Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Информационный поиск»

Студент: М. М. Сисенов
Преподаватель: А. А. Кухтичев
Группа: М8О-410Б
Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №2 «Поисковый робот»

Необходимо написать парсер на любом языке программирования:

- Написать поисковый робот — компоненты обкачки документов, используя любой язык программирования.
- Единственным аргументом поисковому роботу подаётся файл конфигурации (формата YAML или JSON), в котором содержатся параметры работы программы.
- База данных должна быть запущена в docker-контейнере, можно использовать docker-compose.
- В качестве хранилища результатов использовать MongoDB или PostgreSQL.
- Робот должен применять нормализацию URL-адресов.
- Робот должен сохранять в БД сырой HTML документа.
- Необходимо сохранять метаинформацию о каждом документе (дата скачивания, источник URL и т.д.).
- При остановке работы робот должен сохранять контрольную точку так, чтобы при повторном запуске он мог продолжить работу с того документа, с которого он остановился.
- Периодически он должен уметь переобкачивать документы, которые уже есть в базе, но только в том случае, если они изменились.

Задание

Требуется реализовать веб-краулер (поисковый робот), который автоматически собирает документы с психологических порталов **b17.ru** и **psychologies.ru**, выбранных в лабораторной работе №1. Робот должен сохранять полный HTML-контент страниц в базу данных MongoDB вместе с метаинформацией, обеспечивать возможность остановки/возобновления работы и отслеживать изменения документов.

Выбор языка программирования и технологий

Для реализации поискового робота был выбран язык программирования **Python**

Стек технологий:

- **Язык**: Python 3.12.3
- **База данных**: MongoDB 8.x
- **HTTP-клиент**: `requests`
- **HTML-парсинг**: `BeautifulSoup4` (библиотека bs4)
- **Работа с БД**: `pymongo`
- **Конфигурация**: YAML (`PyYAML`)

Схема работы

1. **Инициализация**: Чтение конфигурации из YAML-файла, подключение к MongoDB и создание уникальных индексов для URL.
2. **Управление состоянием**: Использование JSON-файла для хранения номеров страниц, на которых остановился сборщик ссылок (Harvester).
3. **Этап 1: Сбор ссылок (Harvesting)**:
    - Обход страниц-списков в разделах статей;
    - Извлечение и нормализация URL;
    - Проверка необходимости обкачки (отсутствие в базе или истечение заданного интервала времени);
    - Добавление уникальных ссылок в коллекцию-очередь задач MongoDB.
4. **Этап 2: Обкачка контента (Crawling)**:
    - Извлечение задач из очереди;
    - Загрузка HTML-контента с соблюдением задержек;
    - Извлечение очищенного текста и заголовка.
5. **Отслеживание изменений**: Сравнение нового очищенного текста с версией из базы данных. Если текст не изменился, обновляется только метка времени обкачки.
6. **Сохранение**: Запись документов с полями URL, источник, заголовок, «сырой» HTML, очищенный текст и время обкачки в формате Unix timestamp.

Реализация

1. Конфигурационный файл (config.yaml)

```yaml
db:
  host: "localhost"
  port: 27017
  name: "search_engine_db"
  collection_articles: "articles_v5"
  collection_queue: "task_queue"

logic:
  delay_min: 1.0
  delay_max: 3.0
  recrawl_interval: 86400
  state_file: "crawler_state.json"
```

2. Структура документа в MongoDB

```python
doc = {
    'url': url,
    'source': source,
    'title': title,
    'raw_html': html,
    'clean_text': text,
    'timestamp': time.time()
}
```

Пояснение полей:

- `url` — нормализованный URL документа
- `source` — название ресурса
- `title` — Название статьи
- `raw_html` — полный HTML-контент страницы
- `clean_text` — очищенный полезный текст
- `timestamp` — timestamp скачивания (Unix-время)

3. Нормализация URL

```python
def normalize_url(url):
    parsed = urlparse(url)
    clean_url = urlunparse((parsed.scheme.lower(), parsed.netloc.lower
       (), parsed.path, '', '', ''))
    if clean_url.endswith('/'):
        clean_url = clean_url[:-1]
    return clean_url
```

При запуске краулер проверяет наличие checkpoint.json и продолжает работу с последнего обработанного URL.

Исходный код

```
import requests
from bs4 import BeautifulSoup
from pymongo import MongoClient
import time
import random
import json
import os
import sys
import yaml
import argparse
from urllib.parse import urlparse, urlunparse


def load_config(path):
    with open(path, 'r') as f:
        return yaml.safe_load(f)


parser = argparse.ArgumentParser(description='Search Robot')
parser.add_argument('config', type=str, help='Path to config.yaml')
args = parser.parse_args()

cfg = load_config(args.config)

client = MongoClient(cfg['db']['host'], cfg['db']['port'])
db = client[cfg['db']['name']]
a_coll = db[cfg['db']['collection_articles']]
q_coll = db[cfg['db']['collection_queue']]

a_coll.create_index("url", unique=True)
q_coll.create_index("url", unique=True)

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',
    'Accept-Language': 'ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7'
}


def normalize_url(url):
    parsed = urlparse(url)
    clean_url = urlunparse((parsed.scheme.lower(), parsed.netloc.lower(), parsed.path, '', '', ''))
    if clean_url.endswith('/'):
        clean_url = clean_url[:-1]
    return clean_url


def load_state():
    if os.path.exists(cfg['logic']['state_file']):
```

```
        with open(cfg['logic']['state_file'], 'r') as f:
            return json.load(f)
    return {"b17_page": 1, "psych_page": 1}


def save_state(b17_p, psych_p):
    with open(cfg['logic']['state_file'], 'w') as f:
        json.dump({"b17_page": b17_p, "psych_page": psych_p}, f)


def add_to_queue(url, source):
    url = normalize_url(url)
    existing_doc = a_coll.find_one({'url': url})

    if existing_doc:
        last_scan = existing_doc.get('timestamp', 0)
        if time.time() - last_scan < cfg['logic']['recrawl_interval']:
            return False
        else:
            print(f"Re-crawl: {url}")

    try:
        q_coll.update_one(
            {'url': url},
            {'$set': {'url': url, 'source': source, 'added_at': time.
               time()}},
            upsert=True
        )
        return True
    except:
        return False


def harvest_lists():
    state = load_state()
    b17_page = state['b17_page']
    psych_page = state['psych_page']

    print("Starting harvester...")

    try:
        while True:
            # B17
            try:
                r = requests.get(f"https://www.b17.ru/article/?page={
                   b17_page}", headers=HEADERS)
                if r.status_code == 200:
                    soup = BeautifulSoup(r.text, 'lxml')
                    links = soup.find_all('a', href=True)
                    if not links:
                        break

                    count = 0
```

```
                    for a in links:
                        href = a['href'].split('#')[0]
                        if href.startswith('/article/') and href.count('/') == 3:
                            if add_to_queue("https://www.b17.ru" + href, 'b17'):
                                count += 1
                    print(f"B17 page {b17_page}: {count} added")
                    b17_page += 1
            except Exception as e:
                print(f"Error B17: {e}")

            time.sleep(1)

            # Psychologies
            try:
                r = requests.get(f"https://www.psychologies.ru/articles/{psych_page}/", headers=HEADERS)
                if r.status_code == 200:
                    soup = BeautifulSoup(r.text, 'lxml')
                    count = 0
                    for a in soup.select('a.rubric-anons_title, .rubric-anons_list a.link'):
                        href = a.get('href')
                        if href:
                            full = href if href.startswith('http') else "https://www.psychologies.ru" + href
                            if add_to_queue(full, 'psychologies'):
                                count += 1
                    print(f"Psych page {psych_page}: {count} added")
                    psych_page += 1
            except Exception as e:
                print(f"Error Psych: {e}")

            save_state(b17_page, psych_page)
            time.sleep(1)

    except KeyboardInterrupt:
        print("\nStopped.")


def process_queue():
    print(f"Crawler started (Delay: {cfg['logic']['delay_min']}-{cfg['logic']['delay_max']}s)")

    while True:
        task = q_coll.find_one_and_delete({})
        if not task:
            print("Queue empty.")
            break

        url = task['url']
        source = task['source']
```

```
147
148         time.sleep(random.uniform(cfg['logic']['delay_min'], cfg['logic
                ']['delay_max']))
149
150         try:
151             r = requests.get(url, headers=HEADERS, timeout=10)
152             if r.status_code == 200:
153                 html = r.text
154                 soup = BeautifulSoup(html, 'lxml')
155
156                 if source == 'b17':
157                     title, text = extract_b17_data(soup)
158                 else:
159                     title, text = extract_psych_data(soup)
160
161                 if len(text) > 200:
162                     old_doc = a_coll.find_one({'url': url})
163                     if old_doc and old_doc.get('clean_text') == text:
164                         a_coll.update_one({'url': url}, {'$set': {'
                                timestamp': time.time()}})
165                         print(f"Unchanged: {url}")
166                     else:
167                         doc = {
168                             'url': url,
169                             'source': source,
170                             'title': title,
171                             'raw_html': html,
172                             'clean_text': text,
173                             'timestamp': time.time()
174                         }
175                         a_coll.replace_one({'url': url}, doc, upsert=
                                True)
176                         action = "Updated" if old_doc else "Saved"
177                         print(f"{action}: {url}")
178                 else:
179                     print(f"Short content skip: {url}")
180             else:
181                 print(f"Status {r.status_code}: {url}")
182         except Exception as e:
183             print(f"Network error: {url} {e}")
184
185
186 def extract_b17_data(soup):
187     title_tag = soup.find('h1', class_='from_bb_h1') or soup.find('h1')
188     title = title_tag.get_text(strip=True) if title_tag else "No Title"
189     content_div = soup.find('div', attrs={'itmprp': 'articleBody'}) or
            \
190                   soup.find('div', attrs={'itemprop': 'articleBody'})
                      or \
191                   soup.find('div', id='article_body')
192     if content_div:
193         for j in content_div.find_all('div', class_='art_start'): j.
                decompose()
```

```
        for t in content_div(['script', 'style']): t.decompose()
        return title, content_div.get_text(separator=' ', strip=True)
    return title, ""


def extract_psych_data(soup):
    title_tag = soup.find('h1', class_='article__title')
    title = title_tag.get_text(strip=True) if title_tag else "No Title"
    parts = []
    lead = soup.find('p', class_='article__lead-paragraph')
    if lead: parts.append(lead.get_text(strip=True))
    body = soup.find('section', attrs={'itemprop': 'articleBody'})
    if body:
        for bl in body.find_all('div', class_='article__block_type-text
            '):
            parts.append(bl.get_text(separator=' ', strip=True))
    return title, " ".join(parts)


if __name__ == "__main__":
    print("1. Harvest")
    print("2. Crawl")
    choice = input("Choice: ")
    if choice == '1':
        harvest_lists()
    elif choice == '2':
        process_queue()
```

Вывод

В ходе лабораторной работы был разработан полнофункциональный поисковый робот на языке Python, интегрированный с СУБД MongoDB. Реализованная двухэтапная архитектура (разделение на сбор ссылок и загрузку контента) позволила эффективно управлять очередью задач и гибко настраивать интенсивность запросов к каждому источнику.

Использование контрольных точек (checkpoint) обеспечило отказоустойчивость: робот успешно возобновляет работу после прерывания. Была внедрена система отслеживания изменений, исключающая дублирование данных и позволяющая актуализировать корпус документов без лишних затрат ресурсов. Итоговый робот полностью соответствует требованиям ТЗ и сформировал надежную базу данных для последующих этапов построения поискового движка.

Литература

[1] Маннинг, Рагхаван, Шютце *Введение в информационный поиск* — Издательский дом «Вильямс», 2011. — 528 с.