前言

学 Rust 也有一段时间了,网上也有不少官方文档的中文翻译版,但是似乎只有 Rust中文网站 文档一直是最新的,奈何并没有 PDF 供直接下载,是在是不太方便,为了方便阅读以及方便后续文档更新,决定用 Python 写一个爬虫将网页下载下来保持为 PDF. 最后完成结果如下:

Rust编程语言

> 本书同步于官方 The Rust Programming Language 仓库 地址 ！ 本书初始基于 trpl-zh-cn ,感谢作者.现将与Rust官方文档同步更新.

Why Rust

Rust 是一种安全、并发、实用的编程语言，有着惊人的运行速度，能够防止段错误，并保证线程安全,使每个人都能够构建 可靠，高效的软件

性能

Rust非常快速且节省内存：没有运行时或垃圾收集器，它可以为性能关键型服务提供动力，在嵌入式设备上运行，并且可以轻松地与其他语言集成

是的没错,将官网样式也保留下来成功转为 PDF,接下来分享一下整个爬虫的过程,最终的爬虫**可以导出任意 VuePress 搭建的网站为 PDF**.

爬虫

依赖库的选定

1. requests
2. BeautifulSoup4
3. pdfkit

关于 requests 和 BeautifulSoup4 库这里就不做介绍了, 写过爬虫的基本上都接触过, 重点说一下 pdfkit 库, 毫无疑问,它就是导出 PDF 的关键,简单说一下它的用法

PdfKit

PdfKit 库是对 Wkhtmltopdf 工具包的封装类,所以在使用之前,需要去官网下载相应的安装包安装到电脑上, 下载地址

OS	Flavor	Downloads	Comments
Windows	MSVC 2015	win32 / win64	Installer for Windows Vista or later
	MXE (MinGW-w64)	win32 / win64	7z archive for Windows XP/2003 or later
Linux	Debian 9 (stretch)	amd64 / i386 / raspbian	Package (.deb) built on Debian 9.4
	Debian 8 (jessie)	amd64 / i386	Package (.deb) built on Debian 8.10
	Ubuntu 18.04 (bionic)	amd64 / i386	Package (.deb) built on Ubuntu 18.04
	Ubuntu 16.04 (xenial)	amd64 / i386	Package (.deb) built on Ubuntu 16.04.4
	Ubuntu 14.04 (trusty)	amd64 / i386	Package (.deb) built on Ubuntu 14.04.5
	CentOS 7	x86_64 / i686	Package (.rpm) built on CentOS 7.4.1804
	CentOS 6	x86_64 / i686	Package (.rpm) built on CentOS 6.9
macOS	Cocoa	64-bit	Installer (.pkg) for OS X 10.7 or later
	Carbon	32-bit	Installer (.pkg) for OS X 10.7 or later

可选: 安装完成之后可以 Windows 下可以将安装路径添加到系统环境变量中

安装完成之后,说一下 PdfKit 的常用方法,常用方法有三个

from_url

```
def from_url(url, output_path, options=None, toc=None, cover=None,
             configuration=None, cover_first=False):
    """
    Convert file of files from URLs to PDF document

    :param url: URL or list of URLs to be saved
    :param output_path: path to output PDF file. False means file will be returned as string.
    :param options: (optional) dict with wkhtmltopdf global and page options, with or w/o '--'
    :param toc: (optional) dict with toc-specific wkhtmltopdf options, with or w/o '--'
    :param cover: (optional) string with url/filename with a cover html page
    :param configuration: (optional) instance of pdfkit.configuration.Configuration()
    :param configuration_first: (optional) if True, cover always precedes TOC

    Returns: True on success
    """

    r = PDFKit(url, 'url', options=options, toc=toc, cover=cover,
               configuration=configuration, cover_first=cover_first)

    return r.to_pdf(output_path)
```

从函数名上就很容易理解这个函数的作用,没错就是根据 url 下载网页为 PDF

from_file()

```
def from_file(input, output_path, options=None, toc=None, cover=None, css=None,
              configuration=None, cover_first=False):
    """
    Convert HTML file or files to PDF document

    :param input: path to HTML file or list with paths or file-like object
    :param output_path: path to output PDF file. False means file will be returned as string.
    :param options: (optional) dict with wkhtmltopdf options, with or w/o '--'
    :param toc: (optional) dict with toc-specific wkhtmltopdf options, with or w/o '--'
    :param cover: (optional) string with url/filename with a cover html page
    :param css: (optional) string with path to css file which will be added to a single input file
    :param configuration: (optional) instance of pdfkit.configuration.Configuration()
    :param configuration_first: (optional) if True, cover always precedes TOC

    Returns: True on success
    """

    r = PDFKit(input, 'file', options=options, toc=toc, cover=cover, css=css,
               configuration=configuration, cover_first=cover_first)

    return r.to_pdf(output_path)
```

这个则是从文件中生成 PDF, 也是我最后选择的方案,至于为什么没有选择 *from_url()* ,稍后等我分析完,就会明白了.

from_string

```
def from_string(input, output_path, options=None, toc=None, cover=None, css=None,
                configuration=None, cover_first=False):
    """
    Convert given string or strings to PDF document

    :param input: string with a desired text. Could be a raw text or a html file
    :param output_path: path to output PDF file. False means file will be returned as string.
    :param options: (optional) dict with wkhtmltopdf options, with or w/o '--'
    :param toc: (optional) dict with toc-specific wkhtmltopdf options, with or w/o '--'
    :param cover: (optional) string with url/filename with a cover html page
    :param css: (optional) string with path to css file which will be added to a input string
    :param configuration: (optional) instance of pdfkit.configuration.Configuration()
    :param configuration_first: (optional) if True, cover always precedes TOC

    Returns: True on success
    """

    r = PDFKit(input, 'string', options=options, toc=toc, cover=cover, css=css,
               configuration=configuration, cover_first=cover_first)

    return r.to_pdf(output_path)
```

这个方法则是从字符串中生成 PDF,很明显没有办法保持网页样式,所以不考虑.关于更多 PdfKit 的用法,可以去 wkhtmltopdf文档 查看

分析目标网页

依赖库选定完毕,接下来就是分析目标网页,开始写爬虫的过程了.

测试 PdfKit

PdfKit 自带一个 from_url 生成 PDF 的功能,如果可以生成合适的 PDF,那我们只需要获取所有网页链接就可以了,可以节省很多时间,

先测试一下生成的效果

```
import pdfkit
pdfkit.from_url("https://rustlang-cn.org/office/rust/book/", 'out.pdf', configuration=pdfkit.configuration(
                wkhtmltopdf="path/to/wkhtmltopdf.exe"))
```

导出结果如下:

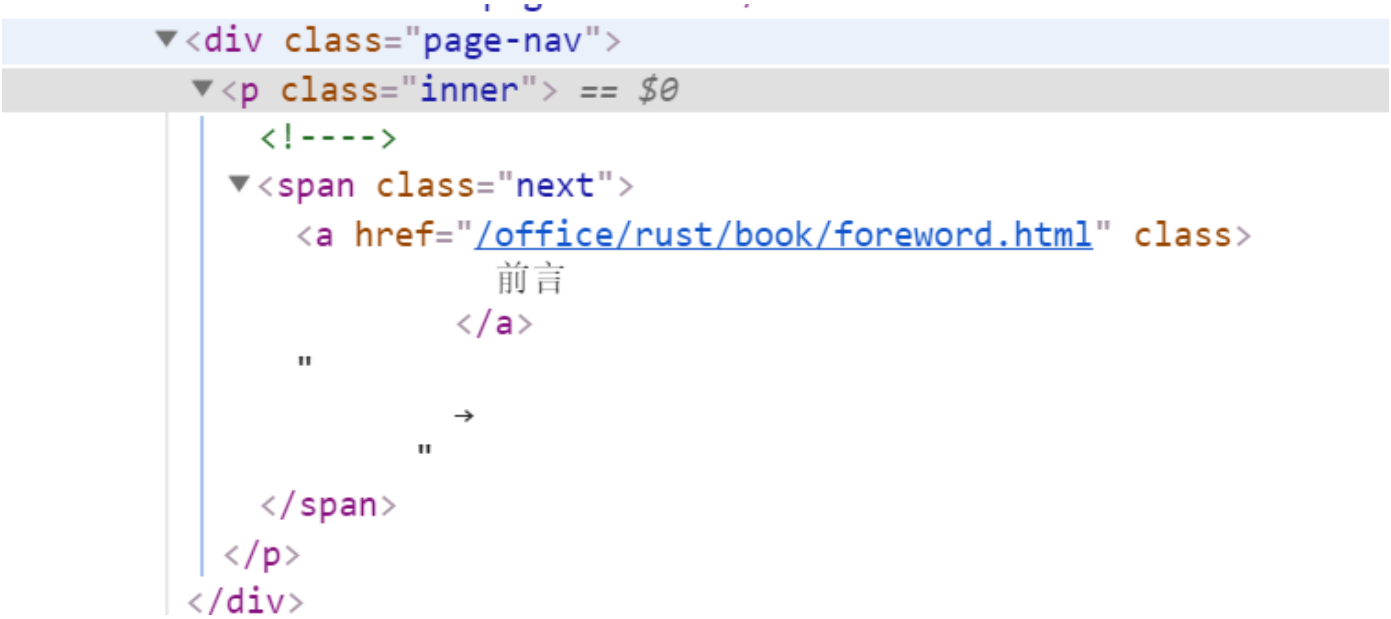
从结果不难看出,网页的样式保存下来了,但是侧边栏,顶部和底边导航栏也都被保留下来了,并且侧边栏还挡住了主要内容,所以使用 from_url 这个方法就被排除了.

最终方案

通过测试,我们得知不能使用 from_url 那么只能通过使用 from_file 去导出了, 并且在我们将网页下载下来保存到本地之前,我们需要修改网页内容,移除**顶部导航栏,侧边栏,以及底部导航栏**

获取相应元素

现在让我们先获取页面下一页链接,打开浏览器调试模式,审查一下网页元素,不难发现所有下一页导航,都处于 之下的超链接 中,如下图:



通过同样的方法,不难发现顶部导航栏,侧边栏,以及底部导航栏对应的元素,依次为

```
<div class="navbar"></div>
<div class="sidebar"></div>
<div class="page-edit"></div>
```

找到对应的元素接着就是**获取链接和销毁不必要元素**

```
class DownloadVuePress2Pdf:

    def get_content_and_next_url(self, content): # content 为网页内容
         # 获取链接和销毁不必要元素
        navbar = soup.select('.navbar')
        if len(navbar):
            navbar[0].decompose()

        sidebar = soup.select('.sidebar')
        if len(sidebar):
            sidebar[0].decompose()

        page_edit = soup.select('.page-edit')
        if len(page_edit):
            page_edit[0].decompose()

        # 注意下一页链接在底部导航栏元素中,
        # 要先获取链接后,才能销毁元素,顺序不能颠倒
        next_span = soup.select(".next")
        if len(next_span):
            next_span_href = next_span[0].a['href']
        else:
            next_span_href = None

        page_nav = soup.select('.page-nav')
        if len(page_nav):
            page_nav[0].decompose()
```

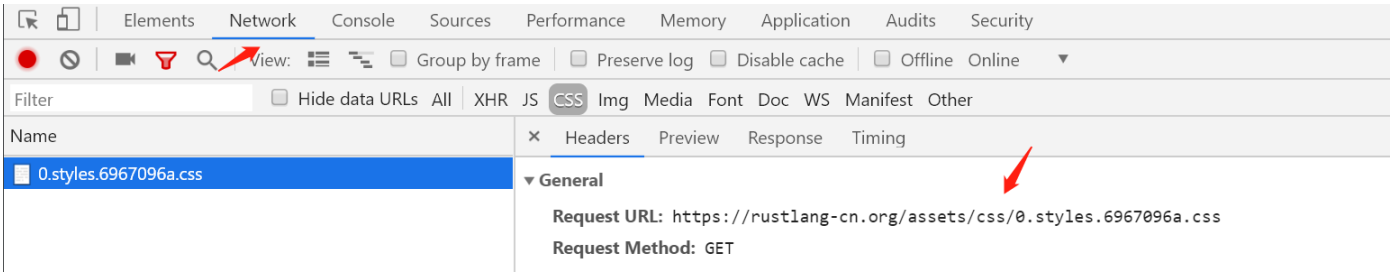
保持导出 PDF 样式

为了使得导出 PDF 的样式和网页一致,我们有俩种方法:

1. 根据源码在对应目录建立本地 css 文件,显然这种方法不具有普遍性,不能每导出一个网站,我们就新建一个 css 文件
2. 既然本地的不行,那我们就将网页中的 css 链接 href 地址指向远程 css

在上述代码中添加如下代码:

```
for link in links:
     if not link['href'].startswith("http"):
        link['href'] = css_domain + link['href'] # css_domain 为 css 默认域名,需要设置,获取方式可见下图
```



导出

通过上述的方式,我们将网页下载下来保存到本地,全部下载完成之后,最后就是导出为 PDF 了,通过 from_file() 方法很容易完成导出这个操作

```
pdfkit.from_file([文件列表], "导出的文件名称.pdf", options=options, configuration=config)
```

Copy

至此导出 Rust 官网文档为 PDF 的过程全部完成,效果如开头展示的那样

注意: 由于 VuePress 搭建的网站基本上布局格式一样, 所以上面的代码同样可以用来导出其他由 VuePress 构建的网站

完整代码

搜索公众号 **LeeTao**，回复 20190509 即可获得