SE 550: Lecture 9

Overview

Using MiniSOAP RMI

Remote task execution

Stubs for remote objects

Remote Method Call

Naming service

RMIC

Using MiniSOAP RMI

Redoing the hash example using minisoap:

1. Write a remote interface Hash extends Remote.
2. Write a remote class HashImpl implements Hash.
3. Compile javac ajeffrey/teaching/hash2/*/*.java.
4. Run java ajeffrey.teaching.minisoap.RMIC ajeffrey.teaching.hash2.iface.Hash.
5. Compile javac ajeffrey/teaching/hash2/iface/HashSTUB.java.
6. Run java ajeffrey.teaching.minisoap.RMIRegistry.
7. Run java ajeffrey.teaching.hash2.server.Main.
8. Run java ajeffrey.teaching.hash2.client.Main.

Hooray, an RMI application using our own RMI infrastructure!

Implementing MiniSOAP RMI

Some components of an RMI system:

1. Remote execution task execution.
2. Stubs for remote objects.
3. Remote method invocation.
4. Naming service.
5. RMIC.

Remote task execution

An interface for point-to-point remote task execution:

```
public interface Executor {
    public void executeLocal (Task task);
    public void executeRemote (Task task) throws IOException;
    ...
}
public interface Task {
    public void run (Executor exec);
}
```

How can we use this interface to print "hello world" on a remote site with executor exec?

How can we implement the Executor interface?

Remote task execution

A factory for building executors:

```
public interface ExecutorFactory {
    public Executor build (Host host) throws IOException;
    ...
}
public interface Host {
    public String getName ();
    public int getPort ();
    ...
}
```

How can we use this interface to print "hello world" using port 1099 on site "foo.com"?

How can we implement the ExecutorFactory interface?

Stubs for remote objects

An interface for remote objects:

```
public interface Remote {
    public String getRemoteInterface ();
}
```

How can we implement a Hash remote interface?

Stubs for remote objects

Stubs are generated when a remote object is serialized:

```
class RMISoapWriterImpl extends SoapWriterImpl {

   ...

    public void serialize (final Object obj) throws IOException {
        if (obj instanceof Remote) {
            super.serialize (Stub.factory.build ((Remote)obj));
        } else {
            super.serialize (obj);
        }
    }

}
```

If we serialize a HashImpl object, what should be sent?

Stubs for remote objects

An interface for stubs:

```
public interface Stub {
    Host getHost ();
    String getObjectId ();
    public static final StubFactory factory = new StubFactoryImpl ();
    public static final HashMap lookup = new HashMap ();
}
public interface StubFactory {
    public Stub build (Remote object);
    ...
}
```

How can we implement StubFactory?

Remote Method Call

An interface for remote method call:

```
public interface RMI {

    public Object remoteCall (Host host, String objectId, String methodId,
      String[] argTypeIds, Object[] args) throws Exception;

    public static final RMI singleton = new RMIImpl ();

}
```

How can we use this to remotely call `hash.get ("fred")`?

Remote Method Call

A first shot at implementing RMI:

```
class RMIImpl implements RMI {

    public Object remoteCall (Host host, String objectId, String methodId,
      String[] argTypeIds, Object[] args) throws Exception {
        Task call = new Call (objectId, methodId, argTypeIds, args);
        Executor exec = Executor.factory.build (host);
        exec.executeRemote (call);
    }

}
```

What message is sent when we remotely call hash.get ("fred")?

How can we implement Call?

Remote Method Call

A problem: how do we get results back?

```
class Result {

    static final HashMap lookup = new HashMap ();
    static int numResults;

    final String resultId = "result" + ++numResults;

    Result () { lookup.put (resultId, this); }

    synchronized void returnResult (Object returned) {
        ... what goes here ...?
    }

    synchronized Object get () throws Exception {
        ... what goes here ...?
    }

}
```

Remote Method Call

The real thing:

```
class RMIImpl implements RMI {

    public Object remoteCall (Host host, String objectId, String methodId,
      String[] argTypeIds, Object[] args) throws Exception {
        Result result = new Result ();
        Task call = new Call (objectId, methodId, argTypeIds, args, result.resultId);
        Executor exec = Executor.factory.build (host);
        exec.executeRemote (call);
        return result.get ();
    }

}
```

What changes need made to Call?

Hint: use a Return task!

Remote Method Call

A gotcha:

Client 1 builds a new remote object `Foo foo`, and serializes it to Server. What gets sent?

Client 2 contacts Server and requests `foo`, and the server serializes it to them. What gets sent?

Client 2 calls `foo.hello ("world")`. What gets sent?

Two possible implementations:

a) Run RMI as a peer-to-peer network.

b) Have RMI servers perform call forwarding.

Tradeoffs? What does Sun do?

Naming service

This is just an RMI call on a known port and object id.

```
public interface Naming {

    public void rebind (String url, Object object) throws MalformedURLException;
    public Object lookup (String url) throws MalformedURLException;

    public void rebindLocal (String objectId, Object object);
    public Object lookupLocal (String objectId);

    public final static Naming singleton = new NamingImpl ();
    public final static int port = 1099;
    public final static String objectId = "naming";

}
```

When we call `Naming.singleton.lookup ("//foo.com/bar")` what remote method is called?

How can we implement `Naming`?

RMIC

Finally, we need to get stub code from somewhere:

```
public class RMIC {

    public static void main (String[] args) throws Exception {
        for (int i=0; i<args.length; i++) {
            Class contract = Class.forName (args[i]);
            String fileName = args[i].replace ('.','/');
            PrintWriter stubOut = new PrintWriter
                (new FileOutputStream (fileName + "STUB.java"));
            generateStub (contract, stubOut);
            stubOut.close ();
        }
    }

    ...

}
```

What stub code should be generated for HashSTUB?

Summary

Implementing a simple RMI system can be achieved in less than 1,000 lines of source.

Components are: remote task execution, remote object stubs, remote method invocation, naming and rmic.

XML can be used as the data representation language.

Next week: RMI and JDBC.