Joël Cancela Vaz
Nikita Rousseau

17/04/2018

Mini Project – Java RMI Program

Introduction

Syndication is a form of communication in which content is made available from one entity to other entities. The basic idea is to broadcast information to a bunch of subscribed entities. Thus, as a service provider, we must provide information about a resource and its modifications. Clients of such a service will, for example, download a file that contains information and history of a resource over the time. In addition, clients will have to parse and extract information from the downloaded file.

There are several ways to achieve this goal. We will deal with the "HTTP Streaming" and "Web Hook" techniques. In this project, we have implemented the solution using the "Web Hook" method. In the next section we will try to draft a "poor implementation" of "HTTP Streaming" using Java RMI.

HTTP Streaming Protocol Over JAVA RMI

The process of streaming is simple: the client tells the server that he is ready to receive contents on a specific port (the client has subscribed to the server). Then, when the server has contents to deliver, it will send the contents as chunks to the client on its opened port. The client will reassemble the chunks to have the complete file. We propose two approaches:

a) Fake streaming: download chunk by chunk with RMI

From the client, we would have to call the remote server on the resource and ask the size of the resource as well as the maximum chunk size. Then, on the client, we will query the remote server for each chuck until we have reached the size of the resource to fetch (loop and query until the download is complete).

This method is not a real streaming method, because each chunk download is managed by the client that initiated the file transmission from the server. The issue is that the streaming protocol ensure that the server is free to send data to the client until this one has unsubscribed (or is no more reachable), not upon explicit requests.

b) Callback

Another more convenient approach would be using the Hollywood principle "don't call us, we will call you": the client will initiate the streaming by sharing a callback method with the remote server. When the server is ready to transmit, then it will use the callback method. In this scenario, the client and server roles are exchanged. This solution can be implemented but RMI is not designed for this kind of communication. Moreover, in this configuration, we are closer to a Web Hook approach and so, is not suitable for this project.

In both scenarios, we are not able to properly "keep-alive" the connection between the client and the server. This is the main issue with RMI.

Joël Cancela Vaz
Nikita Rousseau

17/04/2018

Why RMI is a bad idea

RMI is working through a RPC approach. Since, **RPC is a request–response protocol**, it is impossible to have a real streaming protocol using RMI. The underlying concept of streaming is to not awaiting a single response from a query, but being ready to receive **responses** until communication end. HTTP Streaming does not imply that the client explicitly respond to the server at each communication transmission. Moreover, streaming with RMI implies that the client is also a server ; since the project cannot be implemented with a client handling both roles, http streaming over RMI implementation is impossible.

Conclusion

As we have exposed above, streaming over RMI is a bad idea since RMI is the bottleneck in that case.

In complement, Java RMI is marked as a deprecated technology. As an alternative, we would suggest using "SIMON" that is also relying on RPC calls, but with new Java interfaces (http://dev.root1.de/projects/simon/wiki). Note that even with the SIMON framework, the problem would be the same: **RPC IS NOT DESIGNED for HTTP Streaming.**