Разработка серверного ПО. Лекция 5. Обращение к эндпоинтам других сервисов. Источники данных Spring

RestTemplate

Начиная с версии Spring 5, этот инструмент переведен в режим поддержки, поэтому однажды устареет. RestTemplate используется для вызова эндпоинтов REST в большинстве существующих сегодня проектов Spring — когда эти проекты зарождались, данный инструмент был единственным или наилучшим вариантом для реализации нужного функционала. Для некоторых из них возможностей RestTemplate вполне достаточно и сейчас, поэтому замена не имеет смысла. В других случаях внедрение нового решения потребовало бы слишком много времени и затраты бы не окупились. В любом случае Spring-разработчик все еще обязан уметь взаимодействовать с этим инструментом.

У RestTemplate есть недостатки:

- выбор синхронной и асинхронной работы;
- приходится писать больше кода и обрабатывать больше исключений (избегать шаблонного кода);
- невозможность выполнять повторные вызовы и операции отката (логика, которая осуществляется, когда приложение по какой-то причине не может выполнить определенный вызов REST).

Чтобы создать вызов, нужно выполнить следующие операции.

1. Описать HTTP-заголовки, создав и настроив конфигурацию экземпляра HttpHeaders.
2. Создать экземпляр HttpEntity, в котором представлены данные запроса (заголовки и тело).
3. С помощью метода exchange() отправить HTTP-вызов и получить HTTP-ответ.

Пример в RestTemplateExample

OpenFeign из Spring Cloud

Для OpenFeign достаточно определить интерфейс (контракт) и сообщить, где этот контракт находится. В соответствии с конфигурацией, описанной аннотациями, OpenFeign создаст реализацию интерфейса в виде бина, помещенного в контекст Spring. Этот бин затем можно будет внедрить в любой объект приложения.

Пример в OpenFeignExample

Источники данных

Источник данных — это компонент, устанавливающий соединение с системой управления базой данных (СУБД). Для установки соединений, которыми он управляет, источник данных использует драйвер JDBC. Назначение источника данных — повысить производительность приложения, позволяя логике приложения повторно использовать соединения с СУБД и устанавливать новые соединения

только при необходимости. Источник данных также обеспечивает закрытие соединений, когда они больше не нужны.

Работа с любым инструментом, имеющим отношение к хранению данных в реляционной базе, в Spring подразумевает наличие источника данных.

В приложениях Java функционал языка для соединения с реляционной базой данных называется Java Database Connectivity (JDBC). JDBC — это способ установить подключение к СУБД для взаимодействия с базой данных. Но в JDK нет отдельных реализаций для работы с конкретными технологиями (такими как MySQL, Postgres или Oracle). JDK только предоставляет абстракции для объектов, необходимых приложению для соединения с базой. Чтобы получить реализацию этой абстракции и позволить приложению подключаться к конкретной СУБД, необходимо добавить так называемый JDBC-драйвер — зависимость реального времени. Каждый производитель дает свой драйвер, который следует подключить к приложению, чтобы оно могло соединяться с СУБД, построенной по соответствующей технологии. JDBC-драйверы не поставляются в комплекте с JDK или фреймворком, таким как Spring.

JDBC-драйвер обеспечивает соединение с СУБД. Первый вариант — использовать его непосредственно, так что приложение будет запрашивать соединение с базой данных всякий раз, когда потребуется выполнить новую операцию с хранящимися там сведениями. Подобный метод часто встречается в учебниках по основам Java.

Для Java-приложений существует множество вариантов реализации источников данных, но в настоящее время чаще всего используется HikariCP (Hikari connection pool — пул соединений Hikari).

JdbcTemplate — это инструмент Spring, позволяющий упростить код, который необходимо написать для доступа к реляционной базе данных посредством JDBC-драйвера. Для соединения с сервером баз данных JdbcTemplate использует источник данных.

Для отправки запроса, изменяющего данные в таблице, применяется метод update() объекта JdbcTemplate. Чтобы получить данные посредством отправки запросов SELECT, используется один из методов query() объекта JdbcTemplate. Как правило, такие операции нужны для изменения или получения сохраненных данных.

Чтобы создать свой источник данных вместо того, который автоматически предлагает Spring Boot, нужен бин типа java.sql.DataSource. Если объявить бин этого типа в контексте Spring, то Spring Boot будет использовать его, а не опцию по умолчанию. То же самое происходит, если необходимо создать нестандартный объект JdbcTemplate. Как правило, мы применяем те объекты, которые Spring Boot определяет по умолчанию, но иногда встречаются ситуации, когда для различных оптимизаций необходимы специальные конфигурации или реализации объектов.

Если приложение должно соединяться с несколькими базами данных, можно создать несколько объектов — источников данных, по одному для каждого JdbcTemplate. В этом случае, чтобы различать объекты одного типа в контексте приложения, необходимо использовать аннотацию @Qualifier.

Транзакции

- Транзакция — это набор операций, изменяющих данные. Эти операции либо выполняются все вместе, либо не выполняются вовсе. В реальных приложениях во избежание несогласованности данных практически любой сценарий использования должен быть представлен в виде транзакции.

- Если одна из операций завершается неудачно, приложение возвращает данные к тому виду, в котором они находились в начале транзакции. В таких случаях принято говорить, что происходит откат транзакции.
- Если все операции выполнены успешно, то говорят, что происходит фиксация транзакции. Это означает, что приложение сохраняет все изменения, совершенные в процессе выполнения сценария использования.
- Для реализации кода транзакции в Spring используется аннотация `@Transactional`. Она ставится перед методом, который должен выполняться в рамках транзакции. Аннотацию `@Transactional` также можно поставить перед классом, и тогда Spring будет считать транзакционными все его методы.
- При выполнении приложения методы с аннотацией `@Transactional` перехватываются специальным аспектом Spring. Этот аспект открывает транзакцию и, если в процессе проведения транзакции возникает исключение, откатывает ее. Если метод не выбрасывает исключение, транзакция фиксируется, и приложение сохраняет сделанные методом изменения.

Spring Data

Spring Data — это проект экосистемы Spring, который упрощает разработку уровня хранения данных, предоставляя реализацию необходимых объектов в соответствии с выбранной технологией хранения данных. Благодаря этому нам, чтобы определить репозитории Spring-приложения, остается написать лишь несколько строк кода.

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.

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.

В Spring Data есть разные модули, предназначенные для соответствующих технологий. Эти модули не зависят друг от друга, и их можно подключать к проекту, используя различные зависимости. Таким образом при разработке приложения вам не нужна зависимость Spring Data. Ее просто не существует. Spring Data предоставляет отдельные зависимости для каждой поддерживаемой им технологии хранения данных. Например, для соединения с СУБД непосредственно через JDBC используется модуль Spring Data JDBC, а для подключения к базе данных MongoDB — Spring Data Mongo.

Какая бы технология хранения данных ни использовалась в приложении, Spring Data предоставляет один и тот же набор интерфейсов (контрактов), которые нужно расширить, чтобы описать соответствующие функции.

- Repository — 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;

- CrudRepository — 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;

- PagingAndSortingRepository — расширяет CrudRepository, добавляя операции сортировки и чтения определенного количества записей (постранично).

В некоторых модулях Spring Data могут быть и другие контракты, в зависимости от технологии, которую они представляют. Например, с помощью Spring Data JPA можно непосредственно расширить интерфейс JpaRepository. JpaRepository — еще более специализированный контракт, чем PagingAndSortingRepository. Он добавляет операции, применимые только при использовании определенных технологий, реализующих спецификацию Jakarta Persistence API (JPA), таких как Hibernate. Другой пример — реализация технологии NoSQL, такой как MongoDB. Чтобы использовать Spring Data для MongoDB, нужно подключить к приложению модуль Spring Data Mongo. Этот модуль предоставляет, в частности, специализированный контракт MongoRepository, который добавляет в приложение операции, свойственные данной технологии хранения информации. Если в приложении используется определенная технология, то в нем обычно расширяются те контракты Spring Data, в которых объявлены операции, свойственные именно этой технологии. Если в приложении не требуется ничего сверх операций CRUD, можно по-прежнему ограничиться расширением CrudRepository, но специализированные контракты, как правило, предоставляют более удобные решения, созданные для конкретной технологии.

В результате расширения интерфейса CrudRepository нам станут доступны простейшие операции Spring Data, такие как получение значения по его первичному ключу, чтение всех записей из таблицы, удаление записей и т. п. Но это далеко не все, что можно реализовать посредством SQL-запросов.

Добавление операций в Spring Data делается очень просто — иногда даже не приходится писать SQL-запрос. Проект интерпретирует имена методов на основе ряда правил для создания таких имен и генерирует запросы. Предположим, нужно написать операцию получения данных для всех счетов по заданному имени. В Spring Data для этого достаточно метода с именем findAccountsByName. Если имя метода начинается с find, Spring Data знает, что вы хотите прочитать что-то из базы с помощью запроса SELECT. Следующее слово, Accounts, сообщает, что именно вы хотите выбрать. Spring Data даже догадается, что вы имели в виду, назвав метод findByName. Он поймет ваше стремление, поскольку этот метод принадлежит репозиторию AccountRepository.

Главные недостатки состоят в следующем:

- 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;

- правила именования методов в Spring Data придется выучить;

- необходимость перевести имя метода в запрос снижает производительность, из-за чего замедляется инициализация приложения (так как процесс перевода происходит при запуске приложения).

Самый простой способ избежать этих проблем состоит в применении аннотации @Query. Она определяет SQL-запрос, который должно выполнить приложение при вызове метода. Если перед методом стоит @Query, то уже неважно, как вы его назовете. Spring Data не станет переводить имя метода в запрос, а возьмет тот запрос, который предоставите вы. Такое поведение также является более эффективным для производительности.

Аннотация @Query используется одинаково для любых запросов. Но, если в запросе содержатся данные, необходимо также снабдить метод аннотацией @Modifying. @Modifying нужен и при запросах UPDATE, INSERT или DELETE.