BuScrapper - dokumentacja

7 lutego 2017

Spis treści

1 Opis programu

Przeznaczeniem programu jest wyodrębnianie rozkładów jazdy ze stron o znanej strukturze znanej użytkownikowi. Prezentowana wersja pliku konfiguracyjnego jest dostosowana do pobierania rozkładów autobusów krakowskiego mpk (stan z dnia 07.02.2017), jednak poprzez wykorzystanie słabych powiązań pomiędzy klasami, program umożliwia podmianę odpowiednich fragmentów w celu wykonywania pewnych fragmentów zadania w sposób określony przez użytkownika ingerującego w kod, bez konieczności ingerencji w pozostałe części kodu.

Wyodrębnione dane zostają zapisane na dysku w postaci plików (nazwa linii wraz z kierunkiem określa nazwę katalogu dla zestawu przystanków uczęszczających zadaną linią w danym kierunku, w katalogu znajdują się pliki o nazwach równoważnych nazwom przystanków, a każdym pliku znajdują się natomiast godziny odjazdów w konwencji godzina, minuty dnia powszedniego, minuty soboty, minuty niedzieli, oddzielone znakiem :, pomocniczo dla wyszukiwarek tworzony jest katalog o nazwie buStops, w którym zapisywane są pliki, których nazwy odpowiadają nazwom przystanków, a w nich zapisywane są numery linii wraz z kierunkiem, które przejeżdżają przez dany przystanek), z których program jest w stanie wyodrębniać informacje dotyczące godzin odjazdów łączących dwa przystanki połączone przynajmniej jedną linią, używając do tego celu wyszukiwarki, która może zostać w łatwy sposób podmieniona na dowolną implementację, która będzie w stanie wyodrębniać informacje z tak zapisanych plików.

Program nie posiada interfejsu graficznego, cała komunikacja między użytkownikiem a programem odbywa się poprzez plik konfiguracyjny o nazwie Conf w folderze Conf, którego odpowiednie wypełnienie jest kluczowe dla odpowiedniego działania programu. Plik konfiguracyjny posiada określoną konwencję wyjaśnioną w dalszej części instrukcji. Działający program zapisuje wyciągane z

internetu dane do plików, natomiast informacje o wyszukanych połączeniach wypisuje na standardowe wyjście.

2 Opis działania

Sam BuScrapper dzieli zadanie wyciągnięcia danych, na zadania mniejsze, które na drodze wykonania programu są oddelegowywane do odpowiednich komponentów, które informują o powodzeniu swojego zadania do logiki od której zadanie otrzymały lub przenoszą swoje wyniki do kolejnych komponentów, które wykonują swoje zadania.

Najpierw program przy pomocy obiektu Configurator odczytuje wszystkie polecenia użytkownika umieszczone w pliku konfiguracyjny, wyodrębnia z nich szczegóły potrzebne przy tworzeniu zadań dla wątków pobierających, oraz przy pomocy obiektu TaskManager umieszcza utworzone obiekty Task w pamięci, jedynym ograniczeniem dla ilości zadań jest pamięć dostępne w komputerze. Odczytuje również wyrażenia XPath elementów pobranych stron, które mają zostać wyodrębnione przez wątki wyłuskujące oraz dane potrzebne do konstruowania zapytań dla hosta, z których następnie tworzy kolejkę wyszukiwań dla obiektu Browser, znów jedynym ograniczeniem ilości utworzonych zapytań jest pamięć dostępna w komputerze.

Następnie program sprawdza czy użytkownik zadeklarował chęć zaktualizowania danych, jeżeli jego odpowiedź była odmowna, program przejdzie do części odpowiedzialnej za wyszukiwanie, natomiast jeżeli odpowiedź użytkownika była pozytywna, program przeprowadzi aktualizację.

Podczas aktualizacji wątek główny funkcjonuje tak długo, dopóki pozostają do wykonania jakieś zadania. W trakcie wykonywania pojedynczego zadania, tworzy on wątki pobierające oraz wyłuskujące, przypisując dla nich obiekt bufora w pamięci na już pobrane strony, wyrażenia XPath przygotowane przez Configurator, określa ich systemy do składowania logów, a poprzez utworzony obiekt RequestCreator, udostępnia kolejkę zapytań do wykonania, dla aktualnie wykonywanego zadania.

Zadanie jest określone jako niewykonane tak długo, dopóki wątki pobierające oraz wątki wyłuskujące nie zakończą pracy w sposób poprawny, to jest bez wyrzucenia błędu np. związanego z przerwaniem dostępu do internetu. Dla zadania, w trakcie wykonywania którego nastąpiły błędy w wątkach pobierających lub wyłuskujących nie następuje usunięcie go za pośrednictwem obiektu TaskManager, a więc przy pobieraniu kolejnego zadania, będzie ono nadal uwzględnianie. W ten sposób program zabezpiecza się np. przed przerwami w dostępie do strony z rozkładami, w trakcie wykonywania programu, co bez tego typu zabezpieczeń powodowałoby zawieszenie programu lub niepełną aktualizację zadania, w którym wystąpił błąd.

Wątki odpowiedzialne za pobieranie danych wykonują zapytania ze wspólnej kolejki zapytań, kończąc swoje działanie w sytuacji, gdy ta okaże się pusta lub w trakcie wykonywania zostanie wyrzucony wyjątek. W sytuacji niepoprawnego pobrania wątki informują o niepowodzeniu w trakcie wykonania zadania oraz kończą swoje zadanie. W normalnym działaniu jest również umieszczanie w buforze poprawnie pobranych stron, wątki pobierające sprawdzają bowiem czy otrzymany zasób zasób został zdobyty w poprawną odpowiedzią servera 200 OK. Jeżeli bufor okaże się przepełniony, wątki czekają na wątki wyłuskujące, które po pobraniu strony do przetworzenia wybudzają odpowiednie wątki pobierające.

Wątki wyłuskujące mają za zadanie działać tak długo, jak długo działają wątki pobierające lub w buforze znajduje się jeszcze jakaś strona do przetworzenia. Pobierają i usuwają pojedynczą stronę z bufora, następnie przy pomocy otrzymanych wyrażeń XPath wyciągają z niej żądane informacje, aby w następnej kolejności, w postaci mapy, gdzie kluczem jest nazwa danego wyrażenia, a wartością zawartość wyciągnięta przez to wyrażenie, oddelegować zadanie zapisania tych danych, to przechowywanej przez siebie implementacji interfejsu StoreBusInfo.

W dalszej części wątek wyłuskujący za pośrednictwem odpowiedniego obiektu implementujące StoreBusInfo (np. obiektu FileStoreBusInfo, zapisującego dane w plikach), zapisuje otrzymane informacje w określony przez siebie sposób, wyciągając spośród nich tylko te, które interesują daną implementację (możliwa jest np. implementacja, która wyciąga jedynie numery linii). FileStoreBusInfo dodatkowo sprawdza poprawność danych przed zapisaniem ich do plików, a jeżeli w którymś momencie dane okażą się niepoprawne, nie przeprowadza samego zapisu i kończy iterację wątku wyłuskującego.

Gdy wszelkie aktualizacje danych zostaną już przeprowadzone, program wykorzystuje przygotowaną przez Configurator kolejkę wyszukiwań i wypisuje na standardowym wyjściu wyniki przepuszczania wyszukań przez obiekt Browser. Obiekt Browser dzieli wyszukanie na dwa etapy, w pierwszej kolejności wydobywa dla dwóch ustalonych przez użytkownika przystanków wszystkie łączące je linie za pomocą obiektu LineBrowser, aby następnie przy użyciu HourBrowser odnaleźć wszystkie godziny odjazdów z pierwszego przystanku (bierze więc pod uwagę kierunek przejazdu), które odpowiadają zadanym kryteriom.

Zarówno LineBrowser jak i HourBrowser pobierają dane wprost z plików, dlatego muszą także sprawdzać czy są one zapisane poprawnie, oraz w przypadku obiektu HourBrowser sprawdzana jest również poprawność kryteriów wyszukania (godzina i minuta odjazdu, typ dnia oraz maksymalna ilość godzin po zadanym czasie) podanych przez użytkownika. Komunikaty dotyczące niepoprawności danych, nieistnienia zadanych przystanków, jak i właściwe wyniki wyszukiwania są wypisywane na standardowe wyjście.

3 Opis pliku konfiguracyjnego

W pliku konfiguracyjnym obowiązuje konwencja podania słowa kluczowego, bezpośrednio po nim znaku „=", a po którym aż do końca linijki informacji, które chcemy przypisać danemu słowu kluczowemu, dla przejrzystości słowa kluczowe podzieliłem na trzy osobne sekcje.

3.1 Informacje dotyczące szczegółów wyrażeń XPath

- XPATHNAZWA= - wyrażenie XPath opisujące gdzie w przeszukiwanym dokumencie może znaleźć się nazwa przystanku
- XPATHNUMER= - wyrażenie XPath opisujące gdzie w przeszukiwanym dokumencie może znaleźć się numer linii
- XPATHDIREC= - wyrażenie XPath opisujące gdzie w przeszukiwanym dokumencie może znaleźć kierunek linii

- XPATHCZASY= - wyrażenie XPath opisujące gdzie w przeszukiwanym dokumencie może znaleźć się linijka z godziną oraz minutami dla niej

3.2 Informacje dotyczące szczegółów zadań

- UPDATE= - wartość TRUE ustala ponowne przeprowadzenie aktualizacji danych, przy użyciu zapytań określonych w pozostałych słowach kluczowych każda wartość oprócz TRUE, spowoduje w programie pominięcie aktualizacji
- DOWNLOADMETHOD= - GET lub POST, czyli zapytanie protokołu HTTP, które program będzie wykorzystywał
- PAGEURL= - adres strony początkowej
- HOSTNAME= - nazwa hosta dla którego przygotowujemy zapytania
- ZAKRESLINII= - konwencja zakłada podanie numeru linii początkowej, następnie znak „:", a po nim numer linii końcowej - określa zakres linii, które mają zostać zaktualizowane
- MAXPRZYSTANKOW= - maksymalna liczba przystanków, która może zostać uwzględniona przy tworzeniu zapytań
- MAXKIERUNKOW= - maksymalna liczba kierunków, które mogą zostać uwzględnione przy tworzeniu zapytań

3.3 Informacje dotyczące szczegółów wyszukań

- SEARCH= - linijka pojedynczego wyszukania, kolejność linijek ma znaczenie dla umieszczania ich w kolejce wyszukań do wykonania przez obiekt Browser, linijka wyszukiwania musi zostać zapisana w następującej konwencji: nazwa przystanku początkowego, znak „:", następnie nazwa przystanku końcowego, znak „:", następnie rodzaj dnia (0 = dzień powszedni, 1 = sobota, 2 = niedziela), znak „:", następnie początkowa godzina wyszukania, znak „:", następnie początkowa minuta wyszukania, znak „:", oraz maksymalna ilość godzin od godziny początkowej

4 Opis możliwych zmian w kodzie

Oprócz wykorzystywania zastanego kodu, użytkownik może wprowadzić zmiany w implementacji poszczególnych komponentów. Przede wszystkim może określić własną implementację obiektów SnatchThread oraz DownloadThread, aby w inny sposób pozyskiwać zasoby oraz wyciągać z ich informacjem Może wprowadzić własną implementację interfejsu StoreBusInfo dla obiektu SnatchThread, aby z wyłuskanych danych wybierać tylko określone przez siebie, sprawdzić ich poprawność oraz zapisać dane w ustalony przez siebie sposób. Zmiana obiektu Configurator może wpłynąć na generowanie zupełnie nowych formatów pliku konfiguracyjnego, pod warunkiem, że będzie zwracał dla pozostałych komponentów te same informacje, które zwraca zastana wersja Configuratora. Zmiana obiektu Browser może rozszerzyć zdolności wyszukiwania zapisanych danych. Użytkownik może również wprowadzić swoją wersję implementacji interfejsu PagesBuffer, pamiętając o tym, aby był kontenerem synchronizowanym. Aby zmienić sposób tworzonych zapytań, użytkownik może wprowadzić swoją implementację obiektu RequestCreator.

Dla przykładu, gdybyśmy chcieli zmienić program na wersję pobierającą wyłącznie numery linii ze strony skonstruowanej w sposób inny niż krakowska mpk, musielibyśmy określić jakie wyrażenie XPath opisuje te nazwy, zaimplementować obiekt RequestCreator, podając w nim zmienione szczegóły zasobu oraz zapytania.