SE 550: Lecture 8

Overview

RMI and Callbacks

RMI and Firewalls

RMI and Versioning

RMI and Garbage Collection

RMI and Callbacks

Some interfaces for a library catalog:

```
public interface Catalog {
  public SearchResults search (String[] keywords);
}
public interface SearchResults {
  public int size ();
  public SearchResults narrow (String[] moreKeywords);
  public Book[] books ();
}
public interface Book {
  public String author ();
  public String title ();
  ...
}
```

What would we need to do to make the catalog accessable using RMI?

RMI and Callbacks

Often we want to make callbacks to arguments or results from remote objects:

```
interface Foo extends Remote { Bar getBar () throws RemoteException; }
interface Bar extends Remote, Serializable { void go () throws RemoteException;  }
class FooImpl implements Foo extends UnicastRemoteObject {
  Bar getBar () throws RemoteException { return new BarImpl (); }
}
class BarImpl implements Bar extends UnicastRemoteObject {
  void go () throws RemoteException { System.out.println ("hello world"); }
}
```

At the client we say:

```
Foo myFoo = (Foo)(Naming.lookup (url));
Bar myBar = myFoo.getBar ();
myBar.go ();
```

Where does "hello world" get printed?

RMI and Callbacks

```
interface Foo extends Remote { void useBaz (Baz b) throws RemoteException; }
interface Baz extends Remote, Serializable { void go () throws RemoteException;  }
class FooImpl implements Foo extends UnicastRemoteObject {
  void useBaz (Baz b) throws RemoteException { b.go (); }
}
class BazImpl implements Baz extends UnicastRemoteObject {
  void go () throws RemoteException { System.out.println ("hello world"); }
}
```

At the client we say:

```
Foo myFoo = (Foo)(Naming.lookup (url));
Baz myBaz = new BazImpl ();
myFoo.useBaz (myBaz);
```

Now where does "hello world" get printed?

RMI and Callbacks

Remote objects are serializable, but it's the *remote pointer* that gets sent, *not* the object itself!

When you call `oOut.writeObject (bar)`, if `bar` extends `UnicastRemoteObject` then a remote pointer to `bar` is serialized, *not* the contents of `bar`.

This is very useful when you want to avoid shipping large amounts of data over the network!

You only need to use `Naming.lookup` to get the *first* remote pointer, after that, you can use callbacks.

RMI and Firewalls

What is a firewall?

What does your average firewall do to RMI traffic?

What can we do about this (hint: tunnelling)?

RMI and Versioning

What is versioning?

Why is versioning difficult for non-distributed programming?

Why is versioning even harder for distributed programming?

What support for versioning does Java support?

(*Hint*: binary compatible change; serialVersionUid; the Package class.)

RMI and Garbage Collection

RMI makes garbage collection tricky!

What is garbage collection? When can an object be gc'd?

What is the *reference counting* implementation of gc?

What happens with remote pointers?

RMI and Garbage Collection

RMI requires remote objects to use reference counting gc.

Java VMs running RMI run a protocol where they send referenced and unreferenced messages to indicate whether they 'know about' a remote object.

What happens if the network goes down? (Hint: *object leasing*.)

What happens if the network comes back up again? (Hint: RemoteException.)

Summary

RMI has an interesting interaction with callbacks: it is often difficult to work out when an object is sent, and when a remote pointer to the object is sent.

RMI doesn't interact well with firewalls.

RMI (and serialization in general) makes versioning difficult: programmers have to be version-aware.

RMI makes garbage collection difficult: we need to use object leasing.

Next week: RMI Implementation.