# 第二章 选择必要的高可用性和灾难恢复技术

数据处理是绝大多数应用的核心，在生产环境中，数据库总是扮演着非常重要的角色。任何数据库系统问题都有可能会导致整个生产环境的异常，进而导致不同程度的经济损失。尤其是那些支持实时生产的系统，例如支持制造流水线、订单处理、物流运输、消息传递的应用，几十分钟、甚至几分钟的宕机时间，都有可能对企业造成严重的损失。而另一方面，软件系统在现实生活中始终面临着众多挑战，例如：物理硬件损坏或升级、网络问题、病毒攻击、软件维护和升级等。哪个问题处理得不好，都会导致数据库或者数据服务不可用。因此，如何保证数据库系统的连续可用，一直是各大数据库软件提供商致力研发的领域。

作为一个定位于企业级应用的数据库管理系统，SQL Server提供了多种技术，帮助用户实现他们系统的连续可用性。在每个SQL的新版本里，都或是推出更新的技术，或是对现有的技术进行改进。所以现在用户有了很多选择组合。了解这些技术的适用场合，各自的优点与局限性，对用户规划和实施自己系统的可持续性方案是至关重要的。

在本章中，会介绍什么是“高可用性”和“灾难恢复”，以及SQL Server的四个传统的高可用性和灾难恢复技术：故障转移群集(Cluster)、日志传送(Log Shipping)、复制(Replication)和数据库镜像(database Mirroring)。这四种技术各自独立，又可以组合使用，可以达到不同的效果。在2.6节里，会对这些技术各自的优点和局限性进行分析与比较，指出它们适合的场景。在实际应用中，这些技术经常被组合使用。在本章的2.6节，也会介绍这些技术常见的组合方法，以达到更佳的效果。通过本章的介绍，读者可以了解这四个传统技术的基本概念，适用场景，工作机制，以及常见的故障类型和排错方法。

本章中介绍的内容，对各个SQL Server的版本都基本适用。如果您还在使用SQL 2008，这部分内容也适用您的系统。SQL Server 2012在故障转移群集功能上有了很多改进与完善，在2.2.5节里会有专门的介绍。在第三章里，我们会集中讨论SQL Server 2012新增的一个全新解决方案：AlwaysOn。

## 2.1 什么是SQL Server的“高可用性”与“灾难恢复”

作为一个数据库服务，SQL Server其实可以分成两个部分：

1. SQL Server服务本身

SQL作为一个应用服务，接受用户的连接，处理用户发过来的各项请求等。这部分功能是由一组安装在Windows上的可执行文件来提供的。如果这部分出问题，就需要恢复Windows系统、网络连接、SQL Server的可执行文件、以及SQL的一切相关配置。如何能保证这部分的正常运行，在系统故障时能够快速恢复，是非常重要的。

1. 存储的数据（数据库）

SQL Server将其处理和存储的数据以文件的形式存储在硬盘上。SQL可执行程序需要访问这些文件，才能访问到数据。如果数据文件发生损坏，管理员需要将文件里的内容恢复到“正确”的状态。当SQL Server的数据库文件达到几百GB，甚至上TB时，在磁盘上全部恢复这么大的数据量是很花时间的。所以一定需要使用相关技术，减少数据恢复时间。

而一个服务的生命周期是怎么样的呢？对SQL来讲，它无外乎两种状态：

1. 在线

SQL Server服务正常运行，数据库文件也可以被正常读写。这时候用户可以正常使用SQL Server服务。每个数据库管理员，都希望这个状态越长越好。

1. 宕机

也就是SQL Server由于用户连接无法建立，或者由于SQL服务的可执行文件不能正常运行，或者因为数据库文件不能被正常读写，而产生的宕机时间。严格来讲，宕机时间分成两个阶段：

2.1) 离线

从服务发生宕机问题，到被管理员发现，开始采取措施恢复服务之前。

2.2) 修复

管理员修复SQL Server服务所需要的时间。

所以可以得到这样的公式（图2-1）：

数据库服务的可用时间=在线时间/（在线时间+宕机时间）

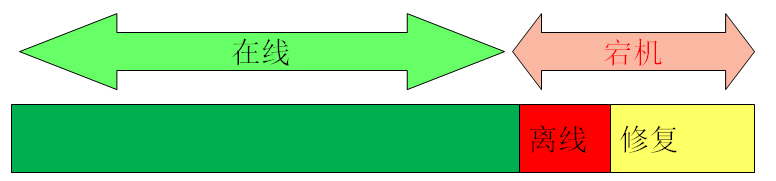


图2-1数据库服务的可用时间

这样分析很有意思，您可以得到这样一个事实：

为了降低宕机时间，管理员可以通过:

1. 努力让系统从来不宕机或极少出现宕机事故，

或者

1. 能够很快地从宕机事故中恢复。也就是说，要能够很快发现SQL Server离线，并且能够快速地修复服务。

提高服务可用性的种种技术和方案，都是围绕着这两点来设计的。

讨论数据库系统的持续可用还需要区分两个概念：“高可用性”和“灾难恢复”。

“高可用性”（High Availability，简称HA）指的是数据库服务能够始终保持在线运行。在数据库系统中的某台服务器发生任何软件和硬件的故障时，还可以保持数据库服务不宕机，或者尽量缩短停机时间（downtime）。换句话说，“高可用性”技术主要关心的，是SQL Server服务本身。它要努力保证在任何一个时间点，客户端都能够找到一个SQL Server服务来响应它们的请求。

“灾难恢复”（Disaster Recovery，简称DR）指的是当数据库中的数据发生损坏或者不可访问时，可以在尽可能短的时间恢复全部数据或者恢复尽可能多的数据，从而保证应用系统的正常运行。有时灾难恢复也简称为“灾备”。“灾难恢复”是要保证数据库里的数据始终可用。理想情况下，假使一份数据因为火灾、地震这样的灾难而损毁，管理员也要能够很快地准备好第二份数据，供SQL Server服务使用。

“高可用性”和“灾难恢复”的概念非常接近，在很多文档中它们都被彼此互相涵盖了。本书倾向于将两者的概念区别开来：“高可用性”针对的是软件或硬件问题造成的数据库系统不可用；而“灾难恢复”针对的是保存在数据库系统中的数据不可用。这两者同等重要。例如用户经常使用的群集技术，它可以在一台服务器发生问题时，自动将SQL Server服务切换到另一台服务器上，从而使宕机时间缩短到几分钟甚至更短。所以它是一个广泛使用的“高可用性”技术。但是，如果数据库存储的磁盘发生损坏，群集的SQL还是不能使用。这时，需要有相应的“灾难恢复”技术，来帮助用户快速恢复数据库中的数据，从而快速恢复SQL Server服务。

SQL Server所提供的各种技术，有些针对于“高可用性”，有些针对于“灾难恢复”，有些对两者都有帮助。所以，用户需要清楚地看到各种SQL Server技术所针对的不同目标，然后根据自己的实际需求，选择最适合自己的方案。在本章接下来内容中，我们会逐个介绍SQL Server 各种传统的“高可用性”和“灾难恢复”技术。

## 2.2 SQL Server故障转移群集

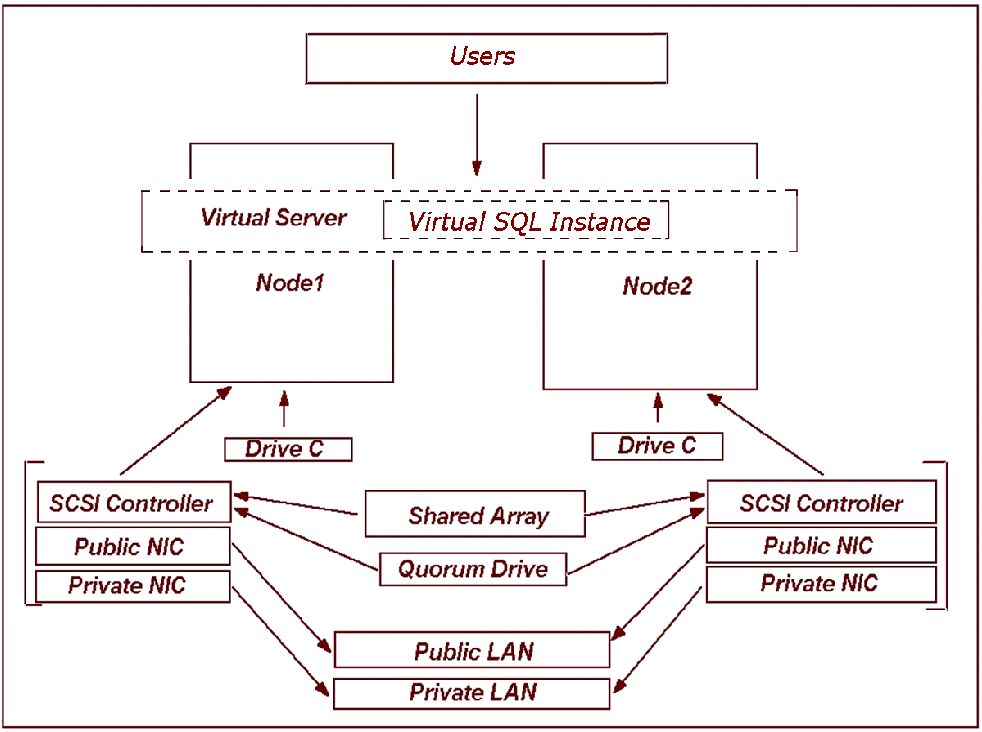
SQL Server使用最广的高可用性技术叫做故障转移群集（SQL Server Failover Cluster）。这是一项基于Windows故障转移群集的一种技术。SQL Server故障转移群集技术集成了微软技术一贯简单易用的特点，在部署和管理上都非常容易，同时又能提供非常良好高可用性，因此目前得到了非常广泛的使用。可以说，它是SQL 2012之前的各个版本，实现高可用性的必选技术。

### 2.2.1 Windows故障转移群集

要了解SQL Server故障转移群集，首先要对Windows故障转移群集有一个基本的了解。下面介绍的内容不求面面俱到地都覆盖Windows群集的知识，但对于一个数据库系统管理员，理解这些概念是非常有用的。

1. Windows故障转移群集的基本组成要素

图（2-2）列出了Windows故障转移群集的基本组成要素：



**图2-2 故障转移群集的结构**

Windows故障转移群集是由多个服务器组成的。每个服务器也被称为“节点”(Node)，每个节点上都运行着Microsoft 群集服务 （MSCS）。这些节点有相同的软硬件配置，并且具有共享的磁盘（Shared Array）。所有需要在几个节点之间共享的东西，例如SQL Server的数据库文件、错误日志等，都会被安放在共享磁盘上。不需要共享的内容则被安放在每个节点的本地磁盘上（即图2-2中的Drive C）。

除了共享磁盘之外，组成群集的多个节点之间通过“私有网络”（private network）和“公共网络”（public network）连接起来。每个节点上有一块私有网卡，这些网卡通过网络互相连接组成了私有网络。节点之间通过私有网络互相发送信号来感知彼此是否已经工作正常，这类信号被称为“心跳线”。一旦某服务器因为某种异常而无法回应信号，这就好像一个人的心跳停止了一样。此时剩余的节点就认为这个节点已经“死了”，于是就把这个节点排除出当前群集。

公用网络，顾名思义是用来被群集外部的资源所使用的一个网络。每个节点上有一块公共网卡，外部资源通过公用网卡来访问这个节点。要注意的是，私有网卡和公共网卡在物理上可以是一块网卡，这种时候群集就通过一个网络来完成私有网络和公共网络的职责，这个网络被称其为混合网络（mixed network）。

一个群集内的所有节点共同组成一个“虚拟”的服务器(Virtual Server)。也就是说，从这个群集的外部来看，只能看到一个服务器，而不是它背后的一堆节点服务器。虚拟服务器具有自己的机器名和IP地址，而这个名称和IP与群集内的任何节点都不同。由于服务器是虚拟的，因此人们称呼它的IP是“虚拟IP”，而它的机器名是“虚拟网络名”（virtual network name）。但事实上“虚拟IP”和“虚拟网络名”都是在DNS服务器上登记在册的，它们本身和一台真实的物理机器的IP和机器名没有任何区别，只是它们对应的服务器是一个虚拟的机器。虚拟IP一定要和公共网络配置在同一个网段里，因为这个IP需要在网络上可以被其他机器访问到。

在Windows群集中，虚拟IP，虚拟网络名，共享磁盘，SQL Server等等，被统称为“资源”。在任意时刻，只有群集中的一个节点能提供用户所需的服务和资源，而其他节点都处于空闲状态。所以，Windows群集无法提供负载均衡的能力。这个技术和NLB是有本质区别的。

那么当用户使用虚拟IP或者虚拟网络名来访问虚拟服务器的时候，到底是哪个节点来为用户提供服务呢？答案是，由“活跃节点”来提供服务。所谓的活跃节点，就是当前拥有该群集资源的节点。

举个例子，假设当前有两个节点，分别是NodeA和NodeB；它们组成的群集的虚拟网络名是Vserevr。在某个时刻，NodeA拥有了虚拟网络名和虚拟IP这两个资源，而服务器B拥有了一个共享磁盘资源DiskX。这种情况下，对虚拟网络名和IP而言，NodeA就是活跃节点，而对于共享磁盘DiskX而言，NodeB才是活跃节点。如果你使用远程桌面工具去连接虚拟网络名Vserver，你实际上是登录到了NodeA上，你可以的本地磁盘上的文件其实是NodeA磁盘中的文件，你运行的程序都是NodeA上安装的程序。虚拟网络名好像一个中转站一样，所有该虚拟服务器的请求都被转向到了NodeA上。但是NodeA上看不到共享磁盘。因为共享磁盘资源当前被NodeB所拥有。您只有直接登录到NodeB上才能看到那个共享磁盘。

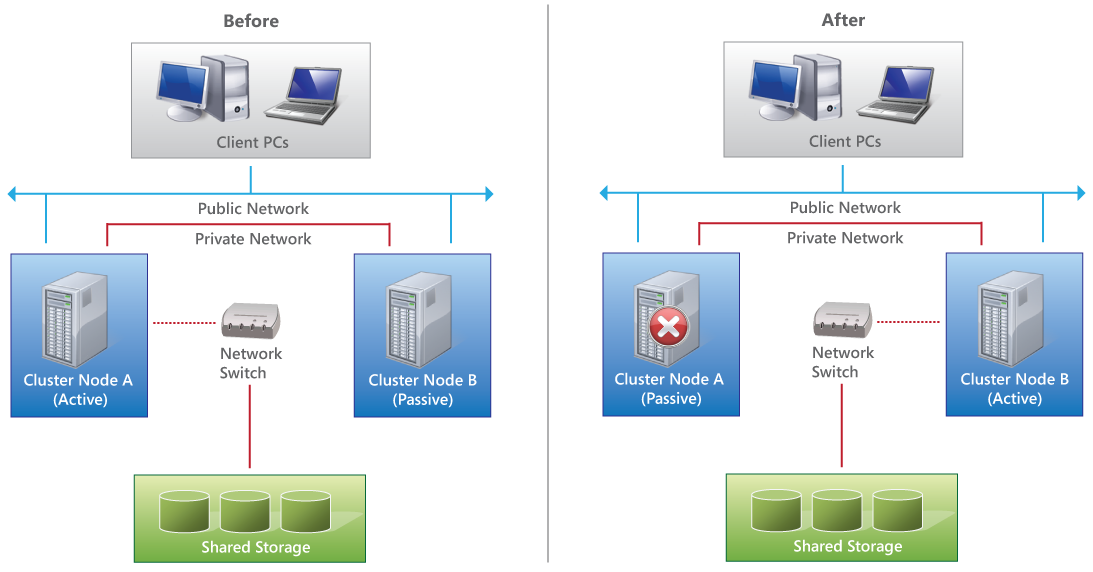
谁是“活跃节点”是Windows群集服务决定的，对用户完全透明。用户不需要关心当前哪个节点正在运行。客户端的连接指令始终都是一样的：指向虚拟IP，或者虚拟网络名。例如用户要通过UNC方式访问共享磁盘上的某个文件，他所发出的指令会是[\\Vsever\DiskX\FolderY\](file:///\\Vsever\DiskX\FolderY\)。

1. 故障转移

接下来再来谈谈群集的“故障转移”（Failover，也称为“切换”）。也就是说，当“活跃节点”服务器出问题的时候，群集服务是怎么实现其“高可用性”的。

还是前面的这个例子，假设此时NodeA拥有DiskX，虚拟网络名和虚拟IP地址三个资源。如果NodeA突然间由于硬件故障蓝屏了，那么Windows群集会立刻探知到这个问题，然后自动发起一个故障转移。转移完成后，这三个资源全部在NodeB上线，也就是被NodeB所拥有（见图2-3）。如果之前有一个用户通过[\\Vsever\DiskX\FolderY\](file:///\\\\Vsever\\DiskX\\FolderY\\) 这样的UNC方式来访问共享磁盘上的某个文件，NodeA的蓝屏对该用户不会造成任何的影响。用户甚至可能没感觉到虚拟服务器内部的故障转移。对他而言通过Vserver里的那个文件一直都是可以访问，而这就是高可用性的目的。

除了由于系统故障所造成的Windows群集的自动切换，系统管理员也可以通过工具或者命令行手动发起切换。



**图2-3 资源的故障转移**

### 2.2.2 SQL Server故障转移群集

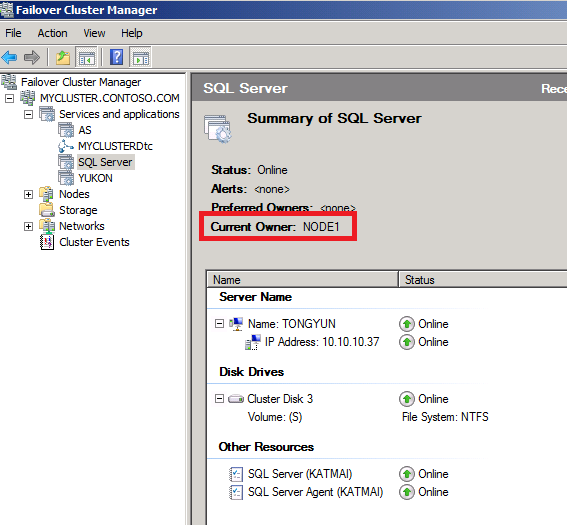
所谓的SQL Server故障转移群集，就是将SQL Server部署在Windows群集中的多个节点上，然后组成一个虚拟的SQL Server实例。这样SQL Server 实例依旧像运行在单台计算机一样显示在网络中。不过它具有一种功能，即在当前运行SQL Server实例的节点不可用时，可以在节点之间进行故障转移，把SQL Server切换到工作正常的节点上去继续为应用程序提供服务。

实现这个功能，需要将SQL Server安装成群集模式，而不是单机模式。安装的具体步骤，请参见本书的第一章。安装成功以后，您就可以看到本节所提到的各个管理界面了。

打开Windows的故障转移群集管理器（Failover Cluster Manager），然后使用Windows群集的虚拟网络名来连接群集。左侧的面板中的“Services and application”节点，下面会列出该Windows群集的所有“资源组”。“资源组”顾名思义就是由一个或者多个资源组成的组。所有的故障转移都是以资源组为单位发生的，在任何时候，每个资源组都仅属于群集中的一个节点，这个节点就是该资源组的“活跃节点”。由于资源组里的资源是一起切换的，所以这些资源应该是彼此关联，并且协同工作来提供某项服务。用户应当尽量避免把无关的资源加入到同一个资源组里。可以简单地把资源组想象成在虚拟服务器上运行的一个个独立的应用程序或者服务，而群集技术为这些应用程序或者服务提供了高可用的特性。SQL Server资源组就是一个很好的例子。

1. SQL Server群集资源组的结构

一个普通的SQL Server群集资源组里会有哪些资源呢？图2-4是一个常见的Windows群集虚拟服务器，其中可以看到该群集上有4个资源组。其中一个资源组的名字叫“SQL Server”，这是一个SQL Server资源组。选中这个资源组，可以看到当前它的拥有者是Node1。这个资源组包含有以下这些资源：



**图2-4 故障转移群集管理器中查看SQL Server资源组的拥有者**

1. **SQL Server网络名和SQL Server IP地址**：

SQL Server群集并不使用Windows群集的虚拟网络名和虚拟IP地址来作为应用程序访问它的接口。在SQL Server的资源组里有该SQL Server实例自己专用的虚拟网络名和 IP地址这两个资源，它们提供了应用程序访问SQL Server时使用的机器名或者IP地址。事实上，无论是Windows群集还是SQL Server群集的虚拟机器名/IP，它们的资源类型都是相同的，只是它们一个服务于Windows虚拟服务器；一个服务于SQL Server群集实例。

1. **SQL Server和SQL Server Agent**

这两个资源是SQL Server故障转移群集的关键资源。它们分别代表了SQL Server服务和SQL Server Agent服务。事实上，SQL Server 2005群集还包含有第三种资源： SQL Server Fulltext Search。但是这个资源在SQL Server 2008以后被并入了SQL Server资源里，因此就只剩下现在这2个资源了。

对于一个SQL Server群集，群集中的所有节点上都会有安装有SQL Server和SQL Server Agent服务以及与服务所对应的二进制文件、注册表键值等。在上图中，Node1是SQL Server的“活跃节点”。如果你到群集中的每个节点上的SQL Server Configuration Manager里查看的话，会发现只有Node1上的SQL Server实例对应的服务是处于启动状态，其余所有节点上服务都是处于停止状态的。

如果SQL Server是默认实例的话，那么SQL Server资源的名字就是“SQL Server”。如果SQL Server是命名实例的话，SQL Server资源名就是“SQL Server (实例名)”。客户端应用程序就是通过“虚拟网络名\实例名”的形式找到它所想要访问的SQL Server实例。

1. **共享磁盘**

共享磁盘资源可以是一块逻辑磁盘，也可以是一块磁盘上的一个mount point。

对于一个SQL Server群集实例，数据库的所有数据文件和事务日志文件（MDF，NDF和LDF），SQL Server和SQL Server Agent的日志文件（ERRORLOG），以及一些其他的文件和目录，都是保存在共享磁盘上的。必须设置共享磁盘和SQL Server资源在一个资源组里，这样就保证了运行SQL Server服务的节点一定能访问到共享磁盘里的数据。

事实上SQL Server资源和共享磁盘资源是具有“依赖”关系的。也就是说在磁盘资源无法在某节点正常运行的时候，SQL Server资源在该节点也无法上线运行。

需要注意的是，一个共享磁盘资源只能属于一个SQL Server实例（事实上，在安装SQL Server群集实例时是不允许把SQL Server安装在一个已经被其他SQL Server群集实例使用的共享磁盘上的）。但是一个SQL Server群集实例可以使用多个共享磁盘。

1. **其他**

除了上述列出的资源之外，SQL Server资源组里还可能包含

1. File Share：如果 SQL Server要使用FileStream，就需要这样一个资源
2. Analysis Services：这个资源和SQL Server/SQL Server Agent资源不同，是属于Generic Service资源。

在每个资源的右侧，您可以看到该资源当前所处的状态。一般而言，每个资源可能处于的状态有：

**上线（online）：**该资源在某个节点上正常工作中。

**离线（offline）：**该资源在某个节点上处于停止工作状态，无法提供相应的服务。

**失败（failed）：**该资源在某节点尝试上线，但是由于某些异常无法上线成功。

**上线挂起（online pending）：**资源尝试进入上线状态，但是还没完全成功上线。

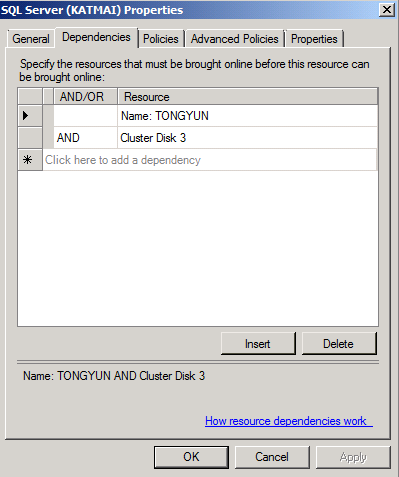
**离线挂起（offline pending）：**资源尝试进入离线状态，但是还没完全成功离线。

以上五个状态中，上线/离线/失败是会长时间保持的状态；除非有人为的操作或者系统异常发生，否则他们会保持这个状态。而两个“挂起”的状态不会长时间保持。经过一段时间之后，它们要么成功“上线”，要么由于某种错误或者由于超时最后进入了“失败”状态。

2. 集群资源的配置

在任意一个资源上点右键，选择“属性”，在弹出窗口上能看到一系列选项卡，这里包含了群集资源的重要设置选项。

这里以SQL Server资源为例。首先来看“Dependencies”选项卡。见图2-5。



**图2-5 SQL Server资源的Dependencies属性页**

这里显示了SQL Server依赖了哪些资源。SQL Server要依赖的资源包括：

* 虚拟网络名资源
* 共享磁盘资源

这两个资源之前的关系是“and”，就是说只有两个资源都online之后，SQL Server资源才可以online。这也符合SQL Server成功启动的要求：（1）能绑定IP地址，（2）能找到系统数据库和错误日志。

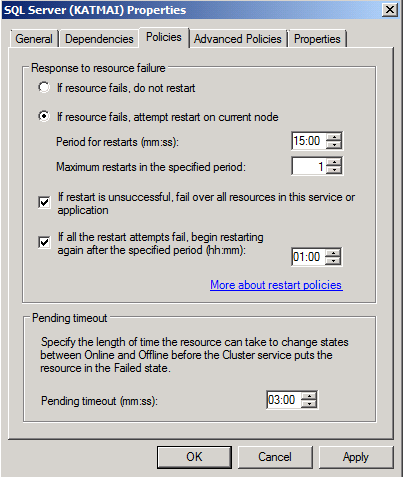
如果去看虚拟网络名的依赖“选项”卡，你会发觉它其实是依赖于虚拟IP资源。这样就组成了一个完整的依赖“链条”。SQL Server资源组里各个资源的依赖关系如下(图2-6)：



**图2-6 SQL Server资源组中属性的依赖关系**

一个资源所依赖的其他资源必须要和这个资源处于同一个资源组里，跨资源组的依赖关系是不存在的。如果用户需要SQL Server群集实例同时可以访问多个共享磁盘资源，必须在SQL Server资源组里添加，然后让SQL Server资源去依赖于它们。没有这样的依赖关系，SQL Server群集实例将无法看到或者使用这些磁盘。

再来看“Policies”选项卡。见图2-7。



**图2-7 SQL Server资源的Policies属性页**

这个选项卡里的选项决定了该资源发生故障转移时的行为。下面介绍一些推荐的设置:

1. 对于SQL Server资源组里的以下类型的资源：

* SQL Server资源
* 共享磁盘资源
* 虚拟IP地址资源
* 虚拟服务名资源

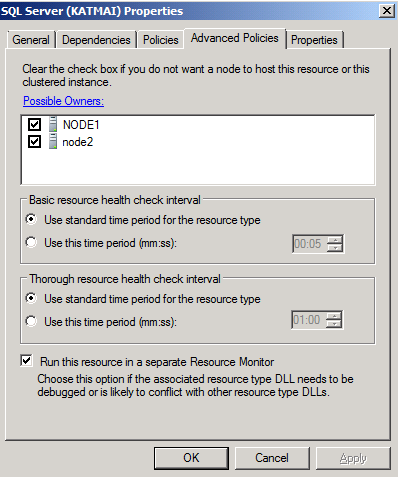
都建议在策略选项卡里保留其默认设置。即确保**If resource fails, attempt restart on current node** 被选中并选择**If restart is unsuccessful, fail over all resources in this service or application。**这样的设置下，上面的资源如果因为软件或硬件故障进入失败状态，会在15分钟内尝试在当前节点重启（一般就是立刻尝试重启，不需要等15分钟那么长），第一次尝试重启失败的话，就会将整个资源组故障转移到另外的节点上。这样的设置被称为“affect the group”。

1. 出于最大化高可用的目的，对以下资源：

* SQL Server Agent资源
* File Share资源
* Analysis Services资源

建议设置为选择 **If resource fails, attempt restart on current node**选项，并且不选择 **If restart is unsuccessful, fail over all resources in this service or application** 选项。这样设置下，上述资源如果由于软硬件异常进入失败状态，并不会导致整个资源组的切换，这些资源不会“affect the group”。对于这些不是非常关键的资源，通常都建议配置成不“affect the group”，这样可以消除那些不必要的故障切换发生，提高数据库服务器的在线时间。千万不要把那些和SQL Server实例无关的资源加入到SQL Server资源组来，并且把它们配置为“affect the group”，因为这样会大大提升SQL Server资源组发生无谓切换的风险。

最后，来看一下“Advanced Policies”选项卡。见图2-8。



**图2-8 SQL Server资源的Advanced Policies属性页**

该选项卡中的“possible owners”部分会列出该资源可以切换到哪些节点上运行。图2-8说明，SQL Server资源可以在Node1或者Node2上（这说明，在安装SQL Server群集的时候，SQL Server被部署到了这两个节点上）。如果任意节点前没有打钩的话，就意味着资源不能在这个节点上上线运行。

在这个选项卡里还有两个重要的选项。一个是“Basic resource health check interval”，另外一个是“Thorough resource health check interval”。Windows Cluster为了每个资源是否工作正常，会使用不同的时间间隔来做的两种不同程度的检查。我们通常把Basic resource health check俗称为“looks alive check”，而把Thorough resource health check称为“is alive check”。我们会在下面一小节里详细介绍这两种检查。

### 2.2.3 SQL Server群集什么时候会发生“故障转移”

SQL Server故障转移群集是如何检测到系统故障，从而触发故障转移群集的呢？

前面说到过，Windows群集是通过做“looks alive check”和“is alive check”这两种不同的检查来判断资源是否工作正常的。如果检查失败，则判定该资源出现异常，然后就按照该资源的“Policies”选项卡里中设置，来进行故障转移。

群集里的每个资源都是一种资源类型。比如，共享磁盘的资源类型是physicaldisk，虚拟IP的资源类型是IP address，SQL Server资源对应的资源类型名字就叫“SQL Server”。根据不同类型的资源，会使用不同的方式进行isalive和looksalive检查。



**图2-9 资源类型、资源DLL和资源的关系**

对于physical disk, IP, network name和DTC这类Windows群集自带的资源类型，它们各自的isalive和lookalive方法都定义在一个Windows群集自带的resource DLL “clusres.dll”中。对于那些非自带的资源（比如说SQL Server），如果它们有自己专属的资源类型并且有专属resource dll，就可以在resource DLL中定义自己的isalive和looksalive检查方法。有些资源没有特定的资源类型，我们称这类资源的资源类型为“通用服务”（Generic Service）。对于Generic Service类型的资源，Windows群集依旧使用clusres.dll来作为它们的resource dll。Clusres.dll里有针对Generic Service的“标准”isalive和lookalive检查方法。Windows群集服务进程clussvc.exe会产生名为RHS.exe的进程，RHS.exe会装载resource dll，并且调用dll中定义的方法来检查相应resource的状态。一个资源其属性的Advanced Polices选项卡中如果没有勾选“run this resource in a separate Resource ”，它的resource dll就会装载在一个默认的RHS.exe中；反之，则会有一个单独的RHS.exe来装载该资源的resource dll。所以在任务管理器中你可能看到多个RHS.exe进程。

在安装SQL Server的时候，会安装两个SQL Server自己的resource dll ： sqsrvres.dll和sqagtres.dll，它们分别服务于SQL Server资源和SQL Server Agent资源。一般只会把SQL Server资源配置成affect the group模式，因此要了解SQL Server群集什么情况下会故障转移，就要了解sqsrvres.dll是怎么定义looksalive和isalive方法的。事实上，从SQL Server 2000到SQL Server 2008 R2，sqsrvres.dll中定义的looksalive和isalive方法都是类似的。具体来讲：

**Looksalive：**通过服务控制管理器（Service Control Manager，简称SCM）来检查SQL Server服务在活跃节点是否处于“启动状态”。根据SQL Server资源的Advanced Polices选项卡中的设置，这个检查默认是每5秒钟做一次。

**Isalive**：根据SQL Server资源的Advanced Polices选项卡中的设置，这个检查默认是每60秒钟做一次，也就是说每12次Looksalive检查就会伴随一个Isalive检查。SQL Server需要Isalive检查，是因为即使SQL Server服务是正在运行状态，也不能说明SQL Server就可以良好地响应应用程序的请求。有的时候可能整个SQL Server已经挂起了，但是服务的状态还是“启动”。所以需要Isalive check来进一步检查SQL Server的状态。此外，一旦lookalive检查的结果失败，Windows群集服务就会立刻触发Isalive检查。

在SQL 2012之前，Isalive所做的事情很简单，Windows群集服务会使用TCP/IP或者Named Pipes来连接SQL Server群集实例。连接上之后，运行一句命令“select @@servername”。如果该语句成功返回结果，那么Isalive检查就成功了。

如果连接不上SQL Server群集实例或者语句运行失败，那么Isalive检查失败。此时Windows群集会再做3-5次（根据Windows的版本和设置不同）Isalive检查。如果这些检查都失败了，就要根据Policies选项卡中的设置开始进行故障转移。

您可以把故障转移简单地想象成SQL Server服务的重启，所不同的是故障转移的时候，SQL Server服务是在当前节点停止，然后在另一个节点上启动起来。因此故障转移所花费的时间和SQL Server服务重启的时间是差不多的。当然共享磁盘和虚拟网络名等资源在另一个节点上线也会额外花费一点时间，不过在大多数情况下这部分时间是比较短的。另外，由于故障转移一般是意外发生的，所以您要预期SQL Server切换到新节点以后，还需要一段时间来做数据库的recovery。

前面也提到过，除了SQL Server和SQL Server Agent以外，SQL Server资源组里可能还会有Analysis Service资源。但是和SQL Server/SQL Server Agent不同的是，Analysis Service资源没有自己的资源类型，也就是说它是一个Generic Service（通用服务）。Analysis Service的isalive和looksalive检查就使用的是clusres.dll中定义的通用服务检查方法。

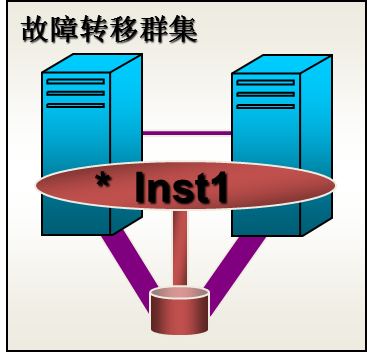
SQL Server的诸多服务和组件中，SQL Server、SQL Server Agent以及Analysis Service这三个服务，无论是有自己专属的资源类型还是通用服务，都是被设计为可以通过resource dll形成群集资源。这种类型的服务被称为cluster-aware。SQL Server还有很多其他资源，比如SQL browser, Reporting Service等，它们被设计成无法通过任何resource dll在Windows群集中形成资源，所以它们不是cluster-aware的。对于不是cluster-aware的服务，即使被安装在了群集的节点上，Windows依旧把它当成是安装在了一个单机环境中，它无法具有故障转移的功能。

需要提一下的是Integration Services是一个比较特别的服务。Integration Services本身不是cluster-aware的服务，但是用户可以通过一些步骤手动把它配置成一个群集资源。但是这样配置出来的Integration Services群集资源不是一个真正的资源，是不具有自动故障转移功能的，因此微软并不推荐这么做。更多的信息可以参考下面两篇技术文章：

* <http://msdn.microsoft.com/en-us/library/ms345193.aspx>
* <http://support.microsoft.com/kb/942176>

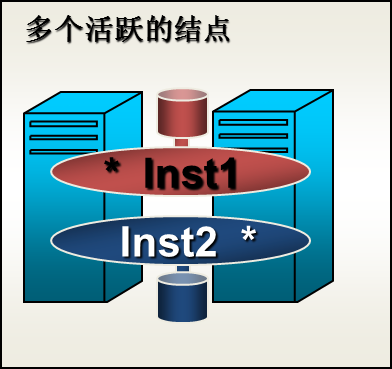
### 2.2.4 SQL Server群集的拓扑结构

最简单的SQL Server故障转移群集拓扑结构就是“活跃/非活跃”群集（图2-10）。这种结构下群集有2个节点，用户在群集上安装一个SQL Server群集实例，该实例的“可能的所有者”（possible owners）包含上述两个节点。这样任意时间只有一个节点上有SQL Server服务在运行，而另一个节点就是“非活跃”节点。这种配置的优点是结构简单明了，无论SQL Server运行在哪个节点上都能获得同样的性能表现。缺点是总有一个节点处于空闲状态，浪费了50%的硬件资源。



**图2-10 “活跃/非活跃”群集**

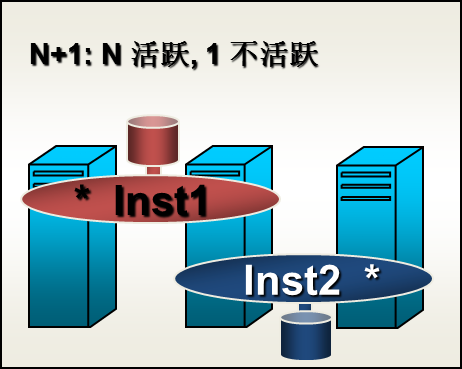
另一种拓扑结构是“活跃/活跃”群集（图2-11）。还是以一个2节点的群集为例。这个时候用户在群集上安装两个SQL Server群集实例，每个实例的“可能的所有者”都包含群集两个节点。在正常情况下，两个实例分别运行在不同的节点上。这样两个节点就都是“活跃”节点。



**图2-11 “活跃/活跃”群集**

这种结构的优势是两个节点的硬件资源都能被充分利用，节约成本。缺点是，一旦某个节点发生故障转移，就会发生另一个节点上同时运行了两个SQL Server实例的情况。此时，这两个实例可能会争用这个节点上的CPU，内存，I/O等资源，导致两个实例的性能都受到影响。有时候可能两边的用户都不能接受。因此要尽快解决异常节点上的问题，尽早把发生故障转移的实例切换回去。

在此基础上，我们来介绍所谓的N+1结构，即N个活跃节点加上1个非活跃节点（图2-12）。以3个节点的群集为例，在上面安装两个SQL Server群集实例，每个实例的possible owner包含群集中的两个节点，但是只有一个节点是两个实例共有的。在正常情况下，两个SQL Server都运行在非共有的那个节点上，互不干涉。一旦某个节点发生故障转移，就会切换到那个共有的非活跃节点上。



**图2-12** N+1群集

这个结构是一个介于“活跃/非活跃”和“活跃/活跃”之间的一种方案。相对于“活跃/非活跃”，它浪费的节点资源比较少（1/N+1）。另外，两个以上的节点同时发生故障转移，需要同时切换到共有节点的概率是比较低的，因此也在一定程度上解决了“活跃/活跃”结构的性能问题。

无论什么样的拓扑结构，有两点是不会变的：

1. SQL Server群集无法提供数据库端的负载均衡功能。作为“活跃/活跃”群集，其实是几个独立的数据库实例，它们彼此间没有联系。再次强调，群集技术只是一个提供”高可用性”的技术，而不是提升性能的技术。
2. 无论群集有几个节点，对于某个数据库实例，它只有一份数据。一旦数据本身出现问题，群集对此无能为力。所以，群集技术不是一个提供数据“灾难恢复”的技术。对重要的数据库，仅仅使用群集技术是不够的。

### 2.2.5 SQL 2012对故障转移群集的改进

SQL Server 2012群集基本沿袭 了SQL Server 2008群集以来的一系列特点。不过它也带来了一些新特性，使得群集具有了更加强大的功能以及更高的可用性。现在就让我们来看看SQL Server 2012群集都有哪些新东西。

**1. 多子网群集的支持**

从Windows Server 2008开始，故障转移群集开始支持所谓“多站点（multi-site）群集”。也就是说，组成一个群集的节点可以被安置在相隔很远的不同的站点。此外，不同站点可以处于不同的子网，因此多子网（multi-subnet）群集也得以实现。在地理上相隔很远的站点进行群集有时被称为拉伸群集（Stretch Clustering）。

对于一个多子网的群集，需要为群集里的每个站点都配置一个虚拟IP地址，而“虚拟网络名”资源会依赖所有这些IP地址资源。当群集运行在某个站点上时，只有对应该站点子网的虚拟IP地址资源可以处于上线状态，其余IP地址资源由于在该子网内无法被成功分配，因此都会处于“失败”或“离线”状态。从Windows Server 2008群集开始，用户可以把虚拟网络名资源配置成以“OR”的关系来依赖这些IP地址资源。因此即使只有一个IP地址可以上线，“虚拟网络名”资源也可以成功上线。可以说是，Windows Server 2008群集对OR依赖关系的支持，是实现多站点和多子网群集的一个关键。

可是，虽然Windows Server 2008群集支持多子网，SQL Server 2008和SQL Server 2008 R2群集却并不支持多子网。这是因为SQL Server 2008本身不支持OR的依赖关系。从SQL Server 2012开始，相应部分遗留的代码得到了重写，因此SQL Server支持多子网群集的障碍也得到了扫除。目前，只有运行在Windows Server 2008 R2群集上的SQL Server 2012群集才完全支持了多子网的功能。

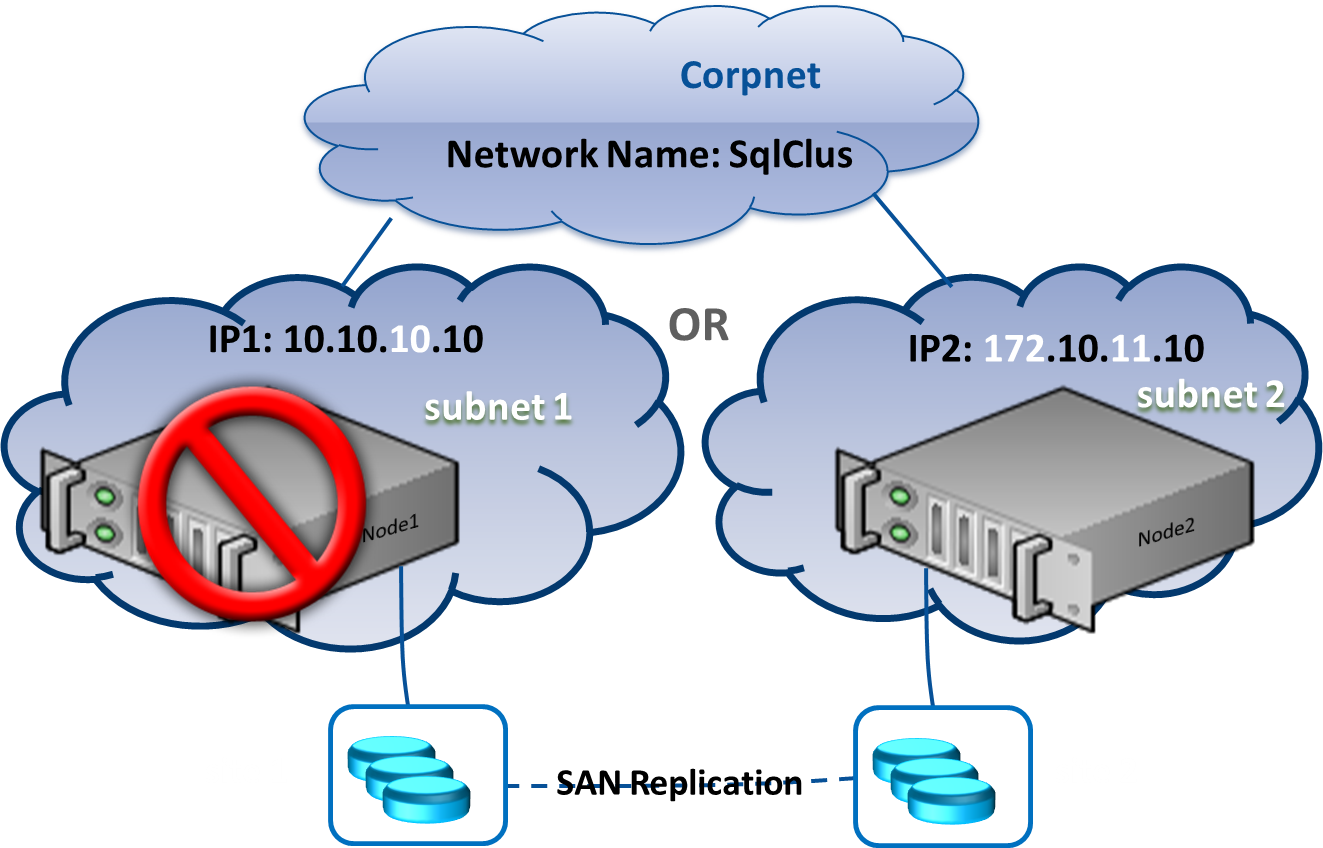


图2-13 SQL Server 2012跨子网故障转移群集结构示意图

在SQL Server 2012中，如果SQL Server发现“虚拟网络名”资源所依赖的某个IP地址不能上线，就会在SQL Server的错误日志中记录以下信息：

SQL Server could not listen on IP address [156.69.72.165] because the cluster resource ‘SQL IP Address 2 <inst>’ is not online (state=3). This is an informational message and may indicate that resource ‘SQL Network Name (CHSU-SQL2)’ has OR type of dependency on several IP addresses some of which are currently offline or in a failed state. Further action is only required if it is generally possible to bind the IP address of the cluster resource ‘SQL IP Address 2 <inst>’ to a network segment on the current hosting node.

这条信息清楚的显示了IP地址[156.69.72.165]的状态，并说明了SQL Server没有绑定该IP地址。

对于成功上线的IP地址，错误日志中依旧会记录SQL Server绑定该IP的信息。该信息和之前版本的SQL Server是一样的：

Server is listening on [10.22.13.16 <ipv4> 60362].

以下是使用多子网的 SQL Server故障转移群集的一些示例配置：

* SQL Server群集 包括 Node1 和 Node2。 节点1连接到 Subnet1。 Node2 连接到 Subnet2。 SQL Server 安装程序将此配置视作一个多子网群集，并且将 IP 地址资源依赖关系设置为 OR。
* SQL Server群集包括 Node1、Node2 和 Node3。 Node1 和 Node2 连接到 Subnet1。 Node3 连接到 Subnet2。 SQL Server 安装程序将此配置视作一个多子网群集，并且将 IP 地址资源依赖关系设置为 OR。 因为 Node1 和 Node2 位于同一子网上，所以此配置还提供本地高可用性。
* SQL Server群集包括 Node1 和 Node2。 Node1 连接到 Subnet1 上。 Node2 同时连接到 Subnet1 和 Subnet2 上。 SQL Server 安装程序将此配置视作一个多子网群集，并且将 IP 地址资源依赖关系设置为 OR。
* SQL Server群集包括 Node1 和 Node2。 Node1 连接到 Subnet1 和 Subnet2。 Node2 也连接到 Subnet1 和 Subnet2。 SQL Server 安装程序将 IP 地址资源依赖关系设置为 AND。**要注意，这种配置不被视作多子网故障转移群集，因为群集的节点实际上是处于完全相同的子网环境中。**

我们所讨论的多子网SQL Server只限于IP地址资源依赖关系是OR的情况，即各个节点彼此处于不同的子网环境，多个IP地址不能在一个节点上全部上线的情况。

SQL Server 2012对多子网的支持，使得它更加适用于部署多站点群集。使用多子网的SQL Server群集，即使某个子网由于网络设备的突然故障而整体瘫痪，也能通过故障转移使用其他子网中的节点继续运行数据库服务，因此在抵御网络故障上具有更强的能力。多子网的SQL Server群集在地理上分散，使得它能更好地抵御例如自然灾害，大规模停电等重大异常。此外，多子网的群集不是在有所有节点间共享数据存储，所以需要在多个子网之间进行硬件存储器级别的数据复制，这样就获得了多个可用数据的副本。因此在硬件支持的前提下，多子网SQL Server群集除了具备高可用性之外，还提供了数据灾难恢复解决方案。

**2. RegisterAllProvidersIP**

多子网的SQL Server一般情况下，一个虚拟网络名对应多个IP地址，而只有其中一个IP地址资源是上线状态的。那么在DNS上到底是只注册了那个上线的IP地址，还是全部IP地址都注册了呢？这取决于“虚拟网络名”资源的一个私有属性，叫做RegisterAllProvidersIP。

如果RegisterAllProvidersIP被设置为0，那么只有上线的IP地址会被注册到DNS上，网络名也只会被解析到上线的IP地址。这看起来没什么问题，但是在发生故障转移的时候可能会带来一些麻烦。假设群集发生了故障转移，从一个子网中的节点切换到了另一个子网中的节点。当故障转移完成后，SQL Server资源组中的虚拟网络名没有发生变化，但是开始启用新子网中的IP地址。这时，客户端需要知道新的IP地址，才能连上SQL Server。这通常要借助DNS服务器。但是DNS服务器上的缓存可能会造成DNS的更新延迟，在一段时间之内DNS服务器依旧会将服务器名解析成之前的IP地址（即老的子网的IP地址）。于是在这段时间里，即使SQL Server已经恢复运行，连接SQL Server的请求还是会由于无效的IP地址而失败，这会增加客户端应用恢复到SQL Server的连接所需的时间。事实上，为了避免这类问题，Windows Server 2008 R2群集在虚拟网络名资源上线的时候会立刻给DNS服务器发一个注册请求，目的就是尽快将服务器名指向新的IP地址。不过如果在DNS服务器端发生任何延迟，影响注册请求的完成，那么在这一段时间内，访问SQL Server的请求还是会受到影响的。

为了彻底避免DNS更新延迟带来的问题，SQL将虚拟网络名资源的RegisterAllProvidersIP的默认值设置为1。也就是说默认情况下，无论这个IP地址处于什么状态，多子网的群集依旧会在 DNS 服务器上注册虚拟网络名到所有IP地址之间的解析关系。当客户端应用要使用虚拟网络名来连接SQL Server的时候，它会从 DNS 服务器检索所有已注册的 IP 地址，并尝试连接到这些地址。这样，多子网SQL Server群集中的客户端恢复连接的时间不再受 DNS 更新延迟的影响。

默认情况下，客户端会按顺序去尝试DNS上所有 IP 地址。 当客户端在其连接字符串中使用MultiSubnetFailover=True 参数时，客户端会改为并行地尝试所有IP 地址，并使用第一台响应它的IP地址来连接SQL Server服务器。这样的机制有助于在发生故障转移时最大程度地减少客户端恢复连接的延迟。目前，只有以下驱动支持在进行TCP连接时使用MultiSubnetFailover参数：

* SQL Native Client 11.0
* .NET Framework 4.02 或更高版本中的SQL Client（必须在连接字符串中指定数据库实例的 TCP 端口）。
* Microsoft JDBC Driver 4.0 for SQL Server

对于其他的驱动，你无法在连接字符串中使用 MultiSubnetFailover 参数，也就是说它们只能顺序去尝试所有IP地址。这增加了遇到连接数据库超时错误的风险。因此，对那些访问多子网SQL Server群集的客户端，建议在连接字符串中适当地增加连接超时的阈值。通常情况下的建议是：资源组中每增加一个IP地址资源，就增加 21秒的连接超时。 这样，客户端在尝试重新连接SQL Server时，它会有足够的时间来依次访问所有 IP 地址。

**3. 存放数据库的物理位置**

SQL Server 2005 以前版本的SQL Server 故障转移群集，数据库的所有数据文件和日志文件都必须被放在共享磁盘上，包括用户数据库和系统数据库。

SQL Server 2008和SQL Server 2008 R2将系统资源数据库（resource DB）与其他的系统数据库分隔开来，单独存放在了每个实例对应的Binn目录下，和其他的SQL Server可执行文件和DLL文件放在了一起。这是因为resource 数据库是只读的不可修改的，它仅是用来提供SQL Server所有的系统对象，因此从功能上来看resource数据库更接近一个SQL Server的DLL而不是一个系统数据库。因此SQL Server 2008和SQL Server 2008 R2的群集中resource数据库的文件是存放在本地磁盘上，而不是共享磁盘上。

从SQL Server 2012开始，除resource数据库以外的所有系统数据库（master，msdb，model和tempdb）及用户数据库不但可以被存放在共享磁盘中，也可以被存放在共享文件夹中。如果你的SQL Server 2012群集使用共享文件夹来存放数据库，你必须使用“\\Servername\ShareName\...”这样的通用命名约定 (UNC) 路径格式。不可以使用环回路径（loopback path，例如 \\localhost\.. \）、管理共享（admin share，例如 \\servername\x$）或映射网络驱动器。共享文件夹可以位于Windows 文件服务器或第三方 的SMB（Server Message Block） 存储设备承载。 如果使用 Windows 文件服务器，该 Windows 文件服务器版本应为 2008 或更高。

相比较共享磁盘，共享文件夹有三个主要的优点：

1. 免去了为共享磁盘配置SAN或者iSCSI等一系列繁琐的操作步骤。
2. 节省了SAN存储硬件。你可以使用任何存储器来提供共享文件夹。
3. 使用共享磁盘，一个Windows群集上可以安装的SQL Server 群集实例的数量取决于可用驱动器号的数量（无论是否使用mount point）。 如果只对操作系统使用一个驱动器号，则最多只能有 25 个 SQL Server 实例。使用共享文件夹的话，就可以突破驱动器号数量的限制，在一个Windows群集上可以安装最多50个SQL Server群集实例。

下图（2-14）是SQL Server 2012群集的安装截图。你会发现在安装的过程中，安装程序并不强制要求你必须选择一个共享磁盘。图中显示了两个可用的共享磁盘，但是你可以两个都不勾选。

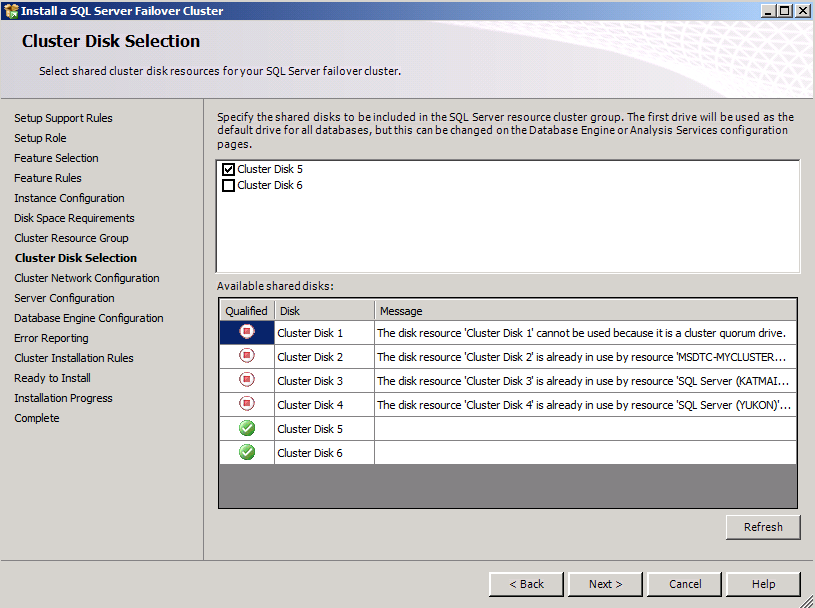


图2-14 安装SQL Server 2012群集 – 选择共享磁盘

在指定数据库存在位置的步骤中（图2-15），你就可以使用共享文件夹了。

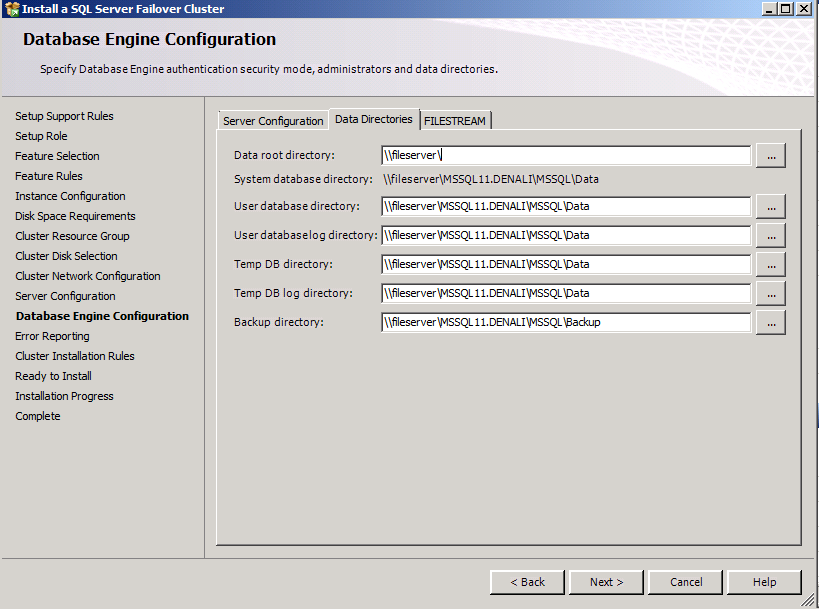


图2-15 安装SQL Server 2012群集 – 输入共享文件夹作为数据库文件的存放位置

SQL Server 2012群集的另一个变化是tempdb存放的位置。之前版本的SQL Server群集，要将所有用户数据库和系统数据库，包括tempdb，都保存在共享磁盘上。这样能确保所以节点都使用同一份数据。故障转移后无论哪个节点变成活跃节点，它所访问的数据都是一致的。但是tempdb是一个特殊的数据库，它在每次SQL Server启动时都会被重新创建。也就是说，故障转移后新的活跃节点得到的永远是一个新的tempdb，它不包含任何以前存在于tempdb中的数据。因此tempdb数据库其实不需要被群集的节点所共享。

SQL Server 2012群集允许用户在安装时将tempdb的存放位置设置在本地磁盘上。这样当SQL Server资源组在某个节点上线时，就会在该节点的本地磁盘上创建新的tempdb的数据文件和日志文件。除了tempdb和资源数据库以外的其他数据库依旧存放在共享磁盘或SMB文件共享上。

从下面的图（2-16）中你可以发现SQL Server 2012群集的安装程序中多了Temp DB directory和Temp DB Log directory这两个文本框，它们允许你为tempdb单独设置区别于其他数据库的存放路径。

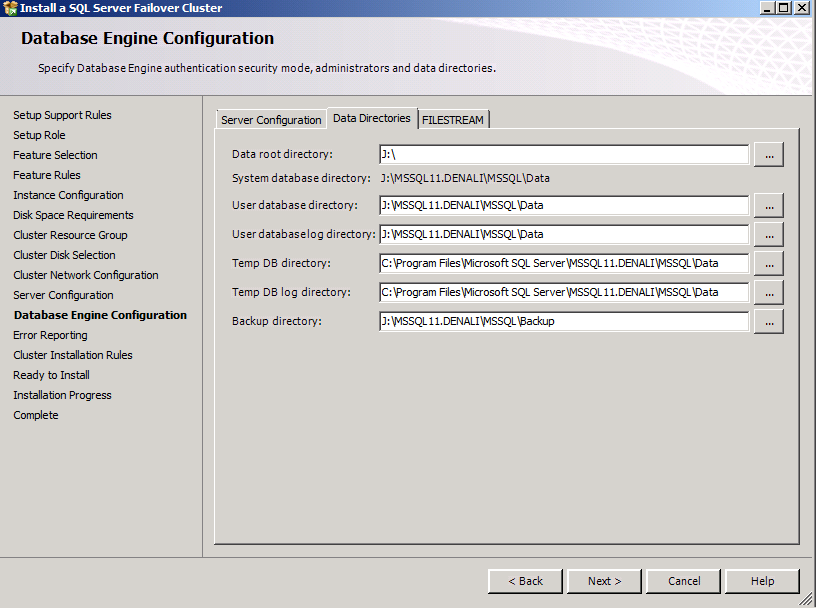


图2-16安装SQL Server 2012群集 – 将tempdb放置在本地磁盘上

将tempdb存放在本地磁盘可以节约共享磁盘的空间。另外，tempdb上一般会发生较多的I/O操作，所以将tempdb存放在本地磁盘还可以减轻共享磁盘的负载，有助于提高其他数据库的I/O性能。

如果将tempdb存放在本地磁盘的目录中，你需要确保群集中的所有节点上都有这个目录并且SQL Server的启动账号对这个目录有读写权限。否则一旦发生故障转移，SQL Server就可能会因为无法创建tempdb而不能上线。

**4. 新的Resource DLL**

前面介绍过SQL Server群集使用两种检查方式来确保群集正常工作并在需要的时候触发故障转移。一种是looksalive检查，它仅是简单的去检查活跃节点上SQL Server服务是否还在运行。另一种是Isalive检查，它是由群集服务开启一个连接到SQL Server里，并执行语句select @@servername，根据语句是否成功执行来进一步判断SQL Server是否依旧在正常提供服务。

很多情况下SQL Server发生故障转移就是由Isalive检查失败所触发的。管理员在故障转移发生以后，需要检查SQL Server日志和Windows系统日志，以了解为什么Isalive检查会失败，当时SQL Server出了什么问题。但是，很大一部分故障转移产生的原因是SQL Server当时出现了性能问题，以至于无法响应Isalive检查。很多性能问题，如CPU使用率100%，工作线程用尽，都不会记录在错误日志中。由此带来的问题就是，数据库管理员能从群集日志或Windows的事件日志中判断出SQL Server当时可能发生了性能问题，但无法得知具体是什么样的性能问题，于是也就无法采取措施来防止问题再次发生。

有时候SQL Server性能问题并不是十分严重，而且可能在很短的时间内就能自动恢复。用户可能不希望在这种情况下，Isalive检查失败就立刻触发故障转移。但是旧有的Isalive检查没有提供客制化的接口，用户或管理员无法自己指定SQL Server发生故障转移的场景。

SQL Server 2012群集使用了全新的Resource DLL（名字依旧是sqsrvres.dll）。新的Resource DLL中，Looksalive检查基本保持了其原有的功能，但是Isalive检查方式发生了变化。新的Isalive不仅依赖检查群集服务和SQL Server之间的连通性来判断SQL Server的健康状况，它还依赖SQL Server内部进行自我诊断，并将诊断结果报告给Resource DLL。总的来说，新的Resource DLL有四个特性：

* 旧的Resource DLL在每次执行Isalive检查时生成一个连接到SQL Server，执行完select @@servername后就关闭连接，直到下次执行Isalive检查时再建立新的连接。而新的Resource DLL会在SQL Server资源上线后就始终使用一个独立的线程来维持一个到SQL Server的持续连接，连接不会中断，直到SQL Server资源进入离线状态。
* 新的Resource DLL改为执行一个全新的存储过程 “sp\_server\_diagnostics”来执行Isalive检查，判断服务器健康情况。Sp\_server\_diagnostics会收集SQL Server当前的诊断信息并返回给resource DLL并记录在扩展事件日志中，缺省情况下sp\_server\_diagnostics每20秒运行一次。SQL Server 2012群集的Isalive检查就依赖sp\_server\_diagnostics返回的信息来判定SQL Server是否工作正常，默认Isalive检查间隔依旧是60秒。可见，这种检查比仅运行select @@servername要可靠很多。
* 新的Resource DLL允许用户自己指定发生故障转移的条件。一旦Isalive或Looksalive发现SQL Server实例处于不健康的状态，那么resource DLL就会将当前的健康状态与用户指定的故障转移条件做对比，决定是否应该发生故障转移。客户可以根据系统的要求，设置什么情况下SQL会做故障转移。
* Resource DLL会保留一段时间的sp\_server\_doagnostics历史诊断信息。这些信息可以更有效地帮助管理员找到发生故障转移的原因。

Resource DLL会根据SQL Server资源名到注册表里查询SQL的版本信息，以决定当前的SQL实例是否是2012版本，是否使用新的扩展功能。这意味从SQL Server 2012开始，管理员不能轻易更改SQL Server的资源名（图2-17），否则就可能会发生Resource DLL找不到该实例的版本、造成SQL Server资源不能上线的问题。

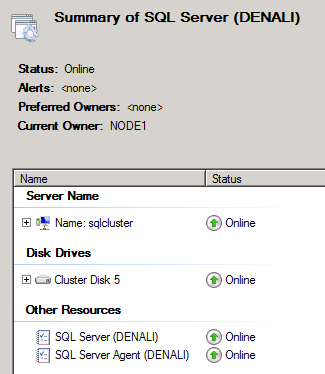


图2-17 SQL Server资源名

因为SQL Server资源类型使用了新的resource DLL，你会发现SQL Server资源有了一些新的私有属性。图2-18是SQL Server 2012群集资源私有属性的截图。

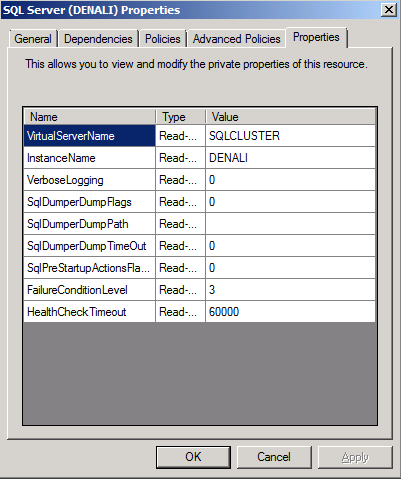


图2-18 SQL Server资源的私有属性

其中大部分属性在以前版本的SQL Server群集中都存在，它们的作用也没什么变化。这里我们重点介绍下新的resource DLL引入的两个新属性。

**FailureConditionLevel**

SQL Server 2012群集定义了从0到5总共6个级别的故障转移条件。用户可以通过设置SQL Server资源的FailoverConditionLevel属性来指定SQL Server在满足何种条件下发生故障转移。对于级别 1-5，每个级别除了自己的条件外，还包括之前级别的所有条件。 这意味着，级别越高，故障转移或重新启动的几率就越大。 下表（2-1）介绍了这些故障条件级别：

|  |  |  |
| --- | --- | --- |
| 级别 | 名称 | 条件 |
| 0 | 无自动故障转移或重新启动 | * 无论发生任何故障，都不会触发故障转移或重新启动。 但是诊断信息依旧被记录在LOG文件夹中的SQLDIAG文件里。此级别仅适用于系统维护目的。 |
| 1 | 服务器关闭时进行故障转移或重新启动 | 当满足以下条件时将触发服务器重新启动或故障转移：   * SQL Server 服务停止。 |
| 2 | 服务器不响应时进行故障转移或重新启动 | 当满足以下任意条件时将触发服务器重新启动或故障转移：   * SQL Server 服务停止。 * SQL Server 实例不响应（Resource DLL 在 HealthCheckTimeout 设置时间内未收到来自 sp\_server\_diagnostics 的数据）。 |
| 3 | 出现严重服务器错误时进行故障转移或重新启动 | 当满足以下任意条件时将触发服务器重新启动或故障转移：   * SQL Server 服务停止。 * SQL Server 实例不响应 * sp\_server\_diagnostics 返回“系统错误”。 |
| 4 | 出现中等服务器错误时进行故障转移或重新启动 | 当满足以下任意条件时将触发服务器重新启动或故障转移：   * SQL Server 服务停止。 * SQL Server 实例不响应。 * sp\_server\_diagnostics 返回“系统错误”。 * sp\_server\_diagnostics 返回“资源错误”。 |
| 5 | 对于任何合格的故障条件进行故障转移或重新启动 | 当满足以下任意条件时将触发服务器重新启动或故障转移：   * SQL Server 服务停止。 * SQL Server 实例不响应。 * sp\_server\_diagnostics 返回“系统错误”。 * sp\_server\_diagnostics 返回“资源错误”。 * sp\_server\_diagnostics 返回“query\_processing 错误”。 |

表2-1 FailureConditionLevel的定义

级别3是FailureConditionLevel的默认值，它最接近于老的Isalive检查的故障转移条件。你可以随时更改FailureConditionLevel的级别，不需要重新启动SQL Server就可以立刻生效。从上面的表格你能看到，sp\_server\_diagnostics所返回的诊断信息对于判断是否应当发生故障转移是至关重要的。在下一节我们会详细介绍sp\_server\_diagnostics。

**HealthCheckTimeout**

这个属性决定了 resource DLL 在认定 SQL Server 实例不响应之前应等待服务器返回sp\_server\_diagnostics诊断信息的时间。HealthCheckTimeout的默认值是60000毫秒。

一旦SQL Server群集资源上线了，resource DLL就会使用**max(5, (HealthCheckTimeout/3))**来获得一个sp\_server\_diagnostics的运行时间间隔。也就是说sp\_server\_diagnostics的查询时间间隔是(HealthCheckTimeout/3)，但不会短于5秒钟。一旦运行时间间隔确定下来，resource DLL就建立连接到SQL Server，并发出语句**sp\_server\_diagnostics <运行时间间隔>**。有了<运行时间间隔>这个参数，sp\_server\_diagnostics就会按这个间隔自动地去不断获取诊断信息并使用那个持续的连接将信息返回，无需resource DLL反复去执行sp\_server\_diagnostics。

Max(5, (HealthCheckTimeout/3))不但是sp\_server\_diagnostics的运行间隔，也是它的查询超时时间。一旦sp\_server\_diagnostics在这个时间范围内无法获得返回结果，那么就产生超时错误，然后开始一次新的sp\_server\_diagnostics查询。因此，默认设置下resource DLL在认定SQL Server发生性能问题或宕机问题前，会尝试执行并等待三次sp\_server\_diagnostics。

而resource DLL连接SQL Server的连接超时是被硬编码为30秒的。不过，在达到HealthCheckTimeout所规定的超时时间之前，resource DLL会不断尝试去连接SQL Server。

对于SQL Server性能慢或者宕机的问题，在没有配置生成dump的情况下，群集的最长响应时间就是（Isalive检查的时间间隔+HealthCheckTimeout）。如果是SQL Server服务停止之类的问题，群集的最长响应时间就是Looksalive检查的时间间隔。

**5. SP\_SERVER\_DIAGNOSTICS**

前面已经多次提到了SQL Server 2012引入的新存储过程sp\_server\_diagnostics和它扮演的重要角色。这里我们具体看一下这个存储过程会提供怎样的信息。

Sp\_server\_diagnostics是一个标准的SQL Server存储过程。除了群集的会自动调用它来获得诊断信息之外，你也可以通过任意的SQL Server客户端工具（如SQL Server Management Studio）来直接使用它。

Sp\_server\_diagnostics只有一个参数 – “重复间隔”。如果不指定这个参数，则Sp\_server\_diagnostics返回一次诊断信息就结束了。一旦指定了这个参数，sp\_server\_diagnostics会根据这个间隔时间不断返回诊断信息。重复间隔时间不能小于5秒，因为sp\_server\_diagnostics需要约5秒钟才能返回结果。之所以需要5秒钟，是因为存储过程需要一段时间的连续运行来生成某些诊断信息。

sp\_server\_diagnostics是用来判断SQL Server健康状况、触发故障转移的重要存储过程，为了确保该存储过程的健壮性，保证它的运行不会受到服务器运行状况的干扰，SQL Server 2012数据库引擎为这个存储过程做了特别的优化：

1. 工作线程一旦开始运行该存储过程就会切换到抢占模式（preemptive mode）。
2. 工作线程的OS优先级会被提升，高于普通的SQL Server线程。
3. 该存储过程所需要的内存会被事先分配好。
4. 存储过程输出的内容被限制为每行256KB。

由于这个存储过程会被频繁地执行，因此其设计目标是不会给SQL带来超过0.1%的额外负载。SQL Server会维护一个全局列表用以记录所有正在运行sp\_server\_diagnostics语句的线程，另外安排Scheduler Monitor定期的去检查这些线程，一旦发现某个线程没能在30秒内完成一次sp\_server\_diagnostics的执行，Scheduler Monitor就会产生一个SQL Server进程的内存转储（dump）。分析这个dump能够帮助你理解为什么sp\_server\_diagnostics执行的这么慢。

下面让我们来看一个sp\_server\_diagnostics的示例输出（图2-19）：



图2-19 sp\_server\_diagnostics的输出结果

输出的结果分为5行，每一行代表一个“组件”。也可以认为是一种分类：

* System：从系统角度收集有关 spinlock、运行过程中产生的严重错误、发生non-yielding问题的任务、页面错误和 CPU 使用情况的数据。
* Resource：从资源的角度收集有关物理和虚拟内存、Buffer Pool、页面、缓存和其他内存对象的数据。
* query\_processing：从查询处理的角度收集有关工作线程、任务、等待类型、CPU 密集型会话和正在阻塞的任务的数据。
* io\_subsystem：收集有关 IO 的数据。 。
* Events：通过服务器记录的相关错误和事件中的存储过程收集数据，包括有关环形缓冲区异常、有关内存 broker 的环形缓冲区事件、内存不足、计划程序监视、缓冲区池、spinlock、安全和连接的详细信息。

输出结果最后一列是以XML形式记录下来的当前SQL Server的运行数据，包含了非常多的信息，对于判断SQL Server性能慢或宕机问题的根源非常有用。SQL Server利用内部的算法来处理这些数据，然后评估出一个运行状态。运行状态有4种：未知（unknown），干净（clean），警告（Warning）和错误（error）。

要注意的是，只有system， resource和query\_processing这三个组件会有“错误”运行状态。这三个组件的运行状态被用做故障转移的条件（从第3级到第5级的FailureConditionLevel）。

下表（2-2）显示了每个组件可以得到的运行状态：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 组件 | 干净 (1) | 警告 (2) | 错误 (3) | 未知 (0) |
| system | √ | √ | √ |  |
| resource | √ | √ | √ |  |
| query\_processing | √ | √ | √ |  |
| io\_subsystem | √ | √ |  |  |
| Events |  |  |  | √ |

表2-2 sp\_server\_diagnostics输出组件可能出于的运行状态

可以看到，io\_subsystem 的状态只可能是干净或警告，它不会得到错误状态；而Events永远只有未知这一种状态。

### 2.2.6 故障转移群集的故障排查

SQL Server 群集的故障排查主要是针对两类问题：

* SQL Server资源无法上线
* SQL Server意外发生故障切换。

这里提供一个排查这两类问题的大致思路，以供参考。

针对SQL Server资源无法上线的问题，首先要了解SQL Server资源上线的两大基本要素：1)SQL Server服务本身是否可以在该节点启动；2)Windows群集服务是否可以成功完成isalive检查，即可以成功连接SQL Server，并且运行sp\_server\_diagnostics语句。

所以，要排查SQL Server资源无法上线的问题，您可以根据这样的顺序来检查：

1. 在故障转移群集管理器中，确保SQL Server资源所依赖的资源都可以上线。如果任何资源不能上线，则去Windows事件日志中去查看是否有对应的错误和警告信息，然后做进一步的排查。
2. 如果所有依赖资源都可以上线，那么到当前节点的本地磁盘中找到SQL Server实例的Binn目录下的sqlservr.exe文件。然后用命令行启动SQL Server。具体的步骤可以参见SQL Server的联机丛书：

<http://msdn.microsoft.com/zh-cn/library/ms180965.aspx>

如果SQL Server启动失败，就可以到共享磁盘的LOG目录里找到SQL Server的错误日志ERRORLOG，看看ERRORLOG里有没有可以帮助排查问题的信息。如果不幸的ERRORLOG没有产生出来，那么就再到Windows事件日志中去查看是否有错误或者警告信息。

1. 如果SQL Server可以通过命令行启动起来，就说明问题出在Windows群集连接SQL Server并执行sp\_server\_diagnostics的过程中。这个时候可以先使用Sqlcmd等工具来连接通过命令行模式启动起来的SQL Server（记得要使用Windows验证）。如果出现连接错误或者认证错误，那么问题就转换成了一个纯粹的连接认证问题。
2. 如果sqlcmd可以成功连接，那么问题很可能与Kerberos/NTLM认证方式有关系。由于Windows Server 2008群集服务的启动账号是Local System，因此当Windows群集通过TCP/IP或者Named Pipes协议去做Isalive检查时，实际上是使用了一个机器账号去连接SQL Server。由于连接SQL Server所使用的用户不同，有可能出现用sqlcmd做连接测试时没问题，但是Windows群集连接出问题的情况。这种情况下问题一般都是Kerberos或者NTLM认证的问题，导致最后isalive检查失败。在后面的第七章我们会更具体的介绍Kerberos和NTLM。

如果SQL Server资源意外的发生了故障转移，则需要去分析故障转移的根本原因。此时，您至少需要收集以下信息：

1. Windows群集的日志。

这个是最重要的日志，通过它可以发现：

1. SQL Server资源所依赖的资源是否发生了问题并进入了失败状态。如果是的话，那么SQL Server资源的故障转移完全是由于它所依赖的资源造成的，和SQL Server本身无关。
2. SQL Server资源组里是否有任何被配置“affect the group”的资源发生了问题并触发了整个资源组的故障转移。如果是这样的话，那么SQL Server就完全是一个受害者。
3. 是否有任何信息表明Windows群集无法完成对SQL Server的isalive检查。根据isalive检查失败的原因不同，信息的内容也会有所不同，这有助于你定位问题。总的而言，isalive失败的原因是两大类（1）Windows群集找不到SQL Server实例，（2）SQL Server出现性能问题无法响应Windows群集的连接请求或者无法成功执行sp\_server\_diagnostics命令。无论是何种原因，你一般都能看到类似下面的错误信息：

00000a04.00000e3c::2011/08/03-14:23:38.719 ERR SQL Server <SQL Server>: [sqsrvres] CheckQueryProcessorAlive: sqlexecdirect failed

00000a04.00000e3c::2011/08/03-14:23:38.719 ERR SQL Server <SQL Server>: [sqsrvres] printODBCError: sqlstate = 08S01; native error = 40; message = [Microsoft][SQL Native Client]Communication link failure

00000a04.00000e3c::2011/08/03-14:23:38.719 ERR SQL Server <SQL Server>: [sqsrvres] OnlineThread: QP is not online.

通过在Windows群集的日志中搜索这些关键字，可以帮助你迅速地找到isalive检查失败的时间点。

Windows Server 2008群集的日志需要通过运行以下命令行来产生：

***cluster log /g***

要注意Windows群集日志的大小是有限制的，它会不断地冲刷掉最老的信息。因此一旦发生故障转移，应当尽快地收集Windows群集日志.

1. SQL Server的错误日志。

SQL Server的错误日志有助于判断为什么isalive会失败。打开SQL Server资源发生故障转移的时间点之前的那个ERRORLOG，到该文件的最后查找有没有如下这条信息：

SQL Server is terminating in response to a 'stop' request from Service Control Manager. This is an informational message only. No user action is required.

如果没有这条信息，那说明SQL Server是被不正常关闭的，有可能是整个进程崩溃或者是被其他进程kill掉的。

如果看到这条信息，那说明SQL Server是因为接收到来自Windows群集的命令才停止的。一般来说，这是因为isalive检查失败导致的。请再去查看这条信息之前，在发生故障转移的时间点附近，SQL Server的ERRORLOG里有没有记录任何与性能问题、连接或者认证失败有关系的信息。这些都会帮助您进一步追踪SQL Server群集发生故障转移的原因。

1. Windows事务日志。

Windows事务日志主要是用来检查节点是否存在I/O系统或者网络层的异常。I/O系统和网络层的异常都可能导致SQL Server群集的意外故障转移。

1. “SQLDIAG”的扩展事件（extended events）日志

在以前版本的SQL Server中，如果resource DLL产生了任何的错误或者警告信息，这些信息会被记录到Windows的事件日志中去。在SQL Server 2012中，这些信息除了会被记录到Windows事件日志，还会被写入到名为“SQLDIAG”的扩展事件（extended events）日志文件中去，而且内容更为丰富。

每隔(HealthCheckTimeout/3)的时间（最小5秒），sp\_server\_diagnostics会获得SQL Server的诊断信息。这些信息也会被记录在SQLDIAG扩展事件日志文件里。由于默认设置HealthCheckTimeout为60秒，所以理论上SQLDIAG中每20秒就会有sp\_server\_diagnostics的信息被记录下来。但在实际的运行中，如果resource DLL发现上次sp\_server\_diagnostics的评估的结果都是“干净”，而本次的评估结果也是“干净”，那么本次的sp\_server\_diagnostics诊断信息就会被跳过而不被记录到SQLDIAG日志中。在记录下一次新的“干净”记录之前，最多可以跳过大约10次。如果有你爱SQLDIAG日志中发现有sp\_server\_diagnostics信息被跳过你没有记录下来，这意味着这段时间内评估结果一直没有改变，一直都是“干净”。如果有任何评估结果的变化，那么SQLDIAG日志中就一定有记录。

如果你把SQL Server资源的Verbose Logging属性设置为1或2的话，resource DLL会记录额外的信息到Cluster log中，而这些信息也会同时记录到SQLDIAG日志中；另外当前的FailureConditionLevel设置也会伴随着每次sp\_server\_diagnostics返回的诊断信息被一并记录到SQLDIAG日志中。一旦SQL Server群集发生了意外的故障转移，你可以通过查看SQLDIAG日志文件中的内容来判断当时SQL Server发生了什么问题。

SQLDIAG扩展事件日志文件和SQL Server的错误日志一样都存放在实例的LOG文件夹中。文件名是：

**<节点名>\_<实例名>\_SQLDIAG \_<序号>\_<创建时间>.xel**

默认设置下，每个SQLDIAG日志文件的大小是100MB。一旦100MB被写满，或者SQL Server重启，就会产生新的文件。SQL Server最多保留10个文件，一旦10个文件写满了，最早的文件就会回收掉，然后SQL Server就可以继续创建新的文件写入信息。

你可以通过运行ALTER SERVER CONFIGURATION SET DIAGNOSTICS LOG语句来控制是否将对 sp\_server\_diagnostics 捕获的诊断数据写入SQLDIAG日志，或者设置SQLDIAG日志的文件大小，存放位置和保留文件数量。

你可能会注意到，SQL Server 2012的LOG目录下新增了3类扩展事件日志文件，除了SQLDIAG文件外，还有：

1. AlwaysOn扩展事件日志文件。
2. System Health扩展事件日志文件。

我们会在第三章中再提到这两个文件。

## 2.3 日志传送

SQL Server有多种技术可以用于实现自动创建冗余数据拷贝，提供数据灾难恢复的功能。其中，日志传送（log shipping）是最早出现的技术之一。虽然日志传送“年纪”很大，但是它依旧凭借着其独有的特点，在SQL Server的灾难恢复技术中扮演着重要的角色。

### 2.3.1 日志传送的结构

日志传送的工作机制非常简单。它主要是通过最基本的数据库事务日志备份和还原任务，来保持两台或者多台机器之间数据库同步，以此来实现数据冗余、灾难恢复的目的。

这里先来看看在日志传送中各个服务器所扮演的角色。

**主服务器和数据库**

日志传送配置的“主服务器”（primary server），是作为生产服务器的 SQL Server实例。主数据库是主服务器上希望备份到其他服务器的数据库。所有用户端发来的数据操作请求，都是由主数据库处理的。

主数据库必须使用完整恢复模式或大容量日志恢复模式，简单恢复模式由于无法做日志备份，所以不被日志传送所支持。在一个日志传送的环境中，如果您将数据库的恢复模式改成简单恢复模式，日志传送会立刻停止工作。

[**[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))辅助服务器和数据库**](javascript:void(0))

辅助服务器（secondary server）是您想要在其中保留主数据库备用副本的服务器。在辅助服务器上的辅助数据库，可以配置为以下两种状态：

1. 无恢复模式(NORECOVERY):既不前滚也不回滚未提交的事务，数据不可读。
2. 备用模式(STANDBY):在恢复日志期间回滚所有未提交的事务，并且将所有未提交的事务保存为一个单独的Transaction Undo File(TUF)文件，恢复过程通过该文件来维护事务的完整性，当恢复下一个事务的时候则恢复所有已提交的事务。这种状态下，数据库是可读的，此时该辅助数据库可被用于生成报表之类的只读的需求。

在日志传送的配置中，一台主服务器可以有多台对应的辅助数据库。在每个辅助数据库上都有主服务器上的主数据库的备份。比如，可以有3台辅助服务器，第1和第2台辅助服务器上的辅助数据库设置成NORECOVERY模式，提供冗余的灾备功能。即使某一台辅助服务器出现了问题，另一台辅助服务器依旧可以提供灾难恢复功能。第3台辅助服务器上的辅助数据库则可以设置为STANDBY模式，这样，可以把所有OLTP类型的请求都交给主服务器处理；把所有数据仓库类型的请求交给辅助数据库。这样一定程度上减轻了主数据库的负载，对数据库系统的性能提高也有好处。您可以通过查询系统表log\_shipping\_primary\_secondaries来知道某个主数据库有哪些对应的辅助数据库。

另外，一台辅助服务器也可以包含存储在多台不同主服务器中数据库的备份副本。比如说，你可能有五台服务器，每台服务器都运行关键数据库系统。在这种情况下，可以只使用一台辅助服务器，而不必使用五台单独的辅助服务器。五个主系统上的备份都可以加载到这个备份系统中，从而减少所需的服务器数量并节省开支。

[**[http://i.msdn.microsoft.com/Hash/030c41d9079671d09a62d8e2c1db6973.gif](javascript:void(0))监视服务器**](javascript:void(0))

监视服务器是可选的，它可以跟踪日志传送的所有细节，包括：

* 主数据库中事务日志最近一次备份的时间。
* 辅助服务器最近一次复制和还原备份文件的时间。
* 有关任何备份失败警报的信息。

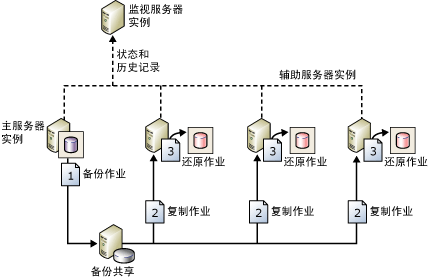
监视服务器应独立于主服务器和辅助服务器，以避免由于主服务器或辅助服务器的丢失而丢失关键信息和中断监视。一台监视服务器可以监视多个日志传送配置。在这种情况下，使用该监视服务器的所有日志传送配置将共享一个警报作业。

### 2.3.2 日志传送的工作机制

日志传送的工作机制非常简单。日志传送组件在主服务器上对数据库做日志备份，然后把日志备份文件复制到所有的辅助服务器上，最后在辅助服务器上将日志备份还原上去。通过不断重复这样的过程，就实现了主服务器和辅助服务器上数据库的同步。

所有这些备份/复制/还原操作当然不可能是人为触发。日志传送会在主服务器，辅助数据库，以及监视数据库上创建相应的作业来分别完成这些工作。每个服务器上的SQL Server Agent服务会每隔一定时间间隔来启动这些作业。

日志传送一共有4个作业，分别是备份作业、复制作业、还原作业和警报作业。这些作业是如何协作运行，完成数据传递的目标呢？下面这个图（2-20）展现了一个拥有一台主服务器，三台辅助服务器和一台监视服务器的日志传送配置，以及各个作业的执行顺序。



**图2-20** 日志传送的4个作业

现在来分别讨论每个作业。

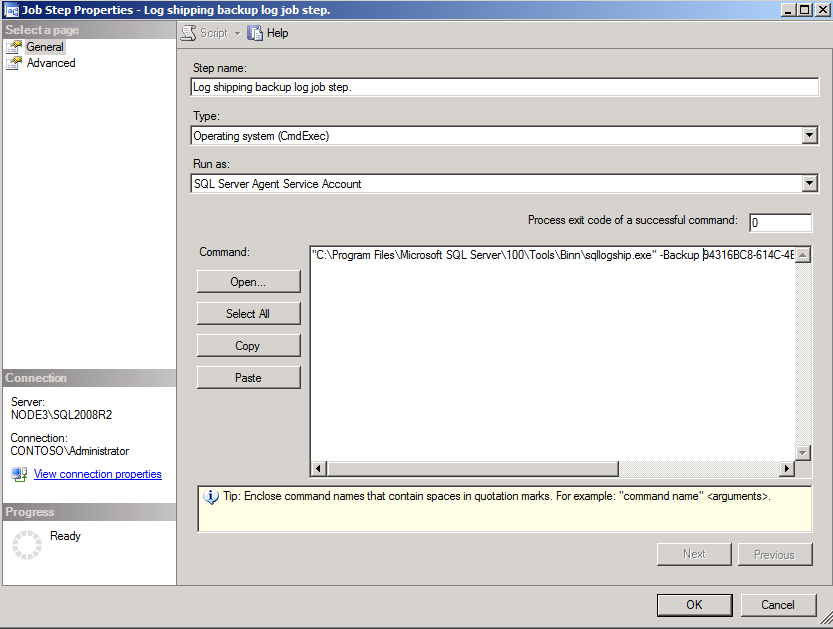
**备份作业**

日志传送在主服务器上为每个主数据库创建一个备份作业。它负责对数据库做日志备份，将备份操作的历史信息记录到本地服务器和监视服务器上，并根据作业的设置来删除旧的日志备份文件和历史记录信息。默认情况下，SQL Server Agent服务每 15 分钟执行一次此作业，当然间隔也是可以修改的。备份作业会把日志备份文件放到一个指定的共享目录上去，等待复制作业的操作。

从SQL Server 2008开始，SQL Server的企业版支持备份压缩的功能。用户在创建日志传送的时候，可以选择是否为备份作业启用备份压缩。这对于一些非常繁忙的系统会很有用，它可以减少服务器上由于备份造成的I/O操作，也可以减低复制日志备份文件所需要的时间。

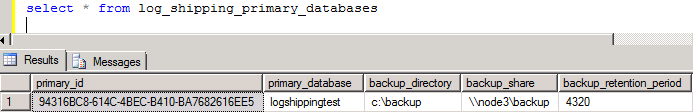
SQL Server 2000日志传送的备份作业就是直接通过SQL Server Agent来执行Backup log命令。SQL Server 2005开始，日志传送开始使用一个叫做sqllogship.exe的可执行文件来执行备份/复制/还原的操作。您在SQL Server Agent中会找到一个名字为LSBackup\_<database>的作业（图2-21）。此处<database>为启用日志传送的主数据库名字。该作业只有一个步骤，执行的操作是:

sqllogship.exe -Backup **<GUID>** -server **<instance name>**



**图2-21 备份作业**

命令中的<GUID>对应的就是主数据库的唯一标识号。如果去查询主服务器的MSDB数据库中的log\_shipping\_primary\_databases表的话你就会找到对应的primary\_id （图2-22）。



**图2-22 查询主数据的GUID**

<instance name>对应的就是主服务器的实例名。Sqllogship.exe通过<instance name>连接到主服务器，然后通过<GUID>找到的主数据库，最后运行backup log命令来备份日志文件。

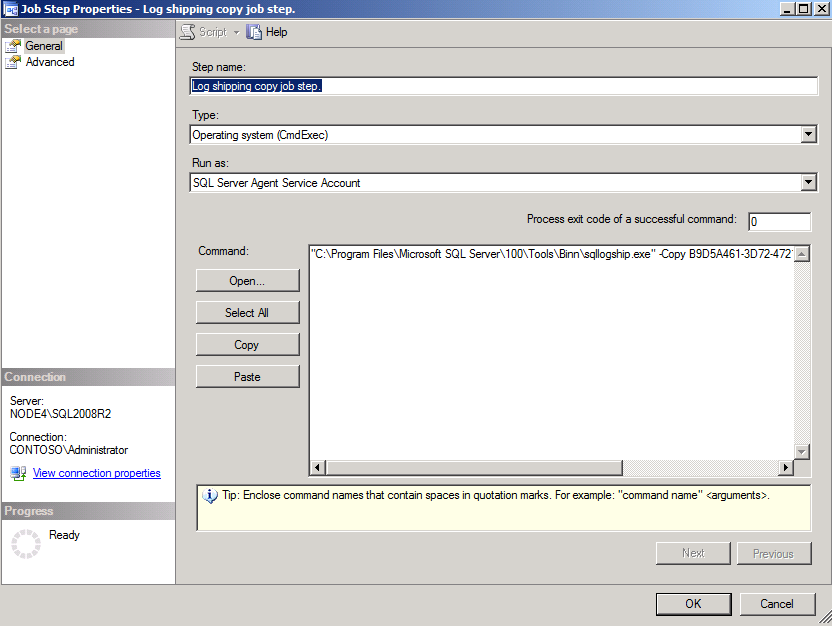
**复制作业**

日志传送在每个辅助服务器上会创建复制作业。它的作用就是将日志备份文件从主服务器的共享目录中复制到辅助服务器上的一个本地目录，并在辅助服务器和监视服务器中记录历史记录。与备份计划一样，该作业的默认执行间隔是15分钟一次，当然也可以修改成您做想要的间隔。

该作业的名字是LSCopy\_<Instance Name>\_<database>。这里的<instance name>是主服务器的实例名，而<database>是辅助数据库名。

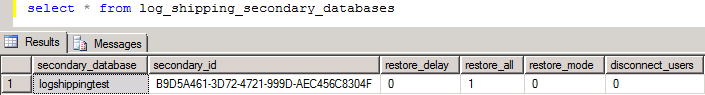
和备份作业一样，从SQL Server 2005开始日志传送使用sqllogship.exe来完成复制的操作（图2-23）。你会发现备份作业只有一个步骤，执行命令是

sqllogship.exe -Copy **<GUID>** -server **<instance name>**



**图2-23 复制作业**

命令中的<GUID>是辅助数据库的唯一标识号。如果去查询辅助数据库实例上的MSDB数据库中的log\_shipping\_secondary\_databases或者log\_shipping\_secondary表的话，可以找到对应的secondary\_id。此外，命令中的<instance name>是辅助数据库的实例名（图2-24）。



**图2-24 查询辅助数据库的GUID**

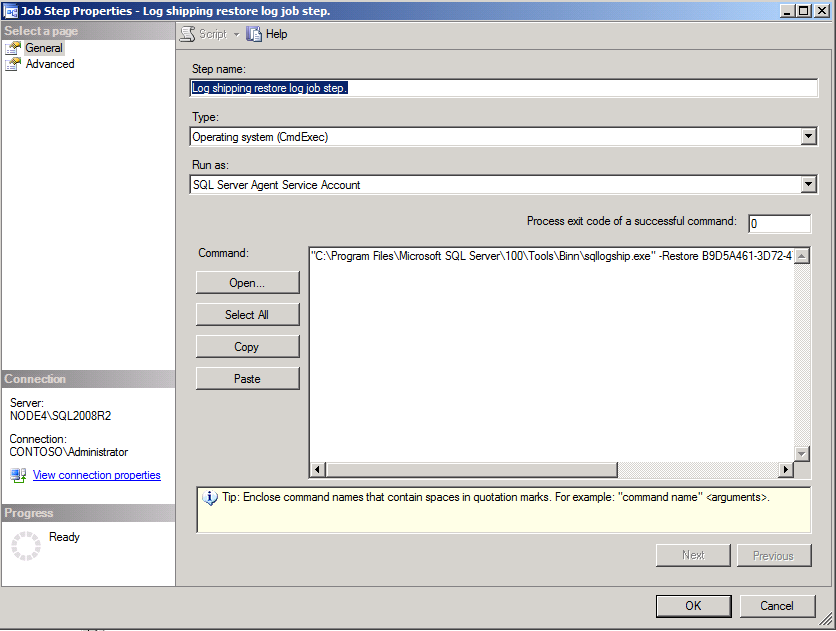
**还原作业**

日志传送在辅助服务器上为每个辅助数据库创建一个还原作业。此作业将复制到本机目录的日志备份文件，还原到辅助数据库。同时还原作业会将历史记录信息记录在本地服务器和监视服务器上，并删除旧文件和旧历史记录信息。还原的时候，根据日志传送配置的不同，还原作业会把数据库还原成NORECOVERY 模式或者是STANDBY 模式。该作业的默认执行间隔是15分钟。

还原作业的名字是 LSRestore\_<Instance Name>\_<database>。这里的<instance name>是主服务器的实例名，而<database>是辅助数据库名。

SQL Server 2005开始的日志传送使用sqllogship.exe来完成还原操作（图2-25）。还原作业只有一个步骤，执行的命令是：

sqllogship.exe -Restore **<GUID>** -server **<instance name>**



**图2-25 还原作业**

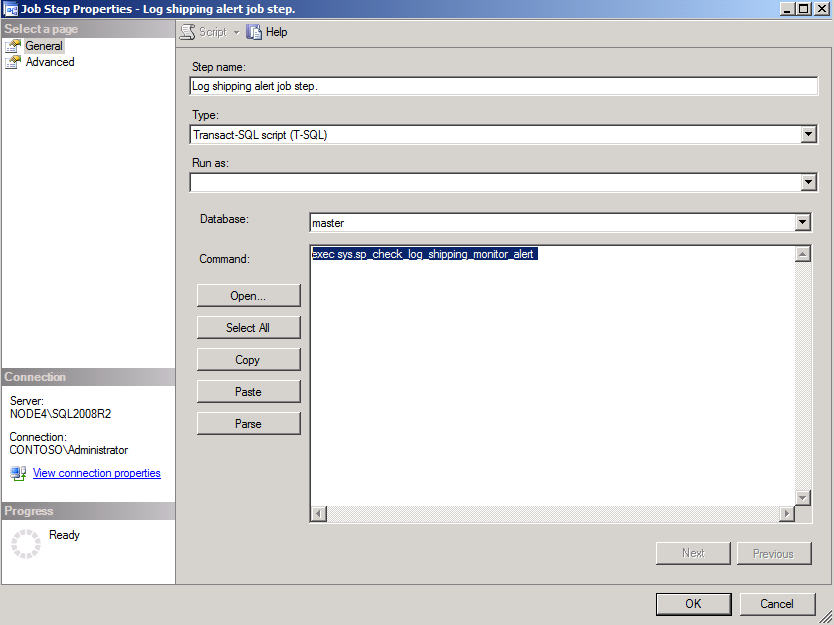
命令中的<GUID>和<instance name>参数与复制作业做执行的sqllogship.exe的参数是一样的。Sqllogship.exe通过<instance name>连接辅助数据库实例，然后根据<GUID>找到对应的辅助数据库执行还原操作。

**警报作业**

如果使用了监视服务器，日志传送将在监视服务器上创建一个警报作业。如果有多个数据库配置了日志传送并且它们都是用了这个监视服务器，那么所有这些日志传送配置都共享这一个警报作业。如果主数据库无法在指定时间范围内完成备份操作，或者辅助数据库无法再指定时间范围内完成还原操作，警报作业就会触发主数据库或辅助数据库的警报（Alert）。

如果没有使用监视服务器，日志传送将在主服务器和每个辅助服务器上创建一个警报作业。如果主数据库无法在指定时间范围内完成备份操作，主服务器实例上的警报作业会生成一个错误。如果或者辅助数据库无法再指定时间范围内完成还原操作，辅助服务器实例上的警报作业会调用raiseerror来引发一个错误。

警报作业的名字是LSAlert\_<Instance Name>。<Instance Name>就是该作业所在的SQL Server实例名。该作业其实是在定期的执行一个系统存储过程sys.sp\_check\_log\_shipping\_alert，执行间隔默认是2分钟 （图2-26）。



**图2-26 警报作业**

有兴趣的话，您可以用sp\_helptext来看看这个存储过程的定义。

### 2.3.3 日志传送作业的执行间隔

日志传送是利用SQL Server Agent来执行备份还原的作业，从而达到主副数据库同步的目的。由于作业是每隔一定的时间间隔才被SQL Server Agent触发的，因此主副数据库的同步也不是实时的。它们之间会有一定程度的延迟。这是日志传送技术的一个重要特征。如果发生灾难，这部分延迟可能会带来数据丢失。

日志传送的主副数据库同步的延迟到底会有多长呢？这是由备份作业、复制作业和还原作业的运行间隔决定的。不过，用户所关心的最大数据损失量仅是由备份作业运行的间隔所决定的。例如，上述3个作业的执行间隔都被设置为15分钟，那么主副数据库之前同步的延迟会在15到45分钟之间。但是，只要日志被成功备份下来，无论之后复制和还原操作会延迟多长时间，日志备份始终在那里，最终都会被还原到辅助数据库上去。最大的数据丢失，是从最后一次成功的日志备份开始到数据库发生异常这段时间范围内的所有数据变化。因此，对于备份间隔是15分钟的日志传送配置，最坏情况下就是损失15分钟内的数据。

你可以根据你所能接受的最大数据损失量来指定备份作业的执行间隔。对于非常繁忙的OLTP系统，建议可以适当地缩短备份间隔。这样除了可以减小数据库发生异常后的数据的损失量，还可以有效地控制主数据库的日志文件大小，防止由于没有及时截断日志而导致日志文件太大的问题。

但是，无论怎么减小备份作业执行的间隔，日志传送是永远无法保证两个数据库是完全同步的。SQL Server允许的作业执行最小间隔是10秒钟，因此理论上您至少会有10秒的数据损失。此外，将备份间隔设置的太小也会给主服务器带来额外的负担，一定程度上影响数据库的性能。如果你追求的是零数据损失的灾备方案，日志传送并不适合你。

在给定的辅助服务器上，用户可以让复制作业和还原作业使用和主服务器的备份作业一样的执行间隔，并且控制复制作业和还原作业开始执行的时间，让复制作业尽可能紧随着备份作业完成后执行，让还原作业紧随着复制作业完成后执行。这样，每个日志备份被创建后可以立即将其复制和还原。这有助于减少在主服务器出现故障之后使辅助服务器上线所需的修复时间。

相反，用户也可以有意地延迟复制作业和还原作业，或者把复制作业和还原作业的执行间隔设置为远大于备份作业的执行间隔。这样做可以延迟将事务日志备份恢复到辅助数据库。该延迟提供了一个时间间隔。在这个时间间隔内，用户可以响应主服务器 上发生的异常或故障。这对于处理严重的用户错误是很有用的。比如说，如果用户错误地删除了某张表或表中的某些行，并且管理员知道误操作的时间，他就可以立刻暂停复制和还原作业，不让误操作被同步到辅助数据库上。然后，手动地将包含错误操作的日志备份复制到辅助服务器上，并使用restore database命令中的stop at参数将数据库在辅助服务器上还原到误操发生前的状态。然后，管理员可以选择切换数据库的角色，让辅助数据库上线运行；或者选择导出主数据库上已经丢失的数据，然后将其导回主数据库。

总之，可以灵活地设置各个作业的执行间隔，来满足不同的业务需求。

### 2.3.4 日志传送的故障转移

当主数据库发生异常后，您需要将应用切换到辅助数据库上。本节将介绍日志传送的故障转移过程。比较群集的故障转移，日志传送的故障转移无疑是要更麻烦些，需要很多的手工操作。所以如果仅使用日志传送，是很难达到很好的高可用性的。

日志传送故障转移的基本流程如下：

1. 备份主数据库的尾日志（如果可能的话）

通常，发生故障的时候，主数据库与辅助数据库是不同步的，因为主数据库在其最近的备份作业后会继续有更新。此外，在某些情况下，主服务器上最新的日志备份尚未复制到辅助服务器中，或者某些已复制的日志备份可能尚未应用到辅助数据库中。因此首先要将辅助数据库与主数据库同步。

此时，如果此时主数据库还是可以访问的话，你需要手工对主数据库做一个日志备份，也就是备份“尾日志”，这样就获得了最后一次备份作业执行之后所有数据更新。建议使用with norecovery参数来备份尾日志，这样使得主数据库处于还原状态，保证在做故障转移的时候不会有新的数据更新进入主数据库，可以确保主数据和辅助数据库的同步。

接下来，把主服务器的共享目录中所有未被复制的备份文件，连同尾日志备份，复制到所有辅助服务器上日志传送所用的本地目录中。

1. 在辅助服务器上，通过检查还原作业的历史信息，可以知道最后一个被还原的日志备份是哪个。然后，要将所有未还原的事务日志备份按顺序还原到所有辅助数据库上。这样，数据同步的工作就完成了。
2. 同步辅助服务器之后，你可以选择恢复辅助数据库来作为故障转移的目标数据库。对该辅助数据库执行恢复操作（restore database… with recovery），恢复操作将使数据库处于一致的状态并使其联机。然后删除辅助服务器上的复制作业和还原作业，故障转移就完成了。
3. 日志传送只同步用户数据库的内容，但不会同步保存在master数据库中的登录信息。要使得应用可以使用新的主数据库，你需要手动把登录信息从原始的主服务器实例迁移到新的主服务器实例上。在新的主服务器上建立登录，除了要保证登录名和密码要和原始的主服务器一致之外，还要保证登录名对应的SID也一致；否则该登录名就无法被正确映射成为新的主数据库中的用户，导致应用依旧无法使用数据库。

要正确的迁移登录信息，建议参照以下文章提供的方法。

How to transfer the logins and the passwords between instances of SQL Server 2005 and SQL Server 2008 (<http://support.microsoft.com/kb/918992>)

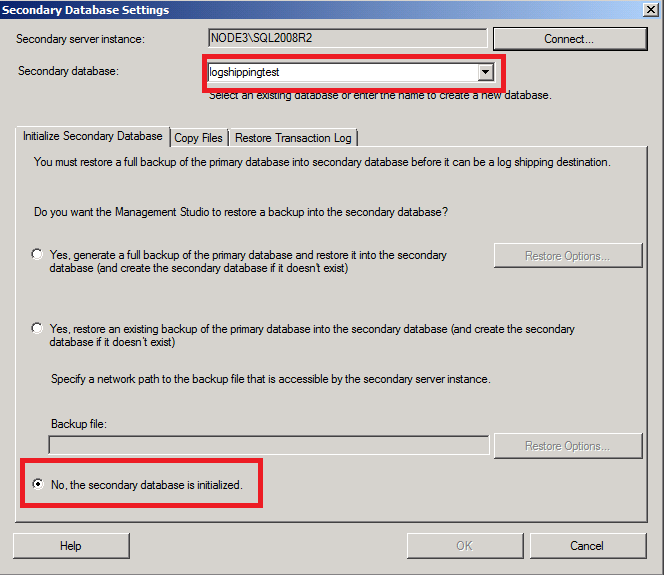
1. 最后你需要通过修改连接字符串，使用别名等方法把应用重定向到新的主服务器上。至此一个日志传送的“故障切换”就大功告成了。

可以看到日志传送的故障转移操作起来是比较麻烦的，需要一定的时间来完成。操作人员需要很熟悉整个过程。因此日志传送不是一个好的高可用性方案，它更多的作为一个灾难恢复的方案，为主数据库提供一个或多个副本来作为保障。

恢复辅助数据库之后，它就成为了新的主数据库。当故障服务器（原先的主服务器）修复后，你可以重新配置日志传送，让原先的辅助数据库开始扮演主数据库的角色，而原先故障的主数据库转为扮演辅助数据库。（日志传送的故障切换是单向的，不提供自动地将主数据库转换角色的功能。）

1. 如果原先的主数据库已经完全不可用，就必须完全重建日志传送，在重建时把原先的主服务器选择为新的辅助服务器。
2. 如果原先的主数据库还可以访问，只需执行以下步骤就可以比较快速地交换主数据库和辅助数据库的角色。
3. 确保已经使用了with norecovery备份过了原始主数据库上的尾日志（见故障转移流程的步骤1）。
4. 删除原先的主服务器上的日志传送备份作业以及原先的辅助服务器（现在的主服务器）上的复制和还原作业。
5. 使用 SQL Server Management Studio 在原始的辅助数据库（要用作新的主数据库的数据库）上配置日志传送。要注意以下步骤：

* 添加辅助数据库时，在“Secondary Database Settings”对话框的“Secondary database”框中输入原先的服务器上的主数据库的名称。见图2-27中的logshippingtest。
* 在“Initialize Secondary Database”对话框中，选中“No, the secondary database is initailizaed”。（图2-27）

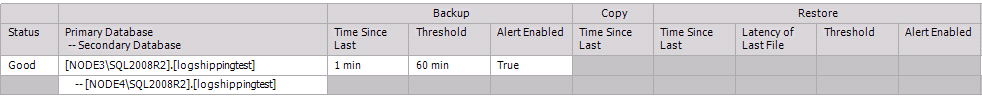


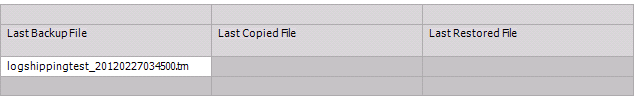
**图2-27 Secondary Database Settings对话框**

无论是原始的主数据库是否还可用，在交换角色的过程中，都建议使用与切换之前相同的共享文件夹来存放新的主数据库的日志备份。这样做的好处是，如果原始的日志传送配置中有多个辅助服务器的话，其余那些没有参与角色交换的辅助服务器，依旧可以通过原本配置的复制作业和还原作业，继续和新的主数据库同步。

### 2.3.5 日志传送的监控和故障排查

如何监控日志传送的运行状态？最直观的办法是使用SQL Server Management Studio中的报表。用户可以在SQL Server Management Studio中右键实例名，选择“报表”，然后选择“标准报表”，最后选择“事务日志传送状态”。

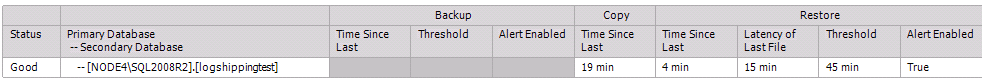


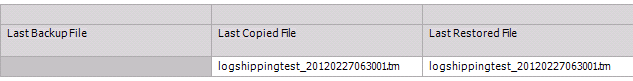


**图2-28 主服务器上的事务日志传送状态报表**

图2-28是从主服务器端打开的报表。最前面的”status”栏显示当前日志传送的运行状况，除此之外还能看到备份作业的配置，以及最后一次被成功备份的日志文件是哪个。但是复制作业和还原作业由于是运行在辅助服务器端的，它们的信息只能在辅助服务器上看到。

如果在辅助服务器端打开这个报表的话，就可以看到辅助服务器上日志传送的状态（图2-29）。





**图2-29 辅助服务器上的事务日志传送状态报表**

在辅助服务器端，除了日志传送的状态外，还能看到最后一个被复制的日志备份文件和最后一个被还原的日志备份文件。

除了使用报表之外，直接在master数据库中执行存储过程sp\_help\_log\_shipping\_monitor也能获得同样的信息。如果查看这个存储过程的定义，可以发现它实际上查询的是以下两张表。

* msdb.dbo.log\_shipping\_monitor\_primary
* msdb.dbo.log\_shipping\_monitor\_secondary

如果发现日志传送发生任何异常，可以充分利用这些报表或者存储过程所提供的信息来做进一步的判断。

首先，要明确是什么异常导致了日志传送不能工作。由于日志传送完全是依靠几个SQL Agent的作业来执行工作的，所以第一步应该去查看那些作业的历史信息，从历史信息中可以大致了解遇到的是什么问题。

查看作业历史信息的简单办法，当然是在SQL Server Management Studio里找到作业，并且直接选择查看该作业的历史信息。用命令的方式查询作业历史信息，可以选择的方法有：

1. 在主服务器和辅助服务器上执行查询select \* from [msdb].[dbo].[sysjobs] where category\_id = 6。获得所有日志传送作业的作业ID。然后查询msdb..sysjobhistory中对应的作业历史信息，查看具体的错误。这个方法比较传统，即使对日志传送的系统表不熟悉也可以使用。
2. 在主服务器，辅助服务器和监视服务器（如果有的话）上执行查询SELECT \* FROM [msdb].[dbo].[log\_shipping\_monitor\_history\_detail]。这个表会包含日志传送相关作业的历史信息。其中也自然包含出现错误作业执行以及详细的错误信息。
3. 在主服务器，辅助服务器和监视服务器（如果有的话）执行查询SELECT \* FROM [msdb].[dbo].[log\_shipping\_monitor\_error\_detail]。这个表包含所有出现错误的日志传送作业的历史信息。

用这些方法，可能可以得到更准确的信息。有了错误信息，接下来该怎么做就会有思路了。

这里先来明确两个问题。理解这两个问题，会有助于您找到解决日志传送问题的关键。

1) 复制作业是如何知道当前应该复制哪个日志备份文件的？

由于备份作业只会定期清理一段时间以前的日志备份，因此在共享目录中，一定会存在一些已经被复制到辅助服务器端的日志备份。总不见得每次都把目录中所有的文件都复制一遍吧？这个时候，复制作业其实是根据日志备份的文件名来判断应该复制哪些文件的。你会发现备份作业生成的日志备份的文件名格式是“主服务器名\_<时间戳>”。这里的时间戳其实就是备份文件生成的时间。另外，在辅助服务器的MSDB数据库中会记录最近一次被复制的日志备份文件是什么。查询log\_shipping\_secondary表，就能够得到被复制的文件名，以及是什么时间做的复制。日志传送会检查共享目录中所有文件的文件名，比较辅助服务器上的记录。只有文件名中的时间戳大于最近一次被复制的文件的时间戳，文件才会被复制作业复制到辅助服务器上。

2) 还原作业如何决定应该还原哪个文件？

和第一个问题类似，它也是简单地使用了时间戳的比较。在MSDB中会记录最近一次被还原的备份文件名。查询log\_shipping\_secondary\_databases这张系统表，就可以看到最近一次日志备份文件执行还原操作的时间。

通常来说，日志传送发生的问题大部分出现在日志传送执行复制作业和还原作业的时候。如果备份作业出现问题，通常只需要把它当做一个单纯的备份失败问题进行处理就行了。

复制和还原这两步，出问题相对比较麻烦。和复制作业相关的问题，通常会有以下三类问题：

1. 权限问题。复制作业没有权限访问存放日志备份的共享目录；或者复制作业无法写入辅助服务器上的本地目录。这个时候要检查辅助服务器上的SQL Server Agent启动账号是否具有访问相应目录的权限。
2. 共享目录不存在。有可能是该目录被移走了，或者没有被共享。也可能是由于网络原因导致访问失败（比如DNS解析问题等）。
3. 日志备份文件名被人为修改了，时间戳小于了最近一次被复制的文件的时间戳，导致备份文件无法被复制作业复制到辅助服务器端。

和还原作业相关的问题会相对更复杂一些。常见的会有以下四类：

1. 还原作业无法访问日志备份文件，因为该文件正在被其他进程使用。

这时您可以使用Process Explorer工具（可以在微软的网站免费下载）来查看是什么进程正在使用日志备份文件。如果Process Explorer无法找到使用该文件的进程，而且这个问题只是偶尔发生，有可能是服务器上有杀毒软件或者监控软件。当这些软件正在扫描到日志备份文件的时候，这个文件就不能被还原作业使用了。

1. 还原作业没有权限读取本地目录中的日志备份文件。这种情况你只需要去检查辅助服务器上的SQL Server Agent启动账号对本地目录和其中文件的访问权限即可。
2. 日志备份文件损坏无法还原。您可以尝试手动把该日志 备份文件从主服务器再次复制到辅助服务器上，然后运行一个restore命令来还原这个日志备份。如果依然还是得到同样的错误，那说明那个日志备份文件本身已经发生了损坏。这时别无选择，只能重建日志传送。
3. 日志备份文件的LSN不在允许还原的范围内。在还原时，一个有效的日志备份文件，它的起始LSN一定不大于辅助数据库的最末LSN；日志备份文件的最末LSN一定大于辅助数据库的最末LSN。您可以使用下面的办法来获取LSN的信息。

（1） 在主数据库上运行语句：

SELECT DISTINCT s.first\_lsn,s.last\_lsn,s.backup\_finish\_date,y.physical\_device\_name FROM msdb..backupset AS s INNER JOIN

msdb..backupfile AS f ON f.backup\_set\_id = s.backup\_set\_id INNER JOIN

msdb..backupmediaset AS m ON s.media\_set\_id = m.media\_set\_id INNER JOIN

msdb..backupmediafamily AS y ON m.media\_set\_id = y.media\_set\_id

WHERE (s.database\_name = 'xxxx') ORDER BY s.backup\_finish\_date DESC;

这样可以得到每个日志备份文件的起始LSN和最末LSN。

（2） 在辅助服务器上运行restore headeronly来检查日志备份文件的起始LSN和最末LSN。

（3） 运行DBCC DBTABLE ('数据库名‘)命令来获得数据库的最末LSN。

如果发现备份文件的起始LSN大于数据库的最末LSN，就是个典型的日志不连续的状态。此时可以使用步骤(1)中的查询来找到正确的备份文件，该文件具有和辅助数据库连续的LSN。找到有这样一个文件后，接下来就要研究为什么还原作业没有去还原这个日志备份文件，而是去直接还原更晚产生的日志备份文件？最可能的情况，是那个具有正确LSN的备份文件不是由日志传送生成，而是由某人手动执行备份命令生成。这样手动生成的备份文件，要么是文件名没有时间戳信息，要么是备份文件没有被放在到日志传送的共享目录。无论什么原因都会导致复制作业忽视了这个文件，没有把它复制到辅助服务器上。

对于启用日志传送的数据库，任何手动的日志备份都会破坏日志传送的工作。如果找不到手动生成的备份文件，那日志传递链条就此中断，就只能重建整个日志传送。这点是管理数据库的时候要特别注意的。此外，对已经启用了日志传送任务的数据库，配置维护计划要特别注意，不能在维护计划中创建事务日志备份。你可以在维护计划中执行完整数据库备份和差异数据库备份，这些是不会与日志传送产生冲突的。

## 2.4 数据库镜像

根据前面的介绍读者会发现，故障转移群集技术和日志传送技术都不是那么“完美”。故障转移群集可以让系统在遇到故障时自动恢复，但是不能提供数据冗余，以对抗数据库损坏事故。日志传送可以提供冗余的数据拷贝，但是主数据库和辅助数据库之间的时间差可能比较长，故障切换也比较麻烦。所以仅有这两种技术，还是不能满足SQL Server用户的需求。

数据库镜像功能首次出现在SQL Server 2005 SP1中。它设计的目的是试图为SQL Server提供一个具有实时性数据同步的灾难恢复技术，即能提供数据冗余备份，切换起来也比较方便。相比较故障转移群集，数据库镜像同样也能够为客户端应用提供统一的连接方式，但同时它又具备故障转移群集所没有的抵御数据损失的能力。相对日志传送，它进行灾难切换要快很多，又基本对客户端应用透明。这些优点都使得数据库镜像成为了一项兼备高可用性和灾难恢复功能的技术。现在，数据库镜像技术在企业客户环境中，使用得越来越多了。

### 2.4.1 数据库镜像的基本概念

相对前两种技术，数据库镜像技术要相对更加复杂一些。本节先介绍数据库镜像里的基本术语和角色。

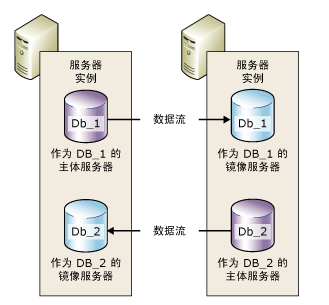
数据库镜像会为目标数据库创建一个副本数据库。这两个数据库分别运行在不同的SQL Server实例上，作为“伙伴”建立一个会话（session）。通过这个对话，两个数据库互相进行通信和协作，扮演互补的角色：“主体角色”和“镜像角色”，以实现“镜像”的效果。在任何给定的时间，都是一个伙伴扮演主体角色，而另一个伙伴扮演镜像角色。扮演主体角色的数据库就是主体数据库（principal database），另一个扮演镜像角色的数据库则是镜像数据库（mirror database）。每个主体数据库只能有一个镜像数据库。镜像数据库作为主体数据库的一个副本，在主体数据库发生故障、不可访问时，能迅速恢复数据库访问，提供了灾难恢复的功能。

只有使用完整恢复模式的数据库才能使用数据库镜像功能。主体数据库可以象一个普通数据库一样，执行数据库查询、修改、数据库管理操作。只有很少的操作（比如还原数据库备份）由于数据库镜像的限制而不被允许执行。而镜像数据库是一个一直处于“恢复”状态的数据库，因此是不能被直接访问的。这点和日志传送不一样。但是，你可以为镜像数据库创建一个快照，这样就可以间接地访问镜像数据库里的数据了。那些用作报表查询的应用可以通过镜像服务器的快照来获取数据，这能减轻主体服务器的工作负载。

需要特别指出，只有用户数据库才能配置数据库镜像，所有的系统数据库都不能被镜像，对于系统数据库还是需要通过备份的方式来保障安全。

运行主体数据库的SQL Server实例被称为主体服务器（principal server），而运行镜像数据库的SQL Server被称为镜像服务器（mirror server）。主体服务器和镜像服务器必须是两个不同的SQL Server实例。这两个实例上可以有多个数据库镜像的会话。这两个实例可以运行在同一台计算机上，但是一般不建议这么做。因为这种配置下，一旦服务器的磁盘发生物理损坏，那么主体数据库和镜像数据库就都无法使用了，灾难恢复也就无从谈起了。

下图（2-30）说明了作为伙伴参与两个镜像会话的两个服务器实例。一个会话用于名为 **Db\_1** 的数据库做镜像，另一个会话用于名为 **Db\_2** 的数据库。

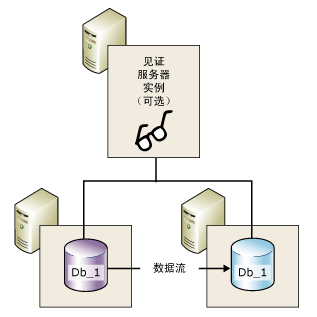


**图2-30 配置了多个会话的数据库镜像（无见证服务器）**

每个数据库镜像的会话都只针对一个用户数据库，而且是独立的。例如，一个应用程序需要同时访问一个SQL服务器实例上的两个数据库，那管理员就需要配置两个镜像会话。如果其中一个数据库发生故障，它的镜像会将数据库转移到镜像服务器上，而另一个工作正常的数据库则继续在主体服务器上运行。这时对SQL Server来讲，两个数据库都能被使用。但是对应用程序来讲，它就变得无所适从，连接哪个服务器都不能正常工作。由于镜像数据库相互独立，这些数据库不能作为一个组来进行故障转移。这是数据库镜像的一个很大限制。

除了主体服务器和镜像服务器之外，数据库镜像还可以配置一个叫做“见证服务器”（witness server）的SQL Server实例。见证服务器不是必须的。有了见证服务器，主体数据库和镜像数据库除了保持和伙伴的会话，也会和见证服务器进行对话。见证服务器上没有数据库的副本，它只是做为一个中立的仲裁，和主体/镜像服务器建立会话，判定数据库的健康状况，并在主体数据库发生异常的时候，触发自动的故障转移，让镜像数据库和主体数据库转换角色。见证服务器必须是一个独立于主体服务器和镜像服务器的实例。多个数据库镜像伙伴（无论它们是否使用相同的主体服务器和镜像服务器）可以共用一个见证服务器。最佳情况下，见证服务器实例应该配置在一台单独的计算机上，这样见证服务器的工作不会受到主体或镜像服务器异常的干扰。如果必须将见证服务器和镜像伙伴中某个实例运行在同一台机器上的话，要确保该机器具有足够资源以减少两个实例的资源争用。

可以说，配置了见证数据库的数据库镜像兼备了高可用性和灾难恢复的功能。如果数据库镜像没有配置见证服务器，就无法实现自动的故障转移。一旦主体数据库发生异常，数据库管理员需要手动进行故障转移。



**图2-31 配置了一个会话的数据库镜像（有见证服务器）**

主体服务器，镜像服务器和见证服务器实例上都会建有一个叫做“端点”（endpoint）的对象，每个端点都侦听在一个TCP的端口上，默认的端口是5022。主体服务器，镜像服务器和见证服务器三者就是通过建立在实例上的端点，向另两个服务器发送TCP数据包来进行“会话”。一个SQL Server实例上，无论配置有多少个数据库镜像会话，它们都只使用唯一的端点。如果这三个实例中的某两个运行在同一台机器上，要确保这两个实例上的端点侦听在不同的TCP端口上。

到这里，数据库镜像的基本概念讲得差不多了。那主体数据库和镜像数据库是如何同步数据的呢？

SQL Server数据库中任何的数据变化都会先记录到事务日志中去，然后才会真正更新数据页面。而事务日志是先保存在该数据库的日志缓存（log buffer）里，然后将缓存中的日志块固化到磁盘上的LDF文件中去。在数据库镜像中，主体服务器在将主体数据库的日志从日志缓存固化到磁盘的同时，还会使用另一个线程来将日志块发送到镜像服务器的端点。当镜像服务器通过端点接收到日志块之后，它先将日志块放到镜像数据库的日志缓存里，然后将缓存里的日志块固化到磁盘上。一旦日志块被固化之后，镜像服务器会根据日志来对镜像数据库执行“重做”（redo），最终更新数据页面。当镜像服务器重做日志时，镜像数据库实际就是在执行日志的前滚操作。如果重做失败，则镜像服务器通过将数据库置于 SUSPENDED 状态来暂停会话。数据库管理员必须找到问题的原因并解决问题才能继续会话。当主体服务器截断或收缩主体数据库的日志时，镜像服务器也将在日志的同一点收缩日志。

可以看到，数据库镜像其实就是通过发送日志来保持伙伴之间的同步。从SQL Server 2008开始，日志块在被主体服务器发送到网络之前会做压缩处理。这么做的目的是为了提升日志发送和接受的效率，降低日志块传输对网络链路和网络设备所带来的负载。对于那些异常繁忙的生产系统，这项功能不但降低了由于网络不胜负荷导致的镜像对话异常中断，也降低由于网络延迟导致的数据库镜像性能问题，可谓一举两得。

无论日志块压缩与否，主体数据库上新日志生成的速率都可能会快于日志发送的速率，有些日志记录会来不及发送，而在主服务器上暂时“积累”起来。这部分累积未发送的日志产生了一个队列，叫做发送队列（send queue）。发送队列不会占用主服务器上额外的存储空间或者是内存，它是存在于主体数据库的事务日志文件(LDF)上。事务日志里会有一个标记，指向所有没被发送到镜像服务器的日志记录中最早的那个。那么从这个标记开始到最新的日志为止就是“发送队列”。主体服务器就不断地从这个队列中取出最早的日志，并发送给镜像数据库。

在镜像数据库里，如果日志重做的速度如果赶不上日志在镜像端固化的速度，那么也在镜像数据库的LDF文件中等待重做的事务也会产生一个队列，叫做重做队列（redo queue）。和发送队列类似，重做队列也不使用额外的存储空间和内存。它仅是使用一个标记，指向镜像数据库的事务日志文件中最早的那个还未重做的日志。镜像服务器从这个标记开始按顺序重做日志，更新数据。重做队列中等待的未重做日志数量决定了故障转移到镜像数据库所要花费的时间。

通过上面的介绍可以发现，数据库镜像技术和日志传送一样，其同步的信息也是日志记录，而不是数据操作命令。它们不同的是，日志传送是通过数据库日志备份和恢复。而数据库镜像更“直接”，它能够直接读取和标记日志文件里的日志记录。所以镜像技术的实时性会更好。

### 2.4.2 数据库镜像操作模式

那数据库镜像能做到多“实时”呢？它能否保证镜像数据库完全与主体数据库同步呢？这取决于会话的运行模式。数据库镜像有3种操作模式：高可用、高保护和高性能。使用哪一种模式取决于两点：

1. 数据库镜像的“事务安全性”设置

2. 是否有见证服务器。

下面的表（2-3）列出了三种模式及它们之间的区别。

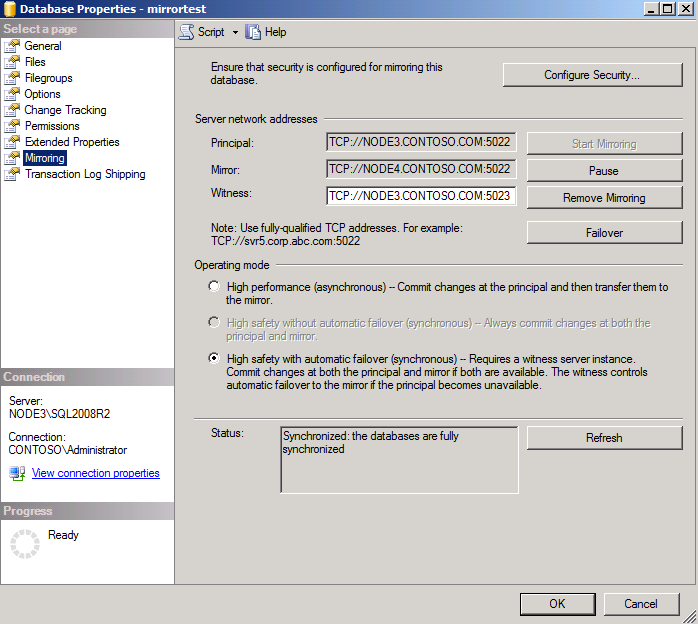
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 操作模式 | 事务安全 | 传输模式 | 见证服务器 | 故障切换类型 |
| 高可用 | Full | 同步 | 有 | 自动或手动切换 |
| 高保护 | Full | 同步 | 没有 | 只能手动切换 |
| 高性能 | Off | 异步 | 无需配置 | 只能强制切换 |

**表2-3 数据库镜像操作模式**

在对每个操作模式做详细介绍之前，先来了解下“事务安全性”和“传输模式”。

事务安全性是一个数据库镜像的设置选项。如果把它设置为Full的话，数据库镜像就处于“同步”传输模式。如果把该选项设置为off的话，数据库镜像就处于“异步”传输模式。

在SQL Server Management Studio中，如果你打开一个已经配置镜像的数据库的属性（图2-32），在”Operating Mode”这个区域你就能看到数据库镜像当前的事务安全性设置，并根据你的需求进行更改。



**图2-32 查看数据库镜像的事务安全性设置**

你也可以使用TSQL语句修改事务安全性设置。以下TSQL语句会将事务安全性设置为FULL:

ALTER DATABASE [<dbname>] SET SAFETY FULL;

同步传输模式和异步传输模式的区别在哪里呢？

同步传送模式下，数据库镜像是这样处理事务的：

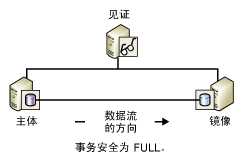
* 从客户端收到事务请求时，主体服务器将事务的日志写入事务日志中。
* 主体服务器将事务写入数据库中，同时将日志发送到镜像服务器。这时主体服务器不是提交或回滚事务，而是等待镜像服务器完成第三步。
* 镜像服务器将日志固化到磁盘，并向主体服务器返回确认。
* 主服务器收到来自镜像服务器的确认后，结束事务操作，并向客户端发送事务完成的确认消息。

用这样的方法，镜像服务器那里没有做完，主体服务器就不会结束事务。同步传送模式通过上述的过程来保证数据在主体数据库和镜像数据库上是始终保持完全同步的。所有被提交的事务都保证会被写入镜像服务器的磁盘上。这样，事务被认为是“安全”地在两个伙伴间传输，不会有两边数据不同步的现象。但是，同步传送模式会给主体服务器上事务的提交带来额外的延迟，即使延迟可能非常微小。所以数据安全是以牺牲一定的性能来达到的。为了尽量减小延迟，你需要给伙伴服务器之间配备高速可靠的网络，同时要确保伙伴服务器拥有良好的磁盘性能。一台I/O表现不佳的镜像服务器会因为日志固化的缓慢而拖累主体服务器的性能。一般情况下，推荐伙伴服务器具有相同的硬件配置。这既是为了保障较小的延迟，也是为了确保无论那台机器扮演主体服务器都会有相同的性能表现。

异步传送模式下，主体数据库向镜像服务器发送日志之后，不会等待镜像服务器的确认，而是直接向客户端发送事务完成的确认消息。主体数据库上提交事务不需要等待镜像服务器将日志写入磁盘。因此，主体数据库和镜像数据库可能不是完全同步的。镜像服务器会尽可能的“追赶”主体服务器，想与之保持同步。但是由于网络、负载等因素的影响，镜像数据库还是有可能在某种程度上滞后于主体数据库。尽管这两个数据库之间的同步差距通常很小，但一旦主体数据库突然出了问题，在镜像数据库上可能会有一部分数据丢失。所以这种方法性能好，但是不能完全保证数据同步。

了解了事务安全性和传输模式后，你很容易就能理解数据库镜像的每种操作模式。

**1. 高可用操作模式**



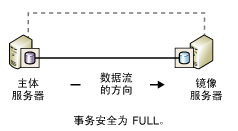
**图2-33 高可用操作模式的数据库镜像**

高可用模式要求数据库镜像的安全性设置必须设置为“完整”，因此日志传输模式是“同步”模式，这样保证了在主体数据库提交的事务必定在镜像数据库端也提交了。除此之外，必须配置见证服务器，能保证在主体数据库发生异常时自动触发故障转移，这是“高可用操作模式”命名的由来。

发生自动故障转移的场景是：见证服务器和镜像服务器保持会话，而主体服务器由于发生系统故障，中断了与镜像服务器和见证服务器的会话。这种情况下，镜像服务器会将镜像数据库的角色转换成“主体”角色。除了由见证服务器触发的自动故障转移，数据库管理员也可以执行手动故障转移。

高可用操作模式适用于主体服务器和镜像服务器之间具有高速且可靠的网络连接的环境，并且用户需要系统有自动故障转移的功能。需要注意的是，如果系统有多个数据库都配置了镜像，而应用程序需要同时访问到所有这些数据库才能正常运行，那其中一个数据库发生自动故障转移并不能触发其他镜像数据库的自动转移。这时这些数据库可能会分别运行在不同的SQL实例上，从而可导致整个应用程序不能同时找到所有数据库，最终无法使用。由于数据库镜像不能实现多个数据库一同转移，高可用模式只保证了数据库层面的可用，而不能保证应用层面的一定可用。这是数据库镜像技术的限制之一。

**2. 高保护操作模式**

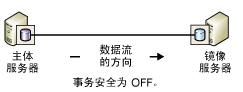


**图2-34 高保护操作模式的数据库镜像**

高保护操作模式要求数据库镜像的安全性设置必须设置为“完整”，但数据库镜像没有配置见证服务器。因为使用的是”完整”事务安全性，所以主体数据库和镜像数据库的是日志是保持同步的。由于没有见证服务器，没有一个仲裁的角色来判断哪个数据库出现了问题。所以一旦主体服务器无法继续工作，数据库管理员需要手动操作才能切换数据库的角色。当主体数据库和镜像数据库保持着会话且两边数据库已同步时，数据库管理员可以执行手动故障转移。如果镜像服务器实例出现故障，则主体服务器实例不会受到影响，主体数据库状态会变为“断开”连接的状态，不过依旧可以被访问。如果主体服务器出问题，则镜像对话就会被挂起，但可以将主体数据强制故障转移到原始的镜像服务器。强制切换可能造成一定量的数据丢失，因为原始的主体服务器上可能还有一些日志没有来得及发送到镜像数据库上。

如果数据库管理员不想让镜像自动切换，想要手动响应和解决主体或镜像服务器上的问题，高保护操作模式会比较适合。同样，使用高保护操作模式需要一个高速可靠的网络来支持。

**3. 高性能操作模式**



**图2-35 高性能操作模式的数据库镜像**

在高性能操作模式下，事务安全性被设置为关闭（off）。这意味着镜像数据库的日志可能会滞后于主体数据库（异步传输模式）。该模式下，主体数据库不需要等待镜像服务器接受和固化日志，就可以继续响应客户端的请求，因此不会受困于镜像服务器导致的延迟。相比较另外两个模式，处于高性能操作模式下的主体数据库会有更好的性能表现。

高性能操作模式下不需要见证服务器，唯一可以使用的切换方式是强制故障转移。这意味着高性能操作模式下的角色切换永远存在着损失数据的风险。但是这种模式，适用于对于性能要求非常高，不能忍受任何延迟的数数据库环境。如果网络状况不稳定，或者镜像的两个伙伴相隔很远，或者用户可用接受一定量的数据丢失，也可以考虑使用高性能操作模式。在现实应用中，很多用户选择了这种方式。

无论是哪种操作模式，镜像数据库总是努力跟上主体数据库的步伐，尽量保持数据的同步，在主体数据库问题发生时，你可以通过或手动或自动的方式来启用镜像数据库，保障你的应用。

### 2.4.3 客户端连接重定向及超时控制

和故障转移群集不同，数据库镜像无法提供一个虚拟的服务器名给使用数据库的应用程序。也就是说，应用程序需要知道当前的主体服务器是哪一个，才能连接上正确的SQL Server获得数据服务。一旦发生数据库镜像故障转移，用户需要通过修改连接字符串，建立别名等方式来将应用重定向到新的主体服务器。这极大地影响了数据库镜像的高可用性：对于一个高可用操作模式的数据库镜像，虽然可以自动进行故障转移，但是在手动重定向应用的连接之前，实际上应用还是不工作的。

为了增强数据库镜像的高可用性，必须能够使得应用程序可以意识到镜像服务器的存在，并且能够在主体服务器不存在时，自动重定向到那个镜像服务器。为此，微软为ADO.NET和SQL Native Client这两个数据库驱动程序添加一项特性：你可以在连接字符串中指定首选的服务器，同时也可以指定故障切换后的备选数据库。那些使用ADO.NET或者SQL Native Client来连接SQL Server的应用，通过这个特性可以获得自动重定向连接的能力。

下面是一个示例的连接字符串，Data Source属性指定的是首选服务器，而 Failover Partner属性指定的是备选服务器：

**"Data Source=SeverA;Failover Partner=ServerB;Initial Catalog=AdventureWorks;Integrated Security=True;"**

通常情况下我们把大部分时间扮演主体角色的服务器赋给Data Source参数，把多数时间扮演镜像角色的服务器赋给Failover Partner参数。

有了failover partner参数后，应用程序连接SQL Server的行为就变得比较复杂了：

* 当应用程序尝试连接到Data Source所指定的主体服务器并且失败的话，它就会去尝试连接Failover Partner指定的镜像数据库。因此，在数据库镜像发生故障转移（自动、手动或强制）之后，应用程序始终可以凭借这两个属性找到可以提供数据库服务的服务器。
* 如果应用程序尝试连接Data Source指定的主体服务器并且成功了，那么连接字符串中的Failover Partner属性指定的服务器就会被忽略掉。此时驱动程序会从通过主体服务器获得它的故障转移伙伴的名字，把这个名字缓存在应用程序的内存中。要注意的是，这个被缓存的名字会覆盖连接字符串中的failover partner属性值。也就是说，即使你在连接字符串中没有指定failover partner，甚至你有意把failover partner设置为一个错误的、非镜像服务器的实例，只要应用程序第一次能成功连接到主体服务器，应用程序就能获得正确的镜像服务器名称。

无论是通过上面两个情况中哪一种，只要应用程序得到了两个伙伴的名称，驱动程序就会遵循“连接重试算法”。不论是初次与会话建立连接，还是在中断已建立连接后重新连接，当应用程序成功无法连接到当时主体服务器后，就会去尝试连接故障转移伙伴。这时又会有两种情况：

1. 如果成功连接，那么就会读取新的主体服务器的镜像配置信息，得到新的镜像服务器名，并缓存在应用程序的内存里。
2. 如果连接失败，那么驱动程序会继续尝试去访问Data Source中指定的首选服务器。驱动程序就会这样不断的尝试这两个服务器，直到达到了连接超时的阈值。

这就是数据库镜像的“连接重试算法”。由于连接重试算法的存在，连接一个镜像数据库的时候，计算“超时”错误的方法也和连接普通的SQL Server数据库不同。当数据库是镜像数据库的时候，你在连接字符串中指定的connection timeout值和非镜像数据库的connection timeout值尽管大小一样，但实际上效果是不一样的。

假定你指定connection timeout 15 秒：

**第一轮**：连接Data Source指定的首选服务器，超时时间为总timeout的8% 即：15\*8%=1.2秒

这时候如果连接没有成功，就试图连接故障转移伙伴（无论是从缓存中的还是从failover partner参数中获得伙伴的名字），超时时间也是1.2秒。

**第二轮**：如果连接故障转移伙伴失败，这时候，客户端数据库接口再试图连接首选服务器，超时时间设为 总timeout的16% 即：15\*16%=2.4秒。 如果没有成功，就再次连接故障转移伙伴，超时时间也是2.4秒。

**第三轮**，超时时间是总timeout时间的24%。在后续的每轮中，连接尝试的重试时间会逐渐变大。前八次连接尝试的重试时间如下：

8%, 8%, 16%, 16%, 24%, 24%, 32%, 32%

重试时间使用以下公式进行计算：

RetryTime = PreviousRetryTime + ( 0.08 \* TotalLoginTimeout )

（PreviousRetryTime 初始值为 0）

例如，如果使用默认的登录超时期限 15 秒，则 *LoginTimeout = 15*。在这种情况下，前三轮中分配的重试时间如下：

|  |  |  |
| --- | --- | --- |
| **轮次** | ***RetryTime* 计算** | **每次尝试的重试时间** |
| 1 | 0 **+** **(**0.08 **\*** 15**)** | 1.2 秒 |
| 2 | 1.2 **+** **(**0.08 **\*** 15**)** | 2.4 秒 |
| 3 | 2.4 **+** **(**0.08 **\*** 15**)** | 3.6 秒 |
| 4 | 3.6 **+** **(**0.08 **\*** 15**)** | 4.8 秒 |

表2-4 重试时间的计算方式

下图（2-36）说明了这些后续连接尝试的重试时间，每个重试时间均超时。

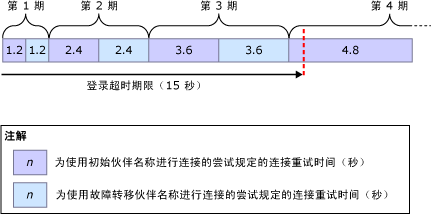


图2-36 数据库镜像连接重试及超时

也就是说，对于15秒的timeout值，实际连接服务器的最大timeout值是3.6秒！ 如果前面3轮连接（1.2秒，2.4秒，3.6秒）都不成功，在你还没有用尽第四轮的4.8秒时间之前（实际上此时第四轮只剩下0.4秒），15秒超时时间就到了，你的应用程序就会报超时错误。

要增加连接服务器的真实timeout值，你只能增加总的connection timeout值。比如，当我们将timeout设为120秒时，那么第一轮的重试时间是120\*8%=9.6秒。即便如此，第一轮的尝试连接时间还是比非镜像数据库的缺省15秒timeout时间要短。假设你现有的应用本就需要较长的连接超时，那么如果你需要把数据库配置镜像的话，一定记住要把connection timeout增大，否则你的应用可能非常容易就得到超时错误。

对于不是ADO.NET和SQL Native Client的驱动，即使数据库是镜像数据库，在应用程序端也不会缓存故障转移伙伴的名称，同时也不支持failover partner参数，因此就必须要考虑其他的方法来实现连接重定向，比如说在应用程序的代码中加入重试和重定向的逻辑。

其实，镜像切换还有许多需要考虑的事情。举例来说，数据库镜像只能保持用户数据库中数据的同步，而系统数据库是无法配置镜像的。也就是说，登录名、SQL Server实例的配置、维护计划、作业等保存在master、msdb或 model数据库里的信息在镜像的两台服务器上并不是同步的。一旦发生故障转移，即使数据库依旧可用，应用程序可能会因为各种原因无法正常工作，比如：

* 应用程序使用的登录名在新的主体服务器上不存在
* 新的主体服务器上虽然已经有同名的登录名，但是该登录名的SID和原始主体服务器上的登录名不相同。于是在故障转移后形成了孤立用户。这时可以用sp\_change\_users\_login存储过程来解决问题。
* SQL作业会使用一些SQL Server以外的资源（例如一些文件），但是这些资源在两台服务器上存放的路径不同。
* 两台服务器的配置不同，新的主体服务器由于不够优化而导致性能极差。

凡此种种，都是你可能面临的问题。如何在主体数据库和镜像数据库间同步用户数据库以外的信息，是一个必须考虑的问题。

### 2.4.4 数据库镜像的监控和故障排查

要了解数据库镜像的运行状态，对发生的问题进行故障排查，我们可以收集以下数据。

1. SQL Server的ERRORLOG。

在主体服务器，镜像服务器和见证服务器上，ERRORLOG中都会记录每个镜像数据库所处的状态，每个数据库当时处于什么角色，以及服务器彼此间的连通状况。

比如，这段日志表明镜像的端点已经成功监听在了端口5022上。

2006-03-29 00:34:27.87 spid12s Server is listening on [ 'any' <ipv4> 5022].  
2006-03-29 00:34:27.87 spid12s The Database Mirroring protocol transport is now listening for connections.

下面这段日志说明数据库处于镜像角色

Database mirroring is active with database 'xxx' as the mirror copy. This is an informational message only. No user action is required.

如果发生连接故障的话，ERRORLOG中会记录镜像数据库的状态变化，以及连接问题的错误代码。这些错误代码对我们分析和定位连接会非常有用。这里举一些例子：

1. 由于发生了自动故障转移，数据库转换了角色。

镜像服务器的ERRORLOG:

The mirrored database "XXX" is changing roles from "MIRROR" to "PRINCIPAL" due to Auto Failover.

主体服务器的ERRORLOG:

The mirrored database "XXX" is changing roles from "PRINCIPAL" to "MIRROR" due to Role Synchronization.

1. 伙伴服务器之间发生网络连接错误。

2011-06-05 07:51:10.73 spid40s Error: 1474, Severity: 16, State: 1.

2011-06-05 07:51:10.73 spid40s Database mirroring connection error 4 'An error occurred while receiving data: '10054(An existing connection was forcibly closed by the remote host.)'.' for 'tcp://test.testdomain.com:5022'.

2011-06-05 07:51:17.27 spid40s Error: 1474, Severity: 16, State: 1.

2011-06-05 07:51:17.27 spid40s Database mirroring connection error 4 'An error occurred while receiving data: '64(The specified network name is no longer available.)'.' for 'tcp://test.testdomain.com:5022'.

1. 伙伴服务器不响应连接请求，可能是出现了严重的性能问题。

2007-10-24 23:22:22.87 spid30s Error: 1479, Severity: 16, State: 2.  
2007-10-24 23:22:22.87 spid30s The mirroring connection to "tcp://test.testdomain.com:5022" has timed out for database "xxx" after 30 seconds without a response. Check the service and network connections.

除此之外，ERRORLOG中当然还记录了各种SQL Server遇到的异常，这些异常可能是非常重要的参考信息，可以用来推断数据库镜像发生故障转移或者无法正常工作的原因。

1. 性能监视器。

数据库镜像在性能监视器中有一个专门的对象SQL Server:Database Mirroring。这个对象下的计数器对于了解数据库镜像运行的状态，有着非常重要的作用。

在主体数据库上，你可以检查 Transaction Delay 计数器以确定数据库镜像是否影响主体服务器的性能。检查Log Send Queue KB计数器，以确定发送队列的大小。如果send queue很长的话，有可能是镜像数据库处理日志慢导致的，也可能是由于网络传输慢导致的。主体服务器的 Log Bytes Sent/sec 计数器显示了每秒发送的日志量。可以计算“(Log Send Queue)/( Log Bytes Received /sec)”来估算镜像数据库需要多长时间才能追赶上主体数据库。

在镜像数据库上，Log Bytes Received/sec计数器显示了每秒接收到的日志量。你可以检查Redo Queue KB计数器来确定重做队列的大小，就知道有多少日志没有被重做。如果Redo queue很大的话，这往往意味着镜像数据库存在I/O瓶颈。镜像数据库上的Redo Bytes/sec计数器显示了执行重做操作的速率。前面已经提到过，发生故障转移的时候，原始的镜像数据库需要把重做队列里的日志都重做完才能真正上线运行。我们可以通过计算“(Redo Queue)/(Redo Bytes/sec)”来估算一旦发生故障转移，需要多长时间才能完成全部重做操作。

这些计数器，配合一些系统自带的计数器（例如，物理磁盘，网络等对象下的计数器），可以很直观地定位到数据库镜像的性能瓶颈。

1. 数据库镜像监视器。

数据库镜像监视器是一个图形用户界面的工具（图2-37），你可以用它来来监视镜像镜像伙伴以及镜像会话的状态，利用这些信息你可以评估在当前状况下完成故障转移所需的时间，和潜在的数据丢失量。你也可以为一些您认为很重要的指标（如send queue，redo queue）配置警告阈值，一旦超过你设置的阈值就触发警告来通知数据库管理员。

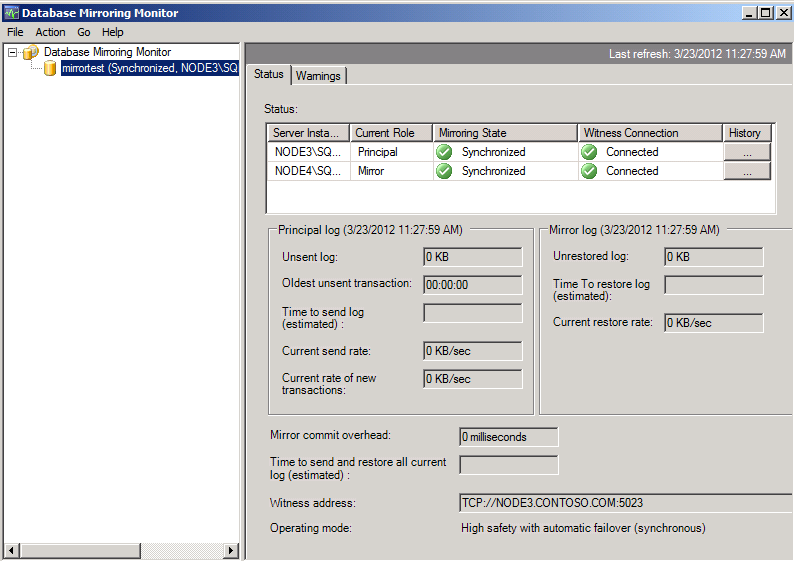


图 2-37数据库镜像监视器

1. 系统存储过程和系统表

可以通过运行 sp\_dbmmonitorresults 系统存储过程来查看当前数据库镜像的状态。你可能会感兴趣数据库镜像监视器或者sp\_dbmmonitorresults是从哪里获取信息的。其实，这两者都是来自于一个MSDB数据库中的系统表dbm\_monitor\_data。而这个系统表的更新由以下两者完成：

1. 当使用 SQL Server Management Studio 启动镜像会话时，会自动创建一个“数据库镜像监视器作业”。这个作业每分钟更新一次dbm\_monitor\_data系统表。如果你是使用 ALTER DATABASE database\_name SET PARTNER 命令来创建镜像会话的话，这个作业就不存在了。但是你可以执行sp\_dbmmonitoraddmonitoring 存储过程来手动创建这个作业。
2. 数据库镜像监视器每30秒更新一次dbm\_monitor\_data系统表。

事实上这两个方法背后都是调用了sp\_dbmmonitorupdate存储过程来更新信息。你可以手动运行这个存储过程来更新dbm\_monitor\_data。无论你使用什么方法，SQL Server会保证每两个更新的间隔不小于15秒。

sp\_dbmmonitorupdate是一个非常重要的存储过程。如果你有兴趣的话，你可以去查看它的定义，你会发现这个存储过程是通过查询sys.dm\_os\_performance\_counters这个视图来获得性能相关的数据，并且sp\_dbmmonitorupdate通过将用户设置的阈值与sys.dm\_os\_performance\_counters进行比较来确定是否并触发警告。而sys.dm\_os\_performance\_counters中的数据和性能监视器中SQL Server:Database Mirroring下的计数器是一一对应的。

以上就是诊断数据库镜像问题时通常需要收集的信息。数据库镜像的问题主要是两类：

1. 诊断发生故障转移的原因（对于非高可用模式的镜像，就是诊断镜像停止工作的原因）。

对于这类问题，你需要收集主体服务器，镜像服务器和见证服务器（如果有的话）上的ERRORLOG。如上面所讲的，ERRORLOG里会记录什么时候镜像发生了问题，发生了什么类型的问题，以及最终的结果如何。最重要的是，你不能孤立地去分析各个ERRORLOG，而是应该将它们彼此作为参考，互相印证。举例来说，如果你在主体服务器的ERRORLOG里发现了主体服务器失去了镜像服务器的连接，你就应当去查看镜像服务器和见证服务器ERRORLOG中相应时间点的日志，去了解当时镜像服务器和见证服务器是否能连接到主体服务器。只有了解了所有服务器彼此之间的连通状况，才能正确判断出当时是那台服务器的网络端出现了问题。

2. 性能问题。

由于只有主体服务器才能影响应用端的请求，性能问题都会体现在主题服务器端。例如主体服务器上事物提交有严重的延迟，或者是发送队列很长等等。但是这类问题的最终症结往往却不在主体服务器上，而在镜像服务器上或者是在网络上。要诊断这类问题，需要在主体服务器和镜像服务器上打开性能监视器来记录各项镜像相关的数据，以及磁盘、网络、内存、CPU相关的数据。此外数据库镜像监视器也能帮助你了解主体/镜像服务器的运行状态。通过这些数据，你可以了解镜像服务器的处理事务的效率，网络传送事务的速度，以及镜像端的磁盘负载和磁盘性能（往往是最大瓶颈）等情况。掌握了这些信息，你就可以判断出问题的源头：是否主体服务器的负载已经超出了镜像数据库的容纳范围，是否网络性能低于设计容量，等等。

## 2.5 复制

复制（replication）也是SQL Server上一个非常“资深”的功能了。它采用了独特的“发布-订阅”的模式，将SQL Server中的数据分发到各个服务器或客户端上去。

需要说明的是，复制并不是一个为灾难恢复而设计的的功能，它更注重的，是数据同步。用复制来作为灾难恢复的解决方案，往往是使用较多的代价和资源，但只是实现的效果很一般，可以说是性价比不高。大多数情况下，你可以使用日志传送或者数据库镜像，以更低的代价获得类似的甚至更好的效果。复制真正强大的地方是在于它灵活的数据同步功能。比如，它能够在多个数据源之间进行数据的同步与合并，移动设备可以通过复制功能从SQL Server上同步到它所需要的数据等等。

由于复制不是一个“合格”的灾难恢复/高可用技术，本书只对复制做简单的介绍，不会深入地展开。

### 2.5.1 复制的基本概念

复制使用了出版发行领域的术语来描述它的组件。虽然这些术语无法完全覆盖复制的各种功能，你依旧可以通过这些术语来理解每个组件在复制的拓扑结构所处的位置和作用。

**项目（Article）**

项目是SQL Server数据库中的对象，它可以是表、视图、存储过程，自定义函数或其他的对象。项目就好比是杂志中的文章，文章可以有不同的类型，包括杂文，传记，新闻等等。如果项目是一个表的话，你可以通过添加筛选条件来过滤掉表中的某些行和列，这就好比是对文章经行了编辑工作，确保了订阅者可以只会收到有限的内容。

**发布（Publication）**

发布是一个或多个项目的集合， 一次发布可以包含不同类型的项目。发布就好比是一本杂志，其中包含了各种类型的文章。发布是用于复制的基本单位。 通常我们将逻辑上相关联的数据库对象（比如外键和被外键关联的两个表）组合在一起形成一个发布，这样可以确保被复制的内容在逻辑上一致。

**发布服务器（Publisher）**

发布服务器就好比是杂志的出版商，它生产一种或者多种杂志提供给订阅者。在复制的结构中，发布服务器是一个数据库实例，上面配了一个或多个发布。这些发布中的对象和数据通过复制发送给其他的服务器或客户端。

**分发服务器（Distributer）**

分发服务器也是一个数据库实例，它服务于一个或多个发布服务器。 每个发布服务器都对应分发服务器中的一个数据库，这种关联的数据库被称作分发数据库。 分发数据库上保存着复制的状态信息以及每个发布的元数据。从发布服务器发往订阅者的数据会在分发服务器上排队，依次发送出去。分发服务器就好像是发行商，杂志通过它们到达最终的订阅者手中。

**订阅（Subscription）**

订阅就是将发布从分发服务器发送到订阅服务器上，它定义了“什么时间，从哪里，得到哪些发布”。这就好比一个订阅者要获得他想要的杂志，需要知道什么时候这期的杂志会出版，到哪里可以买到，以及买什么样的杂志。根据数据发送的方式，订阅有两种类型：推送订阅和请求订阅。

推送订阅：发布服务器主动将数据更改发送到订阅服务器，而无需订阅服务器发出请求。 就好比你是订阅杂志的用户，杂志会由发行商寄到你家来。你有三种选择来决定何时把发布推送到订阅服务器上：手动发送，连续发送，按计划时间定期发送。

请求订阅：订阅服务器主动请求去获得发布服务器上所做的更改。 这就好比订阅者是主动去到发行商的销售网点去买杂志。这种模式下，接收发布的时间是由订阅服务器来确定的。

**订阅服务器 （Subscribers）**

订阅服务器就好像是杂志的订阅者。订阅服务器是接收复制数据的数据库实例。 订阅服务器可以接收来自多个发布服务器的数据。 根据所选的复制类型，用户可以在订阅服务器上对复制过来的数据做修改，然后将修改发送回发布服务器或者将数据重新发布到其他订阅服务器上。订阅服务器不一定是SQL Server，Oracle 和 IBM DB2 也可以使用推送订阅方式来订阅“快照”和“事务发布”。关于快照和事物发布的概念，稍后我们再讲。

把上面这些组件全部组合起来，就形成了数据如何从发布服务器发送到订阅服务器的全过程。通过下面的图（2-38）你能清楚的了解整个复制的拓扑结构。

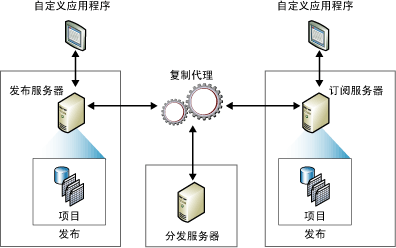


图2-38 复制的拓扑结构

### 2.5.2 复制的类型

SQL Server的复制有3种基本类型：快照复制，事务复制以及合并复制。要选择一种适合的复制的类型，你需要考虑多种因素，包括：被复制数据的类型和数量，是否在订阅服务器上更新数据，复制中所涉及的计算机数量和位置，以及这些计算机是客户端（工作站、便携式电脑或手持设备）还是服务器，等等。

**快照复制（Snapshot Replication）**

快照代理将数据库的某个瞬时状态生成一个快照，并将这个快照发送到订阅服务器。由于快照文件通常都比较大，因此快照复制同步数据的时候会比较长。而且快照是静态的，复制不会将数据变化自动传递给订阅服务器。

下图（2-39）显示了快照复制的主要组件。

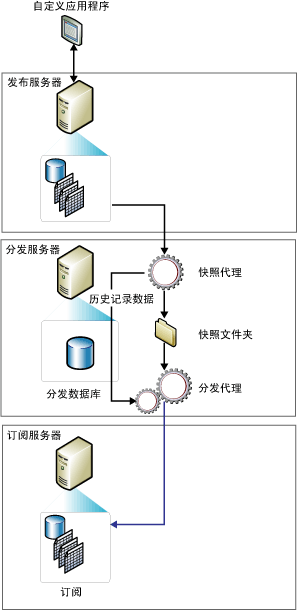


图2-39 快照复制的相关组件

可以看到，在快照复制的过程中，快照是通过文件夹来传送到订阅服务器上的，它没有经过分发服务器。但是快照代理程序会在分发数据库中记录历史信息。

当你的复制环境符合以下条件之一时，选择快照复制会比较合适：

* 数据很少更改数据。
* 用户允许订阅服务器使用相对已过时的数据副本。
* 数据复制量小。
* 数据库在短期内出现了大量更改。

在数据每次更改量很大，但更改次数不多时，快照复制是最合适的。（例如，用户用一句Delete语句，一次删掉了100万条记录。）另外，如果发布的表比较小，且可以接受一定的同步滞后时间，也可以选择快照复制来传递表上的数据更改。

**事务复制（Transactional Replication）**

快照复制每次同步的是整个订阅的内容。如果发布包含非常大的表，那么势必每次同步都需要很长时间，同步间隔也会很长。事务复制能够有效地解决快照复制的这些缺陷，实现快速的数据同步。

下图（2-40）显示了事务复制的主要组件。

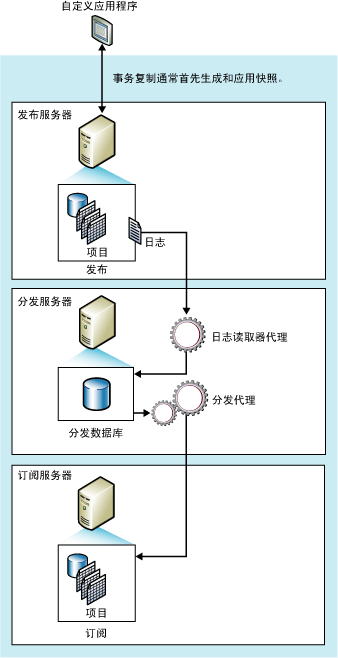


图2-40 事务复制的相关组件

SQL通过日志读取代理和分发代理程序，将发布服务器上所做的数据更改和架构修改几乎实时地传递给订阅服务器。 所有的数据更改都会以事务为单位，按照其在发布服务器上发生的顺序，应用订阅服务器上。这样，订阅服务器就获得了与发布服务器相同的数据副本。

这种工作流程下的事务复制，所有的数据或架构更改必须在发布服务器中进行；而订阅服务器应作只读处理，不应发生任何更改，因为这里的更改不会传播回发布服务器。但是，如果你真的需要在订阅服务器上进行数据更改并且希望这些更改能回传到发布服务器上，有两个选择：

1. 可以通过使用事务复制的“立即更新”或“排队更新”选项，来启动所谓的“可更新订阅的事务复制”。使用排队更新选项时，SQL Server在事务复制中又引入了第三种代理程序 - 队列读取器代理 (Queue Reader Agent)。队列读取器代理运行于分发服务器，它用于将订阅服务器上所做更改传递回发布服务器。“可更新订阅的事务复制”可能会发生订阅服务器和发布服务器之间的更新冲突。
2. 在SQL Server 2005，加入了一种新的事务复制类型，叫做“对等事务复制”。在对等事务复制的拓扑结构中，可以有多个数据库实例彼此互为订阅服务器和发布服务器，并且对于彼此间的更新冲突，对等事务复制也定义了冲突解决的机制。下图（2-41）是简单的对等事务复制的拓扑结构图。

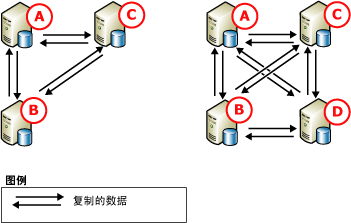


图2-41 对等事务复制

关于“可更新订阅的事务复制”和“对等事务复制”，碍于篇幅这里就不详细展开了。你可以参考SQL Server的联机丛书来获得更多的信息。

事务复制是一种最常见的复制方法，通常用于服务器到服务器环境中。在以下各种情况下适合采用事务复制：

* 只想把数据更新发送给订阅服务器而不是整个数据集。
* 需要较短的同步时间，数据更改要较快地从发布服务器上传送给订阅服务器。
* 订阅服务器上的应用环境需要追踪数据更改的中间状态。 例如，如果某一行在发布服务器上更改了五次，在订阅服务器上要针对每一次更改都触发相应的操作（例如，激活五次触发器）。
* 发布服务器有大量的插入、更新和删除动作。
* 发布服务器或订阅服务器不是 SQL Server 数据库。

**合并复制（Merge Replication）**

不管是快照复制还是事务复制，数据修改只能在发布服务器上。而合并复制能允许用户同时修改订阅服务器和发布服务器上的数据，并把这些修改“合并” 成一个统一的结果。由于更新是在多个服务器上进行的，同一数据可能在发布服务器和多个订阅服务器上都进行了更新。 因此，在合并更新时可能会产生冲突。为此合并复制还提供了多种处理冲突的方法。

合并复制的实现机制最为复杂。下面（2-42）的关系图显示了合并复制中的主要组件。

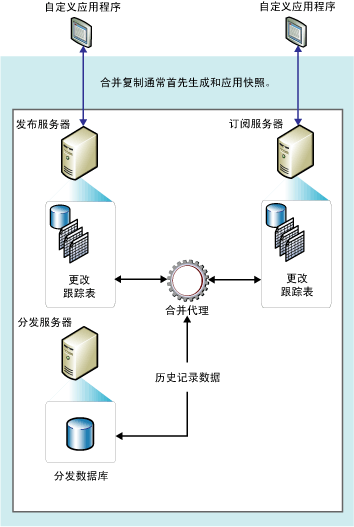


图2-42 合并复制的相关组件

合并复制适用于下列情况：

* 多个订阅服务器可能会在不同时间更新同一数据，并将其更改传播到发布服务器和其他订阅服务器。
* 订阅服务器需要接收数据，脱机更改数据，并在以后与发布服务器和其他订阅服务器同步更改。
* 多个服务器同时更改一个数据，可能会发生引发冲突。你需要具有检测和解决冲突的功能。
* 应用程序需要最终的数据更改结果，而不是访问中间数据状态。 例如，如果在订阅服务器与发布服务器进行同步之前，订阅服务器上的行更改了五次，则该行在发布服务器上仅更改一次来反映最终数据更改（也就是第五次更改的值）。

到这里，已经介绍了三种基本的复制类型。不管是那种类型，它们最终的目的都是把数据从发布服务器同步到订阅服务器。但是每种类型的复制跟踪数据更改的方式都不同。快照复制不会跟踪快照生成后任何的数据更改，因此要同步数据变更就需要每次都把新的快照应用订阅服务器上，完全覆盖现有数据。事务复制通过 SQL Server的 事务日志跟踪更改，复制以事务为单位将更改发送到订阅服务器。而合并复制则通过触发器和系统表来跟踪数据和架构更改。一般情况下，数据更改都发生在发布服务器上，订阅服务器上不应有数据更改；但是合并复制、对等事务复制以及可更新订阅的事物复制允许在订阅服务器上进行更改，并使这些更改流向发布服务器。

### 2.5.3 灾难恢复和复制

一个灾难恢复的方案，必定需要一个和主数据库保持同步的备用数据库。因为复制可以创建并维护一份数据副本，所以也可以用来创建备用数据库。在作备份数据库时，一般用户都希望同步的间隔能越短越好，最好是完全同步；并且希望同步给主服务器带来的负担越少越好。基于这个要求，如果要使用复制来作为灾难恢复的方案，事务复制是最佳的选择。

当使用事务复制来作为灾难恢复方案时，应当把主服务器选为发布服务器，把备用服务器选为订阅服务器。关于订阅模式，一般我们建议使用推送订阅，并且设置为“连续发送”，这样分发服务器会立刻将接收到的发布给发送到订阅服务器上。一旦发布数据库出了问题，订阅数据库上还有副本数据可以使用。复制的订阅服务器是可以访问的，因此你可以将一些只读的应用，如报表系统，运行在订阅服务器上，这能够分流发布数据库的负载。但是，由于数据库是用于灾难恢复的目的，你应该禁止任何人或应用更新订阅服务器上的数据。

一般来说，事务复制的同步滞后时间会小于日志传送，但是无法像数据库镜像技术那样实现完全同步和数据零损失。同时，事务复制带来的额外负载一般会高于日志传送。事实上，事务复制最适合用于数据库对象级别的复制，例如仅复制一个或几个表，而不是复制整个数据库。事务复制的强大之处在于它可以只同步几个表，甚至是表中的部分数据，这是数据库镜像和日志传送都无法达到的功能。微软也建议只在同步比较小量数据的情况下使用事务复制作为灾难恢复方案。

另外，复制只同步表格里的数据，而不会同步其他数据库对象，例如，数据库用户、存储过程、索引、约束、外键、触发器等。仅仅使用复制，是无法得到一个完整的数据库的。应用程序一般无法直接使用订阅服务器上的数据库。用户常常是在主服务器出现问题的时候，先恢复数据库结构，然后将订阅服务器上的表导回到主服务器（发布服务器）。在服务器上恢复数据库的某几个表往往比恢复整个数据库更困难，你除了需要确保表中的数据恢复回来之外，还要确保和表相关的对象（索引，约束，外键，触发器等）也被全部恢复回来。

所以，复制是一个用来实现数据同步的技术，而不是设计用来实现灾难恢复的。复制作为灾难恢复方案，存在以下这些问题：

1. 复制是以表和其他数据库对象为单位的数据同步功能，使用系统表来维护所有表的变更，因此需要大量的额外开销（如触发器等）来监控数据的变更并维护系统表。如果要把整个数据库中的所有内容都发布出去，则这部分开销会变得非常大，一般是不建议的。
2. 复制在主服务器（发布服务器）和备用服务器（订阅服务器）之间，还使用一个额外的角色分发服务器来做中转。这额外增加了同步的开销并且使滞后时间更长。复制需要使用不同的代理程序来传送数据，这些代理程序都是独立于SQL Server进程以外的可执行程序，这些程序和SQL Server之间的进程交互也增加了复制带来的开销。
3. 复制需要将事务日志文件的内容解析成二进制格式的命令写入系统表中，然后再将这些命令发送到订阅端的数据库上去执行。这些造成了两次数据传递过程。当变更量非常大的时候，这样的过程会造成较长时间的延迟。
4. 复制对表的定义有要求，不是所有的表格都能做复制。它需要在所有参与复制的表上创建主键或者增加rowguid的列。因此实施复制需要在数据库设计阶段就要考虑。对已经上线的系统中使用复制功能，可能需要一些二次开发的工作。
5. 复制非常依赖分发数据库来作为数据的中转。一旦分发数据库出了问题，保存在分发数据库中那些还没发送出去的更新会全部丢失，也就是说分发数据库的存在增加了整个灾备方案失败和数据损失的可能性。此外，一旦分发数据库出问题，整个复制也需要重建，重建意味着重新在订阅服务器上做初始化，这都是费时费力的工作。
6. 与数据库镜像和日志传送不同，订阅数据库本身不限制写操作。也就是说从系统层面上复制无法杜绝在订阅数据库发生更新操作，而只能依靠人为的管理和安全措施。因此，订阅服务器上数据的可靠性是存在一定风险的。

## 2.6 高可用和灾难恢复技术的选择

通过前面的介绍，相信你对各种高可用和灾难恢复技术已经有一定程度的了解。这些技术都已经被广泛的应用在了全世界的各种企业环境中，为应用持续运行提供保障。

### 2.6.1 高可用和灾难恢复技术的比较

每一种技术都有其优点和局限，这些因素决定了它适用于怎么样的环境。要比较各种不同的高可用和灾难恢复技术，就要了解它们的优点和局限。这样才能根据你的需求选择最合适的一款。

**故障转移群集**

故障转移群集是SQL Server最早的高可用技术，它的功能也随着每次SQL Server的版本升级而得到加强。它的优势包括：

* 得益于Windows故障转移群集，SQL Server故障转移群集可以自动检测SQL Server的健康状况，并在发生故障的时候自动进行切换。
* 切换所需的时间基本等于SQL Server服务重启的时间，如果没有太大的事务需要前滚，SQL Server可以很快完成数据库恢复并继续提供服务，停机时间一般比较短。
* 提供了一个虚拟网络名给客户端。对客户端而言，当前是群集的哪个节点在运行SQL Server是完全透明的。服务器发生故障切换后，客户端能够自动切换连接，不需要修改任何配置。
* 从SQL Server 2008开始，你可以现在非活跃节点上进行升级维护的工作，然后手工故障转移到这个升级后的节点，最后再升级维护之前那个活跃节点。于是整个升级维护带来的停机时间，只是SQL Server做一次故障转移（failover）所花的时间。

故障转移群集的限制包括：

* 由于节点间使用的是共享磁盘，数据库只有一份。一旦数据库损坏，或者共享磁盘出现故障，整个SQL Server就瘫痪了。可以说故障转移群集完全无法提供灾难恢复的功能。
* 非活跃节点上的SQL Server实例完全处于停止状态，不能提供给报表等程序使用，无法分担活跃节点的负载，浪费了一定的硬件资源。
* 实施成本相对较高。群集需要标准版或以上的SQL Server和Windows Server。用来实施Windows群集的硬件以及硬件设置必须符合微软的硬件兼容列表的要求（Windows 2003群集）或者通过Windows群集验证工具的检查（Windows 2008群集），否则群集无法建立。
* 故障转移群集发生在整个实例范围。个别数据库的问题可能会触发整个实例故障转移，导致那些“无辜”的数据库也发生了短暂的停机。
* 维护群集环境需要额外的成本。举例来说，对于一个活跃/非活跃模式的SQL Server实例，每次对这个实例进行维护的时候，都需要同时维护两个节点来保持两个节点一致。相对单机而言，维护成本上升了一倍。
* 虽然群集支持将几个节点部署在相隔很远的物理位置，但是这种部署都网络和存储的要求迥异于普通的群集，实施的成本会高很多。

**日志传送**

日志传送一直以其“物美价廉”的特点而获得SQL Server用户的青睐。虽然它的功能不是最强大，但是它简单易用的特点和几乎不给主服务器带来额外负担的特性，使得它依旧是生命力非常强大的技术。日志传送可以用于补充或替代数据库镜像。虽然异步数据库镜像与日志传送在概念上很相似，但它们仍有很大差异。日志传送的优势包括：

* 辅助数据库可以设置为只读模式，在上面可以运行例如报表之类的只读应用，能够一定程度上分担主数据库的负担。
* 支持多个冗余数据库副本。针对单个主数据库可以在多个服务器实例上配置多个辅助数据库，只读的辅助数据库用于报表，而处于还原模式的辅助数据库作为灾难恢复的备用数据库库。
* 日志传送所需的软硬件成本低，工作组版本的SQL Server就可以使用日志传送，对硬件也没有特殊的要求。
* 由于使用的是备份/还原的方式来同步主数据库和辅助数据库，且备份操作的执行有一定的时间间隔，因此对主服务器的性能带来的额外开销比较小。
* 如果日志传送的备份/复制/还原作业设置的相对较长，则意味着主数据库和辅助数据库间的同步间隔较长。如果主数据库上的数据被误操作意外更改或删除，由于较长的同步间隔，您可以在辅助数据库上得到仍未更改或删除的数据。管理员可以用辅助数据库上的数据，恢复误操作。
* 日志传送的工作原理决定了它对网络的速度和可靠性的依赖程度相对较低，可用于非常远距离的异地灾备。
* 较简单的实现机制，一旦日志传送不能正常工作，问题排查起来相对简单。

再来看看日志传送的局限所在：

* 不支持在不同版本SQL Server间实现日志传送。
* 无论把作业执行的间隔设置的多短，日志传送都不能达到完全的数据同步。也就是说一旦主数据库发生故障，你有可能会损失一部分数据。相比较而言，同步模式的数据库镜像可以做到完全同步，确保在主体数据库出现故障时，不会有数据损失发生。
* 日志传送不提供自动侦测和解决故障的能力，无法提供自动故障切换。主体数据库发生故障后，管理员必须人工发现问题，然后手动执行故障转移才能解决问题。整个故障转移的操作步骤相对较繁琐，停机时间较长。
* 日志传送。一旦执行了故障转移后，日志传送就被破坏了，需要重新建立。
* 不能为客户端提供一个自动切换的连接接口。当发生故障转移后，客户端需要做手动干预才能被重定向到新的服务器。
* 虽然辅助数据库可以被配置为“只读”。但是在还原作业正在进行还原日志操作的时候，整个数据库是不可用的。所以辅助数据库不是每时每刻都能够被访问的。
* 日志传送提供的同步是以数据库为单位的。你无法使用日志传送来仅同步一个数据库中的部分对象，也无法同步系统数据库。

**数据库镜像**

很多用户对数据库镜像技术给予厚望，因为它是SQL Server第一个同时能够实现高可用性和灾难恢复的技术。也就是说，它既能够在系统发生故障时，实现一定程度上的自动切换，也能够提高数据冗余拷贝，预防数据库损坏灾难。在它身上，集合着故障转移群集和日志传送的优点。具体来讲，它的优势有：

* 数据库镜像提供了一个自动化的、完全同步或者基本同步的数据库副本。配置了镜像，就能在镜像服务器上得到一份和主服务器上一模一样的数据库备份。只要镜像工作正常，这两份数据完全一致，或者差距不大。这个比自己配置备份恢复任务要省心，效率也比较高。一旦主数据库出现故障，你可以用这个副本进行灾难恢复，而不会承受（或者承受很少的）数据丢失。
* 在高可用操作模式下，数据库镜像会在侦测到故障时自动经行角色切换，其高可用性可以和群集技术媲美。在高保护操作模式/高性能操作模式下，虽然需要人工干预执行切换，但是其切换的速度比日志传送要快很多，难度也低。
* 数据库镜像的故障转移是数据库级别的，发生角色切换后，只要数据库完成恢复操作就可以正常使用。相比较群集的实例级别的故障转移，数据库镜像切换的速度更快。
* 在客户端使用SQL Native client或者ADO.NET的时候，客户端能够在连接字符串中加入failover partner属性，这样客户端就知道了镜像数据库的名字。当发生镜像数据库自动角色切换后，客户端会根据连接重试算法自动重定向到新的主体服务器上，不需要人为干预来手动重定向客户端到SQL Server的连接。
* 同步模式下，数据库镜像能确保两个伙伴服务器完全同步，就算发生故障转移也不会有数据损失。而异步模式下，数据的同步也远比日志传送更快。
* 数据库镜像对服务器的硬件和硬件配置没有特殊要求。
* 在对SQL Server进行升级的时候，你可以先升级镜像服务器，然后手动执行镜像角色切换，再升级新的镜像服务器。这样停机时间可以缩短为一个故障切换的时间，提高了数据库系统的可用性。

但是，数据库镜像技术还不能完全替代故障转移群集和日志传送技术。它的局限性主要表现在：

* 主体数据库和镜像数据库必须是相同版本的SQL Server。这个限制故障转移群集和日志传送也都有。
* 一个数据库镜像管理和切换的是一个数据库。数据库镜像无法使多个数据库一起切换，这个对于需要同时使用多个数据库的应用，其高可用性还不够完全。
* 数据库镜像的同步是以数据库为单位的。你无法使用日志传送来仅同步一个数据库中的部分对象。
* 一个主体数据库只能配置一个镜像数据库。不支持多个冗余数据库副本。
* 系统数据库无法配置数据库镜像。因此，登录、SQL Server实例的配置、维护计划、作业等保存在master、msdb或 model数据库里的信息不能自动同步到镜像服务器上，需要管理员事先同步好。
* 对于非ADO.NET和SQL Native Client的客户端，在镜像发生切换后无法自动重定向到新的主体服务器。对于那些以前开发的使用ADO.NET和SQL Native Client的客户端，如果要获得自动重定向的特性，也需要修改连接字符串加入failover partner属性。从这点上来讲，数据库镜像还是达不到群集技术对客户端那样的透明度。
* 内部实现机制复杂。一旦数据库镜像没有按预期那样的工作，故障排查的难度比较高。
* 镜像数据库是不可访问的，无法在上面运行报表程序来分担主体服务器的负载。一个变通方案是给镜像数据库做做一个快照，然后在快照上运行报表程序。不过这样一来报表程序获得的数据必定滞后于实际的数据。
* 在同步模式下，为了实现零数据丢失，会在主体数据库上使用延迟提交，这一定程度上影响了主体数据库的性能。
* 同步模式虽然实现了零数据丢失，但是对网络的速度和可靠性要求非常高，因此不适合作为一个远距离的异地灾备方案。
* 在负载较高的情况下，对镜像服务器的磁盘性能，主体服务器与镜像服务器间的网络速度及可靠性的要求比较高。

**事务复制**

正如前面提到的，复制本身并不是一个很好的灾难恢复技术。复制更适合用于纯粹的数据同步目的，而不应该优先考虑其作为灾难恢复方案的一部分。前面已经列出了事务复制作为灾难恢复方案的局限所在。这里我们把事务复制的优势和局限再总结下。事务复制的优势在于：

* 事务复制可以实现数据库对象级别的数据同步，你甚至可以筛选表中的部分数据来同步。也就是说主数据库可以只给灾备数据库提供其数据的子集。
* 灾备数据库可以访问，因此可用于报表等只读功能，分担主服务器的负载。
* 由于灾备数据库始终处于可以访问的状态，故障转移比较简单，仅需要重定向客户端即可。
* 支持多个冗余数据库副本，可用于灾难恢复。
* 使用推送订阅并将分发代理配置为连续发送的情况下，数据同步的速度快于日志传送。
* 对网络要求较低，可用于远距离的异地灾备方案。
* 主数据库和灾备数据库可以使用不同版本的SQL Server。

限制：

* 复制不提供自动侦测和解决故障的能力。主数据库发生故障后，你必须自己发现这个问题，然后手动执行故障转移操作才能解决问题。一旦执行了故障转移后，复制就被破坏了，需要重新建立。
* 不能为客户端提供一个透明的连接接口。当发生故障转移后，客户端需要做手动将客户端重定向到新的服务器。
* 无法像同步模式的数据库镜像那样提供完全零数据损失的保障。
* 内部实现机制复杂。一旦事务复制运行出现故障，排查的难度比较高。
* 复制的运行机制给主服务器带来较大的开销。同步的过程涉及到多个实例和代理程序，这带来了一定的同步延迟。
* 复制是在订阅服务器端建立一个ODBC或OLEDB的连接，然后执行保存在分发数据库中的命令来重做事务。而日志传送是使用还原日志的方法来重做事务。因此，日志传送在灾备服务器上重做事务的速度会快于事务复制。
* 复制对表的定义有要求。在已经上线的系统中使用复制功能可能会带来一些二次开发的工作。
* 与数据库镜像和日志传送不同，订阅数据库本身不限制写操作。订阅服务器上的数据存在被人错误改写的风险。而且很可能你完全没意识改写的行为的发生，在进行故障转移后就使用了错误的数据。

选择高可用和灾难恢复技术是个系统工程。没有一种技术是完美无缺的，都有自己最擅长的领域和不适合的领域。在选择技术之前，你首先要将明确自己的关键需求，有了需求才能找到最合适的技术。大多数情况下，鱼和熊掌不可兼得，完全同步和性能之间总是有一个要妥协的。这个时候就你对自己需求有个清楚优先级，哪些是不容妥协的，哪些是可以放宽的。经过这样一个筛选的过程就一定能找到符合自己要求的高可用和灾难恢复方案。

下面的表格（2-5）中横向罗列了各种技术之间的关键区别，希望能有助于你决定哪个技术更符合你的需求。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **功能** | **故障转移群集** | **日志传送** | **数据库镜像** | **事务复制** |
|
| 保护级别 | 实例级 | 数据库级 | 数据库级 | 数据库对象级 |
| 是否有数据损失 | / | 可能有少量数据损失 | 无数据损失（同步模式） | 可能有少量数据损失 |
| 自动故障转移 | 是 | 否 | 是（高可用操作模式） | 否 |
| 故障转移后是否可逆 | 是 | 否 | 是 | 否 |
| 对客户端是否透明 | 是，自动重连接到相同IP的另一个节点 | 否 | 是，自动重定向（需要驱动程序支持） | 否 |
| 停机时间 | 约等于SQL Server服务重启的时间+数据库恢复时间 | 较长 | 约等于数据库恢复时间 | 较长 |
| 多个备用数据副本 | 否 | 是 | 否 | 是 |
| 备用数据副本可读 | / | 是 | 否 | 是 |
| 能抵御用户误操作 | 否 | 是 | 否 | 否 |
| 能抵御磁盘故障 | 否 | 是 | 是 | 是 |
| 是否有特定硬件要求 | windows群集 | 无 | 要求有较好的磁盘和网络 | 无 |
| 对性能的影响 | 低 | 中 | 中 | 高 |
| 版本支持 | SQL Server 2000及以后 | SQL Server 2000及以后 | SQL Server 2005及以后 | SQL Server 2000及以后 |

表2-5各种技术之间的关键区别

### 2.6.2 高可用和灾难恢复技术的组合

当某个单一的技术不能完全满足你的需求时，你可以考虑将几种技术组合起来，以此实现更强大的高可用和灾难恢复的方案。技术组合的目的是扬长避短，将几种技术的优势组合起来，弥补单一技术的不足。不过，作为获得更强大的功能的代价，你也需要投入更多的硬件资源来实现这些方案。接下来我们就来看看一些比较典型的组合。

**故障转移群集+日志传送**

这个组合是在数据库镜像出现之前最常见的方案。故障转移群集提供了非常好的高可用性，但是对于数据损坏问题却非常脆弱。日志传送提供了灾难恢复的功能，但是在高可用性上很薄弱。它们两者的互补性非常强。

整个方案需要至少3台服务器。两台服务器用于实现故障转移群集，可以抵御SQL Server服务突然出现故障的情况。故障转移群集实例需要被配置为日志传送的主服务器，而第三台服务器作为日志传送的辅助服务器维护一个冗余的副本数据库。一旦故障转移群集的数据库出了问题，你有两个选择：

1. 将第三台服务器（日志传送的辅助服务器）上的数据库恢复使其上线。将应用程序重定向到这台服务器上。如果故障转移群集上数据库的问题是由于磁盘问题导致的，短期内很难解决，这个选择会比较合适。这个选择需要你在发生故障转移后将应用程序重定向到新的主服务器上。
2. 将第三台服务器上的数据库复制到故障转移群集实例上，替代损坏的数据库上线。这个方案适用于仅是数据库文件损坏而磁盘系统正常的情况。虽然复制数据库可能需要花费一定的时间，但是好处是不用重定向应用程序，而且问题解决后你的SQL Server依旧拥有群集提供的高可用性。

另外要注意的是，一旦做过日志传送的故障转移，日志传送就被破坏了。因此无论你的选择上面哪种办法，都要在完成灾难恢复后重新配置日志传送。

**故障转移群集+数据库镜像**

由于日志传送无法实现主数据库和辅助数据库的完全同步，因此故障转移群集+日志传送的组合在执行灾难恢复的时候可能会有一定的数据损失。为了弥补这个缺陷，从SQL Server 2005开始，你可以选择故障转移群集+数据库镜像这样的组合。数据库镜像替代日志传送，提供了数据损失更少（甚至是零损失）的灾难恢复能力；同时也可以提供更快的故障转移，提高了可用时间。

当镜像与群集一起使用时，你有三种部署的方案:

1. 两台服务器的方案。两个服务器配置成一个群集，在该群集上安装两个SQL Server实例，分别运行在两个节点，即所谓的活跃/活跃模式。而镜像的主体服务器在其中一个实例上运行，镜像服务器在另一个实例中运行。这个方案虽然很省硬件但是风险也很大。一旦某一个群集节点失败，就会造成镜像的主体服务器和镜像服务器运行在同一个节点上，该节点的硬件资源很可能捉襟见肘，严重影响主体服务器的性能。另外，主体服务器和镜像服务器作为一个Windows群集上的两个实例，意味它们其实是共用一个硬件存储器的。一旦这个存储器出了问题，那么主体服务器和镜像服务器上的两个数据副本就都不可用了，灾难恢复和高可用也无从谈起。
2. 三个服务器的方案。这个方案中，故障转移群集作为主体服务器，而镜像服务器驻留在一个单独的非群集的服务器上。
3. 四台服务器的方案。四台服务器分别组成两个故障转移群集，一个群集作为镜像的主体服务器实例，另一个群集作为镜像的镜像服务器实例。最佳情况下，这两个群集使用两个独立的存储器，避免存储区损坏导致整个方案失败的可能性。

数据库镜像的故障转移是可逆的，因此在解决故障后可以比较简单地将一切都恢复到出问题前的样子。另外，对于使用ADO.NET和SQL Native Client的应用，无论是使用哪种方案，无论是镜像还是群集发生了故障转移，都不需要重定向连接。

当决定如何在群集环境中配置数据库镜像时，所使用的镜像操作模式至关重要。

* 高可用操作模式

如果镜像使用这种操作模式，你可以选择上述第三种方案，因为该方案提供了最高的可用性。见证服务器可以安装在一个第三个群集上或一个非群集的服务器上。

假设现在有两个群集：ClusterA和ClusterB，还有另一台非群集的服务器ServerC作为见证服务器。ClusterA上的SQL Server实例作为主体服务器，而ClusterB上的SQL Server实例作为镜像服务器。如果ClusterA的活跃节点失败，将在几秒钟内先开始进行数据库镜像自动故障转移，同时群集也会开始启动故障转移。数据库镜像的故障转移完成后， ClusterB就从镜像服务器变为了主体服务器。在ClusterA群集故障转移完成（通常需要几分钟）之后，ClusterA的SQL Server就被切换到另一个节点，并且该实例变为了镜像服务器。这样配置的优点在于：

1. 镜像的故障转移速度比群集更快，因此停止时间更短。
2. 一旦在短时间内ClusterB也出了问题，而原来ClusterA的活跃节点上的故障还没来得及解决，SQL Server还能切换到ClusterA上新的活跃节点继续工作。如果是单机环境的数据库镜像就无法实现这点。

* 高保护操作模式

这种操作模式下，第二种方案和第三种方案都适用。你也不需要额外的服务器做见证服务器。

假设现在的环境是一个群集ClusterA和一个非群集的服务器ServerB。ClusterA作为主体服务器而ServerB作为镜像服务器。一旦ClusterA的活跃节点失败，并不会发生数据库镜像的故障转移，而是等到ClusterA的群集故障转移完成后，Cluster新的活跃节点继续维持主体服务器的角色并和ServerB维护镜像对话。

这种方案和单纯的故障转移群集相比，通过数据库镜像获得了额外的灾难恢复的能力（镜像数据库提供数据库副本）。但是就高可用性而言，该方案和单纯的故障转移群集差别不大。另外，如果群集的两个节点都失败了，整个数据库就不可用了，你需要手动的执行镜像的故障转移才能解决问题。

* 高性能操作模式

此操作模式的部署方案基本和高保护模式相同。区别是你可以考虑将主体服务器放置在群集的故障转移群集实例中，而将镜像服务器放置在远程位置的非群集服务器上，这样实现了异地灾备。

**故障转移群集+数据库镜像+日志传送**

故障转移群集+数据库镜像已经能提供非常高的可用性和不错的灾难恢复能力了。不过如果你觉得还是不放心，对于你的关键应用来说每一分钟的意外停机时间都会带来的巨大的损失，那你需要一个更加强大的方案。下面这个方案可能对你有帮助。

1. 你的主机房设在城市A，在主机房内，你选择使用故障转移群集+数据库镜像的第三套方案，即双群集和数据库镜像的方案，这样可以提供最好的高可用性。
2. 由于双群集都在一个机房里，当面临一些不可抗的因素，比如全市范围的停电，整个SQL Server系统还是会受到影响。为了进一步提高可用性和抵御灾难的能力，你可以为数据库镜像的主体数据库配置日志传送。主体服务器作为日志传送的主服务器，然后为其配置一个或多个远程的辅助服务器。你可以考虑配置两个远程的辅助服务器，一个设立在离城市A比较近的城市B，另一个设立在离城市A很远的城市C。这样你的数据在面对一些自然灾害时也有了一定的抵御能力。
3. 若要使日志传送在数据库镜像故障转移后仍能继续进行，还必须使用相同的配置将镜像服务器也配置为日志传送的主服务器（使用同样的备份共享目录，同样辅助服务器等）。由于镜像数据库平时处于还原状态，这 样可以防止备份作业备份镜像数据库中的日志。这确保了镜像数据库配置的日志传送不会影响主体数据库的日志传送。
4. 如果数据库镜像进行故障转移，以前的镜像数据库将作为主体数据库并联机。此时，该数据库也将作为日志传送主数据库进入活动状态。以前无法在该数据库中完成的备份作业也将开始执行备份并传送日志。相反，故障转移将使以前的主体数据库进入还原状态，同时该数据库上日志传送的备份作业也将停止备份日志。

这种方案可靠性最高，代价是硬件成本和维护复杂度也很高。

## 2.7 小结

本章中，我们介绍了SQL Server的几个“传统”的高可用性和灾难恢复技术。通过对故障转移群集，数据库镜像，日志传送和复制的了解，你可能会遗憾的发现，没有一种技术能既让你获得一个完全同步的数据副本，又能在远程获得一个允许一定同步滞后的数据副本，同时还能提供对客户端完全透明的特性。上述几种技术的组合虽然能够实现这样要求，但是部署复杂程度无疑是大了不少。

好消息是，随着SQL Server 2012的发布，一项全新的高可用和灾难恢复技术”AlwaysOn”横空出世了。AlwaysOn提供了以往需要几种技术结合才能实现的功能，因而广受期待。在下一章节里，你就能了解到AlwaysOn有哪些强大的特性。