Sunday morning T-SQL

Using Service Broker to send messages

2013-04-142013-04-06 / DANIEL HUTMACHER

Until now, I've never really looked any closer at Service Broker and message queues in SQL Server, but it turns out it's a really useful feature if you need asynchronous processing or any other kind of queued messaging logic. Messages can be sent within a database, a server, or even between servers.

Setting up the database

For this tutorial code to work, you first need to enable Service Broker for the database you're working in:

```
ALTER DATABASE AdventureWorks2008R2 SET ENABLE_BROKE
```

Now, we'll set up the message type(s) – these are basically templates for the messages you'll be sending, including any validation that you would like to add. In this example, we're not going to validate messages.

```
CREATE MESSAGE TYPE TheMessageType VALIDATION=NONE;
```

Contracts are used to connect the sending and receiving services later on.

```
CREATE CONTRACT TheContract (
    TheMessageType SENT BY INITIATOR
);
```

Create the two queues that will be used. We're going to use the SenderQueue as our sender and the RecipQueue as the recipient. The sender queue is used to send, the recipient queue is polled by the recipient.

```
CREATE QUEUE SenderQueue;
CREATE QUEUE RecipQueue;
```

Now, create two services, one for each queue. When defining the services, we also define their contracts.

```
CREATE SERVICE SenderService ON QUEUE SenderQueue(Th
CREATE SERVICE RecipService ON QUEUE RecipQueue(TheC
```

Sending a simple message

Finally, we can transmit our message. The @conversation variable is used as a handle that points to the conversation, so you can specify which conversation is used to send the message.

```
DECLARE
    @conversation uniqueidentifier,
    @msg varchar(max)='Hello world!';

--- Start a conversation:
BEGIN DIALOG @conversation
    FROM SERVICE SenderService
    TO SERVICE N'RecipService'
    ON CONTRACT TheContract
    WITH ENCRYPTION=OFF;

--- Send the message
SEND ON CONVERSATION @conversation
    MESSAGE TYPE TheMessageType
    (@msg);
```

Retrieving the message

Meanwhile, on a different connection, you can retrieve the message (or, actually, wait for one to appear, and then retrieve it) using the following code.

```
DECLARE
    @conversation uniqueidentifier,
    @senderMsgType nvarchar(100),
    @msg varchar(max);

WAITFOR (
    RECEIVE TOP(1)
        @conversation=conversation_handle,
        @msg=message_body,
        @senderMsgType=message_type_name
    FROM RecipQueue);

SELECT @msg AS RecievedMessage,
       @senderMsgType AS SenderMessageType;

END CONVERSATION @conversation;
```

In this highly simplified example, we end the conversation once we've received a message, without replying and ignoring any other messages that may be in the RecipQueue.

Using a stored procedure that runs on incoming messages

You can create a procedure that automatically runs whenever a message arrives on a queue. This is called "internal activation". First off, we'll build a table (msg) where we'll store incoming messages.

```
CREATE TABLE dbo.msg (
    message_id    int IDENTITY(1, 1) NOT NULL,
    [date]        datetime NOT NULL,
    [message]    varchar(max) NOT NULL,
    CONSTRAINT msg_pk PRIMARY KEY CLUSTERED (message
);
```

After that, we'll create the stored procedure that does pretty much the same work as the recipient code we just looked at, except instead of displaying the incoming message interactively with a SELECT statement, the procedure stores it in the msg table with INSERT:

```
CREATE PROCEDURE dbo.sp_MessageReader
AS

DECLARE
    @conversation uniqueidentifier,
    @senderMsgType nvarchar(100),
    @msg varchar(max);

WAITFOR (
    RECEIVE TOP(1)
        @conversation=conversation_handle,
        @msg=message_body,
        @senderMsgType=message_type_name
    FROM RecipQueue);

IF (@senderMsgType='TheMessageType')
    INSERT INTO dbo.msg ([date], [message])
    SELECT GETDATE(), @msg;

END CONVERSATION @conversation;

GO
```

Now, all we need to do is to tell the recipient queue, RecipQueue, that it should run this procedure every time a message arrives (unless the procedure is already running). This is done with the ALTER QUEUE statement:

```
ALTER QUEUE RecipQueue WITH ACTIVATION (
    STATUS=ON,
    PROCEDURE_NAME=dbo.sp_MessageReader,
    EXECUTE AS SELF,
    MAX_QUEUE_READERS=1);
```

The MAX_QUEUE_READERS parameter is how many parallel instances of the procedure can be run if there's a large number of incoming messages.

Now, try to send a message again, and check the contents of the dbo.msg table. You may notice a delay of a few fractions of a second, but that's how queues work, right?

Removing all the objects

Finally, to clean up all these Service Broker objects that we've created, use the following DROP statements:

```
DROP SERVICE RecipService;
DROP QUEUE RecipQueue;
DROP SERVICE SenderService;
DROP QUEUE SenderQueue;
DROP CONTRACT TheContract;
DROP MESSAGE TYPE TheMessageType;
```

More on the subject

There's a ton of features that aren't covered in this very brief introduction. Check out the following starting points on MSDN for more reading:

- MSDN: Service Broker Tutorials (http://msdn.microsoft.com/en-us/library/bb839489.aspx)
- MSDN: SQL Server Service Broker (http://msdn.microsoft.com/en-us/library/bb522893.aspx)

Advanced, SQL Server concepts, T-SQL
CONTRACT **CONVERSATION** **DIALOG**
MESSAGE TYPE **MESSAGING** **QUEUE**
RECEIVE **SEND** **SERVICE** **SERVICE BROKER**