**永中Office二次开发技术手册**

## 前言

#### 一、宏和二次开发

在实现办公自动化的今天，我们经常会碰到这样一个问题，在使用办公软件时总要重复流程一样的相同操作，这使工作变得繁琐而枯燥，能不能提供一种方法，使这些相同操作能够通过一个命令自动完成或者使用更为简单的途径得到呢？宏因此而产生了。

宏是指用一个符号名称所代表的一段程序代码，将定义代码段按实际参数进行加工，形成实际代码直接插入程序的执行序列中。许多应用程序具有建立宏的功能，它们可以把操作人员为解决某项问题而执行的一些操作命令记下来，形成一个文件段，并且用一个符号表示。当同样的问题重复执行时，只要输入代表这一系列命令的宏命令即可。这些代码可以是由用户编写，也可以通过办公软件提供的宏录制功能记录下简单任务而生成。

结合自己的需要，利用宏充分开发应用程序潜在的功能，可以获得使用上最大限度的满足，这也就是通常意义上的二次开发，即利用宏语言扩展宿主程序的功能、集成不同类型的数据以实现办公软件在具体事件中的解决方案。

永中Office为您提供了非常丰富而全面的宏API（API表示应用程序接口），通过这些API您可以非常自如操作永中Office。并且使您在实现自定义的解决方案时更加得心应手，更灵活的发挥永中Office 的集成优势。

#### 二、本手册结构

**第一部分为二次开发环境。**

永中Office的二次开发，我们推荐，一是利用永中Office自带的宏编辑器进行开发，二是可以在Eclipse上开发。

**第二部分为应用和API**

我们在第2章给出了二次开发结构，也即永中Office二次开发的类层次图。有了该类层次图，就可以了解永中Office二次开发中的层次和对象了。

第3章为总控,这里有二次开发应用的总入口

第4章为电子表格应用

第5章为文字处理应用,自由表格也在这一章

第6章为幻灯片演示

第7章为公共组件，包含2个主要组件：自选图形和图表

**第三部分为其它语言调用与应用**

第8章为永中Office COM组件开发

第9章为嵌入浏览器样例

第10章为嵌入Java应用程序样例

**附录**

附录一为永中Office 二次开发技巧集,我们列举了一些二次开发技巧

附录一为永中Office 二次开发FAQ

我们希望起到抛砖引玉的作用。详细的API说明可以参见永中Office的API文档。

## 第一章 二次开发环境

#### 1.1宏编辑器中进行二次开发

1、从菜单“工具”—》“宏”—》“宏编辑器”，打开宏编辑器。

2、在宏编辑器中，从菜单“插入”—》“模块”，新建一个模块。

3、在模块中，可以添加宏，通过宏，就可以操作永中Office。

4、示例：将下面的宏代码添加到模块中，宏的名字是“insertValue”。

**样例1.1.1**

//在当前的工作表中，往单元格（2，3）中设值“apple” 。

void insertValue()

{

Application.gotoProduct(GlobalConstants.SPREADSHEET);

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

sheet.setCellValue(2, 3, "apple");

}

**注意**：永中Office的宏语言是JAVA。

在模块中添加宏时，请不要添加方法的修饰符，如 public，private，protected，static，final 等。

永中Office的宏只能在永中Office自己的文件格式（.eio）中使用，保存为微软格式文件时宏代码会丢失。

#### 1.2使用Eclipse 进行二次开发

1. 下载并解压Eclipse包。
2. 在Eclipse目录下点击Eclipse可执行文件，启动Eclipse。
3. 在Eclipse中，新建一个项目：

在菜单“File”—》“New”—》“Project...”弹出“New Project”对话框，在列表中选中“Java Project”选项，下一步，给定一个项目名，假设项目名为“example”，完成，这样就建立了一个名为 “example”的项目。

4、在Eclipse的“Package Explorer”里，选中“example”节点，右键弹出菜单，点击“Properties”菜单项，在弹出的“Properties”对话框中选中左边的“Java Build Path”，选中对话框右边的第三项“Libraries”页面，点击“Add External Jars...”按钮，将永中Office 按装目录下的“Yozo\_Office.jar”添加进编译环境。这样就可以应用永中Office 的开发包对永中Office 进行二次开发了。

5、在Eclipse 的“Package Explorer”里，选中“example”节点，选择菜单“File”->“New”->“Class”弹出“New Java Class”对话框，填写类名称，假设为“MyTest”，点击完成按钮，这样就在项目“example”中建立了一个名为 “MyTest”的新类。

## 第二章 二次开发对象层次结构

API是Application Programming Interface（应用程序编程接口）的缩写，即一个主应用程序的开发者提供给其他开发者使用的服务和功能。这意味着API允许一些外部程序访问提供API的程序，即另外的程序通过主应用程序的API而使用某一个子程序的功能。许多大的应用程序都提供了API供二次开发者使用，如各种程序开发语言、各种操作系统及办公软件等。

API不仅仅可以被外部应用程序使用，许多大型应用程序也使用API 以便于自身程序的一部分与另一部分联系。如果一套服务或帮助功能——一个大应用程序的内核已被建立，那么，一个做完了的API就可以使开发应用程序的剩余部分变得十分轻松；此外，将其中一部分常用而且稳定的API开放给用户，用户在此基础上可以根据自己的需要开发出更加强大和个性化的功能，产品功能在适用过程中得到扩展，与用户的交互性也因此而更强。

基于跨平台操作的特性，永中Office充分考虑到用户对功能扩展的要求、不同需求对解决方案的要求及个性化等其它各方面的要求，在其内置的宏编辑器中提供了一整套实现内部功能的API供用户调用；此外，永中Office宏编辑器还支持导入JAVA的类库以使用JAVA 语言强大而丰富的类。

任何API都是基于对象的操作，在操作对象的基础上实现功能。永中Office二次开发的API即相当于JAVA语言中的方法，既可以是对某一对象单个属性或事件的控制和操作，也可以是对整个对象进行的操作，从而实现形式多样的功能。

#### 2.1单实例和多实例

永中Office二次开发体系支持单实例和多实例两种方式。单实例方式通常在永中Office的宏编辑器中使用，多实例方式通常在外部调用永中Office时使用。单实例和多实例在代码中的差别体现在创建永中Office实例时，下面是示例代码：

单实例示例：

//创建永中Office桌面应用单实例示例

Application.getApplicationInstance()

多实例示例：

**样例2.1.1**

//创建永中Office桌面应用多实例

//永中Office初始化参数

AppParameters appParams = AppParameters.getInstance();

//创建永中Office实例

ApplicationFactory.createInstance(appParams);

单实例和多实例除创建永中Office实例外，在具体功能性API调用方面没有差别。更多单实例和多实例的操作见[C3.1](#C31)。

**注意：**本手册中为方便描述，除明确说明外，其它代码均以多实例方式描述。

#### 2.2二次开发对象层次结构

永中Office API的对象体系是基于一种继承的结构，要访问一个对象，必须清楚该对象在整个对象体系中的位置，通过JAVA的对象运算符“.”从包含该对象的最外层开始，由外及内逐层找到其子对象，一直到所要访问的对象。

如要访问文件名为“文件1.eio”的电子表格文档中“工作表1”的“A1”单元格，需按如下格式书写：

Application.getWorkbooks().getWorkbook("C:/文件1.eio ").getWorksheet("工作表1").getCell("A1");

为了使代码简洁，可以使用访问对象的简化格式。如果某对象的父对象是当前对象，那么就可以直接对其子对象进行访问，例如以下是一个先打开一个文档然后对其子对象进行操作的代码示例：

**样例2.1.2**

//设置指定单元格内容

//新建电子表格应用

Workbook workbook =

app.getWorkbooks().addWorkbook(GlobalConstants.SPREADSHEET);

//获得电子表格应用文件中的工作表1

Worksheet worksheet = workbook.getWorksheets().getWorksheet("工作表1");

//在当前工作表中选择单元格“A1”

Cell a1 = worksheet.getCell("A1");

//在A1单元格中设值

a1.setValue("apple");

//在当前工作表中选择单元格“A2”

ActiveWorksheet.select("A2");

//当前单元格设值

ActiveCell.setValue("Lenny");

在永中Office的宏中，把所有具有相同属性和方法的，在对象体系中位于同一层次的对象划归为一个对象类，例如，在电子表格中，工作表的每个单元格都是一个Cell对象，这些Cell对象共同构成了Cell对象类。

永中Office宏编辑器提供的所有对象层次的顶部是Application，所有这些对象的访问都必须从Application 开始。以下是永中Office宏编辑器常用对象的层次结构图。

**永中office二次开发对象层次图**

集合对象

单个对象

Windows

Window

窗体集合

窗体

Application

Options

永中Office单实例

应用设置

CommandBar

CommandBars

IApplication

永中Office实例

工具栏

工具栏集合

Workbook

Workbooks

ApplicationFactiroy

永中Office多实例管理器

集成文件

集成文件集合

永中Office多实例

PDFAttribute

PDF文件属性

Versions

版本管理

Charts

Chart

图表

图表集合

Worksheets

Worksheet

工作表

电子表格工作表集合

Documents

Document

Workbook

集成文件

文字处理文档

文字处理文档集合

Presentations

Presentation

简报

简报制作文档集合

Tables

Table

表格

表格集合

FileProperties

文件属性

Shape

Shapes

自选图形

自选图形集合

**电子表格对象层次图**

集合对象

单个对象

Characters

字符属性

FontAttribute

Cell

字体属性

单元格

FillAttribute

列操作集

填充属性

Columns

FormatCellAttribute

格式属性

控件管理器

ComponentManager

BorderAttribute

边框属性

Names

名称管理器

ConditionalFormating

条件格式

Print

打印设置

ConditionalFormationgData

Range

条件格式设置

Rows

Worksheet

单元格区域

SortProperties

排序设置

工作表

行操作集

TextSplittingProperties

文本分列设置

ValidationProperties

SsPageSetup

数据校验设置

页面设置

Scenarios

SsComments

Scenario

方案集合

方案

Comment

批注集合

批注

Hyperlink

SsHyperlinks

超链接

超链接集合

SsShape

SsShapes

自选图形

自选图形集合

**文字处理二次开发对象层次图**

集合对象

单个对象

Document

文档

Section

节

Paragrahps

BaseText

段落集合

正文编辑区域

EndNoteOptions

FootNote

脚注属性（正文管理）

脚注编辑区域

FootNoteOptions

EndNote

尾注属性（正文管理）

尾注编辑区域

Header

Paragraph

页眉编辑区域

段落

Footer

页脚编辑区域

TextRange

文本区域对象

[Table](#Table)

Tables

表格集合

表格

[Style](#Style)

Styles

样式

样式集合（文档管理）

[Function](#Function)

具体功能点

FontAttribute

Style

字体属性（样式格式）

样式

ParagraphAttribute

段落属性（样式格式）

BorderAttribute

边框属性（样式格式）

PageBorderAttribute

ShadingAttribute

页面边框属性

WordBulletLevel

底纹属性

TabStops

Hyperlink

项目符号与编号属性（样式格式）

WaterAttribute

TabStop

制表位

制表位集合（样式格式）

GradientAttribute

超链接属性

PatternAttribute

渐进色填充效果属性

PictureAttribute

水印属性

图案填充效果属性

FillAttribute

填充属性

图片填充效果属性

TextureAttribute

纹理填充效果属性

Function

BookMarks

替换对象的格式

查找匹配的制表位集合

查找替换

当前选取的多个自选图形

WpShapeRange

Replacement

FindTabStops

FindReplace

BookMark

书签

书签集合（文档管理）

具体功能点

TrackChanges

TrackChange

修订

Field

修订集合（文档管理）

Fields

域对象

域对象集合

Comment

WpComments

批注

批注集合（文档管理）

PenMarkManager

手写批注（文档管理）

PageNumberAttribute

页码属性

Column

分栏（章节管理）

WpPageSetup

页眉设置（章节管理）

Index

索引

IndexEntry

索引项

TableOfContent

目录

Barcode

条码

BarcodeInfo

条码信息属性

FindReplace

FindTabStops

Replacement

WpShapeRange

自选图形管理器

（文档管理）

某个自选图形

当前选取的多个自选图形

查找替换

查找匹配的制表位集合

替换对象的格式

FindReplace

FindTabStops

Replacement

WpShapeRange

自选图形管理器

（文档管理）

某个自选图形

当前选取的多个自选图形

查找替换

查找匹配的制表位集合

替换对象的格式

MailingLabel

标签

MailMerge

邮件合并

WordCount

字数统计（文档管理）

WpShape

WpShapes

自选图形

自选图形管理器（文档管理）

Replacement

FindReplace

替换对象的格式

查找替换

FindTabStops

查找匹配的制表位集合

TableAttribute

Table

表格

表格属性

Rows

Row

表格行

表格行集合

Cells

单元格集合

Columns

Column

表格列

表格列集合

Cell

单元格

Paragraphs

CellAttribute

单元格属性

**简报制作二次开发对象层次图**

集合对象

单个对象

Slide

Presentation

Slides

幻灯片

幻灯片集合

简报

**自选图形对象层次图**

集合对象

单个对象

ShapeText

图形对象文本设置

GradientAttribute

渐进色属性

ConnectFormart

连接线设置

PatternAttribute

FillAttribute

Shape

Shapes

图案属性

自选图形

自选图形集合

ThreeDFormart

填充属性

PictureAttribute

图片属性

PictureFormart

三维格式

TextureAttribute

纹理属性

图片格式

PatternFormart

LineFormart

线型图案格式

线型对象格式

## 第三章 总控

本章将为您介绍永中Office总控部分的宏及二次开发。

本章内容共分为以下6节：

3.1 Application类和ApplicationFactory 类

3.2 文件管理器（Workbooks）

3.3 文件（Workbook）

3.4 打印（Print)

3.5 对话框（Dialogs)

3.6 菜单和工具条 (Commands)

#### 3.1 Application类和ApplicationFactory 类

Application类是用作为单实例的API调用的总入口，而ApplicationFactory类是用作为多实例的API调用的总入口。在多实例情况下，从ApplicationFactory 总入口处可以获得相当于Application类的接口，即IApplication接口。

下面是在多实例情况下的一个应用示例的代码：

**样例3.1.1**

//新建一个永中Office实例，且主窗体不显示，获得打印机并打印工作表

//在永中Office启动之前需要设置的参数AppParameters params = AppParameters.getInstance();

//启动参数设置隐藏永中Office主窗口params.hideMainFrame();

//通过ApplicationFactory 类获得 IApplication 接口IApplication iapp = ApplicationFactory.createInstance(params);

//由IApplication 接口获得一个添加的Workbook对象，Workbook book =

iapp.getWorkbooks().addWorkbook(GlobalConstants.SPREADSHEET);

//获得当前工作表Worksheet sheet = book.getWorksheets().getActiveWorksheet();

//在工作表的一个区域"B3:D6"设值sheet.getRange("B3:D6").setValue("abc");

//获得工作表的打印机对象Print print = sheet.getPrint();

//设置打印机的等待打印对话框为FALSE，即隐藏等待打印对话框print.setWaitingDialogVisible(false);

//关闭永中Office实例

iapp.exitSystem();

上述示例在单实例情况下的代码如下：

**样例3.1.2**

//样例：启动永中单Office实例，且主窗体不显示，获得打印机并答应工作表

//启动永中office

Application.getApplicationInstance();

Application.setMainFrameVisible(false);

//由Application 类获得一个添加的Workbook对象，

Workbook book =

Application.getWorkbooks().addWorkbook(

GlobalConstants.SPREADSHEET);

//获得当前工作表

Worksheet sheet = book.getWorksheets().getActiveWorksheet();

//在工作表的一个区域"B3:D6"设值

sheet.getRange("B3:D6").setValue("abc");

//获得工作表的打印机对象

Print print = sheet.getPrint();

//设置打印机的等待打印对话框为false，即隐藏等待打印对话框

print.setWaitingDialogVisible(false);

//打印工作表

print.printOut();

//退出永中Office

Application.exitSystem();

#### 3.2 Application类和IApplication类

Application类是一个静态类，类中的API均可直接调用；而IApplication类需要通过ApplicationFactory获得。Application类与单实例的永中Office应用实例对应，IApplication类与多实例的永中Office应用实例对应。Application类和IApplication类在API方面的主要区别在于Application类中包含了初始化永中Office实例（单实例）的API，而在永中Office内部功能API方面保持一致。

由于Application类和IApplication类在永中Office内部功能API方面保持一致，下面将以Application类为代表，详细说明其功能。

**注意**：手册中的Java示例大多用IApplication类做说明，如需在单实例情况下运行，请修改入口部分。

IApplication类在永中Office内部功能方面定义了对整个永中Office应用程序进行操作、控制、管理的方法。通过它可以获得文件管理器，窗口管理器，工具条管理器等等。该类的方法都是静态方法，所以无需生成该类对象而直接通过类来调用其中方法。

先看以下几个关于Application类和IApplication类的代码：

//新建JFrame和永中office单实例，并把这实例放入JFrame容器中

JFrame frame = new JFrame();

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

//启动永中office

Application.createInstance(frame.getContentPane());

frame.add(Application.getInstance());

frame.setSize(800, 600);

frame.setVisible(true);

与之对应的多实例代码

//新建JFrame和永中office实例，并把这实例放入JFrame容器中

JFrame frame = new JFrame();

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

//永中Office初始参数

AppParameters appParam = AppParameters.getInstance();

//设置承载永中Office的容器

appParam.setContainer(frame.getContentPane());

//创建永中Office

ApplicationFactory.createInstance(appParam);

frame.setSize(800, 600);

frame.setVisible(true);

//在applet中启动永中office单实例

public class SCreateOfficeInApplet extends JApplet

{

public void start()

{

//启动永中office

Application.createInstance(getContentPane());

//将永中Office添加到Applet中

getContentPane().add("Center", Application.getInstance());

}

}

//在applet中启动永中office实例

public class MCreateOfficeInApplet extends JApplet

{

public void start()

{

//永中Office初始参数

AppParameters appParams = AppParameters.getInstance();

//设置承载永中Office的容器

appParams.setContainer(getContentPane());

//创建永中Office

ApplicationFactory.createInstance(appParams);

}

}

下面列举了Application中启动永中office的相关方法：

//在JApplet或者JFrame中调用，新建一个永中Office实例。

public static void createInstance(Component component)

//在JApplet程序中调用,返回一个永中Office的实例。

public static Container getInstance()

//在Application程序中调用，返回一个永中Office的实例。

public static Container getApplicationInstance()

通过Application或IApplication可以获取Workbooks（文件管理器）、CommandBars（工具条管理器）、Windows（永中Office窗口对象）等，如同类层次图显示的那样，所有的二次开发API的调用都要经过Application或IApplication入口。Application或IApplication类也提供了一些获取永中Office应用程序相关信息的方法，如获取版本号，默认存盘路径，安装路径等。

请看下面两个样例：

**样例3.2.3**

//启动永中office应用程序并新建一电子表格文件

//永中Office初始参数

AppParameters appParam = AppParameters.getInstance();

//创建永中Office实例

IApplication iApp = ApplicationFactory.createInstance(appParam);

//获取文件管理器

Workbooks books = iApp.getWorkbooks();

//新建一电子表格文件

books.addWorkbook(GlobalConstants.SPREADSHEET);

**样例3.2.4**

//获得永中Office安装路径

//永中Office初始参数

AppParameters appParam = AppParameters.getInstance();

//创建永中Office实例

IApplication iApp = ApplicationFactory.createInstance(appParam);

//永中Office安装路径

String installPath =

iApp.getSystemPath(GlobalConstants.TPYE\_EIO\_INSTALL\_PATH);

下面列举了Application和IApplication中获取一些对象的相关方法：

//获取永中Office文件管理器

public static Workbooks getWorkbooks()

//获得菜单和工具条的管理器

public static CommandBars getCommandBars()

//返回永中Office软件的版本号

public static String getVersion()

//获取永中Office的安装路径

public static String getSystemPath()

//获取永中Office文件的默认存盘文件夹的路径

public static String getFileSavePath()

Application和IApplication类还提供了一些设置永中office界面的方法，除了通过得到工具条管理器来添加删除工具条外，还有如显示或隐藏任务面板，设置UI属性等，通过Application和IApplication类还能设置永中office中的某些菜单项是否可用等。

**样例3.2.5**

//启动永中office应用程序，隐藏文件菜单

//永中Office初始参数

AppParameters appParam = AppParameters.getInstance();

//设置不保存自定义动作

appParam.disableCustomizeSave();

//创建永中Office实例

IApplication iApp = ApplicationFactory.createInstance(appParam);

//获得文件菜单

CommandBar fileMenu = iApp.getCommandBars().getCommandBar("文件");

//隐藏文件菜单

fileMenu.setVisible(false);

以下列举了Application中其他一些设置office界面等的常用方法：

//显示或隐藏任务面板

public static void setTaskPanelVisible(int paneID, boolean show)

设置永中Office 的UI属性

public static void setUIStyle(int flags)

//设置是否显示编辑栏

public static void setFormulaBarVisible(boolean show)

//设置是否显示状态栏

public static void setStatusBarVisible(boolean show)

//设置永中Office菜单项可用或不可用

public static void setActionEnable(int index, boolean enabled)

//定位到某个应用

public static void gotoProduct(final int type)

相对应于以上这些设置方法，还有相应的获取属性或者状态的方法，如：

//通过面板ID判断面板是否显示

public static boolean isTaskPaneVisible(int paneID)

//是否显示编辑栏

public static boolean isFormulaBarVisible()

//是否显示状态栏

public static boolean isStatusBarVisible()

//判定某个菜单项是否有效

public static boolean isActionEnable(int index)

Application（IApplication）类作为永中Office应用示例的总入口，功能较多，API的详细功能描述请参照API说明。

以下是一个完整的代码示例

**样例3.2.6**

import application.AppParameters;

import application.ApplicationFactory;

import application.IApplication;

public class MyOffice

{

private IApplication iApp; // 永中Office实例

public MyOffice()

{

// 创建永中Office实例

iApp = ApplicationFactory.createInstance(AppParameters.getInstance());

// 添加文字处理应用

iApp.getWorkbooks().addWorkbook(GlobalConstants.WORDPROCESSOR);

}

public void setVisible()

{

// 设置不显示状态栏

iApp.setStatusBarVisible(false);

// 设置不显示编辑栏

iApp.setFormulaBarVisible(GlobalConstants.FORMULABAR\_INVISIBLE);

// 设置隐藏导航面板

iApp.setTaskPanelVisible(GlobalConstants.CURRENT\_PANE, false);

// 设置文件打印菜单项不可用

iApp.setActionEnable(ActionNameConstants.FILE\_PRINT, false);

}

// 在应用中添加一艺术字"Hello Application"

public void showHello()

{

// 获得自选图形集合

Shapes shapes = iApp.getWorkbooks().getActiveWorkbook()

.getActiveShapes();

// 添加艺术字

shapes.addWordArt(1, "Hello Application", "宋体", 1, 24, 250, 180, 400,

320);

}

public static void main(String[] args)

{

MyOffice myOffice = new MyOffice();

myOffice.setVisible();

myOffice.showHello();

}

}

#### 3.3 文件管理器（Workbooks）

application.Workbooks类对应于文件管理器，通过它可以添加、获取、保存文件等，因此，对文件进行的二次开发一般都会通过Workbooks这个类，Workbooks还能进行文件和byte数组的转换，这种方式很好的支持了文件在数据库的上传和下载。

先看几个对文件进行操作的代码

**样例3.3.1**

//添加一个文字处理文件

// 创建永中Office实例

IApplication iApp =

ApplicationFactory.createInstance(AppParameters.getInstance());

// 获得文件管理器

Workbooks wbs = iApp.getWorkbooks();

// 添加文字处理应用

wbs.addWorkbook(GlobalConstants.WORDPROCESSOR);

**样例3.3.2**

//打开一个文件

// 创建永中Office实例

IApplication iApp =

ApplicationFactory.createInstance(AppParameters.getInstance());

// 获取文件管理器

Workbooks books = iApp.getWorkbooks();

// 通过管理器打开D盘下test.eio文件

books.openWorkbook("D:/test.eio");

// 在控制台输出文件大小

以下列举了在Workbooks类中添加，获取，保存，关闭文件的常用方法：

//新建一个文件

public Workbook addWorkbook(int type)

//新建一个自定义名字的文件

public Workbook addWorkbook(String bookName, int type)

//打开一个文件

public Workbook openWorkbook(final String fileName)

//获取当前活动文件

public Workbook getActiveWorkbook()

//根据文件名称获取文件对象

public Workbook getWorkbook(String bookName)

//关闭永中Office中的所有文件

public void closeAll()

//保存永中Office中的所有文件

public void saveAll()

Workbooks对象中包涵了文件数据和byte数组转换的API，先看下例：

**样例3.3.2**

// 把当前文件复制成pdf类型的byte数组，然后利用其创建一pdf文件

byte[] bytes = null;

// 创建永中Office实例

IApplication iApp =

ApplicationFactory.createInstance(AppParameters.getInstance());

// 获取文件管理器

Workbooks books = iApp.getWorkbooks();

// 获取当前文档

Workbook book = books.getActiveWorkbook();

try

{

// 把此文件复制成pdf类型的byte数组

bytes = books.getWorkbookAsByteArray(book, FileConstants.TYPE\_PDF);

// 创建文件并写入数据，最后关闭它

File file = new File("D:/test.pdf");

FileOutputStream fos = new FileOutputStream(file);

fos.write(bytes);

fos.flush();

fos.close();

}

catch (IOException e)

{

e.printStackTrace();

}

以下列举了在Workbooks类中关于复制成byte数组的几个常用方法：

//读取一个永中Office文件,在内存中复制,将复制的永中Office文件以byte数组返回

public byte[] getWorkbookAsByteArray(Workbook book) throws IOException

//读取一个永中Office文件,在内存中复制,将复制的永中Office文件以指定类型的byte数组返回

public byte[] getWorkbookAsByteArray(Workbook book, int type) throws IOException

//从一个byte数组创建一个永中Office文件

public Workbook createWorkbookFromByteArray(byte[] data, String bookName) throws IOException

Workbooks类中提供了生成、管理文件的方法，最后我们提供一个访问数据库的示例。

**样例3.3.3**

import application.AppParameters;

import application.ApplicationFactory;

import application.IApplication;

import application.Workbooks;

public class SaveFileToDB

{

private Connection con;

private PreparedStatement stmt;

private String url;

private String user;

private String password;

private String databaseName;

private static SaveFileToDB dataBaseCon;

public static SaveFileToDB getInstance() throws Exception

{

if(dataBaseCon == null)

{

return new SaveFileToDB();

}

return dataBaseCon;

}

private SaveFileToDB() throws Exception

{

user = "sa";

password = "";

databaseName = "databaseName";

Class.forName(

getConnection();

}

public void getConnection() throws SQLException

{

if(con == null || con.isClosed())

{

con = DriverManager.getConnection(url, user, password);

}

}

/\*\*

\* 关闭数据库连接。

\* @throws SQLException

\*/

public void close() throws SQLException

{

con.close();

}

/\*\*

\* 打开指定路径的文件并将其保存到数据库。

\* @param path 文件的指定路径

\* @param name 保存的文件名称

\*/

public void upload(String path, String name) throws IOException, SQLException

{

String day = getDate();

try

{

// 创建永中Office实例

IApplication iApp =

ApplicationFactory.createInstance(

AppParameters.getInstance());

// 获取文件管理器

Workbooks books = iApp.getWorkbooks();

//打开文档

books.openWorkbook(path);

// 获取当前文档

Workbook book = books.getActiveWorkbook();

// 获得当前文档的二进制流

byte[] bytes = books.getWorkbookAsByteArray(book);

stmt = con.prepareStatement("INSERT INTO " + databaseName +

" (name,data,upday) VALUES (,,)");

stmt.setString(1, name);

stmt.setBytes(2, bytes);

stmt.setString(3, day);

stmt.executeUpdate();

}

finally

{

try

{

stmt.close();

}

catch(SQLException e)

{

}

}

}

/\*\*

\* 获取时间

\* @return

\*/

private String getDate()

{

long mills = System.currentTimeMillis();

Date date = new Date(mills);

String time = DateFormat.getDateInstance().format(date);

return time;

}

public static void main(String[] args) throws Exception

{

SaveFileToDB dbcon = SaveFileToDB.getInstance();

dbcon.upload("D:/1.eio", "testupload");

dbcon.close();

}

}

#### 3.4 文件（Workbook）

先看下例：

**样例3.4.1**

// 在当前文件中添加一新文字处理应用

// 创建永中Office实例

IApplication iApp =

ApplicationFactory.createInstance(AppParameters.getInstance());

// 获得文件管理器

Workbooks wbs = iApp.getWorkbooks();

// 获取打开的文件

Workbook book = wbs.openWorkbook("D:/test.doc");

// 获取文字处理管理器

Documents docs = book.getDocuments();

// 添加一新文字处理应用

docs.addDocument();

永中office现了电子表格，文字处理，简报制作三大应用。因此从Workbook可以得到这三个应用的管理器，进而可以添加，获取这些应用等等。

以下列举了获取三大应用管理器的方法：

// 返回电子表格管理器对象。

public Worksheets getWorksheets()

// 返回文字处理管理器对象。

public Documents getDocuments()

// 返回简报制作管理器对象。

public Presentations getPresentations()

**样例3.4.2**

// 获取当前应用图形管理器，并添加一矩形

// 创建永中Office实例

IApplication iApp =

ApplicationFactory.createInstance(AppParameters.getInstance());

// 获得文件管理器

Workbooks wbs = iApp.getWorkbooks();

// 获得打开的文件

Workbook book = wbs.openWorkbook("D:/test.doc");

// 获取当前应用图形管理器

Shapes shapes = book.getActiveShapes();

// 在当前应用添加一矩形

shapes.addShape(ShapeConstants.SHAPE\_RECTANGLE, 120, 130, 220, 250);

通过Workbook可以直接获取当前应用的自选图形管理器，这与先获取当前应用，再通过当前应用获取自选图形管理器的方法相类似。通过Workbook还可以获取图表管理器。

以下列举了获取自选图形管理器和图表管理器的两个方法：

//获取当前应用的自选图形管理器。

public Shapes getActiveShapes()

//返回图表的管理器。

public Charts getCharts()

**样例3.4.3**

// 获取当前文件的名称，大小

// 创建永中Office实例

IApplication iApp =

ApplicationFactory.createInstance(AppParameters.getInstance());

// 获得文件管理器

Workbooks wbs = iApp.getWorkbooks();

// 获取打开的文件

Workbook book = wbs.openWorkbook("D:/test.doc");

// 获取文件名称

String bookName = book.getName();

// 获取文件属性对象

FileProperties pro = book.getFileProperties();

// 获取文件大小

String size = pro.getFileSize();

iApp.getDialogs().showMessageDialog("文件名称：" + bookName +

" 文件大小：" + size);

Workbook提供了获取文件各类属性的方法，比如名称，文件属性对象，文件自定义属性对象，文件版本管理对象等等。通过这些方法你能得到文件的完整信息。

以下列举了获取文件各个属性的方法：

// 获得Workbook的名字，不含绝对路径。

public String getName()

// 获得Workbook的名称，名称包含其绝对路径。

public String getAbsolutePath()

// 获取当前文件属性。

public FileProperties getFileProperties()

// 获取当前文件自定义属性对象。

public CustomFileProperties getCustomFileProperties()

// 得到文件的版本管理对象。

public Versions getVersions()

**样例3.4.3**

// 在当前应用添加一艺术字，然后保存并关闭当前文件

// 创建永中Office实例

IApplication iApp =

ApplicationFactory.createInstance(AppParameters.getInstance());

// 获得文件管理器

Workbooks wbs = iApp.getWorkbooks();

// 获取打开的文件

Workbook book = wbs.openWorkbook("D:/test.doc");

// 获得当前文档的自选图形管理器

Shapes shapes = book.getActiveShapes();

// 添加艺术字

shapes.addWordArt(1, "艺术字", "宋体", 1, 12, 250, 180, 400, 320);

// 保存文件

book.save();

// 关闭文件

book.close();

Workbook类包含了保存，另存为，关闭文件，添加，移除文件监听器等方法。列举如下：

// 向Workbook添加监听器

// 移除向Workbook添加的监听器

// 保存当前文件

public void save()

// 把当前文件另存为别的文件名称

public void saveAs(String name)

// 关闭当前文件

public void close()

Workbook类提供了针对单个文件的操作方法，因为只有通过它才能获取文件中各个应用的管理器，所以它也是通往各个应用的必经通道。

下面提供一个完整的示例：

**样例3.4.3**

import application.AppParameters;

import application.ApplicationFactory;

import application.IApplication;

import application.Workbooks;

/\*\*

\* 新建电子表格文档，插入自选图形，保存文档并关闭

\*/

public class MyWorkbook implements WorkbookListener

{

// 永中Office实例

private IApplication iApp;

// 文件

private Workbook book;

public MyWorkbook()

{

// 创建永中Office实例

iApp = ApplicationFactory.createInstance(AppParameters.getInstance());

// 获得文件管理器

Workbooks wbs = iApp.getWorkbooks();

// 获得新建的电子表格文件

book = wbs.addWorkbook(GlobalConstants.SPREADSHEET);

// 添加文件监听器

book.addWorkbookListener(this);

}

/\*\*

\* WorkbookListener事件

\*/

public void bookChanged(WorkbookEvent e)

{

if (e.getType() == WorkbookEvent.AFTER\_CLOSE)

{

}

}

public void bookExample()

{

// 获取电子表格管理器

Worksheets sheets = book.getWorksheets();

// 添加工作表

sheets.addWorksheet();

// 获取当前应用自选图形管理器

Shapes shapes = book.getActiveShapes();

// 在当前应用添加云形标注

shapes.addShape(ShapeConstants.SHAPE\_CLOUD\_CALLOUT,

120, 130, 220, 250);

// 保存文件

book.saveAs("D:/test.eio");

// 关闭文件

book.close();

// 去除文件监听器

book.removeWorkbookListener(this);

}

public void exit()

{

// 退出永中office

iApp.exitSystem();

}

public static void main(String[] args)

{

MyWorkbook myWorkbook = new MyWorkbook();

myWorkbook.bookExample();

myWorkbook.exit();

}

}

#### 3.5 打印（Print)

类Print对应了打印功能，提供了设置打印对话框中打印属性，范围等的一系列方法，比如设置打印机名称，设置双面打印，反片打印，设置打印区域等等。Print对象可以通过Worksheet（电子表格类），Document（文字处理类），Presentation（简报制作类）中的getPrint()方法获取，这三个类后面也将陆续谈到。

先请看以下两个关于打印属性的代码

**样例3.5.1**

//手动双面打印当前电子表格

//创建永中Office实例

IApplication iApp =

ApplicationFactory.createInstance(AppParameters.getInstance());

//获得文件管理器

Workbook book = iApp.getWorkbooks().openWorkbook("D:/test.eio");

// 获得当前工作簿

Worksheet sheet = book.getWorksheets().getActiveWorksheet();

// 获得打印对象

Print print = sheet.getPrint();

// 设置双面打印

print.setDuplex(true);

// 打印

print.print();

**样例3.5.2**

//设置反片打印并打印当前文字处理文档

// 创建永中Office实例

IApplication iApp =

ApplicationFactory.createInstance(AppParameters.getInstance());

// 获得文件管理器

Workbook book = iApp.getWorkbooks().openWorkbook("D:/test.eio");

// 获得当前文字处理文档

Document doc = book.getDocuments().getActiveDocument();

// 获得打印对象

Print print = doc.getPrint();

// 设置反片打印

print.setMirror(true);

// 打印

print.print();

类Print中包括各个应用中的打印属性的方法，方法列表如下：

// 设置用户选择的打印机名称

public void setPrinterName(String printName)

// 设置手动双面打印的标志

public void setDuplex(boolean duplex)

// 设置反片打印的标志

public void setMirror(boolean mirror)

// 设置信封选项是否选中的标志

public void setEnvelop(boolean envelop)

// 设置用户打印到文件的名称

public void setPrintToFileName(String name)

**样例3.5.3**

// 打印电子表格中区域为"A1:B3"部分

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

Print print = sheet.getPrint();

print.setRange("A1:B3");print.print();

**样例3.5.4**

// 打印当前文字处理文档中的2到3页

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Workbook book = Application.getWorkbooks().getActiveWorkbook();Document doc = book.getDocuments().getActiveDocument();Print print = doc.getPrint();print.setPageNumbers("2-3");print.print();

示例5 ：设置打印3份简报制作中自定义播放幻灯片

Workbook book = Application.getWorkbooks().getActiveWorkbook();Presentation present = book.getPresentations().getActivePresentation();Print print = present.getPrint();print.setCustomshow(true);print.setCustomshow("abc");

print.setCopies(3);print.print();

就如同在打印对话框中的那样，在电子表格中Print类能够打印指定页，指定区域等。而在文字处理中，能够选择打印指定页，奇偶数页等等。在简报制作中，能够指定打印自定义播放幻灯片等等。

以下列举了Print类中打印范围和打印内容的方法：

public void setPrintAll()

设置打印全部

public void chooseCurrentPage()

设置选择打印当前页

public void setCopies(int copies)

设置用户定义的副本份数

public void setPrintPages(int index)

设置要打印的文档页面范围

public void setPrintItem(int index)

设置打印内容

public void setCustomshow(boolean isshow)

设置幻灯片对话框中是否是打印自定义播放的标志

public void setCustomshow(String show)

设置幻灯片对话框中打印自定义播放内容

public void setHandoutPages(String pages)

设置幻灯片对话框中讲义的页数

public void setSelectedRange(boolean selectedRange)

设置电子表格选定区域的标志

public void setRange(String rangeAddress)

设置打印电子表格的区域

public void setEntireBook(boolean entireBook)

设置电子表格选中整个工作簿的标志

下面提供一个完整的打印代码示例，设置电子表格打印属性和范围并进行打印。

//样例3.4

import application.Application;

public class MyPrint

{

private Workbook book;

public MyPrint()

{

//启动永中office应用程序

Application.getApplicationInstance();

book = Application.getWorkbooks().addWorkbook(GlobalConstants.SPREADSHEET);

}

public void printExample()

{

//获取当前电子表格并对其中一块区域进行设置

Worksheet sheet = book.getWorksheets().getActiveWorksheet();

Range range = sheet.getRange("A1:Q1000");

range.setValue("MyPrint");

//设置打印属性和范围，并进行打印

Print print = sheet.getPrint();

print.setRange("G5:M50");

print.setDuplex(true);

print.print();

//保存并关闭文件

book.saveAs("D:\\test.eio");

book.close();

}

public void exit()

{

//退出永中office

Application.exitSystem();

}

public static void main(String[] args)

{

MyPrint myPrint = new MyPrint();

myPrint.printExample();

myPrint.exit();

}

}

#### 3.6 对话框（Dialogs)

先看如下两个关于显示对话框的方法：

示例1 ：显示信息为"My Message"的信息对话框

Dialogs.showMessageDialog("MyMessage",OfficeDialogConstants.OK\_CANCEL);

示例2 ：显示一打开文件对话框

String[]aa = {"txt","jpg","jar"};

Dialogs.showOpenDialog(null,"Input title",aa);

以下列举了显示各类对话框的方法：

String title)

显示信息对话框

public static Color showColorDialog(Dialog dlgParent)

显示颜色对话框

public static FontAttribute showFontDialog(Frame parent)

显示字体对话框

public static String showSaveDialog(Frame parent, String title, String initfile,String extension)

显示保存对话框

public static String showOpenDialog(Frame parent, String title, String extension,String description)

显示打开对话框

public static String showInputDialog(Frame parent, String title, String prompt)

显示输入对话框

public static String showNewDialog(Frame parent, boolean modal)

显示新建对话框

Dialogs还能显示永中office指定对话框，见下例：

示例2 ：显示插入页码对话框

Dialogs.showOfficeDialog(OfficeDialogConstants.INSERT\_PAGE\_NUMBERS );

此方法即是：

public static void showOfficeDialog(int type)

显示指定的对话框

#### 3.7 菜单和工具条（CommandBar）

application.CommandBars类是永中Office应用程序中的菜单和工具条的管理器，包含添加，删除，获取菜单，右键菜单和工具条等操作。得到这个管理器对象可以通过Application类中方法getCommandBars()方法。

先看以下关于菜单的示例：

示例1 ：在当前Office应用程序中在文件菜单项中关闭菜单项下插入复制菜单项。

//获取菜单和工具条管理器

CommandBars bars = Application.getCommandBars();

//获取文件菜单

CommandBar bar = bars.getCommandBarByID(CommandBar.MENU, MenuIDConstants.FILE);

//添加复制菜单项

bar.addItemByID(MenuIDConstants.FILE\_CLOSE, MenuIDConstants.EDIT\_COPY);

ComamandBars类内含两个直接返回CommandBar的方法，分别通过名称和ID：

public CommandBar getCustomCommandBar(int type, String name)

根据自定义工具条的类型和名称获得工具条对象。

public CommandBar getCommandBarByID(int type, int ID)

根据ID获得工具条或菜单对象。

CommandBar类内含两个添加菜单项的方法：

public void addItemByID(int index, int item\_ID)

添加工具条或菜单的内容。

public void addItemByID(int productType, int index, boolean beforeFlag, int item\_ID)

添加工具条或菜单的内容。

示例2 ：在当前Office应用程序中添加一自定义工具条。

//获取菜单和工具条管理器

CommandBars bars = Application.getCommandBars();

//添加一自定义工具条

CommandBar bar = bars.addToolBar("MyToolBar");

//在工具条上面添加内容

bar.addItemByID(0, MenuIDConstants.EDIT\_COPY);

bar.addItemByID(1, MenuIDConstants.EDIT\_CUT);

bar.addItemByID(2, MenuIDConstants.EDIT\_PASTE);

示例3 ：在当前Office应用程序中添加两个自定义工具条，然后删除其中之一。

//获取菜单和工具条管理器

CommandBars bars = Application.getCommandBars();

//添加两个自定义工具条

CommandBar bar1 = bars.addToolBar("MyToolBar1");

CommandBar bar2 = bars.addToolBar("MyToolBar2");

//在工具条上删除第一个添加的自定义工具条

bars.delete(bar1);

ComamandBars类中的添加，删除工具条的方法如下：

public CommandBar addToolBar(String name)

新建工具条。

public void delete(CommandBar bar)

删除工具条。

public void deleteByID(int ID)

根据已知工具条ID删除工具条。

新建自定义工具条的ID是从2048开始的，当删除一个后，再添加进来的自定义工具条ID会覆盖掉前面所删除的ID。如新建三个自定义工具条，则ID为2048，2049，2050。然后删除第二个，则剩下为2048，2050。之后再添加一自定义工具条，其ID则为2049。

请看下面这个示例：

//示例4：

//获取菜单和工具条管理器

CommandBars bars = Application.getCommandBars();

//插入三个自定义工具条并显示其ID

CommandBar bar1 = bars.addToolBar("MyToolBar1");

CommandBar bar2 = bars.addToolBar("MyToolBar2");

CommandBar bar3 = bars.addToolBar("MyToolBar3");

//删除第二个自定义工具条

bars.delete(bar2);

//插入第四个自定义工具条并显示其ID

CommandBar bar4 = bars.addToolBar("MyToolBar4");

下面附一个完整的有关自定义菜单的样例

// 样例3.5：

//自定义工具条，包含2个按钮：“打开”和“关闭”

import application.AppParameters;

import application.ApplicationFactory;

import application.CommandBars;

import application.IApplication;

import application.KeyBindings;

public class TestComanndBar extends JFrame

{

IApplication iapp;

//action

TestAction action;

public void init()

{

init();

//initKeyBinding();

initCommandBar();

}

public void init()

{

AppParameters parameters = AppParameters.getInstance();

parameters.setContainer(getContentPane());

iapp = ApplicationFactory.createInstance(parameters);

getContentPane().add(iapp.getInstance());

Workbook book = iapp.getWorkbooks().addWorkbook(GlobalConstants.WORDPROCESSOR);

Document doc = book.getDocuments().getActiveDocument();

doc.getSelectionRange().insertTextAfter("this is a test!");

doc.activate();

action = new TestAction();

action.setApplication(iapp);

}

//设置快捷键

public void initKeyBinding()

{

KeyBindings bindings = iapp.getKeyBindings();

bindings.getKeyBinding(KeyEvent.VK\_N, KeyEvent.CTRL\_MASK).setEnabled(false);

bindings.getKeyBinding(KeyEvent.VK\_W, KeyEvent.CTRL\_MASK).setEnabled(false);

bindings.getKeyBinding(KeyEvent.VK\_O, KeyEvent.CTRL\_MASK).setEnabled(false);

bindings.getKeyBinding(KeyEvent.VK\_P, KeyEvent.CTRL\_MASK).setEnabled(false);

bindings.getKeyBinding(KeyEvent.VK\_G, KeyEvent.CTRL\_MASK).setEnabled(false);

}

public void initCommandBar()

{

try

{

iapp.applyForAllProduct(true);

initMenuBar();

initToolBar();

}

catch(Exception e)

{

e.printStackTrace();

}

}

private void initMenuBar()

{

CommandBars bars = iapp.getCommandBars();

//获取“文件”菜单

CommandBar bar = bars.getCommandBar("file");

CommandBarControls controls = bar.getControls();

//删除“打开”菜单项

controls.getControlByID(MenuIDConstants.FILE\_OPEN).delete();

//在最后添加“组合”菜单项

CommandBarControl control = controls.append(2, ControlConstants.CONTROL\_MENU, "组合");

//添加打开菜单项

CommandBarControl openMenuItem = control.getControls().append(Constants.MENU\_OPEN\_MENU\_ID,

ControlConstants.CONTROL\_BUTTON, "打开");

addOpenItem(openMenuItem);

//添加关闭菜单项

CommandBarControl closeMenuItem = control.getControls().append(

Constants.MENU\_CLOSE\_MENU\_ID, ControlConstants.CONTROL\_BUTTON, "关闭");

addCloseItem(closeMenuItem);

control.getControls().append(MenuIDConstants.EDIT\_COPY).setGroup(true);

control.getControls().append(MenuIDConstants.EDIT\_PASTE);

control.getControls().append(MenuIDConstants.EDIT\_CLEAR);

control.getControls().append(MenuIDConstants.FORMAT\_BORDERS).setGroup(true);

control.getControls().append(MenuIDConstants.FORMAT\_BACKGND);

}

private void initToolBar()

{

CommandBars bars = iapp.getCommandBars();

CommandBar testBar = bars.getCommandBar("test");

if (testBar != null)

{

bars.delete(testBar);

}

testBar = bars.addToolBar("test");

CommandBarControl openToolBarItem = testBar.getControls().append(

Constants.TOOLBAR\_OPEN\_MENU\_ID, ControlConstants.CONTROL\_BUTTON, "打开");

addOpenItem(openToolBarItem);

CommandBarControl closeToolBarItem = testBar.getControls().append(

Constants.TOOLBAR\_CLOSE\_MENU\_ID, ControlConstants.CONTROL\_BUTTON, "关闭");

addCloseItem(closeToolBarItem);

}

private void addOpenItem(CommandBarControl control)

{

//设置action路径

// control.setAction(actionID, "testcommandbar.TestAction");

int type = control.getCommandBar().getType();

if (type == CommandBar.MENU)

{

//设置action

control.setAction(Constants.MENU\_OPEN\_ACTION\_ID, action);

//设置快捷键

control.binding(KeyEvent.VK\_A, KeyEvent.CTRL\_MASK);

control.setOrder(Constants.MENU\_OPEN\_ORDER);

}

else if (type == CommandBar.TOOLBAR)

{

control.setAction(Constants.TOOLBAR\_OPEN\_ACTION\_ID, action);

control.binding(KeyEvent.VK\_B, KeyEvent.CTRL\_MASK);

control.setOrder(Constants.TOOLBAR\_OPEN\_ORDER);

}

}

private void addCloseItem(CommandBarControl control)

{

//设置action路径

//control.setAction(actionID, "testcommandbar.TestAction");

//设置快捷键

int type = control.getCommandBar().getType();

if (type == CommandBar.MENU)

{

//设置action

control.setAction(Constants.MENU\_CLOSE\_ACTION\_ID, action);

//设置快捷键

control.binding(KeyEvent.VK\_C, KeyEvent.CTRL\_MASK);

//设置order

control.setOrder(Constants.MENU\_CLOSE\_ORDER);

}

else if (type == CommandBar.TOOLBAR)

{

control.setAction(Constants.TOOLBAR\_CLOSE\_ACTION\_ID, action);

control.binding(KeyEvent.VK\_D, KeyEvent.CTRL\_MASK);

control.setOrder(Constants.TOOLBAR\_CLOSE\_ORDER);

}

}

public static void main(String[] args)

{

TestComanndBar frame = new TestComanndBar();

frame.setSize(400, 300);

frame.addWindowListener(new WindowAdapter()

{

public void windowClosing(WindowEvent e)

{

System.exit(0);

}

});

frame.setVisible(true);

frame.init();

}

}

import application.IApplication;

public class TestAction extends EIOAction

{

private IApplication iapp;

public void setApplication(IApplication iapp)

{

this.iapp = iapp;

}

protected void performAccelerator(int menuID)

{

actionPerformed(menuID);

}

public void openFile()

{

iapp.actionPerformed(ActionNameConstants.FILE\_OPEN, MenuIDConstants.FILE);

}

public void closeFile()

{

iapp.actionPerformed(ActionNameConstants.FILE\_CLOSE, MenuIDConstants.FILE);

}

private void actionPerformed(int menuID)

{

switch (menuID)

{

case Constants.MENU\_OPEN\_ORDER:

case Constants.TOOLBAR\_OPEN\_ORDER:

openFile();

break;

case Constants.MENU\_CLOSE\_ORDER:

case Constants.TOOLBAR\_CLOSE\_ORDER:

closeFile();

break;

default:

return;

}

}

public void actionPerformed(ActionEvent e)

{

Object source = e.getSource();

if (source != null)

{

actionPerformed((Integer)MenuUtilities.getMenuOrder(source));

}

}

}

## 第四章 电子表格

本章将为您介绍永中Office电子表格的宏及二次开发。

本章内容共分为以下6节：

* 1. 工作表管理器 (Worksheets)
  2. 工作表 (Worksheet)
  3. 单元格区域 (Range)
  4. 行对象 (Rows)
  5. 列对象 (Columns)
  6. 单元格 (Cell)

#### 4.1 工作表管理器（Worksheets）

在电子表格应用中，工作表管理器

请看以下几个关于工作表管理器操作的代码

示例1：获得当前工作簿中工作表2

//获得当前工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得Worksheets对象

Worksheets worksheets = workbook.getWorksheets();

//获得工作表2

Worksheet worksheet = worksheets.getWorksheet("工作表2");

//在工作表2的单元格（1，1）中设值

worksheet.setCellValue(1, 1, "apple");

示例2：在当前工作簿中添加一个名为结构的新工作表

//获得当前工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得Worksheets对象

Worksheets worksheets = workbook.getWorksheets();

//获得工作表2

Worksheet worksheet = worksheets.addWorksheet("结构");

以下列出Worksheets获得工作表的的一些API：

Worksheet getWorksheet(String name)

通过工作表名称取得相应的工作表。

Worksheet getWorksheet(int index)

通过工作表的索引号取得相应的工作表(注意：索引号从1开始)。

Worksheet getActiveWorksheet()

获得当前活动工作表。如果当前不是在电子表格应用中，则返回null。

Worksheets[] getAllWorksheets()

获得当前工作簿中所有的工作表。

在文件中插入一张新的工作表是工作表管理器的基本操作之一。以下列举了由工作表管理器进行新工作表插入的一些常用API：

Worksheet insertWorksheet(int index, String name)

在指定的工作表前插入新工作表（索引号从1开始）。

参数：

index 工作表索引

name 工作表名称（长度范围在1至31之间）

Worksheet insertWorksheet(int index)

在指定的工作表前插入新工作表（索引号从1开始）。

参数：index 工作表索引

除了以上在指定位置插入工作表的方法外，还有一种方法是直接在工作表的最后插入一张新的工作表，方法如下：

Worksheet addWorksheet()

在最后一张工作表后面添加一张新的工作表。

Worksheet addWorksheet(String name)

在最后一张工作表后面添加一张以name为工作表名的新的工作表。

参数：name 工作表名称（长度范围在1至31之间）

示例3：在当前文件中删除名为“工作表1”的工作表。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得Worksheets对象

Worksheets worksheets = workbook.getWorksheets();

//删除工作表1

worksheets.removeWorksheet("工作表1");

下面是工作表管理器中删除工作表的API:

void removeWorksheet(String sheetName)

在当前文件中删除工作表名称为sheetName的工作表。

void removeWorksheets(String[] sheetsName)

在当前文件中删除工作表名称为sheetsName中字符串的多个工作表。

工作表管理器还可以对指定的工作表进行重命名操作，其API如下：

boolean renameWorksheet(String oldSheetName, String newSheetName)

参数：

oldSheetName 原工作表名称

newSheetName 新工作表名称(名称长度范围在1至31之间)

代码示例：

//获得工作表管理器

Worksheets worksheets = Application.getWorkbooks().getActiveWorkbook().getWorksheets();

//将“工作表2”重命名为“新工作表”

worksheets.renameWorksheet("工作表2" ,"新工作表");

下面提供了一个完整的示例代码

import application.Application;

public class TestWorksheets

{

public TestWorksheets()

{

//开启永中office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.SPREADSHEET);

}

public void process()

{

//获得当前活动工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得工作表管理器

Worksheets worksheets = workbook.getWorksheets();

//在最后一张工作表后面插入一张名为last的工作表

Worksheet sheet = worksheets.addWorksheet("last");

//在工作表"last"的单元格(3,3)中设值

sheet.setCellValue(3, 3, "last");

//删除“工作表1”

worksheets.removeWorksheet("工作表1");

//将“工作表2”重命名为“first”

worksheets.renameWorksheet("工作表2", "first");

//在第三张工作表之前插入一张名为“third”的工作表

sheet = worksheets.insertWorksheet(3, "third");

//在工作表"third"的单元格(2,2)中设值

sheet.setCellValue(2, 2, "third");

}

public static void main(String args[])

{

TestWorksheets myTest= new TestWorksheets();

myTest.process();

}

}

#### 4.2 工作表（Worksheet)

Worksheet 是电子表格应用中一个很重要的类，通过它可以获取单元格区域、行、列、单元格等对象。该类还定义了对工作表对象进行操作的一些方法，如更改工作表名称，设值，定位到工作表某一区域，设置超链接，插入注释等。

先请看以下两个关于工作表操作的代码

示例1 ：在当前活动工作表中获得单元格区域 “B3:E5”

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet worksheet = sheets.getActiveWorksheet();

Range range = worksheets.getRange("B3:E5");

我们在先前的类层次图上可以了解到工作表中的类层次和结构。Worksheet 从 Worksheets 处获得。而从 Worksheet 可以获得 Range（单元格区域）、Cell（单元格）、Rows（行对象）、Columns（列对象）等工作表组成对象，而且象Range 等的工作表组成对象也只能从Worksheet处获得。

以下列举了在Worksheet 中获得工作表的组成对象的一些API

Cell getActiveCell()

获取工作表中的活动单元格对象。

Range getActiveRange()

获取工作表中的活动单元格区域对象。

Range getRange(String rangeAddress)

Range getRange(int startRow, int startCol, int endRow, int endCol)

获得单元格区域对象。

Rows getRows(int startRow, int endRow)

Rows getRows(String rowsAddress)

获取行对象。

Columns getColumns(int startColumn, int endColumn)

Columns getColumns(String columnAddress)

获取指定列对象。

Cell getCell(int row, int column)

Cell getCell(String cellAddress)

获取单元格对象。

示例 2 ：在当前活动工作表中设置数据和获取数据

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

sheet.setCellValue(2, 3, "apple");

String value = sheet.getCellValue(2, 3);

sheet.setCellValue(3, 3, "my" + value);

工作表的一个很重要的功能就是存储和显示数据。从某中意义上看，它就是一个小型的数据库。Worksheet 提供了非常丰富的API 来存储和显示数据。不仅有设置单个单元格数值的API，还提供对以数组形式的批量数据进行存储的API。特别是当数据来自于一个数据库时，利用Worksheet 的API 可以提供简洁的操作。以下仅列举几个API，其它API可查阅永中Office的API 文档。

void setCellValue(int row, int column, String value)

给行号和列号指定的单元格设置内容。

void setCellValue(int row, int column, int[] values, boolean horizontal)

给以指定的行号和列号为起点的单元格区域设置内容。

注意：此方法对数据不作解析，直接保存。 对于受保护单元格中的内容不作替换。

参数:

row 起始行号

column 起始列号

values 数据

horizontal 数据的排列顺序，true为横排，false为竖排

关于获取单元格的值，Worksheet 提供了两个API：

Object getCellValue(int row, int column)

String getCellString(int row, int column)

第一个API 返回的Object 对象。如果单元格的值是整型，则Object 对象是Integer 型，类似的，浮点型对应于Double,字符串对应于String 等。

第二个API 返回的是单元格的值对象的String对象。

Worksheet 还管理其它一些功能对象，象批注、超连接，打印、页面设置和自选图形等。

即要在工作表上做这些操作，需要先从Worksheet获得功能对象或功能对象的管理器。如：

SsComments getComments()

获得批注管理器。

代码示例：

//获得当前工作表

Worksheet worksheet = Application.getWorkbooks().getActiveWorkbook() .getWorksheets().getActiveWorksheet();//获得当前工作表的批注管理器对象SsComments comments = worksheet.getComments();//在单元格 “C2”中插入批注

Comment comment = comments.insertComment(2,3);

//在批注中添加内容comment.setText("我的批注");

SsHyperlinks getHyperlinks()

获得超链接管理器。

代码示例：

Worksheet sheet = Application.getWorkbooks().getActiveWorkbook() .getWorksheets().getActiveWorksheet();//获得超链接管理器

SsHyperlinks hyperlinks = sheet.getHyperlinks();

//新建一个超链接对象并设置属性

Hyperlink hyperlink = new Hyperlink();hyperlink.setHyperlinkType(HyperlinkConstants.THIS\_BINDER\_POSITION);hyperlink.setShowText("工作表2!A1");hyperlink.hyperlinkToCell("工作表2!A1");

//在当前工作表的单元格“E3”中插入该超链接hyperlinks.insertHyperlink(3,5,hyperlink);

Print getPrint()

获得打印机。

代码示例：

Worksheet sheet = Application.getWorkbooks().getActiveWorkbook() .getWorksheets().getActiveWorksheet();//获得打印机

Print print = sheet.getPrint();

//设置用户选择的打印机名称

print.setPrinterName("Adobe PDF");

//设置打印全部

print.setPrintAll();

//开始打印文档

print.print();

总之，Worksheet 是一个功能非常强大的类，相应的，它也提供了非常丰富的API以提供对永中Office 的二次开发。

下面提供一个完整的示例代码，表示更改当前工作表名称，并给单元格设置以数组形式表示的数据

//样例4.2

import application.Application;

public class TestWorksheet

{

public TestWorksheet()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.SPREADSHEET);

}

public void process()

{

//获得当前活动工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得当前活动工作表

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

//更改工作表名称

sheet.setName("my");

//给单元格“B1”设值

sheet.setCellValue(1, 2, "fruit");

String[] data = {"apple", "orange", "grape", "peach"};

//从单元格"B2"开始按行设值

sheet.setCellValue(2, 2, data, false);

}

public static void main(String[] args)

{

TestWorksheet myTest = new TestWorksheet();

myTest.process();

}

}

#### 4.3 单元格区域（Range）

在电子表格应用中,单元格区域对应于类：

Range是电子表格应用中一个非常重要的类。许多操作需要通过Range来实现。Range类不仅提供了包含单元格区域的设值和设置如字体等的单元格格式化操作，还有复制和粘贴单元格区域，在单元格区域内查找和替换数据等，而且，对于数据的操作，如排序和筛选数据等操作也由Range提供，故可以说，Range类功能非常强大。下面就让我们简要介绍Range类。

先请看以下几个关于单元格区域操作的示例代码：

示例1：在单元格区域“A3:D6”设值并设置红色字体

//获得当前工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得当前工作表

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

//获得单元格区域"B3:D6"

Range range = sheet.getRange("B3:D6");

//设值

range.setValue("abc");

//新建一个字体属性

FontAttribute font = new FontAttribute();

//设置字体的前景色

font.setForeground(255,0,0);

//给单元格区域设置字体属性

range.setFontAttribute(font);

以上示例代码用到了Range类以下一些API：

void setValue(String value)

给单元格区域设置字符串类型的数据内容。

void setFontAttribute(FontAttribute fa)

设置字体属性。

参数：fa 字体属性

此外，Range 还提供了对于填充、边框、单元格格式属性等的单元格区域格式化操作：

void setFormatCellAttribute(FormatCellAttribute attr)

设置单元格格式。

参数: attr 单元格格式

代码示例：将单元格区域"B3:E5"中的单元格格式设置为水平居中对齐

Worksheet worksheet = Application.getWorkbooks().getActiveWorkbook() .getWorksheets().getActiveWorksheet();//获得单元格区域 "B3:E5"

Range range = worksheet.getRange("B3:E5");

//新建一个单元格格式属性FormatCellAttribute fcAttribute = new FormatCellAttribute();

//单元格格式设置为水平居中对齐fcAttribute.setHorizontalIndex(CellFormatConstants.HOR\_MIDDLE);

//设置单元格格式属性range.setFormatCellAttribute(fcAttribute);

void setFillAttribute(FillAttribute fa)

设置填充属性。

参数: fa 填充属性

代码示例：给单元格区域"B3:E5"的设置填充属性

Worksheet worksheet = Application.getWorkbooks().getActiveWorkbook() .getWorksheets().getActiveWorksheet();//获得单元格区域

Range range = worksheet.getRange("B3:E5");

//新建一个填充属性FillAttribute fill = new FillAttribute();

//给单元格区域设置填充属性range.setFillAttribute(fill);

注意：Range 没有提供获取这些格式化属性的API，但是Cell提供了相应的API。如要获得某格式化属性，可通过单元格（Cell）来获得，具体可参见单元格（Cell）。

下面是一些关于数据操作方面的API：

对所选单元格区域中的数据进行排序。

参数：sortProperties 排序属性

代码示例：对单元格区域排序，设置排序主关键字索引值为2，并且按升序排列，包含标题行

Worksheet sheet = Application.getWorkbooks().getActiveWorkbook()

.getWorksheets().getActiveWorksheet();

//获得单元格区域

Range range = sheet.getRange("B4:E16");

range.fillRandom();

String[] heading = {"aa","bb","cc","dd"};

//设置单元格区域的行头

sheet.setCellValue(3,2,heading,true);

//获得带行头的单元格区域

range = sheet.getRange("B3:E16");

//新建一个排序属性

SortProperties sortProperties = new SortProperties();

//设置排序主关键字为单元格区域的第2列

sortProperties.setSortKey1(2);

//设置主关键字排序方式为按升序进行排序

sortProperties.setOrder1(true);

//设置排序包含标题行

sortProperties.setRowHeader(true);

//对单元格区域进行排序

range.sort(sortProperties);

int[] advancedFilter(String criteriaRange, String pasteRange, boolean isCopy, boolean remove)

高级筛选。(基于给定的条件区域从数据清单中筛选或复制数据)

参数：

criteriaRange 条件区域

pasteRange 粘贴区域

isCopy 布尔型，如为true，则把筛选结果拷贝到其他区域；否则，不把筛选结果拷贝到其他区域

remove 布尔型，如为true，要把重复的内容除去；否则，不把重复的内容除去

代码示例：对单元格区域"B2:E8"进行高级筛选。条件区域为"G2:H3""

Worksheet ws = Application.getWorkbooks().getActiveWorkbook().

getWorksheets().getActiveWorksheet();

String[] heading2= {"AA","BB","CC","DD"};

String[] items1 = {"panda","tiger","sheep","pet","dog","cat"};

int[] items2 = {4,2,6,3,7,6};

String[] items3 = {"Worksheet","Range","Rows","Column","Cell","Workbook"};

//设置单元格区域的行头

ws.setCellValue(2,2,heading2,true);

//设置单元格区域数据

ws.setCellValue(3,2,items1,false);

ws.setCellValue(3, 3, items2,false);

ws.setCellValue(3, 4, items3, false);

//单元格区域随机填充

ws.getRange("E3:E8").fillRandom();

//设置高级筛选的条件区域

String[] cateria = {"BB", "BB"};

String[] cateriaData = {">3", "<7"};

ws.setCellValue(2, 7, cateria, true);

ws.setCellValue(3, 7, cateriaData, true);

//获得要筛选的单元格区域

Range range = ws.getRange("B2:E8");

//对单元格区域按条件区域进行高级筛选，获得筛选结果的行号

int[] indexes = range.advancedFilter("G2:H3", "", false, false);

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

{

}

以下再列举一些Range在二次开发中用到的重要的API：

Cell[] getCells()

获得单元格区域的所有的单元格。

Cell[] find(String text)

查找。

参数：text 查找内容

返回：所有找到的单元格

void replaceAllWith(String findWhat, String replaceWith)

缺省全部替换。

参数：

findWhat 查找内容

replaceWith 替换内容

void merge()

合并单元格。

void mergeAcross()

按选取单元格区域的行合并单元格(只合并行不合并列)。

void clear()

清除所选区域中单元格的全部特性，如格式、内容、注释等。

void clearFormats()

清除所选区域中单元格的格式。

void clearContents()

清除所选区域中单元格的内容。

void autofit()

将列宽、行高设置为最佳。

以上对单元格区域类（Range）中一些比较重要的方法进行了介绍，如果对以上的说明有疑问，或想更进一步了解单元格区域类的使用，请参阅永中Office二次开发文档。

下面通过一段完整的示例代码来详细地介绍单元格区域类的使用：

//样例4.3

import application.Application;

public class RangeTest

{

public RangeTest()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.SPREADSHEET);

}

public void process()

{

//获得当前活动工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得当前活动工作表

Worksheet worksheet = workbook.getWorksheets().getActiveWorksheet();

//获得单元格区域"A1:D4"

Range range = worksheet.getRange("A1:D4");

//用0至1间的随机数来填充单元格区域

range.fillRandom();

//创建一个单元格格式化属性对象

FormatCellAttribute formatCellAttribute = new FormatCellAttribute();

//设置单元格格式为水平居中对齐

formatCellAttribute.setHorizontalIndex(CellFormatConstants.HOR\_MIDDLE);

//将单元格格式化属性设置到单元格区域中

range.setFormatCellAttribute(formatCellAttribute);

//创建一个填充属性

FillAttribute fillAttribute = new FillAttribute();

//将填充属性的的颜色设置为(255,204,153)

fillAttribute.setTexture(255,204,153);

//将填充属性设置到单元格区域中

range.setFillAttribute(fillAttribute);

//创建一个字体属性

FontAttribute fontAttribute = new FontAttribute();

//设置字体的粗体属性

fontAttribute.setBold(true);

//将字体属性设置到单元格区域中

range.setFontAttribute(fontAttribute);

//创建一个边框属性

BorderAttribute borderAttribute = new BorderAttribute();

//设置边框架类型为全部边框架格式

borderAttribute.setBorderType(BorderAndShadingConstants.BORDERTYPE\_ALL);

//设置上，左，右，下边框的颜色

borderAttribute.setBorderLineColor(BorderAndShadingConstants.TOP,Color.RED);

borderAttribute.setBorderLineColor(BorderAndShadingConstants.LEFT,Color.CYAN);

borderAttribute.setBorderLineColor(BorderAndShadingConstants.RIGHT,

Color.YELLOW);

borderAttribute.setBorderLineColor(BorderAndShadingConstants.BOTTOM,

Color.ORANGE);

//设置上，左，右，下边框的宽度

borderAttribute.setBorderLineWidth(BorderAndShadingConstants.TOP,8f);

borderAttribute.setBorderLineWidth(BorderAndShadingConstants.LEFT,5f);

borderAttribute.setBorderLineWidth(BorderAndShadingConstants.RIGHT,3f);

borderAttribute.setBorderLineWidth(BorderAndShadingConstants.BOTTOM,1f);

//将边框属性设置到单元格区域中

range.setBorderAttribute(borderAttribute);

//复制单元格区域的内容

range.copy();

//将单元格区域的内容粘贴至E5开始的单元格区域

worksheet.getRange("E5").paste();

//清除单元格区域的格式

range.clearFormats();

//清除单元格区域“E5：F6”中的内容

worksheet.getRange("E5:F6").clearContents();

//将单元格区域“E1:F1”合并

worksheet.getRange("E1:F1").merge();

}

public static void main(String[] args)

{

RangeTest rangeTest = new RangeTest();

rangeTest.process();

}

}

#### 4.4 行对象(Rows)

Rows。Rows类定义了为行对象设置行高，隐藏行和设置列属性等方法。

行对象由工作表处获得。

先请看以下关于行对象的代码

示例：选定第三行到第五行并设置行高和插入行。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//获得第三行到第五行的行对象

Rows rows = sheet.getRows(3 , 5);

//把13毫米转换为单位为磅的值

double ptValue = Utilities.otherToPoint(13.0,UnitConstants.MILLIMETER);

//设置行高

rows.setRowHeight(ptValue);

以下仅列举几个有关行对象的API，其它API可查阅永中Office的API 文档。

取得单元格区域对象：

Range getRange()

代码示例：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//获得第三行到第五行的行对象

Rows rows = sheet.getRows(3 , 5);

//获得其单元格区域对象

Range range = rows.getRange();

在Rows类中不提供插入行操作的API，要实现在选定的行处插入行这样的操作，在Worksheet类中提供了这样的API：

void insertEntireRows(int startRow, int endRow)

在选定行处插入整行。

代码示例：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//在第三行到第五行的之间插入整行

sheet.insertEntireRows(3, 5);

设置选定行的行高

void setRowHeight(double height)

设置该选定行的行高(默认度量单位：磅)。

注意：行高，为double型,单位:磅,其取值范围在0至409.0 磅

public static float otherToPoint(double length, int unit)

把以英寸、厘米或毫米为单位的值转换为以磅为单位的值。

参见：

参数：

length 长度或距离

unit 计量单位，其取值为如下常量

UnitConstants.INCH 英寸 UnitConstants.CENTIMETER 厘米 UnitConstants.MILLIMETER 毫米 UnitConstants.POINT 磅

代码示例：把1厘米转换为以磅为单位的值

float length = 1.0f;

代码示例：把第三行到第五行的行高设置为13.0毫米

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//获得第三行到第五行的行对象

Rows rows = sheet.getRows(3, 5);

double ptValue = Utilities.otherToPoint(13.0,UnitConstants.MILLIMETER);

//设置行高为13.0毫米

rows.setRowsHeight(ptValue);

void setRowHidden(boolean hide)

隐藏行。

void setFillAttribute(FillAttribute fa)

设置填充属性。

代码示例：给第三行设置背景色为红色的填充属性

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

//获得当前工作表

Worksheet sheet = sheets.getActiveWorksheet();

//获得第三行的行对象

Rows rows = sheet.getRows(3, 5);

//新建一个填充属性

FillAttribute fill = new FillAttribute();

//设置背景色

fill.setTexture(255,0,0);

//给行对象设置填充属性

rows.setFillAttribute(fill);

总之，还可以对行对象进行很多的属性设置，这里就不详细说明，可参照永中Office的API文档。

下面提供一个完整的示例代码

import application.Application;

public class TestRows

{

public TestRows()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.SPREADSHEET);

}

public void process()

{

//获得当前活动工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得当前活动工作表

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

//选中第三行到第五行

Rows rows = sheet.getRows(3, 5);

//选中行的行数

Int count = rows.getCount();

//得到选中行的行数

double ptValue = Utilities.otherToPoint(13.0,UnitConstants.MILLIMETER);

//设置行高为13.0毫米

rows.setRowsHeight(ptValue);

//获得第三行的行对象

Rows rows = sheet.getRows(3, 5);

//新建一个填充属性

FillAttribute fill = new FillAttribute();

//设置背景色

fill.setTexture(255,0,0);

//给行对象设置填充属性

rows.setFillAttribute(fill);

}

public static void main(String[] args)

{

TestRows myTest = new TestRows();

myTest.process();

}

}

#### 4.5 列对象 (Columns)

Columns。Columns类定义了对整列进行操作的一些方法，如设置列宽，隐藏列，设置列属性等。

列对象由工作表处获得。

先请看以下关于列对象的代码

示例：选定第三列到第五列并设置列高

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

//获得当前工作表

Worksheet sheet = sheets.getActiveWorksheet();

//获得第三列到第五列的列对象

Columns columns = sheet.getColumns(3, 5);

//把20毫米转换为单位为磅的值

double ptValue = Utilities.otherToPoint(20.0,UnitConstants.MILLIMETER);

//给列对象设置列宽

columns.setColumnWidth(ptValue);

以下仅列举几个有关列对象的API，其它API可查阅永中Office的API 文档。

取得单元格区域对象：

Range getRange()

代码示例：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//获得第三列到第五列的列对象

Columns columns = sheet.getColumns(3, 5);

//获得第三列到第五列的单元格区域

Range range = columns.getRange();

在Columns类中不提供插入列操作的API，要实现在选定的列处插入列这样的操作，在Worksheet类中提供了这样的API：

void insertEntireColumns(int startRow, int endRow)

在选定列处插入整列。

代码示例：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//在第三列到第五列的之间插入整列

sheet.insertEntireColumns(3, 5);

设置选定列的列宽：

void setColumnWidth(double width)

设置该选定列的列宽(默认度量单位：磅)。

注意：列宽，为double型,单位:磅,其取值范围在0至409.0 磅

public static float otherToPoint(double length, int unit)

把以英寸、厘米或毫米为单位的值转换为以磅为单位的值。

参见：

参数：

length 长度或距离

unit 计量单位，其取值为如下常量

UnitConstants.INCH 英寸 UnitConstants.CENTIMETER 厘米 UnitConstants.MILLIMETER 毫米 UnitConstants.POINT 磅

代码示例：把1厘米转换为以磅为单位的值

float length = 1.0f;

代码示例：把第三列到第五列的列对象的列宽设置为13毫米

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//获得第三列到第五列的列对象

Columns columns = sheet.getColumns(3, 5);

double ptValue = Utilities.otherToPoint(13.0,UnitConstants.MILLIMETER);

//设置列宽为13.0厘米

columns.setColumnWidth(ptValue);

void setColumnHidden(boolean hide)

隐藏选定的列。

代码示例：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//获得第三列到第五列的列对象

Columns columns = sheet.getColumns(3, 5);

//隐藏选定的列

columns.setColumnHidden(true);

void setFontAttribute(FontAttribute ca)

设置字体属性。

代码示例：给第三列设置前景色为红色的字体

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//获得第三列的列对象

Columns columns = sheet.getColumns(3, 3);

//新建一个字体属性

FontAttribute font = new FontAttribute();

//设置前景色

font.setForeground(255,0,0);

//给列对象设置字体属性

columns.setFontAttribute(font);

总之，还可以对列对象进行很多的属性设置，这里就不详细说明，可参照永中Office的API文档。

下面提供一个完整的示例代码，

import application.Application;

public class TestColumns

{

public TestColumns()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.SPREADSHEET);

}

public void process()

{

//获得当前活动工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得当前活动工作表

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

//获得第三列到第五列的列对象

Columns columns = sheet.getColumns(3, 5);

//获得列对象的列数

int count = columns.getCount();

//获得第三列的列对象

//获得第三列的列对象

Columns columns = sheet.getColumns(3, 3);

//新建一个字体属性的类

FontAttribute font = new FontAttribute();

//设置前景色

font.setForeground(255,0,0);

//给列对象设置字体属性

columns.setFontAttribute(font);

}

public static void main(String[] args)

{

TestColumns myTest = new TestColumns();

myTest.process();

}

}

#### 4.6 单元格 (Cell)

Cell。Cell 是电子表格应用中一个常用的类，该类定义了对单元格对象进行操作的一些方法，如设置和获得单元格的值，插入注释，设置和获取单元格的属性等。

先请看以下关于单元格的代码

示例： 在当前活动单元格设置数据和获得数据

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

//获得当前工作表

Worksheet sheet = sheets.getActiveWorksheet();

//获得当前活动单元格

Cell cell = sheet.getActiveCell();

//给单元格设值

cell.setValue("永中Office");

//获得单元格的值

Object val = cell.getValue();

//获得单元格的值的字符串值

String valStr = cell.getCellString();

单元格可以方便用户填写类似表格之类的报告。以下仅列举几个API，其它API可查阅永中Office的API 文档。

设置单元格的值，Cell提供了以下几个API：

void setValue(String value)

void setValue(long value)

void setValue(boolean bol)

void setValue(double value)

给单元格设置内容。设置的参数类型可为以下几种：字符串、长整型、布尔值、双精度型。

获得单元格的值，Cell提供了2个API：

Object getValue()

String getCellString()

以上第一个API返回的是Object 对象。如果单元格的值是整型，则Object 对象是Integer 型，类似的，浮点型对应于Double,字符串对应于String 等。

第二个API 返回的是单元格的值的String值。

复制、剪切、粘贴单元格：

void copy()

void cut()

void paste()

第一个API为复制，第二个API为剪切，第三个API为粘贴。

代码示例：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//获得第二行第二列的单元格

Cell cell = sheet.getCell(2, 2);

//设置单元格的值

cell.setValue("永中Office");

//复制单元格

cell.copy();

Cell cell2 = sheet.getCell(4 , 4);

//粘贴单元格

cell2.paste();

String val = cell2.getCellString();

Cell提供了设置和获得单元格格式化属性的几个API， 如字体、填充、边框和单元格格式属性。

void setFormatCellAttribute(FormatCellAttribute attr)

设置单元格格式属性。

FormatCellAttribute getFormatCellAttribute()

获得单元格格式属性。

void setFillAttribute(FillAttribute fa)

设置填充属性。

FillAttribute getFillAttribute()

获得填充属性。

void setFontAttribute(FontAttribute ca)

设置字体属性。

FontAttribute getFontAttribute()

获得字体属性。

void setBorderAttribute(BorderAttribute ba)

设置边框属性。

BorderAttribute getBorderAttribute()

获得边框属性。

代码示例：设置单元格字体并获得单元格字体

//ActiveWorksheet.select("E12");

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

//获得当前活动单元格

Cell cell = sheet.getActiveCell();

//新建一个字体属性

FontAttribute font = new FontAttribute();

font.setBold(true);

font.setForeground(255,0,0);

//设置字体

cell.setFontAttribute(font);

//获得单元格的字体属性

FontAttribute font2 = cell.getFontAttribute();

提供了对单元格批注操作的API：

Comment insertComment()

插入注释。

Comment getComment()

获取单元格内的注释。

boolean hasComment()

判断单元格是否有注释。

void clearComments()

清除单元格中的注释。

代码示例：在当前单元格中插入批注

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

//获得当前活动单元格

Cell cell = sheet.getActiveCell();

//在单元格中插入批注

Comment comment = cell.insertComment();

//给批注设置内容

comment.setText("It's my comments.");

设置单元格的链接区域：

void linkRange(String linkAddress)

参数为链接地址。

代码示例：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//获得单元格区域

Range range = sheet.getRange("A1:B4");

//对单元格区域进行设置

range.setValue("永中Office");

Cell cell = sheet.getCell("C1");

//设置单元格C1的链接区域"A1：B4"

cell.linkRange("A1:B4");

在单元格里插入图片：

void insetPicture(String pictureName)

参数： pictureName 为带完全路径的图片的文件名

代码示例：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

Cell cell = sheet.getCell("A1");

//在单元格A1中插入图片

cell.insertPicture("d:\\temp\\样品.jpg");

在单元格内插入影音文件：

void insertMultimedia(String name)

参数 name 为带完全路径的影音文件的文件名。

代码示例：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

Cell cell = sheet.getCell("A1");

//在单元格内插入影音文件

cell.insertMultiMedia("C:\WINDOWS\\clock.avi");

总之，还可以对单元格进行很多的属性设置，这里就不详细说明，可参照永中Office相关的ＡＰＩ文档。

下面提供一个完整的示例代码，对当前单元格进行设值并插入注释。

import application.Application;

public class TestCell

{

public TestCell()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.SPREADSHEET);

}

public void process()

{

//获得当前活动工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得当前活动工作表

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

//获得当前单元格

Cell cell = sheet.getActiveCell();

//对当前单元格进行设置

cell.setValue("永中Office");

//给当前单元格插入注释

Comment comment = cell.insertComment()

comment.setText("It’s my comment.");

}

public static void main(String[] args)

{

TestCell myTest = new TestCell();

myTest.process();

}

}

## 第五章 文字处理

本章将为您介绍永中Office文字处理的宏及二次开发。

本章内容共分为以下15节：

5.1 文档集合（Documents）

5.2 文档对象（Document）

5.3 正文对象（BaseText）

5.4 节对象（Section）

5.5 页眉（Header）、页脚对象（Footer）

5.6 脚注（FootNote）、尾注对象（EndNote）

5.7文本范围对象（TextRange）

5.8 段落集合（Paragraphs）

5.9 段落对象（Paragraph）

5.10 自由表格集合（Tables）

5.11 表格对象（Table）

5.12 书签 (Bookmarks、Bookmark)

5.13修订（TrackChanges、TrackChange）

5.14文字处理的常用属性集（字体、段落、边框、底纹）

5.15工具类（Utilities）

#### 5.1 文档集合（Documents）

后缀名.EIO的文件是永中Office默认的文件格式，单个文件对应一个Workbook对象，每个文件可以包含多篇文档，管理当前文件中多个文档的集合，Office中定义了一个集合对象Documents。

**获得Documents对象**

从Workbook对象中可以获得当前文件所包含的所有文档的集合Documents。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

**使用Documents对象**

Documents是一个集合对象，管理着文档对象Document的添加、删除、获取、重命名等常用的操作。

用 addDocument方法可创建一篇新的空文档，并将其添加到 Documents 集合中。以下示例在当前集合文件中创建一个空文档，并显示新文档的名字。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

说明：当使用addDocument方法创建的空文档，返回的是一个默认格式的名字。格式如下：以字符串“文档”为前缀，后面加上一个整型数字，数字从1开始逐步累加。如“文档1”，“文档2”等等。

此外还可以通过参数创建一个指定名字的文档。

Document addDocument(String name)

参数name为指定的名字

通过getDocument(String name)方法可以得到指定名字的文档对象。参数name就是指定的名字。此方法返回一个Document对象实例。以下示例得到指定名字的文档，并选择整个文档的内容。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.getDocument("文档1");

document.selectAll();

说明：“文档1”就是先前创建过的文档的名字。有一点用户必须注意：参数name必须是当前文件中已经打开的文档的名字。如果文档不存在或为关闭状态，则返回Null。

我们经常在开发中使用当前激活的文档对象，可以通过getActiveDocument方法得到这个活动状态的对象。

以下示例删除刚才创建的文档。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

documents.removeDocument("文档1");

closeDocument方法可以将文件中的某个文档关闭，使其暂时从Documents集合中移除。但是与removeDocument方法不同的是，关闭文档只是临时改变文档的显示状态，并不将其从文件中删除，可以通过openDocument重新打开这个文档。

以下示例关闭先前创建的文档。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

documents.closeDocument("文档1");

#### 5.2 文档对象（Document）

作为文字处理应用的实体，Document对象代表一篇文档，是Documents集合中的一个元素，可通过这个对象对文档内容添加删除修改或者得到当前文档的指定内容等。文档的内容是十分丰富的，可以有文本、图形、表格、样式等。



文档作为文字处理应用编辑存储内容的载体，从体系结构可划分为模型层和表现层两个部分。模型层代表文档内容定义存储的框架，一般由正文、段落、文本等组成。表现层指的是文档布局显示的一种方式，也就是我们通常所说的文档的视图。一般有普通视图、页面视图、大纲视图三种。

我们写一篇文档内容通常是在四个页边距以内的空白区域编辑操作，如左图所示，这个空白区域称为正文编辑区域，BaseText对象负责管理着这个编辑区域。我们在编辑复杂文档如用户手册时通常还需要在文档每页的顶部区域显示每个章节的标题，在页面的底部位置显示每页的页码，这个时候就可以用到文档的页眉和页脚功能区域。页眉是文档中每个页面页边距的顶部区域，页脚即是这个页面的底部区域，他们可以在每页显示相同的内容。页眉编辑区域用Header对象表示，而Footer对象代表某一个页脚编辑区域。有时为了给文档中特定的文本提供解释、批注以及相关的参考资料，可用脚注尾注功能进行标识。通常可用脚注对文档内容进行注释说明，而用尾注说明引用的文献。FootNote对象代表一个脚注区域，尾注编辑区域则用EndNote表示。通常一篇文档可以有多个脚注、尾注用于文档内容的注释说明，这些对象是在给正文某一段内容添加了说明标识后创建的。

为了满足用户个性化的要求，使文档更加美观，可以通过添加自选图形或文本框来丰富文档的排版。每个自选图形或文本框都是一个小的正文编辑区域，可以实现简单的文本编辑，我们这里用Shape对象表示一个自选图形或文本框，而要编辑内容时需使用文本框自带的编辑器，这个文本编辑器就是ShapeText对象，可从Shape对象得到。

以上六个编辑区域组成了文档编辑的入口，在这些区域中通过添加文本、图像或其他对象充实文档的内容。文本内容的组合根据其逻辑结构可划分为多个段落，每个段落代表逻辑上相关联的一段连续文本。段落对象是文本组成的重要逻辑对象，Paragraph代表一个段落，每个编辑区域都可以有多个段落。

下图清晰地表达了文档内容的基本模型：

文档

正文

页眉

页脚

脚注

尾注

段落

文本

章节

文本框

文本框集合

作为文档内容基本模型的根节点，文档对象管理着整个文档的状态及样式，并且对文档的各种复杂操作都是从文档对象进入的，Document对象是以文档操作的入口定义的。所以创建一篇精美的文档必须从Document对象着手，Document对象在整个文字处理中的作用不言而喻。

**获得Document对象**

从文档集合Documents中可以得到指定的文档对象。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.getDocument("文档1");

如果某个文档处于活动状态，可以通过文档集合Documents中的getActiveDocument方法得到当前激活窗口的文档对象。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.getActiveDocument();

在文档集合中创建一个空文档或者打开文档也能返回一个Document对象，可以通过这个文档对象对一篇文档进一步地处理工作。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument("新文档");

**使用Document对象**

Document对象作为文字处理的入口，管理着整个文档的状态。通过上面文档模型图的介绍，我们知道文档的内容是由章节、编辑区域、段落、文本等以层次递进的方式组成的，可以这种方式获取文档的任意内容，添加、删除、更改。文字处理功能繁多，可对文档部分文本添加批注，插入图形、表格等来修饰文档的页面，为了方便用户对文档某些关键字段的跟踪定位，还可定义书签，创建索引目录等。这些功能都是文档级的，也就是创建的批注、图形、书签等由文档统一管理，但是如果将这些功能模块都放在Document对象中定义使用，势必会让Document对象臃肿难当，给用户的二次开发造成诸多不便。所以我们在封装API时用到了对象管理器这种新的模式。

Document对象的功能主要划分为以下几种：

1、整个文档状态的管理。

永中Office是一个集成的办公软件，可以同时打开多个文档，但某一时刻只有一篇文档是被激活的，我们可以用isActivate ()方法判断文档是否处于活动状态，也可以使用activate, ()方法激活这个文档对象。以下示例依次创建两篇空白文档，然后激活第一篇文档，最后打印文档的状态。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document doc1 = documents.addDocument("文档1");

Document doc2 = documents.addDocument("文档2");

doc1.activate();

如果addDocument()不传任何参数，我们使用永中Office默认的命名规则创建文档，文档名字依次为“文档 1”、“文档 2”……。这个名字并不一定是用户认可的文档名字，用户可对文档重命名。Document对象有一个Name属性，我们可通过get、set方法改变获取文档的名字。以下示例创建一个空白文档，修改文档的名字，然后打印名字。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

document.setName("我的文档");

默认情况下我们创建的文档具有最高权限，既可以查看也可以编辑修改，但是有时候我们可能会有这样的需求，用户甲撰写的文档流转给其他用户时，只允许其他用户查看此文档，修改权限只赋予撰写者本人。通过文档保护功能，我们可以实现这个需求。永中Office的文档保护功能既可以锁定整个文档，也可以将文档设置成修订保护状态，还能够仅允许用户修改批注，强大的保护功能增加了公文流转时文档操作权限的灵活性。

protect(int protectType, String password)

设置文档的保护状态。

参数protectType是一个整型常量，表示文档保护的方式，可以取以下值：

ProtectConstants.NO\_PROTECT = -1 (无保护，相当于取消保护状态)

ProtectConstants.TRACKCHANGES = 0 （修订保护方式，即操作记录改为修订

状态）

ProtectConstants.COMMENTS = 1 （仅允许插入或编辑注释）

ProtectConstants.DOCUMENT = 2 （锁定全文）

ProtectConstants.SECTION = 3 (保护章节)

参数password指定保护文档的密码

以下示例创建一篇新文档，然后将文档设置为保护状态，保护的类型为保护全文，保存文件至桌面，最后关闭文件。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument("我的文档");

Document.protect(ProtectConstants.DOCUMENT, "123456");

// 保存文件

workbook.saveAs("C:\\Documents and Settings\\coffee\\桌面\\测试文件1.eio");

// 关闭文件

workbook.close();

运行代码，然后打开刚才的文档“测试文件1.eio”，你可以发现文档被锁定了，

说明：workbook.saveAs("C:\\Documents and Settings\\coffee\\桌面\\测试文件1.eio")的参数为保存的文件完整路径，用户可以改变这个参数到相关的路径。

protect方法只有在当前文档的状态是非保护状态时才能使用，如果文档当前已经是保护状态，必须先进行取消保护操作。判断是否为保护状态，可以调用getProtectType()方法，如果返回值非-1的整数，则为保护状态。

取消文档的保护状态调用unprotect(String password)方法，此时你需要通过保护时的密码解锁。

以下示例打开桌面文件“测试文件1.eio”，打印文档的状态，然后取消文档的保护状态，再次打印当前文档的状态，以比较取消保护前后文档状态的变化。

Workbooks workbooks = Application.getWorkbooks();

Workbook workbook = workbooks.openWorkbook("C:\\Documents and Settings\\coffee\\桌面\\测试文件1.eio");

Documents documents = workbook.getDocuments();

Document document = documents.getDocument("我的文档");

前面我们说的都是保护整个文档，如果文档中有多个章节，也可以仅对某一个或几个章节锁定。永中Office提供了protectSections(int[] sectionsIndex, String password)锁定某写章节，参数sectionsIndex是一个整型数组，代表需要被保护的章节索引号。参数password为锁定章节设置的密码。

2、整个文档内容的管理

从Document对象中可以得到当前文档的基本信息，如文档正文的内容、文档的长度、文档的页数等。下面几个get方法可获得二次开发中常用的文档信息。

long getLength () 返回全文的文本长度。说明：只返回文档正文部分的长度。

String getText() 返回全文的文本。说明：只返回文档正文部分的文本内容。

int getPageCount() 返回文档的总页数。

int getCurrentPageIndex() 获得当前光标所在位置的页码索引。

int getSectionCount() 返回全文的章节数。说明：默认情况下，空白文档只有一个章节，当在正文编辑区域插入分节符后，文档被分隔成多个章节，每个节可进行不同的页面设置。

以下示例创建一篇空白文档，添加多个段落，在文本末尾插入分节符，然后在第二个章节输入文本，打印全文的信息，如全文长度、文档总页数，章节数。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 获得文档的正文编辑区域

BaseText baseText = document.getBaseText();

// 获得正文区域的段落集合，然后添加二个段落

Paragraphs paragraphs = baseText.getParagraphs();

paragraphs.addParagraph("第一个段落的内容");

paragraphs.addParagraph("第二个段落的内容");

// 获得全文的长度

long length = document.getLength();

// 在文档的末尾插入一个分节符

document.insertBreak(length, BreakConstants.BREAK\_SECTION\_NEXTPAGE);

length = document.getLength();

baseText.insertText(length, "第三个段落的内容");

上面的例子在文本的末尾插入一个分节符，为此我们在Document对象中封装了一个方法document.insertBreak(length, BreakConstants.BREAK\_SECTION\_NEXTPAGE);。此方法的作用是在文档的末尾插入一个分节符，类型为下一页。除了分节符外，通过类型常量我们还能插入分页符、分栏符等。

void insertBreak(long offset, int breakType)

在文档指定位置插入一个分隔符。

参数offset指明了分隔符插入的位置。说明：分隔符只能在正文中插入，所以offset取的是当前文档的绝对位置。

参数breakType代表分隔符的类型常量，通过这个常量我们可以插入不同的分隔符。

具体常量如下

BreakConstants.BREAK\_SECTION\_NEXTPAGE = 0 (分节符下一页)

BreakConstants.BREAK\_SECTION\_CONTINUOUS = 1 (分节符连续)

BreakConstants.BREAK\_SECTION\_EVEN = 2 (分节符偶数页)

BreakConstants.BREAK\_SECTION\_ODD = 3 (分节符奇数页)

BreakConstants.BREAK\_PAGE = 4 (分页符)

BreakConstants.BREAK\_COLUMN = 5 (分栏符)

BreakConstants.BREAK\_LINE = 6 (换行符)

细心的读者可能会发现，上面的代码还一些方法或类比较陌生，没关系，我们将在以后具体对象的讲解中详细阐述。paragraphs.addParagraph方法参考Paragraphs集合对象，baseText.insertText方法参考BaseText对象。说明：在以后的示例中经常用的其他对象的方法，我们不再一一解释，可参考对应方法所在的对象。

3、访问文字处理模型层子对象。

我们知道写文档通常在四个页边距内的空白位置处进行编辑，这个编辑区域称为正文编辑区域，用BaseText对象表示。正文对象BaseText是文字处理最常用的对象，从Document对象可以得到整个文档的正文编辑区域对象。以下示例在新建的文档中输入文字。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0,"中华人民共和国");

简单的文档一个章节就可以了，我们平常创建的空白文档就包含唯一的一个章节。但是有时候文档逻辑比较复杂，我们需要将一篇文档按逻辑划分为多个章节，每个章节可以有不同的页面布局方式。比如说打印文档时，可以按章节以不同的页面排版方式打印，我们可以将第一个章节用正常的竖排方式打印，而有时候流程图、结构图等竖排的A4纸容不下，这时就可以在文档中插入第二个章节，然后对这个章节重新修改页面的设置，可以将第二章节设置为横排打印方式或者改变纸张来源为A3纸，这样任何复杂的文档都能够打印了。有了节对象，我们可以灵活地改变一篇文档各个部分页面视图的布局，就像上面介绍的打印设置，也可以为每个章节设置不同的网格视图。为了方便对某一个章节下的正文部分操作，我们给BaseText对象赋予了另一层语义。如果从某一个节对象Section获得的BaseText对象，则此时这个对象代表某一个章节的正文编辑区域。这样修改某一章节的内容时就能以相对于当前章节的位置信息操作数据了。

以下示例在新建的文档中输入文字，然后在文本的末尾插入一个分节符添加新章节，在下一个章节的起始位置出输入文字，最后添加指定位置处的页眉信息。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 获得全文的正文编辑区域

BaseText baseText = document.getBaseText();

baseText.insertText(0,"中华人民共和国");

// 获得全文的长度

long length = document.getLength();

// 在文档的末尾插入一个分节符

document.insertBreak(length, BreakConstants.BREAK\_SECTION\_NEXTPAGE);

// 得到第二个章节

Section section2 = document.getSection(1);

// 得到第二章节的正文部分

BaseText baseText2 = section2.getBaseText();

// 在第二章节的正文起始位置插入文字

baseText2.insertText(0, "江苏省无锡市");

// 返回指定位置的章节

Section section1 = document.getSectionByOffset(3);

// 在返回章节的页眉中输入文字

Header header = section1.getHeader();

header.insertText(0,"地理信息");

当用户使用代码对文本进行操作或格式化时，必须指定文本操作的范围。文本编辑区域中的方法都需要传入offset和length两个参数指定一个范围。但是如果对某一个范围指定多个操作，调用多次方法势必会增加相同参数的传递次数，降低了方法的便利性。为此我们定义了一个文本范围对象，用TextRange对象表示。使用 TextRange 对象可以定义文档的某一部分，每一个 TextRange 对象由一起始和一终止字符位置定义。一个 TextRange 对象小至只是一个插入点大至包括整篇文档，所以使用是非常灵活方便的。

通过Document对象的TextRange getRange(long startOffset, long endOffset)方法我们可以定义文档的某一个范围，参数startOffset、endOffset分别代表文本范围在当前文档中的起始和终止位置。之所以称TextRange对象为动态文本范围是因为这个文本范围是动态可变的，如果在这个范围内插入文字，TextRange对象范围也会跟着变大，反之删除内容时，范围也动态的缩小。

以下示例新建文档，输入文字，指定一个文本范围，然后对这个文本范围进行选中操作。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 获得全文的正文编辑区域

BaseText baseText = document.getBaseText();

baseText.insertText(0, "中华人民共和国于1949年10月1日建国啦！");

// 得到文档正文指定位置范围的文本范围，这里指定是1949年10月1日这段文本

TextRange range = baseText.getRange(8, 18);

// 将得到的文本范围标识为选中状态

range.select();

有了TextRange对象，我们可以方便地对用户当前选中的一段区域进行操作了，这在定义一些快捷功能时特别有用。我们通过Document对象的TextRange getSelectionRange()方法获得窗口当前选中的文本范围，你不需要知道这段区域位于文档的哪个逻辑位置，只需要直接调用方法就可以了，而且这些方法都不需要再次传入位置参数。怎么样，是不是很方便啊？

以下示例将当前选中的文字格式化为粗体。请注意：您必须在当前的文字处理中选中一段文字，然后再运行下面的代码，效果为选中文字设为粗体了。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 得到文字处理当前选中的文本范围

TextRange range = document.getSelectionRange();

// 初始化一个字体属性

FontAttribute font = new FontAttribute();

// 设置字体属性为粗体

font.setBold(true);

// 将粗体字体属性应用于当前选中区域

range.setFontAttribute(font);

4、获得文字处理视图层对象。

刚才我们介绍了文字处理的模型层，现在再谈谈文字处理在视图上的表现---视图层。所谓视图就是模型内容对应的一种具体显示方式，视图不需要存盘，用户打开文档后根据视图的类型重新布局文档的页面。文字处理的视图层可分为普通视图、页面视图、大纲视图。通常新建一篇空白文档后，永中Office自动切换到页面视图，也就是说页面视图是我们默认的视图表现。通过文档视图管理器DocumentView对象我们可以切换到指定的视图。例如：documentView.getPageView()切换到页面视图，documentView.getOutlineView()切换到大纲视图。大纲视图的作用是简化了文本内容的格式设置，以便您将精力集中在文档结构上。大纲视图体现了文档的一种样式结构。文本的基本样式可分为标题1、标题2……正文等，每一种样式对应文档结构的一个层次，比如标题1在文档结构中是最高级的，依次为标题2、标题3等。我们在大纲视图中可以清楚地看出这种文档层次结构的变化。

以下示例创建一篇空白文档，输入中文，切换到页面视图，设置文档页面为稿纸样式。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Document document = workbook.getDocuments().addDocument();

document.getSelectionRange().insertTextAfter("中华人民共和国江苏省");

// 得到当前文档的视图管理对象

DocumentView docView = document.getDocumentView();

// 切换到页面视图，并获得这个视图对象

PageView pageView = docView.getPageView();

// 从指定位置获得页面视图的稿纸信笺管理器

LetterPaper paper = pageView.getLetterPaper(2, 6);

// 设置页面稿纸方式为方格字

paper.setStyle(GridConstants.GRID\_STYLE);

// 将页面设置的属性应用于当前的页面视图

pageView.setLetterPaper(2,6,paper);

以下示例创建一空白文档，插入多行文字，对应每一行应用不同的样式，如首行为标题1，第二行标题2等，切换到大纲视图，查看页面视图的变化。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 获得文档正文的段落管理器

Paragraphs paragraphs = document.getBaseText().getParagraphs();

// 在正文中添加三个段落

Paragraph para1 = paragraphs.addParagraph("中华人民共和国");

Paragraph para2 = paragraphs.addParagraph("江苏省");

Paragraph para3 = paragraphs.addParagraph("无锡市");

// 获得当前文档的所有样式的集合

Styles styles = document.getStyles();

// 将每一段格式化为不同的样式

para1.formatStyle(styles.getStyle("样式 1"));

para2.formatStyle(styles.getStyle("样式 2"));

para3.formatStyle(styles.getStyle("样式 3"));

// 得到当前文档的视图管理对象

DocumentView docView = document.getDocumentView();

// 切换到大纲视图

OutlineView outlineView = docView.getOutlineView();

5、获得文字处理功能模块的管理器。

在制作复杂文档时，我们经常会用到自选图形、表格、域代码等功能模块，政府公文流转过程中最实用的是修订功能，而书签模块可以进行文本定位。以上这些功能模块大都是对整个文档进行的操作，我们通过前面的介绍可知，对于文档的具体操作，文字处理一般都封装了各个功能模块的管理器，方便用户使用。

下面例举了文字处理常用的功能模块：

BookMarks getBookmarks() 得到书签模块的管理器

TrackChanges getTrackChanges() 得到修订模块的管理器

WpShapes getShapes() 得到自选图形模块的管理器

Tables getTables() 得到表格模块的管理器

WpComments getComments() 得到批注模块的管理

Fields getFields() 得到域模块的管理器

Styles getStyles() 得到样式模块的管理器

这些管理器分管各自的功能模块，其实每个功能模块都有多个实例，每个实例封装了实现这个功能的对象。比如说书签功能模块，我们可以为指定的文本范围添加一个书签标识以便进行文本定位或书签内容的替换。书签是文档级的功能，可以在当前文档中添加多个书签标识，每个书签标识就是一个书签对象BookMark，而BookMarks对象作为书签对象的管理器提供了书签标识的添加、删除、获取等集合对象常见的操作，所以说BookMarks等管理器对象也是一个集合对象，就像前面我们提到的Documents集合对象一样。管理器是集合对象的一个扩展，除了添加删除等集合操作外，还可以自定义一些某些功能模块特定的方法，比如locateBookmark(String bookmarkName)可以通过书签标识名定位到书签指定的位置。

以下示例在创建一空白文档，添加文字并在指定位置添加一个书签标识，然后定位到这个书签位置，替换书签的内容。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 得到文档的正文编辑区域

BaseText area = document.getBaseText();

// 插入文字

area.insertText(0, this is a test!\nthis is a test！");

// 得到指定位置的文本范围

TextRange range = area.getRange(9, 12);

// 得到文档的书签管理器

BookMarks bookmarks = document.getBookmarks();

// 添加书签

BookMark bookmark = bookmarks.add(range, "书签1");

// 改变书签的内容

bookmark.setValue("保佑您！");

// 将光标定位到此书签位置

bookmarks.locateBookmark("书签1");

通过管理器BookMarks集合对象的add（TextRange range, String bookmarkName）方法我们添加了一个书签标识，这里需要传两个参数range和书签名。range对象指定一段连续的文档范围，可以通过Document对象的getRange(long start, long end)得到这个范围，参数指定了范围的起始结束位置。添加书签能够将指定范围与书签标识有机的结合在一起，可书签标识操作这个范围对象。

Shpaes对象是当前文档中所有自选图形Shape对象的一个集合，管理着Shape对象的添加、删除、获取等操作。我们可以通过Shapes集合对象获取指定的Shape对象，然后就可以对这个图形移动修改了。

以下示例在文档中添加一个矩形和椭圆形。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 得到自选图形模块的管理器

Shapes shapes = document.getShapes();

// 在文档中添加一个矩形和椭圆形

shapes.addShape(ShapeConstants.SHAPE\_RECTANGLE, 100, 100, 250, 200);

shapes.addShape(ShapeConstants.SHAPE\_OVAL, 300, 300, 450, 400);

说明：Shape addShape(int type, double startX, double startY, double endX, double endY)将某个自选图形添加到文档页面上，参数type为自选图形的类型，是一个ShapeConstants的常量值，我们这里ShapeConstants.SHAPE\_RECTANGLE指定就是矩形，后面四个参数指定自选图形添加在页面的位置及大小，以像素为单位，起始位置（0，0）从页面左上角位置开始计算。

还有一些管理器不再一一介绍，请参看永中Office的API文档。

6、文档页面属性。

为了使文档做得美观，可以对文档的页面进行编排修饰，如设置页面边框、设置水印等。打印文档前需使文档页面符合纸张的大小，所以可以通过Document得到页面设置管理器等。

为了对完稿的文档进行页面的编排以适应打印纸张的大小，我们可以通过PageSetup对象进行页面的设置。例如，我们将要用A3纸打印文档，所以需要将源文档的页面重新调整以适应A3的大小，可以调用pageSetup对象的setPaperType（String paperType）方法设置打印页面大小的属性。参数paperType为打印纸张的类型，有“A3、A4、自定义”等。请注意：在属性管理器（又称属性集）中设置完属性后必须将属性集应用到相关对象上才能使设置生效。如：设置打印纸张为A3，然后调用document对象的setPageSetup(PageSetup pageSetup)方法将这个属性集应用于指定的文档，此时你可以看见文档的页面变成A3纸了。以上说的是设置对象属性集的一种通用方式，不仅对应于Document对象，FontAttribute（字体属性）、ParagraphAttribute（段落属性）、PageBorderAttribute（页面边框属性）对象也是一样的用法。

以下示例修改新建文档的页面纸张尺寸为A3，然后打印该文档。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 获得当前文档的页面设置管理器

WpPageSetup pagesetup = document.getPageSetup();

// 设置属性页面纸张尺寸为A3

pagesetup.setPaperType("A3");

// 将设置过的属性应用于当前的文档

document.setPageSetup(pagesetup);

// 得到当前文档的打印管理器

Print print = document.getPrint();

// 打印文档

print.print();

在平常的业务中，我们经常能够看见有些文件印有“保密”、“草案”等背景，这些就是通过水印功能打印出来的，用于标识文件的重要程度。以下示例给文档添加一个文本水印，水印内容为“绝密”。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 得到水印管理器

WaterMarkAttribute watermark = document.getWatermarkAttribute();

//设置水印类型为文本水印

watermark.setType(WatermarkConstants.TEXT\_WATERMARK);

//设置文本水印的内容

watermark.setText("保密");

// 将设置好的水印属性应用于文档

document.setWatermarkAttribute(watermark);

#### 5.3 正文对象（BaseText）

正文编辑区域位于文档页面上、下、左、右四个也边距之内的空白区域。所谓页边距指的是正文区域四周到页面各边界之间的距离。通常，可在页边距的上方放置页眉，页边距的下方放置页脚。页边距的左边预留空白，一般放置装订线，用于文档的装订。通常，文档内容的主体就是放置在正文编辑区域中的。正文编辑区域在Office中定义为一个正文对象-----BaseText对象。

**获取BaseText对象**

BaseText对象代表文档的正文部分，用于放置文档的主体内容。通过Document对象的BaseText getBaseText()方法可以得到整个文档的正文编辑区域。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

简单的文档一个章节就可以了，我们平常创建的空白文档就包含唯一的一个章节。但是有时候文档逻辑比较复杂，我们需要将一篇文档按逻辑划分成多个章节，每个章节可以有不同的页面布局方式。如果从某一个节对象Section获得的BaseText对象，则此时这个对象代表某一个章节的正文编辑区域。这样修改某一章节的内容时就能以相对于当前章节的位置信息操作数据了。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 得到当前光标位置的文本范围对象

TextRange range = document.getSelectionRange();

range.insertTextAfter("中华人民共和国");

// 获得全文的长度

long length = document.getLength();

// 在文档的末尾插入一个分节符

document.insertBreak(length, BreakConstants.BREAK\_SECTION\_NEXTPAGE);

// 得到第二个章节

Section section2 = document.getSection(1);

// 得到第二章节的正文部分

BaseText baseText2 = section2.getBaseText();

**使用BaseText对象**

BaseText对象代表文档的正文部分，保存着文档内容的主体部分。通过BaseText对象，可以对文档的主体内容进行各种各样地文本操作及格式化操作，改变文档的内容及外观。BaseText对象实现文字处理的常用操作，主要概括为以下几类：简单文本操作、访问正文的内容、丰富文档的内容、文本的格式化、复制粘帖操作、复杂文本操作等。

1. 简单文本操作

简单文本操作顾名思义就是改变文本内容的基本操作，一般有添加文本、删除文本、添加段落等，也可以通过方法转换文本的内容，例如可对指定的文本范围进行大小写转换，简繁体转换等。

insertText(long offset, String content)

代表在正文的指定位置添加一个字符串。参数offset指定文本插入的位置，这个偏移量相对与当前正文的起始位置。参数content代表添加的内容。

Paragraph insertParagraph(long offset)

代表在正文的指定位置插入一个回车符，这时文本以参数offset指定的位置作为边界划分成两个段落。参数offset指定回车符插入的位置。

以下示例在新建的空白文档的起始位置添加一段文字，然后在偏移位置8处插入一个回车符。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 在正文的指定起始位置处添加文字

baseText.insertText(0, "this is a test！");

// 在偏移位置8处插入一个回车符，将文本分为2个段落

baseText.insertParagraph(8);

通过insertSymbol(long offset, char symbol)方法可插入一个特殊字符，参数offset指定插入字符的位置，参数symbol是一个char类型，可通过Unicode码添加一个特殊字符。Unicode码是以16进制编码的描述所有字符的一种编码方式，在Java中以\u开头，后面连接一个16进制数。如：\u0110代表字符**?**。我们也可以通过insesrtText(long offset, String content)方法添加一段特殊字符，只要将特殊字符在字符串中以Unicode码代替就可以了。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 在正文起始位置插入一个↑箭头，Unicode码为0x2191

baseText.insertSymbol(0, '\u2191');

调用delete(long offset, long length)方法可以删除正文文本的内容，参数offset指定删除内容范围的起始位置，参数length指定范围的长度。delete方法除了删除文本外，还能删除图片，域，书签等对象，只要这些对象嵌入与文本，并且在删除范围之内就都会清除。以下示例删除指定范围的全部内容。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "this is a test！");

//删除偏移位置为2，长度为6的文本范围的内容

baseText.delete(2, 6);

文字处理支持将指定范围的文本转换为繁体，或者进行大小写转换。

ChangeCase(long offset, long length, int type) 代表将指定范围的文本进行大小写转换。参数offset指定转换范围的起始位置，参数length指定转换范围的长度。参数type代表转换的类型，可以取以下常量：

ChangeCaseConstants.SENTENCECASE 表示句首字母大写

ChangeCaseConstants.LOWERCASE 表示小写

ChangeCaseConstants.UPPERCASE 表示大写

ChangeCaseConstants.TITLECASE 表示词首字母大写

ChangeCaseConstants.TOGGLECASE 对所选内容进行大小写转换，即大写字母转

换为小写，小写字母转换为大写

ChangeCaseConstants.DBCCASE 表示半角

ChangeCaseConstants.SBCCASE 表示全角

tcscConverter(long offset, long length, boolean tcsc) 代表将指定范围的文本进行简繁体转换。参数offset指定转换范围的起始位置，参数length指定转换范围的长度。参数tcsc是boolean值，如果为true，代表简体转换到繁体，反之表示繁体转换到简体。

还能对正文的某一范围文本所在的段落进行增加缩进或者减少缩进。

increaseIndent(long offset, long length)代表对参数指定的文本范围增加缩进一个制表位。

decreaseIndent(long offset, long length)代表对参数指定的文本范围增加减少一个制表位。

以下示例新建文档，添加文字，然后对指定范围的文本进行大小写和简繁体的转换，并且对指定文本范围的内容增加缩进一个制表位。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "Welcome to wuxi！\n");

baseText.insertText(0, "this is a test!");

// 将偏移位置为0，长度为20的文本范围的内容转换为大写

baseText.changeCase(0 ,20, ChangeCaseConstants.UPPERCASE);

// 将偏移位置为0，长度为20的文本范围的内容转换为繁体

baseText.tcscConverter(21, 13, true);

// 将偏移位置为2，长度为10的文本范围的内容增加缩进一个制表位

baseText.increaseIndent(2, 10);

1. 访问正文内容

BaseText对象提供了一些方法可以返回正文编辑区域的文本内容，如文本内容、文本长度、段落个数等。

通过long getLength()可返回当前正文部分的文本长度，String getText(long offset, long length)返回指定范围的文本内容，参数offset，length指定了一个文本的范围，offset代表文本范围的起始位置，length代表文本范围的长度。说明：我们在正文中对文本的访问或操作都需要指定一个文本的范围，参数offset和length指定了这个范围，以后的方法参数也是代表这个意思，我们不再具体介绍。

以下示例新建空白文档，输入文字，然后打印正文的长度和指定范围的文本内容。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "this is a test！");

// 打印正文的长度

// 打印正文偏移位置为9，长度为3文本范围的内容

通过Paragraphs getParagraphs()可得到当前正文的段落对象管理器，即Paragraphs集合对象，从这个集合对象我们就可以知道段落的个数。通过BaseText对象的Paragraph getParagraph(long offset)方法可以返回指定位置的段落对象。参数offset代表指定的正文位置。

从以下示例新建文档，添加多个段落，然后打印正文的段落总数及指定位置9的段落文本内容。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "this is a test！");

// 得到当前正文所有段落的集合

Paragraphs paragraphs = baseText.getParagraphs();

// 打印正文段落的数量

// 获得正文偏移位置为9的段落，并打印其文本内容

Paragraph para = baseText.getParagraph(9);

我们可以通过BaseText对象的TextRange getRange(long start, long end)方法定义一个正文中的文本范围，参数start、end指定这个文本范围在正文中的起始和结束位置。以下示例返回一个指定的文本范围对象，然后将这个范围高亮选中。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "this is a test！");

TextRange range = baseText.getRange(0, 8);

range.select();

1. 丰富文档的内容

BaseText对象除了可以进行简单的文本操作（如添加删除文字、转换文字格式等）外，还能添加图片声音等多媒体的内容，丰富文档的内容。BaseText对象对于图片声音等多媒体内容的添加支持两种方式。一种是以多媒体文件默认的大小插入文档，还可以通过参数指定多媒体内容在文档中的实际显示大小。

insertPicture(long offset, String filepath) 代表在正文指定的位置插入默认文件大小的图片。参数offset指定了图片在正文中的插入点位置，参数filepath代表了图片文件的实际路径。

insertPicture(long offset, String filepath, float width, float height) 代表在正文指定的位置插入指定显示尺寸的图片。参数width、height指定了图片的显示尺寸宽和高，度量单位为磅值，您可以通过工具类Utilities的float centimeterToPoint(float centimeter)将厘米转换为磅值，也可以使用Utilities对象的其他静态方法进行单位的转换。

insertAudio(long offset, String filepath) 代表在正文指定位置插入默认文件大小的声音。

insertAudio(long offset, String filepath, float width, float height) 代表在正文指定位置插入指定显示尺寸的声音。

insertVideo(long offset, String filepath) 代表在正文指定位置插入默认文件大小的视频。

insertVideo(long offset, String filepath, float width, float height) 代表在正文指定位置插入指定显示尺寸的。

以下示例在新建的文档内添加两张图片，第一张图片以文件默认尺寸插入，第二张指定了图片的显示尺寸。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 在正文的起始位置插入图片sample1.jpg，图片大小为图片文件默认的大小

baseText.insertPicture(0, "d:\\sample1.jpg");

// 在正文的末尾插入图片sample2.jpg，同时指定图片大小为宽3厘米、高2厘米

baseText.insertPicture(baseText.getLength(), "d:\\sample2.jpg", Utilities.centimeterToPoint(3.0f), Utilities.centimeterToPoint(2.0f));

我们希望文档中的某些内容能够随着时间或不同的使用者自动变化。例如当您在正文中添加了一个作者名时，文本显示创建文档的作者。有时文档的作者改变了，比如有新人接替了原来的撰写者，在文字处理中通过文档属性对话框可以修改这些值。这时我们希望正文中添加的作者名也能够自动更新。幸好文字处理的自动图文集或域代码可以实现上述的要求。域代码是文字处理十分有用的功能，通过域代码可以实现文本内容的自动更新。比如我们刚才提到的作者名等自动图文集就是域代码的一种。以后提到的日期时间、页码、数字等其实也是用域代码实现的，您可以用域代码添加需要自动更新的内容。

insertAutoText(long offset, String entryName) 代表在正文的指定位置添加一个自动图文集，参数entryName指定了自动图文集的条目，您可以在AutoTextConstants常量接口中找到指定的条目。以下例举几个条目对应的常量：

AutoTextConstants.PAGE "- 页码 -";

AutoTextConstants.CREATED\_BY "作者"

AutoTextConstants.CREATED\_ON "创建时间"

AutoTextConstants.FILENAME\_PATH "文件名称和路径"

以下示例在空白文档的其实位置添加一条自动图文集。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 插入自动图文集---机密、页码、日期

baseText.insertAutoText(0,AutoTextConstants.CONFIDENTIAL\_PAGE\_DATE);

您可以通过insertNumber(long offset, int number, int format)方法添加指定样式的数字。参数number代表添加的某数字，参数format代表插入数字的样式。数字也是域的一种，所以可取下列域常量：

FieldConstants.DEFAULT 使用默认数字样式

FieldConstants.Arabic 数字样式为“1,2,3,...”

FieldConstants.ALPHABETIC 数字样式为“A,B,C,...”

FieldConstants.alphabetic 数字样式为“a,b,c,...”

FieldConstants.ROMAN 数字样式为“I,II,III,...”

FieldConstants.roman 数字样式为“i,ii,iii,...”

FieldConstants.CHINESENUM3 数字样式为“一,二,三(简),...”

FieldConstants.ZODIAC1 数字样式为“甲,乙,丙,...”

FieldConstants.ZODIAC2 数字样式为“子,丑,寅,...”

FieldConstants.CHINESENUM2 数字样式为“壹,贰,叁 ,...”

FieldConstants.GB1 数字样式为“⒈,⒉,⒊,...”

FieldConstants.GB2 数字样式为“⑴,⑵,⑶,...”

FieldConstants.GB3 数字样式为“①,②,③,...”

FieldConstants.GB4 数字样式为“㈠,㈡,㈢,...”

以下示例在文档的起始位置中插入数字5，数字样式为“子,丑,寅,...”。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 在起始位置插入数字5，数字样式为“子,丑,寅,...”

baseText.insertNumber(0, 5, FieldConstants.ZODIAC2);

当您在文档中添加了日期、时间后我们发现这些文字随着时间的改变会自动更新。其实日期时间也是域代码的一种表现形式。您完全可以使用域对象Field实现日期时间等的添加。

Field对象代表一个域代码，可以通过域代码集合Fields集合对象添加删除域代码到文档。由于我们在文档的任何位置添加的域代码都是有Fields集合对象进行管理的，所以可以通过Document对象的Fields getFields()方法获得域代码管理器，进行域的添加。

Fields对象的Field insert(TextRange range, int fieldID, String codeStr)方法代表在指定的文档位置添加一个域代码。

参数range指定了域代码在文档中插入的位置，这里使用TextRange文本范围对象是因为您可以在正文、页眉页脚、脚注尾注等任何编辑区域内添加，插入点为文本范围的起始位置。参数fieldID是一个域编号，代表域的类型，使用FieldConstants常量接口中的常量值。参数codeStr代表具体的域代码，是一个字符串。您可以通过域代码实现各种域效果。具体的域请参看永中Office的API文档。

以下示例在光标位置插入一个域代码，域类型为文件的最后编辑时间。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Document document = workbook.getDocuments().addDocument();

BaseText baseText = document.getSection(0).getBaseText();

TextRange range = document.getSelectionRange();

// 得到域管理器

Fields fields = document.getFields();

// 在光标位置插入一个域代码

Field field = fields.insert(0,baseText, FieldConstants.FIELDTYPE\_EDIT\_TIME, "EDITTIME \\# \"0\" \\\* Arabic");

通过BaseText对象的insertDate(long offset, String type, boolean update, boolean isSBC)方法可以在正文指定的位置添加一个指定格式的日期域。参数type插入日期域的格式。您可以从DateTimeConstants常量接口中获得指定的常量值，以下例举几个常量：

DateTimeConstants.DATE\_EEEE\_O\_A 日期格式为“二○○二年八月九日”

DateTimeConstants.DATE\_yyyy\_M\_d 日期格式为“2002年8月9日”

DateTimeConstants.DATE\_aaaa 日期格式为“星期五”

DateTimeConstants.DATE\_yyyy\_m\_d 日期格式为“2002-8-9”

DateTimeConstants.DATE\_yyyy\_m\_d\_h\_mm\_PM 日期和时间格式为“2002-8-9 1:30 PM”

参数update是一个boolean值，代表是否更新域。如果为true，则更新域；否则不更新

参数isSBC 代表日期是否为全角字符，只有当参数update为false时才可用

insertTime(long offset, String type, boolean update, boolean isSBC)代表在正文指定的位置添加一个指定格式的时间域。参数与insertDate(long offset, String type, boolean update, boolean isSBC)方法相似，您可以参看相关说明。以下例举几个插入时间的格式常量：

DateTimeConstants.TIME\_hh\_mm 时间格式为“14:30”

DateTimeConstants.TIME\_h\_mm\_PM 时间格式为“2:30 PM”

DateTimeConstants.TIME\_hh\_mm\_ss 时间格式为“14:30:00”

DateTimeConstants.TIME\_HH\_MM\_SS 时间格式为“14时30分00秒”

DateTimeConstants.TIME\_PM\_H\_M 时间格式为“下午2时30分”

以下示例在空白文档中先后添加日期域和时间域，样式由DateTimeConstants常量指定。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 在起始位置插入日期，日期格式为“2002年8月9日”

baseText.insertDate(0, DateTimeConstants.DATE\_yyyy\_M\_d, true, false);

// 在正文末尾插入时间，时间格式为“14时30分”

baseText.insertTime(baseText.getLength(), DateTimeConstants.TIME\_HH\_MM, true, false);

文字处理支持HTML文档中常见的超链接方式，您可以在文档中为某一部分文字或图形添加一个超链接信息，然后当点击这个对象时就可以马上链接到超链信息指定的地址。这个超链接信息的地址是多种多样的，您可以指定一个网页的HTTP地址，一封Email邮件，也可以指定本文档中的某一个区域，如一个书签、标题、或者电子表格的一部分。我们在Office中定义了一个超链接的对象------Hyperlink对象。此对象负责保存超链接的一些信息，如链接地址，链接提示，链接类型等。

通过BaseText对象的Hyperlink getHyperlink(long offset, long length)方法可以得到正文指定范围的一个超链接属性集，如果当前文本范围中没有插入超链接，则返回一个初始化的超链接属性集。您可以通过Hyperlink属性集对象设置超链接的信息。最后通过insertHyperlink(long offset, long length, Hyperlink hyperlink)方法，将超链接信息设置到指定范围的文本或图形对象中。

Hyperlink对象是超链接信息的属性集，您可以通过Hyperlink对象的方法设置超链的类型、链接地址、显示的文本、提示信息等。

Hyperlink对象的setHyperlinkType(int type)方法代表设置超链接的类型。参数type指定超链接的类型，可以取以下的常量值：

HyperlinkConstants.OLD\_FILE - 表示原有的文件或网页；

HyperlinkConstants.THIS\_BINDER\_POSITION - 表示本文档中的位置；

HyperlinkConstants.NEW\_BINDER - 表示新建文档；

HyperlinkConstants.EMAIL\_ADDRESS - 表示邮件地址

setScreenTip(String tipText) 代表设置超链的提示信息。参数tipText指定提示信息。

setAddress(String address) 代表设置超链接的地址。参数address指定链接的地址，这个地址必须是网页的HTTP地址，如果要链接到其他的地方，如链接至邮件或者电子表格的一部分，您可以使用Hyperlink对象的其他方法。例如：setEmailAddress(String emailAddress)代表链接到指定的电子邮件。hyperlinkToCell(String sheetName, String cellAddress)代表链接到本文件指定工作表的某单元格。参数sheetName指定工作表的名字，参数cellAddress指定单元格地址。单元格地址是这样定义的：取行号+列号。行号以A、B、C等英文字母为序，列号用数字表示。例如“B5”代表第2行第5列的单元格地址。

以下示例为“this is a test”添加一个超链接，链接到新浪的主页，并且设置了提示信息。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "this is a test！");

// 得到指定文本范围的超链接属性，如果没有则创建一个新对象

Hyperlink hyperlink = baseText.getHyperlink(0, 8);

// 设置超链接的类型为链接到网页

hyperlink.setHyperlinkType(HyperlinkConstants.OLD\_FILE);

// 设置链接提示信息

hyperlink.setScreenTip("超链到新浪网");

// 设置链接的新浪主页的HTTP地址

// 为指定范围的文本设置超链

baseText.insertHyperlink(0, 8, hyperlink);

脚注和尾注主要用于在打印文档中为文档中的文本提供解释批注以及相关的参考资料。可以为正文中的某一范围的文本添加一个脚注或尾注，用于解释这个区域的文本内容或者提供一些参考资料。BaseText对象提供了添加脚注尾注的方法。

FootNote addFootNote(long offset, long length, int applyTo)代表位置正文中指定范围的文本添加一个脚注信息。参数applyTo指定脚注标识更改应用于的范围，可以取以下常量：

NoteConstants.APPLY\_TO\_WHOLE = 0; 将更改应用于整篇文档

NoteConstants.APPLY\_TO\_SECTION = 1; 将更改应用于本节

NoteConstants.APPLY\_TO\_TEXT = 2; 将更改应用于所选文字

说明：当给正文指定的范围文本添加了一个脚注信息后，在这个文本范围的上方回来显示一个数字标识，这个标识称为脚注标识，用于标记脚注在文档中的索引。您可以通过脚注尾注的属性管理器来更改这个标识的样式。脚注属性管理器Office中定义了FootNoteOptions对象。FootNoteOptions对象管理文档中某一范围脚注标识所有的显示格式、起始编号、编号方式等。通过FootNote addFootNote(long offset, long length, int applyTo)方法参数applyTo我们指定了添加的脚注标识应用于文档的哪一个范围，也就是这个脚注标识是由哪一个FootNoteOptions对象管理的。FootNoteOptions对象管理范围内的所有的脚注的标识的格式及序号等都是相关的。

通过BaseText对象的FootNoteOptions getFootNoteOptions(long offset)方法您可以得到某个范围的脚注属性管理器。FootNoteOptions对象提供了一些改变脚注标识格式或者其他信息的方法。这里例举几个，如果使用请参考永中Office API文档的FootNoteOptions类介绍。

setPosition(int position) 代表设置脚注的位置。参数position指定脚注内容在文档的显示位置。

NoteConstants.BELLOW\_PAGE = 0; 代表脚注显示在页面底端

NoteConstants.BELLOW\_WORD = 1; 代表脚注显示在文字下面

setNumberStyle(int index) 代表设置脚注的编号格式。参数index代表脚注编号格式的索引，在NoteConstants常量接口中可找到对应的常量，以下例举几个常量值：

NoteConstants.NUMBER\_DECIMAL 编号格式为“1,2,3,...”

NoteConstants.NUMBER\_UPPER\_ALPHA 编号格式为“A,B,C,...”

NoteConstants.NUMBER\_UPPER\_ROMAN 编号格式为“I,II,III,...”

NoteConstants.NUMBER\_UPPER\_CHINA 编号格式为“壹,贰,叁,...”

NoteConstants.NUMBER\_SIMPLE\_LOWER\_CHINA 编号格式为“一,二,三,...（简）”

NoteConstants.NUMBER\_EAVEN 编号格式为“甲,乙,丙,...”

setStartingNumber(int startingNumber) 代表设置脚注的起始编号。参数startingNumber指定脚注的起始编号。

刚才我们介绍的都是脚注属性的set方法，FootNoteOptions对象为提供了相应的get方法用于访问指定的属性。

FootNote addFootNote(long offset, long length, String customSymbol, String fontName,

int applyTo)代表位置正文中指定范围的文本添加一个自定义标识的脚注信息。参数customSymbol代表脚注的自定义标记。如果参数为null，则脚注标识采用数字编号，编号格式索引根据FootNoteOptions属性集得到。参数fontName代表自定义标记的字体属性名，如果参数为null或字体名在系统中不存在，则取默认字体“宋体”。

我们可以通过以下方法添加一个尾注。EndNoteOptions与FootNoteOptions对象相似，定义了一个尾注的属性集，可使用脚注属性集相似的方法获取EndNoteOptions对象。由于尾注添加的方式与脚注相似，我们这里不再重复介绍，请参看上面介绍的脚注对象。

EndNote addEndNote(long offset, long length, int applyTo) 代表位置正文中指定范围的文本添加一个脚注信息。

EndNote addEndNote(long offset, long length, String customSymbol, String fontName,

int applyTo) 代表位置正文中指定范围的文本添加一个自定义标识的脚注信息。

以下示例为“永中公司”添加一个尾注，然后在“欢迎您”的文本范围内插入一个脚注标识。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 得到文档第一个节对象

Section section1 = document.getSection(0);

// 得到第一节的正文部分

BaseText baseText = section1.getBaseText();

baseText.insertText(0, "永中公司\n欢迎您！");

// 为指定文本范围添加一个尾注

EndNote endnote = baseText.addEndNote(0, 4, NoteConstants.APPLY\_TO\_WHOLE);

// 为指定文本范围添加一个脚注

FootNote footnote = baseText.addFootNote(9, 3, NoteConstants.APPLY\_TO\_TEXT);

// 填写尾注、脚注的内容

endnote.insertText(0, "无锡新区管委会投资的软件公司");

footnote.insertText(0, "英文为welcome to you");

1. 格式化文本

文字处理支持修改文本的样式，美化文本的显示效果。文字处理的文本样式包括段落样式和字符样式两种。其中段落样式用于设置段落外观的格式例如对齐方式、缩进格式、制表位、行间距等。而字符样式用于设置段落内选定文字的格式。例如文字的字体、字号、字形等。即使某段落已整体应用了某种段落样式，该段中的字符仍可以有自己的样式格式化字体属性。

1、字体属性

文字处理的字符样式最常用的就是修改文本的字体属性。Office为文本的字体属性封装了一个属性对象，用于管理字体的相关属性，如字体名、大小、颜色等。FontAttribute对象在Office中定义为管理字体的一个属性集。通过BaseText对象的FontAttribute getFontAttribute(long offset)方法可以得到指定位置的字体属性。我们也可以通过FontAttribute的构造器创建一个初始化状态的字体属性集。通过FontAttribute对象我们可以设置字体的大小、字形、颜色等，甚至还能定义文字的字符间距属性，使文字之间的排版更适于用户的需要。当设置好字体属性后，通过BaseText对象的setFontAttribute(long offset, long length, FontAttribute fontAttr)方法可以将改变过的字体属性应用与指定的文本范围。调用上面的setFontAttribute方法将使范围的文本应用FontAttribute对象的所有属性。如果只是希望将修改过的字体属性应用于文本，而其他字体属性保留不变时，可以使用下面的方法。

setFontAttribute(long offset, long length, FontAttribute fontAttr, boolean replace) 代表将字体属性集改变过的属性应用于指定的文本范围。参数offset、length指定了应用的范围，参数fontAttr代表格式化的字体属性，参数replace是一个boolean值，如果为true，表示表示替换原先的属性；否则，表示保留原先属性。

以下例举一些常用的字体属性：

setFontFamily(String fontName) 代表设置英文字体。参数fontNmae指定英文字体的名字，如果当前不支持参数字体，则不作任何操作。

setAsianFontFamily(String fontName) 代表设置亚洲字体。参数fontNmae指定亚洲字体的名字，如果当前不支持参数字体，则不作任何操作。

setItalic(boolean isItalic) 代表设置字体为斜体。参数isItalic如果为true，则字体为斜体。

setBold(boolean isBold) 代表设置字体为粗体。参数isBold如果为true，则字体为粗体。

setFontSize(double fontSize) 代表设置字体的大小。参数fontSize以浮点数精确表示字体的尺寸，取值范围为1 ～ 1638磅之间。

setForeground(Color color) 代表设置字体的前景色。参数color代表字体的颜色。

以下示例将指定范围的文本格式化为指定样式的字体属性，设置字体为粗体，前景色为红色。最后打印当前范围字体的颜色，以确定指定的字体格式被应用了。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

BaseText baseText = section1.getBaseText();

baseText.insertText(0, "永中公司\n欢迎您！");

// 初始化一个字体属性集

FontAttribute fontAttr = new FontAttribute();

// 设置字体为粗体

fontAttr.setBold(true);

// 设置字体的前景色为红色

fontAttr.setForeground(Color.red);

// 将改变的字体属性应用于指定的文本范围

baseText.setFontAttribute(0, 8, fontAttr);

// 从指定位置获取字体属性，并打印字体的颜色

fontAttr = baseText.getFontAttribute(0);

2、段落属性

文字处理的段落样式指的是以文本段落为格式化的基本单位，设置段落的基本外观。段落的外观属性最常见的是设置段落的对齐方式、段间距、缩进等段落属性，还可以边框、底纹等属性为整个段落添加边框背景等，使文本段落更加美观。段落属性在Office中定义了一个属性集------ParagraphAttribute对象。通过BaseText的ParagraphAttribute getParagraphAttribute(long offset)方法可以得到指定位置所在段落的段落属性集。我们也可以通过ParagraphAttribute的构造器创建一个初始化状态的段落属性集。当设置好段落属性后，通过BaseText对象的setParagraphAttribute(long offset, long length, ParagraphAttribute paragraphAttribute)方法可以将改变过的段落属性应用与指定的段落区域。和字体属性一样，我们也可以通过参数replace来控制是否需要将指定的段落属性全部替换某个段落属性。ParagraphAttribute对象提供了对齐方式、段落缩进、段间距、换行分页等属性，可以控制整个段落或各段落之间的布局。以下例举一些常用的段落属性设置方法。同样可通过get方法访问相应的属性。

setAlignment(int alignment) 代表设置段落的对齐方式。参数alignment指定段落的对齐方式，可以取以下几个常量：

ParagraphConstants.ALIGNMENT\_LEFT = 0 左对齐

ParagraphConstants.ALIGNMENT\_CENTER = 1 居中

ParagraphConstants.ALIGNMENT\_RIGHT = 2 右对齐

ParagraphConstants.ALIGNMENT\_JUSTIFIED =3 两端对齐

ParagraphConstants.ALIGNMENT\_DISTRIBUTED =4 分散对齐

setLeftIndent(float leftIndent) 代表设置段落的左缩进距离。参数leftIndent指定段落左缩进的距离，以磅值为单位，取值范围在-1584到1584之间。

setLineSpacingAt(float lineSpacingAt) 代表设置段落的行间距。参数lineSpacingAt指定段落的行间距，参数值语义依据行间距的类型来定义。如果行距类型为单倍、1.5倍、双倍行距，则设置行间距无效。如果行距类型为多倍行距，则参数值单位为行数，范围在0.06到132字行之间。如果行距类型为最小值、固定值，则参数值单位为磅值，范围在0.7到1584之间。

setSpaceAbove(float spaceAbove) 代表设置段前间距。参数spaceAbove指定段前间距的大小，以磅值为单位，取值范围在0到1584之间。

setFirstLineIndent(float firstLineIndent) 代表设置首行缩进距离。参数firstLineIndent指定首行缩进的大小。

以下示例格式化指定文本范围的段落属性，设置段落首行缩距离进为2厘米，然后打印当前范围段落的首行缩进属性。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

BaseText baseText = section1.getBaseText();

baseText.insertText(0, "永中公司\n欢迎您！");

// 初始化段落属性

ParagraphAttribute paraAttr = new ParagraphAttribute();

// 设置段落为首行缩进，缩进距离为2厘米

paraAttr.setFirstLineIndent(Utilities.centimeterToPoint(2.0f));

// 将设置的属性应用于指定的段落区域

baseText.setParagraphAttribute(0, 8, paraAttr);

// 从指定位置得到段落属性，并打印首行缩进的大小

paraAttr = baseText.getParagraphAttribute(0);

float firstIndent = Utilities.pointToCentimeter(paraAttr.getFirstLineIndent());

3、格式化边框及底纹属性。

边框和底纹可以有效地美化文档版面的外观。在文字处理中可以给页面、文字、图片以及表格添加边框，并可用底纹填充文字及表格的背景，用于强调格式化边框或者底纹的文本的内容，这样能够大大吸引人的注意力。边框属性在Office中定义为BorderAttribute对象，而页面边框用PageBorderAttribute对象定义。BorderAttribute对象定义了某一个文本对象或者表格、图片的边框。边框是由上、下、左、右等方向的边框线组成的，每条边框线可设置其的线形、颜色、粗细等属性。根据边框应用于的对象，边框的边框线个数也不同。如果边框应用于文本或者段落，则边框线最多可以有上、下、左、右四个方向。如果边框应用与表格或单元格区域，则处理四条外边框线外，还能设置中间水平、中间垂直、左上对角线、右上对角线最多八条边框线。通过setBorderType(int borderType)方法指定边框的类型。参数borderType定义了边框的类型，通常取以下几个常量：

BORDERTYPE\_NONE = 0 表示无边框

BORDERTYPE\_BOX = 1 表示方框格式

BORDERTYPE\_SHADOW = 2 表示阴影边框格式

BORDERTYPE\_THREED = 3 表示三维边框格式

BORDERTYPE\_CUSTOM = 4 表示自定义格式

这里需要特别说明的是：并不是说设置了边框的某种类型，边框就采用某中效果。边框的显示效果还与边框线的设置有关。例如：如果我们设置边框的类型为BORDERTYPE\_NONE，但是却定义了边框线的式样，这时Office的底层系统会自动将边框的类型设置为自定义格式。这种机制使边框属性的设置更灵活也更简单。每一条边框线都有样式、粗细、颜色等属性，我们通过在BorderAttribute对象的方法中传递参数borderPosition来指定设置或者访问的是哪一条边框线。以下介绍了边框线属性的常用方法：

setBorderLineType(int borderPosition, int lineType) 代表设置指定边框线的线形。参数borderPosition指定了设置的是哪一条边框线，也就是边框线的方向，可以取以下方向常量：

BorderAndShadingConstants.LEFT = 0 左边缘边框

BorderAndShadingConstants.TOP = 1 上边缘边框

BorderAndShadingConstants.RIGHT = 2 右边缘边框

BorderAndShadingConstants.BOTTOM = 3 下边缘边框

BorderAndShadingConstants.MIDDLE\_HORIZON = 4 中间水平线边框

BorderAndShadingConstants.MIDDLE\_VERTICAL = 5 中间垂直线边框

BorderAndShadingConstants.DIAGONAL\_FORWARD = 6 左上对角线边框

BorderAndShadingConstants.DIAGONAL\_BACKWARD = 7 右上对角线边框

参数lineType指定边框线的线形，是一个整型的索引值，对应边框对话框的线形选项。

这里需要注意的是：参数指定的边框线必须是应用的边框允许使用的，如果边框应用于文字或者段落，则常量4~7就设置无效，访问值始终为0。

setBorderLineColor(int borderPosition, Color color) 代表设置指定边框线的颜色。参数color指定线的颜色。参数borderPosition同上。

setBorderLineWidth(int borderPosition, float width) 代表设置指定边框线的粗细。参数color指定线的宽度。

可以通过get方法得到相应边框线的属性。例如：float getBorderLineWidth(int borderPosition) 代表某一条得到边框线的宽度。参数borderPosition指定返回线宽的某一条边框线的常量，与set方法一致。

底纹属性在Office中定义为ShadingAttribute对象。通过BaseText的ShadingAttribute getShadingAttribute(long offset)方法可以得到指定位置的底纹背景。我们也可以通过ShadingAttribute的构造器创建一个初始化状态的底纹属性集。当设置好底纹属性后，通过BaseText对象的setShadingAttribute(long offset, long length, ShadingAttribute shading)方法可以将改变过的底纹属性应用与指定的文本区域。底纹属性定义了某一文本范围背景的图案或颜色。通过setTextureIndex(int textureIndex)方法，可以设置底纹的图案或纹理，参数textureIndex指定底纹的纹理在底纹对话框中的索引。

以下示例新建文档，添加文字，然后给指定的文本范围设置一个方框边框属性，设置上边框的线形为实线，粗细为8磅，颜色为红色。最后打印当前文本范围的上边框的颜色。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

BaseText baseText = section1.getBaseText();

baseText.insertText(0, "永中公司\n欢迎您！");

// 初始化一个边框属性

BorderAttribute border = new BorderAttribute();

// 设置边框的类型为方框

border.setBorderType(BorderAndShadingConstants.BORDERTYPE\_BOX);

// 设置上边框的线形为实线

border.setBorderLineType(BorderAndShadingConstants.TOP,1);

// 设置上边框的颜色为红色

border.setBorderLineColor(BorderAndShadingConstants.TOP,Color.RED);

// 设置上边框的粗细为8磅

border.setBorderLineWidth(BorderAndShadingConstants.TOP,8f);

// 将更改的边框属性应用与指定范围

baseText.setBorderAttribute(0, 8, border);

// 得到指定位置的边框属性，并打印上边框的颜色

Border = baseText.getBorderAttribute(0);

以下示例新建文档，添加文字，然后给指定的文本范围设置一个底纹背景，底纹图案的索引为15，底纹前景色为红色。最后打印当前文本范围的底纹的索引。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

BaseText baseText = section1.getBaseText();

baseText.insertText(0, "永中公司\n欢迎您！");

// 初始化一个底纹背景属性

ShadingAttribute shading = new ShadingAttribute();

// 设置底纹图案的索引为15

shading.setTextureIndex(15);

// 设置底纹的前景色为红色

shading.setForeground(Color.red);

// 将更改的底纹属性应用与指定范围

baseText.setShadingAttribute(0, 8, shading);

// 得到指定位置的底纹属性，并打印底纹图案的索引

shading = baseText.getShadingAttribute(0);

1. 拷贝粘帖操作

Office支持强大的拷贝粘帖功能，通过BaseText对象的copy(long offset, long length)方法将参数指定文本范围的内容复制到剪贴板，然后可通过paste(long offset)方法将剪贴板的内容粘帖到参数指定的位置。

以下示例复制正文位置0至9范围的内容，然后将其粘帖到正文的末尾。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

BaseText baseText = section1.getBaseText();

baseText.insertText(0, "永中公司\n欢迎您!");

baseText.copy(0, 9);

baseText.paste(baseText.getLength());

前面介绍的copy、paste方法只能粘帖普通不带格式的文本，如果相应复制粘帖带格式的文本或者粘帖链接的话就需要调用BaseText对象的pasteSpecial(long offset, int type, int format)方法。方法代表将剪贴板的内容进行选择性粘帖，参数代表选择性粘帖的方式。参数offset指定粘帖的位置。

参数type代表选择性粘帖的类型，只能取以下常量：

PasteSpecialConstants.PASTE\_NORMAL 粘贴

PasteSpecialConstants.PASTE\_LINK 粘贴链接

PasteSpecialConstants.PASTE\_HYPERLINK 作为超级链接

参数format代表粘帖的格式，只能取以下常量：

PasteSpecialConstants.PASTE\_PLAINTEXT 未格式化文本

PasteSpecialConstants.PASTE\_RICHTEXT 格式化文本

PasteSpecialConstants.PASTE\_SHEET 工作表对象

PasteSpecialConstants.PASTE\_SLIDE 幻灯片对象

这里需要说明的是：只有当剪贴板的内容为文本，表格，幻灯片时才能调用pasteSpecial方法以选择性粘帖的方式粘帖。如果您需要复制粘帖的内容是图片或者自选图形，可调用paste(long offset)方法。paste方法对任何对象的粘帖都有效。

以下示例在新建的文档中添加文本，然后格式化文本字体大小为18.5磅，字体颜色为粉红，然后拷贝这段格式化的文本，以格式化富文本的方式粘帖到指定文档的末尾。这里粘帖选择性粘帖的类型为粘帖链接。粘帖链接代表复制源的内容改变，粘帖的内容也会跟着改变。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

BaseText baseText = section1.getBaseText();

baseText.insertText(0, "永中公司\n欢迎您!");

// 设置字体名为“用中楷体”，字体前景色为粉红色，字体大小为18.5磅

FontAttribute fontAttr = new FontAttribute();

fontAttr.setAsianFontFamily("楷体");

fontAttr.setFontSize(18.5);

fontAttr.setForeground(Color.pink);

// 将指定范围的文本进行字体格式化

baseText.setFontAttribute(0, 8, fontAttr);

// 复制指定范围的文本内容进入剪贴板

baseText.copy(0, 9);

// 以格式化富文本的方式粘帖链接到文档的末尾

baseText.pasteSpecial(baseText.getLength(), PasteSpecialConstants.PASTE\_LINK, PasteSpecialConstants.PASTE\_RICHTEXT);

#### 5.4 节对象（Section）

为了便于文档的格式化以及阅读的方便可以利用分节符将文档分成多个章节。文字处理中各个节的起始位置和终止位置是由用户自己定义的。节既可以是一个段落也可以只是段落中的某一部分还可以是多个段落甚至可以是整篇文档。不进行分节时，文字处理默认整个文档仅为一个章节。如果文档需要多个章节，可在合适的位置插入分节符。分节符是节的结束标记。在分节符中保存了节的各种格式设置元素。文档分节后，可以对各节独立地进行格式化而不会影响其他节的格式。总之节的设置是非常灵活的，将文档分为节的功能给文档的制作带来了极大的方便。节在永中Office中定义为一个Section对象。

插入的节根据分节符类型的不同，表现出不同的作用域。分节符类型共包括四个选项：下一页、连续、偶数页以及奇数页。下一页分节符代表将插入一个分节符的新节从下一页开始。连续分节符代表将插入一个分节符的新节从本页开始。偶数页或奇数页分节符代表将插入一个分节符的新节从下一个偶数页或奇数页开始。分节符在页面中显示为包含有分节符字样及分节符类型说明的双虚线。我们可以根据这个标记查看文档的节。

**获得Section对象**

通过Document对象的Section getSection(int index)方法，你可以得到某一个章节，方法返回指定的节对象。你可以从Section对象中获得本章节的正文、页眉、页脚、脚注尾注等文本编辑区域。文字处理默认整个文档只为一个章节，如果需要多个章节，则需在文档合适位置插入分节符。分节符将文档划分为多个章节，每个节可以有自己的格式化。Document对象的介绍中我们曾经提到过insertBreak(long offset, int breakType)方法代表插入一个分隔符，当参数breakType取0~3的整型值时，表示插入的是分节符。

BreakConstants.BREAK\_SECTION\_NEXTPAGE = 0 分节符下一页

BreakConstants.BREAK\_SECTION\_CONTINUOUS = 1 分节符连续

BreakConstants.BREAK\_SECTION\_EVEN = 2 分节符偶数页

BreakConstants.BREAK\_SECTION\_ODD = 3 分节符奇数页)

参数offset表示分节符在全文的插入位置。

以下示例将文档分为两个章节，然后分别获取这两个节对象。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

// 第一个节的正文部分

BaseText basetext1 = section1.getBaseText();

basetext1.insertText(0,"中华人民共和国\n江苏省无锡市\n新区管委会");

// 插入偶数页分节符

document.insertBreak(7, BreakConstants.BREAK\_SECTION\_EVEN);

// 获得第二个节对象

Section section2 = document.getSection(1);

// 第二个节的正文部分

BaseText basetext2 = section2.getBaseText();

basetext2.insertText("永中公司欢迎您！");

**使用Section对象**

节作为文档模型的一部分，可以从Section对象中取得本节的正文、页眉、页脚、脚注尾注等文本编辑区域。所以说节是Document文档对象访问其他编辑区域子对象的一个桥梁。鉴于上面介绍的每个文本编辑区域，我们都会详加介绍的，所以这里只是简单的例举。

BaseText getBaseText() 获得本节的正文编辑区域。

Header getHeader() 获得本节的默认页眉编辑区域。

Header getHeader(int type) 获得本节某一种类型的页眉编辑区域。参数type可以取首页、奇数页、偶数页中的一种。

Footer getFooter() 获得本节的默认页脚编辑区域。

Footer getFooter(int type) 获得本节某一种类型的页脚编辑区域。参数与getHeader方法相同。

FootNote getFootNote(int index) 获得本节指定的脚注编辑区域

EndNote getEndNote(int index) 获得本节指定的尾注编辑区域

节的设置便于文档的格式化以及打印阅览，给文档的制作带来了极大的方便。我们可以为每一个章节设置不同的页面分栏，也可以改变每一个节的页面设置，打印出个性化的效果。

以下示例将新建的文档划分为两个章节，并将第二个节的页面设置为横向打印。设置页眉距页边距1.5厘米，页脚距页边距1厘米。最后打印整篇文档。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 获得整个文档的正文编辑区域

BaseText baseText = document.getBaseText();

baseText.insertText(0,"中华人民共和国\n江苏省无锡市\n新区管委会");

// 插入下一页分节符

document.insertBreak(7, BreakConstants.BREAK\_SECTION\_NEXTPAGE);

// 获得第二个节对象

Section section2 = document.getSection(1);

// 获得第二个节的页面设置

WpPageSetup pagesetup = section2.getPageSetup();

// 设置横排页面打印方式

pagesetup.setOrientation(PrintConstants.HORIZONTAL);

// 设置页眉、页脚距正文的距离

pagesetup.setHeaderPosition(Utilities.centimeterToPoint(1.5f));

pagesetup.setFooterPosition(Utilities.centimeterToPoint(1.0f));

// 将页面设置的属性应用与第二个章节

section2.setPageSetup(pagesetup);

// 获得文档打印管理器，并打印全文

Print printer = document.getPrint();

printer.print();

通过Section对象我们得到本节的页面设置管理器。setOrientation(int oritation)方法可以设置本章节的页面排版方向，参数oritation代表页面方向，可以取以下两个常量：

PrintConstants.VERTICAL = 0 竖排方式

PrintConstants.HORIZONTAL = 1 横排方式

需要注意的是：页面方向为竖排还是横排与页面的宽度高度有关。竖排方式下的页面高度等于横排方式下的页面宽度，反之亦然。

SetHeaderPosition(float position)代表设置本节页眉上端与页面顶部之间的距离。setFooterPosition(float position)代表设置本节页脚下端与页面底部之间的距离。参数postion以磅值为默认的单位。这里需要提醒读者的是文字处理中所涉及到的度量单位一般都是以磅值作为默认的度量单位的，除非方法参数中作了特别的说明。对于中国用户更倾向于使用厘米作为基本的度量单位，这时我们可以调用工具类的度量单位转换方法将厘米转换成磅值。上述方法Utilities.centimeterToPoint(1.5)就是将1.5厘米转换为相应的磅值。Office还提供了其他的度量单位的转换，请参看工具类Utilities的说明。

最后再次提醒用户的是一定要将修改的属性应用于指定对象后，这个设置才起作用。例如：Section对象的setPageSetup(WpPageSetup pagesetup)方法，将页面设置应用于本节。

#### 5.5 页眉（Header）、页脚对象（Footer）

使用页眉和页脚，就可以在页面的顶端和底端打印页码、日期、公司徽标、文档标题、文件名或作者名等等经常要重复的信息。页眉和页脚使文档看起来更专业化一些，而页号则使你在杂乱不堪的工作环境中还能保持文档页不致混乱。页眉和页脚作为章节管理的编辑区域，在文档中显示在每个页面页边距的顶部和底部区域。

**获得Header、Footer对象**

最简单的是通过节对象的getHeader()方法获得当前章节的页眉区域，同样，getFooter()方法表示获得当前章节的页脚区域。以下示例获得第一个章节的页眉和页脚区域对象。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

Header header = section1.getHeader();

Footer footer = section1.getFooter();

通常情况下某章节每个页面的页眉页脚内容都是相同的，但是在打印双面纸的文档时，我们需要打印出来的页号出现在一页底端靠外的位置，也就是正反两页打印能够自动调整页脚内容显示在合适的位置。这时我们可以用奇偶页打印实现这样的功能。首先我们需要调整页面设置\ 版式\ 奇数页与偶数页不同的选项，然后得到某个章节奇数页或偶数页的页眉页脚对象，设置不同的内容，这时我们就可以看到奇偶页面不同的显示结果。

奇偶页不同选项可以通过页面设置管理器对象WpPageSetup的setOddAndEven(boolean isDifferent)方法进行设置，当参数为true时表示奇偶页不同。页面设置由节对象管理，每个章节可以修改不同的页面设置，我们通过节对象Section的WpPageSetup getPageSetup()方法得到当前章节的页面设置管理器。

一个章节最多可以存在三种不同的页眉或页脚区域，可以通过节对象Section的Header getHeader(int headerType)方法得到特定类型的页眉区域。参数headerType指的是页眉的类型，可以取以下的常量值：

HeaderFooterConstants.EVEN = 0; 表示获得偶数页的页眉

HeaderFooterConstants.ODD = 1; 表示获得奇数页的页眉

HeaderFooterConstants.FIRST\_PAGE = 2; 表示获得首页的页眉

有时候文档的首页作为一个封面或漂亮的标题页的情况，所以首页的页眉页脚内容可能与其他页不同，这时我们可以根据上面的常量得到首页的页眉或页脚区域。需要注意的是如果要求页眉首页与其他页不同，必须调整当前页面设置\ 版式\ 首页不同的选项。通过WpPageSetup的setFirstPageDifferent(boolean isDifferent)方法进行设置，当参数为true时表示首页不同。

以下示例获得第一个章节的奇数页的页眉对象。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

WpPageSetup pagesetup = section1.getPageSetup();

// 设置当前章节奇偶页不同

pagesetup.setOddAndEven(true);

// 获得当前章节奇数页的页眉

Header oddHeader = section1.getHeader(HeaderFooterConstants.ODD);

**使用Header、Footer对象**

页眉、页脚位于每个页面页边距的顶部或底部区域，作为六个文本编辑区域中的一个，能够满足插入文本、设置段落等平常的文本编辑操作。这些功能我们在正文对象中已经详加介绍，可参考BaseText对象。

#### 5.6 脚注（FootNote）、尾注对象（EndNote）

脚注和尾注主要用于在打印文档中为文档中的文本提供解释批注以及相关的参考资料。其中脚注出现在文档中每一页的底端用来对本页中的文档内容进行注释说明，而尾注一般位于整个文档的结尾，常用来注明文档中引用过的文献。

**获得FootNote、EndNote对象**

脚注、尾注在文档中并不是必备的。如果需要为文档中的文本提供解释批注以及相关的参考资料，则必须对正文中的某个文本区域引用一个脚注或尾注对象。此后便能对插入的脚注或尾注区域进行编辑操作。

为文本引用一个脚注可通过BaseText对象的FootNote addFootNote(long offset, long length, int applyTo)方法，返回当前插入的脚注对象。参数offset、length为引用文本的范围，参数applyTo代表脚注类型更改的应用范围，可以取以下几个值：

NoteConstants.APPLY\_TO\_WHOLE = 0; 将更改应用于整篇文档

NoteConstants.APPLY\_TO\_SECTION = 1; 将更改应用于本节

NoteConstants.APPLY\_TO\_TEXT = 2; 将更改应用于所选文字

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

BaseText baseText = section1.getBaseText();

baseText.insertText(0, "太湖美，美就美在太湖水\n");

FootNote footnote = baseText.addFootNote(0, 3, NoteConstants.APPLY\_TO\_WHOLE);

如果要对某一个脚注进行编辑，也可以通过节对象Section的getFootNote(int inex)方法返回引用的脚注对象，参数index指的是当前章节插入的第几个脚注。说明：我们把在某一个章节内插入的所有脚注作为一个集合通过节对象进行管理，集合的索引值是通过正文中插入脚注的注释引用标记的编号顺序排列的，也就是依据引用脚注文本在正文中的位置进行编号。尾注与脚注获取方式比较相近，这里不再重复解释，可参考脚注或相应的API文档。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

BaseText baseText = section1.getBaseText();

baseText.insertText(0, "太湖美，美就美在太湖水\n");

baseText.addFootNote(0, 3, NoteConstants.APPLY\_TO\_WHOLE);

FootNote footnote = section1.getFootNote(0);

**使用FootNote、EndNote对象**

脚注、尾注用于解释文本的批注及相关参考资料，是六个文本编辑区域中的一个，所以通常的文本编辑操作在脚注尾注中都能满足，如文本的操作，段落的操作，格式化边框属性等，可参考BaseText对象。

#### 5.7文本范围对象（TextRange）

用户在写代码的时候经常会遇到这样的烦恼，在正文对象BaseText或者其他编辑区域对象的方法时都需要传入位置信息的参数指定文本操作的范围。例如格式化某区域的字体我们调用BaseText对象的setFontAttribute(long offset, long length, FontAttribute fontAttribute)方法，而格式化段落的方法是setParagraphAttribute(long offset, long length, ParagraphAttribute paragraphAttribute)，这里都需要传入offset和length，表示将要格式化操作的文本范围。频繁地传递位置参数，降低了方法的便利性，因为我们可能需对同一个区域进行多种格式化操作。那么有没有什么办法来简化经常的操作呢？如果指定的文本范围封装了一个对象，那么我们调用方法就不需要再传递位置参数了，因为从文本范围对象中可以知道操作的范围。文本范围对象在Office中由TextRange对象表示。使用 TextRange 对象可以定义文档的某一部分，每一个 TextRange 对象由一起始和一终止字符位置定义。一个 TextRange 对象小至只是一个插入点大至包括整篇文档，所以使用是非常灵活方便的。

**获得TextRange对象**

TextRange对象代表在文档中的某一个文本范围，所以可以从Document对象中获得这个范围。TextRange getRange(long start, long end)代表定义文档的某一个文本范围。参数start、end代表定义范围在整个文档中的起始和终止位置。

定义TextRange对象的范围可大可小，小到只是一个插入点。这时将start、end参数设定成相同的数值即可。还有一个简便的方法，TextRange getRange(long offset)定义了一个某插入点的文本范围，参数offset指定这个插入点在文档中的位置。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

baseText.insertText(0, "中华人民共和国\n江苏省无锡市");

// 定义起始位置为2终止位置为5的文本范围对象

TextRange range1 = document.getRange(2, 5);

// 定义位置为10的插入点的文本范围对象

TextRange range2 = document.getRange(10);

通过Document对象的TextRange getSelectionRange()方法获得窗口当前选中的文本范围。如果当前窗口没有选中文本内容，则getSelectionRange()方法获得当前光标位置的文本范围。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Document document = workbook.getDocuments().addDocument();

BaseText baseText = document.getSection(0).getBaseText();

baseText.insertText(0, "中华人民共和国\n江苏省无锡市");

TextRange range = document.getSelectionRange();

Document对象中介绍的方法是在整个文档中定义某个文本范围，还可以相对于某个章节或者文本编辑区域定义一个TextRange对象。文字处理定义的六个文本编辑区域都提供了TextRange getRange(long start, long end)方法，代表相对于某文本编辑区域的起始位置定义一个文本范围对象，参数start、end指定范围相对于编辑区域的初始和终止位置。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Section section1 = document.getSection(0);

// 获得第一章节的页眉编辑区域

Header header = section1.getHeader();

header.insertText(0, "中华人民共和国\n江苏省无锡市");

TextRange range = header.getRange(2, 5);

TextRange对象的作用域是动态的，可以是一个选中的文本区域，也可以就是某一段落的内容。通过段落对象Paragraph的TextRange getRange()方法，可以获得当前段落所在的文本范围。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "中华人民共和国\n江苏省无锡市");

// 获得正文内所有段落的集合

Paragraphs paragraphs = baseText.getParagraphs();

// 获得第一个段落对象

Paragraph paragraph = paragraphs.getFirst();

// 获得当前段落所在的文本范围

TextRange range = paragraph.getRange();

如果填写表格的内容，必须指定表格的单元格，将内容填写入某单元格内。通过表格单元格Cell对象的TextRange getRange()方法，获得当前单元格的内容范围，也就是当前单元格内容在文档中所定义的文本范围。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 定义当前光标位置为一个文本范围

TextRange range = document.getSelectionRange();

// 获得表格集合

Tables tables = document.getTables();

// 在光标当前位置添加一个5行6列的表格对象

Table table = tables.addTable(range, 5, 6);

// 获得第二行第三列的单元格对象

// 得到本单元格内容的Range对象

TextRange range = cell.getRange();

这里有些代码读者可能不熟悉，表格是文字处理一个很重要的功能，有助于管理文档的内容，使文档内容更加简洁，条理更加清楚。我们将在后面介绍表格对象Table时详加讲解，请参阅相关章节。

通过TextRange对象的TextRange offset(long start, long end)方法可以根据当前文本范围对象进行偏移返回一个新的文本范围对象。新范围的偏移位置是这样计算的：新范围起始位置相对原文本范围的起始位置偏移参数start个单位，终止位置相对原文本范围的终止位置偏移参数end个单位。

以下示例定义一个文本范围对象，范围相对当前正文起始位置为0，终止位置为7，即文本“中华人民共和国”，然后再将这个范围进行偏移，返回一个新的文本范围对象。新范围的偏移位置是这样计算的：新范围起始位置相对原文本范围的起始位置偏移8个单位，终止位置相对原文本范围的终止位置偏移4个单位，返回的文本内容为“江苏省”。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "中华人民共和国\n江苏省无锡市");

// 定义起始位置为0终止位置为7的文本范围对象

TextRange range1 = document.getRange(0, 7);

// 根据当前文本范围对象进行偏移返回一个新的文本范围对象

TextRange range2 = range1.offset(8, 4);

**使用TextRange对象**

TextRange对象定义了文档中或某个编辑区域的某一个可进行文本操作的范围。之所以称之为文本范围对象是因为TextRange对象实现了文本编辑区域，如正文（BaseText）、页眉（Header）、页脚（Footer）等的文本操作及格式化的常用功能。我们可以通过定义一个TextRange对象对某个文本范围的内容进行文本及格式的修改。这样做的好处是您不需要在TextRange对象的文本操作方法中传入位置参数指定一个操作范围（BaseText等编辑区域对象操作文本时都必须传递位置参数指定范围），因为在您定义文本范围对象的时候，TextRange对象中已经自动获取了文本操作的范围。简化参数的传递、调用方法的便利，这也是TextRange对象相对于其他编辑区域对象最大的优势。

由于TextRange对象实现的是文本编辑区域文本操作及格式化的常用功能，这些功能在BaseText对象中已经详细介绍过了，我们这里只是例举两个简单的例子。如要使用请参看正文对象（BaseText）。

以下示例新建空白文档，插入文字，定义一个文本范围对象，范围起始位置为8，终止位置11，也就是“江苏省”这个文本内容范围，然后将范围内容设置为红色粗体，最后在范围的起始位置处添加文本。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "中华人民共和国\n江苏省无锡市");

// 定义起始位置为8终止位置11的文本范围对象

TextRange range = document.getRange(8, 11);

// 将文本范围内容“江苏省”的字体设置为红色粗体

FontAttribute fontAttr = new FontAttribute();

fontAttr.setBold(true);

fontAttr.setForeground(Color.red);

range.setFontAttribute(fontAttr);

// 在文本范围的起始位置添加文字

range.insertText("终于成立啦！");

代码range.setFontAttribute(fontAttr)代表将文本范围内容的字体设置红色粗体，方法并没有指定操作的范围，TextRange对象的默认操作范围为整个文本范围。如果方法只是指定一个插入点，那么TextRange对象指定文本范围的起始位置为插入点。例如例子中用到的代码range.insertText("终于成立啦！");代表在文本范围的起始位置添加文本。

TextRange对象指定一个文本的范围，具有强大的可伸缩性。如果在这个区域内部添加了文本，则TextRange对象的长度随之扩大；反之，在区域内部删除文本，TextRange对象长度随之缩小。可见TextRange对象是一个动态的文本范围。

以下示例新建文档，然后手工选中一段文本内容，运行宏的效果是将当前选中文本范围内容增加缩进一个制表位。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.getActiveDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "中华人民共和国\n江苏省无锡市");

// 返回手工选中的文本范围

TextRange range = document.getSelectionRange();

// 将文本范围的内容增加缩进一个制表位

range.increaseIndent();

#### 5.8 段落集合（Paragraphs）

段落集合由选定内容、范围或文档中的段落对象Paragraph组成的集合。管理着当前多个段落的添加删除等操作，Office中定义了一个集合对象Paragraphs。

**获得Paragraphs集合对象**

Paragraphs管理多个段落，可从正文、页眉、页脚、脚注、尾注和文本框这六个文本编辑区域内得到，代表相应的编辑区域内的所有段落的集合。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 获得正文内所有段落的集合

Paragraphs paragraphs = baseText.getParagraphs();

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 得到文档第一个节对象，新建的空白文档默认只有一个章节

Section section = document.getSection(0);

// 获得第一章节的页眉编辑区域

Header header = Section.getHeader();

// 获得页眉内所有段落的集合

Paragraphs paragraphs = header.getParagraphs();

也可以得到指定的文本范围内所有段落，这时从TextRange对象得到Paragraphs集合对象，代表这个文本范围内所有段落的集合。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市");

// 获得正文编辑区域指定的文本范围

TextRange range = baseText.getRange(2, 15);

// 得到这个动态文本范围内的所有段落的集合

Paragraphs paragraphs = range.getParagraphs();

**使用Paragraphs集合对象**

Paragraphs是一个集合对象，管理当前集合内所有段落的添加、删除、获取等集合操作。Paragraphs集合对象根据获取的来源不同，集合定义的范围也不一样。通过BaseText对象的Paragraphs getParagraphs()方法我们获得的是正文所有段落的集合，您也可以得到页眉、页脚等其他编辑区域的段落集合。

段落集合的灵活性在于它的范围是可变的，大到整个文档，小至某一个段落。您可以定义一个段落集合的范围。方法是从文本范围TextRange对象中获取段落集合Paragraphs对象。TextRange在之前已经详细介绍过，它的最大的优点在于范围的可伸缩性，一个 TextRange对象小至只是一个插入点大至包括整篇文档，所以从TextRange获得的段落集合Paragraphs对象也同样具有文本范围的特点。

TextRange对象代表文档中的一个范围。您可以使用TextRange 对象可以定义文档的某一部分。每一个TextRange 对象由一起始和一终止字符位置定义。在Document、BaseText或其他文本编辑区域中我们都提供了TextRange getRange(long start, long end)方法。甚至表格的某个单元格内容也是由TextRange对象定义的。

通过Paragraphs集合对象的Paragraph addParagraph(String content)方法，我们可以在指定区域的末尾添加新的段落，参数content为新段落的内容。如果不想添加在末尾，您可以使用Paragraph insertParagraph(int index, String content)方法将新段落插入在某一段落的前面。参数index指定新段落添加在哪一段的前面。参数content代表新段落的内容。说明：这个指定区域指的是Paragraphs集合对象的来源区域。如果从BaseText对象中获得，这个区域就是正文，如果从TextRange对象中获得，这个区域就是前面指定的文本范围。

以下示例在新建的空白文档中添加两个段落，然后在第一个段落的后面插入一个新段落。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

Paragraphs paragraphs = baseText.getParagraphs();

baseText.insertText(0, "中华人民共和国");

paragraphs.addParagraph("无锡市");

// 在"中华人民共和国"后插入一个新段落

paragraphs.insertParagraph(1, "江苏省");

如果删除文档中的某段文本，我们可以调用Paragraphs集合对象的deleteParagraphs(int startIndex, int endIndex)方法。参数startIndex、endIndex分别删除段落在集合内的起始索引和结束索引。这样我们就将集合所在的区域内的某些段落移除了。如果您只是删除某一个段落，可以调用deleteParagraph(int index)方法，删除指定的段落。参数index为指定段落在集合中的索引。我们可能删除指定区域内的所有段落，这时只要调用指定区域的段落集合对象deleteParagraphs()方法就可以了。

以下示例在新建的空白文档中插入多个段落，删除第一第二段落，然后打印删除前后正文段落总数的变化。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 插入多个段落的文本

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市");

Paragraphs paragraphs = baseText.getParagraphs();

paragraphs.deleteParagraphs(0, 1);

从集合中获取某个元素，段落集合就像其他集合对象一样可以从Paragraphs对象中获得指定的段落。Paragraph getParagraph(int index)代表获得集合指定的段落。参数index为指定段落在集合中的索引。通过Paragraph getFirst()方法可以得到集合的第一个元素，即首段落。Paragraph getLast()代表获得集合内的尾段落。

以下示例在新建的空白文档中插入多个段落，然后分别打印首段落、尾段落和第二个段落的文本信息。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 插入多个段落的文本

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市\n新区科技园\n");

Paragraphs paragraphs = baseText.getParagraphs();

// 获得首段落，并打印其文本信息

Paragraph firstParagraph = paragraphs.getFirst();

// 获得尾段落，打印其文本信息

Paragraph endParagraph = paragraphs.getLast();

// 获得第2个段落，打印其文本信息

Paragraph paragraph = paragraphs.getParagraph(1);

Paragraphs集合对象提供了改变段落位置的方法。moveDownParagraph(int startIndex, int endIndex)方法代表将集合内某一范围的段落向下移动。参数startIndex、endIndex指定了移动段落的范围。moveUpParagraph(int startIndex, int endIndex)代表段落范围向上移动。以下示例在新建的空白文档中插入多个段落，然后将前三个段落移至第四段落的末尾。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 插入多个段落的文本

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市\n新区科技园\n");

Paragraphs paragraphs = baseText.getParagraphs();

// 将第一至三段落移到第四段落的末尾

paragraphs.moveDownParagraph(0, 2);

前面我们曾经在BaseText正文对象中介绍过一些文本、段落的基本操作。一般方法都需要传入两个参数offset和length，代表操作数据的范围，这个范围指定一个文本区域。如果我们需要对整个段落或多个段落操作，就要知道操作范围的起始和终止位置，这样十分不便。我们在段落集合Paragraphs和段落对象Paragraph对象中封装了一些常用的文本操作方法，可以很容易地使用这些方法对文本或者段落进行处理。

例如increaseIndent(int startIndex, int endIndex)方法代表对集合内指定范围的段落增加缩进操作，参数startIndex、endIndex代表指定范围的段落起始索引及结束索引。以下示例在新建的空白文档中插入多个段落，然后对当前正文第一至第三段落增加缩进一个制表位。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 插入多个段落的文本

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市\n新区科技园\n");

// 获得正文编辑区域内的段落集合

Paragraphs paragraphs = baseText.getParagraphs();

// 对当前正文第一至第三段落增加缩进一个制表位

paragraphs.increaseIndent(0, 2);

以下示例在新建的空白文档中插入多个段落，然后将参数指定位置的文本范围内的所有段落设置为居中对齐。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

// 插入多个段落的文本

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市\n新区科技园\n");

// 得到指定位置的动态文本范围

TextRange range = baseText.getRange(8, 15);

// 获得这个文本范围内的段落集合

Paragraphs paragraphs = range.getParagraphs();

ParagraphAttribute paraAttr = new ParagraphAttribute();

// 设置段落对齐方式为居中对齐

paraAttr.setAlignment(ParagraphConstants.ALIGNMENT\_CENTER);

//将居中对齐属性应用于指定区域的段落集合

paragraphs.setParagraphAttribute(paraAttr);

#### 5.9 段落对象（Paragraph）

为了文档内容条理清晰，逻辑分明，我们按逻辑上的关系把文本的内容划分成一个个段落，每个段落以回车符\n作为段落的终止符，代表逻辑上独立的一段文本。Office中把Paragraph定义为段落对象。

**获得Paragraph对象**

段落对象由段落集合进行管理，所以从Paragraphs集合对象中可以得到指定的Paragraph对象。通过getFirst()可得到当前段落集合中的首段落，getEnd()得到的是尾段落，getParagraph(int index)得到参数index指定的段落，index从0开始计算。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市\n新区科技园\n");

Paragraphs paragraphs = baseText.getParagraphs();

// 得到首段落

Paragraph firstParagraph = paragraphs.getFirst();

// 得到尾段落

Paragraph endParagraph = paragraphs.getLast();

// 得到第二个段落，这时参数index为1

Paragraph secondParagraph = paragraphs.getParagraph(1);

有时侯我们并不知道指定段落的索引，Office提供了遍历的方法来查找到指定的段落。可以从当前的段落对象获取上一个或者下一个段落，以此遍历查找到满足条件的段落。Paragraph对象的Paragraph previousParagraph()方法代表获取当前段落的上一个段落对象，Paragraph nextParagraph()方法得到当前段落的下一个段落对象。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

Paragraphs paragraphs = baseText.getParagraphs();

**使用Paragraph对象**

段落作为文档内容逻辑上非常重要的一个对象，定义了很多有用的方法来进行段落的文本操作。主要有以下几个功能：1、访问段落的状态。2、段落的文本及格式化操作。3、制表位。下面我们一一介绍。

一、访问段落的状态。

段落是文本内容的一个逻辑部分，总是存在于某个文本编辑区域。我们可以通过文本编辑区域或者定义的文本范围对象获取指定的段落集合。段落集合管理当前区域内所有段落的集合操作，通过集合操作得到指定的段落，由此可以访问当前段落在集合中或区域中的位置信息。

int getIndex()返回当前段落在段落集合中的索引位置，以0为起始索引。

段落必然存在于某个文本编辑区域，我们可以通过Paragraph对象的long getStartOffset()和long getEndOffset()获得当前段落所在文本编辑区域的起始和结束位置。

以下示例在新建的空白文档中添加三个段落，然后在文档的页眉区域添加文本，分别打印正文和页眉区域中某段落的索引和位置信息。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

Section section = document.getSection(0);

baseText.insertText(0, "中华人民共和国");

// 得到正文部分的段落集合

Paragraphs baseParagraphs = baseText.getParagraphs();

// 添加两个段落

baseParagraphs.addParagraph("江苏省无锡市");

baseParagraphs.addParagraph("新区科技园");

// 得到第一章节的默认页眉

Header header = section.getHeader();

header.insertText(0, "永中公司");

// 得到页眉部分的段落集合

Paragraphs headParagraphs = header.getParagraphs();

// 得到正文和页眉的首段落

Paragraph para1 = baseParagraphs.getFirst();

Paragraph para2 = headParagraphs.getFirst();

访问段落的内容可以有两种方式。最简单的是调用Paragraph对象的String getText()方法，获取当前段落的文本内容，返回的是一个String对象。但是如果段落中还有图片，域等其他非文本的对象时，String getText()方法就不能返回完整的内容，我们可以通过TextRange getRange()方法返回当前段落的文本范围，通过TextRange这个强大的对象对文本内容各种访问和操作。通过TextRange getRange(long start, long end)方法中传入参数，我们可以得到段落中的某一范围。参数start、end为相对于当前段落的起始位置和结束位置。

以下示例在新建的文当中添加几段内容，然后打印第二段的内容，最后把第三段3位置后的文本格式化粗体。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

Section section = document.getSection(0);

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市\n");

Paragraphs paragraphs = baseText.getParagraphs();

// 得到第二段和第三段

Paragraph para2 = paragraphs.getParagraph(1);

Paragraph para3 = paragraphs.getParagraph(0);

// 高亮选中第二段

para2.getRange().select();

// 根据段落的起始位置定义一个文本范围，范围从当前段落3位置开始到段落结束位置

TextRange range = para3.getRange(3, para3.getLength());

FontAttribute font = new FontAttribute();

font.setBold(true);

// 将定义的范围格式化为粗体

range.setFontAttribute(font);

二、段落的文本及格式化操作。

对段落内的文本

以下示例在新建的文档中添加两个段落，然后改变第二个段落的段落属性，设置段落对齐方式为居中对齐，段前间距为1.5行倍距。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

Section section = document.getSection(0);

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市\n");

Paragraphs paragraphs = baseText.getParagraphs();

// 得到第二段

Paragraph para2 = paragraphs.getParagraph(1);

// 得到第二段的段落属性

ParagraphAttribute paragraphAttribute = para2.getParagraphAttribute();

// 设置段落对齐方式为居中对齐

paragraphAttribute.setAlignment(ParagraphConstants.ALIGNMENT\_CENTER);

// 设置段前间距为1.5行倍距

paragraphAttribute.setSpaceAboveOfline(1.5f);

// 将修改的段落属性应用于第二段落

para2.setParagraphAttribute(paragraphAttribute);

三、制表位。

文字中要设置空白间隔时，除了使用空格键外，还可以善用制表位的设置，迅速调整精确的显示位置。一行文字中可以设置多个制表位，当按下Tab键时，光标就会移到下一个制表位的位置。文件使用制表位后，可以任意修改制表位的位置，并设置文字在制表位的对齐方式。制表位共有5种对齐方式：

|  |  |  |
| --- | --- | --- |
| 符号 | 制表符类型 | 功能 |
|  | 左对齐制表符 | 文本左侧在制表符处对齐 |
|  | 右对齐制表符 | 文本右侧在制表符处对齐 |
|  | 居中制表符 | 文本在制表符处居中对齐 |
|  | 小数点对齐制表符 | 含有小数点的数字在制表符处对齐小数点 |
|  | 竖线对齐制表符 | 在制表符处插入一个短竖线 |

每个段落文字中都能设置多个制表位，用于控制文本显示的精确位置。一个段落中的多个制表位有一个集合对象进行管理，Office中定义管理当前段落制表位的集合为TabStops集合对象。TabStops集合对象管理着制表位的添加、删除、访问等操作，还能够设置默认的制表位位置。单个制表位在Office中也定义为一个对象TabStop。

1、TabStops集合对象的获取。

通过Paragraph对象的TabStops getTabStops()方法可以得到当前段落的制表位管理器TabStops集合对象。

2、TabStops集合对象的使用

通过这个集合对象我们可以在当前段落添加新的制表位或者删除某个制表位，改变制表位的位置等。

通过TabStops集合对象中的addTabStop(TabStop tabstop)方法，我们能够在当前段落中添加一个制表位到指定的位置，然后在需要定位的文本前按Tab键就能将文字精确定位到制表位指定的显示位置。制表位的设置，对于迅速调整文本的精确位置十分有效。

以下示例在新建的空白文档中添加多个段落，然后在第二段中添加一个制表位，制表位位置为5厘米，对齐方式为右对齐，前导符为点居中，这样我们在第二段按Tab键，光标能迅速定位到制表位指定的位置，即距离左页边距5厘米的位置。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

Section section = document.getSection(0);

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市\n");

Paragraphs paragraphs = baseText.getParagraphs();

// 得到第二段

Paragraph para2 = paragraphs.getParagraph(1);

// 获得第二段落的制表位集合

TabStops tabstops = para2.getTabStops();

// 初始化一个制表位对象，制表位位置为5厘米，对齐方式为右对齐，前导符为点居中

TabStop tabstop = new TabStop(Utilities.centimeterToPoint(5.0f), TabStopConstants.ALIGNMENT\_RIGHT, TabStopConstants.MIDDLEDOTS);

// 当前段落中添加一个制表位

tabstops.addTabStop(tabstop);

您也可以设置默认的制表位位置，改变按下Tab键后光标默认偏移的长度。例如文档默认制表位的位置为0.74厘米，setDefaultTabStop(float value)方法可以改变这个值，是默认制表位位置为指定的大小，参数value为默认制表位的位置，度量单位默认取磅值，您可以使用Utilities工具类转换自己需要的单位。以下示例改变默认制表位的位置为1.2厘米。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

Section section = document.getSection(0);

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市\n");

Paragraphs paragraphs = baseText.getParagraphs();

Paragraph para2 = paragraphs.getParagraph(1);

// 获得第二段落的制表位集合

TabStops tabstops = para2.getTabStops();

// 设置第二段默认制表位的位置为1.2厘米

tabstops.setDefaultTabStop(Utilities.centimeterToPoint(1.2f));

3、TabStop对象的获取

制表位对于文本精确定位十分方便，我们可以有两种方法获得制表位TabStop对象。一种是通过构造器，初始化一个制表位对象。

TabStop(float position, int alignment, int leader)

参数position指定制表位距页面左边距的精确位置，默认以磅值为度量单位，您可以使用Utilities工具类改变制表位位置的度量单位。

参数alignment代表制表位处文字在制表位的对齐方式。可以取以下五种常量：

TabStopConstants.ALIGNMENT\_LEFT = 0 左对齐

TabStopConstants.ALIGNMENT\_CENTER = 1 居中对齐

TabStopConstants.ALIGNMENT\_RIGHT = 2 右对齐

TabStopConstants.ALIGNMENT\_DOT = 3 小数点对齐

TabStopConstants.ALIGNMENT\_PERPENDICULAR = 4 竖线对齐

参数leader代表制表位前导符的样式，可以有以下几种样式供选择：

TabStopConstants.SPACES = 0 前导符 无

TabStopConstants.DOTS = 1 前导符 ...

TabStopConstants.DASHES = 2 前导符 ---

TabStopConstants.LINES = 3 前导符 \_\_\_

TabStopConstants.MIDDLEDOTS = 4 前导符 ...(点居中）

如果当前段落的制表位集合中已经添加过了制表位对象，我们也可以通过集合对象的get方法得到某一个制表位元素。TabStop getTabStop(int index)得到本段落指定的制表位对象。参数index代表制表位在集合中的索引，以0为起始索引值。

4、TabStop对象的使用

制表位是本段落中文本光标定位的一个位置标记，您可以使用制表位将文本或光标定位到精确指定的位置。并且这个位置信息随时都能改变以适应文本新的排版要求。

position属性代表制表位距页面左边距的精确位置，我们可以通过get、set方法访问或者修改这个位置。您也可以用字符作为度量单位修改制表位的位置。方法是调用setPositionOfChar(float position)，参数position指定制表位距页面左边距的字符数。alignment、leader属性我们在TabStop对象的构造器中已经详加阐述了，这里就不再介绍。总之我们可以修改制表位的属性来改变文本的排版。

以下示例在文档的第二段落中添加一个制表位，并修改制表位的位置为2字符，前导符为实线。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

BaseText baseText = document.getBaseText();

Section section = document.getSection(0);

baseText.insertText(0, "中华人民共和国\n江苏省\n无锡市\n");

Paragraphs paragraphs = baseText.getParagraphs();

Paragraph para2 = paragraphs.getParagraph(1);

// 获得第二段落的制表位集合

TabStops tabstops = para2.getTabStops();

// 初始化一个制表位对象，制表位位置为5厘米，对齐方式为右对齐，前导符为点居中

TabStop tabstop = new TabStop(Utilities.centimeterToPoint(5.0f), TabStopConstants.ALIGNMENT\_RIGHT, TabStopConstants.MIDDLEDOTS);

// 当前段落中添加一个制表位

tabstops.addTabStop(tabstop);

// 修改制表位位置为2字符

tabstop.setPositionOfChar(2.0f);

// 修改制表位前导符为实线

tabstop.setLeader(TabStopConstants.LINES);**5.10自由表格集合（Tables）**

表格使文档内容更加简洁，条理更加清楚。我们可以在文档中添加多个表格，管理当前文档中的所有表格，Office定义了一个集合对象Tables。

**获得Tables集合对象**

由于表格对象可以跨越多个章节，表格是文档级的功能模块，所以可以从文档对象Document获得表格的管理器Tables集合对象。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Tables tables = document.getTables();

**使用Tables集合对象**

Tables对象是一个集合对象，管理着当前文档中的所有表格对象。我们在文档中对表格的添加、删除等功能都是通过这个集合对象进行操作的。Table addTable(TextRange range, int row, int column)方法代表在文档中添加一个表格对象，参数row、column指定了添加表格的行数及列数。参数TextRange代表一个文本范围，指定了表格插入的位置，通常我们可从Document对象的TextRange getRange(long start, long end)方法获取指定位置的文本范围。如果需要在当前光标位置处添加一个表格，这时需要定义一个当前光标作用域的文本范围，可以从Document对象的TextRange getSelectionRange()得到这个范围。

以下示例在新建的文档光标指定位置处添加一个5行6列的表格对象。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

// 得到文字处理当前选取的文本区域，因为当前光标没有选中文本，所以我们得到光标指定的个位置，这个位置也是一个文本区域，起始结束位置是同一个

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

我们可以对添加的表格定义更多地要求，比如说可以给表格起一个名字，还可以指定添加表格的列宽，指定表格的布局等。

Table addTable(TextRange range, int row, int column, String tableName, int autoFit,

float cellWidth)

代表在文档指定文本范围内添加一张表格，参数定义了添加表格的一些默认属性。

参数range 代表表格插入的一个位置，这个TextRange对象指定了文档中的一个范围，如果range对象定义了一个光标位置，则在此位置处添加表格。如果range对象包含了多个文本的范围，则在插入表格前需先删除范围所含的内容。

参数row、column 表示添加表格对象的行数和列数。

参数tableName 为添加的表格定义了个名字，以便通过名字取得表格对象。

参数autoFit 代表添加的表格列宽是否要自适应文本内容，可以有下列三种类型供选择：

TableConstants.FIXED\_COLUMN\_WIDTH 代表表格的列宽是一个固定的度量值

TableConstants.AUTOFIT\_CONTENT 代表表格的列宽能够自适应文本的内容

TableConstants.AUTOFIT\_WINDOW 代表根据窗口调整表格的列宽

参数cellWidth 代表表格的固定列宽值，以磅值为单位，范围在0~1584磅之间。需要说明的是：这个参数值只在参数autoFit为TableConstants.FIXED\_COLUMN\_WIDTH固定列宽时才有效，如果autoFit为其他两个类型，方法addTable对此参数忽略。如果cellWidth为0，表示表格的列宽值设为自动，也就是说表格的列宽是根据插入表格时页面的宽度自动计算出来的。

以下示例在新建的文档光标指定位置处添加一个5行5列，固定列宽为50磅的表格对象，表格的名字取“我的表格”。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 5, "我的表格",

TableConstants.FIXED\_COLUMN\_WIDTH, 50.0f);

我们可以从Tables集合中获得某一个表格对象。Table getTable(int index)返回一个表格对象，参数index是表格的索引值，代表表格在文档中插入的先后顺序。以下示例获得文档第一个表格对象，index取0代表java的集合对象都是从0开始计算索引的。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Tables tables = document.getTables();

// 在参数指定的文本范围的起始位置插入一个5行6列表格对象

tables.addTable(document.getSelectionRange(), 5, 6);

Table table1 = tables.getTable(0);

同样我们也可以通过表格的名字得到指定的表格对象。以下示例在新建的文档中添加一个表格，然后选中这个表格对象。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Tables tables = document.getTables();

TextRange range = document.getRange(0, 0);

tables.addTable(range, 5, 6,"表格1");

Table table = tables.getTable("表格1");

table.select();

通过Tables集合对象的removeTable(int index)方法我们可以移除集合内的表格对象，参数index代表表格对象在集合中的索引，以插入表格的先后顺序排列。以下示例在新建的空白文档中添加表格，然后打印全文的表格数，接着删除这个表格，再次打印表格数，与删除前进行比较。要获得全文的表格数，可调用Tables集合的int getCount()方法，返回当前文档的表格总数。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

tables.removeTable(0);

Tables集合对象还能够控制当前文档中所有表格共有的一些状态。有时候处理一些文档时可能会有一些特别的需要。比如，我们可能不希望别人随意删除文档的表格，在Tables集合对象中提供了方法setAllowDeleted(boolean deleted)控制文档表格是否可被删除的状态。参数delete表示可以删除表格对象。这是我们文档默认的属性，如果不希望用户删除表格，可以将参数设为false。同样方法setAllowSelected(boolean seleted)表示集合中的表格能否被用户选中，如果将参数设为false，则不能对文档中的所有表格及表格的内容进行任何的操作。相当于表格成了只读的不可编辑的对象。

以下示例在新建的空白文档中添加一张5行6列的表格，并将表格设置为不可删除状态。此时如果调用API删除表格，则抛出不能删除表格的异常反馈给用户。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

tables.setAllowDeleted(false);

// 此时不能删除表格，调用删除表格方法抛异常

tables.removeTable(0);

#### 5.11表格对象（Table）

利用表格可以组织和显示需要分门别类的信息。与普通文本相比，表格将使文档内容更加简洁，条理更加清楚。如果可以将某些冗长的文本用精美的表格来代替，必定能给文档的版面外观带来很好的修饰效果。永中Office 提供了强大的表格功能，用户不仅能够利用这些功能非常方便地制作出各种形式的表格，还可以完成诸如排序计算等工作。

**得到Table对象**

如果已经用API在文档中添加了一个表格对象，那么最方便地方法是直接使用addTable返回的表格对象。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Tables tables = document.getTables();

TextRange range = document.getRange(0, 0);

Table table = tables.addTable(range, 5, 6);

此外通过Tables集合对象的Table getTable(int index)方法可以返回指定的表格对象。参数index为表格的索引值，代表表格在文档中插入的先后顺序。

通过表格的名字获取表格对象不失为一个好办法，前提是你必须为添加的表格定义了个名字。Table getTable(String name)方法代表获取指定名字的表格对象。如果没有为表格指定名字，表格返回一个默认规则的名字，新创建的表格以“表格1”、“表格2”依次编码。

以下示例提供两种办法获得表格对象。一种是根据索引获得，另一种根据名字获得。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

Tables tables = document.getTables();

TextRange range = document.getRange(0, 0);

tables.addTable(range, 5, 6, "我的表格");

// 根据索引获得第一个表格对象

Table table1 = tables.getTable(0);

// 根据名字获得表格对象

Table table2 = tables.getTable("我的表格");

// 判断这两个表格是否是同一个对象

**使用Table对象**

默认情况下，我们添加一个表格对象并不指定它的名字。如果我们以默认的名字获取表格的话将会比较困难，因为默认的表格名字不容易辨别。可以有两种办法给表格添加一个名字。

一、在Tables集合对象中我们提到过addTable方法传入名字参数可以给添加的表格定义一个名字。二、我们可以给获得的表格对象重命名，Table对象定义了setName(String name)方法实现上述的功能。参数name就是表格的名字。String getName()方法可以获得当前表格的名字。

以下示例新建一空白文档，然后在光标位置处插入一个5行6列的表格对象，打印当前表格的默认名字。接着可以对这个表格重新命名，此时我们可以看到表格的名字被改变了。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

table.setName("我的表格");

通过Table对象的select()方法，我们可以选中整个表格。以下示例选择新添加的表格对象。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

table.select();

实际应用过程中往往需要对最初建立的表格结构进行一定程度的调整。例如需要插入或者删除某些行列或单元格等。在永中Office中建立空表格之后可以随时对表格的结构进行修改。

横向的同一排单元格组成了一个表格行对象，纵向的同一列单元格的集合就是一个表格列对象。在文字处理中Row对象定义为表格的行对象，表示同一行单元格的集合。Column对象定义为表格的列对象，表示同一列单元格的集合。我们可以通过Row或Column对象同时对某一行或某一列进行操作。一个表格从数据结构上看可以定义为一个二维数组。横向由若干个行对象组成一个行集合，Office中定义了Rows集合对象为行对象的集合，管理着若干个表格行Row的添加删除等操作。同样从纵向看表格又是由若干个列对象组成，Office中定义了Columns集合对象为列对象的集合，管理着若干个表格列Column的操作。请读者注意：Rows或Columns集合对象只是定义了Row或Column对象的一个集合，这个集合的大小是由用户根据参数指定的。您可以不传任何参数从表格对象Table中获取整个表格的所有行的集合，也可以通过参数定义一定范围的表格行集合。列集合同样处理。

Rows getRows()

代表获取当前表格所有行的一个集合。

Rows getRows(int startRow, int endRow)

代表获取指定范围的表格行的集合。参数startRow、endRow指定了行集合的在整个表格中的范围。startRow表示范围的在表格中的起始行号，endRow为结束行号。需要说明的是：这里的参数是以1为起点进行计算的，而以前的方法默认都是以0为初始值。这么做的原因是方便用户对表格的定位操作。我们习惯上定位表格单元格时都是从1开始数的，比如第一行第一列，第二行第三列，这样方法就与用户的习惯保持一致。请读者一定注意了：如果方法说明中没有指定参数从1开始的话，所有参数不管是位置还是索引值都是默认以0为起始位置。

以下示例新建文档中添加5行6列的表格，然后定义一个行集合对象，范围从第二行到第四行。我们取集合第二个对象，也就是表格的第三行对象，选择这个行。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

// 定义行集合的范围为第二行到第四行

// 获取集合第二个行对象，在表格中位于第三行

Row row = Rows.getRow(1);

row.select();

Rows集合对象定义了表格行的添加删除等集合操作，同样Columns集合对象定义了表格列的集合操作。通过Rows集合对象的Row addRow(Row beforeRow, int direction)方法我们在指定行的上方或下方添加了一个新行，方法返回新行对象。参数beforeRow代表指定的行对象，我们将以这一行为基准位置添加新的行。参数dirction代表插入行的方向。

TableConstants.INSERT\_ROW\_ABOVE 在指定行的上方插入新行

TableConstants.INSERT\_ROW\_BELLOW 在指定行的下方插入新行

与其他的集合对象一样，我们还能通过remove方法执行删除操作。removeRow(int index)删除行集合中指定的行对象。参数index指定将要删除的行在集合中的索引。

删除表格中的对象还有一个简便的方法就是直接调用这个对象的delete()方法。例如：假设我们已经得到了某个行对象Row。通过Row对象的delete方法将当前对象在行集合中移除掉。

通过Rows集合对象的getRow(int index)方法，我们可以得到指定的行对象。参数index代表指定行在集合中的索引值。这些操作在其他集合中基本也会碰到，我想读者应该是很熟悉的了，这里不再详述。

以下示例在新建的空白文档中添加一个5行6列的表格对象，然后在第四行的上方插入一个行对象。此时，这个新添加的行对象在表格行中的位置为第四行，而原来第四行变成了第五行的行对象，后面的行以此类推，每个行号递增1。最后我们将新添加的行高亮选中。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

// 得到当前表格所有行的集合

// 得到表格第四个行对象，参数为3是因为我们从集合中获取对象，索引都是从0开始计算的

Row beforeRow = rows.getRow(3);

// 在第四行的上方插入一个新行，方法返回了新行的行对象

Row newRow = rows.addRow(beforeRow, TableConstants.INSERT\_ROW\_ABOVE);

// 选中新行

newRow.select();

前面我们介绍了怎么添加一个表格，并且对最初建立的空表格结构进行一定程度的调整。例如，需要插入或者删除某些行列或单元格等，建立空表格之后可以随时对表格的结构进行修改。下面我们来填写表格的内容。表格的数据是存储在单元格中的。单元格对象是表格最基层的对象，Office中定义了Cell对象来表示单个单元格。我们可以通过Table对象的Cell getCell(int row, int column)方法得到指定行列号的单个单元格对象。参数row，column指定了单元格在表格中的行号和列号。与前面介绍的Rows getRows(int startRow, int endRow)方法一样，这里的参数也是从1开始计算的。因为这更符合我们平时的习惯，我们很容易通过行列号找到对应的单元格。

单元格的内容是非常丰富的，可以在单个单元格中放置文本、段落、图片甚至域代码等复杂的对象。用户已经可以感觉到单元格虽小，但是功能却五脏俱全，只要普通文本编辑区域所具有的功能，它也都能实现。为了方便操作单元格的内容，我们在Cell对象中定义了一个非常有用的方法TextRange getRange()，方法返回当前单元格的文本区域对象。这个对象用于存储单元格的内容。通过TextRange对象我们可以很便捷地进行各种文本操作，就像前面介绍的在正文编辑区域中定义的动态文本范围TextRange对象一样。

以下示例在新建的空白文档中添加一个5行6列的表格，然后在表格的第二行第三列的单元格内添加“this is a test!”这段文本。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

// 获得第二行第三列的单元格对象

// 得到本单元格的Range对象，然后添加文本

TextRange range = cell.getRange();

range.insertText("this is a test!");

刚才我们只是填充了某一个单元格的内容，如果要填充的是一定范围的单元格区域，可以使用循环指令。下面示例新建一个5行6列的表格，然后填充这个表格中每个单元格的内容。这里单元格的内容为各个单元格的行列号。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

// 得到整个表格的行集合

Rows rows = table.getRows();

// 使用循环指令填充整个表格的内容

for (int i = 0; i <rows.getCount(); i++)

{

// 获取每一行

Row row = rows.getRow(i);

// 这里获得的是某一行的所有单元格的一个集合

Cells cells = row.getCells();

// 遍历行对象中的一个个单元格对象

for (int j = 0; j <cells.getCount(); j++)

{

Cell cell = cells.getCell(j);

// 填充单元格的内容

Cell.getRange().insertText("本单元格行号为"+ j + "列号为" + j);

}

}

读者可能会对上面的代码有些疑惑。上述代码使用了一个新的对象Cells。表格中一定范围的单元格区域在Office中有一个专有的对象来表示，这就是Cells。Cells对象定义了一个单元格的范围，是一个集合对象。Cells想其他集合对象一样，管理着单元格的添加、删除、获取等操作。我们可以通过以下几个途径得到Cells集合对象。

Table对象提供了一个方法Cells getSelectingCells()，代表获取当前选取的单元格区域，如果当前表格没有单元格区域被选择，则返回null。

单元格是表格逻辑上最小的单位，所以行、列也是由单元格组成的。可以通过Row对象的Cells getCells()方法得到这个行的单元格集合，当然Column中也能得到。

单元格集合Cells对象中可以得到指定的单元格，方法是Cell getCell(int index)，参数index表示指定的单元格索引。我们查找单元格时是按行进行遍历的，所以Cells集合中指定的单元格索引也是按行进行遍历的。比如我们定义一个2行3列的单元格范围，方法getCell(4)指的是获取第2行第2列的单元格对象，索引是从0开始计算的，也就是集合第5个单元格。

为了使表格的结构更加合理外观更加精美常常需要对表格进行格式化。格式化表格包括格式化表格本身和格式化表格内容两部分。对单元格中的文本内容进行格式化时方法与格式化普通文本相同。因此我们这里着重介绍的将是对表格本身的格式化。表格本身的格式化属性按照表格属性对话框的选项卡分可以分成4大类，每一类构成一个属性集。下面依次介绍这4个属性集。

表格属性集------代表应用于整个表格的表格属性。Office中用TableAttribute对象定义了一个表格属性集。通过表格属性可以设置表格的宽度、表格在页面中的对齐方式以及文字环绕表格的方式等。

行属性集------代表应用于某一表格行或行集合的属性。行属性集包含在本行或行集合的Row对象或Rows集合对象中。可通过其中的set、get方法设置或者获取相应的行属性，例如表格的行高，首行内容以标题行形式出现在下一页等。

列属性集------代表应用于某一表格列或列集合的属性。列属性集包含在本列或列集合的

Column对象或Columns集合对象中。同样可通过其中的set、get方法设置或者获取相应的列属性，例如表格的列宽。

单元格属性集------代表应用于某一单元格或单元格范围的属性。Office中用CellAttribute对象定义了一个单元格属性集。通过单元格属性可以调整单元格的尺寸、设置单元格内文字的垂直对齐方式，还可调整单元格内文字与单元格顶端、底端、左侧或右侧边缘的距离。

以下示例在新建的空白文档中添加表格，然后设置表格的一些属性，将表格设置为右对齐方式，允许表格重调整列宽以适应内容等。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

// 获得整个表格的表格属性

TableAttribute tableAttribute = table.getTableAttribute();

// 设置表格在页面中的对齐方式为右对齐

tableAttribute.setAlignment(TableConstants.TABLE\_RIGHT);

// 设置允许表格的列宽重调整尺寸以适应内容

tableAttribute.setAllowAutoFit(true);

// 将改变的属性应用于表格

table.setTableAttribute(tableAttribute);

TableAttribute对象的alignment属性代表表格在页面中的对齐方式。有三种对齐方式供选择，分别是：TableConstants.TABLE\_LEFT = 0 左对齐

TableConstants.TABLE\_CENTER = 1 居中对齐

TableConstants.TABLE\_RIGHT = 2 右对齐

代码tableAttribute.setAlignment(TableConstants.TABLE\_RIGHT);表示设置alignment属性为右对齐，如果要得到表格的对齐方式可以属性相应的get方法int getAlignment()。

allowAutoFit属性代表是否允许表格的列宽重调整尺寸以适应内容。就像以前我们介绍的属性集一样，你必须将改变的属性应用于相应的对象才能使设置有效。代码调用Table对象的setTableAttribute(TableAttribute tableAttribute)方法将更改属性应用于表格。

以下示例在新建的文档中添加5行6列的表格，然后修改第一行的行高为2厘米，修改第一行各单元格的宽度为2.5厘米。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

// 获得表格所有行的集合

Rows rows = table.getRows();

// 从集合中获得首行

Row row = rows.getRow(0);

// 设置首行的行高为2厘米

row.setRowHeight(Utilities.centimeterToPoint(2.0f));

// 获得某行的单元格集合

Cells cells = row.getCells();

// 获得某行的单元格属性

CellAttribute cellAttribute = cells.getCellAttribute();

// 设置单元格宽度的度量类型为绝对值

cellAttribute.setPreferredWidthMeasure(TableConstants.CELL\_WIDTH\_MEASURE);

// 设置单元格宽度为2.5厘米

cellAttribute.setPreferredWidth(Utilities.centimeterToPoint(2.5f));

// 将修改的属性应用于单元格集合

cells.setCellAttribute(cellAttribute);

行高作为表格行对象的一个属性而存在的，我们通过Row对象的rowHeigth属性可以设置或取得当前行的高度。setRowHeight(float height)代表设置本行的行高，参数height表示行的高度，默认以磅值为度量单位。我们可以通过Utilities对象的centimeterToPoint方法将厘米转换为磅值。

后面的代码将本行的所有单元格设置为同样的宽度。从Row对象中我们可以得到所有的单元格，即单元格集合Cells。然后改变这个单元格集合的单元格宽度属性，即可以更改每个单元格的宽度。单元格的宽度值与单元格宽度的度量类型有关。如果度量类型取固定值，则宽度值可以取磅值、厘米、英寸等。如果度量类型取百分比，则宽度值为一个相对值，实际大小为页面宽度乘以百分比值。

CellAttribute对象的setPreferredWidthMeasure(int measure)方法，代表设置指定单元格宽度的度量方式。参数measure为度量类型，可以取以下常量：

TableConstants.CELL\_WIDTH\_NONE\_SET 不指定单元格宽度

TableConstants.CELL\_WIDTH\_MEASURE 指定宽度为固定值

TableConstants.CELL\_WIDTH\_PRECENT 指定宽度为百分比

CellAttribute对象的setPreferredWidth(float width)方法，代表设置指定单元格的宽度。参数width依据度量类型不同，单元格宽度值的语义也不一样。如果，度量类型为固定值，则返回单元格宽度为磅值，范围在0~1584磅。如果度量类型为百分比，则返回单元格宽度是一个相对值，范围在0~100%。

边框和底纹可以有效地美化文档版面的外观。Office支持表格边框的重新定义。我们可以通过BorderAttribute属性集对表格的边框或某个单元格范围的边框进行样式格式化。在前面BaseText对象的介绍中我们曾经提到过BorderAttribut对象在Office中是作为边框属性集存在的。BorderAttribute对象不仅能够应用于文字、段落，还能应用于表格。表格中的边框属性BorderAttribute对象有多种途径得到。Table对象的BorderAttribute getBorderAttribute()方法可以得到整个边框的边框属性。同样如果要获得某一行或列的边框属性，可以从相应的Row或者Column对象中获取。Cells集合对象表示某个单元格的区域，如果我们要修改这个单元格范围的边框，可以从此对象中调用BorderAttribute getBorderAttribute()方法获得边框属性，然后将修改的属性应用于这个单元格范围即可。BorderAttribute对象在以后的文字处理常用属性集的介绍中会有详细的讲解，这里只是例举一个例子。

以下示例添加5行6列的表格，然后改变整个表格边框的属性。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

// 获取整个表格的边框属性

BorderAttribute borderAttr = table.getBorderAttribute();

// 设置表格上、下、左、右四条外边框的类型为线型2

borderAttr.setBorderLineType(BorderAndShadingConstants.TOP, 1);

borderAttr.setBorderLineType(BorderAndShadingConstants.BOTTOM, 1);

borderAttr.setBorderLineType(BorderAndShadingConstants.LEFT, 1);

borderAttr.setBorderLineType(BorderAndShadingConstants.RIGHT, 1);

// 设置表格上、下、左、右四条外边框的颜色分别为红、黄、蓝、绿

borderAttr.setBorderLineColor(BorderAndShadingConstants.TOP, Color.RED);

borderAttr.setBorderLineColor(BorderAndShadingConstants.BOTTOM, Color.YELLOW);

borderAttr.setBorderLineColor(BorderAndShadingConstants.LEFT, Color.BLUE);

borderAttr.setBorderLineColor(BorderAndShadingConstants.RIGHT, Color.GREEN);

// 设置表格上、下、左、右四条外边框的宽度为1、2、2、1厘米

borderAttr.setBorderLineWidth(BorderAndShadingConstants.TOP, Utilities.centimeterToPoint(1.0f));

borderAttr.setBorderLineWidth(BorderAndShadingConstants.BOTTOM, Utilities.centimeterToPoint(2.0f));

borderAttr.setBorderLineWidth(BorderAndShadingConstants.LEFT, Utilities.centimeterToPoint(2.0f));

borderAttr.setBorderLineWidth(BorderAndShadingConstants.RIGHT, Utilities.centimeterToPoint(1.0f));

// 将修改过的边框属性应用于表格

table.setBorderAttribute(borderAttr);

使用“自动套用格式”快速制作表格，达到专业化的效果。Table对象的setTbleStyle(int index)方法您可以使用表格模板定义好的格式快速地美化表格。参数index代表定义好的表格模板的索引值。您可以通过表格自动套用格式对话框的预设格式选项来查找需要的模板。

以下示例添加5行6列的表格，然后自动套用表格格式为“彩色型1”，即对应于预设格式第二个选项，索引取1，从0开始计算。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents documents = workbook.getDocuments();

Document document = documents.addDocument();

TextRange range = document.getSelectionRange();

Tables tables = document.getTables();

Table table = tables.addTable(range, 5, 6);

// 将表格的预设格式设置为“彩色型1”

table.setTableStyle(1);

#### 5.12书签管理器 (Bookmarks)

在文字处理中,书签管理器s.Bookmarks这个类在文字处理中非常重要,它包括了对当前章节中插入书签，定位和删除当前文档中书签,修改书签的属性,书签名称排列的方式,是否显示隐藏书签的一些方法。

先请看以下关于书签管理器操作的代码。

示例1 ：在文档中添加一个书签。

Documents documents = Application.getWorkbooks().getActiveWorkbook().getDocuments();

Document doc=documents.getActiveDocument();

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

baseText.insertText(0,"this is a test!");

BookMarks bookmarks = doc.getBookmarks();

bookmarks.add(baseText.getRange(5, 10),"书签1");

我们在先前的类层次图上可以了解如何得到BookMarks这个对象.由于在增加书签时,首先通过BaseText类中的方法来添加文字.然后通过Range对象来获得添加书签所需要的位置.如何获得这个Range对象,在Range类中有详细的介绍。

以下列举了在Bookmarks 对书签基本操作的一些API。

BookMark add(TextRange range, String name)

在文档中添加一个书签。

参数:

range 是指书签的位置

name 是书签名

Void locateBookmark(String bookmarkName)

将光标定位到指定的书签。

BookMark get(String bookmarkName)

返回指定的书签。

Documents documents = Application.getWorkbooks().getActiveWorkbook().getDocuments();

Document doc=documents.getActiveDocument();

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

baseText.insertText(0,"this is a test!");

//得到书签管理器

BookMarks bookmarks = doc.getBookmarks();

bookmarks.add(0,2,baseText,"书签1");

//得到书签名为"书签1"的书签

BookMark bookmark = bookmarks.get("书签1");

//打印该书签名

Void setShowHide(boolean hide)

设置是否显示隐藏书签。

Boolean isShowHide()

返回是否显示隐藏书签

Static void setSortType(int type)

设置书签名称排列的方式。

参数:

Type 0 : 按名称排列

1 : 按位置排列

Static int getSortType()

返回书签名称排列的方式。

BookMark[] getAllBookmarks()

返回当前文档内所有书签名的一个字符串集合。

Void remove(String bookmarkName)

在当前文档中删除指定的书签。

Void removeAll()

在当前文档中删除所有的书签。

String[] getBookmarkNames()

返回当前文档内所有书签名的一个字符串集合。

下面提供一个完整的示例代码，表示更改当前工作表名称，并给单元格设置以数组形式表示的数据

//样例4.2

import application.Application;

public class MyTest

{

public MyTest()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.WORDPROCESSOR);

}

public void process()

{

Documents documents = Application.getWorkbooks().getActiveWorkbook().

getDocuments();

Document doc=documents.getActiveDocument();

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

//插入文字

baseText.insertText(0,"this is a test!");

//得到书签管理器

BookMarks bookmarks = doc.getBookmarks();

//添加书签

bookmarks.add(baseText.getRange(5, 10),"书签1");

//书签是否隐藏

/ /删除书签

bookmarks.remove("书签1");

}

public static void main(String[] args)

{

MyTest myTest = new MyTest();

myTest.process();

}

}

**书签 (Bookmark)**

在文字处理中,书签管理器bookmarks.BookMark

在这个类中,是对已经定义好的书签,进行一系列的操作.比如设置书签的开始与结束位置,设置书签的内容等等操作.

先请看以下关于书签操作的代码。

示例1 ：在文档中设置书签的开始位置

Documents docs = Application.getWorkbooks().getActiveWorkbook().getDocuments();

Document doc=docs.tActiveDocument();

//得到第一个节

Section section = doc.getSection(0);

//得到正文

BaseText baseText = section.getBaseText();

baseText.insertText(0,"this is a test!");

//得到书签管理器

BookMarks bookmarks = doc.getBookmarks();

bookmarks.add(10,5,baseText,"书签1");

BookMark bookmark = bookmarks.get("书签1");

//设置书签的起始位置

bookmark.setStartPosition(1);

在这段代码中,简单的介绍了如何设置书签的起始位置.首先要插入一些文字,然后在书签管理器中添加一个名为"书签1"的标签,然后通过setStartPosition(1)这个方法,设置书签的位置.为检验这个方法有效,同时添加了一个getStartPosition()这个方法,来检验set方法是否有效.

以下列举了在Bookmarks对书签基本操作的一些API。

public long getEndPosition()

返回书签的结束位置

public void setEndPosition(long offset)

设置书签的结束位置。

参数:

office 书签的结束位置

public String getName()

返回书签的名字

public String getValue()

得到当前书签所包含的内容。

public void setValue(String value)

修改当前书签的内容.

参数:

value 要修改的内容

public void setValue(String value, FontAttribute fontAttribute)

以指定字体修改当前书签的内容.

参数:

value 要修改的内容

fontAttribute 指定修改内容的字体

下面提供一个完整的示例代码，表示更改当前工作表名称，并给单元格设置以数组形式表示的数据

import application.Application;

public class MyTest

{

public MyTest()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.WORDPROCESSOR);

}

public void process()

{

Documents docs = Application.getWorkbooks().getActiveWorkbook().

getDocuments();

Document doc =docs.getActiveDocument();

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

baseText.insertText(0,"this is a test!");

BookMarks bookmarks = doc.getBookmarks();

bookmarks.add(5,5,baseText,"书签1");

FontAttribute fontAttribute = new FontAttribute();

//设置字体属性

fontAttribute.setFontSize(24.0);

fontAttribute.setBold(true);

fontAttribute.setForeground(Color.red);

BookMark bookmark = bookmarks.get("书签1");

bookmark.setValue("书签新名字", fontAttribute);

}

public static void main(String[] args)

{

MyTest myTest = new MyTest();

myTest.process();

}

}

#### 5.13修订管理器(Revisions)

在文字处理中对应的类:

在这个类中主要实现了修订得一些基本的方法,包括开始修订,停止修订,接受修订,决绝修订等一些访法,同时它还对修订面板的设置提供了一些必要的方法,包括修订的颜色,修订时新插入文本的颜色,标记新插入文本的显示格式,标记被删除文本的颜色等等一些方法.这些方法的提供更方便用户对修订管理器的使用.

先请看以下关于书签管理器操作的代码。

示例1 修订的一些基本方法

Document doc= Application.getWorkbooks().getActiveWorkbook().

getDocuments().getActiveDocument();

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

baseText.insertText(0,"this is a test!");

baseText.insertText(baseText.getTextLength(),"开始修订");

Revisions revisions = doc.getRevisions();

doc.beginTrackRevision(); //修订开始

baseText.insertText(baseText.getTextLength(),"开始修订后插入文字");

revisions.acceptAll(); //接受所有的修订

doc.stopTrackRevision();//停止修订

baseText.insertText(baseText.getTextLength(),"停止修订后插入字符串");

在这个例子中,它介绍对修订控制的一些方法.包括修订的开始,到修订的结束,以及接受修订.

修订的基本的API方法的介绍：

public void acceptAll()

跟踪变化时，接受所有修订

public void rejectAll()

跟踪变化时，拒绝所有修订

示例2 对修订面板的一些设置

Document doc = Application.getWorkbooks().getActiveWorkbook().

getDocuments().getActiveDocument();

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

baseText.insertText(0,"this is a test!");

Revisions revisions = doc.getRevisions();

doc.beginTrackRevision();//修订开始

baseText.insertText(0,"new1");

//标记新插入文本的颜色

revisions.setInsertColor(TrackChangesConstants.BRIGHT\_GREEN);

baseText.insertText(5,"new2");

这个示例中主要是在修订开始后对标记新插入的文本的颜色的设置,通过不同的index的值来设置颜色,同样也可以通过get方法得到的index的值.

修订面板API方法基本介绍：

public void setInsertColor(int index)

设置标记修订时，标记新插入文本的颜色

参数:

Index 颜色的索引值，以下列举颜色的索引值：

BY\_AUTHOR 颜色（根据作者）

AUTOMATIC 颜色（自动）

BLACK 黑色

BLUE 蓝色

TURQUOUISE 青绿色

DARK\_GRAY 深灰色

GRAY 灰色

BRIGHT\_GREEN 亮绿色

LIGHT\_GRAY 浅灰色

MAGENTA 洋红色

GOLD 金色

PRINK 粉红色

RED 红色

WHITE 白色

YELLOW 黄色

DARK\_BLUE 深蓝色

TEAL 青色

GREEN 绿色

VIOLET 紫罗兰色

DARK\_RED 暗红色

DARK\_YELLOW 暗黄色

public int getInsertColor()

获得修订时新插入文本的颜色

public void setFormatMark(int index)

设置标记修订时，标记改变过格式的文本的显示格

参数:

Index 颜色的索引值，以下列举颜色的索引值：

NONE 无标志

ONLY\_COLOR 仅颜色

BOLD 加粗标志

ITALIC 斜体标志

SOLID\_UNDERLINE 下划线标志

DOUBLE\_UNDERLINE 双下划线标志

public int getFormatMark()

获得修订时改变过格式的文本的显示格式

public void setChangeMark(int index)

设置标记修订时，标记修订行的显示格式

参数:

Index 颜色的索引值，以下列举颜色的索引值：

NONE 无标志

LEFT\_BORDER 左侧框线

RIGHT\_BORDER 右侧框线

OUTSIDE\_BORDER 外侧框线

public int getChangeMark()

获得修订时修订行的显示格式

public void setChangeColor(int index)

设置标记修订时，标记修订线颜色

参数:

Index 颜色的索引值

public int getChangeColor()

获得修订时修订线颜色

public void setCommentVisible(boolean visible)

设置修订时是否显示批注

参数:

Visible true 显示

False 不显示

public void setCommentViewType(int type)

设置批注框的视图类型

参数:

Type 批注框的视图类型，只能取以下常量：

TrackChangesConstants.ALWAYS 总是

TrackChangesConstants.NEVER 从不

TrackChangesConstants.ONLY\_FOR\_COMMENT\_FORMATTING 仅用于批注/格式

public int getCommentViewType()

得到批注框的视图类型

public void setCommentAlignment(int alignment)

设置批注框的边距类型

参数:

Alignment 批注框的边距类型，只能取以下常量：

TrackChangesConstants.LEFT 靠左

`TrackChangesConstants.RIGHT 靠右

public int getCommentAlignment()

返回批注框的边距类型

**修订(TrackChange)**

在文字处理中,书签管理器trackchanges.Revision

在这个类中有两个方法,接受修订,拒绝修订.

示例1

Document doc = Application.getWorkbooks().getActiveWorkbook().

getDocuments().getActiveDocument();

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

baseText.insertText(0,"this is a test!");

Revisions revisions = doc.getRevisions();//得到修订管理器

doc.beginTrackRevision();//开始修订

baseText.insertText(0,"new1");

baseText.insertText(10,"new2");

Revision revision = revisions.getRevision(0);//得到下一个修订

revision.reject();//拒绝修订

基本API介绍：

public void accept()

接受修订

public void reject()

拒绝修订

#### 5.14文字处理的常用属性集

1. **字体属性(FontAttribute)**

该类所定义的功能,都是用户使用较为频繁的.例如我们在文字处理中,设置某一段文字的字体,字号等等,该类具有对字体一般操作的功能.另外我们在电子表格,简报制作中,对文字的编辑也同样使用该类.

以下是在文字处理中对文字的编辑示例:

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Document doc = workbook.getDocuments().getActiveDocument();//得到当前活动文档

Section section = doc.getSection(0); //得到文档中第一小节

BaseText baseText = section.getBaseText(); //得到文档中的正文

baseText.insertText(0,"这是插入到第一段的文字示例!"); //在正文插入一段文字

FontAttribute font = new FontAttribute();

font.setFontName("隶书"); //设置字体为隶书

baseText.setFontAttribute(2,5,font); //把正文的第3到第7个字,共5个文字设置为隶书

以下是在电子表格中对文字的编辑示例:

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheet worksheet = workbook.getWorksheets().getActiveWorksheet();//得到活动工作表

Range range =worksheet.getRange("A1:D4"); //单元格区域 A1:D4

FontAttribute font = new FontAttribute();

font.setFontName("Roman"); //设置字体为Roman

font.setBold(true); //设置粗体

font.setForeground(255,0,0); //设置字体颜色

font.setStrikeThrough(true); //设置删除线

range.setFontAttribute(font);

以下是在简报制作中对文字的编辑示例:

对简报的标题设置字体,颜色,字号等.

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Presentation present = workbook.getPresentations().getActivePresentation();

Shape shape=present.getActiveSlide().getShapes().getAllShapes()[0];

FontAttribute font = new FontAttribute();

font.setFontName("Roman"); //设置字体为Roman

font.setBold(true); //设置粗体

font.setForeground(255,0,0); //设置字体颜色

font.setFontSize(50); //字号50

shape.getShapeText().setFontAttribute(font);

以下列举了FontAttribute 中一些重要API

setAsianFontFamily(String fontName) 设置亚洲字体。

setFontFamily(String fontName) 设置英文字体

setFontName(String fontName) 设置字体名称（英文和亚洲字体都有效）

setFontSize(double fontSize) 设置字体大小

setBold(boolean isBold) 设置字体是否为粗体

setItalic(boolean isItalic) 设置字体斜体

setForeground(Color fontForeColor) 设置字体前景色

setForeground(int fr, int fg, int fb) 设置字体前景色

**2．段落属性(ParagraphAttribute)**

该类实现了对段落的一些常用操作,例如:段落的对齐方式,缩进与间距,换行与分页等等.在文字处理,电子表格,简报制作中,只要是对段落的设置,一般都用到此类.

得到ParagraphAttribute对象

Workbook workbook=Application.getWorkbooks().getActiveWorkbook();

Document doc = workbook.getDocuments().getActiveDocument();

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

ParagraphAttribute paragraphAttr = baseText.getParagraphAttribute(2);

以下是在文字处理中示例如下:

示例:在文字处理的正文部分,设置居中对齐,设置段落的行间距为20磅

Workbook workbook=Application.getWorkbooks().getActiveWorkbook();

Document doc = workbook.getDocuments().getActiveDocument();

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

ParagraphAttribute paragraphAttr = baseText.getParagraphAttribute(2);

ParagraphAttr.setOutlineLevel(0);

paragraphAttr.setAlignment(ParagraphConstants.ALIGNMENT\_CENTER);

paragraphAttr.setLineSpacingType(ParagraphConstants.LINESPACING\_MULTIPLE);

paragraphAttr.setLineSpacingAt(20.0f);

baseText.setParagraphAttribute(2,6,paragraphAttr,true);

以下列举了ParagraphAttribute 中一些重要API

setAlignment(int alignment) 设置段落的对齐方式。

setOutlineLevel(int level) 段落的大纲级别（0-9级，其中0级表示正文）

setFirstLineIndent(float firstLineIndent) 设置首行缩进距离（默认单位为磅）

setSpaceAbove(float spaceAbove) 设置段前间距（默认单位为磅

setRightIndent(float right) 设置段落右缩进距离（默认单位为磅）

**3 边框属性(BorderAttribute)**

该类主要争对文字,段落,单元格,表格进行加边框设置.

示例:在文字处理中对一段选中的文字进行加边框

BorderAttribute borderAttribute = WpSelection.getBorderAttribute();

borderAttribute.setApplyType(BorderAndShadingConstants.APPLY\_TEXT);

borderAttribute.setBorderType(BorderAndShadingConstants.BORDERTYPE\_BOX);

borderAttribute.setBorderLineType(BorderAndShadingConstants.LEFT,0);

borderAttribute.setBorderLineWidth(BorderAndShadingConstants.LEFT,0.5f);

borderAttribute.setBorderLineType(BorderAndShadingConstants.TOP,0);

borderAttribute.setBorderLineWidth(BorderAndShadingConstants.TOP,0.5f);

borderAttribute.setBorderLineType(BorderAndShadingConstants.RIGHT,0);

borderAttribute.setBorderLineWidth(BorderAndShadingConstants.RIGHT,0.5f);

borderAttribute.setBorderLineType(BorderAndShadingConstants.BOTTOM,0);

borderAttribute.setBorderLineWidth(BorderAndShadingConstants.BOTTOM,0.5f);

WpSelection.setBorderAttribute(borderAttribute);

示例:在文字处理中对选中的文字加边框

BorderAttribute borderAttr=new BorderAttribute();

borderAttr.setApplyType(BorderAndShadingConstants.APPLY\_TEXT);

borderAttr.setBorderLineType(BorderAndShadingConstants.TOP,1);

borderAttr.setBorderLineType(BorderAndShadingConstants.LEFT,3);

borderAttr.setBorderLineType(BorderAndShadingConstants.RIGHT,6);

borderAttr.setBorderLineType(BorderAndShadingConstants.BOTTOM,8);

borderAttr.setBorderLineColor(BorderAndShadingConstants.TOP,Color.RED);

borderAttr.setBorderLineColor(BorderAndShadingConstants.LEFT,Color.CYAN);

borderAttr.setBorderLineColor(BorderAndShadingConstants.RIGHT,Color.YELLOW);

borderAttr.setBorderLineColor(BorderAndShadingConstants.BOTTOM,Color.ORANGE);

WpSelection.setBorderAttribute(borderAttr);

以下列举了BorderAttribute 中一些重要API

setApplyType(int bordersApplyType) 设置边框应用范围

setBorderLineColor(int borderPosition, Color color) 设置边框线颜色

setBorderLineType(int borderPosition, int lineType) 设置边框线条属性

setBorderType(int borderType) 设置边框类型

注:对一段文字进行加边框操作,必须对其四个边框分别设置,才有效.

**4. 页面边框属性(PageBorderAttribute)**

该类定义了页面边框的属性,例如:页面边框的类型,颜色,线型,宽度等.

示例:以下是在文字处理中,设置页面边框.

PageBorderAttribute pageBorderAttr = WpSelection.getPageBorderAttribute();

pageBorderAttr.setApplyType(BorderAndShadingConstants.APPLY\_TO\_SECTION);

pageBorderAttr.setBorderType(BorderAndShadingConstants.BORDERTYPE\_BOX);

pageBorderAttr.setBorderLineType(BorderAndShadingConstants.LEFT,0);

pageBorderAttr.setBorderLineWidth(BorderAndShadingConstants.LEFT,0.5f);

pageBorderAttr.setBorderLineType(BorderAndShadingConstants.TOP,0);

pageBorderAttr.setBorderLineWidth(BorderAndShadingConstants.TOP,0.5f);

pageBorderAttr.setBorderLineType(BorderAndShadingConstants.RIGHT,0);

pageBorderAttr.setBorderLineWidth(BorderAndShadingConstants.RIGHT,0.5f);

pageBorderAttr.setBorderLineType(BorderAndShadingConstants.BOTTOM,0);

pageBorderAttr.setBorderLineWidth(BorderAndShadingConstants.BOTTOM,0.5f);

pageBorderAttr.setDistanceFrom(1);

WpSelection.setPageBorderAttribute(pageBorderAttr);

以下列举了BorderAttribute 中一些重要API

setApplyType(int applyType) 设置边框应用范围

setBorderType(int borderType) 设置边框类型

setBorderLineColor(int borderPosition, Color color) 设置边框线颜色

setBorderLineType(int borderPosition, int lineType) 设置边框线条属性

**5.底纹属性(ShadingAttribute)**

该类主要是设置段落,正文的底纹.其中包括选中的文字底纹颜色,纹理等.

示例:给选中的文字加底纹.

Workbook workbook=Application.getWorkbooks().getActiveWorkbook();

Document doc = workbook.getDocuments().getActiveDocument();

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

ShadingAttribute shadingAttr = baseText.getShadingAttribute(0);

shadingAttr.setTextureIndex(45);

baseText.setShadingAttribute(2,4,shadingAttr);

TextRange textRange = doc.getSelectionRange();

ShadingAttribute shadingAttr=textRange.getShadingAttribute();

shadingAttr.setTextureIndex(12);

textRange.setShadingAttribute(shadingAttr);

#### 5.15工具类（Utilities）

为了辅助用户二次开发永中Office的应用程序，我们定义了一个辅助工具类对象Utilities，其中的方法都是以public static为访问访问权限的，用户可以随时调用。由于Office的API方法参数一般以磅值作为默认单位，而中国人习惯以厘米作为默认的度量单位来使用，所以必须提供一种简便的方法来进行单位的转换。以下是度量单位常用的转换方法：

static float pointToCentimeter(float measure) 将磅值转换为厘米。参数measure代表指定的磅值，返回一个转换后的厘米值。

static float centimeterToPoint(float measure) 将厘米转换为磅值。参数measure代表指定的厘米值，返回一个转换后的磅值。

static float pointToPixel(float measure) 将磅值转换为像素。参数measure代表指定的磅值，返回一个转换后的像素值。

static float pixelToPoint(float measure) 将像素值转换为磅值。参数measure代表指定的像素值，返回一个转换后的磅值。

以下示例打印厘米与磅的相互转换，像素与磅的相互转换。

float point = Utilities.centimeterToPoint(2.0f);

float centimeter = Utilities.pointToCentimeter(50.0f);

float pixel = Utilities.pointToPixel(30.0f);

float point = Utilities.pixelToPoint(100.0f);

## 第六章 简报制作

本章将为您介绍永中Office简报制作的宏及二次开发。

本章内容共分为以下4节：

6.1 简报制作管理器 (Presentations)

6.2 简报制作应用 (Presentation)

6.3 幻灯片管理器（Slides)

6.4幻灯片（Slide)

#### 6.1 简报制作管理器（Presentations)

Presentations 是简报制作应用中一个很重要的类，通过它可以很好的对简报制作应用进行操作。比如：添加、隐藏、关闭、获取、重命名简报制作文档。

下面请看以下几个关于简报制作管理器操作的代码

示例1 ：在当前简报制作管理器中添加一个新的简报制作应用

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得Presentations对象

Presentations presentations = workbook.getPresentations();

//添加一个新的简报制作应用

Presentation presentation = presentations.addPresentation();

示例2 ：在当前简报制作管理器中根据名字获取一个简报制作应用的对象

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Presentations presentations = workbook.getPresentations();

//获取一个简报制作应用的对象

Presentation presentation = presentations.getPresentation("简报1");

示例 3 ：在当前简报制作管理器中移除一个已经存在的简报制作应用

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Presentations presentations = workbook.getPresentations();

//在当前简报制作管理器中添加一个简报制应用

Presentation presentation = presentations.addPresentation();

//获得所添加的那个简报制应用的名称

String name = presentation.getName();

//移除一个已经存在的简报制作应用

presentations.removePresentation(name);

以下列举了在Presentations 中简报制作应用的一些API

Presrntation getPresentation(String name)

根据名字获取简报制作应用对象。

Presentation[] getAllPresentations()

获取所有该简报制作管理器中所有简报简报制作应用对象。

Presentation addPresentation()

添加一个新的简报制作应用。

openPresentation(String name)

打开一个简报制作文档（注：是该文件中隐藏的简报制作文档）。

closePresentation(String name)

关闭一个简报制作文档。

removePresentation(String name)

移除一个指定的简报制作应用。

下面提供一个完整的示例代码，当中基本上包括了Pesentationsl类中常用的API

//样例4.2

import application.Application;

public class TestPresentations

{

public TestPresentations()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.PRESENTATION);

}

public void process()

{

//获得当前活动工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得当前简报制作管理器

Presentations presentations = workbook.getPresentations();

//添加一个简报制作应用

presentations.addPresentation();

//返回当前简报的总数

int count = presentations.getCount();

//获得所添加的简报制作应用的名称

Presentation presentation = presentations.getActivePresentation();

String name = presentation.getName();

//重命名这个简报制作的名称

presentations.renamePresentation(name, "重命名的简报");

Presentation prestation = presentations.getActivePresentation();

String newName = presentation.getName();

//移除这个简报制作

presentations.removePresentation("重命名的简报");

int newCount = presentations.getCount();

}

public static void main(String[] args)

{

TestPresentations myTest = new TestPresentations();

myTest.process();

}

}

#### 6.2 简报制作应用（Presentation)

Presentation是简报制作应用中一个常用的类，该类定义了对幻灯片对象进行操作的一些方法，如添加新幻灯片母版、切换幻灯片视图、删除指定的幻灯片、选定幻灯片、设置幻灯片的背景效果、得到幻灯片的背景效果、应用幻灯片切换方式、应用幻灯片的配色方案、设置自定义播放列表管理器等。下面我们对己个常用的API进行讲解

我们先看一下关于切换幻灯片视图的代码

示例： 切换幻灯片的视图

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Presentation present = workbook.getPresentations().getActivePresentation();

present.switchView(3);

其中“3”是指将幻灯片视图切换到幻灯片浏览视图，在此类中可以通过getViewType()方法的到当前幻灯片的视图。

下面是切换到其它的视图的常量：

PresentationConstants.NORMALVIEW = 0(常规视图);

PresentationConstants.SLIDESORTVIEW = 3(幻灯片浏览视图);

PresentationConstants.SHOWVIEW = 4(放映视图);

PresentationConstants.SLIDEMASTER = 5(幻灯片母版);

PresentationConstants.HANDOUTMASTER = 7(讲义母版 );

PresentationConstants.NOTEMASTER = 8(备注母版);

PresentationConstants.NOTEPAGEVIEW = 9(备注页视图);

下面给出一个关于设置自定义幻灯片播放管理器的示例。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Presentation present = workbook.getPresentations().getActivePresentation();

CustomShowLists custonshowlists = present.getCustomShowLists();

int[] showlist = {1, 3, 2};

custonshowlists.setCustomShow("自定义播放1", "newplayer", showlist);

present.setCustomShowLists(custonshowlists);

首先要得到自定义播放管理器的对象之后才能设置新的自定义播放管理器,showlist是新的自定义播放幻灯片列表，newplayer为新的自定义播放名称。

注意：如果设置幻灯片播放列表超出当前幻灯片张数将抛出异常。

以下是一些关于自定义播放的其他API,全部位于CustomShowLists类中：

//得到所有的自定义播放。

CustomShowList[] getAllCustomShowList()

//替换原来的某个自定义播放。

//setCustomShow(CustomShowList value, int i)

/根据名称得到某个自定义播放。

CustomShowList getCustomShowList(String name)

//设置自定义播放。

setCustomShow(String oldName, String name, int[] showList)

根据索引得到相应的自定义播放对象。

CustomShowList getCustomShow(int index)

我们再给出一个关于配色方案的使用的示例：

示例（1） 应用单个幻灯片配色方案：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Presentation present = workbook.getPresentations().getActivePresentation();

present.applyColorScheme(1, 4);

示例（2） 应用所有幻灯片配色方案：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Presentation present = workbook.getPresentations().getActivePresentation();

present.applyAllColorScheme(1, 6);

这两个API只是方法名称不同，其中的两个参数的说明如下：

参数:

index 要设置的幻灯片索引值

colorSchemeIndex 配色方案索引

其它的一的Presentationl类的API的用法可以参照我们永中Office的API帮助文档

下面我们给出一个示例：

import application.Application;

public class TestPresentation

{

public TestPresentation()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.PRESENTATION);

}

public void process()

{

//获得当前活动工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得当前简报制作管理器

Presentations presentations = workbook.getPresentations();

//添加一个简报制作应用

presentations.addPresentation();

//得到一个当前活动的Presentation对象

Presentation presentation = presentations.getActivePresentation();

String name = presentation.getName();

//将其切换到幻灯片浏览视图

presentation.switchView(3);

int type = presentation.getViewType();

//设置当前视图的Zoom值

presentation.setViewZoom(78);

int zoom = presentation.getViewZoom();

//添加新的幻灯片母版

presentation.appendSlideMaster(1);

//重命名当前简报的名称

presentation.rename("新简报名称");

String newName = presentation.getName();

}

public static void main(String[] args)

{

TestPresentation myTest = new TestPresentation();

myTest.process();

}

}

#### 6.3 幻灯片管理器（Slides)

幻灯片管理器定义了一系列的对幻灯片进行操作的方法。其中，包括在最后位置添加幻灯片、在当前位置添加幻灯片、在指定位置处插入幻灯片、删除幻灯片、返回单个幻灯片的对象、剪切\复制幻灯片、粘贴幻灯片等等。

先请看以下两个关于幻灯片操作的代码

示例1 ：在当前活动简报制作应用中最后位置添加幻灯片

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Presentation present = workbook.getPresentations().getActivePresentation();

//获得Slides对象

Slides slides = present.getSlides();

//添加幻灯片

Slide slide = slides.appendSlide();

从本示例中可以看到幻灯片管理器对象是从Presentation类中得到的，说明了他们之间的关系。

示例 2 ：返回全部幻灯片。

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Presentation present = workbook.getPresentations().getActivePresentation();

Slides slides = present.getSlides();

Slide[] slide = slides.getAllSlides();

以下列举了其它一些操作幻灯片的API

Slide addSlide()

在当前位置添加幻灯片。

Slide insertSlide(int index)

在指定位置处插入幻灯片，index为指定位置。

public void removeSlide(int index)

删除幻灯片。

public void copy(int startSlide, int length)

复制幻灯片到剪切板。

参数

startSlide 开始处的幻灯片索引值

length 复制幻灯片的张数

public void paste(int index, boolean hasFormat)

剪切板上的幻灯片粘贴到指定幻灯片索引值位置。

参数

index 粘贴到的幻灯片索引值位置

hasFormat 布尔型值，是否带格式。若为true，则带格式；为false，不带格式

最后我们给出一个实用一点的例子，幻灯片粘贴连接的操作：

import application.Application;

public class TesSlides

{

public TesSlides()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.PRESENTATION);

}

public void process()

{

//获得当前活动工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得当前简报制作管理器

Presentations presentations = workbook.getPresentations();

//添加一个简报制作应用

presentations.addPresentation();

//得到一个当前活动的Presentation对象

Presentation presentation = presentations.getActivePresentation();

String name = presentation.getName();

//得到幻灯片管理器

Slides slides = presentation.getSlides();

int counts = slides.getSlideCount();

//在末尾添加5张幻灯片

slides.appendSlide();

slides.appendSlide();

slides.appendSlide();

slides.appendSlide();

slides.appendSlide();

//在当前位置添加幻灯片

slides.addSlide();

//返回幻灯片的总数

int count = slides.getSlideCount();

//复制幻灯片

slides.copy(1, 3);

//粘贴链接幻灯片,两个参数分别为粘贴的位置和是否带有格式粘贴

slides.pasteLink(4, true);

}

public static void main(String[] args)

{

TesSlides myTest = new TesSlides();

myTest.process();

}

}

#### 6.4 幻灯片（Slide)

Slide是简报制作应用中一个很重要的类，主要是对单个的幻灯片进行的操作。通过它可以获得幻灯片ID、设置幻灯片的布局、获取幻灯片中的浮动图形对象、设置隐藏幻灯片、设置幻灯片是否显示、设置 ColorScheme 对象、返回 Comments 对象（该对象代表批注的集合）、设置代表设计的 Design 对象、根据动画方案列表中组的索引值和对应的数据的索引值添加动画方案、应用幻灯片切换方式等等。

先请看一个关于幻灯片布局的操作代码

示例1 ：将幻灯片布局设置为标题幻灯片

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Presentation present = workbook.getPresentations().getActivePresentation();

//获得Slides对象

Slides slides = present.getSlides();

//获得当前幻灯片

Slide slide = slides.getActiveSlide();

//将幻灯片布局设置为标题幻灯片

slide.setSlideLayout(SlideConstants.LAYOUT\_TITLE);

我们在先前的类层次图上可以了解到Slide的类层次和结构。Presentation从Presentations中得到，Slides从Presentation中得到，最后Slide是从Slides中得到的。而从Slide类中继续下去可以得到如Designs、HeaderFooter等等对象。

以下列举了在Slide 中获得其他对象的一些API

public PgShapes getShapes()

获取幻灯片中的浮动图形对象。

public FillAttribute getFillAttribute()

返回幻灯片背景。

public ColorScheme getColorScheme()

返回 ColorScheme 对象。

public PgComments getComments(PgShape shape)

返回 Comments 对象，该对象代表批注的集合。

public Design getDesign()

返回代表设计的 Design 对象。

public HeaderFooter getHeaderFooter(int type)

返回其HeaderFooter对象。

public Master getMaster()

返回一个代表幻灯片母版的 Master 对象。

public SlideTransitionSetting getSlideShowTransition()

返回一个 SlideTransitionSetting 对象，该对象代表指定幻灯片切换的特殊效果。

public TimeLine getTimeLine()

返回 TimeLine 对象，该对象代表幻灯片的动画日程表。

示例 2 ：设置幻灯片的背景

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Presentation present = workbook.getPresentations().getActivePresentation();

//得到幻灯片片管理器

Slides slides = present.getSlides();

Slide slide = slides.getActiveSlide();

//得到背景设置的对象

FillAttribute fill = slide.getFillAttribute();

//设置背景的颜色类型

fill.setFillColorType(ChartConstants.FILL\_CUSTOM);

//设置背景值

fill.getTextureAttribute().setTextureIndex(2);

slide.setFillAttribute(fill);

对应的类中还有其它如设置代表设计的 Design 对象、设置一个 HeaderFooter 对象等等，他们的API操作都和设置幻灯片的背景相类似。

在简报制作应用中，Slide是一个功能非常强大不可或缺的类，相应的，它也提供了非常丰富的API以提供对永中Office 的二次开发。

最后我们给出一个关于Slide类综合应用的样例

import application.Application;

public class TestSlide

{

public TestSlide()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.PRESENTATION);

}

public void process()

{

//获得当前活动工作簿

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

//获得当前简报制作管理器

Presentations presentations = workbook.getPresentations();

//添加一个简报制作应用

presentations.addPresentation();

//得到一个当前活动的Presentation对象

Presentation presentation = presentations.getActivePresentation();

String name = presentation.getName();

//得到幻灯片管理器

Slides slides = presentation.getSlides();

//在末尾添加1张幻灯片

slides.appendSlide();

//返回幻灯片的总数

int count = slides.getSlideCount();

//得到当前活动的幻灯片的对象

Slide slide = slides.getActiveSlide();

//得到当前活动的幻灯片的ID

int id = slide.getID();

//设置幻灯片的布局为只有标题

slide.setSlideLayout(4);

int layout = slide.getLayout();

//将本幻灯片移动到同一集合中的指定位置，并适当地对集合中所有其他项目进行重新编号。

slide.moveTo(1);

//根据动画方案列表中组的索引值和对应的数据的索引值添加动画方案

slide.setAnimationScheme(2, 3);

int type = slide.getAnimationSchemeType();

int index = slide.getAniamtionSchemeIndex();

}

public static void main(String[] args)

{

TestSlide myTest = new TestSlide();

myTest.process();

}

}

## 第七章 公共组件

本章将为您介绍永中Office公共组件的宏及二次开发。公共组件包括两个部分：自选图形和图表。

本章内容共分为以下4节：

* 1. 自选图形管理器（Shapes)
  2. 自选图形 （Chart）
  3. 图表管理器 (Charts)
  4. 图表 (Chart)

#### 7.1 自选图形管理器（Shapes)

先看以下两个添加自选图形的方法

示例1 ：在当前活动工作表中添加一文本为"艺术字"的艺术字

//得到当前活动工作表

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//通过工作表获得自选图形管理器

Shapes shapes = sheet.getShapes();

Shape shape = shapes.addWordArt(WordArtConstants.TEXT\_EFFECT\_2, "艺术字", "宋体", 1, 12, 250, 180, 400, 320);

示例2 ：在文字处理中添加一文本为"艺术字"的艺术字

//得到当前文字处理

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents docs = workbook.getDocuments();

Document doc = docs.getActiveDocument();

//通过文字处理获得自选图形管理器

Shapes shapes = doc.getShapes();

//添加艺术字

Shape shape = shapes.addWordArt(WordArtConstants.TEXT\_EFFECT\_2, "艺术字", "宋体", 1, 12, 250, 180, 400, 320);

以下列举了在Shapes类中添加自选图形的方法

public Shape addTextBox(int director, double startX, double startY, double endX, double endY)

添加一个文本框。

public Shape addPicture(String fileName)

添加一个剪贴画或图片。

public Shape addShape(int type, double startX, double startY, double endX, double endY)

添加一个自选图形。

public Shape addLine(int type, double startX, double startY, double endX, double endY)

添加线条。

public Shape addConnector(int type, double startX, double startY, double endX, double endY)

添加连接线。

public Shape addWordArt(int wordArtStyle, String text, String fontName, int fontType,

int fontSize, double startX, double startY, double endX, double endY)

添加艺术字。

示例3 ：在当前活动工作表中获取一文本框，然后插入字符串

//得到当前活动工作表

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Worksheets sheets = workbook.getWorksheets();

Worksheet sheet = sheets.getActiveWorksheet();

//通过工作表获得自选图形管理器

Shapes shapes = sheet.getShapes();

//获取文本框

Shape shape = shapes.getShape("文本框 1");

//插入字符串"Hello Shapes"

shape.getShapeText().insertText("Hello Shapes");

示例4 ：获取所有文字处理中自选图形的数组

//得到当前文字处理

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents docs = workbook.getDocuments();

Document doc = docs.getActiveDocument();

//通过文字处理获得自选图形管理器

Shapes shapes = doc.getShapes();

//获取所有自选图形的数组

Shape[] allShapes = shapes.getAllShapes();

我们可以通过名称来返回自选图形对象，也可以直接返回所有自选图形数组等等。通过名称来得自选图形的时候，名称即是自选图形类型加上ID，如“矩形 1”，“椭圆 3”等等。ID是根据插入的顺序来定的，一旦分配给某个自选图形一个ID，那么这个自选图形和这ID就永远捆绑在了一起，如果这个自选图形被删除了，那么这个ID也就被废弃而不会再被使用。如：我们连续插入了三个自选图形：矩形，正五边形，正六边形。那么对应的名称就是“矩形 1”，“正五边形 2”，“正六边形 3”。然后我们删除正五边形，那么所剩的两个自选图形也就是“矩形 1”，“正六边形 3。此时我们若再插入一个椭圆，则其名称为“椭圆 4”。

以下列举了在Shapes类中获取自选图形的方法，这几个方法都是抽象方法，被WpShapes，SsShapes，PgShapes所实现。

public abstract Shape getShape(String shapeName);

根据名字获取图形对象

public abstract Shape[] getAllShapes();

获得当前应用中的所有自选图形数组。

public abstract Shape[] getShapes(String name);

根据名称返回自选图形数组。

示例5 ：选中所有文字处理中自选图形

//得到当前文字处理

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Documents docs = workbook.getDocuments();

Document doc = docs.getActiveDocument();

//通过文字处理获得自选图形管理器

Shapes shapes = doc.getShapes();

//选中所有自选图形

shapes.selectAll();

Shapes类提供了选中图形对象的一系列方法，列举如下：

public void selectAll()

选中所有图形对象。

public void unSelect()

没有图形对象被选中。

public void selectRange(String[] objectNames)

选中自选图形，文本框或艺术字等图形对象。

下面我们提供一个完整的示例代码

//样例7.1

import application.Application;

public class MyShapes

{

private Workbook book;

public MyShapes()

{

//启动永中office应用程序

Application.getApplicationInstance();

book = Application.getWorkbooks().addWorkbook(GlobalConstants.SPREADSHEET);

}

public void shapesExample()

{

Shapes shapes = book.getActiveShapes();

Dialogs.showMessageDialog("ready to add autoshapes");

//添加一矩形并设置高度为400

shapes.addShape(ShapeConstants.SHAPE\_RECTANGLE, 120, 130, 220, 250);

shapes.getShape("矩形 1").setHeight(400);

//插入一文本框

shapes.addTextBox(2, 320, 380, 520, 5);

//插入一艺术字

shapes.addWordArt(WordArtConstants.TEXT\_EFFECT\_1, "艺术字", "宋体", 1, 12, 250, 180, 400, 320);

//全选当前自选图形

shapes.selectAll();

//组合当前自选图形

ShapeRange range = shapes.getRange();

range.group();

//保存并关闭文件

book.saveAs("D:\\test.eio");

Dialogs.showMessageDialog("ready to close workbooks");

book.close();

}

public void exit()

{

//退出永中office

Application.exitSystem();

}

public static void main(String[] args)

{

MyShapes myShapes = new MyShapes();

myShapes.shapesExample();

myShapes.exit();

}

}

#### 7.2 自选图形（Shape)

先请看以下关于自选图形设置属性的代码

示例1 ：在当前应用插入一圆柱形，并设置其属性。

Shapes shapes = Application.getWorkbooks().getActiveWorkbook().getActiveShapes();

Shape shape = shapes.addShape(ShapeConstants.SHAPE\_CAN,200,200,400,400);

//设置宽度为498

shape.setWidth(498);

//设置不能删除

shape.setCanDelete(false);

//设置旋转角度45度

shape.setRotateAngle(45);

示例2 ：在当前应用插入一矩形，获取其填充格式并设置其填充色为蓝色。

Shapes shapes = Application.getWorkbooks().getActiveWorkbook().getActiveShapes();

Shape shape = shapes.addShape(ShapeConstants.SHAPE\_RECTANGLE,200,200,400,400);

//设置设置其填充色为蓝色

FillAttribute fill = shape.getFillAttribute();

fill.setTexture(Color.BLUE);

shape.setFillAttribute(fill);

设置自选图形属性不但包括大小，转角，填充属性等对应于界面上对话框的方法，也包含设置能否移动，复制，删除等等。

以下列举了在Shape中设置自选图形属性的一些API

public void setLockRatio(boolean isLockRatio)

设置当前图形对象是否锁定高宽比例。

public void setRotateAngle(double rotateAngle)

设置当前图形对象旋转角度。

public void setPrintShape(boolean flag)

是否打印这个对象。

public void setCanDelete(boolean flag)

设置自选图形可否被删除。

public void setHalfDrag(boolean flag)

设置自选图形可否被垂直水平拖拉。

public void setFillAttribute(FillAttribute fill)

设置图形对象的填充格式。

public void setLockPosition(boolean isLock)

设置当前图形是否锁定位置使之不能拖动(本方法只针对非组合对象)

public void setCanMoveLocation(boolean flag)

设置是否能移动到有效位置以外

public void setCanGroup(boolean flag)

设置是否能与其它自选图形组合。

public void setLockSize(boolean isLock)

设置自选图形是否锁定大小

public void setLockLocation(boolean isLock)

设置自选图形是否锁定不能旋转，翻转。

public void setCanFlip(boolean isFlip)

设置自选图形是否翻转

public void setCanRotate(boolean isRotate)

设置自选图形是否旋转

示例3 ：在当前应用中插入一文本框，返回ShapeText对象来插入文本，返回文本框属性对象来设置自适应文字。

Shapes shapes = Application.getWorkbooks().getActiveWorkbook().getActiveShapes();

Shape shape = shapes.addTextBox(2, 120, 180, 380, 380);

//得到ShapeText对象，通过它插入文本

ShapeText text = shape.getShapeText();

text.insertText("My TextBox");

//得到文本框属性对象，设置它为自适应文字

TextFrame textFrame = shape.getTextFrame();

textFrame.setAutoSize(true);

注：ShapeText类定义了对自选图形文本进行操作的方法。

Shape类提供了很多方法来返回Shape各类属性，如阴影，3D，大小位置等等。一些常见API列举如下：

public boolean isPicture()

判断当前图形对象是否是图片。

public String getName()

获取图形对象的名字。

public boolean isLockRatio()

判断当前图形对象是否为锁定高宽比例。

public double getRotateAngle()

获取图形对象（自选图形，文本框，艺术字）的旋转角度。

public LineFormat getLineFormat()

获取当前图形对象的线条格式对象。

public ShadowFormat getShadowFormat()

获取当前图形对象的阴影格式对象。

public ThreeDFormat getThreeDFormat()

获取图形对象的三维格式。

public PictureFormat getPictureFormat()

获取图片格式。

public TextEffectFormat getTextEffectFormat()

获取艺术字格式。

public ConnectFormat getConnectFormat()

获取连接线格式。

public int getType()

获取自选图形的类型。

public FillAttribute getFillAttribute()

获取图形对象的填充格式。

public ShapeText getShapeText()

返回自选图形文本对象

public TextFrame getTextFrame()

得到文本框的属性。

public boolean isLockLocation()

返回自选图形是否不能旋转，翻转。

public boolean isCanRotate()

返回自选图形是否可以旋转。

下面我们提供一个完整的示例代码，插入一自选图形，并对其进行相关设置

//样例7.2

import application.Application;

public class MyShape

{

private Workbook book;

public MyShape()

{

//启动永中office应用程序

Application.getApplicationInstance();

book = Application.getWorkbooks().addWorkbook(GlobalConstants.WORDPROCESSOR);

}

public void shapeExample()

{

Shapes shapes = book.getActiveShapes();

//插入自选图形

Dialogs.showMessageDialog("ready to add autoshapes");

Shape shape = shapes.addShape(ShapeConstants.SHAPE\_HEXAGON, 120, 130, 320, 350);

//设置自选图形填充色

FillAttribute fill = shape.getFillAttribute();

fill.setTexture(Color.ORANGE);

shape.setFillAttribute(fill);

//设置自选图形线条

LineFormat lineAttr = shape.getLineFormat();

lineAttr.setWidth(2.2);

//插入自选图形文本

ShapeText shapeText = shape.getShapeText();

shapeText.insertText("shapeExample");

//设置自选图形不能删除和移动

shape.setCanDelete(false);

shape.setLockPosition(true);

//保存并关闭文件

book.saveAs("D:\\test.eio");

Dialogs.showMessageDialog("ready to close workbooks");

book.close();

}

public void exit()

{

//退出永中office

Application.exitSystem();

}

public static void main(String[] args)

{

MyShape myShape = new MyShape();

myShape.shapeExample();

myShape.exit();

}

}

#### 7.3 图表管理器 (Charts)

先请看以下两个关于图表操作的代码

示例1 ：在工作表1中添加浮动图表

//获得当前工作簿

Workbook book = Application.getWorkbooks().getActiveWorkbook();

//获得工作表1

Worksheet sheet = book.getWorksheets().getWorksheet("工作表1");

Range range=sheet.getRange("A5:E12");

//单元格区域填充随机数并选中

range.fillRandom();

//获得Charts对象

Charts charts = book.getCharts();

//在当前工作表中添加浮动图表

charts.addFloatingChart(ChartConstants.EMOCHART\_COLUMN,ChartConstants.EMOCOLUMN\_STACKED,"工作表1");

从以上示例代码中，我们可以了解到，Charts对象从Workbook对象处获得。即Charts对象管理整个工作簿中的图表。另外，要添加图表的数据源在工作表上，而且数据源要已被选中，以上示例代码中 range.fillRange() 这一段代码就表示选中单元格区域并填充随机数以作为要添加图表的数据源。上面添加的图表是浮动图表。永中Office 还有另外两种图表，它们分别是工作表图表和单元格图表。它们的添加方法有如下面的API：

Chart addFloatingChart(int type, int subType, String workSheetName)

添加浮动图表。

参数:

type 图表主类型

subType 图表次类型

workSheetName 要添加图表的工作表名

返回: 图表

Chart addWorksheetChart(int type, int subType, String chartSheetName)

添加工作表图表。

参数:

type 图表主类型

subType 图表次类型

chartSheetName 要添加的图表工作表名

返回: 图表

Chart addCellChart(int type, int subType, String workSheetName, int row, int col)

添加单元格图表。

参数:

type 图表主类型

subType 图表次类型

workSheetName 工作表的名字

row 单元格的行

col 单元格的列

返回: 图表

Charts 提供了获得图表的API，有如下这些：

Chart getActiveChart()

获得当前工作表下活动图表。

Chart[] getAllCharts(String sheetName)

取得指定工作表上的所有图表。

Chart getWorksheetChart(String sheetChartName)

通过名称获取工作表图表的对象。

Chart getCellChart(String workSheetName, int row, int col)

通过工作表名字和行列号获取单元格图表对象。

以下几个API 是有关Chart的名称的。Chart的名称我们可以在Charts中获得，也可以在Charts中改变。这是为了提供某种方便。

String getFloatingChartName()

取得当前活动的浮动图表的名字。

Chart getFloatingChart(String chartName)

在当前工作表通过名字获得活动图表对象。

chartName 表示的是图表名称

Chart getFloatingChart(String workSheetName, String chartName)

通过工作表名字和图表名称获取浮动图表对象。

void setFloatingChartName(String oldName, String newName)

改变浮动图表的名字。

下面提供一个完整的示例代码，表示在工作表1中输入数据源，生成一个浮动图表，一个单元格图表和工作表图表

//样例7.3

import application.Application;

public class TestCharts

{

private Workbook workbook;

public TestCharts()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.SPREADSHEET);

//获得当前活动工作簿

workbook = Application.getWorkbooks().getActiveWorkbook();

}

public void process()

{

Worksheet sheet = workbook.getWorksheets().getWorksheet("工作表1");

Range range=sheet.getRange("A5:E12");

range.fillRandom();

//获得Charts 对象

Charts charts = workbook.getCharts();

//添加浮动图表

Chart chart = charts.addFloatingChart(ChartConstants.EMOCHART\_COLUMN,

ChartConstants.EMOCOLUMN\_STACKED,"工作表1");

String chartName = charts.getFloatingChartName();

//添加工作表图表

charts.addWorksheetChart(ChartConstants.EMOCHART\_COLUMN,

ChartConstants.EMOCOLUMN\_STACKED,"图表1");

//添加单元格图表

sheet.activate();

range.select();

charts.addCellChart(ChartConstants.EMOCHART\_COLUMN,

ChartConstants.EMOCOLUMN\_STACKED,"工作表1",2,1);

Chart[] cs = charts.getAllCharts("工作表1");

}

public static void main(String[] args)

{

TestCharts myTest = new TestCharts();

myTest.process();

}

}

#### 7.4 图表 (Chart)

Chart是图表应用中的一个主要的类。通过它，我们可以对图表的类型、外观属性、图表的数据源以及图表的标题等进行操作。

下面先看一个图表操作的示例：生成一个活动图表并添加图表标题

Workbook book = Application.getWorkbooks().getActiveWorkbook();//获得当前工作表

Worksheet sheet = book.getWorksheets().getActiveWorksheet(); Range range=sheet.getRange("A5：E12");//填充单元格区域，并作为图表的数据源

range.fillRandom();//获得Charts 对象

Charts charts = book.getCharts();

//添加一个浮动图表charts.addFloatingChart(ChartConstants.EMOCHART\_COLUMN,

ChartConstants.EMOCOLUMN\_STACKED,sheet.getName()); //获得当前活动图表

Chart chart = charts.getActiveChart();//给图表添加图表标题

chart.addTitle(ChartConstants.CHART\_TITLE, "生产情况");

void addTitle(int titleType, String text)

添加标题。

参数：

titleType 标题类型，其取值为如下常量：

ChartConstants.CHART\_TITLE ChartConstants.X\_TITLE ChartConstants.Y\_TITLE

text 标题名称

以下几个API 是关于Chart 的类型操作的：

void setChartType(int type, int subType)

设置图表的类型。

参数：

type 图表主类型索引 ，其类型索引在类ChartConstants中定义

subType 图表次类型索引 ，其类型索引在类ChartConstants中定义

int getMainType()

获得图表的主类型。

int getChartSubType()

取得图表次类型。

以下几个API 是关于Chart 的数据源操作的：

void setSourceData(String rangeSource, boolean flag)

设置图表数据源。

参数：

rangeSource 图表数据源

flag 按行按列标志 true 表示按行 false 表示按列

代码示例：设置当前活动图表的数据源

Workbook book = Application.getWorkbooks().getActiveWorkbook();//获得当前工作表

Worksheet sheet = book.getWorksheets().getActiveWorksheet(); Range range=sheet.getRange("A5：E12");//填充单元格区域，并作为图表的数据源

range.fillRandom();Charts charts = book.getCharts();//添加一个浮动图表

charts.addFloatingChart(ChartConstants.EMOCHART\_COLUMN,

ChartConstants.EMOCOLUMN\_STACKED,sheet.getName()); //获得当前活动图表

Chart chart = charts.getActiveChart();//设置图表的数据源

chart.setSourceData("='工作表1'!A5：E12", false);

String getSourceData()

获得图表数据源。

void setSeriesByRow(boolean isRow)

设置图表系列是按行或按列生成。

参数：

isRow 行列标志 True表示按行,False表示按列

boolean isSeriesByRow()

获得图表系列是按行还是按列生成。

以下几个API是关于图表的外观属性的，包括字体、填充、圆角、阴影等：

FontAttribute getFontAttribute()

获取字体属性。

void setFontAttribute(FontAttribute fontAttribute)

设置图表区域的字体属性（字形，字体颜色，效果等）。

参数：

fontAttribute 字体属性

代码示例：设置当前活动图表的字体属性

//获得当前活动图表

Chart chart = Application.getWorkbooks().getActiveWorkbook().getCharts().getActiveChart();

//获得图表的字体属性FontAttribute fontAttribute = chart.getFontAttribute();

FillAttribute getFillAttribute()

取得图表的填充属性。

void setFillAttribute(FillAttribute fillAttr)

设置图表的填充属性。

参数：

fillAttr 填充属性

代码示例：设置当前活动图表的填充属性

//获得当前活动图表

Chart chart = Application.getWorkbooks().getActiveWorkbook().getCharts().getActiveChart();//获得图表的填充属性

FillAttribute fill = chart.getFillAttribute();fill.getTextureAttribute().setTextureIndex(2);//设置图表的填充属性

chart.setFillAttribute(fill);

void setHasShadow(boolean hasShadow)

设置图表是否有阴影。

boolean hasShadow()

获取阴影属性。

void setHasCorner(boolean hasCorner)

设置图表是否有圆角效果。

boolean hasCorner()

获取圆角属性。

以下几个API是关于图表的组成部分的显示属性的：

void setTableAreaVisible(boolean isShow)

图表是否显示数据表

boolean isTableAreaVisible()

数据表对象是否显示。

void showLegend(boolean isShow)

图表是否显示图例。

boolean isLegendVisible()

图例对象是否显示。

以下API是关于浮动图表转换为单元格图表的：

void floatChartInToCell(String worksheetName, int row, int column)

把浮动图表插入到指定工作表指定的单元格中.

参数：

worksheetName 指定的工作表名称(若worksheetName为Null,表示是当前工作表)

row 指定插入的行

column 指定插入的列

例子：把当前活动的浮动图表插入到工作表2的指定单元格中//获得当前工作簿

Workbook book = Application.getWorkbooks().getActiveWorkbook();Charts charts = book.getCharts();//获得当前活动图表

Chart chart = charts.getActiveChart();

if (chart != null)

{

//将当前活动图表插入到工作表2的（5，7）单元格中

chart.floatChartInToCell("工作表2", 5, 7);

}

下面提供一个完整的示例代码，表示在当前工作表根据四年来中国各类教育毕业生情况表，生成一个浮动图表，添加图表的标题，并设置图表的字体和填充属性

//样例7.3

import application.Application;

public class TestChart

{

private Workbook workbook;

public TestChart()

{

//开启永中Office

Application.getApplicationInstance();

Application.gotoProduct(GlobalConstants.SPREADSHEET);

//获得当前活动工作簿

workbook = Application.getWorkbooks().getActiveWorkbook();

}

public void process()

{

prepareData();

Chart chart = createChart();

setChartFont(chart);

setChartFill(chart);

}

private void prepareData()

{

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

sheet.setCellValue(1,2,"四年来中国各类教育毕业生情况（单位：万人）");

String[] colHeads = {"研究生","普通教育本专科","成人教育本专科","普通高中"};

String[] rowHeads = {"2001年", "2002年", "2003年", "2004年"};

int[][] data = { {55, 59, 68, 81},

{85, 95, 104,334},

{89, 88, 93, 118},

{263, 302,341,384}

};

sheet.setCellValue(3, 1, colHeads, false);

sheet.setCellValue(2, 2, rowHeads, true);

sheet.setCellValue(3, 2, data, true);

sheet.select("A1:E1");

ActiveRange.merge(true);

FillAttribute fill = new FillAttribute();

fill.setTexture(255,255,0);

ActiveRange.setFillAttribute(fill);

}

private Chart createChart()

{

//获得当前工作表

Worksheet sheet = workbook.getWorksheets().getActiveWorksheet();

sheet.select("A2:E6");

//获得Charts 对象

Charts charts = workbook.getCharts();

//添加一个浮动图表

charts.addFloatingChart(ChartConstants.EMOCHART\_COLUMN,0,sheet.getName());

//获得当前活动图表

Chart chart = charts.getActiveChart();

//给图表添加图表标题

chart.addTitle(ChartConstants.CHART\_TITLE, "四年来中国各类教育毕业生情况（单位：万人）");

//给图表添加Y轴标题

chart.addTitle(ChartConstants.Y\_TITLE, "万人");

return chart;

}

private void setChartFont(Chart chart)

{

//获得图表的字体属性

FontAttribute fontAttribute = chart.getFontAttribute();

fontAttribute.setBold(true);

//设置图表的字体属性

chart.setFontAttribute(fontAttribute);

FontAttribute fontAttribute1 = chart.getFontAttribute();

}

private void setChartFill(Chart chart)

{

//获得图表的填充属性

FillAttribute fill = chart.getFillAttribute();

fill.getTextureAttribute().setTextureIndex(2);

//设置图表的填充属性

chart.setFillAttribute(fill);

}

public static void main(String[] args)

{

TestChart myTest = new TestChart();

myTest.process();

}

}

## 第八章 永中Office COM组件开发

永中office COM组件通过JNI框架，实现从COM客户端到Java服务器端的调用。相关组件安装在...\Yozosoft\Yozo\_Office\sdk目录下。下面介绍一下如何使用Microsoft Visual Studio 6.0在VB和VC环境开发COM客户端去调用永中office。

#### 8.1 VB样例的开发

1．打开Microsoft Visual Basic 6.0，看到下图页面点打开或者从File菜单项－>New Project－>Standard EXE 项目。

2．导入类型库：Project菜单－>References－>选中引用项 “YoZoffice in-proc Type Library” 这个选项。

3．在form上创建两个frame：frame1，frame2，并在右边的属性表设置它们的属性，（例如Caption等等）。设定在frame1上添加控件，在frame2上嵌入永中office。添加form的Load和Unload事件。

4．在frame1上添加按钮，设置其Caption为新建。双击按钮，进入代码页面，添加事件代码：Form的Load和Unload事件，Command1的Click事件。

//源代码为：

Dim apps As officecom.Applications

Dim appPar As officecom.AppParameters

Dim app As officecom.Application

Private Sub Command1\_Click()

On Error GoTo errorhandle

errorhandle:

End Sub

Private Sub Form\_Load()

On Error GoTo errorhandle

Set app = CreateObject("YOZOfficeServer.Application")

Set appPar = apps.getAppParameters

appPar.setJavaWindow Frame2.hWnd, 0, 0, Frame2.Width, Frame2.Height

Set app = apps.createApplication(appPar)

app.updateJavaWindow

app.setLookFeel 0

flag = True

errorhandle:

End Sub

Private Sub Form\_Resize()

On Error GoTo errorhandle

If flag = True Then

app.setJavaWindowBounds Frame1.Width + 10, 0, Form1.ScaleWidth - Frame1.Width - 10, Form1.ScaleHeight

End If

errorhandle:

End Sub

Private Sub Form\_Unload(Cancel As Integer)

On Error Resume Next

app.getWorkbooks().closeAll

app = Nothing

End Sub

5．生成exe文件，点击按钮，可以新建一个文字处理。下面的三张截图为运行效果。

图1.1：运行刚创建的VB工程



图1.2：点击按钮激发事件的效果



图1.3：改变Form的大小，调整Office布局后的效果

注意事项：

* 当COM组件安装路径改变过的时候，可能出现在引用中找不到类型库，需要手动导入,可以使用regtlib命令将需要引用的类型库注册一下；
* 在设置Form的resize事件时，需要将其属性中ScaleMode设置为3-Pixels；
* 在debug模式下，start后end了，frame2会出现YoZo未被装载进去的现象，保存vbp重新打开可以正常。

#### 8.2 VC样例的开发

打开Visual C++，新建一个基于对话框的工程。File菜单项－>New Project－>MFC AppWizard（exe），输入文件名（YOZO\_COM）后点ok，选择Dialog Based，然后点Finish。

新建成功后，将永中Office安装目录下的sdk\bin目录下的OfficecomBridge.dll这个文件拷贝到新建的工程文件目录下。

在stdafx.h文件中添加导入类型库。

#import "OfficecomBridge.dll"

4．打开对话框资源，在对话框上添加两个Group Box，ID分别为IDC\_EIO，IDC\_BEAN，分别在对话框的构造器中添加数据成员，并在对话框的数据交换方法中添加DDX/DDV 调用代码。

class CEIO\_COMDlg : public CDialog

{

// Construction

public:

CEIODlg(CWnd\* pParent = NULL); // standard constructor

// Dialog Data

//{{AFX\_DATA(CEIODlg)

enum { IDD = IDD\_EIO\_DIALOG };

// NOTE: the ClassWizard will add data members here

CButton m\_control;

CButton m\_bean;

//}}AFX\_DATA

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CEioDlg)

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV support

//}}AFX\_VIRTUAL

void init();

// Implementation

protected:

HICON m\_hIcon;

// Generated message map functions

//{{AFX\_MSG(CEIODlg)

virtual BOOL OnInitDialog();

afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);

afx\_msg void OnPaint();

afx\_msg HCURSOR OnQueryDragIcon();

afx\_msg void OnButton1();

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

------------------------------------------------------------------------------------------------------------------------------------------

void CEIO\_COMDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialog::DoDataExchange(pDX);

//{{AFX\_DATA\_MAP(CEIO\_COMDlg)

// NOTE: the ClassWizard will add DDX and DDV calls here

DDX\_Control(pDX, IDC\_EIO, m\_control);

DDX\_Control(pDX, IDC\_BEAN, m\_bean);

//}}AFX\_DATA\_MAP

}

5．在应用程序初始化代码（EIO\_COM.cpp）中添加初始化COM库代码（红色标出）。

BOOL CEIO\_COMApp::InitInstance()

{

AfxOleInit();

AfxEnableControlContainer();

// Standard initialization

// If you are not using these features and wish to reduce the size

// of your final executable, you should remove from the following

// the specific initialization routines you do not need.

#ifdef \_AFXDLL

Enable3dControls(); // Call this when using MFC in a shared DLL

#else

Enable3dControlsStatic(); // Call this when linking to MFC statically

#endif

CEIO\_COMDlg dlg;

m\_pMainWnd = &dlg;

int nResponse = dlg.DoModal();

if (nResponse == IDOK)

{

// TODO: Place code here to handle when the dialog is

// dismissed with OK

}

else if (nResponse == IDCANCEL)

{

// TODO: Place code here to handle when the dialog is

// dismissed with Cancel

}

// Since the dialog has been closed, return FALSE so that we exit the

// application, rather than start the application's message pump.

return FALSE;

}

6．在对话框头文件（EIO\_COMDlg.h）中添加init（）方法，并在对话框实现程序（EIO\_COMDlg.cpp）中添加其实现。

-------------------------------------------------------------------------------------------------------

appPtr、booksPtr是在头文件EIO\_COMDlg.h中定义的变量：

private:

officecom::IApplicationsPtr appsPtr;

officecom::IApplicationPtr appPtr;

officecom::IWorkbooksPtr booksPtr;

IDispatch\* pDispApp;

SAFE\_CHECK是在实现文件EIO\_COMDlg.cpp中定义的宏。

#define SAFE\_CHECK(f) try {f} catch(\_com\_error &e)

{AfxMessageBox(e.ErrorMessage());}

-------------------------------------------------------------------------------------------------------

class CEIO\_COMDlg : public CDialog

{

// Construction

public:

CEIODlg(CWnd\* pParent = NULL); // standard constructor

// Dialog Data

//{{AFX\_DATA(CEioDlg)

enum { IDD = IDD\_EIO\_DIALOG };

// NOTE: the ClassWizard will add data members here

CButton m\_control;

CButton m\_bean;

//}}AFX\_DATA

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(CEioDlg)

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV support

//}}AFX\_VIRTUAL

void init();

// Implementation

protected:

HICON m\_hIcon;

// Generated message map functions

//{{AFX\_MSG(CEioDlg)

virtual BOOL OnInitDialog();

afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);

afx\_msg void OnPaint();

afx\_msg HCURSOR OnQueryDragIcon();

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

----------------------------------------------------------------------------------------------------------------------

void CEIO\_COMDlg::init()

{

if (appPtr == NULL)

{

//通过永中Office的Prog ID创建COM实例

HRESULT hr = appPtr.CreateInstance(L"YOZOfficeServer.Application");

ASSERT( SUCCEEDED( hr ) );

try

{

CWnd\* pWnd = this;

CRect rect,rect2;

pWnd->GetClientRect(&rect);

m\_control.GetClientRect(&rect2);

officecom::IAppParametersPtr appParsPtr = appsPtr->getAppParameters();

appParsPtr->setJavaWindow((long)m\_bean.m\_hWnd,0,0,rect.right-rect2.Width()-5,rect.bottom);

appPtr = appsPtr->createApplication(appParsPtr);

appPtr->setLookFeel(0);

booksPtr = appPtr->getWorkbooks();

}

catch(\_com\_error &e)

{

AfxMessageBox(e.ErrorMessage());

}

}

}

7．在对话框始初始化时添加启动永中Office库代码。

BOOL CEIO\_COMDlg::OnInitDialog()

{

CDialog::OnInitDialog();

// Set the icon for this dialog. The framework does this automatically

// when the application's main window is not a dialog

SetIcon(m\_hIcon, TRUE); // Set big icon

SetIcon(m\_hIcon, FALSE); // Set small icon

// TODO: Add extra initialization here

init();

return TRUE; // return TRUE unless you set the focus to a control

}

8．在IDC\_EIO上新建控件，如新建按钮，ID为IDC\_NEW，双击添加事件。本例事件为新建文字处理。

void CEIO\_COMDlg::OnNew()

{

SAFE\_CHECK(appPtr->getWorkbooks()->addWorkbook(1););

}

9.生成exe文件。点击按钮，可以新建一个文字处理。以下为截图：

图2.1：运行刚开发的VC工程

图2.2：点击新建按钮，创建一个文字处理

#### 8.3 C#样例的开发

前面的VC、VB样例都是在Microsoft Visual Studio 6.0环境下开发的，本样例是用C#开发的，需要在Microsoft Visual Studio.net环境下开发，这里用的是Microsoft Visual Studio 2005简体中文版。

1. 打开Microsoft Visual Studio 2005，从菜单选“文件”——“新建”——“项目”，弹出如图对话框，选“项目类型”为Visual C#——Windows，“模板”为Windows应用程序，输入“名称”和“解决方案名称”，此处为csharpeio，点确定进入下一步。

图3.1 新建项目对话框

图3.2 C#工程csharpeio

1. 进入如图2所示的UI界面，在“解决方案管理器窗口”中“解决方案csharpeio”树的“引用”节点处点击右键，选“添加引用”，在弹出的属性对话框（如图3所示）中选“COM”属性页，选中“YoZoffice in-proc Type Library”，点确定。

图3.3 添加引用属性对话框

1. 修改“Form1”的Text属性为“永中Office COM组件”，在其上面添加容器GroupBox和Panel，分别为groupbox1和panel1，将groupbox1的Text属性改为“操作”，在groupbox1上添加公共控件Button，即buton1，将其Text属性改为“新建”。并调整这些组件的大小和布局。得到如图4所示。

图3.4 组件以及布局

4. 双击“新建”按钮，进入代码页面，添加using 指令：using officecom;，分别为Form1的Load事件和buton1的Click事件添加代码，即方法private void Form1\_Load(object sender, EventArgs e)和private void button1\_Click(object sender, EventArgs e)，最后代码为：

using System;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using officecom;

namespace pgtemp

{

public partial class Form1 : Form

{

private IApplications apps;

private IApplication app;

private AppParameters appParams;

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

Type type = Type.GetTypeFromProgID("YOZOfficeServer.Application");

apps = (IApplications)Activator.CreateInstance(type);

appParams = apps.getAppParameters();

appParams.hideApplicationBar();

app = (IApplication)apps.createApplication(appParams);

app.updateJavaWindow();

}

private void button1\_Click(object sender, EventArgs e)

{

app.getWorkbooks().addWorkbook(2);

}

}

}

此处的代码功能是点击按钮，新建一个简报制作。可以重复上述添加按钮的步骤，在GroupBox上添加更多按钮，并给双击每个按钮添加不同的事件。

5.启动调试，生成COM组件，在csharpeio\bin\Debug(Release)目录下。COM组件如图5所示，点击按钮后如图6所示。

图3.5 C#COM组件样例

图3.6 监听了按钮事件的C# COM组件样例

## 第九章 嵌入浏览器样例

永中Office 提供了以小应用程序（Applet）形式嵌入网页运行的解决方案。在浏览器中运行的永中Office，几乎具备永中Office 的全部功能，功能十分强大，其运行速度也较快。

本章将为您介绍把永中Office嵌入浏览器的样例。

本章内容共分为以下2节：

9．1 永中Office嵌入浏览器的具体实施

9．2 JavaScript与YoZo\_Applet互动

#### 9.1 永中Office嵌入浏览器具体实施

**1、Applet 嵌入浏览器的原理**

Applet 是透过Web Server 将编译好的Java Byte-Code 送至使用者端的浏览器中执行，将计算负载分散给使用者的电脑。原理图如下：

图9-1

**2、Applet 嵌入浏览器的实施流程图**

图9-2

**3、 配置Applet 环境**

**Java 安全机制的原理**

从Java 技术提供者的角度来看，Java 的安全性包括以下两个方面：Java 平台是一个安全、现成的平台，在平台上可以以一种安全的方式实现基于Java 的功能；Java 所实现的安全工具和服务使得一些领域（如企业）对安全性的需求得以满足。Java 的安全性主要体现在以下几个方面：

1. Java语言和平台：Java语言是很严格的，它没有任何不安全的结构。Java平台包括Java语言、JVM 和应用程序界面（API 库）。

2. 基本安全结构：Java 版本均采用了基本的安全结构。任何应用程序的安全是由它所运行的平台的安全性和设计应用它的安全性功能所决定的，因为安全性功能被建立在平台内，Applet 开发者就能集中精力来为Applet 定义一个安全策略，一个安全策略在Applet 中提供的安全性有：

**·** 鉴定—— 对Applet发布命令的主机应用的身份应被鉴定，在对象共享中，服务器Applet在给予一个服务前也应验证客户Applet 的身份。

**·** 保密性—— Applet 数据的隐性必须被保护。对于安全数据没有合法的认证就不能被访问

**·** 正确性 ——Applet数据的正确性必须被保护，和适当的身份验证。同时不能改变Applet数据，数据改变应通过错误检查来保护。

3. 字节代码验证和类型安全：字节代码的验证已成为发现潜在安全漏洞的关键。系统中一般用验证器JVM 在运行代码段之前进行一系列的测试。因为验证器独立于Java语言或编译器，所以它也可以核查是否有Java 源语言派生的字节代码。Java 安全结构的最重要的目标是保证正确的Java 语言规范和JVM 规范。

4. 签名应用：JDK1.1 增加了对数字签名的支持，这样一个Applet 的类文件在开发结束后就可以和类文件的签名一起标记并存储在JAR（Java Archive）格式中。在每一次安装过程中，可以详细指明哪个签名者（或公钥），这样当一个数字签名正确的Applet下载时，一旦Applet 的签名者被验证为可信，这个Applet 就可以像本地代码一样自由访问系统。

**永中Office 中安全策略的配置**

我们以一种快速而便捷的方式来解决java applet 运行的安全策略问题，将对本机操作的所有的权限都开放给applet 程序。注意：在永中Office附带的jre中已经做了相应的改动，所以无需用户再做修改，但用户如果想使用其它jre版本，则需要安装以下修改步骤做修改。

1、Windows 系统

进入j2re 的安装目录，并修改java 的安全性文档，例如：

用记事本或写字板打开java.policy文件，将其中的文字

grant codeBase “file:${java.home}/lib/ext/\*” {

};

修改为

grant {

};

2、Linux 系统

进入jre 的安装目录修改java 的安全性文档

例如：cd /usr/local/Yozosoft\Yozo\_Office/jre/lib/security

编辑策略文件：

Kedit java.policy

将其中的文字

grant codeBase “file:${java.home}/lib/ext/\*” {

};

修改为grant {

};

**Java Plug-in（浏览器插件）的配置**

**Windows 系统**

步骤一、点击开始->控制面板

图9-3

步骤二、点击控制面板中的Java图标 ，弹出下图

图9-4

步骤三、点击java选项，弹出下图：

图9-5

步骤四、在Java Runtime 参数中填写类路径：

**Linux 系统**

步骤一、进入jre 的安装目录例如：cd /usr/local/Yozosoft/Yozo\_Office/jre/bin

下图显示的是永中office中jre

图9-6

步骤二、运行ControlPanel，弹出下图

图9-7

步骤三、在java选项卡中的Java Runtime 参数中填写类路径

图9-8

点击应用后，关闭Java Plug-in 控制面板。

**拷贝jar文件**

当jre配置完毕后，用户还需要将永中Office使用到的jar文件复制到用户指定使用的jre下，否则在启动applet网页是会抛ClassNotFoundException。用户可以将YOZO\_HOME\jre\lib\ext文件夹下所有的jar文件拷贝至用户使用的jre\lib\ext文件夹中即可。

**Linux 下mozilla 的配置**

您可以在JAVA\_HOME/jre/plugin/i386/ns7/ 下发现它的插件,然后把它链接到

mozilla\_home/plugins下!

命令:ln -s JAVA\_HOME/jre/plugin/i386/ns7/libjavaplugin\_oji.so mozilla\_home/plugins

记住不要复制过去!

还有:ln -s JAVA\_HOME/jre/plugin/i386/ns7/libjavaplugin\_oji.so ~/.mozilla/plugins (没有

plugins 目录,就创建一个)

**4、示例代码**

**编写永中Office的Applet程序**

下面的程序演示了如何取得永中Office实例，并在Applet中展示出来。

/\*\*

\* AppletDemo.java

\*/

import application.Application;

/\*\*

\* @author

\*/

{

/\*\* Creates a new instance of AppletDemo \*/

public AppletDemo()

{

}

public void start()

{

//创建Applet 实例

Application.createInstance(this);

// 隐藏菜单和工具条

Application.setToolbarVisible(BarConstants.MENUBAR\_INDEX, false);

Application.setToolbarVisible(BarConstants.STANDARD\_INDEX, false);

Application.setToolbarVisible(BarConstants.FORMATTING\_INDEX, false);

Application.setTaskPanelVisible(GlobalConstants.NAVIGATION\_PANE, false);

// 取得Applet 实例

Container content = Application.getInstance();

Container contentPane = getContentPane();

//添加文件

Application.getWorkbooks().addWorkbook(GlobalConstants.WORDPROCESSOR);

//获取文档

Document doc = Application.getWorkbooks().getActiveWorkbook().getDocuments()

.getActiveDocument();

//获取文档中的第一节

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

FontAttribute font = new FontAttribute();

font.setBold(true);

font.setFontSize(18.0);

//插入文字

baseText.insertText(0, " 运行在浏览器中的永中office", font);

// 将EApplet 添加到Applet 面板中

contentPane.add(content);

}

}

**编写网页文件**

在网页中嵌入永中Office，只需要在网页代码中加入<applet>标识即可，如下所示：

<HTML>

<HEAD>

<TITLE>永中Office 嵌入浏览器实例</TITLE>

</HEAD>

<BODY>

<P>

<APPLET code="AppletDemo.class" width="1024" height="768"></APPLET>

</P>

</BODY>

</HTML>

注意，确保该网页文件和 AppletDemo.class文件在同一目录下。

**打开网页文件**

此时已是万事具备，只需要双击打开网页运行，查看运行即可。下图是在Windows XP下的运行截图。

图9-9

在嵌入网页的永中Office中，可以同运行在桌面的永中Office操作一样，永中Office所提供的功能，绝大多数也适用与Applet。

#### 9.2 JavaScript与YoZo\_Applet互动

我们知道，永中Office通过Applet方式，能非常方便地嵌入浏览器（为方便起见，我们称嵌入网页的永中Office为“YOZO\_Applet”）。现在的网页，大量用到动态元素，让网页更生动，也实现了网页与用户的交互。JavaScript作为最早的一种网页脚本语言，以其功能强大、使用灵活，得到了大量地应用。JavaScript能和嵌入网页的永中Office互动吗？答案是肯定的。下面我们就以一个最简单的例子来讲述如何实现 JavaScript与YOZO\_Applet互动。

图9-10

上面这附图是我们设计的一个最简单的演示样例。我们设想当用户点击网页上的“插入忘本”按钮时，YOZO\_Applet能够在文档开始处插入一段文本，如下图所示：

图9-11

这是如何实现的呢？我们先来看网页源代码。

<HTML>

<HEAD>

<TITLE>永中Office 嵌入浏览器实例</TITLE>

</HEAD>

<script language="JavaScript">

<!--

//利用JavaSript函数调用Applet中的insertText()方法

function insertText()

{

}

// -->

</script>

<body>

<form method="POST" action="" >

<p>

<font size=4><center><b>网页Form控件与YOZO\_Applet互动示例</b></center></font>

</p>

<center>

<input type="button" value="插入文本" name="insertion" onclick="insertText();">

<p>

<applet name="postil" code="YOZOApplet.class" width="1024" height="768" >

</applet> </p>

</form>

</body>

</html>

你可能马上注意到这个网页中包含JavaScript程序和Form控件，Form控件是一个按钮，其显示值是“插入文本”，其动作是调用JavaScript编写的“ insertText(); ”方法。

这里需要解释一下，document是任何一个网页都有的“文档对象”， postil是YOZOApplet.这个applet，postil是document中的一个对象，所以用 document.postil就可以调用 YOZO\_Applet了。 很自然的，你可能马上就会想 insertText()是 postil这个applet中的一个方法了，也就是YOZO\_Applet的一个方法，不错，就是这样。看看 YOZOApplet.class源代码：

/\*\*

\* YOZOAppletDemo.java

\* Created on 2003 年4 月9 日, 上午10:30

\*/

import application.Application;

{

/\*\* Creates a new instance of YOZOAppletDemo \*/

public YOZOApplet()

{

}

public void start()

{

//创建YOZO\_Applet 实例

Application.createInstance(this);

// 隐藏菜单和工具条

Application.setToolbarVisible(BarConstants.MENUBAR\_INDEX, false);

Application.setToolbarVisible(BarConstants.STANDARD\_INDEX, false);

Application.setToolbarVisible(BarConstants.FORMATTING\_INDEX, false);

Application.setTaskPanelVisible(GlobalConstants.NAVIGATION\_PANE, false);

// 取得YOZO\_Applet 实例

Container content = Application.getInstance();

Container contentPane = getContentPane();

//添加文件

Application.getWorkbooks().addWorkbook(GlobalConstants.WORDPROCESSOR);

// 将YOZO\_Applet 添加到Applet 面板中

contentPane.add(content);

}

/\*\*

\* 插入文本。

\*

\*/

public void insertText()

{

//获取文档

Document doc = Application.getWorkbooks().getActiveWorkbook().getDocuments()

.getActiveDocument();

//获取文档中的第一节

Section section = doc.getSection(0);

BaseText baseText = section.getBaseText();

FontAttribute font = new FontAttribute();

font.setBold(true);

font.setForeground(Color.red);

font.setFontSize(16.0);

//插入文字

baseText.insertText(0, " this is a test!", font);

}

}

## 第十章 嵌入Java应用程序样例

永中Office可以Applet的方式嵌入浏览器，也可以Panel的方式嵌入Java应用程序中。

本章简要介绍在Java应用程序中嵌入永中Office的应用方案。

我们通过下面这个简单的例子来描述如何把永中Office嵌入Java应用程序中。

//OfficeFrame.java

/\*

\* OfficeFrame.java

\*/

import javax.swing.\*;

import java.awt.\*;

import application.\*;

import application.constants.\*;

import application.workbooks.\*;

import application.exceptions.\*;

/\*\*

\*

\*/

public class OfficeFrame extends JFrame implements MacroConstants

{

private JLabel label;

private JButton button;

private JPanel panel;

/\*\* Creates a new instance of FrameTest \*/

public OfficeFrame()

{

super();

init();

// 设置JFrame的尺寸

this.setSize(800, 600);

}

private void init()

{

// 设置按钮属性

JPanel panel = new JPanel();

button = new JButton("新建一个文字处理文档");

panel.add(button);

this.getContentPane().add(panel, BorderLayout.NORTH);

// 将永中Office的实例嵌入JPanel面板

Application.createInstance(this.getContentPane());

// 隐藏导航面板

Application.setTaskPanelVisible(GlobalConstants.CURRENT\_PANE,false);

// 隐藏系统的工具条

Application.setToolbarVisible(BarConstants.STANDARD\_INDEX, false);

Application.setToolbarVisible(BarConstants.FORMATTING\_INDEX, false);

Application.setToolbarVisible(BarConstants.DRAWING\_INDEX, false);

Application.setToolbarVisible(BarConstants.MENUBAR\_INDEX, false);

Application.setToolbarVisible(BarConstants.TASKBAR, false);

// 将JPanel面板嵌入到Frame中

panel = (JPanel) Application.getInstance();

this.getContentPane().add(panel, BorderLayout.CENTER);

// 设置默认关闭方式

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

// 给按钮加上事件监听器

{

{

openFile(evt);

}

});

}

// 事件处理方法

{

// 新建一个空白文字处理文档

Application.gotoProduct(1);

}

public static void main(String[] args)

{

new OfficeFrame().setVisible(true);

}

}

运行方法为先编译上述java文件（确保永中Office已经安装了）

javac -classpath %YOZO\_HOME%/Yozo\_Office.exe;. OfficeFrame.java

其中YOZO\_HOME为安装永中Office时系统自动设置的永中Office的路径；

用下列方式运行：

java -classpath %YOZO\_HOME%/Yozo\_Office.exe; OfficeFrame

运行状况如下图所示：

点击“新建一个文字处理文档”按钮，将新建一个空白的文字处理文档，如下图：

此时可以向在桌面操作永中Office一样进行各种编辑了。

## 附录一 永中Office 二次开发技巧集

本文根据日常开发实践，收集整理了利用永中Office进行二次开发的一些经验技巧，现共享给大家，希望对大家的开发有所帮助。

[**其他应用程序如何调用永中Office？**](#first)

[**使用线程解决同步问题**](#sec)

[**在文档中快速定位**](#thir)

[**为表格设置名称**](#fou)

[**随时随地获得光标的偏移量**](#five)

[**地下工作者——让YOZO\_Applet隐藏工作**](#six)

**其他应用程序如何调用永中Office？**

我们知道，永中Office的启动方式是java命令通过jar参数调用Yozo\_Office.jarr文件，所以只需要执行相应的命令行命名，就可以启动永中Office。即：

%JAVA\_HOME %\ java -jar %YOZO\_HOME%\Yozo\_Office.exe [filename.\*]

其中

%YOZO\_HOME%为永中Office的安装路径，默认为C:\Progra~1\Yozosoft\Yozo\_Office；

[filename.\*]为可选项目，如果存在，表示在启动永中Office的同时，打开 filename.\* 文件。 filename.\*可以是永中自己的eio文件，也可以是doc、xls、ppt、rtf、txt、html等永中Office支持的文件格式。

下面我们以一个简单的java应用程序为例，演示在java应用程序中调用永中Office并打开C盘的一个名为test.doc文件。

public static void main(String[] args)

{

Runtime rt = Runtime.getRuntime();

try

{

rt.exec("C:\\Progra~1\\Java\\j2re1.6.0\_21\\bin\\java -jar C:\\Progra~1\\Yozosoft\\Yozo\_Office\\Yozo\_Office.exe d:\\test.doc");

} catch ( ie) { }

}

用其他语言开发的应用程序也可采用类似的方法调用永中Office。

**使用线程解决同步问题**

在调用永中Office时，特别是初始化永中Office，如果同时打开一个文件，并对该文件进行一些操作，有时可能产生一些意外的效果。比如，初始化后，通过Workbook.open()方法打开一个eio文件（假设该文件包括电子表格和文字处理应用，打开后的界面应为电子表格应用），如果此时紧接着调用Application.gotoProduct(1)方法切换到文字处理应用，则可能产生这种效果：最后是进入了文字处理应用界面，但该界面内容是空的。分析原因就是，打开eio文件还没有结束，就执行Application.gotoProduct(1)方法，则系统新建了一个空白的文件。这也就是一个方法/过程还没有执行完之前，不能执行下一个方法/过程，如何解决此类问题呢？

可以使用线程方法解决同步问题。如以下所示。

import ;

import application.Application; //导入永中Office的类

import ;

/\*\*

\*

\* @author

\*/

public class YOZOApplet extends

{

public YOZOApplet()

{

}

public void start()

{

container = getContentPane();

container.setLayout(new ());

container.add(panel);

Application.createInstance(panel);

panel.add("Center",Application.getInstance());

//Application.displayNavigationPane(false) ;

(new Runnable()

{

public void run()

{

Application.getWorkbooks().openWorkbook("D:\\test.eio", 0,1);

}

});

load();

}

private void load()

{

(new Runnable()

{

public void run()

{

Application.gotoProduct(1);//切换到文字处理界面

}

});

}

panel = new ();

}

在上例中，通过调用 invokeLater方法实现方法的同步。

从上例也可以看出，我们应该尽量将与视图操作有关的动作，放置到run方法中处理，而初始化永中Office的实例的操作则应放置在run方法之外，因为run方法会常驻内存，如此可减少内存占用，提高性能。

**在文档中快速定位**

有时候我们需要在文字处理文档中快速定位到某个特定位置，一般有如下几种方法：

一、事先在该位置定义书签（Bookmark），然后通过定位API定位到相应的位置，例如，如下语句定位到名称为"bmName1"书签：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Document doc = workbook.getDocuments().getActiveDocument().

BaseText bt = getSection(0).getBaseText();

bt.gotoPosition(FindConstants.BOOKMARK,FindConstants.PREVIOUS,1,"bmName1");

二、如果该位置是表格，则可通过先定位到表格，在定位到表格单元格的方式实现定位：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Document doc = workbook.getDocuments().getActiveDocument().

BaseText bt = getSection(0).getBaseText();

Table table1 = bt.getTable("table1");

String str = table1.getCellValue(1,1);

三、通过超级链接实现快速跳转到特定位置，超级链接配合书签定位，不需要编写代码即可实现。

**为表格设置名称**

在永中Office文字处理中，可以通过如下语句获得名称为“table1”的表格：

Workbook workbook = Application.getWorkbooks().getActiveWorkbook();

Document doc = workbook.getDocuments().getActiveDocument().

BaseText bt = getSection(0).getBaseText();

Table table1 = bt.getTable("table1");

但是一般情况下，手动插入的表格是没有名称的，也无法设置名称。我们可以通过永中Office提供API为表格设置名称。这个API就是Table.setName(String name)。

**随时随地获得光标的偏移量**

文字处理是一种流式灌排体系，其内容多由一个接一个的文字或者符号组成，这些元素的排列是通过Offset，也就是“偏移量”进行定位的，当前光标的活动位置更是与偏移量密切相关。如何获得当前光标的具体位置呢？永中Office的文档类Document提供getOffice()方法，可以即时反馈Offset的值。

下面我们开发一个小程序，让我们即时获得当前光标的Offset值。

这个例子直接在永中Office内部开发。

按Alt＋F11，进入永中Office的宏编辑器。插入一个UserForm，在其中添加一个EIOLabel控件和EIOButton控件，双击EIOButton控件，进入编程环境，填写如下代码：

Document document = Application.getWorkbooks().getActiveWorkbook().getDocuments().getActiveDocument();

long offset = document.getOffset());

EIOLabel1.setCaption("当前光标偏移量为："+Long.toString(offset);

按F5运行该UserForm，点击EIOButton控件，则EIOLabel控件显示“当前光标偏移量为：100”等信息。

但是此时不能改变光标的位置，重新获得光标的偏移量。可以让光标随意改动吗？可以的，只要修改UserForm的属性ShowModal即可。该属性默认为ture，修改为false后，则运行UserForm时，仍然可以操作光标在文档中移动，或者编辑内容等，点击EIOButton控件可获得光标的即时位置信息。

同理，Document类还提供了setOffset(long position)方法，也就是将当前光标设置到偏移量为 position的位置。

**地下工作者——让YOZO\_Applet隐藏工作**

有时候，我们只需要调用永中Office进行文件打印、文件转换等方面的工作，不希望永中Office的界面出现，此时可以设置applet的高度和宽度为0，达到隐藏YOZO\_Applet的目的。

下面的例子演示了将Doc文件转换为永中Office手写批注文件Ept格式的过程。

import application.Application; //导入永中Office的类

import java.applet.\*;

import java.awt.\*;

/\*\*

\*

\* @author

\*/

{

/\*\* Creates a new instance of YOZO\_Applet \*/

public YOZOApplet()

{

}

private void createInstance()

{

//创建永中Office的applet实例

Application.createInstance(this);

getContentPane().add("Center", Application.getInstance());

}

public void start()

{

createInstance();

}

public void convert2EPT()

{

Application.getWorkbooks().openWorkbook("D:\\test.doc", 0, 1);

Document doc = Application.getWorkbooks().getActiveWorkbook().getDocuments().getActiveDocument();

doc.saveToPostilFile("d:\\ttt.ept");

}

}

在网页中，我们将Applet的大小设置为0，也就是不可见，点击按钮时仍然调用Applet的方法，实现文件格式的转换。

<head>

<meta http-equiv="Content-Type" content="text/html; charset=gb18030">

<title>文件转换</title>

<script language="JavaScript">

<!--

function doc2ept()

{

//将生成的ept文件保存到指定位置

}

// -->

</script>

</head>

<body>

文件转换

<applet code="YOZOApplet.class" codebase="." width="0" height="0" name="office"/>

<input type="button" value="上传批注文件" name="convertfile" onclick="doc2ept();">

</body>

</html>

## 附录二 永中Office二次开发FAQ

下面列举一些永中Office二次开发中遇到的FAQ:

**一、**[**二次开发基础**](#基础)

**二、**[**二次开发技术**](#技术)

**三、**[**二次开发部署**](#部署)

**四、**[**二次开发应用**](#应用)

**二次开发基础**

1、什么是Office二次开发，二次开发能做什么？

答：结合自己的需求，利用Office软件所提供的各种应用接口，来进行相互的嵌入或调用，充分开发应用程序潜在功能，从而完成一定的功能扩展，获得使用上最大限度的满足。即利用编程语言扩展宿主的功能、集成不同类型的数据以实现办公软件在具体事件中的解决方案。利用二次开发，可以完成很多具有实际意义的工作，例如通过电子表格来构建一个进销存管理程序，分类汇总，自动生成各种报表，让Office办公之余，也能成为会计专业软件。或者人员管理系统，利用永中Office的集成，方便准确地管理相关数据，可以与同类的管理软件相蓖美。或者，还可以进行教学课程的安排，各种方案的整合，实行全自动管理……

2、永中Office二次开发有什么特点？

答：作为国内最大的Office开发团队，拥有自主开发知识产权，以强大的研发实力作保障，可根据客户需求量身定制；提供本地化的技术支持服务；该产品可跨平台使用，可在多种平台上进行开发；宏语言为Java，能与Java开发的业务系统无缝集成，也方便其他语言调用。

3、利用永中Office进行二次开发，需要掌握哪些技能？

答：首先假定你熟悉使用Office软件，了解基本操作，这是前提；其次你明白二次开发是怎么一回事，知道你想要做什么；第三是你有编程基础，最好懂Java编程（不懂也行，只要有其他语言编程经验，学习学习Java编程，也能赶上）；最后要看你的项目的实际情况，可能需要了解网页技术，如HTML、JSP、J2EE等技术，还有数据库技术等。

4、我会用微软Office做二次开发，换用永中Office难吗？

答：不会太难，总的原理是一样的，就是语言方面有所差异。微软Office用VBA做宏开发，永中Office用Java作为宏语言。开发语言虽然不同，接口、方法却有很多相似之处，所以说，熟悉微软Office二次开发的，经过一段时间的语言转换的适应，能够很快熟悉永中Office的二次开发技术。

5、我是初学者，应该如何学习永中Office的二次开发？

答：假定你会Java编程，恭喜你，永中Office二次开发的大门已经为你敞开！你可以先看看《永中Office二次开发技术手册》对永中Office二次开发的总体介绍；然后你想亲自动手操作了吧，你可以先体验一下永中Office的录制宏功能，看看录制和运行的效果，以及宏编辑器中的宏代码；你也可以插入一个UserForm，放置几个控件，编写一些简单的代码，体验在永中Office宏编辑器中编程的乐趣。之后你建议你看看“永中Office二次开发技术手册”中介绍的基础样例，安装样例中的说明，自己动手试验试验，你会发现自己已经掌握永中Office二次开发的基础技术了。永中Office博大精深，要想全面细致了解，你还是多试验，多研究其API和文档吧。

假如你还不会Java，建议你先学习学习Java编程，你甚至可以直接在永中Office的宏编辑器中学习。

?6、永中Office能嵌入浏览器吗？能嵌入Linux下的Mozila浏览器吗？

答：可以。永中Office支持以Applet（小应用程序）的方式嵌入浏览器（浏览器需要支持Java插件），包括Linux下常用的Mozila系列浏览器。详细信息请参阅有关永中Office嵌入浏览器的技术文章。?

7、永中Office能够作为控件嵌入应用程序中吗？

答：目前，永中Office可以以Panel的方式加入到Java应用程序中，该应用程序中的其他组件可通过永中Office的接口进行调用。详细信息请参阅有关永中Office嵌入应用程序的技术文章。?

8、作为Java应用程序，永中Office提供JavaDoc API文档吗？在哪里可以获取？

答：永中Office提供了详细的中文API文档，在永中Office产品光盘的API\_doc目录下，打开其中的index.html文件即可查阅全部包、接口和类，以及相关的方法等。

?9、 JRE与JDK有什么区别？

答：JRE是Java Runtime Environment 的缩写，即“Java运行环境”；JDK是Java Development Kit 的缩写，即“Java开发包”。JRE提供Java应用程序运行的基础，安装永中Office时会自动检测并安装相应的JRE；JDK除了包括JRE外，还包含很多Java应用程序开发工具，如javac编译工具等，以及大量的样例和文档。

?10、如何编译java程序？

答：在命令行方式下，编译java程序的命令是javac，编译命令为：javac <option> <source files>。例如，编写与永中Office有关的java应用程序YOZOApplet.java后，编译命名为：javac -classpath %YOZO\_HOME%\Yozo\_Office.exe;. YOZOApplet.java ， 其中 %YOZO\_HOME%为永中Office的安装路径，因为YOZOApplet.java文件中引用了永中Office开发包，故此处设置了 -classpath参数。更详细的命令可能为：

c:\j2sdk1.6.0\_21\bin\javac -classpath "c:\\program files\\Yozosoft\\Yozo\_Office\\Yozo\_Office.exe" YOZOApplet.java

?11、使用永中Office做开发，配置复杂吗？

答：不是很复杂，因为永中Office在安装的过程中已经将有关配置自动设置好了,大大简化工作，提高效率。如在系统的环境变量中添加YOZO\_HOME，并设置为安装的路径，方便及时的引用。

?12、在JBuilder开发环境中如何导入永中Office开发包？

答：JBuilder是功能强大的IDE开发工具，要想在其中导入永中office开发包，请按如下步骤：在JBuilder的菜单栏里选择**ToolsConfigure LibrariesNew LibraryAdd**，添加永中office安装目录下的Yozo\_Office.jar包（默认是c:\program?files\Yozosoft\Yozo\_Office\Yozo\_Office.jar



注:此处的Yozo\_Office.jar是把Yozo\_Office.ese修改扩展名所得），这样永中office开发包就被导入，只要在新建project时，在add libraries里加入之前导入的永中开发包即可。

?13、在Forte、NetBean或者Sun ONE开发环境中如何导入永中Office开发包？

答：在Explorer里，右击FileSystems，在出现菜单里**mountArchive Files**，选择



永中office安装目录下的Yozo\_Office.jar包（同上）即可。

14、在Eclipse开发环境中如何导入永中Office开发包？

答：菜单栏**PreferencesJavaBuild PathUser LibrariesNewAdd Jars** （同上）即可。?



**二次开发技术**

15、网页中的YOZO\_Apple无法显示，怎么办？

答：首先检查浏览器是否安装并支持Java插件，接着看java.policy文件是否修改好了，最后如果Applet仍然显示一个红“X”符号，则可能是你的机器上有多种版本的JRE或者JDK，建议你按照“先装JDK，再装Java IDE，最后安装永中Office”的顺序重新安装永中Office。

?16、如何调试运行在浏览器中的Applet？

答：在浏览器中运行Applet时，系统任务栏会显示一个Java图标，右键点击，选择“打开控制台”，出现“Java控制台”信息框，其中有很多命令，如果Applet有异常，其异常信息也会显示在“Java控制台”中，这些信息可用于调试Applet。

?17、我想调用某个功能，但是不知道对应的API是什么，怎么办？

答：提供两种办法，一是直接到永中Office中执行录制宏，一边录制，一边操作你要调用的功能，最后到宏编辑器（Alt+F11切换)中查看该宏的源代码，可以找到你要得相关API信息；另外一种办法是查看API文档，打开index.html文件，点击其中的“Index”可以看到按照从A到Z排列的全部方法或者属性，此时按Ctrl＋F快捷键，调出浏览器的查找对话框，输入你要查询API的相似信息，查找即可。如你想知道打开文件是什么API，在查找对话框中输入“open”，逐个查找到Workbook.openWorkbook(String, int, int)，就是打开文件的API了。

?18、永中Office支持控件吗？可对控件进行编程吗？

答：支持。永中Office提供了一套自己的控件，包括命令按钮、复选框、单选按钮、组合框、文字框等，打开“控件工具箱”，选择任一控件，可插入到文字处理正文或者电子表格单元格中。双击该控件，进入宏编辑器代码编辑区，可进行编程；回到文字处理或电子表格，点击“控件工具箱”第一个按钮“退出设计模式”，控件进入运行模式，点击控件，将开始执行程序。

?19、 C语言可以调用永中Office的API吗？其他语言呢？

答：永中Office 2010开发版支持C语言调用永中Office接口，同时也支持其他的主流语言,如VB，Delphi等。

?20、永中Office支持痕迹保留吗？

答：永中 Office提供了痕迹保留功能，也就是我们常说的“修订”功能。永中Office 文字处理的修订功能有如下特点：可记录修订人进行的插入内容、删除内容和改变内容格式的操作；对插入的内容可通过无变化、加粗、斜体、下划线和双下划线等方式标记，并可自动设置颜色或自定义颜色显示；对删除的内容可通过隐藏、删除线和双删除线等方式标记，并可自动设置颜色或自定义颜色显示；对格式的改变可通过无变化、加粗、斜体、下划线和双下划线等方式标记，并可自动设置颜色或自定义颜色显示；对修订过的行可设置无、左侧框线或右侧框线标记，并可自动设置颜色或自定义颜色显示；可根据修订人信息自动设置修订格式，如颜色设置等；可显示文档的原始状态、修订状态和最终状态；可对修订内容进行审阅，选择接收修订或者拒绝修订。

21、永中Office二次开发支持与数据库连接吗？

答：当然支持。可以通过加载数据库驱动，如ODBC、JDBC等，并透过JSP、ASP、PHP等页面技术，直接与数据库连接并操作，支持当前各大主流数据库，Ms SQLServer、Oracal、DB2、Mysql、Sybase等。

22、永中Office可以转换文档文件为PDF文件并提供相应的API吗？

答：可以。永中Office为此提供了相应的API，在workbook类中提供了一个saveAs(String newName)方法,可以将当前文档存成pdf，html等格式，完成转换。

23、永中Office可以转换文档文件为图片文件并提供相应的API吗

答：永中Office为此提供了三个相关的API：convertToJPEG(int pageIndex, String fileName)，其功能是将页面视图中的第 pageIndex页转换为名为 fileName的JPG图片； convertToJPEG(String fileName)，其功能是把所有页面视图转换成一系列JPEG图像文件；convertToOneJPEG(String fileName)，其功能是把所有页面视图转换成一JPEG图像文件。

24、永中Office电子签章使用的证书和私钥格式是什么？

答：电子签章是永中Office的特色功能。所使用的证书是X.509证书，公钥文件格式为\*.der，密钥文件格式为\*.pfx。完全符合证书认证机构（CA）标准，能广泛地被流转使用，通行无阻。

25、永中Office支持Lotus Domino/Notes调用吗？

答：Lotus Domino/Notes作为长期存在的办公软件，具有相当数量客户群。永中Office支持它的调用，并能与之高效工作。

**二次开发部署**

26、对于嵌入浏览器中的永中Office系统，部署时，客户端需要安装永中Office吗？

答：需要安装永中Office。理论上，对于嵌入浏览器中的永中Office，完全支持客户端通过服务器上装载的永中Office来进行操作。但由于Office软件是一款功能强大的集成程序，文件较大，以目前的带宽，下载后再运行，会需要一定的时间，从而带来不便。因此当前所有嵌入浏览器中的Office系统，部署时，都建议用户在客户端安装Office软件，永中Office亦不例外。

27、嵌入网页的永中Office小应用程序安全吗？

答：永中Office是一款完全拥有自主知识产权的国产软件，较国外的同类产品，安全性不言而喻；而且，永中Office所采用的编程语言Java，其安全性是目前众多编程语言中最优秀的一种，特别是网络安全做了周密的考虑。通过字节码验证、内存调用方式、资源使用权限等进行多层次的安全管理。具体来说，永中Office作为Applet的方式嵌入在浏览器里，Web浏览器启动的Java应用通常会建立不同的applet类载入程序对象，因而不同来源的类文件由不同的类载入程序对象装入，并被放入主Java应用内不同的命名空间中。因为不同来源的applet类文件放在隔离开的命名空间中，所以破坏性的代码就不能与从其他来源下载的代码直接接触。

**二次开发应用**

30、永中Office有哪些成功的二次开发应用案例？

答：随着永中Office的不断发展和完善，二次开发全面出击，以强大的二次开发接口，极佳的扩展性，吸引了各行业的目光，并与多家单位共同合作。如西安科技局——基于NC的办公自动化，永中Office被嵌入其信息化系统中，作为公文处理、电子表格、报表制作等办公自动化平台；同时，针对政府部门的电子政务建设，永中Office专门提供了公文制作、痕迹保留、公文流转、电子签章、手写签名和批注插件等功能，形成有机整体，具有广泛的应用价值。