Jacob 操作 office 文档（ Word,PPT,Excel ）

jacob 操作 word

1. public   boolean  doc2pdf(String srcFilePath, String pdfFilePath) {

2.         ActiveXComponent app =  null ;

3.         Dispatch doc =  null ;

4.          try  {

5.             ComThread.InitSTA();

6.             app =  new  ActiveXComponent( "Word.Application" );

7.             app.setProperty( "Visible" ,  false );

8.             Dispatch docs = app.getProperty( "Documents" ).toDispatch();

9.             doc = Dispatch.invoke(docs,  "Open" , Dispatch.Method,

10.                      new  Object[] { srcFilePath,

11.                                                   new  Variant( false ),

12.                                                   new  Variant( true ), // 是否只读

13.                                                   new  Variant( false ),

14.                                                   new  Variant( "pwd" ) },

15.                      new   int [ 1 ]).toDispatch();

16. //          Dispatch.put(doc, "Compatibility", false);  // 兼容性检查 , 为特定值 false 不正确

17.             Dispatch.put(doc,  "RemovePersonalInformation" ,  false );

18.             Dispatch.call(doc,  "ExportAsFixedFormat" , pdfFilePath, wdFormatPDF);  // word 保存为 pdf 格式宏，值为 17

19.

20.              return   true ;  // set flag true;

21.         }  catch  (ComFailException e) {

22.              return   false ;

23.         }  catch  (Exception e) {

24.              return   false ;

25.         }  finally  {

26.              if  (doc !=  null ) {

27.                 Dispatch.call(doc,  "Close" ,  false );

28.             }

29.              if  (app !=  null ) {

30.                 app.invoke( "Quit" ,  0 );

31.             }

32.             ComThread.Release();

33.         }

34.     }

Jacob 操作 ppt

1. public   boolean  ppt2pdf(String srcFilePath, String pdfFilePath) {

2.         ActiveXComponent app =  null ;

3.         Dispatch ppt =  null ;

4.              try  {

5.                 ComThread.InitSTA();

6.                 app =  new  ActiveXComponent( "PowerPoint.Application" );

7.                 Dispatch ppts = app.getProperty( "Presentations" ).toDispatch();

8.

9.                  //  因 POWER.EXE 的发布规则为同步，所以设置为同步发布

10.                 ppt = Dispatch.call(ppts,  "Open" , srcFilePath,  true , // ReadOnly

11.                          true , // Untitled 指定文件是否有标题

12.                          false // WithWindow 指定文件是否可见

13.                         ).toDispatch();

14.

15.                 Dispatch.call(ppt,  "SaveAs" , pdfFilePath, ppSaveAsPDF);  //ppSaveAsPDF 为特定值 32

16.

17.                  return   true ;  // set flag true;

18.             }  catch  (ComFailException e) {

19.                  return   false ;

20.             }  catch  (Exception e) {

21.                  return   false ;

22.             }  finally  {

23.                  if  (ppt !=  null ) {

24.                     Dispatch.call(ppt,  "Close" );

25.                 }

26.                  if  (app !=  null ) {

27.                     app.invoke( "Quit" );

28.                 }

29.                 ComThread.Release();

30.             }

31.     }

Jacob 操作 Excel

1. package  com;

2. import  java.io.ObjectInputStream.GetField;

3. import  java.util.ArrayList;

4. import  java.util.Date;

5. import  java.util.List;

6.

7. import  com.jacob.activeX.ActiveXComponent;

8. import  com.jacob.com.ComThread;

9. import  com.jacob.com.Dispatch;

10. import  com.jacob.com.Variant;

11.

12.

13. public   class  ready {

14.      private   static  ActiveXComponent xl =  null ;  //Excel 对象 ( 防止打开多个 )

15.      private   static  Dispatch workbooks =  null ;   // 工作簿对象

16.      private  Dispatch workbook =  null ;  // 具体工作簿

17.      private  Dispatch sheets =  null ; //  获得 sheets 集合对象

18.      private  Dispatch currentSheet =  null ; //  当前 sheet

19.      /\*\*

20.      \*  打开 excel 文件

21.      \* @param filepath  文件路径名称

22.      \* @param visible   是否显示打开

23.      \* @param readonly  是否只读方式打开

24.      \*/

25.      private   void  OpenExcel(String filepath,  boolean  visible) {

26.          try  {

27.             initComponents();  // 清空原始变量

28.             ComThread.InitSTA();

29.              if (xl== null )

30.                 xl =  new  ActiveXComponent( "Excel.Application" );  //Excel 对象

31.             xl.setProperty( "Visible" ,  new  Variant(visible)); // 设置是否显示打开 excel

32.              if (workbooks== null )

33.                 workbooks = xl.getProperty( "Workbooks" ).toDispatch();  // 打开具体工作簿

34.                 workbook = Dispatch.invoke(workbooks,  "Open" , Dispatch.Method,

35.                 new  Object[] { srcFilePath,

36.                                          new  Variant( false ),  //  是否以只读方式打开

37.                                          new  Variant( true ),

38.                                           "1" ,

39.                                          "pwd"  },    // 输入密码 "pwd", 若有密码则进行匹配，无则直接打开

40.                                           new   int [ 1 ]).toDispatch();

41.         }  catch  (Exception e) {

42.             e.printStackTrace();

43.             releaseSource();

44.         }

45.     }

46.      /\*\*

47.      \*  工作簿另存为

48.      \* @param filePath  另存为的路径

49.      \*  例如  SaveAs="D:TEST/c.xlsx"

50.      \*/

51.      private   void  SaveAs(String filePath){

52.            Dispatch.call(workbook,  "SaveAs" ,filePath);

53.       }

54.      /\*\*

55.      \*  关闭 excel 文档

56.      \* @param f  含义不明   （关闭是否保存？默认 false ）

57.      \*/

58.      private   void  CloseExcel( boolean  f) {

59.          try  {

60.             Dispatch.call(workbook,  "Save" );

61.             Dispatch.call(workbook,  "Close" ,  new  Variant(f));

62.         }  catch  (Exception e) {

63.             e.printStackTrace();

64.         }  finally  {

65.                 releaseSource();

66.         }

67.     }

68.      /\*

69.      \*  初始化

70.      \* \*/

71.      private   void  initComponents(){

72.         workbook =  null ;

73.         currentSheet =  null ;

74.         sheets =  null ;

75.     }

76.      /\*\*

77.      \*  释放资源

78.      \*/

79.      private   static   void  releaseSource(){

80.          if (xl!= null ){

81.             xl.invoke( "Quit" ,  new  Variant[] {});

82.             xl =  null ;

83.         }

84.         workbooks =  null ;

85.         ComThread.Release();

86.         System.gc();

87.     }

88.      /\*\*

89.      \*  得到当前 sheet

90.      \* @return

91.      \*/

92.      private  Dispatch getCurrentSheet() {

93.         currentSheet = Dispatch.get(workbook,  "ActiveSheet" ).toDispatch();

94.          return  currentSheet;

95.     }

96.      /\*\*

97.      \*  修改当前工作表的名字

98.      \* @param newName

99.      \*/

100.      private   void  modifyCurrentSheetName(String newName) {

101.         Dispatch.put(getCurrentSheet(),  "name" , newName);

102.     }

103.

104.      /\*\*

105.      \*  得到当前工作表的名字

106.      \* @return

107.      \*/

108.      private  String getCurrentSheetName(Dispatch sheets) {

109.          return  Dispatch.get(sheets,  "name" ).toString();

110.     }

111.      /\*\*

112.      \*  通过工作表名字得到工作表

113.      \* @param name sheetName

114.      \* @return

115.      \*/

116.      private  Dispatch getSheetByName(String name) {

117.          return  Dispatch.invoke(getSheets(),  "Item" , Dispatch.Get,  new  Object[]{name},  new   int [ 1 ]).toDispatch();

118.     }

119.      /\*\*

120.      \*   得到 sheets 的集合对象

121.      \* @return

122.      \*/

123.      private  Dispatch getSheets() {

124.          if (sheets== null )

125.             sheets = Dispatch.get(workbook,  "sheets" ).toDispatch();

126.          return  sheets;

127.     }

128.      /\*\*

129.      \*  通过工作表索引得到工作表 ( 第一个工作簿 index 为 1)

130.      \* @param index

131.      \* @return  sheet 对象

132.      \*/

133.      private  Dispatch getSheetByIndex(Integer index) {

134.          return  Dispatch.invoke(getSheets(),  "Item" , Dispatch.Get,  new  Object[]{index},  new   int [ 1 ]).toDispatch();

135.     }

136.

137.      /\*\*

138.      \*  得到 sheet 的总数

139.      \* @return

140.      \*/

141.      private   int  getSheetCount() {

142.          int  count = Dispatch.get(getSheets(),  "count" ).toInt();

143.          return  count;

144.     }

145.      /\*\*

146.      \*  给所有的 sheet 添加背景

147.      \* @param filepath  图片路径

148.      \*/

149.      public   void  setBlackGroudPrituce(String filepath)

150.     {

151.          int  num= this .getSheetCount();

152.          for  ( int  i =  1 ; i <= num; i++) {

153.             Dispatch sheets= this .getSheetByIndex(i);

154.             Dispatch.call(sheets, "SetBackgroundPicture" ,filepath);

155.         }

156.     }

157.      /\*\*

158.      \*   添加新的工作表 (sheet) ，并且隐藏（添加后为默认为当前激活的工作表）

159.      \*/

160.      public   void  addSheet(String name) {

161. //      for (int i = 1; i <= this.getSheetCount(); i++) {

162. //          Dispatch sheets=this.getSheetByIndex(i);

163. //         if(name.equals(this.getCurrentSheetName(sheets)))

164. //            {

165. //                return false;

166. //             }

167. //         }

168.           currentSheet=Dispatch.get(Dispatch.get(workbook,  "sheets" ).toDispatch(),  "add" ).toDispatch();

169.          //  Dispatch.put(currentSheet,"Name",name);

170.           Dispatch.put(currentSheet,  "Visible" ,  new  Boolean( false ));

171.           System.out.println( " 插入信息为 :" +name);

172.     }

173.      /\*\*

174.      \*  得到工作薄的名字

175.      \* @return

176.      \*/

177.      private  String getWorkbookName() {

178.          if (workbook== null )

179.              return   null ;

180.          return  Dispatch.get(workbook,  "name" ).toString();

181.     }

182.      /\*\*

183.      \*   获取所有表名

184.      \*/

185.      public  List findSheetName()

186.     {

187.          int  num= this .getSheetCount();

188.         List list= new  ArrayList();

189.          for  ( int  i =  1 ; i <= num; i++) {

190.             currentSheet= this .getSheetByIndex(i);

191.             list.add( this .getCurrentSheetName(currentSheet));

192.            }

193.          return  list;

194.     }

195.      /\*\*

196.      \*  设置页脚信息

197.      \*/

198.      private   void  setFooter(String foot) {

199.         currentSheet= this .getCurrentSheet();

200.         Dispatch PageSetup=Dispatch.get(currentSheet, "PageSetup" ).toDispatch();

201.         Dispatch.put(PageSetup, "CenterFooter" ,foot);

202.     }

203.      /\*\*

204.      \*  获取页脚信息

205.      \*/

206.      private  String getFooter() {

207.         currentSheet= this .getCurrentSheet();

208.         Dispatch PageSetup=Dispatch.get(currentSheet, "PageSetup" ).toDispatch();

209.          return  Dispatch.get(PageSetup, "CenterFooter" ).toString();

210.     }

211.      /\*\*

212.      \*  锁定工作簿

213.      \*/

214.      private   void  setPassword() {

215.         Dispatch.call(workbook,  "Protect" , 123 , true , false );

216.     }

217.      /\*\*

218.      \*  设置名称管理器

219.      \* @param name  名称管理器名   不能以数字或者下划线开头，中间不能包含空格和其他无效字符

220.      \* @param comment  备注

221.      \* @param place  备注位置

222.      \* @return

223.      \*/

224.      public   void  setName(String name,String place,String comment) {

225.         Dispatch Names=Dispatch.get(workbook,  "Names" ).toDispatch();

226.         Dispatch.call(Names, "Add" ,name,place, false ).toDispatch();

227.         Dispatch.put(Names,  "Comment" , comment);  // 插入备注

228.     }

229.      /\*\*

230.      \*  获取名称管理器

231.      \* @param name  名称管理器名

232.      \* @return

233.      \*/

234.      public  String getName(String name) {

235.         Dispatch Names=Dispatch.get(workbook,  "Names" ).toDispatch();

236.         Dispatch Name=Dispatch.call(Names, "Item" ,name).toDispatch();

237.          return  Dispatch.get(Name,  "Value" ).toString();

238.     }

239.      /\*\*

240.      \*   单元格写入值

241.      \* @param sheet   被操作的 sheet

242.      \* @param position  单元格位置，如： C1

243.      \* @param type  值的属性   如： value

244.      \* @param value

245.      \*/

246.      private   void  setValue(String position, Object value) {

247.         currentSheet= this .getCurrentSheet();

248.         Dispatch cell = Dispatch.invoke(currentSheet,  "Range" ,

249.                 Dispatch.Get,  new  Object[] { position },  new   int [ 1 ])

250.                 .toDispatch();

251.         Dispatch.put(cell,  "Value" , value);

252.         String color= this .getColor(cell);

253.          this .setFont(cell,color);

254.     }

255.      /\*\*

256.      \*   设置字体

257.      \*/

258.      private   void  setFont(Dispatch cell,String color)

259.     {

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

261.          //Dispatch.put(font,"FontStyle", "Bold Italic");

262.         Dispatch.put(font, "size" ,  "1" );

263.         Dispatch.put(font, "color" ,color);

264.     }

265.      /\*\*

266.      \*   获取背景颜色

267.      \*/

268.      private  String getColor(Dispatch cell)

269.     {

270.         Dispatch Interior=Dispatch.get(cell,  "Interior" ).toDispatch();

271.         String color=Dispatch.get(Interior,  "color" ).toString();

272.          return  color;

273.     }

274.      /\*\*

275.      \*  单元格读取值

276.      \* @param position  单元格位置，如：  C1

277.      \* @param sheet

278.      \* @return

279.      \*/

280.      private  Variant getValue(String position) {

281.         currentSheet= this .getCurrentSheet();

282.         Dispatch cell = Dispatch.invoke(currentSheet,  "Range" , Dispatch.Get,

283.                  new  Object[] { position },  new   int [ 1 ]).toDispatch();

284.         Variant value = Dispatch.get(cell,  "Value" );

285.          return  value;

286.     }

287.      /\*\*

288.      \*  获取最大行数

289.      \* @return

290.      \*/

291.      private   int  getRowCount() {

292.         currentSheet= this .getCurrentSheet();

293.         Dispatch UsedRange=Dispatch.get(currentSheet,  "UsedRange" ).toDispatch();

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

295.          int  num=Dispatch.get(rows,  "count" ).getInt();

296.          return  num;

297.     }

298.      /\*\*

299.      \*  获取最大列数

300.      \* @return

301.      \*/

302.      private   int  getColumnCount() {

303.         currentSheet= this .getCurrentSheet();

304.         Dispatch UsedRange=Dispatch.get(currentSheet,  "UsedRange" ).toDispatch();

305.         Dispatch Columns=Dispatch.get(UsedRange,  "Columns" ).toDispatch();

306.          int  num=Dispatch.get(Columns,  "count" ).getInt();

307.          return  num;

308.     }

309.      /\*\*

310.      \*  获取位置

311.      \* @param rnum  最大行数

312.      \* @param cnum  最大列数

313.      \*/

314.      private  String getCellPosition( int  rnum, int  cnum)

315.     {

316.           String cposition= "" ;

317.            if (cnum> 26 )

318.           {

319.                int  multiple=(cnum)/ 26 ;

320.                int  remainder=(cnum)% 26 ;

321.                char  mchar=( char )(multiple+ 64 );

322.                char  rchar=( char )(remainder+ 64 );

323.               cposition=mchar+ "" +rchar;

324.           }

325.            else

326.           {

327.               cposition=( char )(cnum+ 64 )+ "" ;

328.           }

329.           cposition+=rnum;

330.            return  cposition;

331.     }

332.

333.       /\*

334.       \*  取消兼容性检查，在保存或者另存为时改检查会导致弹窗

335.       \*/

336.      private  viod setCheckCompatibility(){

337.         Dispatch.put(wookbook,  "CheckCompatibility" ,  false );

338.    }

339.

340.       /\*

341.       \*   为每个表设置打印区域

342.       \*/

343.      private   void  setPrintArea(){

344.          int  count = Dispatch.get(sheets,  "count" ).changeType(Variant.VariantInt).getInt();

345.          for  ( int  i = count; i >=  1 ; i--) {

346.                sheet = Dispatch.invoke(sheets,  "Item" ,

347.                        Dispatch.Get,  new  Object[] { i },  new   int [ 1 ]).toDispatch();

348.            Dispatch page = Dispatch.get(sheet,  "PageSetup" ).toDispatch();

349.            Dispatch.put(page,  "PrintArea" ,  false );

350.            Dispatch.put(page,  "Orientation" ,  2 );

351.            Dispatch.put(page,  "Zoom" ,  false );       // 值为 100 或 false

352.            Dispatch.put(page,  "FitToPagesTall" ,  false );   // 所有行为一页

353.            Dispatch.put(page,  "FitToPagesWide" ,  1 );       // 所有列为一页 (1 或 false)

354.     }

355.    }

356. }