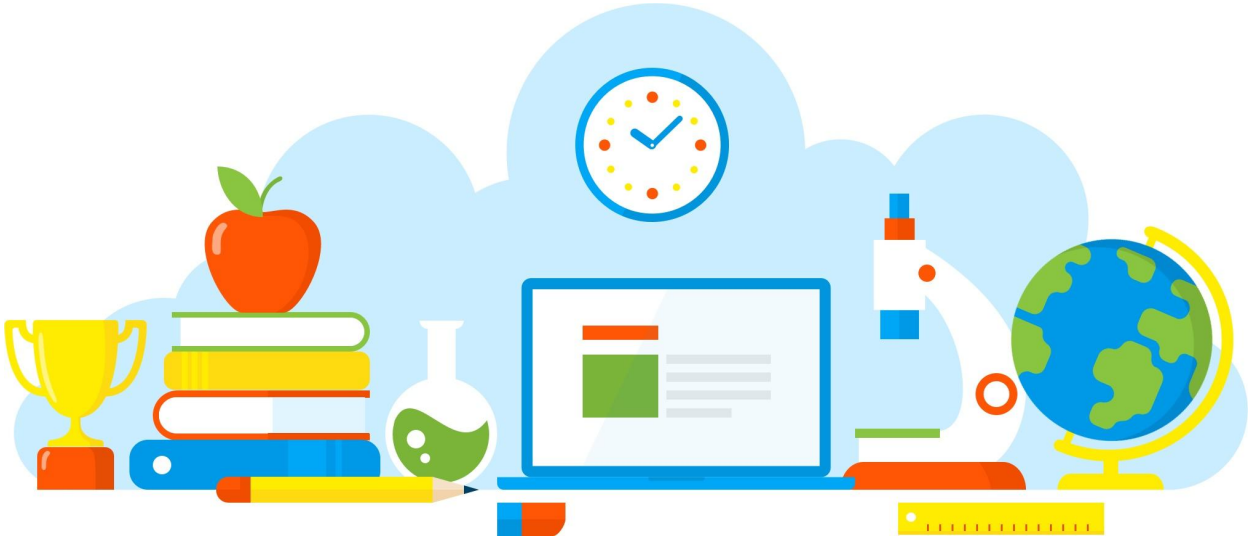


JAVA RMI REPORT

6NTCM001W - Applied Distributed System Programming

Jan Ayag
W1651125
Dr David Huang
1st April 2021

Introduction

Java RMI or Remote Method Invocation, allows a java object to invoke method on an object running remotely. It provides remote communication between programs and is used in building distributed applications.

A RMI application can be separated into two programs; the Server program and the client program. The server program creates remote objects and allows the client side access so they can invoke methods from it. Whereas, the client side requests for access to the remote objects so they can invoke it for their own use.

Furthermore, in an RMI application, stub and skeleton are two important features that are required for communication between the server and the client. A stub acts as a gateway for the client program, it is located in the client side and communicates with the skeleton; while the skeleton is located in the parent process, it is responsible for handling the client's stub and passes it to the server’s remote object.

Methodology

Creating a RMI application

The following are the minimal steps needed to build a working RMI application:

- Define a Remote Interface
- Implement the Remote Interface
- Create and start the remote application(server application)
- Create and start the client application

Based on the steps specified above, the RMI application created for this coursework follows the following steps as guidelines but with minute changes:

```java
package Server;

import java.io.File;
import java.rmi.*;

/**
 *
 * @author xeals
 */
public interface ServerInt extends Remote {
    public void saveFile(File file, String msg) throws RemoteException;
}
```

The code shown above depicts how a remote interface is defined and the methods shown above are the methods that can be invoked remotely by the client application. For the reason that the client application communicates with the remote interface and not the java classes implementing it. Furthermore, to create/define a remote interface, it must extend “Remote” from the “java.rmi” package.

```java
package Server;

import Client.ChatView;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import java.rmi.server.*;
import java.rmi.*;

/**
 *
 * @author xeals
 */
public class ServerImp extends UnicastRemoteObject implements ServerInt {
    private static final long serialVersionUID = 1L;
    ChatView view;

    protected ServerImp () throws RemoteException {
        super();
    }

    @Override
    public void saveFile(File file, String msg) throws RemoteException {
        if (file == null) {
            throw new IllegalArgumentException("Illegal argument, File cannot be null");
        } if (msg == null) {
            throw new IllegalArgumentException("Illegal argument, Msg cannot be null");
        } try (BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(file.getPath())))) {
            writer.write(msg);
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}
```

The code shown above is how the remote interface is implemented. This is done by extending UnicastRemoteObject from the java.rmi.server.UnicastRemoteObject library.

```java
package Server;

import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RMISecurityManager;

/**
 *
 * @author xeals
 */
public class Server {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.setProperty("java.security.policy", "file:./server.policy");

        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }

        try {
            String welcome = "Starting the Server...\n...Please wait\n";
            System.out.println(welcome);
            ServerImp server = new ServerImp();
            System.out.println("....Server successfully started...\n");
            Registry registry = LocateRegistry.createRegistry(8000);
            registry.rebind("Server", server);
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}
```

Above depicts the main java class for the server side of the RMI application. The server uses rebind() to host the RMI services, the first value in rebind() is the name of the remote object and the second value is the of the java instance.

```java
package Client;

import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RMISecurityManager;

import Server.ServerInt;

/**
 *
 * @author xeals
 */
public class Client {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.setProperty("java.security.policy", "file:./client.policy");

        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        try {
            Registry registry = LocateRegistry.getRegistry("localhost", 8000);
            ServerInt si = (ServerInt)registry.lookup("Server");
            new ChatView(si);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
```

Similarly, the picture above depicts the main java class for the client side of the RMI application. Here all remote method invocation is done, and the client uses lookup() to communicate with the server application and if the remote invocation is successful a reference is returned to the RMI interface as shown above.

Finally, in order to run the application, the following steps must be followed: all java files must be compiled; start RMI registry; run the server application and lastly run the client application in a separate terminal.

Conclusion

To conclude, Java RMI is a very interesting topic and a challenging one; one must understand how the java classes operate and how they communicate with other java files. Furthermore, a passable amount of knowledge is required to create and implement a working RMI application as it involves methods that have been decrepitated as RMI is a very old technology that is mostly unused nowadays.

Source Code

Below is the source code for the RMI application created for this coursework project:

Client

```
package Client;

import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RMISecurityManager;

import Server.ServerInt;

/**
 *
 * @author xeals
 */
public class Client {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.setProperty("java.security.policy", "file:./client.policy");

        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        try {
            Registry registry = LocateRegistry.getRegistry("localhost", 8000);
            ServerInt si = (ServerInt)registry.lookup("Server");
            new ChatView(si);
        } catch (Exception e) {
            System.out.println(e);
        }
    }

}
```

Server

```
package Server;

import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RMISecurityManager;

/**
 *
 * @author xeals
 */
public class Server {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.setProperty("java.security.policy", "file:./server.policy");

        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }

        try {
            String welcome = "Starting the Server...\n...Please wait\n";
            System.out.println(welcome);
            ServerImp server = new ServerImp();
            System.out.println("....Server successfully started...\n");
            Registry registry = LocateRegistry.createRegistry(8000);
            registry.rebind("Server", server);
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}
```

Server Interface

```java
package Server;

import java.io.File;
import java.rmi.*;

/**
 *
 * @author xeals
 */
public interface ServerInt extends Remote {
    public void saveFile(File file, String msg) throws RemoteException;
}
```

Server Interface Implementation

```java
package Server;

import Client.ChatView;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import java.rmi.server.*;
import java.rmi.*;

/**
 *
 * @author xeals
 */
public class ServerImp extends UnicastRemoteObject implements ServerInt {
    private static final long serialVersionUID = 1L;
    ChatView view;

    protected ServerImp () throws RemoteException {
        super();
    }

    @Override
    public void saveFile(File file, String msg) throws RemoteException {
        if (file == null) {
            throw new IllegalArgumentException("Illegal argument, File cannot be null");
        } if (msg == null) {
            throw new IllegalArgumentException("Illegal argument, Msg cannot be null");
        } try (BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(file.getPath())))) {
            writer.write(msg);
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}
```

Serializable Object

```java
package Server;

import java.io.Serializable;
import java.util.Date;

/**
 *
 * @author xeals
 */
public class Message implements Serializable {
    private String message;
    private Date date;

    public String getMsg() {
        return message;
    }

    public void setMsg(String message) {
        this.message = message;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }
}
```

GUI

Client GUI

```java
package Client;

import Server.Message;
import Server.ServerInt;
import java.io.File;
import java.rmi.RemoteException;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

/**
 *
 * @author xeals
 */
public class ChatView extends javax.swing.JFrame {
        private static final long serialVersionUID = 1L;
        ServerInt server;
```

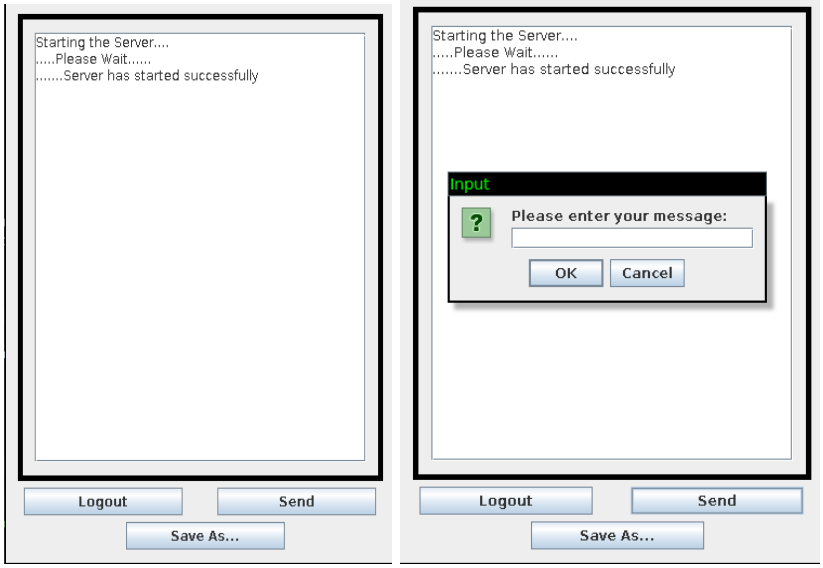
```
        String msg;
        Message m = new Message();
        /**
        * Creates new form ChatView
        * @param server
        */
        public ChatView(ServerInt server) {
        initComponents();
        this.server = server;
        execute();
        }

        /**
        * This method is called from within the constructor to initialize the form.
        * WARNING: Do NOT modify this code. The content of this method is always
        * regenerated by the Form Editor.
        */
        @SuppressWarnings("unchecked")
        // <editor-fold defaultstate="collapsed" desc="Generated Code">
        private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        textArea = new javax.swing.JTextArea();
        Logout = new javax.swing.JButton();
        saveButton = new javax.swing.JButton();
        sendButton = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jPanel1.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0, 0, 0),
5));
        jPanel1.setName("Log Viewer"); // NOI18N

        textArea.setEditable(false);
        textArea.setColumns(20);
        textArea.setLineWrap(true);
        textArea.setRows(5);
        jScrollPane1.setViewportView(textArea);

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
        jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jScrollPane1)
        .addContainerGap())
```

```
        );
        jPanel1Layout.setVerticalGroup(
        jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 388,
Short.MAX_VALUE)
        .addContainerGap())
        );

        Logout.setText("Logout");
        Logout.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
        LogoutActionPerformed(evt);
        }
        });

        saveButton.setText("Save As...");
        saveButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
        saveButtonActionPerformed(evt);
        }
        });

        sendButton.setText("Send");
        sendButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
        sendButtonActionPerformed(evt);
        }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())
                .addGroup(layout.createSequentialGroup()
                .addGap(6, 6, 6)
                .addComponent(Logout, javax.swing.GroupLayout.PREFERRED_SIZE, 160,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 35,
Short.MAX_VALUE)
```

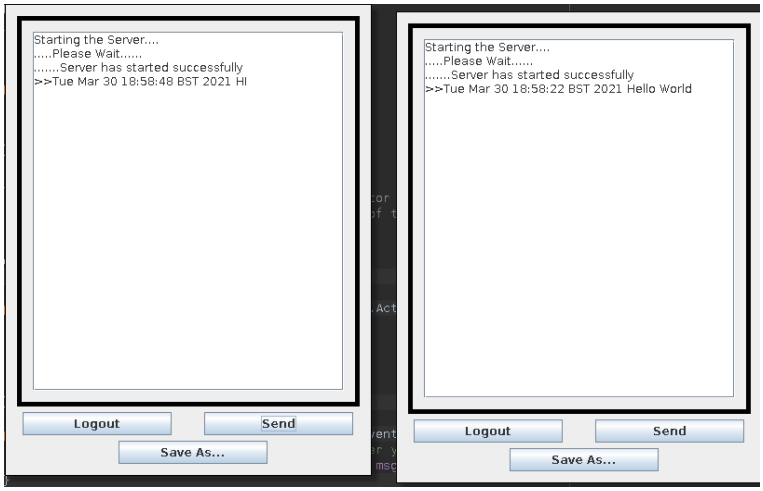
```
                    .addComponent(sendButton, javax.swing.GroupLayout.PREFERRED_SIZE, 160,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(19, 19, 19))))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(saveButton, javax.swing.GroupLayout.PREFERRED_SIZE, 160,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(111, 111, 111))
            );
            layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(Logout)
                    .addComponent(sendButton))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(saveButton)
            .addContainerGap())
            );

            pack();
            }// </editor-fold>

            private void LogoutActionPerformed(java.awt.event.ActionEvent evt) {

            try {
            System.exit(0);
            } catch (Exception e){
            System.err.println(e);
            }
            }

            private void sendButtonActionPerformed(java.awt.event.ActionEvent evt) {

            msg = JOptionPane.showInputDialog("Please enter your message:");
            textArea.append("\n>>" + (new Date()) + " " + msg);
            }

            private void saveButtonActionPerformed(java.awt.event.ActionEvent evt) {

            File file = new File("RMIServerLog.txt");
            try {
            msg = textArea.getText();
            server.saveFile(file, msg);
```

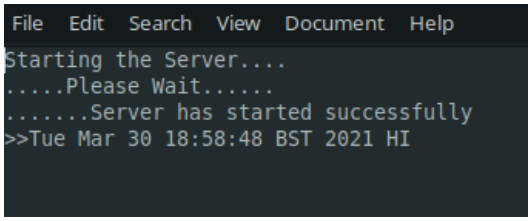
```
        } catch (RemoteException ex) {
        Logger.getLogger(ChatView.class.getName()).log(Level.SEVERE, null, ex);
        }
        }

        private void execute() {
        this.setVisible(true);
        this.setTitle("Server Log Screen");
        textArea.append("Starting the Server....\n" + ".....Please Wait.....\n" + ".......Server has started
successfully");
        }


        // Variables declaration - do not modify
        private javax.swing.JButton Logout;
        private javax.swing.JPanel jPanel1;
        private javax.swing.JScrollPane jScrollPane1;
        private javax.swing.JButton saveButton;
        private javax.swing.JButton sendButton;
        public javax.swing.JTextArea textArea;
        // End of variables declaration
}
```

Outputs:




Starting the Server....
.....Please Wait.....
.......Server has started successfully
>>Tue Mar 30 18:58:22 BST 2021 Hello World
Logout
Send
Save As...


Starting the Server....
.....Please Wait.....
.......Server has started successfully
>>Tue Mar 30 18:58:48 BST 2021 HI
Logout
Send
Save As...
Starting the Server....
.....Please Wait.....
.......Server has started successfully
>>Tue Mar 30 18:58:22 BST 2021 Hello World
Logout
Send
Save As...


File Edit Search View Document Help
Starting the Server....
.....Please Wait.....
.......Server has started successfully
>>Tue Mar 30 18:58:48 BST 2021 HI