Fun RMI

Invode methods on remote methods, have a fun time.

Building and Testing

To compile all source files, run: ./compile.sh

The pettingzoo package is an example application which uses this RMI framework. To run it and watch it in action, run: ./test.sh

Overview of Project

```
rmi.
  RemoteMarker          (objects which implement this are Remote objects)
  ROR                   (Reference to a Remote object on a host)

  Stub                  (client/proxy object for a Remote object)
  RMIServer             (endpoint for invocation Remote objects on a host)
  RMIException          (Exception which may be raised by calling a stub method, since network is unreliable)


  registry.
    Registry            (Registry server, keeps track of key-ROR mapping)
    RegistryClient

  messaging.            (internal library for messages over sockets)
    RMIMessage
    MessageType

pettingzoo.             (example application utilizing rmi)
  Animal
  Dog
  Cat
  Monkey

  ZooServer
  PettingClient

  TestRunner            (Contains main function, sets up and runs system)
```

Example Use Case: PettingZoo

In this PettingZoo, you have Animals, specifically Dogs, Cats, and Monkeys.

Animals live on long-running ZooServers, and short-lived PettingClients can use RMI to remotely invoke methods on the animals.

See pettingzoo/PettingClient.java pettingzoo/ZooServer.java for code examples, and pettingzoo/TestServer.java to see how the whole system is setup.

TestRunner Details

The setup and operation of the system is demonstrated in pettingzoo.TestRunner. Although it runs on a single host, the usage of Hostnames and Ports for server connections suggests the same setup strategy would work if implemented on multiple hosts. The following is a breakdown of the steps involved:

1. Start a long-running rmi.registry.Registry server
2. Start two ZooServers
    1. Each ZooServer creates an RMIServer
3. Tell ZooServers to create a few Dogs and Cats
    1. Each ZooServer instantiates objects
    2. Then they register the objects with the RMIServer using the RMIServer.addObj
4. Start two PettingClients
    1. Each creates one RegistryClient
    2. ZooClients get references to created objects

External API

Registry

rmi.registry.Registry

To start the registry server on a host, you can write a simple Java class with a main function, in which you instantiate a rmi.registry.Registry (you can provide a port number to bind on). To start it on the same thread, call the serveForever method. Registry objects implement Runnable so you can also start the Registry in another thread.

Server

rmi.RMIServer(String hostname, int port, String registryHost, int registryPort)

You can create an RMIServer when you want to expose objects on a host to clients via the RMI facility. You can create multiple RMIServers across multiple hosts, and the registry running at the given host and port will keep track of all objects.

Methods

rmi.RMIServer.addObj(String name, Object obj)

Call this method when you want to expose an object via the RMI Server so that a client may invoke a method on it remotely. Give it a name for propective clients to be able to refer to it as (see RegistryClient.lookupStub).

Client

rmi.registry.RegistryClient(String registryHost, int registryPort)

This is an object a client can use to communicate with an RMI Registry, potentially on another host. Note that this object is also used by the RMI server to communicate with the registry - but the application programmer need not care about this implementation detail.

Methods

rmi.registry.RegistryClient.lookupStub(String name)

Stub system

When you call RegistryClient.lookupStub, you get back a subclass Stub object representing a proxy of remote object. Therefore, for every class of objects you want to make RMI-available, you'll need to write Stub classes.

For example, to make pettingzoo.Dog available to RMI, you have to make a class called pettingzoo.DogStub that implements the RemoteMarker interface. The naming rule matters, it has to be className + "Stub". The RemoteMarker interface doesn't define any methods, it's just to mark the stub as a stub for a remote object.

Writing a stub class is more thoroughly documented in pettingzoo/CatStub.java and DogStub.java.

Known limitations

All the arguments must be serializable objects (they should implement Serializable). Primitive types are not supported but you can use Wrapper classes to work around this limitation (ie. int -> Integer). Since RORs are Serializable, you can send RORs or return RORs over RMI if you want.

Internal Design and Implementation

Our design allows multiple RMI Servers to register their objects with a central registry server, and multiple clients to locate and fetch stubs for the objects listed in the registry.

System Architecture

Stub: (used by a client)

- Proxy object for a Remote object.
- Has the same method interface as the remote object.
- Has a ROR for the real object.
- A method call actually invokes Socket communication to the RMI Server of the host (given by the ROR) where the object is located.
- Returns result of the method invocation on the RMI Server, may raise an exception.
- In addition to the exceptions raised by the real object, this may also raise RMIException, which occurs in the case of an error caused by unreliability of the network.

Remote Object Reference (ROR): (used throughout the system)

- Object which encodes the hostname, port, object, object class name, of a Remote object.
- Remote object means that the real object is on a host where a RMI Server is awaiting a method invocation message for the object.

RMI Server: (used by a server)

- Communication endpoint for a Remote object.
- Receives a Message to invoke a method on an object.
- Dynamically invokes method, returns result/exception to the requester.

Registry: (run as a daemon, Singleton in a system)

- Global record of Remote objects, with unique names.
- Allows an RMIServer to register it's object as available.
- Allows a client to locate an object by name, and henceforth get a stub with a ROR.
- Uses Remote Object Reference to encode the location of an object.

rmi.RMIServer

rmi.RMIServer(String hostname, int port, String registryHost, int registryPort)

This constructor creates an RMIServer, which is a socket server that listens for Remote Method Invocation messages by an Remote object Stub. When the constructor is called, it attempts to connect to the Registry server to validate that it's running.

rmi.RMIServer.addObj(String name, Object obj)

This method creates a Remote Object Reference (ROR), adds the (ROR, obj) to a HashMap, tells the Registry about (name, ROR).

rmi.registry

The Registry is a server/service that keeps track of the RORs exposed via RMI, and the key / ROR association. It currently runs on one host, as an Inet Socket Server. It supports registering and looking up a ROR by key/name. It does not support de-registering an object (so no garbage collection). Details about Messages (marshalling etc) is somewhere else in this report.

The RegistryClient is a class you can use to easily communicate with a Registry via java methods which encapsulate Socket message communication.

rmi.registry.Registry

rmi.registry.Registry(int port)

Tell the Registry which port it will bind to when you run it. It implements Runnable so you can run it in a Thread, and it has a serveForever method which will start the server and server until killed externally.

Communicate with the registry via a RegistryClient.

rmi.registry.RegistryClient

rmi.registry.RegistryClient(String registryHost, int registryPort)

Contains methods for performing socket communication with a Registry.

rmi.registry.RegistryClient.lookupStub(String key)

Uses inet socket communication to ask Registry for the remote object reference with the given key/name. Then creates a stub from the remote object reference and returns it. The resulting stub should have the same method interface as the class it stubs for.

rmi.registry.RegistryClient.registerRor(String key, ROR ror)

Uses inet socket communication to tell Registry about the given remote object reference and associate it with the given key/name.

rmi.messaging

This packages contains a few utility classes used internally by RMI to implement Inet socket-based message passing.

rmi.messaging.MessageType

Contains enums of all the different types

rmi.messaging.RMIMessage

The Message class we pass around over the sockets for consistency. It has a variety of constructors, which are not written with the best style, but we implicitly remember which attributes are set for each MessageType.