Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э.Баумана (национальный исследовательский университет)»

Группа: ИУ10-24

Студент: Тазалова А.В.

Хабаровск - 2021

**Транслятор унифицированных запросов**

# **Введение**

Многообразие существующих на сегодняшний день систем управления базами данных способно существенно замедлить процесс разработки приложений, поскольку структуры запросов для конкретной базы заметно отличаются. Различие есть даже в типе данных запросов – где-то это строка, а где-то - json-файл. В таком случае крайне полезным и **актуальным** было бы разработать инструмент трансляции запросов (и их типов), так как он позволит значительно ускорить работу с базами данных на серверной стороне какого-либо приложения, избавляя разработчика от необходимости переводить и переписывать запросы с одного языка на другой вручную.

Другим немаловажным аспектом разработки является скорость выполнения алгоритма и/или возможность его оптимизации, что во многом зависит не только от знаний и умений программиста, но и от языка программирования, выбранного разработчиком для написания кода драйвера или приложения. Одним из языков, удобных для написания инструмента трансляции, является golang, разработанный компанией Google.

Таким образом, **целью** моей работы является написание инструмента трансляции для заранее сформированного унифицированного запроса в формате json-файла в запросы sql и elasticsearch.

Главные задачи:

1. Изучить синтаксис языка golang
2. Создать программное представление (модели) унифицированного запроса и elasticsearch запроса, так как они представлены json-файлом.
3. Создать методы моделей унифицированного и elasticsearch запросов, которые будут заполнять вышеуказанные модели (для унифицированного запроса – метод чтения, для elasticsearch запроса – методы трансляции)
4. Создать методы модели унифицированного запроса, которые будут преобразовывать его в строку sql запроса
5. Создать тесты для проверки работоспособности инструмента трансляции. В случае положительного результата загрузить исходный код в репозиторий на github

# **Структура транслятора**

## **1.1 Модели запросов**

Для того, чтобы обработка запросов различными методами трансляции стала возможной, необходимо в первую очередь представить сами запросы в виде некоторых программных моделей. В языке Go для реализации моделей запросов, представленных в виде json-файла (а таковыми являются унифицированный запрос и elasticsearch-запрос) удобнее всего использовать структуры – специально создаваемый составной тип, в котором сгруппированы несколько переменных разных типов. При этом элементы структуры называются полями структуры (2, стр. 164). Объявление структуры имеет вид: type StructName struct { Field1 type1 Field2 type2 … Fieldn typen } . При этом имена полей структуры обычно начинаются с заглавной буквы (зависит от предстоящей обработки полей). Создание новой переменной типа созданной выше структуры имеет вид: var s1 StructName. Доступ к определённому полю структуры по его имени осуществляется с помощью обращения «через точку» вида: s1.Fieldi, i = 1, .., n. Литерал структуры определяется выражением вида: p1 := StructName{ typeExpression1, typeExpression2, .., typeExpressionn }. Но поскольку запомнить последовательность типов полей структуры иногда очень сложно, в языке Go есть возможность использования другой формы для определения литерала структуры: p1 := StructName{ Fieldi1: typeExpressioni1, Fieldi2: type Expressioni2, .., Fieldin: typeExpressionin }, где порядок полей Fieldij (ij = 1, .., N) произвольный. Кроме того, во второй форме записи определения литерала структуры в фигурных скобках не обязательно должны присутствовать все поля структуры, некоторые из них могут остаться с не присвоенным начальным значением.

Типы в языке Go (в частности структуры), как правило, объявляются вне функции main(), чтобы они относились к глобальной области видимости и были доступны для всего пакета Go. Противное допускается только в том случае, когда необходимо специально акцентировать внимание на том, что конкретный пользовательский тип полезен лишь в пределах текущей области видимости и его использование в других местах не предполагается (2б стр. 165).

Чтобы чтение унифицированного запроса из файла представлялось возможным, необходимо в первую очередь в исходном коде программы создать модель – тип, который будет подходящим для дальнейшей программной обработки запроса. Как было сказано ранее, моделью унифицированного запроса будет являться структура. Однако, поскольку унифицированный запрос сам по себе включает части, вид и тип которых заранее предугадать не представляется возможным, модель унифицированного запроса будет состоять не из одной структуры, а из целой совокупности структур, вложенных друг в друга. Далее будет представлено подробное описание каждой части унифицированного запроса.

Основой модели является структура UnifiedRequest structures. Она имеет четыре поля: Source, Requirements, Fields и поле, находящееся под разработкой – Modifiers. Поле Source представляет собой поле структуры строкового типа и служит для указания источника данных (место на локальном или на удалённом сервере, откуда должны поступить данные в соответствии с запросом). Инициализация данного поля происходит вне рамок текущей курсовой работы и индивидуальна для каждого конкретного запроса.

Вложенность модели структуры унифицированного запроса начинается с поля Fields. Данное поле представляет собой массив структур Field, и отображает столбцы таблицы базы данных, которые после выполнения запроса поступают на выход с заполненными значениями. Структура поля Field представляет собой структуру с двумя полями – Child и Name. Поле Child по своей сути является ещё одним полем типа структуры Field и указывает на дочерний элемент (в качестве примера – обращение к подстолбцам sql таблицы: Column1.SubColumn1). Таким образом, структура поля Field является рекурсивной структурой. Однако, поскольку язык Go требует для определения типа строгого значения выделяемой под данный тип объёма памяти, вместо определения типа рекурсивно, необходимо определить тип как указатель на структуру Field (\*Field). Впоследствии при преобразовании в типизированый запрос будет осуществлён рекурсивный обход по полю Child для нахождения полного наименования требуемого для вывода столбца. Поле Name олицетворяет имя текущего поля или подполя, и имеет строковый тип. При рекурсивном обходе по полю Child основное внимание уделяется имени поля или подполя на текущем шаге рекурсии, которое добавляется, например, в конец после точки в sql запросе.

Отдельное внимание стоит уделить полю модели структуры унифицированного запроса Requirements. Оно содержит основную часть запроса, указывающую условия получения данных (определяет сами возвращаемые значения). Тем не менее, именно это поле структуры содержит наиболее варьирующиеся и наиболее разнообразные вложенные типы, представленные аналогично с помощью структур. Рассмотрим подробно всю вложенную систему поля Requirements. Само по себе поле представляет указатель на структурный тип выражения требований (RequirementExpression struct). Указатель используется по причине, аналогичной причине использования в структуре Field в поле Child (необходимость вычисления объёма памяти, выделяемого под поле структуры). Структура выражения требования (RequirementExpression) имеет три поля: Type, OrAnd и Requirement. Поле Type является идентификатором типа выражения и имеет строковый тип. Всего возможны два варианта значений поля Type: “requirement” и “or\_and”. От значения поля Type зависит заполненность остальных полей структуры. Особенность структуры RequirementExpression заключается в том, что она по определению не может содержать в себе все заполненные поля одновременно, так как запрос может либо представлять собой простое условие (значение поля типа при этом равно “requirement”), либо являться логическим выражением, включающим в себя массив выражений простых условий и тип логический операции, которая применяется ко всем простым условиям из массива. Таким образом, если значение поля Type равно “requirement”, то в структуре выражения требования заполняется только поле Requirement, а поле OrAnd остаётся пустым, и тогда выражение требования является простым условием. Иначе, если значение поля Type равно “or\_and”, то заполняется только поле OrAnd, и выражение требования в свою очередь является логическим выражением, включающим в себя совокупность связанных логическим оператором простых условий.

Предположим, что запрос содержит в себе логическое выражение, соединяющее одним из двух возможных логических операторов (и/или) некоторое множество простых условий. Тогда строка типа в RequirementExpression равна “or\_and” и заполнено поле OrAnd. Рассмотрим тип поля OrAnd. Тип носит такое же название и является пользовательским типом, представляющим массив структур RequirementExpression, то есть, другими словами, если структура RequirementExpression является логическим выражением, то она содержит внутри одного из своих полей массив таких же структур RequirementExpression, которые в свою очередь могут являться либо простыми условиями, либо, аналогично первому уровню «вложенности», содержать внутри себя массив ещё одних таких же структур-логических выражений. Теперь предположим, что RequirementExpression является выражением требования, содержащим простое условие. Соответственно: type = “requirement”, поле OrAnd не заполнено, а поле Requirement содержит запрос. Рассмотрим подробнее тип Requirement. Данный тип представляет собой структуру с тремя полями – Operator, Field и Value. Поле Operator является полем, определяющим отношение сравнения между столбцом, указанным в поле Field, и значением, указанным в поле Value. Поле Field имеет одноимённый структурный тип, указывающий на столбец, из которого выбирается значение, соответствующее условию Operator’а. В случае если для выполнения условия проверяется не основное поле, а подполе, структура в поле Field имеет вложенные поля, указывающие на дочерний элемент (подполе). Поле Value имеет собственно значение, с которым, согласно оператору, сравниваются значения столбца Field. Value является структурой RequirementOperand, содержащей два поля: Type и Spec. Поле Type в рамках данной курсовой работы может содержать только одно строковое значение – «raw\_value». Однако, помимо вышеуказанного значения, данное поле может также содержать значение «Hook». О работе транслятора при таком значении типа операнда требования информации указано не будет, но, тем не менее, стоит отметить, что в таком случае второе поле структуры содержит Hook-запрос. Второе поле структуры операнда требования – поле Spec – содержит в себе значение, которое может принимать произвольный тип, и потому тип данного поля - пустой интерфейс. Приведём некоторые положения об интерфейсах в языке программирования Go.

Интерфейсный тип в Go определяет поведение других типов путем предоставления списка методов, которые необходимо реализовать. Чтобы тип соответствовал интерфейсу, в нем необходимо реализовать все методы, предусмотренные этим интерфейсом, — обычно этих методов не очень много.

Проще говоря, интерфейс — это абстрактный тип, определяющий множество функций, которые необходимо реализовать для типа, чтобы его можно было считать экземпляром интерфейса. Если это сделано, можно сказать, что тип удовлетворяет данному интерфейсу. Таким образом, интерфейс — это две составляющие: набор методов и тип. Интерфейс используется для определения поведения других типов.

Главное преимущество, которое мы получаем благодаря наличию и использованию интерфейса, состоит в том, что мы можем передавать переменную типа, реализующего этот интерфейс, любой функции, которая ожидает параметр данного интерфейса. Без такой удивительной возможности интерфейсы были бы лишь формальностью без сколь-нибудь значительной практической пользы.

Операции утверждения типа помогают выполнить две задачи. Во-первых, они проверяют, соответствует ли значение интерфейса определенному типу. При таком использовании операция утверждения типа возвращает два значения: основное и логическое значение. Скорее всего, вам нужно именно основное значение, однако логическое значение указывает на то, была ли операция утверждения типа успешной.

Во-вторых, операции утверждения типа позволяют использовать конкретное значение, хранящееся в интерфейсе, или присвоить его новой переменной. Например, если в интерфейсе есть переменная типа int, то можно получить это значение, используя операцию утверждения типа. Но если операция утверждения типа завершилась неудачно и этот сбой не был обработан, то программа запаникует.

Говоря об использовании интерфейсов в полях структур моделей транслятора, следует акцентировать внимание на том, что интерфейсы являются пустыми. Пустые интерфейсы (иными словами, интерфейсы, у которых не указаны методы типов в фигурных скобках) ничем не отличаются от интерфейсов с указанными методами, только пустым интерфейсам удовлетворяет любой тип. То есть, пустой интерфейс в поле Spec говорит о том, что данное поле может принимать абсолютно любые выражения любых типов (которые впоследствии будут внутри соответствующих методов доопределяться для дальнейшей работы со значением).

Помимо унифицированного запроса в формате json-файла представлен также и elasticsearch запрос. Следовательно, его также необходимо представить в виде структурированной модели на языке Go. Основной трудностью при создании модели elasticsearch запроса является форма записи условия в виде «столбец:значение», при этом сам оператор сравнения объявляется выше, в другом поле json-файла. По этой причине для многих из полей было бы удобно использовать тип, представленный видом «ключ:значение», где ключ имел бы строго заданный тип (так как тип значений, находящихся в столбцах, практически всегда известен заранее и является строковым типом), а значение – произвольный. Для выполнения такой задачи идеально подходит тип карты. Рассмотрим теоретические сведения о типе карт в языке программирования Go.

Хеш-таблицы, или карты (map) Go — то же, что известные хеш-таблицы, существующие во многих других языках программирования. Основным их преимуществом является способность использовать любой тип данных в качестве индекса, который в этом случае называется ключом карты или просто ключом. Несмотря на то что хеш-таблицы Go допускают использование в качестве ключей любых типов данных, тип данных, используемый в качестве ключа, должен быть сопоставимым, то есть у компилятора Go должна быть возможность отличать один ключ от другого, или, проще говоря, ключи хеш-таблицы должны поддерживать оператор ==.

К счастью, почти все типы данных сопоставимы. Однако, как можно догадаться, использование типа данных bool в качестве ключа хеш-таблицы определенно ограничило бы ее всего двумя значениями. Кроме того, применение в качестве ключей чисел с плавающей точкой может повлечь проблемы, вызванные точностью, принятой на разных машинах и в разных операционных системах.

Вставить данные в хеш-таблицу со значением nil не получится. Однако поиск, удаление, определение длины и использование циклов range для хеш-таблицы со значением nil не приведут к краху кода. Хеш-таблицы более универсальны, чем срезы и массивы. Правда, за такую гибкость приходится платить дополнительными вычислительными затратами на реализацию хеш-таблиц Go. Однако встроенные структуры Go очень быстрые, поэтому не бойтесь использовать хеш-таблицы Go, когда это необходимо. Следует помнить, что хеш-таблицы Go очень удобны и позволяют хранить различные типы данных, при этом просты для понимания и быстро работают.

Вернёмся к рассмотрению структур, описывающих модель elasticsearch запроса. Основной («собирающей») структурой является структура ESRequest, включающая в себя два поля: Source и Query. На самом деле, структура elasticsearch запроса гораздо более сложная и включает в себя, кроме прочего, ещё множество служебных полей, однако они не были рассмотрены в рамках данной курсовой работы. Поле Sourceпредставляет собой список полей таблицы базы данных, необходимых для вывода в результате выполнения запроса и получения в соответствии с ним табличных данных. По этой причине поле Source имеет тип строкового массива, где каждый элемент – название столбца (возможно, имеющего несколько дочерних столбцов), значения из которого поступят на выход. Поле Query несёт основную смысловую нагрузку всего запроса в целом, так как именно оно содержит условие выбора значений из таблиц баз данных, к которым предполагается адресовать запрос. Оно имеет тип Query, который в свою очередь является структурой, содержащей пять полей: Term, Terms, Bool, Range и MatchAll. Примечательно, что структура Query, как и структура RequirementExpression struct, не может содержать заполненными все поля одновременно. Но принципиальное отличие между этими двумя структурами есть. Заключается это отличие в том, что если поле, которое должно быть заполненным в структуре RequirementExpression, определялось полем Type, способным принимать два возможных значения, то в структуре Query может быть заполнено всего лишь одно из пяти полей. Рассмотрим каждое из них.

Поле Term по своей сути чем-то напоминает поле Requirement структуры RequirementExpression и представляет собой простое условие вида «столбец:значение». Понятие простого условия в elasticsearch запросе принципиально отлично от понятия простого условия унифицированного запроса. В первом случае отсутствуют операторы сравнения, поэтому простое условие (Term) имеет только одно значение – равенство значения(-ий) столбца данному по условию значению.

# **Источники информации**

1. The Go Programming Language [<https://golang.org>]