (Netscape, Firefox, IE...).


Client Browser
1) send("alert('Hello World!!')", sessionId)
WEB Application Server
Browser
Windows Internet Explorer
3) eval("alert('Hello World!!')")
Embedded RJAX Applet
HTTP (Port 80)
RJAX (Port 8090)
2) alert('Hello World!!')
Firewall
Web Application
RJAX Server

RJAX is an open source SourceForge project available at: http://sourceforge.net/projects/rjax.

NOTE: RJAX has been developed using Java 1.5 jdk. This should not be to complicated to rewrite it for Java 1.4 (I'm not going to do this anyway).

Building an application using RJAX

As described previously, in order to add RJAX features to your application, you need 2 things:

- An RJAX applet embedded in your HTML page
- An RJAX server embedded in your web application

Setting up the RJAX Applet

RJAX applet Jar file

Copy the rjaxApplet.jar file in the folder of your web application containing the applets. For example: in the /applets folder (Same level as /WEB-INF and /META-INF folders).

Applet tag

There are few lines of code needed to embed the RJAX applet in your page. Place the following code inside the <body> tag of your HTML page:

```
1   <applet
2   codebase = "/rjax/applets"
3   code     = "rjax.applet.RjaxApplet.class"
4   archive  = "rjaxApplet.jar"
5   name     = "RjaxApplet"
6   width    = 0
7   height   = 0
8   hspace   = 0
9   vspace   = 0
10  align    = top
11  MAYSCRIPT
12  style = "display: none"
13  >
14        <param name="rjaxServerIP" value="127.0.0.1">
15        <param name="rjaxServerPort" value="8090">
16        <param name="rjaxSessionId"
17  value="216212F4B0BFC849186271B7EB127DA7">
18   </applet>
```

Explanation:

Lines 1, 13 and 18: Open and close the applet tag

Line 2: codebase attribute: Application folder where your RJAX applet JAR file is stored in your web application.

Line 3: code attribute: Do not change this value. This is fully qualified name of the RJAX applet class

Line 4: archive attribute: Do not change this value. This is the JAR file name (in your web application).

Line 5: name attribute. This is the RJAX applet name.

Lines 6, 7, 8, 9: width height, hspace, vspace attributes: We do not want this applet to be displayed in the page. So we reduce its size to 0.

Line 10: align attribute: top is fine. This is not important since the applet is not visible.

Line 11: MAYSCRIPT: Do not change this attribute. This tells the browser that the applet may run some script (this is what we want the applet to do!).

Line 12: Style attribute. We set it to “display: none” for Internet Explorer only. We do not want this line with Firefox. If this line is not used with IE, the applet is visible (you can see one white pixel). If this line is used with Firefox, the applet will not run.

Lines 14, 15, 16: Applet parameters:

- rjaxServerIP: IP address of the machine running the RJAX server
- rjaxServerPort: The RJAX server port number
- rjaxSessionId: A unique ID identifying this RJAX session. Typically, the J2EE web session ID.

Using a JSP

As you can see from the previous example, a lot of values should be dynamically set up. To do so, you can generate your page using a JSP. To get the previous lines generated dynamically according to your server and client settings, insert the following JSP lines:

```
1  <%
2  String ua = request.getHeader("User-Agent");
3  boolean isMSIE = ( ua != null && ua.indexOf( "MSIE" ) != -1 );
4  %>
5  <applet
6  codebase = "<%= request.getContextPath() %>/applets"
7  code     = "rjax.applet.RjaxApplet.class"
8  archive  = "rjaxApplet.jar"
9  name     = "RjaxApplet"
10 width    = 0
11 height   = 0
12 hspace   = 0
13 vspace   = 0
14 align    = top
15 MAYSCRIPT
16 <% if (isMSIE) { %>
17 style = "display: none"
18 <% }              %>
19 >
20 <param name="rjaxServerIP" value="<%= request.getLocalAddr()%>">
21 <param name="rjaxServerPort" value="<%=
22 config.getServletContext().getInitParameter("rjax.port") %>">
23 <param name="rjaxSessionId" value="<%= session.getId() %>">
24 </applet>
```

Explanation:

Lines 1, 2, 3, 4: Tests if the browser is IE or not.

Line 6: request.getContextPath() returns the context path of your application (this corresponds to the path attribute of the <Context> tag of your application).

Lines 16, 17, 18: add the style attribute set to "display: none" if the browser is IE.

Line 20: request.getLocalAddr() returns the server IP address

Lines 21, 22: config.getServletContext().getInitParameter("rjax.port"): Returns the RJAX server port number from the web.xml file of your web application. This assumes that the following entries exists in your web.xml file:

```
 <context-param>
       <param-name>rjax.port</param-name>
       <param-value>8090</param-value>
</context-param>
```

Line 23: session.getId() returns the Java web session ID.

NOTE: The next version of RJAX should provide a JSP tag library to help you with inserting the RJAX applet in your page.

Setting up the RJAX Server

RJAX applet Jar file

Copy the rjaxServer.jar file in the /WEB-INF/lib folder of the web application.

Log messages

RJAX relies on Log4j and Apache commons-logging APIs to log messages while running.

In order to see the messages logged by the RJAX server, you need to add the following 3rd party APIs jar files in your web application class path:

- Log4j available at: http://logging.apache.org/log4j
- Apache commons-logging available at: http://jakarta.apache.org/commons/logging/

If Log4j is not already configured for your web application, the rjaxServer.jar file has a utility Class to help you with this task. Just add the following lines in your application web.xml file:

```
<context-param>
    <param-name>log4j.init.file</param-name>
    <param-value>WEB-INF/log4j.properties</param-value>
</context-param>
<listener>
    <listener-class>rjax.server.util.Log4jSetup</listener-class>
</listener>
```

This assumes that you also have a file named log4j.properties in your WEB-INF folder. This file contains the properties needed to configure log4j for the RJAX server.

Here is an example of such a file:

```
#########################################################
# LOG4J                                                 #
# See http://logging.apache.org/log4j/docs/manual.html   #
#########################################################
# Appenders for development
log4j.rootLogger=ERROR, A1
log4j.logger.rjax=DEBUG

# A1: Console
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d %-5p %c - %m%n
```

Starting and stopping the RJAX server

The idea here is to get your web application start the RJAX server when it gets deployed and stop the RJAX server when it gets un-deployed.

The rjaxServer.jar file has a utility Class to help you with this task. Just add the following lines in your application web.xml file:

```
<context-param>
   <param-name>rjax.port</param-name>
   <param-value>8090</param-value>
</context-param>
<context-param>
   <param-name>rjax.backlog</param-name>
   <param-value>10</param-value>
</context-param>
 <listener>
   <listener-class>rjax.server.util.RjaxServerManager</listener-
class>
 </listener>
```

The first parameter, rjax.port, tells the helper class which port the RJAX server should be listening to.

The second parameter, rjax.backlog, tells the RJAX server the number of TCP/IP connection that can wait to be processed.

Start message

If you have correctly followed the previous steps, you should get something like this in your logs when your web application starts:

```
2007-05-03 22:22:57,578 INFO  rjax.server.util.RjaxServerManager - Starting
RJAX server on port 8090 with backlog set to 10
2007-05-03 22:22:57,578 INFO  rjax.server.RjaxServer -
 ____        _      _     __  __
|  _ \      | |    / \    \ \/ /
| |_) |  _  | |   / _ \    \  /
|  _ <  | |_| |  / ___ \   /  \
|_| \_\  \___/  /_/   \_\ /_/\_\

Server version: 1.0
```

Stop message

... and you should get this lines when your web application stops:

```
2007-05-04 00:00:51,750 INFO  rjax.server.util.RjaxServerManager - Stopping
RJAX server
2007-05-04 00:00:51,750 DEBUG rjax.server.RjaxServer - Stopping RJAX server
```

Using RJAX

Executing a JavaScript command remotely

Now that everything is configured, you can send some JavaScript commands to your client page using the following lines of code:

```
RjaxServer rjaxServer = ((RjaxServer)
session.getServletContext().getAttribute("RjaxServer"));

rjaxServer.send("Some JavaScript command", sessionId);
```

Explanation:

1st line of code: The rjax.server.util.RjaxServerManager utility class (seen previously) has started an instance of the RjaxServer class and stored the object in the Servlet context using the attribute id "RjaxServer", so that you can retrieve it from almost everywhere in your web application.

2nd line of code: call the send method on the RJAX server passing the JavaScript command as a string and the identifier of the client page (its web session Id for example). This assumes that the client page containing the RJAX applet has already been loaded.

Knowing if a client is connected

Use the following code to know whether the client is connected or not:

```
rjaxServer.isConnectionEstablished (sessionId)
```

Getting a list of connected clients

Use the following code (useful to implement a broadcast mechanism):

```
for (String sid : rjaxServer.getRegisteredSessionIds()) {your code here}
```

Known issues

Apart the features missing in this release, here is the list of known issues:

1) Bad JavaScript command and Firefox:

When a malformed command is sent to a Firefox browser, the applet crashes and stops running. This does not happen with IE. Anyway, this only occurs with commands having a bad syntax and because commands are sent by the server, and since you know what you send, this should not happen. This issue should be fixed in the next release.