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: В. М. Филиппов
Преподаватель: А. А. Кухтичев
Группа: М8О-410Б

Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №1 «Добыча корпуса документов»

Необходимо написать парсер на любом языке программирования.

Написать поисковый робот — компоненты обкачки документов, используя любой язык программирования; Единственным аргументом поисковому роботу подаётся путь до yaml-конфига, содержащий: Данные для базы данные в секции db; Данные для робота в секции logic: задержка между обкачкой страницы; Любые другие данные, необходимые для реализовации логики поискового робота. Сохранять в базе данных (например, MongoDB) документы со следующими полями: url, нормализованный; «сырой» html-текст документа; название источника; Дата обкачки документа в формате Unix time stamp. Поисковый робот можно остановить в любой момент и при повторном запуске робот должен начать 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.

1 Описание

Для реализации поискового робота был выбран язык программирования `Go`, так как его средства конкурентного программирования позволяют эффективно распараллелить процесс сбора данных. В качестве хранилища документов используется база данных `MongoDB`.

Сбор данных осуществляется с помощью библиотеки `gocolly`. Она позволяет гибко настраивать параметры краулера: устанавливать задержку между запросами (включая случайное смещение (Jitter) для эмуляции поведения реального пользователя), регулировать количество одновременных потоков и настраивать `User-Agent`.

1 Алгоритм работы

Глобально логика работы робота выглядит следующим образом:

1. Загрузка конфигурации из аргументов командной строки, инициализация и настройка экземпляра краулера.
2. Проверка наличия устаревших документов в базе (требующих повторного сбора). Если таковые есть — они добавляются в очередь.
3. Запуск основного цикла поиска и загрузки: получение страницы, валидация (проверка на «мусорный» контент или служебные разделы), добавление в очередь.

Процесс обработки различается в зависимости от источника:

Wikipedia. Если URL содержит домен `wikipedia.org`, проверяется тип страницы. Наличие подстроки `/Category:` в пути указывает на то, что это категория. В противном случае страница считается статьей.

- **Для категорий:** производится поиск всех ссылок по селекторам `#mw-pages a[href]` и `#mw-subcategories a[href]`, найденные ссылки добавляются в очередь.
- **Для статей:** страница сохраняется в БД. Перед сохранением производится проверка на дубликаты путем сравнения хэш-суммы текущей версии статьи с версией, уже находящейся в базе.

The Guardian. Если URL содержит домен `theguardian.com`, тип страницы определяется по структуре ссылки:

- **Статьи:** как правило, содержат в пути дату публикации (год, месяц, день), например: `/2023/oct/25`.
- **Категории:** имеют малую вложенность, например: `/sport/tennis`.

Если страница не удовлетворяет ни одному из условий, она помечается как имеющая неизвестный тип (вероятно, служебная) и игнорируется.

2 База данных

В базе данных я храню следующие поля.

- ID документа
- URL страницы
- Нормализованный URL
- Источник страницы (wikipedia/theguardian)
- HTML-код страницы
- Заголовок страницы
- Хэш HTML-кода
- Время первой обкачки
- Время последней обкачки
- Размер HTML-кода страницы в байтах

Итого, при помощи этого робота, у меня получилось загрузить 305135 документов, из которых 254904 - это статьи с Википедии, а 50251 с The Guardian. База данных получилась размером в 10,78920657 ГБ. Не пропорциональность в загруженных документах объясняется тем, что Википедия даёт очень много ссылок с одной категориальной страницы, тем самым забивая очередь на обработку.

2 Исходный код

Архитектурно приложение разделено на три ключевых слоя:

Слой сбора. Это основная логика в пакете app (Spider). Краулер запускает асинхронные HTTP-воркеры (через colly), которые обходят страницы. Внутри коллбэков (OnHTML) происходит эвристический анализ: URL классифицируется как «Статья» или «Категория». Если это статья, контент очищается через go-readability и отправляется в буферизированный канал saveChan. Если категория — извлекаются новые ссылки для рекурсивного обхода (до заданной глубины MaxDepth).

Слой сохранения (Consumer): Отдельная горутина (startWriter) постоянно слушает канал saveChan и записывает обработанные документы в MongoDB. Это позволяет не блокировать быстрый процесс скачивания страниц медленными операциями записи в БД.

Слой управления и конфигурации: Приложение управляется через CLI с использованием атомарных флагов и context для синхронизации. Логика запуска разделена на две фазы: обновление устаревших записей из БД и поиск новых страниц по стартовым URL. Все параметры (лимиты, таймауты, селекторы) вынесены в YAML-конфигурацию.

3 Выводы

При написании поискового робота я научился обкачивать документы с сайтов с использованием языка Go. Узнал про существование robots.txt и принципах вежливого скрапинга, научился обходить капчи, редиректы и прочие препятствия на пути. На практике познакомился с тем, как устроены HTML-страницы, научился их разбирать. Ранее я никогда не занимался web-скрапингом, для меня это был интересный опыт, который обязательно пригодится мне в будущем!

Список литературы

[1] Маннинг, Рагхаван, Шютце *Введение в информационный поиск* — Издательский дом «Вильямс», 2011. Перевод с английского: доктор физ.-мат. наук Д. А. Клюшина — 528 с. (ISBN 978-5-8459-1623-4 (рус.))

[2] Веб-скрапинг при помощи Go | Полное руководство 2024 // Nstbrowser: [сайт]. URL: https://www.nstbrowser.io/ru/blog/golang-web-scraping (дата обращения: 12.12.2025).