7

设计MQSI消息流

MQSI 具有设计和开发消息流的功能，因而您可以将请求路由到后台系统，以便根据应用程序要求进行检索、聚集和／或更新信息。使用 MQSI 控制中心，您可以创建消息流，并可将其应用到一个或多个 MQSI 代理。

本章讲述如何设计 MQSI 消息流以便执行这些功能。第 237 页的第十四章 “ 创建 MQSI 应用程序”将详细讲述使用这些功能的应用程序的完整实例。

7.1 MQSI应用程序设计

毫无疑问，我们需要清楚地了解需要消息流完成的商业操作。逻辑设计以高超的水平描述各种要求，并为定义 WebSphere 应用程序组件和 MQSI 域之间的操作合同提供关键性帮助。针对 MQSI 实施满足操作合同要求的解决方案涉及若干消息路由和转换步骤。同时，根据您企业的具体情况，可能需要不止一个代理程序。

我们使用的实例应用程序具有以下特征：

- 接受来自 WebSphere 的请求，以便减轻从 Guild 向各站传输单元时记录交易的负担。
- 接受来自用户界面层的请求，以便创建可返回和显示的活动报表供用户使用。

在 MQSI 中，每个转换和路由过程都是作为“ 消息流 ”开发的。消息流创建为一系列的操作，每项操作都采取节点的形式。各种类型的节点都具有相应的连接器，允许直观地定义节点的输入和输出。

为了完成应用程序所要求的任务，我们需要能够执行以下操作的消息流：

- 通过输入队列从用户处获取请求
- 把消息分成若干后端请求
- 接收后端回复
- 应用转换规则，并使消息相互关联
- 执行数据库操作，以便缓存或增强数据
- 将回复消息返回用户，供其使用

在 MQSI 中可以使用各种节点创建消息流，许多消息流具有高度可配置的特性。消息流应用到某个代理程序的执行引擎，根据在消息流中开发的商业规则，其可接收和处理消息。

如若设计复杂的消息流，则考虑其在较广泛的上下文应用程序内所起的作用很有帮助。实际上，消息流就是有关文档交换的所有情况，其中涉及诸如文档转换、数据库操作之类的操作。

进入 MQSI 中的开发环境并将一组大量相互连接的节点集中以执行全面而复杂的消息转换，是非常简单的事情。遗憾的是，此类消息流开发方式将来会导致极高的维护费用。

可通过一般输入和输出终端配置消息流，这样，其他消息流就可将该消息流作为其组件进行重复利用。这些类型的消息流不是应用到代理程序，而是作为大型消息流的组成部分。

金科玉律：各个消息流组件越简单越好！此后可比较容易地测试和使用这些消息流组件。如果消息流包含许多节点及更多连接，跟踪其中的故障就会非常困难。

7.1.1 与应用程序的设计合同

作为高级概念，应用程序与 MQS 的合同定义要提供的消息文档、有哪些预期操作以及预期的回复文档。该合同包括诸如是同步还是异步执行一项操作这样的详细信息。

该过程将从 MQSI 端接收消息以启动所需操作。这些操作可能包括以回复形式存在的检索和返回信息、更新其他系统中的数据，或根据合同细节对上述情况的任意组合。

7.1.2 消息流结构

为处理此合同而应用的完整消息流可获取传入文档，并执行所需的系列功能，这些功能负责完成合同中规定的商业操作。

开发标准文档的需要非常明确，这些标准文档支持 MQSI 和 WebSphere 应用程序之间的通信。但是，要完成整个工作，可能需要把原始文档多次转换为其他形式，从而执行各项操作。为此，开发文档标准还有很大的好处，即支持子消息流之间的交互。

可将基于标准传入文档类型的通用数据库操作保存为消息流组件。为使用该类组件，必须定义通用文档转换并将其保存为消息流组件。

下面让我们看看作为 MQSI 消息流组件部分，如何将建模的商业操作链接以形成完整的商业流程。例如，具有“事务日志”文档类型的传入消息可能需要某些转换，才能创建可用于执行数据库操作的中间文档。

还可能创建这样的消息流：可分析传入消息的内容并将该消息路由到能够使用该类消息的子消息流。

7.1.3 定义文档类型

要使消息流组件彼此兼容，必须定义在消息流组件间交换的文档类型。定义必须详细描述将用来解释消息的分析程序，以及消息本身的结构和内容。

对于 XML 文档，是通过使用文档类型定义（DTD）实现的。在本红皮书中，我们假定读者了解一些有关 XML 的基本知识，并且使用 DTD。有关 XML 的信息，请参见《XML 文件：在企业到企业和企业到消费者应用中使用 XML》编号：SG24－6104。

实例 DTD

下面是 XML DTD 的简单实例。实例 7－1 中的 DTD 描述名为“视图－报表－命令”的文档。请注意 DTD 如何定义构成该“视图－报表－命令”的数据元素。

实例 7－1：XML DTD

```
<!－－view__stmt__cmd.dtd －－>
<!ELEMENT view－statement－command (user, interest－rate+) >
<!ELEMENT user (station, role) >
<!ELEMENT station (#PCDATA) >
<!ELEMENT role (#PCDATA) >
<!ELEMENT interest－rate (type, year, credit－rating, duration) >
<!ELEMENT type (#PCDATA) >
<!ELEMENT year (#PCDATA) >
<!ELEMENT credit－rating (#PCDATA) >
<!ELEMENT duration (#PCDATA) >
```

此 DTD 中有两个子元素：用户和利率。每个元素都包含其他元素。例如，用户元素包含站元素和角色元素，并在用户元素下面定义。定义一直维持这种情况，直到使用一种给定数据表示方式定义叶元素（数据本身）。这里显示的所有元素都具有类型#PCDATA，该术语的意思是“可分析的字符数据”，它基本上是一个字符串。

7.2 消息流组件

MQSeries Integrator 主要以基于 MQSeries 产品提供的消息传输层为基础。MQSI 内工作的消息流必须把 MQSeries 消息作为其输入，并可能产生一条或几条 MQSeries 消息作为其输出。

MQSI 具有编程接口，它使您可以创建提供新消息处理功能或替换现有功能的新节点。您还可以创建新的消息分析程序，从而访问 MQSI 未定义的消息类型。

但是，应注意，MQSI 是作为可扩展的消息代理程序框架而设计的，而且将来此框架可能用于从非 MQSeries 的其他来源获取输入。例如，考虑这样一个 MQSI 节点的潜力，该节点可把直接输出流从某个输入流馈送到 EJB。

由于此可扩展性，建议经常访问位于以下地址的 MQSeries 系列 SupportPacs 网页：http：／／www.ibm.com／software／ts／mqseries／txppacs／，从而了解有关支持产品扩展的最新信息。

至于本红皮书，我们将会把 MQSeries 消息作为用于应用程序服务器和应用程序路由器之间进行通信的媒体，并以此为基础进行探讨。

消息流作为逻辑连接的事件序列进行实施，这些事件以消息流 *节点*的形式进行定义。每个节点执行消息流中的一个特定功能。MQSI 包含一组消息流节点，它们称为“ IBM 基元”，而且具有种类繁多的功能。

本章介绍以下消息类型：

- MRM
- NEON
- XML
- BLOB

我们开发的应用程序使用 XML 消息，这也是我们讨论的焦点所在。

我们还将讨论以下基元节点类型：

- 转换节点
- 数据库节点
- 逻辑控制节点
- 可复用消息流

此外，我们还将讨论提供分解和重组功能的节点类型。这些节点是通过 IBM MQSI Aggregator 插件提供的。

7.2.1 消息类型

流经系统的消息通常包含特定格式和内容。在消息发送术语中，称之为“ 消息模板 ”。此模板拥有描述消息中所包含数据结构的信息。可使用 MQSI 中的功能定义多种消息模板，这些功能可定义在转换操作中使用的消息模板信息。也可使用 MQSI 内的定制 XML 消息，这种方法日益普遍。

至于 MQSI 内的转换和编程，许多节点类型可以使用一种非常有效的内部 ESQL 语言。ESQL 可用于交换管理结构化消息或数据库表的内容。这在处理以 XML 格式或其他消息格式保存的消息内容时，功能尤其强大。

MQSI 可配置为接受多种消息类型。作为起点，MQInput 节点的 MQSI 可以接受包含简单字符串消息正文的 MQSeries 消息。

但要在 MQSI 内使用，则 MQSI 的许多可用分析程序中必须有一个可以接受该消息。发送端应用程序负责包含充足的信息，从而使 MQSI 能够利用其消息。为此，可在标准的 MQSeries 消息正文开头包含另一个称为 MQRFH2 标题的消息标题，这第二个消息标题包含 MQSI 可用来解释消息正文的详细信息。如果没有 MQRFH2 标题，MQSI 就尝试选择最适合的分析程序来解释消息内容。

有关 MQRFH2 标题中可以使用的结构和选项的详细信息，请参见《MQSeries Integrator 编程指南 2.0.2》编号：SC34－5603－02。

在完全分析位流当前部分之后，每个分析程序都有决定接下来需要什么分析程序的作业。如果当前分析程序提名代理程序不可用的分析程序作为其后继者，则使用输入节点上定义为“ 消息域 ”的默认设置。

消息集和 MRM

在 MQSI 2 版本中，消息存储库管理器（MRM）是“ 控制中心 ”中的一个内置图形工具，它用于定义要在消息流中使用的结构化数据和消息定义（消息模板）。

通过定义可整理为复合数据类型的各个数据元素，可以定义可复用的数据结构，并可将这些数据结构用作构造完备的消息布局中的构件，因而能够快速进行复杂消息结构的 GUI 驱动型开发。这样的结构可以组合为各个超集，称为“ 消息集 ”。

MRM 的独特优点是，通过导入现有的已定义消息结构（用 C 或 COBOL 定义），能够创建消息集内容。

MRM 中定义的数据结构可与一个以上分析程序共同使用。这样就可以将同一逻辑数据结构与不同物理表示一起使用。两个主要选项是：

- 定制线路格式（Custom Wire Format） （CWF），其中，分析器接受／创建结构化文档，在此文档中，用给定数据类型（在字符串数据的情况下，还用“ 长度属性 ”）定义每个字段。
- 可扩展标记语言（XML），其中，分析器接受／创建 XML 文档，该文档具有 MRM 结构描述的窗体。这里需要注意的是，这与我们后面将要讲述的一般 XML 分析程序不同。

如果消息结构中某个地方也提供长度值，则支持可变长度字符串。

在 MRM 中定义消息定义，并保存为消息集时，消息流节点可利用控制中心的拖放编程功能。

尽管 MRM 具有一些强大功能，但在某些应用程序中使用 MRM，会限制在处理消息字段格式时所必需的灵活性。例如，在 MQSI 2.0.2 中，目前不支持标记分隔数据或不支持长度字段的可变长度字符串。

NEON 格式化程序

MQSI 2 版以基元节点的形式提供 MQSI 1 版中拥有的 NEON 规则和格式化程序功能。这就为目前实施 MQSI 1 版产品的客户提供了直接迁移路径。

NEON 格式化程序是功能强大的数据格式定义工具，具有与消息集及 MRM 相似的功能。当需要分析标记分隔输入时，此工具尤其有用。

XML

MQSI 的最大优势之一是能够分析和管理用 XML 文档形式表示的消息数据。

MRM 中定义的消息结构可以作为 XML 消息进行分析，但这具有内在的局限性，因为 MRM 主要用来提供与历史系统数据格式的兼容性，这就使 XML 在其灵活的定制数据表示方式中可提供的优点无法得以发挥。

为此，可以使用一般 XML 文档分析程序，该分析程序能够分析任何标准格式的 XML 消息正文。尽管可提供 MQRFH2 标题来明确地把消息正文声明为 XML，但不必如此，因为 MQSI 会自动检测传入的消息正文是否为 XML 消息。

作为数据表示新标准，XML 的采用速度非常快。XML 是一种真正的便携式数据交换媒体，因而是 Web 应用程序中数据表示的自然选择。为此，我们在自助实例中选择使用一般 XML 消息格式。

MQSI 2 版中目前存在的限制是，无法以图形方式单独完成基于一般 XML 文档的转换节点编程。但是，完全可以用 MQSI 节点编程语言 ESQL 对这样的转换逻辑进行编码。事实上，这种方式更为容易。

BLOB

如果没有提供分析程序，或提供的分析程序不可用，BLOB 就是默认分析程序。BLOB 分析程序有两个元素：

- 一个元素称为 BLOB(其 ESQL 值类型为 BLOB)。这包含可以引用和操作的位流。
- 另一个元素是 UnknownParserName。由于没有与该元素关联的格式，因此，要使用该元素，首先必须理解位流。

最好定义一个 MRM，这样就可按名称引用——与位流中的偏移相反。如果您自己解释位流，要牢记其位流本质。

7.3 IBM基元节点

在实施任何消息流之前，MQSI 控制中心会制作一组基本节点，称之为 IBM 可用基元。这些基本节点是基本构件，将其进行组合可构成完整的消息流。

每个基元节点都有一组输入和输出终端，这些终端可用于把给定节点的输出连接到另一个节点的输入。例如，MQInput 节点就是一种可配置为从 MQSeries 队列获取消息的节点。MQInput 节点具有输出终端，可以把该终端连接到计算机节点（可配置功能的另一种类型，可促进消息转换）的终端。

在《使用控制中心的 MQSeries Integrator 2.0.2 版》编号：SC34－5602－03 一书中，详细讲述可在 MQSeries Integrator 中使用的全部基元节点。在这里，我们主要讨论创建自助模式实例时发现的有用节点和功能。多数消息流中一般都可以找到这些节点（聚合节点除外）：

- MQInput
- 计算
- 数据库节点
- 过滤器
- MQOutput
- MQReply
- 可复用消息流
- 聚合节点支持包 IA72

7.3.1 MQInput节点

MQInput 节点是所有消息流的开端。MQInput 节点从其分配的队列接收消息，并把消息通过其输出节点传递到另一个节点的输入。

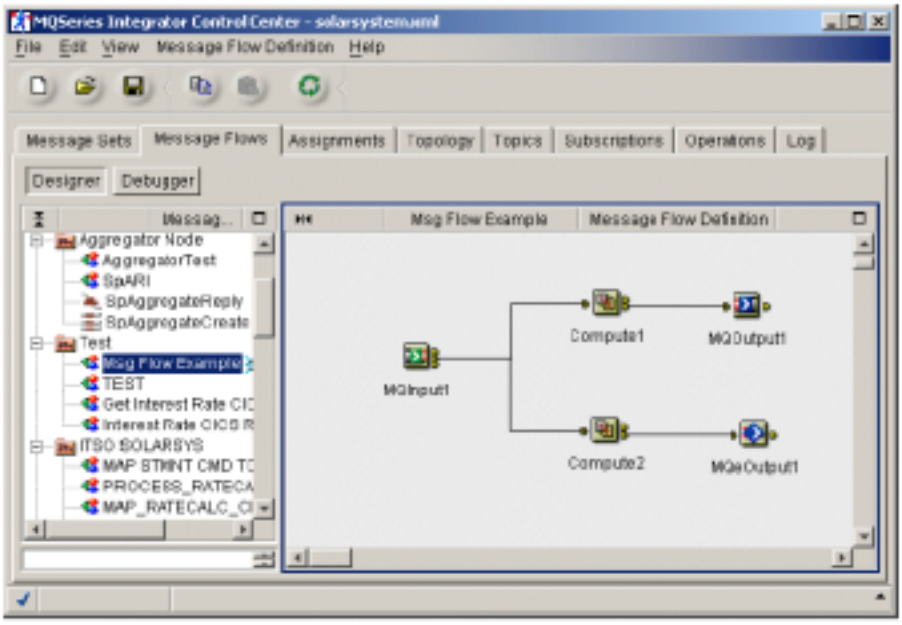

图7－1 有多个子消息流的MQInput 节点

如图 7－1 所示，MQInput 节点是消息流的开端，但是，可在 MQInput 输出终端与多个节点的输入终端之间建立连接。

这样，传入的消息就可以启动一个以上的事件流。例如，您可能会获取一条用来携带某客户余额请求和个人利率查询的消息，并通过多个子消息流把该消息发送出去，每个子消息流用来访问不同的后台系统。

7.3.2 转换节点

转换节点执行获取输入消息、并根据需要将其映射到输出消息的任务。

计算节点

计算节点是可从调色板上使用的最通用转换节点之一。它可创建输出消息，并可用来执行几乎所有数据操作。为此，计算节点可能会需要一个以上输入数据源。显然，主要输入源是经由计算节点输入终端的传入消息。计算节点有选择地把数据库作为其功能的组成部分。

计算节点拥有接收输入消息的输入终端，及两个输出终端：一个处理正常操作下的输出，另一个在转换过程发生故障时重新定向原始消息。

可通过计算节点的“属性”窗口配置计算节点，如图 7－2 所示。

让我们看看“属性”窗口的布局。主窗口分为两个部分：输入消息和输出消息。计算节点可执行的最简单操作是把输入消息复制到输出消息。这可通过选中“复制整个消息”单选按钮来实现。选中“复制消息标题”单选按钮可自动生成代码，该代码可将消息正文之前的输入消息的各部分复制到输出。

为执行这些任务而生成的代码显示在 ESQL 选项卡中。

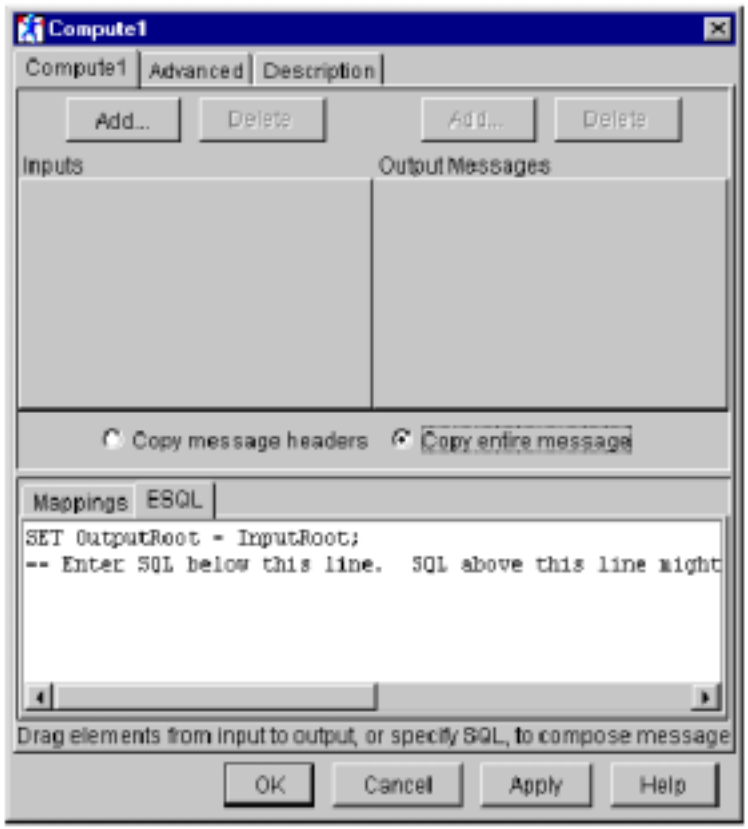

图 7－2 计算节点窗口

在每个案例中都会生成一个注解行，建议您把可添加到节点的任何定制代码添加到该注解行下面。之所以这样，是因为可使用 GUI 构造转换细节。这些操作的结果会影响节点属性中包含的 ESQL 代码。自动生成的代码保持在注解行以上；用户代码可在注解行之下输入。

使用“ 控制中心 ”配置计算节点时，您很快会发现 ESQL 窗格的大小有限，这使读取节点背后的代码极为困难。使操作 ESQL 较为容易的技巧是使用 Ctrl+a 选中节点后面的所有代码，再用 Ctrl+c 复制该选定内容，然后使用 Ctrl+v 把内容粘贴到您喜欢的文本编辑器中进行编辑。代码更改完成后，可以把代码复制／粘贴回 ESQL 窗格。

ESQL 窗格的功能之一是在您键入时会动态验证代码语法。如果您选择在此窗格中编辑代码，当代码中出现语法错误时，代码窗格下就会出现一个红色“ X ”。当您在代码的结构内尝试新的技巧或功能组合时，此功能非常有用。

使用 ESQL 进行消息转换

ESQL 是一种功能强大的数据转换语言，可用于包括计算节点在内的许多 MQSI 转换节点。ESQL 能够和用 SQL 方式执行常规数据库操作一样的方法在消息内操作结构化数据。这样，就可以自由地在消息内容和数据库数据源之间交换数据，作为“ 计算 ”操作的组成部分。

“ 根 ”元素是消息结构的最高级别元素。由于计算节点拥有输入和输出消息终端，因此这些元素分别称为 InputRoot 和 OutputRoot。构成整个文档的元素被视为“ 根 ”元素的子元素，并以如下方式引用：

```
  InputRoot.element1
  InputRoot.element2
```

下面是可放在队列上的 XML 消息的实例：

```
<transactionrecord>
  <station>JUPITER<／station>
  <amount>1600<／amount>
  <transdate>20001－11－09 23：00：51.123000<／transdate>
<／transactionrecord>
```

在消息流中，当通过 MQInput 节点从队列检索消息，并将其传递到进行消息发送的计算节点时，该消息具有以下形式结构：

```
Root
  Properties
    CreationTime=GMTTIMESTAMP '1999－11－24 13：1：'
       (a GMT timestamp field)
        ..and other fields...
  MQMD
    PutDate=DATE '19991124'
       (a date field)
    PutTime=GMTTIME '131 '
       (a GMTTIME field)
      ...and other fields...
  MQRFH
    mcd
    msd='xml'
       (a character string field)
      ..and other fields...
  XML
    transactionrecord
      station=Jupiter
         (a character string field)
```

```
amount=1600
   (a character string field)
transdate=20001－11－09 23：00：51.123000
   (a character string field)
```

我们可以从这里看到在消息内发现的多种子结构：

- 一般消息属性
- MQSeries 消息描述符（MQMD）
- MQSeries 集成器规则和格式化标题 （MQRFH2）
- 在此案例中，消息正文本身是 XML 文档

要引用输入消息内的 transactionrecord 数据元素，我们可使用 ESQL 表达式：

```
InputRoot.XML.transactionrecord
```

或

```
InputRoot.XML.（XML.tag）transactionrecord
```

后者显示一个实例，其中事务节点被完全限定为 XML 标记元素。尽管在上面的实例中，并非必须使用此完全限定的批注，但把（XML.tag）限定符包含在内是很好的做法，因为在带有 XML 文档标题的标准格式 XML 文档中，分析程序可能会将 XML 消息正文标记的引用错误解释为 XML 文档标题属性的引用。

上面概述了 ESQL 中数据元素引用的基本形状。有关详细信息，请参见《MQSeries Integrator ESQL 引用 2.0.2 版》编号： SC34－5923－00，其中的引用是相当全面的 ESQL 引用。该手册讲述了可用 ESQL 建立的全套有效构造，在此我们不再重复。相反，我们将继续集中介绍建立实例所使用的技巧。

实例：把元素从输入复制到输出

作为实例，请参看这样一种情形，我们要使用输入 XML 消息和从数据库检索的信息创建新的 XML 消息。表 7－1 显示要使用的输入和输出。

表 7－1 创建新的 XML 记录

输入： transactionrecord	**来自数据库的输入**	**输出：logentry**
站=Jupiter		站=Jupiter
金额=1600		金额=1600

输入： transactionrecord	来自数据库的输入	输出：logentry
transdate=20001－11－09 23:00:51.123000		transdate=20001－11－09 23:00:51.123000
	余额=25000	新余额=25000

为此，我们使用以下代码摘录：

```
SET OutputRoot.XML."logentry"."transdate"= InputRoot.XML."transactionrecord"."transdate"；
SET OutputRoot.XML."logentry"."amount"=
InputRoot.XML."transactionrecord"."amount"；
SET OutputRoot.XML."logentry"."station"=
InputRoot.XML."transactionrecord"."station"；
SET OutputRoot.XML."logentry"."newbalance"=
THE（SELECT ITEM T.BALANCE FROM Database.PDK.STATIONSFUNDING AS T WHERE T.NAME
=TRIM（InputRoot.XML."transactionrecord"."station"））；
```

在这里，我们看到九个代码行上的四个 ESQL 代码语句。为增强易读性，语句可以按所需的行数任意换行，但是，每个语句都必须以一个分号结尾。请注意，最后一个 SET 语句分布在三行上，表明语句到分号处才结束。

第一行把输入 “transactionrecord” XML 文档元素的 “transdate” 子元素复制到输出 “logentry” XML 文档元素的 “transdate” 子元素。

在这种分配操作中，引用的元素可能是叶元素（即它们本身不包含子元素），也可能是包含任意数量子元素级别的子结构。无论适用于何种情况，分配语句都会把完整元素从输入位置复制到输出位置，输出位置包含可能存在的任何子元素。后面的两个语句相似。最后一个语句显示您如何使用数据库选择将信息添加到仍在消息流中使用的数据流。

实例：更为复杂的任务

下面请看此实例：

```
SET OutputRoot.XML."RATECALC"."LOG"=THE （SELECT L.MQMD__MSGID，
L.MQMD__CORRELID，L.MQMD__REPLYTOQ，L.MQMD__REPLYTOQMGR，L.MQMD__USERID，
L.MQMD__FORMAT，L.MQMD__ENCODING，L.MQMD__CODEDCHARSETID，L.MQMD__VERSION，
L.MQMD__FEEDBACK，L.MQMD__REPORT，L.MQMD__MSGTYPE
FROM Database.RATECALCMSGLOG AS L
WHERE L.MQMD__MSGID =CAST（InputRoot.MQMD.CorrelId AS CHAR））；
```

在这里引入许多概念。

ESQL 可用来从结构化消息提取数据元素列表，其方法与使用 SQL 查询关系数据库（如 DB2）的方法相似。在上面的实例中，选择查询的目标是复合数据元素“RATECALC”.”LOG”，该元素本身就是 OutputRoot 的 XML 根元素。

我们还可以看看如何使用 THE 谓词。包含此谓词表示查询仅产生一行，省略 THE 谓词表示查询产生不只一行。因此，SET 语句的被分配者可能会具有多个实例。在此实例中，被分配者应带有数组[]后缀，表明由于此操作可能会创建被分配者的一个以上的实例。

提示：在上面的每个实例中，通常的做法是把所有数据元素名称包含在“双引号”之内。这种方法会避免在元素名称中使用特殊字符时可能会出现的任何潜在混乱。

如果在代码中分配文字串值，应在“单引号”内表示这些文字串值。

7.3.3 访问数据库：计算节点和数据库节点

在计算节点中可以包含 ESQL，以便选择、更新或删除数据库数据源中的条目。在消息转换并非必需但却要求数据库操作的情况下，取代使用计算节点的比较好的选择就是使用数据库节点。

MQSI 包含许多适合于各项关键 SQL 操作的专用数据库节点，即 DataInsert、DataUpdate 以及 DataDelete 节点。在使用可通过图形方式操作的消息数据时，这些节点非常有用，除此之外，不会带来任何其他好处。

实例中，我们在所有数据库操作中都使用了一般数据库节点。图 7－3 显示的实例消息流包含一个计算节点（访问数据库表）和一个数据库节点（如果满足“匹配”过滤器节点中的条件，该数据库节点将删除数据库表中的条目）。

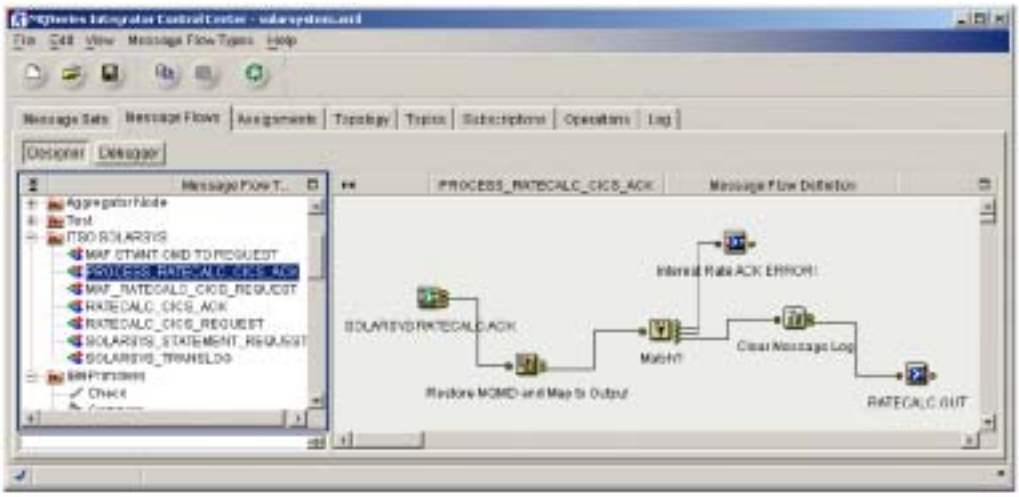

图7－3 组成数据库通路的消息流实例

为了说明计算节点与数据库节点之间的区别，让我们仔细看一下图 7－3 中显示的“ 恢复 MQMD 和映射到输出 ”（Restore MQMD and Map to Output）计算节点及“ 清除消息日志 ”（清除消息日志）数据库节点。

成功获得后，“ SOLARSYS.RATECALC.ACK ”节点的输出节点被传递到“ 恢复 MQMD 和映射到输出 ” 计算节点。这会导致执行此节点的逻辑。图 7－4 显示部分定制逻辑。在第 271 页的“ 计算节点：恢复 MQMD 和映射到输出 ” 中将详细讲述此节点。

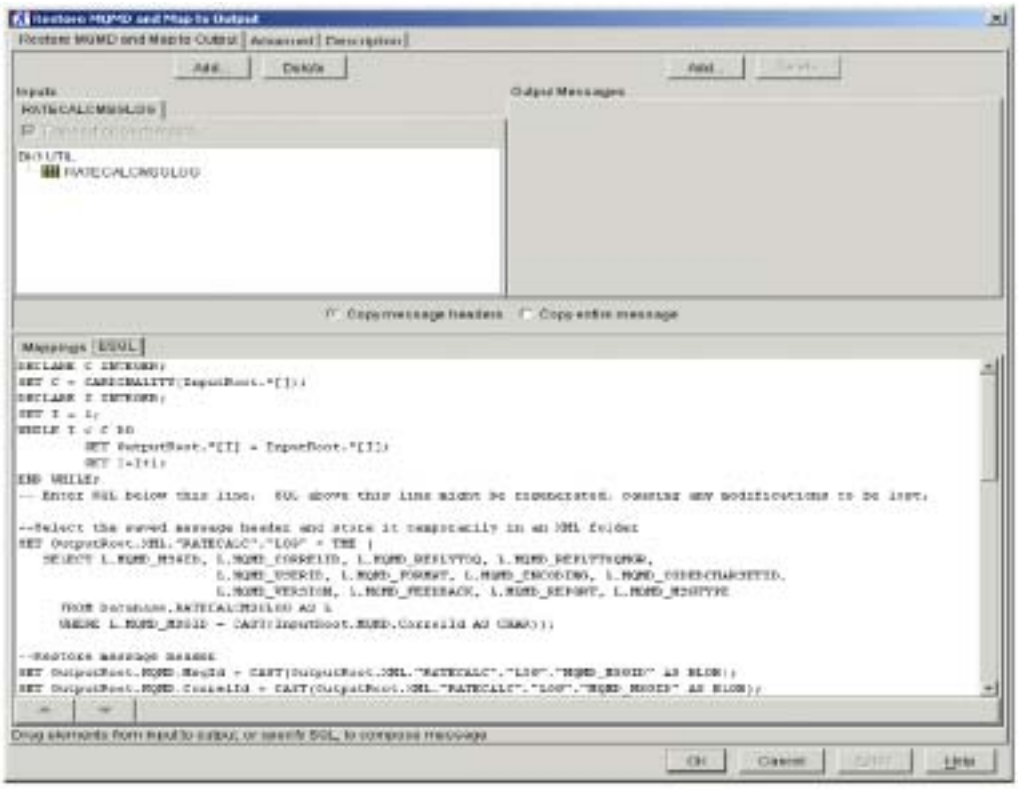

图7－4 恢复 MQMD 和输出计算节点图

注意“ 输入 ”窗格中的数据库。要引用 SELECT 语句中的数据库，这是必需的。SELECT 的结果会临时存储在 “RATECALC”. “ LOG ” XML 文件夹中，然后通过引用此结果来生成 OutputRoot。

计算节点的 OutputRoot 将传递到检查必需条件的过滤器节点。如果条件得到满足（结果为 true），消息就会传递到 “ 清除数据库日志 ”（Clear Database Log）数据库节点（参见图 7－5），并且从数据库表中删除指定的行。

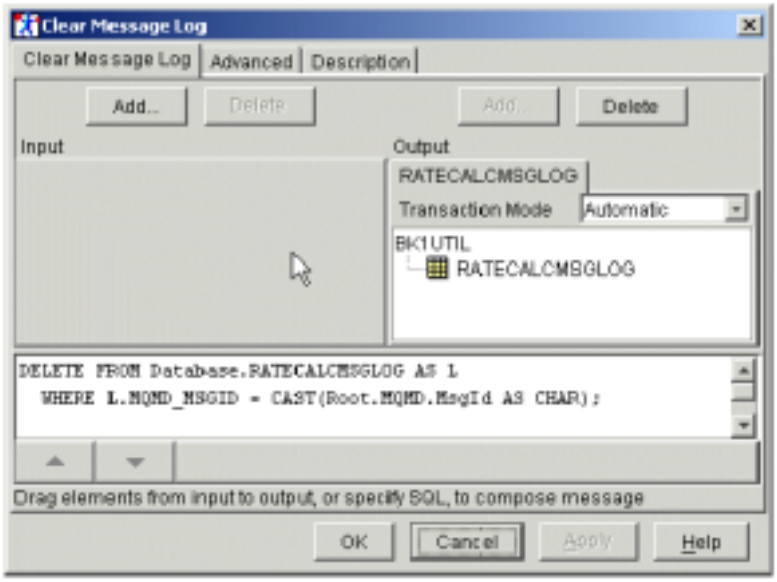

图7－5　清除消息日志

“ 清除消息日志 ”数据库节点只有一个功能，即执行数据库操作。在此案例中，执行的数据库操作为 DELETE FROM。

7.3.4　过滤器节点

在消息流中，根据执行时存在的条件来控制流向非常有用。为此，我们使用过滤器节点。在过滤器节点中，您可以使用 ESQL 建立一个得出“ 真 ”或“ 假 ”值的表达式。根据解析表达式的结果，把控制传送到适用的输出终端。

再看看图 7－3 显示的消息流。您会注意到，对于“ 匹配 ”（Match）过滤器节点，有四个可能的输出终端。过滤器节点上的可能输出为：“ 真 ”、“ 假 ”、“ 未知 ”或“ 失败 ”。

在图 7－6 中，我们可以看到此节点的属性。ESQL 表达式描述了来自作为该节点输入消息的元素。测试该值看是否为“ 不为空 ”（NOT NULL）。如果此条件为真，则把控制分别传送到“ 真 ”输出终端以及其他节点。

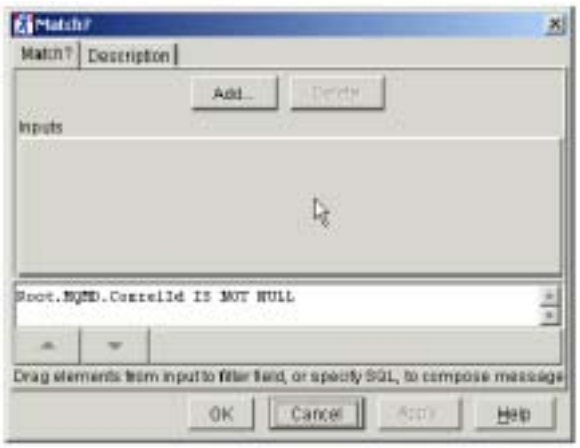

图7－6 匹配过滤器节点属性

您可为过滤器节点添加更多输入以便添加数据库表，该数据表作为表达式中使用的数据源。这样，您就可以使用传入的数据元素从数据库查找值，并可在表达式中测试这些值。

RouteToLabel 节点

根据消息目标列表的内容，您可以使用 RouteToLabel 节点提供动态路由。目标列表保存一个或多个目标标签节点的条目，这些目标标签节点是使用其“ 标签名称 ”（Label Name）的属性识别的，而不是使用节点名称进行识别。

标签

标签节点是 RouteToLabel 节点处理的消息的已命名目标。如上所述，标签节点是 RouteToLabel 节点处理消息时，由消息的目标列表中某个条目识别的。

可在《使用控制中心的 MQSeries Integrator 版本 2.0.2》编号：SC34－5602－03 中，找到有关 RouteToLabel 节点和 Label 节点更为全面的解释和实例。.

7.3.5 MQOutput

MQOutput 节点通常是消息流的行尾，这些消息已进行了一些转换或执行了一些数据消息发送，并已产生新的输出消息。参阅图 7－3，您就可以看到，如果没有发生错误，就把转换的消息放在标有“ RATECALC.ACK.TEST.OUT ”的 MQOutput 节点上。可以在 MQOutput 节点的属性页中指派输出队列，也可以使用“ 目标列表 ”（Destination List）动态指派属性列表。

7.3.6 MQReply

MQReply 节点除了把输出消息放在消息标题 ReplyToQ 指定的队列上之外，其他方面都与 MQOutput 节点相似。这对于需要回应的请求来说非常理想。不能对此节点进行太多配置，因为该节点完全依赖于消息标题 ReplyToQ 和 ReplyToQMgr 属性把消息放到输出队列上。

7.3.7 IBM MQSI Aggregator插件

IBM MQSI Aggregator 插件——SupportPac IA72，将为我们提供应用程序执行分解／重组功能时所需的节点。该插件提供若干工具以获取消息，并将其拆分（分解）为多条发送到多个目标的消息。然后，该插件侦听响应，并将其收集（重组）为一个回复。聚集器插件具有一些执行这些任务的独特节点。在以下的讲述中，我们把该插件所提供的功能集称为“ 聚集器 ”。

IBM MQSI Aggregator 插件是受支持的 SupportPac。有关此节点的最新信息，请访问以下网页：http：／／www－4.ibm.com／software／ts／mqseries／txppacs／ia72.html

第 118 页上的图 7－7 显示 SOLARSYS__STATEMENT__REQUEST 消息流，该消息流使用聚集器接受请求，并把消息拆分成三个不同后端请求。然后，聚集器接收来自后端应用程序的回复，把它们关联为一个回复，并根据原始输入消息 ReplyToQ 和 ReplyToQMgr 属性返回给用户。

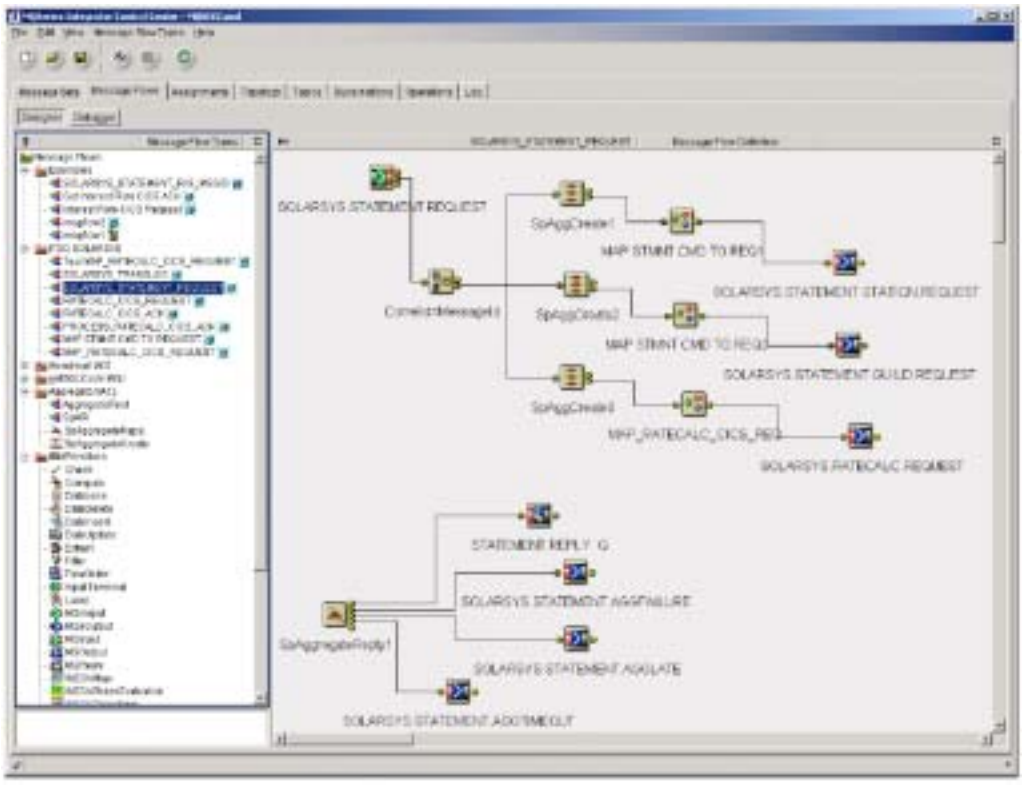

图7－7 是用聚集器节点的S__STATEMENT__REQUEST 消息流

SpAggregateCreate 和 SpAggregateReply 节点共同负责扇出消息和聚集回复。您必须至少有一个 SpAggregateCreate 节点，并至多有一个 SpAggregateReply 节点。两个节点通过其 AggregateName 属性进行关联。

请参阅图 7－7，让我们仔细观察具体工作原理。

在此消息流中，MQInput 节点监控队列 SOLARSYS.STATEMENT.REQUEST。成功获取后，消息发送到计算节点“ CorrelId = MessageId ”。此节点只用来确保存在 MQMD.CorrelId，这是聚集器节点的要求，即您必须具有 MQMD.CorrelId。MQMD.CorrelId 和 MQMD.MessageID *不*必相等。

从“计算”（Compute）节点，请求拆分为三个单独的请求。需要将每个请求传送到 SpAggregateCreate 节点的输入终端。

每个 SpAggregateCreate 节点均有一个输入节点和两个输出节点。输入终端接收要处理的消息。输出节点“输出”和“失败”是两个终端，消息传送到此并分别进行正常处理和错误处理。

图 7－8 显示 SpAggregateCreate 节点的基本属性。除序列号之外，每个接受扇出消息的 SpAggregateCreate 节点的设置都将是相同的。稍后，序列号将由 SpAggregateReply 节点来区分每个消息的响应。AggregateName 用于将 SpAggregateCreate 节点与 SpAggregateReply 节点链接起来。

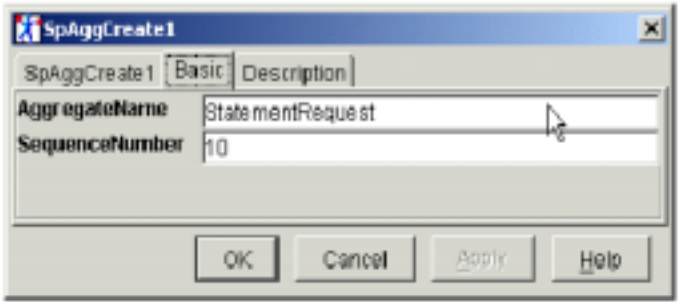

图 7－8 SpAggCreate1 节点

图 7－9 显示 SpAggregateReply 节点的基本属性，该节点将处理各个响应。AggregateName 与 SpAggregateCreate 节点相同，把对请求的回复与 SpAggregateReply 节点链接起来，并将其重组为一个回复，然后返回给请求者。

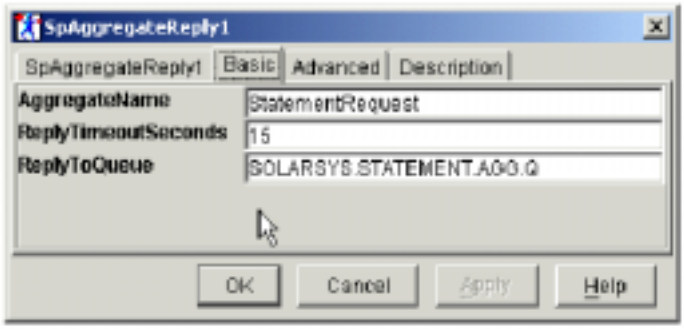

图 7－9 SpAggregateReply1 基本属性

SpAggregateReply 节点具有一个输入终端和五个输出终端。其输出终端为：

- Out
- Failure
- LateReplies
- TimedOut
- Catch

上面这些输出终端名称的意思非常明确。

可以把 SpAggregateCreate 节点的输出终端连接到计算节点，或连接到针对后端请求执行特定转换的子消息流。

尽量不要更改 MQMD.MessageID 和 MQMD.CorrelId 的内容。SpAggregateCreate 节点已经记录了这些属性，而且当回复从各个后端请求返回时，SpAggregateReply 节点都要求原始的 MQMD.MessageID 在 MQMD.CorrelId 之中。这是用于请求／回复消息发送标准的 MQSeries 编程。

另外，SpAggregateCreate 节点已更改了 RepyToQ 的内容，这样回复就返回上面显示的 SpAggregateReply 节点中指定的 ReplyToQueue。

如果由于历史系统的要求而无法使 MQMD 属性保持相同，您可以把消息标题的内容保存到同一数据库。有关实例，请参阅后面第 262 页上的 14.3.3 小节：“ MAP__RATECALC__CICS__REQUEST 子消息流 ”。

图 7－10 显示 SpAggregateReply 节点的“ 高级 ”属性。

图 7－10 SpAggregateReply1 高级属性

在 gregatedReplyDestination 字段中，我们指定回复节点应使用哪种方法把回复消息放到回复队列。使用 ReplyToQueue 方法，即此节点应使用 MQMD ReplyToQ 和 ReplyToQMgr 属性以确定回复消息目标。这些值将会被还原为从原始请求消息记录的原始值，然后用来发送回复消息。MQMD.CorrelId 将通过原始请求消息的 MQMD.MessageID 进行设置，这同样是标准的 MQSeries 编程方法。

我们没有指定 ReplyToQueue，而是指定 DestinationList 或 Default。选择 DestinationList 会导致把原始请求消息的 ReplyToQ 和 ReplyToQMgr 添加到聚集消息 DestinationList 中。然后，可以由将 Destination 模式设置为 DestinationList 的 MQOutput 节点对此进行处理。选择 Default 会导致建立不包含任何特殊细节的聚集消息，并假定开发者将把 Out 终端连接到适当的输出队列。

AggregatedReplyDataType 允许您从 BLOB 或 XML 中进行选择。如果选择了 XML，返回的回复消息需要是有效的 XML 格式。

AggregatedReplyLabelName 是字符串，它将用来构造 XML 回复的标记名称。使用的实际标记名称将是 AggregatedReplyLabelName 与在 SpAggregateCreate 节点中指派的序列号的组合。在我们的实例中，回复消息将会是一个 XML 流，各个回复都分别带有标记名称“ STATEMENT__REPLY10 ”、“ STATEMENT__REPLY20 ”和“ STATEMENT__REPLY30 ”。这样可以非常容易地对 XML 回复进行分析。对于 BLOB 消息类型，您可以使用 SeparatorString 或 HexSeparatorString 来区分要分析的回复。

Persistent 属性是控制聚合实例持久性的 Boolean 值。默认值为 false，即聚合数据将仅存储在内存中，因此，如果代理程序失败，就无法恢复此消息流。设置为 true，聚集器就会在 SpAggregateCreate 和 SpAggregateReply 节点中处理聚合消息时，把恢复消息写到本地队列。如果将此值设置为 true，应确保消息流中的输入和输出消息操作都是事务处理性的，因为会将恢复消息以事务处理方式编写为现有消息流的组成部分。

7.3.8 可复用消息流

MQSI 的一项主要功能是：能够把消息流存储为可复用组件，将这些组件链接在一起，并用作大型操作性消息流（我们称之为“ 可应用 ”消息流）的组成部分。这样，就可以自然条件及明确的输入、操作和输出规范，对完整操作中的各个组件进行开发和测试。例如，有些转换可以要求以 ESQL 形式编写复杂代码。明显优势是只需执行此操作一次，而且重要的是，MQSI 共享配置中只需有给定转换定义的一个实例。 一旦用此方法定义复杂操作的组件，那么将子消息流串接在一起共同编译完整的商业操作就会变得非常简单。

通过将 InputTerminal 和 OutputTerminal 基元用作输入和输出来源以定义消息流，可反复把该消息流用作构成更大消息流组成部分的节点。因此，不能把消息流节点应用到代理程序，因为消息流节点不以独立的方式工作。它们只能用作更大消息流的组成部分。

可以建立通用操作，作为可复用消息流组件，而这些组件可用作更大消息流的节点。图 7－11 显示一个实例，该实例说明如何使“ RATECALC__CICS__REQUEST ”消息流成为可复用的消息流，从而使其逻辑可在更大消息流中重复使用。对于通用数据库更新、查找以及商业代理系统中的其他通用方法来说，这可能是一项强大的功能。

图 7－11 可复用的消息流：MAP__RATECALC__CICS__REQUEST

输入和输出是通过使用终端节点类型 InputTerminal 和 OutputTerminal 定义的。而消息流的其余部分则是用与可应用消息流同样的方式定义的。从该语句，您应推断出不可以孤立应用组件消息流，而只能把它们作为可应用消息流的组成部分。

在保存此消息流时，可以将它拖动到另一个消息流，在那里，它将显示为消息流节点。在第 118 页上的图 7－7 中，您会看到这样的实例。在此实例中，名为‘ MAP__RATECALC__CICS__REQ ”的节点是上面定义过的“ MAP__RATECACL__CICS__REQUEST ”组件消息流的实例。

很明显，使用消息流节点需根据其规范准备其输入。在“ MAP__RATECACL__CICS__REQUEST ”流的情况下，该消息流节点需要具有“ 视图－语句－命令 ”类型的 XML 文档，该文档的内容将匹配已由用户界面组件准备好的文档。

在消息流节点中使用属性升级

将常用操作保存为消息流节点，是一种实现重复利用 MQSI 代码的有用方法。对此功能的一项扩展是，通过属性升级实现消息流节点可配置性的能力。属性升级使消息流用户可为属于消息流级别的节点的已升级属性设置各种值。

例如，图 7－11 显示可复用消息流“ MAP__RATECALC__CICS__REQUEST ”。我们可能需要升级“ 保存消息标题 ”（Save Message Header）节点的“ 数据源 ”（Data Source）属性，因此，在测试消息流的同时，我们可以使 Data Source 属性指向与生产数据库相反的数据库的测试版本。

如果不升级此属性，您就得把消息流内所有单个节点的 Data Source 名称设置为引用不同的 Data Source。然后，当您把此节点应用到生产中时，就必须更改 Data Source 名称的每个引用。

升级 Data Source 属性使您可以将所有单个节点的数据源设置为 PRODDATA，并可在测试时把已升级数据源的值设置为 MYTESTDATA。已升级属性始终优先于相关节点内属性的设置。

有关详细信息，请参阅《使用控制中心的 MQSeries Integrator 版本 2.0.2》编号：SC34－5602－03，其中，详细讲述如何升级消息流节点属性。

7.3.9 测试消息流组件

通过将完整的消息流创建为一组更小、更简单的组件，测试过程成为一项不太艰巨的工作。利用如图 7－12 所示的测试导线，可以单独测试每个子消息流。通过把消息流节点连接成这样一根导线，您可以使用简单的 MQSeries 应用程序把具有所需格式的消息放到输入队列，并可以使用获取消息的简单应用程序监控其结果。

只要设计具有一个以上输或输出终端的消息流组件，就会需要这种方法的一个变体（修改的方法）。

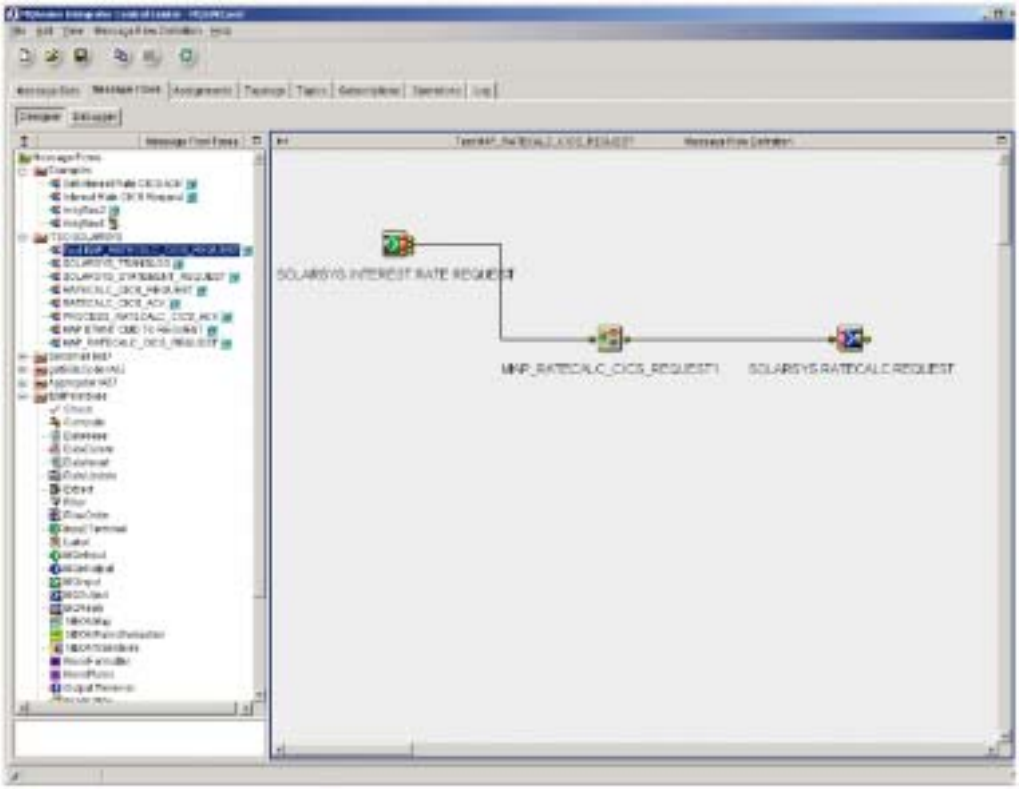

图7－12 测试导线实例

至于测试，我们创建了简单的消息流，通过简单改编提供有 MQSeries 的 MQSample 应用程序，达到了把消息放到队列以及从队列获取消息的目的。 在需要数据库操作的地方，我们使用 DB2 控制中心对数据库内容进行查询。

7.4 使用队列别名

要想在设计中具有灵活性，好的做法是使应用程序代码独立于其基本结构。在应用程序代码内，使用特定队列管理器和队列名称会不可避免地造成将来维护费用的过高。

避免该缺陷的一种方法是，在用于队列管理器以及与其进行交互的队列的应用程序代码中使用别名，把解释别名的工作留给支持性基础结构 MQSeries。可能还有其他更可取的方法。例如：使用属性文件、ini 文件或某个数据库中存储的信息。 要使用数据库信息，请在属性文件中存储刚好够用的信息连接数据库，然后在数据库表中查找队列排队连接参数。此技术可用于针对您所连接的环境实例（开发、测试、质量保证或生产）的许多应用程序变量。

当把 JMS 用作 API 时，一个选项是使用 JMS 配置详细信息来处理应用程序对象名称和真实对象引用之间的转换。在已使用 MQSeries 类（用于 Java）的地方，必须使用一组使消息发送功能能够指向适用队列的适用队列和队列管理器别名，来委托队列管理器。您可以针对各种环境使用不同的 JMS 配置存储库，这样，对象名称就可始终保持相同。队列排队定义下面的对象名称具体取决于您要与 JMS 名称服务器环境中的哪个实例进行交互。

另一个 API——应用程序消息发送接口（AMI）——是基于策略的编程接口。利用 AMI，可根据某种策略将消息发送到服务，而 AMI 将把消息转换为队列名称、队列管理器名称以及其他 MQSeries 选项。这样，就没有必要使程序员了解您的 MQSeries 网络的实施细节。