Distributed Objects

An RMI implementation

- VERY IMPORTANT NOTES-

VERY IMPORTANT: Parameter passing

Java Standard:

`void f(int x) :`

`Parameter x is passed by copy`

`void g(Object k) :`

Parameter k and return value are passed by reference

Java RMI:

`void h(Object k) :`

Parameter k is passed by copy!

UNLESS k is a REMOTE OBJECT (in which case it is passed as a REMOTE REFERENCE, i.e. its stub is copied if needed)

IMPORTANT: Parameter passing

Passing By-Value

When invoking a method using RMI,all parameters to the remote method are passed *by-value* .This means that when a client calls a server,all parameters are copied from one machine to the other.

Passing by remote-reference

If you want to pass an object over the network by-reference,it must be a remote object, and it must implement java.rmi.Remote.A stub for the remote object is serialized and passed to the remote host. The remote host can then use that stub to invoke callbacks on your remote object. There is only one copy of the object at any time,which means that all hosts are calling the same object.

Serialization

- Any basic primitive type (int,char,and so on) is automatically serialized with the object and is available when deserialized.

- Java objects can be included with the serialized or not:

- Objects marked with the ***transient*** keyword are not serialized with the object and are not available when deserialized.

- Any object that is not marked with the transient keyword must implement ***java.lang.Serializable*** .These objects are converted to bit-blob format along with the original object. If your Java objects are neither transient nor implement ***java.lang.Serializable,*** a NotSerializable Exception is thrown when ***writeObject()*** is called.

Serialization

- All serializable classes must declare a

private static final field named serialVersionUID

to guarantee serialization compatibility between versions.

If no previous version of the class has been released, then the value of this field can be any long value, as long as the value is used consistently in future versions.

```
private static final long serialVersionUID = 227L;
```

When not to Serialize

- The object is large. Large objects may not be suitable for serialization because operations you do with the serialized blob may be very intensive. (one could save the blob to disk or transporting the blob across the network)

- The object represents a resource that cannot be reconstructed on the target machine.Some examples of such resources are database connections and sockets.

- The object represents sensitive information that you do not want to pass in a serialized stream..

Distributed Objects

An RMI implementation

- Addendum -

RMI-IIOP

- **RMI-IIOP is a special version of RMI that is compliant with CORBA.**

  RMI has some interesting features not available in RMI-IIOP,such as distributed garbage collection, object activation and downloadable class files.

  EJB and J2EE mandate that you use RMI-IIOP, not RMI.

  rmic –iiop generates IIOP stub and tie (instead of stub and skeleton)
  rmic –idl generates OMG IDL

  See docs.oracle.com/javase/7/docs/technotes/tools/#rmi

Preparing and executing

NOTES:

- **starting from Java 2 the skeleton may not exist (its functionality is absorbed by the class file).**

Starting from Java 5 the rmic functionality has been absorbed by javac, so the whole process becomes transparent (but even more misterious…)

See docs.oracle.com/javase/tutorial/rmi/ for an example of current usage of rmi

Preparing and executing - security

The JDK security model requires code to be granted specific permissions to be allowed to perform certain operations.

You need to specify a policy file when you run your server and client.

```
grant { permission java.net.SocketPermission "*:1024-65535",
"connect,accept";
permission java.io.FilePermission "c:\\…path…\\", "read"; };
```

```
java -Djava.security.policy=java.policy executableClass
```

Access to system properties

- **Nota: instead of specifìying a property at runtime (-D switch of java command), You can hardwire the property into the code:**

```
-Djava.security.policy=java.policy

System.getProperties().put(
            "java.security.policy",
            "java.policy");
```