# 第 3 章　Word 操作自动化

在这个日新月异的信息化时代，Microsoft Word 已经深深地渗透到人们的工作和生活中。无论是编写报告、起草合同、创作书籍，还是撰写论文，Word 都是首选工具。然而，随着待处理文档数量和复杂性的增加，手动操作 Word 文档变得越来越烦琐和效率低下。为了解决这些问题，Python 自动化办公的概念应运而生。Python 是一种强大的编程语言，其简洁的语法和丰富的库使其成为自动化办公的理想工具，特别是在处理 Word 文档时，Python-docx 库发挥了重要的作用。

本章主要包括多个实战案例，通过丰富多样的实战案例，读者将掌握以下关键技能：

- Python-docx 库的基本知识和操作。
- 通过 Python 自动创建报告、合同、目录和索引。
- 批量处理 Word 文档，包括内容替换、文档合并、批量转换为 PDF、修改文档属性、提取文本和图片、添加或修改页眉页脚等。
- 在 Word 文档中插入图片、表格、超链接及书签。
- 自动将 Excel 或 CSV 数据导入 Word 文档生成表格。
- 自动生成批注、脚注、表格目录和图表目录。
- 自动生成多级标题及其编号。
- Word 文档的加密和解密。

本章将详细介绍如何使用 Python-docx 库来处理 Word 文档。

## 3.1　Word 操作自动化概述

Microsoft Word 作为全球最广泛使用的文档处理工具，提供了丰富的功能用于文档的创建、编辑和格式化等。然而，随着需要处理的文档数量和复杂性的增加，手动操作 Word 文档变得既耗时又低效，这就需要用自动化的方式来解决。通过编程语言，特别是 Python，可以批量处理文档，进行内容替换，统一格式，甚至生成报告和合同。

## 3.2　Python-docx 库简介

Python-docx 是一个开源的 Python 库，用于创建、修改 Microsoft Word (.docx) 文件，

其标志如图 3.1 所示。Python-docx 提供了大量的接口用以进行文档操作，例如处理段落、文本、样式、图片、表格、页眉、页脚，以及 Word 文档中的其他元素。

图 3.1　Python-docx 标志

在使用 Python-docx 之前，需要先将其安装到 Python 环境中。这一操作可以通过 pip 命令轻松完成，如代码 3-1 所示。

代码 3-1　用pip命令安装Python-docx

```
pip install python-docx
```

如果使用的是 Anaconda 环境，也可以通过 conda 命令安装，如代码 3-2 所示。

代码 3-2　用conda命令安装Python-docx

```
conda install -c conda-forge python-docx
```

## 3.3　Python-docx 库的基本操作

有了 Python-docx 库，就可以对 Word 文档进行操作了。Python-docx 的基本操作主要包括创建新文档、添加段落、保存文档、打开现有文档、读取并打印所有段落等，如代码 3-3 所示。

代码 3-3　Python-docx的基本操作

```
from docx import Document

# 创建新的 Word 文档
doc = Document()

# 添加段落
p = doc.add_paragraph('Hello, World!')

# 保存文档
doc.save('demo.docx')

# 打开现有文档
```

```
doc = Document('demo.docx')

# 读取并打印所有段落
for p in doc.paragraphs:
    print(p.text)
```

以上是 Python-docx 库的一些基本操作，通过这些操作可以创建、打开、编辑和保存 Word 文档。

接下来的实战案例将全面展示 Python-docx 的各项功能，通过案例学习如何使用 Python-docx 自动处理 Word 文档。在实战案例中将解释所用到的 Python-docx 接口，提供详细的代码，并解读代码的工作原理。

## 3.4　实战案例 1：批量提取 Word 文档中的文本

在处理 Word 文档时，经常需要从文档中提取文本内容，以分析或用于其他处理过程。Python-docx 库提供了简单而直接的接口，可以轻松实现从文档中读取文本的功能。可以将任何一个文档视为一系列的段落，每个段落都包含一些文本。Python-docx 则提供了一个结构化的方式来处理 Word 文档。

为了实现批量提取文档的功能，首先需要读取文档，然后遍历其中的所有段落，把每个段落的内容保存起来。在 Python-docx 中，可以使用 Document 类表示一个 Word 文档，通过 Document 的 paragraphs 属性，可以获取文档中的所有段落，如代码 3-4 所示。

**代码 3-4　批量提取Word文档中的文本**

```
from docx import Document

def extract_specific_text(filename, specific_text):
    """
    从指定的 Word 文档中提取包含特定文本的段落

    参数:
    filename -- Word 文档的文件名
    specific_text -- 特定的文本

    返回:
    包含特定文本的段落列表
    """
    doc = Document(filename)
    return [para.text for para in doc.paragraphs if specific_text in
para.text]

# 使用 extract_specific_text()函数，从 example.docx 文档中提取出包含"Python"
# 的段落
text = extract_specific_text('example.docx', 'Python')
for line in text:
    print(line)
```

上述代码定义了一个函数 extract_specific_text()，它接收两个参数：一个是 Word 文档的文件名，另一个是要提取的特定文本。这个函数遍历文档中的每个段落，检查段落

的文本是否包含特定文本。只有当段落的文本包含特定文本时，该段落的文本才被添加到结果列表中。

该函数的返回值是一个字符串列表，每个字符串对应文档中包含特定文本的一个段落。这样用户就能专注地处理那些包含特定文本的段落。

这个实战案例展示了如何从 Word 文档中提取包含特定文本的段落。这是一个常见的任务，例如，想要从一个大的文档中提取所有提到某个关键词的段落，通过 Python-docx 库，便可以用简洁的代码实现这个任务。

## 3.5　实战案例 2：Word 文档内容替换

在处理 Word 文档时，经常需要对文档中的一些文本进行替换，例如更改产品名称、更新公司名称或调整某些术语。Python-docx 库能够方便地完成这种替换操作。本实战案例将学习如何使用 Python-docx 库实现文档内容的替换功能。

实现替换功能的基本思路是遍历文档中的所有段落，对每个段落的文本进行检查，找到需要替换的内容并进行替换。为了更有效地实现这个功能，可以将文本替换的操作封装为一个函数，这样便可以轻松地对不同的文档和文本进行替换，示例如代码 3-5 所示。

**代码 3-5　Word文档内容替换**

```
from docx import Document

def replace_text_in_document(filename, old_text, new_text):
    """
    在 Word 文档中替换文本

    参数:
    filename -- Word 文档的文件名
    old_text -- 需要替换的文本
    new_text -- 替换后的文本
    """
    doc = Document(filename)
    for para in doc.paragraphs:
        if old_text in para.text:
            para.text = para.text.replace(old_text, new_text)
    doc.save('updated_' + filename)

# 使用 replace_text_in_document()函数,将 example.docx 文档中的"Python"替换为
# "Python3"
replace_text_in_document('example.docx', 'Python', 'Python3')
```

上述代码定义了一个函数 replace_text_in_document()，它接收 3 个参数，即 Word 文档的文件名、需要替换的文本和替换后的文本。这个函数遍历文档的每个段落，检查段落的文本是否包含需要替换的内容。如果包含，则使用 para.text.replace()方法进行替换。最后，将替换后的文档保存为一个新文件。

这个实战案例展示了如何使用 Python-docx 库替换 Word 文档中的文本。这是一个非常实用的功能，可以批量修改文档，提高工作效率。Python-docx 库可以轻松实现这个任务，无须手动操作每个文档。

## 3.6　实战案例 3：自动化创建和更新书签

在处理 Word 文档时，书签是一个非常有用的功能，可以帮助用户快速定位和跳转到文档中的特定位置。Python-docx 库可以实现自动化创建和更新书签的功能，从而更方便地管理和操作 Word 文档。

本实战案例将学习如何使用 Python-docx 库实现自动化创建和更新书签的功能。首先定义一个函数来创建书签，然后创建另一个函数来更新书签指向的内容，如代码 3-6 所示。

**代码 3-6　自动化创建和更新书签**

```
from docx import Document
from docx.oxml.ns import qn
from docx.oxml import OxmlElement

def create_bookmark(paragraph, bookmark_name):
    """
    在指定的段落中创建书签

    参数:
    paragraph -- 段落对象
    bookmark_name -- 书签名称
    """
    run = paragraph.add_run()
    start = OxmlElement('w:bookmarkStart')
    start.set(qn('w:id'), '0')
    start.set(qn('w:name'), bookmark_name)
    run._r.append(start)
    end = OxmlElement('w:bookmarkEnd')
    end.set(qn('w:id'), '0')
    run._r.append(end)

def update_bookmark(document, bookmark_name, new_text):
    """
    更新指定书签的内容

    参数:
    document -- Document 对象
    bookmark_name -- 书签名称
    new_text -- 新的内容
    """
    for paragraph in document.paragraphs:
        if any(elem.get(qn('w:name')) == bookmark_name for elem in
paragraph._element.findall('.//' + qn('w:bookmarkStart'))):
            paragraph.clear()
            paragraph.add_run(new_text)
```

```
            create_bookmark(paragraph, bookmark_name)
            break

# 创建一个新的 Word 文档并添加一个段落
doc = Document()
para = doc.add_paragraph('This is a test paragraph.')

# 在段落中创建一个书签
create_bookmark(para, 'test_bookmark')

# 保存文档
doc.save('bookmarked.docx')

# 打开文档并更新书签的内容
doc = Document('bookmarked.docx')
update_bookmark(doc, 'test_bookmark', 'This is the updated content.')

# 保存更新后的文档
doc.save('updated_bookmarked.docx')
```

上述代码首先定义一个 create_bookmark()函数，用于在指定的段落中创建一个书签。然后，创建一个 update_bookmark()函数，用于更新指定书签的内容。这两个函数都利用了 Python-docx 库中提供的底层 OxmlElement 类，以便直接操作 Word 文档的底层 XML 结构。

这个实战案例展示了如何使用 Python-docx 库实现自动化创建和更新书签的功能。这是一个非常实用的功能，可以帮助用户更有效地管理和操作 Word 文档。通过使用 Python-docx 库，用户可以轻松实现这个任务，无须手动创建和更新每个书签。

## 3.7　实战案例 4：批量添加或修改页眉与页脚

页眉和页脚在 Word 文档中扮演着重要角色，它们可以用来显示文档标题、作者、页码等重要信息。在处理多个文档时，经常需要为这些文档批量添加或修改页眉、页脚。Python-docx 库提供了操作页眉和页脚的功能，可以自动化完成这个任务。

本实战案例将学习如何使用 Python-docx 库实现批量添加或修改页眉、页脚的功能。首先定义一个函数来添加或修改页眉，然后创建另一个函数来添加或修改页脚，如代码 3-7 所示。

**代码 3-7　批量添加或修改页眉、页脚**

```
from docx import Document

def add_or_update_header(document, header_text):
    """
    为 Word 文档添加或修改页眉

    参数:
    document -- Document 对象
    header_text -- 页眉文本
```

```
        """
        header = document.sections[0].header
        if header.paragraphs:
            header.paragraphs[0].text = header_text
        else:
            header.add_paragraph(header_text)

    def add_or_update_footer(document, footer_text):
        """
        为 Word 文档添加或修改页脚

        参数:
        document -- Document 对象
        footer_text -- 页脚文本
        """
        footer = document.sections[0].footer
        if footer.paragraphs:
            footer.paragraphs[0].text = footer_text
        else:
            footer.add_paragraph(footer_text)

    # 打开一个 Word 文档
    doc = Document('example.docx')

    # 添加或修改页眉
    add_or_update_header(doc, 'This is a new header.')

    # 添加或修改页脚
    add_or_update_footer(doc, 'This is a new footer.')

    # 保存更新后的文档
    doc.save('updated_example.docx')
```

上述代码定义了两个函数：add_or_update_header()和 add_or_update_footer()。这两个函数都接收一个 Document 对象和一个文本参数，分别用于添加或修改页眉和页脚。这两个函数通过访问文档的 sections 属性来操作页眉和页脚。

这个实战案例展示了如何使用 Python-docx 库实现批量添加或修改页眉、页脚的功能。这是一个非常实用的功能，可以帮助用户在处理多个文档时保持一致的格式。通过使用 Python-docx 库，无须手动为每个文档添加或修改页眉、页脚便可以轻松实现这个任务。

## 3.8　实战案例 5：自动生成文档（报告和合同等）

在许多企业和组织中，编写报告和合同是一项日常工作。这项工作通常需要处理大量的结构化数据，并根据这些数据创建符合特定格式的 Word 文档。尽管这些工作可以手动完成，但往往效率低下，且容易出错。幸运的是，Python-docx 库能自动化这个过程，从而大大提高工作效率。

这个实战案例将使用 Python-docx 库来自动创建一个报告。首先定义一个模板，然后使用数据填充模板，最后保存结果文档。这样，每次需要创建新的报告或合同时，只

需要提供新的数据，就可以快速生成新的文档，如代码 3-8 所示。

**代码 3-8　自动生成文档**

```
from docx import Document

def create_report(template_filename, data, output_filename):
    """
    使用模板和数据自动生成报告

    参数:
    template_filename -- 模板文档的文件名
    data -- 一个字典，键是模板中的占位符，值是要替换占位符的数据
    output_filename -- 输出文档的文件名
    """
    doc = Document(template_filename)

    # Replace placeholders with actual data
    for key, value in data.items():
        for paragraph in doc.paragraphs:
            if key in paragraph.text:
                paragraph.text = paragraph.text.replace(key, str(value))
        for table in doc.tables:
            for row in table.rows:
                for cell in row.cells:
                    if key in cell.text:
                        cell.text = cell.text.replace(key, str(value))

    # Save the modified document
    doc.save(output_filename)

# 使用模板和数据创建新的报告
data = {'{TITLE}': 'Quarterly Report', '{QUARTER}': 'Q1', '{YEAR}':
'2023', '{AUTHOR}': 'John Doe', '{CONTENT}': 'This is the content of the
report.'}
create_report('report_template.docx', data, 'new_report.docx')
```

上述代码定义了一个函数 create_report()，它接收一个模板文档的文件名、一个数据字典和一个输出文档的文件名作为参数。该函数首先打开模板文档，然后遍历每个段落，查找并替换占位符。最后，保存新的文档。

这个实战案例展示了如何使用 Python-docx 库来自动生成文档。这是一个非常实用的功能，可以在处理大量格式相似的文档时提高工作效率并减少错误。通过使用 Python-docx 库，无须手动创建每个文档便可以轻松实现这个任务。

## 3.9　实战案例 6：插入图片和表格并将 Excel 或 CSV 数据导入 Word 文档生成表格

Word 文档不仅包含文本，还包含图片和表格。Python-docx 库提供了插入图片和表格的功能。当需要在 Word 文档中创建大量数据的表格时，可以利用 pandas 库先从 Excel

或 CSV 文件中读取数据，然后利用 Python-docx 库将数据插入 Word 文档中。

本实战案例介绍如何使用 Python-docx 库插入图片和表格，以及如何从 Excel 或 CSV 文件中读取数据，并将数据插入 Word 文档中，如代码 3-9 所示。

**代码 3-9　插入图片和表格并将Excel或CSV数据导入Word文档生成表格**

```
from docx import Document
from docx.shared import Inches
import pandas as pd

def insert_image_and_table(filename, image_path, table_data):
    """
    在 Word 文档中插入图片和表格

    参数:
    filename -- Word 文档的文件名
    image_path -- 图片的文件路径
    table_data -- 一个二维列表，表示表格的数据
    """
    doc = Document(filename)

    # 插入图片
    doc.add_picture(image_path, width=Inches(4.25))

    # 插入表格
    table = doc.add_table(rows=1, cols=len(table_data[0]))
    for row in table_data:
        cells = table.add_row().cells
        for i in range(len(row)):
            cells[i].text = str(row[i])

    # 保存文档
    doc.save(filename)

def import_data_from_excel(filename):
    """
    从 Excel 文件中导入数据

    参数:
    filename -- Excel 文件的文件名

    返回:
    一个二维列表，表示表格的数据
    """
    df = pd.read_excel(filename)
    return df.values.tolist()

# 从 Excel 文件中导入数据
table_data = import_data_from_excel('data.xlsx')

# 在 Word 文档中插入图片和表格
insert_image_and_table('document.docx', 'image.png', table_data)
```

上述代码定义了一个函数 insert_image_and_table()，该函数接收一个 Word 文档的文件名、一个图片的文件路径以及一个二维列表（表示表格的数据）作为参数。该函数首先打开指定的 Word 文档，然后在文档的末尾插入图片，并创建一个新的表格，使用提供的数据填充表格，最后保存修改后的文档。

此外，上述代码还定义了一个函数 import_data_from_excel()，该函数接收一个 Excel 文件的文件名作为参数，使用 pandas 库从 Excel 文件中读取数据，然后返回一个二维列表，表示表格的数据。

使用这种方法很容易在 Word 文档中插入图片和表格，甚至可以从 Excel 或 CSV 文件中导入数据，这无疑大大提高了处理 Word 文档的效率。

## 3.10　实战案例 7：Word 文档合并

在日常工作中，经常需要合并多个 Word 文档，尤其是当处理大量独立的报告或章节时。Python-docx 库提供了这样的功能，可以将多个文档合并成一个文档。

这个实战案例使用 Python-docx 库合并两个 Word 文档。首先创建一个新的文档，然后将两个源文档中的所有段落添加到新文档中，这样新文档就会有所有源文档的内容，如代码 3-10 所示。

**代码 3-10　Word文档合并**

```
from docx import Document

def merge_documents(output_filename, *filenames):
    """
    合并多个 Word 文档

    参数:
    output_filename -- 输出文档的文件名
    filenames -- 要合并的 Word 文档的文件名
    """
    merged_doc = Document()

    for filename in filenames:
        sub_doc = Document(filename)
        for element in sub_doc.element.body:
            merged_doc.element.body.append(element)

    merged_doc.save(output_filename)

# 合并 document1.docx 和 document2.docx
merge_documents('merged_document.docx', 'document1.docx', 'document2.docx')
```

这段代码定义了一个函数 merge_documents()。该函数接收一个输出文档的文件名和多个要合并的文档的文件名；然后创建一个新的 Document 对象，遍历所有要合并的文

档，将每个文档的所有元素添加到新文档中；最后保存合并后的文档。

这个功能特别适合用于合并大量的小文档，例如独立的报告或章节。通过使用文档合并功能，可以自动化这个经常手动完成的任务，从而提高效率，减少错误。

## 3.11　实战案例 8：批量将 Word 文档转换为 PDF

将 Word 文档转换为 PDF 格式是日常工作中常见的需求，因为 PDF 是一个更加便携和通用的格式。尽管 Python-docx 库本身不提供将 Word 文档转换为 PDF 的功能，但可以利用 Microsoft Word 的 COM 组件（仅在 Windows 平台上可用）或者使用第三方库，如 Python-docx2pdf 来实现这个功能。

本实战案例将介绍如何使用 Python-docx2pdf 库将 Word 文档批量转换为 PDF。首先，需要安装对应的库，可以使用以下命令安装，如代码 3-11 所示。

代码 3-11　安装Python-docx2pdf库

```
pip install docx2pdf
```

安装完成后，可以使用代码 3-12 进行批量转换。

代码 3-12　批量将Word文档转换为PDF

```
from docx2pdf import convert
import os

def batch_convert_word_to_pdf(input_folder, output_folder):
    """
    批量将 Word 文档转换为 PDF

    参数:
    input_folder -- 包含 Word 文档的输入文件夹
    output_folder -- 保存 PDF 文档的输出文件夹
    """
    for filename in os.listdir(input_folder):
        if filename.endswith(".docx"):
            input_path = os.path.join(input_folder, filename)
            output_path = os.path.join(output_folder, filename.replace
(".docx", ".pdf"))
            convert(input_path, output_path)

# 使用示例
input_folder = "word_documents"
output_folder = "pdf_documents"
batch_convert_word_to_pdf(input_folder, output_folder)
```

这段代码定义了一个函数 batch_convert_word_to_pdf()，该函数接收两个参数：一个是包含 Word 文档的输入文件夹，另一个是保存 PDF 文档的输出文件夹。该函数遍历输入文件夹中的所有文件，如果文件是一个.docx 文件，就将其转换为 PDF 并将结果保存到输出文件夹中。注意，这个操作可能需要一些时间，具体取决于文档的数量和大小。

这个功能可以轻松地将大量的 Word 文档转换为 PDF 格式，这样，我们可以更方便地分享和分发这些文档。

## 3.12　实战案例 9：批量修改 Word 文档属性

Word 文档的属性包括一些元数据，如标题、作者、主题、关键字等。修改这些属性可以更好地管理和搜索文档。Python-docx 库提供了接口，可以获取和修改这些属性。

这个实战案例将介绍如何使用 Python-docx 库批量修改 Word 文档的属性。首先定义一个函数，该函数接收一个文档文件名和一些新的属性值，然后将这些新的属性值应用到文档中，如代码 3-13 所示。

**代码 3-13　批量修改Word文档属性**

```
from docx import Document

def modify_document_properties(filename, title=None, author=None,
subject=None):
    """
    修改 Word 文档的属性

    参数:
    filename -- Word 文档的文件名
    title -- 新的标题（可选）
    author -- 新的作者（可选）
    subject -- 新的主题（可选）
    """
    doc = Document(filename)

    if title is not None:
        doc.core_properties.title = title
    if author is not None:
        doc.core_properties.author = author
    if subject is not None:
        doc.core_properties.subject = subject

    doc.save(filename)

# 修改 example.docx 的属性
modify_document_properties('example.docx', title='New Title', author='New
Author', subject='New Subject')
```

这段代码定义了一个函数 modify_document_properties()，该函数接收一个文档的文件名和一些可选的新的属性值。如果提供了新的属性值，这些新的属性值便会被应用到文档的相应属性中。最后，保存文档，以便保存这些更改。

这个功能对批量管理大量文档非常有用。通过批量修改文档的属性，可以更好地组织文档，更容易地找到特定的文档。

# 3.13　实战案例 10：Word 文档的加密和解密

Word 文档的加密和解密是常见的需求，特别是在处理敏感信息时。然而，Python-docx 库本身并不支持 Word 文档的加密和解密。在这种情况下，可以借助 Python 的其他库，如 PyPDF2 库，虽然这是一个处理 PDF 的库，但也可以用于 Word 文档的加密和解密。首先，需要安装对应的库，可以使用以下命令安装，如代码 3-14 所示。

代码 3-14　安装PyPDF2 库命令

```
pip install PyPDF2
```

**注意：这种方法仅适用于 PDF 格式的文档。如果文档是 Word 格式的，需要首先将其转换为 PDF 格式，然后进行加密或解密。具体内容可以参考 3.11 节，了解如何将 Word 文档转换为 PDF 文档。**

用 PyPDF2 进行文档加密的示例如代码 3-15 所示。

代码 3-15　用PyPDF2 进行文档加密

```
from PyPDF2 import PdfFileWriter, PdfFileReader

def encrypt_pdf(file, password):
    """
    加密 PDF 文档

    参数:
    file -- PDF 文档的文件名
    password -- 密码
    """
    parser = PdfFileWriter()
    pdf = PdfFileReader(file)

    for page in range(pdf.getNumPages()):
        parser.addPage(pdf.getPage(page))

    parser.encrypt(password)

    with open(f'encrypted_{file}', 'wb') as f:
        parser.write(f)

    print(f'encrypted_{file} Created...')

# 使用示例
encrypt_pdf('example.pdf', 'password')
```

使用 PyPDF2 进行文档解密的示例如代码 3-16 所示。

代码 3-16　用PyPDF2 进行文档解密

```
from PyPDF2 import PdfFileWriter, PdfFileReader

def decrypt_pdf(file, password):
    """
    解密 PDF 文档

    参数:
    file -- PDF 文档的文件名
    password -- 密码
    """
    parser = PdfFileWriter()
    pdf = PdfFileReader(file)

    if pdf.isEncrypted:
        pdf.decrypt(password)

        for page in range(pdf.getNumPages()):
            parser.addPage(pdf.getPage(page))

        with open(f'decrypted_{file}', 'wb') as f:
            parser.write(f)

        print(f'decrypted_{file} Created...')
    else:
        print('Document is not encrypted.')

# 使用示例
decrypt_pdf('encrypted_example.pdf', 'password')
```

在这两段代码中，首先打开 PDF 文档，然后通过 PyPDF2 库的 encrypt()和 decrypt()函数对文档进行加密和解密。加密或解密后的文档被保存为一个新的文件。

通过这种方法，可以对包含敏感信息的文档进行加密，以防止未经授权的访问。同样，也可以对加密的文档进行解密，以便可以访问文档的内容。

## 3.14　实战案例 11：自动创建目录和索引

在一个长的 Word 文档中，创建目录和索引是一个重要的步骤，它可以帮助用户更好地理解文档的结构，并快速找到用户感兴趣的内容。然而，手动创建目录和索引是一个非常烦琐的任务，尤其是对一个包含大量内容的长文档来说更是如此。其实，Python-docx 库并没有直接支持创建目录的功能，需要使用一种间接的方式来实现这个功能。具体来说，首先创建一个包含所有标题的列表，然后把这个列表插入文档的开始位置，如代码 3-17 所示。

代码 3-17　自动创建目录

```
from docx import Document
from docx.shared import Pt
from docx.oxml.ns import nsdecls
from docx.oxml import parse_xml
```

```
def create_table_of_contents(filename):
    """
    在 Word 文档中创建目录

    参数:
    filename -- Word 文档的文件名
    """
    doc = Document(filename)
    paragraphs = doc.paragraphs

    # 创建一个新的段落用于存放目录
    toc = doc.add_paragraph()
    toc_run = toc.add_run()

    # 遍历文档的所有段落，找到所有的标题
    for para in paragraphs:
        if para.style.name.startswith('Heading'):
            # 向目录中添加一个新的条目
            entry = toc_run.add_hyperlink(para._p, '', is_external=False)
            entry.text = para.text
            entry.font.size = Pt(12)
            toc_run.add_break()

    # 将目录插入文档的开始位置
    doc.element.body.insert(0, toc._p._tc)

    doc.save('new_'+filename)

# 使用示例
create_table_of_contents('example.docx')
```

在这段代码中，首先打开文档，遍历文档中的所有段落，找出所有的标题。然后，创建一个新的段落，用于存放目录，把找到的每个标题作为一个链接添加到目录中，链接的目标是标题对应的段落。最后，把目录插入文档的开始位置。通过这种方式，可以自动创建一个包含所有标题的目录。

**注意：这个目录并不是 Word 中的动态目录，也就是说，如果用户在文档中添加或删除了标题，目录不会自动更新。目前，Python-docx 还不支持创建索引。创建索引通常需要对文档的内容进行更深入的分析，这超出了 Python-docx 的功能范围。要创建索引，需要使用更强大的工具，如 Apache Lucene 或 Elasticsearch。**

## 3.15　实战案例 12：批量提取 Word 文档中的图片

在处理 Word 文档时，经常会遇到需要从文档中提取图片的情况。例如，获取文档中的所有图片，然后将它们保存为文件，以便在其他地方使用。Python-docx 库提供了直接的接口来访问 Word 文档中的图片，这使得从文档中提取图片变得非常简单。

首先需要了解的是，Word 文档中的图片被嵌入段落中。每个段落都可以包含多个“运行”，“运行”可以包含文本或图片。所以，为了提取图片，需要遍历文档的所有段落和运行，如代码 3-18 所示。

**代码 3-18　批量提取Word文档中的图片**

```
from docx import Document
import os

def extract_images_from_docx(filename):
    """
    从 Word 文档中提取所有的图片并保存到当前目录下的 images 文件夹中

    参数:
    filename -- Word 文档的文件名
    """
    doc = Document(filename)

    # 创建一个文件夹用于存放提取的图片
    if not os.path.exists('images'):
        os.makedirs('images')

    # 遍历文档的所有段落和运行
    for rel in doc.part.rels.values():
        if "image" in rel.reltype:
            # 找到图片
            image_data = rel.blob
            # 提取图片的扩展名
            image_ext = rel.reltype.split('/')[-1]
            # 创建图片的文件名
            image_name = 'images/image{}.{}'.format(rel.rId, image_ext)
            # 将图片数据写入文件
            with open(image_name, 'wb') as img_file:
                img_file.write(image_data)

# 使用示例
extract_images_from_docx('example.docx')
```

在这段代码中，首先打开 Word 文档，然后遍历文档的所有关系对象。关系对象是 Word 文档中用于表示各种元素（如文本、图片等）之间关系的对象。当找到一个关系对象的类型是“image”时，说明找到了一张图片。然后，提取出图片的数据，并将其保存为一个文件。使用关系对象的 ID 作为图片文件的名称，以确保每个图片文件的名称都

是唯一的。

通过这种方法可以轻松地从 Word 文档中提取所有的图片。

**注意：这个方法只能提取嵌入在文档中的图片，不能提取链接到外部文件的图片。如果用户想提取这些图片，需要使用其他的工具或库。**

## 3.16　实战案例 13：自动生成批注和脚注

批注和脚注是 Word 文档中的重要元素，它们被广泛用于注释文档内容，提供额外的解释或者信息。例如，在学术论文、法律文档或者技术文档中，经常会看到大量的批注和脚注。在 Python-docx 库中，批注和脚注可以通过简单的 API 来添加。

首先要明白，在 Word 文档中批注和脚注的本质是一种特殊的“运行”。它们都有自己的文本，可以被插入任何段落中。所以，为了添加批注和脚注，需要在指定的段落中创建一个新的批注或脚注的“运行”，如代码 3-19 所示。

**代码 3-19　自动生成批注和脚注**

```
from docx import Document

def add_footnotes_and_comments(doc, paragraph_index, text,
footnote_text,
comment_text):
    """
    在给定的段落中添加脚注和批注的文本

    参数:
    doc -- 文档对象
    paragraph_index -- 要操作的段落的索引
    text -- 要插入的文本
    footnote_text -- 脚注的文本
    comment_text -- 批注的文本
    """
    paragraph = doc.paragraphs[paragraph_index]
    run = paragraph.add_run(text)
    footnote = doc.add_footnote(footnote_text)
    comment = doc.add_comment(comment_text, run)
    return doc

# 使用示例
doc = Document('example.docx')
add_footnotes_and_comments(doc, 1, 'This is a test text', 'This is a test
footnote', 'This is a test comment')
doc.save('example_with_footnote_and_comment.docx')
```

在这段代码中，首先打开一个 Word 文档，然后在指定的段落中添加一个新的“运

行”。然后，在这个“运行”的基础上添加一个脚注和一个批注。最后，保存修改后的文档。

通过这种方式，可以自动地在 Word 文档中添加批注和脚注，这对批量处理 Word 文档，或者创建复杂的 Word 文档非常有用。

## 3.17　实战案例 14：自动生成多级标题和标题编号

在处理 Word 文档时，经常需要对文档结构进行调整，其中，多级标题和标题编号是常见的需求。Python-docx 库提供了创建多级标题和自动编号的功能，可以方便地生成具有层次结构的标题。

首先需要明确的是，Word 文档中的标题是通过样式来定义的。每个标题级别都有对应的样式，可以通过样式名称或样式索引来引用。可以使用 docx.enum.style.WD_STYLE_TYPE.PARAGRAPH 枚举类型中定义的样式类型来识别标题样式。

接下来，创建一个标题，并指定其级别和编号，如代码 3-20 所示。

**代码 3-20　自动生成多级标题和标题编号**

```
from docx import Document
from docx.enum.style import WD_STYLE_TYPE
from docx.oxml.ns import nsdecls
from docx.oxml import parse_xml

def add_heading_with_numbering(doc, text, level):
    """
    在文档中添加具有编号的多级标题

    参数:
    doc -- 文档对象
    text -- 标题文本
    level -- 标题级别
    """
    # 创建一个新的段落
    paragraph = doc.add_paragraph()

    # 设置段落的样式为标题样式
    paragraph.style = doc.styles['Heading %d' % level]

    # 获取段落的运行对象
    run = paragraph.add_run()

    # 创建标题编号
    numbering_element = parse_xml(r'<w:numPr xmlns:w="http://schemas.
openxmlformats.org/wordprocessingml/2006/main">'
                                  r'<w:ilvl w:val="%d" />' % (level-1) +
                                  r'<w:numId w:val="1" />'
                                  r'</w:numPr>')
    run._r.append(numbering_element)

    # 添加标题文本
```

```
        run.text = text

    # 使用示例
    doc = Document()
    add_heading_with_numbering(doc, 'Chapter 1', 1)
    add_heading_with_numbering(doc, 'Section 1.1', 2)
    add_heading_with_numbering(doc, 'Section 1.2', 2)
    add_heading_with_numbering(doc, 'Chapter 2', 1)
    add_heading_with_numbering(doc, 'Section 2.1', 2)
    doc.save('example_with_headings.docx')
```

在这段代码中，首先创建一个新的段落，并将其样式设置为指定级别的标题样式。然后，在段落的运行对象上创建一个标题编号，将其添加到运行对象的 XML 表示中。最后，在运行对象中设置标题文本。

通过这种方式，可以轻松地在 Word 文档中生成具有多级标题和标题编号的内容。这对创建结构化的文档或自动生成报告非常有用。

**注意：要使标题样式正确显示，须确保在文档中事先定义了标题样式，且级别与代码中使用的级别对应。**

## 3.18　小　　结

本章介绍了使用 Python 进行 Word 文档处理的相关技术和实践，并通过实战案例展示了多个常见的 Word 自动化任务的解决方案。

本章探索了如何批量提取 Word 文档中的文本、进行内容替换、自动创建和更新书签，以及批量添加或修改页眉和页脚等操作；如何生成各种类型的文档，如报告和合同，并了解了插入图片和表格的方法；研究了如何合并多个 Word 文档、将 Word 文档转换为 PDF 格式，以及对 Word 文档进行加密和解密。

通过本章的学习，用户可以掌握使用 Python-docx 库进行 Word 自动化处理的基础知识。这些技术和实践将帮助用户提高工作效率，减少重复工作，并能够应对各种与 Word 文档相关的任务。

在实际应用中，用户可以根据自己的需求和场景选择适用的方法和技巧，从而更好地运用 Python 进行 Word 自动化处理。通过运用这些技术，可以轻松处理大量文档，创建复杂的文档结构，并实现自动化的报告生成、数据导入等任务。