SYBASE®

设计指南

Replication Server®

12.6

文档 ID：DC31036-01-1260-01
最后修订日期：2003 年 10 月

本手册适用于 Sybase 软件及所有后续版本，除非在新版本或技术注释中另有说明。本手册中的信息如有更改，恕不另行通知。本手册中所介绍的软件按许可协议提供，其使用和复制必须符合协议条款。

若要订购其他文档，美国和加拿大客户请拨打客户服务部门电话 (800) 685-8225，或发传真至 (617) 229-9845。

持有美国许可协议的其他国家 / 地区的客户可通过上述传真号码与客户服务部门联系。所有其他国际客户请与 Sybase 子公司或当地的分销商联系。我们只在软件的定期发布日提供升级文档。未经 Sybase, Inc. 事先书面授权，不得以任何形式或通过任何手段（电子的、机械的、手工的、光学的或其他手段）复制、传播或翻译本手册的任何部分。

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目录

关于本手册

Replication Server® 在网络中的多个站点上维护复制的数据。如果企业的各个站点在地域上相距甚远，可以使用 Replication Server 来创建分布式数据库应用程序，其性能和数据可用性都优于集中式数据库系统。

本手册介绍基于复制技术的分布式数据库系统，帮助您设计复制系统。

读者

《Replication Server 设计指南》面向所有使用 Replication Server 的用户。如果您对 Replication Server 感到陌生，可以先阅读本手册中对 Replication Server 以及对使用复制的数据的应用程序所作的介绍。

如果您要为 Replication Server 设计新的应用程序，应先阅读本手册，然后再安装 Replication Server 软件。请使用本手册中提供的信息来规划复制系统，以便弄清楚构成复制系统的软件组件的安装位置。

如何使用本手册

本手册中的信息分为以下几部分：

- 第 1 章 “简介”介绍 Replication Server 及其功能。
- 第 2 章 “复制系统的应用程序体系结构”讨论复制系统的设计问题。
- 第 3 章 “实现策略”介绍一些可供您用来实现您设计的复制系统的模型。
- 第 4 章 “规划备份和恢复”介绍一些复制系统故障预防措施和排除措施。
- 第 5 章 “Replication Agent 介绍”介绍 Sybase Replication Agent 产品，这些产品可以用来从非 Sybase® Adaptive Server 或 Sybase SQL Server 数据库复制数据。
- 第 6 章 “将数据复制到异构数据服务器”介绍将数据复制到非 Adaptive Server 或 Sybase SQL Server 数据服务器时所需的复制系统组件。
- 第 7 章 “国际化复制设计注意事项”介绍如何为国际环境配置语言、字符集和排序顺序。

- 附录 A“容量规划”介绍如何估算 Replication Server 分区需要的磁盘空间量。
- 附录 B“日志传送语言”介绍 Replication Agent 将事务操作和存储过程调用数据发送到 Replication Server 时使用的日志传送语言 (LTL)。

相关文档

Sybase® Replication Server® 文档集包含以下文档：

- 针对您所用平台的发行公告 — 包含一些没来得及添加到本手册中的最新信息。

  万维网上可能有发行公告的较新版本。若要查看在产品 CD 发布后新增的重要产品或文档信息，请使用 Sybase Technical Library。
- 针对您所用平台的《安装指南》— 介绍所有 Replication Server 产品和相关产品的安装和升级步骤。
- 《Replication Server 新增功能》— 介绍 Replication Server 版本 12.6 中的新增功能和为支持这些功能而实施的系统更改。
- 《管理指南》— 包含对复制系统的介绍。这本手册中包含一些有关如何创建和管理复制系统、如何设置安全性、如何从系统故障中恢复和如何提高性能的信息和准则。
- 针对您所用平台的《配置指南》— 介绍所有 Replication Server 产品和相关产品的配置过程，并介绍如何使用 rs_init 配置实用程序。
- 《设计指南》— 包含有关如何设计复制系统和如何将异构数据服务器集成到复制系统中的信息。
- 《Replication Server 入门》— 分步介绍如何安装和设置一个简单复制系统。
- 《异构复制指南》— 介绍如何使用 Replication Server 在由不同供应商提供的数据库之间复制数据。
- 《参考手册》— 包含以下内容：复制命令语言 (RCL) 中的 Replication Server 命令的语法和详细说明； Replication Server 系统函数；与 Replication Server 一起使用的 Sybase Adaptive Server® 命令、系统过程和存储过程； Replication Server 可执行程序；Replication Server 系统表。
- 系统表框图 — 以海报格式说明系统表和它们的实体关系。只提供印刷版本。
- 《故障排除指南》— 包含帮助用户诊断和解决复制系统中的问题的信息。
- Replication Server 插件帮助，其中包含有关如何使用 Sybase Central™ 来管理 Replication Server 的信息。

其他信息来源

可以使用 Sybase Getting Started CD、 Sybase Technical Library CD 和 Technical Library Product Manuals Web 站点来了解有关产品的更多信息：

- Getting Started CD 包含 PDF 格式的发行公告和安装指南，而且可能还包含 Technical Library CD 上没有的其他文档或更新信息。Getting Started CD 随软件附送。要阅读或打印 Getting Started CD 上的文档，您需要 Adobe Acrobat Reader （可从 Adobe Web 站点免费下载，该 CD 上提供了该站点的链接）。
- Technical Library CD 包含产品手册和技术文档，并随软件附送。使用 DynaText 浏览器 （随 Technical Library CD 附送）可以使用一种易用的格式来访问有关产品的技术信息。

  有关安装和启动 Technical Library 的说明，请参阅文档包中的《Technical Library 安装指南》。
- Technical Library Product Manuals Web 站点是 Technical Library CD 的 HTML 版本，您可以使用标准 Web 浏览器来访问该站点。除产品手册外，该站点还提供了 “EBF/Update”、“Technical Document”、“Case Management”、“Solved Case”、新闻组和 Sybase Developer Network 的链接。

  要访问 Technical Library Product Manuals Web 站点，请转到位于 http://www.sybase.com/support/manuals/ 上的 Product Manuals （产品手册）。

Web 上的 Sybase 认证

Sybase Web 站点上的技术文档经常更新。

❖ **查找有关产品认证的最新信息**

1. 将 Web 浏览器指向位于 http://www.sybase.com/support/techdocs/ 上的 Technical Documents （技术文档）。
2. 从左侧的导航栏中选择 “Products”。
3. 从产品列表中选择一个产品名称，然后单击 “Go”。
4. 选择 “Certification Report”过滤器，指定时间范围，然后单击 “Go”。
5. 单击一个 “Certification Report”标题将显示该报告。

❖ **创建 Sybase Web 站点的个性化视图 （包括支持页）**

建立 MySybase 配置文件。 MySybase 是一项免费服务，可用于创建 Sybase Web 页的个性化视图。

1. 将 Web 浏览器指向位于 http://www.sybase.com/support/techdocs/ 上的 Technical Documents （技术文档）。
2. 单击 “MySybase”并创建一个 MySybase 配置文件。

Sybase EBF 和软件更新

❖ **查找有关 EBF 和更新的最新信息**

1 将 Web 浏览器指向位于 http://www.sybase.com/support 上的 Sybase 支持页。

2 选择 “EBFs/Updates”。如果出现提示，请输入用户名和口令信息（对于现有 Web 帐户），或创建新的帐户 （免费服务）。

3 选择产品。

4 指定时间范围，然后单击 “Go”。

5 单击 “Info”图标可显示 “EBF/Update”报告；单击产品说明可下载该软件。

约定

本节说明本手册所使用的风格和语法约定、 RCL 命令格式约定和图标。

风格约定

显示命令语法和选项的语法语句显示如下：

alter user *user*
set password *new_passwd*
[verify password *old_passwd*]

有关详细信息，请参见第 xiii 页的 “语法约定”。

说明 Replication Server 命令用法的示例显示如下：

```
alter user louise
 set password somNIfic
 verify password EnnuI
```

命令名、命令选项名、程序名、程序标志、关键字、函数和存储过程显示如下：

使用 alter user 可更改登录名的口令。

变量、参数和用户提供的内容在语法和段落文本中用斜体显示，如下所示：

set password *new_passwd* 子句指定新口令。

数据库对象 （如数据库、表、列和数据类型）的名称在段落文本中用斜体显示，如下所示：

Items 表中的 base_price 列的数据类型是 money。

复制对象 （如函数字符串类、错误类、复制定义和预订）的名称用斜体显示。

语法约定

下表总结了各种语法格式约定。后面给出了包含这些元素的示例。

表 1：语法格式约定

键	定义
{ }	大括号表示至少必须选择括号中的一个选项。命令中不要包括大括号。
[ ]	方括号表示可以选择也可以不选择括号内的选项。命令中不要包括方括号。
|	竖线表示最多可以选择一个选项 （括在大括号或方括号内的选项）。
,	逗号表示可以选择任意多个所需选项 （括在大括号或方括号内的选项）。在命令中应键入逗号来分隔选项。 其他语法环境中也可能需要逗号。
( )	小括号应作为命令的一部分输入。
...	省略号 （三个点）表示可以根据需要将最后一部分重复使用任意次。命令中不要包括省略号。

必选的选项

- 大括号和竖线 — 只选择一个选项。

```
{red | yellow | blue}
```

- 大括号和逗号 — 选择一个或多个选项。如果选择多个选项，请使用逗号分隔选项。

```
{cash, check, credit}
```

可选的选项

- 方括号中只有一项 — 可以选择该选项，也可以不选择该选项。

  ```
  [anchovies]
  ```

- 方括号和竖线 — 不选择或只选择一项。

  ```
  [beans | rice | sweet_potatoes]
  ```

- 方括号和逗号 — 不选择、选择一个或选择多个选项。如果选择多个选项，请使用逗号分隔选项。

  ```
  [extra_cheese, avocados, sour_cream]
  ```

重复元素

省略号 (...) 表示可以根据需要将最后一部分重复任意多次。例如，在使用 alter replication definition 命令时，可在 add 子句或 add searchable columns 子句中列出一列或多列及其数据类型：

alter replication definition *replication_definition*
{add *column datatype* [, *column datatype*]... |
add searchable columns *column* [, *column*]... |
replicate {minimal | all} columns}

RCL 命令的格式

RCL 命令与 Transact-SQL® 命令相似。以下各节说明各种格式规则。

命令格式和批处理命令

- 除了关键字或标识符中间，可以在任何地方断行。可以通过在行尾输入反斜杠 (\)，使字符串在下一行继续。
- 除了在反斜杠之后，额外的空格将被忽略。请不要在反斜杠之后输入任何空格。
- 如果没有其他说明，可在一个批处理中输入多个命令。
- RCL 命令不是事务型的。各个命令都独立执行，并且不受批处理中其他命令完成状态的影响。然而，命令中的语法错误会导致 Replication Server 无法执行同一批处理中后面的命令。

大小写的区分

- RCL 命令中的关键字不区分大小写。您可以用大写或小写字母的任意组合形式输入这些关键字。
- 标识符和字符数据是否区分大小写取决于所使用的排序顺序。
  - 如果使用区分大小写的排序顺序，例如 “binary”，则必须用正确的大写和小写字母组合形式输入标识符和字符数据。
  - 如果使用不区分大小写的排序顺序，例如 “nocase”，则可以用任意大写或小写字母组合形式输入标识符和字符数据。

标识符

标识符就是为数据库对象或复制对象指定的名称，包括对象名、列名、变量名和参数名。

- 标识符的长度可以为 1— 30 个字节（相当于 1— 30 个单字节字符）而且必须以字母、 @ 符号或字符 _ 开头。
- Replication Server 函数参数是唯一能够以字符 @ 开头的标识符。在函数参数名中，字符 @ *之后* 可以有 30 个字符。
- 第一个字符后，标识符中可以包括字母、数字和 #、$ 或 _ 字符。不允许包括空格。

函数字符串中的参数

除以下情况之外，函数字符串中参数的规则与标识符的规则相同：

- 参数括在问号 (?) 中。这样， Replication Server 就可以在函数字符串中找出参数。使用两个连续问号 (??) 可在函数字符串中表示一个实际问号。
- 感叹号 (!) 后面是参数修饰符，它指示要在运行时替代参数的数据的来源。有关修饰符的完整列表，请参见 《Replication Server 管理指南》。

数据类型支持

Replication Server 支持所有 Adaptive Server 数据类型。

用户定义的数据类型不受支持。 timestamp、 double precision、 nchar 和 nvarchar 数据类型间接受支持；它们被映射到其他数据类型。使用 timestamp 数据类型的列映射到 varbinary(8)。

有关所支持的数据类型的详细信息，包括如何设置其格式，请参见 《Replication Server 参考手册》。

图标

本手册中的图示使用图标来表示复制系统的各种组件。

Replication Server

此图标代表 Sybase 服务器程序 Replication Server，它在局域网 (LAN) 上维护复制数据，并处理从广域网 (WAN) 上的其他 Replication Server 接收到的数据事务。

Adaptive Server 或数据服务器

此图标代表 Sybase 数据服务器 Adaptive Server。数据服务器管理包含主数据或复制数据的数据库。 Replication Server 也可以与 SQL Server 和异构数据服务器一起工作，因此，如果没有其他说明，此图标可以代表复制系统中的任何数据服务器。

Replication Agent

此图标代表 Replication Agent，一个将主数据库的事务日志信息传送到 Replication Server 的复制系统进程或模块。用于 Adaptive Server 的 Replication Agent 是 RepAgent ； 用于 Sybase SQL Server 的 Replication Agent 是日志传输管理进程 (LTM)。 Sybase 为 Adaptive Server Anywhere、 DB2、 Informix、 Microsoft SQL Server 和 Oracle 数据服务器提供了 Replication Agent 产品。

除了 RepAgent 是一个 Adaptive Server 线程外，所有 Replication Agent 都是独立的进程。通常，只有表示 Replication Agent 是独立进程时，此图标才会出现。

客户端应用程序

此图标代表客户端应用程序。客户端应用程序是指连接到数据服务器的用户进程或应用程序。它既可以是由用户执行的前端应用程序，也可以是作为系统扩展执行的程序。

Replication Server Manager

此图标代表 Sybase Central 的 Replication Server Manager (RSM)，一个复制系统管理员用于监控、分析、排除故障和管理复制系统的软件工具。

RSM 由以下组件组成：

- *RSM Client*（Sybase Central + Replication Server 插件）：一个 Open Client 应用程序，可用于监控、诊断和管理复制系统。 RSM Client 具有一个与 Sybase Central 集成的图形化用户界面 (GUI)。
- *RSM Server*：一个 Open Server 应用程序，它转换并传递来自 RSM Client 的请求。 RSM Server 还监控和响应复制系统事件。

如果需要帮助

对于购买了支持合同的客户安装的每一个 Sybase 产品，都会有一位或多位指定人员获得与 Sybase 技术支持部门联系的授权。如果使用手册或联机帮助不能解决问题，可以让指定人员与 Sybase 技术支持部门或与所在区域的 Sybase 子公司联系。

第 1 章

简介

本章介绍 Replication Server 并介绍如何使用它创建和维护分布式数据应用程序。

集中式和分布式数据库系统

在传统企业级计算模型中，由信息系统部门控制集中式企业数据库系统。通常由位于企业总部的主机提供所需的性能级别。远程节点使用信息系统部门提供的应用程序，通过广域网 (WAN) 访问企业的数据库。

企业环境正在向非集中式的运作方向变化，这种变化也促使企业向分布式数据库系统的方向发展，这种数据库系统更能满足新的非集中式企业的需要。

当今的全球化企业会拥有许多通过 WAN 连接的局域网 (LAN)，以及局域网上的其他数据服务器和应用程序。各节点的客户端应用程序需要在本地通过局域网访问数据，或通过广域网远程访问数据。例如，东京的客户端可能要在本地访问位于东京的数据服务器上存储的表，或远程访问位于纽约的数据服务器上存储的表。

在分布式数据库环境中，企业总部或地区总部可能需要通过大型主机维护敏感的企业数据，同时，远程节点的客户端使用小型计算机和服务器级工作站进行本地处理。

集中式和分布式数据库系统都必须应对与远程访问相关的问题：

- 当 WAN 通讯量繁重时，网络响应会变慢。例如，当决策支持应用程序请求大量的行时，关键事务处理应用程序可能会受到负面影响。
- 如果较大的用户群体争夺对数据服务器的访问，集中式数据服务器便会成为瓶颈。
- 网络出现故障时，数据将不可用。

复制数据的优点

与远程数据库访问相关的性能和可用性问题可以通过将数据从其源数据库复制到本地数据库得到解决。 Replication Server 提供了成本高效的容错系统用于复制数据。

Replication Server 在多个数据库中保持数据是最新的，这样客户端便可以访问本地数据，而不必访问远程的集中式数据库。与集中式数据系统相比，复制系统提供了更好的系统性能和数据可用性，并且降低了通信开销。

由于 Replication Server 传输的是事务而不是数据行，所以它可以在提高数据可用性的同时，在整个系统中保持复制的数据的完整性。
Replication Server 还允许复制存储过程调用，从而进一步提高了性能。

提高的性能

在分布式复制系统中，数据请求在本地数据服务器上完成，而不需要客户端访问 WAN。本地客户端性能的提高是由于：

- LAN 数据传输率比 WAN 数据传输率快。
- 本地客户端共享本地数据服务器资源，而不必争用中央数据服务器的资源。
- 通讯量和锁争用明显减少，因为本地决策支持应用程序已经与集中式 OLTP 应用程序分离。

更好的数据可用性

在分布式复制系统中，数据在本地节点和远程节点进行复制。这样，无论主数据源或 WAN 上出现什么情况，客户端都可以继续工作。

- 当远程节点出现故障时，客户端可以继续使用复制的数据的本地副本。
- 当 WAN 出现故障时，客户端可以继续使用本地复制的数据。
- 当本地数据服务器出现故障时，客户端可以切换到其他节点上复制的数据。

当 WAN 通信出现故障时，其他节点的 Replication Servers 将事务存储在稳定队列 （磁盘存储）中。这样，不可用节点上复制的表将在通信恢复后得到更新。当在源数据库中初始化复制的函数时，在该函数可以被传递到目标节点之前，它将被存储在稳定队列中。

使用 Replication Server 分发数据

Replication Server 通过以下方式在网络上分发数据：

- 为应用程序开发人员和系统管理员提供灵活的 “发布 - 预订”模型，用于标记要复制的数据和存储过程
- 在整个网络中保持事务完整性的同时管理复制的事务

因为 Replication Server 复制事务 （增量变化而不是数据复制）和存储过程调用 （而不是存储过程本身），所以可以在保持数据完整性的同时提供高性能的分布式数据环境。

“发布 - 预订”模型

在正常工作的 Replication Server 系统中， Replication Agent 将检测源数据库中发生的事务并将其传输到本地 Replication Server，本地 Replication Server 将信息通过 LAN 和 WAN 分发到目标节点的 Replication Server。随后，目标节点的 Replication Server 再根据远程客户端的要求更新目标数据库。如果某个网络或系统组件出现故障，正在传递的数据将被临时存储在队列中。当出现故障的组件恢复运行后，复制系统将重新同步数据的副本并恢复正常的复制。

主数据是 Replication Server 在其他数据库中复制数据的来源。您在主节点“发布”数据，其他节点的 Replication Server “预订”该数据。首先创建**复制定义**，指定主数据的位置。复制定义描述表的结构，并命名包含表的主副本的数据库。为便于管理，可以将复制定义收集到**发布**中。

创建复制定义或发布操作本身并不会使 Replication Server 复制数据。您必须根据复制定义（或发布）创建预订，指示 Replication Server 在另一个数据库中复制数据。预订类似于 SQL select 语句。它可以包含 where 子句以指定要复制本地数据库中表的哪些行，这样便可以只复制所需的数据。

从 Replication Server 11.5 版开始，主表可以有多个复制定义。复制表可以预订不同的复制定义以获得数据的不同视图。

创建了复制定义或发布的预订之后， Replication Server 将事务复制到预订数据的数据库。

复制函数

Replication Server 可用于在数据库之间异步复制 Adaptive Server 存储过程调用。该方法通过在一个**复制函数**中封装许多更改来提高普通数据复制的性能。因为复制函数与表复制定义不关联，所以可以执行那些会或不会直接修改数据的存储过程。

可以将存储过程调用从主数据库复制到复制数据库，或从复制数据库复制到主数据库。有关详细信息，请参见第 35 页的 “使用应用函数”和第 63 页的 “请求函数”。

使用复制函数，您可以在另一个数据库中执行存储过程。复制函数用于：

- 将 Adaptive Server 存储过程的执行复制到预订节点
- 通过只复制存储过程的名称和参数而不是实际更改来提高性能

和表一样，复制的存储过程可以具有复制定义（称为*函数复制定义*）和预订。当执行复制的存储过程时， Replication Server 将其名称和执行参数传递给预订节点，在预订节点执行相应的存储过程。

在主数据节点创建函数复制定义。 Replication Server 支持应用函数和请求函数：

- *应用函数* 从主数据库复制到复制数据库。可以在复制节点为函数复制定义创建预订，并在主数据库中标记要复制的存储过程。
- *请求函数* 从复制数据库复制到主数据库。请求函数没有预订。可以在复制数据库中标记要复制的存储过程。

事务管理

Replication Server 依靠数据服务器提供保护存储数据所需的事务处理服务。为保证分布式数据的完整性，数据服务器必须遵从基本单元和一致性等事务处理约定。

存储主数据的数据服务器提供分布式数据库系统所需的大部分并发控制。如果事务未能用主数据更新表， Replication Server 不会将事务分发到其他节点。当事务更新主数据时， Replication Server 分发更改，除非出现故障，否则会在所有预订数据的节点上完成更新。

Replication Server 使用开放式并发控制保持复制的数据的一致性。这种方法与保守式分布并发控制方法 （例如两阶段提交）不同，因为前者在出现故障后对故障进行处理。

在复制系统中开放式并发控制具有以下优点：

- 提升数据的高可用性，因为该方法在分布式事务过程中不锁定数据。
- 处理事务所需要的系统资源较少。
- 不需要数据服务器具备特殊的分布式事务处理功能即可参与分布式事务。

失败的复制事务

如果修改了主数据，则在另一个节点更新该数据的复制副本可能失败。主版本是 “正式的”副本，在主数据库成功的更新操作在包含复制副本的节点也应该成功。

更新复制的表失败的原因如下：

- 数据服务器的维护用户登录名没有更新复制数据所需的权限。
- 数据的复制版本和主版本在系统恢复之后不一致。
- 客户端直接更新复制数据，而不是更新主版本。
- 存储复制表的数据服务器有存储主版本的数据服务器未强制实施的约束。
- 由于系统故障 （例如数据库空间不足），存储表复制副本的数据服务器拒绝事务。

如果事务失败， Replication Server 将从数据服务器接收到一个错误。数据服务器错误映射到 Replication Server 错误操作。对失败事务的缺省操作是在 Replication Server 错误日志中写入一条消息 （包括由数据服务器返回的消息），然后挂起数据库连接。在纠正导致失败的原因之后，可以恢复数据库连接， Replication Server 将重试失败的事务。

也可以让 Replication Server 在例外日志 （RSSD 中的三个表）中记录失败的事务，然后继续处理下一个事务。有关 RSSD 的说明，请参见第 7 页的 “Replication Server”。

如果使用例外日志，您必须手工解决日志中保存的事务，使复制数据和主数据一致。在某些情况下，可以将处理过程自动化，方法是将处理被拒绝事务的逻辑封装在智能应用程序中。

在多个数据服务器和数据库中修改数据的事务

在多个数据服务器中修改主数据的事务可能需要附加的并发控制。根据事务处理的要求，或者执行事务内的所有操作，或者不执行任何操作。如果某个事务在一个数据服务器上失败，则必须在该事务中更新的所有其他数据服务器上回退它。

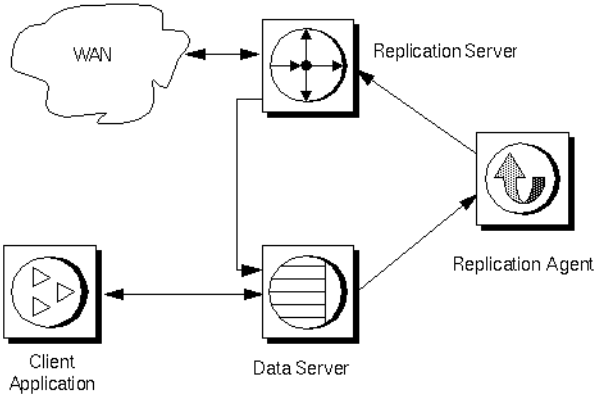
通常，每个主数据库只有一个 Replication Agent。如果单个事务更新多个主数据库，则该事务将作为多个独立的事务进行复制，每个主数据库一个事务。或者，可以选择在单个存储过程中封装此类事务，然后该存储过程作为一个基本单元流动到预订节点。

复制系统组件

Replication Server 具有开放式体系结构，允许从现有的系统和应用程序构造复制系统，并随着企业的发展变化增加复制系统。

图 1-1 图示了使用 Replication Server 的基于 WAN 的分布式数据库系统中的一个节点。以下的章节说明了各个组件。

图 1-1：复制系统



Replication Server

每个节点的 Replication Server 协调本地数据服务器的数据复制活动，并与其他节点的 Replication Server 交换数据。

Replication Server：

- 通过 Replication Agent 从数据库接收主数据事务，然后将它们分发到预订数据的节点
- 从其他 Replication Server 接收事务，然后将事务应用于本地数据库

Replication Server 系统表存储完成这些任务所需的信息。系统表包括对复制定义和预订等复制数据和复制对象的说明、 Replication Server 用户的安全记录、其他节点的路由信息、本地数据库的访问方法及其他管理信息。

Replication Server 系统表存储在名为 Replication Server 系统数据库 (RSSD) 的 Adaptive Server Enterprise 数据库中或名为 ERSSD （嵌入式 RSSD）的 Adaptive Server Anywhere 数据库中。每个 Replication Server 分配一个 RSSD 或 ERSSD。带 RSSD 的 Adaptive Server Enterprise 数据服务器还可以存储应用程序数据库。

有关 ERSSD 的详细信息，请参见 《Replication Server 管理指南》。

可以使用复制命令语言 (RCL) 或 Sybase Central 的 Replication Server Manager 组件管理 Replication Server 中的信息。 RCL 命令类似于 SQL 命令，可以使用 isql （Sybase 交互式 SQL 实用程序）在 Replication Server 上执行。《Replication Server 参考手册》是 RCL 的完整参考资料。有关 Replication Server Manager 的信息，请参见 《Replication Server 管理指南》和 Replication Server 的联机帮助。

分区和稳定队列

Replication Server 将消息存储在磁盘上，以确保可以在故障后传递这些消息。当安装 Replication Server 时，可以分配一个初始*磁盘分区*，Replication Server 使用该分区作为其磁盘存储。完成 Replication Server 的安装后可以再添加其他分区。

分区可以是原始磁盘设备或操作系统文件。因为 UNIX 操作系统对文件 I/O 进行缓冲，所以故障后可能无法完全恢复数据。在这样的系统上，仅在测试环境中将操作系统文件用作分区。在生产环境中请使用原始磁盘分区。有关添加分区的详细信息，请参见《Replication Server 参考手册》。

Replication Server 从磁盘分区中为其服务的路由和连接分配*稳定队列*。消息在确认被目标接收之前一直保存在稳定队列中。

应该为 Replication Server 分区分配的磁盘空间量取决于应用程序的事务大小和事务率。稳定队列在数据流经复制系统时充当数据缓冲。如果远程节点的 Replication Server 在网络发生故障时无法访问，主 Replication Server 在通信恢复之前将事务存储在稳定队列中。为磁盘分区分配的空间越多， Replication Server 在不中断主数据库操作的情况下在队列中保留数据的时间就越长。

附录 A“容量规划”详细解释了如何计算 Replication Server 的分区空间。

数据服务器

数据服务器管理包含主数据或复制的数据的数据库。客户端使用数据库存储和检索数据，并处理查询和事务。 Replication Server 通过以数据库用户身份登录维护数据服务器中的复制数据。

Replication Server 通过开放接口支持异构数据服务器。只要支持一组必要的数据操作和事务处理指令，任何存储数据的系统都可用作数据服务器。

有关数据服务器要求的详细信息，请参见第 13 页的 “异构数据服务器支持”。

Replication Agent

Replication Agent 将事务日志信息 （代表对主数据进行的更改）从数据服务器传送到 Replication Server，以分发到其他数据库中。 Adaptive Server 的 Replication Agent 是一个 Adaptive Server 线程 RepAgent。而 Sybase SQL Server 的 Replication Agent 是名为 LTM (Log Transfer Manager) 的一个独立进程。

Replication Agent 读取数据库事务日志，并将复制表和复制存储过程的日志记录传送给管理数据库的 Replication Server。 Replication Server 重新构造事务，并将其转发给预订了数据的节点。

如果数据库包含主数据或执行复制存储过程，则需要 Replication Agent。如果数据库仅包含复制数据的副本，而没有复制存储过程，则不需要 Replication Agent。

因为 RepAgent 是一个 Adaptive Server 线程，所以本手册的大多数系统框图都不包含 Replication Agent 图标。但在第 5 章 “Replication Agent 介绍”中介绍的 Replication Agent 是一个独立的进程，为区别起见，系统框图中包含 Replication Agent 图标。

客户端应用程序

客户端应用程序是用于访问数据服务器的用户程序。当数据服务器是 Adaptive Server 时，客户端应用程序可以是使用 Open Client Client Server、 Embedded SQL、 PowerBuilder® 或任何其他与 Sybase 客户端 / 服务器接口 (C/SI) 兼容的前端开发工具创建的任何程序。

客户端应用程序应该只更新主数据。 Replication Server 将更改分发到其他节点。不修改数据的客户端应用程序不需要区别主数据和复制数据。

连接复制系统组件

Replication Server、 Replication Agent 和 Adaptive Server 使用 Sybase 客户端 / 服务器接口通过网络进行通信。另外，Replication Server 还使用路由和连接向其他 Replication Servers 和数据库发送消息。以下各节将对接口文件、路由和连接进行介绍。

接口文件

数据服务器、 Replication Server 和 Replication Agent 等服务器程序均在接口文件或轻量目录访问协议 (LDAP) 服务器中注册，以便客户端应用程序和其他服务器程序可以定位它们。

通常，每个节点的一个接口文件包含所有本地和远程 Replication Servers 和数据服务器的条目。每个服务器的对应条目包括其唯一名称以及其他服务器和客户端程序连接到该服务器所需的网络信息。

使用文本编辑器维护接口文件。有关 LDAP 服务器的信息，请参见《Replication Server 管理指南》。

注释 如果使用的是基于网络的安全性机制 （Replication Server 12.0 版和更高版本提供），请使用网络安全性机制的目录服务（而不是接口文件）注册 Replication Server、Adaptive Server 和网关软件。有关详细信息，请参见基于网络的安全性机制附带的文档。

路由和连接

路由和连接可使 Replication Server 互相之间发送消息，以及向数据库发送命令。*路由* 是一个 Replication Server 向另一个 Replication Server 发送请求的单向消息流。*连接* 是从 Replication Server 到数据库的消息流。在热备份应用程序中， Replication Server 使用*逻辑连接* 表示活动和备用的数据库。

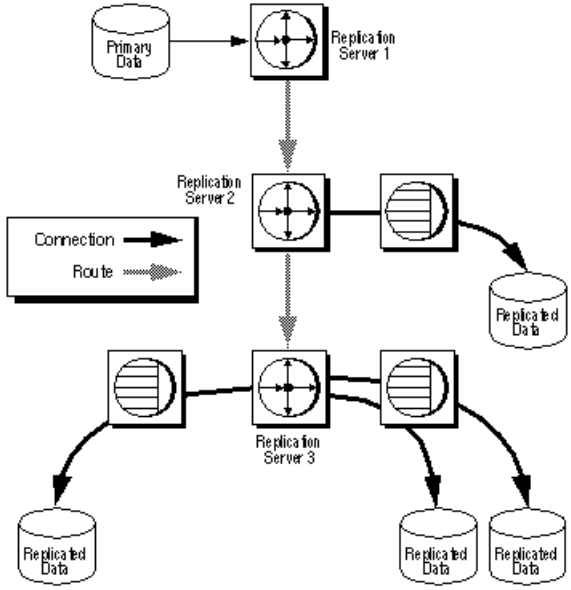
若要将数据从一个数据库复制到另一个数据库，必须首先建立路由和连接，使 Replication Server 能够将数据从其源数据库移动到目标数据库。

向复制系统中添加数据库时， Sybase Central 或 rs_init 会为您创建连接。除非要将数据复制到非 Adaptive Server 或 SQL Server 数据库，否则不需要直接创建连接。

如果复制系统中有多个 Replication Server，则必须在它们之间创建路由。如果只有一个 Replication Server，则不需要创建路由。

图 1-2 图示了三个 Replication Server、一个存储主数据的数据库和四个存储复制数据的数据库之间的连接和路由。

图 1-2：连接和路由



创建从主 Replication Server 到复制 Replication Server 的路由时，事务从主服务器流向复制服务器。

如果计划在复制数据库中执行复制存储过程以更新主数据库，必须同时创建从复制 Replication Server 到主 Replication Server 的路由。

直接路由和间接路由

在拥有一个主 Replication Server 和多个复制 Replication Server 的复制系统中，您可以使用间接路由减少主 Replication Server 上的负载。间接路由使 Replication Server 可以通过单个中间 Replication Server 向多个目标发送消息。

具有中间节点的路由具有明显的优势：

- 更小的 WAN 流量

  Replication Server 向每个中间节点分发一条消息副本。中间节点上的 Replication Server 为它们的每个传出队列复制消息。

- 更低的 Replication Server 负载

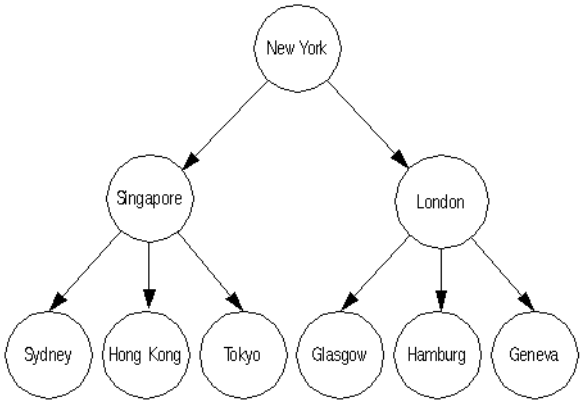
  在独立的计算机上运行的其他 Replication Server 会共同分担处理负载，从而降低主节点上 Replication Server 需要处理的量。

- 容错

  存储在中间节点上的消息可用于从远程节点发生的分区故障中恢复。有关详细信息，请参见 《Replication Server 管理指南》。

图 1-3 显示如何使用中间节点处理消息的分发。消息沿着直接路由到达中间节点。然后从中间节点开始，沿着直接路由到达本地节点。通过这种路由安排，主节点只需要发送两条消息，而不必发送八条。

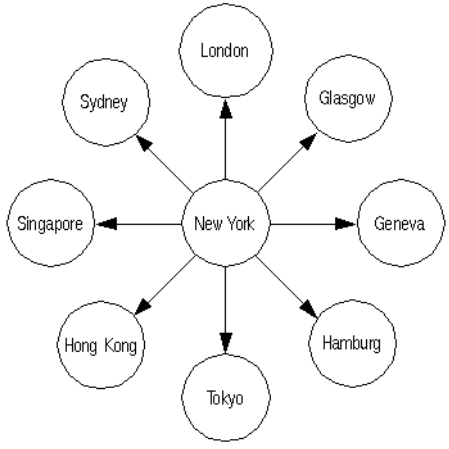
图 1-3：分层配置中的路由



中间节点减少了主节点的消息量，但延长了在主服务器和复制服务器进行更新的时间。请仔细规划路由；中间节点的数目够用即可。

如果不使用中间节点，路由将以星形配置的形式建立，如图 1-4 所示。

图 1-4：星形配置中的直接路由



当在主节点上更新某行时，主 Replication Server 通过 WAN 将消息发送到预订了该行的每个远程节点。例如，在图 1-4 中，纽约可以通过八条不同的路由发送同一数据。如果节点太多，网络很快就会因冗余消息而过载。

以分层排列的形式创建路由可以减少连接数和从主节点分发的消息数，从而实现负载平衡。在独立的计算机上运行的其他 Replication Server 会共同分担处理负载，从而降低主节点上 Replication Server 需要处理的量。

异构数据服务器支持

Replication Server 的开放式体系结构允许在复制系统中使用异构数据服务器。对 Adaptive Server 的支持已内置到 Replication Server 中。您可以通过创建 Replication Server 将对其他数据服务器使用的指令，建立相应数据服务器的接口。本节包含开放式体系结构各组成部分的简要概述。有关详细信息，请参见 《Replication Server 异构复制指南》和第 6 章 “将数据复制到异构数据服务器”。

开放式体系结构包括：

- 客户端 / 服务器接口
- Replication Agent
- 错误类及错误处理操作
- 函数、函数字符串和函数字符串类

客户端 / 服务器接口 (C/SI)

Sybase 客户端 / 服务器接口 (C/SI) 包括用于客户端 / 服务器通信的例程和协议。 Replication Server 使用 C/SI 以客户端的形式与数据服务器连接。如果数据服务器不支持 C/SI，则可以使用 Sybase 网关产品 （例如 DirectConnect），使 Replication Server 可以访问数据服务器。网关接受来自 C/SI 客户端的连接，并使用数据服务器的本机接口将客户端请求提交给数据服务器进行处理。数据服务器返回结果后，网关使用 C/SI 例程将结果返回给客户端。

Replication Agent

每个存储主数据或启动复制函数的数据库都需要 Replication Agent。Replication Agent 读取数据服务器的事务日志以检测主数据的更改情况和复制存储过程的执行情况。事务日志提供有关主数据修改的可靠信息来源，因为它包含已提交、可恢复事务的记录。

第 5 章“Replication Agent 介绍”介绍非 Adaptive Server 或 Sybase SQL Server 数据服务器的 Replication Agent。 Sybase 提供某些非 Sybase 数据服务器 （例如 DB2、 Informix、 Microsoft SQL Server 和 Oracle）的 Replication Agent。

处理数据服务器错误

Replication Server 根据您的指令处理数据服务器返回的错误和状态代码。每个供应商的数据服务器都有不同的错误代码集。复制命令语言 (RCL) 命令可用于：

- 为数据库创建错误类以对错误代码映射进行分组。
- 为数据服务器错误代码指派错误操作，例如 warn、 retry_log 和 stop_replication。
- 将错误类与数据库相关联。

函数、函数字符串和函数字符串类

为在异构数据库环境中运行， Replication Server 将数据库命令与用来向其他节点分发数据服务器请求的函数区别开来。 A *函数* 是一种 Replication Server 对象，代表数据服务器的各种操作，例如 insert、delete 和 begin transaction 等。Replication Server 使用函数字符串将函数转换为特定于数据服务器的命令。*函数字符串* 是 Replication Server 用来生成命令的一种模板，数据服务器将命令解释为事务控制指令或数据修改指令。

函数字符串类 是数据库使用的所有函数字符串的集合。函数字符串类为 Adaptive Server 和 DB2 数据服务器提供。事务控制指令的函数字符串仅为每个函数字符串类定义一次。插入行、删除行或更新行的函数字符串为数据库中的每个复制表定义一次。

函数字符串可以包含变量 （用问号 (?) 括起来的标识符），代表各列的值、过程参数、系统定义信息和用户定义变量。 Replication Server 在将函数字符串发送到数据服务器之前，会用实际值替换这些变量。

函数字符串依据格式可用于生成 RPC 或数据库命令，例如 SQL 语句。RPC 格式的函数字符串包含后接数据参数列表的远程过程调用。嵌入变量可用于为参数指派运行时值。 Replication Server 解释 RPC 函数字符串，建立远程过程调用，并将变量替换为运行时数据值。 RPC 可以执行在连接数据服务器的 Open Server 网关中注册的过程或 Adaptive Server 中的存储过程。

语言格式的函数字符串将数据库命令传递给数据服务器。 Replication Server 不会尝试解释该字符串，除非需要将嵌入变量替换为运行时数据值。例如，几个关系数据库服务器都使用 SQL 数据库语言。 SQL 命令应使用语言函数字符串表示。

RPC 函数字符串会比语言函数字符串更为有效，这是因为从 Replication Server 发出的网络数据包压缩比更高。

Replication Server 安全性

Replication Server 安全性包括由口令保护的登录名及基于 grant 和 revoke 命令的权限系统。 Replication Server 12.0 及更高版本支持第三方安全性服务，以确保跨网络消息传输的安全，并启用用户鉴定。 Replication Server 12.5 及更高版本通过 “高级安全性”选项支持安全套接层 (SSL) 这一基于会话的安全性。

登录名

每个 Replication Server 都使用与数据服务器登录名不同的登录名。许多客户端不需要 Replication Server 登录名，因为它们通过数据服务器应用程序即可完成工作。

Replication Server 登录名

安装 Replication Server 时， rs_init 会创建其他 Replication Server 和 Replication Agent 登录到此 Replication Server 时使用的 Replication Server 登录名。

复制系统管理员创建和管理 Replication Server 登录名和口令，用于管理复制数据或复制系统功能 （例如添加新用户或更改路由）。可以对口令进行加密。

数据服务器登录名

数据服务器登录名在客户端应用程序连接到数据服务器时使用。客户端应用程序使用数据服务器存储的数据，包括 Replication Server 复制的数据。数据库管理员创建和管理数据服务器登录帐号。

数据库管理员同时还管理客户端对表的复制副本的访问。Sybase 建议将复制表设为只读，因此，允许客户端查看复制数据，但应禁止插入、删除或更新行。

若要修改复制表，客户端必须修改主数据，使 Replication Server 可以将所进行的更改分发至预订数据的复制数据库。若要修改表，客户端必须在存储主数据的数据服务器上有登录名，并且拥有更新主数据所需的权限。

数据服务器维护用户登录名

Replication Server 对每个包含复制表的本地数据服务器数据库使用一个维护用户登录名。之所以称之为*维护用户*登录名，是因为 Replication Server 使用该登录名维护复制表。数据库管理员必须确保维护用户登录名拥有更新数据库中复制表所需的权限。

通常，维护用户所应用的事务将被 Replication Agent 过滤，以便这些事务不会被复制到数据库之外。但在某些应用程序中，必须复制这些事务。有关这些应用程序类型的详细信息，请参见第 3 章 “实现策略”。

权限

grant 和 revoke 命令用于为客户端授予和撤消 Replication Server 权限。表 1-1 列出了可授予客户端的权限。

表 1-1：Replication Server 权限

权限	能力
sa	授予被授权者系统管理员权限。具有 sa 权限的客户端可以执行任何 Replication Server 命令。
create object	允许被授权者创建、变更或删除包括复制定义和预订在内的 Replication Server 对象。
primary subscribe	授予被授权者在主数据库中创建预订 （但不能创建其他对象）的权限。若要在远程节点上创建预订，客户端需要在复制数据库中拥有 create object 权限，并且在主数据库中拥有 create object 或 primary subscribe 权限。
connect source	Replication Agent 使用的登录名必须拥有此权限。它使被授权者可以执行为 Replication Agent 保留的 RCL 命令。

基于网络的安全性

使用第三方基于网络的安全性机制，用户在登录时将经过安全系统的鉴定。鉴定是指检验用户与其说明的身份是否相符的过程。用户会收到一个认证，该认证可以代替口令传送到远程服务器。这样，用户只需进行一次登录，就可以实现对复制系统各组件的无缝访问。

基于网络的安全性机制还提供许多数据保护服务，如消息保密性和顺序混乱检查。 Replication Server 请求服务、准备数据并将数据发送到网络服务器进行加密或验证。执行完服务后，即会将数据返回到发出请求的 Replication Server 进行分发。

建立安全路径后，数据可以双向移动。路径的两端都必须支持相同的安全性机制并进行相同的配置。采用了网络安全性的所有计算机上都必须安装安全性机制，并且所有参与的计算机上必须安装 Replication Server 12.0 或更高版本。

有关复制系统中基于网络的安全性的信息，请参见 《参考手册》。

“高级安全性”选项

Replication Server 的 “高级安全性”选项提供安全套接层 (SSL)， SSL 是一种基于会话的安全性。SSL 是在因特网上安全传输敏感信息（例如信用卡号和股票交易）的标准。

SSL 通过几种加密算法提供轻量并且易于管理的安全性机制。该协议一般对安全性要求较高的数据库连接和路由使用。

有关使用 “高级安全性”选项的信息，请参见 《管理指南》。

总结

- Replication Server 维护网络上不同数据库中的表副本。复制的副本对节点的用户有两大好处：更快的响应速度和更高的可用性。
- 基于 Replication Server 建立的复制系统使用 Sybase 客户端 / 服务器接口连接各组件，其中包括数据服务器、 Replication Server、Replication Agent 和客户端应用程序等。
- Replication Server 设计了开放式接口，允许在复制系统中使用非 Sybase 数据服务器。
- 有一个表是主版本。所有其他表都是复制副本。
- 如果使用预订，表的复制副本可以包含表中的部分行。
- Replication Server 安全性由登录名、口令和权限组成。 Replication Server 还支持第三方基于网络的安全性机制。
- Replication Server 使用开放式并发控制，在故障发生时进行处理。与其他方法相比，开放式并发控制提供了更高的数据可用性、使用的资源更少并且与异构数据服务器配合地更好。

第 2 章

复制系统的应用程序体系结构

本章讨论几个很重要的主题，无论是应用程序设计人员还是用户，都有必要在实施 Replication Server 之前了解这些内容。

应用程序类型

确定您建立的 Replication Server 应用程序类型将在一定程度上决定实施该应用程序时使用的复制策略类型。第 3 章 “实现策略”介绍了几种不同的复制情况。

Replication Server 支持以下几种基本应用程序类型：

- 决策支持
- 分布式联机事务处理 (OLTP)
- 使用请求函数的远程 OLTP
- 热备份

其中每种应用程序类型在更新主数据以及在复制系统内分发主数据和复制数据的方式各不相同。

决策支持应用程序

决策支持客户端和生产联机事务处理 (OLTP) 客户端使用数据的方式不同。决策支持客户端执行长时间查询，该查询持有表锁以确保连续的一致性。相反， OLTP 客户端执行的事务必须迅速完成，不能接受由决策支持客户端的数据锁造成的延迟。如果这两类客户端各自维护自己的复制表，就不会相互干扰。

Replication Server 将与决策支持应用程序关联的处理从集中式联机事务处理应用程序分流到本地服务器。主数据库管理事务处理，而本地节点上的复制数据库处理决策支持客户端的信息请求。因为提供了独立、仅供参考的数据副本，所以 OLTP 系统可以保持畅通无阻。

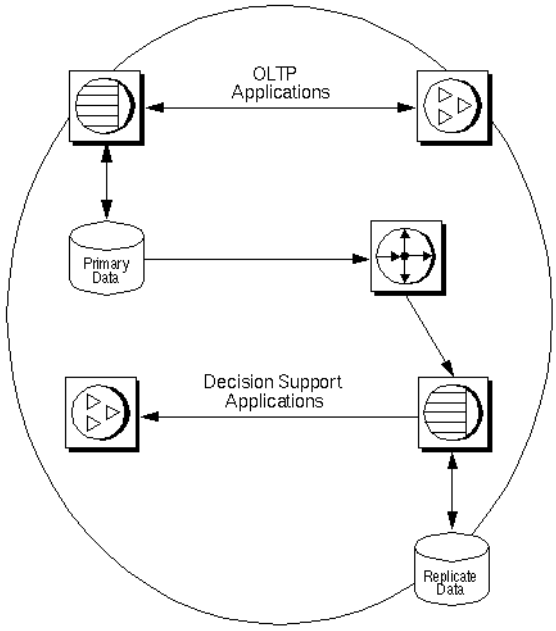
包含决策支持应用程序中使用的主数据的多份表副本可以在单个节点上维护，也可以通过网络在多个节点上维护。

单个节点上的多个副本

有时，需要在单个节点上维护表的多个复制副本。预订指定 Replication Server 维护复制数据的数据库。您可以为一个表的多份副本创建预订，方法是在同一个节点上的各个数据库中创建表，然后为每个表创建预订。

如果 OLTP 和决策支持客户端在同一个 LAN 上，一个 Replication Server 可以同时管理主数据和复制数据。图 2-1 图示了这种情况。

图 2-1：单个 LAN 的决策支持复制



为实现最佳性能，数据库通常由不同的数据服务器维护。通过预订可以请求每个数据库中要维护的不同数据子集，因此各个复制副本不必一致。

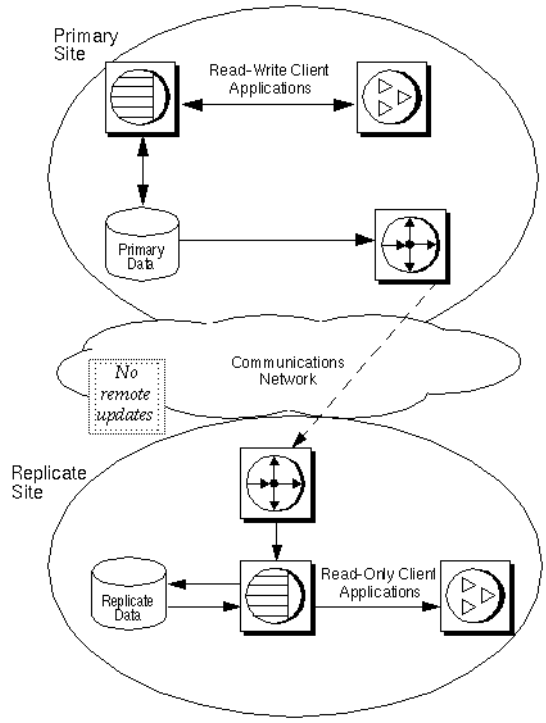
如果在同一个数据库中必须有表的两份副本，则可以对主表使用多个复制定义。一个复制定义可以使用 publishers 作为复制表名，另一个则使用 publishers2。如果希望不同的复制接收不同的列子集，也会用到多个复制定义。请参见第 55 页的 “多个复制定义”。

在 Adaptive Server 数据库中更新多个表的另一种方法是使用存储过程。将多个更新编码到存储过程中，然后编写 Replication Server 函数字符串以执行存储过程。也可以使用复制函数和存储过程更新多个表。

通过网络分发多份副本

当决策支持应用程序中通过 WAN 分发表的多份副本时，所有更新均由主节点上执行的应用程序执行，并分发给预订数据的远程节点。图 2-2 图示了这种情况。

图 2-2：多个 LAN 的决策支持复制



这种系统使用集中式主节点维护方法更新主数据。远程节点的客户端预订主数据的复制定义或发布。它们不更新主数据。有关此方法的信息，请参见第 26 页的 “集中式主数据维护”。

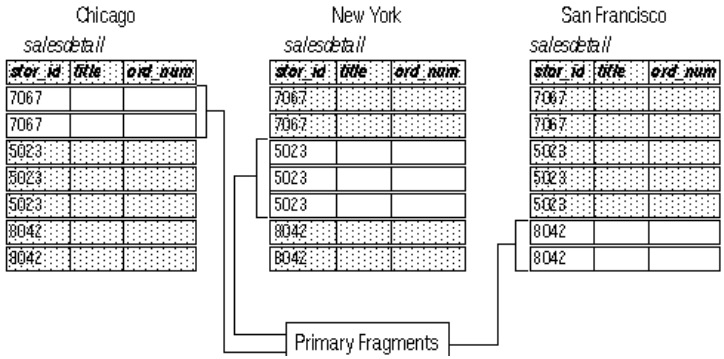
分布式 OLTP 应用程序

虽然某些分布式事务处理应用程序维护集中的主数据，但其他应用程序则将主数据分散在各个复制节点上。

主段是表的一个水平段，其中包含一组行的主版本。更新首先应用到主版本，然后分发至具有此数据的复制副本的节点。

按定义负责或具有部分表的各节点有多个主段。例如，图 2-3 中的 salesdetail 表在芝加哥、纽约和旧金山具有主段：

图 2-3：具有多个主段的表



由一列或多列构成的键可标识某行所属的主段。表 salesdetail 的键是 stor_id 列。

- stor_id 列为 “7067” 的行属于芝加哥节点上的主段。
- stor_id 列为 “5023” 的行属于纽约节点上的主段。
- stor_id 列为 “8042” 的行属于旧金山节点上的主段。

基于多个主段的应用程序模型有三种：

- 分布式主段 — 在此模型中，每个节点上的表既包含主数据又包含复制数据。对主版本的更新分发到其他节点。对非主数据的更新则从主节点接收。
- 全局集中 — 在此模型中，远程节点维护的多个主段将合并到中央节点上的单个集合复制表中。
- 再分布式全局集中 — 除了重新分发合并的表以外，此模型与全局集中模型相同。

有关这些模型的详细信息，请参见第 3 章 “实现策略”

使用请求函数的远程 OLTP

复制函数可用于远程执行事务。远程节点的客户端应用程序可以使用请求函数异步更新主数据。客户端应用程序不需要与主节点的网络连接，即使主节点无法访问，请求也可以被 Replication Server 接受。

请求函数在主数据库中执行存储过程后， Replication Server 即可复制在主数据库中所进行的部分或全部数据更改。这些更改可以作为数据行或应用函数传播到复制数据库。

本地更新应用程序

本地更新应用程序使远程节点的客户端可在复制系统从主节点返回所输入的更新之前查看这些更新。例如，如果更新了远程节点上的某个客户帐号，即使主节点无法访问，该节点的客户端也可以查看事务的结果。

本地更新可以通过使用待定更新表执行。对于每个复制表，相应的本地表都包含临时更新，即已提交给主节点但尚未经复制系统返回的更新。客户端应用程序将更新待定事务表，同时向主节点发送请求函数。有关实施该应用程序类型的信息，请参见第 65 页的 “使用本地待定表的示例”。

对主副本的更新成功后，更新将分发到各远程节点，包括发起该事务的节点。您可以创建函数字符串或复制存储过程，用于更新复制表和从待定表中删除本地更新。这样，客户端应用程序就可以了解哪些事务以被确认，哪些事务还待定。

备份应用程序

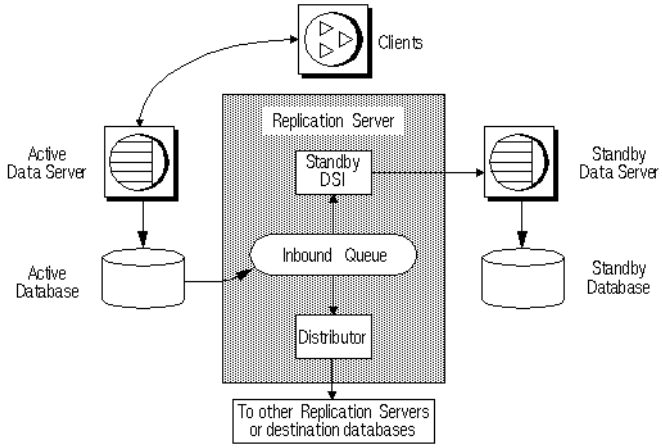
热备份应用程序是维护一对 Adaptive Server 或 SQL Server 数据库的 Replication Server 应用程序，其中一个数据库充当另一个的备份副本。

客户端应用程序通常更新**活动数据库**，而 Replication Server 则维护**备份数据库**作为活动数据库的副本。 Replication Server 通过复制从活动数据库事务日志中检索的事务，使备份数据库与活动数据库保持一致。

如果活动数据库出现故障，或者需要对活动数据库或数据服务器执行维护操作，则可以切换到备份数据库，以便客户端应用程序可以在几乎没有中断的情况下恢复工作。

图 2-4 图示了一个热备份系统。

图 2-4：热备份系统



热备份应用程序中的两个数据库看起来就像是复制系统中的单个逻辑数据库。根据应用程序的不同，此逻辑数据库可能不参与复制，也可能分别成为复制系统中其他数据库的主数据库或复制数据库。

有些 Replication Server 和 RepAgent 功能显式支持热备份应用程序。有关热备份应用程序的详细信息，请参见 《Replication Server 管理指南》。

松散一致性对应用程序的影响

复制数据库中的数据与主数据库中的数据之间保持 “松散的一致性”。复制数据稍微滞后于主数据，滞后时间为从主数据库分发更新所需的时间。当系统正常运行时，延迟时间可能为数秒 （或更短）。如果某个组件出现故障 （如网络连接暂时中断），更新可能会延迟数分钟、数小时或数天。

虽然复制数据可能滞后于主数据，但在事务上与主数据一致。Replication Server 按照事务提交到主数据库的顺序将事务传递给复制数据库。这样可以确保复制数据经历的一系列状态与主数据相同。

松散一致性的重要性因应用程序的不同而有所不同，即使在同一应用程序内也会不同。有些应用程序可以容忍平均系统滞后时间 （有时还能容忍更长的延迟时间），不需特别规定。有些应用程序在滞后时间太长时需要特别处理，有些应用程序则要求对特定事务类型进行特殊处理。

控制重要事务的风险

由数据复制引起的延迟增加了某些业务决策的风险。例如，用于批准现金提取的某个银行业应用程序使用可获得的最新帐户信息，验证客户的余额是否足以支付提取额。如果在主数据库中进行提取的信息未到达复制数据库，则使用复制数据库的应用程序批准的提取额会有超出客户帐户中可用资金的风险。

为降低风险，银行业应用程序可以区分大额交易和小额交易。例如，批准 100 美元的提款只需基于本地复制数据库中的帐户余额即可，但在批准 1000 美元的提款之前，就需要登录到主数据库检查帐户余额。

测量滞后时间

测量复制节点的延迟时间可以降低某些事务的风险。较短的滞后时间表明主数据和复制数据基本一致。较长的滞后时间表明主数据和复制数据之间可能会有较大差异。

应用程序可以通过测量滞后时间来：

- 降低风险，方法是在滞后时间延长时限制客户端可以执行的事务。例如，上一节中介绍的银行业应用程序可以将滞后时间包括在其批准规则中。当延迟时间小于一分钟时，可以根据本地复制表中的余额最多提取 1000 美元。但当滞后时间超过一分钟时，该应用程序需要登录到主数据库才能批准 500 美元以上的提款。
- 为客户端提供数据复制的 “性能表”。客户端可以使用估计的滞后时间作为参考。例如，决策支持用户如果注意到滞后时间很长，可以等待本地数据与主数据一致，并且在滞后时间缩短后，再基于复制数据进行分析。

注释 Sybase Central 包括心跳功能，使您可以跟踪滞后时间。有关详细信息，请参见 Replication Server 的联机帮助。

更新主数据的方法

在复制系统中，数据行的主副本就是确定副本。在主数据库中提交的更新是有权威的，将分发给预订数据的所有数据库。

在主数据库中提交了事务后， Replication Server 将分发这些事务。由于不分发对复制数据的更改，所以将复制数据库中的数据设为对客户端只读，并且将所有客户端事务路由到主数据库。

在基于 Replication Server 的复制系统中，有四种方法可以更新主数据：

- 主数据维护集中在主节点进行。客户端不能从远程节点更新主数据。
- 远程节点的客户端通过网络连接更新主数据。
- 远程节点的客户端使用请求函数更新主数据。
- 主数据维护分布在多个主节点上进行。导致的任何冲突都必须避免或解决。

集中式主数据维护

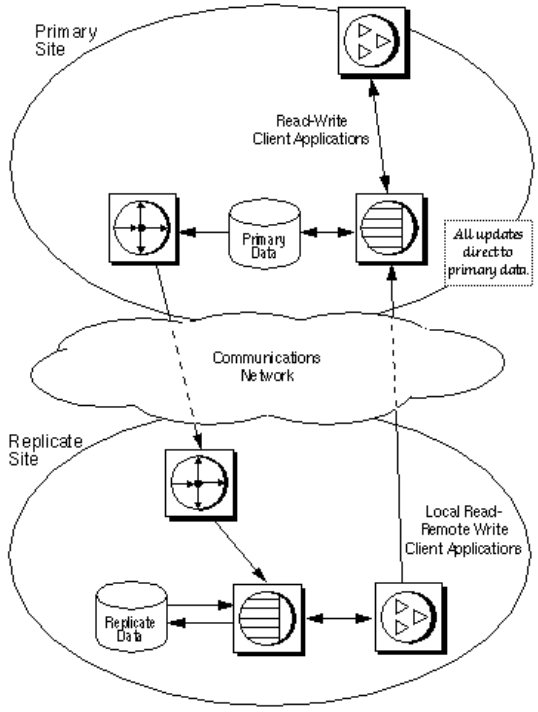
对于远程客户端，此方法最简单，限制也最多。远程节点的客户端应用程序仅使用复制数据作为参考。这种体系结构可用于创建生产 OLTP 系统的副本，使决策支持应用程序可独立于 OLTP 系统运行。有关使用此方法更新主数据的应用程序的信息，请参见第 19 页的 “决策支持应用程序”。

通过网络连接维护主数据

对于某些应用程序，远程节点的客户端必须更新主数据。客户端实现此目标最简单的方法是通过网络直接连接到主数据服务器上。 Replication Server 按惯例将更新从主节点分发到远程节点。

图 2-5 图示了这种设计。

图 2-5：通过网络维护主数据



此体系结构通过 WAN 使用客户端连接。此方法适用于拥有大量数据且更新频率较低的应用程序。因为直接对主数据进行更新，所以可以将更新分发到预订数据的所有节点。在远程节点上，数据的本地复制副本可用于决策支持应用程序以降低网络负载。

通过请求函数维护主数据

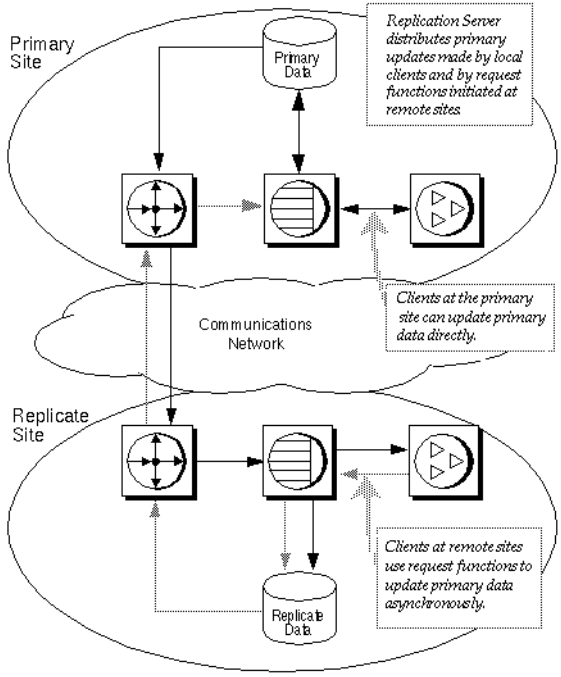
在复制数据库中执行的复制存储过程称为**请求函数**。之所以称为请求函数，是因为该存储过程请求在主数据库中进行更改。如果使用请求函数修改主数据，可不通过 WAN 连接到主数据服务器。

当在复制数据库中执行用户存储过程时， Replication Server 会将过程发送给创建函数复制定义的主数据库。

当执行存储过程时， Adaptive Server 将在事务日志中记录过程的执行情况，以使 RepAgent 可以将请求传递给复制系统。尽管请求函数可以执行本地数据库中的任何工作，但通常并不需要这样做。实际工作在主 Replication Server 接收到过程并在主数据库中执行时完成。

在图 2-6 中，灰色箭头显示传递请求函数的流程。黑色箭头显示复制数据从主数据库到复制数据库的流程。

图 2-6：通过传递请求函数修改主数据



复制节点上的 Replication Server 将请求函数存储在队列中，并尽快将请求传递到主 Replication Server。如果无法访问主 Replication Server，则远程 Replication Server 会存储该过程，直到问题解决为止。

管理多个主节点的更新冲突

如果存在多个主节点，则应设计远程更新，使它们不会因为多个远程节点同时请求更新同一信息而导致错误。

如果来自两个不同节点的更新在同一主节点上应用时产生冲突，其中一个更新将被拒绝，并且更新被拒绝的节点上的复制数据将与主数据不一致。

若要处理存在多个主节点时发生的节点间并发冲突，请遵循以下准则：

- *如果每行都有所有者* — 设计应用程序，以避免出现节点间冲突。例如，将在一个节点上执行的更新限制为那些不能由其他节点的客户端更新的行。这样可以确保主节点上的更新不会发生冲突。
- *如果没有所有权分段* — 向函数字符串中添加版本控制信息，以便检测和处理冲突。

通过设计将冲突排除在应用程序之外

处理来自不同节点的冲突更新的方法之一是构建应用程序和环境，以便冲突不会发生。例如，将客户帐号信息分发给每个分部的应用程序只需要在客户当地的分部进行更新。这样就可以防止两个客户端同时在不同数据库中更新同一帐号。

另一种方法是在复制表的主键中加入位置键，例如分部 ID。如果每个节点对其所有事务都使用唯一的位置键，就不会发生节点间冲突。

通过版本控制的更新

您可以使用版本控制更新执行以下操作：

- 检测和解决冲突更新
- 在没有单主节点源时解决多主节点冲突
- 向单个主节点发出多个请求

接受还是拒绝更新取决于每次更新行时所更改的版本列。版本列可以是随每次更新增大的数字、时间戳或一组唯一值中的某个其他值。

若要更新行，应用程序必须提供主节点上版本列的当前值。通常，该值作为参数提供给复制存储过程。主节点的存储过程将检查版本参数，并在检测到冲突后采取适当的措施。如果应用程序选择回退该事务，则将其写入例外日志。

注释 使用版本控制管理更新冲突需要仔细规划和设计。当存在多个主节点时，为表的每一行设置所有者通常更为简单有效。

第 3 章

实现策略

本章介绍可用来实现您设计的 Replication Server 应用程序的若干模型和策略。本章提供了一些可按照您自己的应用程序进行相应调整的过程和示例脚本。

模型和策略概述

本章讨论的模型包括：

- 基本主复制模型 — 包括集中式主数据和分布式复制数据
- 分布式主段模型 — 包括在复制系统中分发的主数据和复制数据
- 全局集中 — 包括分布式主数据和集中式复制数据
- 再分发式全局集中 — 与全局集中相同，但更新数据将被重新分发到复制数据库
- 热备份应用程序 — 包括两个数据库，其中的一个作为另一个的备份，可以共同作为一个逻辑单元参与复制

此外，本章还介绍了可利用的模型变化形式和其他策略：

- 多个复制定义
- 发布
- 请求函数
- 待定表
- 主 / 明细关系

这些方法在第 55 页的 “模型变化形式和策略”中有详细介绍。

您要构建的应用程序的类型、更新主数据的方式以及管理潜在更新冲突的方式共同决定了实现复制应用程序时需要使用的模型。

例如，可以使用基本主复制模型实现一个决策支持应用程序或小容量分布式 OLTP 系统。您可以使用基本主复制模型或再分发全局集中模型来实现一个决策支持应用程序，具体使用哪种模型取决于主数据是集中式的还是分段式的。使用分布式主段模型实现分布式 OLTP 应用程序时可以结合使用全局集中，也可以不使用全局集中，具体取决于是否有其他决策支持需要。

基本主复制模型

基本主复制模型可用于将数据从主数据库复制到目标数据库。此模型非常适用于决策支持应用程序，不过小容量事务处理应用程序可以直接通过 WAN 或通过请求函数 （复制的存储过程）以远程方式更新主数据。从远程节点更新的主数据接着可以复制回预订节点。

您可以使用所有以下方法或其中的某种方法实现基本主复制模型：

- 表复制定义
- 应用函数
- 请求函数

本节提供了关于使用表复制定义和应用函数的一些基本示例。有关请求函数的示例以及主复制模型的其他更高级的应用，请参见第 55 页的 “模型变化形式和策略”。

使用表复制定义

使用表复制定义可以将主源中的数据作为只读副本来复制。

您可以为一个主表创建一个或多个复制定义，不过特定复制表只能预订其中的一个复制定义。有关使用多个复制定义的示例，请参见第 55 页的 “多个复制定义”。

您也可以将复制定义收集到一个发布中，然后通过一个发布预订一次预订其中的所有定义。有关使用发布的示例，请参见第 58 页的 “发布”。

对于每个您要根据基本主复制模型复制的表，您需要：

- 在不同的 Replication Server 之间建立路由和连接。
- 在主数据库中创建要复制的表。
- 在目标数据库中创建您要将数据复制到其中的一个或多个表。
- 创建索引并授予对这些表的、适当的访问权限。

在主节点上：

- 使用 sp_setreptable 系统过程标记要复制的主表。
- 在主 Replication Server 上为该表创建一个 （或多个）复制定义。

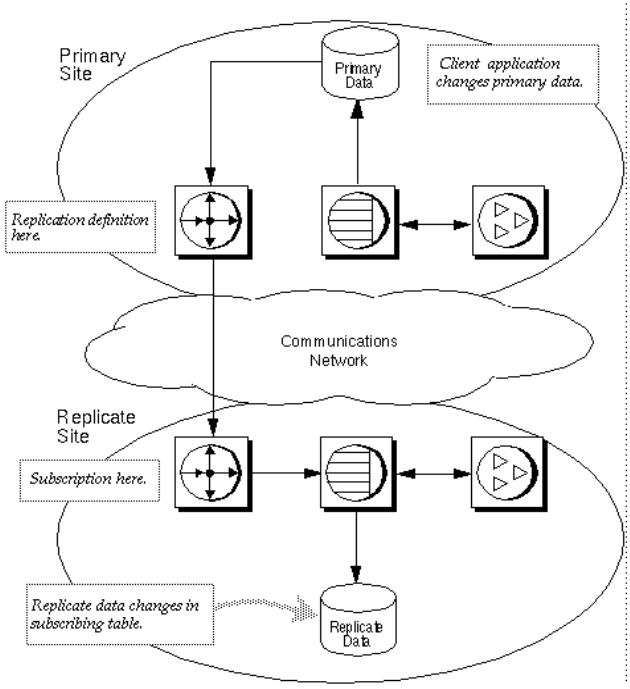
在复制节点上：

- 在每个复制 Replication Server 上为表复制定义创建预订。

有关建立基本主复制模型的详细信息，请参见 《Replication Server 管理指南》。

在图 3-1 中，主 （东京）节点上的客户端应用程序对主数据库中的 publishers 表作了一些更改。在复制 （悉尼）节点上， publishers 表预订了主 publishers 表中 pub_id 大于或等于 1000 的那些行。

图 3-1：使用表复制定义的基本主复制模型



标记要复制的表

以下脚本将 publishers 表标记为要复制的表。

```
-- Execute this script at Tokyo data server
 -- Marks publishers for replication
 sp_setreptable publishers, 'true'
 go
 /* end of script */
```

复制定义

以下脚本在主 Replication Server 上为 publishers 表创建了表复制定义。

```
-- Execute this script at Tokyo replication server
 -- Creates replication definition pubs_rep
 create replication definition pubs_rep
 with primary at TOKYO_DS.pubs2
 with all tables named 'publishers'
```

```
(pub_id char(4),
   pub_name varchar(40)
   city varchar(20)
   state varchar(2)
primary key (pub_id)
go
/* end of script */
```

预订

以下脚本为在主 Replication Server 上定义的复制定义创建了一个预订。

```
-- Execute this script at Sydney replication server
 -- Creates subscription pubs_sub
 Create subscription pubs_sub
 for pubs_rep
 with replicate at SYDNEY_DS.pubs2
 where pub_id >= 1000
 go
 /* end of script */
```

使用应用函数

您也可以使用应用函数向具有复制数据的远程节点复制存储过程调用。使用应用函数复制主数据可以：

- 降低 WAN 的网络通信量
- 因为应用函数执行速度更快，所以可增加吞吐量，缩短延迟时间
- 使系统设计更加模块化

在下例中，主 （东京）节点上的客户端应用程序通过执行用户存储过程 upd_publishers_pubs2 对主数据库中的 publishers 表进行了更改。执行 upd_publishers_pubs2 将调用函数复制，进而使相应的存储过程 （名称也是 upd_publishers_pubs2）在复制数据服务器上执行。

要为实现基本主复制模型的应用程序创建应用函数，需要执行以下操作：

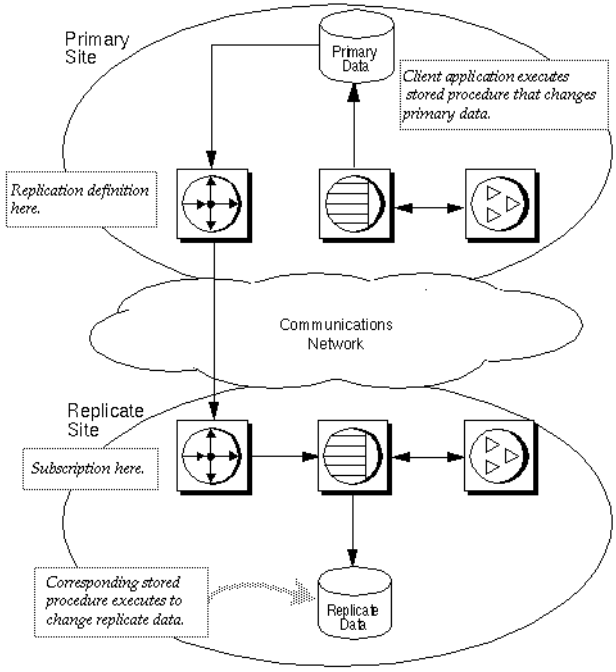
在主节点上：

- 在主数据库中创建用户存储过程。
- 使用 sp_setrepproc 将该用户存储过程标记为用于传递复制函数。
- 向相应用户授予适当的过程权限。
- 在主 Replication Server 上，为其名称、参数和数据类型与该存储过程的名称、参数和数据类型匹配的存储过程创建函数复制定义。可以仅指定要复制的参数。

在复制节点上：

- 在复制数据库中，创建一个其参数（或参数子集）及数据类型与主数据库中创建的存储过程相同的存储过程。向维护用户授予适当的过程权限。
- 在复制 Replication Server 中，创建一个对函数复制定义的预订。

图 3-2：使用应用函数的基本主复制模型



存储过程

以下脚本在主节点和复制节点上为 publishers 表创建存储过程。

```
-- Execute this script at Tokyo and Sydney data servers
 -- Creates stored procedure upd_publishers_pubs2
 create procedure upd_publishers_pubs2
 (@pub_id         char(4),
 @pub_name          varchar(40),
 @city             varchar(20),
 @state            char(2))
 as
 update publishers
 set
    pub_name = @pub_name,
    city = @city,
    state = @state
 where
    pub_id = @pub_id
 go
 /* end of script */
```

函数复制定义

以下脚本在主 Replication Server 上为 publishers 表创建了一个函数复制定义。该复制定义使用的名称、参数和数据类型与主数据库中的存储过程使用的相同。

```
-- Execute this script at Tokyo replication server
 -- Creates replication definition
         upd_publishers_pubs2
 create function replication definition
upd_publishers_pubs2
 with primary at TOKYO_DS.pubs2
 (@pub_id char(4),
 @pub_name varchar(40),
 @city varchar(20),
 @state     char(2))
 go
 /* end of script */
```

预订

可以使用以下两种方法之一为函数复制定义创建预订：

- 使用 create subscription 命令和非实现方法。

  如果主数据已装载到复制节点并且当前没有进行更新，则使用此方法。

- 使用 define subscription、 activate subscription 和 validate subscription 命令以及批量实现方法。

  如果要让装载数据和更新数据同时进行，则使用此方法。

下面给出了这两种方法的示例。

使用非实现方法

以下脚本使用非实现方法在复制 Replication Server 上为主 Replication Server 上定义的复制定义创建预订。

```
-- Execute this script at Sydney replication server
 -- Creates subscription using no-materialization
        for upd_publishers_pubs2
 create subscription upd_publishers_pubs2_sub
 for upd_publishers_pubs2
 with replicate at SYDNEY_DS.pubs2
 without materialization
 go
 /* end of script */
```

使用批量实现

以下脚本在复制 Replication Server 上为主 Replication Server 上定义的复制定义创建、激活并验证了一个预订。

```
-- Execute this script at Sydney replication server
 -- Creates subscription using bulk materialization
        for upd_publishers_pubs2
 define subscription upd_publishers_pubs2_sub
 for upd_publishers_pubs2
 with replicate at SYDNEY_DS.pubs2
 go
```

```
activate subscription upd_publishers_pubs2_sub
 for upd_publishers_pubs2
 with replicate at SYDNEY_DS.pubs2
 go
 /* Load data. If updates are in progress, use activate
subscription with the "with suspension" clause and re-
sume connection after the load. */

validate subscription upd_publishers_pubs2_sub
 for upd_publishers_pubs2
 with replicate at SYDNEY_DS.pubs2
 go
 /* end of script */
```

分布式主段模型

在此模型中，每个节点上的表都包含主数据和复制数据。不过，各个节点分别用作特定行集（称为*段*）的主节点。对主段的更新将分发给其他节点。对非主数据的更新则从其他段的主节点接收。

采用分布式主段模型的应用程序使用分布式表来包含主数据和复制数据。每个节点上的 Replication Server 都把对本地主数据所做的修改分发给其他节点，并将从其他节点接收的修改应用到本地复制的数据中。

必须在每个节点上执行以下任务，以便在分布式主段模型中复制表：

- 在每个数据库中创建表。在每个数据库中，表的结构都应该相同。
- 创建索引并授予对这些表的、适当的访问权限。
- 允许使用 sp_setreptable 系统过程复制表。
- 在每个节点上为该表创建复制定义。
- 在各个节点上，分别创建一个对其他节点上的复制定义的预订。如果 *n* 代表节点数，则创建 *n*-1 个预订。

图 3-3 绘制了分布式主段的数据流：

图 3-3：分布式主段模型

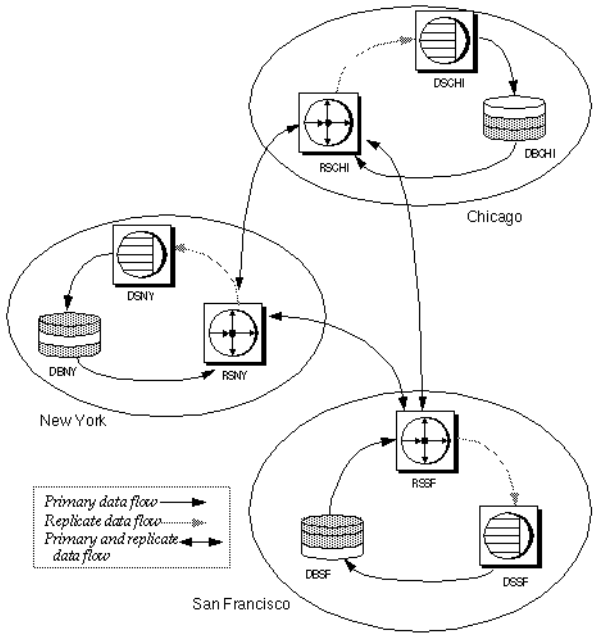
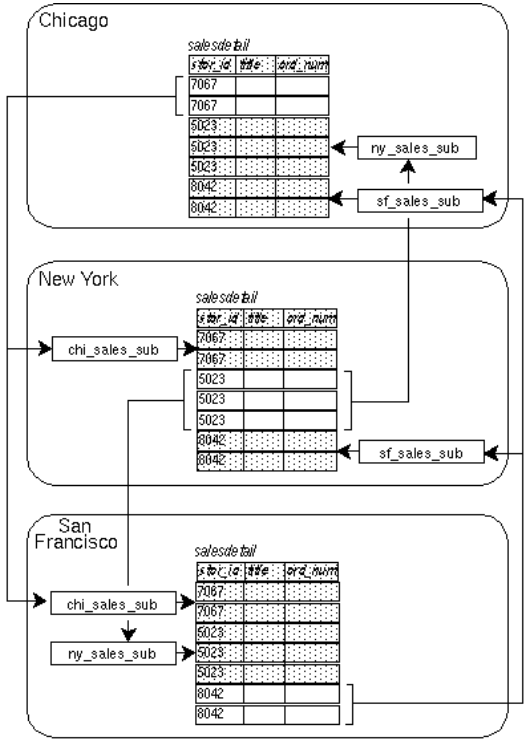


图 3-4 展示了使用三个节点的分布式主段建立的 salesdetail 表。每个节点通过两个预订接收复制数据。

图 3-4：带有三个分布式主段的表



复制定义

以下脚本在各个节点上为 salesdetail 表创建复制定义：

```
-- Execute this script at Chicago RSCHI.
 -- Creates replication definition chi_sales.
 create replication definition chi_sales_rep
     with primary at DSCHI.DBCHI
     with all tables named 'salesdetail'
     (stor_id char(4),
         ord_num varchar(20),
         title_id varchar6),
         qty smallint,
         discount float)
```

```
    primary key (stor_id, ord_num)
    searchable columns
        (stor_id, ord_num, title_id)
go
/* end of script */
-- Execute this script at New York RSNY.
 -- Creates replication definition ny_sales.
 create replication definition ny_sales_rep
    with primary at DSNY.DBNY
    with all tables named 'salesdetail'
    (stor_id char(4),
        ord_num varchar(20),
        title_id varchar6),
        qty smallint,
        discount float)
    primary key (stor_id, ord_num)
    searchable columns
        (stor_id, ord_num, title_id)
    go
 /* end of script */
-- Execute this script at San Francisco RSSF.
 -- Creates replication definition sf_sales.
 create replication definition sf_sales_rep
    with primary at DSSF.DBSF
    with all tables named 'salesdetail'
    (stor_id char(4),
        ord_num varchar(20),
        title_id varchar6),
        qty smallint,
        discount float)
    primary key (stor_id, ord_num)
    searchable columns
        (stor_id, ord_num, title_id)
    go
 /* end of script */
```

预订

每个节点都有对其他两个节点的复制定义的预订。以下脚本用于创建预订：

```
-- Execute this script at Chicago RSCHI.
 -- Creates subscriptions to ny_sales and sf_sales.
 create subscription ny_sales_sub
     for ny_sales_rep
     with replicate at DSCHI.DBCHI
     where stor_id = '5023'
 go
 create subscription sf_sales_sub
     for sf_sales_rep
     with replicate at DSCHI.DBCHI
     where stor_id = '8042'
 go
 /* end of script */

-- Execute this script at New York RSNY.
 -- Create subscriptions to chi_sales and sf_sales.
 create subscription chi_sales_sub
     for chi_sales_sub
     with replicate at DSNY.DBNY
     where stor_id = '7067'
 go
 create subscription sf_sales_sub
     for sf_sales_rep
     with replicate at DSNY.DBNY
     where stor_id = '8042'
 go
 /* end of script */

-- Execute this script at San Francisco RSSF.
 -- Creates subscriptions to chi_sales and ny_sales.
 create subscription chi_sales_sub
     for chi_sales_rep
     with replicate at DSSF.DBSF
     where stor_id = '7067'
 go
 create subscription ny_sales_sub
     for ny_sales_rep
     with replicate at DSSF.DBSF
     where stor_id = '5023'
 go
 /* end of script */
```

全局集中

在此模型中，在远程节点上维护的多个主段合并为中央节点上的一个集合复制表。

全局集中模型具有多个分布式主段和一个集中式合并复制表。每个主节点上的表只包含对该节点而言是主数据的那些数据。并没有数据复制到这些节点。全局集中表是各主节点数据的 “集中”形式。

全局集中模型要求每个主节点上的复制定义都是唯一的。合并数据的节点具有对每个主节点上的复制定义的预订。

主节点需要 Replication Agent 而中央节点不需要，因为数据并不是从中央节点复制的。

必须执行以下任务才能从分布式主段创建全局集中模型：

- 在每个主数据库和中央节点上的数据库中创建表。这些表应具有相同的结构和名称。
- 创建索引并授予对这些表的、适当的访问权限。
- 在每个远程数据库中，允许使用 sp_setreptable 系统过程复制表。
- 在每个远程节点上为该表创建复制定义。
- 在要合并数据的总部节点上，创建对远程节点上的复制定义的预订。

图 3-5 展示了全局集中应用程序模型的数据流。

图 3-5：具有分布式主段的全局集中模型

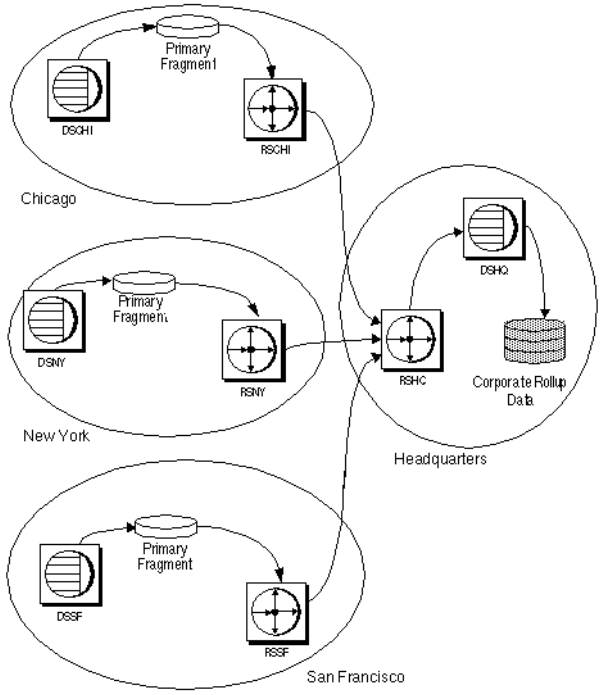
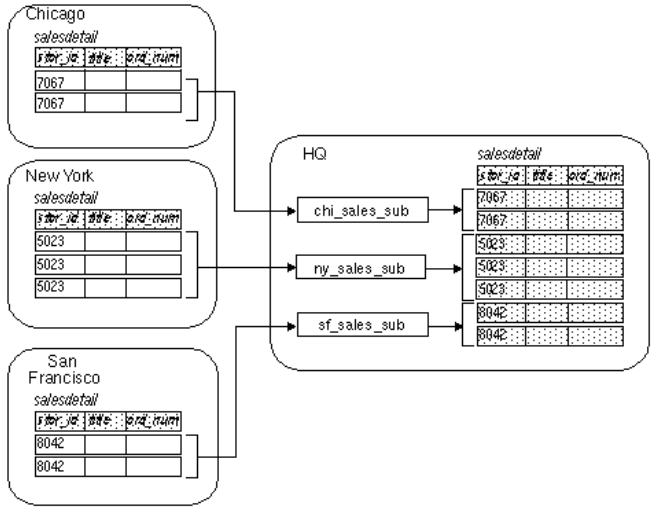


图 3-6 展示了在总部节点进行全局集中的 salesdetail 表。总部节点通过三个预订从远程节点接收数据。

图 3-6：具有多个主段的表



复制定义

以下脚本在各个主节点上为 salesdetail 表创建复制定义：

```
-- Execute this script at Chicago RSCHI.
 -- Creates replication definition chi_sales.
 create replication definition chi_sales_rep
     with primary at DSCHI.DBCHI
     with all tables named 'salesdetail'
     (stor_id char(4),
         ord_num varchar(20),
         title_id varchar6),
         qty smallint,
         discount float)
     primary key (stor_id, ord_num)
     searchable columns
         (stor_id, ord_num, title_id)
 go
 /* end of script */
```

```
-- Execute this script at New York RSNY.
 -- Creates replication definition ny_sales.
 create replication definition ny_sales_rep
     with primary at DSNY.DBNY
     with all tables named 'salesdetail'
     (stor_id char(4),
         ord_num varchar(20),
         title_id varchar6),
         qty smallint,
         discount float)
     primary key (stor_id, ord_num)
     searchable columns
         (stor_id, ord_num, title_id)
 go
 /* end of script */
-- Execute this script at San Francisco RSSF.
 -- Creates replication definition sf_sales.
 create replication definition sf_sales_rep
     with primary at DSSF.DBSF
     with all tables named 'salesdetail'
     (stor_id char(4),
         ord_num varchar(20),
         title_id varchar6),
         qty smallint,
         discount float)
     primary key (stor_id, ord_num)
     searchable columns
         (stor_id, ord_num, title_id)
 go
 /* end of script */
```

预订

总部节点上具有对所有三个主节点上的复制定义的预订。各主节点上没有预订。以下脚本在 RSHQ Replication Server 中创建预订：

```
-- Execute this script at Headquarters RSHQ.
 -- Creates subscriptions to chi_sales, ny_sales,
 -- and sf_sales.
 create subscription chi_sales_sub
     for chi_sales_rep
     with replicate at DSHQ.DBHQ
     where stor_id = '7067'
 go
create subscription ny_sales_sub
```

```
    for ny_sales_rep
    with replicate at DSHQ.DBHQ
    where stor_id = '5023'
go
create subscription sf_sales_sub
    for sf_sales_rep
    with replicate at DSHQ.DBHQ
    where stor_id = '8042'
go
/* end of script */
```

再分发全局集中

再分发全局集中与全局集中模型相同，只不过需要将合并表重新分发给远程节点。

远程节点上分布的主段被集中到中央节点上的合并表中。在合并各段的节点上， RepAgent 会像处理主数据那样处理合并表。

合并表通过复制定义来描述。其他节点可以创建对此表的预订。

通常，Adaptive Server 的 RepAgent 会过滤出由维护用户进行的更新。这样可确保复制数据不会作为主数据重新分发。

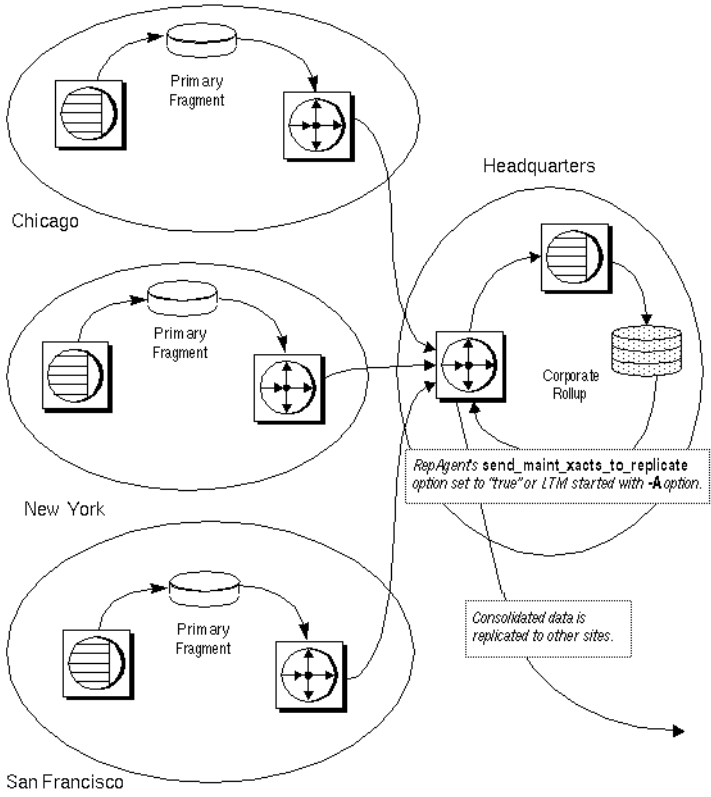
可以在再分发全局集中模型中使用 RepAgent send_maint_xacts_to_replicate 选项。如果启动 RepAgent 时将 send_maint_xacts_to_replicate 设置为“true”， RepAgent 会将所有更新都提交给 Replication Server （就如同这些是客户端应用程序所做的更新）。（如果使用的是 LTM，则应使用 -A 选项启动该程序。）

如果使用再分发全局集中模型：

- 应禁止主节点再次预订其主数据。如果未加禁止，事务可能会在系统中无限循环。
- 禁止应用程序更新全局集中表。所有更新都应从主节点执行。

图 3-7 展示了基于再分发全局集中模型的应用程序中的数据流。

图 3-7：具有分布式段的再分发全局集中



再分发全局集中模型的设计与全局集中模型基本相同，只有以下几方面除外：

- RepAgent 必须安装在 DBHQ 数据库所在的总部节点上。启动 RepAgent 时必须设置 send_maint_xacts_to_replicate 选项，以便其传送维护用户的日志记录。

  如果使用的是 LTM，则必须使用 -A 选项启动 LTM。

- RSHQ RSSD 需要 RepAgent，因为要从该节点分发数据。

- 必须在总部节点上为 salesdetail 表创建复制定义。其他节点可以创建对此复制定义的预订，但主节点不能预订其自身的主数据。
- RSHQ Replication Server 必须能够路由到其他能够创建对合并复制表的预订的节点。如果主节点创建了预订，则必须创建从 RSHQ 到这些主节点的路由。

热备份应用程序

在热备份应用程序中， Replication Server 维护一对 Adaptive Server（或 SQL Server），其中一个作为另一个的备份。

通常，在客户端应用程序更新活动数据库时， Replication Server 负责维护另一个数据库，作为活动数据库的备用副本。如果活动数据库失败，或者如果需要对活动数据服务器或数据库进行维护，则可以在几乎不影响客户端应用程序的情况下切换到备用数据库 （再切换回来）。

在热备份应用程序中，可以建立三个连接：

- 从 Replication Server 映射到当前活动数据库的逻辑连接
- 活动数据库的物理连接
- 备用数据库的物理连接

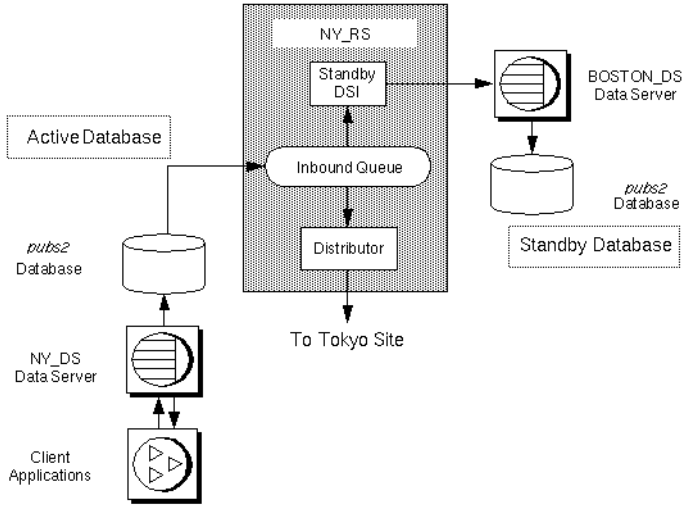
相对于复制系统中的其他数据库，热备份应用程序中的逻辑数据库可作为以下数据库之一：

- 不参与复制的数据库
- 主数据库
- 复制数据库

本节中的过程演示了如何为充当复制系统中的主数据库的数据库建立热备份系统。

图 3-8 展示了 NY_DS 数据服务器上的 pubs2 数据库的热备份应用程序，该应用程序运行在 BOSTON_DS 数据服务器上。该数据库复制到 TOKYO_DS。

图 3-8：热备份系统



在这种情况下， pubs2 数据库充当复制环境中的主数据库。创建备用数据库的 pubs2 主数据库称为*活动数据库*。

建立热备份应用程序

可通过以下过程为活动数据库建立热备份应用程序。本过程中已经建立了一个活动数据库。如果尚未创建活动数据库，则过程会稍有不同。继续操作前一定要查阅 《Replication Server 管理指南》中的热备份信息。

注释 在此过程中必须使用 Adaptive Server 或 SQL Server 数据库。

在活动数据库中：

1 使用 sp_reptostandby 存储过程将整个活动数据库标记为要复制到备用数据库。

sp_reptostandby 支持复制数据操纵语言 (DML) 以及受支持的数据定义语言 (DDL) 命令和存储过程。有关详细信息，请参考《Replication Server 管理指南》中的第 15 章 “管理热备份应用程序”。

2 使用设置了 send_warm_standby_xacts 选项的 sp_config_rep_agent 存储过程重新配置 RepAgent。重新启动 RepAgent。

3 为活动数据库维护用户授予 replication_role。

4 在活动数据服务器上，向活动数据库添加备用数据库的维护用户，并为这个新维护用户授予 replication_role。此步可确保装载备用数据库 （步骤 8）后，备用数据库中仍存在维护用户 ID。

5 登录到要管理热备用数据库的 Replication Server 中，并使用 create logical connection 命令为活动数据库创建逻辑连接。逻辑连接的名称必须与活动数据库的名称相同。

注释 如果在创建活动数据库连接*之前*创建逻辑连接，逻辑连接和活动数据库应使用不同的名称。

6 在备用数据服务器上，创建大小与活动数据库相同的备用数据库。

7 使用 Sybase Central 或 rs_init 创建备用数据库连接。有关详细信息，请参见所在平台的 Replication Server 联机帮助及 Replication Server 安装和配置指南。

创建连接后，登录到 Replication Server 中并使用 admin logical_status 命令确保新连接处于 “活动”状态。

8 使用不带 rs_init “转储标记”选项的 dump 和 load 初始化备用数据库。（也可以使用 bcp。有关详细信息，请参考 《Replication Server 管理指南》。）

a 在 Replication Server 上，挂起活动数据库连接。

注释 如果无法挂起活动数据库，则应使用带 rs_init “转储标记”选项的 dump 和 load。

b 在活动的 Adaptive Server 上，转储活动数据库。

c 将活动数据库转储内容装载到备用数据库中。

d 在备用 Adaptive Server 上，令备用数据库联机。

9　在 Replication Server 上，使用 resume connection 命令恢复活动数据库和备用数据库的连接。

使用 admin logical_status 命令检查逻辑状态。在活动数据库和备用数据库都被标记为 “活动”后再继续操作。

10　检验发生在活动数据库和备用数据库之间的修改。

使用 isql 更新活动数据库中的某条记录，然后在备用数据库中检验是否更新。

切换到备用数据库

如果有必要从活动数据库切换到备用数据库，则需要采取步骤，防止客户端应用程序针对活动数据库执行事务或进行更新。切换完成后，客户端可以连接到新的活动数据库，继续工作。有关详细信息，请参见第 54 页的 “将客户端切换到新数据库”。

在切换到备用数据库之前，应确定是否有必要切换：

- 如果活动数据服务器遇到瞬时故障，则不要切换。瞬时故障是指无需额外的恢复步骤即可在 Adaptive Server 重新启动时恢复的故障。
- 如果活动数据库长时间不可用，则需要切换。

必须使用 switch active 命令切换活动数据库和备用数据库。下面的过程说明如何令第 51 页图 3-8 中所示的热备份系统从活动数据库切换到备用数据库。

1　在 Replication Server 上，使用 switch active 将处理操作切换到备用数据库。

2　监控切换的进度。当备用连接被激活而先前的活动连接挂起时，切换即告完成。

   a　在 Replication Server 上，使用 admin_logical_status 命令检查逻辑状态。

   b　要跟踪切换的进度，应检查 Replication Server 错误日志中的最后几个条目。

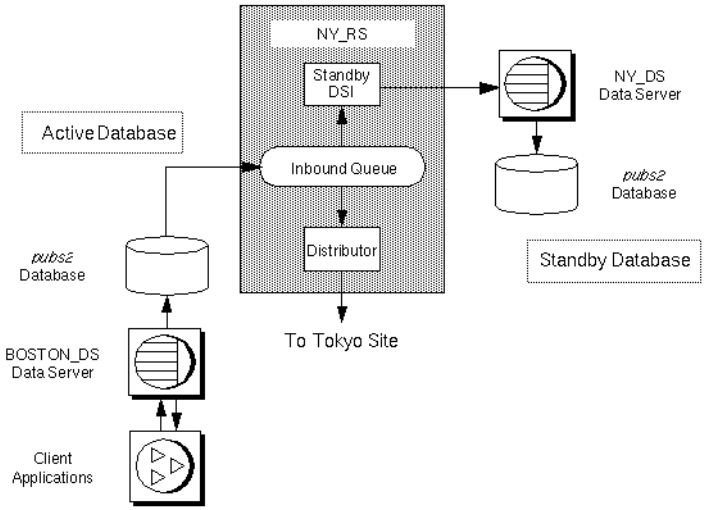
3　启动新的活动数据库的 RepAgent

4 确定要对旧的活动数据库执行的操作。您可以：

- 令该数据库作为新的备用数据库联机并恢复连接，以便 Replication Server 应用新事务；或者
- 使用 drop connection 命令断开数据库连接。以后可以再次将其添加为新的备用数据库。

5 使用 isql 在新的活动数据库中更新某条记录，然后在新的备用数据库中检查是否更新。

图 3-9：切换之后的热备份系统



将客户端切换到新数据库

从活动数据库切换到备用数据库并不能将客户端应用程序切换到新的活动数据服务器和数据库。必须设计一种处理客户端切换的方法。例如，您可以：

- 设置两个接口文件，一个用于客户端应用程序，另一个用于 Replication Server。切换期间修改客户端接口文件，令其指向新的活动服务器。
- 创建包含数据服务器符号名称的接口文件条目，供客户端应用程序使用。在切换期间修改与该符号名称关联的地址信息。
- 利用中间 Open Server 之类的机制，将客户端应用程序数据服务器连接自动映射到当前的活动数据服务器。

有关详细信息，请参见 《Replication Server 管理指南》。

模型变化形式和策略

本节介绍用于实现复制系统设计的若干模型变化形式和其他策略。具体包括：

- 多个复制定义 — 该策略通过多个主表复制定义来指定不同的表名、列集和列名，以便为预订复制表提供不同的主表视图
- 发布 — 该策略允许用户通过单个预订对一组复制定义进行预订
- 请求函数 — 基本主复制模型的变化形式，它允许远程节点无需直接访问主数据库即可更改主数据
- 待定表 — 该策略与请求函数配合使用，允许用户在主数据更新返回到复制节点之前查看更新结果
- 实现主 / 明细关系 — 使用请求函数和存储过程来确保正确的预订迁移

多个复制定义

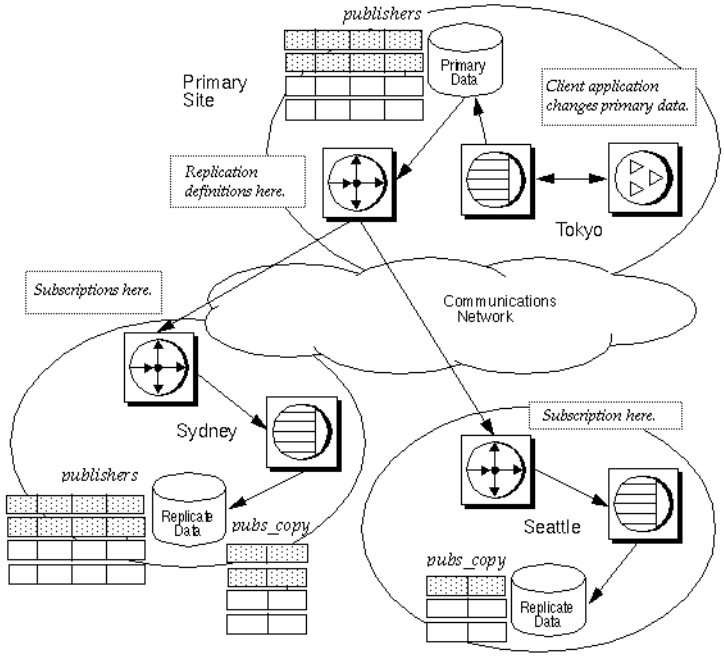
可以为单个主表创建多个复制定义。每个复制定义都可以指定不同的表名、列集和列名，以便为各个预订复制表提供不同的主表视图。

注释 可以为一个主表创建多个复制定义，而一个复制表也可以预订到多个表复制定义。但是，一个复制表只能对每个主表的一个复制定义进行预订。

若要使用多个复制定义建立系统，请按照第 33 页的“使用表复制定义”中的指导说明，按需创建多个复制定义，并为每个复制定义创建一个预订。使用多个复制定义就是在使用主复制模型的一种变化形式。

在图 3-10 中，主 （东京）节点上的客户端应用程序对主数据库中的 publishers 表进行了更改。在一个复制 （悉尼）节点上， publishers 表预订到完整表，而 pubs_copy 表只预订到 pub_id 和 pub_name 列。在另一个复制节点 （西雅图）上， pubs_copy 表预订到 pub_id 和 pub_name 列，其中 pub_id 等于或大于 1000。

图 3-10：多个复制定义



复制定义

以下这些脚本在主 Replication Server 上为 publishers 表创建表复制定义。各个复制定义分别描述了主表的不同视图。

```
-- Execute this script at Tokyo replication server
 -- Creates replication definitions pubs_rep and
          pubs_copy_rep
 create replication definition pubs_rep
 with primary at TOKYO_DS.pubs2
 with all tables named 'publishers'
 (pub_id char(4),
    pub_name varchar(40)
    city varchar(20)
    state varchar(2)
 primary key (pub_id)
 go

 create replication definition pubs_copy_rep
 with primary at TOKYO_DS.pubs2
```

```
with primary table named 'publishers'
with replicate table named 'pubs_copy'
   (pub_id char(4),
   pub_name varchar(40)
primary key (pub_id)
go
/* end of script */
```

预订

以下脚本为主 Replication Server 上定义的复制定义创建了预订。

```
-- Execute this script at Sydney replication server
 -- Creates subscription pubs_sub and pubs_copy_rep
 Create subscription pubs_sub
 for pubs_rep
 with replicate at SYDNEY_DS.pubs2
 go

 create subscription pubs_copy_sub
 for pubs_copy_rep
 with replicate at SYDNEY_DS.pubs2
 go
 /* end of script */
-- Execute this script at Seattle replication server
 -- Creates subscription pubs_copy_sub
 create subscription pubs_copy_sub
 for publ_copy_rep
 with replicate at SEATTLE_DS.pubs2
 where pub_id >= 1000
 go
 /* end of script */
```

发布

使用发布可以收集表和 / 或存储过程的复制定义，然后作为一个组预订到这些定义。

使用发布可以监控对一组表和过程的发布预订的状态。“发布”用法并不构成独立的模型；它提供了一种任何模型都可以使用的分组方法。

使用发布时，需要创建和管理以下对象：

- 项目 — 表或存储过程的复制定义扩展，可用于将表或函数复制定义放入发布。
- 发布 — 来自同一主数据库的多个项目的组
- 发布预订 — 对一个发布的预订。创建发布预订时， Replication Server 会为发布的每个项目都创建一个预订。

以下步骤概括了使用发布复制数据的过程。

在主节点上：

1 创建或选择要纳入发布的复制定义。

2 使用 create publication 命令创建发布。

3 使用 create article 命令创建引用所选复制定义的项目。

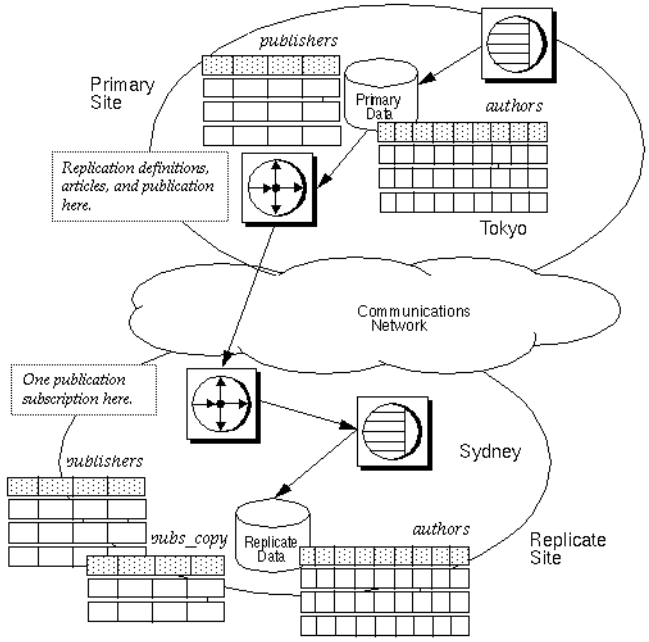
4 使用 validate publication 命令验证发布。

在复制节点上：

1 使用 create subscription 命令可创建对一个发布的预订。

在图 3-11 中，两个项目引用的表复制定义 pubs_rep 和一个项目引用的函数复制定义均被收集到发布 pubs2_pub 中。

图 3-11：发布



存储过程

以下脚本创建存储过程 update_ authors_pubs2，该存储过程可用于更新 pubs2 数据库中的 authors 表。在主节点和复制节点都创建同一过程。

```
-- Execute this script at the Tokyo and Sydney
        data servers
 -- Creates the stored procedure
        update_authors_pubs2
create procedure upd_authors_pubs2
(@au_id          id,
au_lname           varchar(40),
au_fname          varchar(20),
phone                char(12),
address           varchar((12),
city                     varchar(20),
state             char(2),
```

```
country              varchar(12),
postalcode            char(10)
as
update authors
set
   au_lname = @varchar(40),
   au_fname = @varchar(20),
   phone = @char(12),
   address = @varchar((12),
   city = @varchar(20),
   state = @char(2),
   country = @varchar(12),
   postalcode = @char(10)
where au_id = @au_id
go
/* end of script */
```

函数复制定义

以下脚本在主节点上创建函数复制定义。

```
-- Execute this script at the Tokyo
        replication server
 -- Creates the function replication definition
        update_authors_pubs2
 create function replication definition
 upd_authors_pubs2
 with primary at TOKYO_DS.pubs2
 (@au_id          id,
 au_lname           varchar(40),
 au_fname          varchar(20),
 phone               char(12),
 address           varchar((12),
 city                    varchar(20),
 state             char(2),
 country             varchar(12),
 postalcode           char(10)
 go
 /* end of script */
```

表复制定义

以下脚本在主 Replication Server 上为 publishers 表创建表复制定义。

```
-- Execute this script at Tokyo replication server
 -- Creates replication definitions pubs_rep
 create replication definition pubs_rep
 with primary at TOKYO_DS.pubs2
 with all tables named 'publishers'
 (pub_id char(4),
    pub_name varchar(40)
    city varchar(20)
    state varchar(2)
 primary key (pub_id)
 go
 /* end of script */
```

发布

以下脚本在主 Replication Server 上创建 pubs2_pub 这一发布。

```
-- Execute this script at Tokyo replication server
 -- Creates publication pubs_pub
 create publication pubs2_pub
 with primary at TOKYO_DS.pubs2
 go
 /* end of script */
```

项目

以下脚本在主 Replication Server 上为发布 pubs2_pub 创建项目。该脚本为复制定义 pubs_rep 创建了两个项目。

```
-- Execute this script at Tokyo replication server
 -- Creates articles upd_authors_art,
           pubs_art, and pubs_copy_art
 create article upd_authors_art
     for pubs2_pub
 with primary at TOKYO_DS.pubs2
 with replication definition upd_authors_rep
 go

create article pubs_art
     for pubs2_pub
 with primary at TOKYO_DS.pubs2
 with replication definition pubs_rep
 go
```

```
create article pubs_copy_art
     for pubs2_pub
 with primary at TOKYO_DS.pubs2
 with replication definition pubs_rep
 where pub_id >= 1000
 go
 /* end of script */
```

变化形式

以下脚本将发布 pubs2_pub 的状态改为 “有效”。

```
-- Execute this script at Tokyo replication server
 -- Validates the publication pubs2_pub
 validate publication pubs2_pub
 with primary at TOKYO_DS.pubs2
 go
 /* end of script */
```

预订

以下脚本创建了对发布 pubs2_pub 的预订 pubs2_pub_sub。运行此脚本时，Replication Server 会为 upd_authors_art、pubs_art 和 pubs_copy_art 创建项目预订。

```
-- Execute this script at Sydney replication server
 -- Creates publication subscription pubs2_pub_sub
 create subscription pubs2_pub_sub
 for publication pubs2_pub
    with primary at TOKYO_DS.pubs2}
 with replicate at SF.pubs2
   without materialization
 go
 /* end of script */
```

请求函数

可以使用请求函数对主数据调用存储过程。请求函数允许远程节点上的客户端无需直接访问主数据库即可更新主数据。 Replication Server 将对主数据库调用存储过程，由该存储过程执行所请求的事务。此方法在保持基本主复制模型的同时又允许分发事务。您不必在多个复制节点间分段存储主数据。

以下两个示例分别说明：

- 使用请求函数的系统
- 使用请求函数、应用函数和本地待定表的系统

使用请求函数的基本示例

在本例中，复制 （悉尼）节点上的客户端应用程序执行了存储过程 upd_publishers_pubs2_req，该过程并未更改复制数据库但却引发了相关存储过程 upd_publishers_pubs2 的执行，在主 （东京）节点上更改数据。upd_publishers_pubs2_req 是一个*空正文*存储过程。

执行以下任务创建请求函数。主数据库不需要预订；在复制定义主节点之外的其他节点执行的函数将自动流向主节点。

在复制节点上：

- 在复制数据服务器上创建空正文存储过程，并且
  - 使用 sp_setrepproc 将该过程标记为用于传递复制函数。
  - 为相应用户授予适当的过程权限。

空正文过程要么不执行任何操作，要么显示一条消息，指明更新待定。它的名称可能与主数据库中的关联存储过程并不相同。

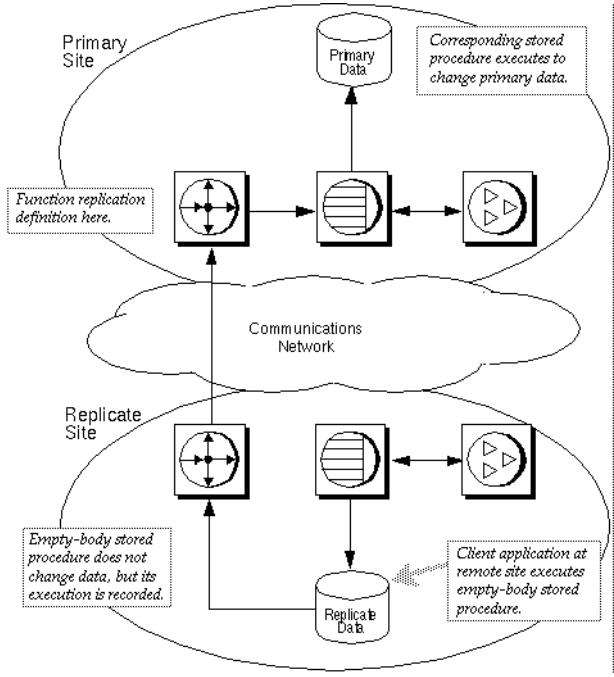
在主节点上：

- 创建用于更新主数据的存储过程。为相应用户授予权限。
- 在主 Replication Server 上，为复制数据库中的空正文存储过程创建函数复制定义。函数复制定义和空正文存储过程应使用相同的名称。

一旦通过复制请求存储过程更新了主数据，即可通过表复制或应用函数复制将这些数据传递到复制节点 （包括请求节点）。

注释 不必为请求函数创建预订。

图 3-12：请求函数



存储过程

在主节点上：　上面的脚本在主节点上创建了用户存储过程 upd_publisher_pubs2。

在复制节点上：　以下脚本创建空正文存储过程 upd_publishers_pubs2_req。

```
-- Execute this script at Sydney replication server
 -- Creates stored procedure
        upd_publishers_pubs2_req
 create procedure upd_publishers_pubs2_req
 (@pub_id char(4),
 @pub_name  varchar(40),
 @city  varchar(20),
 @state  char(2))
 as
 begin
 if (select 1) > 1
    print "Submitting request."
 end
 go
 /* end of script */
```

函数复制定义

以下脚本在主 Replication Server 上创建函数复制定义。它与空正文存储过程同名。 deliver as 子句通知主数据服务器在主节点上执行 *upd_publisher_pubs2*。

```
-- Execute this script at Tokyo replication server
 -- Creates replication definition
 create function replication definition
upd_publishers_pubs2_req
 with primary at TOKYO_DS.pubs2
 deliver as 'upd_publisher_pubs2'
    (@pub_id char(4),
    @pub_name varchar(40),
    @city varchar(20),
    @state  char(2))
 go
 /* end of script */
```

使用本地待定表的示例

待定表对应用函数和请求函数进行了设计上的改进，允许远程节点上的客户端更新主数据并在从主节点返回更新前查看更新内容。使用此模型可以实现本地更新应用程序。

在此策略中，远程节点上的客户端应用程序执行用户存储过程，使用请求函数来更新主节点上的数据。对主数据的更改将通过应用函数复制到远程节点。本地待定表允许远程节点上的客户端在复制系统返回更新内容之前在复制节点上查看待定的更新。

在复制数据服务器上执行用户存储过程时，客户端应用程序将执行以下操作：

- 引发关联存储过程的执行并更新主节点上的数据
- 在本地待定表中输入这些更新

在主数据库上更新成功后，更新将分发到远程节点，包括最初发出事务的节点。在复制节点上，存储过程更新复制表并从待定表中删除相应的更新。

要使用应用函数、请求函数和本地待定表，必须完成以下任务。

在复制节点上：

- 在复制数据库中创建待定表。授予适当的权限。
- 在复制数据库中创建用户存储过程，启动请求函数并将更新数据插入待定表。
- 使用 sp_setrepproc 将该用户存储过程标记为用于传递复制函数。
- 为相应用户授予过程权限。
- 在复制数据库中创建用户存储过程，更新复制表并从待定表中删除相应的更新。为维护用户授予适当的权限。
- 创建对函数复制定义的预订。

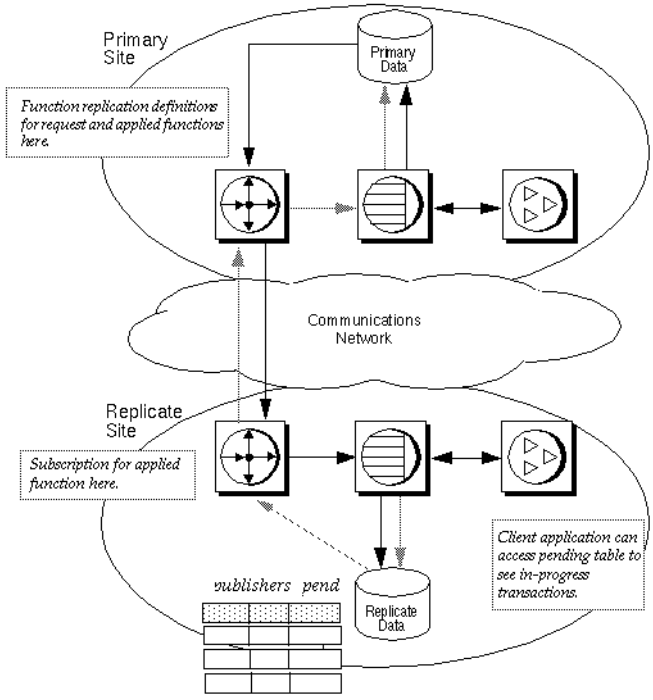
在主节点上：

- 创建修改主数据的存储过程。
- 创建应用函数的函数复制定义。
- 创建请求函数的函数复制定义。

在本例中，复制（悉尼）节点上的客户端应用程序执行了存储过程 upd_publishers_pubs2_req，该过程在 publishers_pend 表中插入值并引发关联存储过程 upd_publishers_pubs2 在主（东京）节点上执行。在主节点上执行 upd_publishers_pubs2 导致存储过程 upd_publishers_pubs2 在复制节点上执行，从而更新 publishers 表并从 publishers_pend 表中删除相应信息。

图 3-13 展示了使用应用函数、请求函数和本地待定表时的数据流。灰色箭头表示传递请求函数的流程。灰色箭头表示传递应用函数的流程。

图 3-13：请求函数和本地待定表



待定表

以下脚本在复制数据库中创建待定表。

```
-- Execute this script at Sydney data server
 -- Creates local pending table
 create table publishers_pend
 (pub_id char(4) not null,
 pub_name varchar(40) null,
 city varchar(20) null,
 statechar(2) null)
 go
 /* end of script */
```

存储过程

以下脚本在主节点上创建存储过程 upd_publisher_pubs2。

以下脚本在复制节点上修改 upd_publishers_pub2_req 存储过程。insert into 子句通知复制 Replication Server 将值插入 publishers_pend 表。

```
-- Execute this script at Sydney data server
 -- Creates stored procedure
 create procedure upd_publishers_pubs2_req
 (@pub_id    char(4),
 @pub_name    varchar(40),
 @city       varchar(20),
 @state            char(2))
 as
 insert into publishers_pend
 values
    (@pub_id, @pub_name, @city, @state)
 go
 /* end of script */
```

以下脚本为复制节点修改 upd_publishers_pubs2 过程。该过程更新 publishers 表并从 publishers_pend 表中删除相应信息。

```
-- Execute this script at Sydney data server
 -- Creates stored procedure upd_publishers_pubs2
 create procedure upd_publishers_pubs2
 (@pub_id char(4),
 @pub_name varchar(40),
 @city varchar(20),
 @state  char(2))
 as
 update publishers
 set
    pub_name = @pub_name,
    city = @city,
    state = @state
 where
    pub_id = @pub_id
 delete from publishers_pend
 where
    pub_id = @pub_id
 go
 /* end of script */
```

函数复制定义

第 65 页的 “函数复制定义”中的脚本在主节点上为**请求函数**创建了函数复制定义 upd_publishers_pubs2_req。

第 37 页的 “函数复制定义”中的脚本在主节点上为**应用函数**创建了函数复制定义 upd_publishers_pubs2。

预订

第 38 页的 “预订”中的脚本在复制节点上为函数复制定义 upd_publisher_pubs2 创建了预订 upd_publishers_pubs2_sub。

实现主 / 明细关系

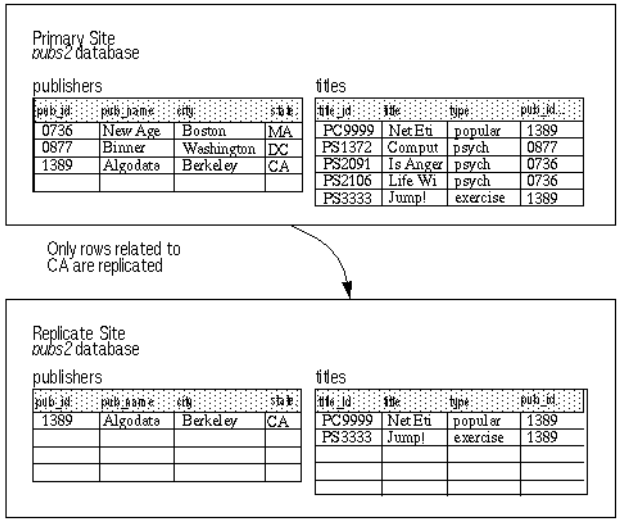
可以使用应用函数只将所选数据复制到远程节点。以这种方式使用应用函数可以降低网络通信量。

要实现主 / 明细关系，需使用应用函数对主 / 明细表进行选择性预订。在下例中，

- publishers 和 titles 表存在于主节点和复制节点上的 pubs2 数据库中。
- NY_DS 为主节点数据服务器， SF_DS 为复制节点数据服务器。

图 3-14 描述了主节点和复制节点上的 publishers （主表）和 titles （明细表）：

图 3-14：主 / 明细关系中使用的示例表



主节点包含所有记录，但复制节点只关注与加利福尼亚州 (CA) 有关的记录。只需要复制根据 state 列选择的 publishers 和 titles 记录。不过，只有 publishers 表才包含 state 列。

向 titles 表添加 state 列将增加系统的冗余。另一种更为有效的解决方法是：通过存储过程连接主表与明细表的更新，然后使用应用函数复制存储过程。维护选择性预订的逻辑即包含在存储过程中。

例如，如果在主节点， publisher 所在的 state 由 NY 改为 CA，则必须在复制节点上插入该 publisher 对应的记录。通过令预订中的行发生更改的更新来插入或删除复制行，这称为 *预订迁移*。

为确保正确地迁移预订，需要预订一组 “高级”的存储过程，用于控制实际执行更新的存储过程。从主节点到复制节点复制的实际是对高级存储过程的调用。

要处理 state 列中的更改，复制节点必须在新或旧的 state 为 CA 时预订这些更新。

以下各节列出了为在复制 Replication Server 上支持选择性替代而必须执行的操作。

在主节点和复制节点上：

- 创建向 publishers 表和 titles 表插入记录的存储过程，并创建控制执行这两个插入过程的高级存储过程。
- 创建从 publishers 表和 titles 表中删除记录的存储过程，并创建控制执行这两个删除过程的高级存储过程。
- 创建更新 publishers 表和 titles 表中记录的存储过程，并创建控制执行这两个更新过程的高级存储过程。
- 为所有高级存储过程授予适当的权限。

在主节点上：

- 使用 sp_setrepproc 标记用于复制的各个高级存储过程。
- 为各个高级存储过程创建函数复制定义。

在复制节点上：

- 创建对函数复制定义的预订。

包含 insert 子句的存储过程

主节点与复制节点上的插入过程完全相同。控制插入过程的高级存储过程和插入过程本身均遵守以下逻辑规则：

- 只有不存在 title ID 时才插入 publisher 记录。
- 只有存在 publisher 时才插入 title 记录。

以下脚本创建 ins_publishers 和 ins_titles 插入存储过程及高级存储过程 ins_pub_title。

```
-- Execute this script at NY and SF data servers
 -- Creates stored procedure
 create procedure ins_publishers
 (@pub_id char(4), @pub_name varchar(40)=null,
 city varchar(20)=null, @state char(2)=null)
 as
       insert publishers values (@pub_id,
       @pub_name, @city, @state)
 /* end of script */
-- Execute this script at NY and SF data servers
 -- Creates stored procedure
 create procedure ins_titles
 (@title_id tid, @title varchar(80), @type char(12),
 @pub_id char(4)=null, @price money=null, @advance money=null,
 @total_sales int=null, @notes varchar(200)=null,
 @pubdate datetime, @contract bit)
 as
```

```
 if not exists (select pub_id from publishers
          where pub_id=@pub_id)
       raiserror 20001 "** FATAL ERROR: Invalid publishers id **"
 else
       insert titles values (@title_id, @title, @type, @pub_id,
       @price,@advance, @total_sales, @notes, @pubdate, @contract)
 /* end of script */
-- Execute this script at NY and SF data servers
 -- Creates stored procedure
 create procedure ins_pub_title
 (@pub_id char(4), @pub_name varchar(40)=null,
 @city varchar(20)=null, @state char(2),
 @title_id tid=null, @title varchar(80)=null, @type char(12)=null,
 @price money=null, @advance money=null,
 @total_sales int=null, @notes varchar(200)=null,
 @pubdate datetime=null, @contract bit)
 as
 begin
       if @pub_name != null
       exec ins_publishers @pub_id, @pub_name, @city, @state
       if @title_id != null
       exec ins_titles @title_id, @title, @type, @pub_id, @price,
       @advance, @total_sales, @notes, @pubdate, @contract
 end/*
 end of script */
```

包含 delete 子句的存储过程

主节点与复制节点上的删除过程完全相同。控制删除过程的高级存储过程和删除过程本身均遵守以下逻辑规则：

- 删除某条记录后，所有相关子记录也随之删除。
- 存在 title 记录时不删除 publisher 记录。

以下脚本创建 del_publishers 和 del_titles 存储过程及高级存储过程 del_pub_title。

```
-- Execute this script at NY and SF data servers
 -- Creates stored procedure
 create procedure del_publishers
 (@pub_id char(4))
 as
 begin
 if exists (select * from titles where pub_id=@pub_id)
       raiserror 20005 "**FATAL ERROR: Existing titles**"
 else
```

```
      delete from publishers where pub_id=@pub_id
 end
 /* end of script */
-- Execute this script at NY and SF data servers
 -- Creates stored procedure /
 create procedure del_titles
 (@title_id tid, @pub_id char(4)=null)
 as
 if @pub_id=null
      delete from titles where title_id=@title_id
 else
      delete from titles where pub_id=@pub_id
 end
 /* end of script */

-- Execute this script at NY and SF data servers
 -- Creates stored procedure
 create procedure del_pub_title
 (@pub_id char(4), @state char(2), @title_id tid=null)
 as
 begin
      if @title_id != null
         begin
            exec del_titles @title_id
            return
         end
      if @pub_id != null
         begin
            exec del_titles @title_id, @pub_id
            exec del_publishers @pub_id
         end
 end
 /* end of script */
```

包含 update 子句的存储过程

主节点上的更新过程不同于复制节点上的更新过程。

在主节点上：

更新过程遵守以下逻辑规则：

- 对未知 *pub_id* 发出错误。
- 如果不存在 title，则插入一个。

以下脚本创建 upd_publishers 和 upd_titles 两个存储过程，并创建控制 upd_publishers 和 upd_titles 两者的执行的高级存储过程 upd_pub_title。

请注意 upd_pub_title 存储过程另有一列 old_state，该列支持复制节点预订那些迁移的行。

```
-- Execute this script at NY data servers
 -- Creates stored procedure
 create procedure upd_publishers
 (@pub_id char(4), @pub_name varchar(40),
 @city varchar(20), @state char(2))
 as
 if not exists
    (select * from publishers where pub_id=@pub_id)
    raiserror 20005 "**FATAL ERROR: Unknown publishers id**"
 else
    update publishers set
    pub_name=@pub_name,
    city=@city,
    state=@state
    where pub_id = @pub_id
 end
 /* end of script */
-- Execute this script at NY data servers
 -- Creates stored procedure
 create procedure upd_titles
 (@title_id tid, @title varchar(80), @type char(12),
 @pub_id char(4)=null, @price money=null, @advance money=null,
 @total_sales int=null, @notes varchar(200)=null,
 @pubdate datetime, @contract bit)
 as
 if not exists
    (select * from titles where title_id=@title_id)
    raiserror 20005 "**FATAL ERROR: Unknown title id**"
 else
    update titles set
    title=@title,
    type=@type,
    pub_id=@pub_id,
    price=@price,
    advance=@advance,
    total_sales=@total_sales,
    notes=@notes,
    pubdate=@pubdate,
    contract=@contract
    where title_id = @title_id
 end
 /* end of script */
-- Execute this script at NY data server
 -- Creates stored procedure
```

```
create procedure upd_pub_title
(@pub_id char(4), @pub_name varchar(40)=null,
@city varchar(20)=null, @state char(2)=null,
@title_id tid=null, @title varchar(80)=null, @type char(12)=null,
@price money=null, @advance money=null,
@total_sales int=null, @notes varchar(200)=null,
@pubdate datetime=null, @contract bit, @old_state char(2))
as
begin
if not exists (select * from publishers where pub_id=@pub_id)
   raiserror 20005 "**FATAL ERROR: Unknown publishers id**"
else
   exec upd_publishers @pub_id, @pub_name, @city, @state
if @title_id != null
begin
   if not exists
      (select * from titles where title_id=@title_id)
      exec ins_titles @title_id, @title, @type, @pub_id,
         @price, @advance, @total_sales, @notes, @pubdate,
         @contract
   else
   exec upd_titles @title_id, @title, @type, @pub_id, @price,
      @advance, @total_sales, @notes, @pubdate, @contract
end
end
/* end of script */
```

在复制节点上：

更新过程遵守以下逻辑规则：

- 对未知 *pub_id* 发出错误。
- 如果不存在 title，则插入一个。
- 按表 3-1 中所示实现正确的更新迁移。

表 3-1：复制节点 (CA) 的迁移策略

原州 / 省	新州 / 省	更新过程需要
CA	CA	通常情况下，更新 publishers 和 titles 表。
CA	NY	删除 publisher 并级联删除与 publisher 关联的所有 title。
NY	CA	插入新的 publisher 和 title （如果有）。

以下脚本创建 upd_publishers 和 upd_titles 两个存储过程，并创建用于控制 upd_publishers 和 upd_titles 两者的执行的管理存储过程 upd_pub_title。

```
-- Execute this script at SF data servers
 -- Creates stored procedure
 create procedure upd_publishers
```

```
 (@pub_id char(4), @pub_name varchar(40),
 @city varchar(20), @state char(2))
 as
 if not exists
    (select * from publishers where pub_id=@pub_id)
    raiserror 20005 "**FATAL ERROR: Unknown publishers id**"
 else
    update publishers set
    pub_name=@pub_name,
    city=@city,
    state=@state
    where pub_id = @pub_id
 end
 /* end of script */
-- Execute this script at SF data servers
 -- Creates stored procedure
 create procedure upd_titles
 (@title_id tid, @title varchar(80), @type char(12),
 @pub_id char(4)=null, @price money=null, @advance money=null,
 @total_sales int=null, @notes varchar(200)=null,
 @pubdate datetime, @contract bit)
 as
 if not exists
    (select * from titles where title_id=@title_id)
    exec ins_titles @title_id, @title, @type, @pub_id,
       @price, @advance, @total_sales, @notes, @pubdate,
       @contract
 else
    update titles set
    title=@title,
    type=@type,
    pub_id=@pub_id,
    price=@price,
    advance=@advance,
    total_sales=@total_sales,
    notes=@notes,
    pubdate=@pubdate,
    contract=@contract
    where title_id = @title_id
 end
 /* end of script */
-- Execute this script at SF data servers
 -- Creates stored procedure
 create procedure upd_pub_title
 (@pub_id char(4), @pub_name varchar(40)=null,
 @city varchar(20)=null, @state char(2),
```

```
    @title_id tid=null, @title varchar(80)=null, @type char(12)=null,
    @price money=null, @advance money=null,
    @total_sales int=null, @notes varchar(200)=null,
    @pubdate datetime=null,@contract bit, @old_state char(2))
    as
       declare @rep_state char (2)
    begin
       select @rep_state=state from publishers
       where pub_id=@pub_id

       if @old_state = @state
       begin
          exec upd_publishers @pub_id, @pub_name,@city, @state
          if @title_id != null
             exec upd_titles @title_id, @title, @type,
                @pub_id, @price,@advance, @total_sales,
                @notes, @pubdate, @contract
       end
       else if @rep_state = @old_state
          begin
             exec del_titles @title_id, @pub_id
             exec del_publishers @pub_id
          end
       else if @rep_state = null
          begin
             exec ins_publishers @pub_id, @pub_name, @city,
                @state
             if @title_id != null
                exec ins_titles @title_id, @title, @type,
                   @pub_id,@price, @advance, @total_sales,
                   @notes, @pubdate,@contract
          end
    end
    /* end of script */
```

函数复制定义

在主 Replication Server 上为 ins_pub_title、del_pub_title 和 upd_pub_title 创建函数复制定义。请注意，对于插入和删除操作，只有 state 是可搜索的；而对于更新操作， old_state 也是可搜索的。

```
-- Execute this script at NY data servers
 -- Creates replication definition del_pub_title
 create function replication definition del_pub_title
 with primary at MIAMI_DS.pubs2
    (@pub_id char(4),
```

```
    @state char(2),
    @title_id varchar(6))
 searchable parameters (@state)
 go
 /* end of script */
-- Execute this script at NY data servers
 -- Creates replication definition ins_pub_title
 create function replication definition ins_pub_title
 with primary at MIAMI_DS.pubs2
    (@pub_id char(4),
    @pub_name varchar(40),
    @city varchar(20),
    @state char(2),
    @title_id varchar(6),
    @title varchar(80),
    @type char(12),
    @price money,
    @advance money,
    @total_sales int,
    @notes varchar(200),
    @pubdate datetime,@contract bit)
 searchable parameters (@state)
 go
 /* end of script */
-- Execute this script at NY data servers
 -- Creates replication definition upd_pub_title
 create function replication definition upd_pub_title
 with primary at MIAMI_DS.pubs2
   (@pub_id char(4),
   @pub_name varchar(40),
   @city varchar(20),
   @state char(2),
   @title_id varchar(6),
   @title varchar(80),
   @type char(12),
   @price money,
   @advance money,
   @total_sales int,
   @notes varchar(200),
   @pubdate datetime,
   @contract bit,
   @old_state char(2))
 searchable parameters (@state, @old_state)
 go
 /* end of script */
```

预订

以下脚本使用非实现方法在复制 Replication Server 上创建预订。如果不需要在复制节点上装载数据，可使用此方法。

要确保正确地迁移预订，必须为 upd_pub_title 创建两个预订。

```
-- Execute this script at SF data servers
 -- Creates subscription for del_pub_title, ins_pub_title,
        and upd_pub_title
 create subscription del_pub_title_sub
 for del_pub_title
 with replicate at SF_DS.pubs2
 where @state='CA'
 without materialization
 go

create subscription ins_pub_title_sub
 for ins_pub_title
 with replicate at SF_DS.pubs2
 where @state='CA'
 without materialization
 go

create subscription upd_pub_title_sub1
 for upd_pub_title
 with replicate at SF_DS.pubs2
 where @state='CA'
 without materialization
 go

create subscription upd_pub_title_sub2
 for upd_pub_title
 with replicate at SF_DS.pubs2
 where @old_state ='CA'
 without materialization
 go
 /* end of script */
```

第 4 章

规划备份和恢复

本章介绍在系统组件出现故障后可用来将主节点和复制节点恢复到一致状态的工具和方法。

防止数据丢失

Replication Server 在分布式数据库系统中运行，这些系统还包含许多其他硬件和软件组件，包括 Adaptive Server 和其他数据服务器、Replication Agent、 LAN、 WAN 和客户端应用程序。

这些组件 （包括 Replication Server）都可能会偶尔出现故障。Replication Server 是一种容错系统，在设计上考虑到了发生故障的可能性。出现故障时，大多数情况下它会在排除故障后继续工作。如果发生的故障需要重新启动 Replication Server 或 Replication Agent，启动过程可以确保复制功能恢复正常，而且既不会丢失数据，也不会导致数据重复。

在复制系统中防止数据丢失与在集中式数据库系统中相同。两种情况下数据都只有一个确定的版本，因此都应当制定完善的计划并投入大量的资源来保护数据。

在复制系统中，如果主数据得到了保护，所有复制数据最终都可以得到恢复。节点仅通过重新创建丢失的复制数据的预订即可恢复这些复制数据。

如果在主节点丢失了复制事务 （例如，将主数据库回退到前一个转储时就会出现这种情况），则主数据和复制数据可能会不一致。

复制系统中提供的备份和恢复方法包括预防措施和恢复措施。本章将介绍这两种措施。

预防措施 包括：

- 热备份
- 硬件数据镜像 （热备份）
- 延长保存间隔
- 协调转储

恢复措施 包括：

- 预订初始化
- 预订调和实用程序 (rs_subcmp)
- 数据库恢复
- 恢复协调转储

预防措施

以下各节介绍了在复制系统中为保护数据可以采取的预防措施。

备份应用程序

通过维护单独的 （备用）主数据副本，可以保护复制系统中的数据。两种可能的备份方法为：

- *热备份应用程序* — 一对 Adaptive Server 或 SQL Server 数据库，其中一个作为另一个的备用副本。客户端应用程序更新活动数据库；Replication Server 通过复制所支持的活动数据库操作来维护备用数据库。
- *硬件数据镜像* — 热备份应用程序的一种形式。数据镜像使用额外的硬件，通过复制对主数据执行的*所有*操作来维护主数据的精确副本。

方法比较

在热备份应用程序中，备用数据库可以在既不中断客户端应用程序，也不丢失任何事务的前提下投入使用。热备份数据库确保在活动数据库上提交的事务在备用数据库上也会提交。当两个数据库都在运行时，活动数据库和备用数据库处于同步状态，而且热备份数据库可以立即投入使用。

此外，由 Replication Server 维护的热备份应用程序：

- 可以用于无法使用数据镜像应用程序的环境，特别是在没有必要硬件的情况下。
- 与一些热备份应用程序相比，容忍临时网络故障的能力更强，因为即使备用数据库已关闭，提交的事务也可以存储在活动数据库上。
- 由于活动数据库不需要校验数据库是否同步，因此可以减少活动数据库的开销。

不过，由 Replication Server 维护的热备份应用程序还具有以下特性：

- 切换到备用数据库时，需要中断某些客户端应用程序。
- 可能尚未在备用数据库中执行活动数据库中最新提交的事务。

热备份

《Replication Server 管理指南》的第 15 章“管理热备份应用程序”中对 Replication Server 的热备份应用程序进行了详细介绍。

注释 Replication Server 版本 12.0 和更高版本支持 Adaptive Server Enterprise 版本 12.0 中提供的 Sybase 故障切换功能。故障切换支持不能替代热备份。热备份应用程序保留数据库的副本，而故障切换支持是通过另一台计算机访问相同的数据库。从 Replication Server 到热备用数据库的连接以相同的方式工作。

有关故障切换支持在 Replication Server 中如何工作的详细信息，请参见《Replication Server 管理指南》的第 19 章 “复制系统恢复”中的“配置复制系统以支持 Sybase 故障切换”和附录 D “Sun Cluster 2.2 上的高可用性”。

硬件数据镜像

若要确保最高的数据可用性，可以对复制系统中最关键的数据进行镜像。镜像通过复制 I/O 操作来维护两个完全相同的数据副本。

如果活动介质出现故障，会立即使备用介质联机。镜像几乎消除了事务丢失的可能性。

下面按优先顺序列出了复制系统中最适合使用镜像的位置：

1 主数据库事务日志

事务日志存储了尚未转储到磁带的事务。如果主事务日志丢失，则必须重新提交事务。

2 主数据库

可以通过重装以前的数据库转储和其后的事务转储来恢复数据库。但是，恢复存储主数据的数据库还需要恢复或重新初始化已在整个企业中复制的数据。对于 OLTP 系统来说，停机时间延长通常是灾难性的。镜像主数据可以防止此类灾难。

3 Replication Server 稳定队列

Replication Server 将事务存储在称作稳定队列的转发磁盘队列中。它从使用 add partition 命令指派给 Replication Server 的磁盘分区分配队列。

存储在稳定队列中的数据是冗余数据，源自主数据库事务日志。但是，如果稳定队列丢失， Replication Server 就无法将事务传递到复制节点。这样，就必须重新初始化复制节点的预订。镜像磁盘分区可以保护稳定队列，并且尽可能缩短复制数据库的潜在停机时间。

4 Replication Server 系统数据库 (RSSD)

如果复制定义、预订、路由，或者函数或错误类等数据自上次备份后已被修改，从 RSSD 故障恢复就可能比较复杂。有关详细恢复信息，请参阅 《Replication Server 管理指南》的第 19 章 “复制系统恢复”。

镜像 RSSD 可以防止系统数据丢失，避免复杂的恢复过程。如果不镜像 RSSD，在执行了任何更改系统数据的 RCL 之后，一定要备份 RSSD。

保存间隔

可以配置从一个 Replication Server 到另一个 Replication Server 的路由，或者配置从 Replication Server 到数据库的连接，使 Replication Server 在将稳定队列消息传递到目标之后，会将这些消息存储一段时间。这段时间就称作保存间隔。

保存间隔在数据源处创建消息备份日志，这样，如果分区故障在保存间隔内得到解决，复制系统就能够容忍该故障。如果目标处的稳定队列出现故障，可以重建这些队列，并让源 Replication Server 重发备份日志中的消息。

有关详细信息，请参阅 《Replication Server 管理指南》。

协调转储

必须通过恢复备份来恢复数据库时，必须通过某种方式使其他节点上受到影响的数据库中的复制数据与主数据保持一致。Replication Server 提供了一种在分布式系统的所有节点上协调数据库转储和事务转储的方法。数据库转储或事务转储是从主数据库启动的。Replication Agent 从日志中检索转储记录并将其提交到 Replication Server，这样，转储请求即可分配到复制节点。这种方法能够保证将数据恢复到已知的一致点。

协调转储只能用于存储主数据或复制数据的数据库，但不能用于同时存储两种数据的数据库。它是从主数据库内启动的。

有关创建协调转储的说明，请参阅 《Replication Server 管理指南》。

恢复措施

以下各节介绍恢复在复制系统中出现组件故障之后丢失的数据的方法。

重新创建预订

数据的主版本是确定的，因此应该按照主版本解决所有的不一致。要从远程节点的故障恢复，一种方法是重新创建预订。对于较小的复制表，这种恢复方法最方便。对于较大的预订和主数据故障，则需要使用其他恢复方法。

预订调和实用程序 (*rs_subcmp*)

rs_subcmp 检测复制表中是否缺失行，是否有孤立行或不一致的行，并纠正不一致之处。与协调装载等影响较大的恢复过程相比，使用 rs_subcmp 命令可能更适合解决较小的不一致问题。有关执行 rs_subcmp 的说明，请参见 《Replication Server 参考手册》。

数据库恢复

当主数据库出现故障时，如果数据库和事务日志未损坏，所有已提交的事务均可以恢复。如果数据库或事务日志已损坏，则必须装载数据库转储和事务转储，使数据库恢复到一个已知状态，然后重新提交在上一次事务转储之后所执行的事务。

以恢复模式运行 Replication Server 和 Replication Agent 时，可以重放重装转储中的事务，确保主数据库中的所有事务均已复制，并且没有重复的事务。《Replication Server 管理指南》中介绍了如何使用转储恢复主数据库。

恢复协调转储

恢复由协调转储进程创建的数据库转储或事务转储可以使主数据和复制数据恢复到以前的一致状态。

有关协调装载过程的详细信息，请参见 《Replication Server 管理指南》的第 19 章 “复制系统恢复”。

第 5 章

Replication Agent 介绍

本章概括介绍 Sybase 复制系统的 Replication Agent 组件，此外还详细介绍了 Adaptive Server 的 RepAgent 以及其他 Replication Agent 产品的工作原理。

Replication Agent 概述

Replication Agent 是一个 Replication Server 客户端，它从主数据库的事务日志中检索信息，并为主 Replication Server 格式化这些信息。RepAgent 是适用于 Replication Server 11.5 和更高版本的 Replication Agent 组件。日志传输管理进程 （Log Transfer Manager，简称 LTM）是适用于 Sybase SQL Server 11.0.x 和更低版本的 Replication Agent 组件。

Replication Agent 检测对主数据所做的更改，而忽略对非主数据所做的更改。通过使用复制控制语言 (Replication Control Language, RCL) 的一个子集，即使用日志传送语言 (Log Transfer Language, LTL)， Replication Agent 将主数据的更改发送到主 Replication Server，主 Replication Server 又将这些信息分配到复制数据库。

Sybase 为其他 Sybase 和非 Sybase 数据服务器（包括 Adaptive Server Anywhere、DB2 Universal Database、Informix、Microsoft SQL Server 和 Oracle）提供 Replication Agent 组件。

Replication Agent for DB2 是适用于基于 OS/390 的 DB2 Universal Database 的 Replication Agent 组件。Sybase Replication Agent 是适用于 DB2 Universal Database （在 UNIX 和 Windows 平台上）、Informix、Microsoft SQL Server 和 Oracle 的 Replication Agent 组件。

RepAgent 是一个 Adaptive Server 线程。 LTM 是一个用于 Sybase SQL Server 数据库的独立线程。 Replication Agent for DB2 是一个驻留在 OS/390 主机上的独立进程。Sybase Replication Agent 是一个驻留在 LAN 服务器主机上的独立应用程序。

Replication Agent 执行以下任务：

1 登录到 Replication Server。

2 发出 connect source 命令，将会话标识为日志传送源，并指定要传送哪个数据库的事务信息。

3 从 Replication Server 获取该数据库的维护用户的名称。RepAgent 将维护用户执行的操作滤出，除非 send_maint_xacts_to_replicate 或 send_warm_standby_xacts 配置参数设置为 true。

4 从 Replication Server 请求该数据库的辅助截断点。此操作返回一个称为源点队列 ID 的值，RepAgent 使用该值在事务日志中查找开始传送事务操作的位置。 Replication Server 已接收到该位置之前的操作。

5 从辅助截断点之后的记录开始，从事务日志中检索记录，并将信息格式化为 LTL 命令。

事务日志

Replication Agent 可以使用以下两种事务日志中的一种：

- 主数据库的本机事务日志
- 由 Replication Agent 创建和维护的事务日志

RepAgent 线程（用于 Adaptive Server）、LTM（用于 Sybase SQL Server 的早期版本）、 Replication Agent for DB2 和 Sybase Replication Agent （用于 UNIX 和 Windows 平台上的 DB2 Universal Database）使用本机事务日志。

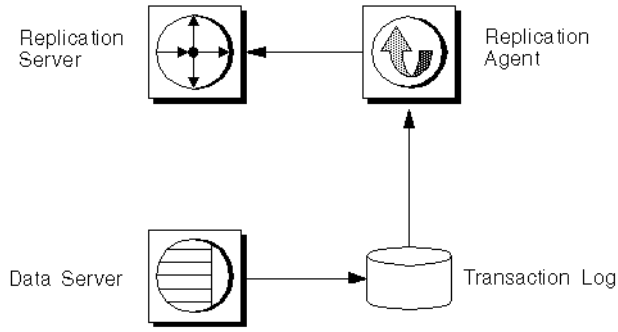
Sybase Replication Agent （用于 Informix、 Microsoft SQL Server 和 Oracle）使用它在主数据库中创建并维护的事务日志。 Sybase Replication Agent 事务日志由主数据库中的数据库对象 （触发器、表和过程）组成。

本机事务日志

由于本机数据库事务日志由主数据库自己维护，并且包含可恢复的永久更新，因此本机数据库事务日志是主数据更改信息的理想来源。如果 Replication Agent 不能访问主数据库的本机事务日志，则 Replication Agent 必须使用另一个事务信息来源。

图 5-1 显示了数据服务器、事务日志、 Replication Agent 和 Replication Server 之间的数据流。

图 5-1：数据库事务日志



Replication Agent 事务日志

某些 Replication Agent 不能访问主数据库的本机事务日志以获取复制事务所需的信息。

Sybase Replication Agent（用于 Informix、Microsoft SQL Server 和 Oracle）使用其自身专有的事务日志来捕获并记录主数据库中要复制的事务。Sybase Replication Agent 生成在主数据库中运行以创建 Replication Agent 事务日志的 SQL 脚本。有关 Replication Agent 事务日志的详细信息，请参见《Sybase Replication Agent 管理指南》。

Replication Agent 产品

Replication Agent 产品使得非 Sybase 数据库服务器也可用作 Sybase 复制系统中的主数据库服务器，从而扩展了 Replication Server 的功能。

Sybase 提供以下用于非 Sybase 数据库的 Replication Agent 产品：

- Replication Agent for DB2
- Sybase Replication Agent

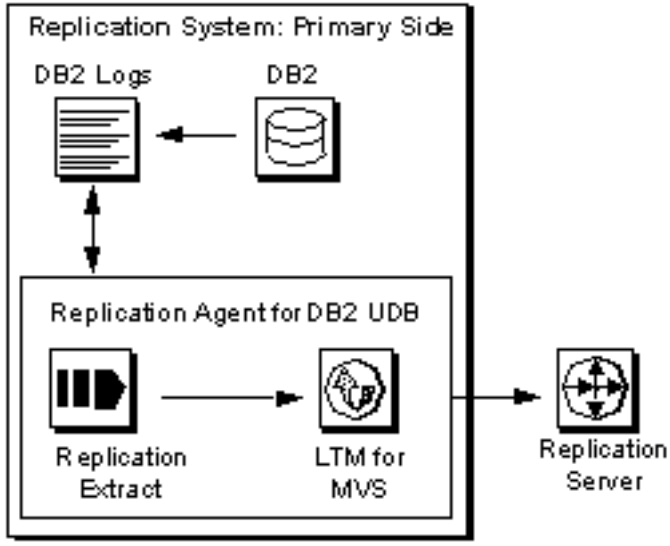
以下各节介绍这些 Replication Agent 产品。

Replication Agent for DB2

Replication Agent for DB2 是一个复制系统组件，它捕获 OS/390 大型机平台上的 DB2 数据库中的数据库事务，并将其发送到 Replication Server。

图 5-2 显示了 Replication Agent for DB2 如何将数据发送到 Replication Server。

图 5-2：DB2 Replication Agent 数据流



Replication Agent for DB2 通过以下方式融入复制系统：

- 对于 Replication Agent for DB2，主数据库是作为 OS/390 中的子系统运行的 DB2。事务日志是 DB2 日志。
- Replication Agent for DB2 提供了称为 Replication Extract 的日志抽取，用于读取 DB2 日志、为标记为要复制的表检索相关的 DB2 活动和存档日志条目。
- LTM for MVS 从 Replication Extract 接收标记为要复制的数据，并使用 TCP/IP 通信协议将这些数据传送到 Replication Server。
- Replication Server 然后将更改应用到复制数据库。

DB2 事务日志

DB2 数据库即时记录对 DB2 表中的行所做的更改。写入事务日志的信息包括更改前后的数据副本。在 DB2 中，这些记录称为撤销和重做记录。系统会为提交和中止操作写入控制记录。这些记录会转换为提交和回退。

DB2 日志由一系列数据集组成。复制抽取使用这些日志数据集来标识 DB2 数据更改。由于 DB2 即时将更改记录写入活动日志，因此复制抽取可以在输入日志记录后立即对其进行处理。

Sybase Replication Agent

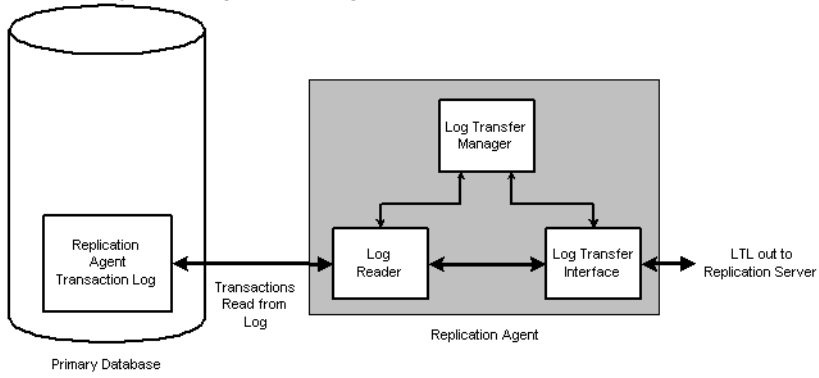
Sybase Replication Agent 是一个复制系统组件，它捕获 DB2 Universal Database （在 UNIX 和 Windows 平台上）、 Informix、 Microsoft SQL Server 或 Oracle 主数据库中的事务，然后将这些事务传送到 Replication Server。

用于 DB2 Universal Database 的 Sybase Replication Agent 使用本机 DB2 事务日志来获取要复制的事务数据。

用于 Informix、 Microsoft SQL Server 和 Oracle 的 Sybase Replication Agent 创建其自己的事务日志来记录要复制的事务 （或过程调用）。Replication Agent 的日志读取器组件读取事务日志，以便从主数据库中检索事务。

从主数据库中检索到事务数据后， Replication Agent 的日志传送接口 (LTI) 组件处理事务和产生的 “更改集”数据，并生成 LTL 输出。Replication Server 使用该输出将事务分配到发出预订的复制数据库。

图 5-3：Sybase Replication Agent 的数据流



Sybase Replication Agent 使用存储在主 Replication Server 的 Replication Server 系统数据库 (RSSD) 中的信息来确定如何处理复制事务以生成最有效的 LTL。

从 Replication Agent 接收到 LTL 后，主 Replication Server 将直接或经由复制 Replication Server 将复制事务发送到复制数据库。该复制 Replication Server 将复制数据转换为复制数据库的本机语言，然后将其发送到复制数据库服务器进行处理。复制数据库在成功地处理了复制事务后，将与主数据库同步。

Sybase Replication Agent 作为独立的应用程序运行。它独立于主数据库服务器、主 Replication Server 和复制系统的任何其他组件。

Sybase Replication Agent 可以与主数据库或复制系统的任何其他组件驻留在同一台主机上，也可以独立于任何其他复制系统组件，驻留在一台单独的计算机上。

Replication Agent 的通信

Sybase Replication Agent 的所有通信均使用 Java 数据库连接 (JDBC) 协议。

Replication Agent 使用 Sybase JDBC 驱动程序 (jConnect for JDBC) 的单个实例来管理与 Open Client/Open Server 应用程序的所有连接，包括主 Replication Server 及其 RSSD 和 Replication Server Manager (RSM)。

对于主数据库服务器， Sybase Replication Agent 连接到主数据库的 JDBC 驱动程序。

在复制事务的同时， Replication Agent 保持与主数据库和主 Replication Server 的连接。此外， Replication Agent 偶尔会连接到主 Replication Server 的 RSSD 以检索复制定义数据。

Java 实施

Sybase Replication Agent 组件使用 Java 编程语言实施。因此，要运行 Sybase Replication Agent，必须在将作为 Replication Agent 主机的计算机上安装 Java Runtime Environment (JRE)。

第 6 章

将数据复制到异构数据服务器

本章介绍将数据复制到非 Adaptive Server 或非 Sybase SQL Server 的异构数据服务器所需要的复制系统组件。

与异构数据服务器的接口

Replication Server 通过将请求提交到数据服务器来更新存储在数据库中的复制数据。 Replication Server 提供了对 Adaptive Server Enterprise 和 Sybase SQL Server 的支持。如果使用非 Adaptive Server 或非 SQL Server 数据服务器来管理数据库，则必须提供一个接口供 Replication Server 使用。

该接口包括：

- 一个 Open Server 网关应用程序，它接收来自 Replication Server 的指令，并将接收的指令应用到数据服务器。
- 一个供 Replication Server 用于登录到该网关的维护帐户。
- 一个与数据库配合使用的函数字符串类。该类中的函数字符串告诉 Replication Server 如何格式化数据服务器的请求。
- 错误类和错误操作指派，用于处理数据服务器通过网关返回到 Replication Server 的错误。

- 每个包含复制数据的数据库中的 rs_lastcommit 表。Replication Server 使用该表来跟踪数据库中已成功提交的事务。
- 一个 rs_get_lastcommit 函数调用，用来检索各个源主数据库的最后一个事务。

Sybase 提供了多种 Open Server 网关应用程序产品，可用于访问承载复制数据库的非 Sybase 数据库服务器。

Replication Server 的异构数据类型支持 (HDS) 功能为所支持的数据库服务器 （DB2、 Informix、 Microsoft SQL Server 和 Oracle）提供了多种组件用于实现此接口。 HDS 功能提供了函数字符串类、错误类和用于在复制数据库中创建所必需的表和过程的脚本。

有关 HDS 功能的详细信息，请参见相应于所用平台的 《Replication Server 管理指南》和 《Replication Server 配置指南》。

Sybase 数据库网关产品

Sybase DirectConnect 数据访问服务器是 Sybase 的中间件构件块，用于连接基于 LAN 的客户端和企业数据源。使用 DirectConnect 数据访问服务器可以简化非 Sybase 复制数据库与 Sybase 复制系统的集成。

使用 DirectConnect 服务器即可直接连接到数据库服务器。通过将 Replication Server 所使用的 Open Client/Server 协议解释为非 Sybase 复制数据库使用的本机通信协议， DirectConnect 服务器可用作 Open Server 网关。

Sybase 为下列数据源和数据库服务器提供了 DirectConnect 数据访问库：

- AS/400
- Informix
- Microsoft SQL Server
- OS/390 (DB2)
- Oracle
- ODBC 可访问数据源

有关 DirectConnect 数据访问服务器的详细信息，请参见相应于所用平台的 《DirectConnect 访问服务用户指南》或 《DirectConnect Server 管理指南》。

Open Server 网关应用程序

网关应用程序使用 Open Server Server-Library/C 例程，因此可以接受来自 Replication Server 的登录，并且可以处理 Replication Server 的请求。网关应用程序将请求转发给包含复制数据的异构数据服务器，并通过 Server-Library 例程将请求的状态返回到 Replication Server。

注释 本手册介绍数据服务器的 Open Server 网关的要求，但不讨论 Open Server 编程。有关 Open Server 编程的详细信息，请参阅 《Open Server Server-Library/C 参考手册》。

在设计用于数据服务器的网关时，必须决定如何编写函数字符串以及网关将如何处理这些函数字符串。例如，如果使用语言函数字符串，网关将需要一个语法分析程序，并且必须定义函数字符串的语法，使网关可以对其进行语法分析。

如果创建 RPC 函数字符串，则可以使用 Open Server 注册过程来进行处理。这比创建语法分析程序要容易。

网关应用程序必须：

- 使用数据服务器供应商的应用程序编程接口 (API) 来访问数据库。
- 进行所有必需的转换，以便将从 Replication Server 函数调用接收的请求重新格式化为数据库 API 所需要的格式。
- 将行结果、消息和错误转换成 Open Server 结果格式，并将其返回到 Replication Server。

必须在 Replication Server 的 Interfaces 文件中定义 Open Server 网关。

使用 create connection 命令来创建从管理数据库的 Replication Server 到网关的连接。

维护用户

Replication Server 以 create connection 中为数据库指定的维护用户的身份登录到网关。网关可以使用相同的登录名登录数据服务器，也可以使用其他登录名。唯一的要求是，该登录名必须具有修改复制数据所需的权限。

函数字符串类

管理数据库的 Replication Server 需要一个函数字符串类。 Replication Server 为 Adaptive Server 提供函数字符串类。利用其异构数据类型支持 (HDS) 功能， Replication Server 版本 12 为 DB2、 Informix、 Microsoft SQL Server 和 Oracle 数据服务器提供了函数字符串类。

如果要复制到 HDS 功能不支持的非 Sybase 数据服务器，则必须为该数据服务器创建一个函数字符串类。您可以：

- 创建一个函数字符串类，从系统提供的类继承函数字符串，或
- 自己创建所有的函数字符串。

Replication Server 向网关发送一条命令，该命令是它通过将运行时值映射到为函数提供的函数字符串构建的。根据编写函数字符串的方式和数据服务器的要求，网关可以直接将命令传递到数据服务器，或在将请求发送到数据服务器之前对命令做某些处理。

有关数据库网关可能需要处理的 Replication Server 系统函数的列表，请参阅 《Replication Server 管理指南》的第 14 章 “自定义数据库操作”。

使用继承创建函数字符串类

Replication Server 允许通过使用函数字符串继承机制创建类之间的关系，在函数字符串类之间共享函数字符串定义。

系统提供的类 rs_default_function_class 和 rs_db2_function_class 可以作为派生类的父类，这些派生类继承父类的函数字符串。若要针对数据服务器自定义某些函数字符串，同时又要保留父类的所有其他函数字符串，可以创建派生类。

使用 create function string class 命令可以创建继承自父类 rs_default_class 或 rs_db2_function_class 的派生类。只应在需要时才创建自定义函数字符串。

注释 rs_db2_function_class 不支持复制 text 或 image 数据。若要针对 DB2 数据库启用 text 或 image 数据的复制，必须使用 RPC 方法通过网关自定义 rs_writetext 函数。有关 rs_writetext 的信息，请参阅第 99 页的 “创建独立的函数字符串类”。

有关函数字符串继承的详细说明，请参见 《Replication Server 管理指南》的第 14 章 “自定义数据库操作”。

创建独立的函数字符串类

如果使用的类不从系统提供的类继承，则必须自己创建所有函数字符串，而且只要创建了新表或函数复制定义，就必须添加新的函数字符串。

使用 create function string class 命令可以创建新的函数字符串类，然后为所有具有函数字符串类作用域的函数创建函数字符串。

必须为数据库中复制的每个表创建 rs_insert、 rs_update 和 rs_delete 函数字符串。

如果要复制数据类型为 text 或 image 的列，则必须为每个 text 或 image 列创建 rs_datarow_for_writetext、 rs_get_textptr、 rs_textptr_init 和 rs_writetext 函数字符串。函数字符串名必须是表复制定义的 text 或 image 列名。

只有数据库中包含复制定义的主数据时，才需要 rs_select 和 rs_select_with_lock 函数字符串。

错误类

错误类确定 Replication Server 如何处理网关返回的错误。必须使用 create error class 命令为网关创建错误类。

使用 assign action 命令可以告诉 Replication Server 如何响应网关返回的错误。表 6-1 列出了可能的操作。

表 6-1：数据服务器错误的 Replication Server 操作

操作	说明
ignore	假设命令成功执行并且没有要处理的错误或警告情况。此操作可以用于表示成功执行的返回状态。
warn	记录一条警告消息，但不会回退该事务或中断执行。
retry_log	回退该事务并重试。使用 configure connection 可以设置尝试进行重试的次数。如果重试后再次出现该错误，将把该事务写入例外日志，并继续执行下一个事务。
log	回退当前事务并将其记录到例外日志中。然后继续执行下一个事务。
retry_stop	回退该事务并重试。使用 configure connection 可以设置尝试进行重试的次数。如果重试后再次出现该错误，将挂起数据库的复制。
stop_replication	回退当前事务并挂起数据库的复制。这与使用 suspend connection 命令等效，并且是缺省操作。 由于此操作将停止数据库的所有复制活动，因此应找出不需要关闭数据库连接就能够处理的数据服务器错误，并为这些错误指派其他操作。

缺省错误操作是 stop_replication。如果没有为错误指派其他操作，Replication Server 将断开与网关的连接。

有关 create error class 和 assign action 的详细信息，请参见 《Replication Server 参考手册》。

rs_lastcommit 表

rs_lastcommit 表中的每一行都标识最近提交的从主数据库分配到数据库的事务。 Replication Server 使用这些信息来确保所有事务均已分配。

rs_lastcommit 表应由 rs_commit 函数字符串在提交事务之前更新。这样可以确保用 Replication Server 在数据库中提交的每个事务更新了该表。

Replication Server 以数据库维护用户的身份维护 rs_lastcommit 表。必须确保维护用户对该表具有所有所需的权限。

表 6-2 列出了 rs_lastcommit 表中的列。

表 6-2：rs_lastcommit 表结构

列名	数据类型	说明
origin	int	由 Replication Server 指派的一个整数，用于唯一地标识事务源自哪个数据库
origin_qid	binary(36)	事务中提交记录的源点队列 ID
secondary_qid	binary(36)	预订实现期间使用的稳定队列的队列 ID
origin_time	datetime	事务发生的时间
dest_commit_time	datetime	在目标点提交事务的时间

origin_time 和 dest_commit_time 列不是必需的。

origin 列是该表的唯一键。对于数据复制到此数据库中的每个主数据库，该表中都有一行。

如果对数据库使用了协调转储，应使用 rs_dumpdb 和 rs_dumptran 函数字符串来更新 rs_lastcommit 表。

对于 Adaptive Server 和 SQL Server 数据库， rs_commit、 rs_dumpdb 和 rs_dumptran 函数字符串执行名为 rs_update_lastcommit 的存储过程来更新 rs_lastcommit 表。下面是该存储过程的文本：

```
/* Create a procedure to update the
 ** rs_lastcommit table. */
 create procedure rs_update_lastcommit
     @origin         int,
     @origin_qid     binary(36),
     @secondary_qid  binary(36),
     @origin_time    datetime
 as
     update rs_lastcommit
         set origin_qid = @origin_qid,
         secondary_qid = @secondary_qid,
         origin_time = @origin_time,
         commit_time = getdate()
     where origin = @origin
     if (@@rowcount = 0)
     begin
         insert rs_lastcommit (origin,
             origin_qid, secondary_qid,
             origin_time, commit_time,
             pad1, pad2, pad3, pad4,
             pad5, pad6, pad7, pad8)
```

```
            values (@origin, @origin_qid,
                @secondary_qid,@origin_time,
                getdate(), 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00)
        end
    go
```

有关详细信息，请参见 《Replication Server 参考手册》中 rs_commit、rs_dumpdb 和 rs_dumptran 三个函数的参考页。

rs_get_lastcommit 函数

Replication Server 向网关发送 rs_get_lastcommit 函数调用，从各个源主服务器中检索数据库中提交的最后一个事务。网关应返回表 6-2 中介绍的前三列。

rs_get_lastcommit 的函数字符串可以执行一条简单的 select 语句：

```
select origin, origin_qid, secondary_qid
     from rs_lastcommit
```

对异构数据库使用 Sybase Central

您可以使用 Sybase Central 的 RSM 插件来管理包含非 Sybase 数据服务器的复制系统。下列要求和限制适用：

- 必须为每个非 Sybase 数据服务器配置一个 Open Server 网关应用程序，并完成第 95 页的 “与异构数据服务器的接口”中介绍的接口。
- Replication Server 树的逻辑视图在 Sybase Central 中不显示，但是您可以在 Replication Server 和数据库的属性表中查看有关复制定义、预订和发布的、数据特定的信息。
- 您不能使用 Sybase Central 来创建复制定义或发布，但可以使用 Sybase Central 为那些使用 RCL 命令和实用程序 （例如， isql）创建的复制定义或发布创建预订。

第 7 章

国际化复制设计注意事项

本章讨论有关在一种国际环境中设置复制系统的问题。

设计国际化复制系统

Replication Server 和 Replication Server Manager (RSM)，以及可用于管理复制系统的 Sybase Central 图形化插件，都支持国际环境。它们提供以下功能：

- 消息本地化为多种语言。
- 支持所有 Sybase 支持的字符集，并且可在不同的 Replication Server 节点之间进行字符集转换。
- 支持非二进制排序顺序。

当您为一个国际环境设计复制系统时，应当了解语言、字符集和排序顺序设置对系统的影响，这一点非常重要。 Replication Server 和 RSM Server 为配置这些设置提供了很大的灵活性。但是，如果不按照本章中提供的配置准则进行操作，这种灵活性可能会导致意外或不利的结果。

消息语言

您可将 Replication Server 和 RSM Server 配置为以多种语言将消息输出到错误日志或客户端。所选语言必须与所选字符集兼容。缺省语言为英语；它与所有 Sybase 字符集都兼容。有关所支持语言的列表，请参见相应于所用平台的发行公告。

复制系统中的各个服务器程序 （包括 Replication Server、 Adaptive Server、其他数据服务器、 RSM Server 和 LTM）都会以所配置的语言将消息写入其错误日志。但是，是否以客户端的语言将消息发送到客户端取决于服务器。

例如，Adaptive Server 会检查其客户端 (Replication Server) 的语言设置，然后以该语言返回消息。 RepAgent （Adaptive Server 的一个线程）也以客户端的语言返回消息。

但是，Replication Server、RSM Server 和 LTM 不检查客户端的语言，而是以它们自己的语言将消息返回到客户端。因此，如果服务器所配置的语言不同，则错误日志中可能会包含不同语言的消息。

注释 为了避免混合语言错误日志可能会导致的混淆，对于给定节点上的所有服务器和客户端，应配置相同的语言设置。

❖ 更改 Replication Server 消息语言

使用以下过程可以更改 Replication Server 的消息语言。

注释 由于 RepAgent 会自动以 Replication Server 的语言返回消息，因此不必为 RepAgent 设置语言参数。

---

1 关闭 Replication Server。

   如果 Replication Server 使用 LTM 作为 Replication Agent，请关闭 LTM。

2 使用文本编辑器在 Replication Server 配置文件中更改 RS_language 的值。

   如果在使用 LTM，请使用文本编辑器在 LTM 配置文件中更改 LTM_language 的值。

3 重新启动 Replication Server 并重新启动 LTM （如果适用）。

字符集

Replication Server 和 RSM Server 支持所有 Sybase 支持的字符集，并且在主节点和复制节点之间对数据和标识符执行字符集转换。字符集必须兼容，字符集转换才会成功。例如，不能将单字节字符集数据转换为多字节字符集。有关字符集兼容性的详细信息，请参见 《Open Client Client-Library 参考手册》。

对于给定的服务器，对字符集的选择受以下因素的影响：该服务器支持的语言，运行该服务器的硬件和操作系统，以及要与其进行交互的系统。

Sybase 支持的字符集具有以下特点：

- 它们都是 7 位 ASCII 字符集的超集。
- 有些字符集相互之间完全不兼容，也就是说，除 7 位 ASCII 字符外，它们没有任何公共字符。
- 在兼容的字符集之间，有些字符并不是在两个字符集中都存在，也就是说，没有任何两个字符集具有完全相同的字符。

若要更改缺省字符集，请按照第 115 页的 “更改字符集和排序顺序”中提供的过程操作。虽然更改缺省字符集只需在 Replication Server 配置文件中更改 RS_charset 参数的值，但请按照该过程中的步骤进行操作以确保该更改不会损坏复制数据。

字符集转换

字符集转换在目标 Replication Server 上进行。为 Replication Server Interface (RSI) 打包的每条消息都包含源 Replication Server 的字符集的名称。目标 Replication Server 使用该信息将字符数据和标识符转换为它自己的字符集。

尝试转换字符集时， Replication Server 将检查字符集是否兼容。如果字符集不兼容，将不会执行转换。如果字符集兼容，但遇到了一个或多个并非两个字符集中都有的字符，将使用“?”（问号）替换未识别的字符。

在 Replication Server 中，是将字符替换为 “?”还是引发字符集转换异常由环境确定。例如，如果 Replication Server 检测到要复制的字符不兼容，它会用 “?”替换该字符；但如果它在转换配置文件参数时检测到字符不兼容，它会输出一条错误消息并关闭。

可以使用 dsi_charset_convert 配置参数来指定 Replication Server 是否尝试进行字符集转换。有关该参数的说明，请参见 《Replication Server 参考手册》中的 configure connection 命令。

有关 Replication Server 在预订实现、解析和调和过程中如何处理字符集转换的详细信息，请参见第 109 页的 “预订”。

Unicode UTF-8 和 UTF-16 支持

Replication Server 支持缺省字符集 Unicode UTF-8 和两种 Unicode 数据类型：unichar 和 univarchar。这两种数据类型使用 UTF-16 编码。利用 Unicode，可以在同一台服务器上混用不同语言组中的不同语言。有关使用 Unicode 字符集的详细信息，请参见《Adaptive Server 管理指南》。

UTF-8

UTF-8（UCS 转化格式， 8 位形式）是一个国际字符集，它支持世界上 650 多种语言。 UTF-8 使用 8 位序列，是 Unicode 标准的一种变长编码格式。 UTF-8 支持所有 ASCII 代码值（从 0 到 127）以及其他许多语言中的值。每个非替换代理代码值用 1 个、 2 个或 3 个字节表示。基本多语言平面 (Basic Multilingual Plane, BMP) 之外的代码值需要替换代理对，占用 4 个字节。

Adaptive Server、 Oracle、 IBM UDB、 Microsoft SQL Server 和 Informix 数据服务器都支持 UTF-8。

UTF-16

UTF-16 （UCS 转化格式， 16 位形式）是 Unicode 标准的一种定长编码格式，它使用 16 位序列，其中每个字符的长度均为 2 个字节。而在 UTF-8 中，BMP 之外的代码值用替换代理对表示，而且需要 4 个字节。 Replication Server 和 Adaptive Server 都使用 UTF-16 对 unichar 和 univarchar 值进行编码。

要求

若要使用 Unicode UTF-8 缺省字符集或 unichar 和 univarchar 数据类型，运行的必须是 Replication Server 版本 12.5 或更高版本，而且必须将节点版本设置为 12.5 或更高。

RSM Server 问题

总体来说，RSM Server 和 Replication Server 支持国际化的方式相同。本节介绍 RSM Server 的其他字符集问题。

读取错误日志文件

可将 RSM Server 配置为读取 Replication Server 和 Adaptive Server （或 SQL Server）错误日志文件。但是， RSM Server 只能直接监控具有相同字符集的服务器的日志文件。

若要监控字符集与 RSM Server 不相同的服务器 （例如，服务器日志文件以 SJIS 输出，而 RSM Server 配置为使用 EUCJIS），则必须通过另一个 RSM Server 间接对该服务器进行监控。设置一个本地 RSM Server，然后将其配置为使用其他服务器的字符集。这样，您的 RSM Server 就可以通过该本地 RSM Server 来读取日志文件了。

有关详细信息，请参阅 Replication Server 联机帮助中的“Rep Server：在本地监控错误日志”。

更改 RSM Server 的字符集

RSM Server 假定所有配置文件都是用 RSM Server 的字符集写入的。（对于 Replication Server，可以使用一种字符集写入配置文件，而将 Replication Server 配置为使用另一种字符集运行。）如果更改了 RSM Server 的字符集，则必须运行 rsmgen (UNIX) 或 RSMSetup (Windows NT) 重新以新的字符集写入配置参数。不要直接在配置文件中更改值。

有关运行 rsmgen 和 RSMSetup 的信息，请参阅相应于所用平台的《Replication Server 配置指南》。

RSM Client 问题

Sybase Central 和 Sybase Replication Server 插件 (RSM Client) 使用本地 *$SYBASE* 安装的缺省地区（语言和字符集）设置来连接到 RSM Server。不能直接配置 Sybase Central 和 RSM Client 地区设置。有关配置本地 *$SYBASE* 地区设置的详细信息，请参阅《Open Client/Server 配置指南》。

我们强烈建议安装 RSM Client 的计算机与 RSM Server 使用相同的语言和字符集。否则，从 RSM Server 返回的将是另一种语言的消息，或者，如果字符集不兼容，返回的消息将无法看懂。

有关使用字符集的指导信息

设置复制系统时，我们强烈建议给定 Replication Server 节点上的所有服务器都使用相同的字符集。我们还建议复制系统中的所有 Replication Server 都使用兼容的字符集。

按照以下指导信息进行操作，可以尽量避免字符集转换问题：

- 所有数据服务器、数据和对象名称均使用 7 位 ASCII 字符（如果可能）。

  如果数据和对象名称都使用 7 位 ASCII 字符，或者如果所有数据服务器和 Replication Server 都使用相同的字符集，则字符集转换不会有任何问题。

- 如果需要在单字节服务器和多字节服务器之间复制数据，则应将字符数据和对象名称限制为使用 7 位 ASCII 字符以避免损坏。否则，可能会出现问题。例如， Replication Server 不将服务器名称限制为使用 7 位 ASCII 字符，但 Adaptive Server 或 Connectivity Libraries 可能会限制。
- 在具有不同但兼容的字符集 （例如， ISO_1 和 CP850）的服务器之间进行复制时，应确保对象名称和字符数据中没有任何只存在于其中一个字符集中的 8 位字符。

我们建议 RSM Server 和由 RSM Server 管理的所有服务器都使用相同或兼容的字符集。如果 RSM Server 域中的服务器所使用的字符集不同，用于配置 RSM Server 的字符集应该是它所管理的服务器的字符集的超集。例如，如果 RSM Server 域中的服务器配置为 ISO_1 和 SJIS，则应使用 SJIS 来配置 RSM Server，因为该字符集是 ISO_1 的超集。

排序顺序

Replication Server 和 RSM Server 使用排序顺序（或称为排序序列）来确定如果对字符数据和标识符进行比较和排序。 Replication Server 和 RSM Server 支持 Sybase 支持的所有排序顺序，包括非二进制排序顺序。若要正确对欧洲语言的字符数据和标识符进行排序，必须使用二进制排序顺序。

若要更改缺省或 Unicode 排序顺序，请按照第 115 页的 “更改字符集和排序顺序”中提供的过程进行操作。虽然更改缺省排序顺序只需在 Replication Server 的配置文件中更改 RS_sortorder 参数的值，但请按照该过程中的步骤进行操作以确保该更改不会损坏复制数据。

注释 若要确保数据和标识符的排序方式在整个复制系统中均一致，*应使用相同的排序顺序来配置 RSM Server 和所有 Replication Server 组件*。

预订

预订涉及在以下位置对数据进行比较：

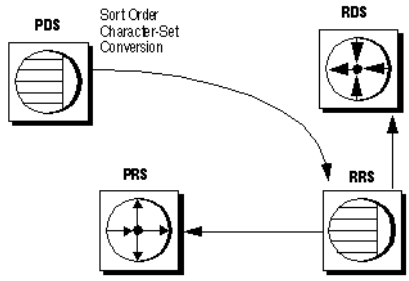
- 实现期间的主数据服务器
- 解析期间的主 Replication Server
- 初始化和删除期间的复制 Replication Server
- 删除期间的复制数据服务器

排序顺序和字符集转换在处理预订的过程中起着重要的作用。排序顺序和字符集转换必须一致，预订才有效。

预订实现

图 7-1 显示了预订实现期间的典型消息流。

图 7-1：预订实现



在预订实现期间：

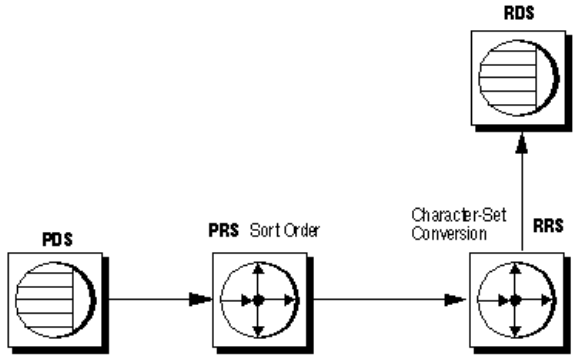
- 复制 Replication Server 登录到主数据服务器并发出 select 语句以检索主数据。
- 主数据服务器将所有字符数据转换为复制 Replication Server 的字符集。如果复制 Replication Server 的字符集不同，必须在主数据服务器上安装该字符集。
- 复制 Replication Server 在复制数据服务器上插入数据。

注释 在批量实现中，预订由复制系统之外用户选择的机制初始化。因此，必须确保主数据服务器上的初始数据选择使用正确的排序顺序，并确保根据需要将字符数据转换为复制数据服务器的字符集。

预订解析

图 7-2 显示了预订的正常生命周期中的典型消息流。

图 7-2：预订解析



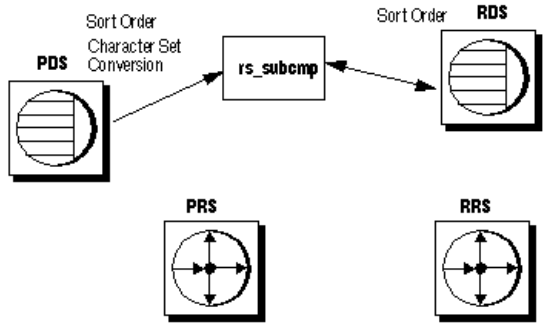
在预订解析期间：

- Adaptive Server 的 RepAgent 线程扫描日志来搜索更新。
- 主 Replication Server 使用其排序顺序来确定哪些行符合预订条件。主 Replication Server 还会将主 Replication Server 的字符集的名称添加到 RSI 消息中。
- 复制 Replication Server 将数据转换为其字符集 （如果必要），并将更新应用到复制数据服务器。

预订调和

图 7-3 显示了预订调和期间的 rs_subcmp 进程。

图 7-3：预订调和



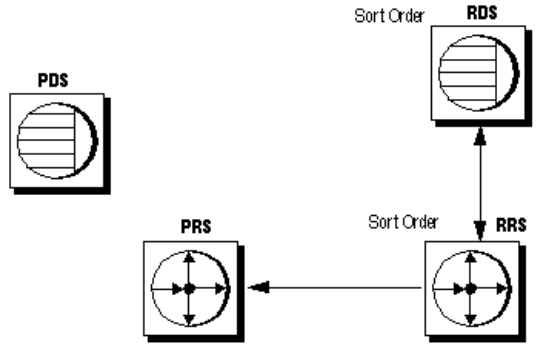
在预订调和期间：

- rs_subcmp 使用复制数据服务器的字符集连接到主数据服务器和复制数据服务器。
- 主数据服务器将所有字符数据转换为复制数据服务器的字符集（所有 rs_subcmp 操作均使用复制数据服务器的字符集执行）。如果复制数据服务器的字符集与主数据服务器的字符集不同，则必须在主数据服务器上安装该字符集。
- rs_subcmp 将一条 select 语句发送到两台数据服务器。各台数据服务器的排序顺序必须相同，该进程才会产生合理的结果。

取消实现

图 7-4 显示了预订取消实现期间的典型消息流。

图 7-4：预订取消实现



在预订取消实现期间：

- 复制 Replication Server 从复制数据服务器中选择数据来构造取消实现队列。复制数据服务器使用其排序顺序来选择行。
- 复制 Replication Server 使用其排序顺序来排除属于其他预订的行。
- 复制 Replication Server 从复制数据库中删除剩余的行。

Unicode 排序顺序

Unicode 排序顺序与 Replication Server 排序顺序不同，必须单独设置。若要设置 Unicode 排序顺序，请使用文本编辑器将下面一行添加到 Replication Server 的配置文件中：

```
RS_unicode_sort_order=unicode_sort_order
```

unicode_sort_order 可以是表 7-1 中列出的 Sybase 所支持的 Unicode 排序顺序中的任何一种。缺省值为 “binary”。

应确保在更改 Unicode 排序顺序之前挂起与数据服务器的连接并关闭 Replication Server。

若要更改当前排序顺序，请使用第 115 页的 “更改字符集和排序顺序”中介绍的过程。

表 7-1：支持的 Unicode 排序顺序

名称	说明
defaultml	UTF-16 缺省 ML
altnoacc	CP850 备选：没有变音
altdict	cp850 备选：小写优先
altnocsp	CP850 备选：没有大小写优先级
scandict	CP850 斯堪的纳维亚语字典
scannocp	CP850 斯堪的纳维亚语，没有大小写优先级
binary	UTF-16 二进制
dict	Latin-1 英语字典
nocase	Latin-1 英语，没有大小写
nocasep	Latin-1 英语，没有大小写优先级
noaccent	Latin-1 英语，没有变音
espdict	Latin-1 西班牙语字典
espnocs	Latin-1 西班牙语，没有大小写
espnoac	Latin-1 西班牙语，没有变音
rusnocs	8859-5 俄语，没有大小写
cyrnocs	8859-5 古斯拉夫语，没有大小写
elldict	8859-7 希腊语字典
hundict	8859-2 匈牙利语字典
hunnoac	8859-2 匈牙利语，没有变音
hunnocs	8859-2 匈牙利语，没有大小写
turknoac	8859-9 土耳其语，没有变音
turknocs	8859-9 土耳其语，没有大小写
thaidict	CP874 泰语字典
utf8bin	符合 UTF-8 的 UTF-16 排序

也可以为 rs_subcmp 指定 Unicode 排序顺序。请参见《Replication Server 参考手册》中的 “rs_subcmp”。

更改字符集和排序顺序

如果更改了 Adaptive Server 的字符集或排序顺序，则必须同时更改以下服务器的字符集或排序顺序：

- 管理服务器复制的每个 Replication Server
- 驻留在单独的 Adaptive Server 上的每个关联 RSSD
- 每个关联 LTM （如果适用）

如果更改了 Adaptive Server 的排序顺序，则必须同时更改复制 Replication Server 和复制数据服务器的排序顺序，以确保对预订的处理一致。

在更改了字符集或排序顺序后，预订语义可能会改变。更改排序顺序的后果非常显著。假定预订中包含子句 “where last_name = MacGregor”。例如，如果将排序顺序从 “dict”更改为 “binary”，则 “MacGregor”不再符合排序条件。

同步主数据库和复制数据库

在更改了字符集或排序顺序后，必须确保对主数据库和复制数据库重新进行同步。 Sybase 建议您使用以下方法之一：

- 在更改了字符集或排序顺序*之后*，使用 rs_subcmp，或
- 在更改字符集或排序顺序*之前*，清除所有预订，然后，在更改了字符集或排序顺序*之后*，重新实现所有预订。

注释 如果有任何预订包含字符子句，应使用清除再重新实现的方法。只有这个方法能够确保对带有字符子句的预订重新进行同步。

❖ **更改字符集或排序顺序**

在修改字符集或排序顺序之前，所有源自 Adaptive Server 的复制事务必须均到达复制数据服务器。除了更改排序顺序或字符集之外，此过程还确保更改字符集或排序顺序不会导致任何数据损坏。

1 找出所有与主 Adaptive Server 相关联的 Replication Server、Adaptive Server 和 LTM （如果适用），包括关联 Replication Server 的所有 RSSD。

注释 *如果要更改字符集：*查找 Replication Server 域中所有 Adaptive Server 的字符集，确定是否必须同时更改这些字符集。对于同一个域内的服务器， Sybase 支持备选字符集，但对用户的暗示非常重要。

*如果要更改排序顺序：*Sybase 建议 Replication Server 域中的所有数据服务器都使用相同的排序顺序。这样可以确保整个复制系统中对数据和标识符的排序一致。

---

2 停顿所有主更新，并确保 Replication Server 已对这些更新进行了处理。

注释 如果要更改字符集，应确保有足够的空事务来跨越 Adaptive Server 中的页。这样可以确保在清空了 Adaptive Server 事务日志（请参见第 9 步）后，没有任何数据仍使用旧的字符集。

---

3 停顿所有关联 Replication Server。

4 关闭所有关联的 Replication Server 和 RepAgent （或 LTM）。

5 在 Replication Server 和 LTM （如果适用）的配置文件中更改字符集和 / 或排序顺序。

6 按照 Adaptive Server 过程进行操作，更改关联的各个 Adaptive Server 的缺省字符集和 / 或排序顺序。请参见 《Adaptive Server 管理指南》中的 “配置字符集、排序顺序和语言”。

7 关闭所有关联的 Adaptive Server。

8 除非可以保证主数据库和复制数据库不会有任何活动，否则，应以单用户模式启动关联的 Adaptive Server。

9 删除每个关联的 Adaptive Server 的辅助截断点。这一步使 Adaptive Server 能够截断 RepAgent（或 LTM）尚未传送到 Replication Server 的日志记录。从 Adaptive Server 输入以下命令：

```
dbcc settrunc('ltm', 'ignore')
```

10　截断事务日志。从 Adaptive Server 输入以下命令：

```
dump transaction db_name with truncate_only
```

11　重新设置辅助截断点。从 Adaptive Server 输入以下命令：

```
dbcc settrunc('ltm', 'valid')
```

12　将主数据库的定位符值重新设置为零。这一步指示 Replication Server 从 Adaptive Server 获取新的辅助截断点，并指示它将定位符设置为该值。从 Adaptive Server 输入以下命令：

```
rs_zeroltm data_server, db_name
```

13　以正常模式关闭并重新启动 Adaptive Server。

14　重新启动关联的 Replication Server。

15　通过恢复日志传送，使 RepAgent （或 LTM）能够重新连接到 Replication Server。从 Replication Server 输入以下命令：

```
resume log transfer from data_server.db_name
```

16　启动 RepAgent （或 LTM）。

17　重新启动复制。

如果更改字符集改变了字符宽度

如果字符集更改涉及字符宽度的更改，则必须重装由 Replication Server 控制的所有数据库的存储过程消息。存储过程消息位于 *$SYBASE/$SYBASE_REP/scripts/rsspmsg1.sql* 和 *rsspmsg2.sql* 中。

从单字节字符集更改为多字节字符集时：

```
isql -Uuser_name -Ppassword -Srssd_name -Jeucjis
       <$SYBASE/$SYBASE_REP/scripts/rsspmsg2.sql
```

从多字节字符集更改为单字节字符集时：

```
isql -Uuser_name -Ppassword -Srssd_name -Jiso_1
       <$SYBASE/$SYBASE_REP/scripts/rsspmsg1.sql
```

更改为 UTF-8 字符集时，应同时安装 *rsspmsg1.sql* 和 *rsspmsg2.sql*

总结

- 可将 Replication Server 和 RSM Server 配置为以英语、法语、德语和日语将消息输出到错误日志和客户端。英语是缺省语言。
- Sybase 建议使用相同的语言配置位于复制节点的所有服务器。
- Replication Server 和 RSM Server 支持所有 Sybase 所支持的字符集和排序顺序，包括非二进制排序顺序和 Unicode UTF-8 字符集。
- Replication Server 可在主 Replication Server 和复制 Replication Server 之间以及主数据库和复制数据库之间，对数据和标识符执行字符集转换。
- Sybase 建议让复制节点上的所有服务器都使用相同的字符集，让您的系统中的所有 Replication Server 和 RSM Server 都使用兼容的字符集。
- 排序顺序在处理预订的过程中起着重要的作用。排序顺序必须在所有位置都保持一致，预订才有效。

附 录 A

容量规划

本附录包含帮助您规划复制系统所需的 CPU、内存、磁盘和网络资源的信息。

要求概述

复制系统由 Replication Server、Replication Agent（RepAgent、LTM 或其他 Replication Agent）及数据服务器构成。

警告！ 随着 Adaptive Server 版本的变化， Replication Server 的容量规划也可能因新的 Adaptive Server 事务日志空间要求而发生变化。

本章中的所有容量规划均假定 Adaptive Server 使用的页大小为 2KB。如果使用较大的页大小，则必须重新计算所需的空间占用量，以适应 Adaptive Server 较大的页大小。

Replication Server 要求

每个 Replication Server 的最低要求如下：

- 如果存在源自此 Replication Server 的路由，则 Replication Server 系统数据库 (RSSD) 需要一个 Replication Agent。
- 至少需要一个 20MB 的原始分区或操作系统文件用于稳定队列。
- 用于 RSSD 的 Adaptive Server，并且必须具有：
  - 至少 10MB 的可用设备空间用于 RSSD
  - 另外 10MB 可用设备空间用于 SRSSD 日志
- 除 Adaptive Server 用户所需的用户连接数之外，还需要 20 个用于 RSSD 的用户连接。Replication Server 启动时，同时会有多个线程尝试读取 RSSD。为适应这一要求，应增加 20 个用户连接。
- 复制系统中的每个 RSM 都需要一个 RSSD 用户连接，并且对于每个数据服务器，每个 RSM 进程都需要一个用户连接。
- 每个包含主数据的数据库都需要两个用户连接。
- 每个仅复制数据库都需要一个用户连接。
- 至少 8MB RAM 用于 Replication Server 可执行程序以及数据和堆栈内存。
- 每个 LTM 或其他 Replication Agent 至少需要 5MB RAM。（RepAgent 是 Adaptive Server 线程，并不需要占用任何 Replication Server 内存。）

Replication Server 对主数据库的要求

对于管理的每个主数据库， Replication Server 需要：

- RepAgent 线程用于 Adaptive Server 数据库， LTM 用于 Sybase SQL Server 数据库，或其他 Replication Agent 用于非 Sybase 数据库
- 一个入站稳定队列
- 一个出站稳定队列
- 一个或两个数据服务器连接 （一个用于 LTM，一个用于 DSI）

有关 Adaptive Server 兼容性要求的详细信息，请参见发行公告。

注释 如果在非 Sybase 数据源中使用 Replication Agent，请参见相应的 Replication Agent 文档，了解有关兼容性要求的信息。

Replication Server 对复制数据库的要求

对于管理的每个复制数据库， Replication Server 需要：

- 一个出站稳定队列
- 一个数据服务器连接 (DSI)

Replication Server 对路由的要求

对每个指向另一个 Replication Server 的直接路由，Replication Server 需要：

- 一个出站稳定队列

数据量 （队列磁盘空间要求）

在估计复制系统所需的资源量时，最重要的因素是复制的数据量和数据的更新速率。

要计算数据量，您需要了解有关复制系统的以下事项：

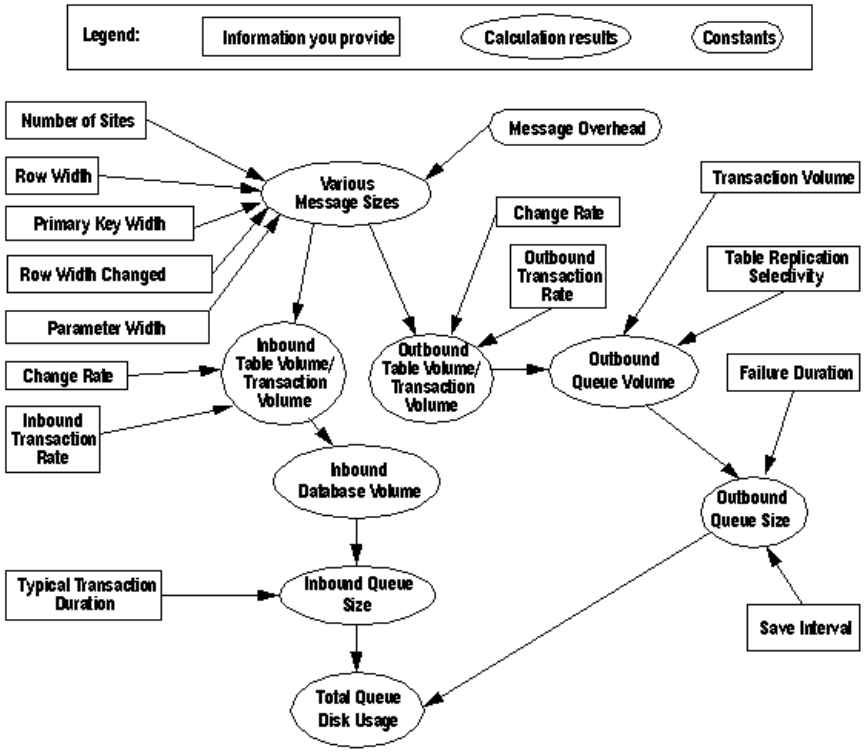
- 节点数
- 被复制表中行的宽度
- 被复制函数中参数的宽度
- 每秒的修改次数
- 典型事务的持续时间
- 被复制表的复制选择性 （该表中通过队列复制的部分）
- 目标节点不可用时需要队列将事务保留多长时间

以下各节提供了计算队列大小要求的公式。也可以使用 RSSD 中的 rs_fillcaptable 和 rs_capacity 过程计算队列大小的估计值。有关这些存储过程的信息，请参见 《Replication Server 参考手册》。

磁盘队列大小计算的概述

图 A-1 图示了计算数据量和队列磁盘使用率的顺序和流程。

图 A-1：计算队列磁盘使用率



消息大小

Replication Server 使用 ASCII 格式的消息分发数据库修改信息。大多数复制系统消息对应于删除、插入、更新和函数执行等操作。

每个表对插入或删除操作使用一种消息大小，对更新操作使用另一种消息大小。每个函数都有各自的消息大小。消息大小用字节数表示。

根据消息是在入站队列中还是在出站队列中，同一类修改（插入、删除或更新）的消息大小可能有所不同。

可以使用以下所示的公式为表更新、插入和删除操作、为函数以及开始 / 提交对计算消息的字节数。有关公式构成的说明，请参见第 124 页的 “公式构成”。

表更新

以下公式提供对消息大小上限的粗略估计：

```
InboundMsgSizeupdate = InboundMsgOverhead +
  ColOverhead + (RowWidth*2)
OutboundMsgSizeupdate = OutboundMsgOverhead +
  (RowWidth*2) + (NumSites*8)
```

若要获得精确估计，请在计算中使用 *RowWidthChanged* 的值，如下所示：

```
InboundMsgSizeupdate = InboundMsgOverhead +
  ColOverhead + RowWidth + RowWidthChanged
OutboundMsgSizeupdate = OutboundMsgOverhead +
  RowWidth + RowWidthChanged + (NumSites*8)
```

如果使用最少列数功能，请使用以下公式计算消息大小：

```
InboundMsgSizeupdate = InboundMsgOverhead +
  ColOverhead + (RowWidthChanged*2) +
  PrimaryKeyWidth
OutboundMsgSizeupdate = OutboundMsgOverhead +
  (RowWidthChanged*2) + PrimaryKeyWidth +
  (NumSites*8)
```

有关这些计算的示例，请参见第 133 页的 “计算表更新”。

表插入

使用以下公式计算表插入操作的消息大小。（使用最少列数功能时也适用此公式。）

```
InboundMsgSizeinsert = InboundMsgOverhead +
  ColOverhead + RowWidth
OutboundMsgSizeinsert = OutboundMsgOverhead +
  RowWidth + (NumSites*8)
```

表删除

如果*不*使用最少列数功能，则使用此公式计算表删除操作的消息大小：

```
InboundMsgSizeinsert = InboundMsgOverhead +
  ColOverhead + RowWidth
OutboundMsgSizeinsert = OutboundMsgOverhead +
  RowWidth + (NumSites*8)
```

如果使用最少列数功能，请使用以下公式计算表删除操作的消息大小：

```
InboundMsgSizedelete = InboundMsgOverhead +
  ColOverhead + PrimaryKeyWidth
OutboundMsgSizedelete = OutboundMsgOverhead +
  PrimaryKeyWidth + (NumSites*8)
```

函数

使用以下公式计算函数的消息大小：

```
InboundMsgSizefunction = InboundMsgOverhead +
  ParameterWidth +(RowWidth*2)
OutboundMsgSizefunction = OutboundMsgOverhead +
  ParameterWidth + (NumSites*8)
```

在计算入站消息大小的公式中，由于复制函数的前映像和后映像不发送到入站队列中，所以 *RowWidth* 不适用于复制函数功能。

开始 / 提交对

使用以下公式计算开始 / 提交对的消息大小：

```
InboundMsgSizebegin = OutboundMsgSizebegin = 250
InboundMsgSizecommit = OutboundMsgSizecommit = 200
```

开始 / 提交对的总大小为 450 个字节。如果典型事务进行了多次修改，则可以在计算中省略开始或提交消息大小。它们对消息总大小的影响可以忽略不计。

公式构成

在上面的公式中：

- *InboundMsgOverhead* 等于 380 个字节。每条消息都包含事务 ID、重复的检测序列号等。
- *OutboundMsgOverhead* 等于 200 个字节。每条消息都包含事务 ID、重复的检测序列号等。
- *ColOverhead* 仅适用于入站队列，每列的开销等于 30 个字节，如下所示：

  对于不使用最少列数功能的更新操作：

  ```
  (NumColumns+NumColumnsChanged) * 30
  ```

  对于使用最少列数功能的更新操作：

  ```
  ((NumColumnsChanged*2)+NumPrimaryKeyColumns) * 30
  ```

对于使用或不使用最少列数功能的插入操作：

```
NumColumns * 30
For a delete operation without minimal columns:
NumColumns * 30
```

对于使用最少列数功能的删除操作：

```
NumPrimaryKeyColumns * 30
```

- *RowWidth* 是用 ASCII 格式表示的表列的大小。例如：char(10) 列使用 10 个字节， binary(10) 列使用 22 个字节。

  对于表更新，由于行的前映像和后映像都要分发到复制节点中，所以 *RowWidth* 要乘以 2。

- *RowWidthChanged* 是行中已更改列的宽度。例如，如果有 10 列的总 *RowWidth* 为 200 个字节，行中的半数列更改，则 *RowWidthChanged* 大约为 100 个字节。

- *NumSites* 是消息要发送到的节点数。这个测量值仅在将较短的消息分发到许多节点时才有意义；在节点不多的情况下可能并不重要。如果节点数量不多，则最好从相应公式中省略节点数因子。

- *ParameterWidth* 是用 ASCII 格式表示的函数参数的大小。

- *Begin/Commit Pair* 是消息的开始标头和提交标尾的大小之和，等于 450 个字节。

更改速率 （消息数）

更改速率是每一时间段对表执行的修改 （插入、删除和更新）的最大次数。不同表的更改速率各有不同。

规划容量时，使用的更改速率应始终大于为该表记录的最大更改速率。

注释 更改速率值表示为*单位时间的操作数*（例如，每秒 5 次更新）。计算速率时应始终使用同一个时间段。如果测量每秒更新数，则必须以秒为单位进行所有其他时间段的计算。

更改量 （字节数）

更改量是指在每个时间段内表中更改的数据量。这个量因表而异。它是表的更改速率和数据大小的函数。

计算表容量

复制的数据量等于每条消息的大小乘以每秒复制的消息数。如果知道每次对表进行修改以及每个复制函数的消息大小，则可以通过对各表和复制函数的容量求和，计算数据库中生成的消息总量。

表容量上限方法

为方便以后计算，应分开计算入站和出站队列的表容量和数据库容量。或者，也可以计算出站队列的容量，并用该数字作为入站和出站容量的近似值。

通过用各表的更新速率乘以最长消息的大小，可以计算这些表的容量上限。这样即可求出表容量的上限。

```
InboundTableVolumeupper = (Max[InboundMsgSize] *
    ChangeRate)
OutboundTableVolumeupper = (Max[OutboundMsgSize] *
    ChangeRate)
```

其中：

- *Max[InboundMsgSize]* 和 *Max[OutboundMsgSize]* 分别是用字节表示的最大入站消息和出站消息的大小 （通常就是具有最大参数的消息大小）。
- *ChangeRate* 是每个时间段对表进行修改的最大次数。

表容量值求和方法

更为精确的计算数据量的方法是：对各种不同类型的消息的容量求和。

计算 *InboundTableVolume*：

```
InboundTableVolume =
     (InboundMsgSizeupdate * ChangeRateupdate) +
     (InboundMsgSizeinsert * ChangeRateinsert) +
     (InboundMsgSizedelete * ChangeRatedelete) +
     (InboundMsgSizefunction1 * ChangeRatefunction1) +
     (InboundMsgSizefunction2 * ChangeRatefunction2) +
     ...
```

计算 *OutboundTableVolume*：

```
OutboundTableVolume =
     (OutboundMsgSizeupdate * ChangeRateupdate) +
     (OutboundMsgSizeinsert * ChangeRateinsert) +
     (OutboundMsgSizedelete * ChangeRatedelete) +
     (OutboundMsgSizefunction1 * ChangeRatefunction1) +
     (OutboundMsgSizefunction2 * ChangeRatefunction2) +
     ...
```

其中：

- *ChangeRate* 是每个时间段对表进行修改的最大次数。$ChangeRate_{update}$ 指的是数据更新次数，$ChangeRate_{insert}$ 指的是数据插入次数，依此类推。
- *InboundMsgSize* 和 *OutboundMsgSize* 是不同类型的入站和出站最大消息大小。（请参见第 123 页的 “消息大小”。）

事务量

TransactionVolume 估计由开始和提交记录生成的数据量。

InboundTransactionVolume 的公式如下：

```
InboundTransactionVolume = (MsgSizecommit +
MsgSizebegin) * InboundTransactionRate
```

其中：

- $MsgSize_{begin} + MsgSize_{commit}$ 等于 450 个字节 （每个事务）。
- *InboundTransactionRate* 是在主数据库中每个时间段处理的事务总量。其中包含所有事务 — 更新和未更新复制表的事务都包含在内。一个事务可能涉及针对一个或多个表的一个或多个修改。

计算 *OutboundTransactionVolume* 的公式如下：

```
OutboundTransactionVolume = (MsgSizecommit +
MsgSizebegin) * OutboundTransactionRate
```

其中：

- $MsgSize_{begin} + MsgSize_{commit}$ 等于 450 个字节 （每个事务）。
- *OutboundTransactionRate* 是每个时间段通过特定出站队列复制的复制事务总数。每个事务可能涉及针对一个或多个表的一个或多个修改。

OutboundTransactionRate 取决于：

- 实际更新复制数据的事务数
- 这些事务中通过特定出站队列复制的事务数

例如，如果 *InboundTransactionRate* 为每秒 50 个事务，并且其中一半事务更新了复制表，对复制表的更新中有 20% 是通过队列 1 复制的，则队列 1 的 *OutboundTransactionRate* 为每秒 5 个事务 (50 * 0.5 * 0.2)。

如果事务包含针对许多不同复制表 （复制选择性各有不同）的更新，则计算 *OutboundTransactionRate* 会较为复杂。在这种情况下，*InboundTransactionRate* 为 *OutboundTransactionRate* 提供了方便的上限，可代替 *OutboundTransactionRate* 在公式中使用。

如果数据库中的所有事务都通过出站队列复制，则 *OutboundTransactionRate* 将与 *InboundTransactionRate* 相同。

数据库容量

通过对每个表的消息速率上限求和，可以计算出数据库容量上限的粗略估计值。

```
InboundDatabaseVolumeupper =
sum(InboundTableVolumeupper) + InboundTransVolume
```

有关这些计算的示例，请参见第 135 页的 “入站数据库容量”。

更精确的计算 *InboundDatabaseVolume* 的方法是对使用第 126 页的 “表容量上限方法”中所述方法计算出的 “表容量”求和。

```
InboundDatabaseVolume =
sum(InboundTableVolume) + InboundTransactionVolume
```

入站队列大小

入站队列包含主数据库中所有复制表的更新。入站队列保留这些更新的时间也就是最长打开事务的持续时间。队列大小用字节数表示。

计算入站队列的平均大小：

```
InboundQueueSizetypical = InboundDatabaseVolume *
TransactionDurationtypical
```

计算入站队列的最大大小：

```
InboundQueueSizelongest = InboundDatabaseVolume *
TransactionDurationlongest
```

其中：

- *InboundDatabaseVolume* 是用第 128 页的 “数据库容量”中所述的公式计算出的消息量。

  如果不求精确，则可以使用 *OutboundDatabaseVolume* 计算 *InboundQueueSize*。

- $TransactionDuration_{typical}$ 表示典型事务的秒数。
- $TransactionDuration_{longest}$ 表示绝对事务的秒数。

有关计算入站队列的最大大小的示例，请参见第 135 页的 “入站队列大小计算示例”。

实际上，入站队列的最大大小略高于计算值，因为读取长事务的同时还要向日志添加新事务。由于最长事务的持续时间是一个近似值，所以在确定其值时，应加入从稳定队列中读取并处理该事务所需的较短时间的估计值。

还要注意如果消息缓慢进入 Replication Server，队列可能略大一些。Replication Server 以固定的 16K 块大小向稳定队列写入消息，而且为缩短延迟，每秒甚至还写入不完整的块。（可以使用 init_sqm_write_delay 配置参数将时间延迟改为 1 秒之外的值。）如果有多条消息几乎同时到达，所有消息都将写入一个块。但如果同样的消息在不同的间隔到达，每条消息可能独占一个块。

注释 由于 Adaptive Server 事务日志要求可能因 Adaptive Server 版本更新而提高， Replication Server 稳定队列的空间要求也可能随之提高。

出站队列容量

出站队列将数据发送到另一个 Replication Server 或是数据服务器。发送目标在这些计算中被称为“直接目标节点”。对于复制到“直接目标节点”或通过其复制的每个数据库，“表容量”的某个部分会经过队列。表中通过该队列复制的那部分就称为复制选择性。

通过假定所有表的复制选择性最高为 1 （即 100%），可以计算出 *OutboundQueueVolume* 的上限。这样， *OutboundQueueVolume* 就是对通过队列复制的所有表的 *OutboundTableVolumes* 求和。

```
OutboundQueueVolumeupper= sum(OutboundTableVolumes) +
OutboundTransactionVolume
```

例如，如果通过出站队列复制两个表， *OutboundTableVolumes* 分别为 20K/ 秒和 10K/ 秒，并且假定 *OutboundTransactionVolume* 为 1K/ 秒，则 *OutboundQueueVolume* 计算如下：

```
20K/Sec + 10K/Sec + 1K/Sec = 31K/Sec
```

通过对复制表选择性数值进行分解，可以更准确地测量队列容量。使用 *ReplicationSelectivity* 的 *OutboundQueueVolume* 的公式如下：

```
OutboundQueueVolume = sum(OutboundTableVolume *
ReplicationSelectivity) + OutboundTransactionVolume
```

例如，在上例中，如果复制了第一个表的 50%，第二个表的 80%，则 *OutboundQueueVolume* 计算如下：

```
(20K/Sec*0.5) + (10K/Sec*0.8) + 1K/Sec = 19K/Sec
```

有关上述计算的示例，请参见第 136 页的 “出站队列容量计算示例”。

故障持续时间

如果直接目标节点不可用，队列将缓冲其中的消息。设置队列大小时所用的故障持续时间数值决定队列能够经受故障的持续时间。

保存间隔

保存间隔指定在删除数据之前在出站队列中将数据保留多长时间。保存间隔可以在磁盘或节点丢失数据时帮助恢复。

使用 configure connection 命令可以为数据库连接指定保存间隔。使用 configure route 命令可以为指向另一个 Replication Server 的路由指定保存间隔。有关其他信息，请参见 《Replication Server 参考手册》。

出站队列大小

使用以下公式可以计算出站队列的大小：

```
OutboundQueueSize = OutboundQueueVolume *
(FailureDuration + SaveInterval)
```

其中：

- *OutboundQueueVolume* 是每个时间段进入队列的数据量 （用字节表示）。
- *FailureDuration* 是期望队列为不可用节点缓冲数据的最长时间。
- *SaveInterval* 是在删除消息之前将其保留在队列中的时间。

如果 *OutboundQueueVolume* 为每秒 18K 的出站队列的 *SaveInterval* 为 1 小时，并预计经受目标节点不可用的时间为 1 小时，则队列的大小应为：

```
OutboundQueueSize =
 18K/sec * (60min + 60min) =
 0.018MB/sec * 120min * 60sec/min = 130MB
```

有关这些计算的更详尽的示例，请参见第 136 页的 “出站队列大小”。

实际上，如果实际更改速率和表选择性造成每秒填充的块不足完整的 16K，那么上面计算的值就要低一些。 Replication Server 每秒写入一个块，不论这个块是否填满。执行小型事务的时间超出一秒时，队列空间的利用率最低。统计数据表明多数块的容量都在 50% 到 95% 之间。

总体队列磁盘使用情况

若要计算在最坏条件下使用队列所需的总磁盘空间，请使用以下公式：

```
Sum[InboundQueueSize] + Sum[OutboundQueueSize]
```

其中：

- *Sum[InboundQueueSize]* 是各个数据库的入站队列大小之和。
- *Sum[OutboundQueueSize]* 是各个直接目标的出站队列大小之和。

其他注意事项

规划复制系统时，需要考虑以下事项：

- 如有某个远程节点因网络故障而不可用，同一网络中的其他节点也可能不可用，以致众多队列同时增长。因此，需要为所有队列分配充足的磁盘空间，以应对规划期间同时出现故障的情况。
- 随着队列开始不断扩充，可以在有可用磁盘空间的前提下随时添加更多分区。
- 如果所有队列均已填满并且入站队列无法接收更多的消息，主数据库将无法截断其日志。如果手工忽略这方面的限制，复制数据库将无法接收截断的更新。

队列使用情况计算示例

以下各节提供了一系列使用上文公式进行计算的示例。

示例 , 计算参数

这些计算针对示例复制系统采用了以下假设：

- 表 T1 和 T2 位于数据库 DB1 中。
- 表 T3 位于 DB2 中。
- 所有三个表都复制到节点 S1、 S2 和 S3。
- 参数显示在以下各表中 （请注意并不是所有消息都复制到所有节点）。

表 A-1：表参数

数据库	表	列数	更改的列数	更改速率（数目 / 秒）	行宽（字节数）	更改的行宽	节点数
DB1	T1	10	5	20	200	100	3
DB1	T2	10	5	10	400	200	2
DB2	T3	120	100	2	1500	1000	2

表 A-2：节点参数 — 表复制选择性

表	S1 更新百分比	S2 更新百分比	S3 更新百分比
T1	10%	40%	40%
T2	40%	80%	0%
T3	20%	0%	20%

表 A-3：事务发生率

事务发生率	DB1	DB2	S1	S2	S3
入站	20/ 秒	20/ 秒	-	-	-
出站	-	-	5/ 秒	10/ 秒	8/ 秒

消息大小计算示例

以下示例在计算中使用了 *RowWidthChanged* 公式（不使用最少列数功能）。有关可用于计算消息大小的其他公式，请参见第 123 页的 “消息大小”。

计算表更新

使用以下公式：

```
InboundMsgSizeupdate = InboundMsgOverhead +
  ColOverhead + RowWidth + RowWidthChanged
OutboundMsgSizeupdate = OutboundMsgOverhead +
  RowWidth + RowWidthChanged + (NumSites*8)
```

每个节点的更新计算如下：

```
InboundMsgSizeT1 = 380 + 450 + 200 + 100 = 1130
  bytes
InboundMsgSizeT2 = 380 + 450 + 400 + 200 = 1430
  bytes
InboundMsgSizeT3 = 380 + 6600 + 1500 + 1000 = 9480
  bytes
OutboundMsgSizeT1 = 200 + 200 + 100 + 24 = 500 bytes
OutboundMsgSizeT2 = 200 + 400 + 200 + 16 = 800 bytes
OutboundMsgSizeT3 = 200 + 1500 + 1000 + 16 = 2700
  bytes
```

开始 / 提交对

```
MsgSizebegin = 250
MsgSizecommit = 200
```

更改速率

更改速率是在每个时间段内对表执行的最大数据修改次数，所以复制系统中的每个表的更改速率自然是各不相同的。

以下计算示例使用第 132 页的表 A-1 中所示的更改速率。

表容量计算示例

现在为表 T1、 T2 和 T3 计算表容量。为简单起见，示例中说明的是对最坏情况的分析，即假定所有更改都源自 SQL update 语句及与其关联的开始 / 提交对。

使用上限公式计算 *InboundTableVolume*：

```
InboundTableVolume = (Max[InboundMsgSize]*ChangeRate)
```

表 T1、 T2 和 T3 的入站表容量分别为：

```
InboundTableVolumeT1 = 1130*20 = 23K/sec
InboundTableVolumeT2 = 1430*10 = 14K/sec
InboundTableVolumeT3 = 9480*2 = 19K/sec
```

使用上限公式计算 *OutboundTableVolume*：

```
OutboundTableVolume = (Max[OutboundMsgSize]*
ChangeRate)
```

表 T1、 T2 和 T3 的出站表容量分别为：

```
OutboundTableVolumeT1 = 524*20 = 10K/sec
OutboundTableVolumeT2 = 816*10 = 8K/sec
OutboundTableVolumeT3 = 2716*2 = 5K/sec
```

其中：

- *InboundMsgSize* 和 *OutboundMsgSize* 的值来自第 133 页的 “计算表更新”中的计算结果。
- *ChangeRate* 的值来自第 132 页的表 A-1。

RSSD 和日志大小计算示例

每个 Replication Server 都有一个拥有自己的入站队列的 RSSD。RSSD 也会加大出站队列的队列容量。不过，由于 RSSD 中的活动量微乎其微而且所有事务都非常小，您可以假定 RSSD 的磁盘空间要求很低：

- 入站队列 = 2MB
- 出站队列 = 无

入站数据库容量

基于上面计算的 *InboundTableVolumes*，可以使用下限公式计算 *InboundDatabaseVolume*：

```
InboundDataBaseVolume = sum(InboundTableVolume) +
InboundTransactionVolume
```

其中：

- *InboundTableVolumes* 的值来自上一页的计算结果。
- *TransactionVolume* 是 $Message_{begin}$ 与 $Message_{commit}$ 对的和 （250 个字节 + 200 个字节）乘以 *TransactionRate*。
- 在上述示例中，每个数据库的事务发生率均来自第 132 页的表 A-3。

因此， *InboundDatabaseVolumes* 计算如下：

```
InboundDatabaseVolumeDB1 = 23K + 14K +
 (450 bytes/tran * 20 tran/sec) = 46K/sec
InboundDatabaseVolumeDB2 = 19K +
 (450 bytes/tran * 20 tran/sec) = 28K/sec
```

入站队列大小计算示例

基于 *InboundDatabaseVolumes*，可以使用以下公式计算 DB1 和 DB2 入站队列的最大大小：

```
InboundQueueSize = InboundDatabaseVolume *
TransactionDuration
```

其中：

- *Inbound DatabaseVolume* 是上面计算的消息量。
- *TransactionDuration* 是绝对最长事务的秒数。为便于在本例中说明，假定 DB1 和 DB2 的 *TransactionDuration* 分别为 10 分钟和 5 分钟。

InboundQueue 的大小为：

```
InboundQueueDB1 = 46K/sec * 600sec = 28MB
InboundQueueDB2 = 28K/sec * 300sec = 8MB
```

出站队列容量计算示例

现在为表 T1、 T2 和 T3 计算 *OutboundQueueVolume*。由于是基于表（而不是数据库）测量选择性，必须为每个通过出站队列复制的表计算出站队列大小。计算 *OutboundQueueVolume* 最大值的公式如下：

```
OutboundQueueVolume = sum(OutboundTableVolume *
ReplicationSelectivity) + OutboundTransactionVolume
```

其中：

- *OutboundTableVolume* 的值来自第 126 页的 “计算表容量”中计算的结果。
- *ReplicationSelectivity* 的值来自*表 A-2： 节点参数 — 表复制选择性*。
- *TransactionVolume* 是 $Message_{begin}$ 和 $Message_{commit}$ 对的大小（250 个字节 + 200 个字节）乘以 *TransactionRate*。

计算节点 1 的 *OutboundQueueVolume* 的公式如下：

```
OutboundQueueVolumeS1 = OutboundTransactionVolume +
(TableVolumeT1*ReplicationSelectivityT1,S1) +
(TableVolumeT2*ReplicationSelectivityT2,S1) +
(TableVolumeT3*ReplicationSelectivityT3,S1)
```

三个节点的 *OutboundQueueVolumes* 分别如下：

```
Site1 = (450*5) + (10K/sec*0.1) + (8K/sec*0.4) +
(5K/sec*0.2) = 7K/sec
Site2 = (450*10) + (10K/sec*0.4) + (8K/sec*0.8) +
(5K/sec*0) = 15K/sec
Site3 = (450*8) + (10K/sec*0.4) + (8K/sec*0) +
(5K/sec*0.2) = 9K/sec
```

出站队列大小

使用以下公式可以计算出站队列的大小：

```
OutboundQueueSize = OutboundQueueVolume *
(FailureDuration + SaveInterval)
```

其中：

- *OutboundQueueVolume* 是每个时间段进入队列的数据量 （用字节表示）。
- *FailureDuration* 是期望队列为不可用节点缓冲数据的最长时间。为便于这些计算示例的计算，假定故障持续时间设置为 4 小时（14,400 秒）。

- *SaveInterval* 是为该特定队列配置的时间。为便于这些计算示例的计算，假定保存间隔设置为 2 小时 （7200 秒）。
- *FailureDuration* + *SaveInterval* 等于 21,600 秒。

三个节点的 *OutboundQueueSizes* 分别如下：

```
Site1 = 7K/sec * 21,600sec = 151MB
Site2 = 15K/sec * 21,600sec = 324MB
Site3 = 9K/sec * 21,600sec = 194MB
```

总磁盘队列使用情况计算示例

若要计算发生最坏的故障情况期间 （4 小时）处理所有队列所需的总磁盘空间，请使用以下公式：

```
Sum(InboundQueueSizes) + Sum(OutboundQueueSizes)
```

其中：

- *InboundQueueSizes* 指在第 135 页的 “入站队列大小计算示例”中计算的两个队列。
- *OutboundQueueSizes* 指上面计算的三个队列。

最坏条件下所需的总磁盘空间 （包括 2MB 用于 RSSD 入站队列）的计算公式如下：

```
2MB + 28MB + 8MB + 151MB + 324MB + 194MB = 707MB
```

Sybase 建议您分配足够的空间以应对最坏的情况。如果利用保存间隔功能 （即便将消息发送到下一个节点也不截断出站队列），则必须确保分配足够的队列空间以应对峰期的事务处理量。如果不使用保存间隔，通常情况下的队列利用率会很低，每个队列也许只有 1MB 或 2MB。

计算示例中假定所有出站队列都要经受相同的故障持续时间。这种假设可能在您的环境中并不成立。通常， WAN 连接必须能比本地连接经受更长的故障持续时间。

其他磁盘空间要求

本节涉及复制系统中的 Replication Server、Replication Agent 和数据服务器的空间要求。 LTM 本身不占用磁盘空间，但其错误日志文件除外。

稳定队列

稳定队列的计算在本章前面各节中已进行了说明。

RSSD

对于 RSSD，至少要为数据分配 10MB，为日志分配 10MB。这些复制系统缺省值是为较小的系统设计的。如果需要为数以百计的复制定义和数以千计的预订准备充足的空间，应该将数据和日志空间增大到 12MB。

错误和被拒绝的事务也放置在 RSSD 中。管理员应定期截断包含这些事务的表。如果将稳定队列转储到 RSSD 以帮助诊断问题，一定要在不需要表时将其截断。

日志

Replication Server 和 LTM 将信息、错误消息和跟踪输出写入错误日志文件。应该为这些日志文件分配 1MB 到 2MB 的磁盘存储空间。如果 Sybase 技术支持部门要求打开跟踪标志，您可能需要增加大量的可用日志空间。

RepAgent 将消息写入 Adaptive Server 错误日志文件。

内存使用情况

Replication Server 和 LTM 是复制系统中的不同进程。 RepAgent 是 Adaptive Server 线程。

以下各节估计了上面各个进程 / 线程的大致内存要求。

Replication Server 内存要求

在 Sun SPARCstation 上，一般需要估计以下值：

- 新安装的 Replication Server 的数据和堆栈使用大约 7MB 的内存。
- 每个 DSI 连接大约增加 500K。如果为 MD 内存（md_source_memory_pool 配置参数）配置更大的值，该值将随之增大。
- 每个 RepAgent （或 LTM）连接增加 500K。如果为 SQT 内存（sqt_max_cache_size 配置参数）配置更大的值，该值将随之增大。
- 如果有数以千计的预订或者如果增大函数字符串高速缓存的大小，则必须为此分配更多的内存空间。
- 预订规则使用的内存是预订中引用的列和规则数的函数。典型预订增加不到 80 个字节再加上所有预订谓词值的总大小。
- 函数字符串高速缓存大小可增至 200K 或 fstr_cachesize 配置参数的值。
- 其余系统表高速缓存的大小取决于定义的对象数。每个复制定义占用的内存量约等于其列数占用量的 250 倍。如果有大量复制定义包含许多列，这可能也是一个要考虑的因素。

RepAgent 内存要求

大多数 RepAgent 内存来自分配的 Adaptive Server 过程高速缓存 （共享内存）。这部分内存用于以下方面：

- 开销
- 模式高速缓存
- 事务高速缓存
- text 和 image 高速缓存

有关增加服务器内存的信息，请参见《Sybase Adaptive Server 管理指南》。

开销

无论是否启用 RepAgent（或 LTM），Adaptive Server 都会为每个数据库分配 5612 个字节用于传输日志。

启用 RepAgent 后， Adaptive Server 一启动就会为每个 RepAgent 分配额外 2332 个字节的内存。

模式高速缓存

用于模式高速缓存的内存量取决于复制的对象数 （表和存储过程）以及描述符数 （列和参数）。每个对象和描述符需要 128 个字节。

启动时， Adaptive Server 分配 8K 内存，该内存量足够 64 个对象 / 描述符使用。此后，内存以 2K 的块分配。 Adaptive Server 在启动时还会分配 2048 个字节用于散列表。

“最近使用最少的” (LRU) 机制可以从内存中删除那些最近未引用的对象 / 描述符，从而控制模式高速缓存的大小。因此， RepAgent 不需要大量内存来描述所有复制对象。但 RepAgent 至少需要有足够的内存来描述一个复制对象。

事务高速缓存

RepAgent 要求为每个打开的事务分配 256 个字节。事务高速缓存以 2K 块为单位分配。随着事务的提交和中止，可用内存被放入内存池，必须先用完这部分内存然后再分配新内存。

RepAgent 需要可用于最大数量的打开事务的内存。若不能提供充足的可用内存， RepAgent 就会关闭。

启动时， Adaptive Server 会分配 2048 个字节用于散列表。

text 和 *image* 高速缓存

text 和 image 高速缓存不受 text 和 image 数据大小的影响。相反，所用内存量取决于包含 text 和 image 数据的复制表数以及包含 text 和 image 数据的表中的列数。每个包含 text 和 image 数据的复制表需要 170 个字节，每个复制列需要 52 个字节。

text 和 image 高速缓存以 2K 块为单位分配。只有存在包含 text 和 image 数据的复制表时才会分配内存。

text 和 image 高速缓存使用可用内存池并需要足够的内存用于所有 text 和 image 数据。

其他内存

- 如果启用 RepAgent，Adaptive Server 会为 RepAgent 另外分配一个进程描述符。
- RepAgent 使用 ct-lib 连接 Replication Server。ct-lib 可按需直接分配内存 （动态内存）。

LTM 内存要求

除可执行程序外， LTM 有以下两方面的内存要求：

- 数据 LTM 在内存中维护
- 对固定数据结构的一些最低要求 (4MB)

例如，如果 LTM 需要保留在内存中的最大数据量为 1MB，那么该 LTM 的内存要求为 5MB。

LTM 数据内存要求

从最早打开的事务开始， LTM 将每个事务的环境保留在内存中。最早打开事务不必更新任何复制表。下面列出了近似计算 LTM 内存使用率的一种不错的经验做法：

```
TransactionRate * DurationOfLongestTransaction *
LTMTransactionOverhead
```

其中：

- *TransactionRate* 是复制系统的每个时间段的事务数。
- *DurationOfLongestTransaction* 取决于复制系统的实际性能。
- *LTMTransactionOverhead* 大约为 250 个字节。

例如，如果数据库 *TransactionRate* 为每秒 10 个事务，并且 *DurationOfLongestTransaction* 为 5 分钟，则 LTM 的最大内存使用率为:

```
10 tran/sec * 300 seconds * 250 bytes/tran = 750K
```

CPU 使用率

Replication Server 在一个处理器上运行。它可以在 SMP 系统上运行，但不利用更多的 CPU。

Replication Server 扮演以下三种角色：

- 通过 Replication Agent 获得并处理从主数据库分发的数据
- 将分发的数据应用于复制数据库
- 从另一个 Replication Server 接收消息，并将其路由到其他 Replication Server

每种角色占用 CPU 的时间各有不同。可以一个 Replication Server 扮演所有角色，也可以多个 Replication Server 扮演不同的角色。

总体性能取决于表中的列数、列的大小、预订数、预订节点数等。

网络要求

计算出出站队列的插入速率后，即可了解所需的网络吞吐量。这个吞吐量应能让 Replication Server 发送消息的速度快于消息加入队列的速度。网络消息通常稍大于写入出站队列的消息。

有时，填充队列的数据暴增而通过带宽可能很低的网络流出的速度又很慢。在计算队列大小时应考虑到这种情况，确保队列除处理连接和目标节点故障外，还能处理暴增的流入数据。

附 录 B

日志传送语言

本附录介绍 Replication Agent 组件发送到主 Replication Server 的日志传送语言 (LTL)。

日志传送语言概述

表 B-1 列出了 RepAgent （Adaptive Server Enterprise 数据库的 Replication Agent 组件）所使用的 LTL 命令。

表 B-1：日志传送语言命令

命令	说明
connect source	标识作为日志传送会话的 Replication Server 连接。
get maintenance user	检索维护用户的登录名。
get truncation	从 Replication Server 检索数据库的截断点。
distribute	将日志记录提交给 Replication Server。此命令有几个子命令。请参见第 148 页的 “distribute”。

表 B-1 中的前三个命令用于与 Replication Server 协调 Replication Agent 会话。最后一个命令 distribute 用于向 Replication Server 提交日志记录。

以下各节说明了 LTL 命令的语法和用法。

connect source

登录到 Replication Server 之后， RepAgent 会发出 connect source 命令进行自我标识。该命令指定数据服务器和 RepAgent 要转发的数据库日志，以及建议使用的 LTL 版本。

以下是 connect source 命令的语法：

```
connect source lti data_server.database version_no
[sendallxacts] [warmstdb] [in recovery]
```

其中 *data_server* 和 *database* 标识数据服务器和 RepAgent 要转发的日志所属的数据库。 *version_no* 参数标识要在 connect source 命令中使用的 LTL 版本。

RepAgent （或 LTM）和 Replication Server 协商决定会话中使用的 LTL 版本。这样可以确保 Replication Server 的新版本能够使用为 Replication Server 早期版本编写的 Replication Agent。表 B-2 列出了 Replication Server 与 LTL 之间的兼容性。

表 B-2：Replication Server 与 LTL 的兼容性

Replication Server 版本	LTL 版本
12.5	400
12.1	300
12.0	300
11.5	200
11.0	103
10.1.1	102
10.1	101
10.0	100

- 如果 Replication Agent 使用 Replication Server 12.5 版的功能，则必须使用 LTL 400 版并连接到 Replication Server 12.5 或更高版本。
- 如果 Replication Agent 使用 Replication Server 12.x 版的功能，则必须使用 LTL 300 版并连接到 Replication Server 12.x 或更高版本。
- 如果 Replication Agent 使用 Replication Server 11.5 版的功能，则必须使用 LTL 200 版并连接到 Replication Server 11.5 或更高版本。
- 如果 Replication Agent 使用 Replication Server 11.0 的功能，则必须使用 LTL 103 版并且必须连接到 Replication Server 11.0 版。

- 如果 Replication Agent 不使用 Replication Server 11.0 的功能，但使用 Replication Server 10.1 的功能，则必须使用 LTL 101 或 102 版并且可以连接到 Replication Server 10.1 或更高版本。
- 如果 Replication Agent 只使用 Replication Server 10.0 的功能，则它可以使用 LTL 100 并且可以连接到任意 Replication Server。

关键字

RepAgent 根据为它设置的参数来为 connect source 命令使用不同的关键字。这些关键字包括：

- sendallxacts — 以 send_maint_xacts_to_replicate 标志启动时，RepAgent 会将所有对复制表的更新 （包括维护用户所作的更新）提交到 Replication Server，以便分发到预订复制节点。
- warmstdb — 以 send_warm_standby_xacts 标志启动时，RepAgent 会将所有对复制表的更新 （包括维护用户所作的更新）提交到 Replication Server，以便应用于备份数据库。
- in recovery — 以 for_recovery 标志启动时， RepAgent 将处于恢复模式。处于恢复模式的 Replication Server 只允许与处于恢复模式的 RepAgent 建立连接。恢复模式用于重放恢复的事务日志，以便能恢复丢失的消息。有关处于恢复模式的 RepAgent 的作用的详细信息，请参见 《Replication Server 管理指南》。

升级定位符

在 Replication Server 11.0 和更高版本中， connect source 另外返回一行，提供 Replication Server 系统版本号和升级定位符。

升级定位符提供在升级系统之前写入入站队列的最后一条消息的源队列 ID。有关源队列 ID 的详细信息，请参见第 147 页的 “源队列 ID 的格式”。

在创建 Replication Agent，或从 Replication Server 11.0 之前的版本升级到 11.0 或更高版本时，或是在混合模式事务 （同时包含应用函数和请求函数）进行的过程中，升级定位符十分有用。

connect source 示例

下面的 connect source 示例标识作为 NY_DS 数据服务器中 Stocks_db 数据库的 RepAgent 会话的会话。 RepAgent 和 Replication Server 在为 Replication Server 11.5 版使用 LTL 200 版方面达成一致。

```
connect source lti NY_DS.Stocks_db 200
VERSION
-----------
200
Sysversion     UpgradeLocater
-----------    --------------
1150         0x000000000...
```

get maintenance user

连接到 Replication Server 后， RepAgent 会发出 get maintenance user 命令，以查找数据库维护用户的登录名。 Replication Server 以维护用户的身份更新复制的数据副本。 RepAgent 使用登录名来区分主数据更新与通过复制系统分发的更新。

- 维护用户进行的更改是分发的结果，并且 Replication Server 不会再重新分发这些更改 （热备份数据库除外，这种情况下， RepAgent 被使用 send_warm_standby_xacts 配置为这样操作）。
- 维护用户以外的用户对主数据进行的更改是 Replication Server 分发到其他数据库的主数据更新。

处理复制数据所在数据库的日志的 RepAgent 必须过滤出维护用户所作的所有更改。

注释 Adaptive Server 的 RepAgent 或 SQL Server 的 LTM 可以在不过滤维护用户所执行事务的模式下运行。将合并复制表复制到其他节点时可以使用此功能。

对 get maintenance user 命令使用以下语法：

```
get maintenance user for data_server.database
```

Replication Server 返回包含 char(30) 列的一行，该列显示数据库维护用户的登录名。

下例查找 NY_DS 数据服务器中的 pubs2 数据库的维护用户：

```
get maintenance user for NY_DS.pubs2_db


Maintenance_user
------------------------------
pubs2_db_maint
```

get truncation

get truncation 命令返回一个称为源队列 ID 的 36 字节二进制值。下节将对这个源队列 ID 的格式进行说明。

RepAgent 使用源队列 ID 执行以下操作：

- 查找保存在 Replication Server 入站队列中的最后一个日志记录
- 更新数据库日志中的截断点

必须定期截断数据服务器日志，以便为更多的日志记录留出空间。RepAgent 使用源队列 ID 更新截断点，并允许数据服务器截断 Replication Server 已经接收的日志记录。

对 get truncation 命令使用以下语法：

get truncation *data_server.database*

其中， *data_server* 和 *database* 标识 Replication Agent 要转发的日志所属的数据库。

Replication Server 返回一行一列的 36 字节内容，其中包含在其入站队列中保存的最后一个命令的源队列 ID。该 ID 的前 32 个字节由 Replication Agent 生成。最后 4 个字节是由 Replication Server 为自身使用而附加的，Replication Agent 将忽略这些字节。

源队列 ID 的格式

源队列 ID 是一个唯一的 32 字节二进制字符串，它的值随每个新日志记录的传送而增加。由于 Replication Server 忽略 ID 低于存储在入站队列中的最高 ID 的记录，所以该值必须增加。

Replication Server 重新启动后，必须能够将源队列 ID 映射到原始日志记录，这样它才能发送 ID 值已增加的下一个日志记录。

表 B-3 显示了 Adaptive Server RepAgent 生成的源队列 ID 的格式。只要值增加，不同数据服务器的 Replication Agent 就能生成任意格式的源队列 ID，并且该 ID 能用来查找原始日志记录。

表 B-3：Adaptive Server RepAgent 的源队列 ID 的格式

字节	内容
1— 2	在重装协调式转储之后恢复时所用的数据库世代号。
3— 8	当前记录的日志页时间戳。
9— 14	当前行的行 ID。行 ID = 页码 （4 个字节） + 行号 （2 个字节）。
15— 20	最早打开事务的起始记录的行 ID。
21— 28	最早打开事务的起始记录的日期和时间。
29— 30	RepAgent 用以回退孤立事务的扩展。
31— 32	未使用。

字节 21— 28 包含一个 8 字节 datetime 数据类型值，该值是数据库日志中最早的部分传送事务的时间。 Replication Server 在消息的这一字段中输出日期和时间，以帮助数据库管理员查找执行恢复所需的脱机转储。如果不在该字段中存储有效日期，消息中输出的日期和时间就没有意义。不过，消息还包含以十六进制形式输出的完整的源队列 ID，因此如果将日期和时间放在字节 21— 28 之外的其他位置，复制系统管理员也能从源队列 ID 中将它提取出来。

distribute

distribute 命令描述事务控制和数据操纵操作。它还传达 Replication Server 用以确保发生故障时，既不会遗漏也不会重复传送事务日志信息的信息。

通常，在建立会话之后， RepAgent 为它从数据库日志中检索到的每个操作都生成一个 distribute 命令。

distribute 命令的格式如下：

```
distribute command_tags subcommand [values]
```

该命令有三个组成部分：

- *command_tags* 字段由 Replication Server 用来在命令与其事务之间建立关联，并确保可靠地传送数据库日志。
- *subcommand* 名称用于指定操作。
- 与操作关联的数据 *values* 对于 commit transaction 和 rollback transaction 之外的所有子命令都是必需的。

命令标记

Replication Server 使用命令标记通常有两个目的：

- 重新组合事务中的命令，以便远程节点上的数据服务器能将事务作为一个单元执行
- 确保每个命令只处理一次

command_tags 的语法如下：

```
[@origin_time=datetime_value]
 @origin_qid=binary_value
 @tran_id=binary_value
 [@mode=0x08]
 [@standby_only={1 | 0}]
```

origin_time

origin_time 参数是一个 datetime 值，该值指定事务或数据操纵操作发生的时间。可使用该参数来报告错误。 origin_time 只能在以下事务控制子命令中使用：begin transaction、 commit transaction 和 rollback transaction。

origin_qid

origin_qid 参数是一个在日志中唯一标识命令的 32 字节二进制值。这是一个顺序号，由 Replication Server 用来在重建 RepAgent 连接后拒绝重复的命令。生成的值如第 148 页的表 B-3 所示。

tran_id

tran_id 参数是一个 120 字节的二进制值，用来标识命令所属的事务。事务 ID 在全局范围内必须是唯一的。确保唯一性的一种方法是，首先为数据库日志构造一个唯一的事务 ID，然后在这个 ID 后附加数据服务器名和数据库名。

例如， RepAgent 按以下格式构造 *tran_id*：

tran_id.data_server.database

其中， *tran_id* 是 RepAgent 使用日志信息生成的值。它包含世代号、日志页时间戳和日志记录的行号。确保该值在数据库范围内是唯一的。*data_server* 和 *database* 标识 RepAgent 传送的日志所属的数据库。

mode

如果在 Replication Server 查找复制定义时要使用所有者名称，则应设置 mode 参数。此参数对于应用的命令而言是可选的。如果所有者名称不可用，则不应设置此参数。

mode 是一个 LTL 200 版参数，适用于 Replication Server 11.5 或更高版本。

standby_only

standby_only 参数决定是否将命令发送到备用和 / 或复制数据库。如果将 standby_only 设置为 1，则命令发送到备用数据库而不发送到复制数据库。如果将 standby_only 设置为 0，则命令同时发送到备用数据库和复制数据库。

standby_only 是一个 LTL 200 版参数，适用于 Replication Server 11.5 或更高版本。此参数对于应用的命令而言是可选的。

事务控制子命令

事务控制子命令包括 begin transaction、 commit transaction 和 rollback transaction。

begin transaction

begin transaction 子命令的语法如下：

```
distribute command_tags begin [system] transaction
 ['tran_name'] [for 'user'['password' | no_password ]]
```

- 有关 *command_tags* 的语法和说明，请参见第 149 页的“命令标记”。
- system 关键字通知 Replication Server 不应用 begin transaction/commit transaction 对内的这个事务。在 Adaptive Server 中，它用于在内部启动或在系统存储过程中启动的事务。对于 LTL 200 或更高版本，system 适用于 Replication Server 11.5 或更高版本。
- *tran_name* 是用于标识事务的可选 varchar(30) 值。事务名无须唯一。Replication Server 使事务名可用于函数字符串中的系统定义参数。
- *user* 和 *password* 是 varchar(30) 值，用于标识执行事务的用户的登录名和口令。应将 *user* 和 *password* 引在引号内。这两个变量对于从主节点之外的节点提交的异步过程调用来说是必需的。

  对于 LTL 101 或更高版本，*password* 是可选的；可以在提供 *user* 时省略它。对于 LTL 100 版，如果提供了 *user*，必须同时提供 *password*。

如果主数据库使用 “统一登录”或者主数据库上的用户设置了代理，则应使用 no_password 选项。在这两种情况下， RepAgent 都无法识别用户口令。对于 LTL 200 或更高版本， no_password 适用于 Replication Server 11.5 或更高版本。

commit transaction、 rollback transaction 和 rollback

在 RepAgent 将事务中的所有命令提交到 Replication Server 之后，它会发送 commit transaction、 rollback transaction 或 rollback 命令。

为 commit transaction、rollback transaction 和 rollback 使用的语法如下所示：

```
distribute command_tags
{commit transaction |
 rollback transaction |
 rollback [from oqid] to] oqid}
```

commit transaction 和 rollback transaction 不使用值。

不带 transaction 关键字的 rollback 子命令要求指定源队列 ID (*oqid*) 值。这个子命令的三种可能的形式为：

- rollback *oqid* — 回退与指定源队列 ID 对应的单个日志记录。此选项支持 DB2 中的微型回退功能。
- rollback to *oqid* — 回退指定源队列 ID 与当前日志记录之间的所有日志记录。
- rollback from *oqid1* to *oqid2* — 回退源队列 ID 位于指定范围的一连串日志记录。

注释 Replication Server 忽略从 RepAgent 接收的回退事务。

applied 子命令

applied 子命令说明在数据库中记录的操作，包括：

- 更新行
- 插入行
- 删除行
- 执行应用的存储过程 （请求存储过程使用 execute 子命令）
- 修改 text 或 image 数据

applied 子命令的语法如下：

```
distribute command_tags applied [owner=owner_name]
 {'table'.rs_update
    yielding before param_list after param_list |
 'table'.rs_insert yielding after param_list |
 'table'.rs_delete yielding before param_list |
 'table'.function_name [param_list]
    yielding after param_list before param_list |
 'table'.rs_datarow_for_writetext
    yielding datarow column_list |
 'table'.rs_writetext
    append [first] [last] [changed] [with log]
    [textlen=100] column_list}
```

- 有关 *command_tags* 的语法和说明，请参见第 149 页的“命令标记”。
- *table* 是应用了操作的数据库表的名称。必须将它引在引号内。
- Replication Server 使用 *table* 将命令与复制定义关联起来。从 Replication Server 11.5 版和 LTL 200 版开始，如果设置了标记 @mode=0x08， Replication Server 还将所有者名称与复制定义关联起来。 create replication definition 命令的 with all tables named *table_identifier* 子句确定如何将 *table* 映射到复制定义：
  - 如果复制定义有 with all tables named *table_identifier* 或 with primary table named *table_identifier* 子句，上述 *table* 将与 *table_identifier* 或指定主表相匹配。
  - 如果省略 with all tables named *table_identifier* 子句和 with primary table named *table_identifier* 子句，那么上述 *table* 就是复制定义的名称。

  RepAgent 不需要知道复制定义。它可以使用数据源的表名。

yielding 子句

对于 rs_update、 rs_insert 和 rs_delete， yielding 子句引入了受操作影响的行的操作前映像和操作后映像。必须根据操作的要求提供操作前映像和 / 或操作后映像。表 B-4 显示哪些操作需要操作前映像和操作后映像：

表 B-4：applied 子命令的操作前映像和操作后映像

操作	操作前映像	操作后映像
rs_update	要求	要求
rs_insert	—	要求
rs_delete	要求	—

table.function_name 形式的 applied 子命令用于在您使用与表复制定义关联的方法时分发复制的存储过程。《Replication Server 管理指南》的附录 A “异步过程”中对此方法进行了说明。

注释 《Replication Server 管理指南》的第 10 章 “管理复制函数”中说明了使用应用函数和请求函数复制存储过程的首选方法。此方法使用 execute 子命令分发复制的存储过程 （即复制函数）。

如果执行存储过程时引发插入或删除操作， RepAgent 会将操作转换为 rs_insert 或 rs_delete LTL 命令。如果执行时导致更新操作， RepAgent 将使用 *function_name* 形式并将被更新行的操作前映像和操作后映像提供给 Replication Server。

名称和参数与存储过程相同的 Replication Server 函数是使用 create function 命令定义的，并且 applied 命令中的 *function_name* 引用此函数。函数名之后的 *param_list* 是存储过程的参数列表。

yielding 子句包含函数修改的表行的操作前映像和操作后映像。针对该表的预订决定将函数分发到的位置。

操作前映像和操作后映像由 *param_list* （列值或参数值的列表）指定。*param_list* 的语法如下：

[@param_name=]*literal*[, [@*param_name*=]*literal*]...

- *param_name* 是列名，对于复制存储过程而言是参数名。
- *literal* 是列值或参数值。

复制定义中的所有列名都必须显示在列表中。 Replication Server 忽略其他所有列。如果提供值时的顺序与这些值在复制定义中的定义顺序相同，那么可以省略列名或参数名。虽然按复制定义顺序提供各个列有助于提高性能，但如果包含列名，也可以按任意顺序列示。

Replication Server 10.1 和更高版本支持优化的 yielding 子句。如果操作后映像值与操作前映像值相同，则可以省略操作后映像值。例如，如果表有 a、 b、 c 三列，而更新时只有 b 列发生更改，则 yielding 子句为：

```
yielding before @a=5, @b=10, @c=15 after @b=12
```

注释 如果使用最少列数功能，则使用 LTL 101 或更高版本的 RepAgent *必须* 省略相同的操作后映像。有关复制最少的列的详细信息，请参见《Replication Server 参考手册》中的 create replication definition 命令。

修改 text 或 image 数据

applied 子命令的 rs_datarow_for_writetext 和 rs_writetext 形式用于分发对 text 或 image 数据所做的修改。这些子命令建立在 Replication Server 10.1 版中将数据打包为结构化标识这一性能特点的基础上。每个 text 或 image 列都有一个特殊的字符和长度字段，后跟实际数据值。按此方法打包数据可使 Replication Server 无需再解释数据的每个字节，从而提高了性能。

rs_datarow_for_writetext 传递数据行的映像，而该数据行与 Transact-SQL writetext 命令、 Client-Library 函数、 ct_send_data 或 dbwritetext 和 dbmoretext 这两个 DB-Library™ 函数修改的 text 或非 image 列关联。Replication Server 使用此映像为随后在复制数据库中进行的修改构造主键。 rs_datarow_for_writetext 的语法如下：

```
distribute command_tags applied
 'table'.rs_datarow_for_writetext
 yielding datarow column_list
```

yielding datarow *column_list* 传递列名、非 text 或 image 列的值以及 text 或 image 列的复制状态。复制状态可以为 always_rep、 rep_if_changed 或 never_rep。 *column_list* 字段还传递关于 text 或 image 列的附加信息（称为 text_status）。 text_status 可以是以下关键字之一：

表 B-5：text 和 image 数据的 text_status 值

关键字	说明
tpnull	列的文本指针为空。没有对 text 或 image 列进行修改。
tpinit	在主数据库中进行了修改，从而导致分配文本指针。
hastext	其后是当前 text 或 image 数据值。
notrep	未复制 text 或 image 列。由于数据值未更改并且 text 或 image 列的状态为 replicate_if_changed，因此不需要在复制数据库中执行命令。
zerolen	在主数据库中执行某项操作后， text 或 image 列包含空值。例如，在分配文本指针后， text 或 image 列中可能有数据值，而主数据库上的应用程序将这些值设为空。

rs_insert 和 rs_update 也传递 text 和 image 列的复制状态和其他 text_status 信息。 rs_delete 只传递复制状态信息。

applied 子命令的 rs_writetext 形式传递 text 或 image 数据。rs_writetext 最多可传递 4K 的 text 或 image 数据，因此，可对数据进行分段，然后在多个 rs_writetext 迭代中传递。 rs_writetext 的语法如下：

```
distribute command_tags applied
 'table'.rs_writetext
  append [first] [last] [changed] [with log]
  [textlen=100] column_list
```

- append 表示后面还有更多的 text 或 image 数据段。
- first 标记 text 或 image 列的第一个数据段。
- last 标记 text 或 image 列的最后一个数据段。
- changed 表示 text 或 image 的列值已更改。如果省略 changed 关键字， text 或 image 值不会更改。最少列数功能使用此标志放弃 Replication Server 认为不需要的数据。
- with log 表示将修改记入主数据库的事务日志中。只有 text 或 image 的第一个数据段才需要它。
- textlen 表示 text 或 image 列的总大小。只有 text 或 image 的第一个数据段才需要它。

- *column_list* 包含列名，后跟 text 或 image 数据。数据的开始部分是一个标识标头，其构造顺序如下所示：

  a 波浪 (~) 字符，表示结构化标识。

  b 句点 (.) 字符 （若为 text 数据），或斜线 (/) 字符 （若为 image 数据）。

  c 一个 3 字节字符，它表示在 rs_writetext 命令中传递的 text 或 image 数据段的长度。3 字节长度的计算方法是，将长度转换为 base-64 表示形式，然后将每个数字与基本字符 ! 相加以确保该数字是可打印字符。计算 base-64 数字的公式是在 d3、d2 和 d1 中生成的 3 个数：

  d3 = len/(64*64)
  len = len - (d3*64*64)
  d2 = len/64
  d1 = len%64

  例如，text 段的长度为 126，且 d3= 0、d2=1、d1=62。这些数字加上基本字符 !（基本字符的整数值为 33）就变成 !、 " 和 _ (0+33、 1+33 及 62+33)。于是，结构标识标头可以表示为：~.!"_ 。

- 下面是 LTL 命令的一个示例，其中的 LTL 命令是通过用于更新名为 blurb 的 text 列的 writetext 命令生成的：

```
distribute
 @origin_qid=0x00010000010000,
 @tran_id=0x00018238,
 applied 'texttest'.rs_datarow_for_writetext
 yielding datarow @title_id='BU1032', @price=$90.00,
      @blurb=hastext always_rep,
      @picture = hastext always_rep
distribute
 @origin_qid=0x00010000010001,
 @tran_id=0x00018238,
 applied 'texttest'.rs_writetext
 append first last changed with log textlen = 126
 @blurb = ~.!"_ Straight Talk About Computers is an
 annotated analysis of what computers can do for
 you: a no-hype guide for the critical user
distribute
 @origin_qid=0x00010000010002,
 @tran_id=0x00018238,
 applied 'texttest'.rs_writetext
 append first with log textlen = ...
 @picture = ~/&*^ '0X010203...'
```

```
distribute
 @origin_qid=0x00010000010003,
 @tran_id=0x00018238,
 applied 'texttest'.rs_writetext
 append last @picture = ~/!(* 0x4990...
```

execute 子命令

execute 子命令用于将复制的函数或存储过程调用发送到另一个 Replication Server。此子命令与分发存储过程的首选方法 （应用函数和请求函数）以及较早的方法 （请求存储过程）一起使用。

execute 子命令的语法如下：

distribute *command_tags* execute
{[repfunc] *function* | [*replication_definition*.]*function* |
sys_sp *stored_procedure*} [*param_list*]

- repfunc 关键字 （仅适用于 LTL 103 或更高版本）表示其后的 *function* 名称是与函数复制定义关联的用户定义函数。为复制的存储过程创建函数复制定义时，将同时自动创建一个同名的用户定义函数。这种情况下， execute 子命令将不包含函数复制定义名称。

  对于应用函数， Replication Server 会将 execute repfunc 子命令从主 Replication Server 分发至预订了关联函数复制定义的所有复制 Replication Server。

  对于请求函数， Replication Server 会为函数复制定义将 execute repfunc 子命令从复制 Replication Server 分发至主 Replication Server。

- 如果省略 repfunc 关键字，其后的 *function* 名称将是与表复制定义关联的用户定义函数，而 *replication_definition* 则是复制定义的名称。有关用户定义函数的详细说明，请参见 《Replication Server 管理指南》。

  如果没有 repfunc 关键字， execute 子命令仅适用于与表复制定义关联的请求存储过程。（与表复制定义关联的应用存储过程则使用 applied 子命令。） Replication Server 会为表复制定义将 execute 子命令从复制 Replication Server 分发至主 Replication Server。

  如果 execute 子命令不指定复制定义， Replication Server 将在其系统表中搜索函数名，然后查找关联的表复制定义。如果函数名不唯一且未指定复制定义，将出现一条错误消息，指出函数名对多个复制定义有效。

- *function* 既是用户定义函数的名称，又是复制存储过程的名称。当 Replication Server 收到 execute 命令时，它会将 *function* 名称映射到以前用 create function replication definition 命令或 create function 命令创建的用户定义函数。
- 利用 LTL 200 或更高版本，RepAgent 可使用 sys_sp 将系统存储过程发送至备用数据库。
- *param_list* 是执行该过程时提供的数据值列表。必须将参数值括在括号内。

有关 create function replication definition 和 create function 命令的详细信息，请参见 《Replication Server 参考手册》。有关复制函数和存储过程的详细信息，另请参见 《Replication Server 管理指南》。

处理 rs_marker 函数

执行 rs_marker 存储过程时， RepAgent 对其进行处理，以便 Replication Server 对预订实现周期和热备份应用程序进行同步。

rs_marker 是通过一个名为 *@rs_api* 的 varchar(255) 参数执行的。RepAgent 使用 distribute 命令将该参数传递给 Replication Server，语法如下：

```
distribute command_tags param_string
```

例如，如果客户端使用以下命令执行 rs_marker：

```
rs_marker @rs_api='0x1234567'
```

Adaptive Server RepAgent 将以下命令提交给 Replication Server：

```
distribute command_tags 0x1234567
```

请注意， *@rs_api* 参数不带引号。

sqlddl append 子命令

sqlddl append 子命令（LTL 200 或更高版本）用于将 DDL 命令（如 create table）以原始文本字符串的形式应用到热备份应用程序中。较长的 DDL 可能跨越多条命令，而 sqlddl append 使您可以指定要应用的 DDL 命令的第一个和最后一个文本字符串。

sqlddl append 的语法如下：

```
sqlddl append [ first | last ] ddl_string
```

- first 表示 DDL 序列的第一部分。
- last 表示 DDL 序列的最后一部分。
- *ddl_string* 是 DDL 命令的一部分。

dump 子命令

dump 子命令用于在主节点上启动协调式转储。 RepAgent 从事务日志中检索转储记录，并向 Replication Server 发送一条 dump 子命令，以便将 rs_dumptran 或 rs_dumpdb 函数分发至预订了数据库数据的所有节点。

distribute *command_tags* dump [database | transaction] *database_name*, *dump_label id*

- *database_name* 是被转储的数据库的名称。
- *dump_label* 是包含用于标识转储的信息的 varchar(30) 值。 RepAgent 使用转储日期和时间作为此变量的值。
- *id* 是转储的 varbinary(36) 时间戳。

必须在事务内部发送 dump 命令。

purge 子命令

purge 子命令指示 Replication Server 清除入站队列中的特定打开事务，这些事务的 begin 记录的源队列 ID (*oqid*) 都小于命令中指定的 ID。

distribute *command_tags* purge open_xact to *oqid*

- *oqid* 是一个源队列 ID 号，表示您要清除源队列 ID 号低于该值的所有打开事务。

purge 子命令要求使用 LTL 102 或更高版本。

RepAgent 会话示例

本节包含 RepAgent 和 Replication Server 之间的一个示例对话。本例中有两个事务被传送到 Replication Server。

事务日志包含 “T1” 和 “T2” 两个并发事务。日志记录为：

```
T1: begin transaction
T2: begin transaction
T1: insert into authors ('karsen', '510 534-9219')
T2: update authors set phone = '510 986-7020'
     where name = 'green' and phone = '415 986-7020'
T2: commit transaction
T1: commit transaction
```

注释 LTL 命令通常由 RepAgent 使用 Open Client Client-Library 例程提交给 Replication Server。不过，您也可以使用 isql 以交互方式提交 LTL 命令。

```
distribute
     @origin_time='Dec 10 1992  8:48:12:750AM',
     @origin_qid=0x00000000000000000000000000000001,
     @tran_id=0x000000111111
     begin transaction 'T1' for 'user'/'password'
distribute
     @origin_time='Dec 10 1992  8:48:12:750AM',
     @origin_qid=0x00000000000000000000000000000002,
     @tran_id=0x000000222222
     begin transaction 'T2' for 'user'/'password'
distribute
     @origin_time='Dec 10 1992  8:48:13:750AM',
     @origin_qid=0x00000000000000000000000000000003,
     @tran_id=0x0000001111
     applied 'authors'.rs_insert yielding
         after @name='karsen', @phone='510 534-9219'
distribute
     @origin_time='Dec 10 1992  8:48:13:750AM',
     @origin_qid=0x00000000000000000000000000000004,
     @tran_id=0x000000222222
     applied 'authors'.rs_update yielding
         before @name='green', @phone='415 986-7020'
         after @name='green', @phone='510 986-7020'
distribute
     @origin_time='Dec 10 1992  8:48:14:750AM',
     @origin_qid=0x00000000000000000000000000000005,
     @tran_id=0x000000222222
     commit transaction
```

```
distribute
     @origin_time='Dec 10 1992  8:48:14:750AM',
     @origin_qid=0x00000000000000000000000000000006,
     @tran_id=0x000000111111
     commit transaction
```

可以使用 get truncation 命令从最后一个 distribute 命令检验截断点是否已设置为 origin_qid。

索引

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z