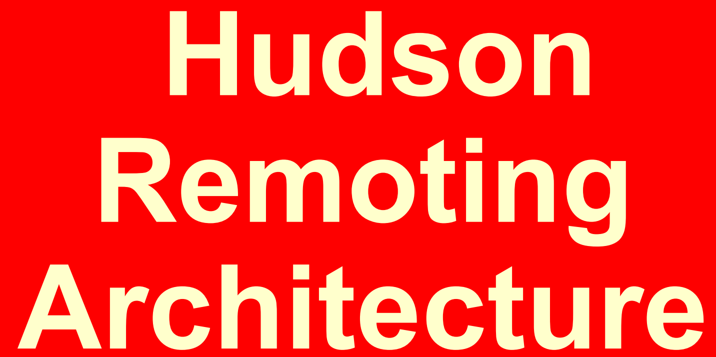


Hudson Remoting Architecture

ORACLE

Winston Prakash

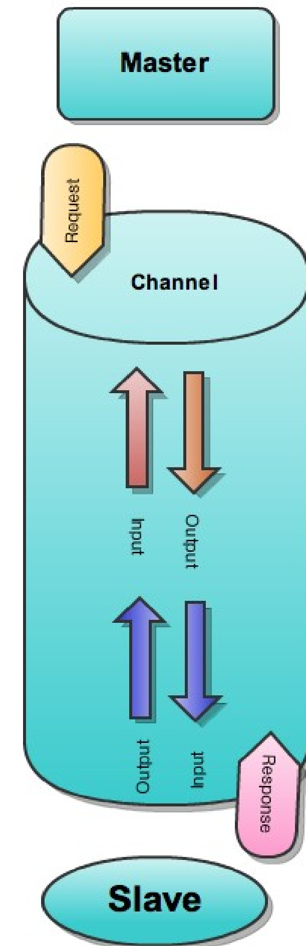
Hudson Remoting Architecture

Hudson is an distributed execution platform. The master can send closures to remote machines, then get the result back when that closure finishes computation. This mechanism is called Hudson remoting.

Callable
An interface that is implemented to create the closure and understood by hudson remoting mechanism.

Channel
A dispatching mechanism of closures from master to slaves. Encapsulates the socket I/O stream of master and slave.

Request/Response
Envelopes that carries the closure and the computed result to and fro from master and slave



Objects constitute Remoting

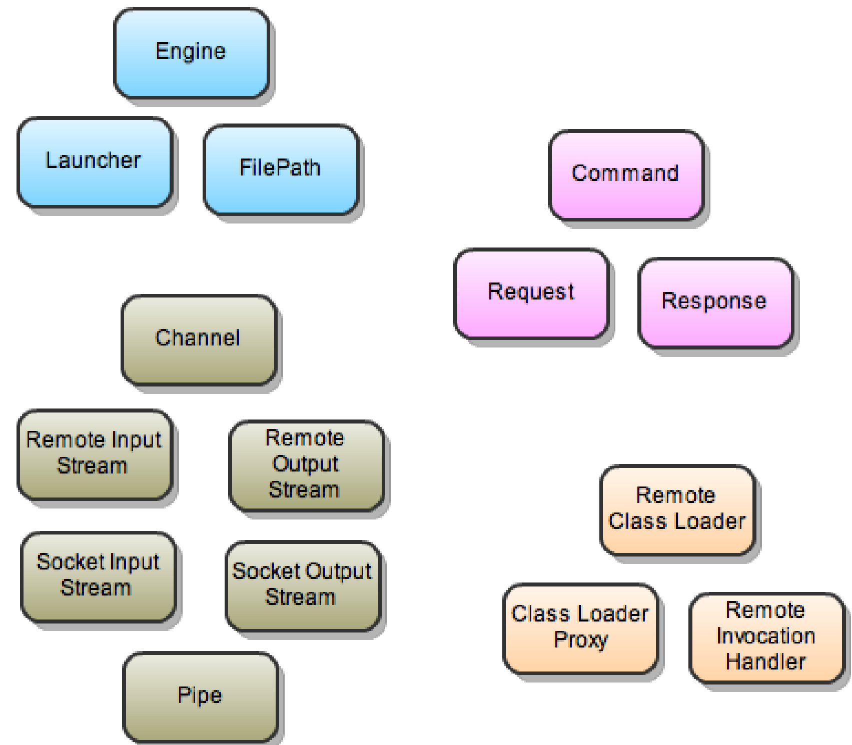
Engine is the main Slave agent object that proactively connects to the Hudson master. Once the connection is established, a **Channel** which represents the peer communication channel is created.

Pipe represents the piping between remote callable and local program to talk to each other.

Remote Input/OutputStream and streams sent over the remote channel so that remote callable can write to local I/O.

Command is a one-way command send to the remote slave agent and executed there. A **Request** on the other hand is a command that has an associated **Response** command. Once the Remote Agent gets the request, executes the command, encapsulate the result in a response and send it back to the master.

Hudson has several key abstractions such as **Launcher** and **FilePath** to make the remoting seamless for plugin developers. FilePath works on a local file or remote file hiding details from the developers. Same way Launcher launches the process locally or remotely depending on the available executor.

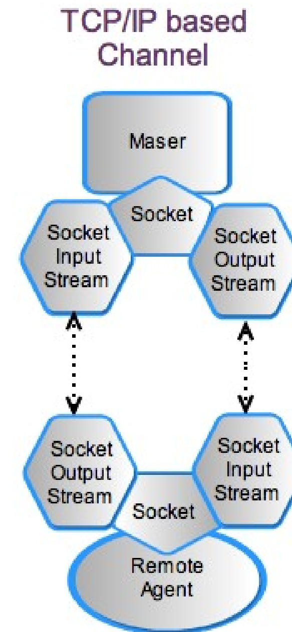


Remote Agent Connection

When a remote agent establishes a connection with master, a communication channel is created between the master and slave. The communication channel is made of four I/O stream. The I/O stream used are based on the type of connection. There are two types of connection

TCP connection

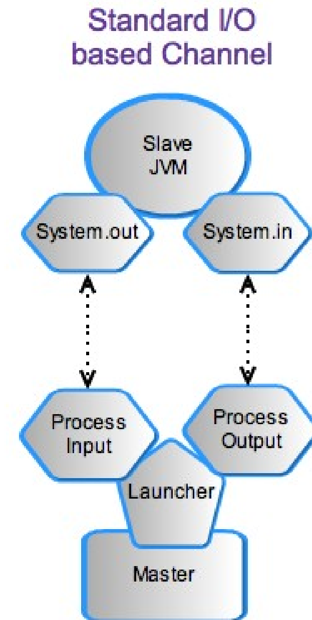
In this case the I/O stream is the socket Input/Output Stream created from the socket connection made by the Slave agent to the master. The TCP connection can be made from a fixed port number known to the Slave Agent or a JNLP TCP connection created by a Slave Agent that was started by a JNLP launcher. In case of JNLP TCP connection, the agent communicates with master via HTP port to get the socket port.



Standard Input/Output connection

This is a managed connection. The slave creates the Channel with the standard input and output of the JVM (System.out and System.in). To make sure plugin by mistake does not writes to the System.out, System.out is redirected to System.err.

The master creates the channel using the Standard Input and Output of the process through which it started the slave agent locally or remotely with SSH.



Obtaining Socket port for JNLP TCP Connection

The JNLP Slave Agent is started by obtaining the JNLP from Master and then running it in the Slave machine. When the Slave Agent is started, a secret key and URL of the Master is passed as parameter.

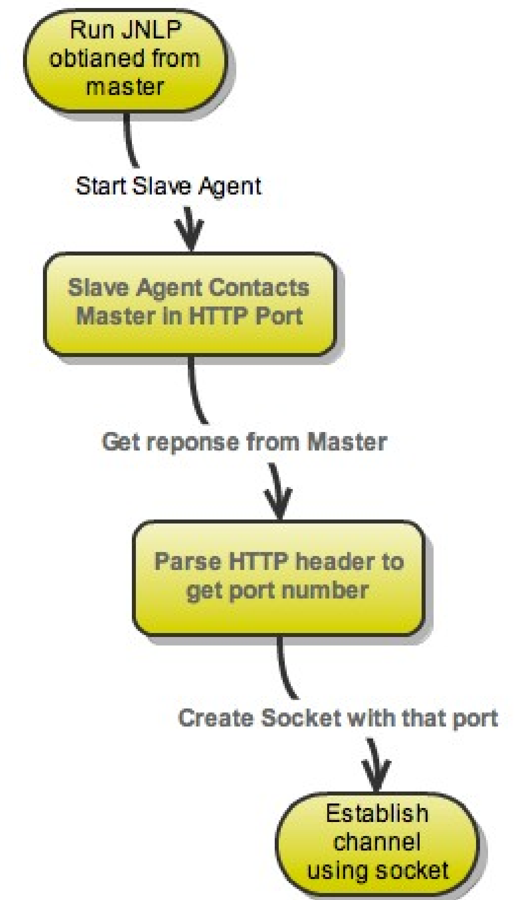
Slave Agent contacts the master in the HTTP port and issue

`GET tcpSlaveAgentListener`

In the master, *`tcpSlaveAgentListener`* is resolved via Hudson Model Object getTarget().

The `index.jelly` at *`hudson/TcpSlaveAgentListener`* is send back to the Slave Agent. `index.jelly` sets the header “X-Hudson-JNLP-Port” to the port on which the `ServerSocket` is listening for connection request from Slave Agent.

Once the port is obtained from the header, Slave Agent request socket connection in that port and establishes the channel for remoting.



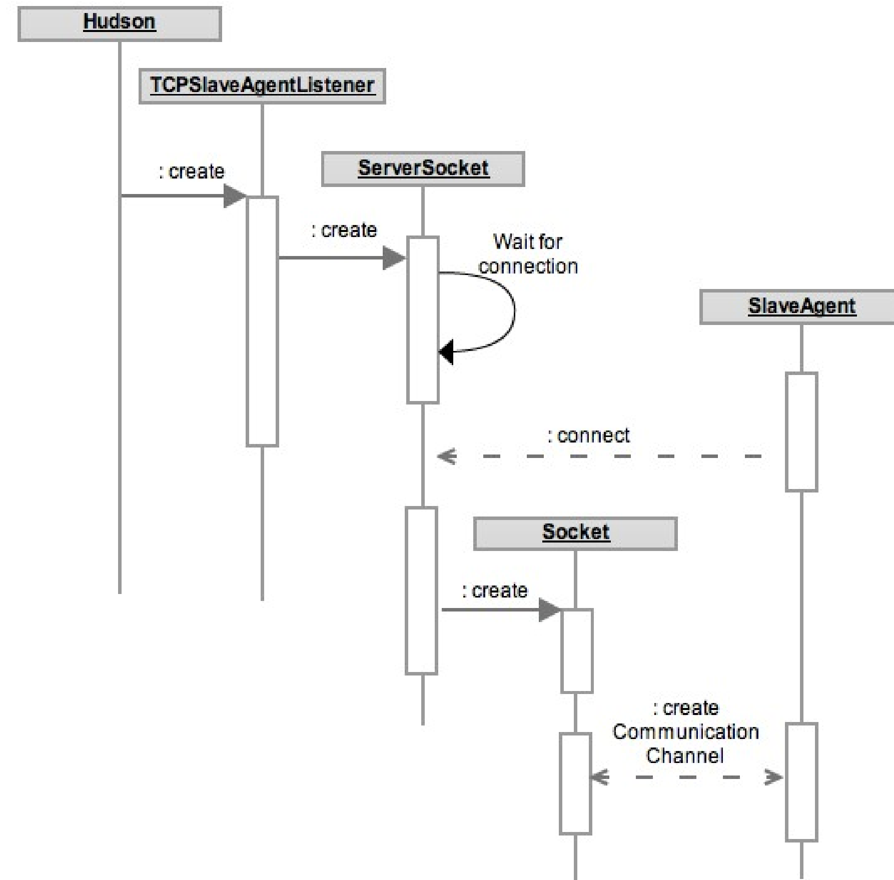
Establishing JNLP TCP Connection

When Hudson Model Object is initialized a `TcpSlaveAgentListener` object is created to accept socket connections from Slave Agent.

TcpSlaveAgentListener creates the ServerSocket and makes it to listen for connections indefinitely in a Thread.

When ServerSocket accepts a connection request from Slave Agent, it creates a Socket and hands over it to another thread called ConnectionHandler.

Connection handler initiates the hand shaking with Slave Agent.



Establishing the Channel

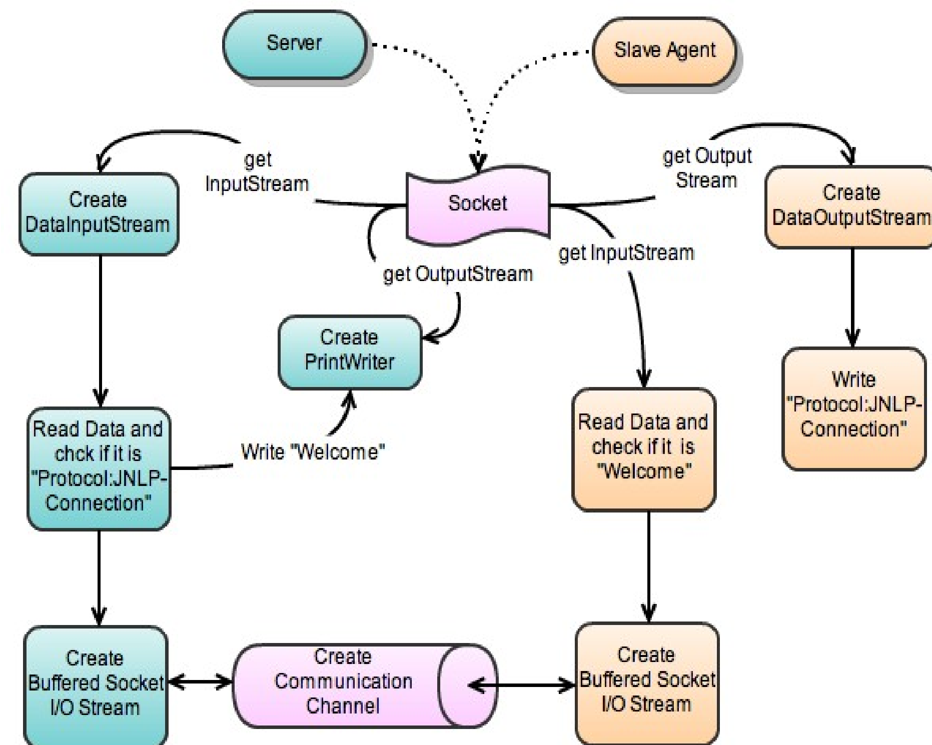
After the socket connection is established, in the server side, a DataInputStream is created from the Socket Input Stream and a PrintWriter from Socket Output Stream. In the Slave Agent side, a DataOutputStream is created from the Socket Output Stream and a BufferedReader from Socket Input Stream.

Slave Agents outputs the required String. Server Reads the String and verifies if it matches as “Protocol:JNLP-Connection” then initiates a JNLP Channel. If the String is “Protocol:CLI-Connection”, then the socket request is from a Command Line client.

Server outputs the string “Welcome” and then creates Buffered Socket I/O stream and creates the Communication channel using those I/O streams.

If client receives the message “Welcome”, then it creates the Communication Channel from Socket I/O stream.

The client server communication happens via this established channels.



ORACLE

Sending the Remote Request

Callable is the closure that need to be sent to the Remote Slave for execution. Master does that using `Channel.call(Callable)`.

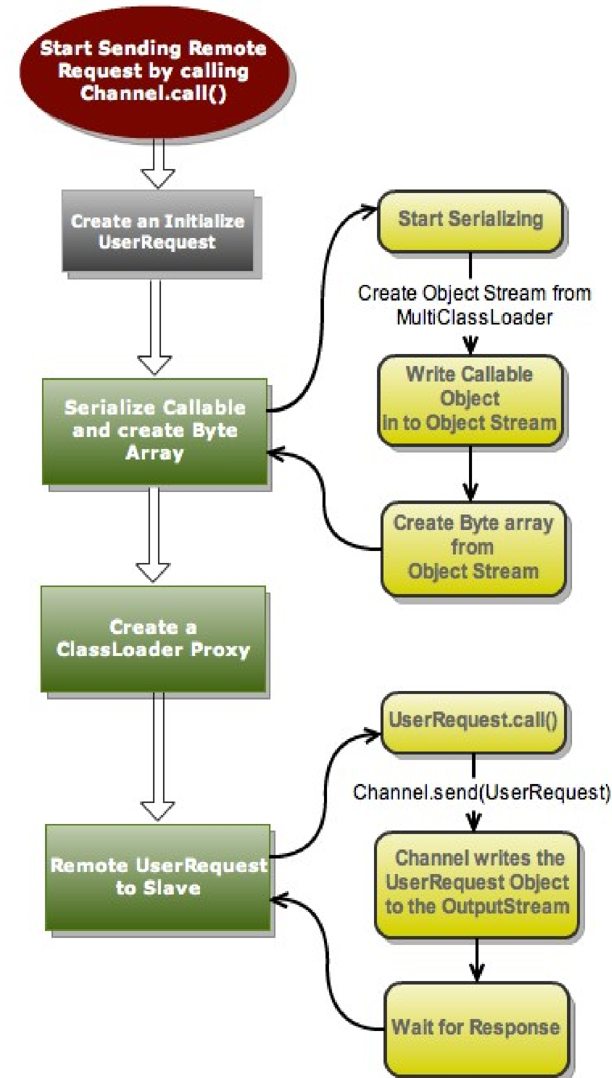
Channel sends the Callable by placing it in an envelop called UserRequest. This envelop also encloses the ClassLoader proxy to be used in the Remote end.

UserRequest initialization method creates a MultiClassLoader and creates an Object Stream from it. Then the callable is serialized in to it and later converted in to a Byte Array.

A classLoader Proxy is created and kept in the UserRequest Object.

Then Channel writes the UserRequest Object in to the ObjectOutputStream of the socket connection and waits for the Response from Slave. Channel is blocked until response arrives from slave if it is not a asyncCall.

The serialized UserRequest Object encapsulates the Closure (callable) and ClassLoaderProxy.



Slave Receiving the Remote Request

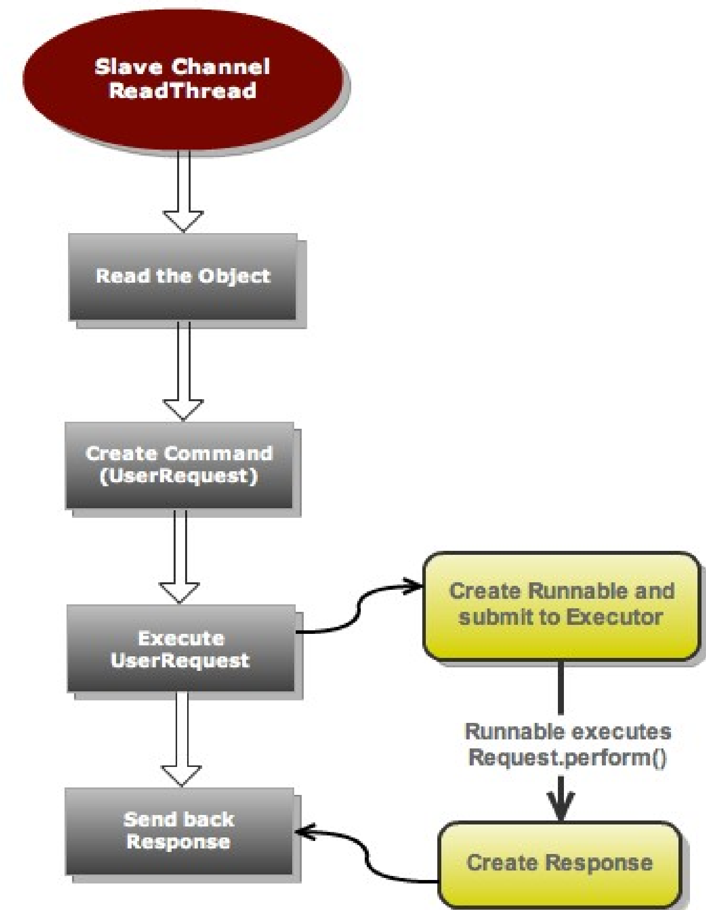
The Channel's ReaderThread (slave side) indefinitely waits for requests from master. When the request come over the channel, the ReaderThread reads the Object.

A Command (Super class of UserRequest) is created from the ObjectInputStream.

Channel creates a Runnable using the Command and pass it on to the Executor to execute the Request Object.

The Runnable calls `UserRequest.perform()` to do actual closure execution.

Finally a Response Object is created from the output of Request Object execution and send back to the master.



Slave Executing the Remote Request

When the UserRequest object is executed by the Executor, first it tries to deserialize the closure (Callable) it encapsulates.

It uses the Remote ClassLoaderProxy to do the serialization.

It creates the ObjectInputStream from ClassLoaderProxy passing in the Callable stored as ByteArray.

Reads the Closure Object from the ObjectInputStream.

When ClassLoaderProxy loads the class, it acquires all the classes it needs from the master.

Finally Callable.call() method is executed to full fill the remote execution.