The Server

In this Section we will discuss all participants on the Server side of the Application and their interaction when a new Database Layer must be created and properly destroyed. We well explain the mechanism of automatic Database Layer destruction if the Client fails to do so. And we will discuss the ability to gain access to the Server in order to perform its administration.

Introduction

The Server is a remote object, that is responsible for the creation and management of its Database Layers including their destruction if something happens to the Client - if the Client crashes or if the network connection is lost.

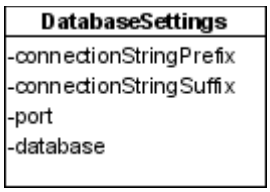
Server can be controlled, even remotely, which is why the Server is a remote object itself.

Participants

There are several participants and interfaces. They can be found in these two packages: net.sf.plantlore.middleware and net.sf.plantlore.server. The most important participants are the RemoteDBLayerFactory and the Server.

DatabaseSettings

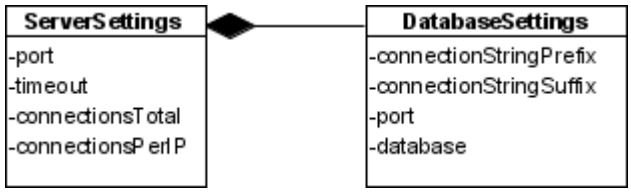
The Database Settings is a simple holder object describing the settings of the database to which the RemoteDBLayerFactory should connect in order to create a new Database Layer. Database Settings are in the file net.sf.plantlore.server.DatabaseSettings.



There are only a few relevant settings that are supplied by the Administrator of the Server: the database type (engine), port and suffix (which is the database parameter). From these values the connectionPrefixString is created - it is the JDBC connection string and the suffix is the part the goes after the question mark (?). The rest of the settings required to initialize the Database Layer is supplied by the Client (i.e. the name of the database, the user name and the password).

ServerSettings

Server Settings configure the Server's policy to accept or reject a request to create another Database Layer and configure the Server itself. Server Settings are in the file net.sf.plantlore.server.ServerSettings.

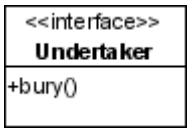


One part of the Server Settings are the Database Settings. Then there is th port on which the rmiregistry with the RemoteDBLayerFactory should be available, total number of connections and maximum number of connections per IP.

Undertaker

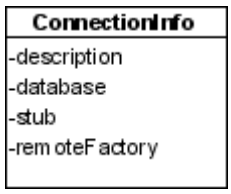
The RMI allows the remote object to monitor the state of its remote references. If there are no remote references to the remote object left (either because clients discarded them properly or because they crashed), the method **unreferenced()** of the Unreferenced interface is called.

The Undertaker denotes an object that can properly destroy stranded Database Layers, i.e. Database Layers whose clients crashed. After the time given by the timeout in Server Settings is over and the Client does not respond, its Database Layer is destroyed. The Interface is in the file net.sf.plantlore.server.Undertaker.



ConnectionInfo

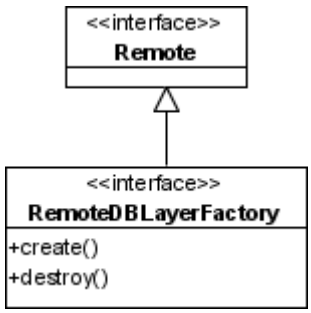
Connection Info stores information about the created Database Layer (the remote object) and its stub and about the RemoteDBLayerFactory that created it. It can be found in the net.sf.plantlore.server.ConnectionInfo.



RemoteDBLayerFactory

This interface is located in the net.sf.plantlore.middleware.RemoteDBLayerFactory and those who implement it must be able to

1. create a new Database Layer, export it and return its stub,
2. unexport and destroy the Database Layer when given the stub.

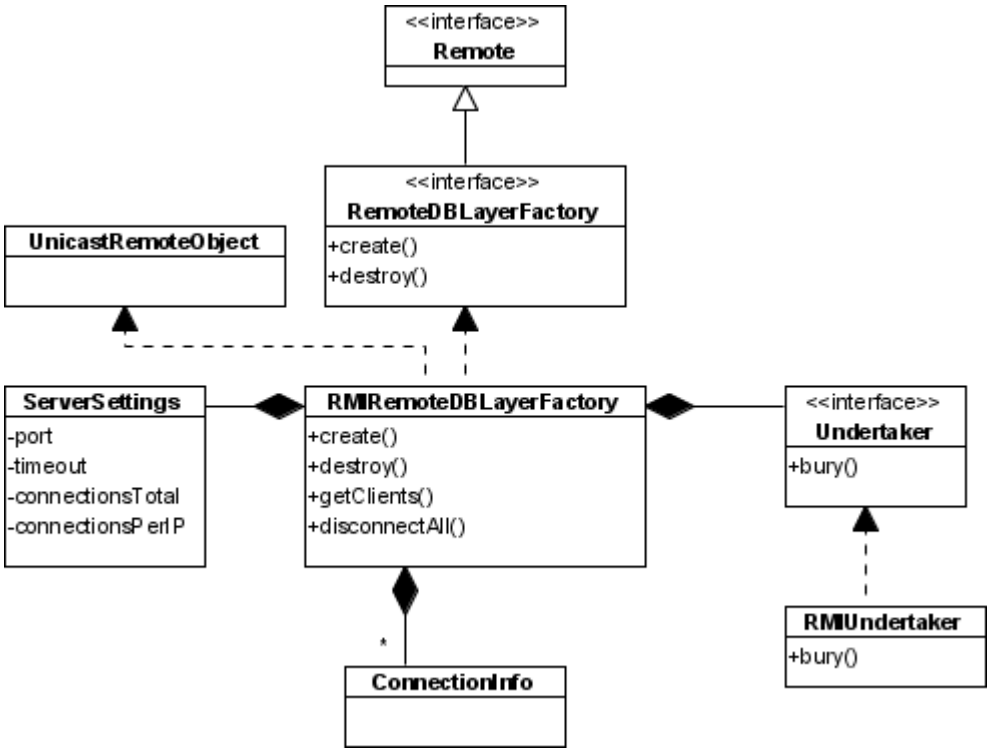


Note that the Database Layer should not made available in the Naming service to anyone - it should be exported and become a remote object, but the stub of the Database Layer should be returned to the caller only. No one else should be able to access it. Therefore every caller will have its own Database Layer. In order to monitor the number of Database Layers created, the RemoteDBLayerFactory may implement some policy and reject creation of new Database Layers if the policy would be violated.

RMIRemoteDBLayerFactory

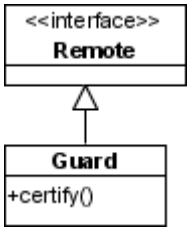
RMIRemoteDBLayerFactory is the particular implementation of the RemoteDBLayerFactory interface that can perform all the tasks that are delegated to it. It also implements its own policy to limit the total number of Database Layers created as well as the maximum number of Database Layers created for clients with the same IP address. If those numbers would be exceeded the creation of another Database Layer is rejected.

The class is equipped with its own implementation of the Undertaker that can get rid of Database Layers whose clients crashed or terminated without destroying their Database Layers properly.



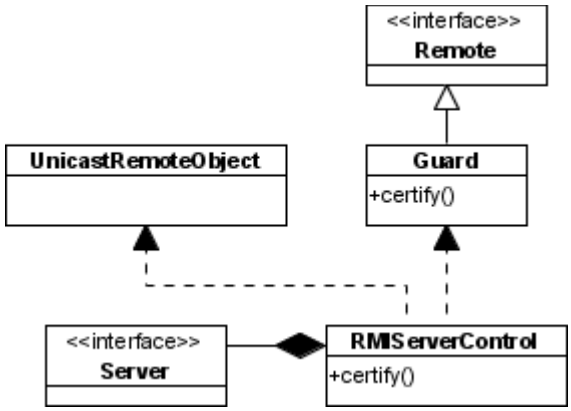
Guard

Guard is an interface that provides access to the Server for its Administration. The Server should not be publicly available in the rmiregistry. The interface is int he file net.sf.plantlore.server.Guard.



RMIServerControl

The RMIServerControl is the implementation of the Guard. It stored the stub of the Server as well as the password guarding the access to the it. If the supplied certification information fits, the stub is returned. The source is avaliable at net.sf.plantlore.server.RMIServerControl.

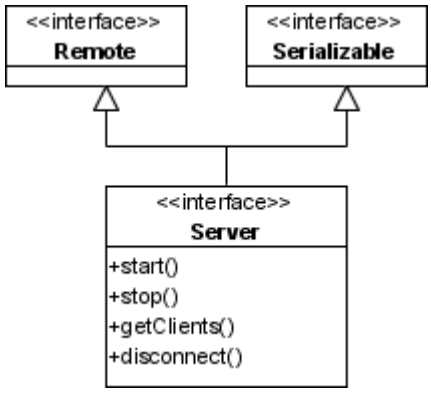


This is how even a remote administration of the Server is achieved. Anyone who wishes to control the Server, contacts the Guard, that is publicly available in the Naming service, calls its **certify()** method and passes the authorization information. If the information is valid, the stub of the Server is returned.

Server

The Server interface serves to control the Server - especially to start and terminate it. If the Server is asked to start, it should create a new RemoteDBLayerFactory, export it and bind it to the rmiregistry under a well known name. The Server itself should not be available in the rmiregistry, because then anyone would be able to obtain it and terminate which is not desirable.

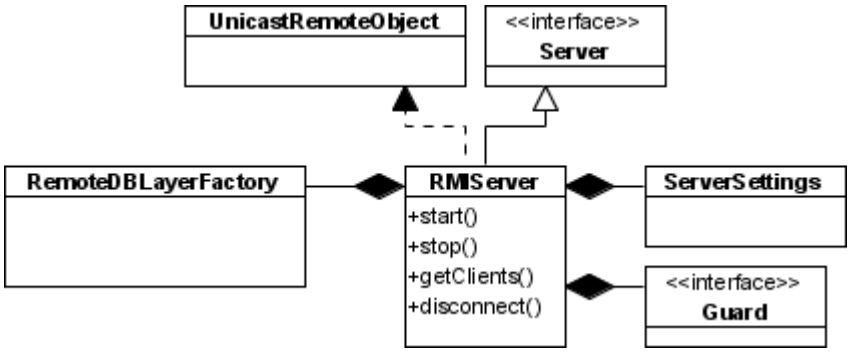
When the Server is asked to terminate it must unbind the RemoteDBLayerFactory from the rmiregistry, unexport it and instruct it to destroy all the Database Layers it has already created, so that the connection of all clients is terminated as well.



The Server may be able to return the list of currently connected clients (i.e. the list of Database Layers that have been created) and may be able to disconnect some of its clients (i.e. unexport and destroy the selected Database Layer).

RMIServer

The RMIServer is the implementation of the Server interface. It does exactly all the things the interface specifies. RMIServer is in the file net.sf.plantlore.server.RMIServer.



The RMIServre contains the RemoteDBLayerFactory that constructs and destroys DatabaseLayers, Guard that protects the access to the Server for administration and Server Settings that describe the configuration of the Server.

Database Layer

The Database Layer to be created. It is described in particular detail in the Section **Database Layer**. Although the RemoteDBLayerFactory creates and destroys it, it has no other interest in it.

The Database Layer knows that it can be a remote object and must be prepared to take some extra measures. Since the Database Layer cannot allow the object SelectQuery to leave the JVM where the DBL lives, because if it did, then anyone could modify it and supply a malicious SelectQuery which would completely undermine the system of access rights. Therefore the Database Layer makes the SelectQuery another remote object that is exported by the Database Layer, every time the Database Layer is asked to create a new Query. The Database Layer must also destroy (unexport)

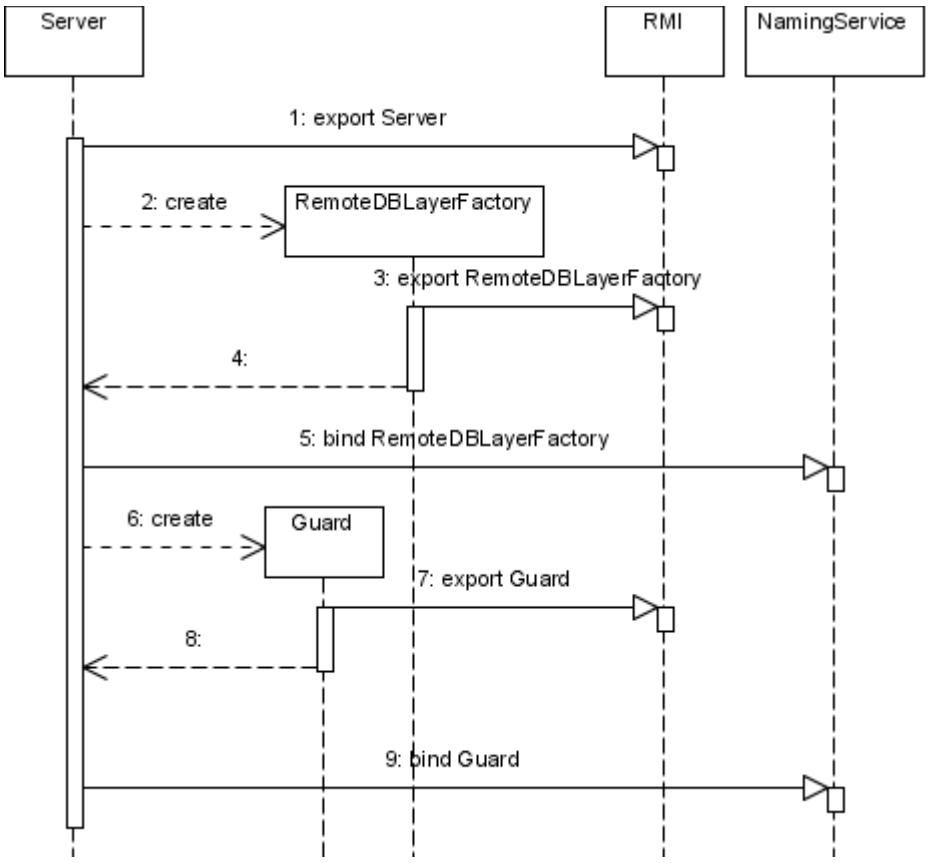
the SelectQuery. This is what the Database Layer's methods **createQuery()** and **closeQuery()** do.

Interaction

In this section we provide the detailed information about the interaction between all participants in several situations.

Starting the Server

Starting the Server is a very straightforward operation. Note that the Server is indeed not bound to the rmiregistry (Naming Service) while both RemoteDBLayerFactory and Guard are.



Stopping the Server

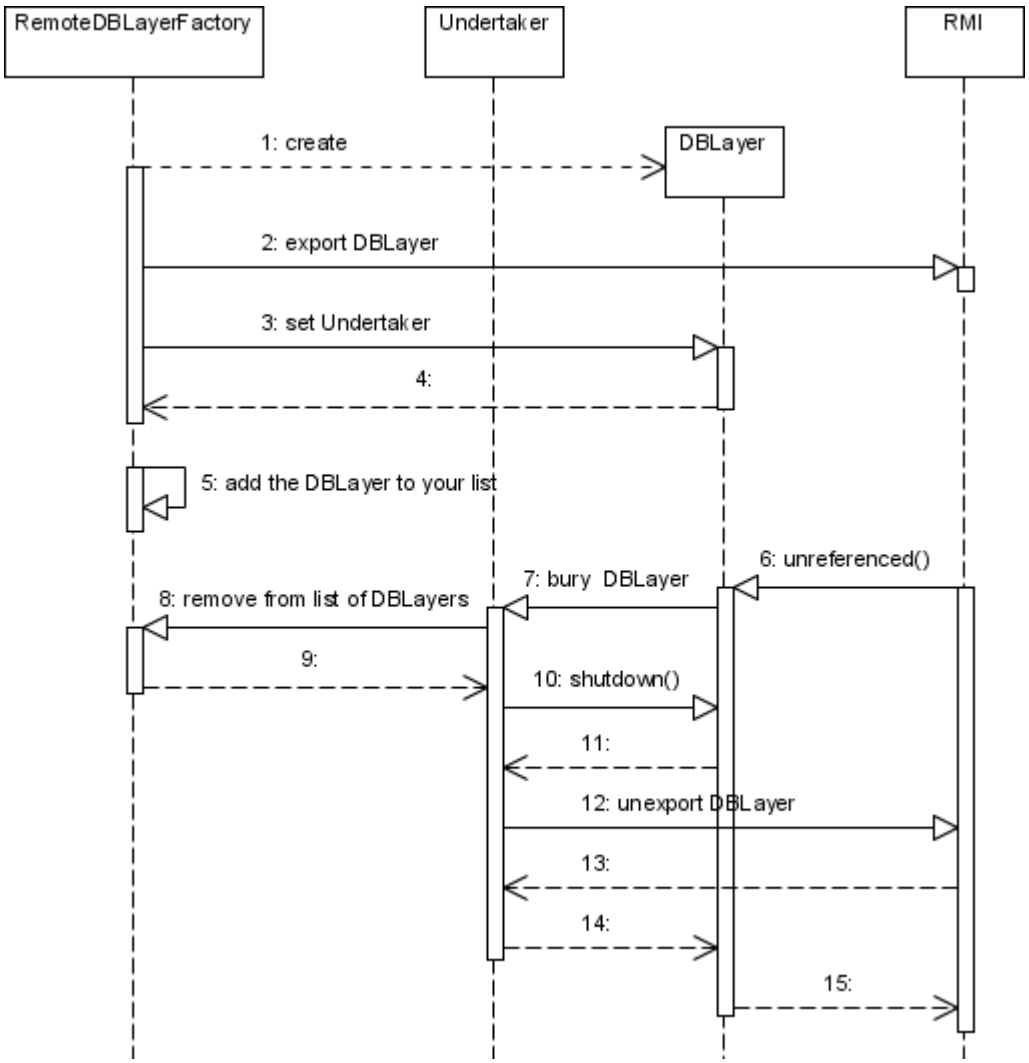
When the Server is to be stopped, it performs all operations but in the reverse order and their counterparts - unbind, unexport, and destroy.

Creating a New Database Layer and Destroying It

In this picture there you can see the process of creation of a new Database Layer, if the RemoteDBLayerFactory is asked to. It takes steps 1-4. The caller does not obtain the Database Layer itself (because it is the remote object) but its remote reference.

The process of proper destruction of a Database Layer is very similar to the process of its creation, but it is in the reverse order and counterparts of methods are used - unexport and destroy.

Steps 5-12. depict the situation when the network connection with the client has failed and the RMI considers the stub to be dead. In case there are no remote references (stubs) to the remote object Database Layer left, the RMI signals it to the Database Layer via the Unreferenced interface. In this interface the Database Layer calls its Undertaker that should take all steps necessary for its destruction. The Undertaker properly destroys the Database Layer by calling its shutdown() method and unexports it preventing it from accepting remote calls.


RemoteDBLayerFactory
Undertaker
RMI
DBLayer
1: create
2: export DBLayer
3: set Undertaker
4:
5: add the DBLayer to your list
6: unreferenced()
7: bury DBLayer
8: remove from list of DBLayers
9:
10: shutdown()
11:
12: unexport DBLayer
13:
14:
15:

The Server Manager

Server manager is a simple object that can

- create a new Server,
- connect to a running Server,
- manage the Server it has created or to which it has connected.

Introduction

The Server manager can create a new Server and give it the order to start, stop, disconnect a client, or return the list of currently connected clients. It simply makes use of the interface that the Server implements.

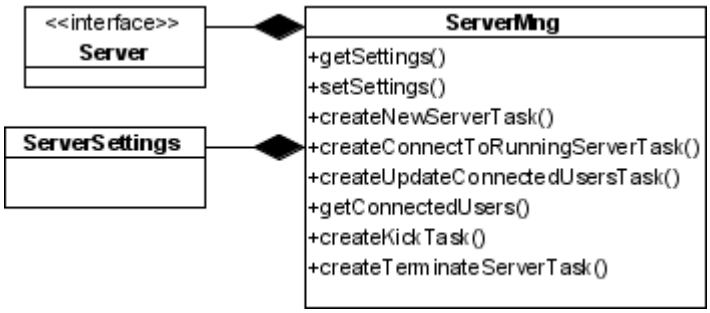
In case the Server is already running on some computer, the Server manager contacts the Guard, obtains the stub of the Server from it, and (because the Server is a remote object) can call the same methods as in the previous case. This means, that the Server can be stopped or have its clients disconnected remotely.

Participants

There is only one participant this time.

ServerMng

The Server Manager is capable of running a new Server with the specified Server Settings that can be stored and loaded from a file. The Server Manager is located in the file net.sf.plantlore.server.manager.ServerMng.java.



It works as a task factory. To learn more about tasks see the Section **Tasks and Dispatcher**. These tasks are very simple as they call methods of the Server, that has been either created or connected to (obtained from the Guard after certification).

Notes

The Server is configurable and it is possible to store the configuration in a file. The configuration is restored next time the Server is started. In case the configuration file is missing, the default hardwired configuration will be used - it supports up to 32 total connections and maximum of 3 connection per one IP address.

The file can be found in the Plantlore configuration directory under the name plantlore.server.xml. In this file there is a simple tree describing the Server Settings.

Most of these settings can be set using the GUI except the number of connections per ip and the total number of connections. These settings are not documented in the User manual because they were not required to be part of the Server.

The format of the configuration file:

```
<config>
  <server>
    <port>1099</port>
    <connections>20</connections>
    <perip>3</perip>
    <database>
      <engine>postgresql</engine>
      <port>5432</port>
      <parameter></parameter>
    </database>
  </server>
</config>
```