The Communication Layer

The Communication Layer is designed to be able to create and mediate the connection between a Presentation Layer and a Database Layer that will most likely "live" in different Virtual Machines. The Communication Layer uses the Remote Method Invocation, RMI.

Its greatest advantage is that it has the same interface as the Database Layer, so that the Presentation Layer, or at least most parts of it, are not aware of the network connection.

Introduction

First, we describe the RMI - how it works and its terminology.

How the RMI Works

The Remote Method Invocation allows us to invoke methods of objects, that do not "live" in the same JVM (i.e. that were not created in the same JVM) as if they were only local objects. The RMI machinery is quite complex and it will not be covered here.

The use of this mechanism is simple and involves five steps:

1. A program **P** creates an object **O** in its **JVM**.
2. The RMI must become aware of existence of this objects, so that the Object can accept remote calls. The program **P** must **export** it = make RMI aware of it. If an object is exported, it is called a **remote object**.
3. The program **P** should **bind** the remote object to a **Naming service**, that is easily accessible, so that anyone can obtain the **remote reference**. If an object is exported, a remote reference (or **stub**) is created. The stub is a proxy that contains information about the exported object and allows its owner to invoke calls on the remote object. Stubs can be passed via the network and can be available at different JVM, while the remote object never leaves the JVM in which it was created.
4. Clients that wish to call methods of a remote object must obtain its stubs first. This is achieved by asking the Naming service (**lookup**), which returns the appropriate stub for the remote object.
5. Clients can call methods of the object via the stub.

The Client always has a **stub** and can use it to invoke methods of the **remote object**. Each method call is handled by the RMI – parameters are **marshalled** (packed, serialized) and send to the remote RMI, which must recreate those objects (unmarshall, deserialize) and fill them with sent values, perform the method call, marshall the result, send it back to the other RMI, which unmarshalls them and returns them as a result of the method call.

The Naming Service is a remote object, that is capable of

- storing pairs `[Name, Stub]` in a hash-table,
- and returning the stub when asked for it by the associated name.

For our purposes we use the default Naming Service Java provides - the **rmiregistry**.

A stub is generated for each remote object. Stubs are objects, they have their own classes. If we use the default Naming service, the stub's class is downloaded automatically, so that the stub can be created on the local client.

If we for some reason do not wish to use the Naming service and if we have our own way of handing over the stub to the client, we must ensure, that the client has an appropriate class, to be

able to re-create the stub.

What is Required

First, everything that is passed as an argument of a remote method call, must be serializable. Almost all objects that are available at java.lang and java.util are already serializable. Making your own object serializable requires you to implement the "marker" interface Serializable. The interface is empty.

Second, every method, that accepts a remote call, must throw a RemoteException. This way the RMI mechanism lets us know that something went wrong during the remote call. After all we work with the network.

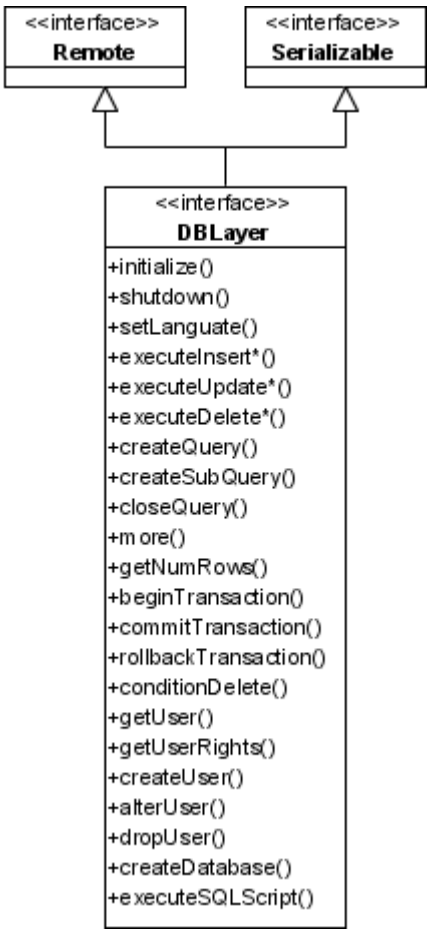
Third, every object, whose methods are to be called, must either implement the Remote interface or extend the UnicastRemoteObject. Both ways are equal, but if the object extends the UnicastRemoteObject, it is exported automatically the moment it is created. If it implements the Remote interface, we must export it ourselves.

Participants

There is only one participant in this case - it is the Remote interface DBLayer. Everything else is handled by the RMI mechanism (assuming all objects were properly exported).

The DBLayer Interface

The DBLayer interface is located in the file net.sf.plantlore.middleware.DBLayer.java.



This is the interface that is presented to the Presentation Layer. The Presentation Layer can use all its methods. The implementation of the interface is different: in the client JVM it is the stub and in

the remote JVM it is the remote object.

Implementation

The particular implementation can differ but in our case it is the HibernateDBLayer.

HibernateDBLayer

This is the implementation of the DBLayer interface. For further details see the Section **Database Layer**.

Notes

Stubs must be generated manually for every remote object that will not be bound to the Naming Service. Those stubs must be made available to both client and server; the RMI mechanism would not work properly otherwise.

To generate a stub of a remote object you must use the rmi compiler (rmic).