

# 基于 RMI 技术的远程词典应用

硕 3035 班

董璐

3113034016

2014 年 7 月 8 日 星期二

## 1. RMI 的工作原理

RMI系统结构，在客户端和服务端都有几层结构。方法调用从客户对象经占位程序（Stub）、远程引用层（Remote Reference Layer）和传输层（Transport Layer）向下，传递给主机，然后再次经传输层，向上穿过远程调用层和骨干网（Skeleton），到达服务器对象。占位程序扮演着远程服务器对象的代理的角色，使该对象可被客户激活。远程引用层处理语义、管理单一或多重对象的通信，决定调用是应发往一个服务器还是多个。传输层管理实际的连接，并且追踪可以接受方法调用的远程对象。服务器端的骨干网完成对服务器对象实际的方法调用，并获取返回值。返回值向下经远程引用层、服务器端的传输层传递回客户端，再向上经传输层和远程调用层返回。最后，占位程序获得返回值。

## 2. RMI 的具体实现和运行步骤

下面结合一个简单的实例来说明RMI的具体实现步骤。假设有一台应用服务器以RMI的方式向客户端提供英汉互译词典的服务。完成服务器端程序的编码和一个客户端应用，并分别部署到两台计算机上进行测试。具体实现过程如下：

### 2.1 定义 RMI 服务接口

任何使用RMI的系统都将使用服务接口。服务接口定义可以远程调用的对象方法，并指定其参数、返回值类型以及可能抛出的异常。与RMI服务一样，stub对象和skeleton对象将实现这个接口。所有RMI服务都扩展了java.rmi.Remote接口，该接口有助于标识可能被远程执行的方法。要为RMI系统定义新接口，必须声明扩展了Remote接口的新接口。只有在java.rmi.Remote接口(或它的子类)中定义的方法才可被远程执行，而对象的其他方法则对RMI客户隐藏。下面的程序定义了一个继承Remote的接口Translator。

```
/*
 * @file: Translator.java
 * @author: HeX
 * @email: hexiaofei@stu.xjtu.edu.cn
 * @date: July 5, 2012
 * @copyright (c) All rights reserved.
 */
import java.rmi.*;
// RMI本地接口必须从Remote接口派生
public interface Translator extends Remote {
```

```

        // 接口中的具体方法声明，注意必须声明抛出RemoteException
        String Translate(String str) throws RemoteException;
    }

```

## 2.2 实现 RMI 服务接口

定义了服务接口之后，下一步就是实现它。这个实现将给出每个方法的功能（也还可以定义其他方法）。然而，只有由RMI服务接口所定义的方法才能远程访问。下面程序实现了Translator。

```

/*
 * @file: TranslatorImpl.java
 * @author: HeX
 * @email: hexiaofei@stu.xjtu.edu.cn
 * @date: July 5, 2012
 * @copyright (c) All rights reserved.
 */
import java.rmi.*;
import javax.rmi.PortableRemoteObject;
import java.util.*;

public class TranslatorImpl extends PortableRemoteObject implements Translator
{
    /* 构造函数 */
    public TranslatorImpl() throws RemoteException {
        super();
    }

    /* 实现本地接口中声明的'Translate()'方法 */
    public String Translate(String str) throws RemoteException {
        Map map = new HashMap();
        map.put("a", "art. 一；任一；每一");
        map.put("b", "n. 乙等（成绩）；英语字母的第二个字母");
        map.put("c", "symb. 碳（carbon）\nn. 字母C");
        map.put("d", "n. 字母D");
        map.put("e", "n. 英语字母中的第五个字母");
        map.put("f", "n. 不及格；字母F\nabbr. 华氏温标（Fahrenheit）");
        map.put("g", "n. 英语字母的第七个字母");
        map.put("h", "n. 英语字母中的第八个字母，小写为h\nsymb. 氢（化学元素）");

        map.put("i", "pron. 我\nn. 碘元素；英语字母I");

        if (map.get(str) != null) {
            return map.get(str).toString();
        } else {

```

```

        return null;
    }
}

```

## 2.3 创建 stub 类和 skeleton 类

实现服务接口后，就可以用与JDK一起交付的rmic工具来创建这两个类文件。使用rmic工具程序可自动产生RmiTestSever-Stub.class和RmiTestSever-Skel.class 两个程序，分别担负Stub以及Skeleton的角色。由于RMI客户和RMI注册都将需要这两个类以及服务接口类，可将它们复制到本地文件系统，或通过动态类加载的Web服务器让它们在远程分布。

如果是JDK1.4以上的版本，则不会生成\_Skel文件，用\_Stub文件代之即可。

## 2.4 创建 RMI 服务器

RMI 服务器负责创建服务实现的实例，然后向远程方法调用注册处(rmiregistry)注册。下面程序段创建了TranslatorServer服务器：

```

/*
 * @file: TranslatorServer.java
 * @author: HeX
 * @email: hexiaofei@stu.xjtu.edu.cn
 * @date: July 5, 2012
 * @copyright (c) All rights reserved.
 */
import java.rmi.*;
public class TranslatorServer {
    public static void main(String[] args) {
        try {
            System.out.println("开始 RMI Server ...");
            /* 创建远程对象的实现实例 */
            TranslatorImpl tImpl = new TranslatorImpl();
            System.out.println("将实例注册到专有的URL ");
            Naming.rebind("TranslatorService", tImpl);

            System.out.println("等待RMI客户端调用...");
            System.out.println("");
        } catch (Exception e) {
            System.out.println("错误: " + e);
        }
    }
}

```

```
    }  
}
```

## 2.5 创建 RMI 客户端

编写RMI 客户端程序时，客户端只需取得指向远程接口的对象引用，而不需关心怎样发送和接收消息及服务位置。下面程序段创建了RmiTestClient 客户端：

```
/*  
 * @file: TranslatorClient.java  
 * @author: HeX  
 * @email: hexiaofei@stu.xjtu.edu.cn  
 * @date: July 5, 2012  
 * @copyright (c) All rights reserved.  
 */  
import java.rmi.*;  
import java.util.*;  
import java.io.*;  
public class TranslatorClient {  
    public static void main(String[] args) {  
        // 在服务器端设置安全机制  
        /*  
        if (System.getSecurityManager() == null) {  
            System.setSecurityManager(new RMISecurityManager());  
        }  
        */  
        /* 默认为本地主机和默认端口 */  
        String host = "localhost:1099";  
        /* 带输入参数时，将host设置为指定主机 */  
        if (args.length > 0)  
            host = args[0];  
        try {  
            /* 根据指定的URL定位远程实现对象 */  
            /* “t” 是一个标识符，我们将用它指向实现 “Translator” 接口的远程  
对象 */  
            Translator t = (Translator) Naming.lookup("rmi://" + host +  
"/TranslatorService");  
  
            System.out.println("我在客户端，开始调用RMI服务器端的'Translate'  
方法");  
  
            Scanner sc = new Scanner(System.in);  
            System.out.println("请输入单词：(输入q退出)");  
            String in = sc.next();
```

```

        while(in.equals("q") == false){
            System.out.println("释义：" + t.Translate(in));
            System.out.println("请输入单词：(输入q退出)");
            in = sc.next();
        }

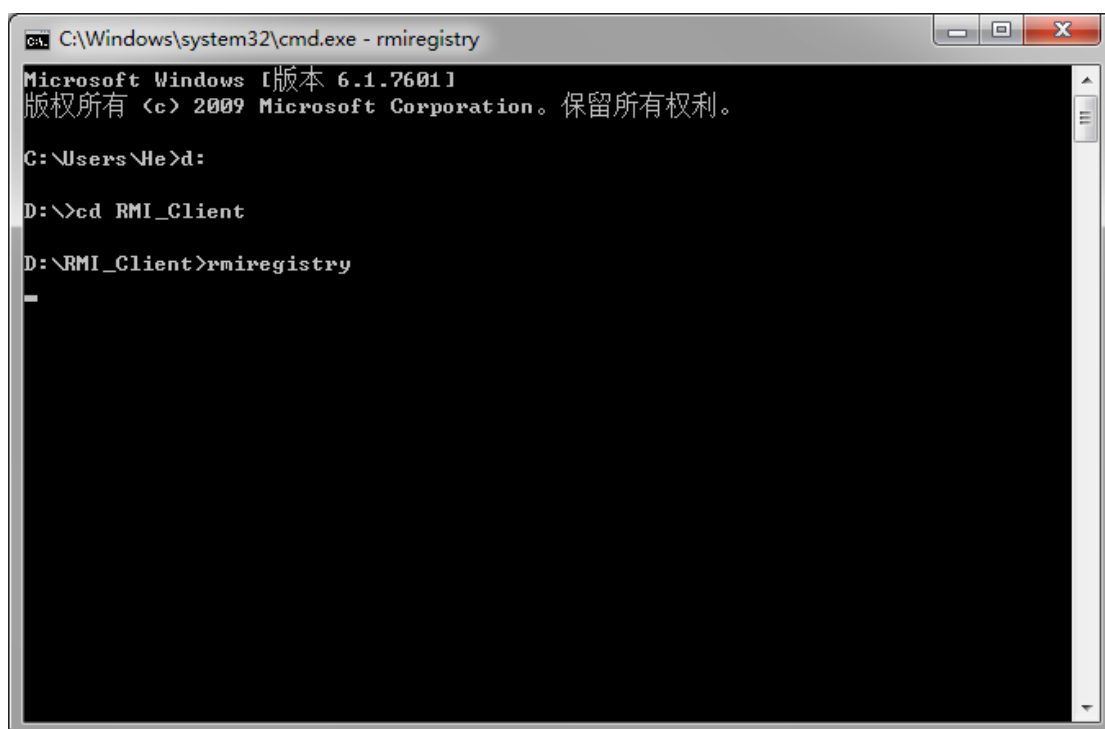
    } catch (Exception ex) {
        System.out.println("错误：" + ex);
    }
}
}
}

```

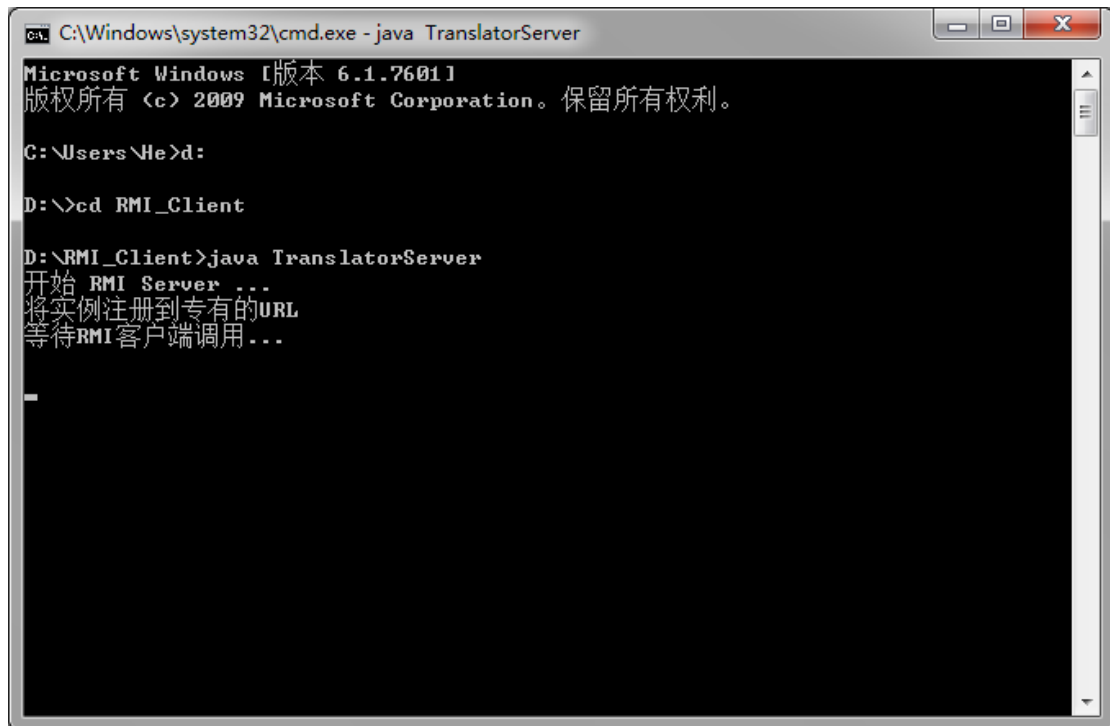
## 2.6 运行 RMI 系统

可以按下列步骤运行RMI系统：

- ①所有必需的文件复制到所有客户端和服务器的本地文件系统上的某个目录下。
- ②检查当前目录是否被包含在类路径中，或者类文件是否被存放在另一个目录下。
- ③将当前目录更改为存放类文件的目录，并运行rmiregistry程序。



- ④在独立的控制台窗口中，运行服务器。



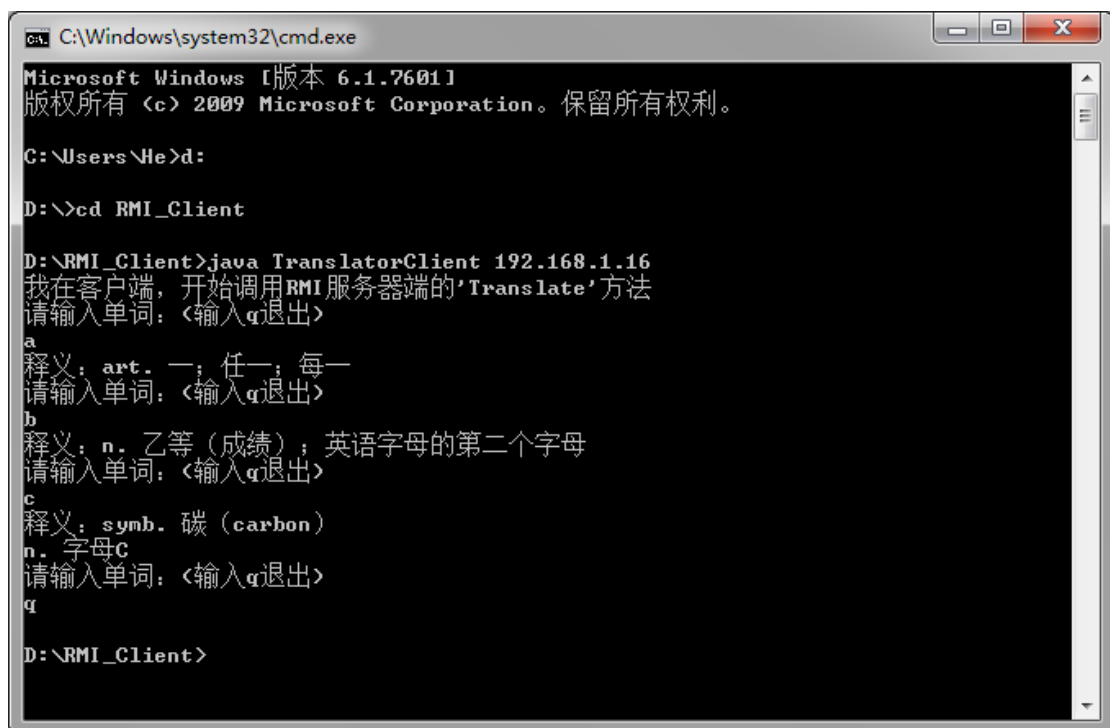
```
C:\Windows\system32\cmd.exe - java TranslatorServer
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\He>d:

D:\>cd RMI_Client

D:\RMI_Client>java TranslatorServer
开始 RMI Server ...
将实例注册到专有的URL
等待RMI客户端调用...
```

⑤在独立的控制台窗口中，而且最好是另一台机器上，运行客户端。



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\He>d:

D:\>cd RMI_Client

D:\RMI_Client>java TranslatorClient 192.168.1.16
我在客户端，开始调用RMI服务器端的'Translate'方法
请输入单词: <输入q退出>
a
释义: art. 一; 任一; 每一
请输入单词: <输入q退出>
b
释义: n. 乙等(成绩); 英语字母的第二个字母
请输入单词: <输入q退出>
c
释义: symb. 碳(carbon)
n. 字母c
请输入单词: <输入q退出>
q

D:\RMI_Client>
```

### 3. 总结

分布式对象技术主要是在分布式异构环境下建立应用系统框架和对象构件。在应用系统框架的支撑下，开发者可以将软件功能封装为更易管理和使用的对象，这些对象可以跨越不同的软、硬件平台进行互操作。

远程方法调用（RMI，Remote Method Invocation）是jdk1.1中引入的分布式对象软件包，它的出现大大简化了分布异构环境中Java应用之间的通信。