

Java 网络浏览器组件的设计与实现

陈奕敏^{1,2}, 张继超², 袁 奕²

(1. 清华大学 软件学院, 北京 100084; 2. Sun 中国工程研究院, 北京 100084)

(chenyimin03@mails.tsinghua.edu.cn)

摘 要:Java 标准组件 JEditorPane 支持 HTML3.2 标准,但不支持被广泛采用的 HTML4.0 标准,无法表达网页中嵌入的多媒体信息。为此提出了一个基于本地浏览器的 Java AWT 浏览器组件设计方案,采用 Socket 在本地浏览器和 Java 虚拟机之间进行通信,利用 Java 的 AWT 本地接口将浏览器组件嵌入 Java AWT/Swing 应用程序,实现了在 Java 应用程序中提供网络浏览器的功能。

关键词:Java; JNI; 浏览器

中图分类号: TP311.52 **文献标识码:** A

Design and implementation of Java Web browser component

CHEN Yi-min^{1,2}, ZHANG Ji-chao², YUAN Yi²

(1. School of Software, Tsinghua University, Beijing 100084, China;

2. Sun China Engineering and Research Institute, Beijing 100084, China)

Abstract: JEditorPane is a standard Java component, and it provides HTML 3.2 support but does not support HTML 4.0 which is widely used now. It can not render the multimedia contained in HTML page. A design of Java Web browser component was presented based on a native browser. The component made use of native browser through a native process which invoked the real browser and communicated with Java process via socket channel. By using Java AWT native interface, native browser was embedded into Java AWT/Swing application. This component provides the ability of rendering HTML page in Java application.

Key words: Java; JNI; browser

0 引言

随着 Java 在客户端程序的广泛应用,越来越多的程序员希望 JDK 能提供一个网络浏览器组件来显示 HTML 格式的网页信息,例如用于浏览应用系统的帮助信息。由 W3C 组织制定的 HTML 格式的网页可以表达丰富的信息类型,包括文本、图像、音频、视频等信息。HTML 格式已经成为一种最流行的信息存储方式之一。浏览网页信息的组件被称为网络浏览器,它是一个十分复杂的软件系统。在 Java 程序中显示网页有两种主要的解决方案,第一种方案是完全由 Java 解析 HTML 页面并进行显示,另一个方案就是利用其他独立的浏览器进行解析和显示,并将其嵌入 Java 程序中。Swing 组件 JEditorPane 属于第一种方案,它是 JDK 提供的一个标准组件,可以用于显示 HTML 格式的文件。其优点是它是一个纯 Java 的标准类,使用这个组件无需附加任何第三方的库,然而该组件的功能非常有限,它只支持 HTML3.2 标准,并且不支持其他的扩展功能,无法表达网页中嵌入的音频、视频以及 Flash 动画等信息,也不支持广泛使用的 CSS 样式等。JExplorer 组件(要了解 JExplorer 的详细信息可以访问: <http://www.jniwrapper.com/jexplorer.jsp>)属于第二种方案,它是一个 AWT 组件可以用于标准的 AWT/Swing 程序,但它嵌入的是 IE,也就是它只支持 Windows 平台,并且它的源代码是不公开的。利用现有的、成熟的浏览器来实现一个功能完备的浏览器组件是个不错的选择。

由于平台无关是 Java 必须保持的特性,所以最好能利用一个跨平台的浏览器来实现这个 Java 浏览器组件。Mozilla 基金会的 Mozilla 浏览器(要了解 Mozilla 基金会及 Mozilla 浏览器,请访问: <http://www.mozilla.org>)在多种操作系统和硬件平台上都可以使用,是理想的选择。然而,很多个人电脑上并没有安装 Mozilla 浏览器,所以我们必须实现在 Java 中嵌入 IE 的功能。本文描述的浏览器组件已经成功地实现了将 IE 和 Mozilla 嵌入 Java 的 Swing/AWT 程序中。

1 框架设计

1.1 整体结构设计

设计目标是实现一个称为 WebBrowser 的 Java 类,该类可以用于显示 HTML 页面,并且该类可以被加入到任何 Java 的其他容器类(java.awt.Container 类或其子类)中,从而实现在 Java 的 AWT/Swing 图形界面中嵌入一个成熟的网络浏览器的目的。

由于我们需要调用本地浏览器(IE 或 Mozilla)来实现 WebBrowser,所以代码必须是本地代码而不是 Java 代码,于是有两个选择:1)通过 JNI 调用本地代码实现对本地浏览器的调用;2)通过一个独立的本地进程来实现。如果采用第一种方法,也就是在 Java 进程中执行对浏览器的调用,那么当浏览器浏览某个含有 <applet> 标签的网页时,当前正在运行的虚拟机将会被直接用于运行 Java Applet,而当前虚拟机事实上正在运行一个 Java 的应用程序。这是因为,当浏览器浏览

某个含有 <applet> 标签的网页时,浏览器的 Java Plug-in 将会判断当前进程是否运行在一个 Java 虚拟机中,如果不是,它将启动一个新的 Java 虚拟机用于运行 <applet> 标签所指定的 Java Applet;如果 Java Plug-in 判断当前进程已经是运行在一个 Java 虚拟机中,则直接用当前的虚拟机来运行该 Java Applet。所以,必须在一个独立的本地进程中调用本地浏览器,并通过 socket 和 Java 虚拟机进行通信。本地进程和 Java 进程分别扮演服务器和客户端的角色。

采用在独立的进程来实现对本地浏览器的调用还有其他方面的好处:

1)使系统更稳定,当本地代码发生错误时,只会终止本地进程,而不会导致 Java 虚拟机地崩溃;

2)不用处理和 Java 虚拟机之间的关系,使代码结构更清晰,增强代码的可读性,并极大地简化了调试工作。

程序的主要流程是:Java 程序首先获得本地一个可用的网络端口,然后启动一个用于实现对本地浏览器进行调用的本地进程,并在启动的命令行参数中将前面获得的端口号传递给本地进程。本地进程将监听在命令行参数中指定的端口。Java 程序等待本地进程建立对约定端口的监听之后,通过 Socket 向该端口发送请求(包括建立新浏览器窗口、浏览特定网址、浏览前一页、浏览后一页、停止等),并且把某个特定区域的界面绘制工作移交给本地进程执行。而本地进程接到 Java 端的请求之后将执行相应的动作,并绘制该屏幕区域,同时将执行结果通过 Socket 返回给 Java 端。Java 进程接到本地进程发来的结果后,执行必要的处理动作。Java 进程和本地进程之间的协作关系如图 1 所示。

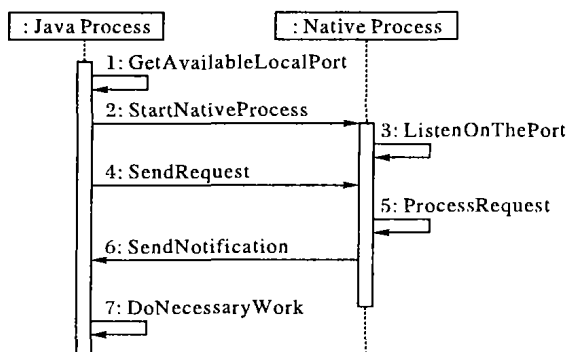


图1 Java进程和本地进程之间协作的顺序

图1中步骤1、2、3对于整个应用程序都只执行一次。步骤4、5对应于每次从Java进程向本地进程发送消息的处理过程,而步骤6、7则对应于从本地进程向Java进程发送消息的处理过程。其中4和6并不是一一对应的,本地进程处理Java进程发送的请求之后可能向Java进程返回一条或多条响应信息。

1.2 通信协议设计

对于采用Socket通信的双方,必须制定一套通信的协议。这里采用简单的字符串形式发送消息,消息格式为:

<Instance ID>, <Event Type>, [Event Data]

其中<Instance ID>是对应的WebBrowser对象的实例编号(一个应用程序中可能有多个WebBrowser类的实例);<Event Type>是在程序中定义的事件编号,用于说明事件类型;[Event Data]是可选部分,用于提供附加的事件数据(例如浏览某个网页时提供网址,或者改变窗口大小时提供新窗口边界数据等)。

1.3 屏幕绘制原理与设计

不论是在微软的Windows窗口系统中,还是UNIX/Linux

的X Window窗口系统中,每一个窗口都有一个对应的窗口句柄。利用该句柄就可以获得句柄所代表的窗口的一些属性,并可以对窗口进行一些操作。Java程序的图形界面是建立在这些窗口系统之上的,所以Java程序中的每个窗口也都有对应的窗口句柄。为了实现将本地浏览器嵌入到Java图形界面中,本地进程需要获得对应的Java组件的窗口句柄,然后在该句柄所代表的窗口区域内进行HTML页面(或者其他内容)的绘制,从而实现“嵌入”的目的。

每个AWT组件都对应一个本地窗口(Native Window),而Swing组件则只有几个顶级的窗口组件(JFrame, JWindow, JDialog, JApplet)具有对应的本地窗口。一旦获得该本地窗口对应的窗口句柄,就可以对该AWT/Swing组件所占据的窗口区域进行一些操作,例如屏幕绘制等。由于必须将浏览器嵌入到Java应用程序的图形界面中,所以不能使用顶级窗口组件来代表浏览器,而且该组件必须有对应的本地窗口,所以必须采用AWT组件来实现WebBrowser类。在实现中采用从java.awt.Canvas类扩展实现WebBrowser类,符合一般开发AWT组件的习惯。

从Canvas类扩展的WebBrowser类的实例对象一旦被加入到图形界面中,将获得一个对应的窗口句柄,利用JNI可以获得该句柄,从而可以将屏幕的绘制工作交给独立的本地浏览器进程来完成。WebBrowser类的实例从Java端向本地进程端发送的第一个请求消息总是请求本地进程建立一个新浏览器窗口,在请求消息的[Event Data]部分将包含该WebBrowser实例对应的窗口句柄。本地进程获得该窗口句柄后,可以通过窗口系统提供的API来操作该窗口,包括绘制窗口所占据的屏幕区域、设定窗口属性等。WebBrowser类必须将所有屏幕的绘制工作移交给本地进程。

1.4 消息处理的设计

Java进程在启动本地进程的命令行参数中指定一个端口号,在本地进程中新建一个线程用于监听命令行参数指定的端口。本地进程收到任何Java进程发送过来的请求消息,首先按照约定的消息格式解析消息内容,然后根据消息中的实例编号和事件类型,调用适当的本地浏览器的API,从而使浏览器窗口响应Java进程的请求。另一方面,WebBrowser类提供一系列的方法,用于控制浏览器的行为,例如:setURL(java.net.URL url),back(),forward(),stop()等方法。在这些方法的实现中,根据约定的消息格式和对应的事件类型编号,构造适当的消息(字符串),然后简单地将这些消息写到一个“发送缓冲区”。在Java进程中将启动一个新线程用于监听约定的端口以及“发送缓冲区”,如果发送缓冲区中有未处理的消息,则将其通过Socket发送给本地进程,由本地进程处理这些WebBrowser的请求消息;如果监听的端口中有新消息到达,则解析该消息,然后调用WebBrowser类相应的处理方法,由WebBrowser类响应本地进程发送的消息。从而实现了Java进程和本地进程之间互相响应对方的消息(事件),使得被嵌入的本地浏览器看起来像是Java程序的一部分。

2 实现中的关键点

2.1 获得WebBrowser对象的窗口句柄

首先在WebBrowser类中声明一个本地方法:

```
public native int nativeGetWindow(String javaHome);
```

其中参数javaHome是由java.lang.System.getProperty(“java.home”)返回的String类对象,用于在本地进程中装载

<JAVA_HOME>/bin/jawt.dll 动态库。该动态库提供了获得 AWT 组件对应的窗口句柄的方法。

然后用 C++ 实现这个本地方法,在 Windows 系统中的代码如下(在 UNIX/Linux 系统中的代码与之类似):

```
JNIEXPORT jint JNICALL
Java_org_jdesktop_jdic_browser_WebBrowser_nativeGetWindow
(JNIEnv * env, jobject canvas, jstring javaHome)
{ typedef jboolean (JNICALL *PJAWT_GETAWT)
  (JNIEnv *, JAWT *);
  HMODULE _hAWT;           // JAWT module handle
  JAWT awt;
  JAWT_DrawingSurface * ds;
  JAWT_DrawingSurfaceInfo * dsi;
  JAWT_Win32DrawingSurfaceInfo * dsi_win;
  jboolean result;
  jint lock;
  HWND hWnd = 0;
  // Load jawt.dll from <java.home> \bin
  char dllLocation[ MAX_PATH ] = {0};
  const char * javaHomeStr = env->
    GetStringUTFChars( javaHome, NULL );
  sprintf( dllLocation, "%s%s", javaHomeStr,
    "\\bin\\jawt.dll" );
  env->ReleaseStringUTFChars( javaHome, javaHomeStr );
  _hAWT = LoadLibrary( (LPCTSTR) dllLocation );
  if ( !_hAWT )
  { return -1; }
  else
  { PJAWT_GETAWT JAWT_GetAWT = ( PJAWT_GETAWT )
    GetProcAddress( _hAWT, "_JAWT_GetAWT@8" );
    if ( JAWT_GetAWT )
    { awt.version = JAWT_VERSION_1_3;
      result = JAWT_GetAWT( env, &awt );
      if ( result != JNI_FALSE )
      { ds = awt.GetDrawingSurface( env, canvas );
        if ( ds != NULL )
        { lock = ds->Lock( ds );
          if ( (lock & JAWT_LOCK_ERROR) == 0 )
          { dsi = ds->GetDrawingSurfaceInfo( ds );
            dsi_win =
              ( JAWT_Win32DrawingSurfaceInfo * )
              dsi->platformInfo;
            hWnd = dsi_win->hWnd;
            ds->FreeDrawingSurfaceInfo( dsi );
            ds->Unlock( ds );
          }
        }
        awt.FreeDrawingSurface( ds );
      }
    }
  }
  return (jint) hWnd;
}
```

通过以上方法,在 Java 程序中就可以获得 WebBrowser 实例的窗口句柄。将该句柄发送给本地进程,本地进程将创建一个以该句柄所代表的窗口为父窗口的新浏览器窗口,它将占据 WebBrowser 实例所占有的全部屏幕区域,并且将父窗口接收到的所有消息都转发给新窗口处理,新窗口控制屏幕的绘制工作。Java 进程向本地进程发送第一条创建浏览器窗口请求的时间点必须选在 WebBrowser 实例被添加到界面上以后(也就是在其对应的本地窗口资源被虚拟机创建以后,更

具体地说是在 addNotify() 方法被调用以后),在实现中选择在 paint(Graphics g) 方法第一次被调用的时候向本地进程发送创建窗口的请求,因为虚拟机在调用 paint 方法之前,必然已经创建了所需要的本地窗口资源。

2.2 将 WebBrowser 类的屏幕绘制工作移交给本地进程

在 WebBrowser 类中必须覆盖父类 java.awt.Canvas 的 paint(java.awt.Graphics g) 方法。在该方法的实现中,WebBrowser 将判断当前调用是否是第一次调用该方法,如果是,则向本地进程发送创建窗口的请求(如上小节所述);如果不是第一次调用该方法,则直接返回而不做任何工作。同时还必须覆盖父类的 setBounds(int x, int y, int w, int h) 方法,在该方法的实现中向本地进程发送一条消息,指明 <Event Type> 为 EVENT_SET_BOUNDS(在程序中定义),并且 [Event Data] 为新边界的 x, y, w, h 值,从而请求本地进程完成窗口定位。这样,WebBrowser 类就将屏幕的绘制工作完全移交给本地进程,从而实现了在 AWT/Swing 界面中嵌入一个本地浏览器窗口的目的。

2.3 处理 WebBrowser 类和其他容器类之间的关系

在 Java 中,当某个组件被添加到一个容器类(java.awt.Container 类及其子类),则这个组件的 addNotify() 方法被调用,该方法用于创建组件所对应的本地屏幕资源,例如本地窗口等。当某个组件从一个容器类中被删除,则这个组件的 removeNotify() 方法被调用,该方法用于释放组件所占有的资源。当 removeNotify() 方法被调用后,该组件将不能被显示,如果需要重新将该组件添加到某个容器中,则容器将会自动调用该组件的 addNotify() 方法以重新创建对应的本地屏幕资源。

然而,这个机制对于 WebBrowser 并不适用,因为 WebBrowser 的部分本地屏幕资源是由本地进程创建的,如果调用了 removeNotify(),则这部分资源也被释放,当重新将 WebBrowser 类的实例添加到其他容器时,必须向本地进程重新发送创建浏览器窗口请求,以创建浏览器窗口屏幕资源,而且需要重新定位新创建的浏览器到 removeNotify() 方法调用之前所浏览的网页地址,这样也就需要重新从网络上下载该网页内容。显然,根据这个方案实现起来的代码将会复杂且低效,我们需要设计一个替代方案。

一个可行的方案就是,WebBrowser 覆盖父类的 addNotify() 和 removeNotify() 方法。在 removeNotify() 方法的实现中只是简单的调用 setVisible(false) 使该组件隐藏,而不是释放资源。而在 addNotify() 方法的实现中可以先调用父类的 addNotify(),然后调用 setVisible(true) 方法使得 WebBrowser 被显示。由于组件对应的本地屏幕资源并没有被释放,所以 addNotify() 并不重新创建本地屏幕资源。这样,WebBrowser 类还必须提供一个 dispose() 方法,用于真正的释放资源。该方法的实现中创建一个新线程,在新线程中向本地进程发送一条消息,请求本地进程释放组件对应的由本地进程创建的资源,然后挂起新线程。本地进程释放资源后向 Java 进程发送应答,Java 进程接到应答后将唤醒被挂起的新线程。然后新线程调用组件的 super.removeNotify() 方法释放组件由 Java 虚拟机创建的本地资源。dispose() 方法用于在必要时释放 WebBrowser 实例对应的资源。这样,对于 WebBrowser 类的实例,当其不会再被使用时,可以通过调用 dispose() 方法来释放资源。

(下转第 1902 页)

3 用智能卡协调管理器进行应用开发

本节给出使用智能卡协调管理器接口函数进行应用开发的基本流程,如图3。

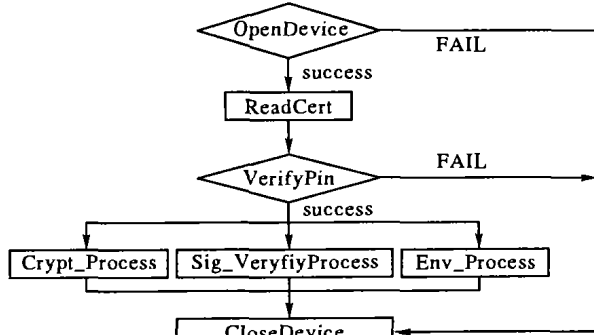


图3 应用开发的基本流程

主要实现代码如下:

```

include "predefined.h" //预定义的头文件
deviceInfo = "type = MemoryIC; device = ; deviceindex = 1; baudrate =
9600; mediatype = ";
//打开设备
ret = CTCA_OpenDevice( deviceInfo, ctx);
if (ret == SUCCESS)
{ CTCA_VerifyPin( ctx, adminPin); //验证使用者的密码
...
//使用加密证书进行加密操作
ret = CTCA_PubKeyProc( ctx, TYPE_ENCRYPT, cryptCert,
inData, inDataLen, outData, outDataLen);
if (ret == SUCCESS)
{ ret = CTCA_SignData( ctx, ALG_MD5_WITH_RSA, inData,
inDataLen, signedData);

```

```

}
}
ret = CTCA_CloseDevice( ctx); //关闭设备
...
free(); //释放所有分配的空间

```

4 结语

通过对中国电信加解密标准、数字证书存储介质规范和 PKCS11 的研究,结合中国电信 CTCA 具体应用需求,提出了一种新的基于中国电信 CTCA 的智能卡协调管理器模型,并给出了相应的软件实现模块。该模型对基于 CTCA 的智能卡在安全应用领域有很强的扩展性和较大的灵活性。此系统已经在中国电信安全公务平台(www.chntel.com)中成功使用,在使用过程中,为用户、智能卡厂商、应用开发商提供了一个良好的开发和使用平台,已经取得了巨大的经济效益。同时,基于 CTCA 的智能卡协调管理器在应用中也暴露出不少问题,如:对多个应用同时访问一个智能卡的支持不稳定;对热插拔智能卡操作,错误处理能力不完善等,还需要进一步的研究和探讨。

参考文献:

- [1] 中国电信. 中国电信数字证书存储介质规范[Z]. 2003.
- [2] 中国信息安全产品测评认证中心. 信息安全理论与技术[M]. 北京:人民邮电出版社, 2003.
- [3] 王晓京, 陈光喜. 程序语句生成子与程序设计自动化[J]. 计算机应用, 1998, 18(11).
- [4] NASH A. PKI: implementing and managing E-security[M]. New York: Osborne/McGraw-Hill, 2001.
- [5] HOUSLEY R, POLK T. Planning for PKI: best practices guide for deploying public key infrastructure[M]. New York: Wiley, 2001.
- [6] 周学广, 刘艺. 信息安全学[M]. 北京:机械工业出版社, 2003.

(上接第 1898 页)

3 结语

本文详细描述了在 Java 程序中集成本地浏览器的设计与实现的过程。该组件在 UNIX/Linux 平台支持对 Mozilla 浏览器的嵌入,在 Windows 平台支持对 IE 和 Mozilla 浏览器的嵌入,实现了跨平台的目标。采用 WebBrowser 类,在 Java 程序中可以很容易地加入浏览器的功能,并保持程序的跨平台特性。并且所实现的浏览器具有强大的功能,因为该浏览器实际上是本地默认浏览器(IE 或者 Mozilla),其对 HTML 的支持已经相当完善。WebBrowser 组件的演示结果如图 2 所示,可以利用 Java Web Start 通过网络点击 <http://javadesktop.org/jdic/demo/Browser/browser.jnlp> 直接运行该演示程序。图 3 为采用 javax.swing.JEditorPane 组件显示同一个网页的结果,可以看出其对 HTML 的支持不完善。

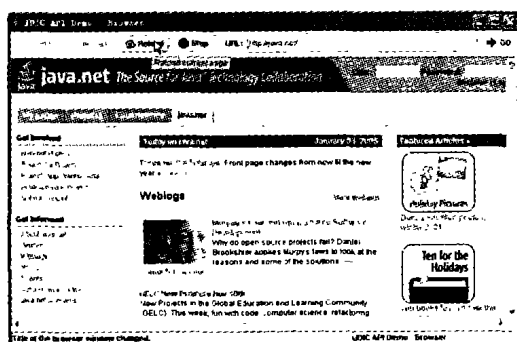


图2 WebBrowser 演示结果

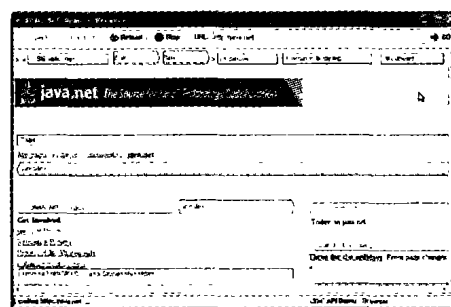


图3 采用 JEditorPane 显示的网页

WebBrowser 浏览器组件是 JDIC (JDesktop Integration Components) 项目的一个组成部分。JDIC 项目是一个开放源代码项目,整个项目的所有源代码均可以通过 <https://jdic.dev.java.net> 免费下载。Sun 公司计划在将来的 JDK 中集成 JDIC 的 WebBrowser 组件。该浏览器组件目前还不够成熟,例如还不支持 W3C 的 DOM 规范,这正是 JDIC 项目将要实现的工作。

参考文献:

- [1] SHENG LIANG. The Java Native Interface[M]. USA: Addison-Wesley, 1999.
- [2] The AWT Native Interface[EB/OL]. http://java.sun.com/j2se/1.5.0/docs/guide/awt/1.3/AWT_Native_Interface.html, 1999.
- [3] Java Native Interface Specification[S/OL]. <http://java.sun.com/j2se/1.5.0/docs/guide/jni/spec/jniTOC.html>, 2003.
- [4] GEARY Graphic Java 2: Mastering the JFC[M]. California, USA: Prentice Hall, 1999.
- [5] 邱仲潘. Java 网络编程指南[M]. 北京:电子工业出版社, 2002.