УДК 004.021 **Информационные технологии**

Макаров Олег Сергеевич, *студент магистратуры*
ФГБОУ ВО «Национальный исследовательский Мордовский государственный университет им. Н. П. Огарёва», Россия, г. Саранск

ВЫБОР ПОДХОДА К СИНХРОНИЗАЦИИ БАЗ ДАННЫХ

Аннотация: В статье описываются подходы к созданию синхронизации баз данных между двумя приложениями. Рассмотрены основные проблемы и технические сложности, предложены рекомендации к преодолению различных ограничений. Описан общий алгоритм подготовки к реализации синхронизации независимо от используемых программных технологий и СУБД.

Ключевые слова: Синхронизация, база данных, сущность, реакция на изменение, передача данных, брокер сообщений

Abstract: This article describes approaches to providing database synchronization between two applications. The main problems and technical difficulties are considered, recommendations for overcoming various limitations are proposed. A general algorithm for preparing for the implementation of synchronization is described, regardless of the software technologies and used DBMS.

Key words: Synchronization, database, entity, reaction for change, data transfer, message broker.

Постановка задачи

Перед программными продуктами часто возникает проблема синхронизации смежной информации. Предположим, есть два приложения, каждое из которых использует собственную базу данных (БД) для своей работы. В общем случае, такие приложения могут быть написаны на разных

языках программирования и могут использовать разные СУБД. Допустим, в обеих БД хранятся похожие по своей структуре объекты, которые необходимо синхронизировать между собой. То есть изменения, произошедшие с сущностью в одной БД, должны отразиться на соответствующей ей сущности в другой БД в течение определенного промежутка времени, и наоборот. Отметим, что такую задачу невозможно решить лишь с помощью встроенных средств СУБД (например, репликации [1]), поскольку может быть задействована какая-либо предобработка данных (сущности не всегда имеют одинаковую структуру) либо иная условная логика.

Определение требований

Для решения вышеописанной проблемы вначале необходимо изучить бизнес-требования и ответить на следующие вопросы:

- Насколько критична потеря данных при синхронизации?
- Насколько критичны задержки синхронизации одной сущности во времени?
- Необходимо ли реагировать на каждое изменение сущности или достаточно за определенный интервал времени генерировать суммарные изменения?
- Что считать изменением сущности и как реагировать на это изменение в другом приложении?

После этого в соответствии с техническими ограничениями необходимо выполнить следующие действия:

- выбрать структуру передаваемого объекта синхронизации;
- выбрать механизм обмена этими объектами;
- выбрать механизм реакции на изменение сущности;
- ввести дополнительные поля для синхронизируемых сущностей.

Ниже представлены рекомендации к осуществлению перечисленных действий и принципы выбора того или иного решения.

Выбор структуры передаваемого объекта синхронизации

Существует две разновидности передаваемых объектов: полная общая модель сущности со всеми ее свойствами и патч-модель, в которой проинициализированы только те свойства, которые были изменены.

Преимущество первой модели заключается в том, что она позволяет написать более простую логику при изменении сущности. Полную модель стоит использовать, когда изменение модели затрагивает сразу несколько полей, а сама модель содержит небольшое количество свойств.

Преимущество второй модели заключается в том, что она позволяет делать атомарные обновления, которые не затрагивают остальных свойств. Например, если в примерно равный промежуток времени на обеих сторонах произошли обновления разных полей одной сущности, то патч-синхронизация позволит сохранить оба изменения и не перепишет поле по принципу «кто последний, тот и прав» (так называемая проблема «потерянного обновления» [2]). Дополнительно, это позволит сэкономить трафик при передаче объектов между приложениями, особенно когда в больших по объему сущностях происходят незначительные изменения.

Выбор механизма обмена объектами

Существует два способа передачи синхронизируемых объектов:

Во-первых, это непосредственно прямые запросы, когда одно из приложений вызывает метод другого приложения для осуществления синхронизации, дожидается завершения обработки и получает результат. Взаимодействие в таком случае может основываться на технологиях RPC, HTTP REST API, WebSockets и т.д.

Во-вторых, обмен сущностями может происходить с помощью программы-посредника, например, брокера программных сообщений [3]. Примерами таких технологий могут быть RabbitMQ, MSMQ, ZeroMQ и др.

Рассмотрим недостатки и преимущества каждого способа. Если задержки в синхронизации должны быть минимальны, то предпочтительно использовать прямые запросы. Однако это может обернуться следующей проблемой: приложение-адресат в какой-то момент времени может быть недоступно по

различным причинам, что может привести к потере данных и дальнейшему нарушению синхронизации.

Использование брокера программных сообщений оправданно только в том случае, когда задержки во времени не критичны. При таком сценарии объекты синхронизации будут поступать в две очереди (по одной с каждой стороны), которыми управляет брокер. Считается, что брокер «надежен», то есть доступен для обоих приложений в любой момент времени. Как правило, он располагается на отдельной машине. Сообщения в очереди могут быть постоянными (англ. persistent, хранятся на диске) и непостоянными (англ. non persistent, хранятся в оперативной памяти и теряются всякий раз при перезагрузке брокера). Обработка непостоянных сообщений в брокере происходит значительно быстрее, но такой вариант следует рассматривать, только если потеря данных синхронизации не критична. Также если сущности независимы между собой, рекомендуется использовать несколько очередей либо обработчиков для распараллеливания процесса синхронизации.

Внутренний механизм реакции на изменение сущности

Для того чтобы запустить процесс синхронизации, необходимо уметь обрабатывать каждое изменение сущности. Существует две разновидности такой обработки: синхронная и асинхронная.

Синхронная реакция подразумевает передачу объекта синхронизации другому приложению сразу же после добавления/обновления/удаления сущности, тем самым задерживая текущий поток выполнения программы до завершения синхронизации. Подходит для программных продуктов, в которых более приоритетна сама синхронизация, чем «отзывчивость» приложения. Однако для большинства современных программ такой подход не является приемлемым, поэтому используют асинхронную реакцию.

Асинхронная реакция позволяет запустить логику синхронизации в другом потоке, который практически не влияет на основной поток изменения сущности. Обычно приложение управляет очередью ссылок на измененные сущности и обрабатывает эти изменения в зависимости от доступных ресурсов

компьютера, отдавая приоритет основным операциям приложения. Проще всего организовать такую обработку с помощью механизма доменных событий [5].

Но если средняя частота возникновения асинхронных реакций в системе превышает среднюю частоту отправки объектов синхронизации, то приложение не справится с возрастающим набором необработанных изменений. Если не требуется реагировать на каждое промежуточное изменение сущности, рекомендуется использовать batch-обновление. Для этого сторона-отправитель накапливает все реакции на изменение сущности за определенный промежуток времени и собирает из них агрегатный объект для передачи. Если в качестве объекта синхронизации используется патч-модель, то все конечные изменения полей за промежуток времени группируются по ключу поля и затем превращаются в одно суммарное изменение для каждого поля. Если используется полная модель, то из всех изменений для отправки выбирается последнее. Такой подход позволяет гибко управлять нагрузкой в системе, однако неизбежно теряются промежуточные обновления, что не всегда приемлемо.

Если же время отправки одного объекта синхронизации существенно превышает время его обработки, рекомендуется использовать другую разновидность batch-синхронизации, когда объекты синхронизации отправляются не поодиночке, а массивом. Иногда это позволяет линейно снизить задержку на прием и отправку единичных сообщений, ускорив процесс обработки данных, особенно, когда отдельно взятые сериализованные сообщения имеют очень малый вес.

Введение дополнительных полей для синхронизации

Также необходимо изменить модель синхронизируемых данных в БД.

Во-первых, чтобы постоянно поддерживать синхронизацию, необходимо с каждой стороны для изменяемой сущности добавить свойство, которое каким-то образом позволит определить, актуальна ли эта сущность или нет. Простейшим примером может выступать поле типа «дата/время»,

отображающее последнее обновление сущности на каждой из сторон синхронизации. Программист должен обеспечить корректное обновление значения данного поля. Это достаточно просто осуществить, например, всякий раз проставлять текущую дату в поле при перезаписи сущности в БД [4]. При сравнении данного поля с такой же меткой времени из пришедшего объекта синхронизации, программа сможет определить, нуждается ли текущая версия сущности в обновлении или же она более актуальна.

Во-вторых, необходимо ввести поля, которые обеспечат связь один-к-одному между сущностями. Как правило, для этого в одной БД достаточно хранить ссылку на идентификатор синхронизируемой сущности из другой БД, и наоборот.

В-третьих, если основная логика изменения сущности и логика изменения сущности при синхронизации одна и та же, необходимо иметь поле-флаг, которое указывает, каким образом была изменена сущность последний раз (внутренне или внешне). Это поможет защититься от рекурсивных обновлений, когда объект синхронизации, пришедший из первого источника, вызывает новую внутреннюю реакцию на изменение сущности во втором источнике, что приведет к новому витку синхронизации в первом источнике и так далее. Как правило, для обновления сущности в приложении всегда используется один и тот же метод, поэтому такая ситуация наиболее вероятна.

Отметим, что вышеописанные поля можно хранить как в самих сущностях, так и в отдельных таблицах/коллекциях БД, в зависимости от технических ограничений.

Заключение

После последовательного выполнении всех действий, указанных в данной статье, программист сможет без труда разработать надежный механизм синхронизации данных между двумя и более приложениями. Ориентируясь на предложенные решения широко распространенных проблем, можно обезопасить программный продукт от существенных трудностей, которые

могут возникнуть во время расширения системы синхронизации или усложнения бизнес-требований.

Библиографический список:

1. Репликация данных [Электронный ресурс]. – [Б. м. : Б. и.], [20–]. – Режим доступа: https://ruhighload.com/Репликация+данных. – Загл. с экрана (дата обращения: 18.06.2020).

2. Concurrency problems in DBMS Transactions [Электронный ресурс]. – [Б. м. : Б. и.], [20–]. – Режим доступа: https://www.geeksforgeeks.org/concurrency-problems-in-dbms-transactions/. – Загл. с экрана (дата обращения: 07.05.2020).

3. Message Brokers [Электронный ресурс]. – [Б. м. : Б. и.], [2020]. – Режим доступа: https://www.ibm.com/cloud/learn/message-brokers. – Загл. с экрана (дата обращения: 11.07.2020).

3. Store the date and time when a record is modified [Электронный ресурс]. – [Б. м. : Б. и.], [20–]. – Режим доступа: https://support.microsoft.com/en-gb/office/store-the-date-and-time-when-a-record-is-modified-0c46efc5-5b43-4751-b3a8-c246505af66d. – Загл. с экрана (дата обращения: 17.07.2020).

4. Using Domain Events within a .NET Core Microservice [Электронный ресурс]. – [Б. м. : Б. и.], [20–]. – Режим доступа: https://devblogs.microsoft.com/cesardelatorre/using-domain-events-within-a-net-core-microservice/. – Загл. с экрана (дата обращения: 30.06.2020).