Distributed computing in Java

Remote Method Invocation

Jan Jusko

FEE CTU

March 14, 2011

Outline

Outline

Outline

Outline

Outline

Outline

Distributed Object Application

- using remote objects as local
- taking advantage of remote computing resources
- client/server architecture
- **RMI**, CORBA, DCOM, SOAP

RMI

- available only for JAVA
- RMI applications are usualy client-server
- server creates remote objects and make them accessible
- client retrievs these remote objects and invoke methods on them
- RMI uses registry
- enables to load class definition at runtime

Remote Objects

- can be passed to another (client) VM by reference
- client VM can invoke methods on these objects
- computation takes place on server VM
- must extend the Remote interface
- every interface method must throw a RemoteException

Interface Definition

```
package compute;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Compute extends Remote {
    <T> T perform(Task<T> t) throws RemoteException;
}
```

Remarks

- any object implementing Remote interface can be remote
- RemoteException is thrown by the RMI system
- checked exception
- client is able to invoke only methods defined in this interface

Task Interface

Task Interface

```
public interface Task<T> {
    T execute();
}
```

- these two interfaces must be available both in client and server
- they define a **protocol** for client-server communication

Remarks

- this way the server can compute any task that implements the Task interface
- even those that are not yet known at server compilation time
- RMI fetches their class definition at runtime
- any data passed between the client and the server must be either primitive type or an object implementing Serializable or Remote

Server Implementation

- create remote object class definition
- create server application
- server must create all remote objects and register them before they can be retrieved by a client

Remote Object Class Definition

Class Definition

```
public class ComputeEngine implements Compute {

    public ComputeEngine() {
        super();
    }

    public <T> T perform(Task<T> t) {
        return t.execute();
    }

    public void anotherMethod() {
// do something
    }
}
```

Remarks

- constructor and *anotherMethod* can not be invoked by the client
- restriction on input and output parameters of remote methods

Server Main Program

```
public static void main(String[] args) {
  if (System.getSecurityManager() == null) {
    System.setSecurityManager(new SecurityManager());
  }
  try {
    String name = "Compute";
    Compute engine = new ComputeEngine();
    Compute stub =
      (Compute) UnicastRemoteObject.exportObject(engine, 0);
    Registry registry = LocateRegistry.getRegistry();
    registry.rebind(name, stub);
    System.out.println("ComputeEngine bound");
  } catch (Exception e) {
    System.err.println("ComputeEngine exception:");
    e.printStackTrace();
  }
}
```

Remarks

- *exportObject* creates stub for a remote object
- only stub is transferred to the client, actual remote object never leaves its VM
- second parameter specifies on which port the object listens for method invocations
- stub is then added to the registry to be available to the clients
- *getRegistry()* method takes one argument - port on which the registry is running on, default 1099
- *bind*, *unbind* and *rebind* methods can be called only on registry running on the same host
- these methods throw RemoteException - need to handle it

Defining Task implementation

Class Definition

```
public class FirstTask implements Task<Double>,Serializable {

    public FirstTask() {
        // ...
    }

    public Double execute() {
        // ...
    }
}
```

- must implement *Serializable*

Client Main Code

Class Definition

```
public static void main(String args[]) {
  if (System.getSecurityManager() == null) {
    System.setSecurityManager(new SecurityManager());
  }
  try {
    String name = "Compute";
    Registry registry = LocateRegistry.getRegistry(args[0]);
    Compute comp = (Compute) registry.lookup(name);
    FirstTask task = new FirstTask();
    Double result = comp.executeTask(task);
    System.out.println(result);
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Remarks

- primitive, objects implementing Serializable or Remote
- implications
- security considerations

Running Application

- classes (both client and server must be publicly available)
- need to define security rules

Server Policy File

```
grant codeBase "classes_location" {
    permission java.security.AllPermission;
};
```

Client Policy File

```
grant codeBase "classes_location" {
    permission java.security.AllPermission;
};
```