УДК 004.42

ТИПОБЕЗОПАСНОЕ АСИНХРОННОЕ ВЫПОЛНЕНИЕ SQL ЗАПРОСОВ ДЛЯ ЯЗЫКОВ JVM

О. Ю. Рязанов, Ю. Д. Рязанов

Белгородский государственный технологический университет им. В. Г. Шухова

Поступила в редакцию 29.03.2019 г.

Аннотация. В настоящее время для разработки высоконагруженных приложений применяются асинхронные фреймворки, библиотеки и драйверы. Одной из особенностей разработки приложений на основе асинхронных фреймворков является недопустимость вызова блокирующих функций в потоке обработки сообщений. Эта особенность создает трудности в использовании актуальных по назначению библиотек, написанных для однопоточной и многопоточной модели приложения, т. к. они могут содержать блокирующие функции или другие блокирующие драйверы. Примером такой библиотеки является библиотека jOOQ, которая предназначена для создания типобезопасных SQL запросов в коде программ на языке Java. В статье описан метод адаптации блокирующей библиотеки jOOQ для использования в асинхронном фреймворке, а также его реализация в виде библиотеки Vjooqx.

Ключевые слова: асинхронное программирование, параллельное программирование, база данных, SQL запрос.

ВВЕДЕНИЕ

Продолжительное время в разработках на языке Java использовались и используются в настоящее время различные фреймворки и библиотеки, в основе которых лежит пул потоков. Но подход с использованием пула потоков имеет определённые недостатки, которые заключаются в том, что при маленьком пуле потоков приложение не способно обработать большое количество запросов параллельно, а при большом пуле потоков много времени уходит на распределение ресурсов сервера между потоками из пула. Одним из способов устранения недостатков приложений, написанных с использованием пула потоков, является разработка приложений с использованием асинхронных фреймворков, библиотек и драйверов. В настоящее время появляется все больше средств для асинхронного выполнения задач [1–4]. Одной из главных отличительных особенностей разработки приложений на основе асинхронных фреймворков является недопустимость использования блокирующих функций. Блокирующие функции не наносят значительного вреда, когда используются в одном из потоков пула, потому что остальные потоки будут продолжать выполнять свои действия. Наличие блокирующей функции в Event-loop'е асинхронного контекста приведет к полной блокировке всей системы.

Эта особенность создает трудности в использовании актуальных по назначению библиотек, написанных для однопоточной или многопоточной модели, т. к. такие библиотеки могут содержать блокирующие функции или другие блокирующие драйверы. Примером такой библиотеки является библиотека jOOQ, которая предназначена для создания типобезопасных SQL запросов в коде программ на языке Java [4–6]. Обычное применение этой библиотеки в асинхронном фреймворке невозможно из-за того, что для выполнения построенных запросов в ней используется блокирующий драйвер JDBC [6].

В этой статье описан метод адаптации блокирующей библиотеки jOOQ для использования в асинхронном фреймворке, а также его реализация в виде библиотеки Vjooqx [7], которая расширяет функционал рассматриваемой библиотеки.

МЕТОД АДАПТАЦИИ БЛОКИРУЮЩЕЙ БИБЛИОТЕКИ JOOQ ДЛЯ ИСПОЛЬЗОВАНИЯ В АСИНХРОННОМ ФРЕЙМВОРКЕ

Библиотека jOOQ предоставляет программисту возможность написания SQL запросов в виде цепочки вызовов методов. Методы цепочки являются функциями, которые добавляют ключевые слова языка SQL и параметры запроса в строку SQL запроса. Библиотека генерирует классы-модели таблиц базы данных. Благодаря наличию таких классов запросы становятся типобезопасными [8]. Ошибки несоответствия типов параметров, отсутствия столбцов в таблицах и другие обнаруживаются на этапе компиляции. При использовании обычных строк для SQL запросов такие ошибки возникают во время выполнения программы, в результате чего ее работа становится некорректной. Недостатком библиотеки является неполная поддержка SQL синтаксиса и отсутствие возможности расширения преобразования результатов запроса в Java объекты. Данная библиотека обладает таким функционалом, как получение готовой строки SQL запроса без его исполнения. Это делает возможным создание адаптера для использования типобезопасных запросов в асинхронном контексте.

Существует асинхронный драйвер для базы данных PostgreSQL vertx-postgres [1]. По результатам тестов Techempower этот драйвер является самым быстрым для доступа к базе данных [9]. Он предоставляет возможность осуществлять SQL запросы, создавать транзакции, выполнять хранимые функции и процедуры. Входными данными для исполнения SQL запросов являются символьные строки, выходными данными – набор строк таблиц ответа. Таким образом, можно сделать вывод, что необходим адаптер, задачей которого является выполнение SQL запроса, составленного библиотекой jOOQ, через асинхронный драйвер, предоставляемый программистом, и преобразование результата выполнения запроса в Java объекты.

Поскольку в задачу адаптера входит преобразование данных, в рамках его реализации можно решить проблему, которая возникает при работе с реляционными базами данных. Она заключается в том, что в результате join запросов могут находиться повторяющиеся данные. Например, есть таблица стадионов, команд и игроков. Необходимо извлечь из базы данных информацию о стадионе, командах, которые на нём играют и составы команд. Это можно сделать в несколько запросов: узнали стадион, отфильтровали команды, отфильтровали игроков. Но операции запросов являются долгими, поэтому количество запросов нужно по возможности сокращать. Возможно объединить три таблицы, но тогда для каждого игрока будет повторяться информация о стадионе, а для игроков одной команды будет повторяться информация о команде. Помимо повторения в результате, данные будут повторяться в Java модели, которая будет содержать в себе три сущности.

Предлагаемое решение заключается в группировке объектов и получении древовидной структуры данных на этапе преобразования табличных данных в Java модель. Результатом такого преобразования для приведенного примера будет объект класса стадион, который агрегирует массив объектов класса команда, которые в свою очередь агрегируют массив объектов класса игрок.

БИБЛИОТЕКА VJOOQX

Библиотека Vjooqx написана на языке Kotlin. Основным классом в этой библиотеке является класс Vjooqx.kt. В нем реализовано три метода:

1) fun fetch(query: (DSLContext) –> Query): MapperStep – метод для получения выборки из базы данных по составленному запросу. Удобен для выполнения select запросов;

2) fun execute(query: (DSLContext) –> Query): Single<Int> – метод для выполнения запросов, результатом которых является количество измененных строк в базе данных;

3) fun transaction(): TransactionContext – метод для создания транзакционного контекста. В рамках этого контекста можно определить откат транзакции в случае ошибки или по какому либо условию сохранить изменения.

Для создания экземпляра объекта Vjooqx предусмотрены два способа. В первом способе нужно передать в конструктор:

1) AsyncSQLClient – клиент асинхронного драйвера;

2) DSLContext – интерфейс, определенный в библиотеке jOOQ и использующийся для построения запросов на конкретном SQL диалекте;

3) JsonParser – необходимый для маппинга результата запроса в Java-объекты;

4) Logger – интерфейс, который используется для логирования событий (необязательный параметр).

Во втором, более предпочтительном способе создания экземпляра объекта Vjooqx, используется класс VjooqxBuilder, который предоставляет возможность построения объекта в стиле FluentDesign.

Используя бибилотеку Vjooqx, выборку из базы данных можно осуществить следующим способом:

```
vjooqx.fetch{
      it.select().from(ENTITY_
TABLE).where(ENTITY_TABLE.id > 5)
} .toListOf(Entity::class.java)
```

Результатом выполнения этого метода будет объект класса Single<Entity>, который предоставляет возможность получить асинхронный результат операции. Рассмотрим подробнее все вызовы функций, их параметры и возвращаемые результаты.

vjooqx – объект класса Vjooqx, описанного выше. Сигнатура метода fetch описана ранее. Входным параметром этого метода является функция, которая в качестве аргумента принимает DSLContext, определенный на этапе создания экземпляра класса Vjooqx. DSLContext позволяет построить запрос. Выходным параметром функции, которая является аргументом функции fetch, является объект класса Query, определенный в библиотеке jOOQ. Из этого объекта можно получить строку SQL запроса, построенную внутри функции, переданной в fetch. В функции fetch осуществляется извлечение SQL строки, получение соединения с базой данных и выполнение запроса. Результатом запроса является Single<ResultSet>. Результатом выполнения функции fetch является экземпляр класса MapperStep. Он создается на основе результата выборки из базы данных, объекта класса JsonParser, установленного во время инициализации объекта Vjooqx и логгера. В этом классе объявлены следующие публичные методы:

```
   fun <T> to(pClass: Class<T>):
Single<T>
   fun <T> toListOf(pClass: Class<T>):
Single<List<T>>
   fun <T> toTree(pClass: Class<T>,
listAliases: List<String>): Single<T>
   fun <T> toTreeList(pClass:
Class<T>, listAliases: List<String>):
Single<List<T>>
```

Входным параметром метода to является класс, в который будет преобразован результат запроса. Результатом метода является объект класса Single, который испускает один объект переданного в функцию класса, либо ошибку, которая произошла во время выполнения запроса или преобразования результата в экземпляр класса. Этот метод используется для выборки одной строки. Согласно спецификации [10] Single не может испускать null. Поэтому, если в результате было выбрано ноль строк, Single выпустит ошибку EmptyResultSet.

Метод toListOf предназначен для выборки и преобразования в класс, который является входным параметром, нескольких строк результата SQL запроса. В отличие от метода to Single, который является выходным параметром функции, никогда не испустит ошибку EmptyResultSet, а испустит пустой массив в случае, если будет выбрано ноль строк.

Методы toTree и toTreeList предназначены для решения задачи создания древовидных объектов из таблиц. Входными параметрами методов являются класс, в который преобразуется результат запроса, и список названий полей, по которым необходимо проводить объединение N строк в одну. Объединение будет происходить в том случае, если поле, по которому выполняется свертка, имеет одинаковое значение в строках. Методы позволяют делать свертку для неограниченного количества вложенных объектов. Для обозначения

Таблица 1

column1	column2.pos1	column2.pos2.field1	column2.pos2.field2
1	3	7	a
1	3	7	b
1	3	8	c
1	4	9	d
1	4	9	e
2	5	9	e
2	5	9	e
2	6	12	h
2	6	12	h
2	6	12	h

уровня вложенности используется символ точка в названии столбца результата запроса.

Рассмотрим пример выполнения метода toTree.

Допустим, в результате выборки была получена таблица (табл. 1).

Далее представлены объявление класса A, в объекты которого преобразовывается результат SQL запроса, и агрегируемых им классов:

```
Class A {
   val column1 : Int
   val column2 : List<B>
}

Class B {
   val pos1 : Int,
   val pos2 : List<C>
}

Class C {
   val field1 : Int,
   val field2 : Int
}
```

В результате выполнения фрагмента кода:

```
val vjooqx : Vjooqx
vjooqx.fetch(/*SQL запрос
*/).toTreeList(A::class.java,
listOf('column1', 'pos1', 'field1', ))
```

будет получен список объектов, который представлен на рис. 1.

Алгоритм выполнения преобразования toTreeList следующий:

1. Преобразовать строку в объект требуемого класса.


```
[{
        "column1": 1,
        "column2": [{
            "pos1": 3,
            "pos2": [{
                "field1": 7,
                "field2": ["a", "b"]
            }, {
                "field1": 8,
                "field2": ["c"]
            }]
        }, {
            "pos1": 4,
            "pos2": [{
                "field1": 9,
                "field2": ["d", "e"]
            }]
        }]
    },
    {
        "column1": 2,
        "column2": [{
                "pos1": 5,
                "pos2": [{
                    "field1": 9,
                    "field2": ["e"]
                }]
            },
            {
                "pos1": 6,
                "pos2": [{
                    "field1": 12,
                    "field2": ["6"]
                }]
            }
        ]
    }
]
```


Рис. 1. Список объектов

2. Если результирующий массив пуст, поместить в него объект и перейти к пункту 6.

3. Если в списке (далее list) объекта из результирующего массива, который хранится в поле (далее fieldName), имя которого указано в списке полей для объединения, содер-

жится объект, который равен объекту в поле fieldName для текущей строки, а остальные поля текущего объекта равны полям объекта из результирующего списка, то добавить в list объекта из результирующего списка объект из поля fieldName текущего объекта. Перейти к пункту 1.

4. Преобразовать поля текущего объекта из списка полей для объединения в списки со значением поля.

5. Добавить текущий объект в результирующий список.

6. Если не все строки обработаны, выбрать следующую строку и перейти к пункту 1, иначе конец.

Библиотека Vjooqx предоставляет разработчику удобные инструменты для создания транзакций, подтверждения и отката изменений при выполнении условий или отката при возникновении ошибок. Как было сказано выше, транзакционный контекст создается при помощи вызова метода transaction у экземпляра класса Vjooqx. В транзакционном контексте, также как и в обычном, выполняется запрос методами fetch или execute. Результат запроса преобразуется в объект методами to, toListOf, toTree, ToTreeList. Результатом вызова последовательности методов для создания запроса, исполнения и преобразования его результата в транзакционном контексте является экземпляр класса TransactionStep. Этот класс необходим для создания условий подтверждения или отката транзакций. Для этого в нем реализованы следующие методы:

fun commit(): Single<T> – используется для подтверждения транзакции.

fun <E> then(action: (T, TransactionContext) –> Execution<E>): TransactionStep<E> – предназначен для выполнения еще каких-то запросов в том случае, если не произошла ошибка и откат в случае ошибки. Входным параметром является функция, принимающая результат предыдущих запросов и транзакционный контекст для составления последующих запросов.

fun thenCommit(action: (T) –> T): Single<T> – предназначен для выполнения какого-то действия с результатом запроса, и, в случае успешного выполнения действия, подтверждения транзакции. Входным параметром является функция, принимающая результат запроса и возвращающая объект такого-же класса.

fun rollBackIf(action: (T) –> Boolean): Completable – предназначен для выполнения отката изменений в случае выполнения условия action. Входным параметром является функция, которая принимает результат запроса и возвращает истину, если необходимо откатить транзакцию.

fun rollBackOnError() : Single<T> – предназначен для возвращения результата запроса и отката изменений в случае возникновения ошибки.

ЗАКЛЮЧЕНИЕ

При использовании библиотеки Vjooqx, в отличие от простых строковых SQL запросов, гарантируется, что во время работы программы не возникнут ошибки, связанные с синтаксически неправильными SQL запросами, с запросами, которые содержат в себе обращения к несуществующим таблицам, к несуществующим столбцам. Использование библиотеки позволяет значительно сократить время исправления SQL запросов в случае изменения структуры таблиц, изменения названий столбцов, их типов. Это достигается обнаружением ошибок на этапе компиляции, а не во время работы программы.

Библиотека разработана и реализована для работы с современным, быстрым фреймворком vert.x. Благодаря этому можно достичь высокой производительности без затрат на улучшение аппаратной составляющей системы. Представленная в статье библиотека имеет функционал по преобразованию структур данных, которого нет в известных аналогах [5, 6, 11].

Статья подготовлена в рамках программы развития опорного университета на базе БГТУ им. В. Г. Шухова

СПИСОК ЛИТЕРАТУРЫ

1. Async, Netty based, database drivers for PostgreSQL and MySQL written in Scala – Режим доступа: https://github.com/mauricio/postgresql-async. (Дата обращения: 04.03.2019).

2. *Каскиаро, М.* Шаблоны проектирования Node.JS / М. Каскиаро, Л. Маммино – М.: ДМК Пресс, 2017, 396 с.

3. *Nurkiewicz, T.* Reactive Programming with RxJava / T. Nurkiewicz, B. Christensen – O'Reilly Media, 2016 – 372 p.

4. *Bernhardt, M.* Reactive Web Applications / M. Bernhardt – Manning Publications, 2016 – 328 p.

5. jOOQ: The easiest way to write SQL in Java – Режим доступа: http://www.jooq.org. (Дата обращения: 04.03.2019).

6. The jOOQ User Manual. Single Page – Режим доступа: http://www.jooq.org/doc/3.11/manual-single-page. (Дата обращения: 04.03.2019).

7. Vjooqx – Режим доступа: https://github.com/weery28/vjooqx. (Дата обращения: 04.03.2019).

8. Typesafe sql – Режим доступа: http://www.jooq.org/#a=usp-typesafe-sql. (Дата обращения: 04.03.2019).

9. Round 16 results – TechEmpower Framework Benchmarks – Режим доступа: http://www.techempower.com/benchmarks/#section=data-r16&hw=ph&test=update. (Дата обращения: 04.03.2019).

10. ReactiveStreams – Режим доступа: https://github.com/reactive-streams/reactive-streams-jvm/blob/v1.0.2/README.md#specification. (Дата обращения: 04.03.2019).

11. Vertx-jooq – Режим доступа: https://github.com/jklingsporn/vertx-jooq. Дата обращения: 04.03.2019).

Рязанов О. Ю. – магистрант кафедры программного обеспечения вычислительной техники и автоматизированных систем, Белгородский государственный технологический университет им. В. Г. Шухова.
E-mail: weery@live.ru

Рязанов Ю. Д. – доцент кафедры программного обеспечения вычислительной техники и автоматизированных систем, Белгородский государственный технологический университет им. В. Г. Шухова.
E-mail: razanov.yd@bstu.ru

TYPE SAFE EXECUTION OF SQL QUERIES FOR JVM LANGUGAES

O. Yu. Ryazanov, Yu. D. Ryazanov

Belgorod State Technological University name after V. G. Shukhov

Annotation. At the current time asynchronous frameworks, libraries and drivers used for developing highload applications. One of the feature of developing applications based on asynchronous frameworks is inadmissibility for calling blocking functions in the event loop thread. This feature is creating difficulties for using actual by purpose libraries, that are written for single-thread or multi-threads models of application, because that can be contains blocking functions or other blocking drivers. The example of similar library is jOOQ, that intended for writing type safe SQL queries in the Java code. In this article describes the blocking library jOOQ adaptation method for using in asynchronous framework and implementation of its in library Vjooqx.
Keywords: asynchronous programming, parallel programming, database, SQL query.

Ryazanov O. Yu. – Graduate Student of the Department of Software Computer and Automated Systems, BSTU after V. G. Shukhov,
E-mail: weery@live.ru

Ryazanov Yu. D. – Associate Professor of the Department of Software Computer and Automated Systems, BSTU after V. G. Shukhov,
E-mail: razanov.yd@bstu.ru