[**有关jacob的word操作研究**](http://zhuxue-feng.javaeye.com/blog/35350)

关键字: jacob

程序测试环境：

windowsXP sp2

office2003

jacob\_1.11-pre1.zip jacob开发包

package com.langchaoauto.eimip.common;

import java.util.HashMap;

import com.jacob.activeX.ActiveXComponent;  
import com.jacob.com.Dispatch;  
import com.jacob.com.Variant;

/\*\*  
 \*   
 \* @author zhuzhen\_hua@yahoo.com.cn  
 \*   
 \*/

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;

public MSWordManager() {  
 if (word == null) {  
 word = new ActiveXComponent("Word.Application");  
 word.setProperty("Visible", new Variant(false));  
 }  
 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);  
 }  
 /\*\*  
 \* 在当前文档拷贝剪贴板数据  
 \* @param pos  
 \*/  
 public void pasteExcelSheet(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) {  
 // 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 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位置  
 \* @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();  
 // 所有表格  
 Dispatch tables = Dispatch.get(doc2, "Tables").toDispatch();  
 // 要填充的表格  
 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  
 \* 制定列的序号 (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 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 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.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();  
 try {  
 msWordManager.openDocument("D:\\SUPERADMIN20061121100.doc");  
 msWordManager.callWordMacro();  
   
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 msWordManager.close();  
 }

}