

作为一名程序员，经常要搜一些教程，有的教程是在线的，不提供离线版本，这就有些局限了。那么同样作为一名程序员，遇到问题就应该解决它，今天就来将在线教程保存为PDF以供查阅。

## 1、网站介绍

## 2、准备工作

### 2.1 软件安装

### 2.2 库安装

## 3、爬取内容

### 3.1 获取教程名称

### 3.2 获取目录及对应网址

### 3.3 获取章节内容

### 3.4 保存pdf

### 3.5 合并pdf

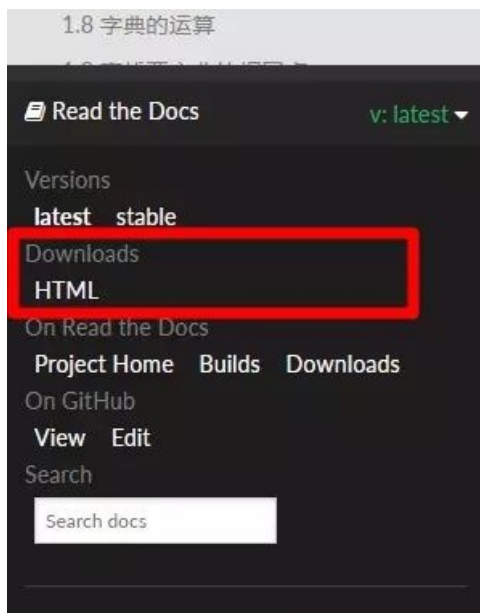
# 1、网站介绍

之前再搜资料的时候经常会跳转到如下图所示的在线教程：



包括一些github的项目也纷纷将教程链接指向这个网站。经过一番查找，该网站是一个可以创建、托管和浏览文档的网站，其网址为：<https://readthedocs.org>。在上面可以找到很多优质的资源。

该网站虽然提供了下载功能，但是有些教程并没有提供PDF格式文件的下载，如图：



该教程只提供了 HTML 格式文件的下载，还是不太方便查阅，那就让我们动手将其转成 PDF 吧！

## 2、准备工作

### 2.1 软件安装

由于我们是要把 html 转为 pdf，所以需要手动 **wkhtmltopdf**。Windows 平台直接在 <http://wkhtmltopdf.org/downloads.html> 下载稳定版的 **wkhtmltopdf** 进行安装，安装完成之后把该程序的执行路径加入到系统环境 `$PATH` 变量中，否则 `pdftk` 找不到 **wkhtmltopdf** 就出现错误 “No wkhtmltopdf executable found”。Ubuntu 和 CentOS 可以直接用命令行进行安装

```
$ sudo apt-get install wkhtmltopdf # ubuntu
$ sudo yum install wkhtmltopdf # centos
```

### 2.2 库安装

- `pip install requests` # 用于网络请求
- `pip install BeautifulSoup4` # 用于操作 html
- `pip install pdftk` # **wkhtmltopdf** 的 Python 封装包
- `pip install PyPDF2` # 用于合并 pdf

## 3、爬取内容

本文的目标网址为：[http://python3-cookbook.readthedocs.io/zh\\_CN/latest/](http://python3-cookbook.readthedocs.io/zh_CN/latest/)。

### 3.1 获取教程名称

页面的左边一栏为目录，按 F12 调出开发者工具并按以下步骤定位到目录元素：

- ① 点击开发者工具左上角“选取页面元素”按钮；
- ② 用鼠标点击左上角教程名称处。

通过以上步骤即可定位到目录元素，用图说明：

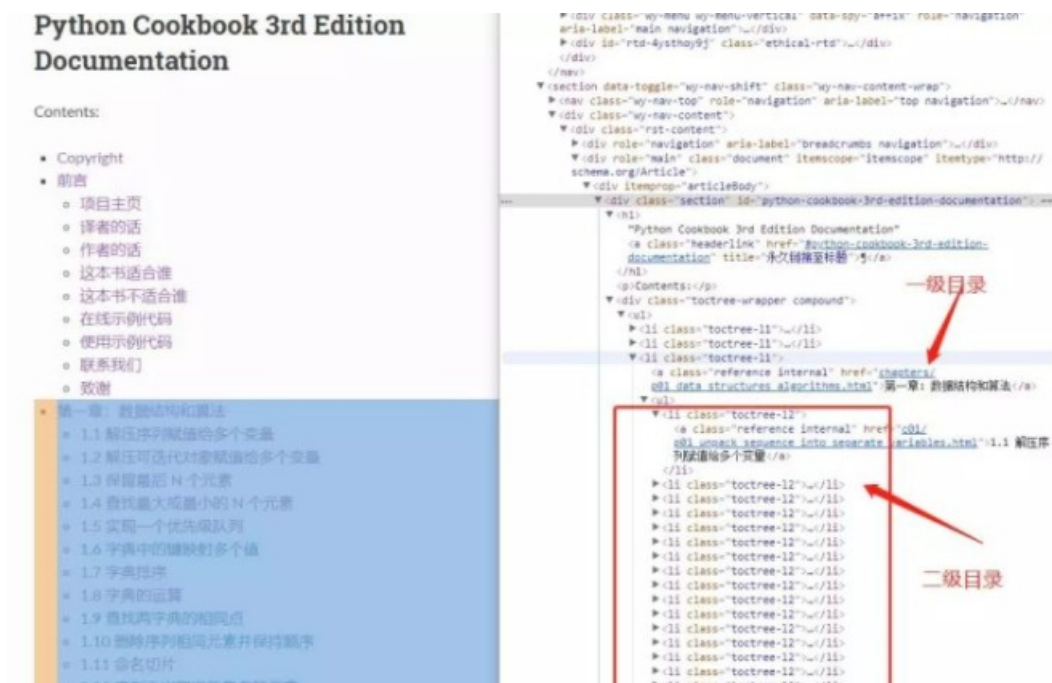


从图看到我们需要的教程名称包含在<div></div>之间的a标签里。假设我们已经获取到了网页内容为html，可以使用以下代码获取该内容：

```
book_name = soup.find('div', class_='wy-side-nav-search').a.text
```

## 3.2 获取目录及对应网址

使用与 2.1 相同的步骤来获取：



从图看到我们需要的目录包含在<div></div>之间，<li></li>标签里为一级目录及网址；<li></li>标签里为二级目录及网址。当然这个url是相对的url，前面还要拼接http://python3-cookbook.readthedocs.io/zh\_CN/latest/。

使用BeautifulSoup进行数据的提取：

```

# 全局变量
base_url = 'http://python3-cookbook.readthedocs.io/zh_CN/latest/'
book_name = ''
chapter_info = []

def parse_title_and_url(html):
    """
    解析全部章节的标题和url
    :param html: 需要解析的网页内容
    :return None
    """
    soup = BeautifulSoup(html, 'html.parser')

    # 获取书名
    book_name = soup.find('div', class_='wy-side-nav-search').a.text
    menu = soup.find_all('div', class_='section')
    chapters = menu[0].div.ul.find_all('li', class_='toctree-l1')
    for chapter in chapters:
        info = {}
        # 获取一级标题和url
        # 标题中含有'/'和'*'会保存失败
        info['title'] = chapter.a.text.replace('/', '').replace('*', ' ')
        info['url'] = base_url + chapter.a.get('href')
        info['child_chapters'] = []

```

```

# 获取二级标题和url
if chapter.ul is not None:
    child_chapters = chapter.ul.find_all('li')
    for child in child_chapters:
        url = child.a.get('href')
        # 如果在url中存在'#', 则此url为页面内链接, 不会跳转到其他页
        # 所以不需要保存
        if '#' not in url:
            info['child_chapters'].append({
                'title': child.a.text.replace('/', '').replace('*', ' '),
                'url': base_url + child.a.get('href'),
            })

chapter_info.append(info)

```

代码中定义了两个全局变量来保存信息。章节内容保存在`chapter_info`列表里，里面包含了层级结构，大致结构为：

```
[
  {
    'title': 'first_level_chapter',
    'url': 'www.xxxxxx.com',
    'child_chapters': [
      {
        'title': 'second_level_chapter',
        'url': 'www.xxxxxx.com',
      }
      ...
    ]
  }
  ...
]
```

### 3.3 获取章节内容

还是同样的方法定位章节内容：

The screenshot shows a web page with a sidebar and a main content area. The sidebar contains a list of chapter topics, and the main content area shows the text of the selected chapter. A red box highlights the 'itemprop' attribute in the HTML code, which is used to identify the content element for structured data.

**第一章：数据结构和算法**

Python 提供了大量的内置数据结构，包括列表、集合以及字典。使用这些数据结构是很简单的。但是，我们也会经常碰到排序和过滤等等这些普遍存在的问题。因此，这一章的目的就是常见的问题和算法。另外，我们也会给出在集合模块 `collections` 中这些数据结构的方法。

- 1.1 解压缩赋值给多个变量
- 1.2 解压缩迭代对象赋值给多个变量
- 1.3 保留最后 N 个元素
- 1.4 查找最大或最小的 N 个元素
- 1.5 实现一个优先队列
- 1.6 字典中的键映射多个值
- 1.7 字典排序
- 1.8 字典的运算
- 1.9 查找两字典的相同点
- 1.10 删除序列相同元素并保持顺序
- 1.11 命名切片
- 1.12 序列中出现次数最多的元素
- 1.13 通过某个关键字排序一个字典列表
- 1.14 排序下支持项生比较的对象
- 1.15 通过某个字段得记录分组
- 1.16 过滤序列元素
- 1.17 从字典中提取子集

Python 提供了大量的内置数据结构，包括列表、集合以及字典。大多数情况下使用这些数据结构是很简单的。但是，我们也会经常碰到排序和过滤等等这些普遍存在的问题。因此，这一章的目的就是常见的问题和算法。另外，我们也会给出在集合模块 `collections` 中这些数据结构的方法。

当操作这些数据结构的方法。

当操作这些数据结构的方法。

### 05.获取章节内容

代码中我们通过 `itemprop` 这个属性来定位，好在一级目录内容的元素位置和二级目录内容的元素位置相同，省去了不少麻烦。

```

html_template = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
</head>
<body>
{content}
</body>
</html>
"""

def get_content(url):
    """
    解析URL，获取需要的html内容
    :param url: 目标网址
    :return: html
    """
    html = get_one_page(url)
    soup = BeautifulSoup(html, 'html.parser')
    content = soup.find('div', attrs={'itemprop': 'articleBody'})
    html = html_template.format(content=content)
    return html

```

## 3.4 保存pdf

```

def save_pdf(html, filename):
    """
    把所有html文件保存到pdf文件
    :param html: html内容
    :param file_name: pdf文件名
    :return:
    """
    options = {
        'page-size': 'Letter',
        'margin-top': '0.75in',
        'margin-right': '0.75in',
        'margin-bottom': '0.75in',
        'margin-left': '0.75in',
        'encoding': "UTF-8",
        'custom-header': [
            ('Accept-Encoding', 'gzip')
        ],
        'cookie': [
            ('cookie-name1', 'cookie-value1'),
            ('cookie-name2', 'cookie-value2'),
        ],
        'outline-depth': 10,
    }

    pdfkit.from_string(html, filename, options=options)

```

```

def parse_html_to_pdf():
    """
    解析URL，获取html，保存成pdf文件
    :return: None
    """
    try:
        for chapter in chapter_info:
            ctitle = chapter['title']
            url = chapter['url']
            # 文件夹不存在则创建（多级目录）
            dir_name = os.path.join(os.path.dirname(__file__), 'gen', ctitle)
            if not os.path.exists(dir_name):
                os.makedirs(dir_name)

            html = get_content(url)

            pdf_path = os.path.join(dir_name, ctitle + '.pdf')
            save_pdf(html, os.path.join(dir_name, ctitle + '.pdf'))

            children = chapter['child_chapters']
            if children:
                for child in children:
                    html = get_content(child['url'])
                    pdf_path = os.path.join(dir_name, child['title'] + '.pdf')
                    save_pdf(html, pdf_path)

    except Exception as e:
        print(e)

```

### 3.5 合并pdf

经过上一步，所有章节的pdf都保存下来了，最后我们希望留一个pdf，就需要合并所有pdf并删除单个章节pdf。

```

from PyPDF2 import PdfFileReader, PdfFileWriter

def merge_pdf(infnList, outfn):
    """
    合并pdf
    :param infnList: 要合并的PDF文件路径列表
    :param outfn: 保存的PDF文件名
    :return: None
    """

    pagenum = 0
    pdf_output = PdfFileWriter()

    for pdf in infnList:
        # 先合并一级目录的内容
        first_level_title = pdf['title']
        dir_name = os.path.join(os.path.dirname(
            __file__), 'gen', first_level_title)
        padf_path = os.path.join(dir_name, first_level_title + '.pdf')

        pdf_input = PdfFileReader(open(padf_path, 'rb'))
        # 获取 pdf 共用多少页
        page_count = pdf_input.getNumPages()
        for i in range(page_count):
            pdf_output.addPage(pdf_input.getPage(i))

        # 添加书签
        parent_bookmark = pdf_output.addBookmark(
            first_level_title, pagenum=pagenum)

        # 页数增加
        pagenum += page_count

```

```

# 存在子章节
if pdf['child_chapters']:
    for child in pdf['child_chapters']:
        second_level_title = child['title']
        padf_path = os.path.join(dir_name, second_level_title + '.pdf')

        pdf_input = PdfFileReader(open(padf_path, 'rb'))
        # 获取 pdf 共用多少页
        page_count = pdf_input.getNumPages()
        for i in range(page_count):
            pdf_output.addPage(pdf_input.getPage(i))

        # 添加书签
        pdf_output.addBookmark(
            second_level_title, pagenum=pagenum, parent=parent_bookmark)
        # 增加页数
        pagenum += page_count

# 合并
pdf_output.write(open(outfn, 'wb'))
# 删除所有章节文件
shutil.rmtree(os.path.join(os.path.dirname(__file__), 'gen'))

```

本来PyPDF2库中有一个类PdfFileMerger专门用来合并pdf，但是在合并过程中会抛出异常，网上有人也遇到同样的问题，解



决办法是修改库源码，本着“不动库源码”的理念，毅然选择了上面这种比较笨的办法，代码还是比较好理解的。

经过以上几个步骤，我们想要的pdf文件已经生成，一起来欣赏一下劳动成果：

Copyright
前言
第一章：数据结构和算法
第二章：字符串和文本
第三章：数字日期和时间
3.1 数字的四舍五入
3.2 执行精确的浮点数运算
3.3 数字的格式化输出
3.4 二八十六进制整数
3.5 字节到大整数的打包与解包
3.6 复数的数字运算
3.7 无穷大与NaN
3.8 分数运算
3.9 大数组组运算
3.10 矩阵与线性代数运算
3.11 随机选择
3.12 基本的日期与时间转换
3.13 计算最后一个周五的日期
3.14 计算当前月份的日期范围
3.15 字符串转换为日期
3.16 结合时区的日期操作
第四章：迭代器与生成器
4.1 手动遍历迭代器
4.2 代理迭代
4.3 使用生成器创建新的迭代模式
4.4 实现迭代器协议
4.5 反向迭代
4.6 带有外部状态的生成器函数
4.7 迭代器切片
4.8 跳过可迭代对象的开始部分
4.9 排列组合的迭代
4.10 序列上索引值迭代
4.11 同时迭代多个序列
4.12 不同集合上元素的迭代
4.13 创建数据处理管道
4.14 展开嵌套的序列

## 1.1 解压序列赋值给多个变量

### 问题

现在有一个包含 N 个元素的元组或者是序列，怎样将它里面的值解压后同时赋值给 N 个变量？

### 解决方案

任何的序列（或者是可迭代对象）可以通过一个简单的赋值语句解压并赋值给多个变量。唯一的前！必须跟序列元素的数量是一样的。

代码示例：

```
>>> p = (4, 5)
>>> x, y = p
>>> x
4
>>> y
5
>>>
>>> data = [ 'ACME', 50, 91.1, (2012, 12, 21) ]
>>> name, shares, price, date = data
>>> name
'ACME'
>>> date
(2012, 12, 21)
>>> name, shares, price, (year, mon, day) = data
>>> name
'ACME'
>>> year
2012
>>> mon
12
>>> day
21
>>>
```

如果变量个数和序列元素的个数不匹配，会产生一个异常。

代码示例：

```
>>> p = (4, 5)
>>> x, y, z = p
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: need more than 2 values to unpack
>>>
```