第3章 WPS及其自动化处理

CHAPTER 3

3.1 WPS 介绍

WPS Office 是由金山软件股份有限公司自主研发的一款功能强大的办公软件套装，它具有以下特点：

（1）多平台支持。WPS Office 支持多个操作系统，包括 Windows、Linux、macOS、Android 和 iOS 等，可满足不同用户的使用需求。

（2）功能全面。WPS Office 提供了文字处理、表格计算、演示制作、PDF 阅读等多种办公软件常用功能，能够满足日常办公的各种需求。

（3）内存占用低。WPS Office 在运行效率上表现出色，具有内存占用低的特点，即使在配置较低的计算机上也能顺畅运行。

（4）云服务与协作。WPS Office 支持云存储服务，可以实现多人在线协同办公和实时协作，方便团队成员之间的文件共享和编辑。

（5）安全性高。WPS Office 独有内容级安全保护措施，可确保文档的安全性，全程留痕可追溯，保护用户数据不被非法访问或泄露。

（6）企业协作管理。登录企业账号后，用户可以体验 WPS 提供的企业协作管理能力，进一步提升团队工作效率。

（7）兼容性强。WPS Office 能够兼容多种文件格式，包括 Microsoft Office 系列软件的文件格式，方便用户在不同软件间进行文件转换和使用。

（8）界面友好。WPS Office 的用户界面设计简洁直观，便于用户快速上手操作。

（9）打印功能。WPS Office 提供了丰富的打印功能和输出格式控制，使用户可以轻松打印文档。

WPS 还有一个重要的特点，对于办公自动化或者说提高办公效率意义重大，那就是支持 JavaScript 脚本，支持使用 JavaScript 进行宏编程，这一变化对于习惯使用 JavaScript 的用户来说是一个好消息，JavaScript 会大大降低学习成本，提高办公效率。与 VBA 相比，JavaScript 具有以下特点：

（1）语言表达能力强。JavaScript 作为一种编程语言，其表达能力比传统的 VBA（Visual Basic for Applications）更强，这使得开发者可以使用更加先进和强大的编程特性来完成复杂的自动化任务。

(2) 跨平台支持。WPS 对 JavaScript 的支持是跨平台的,这意味着不仅可以在 Windows 系统的 WPS 中使用,还可以在 Linux 等其他操作系统的 WPS 中使用 JavaScript 进行宏编程。其大部分功能在银河麒麟 V10 桌面系统中均可使用。

(3) 丰富的资源。由于 JavaScript 是 Web 开发中广泛使用的语言,开发者可以利用大量的现有资源和第三方库来辅助开发,尽管目前 WPS 中调用第三方库的方法还不是很明确,但可以参照第三方库对其部分功能进行二次开发。

(4) JSAPI 接口。WPS 为 JavaScript 宏提供了 JSAPI 接口,这些接口与 VBA 宏有所差异,但大部分 API 的使用是正常的,在进行开发时,需要注意这些差异以进行有效的开发。其官方文档比较全面,网上也有相关的教程,为我们学习开发提供了很大便利。

(5) 性能考虑。WPS 的 JavaScript 宏被隔离在独立的进程中运行,它与 WPS 主进程的关系类似于客户端与服务器端的关系。虽然可以通过数据共享交换数据,但这种方式可能会受到性能上的限制。

(6) 持续改进。WPS 官方正在不断地改进 JavaScript 宏的功能,尽管目前可能还存在一些小问题(bug),但官方已经在积极解决这些问题,以提供更好的用户体验。

总的来说,WPS Office 是一款适合个人和企业用户的办公软件,无论是用于日常的文档编辑、数据处理,还是用于团队合作和项目管理,都能提供高效便捷的工作体验。WPS 对 JavaScript 的支持为文档自动化和宏编程带来了新的可能性,特别是对那些熟悉 JavaScript 语言的开发者来说,这是一个值得尝试的新功能。

3.2 WPS 表格及其自动化处理

3.2.1 WPS 表格介绍


视频讲解

WPS 表格(WPS spreadsheet)是一款功能强大的电子表格软件,它是 WPS Office 套件的重要组成部分。具有以下主要特点和功能:

(1) 界面友好。WPS 表格采用了简洁明了的界面设计,用户可以轻松找到所需的功能。同时,它还支持自定义工具栏和快捷键,方便用户根据自己的习惯进行操作。

(2) 数据处理。WPS 表格提供了强大的数据处理能力,支持多种数据排序、筛选和查找方式。用户可以轻松处理大量数据,提高工作效率。

(3) 公式和函数。WPS 表格支持丰富的公式和函数,包括常用的数学、统计、财务等方面。用户可以通过简单的拖曳操作,快速完成复杂的计算任务。

(4) 图表功能。WPS 表格提供了多种图表类型,如柱状图、折线图、饼图等,用户可以根据需要轻松创建图表,直观展示数据。

(5) 数据分析。WPS 表格内置了数据分析工具,如数据透视表、条件格式等,帮助用户深入挖掘数据价值,发现潜在规律。

(6) 协作共享。WPS 表格支持多人协同编辑,用户可以邀请团队成员共同编辑同一个文档,从而提高团队协作效率。此外,WPS 表格还支持云存储,用户可以将文件保存在云端,实现跨设备访问和共享。

(7) 兼容性。WPS 表格兼容 Microsoft Excel 的文件格式,用户可以无缝切换使用。同

时，WPS 表格还支持导入和导出 CSV、TXT 等多种文件格式，方便用户在不同平台之间进行数据交换。

总之，WPS 表格是一款功能强大、易用的电子表格软件，适合个人和企业用户使用。通过使用 WPS 表格，用户可以高效地处理数据、创建图表、分析数据，满足各种办公需求。

在学习使用 WPS 表格前，我们先了解一下其在国产麒麟 V10 操作系统中运行的基本结构层次，如图 3.1 所示。

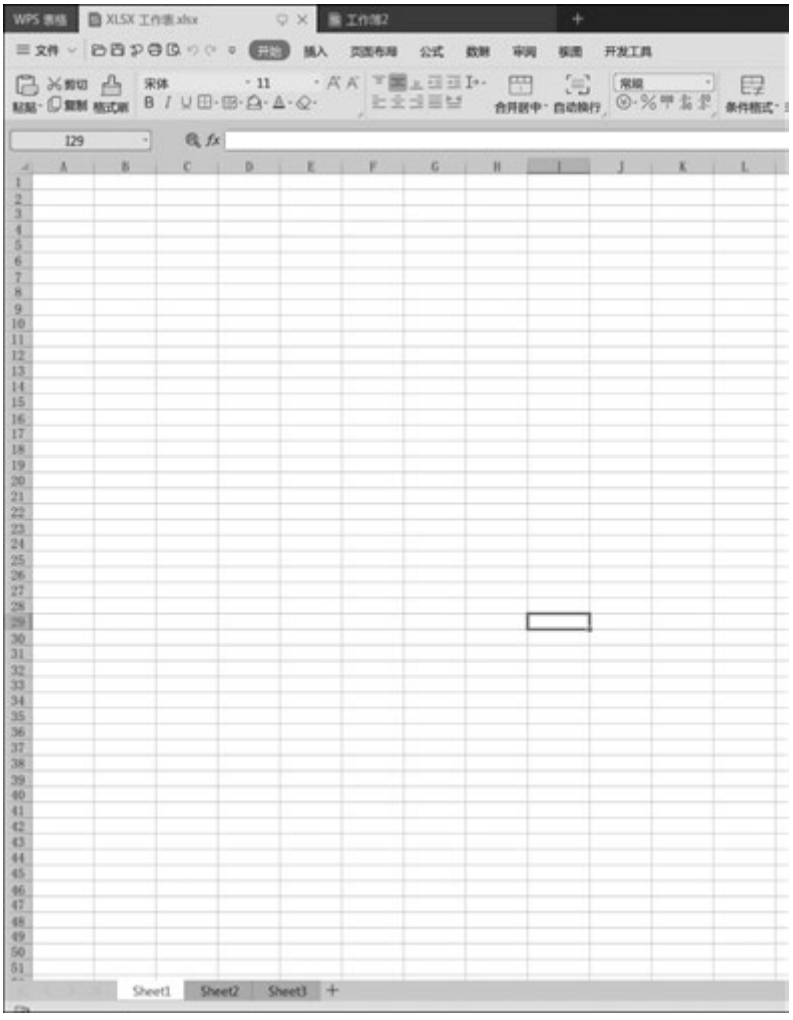


图 3.1　WPS 表格结构层次

如图 3.1 所示，WPS 文件的结构层次为：打开的整个 WPS 程序称为 Application，里面的每一个. xlsx 文件为一个 Workbook(工作簿)，. xlsx 文件中的每一个表格 Sheet 指的就是 Sheet(工作表)，以 A，B，C，…标记列，以 1，2，3，…标记行，每一行和每一列的交叉处为一个单元格，清楚这些对后面的脚本编程或者 Python 编程很有用，这是实现表格办公自动化的基础。

在自动化办公的过程中，对行、列、单元格的操作就是定位到对应位置进行读写的过程，对对象(比如 chart)进行操作时，要知道插入之前应定义好使用的数据和 chart 类型，这非

常关键，还要了解使用的类库对操作的支持程度，这部分内容在本节的代码操作表格中有详细介绍。

3.2.2 使用 JavaScript 编写脚本

WPS 自带了宏编辑器来编写宏代码，但是在银河麒麟 V10 系统使用的 WPS 2019 中，宏编辑器没有调试代码的功能，这一点给宏代码编写带来了很多不便，升级的 WPS 365 可以正常使用调试功能。本节以 WPS 表格为载体，介绍使用 JavaScript 编程语言进行宏编程的相关知识，后面 Word、PPT 在使用宏编程方面与表格类似，但使用频度上不如表格，故不再介绍。

1. 宏编辑器介绍

WPS Office 宏编辑器是一个为二次开发者设计的编写和编辑 JavaScript 代码来调用或扩展 WPS 功能的程序。除了可通过编写 JavaScript 脚本来加速执行日常任务外，还可以使用 JavaScript 为 WPS Office 添加新功能，或以特定于业务需要的方式来提示文档用户并与之交互。

宏编辑器为用户提供了常用的代码编辑、录制宏、运行宏、自定义公式、设计 UI 控件、执行宏以及调试器等功能。开发者可以通过代码编辑器，基于 JavaScript 的语法，把想要执行的动作编写成宏，也可以通过录制宏功能，把常见的动作自动转换成宏代码，制作出包含宏的文档。文档使用者在有工作需要时，可以打开宏对话框去运行宏，也可以通过结合可视化的控件，在合适的控件响应时执行宏。在表格中，还可以通过编写设计宏来扩展公式。

但是，在国产银河麒麟系统中，WPS 2019 的宏编辑器并非完整版本，功能并不全面，比如，没有重要的调试功能，对 JavaScript 支持并不好，这就要求我们在编写代码时要多测试，可以多使用 MsgBox 或者 Debug 进行调试，其中，MsgBox 弹出对话框，Debug. Print 可打印内容到宏编辑器的立即窗口。

2. 使用录制宏来学习编写过程

刚开始使用宏来编写脚本时，建议通过录制宏来实现，打开一个 WPS 文件，单击菜单栏的“开发工具”→“录制新宏”图标，如图 3.2 所示。

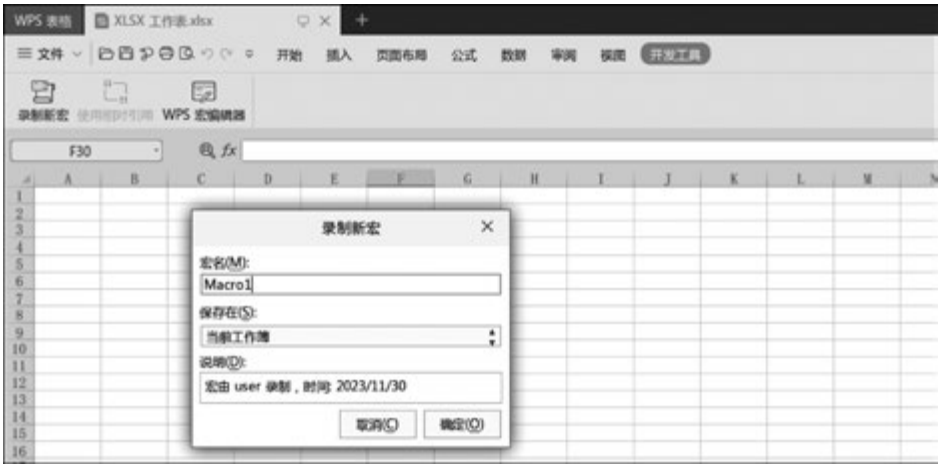


图 3.2 录制新宏

填写录制新宏的名字，这里使用默认名称(这不是一个好的习惯)，最好是设置一个有实际意义的名字，看到宏的名字就能知道该宏实现的功能。单击“确定”按钮即可开始录制，如图 3.3 所示。

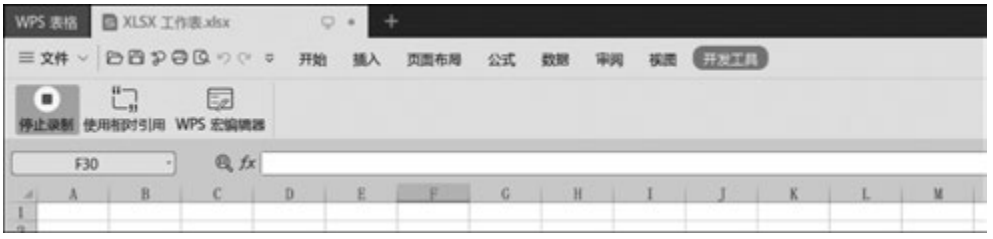


图 3.3　开始录制

接下来可以对表格进行各种操作，WPS 会记录使用宏的过程，记录每一次操作对应的脚本代码，通过宏的代码和操作进行对比，可以知道通过宏如何实现刚刚的操作，进而完成表格操作的自动化。比如，对表格进行冻结首行和将 A4～A7 这 4 个单元格合并居中的操作，录制后的代码为：

```
/**
 * Macro1 Macro
 * 宏由 user 录制,时间: 2023/11/30
 */
function Macro1()
{
ActiveWindow.Split = false;
Range("A4:A7").Select();
(obj=>{
obj.Merge(false);
obj.HorizontalAlignment = xlHAlignCenter;
})(Selection);
}
```

注意：录制宏时，尽量不要有多余操作，因为每一个操作都会产生对应的代码，如果操作过多，不便于我们理解宏产生的关键代码。录制的宏不一定仅对这个表格起作用，如果编写的宏代码由 Application 事件触发，那么该宏程序会对该 WPS 程序的所有 Workbook 起作用。当我们再次打开一个表格时，程序默认不会另外创建一个进程 Application，而是在已有的 WPS 应用程序中打开一个新的 Workbook，所以，宏代码对整个 Application 有效。如果编写的宏代码是 Workbook 事件触发，那么仅仅对该 Workbook 有效。

3. 使用官方文档学习编写过程

官方文档地址 https://open.wps.cn/docs/client/wpsLoad，选择“WPS 基础接口”，在这里可以查看 WPS JavaScript 相关接口，如图 3.4 所示。

在使用 JavaScript 编写 WPS 宏时，与微软办公套件中使用 VBS 编写宏类似，可以使用 WPS 宏编辑器编辑界面和编写代码，其中界面的编辑是拖曳式、所见即所得的。

注意：当 WPS 宏编辑器集成了一个 V8 引擎的 JavaScript 运行时，支持大部分 ES6 语法，因此宏编辑器支持 JavaScript 标准内置对象，JavaScript 内置对象和浏览器的内置对象是不同的，WPS 宏编辑器集成的是 JavaScript 运行时，而不是浏览器，因此 WPS 宏编辑器不支持浏览器的内置对象。这在编写代码时尤其要注意！具体 API 参见 https://developer

图 3.4 WPS JavaScript 接口文档

. mozilla. org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects。推荐学习 ES6 语法的参考网站：https://www. w3cschool. cn/escript6/。

在进行宏编写代码前，不仅需要掌握基本的 JavaScript 知识，更重要的是要知道代码执行的时机，即要在什么时刻执行指定的功能，代码应该写在哪里，才能确保代码按照我们的要求和时机进行运行，如果是界面的响应函数，那么直接在对应的函数体内编写代码即可。

其实，WPS 中内置了部分事件，可以通过对 WPS 应用程序触发的事件响应函数中添加 JavaScript 代码进行处理。在通知型事件中，可以接收已经发生变化的消息，比如通过 WindowActivate 事件，可以对文档的切换做一些功能性处理；在询问型事件中，可以控制是否继续执行当前操作，比如通过 WorkbookBeforeClose 事件，可以取消文档的关闭操作。

在 WPS 表格中，在 WPS 宏编辑器左侧的“工程”栏选中“代码”，右击，选择“插入”→“模块”后查看内置的事件。WPS 表格内置事件主要包括两个层次：Application 事件和 Workbook 事件。每个层次的具体事件函数和参数如表 3.1 和表 3.2 所示。

表 3.1 Application 事件

事件函数	触发时机	参数说明
NewWorkbook()	新建表格时触发此事件，打开已有表格不触发	无，可以添加 Wb 参数，Wb：Object 表示新建工作簿对象
SheetActivate(Sh)	当激活任意一个 Sheet 时触发此事件	激活的 Sheet 对象

续表

事件函数	触发时机	参数说明
SheetBeforeDelete(Sh)	当删除任意一个 Sheet 之前触发此事件	删除的 Sheet 对象
SheetBeforeDoubleClick(Sh, rg,cancel)	当双击任意一个 Sheet 之前触发此事件	Sh：Object,双击的工作表对象；rg：Range 对象,双击区域所在的单元格对象；cancel：Object,如果设置其属性 Value 为 true,则取消此次双击
SheetBeforeRightClick(Sh, rg,cancel)	当右击任意一个 Sheet 之前触发此事件	Sh：Object,右击的工作表对象；rg：Range 对象,右击区域所在的单元格对象；cancel：Object,如果设置其属性 Value 为 true,则取消此次右击
SheetCalculate(Sh)	在任意一个 Sheet 内进行计算时触发此事件	Sh：Object,计算的工作表对象
SheetChange(Sh,rg)	当任意一个 Sheet 内单元格变化时触发此事件	Sh：Object,修改的工作表对象；rg：Range 对象,修改的单元格 Range 对象
SheetDeactivate(Sh)	当任意一个 Sheet 被切换到非激活状态时触发此事件	Sh：Object,被切换到非激活状态的工作表对象
SheetFollowHyperlink(Sh, Target)	当单击任意一个 Sheet 内的超链接时触发此事件	Sh：Object,超链接所在的工作表对象；Target：Hyperlink,代表超链接的目标
SheetSelectionChange(Sh, Target)	当任意一个 Sheet 内的选定区域发生变化时触发此事件	Sh：Object,右击的工作表对象；Target：Range,新选定的区域
WindowActivate(Wb,Wn)	当任意一个 Workbook 窗口被激活时触发此事件	Wb：Workbook,激活工作簿对象；Wn：Window,激活窗口对象
WindowDeactivate(Wb,Wn)	当任意一个 Workbook 窗口被切换到非激活状态时触发此事件	Wb：Workbook,被切换到非激活状态的工作簿对象；Wn：Window,被切换到非激活状态的窗口对象
WindowResize(Wb,Wn)	当任意一个 Workbook 窗口大小被调整时触发此事件	Wb：Workbook,调整窗口大小的工作簿对象；Wn：Window,调整窗口大小的窗口对象
WorkbookActivate(Wb)	当任意一个 Workbook 被激活时触发此事件	Wb：Workbook,被切换的工作簿对象
WorkbookAfterSave(Wb, Success)	当任意一个 Workbook 被保存后触发此事件	Wb：Workbook,保存的工作簿；Success：Boolean,如果保存操作成功,则为 true,否则为 false
WorkbookBeforeClose(Wb, Cancel)	当任意一个打开的 Workbook 被关闭之前触发此事件	Wb：Workbook,关闭的工作簿；Cancel：Object,如果设置其属性 Value 为 true,则取消此次关闭
WorkbookBeforePrint(Wb, Cancel)	当任意一个打开的 Workbook 被打印之前触发此事件	Wb：Workbook,打印的工作簿；Cancel：Object,如果设置其属性 Value 为 true,则取消此次打印
WorkbookBeforeSave(Wb, SaveAsUI,Cancel)	当任意一个 Workbook 被保存之前触发此事件	Wb：Workbook,保存的工作簿；SaveASUI：Boolean,如果为 true,则表示此次保存操作将会弹出保存或“另存为”对话框；Cancel：Object,如果设置其属性 Value 为 true,则取消此次关闭

续表

事件函数	触发时机	参数说明
WorkbookDeactivate(Wb)	当任意一个 Workbook 被切换到非激活状态时触发此事件	Wb：Workbook，被切换到非激活状态的工作簿
WorkbookNewChart(Wb,Ch)	当任意一个 Workbook 添加 Chart 时触发此事件	Wb：Workbook，关闭的工作簿；Ch：Object，被添加的 Chart 对象
WorkbookNewSheet(Wb,Sh)	当任意一个 Workbook 创建 Sheet 时触发此事件	Wb：Workbook，新建工作表所在的工作簿；Sh：Object，新建的工作表
WorkbookOpen(wk)	当打开一个 Workbook 时触发此事件	wk：Workbook，打开的工作簿

表 3.2 Workbook 与表格交互事件

事件函数	触发时机	参数说明
Activate()	当该 Workbook 被激活时触发此事件	无
AfterSave(Success)	当该 Workbook 被保存之后触发此事件	Success：Boolean，如果保存成功，则为 true，否则为 false
BeforeClose(Cancel)	当该 Workbook 被关闭之前触发此事件	Cancel：Object，如果设置其属性 Value 为 true，则取消此次关闭
BeforePrint(Cancel)	当该 Workbook 被打印之前触发此事件	Cancel：Object，如果设置其属性 Value 为 true，则取消此次打印
BeforeSave(SaveAsUI, Cancel)	当该 Workbook 被保存之前触发此事件	SaveASUI：Boolean，如果为 true，则表示此次保存操作将会弹出保存或“另存为”对话框；Cancel：Object，如果设置其属性 Value 为 true，则取消此次关闭
Deactivate()	当该 Workbook 被切换到非激活状态时触发此事件	无
NewChart(Ch)	当该 Workbook 添加新 Chart 时触发此事件	Ch：Object，新添加 Chart 对象
NewSheet(Sh)	当在该 Workbook 中创建 Sheet 时触发此事件	Sh：Object，新建的工作表
Open()	当该 Workbook 被打开时触发此事件	无
SheetActivate(Sh)	当激活该 Workbook 中任意一个 Sheet 时触发此事件	Sh：Object，激活的工作表
SheetBeforeDelete(Sh)	当删除该 Workbook 中任意一个 Sheet 之前触发此事件	Sh：Object，删除的工作表
SheetBeforeDoubleClick(Sh, rg, cancel)	当双击该 Workbook 中任意一个 Sheet 之前触发此事件	Sh：Object，双击的工作表对象；rg：Range 对象，双击区域所在的单元格对象；cancel：Object，如果设置其属性 Value 为 true，则取消此次双击
SheetBeforeRightClick (Sh, rg, cancel)	当右击该 Workbook 中任意一个 Sheet 之前触发此事件	Sh：Object，右击的工作表对象；rg：Range 对象，右击区域所在的单元格对象；cancel：Object，如果设置其属性 Value 为 true，则取消此次右击

续表

事件函数	触发时机	参数说明
SheetCalculate(Sh)	当在该 Workbook 中任意一个 Sheet 内进行计算时触发此事件	Sh：Object，计算的工作表对象
SheetChange(Sh，rg)	当该 Workbook 中任意一个 Sheet 内的单元格被更改时触发此事件	Sh：Object，修改的工作表对象；rg：Range 对象，修改区域所在的单元格对象
SheetDeactivate(Sh)	当该 Workbook 中任意一个 Sheet 被切换到非激活状态时触发此事件	Sh：Object，被切换到非激活状态的工作表对象
SheetFollowHyperlink(Sh，Target)	当单击该 Workbook 中任意一个 Sheet 内的超链接时触发此事件	Sh：Object，超链接所在的工作表对象；Target：Hyperlink 代表超链接的目标
SheetSelectionChange(Sh，Target)	当该 Workbook 中任意一个 Sheet 内选定区域更改时触发此事件	Sh：Object，选取改变的工作表对象；Target：Range，新选定的区域

关于表 3.1 和表 3.2 中的 Object 对象有哪些具体属性，可以在对应函数中执行以下代码获取：

```
MsgBox(typeof(wb))
var keys_ = Object.keys(wb);
MsgBox(keys_.length)
var ret_;
for(let key of keys_){
ret_ = ret_ +key+";";
}
MsgBox(ret_)
MsgBox(wb.Author)
Debug.Print("调试信息")
```

知道对象具有的属性后，可以通过属性名直接访问：

```
wb.Author,可以访问工作簿的创建者
```

这里使用这段代码查看 cancel 对象的内容，将上面的代码复制到 function Application_SheetBeforeRightClick(Sh，rg，cancel)函数中，并且将 wb 对象修改为 cancel，执行结果为：

```
MsgBox(typeof(cancel)) ==» Object
MsgBox(keys_.length)   ==» 1
MsgBox(ret_)           ==» Value
```

Value 的值为：

```
MsgBox(cancel.Value)     ==» false
```

WPS 宏编辑器提供了命令按钮、标签、文本框、复合框、列表框、复选框等控件，在使用界面的时候，可以通过拖曳的方式进行布局，每个控件都有对应的 API 对象，通过对象提供的方法可以实现控件的操作和控制。添加控件到界面后，可以修改其属性，也可以通过代码界面添加事件(同 Application 事件一样，通过选择控件选取)，这里添加一个 Label 控件，并

创建双击事件响应函数，添加代码如下：

```
function UserForm1_Label1_DblClick(cancel)
{
MsgBox("Label")
UserForm1.Label1.BackColor = 0x5B665898
//UserForm1.Label1.Move(100,100,100,100)
UserForm1.Label1.Height = 100
}
```

其中，函数体为宏编辑器自动添加，这里控件名称默认为 Label1，响应的事件为双击事件。UserForm1.Label1.BackColor＝0x5B665898 用于改变 Label1 的背景颜色，UserForm1.Label1.Height＝100 用于改变 Label1 的高度。

注意：在国产银河麒麟操作系统中表示颜色用的是 8 位十六进制数，而不是 6 位；控件名称在添加控件事件后不能改变，否则会造成改变前的事件不能响应；UserForm1 为默认的窗体名，窗体可以添加，也可以更改名称，同样，如果窗体内已经添加了内容，改变窗体名称也会导致窗体内控件不能正确执行。

控件与 API 对象对应关系如表 3.3 所示。

表 3.3 控件及其 API 对象对应关系

控件类型	API 对象	功能
窗体	UserForm	用于选中或是移动控件对象
命令按钮	CommandButton	创建一个可以让用户选择，以完成一个命令的按钮
标签	Label	用于显示用户不能编辑的文本或图像，例如，图形下的标题文本
文本框	TextEdit	用于输入或改变文本内容
复合框	ComboBox	复合框由一个文本输入控件和一个下拉菜单组成，用户可以从预先定义的列表中选择一个选项，同时也可以直接通过文本框输入
列表框	ListBox	用来显示用户可以选择的项目列表。不能一次显示全部项目时，列表可以滚动
复选框	CheckBox	可以显示出多重选择，用户可以选择一个或者多个选项，如果选中多个选项，则显示出多重选择
选项按钮	OptionButton	可以显示出多重选择，用户只能选择一个
滚动条	ScrollBar	提供在长列表工程或大量信息中快速浏览的图形工具，以比例方式指示出当前位置，或是作为一个输入设备，或速度及数量的指示器
水平布局器	Hlayout	将内部的控件按照水平方向排布，一列一个
垂直布局器	Vlayout	将内部的控件按照垂直方向排布，一行一个
水平间隔	HSpacer	填充水平方向上无用的空隙
垂直间隔	VSpacer	填充垂直方向上无用的空隙
旋转按钮	SpinButton	一种与其他控件并用微调控制的控件，也可以用来对一个范围的值或工程列表做向前或向后的遍历
切换按钮	ToggleButton	创建一个切换开关的按钮
框架	Frame	可以创建一个图形或控件的功能组。要为多个控件创建组，可先画框架，接着在框架中添加控件
多页	MultiPage	包含一个选项卡栏和一个页面区的容器类组件，通过切换选项卡来切换不同页面，从而在同一个窗口中自由切换不同的内容

WPS 宏编辑器还提供了函数处理进程的消息队列中的消息，DoEvents() 函数将暂时停止当前宏的执行，在处理完进程的消息队列中的消息后再返回继续宏的执行。

WPS 还提供了交互的接口 InputBox()函数，该函数弹出显示自定义提示信息的输入对话框，等待用户在输入框中输入文本或单击按钮，然后返回输入框的内容。

使用方法如下：

```
var ret = InputBox(prompt, title, default, xpos, ypos)
```

其中，prompt 为提示信息，string 类型；title 为输入框标题，string 类型，如果省略了 title，则标题栏中将显示应用程序名称；default 为文本框中显示的字符串表达式，string 类型，在未提供其他输入时作为默认响应，如果省略了 default，文本框将显示为空；xpos 为指定对话框的左边缘与屏幕的左边缘的水平距离，int 类型，如果省略了 xpos，对话框将水平居中；ypos 为指定对话框的上边缘与屏幕的顶部的垂直距离，int 类型，如果省略了 ypos，对话框将垂直居中。

这里结合分类统计数字化城市管理系统事部件上报数据的实例来再次熟悉 WPS 宏编辑器界面，了解界面创建和 WPS 宏程序的执行过程，程序界面和部分代码如图 3.5 所示。

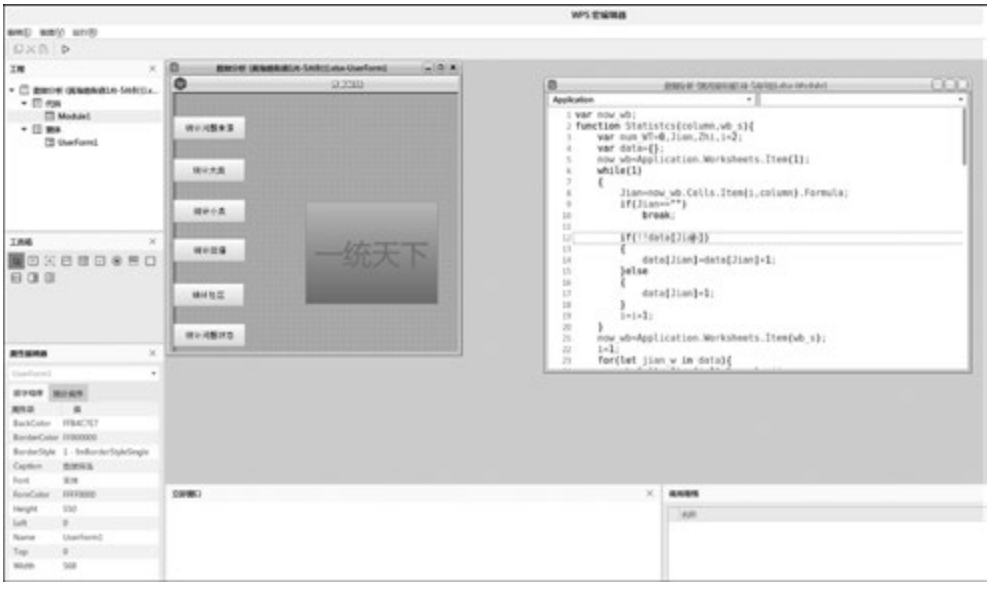


图 3.5　WPS 宏编辑器界面和代码编辑

统计单元格内容重复出现次数的实例代码如下：

```
//定义全局变量，当前操作的 sheet
var now_wb;
//统计数量的函数
function Statistcs(column,wb_s){
var num_WT = 0,Jian,Zhi,i = 2;
//定义字典数据结构，存储每一项的名称和数量
var data = {};
//打开第 1 个 sheet
now_wb = Application.Worksheets.Item(1);
while(1)
{
```

```
//读取指定列的每一行的值作为字典的键
Jian = now_wb.Cells.Item(i,column).Formula;
if(Jian == "")
break;
//判断数据结构中是不是已经存在,如果存在,对应键的值加 1,如果不存在添加键和对应值为 1
if(!!data[Jian])
{
data[Jian] = data[Jian] + 1;
}else
{
data[Jian] = 1;
}
i = i + 1;
}
//打开指定的 sheet,将统计的数据写入 sheet 中
now_wb = Application.Worksheets.Item(wb_s);
i = 1;
for(let jian_w in data){
now_wb.Cells.Item(i,1).Formula = jian_w;
now_wb.Cells.Item(i,2).Formula = data[jian_w];
i = i + 1;
}
//切换到第 1 个 sheet
now_wb = Application.Worksheets.Item(1);
}
//响应按钮:统计问题来源
function UserForm1_CommandButton1_Click()
{
//统计第 2 列数据,写入第 2 个 sheet
Statistcs(2,2);
MsgBox("统计问题来源结束");
}
//获取当前活跃的 sheet,该程序未使用
function Application_SheetActivate(Sh)
{
now_activesheet = Sh;
}
//响应按钮:统计大类
function UserForm1_CommandButton2_Click()
{
//统计第 3 列数据,写入第 3 个 sheet
Statistcs(3,3);
MsgBox("统计大类结束");
}
//响应按钮:统计小类
function UserForm1_CommandButton3_Click()
{
//统计第 4 列数据,写入第 4 个 sheet
Statistcs(4,4);
MsgBox("统计小类结束");
}
//响应按钮:统计街道
function UserForm1_CommandButton4_Click()
{
```

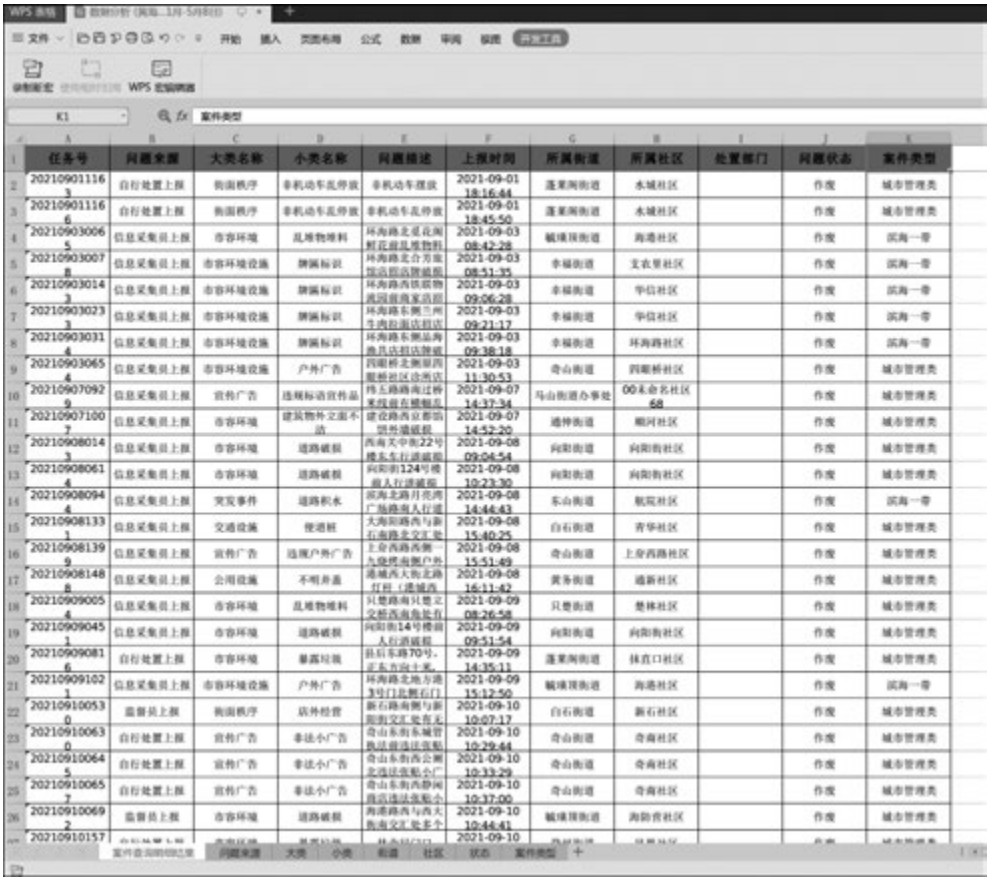
```
//统计第 7 列数据,写入第 5 个 sheet
Statistcs(7,5);
MsgBox("统计街道结束");
}
//响应按钮:统计社区
function UserForm1_CommandButton5_Click()
{
//统计第 8 列数据,写入第 6 个 sheet
Statistcs(8,6);
MsgBox("统计社区结束");
}
//响应按钮:统计问题状态
function UserForm1_CommandButton6_Click()
{
//统计第 10 列数据,写入第 7 个 sheet
Statistcs(10,7);
MsgBox("统计问题状态结束");
}
//响应按钮:统计案件类型
function UserForm1_CommandButton7_Click()
{
//统计第 11 列数据,写入第 8 个 sheet
Statistcs(11,8);
MsgBox("统计案件类型结束");
}
//响应按钮:一统天下
function UserForm1_CommandButton8_Click()
{
Statistcs(2,2);
Statistcs(3,3);
Statistcs(4,4);
Statistcs(7,5);
Statistcs(8,6);
Statistcs(10,7);
Statistcs(11,8);
MsgBox("完成统一!!!");
}
```

实例使用的表格文件如图 3.6 所示,该程序处理的是数字化城市管理平台案件导出的文件,目的是统计表格中各类别数据的数量,用于分类别统计每类问题出现的频次,便于有针对性地指导和开展工作。

执行程序后输出的数据如图 3.7 所示。

下面再通过修正信息采集员出网格时间的实例代码来了解国产办公环境下 WPS 宏通过 JavaScript 处理字符串的过程。

```
function UserForm1_CommandButton1_Click()
{
for(i = 2;i < 20001;i++)
{
var error_time = 0;
var temp_time = Cells.Item(i,8).Formula;
var temp_start_time = Cells.Item(i,6).Formula;
var temp_strs = temp_start_time.split(" ")
```



任务号	问题来源	大类名称	小类名称	问题描述	上报时间	所属街道	所属社区	处置部门	问题状态	案件类型
202109011163	自行处置上报	街面秩序	非机动车乱停放	非机动车乱停放	2021-09-01 18:16:44	蓬莱阁街道	水城社区		作废	城市管理类
202109011166	自行处置上报	街面秩序	非机动车乱停放	非机动车乱停放	2021-09-01 18:45:50	蓬莱阁街道	水城社区		作废	城市管理类
202109030065	信息采集员上报	市容环境	乱堆物堆料	环海路北延花海鲜花前乱堆物料	2021-09-03 08:42:28	毓璜顶街道	海港社区		作废	滨海一带
202109030078	信息采集员上报	市容环境设施	牌匾标识	环海路北合万海馆店前店牌破损	2021-09-03 08:51:35	幸福街道	文农里社区		作废	滨海一带
202109030143	信息采集员上报	市容环境设施	牌匾标识	环海路西铁观物流园前商家店招	2021-09-03 09:06:28	幸福街道	华信社区		作废	滨海一带
202109030233	信息采集员上报	市容环境设施	牌匾标识	环海路东侧兰州牛肉拉面店招店	2021-09-03 09:21:17	幸福街道	华信社区		作废	滨海一带
202109030314	信息采集员上报	市容环境设施	牌匾标识	环海路东侧品海鱼具店招店牌破	2021-09-03 09:38:18	幸福街道	环海路社区		作废	滨海一带
202109030654	信息采集员上报	市容环境设施	户外广告	四期桥北侧原四期桥社区诊所店	2021-09-03 11:30:53	奇山街道	四期桥社区		作废	城市管理类
202109070920	信息采集员上报	宣传广告	违规标语宣传品	纬上路路南过桥米线前有横幅乱	2021-09-07 14:37:34	马山街道办事处	00未命名社区68		作废	城市管理类
202109071007	信息采集员上报	市容环境	建筑物外立面不洁	建设路西京都街饰外墙破损	2021-09-07 14:52:20	通神街道	顺河社区		作废	城市管理类
202109080143	信息采集员上报	市容环境	道路破损	西海文华东22号楼东车行道破损	2021-09-08 09:04:54	向阳街道	向阳街社区		作废	城市管理类
202109080614	信息采集员上报	市容环境	道路破损	向阳街124号楼前人行道破损	2021-09-08 10:23:30	向阳街道	向阳街社区		作废	城市管理类
202109080944	信息采集员上报	突发事件	道路积水	振海北路月亮湾广场路南人行道	2021-09-08 14:44:43	东山街道	航院社区		作废	滨海一带
202109081331	信息采集员上报	交通设施	便道桩	大海阳路西与新石南路北交汇处	2021-09-08 15:40:25	白石街道	青华社区		作废	城市管理类
202109081399	信息采集员上报	宣传广告	违规户外广告	上京西路西侧一九依纬商贸户外	2021-09-08 15:51:49	奇山街道	上京西路社区		作废	城市管理类
202109081488	信息采集员上报	公用设施	不明井盖	通城西大街北路灯杆（通城西	2021-09-08 16:11:42	黄务街道	通新社区		作废	城市管理类
202109090054	信息采集员上报	市容环境	乱堆物堆料	只楚路南只楚立交桥西南角处有	2021-09-09 08:26:58	只楚街道	楚林社区		作废	城市管理类
202109090451	信息采集员上报	市容环境	道路破损	向阳街14号楼前人行道破损	2021-09-09 09:51:54	向阳街道	向阳街社区		作废	城市管理类
202109090816	自行处置上报	市容环境	暴露垃圾	县后东路70号，正东方向土堆	2021-09-09 14:35:11	蓬莱阁街道	抹直口社区		作废	城市管理类
202109091021	信息采集员上报	市容环境设施	户外广告	环海路北地东港3号门北侧石门	2021-09-09 15:12:50	毓璜顶街道	海港社区		作废	滨海一带
202109100530	监督员上报	街面秩序	店外经营	新石路南侧与新阳街交汇处东北	2021-09-10 10:07:17	白石街道	新石社区		作废	城市管理类
202109100630	自行处置上报	宣传广告	非法小广告	奇山东街东城管执法前违法张贴	2021-09-10 10:29:44	奇山街道	奇商社区		作废	城市管理类
202109100645	自行处置上报	宣传广告	非法小广告	奇山东街西公厕北违法张贴小广	2021-09-10 10:33:29	奇山街道	奇商社区		作废	城市管理类
202109100657	自行处置上报	宣传广告	非法小广告	奇山东街西静闻商店违法张贴小	2021-09-10 10:37:00	奇山街道	奇商社区		作废	城市管理类
202109100692	监督员上报	市容环境	道路破损	海港路西与西大街南交汇处多个	2021-09-10 10:44:41	毓璜顶街道	海防营社区		作废	城市管理类
202109101570					2021-09-10					

图 3.6　实例使用的表格文件数据格式

```
var str_start = temp_strs[1].split(":")
var temp_end_time = Cells.Item(i,7).Formula;
var temp_stre = temp_end_time.split(" ")
var str_end = temp_stre[1].split(":")
if(str_start[0]>="14")
{
error_time = temp_time;
}
else if(str_end[0].concat(str_end[1])<="1130")
{
error_time = temp_time;
}
else if(str_start[0].concat(str_start[1])<"1130" && str_end[0].concat(str_end[1])<="1400"
&& str_end[0].concat(str_end[1])>"1130")
{
//MsgBox((Number(str_end[0]) - 11) * 60 - 30 + Number(str_end[1]));
error_time = temp_time - ((Number(str_end[0]) - 11) * 60 - 30 + Number(str_end[1]));
}
else if(str_start[0].concat(str_start[1])>="1130" && str_end[0].concat(str_end[1])<=
"1400")
{
error_time = 0;
}
```

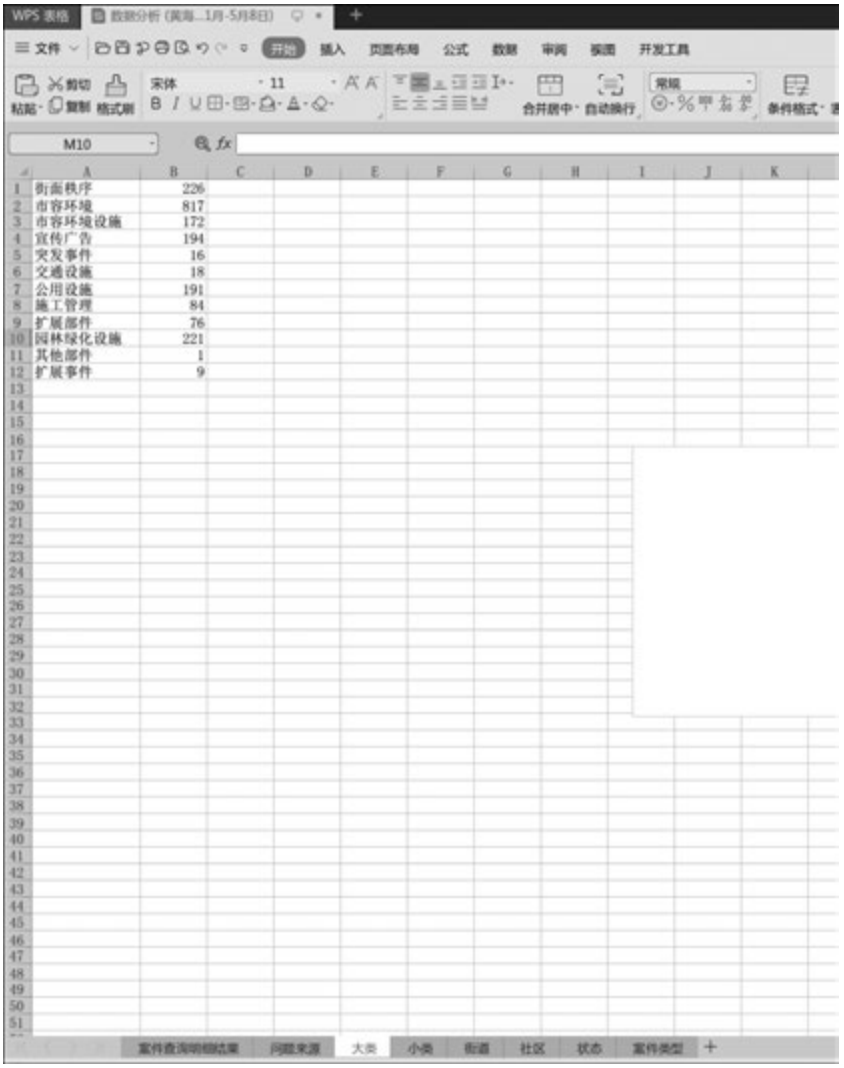


图 3.7　宏执行后统计的数据

```
else if(str_start[0].concat(str_start[1])<="1130" && str_end[0].concat(str_end[1])>="1400")
{
error_time=temp_time-150;
}
else
{
error_time=temp_time-((14-Number(str_start[0]))*60-Number(str_start[1]));
}
Cells.Item(i,9).Formula=error_time;
}
}
```

该实例使用的表格格式如图 3.8 所示，该宏代码处理的是数字化城市管理平台信息监督员管理中出网格情况的统计数据，主要修正最后一列的时间，根据开始时间和结束时间，剔除信息采集员的非工作时间。

图 3.8 程序使用的表格格式

3.2.3 Python 操作 Excel

Python 可以操作 Excel 的库比较多，但是不同的库支持的环境和可实现的操作不同，这里汇总了各个常用 Python 库的情况，如表 3.4 所示。

表 3.4 Python 常用来操作 Excel 的库汇总

包 名	支持的文件格式		支持的操作			不需要安装 Excel
	.xls	.xlsx	读取	写入	修改	
xlrd	√	×	√	×	×	√
xlwt	√	×	×	√	√	√
openpyxl	×	√	√	√	√	√
xlsxwriter	×	√	×	√	×	√
pylightxl	×	√	√	√	√	√
xlwings	√	√	√	√	√	×
pandas	√	√	√	√	×	√

新版的 xlrd 已放弃了对.xlsx 文件的支持，如果希望读取.xlsx 文件有两种方法：第一种是降低 xlrd 版本，选择 1.2.0；第二种是将.xlsx 格式另存为.xls 格式。

在银河麒麟 V10 中，Python 2.7 可以使用 Pandas，Python 3.5 不可以使用，如果想在 Python 3 中使用 pandas，需要升级 Python，本书开发环境使用的是 Python 3.10，可以使用 pandas。

在操作 Excel 之前，首先来看 Excel 的结构，关于 Excel 结构参见 3.2.1 节 WPS 表格介绍。

下面从以下几个方面介绍 Python 操作 Excel。

1. 使用 xlrd 读取 Excel 数据

xlrd 不支持读取 Excel 表中的嵌入对象，包括图、表、宏等，也不支持读取单元格的颜色。xlrd 库可以使用的表格数据，实际上就是读取 Excel 单元格值和属性，本节使用的 Excel 文件内容如图 3.9 所示。

先看一个官方实例：

```
import xlrd as rd
```

	A	B	C	D	E	F	G
1	责任网格名称	7月案件数	面积	属性	事部件发生率	均值（率）	均值（数）
2	6	192	1.92	k-yellow	100	105.12806	150.525
3	7	193	2.92	k-yellow	101	106.12806	151.525
4	http://baidu.com	194	3.92	http://qq.com	qq.com	12	d
5							
6							
7							
8							
9							
10							

图 3.9　使用的 Excel 文件内容

```
book = rd.open_workbook("0.xls")
print("The number of worksheets is {0}".format(book.nsheets))
print("Worksheet name(s): {0}".format(book.sheet_names()))
sh = book.sheet_by_index(0)
print("{0} {1} {2}".format(sh.name, sh.nrows, sh.ncols))
print("Cell D1 is {0}".format(sh.cell_value(rowx=0, colx=3)))
for rx in range(sh.nrows):
print(sh.row(rx))
```

说明：

import xlrd as rd——将 xlrd 包导入，用来操作 Excel。

book=rd.open_workbook("0.xls")。打开同目录下的 0.xls 文件，返回一个 Book 类(包含表格内容)的实例，如果文件打开错误，返回 None；open_workbook()函数的参数及含义如表 3.5 所示。

表 3.5　open_workbook()函数的参数

参　数　名	默　认　值	参 数 含 义
filename	None	要打开表格的路径
logfile	<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>	用来写入消息和诊断信息的文件句柄
verbosity	0	增加写入日志文件的跟踪材料的数量
use_mmap	True	是否使用 mmap 模块是启发式地确定的。使用此参数可覆盖结果。当前的启发式：如果存在，则使用 mmap
file_contents	None	字符串或 mmap.mmap 对象或其他行为相似的对象。如果提供了 file_contents，则不会使用 filename，除非(可能)在消息中使用
encoding_override	None	用于克服旧版本文件中缺少或不正确的代码页信息
formatting_info	False	默认值为 False，这样可以节省内存。在这种情况下，“空白”单元格是指那些有自己的格式信息但没有数据的单元格，通过忽略文件的 Blank 和 MulBlank 记录，这些单元格将被视为空。这会剪切空单元格或空白单元格行的底部或右侧边距。只有 cell_value()和 cell_type()可用。当为 True 时，将从电子表格文件中读取格式信息。这提供了所有单元格，包括空单元格和空白单元格。每个单元格都有格式信息。请注意，当与.xlsx 文件一起使用时，将引发 NotImplementedError

续表

参 数 名	默 认 值	参数含义
on_demand	False	控制表格加载时机，默认加载全局资源和所有表格
ragged_rows	False	默认值为 False，意味着所有行都用空单元格填充，以便所有行的大小与 ncols 中的大小相同。True 表示行的末尾没有空单元格。如果行的大小变化很大，则可以节省大量内存
ignore_workbook_corruption	False	此选项允许读取损坏的工作簿。如果为 False，则可能会出现 CompDocError（工作簿损坏）问题。当为 True 时，该异常将被忽略

open_workbook 函数返回的结果是 Book 类的对象，它包含整个电子表格或者工作簿的内容，其主要函数及含义如表 3.6 所示。

表 3.6 Book 类的几个常用函数

函 数 名	含 义	本机实验结果
datemode	文件最后一次保存生效的时间系统，0：1900 系统（Excel for Windows 默认设置）。1：1904 系统（默认为 Excel for Macintosh）	0
biff_version	用于创建文件的 BIFF（二进制交换文件格式）的版本。最新版本是 8.0（此处表示为 80），由 Excel 97 引入。该模块最早支持的版本：2.0（表示为 20）	80
codepage	一个整数，表示用于此文件中字符串的字符集。对于 BIFF 8 及更高版本，这将是 1200，意思是 Unicode；更确切地说，是 UTF_ 16_LE	1200
countries	一个元组，包含以下各项的电话国家/地区代码：[0]：创建文件时的用户界面设置。[1]：区域设置	(86, 86)
user_name	（如果有的话）被记录为保存文件的最后一个用户的名称	user
sheets()	返回表格中所有 Sheet 的列表。将加载所有尚未加载的图纸	[< xlrd. sheet. Sheet object at 0x769578b1f0 >, < xlrd. sheet. Sheet object at 0x769578b2e0 >, < xlrd. sheet. Sheet object at 0x769578b310 >]
sheet_by_index (sheetx)	sheetx 为工作表索引，返回一个对应的 Sheet	< xlrd. sheet. Sheet object at 0x79d295b1f0 >
sheet_by_name (sheet_name)	参数区分字母大小写，是 Sheet 名字，返回 Sheet	< xlrd. sheet. Sheet object at 0x7927303220 >
sheet_names()	返回电子表格文件内的所有 Sheet 名称	['Sheet1', 'Sheet2', 'Sheet3']
nsheets	返回电子表格中 Sheet 数	3

sh＝book. sheet_by_index(0)——该函数是按照序列返回 Sheet 对象，这里参数 0 表示返回电子表格中第一个 Sheet 的实例，该实例属于 sheet 类，包含一个工作表的数据，其主要函数及含义如表 3.7 所示。

表 3.7 sheet 类的几个主要函数

函 数 名	含 义	本机实验结果
col(colx)	返回给定列中单元格对象的序列，当 colx=1 时，表示第二列	[text:'7 月案件数',number:192.0]
vert_split_pos	返回左窗格的列数，如果冻结首列，结果为 1，如果没有冻结首列，结果为 0	0
horz_split_pos	返回上窗格的行数，如果冻结首行，结果为 1，没有冻结首行，结果为 0	0
book	返回对该 Sheet 所在电子表格的引用	<xlrd.book.Book object at 0x73d23563b0>
name	返回 Sheet 的名称	Sheet1
nrows	返回行数	2
ncols	返回列数	7
hyperlink_list	返回 Sheet 中超链接的列表，list	[<xlrd.sheet.Hyperlink object at 0x7b331076a0>, <xlrd.sheet.Hyperlink object at 0x7b331075b0>]
hyperlink_map	返回带定位的超链接映射，dict	{(3,0):<xlrd.sheet.Hyperlink object at 0x7b331076a0>, (3,3):<xlrd.sheet.Hyperlink object at 0x7b331075b0>}
cell(rowx, colx)	给定行和列中的单元格对象	text:'责任网格名称'
cell_value(rowx, colx)	给定行和列中的单元格的值	责任网格名称
cell_type(rowx, colx)	给定行和列中单元格的类型 0：empty string；1：a Unicode string；2：float；3：float；4：int，1 为 True，0 为 False	1
row_len(rowx)	返回给定行中的有效单元格数。用于 open_workbook(ragged_rows=True)，它可能会生成少于 ncols 单元格的行	7
row(rowx)	返回指定行单元格对象的列表	[text:'责任网格名称', text:'7 月案件数', text:'面积', text:'属性', text:'事件发生率', text:'均值(率)', text:'均值(数)']
row_values(rowx, start_colx=0, end_colx=3)	返回指定行的值切片，list	['责任网格名称', '7 月案件数', '面积']
col_values(colx, start_rowx=0, end_rowx=2)	返回指定列的值切片，list	['责任网格名称', 6.0]
row_slice(rowx, start_colx=0, end_colx=3)	返回指定行的单元格切片，list	[text:'责任网格名称', text:'7 月案件数', text:'面积']
col_slice(0, start_rowx=0, end_rowx=2)	返回指定列的单元格切片，list	[text:'责任网格名称', number:6.0]

2. 读取 Excel 内的图片

本节使用的 Excel 文件格式如图 3.10 所示。

xlrd 并不能直接读取图片，需要结合 xml 来解析 Excel 文件，前面已经对 xlrd 读取 Excel 数据进行了详细介绍，下面主要介绍 xml 解析 Excel 文件的过程。

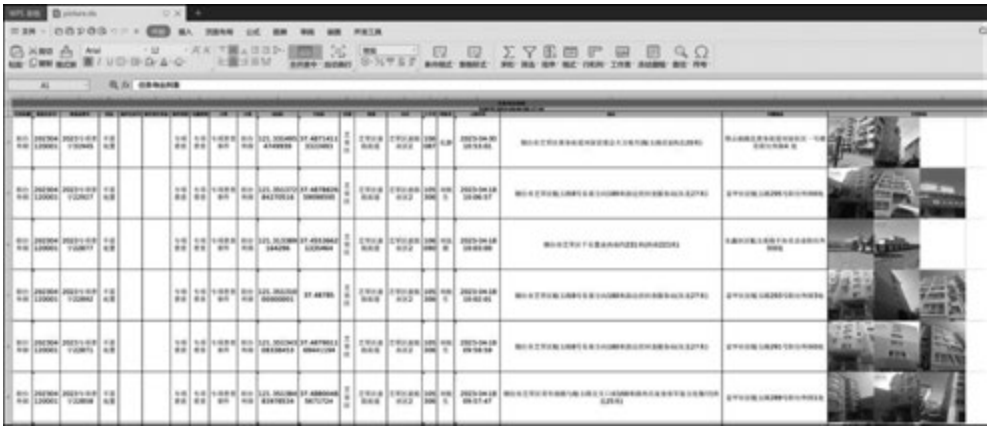


图 3.10 使用的 Excel 原始数据

实例代码如下：

```
import xlrd as rd
from xml.dom import minidom
import shutil
import os
import zipfile
#获得当前执行的 Python 文件所在目录
path_parent = os.getcwd()
#将含有图片的目标 Excel 复制到 tmp 目录,并重命名为 ZIP 格式
shutil.copy(path_parent + '/picture.xlsx', path_parent + '/tmp/tmp.xlsx')
os.rename(path_parent + '/tmp/tmp.xlsx', path_parent + '/tmp/tmp.zip')
#准备解压缩的文件
zip_file = zipfile.ZipFile(path_parent + '/tmp/tmp.zip')
# 判断解压缩的目标路径是否已经存在,如果存在,将其全部删除
if os.path.exists(path_parent + '/tmp/extra_tmp'):
    shutil.rmtree(path_parent + '/tmp/extra_tmp')
#解压缩
zip_file.extractall(path_parent + '/tmp/extra_tmp/')
#判断图片存储的最终目录是否存在,如果存在,将其删除,主要实现清空的功能
if os.path.exists("/home/user/PycharmProjects/pythonProject2/test_excel/done"): shutil.
rmtree("/home/user/PycharmProjects/pythonProject2/test_excel/done")
# 加载 Excel 文件,用于读取问题描述单元格
my_book = rd.open_workbook('picture.xlsx')
my_sheet = my_book.sheet_by_index(0)
'''
解析 Excel 文件,用问题描述对图片进行命名
'''
#将 xml 结构解析为 dom 树结构
DOMTree = minidom.parse("/home/user/PycharmProjects/pythonProject2/test_excel/tmp/extra_
tmp/xl/drawings/drawing1.xml")
#提取文档唯一的根元素
myDOM = DOMTree.documentElement
#搜索全部具有特定元素类型名称的后继元素
xdr_twoCellAnchors = myDOM.getElementsByTagName("xdr:twoCellAnchor")
i = 0
#创建图片存储的最终生成目录
os.mkdir("/home/user/PycharmProjects/pythonProject2/test_excel/done")
```

```
for xdr_twoCellAnchor in xdr_twoCellAnchors:
    xdr_froms = xdr_twoCellAnchor.getElementsByTagName("xdr:from")
    xdr_row = xdr_froms[0].getElementsByTagName("xdr:row")
    row_num = xdr_row[0].childNodes[0].nodeValue                    #获得第几行
    print(my_sheet.cell_value(int(row_num), 19))
    xdr_pics = xdr_twoCellAnchor.getElementsByTagName("xdr:pic")
    xdr_blipFill = xdr_pics[0].getElementsByTagName("xdr:blipFill")
    pic_str = xdr_blipFill[0].childNodes[0].getAttribute("r:embed")   #获得图片编号
    pic_name_source = pic_str.replace(pic_str[:3], "image") + ".jpeg"
    pic_name_dis = my_sheet.cell_value(int(row_num), 14) + "_" + my_sheet.cell_value(
int(row_num), 2) + "_" + str(
        int(row_num) + 1) + "_" + pic_str.replace(pic_str[:3], "p") + "_" + my_sheet.cell_value(
int(row_num), 13) + "_" + \
                  my_sheet.cell_value(int(row_num), 19) + ".jpeg"
shutil.copyfile("/home/user/PycharmProjects/pythonProject2/test_excel/tmp/extra_tmp/xl/
media/" + pic_name_source, "/home/user/PycharmProjects/pythonProject2/test_excel/done/" +
pic_name_dis)
    print(pic_name_dis)
    i = i + 1
print(i)
```

首先导入需要的模块,其中,shutil 和 os 可以操作目录和文件,这里用来实现文件夹的创建、清空和文件的复制等操作;zipfile 模块用来实现压缩文件的解压。

.xlsx 文件实际上是基于 ZIP 压缩格式的文件,它包含了许多.xml 文件和相关资源,这些文件共同构成了一个 Excel 工作簿。可以通过将.xlsx 文件的扩展名改为.zip,然后使用解压工具来解压,进而查看其中的内容。

解压后通常包含的文件夹和文件如图 3.11 所示。

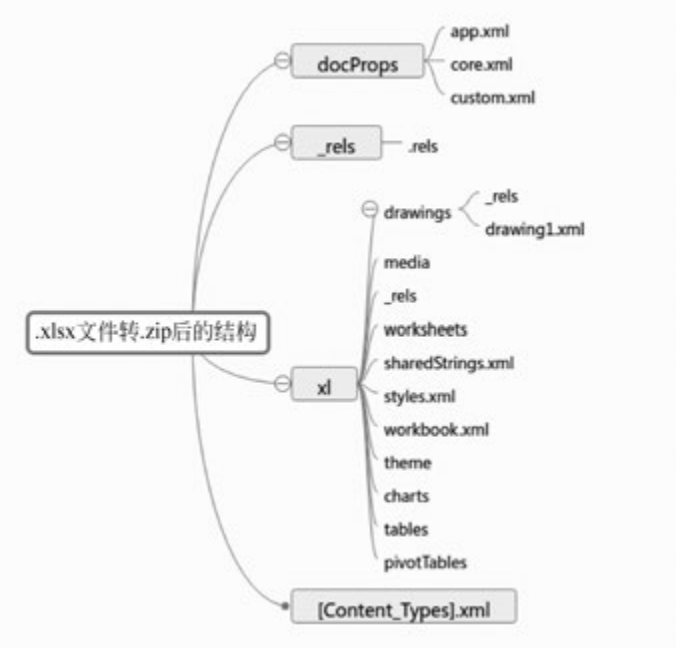


图 3.11　.xlsx 文件解压缩后的结构及说明

各个文件夹及文件的说明如下：

- docProps 文件夹通常包含 app.xml、core.xml 和 custom.xml，其中，app.xml 包含应用程序特定的文档属性，如创建应用程序的名称、文档的创建日期等；core.xml 包含核心的文档属性，如文档标题、作者、创建日期等，custom.xml 包含自定义文档属性。
- _rels 文件夹通常包含.rels 文件，它定义了文件之间的关系，比如工作簿文件和其他文件的关系。
- xl 文件夹是.xlsx 文件的核心部分，通常包含以下子文件夹和文件，其中，_rels 文件夹定义了工作簿文件与其他文件的关系，如与工作表文件的关系；worksheets 文件夹中每个 XML 文件对应一个工作表，包含工作表的数据和格式信息；styles.xml 包含工作簿中所有样式定义，如字体、颜色、边框等；workbook.xml 定义了工作簿的结构，如工作表的顺序、隐藏的工作表、工作簿的视图设置等；sharedStrings.xml 包含工作簿中所有唯一字符串的列表，这些字符串在工作表中以索引引用；theme 文件夹包含工作簿的主题颜色和效果；charts 文件夹包含图表的数据和格式信息；drawings 文件夹包含绘图对象（如形状、图片等）的数据和格式信息，该文件夹中的 drawing1.xml 文件包含了图片和.xlsx 文件单元格之间的对应关系；tables 文件夹包含表格的数据和格式信息；pivotTables 文件夹包含数据透视表的数据和格式信息；media 文件夹包含工作簿中使用的所有图片、图标等资源文件。对 Excel 图片的提取，实际上就是通过解析 drawing1.xml 来找到 media 文件夹中的对应图片，将其复制出来并进行重新命名。
- [Content_Types].xml 文件定义了文件夹中各种文件的 MIME 类型和扩展名，告诉解压工具和应用程序如何处理这些文件。

3. 写入 Excel

将数据写入 Excel，可以使用 xlwt 库来完成。xlwt 库不但可以写入数据，还可以写入格式信息。因为写入的是.xls 文件，所以其写入数据量有上限，最多 65 536 行。

1）插入数据

```
import xlwt as wt
myBook = wt.Workbook()
mysheet = myBook.add_sheet("test", cell_overwrite_ok = False)
mysheet.write(0, 0, "内容")
myBook.save("test_excel_write.xls")
```

使用 Workbook 类的初始化函数创建一个 Excel 文件，可返回 Workbook 类的一个实例。具体来说，可使用 Workbook 类的 add_sheet()函数，创建一个 sheet，返回 Worksheet 类的一个实例，这样就可以通过 Worksheet 的 write()函数写入数据，最后使用 Workbook 类的 save()函数保存 Excel 文件。

2）设置样式

```
import xlwt as wt
from datetime import datetime
myBook = wt.Workbook()
mysheet = myBook.add_sheet("test", cell_overwrite_ok = False)
# 初始化样式
```

```
style1 = wt.XFStyle()
# 为样式创建字体
font = wt.Font()
font.charset = 0x86
font.name = 'Times New Roman'          #字体
font.bold = True                       #加粗
font.underline = True                  #下画线
font.italic = True                     #斜体
#设置对齐方式
alignm = wt.Alignment()
alignm.horz = wt.Alignment.HORZ_CENTER
alignm.vert = wt.Alignment.VERT_CENTER
# 设置样式
style1.font = font
style1.alignment = alignm
# 使用样式
mysheet.write(0, 0, "内容:字体加粗、加下画线、斜体", style1)
# 边框部分
borders = wt.Borders()
# 设置线型
borders.left = wt.Borders.DASHED
borders.right = wt.Borders.DASHED
borders.top = wt.Borders.DASHED
borders.bottom = wt.Borders.DASHED
# 设置颜色
borders.left_colour = 0x22
borders.right_colour = 0x44
borders.top_colour = 0x66
borders.bottom_colour = 0x88
style2 = wt.XFStyle()
style2.borders = borders
# 使用样式
mysheet.write(2, 1, "内容:设置边框", style2)
myBook.save("test_excel_write.xls")
```

xlwt 定义的几种样式对应关系如表 3.8 所示。

表 3.8　xlwt 几种样式的对应关系

类　型	标　识	值
部分字符集	ansi_latin	0x00
	sys_default	0x01
	symbol	0x02
	apple_roman	0x4d
	ansi_chinese_gbk	0x86
	ansi_chinese_big5	0x88
	no_line	0x00
部分线型	thin	0x01
	medium	0x02
	dashed	0x03
	dotted	0x04
	thick	0x05
	double	0x06
	aqua	0x31

续表

类　型	标　识	值
部分颜色	black	0x08
	blue	0x0C
	blue_gray	0x36
	bright_green	0x0B
	dark_blue	0x12
	dark_green	0x3A
	dark_red	0x10
	dark_yellow	0x13
	pink	0x0E
	red	0x0A
	white	0x09
	yellow	0x0D

程序运行后，结果如图 3.12 所示。

图 3.12　设置格式的数据写入

3）插入图片

xlwt 只能插入.bmp 格式的图片，如果需要插入其他格式的图片，需要使用图形处理库进行转换或者使用其他的 excel 操作库，比如 openpyxl、xlwings、xlsxwriter，当然也可以将 jpg 格式的图片通过其他软件或在线转换网站转换为 bmp 格式后再插入。

实例：

```
import xlwt as wt
import cv2 as cv
myBook = wt.Workbook()
mysheet = myBook.add_sheet("test", cell_overwrite_ok = False)
img = cv.imread('title.png')
cv.imwrite("image.bmp", img)
mysheet.insert_bitmap("image.bmp", 2, 5, 2, 2, 0.5, 0.5)
myBook.save("test_excel_write.xls")
```

其中，import cv2 as cv 用于导入 OpenCV 库，较新版本的 OpenCV 库在麒麟系统 V10 版本中的 Python 2 和 Python 3.5 均安装不成功，升级为 Python 3.10 后可以安装 OpenCV 4.8。

```
img = cv.imread('title.png')
cv.imwrite("image.bmp", img)
```

上面两行代码实现了图片格式的转换，将 png 格式的图片转换为 bmp 格式，为后面的

图片插入做好准备。

mysheet.insert_bitmap("image.bmp", 2, 5, 10, 10, 0.5, 0.5)使用的是 Worksheet 中的 insert_bitmap()函数,该函数的说明如下:

```
insert_bitmap(filename, row, col, x = 0, y = 0, scale_x = 1, scale_y = 1)
```

filename 表示要插入的图像地址,row 表示行,col 表示列; x 和 y 表示相对原来位置向下向右偏移的距离,如果不设置,默认为图片左上角对齐 row、col 表示单元格的左上角,scale_x、scale_y 表示相对原图宽高的比例,图片可放大缩小。程序运行后结果如图 3.13 所示。

图 3.13 插入.bmp 格式的图片

4. 修改 Excel

对于 Excel 的修改,有以下几种方法可以实现。

(1) 使用 xlrd 和 xlutils 配合,通过将原来 Excel 文件复制到内存进行修改后再另存的方式实现。

代码如下:

```
import xlrd
from xlutils.copy import copy
oldwork = xlrd.open_workbook("0.xls")
#根据索引值选取 sheet,用于获取原来 sheet 的基本信息
oldsheet = oldwork.sheet_by_index(0)
rowNumbers = oldsheet.nrows
colNumbers = oldsheet.ncols
#复制 workbook
new_book = copy(oldwork)
#通过索引获得 sheet
new_table = new_book.get_sheet(0)
for i in range(rowNumbers):
    new_table.write(i, 0, "test")
new_book.save("00.xls")
```

在修改时一定要先得到原来表格的基本信息,用来控制整个修改过程,程序运行前后对比如图 3.14 和图 3.15 所示。

(2) 使用 xlrd 和 xlwt 配合实现,通过 xlrd 读取,然后通过 xlwt 对新创建的 Excel 文件进行修改后再写入。

图 3.14 使用 xlrd 修改 Excel 文件前

图 3.15 使用 xlrd 修改 Excel 文件后

代码如下：

```
import xlrd
import xlwt
# 打开已有的 Excel 文件
oldworkbook = xlrd.open_workbook('0.xls')
# 获取第一个工作表
oldsheet = oldworkbook.sheet_by_index(0)
# 获取工作表的行数和列数
rowNumbers = oldsheet.nrows
colNumbers = oldsheet.ncols
# 创建一个新的工作簿
new_workbook = xlwt.Workbook()
# 添加一个工作表
new_worksheet = new_workbook.add_sheet('Sheet1')
# 复制已有的数据
for r in range(rowNumbers):
    for c in range(colNumbers):
        # 获取单元格的值
        cell_value = oldsheet.cell_value(r, c)
        # 将单元格的值写入新的工作表
        new_worksheet.write(r, c, cell_value)
```

```
# 写入新的数据
new_worksheet.write(rowNumbers, 0, '第一列追加数据')
new_worksheet.write(rowNumbers, 1, '第二列追加数据')
# 保存新的工作簿
new_workbook.save('000.xls')
```

通过 xlrd 和 xlwt 的配合，不仅可以修改数据，还可以实现样式的修改，具体修改方法参考第 2 点的内容。程序运行后，Excel 文件修改前后的对比如图 3.16 和图 3.17 所示。

	A	B	C	D	E	F	G
1	责任网格名称	7月案件数	面积	属性	事部件发生率	均值（率）	均值（数）
2	6	192	1.92	k-yellow	100	105.12806	150.525
3	7	193	2.92	k-yellow	101	106.12806	151.525
4	http://baidu.com	194	3.92	http://qq.com	qq.com	12	d

图 3.16　使用 xlwt 修改 Excel 文件前

	A	B	C	D	E	F	G
1	责任网格名称	7月案件数	面积	属性	事部件发生率	均值（率）	均值（数）
2	6	192	1.92	k-yellow	100	105.1281	150.525
3	7	193	2.92	k-yellow	101	106.1281	151.525
4	http://baidu.com	194	3.92	http://qq.cc	qq.com	12	d
5	第一列追加数据	第二列追加数据					

图 3.17　使用 xlwt 修改 Excel 文件后

视频讲解

3.3　WPS 文字及其自动化处理

3.3.1　WPS 文字介绍

WPS 文字是 WPS Office 套件中的一个重要组成部分，主要用于处理文字文档。以下是 WPS 文字的一些主要特点和功能：

（1）多功能集成。WPS 文字提供了丰富的文字编辑功能，包括但不限于字体设置、段落排版、样式应用等，可满足日常办公和学术写作的需求。

(2) 兼容性强。它支持多种文件格式的导入和导出,例如,.doc、.docx、.pdf 等,确保了与其他流行文字处理软件的良好兼容性。

(3) 页面设置灵活。用户可以根据需求调整页面尺寸、边距、页眉页脚等,以及使用分节符进行不同部分的页面设置,使文档更加专业和符合特定要求。

(4) 表格和图形处理。除了文字处理,WPS 文字还支持创建和编辑表格,插入图形和图片,使得文档内容更加丰富和直观。

(5) 跨平台支持。WPS Office 覆盖 Windows、Linux、Android、iOS 等多个平台,支持桌面和移动办公,方便用户在不同设备上进行文档处理。

综上所述,WPS 文字是一个功能强大、操作简便、兼容性好的文字处理工具,无论是学生、教师还是企业员工,都可以通过它高效地完成各种文档处理任务。

3.3.2 WPS 文字结构

WPS 文字结构是指使用 WPS Office 软件中的文字处理功能来创建和编辑文档的结构。WPS Office 是一款功能强大的办公软件,其中的文字处理功能可以帮助用户轻松地创建和编辑各种类型的文档,如报告、论文、简历等。

WPS 文字结构主要包括以下几个方面:

- 页面布局。WPS 文字处理功能允许用户自定义页面的大小、边距、方向等,以满足不同文档的需求。
- 字体设置。用户可以为文档中的文字选择不同的字体、字号、颜色等,以增强文档的可读性和美观性。
- 段落格式。WPS 文字处理功能提供了多种段落格式设置,如对齐方式、行距、缩进等,以帮助用户创建结构清晰、层次分明的文档。
- 标题和目录。用户可以为文档添加多级标题,并自动生成目录,方便用户快速了解文档的主要内容。
- 列表和编号。WPS 文字处理功能支持项目符号列表和编号列表,帮助用户组织和呈现信息。
- 表格和图表。用户可以在文档中插入表格和图表,以便更直观地展示数据和信息。
- 图片和形状。WPS 文字处理功能支持插入图片和形状,丰富文档的内容和形式。
- 页眉和页脚。用户可以为文档添加页眉和页脚,包含文档标题、页码等信息,方便读者查阅。
- 超链接和书签。WPS 文字处理功能支持插入超链接和书签,帮助用户快速跳转到相关部分。

通过以上这些功能,用户可以轻松地创建出结构清晰、内容丰富的文档。同时,WPS Office 还提供了丰富的模板资源,可帮助用户快速创建各种类型的文档。

3.3.3 使用 python-docx 操作 Word

Python-docx 是一个用于创建和更新 Word (.docx)文件的 Python 库。它提供了一系列功能,使得用户能够通过 Python 代码以访问对象的方式来操作 Word 文件,其主要特点和功能如下:

- 创建新文档——可以使用 Python-docx 创建新的 Word 文件，并在其中添加文本、表格、图片等元素。
- 编辑现有文档——Python-docx 允许用户打开现有的 Word 文件，并进行修改，如添加或删除文本、更改字体样式等。
- 段落操作——Python-docx 中的段落(paragraph)是文档的基本组成单位。用户可以添加新的段落或者对现有段落进行编辑。
- 表格处理——可以生成表格，并对表格中的数据进行操作，如插入行、列或单元格，以及编辑单元格内容。
- 图片和图表——可以在文档中插入图片和图表，并对它们进行必要的格式化。
- 样式应用——可以对文档中的文本应用不同的样式，包括字体、大小、颜色、对齐方式等。
- 目录和页眉页脚——可以添加目录、页眉和页脚，以及执行其他高级文档格式化操作。
- 兼容性——Python-docx 专注于处理.docx 格式的文件，这是微软 Office 2007 及以后版本使用的默认 Word 文档格式。
- 与其他库结合——Python-docx 可以与其他 Python 库(如 Pandas)结合使用，方便地在 Word 文档中插入 Excel 表格，节省大量手动操作的时间。

Python-docx 是一个功能强大的库，它使得 Python 开发者能够以编程的方式高效地处理 Word 文档。无论是自动化报告生成、批量文档编辑，还是其他需要与 Word 交互的场景，Python-docx 都是一个值得考虑的工具。

在编写本书时，基于国产银河麒麟 V10 系统，安装的 Python-docx 库版本是 0.8.11，本书所有代码均使用 0.8.11 版本开发。从 2023 年 5 月至 2023 年 11 月，Python-docx 共经历了 1.0.0、1.0.1 和 1.1.0 版本，下面列出官方发布的 Python-docx 库更新内容。

1.0.0 版本：

不再支持 Python 2，支持 Python 3.7 以上版本

修复＃85：Paragraph.text 包含超链接文本

添加 Hyperlink.address 属性

添加 Hyperlink.contains_page_break 属性

添加 Hyperlink.runs 属性

添加 Hyperlink.text 属性

添加 Paragraph.contains_page_break 属性

添加 Paragraph.hyperlinks 属性

添加 Paragraph.iter_inner_content()函数

添加 Paragraph.rendered_page_breaks 属性

添加 RenderedPageBreak.following_paragraph_fragment 属性

添加 RenderedPageBreak.preceding_paragraph_fragment 属性

添加 Run.contains_page_break 属性

添加 Run.iter_inner_content()函数

添加 Section.iter_inner_content()函数

1.0.1 版本：

修复＃1256：parse_xml()

添加 Hyperlink.fragment 属性

添加 Hyperlink.url 属性

1.1.0 版本：

添加 BlockItemContainer.iter_inner_content()函数

下面将利用 Python-docx 库解决办公自动化过程中遇到的问题。

1. 输出 Word 文件格式的确定

在使用 Word 进行办公自动化时，Word 文件格式一般都比较固定，这时可以通过设置或者使用模板来避免 Python 对 Word 文件格式的操作。

1）定制 Word 输出模板

在 Python-docx 安装目录下修改基本的样式，本机目录为/home/user/PycharmProjects/pythonProject2/venv/lib/python3.10/site-packages/docx/templates，找到 default.docx，设置其中的样式，下面以普查数据处理的输出格式设置为例进行说明。

标题：方正小标宋简体，小二号，居中，段后 15 磅，单倍行距；

标题 1：楷体，三号，居左，段前 0.5 行，段后 0.5 行，单倍行距；

正文：中文是宋体，西文为微软雅黑，四号，单倍行距。

下面的代码创建了一个新的 Word 文档，对其进行修改后保存为 test.docx。

```
from docx import Document                    #用来建立一个 Word 对象
#创建一个空白的 Word 文件
doc = Document()
#设置标题
para_heading0 = doc.add_heading('我是文章标题',level = 0)      #返回标题段落对象
para_heading = doc.add_heading('',level = 1)                   #返回一级标题段落对象
run = para_heading.add_run("一、我是一级标题第 1 个")
p = doc.add_paragraph()
r = p.add_run("安装的库版本是 0.8.11,Python - docx 在 2023 年 11 月 4 日发布了 1.1.0 版本,在编写本书时所有代码均使用的是 0.8.11 版本。"\
              + "安装的库版本是 0.8.11,Python - docx 在 2023 年 11 月 4 日发布了 1.1.0 版本,在编写本书时所有代码均使用的是 0.8.11 版本。"\
              + "安装的库版本是 0.8.11,Python - docx 在 2023 年 11 月 4 日发布了 1.1.0 版本,在编写本书时所有代码均使用的是 0.8.11 版本。"\
              + "安装的库版本是 0.8.11,Python - docx 在 2023 年 11 月 4 日发布了 1.1.0 版本,在编写本书时所有代码均使用的是 0.8.11 版本。")
para_heading = doc.add_heading('',level = 1)   #返回一级标题段落对象,标题也相当于一个段落
run = para_heading.add_run("二、我是一级标题第 2 个")
doc.save("test.docx")
```

程序没有任何设置格式的操作语句，只关注 Word 内容本身，可以提高编写代码的效率，当然这也缺少了灵活性，程序运行结果前后对比如图 3.18 和图 3.19 所示。

2）使用临时的空 Word 文件

对于不经常使用的 Word 样式，可以将一个 Word 目标文件复制到 Python 文件目录下，并且将 Word 内的内容清空，这里直接复制原来的模板，将之重新命名为 default-0.docx，修改程序，直接打开该 Word 文件，最后保存即可，这与第一种方法的主要区别在于，打开新文档默认有一个段落，后续插入的标题实际上是第二个段落，在进行段落操作时

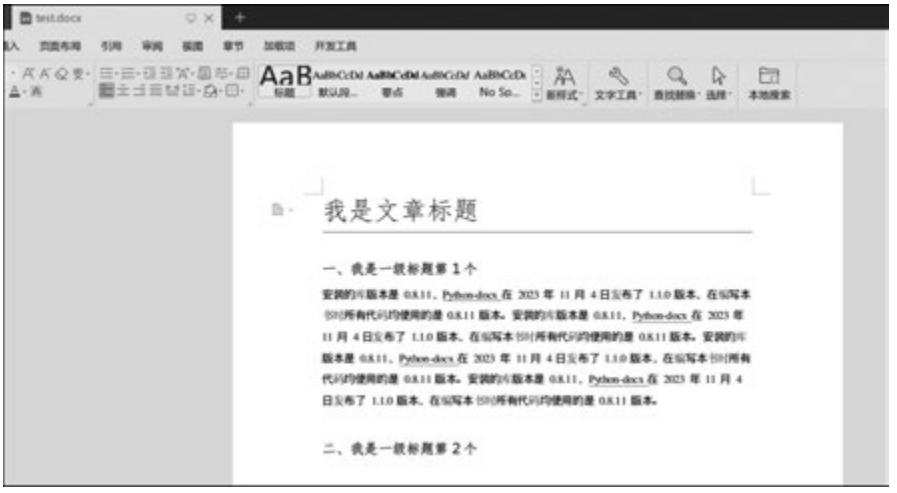


图 3.18　修改模板样式前的 Word 文件输出

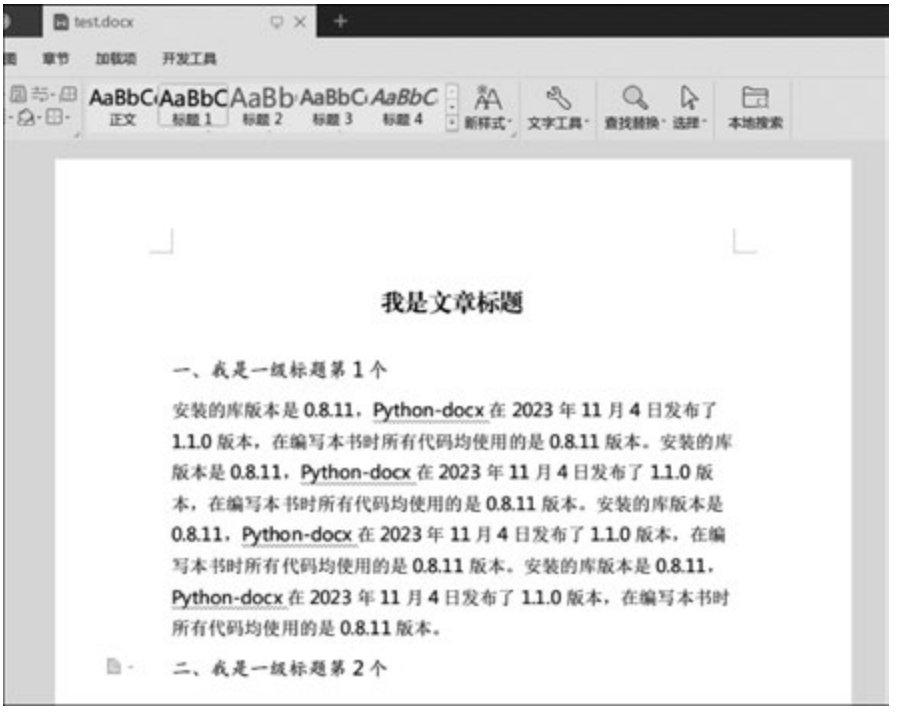


图 3.19　修改模板样式后的 Word 输出

要注意区分。

实例代码除以下不同外，其他同第一种方式：

```
#创建一个空白的 Word 文档
doc = Document("default - 0.docx")
...
doc.save("test - 0.docx")
```

程序运行后，打开 test-0.docx，其内容同 test.docx，与第一种方式的输出相同。

2. Word 标题操作

实例代码：

```
from docx import Document                                    #用来建立一个 Word 对象
from docx.shared import Pt                                   #用来设置字体的大小
from docx.shared import Inches
from docx.enum.text import WD_LINE_SPACING
from docx.enum.text import WD_PARAGRAPH_ALIGNMENT            #设置对齐方式
#创建一个空白的 Word 文档
doc = Document("default - 0.docx")
#设置标题
para_heading0 = doc.add_heading('我是文章标题',level = 0)    #返回标题段落对象
para_heading0.paragraph_format.alignment = WD_PARAGRAPH_ALIGNMENT.CENTER
para_heading0.paragraph_format.space_before = Pt(0)
para_heading0.paragraph_format.space_after = Pt(0)
para_heading0.paragraph_format.line_spacing = 1.5
para_heading0.paragraph_format.left_indent = Pt(0)
para_heading0.paragraph_format.right_indent = Inches(0)
para_heading1 = doc.add_heading('我是一级标题',level = 1)    #返回一级标题段落对象
para_heading1.paragraph_format.left_indent = Pt(0)
para_heading1.paragraph_format.page_break_before = True
para_heading1.paragraph_format.line_spacing_rule = WD_LINE_SPACING.DOUBLE
para_heading1.paragraph_format.line_spacing = 6.0
para_heading1.paragraph_format.first_line_indent = Pt( - 50)
para_heading2 = doc.add_heading('我是二级标题',level = 2)    #返回二级标题段落对象
para_heading3 = doc.add_heading('我是三级标题',level = 3)    #返回三级标题段落对象
doc.save("test - title.docx")
```

在对标题进行操作时，主要涉及几级标题，如果是 level＝0，则是指 Word 里的标题样式，其他数字对应标题 1～9，大于 9 会报错(level 的值必须是 0～9)，说明 Word 模板中没有对应的标题。另外一个问题就是标题格式的设置，通过 add_heading()函数添加标题，返回的是一个标题的段落对象，设置对齐格式，可以参考 Word 里的段落设置，在 Python-docx 中主要通过 paragraph_format 来进行选择和设置，paragraph_format 是 ParagraphFormat 类的一个实例，该类定义在 Python-docx 安装目录的 parfmt.py 文件中，本机目录为：/home/user/PycharmProjects/pythonProject2/venv/lib/python3.10/site-packages/docx/text/parfmt.py，文件定义了 ParagraphFormat 类，该类使用 Python 的装饰器装饰了几个属性供用户使用，具体如表 3.9 所示。

表 3.9 ParagraphFormat 属性、作用及使用方法

属 性	作 用	可 用 参 数
alignment	指定段落的对齐方式	WdParagraphAlignment 成员
first_line_indent	指定段落第一行缩进的相对差异的值。正值会使第一行缩进。负值会产生挂起的缩进	Pt(0)，数值可正可负，负表示向左侧
keep_together	段落是否保持“完整”，不跨页面	True 或者 None
keep_with_next	该段是否应与下一段保持在同一页上。例如，此属性可用于将节标题与其第一段保持在同一页上	True 或者 None
left_indent	段落左缩进	Inches(0)
line_spacing	设置行距，可以设任意值	任意数值
line_spacing_rule	选择可用的参数来设置行距	WD_LINE_SPACING 成员

续表

属　性	作　用	可用参数
page_break_before	是否这一段出现在页面的顶部	True 或者 None
right_indent	段落右缩进	Inches(0)
space_after	指定此段落和后面一段落间的间隔,默认继承模板样式	Pt(0)等
space_before	指定此段落和前一段落间的间隔,默认继承模板样式	Pt(0)等

在操作格式时,使用了几个主要文件,这些文件定义了相关的类或者枚举对象,具体有docx/shared.py、docx/enum/text.py,其中,shared.py 文件定义了长度相关的类,具体有Inches、Cm、Emu、Mm、Pt、Twips 和 RGBColor; text.py 文件定义了 WD_PARAGRAPH_ALIGNMENT、WD_BREAK_TYPE、WD_COLOR_INDEX、WD_LINE_SPACING、WD_TAB_ALIGNMENT、WD_TAB_LEADER、WD_UNDERLINE 等包含已定义相关枚举属性的类,可以通过导入直接访问。

3. Word 段落操作

再次说明:以下对段落的定位都是基于打开临时文件创建的 Word,在开始默认存在一个段落,如果从模板创建,则不存在这个段落。

实例代码:

```
from docx import Document                          #用来建立一个 Word 对象
#创建一个空白的 Word 文件
doc = Document("default - 0.docx")
#设置标题
para_heading0 = doc.add_heading('Paragraph 测试',level = 0)        #返回标题段落对象
para_heading1 = doc.add_heading('一、添加新的段落',level = 1)      #返回一级标题段落对象
#显示此句代码前总的段落个数,默认打开的文档有一个,增加两个标题两个,共 3 个
print(doc.paragraphs)
doc.add_paragraph("这是第一个文本段落的内容,在全文中,我不是第一个段落")
para_heading2 = doc.add_heading('二、修改段落样式',level = 1)      #返回二级标题段落对象
#显示第三段的文本
print(doc.paragraphs[2].text)
#增加新一页
doc.add_page_break()
#在指定段落前插入新的段落
doc.paragraphs[3].insert_paragraph_before("该段在第四段之前插入,也就是在"这是第一个文本段落的内容,在全文中我不是第一个段落"前插入")
doc.save("test - paragraph.docx")
```

在文档的末尾插入段落可以直接使用 Document 类的 add_paragraph()函数,如果想定位在某个段落前插入,可以使用 Document 的 Paragraphs 属性定位到具体段落。下面介绍 Document 类的主要函数和 Paragraph 类的主要函数,Document 类的主要函数介绍如表 3.10 所示。

表 3.10　Document 类的主要函数介绍

函　数	参　数	返回值类型	作　用
add_heading(text="", level=1)	text:标题文本;level:标题号	返回新添加到文档末尾的段落	插入指定标题格式的标题
add_page_break()	无参数	返回仅包含分页符的段落对象	插入新的一页

续表

函　数	参　数	返回值类型	作　用
add_paragraph(text='', style=None):	text：段落文本；style：文本样式	返回新添加到文档末尾的段落	插入一个新的文本段落
add_picture(image_path_or_stream, width=None, height=None):	image_path_or_stream：图片路径；width：宽；height：高	返回一个 inlineshape 类的图形实例	插入图片
add_section(start_type=WD_SECTION.NEW_PAGE)	start_type：WdSectionStart 的枚举成员	返回 Section 类的实例	插入节符号
add_table(rows,cols, style=None)	rows：行数；cols：列数；style：样式	返回 Table 对象的实例	插入表格
inline_shapes	属性	返回 InlineShape 类型的序列，可以使用 len()查看长度，并通过下标访问	得到 InlineShape 序列
part	属性	此文档的 DocumentPart 对象	得到 DocumentPart 对象
sections	属性	sections	提供对文档每一节的访问
tables	属性	table 序列	文档中按顺序对应的 table 列表，不包括嵌套表格
settings	属性	settings	提供对文档 settings 的访问
styles	属性	styles	提供对文档 styles 的访问

其中，WdSectionStart 类在 docx/enum/section.py 文件中定义，导入后可以使用其中的枚举成员；Section 类在 docx/section.py 文件中定义，定义了对每一节的属性进行操作的函数；Table 类在 docx/table.py 文件中定义，定义了对表格内容和样式进行操作的函数；InlineShape 类在 docx/shape.py 中定义，主要进行插入对象宽度和高度的修改；DocumentPart 类在 docx/parts/document.py 中定义，是 WordprocessingML(WML)包的主文档部分，也称为.docx 文件，包括了对文档部分对象的引用，比如页眉、页脚、文档样式信息等。

Paragraph 类的主要函数介绍如表 3.11 所示。

表 3.11　Paragraph 类的主要函数介绍

函　数	参　数	返回值类型	作　用
add_run(text=None, style=None)	text：文本；style：样式	Run	在包含 * text * 并具有由样式 ID * style * 标识的字符样式的段落中附加一段 * text * 。可以包含制表符(\\t)，这些字符被转换为制表符的适当 XML 形式。 * text * 还可以包括换行符(\\n)或回车符(\\r)，每个字符都被转换为换行符，返回 Run 类的对象实例

续表

函　数	参　数	返回值类型	作　用
alignment	属性	无	WdParagraphAlignment 枚举指定此段落的对齐设置
clear()	无	paragraph	删除所有内容后返回同一段落。保留段落级别的格式，例如样式
insert_paragraph_before (text=None,style=None)	text：文本；style：样式	paragraph	在此段前插入新的段落，包含 text 文本和 style 样式
paragraph_format	属性	无	ParagraphFormat 对象，用于访问此段落的格式设置属性，如行距和缩进
runs	属性	无	与该段落中的<w:r>元素相对应的 Run 实例的序列
style	属性	BaseStyle 的子类	读/写。ParagraphStyle 对象，表示分配给该段落的样式。如果未为此段落指定明确的样式，则其值为文档的默认段落样式。可以指定段落样式名称来代替段落样式对象。指定 None 将删除任何应用的样式
text	属性	Text	设置段落的内容

BaseStyle 是各种类型，包括对象、段落、字符、表格和编号的样式基类。BaseStyle 类中的属性和方法由所有样式对象继承，虽然通过 style 可以更改段落样式，但是 Python-docx 类库的作者给出的最终建议是：此值可由 Word 重写，除非对所涉及的内部结构非常了解，否则通常不应更改。那么怎样更改段落的样式呢？可以通过 paragraph 的 paragraph_format 和 Run 的对象实现。Run 类是关于 paragraph 操作的一个重要的类，通过该类的对象可以实现很多 Word 操作，当使用 Run 类对象操作文本时会保留原来段落的格式，其主要函数介绍如表 3.12 所示。

表 3.12　Run 类的主要函数介绍

函　数	参　数	返回值类型	作　用
add_break(break_type=WD_BREAK.LINE)	break_type：WD_BREAK 的一种类型	无	添加一个终止的类型，可以是添加新的一行或者一页
add_picture(image_path_or_stream,width=None,height=None)	mage_path_or_stream：图片路径；width：宽，height：高	inlineshape 类的图形实例	插入图片
add_tab()	无	无	类似 Tab 键
add_text(text)	text：插入的文本	Text	插入文本
bold	属性	无	加粗
clear()	无	Run	清除内容，保留格式
font	属性	无	设置字体，使用 Font 类的属性
italic	属性	无	斜体
underline	属性	无	下画线

要深入理解字体相关设置，就有必要详细研究 Font 类，它提供对字符属性（如字体名称、字体大小、粗体和下标）的访问，该类的属性、可取值及作用如表 3.13 所示。

表 3.13 Font 类介绍

属 性	可 取 值	作 用
all_caps	True、False、None	使此字体中的文本以大写字母显示
bold	True、False、None	使此字体中的文本以粗体显示
color	ColorFormat	提供修改字体颜色的接口，需使用 ColorFormat 类实例的属性
double_strike	True、False、None	使此字体中的文本出现双删除线
emboss	True、False、None	使此字体中的文本看起来像是从页面上凸起的浮雕
hidden	True、False、None	使此字体中的文本从显示中隐藏，除非应用程序设置强制显示隐藏的文本
highlight_color	WdColorIndex	使此字体中的文本以某种颜色突出显示
imprint	True、False、None	使此字体中的文本显示为好像被压入页面一样
outline	True、False、None	通过在每个字符字形的内外边界周围绘制一个像素宽的边界，使运行中的字符看起来像有轮廓一样
shadow	True、False、None	使此字体中的文本有阴影效果
size	Length 子类表示长度大小，例如 Pt	设置文本字号大小
snap_to_grid	True、False、None	使此字体中的文本使用文档中由 docGrid 元素定义的每行文档网格字符设置，即每页有多少行，每行有多少字符
strike	True、False、None	使此字体中的文本显示为一条穿过行中心的水平线
subscript	True、False、None	指示此 Font 中的字符是否显示为下标
underline	True、False、None 或者 WdUnderline 中的线型	使此字体中的文本显示下画线
web_hidden	True、False、None	指定当文档显示在网页视图中时，应隐藏此运行的内容

ColorFormat 类在文件 docx/dml/color.py 中定义，提供对颜色设置的访问，例如，RGB 颜色、主题颜色和亮度调整，在修改字体颜色时可以通过该类的 rgb 属性来修改，具体请参见下面的实例。WdColorIndex 是 WD_COLOR_INDEX 类的别名，在 docx/enum/text.py 中定义，主要包括一些常用颜色的索引。

关于部分样式操作的实例代码如下：

```
#修改样式
para_wait_for_update = doc.add_paragraph()
para_wait_for_update.paragraph_format.left_indent = Inches(2)        #设置左缩进
para_wait_for_update.paragraph_format.right_indent = Inches(0)       #设置右缩进
run_wait_for_update = para_wait_for_update.add_run("我的样式可以修改了。do it!")
run_wait_for_update.add_break(break_type = WD_BREAK.LINE_CLEAR_LEFT)
run_wait_for_update.underline = True
run_wait_for_update.font.all_caps = None
run_wait_for_update.font.color.rgb = RGBColor(0,0,0)
#run_wait_for_update.font.emboss = True
#run_wait_for_update.font.outline = True
#run_wait_for_update.font.imprint = True
run_wait_for_update.font.snap_to_grid = True
run_wait_for_update.font.size = Pt(24)
print(para_wait_for_update.style)
```

4. Word 表格操作

在进行表格操作时，必须先使用 document 的 add_table()函数添加表格，然后使用返回的 table 实例来操作表格。

实例代码：

```
from docx import Document                                    #用来建立一个 Word 对象
from docx.shared import Emu
from docx.enum.table import WD_ALIGN_VERTICAL
#创建一个空白的 Word 文档
doc = Document("default - 0.docx")
#设置标题
para_heading0 = doc.add_heading('Table 测试',level = 0)     #返回标题段落对象
para_heading1 = doc.add_heading('一、此代码显示的是 Word 表格操作相关说明',level = 1)
#返回一级标题段落对象
#添加表格 3 行、2 列
my_table = doc.add_table(3,2)
#右侧插入一列
my_table.add_column(Emu(457200))
#打印第一行的 cells
print(my_table.rows[0].cells)
#在第一行第一个 cell 中添加一个新的 2 行 2 列的表格
my_table.rows[0].cells[0].add_table(2,2)
#合并第二行第二个 cell 和第三个 cell
my_table.rows[1].cells[1].merge(my_table.rows[1].cells[2])
print(len(my_table.rows[0].cells[0].tables))
#在单元格第一行第一列中插入文本
my_table.rows[0].cells[0].add_paragraph("我是第 1 行第 1 列")
print(my_table.rows[0].cells[0].text)
#整个单元格替换
my_table.rows[0].cells[0].text = "新的文本"
#设置第 1 行第 1 列单元格文本垂直居中
my_table.rows[0].cells[0].vertical_alignment = WD_ALIGN_VERTICAL.CENTER
print(my_table.rows[0].cells[0].text)
doc.save("test - table.docx")
```

Table 类的主要函数介绍如表 3.14 所示。

表 3.14 Table 类的主要函数介绍

函　数	参　数	返回值类型	作　用
add_column(width)	width=Emu(457200)	_Column	在右侧插入由 Emu 定义宽度的一列
add_row()	无	_Row	在表格下方插入一行
alignment	属性	无	对齐，使用 WdRowAlignment 中枚举成员赋值
autofit	属性	无	是否自动调节列宽
cell(row_idx,col_idx)	row_idx：行序号；col_idx：列序号	_Cell	读取某个表格单元格的实例，(0,0)表示左上方单元格
column_cells(column_idx)	column_idx：列序号	_Cell 列表	返回表格某一列的所有单元格实例的列表

续表

函　数	参　数	返回值类型	作　用
row_cells(row_idx)	row_idx：行序号	_Cell 列表	返回表格某一行的所有单元格实例的列表

style	属性	无	#获取所有表格样式
			from docx.enum.style import WD_STYLE_TYPE
			styles = doc.styles
			for style in styles:
			　　if style.type == WD_STYLE_TYPE.TABLE:
			　　　　print(style)
			该值的作用是设置表格样式，通过以上代码可以查看可以设置的所有表格样式

columns	属性	_Columns 列表	得到一个包括表格每一列的列表
rows	属性	_Rows 列表	得到一个包括表格每一行的列表

在对表格进行操作时，返回的数据类型包括_Column、_Cell、_Row，下面简单介绍 3 个类的主要函数及可以实现的功能。

_Cell 类的主要函数介绍如表 3.15 所示。

表 3.15　_Cell 类的主要函数介绍

函　数	参　数	返回值类型	作　用
add_paragraph(text='', style=None)：	text：文本；style：样式	paragraph	指定单元格里添加文本
add_table(rows, cols)	rows：行数；cols：列数	table	指定单元格里添加新的表格
merge(other_cell)	other_cell：另一个_Cell 对象	_Cell	合并该单元格到另一个单元格之间的单元格
paragraphs	属性	paragraph 列表	读取单元格中所有段落
tables	属性	table 列表	读取单元格中所有表格
text	属性	文本	可读可写，写的时候会清除单元格中所有内容，用 text 文本替换
vertical_alignment	属性，WD_CELL_VERTICAL_ALIGNMENT 类的枚举元素或者 None	无	设置单元格垂直对齐方式
width	属性	无	单元格宽度，可赋值由 Emu 定义的长度值

WD_CELL_VERTICAL_ALIGNMENT 在 docx/enum/table.py 中定义，里面有垂直对齐的几种值可供选择。_Column 类主要有 cells 和 width 属性，其中 cells 为只读属性；_Columns 类提供了迭代器，方便访问其中的成员；_Row 类主要有 cells 和 height 属性，其中 cells 为只读属性；_Rows 类提供了迭代器，方便访问其中的成员；这 4 个类都提供了 table 属性，返回其对象属于哪一个表格对象，通过该属性可以直接访问表格。

5. Word 图片操作

python-docx 对图片操作提供的方法有限，在 InlineShape 类中仅提供了 height、width 和 type 属性，其中，type 为只读属性，也就是说，只能通过 InlineShape 类改变图片的大小；这里建议使用 run 类的添加图片功能，可以通过 run 类和 paragraph 类的其他函数来实现图片居中等排版操作。

注意：通过在添加文本中加入\\r 或者\\n 并不能实现换行。换行可以使用 run 的 add_break()函数来实现。

实例代码：

```
from docx import Document                                    #用来建立一个 Word 对象
from docx.enum.text import WD_ALIGN_PARAGRAPH                #设置对其方式
from docx.oxml.ns import qn                                  #设置字体
from docx.shared import RGBColor                             #设置字体的颜色
from docx.shared import Pt                                   #用来设置字体的大小
#创建一个空白的 Word 文档
doc = Document("default - 0.docx")
#设置标题
para_heading0 = doc.add_heading('图片测试',level = 0)        #返回标题段落对象
para_heading1 = doc.add_heading('一、此代码显示的是 Word 图片操作相关说明',level = 1)
                                                             #返回一级标题段落对象
#document 类函数插入图片
doc.add_picture("title.png")
doc.add_paragraph("图 1 第一种插入图片方式")
#run 类函数插入图片
my_para = doc.add_paragraph()
my_run = my_para.add_run()
my_run.add_picture("title.png")
#使用段落对齐方式将图片居中
my_para.alignment = WD_ALIGN_PARAGRAPH.CENTER                #设置为左对齐
my_para.paragraph_format.space_before = Pt(0)                #设置段前 0 磅
my_para.paragraph_format.space_after = Pt(0)                 #设置段后 0 磅
#实现换行
my_run.add_break()
#添加图片说明,设置字体
my_run.add_text("图 1 图片说明")
my_run.font.name = u'宋体'                                   #设置为宋体
my_run._element.rPr.rFonts.set(qn('w:eastAsia'), u'宋体')    #设置为宋体,和上边的一起使用
my_run.font.size = Pt(12)        #设置 1 级标题文字的大小为"小四",即 12 磅
my_run.font.color.rgb = RGBColor(0,0,0)                      #设置颜色为黑色
doc.save("test - pic.docx")
```

操作结果如图 3.20 所示。

6. Word 页面属性操作

使用 python-docx 操作 Word 页面属性和页眉页脚主要通过 Section 类实现。

实例代码：

```
from docx import Document                    #用来建立一个 Word 对象
from docx.enum.section import WD_ORIENT, WD_SECTION
from docx.enum.text import WD_BREAK
from docx.oxml.ns import qn                  #设置字体
from docx.shared import RGBColor             #设置字体的颜色
```

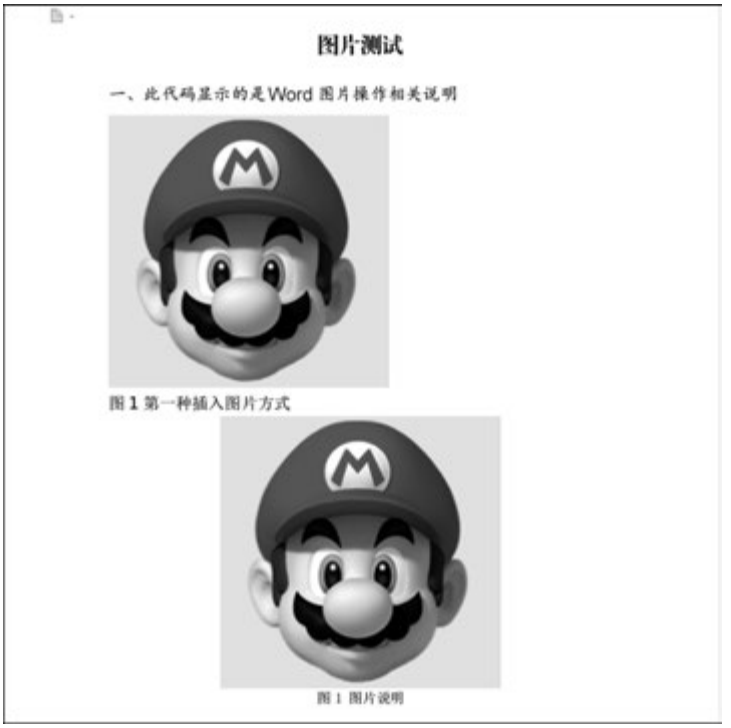


图 3.20 图片操作结果

```
from docx.shared import Inches
from docx.shared import Emu
#创建一个空白的 Word 文档
doc = Document("default - 0.docx")
#设置标题
para_heading0 = doc.add_heading('页面设置测试',level = 0)      #返回标题段落对象
para_heading1 = doc.add_heading('一、此代码显示的是 Word 页面操作相关说明',level = 1)
                                                                #返回一级标题段落对象
doc.add_paragraph("这里是第一节")
#添加一个新的节,默认是新起一页
doc.add_section()
doc.add_paragraph("这里是第二节")
print(len(doc.sections))
#修改第二节属性
print(doc.sections[1].bottom_margin)
#设置装订线距离
doc.sections[1].gutter = Emu(914400)
#设置页面方向,需要设置页面宽度和高度才能实现
doc.sections[1].page_height = Inches(8.5)
doc.sections[1].page_width = Inches(11)
my_para = doc.add_paragraph("还是第二节")
my_run = my_para.add_run()
#插入新的一页,通过这种方式可以人为控制下一节从哪一页开始
my_run.add_break(break_type = WD_BREAK.PAGE)
doc.add_paragraph("还是第二节,对啊.")
print(doc.sections[0].header.paragraphs)
print(doc.sections[1].header.paragraphs)
#  #不分奇数和偶数页,如果存在多个节,可以使用任意一个节的页眉页脚设置
```

```
# doc.sections[0].header.paragraphs[0].text = "这是整个文档页眉"
# doc.sections[0].footer.paragraphs[0].text = "这是整个文档页脚"
#设置奇数偶数页眉页脚不同,所有节共同设置
doc.settings.odd_and_even_pages_header_footer = True
doc.sections[0].even_page_header.paragraphs[0].text = "偶数页页眉"
doc.sections[0].header.paragraphs[0].text = "奇数页页眉"
doc.sections[0].even_page_footer.paragraphs[0].text = "偶数页页脚"
doc.sections[0].footer.paragraphs[0].text = "奇数页页脚"
#首页不同的页眉页脚
doc.sections[0].different_first_page_header_footer = True
doc.sections[0].first_page_header.paragraphs[0].text = "首页页眉"
doc.sections[0].first_page_footer.paragraphs[0].text = "首页页脚"
```

Section 类的主要函数介绍如表 3.16 所示。

表 3.16 Section 类主要函数介绍

函　数	参　数	返回值类型	作　用
bottom_margin	属性	Length 子类 Emu	访问页面下边距,以 Emu 定义的单位表示
different_first_page_header_footer	属性	无	True 表示此节首页页眉页脚不同
even_page_footer	属性	_Footer	访问偶数页页脚的对象
even_page_header	属性	_Header	访问偶数页页眉的对象
first_page_footer	属性	_Footer	访问首页页脚的对象
first_page_header	属性	_Header	访问首页页眉的对象
footer	属性	_Footer	访问该节默认页脚对象
footer_distance	属性	无	访问页面底边到页脚底边的距离
gutter	属性	无	访问装订线距离,以 Emu 定义的单位表示,表示页边距以外的额外距离
header	属性	_Header	访问该节默认页眉对象
header_distance	属性	无	访问页面上边缘到页眉上边缘的距离
left_margin	属性	无	访问左边距,以 Emu 定义的单位表示
orientation	属性	无	访问此节的页面方向,可以取值为 WD_ORIENT.PORTRAIT 或者 WD_ORIENT.LANDSCAPE
page_height	属性	无	访问页面高度,如果页面设置为横向,此值必须设置,可取 Inches(8.5)
page_width	属性	无	访问页面宽度,如果页面设置为横向,此值必须设置,可取 Inches(11)
right_margin	属性	无	访问右边距,以 Emu 定义的单位表示
start_type	属性	无	指定分节符的起始类型,WD_SECTION_START 枚举成员
top_margin	属性	无	访问上边距,以 Emu 定义的单位表示

WD_ORIENT 和 WD_SECTION_START 在 docx/enum/section.py 文件中定义,其中,WD_SECTION_START 可取的值有 CONTINUOUS、NEW_COLUMN、NEW_PAGE、EVEN_PAGE 和 ODD_PAGE。

3.3.4 使用 python-docx-template 操作 Word

在 PyCharm 下使用命令行安装,命令为:pip install docxtpl,不要通过“设置”菜单去安

装 python-docx-template。

docxtpl 库是 Python 中的一个文档自动化工具，docxtpl 是基于 Jinja2 模板引擎开发的，用该库可以根据模板动态生成. docx 文件。模板需要提前制作好，在需要填充数据的地方使用了 Jinja2 模板引擎的语法，例如，用{{ user_name }}来表示变量 user_name，生成的文档可以包含文本、图片、表格和包含图片的表格等。在 docxtpl 中，可以为模板中的文本、图片和表格设置变量赋值，使用变量和数据来动态填充模板，生成具有特定格式的. docx 文档。

以检查某广场的问题反馈文档为例，制作的模板文件名称为 guangchang. docx，模板的内容如下：

```
广 场 物 业 服 务 抽 查 问 题 清 单
抽查广场：{{ guangchangmingzi }}
抽查时间：{{ riqi }}
抽查人员：张三、李四
```

发现问题如下：

```
{ % for item in itemList % }
{{ item.biaoti }}
{{ item.image }}
{ % endfor % }
```

工作现场照片如下：

```
{{ xianchangzhaopian }}
检查单位:XXXX
                                        {{ riqi }}
```

用双大括号括起来的是变量，变量可以是文本，也可以是图片，后面将用 Python 的 docxtpl 库进行填充和渲染，{%for item in itemList%}、{%endfor%}为 for 循环，也就是说，可以实现以同样的结构插入很多遍数据。

代码实例：

```
from docx.shared import Mm
from docxtpl import DocxTemplate, InlineImage
import os
import sys
import copy
images_path = "images/"
tpl = DocxTemplate('guangchang.docx')
if len(os.listdir(images_path)) == 0:
    print("gc dirs have no files! exit now.")
    sys.exit(0)
else:
    # data for render
    context = {
        'guangchangmingzi': '',
        'riqi': '',
        'shuliang': '0',
        'itemList': []
    }
    str_times = ""
    # save biaoti and image by gcmz
    gc_context_dic = {"whgc": copy.deepcopy(context), "hczgc": copy.deepcopy(context),
"bhgc": copy.deepcopy(context),
```

```
                        "smgc": copy.deepcopy(context), "nzgc": copy.deepcopy(context)}
    for file_name in os.listdir(images_path):
        # itemList_dic.clear()
        itemList_dic = {}
        print(file_name)
        insertImage = InlineImage(tpl, images_path + file_name, width = Mm(150), height =
Mm(160))
        strs_list = file_name.split('.')[0].split('-')
        # print(strs_list[0])
        if str_times == "":
            str_times = strs_list[3][0:4] + '年' + strs_list[3][4:6] + '月' +
strs_list[3][6:8] + '日'
        if strs_list[1] == "文化广场":
            gc_context_dic["whgc"]["guangchangmingzi"] = strs_list[1]
            gc_context_dic["whgc"]["riqi"] = str_times
            if strs_list[2] == "现场照片":
                gc_context_dic["whgc"]["xianchangzhaopian"] = insertImage
            else:
                itemList_dic['biaoti'] = str(len(gc_context_dic["whgc"]["itemList"]) + 1) +
'、' + strs_list[2]
                itemList_dic['image'] = insertImage
                print(itemList_dic)
                gc_context_dic["whgc"]["itemList"].append(itemList_dic)
        if strs_list[1] == "火车站广场":
            gc_context_dic["hczgc"]["guangchangmingzi"] = strs_list[1]
            gc_context_dic["hczgc"]["riqi"] = str_times
            if strs_list[2] == "现场照片":
                gc_context_dic["hczgc"]["xianchangzhaopian"] = insertImage
            else:
                itemList_dic['biaoti'] = str(len(gc_context_dic["hczgc"]["itemList"]) +
1) + '、' + strs_list[2]
                itemList_dic['image'] = insertImage
                gc_context_dic["hczgc"]["itemList"].append(itemList_dic)
        if strs_list[1] == "滨海广场":
            gc_context_dic["bhgc"]["guangchangmingzi"] = strs_list[1]
            gc_context_dic["bhgc"]["riqi"] = str_times
            if strs_list[2] == "现场照片":
                gc_context_dic["bhgc"]["xianchangzhaopian"] = insertImage
            else:
                itemList_dic['biaoti'] = str(len(gc_context_dic["bhgc"]["itemList"]) + 1) +
'、' + strs_list[2]
                itemList_dic['image'] = insertImage
                gc_context_dic["bhgc"]["itemList"].append(itemList_dic)
        if strs_list[1] == "市民广场":
            gc_context_dic["smgc"]["guangchangmingzi"] = strs_list[1]
            gc_context_dic["smgc"]["riqi"] = str_times
            if strs_list[2] == "现场照片":
                gc_context_dic["smgc"]["xianchangzhaopian"] = insertImage
            else:
                itemList_dic['biaoti'] = str(len(gc_context_dic["smgc"]["itemList"]) + 1) +
'、' + strs_list[2]
                itemList_dic['image'] = insertImage
                gc_context_dic["smgc"]["itemList"].append(itemList_dic)
        if strs_list[1] == "南站广场":
            gc_context_dic["nzgc"]["guangchangmingzi"] = strs_list[1]
            gc_context_dic["nzgc"]["riqi"] = str_times
            if strs_list[2] == "现场照片":
```

```
                gc_context_dic["nzgc"]["xianchangzhaopian"] = insertImage
            else:
                itemList_dic['biaoti'] = str(len(gc_context_dic["nzgc"]["itemList"]) + 1) +
'、' + strs_list[2]
                itemList_dic['image'] = insertImage
                gc_context_dic["nzgc"]["itemList"].append(itemList_dic)
        # print(itemList_dic)
    gc_context_dic["whgc"]["shuliang"] = str(len(gc_context_dic["whgc"]["itemList"]))
    print(gc_context_dic["whgc"])
    if gc_context_dic["whgc"]["guangchangmingzi"]:
        tpl.render(gc_context_dic["whgc"])
        tpl.save(gc_context_dic["whgc"]["guangchangmingzi"] + str_times + '.docx')
    gc_context_dic["hczgc"]["shuliang"] = str(len(gc_context_dic["hczgc"]["itemList"]))
    if gc_context_dic["hczgc"]["guangchangmingzi"]:
        tpl.render(gc_context_dic["hczgc"])
        tpl.save(gc_context_dic["hczgc"]["guangchangmingzi"] + str_times + '.docx')
    gc_context_dic["bhgc"]["shuliang"] = str(len(gc_context_dic["bhgc"]["itemList"]))
    if gc_context_dic["bhgc"]["guangchangmingzi"]:
        tpl.render(gc_context_dic["bhgc"])
        tpl.save(gc_context_dic["bhgc"]["guangchangmingzi"] + str_times + '.docx')
    gc_context_dic["smgc"]["shuliang"] = str(len(gc_context_dic["smgc"]["itemList"]))
    if gc_context_dic["smgc"]["guangchangmingzi"]:
        tpl.render(gc_context_dic["smgc"])
        tpl.save(gc_context_dic["smgc"]["guangchangmingzi"] + str_times + '.docx')
    gc_context_dic["nzgc"]["shuliang"] = str(len(gc_context_dic["nzgc"]["itemList"]))
    if gc_context_dic["nzgc"]["guangchangmingzi"]:
        tpl.render(gc_context_dic["nzgc"])
        tpl.save(gc_context_dic["nzgc"]["guangchangmingzi"] + str_times + '.docx')
```

使用的模板和对模板填充渲染后的效果对比如图 3.21 和图 3.22 所示。

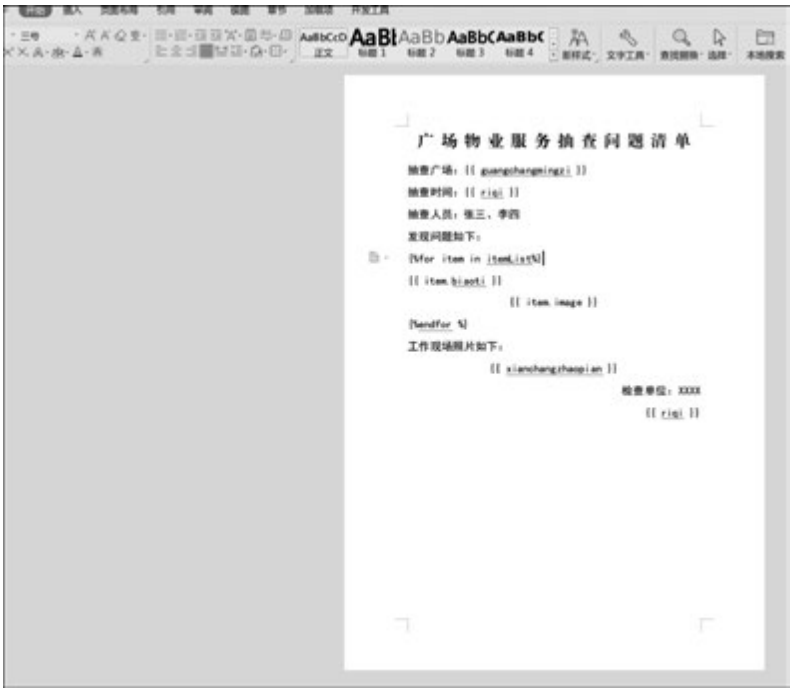


图 3.21 使用的模板

图 3.22　使用 docxtpl 对模板填充渲染后的效果

注意：图片过大会导致渲染失败，一般插入图片不应大于 2MB!

3.4　WPS 演示文稿及其自动化处理

3.4.1　WPS 演示文稿介绍

WPS 演示文稿是一款由金山软件公司开发的演示文稿制作软件，它是 WPS Office 套件的重要组成部分。

WPS 演示文稿提供了丰富的功能和工具，帮助用户轻松创建专业的演示文稿，其具有的主要特点如下：

（1）界面简洁易用。WPS 演示文稿的界面设计简洁明了，用户可以快速找到所需的功能和工具，无须花费大量时间学习软件操作。

（2）模板丰富。WPS 演示文稿提供了多种内置模板，涵盖了各种主题和场景，用户可以根据需要选择合适的模板进行编辑，节省制作时间。

（3）强大的编辑功能。WPS 演示文稿支持插入文本、图片、表格、图表、音频、视频等多种元素，满足用户在演示文稿中展示各种信息的需求。同时，用户还可以对文本和图片进行样式调整，实现个性化设计。

（4）动画效果。WPS 演示文稿支持为幻灯片中的文本、图片等元素添加动画效果，使演示文稿更具动感和吸引力。

（5）幻灯片切换效果。WPS 演示文稿提供了多种幻灯片切换效果，用户可以根据需要选择合适的效果，增强演示文稿的观赏性。

（6）演讲者模式。WPS 演示文稿支持演讲者模式，用户在演示时可以看到备注信息，方便进行讲解。

（7）跨平台兼容。WPS 演示文稿支持 Windows、macOS、Linux、iOS、Android 等多个

平台，用户可以在不同设备上查看和编辑演示文稿。

（8）云存储同步。WPS 演示文稿支持与金山云存储同步，用户可以将演示文稿保存在云端，实现跨设备访问和共享。

总之，WPS 演示文稿是一款功能强大、易于使用的演示文稿制作软件，适用于各种场景，如商务汇报、教学讲座、产品展示等。

3.4.2 WPS 演示文稿结构

WPS 演示文稿的结构与 3.3.2 节 WPS 文字结构基本相同，主要可操作的对象有页面布局、字体设置、段落格式、标题和目录、列表和编号、表格和图表、图片和形状、页眉和页脚、超链接和书签等。与 WPS 文字最大的不同点体现在对象的布局上，WPS 演示文稿的对象不但可以进行平面布局，而且可以在第三维度布局，主要体现在对象的堆叠上。

在制作 WPS 演示文稿时，通常要做好以下几处关键的 WPS 页面设计：

- 封面页。这是演示文稿的第一页，通常包含主题、演讲者的名字和日期等信息。封面页的设计应吸引人，因为它是观众对整个演示的第一印象。
- 目录页。目录页列出了演示文稿的主要部分或章节，帮助观众了解演示文稿的整体结构。这可以作为导航，使观众能够跟踪演示的进度。
- 内容页。内容页是演示文稿的主体部分，每一页都应围绕一个中心主题或概念展开。在内容页中，可以使用智能图形来展示层次结构、流程图等信息，这样可以使内容更加清晰易懂。
- 结束页。结束页通常包含总结信息和行动号召，以及相关的联系信息或致谢。

在进行每个页面设计时，尤其要关注两点：布局设计和模板的使用。其中，布局设计是指在三维空间中对页面包含对象的摆放和堆叠，既要考虑空间合理性，又要考虑颜色的协调性。合理的布局设计可以使演示文稿更加美观，清晰地传递信息，增强观众的注意力和理解力。例如，标题布局通常将标题置于幻灯片顶部居中的位置，以凸显出来并与内容块紧密结合；使用模板是指充分利用 WPS 演示文稿提供的多种内置模板，根据用户自身演示的需要选择合适的模板进行编辑，节省制作时间。

在创建演示文稿时，建议从大纲开始，然后根据大纲添加幻灯片和内容。这样不仅可以确保内容的连贯性，还可以在设计过程中保持清晰的思路，设计出自然、美观的演示文稿。

3.4.3 使用 python-pptx 操作 PowerPoint

操作 PowerPoint 使用的库是 python-pptx，最新版本是 0.6.22，不依赖 PowerPoint 程序，可以通过“pip install python-pptx”命令安装，同时会将其依赖的 3 个包也安装好，包括 lxml、Pillow 和 XlsxWriter。使用 python-pptx 库可以创建、读取和修改.pptx 文件。下面将跟随官方文档详细解析 python-pptx 操作.pptx 文件的相关内容。

母版文件中第一个幻灯片母版的内容如图 3.23 所示。

以下操作均基于此幻灯片母版进行。幻灯片母版查看方法是：使用 WPS 2019 打开要查看母版的.pptx 文件，单击“视图”菜单，再单击幻灯片母版即可。

1. 添加幻灯片之标题幻灯片

与 python-docx 类似，python-pptx 也是从模板创建.pptx 文件的，可以通过替换模板或

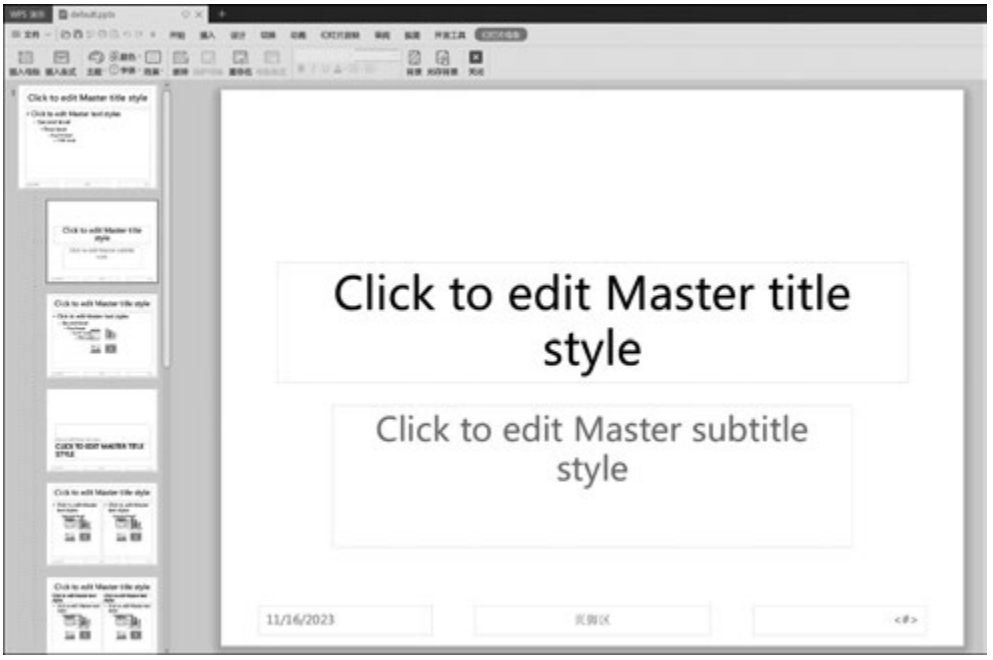


图 3.23　第一张幻灯片母版

者从已存在的.pptx 文件创建新的.pptx 文件，具体操作方法可以参考 3.3.3 节使用 Word 模板的相关介绍。python-pptx 的模板存放在该库的安装目录/pptx/templates 下，无论从哪一个模板创建.pptx 文件都会继承对应的默认母版和样式。

从 default.pptx 创建.pptx 文件代码如下：

```
from pptx import Presentation
prs = Presentation()
prs.save('start.pptx')
```

打开已存在的.pptx 文件来创建新的.pptx 文件代码如下：

```
from pptx import Presentation
prs = Presentation('my.pptx')
prs.save('my_new.pptx')
```

通过打开原有.pptx 文件创建新的.pptx，并不会清空原有.pptx 文件的内容，只是相当于复制重命名。通过打开现有.pptx 文件创建新文件，可能导致在接下来的操作过程中报错，比如下面新建一张幻灯片的代码在打开已有.pptx 文件时报错。

```
from pptx import Presentation
prs = Presentation("1.pptx")
title_slide_layout = prs.slide_layouts[0]
slide = prs.slides.add_slide(title_slide_layout)
title = slide.shapes.title
subtitle = slide.placeholders[1]
title.text = "Hello, World!"
subtitle.text = "python-pptx was here!"
prs.save('start.pptx')
```

错误提示：no placeholder on this slide with idx==1，查询库的代码，发现是访问具有

idx 占位符的 pptx 页面，但 idx 实际上是一个字典键值，如果集合中没有具有该 idx 值的占位符，则会引发 KeyError。通过查看使用.pptx 文件的幻灯片母版可知，该文件幻灯片母版第一张幻灯片内容为空，如图 3.24 所示。

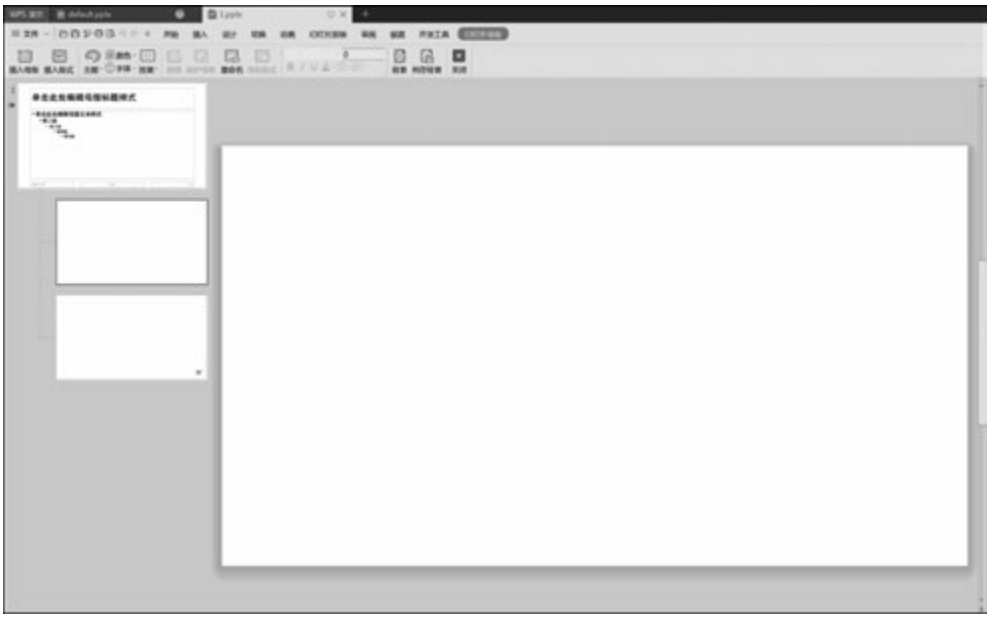


图 3.24　使用.pptx 文件的幻灯片母版

我们来深度解析这段代码，Presentation(pptx=None)函数有一个参数，可以赋值为一个.pptx 文件，表示打开此文件；如果不赋值，则默认按照模板创建一个.pptx 文件，返回.pptx 文件 Presentation 类的对象。该类的主要函数介绍如表 3.17 所示。

表 3.17　Presentation 类的主要函数介绍

函　数	参　数	返回值类型	作　用
core_properties	无	CorePropertiesPart	提供访问该演示文件核心属性的接口
notes_master	无	NotesMaster	提供访问该演示文件母版的接口
save	file：文件名	无	保存文件
slide_height	height：高度	以 Emu 定义的单位表示	幻灯片高度
slide_layouts	无	SlideLayouts	返回演示文件第一个幻灯片母版的布局内容的序列
slide_master	无	SlideMaster	返回演示文件第一个幻灯片母版
slide_width	width：宽度	以 Emu 定义的单位表示	读取或更改幻灯片宽度
slides	无	Slides	返回该演示文件的所有幻灯片

CorePropertiesPart 类对象可以访问的属性包括 author、category、comments、created、identifier、keywords、language、last_modified、title、version 等；NotesMaster 类继承自_BaseMaster 和_BaseSlide，提供了 placeholders、shapes、background、name 共 4 个属性的访问，其中 name 可读可写；SlideLayouts 类提供了对 SlideLayout 对象的访问和操作，可以使用[数字]的形式来访问，也可以迭代访问；还提供了根据布局名称来定位某个布局对象的 get_by_name(self, name, default=None)函数和删除某个布局对象的 remove(self, slide_

layout)函数。

再来看一下幻灯片相关的 Slide 类,其主要函数介绍如表 3.18 所示。

表 3.18 Slide 类的主要函数介绍

函　数	参　数	返回值类型	作　用
background	无	_Background	只读,此属性返回一个_Background 对象,幻灯片是否继承默认背景由 follow_masterbackground 属性的值决定
follow_master_background	无	无	只读,如果 True,则继承默认背景
placeholders	无	SlidePlaceholders	返回包括 SlidePlaceholder 占位符的列表
shapes	无	SlideShapes	返回包括 shape 的序列
slide_id	无	无	返回数字,表示该幻灯片的 id
slide_layout	无	无	得到的是继承的布局相关类的对象 SlideLayout

SlidePlaceholder 继承自_BaseSlidePlaceholder,该类提供了 is_placeholder 和 shape_type 属性,用于判断是否是占位符及形状的类型; SlideShapes 类继承自_BaseGroupShapes 类。SlideShapes 类的主要函数介绍如表 3.19 所示。

表 3.19 SlideShapes 类的主要函数介绍

函　数	参　数	返回值类型	作　用
add_movie(movie_file, left,top,width,height, poster_frame_image= None,mime_type= CT.VIDEO,)	movie_file: 视频 left: 左侧距离 top: 距离上边距离 width: 宽 height: 高 poster_frame_image: 海报图像 mime_type: 视频类型	movie shape	在指定位置添加视频文件,有局限性: 必须指定尺寸; 不提供自动缩放; 应指定视频文件的 MIME 类型,如“video/mp4”; 必须提供海报图像,不能自动提供从视频文件中提取。如果没有提供,将使用默认的“媒体扬声器”图像
add_table(rows,cols,left, top,width, height)	rows: 行数 cols: 列数 left: 左侧距离 top: 距离上边距离 width: 宽 height: 高	GraphicFrame	插入表格
placeholders	无	SlidePlaceholders	该页幻灯片中占位符形状的序列
title	无	无	幻灯片上的标题占位符, SlidePlaceholder 类的对象
add_chart(chart_type,x, y,cx,cy,chart_data)	chart_type: XlChartType 类枚举值 chart_data: ChartData 类的对象,包含了图表的类别和序列值	GraphicFrame	添加图表,图表位于(*x*,*y*),大小为(*cx*,*cy*)
add_connector(self, connector_type, begin_x, begin_y, end_x, end_y)	connector_type: MsoConnectorType 类枚举对象	connector_type	插入连接符

续表

函　数	参　数	返回值类型	作　用
add_picture(image_file, left, top, width = None, height=None)	image_file：图片路径名	Shape	添加图片
add_shape(autoshape_type_id,left, top, width, height)	autoshape_type_id：MsoAutoShapeType 枚举成员	Shape	添加图形
add_textbox(left,top, width,height)	位置信息	Shape	添加文本框
build_freeform(start_x=0,start_y=0,scale=1.0) add_ole_object(object_file,prog_id,left,top, width=None,height=None,icon_file=None, icon_width=None, icon_height=None,)		FreeformBuilder	添加自由形状

SlidePlaceholder 继承自 Shape 类，提供了很多属性，其中最重要的是 text 属性，可读可写，用来设置文本内容。测试代码如下：

```
from pptx import Presentation
prs = Presentation()                                    #使用默认模板
print(prs.core_properties.last_modified_by)             #结果输出 Steve Canny
print(prs.notes_master.name)                            #结果输出 Steve Canny
print(len(prs.slide_layouts))                           #11
#查看幻灯片母版中第一张幻灯片,第一个占位符里的文本
print(prs.slide_layouts[0].placeholders.get(0).text)#Click to edit Master title style
#获得模板文件中第一个幻灯片母版
title_slide_layout = prs.slide_layouts[0]
#二者输出相同,都是指文档的第一个幻灯片母版
print(prs.slide_master.slide_layouts[0])
print(prs.slide_layouts[0])
#获取文档中幻灯片数量
print(len(prs.slides))#0
#添加幻灯片,以第一个幻灯片母版为模板添加
slide = prs.slides.add_slide(title_slide_layout)
print(slide.slide_id)#256
print(slide.placeholders[0].shape_type)                 #PLACEHOLDER(14)
print(slide.shapes)
#获取标题的占位符形状
title = slide.shapes.title
#二者输出相同,都是指文档的第一个幻灯片母版中同一个占位符
print(title)
print(slide.placeholders[0])
#设置标题文本
title.text = "Hello, World!"
print(slide.placeholders[1].shape_type)                 #PLACEHOLDER(14)
#获取该幻灯片的第二个占位符,并设置内容
subtitle = slide.placeholders[1]
```

```
subtitle.text = "python-pptx was here!"
prs.save('start.pptx')
```

2. 创建项目符号幻灯片

本节使用的是幻灯片母版中的第二张幻灯片，内容如图 3.25 所示。

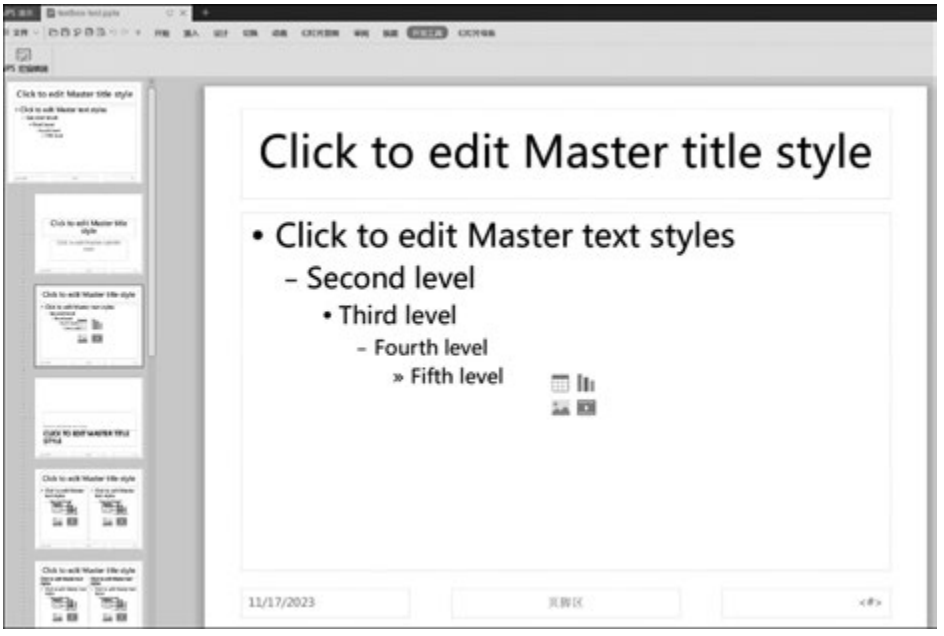


图 3.25　模板中幻灯片母版第二张幻灯片内容

实例代码：

```
from pptx import Presentation
from pptx.enum.text import MSO_AUTO_SIZE,MSO_VERTICAL_ANCHOR
from pptx.dml.color import RGBColor
from pptx.util import Inches
#使用模板创建.pptx文件
prs = Presentation()
#获得幻灯片母版第二张幻灯片布局
bullet_slide_layout = prs.slide_layouts[1]
#以第二张幻灯片为模板创建一张幻灯片
slide = prs.slides.add_slide(bullet_slide_layout)
shapes = slide.shapes
title_shape = shapes.title
body_shape = shapes.placeholders[1]
#查看 body_shape
print(body_shape.is_placeholder)
print(body_shape.shape_type)
print(body_shape.width)#8229600
print(body_shape.height)#4525963
#设置标题占位符内文本内容
title_shape.text = 'Adding a Bullet Slide'
#设置图形占位符中的文本内容和格式
tf = body_shape.text_frame
#设置文本垂直居中
tf.vertical_anchor = MSO_VERTICAL_ANCHOR.MIDDLE
```

```
#tf.fit_text()
print(tf.auto_size)
tf.word_wrap = True#True
print(tf.word_wrap)#
#设置文本距离文本框底边的距离
tf.margin_bottom = Inches(1)
#直接添加文本
tf.text = ('Find the bullet slide layout')
#设置文本自适应文本框,需在生成的.pptx文件中拖动文本框
tf.auto_size = MSO_AUTO_SIZE.TEXT_TO_FIT_SHAPE
#更改文本框大小
body_shape.width = 3600000 * 3
body_shape.height = 3600000
#通过添加段落,添加格式文本
p = tf.add_paragraph()
#设置文字颜色
p.font.color.rgb = RGBColor(255,0,0)
p.text = 'Use _TextFrame.text for first bullet'
p.level = 1
print(p.font.name)                                    #None
print(p.font.color.type)                              #RGB (1)
p2 = tf.add_paragraph()
p2.text = 'Use _TextFrame.add_paragraph() for subsequent bullets'
p2.level = 2
prs.save('text - test.pptx')
```

本节重点研究占位符,SlidePlaceholder 在 pptx/shapes/placeholder.py 中定义,继承自_BaseSlidePlaceholder 类,而_BaseSlidePlaceholder 继承自_InheritsDimensions、Shape 两个类,相关类的主要函数介绍如表 3.20 所示。

表 3.20 SlidePlaceholder 相关类的主要函数介绍

函 数	参 数	返回值类型	作 用
is_placeholder	无	无	只读,如果此形状是一个占位符,则返回 True
shape_type	无	无	只读,MSO_SHAPE_TYPE 类中的枚举类型
height	无	无	此占位符形状的高度;如果没有设置,则其为父布局占位符的高度,以 Emu 定义的单位表示
left	无	无	此占位符形状的左侧距离;如果没有设置,则其为父布局占位符的左侧距离,以 Emu 定义的单位表示
top	无	无	此占位符形状的上方距离;如果没有设置,其为父布局占位符的上方距离,以 Emu 定义的单位表示
width	无	无	此占位符形状的宽度;如果没有设置,其为父布局占位符的宽度,以 Emu 定义的单位表示
auto_shape_type	无	无	只读,自选图形的类型,MSO_SHAPE 类的枚举值,如果不是自选图形,则会报错
fill	无	无	FillFormat 对象,提供对填充属性的访问
has_text_frame	无	无	如果此图形占位符可以包含文本,则返回 True
line	无	无	LineFormat 对象,提供对线条属性的访问
text	无	无	可读可写。读写文本内容
text_frame	无	无	TextFrame 对象,包含文本内容,且提供对文本格式的访问

MSO_SHAPE_TYPE 类在 pptx/enum/shapes.py 中定义；Emu(english metric units)数值 36 000 相当于 1mm；TextFrame 类提供了丰富的文本操作和设置方法，其主要函数介绍如表 3.21 所示。

表 3.21 TextFrame 类的主要函数介绍

函　数	参　数	返回值类型	作　用
add_paragraph	无	_Paragraph	增加文本段落
auto_size	无	无	可读可写。设置文本框的自适应类型，MSO_AUTO_SIZE.NONE 类的枚举成员，也可以为 None，不起作用，需在生成的.pptx 文件中拖动文本框
clear()	无	无	清除文本框中所有段落
fit_text(font_family="Calibri",max_size=18,bold=False,italic=False,font_file=None,)	font_family：字体 max_size：最大字号 bold：加粗 italic：斜体	无	通过设置单词换行并应用“最适合”字体使此文本框中的文本完全包含所有文本
margin_bottom	无	无	文本到文本框底边距
margin_left	无	无	文本到文本框左边距
margin_right	无	无	文本到文本框右边距
margin_top	无	无	文本到文本框上边距
paragraphs	无	无	段落_Paragraph 类对象的序列，用来访问文本框中的所有段落
text	无	无	访问文本框中的文本，包括所有段落
vertical_anchor	无	无	访问文本框中文本的垂直对齐方式，可赋值为 MsoVerticalAnchor 类的枚举成员
word_wrap	无	无	访问自动换行，可设置为 True、False 或 None

在文本框中添加文本，能够使用 TextFrame 中的 text 属性添加，也可以通过添加段落的方式，由_Paragraph 类的函数添加，后者提供了更多对文本操作的方法，其主要函数介绍如表 3.22 所示。

表 3.22 Paragraph 类的主要函数介绍

函　数	参　数	返回值类型	作　用
add_line_break()	无	无	在本段末尾添加换行符
add_run()	无	_Run	在本段中添加一个新的_Run 对象，提供新的文本操作接口
alignment	无	无	访问段落的水平对齐方式，PP_PARAGRAPH_ALIGNMENT 的枚举成员
clear()	无	无	清除段落中所有内容，仅保留段落属性
font	无	无	提供对段落字体访问的接口
level	无	无	访问段落的缩进级别，有 0～8 级，0 为最高级
line_spacing	无	无	访问段落连续行的行间距，用长度单位表示，比如 Pt(12)
runs	无	无	返回此段落中_Run 对象的列表
space_after	无	无	访问此段的段后距离
space_before	无	无	访问此段的段前距离
text	无	无	访问该段落的文本

如果需要对文字的属性进行修改，需要使用_Paragraph 类提供的 font 接口访问 Font 类来实现，Font 类的主要函数介绍如表 3.23 所示。

表 3.23 Font 类的主要函数介绍

函 数	参 数	返回值类型	作 用
bold	无	无	是否加粗，True、False 或 None
color	无	无	提供 ColorFormat 对象，实现对字体颜色访问的接口
fill	无	无	提供 FillFormat 对象，实现对字体填充访问的接口
italic	无	无	是否斜体，True、False 或 None
language_id	无	无	访问该 Font 实例的 language id，该值是 MsoLanguageId 的枚举成员
name	无	无	访问该 Font 实例的字体名称，如果字体从当前主题继承，则返回 None
size	无	无	访问字号，用长度单位表示，比如 Pt(24)
underline	无	无	访问下画线，可以是 True、False 或 None，MsoTextUnderlineType 成员

ColorFormat 在 pptx/dml/color.py 中定义，给出了设置字体颜色 rgb、主体颜色 theme_color 和透明度 brightness 的属性，注意，如果 color.type 值为 None，则读取 rgb 属性时会报错。

3. 文本框操作

对文本框的操作使用的是模板中幻灯片母版的第 7 张幻灯片（后续内容均基于本页幻灯片模板，不再重复说明），具体内容如图 3.26 所示。

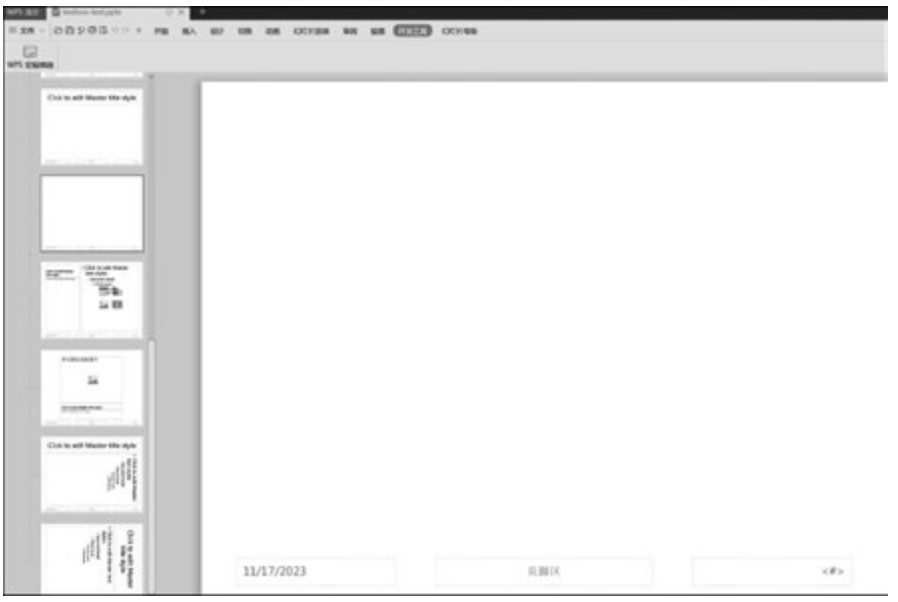


图 3.26 模板中幻灯片母版第 7 张幻灯片内容

实例代码：

```
from pptx import Presentation
from pptx.util import Inches, Cm
```

```
prs = Presentation()
#使用母版的第 7 张幻灯片
blank_slide_layout = prs.slide_layouts[6]
slide = prs.slides.add_slide(blank_slide_layout)
#输出 title,验证选用的幻灯片
print(slide.shapes.title)                              #None
left = top = width = height = Inches(1)
#插入文本框
txBox = slide.shapes.add_textbox(left, top, width, height)
#使用 Shape 的 text_frame 接口操作文本
tf = txBox.text_frame
tf.text = "This is text inside a textbox"
p = tf.add_paragraph()
p.text = "This is a second paragraph that's bold"
p.font.bold = True
p = tf.add_paragraph()
p.text = "This is a third paragraph that's big"
p.font.size = Cm(40)
#查看 Cm 返回值
print(Cm(4))                                           #1440000
prs.save('textbox-test.pptx')
```

首先以母版的第 7 张幻灯片布局为基础，创建一张新的幻灯片，使用幻灯片相关类 Slide 的 shapes 接口访问幻灯片的图形，使用该接口的 add_textbox(left, top, width, height)添加文本框，文本框由 Shape 类定义，使用 Shape 中的函数和属性来对文本框进行操作(Shape 类是 SlidePlaceholder 的基类)，Shape 类的主要函数介绍如表 3.24 所示。

表 3.24 Shape 类的主要函数介绍

函 数	参 数	返回值类型	作 用
auto_shape_type	无	无	只读，自选图形的类型，MSO_SHAPE 类的枚举值，如果不是自选图形，则会报错
fill	无	无	FillFormat 对象，提供对填充属性的访问
has_text_frame	无	无	如果此图形占位符可以包含文本，则返回 True
line	无	无	LineFormat 对象，提供对线条属性的访问
text	无	无	可读可写。读写文本内容
text_frame	无	无	TextFrame 对象，包含文本内容，且提供对文本格式的访问

上面介绍了使用 text_frame 操作文本的方法，下面介绍 Length 类及各种单位的转换关系，Length 类中定义了几种长度单位与 Emu 单位的关系，inch=914 400Emu，centipoint=127Emu，cm=360 000Emu，mm=36 000Emu，pt=12 700Emu，在使用的时候要首先导入库中的对应类，比如使用 cm 度量，执行 from pptx.util import cm 后，在程序中就可以使用 cm 数值，这里的数值是以厘米为单位的，cm 数值会被转换为 Emu 为单位的数据供程序使用。

4. 图片操作

对图片的操作使用的是模板中幻灯片母版的第 7 张幻灯片，实例代码如下：

```
from pptx import Presentation
from pptx.util import Inches, Cm
from pptx.enum.shapes import MSO_SHAPE
```

```
from pptx.dml.color import RGBColor
prs = Presentation()
#使用母版的第7张幻灯片
blank_slide_layout = prs.slide_layouts[6]
slide = prs.slides.add_slide(blank_slide_layout)
#获得幻灯片图片的接口
slide_sh = slide.shapes
#设置图片插入位置
left = top = Inches(1)
#插入图片,原图插入,不修改图片大小
ret_addpic = slide_sh.add_picture("title.png", left, top, width=None, height=None)
slide_sh.add_picture("title.png", Inches(5), Inches(1))
#插入图片,修改图片大小
slide_sh.add_picture("title.png",left, top, width=Cm(1), height=Cm(1))
#图片相关操作
print(ret_addpic.auto_shape_type)
#使用十边形进行覆盖塑形
ret_addpic.auto_shape_type = MSO_SHAPE.DECAGON
#从底部上下压缩
ret_addpic.crop_bottom = -0.5
#访问图像的属性
print(ret_addpic.image.blob)#b'\x89PNG\r\n\x1...
print(ret_addpic.image.content_type)      #image/png
print(ret_addpic.image.dpi)               #(72, 72)
print(ret_addpic.image.ext)               #png
print(ret_addpic.image.filename)          #title.png
print(ret_addpic.image.sha1)              #c693ec2ba5fa8ea5ab6fd100080b69079ace37b1
print(ret_addpic.image.size)              #(240, 234)
#设置图片的边框
print(ret_addpic.line.dash_style)#
#更改边框颜色
ret_addpic.line.color.rgb = RGBColor(255,0,0)
#更改边框线的宽度
ret_addpic.line.width = Cm(0.5)
#设置边框线为渐变色
ret_addpic.line.fill.gradient()
#设置渐变色的停止颜色
ret_addpic.line.fill.gradient_stops[0].color.rgb = RGBColor(255,0,0)
print(ret_addpic.line.fill.gradient_stops[0].color)
prs.save('picture-test.pptx')
```

对图片的操作,尤其要注意.pptx文件的一个特点,那就是.pptx文件中的元素是可以叠加的,因此在插入图片时要考虑好布局。代码执行后的结果如图3.27所示。

对插入图片的操作主要是Picture类,该类继承自_BasePicture类,在pptx/shapes/picture.py中定义,其主要函数介绍如表3.25所示。

表3.25 Picture类的主要函数介绍

函 数	参 数	返回值类型	作 用
auto_shape_type	无	无	可读写。使用MSO_AUTO_SHAPE_TYPE的枚举成员定义的图片形状进行塑形,将该形状外的图片裁剪为不可见
image	无	Image	可读。提供访问图片属性和字节内容的接口

续表

函　数	参　数	返回值类型	作　用
crop_bottom	无	无	将图片从底部裁剪去的比例，如果是 1，图片将不显示，如果值大于 0 且小于 1，裁剪掉对应比例后，图片将被拉伸；如果为负值，图片将被上下压缩变形，但是图片的外框形状不变
crop_left	无	无	将图片从左侧裁剪去的比例，如果是 1，图片将不显示，如果值大于 0 且小于 1，裁剪掉对应比例后，图片将被拉伸；如果为负值，图片将被左右压缩变形，但是图片的外框形状不变
crop_right	无	无	将图片从右侧裁剪去的比例，如果是 1，图片将不显示，如果值大于 0 且小于 1，裁剪掉对应比例后，图片将被拉伸；如果为负值，图片将被左右压缩变形，但是图片的外框形状不变
crop_top	无	无	将图片从上方裁剪去的比例，如果是 1，图片将不显示，如果值大于 0 且小于 1，裁剪掉对应比例后，图片将被拉伸；如果为负值，图片将被上下压缩变形，但是图片的外框形状不变
line	无	LineFormat	可读。提供访问图片外框线条的接口

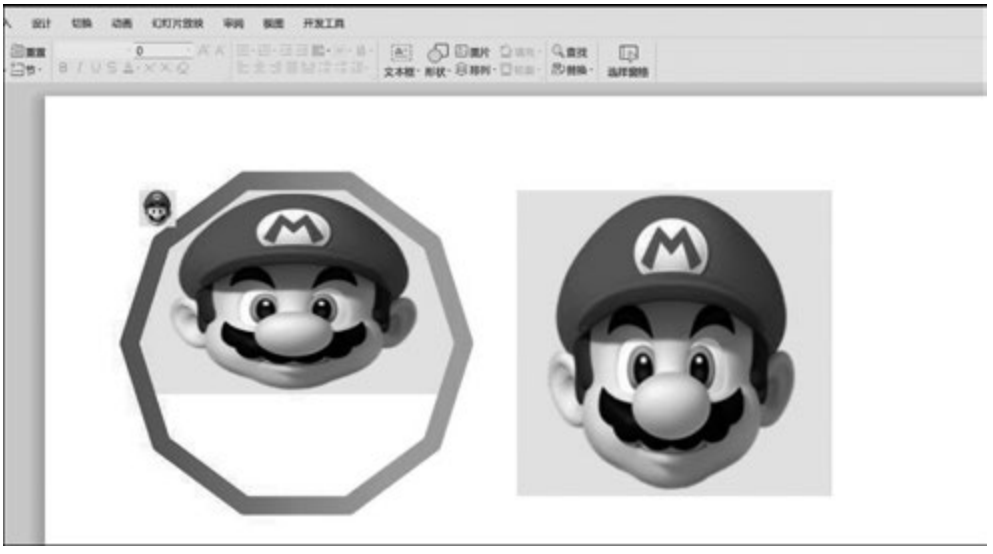


图 3.27　图片操作结果

Image 类在 pptx/parts/image.py 中定义，其主要函数介绍如表 3.26 所示。

表 3.26　Image 类的主要函数介绍

函　数	参　数	返回值类型	作　用
blob	无	无	图片的二进制字节流
content_type	无	无	文件类型
dpi	无	无	每英寸面积内的像素点数
ext	无	无	图片文件的扩展名
filename	无	无	文件名
sha1	无	无	SHA-1 算法的哈希值
size	无	无	图片的宽和高

LineFormat 类在 pptx/dml/line.py 中定义，其主要函数介绍如表 3.27 所示。

表 3.27 LineFormat 类的主要函数介绍

函 数	参 数	返回值类型	作 用
color	无	ColorFormat	提供对边框颜色访问的接口
dash_style	无	无	显示边框样式，MsoLineDashStyle 类的枚举成员，如果未设置，则返回 None
fill	无	FillFormat	提供访问边框填充属性的接口
width	无	无	边框线的宽度

注意：想要查看某个类由哪个文件定义，可以将该类的对象打印出来，比如，

```
print(ret_addpic.line.fill.gradient_stops[0].color)
```

结果是：

```
<pptx.dml.color.ColorFormat object at 0x787b67eb60>
```

它表示 color 这个对象是 pptx/dml/color.py 文件的 ColorFormat 类对象。

ColorFormat 类主要提供了颜色的亮度属性 brightness 和颜色属性 rgb；FillFormat 主要提供了填充对象的背景色、前景色、填充类型和渐变色等的设置。

5. 连接符操作

对连接符的操作较为简单，主要涉及连接符的类型，起始和终点位置等。

实例代码：

```
from pptx import Presentation
from pptx.util import Inches, Cm
from pptx.enum.shapes import MSO_CONNECTOR,MSO_SHAPE   #导入类的别名会提示错误,实际不是错
                                                       #误,不影响使用
from pptx.dml.color import RGBColor
prs = Presentation()
#使用母版的第 7 张幻灯片
blank_slide_layout = prs.slide_layouts[6]
slide = prs.slides.add_slide(blank_slide_layout)
#获得幻灯片图片的接口
slide_sh = slide.shapes
#设置连接符的起点和终点位置
begin_x = begin_y = Inches(1)
end_x = end_y = Inches(5)
#添加连接符,必须设置起点和终点
conn = slide_sh.add_connector(MSO_CONNECTOR.STRAIGHT,begin_x, begin_y, end_x, end_y)
print(conn)
#添加两个图片
shape_1 = slide_sh.add_shape(MSO_SHAPE.RECTANGLE,Inches(0.1),Inches(0.1),Inches(1),Inches(1))
shape_2 = slide_sh.add_shape(MSO_SHAPE.RECTANGLE,Inches(5),Inches(5),Inches(1),Inches(1))
print(shape_2)
conn_1 = slide_sh.add_connector(MSO_CONNECTOR.CURVE,begin_x, begin_y, end_x, end_y)
#使用图片的指定点进行连接
conn_1.begin_connect(shape_1, 3)
conn_1.end_connect(shape_2,0)
#打印 connector 类的相关属性
print(conn.shape_type)#LINE (9)
print(conn.line)
```

```
print(conn.begin_x)
prs.save('conn-test.pptx')
```

操作.pptx文件的连接符用到的是Connector类,其主要函数介绍如表3.28所示。

表3.28 Connector类的主要函数介绍

函　数	参　数	返回值类型	作　用
begin_connect(shape, cxn_pt_idx)	shape：Shape类的对象 cxn_pt_idx：每个形状都有零个或多个连接点,它们由索引标识,从0开始。通常,形状的第一个连接点位于其边界框的顶部中心,编号从那里逆时针进行	无	设置连接符的起点,只对矩形或者图片有效果,其他形状效果不好,可能出现连接位置不准确
begin_x	无	无	起点X坐标
begin_y	无	无	起点Y坐标
end_connect(shape, cxn_pt_idx)	同begin_connect(shape,cxn_pt_idx)	无	设置连接符的终点,只对矩形或者图片有效果,其他形状效果不好,可能出现连接位置不准确的问题
end_x	无	无	终点X坐标
end_y	无	无	终点Y坐标
line	无	LineFormat	提供对连接符线特性(如线颜色、宽度和线样式)的访问接口

关于LineFormat的介绍参考图片操作中对该类的介绍。

6. 表格操作

表格是PPT展示数据的一种重要工具,在幻灯片中插入表格,使用的是SlideShapes的add_table(rows, cols, left, top, width, height)函数,其中的参数依次定义了行数、列数、距离幻灯片左上角的位置、表格的宽度和高度,其返回的并不是表格的访问接口,而是表示形状容器类GraphicFrame的对象,它可能包含chart、table、smart art、media对象。本节实例代码如下:

```
from pptx import Presentation
from pptx.util import Inches, Cm
from pptx.enum.shapes import MSO_CONNECTOR,MSO_SHAPE
#导入类的别名会提示错误,实际不是错误,不影响使用
from pptx.dml.color import RGBColor
prs = Presentation()
#使用母版的第7张幻灯片
blank_slide_layout = prs.slide_layouts[6]
slide = prs.slides.add_slide(blank_slide_layout)
#获得幻灯片图片的接口
slide_sh = slide.shapes
print(slide_sh)                #<pptx.shapes.shapetree.SlideShapes object at 0x7e0efb9f00>
#指定表格左上角的位置
left = top = width = height = Inches(1)
#指定表格行数和列数,rows为行数,cols为列数
rows = 3
cols = 5
t_table = slide_sh.add_table(rows, cols, left, top, width, height)
print(t_table)                 #<pptx.shapes.graphfrm.GraphicFrame object at 0x77d82384c0>
```

```
#关于 GraphicFrame 类属性演示
print(t_table.shape_type)                    #TABLE (19)
print(t_table.has_table)                     #True
print(t_table.has_chart)                     #False
#取得 Table 类定义对象的访问接口
my_table = t_table.table
print(my_table)                              #<pptx.table.Table object at 0x7a2f8466e0>
#Table 类属性演示
print(my_table.first_col)                    #False
print(my_table.first_row)                    #True
print(my_table.last_col)                     #False
print(my_table.last_row)                     #False
print(my_table.vert_banding)                 #False
print(my_table.horz_banding)                 #True
#改变表格的列的宽度和行的高度,改变前后的值通过_graphic_frame.height 和_graphic_frame
#.width 获得
print(my_table.notify_height_changed())      #None,只有行列变化后有用
print(my_table.notify_width_changed())       #None
print(my_table._graphic_frame.height)        #914400
row_1 = my_table.rows[0]
row_1.height = Inches(2)
print(my_table._graphic_frame.height)        #2438400
#_Cell 类属性演示
print(my_table.cell(0,0))                    #<pptx.table._Cell object at 0x72fb78f040
print(my_table.cell(0,0).is_merge_origin)    #False
print(my_table.cell(0,0).is_spanned)         #False
print(my_table.cell(0,0).margin_left)        #91440
print(my_table.cell(0,0).margin_bottom)      #45720
#合并单元格
print(my_table.cell(0,0).span_height)        #1,合并前跨的行数
my_table.cell(0,0).merge(my_table.cell(1,1))
print(my_table.cell(0,0).span_height)        #2,合并后跨的行数
#取消合并单元格
my_table.cell(0,0).split()
#访问单元格中的文本对象
print(my_table.cell(0,0).text_frame)   #<pptx.text.text.TextFrame object at 0x7111a33850>
#访问列
print(my_table.columns[1].width)             #182880
#访问行,访问第一行的第一个单元格
print(my_table.rows[0].height)               #1828800
print(my_table.rows[0].cells[0])             #<pptx.table._Cell object at 0x79c959fe50>
prs.save('table-test.pptx')
```

GraphicFrame 类提供的主要函数介绍如表 3.29 所示。

表 3.29 GraphicFrame 类的主要函数介绍

函 数	参 数	返回值类型	作 用
chart	无	Chart	提供访问 Chart 对象的接口,如果没有 Chart 对象,会报错
chart_part		ChartPart	提供访问 ChartPart 对象的接口,如果没有 ChartPart 对象,则会报错
has_chart	无	无	如果有 chart 对象,则返回 True,否则返回 False
has_table	无	无	如果有 table 对象,则返回 True,否则返回 False

续表

函　数	参　数	返回值类型	作　用
ole_format	无	_OleFormat	如果有 OLE 对象，返回该对象，如果无，则报错
shape_type	无	无	返回 MSO_SHAPE_TYPE 的枚举成员，用于标识此形状的类型，可能是 MSO_SHAPE_TYPE. CHART、MSO_SHAPE_TYPE. TABLE、MSO_SHAPE_TYPE. EMBEDDED_OLE_OBJECT、MSO_SHAPE_TYPE. LINKED_OLE_OBJECT 中的一个，如果类型为 SmartArt，则返回 None
table	无	Table	提供访问 Table 对象的接口

具体的表格操作，有 Table 类定义，该类的主要函数介绍如表 3.30 所示。

表 3.30　Table 类的主要函数介绍

函　数	参　数	返回值类型	作　用
cell(row_idx, col_idx)	row_idx：行序号 col_idx：列序号	_Cell	返回单元格，提供对该单元格访问的接口
columns	无	_ColumnCollection	值为_Column 对象的列表，可以通过其访问所有表格中的列
first_col	无	无	可读可写，当为 True 时，表示第一列具有不同格式，比如表最左边的侧标题列
first_row	无	无	可读可写，当为 True 时，表示第一行具有不同格式，比如列标题
horz_banding	无	无	可读可写，当为 True 时，指示表中的行应交替着色
last_col	无	无	可读可写，当为 True 时，表示最后一列有不同格式，比如最右侧的合计列
last_row	无	无	可读可写，当为 True 时，表示最后一行有不同格式，比如最下方的合计行
notify_height_changed()	无	无	当行的高度改变时，可以通过该函数计算表格的总高度，不需要调用，要获得行的总高度可以用：._graphic_frame.height
notify_width_changed()	无	无	当列的宽度改变时，可以通过该函数计算表格的总宽度，不需要调用，要获得行的总高度可以用：._graphic_frame.width
rows	无	_RowCollection	值为_Row 对象的列表，可以通过其访问所有表格中的行
vert_banding	无	无	可读可写，当为 True 时，指示表中的列应交替着色

注意：在查看库的代码时，如果碰到类继承自另一个类，可以通过继承类的名称，通过对文件开始的导入代码进行分析查看，比如，

```
from pptx.shapes import Subshape
⋮
class _RowCollection(Subshape):
```

Subshape 类来自文件 shapes.py 或者 shapes 文件夹下的__init__.py 文件。

对表格的操作，最终是对单元格、行和列的操作，其中_Cell类的主要函数介绍如表3.31所示。

表3.31 _Cell类的主要函数介绍

函　数	参　数	返回值类型	作　用
fill	无	FillFormat	提供对单元格填充属性的访问接口
is_merge_origin	无	无	判断是不是左上角的单元格，且是合并单元格状态，如果是则返回True
is_spanned	无	无	判断此单元格是不是属于合并单元格范围，如果是则返回True
margin_left	无	无	左边距
margin_right	无	无	右边距
margin_top	无	无	上边距
margin_bottom	无	无	下边距
merge(other_cell)	other_cell（另一个单元格）	无	合并单元格，范围是由本单元格到指定的另一个单元格之间
span_height	无	无	单元格跨的行数，如果单元格没有合并则为1
span_width	无	无	单元格跨的列数，如果单元格没有合并则为1
split()	无	无	取消合并单元格，如果单元格不是合并的单元格，则报错
text	无	无	可读可写，访问单元格内的文本
text_frame	无	TextFrame	可读，返回操作单元格文本的对象，能对文本进行更多的操作，参考表3.21
vertical_anchor	无	无	访问单元格垂直对齐，值可以是MSO_VERTICAL_ANCHOR的枚举成员或者None

MSO_VERTICAL_ANCHOR类在文件pptx/enum/text.py中定义，主要有3种对齐方式：TOP、MIDDLE、BOTTOM；_Column类仅仅提供了width属性，可以读取和设置列的宽度；_Row类提供了height属性，可以读取和设置行的高度；提供了cells只读属性，表示该行中的单元格集合，可以像访问列表一样访问该行中的单元格。

7. 图表操作（柱状图、条形图、折线图和饼图）

实例代码：

```
from pptx import Presentation
from pptx.util import Inches, Cm
from pptx.enum.chart import XL_CHART_TYPE,XL_LEGEND_POSITION
from pptx.chart.data import CategoryChartData
prs = Presentation()
#使用母版的第7张幻灯片
blank_slide_layout = prs.slide_layouts[6]
slide = prs.slides.add_slide(blank_slide_layout)
#获得幻灯片图表的接口
slide_sh = slide.shapes
print(slide_sh) #<pptx.shapes.shapetree.SlideShapes object at 0x7e0efb9f00>
#图表操作
#选择图表的类型
chart_type = XL_CHART_TYPE.LINE
x = y = Inches(1)
```

```
cx = cy = Inches(5)
#定义图表数据
chart_data = CategoryChartData()
chart_data.categories = ["1","2","3","4"]
chart_data.add_series("销售额",(50,100,150,100,9,80))   #如果数据数量超过了类别,即使没
                                                        #有类别也会显示
chart_data.add_series("利润",(12,25,40,66,90,60))
#添加类别
chart_data.add_category("5")     #如果新添加类别没有对应的数据,则不显示
#添加序列数据
print(chart_data.add_series("我的",(2,5,4,6,0,6)))#<pptx.chart.data.CategorySeriesData
                                                  #object at 0x70b6d23b50>
#打印序列的类别列表
print(chart_data.categories.leaf_count)   #5
#数据的引用
print(chart_data.categories_ref)          #Sheet1!$A$2:$A$6
#插入图表
my_chart = slide_sh.add_chart(chart_type,x,y,cx,cy,chart_data)
#输出插入图表的返回值
print(my_chart)               #<pptx.shapes.graphfrm.GraphicFrame object at 0x7aacaa9060>
#访问插入的图表
if my_chart.has_chart == True:
    print(my_chart.chart)          #<pptx.chart.chart.Chart object at 0x7e9d395900>
    real_chart = my_chart.chart
    #设置类别坐标轴的名称
    real_chart.category_axis.axis_title.text_frame.text = "月份"
    print(real_chart.category_axis.axis_title.text_frame.text)
    #选择图表样式
    real_chart.chart_style = 2
    #设置图表标题
    real_chart.chart_title.text_frame.text = "统计销售情况"
    #打印图表类型
    print(real_chart.chart_type)    #LINE (4)
    #图表的字体大小
    print(real_chart.font.size)     #228600
    print(real_chart.has_legend)    #True,有图例
    print(real_chart.has_title)     #True,有标题

    #real_chart.has_title = False
    #real_chart.has_legend = False
    #访问图例,改变图例的位置
    print(real_chart.legend)
    real_chart.legend.position = XL_LEGEND_POSITION.LEFT
    #访问图表中每一个绘制的图形
    print(real_chart.plots[0])      #<pptx.chart.plot.LinePlot object at 0x7e1b3ecf70>
    #设置图表中第一条线为平滑曲线
    print(real_chart.series[0])     #<pptx.chart.series.LineSeries object at 0x7d294e9e70>
    real_chart.series[0].smooth = True
    print(real_chart.value_axis)    #<pptx.chart.axis.ValueAxis object at 0x7725c99f00>
    print(real_chart.value_axis.crosses)     #AUTOMATIC (-4105)
    #设置数据轴的名称
    real_chart.value_axis.axis_title.text_frame.text = "金额/万元"
prs.save('chart-test.pptx')
```

图表是 PPT 演示文稿的重要组成部分,插入图表使用 Slideshapes 的 add_chart(self,

chart_type，x，y，cx，cy，chart_data)(该函数在 Slideshapes 类的父类中定义)，其中，chart_type 指定了图表的类型，该值是 pptx/enum/chart.py 中定义的 XL_CHART_TYPE 类的枚举成员，该类中定义的图表类型很多，但是有一些图表类型在 python-pptx 类库中并没有实现，调用该类图表会报错，常见的图表类型及对应值如表 3.32 所示。

表 3.32 图表类型及对应值

图表类型	值
面积图	AREA
堆积面积图	AREA_STACKED
百分比堆积面积图	AREA_STACKED_100
簇状条形图	BAR_CLUSTERED
堆积条形图	BAR_STACKED
百分比堆积条形图	BAR_STACKED_100
簇状柱形图	COLUMN_CLUSTERED
堆积柱形图	COLUMN_STACKED
百分比堆积柱形图	COLUMN_STACKED_100
圆环图	DOUGHNUT
分解圆环图	DOUGHNUT_EXPLODED
折线图	LINE
带数据标记的折线图	LINE_MARKERS
带数据标记的堆积折线图	LINE_MARKERS_STACKED
带数据标记的百分比堆积折线图	LINE_MARKERS_STACKED_100
堆积折线图	LINE_STACKED
百分比堆积折线图	LINE_STACKED_100
饼图	PIE
分解饼图	PIE_EXPLODED
雷达图	RADAR
填充雷达图	RADAR_FILLED
带数据标记的雷达图	RADAR_MARKERS

散点图系列会报错：'CategoryWorkbookWriter' object has no attribute 'x_values_ref'，如果某个类型的图表没有实现则会报错：XML writer for chart type XXX not yet implemented；创建图表不但要确定图表类型，还要确定图表数据，图表数据由 CategoryChartData 类定义。该类继承自_BaseChartData，其主要函数和属性如表 3.33 所示。

表 3.33 CategoryChartData 类介绍

函数/属性	参数	返回值类型	作用
add_category(label)	lable：类别	Category	将新的类别添加到类别集合中
add_series(name,values=(),number_format=None)	name：数据序列的名称 values：数据的集合	CategorySeriesData	添加一组名称为 name 的序列数据
categories	无	Categories	访问类别数据的列表
categories_ref	无	无	Excel 工作表可引用此图表的类别(不包括列标题)

续表

函数/属性	参　数	返回值类型	作　用
values_ref	无	无	Excel 工作表可引用图表数据系列的值(不包括列标题)

通过 Chart 类可以对插入的图表进行设置和修改,其主要函数和属性如表 3.34 所示。

表 3.34　Chart 类介绍

函数/属性	参　数	返回值类型	作　用
category_axis	无	CategoryAxis	提供访问表示类别的坐标轴的接口,如果没有定义,报错,比如饼图
chart_style	无	无	可读可写。指定此图表的图表样式的整数索引,值为 1～48。如果未指定显式样式,则值为 None,在这种情况下,将使用默认图表样式。整数索引对应于 PowerPoint UI 中图表样式库中样式的位置
chart_title	无	ChartTitle	提供访问标题属性的接口
chart_type	无	XlChartType	详见表 3.32
font	无	Font	访问此图表的默认文本格式的字体对象
has_legend	无	无	图表是否有图例,如果设为 False,则删除图例
has_title	无	无	图表是否有标题,如果设为 False,则删除标题
legend	无	Legend	提供访问图例属性的接口
plots	无	_Plots	提供访问图表中绘制的每一个图形的接口,这些图形的形状和图表类型有关
replace_data(chart_data)	chart_data:CategoryChartData 类的对象,包含图表的类别和数据	无	替换图表中的类别和数据
series	无	SeriesCollection	返回图表的所有数据序列,并按照绘图顺序存放
value_axis	无	ValueAxis	返回访问连续值轴的接口

CategoryAxis 类在 pptx/chart/axis.py 中定义,继承自_BaseAxis 类,主要是包括对图表类别轴的操作和设置;Legend 类 pptx/chart/legend.py 中定义,主要包括设置图例位置、字体等操作;SeriesCollection 中根据图表类型的不同会存储不同类定义的数据,比如 LineSeries、BarSeries 等,不同类的数据可以执行不同的操作,具体可以参考相关类的定义(参考实例代码);ValueAxis 同样在 pptx/chart/axis.py 中定义,继承自_BaseAxis 类,主要包括对数据轴的操作和设置。

8. 给形状添加响应功能

给形状添加响应的功能是由在 pptx/shapes/base.py 文件中定义的 BaseShape 类提供的属性接口 click_action 完成的,该接口返回的对象属于 ActionSetting 类,该类主要定义了指定形状以响应鼠标操作的具体属性。可以设置响应的形状均继承自 BaseShape,具体包括 Shape 类、Picture 类以及图表、表格(二者父类均为 GraphicFrame 类)。

实例代码：

```
from pptx import Presentation
from pptx.util import Inches, Cm
from pptx.enum.shapes import MSO_CONNECTOR,MSO_SHAPE
#导入类的别名会提示错误,实际不是错误,不影响使用
prs = Presentation()
#使用母版的第 7 张幻灯片
blank_slide_layout = prs.slide_layouts[6]
slide = prs.slides.add_slide(blank_slide_layout)
slide2 = prs.slides.add_slide(blank_slide_layout)
#获得幻灯片图片的接口
slide_sh = slide.shapes
#添加形状
button = slide_sh.add_shape(MSO_SHAPE.RECTANGLE,Inches(1),Inches(1),Inches(1),Inches(1))
print(button.click_action)              #<pptx.action.ActionSetting object at 0x73aa7d6a10>
#设置超链接
button.click_action.hyperlink.address = "http://www.baidu.com"
hyper = button.click_action.action
print(hyper)                                        #HYPERLINK (7)
#设置跳转
button.click_action.target_slide = slide2
hyper = button.click_action.action
print(hyper)                                        #NAMED_SLIDE (101)
prs.save('action-test.pptx')
```

关于 ActionSetting 类的主要函数和属性如表 3.35 所示。

表 3.35 ActionSetting 类介绍

函数/属性	参 数	返回值类型	作 用
action	无	无	只读。返回的值为 PpActionType 类的枚举成员，表明在幻灯片放映期间单击指定的形状或文本，或者将鼠标指针放置在该形状上时将产生的操作类型
hyperlink	无	Hyperlink	提供设置超链接的访问接口
target_slide	无	无	实现跳转到指定幻灯片

Hyperlink 类提供了 address 属性，可以设置超链接、http/https 网站、文件路径，在幻灯片播放时实现跳转。

3.5 本章小结

WPS 是主要的国产办公自动化软件。本章主要介绍了 WPS 表格的基本知识，包括宏录制的方法、宏编辑器编程的知识以及使用 Python 操作表格的库及其方法；介绍了 WPS 的 Word 文件的基础知识，并详细讲解了如何使用 Python 中的 python-docx 库来操作 Word 文件的各种元素，此外，还介绍了如何利用 python-docx-template 库基于模板进行文档处理；全面介绍了 WPS 演示文稿的基础知识，并深入讲解如何利用 Python 中的 python-pptx 库来操作演示文稿的各种元素。通过学习本章内容，可掌握基本的自动化操作技能，从而利用 WPS 实现办公自动化。