**通过 Java 调用 OCX 控件有几种方法， JNI 、 JACOB 、 Jawin 等**

1.JNI

最直接的方式，也是最麻烦的方式，需要自己完成所有的工作，不推荐。

2.Jawin

尝试了一下，效果不错，但相对来说，其编程风格更贴近 Windows ，离 Java 有点远

3.Jacob

使用 Jacob 非常方便， Java 编程风格，需要了解的知识比较少。

下载地址 http://sourceforge.net/projects/jacob-project/

# Jacob 的使用方法

## 1. 初始化

ComThread.InitMTA(true);

ActiveXComponent com = new ActiveXComponent(" 组件的 ProgID") ;

Dispatch disp = com.getObject();

## 2. 调用控件里面的方面

### 2.1 调用无参的方法，并返回一个 short 值

Dispatch.call(disp, "Init").getShort();

### 2.2 调用有一个参数的方法，并返回一个 boolean 值

Dispatch.call(disp,"Method",new Variant(args)).getBoolean();

调用多个参数依次类推，注意在传递参数前，将 Java 中的参数转换成 Variant 。

问题解决

### 2.3在使用 Jacob 调用 OCX 控件时，总是出一个异常

Exception in thread "main" com.jacob.com.ComFailException: A COM exception has been encountered:

At Invoke of: Init

Description: 灾难性故障

通过 Jawin 调用，会出现 8000FFFF 错误。

这个错误是由 ActiveX 结构设计造成的。

在 Ole4.0 版本之前，外部程序是可以直接调用 OCX 中方法的。 Ole4.0 之后，每次调用控件中的方法，系统会自动检查是否允许调用，即运行 COleControl.IsInvokeAllowed (DISPID)

该方法检查控件是否正确的初始化或者是否通过持久存储接口正确加载，如果两个条件有一个满足，即返回 TRUE ，否则返回 FALSE 。

当控件在 MFC 中使用时，很多细节，如初始化，都被过滤了，这样，大多数用户都不会遇到这个问题。但是，当我们从 C 、 C++ 的 dll 中调用控件时，不满足上述条件，该方法返回 FALSE ，这时候再调用任何控件方法，都会出现上述异常。

### 2.4解决方法：

很简单，在 OCX 控件中，重写 COleControl.IsInvokeAllowed (DISPID) 方法，即在控件的 CMyNameCtrl 中增加一个方法

BOOL CMyNameCtrl::IsInvokeAllowed (DISPID)

{

return TRUE; }

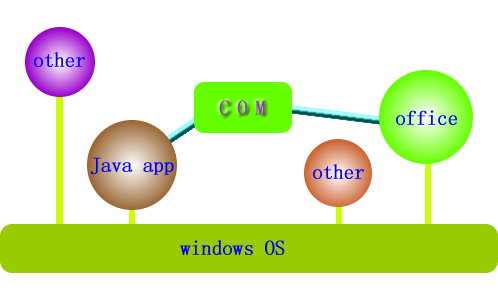
## 3.jacob介绍

### 3.1com的解释

首先，大家先要了解一下 jacob ，官方的解释是 Java COM Bridge ，即 java 和com 组件间的桥梁（进一步了解 com/dcom: http://docs.huihoo.com/com/ ）com 一般表现为 dll 或 exe 等二进制文件，像我们呆会会用到的 jacob.dll 文件。这里说说为什么我们用 java 去操纵 office( 如： word) 要使用 com ，而不直接使用 java 去做？首先，我们清楚 office 是建立在 windows 平台之上的，本身是一个软件，除了他自己提供的宏似乎没有什么能对他进行直接的操作；在 windows 平台上为了解决像这样的不同应用软件，通信缺乏通用 api 问题，推出了 com 的解决方案；我们使用 dll 中的一组或多组相关的函数存取组件数据，总的合称为接口具体到每个细节的实现称为方法；如果我们要调用接口里的方法，唯一的途径就是调用指向接口的指针；

所以总的来说使用就是 dll 完成 api 的转换；如果你听不懂，则称为废（不懂没关系，会用就好，用久自然懂了）； 好了 com 讲完，我们开始我们的主要内容吧！

### 3.2 jacob\_1.9.zip

里面的 jacob.jar 是我们要用的包  
            jacob.dll 就是我前面说的 com 组件  
把包里的 jacob.dll 放到 c:/windows/system32 下  
讲解麻烦，画个图大家看 好  
  
  
  
值得注意的是，不同的版本的系统使用不同的 dll 文件  
所以如果你编译成功，但运行失败一般是 dll 文件问题  
遇到这种情况，可以到  
<http://downloads.sourceforge.net/jacob-project/jacob_1.9.zip?modtime=1109437002&big_mirror=0>

下载其他的版本的 dll 文件。  
先给大家个 word 的测试代码（经过更改该代码在我的机器上运行正常）  
以后有会找时间，推出其他的 office 代码

  
  import  com.jacob.activeX.ActiveXComponent;  
 import  com.jacob.com.ComException;  
 import  com.jacob.com.Dispatch;  
 import  com.jacob.com.Variant;  
  
 public   class  WordDocumentProperties  {  
     // 声明一个 word 对象   
     private  ActiveXComponent objWord;  
  
     // 声明四个 word 组件   
     private  Dispatch custDocprops;  
  
     private  Dispatch builtInDocProps;  
  
    
  
     private  Dispatch document;  
  
     private  Dispatch wordObject;  
  
     
     public  WordDocumentProperties() {  
    }   
  
     /\*\* \*/ /\*\*   
     \*  打开 word 文挡  
      \*/   
     public   void  open(String filename)  {  
         // 创建一个 word 对象   
        objWord  =   new  ActiveXComponent( " Word.Application " );  
  
         // 为 wordobject 组件附值   
       wordObject  =  (Dispatch)(objWord.getObject());            // 改了这里  
  
         // 生成一个只读方式的 word 文挡组件   
        Dispatch.put(wordObject,  " Visible " ,  new  Variant( false ));  
  
         // 获取文挡属性   
        Dispatch documents  =  objWord.getProperty( " Documents " ).toDispatch();  
  
         // 打开激活文挡   
        document  =  Dispatch.call(documents,  " Open " , filename).toDispatch();  
    }   
  
     
     public   void  selectCustomDocumentProperitiesMode()  {  
          
        custDocprops  =  Dispatch.get(document,  " CustomDocumentProperties " )  
                .toDispatch();  
    }   
  
     
     public   void  selectBuiltinPropertiesMode()  {  
          
        builtInDocProps  =  Dispatch.get(document,  " BuiltInDocumentProperties " )  
                .toDispatch();  
    }   
  
     /\*\* \*/ /\*\*   
     \*  关闭文挡     
      \*/   
     public   void  close()  {  
          
        Dispatch.call(document,  " Close " );  
    }   
  
     
     public  String getCustomProperty(String cusPropName)  {  
         try   {  
            cusPropName  =  Dispatch.call((Dispatch) custDocprops,  " Item " ,  
                    cusPropName).toString();  
        }   catch  (ComException e)  {  
            
            cusPropName  =   null ;  
        }   
  
         return  cusPropName;  
    }   
  
     public  String getBuiltInProperty(String builtInPropName)  {  
         try   {  
            builtInPropName  =  Dispatch.call((Dispatch) builtInDocProps,  " Item " ,  
                    builtInPropName).toString();  
        }   catch  (ComException e)  {  
             
            builtInPropName  =   null ;  
        }   
  
         return  builtInPropName;  
    }   
  
    
     public   static   void  main(String[] args)  {  
         try   {  
          
            WordDocumentProperties jacTest  =   new  WordDocumentProperties();  
  
            
            jacTest.open( " s.doc " );  
  
              
            jacTest.selectCustomDocumentProperitiesMode();  
  
              
            jacTest.selectBuiltinPropertiesMode();  
  
              
            String custValue  =  jacTest.getCustomProperty( " Information Source " );  
  
              
            String builtInValue  =  jacTest.getBuiltInProperty( " Author " );  
  
               
            jacTest.close();  
  
              
            System.out.println( " Document Val One:  "   +  custValue);  
            System.out.println( " Document Author:  "   +  builtInValue);  
  
        }   catch  (Exception e)  {  
           System.out.println(e);  
       }   
    }   
}

4. java jacob 操作 word 文档

进行写操作，如生成表格，添加 图片

jacob-1.15-M3-x86.dll   copy 到 c://windows/system32

引入 jacob.jar

示例代码

[java]   view plain copy

1. import  java.io.File;

2. import  com.jacob.activeX.ActiveXComponent;

3. import  com.jacob.com.Dispatch;

4. import  com.jacob.com.Variant;

5. class  WordBean {

6.      //  代表一个 word  程序

7.      private  ActiveXComponent MsWordApp =  null ;

8.      //  代表进行处理的 word  文档

9.      private  Dispatch document =  null ;

10.      public  WordBean() {

11.          // Open Word if we/'ve not done it already

12.          if  (MsWordApp ==  null ) {

13.             MsWordApp =  new  ActiveXComponent( "Word.Application" );

14.         }

15.     }

16.      //  设置是否在前台打开  word  程序   ，

17.      public   void  setVisible( boolean  visible) {

18.         MsWordApp.setProperty( "Visible" ,  new  Variant(visible));

19.          //  这一句作用相同

20.          // Dispatch.put(MsWordApp, "Visible", new Variant(visible));

21.     }

22.      //  创建一个新文档

23.      public   void  createNewDocument() {

24.          // Find the Documents collection object maintained by Word

25.          // documents 表示 word 的所有文档窗口，（ word 是多文档应用程序）

26.         Dispatch documents = Dispatch.get(MsWordApp,  "Documents" ).toDispatch();

27.          // Call the Add method of the Documents collection to create

28.          // a new document to edit

29.         document = Dispatch.call(documents,  "Add" ).toDispatch();

30.     }

31.      //  打开一个存在的 word 文档 , 并用 document  引用   引用它

32.      public   void  openFile(String wordFilePath) {

33.          // Find the Documents collection object maintained by Word

34.          // documents 表示 word 的所有文档窗口，（ word 是多文档应用程序）

35.         Dispatch documents = Dispatch.get(MsWordApp,  "Documents" ).toDispatch();

36.         document = Dispatch.call(documents,  "Open" , wordFilePath,

37.                  new  Variant( true ) /\*  是否进行转换 ConfirmConversions \*/ ,

38.                  new  Variant( false ) /\*  是否只读  \*/ ).toDispatch();

39.          // document = Dispatch.invoke(documents, "Open", Dispatch.Method,

40.          // new Object[] { wordFilePath, new Variant(true),

41.          // new Variant(false)

42.          // }, new int[1]).toDispatch();

43.     }

44.      //  向  document  中插入文本内容

45.      public   void  insertText(String textToInsert) {

46.          // Get the current selection within Word at the moment.

47.          // a new document has just been created then this will be at

48.          // the top of the new doc  获得选   中的内容，如果是一个新创建的文件，因里面无内容，则光标应处于文件开头处

49.         Dispatch selection = Dispatch.get(MsWordApp,  "Selection" ).toDispatch();

50.          //  取消选中 , 应该就是移动光标   ，否则   新添加的内容会覆盖选中的内容

51.         Dispatch.call(selection,  "MoveRight" ,  new  Variant( 1 ),  new  Variant( 1 ));

52.          // Put the specified text at the insertion point

53.         Dispatch.put(selection,  "Text" , textToInsert);

54.          //  取消选中 , 应该就是移动光标

55.         Dispatch.call(selection,  "MoveRight" ,  new  Variant( 1 ),  new  Variant( 1 ));

56.     }

57.      //  向文档中添加   一个图片，

58.      public   void  insertJpeg(String jpegFilePath) {

59.         Dispatch selection = Dispatch.get(MsWordApp,  "Selection" ).toDispatch();

60.         Dispatch image = Dispatch.get(selection,  "InLineShapes" ).toDispatch();

61.         Dispatch.call(image,  "AddPicture" , jpegFilePath);

62.     }

63.      //  段落的处理 , 插入格式化的文本

64.      public   void  insertFormatStr(String text) {

65.         Dispatch wordContent = Dispatch.get(document,  "Content" ).toDispatch();  //  取得 word 文件的内容

66.         Dispatch.call(wordContent,  "InsertAfter" , text); //  插入一个段落到最后

67.         Dispatch paragraphs = Dispatch.get(wordContent,  "Paragraphs" )

68.                 .toDispatch();  //  所有段落

69.          int  paragraphCount = Dispatch.get(paragraphs,  "Count" ).changeType(

70.                 Variant.VariantInt).getInt(); //  一共的段落数

71.          //  找到刚输入的段落，设置格式

72.         Dispatch lastParagraph = Dispatch.call(paragraphs,  "Item" ,

73.                  new  Variant(paragraphCount)).toDispatch();  //  最后一段（也就是刚插入的）

74.          // Range  对象表示文档中的一个连续范围，由一个起始字符位置和一个终止字符位置定义

75.         Dispatch lastParagraphRange = Dispatch.get(lastParagraph,  "Range" )

76.                 .toDispatch();

77.         Dispatch font = Dispatch.get(lastParagraphRange,  "Font" ).toDispatch();

78.         Dispatch.put(font,  "Bold" ,  new  Variant( true ));  //  设置为黑体

79.         Dispatch.put(font,  "Italic" ,  new  Variant( true ));  //  设置为斜体

80.         Dispatch.put(font,  "Name" ,  new  Variant( " 宋体 " ));  //

81.         Dispatch.put(font,  "Size" ,  new  Variant( 12 ));  //  小四

82.         Dispatch selection = Dispatch.get(MsWordApp,  "Selection" ).toDispatch();

83.         Dispatch.call(selection,  "TypeParagraph" ); //  插入一个空行

84.         Dispatch alignment = Dispatch.get(selection,  "ParagraphFormat" )

85.                 .toDispatch(); //  段落格式

86.         Dispatch.put(alignment,  "Alignment" ,  "2" );  // (1: 置中  2: 靠右  3: 靠左 )

87.     }

88.      // word  中在对表格进行遍历的时候   ，是先列后行   先 column  后 cell

89.      //  另外下标从 1 开始

90.      public   void  insertTable(String tableTitle,  int  row,  int  column) {

91.         Dispatch selection = Dispatch.get(MsWordApp,  "Selection" ).toDispatch();  //  输入内容需要的对象

92.         Dispatch.call(selection,  "TypeText" , tableTitle);  //  写入标题内容  //  标题格行

93.         Dispatch.call(selection,  "TypeParagraph" );  //  空一行段落

94.         Dispatch.call(selection,  "TypeParagraph" );  //  空一行段落

95.         Dispatch.call(selection,  "MoveDown" );  //  游标往下一行

96.          //  建立表格

97.         Dispatch tables = Dispatch.get(document,  "Tables" ).toDispatch();

98.          // int count = Dispatch.get(tables,

99.          // "Count").changeType(Variant.VariantInt).getInt(); // document 中的表格数量

100.          // Dispatch table = Dispatch.call(tables, "Item", new Variant(

101.          // 1)).toDispatch();// 文档中第一个表格

102.         Dispatch range = Dispatch.get(selection,  "Range" ).toDispatch(); // / 当前光标位置或者选中的区域

103.         Dispatch newTable = Dispatch.call(tables,  "Add" , range,

104.                  new  Variant(row),  new  Variant(column),  new  Variant( 1 ))

105.                 .toDispatch();  //  设置 row,column, 表格外框宽度

106.         Dispatch cols = Dispatch.get(newTable,  "Columns" ).toDispatch();  //  此表的所有列，

107.          int  colCount = Dispatch.get(cols,  "Count" ).changeType(

108.                 Variant.VariantInt).getInt(); //  一共有多少列   实际上这个数 ==column

109.         System.out.println(colCount +  " 列 " );

110.          for  ( int  i =  1 ; i <= colCount; i++) {  //  循环取出每一列

111.             Dispatch col = Dispatch.call(cols,  "Item" ,  new  Variant(i))

112.                     .toDispatch();

113.             Dispatch cells = Dispatch.get(col,  "Cells" ).toDispatch(); //  当前列中单元格

114.              int  cellCount = Dispatch.get(cells,  "Count" ).changeType(

115.                     Variant.VariantInt).getInt(); //  当前列中单元格数   实际上这个数等于 row

116.              for  ( int  j =  1 ; j <= cellCount; j++) { //  每一列中的单元格数

117.                  // Dispatch cell = Dispatch.call(cells, "Item", new

118.                  // Variant(j)).toDispatch(); // 当前单元格

119.                  // Dispatch cell = Dispatch.call(newTable, "Cell", new

120.                  // Variant(j) , new Variant(i) ).toDispatch(); // 取单元格的另一种方法

121.                  // Dispatch.call(cell, "Select");// 选中当前单元格

122.                  // Dispatch.put(selection, "Text",

123.                  // " 第 "+j+" 行，第 "+i+" 列 ");// 往选中的区域中填值，也就是往当前单元格填值

124.                 putTxtToCell(newTable, j, i,  " 第 "  + j +  " 行，第 "  + i +  " 列 " ); //  与上面四句的作用相同

125.             }

126.         }

127.     }

128.      /\*\* \*/

129.      /\*\*

130.      \*  在指定的单元格里填写数据

131.      \*

132.      \* @param tableIndex

133.      \* @param cellRowIdx

134.      \* @param cellColIdx

135.      \* @param txt

136.      \*/

137.      public   void  putTxtToCell(Dispatch table,  int  cellRowIdx,  int  cellColIdx,

138.             String txt) {

139.         Dispatch cell = Dispatch.call(table,  "Cell" ,  new  Variant(cellRowIdx),

140.                  new  Variant(cellColIdx)).toDispatch();

141.         Dispatch.call(cell,  "Select" );

142.         Dispatch selection = Dispatch.get(MsWordApp,  "Selection" ).toDispatch();  //  输入内容需要的对象

143.         Dispatch.put(selection,  "Text" , txt);

144.     }

145.      /\*\* \*/

146.      /\*\*

147.      \*  在指定的单元格里填写数据

148.      \*

149.      \* @param tableIndex

150.      \* @param cellRowIdx

151.      \* @param cellColIdx

152.      \* @param txt

153.      \*/

154.      public   void  putTxtToCell( int  tableIndex,  int  cellRowIdx,  int  cellColIdx,

155.             String txt) {

156.          //  所有表格

157.         Dispatch tables = Dispatch.get(document,  "Tables" ).toDispatch();

158.          //  要填充的表格

159.         Dispatch table = Dispatch.call(tables,  "Item" ,  new  Variant(tableIndex))

160.                 .toDispatch();

161.         Dispatch cell = Dispatch.call(table,  "Cell" ,  new  Variant(cellRowIdx),

162.                  new  Variant(cellColIdx)).toDispatch();

163.         Dispatch.call(cell,  "Select" );

164.         Dispatch selection = Dispatch.get(MsWordApp,  "Selection" ).toDispatch();  //  输入内容需要的对象

165.         Dispatch.put(selection,  "Text" , txt);

166.     }

167.      //  合并两个单元格

168.      public   void  mergeCell(Dispatch cell1, Dispatch cell2) {

169.         Dispatch.call(cell1,  "Merge" , cell2);

170.     }

171.      public   void  mergeCell(Dispatch table,  int  row1,  int  col1,  int  row2,  int  col2) {

172.         Dispatch cell1 = Dispatch.call(table,  "Cell" ,  new  Variant(row1),

173.                  new  Variant(col1)).toDispatch();

174.         Dispatch cell2 = Dispatch.call(table,  "Cell" ,  new  Variant(row2),

175.                  new  Variant(col2)).toDispatch();

176.         mergeCell(cell1, cell2);

177.     }

178.      public   void  mergeCellTest() {

179.         Dispatch tables = Dispatch.get(document,  "Tables" ).toDispatch();

180.          int  tableCount = Dispatch.get(tables,  "Count" ).changeType(

181.                 Variant.VariantInt).getInt();  // document 中的表格数量

182.         Dispatch table = Dispatch.call(tables,  "Item" ,  new  Variant(tableCount))

183.                 .toDispatch(); //  文档中最后一个 table

184.         mergeCell(table,  1 ,  1 ,  1 ,  2 ); //  将 table  中 x=1,y=1  与 x=1,y=2 的两个单元格合并

185.     }

186.      // ========================================================

187.      /\*\* \*/

188.      /\*\*

189.      \*  把选定的内容或光标插入点向上移动

190.      \*

191.      \* @param pos

192.      \*             移动的距离

193.      \*/

194.      public   void  moveUp( int  pos) {

195.         Dispatch selection = Dispatch.get(MsWordApp,  "Selection" ).toDispatch();  //  输入内容需要的对象

196.          for  ( int  i =  0 ; i < pos; i++) {

197.              // MoveDown MoveLeft moveRight

198.              // moveStart ( Dispatch.call(selection, "HomeKey", new Variant(6));

199.              // )

200.              // moveEnd Dispatch.call(selection, "EndKey", new Variant(6));

201.             Dispatch.call(selection,  "MoveUp" );

202.         }

203.     }

204.      /\*\* \*/

205.      /\*\*

206.      \*  从选定内容或插入点开始查找文本

207.      \*

208.      \* @param toFindText

209.      \*             要查找的文本

210.      \* @return boolean true- 查找到并选中该文本， false- 未查找到文本

211.      \*/

212.      public   boolean  find(String toFindText) {

213.          if  (toFindText ==  null  || toFindText.equals( "" ))

214.              return   false ;

215.         Dispatch selection = Dispatch.get(MsWordApp,  "Selection" ).toDispatch();  //  输入内容需要的对象

216.          //  从 selection 所在位置开始查询

217.         Dispatch find = Dispatch.call(selection,  "Find" ).toDispatch();

218.          //  设置要查找的内容

219.         Dispatch.put(find,  "Text" , toFindText);

220.          //  向前查找

221.         Dispatch.put(find,  "Forward" ,  "True" );

222.          //  设置格式

223.         Dispatch.put(find,  "Format" ,  "True" );

224.          //  大小写匹配

225.         Dispatch.put(find,  "MatchCase" ,  "True" );

226.          //  全字匹配

227.         Dispatch.put(find,  "MatchWholeWord" ,  "True" );

228.          //  查找并选中

229.          return  Dispatch.call(find,  "Execute" ).getBoolean();

230.     }

231.      /\*\* \*/

232.      /\*\*

233.      \*  把选定选定内容设定为替换文本

234.      \*

235.      \* @param toFindText

236.      \*             查找字符串

237.      \* @param newText

238.      \*             要替换的内容

239.      \* @return

240.      \*/

241.      public   boolean  replaceText(String toFindText, String newText) {

242.          if  (!find(toFindText))

243.              return   false ;

244.         Dispatch selection = Dispatch.get(MsWordApp,  "Selection" ).toDispatch();  //  输入内容需要的对象

245.         Dispatch.put(selection,  "Text" , newText);

246.          return   true ;

247.     }

248.      public   void  printFile() {

249.          // Just print the current document to the default printer

250.         Dispatch.call(document,  "PrintOut" );

251.     }

252.      //  保存文档的更改

253.      public   void  save() {

254.         Dispatch.call(document,  "Save" );

255.     }

256.      public   void  saveFileAs(String filename) {

257.         Dispatch.call(document,  "SaveAs" , filename);

258.     }

259.      public   void  closeDocument() {

260.          // Close the document without saving changes

261.          // 0 = wdDoNotSaveChanges

262.          // -1 = wdSaveChanges

263.          // -2 = wdPromptToSaveChanges

264.         Dispatch.call(document,  "Close" ,  new  Variant( 0 ));

265.         document =  null ;

266.     }

267.      public   void  closeWord() {

268.         Dispatch.call(MsWordApp,  "Quit" );

269.         MsWordApp =  null ;

270.         document =  null ;

271.     }

272.      //  设置 wordApp 打开后窗口的位置

273.      public   void  setLocation() {

274.         Dispatch activeWindow = Dispatch.get(MsWordApp,  "Application" )

275.                 .toDispatch();

276.         Dispatch.put(activeWindow,  "WindowState" ,  new  Variant( 1 ));  // 0=default

277.          // 1=maximize

278.          // 2=minimize

279.         Dispatch.put(activeWindow,  "Top" ,  new  Variant( 0 ));

280.         Dispatch.put(activeWindow,  "Left" ,  new  Variant( 0 ));

281.         Dispatch.put(activeWindow,  "Height" ,  new  Variant( 600 ));

282.         Dispatch.put(activeWindow,  "width" ,  new  Variant( 800 ));

283.     }

284. }

285. public   class  JacobTest2 {

286.      public   static   void  createANewFileTest() {

287.         WordBean wordBean =  new  WordBean();

288.          // word.openWord(true);//  打开  word  程序

289.         wordBean.setVisible( true );

290.         wordBean.createNewDocument(); //  创建一个新文档

291.         wordBean.setLocation(); //  设置打开后窗口的位置

292.         wordBean.insertText( " 你好 " ); //  向文档中插入字符

293.         wordBean.insertJpeg( "D:"  + File.separator +  "a.jpg" );  //  插入图片

294.          //  如果   ，想保存文件，下面三句

295.          // word.saveFileAs("d://a.doc");

296.          // word.closeDocument();

297.          // word.closeWord();

298.     }

299.      public   static   void  openAnExistsFileTest() {

300.         WordBean wordBean =  new  WordBean();

301.         wordBean.setVisible( true );  //  是否前台打开 word  程序，或者后台运行

302.         wordBean.openFile( "d://a.doc" );

303.         wordBean.insertJpeg( "D:"  + File.separator +  "a.jpg" );  //  插入图片 ( 注意刚打开的 word

304.          //  ，光标处于开头，故，图片在最前方插入 )

305.         wordBean.save();

306.         wordBean.closeDocument();

307.         wordBean.closeWord();

308.     }

309.      public   static   void  insertFormatStr(String str) {

310.         WordBean wordBean =  new  WordBean();

311.         wordBean.setVisible( true );  //  是否前台打开 word  程序，或者后台运行

312.         wordBean.createNewDocument(); //  创建一个新文档

313.         wordBean.insertFormatStr(str); //  插入一个段落，对其中的字体进行了设置

314.     }

315.      public   static   void  insertTableTest() {

316.         WordBean wordBean =  new  WordBean();

317.         wordBean.setVisible( true );  //  是否前台打开 word  程序，或者后台运行

318.         wordBean.createNewDocument(); //  创建一个新文档

319.         wordBean.setLocation();

320.         wordBean.insertTable( " 表名 " ,  3 ,  2 );

321.         wordBean.saveFileAs( "d://table.doc" );

322.         wordBean.closeDocument();

323.         wordBean.closeWord();

324.     }

325.      public   static   void  mergeTableCellTest() {

326.         insertTableTest(); // 生成 d://table.doc

327.         WordBean wordBean =  new  WordBean();

328.         wordBean.setVisible( true );  //  是否前台打开 word  程序，或者后台运行

329.         wordBean.openFile( "d://table.doc" );

330.         wordBean.mergeCellTest();

331.     }

332.      public   static   void  main(String[] args) {

333.          //  进行测试前要保证 d://a.jpg  图片文件存在

334.          // createANewFileTest();// 创建一个新文件

335.          // openAnExistsFileTest();//  打开一个存在   的文件

336.          // insertFormatStr(" 格式   化字符串 ");// 对字符串进行一定的修饰

337.          //insertTableTest();//  创建一个表格

338.        mergeTableCellTest(); //  对表格中的单元格进行合并

339.     }

340. }

5. java Jacob 最全例子

package com.sgcc.evoms.cs.util;

import com.jacob.activeX.ActiveXComponent;

import com.jacob.com.Dispatch;

import com.jacob.com.Variant;

public class MSWordManager {

// word 文档

private Dispatch doc;

// word 运行程序对象

private ActiveXComponent word;

// 所有 word 文档集合

private Dispatch documents;

// 选定的范围或插入点

private Dispatch selection;

private boolean saveOnExit = true;

/\*\* \*//\*\*

\*

\* @param visible 为 true 表示 word 应用程序可见

\*/

public MSWordManager(boolean visible) {

if (word == null) {

word = new ActiveXComponent("Word.Application");

word.setProperty("Visible", new Variant(visible));

}

if (documents == null)

documents = word.getProperty("Documents").toDispatch();

}

/\*\* \*//\*\*

\* 设置退出时参数

\*

\* @param saveOnExit boolean true- 退出时保存文件， false- 退出时不保存文件

\*/

public void setSaveOnExit(boolean saveOnExit) {

this.saveOnExit = saveOnExit;

}

/\*\* \*//\*\*

\* 创建一个新的 word 文档

\*

\*/

public void createNewDocument() {

doc = Dispatch.call(documents, "Add").toDispatch();

selection = Dispatch.get(word, "Selection").toDispatch();

}

/\*\* \*//\*\*

\* 打开一个已存在的文档

\*

\* @param docPath

\*/

public void openDocument(String docPath) {

closeDocument();

doc = Dispatch.call(documents, "Open", docPath).toDispatch();

selection = Dispatch.get(word, "Selection").toDispatch();

}

/\*\* \*//\*\*

\* 把选定的内容或插入点向上移动

\*

\* @param pos 移动的距离

\*/

public void moveUp(int pos) {

if (selection == null)

selection = Dispatch.get(word, "Selection").toDispatch();

for (int i = 0; i < pos; i++)

Dispatch.call(selection, "MoveUp");

}

/\*\* \*//\*\*

\* 把选定的内容或者插入点向下移动

\*

\* @param pos 移动的距离

\*/

public void moveDown(int pos) {

if (selection == null)

selection = Dispatch.get(word, "Selection").toDispatch();

for (int i = 0; i < pos; i++)

Dispatch.call(selection, "MoveDown");

}

/\*\* \*//\*\*

\* 把选定的内容或者插入点向左移动

\*

\* @param pos 移动的距离

\*/

public void moveLeft(int pos) {

if (selection == null)

selection = Dispatch.get(word, "Selection").toDispatch();

for (int i = 0; i < pos; i++) {

Dispatch.call(selection, "MoveLeft");

}

}

/\*\* \*//\*\*

\* 把选定的内容或者插入点向右移动

\*

\* @param pos 移动的距离

\*/

public void moveRight(int pos) {

if (selection == null)

selection = Dispatch.get(word, "Selection").toDispatch();

for (int i = 0; i < pos; i++)

Dispatch.call(selection, "MoveRight");

}

/\*\* \*//\*\*

\* 把插入点移动到文件首位置

\*

\*/

public void moveStart() {

if (selection == null)

selection = Dispatch.get(word, "Selection").toDispatch();

Dispatch.call(selection, "HomeKey", new Variant(6));

}

/\*\* \*//\*\*

\* 从选定内容或插入点开始查找文本

\*

\* @param toFindText 要查找的文本

\* @return boolean true- 查找到并选中该文本， false- 未查找到文本

\*/

public boolean find(String toFindText) {

if (toFindText == null || toFindText.equals(""))

return false;

// 从 selection 所在位置开始查询

Dispatch find = word.call(selection, "Find").toDispatch();

// 设置要查找的内容

Dispatch.put(find, "Text", toFindText);

// 向前查找

Dispatch.put(find, "Forward", "True");

// 设置格式

Dispatch.put(find, "Format", "True");

// 大小写匹配

Dispatch.put(find, "MatchCase", "True");

// 全字匹配

Dispatch.put(find, "MatchWholeWord", "True");

// 查找并选中

return Dispatch.call(find, "Execute").getBoolean();

}

/\*\* \*//\*\*

\* 把选定选定内容设定为替换文本

\*

\* @param toFindText 查找字符串

\* @param newText 要替换的内容

\* @return

\*/

public boolean replaceText(String toFindText, String newText) {

if (!find(toFindText))

return false;

Dispatch.put(selection, "Text", newText);

return true;

}

/\*\* \*//\*\*

\* 全局替换文本

\*

\* @param toFindText 查找字符串

\* @param newText 要替换的内容

\*/

public void replaceAllText(String toFindText, String newText) {

while (find(toFindText)) {

Dispatch.put(selection, "Text", newText);

Dispatch.call(selection, "MoveRight");

}

}

/\*\* \*//\*\*

\* 在当前插入点插入字符串

\*

\* @param newText 要插入的新字符串

\*/

public void insertText(String newText) {

Dispatch.put(selection, "Text", newText);

}

/\*\* \*//\*\*

\*

\* @param toFindText 要查找的字符串

\* @param imagePath 图片路径

\* @return

\*/

public boolean replaceImage(String toFindText, String imagePath) {

if (!find(toFindText))

return false;

Dispatch.call(Dispatch.get(selection, "InLineShapes").toDispatch(),

"AddPicture", imagePath);

return true;

}

/\*\* \*//\*\*

\* 全局替换图片

\*

\* @param toFindText 查找字符串

\* @param imagePath 图片路径

\*/

public void replaceAllImage(String toFindText, String imagePath) {

while (find(toFindText)) {

Dispatch.call(Dispatch.get(selection, "InLineShapes").toDispatch(),

"AddPicture", imagePath);

Dispatch.call(selection, "MoveRight");

}

}

/\*\* \*//\*\*

\* 在当前插入点插入图片

\*

\* @param imagePath 图片路径

\*/

public void insertImage(String imagePath) {

Dispatch.call(Dispatch.get(selection, "InLineShapes").toDispatch(),

"AddPicture", imagePath);

}

/\*\* \*//\*\*

\* 合并单元格

\*

\* @param tableIndex

\* @param fstCellRowIdx

\* @param fstCellColIdx

\* @param secCellRowIdx

\* @param secCellColIdx

\*/

public void mergeCell(int tableIndex, int fstCellRowIdx, int fstCellColIdx,

int secCellRowIdx, int secCellColIdx) {

// 所有表格

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

// 要填充的表格

Dispatch table = Dispatch.call(tables, "Item", new Variant(tableIndex))

.toDispatch();

Dispatch fstCell = Dispatch.call(table, "Cell",

new Variant(fstCellRowIdx), new Variant(fstCellColIdx))

.toDispatch();

Dispatch secCell = Dispatch.call(table, "Cell",

new Variant(secCellRowIdx), new Variant(secCellColIdx))

.toDispatch();

Dispatch.call(fstCell, "Merge", secCell);

}

/\*\* \*//\*\*

\* 在指定的单元格里填写数据

\*

\* @param tableIndex

\* @param cellRowIdx

\* @param cellColIdx

\* @param txt

\*/

public void putTxtToCell(int tableIndex, int cellRowIdx, int cellColIdx,

String txt) {

// 所有表格

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

// 要填充的表格

Dispatch table = Dispatch.call(tables, "Item", new Variant(tableIndex))

.toDispatch();

Dispatch cell = Dispatch.call(table, "Cell", new Variant(cellRowIdx),

new Variant(cellColIdx)).toDispatch();

Dispatch.call(cell, "Select");

Dispatch.put(selection, "Text", txt);

}

/\*\* \*//\*\*

\* 在当前文档拷贝数据

\*

\* @param pos

\*/

public void copy(String toCopyText) {

moveStart();

if (this.find(toCopyText)) {

Dispatch textRange = Dispatch.get(selection, "Range").toDispatch();

Dispatch.call(textRange, "Copy");

}

}

/\*\* \*//\*\*

\* 在当前文档粘帖剪贴板数据

\*

\* @param pos

\*/

public void paste(String pos) {

moveStart();

if (this.find(pos)) {

Dispatch textRange = Dispatch.get(selection, "Range").toDispatch();

Dispatch.call(textRange, "Paste");

}

}

/\*\* \*//\*\*

\* 在当前文档指定的位置拷贝表格

\*

\* @param pos 当前文档指定的位置

\* @param tableIndex 被拷贝的表格在 word 文档中所处的位置

\*/

public void copyTable(String pos, int tableIndex) {

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

Dispatch table = Dispatch.call(tables, "Item", new Variant(tableIndex))

.toDispatch();

Dispatch range = Dispatch.get(table, "Range").toDispatch();

Dispatch.call(range, "Copy");

if (this.find(pos)) {

Dispatch textRange = Dispatch.get(selection, "Range").toDispatch();

Dispatch.call(textRange, "Paste");

}

}

/\*\* \*//\*\*

\* 在当前文档末尾拷贝来自另一个文档中的段落

\*

\* @param anotherDocPath 另一个文档的磁盘路径

\* @param tableIndex 被拷贝的段落在另一格文档中的序号 ( 从 1 开始 )

\*/

public void copyParagraphFromAnotherDoc(String anotherDocPath,

int paragraphIndex) {

Dispatch wordContent = Dispatch.get(doc, "Content").toDispatch(); // 取得当前文档的内容

Dispatch.call(wordContent, "InsertAfter", "$selection$");// 插入特殊符定位插入点

copyParagraphFromAnotherDoc(anotherDocPath, paragraphIndex,

"$selection$");

}

/\*\* \*//\*\*

\* 在当前文档指定的位置拷贝来自另一个文档中的段落

\*

\* @param anotherDocPath 另一个文档的磁盘路径

\* @param tableIndex 被拷贝的段落在另一格文档中的序号 ( 从 1 开始 )

\* @param pos 当前文档指定的位置

\*/

public void copyParagraphFromAnotherDoc(String anotherDocPath,

int paragraphIndex, String pos) {

Dispatch doc2 = null;

try {

doc2 = Dispatch.call(documents, "Open", anotherDocPath)

.toDispatch();

Dispatch paragraphs = Dispatch.get(doc2, "Paragraphs").toDispatch();

Dispatch paragraph = Dispatch.call(paragraphs, "Item",

new Variant(paragraphIndex)).toDispatch();

Dispatch range = Dispatch.get(paragraph, "Range").toDispatch();

Dispatch.call(range, "Copy");

if (this.find(pos)) {

Dispatch textRange = Dispatch.get(selection, "Range")

.toDispatch();

Dispatch.call(textRange, "Paste");

}

} catch (Exception e) {

e.printStackTrace();

} finally {

if (doc2 != null) {

Dispatch.call(doc2, "Close", new Variant(saveOnExit));

doc2 = null;

}

}

}

/\*\* \*//\*\*

\* 在当前文档指定的位置拷贝来自另一个文档中的表格

\*

\* @param anotherDocPath 另一个文档的磁盘路径

\* @param tableIndex 被拷贝的表格在另一格文档中的序号 ( 从 1 开始 )

\* @param pos 当前文档指定的位置

\*/

public void copyTableFromAnotherDoc(String anotherDocPath, int tableIndex,

String pos) {

Dispatch doc2 = null;

try {

doc2 = Dispatch.call(documents, "Open", anotherDocPath)

.toDispatch();

Dispatch tables = Dispatch.get(doc2, "Tables").toDispatch();

Dispatch table = Dispatch.call(tables, "Item",

new Variant(tableIndex)).toDispatch();

Dispatch range = Dispatch.get(table, "Range").toDispatch();

Dispatch.call(range, "Copy");

if (this.find(pos)) {

Dispatch textRange = Dispatch.get(selection, "Range")

.toDispatch();

Dispatch.call(textRange, "Paste");

}

} catch (Exception e) {

e.printStackTrace();

} finally {

if (doc2 != null) {

Dispatch.call(doc2, "Close", new Variant(saveOnExit));

doc2 = null;

}

}

}

/\*\* \*//\*\*

\* 在当前文档指定的位置拷贝来自另一个文档中的图片

\*

\* @param anotherDocPath 另一个文档的磁盘路径

\* @param shapeIndex 被拷贝的图片在另一格文档中的位置

\* @param pos 当前文档指定的位置

\*/

public void copyImageFromAnotherDoc(String anotherDocPath, int shapeIndex,

String pos) {

Dispatch doc2 = null;

try {

doc2 = Dispatch.call(documents, "Open", anotherDocPath)

.toDispatch();

Dispatch shapes = Dispatch.get(doc2, "InLineShapes").toDispatch();

Dispatch shape = Dispatch.call(shapes, "Item",

new Variant(shapeIndex)).toDispatch();

Dispatch imageRange = Dispatch.get(shape, "Range").toDispatch();

Dispatch.call(imageRange, "Copy");

if (this.find(pos)) {

Dispatch textRange = Dispatch.get(selection, "Range")

.toDispatch();

Dispatch.call(textRange, "Paste");

}

} catch (Exception e) {

e.printStackTrace();

} finally {

if (doc2 != null) {

Dispatch.call(doc2, "Close", new Variant(saveOnExit));

doc2 = null;

}

}

}

/\*\* \*//\*\*

\* 创建表格

\*

\* @param pos 位置

\* @param cols 列数

\* @param rows 行数

\*/

public void createTable(String pos, int numCols, int numRows) {

if (find(pos)) {

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

Dispatch range = Dispatch.get(selection, "Range").toDispatch();

Dispatch newTable = Dispatch.call(tables, "Add", range,

new Variant(numRows), new Variant(numCols)).toDispatch();

Dispatch.call(selection, "MoveRight");

}

}

/\*\* \*//\*\*

\* 在指定行前面增加行

\*

\* @param tableIndex word 文件中的第 N 张表 ( 从 1 开始 )

\* @param rowIndex 指定行的序号 ( 从 1 开始 )

\*/

public void addTableRow(int tableIndex, int rowIndex) {

// 所有表格

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

// 要填充的表格

Dispatch table = Dispatch.call(tables, "Item", new Variant(tableIndex))

.toDispatch();

// 表格的所有行

Dispatch rows = Dispatch.get(table, "Rows").toDispatch();

Dispatch row = Dispatch.call(rows, "Item", new Variant(rowIndex))

.toDispatch();

Dispatch.call(rows, "Add", new Variant(row));

}

/\*\* \*//\*\*

\* 在第 1 行前增加一行

\*

\* @param tableIndex word 文档中的第 N 张表 ( 从 1 开始 )

\*/

public void addFirstTableRow(int tableIndex) {

// 所有表格

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

// 要填充的表格

Dispatch table = Dispatch.call(tables, "Item", new Variant(tableIndex))

.toDispatch();

// 表格的所有行

Dispatch rows = Dispatch.get(table, "Rows").toDispatch();

Dispatch row = Dispatch.get(rows, "First").toDispatch();

Dispatch.call(rows, "Add", new Variant(row));

}

/\*\* \*//\*\*

\* 在最后 1 行前增加一行

\*

\* @param tableIndex

\* word 文档中的第 N 张表 ( 从 1 开始 )

\*/

public void addLastTableRow(int tableIndex) {

// 所有表格

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

// 要填充的表格

Dispatch table = Dispatch.call(tables, "Item", new Variant(tableIndex))

.toDispatch();

// 表格的所有行

Dispatch rows = Dispatch.get(table, "Rows").toDispatch();

Dispatch row = Dispatch.get(rows, "Last").toDispatch();

Dispatch.call(rows, "Add", new Variant(row));

}

/\*\* \*//\*\*

\* 增加一行

\*

\* @param tableIndex word 文档中的第 N 张表 ( 从 1 开始 )

\*/

public void addRow(int tableIndex) {

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

// 要填充的表格

Dispatch table = Dispatch.call(tables, "Item", new Variant(tableIndex))

.toDispatch();

// 表格的所有行

Dispatch rows = Dispatch.get(table, "Rows").toDispatch();

Dispatch.call(rows, "Add");

}

/\*\* \*//\*\*

\* 增加一列

\*

\* @param tableIndex word 文档中的第 N 张表 ( 从 1 开始 )

\*/

public void addCol(int tableIndex) {

// 所有表格

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

// 要填充的表格

Dispatch table = Dispatch.call(tables, "Item", new Variant(tableIndex))

.toDispatch();

// 表格的所有行

Dispatch cols = Dispatch.get(table, "Columns").toDispatch();

Dispatch.call(cols, "Add").toDispatch();

Dispatch.call(cols, "AutoFit");

}

/\*\* \*//\*\*

\* 在指定列前面增加表格的列

\*

\* @param tableIndex word 文档中的第 N 张表 ( 从 1 开始 )

\* @param colIndex 指定列的序号 ( 从 1 开始 )

\*/

public void addTableCol(int tableIndex, int colIndex) {

// 所有表格

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

// 要填充的表格

Dispatch table = Dispatch.call(tables, "Item", new Variant(tableIndex))

.toDispatch();

// 表格的所有行

Dispatch cols = Dispatch.get(table, "Columns").toDispatch();

System.out.println(Dispatch.get(cols, "Count"));

Dispatch col = Dispatch.call(cols, "Item", new Variant(colIndex))

.toDispatch();

// Dispatch col = Dispatch.get(cols, "First").toDispatch();

Dispatch.call(cols, "Add", col).toDispatch();

Dispatch.call(cols, "AutoFit");

}

/\*\* \*//\*\*

\* 在第 1 列前增加一列

\*

\* @param tableIndex word 文档中的第 N 张表 ( 从 1 开始 )

\*/

public void addFirstTableCol(int tableIndex) {

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

// 要填充的表格

Dispatch table = Dispatch.call(tables, "Item", new Variant(tableIndex))

.toDispatch();

// 表格的所有行

Dispatch cols = Dispatch.get(table, "Columns").toDispatch();

Dispatch col = Dispatch.get(cols, "First").toDispatch();

Dispatch.call(cols, "Add", col).toDispatch();

Dispatch.call(cols, "AutoFit");

}

/\*\* \*//\*\*

\* 在最后一列前增加一列

\*

\* @param tableIndex word 文档中的第 N 张表 ( 从 1 开始 )

\*/

public void addLastTableCol(int tableIndex) {

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

// 要填充的表格

Dispatch table = Dispatch.call(tables, "Item", new Variant(tableIndex))

.toDispatch();

// 表格的所有行

Dispatch cols = Dispatch.get(table, "Columns").toDispatch();

Dispatch col = Dispatch.get(cols, "Last").toDispatch();

Dispatch.call(cols, "Add", col).toDispatch();

Dispatch.call(cols, "AutoFit");

}

/\*\* \*//\*\*

\* 自动调整表格

\*

\*/

public void autoFitTable() {

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

int count = Dispatch.get(tables, "Count").toInt();

for (int i = 0; i < count; i++) {

Dispatch table = Dispatch.call(tables, "Item", new Variant(i + 1))

.toDispatch();

Dispatch cols = Dispatch.get(table, "Columns").toDispatch();

Dispatch.call(cols, "AutoFit");

}

}

/\*\* \*//\*\*

\* 调用 word 里的宏以调整表格的宽度 , 其中宏保存在 document 下

\*

\*/

public void callWordMacro() {

Dispatch tables = Dispatch.get(doc, "Tables").toDispatch();

int count = Dispatch.get(tables, "Count").toInt();

Variant vMacroName = new Variant("Normal.NewMacros.tableFit");

Variant vParam = new Variant("param1");

Variant para[] = new Variant[] { vMacroName };

for (int i = 0; i < para.length; i++) {

Dispatch table = Dispatch.call(tables, "Item", new Variant(i + 1))

.toDispatch();

Dispatch.call(table, "Select");

Dispatch.call(word, "Run", "tableFitContent");

}

}

/\*\* \*//\*\*

\* 设置当前选定内容的字体

\*

\* @param boldSize

\* @param italicSize

\* @param underLineSize 下划线

\* @param colorSize 字体颜色

\* @param size 字体大小

\* @param name 字体名称

\*/

public void setFont(boolean bold, boolean italic, boolean underLine,

String colorSize, String size, String name) {

Dispatch font = Dispatch.get(selection, "Font").toDispatch();

Dispatch.put(font, "Name", new Variant(name));

Dispatch.put(font, "Bold", new Variant(bold));

Dispatch.put(font, "Italic", new Variant(italic));

Dispatch.put(font, "Underline", new Variant(underLine));

Dispatch.put(font, "Color", colorSize);

Dispatch.put(font, "Size", size);

}

/\*\* \*//\*\*

\* 文件保存或另存为

\*

\* @param savePath 保存或另存为路径

\*/

public void save(String savePath) {

Dispatch.call(

(Dispatch) Dispatch.call(word, "WordBasic").getDispatch(),

"FileSaveAs", savePath);

}

/\*\* \*//\*\*

\* 关闭当前 word 文档

\*

\*/

public void closeDocument() {

if (doc != null) {

Dispatch.call(doc, "Save");

Dispatch.call(doc, "Close", new Variant(saveOnExit));

doc = null;

}

}

/\*\* \*//\*\*

\* 关闭全部应用

\*

\*/

public void close() {

closeDocument();

if (word != null) {

Dispatch.call(word, "Quit");

word = null;

}

selection = null;

documents = null;

}

/\*\* \*//\*\*

\* 打印当前 word 文档

\*

\*/

public void printFile() {

if (doc != null) {

Dispatch.call(doc, "PrintOut");

}

}

public static void main(String args[]) {

MSWordManager msWordManager = new MSWordManager(false);

try {

msWordManager.openDocument("D:\\ 退费 .doc");

msWordManager.replaceAllText("tradeFlowNo", "888");

msWordManager.replaceAllText("consName", "laingjing");

msWordManager.replaceAllText("bussType", " 缴费 ");

msWordManager.replaceAllText("payWay", "123456");

msWordManager.replaceAllText("rmbHanzi", "8956");

msWordManager.save("D:\\ 退费 1.doc");

msWordManager.printFile();

msWordManager.close();

} catch (Exception e) {

e.printStackTrace();

}

msWordManager.close();

}

}