RMI 2 — Foo Example

rtable.policy

Referred to by the java job on both client and server side

```
elaine1:~/java/rmi> more socket.policy
grant {
  permission java.net.SocketPermission "*:1024-65535",
     "connect,accept";
  permission java.net.SocketPermission "*:80", "connect";
  permission java.awt.AWTPermission "*";
  permission java.lang.RuntimePermission "*";
};
```

Server Side

rmiregistry 32456 & ## started earlier and left running

elaine21:~/java/rmi2> javac *.java
elaine21:~/java/rmi2> rmic FooServer PipeServer ## do this when *Remote has changed
elaine21:~/java/rmi2> alias rjava ## see the alias for rjava
java -Djava.security.policy=rtable.policy
elaine21:~/java/rmi2> rjava FooServer 32456
2000-05-16 11:46:55.789 FooServer: server bound
2000-05-16 11:47:14.475 FooServer: doit start
2000-05-16 11:47:14.48 FooServer: serverInternal
2000-05-16 11:47:14.481 FooServer: doit done
2000-05-16 11:47:15.0 FooServer: doit start
2000-05-16 11:47:15.001 FooServer: serverInternal
2000-05-16 11:47:15.09 FooServer: doit done
^C

Client Side

elaine20:~/java/rmi2> rjava FooClient elaine21:32456
2000-05-16 11:47:14.493 Client: received from server -- hello there
2000-05-16 11:47:15.012 PipeServer: got message Server says hello
2000-05-16 11:47:15.072 PipeServer: got runnable
This code sent from the server 2
2000-05-16 11:47:15.082 Client: done
^C

% rjava Client *server-machine*

/ / FooRemote.java

```
// The interface exposed by the FooServer -- the client
// sends these on the client end, and they happen on the server
// end.

import java.rmi.*;
import java.rmi.server.*;

public interface FooRemote extends java.rmi.Remote {
   // Run the doit() operation on the server
   public String[] doit() throws RemoteException;

   // Send a pipe over to the server
   public void install(PipeRemote pipe) throws RemoteException;

   public static final String SERVICE = "nickFoo";
}
```

/ / FooClient.java

```
import java.rmi.*;
import java.math.*;

public class FooClient {
   public static void main(String args[]) {
      if (System.getSecurityManager() == null) {
         System.setSecurityManager(new RMISecurityManager());
      }
      try {
         String name = "//" + args[0] + "/" + FooRemote.SERVICE;

         // Get the stub for the server object
         FooRemote foo = (FooRemote) Naming.lookup(name);

         // Send the server a message
         String[] result  = (String[]) foo.doit();
         Log.print("Client: received from server -- " + result[0] + result[1]);


         // Create a pipe here and send it to the server
         PipeRemote pipe = new PipeServer();
         foo.install(pipe);

         // This will call us back on the pipe
         foo.doit();

      } catch (Exception e) {
         System.err.println("FooClient exception: " +
            e.getMessage());
         e.printStackTrace();
      }

      Log.print("Client: done");
```

```
   }
}
```

/ / FooServer.java

```
// Demonstrates RMI
// The client invokes doit() which runs on the server
// The client can send us a pipe object

import java.rmi.*;
import java.rmi.server.*;


public class FooServer extends UnicastRemoteObject
      implements FooRemote
{
   PipeRemote pipe;

    public FooServer() throws RemoteException {
      super();
      pipe = null;
    }

    public String[] doit() throws RemoteException {
      Log.print("FooServer: doit start");

      serverInternal();

      if (pipe != null) pipe.send("Server says hello");

      if (pipe != null) pipe.sendRunnable(new MyRunnable());

      // Make a little array to return
      // (arrays and strings automatically serializable)
      String[] result = new String[2];
      result[0] = "hello";
      result[1] = " there";

      Log.print("FooServer: doit done");
      return(result);
    }

    private void serverInternal() {
      Log.print("FooServer: serverInternal");
    }


    // Sent by the client to give us a pipe
   public void install(PipeRemote pipe) throws RemoteException {
      this.pipe = pipe;
   }


   public static void main(String[] args) {
      if (args.length != 1) {
         System.out.println("Need port number");
         System.exit(0);
```

```
      }

      if (System.getSecurityManager() == null) {
         System.setSecurityManager(new RMISecurityManager());
      }
      String name = "//localhost:" + args[0] +
         "/" + FooRemote.SERVICE;
      try {
         FooRemote impl = new FooServer();
         Naming.rebind(name, impl);
         Log.print("FooServer: server bound");
      } catch (Exception e) {
         System.err.println("FooServer exception: " + e.getMessage());
         e.printStackTrace();
      }
   }
}


// A separate runnable object. We create one of these and send
// back to the client on the pipe. It runs on the client.
class MyRunnable implements Runnable, java.io.Serializable {
   public void run() {
      int i = 1;
      i = i + 1;
      System.out.println("This code sent from the server " + i);
   }
}
```

Reverse Pipe Trick

Pipe implements the standard PipeRemote / PipeServer structure
Allocate the PipeServer on the client side
Pass it back to the server side
In this case, the server has the stub and the client has the real one
When the server sends a message to its stub, it executes on the client

/ / PipeRemote.java

```
import java.rmi.*;
import java.rmi.server.*;

// A creates one of these and sends it to B.
// B can then invoke the send() operation on theirs,
// and it is received on the instance held by A.
// Sending a runnable sends code back to A -- the object
// must be serializable.

public interface PipeRemote extends java.rmi.Remote {
   public void send(String message) throws RemoteException;
   public void sendRunnable(Runnable runnable) throws RemoteException;
}
```

/ / PipeServer.java

```
import java.rmi.*;
import java.rmi.server.*;


public class PipeServer extends UnicastRemoteObject
   implements PipeRemote {

   public PipeServer() throws RemoteException {
      super();
   }

   public void send(String message) throws RemoteException {
      Log.print("PipeServer: got message " + message);
   }

   public void sendRunnable(Runnable runnable) throws RemoteException {
      Log.print("PipeServer: got runnable");
      // Run the thing they sent us in its own thread
      Thread thread = new Thread(runnable);
      thread.start();
   }
}
```