

# 六步教你学会简单 RMI（上）

江苏 无锡 缪小东

（以下所有 java 文件、.class 文件和 policy.txt 文件都在 c 盘根目录哦！101.txt 在 c 盘的子目录 11 下哦！一定要放对!!!）

## 一、 定义远程接口

远程接口就是远程机器上可供客户使用的方法的集合。很幸运它用 java 语言的接口表示！我们定义这样一个接口只有一个下载远程机器上的指定名称的文件。

```
//FileServerInterface.java
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface FileServerInterface extends Remote {
    public byte[] download(String Filename)throws RemoteException ;
}
```

## 二、实现远程接口

实现上面远程接口的方法，同时继承 UnicastRemoteObject 类！

```
//FileServerImpl.java
import java.io.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject ;

public class FileServerImpl extends UnicastRemoteObject implements FileServerInterface{
    private static final String initDir = "c://11//";
    public FileServerImpl( ) throws RemoteException{
        super();
    }

    public byte[] download(String filename){
        try{
            File file = new File(initDir + filename);
            byte[] buffer = new byte[(int)file.length()];
            BufferedInputStream bis = new BufferedInputStream(new FileInputStream(file));
            bis.read(buffer,0,buffer.length);
            bis.close();
            return buffer ;
        }catch(Exception e ){
            System.out.println("FileServerImpl:  " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

```

        return null ;
    }
}
}

```

### 三、编写服务器端

1. 创建并安装一个 `RMISecurityManager` 实例；
2. 创建一个远程对象的实例；
3. 使用 `RMI` 注册工具注册该远程实例对象。

```

//FileServer.java
import java.io.*;
import java.rmi.*;

public class FileServer {
    public static void main(String[] args){
        if(System.getSecurityManager()==null ){
            System.setSecurityManager(new RMISecurityManager());
        }
        try{
            FileServerImpl fi = new FileServerImpl("FileServer");
            java.rmi.Naming.rebind("//127.0.0.1/FileServer",fi);
        }catch(Exception e){
            System.out.println("FileServer:  " + e.getMessage());
            e.printStackTrace();
        }
    }
}

```

### 四、编写客户端

```

//FileClient.java
import java.io.*;
import java.rmi.*;

public class FileClient {
    public static void main(String[] args){
        if(args.length != 2 ){
            System.out.println("Usage: java FileClient Filename machinenam");
            System.exit(0);
        }
    }
}

```

```

try{
    String name = "/" + args[1] + "/FileServer";
    FileServerInterface fi = (FileServerInterface)Naming.lookup(name);
    byte[] filedata = fi.download(args[0]);
    File file = new File(args[0]);
    BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream(file.getName()));
    bos.write(filedata,0,filedata.length);
    bos.flush();
    bos.close();
}catch(Exception e){
    System.out.println("FileClent: " + e.getMessage());
    e.printStackTrace();
}
}
}

```

## 五、创建一个 policy 文件

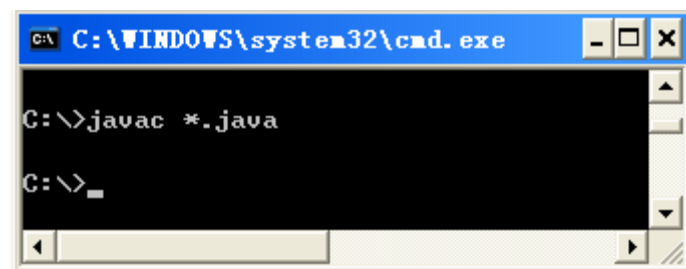
```

//policy.txt
grant{
    permission java.security.AllPermission "" , "" ;
};

```

## 六、运行程序

### 1. javac \*.java



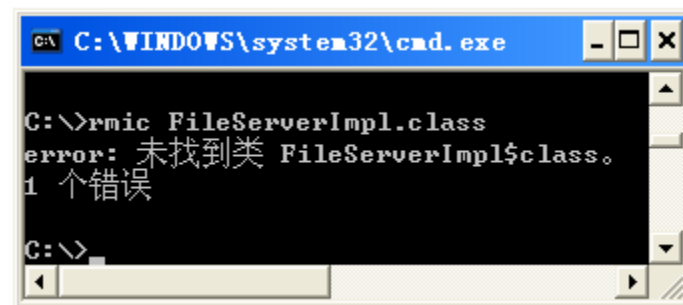
这个都会吧！要么你就一个一个编译好了！

## 2. rmic FileServerImpl



```
C:\WINDOWS\system32\cmd.exe
C:\>rmic FileServerImpl
C:\>
```

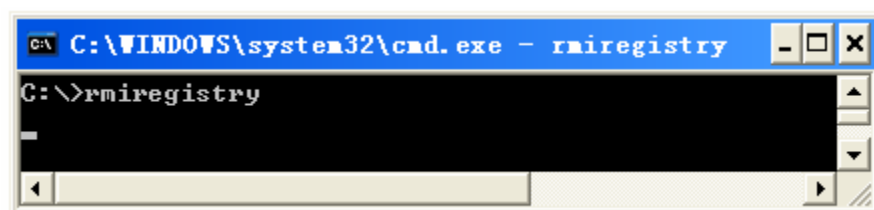
注意哦，不是 FileServerImpl.class 哦！看看下面的错误：



```
C:\WINDOWS\system32\cmd.exe
C:\>rmic FileServerImpl.class
error: 未找到类 FileServerImpl$class。
1 个错误
C:\>
```

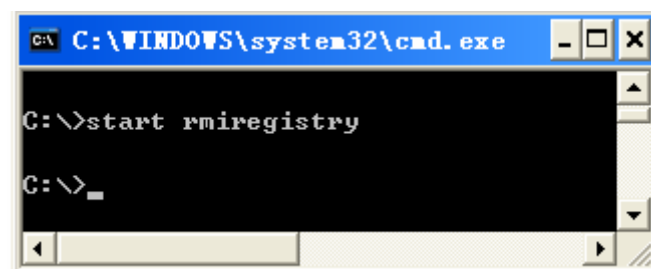
当成你要使用 rmic 命令编译一个 FileServerImpl 的内部类 class 哦! (\$代表什么, 请阅读《Think in java》)

## 3. rmiregistry 或 start rmiregistry



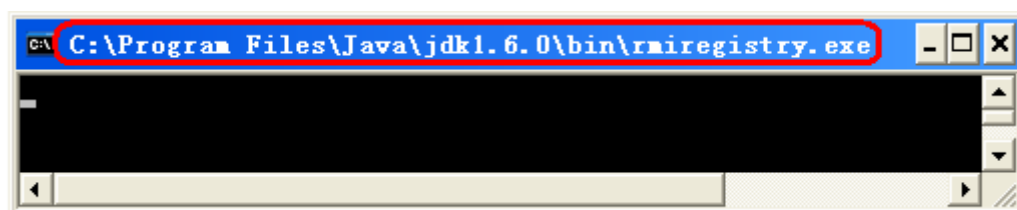
```
C:\WINDOWS\system32\cmd.exe - rmiregistry
C:\>rmiregistry
```

使用 rmiregistry 的命令窗口，启动一个 rmiregistry，本窗口“阻塞”（不太精确哦，就是不能输入其它命令）。



```
C:\WINDOWS\system32\cmd.exe
C:\>start rmiregistry
C:\>
```

使用 start rmiregistry 命令，本窗口出现“提示符”，可以继续输入命令，且跳出一个窗口，也是“阻塞”的哦。表示正在工作，方法没有返回而已。

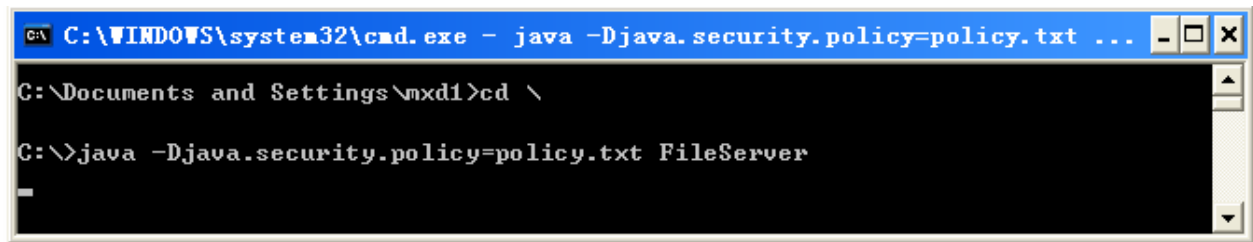


```
C:\Program Files\Java\jdk1.6.0\bin\rmiregistry.exe
```

注意该窗口的标题了没有！其中奥妙，慢慢体会！

#### 4. java -Djava.security.policy=policy.txt FileServer

(使用此命令时 **policy.txt** 文件必须和 **FileServer.class** 在同一目录 (本例 c 盘根目录) 中哦!)



```
C:\WINDOWS\system32\cmd.exe - java -Djava.security.policy=policy.txt ...
C:\Documents and Settings\mxd1>cd \
C:\>java -Djava.security.policy=policy.txt FileServer
```

#### 5. java FileClient 101.txt 127.0.0.1

(使用此命令时 **101.txt** 必须位于 c 盘下 **11** 目录下哦, 要不然没有下面的结果哦!)

看看你的 c 盘根目录, 是不是多了一个 101.txt 啊, 是不是和你放在 11 目录下的 101.txt 一样啊! 成了, 到此结束! ——六步学会简单 RMI。是否很有成就感啊!

(简单地把此代码拷贝过去, 按照上面的顺序做就行了! 立刻行动吧!)

不过以上程序尽管是一个 RMI 的网络程序, 是否太简单了! Java 手册中不是说可以传递 java 对象吗? 要是真的能传递对象那就很强大啦! 好像有些细节还不是很明白吧!

请看下一篇! 满足你以上的需求! 下篇给你讲述各个接口和类的含义, 以及 RMI 程序符合发布 (哪些类位于客户端, 哪些类位于服务器端) 等等!

## 六步教你学会简单 RMI (中)

江苏 无锡 缪小东

首先在你的某个目录下建立两个目录 **client** 和 **server**, 用于存放你的类文件! 主要是为了程序的发布!

### 一、 定义远程接口

```
//FileServerInterface1.java
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface FileServerInterface1 extends Remote {
    public FileCarry download(FileCarry file)throws RemoteException ;
}
```

同样, 它是远程主机提供的客户可以使用的方法的集合! 我们在其中仅仅定义了一个方法 (你可以根据实际需要写你需要的任何方法! 你可以将本程序改造象使用 **ftp** 一样, 先 **list** 一下, 看看服务器提供几个可供下载的文件, 然后再下载相应的文件! 很简单, 自己试试吧!), 用于从服务器端下载某一个

文件！（完成与上面一样的功能，主要用于演示 RMI 中传递对象！）

注意：

1. RMI 中使用的接口必须继承 Remote 接口（直接或间接地，RMI 中实现此接口的对象很多一般供编程人员使用的为 RemoteObject、RemoteServer、Activatable 和 UnicastRemoteObject 等）；
2. Remote 接口在 java.rmi 包中，它是一个没有任何方法的标识接口；
3. 继承 Remote 接口中的方法，一定要抛出 RemoteException 异常——它是 RMI 运行时环境发生异常时抛出的异常，与客户实际可能抛出的异常无关。（不过你也可以抛出其父异常哦，如 IOException、甚至 Exception！）

## 二、 定义在 RMI 中传递对象

```
import java.io.*;
```

```
public class FileCarry implements Serializable {
    private String fileName ;
    private String content ;
    public FileCarry(String name){
        fileName = name ;
    }
    public void setFileName(String filename){
        this.fileName = filename ;
    }
    public String getFileName(){
        return this.fileName ;
    }
    public void setContent(String content){
        this.content = content ;
    }
    public String getContent(){
        return this.content;
    }
}
```

一个简单的类，它有两个属性：String 类型的属性 fileName 表示客户向服务器请求下载的文件名称，String 类型的属性 content 表示服务器在接受到客户请求后，传递给客户的文件的内容。其对应的方法也比较简单！一定要注意：用于在 RMI 中传递的对象必须实现 java.io. Serializable 接口（该接口为一个没有任何方法的标识接口）。在 RMI 中主要有两种类型的特殊对象：1. 实现 java.io. Serializable 接口的对象，这些对象主要是在客户和服务器间传输的对象（来来回回的走城门啊!!），一般它属于所谓的值拷贝（value copy）；2. 实现 java.rmi.Remote 接口的对象，这些对象主要是存在于服务器端，供客户使用的对象，客户一般在客户端得到此对象的引用，属于引用拷贝!!（暂且这么说吧!）

（对象的序列有一套自己的机制，有兴趣的朋友可以查阅 IO 相关的资料。不属于本文讨论的范围!）

### 三、 实现远程接口

```
//FileServerImpl1.java
import java.io.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject ;

public class FileServerImpl1 extends UnicastRemoteObject implements FileServerInterface1{
    private   File initDir ;                //定义服务器端的 root directory
    public   FileServerImpl1(File dir) throws RemoteException{
        super();
        initDir = dir ;
    }

    public FileCarry download(FileCarry fileCarry){                //服务器端的方法实现哦！
        try{
            String filename = fileCarry.getFileName();                //得到客户端请求的文件名
            File file = new File(initDir+"/"+filename);                //在服务器端查找请求下载的文件
            BufferedReader br = new BufferedReader (new InputStreamReader(new FileInputStream(file)));
            String line = null;
            StringBuffer sb = new StringBuffer();
            while((line=br.readLine())!= null){
                sb.append(line);
            }
            br.close();
            fileCarry.setContent(sb.toString());
            //以上几步在服务器端完成，请求文件的读取，存入请求对象的String 类型的 content 域中
            return fileCarry;
            //向客户返回此fileCarry 对象。（实际方法设计时，请求对象和返回对象是不同的！
            //读者可以自己设计一下哦！就是一个实现 Seriziable 接口的类罢了）
        }catch(Exception e ){
            System.out.println("FileServerImpl:  " + e.getMessage());
            e.printStackTrace();
            return null ;
        }
    }
}
```

**//以上远程接口的实现相当简单。1.你可以在此方法中向写其它程序一样利用服务器端的任何资源  
//做你想做的任何事。比如调用其它java 类库、与其它机器建立 Socket 连接、调用本地代码、调用  
//远程数据库等等。ObjectWeb 给出了一个用RMI 调用本地数据库的框架，有兴趣的读者可以自行研究！  
//2.你甚至可以在该远程接口实现的代码中，使用RMI 调用客户端——怎么做啊？！  
//不就是Naming.lookup (“XXX ”)，此时你的客户端也必须也启动名称服务器，  
//同时代码中也必须将服务器要查找的对象使用Naming.bind (“”,””) 绑定哦！  
//以上的两个例子你会了，上面的互为服务器也很好建立了吧！！  
//这就是所谓的双向通信，无所谓谁是服务器、谁是客户端！！**

## 四、 编写服务器端

```
//FileServer1.java
import java.io.*;
import java.rmi.*;

public class FileServer1 {
    public static void main(String[] args){
        if(System.getSecurityManager()==null ){
            System.setSecurityManager(new RMISecurityManager());
        } //服务器端的安全管理器，主要是服务器端使用 Socket 要具有权限
        try{
            File serverDir = new File("c://server/txt//"); //看好了，服务器的root dir 在 c://server/txt//哦
                                                    //你可供下载的文件就放在该目录下哦

            FileServerImpl1 fi = new FileServerImpl1(serverDir); //创建远程实现对象
            java.rmi.Naming.rebind("//127.0.0.1/FileServer1",fi); //绑定它，供客户使用

            //1.这种绑定一般在服务器端使用java.rmi.Naming.rebind 方法完成！
            //第一个参数本地 IP 和绑定对象名，一般使用 127.0.0.1 表示本机，加一个对象名
            //第二个参数是要 export 给客户的那个实现远程接口的远程远程对象！
            //该对象会在某个端口，监听客户调用哦！
            //2.当然你也可以将该远程对象绑定到其它远程主机哦！
            //3.bind 和 rebind 的区别是：
            //    在已经为一个名字绑定一个对象，再向此名称绑定（bind）对象，会抛出异常
            //    而使用 rebind 不会抛出异常
            //4.rebind 中的名称使用极其灵活：格式为[RMI]//[host][:port]/name
            //其中带括号的可以省略哦！RMI 可以省略，意义不用说了；host 省略，代表本地
            //    port 省略，代表 default port 1099，你可以使用自己的端口哦
            //绑定对象的名称一定不能省略，且客户查询时必须和服务器端的此名称一至！
            //使用其它 port 时，启动 rmiregistry 也必须带上同样端口号！
            //rmi 先启动，服务器再启动，最后客户端启动，都可以带上一至的 port 值
            //先启动 rmiregistry，再调用服务器程序，会在 rmiregistry 上注册上该对象
            //客户机 lookup 此对象时，必须指定该服务器的 ip 地址哦（同上面的格式）！
            //一定要记好端口和名称

        }catch(Exception e){
            System.out.println("FileServer1:  " + e.getMessage());
            e.printStackTrace();
        }
    }
}

//以上这段代码一般是和“实现远程接口”放在一起的，怎么放啊？！
//将 main 方法全部拷贝过去不就行了吗！！
```

## 五、 编写客户端

```
//FileClient.java
```



```

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

public class FileClient1 {
    public static void main(String[] args){
        if(args.length != 2 ){
            System.out.println("Usage: java FileClient Filename machinename");
            //必须有参数 1. 客户要下载的文件名; 2. 服务器机器名或 ip 地址。
            //一般还要端口的哦! 怎么改啊——自己可以搞定的!! 试试吧!!!
            System.exit(0);
        }
        try{
            String name = "/" + args[1] + "/FileServer1";
            FileServerInterface1 fi = (FileServerInterface1)Naming.lookup(name);
            //向远程命名服务器查找指定的名称的对象, 记好和远程接口实现的名称要一致哦!
            //Naming.lookup(name); 返回的是啥啊? !!
            //是远程对象吗? 远程对象不是实现 FileServerInterface1 接口吗? 很和逻辑吗!!!
            //绝对不是, 是实现 FileServerInterface1 接口的存根对象!!
            //更多精彩请阅读后续文章!!! (卖关子了哦), 很精彩哦, 与其内部通信协议有关哦
            FileCarry in = new FileCarry(args[0]); //客户请求下载文件的封装对象
            FileCarry out = fi.download(in); //调用远程方法
            //爽吧! 和调用本地方法一样!!
            //除了以上 FileServerInterface1 fi = (FileServerInterface1)Naming.lookup(name);
            //方法是客户端和服务端交互的, 其它基本象是本机的。
            String content = out.getContent(); //本地处理内容
            File file = new File(args[0]);
            BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream(file.getName()));
            byte[] buffer = content.getBytes();
            bos.write(buffer, 0, buffer.length);
            bos.flush();
            bos.close();
            //以上是将远程主机上的内容, 通过 FileCarry 对象, 传输到客户端后,
            //客户端处理的过程。
        } catch (Exception e){
            System.out.println("FileClient: " + e.getMessage());
            e.printStackTrace();
        }
    }
}

```

## 六、 创建一个 policy 文件

```

//policy.txt
grant{

```

```
permission java.security.AllPermission "" , "" ;  
};
```

**//供 Socket 使用的策略文件哦！不过一般不需要这么大权限哦！关于安全的知识请多关注本博客**

## 七、 程序的发布

### 7.1 javac \*.java

先编译所有的 java 文件（你也可以一个一个的编译——只要你喜欢！）

### 7.2 rmic FileServerImpl1

使用 rmi 的编译器编译远程接口的实现类，记好了是实现类不是远程接口！

执行此操作成功后，看看你的目录下是否多了一个文件“FileServerImpl1\_Stub.class”，原来的类名加了一个\_Stub 后缀，很有用的文件哦！它是客户端的存根！同时它也是客户端的核心类，完成客户和服务器的交互，是一个典型的 Proxy 模式的应用！（具体模式可以研究设计模式相关的书籍）

它实现了 *FileServerInterface1* 接口！不信啊，下面是用 *JDecompiler* 反编译的代码片断：

```
public final class FileServerImpl1_Stub extends RemoteStub implements FileServerInterface1, Remote {}
```

相信了吧！！（具体细节，后续文章再谈！）

### 7.3 源代码的发布

刚开始，我就让你在目录下建两个目录：client 和 server。建了吧！（我建在 c 盘根目录下）

1. 将 FileServerInterface1.class 拷贝到 client 目录，然后剪切到 server 目录下。一定记好服务器和客户端都有！为什么呢？首先：客户端代码中是否通过 Naming 的查找方法，查找指定名称的对象，将该对象 downcast 为 FileServerInterface1，你说要不要啊！还有客户有一个存根也实现了此接口，没有该接口你的客户能运行吗？？其次：服务器端的代码“FileServerImpl1”一定要实现此接口，也要此接口的吧！！
2. 将 FileServerImpl\_Stub.class 剪切到 client 目录中。它是客户端的存根，是客户端的 Proxy，主要完成与通信相关的请求参数的 marshal 和返回值的 unmarshal。你说要不要啊！！
3. 将 FileCarry.class 拷贝一份到 client 目录，再剪切到 server 目录。首先：客户使用该对象保存一个请求下载的文件名，肯定要吧！其次：服务器端得到此对象，再将本地文件的内容放入此对象中，肯定又要吧！**总之，实现 Serializable 接口的用于客户和服务器传递数据的对象一般在客户端和服务端都会有此对象。**
4. 将 FileClient1.class 剪切到客户端。
5. 将 FileServerImpl1.class 和 FileServer1.class 剪切到 server 目录中。
6. 将 FileServerImpl1\_Stub.class 剪切到 client 目录中。
7. 将 policy 文件剪切到 server 目录中。

总之：

远程接口、实现 `Serializable` 接口用于客户服务器上传递的类在客户端和服务端上都有！  
其它的 `FileClient.class` 和 `FileServerImpl_stub.class` 都在客户端！  
实现远程接口的类和绑定此对象的类都在服务器端。

## 八、 运行程序

### 8.1 rmiregistry

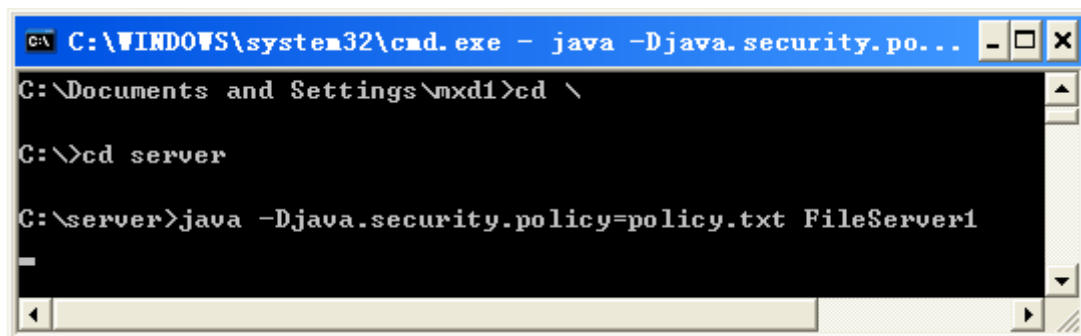
启动 RMI 的命名服务器

在你的服务器端启动命名服务器！这是 `java` 中提供的默认的用于 RMI 的命名服务器哦！不过此命名服务不同于 J2EE 的命名服务哦！它是一个采用 `plain text` 的命名服务器，即此命名服务器的名称是一般的采用文本的名称，不是 J2EE 平台的支持层次型的命名服务，同时此命名服务实现也是极其简单的（天下哪有免费的午餐哦），只有在其启动时，才可以注册对象，在其关闭后，这些对象丢失，就是保存在内存了，不提供持久保存！！在我们研究了 JNDI 命名服务和 JDBC 等数据库后我们可以自己建一个可以持久的命名服务器了！

记好命名服务器在服务器程序以及客户程序之前启动哦！为什么啊？没有电话本，你到哪里查电话啊，你往哪里写电话号码啊！命名服务器就是一个提供类似电话本一样的服务器软件！

### 8.2 java -Djava.security.policy=policy.txt FileServer

先运行服务器端程序！不过此时你得先进入你的 `server` 目录哦！

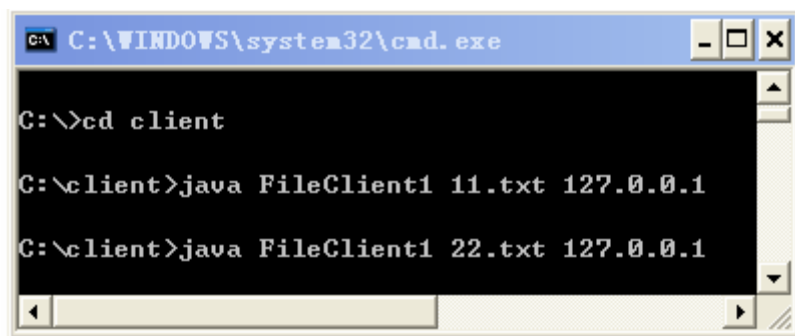


```
C:\WINDOWS\system32\cmd.exe - java -Djava.security.po...
C:\Documents and Settings\mxd1>cd \
C:\>cd server
C:\server>java -Djava.security.policy=policy.txt FileServer1
```

看到了吧！服务器程序 启动了，服务器程序没有返回哦！没有再次出现提示符哦！光标一直在闪动，表示服务器正常工作哦！

### 8.3 java FileClient1 11.txt 127.0.0.1

运行客户端程序！此时你得先进入你的 `client` 目录哦！



运行以上两个命令后，看看你的 client 目录是不是多了两个文件啊！和你服务器端 root 目录中的相应文件名的文件的内容一样吧！！^\_^ 成功了！恭喜你！恭喜你！

以上讲了一个稍微复杂点的，讲述了 RMI 程序的细节、讲述了源程序主要原因以及程序的发布、及其原因。RMI 架构中提供了两种服务器对象，以上这些两篇中的对象属于一类，这些对象在服务器启动后都启动且注册到命名服务器中。但是假如你的服务器端需要几千个对象协同工作，那是否需要命名服务器一启动，就要将这么多对象绑定到命名服务器呢！？RMI 提供了另外一种机制，在需要的时候启动对象，从而做到节省资源的目的！下篇我将给出一个具体的例子！请看下篇吧！

**更多精彩请关注：**

**<http://blog.163.com/miaoxiaodong78/>**

## 六步教你学会简单 RMI（下）

写作中.....

更多精彩请关注：

<http://blog.163.com/miaoxiaodong78/>