Spring Start Here

Chapter-14: Implementing data persistence with Spring Data

Upcode Software
Engineer Team

-2024-

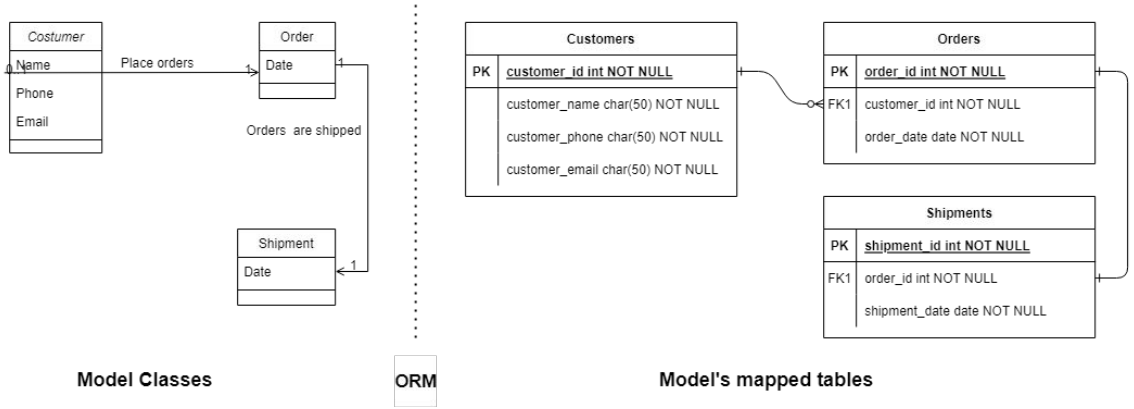
CONTENT

1. Introduction
2. How Spring Data works
3. Defining Spring Data repositories
4. Using Spring Data JDBC to implement a Spring app’s persistence layer

INTRODUCTION(1/5)

Что такое ORM?

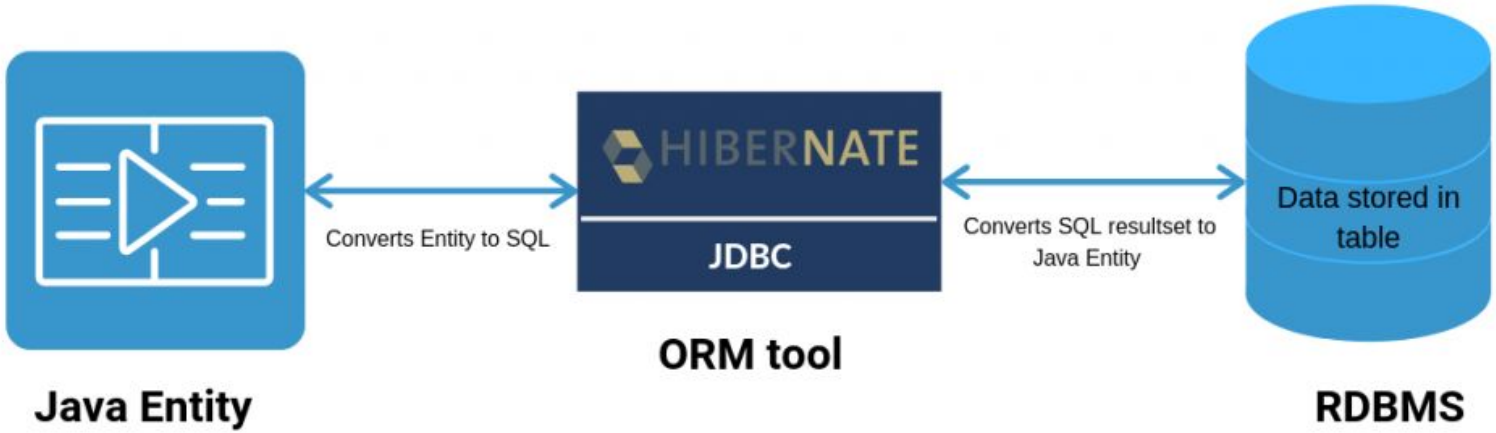
- Инструмент объектно-реляционного отображения(ORM), представляет собой фреймворк, который может помочь и упростить перевод между двумя парадигмами: объектами и таблицами реляционной базы данных;
- Он может использовать определения классов (модели) для создания, обслуживания и обеспечения полного доступа к данным объектов и их сохраняемости в базе данных.



INTRODUCTION(2/5)

Возможности ORM

- Загрузка связанных объектов "по требованию" (lazy loading)
- Обеспечение пессимистической/оптимистической блокировок
- Кэширование загруженных объектов
- SQL-подобные запросы по объектной модели



INTRODUCTION(3/5)

Преимущества ORM

- Нет необходимости писать рутинные insert/update/delete/select для CRUD операций
- Условия связи между объектами (строками таблиц) указываются декларативно в одном месте.
- Возможность использовать полиморфные запросы для иерархий классов
- Высокая степень независимости от конкретной СУБД

INTRODUCTION(4/5)

Недостатки ORM

- Возможны проблемы с производительностью для сложных запросов на объектном SQL.
- Затрудняет использование специфических конструкций языка
- SQL конкретной СУБД.

INTRODUCTION(5/5)

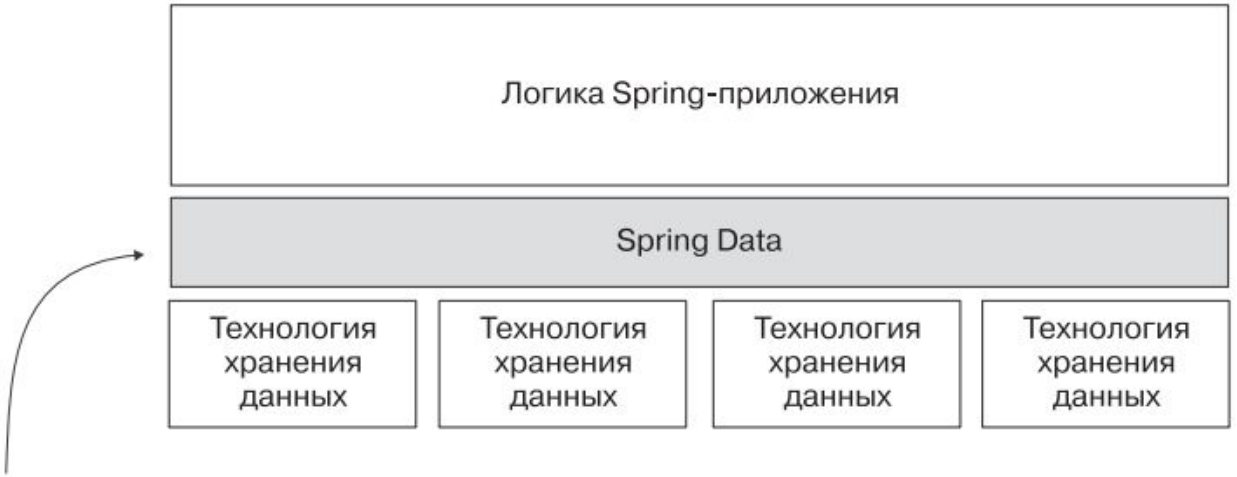
Стандарты ORM

- **EJB 1.1 Entity Beans**
- **Java Data Object (JDO)**
  - JPOX
  - OpenAccess JDO
- **EJB 3.0 Persistence API**
  - Hibernate
  - Oracle TopLink

2.ЧТО ТАКОЕ SPRING DATA

- В экосистеме Java есть множество различных технологий хранения данных, каждая из которых применяется особым способом.
- У каждой из них — свои абстракции и структура классов.
- Spring Data обеспечивает общий уровень абстракций, расположенный над всеми этими технологиями и позволяющий упростить их использование.

2.ЧТО ТАКОЕ SPRING DATA



Spring Data — это надстройка, которая упрощает реализацию хранения данных путем унификации различных технологий хранения данных под общими абстракциями

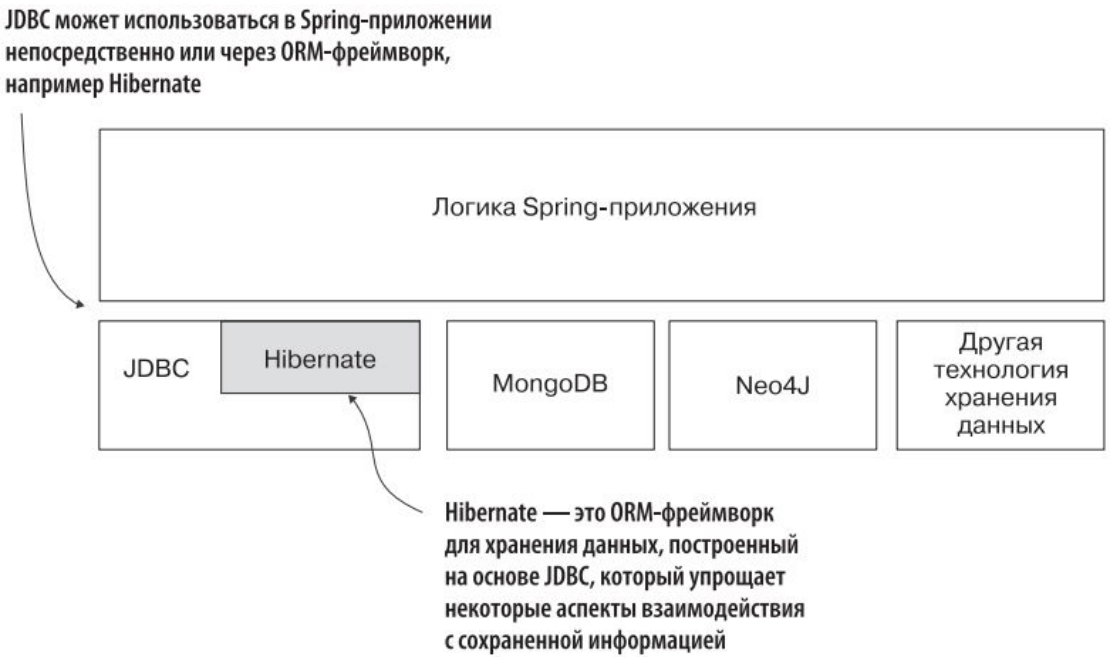
2.ЧТО ТАКОЕ SPRING DATA

Есть много способов внедрения уровня хранения данных. Приложение может соединяться с СУБД напрямую посредством JDBC либо использовать другие библиотеки для подключения к NoSQL-реализациям, таким как MongoDB, Neo4J и другие технологии хранения данных

2.ЧТО ТАКОЕ SPRING DATA

- Соединение с реляционной СУБД посредством JDBC не единственный способ реализации уровня хранения данных в приложении.
- На практике вы будете использовать и другие варианты, причем для каждого из них есть своя библиотека и набор API, которые придется изучить. Такое многообразие сильно усложняет жизнь

2.ЧТО ТАКОЕ SPRING DATA



2.ЧТО ТАКОЕ SPRING DATA

- В некоторых приложениях используются фреймворки, такие как Hibernate, которые являются надстройкой над JDBC.
- Их многочисленность делает реализацию уровня хранения данных еще более запутанной. Хотелось бы избежать этих сложностей в наших приложениях. Ниже вы узнаете, как Spring Data может помочь нам в этом

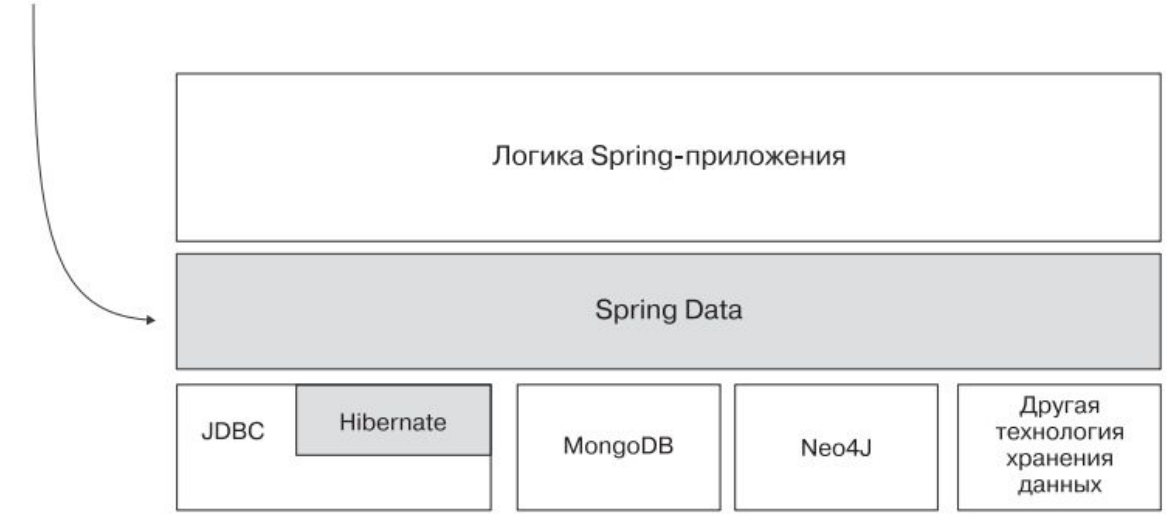
2.ЧТО ТАКОЕ SPRING DATA

Spring Data упрощает реализацию уровня хранения данных следующими способами:

- Предоставляет общий набор абстракций (интерфейсов) для различных технологий хранения данных. Это обеспечивает один и тот же подход при использовании этих технологий;
- Позволяет пользователю создавать операции с сохраненными данными, используя только абстракции, реализации которых предоставляет Spring Data. Таким образом приходится писать меньше кода и можно быстрее настроить функции приложения. Кроме того, благодаря меньшему количеству строк приложение становится более понятным и его проще поддерживать.

2.ЧТО ТАКОЕ SPRING DATA

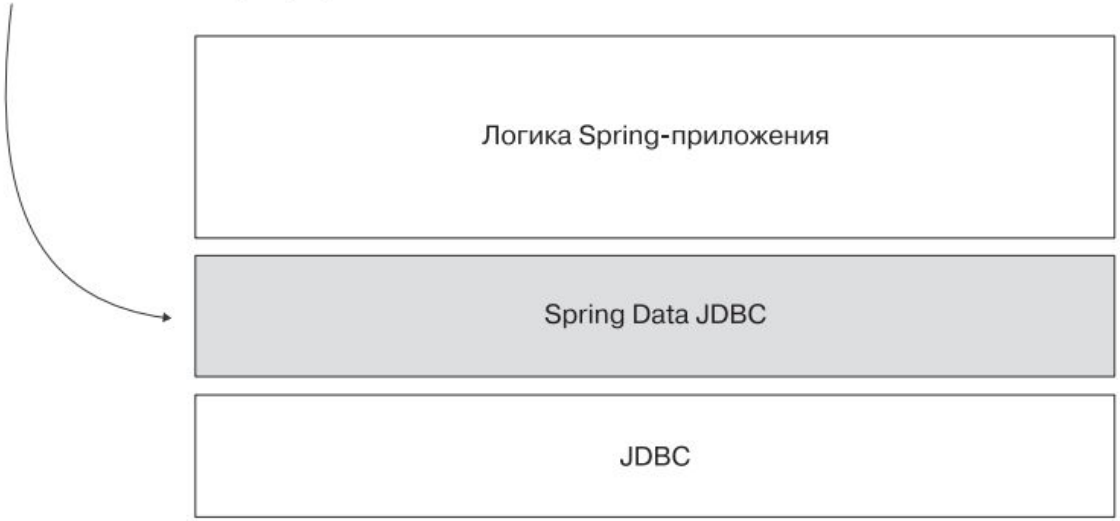
Spring Data — это высокоуровневая надстройка, которая упрощает реализацию хранения данных, так как объединяет различные технологии под одними и теми же абстракциями



Spring Data упрощает реализацию уровня хранения данных, предоставляя общий набор абстракций для разных технологий.

3.КАК РАБОТАЕТ SPRING DATA

Приложение использует только одну из технологий хранения данных, поэтому необходимо подключить тот модуль Spring Data, который ей соответствует. Если в приложении используется JDBC, то нужна зависимость для модуля Spring Data JDBC

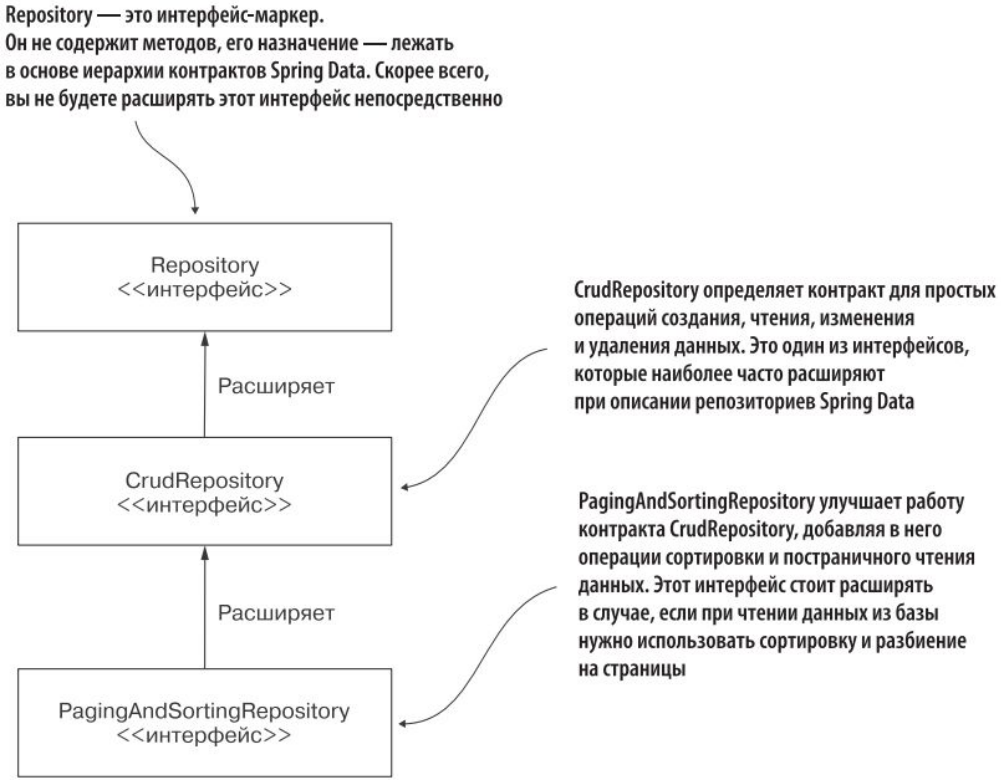


3.КАК РАБОТАЕТ SPRING DATA

Spring Data предоставляет один и тот же набор интерфейсов (контрактов), которые нужно расширить, чтобы описать соответствующие функции:

Repository	CrudRepository	PagingAndSortingRepository
Наиболее абстрактный контракт. Если его расширить, приложение распознает интерфейс, написанный как любой репозиторий Spring Data. Однако этот интерфейс не сможет унаследовать какие-либо предопределенные операции (такие как создание записи, чтение всех записей или чтение записи по первичному ключу). В интерфейсе Repository не объявлен ни один метод;	Простейший контракт Spring Data, который, кроме всего прочего, предоставляет некоторые функции по взаимодействию с сохраненными данными. Если его расширить, чтобы эти функции определить, то станут доступны простейшие операции создания, чтения, изменения и удаления записей;	Расширяет CrudRepository, добавляя операции сортировки и чтения определенного количества записей (постранично).

3.КАК РАБОТАЕТ SPRING DATA



3.КАК РАБОТАЕТ SPRING DATA

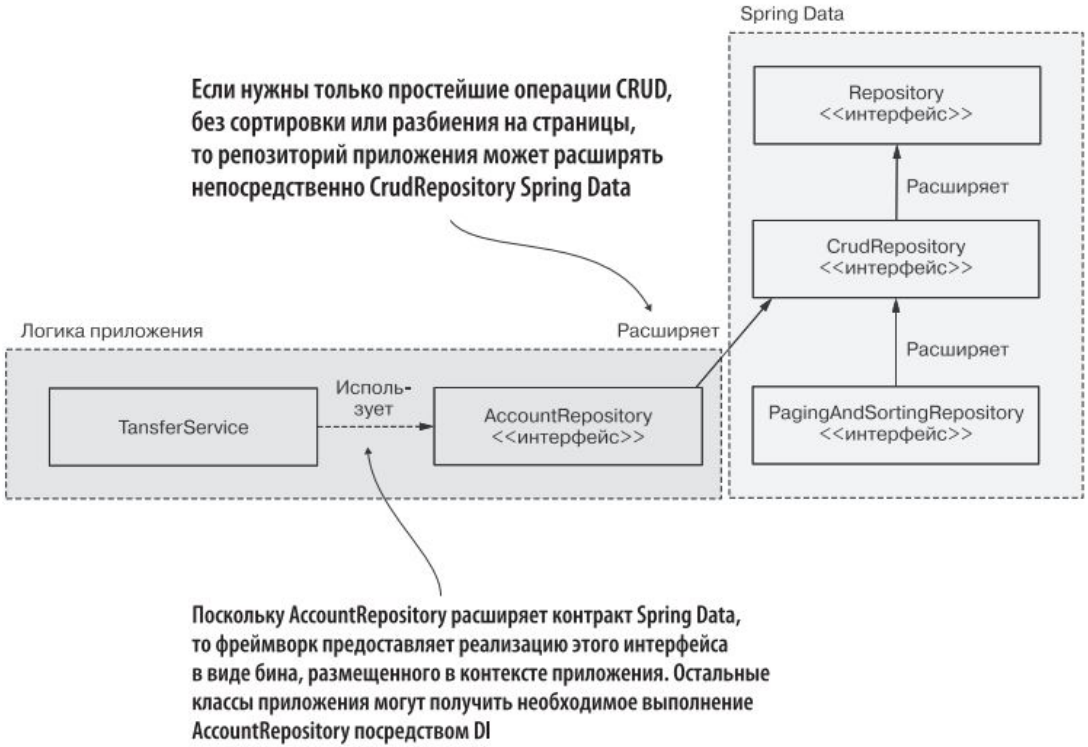
Анализ:
Для реализации репозиториев приложения с помощью Spring Data нужно расширить определенный интерфейс. Основными интерфейсами, представляющими контракты Spring Data, являются Repository, CrudRepository и PagingAndSortingRepository. Для подключения функций приложения по взаимодействию с сохраненными данными нужно расширить один из этих контрактов

3.КАК РАБОТАЕТ SPRING DATA

Примечание!

Не путайте аннотацию @Repository, описанную в главе 4, с интерфейсом Repository из проекта Spring Data. @Repository — это стереотипная аннотация, применяемая к классам, экземпляры которых Spring должен добавить в контекст приложения. Интерфейс Repository, о котором идет речь в этой главе, относится только к Spring Data, и, как вы узнаете, чтобы определить репозиторий Spring Data, нужно расширить этот интерфейс или же другой интерфейс, расширяющий интерфейс Repository

3.КАК РАБОТАЕТ SPRING DATA

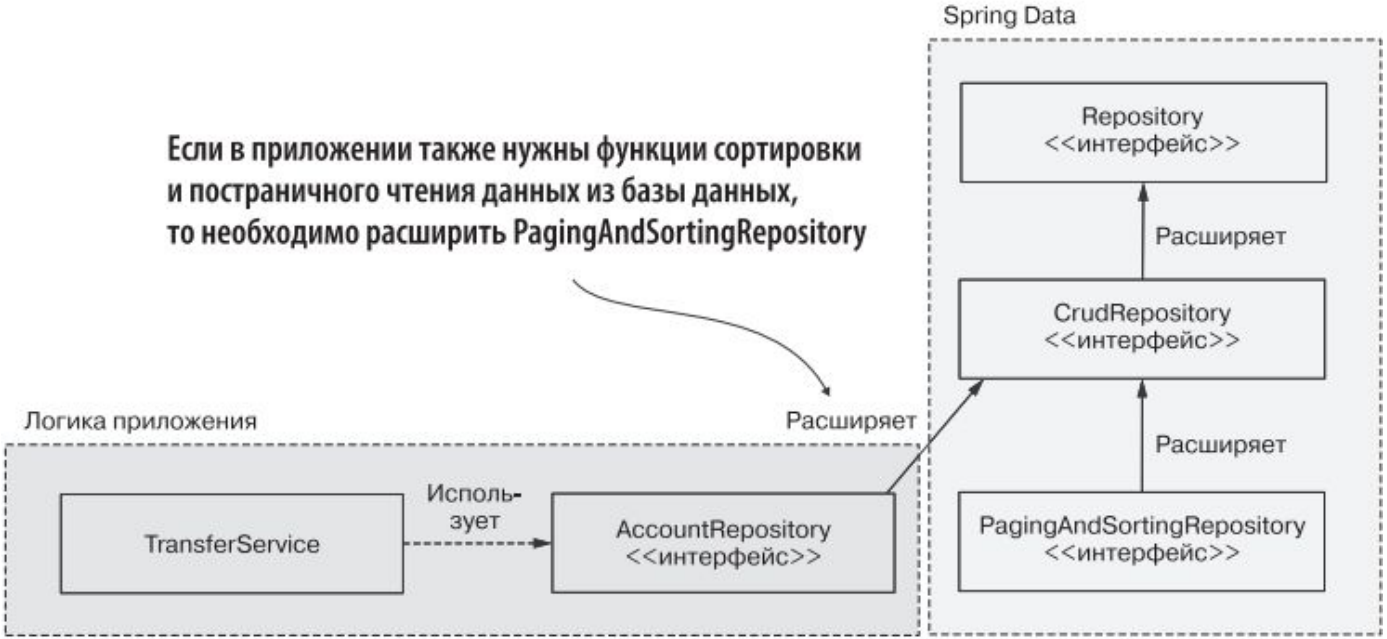


3.КАК РАБОТАЕТ SPRING DATA

Анализ:
Чтобы создать репозиторий Spring Data, нужно определить интерфейс, расширяющий один из контрактов Spring Data. Например, если в приложении понадобятся только операции CRUD, интерфейс, определенный как репозиторий, должен быть расширением CrudRepository. В контекст Spring-приложения добавляется бин, который реализует выбранный контракт, так что остальным заинтересованным компонентам приложения достаточно внедрить этот бин из контекста

3.КАК РАБОТАЕТ SPRING DATA

Расширение контракта CrudRepository для операции CRUD(1/2)



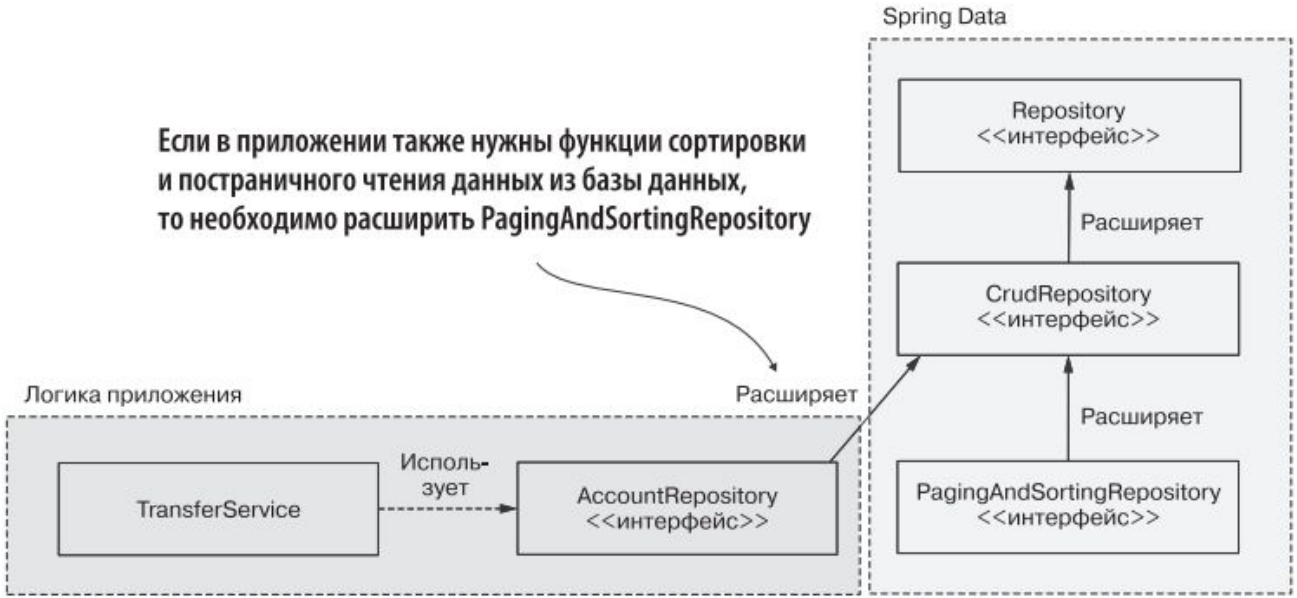
3.КАК РАБОТАЕТ SPRING DATA

Расширение контракта CrudRepository для операции CRUD(2/2)

Анализ:
Если в приложении нужны функции сортировки и постраничного чтения, необходимо расширить более специализированный контракт. В приложении создается бин, реализующий этот контракт. Затем бин может быть внедрен во все остальные компоненты приложения, которые должны его использовать

3.КАК РАБОТАЕТ SPRING DATA

Расширение контракта PagingAndSortingRepository для функции сортировки и постраничного чтения данных(1/2)



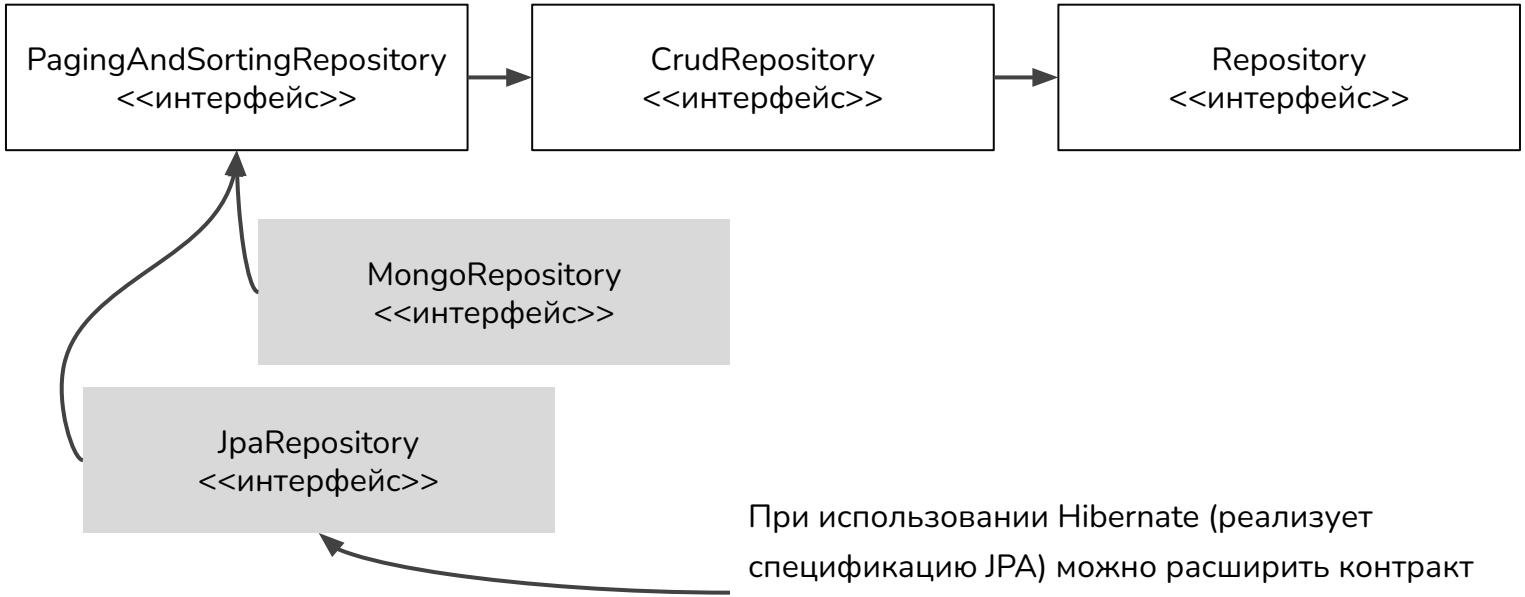
3.КАК РАБОТАЕТ SPRING DATA

Расширение контракта PagingAndSortingRepository для функции сортировки и постраничного чтения данных(2/2).

Анализ:
Если в приложении нужны функции сортировки и постраничного чтения, необходимо расширить более специализированный контракт. В приложении создается бин, реализующий этот контракт. Затем бин может быть внедрен во все остальные компоненты приложения, которые должны его использовать

3.КАК РАБОТАЕТ SPRING DATA

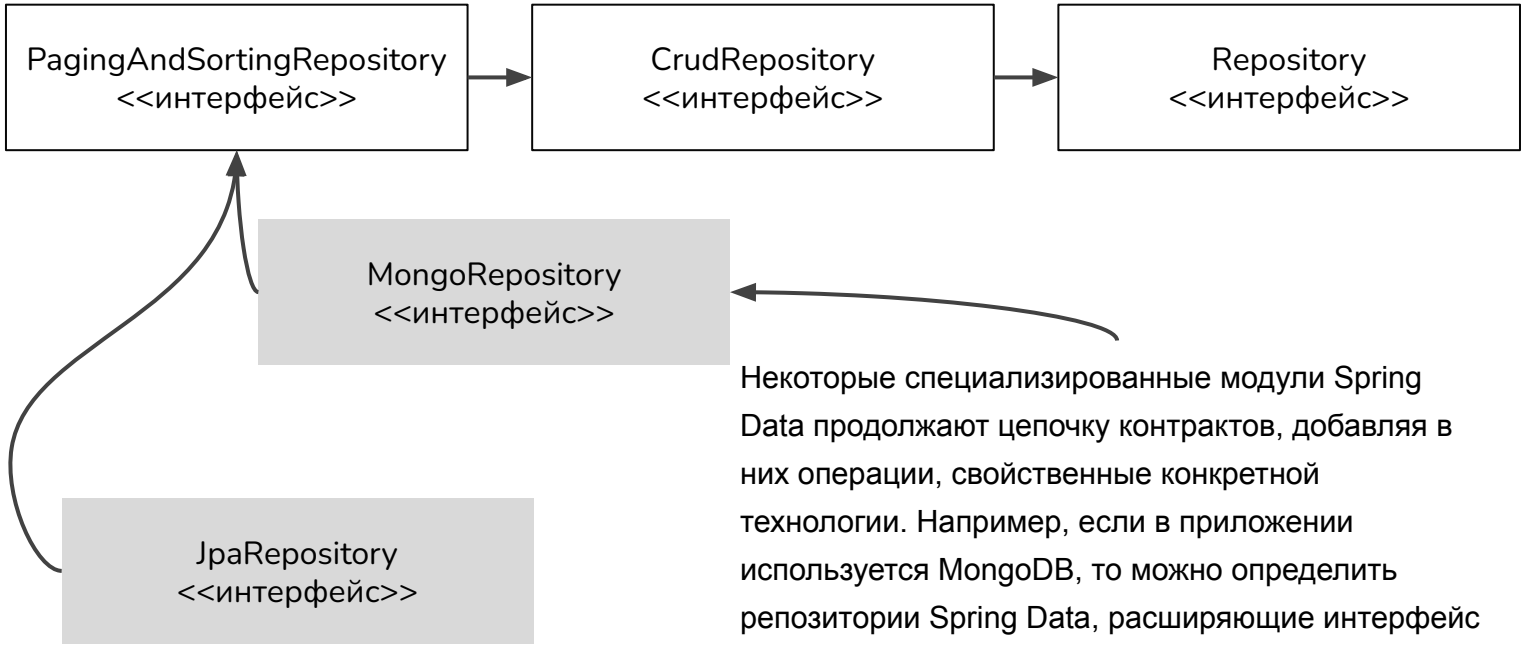
Расширение контракта JpaRepository для Jakarta Persistence API(1/2)



При использовании Hibernate (реализует спецификацию JPA) можно расширить контракт JpaRepository, который добавляет в приложение операции, присущие технологии JPA.

3.КАК РАБОТАЕТ SPRING DATA

Расширение контракта JpaRepository для Jakarta Persistence API(1/2)



Некоторые специализированные модули Spring Data продолжают цепочку контрактов, добавляя в них операции, свойственные конкретной технологии. Например, если в приложении используется MongoDB, то можно определить репозитории Spring Data, расширяющие интерфейс MongoRepository из модуля Spring Data Mongo

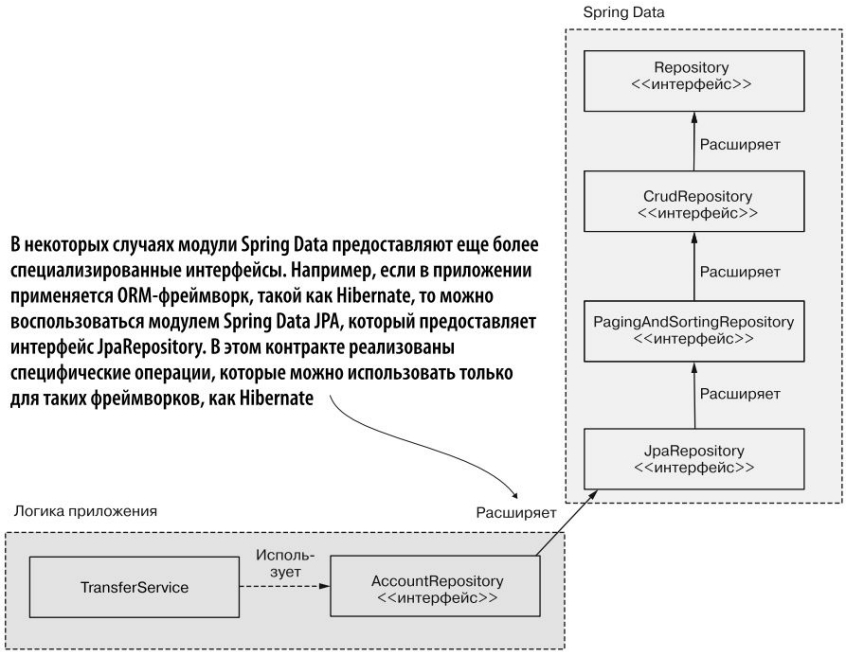
3.КАК РАБОТАЕТ SPRING DATA

Расширение контракта JpaRepository для Jakarta Persistence API(1/2)

Анализ:
В модулях Spring Data, соответствующих данным технологиям, обычно содержатся специализированные контракты, определяющие операции, которые можно использовать только в рамках этих технологий. Если в приложении есть технологии, то в нем, скорее всего, будут и необходимые контракты

3.КАК РАБОТАЕТ SPRING DATA

Расширение контракта JpaRepository для Jakarta Persistence API(1/2)



3.КАК РАБОТАЕТ SPRING DATA

Расширение контракта JpaRepository для Jakarta Persistence API(1/2)

Анализ:
В Spring Data есть и другие модули, которые дают еще более специализированные контракты. Например, при использовании ORM-фреймворка (такого как Hibernate, где реализована спецификация JPA) в сочетании со Spring Data можно расширить интерфейс JpaRepository — еще более специализированный контракт, предоставляющий операции, свойственные только таким JPA-реализациям, как Hibernate.

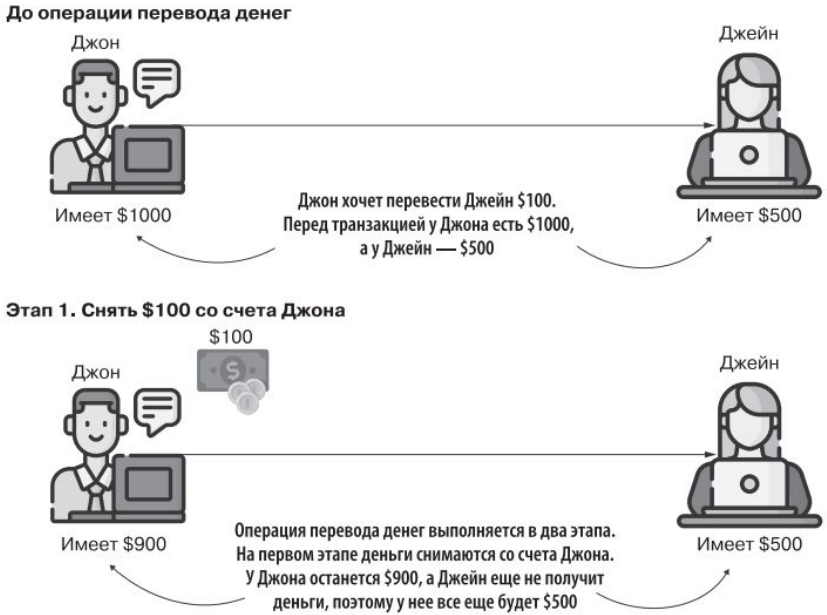
4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Создадим приложение, представляющее собой электронный кошелек для управления счетами пользователей. Пользователь может перевести деньги с одного счета на другой:
1. Снять заданную сумму со счета отправителя.
2. Положить эту сумму на счет получателя.

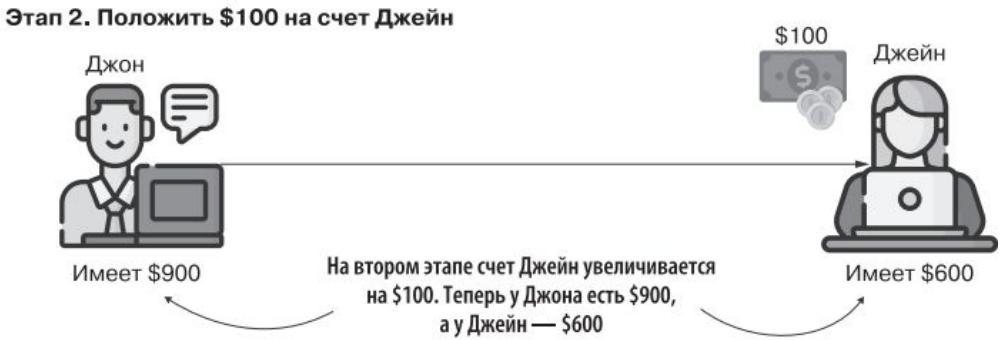
4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.



4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.



Сценарий «перевод денег» выполняется в два этапа. Сначала приложение снимает переводимую сумму со счета отправителя (Джона). Затем приложение кладет переводимую сумму на счет получателя (Джейн)

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Таблица счетов будет состоять из следующих полей:
id — первичный ключ, автоматически увеличивающееся поле типа INT;
name — имя владельца счета;
amount — сумма денег, которую владелец хранит на счету.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Для реализации уровня хранения данных в приложении используем модуль Spring Data JDBC

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Добавим в папку ресурсов проекта Maven файл schema.sql, чтобы создать таблицу счетов в базе данных H2, хранящейся в оперативной памяти. В этом файле содержится следующий DDL-запрос, необходимый для создания таблицы счетов:

```
create table account (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    amount DOUBLE NOT NULL
);
```

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Добавить пару записей в таблицу счетов. Впоследствии, когда мы закончим разработку приложения, эти записи понадобятся для его тестирования. Создадим в папке ресурсов Maven файл data.sql. Чтобы добавить две записи в таблицу счетов, мы внесем в data.sql два оператора INSERT:

```
INSERT INTO account VALUES (NULL, 'Jane Down', 1000);
INSERT INTO account VALUES (NULL, 'John Read', 1000);
```

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Класс Account, представляющий модель записи из таблицы счетов

```
public class Account {

    @Id ◄—— Атрибут, моделирующий первичный ключ, отмечаем аннотацией @Id
    private long id;

    private String name;
    private BigDecimal amount;

    // Геттеры и сеттеры
}
```

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Разработать репозиторий Spring Data. Для этого приложения нам нужны только операции CRUD, поэтому мы напишем интерфейс, который расширяет CrudRepository. Во всех интерфейсах Spring Data содержатся следующие два типа, которые необходимо реализовать:

- класс модели (которую иногда называют сущностью), для которого мы пишем репозиторий;
- тип поля первичного ключа.

Определение репозитория Spring Data:

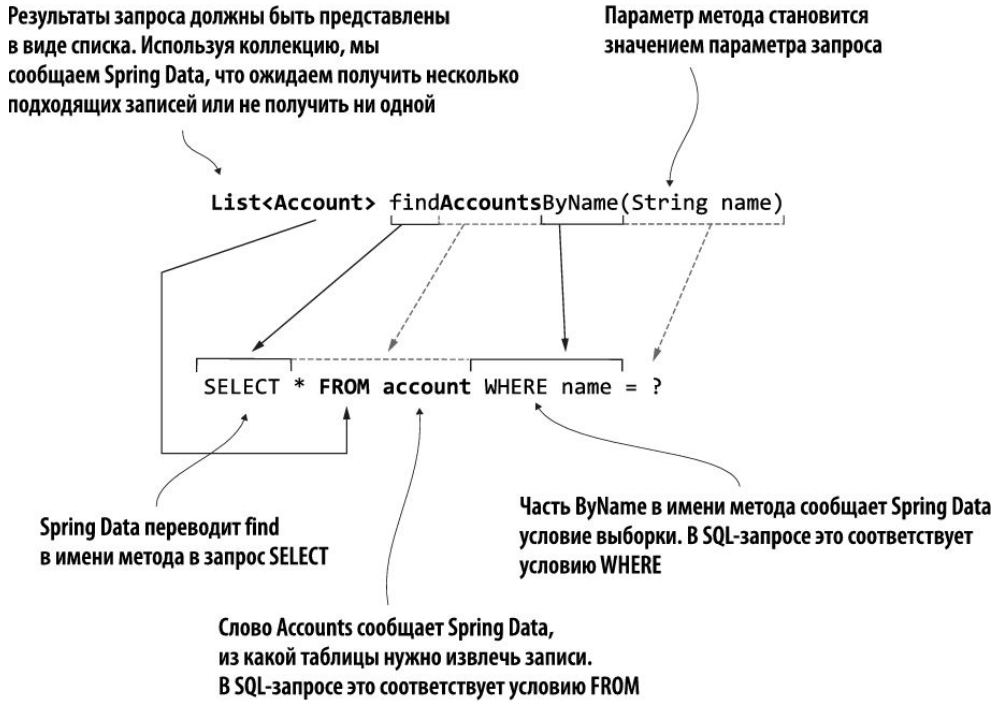
```
public interface AccountRepository
    extends CrudRepository<Account, Long> {

}
```

Первое значение, принадлежащее к параметризованному типу, — это тип класса модели, представляющего таблицу. Второй — тип поля первичного ключа

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Взаимосвязь между именем метода репозитория и запросом, генерируемым Spring Data:



4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Операция репозитория для получения данных обо всех счетах, открытых на заданное имя

```
public interface AccountRepository
        extends CrudRepository<Account, Long> {

    List<Account> findAccountsByName(String name);
}
```

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Главные недостатки Spring Data:

Если операция требует более сложного запроса, имя метода получится слишком длинным и его будет трудно читать	Если при рефакторинге имя метода по ошибке изменят, нарушится поведение приложения, а вы этого можете даже не заметить (к сожалению, не все приложения тщательно тестируются — учитывайте данный факт).	Если только вы не работаете в IDE, где среда сама подсказывает правильное имя метода, правила именования методов в Spring Data придется выучить. Поскольку вы уже знакомы с SQL, то ничего не выиграете, изучив еще и набор правил, применимый только для Spring Data.	Необходимость перевести имя метода в запрос снижает производительность, из-за чего замедляется инициализация приложения (так как процесс перевода происходит при запуске приложения)

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Решение проблемы:

Использование аннотации @Query для определения SQL запроса, необходимого для данной операции.

```
public interface AccountRepository
        extends CrudRepository<Account, Long> {

    @Query("SELECT * FROM account WHERE name = :name")
    List<Account> findAccountsByName(String name);

}
```

Учтите, что имя параметра в запросе должно совпадать с именем параметра в методе. Между двоеточием (:) и именем параметра не должно быть пробела

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Аннотация @Modifying

@Modifying нужен при запросах UPDATE, INSERT или DELETE.

Определение изменяющей операции в репозитории:

```
public interface AccountRepository
    extends CrudRepository<Account, Long> {

    @Query("SELECT * FROM account WHERE name = :name")
    List<Account> findAccountsByName(String name);

    @Modifying
    @Query("UPDATE account SET amount = :amount WHERE id = :id")
    void changeAmount(long id, BigDecimal amount);
}
```

Перед методами, которые определяют операции, изменяющие данные, ставится аннотация @Modifying

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Внедрение репозитория в класс сервиса для реализации сценария использования:

```
@Service
public class TransferService {

    private final AccountRepository accountRepository;

    public TransferService(AccountRepository accountRepository) {
        this.accountRepository = accountRepository;
    }

}
```

Spring Data сам создает динамическую реализацию AccountRepository и добавит бин в контекст приложения.

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Внедрение репозитория в класс сервиса для реализации сценария использования:

```
@Service
public class TransferService {

    private final AccountRepository accountRepository;

    public TransferService(AccountRepository accountRepository) {
        this.accountRepository = accountRepository;
    }

}
```

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Реализация сценария использования «перевод денег»

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Реализация сценария использования «перевод денег»

```
@Service
public class TransferService {

    private final AccountRepository accountRepository;

    public TransferService(AccountRepository accountRepository) {
        this.accountRepository = accountRepository;
    }

    @Transactional
    public void transferMoney(
            long idSender,
            long idReceiver,
            BigDecimal amount) {
```

Помещаем логику сценария использования в рамки транзакции — во избежание рассогласования данных в том случае, если одна из операций завершится неудачно

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Реализация сценария использования «перевод денег»

```
        Account sender =
            accountRepository.findById(idSender)
                .orElseThrow(() -> new AccountNotFoundException());

        Account receiver =
            accountRepository.findById(idReceiver)
                .orElseThrow(() -> new AccountNotFoundException());

        BigDecimal senderNewAmount =
            sender.getAmount().subtract(amount);

        BigDecimal receiverNewAmount =
            receiver.getAmount().add(amount);

        accountRepository
            .changeAmount(idSender, senderNewAmount);

        accountRepository
            .changeAmount(idReceiver, receiverNewAmount);
    }
}
```

Получаем информацию о счетах отправителя и получателя

Вычисляем новые балансы этих счетов, снимая переводимую сумму со счета отправителя и зачисляя ее на счет получателя

Изменяем балансы счетов в базе данных

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

```
public class AccountNotFoundException extends RuntimeException {
}
```

Создание методов сервисов для получения информации о счетах:

```
@Service
public class TransferService {

    // Остальной код

    public Iterable<Account> getAllAccounts() {
        return accountRepository.findAll();
    }

    public List<Account> findAccountsByName(String name) {
        return accountRepository.findAccountsByName(name);
    }
}
```

AccountRepository наследует этот метод от интерфейса CrudRepository, принадлежащего Spring Data

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Предоставление доступа к сценарию использования «перевод денег» посредством конечной точки REST

```
@RestController
public class AccountController {

    private final TransferService transferService;

    public AccountController(TransferService transferService) {
        this.transferService = transferService;
    }

    @PostMapping("/transfer")
    public void transferMoney(
            @RequestBody TransferRequest request
            ) {
        transferService.transferMoney(
            request.getSenderAccountId(),
            request.getReceiverAccountId(),
            request.getAmount());
    }
}
```

Извлекаем из тела HTTP-запроса идентификаторы счетов отправителя и получателя, а также сумму перевода

Вызываем сервис для выполнения сценария использования «перевод денег»

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Реализация DTO TransferRequest, который используется в конечной точке /transfer для преобразования тела HTTP-запроса:

```
public class TransferRequest {

    private long senderAccountId;
    private long receiverAccountId;
    private BigDecimal amount;

    // Геттеры и сеттеры
}
```

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Конечная точка для получения информации о счете:

```
@RestController
public class AccountController {

    // Остальной код

    @GetMapping("/accounts")
    public Iterable<Account> getAllAccounts(
            @RequestParam(required = false) String name
            ) {
        if (name == null) {
            return transferService.getAllAccounts();
        } else {
            return transferService.findAccountsByName(name);
        }
    }
}
```

Используем дополнительный параметр запроса, чтобы передать имя владельца, информацию о счете которого нужно получить

Если мы не передадим имя в виде дополнительного параметра, вернется информация обо всех счетах

Если среди параметров запроса есть имя, мы получим только информацию о счете пользователя с этим именем

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Запустим приложение и проверим записи, вызвав конечную точку /accounts, которая возвращает все счета из базы данных:

```
curl http://localhost:8080/accounts
```

Выполнив эту команду, вы увидите в консоли примерно следующее:

```
[
    {"id":1,"name":"Jane Down","amount":1000.0},
    {"id":2,"name":"John Read","amount":1000.0}
]
```

Чтобы перевести $100 Джону от Джейн, вызовем конечную точку /transfer, используя команду cURL:

```
curl -XPOST -H "content-type:application/json" -d '{"senderAccountId":1,
➥ "receiverAccountId":2, "amount":100}' http://localhost:8080/transfer
```

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Если снова вызовем конечную точку /accounts, увидим, что суммы на счетах изменились. После выполнения перевода у Джейн осталось всего $900, а у Джона теперь есть $1100.

```
curl http://localhost:8080/accounts
```

Результат вызова конечной точки /accounts после перевода денег должен выглядеть так:

```
[
    {"id":1,"name":"Jane Down","amount":900.0},
    {"id":2,"name":"John Read","amount":1100.0}
]
```

Если при вызове конечной точки /accounts использовать параметр запроса name, сформируется другой запрос, который позволит увидеть только счета Джейн:

```
curl http://localhost:8080/accounts?name=Jane+Down
```

4.ИСПОЛЬЗОВАНИЕ SPRING DATA JDBC

Создание приложение электронного кошелька для управления счетами пользователей.

Как показано в следующем фрагменте, в теле ответа, полученного в результате выполнения этой команды, присутствует информация, касающаяся только Джейн:

```
[
    {
        "id": 1,
        "name": "Jane Down",
        "amount": 900.0
    }
]
```

CONCLUSION(1/4)

- Spring Data — это проект экосистемы Spring, который упрощает создание уровня хранения данных в Spring-приложениях. Spring Data предоставляет уровень абстракции, который объединяет многочисленные технологии хранения данных и помогает реализовать данную функцию, предоставляя единый набор контрактов.

- Благодаря Spring Data можно создавать репозитории посредством интерфейсов, которые расширяют следующие стандартные контракты:
  • Repository, не предоставляющий операции с сохраненными данными;
  • CrudRepository, который дает только простейшие операции CRUD — CREATE, READ, UPDATE и DELETE;
  • PagingAndSortingRepository, расширяющий CrudRepository, добавляя операции сортировки и постраничного чтения найденных записей

CONCLUSION(2/4)

- При использовании Spring Data необходимо выбрать определенный модуль в зависимости от применяемой в приложении технологии хранения данных. Например, если приложение соединяется с СУБД посредством JDBC, нужно подключить Spring Data JDBC, а в случае NoSQL, например MongoDB, понадобится Spring Data MongoDB.

- При расширении контракта Spring Data приложение наследует и может использовать операции, определенные в этом контракте. Но можно также создать и дополнительные операции — в виде методов, определенных в интерфейсах репозитория.

- Отметив аннотацией @Query метод, объявленный в репозитории Spring Data, можно определить SQL-запрос, который будет выполняться для данной операции.

CONCLUSION(3/4)

- Если объявить метод без явного указания запроса с помощью аннотации @Query, Spring Data преобразует имя этого метода в SQL-запрос. Чтобы имя метода можно было правильно расшифровать и корректно перевести в запрос, оно должно быть построено в соответствии с правилами Spring Data. Если Spring Data не сможет преобразовать имя метода в SQL-запрос, приложение не будет запущено и выбросит исключение.

- Лучше не полагаться на то, что Spring Data сам преобразует имя метода в запрос, а использовать аннотацию @Query. В случае перевода имен возможны следующие проблемы:
  - Для более сложных операций имена методов получаются слишком длинными и трудно читаемыми, что усложняет поддержку приложения.
  - Преобразование имен замедляет инициализацию приложения.
  - Вам придется выучить соглашения об именовании методов Spring Data.
  - Есть риск нарушить поведение приложения в результате некорректного рефакторинга с изменением имени метода.

CONCLUSION(4/4)

- Любая операция, изменяющая данные (с выполнением запроса INSERT, UPDATE или DELETE), должна сопровождаться аннотацией @Modifying, которая показывает Spring Data, что данная операция модифицирует записи, хранящиеся в базе данных.

REFERENCE

- Baeldung.com
- infourok.ru

Thank you!

Presented by

Moxirbek Maxkamov

(mokhirbek.makhkam@gmail.com)