Университет ИТМО
Факультет ПИиКТ

Низкоуровневое программирование

Лабораторная работа №2

Работу выполнил:
Абузов Ярослав

Группа:
P33302

Вариант:
XPath

Преподаватель:
Кореньков Ю. Д.

Санкт-Петербург
2022

Цель:

Реализовать модуль для разбора некоторого достаточного подмножества языка запросов по выбору в соответствии с вариантом формы данных.

Задачи:

1. Изучить выбранное средство синтаксического анализа
2. Изучить синтаксис языка запросов и записать спецификацию для средства синтаксического анализа.
3. Реализовать модуль, использующий средство синтаксического анализа для разбора языка запросов.
4. Реализовать тестовую программу для демонстрации работоспособности созданного модуля, принимающую на стандартный ввод текст запроса и выводящую на стандартный вывод результирующее дерево разбора или сообщение об ошибке.

Детали реализации:

Для создания модуля использован Bison (см. lexel.l и parser.y). Структуры для AST описаны в zgdbAst.h (описание ниже).

Поддерживаются следующие виды запросов:

- ADD <name> <schema> <path> - создание нового документа
- DELETE <path> - удаление документа
- UPDATE <element_name> <new_value> <path> - обновление элемента документа
- FIND <path> - поиск документа
- JOIN <child_predicate> ON <path> - поиск детей документа с условиями
- PARENT <path> - получить родителя

Детали каждой команды описаны в отчете 1 работы, сейчас же рассмотрим детали языка запросов.

- **<name>** - строчка с именем (все строчки должны быть заключены в двойные кавычки)
- **<schema>** - задает схему для команды ADD в виде (<type>:<key>:<value>), где <type> - тип поля, <key> - строка с символом @ в начале, <value> - начальное значаение
- **<child_predicate>** - в данном случае описывает предикаты для детей, но такой же синтаксис используется и в путях: [<key> <operation> <key>] для сравнения поля с полем внутри документа, [<key> <operation> <value>] для сравнения поля с константой, [<number>] для получения документа по номеру из текущей выборки. Перед

предикатом можно установить !, что будет означать инверсию. Предикаты можно связывать с помощью &(лог и) и |(лог или).

- **<path>** - главная часть языка, описание пути. Состоит из шагов, которые разделены / (абсолютный путь) или // (относительный путь). Шаг содержит имя документа (строка в кавычках) или имя элемента (строка в кавычках с @ в начале) с учетом текущего контекста. К каждому шагу могут быть применены предикаты (синтаксис выше).

P.S. Все нюансы про возможности самого движка исполнения команд описаны в отчёте 1 работы, не вижу смысла повторяться, поэтому далее будут примеры, на которых все будет ясно.

Язык запросов поддерживает следующие возможности:

- ✓ **Условия**
  - На равенство и неравенство для чисел, строк и булевских значений *(!=, =)*
  - На строгие и нестрогие сравнения для чисел *(<, >, <=, >=)*
  - Существование подстроки *(onepamop contains)*

- ✓ **Логическую комбинацию произвольного количества предикатов** *(&, |, !) примеры ниже.*

- ✓ **В качестве любого аргумента условий могут выступать литеральные значения (константы) или ссылки на значения, ассоциированные с элементами данных (поля, атрибуты, свойства)** *(примеры ниже; @"key" = "v", @"key"=@"key")*

- ✓ **Разрешение отношений между элементами модели данных любых условий над сопрягаемыми элементами данных** *(см. JOIN и пр.)*

Примеры запросов:

Простой запрос на создание документа:

```
ADD "new" (INT:@"key":2, DBL:@"key2":3.5) /"test1"//"test2";
```

Результат разбора (вывод AST):

```
Request type: ADD
    New document name: new

    Schema
    INT:key:2
    DBL:key2:3.500000

    Path
    Step name: test1; (absolute path, document step)

        Step name: test2; (relative path, document step)
```

Запрос посложнее на создание документа с предикатами в пути:

```
ADD "new" (INT:@"key":2, DBL:@"key2":3.5, STR:@"key3":"lol",
BOOL:@"key4":"true")
/"test1"[@"key6">"2"]//"test2"[@"key1"="v"]&![@"key2"contains"hi"]/@"test3";
```

Результат разбора (вывод AST):

```
Request type: ADD
    New document name: new

    Schema
    INT:key:2
    DBL:key2:3.500000
    STR:key3:lol
    BOOL:key4:1

    Path
    Step name: test1; (absolute path, document step)
    1p) by value: @key6 > 2;

        Step name: test2; (relative path, document step)
        1p) by value: @key1 = v;
        2p) connector: AND, inverted, by value: @key2 contains hi;

            Step name: test3; (absolute path, element step)
```

Запрос на JOIN с предикатами в пути и условиями над детьми:

```
JOIN [@"key"="lol"]|[@"key2"<="3"] ON /"test1"//"test2"![@"key" > "4"];
```

Результат разбора (вывод AST):

```
Request type: JOIN
    1p) by value: @key = lol;
    2p) connector: OR, by value: @key2 <= 3;

    Path
    Step name: test1; (absolute path, document step)

        Step name: test2; (relative path, document step)
        1p) inverted, by value: @key > 4;
```

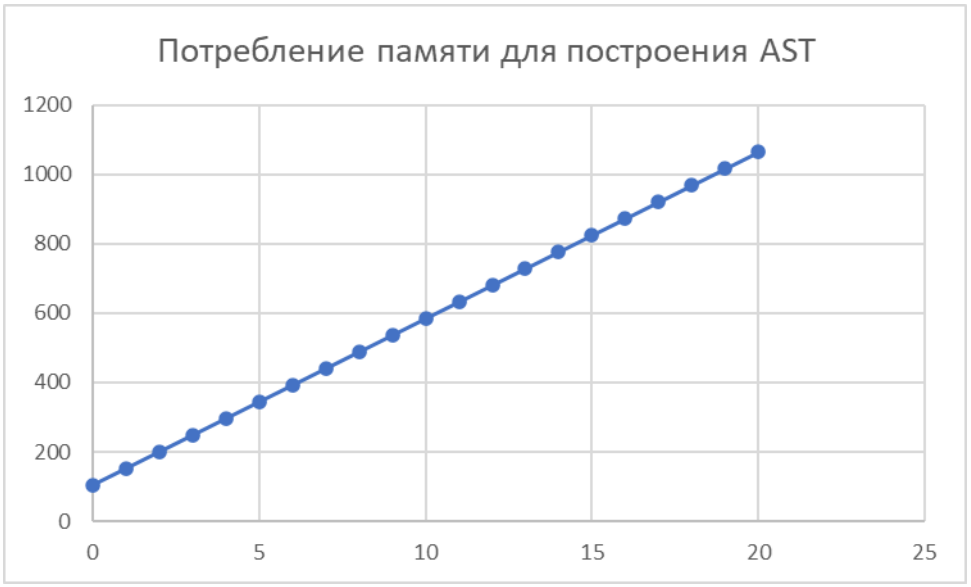
Запрос на обновление со сравнением поля с другим полем документа:

```
UPDATE @"key" "new_value" /"parent"[@"key2"=@"key3"];
```

Результат разбора (вывод AST):

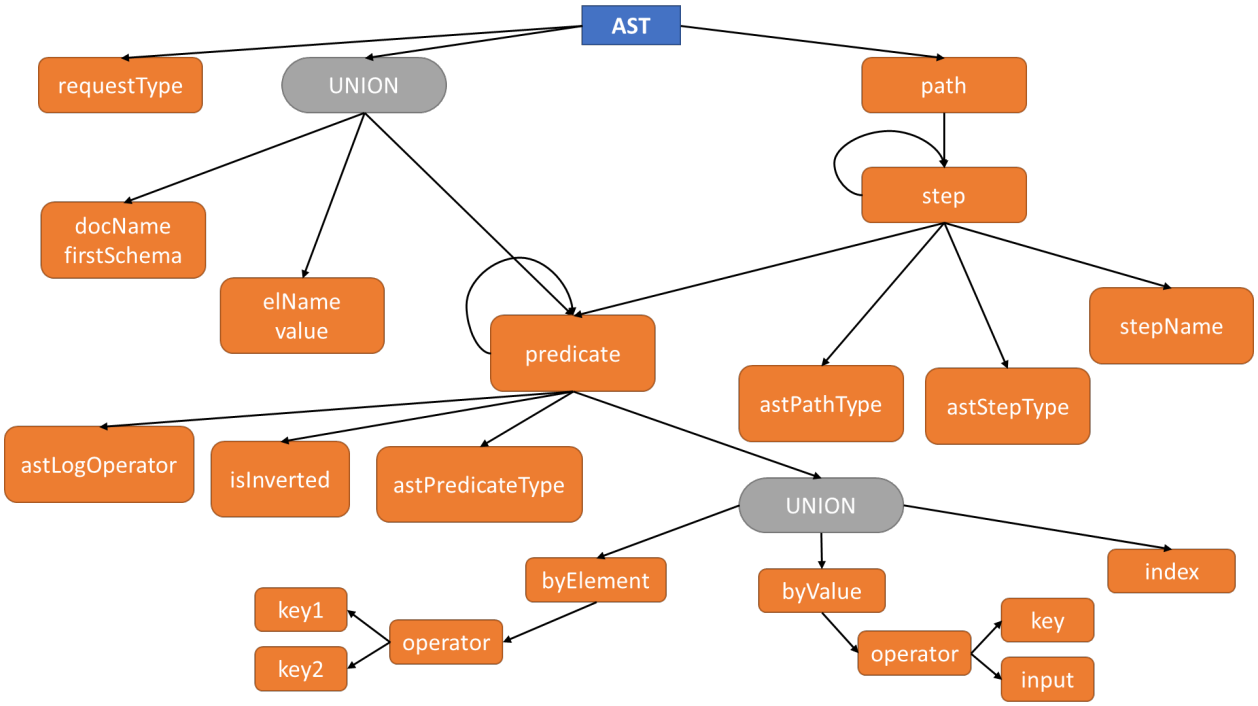
```
Request type: UPDATE
    Element name: key
    New value: new_value

    Path
    Step name: parent; (absolute path, document step)
    1p) by element: @key2 = @key3;
```

Потребление памяти:



Потребление памяти линейное и зависит от количества условий (количество предикатов и шагов). По оси Y – количество байт, по оси X – количество предикатов. График для увеличения шагов аналогичен.

Структура AST:



Подробнее в zgdbAst.h

Вывод:

В рельтате работы был создан модуль для синтакисческого налаиза языка запросов и построения простого AST. Для реализации я изучил язык XPath и Bison, последний позволил без сильных затрат автоматизировать создание анализатора и парсера с помощью простых правил. Дополнительных действих потребовалось мало – только разложить все по нужным структурам т.к. резльтата работы Bison уже хватало. Программа получилась небольшой с точки зрения кода, а также не потребляет много памяти, что видно по графику выше.