

Chapter-12:

Using data sources in Spring apps

Uppcode Software
Engineer Team



КОНТЕНТ

1. Что такое источник данных
2. Взаимодействие с сохраненными данными с помощью **JDBCTEMPLATE**
3. Определение источника данных в файле свойств приложение
4. Использование нестандартного бина DataSource
5. **Заключение**
6. **Ссылка**



1. Что такое источник данных(1/10)

- Современные приложения обычно хранят и управляют информацией с помощью баз данных.
- Реляционные базы данных, благодаря своей простоте и эффективности, уже много лет используются во многих сценариях.
- Spring-приложения, как и другие, часто используют базы данных.
- Поэтому важно научиться работать с этими возможностями в Spring-приложениях
- Реляционные базы данных(relational database)-это тип базы данных, где данные хранятся в таблицах, состоящих из строк и столбцов.

1. Что такое источник данных(2/10)

Теперь вы умеете создавать сервисы для обмена данными между Spring-приложениями и их клиентами (например, мобильными приложениями или веб-приложениями, запускаемыми через браузер)

Сейчас вы находитесь здесь! Вы научитесь организовывать хранение данных в Spring-приложениях



REST

REST



Вы изучили принципы работы контекста Spring и аспектов, узнали, каким образом с их помощью функции фреймворка непосредственно подключаются к приложениям

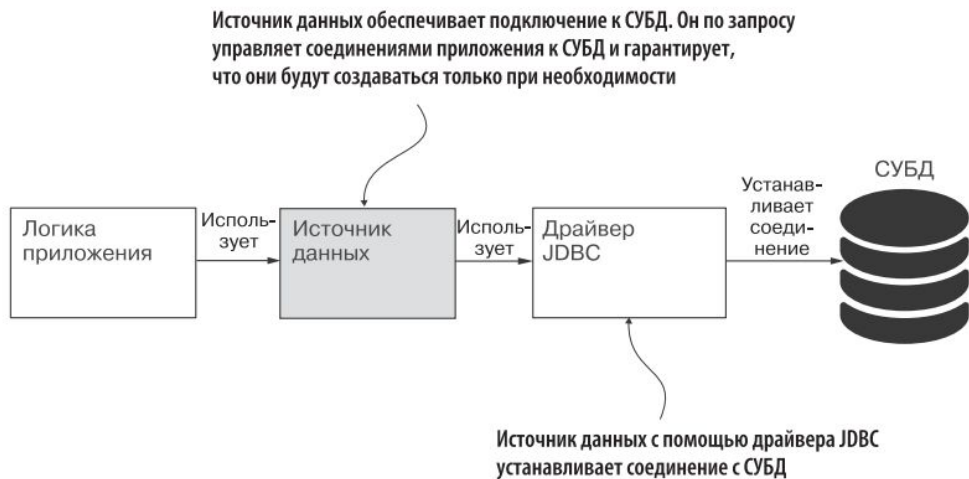


REST

Организация обмена данными между компонентами бэкенда

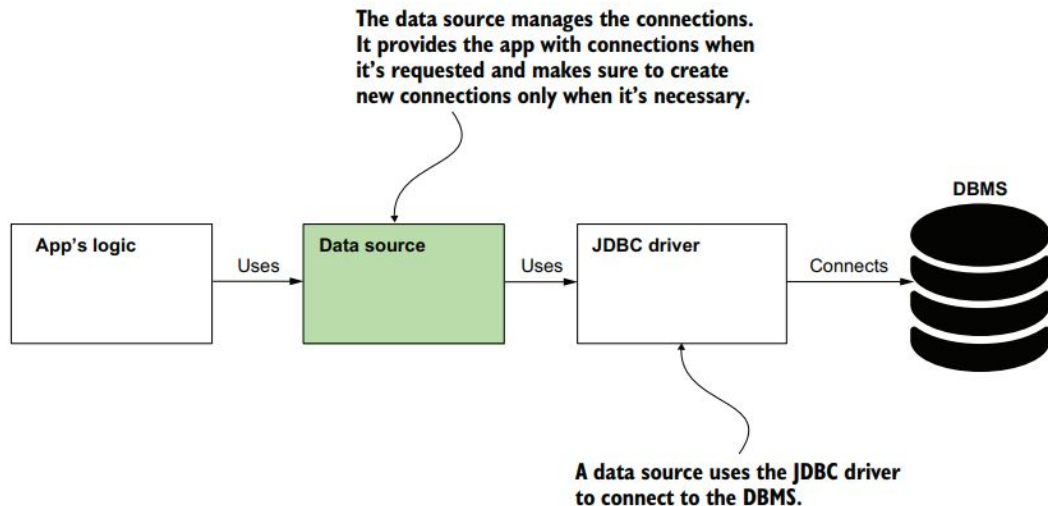
1. Что такое источник данных(3/10)

- СУБД (Система управления базами данных) — это программное обеспечение, предназначенное для эффективного управления данными: их добавления, изменения, получения и обеспечения безопасности. СУБД управляет данными, хранящимися в базе данных.
- База данных — это постоянный набор данных.



1. Что такое источник данных(4/10)

- Источник данных(Data source) — это компонент, который устанавливает и управляет соединениями с СУБД с помощью драйвера JDBC.
- Он повышает производительность приложения за счет повторного использования соединений и закрытия их, когда они больше не нужны.



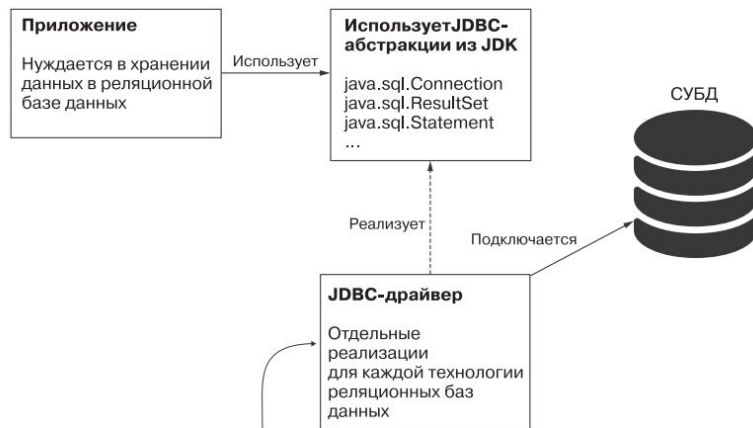


1. Что такое источник данных(5/10)

- В Spring любой инструмент для работы с реляционными базами данных требует наличия источника данных.
- В Java для соединения с реляционной базой данных используется Java Database Connectivity (JDBC).
- JDBC предоставляет абстракции для подключения к СУБД, но требует установки конкретных JDBC-драйверов для работы с определенными технологиями, такими как MySQL, Postgres или Oracle.
- Эти драйверы не входят в состав JDK или Spring и должны быть добавлены отдельно.

1. Что такое источник данных(6/10)

Приложение использует абстракции JDBC из JDK. В приложениях, которым для связи с базой данных нужны только JDBC, обычно применяются такие интерфейсы, как Connection, Statement и ResultSet из пакета java.sql, предоставляемого JDK



Но одних лишь абстракций недостаточно. Приложению нужны их реализации, позволяющие подключаться к конкретной базе данных. JDBC-драйверы обеспечивают такие реализации для разных типов СУБД. Например, если приложение должно подключаться к серверу баз данных MySQL, то нужно установить JDBC-драйвер MySQL, который реализует JDBC-абстракции, предоставляемые JDK, и определяет способ подключения к серверу

1. Что такое источник данных(7/10)

- JDBC-драйвер обеспечивает соединение с СУБД. Один из способов его использования — непосредственный запрос соединения через DriverManager, как в примере:

```
Connection con = DriverManager.getConnection(url, username, password);
```

- Метод getConnection() использует URL для идентификации базы данных и аутентификации с именем и паролем. Однако постоянное создание новых соединений для каждой операции неэффективно, так как это тратит ресурсы и время.





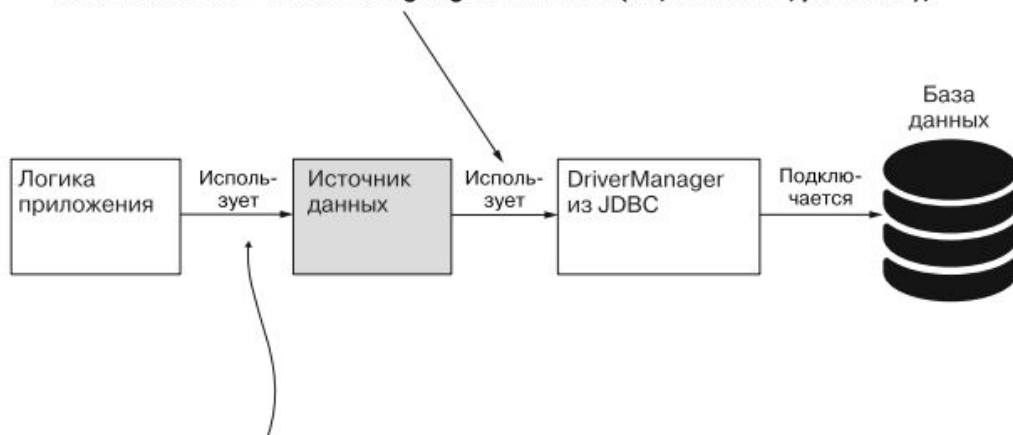
1. Что такое источник данных(8/10)

- Приложение может повторно использовать одно соединение с базой данных. Установка новых подключений без необходимости снижает производительность.
- Для управления соединениями нужен источник данных.
- Источник данных эффективно управляет соединениями, минимизируя ненужные операции.
- Вместо прямого использования менеджера JDBC-драйвера, приложение будет использовать источник данных для установки и управления соединениями.

1. Что такое источник данных(9/10)

- Источник данных — это объект, обязанность которого состоит в управлении соединениями приложения с сервером баз данных. Источник данных гарантирует успешность подключения и повышает производительность операций на уровне хранения данных.

```
Connection con = DriverManager.getConnection(url, username, password);
```



Источник данных управляет соединениями. При необходимости он подключает приложение к серверу баз данных и гарантирует, что новые соединения будут создаваться только тогда, когда это действительно нужно



1. Что такое источник данных(10/10)

- Добавление источника данных в структуру классов экономит время, устраняя лишние операции.
- Источник данных управляет соединениями, устанавливая их по мере необходимости и создавая новые только тогда, когда это нужно.
- Для Java-приложений существует множество вариантов реализации источников данных, но чаще всего используется HikariCP (Hikari connection pool).
- Это программное обеспечение с открытым исходным кодом, к разработке которого вы также можете присоединиться.

```
2024-05-28 11:48:48 INFO [main] com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
2024-05-28 11:48:48 INFO [main] com.zaxxer.hikari.pool.HikariPool - HikariPool-1 - Added connection org.postgresql.jdbc.PgConnection@39a865c1
```

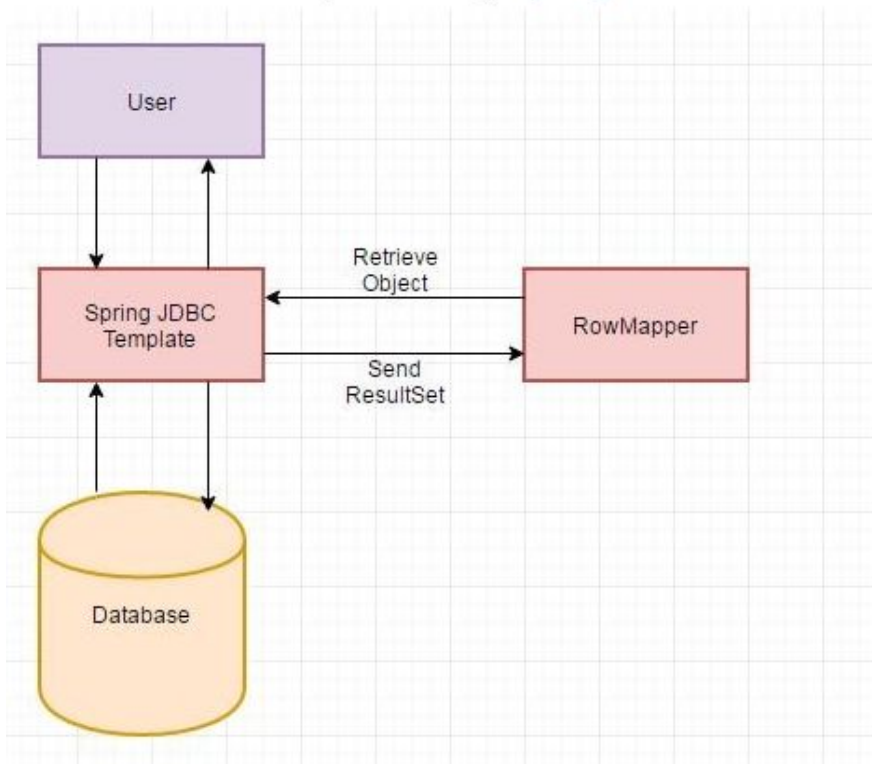


2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(1/15)

```
String sql = "INSERT INTO purchase VALUES (?,?)";  
try (PreparedStatement stmt = con.prepareStatement(sql)) {  
    stmt.setString(1, name);  
    stmt.setDouble(2, price);  
    stmt.executeUpdate();  
} catch (SQLException e) {  
    // сделать что-то в случае исключения  
}
```

- В Spring-приложениях много кода требуется для добавления записи в таблицу, но Spring помогает сократить этот код.
- Существует множество альтернатив для реализации уровня хранения данных, рассмотренных в главах 13 и 14.
- JdbcTemplate — простой инструмент Spring для работы с реляционными базами данных.
- Он подходит для небольших приложений и не требует дополнительных фреймворков, что делает его отличной отправной точкой для изучения уровня хранения данных в Spring-приложениях.

2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(2/15)





2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(3/15)

Рассмотрим работу JdbcTemplate на примере. Выполним следующие операции:

1. Установим соединение с СУБД.
2. Напишем логику для хранилища данных.
3. Вызовем методы хранилища из методов конечных точек REST.

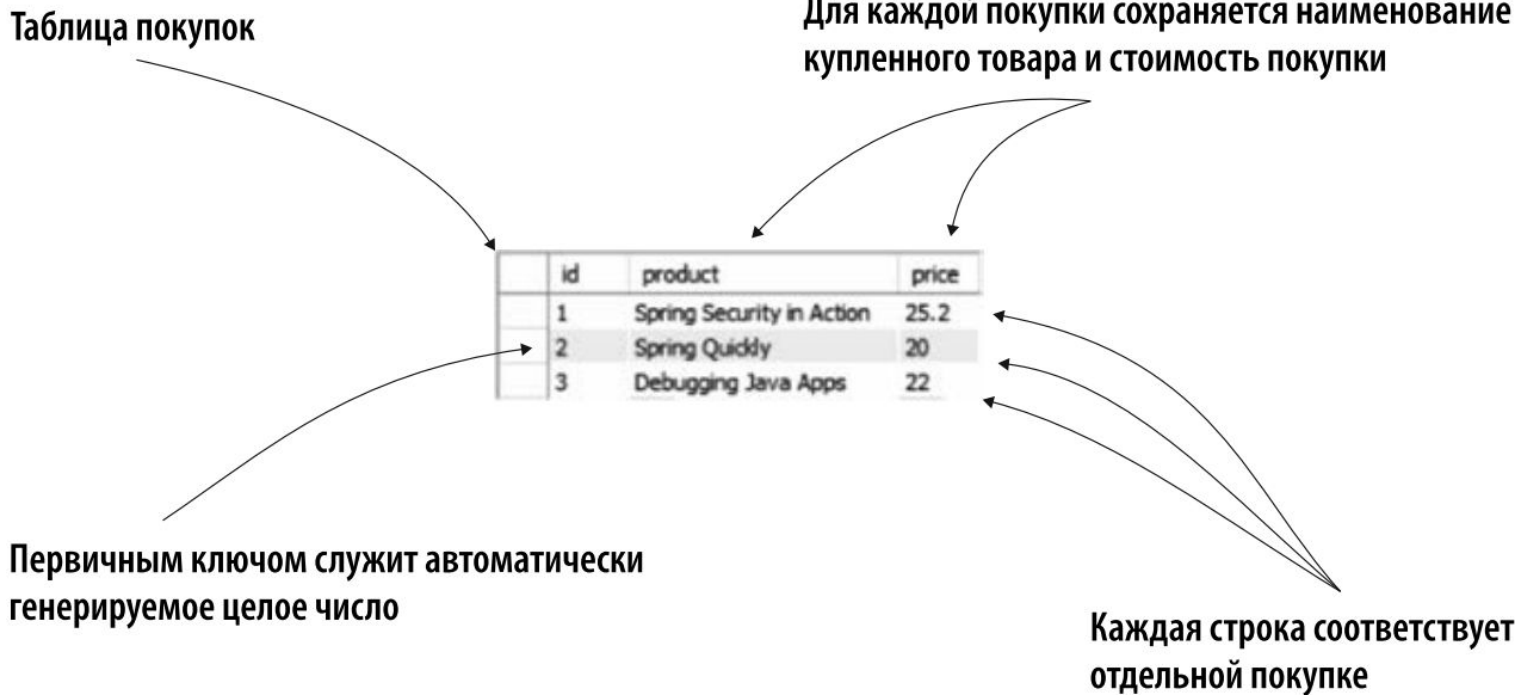
В базе данных создадим таблицу `purchase` для хранения сведений о товарах, приобретенных в онлайн-магазине, и стоимости покупок. Таблица включает столбцы:

- `id` — уникальный первичный ключ, автоматически увеличивающийся;
- `product` — наименование товара;
- `price` — стоимость покупки.

2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(4/15)

Таблица покупок

Для каждой покупки сохраняется наименование купленного товара и стоимость покупки



	id	product	price
	1	Spring Security in Action	25.2
	2	Spring Quiddly	20
	3	Debugging Java Apps	22

Первичным ключом служит автоматически генерируемое целое число

Каждая строка соответствует отдельной покупке



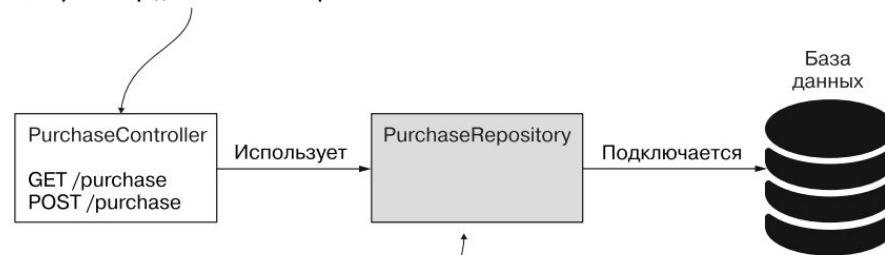
2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(5/15)

- Примеры в этой книге не зависят от конкретной технологии реляционных баз данных и могут использоваться с любой из них.
- Для примеров выбраны H2 (база данных в оперативной памяти, подходит для примеров и интеграционных тестов) и MySQL (бесплатная, легко устанавливаемая на локальных компьютерах).
- Вы можете использовать другие технологии, такие как Postgres, Oracle или MS SQL, при условии использования соответствующего JDBC-драйвера и адаптации SQL-запросов под выбранную СУБД.
- Для базы данных H2 тоже нужен JDBC-драйвер. Но для нее этот драйвер не приходится добавлять специально, он поставляется в комплекте с зависимостью, которую мы добавим в файл pom.xml.

2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(6/15)

- Работая с базой данных, мы представим все возможности уровня хранения данных в виде классов, которые (по соглашению) называются репозиториями.
- Репозиторий — это класс, отвечающий за взаимодействие с базой данных.

PurchaseController — это REST-контроллер. Он предоставляет доступ к двум конечным точкам. Клиенты добавляют новые записи о покупках посредством вызова POST /purchase и получают все существующие в базе данных записи о покупках посредством вызова GET /purchase



PurchaseRepository использует предоставляемый Spring инструмент JdbcTemplate. С помощью источника данных JdbcTemplate подключается к серверу баз данных посредством JDBC

2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(7/15)

- Как обычно, начнем с того, что добавим все необходимые зависимости. В следующем фрагменте кода показано, какие зависимости нужно внести в файл pom.xml:

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

Используем ту же веб-зависимость, что и в предыдущих главах, для создания конечных точек REST

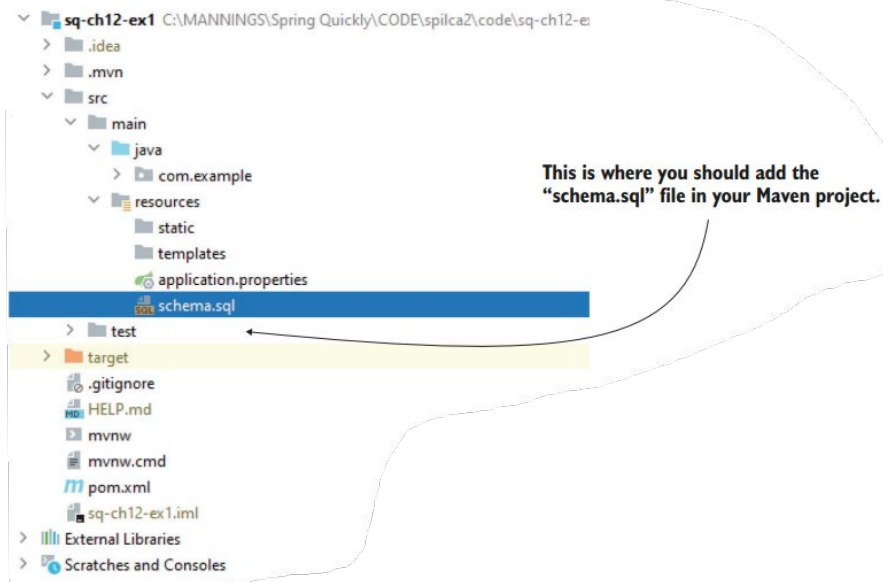
Добавляем JDBC-диспетчер для получения доступа ко всем функциям, необходимым для взаимодействия с базами данных посредством JDBC

Добавляем зависимость H2, чтобы установить базу данных в оперативной памяти, используемую в данном примере, и JDBC-драйвер для доступа к ней

База данных и JDBC-драйвер понадобятся только во время работы приложения, но не на этапе компиляции. Чтобы сообщить Maven, что эти зависимости нужны только при выполнении приложения, добавляем тег <scope> со значением runtime

2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(8/15)

- Зависимость H2 эмулирует базу данных, если у вас нет сервера баз данных. H2 — отличный инструмент для примеров и тестирования приложений, позволяющий проверить функционал продукта без зависимости от базы данных.





2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(9/15)

- Для создания таблицы покупок в теоретических примерах добавим файл **schema.sql** в папку ресурсов проекта Maven. В этом файле будут записаны SQL-запросы, определяющие структуру базы данных (DDL). Пример запроса для создания таблицы покупок:

```
CREATE TABLE IF NOT EXISTS purchase (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    product varchar(50) NOT NULL,  
    price double NOT NULL  
);
```

- Описание структуры базы данных в файле **schema.sql** подходит для теоретических примеров, позволяя сосредоточиться на учебном материале.
- На практике требуется использовать зависимость для управления версиями скриптов взаимодействия с базой данных.



2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(10/15)

- Для описания данных о покупке создайте класс модели. Экземпляры этого класса будут соответствовать строкам таблицы покупок в базе данных, с атрибутами для ID, наименования товара и цены.
- Пример класса модели Purchase:

```
public class Purchase {  
  
    private int id;  
    private String product;  
    private BigDecimal price;  
    // Геттеры и сеттеры  
}
```

2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(11/15)

- Чтобы получить экземпляр `PurchaseRepository` в контроллере, сделаем его бином в контексте Spring с помощью стереотипной аннотации `@Repository`. Это аналогично использованию `@Service` для сервисов. Пример определения класса репозитория:

```
@Repository  
public class PurchaseRepository {  
  
}
```

Добавляем бин этого типа в контекст Spring с помощью стереотипной аннотации `@Repository`

2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(12/15)

- После добавления бина PurchaseRepository в контекст приложения, можно внедрить экземпляр JdbcTemplate для взаимодействия с базой данных. Spring Boot автоматически настраивает источник данных и создает экземпляр JdbcTemplate, когда добавляется зависимость H2 в файл pom.xml.
- При использовании Spring без Spring Boot требуются бины типа DataSource и JdbcTemplate, которые можно добавить в контекст с помощью аннотации @Bean в классе конфигурации.

```
@Repository  
public class PurchaseRepository {
```

```
    private final JdbcTemplate jdbc;
```

```
    public PurchaseRepository(  
        JdbcTemplate jdbc) {
```

```
        this.jdbc = jdbc;
```

```
    }
```

```
}
```

Используем внедрение в конструкторе,
чтобы получить экземпляр JdbcTemplate
из контекста приложения

2. Взаимодействие с сохраненными данными с помощью JdbcTemplate(13/15)

- С имеющимся экземпляром JdbcTemplate можно создать приложение по заданным требованиям. Метод update() в JdbcTemplate позволяет выполнять запросы INSERT, UPDATE и DELETE, принимая SQL-код и параметры. Пример добавления метода storePurchase() в класс PurchaseRepository для создания новой записи в таблице покупок:

```
@Repository
public class PurchaseRepository {

    private final JdbcTemplate jdbc;

    public PurchaseRepository(JdbcTemplate jdbc) {
        this.jdbc = jdbc;
    }

    public void storePurchase(Purchase purchase) {
        String sql = "INSERT INTO purchase VALUES (NULL, ?, ?)";

        jdbc.update(sql,
            purchase.getProduct(),
            purchase.getPrice());
    }
}
```

В качестве параметра метод принимает данные, которые нужно сохранить

Запрос представляет собой строку, в которой вместо значений параметров стоят вопросительные знаки (?). Вместо ID ставим NULL, так как СУБД сама генерирует значение для этого столбца

Метод update() экземпляра JdbcTemplate посылает запрос на сервер баз данных. Первый параметр метода — сам запрос, а остальные — значения параметров запроса. Эти значения в указанной последовательности подставляются в запрос вместо вопросительных знаков



2. Взаимодействие с сохраненными данными с помощью JdbcTemplate(14/15)

Чтобы получать данные из таблицы и преобразовывать их в объекты, необходимо использовать запрос SELECT и RowMapper. RowMapper преобразует строки из ResultSet в объекты модели, такие как Purchase. Вот пример:

1. Напишите и отправьте SELECT-запрос на сервер с помощью JdbcTemplate.
2. Создайте RowMapper для преобразования строки таблицы в объект Purchase.
3. Используйте RowMapper для получения и преобразования данных из базы в объекты Purchase.
 - С помощью RowMapper JdbcTemplate преобразует ResultSet в список экземпляров Purchase. Для каждой строки ResultSet JdbcTemplate вызывает RowMapper, преобразующий строку в экземпляр Purchase.

2. Взаимодействие с сохраненными данными с помощью JDBCTEMPLATE(15/15)

Метод возвращает записи, полученные из базы данных, в виде списка объектов Purchase

@Repository

```
public class PurchaseRepository {
```

```
// Остальной код
```

Определяем запрос SELECT для получения всех записей из таблицы покупок

```
    public List<Purchase> findAllPurchases() {  
        String sql = "SELECT * FROM purchase";
```

```
        RowMapper<Purchase> purchaseRowMapper = (r, i) -> {
```

```
            Purchase rowObject = new Purchase();  
            rowObject.setId(r.getInt("id"));  
            rowObject.setProduct(r.getString("product"));  
            rowObject.setPrice(r.getBigDecimal("price"));  
            return rowObject;
```

```
        };
```

```
        return jdbc.query(sql, purchaseRowMapper);
```

```
    }
```

```
}
```

Создаем объект RowMapper, который сообщает JdbcTemplate, как преобразовать строку, полученную из базы данных, в объект Purchase. Параметр r — лямбда-выражения соответствует ResultSet (данным, полученным из базы), а параметр i — целое число, показывающее номер строки

Заносим данные в экземпляр Purchase. JdbcTemplate будет выполнять эту логику для каждой строки из набора результатов

Отправляем запрос SELECT, используя метод query(), и передаем объект преобразователя строк, чтобы JdbcTemplate знал, как преобразовать полученные данные в объекты Purchase

3. Определение источника данных в файле свойств приложения(1/2)

Пункт 1:

- Для начала подключим приложение к СУБД MySQL.
- Для внесения изменений нам нужно выполнить следующее.
 1. Изменить зависимости проекта, убрав оттуда H2 и добавив соответствующий JDBC-драйвер.
 2. Добавить в файл application.properties свойства для соединения с новой

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```

Добавляем JDBC-драйвер MySQL
как зависимость, которая
подключается при запуске
приложения

3. Определение источника данных в файле свойств приложения(2/2)

Для выполнения пункта 2:

В application.properties добавьте:

- spring.datasource.url для указания URL базы данных.
- spring.datasource.username и spring.datasource.password для аутентификации.
- spring.datasource.initialization-mode=always для использования файла schema.sql при создании таблицы.

Это обеспечит корректное подключение и инициализацию базы данных MySQL.

Указываем URL, соответствующий местоположению базы данных

```
spring.datasource.url=jdbc:mysql://localhost/spring_quickly?  
useLegacyDatetimeCode=false&serverTimezone=UTC
```

```
spring.datasource.username=<dbms username>  
spring.datasource.password=<dbms password>  
spring.datasource.initialization-mode=always
```

Настраиваем параметры доступа
для аутентификации и соединения с СУБД

Включаем режим инициализации always, чтобы
Spring Boot выполнил запросы из файла schema.sql



4. Использование нестандартного бина DataSource(1/2)

Для ручного создания бина DataSource в Spring Boot:

1. Используйте **application.properties** для указания информации о соединении. Это часто достаточно.
2. Ситуации, требующие ручного создания DataSource:
 - Необходима специфическая реализация DataSource.-----
 - Подключение к нескольким базам данных.
 - Определение специфических параметров DataSource.
 - Использование **Spring без Spring Boot**.
3. Определите бин DataSource вручную:
 - Создайте класс конфигурации.
 - Добавьте метод с аннотацией `@Bean`, возвращающий экземпляр DataSource.
 - Это позволяет полностью контролировать процесс создания DataSource.

4. Использование нестандартного бина DataSource(2/2)

```
@Configuration
public class ProjectConfig {

    @Value("${custom.datasource.url}")
    private String datasourceUrl;

    @Value("${custom.datasource.username}")
    private String datasourceUsername;

    @Value("${custom.datasource.password}")
    private String datasourcePassword;

    @Bean
    public DataSource dataSource() {
        HikariDataSource dataSource =
            new HikariDataSource();

        dataSource.setJdbcUrl(datasourceUrl);
        dataSource.setUsername(datasourceUsername);
        dataSource.setPassword(datasourcePassword);
        dataSource.setConnectionTimeout(1000);

        return dataSource;
    }
}
```

← Параметры соединения могут изменяться, поэтому имеет смысл и дальше указывать их отдельно от кода приложения. В данном примере они хранятся в файле application.properties

← Метод возвращает объект DataSource. Если Spring Boot обнаруживает, что в контексте Spring уже есть DataSource, новый он не создает

← Ставим перед методом аннотацию @Bean, чтобы Spring добавил возвращаемое значение в контекст

← В качестве источника данных в этом примере мы будем использовать HikariCP. Но если проект требует чего-то другого, самостоятельно создавая бин, вы можете выбрать любой другой источник данных

← Устанавливаем параметры соединения для источника данных

← Вы можете определить и другие параметры (которые, возможно, понадобятся при определенных условиях). В данном случае я в качестве примера использовал время ожидания подключения (сколько времени источник данных будет ждать установки соединения, прежде чем решит, что оно не удалось)

← Возвращаем экземпляр DataSource, который Spring внесет в контекст



4. Заключение(1/2)

1. JDBC-драйвер:

- Это зависимость, необходимая для соединения Java-приложения с реляционной базой данных.
- JDBC-драйвер обеспечивает абстракции объектов для взаимодействия с базой данных.

2. Источник данных (DataSource):

- Объект, управляющий соединениями с сервером базы данных.
- Spring Boot по умолчанию использует источник данных под названием HikariCP, который оптимизирует соединения с базой данных.
- При необходимости можно использовать другие технологии для источника данных, если они лучше подходят для вашего проекта.

3. JdbcTemplate:

- Инструмент Spring, упрощающий доступ к реляционной базе данных через JDBC-драйвер.
- Для изменения данных в таблице используется метод **update()** объекта JdbcTemplate.
- Для получения данных из таблицы с помощью запросов SELECT используются методы **query()** объекта JdbcTemplate.



4. Заключение(2/2)

1. Создание своего источника данных:

- Можно создать кастомный бин типа `java.sql.DataSource`, который будет использоваться вместо источника данных по умолчанию.
- Для этого необходимо объявить бин в контексте Spring.

2. Несколько баз данных:

- Для соединения с несколькими базами данных можно создать несколько объектов источников данных и соответствующих им объектов `JdbcTemplate`.
- Чтобы различать объекты одного типа, используется аннотация **@Qualifier**.

Таким образом, Spring Boot обеспечивает гибкость и удобство в работе с реляционными базами данных, предоставляя возможность использовать источники данных и `JdbcTemplate` по умолчанию или создавать свои собственные конфигурации для более сложных и оптимизированных сценариев.



REFERENCE

1: [Spring Start Here](#)

Resources





Thank you!

Presented by

Asadbek Quronboyev

(asadbek9805@gmail.com)