**暨南大学本科实验报告专用纸**

课程名称 《Python程序设计》 成绩评定

实验项目名称 爬虫设计 指导教师 卢建朱

实验项目编号 实验项目类型 综合 实验地点

学生姓名 黄凯 学号 2022101541

学院 信息科学技术学院 系 计算机 专业 软件工程

实验时间 2024 年 11 月 10 日 午～ 11 月 21 日 午

温度 ℃湿度

1. **实验目的**

**1）学习和掌握 Python 中 urllib、request 模块的基本用法。**

**2）熟悉网络请求的基础操作，包括发送请求、设置超时时间、以及处理异常等。**

**3）掌握数据流的分块读取方式，提高对大数据量请求的处理能力。**

**4）掌握 parsel 模块的 CSS 选择器用法，用于解析 HTML 数据**

**5）掌握处理多层次网页请求的基本流程**

**6）熟悉 Scrapy 框架的基本结构与工作原理。**

**7）学习如何在 Scrapy 中使用 XPath 提取网页内容。**

**8）提升学生自主学习能力、实践能力、动手能力以及课堂知识的掌握程度**

1. **实验环境**

**编程语言: Python 3.8+**

**开发工具: VS Code / PyCharm / Jupyter Notebook**

1. **实验内容**
2. **用教材11·2节urllib模块爬取网页信息(P143-144)或教程11·4节BeautifulSoup库爬取中小型网站信息【必做】**
3. **用教材11·3节Scrapy框架实现中大型网站信息爬取（给定设计框架）【必做】**
4. **综合1、2在Scrapy框架下实现BeautifulSoup库【选做】**
5. **实验步骤**
6. **实验一**

1）导入 urllib.request 模块，用于构造请求和处理响应。

2）设置请求 URL 为 https://www.baidu.com。

3）使用 Request 对象构造 HTTP 请求。

4）设置 timeout 参数为 5 秒，确保在网络延迟较高时避免无限等待。

5）通过 urlopen 方法打开请求，并读取服务器响应：按照块大小（1024 字节）分块读取数据。

6）使用循环读取数据块，并将数据块追加到列表中。

7）将读取的二进制数据拼接后解码为字符串。

8）打印解码后的内容到控制台。

9）对可能的网络异常和其他运行时异常使用 try-except 捕获并处理。

10）关闭连接，确保资源释放。

代码：

import urllib.request

try:

    # 设置超时时间为5秒

    url = 'https://www.baidu.com'

    req = urllib.request.Request(url)

    with urllib.request.urlopen(req, timeout=5) as response:

        # 分块读取数据

        data = []

        block\_size = 1024  # 每次读取1024字节

        while True:

            block = response.read(block\_size)

            if not block:

                break

            data.append(block)

        content = b''.join(data)

        decoded\_content = content.decode('utf - 8')

        print(decoded\_content)

except urllib.request.URLError as e:

    print(f"网络请求出现错误: {e}")

except Exception as ex:

    print(f"其他错误: {ex}")

1. **实验二**
2. 设置请求参数

1）定义请求头，包括 Cookie 和 User-Agent，模拟真实用户访问，避免被反爬机制拦截。

2）确定爬取目标 URL 为小说目录页面。

2. 发送请求并解析主页面

1）使用 requests.get 方法获取主页面 HTML 内容。

2）利用 parsel.Selector 创建 HTML 解析对象。

3）使用 CSS 选择器提取小说标题、章节名称及章节链接。

3. 逐章节爬取内容

1）循环遍历章节名称和链接。

2）对每个章节发送 HTTP 请求，获取其内容页面 HTML。

3）使用 CSS 选择器提取章节正文内容。

4. 保存内容至文件

按章节顺序，将标题和正文内容写入到本地文件中，采用追加模式 (mode="a")。

5. 运行并验证

执行代码，检查小说文本文件是否生成并包含完整内容。

代码：

# 导入数据请求模块

import requests

# 导入数据解析模块

import parsel

# 定义请求头

headers = {

    # Cookie 用户信息，常用于检测是否有登录账号

    'Cookie': "\_ga=GA1.2.1930843405.1731741854; \_gid=GA1.2.425953489.1731741854; \_ga\_PJ0N20S6JN=GS1.2.1731741854.1.1.1731741872.42.0.0",

    # User-Agent 用户代理，表示浏览器/设备的基本身份信息

    'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36"

}

# url 地址

url = "https://www.quanben.io/n/mingyiguinu/list.html"

# 用requests库发送请求，注意请求参数与请求方法（GET获取信息/POST传递信息）

response = requests.get(url=url, headers=headers)

# 获取响应文本数据(html字符串数据)

html = response.text

# 转换为可解析的对象

selector = parsel.Selector(html)

# 提取书名：.info-name利用class类名进行定位，h1定位h1标签，::text获取标签内文本内容

title = selector.css("h1[itemprop='name headline']::text").get()

print(title)

# 提前章节ID和章节名

chapter\_titles = selector.css(".list3 li a span[itemprop='name']::text").getall()

print(chapter\_titles)

# 章节ID是href=的值，利用::attr获取href属性值

chapter\_ids = selector.css(".list3 li a::attr(href)").getall()

print(chapter\_ids)

# for循环遍历获取列表中的元素

for chapter\_title, link in zip(chapter\_titles, chapter\_ids):

    # print(chapter\_title)

    link\_url = "https://www.quanben.io" + link

    # print(link\_url)

    # 发送请求，获取响应文本数据

    link\_data = requests.get(url=link\_url, headers=headers).text

    # 解析数据提取小说内容

    link\_selector = parsel.Selector(link\_data)

    content\_list = link\_selector.css(".articlebody p ::text").getall()

    # 将列表合并成字符串

    content = "".join(content\_list[1:]) # 去掉第一行(标题)

    with open(title + ".txt", mode="a", encoding="utf-8") as f:

        f.write(chapter\_title)

        f.write("\n")

        f.write(content)

        f.write("\n")

1. **实验三**
2. 创建 Scrapy 项目

1）使用 scrapy startproject my\_project 创建一个 Scrapy 项目。

2）将代码保存为项目中的 Spider 文件（如 my\_spider.py）。

1. 定义爬取目标

1）设置 start\_urls 指定目标网站 URL（如 https://example.com）。

1. 实现数据提取逻辑

1）在 parse 方法中：使用 XPath 表达式提取页面内容，包括标题、段落、链接、图片、表格等。

2）对于链接，递归调用 response.follow，实现多层爬取。

1. 运行爬虫：将URL替换为想要爬取的URL并调用MySpider爬取数据
2. 实验结果验证

确认提取的内容与网页源数据一致。

代码：

import scrapy

# 在Scrapy框架下实现BeautifulSoup库

class MySpider(scrapy.Spider):

    name = "beautifulsoup\_spider"

    start\_urls = [

        'https://example.com',  # 将其替换为你想爬取的URL

    ]

    def parse(self, response):

        # 使用 Scrapy 的选择器来提取页面标题

        title = response.xpath('//title/text()').get()

        print(f"Page title: {title}")

        # 查找所有链接

        for link in response.xpath('//a'):

            href = link.xpath('@href').get()

            text = link.xpath('text()').get()

            print(f"Found link: {href}, Link text: {text}")

            # 在实际项目中，你可以根据需要使用 response.follow 进行后续爬取

            if href and not href.startswith('#'):

                yield response.follow(href, callback=self.parse)

        # 提取段落内容

        paragraphs = response.xpath('//p/text()').getall()

        for p in paragraphs:

            print(f"Paragraph: {p}")

        # 提取图片的 src 属性和 alt 属性

        images = response.xpath('//img')

        for img in images:

            src = img.xpath('@src').get()

            alt = img.xpath('@alt').get()

            print(f"Image found: src={src}, alt={alt}")

        # 提取表格内容

        tables = response.xpath('//table')

        for table in tables:

            rows = table.xpath('.//tr')

            for row in rows:

                cells = row.xpath('.//td/text() | .//th/text()').getall()

                print(f"Table row: {cells}")

        # 提取页面中的所有标题 (h1 - h6)

        for i in range(1, 7):

            headers = response.xpath(f'//h{i}/text()').getall()

            for header in headers:

                print(f"H{i} Header: {header}")

1. **实验结果**
2. **实验一**



1. **实验二（运行时间很长）**



