A minimal web crawler in Java

Introduction and Requirement

A minimal web crawler at its core downloads urls, discovers new urls in the downloaded content and schedules download of the discovered urls, i.e.,

- Fetch a discovered URL
- Store the content on disk
- Discover new URLs by extracting URLs from the fetched content and crawl these etc

The crawler should be able to resume operations if shut down.

Implementation

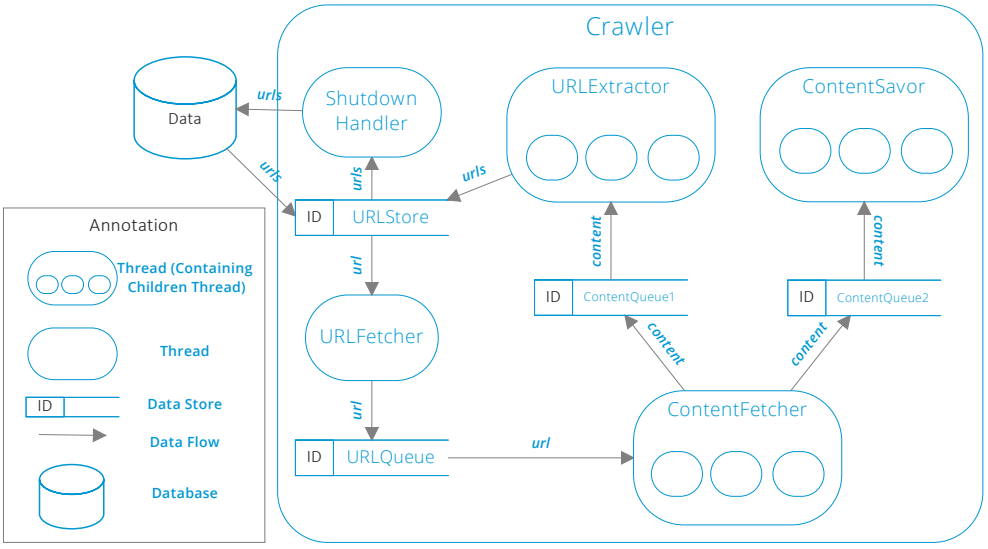

Figure 1 The data diagram of Crawler

The crawler is implemented in Java. **Crawler** is the main class of the implementation. Its data diagram is shown in Figure 1. It contains:

- A **URLStore** used to store urls
- five children threads which are responsible for different tasks
    - A **URLFetcher** fetches urls from **URLStore** and puts fetched urls into a url queue
    - A **ContentFetcher** takes urls from the url queue, downloads contents for the urls and puts the downloaded contents into two content queues
    - A **URLExtractor** takes contents from one content queue, extracts urls

from the contents and puts extracted urls into **URLStore**
    - A **ContentSaver** takes contents from the other content queue and saves the contents as local files
    - A shutdown handler thread performs the following tasks when a shut down signal is issued:
        - Interrupt the thread **URLFetcher**. When the thread **URLFetcher** receives interruption, it clears the url queue, puts an empty url into the url queue and stops itself
        - When the thread **ContentFetcher** takes the empty url from the url queue, it puts an empty content into the two content queues. Then it will stop itself after all its children threads finish
        - When the thread **URLExtractor** take the empty content from its content queue, it will stop themselves after all their child threads finish
        - Similarly, when the thread **ContentSaver** take the empty content from its content queue, it will stop themselves after all their child threads finish
        - After all the four children threads stop, the shutdown handler will save the urls in **URLStore** into database.

    Note that three of the children threads contain also children threads for individual tasks. For example, ContentFetcher contains children threads. Each children thread downloads content for a given url. In this way, these children threads can be executed concurrently.

Crawler performs all the tasks and functions as required. The workflow of the crawl consists of the following steps in sequence:

- pre-configure simply the crawl, i.e., creating a directory named data for containing database and crawled data, if such a directory does not exist
- load urls from database into **URLStore**
- create and start its four children threads
- Register the shutdown handler thread

Usage

```
Usage: java -jar crawler [--url url_strings+] [--file files+])
: --url specify the initial urls to be crawled
: --file specify the files containing the initial urls to be crawled
: Both --url and --file parameters can be omitted. In this case, the initial url is "www.telenor.no".
```

Source code Access

The source code can be accessed using of git from this repository https://github.com/wxlfrank/crawler. It is in a maven project. It can compiled by mvn clean install command.

Assumptions

- URLs are stored in SQLite database. We assume that a large amounts of data can be efficiently and correctly stored and access by SQLite.
- Jsoup is used to extract urls from web content. We assume that Jsoup library can efficiently and correctly extract urls from web content.
- The solution is tested on Windows 10 and Linux (Ubuntu) systems with Java 1.8.
- We assume that memory allocation always successes. In other words, it means that there is always enough memory available.
- We assume that the internet access is available.
- We assume that a url with www is different from the url without www.

Limitations and Future Improvement

- In this implementation, we fetch text based content for http or https urls. For other kind of contents and other kinds of urls, we will leave this for future work.
- During runtime, urls are stored in a hash map where the keys are the host names of the urls. In the future, we can consider more complex data structures to access and store urls more efficiently.
- In this implementation, we did not consider using strategies to reduce redundancy. We will consider this problem in the future.
- For simplicity, we assign the url queue, the content queues and the children threads with equal size. In the future, in order to make best use of concurrency, it is better to assign these data with different size.
- When the program shuts down, we clear urls in the url queue and waiting for other threads finishing. For content queues with large size, the waiting may take longer time. In the future, we should consider to offer the users with options to stop quickly. That is to clean the content queues when the program shuts down.