Руководство по Service Broker

Сергей Минюров

Москва, 2019, версия 1.1

Содержание

Благодарность

Замечательно иметь друзей и коллег, которые несмотря на постоянную рабочую и житейскую суету развиваются, сохраняют живой интерес к профессии и делятся своими мыслями, идеями и проблемами.

Талиб Гасанов очень интересный собеседник, как руководитель ИТ обладает широким кругозором. Несмотря на организационную нагрузку всегда старается использовать новые интересные технологии. Сохраняет вкус к программированию и заряжает энергией.

Мне всегда интересно с Талибом обсуждать разные темы и задачи. Периодически он дает мне профессиональные творческие проекты. Данное руководство является продуктом именно такого проекта.

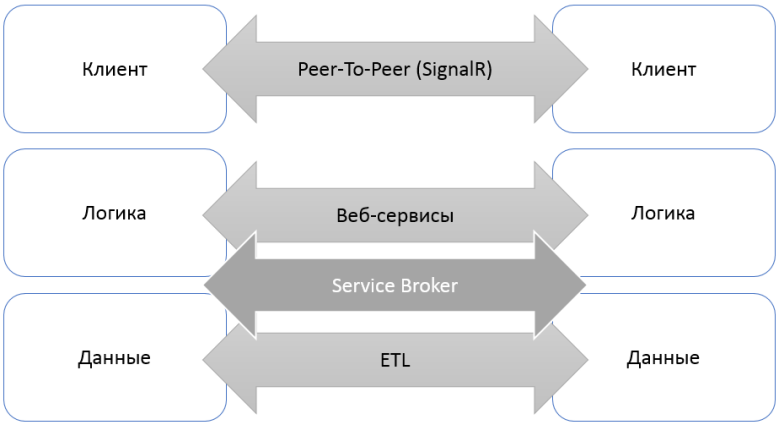
Назначение

Service Broker является программной технологией для разработки интеграционных решений. Для понимания его назначения важно рассмотреть в целом задачи и технологии. Интеграция может выполняться на разных уровнях архитектуры информационной системы и разными способами:

Задача интеграции	Технология	Аспекты		
		Данные	Предметная логика	Взаимодействие
Распределенная база данных	Репликация	Большой объем, сложная структура	Простая	Синхронное / асинхронное
Передача данных	ETL	Большой объем, сложная структура	Простая	Асинхронное
Рабочий процесс	SOA (веб-сервисы)	Средний объем, сложная структура	Сложная, распределенная	Синхронное / асинхронное
	Service Broker	Большой объем, сложная структура		
Коммуникация	Peer-To-Peer (SignalR)	Малый объем, простая структура	Простая, распределенная	Синхронное

При выборе решения важно различать задачи 1) обработки данных и 2) исполнения процессов. Service Broker является гибридной технологией, обеспечивающей обработку большого объема данных со сложной структурой, и исполнение сложной распределенной логики (Workflow). Таким образом, Service Broker объединяет функциональные возможности ETL и веб-сервисов.

Принципиальным ограничением является работа только на платформе SQL Server.



Ближайшим аналогом является SOA (веб-сервисы), которая в отличие от Service Broker является кросс-платформенным решением и может быть более простым решением при построении интеграционного решения с внешними контрагентами.

Критерии выбора разработки решения на Service Broker:

Использовать	**Не использовать**
Балансировка нагрузки для одной системы	Все локальные задачи требуется выполнять синхронно.
Распределенный рабочий процесс	Требуется только обмен данными
Реализация внутри корпоративной системы на платформе Microsoft SQL Server	Требуется реализовать решение независимо от баз данных и платформ, в т.ч. для внешних контрагентов

Универсальным критерием является относительная простота в разработке и поддержке решения, его органичное включение в существующую информационную систему.

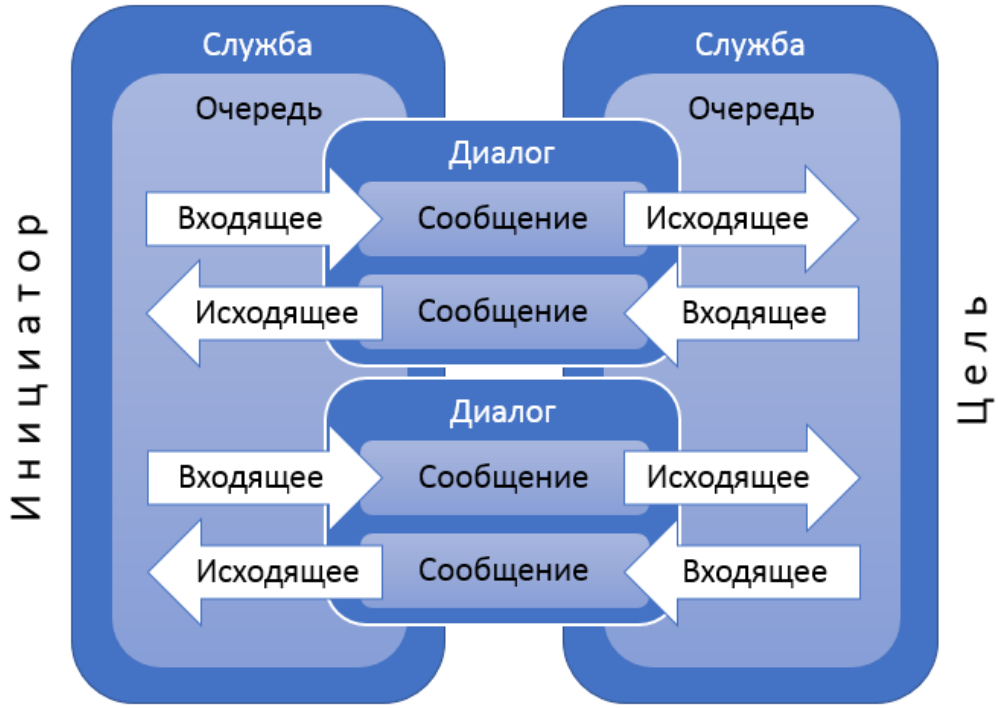
Функциональные возможности

Компонент Service Broker реализует асинхронное и распределенное программирование на стороне сервера базы данных. Это позволяет реализовать сквозные процессы, исполняемые на разных серверах баз данных. А также обеспечивает балансировку нагрузки, позволяя разделять синхронное выполнение критичных задач, и ставить в очередь задачи, которые не требуется выполнять немедленно

Асинхронное программирование основано на **сообщениях**, которые являются частью **диалога** (распределенной логической задачи (distributed unit of work)). Хранилищем сообщений являются **очереди**. С точки зрения определенной системы сообщения могут быть **исходящими** – отправляться в другую систему, и **входящими** – получаемыми из другой системы. Система, которая создает диалог, является **инициатором**. Другая сторона является **целью**.

> Для балансировки нагрузки база данных может оправлять сообщения сама себе.
>
> Для диалога может задаваться **приоритет** от 1 (низкий) до 10 (высокий). Это обеспечивает оперативную обработку важных сообщений. По умолчанию задается приоритет 5.

Каждый диалог принадлежит определенной **группе сообщений**, связанной с конкретной службой.



На каждой стороне определяются службы, являющиеся логическим адресом для отправки сообщений. Каждая служба связана с очередью. Служба на стороне инициатора называется **вызывающей службой**, а на стороне цели называется **целевой службой**.

Базовый сценарий диалога:

- На стороне инициатора:
  1. Программа начинает диалог.
  2. Программа создает сообщение, содержащее данные, необходимые для выполнения задачи.
  3. Компонент отправляет сообщение целевой службе.

- На стороне цели:
    1. Сообщение помещается в очередь, связанную с целевой службой.
    2. Программа получает сообщение из очереди и обрабатывает данные.

Цель также до завершения общения может отправлять сообщение инициатору. Таким образом, может формироваться циклический диалог между разными службами в виде сообщений разных типов. Каждая сторона после обработки последнего сообщения закрывает общение.

Распределенная задача может выполняться на нескольких серверах в разных диалогах, соответственно, для сообщений, относящихся к общей задаче, устанавливается общий идентификатор **группы сообщений.**

Можно настроить маршрутизацию сообщений через несколько серверов.

При разработке распределенных решений Service Broker обеспечивает надежный, безопасный и упорядоченный обмен сообщениями между разными базами данных и серверами на основе сетевого протокола TCP/IP.

> Обмен сообщениями основан на транзакциях и механизме очередей. Отправка сообщения выполняется только после фиксации транзакции на стороне инициатора. Удаление сообщения из очереди также выполняется только после фиксации транзакции на стороне цели. Таким образом, обеспечивается надежность и согласованность обработки данных и сообщений.
>
> Сообщения доставляются в том порядке, как они были отправлены, и только один раз.

Встроенным источником данных являются уведомления о событиях, которые можно настраивать вместо триггеров для асинхронной обработки. Также их можно использовать для встроенного мониторинга производительности – как аналог трассировки SQL.

Service Broker используется почтовой службой SQL Server и для зеркалирования баз данных.

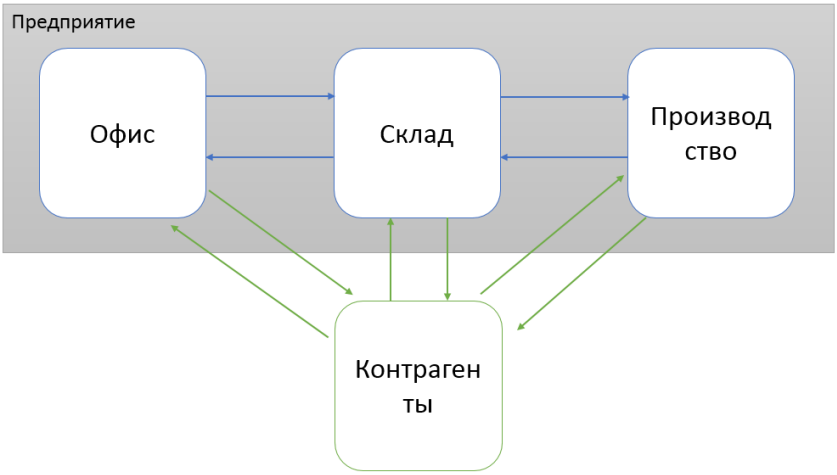
Проектирование

Логически решение на Service Broker является клеточным автоматом. С точки зрения предметной логики исполняется рабочий процесс (workflow), локальный или распределенный.

> В некоторой степени программирование похоже на разработку триггеров, которые являются хранимыми процедурами, обрабатывающими события изменения данных или события уровня сервера.
>
> Service Broker выполняет отправку и получение сообщений. При этом сообщения могут отправляться из любой хранимой процедуры, а обработка сообщений может быть реализовано по расписанию или через активацию – в этом случае возникает полный аналог с триггерами, когда поступление события запускает выполнение его обработчика.

Пример распределенного решения. На предприятии может быть несколько подразделений и, соответственно, приложений: торговое, складское и производственное. Для обработки заказов может потребоваться реализовать распределенный рабочий процесс на основе интеграции этих приложений. Обработка заказов включает в себя планирование и исполнение. Пример процесса планирования:

1. Клиент может выбрать товарные позиции, запросить сроки и цены.
2. На складе проверяется наличие требуемых товаров, при необходимости формируется запрос на производство или подрядчикам.
3. Производство или подрядчик отправляет оценку по срокам и стоимости, которая затем собирается и передается клиенту.



При проектировании решения на основе Service Broker нужно решить 2 задачи:

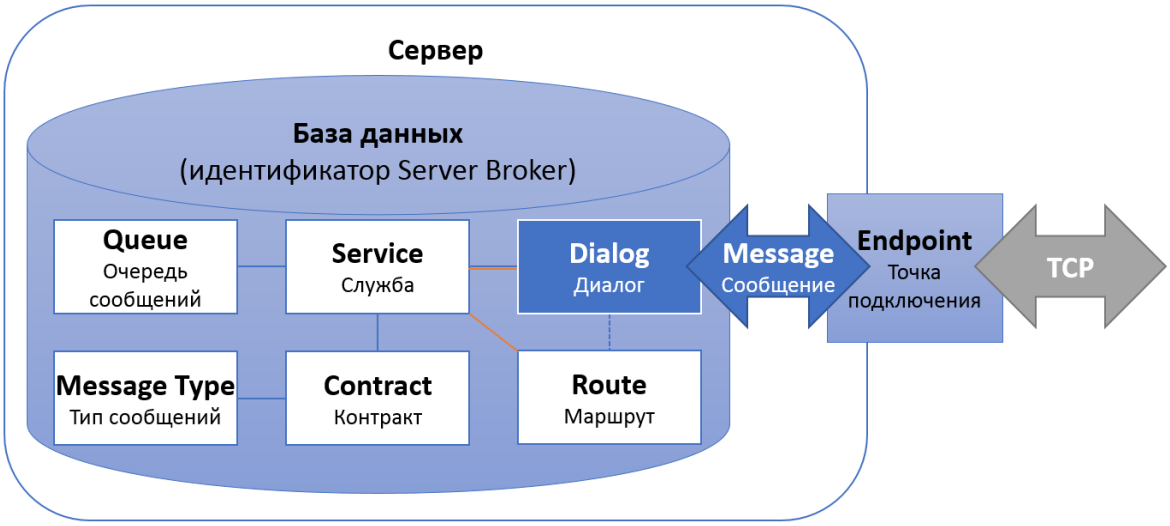
1. Определить топологию решения: какие локальные сети, домены, серверы и базы данных участвуют в рабочем процессе.
2. Определить логику решения: какие типы сообщений, форматы данных и обработчики требуется разработать.

С помощью контрактов описывается взаимодействие между исполнителями (и их информационными системами): например, в офисе оформляется заказ на продажу, а затем выполняется резервирование и отгрузка товара. Контракты определяют типы сообщений, описывающих варианты взаимодействия.

Контракт	Офис	Склад	Производство
Заказ на продажу	Запрос на товар Резервирование товара		
Отгрузка товара	Запрос на отгрузку Подтверждение отгрузки	Запрос на отгрузку Подтверждение отгрузки	
Заказ на производство	План производства Подтверждение плана	Заявка на товар План производства Подтверждение плана Производство товара	

Каждая база данных на SQL Server может является приложением, участвующим в распределенном процессе с помощью Service Broker. Соответственно, база данных имеет специальный глобальный уникальный идентификатор и набор объектов для формирования, отправки и обработки сообщений.

Для доступа к удаленным сервисам требуется на уровне экземпляра сервера базы данных определить точку подключения (Endpoint) для компонента Service Broker.

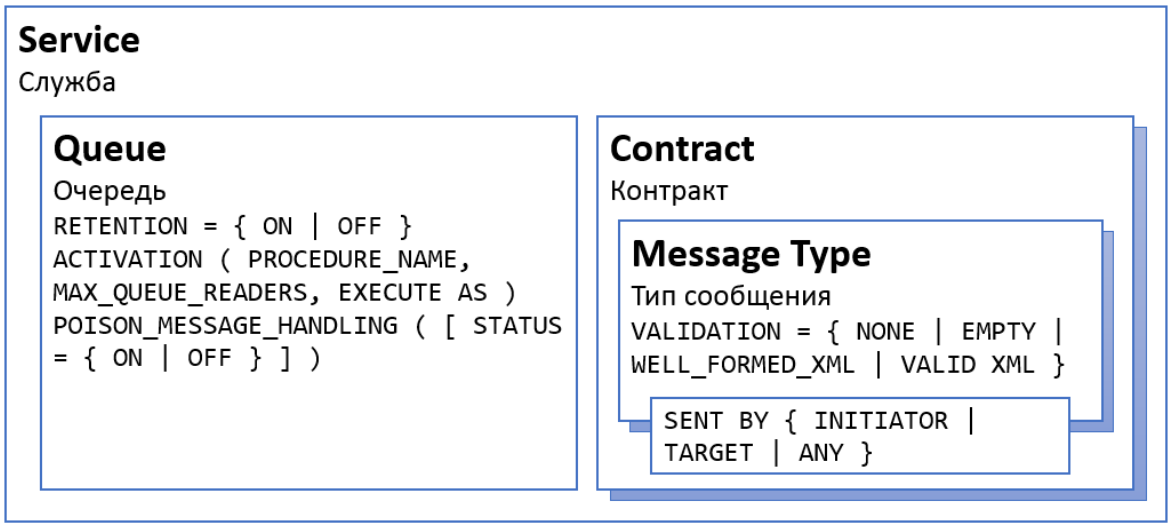


Поскольку передача данных производится через сетевой протокол TCP/IP, требуется настроить файрвол на разрешение передачи данных через указанные для точки подключения Service Broker и включить для экземпляра сервера базы данных данный протокол.

На уровне базы данных дополнительно создаются удаленные привязки к внешним сервисам (remote service binding). Здесь определяется пользователь (служебные учетные данные), связанный с сертификатом, из которого используется открытый ключ для проверки подлинности сообщений и для шифрования сеансового ключа, который затем используется для шифрования сеанса связи.

> Если используется анонимная проверка подлинности (**ANONYMOUS = ON**), то служба вызывающей стороны подключается к конечному сервису как член предопределенной роли базы данных public. По умолчанию члены этой роли не имеют разрешения на подключение к базе данных. Для успешной отправки сообщения целевая база данных должна предоставить роли **public** разрешение **CONNECT** для базы данных и разрешение **SEND** для конечной службы.

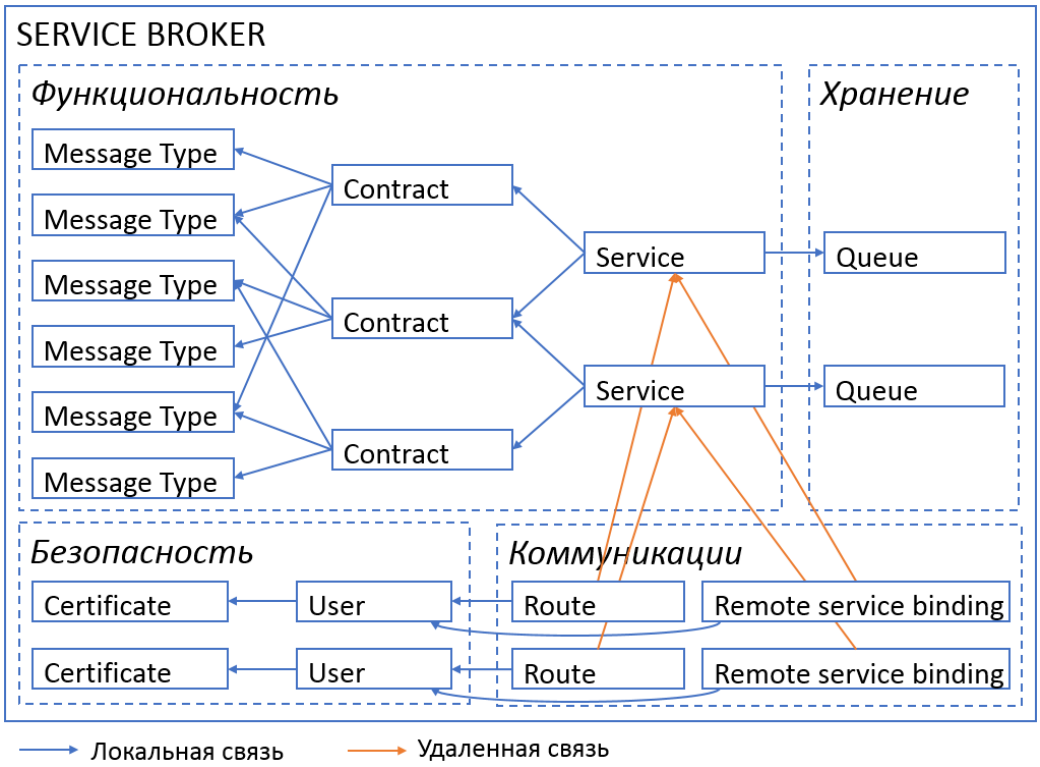
Логическая композиция объектов Service Broker:



Классы объектов Service Broker:

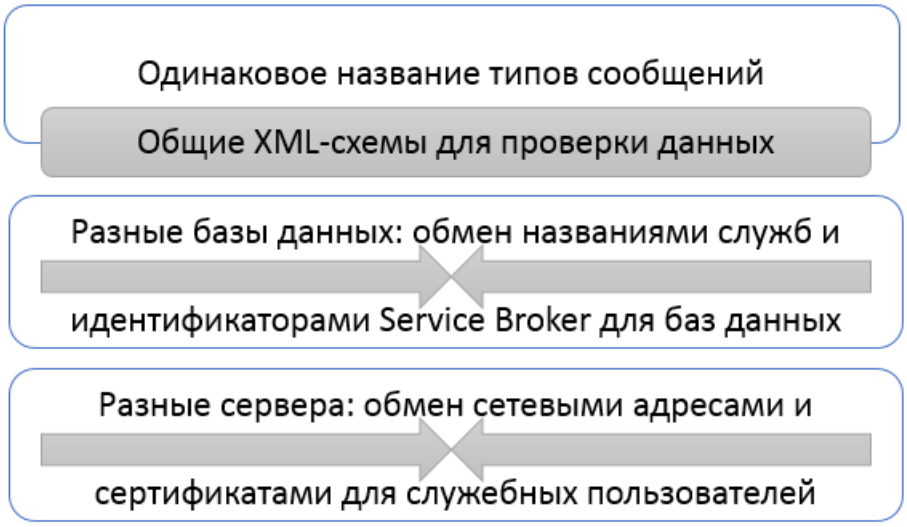
- Тип сообщения (Message Type) – определяет формат сообщения. Может быть пустой, произвольный (обычно для передачи бинарных данных), XML или типизированный XML (на основе XML-схемы).
- Очередь сообщений (Queue) – хранилище актуальных сообщений, отправляемых или получаемых. После обработки сообщения удаляются.
    - Для сохранения сообщений в очереди до завершения диалога нужно устновить параметр **RETENTION = ON**. Это может использоваться для аудита или выполнения компенсирующих транзакций.
    - Для очереди может определяться хранимая процедура как обработчик входящих сообщений (**ACTIVATION (PROCEDURE_NAME)**). Для обработчика можно указывать количество параллельных потоков (**MAX_QUEUE_READERS**) и контекст безопасности (**EXECUTE AS**).
    - Стандартная обработка сбоев (**POISON_MESSAGE_HANDLING**) включена по умолчанию. При этом после 5 сбоев очередь отключается. Если в приложении реализована пользовательская обработка ошибочных сообщений, то этот параметр можно отключить.
- Контракт (Contract) – объединяет типы сообщения в набор, соответствующий определенной задаче. Задает для каждого типа сообщения направление: входящие (**TARGET**), исходящие (**INITIATOR**) или двухсторонние (**ANY**).
- Служба (Service) – логическая точка доступа, объединяет контракты и очереди. Служба связана только с одной очередью, позволяет задать несколько контрактов, сообщения которых будут храниться в соответствующей очереди.
    - Служба может не содержать контрактов, в этом случае она может только инициировать диалог.
- Маршрут (Route) – связывает службы для передачи сообщений определенных типов. При поиске службы по названию регистр различается (производится бинарное сравнение).

Имеется тип сообщений и контракт по умолчанию (**DEFAULT**), которые можно использовать для тестирования или простых сообщений. Также имеется по умолчанию имеется маршрут **AutoCreatedLocal**, который используется для передачи сообщений внутри базы данных.



При настройке важно синхронизировать типы сообщений и контракты между удаленными службами, а для самих служб и маршрутов настроить зеркальное соответствие.

Важно согласовать между участниками интеграции название типов сообщений и разработать общие для всех XML-схемы, используемые для проверки данных в сообщениях.



Для разных баз данных нужно передать друг другу название служб, которые принимают сообщения и идентификаторы Service Broker для баз данных, участвующих в обмене сообщениями.

Для разных серверов требуется также обменяться сетевыми адресами и сертификатами для служебных пользователей, которые используются при создании точек подключения и внешних привязок к службам.

Обмен сообщениями на одном экземпляре сервера базы данных

Самый простой вариант настройки Service Broker выполняется на одном сервере базы данных, поскольку не требуется настройка топологии.

Общая база данных

Для балансировки нагрузки можно настроить обработку сообщений в рамках одной базы данных: т.е. для приложения сообщения будут посылаться сами себе. Таким образом, можно одни операции выполнять синхронно, в рамках транзакции пользователя, а другие операции выполнять асинхронно, в отдельной транзакции обработки очереди.

В этом случае будет использоваться системный маршрут **`AutoCreatedLocal`**. Достаточно определить типы сообщений, контракты, очереди и сервисы. А создавать собственный маршрут не требуется.

1. Создание и настройка базы данных:

```
CREATE DATABASE SBDemoLocal
ALTER DATABASE SBDemoLocal SET ENABLE_BROKER;

USE SBDemoLocal
CREATE MESSAGE TYPE DemoMessageType VALIDATION = NONE;
CREATE CONTRACT DemoContract (DemoMessageType SENT BY ANY);
CREATE QUEUE DemoQueue;

CREATE SERVICE DemoService ON QUEUE DemoQueue (DemoContract);
```

2. Запуск диалога, отправка сообщения и завершение общения:

```
USE SBDemoLocal

DECLARE @h UNIQUEIDENTIFIER;
BEGIN DIALOG CONVERSATION @h FROM SERVICE [DemoService] TO SERVICE 'DemoService'
ON CONTRACT [DemoContract] WITH ENCRYPTION = OFF;

SEND ON CONVERSATION @h MESSAGE TYPE [DemoMessageType] ('test message');

END CONVERSATION @h;
```

3. Получение сообщения и завершение общения:

```
DECLARE @h UNIQUEIDENTIFIER, @msg VARCHAR(MAX);
BEGIN TRANSACTION;
  WAITFOR (
    RECEIVE TOP (1) @h = [conversation_handle], @msg = TRY_CONVERT(varchar, [message_body])
    FROM [DemoQueue]
  ), TIMEOUT 1000
  IF @@ROWCOUNT <= 0 BEGIN
    PRINT 'No messages'
    ROLLBACK TRANSACTION;
    RETURN;
  END;
  PRINT @msg
  END CONVERSATION @h;
COMMIT TRANSACTION;
```

Соответственно, мы получаем возможность разделять выполнение задачи между разными транзакциями (асинхронно), и минимизировать обработку данных в пользовательской транзакции.

Общий экземпляр, разные базы данных

Елси разные базы данных находятся на одном сервере, то дополнительно нужно установить для этих баз данных опцию **TRUSTWORTHY** и настроить маршрут. При настройке маршрута нужно указать идентификатор Service Broker целевой базы данных и встроенный системный адрес **Local**.

База данных 1	**База данных 2**
1. Создание и настройка баз данных: нужно обменяться идентификаторами Service Broker баз данных	

`CREATE DATABASE SBDemo1`
`ALTER DATABASE SBDemo1 SET ENABLE_BROKER;`
`ALTER DATABASE SBDemo1 SET TRUSTWORTHY ON;`
 	
`USE SBDemo1`
`SELECT [service_broker_guid]`
`FROM sys.databases`
`WHERE database_id = DB_ID();`
`-- CA517779-C8B2-491A-BE73-00BA702DA888`
`CREATE MESSAGE TYPE DemoMessageType VALIDATION = NONE;`
`CREATE CONTRACT DemoContract (`
`  DemoMessageType SENT BY ANY);`
`CREATE QUEUE Queue1;`
 	
`CREATE SERVICE Service1`
`  ON QUEUE Queue1 (DemoContract);`
	`CREATE DATABASE SBDemo2`
	`ALTER DATABASE SBDemo2 SET ENABLE_BROKER;`
	`ALTER DATABASE SBDemo2 SET TRUSTWORTHY ON;`
	
	`USE SBDemo2`
	`SELECT [service_broker_guid]`
	`FROM sys.databases`
	`WHERE database_id = DB_ID();`
	`-- B58A5964-3B95-45E3-AA21-55525EB6DD5E`
	`CREATE MESSAGE TYPE DemoMessageType VALIDATION = NONE;`
	`CREATE CONTRACT DemoContract (`
	`  DemoMessageType SENT BY ANY);`
	`CREATE QUEUE Queue2;`
	
	`CREATE SERVICE Service2`
	`  ON QUEUE Queue2 (DemoContract);`

1.1. Настройка маршрутов: используется идентификатор Service Broker цели	

`CREATE ROUTE Route1 WITH`
`  SERVICE_NAME = 'Service2',`
`  ADDRESS = 'Local',`
`  BROKER_INSTANCE =`
`  'B58A5964-3B95-45E3-AA21-55525EB6DD5E'`
	`CREATE ROUTE Route2 WITH`
	`  SERVICE_NAME = 'Service1',`
	`  ADDRESS = 'Local',`
	`  BROKER_INSTANCE =`
	`  'CA517779-C8B2-491A-BE73-00BA702DA888'`

2. Запуск диалога, отправка сообщения и завершение общения:	

`USE SBDemo1`
`DECLARE @h UNIQUEIDENTIFIER;`
`BEGIN DIALOG CONVERSATION @h`
`FROM SERVICE Service1 TO SERVICE 'Service2'`
`ON CONTRACT DemoContract`
`WITH ENCRYPTION = OFF;`
 	
`SEND ON CONVERSATION @h MESSAGE TYPE`
`DemoMessageType ('test message from db 1');`
`END CONVERSATION @h;`	

3. Получение сообщения и завершение общения:	

	`USE SBDemo2`
	`DECLARE @h UNIQUEIDENTIFIER, @msg VARCHAR(MAX);`
	`BEGIN TRANSACTION;`
	`  WAITFOR (`
	`  RECEIVE TOP (1)`
	`  @h = [conversation_handle]`
	`  , @msg = TRY_CONVERT(varchar, [message_body])`
	`  FROM [Queue2]`
	`  ), TIMEOUT 1000`
	`  IF @@ROWCOUNT <= 0 BEGIN`
	`  PRINT 'No messages'`
	`  ROLLBACK TRANSACTION;`
	`  RETURN;`
	`  END;`
	`  PRINT @msg`
	`  END CONVERSATION @h;`
	`COMMIT TRANSACTION;`

Обмен сообщениями на разных серверах базы данных

Перед настройкой требуется проверить файрвол на разрешение передачи данных через указанные для точки подключения Service Broker порты и включить для экземпляров серверов баз данных протокол TCP/IP.

Если используется Kerberos, то настройка Service Broker упрощается. В данном примере мы рассматриваем более универсальный и сложный вариант настройки обмена сообщениями с помощью сертификатов.

При обмене сообщениями между базами данных, которые находятся на разных серверах базы данных необходимо настроить точки подключения (**Endpoint**).

При необходимости можно настроить шифрование сообщений на транспортном или сеансовом уровне. Для шифрования на транспортном уровне нужно при создании точки подключения привязать к ней сертификат. Для шифрования на сеансовом уровне нужно на уровне баз данных создать служебных пользователей и привязать к ним сертификаты (см. пример далее).

Сервер 1	**Сервер 2**
1. Создание мастер ключа и сертификата для точки подключения:	

`CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Pa$$w0rd';`
 	
`CREATE CERTIFICATE Endpoint1Cert WITH`
`  SUBJECT = 'For Service Broker endpoint';`
 	
`CREATE ENDPOINT Endpoint1`
`STATE = STARTED`
`AS TCP (LISTENER_PORT = 4022)`
`FOR SERVICE_BROKER (`
`  AUTHENTICATION = CERTIFICATE Endpoint1Cert);`
	`CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Pa$$w0rd';`
	
	`CREATE CERTIFICATE Endpoint2Cert WITH`
	`  SUBJECT = 'For Service Broker endpoint';`
	
	`CREATE ENDPOINT Endpoint2`
	`STATE = STARTED`
	`AS TCP (LISTENER_PORT = 4022)`
	`FOR SERVICE_BROKER (`
	`  AUTHENTICATION = CERTIFICATE Endpoint2Cert);`

2A. Настройка для анонимного диалога:	
`GRANT CONNECT ON ENDPOINT::Endpoint1 TO public;`	`GRANT CONNECT ON ENDPOINT::Endpoint2 TO public`
2B. Настройка для шифрования диалога: нужно обменяться файлами сертификатов:	

`BACKUP CERTIFICATE Endpoint1Cert`
`TO FILE = 'X\Endpoint1Cert.cert';`
 	
`CREATE LOGIN SBLogin2 WITH PASSWORD = 'Pa$$w0rd';`
`CREATE USER SBUser2 FOR LOGIN SBLogin2;`
`CREATE CERTIFICATE Endpoint2Cert`
`AUTHORIZATION SBUser2`
`FROM FILE = 'X\Endpoint2Cert.cert';`
 	
`GRANT CONNECT ON ENDPOINT::Endpoint1 TO SBLogin2;`
	`BACKUP CERTIFICATE Endpoint2Cert`
	`TO FILE = 'X\Endpoint2Cert.cert';`
	
	`CREATE LOGIN SBLogin1 WITH PASSWORD = 'Pa$$w0rd';`
	`CREATE USER SBUser1 FOR LOGIN SBLogin1;`
	`CREATE CERTIFICATE Endpoint1Cert`
	`AUTHORIZATION SBUser1`
	`FROM FILE = 'X\Endpoint1Cert.cert';`
	
	`GRANT CONNECT ON ENDPOINT::Endpoint2 TO SBLogin1`

3. Создание и настройка баз данных: нужно обменяться идентификаторами Service Broker баз данных	

`CREATE DATABASE SBDemo1;`
`ALTER DATABASE SBDemo1 SET ENABLE_BROKER;`
`ALTER DATABASE SBDemo1 SET TRUSTWORTHY ON;`
 	
`USE SBDemo1`
`SELECT service_broker_guid FROM sys.databases`
`WHERE database_id = DB_ID()`
`-- D6D29F2F-AAA0-493C-92B0-701E29B10EB7`
`CREATE MESSAGE TYPE DemoMessageType VALIDATION = NONE;`
 	
`CREATE CONTRACT DemoContract (`
`  DemoMessageType SENT BY ANY);`
`CREATE QUEUE Queue1;`
`CREATE SERVICE Service1`
`  ON QUEUE Queue1 (DemoContract);`
	`CREATE DATABASE SBDemo2`
	`ALTER DATABASE SBDemo2 SET ENABLE_BROKER;`
	`ALTER DATABASE SBDemo2 SET TRUSTWORTHY ON;`
	
	`USE SBDemo2`
	`SELECT service_broker_guid from sys.databases`
	`WHERE database_id = DB_ID()`
	`-- C6938686-BAA3-41A3-8186-72D57278FAC9`
	`CREATE MESSAGE TYPE DemoMessageType VALIDATION = NONE;`
	
	`CREATE CONTRACT DemoContract (`
	`  DemoMessageType SENT BY ANY);`
	`CREATE QUEUE Queue2;`
	`CREATE SERVICE Service2`
	`  ON QUEUE Queue2 (DemoContract);`

Сервер 1	Сервер 2
3.1. Настройка маршрутов: используется идентификатор Service Broker цели	
`CREATE ROUTE Service2Route WITH SERVICE_NAME = 'Service2', ADDRESS = 'TCP://0.0.0.0:4022', BROKER_INSTANCE = 'C6938686-BAA3-41A3-8186-72D57278FAC9';`	`CREATE ROUTE Service1Route WITH SERVICE_NAME = 'Service1', ADDRESS = 'TCP://0.0.0.0:4022', BROKER_INSTANCE = 'D6D29F2F-AAA0-493C-92B0-701E29B10EB7';`
4A. Настройка анонимного доступа к базе данных и локальной службе	
`GRANT CONNECT TO public` `GRANT SEND ON SERVICE::Service1 TO public`	`GRANT CONNECT TO public` `GRANT SEND ON SERVICE::Service2 TO public`
4B. Настройка доступа к локальной службе: нужно обменяться файлами сертификатов	

`CREATE USER DemoUser1 WITHOUT LOGIN;`
`GRANT CONTROL ON SERVICE::Service1 TO DemoUser1;`
 	
`CREATE MASTER KEY ENCRYPTION`
`BY PASSWORD = 'Pa$$w0rd';`
 	
`CREATE CERTIFICATE DemoUser1Cert`
`AUTHORIZATION DemoUser1`
`WITH SUBJECT = 'For SB Service ';`
 	
`BACKUP CERTIFICATE DemoUser1Cert`
`TO FILE = 'X\DemoUser1Cert.cert';`
	`CREATE USER DemoUser2 WITHOUT LOGIN`
	`GRANT CONTROL ON SERVICE::Service2 TO DemoUser2`
	
	`CREATE MASTER KEY ENCRYPTION`
	`BY PASSWORD = 'Pa$$w0rd'`
	
	`CREATE CERTIFICATE DemoUser2Cert`
	`AUTHORIZATION DemoUser2`
	`WITH SUBJECT = 'For SB Service'`
	
	`BACKUP CERTIFICATE DemoUser2Cert`
	`TO FILE = 'X\DemoUser2Cert.cert'`

5. Настройка доступа к удаленному сервису	

`CREATE USER DemoUser2 WITHOUT LOGIN`
 	
`CREATE CERTIFICATE DemoUser2Cert`
`AUTHORIZATION DemoUser2`
`FROM FILE = 'X\DemoUser2Cert.cert'`
 	
`CREATE REMOTE SERVICE BINDING Service2Binding`
`TO SERVICE 'Service2'`
`WITH USER = DemoUser2, ANONYMOUS = ON / OFF;`
	`CREATE USER DemoUser1 WITHOUT LOGIN`
	
	`CREATE CERTIFICATE DemoUser1Cert`
	`AUTHORIZATION DemoUser1`
	`FROM FILE = 'X\DemoUser1Cert.cert'`
	
	`CREATE REMOTE SERVICE BINDING Service1Binding`
	`TO SERVICE 'Service1'`
	`WITH USER = DemoUser1, ANONYMOUS = ON / OFF;`

6. Запуск диалога, отправка сообщения и завершение общения:	

`USE SBDemo1`
`DECLARE @h UNIQUEIDENTIFIER;`
`BEGIN DIALOG CONVERSATION @h`
`FROM SERVICE Service1 TO SERVICE 'Service2'`
`ON CONTRACT DemoContract`
`WITH ENCRYPTION = OFF;`
 	
`SEND ON CONVERSATION @h MESSAGE TYPE`
`DemoMessageType ('test message from server 1');`
`END CONVERSATION @h;`	

7. Получение сообщения и завершение общения:	

	`USE SBDemo2`
	`DECLARE @h UNIQUEIDENTIFIER, @msg VARCHAR(MAX);`
	`BEGIN TRANSACTION;`
	`  WAITFOR (`
	`  RECEIVE TOP (1) @h = [conversation_handle]`
	`  , @msg = TRY_CONVERT(varchar, [message_body])`
	`  FROM [Queue2]`
	`  ), TIMEOUT 1000`
	`  IF @@ROWCOUNT <= 0 BEGIN`
	`  PRINT 'No messages'`
	`  ROLLBACK TRANSACTION;`
	`  RETURN;`
	`  END;`
	`  PRINT @msg`
	`  END CONVERSATION @h;`
	`COMMIT TRANSACTION;`

Диагностика

Способы диагностики для Service Broker:

1. Просмотр системных представлений или функций.
2. Просмотр событий в SQL Server Profiler или Extended Events.
3. Тестирование с помощью утилиты ssbdiagnose.

При анализе причин сбоев в первую очередь нужно проверить корректность настроек безопасности (см. Безопасность и защита (компонент Service Broker)) и соответствие имен типов сообщений и служб.

Основные понятия устранения неполадок (компонент Service Broker).

Системные представления и функции

Для просмотра объектов Service Broker используются представления:

- Точки подключения для Service Broker: sys.service_broker_endpoints.
- Типы сообщений: sys.service_message_types.
    - Коллекции XML-схем для проверки формата сообщений: sys.message_type_xml_schema_collection_usages.
- Контракты: sys.service_contracts.
    - Использование типов сообщений для контрактов: sys.service_contract_message_usages.
- Очереди: sys.service_queues.
- Службы: sys.services.
    - Использование очередей для служб: sys.service_queue_usages.
    - Использование контрактов для служб: sys.service_contract_usages.
- Маршруты: sys.routes.
- Удаленные привязки служб: sys.remote_service_bindings.
- Приоритеты служб: sys.conversation_priorities.

Для анализа работы и текущего состояния Service Broker используются представления:

- Сообщения в очереди передач: sys.service_broker_endpoints.
  Может содержать общую информацию о проблемах с передачей данных в поле **`transmission_status`**.
  Также имеется поле **`is_conversation_error`** для фильтрации сообщений об ошибке.
- Активные диалоги: sys.conversation_endpoints.
  Позволяет анализировать текущую активность на инициаторе или цели. Поля state и state_desc описывают состояния диалога, в т.ч. наличие ошибок (значения 'ER' и 'ERROR' соответственно).
- Активные группы диалогов: sys.conversation_endpoints.

Дополнительно можно использовать динамические административные представления:

- Хранимые процедуры как обработчики входящих сообщений: sys.dm_broker_activated_tasks.
- Список мониторов очереди, выполняющих активацию хранимых процедур для обработки входящих сообщений: sys.dm_broker_queue_monitors.
- Используемые Service Broker сетевые подключения: sys.dm_broker_connections.
- Сообщения, пересылаемые в данный момент: sys.dm_broker_forwarded_messages.

Эти представления позволяют получить общую информацию о настройках и состоянии, которую можно использовать для аудита. Но они не содержат детальную информацию об ошибках, необходимых для их исправления.

На основе представлений можно разрабатывать запросы для аудита или скрипты как инструменты для мониторинга и управления диалогами. Пример отчета по сервисам и контрактам:

```
SELECT sv.name as [Service], sc.name as [Contract]
FROM sys.services sv
INNER JOIN sys.service_contract_usages scu ON scu.service_id = sv.service_id
INNER JOIN sys.service_contracts sc ON sc.service_contract_id = scu.service_contract_id
```

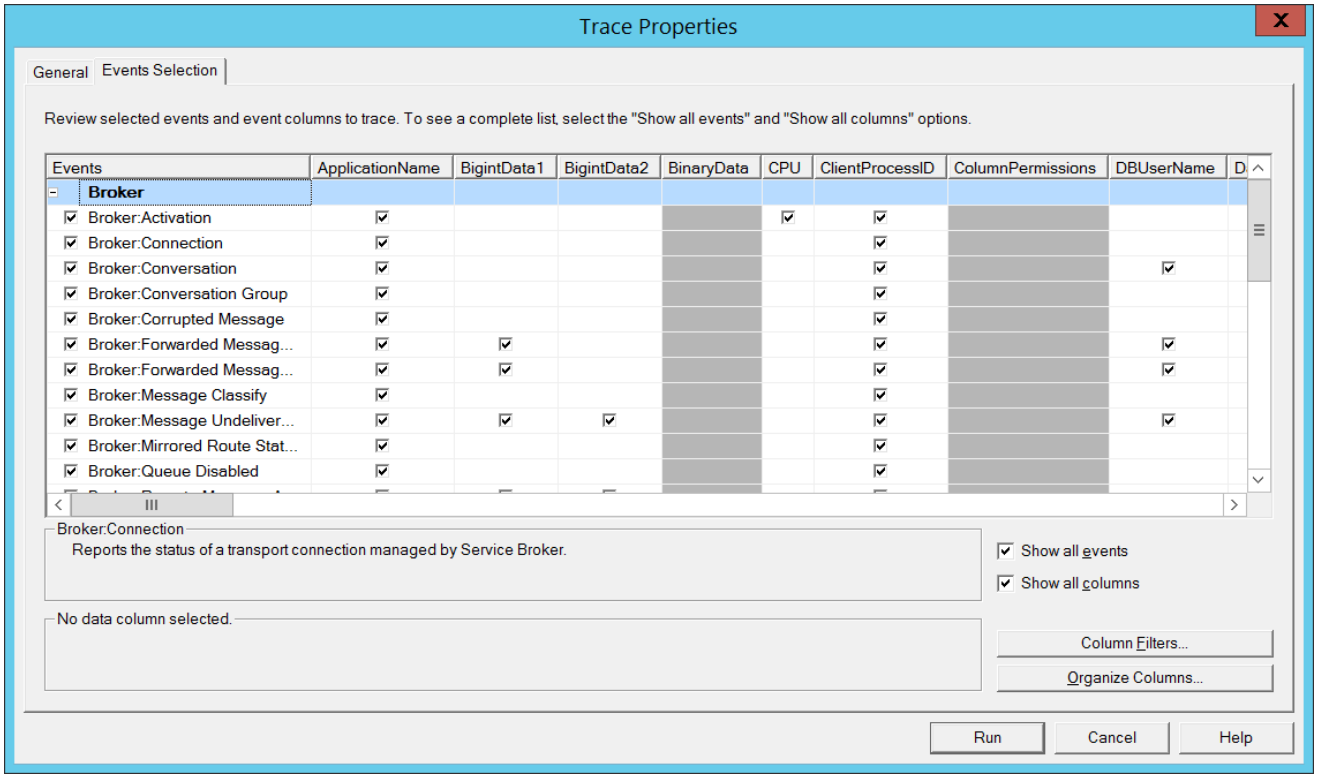
Пример хранимой процедуры для завершения всех диалогов (взят из форума на sql.ru):

```
CREATE PROCEDURE dbo.usp_EndAllConversations AS
BEGIN
    DECLARE @sql NVARCHAR(MAX) = N''
    SELECT @sql = @sql + REPLACE('END CONVERSATION "' + CONVERT(NVARCHAR(36), [conversation_handle]) +
'" WITH CLEANUP', NCHAR(34), NCHAR(39)) + NCHAR(59) + NCHAR(13) + NCHAR(10)
    FROM sys.conversation_endpoints WHERE [state] <> 'CD';
    PRINT @sql;
    EXEC (@sql);
END
```

С помощью системной скалярной функции GET_TRANSMISSION_STATUS можно получить информацию о состоянии последней передачи для каждой стороны диалога, в т.ч. описание ошибки, полезное для диагностики.

Просмотр событий

Для поиска причин сбоев просмотр событий с помощью SQL Server Profiler или Extended Events является самым полезным инструментом. Причем их нужно запускать на каждой стороне. Обычно на одной из сторон можно получить конкретное сообщение об ошибке, помогающее понять и локализовать причину сбоя.



Для просмотра события нужно на каждой стороне запустить, например, SQL Server Profiler, подключиться к серверу, выбрать пустой шаблон, а затем выбрать события Service Broker.

Утилита для диагностики

Утилита командной строки **ssbdiagnose** также является полезным инструментом для тестирования настроек и выдает конкретные описания имеющихся проблем. По умолчанию выводит информацию на экран в консоли. С помощью опции -XML можно настроить сохранение результата проверки в файл.

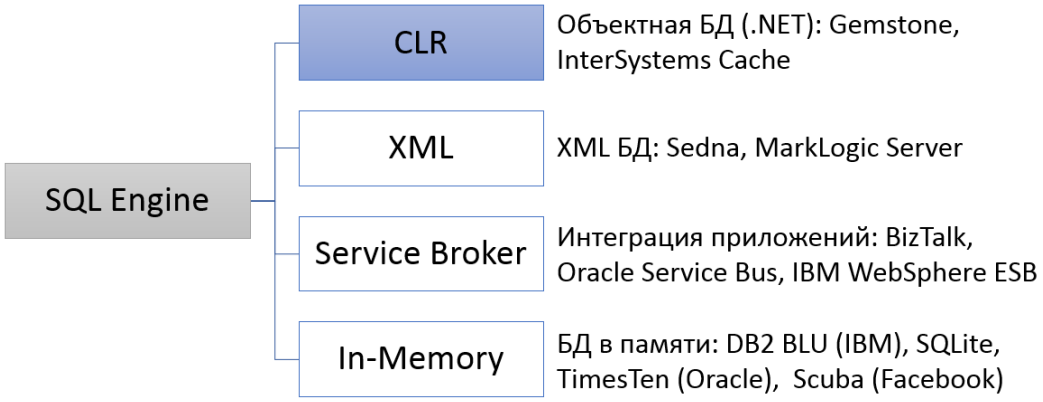
Пример команды для проверки настроек с анонимным доступом:

```
ssbdiagnose CONFIGURATION FROM SERVICE Service1 -S <Computer1> -E -d SBDemo1 TO SERVICE
Service2 -S <Computer2> -E -d SBDemo2 ON CONTRACT DemoContract ENCRYPTION ANONYMOUS
```

Опция **-SHOWEVENTS** позволяет также включить в результат события SQL Server Profiler.

Модель программирования

Принципиальным преимуществом Service Broker как технологии интеграции является возможность напрямую работать с базой данных и использование Transact-SQL как языка программирования. Для меня програмирование на SQL является налучшим способом исполнения бизнес-логики: компактное и выразительное. Тем более, что современные базы данных предлагают разработчикам гибридную модель программирования, принципиально расширяющие традиционные запросы и модульную алгоритмику.



Microsoft предлагает все эти техники программирования сразу же «из коробки», без необходимости приобретения и развертывания дополнительных продуктов, как у других компаний.

> На сегодня программирование **In-memory** доступна во всех редакциях SQL Server, начиная с версии 2016, либо только в коммерческих (платных) редакциях в более старых версиях, начиная с 2012, когда она появилась.

При этом сохраняется разница между даталогическим и объектным мышлением: если нам требуется использовать в базе данных программные объекты для расширения стандартных возможностей, то мы должны использовать уже отдельную технологию программирования .NET. Затем объектные компоненты очень хорошо интегрируются в базу данных, и мы свободно можем использовать эти объекты при программировании на Transact-SQL.

> Таким образом, на основе Microsoft SQL Server мы можем разработать полноценный сервер приложения, или сервер интеграции. Вопрос только в балансировке нагрузки, нашем опыте и подходе при решении задач.

При разработке решений по интеграции мы можем выбрать несколько подходов (стратегии запуска):

- Пакетная обработка данных – простая логика для обмена данными с приемлемой задержкой при обработке данных.

- Событийная обработка данных: обработчик хранимая процедура или внешняя программа.
- Однократный запуск (при старте SQL Server) и постоянное исполнение для максимально быстрой обработки сообщений.

Если мы хотем использовать решение прежде всего для передачи данных, и нам не требуется немедленная их обработка, то мы можем настроить задания для агента SQL Server для периодической отправки или получения сообщений. В этом случае в качестве альтернативы можно также рассмотреть вариант настройки связанных серверов.

Если мы разрабатываем распределенное решение, или хотим использовать аснихронность для балансировки нагрузки, то в этом случае команды для обработки сообщений встраиваются в хранимые процедуры или триггеры.

Для очередей сообщений можно настроить автоматический запуск обработчиков сообщений –активацию. При внутренней активации при поступлении сообщения запускается хранимая процедура. При внешней активации создается событие для программы, работающей независимо от SQL Server.

Транзакции и обмен сообщениями

Отправка и получение сообщений логически являются распределенной задачей, выполняемой различными участниками. Как мы говорили ранее, всегда есть **Инициатор**, который начинает **Диалог**, и, другая стороная – **Цель**, - которая этот диалог продолжает. При обмене сообщениями все стороны равноправны. После выполнения задачи каждая из сторон завершает диалог.

Для надежной обработки данных каждая из сторон выполняет все команды, в т.ч. для сообщений в локальной транзакции. Соответственно, до завершения транзакции отправляемые сообщения накапливаются в буфере, и только при фиксации отправляются получателю. Если при обработке данных или сообщений возникнет ошибка, то будут отменены все команды, выполненные в рамках транзакции.

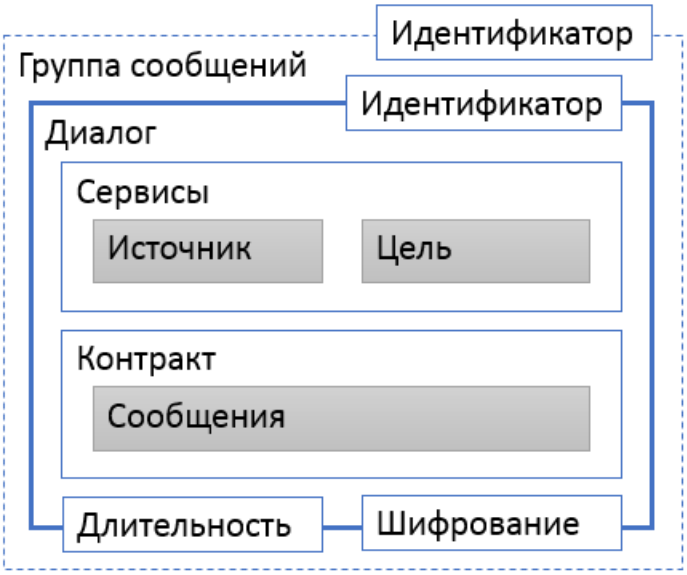
Если выполнять обработку данных и сообщений в разных транзакциях, то нельзя будет гарантировать целостность данных.

> В отличие от распределенных транзакций, которые возникают, например, при использовании связанных серверов, каждая из сторон выполняет команды независимо от другой стороны, что повышает надежность.

Инструкции для диалогов и сообщений

Основным логическим компонентом является **Диалог** как распределенная между несколькими участниками задача. Только после создания диалога можно отправлять сообщения и формировать группы сообщений как комплексную задачу, объединяющую несколько распределенных задач (диалогов).

> По умолчанию каждый диалог формирует собственную группу сообщений. Для испрользования общей группы сообщений в разных диалогах необходимо передавать идентификатор группы сообщений между транзакциями, например, сохраняя его в пользовательской таблице.



Диалог для целевой службы создается автоматически при получении сообщения.

Инструкции для управления диалогом:

- BEGIN DIALOG CONVERSATION – создание сеанса связи между двумя службами для доставки сообщений в рамках определенного контракта.

  Для диалога в опции **RELATED_CONVERSATION** можно указать идентификатор другого диалога, таким образом расширяя число участников и транзакций в обмене сообщений указанного диалога. Также в опции **RELATED_CONVERSATION_GROUP** указать идентификатор групы сообщений.

  По умолчанию диалог не ограничен по времени (задается максимальное значение для int в секундах). Если с помощью опции **LIFETIME** указать длительность, то по истечению времени, если диалог не завершен, будет сгенерирована программная ошибка.

- BEGIN CONVERSATION TIMER – более гибкий способ контроля времени выполнения операций: при истечении заданного интервала в локальную очередь диалога помещается сообщение системного типа `http://schemas.microsoft.com/SQL/ServiceBroker/DialogTimer`.

- END CONVERSATION – завершение сеанса связи, должно выполняться на каждой стороне. При необходиомости можно в опциях **ERROR** и **DESCRIPTION** указать код и описание ошибки. Также имеется опция **CLEANUP** для очистки диалога при его сбое.

Инструкции для групп сообщений:

- GET CONVERSATION GROUP – получение идентификатора группы сообщений для определенной очереди. Возвращает значение, если очередь содержит сообщения. Если имеются сообщения из разных групп, то возвращается группа сообщений с наивысшим приоритетом. Может быть задан интервал (опция **TIMEOUT**) для ожидания поступления сообщений.
- MOVE CONVERSATION – перемещает указанный диалог в определенную группу сообщений.

Инструкции для обработки сообщений:

- SEND – в рамках открытого диалога отправляет сообщение указанного типа.
- RECEIVE – в рамках открытого диалога извлекает сообщение из указанной очереди. На стороне получателя автоматически создает диалог, если есть сообщения. Может быть задан интервал (опция **TIMEOUT**) для ожидания поступления сообщений.

Инструкция **SELECT** также позволяет читать сообщения из очереди, но не извлекает их из нее. В зависимости от настроек очереди инструкция **RECEIVE** извлекает сообщения, либо меняет статус (поле **status = 0**), если у очереди установлена опция **RETENTION = ON**.

Очередь имеет набор полей:

- `service_id, service_name` – идентификатор и название сервиса
- `service_contract_id, service_contract_name` – идентификатор и название контракта
- `message_type_id, message_type_name` – идентификатор и название типа сообщения
- `message_body` – содержимое сообщения
- `validation` – проверка сообщения: **E**=пусто, **N**=нет, **X**=XML
- `status` – состояние сообщения: 0 = готово, 1 = получено сообщение, 2 = еще не завершено, 3 = отправленное сообщение сохранено
- `priority` – приоритет диалога
- `queuing_order` – порядковый номер сообщения

Программные шаблоны

Рассматривается обработка сообщений внутри базы данных (внутренняя активация).

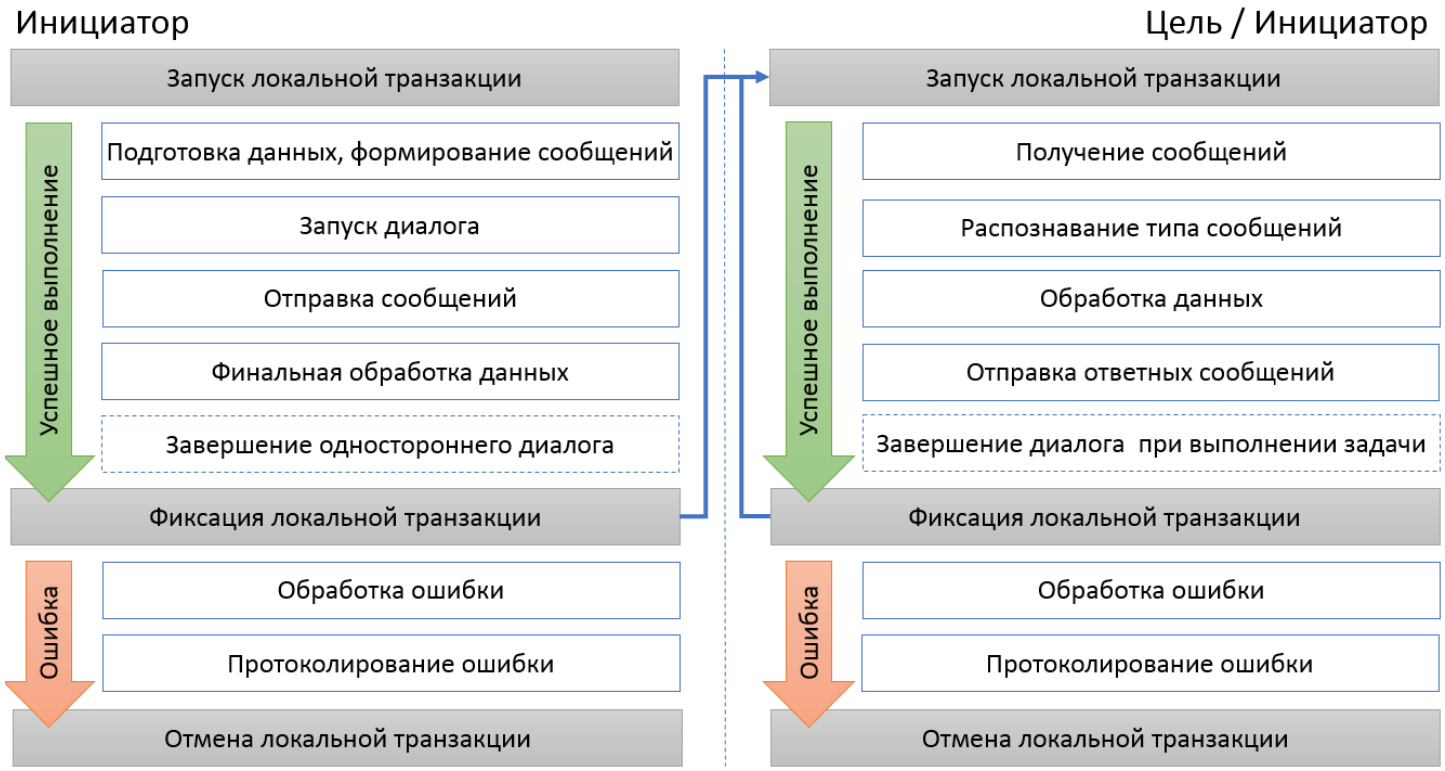
Запуск диалога и обработка сообщений может выполняться в хранимой процедуре или триггере, а также в задаче SQL-агента. Нельзя выполнять соответствующие инструкции в пользовательских функциях.

> Также можно выполнять инструкции в ручном режиме, либо из кода приложения, например, с помощью ADO.NET.

Отправка сообщений является относительно простой задачей, для которой требуется сформировать сообщение, и, при необходимости, задать группу сообщения или подключиться к существующему диалогу.

Для обработки ошибок рекомендуется использовать конструкцию TRY..CATCH.

Затем запускается транзакция и выполняются инструкции по управлению диалогом, обработке сообщений и данных.



Шаблон скрипта для создания диалога и отправки сообщения:

```
DECLARE @h uniqueidentifier
BEGIN TRY
  BEGIN TRAN
    BEGIN DIALOG CONVERSATION @h
    FROM SERVICE <initiator_service> TO SERVICE '<target_service>'
    ON CONTRACT <service_contract>;
    -- Подготовка данных
    -- Формирование сообщений
    SEND ON CONVERSATION @h MESSAGE TYPE <message_type_name> (<message_body>);
    -- Финальная обработка данных
    /* Завершение одностороннего диалога
    END CONVERSATION @h; */
  COMMIT
END TRY
BEGIN CATCH
  -- Распознавание локальной ошибки
  -- Протоколирование ошибки
  ROLLBACK
END CATCH
```

Получение и обработка сообщений является более сложной процедурой. С помощью команды **RECEIVE** можно извлекать необходимые поля из очереди (см. в предыдущем разделе).

Сообщения могут извлекаться из очереди несколькими способами:

- Извлечение одного сообщения (обычно в цикле) в переменные.
- Извлечение нескольких сообщений в табличную переменную.
- Обработка сообщений с помощью курсора.

Далее рассматривается первый вариант обработки.

При получении сообщения с помощью инструкции **RECEIVE** из очереди извлекается извлекается идентификатор диалога для ответа или его закрытия, тип сообщения и само сообщение. С помощью системной функции **@@ROWCOUNT** проверяется наличие данных, а затем проверяется тип сообщения, в зависимости от которого выполяется дальнейшая обработка.

Шаблон скрипта для получения сообщения:

```
DECLARE @h UNIQUEIDENTIFIER
DECLARE @messagetypename NVARCHAR(256)
DECLARE @messagebody XML

BEGIN TRY
  BEGIN TRANSACTION
    WAITFOR (
      RECEIVE TOP (1)
        @h = conversation_handle,
        @messagetypename = message_type_name,
        @messagebody = CAST(message_body AS XML)
      FROM TargetQueue
    ), TIMEOUT <interval>;
    IF (@@ROWCOUNT > 0) BEGIN
      IF (@messagetypename = '<message_type_name>') BEGIN
        -- Обработка сообщения
        -- Формирование ответного сообщения
        SEND ON CONVERSATION @h MESSAGE TYPE <message_type> (<message_body>);
        END CONVERSATION @h; -- Завершение диалога
      END
    END
  COMMIT
END TRY
BEGIN CATCH
  ROLLBACK TRANSACTION
END CATCH
```

В примере сообщение преобразуется сразу же в XML, в более сложных случаях это зависит от типа сообщений.

Помимо сообщений, определенных в контракте, могут такде приходить системные сообщения, например об ошибке, завершении диалога или событие таймера.

Для обработке ошибки после чтения сообщения и перед его обработкой можно установить точку сохранения транзакции **SAVE TRANSACTION**. При этом в случае необходимости повторной обработки (например, при взаимоблокировке), отменяется вся транзакция полностью. При обработке неправильного сообщения выполняется протоколирование ошибки и откат транзакции до точки восстановления.

Шаблон скрипта для получения сообщений в цикле и обработки ошибок:

```
DECLARE @h UNIQUEIDENTIFIER
DECLARE @messagetypename NVARCHAR(256)
DECLARE @messagebody XML

WHILE (1=1) BEGIN
  BEGIN TRANSACTION
    WAITFOR (
      RECEIVE TOP (1)
        @h = conversation_handle,
        @messagetypename = message_type_name,
        @messagebody = CAST(message_body AS XML)
      FROM TargetQueue
    ), TIMEOUT <interval>
```

```
    IF (@@ROWCOUNT = 0) BEGIN
      ROLLBACK TRANSACTION
      BREAK -- Завершение обработки если нет сообщений в очереди
    END
    SAVE TRANSACTION MessageReceivedSavepoint
    IF (@messagetypename = '<message_type_name>') BEGIN
      BEGIN TRY
        -- Обработка сообщения
        -- Формирование ответного сообщения
        SEND ON CONVERSATION @h MESSAGE TYPE <message_type> (<message_body>);
        END CONVERSATION @h; -- Завершение диалога
      END TRY
      BEGIN CATCH
      IF (ERROR_NUMBER() = 1205) BEGIN -- Взаимоблокировка
        ROLLBACK TRANSACTION -- Откат транзакции
        CONTINUE -- Продолжение обработки событий
      END ELSE BEGIN
        ROLLBACK TRANSACTION MessageReceivedSavepoint
        PRINT 'Error occurred: ' + CAST(@messagebody AS NVARCHAR(MAX))
      END
    END CATCH
  END
  IF (@messagetypename = 'http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog') BEGIN
    END CONVERSATION @h;
  END
  COMMIT TRANSACTION
END
```

Обработка группы сообщений

Для включения диалога в группу нужно передать идентификатор группы в соответствующий параметр (**RELATED_CONVERSATION_GROUP**):

```
DECLARE @h uniqueidentifier
DECLARE @gc uniqueidentifier = NEWID()

BEGIN TRY
  BEGIN TRAN
    BEGIN DIALOG CONVERSATION @h
    FROM SERVICE <initiator_service> TO SERVICE '<target_service>'
    ON CONTRACT <service_contract>
    WITH RELATED_CONVERSATION_GROUP = @gc;
    -- Сохранение идентификатора группы сообщений
```

Для включения других диалогов в эту же группу требуется сохранить идентификатор группы и использовать его повторно.

Для обработки группы сообщений нужно сначала сделать запрос идентификатора группы из очереди, а затем во вложенном цикле запросить сообщения из группы:

```
DECLARE @cg UNIQUEIDENTIFIER;
WHILE (1 = 1) BEGIN
  BEGIN TRANSACTION;
    WAITFOR (
      GET CONVERSATION GROUP @cg FROM <queue>
    ), TIMEOUT <interval>
    IF (@@ROWCOUNT = 0) BEGIN
      ROLLBACK
      BREAK
    END
```

```
    DECLARE @h UNIQUEIDENTIFIER;
    DECLARE @messageTypeName NVARCHAR(256);
    DECLARE @messageBody XML;
    WHILE (1 = 1) BEGIN
      WAITFOR (
        RECEIVE TOP (1)
          @messageTypeName = message_type_name,
          @messageBody = CAST(message_body AS XML),
          @h = conversation_handle
        FROM <queue>
        WHERE conversation_group_id = @cg
      ), TIMEOUT <interval>
```

Нотификация событий о проблемных сообщениях

Если для очереди включена опция обработки сбоев **POISON_MESSAGE_HANDLING (STATUS = ON)**, то после 5 последовательных откатов транзакции очередь будет отключена. Для обработки таких ситуаций можно создать нотификацию событий о выключении очередей и создать специальную очередь для системных сообщений:

```
CREATE QUEUE PoisonMessageNotifyQueue
GO
CREATE SERVICE PoisonMessageNotifyService ON QUEUE PoisonMessageNotifyQueue (
  [http://schemas.microsoft.com/SQL/Notifications/PostEventNotification]);
GO
CREATE EVENT NOTIFICATION PoisonMessageNotification ON QUEUE TargetQueue
FOR Broker_Queue_Disabled
TO SERVICE 'PoisonMessageNotifyService', 'current database'
GO
```

После анализа и решения проблем с сообщениями нужно включить очередь сообщений:

```
ALTER QUEUE TargetQueue WITH STATUS = ON
```

Обработка прикладных ошибок

Для улучшения диагностики можно создавать и обрабатывать прикладные ошибки. Для отправки сообщения об ошибке нужно использовать специальный тип сообщения или завершить диалог с кодом и описанием ошибки:

```
END CONVERSATION @h
WITH ERROR = 4242
DESCRIPTION = 'Error message'
```

Сообщение об ошибке будет передано как XML:

```
<Error xmlns="http://schemas.microsoft.com/SQL/ServiceBroker/Error">
<Code>4242</Code>
<Description>Error message</Description>
</Error>
```

При проверке типа сообщения нужно использовать системный тип ошибки и извлекать из сообщения код и описание ошибки:

```
IF (@messagetypename = 'http://schemas.microsoft.com/SQL/ServiceBroker/Error') BEGIN
  -- Извлечение кода и описания ошибки
  SET @errorcode = (SELECT @messagebody.value(N'declare namespace
brokerns="http://schemas.microsoft.com/SQL/ServiceBroker/Error";(/brokerns:Error/brokerns:Code)[1]',
'int'));
  SET @errormessage = (SELECT @messagebody.value('declare namespace
brokerns="http://schemas.microsoft.com/SQL/ServiceBroker/Error";(/brokerns:Error/brokerns:Description)[
1]', 'nvarchar(3000)'));
  -- Протоколирование и обработка ошибки
  END CONVERSATION @h; -- Завершение диалога
END
```

Ссылки

1. Pro SQL Server 2008 Service Broker. Klaus Aschenbrenner. APRESS.
2. SQL Server Service Broker
3. Компонент SQL Server Service Broker
4. Общие сведения (компонент Service Broker)
5. Основные понятия программирования компонента Service Broker
6. Логическая архитектура [компонент Service Broker]
7. Архитектура диалога
8. Уведомления о событиях
9. Настройка безопасности диалогов для уведомлений о событиях
10. Представления каталога компонента Service Broker (Transact-SQL)
11. Динамические административные представления, связанные с компонентом Service Broker (Transact-SQL)
12. Утилита для диагностики ssbdiagnose (компонент Service Broker)
13. SQL.RU: Статьи и обсуждение на русском практики использования Service Broker.
14. Rusanu Consulting: Статьи на английском с примерами использования Service Broker