浙江理工大学科技与艺术学院

KEYI COLLEGE OF ZHEJIANG SCI-TECH UNIVERSITY

程序设计基础

大作业报告

大作业名称 ：校内新闻页面 Python 爬虫设计

大作业时间 ：2025.12.16

专　　业　大数据管理与应用

班　　级　24 大数据管理与应用 1

学生姓名　王俊翰

学　　号　Xc24590101

指导老师　叶杰锋

一、需求分析

（一）功能需求

爬取范围覆盖：严格遵循校内网全站点爬虫的核心要求，聚焦浙江理工大学科技与艺术学院官网核心版块，优先实现新闻中心（https://www.ky.zstu.edu.cn/news/）的定向爬取，为后续扩展至教务系统、二级学院官网等其他校内站点预留架构支持。

目标数据采集：精准抓取新闻、通知类核心信息，包括新闻标题、发布日期、原文链接等关键字段，确保数据详实具体，满足校内信息整合与检索需求。

数据存储规范：采用 MongoDB 数据库存储数据，通过创建唯一索引、文本索引和时间索引，保证数据唯一性、可检索性和有序性，符合任务要求的 “数据规范整理后存入指定数据库” 标准。

辅助功能支持：具备爬取统计、数据导出（JSON 格式）、日志记录功能，可实时监控爬取进度、查看爬取结果，便于后续数据复用与问题排查。

（二）性能需求

稳定性保障：支持分页爬取与失败重试机制，针对网络波动、页面访问超时等异常情况，采用指数退避策略进行最多 3 次重试，确保爬取过程不轻易中断。

高效去重：通过数据库 URL 唯一索引与内存去重结合的方式，避免重复爬取同一篇新闻，提升数据存储效率与爬取质量。

合规爬取：设置请求延迟（1 秒 / 次）与标准请求头，模拟浏览器行为，避免对目标服务器造成过大压力，符合网络爬虫的合规性要求。

（三）数据需求

数据格式：结构化存储爬取信息，每条数据包含标题、URL、发布日期、正文内容、来源、爬取时间等字段，确保数据的完整性与规范性。

数据质量：过滤无效数据（如标题过短、内容为空的条目），限制正文内容长度（2000 字以内），清理页面脚本、样式等冗余信息，保证数据可用性。

（四）文档与交付需求

代码需详实具体，不依赖 AI 全量生成，核心逻辑清晰可追溯，包含完整的注释说明。

配套文档需涵盖操作步骤、网络请求抓取截图、HTML 页面结构分析、数据库落库成功截图等内容，并粘贴核心代码，符合任务指导书的文档编写规范。

二、设计思路

（一）架构设计：模块化分层设计

采用面向对象的模块化设计思想，将爬虫系统拆解为 4 个核心模块，各模块职责单一、耦合度低，便于维护与扩展：

核心爬虫模块：封装于 SchoolNewsCrawler 类中，包含 fetch_page（页面请求）、crawl_news_section（版块爬取）等方法，负责发起网络请求、遍历分页页面，是爬取流程的核心驱动。

数据解析模块：通过 find_news_links（链接提取）、parse_news_page（详情解析）实现数据提取，采用多组 CSS 选择器适配不同页面结构，结合正则表达式提取发布日期，确保解析的灵活性与准确性。

数据存储模块：包含 connect_mongodb（数据库连接）、save_to_mongodb（数据插入）、export_data（数据导出）方法，实现 MongoDB 的连接、索引创建、数据持久化与格式转换，满足数据存储与复用需求。

日志与监控模块：通过 logging 模块配置文件日志与控制台日志，实时记录爬取状态、错误信息，结合 get_statistics 方法提供爬取统计数据，便于问题排查与进度跟踪。

（二）技术选型依据

网络请求：选用 requests 库而非 urllib/urllib3，因其 API 简洁易用，支持 Session 保持连接、异常捕获更完善，能高效实现请求发送、重试与响应处理。

页面解析：采用 BeautifulSoup 库解析 HTML，其 CSS 选择器与标签遍历功能灵活，可适配校内网站多样的页面结构，相比正则表达式更易维护。

数据库：选择 MongoDB 而非 MySQL/Redis，因新闻、通知类数据属于半结构化信息，MongoDB 的文档型存储无需预设固定表结构，支持灵活扩展字段，且索引功能满足去重与检索需求。

辅助工具：使用 urljoin 处理相对 URL，datetime 记录爬取时间，re 提取日期信息，确保数据处理的准确性与规范性。

（三）核心流程设计

初始化阶段：创建爬虫实例，配置请求头（模拟浏览器）、初始化 Session 连接，建立 MongoDB 连接并创建索引，完成日志配置。

爬取阶段：遍历预设的校内网新闻版块→针对每个版块分页请求页面→提取页面中的新闻链接（去重处理）→逐个请求新闻详情页→解析目标字段生成结构化数据。

存储与后续阶段：将结构化数据插入 MongoDB（自动去重）→爬取完成后统计数据总量、最新爬取信息→支持用户选择是否导出 JSON 格式数据→关闭数据库与网络连接，释放资源。

（四）关键问题解决方案

页面结构适配：针对校内不同版块页面结构差异，设计多组 CSS 选择器（如标题选择器、内容选择器），依次匹配直至提取到有效数据，提升解析成功率。

反爬虫应对：设置请求延迟、使用标准 User-Agent、通过 Session 保持连接，模拟正常用户访问行为；避免高频请求，降低服务器封禁风险。

异常处理：捕获数据库连接失败、网络请求超时、重复数据插入等异常，通过日志记录错误信息，确保程序不崩溃，仅跳过无效条目。

分页处理：通过拼接分页 URL（?page=N）与检测 “下一页” 按钮元素结合的方式，实现自动分页爬取，避免遗漏多页数据。

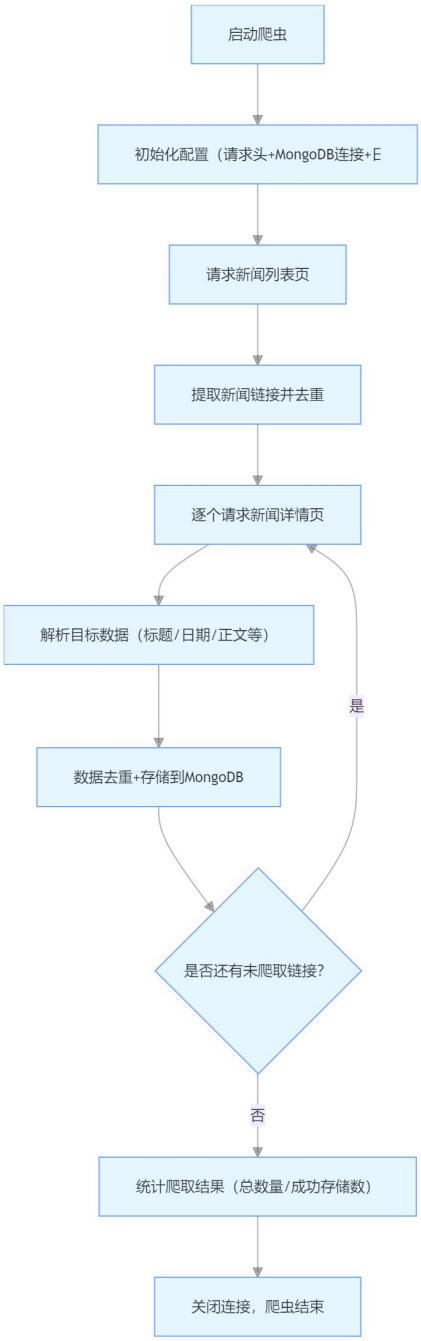

图 1-流程图

三、详细设计

任务一：

1.网络请求的 Network 抓取并分析，请求方法为 GET

图 2-请求方法

2.页面结构的 HTML 分析

图 3-HTML 结构

3.代码编写

导入所需库，如：requests、BeautifulSoup、time 等等

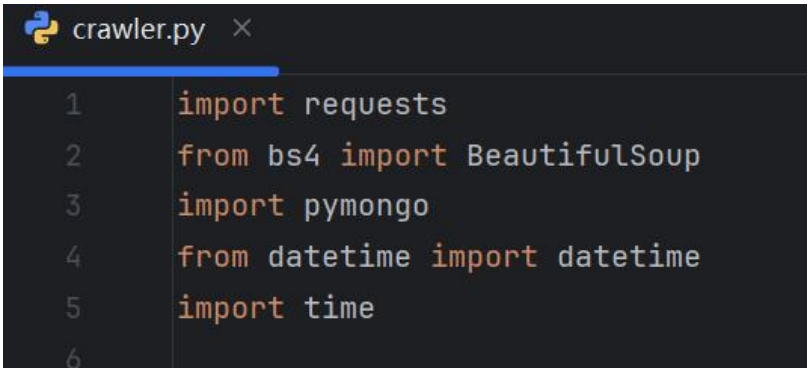

图 4-python 库

进行基础配置（数据库存储使用 MongoDB；配置请求头和对应的 URL）

```
BASE_URL = "https://www.ky.zstu.edu.cn"
# 新闻列表页URL（学院新闻板块）
NEWS_LIST_URL = "https://www.ky.zstu.edu.cn/kyyw.htm"
# MongoDB配置
MONGO_HOST = "localhost"  # 本地MongoDB
MONGO_PORT = 27017  # 默认端口
MONGO_DB = "zstu_news"  # 数据库名
MONGO_COLLECTION = "school_news"  # 集合名
# 请求头
HEADERS = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36 SLBrowse
}
```

图 5-基础配置

创建数据库，进行 MongoDB 的连接，并创建集合

```
def connect_mongo():  1 个用法
    try:
        # 建立连接
        client = pymongo.MongoClient(MONGO_HOST, MONGO_PORT)
        # 创建数据库
        db = client[MONGO_DB]
        # 创建集合
        collection = db[MONGO_COLLECTION]
        print("MongoDB连接成功！")
        return collection
    except Exception as e:
        print(f"MongoDB连接失败：{e}")
        return None
```

图 6 数据库连接

解析新闻列表页，搜索根容器（“wlistee”）

```
def parse_news_list(list_url):  1 个用法
    news_list = []
    try:
        response = requests.get(list_url, headers=HEADERS, timeout=10)
        response.encoding = response.apparent_encoding
        soup = BeautifulSoup(response.text, features: "lxml")

        # 1. 找到新闻列表的根容器
        wlistee_div = soup.find("div", class_="wlistee")
        if not wlistee_div:
            print("未找到新闻列表的根容器（div.wlistee）")
            return news_list
```

图 7-寻找节点

匹配所有对应的新闻节点（“li”）

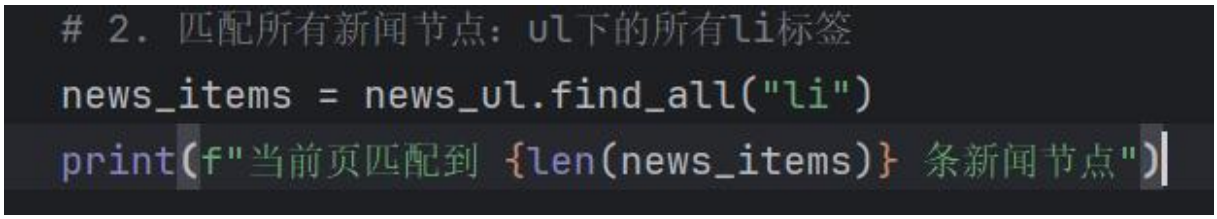

图 8-匹配节点

遍历每个 li 标签，进行数据提取

```python
# 3. 遍历每个li，提取数据
for item in news_items:
    # 提取a标签
    a_tag = item.find("a")
    if not a_tag:
        continue

    # 提取标题
    title = a_tag.get_text(strip=True)

    # 提取详情页URL
    detail_url = a_tag.get("href")
    if detail_url.startswith("#"):
        detail_url = detail_url[1:]
    # 拼接完整URL
    if not detail_url.startswith(("http://", "https://")):
        detail_url = BASE_URL + "/" + detail_url.lstrip("/")

    # 提取发布时间（li内的span标签文本）
    time_span = item.find("span")
    publish_time = time_span.get_text(strip=True) if time_span else "未知时间"

    # 存储单条新闻信息
    news_list.append({
        "title": title,
        "detail_url": detail_url,
        "publish_time": publish_time
    })
    print(f"已获取新闻：{title}")
```

图 9-遍历节点

解析详情页，发送请求并提取正文

```python
#解析新闻详情页
def parse_news_detail(detail_url):  1个用法
    content = ""
    try:
        # 发送请求
        time.sleep(1)  # 间隔1秒
        response = requests.get(detail_url, headers=HEADERS, timeout=10)
        response.encoding = response.apparent_encoding
        soup = BeautifulSoup(response.text, features: "lxml")
        # 提取正文
        content_tag = soup.find("div", class_="article-content")
        if content_tag:
            content = content_tag.get_text(strip=True, separator="\n")
        else:
            content_tag = soup.find("div", class_="content")
            if content_tag:
                content = content_tag.get_text(strip=True, separator="\n")
            else:
                content = "未获取到正文"
    except Exception as e:
        print(f"详情页爬取失败（{detail_url}）：{e}")
    return content
```

图 10-解析 HTML

连接 MongoDB，并爬取网页数据

```python
def main():  1个用法
    # 1. 连接MongoDB
    collection = connect_mongo()
    if not collection:
        return

    # 2. 爬取新闻列表
    news_list = parse_news_list(NEWS_LIST_URL)
    if not news_list:
        print("未获取到新闻数据，爬取终止")
        return
```

图 11-爬取网页

存储数据到 MongoDB

```python
# 3. 遍历新闻列表，爬取详情并存储到MongoDB
success_count = 0
for news in news_list:
    # 爬取正文
    content = parse_news_detail(news["detail_url"])
    full_news = {
        "title": news["title"],
        "detail_url": news["detail_url"],
        "publish_time": news["publish_time"],
        "content": content,
        "crawl_time": datetime.now().strftime("%Y-%m-%d %H:%M:%S")  # 爬取时间（格式化）
    }

    # 存储到MongoDB
    if collection.find_one({"detail_url": news["detail_url"]}):
        print(f"新闻已存在：{news['title']}，跳过存储")
        continue

    # 插入数据
    collection.insert_one(full_news)
    success_count += 1
    print(f"存储成功：{news['title']}")
```

图 12-数据库存储

输出到控制台完成爬虫统计

```python
    # 4. 爬取完成统计
    print("=" * 50)
    print(f"爬取任务完成！")
    print(f"总获取新闻数：{len(news_list)}")
    print(f"成功存储到MongoDB数：{success_count}")
    print("=" * 50)


#启动爬虫程序
if __name__ == "__main__":
    main()
```

图 13-爬取完成

成功存储到 MongoDB，爬取数据如图下所示：

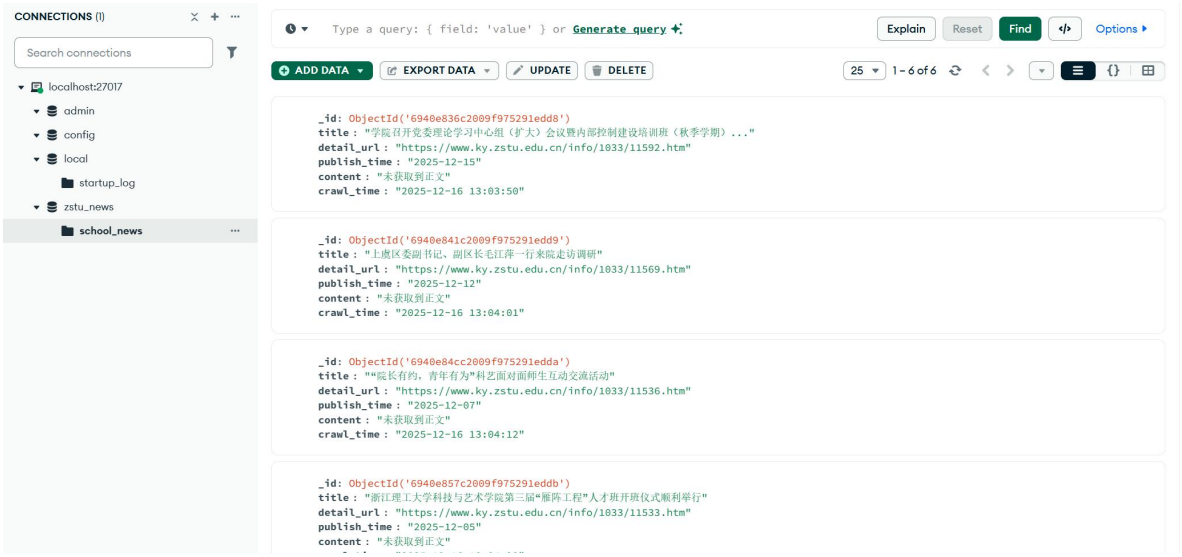

图 14-存储成功

任务二：

一、Request 请求的常见状态码及含义

状态码	类别	含义描述
100	信息性状态码	继续，服务器已接收请求头部，客户端可继续发送请求体
200	成功状态码	请求成功，服务器正常返回目标资源
201	成功状态码	资源创建成功（常用于 POST 请求，如提交表单后创建新数据）

301	重定向状态码	永久重定向，目标资源已永久迁移至新 URL，后续请求应使用新地址
302	重定向状态码	临时重定向，目标资源临时迁移，后续请求仍可使用原 URL
304	重定向状态码	协商缓存命中，资源未修改，服务器无需返回完整数据（提升访问效率）
400	客户端错误码	Bad Request，请求参数错误或格式不规范，服务器无法处理
401	客户端错误码	未授权，请求需要用户身份验证（如登录后才能访问的资源）
403	客户端错误码	禁止访问，服务器理解请求但拒绝提供服务（如无权限访问的页面）
404	客户端错误码	资源未找到，服务器无法找到请求的 URL（可能是地址错误或资源已删除）
405	客户端错误码	方法不允许，请求使用的 HTTP 方法（如 POST）在目标资源上不被支持
500	服务器错误码	内部服务器错误，服务器处理请求时发生未知异常
502	服务器错误码	网关错误，服务器作为网关时，收到上游服务器的

		无效响应
503	服务器错误码	服务不可用，服务器暂时过载或维护中，无法处理请求
504	服务器错误码	网关超时，服务器作为网关时，等待上游服务器响应超时

二、Request 请求的标准 HTTP 方法

HTTP 方法	核心作用	特点说明
GET	获取资源，请求服务器返回指定 URL 的资源	幂等（多次请求结果一致）、可缓存、请求参数拼接在 URL 后（长度有限制）
POST	提交资源，向服务器发送数据以创建或修改资源（如表单提交、文件上传）	非幂等、不可缓存、请求参数放在请求体中（支持大量数据、多种格式）
PUT	全量更新资源，用请求体中的数据替换目标资源的全部内容	幂等、可缓存（需配置）、适用于完整替换资源场景（如更新用户全部信息）
DELETE	删除资源，请求服务器删除指定 URL 对应的资源	幂等、不可缓存、常用于删除数据（如删除订单、删除用户账号）
PATCH	部分更新资源，仅修改目标资源的部分字段（无需传递完整资源信息）	非幂等、灵活性高，适用于增量更新场景（如修改用户手机号）
HEAD	获取资源头部信息，与 GET 功能一致，但仅返回	幂等、可缓存、用于快速获取资源元数据（如文件

	响应头，不返回响应体	大小、修改时间）
OPTIONS	预检请求，获取服务器支持的 HTTP 方法、请求头类型等信息	常用于跨域请求预检（CORS），确认客户端是否有权限发起目标请求
TRACE	回显请求，服务器将收到的请求原样返回，用于调试请求传输过程是否有修改	安全性较低，部分服务器会禁用，实际开发中极少使用

三、URL 输入后从 DNS 解析到页面渲染的完整流程（含协议）

步骤顺序	关键步骤	涉及协议	详细说明
1	URL 解析	-	浏览器拆分 URL（协议：HTTPS；域名：www.ky.zstu.edu.cn；路径：/news/等）
2	DNS 域名解析	DNS 协议（UDP/TCP 53 端口）	1. 浏览器查询本地 DNS 缓存；2. 无缓存则查询路由器 DNS；3. 再查询 ISP DNS 服务器；4. 递归查询根域名服务器→顶级域名服务器→目标域名服务器，最终获取 IP 地址
3	建立 TCP 连接	TCP/IP 协议	基于获取的 IP 地址，与服务器进行“三次握手”：客户端发 SYN→服务器回

			SYN+ACK→客户端发 ACK，建立可靠连接
4	HTTPS 协议握手（若为 HTTPS）	TLS/SSL 协议	1. 服务器发送证书（含公钥）；2. 客户端验证证书有效性；3. 客户端生成随机密钥，用公钥加密发送给服务器；4. 双方协商会话密钥，后续通信数据用该密钥加密
5	发送 HTTP 请求	HTTP/HTTPS 协议	浏览器向服务器发送请求（含请求行、请求头、请求体），如 GET /news/ HTTP/1.1
6	服务器处理请求	-	服务器接收请求后，解析参数、查询数据库/读取文件，生成响应数据
7	服务器返回响应	HTTP/HTTPS 协议	服务器向客户端返回响应（含响应行、响应头、响应体，响应体为 HTML/CSS/JS 等资源）
8	关闭 TCP 连接	TCP/IP 协议	数据传输完成后，双方进行“四次挥手”：客户端发 FIN→服务器回 ACK→服务器发

			FIN→客户端回ACK，释放连接
9	浏览器解析资源	-	1. 解析HTML生成DOM 树（文档对象模型）；2. 解析CSS 生成 CSSOM 树（CSS 对象模型）；3. 合并 DOM 树与 CSSOM 树生成渲染树
10	页面布局与绘制	-	1. 布局（Layout）：计算渲染树中元素的位置、大小；2. 绘制（Paint）：根据布局结果绘制像素到屏幕；3. 合成（Composite）：将绘制的图层合成最终页面并显示

四、常用请求头（Request Headers）及其作用

请求头名称	核心作用	示例值
User-Agent	标识客户端类型（浏览器/设备/爬虫），服务器据此返回适配资源	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Chrome/91.0.4472.124 Safari/537.36
Accept	告知服务器客户端可接收的资源格式（如 HTML、JSON、图片等）	text/html,application/xhtml+xml,application/json;q=0.9,/;q=0.8
Accept-Language	告知服务器客户端偏好的	zh-CN,zh;q=0.9,en;q=0.8

	语言（如中文、英文），用于国际化资源返回	
Accept-Encoding	告知服务器客户端支持的压缩格式（如 gzip、deflate），减少传输数据量	gzip, deflate, br
Cookie	携带客户端存储的会话信息（如登录状态、用户偏好），维持会话一致性	JSESSIONID=abc123xyz; username=testuser
Referer	标识当前请求的来源页面 URL，服务器可用于防盗链、统计来源	https://www.ky.zstu.edu.cn/
Content-Type	告知服务器请求体的数据格式（仅 POST/PUT/PATCH 请求需配置）	application/json; charset=utf-8（JSON 格式）、application/x-www-form-urlencoded（表单格式）
Connection	声明 TCP 连接状态（keep-alive：长连接；close：短连接）	keep-alive
Host	指定目标服务器的域名和端口（HTTP/1.1 必需字段），用于服务器区分多域名站点	www.ky.zstu.edu.cn:443
Cache-Control	控制请求的缓存策略（如 no-cache：不使用缓存；max-age：缓存有效时间）	no-cache

五、HTTP 与 HTTPS 的主要区别

对比维度	HTTP（超文本传输协议）	HTTPS（超文本传输安全协议）
核心本质	无加密的明文传输协议	HTTP + TLS/SSL 加密层的安全传输协议
安全性	明文传输，数据易被窃听、篡改、伪造（无身份验证和数据完整性保障）	数据加密（对称加密）、身份验证（服务器证书）、数据完整性校验（哈希算法），安全性高
端口号	默认使用 80 端口	默认使用 443 端口
证书要求	无需证书	需向权威 CA 机构申请 SSL 证书（免费/付费），否则浏览器会提示“不安全”
传输速度	无加密/解密步骤，传输速度较快	需额外进行 TLS 握手和数据加密/解密，传输速度略慢（开销小，用户无明显感知）
数据传输	数据以明文形式在网络中传输	数据通过会话密钥加密后传输，仅客户端和服务器可解密
适用场景	非敏感数据传输（如公开新闻、静态资源）	敏感数据传输（如登录、支付、个人信息、教务系统数据）
协议头	简单，无额外加密相关字段	包含 TLS 相关字段（如 SSL 证书信息、加密套件标识）

六、Python 中 urllib、urllib3、requests 三个库的差异对比

对比维度	urllib	urllib3	requests
所属类型	Python 标准库（无需额外安装）	第三方库（需 pip install urllib3）	第三方库（需 pip install requests）
API 易用性	接口繁琐（如 GET 请求需构造 Request 对象，POST 需编码数据），学习成本高	接口较简洁，支持连接池、超时设置，但仍需手动处理部分细节（如 JSON 解析）	高层封装，API 极简（如 requests.get()、requests.post()），人性化设计，学习成本低
功能特性	基础 HTTP 请求（GET/POST）、URL 编码/解码、Cookie 处理（需手动管理）	支持连接池复用、线程安全、超时控制、重试机制、SSL 验证，功能比 urllib 丰富	集成 urllib3 的核心功能，支持自动 Cookie 管理、会话保持、JSON 自动解析、文件上传下载，功能最全面
并发支持	无原生并发支持，需结合 threading 手动实现	线程安全，支持多线程并发请求，连接池可提升并发效率	本身无并发能力，需结合多线程/多进程或异步库（如 aiohttp），但单线程使用体验最佳
依赖情况	无外部依赖（Python 自带）	无强依赖，纯 Python 实现	依赖 urllib3 和 chardet（自动安装）
适用场景	简单请求场景、Python 环境受限（无法安装第三方库）时	底层开发、需要自定义连接池或并发控制的场景	绝大多数爬虫开发、API 调用场景（推荐首选）

七、编写基础爬虫常用的第三方库及作用

库名称	核心作用	应用场景
requests	发送 HTTP/HTTPS 请求（GET/POST/PUT 等），自动处理 Cookie、编码转换、响应解析	爬虫核心库，获取网页 HTML、API 接口数据
BeautifulSoup4	解析 HTML/XML 文档，提取目标数据（如标签内容、属性值），支持 CSS 选择器和正则	网页数据解析（如提取新闻标题、链接、正文）
lxml	高性能 HTML/XML 解析库（基于 C 语言），支持 XPath 和 CSS 选择器	替代 BeautifulSoup4 的解析后端，提升解析速度（需配合 BeautifulSoup 使用）
Scrapy	专业爬虫框架，集成请求发送、数据解析、数据存储、反爬虫应对等功能	大规模、高性能爬虫开发（如全站点爬取、分布式爬取）
pymongo/pymysql	连接 MongoDB/MySQL 数据库，实现数据的增删改查操作	爬虫数据持久化存储（如将爬取的新闻、通知存入数据库）
redis-py	连接 Redis 数据库，实现缓存、队列、去重等功能	爬虫去重（存储已爬取 URL）、任务队列（管理待爬取链接）
fake-useragent	生成随机 User-Agent，模拟不同浏览器/设备请求	应对服务器的 User-Agent 检测反爬虫策略
selenium	模拟浏览器行为（如点击、输入、页面渲染），支持动态 JavaScript 页面	爬取动态渲染页面（如 JavaScript 加载的新闻列表、需要登录的教务系统）
Pyppeteer	无头浏览器库（无界面 Chrome），功能类似	动态页面爬取，支持异步操作，比 selenium 更轻量

	selenium，性能更优	化
requests-proxies	为 requests 请求配置代理 IP，隐藏真实 IP 地址	应对服务器的 IP 封禁反爬虫策略

八、Cookie 与 Session 的区别（多维度对比）

对比维度	Cookie	Session
存储位置	客户端（浏览器/本地文件）	服务器端（内存/数据库/Redis）
存储容量	容量限制小（单个 Cookie≤4KB，同一域名最多 20-50 个）	容量无限制（取决于服务器存储资源）
生命周期	可设置过期时间（永久 Cookie：Expires 字段；会话 Cookie：关闭浏览器失效）	默认会话级（关闭浏览器/服务器超时后失效），可手动设置过期时间
安全性	明文存储（可修改），安全性低，仅能存储非敏感信息	存储在服务器，客户端仅持有 SessionID（随机字符串），安全性高，可存储敏感信息
传递方式	每次 HTTP 请求自动通过请求头 Cookie 字段传递给服务器	客户端通过 Cookie 或 URL 参数传递 SessionID，服务器通过 SessionID 查询对应数据
存储内容	仅能存储字符串类型数据	可存储任意类型数据（如字典、对象、列表）
适用场景	存储用户偏好（如主题设置、语言选择）、记住登	存储用户登录信息、会话数据（如购物车、临时操

	录状态（免登录天数）	作记录）
服务器压力	无服务器存储压力（数据在客户端）	大量用户会话会占用服务器资源，需合理设置超时时间和清理策略

九、BeautifulSoup 解析 HTML 的常用方法及功能说明

方法名称	功能描述	用法示例
find()	查找符合条件的**第一个**标签元素，支持按标签名、属性、文本内容筛选	soup.find('h1', class_='news-title') # 查找 class 为 news-title 的第一个 h1 标签
find_all()	查找符合条件的**所有**标签元素，返回列表，支持 limit 参数限制返回数量	soup.find_all('a', href=True) # 查找所有带 href 属性的 a 标签
select()	按 CSS 选择器查找元素，返回列表（支持 id、class、层级选择器）	soup.select('#content .article-p') # 查找 id 为 content 下 class 为 article-p 的标签
select_one()	按 CSS 选择器查找**第一个**符合条件的元素	soup.select_one('div.publish-date') # 查找第一个 class 为 publish-date 的 div 标签
get_text()	提取标签内的文本内容，支持 strip 参数去除首尾空格、separator 参数设置分隔符	tag.get_text(strip=True, separator='\n') # 提取文本并去除空格，换行分隔
get()	获取标签的属性值（如 href、src、class），无该	a_tag.get('href') # 获取 a 标签的链接地址

	属性则返回 None	
find_parent()	查找当前标签的父级标签，支持条件筛选	title_tag.find_parent('div', class_='news-header') # 查找标题的父级新闻头部标签
find_next_sibling()	查找当前标签的下一个同级标签	p_tag.find_next_sibling('p') # 查找当前 p 标签的下一个 p 标签
decompose()	删除标签（如移除 script、style 等冗余标签，净化 HTML 结构）	for script in soup.find_all('script'): script.decompose() # 删除所有 script 标签

十、提升爬虫抓取速度的几种常用手段

1. **多线程/多进程并发爬取**：利用 Python 的 `threading`（多线程）或 `multiprocessing`（多进程）库，同时发起多个请求，突破单线程串行爬取的速度限制（适用于 IO 密集型的爬虫场景）。

2. **异步请求（Async）**：使用 `aiohttp` 库替代 `requests`，通过非阻塞 IO 实现高并发请求，单个线程可同时处理上百个请求，效率远超多线程（推荐大规模爬取使用）。

3. **连接池复用**：使用 `urllib3` 或 `requests` 的连接池功能，避免每次请求都重新建立 TCP 连接，减少连接建立/关闭的开销（默认开启，可手动调整池大小）。

4. **合理控制并发数**：根据目标服务器的承载能力设置并发数（如 10-50 个/秒），避免并发过高导致 IP 被封，同时确保资源利用率最大化。

5. **数据批量存储**：爬取数据时先缓存到内存（如列表、字典），积累一定数量后批量插入数据库，减少数据库连接和 IO 操作次数（如每 100 条数据批量存入 MongoDB）。

6. **避免重复爬取**：使用 Redis、布隆过滤器（`bloom-filter-py`）或数据库索引存储已爬取的 URL，避免重复请求同一资源（尤其全站点爬取时关键）。

7. **优化解析逻辑**：优先使用 `lxml` 解析器（比 BeautifulSoup 默认解析器快 10 倍以上），减少不必要的标签遍历和正则匹配，简化解析代码。

8. **禁用不必要的功能**：关闭请求的缓存、SSL 验证（仅非敏感站点），减少请求过程中的额外开销（如 `requests.get(url, verify=False)`）。

十一、常见的反爬虫策略及应对思路

反爬虫策略	应对思路
User-Agent 检测（仅允许浏览器访问）	1. 收集真实浏览器的 User-Agent 列表；2. 使用 `fake-useragent` 库随机生成 User-Agent；3. 定期更新 User-Agent 池
IP 封禁/频率限制（单 IP 请求过快被封）	1. 控制爬取速度（添加随机延迟：`time.sleep(random.uniform(1,3))`）；2. 使用代理 IP 池（如阿布云、快代理）；3. 分布式爬取（多 IP 节点分担请求）
验证码验证（高频请求后弹出验证码）	1. 降低请求频率，避免触发验证码；2. 使用打码平台（如云打码、超级鹰）自动识别验证码；3. 手动输入验证码（小规模爬取）
动态页面渲染（JavaScript 加载数据）	1. 使用 `selenium` 或 `Pyppeteer` 模拟浏览器渲染；2. 分析网络请求（F12 Network），直接调用后端 API 接口获取 JSON 数据
robots 协议限制（禁止爬虫访问部分页面）	1. 遵守 robots 协议（爬虫伦理）；2. 若需爬取，修改爬虫的 User-Agent，避开协议检测（谨慎使用）
数据加密（页面数据通过 JS 加密后展示）	1. 查看页面 JS 代码，分析加密逻辑（如 AES、MD5 加密）；

	2. 模拟 JS 加密过程，用 Python 实现解密；3. 用 `execjs` 库直接调用 JS 解密函数
Cookie 验证（需登录态或特定 Cookie 才能访问）	1. 模拟登录（通过 requests.Session 保存登录 Cookie）；2. 手动导出浏览器 Cookie，添加到请求头；3. 处理 Cookie 过期问题（定期重新登录）
请求头完整性检测（验证 Referer、Host 等字段）	1. 模拟真实浏览器请求头，补全所有必要字段（如 Referer 设为目标站点域名）；2. 保持请求头格式一致性，不随意修改
动态 URL（URL 含随机参数或时效参数）	1. 从页面 HTML 或 JS 中提取真实请求 URL；2. 分析 URL 参数生成逻辑，动态构造有效 URL
数据分页限制（仅允许访问前 N 页）	1. 查找分页 API 的参数规律（如 `page=1`→`page=100`）；2. 切换不同 IP 或账号访问更多页面

四、心得体会

（一）实验收获

技术能力提升：通过本次实验，深入掌握了 requests 库的网络请求技巧、BeautifulSoup 的页面解析方法，以及 MongoDB 的连接、索引创建与数据操作，清晰理解了 urllib、urllib3 与 requests 的差异 ——requests 的高层 API 确实能大幅简化代码

编写，提升开发效率。同时，对 HTTP 协议的应用场景有了更直观的认识，比如请求状态码、请求头在爬虫中的实际作用。

爬虫逻辑梳理：明确了 “请求 - 解析 - 存储” 的爬虫核心流程，学会了将复杂需求拆解为模块化功能，例如将爬取过程拆分为页面请求、链接提取、详情解析、数据存储等步骤，每个模块单独实现后再整合，大幅降低了开发难度。

合规与细节意识：深刻认识到爬虫开发需兼顾合规性与细节处理 —— 设置请求延迟、尊重网站 robots 协议是爬虫开发的基本准则；而处理相对 URL 转换、日期格式提取、数据去重等细节，直接决定了爬取数据的质量与程序的稳定性。

（二）问题与解决

解析失败问题：初期部分新闻页面因选择器不匹配导致内容提取失败，通过分析目标网站 HTML 结构，补充多组适配不同版块的选择器，并增加内容长度校验（≥50 字），确保解析结果有效。

数据库连接异常：曾出现 MongoDB 连接超时问题，通过添加 serverSelectionTimeoutMS 参数设置超时时间，增加连接测试（admin.command('ping')），并在日志中记录连接状态，快速定位并解决了数据库服务未启动的问题。

重复爬取问题：首次测试时出现重复数据，通过在 MongoDB 中创建 URL 唯一索引，结合内存中 “已爬取链接集合” 的临时存储，双重保障实现了数据去重，避免了冗余存储。

（三）改进方向与反思

本次爬虫实现了官网新闻、通知版块的爬取，针对不同站点的登录验证（如教务系统的身份认证）、动态页面（如 JavaScript 渲 k 染内容），引入 Selenium 或 Playwright 工具适配；同时，可优化爬取速度，通过多线程或异步请求（aiohttp）替代单线程，在合规范围内提升爬取效率。

此次实验让我明白，爬虫开发不仅是技术的应用，更是问题解决能力的体现 —— 从需求分析到架构设计，再到细节调试，每一步都需要结合实际场景灵活调整。未来开发中，需更注重代码的可扩展性与鲁棒性，同时持续关注网络爬虫的合规性要求，做到技术应用与规范遵守并重。

五、核心代码

```
import requests

from bs4 import BeautifulSoup

import pymongo

from datetime import datetime

import time


#基础配置（数据库存储使用 MongoDB）

# 官网域名（基础 URL）

BASE_URL = "https://www.ky.zstu.edu.cn"

# 新闻列表页 URL（学院新闻板块）

NEWS_LIST_URL = "https://www.ky.zstu.edu.cn/kyyw.htm"

# MongoDB 配置

MONGO_HOST = "localhost"  # 本地 MongoDB

MONGO_PORT = 27017  # 默认端口

MONGO_DB = "zstu_news"  # 数据库名

MONGO_COLLECTION = "school_news"  # 集合名

# 请求头

HEADERS = {

    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36 SLBrowser/9.0.6.8151 SLBChan/112 SLBVPV/64-bit"
```

```
}



# MongoDB 连接
def connect_mongo():
    try:
        #  建立连接
        client = pymongo.MongoClient(MONGO_HOST, MONGO_PORT)
        #  创建数据库
        db = client[MONGO_DB]
        #  创建集合
        collection = db[MONGO_COLLECTION]
        print("MongoDB 连接成功！")
        return collection
    except Exception as e:
        print(f"MongoDB 连接失败：{e}")
        return None



#解析新闻列表页
def parse_news_list(list_url):
    news_list = []
```

```
try:
    response = requests.get(list_url, headers=HEADERS, timeout=10)
    response.encoding = response.apparent_encoding
    soup = BeautifulSoup(response.text, "lxml")

    # 1. 找到新闻列表的根容器
    wlistee_div = soup.find("div", class_="wlistee")
    if not wlistee_div:
        print("未找到新闻列表的根容器（div.wlistee）")
        return news_list

    news_ul = wlistee_div.find("ul")  # 找到 ul 标签（新闻列表的父容器）
    if not news_ul:
        print("div.wlistee 内未找到 ul 标签")
        return news_list

    # 2. 匹配所有新闻节点：ul 下的所有 li 标签
    news_items = news_ul.find_all("li")  # 这一步会匹配到所有新闻（截图中至少有 4 条）
    print(f"当前页匹配到 {len(news_items)} 条新闻节点")  # 调试：确认数量

    # 3. 遍历每个 li，提取数据
    for item in news_items:
```

```
            # 提取 a 标签（标题+详情页 URL）
            a_tag = item.find("a")
            if not a_tag:
                continue  # 跳过无 a 标签的异常项

            # 提取标题（a 标签的文本）
            title = a_tag.get_text(strip=True)

            # 提取详情页 URL：去掉开头的#，再拼接 BASE_URL
            detail_url = a_tag.get("href")
            if detail_url.startswith("#"):
                detail_url = detail_url[1:]
            # 拼接完整 URL（兼容所有相对路径）
            if not detail_url.startswith(("http://", "https://")):
                detail_url = BASE_URL + "/" + detail_url.lstrip("/")

            # 提取发布时间（li 内的 span 标签文本）
            time_span = item.find("span")
            publish_time = time_span.get_text(strip=True) if time_span else "未知时
间"

            # 存储单条新闻信息
            news_list.append({
```

```
                "title": title,
                "detail_url": detail_url,
                "publish_time": publish_time
            })
            print(f"已获取新闻：{title}")

    except Exception as e:
        print(f"列表页爬取报错：{str(e)}")
    return news_list


#解析新闻详情页
def parse_news_detail(detail_url):
    content = ""
    try:
        # 发送请求
        time.sleep(1)  # 间隔 1 秒
        response = requests.get(detail_url, headers=HEADERS, timeout=10)
        response.encoding = response.apparent_encoding
        soup = BeautifulSoup(response.text, "lxml")
        # 提取正文
        content_tag = soup.find("div", class_="article-content")
```

```
            if content_tag:
                content = content_tag.get_text(strip=True, separator="\n")
            else:
                content_tag = soup.find("div", class_="content")
                if content_tag:
                    content = content_tag.get_text(strip=True, separator="\n")
                else:
                    content = "未获取到正文"
        except Exception as e:
            print(f"详情页爬取失败（{detail_url}）：{e}")
        return content


#爬取和存储
def main():
    # 1. 连接 MongoDB
    collection = connect_mongo()
    if not collection:
        return

    # 2. 爬取新闻列表
    news_list = parse_news_list(NEWS_LIST_URL)
```

```
    if not news_list:
        print("未获取到新闻数据，爬取终止")
        return

    # 3. 遍历新闻列表，爬取详情并存储到 MongoDB
    success_count = 0
    for news in news_list:
        # 爬取正文
        content = parse_news_detail(news["detail_url"])
        full_news = {
            "title": news["title"],
            "detail_url": news["detail_url"],
            "publish_time": news["publish_time"],
            "content": content,
            "crawl_time":  datetime.now().strftime("%Y-%m-%d  %H:%M:%S")     # 爬取时间（格式化）
        }

        # 存储到 MongoDB
        if collection.find_one({"detail_url": news["detail_url"]}):
            print(f"新闻已存在：{news['title']}，跳过存储")
            continue
```

```
            # 插入数据
            collection.insert_one(full_news)
            success_count += 1
            print(f"存储成功：{news['title']}")

    # 4. 爬取完成统计
    print("=" * 50)
    print(f"爬取任务完成！")
    print(f"总获取新闻数：{len(news_list)}")
    print(f"成功存储到 MongoDB 数：{success_count}")
    print("=" * 50)

#启动爬虫程序
if __name__ == "__main__":
    main()
```

指导老师评价：					
项目评价	**优**	**良**	**中**	**一般**	**较差**
课堂纪律：按时上课，无迟到早退					
完成态度：认真完成大作业设计，勤奋好学，积极思考					
项目完成度：按照要求完成项目设计					
课程质量：代码正确，按时完成，界面美观					
课程总结：思路清晰，调理清楚，全面总结了心得体会和不足					
总评成绩	**签名：** **时间：**				