Ipc

IPC

IPC, for InterProcess Communication, is a fast and easy RPC mechanism. Unlike Sun's standard RPC package, it does not use standard Java serialization and instead requires the author of every class to write the relevant serialization method. That extra work might seem like a drawback, but if you've ever tried to debug Sun's class versioning system, you realize that the extra work is in fact welcome.

IPC does not require a special compiler of any kind to create network stubs and skeletons. Rather, it uses introspection to examine a declared "publicly-available interface" and determine how to marshall/unmarshall arguments.

IPC is used as the internal procedure call mechanism for all of Hadoop and Nutch.

Use Model

IPC is a client/server system. A server process offers service to others by opening a socket and exposing one or more Java interfaces that remote callers can invoke. User server code must indicate the port number and an instance of an object that will receive remote calls. (see RPC.getServer())

A client contacts a server at a specified host and port, and invokes methods exposed by the server. User client code must indicate the target hostname and port, and also the name of the Java interface that the client would like to invoke. While a single IPC server object can expose several interfaces simultaneously, a client can invoke only one of them at a time. (see RPC.getClient())

There is no way for an IPC server to invoke methods of the client. There are places in Hadoop where bidirectional communication is helpful (e.g., in DFS, where the Name and Data nodes must report status to each other). In these cases, one side acts as a client, making the same call over and over again. The server always returns a special "status" object, which the client may then interpret as a request to perform work.

Under the covers

The IPC mechanism automatically inspects the client's requested interface, plus the server's exposed interfaces, and figures out how to marshall /unmarshall arguments for the remote call. This system works fine as long as all arguments in methods consist of either Java's builtin types, or String, or an implementation of the Writable interface. (Or an array of one of those types)