

# 达梦技术丛书



# 前言

## 概述

本文档主要介绍 DM 共享存储集群的系统特性、基本概念、实现原理、主要功能，以及如何搭建 DM 共享存储集群并使用等。

## 读者对象


本文档主要适用于 DM 数据库的：

- 开发工程师
- 测试工程师
- 技术支持工程师
- 数据库管理员

## 通用约定

在本文档中可能出现下列标志，它们所代表的含义如下：

表 0.1 标志含义

标志	说明
 <b>警告：</b>	表示可能导致系统损坏、数据丢失或不可预知的结果。
 <b>注意：</b>	表示可能导致性能降低、服务不可用。
 <b>小窍门：</b>	可以帮助您解决某个问题或节省您的时间。
 <b>说明：</b>	表示正文的附加信息，是对正文的强调和补充。

在本文档中可能出现下列格式，它们所代表的含义如下：

表 0.2 格式含义

格式	说明
宋体	表示正文。
Courier new	表示代码或者屏幕显示内容。
粗体	表示命令行中的关键字（命令中保持不变、必须照输的部分）或者正文中强调的内容。标题、警告、注意、小窍门、说明等内容均采用粗体。
<>	语法符号中，表示一个语法对象。
::=	语法符号中，表示定义符，用来定义一个语法对象。定义符左边为语法对象，右边为相应的语法描述。
	语法符号中，表示或者符，限定的语法选项在实际语句中只能出现一个。
{ }	语法符号中，大括号内的语法选项在实际的语句中可以出现 0...N 次 (N 为大于 0 的自然数)，但是大括号本身不能出现在语句中。
[ ]	语法符号中，中括号内的语法选项在实际的语句中可以出现 0...1 次，但是中括号本身不能出现在语句中。
关键字	关键字在 DM_SQL 语言中具有特殊意义，在 SQL 语法描述中，关键字以大写形式出现。但在实际书写 SQL 语句时，关键字既可以大写也可以小写。

## 访问相关文档

如果您安装了 DM 数据库，可在安装目录的“\doc”子目录中找到 DM 数据库的各种手册与技术丛书。

您也可以通过访问我们的网站 [www.dameng.com](http://www.dameng.com) 阅读或下载 DM 的各种相关文档。

## 联系我们

如果您有任何疑问或是想了解达梦数据库的最新动态消息，请联系我们：

网址：[www.dameng.com](http://www.dameng.com)

技术服务电话：400-991-6599

技术服务邮箱：[dmtech@dameng.com](mailto:dmtech@dameng.com)

# 目录

<b>1</b>	<b>引言 .....</b>	<b>1</b>
<b>2</b>	<b>DMDSC 概述 .....</b>	<b>2</b>
2.1	系统特性 .....	4
2.2	基本概念 .....	5
2.3	使用说明 .....	8
<b>3</b>	<b>DMDSC 使用的环境 .....</b>	<b>10</b>
<b>4</b>	<b>DMDSC 实现原理 .....</b>	<b>11</b>
4.1	事务管理 .....	11
4.2	封锁管理 .....	13
4.3	问管理 .....	13
4.4	缓存交换 .....	14
4.5	重做日志管理 .....	20
4.6	回滚记录管理 .....	22
<b>5</b>	<b>DMCSS 介绍 .....</b>	<b>23</b>
5.1	启动命令 .....	23
5.2	心跳信息 .....	24
5.3	选举 DMCSS 控制节点 .....	24
5.4	选取监控对象控制节点 .....	24
5.5	启动流程管理 .....	24
5.6	状态检测 .....	24
5.7	故障处理 .....	25
5.8	节点重加入 .....	25
5.9	集群指令 .....	26
5.10	状态查看 .....	26
5.11	主普通节点显示信息差异 .....	28
5.12	配置 VIP .....	28
5.13	注意事项 .....	28
<b>6</b>	<b>DMASM 介绍 .....</b>	<b>30</b>
6.1	DMASM 概述 .....	30
6.2	DMASM 基本概念 .....	31
6.3	DMASM 原理 .....	32
6.4	DMASM 技术指标 .....	35
6.5	DMASM 使用说明 .....	36
6.6	DMASMCMD .....	37
6.7	DMASMSVR .....	40
6.8	DMASMAPI .....	41
6.9	DMASMTOOL .....	44

<b>7</b>	<b>DMDSC 启动、关闭流程 .....</b>	<b>50</b>
<b>8</b>	<b>DMDSC 故障处理 .....</b>	<b>51</b>
<b>9</b>	<b>DMDSC 节点重加入 .....</b>	<b>53</b>
<b>10</b>	<b>配置说明 .....</b>	<b>54</b>
10.1	DMDCR_CFG.INI .....	54
10.2	DMDCR.INI.....	58
10.3	DMINIT.INI .....	60
10.4	MAL 系统配置文件（DMMAL.INI、DMASVRMAL.INI） .....	63
10.5	DM.INI.....	64
<b>11</b>	<b>DMDSC 搭建 .....</b>	<b>67</b>
11.1	环境准备 .....	67
11.2	搭建 2 节点 DMDSC（DMASM） .....	67
11.3	搭建 2 节点 DMDSC（裸设备） .....	74
11.4	单节点搭建 DMDSC 测试环境 .....	79
<b>12</b>	<b>故障自动重连 .....</b>	<b>80</b>
12.1	配置服务名（DM_SVC.CONF） .....	80
12.2	体验故障自动重连 .....	81
<b>13</b>	<b>动态扩展节点 .....</b>	<b>83</b>
13.1	动态扩展节点流程 .....	83
<b>14</b>	<b>监控 DMDSC .....</b>	<b>89</b>
14.1	DMCSSM 监视器 .....	89
14.2	动态视图 .....	95
<b>15</b>	<b>备份还原 .....</b>	<b>114</b>
15.1	DMDSC 和单节点差异 .....	114
15.2	远程归档 .....	115
15.3	DMDSC 备份集 .....	117
15.4	DMDSC 备份还原实例 .....	117
15.5	使用说明 .....	119
<b>16</b>	<b>DMDSC 使用说明 .....</b>	<b>120</b>
16.1	统一组件版本 .....	120
16.2	提升 DMDSC 性能 .....	120
16.3	心跳说明 .....	121
16.4	重新格式化 DMASM .....	121
16.5	重新初始化 DMDSC 库 .....	122
16.6	内部网络异常 .....	122
16.7	创建 DBLINK .....	123
16.8	节点硬件故障，如何启动 DMDSC 集群 .....	125
16.9	MOUNT/OPEN 操作 .....	125

16.10	裸设备路径变化 .....	125
<b>17</b>	<b>附录 .....</b>	<b>127</b>
17.1	DMASMAPI 接口 .....	127
17.2	DMCSSM 接口 .....	153

# 1 引言

DM 共享存储数据库集群的英文全称 DM Data Shared Cluster，简称 DMDSC。

DM 共享存储数据库集群，允许多个数据库实例同时访问、操作同一数据库，具有高可用、高性能、负载均衡等特性。DMDSC 支持故障自动切换和故障自动重加入，某一个数据库实例故障后，不会导致数据库服务无法提供。

本文主要介绍 DMDSC 集群的功能、概念、实现原理，并举例说明搭建过程和管理方法。

本手册可以帮助用户：

- 了解 DMDSC/DMCSS/DMA SM 等集群相关概念
- 了解 DMA SM 分布式文件系统、DMDSC 集群，以及基于 DMA SM 的 DMDSC 集群配置过程和应用

## 2 DMDSC 概述

DMDSC 集群是一个多实例、单数据库的系统。多个数据库实例可以同时访问、修改同一个数据库的数据。用户可以登录集群中的任意一个数据库实例，获得完整的数据库服务。数据文件、控制文件在集群系统中只有一份，不论有几个节点，这些节点都平等地使用这些文件。各个节点有自己独立的联机日志和归档日志。这些文件就保存在共享存储上。

DMDSC 集群得以实现的重要基础就是共享存储。DM 支持的共享存储有两种：裸设备和 DMASM。这两种存储的区别在于后者在前者的基础上，部署并使用了 DMASM 文件系统。为了方便对裸设备上的磁盘或文件进行管理，推荐用户使用后者。

DMDSC 集群主要由数据库和数据库实例、共享存储、本地存储、通信网络、以及集群控制软件 DMCSS 组成。下面以部署了 DMASM 的 DMDSC 集群为例，展示 DMDSC 集群系统结构（图 2.1）。



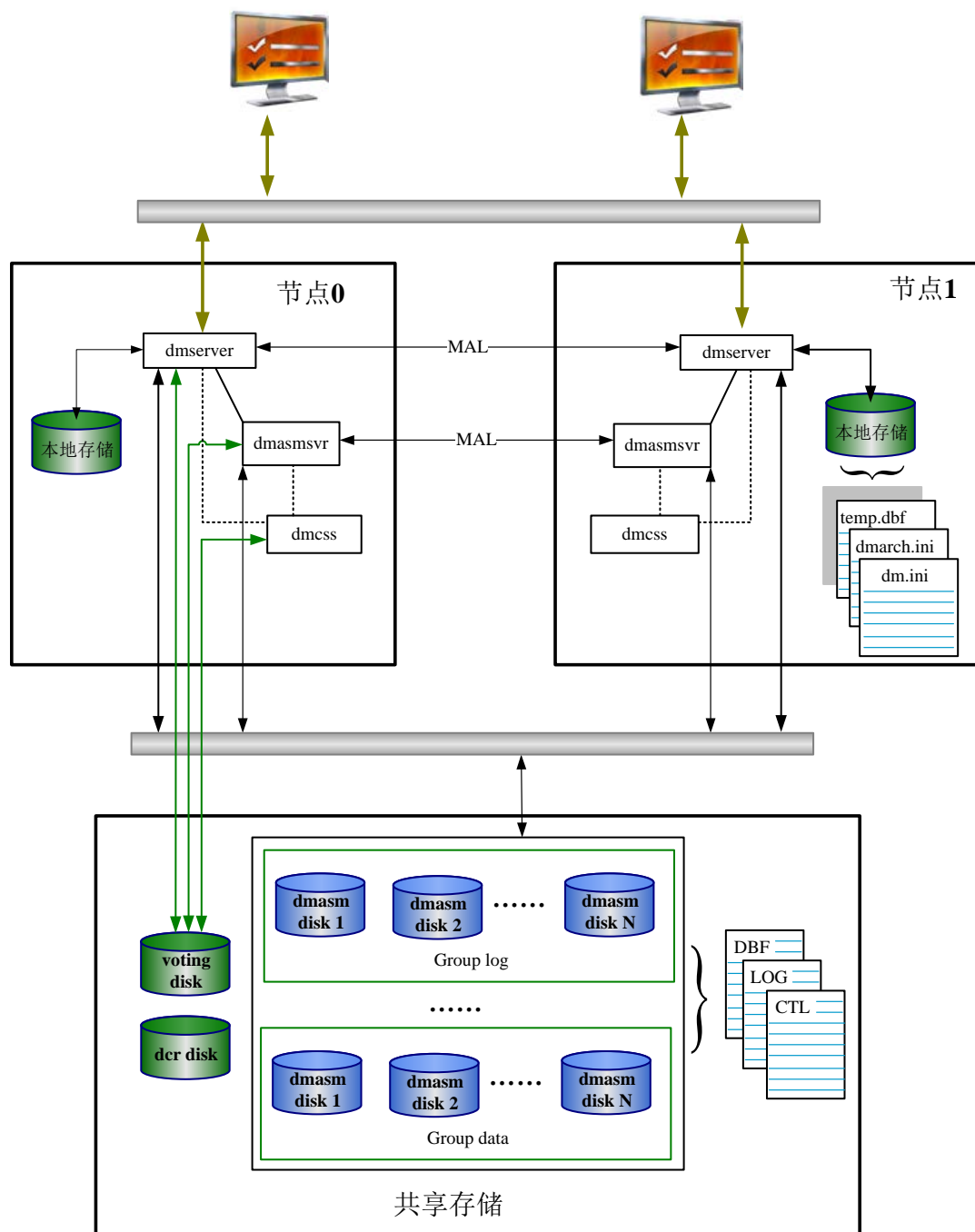


图 2.1 DMDSC 系统结构图

### 数据库和数据库实例

数据库 (Database) 是一个文件集合 (包括数据文件、临时文件、重做日志文件和控制文件等), 保存在物理磁盘或文件系统中。

数据库实例就是一组操作系统进程 (或者是一个多线程的进程) 以及一些内存。通过数据库实例, 可以操作数据库, 一般情况下, 我们访问、修改数据库都是通过数据库实例来完成的。

### 共享存储

DMDSC集群中，为了实现多个实例同时访问、修改数据，要求将数据文件、控制文件、日志文件保存在共享存储上。DMDSC 支持使用裸设备或 DMASM 文件系统作为共享存储。

配置 DMDSC 集群需要的 DCR、Voting disk 也必须保存在共享存储上（目前仅支持裸设备存放 DCR 和 Voting disk）。

### 本地存储

DMDSC集群中，本地存储用来保存配置文件（记录数据库实例配置信息的 dm.ini、dmarch.ini、dmmal.ini），本地归档日志、远程归档日志。

### 通信网络

DMDSC集群中，网络分为内部网络和公共网络两个部分。实际应用中一般还存在服务器到共享存储的网络。内部网络用于数据库实例之间交换信息和数据，MAL 链路使用的就是内部网络。公共网络用于对外提供数据库服务，用户使用公共网络地址登录 DMDSC 集群，访问数据库。

### 集群控制

集群控制是集群系统的重要组成部分。DMCSS 就是一款集群控制软件，专门负责监控集群中各个节点的运行状态。DMCSS 主要功能包括：管理集群的启动和关闭，控制节点故障处理，以及管理节点重加入流程。

## 2.1 系统特性

DMDSC 的主要特点包括：

- 高可用性 只要集群中有一个活动节点，就能正常提供数据库服务。
- 高吞吐量 多个节点同时提供数据库服务，有效提升集群的整体事务处理能力。
- 负载均衡 用户的连接请求被平均分配到集群中的各个节点，确保各个节点的负载大致平衡。

### 2.1.1 高可用性

DMDSC 集群提供了一种达梦数据库高可用解决方案。当出现系统故障、硬件故障、或人为操作失误时，DMCSS 检测故障、并自动将故障节点踢出集群，保证数据库服务的正常提供。

故障节点的用户连接会自动切换到活动节点，这些连接上的未提交事务将被回滚，已提

交事务不受影响；活动节点的用户连接不受影响，正在执行的操作将被挂起一段时间，在故障处理完成后，继续执行。当 DMCSS 检测到故障节点恢复时，自动启动节点重加入流程，将恢复的故障节点重新加入 DMDSC 集群，将集群恢复到正常的运行状态。因此，通过部署 DMDSC 集群，可以在一定程度上避免由软、硬件故障引起的非计划停机，减少这些意外给客户带来的损失。

与同样使用共享存储的双机热备系统相比，DMDSC 具有更快的故障处理速度。双机热备系统故障切换时，需要完整重做 Redo 日志，所有数据需要重新从磁盘加载；而 DMDSC 故障处理时，只需要重做故障节点的 Redo 日志，并且大部分数据页已经包含在处理节点的 Buffer 缓冲区中，不需要重新从磁盘加载。

### 2.1.2 高吞吐量

DMDSC 集群中包含多个数据库实例，数据库实例访问独立的处理器、内存，数据库实例之间通过缓存交换技术提升共享数据的访问速度，每个数据库实例都可以接收并处理用户的各种数据库请求。

与单节点数据库管理系统相比，DMDSC 集群可以充分利用多台物理机器的处理能力，支撑更多的用户连接请求，提供更高的吞吐量。与双机热备系统相比，DMDSC 集群不存在始终保持备用状态的节点，不会造成硬件资源的浪费。

### 2.1.3 负载均衡

用户通过配置 DM 数据库连接服务名来访问 DMDSC 集群，可以实现节点间的自动负载均衡，用户的数据库连接请求会被自动、平均地分配到 DMDSC 集群中的各个节点。并且连接服务名支持 JDBC、DPI、ODBC、DCI、.Net Provider 等各种数据库接口。

## 2.2 基本概念

### 集群 (Cluster)

是由两个或多个节点（服务器）构成的一种松散耦合的计算机节点集合，这个集合在整个网络中表现为一个单一的系统，并通过单一接口进行使用和管理。大多数模式下，集群中的所有计算机都拥有一个相同的名称，集群内任意一个系统都可以被所有的网络用户使用。

每个集群节点都是运行其自己进程的独立服务器，因此每个节点都有自己的运算能力。这些进程间彼此通信进行协调，协同起来向用户提供应用程序、系统资源和数据以及计算能力。本书中涉及到的集群有三种：DMDSC 集群，DMCSS 集群和 DMASM 集群。

### **DMDSC 集群**

DMDSC 集群由若干数据库实例（Instance）组成，这些实例间通过网络（MAL 链路）连接，通过一个特殊的软件（DMCSS，集群同步服务）的协助，共同操作一个数据库。从外部用户视角来看，他们看到的只是一个数据库。数据文件、控制文件等文件在集群中只有一份，所有节点平等地使用这些数据文件。这份数据一般放在共享存储上，每个服务器通过光纤连接到共享存储上。

DMDSC 支持使用裸设备或 DMASM 文件系统存放共享数据库文件。为了方便对裸设备上的磁盘或文件进行管理，DM 推荐使用 DMASM 文件系统。

### **裸设备（Raw Device）**

一种没有经过格式化，不被 Unix/Linux 通过文件系统来读取的特殊字符设备，允许直接访问磁盘而不经操作系统的高速缓存和缓冲器。因为使用裸设备避免了经过操作系统这一层，数据直接从磁盘到数据库服务器进行传输，所以使用裸设备对于读写频繁的数据库应用来说，可以有效提高数据库系统的性能。但是裸设备的使用有很多限制，比如 Linux 主机的每个磁盘最多能划分 16 个分区，去掉一个扩展分区后，可用的只有 15 个；每个分区只支持一个裸设备；每个裸设备只能对应一个文件、裸设备一经创建大小就固定、不能动态调整等。所以实际使用时可能会出现数据库文件空间不够或者空间浪费的情况，需要根据应用实际情况提前分配好裸设备大小。

### **DMASM（DM Auto Storage Manager）**

是一个专用的分布式文件系统。支持多个节点同时访问、修改数据文件，并减少直接使用裸设备存在的诸多限制。DMASM 文件系统把指定的裸设备打包管理，使用 DMASM 文件系统可以方便地创建、删除、扩展、截断文件，不用担心空间不足（空间不足可以通过增加磁盘扩展空间）或空间浪费；不用考虑文件个数限制；可以方便查看空间使用情况；可以在线通过增加裸设备的方式扩展总体使用空间。

DMASM 不是一个通用的文件系统，只能通过 dmasmapi 接口访问。理论上通过 dmasmapi 接口可以存放任何文件，但在 DMDSC 集群中，一般只建议将需要在节点间共享访问的文件存在 DMASM 文件中，如数据文件、联机 Redo 日志文件、控制文件等。归档 Redo 日志文件、备份集文件也可以考虑保存到 DMASM 文件系统中，避免还原、恢复等操作时节

点间的文件拷贝，简化备份、还原操作。其他的一些本地配置文件比如 `dm.ini` 等保存在本地磁盘中。

DMDSC 集群中若配置 DMASM，则要求 DMASM 站点数和 DMCSS 站点数一致，且只能存在一个 DMCSS 组和一个 DMASM 组。这些 DMASM 站点共同构成了一个 DMASM 集群。

### **DMCSS (DM Cluster Synchronization Services)**

DMCSS 是 DM 集群同步服务的简称，是 DMDSC 集群应用的基础，使用 DMDSC 集群或者 DMASM 集群都必须配置 DMCSS。DMCSS 负责集群环境中节点的启动、故障处理、节点重加入等操作。

每个集群节点都需要有一个 DMCSS 服务。这些 DMCSS 服务又共同构成一个 DMCSS 集群。单节点应用时，可以不配置 CSS。

### **DMCSSM (DM Cluster Synchronization Services Monitor)**

DMCSSM (DM Cluster Synchronization Services Monitor) 是 DM 集群监视器的简称。DMCSSM 与 DMCSS 相互通信，获取并监控整个集群系统的状态信息。DMCSSM 还提供了一系列的命令来管理、维护集群。

同一个集群中，允许最多同时启动 10 个监视器，一般建议将监视器放在独立的第三方机器上。

### **DCR (DM Clusterware Registry)**

DCR 是 DM 集群注册表的简称，用于存储、维护集群配置的详细信息，整个集群环境共享 DCR 配置信息，包括 DMDSC、DMASM、DMCSS 资源，包括实例名、监听端口、集群中故障节点信息等。DCR 必须存储在集群中所有节点都可以访问到的共享存储中，并且只支持裸设备。在一个集群环境中只能配置一个 DCR 磁盘。

### **表决磁盘 (Voting Disk)**

表决磁盘记录了集群成员信息，DM 集群通过 Voting Disk 进行心跳检测，确定集群中节点的状态，判断节点是否出现故障。当集群中出现网络故障时，使用 Voting Disk 来确定哪些 DMDSC 节点应该被踢出集群。表决磁盘还用来传递命令，在集群的不同状态（启动、节点故障、节点重加入等）DMCSS 通过 Voting Disk 传递控制命令，通知节点执行相应命令。Voting Disk 必须存储在集群中所有节点都可以访问到的共享存储中，并且只支持裸设备。在一个集群环境中只能配置一个表决磁盘。

集群中各实例启动时，通过访问 DCR 获取集群配置信息。被监控实例从 Voting Disk 读取监控命令，并向 Voting Disk 写入命令响应以及自身心跳信息；DMCSS 也向 Voting

Disk 写入自己的心跳信息，并从 Voting Disk 访问各被监控节点的运行情况，并将监控命令写入 Voting Disk，供被监控实例访问执行。

### HeartBeat（心跳机制）

DMCSS 的心跳机制是通过 Voting Disk 的 Disk Heartbeat。这种机制有最大时延，只有超过最大时延，才认为监测对象故障。

### MAL 链路

MAL 系统是达梦数据库基于 TCP 协议实现的一种内部通信机制，具有可靠、灵活、高效的特性。使用 DMASM 文件系统的 DMDSC 集群中存在两套 MAL 系统，DMASM 服务器之间配置一套 MAL 系统，dmserver 服务器之间配置一套 MAL 系统。一旦 MAL 链路出现异常，DMCSS 会进行裁定，并从集群中踢出一个节点，保证集群环境正常运行。

### 共享内存

共享内存是一种快速、高效的进程间通信手段。所谓共享内存，就是同一块物理内存被映射到多个进程的地址空间，进程 A 可以即时看到进程 B 对共享内存的修改，反之亦然。DMASM 服务器进程和 DMASM 客户端进程之间通过共享内存方式共享 DMASM 文件到实际磁盘的映射关系。

### VIP

VIP(虚拟 IP 地址)，是一个不与特定计算机或者计算机中的网络接口相连的 IP 地址。数据包被发送到这个 VIP 地址，但是所有的数据还是经过真实的网络接口。在集群环境中，应用通过 VIP 连接数据库服务器，实例故障后，把实例配置的 VIP 设置到其他活动节点（叫做 IP 漂移），这样应用可以不用修改配置，继续访问数据库服务。

## 2.3 使用说明

目前 DMDSC 在功能上与单机版 DM 相比存在一定限制，暂不支持下列功能：


1. 支持定时器，但只有控制节点上配置的定时器生效。只支持脱机配置定时器，不支持联机配置
2. 支持作业，但只有控制节点上支持执行作业
3. 不支持 HUGE 表
4. 不支持外部表
5. 不支持堆表


6. 不支持类型别名相关操作
7. 不支持 table 级别的 space limit 功能
8. 不支持全文索引、词库相关操作
9. 不支持安全版本
10. 不支持 DBMS\_ALERT、DBMS\_LOCK 包
11. 不支持数据复制
12. 不能与 MPP 集群混合使用
13. 不支持为表空间文件指定 mirror\_path
14. 不支持闪回查询，不允许打开闪回功能


## 3 DMDSC 使用的环境

部署 DMDSC 集群所用到的硬件和软件环境。


### ■ 硬件环境


 主机两台。用于部署数据库实例 dmserver、DMCSS、DMASMSVR。内存大小要求：至少 2GB。

 共享存储。两台机器可以同时访问到的，可以划分为裸设备的磁盘。

 网卡。每台主机至少准备 2 块网卡。提供内部网络和外部网络服务。

### ■ 软件环境

 操作系统。Linux、Unix、Windows 等。

 达梦数据库软件。安装好 DM 数据库软件之后，将拥有配置和管理 DMDSC 所需的所有软件：dmserver、dminit、dmasmcmd、dmasmsvr、dmasmtool、dmcss、dmcssm 等。这些软件位于安装目录/dmdbms/bin 中。



## 4 DMDSC 实现原理

DMDSC 是一个共享存储的数据库集群系统。多个数据库实例同时访问、修改同一个数据库，因此必然带来了全局并发问题。DMDSC 集群基于单节点数据库管理系统之上，改造了 Buffer 缓冲区、事务系统、封锁系统和日志系统等，来适应共享存储集群节点间的全局并发访问控制要求。同时，引入缓存交换技术，提升数据在节点间的传递效率。

### 4.1 事务管理

多版本并发控制（MVCC）可以确保数据库的读操作与写操作不会相互阻塞，大幅度提升数据库的并发度以及使用体验，大多数主流商用数据库管理系统都实现了 MVCC。DM 的多版本并发控制实现策略是：数据页中只保留物理记录的最新版本数据，通过回滚记录维护数据的历史版本，通过活动事务视图（V\$DSC\_TRX\_VIEW）判断事务可见性，确定获取哪一个版本的数据。

每一条物理记录中包含了两个字段：TID 和 RPTR。TID 保存修改记录的事务号，RPTR 保存回滚段中上一个版本回滚记录的物理地址。插入、删除和更新物理记录时，RPTR 指向操作生成的回滚记录的物理地址。

回滚记录与物理记录一样，也包含了 TID 和 RPTR 这两个字段。TID 保存产生回滚记录时物理记录上的 TID 值（也就是上一个版本的事务号），RPTR 保存回滚段中上一个版本回滚记录的物理地址。

每一条记录（物理记录或回滚记录）代表一个版本。如下图所示：

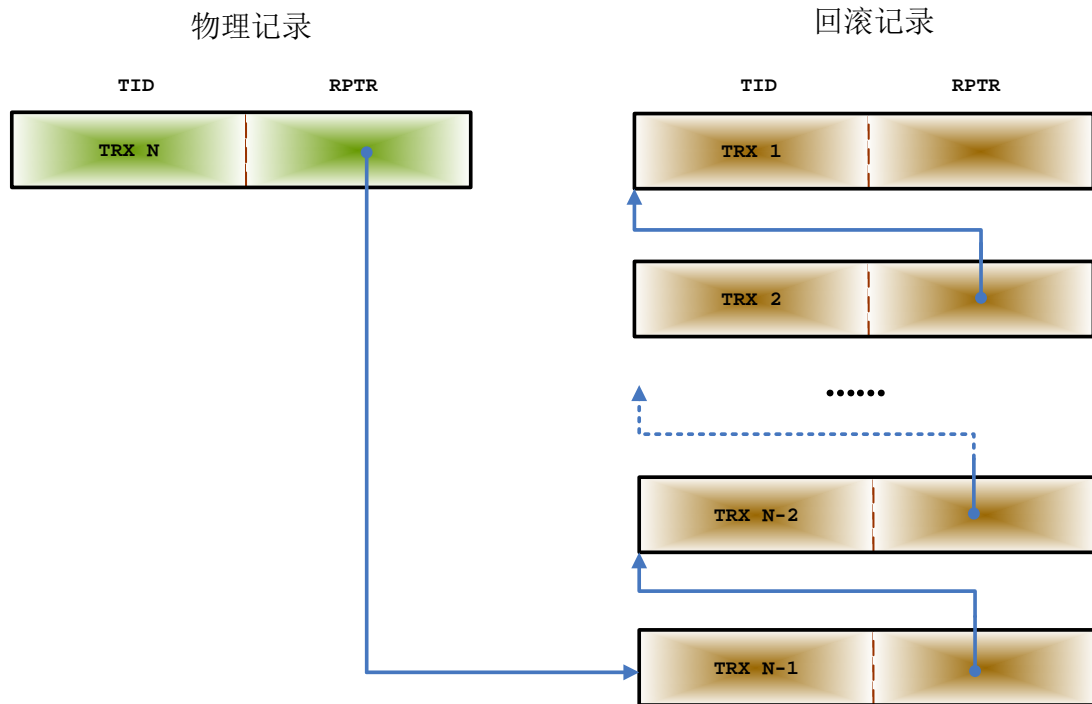


图 4.1 各版本之间的关系

如何找到对当前事务可见的特定版本数据，进行可见性判断，是 DM 实现多版本并发控制的关键。根据事务隔离级别的不同，在事务启动时（串行化），或者语句执行时（读提交），收集这一时刻所有活动事务，并记录系统中即将产生的事务号 NEXT\_TID。DM 多版本并发控制可见性原则：

1. 物理记录 TID 等于当前事务号，说明是本事务修改的物理记录，物理记录可见
2. 物理记录 TID 不在活动事务表中，并且 TID 小于 NEXT\_TID，物理记录可见
3. 物理记录的 TID 包含在活动事务表中，或者  $TID \geq NEXT\_TID$ ，物理记录不可见

为了在 DMDSC 集群中实现与单节点相同的多版本并发控制（MVCC）策略，每个事务需要知道所有节点当前活动的事务信息，根据事务隔离级的不同，在事务启动时（串行化），或者语句执行时（读提交），收集这一时刻所有节点上的活动事务，以及系统中即将产生的事务号 NEXT\_TID，记录到事务的活动事务视图中。DMDSC 集群将事务信息全局化，由控制节点统一管理集群中所有节点的全局事务视图（Global Transaction View，简称 GTV）；与之对应的是每个节点维护一个本地事务视图（Local Transaction View，简称 LTV），在事务启动、收集活动事务信息时通知全局事务视图，并获取相应的信息。

## 4.2 封锁管理

数据库管理系统一般采用行锁进行并发访问控制，避免多个用户同时修改相同数据；通过表锁、字典锁控制 DDL 和 DML 操作的并发访问，保证对象定义的有效性和数据访问的正确性。DM 则采用了独特的封锁机制，使用 TID 锁和对象锁进行并发访问控制，有效减少封锁冲突、提升系统并发性能。

TID 锁以事务号为封锁对象，每个事务启动时，自动以独占（X）方式对当前事务号进行封锁，由于事务号是全局唯一的，因此这把 TID 锁不存在冲突，总是可以封锁成功。同时，在 4.1 事务管理中，介绍了物理记录上包含一个 TID 字段，记录了修改数据的事务号。执行 INSERT、DELETE、UPDATE 操作修改物理记录时，设置事务号到 TID 字段的动作，就相当于隐式地对物理记录上了一把 X 方式的 TID 锁。因此，通过事务启动时创建的 TID 锁，以及写入物理记录的 TID 值，DM 中所有修改物理记录的操作都不再需要额外的行锁，避免了大量行锁对系统资源的消耗，有效减少封锁冲突。特别是在 DMDSC 集群中，需要进行全局封锁，封锁的代价比单节点更高，通过 TID 锁可以有效减少封锁引发的性能损失。

对象锁则通过对象 ID 进行封锁，将对数据字典的封锁和表锁合并为对象锁，以达到减少封锁冲突、提升系统并发性能的目的。

与事务管理类似，DMDSC 集群将封锁管理拆分为全局封锁服务（Global Locking Services，简称 GLS）和本地封锁服务（Local Locking Services，简称 LLS）两部分。整个系统中，只有控制节点拥有一个 GLS。控制节点的 GLS 统一处理集群中所有节点的封锁请求、维护全局封锁信息、进行死锁检测，确保事务并发访问的正确性。每个节点都有一个 LLS。各节点的 LLS 负责与 GLS 协调、通讯，完成事务的封锁请求，DMDSC 集群中所有封锁请求都需要通过 LLS 向 GLS 发起，并在获得 GLS 授权后，才能进行后续操作。

## 4.3 闕管理

闕（Latch）是数据库管理系统的一种内部数据结构，通常用来协调、管理 Buffer 缓冲区、字典缓存和数据库文件等资源的并发访问。与锁（Lock）在事务生命周期中一直保持不同，闕（Latch）通常只保持极短的一段时间，比如修改 Buffer 中数据页内容后，马上会释放。闕（Latch）的封锁类型也比较简单，就是共享（Share）和独占（Exclusive）两种类型。

为了适用 DMDSC 集群,我们同样将闩划分为全局闩服务(Global Latch Services)和本地闩服务(Local Latch Services)两个部分。但是,为了与全局封锁服务 GLS 和本地封锁服务 LLS 的名字简称区分开来,我们以使用最为频繁的 Buffer 来命名全局闩服务。因此,全局闩服务也称为全局缓冲区服务(Global Buffer Services),简称 GBS;本地闩服务也称为本地缓冲区服务(Local Buffer Services),简称 LBS。

整个系统中,每一个节点上都部署一个 GBS 和一个 LBS。GBS 服务协调节点间的 Latch 封锁请求、以及 Latch 权限回收。GBS 与 GTV/GLS 由控制节点统一管理不同,GBS 不是集中式管理,而是由 DMDSC 集群中的所有节点共同管理,Buffer 对象会根据数据页号(Page No)对数据页进行划分,分给某一个节点的 GBS 服务处理。LBS 服务与 LLS/LTV 一样,部署在每一个节点,LBS 服务根据用户请求,向 GBS 发起 Latch 封锁,或者根据 GBS 请求,回收本地的 Latch 封锁。

为了避免两个、或多个节点同时修改同一个数据页,导致数据损坏,或者数据页修改过程中,别的节点读取到无效内容,DMDSC 集群中数据页的封锁流程产生一定变化,与单节点相比,增加了全局 Latch 封锁、释放两个步骤。并且,在获取全局 Latch 授权后,仍然需要进行正常的本地 Latch 封锁,避免节点内访问冲突。

## 4.4 缓存交换

根据目前的硬件发展状况来看,网络的传输速度比磁盘的读、写速度更快,因此,DMDSC 集群引入了缓存交换(Buffer Swap)技术,节点间的数据页尽可能通过网络传递,避免通过磁盘的写入、再读出方式在节点间传递数据,从而减少数据库的 IO 等待时间,提升系统的响应速度。

缓存交换的实现基础是 GBS/LBS 服务,在 GBS/LBS 中维护了 Buffer 数据页的相关信息。包括:1. 闩的封锁权限(LATCH);2. 哪些站点访问过此数据页(Access MAP);3. 最新数据保存在哪一个节点(Fresh EP)中;4. 以及最新数据页的 LSN 值(Fresh LSN)等信息。这些信息作为 LBS 封锁、GBS 授权和 GBS 权限回收请求的附加信息进行传递,因此并不会带来额外的通讯开销。

下面,以两节点 DMDSC 集群(EP0/EP1)访问数据页 P1 为例子。初始页 P1 位于共享存储上,P1 的 GBS 控制结构位于节点 EP1 上。初始页 P1 还没有被任何一个节点访问过,初始页 P1 的 LSN 为 10000。通过几种常见场景分析,逐步深入,解析缓存交换的原理。

### ■ 场景 1

节点 EP0 访问数据页 P1。

1. 节点 EP0 的本地 LBS 向 EP1 的 GBS 请求数据页 P1 的 S LATCH 权限
2. 节点 EP1 的 GBS 修改 P1 控制结构，记录访问节点 EP0 的封锁模式为 S LATCH（数据分布节点为 EP0），并响应 EP0 的 LBS 请求
3. 节点 EP0 的 LBS 获得 GBS 授权后，记录获得的授权模式是 S\_LATCH，P1 数据不在其他节点的 Buffer 中，发起本地 IO 请求，从磁盘读取数据。IO 完成后，修改 LBS 控制结构，记录数据页上的 LSN 信息

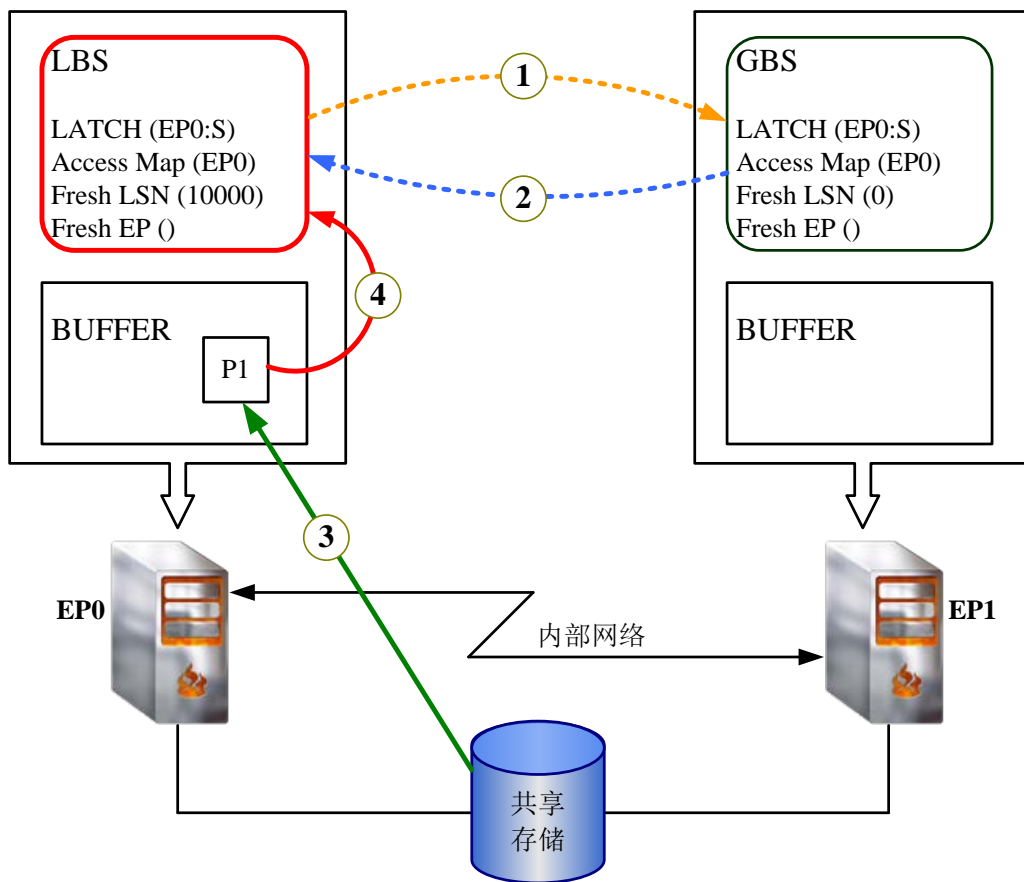


图 4.2 本地 IO

### ■ 场景 2

节点 EP1 访问数据页 P1。

1. 节点 EP1 本地 LBS 向 EP1 的 GBS 请求数据页 P1 的 S LATCH 权限
2. 节点 EP1 的 GBS 修改控制结构，记录访问节点 EP1 的封锁模式为 S LATCH（数据分布节点为 EP0/EP1），并响应 EP1 的 LBS 请求

3. 节点 EP1 的 LBS 获得 GBS 授权后，记录获得的授权模式是 S LATCH，根据数据分布情况，EP1 向 EP0 发起 P1 的读请求，通过内部网络从 EP0 获取数据，而不是重新从磁盘读取 P1 数据

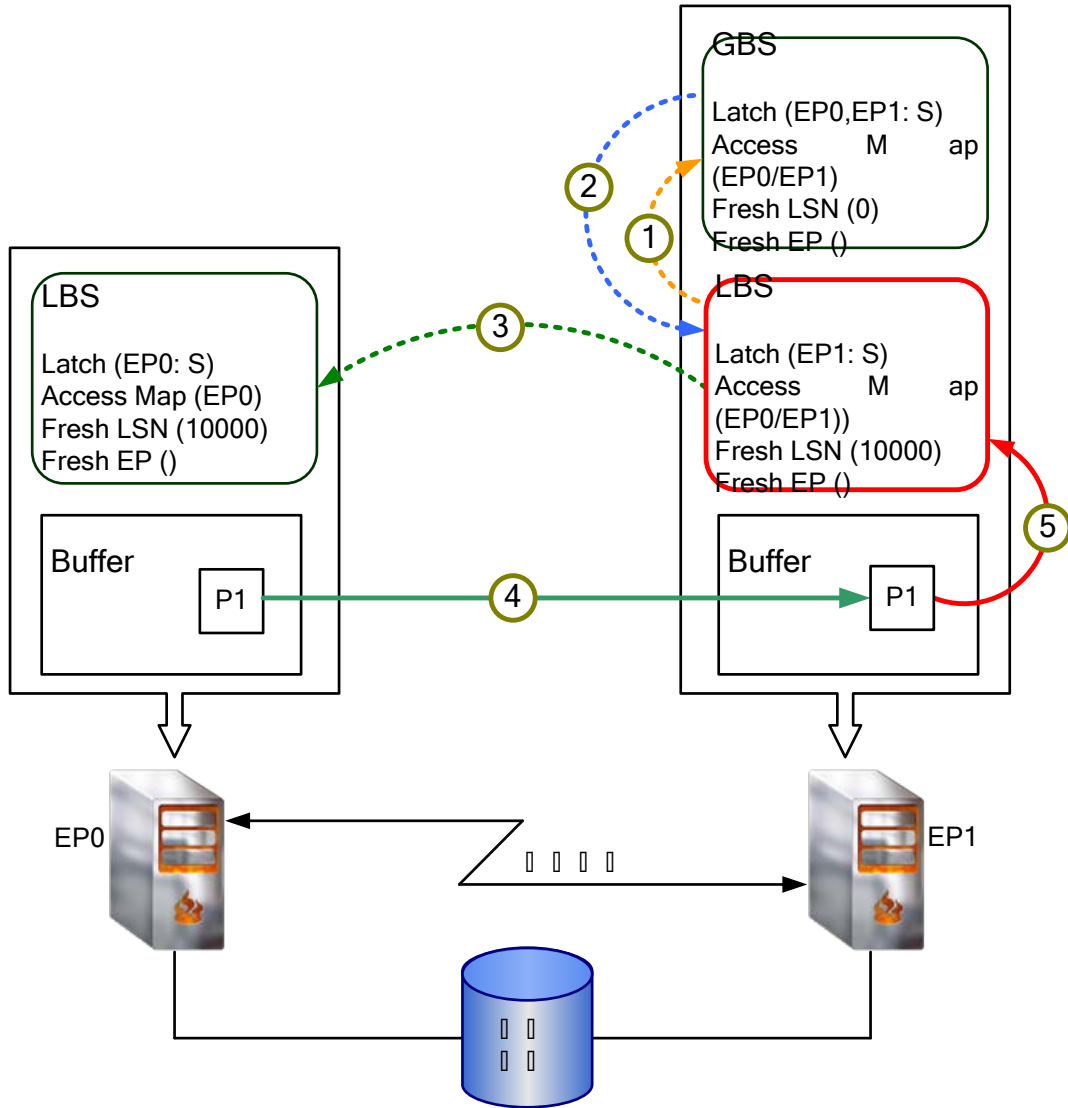


图 4.3 远程 IO

### ■ 场景 3

节点 EP0 修改数据页 P1。

1. 节点 EP0 本地 LBS 向 EP1 的 GBS 请求数据页 P1 的 X LATCH 权限（附加 LSN 信息）
2. 节点 EP1 的 GBS 修改控制结构的 LSN 值，从 EP1 的 LBS 回收 P1 的权限
3. 修改访问节点 EP0 的封锁模式为 S + X LATCH，并响应 EP0 的 LBS 请求
4. 节点 EP0 的 LBS 获得 GBS 授权后，记录获得的授权模式是 S + X LATCH

## 5. 节点 EP0 修改数据页 P1, LSN 修改为 11000

这个过程中, 只有全局 Latch 请求, 数据页并没有在节点间传递。

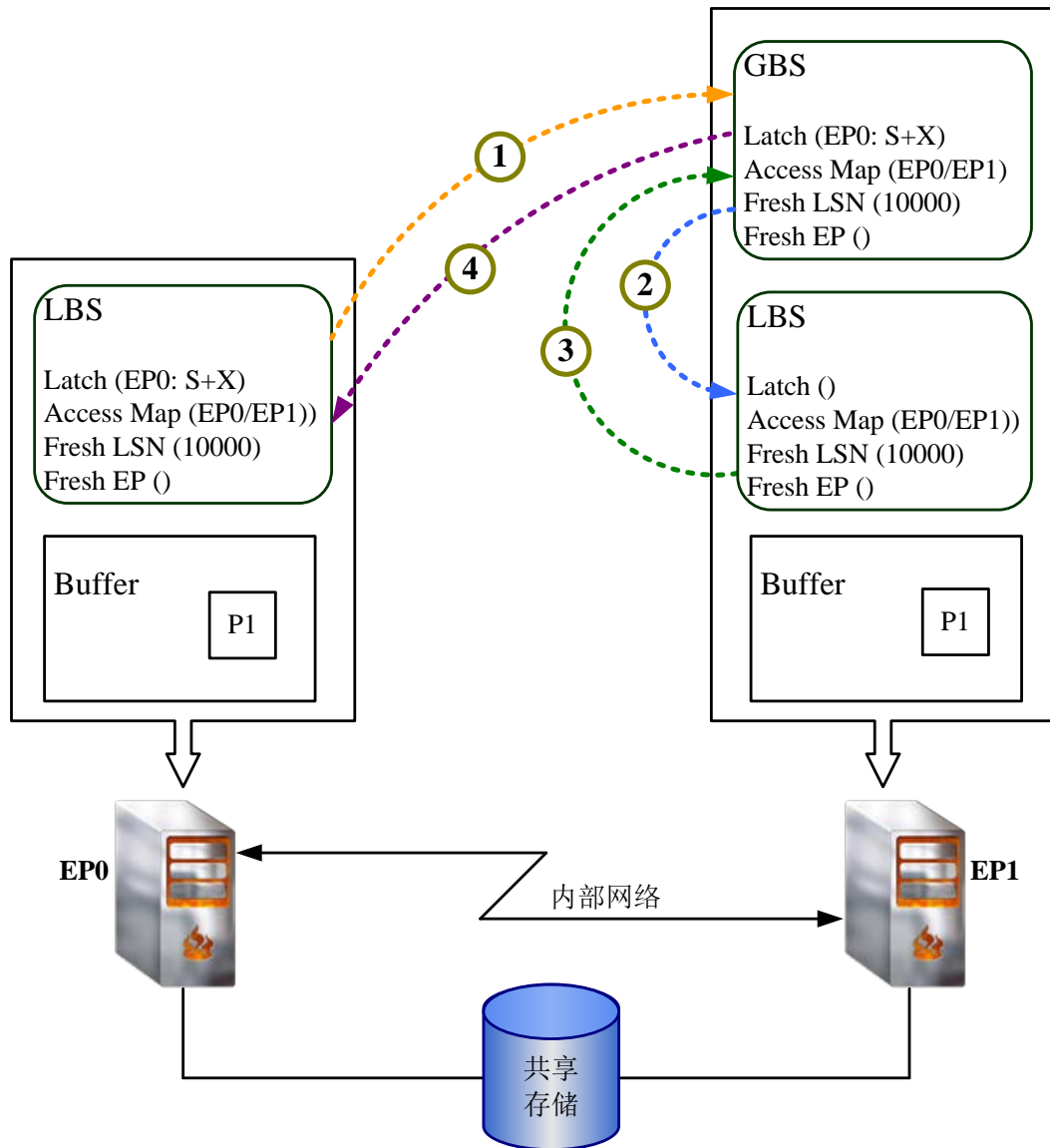


图 4.4 GBS 管理

修改之后, 数据页 P1 的 LSN 修改为 11000。如下所示:

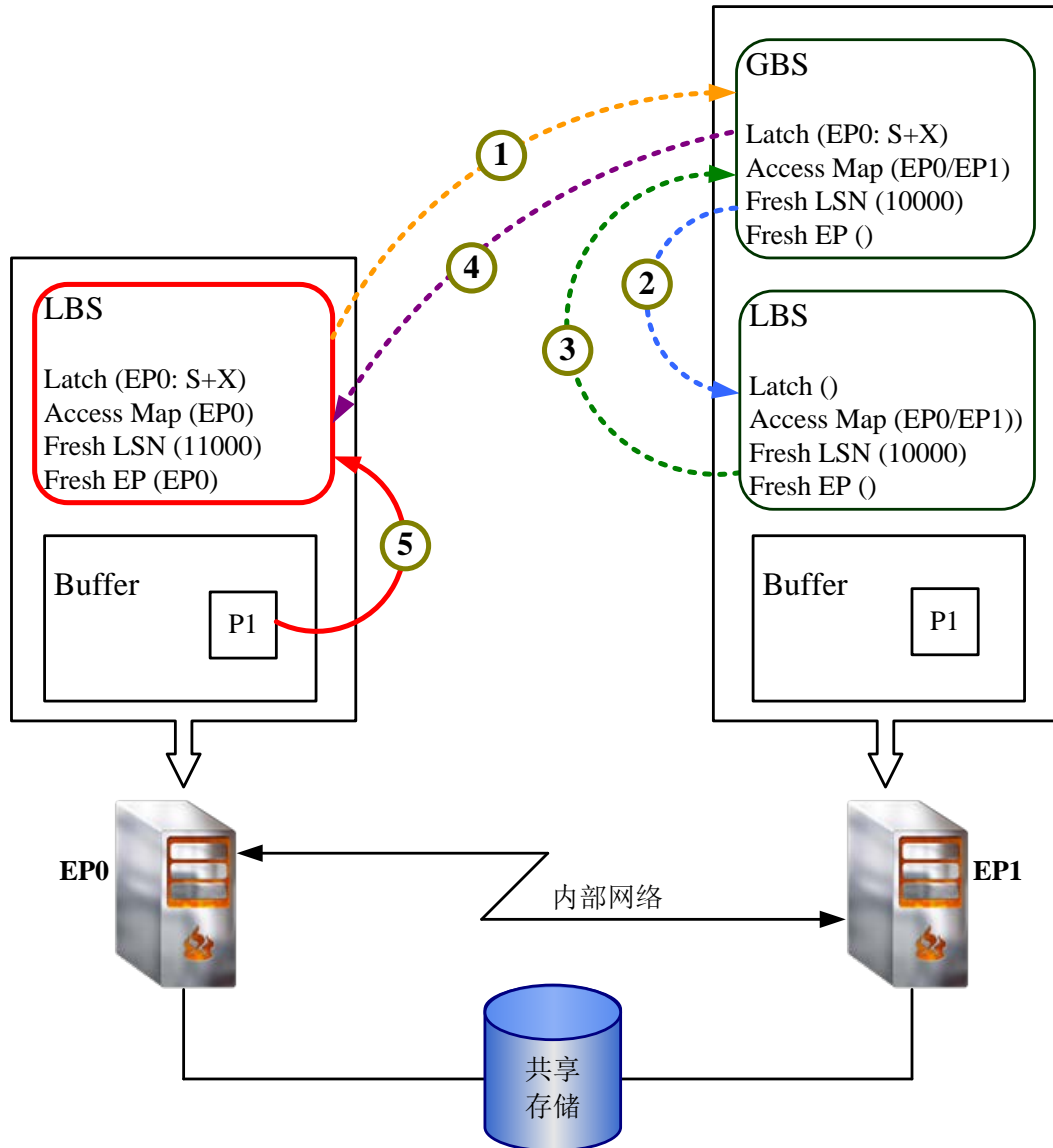


图 4.5 数据修改

#### ■ 场景 4

节点 EP1 修改数据页 P1。

1. 节点 EP1 本地 LBS 向 EP1 的 GBS 请求数据页 P1 的 X LATCH 权限
2. 节点 EP1 的 GBS 发现 P1 被 EP0 以 S + X 方式封锁，向 EP0 发起回收 P1 权限的请求
3. 节点 EP0 释放 P1 的全局 LATCH，响应 GBS，并且在响应消息中附加了最新的 PAGE LSN 值
4. 节点 EP1 的 GBS 收到 EP0 的响应后，修改 GBS 控制结构，记录最新数据保存在 EP0，最新的 LSN 值信息，记录 EP0 获得的授权模式是 S + X LATCH（此时，数据分布节点仍



然是 EP0/EP1)，并授权 EP1 的 LBS

5. 节点 EP1 的 LBS 收到授权信息后，记录获得的授权模式是 S + X LATCH，并根据数据分布情况，向节点 EP0 发起数据页 P1 的读请求

6. 节点 EP1 修改数据页 P1，LSN 修改为 12000

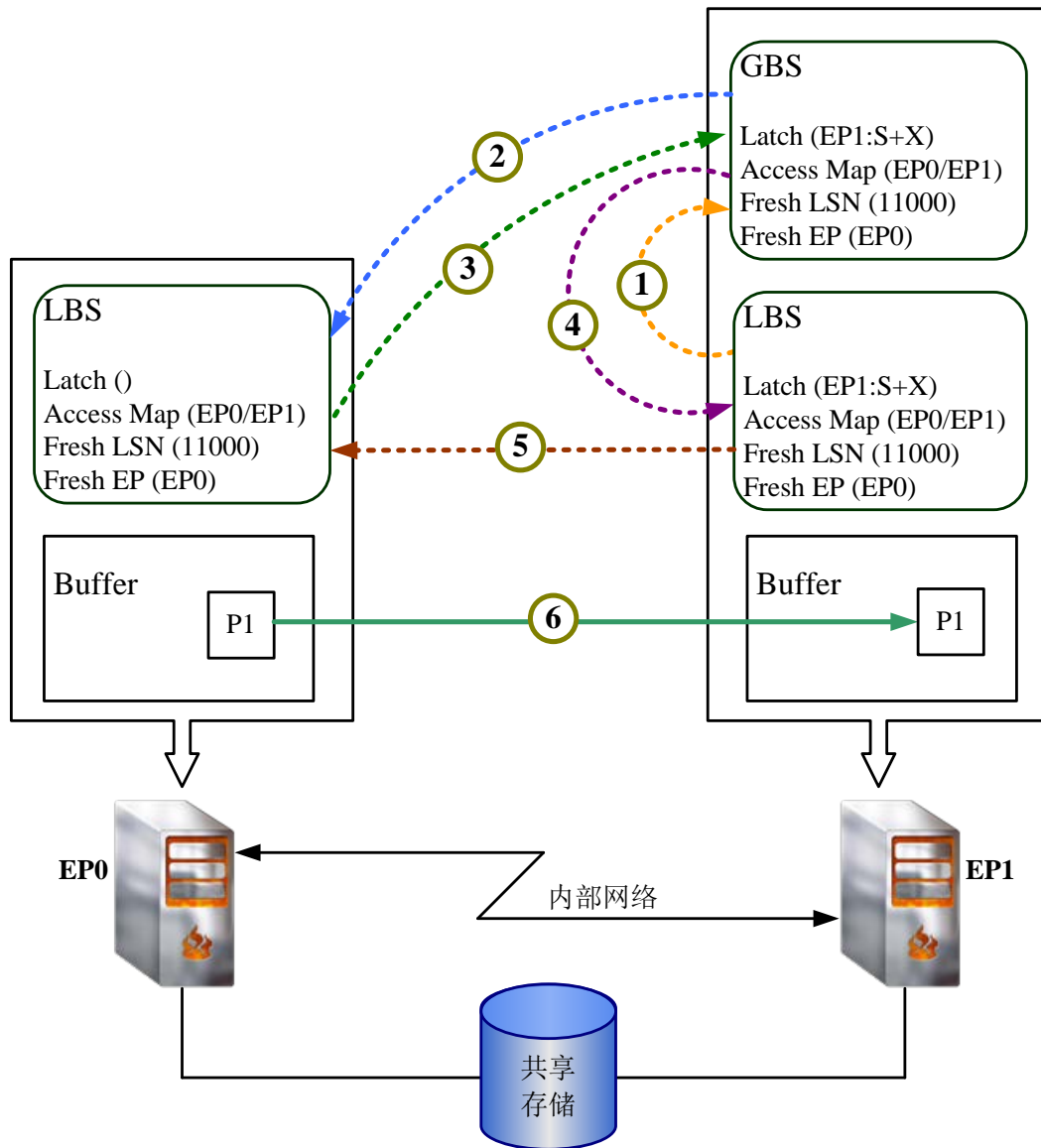


图 4.6 GBS 管理

修改之后，数据页 P1 的 LSN 修改为 12000。如下所示：

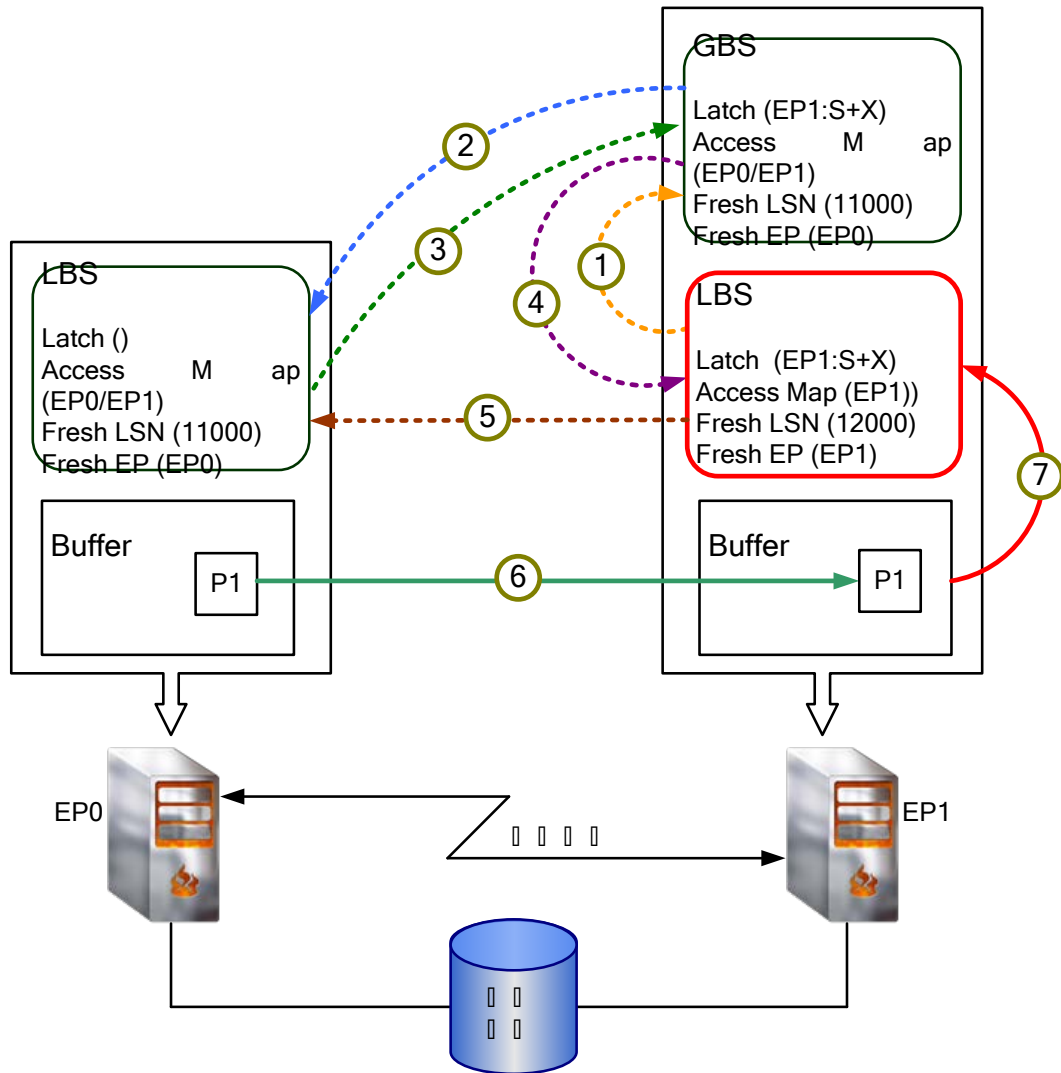


图 4.7 数据修改

这个过程中，数据页 P1 的最新数据从 EP0 传递到了 EP1，但并没有产生磁盘 IO。

## 4.5 重做日志管理

Redo 日志包含了所有物理数据页的修改内容，Insert/delete/update 等 DML 操作、Create Table 等 DDL 操作，最终都会转化为对物理数据页的修改，这些修改都会反映到 Redo 日志中。一般说来一条 SQL 语句，在系统内部会转化为多个相互独立的物理事务来完成，物理事务提交时产生 Redo 日志，并最终写入联机 Redo 日志文件中。

一个物理事务包含一个或者多个 Redo 记录 (Redo Record, 简称 RREC)，每条 Redo 记录都对应一个修改物理数据页的动作。根据记录内容的不同，RREC 可以分为两类：物理 RREC 和逻辑 RREC。物理 RREC 记录的是数据页的变化情况，内容包括：操作类型、修改

数据页地址、页内偏移、数据页上的修改内容，如果是变长类型的 Redo 记录，在 RREC 记录头之后还会有一个两字节的长度信息。逻辑 RREC 记录的是一些数据库逻辑操作步骤，主要包括：事务启动、事务提交、事务回滚、字典封锁、事务封锁、B 树封锁、字典淘汰等，一般只在配置为 Primary 模式时才产生逻辑 RREC。

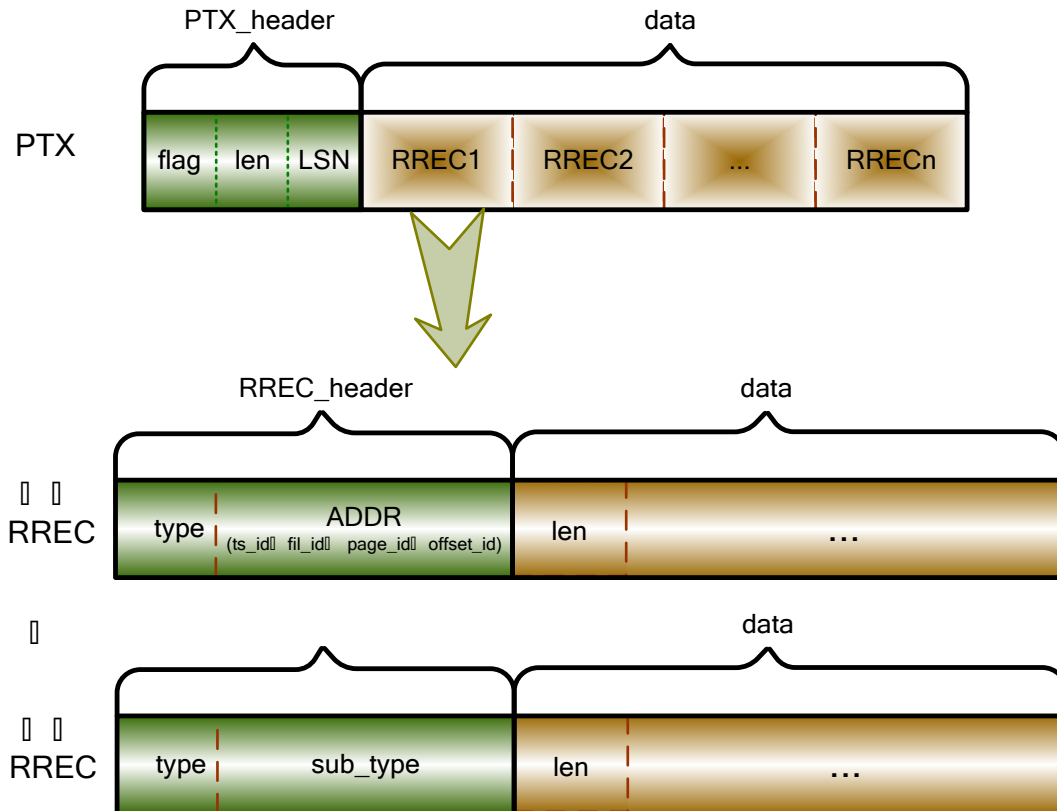


图 4.8 PTX/RREC 结构图

DMDS 集群中，各个节点拥有独立的日志文件，Redo 日志的 LSN 值也是顺序递增的，Redo 日志只会写入当前数据库实例的联机日志文件，与集群系统中的其他数据库实例没有关系。考虑到所有节点都可以修改数据，同一个数据页可能由不同节点先后修改，为了体现修改的先后顺序，确保故障恢复时能够按照操作的顺序将数据正确恢复。DMDS 集群要求对同一个数据页的修改，产生的 LSN 值是全局递增的，各个节点对同一数据页的修改在日志系统中是严格有序的。但是，针对不同数据页的修改并不要求 LSN 是全局递增的，也就是说只有多个节点修改相同数据页时，才会产生全局 LSN 同步问题。并且 LSN 全局同步，是在缓存交换时附带完成的，并不会增加系统的额外开销。

与单节点系统相比，DMDS 的日志系统存在以下差异：

1. 本地 Redo 日志系统中，LSN 值保证是递增的，后提交物理事务的 LSN 值一定更大；但顺序提交的两个物理事务产生的 LSN 值，不能保证一定是连续的

2. 全局 Redo 日志系统中，LSN 值不再严格保证唯一性。不同节点可能存在 LSN 值相等的重做日志记录
3. 故障重启时，控制节点需要重做所有节点的 Redo 日志，重做过程中会根据 LSN 排序，从小到大依次重做
4. 联机 Redo 日志文件需要保存在共享存储中

## 4.6 回滚记录管理

DMDSC 集群的多版本并发控制（MVCC）实现策略是，通过回滚记录获取数据的历史版本，通过活动事务视图判断事务可见性、确定获取指定版本数据。因此，回滚记录也必须进行全局维护，有可能在节点间进行传递。与单节点一样，DMDSC 集群中只有一个回滚表空间，回滚记录保存在回滚页中，回滚页与保存用户记录的数据页一样，由 Buffer 系统管理，并通过缓存交换机制实现全局数据共享。

为了减少并发冲突，提高系统性能，DMDSC 集群中为每个节点分配了一个单独的回滚段（Segment），虽然这些回滚段位于同一个回滚表空间中，但是各个节点的回滚页申请、释放，并不会产生全局冲突。

与重做日志一样，DMDSC 集群故障重启时，控制节点会扫描所有节点的回滚段，收集未提交事务进行回滚，收集已提交事务进行 Purge 操作。

## 5 DMCSS 介绍

DMCSS (Dameng Cluster Synchronization Services) 达梦集群同步服务, 使用 DMASM 集群或 DMDSC 集群都必须配置 DMCSS 服务。在 DMASM 集群或 DMDSC 集群中, 每个节点都需要配置一个 DMCSS 服务。这些 DMCSS 服务自身也构成一个集群, DMCSS 集群中负责监控、管理整个 DMASM 集群和 DMDSC 集群的节点称为控制节点(control node), 其他 DMCSS 节点称为普通节点(normal node)。DMCSS 普通节点不参与 DMASM 集群和 DMDSC 集群管理, 当 DMCSS 控制节点故障时, 会从活动的普通节点中重新选取一个 DMCSS 控制节点。

DMCSS 工作的基本原理是: 在 Voting disk 中, 为每个被监控对象 (dmasmsvr、dmserver、DMCSS) 分配一片独立的存储区域, 被监控对象定时向 Voting Disk 写入信息 (包括时间戳、状态、命令、以及命令执行结果等); DMCSS 控制节点定时从 Voting Disk 读取信息, 检查被监控对象的状态变化, 启动相应的处理流程; 被监控对象只会被动的接收 DMCSS 控制节点命令, 执行并响应。

DMCSS 主要功能包括: 写入心跳信息、选举 DMCSS 控制节点、选取 DMASM/DMDSC 控制节点、管理被监控对象的启动流程、集群状态监控、节点故障处理、节点重加入等, DMCSS 还可以接收并执行 DMCSSM 指令。

### 5.1 启动命令

```
./dmcss HELP
```

格式: dmcss.exe KEYWORD=value

例如: dmcss.exe DCR\_INI=/home/data/DAMENG/dmdcr.ini

关键字说明 (默认)

DCR_INI	dmdcr.ini 路径
-NOCONSOLE	以服务方式启动
HELP	打印帮助信息

## 5.2 心跳信息

DMCSS 实例启动后，每间隔 1 秒向 Voting Disk 指定区域写入心跳信息（包括自身的状态、时间戳等），表示 DMCSS 节点处于活动状态。

## 5.3 选举 DMCSS 控制节点

DMCSS 启动后向 Voting Disk 写入信息，并读取其他 DMCSS 节点的信息，如果 DMCSS 集群中还没有活动的控制节点，则选举 DMCSS 控制节点。DMCSS 选举的原则有两条：

1. 先启动的 DMCSS 作为控制节点
2. DMCSS 同时启动，则选择节点号小的节点为控制节点

## 5.4 选取监控对象控制节点

DMCSS 控制节点启动后，会为基于 DMASM/裸设备的 DMDSC 集群指定控制节点。DMCSS 选取监控对象控制节点的原则有两条：

1. 只有一个活动节点，则设置活动节点为控制节点。
2. 存在多个活动节点，则选择节点号小的节点为控制节点。

## 5.5 启动流程管理

DMASM 和 DMDSC 集群中的实例启动后，一直处于 waiting 状态，等待 DMCSS 的启动命令。DMCSS 控制节点在选取监控对象控制节点后，通知控制节点启动，在控制节点启动完成后，再依次通知其他普通节点启动。

## 5.6 状态检测

DMCSS 维护集群状态，随着节点活动信息的变化，集群状态也会产生变化，DMCSS 控制节点会通知被监控节点执行不同命令，来控制节点启动、故障处理、故障重加入等操作。

DMCSS 控制节点每秒从 Voting Disk 读取被监控对象的心跳信息。一旦被监控对象的时间戳在 DCR\_GRP\_DSKCHK\_CNT 秒内没有变化，则认为被监控对象出现异常。

DMCSS 普通节点定时读取 DMCSS 控制节点的心跳信息，监控 DMCSS 运行状态。

## 5.7 故障处理

DMCSS 控制节点检测到实例故障后，首先向故障实例的 Voting disk 区域写入 Kill 命令（所有实例一旦发现 Kill 命令，无条件自杀），避免故障实例仍然处于活动状态，引发脑裂，然后启动故障处理流程，不同类型实例的故障处理流程存在一些差异。

### ■ DMCSS 控制节点故障处理流程

1. 活动节点重新选举 DMCSS 控制节点
2. 新的 DMCSS 控制节点通知出现 DMCSS 故障节点对应的 dmasmsvr、dmserver 强制退出

### ■ DMASMSVR 实例故障处理流程

1. 挂起工作线程
2. 更新 DCR 的节点故障节点信息
3. 通知故障节点对应 dmserver 强制退出
4. dmasmsvr 进行故障恢复
5. 恢复工作线程

### ■ dmserver 实例故障处理流程

1. 更新 DCR 故障节点信息
2. 重新选取一个控制节点
3. 通知 dmserver 控制节点启动故障处理流程（参考 DMDSC 故障处理）
4. 等待 dmserver 故障处理结束

## 5.8 节点重加入

如果检测到故障节点恢复，DMCSS 会通知控制节点启动节点重加入流程。

### ■ 数据库实例重加入

1. 挂起工作线程
2. 修改节点的状态
3. 执行恢复操作
4. 重新进入 STARTUP 状态，准备启动
5. OPEN 重加入的节点

6. 重启工作线程
7. 执行 OPEN 数据库实例的操作
- DMASM 实例重加入
  1. 挂起工作线程
  2. 修改节点的状态
  3. 执行恢复操作
  4. 重新进入 STARTUP 状态，准备启动
  5. OPEN 重加入的节点
  6. 重启工作线程

## 5.9 集群指令

DMCSS 控制节点通过一系列的集群指令，控制被监控对象的启动、故障处理、状态切换等。DMCSS 控制节点向目标对象的 Voting disk 指令区写入命令，通知目标对象执行相应命令，并等待执行响应。每条指令的功能都比较单一，比如修改状态、设置控制节点、执行一条 SQL 等，复杂的集群流程控制就是由这些简单的指令组合起来完成的。

## 5.10 状态查看

在 DMCSS 控制台输入 show 命令可以看到所监控的集群状态，如下所示。

**group[ ]**行显示的内容为

name:	集群名称
seq:	集群编号
type:	集群类型[CSS/ASM/DB]
control_node:	集群内控制节点

**ep** 行显示的内容为:

inst_name:	节点实例名
seqno:	节点编号
port:	实例对外提供服务的端口号
mode:	模式[控制/普通]
sys_status:	实例系统状态[MOUNT/OPEN 等]



vtd\_status: 实例的集群状态[WORKING/SHUTDOWN/SYSHALT 等]  
 is\_ok: 实例在集群内是否正常, ERROR 的节点暂时从集群内剔除  
 active: 实例是否活动  
 guid: 实例的 guid 值  
 ts: 实例的时间戳

```

===== group[name = GRP_CSS, seq = 0, type = CSS, Control Node = 0]
=====

ep:      inst_name  seqno  port  mode      sys_status  vtd_status  is_ok
active   guid              ts
          CSS1              0          8060      Control      Node
OPEN      WORKING      OK          TRUE      670744319      670748263
          CSS2              1          8061      Normal      Node
OPEN      WORKING      OK          TRUE      670744832      670748770
          CSS3              2          8062      Normal      Node
OPEN      WORKING      OK          TRUE      670745332      670749266

self css info:

[ASM1] auto restart = FALSE

[RAC1] auto restart = FALSE

===== group[name = GRP_ASM, seq = 1, type = ASM, Control Node = 0]
=====

ep:      inst_name  seqno  port  mode      sys_status  vtd_status  is_ok
active   guid              ts
          ASM1              0          8063      Control      Node
OPEN      WORKING      OK          TRUE      670746265      670750197
          ASM2              1          8064      Normal      Node
OPEN      WORKING      OK          TRUE      670746960      670750891
          ASM3              2          8065      Normal      Node
OPEN      WORKING      OK          TRUE      670747502      670751432
  
```

```

=====
=====

n_ok_ep = 3

(0, 0)

(1, 1)

(2, 2)

sta = OPEN, sub_sta = STARTUP

break ep = NULL

recover ep = NULL

crash process over flag is TRUE

```

## 5.11 控制节点和普通节点显示信息差异

DMCSS 控制节点会显示 DMCSS、DMASM、DMDSC 三个集群的信息，而普通节点只会显示 DMCSS 集群的信息。

## 5.12 配置 VIP

如果为集群环境 DB 节点配置了 VIP，则 CSS 会在不同时机配置 VIP（假设节点为 EP1，EP2）：

- 1 DB 启动前，各节点 CSS 会配置对应的 VIP。
- 2 节点 DB1 故障，则故障节点对应的 CSS1 会取消配置的 VIP1，活动节点 EP2 对应的 CSS2 会配置故障节点的 VIP1，此时活动节点 EP2 会有两个 VIP（VIP2，已经故障节点的 VIP1）。原来连接 DB1 的应用，重新连接时会连接到 DB2。
- 3 故障节点 DB1 重加入，则 EP2 对应的 CSS2 会取消 VIP1，之后 EP1 对应的 CSS1 会重新配置 VIP1，这样 VIP 配置信息就恢复最开始的状态了。

## 5.13 注意事项

1. 如果节点 A 的 DMCSS 退出或者故障，活动 DMCSS 会给节点 A 上所监控的 asmsvr

和 dmserver 发送 halt 命令，确保节点 A 上的 asmsvr 和 dmserver 自动退出。

2. 配置 VIP 信息需要 root 权限，所以要确保 dmcss 以 root 权限启动。

## 6 DMASM 介绍

DMASM (DM Auto Storage Manager) 是一个专用的分布式文件系统，使用 DMASM 自动存储管理方案，可以帮助用户更加便捷地管理 DMDSC 集群的数据库文件。DMASM 的主要部件包括：提供存储服务的裸设备、dmasmsvr 服务器、dmasmapi 接口、初始化工具 dmasmcmd 和管理工具 dmasmtool 等。

### 6.1 DMASM 概述

DMDSC 集群可以直接使用裸设备作为共享存储，存放数据库文件。但是，由于裸设备存在的一些功能限制，造成 DMDSC 集群在使用、维护上并不是那么灵活、方便。裸设备的使用限制如下：

1. 不支持动态扩展文件大小；在创建数据文件时，就必须指定文件大小，并且文件无法动态扩展
2. 数据文件必须占用整个裸设备盘，造成空间浪费
3. 不支持类 linux 的文件操作命令，使用不方便
4. 操作系统支持最大裸设备数目较小，无法创建足够的数据库文件

为了克服裸设备的这些使用限制，DM 专门设计了一个分布式文件系统 DMASM，来管理裸设备的磁盘和文件。DMASM 提供了基本的数据文件访问接口，可以有效降低 DMDSC 共享存储的维护难度，DMASM 提供的主要功能包括：

#### 1. 分布式管理

支持多台机器并发访问 DMASM 磁盘和文件，提供全局并发控制。

#### 2. 磁盘组管理

支持创建和删除磁盘组，将裸设备格式化为 DMASM 格式，并由 dmasmsvr 统一管理；一个磁盘组可以包含一个或者多个 DMASM 磁盘；磁盘组支持在线增加 DMASM 磁盘，实现动态存储扩展。

#### 3. 文件管理

支持创建、删除、截断文件等功能；支持创建目录；支持动态扩展文件；文件可以存放

在一个磁盘组的多个磁盘中，文件大小不再受限于单个磁盘大小。

#### 4. 完善、高效的访问接口

通过 `dmasmapi` 可以获得各种文件管理功能。

#### 5. 通用功能的管理工具

`dmasmtool` 提供一套类Linux的文件操作命令用于管理DMASM文件，降低用户学习、使用DMASM文件系统的难度。

## 6.2 DMASM 基本概念

DMASM 的实现主要参考了达梦数据库文件系统，因此，一些概念和实现原理与达梦数据库基本类似，熟悉达梦数据库的用户，就会更加容易理解 DMASM。

表 6.1 DMASM/达梦文件系统概念对照表

DMASM	达梦文件系统
磁盘组 (disk group)	表空间 (tablespace)
DMASM 磁盘 (disk)	数据文件 (datafile)
DMASM 文件 (file)	段 (segment)
簇 (extent)	簇 (extent)
AU (allocate unit)	页 (page)
描述 AU (desc AU)	描述页 (desc page)
Inode AU (inode AU)	Inode 页 (inode page)

### DMASM 磁盘

DMASM 磁盘是指经过 `dmasmcmd` 工具格式化，可以被 `dmassvr` 识别的物理磁盘。DMASM 磁盘是组成磁盘组的基本单位，一个裸设备只能格式化为一个 DMASM 磁盘，不支持分割使用。

### 磁盘组

磁盘组由一个或多个 DMASM 磁盘组成，是存储 DMASM 文件的载体；一块 DMASM 磁盘只能属于一个磁盘组。DMASM 支持动态添加 DMASM 磁盘。DMDSC 集群中，一般建议将日志文件和数据文件保存到不同的磁盘组中。

### DMASM 文件

在 DMASM 磁盘组上创建的文件，称之为 DMASM 文件。一个 DMASM 文件只能保存

在一个磁盘组中，但一个 DMASM 文件的数据可以物理存放在同一磁盘组的多个 DMASM 磁盘中。DMDSC 集群中，需要多个节点共享访问的数据库文件、日志文件、控制文件等，一般会创建为 DMASM 文件。

### 簇 (extent)

簇是 DMASM 文件的最小分配单位，一个簇由物理上连续的一组 AU 构成。簇的大小为 4，也就是说一个 DMASM 文件至少占用 4 个 AU，也就是 4M 的物理存储空间。

### AU (Allocate Unit)

DMASM 存储管理的最小单位，AU 的大小为 1M。DMASM 以 AU 为单位将磁盘划分为若干逻辑单元，DMASM 文件也是由一系列 AU 组成。根据 AU 的不同用途，系统内部定义了一系列 AU 类型，包括：desc AU、inode AU、redo AU、和 data AU。

## 6.3 DMASM 原理

为了帮助用户更好的理解、使用 DMASM，本节从 DMASM 磁盘与文件管理、DMASM redo 日志、簇映射表等方面介绍 DMASM 原理。

### 6.3.1 DMASM 磁盘与文件管理

DMASM 文件系统将物理磁盘格式化后，变成可识别、可管理的 DMASM 磁盘，再通过 DMASM 磁盘组将一个或者多个 DMASM 磁盘整合成一个整体提供文件服务。

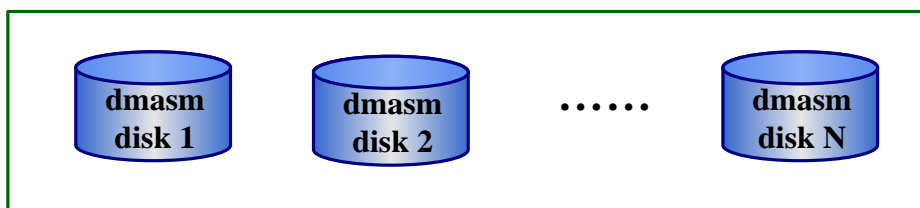


图 6.1 DMASM 磁盘组结构

DMASM 磁盘格式化以后，会逻辑划分为若干簇 (extent)，簇是管理 DMASM 磁盘的基本单位，DMASM 文件的最小分配单位也是簇。这些逻辑划分的簇根据其用途可以分为描述簇、inode 簇和数据簇。

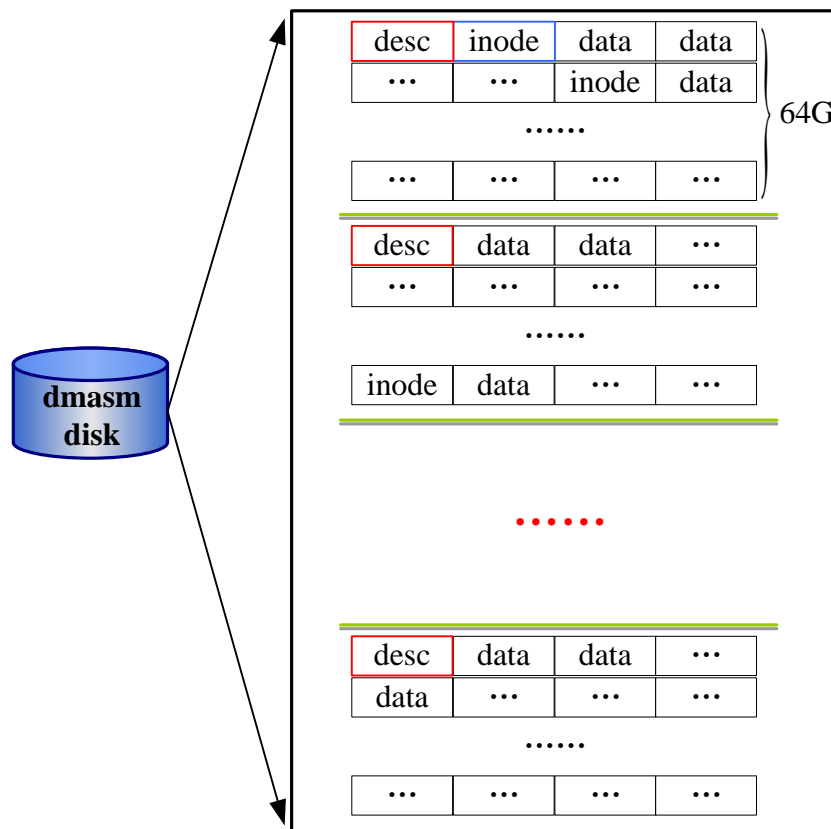


图 6.2 DMASM 磁盘逻辑结构

创建、删除 DMASM 文件操作，在 DMASM 系统内部其实就是转换成修改、维护 inode AU 的具体动作。

而扫描全局的 inode AU 链表就可以获取到磁盘组上所有的 DMASM 文件信息。

### 6.3.2 DMASM redo 日志

DMASM 采用重做日志机制，保证在各种异常（比如系统掉电重启）情况下数据不被损坏。创建、删除 DMASM 文件等 DDL 操作过程中，所有针对 DMASM 描述 AU、inode AU 的修改，都会生成 redo 日志，并且在描述 AU、inode AU 的修改写入磁盘之前，必须确保 redo 日志已经写入磁盘。DMASM 中，只针对描述 AU 和 inode AU 的修改产生 redo 日志，用户修改数据 AU 的动作并不会产生 redo 日志。

DMASM 所有 DDL 操作（创建文件、删除文件、增加磁盘等）都是串行执行的，并且在操作完成之前，会确保所有修改的描述项、inode 项写入磁盘；一旦 DDL 操作完成，所有 redo 日志就可以被覆盖了。

DDL 操作过程中出现异常时，如果 redo 日志尚未写入磁盘，则当前操作对系统没有任

何影响；如果 redo 日志已经写入磁盘，那么重新启动后，系统会重演 redo 日志，修改描述 AU 和 inode AU，将此 DDL 继续完成。

### 6.3.3 簇映射表

创建 DMASM 文件后，用户操作 DMASM 文件的一般流程是：调用 DMASM 文件的 OPEN、READ、WRITE 接口，打开 DMASM 文件并获取一个句柄，再使用这个句柄从文件的指定偏移读取数据、或者写入数据。用户在使用 DMASM 的过程中，只需要获取一个 DMASM 文件句柄，并不需要知道数据最终保存在物理磁盘的什么位置

DMASM 使用簇映射表（extent map）机制维护 DMASM 文件与物理磁盘地址的映射关系，访问 DMASM 文件时，根据文件号、文件偏移等信息，通过簇映射表可以快速获取到物理磁盘地址。

由于 DMASM 并不缓存任何用户数据，与直接读、写裸设备相比，DMASM 文件的读、写操作仅仅增加了簇映射的代价，而这个代价与 IO 代价相比几乎可以忽略，因此，使用 DMASM 并不会引起读、写性能的降低。

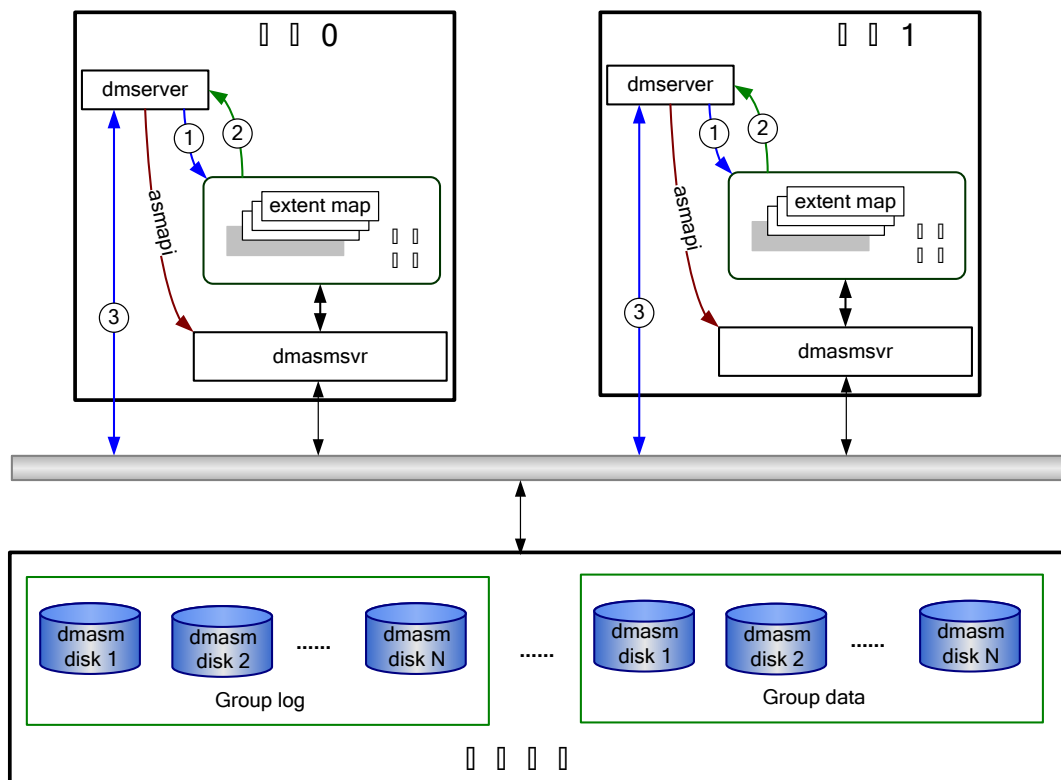


图 6.3 dmasm 数据访问



## 6.4 DMSM 技术指标

本节主要介绍一下 DMSM 中的一些重要技术指标。

表 6.2 DMSM 技术指标

项	大小	说明
AU 大小	1024 * 1024	一个 AU 占用 1M 存储空间
簇大小	4	一个簇包含 4 个物理上连续的 AU
描述项大小	32	一个簇描述项占用 32 个字节的存储空间
描述 AU 管理的最大簇数 目	16 * 1024	一个描述 AU 最多管理 16384 个簇
描述 AU 管理的最大 AU 数	64 * 1024	一个描述 AU 最多可以管理 65536 个 AU
描述 AU 管理的最大磁盘 空间	64G	一个描述 AU 最多可以管理 64G 磁盘空间
Inode 项大小	512	一个 DMSM 文件描述项大小，每个 DMSM 文件/目录都对应着一个文件描述项目
Inode AU 可管理的最大 文件数	2046	一个 Inode AU 最多可以管理 2046 个文件
DMSM 文件最小尺寸	4M	每个 DMSM 文件最少包含一个簇
DMSM 文件最大尺寸	4PB	一个 DMSM 文件最多可以包含 4294967295 个 AU，每个 AU 是 1M，理论上单个 DMSM 文件的最大尺寸是 4PB
一个用户连接可同时打开 的 DMSM 文件数	65536	一个用户连接，最多打开 65536 个 DMSM 文件
DMSM 文件数上限	8388607	一个磁盘组，最多可以创建 8388607 个 DMSM 文件
磁盘组个数上限	124	最多可创建 124 个磁盘组
每 10M 共享内存大小能管 理的磁盘大小	约 600G	每个簇描述项大概占用 64byte 内存空间，每个簇描述项对应 4M 磁盘空间，通过计算 $10M / 64 * 4M = 655G$ ；655G 左右的磁盘，使用 10M 大小的共享内存能保证使用过程中簇描述项不被淘汰

## 6.5 DMASM 使用说明

1. 使用 DMASM 必须先使用 `dmasmcmd` 工具初始化 DMASM 磁盘和磁盘组，并启动 `dmassvr` 服务器。使用者（`dmserver` 等）必须通过 `dmasmapi` 接口登录 `dmassvr` 创建 DMASM 文件，并进行各种 DMASM 文件操作。

2. DMASM 文件的读写接口与普通的操作系统文件类似，主要的区别就是需要使用专用的 `dmasmapi` 接口进行操作。

3. 一台共享存储上，只能搭建一套 ASM 文件系统，多套会导致系统启动失败。

4. 已经 `open`、正在访问的 DMASM 文件不允许删除。

5. DMASM 文件可以重复打开，但是建议用户在使用过程中，尽量避免反复打开同一个 DMASM 文件。如果用户反复打开同一个 DMASM 文件，并且没有及时关闭文件，有可能会降低 DMASM 文件的访问效率。

6. DMASM 文件句柄不保证全局唯一，只保证连接级别的唯一性，一个连接重复打开同一个 DMASM 文件，会返回不同的文件句柄；不同连接打开同一个 DMASM 文件，有可能返回相同的文件句柄。

7. 支持删除磁盘组，但不能单独删除磁盘组中的某一块磁盘

8. 任意文件 `open` 的情况下，其所属的磁盘组不能被删除。

9. DMASM 文件路径都以 `" +GROUP_NAME"` 开头，使用 `" / "` 作为路径分隔符，任何以 `" + "` 开头的文件，我们都认为是 DMASM 文件，`"GROUP_NAME"` 是磁盘组名称。比如，`" +DATA/ctl/dm.ctl"` 表示 `dm.ctl` 文件，保存在 DMASM 文件系统的 `"DATA"` 磁盘组的 `ctl` 目录下。`" + "` 号只能出现在全路径的第一位，出现在任意其他地方的路径都是非法的。

10. DMASM 只提供文件级别的并发控制，访问 DMASM 文件，系统内部会进行封锁操作。比如，正在访问的数据文件不允许被删除；但是 DMASM 并不提供数据文件的读写并发控制。DMASM 允许多个用户同时向同一个文件的相同偏移写入数据，一旦发生这种并发写，我们无法预知最终写入磁盘的数据是什么。因此，DMASM 不是一个通用的分布式文件系统，必须由使用 DMASM 的上层应用来控制数据文件的读写并发。采用这种实现策略的主要原因有两个：

◆ 提升 DMASM 文件的读写效率。通用的分布式文件系统，需要实现读、写操作的全局并发控制，避免并发写入导致数据不一致，这种策略会严重影响读、写性能。

◆ 数据库管理软件已经提供了数据访问的并发控制机制，确保不会同时读、写相同的

数据页，DMASM 不需要实现一套重复的并发控制策略。

11. 目前，DMASM 还未实现异步格式化机制，创建磁盘组、添加磁盘等操作需要较长的执行时间，并且格式化过程中会阻塞创建 DMASM 文件等操作。

12. 目前，DMASM 只提供了基本的数据文件管理功能，并不支持镜像存储、条带化存储、数据再平衡等功能。

## 6.6 DMASMCMD

DMASMCMD 是 DMASM 文件系统初始化工具，用来格式化裸设备为 DMASM 磁盘，并初始化 DCR Disk、Voting Disk。格式化 DMASM 磁盘就是在裸设备的头部写入 DMASM 磁盘特征描述符号，包括 DMASM 标识串、DMASM 磁盘名、以及 DMASM 磁盘大小等信息。其中 Voting Disk 和 DCR Disk 也会被格式化为 DMASM 磁盘。

DMASMCMD 工具的主要功能包括：

1. 格式化 DMASM 磁盘
2. 初始化 DCR Disk，同时指定密码
3. 初始化 Voting Disk
4. 导出 DCR Disk 配置信息
5. 导入 DCR Disk 配置信息
6. 清理 DCR Disk 中指定组的故障节点信息
7. 创建用于模拟裸设备的磁盘文件（用于单机模拟 DMDSC 环境）
8. 列出指定路径下面磁盘属性
9. 联机修改 DCR 磁盘，扩展节点

DMASMCMD 工具用法：

格式：`dmasmcmd KEYWORD=value`

例如：`dmasmcmd SCRIPT_FILE=asmcmd.txt`

1. 用户没有指定脚本文件，则 `dmasmcmd` 进入交互模式运行，逐条解析、运行命令。
2. 用户指定脚本文件（比如 `asmcmd.txt`），则以行为单位读取文件内容，并依次执行，执行完成以后，自动退出 `dmasmcmd` 工具。脚本文件必须以“`#asm script file`”开头，否则认为是无效脚本文件；脚本中其它行以“`#`”表示注释；脚本文件大小不超过 1M。

DMASMCMD 命令行执行界面下支持的语句包括：

- **创建 DMASM 磁盘**

```
Format: create asmdisk disk_path disk_name [size(M)]
Usage: create asmdisk '/home/asmdisks/disk0.asm' 'DATA0'
Usage: create asmdisk '/home/asmdisks/disk0.asm' 'DATA0' 100
```

用来将裸设备格式化为 ASM Disk，会在裸设备头部写入 ASM Disk 标识信息。size 取值最小 32。

```
Format: create dcrdisk disk_path disk_name [size(M)]
Usage: create dcrdisk '/home/asmdisks/disk0.asm' 'DATA0'
Usage: create dcrdisk '/home/asmdisks/disk0.asm' 'DATA0' 100
```

用来将裸设备格式化为 DCR 磁盘，会在裸设备头部写入 DCR 标识信息。size 取值最小 32。

```
Format: create votedisk disk_path disk_name [size(M)]
Usage: create votedisk '/home/asmdisks/disk0.asm' 'DATA0'
Usage: create votedisk '/home/asmdisks/disk0.asm' 'DATA0' 100
```

用来将裸设备格式化为 Voting Disk，会在裸设备头部写入 Voting Disk 标识信息。

以上三个命令中 size 参数可以省略，程序会计算 disk\_path 的大小；但是某些操作系统计算 disk\_path 大小会失败，这时候还是需要用户指定 size 信息，size 取值最小为 32。

- **创建空文件模拟裸设备**

```
Format: create emptyfile file_path size(M) num
Usage: create emptyfile '/opt/data/asmdisks/disk0.asm' size 100
```

创建 disk0.asm 模拟裸设备，注意必须以 .asm 结尾。模拟测试用，真实环境不建议使用。

- **初始化 DCR & Voting Disk**

```
Format: init dcrdisk disk_path from ini_path identified by password
Usage: init dcrdisk '/dev/raw/raw2' from '/home/asm/dmdcr_cfg.ini' identified by
```

```
'aaaaaa'
```

根据配置文件 `dmdcr_cfg.ini` 的内容，初始化 DCR 磁盘。设置登录 ASM 文件系统的密码，密码要用单引号括起来。

```
Format: init votedisk disk_path from ini_path
```

```
Usage: init votedisk '/dev/raw/raw3' from '/home/asm/dmdcr_cfg.ini'
```

根据配置文件 `dmdcr_cfg.ini` 的内容，初始化 Voting Disk。

### ● 导出 DCR 的配置文件

```
Format: export dcrdisk disk_path to ini_path
```

```
Usage: export dcrdisk '/dev/raw/raw2' to '/home/asm/dmdcr_cfg.ini'
```

解析 DCR 磁盘内容，导出到 `dmdcr_cfg.ini` 文件。

### ● 导入 DCR 的配置文件

```
Format: import dcrdisk ini_path to disk_path
```

```
Usage: import dcrdisk '/data/dmdcr_cfg.ini' to '/data/asmdisks/disk0.asm'
```

根据配置文件 `/data/dmdcr_cfg.ini` 的内容，将修改导入 DCR 磁盘。

### 限制：

DCR\_N\_GRP、DCR\_GRP\_N\_EP、DCR\_GRP\_NAME、DCR\_GRP\_TYPE、DCR\_GRP\_EP\_ARR、DCR\_EP\_NAME、EP 所属的组类型、checksum 值，密码不可修改。

必须以组为单位进行修改。

### ● 校验 DCR 磁盘

```
Format: check dcrdisk disk_path
```

```
Usage: check dcrdisk '/dev/raw/raw2'
```

校验 DCR 磁盘信息是否正常，根据打印出来的 code 值判断，如果等于 0，则表示 DCR 磁盘正常，如果小于 0，则说明 DCR 磁盘故障，需要重新初始化。

### ● 清理指定组的故障节点信息

```
Format: clear dcrdisk err_ep_arr disk_path group_name
```

```
Usage: clear dcrdisk err_ep_arr '/dev/raw/raw2' 'GRP_DSC'
```

清理 DCR Disk 中指定组的故障节点信息，可借助 `export` 命令查看对应组的 DCR\_GRP\_N\_ERR\_EP 和 DCR\_GRP\_ERR\_EP\_ARR 信息，清理成功后，指定组的 DCR\_GRP\_N\_ERR\_EP 值为 0，DCR\_GRP\_ERR\_EP\_ARR 内容为空。

- 显示指定路径下面磁盘属性

```
Format: listdisks path
```

```
Usage: listdisks '/dev/raw/'
```

显示 path 路径下面所有磁盘的信息，分为三种类型：

normal disk: 普通磁盘；  
unused asmdisk: 初始化未使用的 asmdisk；  
used asmdisk: 已经使用的 asmdisk。

- 联机修改 DCR 磁盘，扩展节点

```
Format: extend dcrdisk disk_path from ini_path
```

```
Usage: extend dcrdisk 'd:\asmdisks\disk0.asm' from 'd:\dmdcr_cfg.ini'
```

联机修改 DCR 磁盘，扩展节点，会将新增节点信息写回 dcr 磁盘。

## 6.7 DMASMSVR

DMASMSVR 是提供 DMASM 服务的主要载体，每个提供 DMASM 服务的节点都必须启动一个 DMASMSVR 服务器，这些 DMASMSVR 一起组成共享文件集群系统，提供共享文件的全局并发控制。DMASMSVR 启动时扫描 /dev/raw/ 路径下的所有裸设备，加载 DMASM 磁盘，构建 DMASM 磁盘组和 DMASM 文件系统。DMASMSVR 实例之间使用 MAL 系统进行信息和数据的传递。

DMASMSVR 集群的启动、关闭、故障处理等流程由 DMCSS 控制，DMASMSVR 定时向 Voting Disk 写入时间戳、状态、命令、以及命令执行结果等信息，DMCSS 控制节点定时从 Voting Disk 读取信息，检查 DMASMSVR 实例的状态变化，启动相应的处理流程。

DMASMSVR 集群中，只有一个控制节点，控制节点以外的其他节点叫做普通节点，

DMASMSVR 控制节点由 DMCSS 选取；所有 DDL 操作（比如创建文件，创建磁盘组等）都是在控制节点执行，用户登录普通节点发起的 DDL 请求，会通过 MAL 系统发送到控制节点执行并返回；而 DMASM 文件的读、写等操作，则由登录节点直接完成，不需要传递到控制节点执行。

DMASMSVR 启动格式：

格式：dmasmsvr.exe KEYWORD=value

例如：dmasmsvr.exe DCR\_INI=/home/data/DAMENG/dmdcr.ini

关键字说明（默认）

DCR_INI	dmdcr.ini 路径
-NOCONSOLE	以服务方式启动
HELP	打印帮助信息



dmdcr.ini 配置文件记录了 DCR 磁盘路径、实例序列号等信息；如果不指定

注意：DCR\_INI 参数，dmasmsvr 默认在当前路径下查找 dmdcr.ini 文件。



说明：

使用 DMASMT00L 或 DMASMAPI 函数 dmasm\_connect() 登录 DMASMSVR 时，用户名只能为“ASMSYS”。若为本地登录，不校验密码；若为远程登录，应使用初始化 DCR 磁盘时设置的登录 ASM 文件系统的密码。

## 6.8 DMASMAPI

DMASMAPI 是 DMASM 文件系统的应用程序访问接口，通过调用 DMASMAPI 接口，用户可以访问、操作 DMASM 文件。与达梦数据库接口 DPI 类似，访问 DMASM 文件之前，必须先分配一个 conn 对象，并登录到 DMASMSVR 服务器，再使用这个 conn 对象进行创建磁盘组、创建文件、删除文件、读取数据和写入数据等 DMASM 相关操作。

下面举一个简单的例子，说明一下 DMASMAPI 的使用方法，DMASMAPI 接口的详细使用说明请参考附录 1。

```
#include<stdio.h>
#include<stdlib.h>
```

```

#include<string.h>
#include"dmdcr_cfg.h"
#include"dcr_dll.h"
#include"apitype.h"
#include"asmapi2.h"
#define IS_SUCCESS(code)          ((sdint4)code>= 0)

int
main (
int      argc,
char*    argv[]
)
{
sdint4  code = 0;
sdbyte  err_desc[1024];
udint4  err_len;
dmdcr_cfg_sys_t*  dcr_cfg_sys;
dcr_ep_t  dcr_ep;
asm_fhandle_t  fil_handle;
asmcon_handle  conn;
err_len      = sizeof(err_desc);
code          = dmasm_sys_init(err_desc, &err_len, ULINT_UNDEFINED);
if (!IS_SUCCESS(code))
{
fprintf(stderr, "dmasm_sys_init error:[%s]\n", err_desc);
return -1;
}
/* dcr初始化相关 */
code          = dmdcr_cfg_sys_init("dmdcr.ini");
if (!IS_SUCCESS(code))
{
fprintf(stderr, "dmdcr_cfg_sys_init error, code:[%d]\n", code);
return -1;
}
dcr_cfg_sys = dmdcr_cfg_sys_get();
code          = dcr_sys_init(dcr_cfg_sys->dcr_path, err_desc, &err_len);
if (!IS_SUCCESS(code))
{
fprintf(stderr, "dcr_sys_init error:[%s]\n", err_desc);
return -1;
}
dcr_dll_get_ep_info_by_type(DCR_TYPE_ASM, dcr_cfg_sys->inst_seqno, &dcr_ep);
/* 分配连接句柄 */
code          = dmasm_alloc_con(&conn, err_desc, &err_len);

```



```
if (!IS_SUCCESS(code))
{
    fprintf(stderr, "dmasm_alloc_con error:[%s]\n", err_desc);
    return -1;
}

/* 登录asmsvr, 本地登录不校验密码, 可以输入任意字符串, 远程登录会校验用户名密码*/
code = dmasm_connect(conn, "ASMSYS", "xxxxxx", dcr_ep.ep_host,
dcr_ep.ep_lsnr_port, NULL, err_desc, &err_len);
if (!IS_SUCCESS(code))
{
    fprintf(stderr, "dmasm_connect error:[%s]\n", err_desc);
    return -1;
}

/* 创建磁盘组, 组名 GRP01, 磁盘/dev/raw/raw2 */
code = dmasm_create_diskgroup(conn, "GRP01", "/dev/raw/raw2", err_desc,
&err_len);
if (!IS_SUCCESS(code))
{
    fprintf(stderr, "dmasm_create_diskgroup error:[%s]\n", err_desc);
    return -1;
}

/* 在新创建的磁盘组内创建asm文件 */
code = dmasm_file_create(conn, 1, "+GRP01/test1.dta", 20, &fil_handle,
err_desc, &err_len);
if (!IS_SUCCESS(code))
{
    fprintf(stderr, "dmasm_file_create error:[%s]\n", err_desc);
    return -1;
}

/* 关闭打开的文件句柄, 否则删除会报错 */
dmasm_file_close(conn, fil_handle);
code = dmasm_file_delete(conn, "+GRP01/test1.dta", err_desc, &err_len);
if (!IS_SUCCESS(code))
{
    fprintf(stderr, "dmasm_file_delete error:[%s]\n", err_desc);
    return -1;
}

/* 删除asm磁盘组, 同样如果磁盘组中有任何文件open, 删除磁盘组会报错 */
code = dmasm_drop_diskgroup_by_name(conn, "GRP01", err_desc, &err_len);
if (!IS_SUCCESS(code))
{
    fprintf(stderr, "dmasm_drop_diskgroup_by_name error:[%s]\n", err_desc);
    return -1;
}
```

```
dmasm_close_con(conn);  
dmasm_free_con(conn);  
dmasm_sys_deinit();  
return 0;  
}
```

## 6.9 DMASMTOOL

DMASMTOOL 是 DMASM 文件系统管理工具，提供了一套类 Linux 文件操作命令，用于管理 DMASM 文件，是管理、维护 DMASM 的好帮手。DMASMTOOL 工具使用 DMASMAPI 连接到 DMASVR，并调用相应的 DMASMAPI 函数，实现创建、拷贝、删除等各种文件操作命令；DMASMTOOL 还支持 DMASM 文件和操作系统文件的相互拷贝。

DMASMTOOL 可以登录本地 DMASMSVR，也可以登录位于其他节点的 DMASMSVR，并执行各种文件操作命令。一般建议登录本地 DMASMSVR 服务器，避免文件操作过程中的网络开销，提升执行效率。

DMASMTOOL 启动命令：

格式：dmasmtool.exe KEYWORD=value

例如：dmasmtool.exe DCR\_INI=/home/data/DAMENG/dmdcr.ini

关键字说明（默认）

-----	
DCR_INI	dmdcr.ini 文件路径
HOST	asm 服务器地址
PORT_NUM	asm 服务器端口号
USERID	登录 asm 服务器用户名密码，（格式：USER/PWD）
SCRIPT_FILE	asmtool 脚本文件路径
HELP	打印帮助信息

- 1) `dmasmtool` 的最大命令长度是 1024
- 2) 用户没有指定脚本文件, 则 `dmasmtool` 进入交互模式运行, 逐条解析、允许命令; 用户指定脚本文件(比如 `asmttool.txt`), 则以行为单位读取文件内容, 并依次执行, 执行完成以后, 自动退出 `dmasmtool` 工具。脚本文件必须以 `"#asm script file"` 开头, 否则认为是无效脚本文件; 脚本中其他行以 `"#"` 开始表示注释。
- 3) `dmasmtool` 命令直接输入的 `host/ip` 信息配置的是连接 `asmsvr` 的信息, 可以在 `dmdcr_cfg.ini` 里面找到, 分别为要连接的 ASM 节点的 `DCR_EP_HOST` 和 `DCR_EP_PORT`



注意:

DMASMTTOOL 支持的命令说明:

#### 1. 创建磁盘组, 添加磁盘, 删除磁盘组

##### ● 创建磁盘组

```
Format: create diskgroup name asmdisk file_path
```

```
Usage: create diskgroup 'DMDATA' asmdisk '/dev/raw/raw3'
```

`asmdisk` 为磁盘组名, 最长不能超过 32 字节。路径必须是全路径, 不能是相对路径。

##### ● 添加磁盘

```
Format: alter diskgroup name add asmdisk file_path
```

```
Usage: alter diskgroup 'DMDATA' add asmdisk '/dev/raw/raw4'
```

`asmdisk` 路径必须是全路径, 不能是相对路径。

##### ● 删除磁盘组

```
Format: drop diskgroup name
```

```
Usage: drop diskgroup 'DMDATA'
```

创建磁盘组, 或为磁盘组添加磁盘时, 以下情况可能导致失败:

- DMASMSVR 进程没有访问对应磁盘的权限;
- 磁盘路径不在 `dmdcr_cfg.ini` 配置文件中配置的 `DCR_EP_ASM_LOAD_PATH` 路径下;
- 磁盘大小不够, 最少需要 32M。

## 2. 创建文件，扩展文件，截断文件，删除文件

### ● 创建文件

Format: create asmfile file\_path size(M) num

Usage: create asmfile '+DMDATA/sample.dta' size 20



**注意：**文件创建大小最大不超过 256G，即 262144M

### ● 扩展文件

Format: alter asmfile file\_path extend to size(M)

Usage: alter asmfile '+DMDATA/sample.dta' extend to 20



**注意：**文件扩展的目标大小减去原大小的差值最大不超过 256G，即 262144M

### ● 截断文件

Format: alter asmfile file\_path truncatetosize(M)

Usage: alter asmfile '+DMDATA/sample.dta' truncateto 20

### ● 删除文件

Format: delete asmfile file\_path

Usage: delete asmfile '+DMDATA/sample.dta'

### ● 重定向输出文件

Format: spool file\_path [create|replace|append]

Usage: spool /home/dataspool.txt

- 1) create: 如果重定向文件不存在，则创建；如果存在，创建失败。
- 2) replace: 如果重定向文件不存在，则创建；如果存在，则替换掉。默认为 replace。
- 3) append: 如果重定向文件不存在，则创建；如果存在，则追加到文件末尾。
- 4) 多次 spool 重定向文件，第一次成功打开重定向文件之后，如果未关闭，则不再打开其他重定向文件。

### ● 关闭重定向文件

Format: spool off

Usage: spool off

### 3. 兼容 LINUX 一些命令，功能受限，但是很实用

- 到达某目录

Format: `cd[path]`

Usage: `cd +DMDATA/test`

- 拷贝

Format: `cp [-rf] src_file_path dst_file_path`

Usage: `cp '+DMDATA/aa/sample.dta' '+DMDATA/a/b.dta'`

`cp -r '+DMDATA/aa' '+DMDATA/bb'`

`cp -f '+DMDATA/aa/sample.dta' '+DMDATA/a/b.dta'`

- 删除

Format: `rm file_path`

`rm -r directorie`

Usage: `rm '+DMDATA/a/sample.dta'`

`rm -r '+DMDATA/a/'`

`rm -f '+DMDATA/b/'`

- 创建目录

Format: `mkdir [-p] dir_path`

Usage: `mkdir '+DMDATA/a'`

`Mkdir -p '+DMDATA/nodir/bb'`



注意：这里 **-p** 表示自动创建不存在的中间目录

- 查找

Format: `find path file_name`

sage: `find +DMDATA/a 'sample.dta'`

- 显示

Format: `ls [-lr] filename`

Usage: `ls`

`ls -l`

`ls -r`



这里-r 表示递归的意思，不是 linux 中的逆序显示  
注意：

- 显示存储信息

```
Format: df
```

```
Usage: df
```

- 当前目录

```
Format: pwd
```

```
Usage: pwd
```

#### 4. DMSM 特有的一些命令

- 列出所有的磁盘组

```
Format: lsdg
```

```
Usage: lsdg
```

- 列出所有的 DMSM 磁盘

```
Format: lsdsk
```

```
Usage: lsdsk
```

- 列出文件的详细信息

```
Format: lsattr
```

```
Usage: lsattr
```

- 列出所有的信息，包括文件等

```
Format: lsall
```

```
Usage: lsall
```

- 修改密码

```
Format: password
```

```
Usage: password
```

- 登录，在断开连接后，重新登录

```
Format: login
```

```
Usage: login
```

1) `DMASMTTOOL` 工具的上下键, 查找历史记录, 以及 `TAB` 键的自动补齐功能是基于 `readline` 实现的, 由于 `readline` 输入不支持中文, 因此目前 `DMASMTTOOL` 工具不支持中文输入。

2) 若文件名或路径中包含中文字符, 当客户端与服务器的编码不一致时, 可能出现乱码, 因此建议不要使用包含中文字符的文件名或路径。



注意: 3) 在 `dmasmtool` 工具执行命令时, 如果路径名或者文件名首字母为数字, 需要给字符串加单引号'。

4) 在 `asmtool` 工具使用 `cd` 命令, 首字母为 '+', '\', '/' 符号都会从目录最上层开始查找目录。

5) `DMASM` 文件全路径长度不能超过 256 字节。

## 7 DMDSC 启动、关闭流程

DMDSC 是基于共享存储的数据库集群系统，包含多个数据库实例，因此，与单节点的达梦数据库不同，DMDSC 集群需要在节点间进行同步、协调，才能正常地启动、关闭。启动 DMDSC 集群之前，必须先启动集群同步服务 DMCSS，如果使用了 DMASM 文件系统，则 DMASMSVR 服务也必须先启动。

DMASMSVR 控制台执行 `exit` 命令，会通知所有其他节点一起退出；`dmserver` 需要手动退出所有节点，没有同步功能；Linux 环境下 DMASMSVR/dmserver 都监控了操作系统 SIGTERM 信号，Linux 环境执行不带参数的 `kill` 命令，DMASMSVR/dmserver 都能正常退出。

如果 DMCSS 配置了 DMASMSVR/dmserver 自动拉起命令，可以先仅启动 DMCSS，然后启动 DMCSSM，在 DMCSSM 控制台执行命令 `ep startup asm` 启动 DMASMSVR 集群，执行 `ep startup dsc` 启动 dmserver 集群（其中 `asm/dsc` 为 `dmassmsvr/dmserver` 集群的组名）。类似地执行 `ep stop asm/dsc` 可以关闭 `dmassmsvr/dmserver` 集群环境。

Linux 环境下，`dmcss` / `dmassmsvr` / `dmserver` 可以配置成操作系统服务，每次开机自动启动，或者通过 Linux 命令 `service XXX start/stop/restart` (XXX 为配置的服务名) 完成服务的启动、关闭。服务脚本在达梦安装包提供，可能还需要根据实际情况修改部分参数才能使用。



## 8 DMDSC 故障处理

DMDSC 集群出现数据库实例、或者节点硬件故障时，dmserver 的 Voting disk 心跳信息不再更新，DMCSS 一旦监控到 dmserver 发生故障，会马上启动故障处理（参考第 5 章 DMCSS 介绍），各节点 dmserver 收到故障处理命令后，启动故障处理流程。

在 DMDSC 故障处理机制下，一旦产生节点故障，登录到故障节点的所有连接将会断开，所有未提交事务将被强制回滚；活动节点上的用户请求可以继续执行，但是一旦产生节点间信息传递（比如：向故障节点发起 GBS/LBS 请求、或者发起 remote read 请求），当前操作就会被挂起；在 DMDSC 故障处理完成后，这些被挂起的操作可以继续执行。也就是说，DMDSC 产生节点故障时，活动节点上的所有连接会保留，正在执行的事务可能被阻塞一段时间，但最终可以正常完成，不会被强制回滚，也不会影响结果的正确性。

DMDSC 环境 dmserver 故障处理主要包括以下工作：

1. 暂停所有会话线程、工作线程、日志刷盘线程、检查点线程等，避免故障处理过程中产生并发错误；
2. 收集所有活动节点的 BUFFER，丢弃无效数据页，获取最新数据页和需要重做故障节点 REDO 日志的数据页信息，并在排除故障节点后重新构造 LBS/GBS 信息；
3. 重做故障节点 REDO 日志；
4. 收集所有节点事务信息，重新构造锁对象，并重构相应的 LLS/GLS/LTV/GTV 系统；
5. 如果配置有 VIP，进行必要的 VIP 配置操作；
6. 控制节点执行故障节点活动事务回滚和故障节点已提交事务修改的 PURGE 操作。

DMDSC 故障处理分为两个阶段；第一阶段由所有活动节点共同参与，进行全局的信息收集、重构；第二阶段由控制节点执行，将故障节点的活动事务回滚、并 PURGE 故障节点已提交事务的修改。在第一阶段执行期间，数据库实例不提供数据库服务，所有用户请求将被挂起。在第二阶段操作之前，会唤醒所有活动节点，正常提供数据库服务。也就是说故障处理第二阶段的操作与正常的数据库操作在系统内部同时进行。但在第二阶段执行完成之前，DMDSC 故障处理仍然没有真正结束，在此期间，不能处理节点重加入，也不能处理新的节点故障，如果有新的节点故障会主动中止所有节点。

1. 通常意义上讲的故障处理，包含故障检测和故障处理两部分

2. 故障检测时间由 DCR\_GRP\_DSKCHK\_CNT 决定



注意：

3. 影响故障处理时间的因素比较多，主要包括：故障节点 redo 日志数量、修改涉及的数据页多少、需要 ROLLBACK、PURGE 的事务数，以及活动节点 Buffer 使用情况等。

## 9 DMDSC 节点重加入

故障节点恢复正常后，DMCSS 会启动节点重加入处理流程，将故障节点重新加入到 DMDSC 集群中。默认情况下，DMCSS 自动监控并处理节点重加入，不需要用户干预；我们也可以通过 DMCSSM 关闭自动监控功能，改成手动处理节点重加入（参考 DMCSSM 监视器说明）。

DMDSC 集群节点重加入操作会将 DMDSC 集群挂起一段时间，重加入过程中会中断正在执行的操作，暂停响应用户的数据库请求。但是，重加入操作不会终止这些活动事务，在重加入完成操作后，可以继续执行。

DMDSC 节点重加入的基本步骤包括：

1. 根据各节点的 BUFFER 使用情况，重构所有节点的 GBS、LBS 系统
2. 根据各节点的活动事务信息，重构全局的 GTV 系统、以及所有节点的本地 LTV 系统
3. 根据各节点的活动事务的封锁（LOCK）信息，重构全局的 GLS 系统、以及所有节点的本地 LLS 系统

## 10 配置说明

与 DMDSC 相关的配置文件包括：

- DMDCR\_CFG.INI
- DMDCR.INI
- DMINIT.INI
- MAL 系统配置文件（DMMAL.INI、DMASVRMAL.INI）
- DM.INI

下面分别介绍配置文件中各配置项的含义。

### 10.1 DMDCR\_CFG.INI

dmdcr\_cfg.ini 是格式化 DCR 和 Voting Disk 的配置文件。配置信息包括三类：集群环境全局信息、集群组信息、以及组内节点信息。

使用 dmasmcmd 工具，可以根据 dmdcr\_cfg.ini 配置文件，格式化 DCR 和 Voting Disk。

表 10.1 dmdcr\_cfg.ini 配置项

集群环境全局信息	
DCR_VTD_PATH	Voting Disk 路径
DCR_N_GRP	集群环境包括多少个 group，取值范围 1~16
DCR_OGUID	消息标识，dmcsm 登录 dmcsm 消息校验用
集群组信息	
DCR_GRP_TYPE	组类型（CSS\ASM\DB）
DCR_GRP_NAME	组名，16 字节，配置文件内不可重复
DCR_GRP_N_EP	组内节点个数 N，最大 16
DCR_GRP_EP_ARR	组内包含的节点序列号， $\{0, 1, 2, \dots, N-1\}$ 用户不能指定，仅用于从 DCR 磁盘 export 出来可见。
DCR_GRP_N_ERR_EP	组内故障节点个数 用户不能指定，仅用于从 DCR 磁盘 export 出来可见。
DCR_GRP_ERR_EP_ARR	组内故障节点序列号 用户不能指定，仅用于从 DCR 磁盘 export 出来可见。

## DM8 共享存储集群

DCR_GRP_DSKCHK_CNT	磁盘心跳机制，容错时间，单位秒，缺省 60S，取值范围 5~600
<b>节点信息，某些属性可能只针对某一类型节点，比如 SHM_KEY 只针对 ASM 节点有效</b>	
DCR_EP_NAME	节点名，16 字节，配置文件内不可重复，  DB 的节点名必须和 dm.ini 里的 INSTANCE_NAME 保持一致  ASM 的节点名必须和 dmmal.ini 里的 MAL_INST_NAME 一致  同一类型节点的 EP_NAME 不能重复
DCR_EP_SEQNO	组内序号，CSS/ASM 不能配置，自动分配  DB 可以配置，0 ~ n_ep -1，组内不能重复，如不配置则自动分配
DCR_EP_HOST	节点 IP (CSS/ASM 有效，表示登录 CSS/ASM 的 IP 地址)  对 DB 来说，是绑定 VIP 的网卡对应的物理 IP 地址  CSS 中设置表示 DMCSM 通过该 IP 连接 CSS；ASM 中设置表示 DB 通过该 IP 连接 ASM
DCR_EP_PORT	节点 TCP 监听端口 (CSS/ASM/DB 有效，对应登录 CSS/ASM/DB 的端口号)，取值范围 1024~65534。特别对 DB 来说，DB 的 DCR_EP_PORT 与 dm.ini 中的 PORT_NUM 不一致时，DB 端口以 DCR_EP_PORT 为准。  若要使用 VIP 功能，则不同服务器上配置的 DCR_EP_PORT 要相同
DCR_EP_SHM_KEY	共享内存标识，数值类型 (ASM 有效，初始化共享内存的标识符)，应为大于 0 的 4 字节整数
DCR_VIP	节点 VIP (DB 有效，表示配置的虚拟 IP)，需要和 DCR_EP_HOST 在同一网段。若需要取消 VIP 配置，仅需要将 DB 组的 DCR_VIP 和 DCR_EP_HOST 删除即可
DCR_CHECK_PORT	DCR 检查端口号。检查实例是否活动的时候用，各实例不能冲突
DCR_EP_SHM_SIZE	共享内存大小，单位 M，(ASM 有效，初始化共享内存大小)，取值范围 10~1024
DCR_EP_ASM_LOAD_PATH	ASM 磁盘扫描路径，Linux 下一般为 /dev/raw，文件模拟情况，必须是全路径，不能是相对路径

### 使用说明

1. 在用 dmasmcmd 工具执行 `init votedisk disk_path from dcr_cfg_path` 时，指定的 `disk_path` 必须和 `dcr_cfg_path` 里面配置的 `DCR_VTD_PATH` 相同。

2. 如果配置 dmcssm, dmcssm 的 OGUID 必须和 DCR\_OGUID 保持一致
3. DCR\_N\_GRP 必须和实际配置的组数目保持一致
4. CSS 和 ASM 组的 DCR\_GRP\_N\_EP 要相等, DB 的 DCR\_GRP\_N\_EP 要小于等于 CSS/ASM 的 DCR\_GRP\_N\_EP。
5. ASM 节点的 DCR\_EP\_NAME 必须和 DMASM 系统使用的 dmmal.ini 配置文件里的 MAL\_INST\_NAME 保持一致。
6. DB 节点的 DCR\_EP\_NAME 必须和数据库实例使用 dmmal.ini 配置文件里的 MAL\_INST\_NAME、以及 dm.ini 配置文件里的 INSTANCE\_NAME 保持一致。
7. 所有 DB 节点的 DCR\_EP\_NAME 都不能重复, DB 组内的 DCR\_EP\_SEQNO 不能重复。

### 举例说明

```

DCR_N_GRP          = 3

DCR_VTD_PATH       = /dev/raw/raw2

DCR_OGUID          = 63635


[GRP]              #[GRP]表示新建一个 Group

DCR_GRP_TYPE       = CSS

DCR_GRP_NAME       = CSS

DCR_GRP_N_EP       = 2

DCR_GRP_DSKCHK_CNT = 60


[CSS]              #[ ]里的是组名, 与 DCR_GRP_NAME 对应

DCR_EP_NAME        = CSS0

DCR_EP_HOST        = 10.0.2.101

DCR_EP_PORT        = 9341


[CSS]              #[ ]里的是组名, 与 DCR_GRP_NAME 对应

DCR_EP_NAME        = CSS1

DCR_EP_HOST        = 10.0.2.102

DCR_EP_PORT        = 9343


[GRP]              #[GRP]表示新建一个 Group

DCR_GRP_TYPE       = ASM

DCR_GRP_NAME       = ASM

```

```
DCR_GRP_N_EP          = 2

DCR_GRP_DSKCHK_CNT    = 60

[ASM]                  #[ ]里的是组名，与 DCR_GRP_NAME 对应

DCR_EP_NAME           = ASM0

DCR_EP_SHM_KEY        = 93360

DCR_EP_SHM_SIZE       = 10

DCR_EP_HOST           = 10.0.2.101

DCR_EP_PORT           = 9349

DCR_EP_ASM_LOAD_PATH  = /dev/raw

[ASM]                  #[ ]里的是组名，与 DCR_GRP_NAME 对应

DCR_EP_NAME           = ASM1

DCR_EP_SHM_KEY        = 93361

DCR_EP_SHM_SIZE       = 10

DCR_EP_HOST           = 10.0.2.201

DCR_EP_PORT           = 9351

DCR_EP_ASM_LOAD_PATH  = /dev/raw

[GRP]                  #[GRP]表示新建一个 Group

DCR_GRP_TYPE          = DB

DCR_GRP_NAME          = DSC

DCR_GRP_N_EP          = 2

DCR_GRP_DSKCHK_CNT    = 60

[DSC]                  #[ ]里的是组名，与 DCR_GRP_NAME 对应

DCR_EP_NAME           = DSC01

DCR_EP_SEQNO          = 0

DCR_EP_PORT           = 5236

DCR_CHECK_PORT        = 9741

[DSC]                  #[ ]里的是组名，与 DCR_GRP_NAME 对应

DCR_EP_NAME           = DSC02

DCR_EP_SEQNO          = 1

DCR_EP_PORT           = 5236
```

```
DCR_CHECK_PORT    = 9742
```

## 10.2 DMDCR.INI

dmdcr.ini 是 dmcass、dmasmsvr、dmasmtool 工具的输入参数。记录了当前节点序列号以及 DCR 磁盘路径。

表 10.2 dmdcr.ini 配置项

DMDCR_PATH	记录 DCR 磁盘路径
DMDCR_SEQNO	记录当前节点序号(用来获取 ASM 登录信息)
DMDCR_MAL_PATH	保存 dmmal.ini 配置文件的路径, 仅对 dmasmsvr 有效
DMDCR_ASM_RESTART_INTERVAL	DMCSS 认定 DMASM 节点故障重启的时间间隔 (取值 0~86400s), DMCSS 只负责和 DMDCR_SEQNO 节点号相等的 DMASM 节点的故障重启, 超过设置的时间后, 如果 DMASM 节点的 active 标记仍然为 FALSE, 则 DMCSS 会执行自动拉起。如果配置为 0, 则不会执行自动拉起操作, 默认为 60s。
DMDCR_ASM_STARTUP_CMD	DMCSS 认定 DMASM 节点故障后, 执行自动拉起的命令串, 可以配置为服务方式或命令行方式启动, 具体可参考说明部分。
DMDCR_DB_RESTART_INTERVAL	DMCSS 认定 DMDSC 节点故障重启的时间间隔 (取值 0~86400s), DMCSS 只负责和 DMDCR_SEQNO 节点号相等的 DMDSC 节点的故障重启, 超过设置的时间后, 如果 DMDSC 节点的 active 标记仍然为 FALSE, 则 DMCSS 会执行自动拉起。如果配置为 0, 则不会执行自动拉起操作, 默认为 60s。
DMDCR_DB_STARTUP_CMD	DMCSS 认定 DMDSC 节点故障后, 执行自动拉起的命令串, 可以配置为服务方式或命令行方式启动, 具体可参考说明部分。
DMDCR_AUTO_OPEN_CHECK	指定时间内如果节点实例未启动, DMCSS 会自动将节点踢出集群环境, 单位为秒, 取值范围应大于等于 30s。

### 使用说明

1. dmasmsvr 和 dmserver 使用不同的 MAL 系统, 需要配置两套 MAL 系统, 配置文件 dmmal.ini 需要分别生成, 保存到不同的目录下, 并且 dmmal.ini 中的配置项不能重复、冲突。



2. DMDSC 集群环境下, 只有当所有 OK 节点实例都启动的时, 整个集群环境才能启动。可能存在某些场景, 部分 OK 节点无法正常启动, 导致整个集群环境无法正常启动。指定参数 DMDCR\_AUTO\_OPEN\_CHECK 后, 如果超过指定时间节点实例还未启动, DMCSS 自动将未启动节点踢出集群环境, 变为 ERROR 节点, 之后其他活动 OK 节点可以正常启动。

DMDSC 中至少一个 OK 节点启动后 DMCSS 才开始检查, 所有 OK 节点都未启动情况下, DMCSS 不会主动踢出节点。

3. DMCSS 自动拉起故障 DMASMSVR 或 dmserver 实例。

故障认定间隔和启动命令串是配合使用的, DMASMSVR 和 dmserver 实例需要各自配置, 如果没有配置启动命令串, 故障间隔即使配置为大于 0 的值 DMCSS 也不会执行自动拉起操作, DMDCR\_ASM\_RESTART\_INTERVAL 和 DMDCR\_DB\_RESTART\_INTERVAL 默认的 60s 只有在配置有启动命令串时才会起作用。

DMDCR\_ASM\_STARTUP\_CMD 和 DMDCR\_DB\_STARTUP\_CMD 的配置方法相同, 只是执行码名称和 ini 配置文件路径有区别, 可以配置为服务名或命令行启动方式。

DMASMSVR 实例启动命令举例如下:

1. linux 命令行方式启动 (不能出现带有空格的路径):

```
DMDCR_ASM_STARTUP_CMD = /opt/dmdbms/bin/dmasmsvr dcr_ini=/home/data/dmdcr.ini
```

2. linux 服务方式启动:

```
DMDCR_ASM_STARTUP_CMD = service dmasmsvervd restart
```

3. Windows 命令行启动:

```
DMDCR_ASM_STARTUP_CMD = c:\dm\bin\dmasmsvr.exe dcr_ini=d:\asmtest\dmdcr.ini
```

4. Windows 服务方式启动:

```
DMDCR_ASM_STARTUP_CMD = net start 注册服务名
```

dmserver 实例启动命令如下:

1. linux 命令行方式启动 (不能出现带有空格的路径):

```
DMDCR_DB_STARTUP_CMD = /opt/dmdbms/bin/dmserver
path=/home/data/dsc0_config/dm.inidcr_ini=/home/data/dmdcr.ini
```

2. linux 服务方式启动:

```
DMDCR_DB_STARTUP_CMD = service DmServiceDSC01 restart
```

3. Windows 命令行启动:

```
DMDCR_DB_STARTUP_CMD = c:\dm\bin\dmserver.exe
path=d:\asmtest\dsc0_config\dm.inidcr_ini=d:\asmtest\dmdcr.ini
```

#### 4. Windows 服务方式启动:

```
DMDCR_DB_STARTUP_CMD = net start 注册服务名
```

#### 举例说明

```
DMDCR_PATH=/dev/raw/raw1

DMDCR_SEQNO=0

DMDCR_MAL_PATH=/home/data/dmasvrml.ini

#故障认定间隔，以及启动命令中的执行码路径和 ini 路径需要根据实际情况调整

DMDCR_ASM_RESTART_INTERVAL = 60

DMDCR_ASM_STARTUP_CMD = /opt/dmdbms/bin/dmasmsvr dcr_ini=/home/data/dmdcr.ini

DMDCR_DB_RESTART_INTERVAL = 60

DMDCR_DB_STARTUP_CMD = /opt/dmdbms/bin/dmserver
path=/home/data/dsc0_config/dm.ini dcr_ini=/home/data/dmdcr.ini
```

## 10.3 DMINIT.INI

dminit.ini 是 dminit 工具初始化数据库环境的配置文件。与初始化库使用普通文件系统不同，如果使用裸设备或者 ASM 文件系统，必须使用 dminit 工具的 control 参数指定 dminit.ini 文件。dminit 工具的命令行参数都可以放在 dminit.ini 中，比如 db\_name, auto\_overwrite 等，dminit.ini 格式分为全局参数和节点参数。dminit 工具具体用法可以参考《DM8\_dminit 使用手册》，下面列出常用的一些参数。

表 10.3 dminit.ini 配置项

全局参数，对所有节点有效	
SYSTEM_PATH	初始化数据库存放的路径
DB_NAME	初始化数据库名称
MAIN	MAIN 表空间路径
MAIN_SIZE	MAIN 表空间大小

## DM8 共享存储集群

SYSTEM	SYSTEM 表空间路径
SYSTEM_SIZE	SYSTEM 表空间大小
ROLL	ROLL 表空间路径
ROLL_SIZE	ROLL 表空间大小
CTL_PATH	DM.CTL 控制文件路径
CTL_SIZE	DM.CTL 控制文件大小
LOG_SIZE	日志文件大小
HUGE_PATH	huge 表空间路径
AUTO_OVERWRITE	文件存在时的处理方式
DCR_PATH	DCR 磁盘路径
DCR_SEQNO	连接 DMASM 节点节点号

### 节点参数，对具体节点有效

[XXX]	具体节点都是以[XXX]开始，节点实例名就是 XXX
CONFIG_PATH	配置文件存放路径
PORT_NUM	节点端口号
MAL_HOST	节点 MAL 系统使用 IP
MAL_PORT	节点 MAL 端口
LOG_PATH	日志文件路径

### 使用说明

1. SYSTEM\_PATH 目前可以支持 ASM 文件系统，如果不指定 MAIN/SYSTEM/ROLL/CTL\_PATH/HUGE\_PATH，数据文件、控制文件和 huge 表空间路径都会默认生成在 SYSTEM\_PATH/db\_name 下面。
2. 表空间路径和 DM.CTL 路径支持普通操作系统路径、裸设备、ASM 文件路径，如果指定必须指定 SIZE。
3. 只有 dminit 工具使用 ASM 文件系统，才会用到 DCR\_PATH 和 DCR\_SEQNO，并且不会写入其他配置文件。
4. CONFIG\_PATH 暂时只支持裸设备，不支持 ASM 文件系统。DSC 环境配置，各节点的 config\_path 要指定不同路径。
5. MAL\_HOST 和 MAL\_PORT 是为了自动创建 dmmal.ini 文件使用，只有在初始化

DSC 环境时需要指定。

6. LOG\_PATH 可以不指定，默认会在 SYSTEM\_PATH 生成。如果指定，必须指定两个以上。

7. 必须指定实例名，也就是必须配置[xxx]。

8. dminit 工具的 control 参数只能独立使用，不能和其他任何参数一起使用。

### 举例说明

```
system_path= +DMDATA/data

db_name= dsc

main= +DMDATA/data/dsc/main.dbf

main_size= 128

roll = +DMDATA/data/dsc/roll.dbf

roll_size= 128

system = +DMDATA/data/dsc/system.dbf

system_size= 128

ctl_path = +DMDATA/data/dsc/dm.ctl

ctl_size = 8

log_size = 256

dcr_path = /dev/raw/raw1

dcr_seqno = 0

auto_overwrite=1


[dsc01]

config_path = /home/data/config1

port_num = 5236

mal_host = 10.0.2.101

mal_port = 9340

log_path = +DMDATA/log/log101.log

log_path = +DMDATA/log/log102.log


[dsc02]

config_path = /home/data/config2
```

```
port_num = 5237

mal_host = 10.0.2.102

mal_port = 9341

log_path = +DMDATA/log/log201.log

log_path = +DMDATA/log/log202.log
```

## 10.4 MAL 系统配置文件(DMMAL.INI、DMASVRMAL.INI)

dmmal.ini 和 dmasvrml.ini 都是 MAL 配置文件。使用同一套 MAL 系统的所有实例，MAL 系统配置文件要严格保持一致。

表 10.4MAL 系统配置文件-配置项

配置项	配置含义
MAL_CHECK_INTERVAL	MAL 链路检测时间间隔，取值范围（0s-1800s），默认 30s，配置为 0 表示不进行 MAL 链路检测。为了防止误判，DMDSC 集群中，建议将配置值>= DCR_GRP_NETCHK_TIME。
MAL_CONN_FAIL_INTERVAL	判定 MAL 链路断开的的时间，取值范围（2s-1800s），默认 10s
MAL_LOGIN_TIMEOUT	MPP/DBLINK 等实例间登录时的超时检测间隔（3-1800），以秒为单位，默认 15s
MAL_BUF_SIZE	单个 MAL 缓存大小限制，以兆为单位。当此 MAL 的缓存邮件超过此大小，则会将邮件存储到文件中。有效值范围（0~500000），默认为 100
MAL_SYS_BUF_SIZE	MAL 系统总内存大小限制，单位：M。有效值范围（0~500000），默认为 0，表示 MAL 系统无总内存限制
MAL_VPOOL_SIZE	MAL 系统使用的内存初始化大小，以兆为单位。有效值范围（1~500000），默认为 128，此值一般要设置的比 MAL_BUF_SIZE 大一些
MAL_COMPRESS_LEVEL	MAL 消息压缩等级，取值范围（0-10）。默认为 0，不进行压缩；1-9 表示采用 zip 算法，从 1 到 9 表示压缩速度依次递减，压缩率依次递增；10 表示采用 snappy 算法，压缩速度高于 zip 算法，压缩率相对低

[MAL_NAME]	MAL 名称，同一个配置文件中 MAL 名称需保持唯一性
MAL_INST_NAME	数据库实例名，与 dm.ini 的 INSTANCE_NAME 配置项保持一致，MAL 系统中数据库实例名要保持唯一
MAL_HOST	MAL IP 地址，使用 MAL_HOST + MAL_PORT 创建 MAL 链路
MAL_PORT	MAL 监听端口
MAL_INST_HOST	MAL_INST_NAME 实例对外服务 IP 地址
MAL_INST_PORT	MAL_INST_NAME 实例对外服务端口，和 dm.ini 中的 PORT_NUM 保持一致
MAL_DW_PORT	MAL_INST_NAME 实例守护进程监听端口，其他守护进程或监视器使用 MAL_HOST + MAL_DW_PORT 创建 TCP 连接。

### 使用说明

1.DMASMSVR 组成的集群环境使用 MAL 系统进行通讯，需要在 dmdcr.ini 配置文件中配置 DMDCR\_MAL\_PATH 参数，指定 MAL 配置文件路径。例如：DMDCR\_MAL\_PATH=/home/data/dmasvrmal.ini。

2.使用 MAL 系统的 dmserver 实例，需要将 dm.ini 配置项 MAL\_INI 设置为 1。同时 MAL 系统配置文件名称必须为 dmmal.ini。

3.DMASMSVR 和 DMDSC 集群中的 dmserver 实例需要分别配置一套独立的 MAL 系统，两者配置的 MAL 环境不能冲突。

## 10.5 DM.INI

dm.ini 是 dmserver 使用的配置文件，详细介绍可以参考《DM8 系统管理员手册》相关章节。

各 DSC 节点之间的部分 INI 参数必须保持一致，如果不一致，会导致后启动的节点启动失败，日志会记录失败原因。每个节点用系统函数修改 INI 时，只会修改当前节点的值。须保持一致的 INI 参数如下所示：

```

BUFFER
DSC_N_CTL
TS_RESERVED_EXTENTS
TS_SAFE_FREE_EXTENTS

```

```
TS_MAX_ID
TS_FIL_MAX_ID
DIRECT_IO
PWD_POLICY
ALTER_MODE_STATUS
ENABLE_OFFLINE_TS
ORDER_BY_NULLS_FLAG
OPTIMIZER_MODE
CHECK_SVR_VERSION
RLOG_APPEND_SYSTAB_LOGIC
ISOLATION_LEVEL
HPC_N_CTLs
BACKSLASH_ESCAPE
STR_LIKE_IGNORE_MATCH_END_SPACE
CLOB_LIKE_MAX_LEN
MS_PARSE_PERMIT
COMPATIBLE_MODE
CASE_COMPATIBLE_MODE
COUNT_64BIT
CALC_AS_DECIMAL
CMP_AS_DECIMAL
CAST_VARCHAR_MODE
PL_SQLCODE_COMPATIBLE
LEGACY_SEQUENCE
DM6_TODATE_FMT
DROP_CASCADE_VIEW
```

另外，下面几个参数需要按要求进行配置：

■ **INSTANCE\_NAME**: dmserver 如果需要使用 DMASM 文件系统，**INSTANCE\_NAME** 必须和 **DMDCR\_CFG.INI** 里面配置的 **DCR\_EP\_NAME** 相同；

- **MAL\_INI**: 必须设置为 1;

- **DSC\_USE\_SBT**: 是否使用辅助的平衡二叉树。取值 0 或 1。1 是, 0 否。默认 0。

DM 的检查点机制要求 BUFFER 更新链表保持有序性, 所有被修改过的数据页, 要根据其第一次修改的 LSN 值 (**FIRST\_MODIFIED\_LSN**) 从小到大有序排列。而 DMDSC 的缓存交换机制要求数据页在节点间传递, 当一个更新页 P1 普通节点 EP0 传递到节点 EP1 时, 为了不破坏节点 EP1 更新链表 **FIRST\_MODIFIED\_LSN** 的有序性, 需要扫描更新链, 将 P1 加入到更新链中的合适位置。极端情况下, 可能需要扫描整个更新链, 才能找到 P1 的插入位置, 在 BUFFER 比较大, 更新链表比较长的情况下, 这种扫描、定位代价十分昂贵, 特别是在高并发情况下, 会引发严重的并发冲突, 影响并发性能。开启 **DSC\_USE\_SBT** 参数后, 系统内部维护一个平衡二叉树, 在将数据页加入更新链表时, 根据这个平衡二叉树进行  $\log_2^N$  次比较, 就可以找到插入位置;

- **DSC\_N\_POOLS**: LBS/GBS 池数目。有效值范围 (2~1024), 默认值 2。与 **BUFFER\_POOLS** 类似, **DSC\_N\_POOLS** 将系统中的 LBS/GBS 根据页号进行分组, 以降低 LBS/GBS 处理的并发冲突;

- **CONFIG\_PATH**: 指定 dmserver 所读取的配置文件 (**dmml.ini**, **dmarch.ini**, **dmtimer.ini** 等) 的路径。不允许指定 ASM 目录。缺省使用 **SYSTEM\_PATH** 路径。如果 **SYSTEM\_PATH** 保存在 ASM 上, 则直接报错;

- **TRACE\_PATH**: 存放系统 trace 文件的路径。不允许指定 ASM 目录。默认的 **TRACE\_PATH** 是 **SYSTEM\_PATH**; 如果 **SYSTEM\_PATH** 保存在 ASM 上, 则 **../config\_path/trace** 作为 **TRACE\_PATH**。



## 11 DMDSC 搭建

本章将举例说明 DMDSC 的搭建过程，并介绍搭建 DMDSC 的一些注意事项。

### 11.1 环境准备

硬件：两台相同配置机器，2G 内存，100G 本地磁盘，2 块网卡，另有一块共享磁盘 100G。

操作系统：RedHat Linux 64 位。

网络配置：eth0 网卡为 10.0.2.x 内网网段，两台机器分别为 10.0.2.101/10.0.2.102；eth1 为 192.168.56.x 外网网段，两台机器分别为 192.168.56.101/192.168.56.102。内网网段用于 MAL 通讯。

DM 各种工具位于目录：/opt/dmdbms/bin。

配置文件位于目录：/home/data。



真实的生产环境中，建议至少配置两块共享磁盘，分别用来存放联机日志文

说明：件和数据文件。

### 11.2 搭建 2 节点 DMDSC (DMASM)

#### 1. 在共享磁盘上裸设备划分

- 1) 输入 `fdisk /dev/sdb`
- 2) 依次输入 `n` → `p` → `1` → 回车 → `+100M` → 回车，完成第一块磁盘划分
- 3) 依次输入 `n` → `p` → `2` → 回车 → `+100M` → 回车，完成第二块磁盘划分
- 4) 依次输入 `n` → `p` → `3` → 回车 → `+2048M` → 回车，完成第三块磁盘划分
- 5) 依次输入 `n` → `p` → `4` → 回车 → 回车 → 回车，完成第四块磁盘划分
- 6) 编辑 `/etc/udev/rules.d/60-raw.rules` 文件，增加以下语句：

```
ACTION=="add", KERNEL=="sdb1", RUN+="/bin/raw /dev/raw/raw1 %N"

ACTION=="add", KERNEL=="sdb2", RUN+="/bin/raw /dev/raw/raw2 %N"

ACTION=="add", KERNEL=="sdb3", RUN+="/bin/raw /dev/raw/raw3 %N"
```

```
ACTION=="add", KERNEL=="sdb4", RUN+="/bin/raw /dev/raw/raw4 %N"

ACTION=="add", KERNEL=="raw[1-4]", OWNER="root", GROUP="root", MODE="660"
```

最后执行 `start_udev`，完成裸设备绑定

7) 可以通过 `blockdev --getsize64 /dev/raw/raw1` 命令查看裸设备大小

2. 准备 `dmdcr_cfg.ini` 配置文件，保存到 `/home/data/` 目录下面。后续 `DMASMCMD` 工具执行 `init` 语句会使用到。

```
DCR_N_GRP                = 3

DCR_VTD_PATH              = /dev/raw/raw2

DCR_OGUID                 = 63635

[GRP]

    DCR_GRP_TYPE           = CSS

    DCR_GRP_NAME           = GRP_CSS

    DCR_GRP_N_EP           = 2

    DCR_GRP_DSKCHK_CNT     = 60

[GRP_CSS]

    DCR_EP_NAME            = CSS0

    DCR_EP_HOST            = 10.0.2.101

    DCR_EP_PORT            = 9341

[GRP_CSS]

    DCR_EP_NAME            = CSS1

    DCR_EP_HOST            = 10.0.2.102

    DCR_EP_PORT            = 9343

[GRP]

    DCR_GRP_TYPE           = ASM

    DCR_GRP_NAME           = GRP_ASM

    DCR_GRP_N_EP           = 2

    DCR_GRP_DSKCHK_CNT     = 60

[GRP_ASM]

    DCR_EP_NAME            = ASM0
```

```
DCR_EP_SHM_KEY      = 93360

DCR_EP_SHM_SIZE     = 10

DCR_EP_HOST         = 10.0.2.101

DCR_EP_PORT         = 9349

DCR_EP_ASM_LOAD_PATH = /dev/raw
```

#### [GRP\_ASM]

```
DCR_EP_NAME         = ASM1

DCR_EP_SHM_KEY      = 93361

DCR_EP_SHM_SIZE     = 10

DCR_EP_HOST         = 10.0.2.102

DCR_EP_PORT         = 9351

DCR_EP_ASM_LOAD_PATH = /dev/raw
```

#### [GRP]

```
DCR_GRP_TYPE        = DB

DCR_GRP_NAME         = GRP_DSC

DCR_GRP_N_EP        = 2

DCR_GRP_DSKCHK_CNT   = 60
```

#### [GRP\_DSC]

```
DCR_EP_NAME         = DSC0

DCR_EP_SEQNO        = 0

DCR_EP_PORT         = 5236

DCR_CHECK_PORT      = 9741
```

#### [GRP\_DSC]

```
DCR_EP_NAME         = DSC1

DCR_EP_SEQNO        = 1

DCR_EP_PORT         = 5236

DCR_CHECK_PORT      = 9742
```

### 3. 使用 DMASMCMD 工具初始化

```
create dcrdisk '/dev/raw/raw1' 'dcr'
```

```
create votedisk '/dev/raw/raw2' 'vote'

create asmdisk '/dev/raw/raw3' 'LOG0'

create asmdisk '/dev/raw/raw4' 'DATA0'

init dcrdisk '/dev/raw/raw1' from '/home/data/dmdcr_cfg.ini' identified by 'abcd'

init votedisk '/dev/raw/raw2' from '/home/data/dmdcr_cfg.ini'
```

可以启动 dmasmcmd 工具，依次输入以上命令，或者将命令写入 asmcmd.txt 文件，执行 dmasmcmd script\_file=asmcmd.txt，只需在一台机器执行即可。

用户没有指定脚本文件，则 dmasmcmd 进入交互模式运行，逐条解析、运行命令；用户指定脚本文件（比如 asmcmd.txt），则以行为单位读取文件内容，并依次执行，执行完成以后，自动退出 dmasmcmd 工具。脚本文件必须以“#asm script file”开头，否则认为是无效脚本文件；脚本中其它行以“#”表示注释；脚本文件大小不超过 1M。

4. 准备 DMASM 的 MAL 配置文件（命名为 dmasvrmal.ini），使用 DMASM 的所有节点都要配置，内容完全一样，保存到 /home/data 目录下

```
[MAL_INST1]

MAL_INST_NAME      = ASM0

MAL_HOST            = 10.0.2.101

MAL_PORT            = 7236


[MAL_INST2]

MAL_INST_NAME      = ASM1

MAL_HOST            = 10.0.2.102

MAL_PORT            = 7237
```

5. 准备 dmdcr.ini 配置文件，保存到 /home/data 目录下面

DMASM 的两个节点分别配置 dmdcr.ini，dmdcr\_path 相同，dmasvrmal.ini 文件内容也相同，dmdcr\_sego 分别为 0 和 1。

#### 节点 10.0.2.101:

```
DMDCR_PATH          = /dev/raw/raw1

DMDCR_MAL_PATH       = /home/data/dmasvrmal.ini  #dmasmsvr 使用的 MAL 配置文件路径

DMDCR_SEQNO          = 0
```

```
#ASM 重启参数，命令行方式启动

DMDCR_ASM_RESTART_INTERVAL = 0

DMDCR_ASM_STARTUP_CMD = /opt/dmdbms/bin/dmasmsvr

dcr_ini=/home/data/dmdcr.ini


#DB 重启参数，命令行方式启动

DMDCR_DB_RESTART_INTERVAL = 0

DMDCR_DB_STARTUP_CMD = /opt/dmdbms/bin/dmserver

path=/home/data/dsc0_config/dm.ini dcr_ini=/home/data/dmdcr.ini
```

### 节点 10.0.2.102:

```
DMDCR_PATH      = /dev/raw/raw1

DMDCR_MAL_PATH  =/home/data/dmasvrmal.ini  #dmasmsvr 使用的 MAL 配置文件路径

DMDCR_SEQNO     = 1


#ASM 重启参数，命令行方式启动

DMDCR_ASM_RESTART_INTERVAL = 0

DMDCR_ASM_STARTUP_CMD = /opt/dmdbms/bin/dmasmsvr dcr_ini=/home/data/dmdcr.ini


#DB 重启参数，命令行方式启动

DMDCR_DB_RESTART_INTERVAL = 0

DMDCR_DB_STARTUP_CMD = /opt/dmdbms/bin/dmserver

path=/home/data/dsc1_config/dm.ini dcr_ini=/home/data/dmdcr.ini
```

## 6. 启动 DMCSS、DMASM 服务程序

在 10.0.2.101、10.0.2.102 节点先后分别启动 dmcass、dmasmsvr 程序。

手动启动 dmcass 命令：

```
[/opt/dmdbms/bin]# ./dmcass DCR_INI=/home/data/dmdcr.ini
```

手动启动 dmasmsvr 命令：

```
[/opt/dmdbms/bin]# ./dmasmsvr DCR_INI=/home/data/dmdcr.ini
```

如果 DMCSS 配置有自动拉起 dmasmsvr 的功能，可以等待 DMCSS 自动拉起 dmasmsvr

程序，不需要手动启动。

主 DMCSS 启动后屏幕打印如下：

```
[/opt/dmdbms/bin]# ./dmcss DCR_INI=/home/data/dmdcr.ini

dmcss V7.1.5.90-Build(2016.06.08-69758-debug)ENT

设置 CSS[0]为主 CSS

[ASM]: 设置 EP[0]为控制节点

[ASM]: 设置命令[START NOTIFY], 目标节点[0], 命令序号[2]

[ASM]: 设置命令[EP START], 目标节点[0], 命令序号[3]

[ASM]: 设置命令[NONE], 目标节点[0], 命令序号[0]

[ASM]: 设置命令[EP START], 目标节点[1], 命令序号[9]

[ASM]: 设置命令[NONE], 目标节点[1], 命令序号[0]

[ASM]: 设置命令[EP OPEN], 目标节点[0], 命令序号[12]

[ASM]: 设置命令[EP OPEN], 目标节点[1], 命令序号[13]

[ASM]: 设置命令[NONE], 目标节点[0], 命令序号[0]

[ASM]: 设置命令[NONE], 目标节点[1], 命令序号[0]

[ASM]: 设置命令[EP REAL OPEN], 目标节点[0], 命令序号[15]

[ASM]: 设置命令[EP REAL OPEN], 目标节点[1], 命令序号[16]

[ASM]: 设置命令[NONE], 目标节点[0], 命令序号[0]

[ASM]: 设置命令[NONE], 目标节点[1], 命令序号[0]
```

## 7. 使用 dmasmtool 工具创建 DMASM 磁盘组

选择一个节点（10.0.2.101），启动 dmasmtool 工具。

```
[/opt/dmdbms/bin]# ./dmasmtool DCR_INI=/home/data/dmdcr.ini
```

输入下列语句创建 DMASM 磁盘组：

```
#创建日志磁盘组

create diskgroup 'DMLOG' asmdisk '/dev/raw/raw3'

#创建数据磁盘组

create diskgroup 'DMDATA' asmdisk '/dev/raw/raw4'
```

## 8. 准备 dminit.ini 配置文件，保存到/home/data 目录

```
db_name                = dsc

system_path            = +DMDATA/data
```

```

system                = +DMDATA/data/dsc/system.dbf
system_size            = 128
roll                  = +DMDATA/data/dsc/roll.dbf
roll_size              = 128
main                  = +DMDATA/data/dsc/main.dbf
main_size              = 128
ctl_path               = +DMDATA/data/dsc/dm.ctl
ctl_size               = 8
log_size               = 256
dcr_path               = /dev/raw/raw1    #dcr 磁盘路径，目前不支持 asm，只能是裸设备
dcr_seqno              = 0
auto_overwrite         = 1

[DSC0] #inst_name 跟 dmdcr_cfg.ini 中 DB 类型 group 中 DCR_EP_NAME 对应
config_path            = /home/data/dsc0_config
port_num               = 5236
mal_host               = 10.0.2.101
mal_port               = 9340
log_path               = +DMLOG/log/dsc0_log01.log
log_path               = +DMLOG/log/dsc0_log02.log

[DSC1] #inst_name 跟 dmdcr_cfg.ini 中 DB 类型 group 中 DCR_EP_NAME 对应
config_path            = /home/data/dsc1_config
port_num               = 5237
mal_host               = 10.0.2.102
mal_port               = 9341
log_path               = +DMLOG/log/dsc1_log01.log
log_path               = +DMLOG/log/dsc1_log02.log

```

## 9. 使用 dminit 初始化 DB 环境

选择一个节点（10.0.2.101），启动 dminit 工具初始化数据库。dminit 执行完成后，会在 config\_path 目录（/home/data/dsc0\_config 和 /home/data/dsc1\_config）下生成配置文件 dm.ini 和 dmmal.ini。

```
./dminit control=/home/data/dminit.ini
```

屏幕打印如下：

```
[/opt/dmdbms/bin]# ./dminit control=/home/data/dminit.ini
initdb V7.1.5.90-Build(2016.06.08-69758-debug)ENT
db version: 0x70009
file dm.key not found, use default license!
License will expire on 2017-06-08
log file path: +DMLOG/log/dsc0_log01.log
log file path: +DMLOG/log/dsc0_log02.log
log file path: +DMLOG/log/dsc1_log01.log
log file path: +DMLOG/log/dsc1_log02.log
write to dir [+MDATA/data/dsc].
create dm database success. 2016-06-23 15:23:13
```

#### 10. 启动数据库服务器

将 10.0.2.101 机器/home/data/dsc1\_config 目录拷贝到 10.0.2.102 机器相同目录下，再分别启动 dmserver 即可完成 DMDSC 集群搭建。

如果 DMCSS 配置有自动拉起 dmserver 的功能，可以等待 DMCSS 自动拉起实例，不需要手动启动。

如果需要手动启动，可参考下面的操作步骤：

##### 10.0.2.101 机器：

```
./dmserver /home/data/dsc0_config/dm.ini dcr_ini=/home/data/dmdcr.ini
```

##### 10.0.2.102 机器：

```
./dmserver /home/data/dsc1_config/dm.ini dcr_ini=/home/data/dmdcr.ini
```

## 11.3 搭建 2 节点 DMDSC（裸设备）

### 1. 裸设备划分

- 1) 输入 fdisk /dev/sdb
- 2) 依次输入 n → p → 1 → 回车 → +100M → 回车，完成第一块磁盘划分
- 3) 依次输入 n → p → 2 → 回车 → +100M → 回车，完成第二块磁盘划分



- 4) 依次输入 `n` → `e` → `3` → 回车 → 回车, 完成逻辑磁盘划分
- 5) 依次输入 `n` → `1` → 回车 → `+2048M` → 回车, 完成逻辑磁盘划分
- 6) 重复 5 的步骤, 完成其他逻辑磁盘划分
- 7) 编辑 `/etc/udev/rules.d/60-raw.rules` 文件, 增加以下语句:

```
ACTION=="add", KERNEL=="sdb1", RUN+="/bin/raw /dev/raw/raw1 %N"
ACTION=="add", KERNEL=="sdb2", RUN+="/bin/raw /dev/raw/raw2 %N"
ACTION=="add", KERNEL=="sdb3", RUN+="/bin/raw /dev/raw/raw3 %N"
ACTION=="add", KERNEL=="sdb4", RUN+="/bin/raw /dev/raw/raw4 %N"
ACTION=="add", KERNEL=="raw[1-4]", OWNER="root", GROUP="root", MODE="660"
```

最后执行 `start_udev`, 完成裸设备绑定

- 8) 可以通过 `blockdev --getsize64 /dev/raw/raw1` 命令查看裸设备大小
- 9) 使用裸设备作为数据文件, 数据文件大小不能更改, 所以大小要提前规划好。

2. 准备 `dmdcr_cfg.ini` 配置文件放到 `/home/data` 目录下面, 后续 `DMASMCMD` 工具执行 `init` 语句会使用到。

仅使用裸设备, 不使用 `ASM` 文件系统, 不需要配置 `ASM` 信息, 仅配置 `CSS/DB` 信息。

```
DCR_N_GRP                = 2
DCR_VTD_PATH              = /dev/raw/raw2
DCR_OGUID                 = 63635

[GRP]

    DCR_GRP_TYPE          = CSS
    DCR_GRP_NAME          = GRP_CSS
    DCR_GRP_N_EP          = 2
    DCR_GRP_DSKCHK_CNT    = 60

[GRP_CSS]

    DCR_EP_NAME           = CSS0
    DCR_EP_HOST           = 10.0.2.101
    DCR_EP_PORT           = 9341

[GRP_CSS]

    DCR_EP_NAME           = CSS1
```

```
DCR_EP_HOST      = 10.0.2.102
```

```
DCR_EP_PORT      = 9343
```

```
[GRP]
```

```
DCR_GRP_TYPE     = DB
```

```
DCR_GRP_NAME     = GRP_DSC
```

```
DCR_GRP_N_EP     = 2
```

```
DCR_GRP_DSKCHK_CNT = 60
```

```
[GRP_DSC]
```

```
DCR_EP_NAME      = DSC0
```

```
DCR_EP_SEQNO     = 0
```

```
DCR_EP_PORT      = 8343
```

```
DCR_CHECK_PORT   = 9741
```

```
[GRP_DSC]
```

```
DCR_EP_NAME      = DSC1
```

```
DCR_EP_SEQNO     = 1
```

```
DCR_EP_PORT      = 8343
```

```
DCR_CHECK_PORT   = 9742
```

使用 DMASMCMD 工具初始化

```
create dcrdisk '/dev/raw/raw1' 'dcr'
```

```
create votedisk '/dev/raw/raw2' 'vote'
```

```
init dcrdisk '/dev/raw/raw1' from '/home/data/dmdcr_cfg.ini' identified by
'*****'
```

```
init votedisk '/dev/raw/raw2' from '/home/data/dmdcr_cfg.ini'
```

### 3. 准备 dminit.ini 配置文件，放置到/home/data 目录下

```
system_path = /home/data
```

```
db_name=dsc
```

```
main = /dev/raw/raw3
```

```
main_size = 128
```

```
roll = /dev/raw/raw4
```

```
roll_size = 128
```

```
system = /dev/raw/raw5
system_size = 128
ctl_path = /dev/raw/raw6
ctl_size = 8
dcr_path=/dev/raw/raw1
dcr_seqno=0
[dsc0]
config_path=/home/data/dsc0_config
port_num = 5236
mal_host = 10.0.2.101
mal_port = 9340
log_size = 256
log_path = /dev/raw/raw7
log_path = /dev/raw/raw8
[dsc1]
config_path=/home/data/dsc1_config
port_num = 5237
mal_host = 10.0.2.102
mal_port = 9341
log_size = 256
log_path = /dev/raw/raw9
log_path = /dev/raw/raw10
```

#### 4. 使用 dminit 初始化 DB 环境

```
./dminit control=/home/data/dminit.ini
```

屏幕打印如下：

```
[/opt/dmdbms/bin]# ./dminit control=/home/data/dminit.ini
initdb V7.1.5.90-Build(2016.06.08-69758-debug)ENT
db version: 0x70009
file dm.key not found, use default license!
License will expire on 2017-06-08
```

```
log file path: /dev/raw/raw7

log file path: /dev/raw/raw8

log file path: /dev/raw/raw9

log file path: /dev/raw/raw10

FILE "/home/data/dsc0_config/dm.ini" has already existed
FILE "/home/data/dsc1_config/dm.ini" has already existed
FILE "/dev/raw/raw5" has already existed
FILE "/dev/raw/raw4" has already existed
FILE "/dev/raw/raw3" has already existed
FILE "/dev/raw/raw6" has already existed
FILE "/dev/raw/raw7" has already existed
FILE "/dev/raw/raw8" has already existed
FILE "/dev/raw/raw9" has already existed
FILE "/dev/raw/raw10" has already existed

write to dir [/home/data/dsc].

create dm database success. 2016-06-23 15:43:32
```

## 5. 启动 DMCSS 服务程序

在两台机器分别启动 dmcss 程序，如果 dmdcr.ini 文件和执行程序在同一目录，可以不用指定参数，否则要指定 dmdcr.ini 文件的路径。

主 DMCSS 启动后屏幕打印如下：

```
[/opt/dmdbms/bin]# ./dmcss DCR_INI=/home/data/dmdcr.ini

dmcss V7.1.5.90-Build(2016.06.08-69758-debug)ENT

设置 CSS[0]为主 CSS

设置 CSS[0]为主 CSS

[DB]: 设置 EP[0]为控制节点
```

```
[DB]: 设置命令[START NOTIFY], 目标节点[0], 命令序号[2]
[DB]: 设置命令[EP START], 目标节点[0], 命令序号[3]
[DB]: 设置命令[NONE], 目标节点[0], 命令序号[0]
[DB]: 设置命令[EP START], 目标节点[1], 命令序号[5]
[DB]: 设置命令[NONE], 目标节点[1], 命令序号[0]
[DB]: 设置命令[DSC OPEN], 目标节点[0], 命令序号[8]
[DB]: 设置命令[DSC OPEN], 目标节点[1], 命令序号[9]
[DB]: 设置命令[NONE], 目标节点[0], 命令序号[0]
[DB]: 设置命令[NONE], 目标节点[1], 命令序号[0]
[DB]: 设置命令[DSC REAL OPEN], 目标节点[0], 命令序号[11]
[DB]: 设置命令[DSC REAL OPEN], 目标节点[1], 命令序号[12]
[DB]: 设置命令[NONE], 目标节点[0], 命令序号[0]
[DB]: 设置命令[NONE], 目标节点[1], 命令序号[0]
```

## 6. 启动数据库服务

将 101 机器/home/dsc/data/config02 目录拷贝到 102 机器相同目录下。分别启动 dmserver。

### 10.0.2.101 机器:

```
./dmserver /home/data/dsc0_config/dm.ini dcr_ini=/home/data/dmdcr.ini
```

### 10.0.2.102 机器:

```
./dmserver /home/data/dsc1_config/dm.ini dcr_ini=/home/data/dmdcr.ini
```

## 11.4 单节点搭建 DMDSC 测试环境

DMDSC 集群还支持在单节点上进行搭建, 用于测试、验证 DMDSC 集群的功能。

使用 DMASM 搭建单节点 DMDSC 集群的步骤与 11.2 节介绍的步骤基本一致, 有两点不同:

- 需要修改 dmdcr\_cfg.ini 中的 IP 信息;
- 要为每个模拟节点配置一个不同的 SHM\_KEY 值。

使用裸设备搭建单节点 DMDSC 集群的步骤与 11.3 节介绍的步骤基本一致, 只需要修改 dmdcr\_cfg.ini 中的 IP 信息即可。

## 12 故障自动重连

当用户连接到 DM 共享存储集群时，实际上是连接到集群中的一个实例，用户的所有增删改查操作都是由该实例完成的。但是如果该实例出现故障，那么用户连接会被转移到其他正常实例。而这种转移对用户是透明的，用户的增删改查继续返回正确结果，感觉不到异常。这种功能就是故障自动重连。

实现故障自动重连的前提条件是在配置 DM 共享存储集群的时候，必须配置连接服务名。

### 12.1 配置服务名（dm\_svc.conf）

配置 DMDSC 集群，一般要求配置连接服务名，以实现故障自动重连。连接服务名可以在 DM 提供的 JDBC、DPI 等接口中使用，连接数据库时指定连接服务名，接口会随机选择一个 IP 进行连接，如果连接不成功或者服务器状态不正确，则顺序获取下一个 IP 进行连接，直至连接成功或者遍历了所有 IP。

可以通过编辑 dm\_svc.conf 文件，配置连接服务名。dm\_svc.conf 配置文件在 DM 安装时生成，Windows 平台下位于 %SystemRoot%\system32 目录，Linux 平台下位于 /etc 目录。

连接服务名格式：

```
SERVERNAME=( IP[:PORT],IP[:PORT],.....)
```

dm\_svc.conf 文件常用配置项目说明：

#### ■ SERVERNAME

连接服务名，用户通过连接服务名访问数据库。

#### ■ IP

数据库所在的 IP 地址，如果是 IPv6 地址，为了区分端口，需要用 [ ] 封闭 IP 地址。

#### ■ PORT

数据库使用的 TCP 连接端口，可选配置，不配置则使用连接上指定的端口。

#### ■ SWITCH\_TIME

检测到数据库实例故障时，接口在服务器之间切换的次数；超过设置次数没有连接到有效数据库时，断开连接并报错。有效值范围 1~9223372036854775807，默认值为 3。

#### ■ SWITCH\_INTERVAL

表示在服务器之间切换的时间间隔，单位为毫秒，有效值范围 1~9223372036854775807，默认值为 200。

例如，配置一个名为 dmdsc\_svc 的连接服务名，使用 dmdsc\_svc 连接 DMDSC 集群中的数据库，即可实现故障自动重连。

```
dmdsc_svc=(10.0.2.101:5236,10.0.2.102:5237)
SWITCH_TIME=(10000)
SWITCH_INTERVAL=(1000)
```

关于 dm\_svc.conf 的详细设置，请参考《DM8 系统管理员手册》2.1.1.4 dm\_svc.conf 章节。

## 12.2 体验故障自动重连

基于 11.2 节成功架构的共享存储集群系统，下面用一个简单的例子来体验故障自动重连。

### 1. 连接到 dsc

```
Disql system/system@dmdsc_svc
```

### 2. 确认当前用户已经连接到的节点实例

```
SQL> select name from v$instance;
```

行号	NAME
1	DSC0

用户当前连接到节点 0 上 DSC0 实例。不要退出这个会话，第 4 步还是在这个会话中执行。

### 3. 关闭 DSC0 实例，或者将节点 0 所在的这台主机关机。

4. 等待几秒后，再次执行这条语句，还在会话 1 中执行。再次查询当前用户已经连接到的节点实例。

```
SQL> select name from v$instance;
```

行号	NAME
1	DSC1

可见，关闭节点 0，用户连接被自动转移到节点 1 的 DSC1 实例上，而用户并无察觉，实现了故障的自动转移。



## 13 动态扩展节点

DMDSC 集群支持动态扩展节点，每次扩展可以在原有基础上增加一个节点。

动态扩展节点要求当前 DMDSC 集群的所有节点都为 OK 状态，所有 dmserver 实例都处于 OPEN 状态，且可以正常访问。



注意：

扩展节点过程中，不应该有修改数据库状态或模式的操作

### 13.1 动态扩展节点流程

基于 11.2 节成功架构的共享存储集群系统，下面介绍如何动态扩展一个节点。

已搭建好的 DMDSC 集群实例名为 DSC0、DSC1，在此基础上扩展一个节点 DSC2。

#### 13.1.1 环境说明

新增节点环境为：

操作系统：RedHat Linux 64 位。

网络配置：eth0 网卡为 10.0.2.x 内网网段，该机器为 10.0.2.103；eth1 为 192.168.56.x 外网网段，该机器为 192.168.56.103。内网网段用于 MAL 通讯。

DM 各种工具位于目录：/opt/dmdbms/bin。

配置文件位于目录：/home/data。

表 13.1 配置环境说明

实例名	IP 地址	操作系统	备注
DSC0	192.168.56.101 10.0.2.101	RedHat Linux 64 位	192.168.56.101 外部服务 IP； 10.0.2.101 内部通信 IP
DSC1	192.168.56.102 10.0.2.102	RedHat Linux 64 位	192.168.56.102 外部服务 IP； 10.0.2.102 内部通信 IP
DSC2	192.168.56.103 10.0.2.103	RedHat Linux 64 位	192.168.56.103 外部服务 IP； 10.0.2.103 内部通信 IP

### 13.1.2 操作流程

1. 在 10.0.2.101 机器上使用 DMASMCMD 工具 export 出备份 dmdcr\_cfg\_bak.ini

```
[/opt/dmdbms/bin]# ./dmasmcmd
```

```
Asm> export dcrdisk '/dev/raw/raw1' to '/home/data/dmdcr_cfg_bak.ini'
```

2. 为新增节点准备日志文件

- 1) 使用 DISql 登录任意一个节点执行添加日志文件操作:

至少两个日志文件, 路径必须是 ASM 文件格式, 大小可以参考其他两个活动节点。

```
SQL>alter database add node logfile '+DMLOG/log/DSC2_log01.log' size 256,  
'+DMLOG/log/DSC2_log02.log' size 256;
```

2) 使用 dmctlcvt 工具将 dm.ctl 转换为文本文件 dmctl.txt, 查看 dmctl.txt, 新增节点的日志文件信息已经添加进 dm.ctl。

```
./dmctlcvt TYPE=1 SRC=+DMDATA/data/dsc/dm.ctl DEST=/home/data/dmctl.txt  
DCR_INI=/home/data/dmdcr.ini
```

- 3) 使用 dmasmtool 工具登录 ASM 文件系统, 也可以看到新增的节点日志文件

```
[/opt/dmdbms/bin]# ./dmasmtool DCR_INI=/home/data/dmdcr.ini  
ASM>ls +DMLOG/log
```

3. 为新增节点准备 config\_path

将 10.0.2.101 机器/home/data/dsc0\_config 目录拷贝到 10.0.2.103 机器相同目录下, 修改名字为/home/data/dsc2\_config。

修改 dsc2\_config 文件夹下的配置文件:

- 1) 修改 dm.ini

```
CONFIG_PATH      = /home/data/dsc2_config  
  
instance_name    = DSC2
```

- 2) 如果打开了归档参数, 修改 dmarch.ini

4. 新建 dmdcr.ini 配置文件, 保存到节点 10.0.2.103 的/home/data/目录下

注意设置 dmdcr\_sego 为 2, 修改 dm.ini 路径。

```
DMDCR_PATH       = /dev/raw/raw1  
  
DMDCR_MAL_PATH   =/home/data/dmasvrmal.ini #dmasmsvr 使用的 MAL 配置文件路径  
  
DMDCR_SEQNO      = 2
```

```
#ASM 重启参数，命令行方式启动

DMDCR_ASM_RESTART_INTERVAL = 0

DMDCR_ASM_STARTUP_CMD = /opt/dmdbms/bin/dmasmsvr dcr_ini=/home/data/dmdcr.ini


#DB 重启参数，命令行方式启动

DMDCR_DB_RESTART_INTERVAL = 0

DMDCR_DB_STARTUP_CMD = /opt/dmdbms/bin/dmserver
path=/home/data/dsc2_config/dm.ini dcr_ini=/home/data/dmdcr.ini
```

## 5. 修改当前环境的 MAL 配置文件

直接修改当前环境的 dmasvrml.ini 文件，添加新增节点信息，使用 DMASM 的所有节点都要配置，内容完全一样，并且将新增信息后的 dmasvrml.ini 文件拷贝到节点 10.0.2.103 的 /home/data 目录下。

```
[MAL_INST3]

MAL_INST_NAME = ASM2

MAL_HOST = 10.0.2.103

MAL_PORT = 7238
```

直接修改 dmserver 三个实例的 dmmal.ini，添加新增节点信息，所有节点都要配置相同内容，保存到各自的 dsc\_config 目录下。

```
[mal_inst2]

mal_inst_name = DSC2

mal_host = 10.0.2.103

mal_port = 9342
```

后续实例会重新读 MAL 配置文件，更新内存信息。

## 6. 修改 dmdcr\_cfg\_bak.ini，添加新增节点信息，CSS/ASMSVR/DB 都要配置

所有组信息修改：

```
DCR_GRP_N_EP = 3

DCR_GRP_EP_ARR = {0,1,2}
```

每个组增加一个节点信息，注意 DCR\_EP\_SHM\_KEY、端口号不能冲突；各组信息要放在各自的后面，即 [GRP\_CSS] 中 CSS2 放在 CSS1 后面，[GRP\_ASM] 中 ASM2 放在 ASM1 后面，DSC2 放在 DSC1 后面。

```

[GRP_CSS]

DCR_EP_NAME =    CSS2

DCR_EP_HOST   =    10.0.2.103

DCR_EP_PORT   =    9500


[GRP_ASM]

DCR_EP_NAME =    ASM2

DCR_EP_SHM_KEY =    93362

DCR_EP_SHM_SIZE =    10

DCR_EP_HOST =    10.0.2.103

DCR_EP_PORT =    9501

DCR_EP_ASM_LOAD_PATH    =    /dev/raw


[GRP_DSC]

DCR_EP_NAME      = DSC2

DCR_EP_SEQNO     = 2

DCR_EP_PORT      = 5236

DCR_CHECK_PORT   = 9502

```

7. 使用 DMSMCMC 工具将新增节点信息写回磁盘，新增节点作为 **error** 节点

```

[/opt/dmdbms/bin]# ./dmsmcmd

Asm> extend dcrdisk '/dev/raw/raw1' from '/home/data/dmdcr_cfg_bak.ini'

```

8. 在 dmcsm 控制台执行扩展节点命令

```

extend node

```

程序会通知所有实例(CSS/ASMSVR/dmserver)更新信息，在 CSS 控制台执行 SHOW 命令，能看到新增节点信息，ASMSVR/dmserver 是 **error** 节点，程序会通知 ASMSVR/dmserver 更新 MAL 信息。

9. 启动新的 DMCSS、DMSM 服务程序

在 10.0.2.103 节点启动 dmcsm、dmsmsvr 程序。

手动启动新的 dmcsm，dcr\_ini 指向新的 dmdcr.ini 文件：

```

[/opt/dmdbms/bin]# ./dmcsm DCR_INI=/home/data/dmdcr.ini

```

手动启动新的 dmasmsvr, dcr\_ini 指向新的 dmdcr.ini 文件, asmsvr 启动故障重加入流程:

```
[/opt/dmdbms/bin]# ./dmasmsvr DCR_INI=/home/data/dmdcr.ini
```

如果 DMCSS 配置有自动拉起 dmasmsvr 的功能,可以等待 DMCSS 自动拉起 dmasmsvr 程序,不需要手动启动。

## 10. 启动新的数据库服务器

如果 DMCSS 配置有自动拉起 dmserver 的功能,可以等待 DMCSS 自动拉起实例,不需要手动启动。

如果需要手动启动,可参考下面的操作步骤:

### 10.0.2.103 机器:

```
./dmserver /home/data/dsc2_config/dm.ini dcr_ini=/home/data/dmdcr.ini
```

## 11. 配置 dmcsm.ini

如果配置有监视器,则直接修改 dmcsm.ini,增加新扩节点 DMCSS 的 IP:PORT 配置项 CSSM\_CSS\_IP,并重启 dmcsm。

如果没有配置,可参考下面的操作步骤:

以搭建的 3 节点 DSC 环境为基础,配置对应的监视器,监视器放在第三方机器上,为 Linux 操作系统, dmcsm.ini 配置文件路径为 /home/data/cssm/, 可根据实际情况调整配置环境及路径。

### 1) 配置 dmcsm.ini 文件

#和 dmdcr\_cfg.ini 中的 DCR\_OGUID 保持一致

```
CSSM_OGUID = 63635
```

#配置所有 CSS 的连接信息,

#和 dmdcr\_cfg.ini 中 CSS 配置项的 DCR\_EP\_HOST 和 DCR\_EP\_PORT 保持一致

```
CSSM_CSS_IP = 10.0.2.101:9341
```

```
CSSM_CSS_IP = 10.0.2.102:9343
```

```
CSSM_CSS_IP = 10.0.2.103:9500
```

```
CSSM_LOG_PATH = /home/data/cssm #监视器日志文件存放路径
```

```
CSSM_LOG_FILE_SIZE = 32 #每个日志文件最大 32M
```

```
CSSM_LOG_SPACE_LIMIT    =    0           #不限定日志文件总占用空间
```

## 2) 启动 dmcsm 监视器

```
./dmcsm INI_PATH=/home/data/cssm/dmcsm.ini
```



如果由于配置文件错误，动态扩展节点失败，只能停掉所有实例，重新

**注意：** `init dcr` 磁盘，不影响 `dmserver` 数据

### 13.1.3 注意事项

1. 扩展节点前由用户保证所有 `dmcsm/dmasmsvr/dmserver` 节点都是 OK 的，且都是活动的；
2. 每次扩展节点只能扩一个节点，扩展完成后可以继续扩展节点；
3. 扩展节点的过程中不能出现修改实例状态或模式的操作；
4. 扩展节点的过程中，如果发生 `dmcsm/dmasmsvr/dmserver` 实例故障，会导致扩展失败；
5. 扩展过程中操作失误(比如未修改 `dmmal.ini`、`asmsvrml.ini`，未增加日志文件)，会导致扩展失败；
6. 执行完 `extend node` 命令，用户需要查看 `log` 文件，确认扩展操作是否成功；
7. 扩展失败可能会导致集群环境异常，需要退出所有 `dmcsm/dmasmsvr/dmserver`，重新 `init dcr` 磁盘。

## 14 监控 DMDSC

DMDSC 集群的运行情况可以通过 DMCSSM 监视器进行查看，也可以查询 DMDSC 相关的动态视图获取更详细的信息。DMCSSM 监视器支持一些控制命令，可以用来启动、关闭 DMDSC 集群，还可以进行手动控制节点故障处理和节点重加入。

### 14.1 DMCSSM 监视器

#### 14.1.1 功能说明

##### 1. 监控集群状态

DMCSS 每秒会发送集群中所有节点的状态信息、当前连接到 DMCSS 的监视器信息以及 DCR 的配置信息到活动的监视器上，监视器提供对应的 show 命令用于查看各类信息。

##### 2. 打开/关闭指定组的自动拉起

DMCSSM 提供 SET AUTO RESTART ON/SET AUTO RESTART OFF 命令，通知 DMCSS 打开或关闭对指定组的自动拉起功能，此功能和 DMCSS 的监控打开或关闭没有关系。

##### 3. 强制 OPEN 指定组

DMCSSM 提供 OPEN FORCE 命令，在启动 ASM 或 DB 组时，如果组中某个节点发生硬件故障等原因导致一直无法启动，可执行此命令通知 DMCSS 将 ASM 组或 DB 组强制 OPEN，不再等待故障节点启动成功。

##### 4. 启动/退出集群

DMCSSM 提供 EP STARTUP/EP STOP 命令，可以通知 DMCSS 启动/退出指定的 ASM 或 DB 组。

##### 5. 集群故障处理

DMCSSM 提供 EP BREAK/EP RECOVER 命令，在主 CSS 的监控功能被关闭的情况下，可以通过执行这些命令手动进行故障处理和故障恢复。另外在某些特殊场景下还可通过 EP HALT 命令强制退出指定节点，具体可参考 14.1.4 小节。

## 14.1.2 配置文件

DMCSSM 的配置文件名称为 `dmcssm.ini`，所支持的配置项说明如下。

表 14.1 `dmcssm.ini` 配置项

配置项	配置含义
CSSM_OGUID	用于和 DMCSS 通信校验使用，和 <code>dmdcr_cfg.ini</code> 中的 <code>DCR_OGUID</code> 值保持一致。
CSSM_CSS_IP	集群中所有 DMCSS 所在机器的 IP 地址，以及 DMCSS 的监听端口，配置格式为“IP:PORT”的形式，其中 IP 和 PORT 分别对应 <code>dmdcr_cfg.ini</code> 中 DMCSS 节点的 <code>DCR_EP_HOST</code> 和 <code>DCR_EP_PORT</code> 。
CSSM_LOG_PATH	日志文件路径，日志文件命名方式为“ <code>dmcssm_年月日时分秒.log</code> ”，例如“ <code>dmcssm_20160614131123.log</code> ”。 其他请参考附加说明。
CSSM_LOG_FILE_SIZE	单个日志文件大小，范围 16~2048，单位为 M，默认值为 64M，达到最大值后，会自动生成并切换到新的日志文件中。
CSSM_LOG_SPACE_LIMIT	日志总空间大小，取值 0 或者 256~4096，单位为 M，默认值为 0，表示没有空间限制，如果达到设定的总空间限制，会自动删除创建时间最早的日志文件。

### CSSM\_LOG\_PATH 配置说明：

如果 `dmcssm.ini` 中配置有 `CSSM_LOG_PATH` 路径，则将 `CSSM_LOG_PATH` 作为日志文件路径，如果没有配置，则将 `dmcssm.ini` 配置文件所在的路径作为日志文件路径。

在有日志写入操作时，如果日志路径下没有日志文件，会自动创建一个新的日志文件，如果已经有日志文件，则根据设定的单个日志文件大小



(`CSSM_LOG_FILE_SIZE`) 决定继续写入已有的日志文件或者创建新的日志

说明：文件写入。

创建新的日志文件时，根据设定的日志总空间大小

(`CSSM_LOG_SPACE_LIMIT`) 决定是否删除创建时间最早的日志文件。



14.1.3 配置步骤

同一个 DMDSC 集群中，允许最多同时启动 10 个监视器，建议监视器放在独立的第三方机器上，避免由于节点间网络不稳定等原因导致监视器误判节点故障。

下面举例说明监视器的配置步骤，以 11.2 小节搭建的 2 节点 DSC 环境为基础，配置对应的监视器，监视器放在第三方机器上，为 windows 操作系统，dmcssm.ini 配置文件路径为 D:\cssm，可根据实际情况调整配置环境及路径。

1. 配置 dmcssm.ini 文件

```
#和 dmdcr_cfg.ini 中的 DCR_OGUID 保持一致
CSSM_OGUID = 63635

#配置所有 CSS 的连接信息，
#和 dmdcr_cfg.ini 中 CSS 配置项的 DCR_EP_HOST 和 DCR_EP_PORT 保持一致
CSSM_CSS_IP = 10.0.2.101:9341
CSSM_CSS_IP = 10.0.2.102:9343

CSSM_LOG_PATH =D:\cssm\log      #监视器日志文件存放路径
CSSM_LOG_FILE_SIZE = 32         #每个日志文件最大 32M
CSSM_LOG_SPACE_LIMIT = 0       #不限定日志文件总占用空间
```

2. 启动 dmcssm 监视器

```
dmcssm.exe INI_PATH=D:\cssm\dmcssm.ini
```

14.1.4 命令说明

监视器提供一系列命令，支持集群的状态信息查看以及节点的故障处理，可输入 help 命令，查看命令使用说明。

表 14.2 监视器命令

命令名称	含义
help	显示帮助信息
show [group_name]	显示指定的组信息，如果没有指定 group_name，则

	显示所有组信息
show config	显示 dmdcr_cfg.ini 的配置信息
show monitor	显示当前连接到主 CSS 的所有监视器信息
set group_name auto restart on	打开指定组的自动拉起功能（只修改 dmcss 内存值）
set group_name auto restart off	关闭指定组的自动拉起功能（只修改 dmcss 内存值）
open force group_name	强制 open 指定的 ASM 或 DB 组
ep startup group_name	启动指定的 ASM 或 DB 组
ep stop group_name	退出指定的 ASM 或 DB 组
ep halt group_name.ep_name	强制退出指定组中的指定节点
extend node	联机扩展节点
ep crash group_name.ep_name	手动指定节点故障
check crash over group_name	检查指定组故障处理是否真正结束
exit	退出监视器

#### 命令使用说明：

##### 1. help

显示帮助信息。

##### 2. show [group\_name]

显示指定的组信息，如果没有指定 group\_name，则显示所有组信息。

返回的组信息优先从主 CSS 获取，如果主 CSS 故障或尚未选出，则任选一个从 CSS 返回组信息。

##### 3. show config

显示 dmdcr\_cfg.ini 的配置信息，对于 DB 类型的节点，会比 CSS/ASM 节点多一项 DCR\_EP\_SEQNO 的显示值，如果原本的 ini 文件中没有手动配置，则显示的是自动分配的序列值。

返回的配置信息优先从主 CSS 获取，如果主 CSS 故障或尚未选出，则任选一个从 CSS 返回信息。

##### 4. show monitor

显示当前连接到主 CSS 的所有监视器信息，如果主 CSS 故障或尚未选出，则任选一个从 CSS 显示连接信息。

返回的信息中，第一行为当前执行命令的监视器的连接信息。

在数据守护环境中，守护监视器 dmmonitor 的部分命令（启动/停止/强杀实例）最终是由 dmcss 执行的，守护进程 dmwatcher 在命令执行中充当了 dmcssm 的角色，守护进程通知 dmcss 执行完成后，再将执行结果返回给守护监视器。因此 dmcss 上会有 dmwatcher 的连接信息，show monitor 命令也会显示 dmwatcher 的连接信息，可以根据 from\_name 字段值进行区分。

#### 5. set group\_name auto restart on

打开指定组的自动拉起功能。

可借助 show css 命令查看每个节点的自动拉起标记，每个 css 只能控制和自己的 dmdcr.ini 中配置的 DMDCR\_SEQNO 相同节点的自动拉起。

#### 6. set group\_name auto restart off

关闭指定组的自动拉起功能。

可借助 show css 命令查看每个节点的自动拉起标记，每个 css 只能控制和自己的 dmdcr.ini 中配置的 DMDCR\_SEQNO 相同节点的自动拉起。

#### 7. open force group\_name

在启动 ASM 或 DB 组时，如果某个节点故障一直无法启动，可借助此命令将 ASM 或 DB 组强制 OPEN。

此命令需要发送到主 CSS 执行，并且主 CSS 的监控需要处于打开状态，如果主 CSS 故障或尚未选出，则命令执行失败。

#### 8. ep startup group\_name

通知 CSS 启动指定的 ASM 或 DB 组，如果 CSS 已经打开了指定组的自动拉起功能，则命令不允许执行，需要等待 CSS 自动检测故障并执行拉起操作。

每个 CSS 只负责拉起和自己的 dmdcr.ini 中配置的 DMDCR\_SEQNO 相同的 ASM 或 DB 节点，因此需要所有 CSS 都处于活动状态，否则只通知当前活动的 CSS 自动拉起相对应的节点。



说明:

只有在 ASM 组正常启动到 OPEN 状态,并且所有活动的 ASM 节点都处于 OPEN 状态时,才允许启动 DB 组,否则执行 DB 组的启动命令会报错,CSS 自动拉起 DB 组时也需要满足此条件。

在命令执行前,如果 CSS 对指定组的自动拉起功能是关闭的,在节点拉起成功后,会打开对指定组的自动拉起功能。

## 9. ep stop group\_name

退出指定的 ASM 或 DB 组,如果主 CSS 故障或尚未选出,则命令执行失败。

在退出 ASM 组时,需要保证 DB 组已经退出,否则会报错处理。



注意:

在命令执行前,如果 CSS 对指定组的自动拉起功能是打开的,则会先通知 CSS 关闭对指定组的自动拉起功能,再通知指定组退出,避免命令执行成功后节点再次被自动拉起。

## 10. ep halt group\_name.ep\_name

强制退出指定组的指定 EP。适用于下述场景:

■ 一某个 ASM 或 DB 节点故障,CSS 的心跳容错时间 DCR\_GRP\_DSKCHK\_CNT 配置值很大,在容错时间内,CSS 不会调整故障节点的 active 标记,一直是 TRUE,CSS 认为故障 EP 仍然处于活动状态,不会自动执行故障处理,并且不允许手动执行故障处理。

另外,执行 EP STARTUP 或 EP STOP 命令时,会误认为故障 EP 仍然处于活动状态,导致执行结果与预期不符。此时可以通过执行 EP HALT 命令,通知 CSS 再次 HALT 故障 EP,确认 EP 已经被 HALT 后,CSS 会及时调整 active 标记为 FALSE,在此之后,对自动/手动故障处理,EP STARTUP/EP STOP 命令都可以正常执行。

■ 二需要强制 HALT 某个正在运行的 ASM 或 DB 节点,也可以通过此命令完成。

## 11. extend node

DMDSC 集群联机扩展节点时使用。

程序会通知所有实例(CSS/ASMSVR/dmserver)更新 dcr 信息。

使用 show 命令能看到新增节点信息,新增 ASMSVR/dmserver 为 ERROR 状态。

## 12. ep crash group\_name.ep\_name

手动指定节点故障,节点故障后,CSS 只有在 DCR\_GRP\_DSKCHK\_CNT 配置的时间过后才会判定实例故障,开始故障处理流程。用户如果明确知道实例已经故障,可以收到执行此命令,CSS 可以立即开始故障处理流程。

### 13. check crash over group\_name

DB 集群环境故障处理后，需要满足一定条件（控制节点重做故障节点日志产生的数据页修改都已经刷盘完成），才允许故障节点重新加回集群环境。此命令用来显示 DB 集群环境故障处理是否真正结束。

### 14. exit

退出监视器。

## 14.2 动态视图

DMDSC 集群提供一系列动态视图来查看当前的系统运行信息。部分视图是全局的，任意节点登录查询的结果都是相同的，部分视图仅显示登录节点的信息。

### 14.2.1 V\$DSC\_EP\_INFO

显示实例信息，登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	EP_NAME	VARCHAR(128)	实例名称
2	EP_SEQNO	INTEGER	DSC 节点序号
3	EP_GUID	BIGINT	EP 唯一标识码
4	EP_TIMESTAMP	BIGINT	EP 时间戳
5	EP_MODE	VARCHAR(32)	EP 模式，CONTROL NODE 或 NORMAL NODE
6	EP_STATUS	VARCHAR(32)	EP 状态，OK 或 ERROR

### 14.2.2 V\$DSC\_GBS\_POOL

显示 GBS 控制结构的信息，仅显示登录节点的信息。

序号	列	数据类型	说明
1	N_CTL	INTEGER	GBS 控制块总数
2	N_FREE_CTL	INTEGER	空闲的 GBS 控制块数目
3	N_SUB_POOL	INTEGER	GBS_POOL 个数

### 14.2.3 V\$DSC\_GBS\_POOLS\_DETAIL

显示分片的 GBS\_POOL 详细信息，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	N_USED_CTL	INTEGER	正在使用的 GBS 控制块数目
3	N_REQUEST	INTEGER	正在等待 GBS 控制块的请求数目
4	N_FREE_REQUEST	INTEGER	空闲的 GBS 请求控制块数目

### 14.2.4 V\$DSC\_GBS\_CTL

显示 GBS 控制块信息。多个 pool，依次扫描，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	N_REVOKE_X_ONLY	INTEGER	页的优化次数

### 14.2.5 V\$DSC\_GBS\_CTL\_DETAIL

显示 GBS 控制块详细信息。多个 pool，依次扫描，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	TYPE	VARCHAR(32)	详细信息类型 (OWNER/REQUEST/REVOKING)
14	MODE	INTEGER	封锁模式, 0/1/2/4: N_LATCH/X_LATCH/S_LATCH/F_LATCH
15	EP_SEQNO	INTEGER	拥有、请求、或者回收封锁的 EP
16	REAL_FLUSH	CHAR	是否真正执行刷盘请求 ('Y'/'N')
17	N_REVOKE_X_ONLY	INTEGER	页的优化次数

#### 14.2.6 V\$DSC\_GBS\_CTL\_LRU\_FIRST

显示 GBS 控制块 LRU 链表首页信息。多个 pool, 依次扫描, 仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号

5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	N_REVOKE_X_ONLY	INTEGER	页的优化次数

#### 14.2.7 V\$DSC\_GBS\_CTL\_LRU\_FIRST\_DETAIL

显示 GBS 控制块 LRU 链表首页详细信息。多个 pool，依次扫描，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	TYPE	VARCHAR(32)	详细信息类型 (OWNER/REQUEST/REVOKING)



14	MODE	INTEGER	封锁模式，0/1/2/4：  N_LATCH/X_LATCH/S_LATCH/F_LATCH
15	EP_SEQNO	INTEGER	拥有、请求、或者回收封锁的 EP
16	REAL_FLUSH	CHAR	是否真正执行刷盘请求（'Y'/'N'）
17	N_REVOKE_X_ONLY	INTEGER	页的优化次数

#### 14.2.8 V\$DSC\_GBS\_CTL\_LRU\_LAST

显示 GBS 控制块 LRU 链表尾页信息。多个 pool，依次扫描，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	N_REVOKE_X_ONLY	INTEGER	页的优化次数

#### 14.2.9 V\$DSC\_GBS\_CTL\_LRU\_LAST\_DETAIL

显示 GBS 控制块 LRU 链表尾页详细信息。多个 pool，依次扫描，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号

2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	TYPE	VARCHAR(32)	详细信息类型 (OWNER/REQUEST/REVOKING)
14	MODE	INTEGER	封锁模式，0/1/2/4： N_LATCH/X_LATCH/S_LATCH/F_LATCH
15	EP_SEQNO	INTEGER	拥有、请求、或者回收封锁的 EP
16	REAL_FLUSH	CHAR	是否真正执行刷盘请求（'Y'/'N'）
17	N_REVOKE_X_ONLY	INTEGER	页的优化次数

#### 14.2.10 V\$DSC\_GBS\_REQUEST\_CTL

显示等待 GBS 控制块的请求信息。多个 pool，依次扫描，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	MODE	INTEGER	封锁模式，0/1/2/4： N_LATCH/X_LATCH/S_LATCH/F_LATCH

6	EP_SEQNO	INTEGER	请求封锁的 EP
---	----------	---------	----------

### 14.2.11 V\$DSC\_LBS\_POOL

显示 LBS 控制结构的信息，仅显示登录节点的信息。

序号	列	数据类型	说明
1	N_CTL	INTEGER	LBS 控制块总数
2	N_FREE_CTL	INTEGER	空闲的 LBS 控制块数目
3	N_SUB_POOL	INTEGER	LBS_POOL 个数

### 14.2.12 V\$DSC\_LBS\_POOLS\_DETAIL

显示分片的 LBS\_POOL 详细信息。多个 pool，依次扫描，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	N_USED_CTL	INTEGER	正在使用的 LBS 控制块数目
3	N_REQUEST_SPACE	INTEGER	正在等待 LBS 控制块的请求数目
4	N_FREE_REQUEST	INTEGER	空闲的 LBS 请求控制块数目

### 14.2.13 V\$DSC\_LBS\_CTL

显示 LBS 控制块信息。多个 pool，依次扫描，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数

9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用计数
11	MODE	INTEGER	获得 GBS 授权的封锁模式
12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数
14	N_REVOKE_X_ONLY	INTEGER	页的优化次数

#### 14.2.14 V\$DSC\_LBS\_CTL\_LRU\_FIRST

显示 LBS 的 LRU\_FIRST 控制块信息。多个 pool, 依次扫描, 仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用计数
11	MODE	INTEGER	获得 GBS 授权的封锁模式
12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数
14	N_REVOKE_X_ONLY	INTEGER	页的优化次数

#### 14.2.15 V\$DSC\_LBS\_CTL\_LRU\_LAST

显示 LBS 的 LRU\_LAST 控制块信息。多个 pool, 依次扫描, 仅显示登录节点的信息。

序号	列	数据类型	说明
----	---	------	----

1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用计数
11	MODE	INTEGER	获得 GBS 授权的封锁模式
12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数

#### 14.2.16 V\$DSC\_LBS\_CTL\_DETAIL

显示 LBS 控制块详细信息。多个 pool，依次扫描，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用计数
11	MODE	INTEGER	获得 GBS 授权的封锁模式

12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数
14	REQUEST_MODE	INTEGER	请求的封锁模式
15	REVOKE_LSN	BIGINT	回收权限时，GBS 上的最新 LSN 值

### 14.2.17 V\$DSC\_LBS\_CTL\_LRU\_FIRST\_DETAIL

显示 LBS 的 LRU\_FIRST 控制块详细信息。多个 pool，依次扫描，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用计数
11	MODE	INTEGER	获得 GBS 授权的封锁模式
12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数
14	REQUEST_MODE	INTEGER	请求的封锁模式
15	REVOKE_LSN	BIGINT	回收权限时，GBS 上的最新 LSN 值

### 14.2.18 V\$DSC\_LBS\_CTL\_LRU\_LAST\_DETAIL

显示 LBS 的 LRU\_LAST 控制块详细信息。多个 pool，依次扫描，仅显示登录节点的信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用计数
11	MODE	INTEGER	获得 GBS 授权的封锁模式
12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数
14	REQUEST_MODE	INTEGER	请求的封锁模式
15	REVOKE_LSN	BIGINT	回收权限时，GBS 上的最新 LSN 值

### 14.2.19 V\$DSC\_GTV\_SYS

显示 GTV 控制结构的信息，仅登录集群环境控制节点才能获取数据，登录其他节点返回数据无效。

序号	列	数据类型	说明
1	T_INFO_NUM	INTEGER	系统已提交、未 PURGE 事务所修改的表对象个数
2	NEXT_TRXID	BIGINT	下一个事务 ID
3	MAX_PURGABLE_TRXID	BIGINT	最大可 PURGE 的事务 ID
4	UNDO_TRXID	BIGINT	回滚段中，正在被访问的最小事务 ID
5	UNDO_CNT	INTEGER	UNDO_TRXID 被设置的次数

## 14.2.20 V\$DSC\_GTV\_TINFO

显示 TINFO 控制结构的信息，仅登录集群环境控制节点才能获取数据，登录其他节点返回数据为空。暂时没有什么用，查询结果为空。

序号	列	数据类型	说明
1	TABLE_ID	INTEGER	系统已提交、未 PURGE 事务所修改的表 ID
2	TRX_ID	BIGINT	修改操作的事务 ID
3	ROWCNT	BIGINT	修改操作影响的行数

## 14.2.21 V\$DSC\_GTV\_ACTIVE\_TRX

显示全局活动事务信息，仅登录集群环境控制节点才能获取数据，登录其他节点返回数据为空。暂时没有什么用，查询结果为空。

序号	列	数据类型	说明
1	EP_SEQNO	INTEGER	事务所在的节点号
2	TRX_ID	BIGINT	事务 ID
3	NEXT_TRXID	BIGINT	下一个事务 ID
4	MIN_ACTIVE_TRXID	BIGINT	活动事务链表中的最小事务 ID
5	REFED_TRXID	BIGINT	活动事务链表中的事务 ID

## 14.2.22 V\$DSC\_LOCK

显示全局活动的事务锁信息，登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	EP_SEQNO	INTEGER	拥有该锁的节点号
2	ADDR	BIGINT	锁地址
3	TRX_ID	BIGINT	所属事务 ID
4	LTYPE	VARCHAR(10)	锁类型：TID 锁、对象锁
5	LMODE	CHAR(2)	锁模式：S 锁、X 锁、IX 锁、IS 锁



6	BLOCKED	INTEGER	是否阻塞
7	TABLE_ID	INTEGER	对应表锁、字典对象 ID
8	ROW_IDX	BIGINT	被封锁事务 ID

### 14.2.23 V\$DSC\_TRX

显示所有活动事务的信息。通过该视图可以查看所有系统中所有的事务以及相关信息，如锁信息等，登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	EP_SEQNO	INTEGER	事务所在节点号
2	ID	BIGINT	当前活动事务的 ID 号
3	NEXTID	BIGINT	下一个事务 ID 号
4	MIN_ACTIVE_ID	BIGINT	所有活动事务 ID 号最小者
5	STATUS	VARCHAR(20)	当前事务的状态
6	ISOLATION	INTEGER	事务的隔离级别  0: 读未提交  1: 读提交  2: 可重复读  3: 串行化
7	READ_ONLY	CHAR(1)	是否为只读事务 (Y/N)
8	SESS_ID	BIGINT	当前事务所在的会话 ID
9	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
10	INS_CNT	INTEGER	插入回滚记录个数
11	DEL_CNT	INTEGER	删除回滚记录个数
12	UPD_CNT	INTEGER	更新回滚记录个数
13	UPD_INS_CNT	INTEGER	更新插入回滚记录个数
14	UREC_SEQNO	INTEGER	当前 Undo 记录的递增序列号
15	WAITING	BIGINT	事务等待的锁

## 14.2.24 V\$DSC\_TRXWAIT

显示事务等待信息，登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	EP_SEQNO	INTEGER	事务所在节点号
2	ID	BIGINT	事务 ID
3	WAIT_FOR_ID	BIGINT	所等待的事务 ID
4	WAIT_SEQNO	TINYINT	等待的事务序列号
5	WAIT_TIME	INTEGER	当前已等待时间 (s)

## 14.2.25 V\$DSC\_TRX\_VIEW

显示当前事务可见的所有活动事务视图信息。根据达梦多版本规则，通过该视图可以查询系统中自己所见的事务信息；可以通过与 v\$dsc\_trx 表的连接查询它所见事务的具体信息，登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	SELF_ID	BIGINT	活动事务 ID
2	EP_SEQNO	INTEGER	可见事务所在节点号
3	ACTIVE_ID	BIGINT	所见的事务活动事务 ID

## 14.2.26 V\$ASMATTR

如果使用有 ASM 文件系统，可通过此视图查看 ASM 文件系统相关属性，登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	AU_SIZE	INTEGER	单个 AU 大小，单位为字节
2	EXTENT_SIZE	INTEGER	一个簇包含的 AU 个数
3	LOCAL_CODE	VARCHAR(64)	当前所连接的 ASMSERVER 的编码格式
4	LOCAL_LANG	VARCHAR(64)	当前所连接的 ASMSERVER 使用的语言： CN: 中文

			EN: 英文
5	USE_SHM	VARCHAR(8)	是否使用共享内存, TRUE/FALSE

### 14.2.27 V\$ASMGROUP

如果使用有 ASM 文件系统, 可通过此视图查看 ASM 磁盘组信息, 登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	GROUP_ID	INTEGER	磁盘组 ID
2	GROUP_NAME	VARCHAR(128)	磁盘组名称
3	N_DISK	INTEGER	磁盘组中包含的磁盘个数
4	AU_SIZE	INTEGER	单个 AU 大小, 单位为字节
5	EXTENT_SIZE	INTEGER	一个簇包含的 AU 个数
6	TOTAL_SIZE	INTEGER	磁盘组总大小, 单位为 M
7	FREE_SIZE	INTEGER	磁盘组空闲大小, 单位为 M
8	TOTAL_FILE_NUM	INTEGER	磁盘组中总的文件个数, 包括文件和目录

### 14.2.28 V\$ASMDISK

如果使用有 ASM 文件系统, 可通过此视图查看所有的 ASM 磁盘信息, 登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	GROUP_ID	INTEGER	所在的磁盘组 ID, 如果是未使用的磁盘, 则值为-1
2	DISK_ID	INTEGER	磁盘 ID, 如果是未使用的磁盘, 则值为-1
3	DISK_NAME	VARCHAR(128)	磁盘名称
4	DISK_PATH	VARCHAR(256)	磁盘路径
5	SIZE	BIGINT	磁盘大小, 单位为 M
6	FREE_AUNO	BIGINT	磁盘最大 AU 号

7	CREATE_TIME	VARCHAR(64)	磁盘创建时间
8	MODIFY_TIME	VARCHAR(64)	磁盘最近一次修改时间

### 14.2.29 V\$ASMFILE

如果使用有 ASM 文件系统，可通过此视图查看所有的 ASM 文件信息，登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	FILE_ID	BIGINT	文件 ID
2	TYPE	VARCHAR(32)	类型，目录或文件
3	PATH	VARCHAR(256)	文件完整路径
4	SIZE_BYTES	BIGINT	文件实际大小，单位为字节， 目录类型的文件不占用空间，值为 0
5	SIZE_TOTAL	BIGINT	文件占用总空间大小，单位为字节， 目录类型的文件不占用空间，值为 0
6	CREATE_TIME	VARCHAR(64)	文件创建时间
7	MODIFY_TIME	VARCHAR(64)	文件修改时间
8	GROUP_ID	INTEGER	所在磁盘组 ID
9	DISK_ID	INTEGER	inode 项所在磁盘 ID
10	DISK_AUNO	INTEGER	inode 项所在磁盘 AU 编号
11	AU_OFFSET	INTEGER	inode 项在 AU 内的偏移

### 14.2.30 V\$DCR\_INFO

查看 DCR 配置的全局信息，登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	VERSION	INTEGER	DCR 版本号
2	N_GROUP	INTEGER	DCR 配置的组个数
3	VTD_PATH	VARCHAR(256)	Voting Disk 路径
4	UDP_FLAG	INTEGER	是否使用 UDP 心跳机制，已无效

5	UDP_OGUID	BIGINT	校验用
---	-----------	--------	-----

### 14.2.31 V\$DCR\_GROUP

查看 DCR 配置的组信息，登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	GROUP_TYPE	VARCHAR(32)	组类型，CSS/ASM/DB
2	GROUP_NAME	VARCHAR(128)	组名称
3	N_EP	INTEGER	组中配置的 EP 个数
4	DSKCHK_CNT	INTEGER	磁盘容错时间，单位秒
5	NETCHK_TIME	INTEGER	网络容错时间，单位秒

### 14.2.32 V\$DCR\_EP

查看 DCR 配置的节点信息，登录任意节点查询得到的结果一致。

序号	列	数据类型	说明
1	GROUP_NAME	VARCHAR(128)	节点所属的组名
2	EP_NAME	VARCHAR(128)	节点名称
3	EP_SEQNO	INTEGER	节点的组内序号： 对 CSS/ASM 组的节点，是自动分配的序号，对 DB 组的节点，如果没有配置，也是自动分配的序号，否则是实际的配置序号。
4	EP_HOST	VARCHAR(128)	节点的 IP 地址，对 CSS/ASM 组的节点有效，表示登录节点的 IP 地址
5	EP_PORT	INTEGER	节点的 TCP 监听端口，对 CSS/ASM 组的节点有效，对应登录节点的端口号
6	UDP_PORT	INTEGER	节点的 UDP 监听端口，已无效
7	SHM_KEY	INTEGER	共享内存标识，初始化共享内存的标识符，对 ASM 组的节点有效
8	SHM_SIZE	INTEGER	共享内存大小，单位 M，初始化共享内存大小，

			对 ASM 组的节点有效
9	ASM_LOAD_PATH	VARCHAR(256)	ASM 磁盘扫描路径, 对 ASM 组的节点有效

### 14.2.33 V\$DSC\_REQUEST\_STATISTIC

统计 DSC 环境内 TYPE 类型请求时间, 仅显示登录节点的信息。

序号	列	数据类型	说明
1	TYPE	VARCHAR(64)	请求类型
2	TOTAL_REQUEST_COUNT	BIGINT	总请求次数
3	MAX_REQUEST_TIME	INTEGER	最大请求时间, 单位为微秒
4	MIN_REQUEST_TIME	INTEGER	最小请求时间, 单位为微秒
5	AVERAGE_REQUEST_TIME	INTEGER	平均请求时间, 单位为微秒
6	AVERAGE_RLOG_FLUSH_TIME	INTEGER	平均等待日志刷盘时间, 单位为微秒

### 14.2.34 V\$DSC\_REQUEST\_PAGE\_STATISTIC

统计 lbs\_XX 类型最耗时的前 100 页地址信息, 仅显示登录节点的信息。

序号	列	数据类型	说明
1	TYPE	VARCHAR(64)	请求类型
2	TS_ID	INTEGER	表空间 ID
3	FILE_ID	INTEGER	文件 ID
4	PAGE_NO	INTEGER	页号
5	REQUEST_TIME	INTEGER	花费时间, 单位为微秒

### 14.2.35 V\$DSC\_CRASH\_OVER\_INFO

显示 DSC 环境各节点数据页最小 first\_modified\_lsn, 以及故障节点 file\_lsn。

如果活动节点 buffer 中不存在更新页则 min\_first\_modified\_lsn 为 NULL; 节点故障后, 只有在所有 OK 节点 min\_first\_modified\_lsn 都大于或等于故障节点 file\_lsn 之后, 才允许故障节点重加入; 满足所有 OK 节点 min\_first\_modified\_lsn 都大于 crash\_lsn 之后, crash\_lsn 会清零。

序号	列	数据类型	说明
1	EP_SEQNO	INTEGER	DSC 节点号

2	OK_FLAG	INTEGER	节点是否 OK
3	MIN_FIRST_MODIFIED_LSN	BIGINT	最小 first_modified_lsn
4	CRASH_LSN	BIGINT	内存中记录的故障节点 file_lsn

## 15 备份还原

DMDSC 集群备份还原的功能、语法与单节点数据库基本保持一致,本章主要介绍 DMDSC 集群与单节点数据库备份、还原的使用方法差异,并说明在 DMDSC 集群中执行备份还原的一些注意事项。关于备份还原更详细的说明,请参考《DM8 备份与还原》手册。

### 15.1 DMDSC 和单节点差异

达梦数据库中,备份还原的对象包括:表、表空间和数据库。表备份还原的操作对象是数据页,而数据页是通过 BUFFER 获取的,与存储无关,因此 DMDSC 集群的表备份还原与单节点没有任何区别。表空间备份只需要访问属于这个表空间的数据文件,并不需要备份归档日志,因此 DMDSC 集群的表空间备份与单节点没有任何区别。表空间还原要求将表空间数据恢复到最新状态,需要重做归档日志,但 DMDSC 集群中本地归档往往是保存在本地磁盘中的,因此如何访问其他节点生成的归档日志,是 DMDSC 集群需要解决的问题。

数据库备份也需要访问所有节点的本地归档日志文件,同样需要解决如何访问其他节点的归档日志问题;另外,与单节点不同的是,DMDSC 集群备份过程中还需要记录备份开始和结束时各个节点的 LSN 信息,LSN 信息需要在节点间进行传递。数据库还原的过程就是从备份集中读取数据页并重新写入数据库文件中,而 DMDSC 集群的数据库文件是保存在共享存储中的,因此不要特别处理。但是,数据库的恢复操作有可能需要访问本地归档日志文件,因此也需要解决如何访问其他节点归档日志问题。

虽然可以将本地归档 (LOCAL ARCHIVE) 保存到共享存储中,解决其他节点归档日志问题,但是出于数据安全性和成本的考虑,一般建议将数据库备份文件和本地归档日志文件保存在本地磁盘中,避免由于共享存储的损坏,导致所有数据丢失、无法恢复的风险。而拷贝远程节点归档日志文件,到备份还原执行节点的做法,不但操作繁琐,而且在归档日志量比较大的情况,执行效率也存在很大问题。为了简化操作步骤,降低数据丢失风险,达梦数据库提供了远程归档 (REMOTE ARCHIVE) 功能,解决了 DMDSC 集群备份还原过程中访问其他节点归档日志文件问题。

配置远程归档后,DMDSC 集群中各个节点接收其他节点发送的 REDO 日志,并保存在节点的本地目录后,DMDSC 集群备份恢复的使用方法与单节点基本保持一致。



## 15.2 远程归档

所谓远程归档(REMOTE ARCHIVE),顾名思义就是将写入本地归档的 REDO 日志信息,发送到远程节点,并写入远程节点的指定归档目录中。DMDSC 集群中各个节点在配置本地归档之外,再相互配置一个远程归档,就可以在任意一个节点的本地磁盘中,找到 DMDSC 集群所有节点产生的、完整的归档日志文件。远程归档的触发时机是,在 REDO 日志写入本地归档日志文件的同时,将 REDO 日志通过 MAL 系统发送给指定的数据库实例。

远程归档与本地归档的主要区别是 REDO 日志写入的位置不同,本地归档将 REDO 日志写入数据库实例所在节点的磁盘,而远程归档则将 REDO 日志写入到其他数据库实例所在节点的指定归档目录。远程归档日志文件的命名规范和本地归档日志文件保持一致,都是以归档名+归档文件的创建时间进行组合命名。

远程归档与本地归档的另外一个区别就是归档失败的处理策略不同:本地归档写入失败(比如磁盘空间不足),系统将会挂起;而远程归档失败则会直接将远程归档失效,不再发送 REDO 日志到指定数据库实例。当节点间网络恢复、或者远程节点重启成功后,系统会自动检测并恢复远程归档,继续发送新写入的 REDO 日志,但不会主动补齐故障期间的 REDO 日志。因此,在出现节点故障等情况下,远程归档的内容有可能是不完整的,而本地归档的内容肯定是完整的;如果备份还原恰好需要用到这段丢失的远程归档日志,那么可以从源端的本地归档拷贝、补齐这部分内容。

与其他归档类型一样,远程归档也是配置在 dmarch.ini 文件中,远程归档相关的主要几个配置项包括:

1. ARCH\_TYPE 设置为 REMOTE,表示是远程归档
2. ARCH\_DEST 设置为远程数据库实例名,表示 REDO 日志发送到这个节点
3. ARCH\_INCOMING\_PATH 设置为本地存储路径,用于保存 ARCH\_DEST 实例发送的 REDO 日志

一般建议 DMDSC 集群中的节点,在配置本地归档之外,再交叉配置集群中所有其他节点的远程归档。查询 V\$DM\_ARCH\_INI、V\$ARCH\_STATUS 等动态视图可以获取归档配置以及归档状态等相关信息。

下面以两节点 DMDSC 集群为例,说明如何配置远程归档,DSC0 和 DSC1 是 DMDSC 集群中的两个实例,交叉进行 REMOTE 归档配置:

DSC0 实例的 dmarch.ini 配置:

```
[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL

ARCH_DEST           = /dmdata/dameng/arch_dsc0

ARCH_FILE_SIZE     = 128

ARCH_SPACE_LIMIT    = 0

[ARCH_REMOTE1]

ARCH_TYPE          = REMOTE

ARCH_DEST           = DSC1

ARCH_INCOMING_PATH = /dmdata/dameng/arch_dsc1

ARCH_FILE_SIZE     = 128

ARCH_SPACE_LIMIT    = 0
```

DSC1 实例的 dmarch.ini 配置:

```
[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL

ARCH_DEST           = /dmdata/dameng/arch_dsc1

ARCH_FILE_SIZE     = 128

ARCH_SPACE_LIMIT    = 0

[ARCH_REMOTE1]

ARCH_TYPE          = REMOTE

ARCH_DEST           = DSC0

ARCH_INCOMING_PATH = /dmdata/dameng/arch_dsc0

ARCH_FILE_SIZE     = 128

ARCH_SPACE_LIMIT    = 0
```



远程归档必须双向配置，单向配置时目标实例上不会接收归档日志，归档状态将会变成无效状态。

## 15.3 DMDSC 备份集

备份集除了保存备份对象的数据（数据页和归档日志），还记录了备份库节点的描述信息。单节点库生成的备份集，可以认为是只包含一个节点的特殊备份集。与节点相关的描述信息主要包括：

1. DMDSC 库的节点数，单节点库为 1
2. 备份开始时 DMDSC 节点的状态，以及各节点 REDO 日志的起始 LSN 和 SEQ
3. 备份结束时 DMDSC 节点 REDO 日志的结束 LSN 和 SEQ
4. 备份集中记录了执行备份节点的 dm.ini 配置参数，还原时使用备份集中的参数值覆盖目标库节点的 dm.ini 文件

备份操作可以在 DMDSC 集群的任意节点执行，生成的备份集可以存放在本地磁盘上，也可以存放到共享存储的 DMASM 目录中。但考虑到数据安全性，一般建议将备份集保存在本地磁盘上。可以通过以下方式，将备份集生成到本地磁盘：

1. 使用 dminit 初始化库时，将默认备份目录 bak\_path 设置为本地磁盘
2. 修改 DMDSC 集群中所有节点的 dm.ini 配置文件，将 bak\_path 设置为本地磁盘
3. 执行备份时，手动指定备份集路径为本地磁盘

## 15.4 DMDSC 备份还原实例

下面以从归档恢复为例说明 2 节点（DSC0、DSC1）DMDSC 环境下的备份恢复：

- 1) 搭建 DMDSC 环境，每个节点都需要配置双向的远程归档。归档配置示例如下：

DSC0 实例的 dmarch.ini 配置：

```
[ARCHIVE_LOCAL1]

ARCH_TYPE            = LOCAL

ARCH_DEST            = /dmdata/dameng/arch_dsc0

ARCH_FILE_SIZE       = 128

ARCH_SPACE_LIMIT     = 0

[ARCH_REMOTE1]

ARCH_TYPE            = REMOTE

ARCH_DEST            = DSC1
```

```
ARCH_INCOMING_PATH = /dmdata/dameng/arch_dsc1

ARCH_FILE_SIZE      = 128

ARCH_SPACE_LIMIT    = 0
```

DSC1 实例的 dmarch.ini 配置:

```
[ARCHIVE_LOCAL1]

ARCH_TYPE            = LOCAL

ARCH_DEST            = /dmdata/dameng/arch_dsc1

ARCH_FILE_SIZE      = 128

ARCH_SPACE_LIMIT    = 0

[ARCH_REMOTE1]

ARCH_TYPE            = REMOTE

ARCH_DEST            = DSC0

ARCH_INCOMING_PATH = /dmdata/dameng/arch_dsc0

ARCH_FILE_SIZE      = 128

ARCH_SPACE_LIMIT    = 0
```



远程归档必须双向配置，单向配置时目标实例上不会接收归档日志，归档状态将会变成无效状态。

2) 启动 Disql, 联机备份数据库。备份其中任意一个节点即可备份整个 DMDSC 环境。

```
SQL>BACKUP DATABASE BACKUPSET '/home/dm_bak/db_full_bak_for_dsc';
```

3) 还原数据库。还原数据库之前可选择对备份文件进行校验。需要注意的是，待还原的目标库可以是单机库，也可以是 DMDSC 库，且节点个数允许不同。

```
RMAN> RESTORE DATABASE '/opt/dmdbms/data/DAMENG_FOR_RESTORE/dm.ini' FROM
BACKUPSET '/home/dm_bak/db_full_bak_for_dsc';
```

4) 恢复数据库。DMDSC 库恢复要求各节点归档完整性由用户保证，即各节点的本地归档都能够访问到，若本地存在 REMOTE 归档，则可以使用 REMOTE 归档代替远程节点的本地归档。

```
RMAN>RECOVER DATABASE '/opt/dmdbms/data/DAMENG_FOR_RESTORE/dm.ini' WITH
ARCHIVEDIR '/dmdata/dameng/arch_dsc0', '/dmdata/dameng/arch_dsc1';
```

## 5) 数据库更新

```
RMAN>RECOVER DATABASE '/opt/dmdbms/data/DAMENG_FOR_RESTORE/dm.ini' UPDATE  
DB_MAGIC
```

## 15.5 使用说明

1. 配置远程归档时，必须同时配置本地归档。
2. DMDSC 集群环境中，备份还原涉及到的 trace 文件路径、DUMP 命令的映射文件路径、SHOW 命令的备份集信息输出文件路径都不支持 DMASM 类型文件
3. 由于 DMDSC 集群中，各个节点可能存在 LSN 值相同的 REDO 日志，恢复过程中无法严格校验归档日志的完整性；因此，需要用户保证全局归档日志的完整性
4. 在恢复过程中创建的数据文件，优先使用原始路径创建，如果创建失败，则会在 system\_path 目录下创建。因此，在恢复结束后，需要检查一下是否有数据文件创建在本地磁盘上，如果有则需要用户手动执行 SQL，将这些文件重新存放到共享存储、或者 DMASM 文件系统中，确保数据文件可以被 DMDSC 集群中的所有节点访问
5. 归档日志是恢复数据库的关键，建议将归档文件与数据库文件分别保存到不同的物理磁盘上；如果使用 DMASM 文件系统保存数据库文件，则建议将归档配置到本地磁盘上；以降低数据无法修复的风险
6. 如果需要访问 DMASM 文件系统，DMRMAN 必须设置 DCR\_INI 参数，指定 DCR 的访问配置
7. 数据库恢复过程中，需要保证本地归档和远程归档的完整性。如果由于节点故障等原因，导致远程归档不完整，则需要使用 DMRMAN 工具在对应节点修复本地归档后，将修复后本地归档拷贝过来，再进行恢复
8. 还原操作指定 REUSE DMINI 选项时，会将备份集中的 dm.ini 参数更新还原节点上的 dm.ini 配置文件，DMDSC 集群中其他节点的 dm.ini 并不会更新，需要用户手动修改

## 16 DMDSC 使用说明

DMDSC 集群针对同一份数据，提供了多个活动的数据库实例，因此具有负载均衡、高可靠性等特征。但与单机系统相比，DMDSC 具有更复杂的架构、更多的组件，因此对 DMDSC 集群的使用也提出了更高的要求。本章主要说明使用 DMDSC 集群的一些注意事项。

### 16.1 统一组件版本

搭建 DMDSC 环境时，应注意 DMDSC 中各实例使用的 DM 服务器版本应一致，同时还应注意各实例所在主机的操作系统位数、大小端模式、时区及时间设置都应一致。

此外，DMDSC 集群包含 `dmcss/dmasmsvr/dmasmapi/dmserver` 等诸多组件，并且 `dmasmsvr` 和 `dmasmapi` 之间需要用到共享内存进行数据交换。因此，搭建和使用 DMDSC 集群时，要特别注意各个组件的版本号是否一致。是否同为 32 位或 64 位。如果各组件版本号不一致，可能会导致系统异常，无法正常使用。

### 16.2 提升 DMDSC 性能

DMDSC 通过缓存交换和全局事务管理等机制，协调、管理多个节点的 CPU、内存等计算资源，以提升数据库管理系统的事务处理能力。数据和信息需要在节点间通过网络频繁地进行传输，如果多个数据库实例过多地访问、修改相同的数据页，就会导致缓存交换频繁，内部网络流量过高，形成 DMDSC 集群的性能瓶颈。为了提升 DMDSC 集群的整体性能，建议：

1. 使用带宽更大、速度更快的网络设备，降低数据和信息在节点之间传递的从网络延迟，比如使用 InfiniBand 或者万兆的以太网
2. 根据业务逻辑，把不同类别的数据库请求分别部署到不同的节点，减少不同节点对相同数据页或者数据库对象的共享访问、和修改
3. 优化应用逻辑和每一句 SQL，DMDSC 集群并不是解决性能问题的万能方案，未经优化、糟糕的 SQL 查询可能会极大降低数据库的吞吐量
4. 充分分析应用系统特征，以及性能瓶颈在哪里。比如一些 IO 密集型的应用系统，性能瓶颈往往出现在存储的 IO 上，这种情况下单纯地增加 DMDSC 节点数并不会提高性能，反而可能由于节点数增加，缓存交换更加频繁，进一步降低系统性能。针对这种情况，正确

的选择是增加磁盘，将 IO 分散到更多的磁盘；或者使用速度更快的固态盘来提升 IO 性能

5. 使用 'ALTER TABLE XXX WITHOUT COUNTER' 语句，关闭频繁插入、删除记录表的快速计数功能。DM8 默认开启表快速记录功能，系统动态维护表上的记录总数，并将记录数写入到表的控制页中；DMDSC 环境开启这个功能，并且多个节点并发向同一张表插入记录时，控制页的访问权限在节点间不断地进行回收和授权，控制页的内容也会不断地在节点间进行传递，进而影响 DMDSC 整体的并发性能

## 16.3 心跳说明

DMDSC 集群中，DMCSS 依赖心跳机制来判断集群中的各个实例是否处于正常状态。DMDSC 集群支持磁盘心跳检测机制。

磁盘心跳的载体是共享存储上的 Voting Disk，DMCSS、DMASMSVR、和 dmserver 启动后，每间隔 1 秒往 Voting Disk 各自固定的偏移写入时间戳，DMCSS 定时读取 Voting Disk 信息，根据时间戳变化情况来判断被监控对象是否活动。实例故障认定时间取决于配置参数 DCR\_GRP\_DSKCHK\_CNT，一般建议 DCR\_GRP\_DSKCHK\_CNT 至少配置为 60 秒以上，避免在系统极度繁忙情况下，由于操作系统调度原因导致误判实例故障。在规划 DMDSC 集群存储时，最好将 Voting Disk 与 IO 密集的数据库文件和日志文件分开存放，分别保存在不同的物理磁盘上，避免由于数据库 IO 影响心跳信息的写入和读取，导致极端情况下误判实例故障。

DMDSC 集群中磁盘心跳检测失败，DMCSS 会认为对应实例故障，启动故障处理流程。

## 16.4 重新格式化 DMASM

DMASM 文件系统格式是自描述的，DMASMSVR 启动时会从指定路径扫描裸设备，根据裸设备的头信息判读是否是 DMASM 磁盘，进而获取 DMASM 磁盘组元数据信息，初始化 DMASM 文件系统。所以要求一套硬件环境只能搭建一套 DMASM 文件系统，否则会产生冲突，会导致 dmasmshr 启动失败。如果要重新初始化 DMASM 文件系统，需要将指定路径下的所有裸设备头信息格式化一遍，避免新老环境 DMASM 磁盘信息冲突。

如果出现磁盘 id 相同导致 dmasmshr 启动失败的情况，在可以确定某一磁盘为无效磁盘的情况下，用 dmasmcmd 工具执行 create asmdisk disk\_path name 的方式清理无效磁盘信息。

## 16.5 重新初始化 DMDSC 库

dmserver 发生异常的情况下，DMCSS 会根据配置的心跳时间来确认节点故障，并写入故障信息到 DCR Disk 中，然后执行对应的故障处理或故障恢复。

对于 dmserver 节点全部故障的情况，如果想直接重新初始化数据库，则需要手动清理 DCR Disk 中的故障信息（DCR Disk 没有重新初始化），避免 DMCSS 对新初始化的库再次执行故障处理。

注意手动清理的时机，需要在 DMCSS 写入故障信息到 DCR Disk 之后，否则手动清理完成后，DMCSS 再写入故障信息（dmserver 的心跳时间配置比较长时会出现这种情况），清理操作仍然起不到作用。

DMCSS 手动清理之前，可通过下面三种方式避免 DMCSS 在清理之后再次修改 DCR Disk 中的故障信息：

1. 等待配置的心跳时间之后，确保 DMCSS 已写入故障信息之后，再执行手动清理
2. 确定 dmserver 确实故障时，通过监视器的 `ep halt` 命令，通知 DMCSS 写入故障信息，然后再执行手动清理
3. 直接退出 DMCSS，在手动清理完成之后再重启 DMCSS

可使用以下两种方式对指定组的故障节点信息进行清理：

1. 通过 `dmasmcmd` 的 `export` 命令，将 DCR Disk 的配置信息导出，手动修改 dmserver 对应组中的 `DCR_GRP_N_ERR_EP` 值为 0，`DCR_GRP_ERR_EP_ARR` 内容为空，则通过 `asmcmd` 的 `import` 命令导入到 DCR Disk 中
  2. 通过 `dmasmcmd` 的 `clear` 命令，直接清理指定组的故障节点信息
- 命令的具体用法请参考 [6.6 DMASMCMD](#) 的用法说明。

## 16.6 内部网络异常

DMDSC 环境多个 dmserver 之间的很多功能都通过 MAL 系统来传递控制命令或数据（事务管理、封锁管理等等，详见 [4.DMDSC 实现原理](#)）。如果网络故障导致 dmserver 间 MAL 通讯失败，出现 MAL 邮件丢失等情况，会导致 dmserver 一直收不到请求响应，整个系统卡住。

遇到这种情况，dmserver 会通过 Voting Disk 通知 DMCSSMAL 链路故障，由 DMCSS



挑选一个 dmserver 强制其主动 HALT 退出。接下来进行故障处理，将 DMDSC 集群恢复为单节点，对外提供数据库服务。

## 16.7 创建 DBLink

DMDSC 系统可支持 DBLink 的创建和访问。DMDSC 系统和单节点之间、DMDSC 和 DMDSC 系统之间都支持 DBLink 的创建，由于同构的 DBLink 之间依赖于 MAL 系统，只需要通过配置 dmmal.ini 文件即可实现。将每个实例 mal 项都加入 dmmal.ini 文件，并拷贝到每个实例目录下，即可创建 DBLink。

DMDSC 和单节点的 DBLink 配置，需在 DSC 节点的 dmmal.ini 中加入单节点 mal 项，同时复制到单节点，打开单节点的 MAL 配置，全部实例重新启动后即可创建 DBLink。

举例如下：

```
[MAL_INST1]

MAL_INST_NAME      = DSC01
MAL_HOST            = 10.0.2.101
MAL_PORT            = 7236
MAL_INST_HOST       = 192.168.0.101
MAL_INST_PORT       = 5336
MAL_LINK_MAGIC      = 0

[MAL_INST2]

MAL_INST_NAME      = DSC02
MAL_HOST            = 10.0.2.102
MAL_PORT            = 7237
MAL_INST_HOST       = 192.168.0.102
MAL_INST_PORT       = 5336
MAL_LINK_MAGIC      = 0

[MAL_INST3]    #单节点的配置项

MAL_INST_NAME      = EP01
MAL_HOST            = 10.0.2.103
```

```
MAL_PORT                = 7238

MAL_INST_HOST            = 192.168.0.103

MAL_INST_PORT           = 5336

MAL_LINK_MAGIC           = 0
```

两个 DMDSC 系统之间也类似，需在一套 DMDSC 的 `dmml.ini` 中加入另一套的 `dmml.ini` 内容，并拷贝到两套 DSC 系统的所有实例。

举例如下：

```
[MAL_INST1]

MAL_INST_NAME           = DSC011

MAL_HOST                = 10.0.2.101

MAL_PORT               = 7236

MAL_INST_HOST           = 192.168.0.101

MAL_INST_PORT           = 5336

MAL_LINK_MAGIC          = 0

[MAL_INST2]

MAL_INST_NAME           = DSC012

MAL_HOST                = 10.0.2.102

MAL_PORT               = 7237

MAL_INST_HOST           = 192.168.0.102

MAL_INST_PORT           = 5337

MAL_LINK_MAGIC          = 0

#另一套 DSC 系统

[MAL_INST3]

MAL_INST_NAME           = DSC021

MAL_HOST                = 10.0.2.102

MAL_PORT               = 7238

MAL_INST_HOST           = 192.168.0.102

MAL_INST_PORT           = 5338
```

```

MAL_LINK_MAGIC          = 0

[MAL_INST4]

MAL_INST_NAME           = DSC022

MAL_HOST                 = 10.0.2.101

MAL_PORT                = 7239

MAL_INST_HOST           = 192.168.0.101

MAL_INST_PORT           = 5339

MAL_LINK_MAGIC          = 0

```

某些应用场景下，DBLINK 需要跨网段访问数据库实例，我们只要根据实际情况，修改 MAL 配置参数，将处于相同网段实例的 MAL\_LINK\_MAGIC 配置为相同的值，不同网段实例的 MAL\_LINK\_MAGIC 配置为不同的值，就可以实现跨网段访问。详情可参考《DM8 系统管理员手册》dmmal.ini 配置章节。

## 16.8 节点硬件故障，如何启动 DMDSC 集群

正常的 DMDSC 集群启动流程，dmcss 会在监控到所有节点都启动后，再执行正常的数据库启动流程。当某个节点出现硬件故障无法启动时，DMDSC 集群无法自动启动。针对这种情况，dmcssm 监视器提供了 open force 命令，通知 dmcss 将未启动节点设置为故障状态，强制启动活动节点，以尽快恢复数据库服务。

## 16.9 MOUNT/OPEN 操作

登录 DSC 下任意节点都可以执行 MOUNT/OPEN 操作，仅在登录节点生效，如果需要所有节点都 MOUNT 或者 OPEN，需要登录所有节点执行命令。

## 16.10 裸设备路径变化

基于 DMASM 的 DMDSC 在运行过程中，如果磁盘重新进行规划，可能会导致磁盘对应的裸设备名有变化，从而导致 DMASM 环境配置可能也需要进行改变。

DMASM 文件系统是自描述系统，会自动扫描/dev/raw/下的所有裸设备，根据头信息来获取信息，和裸设备名路径没有关系。只是 DCR 和 Voting 磁盘的路径必须指定，若这

两个路径没有变化，可以不用修改 DMASM 环境配置。

如果 DCR 和 Voting 磁盘的路径发生了改变，则需要用 `dmasmcmd` 工具重新格式化 DCR 和 Voting 磁盘，不会影响其他 DMASM 磁盘，也不影响已经存在的 DMASM 文件。格式化后需要对 DCR 和 Voting 磁盘进行初始化：

- 如果保存了原始的 `dmdcr_cfg.ini` 文件，修改 `dmdcr_cfg.ini` 的 `DCR_VTD_PATH` 路径信息，再用 `dmasmcmd` 工具初始化就可以；

- 如果没有保存原始 `dmdcr_cfg.ini` 文件，可以用 `dmasmcmd` 工具先 `export` 出一份 `dmdcr_cfg.ini` 文件，再修改其中 `DCR_VTD_PATH` 路径信息，然后重新初始化。

# 17 附录

## 17.1 DMASMAPI 接口

DMASMAPI 是连接 DMASMSVR 执行操作的接口，所有使用 DMASM 文件系统的程序都使用 DMASMAPI 接口连接 DMASMSVR。

返回值分为三种类型：正数，0 和负数。0 表示正常；正数为警告信息；负数为错误信息，对应的错误码请参考 [17.1.2 错误码汇编](#)。

### 17.1.1 DMASMAPI 接口说明

DMASMAPI 提供的接口如下：

#### 1. dmasm\_sys\_init

函数原型：

---

ASMRETURN

```
dmasm_sys_init(  
  
    sdbyte*      err_desc,  
  
    uint4*       err_len,  
  
    uint4        char_code,  
  
    uint4        lang_id  
  
)
```

---

功能说明：

环境初始化接口。使用 ASMAPI 接口，必须第一个调用 dmasm\_sys\_init 接口。

参数说明：

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

char\_code: 输入参数，编码。

lang\_id : 输入参数，语言。

## 2. dmasm\_sys\_deinit

### 函数原型:

---

```
void  
  
dmasm_sys_deinit()
```

---

### 功能说明:

环境销毁接口。使用 ASMAPI 接口结束时，调用 dmasm\_sys\_deinit 销毁资源。

## 3. dmasm\_alloc\_con

### 函数原型:

---

```
ASMRETURN  
  
dmasm_alloc_con(  
  
    asmcon_handle*  con,  
  
    sbyte*          err_desc,  
  
    uint4*          err_len  
  
)
```

---

### 功能说明:

申请并初始化连接句柄。返回申请的连接句柄。

### 参数说明:

con: 输入参数，连接句柄。

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

## 4. dmasm\_free\_con

### 函数原型:

---

```
void  
  
dmasm_free_con(  
  
    asmcon_handle  con_in  
  
)
```

---

#### 功能说明:

释放连接句柄。

#### 参数说明:

con\_in: 输入参数, 连接句柄。

## 5. dmasm\_connect

#### 函数原型:

---

ASMRETURN

```
dmasm_connect(  
  
    asmcon_handle    con_in,  
  
    sdbyte*          username,  
  
    sdbyte*          password,  
  
    sdbyte*          hostname,  
  
    uint2            portnum,  
  
    asmbool*         con_is_local,  
  
    sdbyte*          err_desc,  
  
    uint4*           err_len  
  
)
```

---

#### 功能说明:

登录接口。ASMSVR 允许本地连接和远程连接, 但是 dmserver 仅允许本地连接, 依据 con\_is\_local 判断是否远程连接。

#### 参数说明:

con\_in: 输入参数, 连接句柄。

username: 输入参数, 用户名。

password: 输入参数, 密码。

hostname: 输入参数, 主机 ip 或主机名。

portnum: 输入参数, 主机监听端口号。

con\_is\_local: 输出参数, 标识远程或本地登录。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

## 6. dmasm\_close\_con

### 函数原型:

---

```
void  
  
dmasm_close_con(  
  
    asmcon_handle  con_in  
  
)
```

---

### 功能说明:

关闭连接。关闭连接句柄，释放资源。

### 参数说明:

conn\_in: 输入参数，连接句柄。

## 7. dmasm\_get\_n\_group

### 函数原型:

---

```
ASMRETURN  
  
dmasm_get_n_group(  
  
    asmcon_handle  conn_in,  
  
    uint2*         num,  
  
    sbyte*         err_desc,  
  
    uint4*         err_len  
  
)
```

---

### 功能说明:

获取 ASM Disk Group 个数。获取有多少个磁盘组。

### 参数说明:

conn\_in: 输入参数，连接句柄。

num: 输出参数，磁盘组个数。

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。



## 8. dmasm\_get\_group\_id\_arr

### 函数原型:

---

ASMRETURN

```
dmasm_get_group_id_arr(  
  
    asmcon_handle  con_in,  
  
    uint2*         id_arr,  
  
    uint2          arr_size,  
  
    uint2*         num,  
  
    sbyte*         err_desc,  
  
    uint4*         err_len  
  
)
```

---

### 功能说明:

获取 ASM Disk Group ID 数组。配合 dmasm\_get\_n\_group 使用，获取所有 Disk Group ID。

### 参数说明:

conn\_in: 输入参数，连接句柄。

id\_arr: 输出参数，磁盘 ID 数组。

arr\_size: 输入参数，数组最大长度。

num: 输出参数，返回数组长度。

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

## 9. dmasm\_get\_disk\_id\_arr\_by\_group

### 函数原型:

---

ASMRETURN

```
dmasm_get_disk_id_arr_by_group(  
  
    asmcon_handle  con_in,  
  
    uint2          group_id,  
  
    uint2*         id_arr,  
  
)
```

---

---

udint2	arr_size,
udint2*	n_disk,
sdbyte*	err_desc,
udint4*	err_len

---

)

---

#### 功能说明:

获取磁盘组内磁盘 ID 数组。根据磁盘组 ID 获取磁盘组内包含的所有磁盘 ID。

#### 参数说明:

conn\_in: 输入参数, 连接句柄。

group\_id: 输入参数, 磁盘组 ID。

id\_arr: 输出参数, 磁盘 ID 数组。

arr\_size: 输入参数, 数组最大长度。

n\_disk: 输出参数, 返回数组长度。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 10. dmasm\_get\_disk\_info

#### 函数原型:

---

ASMRETURN

dmasm\_get\_disk\_info(

asmcon_handle	conn_in,
udint2	group_id,
udint4	disk_id,
sdbyte*	path,
udint2	path_buflen,
sdbyte*	name,
udint2	name_buflen,
udint4*	size,
udint4*	free_auno,

---

---

sdbyte*	create_time,
sdbyte*	modify_time,
sdbyte*	err_desc,
udint4*	err_len

---

)

---

#### 功能说明:

获取 ASM 磁盘详细信息。根据磁盘组 ID 和磁盘 ID 获取 ASM 磁盘详细信息。

#### 参数说明:

conn\_in: 输入参数, 连接句柄。

group\_id: 输入参数, 磁盘组 ID。

disk\_id: 输入参数, 磁盘 ID。

path: 输出参数, 磁盘路径。

path\_bufllen: 输入参数, path 缓冲区长度。

name: 输出参数, 磁盘名称。

name\_bufllen: 输入参数, name 缓冲区长度。

size: 输出参数, 磁盘大小, 单位 M。

free\_auno: 输出参数, 最大 au 号。

create\_time: 输出参数, 磁盘创建时间。

modify\_time: 输出参数, 磁盘最近一次修改时间。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 11. dmasm\_create\_diskgroup

#### 函数原型:

---

ASMRETURN

```
dmasm_create_diskgroup(
    asmcon_handle  conn_in,
    sdbyte*        group_name,
    sdbyte*        disk_path_in,
```

---

---

```
    sbyte*      err_desc,  
  
    uint4*      err_len  
)
```

---

#### 功能说明:

创建 ASM Disk Group。使用 disk\_path 所指的 ASM 磁盘创建 AMS 磁盘组。

#### 参数说明:

conn\_in: 输入参数, 连接句柄。

group\_name: 输入参数, 磁盘组名。

disk\_path\_in: 输入参数, 磁盘路径。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 12. dmasm\_add\_disk\_to\_diskgroup

#### 函数原型:

---

ASMRETURN

```
dmasm_add_disk_to_diskgroup(  
  
    asmcon_handle  conn_in,  
  
    sbyte*         group_name,  
  
    sbyte*         disk_path_in,  
  
    sbyte*         err_desc,  
  
    uint4*         err_len  
)
```

---

#### 功能说明:

磁盘组增加磁盘。往磁盘组增加 ASM 磁盘。

#### 参数说明:

conn\_in: 输入参数, 连接句柄。

group\_name: 输入参数, 磁盘组名。

disk\_path\_in: 输入参数, 磁盘路径。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

### 13. dmasm\_drop\_diskgroup\_by\_name

#### 函数原型:

---

ASMRETURN

```
dmasm_drop_diskgroup_by_name(  
  
    asmcon_handle  conn_in,  
  
    sdbyte*        group_name,  
  
    sdbyte*        err_desc,  
  
    uint4*         err_len  
  
)
```

---

#### 功能说明:

删除磁盘组。ASM 文件系统不支持单独删除 ASM 磁盘, 只能删除整个 ASM 磁盘组

#### 参数说明:

conn\_in: 输入参数, 连接句柄。

group\_name: 输入参数, 磁盘组名。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

### 14. dmasm\_file\_create

#### 函数原型:

---

ASMRETURN

```
dmasm_file_create(  
  
    asmcon_handle  conn_in,  
  
    asmbool p_flag,  
  
    sdbyte*        filepath_in,  
  
    uint8  filesize,  
  
    asm_fhandle_t* fil_handle,  
  
    sdbyte*        err_desc,
```

---

---

```
    uint4*      err_len  
)  

```

---

#### 功能说明:

创建 ASM 文件。在 ASM 文件系统中创建 ASM 文件，如果文件父目录不存在，并且 p\_flag 为 TRUE 的情况，会自动创建父目录，否则会报错。

#### 参数说明:

conn\_in: 输入参数，连接句柄。

p\_flag: 输入参数，创建父目录标记。

filepath\_in: 输入参数，文件路径。

filesize: 输入参数，文件大小，单位 Byte。

fil\_handle: 输出参数，文件句柄。

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

## 15. dmasm\_file\_open

#### 函数原型:

---

ASMRETURN

```
dmasm_file_open(  
  
    asmcon_handle  conn_in,  
  
    sdbyte*        filepath_in,  
  
    asm_fhandle_t* fhandle,  
  
    sdbyte*        err_desc,  
  
    uint4*         err_len  
  
)  

```

---

#### 功能说明:

打开 ASM 文件，获取 ASM 文件句柄。已经打开的文件，不能被删除。

#### 参数说明:

conn\_in: 输入参数，连接句柄。

filepath\_in: 输入参数，文件路径。

fhandle: 输入参数, 文件句柄。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 16. dmasm\_file\_trunc

### 函数原型:

---

ASMRETURN

```
dmasm_file_trunc(  
    asmcon_handle  conn_in,  
    asm_fhandle_t  fhandle,  
    uint8          truncate_size,  
    uint8*         real_size,  
    sbyte*         err_desc,  
    uint4*         err_len  
)
```

---

### 功能说明:

截断 ASM 文件。将 ASM 文件截断到 truncate\_size, 如果 truncate\_size 小于文件大小, 文件会被截断到 truncate\_size; 如果 truncate\_size 大于文件大小, 文件大小不变, 接口返回 EC\_SUCCESS。

### 参数说明:

conn\_in: 输入参数, 连接句柄。

fhandle: 输入参数, 文件句柄。

truncate\_size: 输入参数, 截断后的大小, 单位 Byte。

real\_size: 输出参数, 执行后实际大小, 单位 Byte。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 17. dmasm\_file\_extend

### 函数原型:

---

ASMRETURN

```
dmasm_file_extend(  
  
    asmcon_handle  conn_in,  
  
    asm_fhandle_t  fhandle,  
  
    uint8          offset,  
  
    uint8          extend_size,  
  
    sdbyte*        err_desc,  
  
    uint4*         err_len  
  
)
```

---

#### 功能说明:

扩展 ASM 文件。将文件从 offset 偏移处, 扩展 extent\_size 大小, 最终实际大小为 offset+extent\_size。如果 offset+extent\_size 大于文件大小, 文件会被扩展到 offset+extent\_size 大小; 如果 offset+extent\_size 小于文件大小, 直接返回成功。

#### 参数说明:

conn\_in: 输入参数, 连接句柄。

fhandle: 输入参数, 文件句柄。

offset: 输入参数, 起始偏移。

extend\_size: 输入参数, 扩展大小。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 18. dmasm\_file\_close

#### 函数原型:

---

```
void  
  
dmasm_file_close(  
  
    asmcon_handle  conn_in,  
  
    asm_fhandle_t  fhandle  
  
)
```

---



### 功能说明:

关闭 ASM 文件。关闭打开的 ASM 文件

### 参数说明:

conn\_in: 输入参数, 连接句柄。

fhandle: 输入参数, 文件句柄。

## 19. dmasm\_file\_delete

### 函数原型:

---

ASMRETURN

```
dmasm_file_delete(  
  
    asmcon_handle  conn_in,  
  
    sdbyte*        filepath_in,  
  
    sdbyte*        err_desc,  
  
    uint4*         err_len  
  
)
```

---

### 功能说明:

删除 ASM 文件。删除 ASM 文件, 正在被使用的 ASM 文件不能被删除。

### 参数说明:

conn\_in: 输入参数, 连接句柄。

filepath\_in: 输入参数, 文件路径。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 20. dmasm\_file\_read\_by\_offset

### 函数原型:

---

ASMRETURN

```
dmasm_file_read_by_offset(  
  
    asmcon_handle  conn_in,  
  
    asm_fhandle_t  fhandle,
```

---

---

uint8	offset,
ubyte*	buffer,
uint4	bytes_to_read,
ubyte*	err_desc,
uint4*	err_len

---

)

---

#### 功能说明:

从 ASM 文件读取数据。从 ASM 文件 offset 偏移读取 bytes\_to\_read 大小的内容到缓冲区 buffer，调用者保证缓冲区够用。因为裸设备读写限制，offset，buffer，bytes\_to\_read 都必须能被 512 整除，否则会报错。

#### 参数说明:

conn\_in: 输入参数，连接句柄。

fhandle: 输入参数，文件句柄。

offset: 输入参数，起始偏移。

buffer: 输入参数，缓冲区。

bytes\_to\_read: 输入参数，读取数据大小，单位 Byte。

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

## 21. dmasm\_file\_read\_by\_offset\_normal

#### 函数原型:

---

ASMRETURN

dmasm\_file\_read\_by\_offset\_normal(

asmcon\_handle conn\_in,

asm\_fhandle\_t fhandle,

uint8 offset,

ubyte\* buffer,

uint4 bytes\_to\_read,

ubyte\* err\_desc,

---

---

```

        uint4*      err_len
    )

```

---

#### 功能说明:

从 ASM 文件读取数据。从 ASM 文件 `offset` 偏移读取 `bytes_to_read` 大小的内容到缓冲区 `buffer`，调用者保证缓冲区够用。该接口支持 `offset,buffer,bytes_to_read` 不是 512 整数倍，但是性能比 `dmasm_file_read_by_offset` 慢。

#### 参数说明:

`conn_in`: 输入参数，连接句柄。

`fhandle`: 输入参数，文件句柄。

`offset`: 输入参数，起始偏移。

`buffer`: 输入参数，缓冲区。

`bytes_to_read`: 输入参数，写取数据大小，单位 Byte。

`err_desc`: 输出参数，错误描述信息。

`err_len`: 输入输出参数，错误描述信息长度。

## 22. dmasm\_file\_write\_by\_offset

#### 函数原型:

---

```

ASMRETURN

```

```

dmasm_file_write_by_offset(
    asmcon_handle  conn_in,
    asm_fhandle_t  fhandle,
    uint8          offset,
    sdbyte*        buffer,
    uint4          bytes_to_write,
    sdbyte*        err_desc,
    uint4*         err_len
)

```

---

#### 功能说明:

将数据写入 ASM 文件。将缓冲区中的内容写入 ASM 文件，从 `offset` 偏移开始。因为

裸设备读写限制，offset，buffer 地址，bytes\_to\_write 都必须能被 512 整除。

**参数说明：**

conn\_in: 输入参数，连接句柄。

fhandle: 输入参数，文件句柄。

offset: 输入参数，起始偏移。

buffer: 输入参数，缓冲区。

bytes\_to\_write: 输入参数，写数据大小，单位 Byte。

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

## 23. dmasm\_file\_write\_by\_offset\_normal

**函数原型：**

---

ASMRETURN

```
dmasm_file_write_by_offset_normal(  
  
    asmcon_handle    conn_in,  
  
    asm_fhandle_t     fhandle,  
  
    uint8    offset,  
  
    sdbyte*      buffer,  
  
    uint4        bytes_to_write,  
  
    sdbyte*      err_desc,  
  
    uint4*       err_len  
  
)
```

---

**功能说明：**

将数据写入 ASM 文件。将缓冲区中的内容写入 ASM 文件，从 offset 偏移开始。该接口支持 offset,buffer,bytes\_to\_write 不是 512 倍数，但是性能比 dmasm\_file\_write\_by\_offset 慢。

**参数说明：**

conn\_in: 输入参数，连接句柄。

fhandle: 输入参数，文件句柄。

offset: 输入参数, 起始偏移。

buffer: 输入参数, 缓冲区。

bytes\_to\_write: 输入参数, 写数据大小, 单位 Byte。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 24. dmasm\_file\_copy

### 函数原型:

---

ASMRETURN

```
dmasm_file_copy(  
    asmcon_handle  conn_in,  
    sdbyte*        source_in,  
    sdbyte*        dest_in,  
    asmbool        bOverwriteIfExists,  
    sdbyte*        err_desc,  
    uint4*         err_len  
)
```

---

### 功能说明:

文件拷贝操作。支持 ASM 文件拷贝到 ASM 文件; ASM 文件拷贝到普通文件系统文件; 普通文件系统文件拷贝到 ASM 文件系统; 不支持普通文件拷贝到普通文件。

bOverwriteIfExists: 0 或者 NULL 表示不覆盖, 其他非 0 值表示覆盖。

### 参数说明:

conn\_in: 输入参数, 连接句柄。

source\_in: 输入参数, 源文件路径。

dest\_in: 输入参数, 目标文件路径。

bOverwriteIfExists: 输入参数, 如果目标存在是否删除。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 25. dmasm\_dir\_create

### 函数原型:

---

ASMRETURN

```
dmasm_dir_create(  
  
    asmcon_handle  conn_in,  
  
    asmbool        p_flag,  
  
    sdbyte*        fdir_in,  
  
    sdbyte*        err_desc,  
  
    uint4*         err_len  
  
)
```

---

### 功能说明:

创建目录。ASM 文件系统创建目录，当 p\_flag=TRUE 时会级联创建父目录，否则父目录不存在会报错。

### 参数说明:

conn\_in: 输入参数，连接句柄。

p\_flag: 输入参数，是否级联创建父目录。

fdir\_in: 输入参数，目录路径。

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

## 26. dmasm\_dir\_delete

### 函数原型:

---

ASMRETURN

```
dmasm_dir_delete(  
  
    asmcon_handle  conn_in,  
  
    sdbyte*        fdir_in,  
  
    sdbyte*        err_desc,  
  
    uint4*         err_len  
  
)
```

---

### 功能说明:

删除目录，以及目录下面所有的文件。

### 参数说明:

conn\_in: 输入参数，连接句柄。

fdir\_in: 输入参数，目录路径。

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

## 27. dmasm\_get\_file\_num\_by\_group

### 函数原型:

---

ASMRETURN

```
dmasm_get_file_num_by_group(  
    asmcon_handle    conn_in,  
    uint2            group_id,  
    uint4*           num,  
    sbyte*           err_desc,  
    uint4*           err_len  
)
```

---

### 功能说明:

获取磁盘组内总的文件个数。根据磁盘组 ID 获取总的文件个数，包括文件和目录。

### 参数说明:

conn\_in: 输入参数，连接句柄。

group\_id: 输入参数，磁盘组的 ID 。

num: 输出参数，文件的个数。

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

## 28. dmasm\_get\_extent\_size

### 函数原型:

---

ASMRETURN

```
dmasm_get_extent_size(  
  
    asmcon_handle   conn_in,  
  
    uint4*          ex_size,  
  
    sdbyte*         err_desc,  
  
    uint4*          err_len  
  
)
```

---

#### 功能说明:

获取 ASM 文件系统簇大小。获取 ASM 文件系统簇大小。

#### 参数说明:

conn\_in: 输入参数，连接句柄。

ex\_size: 输出参数，簇的大小，包括几个 AU。

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

## 29. dmasm\_get\_file\_info

#### 函数原型:

---

ASMRETURN

```
dmasm_get_file_info(  
  
    asmcon_handle   conn_in,  
  
    asm_fhandle_t   file_id,  
  
    ASM_FILE_ATTR*  fattr_out,  
  
    sdbyte*         err_desc,  
  
    uint4*          err_len  
  
)
```

---

#### 功能说明:

获取 ASM 文件详细信息。

ASM\_FILE\_ATTR 包括:

type: 目录标记，1 表示文件，2 表示目录。



name: 文件名称。

full\_path: 完整路径。

size: 文件大小 (Byte 为单位), 目录忽略此字段。

c\_type: 创建时间。

m\_time: 修改时间。

group\_id: 所在磁盘组编号。

disk\_id: inode 项所在磁盘 ID。

disk\_auno: inode 项所在磁盘 AU 编号。

offset: inode 项 AU 偏移。

#### 参数说明:

conn\_in: 输入参数, 连接句柄。

file\_id: 输入参数, 打开的文件句柄。

fattr\_out: 输出参数, 文件属性结构。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 30. dmasm\_file\_data\_init\_normal

#### 函数原型:

---

ASMRETURN

```
dmasm_file_data_init_normal(  
  
    asmcon_handle    conn_in,  
  
    asm_fhandle_t    fhandle,  
  
    sdbyte*          err_desc,  
  
    uint4*           err_len  
  
)
```

---

#### 功能说明:

文件清零接口。ASMSVR 只提供创建文件, 分配空间动作, 由于底层是裸设备, 所以文件内容是不确定的, 所以在创建文件和扩展文件后, 如果有需要要由用户主动调用该接口清零。

#### 参数说明:

conn\_in: 输入参数, 连接句柄。

fhandle: 输入参数, 打开的目录句柄。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 31. dmasm\_get\_group\_info\_by\_name

#### 函数原型:

---

ASMRETURN

```
dmasm_get_group_info_by_name(  
    asmcon_handle  con_in,  
    sdbyte*        group_name,  
    udint2*        group_id,  
    udint2*        status,  
    udint2*        n_disk,  
    udint4*        total_size,  
    udint4*        free_size,  
    sdbyte*        err_desc,  
    udint4*        err_len  
)
```

---

#### 功能说明:

通过磁盘组名获取磁盘组详细信息。

#### 参数说明:

con\_in: 输入参数, 连接句柄。

group\_name: 输入参数, 磁盘组名字。

group\_id: 输出参数, 磁盘组 ID。

status: 输出参数, 磁盘组状态(1:正在创建中 2: 正常的 3: 正在删除中)。

n\_disk: 输出参数, 磁盘数。

total\_size: 输出参数, 磁盘组大小, 单位 M。

free\_size: 输出参数, 磁盘组空闲大小, 单位 M。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 32. dmasm\_get\_group\_info\_by\_id

### 函数原型:

---

ASMRETURN

```
dmasm_get_group_info_by_id(  
    asmcon_handle  con_in,  
  
    uint2          group_id,  
  
    sdbyte*        group_name,  
  
    uint2          name_buflen,  
  
    uint2*         status,  
  
    uint2*         n_disk,  
  
    uint4*         total_size,  
  
    uint4*         free_size,  
  
    sdbyte*        err_desc,  
  
    uint4*         err_len  
)
```

---

### 功能说明:

通过磁盘组 ID 获取磁盘组详细信息。

### 参数说明:

con\_in: 输入参数, 连接句柄。

group\_id: 输入参数, 磁盘组 ID。

group\_name: 输出参数, 磁盘组名字。

name\_buflen: 输入参数, 磁盘组的 buf 长度

status: 输出参数, 磁盘组状态(1:正在创建中 2: 正常的 3: 正在删除中)。

n\_disk: 输出参数, 磁盘数。

total\_size: 输出参数, 磁盘组大小, 单位 M。

free\_size: 输出参数, 磁盘组空闲大小, 单位 M。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

### 33. dmasm\_dir\_get\_first

#### 函数原型:

---

ASMRETURN

```
dmasm_dir_get_first(  
    asmcon_handle    conn_in,  
    sdbyte*          path_in,  
    sdbyte*          suffix,  
    asm_dhandle_t*   dir_handle_out,  
    ASM_FILE_ATTR*   fattr_out,  
    asmbool*         exist_flag,  
    sdbyte*          err_desc,  
    uint4*           err_len  
)
```

---

#### 功能说明:

获取目录下第一个文件信息。

#### 参数说明:

conn\_in: 输入参数, 连接句柄。

path\_in: 输入参数, 目录名字。

suffix: 输入参数, 过滤扩展名, 如: ".log"。

dir\_handle\_out: 输出参数, 打开的目录句柄。

fattr\_out: 输出参数, 文件属性结构。

exist\_flag: 输出参数, 是否存在。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 34. dmasm\_dir\_get\_next

### 函数原型:

---

ASMRETURN

```
dmasm_dir_get_next(  
  
    asmcon_handle    conn_in,  
  
    asm_dhandle_t    dir_handle,  
  
    sdbyte*          path_in,  
  
    sdbyte*          suffix,  
  
    ASM_FILE_ATTR*   fattr_out,  
  
    asmbool*         exist_flag,  
  
    sdbyte*          err_desc,  
  
    uint4*           err_len  
  
)
```

---

### 功能说明:

获取目录下下一个文件信息。

### 参数说明:

conn\_in: 输入参数, 连接句柄。

dir\_handle: 输入参数, 打开的目录句柄。

path\_in: 输入参数, 目录名字。

suffix: 输入参数, 过滤扩展名, 如: ".log"。

fattr\_out: 输出参数, 文件属性结构。

exist\_flag: 输出参数, 是否存在。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 35. dmasm\_dir\_close

### 函数原型:

---

ASMRETURN

```
dmasm_dir_close(  
  

```

---

---

```

asmcon_handle    conn_in,

asm_dhandle_t    dir_handle,

sdbyte*          err_desc,

udint4*          err_len

)

```

---

#### 功能说明:

关闭目录。

#### 参数说明:

con\_in: 输入参数, 连接句柄。

dir\_handle: 输入参数, 打开的目录句柄。

err\_desc: 输出参数, 错误描述信息。

err\_len: 输入输出参数, 错误描述信息长度。

## 36. dmasm\_file\_attributes\_get

#### 函数原型:

---

ASMRETURN

```

dmasm_file_attributes_get(

asmcon_handle    conn_in,

sdbyte*          path,

ASM_FILE_ATTR*   fattr_out,

sdbyte*          err_desc,

udint4*          err_len

)

```

---

#### 功能说明:

根据文件路径获取文件详细信息。

#### 参数说明:

conn\_in: 输入参数, 连接句柄。

path: 输入参数, 路径。

fattr\_out: 输出参数, 的文件属性结构。

err\_desc: 输出参数，错误描述信息。

err\_len: 输入输出参数，错误描述信息长度。

### 17.1.2 错误码

DMASMAPI 的错误码值域从-11000 开始，具体请参考 V\$ERR\_INFO。

## 17.2 DMCSSM 接口

### 17.2.1 DLL 依赖库

监视器的 DLL 名称为 dmcssmon.dll (windows) 和 libdmcssm.so (linux)，使用监视器接口时必须加载，另外监视器 DLL 还依赖了其他的一些动态链接库，编程时需要将达梦 bin 目录的以下库拷贝到和 dmcssmon.dll 或 libdmcssm.so 相同目录下，也可以在加载动态链接库时直接指定 bin 目录，就不需要再拷贝库文件：

序号	WINDOWS	非 WINDOWS (LINUX 等)
1	dmcalc.dll	libdmcalc.so
2	dmcfg.dll	libdmcfg.so
3	dmcomm.dll	libdmcomm.so
4	dmcvdt.dll	libdmcvdt.so
5	dmcyt.dll	libdmcyt.so
6	dmdcr.dll	libdmdcr.so
7	dmelog.dll	libdmelog.so
8	dmmem.dll	libdmmem.so
9	dmmout.dll	libdmmout.so
10	dmos.dll	libdmos.so
11	dmsys.dll	libdmsys.so
12	dmutil.dll	libdmutil.so

## 17.2.2 返回值说明

监视器接口总体的返回值策略为：小于0表示执行失败，其他值表示执行成功。

相关的错误码说明请参考17.2.7小节，也可借助`cssm_get_error_msg_by_code`接口获取错误码对应的错误描述信息。

## 17.2.3 接口说明

### 1. `cssm_alloc_handle`

函数原型：

---

```
CSSM_RETURN  
  
cssm_alloc_handle(  
  
    mhandle*      mhdle  
  
);
```

---

功能说明：

分配监视器操作句柄，监视器的接口都通过此句柄调用执行。

参数说明：

`mhdle`：输出参数，分配到的监视器句柄。

返回值：

0：执行成功。

<0：执行失败。

### 2. `cssm_get_error_msg_by_code`

函数原型：

---

```
CSSM_RETURN  
  
cssm_get_error_msg_by_code(  
  
    mhandle      mhdle,  
  
    mint4        code,  
  
    mschar*      buf_msg,
```

---



---

```

    muint4      buf_len,

    muint4*     msg_len_out

);

```

---

#### 功能说明:

获取输入 code 对应的错误描述信息，code 必须是小于 0 的值。

#### 参数说明:

mhandle: 输入参数，分配到的监视器句柄，注意如果是分配句柄或初始化环境时失败，允许 mhandle 是无效的，否则要求 mhandle 必须是有效句柄。

code: 输入参数，需要获取错误信息的错误码 code，注意 code 必须是小于 0 的值。

buf\_msg: 输出参数，输出错误描述信息，注意 buf\_msg 缓存长度建议大于 4096，避免输出信息被截断。

buf\_len: 输入参数，指定 buf\_msg 可写入的最大长度。

msg\_len\_out: 输出参数，buf\_msg 缓存实际写入长度。

#### 返回值:

0: 执行成功。

<0: 执行失败，-10 表示没有找到 code 对应的错误信息。

### 3. cssm\_init

#### 函数原型:

---

```

CSSM_RETURN

cssm_init(

    mhandle      mhandle,

    mschar*      ini_path,

    mschar*      log_path

);

```

---

#### 功能说明:

初始化监视器环境。

#### 参数说明:

mhandle: 输入参数，监视器操作句柄。

`ini_path`: 输入参数, 指定 `dmcssm.ini` 的绝对路径。

`log_path`: 输入参数, 指定日志文件的存放路径, 如果为 `NULL` 或空串, 则将 `dmcssm.ini` 中配置的 `MON_LOG_PATH` 作为日志文件路径, 如果 `dmcssm.ini` 中没有配置 `MON_LOG_PATH`, 则将 `dmcssm.ini` 的同级目录作为日志文件路径。

**返回值:**

0: 执行成功。

<0: 执行失败, -7 表示监视器尝试建立到所有 CSS 的连接失败, 可能 CSS 都未启动, 允许忽略此错误继续执行其他操作, 如果不忽略此错误, 认定初始化失败, 则需要先正常销毁监视器环境 (调用 `cssm_deinit` 接口), 再释放操作句柄 (调用 `cssm_free_handle` 接口)。

## 4. `cssm_msg_event_wait`

**函数原型:**

---

```
void  
  
cssm_msg_event_wait(  
  
    mhandle      mhdle  
  
);
```

---

**功能说明:**

等待消息事件。监视器接口命令执行过程中产生的中间输出消息, 以及收到 CSS 自动处理的消息时都会将消息写入到缓存并通知消息事件, 调用者只需要调用此接口等待即可, 等待事件发生后, 可通过接口 `cssm_get_exec_msg` 去读取输出消息。

这种方式需要单独起一个线程来处理消息, 以免消息被阻塞。

**参数说明:**

`mhdle`: 输入参数, 监视器操作句柄。

**返回值:**

无

## 5. cssm\_get\_exec\_msg

### 函数原型:

---

CSSM\_RETURN

```
cssm_get_exec_msg(  
    mhandle      mhdle,  
    mschar*      buf_msg,  
    muint4       buf_len,  
    muint4*      msg_len_out,  
    muint4*      get_flag  
);
```

---

### 功能说明:

获取输出信息，和 `cssm_msg_event_wait` 配合使用。

### 参数说明:

`mhdle`: 输入参数，监视器操作句柄。

`buf_msg`: 输出参数，保存获取到的输出消息，注意 `buf_msg` 缓存长度需要大于 4096，以免长度不够导致消息被截断。

`buf_len`: 输入参数，指定 `buf_msg` 可写入的最大长度，建议 `buf_msg` 缓存长度及 `buf_len` 输入值大于 4096，避免消息被截断。

`msg_len_out`: 输出参数，写消息成功后，输出实际写入的消息长度。

`get_flag`: 输出参数，是否继续读取消息，TRUE 表示还有消息可读，FALSE 表示已全部读完。

### 返回值:

0: 执行成功。

<0: 执行失败。

## 6. cssm\_get\_msg\_exit\_flag

### 函数原型:

---

CSSM\_RETURN

---

---

```
cssm_get_msg_exit_flag(  
  
    mhandle        mhdle,  
  
    mbool*         exit_flag  
  
);
```

---

#### 功能说明:

如果上层应用程序中有单独的消息线程，可通过调用此接口获取退出标记，在标记为 TRUE 时可正常退出线程。

#### 参数说明:

mhdle: 输入参数，监视器操作句柄。

exit\_flag: 输出参数，输出退出标记，1 表示可以正常退出，0 表示不能退出。

#### 返回值:

0: 执行成功。

<0: 执行失败。

## 7. cssm\_set\_msg\_exit\_event

#### 函数原型:

---

```
void  
  
cssm_set_msg_exit_event(  
  
    mhandle        mhdle  
  
);
```

---

#### 功能说明:

设置消息退出事件。如果上层应用程序中有单独的消息线程，在 cssm\_get\_msg\_exit\_flag 接口获取到退出标记为 TRUE 时，需要调用此接口设置退出事件。

#### 参数说明:

mhdle: 输入参数，监视器操作句柄。

#### 返回值:

无

## 8. cssm\_msg\_event\_deinit

### 函数原型:

---

```
void  
  
cssm_msg_event_deinit(  
  
    mhandle      mhdle  
  
);
```

---

### 功能说明:

通知并等待消息退出。使用等待事件方式获取监视器消息时，需要单独起一个线程，在程序结束，退出线程时，需要调用此接口通知并等待消息线程退出。

消息线程相关的接口有 `cssm_msg_event_wait`、`cssm_get_exec_msg`、`cssm_get_msg_exit_flag`、`cssm_set_msg_exit_event` 和 `cssm_msg_event_deinit`，这几个接口需要配合使用。

### 参数说明:

mhdle: 输入参数，监视器操作句柄。

### 返回值:

无

## 9. cssm\_get\_master\_css\_info

### 函数原型:

---

```
CSSM_RETURN  
  
cssm_get_master_css_info(  
  
    mhandle      mhdle,  
  
    mbyte*       ep_seqno,  
  
    mschar*      ep_name  
  
);
```

---

### 功能说明:

获取主 CSS 节点的组内序号和节点名称。



CSS 需要 5s 的选举时间来确定主 CSS，在 CSS 的选举时间内，此接口会执行失败，获取不到主 CSS 信息。

#### 参数说明：

mhandle：输入参数，监视器操作句柄。

ep\_seqno：输出参数，输出主 CSS 节点的组内序号。

ep\_name：输出参数，输出主 CSS 节点的名称，注意缓存长度不能小于 65，避免长度溢出。

#### 返回值：

0：执行成功。

<0：执行失败。

## 10. cssm\_get\_master\_asm\_info

#### 函数原型：

---

CSSM\_RETURN

```
cssm_get_master_asm_info(  
    mhandle      mhandle,  
    mbyte*       ep_seqno,  
    mschar*      ep_name  
);
```

---

#### 功能说明：

获取主 ASM 节点的组内序号和节点名称。

#### 参数说明：

mhandle：输入参数，监视器操作句柄。

ep\_seqno：输出参数，输出主 ASM 节点的组内序号。

ep\_name：输出参数，输出主 ASM 节点的名称，注意缓存长度不能小于 65，避免长度溢出。

#### 返回值：

0：执行成功。

<0: 执行失败。

## 11. cssm\_get\_master\_db\_info

### 函数原型:

---

CSSM\_RETURN

```
cssm_get_master_db_info(  
    mhandle          mhdle,  
    mschar*          group_name,  
    mbyte*           ep_seqno,  
    mschar*          ep_name  
);
```

---

### 功能说明:

获取主 DB 节点的组内序号和节点名称。

### 参数说明:

mhdle: 输入参数, 监视器操作句柄。

group\_name: 输入参数, 需要获取主 DB 信息的组名, 允许配置有多个 DB 组。

ep\_seqno: 输出参数, 输出主 DB 节点的组内序号。

ep\_name: 输出参数, 输出主 DB 节点的名称, 注意缓存长度不能小于 65, 避免长度溢出。

### 返回值:

0: 执行成功。

<0: 执行失败。

## 12. cssm\_get\_css\_auto\_flag

### 函数原型:

---

CSSM\_RETURN

```
cssm_get_css_auto_flag(  
    mhandle          mhdle,
```

---

---

```
mschar*      css_name,

mbool*       auto_flag

);
```

---

#### 功能说明:

获取指定 CSS 的监控状态。

#### 参数说明:

mhandle: 输入参数, 监视器操作句柄。

css\_name: 输入参数, 需要获取监控状态的 CSS 名称。

auto\_flag: 输出参数, 为 1 表示 CSS 监控处于打开状态, 为 0 表示 CSS 监控处于关闭状态。

#### 返回值:

0: 执行成功。

<0: 执行失败。

## 13. cssm\_get\_css\_auto\_info

#### 函数原型:

---

CSSM\_RETURN

```
cssm_get_css_auto_info(

mhandle      mhandle,

mschar*       css_name,

mbyte*        auto_flag,

mschar*       asm_name,

mbyte*        asm_auto_restart,

mschar*       db_name,

mbyte*        db_auto_restart

);
```

---

#### 功能说明:

获取指定 CSS 上的自动监控、自动拉起信息。

#### 参数说明:



mhandle: 输入参数, 监视器操作句柄。

css\_name: 输入参数, 需要获取信息的 CSS 名称。

auto\_flag: 输出参数, 为 1 表示 CSS 监控处于打开状态, 为 0 表示 CSS 监控处于关闭状态。

asm\_name: 输出参数, 当前 css 可以控制自动拉起的 asm 节点名称。

asm\_auto\_restart: 输出参数, asm\_name 当前的自动拉起状态, 为 1 表示自动拉起打开, 为 0 表示自动拉起关闭, 0xFE 表示 dmdcr.ini 中配置有自动拉起命令串, 但自动拉起检测间隔为 0, 为 0xFF 则表示 dmdcr.ini 中自动拉起参数配置错误。

db\_name: 输出参数, 当前 css 可以控制自动拉起的 db 节点名称。

db\_auto\_restart: 输出参数, db\_name 当前的自动拉起状态, 为 1 表示自动拉起打开, 为 0 表示自动拉起关闭, 0xFE 表示 dmdcr.ini 中配置有自动拉起命令串, 但自动拉起检测间隔为 0, 为 0xFF 则表示 dmdcr.ini 中自动拉起参数配置错误。

**返回值:**

0: 执行成功。

<0: 执行失败。

## 14. cssm\_set\_group\_auto\_restart

**函数原型:**

---

CSSM\_RETURN

```
cssm_set_group_auto_restart(  
  
mhandle          mhandle,  
  
mschar*          group_name,  
  
mbyte            auto_restart_flag  
  
);
```

---

**功能说明:**

设置指定组的自动拉起标记。

**参数说明:**

mhandle: 输入参数, 监视器操作句柄。

group\_name: 输入参数, 输入需要修改自动拉起标记的组名。

`auto_restart_flag`: 输入参数, 输入需要修改的值, 只能是 0 或 1, 为 0 表示关闭指定组自动拉起功能, 为 1 表示打开指定组自动拉起功能。

**返回值:**

0: 执行成功。

<0: 执行失败。

## 15. `cssm_get_n_group`

**函数原型:**

---

CSSM\_RETURN

```
cssm_get_n_group(  
    mhandle mhdle,  
    muint4*   n_group,  
    mbyte*    group_seqno,  
    mschar**  group_name,  
    mschar**  group_type  
);
```

---

**功能说明:**

获取 DSC 集群所有的组信息。

**参数说明:**

`mhdle`: 输入参数, 监视器操作句柄。

`n_group`: 输入输出参数, 输入参数指定要获取的最多组个数, 建议不小于 16, 避免取不到完整信息, 输出参数为实际获取的组个数。

`group_seqno`: 输出参数, 输出各组的序号, 参数为 `mbyte` 数组类型, 数组长度为 `n_group` 的输入值。

`group_name`: 输出参数, 输出各组的名称, 参数为字符串指针数组类型, 数组长度是 `n_group` 的输入值, 每个指针元素指向的缓存长度不能小于 65, 避免长度溢出。

`group_type`: 输出参数, 输出各组的类型, 参数为字符串指针数组类型, 数组长度是 `n_group` 的输入值, 每个指针元素指向的缓存长度不能小于 65, 避免长度溢出。

**返回值:**

0: 执行成功。  
<0: 执行失败。

## 16. cssm\_get\_n\_ep

### 函数原型:

---

CSSM\_RETURN

```
cssm_get_n_ep(  
    mhandle      mhdle,  
    mschar*      group_name,  
    muint4*      n_ep,  
    mbyte*       ep_seqno,  
    mschar**     ep_name  
);
```

---

### 功能说明:

获取指定组中的节点信息。

### 参数说明:

mhdle: 输入参数, 监视器操作句柄。

group\_name: 输入参数, 指定要获取节点信息的组名。

n\_ep: 输入输出参数, 输入参数指定要获取的最多节点个数, 建议不小于 16, 避免取不到完整信息, 输出参数为实际获取的节点个数。

ep\_seqno: 输出参数, 输出各节点的组内序号, 参数为 mbyte 数组类型, 数组长度为 n\_ep 的输入值。

ep\_name: 输出参数, 输出各节点的名称, 参数为字符串指针数组类型, 数组长度是 n\_ep 的输入值, 每个指针元素指向的缓存长度不能小于 65, 避免长度溢出。

### 返回值:

0: 执行成功。  
<0: 执行失败。

## 17. cssm\_get\_group\_info\_by\_name

### 函数原型:

---

CSSM\_RETURN

```
cssm_get_group_info_by_name(  
    mhandle          mhdle,  
    mschar*          group_name,  
    mbyte*           group_seqno,  
    mschar*          group_type,  
    mbyte*           n_ep,  
    mbyte*           ep_seqno_arr,  
    mschar**         ep_name_arr,  
    mbyte*           n_ok_ep,  
    mbyte*           ok_ep_arr,  
    mbyte*           control_ep,  
    mschar*          control_ep_name,  
    mschar*          sta,  
    mschar*          sub_sta,  
    mbyte*           break_ep,  
    mbyte*           recover_ep  
);
```

---

### 功能说明:

获取指定名称的组信息。

### 参数说明:

mhdle: 输入参数, 监视器操作句柄。

group\_name: 输入参数, 指定要获取信息的组名。

group\_seqno: 输出参数, 输出组的序号。

group\_type: 输出参数, 输出组的类型, 注意缓存长度不能小于 65, 避免长度溢出。

n\_ep: 输入输出参数, 输入参数指定最多要获取的节点个数, 建议输入值不小于 16,

避免取不到完整信息，输出参数为实际获取的节点个数。

**ep\_seqno\_arr:** 输出参数，输出各节点的组内序号，是 `mbyte` 数组类型，数组长度为 `n_ep` 的输入值。

**ep\_name\_arr:** 输出参数，输出各节点的名称，是字符串指针数组类型，数组长度为 `n_ep` 的输入值，每个指针元素指向的缓存长度不能小于 65，避免长度溢出。

**n\_ok\_ep:** 输入输出参数，输入参数指定最多要获取的正常节点个数，建议输入值不小于 16，避免取不到完整信息，输出参数为实际获取的正常节点个数，该字段只对 ASM/DB 类型的组有效，CSS 类型的组输出为 0。

**ok\_ep\_arr:** 输出参数，输出组中正常节点的组内序号，是 `mbyte` 数组类型，数组长度是 `n_ok_ep` 的输入值，该字段只对 ASM/DB 类型的组有效。

**control\_ep:** 输出参数，输出组中控制节点的组内序号。

**control\_ep\_name:** 输出参数，输出组中控制节点的名称，注意缓存长度不能小于 65，避免长度溢出。

**sta:** 输出参数，输出组的状态，该字段只对 ASM/DB 类型的组有效，CSS 类型的组输出为空串。

**sub\_sta:** 输出参数，输出组的子状态，该字段只对 ASM/DB 类型的组有效，CSS 类型的组输出为空串。

**break\_ep:** 输出参数，输出组中正在执行故障处理的节点序号，该字段只对 ASM/DB 类型的组有效，CSS 类型的组输出为 0xFF。

**recover\_ep:** 输出参数，输出组中正在执行故障恢复的节点序号，该字段只对 ASM/DB 类型的组有效，CSS 类型的组输出为 0xFF。

**返回值:**

0: 执行成功。

<0: 执行失败。

## 18. `casm_get_ep_info_by_name`

**函数原型:**

---

`CASM_RETURN`

`casm_get_ep_info_by_name(`

---

---

mhandle	mhdle,
mschar*	group_name,
mschar*	ep_name,
mbyte*	css_seqno,
mschar*	css_name,
mschar*	css_time,
mbyte*	ep_seqno,
mschar*	work_mode,
mschar*	inst_stat,
mschar*	vtd_stat,
mschar*	ok_stat,
mschar*	is_active,
mschar*	ep_guid,
mschar*	ep_ts

---

```
);
```

---

#### 功能说明:

获取指定名称的组信息。

#### 参数说明:

mhdle: 输入参数, 监视器操作句柄。

group\_name: 输入参数, 指定要获取信息的组名。

ep\_name: 输入参数, 指定要获取信息的节点名称。

css\_seqno: 输出参数, 输出取得节点信息的 CSS 序号。

css\_name: 输出参数, 输出取得节点信息的 CSS 名称, 注意缓存长度不能小于 65, 避免长度溢出。

css\_time: 输出参数, 输出取得节点信息的 CSS 当前时间, 注意缓存长度不能小于 65, 避免长度溢出。

ep\_seqno: 输出参数, 输出节点的组内序号。

work\_mode: 输出参数, 输出节点的工作模式, 注意缓存长度不能小于 65, 避免长度溢出。

inst\_stat: 输出参数, 输出节点的工作状态, 注意缓存长度不能小于 65, 避免长度溢出。

vtd\_stat: 输出参数, 输出节点在 Voting Disk 中的状态, 注意缓存长度不能小于 65, 避免长度溢出。

ok\_stat: 输出参数, 输出节点状态是否正常 (OK/ERROR), 注意缓存长度不能小于 6, 避免长度溢出。

is\_active: 输出参数, 输出节点是否处于活动状态 (TRUE/FALSE), 注意缓存长度不能小于 6, 避免长度溢出。

ep\_guid: 输出参数, 输出节点的 guid 值。

ep\_ts: 输出参数, 输出节点的时间戳值。

**返回值:**

0: 执行成功。

<0: 执行失败。

## 19. cssm\_get\_header\_config\_info

**函数原型:**

---

CSSM\_RETURN

```
cssm_get_header_config_info(  
    mhandle          mhdle,  
    muint4*          dcr_n_group,  
    mschar*          dcr_vtd_path,  
    muint8*          dcr_oguid,  
);
```

---

**功能说明:**

输出 dmdcr\_cfg.ini 中配置的全局信息。

**参数说明:**

mhdle: 输入参数, 监视器操作句柄。

dcr\_n\_group: 输出参数, 输出配置的组个数。

dcr\_vtd\_path: 输出参数, 输出 Voting Disk 路径, 注意缓存长度不能小于 257,

避免长度溢出。

dcr\_oguid: 输出参数, 输出配置的 OGUID 标识。

返回值:

0: 执行成功。

<0: 执行失败。

## 20. cssm\_get\_group\_config\_info

函数原型:

---

CSSM\_RETURN

```
cssm_get_group_config_info(  
    mhandle          mhdle,  
    mschar*          dcr_grp_name,  
    mschar*          dcr_grp_type,  
    mbyte*           dcr_grp_n_ep,  
    muint4*          dcr_grp_dskchk_cnt,  
    muint4*          dcr_grp_netchk_time  
);
```

---

功能说明:

输出 dmdcr\_cfg.ini 中配置的组信息。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

dcr\_grp\_name: 输入参数, 输入要获取信息的组名。

dcr\_grp\_type: 输出参数, 输出配置的组类型, 注意缓存长度不能小于 65, 避免长度溢出。

dcr\_grp\_n\_ep: 输出参数, 输出组中配置的节点个数。

dcr\_grp\_dskchk\_cnt: 输出参数, 输出指定组配置的磁盘心跳容错时间。

dcr\_grp\_netchk\_time: 输出参数, 输出指定组配置的网络心跳容错时间。

返回值:

0: 执行成功。



<0: 执行失败。

## 21. cssm\_get\_ep\_config\_info

### 函数原型:

---

CSSM\_RETURN

```
cssm_get_ep_config_info(  
    mhandle          mhdle,  
    mschar*          dcr_group_name,  
    mschar*          dcr_ep_name,  
    mbyte*           dcr_ep_seqno,  
    mschar*          dcr_ep_host,  
    muint4*          dcr_ep_port,  
    muint4*          dcr_ep_shm_key,  
    muint4*          dcr_ep_shm_size,  
    mschar*          dcr_ep_asm_load_path  
);
```

---

### 功能说明:

输出 dmdcr\_cfg.ini 中配置的节点信息。

### 参数说明:

mhdle: 输入参数, 监视器操作句柄。

dcr\_group\_name: 输入参数, 输入节点所在的组名。

dcr\_ep\_name: 输入参数, 输入节点名称。

dcr\_ep\_seqno: 输出参数, 输出节点的组内序号, CSS/ASM 类型的节点为自动分配的值, DB 类型的节点如果没有显式配置 DCR\_EP\_SEQNO, 也是自动分配的值, 否则为手动配置的值。

dcr\_ep\_host: 输出参数, 输出节点的 IP 地址, 对 CSS/ASM 类型的节点有效, 表示登录 CSS/ASM 节点的 IP 地址, 注意缓存长度不能小于 65, 避免长度溢出。

dcr\_ep\_port: 输出参数, 输出节点的 TCP 监听端口, 对 CSS/ASM 类型的节点有效, 对应登录 CSS/ASM 的端口号。

dcr\_ep\_shm\_key: 输出参数, 输出节点的共享内存标识, 对 ASM 类型的节点有效。

dcr\_ep\_shm\_size: 输出参数, 输出节点的共享内存大小, 单位 M, 对 ASM 类型的节点有效。

dcr\_ep\_asm\_load\_path: 输出参数, 输出节点的 ASM 磁盘扫描路径, 对 ASM 类型的节点有效, 注意缓存长度不能小于 257, 避免长度溢出。

**返回值:**

0: 执行成功。

<0: 执行失败。

## 22. cssm\_get\_monitor\_info

**函数原型:**

---

CSSM\_RETURN

```
cssm_get_monitor_info(  
    mhandle      mhdle,  
    mbyte*       css_seqno,  
    mschar*      css_name,  
    muint4*      n_mon,  
    mschar**     conn_time_arr,  
    muint8*      mid_arr,  
    mschar**     mon_ip_arr  
);
```

---

**功能说明:**

输出连接到主 CSS 上的所有监视器信息, 如果主 CSS 故障或尚未选出, 则任选一个 CSS 输出所有的连接信息。

**参数说明:**

mhdle: 输入参数, 监视器操作句柄。

css\_seqno: 输出参数, 输出取得监视器连接信息的 CSS 序号。

css\_name: 输出参数, 输出取得监视器连接信息的 CSS 名称, 注意缓存长度不能小于 65, 避免长度溢出。

`n_mon`: 输入输出参数, 输入值指定可获取的最多的监视器个数, 建议不小于 10, 避免取不到完整信息, 输出值为实际获取到的监视器个数。

`conn_time_arr`: 输出参数, 参数类型为字符串指针数组类型, 数组长度为 `n_mon` 的输入值, 输出各监视器连接到 `css_name` 的时间, 注意每个数组元素指向的缓存长度不能小于 65, 避免长度溢出。

`mid_arr`: 输出参数, 参数类型为 `muint8` 数组类型, 数组长度为 `n_mon` 的输入值, 输出各监视器的 `mid` 值。

`mon_ip_arr`: 参数类型为字符串指针数组类型, 数组长度为 `n_mon` 的输入值, 输出各监视器的 IP 地址, 注意每个数组元素指向的缓存长度不能小于 65, 避免长度溢出。

**返回值:**

0: 执行成功。

<0: 执行失败。

## 23. `cssm_css_startup`

**函数原型:**

---

`CSSM_RETURN`

```
cssm_css_startup(  
    mhandle      mhdle  
);
```

---

**功能说明:**

打开当前所有活动 CSS 的监控功能。

**参数说明:**

`mhdle`: 输入参数, 监视器操作句柄。

**返回值:**

0: 执行成功。

<0: 执行失败。

## 24. `cssm_css_stop`

### 函数原型:

---

CSSM\_RETURN

```
cssm_css_stop(  
    mhandle      mhdle  
);
```

---

### 功能说明:

关闭当前所有活动 CSS 的监控功能。

CSS 监控被关闭后，只负责调整各组节点的 `active` 状态，除此之外不会做任何自动处理，也不会自动拉起对应的节点，如果在监控关闭后，通过手动方式（非 `cssm_ep_startup` 接口方式）启动 ASM 或 DB 组，各节点也无法正常启动到 OPEN 状态。

### 参数说明:

`mhdle`: 输入参数，监视器操作句柄。

### 返回值:

0: 执行成功。

<0: 执行失败。

## 25. `cssm_open_force`

### 函数原型:

---

CSSM\_RETURN

```
cssm_open_force(  
    mhandle      mhdle,  
    mschar*      group_name  
);
```

---

### 功能说明:

强制 OPEN 指定的组。

使用场景:

在启动 ASM 或 DB 组时，如果某个节点故障一直无法启动，可借助此接口将 ASM 或 DB

组强制 OPEN。

接口会发送消息到主 CSS 执行，并且主 CSS 的监控需要处于打开状态，如果主 CSS 故障或尚未选出，则接口执行失败。

**参数说明：**

mhandle：输入参数，监视器操作句柄。

group\_name：输入参数，指定要强制 OPEN 的组名。

**返回值：**

0：执行成功。

<0：执行失败。

## 26. cssm\_ep\_startup

**函数原型：**

---

CSSM\_RETURN

```
cssm_ep_startup(  
    mhandle      mhandle,  
    mschar*      group_name  
);
```

---

**功能说明：**

启动指定 ASM 或 DB 组的所有节点。

如果 CSS 已经配置了自动重启，并且 CSS 的监控处于打开状态，则接口不允许执行，需要等待 CSS 自动检测故障并执行重启操作。

每个 CSS 只负责重启和自己的 dmdcr.ini 中配置的 DMDCR\_SEQNO 相同的 ASM 或 DB 节点，因此需要所有 CSS 都处于活动状态，否则只通知当前活动的 CSS 重启相对应的节点。

只有在 ASM 组正常启动到 OPEN 状态，并且所有活动的 ASM 节点都处于



OPEN 状态时，才允许启动 DB 组，否则执行 DB 组的启动操作会报错。

**注意：**另外在接口执行时，如果 CSS 监控处于关闭状态，则会直接打开 CSS 监控，否则 ASM 或 DB 组无法正常启动到 OPEN 状态。

**参数说明：**

mhandle: 输入参数, 监视器操作句柄。

group\_name: 输入参数, 指定要启动的组名。

返回值:

0: 执行成功。

<0: 执行失败。

## 27. cssm\_ep\_stop

函数原型:

---

CSSM\_RETURN

cssm\_ep\_stop(

mhandle            mhandle,

mschar\*            group\_name

);

---

功能说明:

退出指定的 ASM 或 DB 组, 如果主 CSS 故障或尚未退出, 则接口执行失败。

在退出 ASM 组时, 需要保证 DB 组已经退出, 否则会报错处理。



另外如果退出时, CSS 监控没有关闭, 在退出完成后, 达到 CSS 设置的  
注意: 重启间隔之后, CSS 仍然会执行自动拉起, 如果要正常退出集群, 不需要自  
动拉起, 则退出之前需要调用 `cssm_css_stop` 接口停止 CSS 的监控功能。

参数说明:

mhandle: 输入参数, 监视器操作句柄。

group\_name: 输入参数, 指定要退出的组名。

返回值:

0: 执行成功。

<0: 执行失败。

## 28. cssm\_ep\_break

函数原型:

---

CSSM\_RETURN

```
cssm_ep_break(  
  
    mhandle        mhdle,  
  
    mschar*        group_name  
  
);
```

---

#### 功能说明:

此接口只允许在主 CSS 监控关闭的情况下使用。

ASM 组或 DB 组在正常运行时, 如果某个节点出现故障, 则需要调用此接口执行故障处理, 将故障节点从组的 OK 节点数组中摘除。

在主 CSS 监控关闭的情况下, 如果 ASM 组的某个节点故障, 和故障节点

有相同 DCR\_SEQNO 的 DB 节点需要使用 `cssm_ep_halt` 接口执行强制退出,



注意:

否则 DB 节点访问对应的 ASM 文件系统失败, 也会自动 HALT。

在 ASM 和 DB 节点都出现故障的情况下, 需要先对 ASM 组执行故障处理, 再对 DB 组执行故障处理, 否则会报错不允许执行。

#### 参数说明:

mhdle: 输入参数, 监视器操作句柄。

group\_name: 输入参数, 指定要执行故障处理的组名。

#### 返回值:

0: 执行成功。

<0: 执行失败。

## 29. cssm\_ep\_recover

#### 函数原型:

---

CSSM\_RETURN

```
cssm_ep_recover(  
  
    mhandle        mhdle,  
  
    mschar*        group_name  
  
);
```

---

### 功能说明:

此接口只允许在主 CSS 监控关闭的情况下使用。

执行过故障处理的节点重启成功后，可通过调用此接口将故障节点重加入指定的 ASM 或 DB 组，重新恢复组到 OPEN 状态。



如果主 CSS 故障或尚未选出，则接口执行失败。另外只有在 ASM 组的故障节

注意：点重加入成功后，才允许重启对应的 DB 节点，并执行 DB 组的故障重加入。

### 参数说明:

mhandle: 输入参数，监视器操作句柄。

group\_name: 输入参数，指定要执行故障恢复的组名。

### 返回值:

0: 执行成功。

<0: 执行失败。

## 30. cssm\_ep\_halt

### 函数原型:

---

CSSM\_RETURN

```
cssm_ep_halt(  
  
    mhandle      mhandle,  
  
    mschar*      group_name,  
  
    mschar*      ep_name  
  
);
```

---

### 功能说明:

强制退出指定组的指定 EP。

此接口在 CSS 监控打开或关闭的情况下都允许使用，适用于下述场景：

a. 某个 ASM 或 DB 节点故障，CSS 的心跳容错时间 DCR\_GRP\_NETCHK\_TIME 或 DCR\_GRP\_DSKCHK\_CNT 配置值很大，在容错时间内，CSS 不会调整故障节点的 active 标记，一直是 TRUE，CSS 认为故障 EP 仍然处于活动状态，不会自动执行故障处理，并且



不允许手动执行故障处理。

另外执行 `cssm_ep_startup` 或 `cssm_ep_stop` 接口时，会误认为故障 EP 仍然处于活动状态，导致执行结果与预期不符。

此时可以通过执行此接口，通知 CSS 再次 HALT 故障 EP，确认 EP 已经被 HALT 后，CSS 会及时调整 `active` 标记为 `FALSE`，在此之后，对自动/手动故障处理，启动/退出 EP 节点等操作都可以正常执行。

b. 需要强制 HALT 某个正在运行的 ASM 或 DB 节点，也可以通过此接口完成。

#### 参数说明：

`mhandle`：输入参数，监视器操作句柄。

`group_name`：输入参数，指定要强制退出的节点所在的组名。

`ep_name`：输入参数，指定要强制退出的节点名称。

#### 返回值：

0：执行成功。

<0：执行失败。

## 31. `cssm_deinit`

#### 函数原型：

---

`CSSM_RETURN`

`cssm_deinit(`

`mhandle`            `mhandle`

`);`

---

#### 功能说明：

销毁监视器执行环境。

#### 参数说明：

`mhandle`：输入参数，监视器操作句柄。

#### 返回值：

0：执行成功。

<0：执行失败。

## 32. cssm\_free\_handle

函数原型:

---

CSSM\_RETURN

```
cssm_free_handle(  
    mhandle      mhdle  
);
```

---

功能说明:

释放分配到的监视器句柄。

参数说明:

mhdle: 输入参数, 分配到的句柄。

返回值:

0: 执行成功。

<0: 执行失败。

## 17.2.4 接口说明

### 1. cssm\_init\_handle

函数原型:

---

boolean

```
cssm_init_handle();
```

---

功能说明:

分配监视器操作句柄, 监视器的接口都通过此句柄调用执行。

参数说明:

无

返回值:

true: 分配成功, 返回值为监视器操作句柄。

false: 分配失败。

## 2. cssm\_get\_error\_msg\_by\_code

### 函数原型:

---

```
CssMonMsg  
  
cssm_get_error_msg_by_code(  
  
    int                code  
  
);
```

---

### 功能说明:

获取输入 code 对应的错误描述信息，code 必须是小于 0 的值。

### 参数说明:

code: 输入参数，需要获取错误信息的错误码 code，注意 code 必须是小于 0 的值。

### 返回值:

返回 CssMonMsg 对象，可通过对象成员 returnCode 获取执行结果。

0: 执行成功，可通过 CssMonMsg 的其他对象成员获取输出消息。

<0: 执行失败。

## 3. cssm\_init\_env

### 函数原型:

---

```
int  
  
cssm_init_env(  
  
    String      ini_path,  
  
    String      log_path  
  
);
```

---

### 功能说明:

初始化监视器环境。

### 参数说明:

ini\_path: 输入参数，指定 dmcsm.ini 的绝对路径。

log\_path: 输入参数，指定日志文件的存放路径，如果为 NULL 或空串，则将 dmcsm.ini 中配置的 MON\_LOG\_PATH 作为日志文件路径，如果 dmcsm.ini 中没有配

置 MON\_LOG\_PATH, 则将 dmcssm.ini 的同级目录作为日志文件路径。

**返回值:**

0: 执行成功。

<0: 执行失败, -7 表示监视器尝试建立到所有 CSS 的连接失败, 可能 CSS 都未启动, 允许忽略此错误继续执行其他操作, 如果不忽略此错误, 认定初始化失败, 则需要先正常销毁监视器环境 (调用 cssm\_deinit 接口), 再释放操作句柄 (调用 cssm\_free\_handle 接口)。

## 4. cssm\_msg\_event\_wait

**函数原型:**

---

```
void  
  
cssm_msg_event_wait();
```

---

**功能说明:**

等待消息事件。监视器接口命令执行过程中产生的中间输出消息, 以及收到 CSS 自动处理的消息时都会将消息写入到缓存并通知消息事件, 调用者只需要调用此接口等待即可, 等待事件发生后, 可通过接口 cssm\_get\_exec\_msg 去读取输出消息。

这种方式需要单独起一个线程来处理消息, 以免消息被阻塞。

**参数说明:**

无

**返回值:**

无

## 5. cssm\_get\_exec\_msg

**函数原型:**

---

```
CssMonMsg  
  
cssm_get_exec_msg();
```

---

**功能说明:**

获取输出信息, 和 cssm\_msg\_event\_wait 配合使用。

**参数说明:**

无

**返回值:**

返回 CsmMonMsg 对象, 可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过 CsmMonMsg 的其他对象成员获取输出消息。

<0: 执行失败。

## 6. cssm\_get\_msg\_exit\_flag

**函数原型:**

---

boolean

cssm\_get\_msg\_exit\_flag();

---

**功能说明:**

如果上层应用程序中有单独的消息线程, 可通过调用此接口获取退出标记, 在标记为 true 时可正常退出线程。

**参数说明:**

无

**返回值:**

true: 允许线程退出。

false: 不允许线程退出, 需要继续等待获取输出消息。

## 7. cssm\_set\_msg\_exit\_event

**函数原型:**

---

void

cssm\_set\_msg\_exit\_event();

---

**功能说明:**

设置消息退出事件。如果上层应用程序中有单独的消息线程, 在 cssm\_get\_msg\_exit\_flag 接口获取到退出标记为 true 时, 需要调用此接口设置退出事件。

参数说明:

无

返回值:

无

## 8. cssm\_msg\_event\_deinit

函数原型:

---

```
void  
cssm_msg_event_deinit();
```

---

功能说明:

通知并等待消息退出。使用等待事件方式获取监视器消息时，需要单独起一个线程，在程序结束，退出线程时，需要调用此接口通知并等待消息线程退出。

消息线程相关的接口有 `cssm_msg_event_wait`、`cssm_get_exec_msg`、`cssm_get_msg_exit_flag`、`cssm_set_msg_exit_event` 和 `cssm_msg_event_deinit`，这几个接口需要配合使用。

参数说明:

无

返回值:

无

## 9. cssm\_get\_master\_css\_info

函数原型:

---

```
CssMonMasterEpInfo  
cssm_get_master_css_info();
```

---

功能说明:

获取主 CSS 节点的组内序号和节点名称。



CSS 需要 5s 的选举时间来确定主 CSS，在 CSS 的选举时间内，此接口会执行失败，注意：行失败，获取不到主 CSS 信息。

#### 参数说明：

无

#### 返回值：

返回 `CssMonMasterEpInfo` 对象，可通过对象成员 `returnCode` 获取执行结果。

0：执行成功，可通过 `CssMonMasterEpInfo` 的其他对象成员获取主 CSS 信息。

<0：执行失败。

## 10. `cssm_get_master_asm_info`

#### 函数原型：

---

```
CssMonMasterEpInfo  
cssm_get_master_asm_info();
```

---

#### 功能说明：

获取主 ASM 节点的组内序号和节点名称。

#### 参数说明：

无

#### 返回值：

返回 `CssMonMasterEpInfo` 对象，可通过对象成员 `returnCode` 获取执行结果。

0：执行成功，可通过 `CssMonMasterEpInfo` 的其他对象成员获取主 ASM 信息。

<0：执行失败。

## 11. `cssm_get_master_db_info`

#### 函数原型：

---

```
CssMonMasterEpInfo  
cssm_get_master_db_info(  
    String          group_name
```

---

---

```
);
```

---

**功能说明：**

获取主 DB 节点的组内序号和节点名称。

**参数说明：**

group\_name: 输入参数，需要获取主 DB 信息的组名，允许配置有多个 DB 组。

**返回值：**

返回 CssMonMasterEpInfo 对象，可通过对象成员 returnCode 获取执行结果。

0: 执行成功，可通过 CssMonMasterEpInfo 的其他对象成员获取主 DB 信息。

<0: 执行失败。

## 12. cssm\_get\_css\_auto\_flag

**函数原型：**

---

```
boolean  
  
cssm_get_css_auto_flag(  
  
    String          css_name  
  
);
```

---

**功能说明：**

获取指定 CSS 的监控状态。

**参数说明：**

css\_name: 输入参数，需要获取监控状态的 CSS 名称。

**返回值：**

true: 表示 CSS 监控处于打开状态。

false: 表示 CSS 监控处于关闭状态。

## 13. cssm\_get\_css\_auto\_info

**函数原型：**

---

```
CssMonAutoInfo  
  
cssm_get_css_auto_info(  
  

```

---



---

```
String      css_name
);
```

---

**功能说明:**

获取指定 CSS 上的自动监控、自动拉起信息。

**参数说明:**

css\_name: 输入参数, 需要获取信息的 CSS 名称。

**返回值:**

返回 CsmMonAutoInfo 对象, 可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过 CsmMonAutoInfo 的其他对象成员获取相关信息。

<0: 执行失败。

## 14. csmm\_set\_group\_auto\_restart

**函数原型:**

---

```
int
csmm_set_group_auto_restart(
    String      group_name,
    byte        auto_restart_flag
);
```

---

**功能说明:**

设置指定组的自动拉起标记。

**参数说明:**

group\_name: 输入参数, 输入需要修改自动拉起标记的组名。

auto\_restart\_flag: 输入参数, 输入需要修改的值, 只能是 0 或 1, 为 0 表示关闭指定组自动拉起功能, 为 1 表示打开指定组自动拉起功能。

**返回值:**

0: 执行成功。

<0: 执行失败。

## 15. `cssm_get_n_group`

### 函数原型:

---

```
CssMonGrpNameArray  
cssm_get_n_group( );
```

---

### 功能说明:

获取 DSC 集群所有的组信息。

### 参数说明:

无

### 返回值:

返回 `CssMonGrpNameArray` 对象, 可通过对象成员 `returnCode` 获取执行结果。

0: 执行成功, 可通过 `CssMonGrpNameArray` 的其他对象成员获取组信息。

<0: 执行失败。

## 16. `cssm_get_n_ep`

### 函数原型:

---

```
CssMonEpNameArray  
cssm_get_n_ep(  
    String          group_name  
);
```

---

### 功能说明:

获取指定组中的节点信息。

### 参数说明:

`group_name`: 输入参数, 指定要获取节点信息的组名。

### 返回值:

返回 `CssMonEpNameArray` 对象, 可通过对象成员 `returnCode` 获取执行结果。

0: 执行成功, 可通过 `CssMonEpNameArray` 的其他对象成员获取节点信息。

<0: 执行失败。

## 17. `cssm_get_group_info_by_name`

### 函数原型:

---

```
CssMonGrpInfo  
  
cssm_get_group_info_by_name(  
  
    String          group_name  
  
);
```

---

### 功能说明:

获取指定名称的组信息。

### 参数说明:

`group_name`: 输入参数, 指定要获取信息的组名。

### 返回值:

返回 `CssMonGrpInfo` 对象, 可通过对象成员 `returnCode` 获取执行结果。

0: 执行成功, 可通过 `CssMonGrpInfo` 的其他对象成员获取节点信息。

<0: 执行失败。

## 18. `cssm_get_ep_info_by_name`

### 函数原型:

---

```
CssMonEpInfo  
  
cssm_get_ep_info_by_name(  
  
    String          group_name,  
  
    String          ep_name  
  
);
```

---

### 功能说明:

获取指定名称的组信息。

### 参数说明:

`group_name`: 输入参数, 指定要获取信息的组名。

`ep_name`: 输入参数, 指定要获取信息的节点名称。

### 返回值:

返回 `CssMonEpInfo` 对象，可通过对象成员 `returnCode` 获取执行结果。

0: 执行成功，可通过 `CssMonEpInfo` 的其他对象成员获取节点信息。

<0: 执行失败。

## 19. `cssm_get_header_config_info`

### 函数原型:

---

```
CssMonHeaderConfigInfo  
cssm_get_header_config_info();
```

---

### 功能说明:

输出 `dmdcr_cfg.ini` 中配置的全局信息。

### 参数说明:

无

### 返回值:

返回 `CssMonHeaderConfigInfo` 对象，可通过对象成员 `returnCode` 获取执行结果。

0: 执行成功，可通过 `CssMonHeaderConfigInfo` 的其他对象成员获取配置信息。

<0: 执行失败。

## 20. `cssm_get_group_config_info`

### 函数原型:

---

```
CssMonGroupConfigInfo  
cssm_get_group_config_info(  
    String          dcr_group_name  
);
```

---

### 功能说明:

输出 `dmdcr_cfg.ini` 中配置的组信息。

### 参数说明:

`dcr_group_name`: 输入参数，输入要获取信息的组名。

### 返回值:

返回 CcssMonGroupConfigInfo 对象,可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过 CcssMonGroupConfigInfo 的其他对象成员获取配置信息。

<0: 执行失败。

## 21. ccssm\_get\_ep\_config\_info

### 函数原型:

---

```
CcssMonEpConfigInfo  
  
ccsm_get_ep_config_info(  
  
    String      dcr_group_name,  
  
    String      dcr_ep_name  
  
);
```

---

### 功能说明:

输出 dmdcr\_cfg.ini 中配置的节点信息。

### 参数说明:

dcr\_group\_name: 输入参数, 输入节点所在的组名。

dcr\_ep\_name: 输入参数, 输入节点名称。

### 返回值:

返回 CcssMonEpConfigInfo 对象,可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过 CcssMonEpConfigInfo 的其他对象成员获取配置信息。

<0: 执行失败。

## 22. ccssm\_get\_monitor\_info

### 函数原型:

---

```
CcssMonitorInfo  
  
ccsm_get_monitor_info();
```

---

### 功能说明:

输出连接到主 CSS 上的所有监视器信息,如果主 CSS 故障或尚未选出,则任选一个 CSS

输出所有的连接信息。

**参数说明：**

无

**返回值：**

返回 `CssMonitorInfo` 对象，可通过对象成员 `returnCode` 获取执行结果。

0：执行成功，可通过 `CssMonitorInfo` 的其他对象成员获取配置信息。

<0：执行失败。

## 23. `cssm_css_startup`

**函数原型：**

---

```
int  
  
cssm_css_startup();
```

---

**功能说明：**

打开当前所有活动 CSS 的监控功能。

**参数说明：**

无

**返回值：**

0：执行成功。

<0：执行失败。

## 24. `cssm_css_stop`

**函数原型：**

---

```
int  
  
cssm_css_stop();
```

---

**功能说明：**

关闭当前所有活动 CSS 的监控功能。

CSS 监控被关闭后，只负责调整各组节点的 `active` 状态，除此之外不会做任何自动处理，也不会自动拉起对应的节点，如果在监控关闭后，通过手动方式（非

cssm\_ep\_startup 接口方式) 启动 ASM 或 DB 组, 各节点也无法正常启动到 OPEN 状态。

**参数说明:**

无

**返回值:**

0: 执行成功。

<0: 执行失败。

## 25. cssm\_open\_force

**函数原型:**

---

```
int  
  
cssm_open_force(  
  
    String          group_name  
  
);
```

---

**功能说明:**

强制 OPEN 指定的组。

使用场景:

在启动 ASM 或 DB 组时, 如果某个节点故障一直无法启动, 可借助此接口将 ASM 或 DB 组强制 OPEN。

接口会发送消息到主 CSS 执行, 并且主 CSS 的监控需要处于打开状态, 如果主 CSS 故障或尚未选出, 则接口执行失败。

**参数说明:**

group\_name: 输入参数, 指定要强制 OPEN 的组名。

**返回值:**

0: 执行成功。

<0: 执行失败。

## 26. cssm\_ep\_startup

**函数原型:**

---

```
int  
  
cssm_ep_startup(  
  
    String          group_name  
  
);
```

---

#### 功能说明:

启动指定 ASM 或 DB 组的所有节点。

如果 CSS 已经配置了自动重启, 并且 CSS 的监控处于打开状态, 则接口不允许执行, 需要等待 CSS 自动检测故障并执行重启操作。

每个 CSS 只负责重启和自己的 dmdcr.ini 中配置的 DMDCR\_SEQNO 相同的 ASM 或 DB 节点, 因此需要所有 CSS 都处于活动状态, 否则只通知当前活动的 CSS 重启相对应的节点。

只有在 ASM 组正常启动到 OPEN 状态, 并且所有活动的 ASM 节点都处于



OPEN 状态时, 才允许启动 DB 组, 否则执行 DB 组的启动操作会报错。

**注意:** 另外在接口执行时, 如果 CSS 监控处于关闭状态, 则会直接打开 CSS 监控, 否则 ASM 或 DB 组无法正常启动到 OPEN 状态。

#### 参数说明:

group\_name: 输入参数, 指定要启动的组名。

#### 返回值:

0: 执行成功。

<0: 执行失败。

## 27. cssm\_ep\_stop

#### 函数原型:

---

```
int  
  
cssm_ep_stop(  
  
    String          group_name  
  
);
```

---

#### 功能说明:

退出指定的 ASM 或 DB 组, 如果主 CSS 故障或尚未选出, 则接口执行失败。



在退出 ASM 组时，需要保证 DB 组已经退出，否则会报错处理。



另外如果退出时，CSS 监控没有关闭，在退出完成后，达到 CSS 设置的注意：重启间隔之后，CSS 仍然会执行自动拉起，如果要正常退出集群，不需要自动拉起，则退出之前需要调用 `cssm_css_stop` 接口停止 CSS 的监控功能。

#### 参数说明：

`group_name`：输入参数，指定要退出的组名。

#### 返回值：

0：执行成功。

<0：执行失败。

## 28. `cssm_ep_break`

#### 函数原型：

---

```
int  
  
cssm_ep_break(  
  
    String          group_name  
  
);
```

---

#### 功能说明：

此接口只允许在主 CSS 监控关闭的情况下使用。

ASM 组或 DB 组在正常运行时，如果某个节点出现故障，则需要调用此接口执行故障处理，将故障节点从组的 OK 节点数组中摘除。

在主 CSS 监控关闭的情况下，如果 ASM 组的某个节点故障，和故障节点有相同 DCR\_SEQNO 的 DB 节点需要使用 `cssm_ep_halt` 接口执行强制退出，否则 DB 节点访问对应的 ASM 文件系统失败，也会自动 HALT。



注意：

在 ASM 和 DB 节点都出现故障的情况下，需要先对 ASM 组执行故障处理，再对 DB 组执行故障处理，否则会报错不允许执行。

#### 参数说明：

`group_name`：输入参数，指定要执行故障处理的组名。

返回值:

0: 执行成功。

<0: 执行失败。

## 29. `cssm_ep_recover`

函数原型:

---

```
int  
  
cssm_ep_recover(  
  
    String          group_name  
  
);
```

---

功能说明:

此接口只允许在主 CSS 监控关闭的情况下使用。

执行过故障处理的节点重启成功后, 可通过调用此接口将故障节点重加入指定的 ASM 或 DB 组, 重新恢复组到 OPEN 状态。



如果主 CSS 故障或尚未选出, 则接口执行失败。另外只有在 ASM 组的故障节

注意: 点重加入成功后, 才允许重启对应的 DB 节点, 并执行 DB 组的故障重加入。

参数说明:

`group_name`: 输入参数, 指定要执行故障恢复的组名。

返回值:

0: 执行成功。

<0: 执行失败。

## 30. `cssm_ep_halt`

函数原型:

---

```
int  
  
cssm_ep_halt(  
  
    String          group_name,
```

---

---

```
String      ep_name
);
```

---

#### 功能说明:

强制退出指定组的指定 EP。

此接口在 CSS 监控打开或关闭的情况下都允许使用，适用于下述场景：

a. 某个 ASM 或 DB 节点故障，CSS 的心跳容错时间 DCR\_GRP\_NETCHK\_TIME 或 DCR\_GRP\_DSKCHK\_CNT 配置值很大，在容错时间内，CSS 不会调整故障节点的 active 标记，一直是 TRUE，CSS 认为故障 EP 仍然处于活动状态，不会自动执行故障处理，并且不允许手动执行故障处理。

另外执行 cssm\_ep\_startup 或 cssm\_ep\_stop 接口时，会误认为故障 EP 仍然处于活动状态，导致执行结果与预期不符。

此时可以通过执行此接口，通知 CSS 再次 HALT 故障 EP，确认 EP 已经被 HALT 后，CSS 会及时调整 active 标记为 FALSE，在此之后，对自动/手动故障处理，启动/退出 EP 节点等操作都可以正常执行。

b. 需要强制 HALT 某个正在运行的 ASM 或 DB 节点，也可以通过此接口完成。

#### 参数说明:

group\_name: 输入参数，指定要强制退出的节点所在的组名。

ep\_name: 输入参数，指定要强制退出的节点名称。

#### 返回值:

0: 执行成功。

<0: 执行失败。

## 31. cssm\_deinit\_env

#### 函数原型:

---

```
int
cssm_deinit_env();
```

---

#### 功能说明:

销毁监视器执行环境。

#### 参数说明:

无

**返回值:**

0: 执行成功。

<0: 执行失败。

## 32. cssm\_free\_handle

**函数原型:**

---

int

cssm\_free\_handle();

---

**功能说明:**

释放分配到的监视器句柄。

**参数说明:**

无

**返回值:**

0: 执行成功。

<0: 执行失败。

## 17.2.5 编程示例

这里给出 C 程序的编程示例，运行环境为 VS2010，通过配置项目属性，使用 dmcssm\_dll.lib 隐式加载 dll 的方式，注意此处仅为使用示例，实际使用时可以根据自已的需求再做调整。

```
#include "cssm_dll.h"
#include "stdio.h"
#include "stdlib.h"
#include "windows.h"
#include "process.h"

/* 消息线程 */
DWORD
WINAPI
cssm_get_msg_thread(
mhandle      handle
```

```

)
{
    mschar      buf_msg[4097];
    muint4      msg_len_out;
    muint4      get_flag;
    mbool       exit_flag;
    CSSM_RETURN  code;

while (TRUE)
    {
        /* 等待消息事件 */
        cssm_msg_event_wait(handle);

        /* 获取退出标记 */
        cssm_get_msg_exit_flag(handle, &exit_flag);
if (exit_flag == TRUE)
    {
        break;
    }

do
    {
        /* 获取并打印消息 */
code = cssm_get_exec_msg(handle, buf_msg, 4097, &msg_len_out, &get_flag);
if (code >=0 && msg_len_out > 0)
    {
fprintf(stdout, "%s", buf_msg);
    }

        } while (get_flag); /* 判断是否还有未读消息 */
    }

    /* 设置退出事件 */
    cssm_set_msg_exit_event(handle);

return 0;
}

/* 清理环境 */
void
cssm_clear_env(
mhandle      handle
)
{

```

```

/* 通知并等待 dmmon_get_msg_thread 消息线程退出 */
cssm_msg_event_deinit(handle);

/* 销毁监视器环境 */
cssm_deinit(handle);

/* 释放句柄 */
cssm_free_handle(handle);
}

int
main()
{
    CSSM_RETURN    ret;
    mhandle        handle;
    HANDLE         thread_handle;
    char           msg[4097];
    int            msg_len;
    uint4          n_group;
    mbyte          group_seqno[16];
    mschar*        group_name[16];
    mschar*        group_type[16];
    msint2         offset;
    mschar         mem[4096];
    msint2         i;

    /* 分配操作句柄 */
    ret = cssm_alloc_handle(&handle);
    if (ret < 0)
    {
        fprintf(stdout, "dmcssm alloc handle failed!\n");

        /* 获取 ret 对应的错误描述信息 */
        cssm_get_error_msg_by_code(handle, ret, msg, 4097, &msg_len);
        if (msg_len > 0)
        {
            fprintf(stdout, "code:%d, error msg:%s", ret, msg);
        }
    }
    return ret;
}

/* 创建消息线程 */
thread_handle = (HANDLE)_beginthreadex(NULL, 0, cssm_get_msg_thread, handle,
0, NULL);
if (thread_handle == NULL)

```

```

    {
ret = GetLastError();
fprintf(stderr, "_beginthreadex error! desc:%s, code:%d\n", strerror(ret), ret);

return ret;
    }
    /* 初始化监视器环境，可以选择忽略-8 错误（建立到所有 CSS 连接失败，可能当前 CSS 都还未启动） */
ret = cssm_init(handle, "D:\\dmcsm\\dmcsm.ini", "D:\\dmcsm\\log");
if (ret < 0 && ret != -8)
    {
fprintf(stdout, "dmcsm init failed!\n");
        /* 获取 ret 对应的错误描述信息 */
        cssm_get_error_msg_by_code(handle, ret, msg, 4097, &msg_len);
if (msg_len > 0)
    {
fprintf(stdout, "code:%d, error msg:%s", ret, msg);
    }
        cssm_clear_env(handle);
return ret;
    }

offset = 0;
for (i = 0; i < 16; i++)
    {
        group_name[i] = mem + offset;
offset += 129;

        group_type[i] = mem + offset;
offset += 65;
    }
    n_group = 16;
ret = cssm_get_n_group(handle, &n_group, group_seqno, group_name,
group_type);
if (ret < 0)
    {
fprintf(stdout, "dmcsm get group info failed!\n");
        /* 获取 ret 对应的错误描述信息 */
        cssm_get_error_msg_by_code(handle, ret, msg, 4097, &msg_len);
if (msg_len > 0)
    {
fprintf(stdout, "code:%d, error msg:%s", ret, msg);
    }
    }
}

```

```

else
{
fprintf(stdout, "Get group info succeed, n_group: %d\n\n", n_group);
}
/* 执行其他接口调用 */
/*...*/
/* 清理环境 */
cssm_clear_env(handle);
system("pause");

return 0;
}

```

## 17.2.6 Java 编程示例

这里给出Java程序的编程示例,运行环境为Eclipse,使用加载监视器jar包的方式, jar包名称为com.dameng.jni\_7.0.0.jar, 注意此处仅为使用示例, 实际使用时可以根据自己的需求再做调整。

```

package com.dameng.test;
import com.dameng.cssm.*;
public class CssMonTest {
    public CssMonDLL monDll = null;
    public CssMonMsg monMsg = null;
    public Thread msgThread = null;
    public String ini_path = "D:\\dsc\\dmcssm.ini";
    public String log_path = "D:\\dsc\\log";
    //接收消息线程
    public void startThread()
    {
        msgThread = new Thread(new Runnable()
        {
            @Override
public void run()
        {
            while (true)
            {
                //等待消息事件
                monDll.cssm_msg_event_wait();
                //获取退出标记
                if (monDll.cssm_get_msg_exit_flag() == true)
                    break;
            }
        }
    }
}

```



```

        {
            //获取并打印消息
            monMsg = monDll.cssm_get_exec_msg();
            if(monMsg.getReturnCode() < 0)
                break;
            if(monMsg.getMsg().length() > 0)
            {
                System.out.print(monMsg.getMsg());
            }
            } while (monMsg.getMsgFlag() == 1); //是否还有未读消息
        }
//设置退出标记
monDll.cssm_set_msg_exit_event();
    }
});
//启动线程
msgThread.start();
}
//退出线程
public void endThread() throws InterruptedException
{
    if (msgThread != null && msgThread.isAlive())
    {
        //通知并等待消息线程退出
        monDll.cssm_msg_event_deinit();
    }
}
public void test() throws InterruptedException
{
    boolean ret1 = false;
    int ret2 = 0;
    CssMonMsg retMsg = null;
    CssMonGrpNameArray grpArr = null;
    monDll = new CssMonDLL();
    //分配句柄
    ret1 = monDll.cssm_init_handle();
    if (ret1 == false)
    {
        System.out.println("dmcssm alloc handle failed!");
        return;
    }
    else
    {
        System.out.println("dmcssm alloc handle success!");
    }
}

```

```

    }
    //启动消息接收线程
    startThread();
    //初始化监视器，可以选择忽略-7 错误（建立到所有 CSS 连接失败，可能当前 CSS 都还未启
动）

    ret2 = monDll.cssm_init_env(ini_path, log_path);
    if (ret2 < 0 && ret2 != -7)
    {
        System.out.println("dmcssm init failed!");
        retMsg = monDll.cssm_get_error_msg_by_code(ret2);
        if (retMsg.getReturnCode() >= 0)
        {
            System.out.println("code:" + ret2 + "," + retMsg.getMsg());
        }
        //初始化失败，退出线程，销毁句柄
        endThread();
        monDll.cssm_free_handle();
        return;
    }
    else
    {
        System.out.println("dmcssm init success!");
    }
    grpArr = monDll.cssm_get_n_group();
    if(grpArr.getReturnCode() < 0)
    {
        System.out.println("get dmcssm group info failed!");

        retMsg
monDll.cssm_get_error_msg_by_code(grpArr.getReturnCode());
        if (retMsg.getReturnCode() >= 0)
        {
            System.out.println("code:" + grpArr.getReturnCode() + "," +
retMsg.getMsg());
        }
    }
    else
    {
        System.out.println("dmcssm get group info success, n_group:" +
grpArr.getN_group());
    }
    /* 执行其他接口调用 */
    /*...*/
    //退出消息线程

```

```

        endThread();
        //销毁监视器环境
        monDll.cssm_deinit_env();
        //销毁句柄
        monDll.cssm_free_handle();
        System.out.println("\ndmcsm deinit success!");
    }
    /**
     * @param args
     * @throws InterruptedException
     */
    public static void main(String[] args) throws InterruptedException {
        CssMonTest test = new CssMonTest();
        test.test();
    }
}

```

## 17.2.7 错误码汇编

代码	解释
-1	普通错误
-2	无效的句柄
-3	初始化日志信息失败
-4	初始化系统信息失败
-5	读取 dmcsm.ini 文件失败（路径错误或配置错误或没有权限）
-6	无效的日志文件路径
-7	建立到所有 CSS 的连接失败，可能 CSS 都未启动，由用户决定是否忽略此错误码继续往下执行，如果不忽略此错误，认定初始化失败，则需要先销毁监视器环境，再释放操作句柄。
-8	非法的参数，参数为空或长度超长或值非法
-9	没有找到 code 对应的错误信息
-10	内存不足
-11	监视器已达到最大个数（同一套 DSC 集群最多允许同时启动 10 个监视器）
-1001	用户自定义异常或者未知错误
-1003	根据 ep_seqno 获取发送端口失败
-1004	到 dmcsm 的连接断开

-1005	有消息正在发送中,您的命令当前不能执行
-1007	设置 CSS 的 MID 命令执行失败
-1009	监视器操作冲突
-1012	CSS 执行失败
-1017	通知 CSS 执行命令失败
-1018	获取有效的 CSS 通信端口失败, CSS 尚未启动或监视器配置错误
-1019	无效的组名或尚未收到消息
-1020	接收 CSS 消息超时
-1022	非法的 DCR 类型
-1023	只有主 CSS 能执行该操作
-1024	CSS 中没有找到指定类型的组信息
-1025	执行 OPEN FORCE 的组当前不是 SLAVE STARTUP 状态
-1026	无效的 EP 名称或尚未收到消息
-1027	未找到指定类型的组信息
-1028	检测到主 CSS 发生变化
-1029	当前不存在活动的 CSS
-1031	通知 CSS 打开监控失败
-1033	通知 CSS 关闭监控失败
-1035	命令执行失败
-1036	组当前没有活动 EP
-1037	当前存在有活动的 DB 节点, 需要先退出对应的 DB 组, 才允许退出 ASM 组
-1038	组中的 EP 已处于 ACTIVE 状态 (只收集活动 CSS 对应的 EP), 或者 CSS 配置有自动重启, 请等待 CSS 自动检测重启
-1039	当前活动的 CSS 监控都已处于打开状态
-1040	当前活动的 CSS 监控都已处于关闭状态
-1041	获取 CSS 的通信端口失败
-1042	通知 CSS 执行 EP STARTUP 失败
-1045	CSS 中没有找到指定组的信息
-1046	CSS 对指定的组配置了自动重启, 请等待 CSS 自动检测重启对应的 EP

-1047	EP 已经处于 ACTIVE 状态
-1048	ASM 实例未处于 ACTIVE 状态
-1049	ASM 组未处于 OPEN 状态
-1050	指定组中没有活动 EP
-1051	当前存在有活动的 DB 节点，需要先退出对应的 DB 组
-1052	CSS 监控处于关闭状态，不允许执行此命令
-1053	CSS 类型的组不允许执行此命令
-1057	获取主 CSS 通信端口失败，主 CSS 尚未选出或 CSS 未启动或者监视器配置错误
-1058	ASM 组没有活动节点或者活动节点不是 OPEN 状态，或者 ASM 组不是 OPEN 状态，不允许启动 DB 组
-1059	检测到主 CSS 发生变化，命令中断执行
-1065	关闭 CSS 监控失败
-1067	打开 CSS 监控失败
-1069	指定组不是 SLAVE STARTUP 状态，不允许执行 OPEN FORCE
-1070	CSS 监控处于打开状态，不允许手动执行此命令，请等待 CSS 自动处理
-1071	CSS 当前正在做故障处理或故障恢复
-1072	CSS 监控处于打开状态，不允许执行此命令
-1073	没有找到满足故障恢复条件的节点
-1074	CSS 当前正在做故障处理或故障恢复
-1075	没有找到满足故障处理条件的节点
-1076	需要先对 ASM 组做故障处理
-1077	CSS 对指定的组没有配置自动重启参数
-1078	监视器超时仍未等到组中的所有 EP 执行成功，命令执行失败
-1079	监视器超时仍未等到当前 EP 执行成功，命令执行失败
-1080	控制节点尚未选出，获取信息失败
-1081	无效的节点名或尚未收到消息
-1082	指定节点的组内序号非法
-1083	组中的 EP 不是活动状态
-1085	对应的 ASM 组中没有活动节点，无法执行故障处理

-1086	指定组中没有活动节点，无法执行故障处理
-1087	没有找到满足故障恢复条件的节点
-1088	需要先对 ASM 组做故障处理
-1089	监视器正在执行命令，请稍候重试
-1090	CSS 没有对组配置重启参数
-1091	非法的 EP 序号
-1092	对 EP 写入 HALT 命令成功，请等待处理结果
-1094	ASM 组中没有活动 EP，不允许执行故障处理
-1095	指定组中没有活动 EP，不允许执行故障处理
-1096	打开 CSS 对节点的自动拉起成功
-1097	关闭 CSS 对节点的自动拉起成功
-1098	组中没有非活动 EP
-1521	DSC 环境同一机器上的节点不允许配置相同的归档路径

咨询热线：400-991-6599

技术支持：dmtech@dameng.com

官网网址：www.dameng.com



武汉达梦数据库有限公司  
Wuhan Dameng Database Co.,Ltd.

地址：武汉市东湖新技术开发区高新大道999号未来科技大厦C3栋16—19层

16th-19th Floor, Future Tech Building C3, No.999 Gaoxin Road, Donghu New Tech Development Zone,Wuhan,Hubei Province,China

电话：(+86) 027-87588000 传真：(+86) 027-87588810

---