ПРИМЕНЕНИЕ JAVA REFLECTION API ДЛЯ РАЗРАБОТКИ ORM-БИБЛИОТЕКИ

Маркова А.А.,
студентка 4 курса ВГУ имени П.М. Машерова, г. Витебск, Республика Беларусь
Научный руководитель – Ермоченко С.А., канд. физ.-мат. наук

В современных приложениях наиболее значимой задачей является организация длительного хранения и обработки информации. Для решения этой задачи наибольшее распространение получили системы управления реляционными базами данных (БД). Однако в объектно-ориентированных языках программирования работа с данными осуществляется в терминах классов, а не таблиц данных. Процесс преобразования информации из реляционной БД в объектное представление требует написания большого количества схожего повторяющегося кода.

Для обеспечения работы с данными в терминах классов и для преобразования данных классов в данные, пригодные для хранения в реляционных БД используется технология Object-Relational Mapping, реализуемая в так называемых ORM-библиотеках. Благодаря такой технологии нет необходимости писать SQL-код для взаимодействия с БД. При этом схему отображения (mapping) определяет сам разработчик [2].

Актуальность написания собственной ORM-библиотеки обусловлена возможностью автоматизации процесса взаимодействия с БД. ORM позволит практически не задумываться о том, как составляются SQL-запросы, как происходит получение данных и заполнение полученными значениями соответствующих классов.

Целью исследования является анализ возможностей Java Reflection API для разработки ORM-библиотеки.

Материал и методы. Материалом исследования являются CRUD (create, read, update, delete) – операции, автоматизируемые разрабатываемой ORM-библиотекой. В качестве методов исследования используются объектно-ориентированный анализ, метод проектирования реляционных БД и применение Java Reflection API [3].

Результаты и их обсуждение. Для взаимодействия ORM-библиотеки с реляционной БД необходимо задать соответствие между классами языка программирования Java и отношениями в БД. Такое соответствие задается с помощью конфигурационного XML файла.

Листинг 1 – Пример конфигурационного файла

```
<configure dbName="myorm" userName="root" userPass="root"
 url="jdbc:mysql://localhost/myorm"
 driver="com.mysql.jdbc.Driver">
 <entity interfaceName="orm.domain.Book"
 className="orm.domain.BookImpl" tableName="book">
 <property name="name" field="name"
 type="java.lang.String"/>
 <reference name="author" field-fk="author_id"
 type="orm.domain.Author" lazyload="true"/>
 <collection name="authors" type="orm.domain.Author"
 tableName="authors_books"
 joinColumnName="book_id"
 inverseJoinColumnName="author_id"/>
 </entity>
</configure>
```

Корневой тег configure задает параметры для установки соединения с БД. Внутри этого тега может содержаться произвольное количество тегов entity, каждый из которых описывает класс, представляющий некоторую сущность (например, автора или книгу). Теги property представляют собой простые поля в классе. Тег reference описывает ссылки в классе на другие сущности (например, автора книги).

Учитывая информацию в конфигурационном файле, в библиотеке при помощи ретроспекции осуществляется обращение к БД и разработчику предоставляется набор заполненных Java-классов, готовых к использованию.

Для всех сущностей в ORM-библиотеке определен общий базовый интерфейс Entity, определяющий геттер и сеттер для идентификатора сущности (поле id типа Long). Для правильной работы библиотеки все сущности должны реализовать этот интерфейс.

В разработанной ORM-библиотеке реализовано кэширование данных в соответствии с шаблоном проектирования «Identity Map». Для этого в классе, осуществляющем непосредственное взаимодействие с БД посредством выполнения SQL-запросов, реализована карта cache, которая хранит по идентификатору уже считанные из БД сущности. При вызове метода read по идентификатору сначала выполняется поиск объекта в cache. Если объект найден, то подключение к БД не выполняется, а возвращается объект из карты; иначе происходит обычная операция чтения данных.

Также при загрузке данных из БД в память приложения удобно пользоваться загрузкой не только данных об объекте, но и связанных с ним объектов (например, загружать список всех авторов выбранной книги). Однако может возникнуть ситуация, когда будет загружаться большое количество сопряжённых объектов, даже если эти данные реально не нужны и никогда не будут использоваться. Это может плохо сказаться на производительности приложения.

В связи с этим в библиотеке был реализован паттерн «Lazy Load» («Ленивая загрузка»), который используется для загрузки данных по требованию. Т.е. подразумевается отказ от загрузки данных, в которых нет необходимости на текущий момент [1]. Такой подход позволит увеличить производительность приложения.

Объект не будет содержать связанных с ним данных, но будет знать, где при необходимости их взять. Такой функционал был реализован используя следующий подход: в классе, перехватывающем вызов get-тера для связанного объекта, ставится маркер о том, что данные еще не были загружены. При первом вызове этого get-тера данные загружаются и маркер изменяется.

В ORM-библиотеке ленивая загрузка реализована с помощью классов Proxy и InvocationHandler пакета java.lang.reflect. Класс Proxy используется для добавления или изменения функционала уже существующих классов. В таком случае, прокси-объект применяется вместо исходного.

Во многих информационных системах возникает необходимость в использовании между отношениями связи «многие-ко-многим» (например, книгу написали несколько авторов, и каждый автор написал по несколько книг). При возникновении такой необходимости в классе, описывающем сущность, нужно объявить список связанных с ней сущностей. Информацию о необходимости применения связи «многие-ко-многим» необходимо указать в конфигурационным файла в теге collection (см. листинг 1).

Листинг 2 – Пример работы с сущностью «Книга»

```
DaoImpl<Book> bookDao = new DaoImpl<>(configure, Book.class);
Book book = bookDao.read(21);
/* создание proxy-объекта для объекта класса Book */
book = (Book)ProxyBuilder.build(configure, book, Book.class);
```

Заключение. ORM-библиотека разрабатывалась с применением возможностей ретроспекции (Reflection API) языка программирования Java. Разработанная библиотека позволяет автоматизировать процесс взаимодействия с реляционными БД, а именно позволяет избавиться от необходимости написания кода для непосредственной работы с БД, предоставляет функционал для выполнения CRUD операций и позволяет программисту оперировать элементами языка программирования (классами, объектами), а не элементами реляционной модели данных.

1. Lazy Loading Design Pattern [Электронный ресурс] / GeeksforGeeks – 2019. – Режим доступа: https://www.geeksforgeeks.org/lazy-loading-design-pattern/. – Дата доступа: 05.05.2019.
2. ORM (Object-Relational Mapping) [Электронный ресурс] / Национальная библиотека им. Н. Э. Баумана – 2019. – Режим доступа: https://ru.bmstu.wiki/ORM_(Object-Relational_Mapping). – Дата доступа: 18.04.2019.
3. The Reflection API [Electronic resource] / ORACLE. – 2017. – Mode of access: https://docs.oracle.com/javase/tutorial/reflect/. – Data of access: 29.03.2019.