The Design and Implementation of the Java Based Remote File Management System

CHENG ZENG, WEI DU, AIMIN WANG, TIANYUAN XIAO
Department of Automation
Tsinghua University
RM 219, BLDG 24, Tsinghua University, Beijing, 100084, China
CHINA

Abstract:-This paper designs a set of Java Sockets based on application programming interface (API) to realize the remote file management system (RFMS) running on different operating systems, and realizes a Client/Server structure based on RFMS with graphic interface. This set of API is similar to the local file operation classes of the Sun JDK, with which the programmers can realize a RFMS satisfying their requirements conveniently and rapidly.

Key words: -Java; Socket; JDK; File Management System

1 Introduction

With the advance of network technologies, remote education, management and technologies support become popular. Remote file management system (RFMS) contributes most to this trend.

The current RFMS includes two kinds as following [1]:

1) RFMS based on FTP protocol;
2) RFMS inherited from the operating system (OS), used only in the homogeneous OS.

These two RFMS have the same shortcoming that they can only be used in the homogeneous OS, and will cause problems when transferring in different platforms.

Java is an excellent cross-platform program language. The file class of Java will not be restricted to specific platforms. With this attribute, we design an application-programming interface (API), which is similar to the Java local file class. Using this API, we implement a graphical RFMS based on client/sever structure, which can be installed in different operating systems and will facilitate the implementation of user-specific RFMS.

2 The principle of design

2.1 The structure of system

The system is based on traditional Client/Sever two layers structure. The main design principle includes:

1) Both the client and sever are written in pure Java to ensure the cross-platform attribute;
2) The client and sever communicate each other using Sockets.

The program running in the sever listens the request from the client, i.e. the administrator of the RFMS. The communication process can be show in Fig. 1.

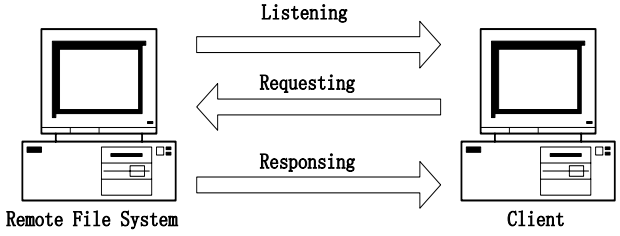

Fig.1:The process of communication

2.2 Java Socket

The communication through stream socket is based on connection, i.e. before starting

communication, the client and sever authenticate their identities each other and then create a specific virtual tunnel to connect, which will be cancelled when the communication is finished [2]. The socket communication is realized through the classes of java.net.Socket and java.net.ServerSocket. The following code shows the process of creating a sever monitoring program to communicate with a client:

```
…...
// to create a monitor server in port 8000
ServerSocket listener = new
ServerSocket(8000);
//blocked till catching a request form client
Socket client = listener.accept();
// to get the output stream to the client
OutputStream out = client.getOutputStream();
//to get the input stream from the client
InputStream in = client.getInputStream();
String strHello = “Hello!”;
byte[] hello = strHello.getBytes();
//to send a message” Hello!” to the client
out.write(hello);
……
```

We can find that the monitoring program will get output stream from severs and input stream from clients after catching the clients' request, and then this program can send or fetch data from clients. In this way, the RFMS gets technologies support[3].

2.3 Java file class

Java has various classes to support the file operation[4], including File Class, FileInputStream Class, FileOutputStream Class, FileSystemView Class, RandomAccessFile Class, etc. File Class is in charge of managing the disk file and directory; FileInputStream and FileOutputStream Class can manage the output and input of files respectively; FileSystemView Class can implement the file operations based on operating system[5]. The following codes show how to tell if a drive is a floppy disk drive using File class and FileSystemView class:

```
FileSystemView fs=
   FileSystemView.getFileSystemView();
File drive = new File(“A:\\”);
if(fs.isFloppyDrive(drive))
   System.out.println(“a: is a floppy disk drive.”);
else
   System.out.println(“a: is not a floppy disk drive”);
```

These classes can easily be used, but only restricted to local file operations[6]. To inherit these benefits to support RFMS, we design an API similar to these classes. In this way, users can easily and efficiently write programs to support RFMS in the same way with using local classes.

3 The building of System

The system is composed of three parts: a related API, sever units and client programs.

3.1 The self-designed API

We design this API following the class of local file operation provided by Sun Corp. However, there is some difference as showed in table 1:

Type / Description	Sun JDK	RFMS
File	java.io.File	org.rmfs.File
Read file stream	java.io.FileInputStream	org.rmfs. FileInputStream
Write file stream	java.io.FileOutputStream	org.rmfs. FileOutputStream
Random read/write	java.io.RandomAccessFile	org.rmfs. RandomAccessFile
File system	javax.swing.filechooser.FileSystemView	org.rmfs. FileSystem

Table1:the difference between the two classes

The principle to design this API is to keep the interface homogenous with Sun JDK's file operation classes and shield all the Socket operations in the bottom layer, which can make the user handle the RFMS like what hey do with java.io package[7].

The kernel classes of this API are FileSystem and File class. The relationship between those classes and with J2SE (Java 2 Standard Edition) is showed in fig.2:

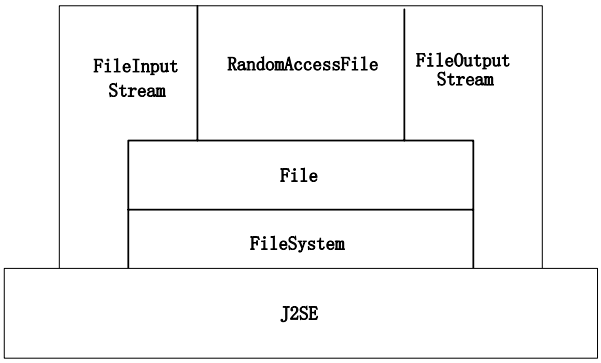

Fig. 2:The relationship between classes

1) org.rmfs.FileSystem

This class is designed corresponding to Sun JDK's javax.swing.filechooser.FileSystemView class and inherit to be one of its sub classes.

FileSystem class describes a remote file system used in client software, with whose static method getFileSystem() we can get an objective of FileSystem class:

```
public class FileSystem extends
    FileSystemView{
    //to connect with the remote file system socket
    private Socket socket;
    //to output to the remote file system's stream
    private OutputStream out;
    //to get the remote file system's input stream
    private InputStream in;
    public static synchronized FileSystem
    getFileSystem(String host,int port){
        ……
        //to connect with the remote file system using
        //the allocated host pc and port
        socket = new Socket(host,port);
        out = socket.getOutputStream();
        in = socket.getInputStream();
        ……
    }
    // to read bytes from a file input stream
    int read(FileInputStream fis,in,byte[] b,int off,int
    len){
        …….
        String cmd = "READ "+fis.getID()+" END";
        out.write(cmd.getBytes[]);
        in.read(b,off,len);
        ……
    }
    // to send bytes to file output stream
    void write(FileOutputStream fos,byte[] b,int
    off,int len){
        ….
        String cmd = "WRITE "+fos.getID()+" END";
        out.write(cmd.getBytes[]);
        out.write(b,off,len);
        ….
    }
    ……
}
```

The fields of the class:

- private Socket socket;

This field describes the connection with the remote file system's socket. All the data between clients and remote system transfer through this socket connection.

- private OutputStream out;

This field describes the output stream sent to remote file system. The command and data sent to remote system are transferred by calling the write method of this stream.

- private InputStream in;

This field describes the input stream from remote file system. The acknowledge command and data of remote system can be got by calling the read method of this stream.

The methods of the class:

- FileSystem getFileSystem(String host,int port)

getFileSystem() method can be used to get an objective of FileSystem class. This method has two parameters: one is to point out the IP of remote host pc and the other is to point out a port number, which is used by sever to monitor the client request. So the

client can call this method to get the object describing the remote file system.

In fact, a remote file system object is a connection with the remote file system's socket. When user call the getFileSystem(), the following code is executed:

String cmd = "WRITE "+fos.getID()+" END";
out.write(cmd.getBytes[]);

"WRITE" means send data to remote file system, fos.getID is the unique flag of FileOutputStream and "END" means the command's over. When receiving this command, the remote file system will be ready to read the data from clients.

Similar to read, the method of write does not have the variable of public. In fact, this method is provided for the call of write () in class of FileOutputStream.

The client can get an object from remote file system using following code:

CODE1::To get a object from remote file system

```
String host = "166.111.167.128";
int port = 8001;
FileSystem remoteFS
    = FileSystem.getFileSystem(host,port);
```

FileSystem class is the basis of remote file system, and all the other classes have to get a realization of FileSystem class.

2) org.rmfs.File

This class corresponds the java.io.File class in Sun JDK, and is used to describe a file or directory in remote system.

```
public class File implements
java.io.Serializable,java.lang.Comparable{
// to describe the current remote file system
   private FileSystem fs;
   //to describe the path of file in this file system
   private String path;
  //to get the current file system where the file
  //locates
   public FileSystem getFileSystem() {
         return fs;
   }
   //constructor
   public File(FileSystem fs,String path){
      if (path == null || fs==null) {
            throw new NullPointerException();
      }
      this.fs=fs;
      this.path=path;
   }
   ……
}
```

The fields of the class:

- private FileSystem fs;

The current remote file system

- private String path;

Describing the absolute path of remote file in the RFMS.

The methods of the class:

- public FileSystem getFileSystem()

return domain fs;

- public File(FileSystem fs,String path)

Constructor, the two parameters point out fs and path;

In addition, file class provides the methods of listFiles(), renameTo(), delete(), etc. which is similar to the java.io.File class. These methods can easily rename and delete the remote files. RemoteFs is a object of remote file system got from CODE1.

CODE2: to get the object of remote files

```
File helloFile = new
   File(remoteFS,"C:\\hello.txt");
```

3) org.rmfs. FileInputStream

This class corresponds the java.io.FileInputStream class in Sun JDK, and is used to get data from a remote file. We will only introduce this class's methods and fields.

```
public class FileInputStream extends
java.io.InputStream {……}
```

The fields of the class:

- private FileSystem fs;

The current remote file system

- private File file;

The remote file object;

The methods of the class:

- public FileInputStream(File file)

Constructor,setting the field fs and file;

- public int read(byte[] b, int off, int len)

throws IOException

Reading data from the input stream; In fact, this method calls the read() method of the FileSystem class;

- public long getID()

Returning the ID of the input stream. When sending commands to the remote file system, the remote file system is able to know the file of which the content should be sent to the client according to this ID;

4) java.io.FileOutputStream

This class is related to the java.io.FileOutputStream class in Sun JDK and is used to write data to a remote file. We will only introduce this class's methods and fields.

public class FileOutputStream extends java.io.OutputStream{……}

The fields of the class:

- private FileSystem fs;

The current remote file system;

- private File file;

The remote file object;

The methods of the class:

- public FileOutputStream(File file)

This is the constructor setting fs and file;

- public void write(byte[] b, int off, int len) throws IOException

This method writes data to the output stream. In fact this method calls the FileSystem class's write() method;

- public long getID()

This method returns the ID of the output stream. When the client sends commands to the remote file system the remote file system is able to know the file that should be written according to this ID.

```
String addToFile = "Hello,remote file!";
//transferring the string to a byte array to match the
//parameter's criterion of the write() method
byte[] b = addToFile.getBytes();
fos.write(b,0,b.length);
```

This code writes the sentence "Hello,remote file!" to the remote file C:\\hello.txt.

5) org.rmfs. RandomAccessFile

This class corresponds the java.io.RandomAccessFile class in Sun JDK, and the functions of it are similar to java.io.RandomAccessFile class. Because the 4 classes introduced before are able to realize the remote file management system, this class will not be introduced particularly.

3.2 The Software in Server

The software in sever provide the clients access to remote file system. Actually, it is a monitor program running in some port. There is a domain socket in the FileSystem class, which actually is a object connected with sever program who directly communicates with FileSystem class and transfer data. When sever catching a request for connection, that means a client is trying to connect the sever. Consequently, the sever will create a thread to handle with the communication with the client.

3.3 The Software in Clients

The program in clients uses the five classes referred above to communicate with RFMS. A typical communication process includes:

Call the method of FileSystem getFileSystem(String host,int port) to get a object of remote file system

Call the method of File (FileSystem fs,String path) to get a remote file system's object

Finish the related operations, such as delete files, write data to files, etc.

In this way, user can easily realize network communication with a complete shield of detail operations. The period of development can be shortened while the complexity of program can be decreased dramatically.

4 Case Study

Based on the principle and structure referred above, we implement a RFMS with a graphical interface. Fig 3 shows the interface of sever program, Fig 4 and 5 show the interface of client program. As Fig.3, the program can start the RFMS after pointing out the port number while Fig. 4 and 5 show the access and management to RFMS after pointing out the IP and port number of RFMS.

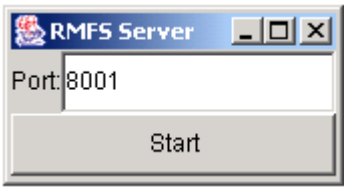

Fig.3:The interface of sever--boot

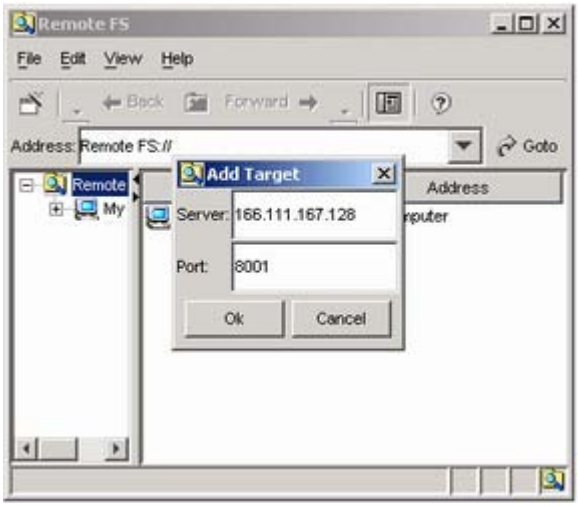

Fig. 4 the interface of clients—point out the IP and port number of RFMS

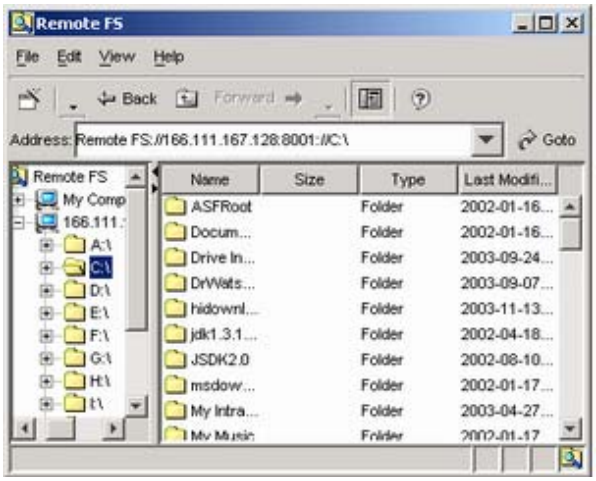

Fig. 5:interface of clients- connecting with RFMS

5 Conclusions

We design an API class similar to the Sun class of standard local file operation. Based on this API class, we implement a RFMS with a structure of Client/Sever. Because of the homogeneousness between the interface of this API and that of Sun' file operation class, and the complete shield of bottom socket operations, user can easily and efficiently implement a specific RFMS with the aid of this API.

References:

[1] Shinzo Doi, Atsuhiro Tsuji,Yukiko Itoh,Kouji Kubota, Tsutomu Tanaka, Real-Time Remote File System for Multimedia Application, Multimedia and Expo, *ICME 2000. 2000 IEEE International Conference* , Volume: 3 , 2000 ,pp.1727 - 1730

[2] Bruce Eckel, *Thinking in Java*, Prentice Hall, 2000

[3] O'Connell, M., Nixon, P., JFS: a secure distributed file system for network computers, *EUROMICRO Conference*, 1999. Proceedings. 25th , Volume: 2,pp.450-453

[4] Jackson, D., Waingold, A.,Lightweight extraction of object models from bytecode, *Software Engineering, 1999. Proceedings of the 1999 International Conference*,pp.70-73

[5] Ahuja, S.P., Quintao, R., Performance evaluation of Java RMI: a distributed object architecture for Internet based applications, *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000. Proceedings. 8th International Symposium*, pp.565-569

[6] Ekwall, R.,Urban, M., Robust TCP connections for fault tolerant computing, *Parallel and Distributed Systems, 2002. Proceedings. Ninth International Conference*, pp.503-505

[7] Ng, K.T., Siu, Y.M., The development of a betting system on the Internet, *Information Technology: Coding and Computing, 2000. Proceedings. International Conference*, pp.308-309