JAVA 应用在 Win32 系统中的引导程序设计

Biuld Launcher in Win32 System for Java Application

肖雁鹏　（中国电子科技集团第 34 研究所　桂林 541004）

摘要：本文提出一种在 Win32 系统中用 C + + 语言建立引导程序（Launcher），以运行 JAVA 程序的方法。并且通过一个简单但典型的实例来说明该方法，并给以了完整实现。

关键词：WinAPI　JAVA 应用　引导程序

1　介绍

Java 应用程序是由若干个 .class 文件组成的。运行时要到控制台下使用 java 命令或 javaw 命令来运行。其语法为：java [-options] class [args...] 。其中[-options]是可选的命令参数，class 指的是要启动的程序的主类名称，即含 main 函数的类，[args...]是 java 程序需要用到的参数。

通常的情况下，执行 Java 程序时，我们需要把所有用到的包的根目录指定给 CLASSPATH 环境变量或者 java 命令的 -cp 参数，必要时还需要用 -D 指定 java 命令的属性参数，当然有些时候还要加上传给 java 程序从运行参数。在 Windows 环境中，如果需要象别的 windows 程序一样，直接双击运行，可以有以下的方法：

（1）写 Windows 的批处理文件（.bat）。但运行批处理文件时会同时显示 Windows 的控制台窗口，破坏 Java 程序的运行时美观。

（2）生成一个详细的 Windows 快捷方式文件。然而这也限制了应用程序使用的灵活性。例如在运行使用了 RMI 的程序的服务端时，需要首先运行 RMI 注册表：rmiregistry.exe，这不可能通过一个快捷方式文件实现。

（3）制作一个可执行的 JAR 文件包来发布程序。然而实际上 windows 双击运行 jar 文件就是 windows 为扩展名为 jar 的文件，在注册表里登记了"…\bin\javaw.exe" -jar "%1" % * 的打开方法（这里的…表示 JRE 的安装路径）。但这也仅能适用于部分情况。例如在运行使用了 RMI 的程序的客户端时，通常情况下都需要使用特定的 Java 安全策略文件。即在 Java 命令中增加 -Djava.security.policy = my.policy 的系统属性参数。然而在打开方法中使用不了这些的参数。

因此在这里本文考虑在 Win32 系统中，用 WinAPI 设计一可执行程序，来引导 Java 程序。

2　建立示例 Java 程序

首先本文参照《计算机系统应用》2003 年第六期"Java 远程方法调用（RMI）技术及实现方法"一文（见参考文献[1]，下简称：远文），重写其远程调用的服务端，将程序封装在 JFrame 窗口内，以此建立我们要试验的 Java 应用程序。'远文'中 RmiTest 接口不变；改写其服务端程序如下：

服务端：RmiTestServer.java

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.lang.Math;
import javax.swing.*;
public class RmiTestServer extends UnicastRemoteObject
implements RmiTest
{
    public float getRmiTest(int r) throws RemoteException{
        float circlerate = (float)(2 * Math.PI * r);
        return circlerate;
    }
    public static void main(String args[]){
        JLabel JL = new JLabel("", SwingConstants.
```

```
CENTER);
        JFrame JF = new JFrame("RmiTest – Server");
        JF. setDefaultCloseOperation(JFrame. EXIT_ON_
CLOSE);
        JF. getContentPane(). add(JL);
        JF. setBounds(0,0,230,120);
        try { Naming. rebind ("//127. 0. 0. 1: 1099/
RmiTestObj",new RmiTestServer());
          JL. setText("RMI Connecting. .");
        }
        catch(Exception e){ JL. setText("RMI can't con-
nect. ." +e);}
        JF. setVisible(true);
    }
}
```

将以上程序编译后，用命令 rmic 生成 Stub 和 Skeleton 程序，共得四个 class 文件：RmiTest. class、RmiTestServer. class、RmiTestServer_Skel. class，RmiTestServer_Stub. class。

由此可见依次输入：

```
>rmiregistry 1099
>javaw RmiTestServer
```

就可以启动该 Rmi 服务端程序。运行正常时窗口显示"RMI Connecting. ."字样，表明程序正常启动，对象注册成功。

3　建立 Win32 程序

下面用 C + +语言设计一个程序，首先打开 VisualC + +6. 0 编译环境，使用项目向导新建一空的 Win32 Application 项目，在项目中添加一新 cpp 文件，包含主要代码的文件清单如下：

```
// server. cpp 从 Win32 系统中装载 Java 程序的引导
器
#include <windows. h>
#include <process. h>
#include <stdio. h>
#include <malloc. h>
// 定义要查找的 JRE 的键目录
#define JRE_KEY "Software\JavaSoft\Java Runtime Envi-
ronment"
static BOOL exist(char *filename)
{// 函数返回是否存在指定的文件}
static void AddJarsToClasspath(char * classpath,
const char * dir)
{//将 dir 指定的目录中的所有 Jar 文件(. jar)加到
Classpath 字符串中，以';'隔开*/}
static BOOL GetPublicJREHome(char *buf, int bufsize)
{// 在注册表里查找 JRE 的安装位置}

int APIENTRY WinMain(HINSTANCE hInstance, HIN-
STANCE hPrevInstance, LPSTR lpCmdLine, int nCmd-
Show)
{
    char command[MAX_PATH + 1]; //带绝对路径
的 javaw 命令文件
    char **progargv, **argseq; //javaw 命令的
参数
    char property[10 * MAX_PATH + 1 + 2]; //ja-
vaw 命令的系统属性值选项
    char cp[4];
    char classpath[10 * MAX_PATH + 1]; //指定用
户类的类路径
    char mainclass[MAX_PATH + 1]; //要启动的
main 类文件
    char rmireg[MAX_PATH + 1]; //带绝对路径的
rmiregistry 命令文件
    char **rmiargv; //rmiregistry. exe 的参数
    rmiargv = (char **) calloc(2, sizeof(char
*));
    *rmiargv = "1099"; //如果不指定端口，填入空
格即可
    command[0] = rmireg[0] = 0; //初始化为
NULL
    if (GetPublicJREHome(command, sizeof(com-
        mand))) {
        sprintf(rmireg, command);
        strcat(command, "\bin\javaw. exe");
        strcat(rmireg, "\bin\rmiregistry. exe");
```

```
if ( ! ( exist( command ) && exist( rmireg ) ) ) { return 1; } // 找不到 Java 目录退出
spawnv( _P_DETACH, rmireg, rmiargv ); //在一个新的进程中执行 rmireg 指定的命令
sprintf( property, " -D" );
/* 根据需要,在这里加入系统属性值,如 sprintf( property, " java. security. policy = my. policy" ); */
sprintf( cp, " -cp" );
sprintf( classpath, "." );
AddJarsToClasspath( classpath, "." ); // 在当前目录中查找所有 jar 文件
/* 根据需要,在这里可以加入更多语句以搜索更多路径中的 Jar 文件 */
sprintf( mainclass, " RmiTestServer" );
// 下面组装 javaw 命令的参数表
progargv = argseq = ( char * * ) calloc( 5, sizeof( char * ) );
* argseq + + = property;
* argseq + + = cp;
* argseq + + = classpath;
* argseq + + = mainclass;
return ( execv ( command, progargv ) ); //执行 command 指定的命令,并带有参数。
}
```

4 程序分析

因为事先并不知道 Java 运行时环境(JRE)的确切安装目录,所以首先必须获得 Windows 系统中 JRE 的安装路径,这可以通过访问 Windows 注册表读得。如以上程序中 GetPublicJREHome 函数所示,该函数通过 WinAPI 的注册表函数,返回包含 JRE 的绝对路径的字符串。

程序在 WinMain 函数中调用了 AddJarsToClasspath 函数,将指定的目录中的所有 Jar 文件加到 Classpath 中,此函数是为了在需要时配合 java 命令的 -cp 选项参数。需要指定的目录可以在程序设计时自由添加,其中使用. 和. . 分别对应着当前目录和父目录。典型地可以把全部 Jar 文件集中放在某一目录下,然后在该引导程序中将目录指定给 AddJarsToClasspath 函数。

程序中 property 字符串,就是 Java 命令的系统属性值,它由' -D'选项指定。这里应该注意' -D'前应有一个空格符,而它与后面的系统属性值之间不能有空格。

在 WinMain 函数中要做的主要任务就是组装要发出的 javaw 命令和 rmiregistry 命令以及它们的参数。如源程序所示。可以调用 API 函数 Spawnv 和 execv 函数发出执行命令。Spawnv 函数功能是:创建并执行新的进程。execv 函数功能是:加载并执行新的进程。

Spawnv 和 execv 函数的原型分别为:

```
int spawnv( int mode, const char * cmdname, const char * const * argv );
int execv( const char * cmdname, const char * const * argv );
```

其中 cmdname 为带路径的程序名,argv 为参数数组的指针。而 Spawnv 函数的 mode 参数是被调用进程的执行模式的常量,其不同值的含义参见微软的 MSDN。值得注意的是,由于 rmiregistry 命令并不立即返回,而是做为服务程序留在后台,因此这里对应地必须使用 Spawnv 函数,以在一个新的进程中运行程序,所以其 mode 参数设置为'_P_DETACH',即继续执行原进程,新进程置于后台运行,不接受控制台和键盘响应。

分析程序可得,以上程序向 Windows 操作系统提交的命令如下(设 JRE 的绝对路径为 C:\Java\jre\)

```
>C:\Java\jre\bin\rmiregistry.exe 1099
>C:\Java\jre\bin\javaw.exe -D -cp . RmiTestServer
```

当双击其生成的可执行文件时,由该文件完成预期的任务,启动了 Java 程序。根据实际应用中命令的不同,我们可以很容易的修改源程序中 WinMain 函数,以满足需要。

5 结论

编译该项目,得到的 release 程序 server.exe 大小仅 36K,在拥有四个 class 文件的目录中,双击 server.exe 文件就可以直接启动 RMI 注册表和该 RmiTestServer 程序。其运行结果如图 1 所示,该程序注册的远程对象同样可以由'远文'中的客户端成功调用。根据

(下转第 86 页)

（上接第 83 页）

以上方法还可以改写客户端，并制作客户端的引导程序。另外还可以添加代码，使用 WinApi 为该 Windows 程序加入个性化的图标。

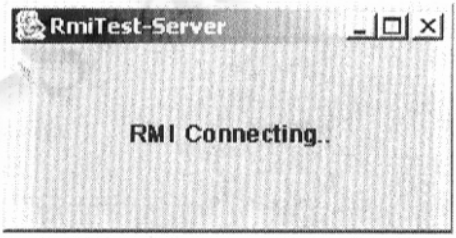

图 1

综上所述，依照类似的结构，完全可以为任何 Java 应用程序构造合适的引导程序。并且越是庞大、复杂的 Java 应用系统，越是能彰显引导程序的优势。

参考文献

1 陈桦、李鹏，Java 远程方法调用（RMI）技术及实现方法，计算机系统应用，2003.06。

2 MSDN：http://msdn.microsoft.com/

3 （美）Qusay H. Mahmoud. JAVA 分布式程序设计〔M〕国防工业出版社，2002。