

达梦技术丛书

DM8 数据守护与读写分离集群 V4.0



前言

概述

本文档主要介绍 DM 数据守护的系统特性、基本概念、主要功能、各组成部件的详细介绍，以及如何搭建数据守护环境并使用等。

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
 警告：	表示可能导致系统损坏、数据丢失或不可预知的结果。
 注意：	表示可能导致性能降低、服务不可用。
 小窍门：	可以帮助您解决某个问题或节省您的时间。
 说明：	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 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

技术服务电话：400-991-6599

技术服务邮箱：dmtech@dameng.com

目录

1 引言.....	1
2 概述.....	3
2.1 系统特性.....	5
2.1.1 主要特性.....	5
2.1.2 主要改进（V2.0 版本）.....	7
2.1.3 主要改进（V2.1 版本）.....	8
2.1.4 兼容性说明（V2.1 版本）.....	9
2.1.5 主要改进（V3.0 版本）.....	10
2.1.6 兼容性说明（V3.0 版本）.....	11
2.1.7 主要改进（V4.0 版本）.....	11
2.1.8 兼容性说明（V4.0 版本）.....	12
2.2 基本概念.....	14
2.2.1 数据库.....	14
2.2.2 DMDSC 状态.....	14
2.2.3 数据库模式.....	14
2.2.4 数据库状态.....	15
2.2.5 LSN 介绍.....	17
2.2.6 Redo 日志.....	19
2.2.7 Redo 日志包（RLOG_PKG）.....	20
2.2.8 包序号介绍.....	21
2.2.9 KEEP_PKG 介绍.....	22
2.2.10 联机 Redo 日志文件.....	23
2.2.11 归档介绍.....	24
2.2.12 MAL 系统.....	30
2.2.13 OGUID.....	31
2.2.14 守护进程组.....	31
2.2.15 组分裂.....	31
2.2.16 脑裂.....	31
2.3 术语定义.....	32
2.4 实时主备.....	34
2.4.1 主要功能.....	34
2.4.2 归档流程.....	35
2.5 MPP 主备.....	36
2.5.1 功能扩展.....	37
2.5.2 dmmpp.ctl 维护.....	37
2.6 读写分离集群.....	39
2.6.1 归档流程.....	39

2.6.2 实现原理.....	41
2.6.3 事务一致性.....	42
2.6.4 性能调整.....	44
2.6.5 实时归档的读写分离.....	45
2.7 DMDSC 数据守护	45
2.7.1 总体结构.....	46
2.7.2 系统连接.....	47
2.7.3 归档配置.....	48
2.7.4 日志发送.....	48
2.7.5 重演实例.....	48
2.7.6 备库日志重演.....	49
2.7.7 守护控制文件.....	49
2.7.8 远程归档修复.....	49
2.7.9 DMDSC（主）和单节点（备）	51
2.7.10 单节点（主）和 DMDSC（备）	52
2.7.11 DMDSC 集群的管理规则	52
2.7.12 场景说明	57
2.8 异步备库.....	61
2.9 Huge 表	62
3 守护进程.....	63
3.1 主要功能.....	63
3.1.1 监控数据库实例	63
3.1.2 发送状态信息.....	64
3.1.3 监控其他守护进程.....	64
3.1.4 接收监视器消息.....	64
3.1.5 主备库启动运行	65
3.1.6 备库故障处理.....	65
3.1.7 备库异常处理.....	66
3.1.8 主库故障处理.....	67
3.1.9 故障恢复处理.....	67
3.2 守护类型.....	70
3.3 守护模式.....	71
3.4 守护状态.....	72
3.5 控制文件.....	74
3.6 可加入（分裂）判断规则	75
3.7 守护进程命令	76
4 监视器.....	80
4.1 监视器类型.....	81

4.2 状态确认.....	81
4.3 自动接管.....	82
4.4 监视器命令.....	84
4.5 监视器 LOG 日志.....	113
5 配置文件说明	114
5.1 dm.ini.....	114
5.2 dmmal.ini.....	118
5.3 dmarch.ini	121
5.4 dmwatcher.ini	122
5.5 dmmonitor.ini	125
5.6 dmtimer.ini.....	128
5.7 端口配置关系说明.....	131
5.8 服务名配置.....	135
5.9 建库参数说明.....	136
6 数据守护使用说明.....	137
6.1 正常运行状态.....	137
6.2 数据守护的启动.....	137
6.3 强制 Open 数据库	138
6.4 关闭数据守护系统.....	139
6.5 主备库切换.....	141
6.6 主库故障、备库接管	143
6.7 备库强制接管.....	144
6.8 主库故障重启（备库接管前重启）	145
6.9 备库故障处理.....	145
6.10 公共网络故障	147
6.11 内部网络故障	147
6.12 备库异常处理	149
6.13 故障库数据同步	149
6.14 备库重建	150
6.15 守护进程组分裂	152

6.16 dmmppctl 不一致	152
6.17 确认监视器未启动	154
6.18 备库维护	154
6.19 滚动升级	155
6.20 实时/读写分离/MPP 备库数据同步情况分析	157
6.21 异步备库数据同步情况分析	159
6.22 MPP 主备版本升级（从 V2.0 升级）	160
6.23 MPP 主备限制登录说明	160
6.24 备库查询限制说明	160
6.25 异步备库延时和定点重演	161
6.26 备库归档磁盘空间不足	163
6.27 注意事项	164
7 数据守护搭建	168
7.1 数据准备	168
7.1.1 脱机备份、脱机还原方式	169
7.1.2 联机备份、脱机还原方式	169
7.2 配置实时主备	170
7.2.1 环境说明	171
7.2.2 数据准备	172
7.2.3 配置主库 GRP1_RT_01	172
7.2.4 配置备库 GRP1_RT_02	176
7.2.5 配置监视器	179
7.2.6 启动守护进程	182
7.2.7 启动监视器	182
7.3 配置读写分离集群	182
7.3.1 环境说明	183
7.3.2 数据准备	184
7.3.3 配置主库 GRP1_RWW_01	185
7.3.4 配置备库 GRP1_RWW_02	188
7.3.5 配置备库 GRP1_RWW_03	192
7.3.6 配置监视器	196
7.3.7 启动守护进程	198
7.3.8 启动监视器	198
7.3.9 接口说明	199
7.4 配置 MPP 主备	202
7.4.1 环境说明	202

7.4.2 数据准备.....	203
7.4.3 配置主库 GRP1_MPP_EP01.....	204
7.4.4 配置主库 GRP2_MPP_EP02.....	208
7.4.5 配置备库 GRP1_MPP_EP11.....	210
7.4.6 配置备库 GRP2_MPP_EP22.....	212
7.4.7 配置 dmwatcher.ini.....	214
7.4.8 配置监视器.....	216
7.4.9 启动守护进程.....	219
7.4.10 启动监视器.....	220
7.5 配置 DMDSC 主备环境.....	220
7.5.1 配置说明.....	220
7.5.2 环境说明.....	221
7.5.3 配置 DMDSC 主库环境.....	222
7.5.4 配置单节点备库.....	224
7.5.5 配置 dm.ini.....	224
7.5.6 配置 dmmal.ini.....	225
7.5.7 配置 dmarch.ini.....	226
7.5.8 配置 dmwatcher.ini.....	228
7.5.9 配置 dmmonitor.ini.....	230
7.5.10 配置 dmdcr.ini.....	230
7.5.11 启动主备库.....	231
7.5.12 设置 OGUID.....	232
7.5.13 修改主备库模式.....	232
7.5.14 启动守护进程.....	232
7.5.15 启动监视器.....	233
7.6 配置异步备库.....	233
7.6.1 环境说明.....	233
7.6.2 数据准备.....	234
7.6.3 配置主库 GRP1_RT_01.....	234
7.6.4 配置备库 GRP1_RT_02.....	236
7.6.5 配置异步备库 GRP1_LOCAL_01.....	238
7.6.6 配置监视器.....	240
7.6.7 启动守护进程.....	242
7.6.8 启动监视器.....	243
7.7 注册服务.....	243
7.8 动态增加读写分离集群节点.....	243
7.8.1 数据准备.....	244
7.8.2 配置新备库.....	244
7.8.3 动态添加 MAL 配置.....	248
7.8.4 动态添加归档配置.....	248
7.8.5 修改监视器 dmmonitor.ini.....	249
7.8.6 启动所有守护进程以及监视器.....	249

7.9 动态增加实时备库	250
7.9.1 数据准备	250
7.9.2 配置新备库	251
7.9.3 动态添加 MAL 配置	254
7.9.4 动态添加归档配置	254
7.9.5 修改监视器 dmmonitor.ini	255
7.9.6 启动所有守护进程以及监视器	255
7.10 动态增加实时 DMDSC 备库	255
7.10.1 数据准备	256
7.10.2 配置新备库 DMDSC 环境	257
7.10.3 动态添加 MAL 配置	264
7.10.4 动态添加归档配置	264
7.10.5 修改监视器 dmmonitor.ini	265
7.10.6 启动所有守护进程以及监视器	265
8 利用 DEM 工具搭建数据守护	266
8.1 实时主备管理	266
8.1.1 部署	267
8.1.2 监控	278
9 版本升级	280
9.1 日志格式升级	280
9.1.1 版本说明	280
9.1.2 升级步骤	281
9.2 回滚管理段升级	284
9.2.1 版本说明	285
9.2.2 升级步骤	285
9.3 数据页格式升级	286
9.3.1 版本说明	286
9.3.2 升级步骤	287
9.4 HUGE 表空间升级	287
9.4.1 版本说明	287
9.4.2 升级步骤	288
10 附录	290
10.1 系统视图	290
10.2 监视器接口	306
10.2.1 DLL 依赖	306
10.2.2 返回值说明	306
10.2.3 接口调整说明 (V4.0)	307
10.2.4 C 接口说明	309

10.2.5 JNI 接口说明	398
10.2.6 C 编程示例	455
10.2.7 Java 编程示例	473
10.2.8 错误码汇编	477

1 引言

DM 数据守护 (Data Watch) 是一种集成化的高可用、高性能数据库解决方案，是数据库异地容灾的首选方案。通过部署 DM 数据守护，可以在硬件故障（如磁盘损坏）、自然灾害（地震、火灾）等极端情况下，避免数据损坏、丢失，保障数据安全，并且可以快速恢复数据库服务，满足用户不间断提供数据库服务的要求。与常规的数据库备份 (Backup)、还原 (Restore) 技术相比，数据守护可以更快地恢复数据库服务。随着数据规模不断增长，通过还原手段恢复数据，往往需要数个小时、甚至更长时间，而数据守护基本不受数据规模的影响，只需数秒时间就可以将备库切换为主库对外提供数据库服务。

DM数据守护提供多种解决方案，可以配置成实时主备、MPP主备、DMDSC主备或读写分离集群，满足用户关于系统可用性、数据安全性、性能等方面的综合需求，有效降低总体投入，获得超值的投资回报。

实时主备由一个主库以及一个或者多个配置了实时 (Realtime) 归档的备库组成，其主要目的是保障数据库可用性，提高数据安全性。实时主备系统中，主库提供完整的数据库功能，备库提供只读服务。主库修改数据产生的Redo日志，通过实时归档机制，在写入联机Redo日志文件之前发送到备库，实时备库通过重演Redo日志与主库保持数据同步。当主库出现故障时，备库在将所有Redo日志重演结束后，就可以切换为主库对外提供数据库服务。

MPP主备就是在MPP集群的基础上，为每一个MPP节点配置一套实时主备系统，这些实时主备系统一起构成了MPP主备系统。我们将一个MPP节点对应的主备系统称为一个数据守护组 (Group)，MPP主备系统中各个数据守护组保持相对独立，当某个MPP主节点出现故障时，在其对应的数据守护组内挑选一个备库切换为主库后，就可以确保整个MPP集群的正常使用。

DMDSC主备与单节点主备功能一致，DMDSC主备支持DMDSC集群和单节点之间互为主备库，一般建议将DMDSC集群部署为主库，将单节点部署为备库。当DMDSC集群为主库时，DMDSC集群控制节点收集所有节点的Redo日志发送到备库，备库严格按照各节点修改数据页的先后顺序重演Redo日志保持数据同步；当DMDSC集群为备库时，主库将Redo日志发送至DMDSC集群控制节点，DMDSC集群控制节点重演Redo日志保持数据同步。

读写分离集群由一个主库以及一个或者多个配置了即时（Timely）归档或实时（Realtime）归档的备库组成，其主要目标是在保障数据库可用性基础上，实现读、写操作的自动分离，进一步提升数据库的业务支撑能力。读写分离集群通过配置事务一致模式保证主、备库数据一致性，并配合达梦数据库管理系统的各种接口（JDBC、DPI等），将只读操作自动分流到备库，有效降低主库的负载，提升系统吞吐量。

2 概述

DM 数据守护（DM Data Watch）的实现原理非常简单：将主库（生产库）产生的 Redo 日志传输到备库，备库接收并重新应用 Redo 日志，从而实现备库与主库的数据同步。DM 数据守护的核心思想是监控数据库状态，获取主、备库数据同步情况，为 Redo 日志传输与重演过程中出现的各种异常情况提供一系列的解决方案。

DM 数据守护系统结构参考图 2.1。主要由主库、备库、Redo 日志、Redo 日志传输、Redo 日志重演、守护进程（dmwatcher）、监视器（dmmonitor）组成。

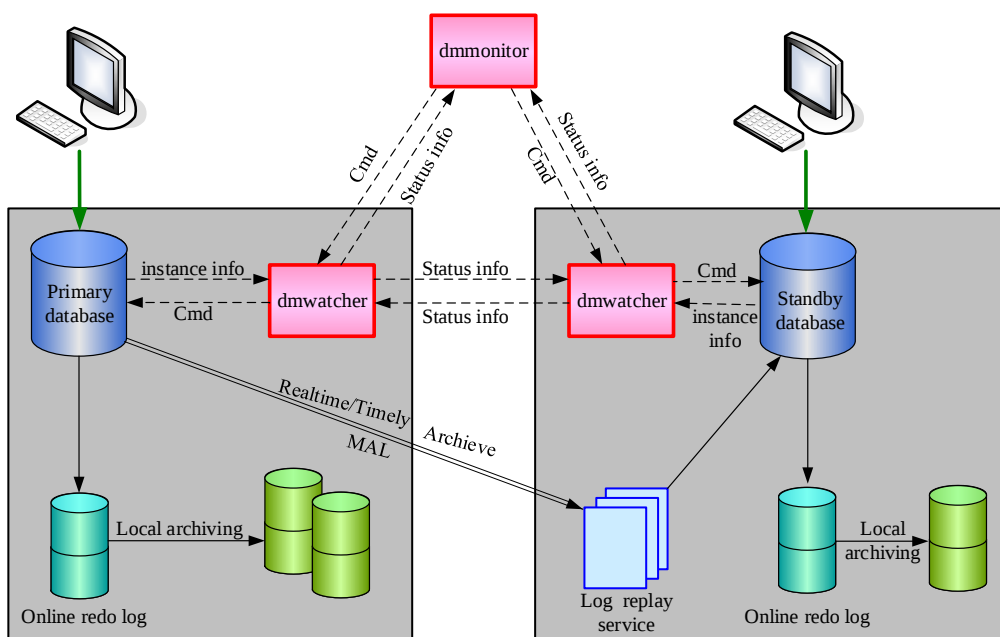


图 2.1 数据守护系统结构图

数据库与数据库实例

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），保存在物理磁盘或文件系统中。

数据库实例（Instance）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。

本文档将不再严格区分数据库和数据库实例的概念，很多地方会笼统的以库来代替。考虑到数据守护系统中，数据库实例名是唯一的，为了更准确地进行描述，很多情况下我们会以实例 xxx 来标记某一个主库或者备库。

主库

Primary 模式，提供完整数据库服务的实例，一般来说主库是用来直接支撑应用系统的生产库。

备库

Standby 模式，提供只读数据库服务的实例。备库除了用于容灾，还可以提供备份、查询等只读功能，并且备库还支持临时表的 Insert/Delete/Update 操作。

备库支持临时表修改主要基于两个因素：1. 临时表数据的修改不会产生 Redo 日志，主库对临时表的修改无法同步到备库；2. 可以提供更大灵活性，适应更多应用场景。

根据数据同步情况，备库又可以分为可切换备库和不可切换备库。可切换备库是指，主备库之间数据完全同步，主库发生故障、备库切换为主库后，不会造成任何数据丢失的备库。

Redo 日志

Redo 日志记录物理数据页内容变动情况，是数据库十分重要的一个功能，在数据库系统故障（比如服务器掉电）重启时，利用 Redo 日志可以把数据恢复到故障前的状态。

Redo 日志也是数据守护的实现基础，数据库中 Insert、Delete、Update 等 DML 操作以及 Create TABLE 等 DDL 操作最终都会体现为对某一个或者多个物理数据页的修改，因此备库通过重做 Redo 日志可以与主库数据保持一致。

Redo 日志传输

主备库之间的 Redo 日志传输，以日志包 RLOG_PKG 为单位，主库通过 MAL 系统发送 Redo 日志到备库。各种不同数据守护类型的区别，就在于主库日志包 RLOG_PKG 的发送时机，以及备库收到 Redo 日志后的处理策略。

Redo 日志重演

Redo 日志重演的过程，就是备库收到主库发送的 Redo 日志后，在物理数据页上，重新修改数据的过程。Redo 日志重演由专门的 Redo 日志重演服务完成，重演服务严格按照 Redo 日志产生的先后顺序，解析 Redo 日志、修改相应的物理数据页，并且重演过程中备库会生成自身的 Redo 日志写入联机日志文件。

守护进程

守护进程（dmwatcher）是数据守护系统的核心工具，监控数据库实例的运行状态和主备库数据同步情况，在出现故障时启动各种处理预案。守护进程是各种消息的中转站，接收数据库实例、其他守护进程、以及监视器发送的各种消息；同时，守护进程也会将收到的数据库实例消息转发给其他守护进程和监视器。守护进程必须和被守护的数据库实例部署在同一台机器上。

监视器

监视器（dmmonitor）用来监控守护系统内守护进程、数据库实例信息，执行用户输入命令、监控实例故障、实现自动切换等。监视器一般配置在数据库实例和守护进程以外的机器上。

2.1 系统特性

2.1.1 主要特性

DM 数据守护的主要特性包括：

完整功能的主库

主库提供完整的数据库服务，与普通单节点数据库相比，主要的功能限制包括：不支持修改表空间文件名、不支持修改 arch_inn 参数。

活动的备库

基于独特的字典缓存技术和日志重演技术，备库在 Open 状态下执行数据同步，是真正意义上的热备库；在实现异地容灾的同时，用户可以只读访问备库，执行报表生成、数据备份等功能，减轻主库的系统负载，提高资源利用率。

多重数据保护

每个备库都是一个完整的数据库备份，可以同时配置多个备库，为数据安全提供全方位的保护。

高可用性

主库出现故障时，可以快速将备库切换为主库，继续提供数据库服务，确保数据库服务不中断。切换过程一般在数秒钟之内完成。

多种守护模式

提供自动切换和手动切换两种守护模式，满足用户不同需求。其中，配置自动切换的前提是已经部署确认监视器。在提供第三方机器部署确认监视器情况下，可以配置为故障自动切换模式，主库出现故障时，系统自动将备库切换为主库对外提供数据库服务。

多种守护类型

守护进程可以配置为全局守护（提供实时主备、MPP 主备、读写分离集群功能）或者本地守护，适应各种应用需求。

故障自动重连

配置、使用连接服务名访问数据库，在发生主备库切换后，接口会自动将连接迁移到新的主库上。

故障库自动重加入

主库故障，发生主备库切换。故障主库重启后，可以自动切换为 Standby 模式，作为备库重新加入数据守护系统。

历史数据自动同步

故障备库恢复后，可以自动同步历史数据，无需用户干预，并在同步完成以后，自动恢复为可切换备库。

自动负载均衡

配置读写分离集群，可以将只读操作分流到备库上执行，减轻主库访问压力，提高数据库系统的吞吐量。读写分离的过程由 JDBC 等接口配合服务器自动完成，无需用户干预，也不需要修改应用程序。

滚动升级

可以在不中断数据库服务的情况下，滚动地对数据守护系统中的主备库进行数据库软件版本升级。

灵活的搭建方式

可以在不中断数据库服务的情况下，将单节点数据库升级为主备系统。DM 提供多种工具来完成数据守护搭建，如 SHELL 脚本或 DEM 工具，均能方便地完成数据守护搭建。

完备的监控工具

通过命令行监控工具 dmmonitor、DEM 工具可以实时更新、监控主备库的状态和数据同步情况。

完善的监控接口

提供完善的数据守护监控接口，可以定制监控项，并方便地集成到应用系统中。

丰富的守护命令

提供主备库切换、强制接管等功能，通过简单的命令就可以实现主备库角色互换、故障接管等功能。

支持 DMDSC 守护

支持 DMDSC 和单节点、单节点和 DMDSC 之间互为主备的数据守护环境。

DM 的单节点和主库提供读未提交 (Read Uncommitted)、读提交 (Read Committed) 和串行化 (Serializable) 三种事务隔离级, 可重复读 (Repeatable Read) 升级为更严格的串行化事务隔离级。但是, 备库只



说明: 支持读提交 (Read Committed) 事务隔离级别:

1. 不能访问所有未提交事务的修改
2. 可以访问所有已提交事务的修改

2.1.2 主要改进 (V2.0 版本)

与 DM 数据守护 V1.0 版本相比, DM 数据守护 V2.0 版本的主要改进包括:

1. 检测并处理组分裂

检测并处理组分裂场景, 是 DM 数据守护 V2.0 的一个重大改进。在 V1.0 版本中, 主备库之间的数据一致性完全由用户保证, 备库强制接管后, 没有办法检查重新恢复的主库是否可以作为备库重新加入主备系统。V2.0 版本引入控制文件 dmwatcher.ctl, 将备库接管时的相关信息记录在守护进程控制文件 dmwatcher.ctl 中, 故障主库恢复后, 根据 dmwatcher.ctl 控制文件中信息来判断是否满足故障重加入条件。

2. 支持 TCP 协议, 取消 UDP 协议

数据库实例与守护进程、守护进程与守护进程、以及守护进程与监视器之间的信息传递, 都是基于 TCP 协议完成的。相比于采用 UDP 协议的 DM 数据守护系统 V1.0 版本, 使用 TCP 协议的 V2.0 版本, 具有以下优势:

- 1) 支持跨网段部署数据守护系统, 降低部署的硬件要求;
- 2) 支持在一台物理机器上搭建数据守护系统, 方便测试环境搭建;
- 3) 更加简化、统一的配置文件和配置参数。

3. 实时主备/MPP 主备/读写分离集群均支持故障自动切换

数据守护系统配置为自动切换模式, 主库故障时, 备库可以自动切换为主库, 这个过程叫作故障自动切换。在 V1.0 版本中, 只有实时主备支持故障自动切换, 在 V2.0 版本中, MPP 主备和读写分离集群也支持故障自动切换。

4. 可配置、可中断的备库恢复流程

缺省情况下, 备库故障重启后, 系统会自动进行检测、并启动恢复流程 (同步历史数据)。某些情况下, 用户可能需要对备库进行一些系统维护, 并不希望备库启动后马上进行数据同

步。V2.0 版本提供了备库维护功能，可以通过监视器命令将备库恢复功能暂时屏蔽。

当备库长时间中断后恢复，主备库之间数据差异很大情况下，同步历史数据可能需要很长一段时间。在这个过程中，如果出现新的故障场景需要处理，或者在监视器上执行其他命令，则允许中断当前的恢复流程，待故障处理（用户命令）执行完成后，再次启动恢复流程。

5. 完善异步备库配置

异步备库一般用于历史数据统计、周期报表等对数据实时性要求不高的业务场合。异步归档时机一般选择在源库相对空闲的时候，以避免影响源库的性能。

V1.0 版本只支持在主库上配置异步备库，而 V2.0 版本则可以在主库或备库上配置各自的异步备库，提供了更大的灵活性，可以适应更广泛的应用场景。

6. 任意场景支持强制接管，避免繁琐的手工 SQL 干预

实时主备/MPP 主备/读写分离三种类型全部实现了强制接管命令。

在 V1.0 版本中，实时守护没有监视器命令，需要手工执行一系列 SQL 语句完成强制接管，比较繁琐；读写分离集群和 MPP 主备提供了强制接管命令，但使用场景受限，比如待接管备库处于 MOUNT 状态的情况下，无法通过强制接管命令将备库切换为主库。并且强制接管后，用户无法判断主备库数据是否一致，存在较大的数据安全隐患。

在 V2.0 版本中，实时主备/MPP 主备/读写分离集群都支持强制接管命令，并且根据守护进程控制文件信息，可以判断强制接管后，是否产生组分裂，主备库数据是否可以恢复到一致状态。

7. 配置参数调整，风格统一、步骤简化。提供多种搭建数据守护的自动化工具

在 V1.0 版本中，实时主备、MPP 主备和读写分离集群，各自有一套配置参数。在 V2.0 版本中，将三类守护系统统一为一套配置参数，并且简化掉一部分配置，大大提高了数据守护系统的搭建效率。

在 V2.0 版本中，除了支持手动搭建数据守护系统，DM 还提供多种自动化工具来完成数据守护搭建过程，如 SHELL 脚本、DEM 工具均能简便灵活地完成数据守护系统的搭建。

2.1.3 主要改进（V2.1 版本）

DM 数据守护 V2.1 版本在 V2.0 的基础上，主要改进包括：

1. 统一 MARCH 和 REALTIME 归档实现逻辑

取消 MARCH 归档类型，保留 REALTIME 实时归档类型，MPP 主备和实时主备统一配置

REALTIME 归档。

2. 实时主备功能扩展

扩展实时主备的备库数量，允许最多配置 8 个实时备库（V2.0 只支持配置一个实时备库）；实时主备增加 HUGE 表数据同步支持。

3. 优化 Redo 日志重演性能

采用全新的预解析、预加载技术，大幅优化备库 Redo 日志重演性能，解决了高并发、高压情况下，备库 Redo 日志堆积问题。

提高了极端情况下集群的整体性能，缩短了极端情况下的故障切换时间。读写分离集群在事务一致性模式下，主库需要等待备库 Redo 日志重演完成再响应用户，优化以后有效降低了主库延迟，提升了读写分离集群系统的吞吐量。

4. 对备库进行实时的异常监控和异常恢复

主库守护进程可以实时监控主备之间的数据同步和备库的日志重演情况，一旦出现网络异常或备库自身软硬件异常（比如备库磁盘读写速度异常降低）等原因造成备库无法及时响应主库的情况时，主库守护进程可通知主库将此备库归档失效，暂停到此备库的数据同步，避免拖慢主库性能。

主库会根据配置的恢复间隔定时向异常备库同步数据，如果检测到归档发送速度和备库的重演速度恢复正常，则将其归档重新恢复有效，恢复正常的数据库同步。

5. 简化、统一守护进程配置参数

取消 dmwatcher.ini 中的 MARCH/REALTIME/TIMELY 类型配置，数据守护类型由服务器的类型（比如：是否 MPP 集群）、以及其配置的归档类型来决定。

守护进程可以配置为 GLOBAL、LOCAL 两种类型，表示是否需要进行全局信息同步、以及是否参与集群管理。

2.1.4 兼容性说明（V2.1 版本）

升级到 V2.1 版本后，数据守护可以继续使用 V2.0 的配置文件，不需要任何修改。守护进程 dmwatcher 可以读写、并正确解析 V2.0 版本的 dmwatcher.ini 配置文件。但 V2.0 版本无法解析 V2.1 版本的配置文件。数据库服务器 dmserver 可以正确解析以前版本的归档配置，自动将 MARCH 归档转换成 REALTIME 归档进行处理。

但是，V2.1 版本与 V2.0 版本的 MPP 主备配置存在一定差异，从 V2.0 升级到 V2.1 版本后，需要修改备库的 dm.ini 配置文件，将 MPP_INI 配置项设置为 1，并从主库拷贝

MPP 控制文件 dmmpc.ctl 保存到 ctl_path 目录。否则，数据守护升级到 V2.1 版本后，MPP 主备系统将无法正常运行。

2.1.5 主要改进（V3.0 版本）

DM 数据守护 V3.0 版本在 V2.1 的基础上，主要改进包括：

1. 支持数据共享集群的守护系统

支持数据共享集群（DMDSC）的守护系统，DMDSC（主）和单节点（备）、单节点（主）和 DMDSC（备）之间都可以组成互为主备的数据守护系统。

2. DMDSC 集群守护增加 REMOTE 远程归档

用于 DMDSC 库的恢复以及主备库的异步恢复，发送日志时能够直接在本地访问到其他节点的归档日志。

3. 完善日志校验方式

主备库日志连续性校验方式，单节点日志的 LSN 值是连续递增的，日志 LSN 值都是唯一的（PWR 日志除外），如果把 DMDSC 集群作为一个整体看待，日志的 LSN 值也是连续递增的，但各个节点日志的 LSN 可能存在重复，每个节点的日志 LSN 是递增的但不一定是连续的。因此，DMDSC 主备库之间的日志校验更加复杂，新版本结合物理日志进行校验。

4. 引入适用于 DMDSC 主库的重演 APPLY_LSN 机制

如果主库是 DMDSC 集群，则需要发送所有节点的日志到备库的重演节点进行重演（如果备库是 DMDSC 集群，则重演节点是指备库的控制节点），在主备库需要同步历史数据时（比如备库故障重启），主库需要知道自己的每个节点在备库上的重演情况，以决定每个节点分别从哪里开始同步数据。备库记录 DMDSC 主库各个节点日志重做情况的 LSN，称为 APPLY_LSN。主库是单节点时，备库的 APPLY_LSN 等同于 CUR_LSN。

5. 守护进程以库为单位管理 DMDSC 集群

守护进程以库为单位进行管理，对 DMDSC 集群是指包含有多个节点的库，对单节点则是只有一个节点的库，并新增若干状态进行 DMDSC 集群的管理。

6. 守护进程可以处理 DMDSC 主库或备库的重加入

控制文件 dmwatcher.ctl 改造，结合备库的重演 LSN 概念，将之前的 SLSN 字段变成重演 APPLY_LSN 数组，同时调整判断主备库是否可加入的逻辑。

7. 监视器支持对 DMDSC 集群的监控和管理

监视器调整及扩展一批命令及接口，以适应 DMDSC 集群的监控及管理。

2.1.6 兼容性说明（V3.0 版本）

从 V2.0 或者 V2.1 升级到 V3.0 版本后，如果守护系统中不扩展新增 DMDSC 库，仍然是只包含单节点的数据守护环境，则可以继续使用原来的配置文件，不需要任何修改。守护进程可以正确读写 dmwatcher.ctl 文件及其配置文件，数据库服务器和监视器也都可以正确读取版本升级前的配置文件，但是 V2.0 或 V2.1 无法正确解析 V3.0 的配置文件。

升级成 V3.0 版本后，为保持格式兼容，守护进程仍然是按照老的版本格式读写 dmwatcher.ctl 文件的，如果在升级后的守护系统中要扩展新增 DMDSC 库，则要按照 V3.0 版本的配置要求重新修改相关的 ini 配置文件，并且要重新生成新版本的 dmwatcher.ctl 文件。

另外如果是从 V2.0 直接升级到 V3.0，在 MPP 主备的配置上存在一定差异，从 V2.0 升级到 V3.0 版本后，需要修改备库的 dm.ini 配置文件，将 MPP_INI 配置项设置为 1，并从主库拷贝 MPP 控制文件 dmmpp.ctl 保存到 ctl_path 目录。否则，数据守护升级到 V3.0 版本后，MPP 主备系统将无法正常运行。

2.1.7 主要改进（V4.0 版本）

DM 数据守护 V4.0 版本在 V3.0 的基础上，主要改进包括：

1. 主备库以 RLOG_PKG 为单位同步数据

数据守护 V4.0 在 RLOG_PKG 改造基础上重新实现，主库产生的 Redo 日志以 RLOG_PKG 包的形式发送给备库重演。RLOG_PKG 具有自描述、自校验特征，数据的组织形式更加灵活、高效，支持 HUGE 表操作产生 Redo 日志，并且支持以 RLOG_PKG 为单位进行日志加密和压缩。

2. 备库完整归档主库产生的日志

备库在收到主库发送过来的 RLOG_PKG 后，直接将 RLOG_PKG 写入本地的归档日志文件，完整保留主库上的原始日志，备库重演 RLOG_PKG 产生的联机日志不再进行归档。

采用这种备库归档策略后，不管主备库如何切换，集群中所有数据库的归档日志内容是完全一致的，简化了历史数据同步处理逻辑。

3. 更简便的日志连续性校验

由于 RLOG_PKG 包具有自校验特性，利用 RLOG_PKG 的 G_PKG_SEQNO 和 MIN_LSN、MAX_LSN、PREV_LSN 等信息，结合备库重演信息（APPLY STAT）记录的 P_DB_MAGIC、N_APPLY_EP、PKG_SEQNO_ARR 数组、APPLY_LSN_ARR 数组，可以准确地校验出日志是否连续。简化了主备库（特别是 DSC 主备库）日志连续性校验逻辑，主库只负责发送日志，备库根据接收到的日志和自己维护的重演信息校验日志是否连续。

4. 更可靠的主库变迁记录

增加 SYSOPENHISTORY 系统表，记录数据库的 Open 历史记录，在 Primary 或 Normal 模式库 Open 时，向系统表写入记录（称之为 Open 记录）。主库切换、Open 等过程完整地记录在数据库中，并通过 Redo 日志传送到备库，完成主备库之间 Open 记录的同步。

简化 dmwatcher.ctl 功能，取消 Open 记录，仅保留 status 和 desc 两个字段，在本地数据库分裂时，设置分裂状态和对应的分裂描述信息。并且，dmwatcher.ctl 文件不再需要在主备库之间进行同步。

5. 更完备的库分裂检测机制

优化了主备库分裂场景判断逻辑，守护进程根据主备库 Open 记录的包含关系、主库各节点 LSN 信息和备库日志重演信息，判断是否出现主备库分裂。在未出现分裂情况下，综合考虑各数据库模式、状态要素，选择正确的主库并进行自动 Open 数据库。

6. 更完善的异步备库功能

异步备库支持多源配置，允许 Realtime、Timely 主备库配置相同的异步备库，系统自动识别，仅从主库向异步备库同步历史数据。不管主备库如何切换，始终保持异步备库与主库的数据同步。

2.1.8 兼容性说明（V4.0 版本）

系统自动识别 dmwatcher.ctl 控制文件的版本号，忽略老版本控制文件。在数据库分裂时，自动覆盖并生成新格式的 dmwatcher.ctl 文件。

1. 从 v2.1 或者 v3.0 升级到 v4.0

如果守护系统不再扩展配置其他库（单节点或者 DSC 集群），则可以继续使用原来的配置文件，不需要做任何修改。同时，建议删除主备库各自目录下的 dmwatcher.ctl 文件。

2. 从 v2.0 升级到 v4.0

如果守护系统不再扩展配置其他库（单节点或者 DSC 集群），对实时主备和读写分离集

群，则可以继续使用原来的配置文件，不需要做任何修改。同时，建议删除主备库各自目录下的 dmwatcher.ctl 文件。

MPP 主备还需要修改备库的 dm.ini 配置文件，将 MPP_INI 配置项设置为 1，并从主库拷贝 MPP 控制文件 dmmpp.ctl 保存到 ctl_path 目录。否则，数据守护升级到 V4.0 版本后，MPP 主备系统将无法正常运行。

3. 数据库升级说明

由于 V4.0 的数据库 REDO 日志采用 RLOG_PKG 为单位在日志文件中进行保存，和 RLOG_BUF 机制的格式不兼容，V4.0 版本无法解析老库日志。为了避免库启动时进行重做 REDO 日志、归档文件修复等动作而导致无法升级的情况发生，因此要求升级前的库必须是使用之前版本的服务器执行码正常退出的库，否则无法自动升级。

数据库自动升级的动作包括：根据当前库信息，初始化联机日志文件头、创建一个初始的日志包格式化到文件中、创建 SYSOPENHISTORY 系统表等。

由于升级后老的归档文件无法再继续使用，因此需要确保备库在升级前和主库的数据处于一致状态，否则升级完成后，主库无法再从老的归档文件中收集数据同步到备库上。



注意：

以上描述仅为升级的简单说明，具体的升级操作请咨询达梦技术服务人员。

4. 数据库降级说明

对数据守护 V4.0 环境，支持将主库重新降级到 V3.0 版本（仅支持降级到 V8.1.0.182 版本），备库不支持直接降级。

由于 REDO 日志格式降级前后是不兼容的，为了避免降级后的数据库启动时进行重做 REDO 日志、归档文件修复等动作导致无法正常启动的情况发生，因此要求降级前的库必须是使用之前版本的服务器执行码正常退出的库，否则不允许执行降级。

数据库降级的动作包括：按照 V3.0 的格式降级联机日志文件头、降级数据字典版本号、删除 SYSOPENHISTORY 系统表、按照老的视图定义重建一批动态视图等。



注意：

以上描述仅为降级的简单说明，具体的降级操作请咨询达梦技术服务人员。

2.2 基本概念

在介绍数据守护主要功能之前，我们先熟悉几个基本概念，只有充分理解这些概念，才能更加深入地理解数据守护。

2.2.1 数据库

数据守护以库为单位进行管理，一个 DMDSC 集群的所有实例作为一个整体库来考虑。DMDSC 集群的库信息，例如库的状态、模式等需要综合考虑集群内所有实例，同时需要结合 DMDSC 本身的状态。

2.2.2 DMDSC 状态

DMDSC 状态标识 DMDSC 集群节点状态，和数据库的状态不同。包括以下几种：

- **Startup** 节点启动状态，需要通过 DMCSS 工具交互，确定控制节点，执行日志重做等相关步骤，进入到 OPEN 状态。
- **Open** 实例正常工作状态，当集群内发生节点故障或启动节点重加入步骤时，可以进入 crash_recv 或者 err_ep_add 状态，处理完成后会再回到 Open 状态。
- **Crash_recv** 节点故障处理状态。
- **Err_ep_add** 故障节点重加入状态。

2.2.3 数据库模式

DM 支持 3 种数据库模式：Normal 模式、Primary 模式和 Standby 模式。

Normal 模式

提供正常的数据库服务，操作没有限制。正常生成本地归档，但不发送实时归档（Realtime）、即时归档（Timely）和异步归档（Async）。

Primary 模式

提供正常的数据库服务，操作有极少限制。该模式下部分功能受限，包括：不支持修改表空间文件名、不支持修改 arch_ini 参数。正常生成本地归档，支持实时归档（Realtime）、即时归档（Timely）和异步归档（Async）。Primary 模式下，对临时表

空间以外的所有的数据库对象的修改操作都强制生成 Redo 日志。

Standby 模式

可以执行数据库备份、查询等只读数据库操作。正常生成本地归档，正常发送异步归档 Redo 日志；但实时归档（Realtime）、即时归档（Timely）均强制失效。该模式下时间触发器、事件触发器等都失效。

可以通过 SQL 语句切换数据库模式，模式切换必须在 Mount 状态下执行。切换模式 SQL 语句如下：

将数据库切换为 Primary 模式：

```
ALTER DATABASE PRIMARY;
```

将数据库切换为 Standby 模式：

```
ALTER DATABASE STANDBY;
```

将数据库切换为 Normal 模式：

```
ALTER DATABASE NORMAL;
```



修改 DMDSC 库的模式必须在 DMDSC 库所有实例都处于 MOUNT 状态下才能进

注意：行，只需要在一个节点上执行以上语句即可。

2.2.4 数据库状态

DM 的数据库状态包括：

Startup 状态

系统刚启动时设置为 Startup 状态。

After Redo 状态

系统启动过程中联机日志重做完成后，回滚活动事务前设置为 After Redo 状态。非 Standby 模式的实例在执行 alter database open 操作前，也会将系统设置为 After Redo 状态。

Open 状态

数据库处于正常提供服务的状态，但不能进行归档配置等操作。

Mount 状态

数据库在 Mount 状态下，不能修改数据，不能访问表、视图等数据库对象，但可以执

行修改归档配置、控制文件和修改数据库模式等操作，也可以执行一些不修改数据库内容的操作，比如查询动态视图或者一些只读的系统过程。由于 Mount 状态不生成 PWR 日志，因此数据页可以正常刷盘，也正常推进检查点。

系统从 Open 状态切换为 Mount 状态时，会强制回滚所有活动事务，但不会强制清理（Purge）已提交事务，不会强制断开用户连接，也不会强制 Buffer 中的脏页刷盘。

Suspend 状态

数据库在 Suspend 状态下，可以访问数据库对象，甚至可以修改数据，但限制 Redo 日志刷盘，一旦执行 COMMIT 等触发 Redo 日志刷盘的操作时，当前操作将被挂起。

相比 Open 到 Mount 的状态切换，Open 到 Suspend 的状态切换更加简单、高效，不会回滚任何活动事务，在状态切换完成后，所有事务可以继续执行。

一般在修改归档状态之前将系统切换为 Suspend 状态，比如备库故障恢复后，在历史数据（归档日志）同步完成后，需要重新启用实时归档功能时：

1. 将系统切换为 Suspend 状态，限制 Redo 日志写入联机 Redo 日志文件；
2. 修改归档状态为 Valid；
3. 重新将数据库切换为 Open 状态，恢复 Redo 日志写入功能；
4. 备库与主库重新进入实时同步状态。

另外，实时归档失败时（比如网络故障导致），Primary 实例将试图切换成 Suspend 状态，防止后续的日志写入。因为一旦写入，主备切换时有可能备库没有收到最后那次的 RLOG_PKG，导致主库上多一段日志，很容易造成主备数据不一致。当实例成功切换为 SUSPEND 状态时，可直接退出，强制丢弃多余的日志，避免主备数据不一致。



修改 DMDSC 库的状态为 SUSPEND 时，库内所有实例都不能处于 MOUNT 状态，

注意：只需要在一个节点上执行 ALTER DATABASE SUSPEND 语句即可。

Shutdown 状态

实例正常退出时设置为 Shutdown 状态。

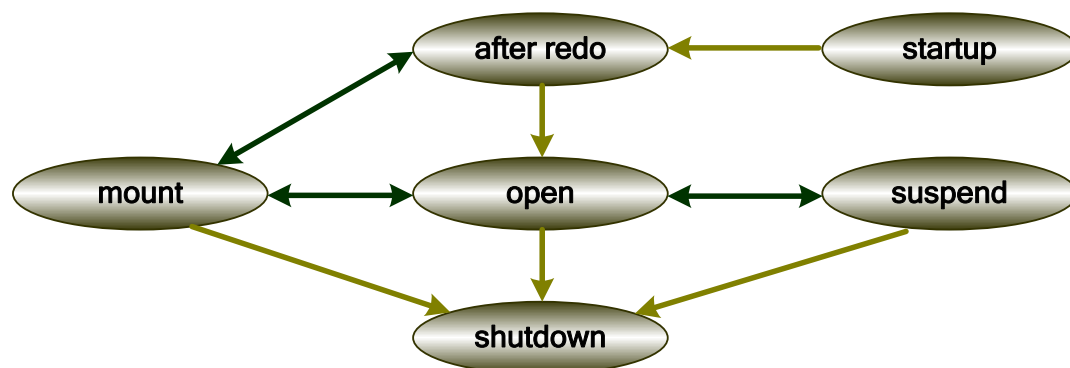


图 2.2 数据库状态转换

不同数据库状态之间的转换规则如图 2.2 所示。用户可以通过 SQL 语句进行数据库状态切换：1. Open 状态与 Mount 状态可以相互切换；2. Open 状态与 Suspend 状态可以相互切换；3. Mount 和 Suspend 状态不能直接转换；4. 其他状态为系统内部状态，用户不能主动干预。



对 DMDSC 集群，除了修改 Suspend 是同步操作，只需要在一个节点执行外，

注意：其他状态修改都需要在每个节点上各自单独执行。

切换数据库状态的 SQL 如下：

1. 将数据库修改为 Open 状态。当系统处于 Primary/Standby 模式时，必须强制加上 FORCE 子句。

```
ALTER DATABASE OPEN [FORCE];
```

2. 将数据库修改为 Mount 状态。

```
ALTER DATABASE MOUNT;
```

3. 将数据库修改为 Suspend 状态。

```
ALTER DATABASE SUSPEND;
```

由于 dmwatcher 根据数据库模式、状态等信息作为故障处理、故障恢复的



依据，建议在配置数据守护过程中，修改 dm.ini 参数

小窍门：ALTER_MODE_STATUS 为 0，限制用户直接通过 SQL 语句修改数据库状态、

模式以及 OGUID，避免 dmwatcher 做出错误的决策。

2.2.5 LSN 介绍

LSN (Log Sequence Number) 是由系统自动维护的 Bigint 类型数值，具有自动

递增、全局唯一特性，每一个 LSN 值代表着 DM 系统内部产生的一个物理事务。物理事务（Physical Transaction，简称 ptx）是数据库内部一系列修改物理数据页操作的集合，与数据库管理系统中事务（Transaction）概念相对应，具有原子性、有序性、无法撤销等特性。

DM 数据库中 与 LSN 相关的信息，可以通过查询 V\$RLOG 和 V\$RAPPLY_PARALLEL_INFO 表来获取。DM 主要包括以下几种类型的 LSN：

- **CUR_LSN** 是系统已经分配的最大 LSN 值。物理事务提交时，系统会为其分配一个唯一的 LSN 值，大小等于 CUR_LSN + 1，然后再修改 CUR_LSN=CUR_LSN+1。

- **FILE_LSN** 是已经写入联机 Redo 日志文件的日志包的最大 LSN 值。每次将 Redo 日志包 RLOG_PKG 写入联机 Redo 日志文件后，都要修改 FILE_LSN 值。

- **FLUSH_LSN** 是已经发起日志刷盘请求，但还没有真正写入联机 Redo 日志文件的最大 LSN 值。

- **CKPT_LSN** 是检查点 LSN，所有 LSN ≤ CKPT_LSN 的物理事务修改的数据页，都已经从 Buffer 缓冲区写入磁盘，CKPT_LSN 由检查点线程负责调整。

- **APPLY_LSN** 是备库已写入联机 Redo 日志文件的日志包的原始最大 LSN 值，此 LSN 取自主库对应的原始日志包中的最大 LSN 值。如果主库是 DMDSC 集群，备库分别为主库每一个节点维护一个 APPLY_LSN。

- **RPKG_LSN** 是备库重演 LSN，表示备库已经重演完成的最大 LSN。如果主库是 DMDSC 集群，备库分别为主库每一个节点维护一个 APPLY_LSN。

与上述 LSN 对应，DM 数据守护也定义了一批 LSN：

- ✓ **CLSN** 与 CUR_LSN 保持一致，数据库已经分配的最大 LSN 值。
- ✓ **FLSN** 与 FILE_LSN 保持一致，已写入联机日志文件的 LSN 值。
- ✓ **ALSN** 与 APPLY_LSN 保持一致，备库已写入联机日志文件的原始 LSN 值。
- ✓ **RLSN** 与 RPKG_LSN 保持一致，备库已经重演完成的最大 LSN 值。
- ✓ **SLSN** 是 Standby LSN 的缩写，表示备库明确可重演的最大 LSN 值。
- ✓ **KLSN** 是 Keep LSN 的缩写，表示备库已经收到、但未明确是否可以重演的 RLOG_PKG 的最大 LSN 值。在读写分离集群中 KLSN == SLSN。

2.2.6 Redo 日志

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。一般说来一条修改数据的 SQL 语句（比如 Insert），在系统内部会转化为多个相互独立的物理事务来完成，物理事务提交时会将产生的 Redo 日志写入日志包 RLOG_PKG 中。

一个物理事务包含一个或者多个 Redo 记录（Redo Record），每条 Redo 记录（RREC）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。

逻辑 RREC 记录是专门为数据守护增加的记录类型，用来解决备库重演 Redo 日志与用户访问备库之间的并发冲突，以及主库执行 DDL 后导致的主备数据库字典缓存不一致问题。备库解析到逻辑 RREC 记录时，根据记录内容，生成相应的事务，封锁对应的数据库对象，并从字典缓存中淘汰过期的字典对象。

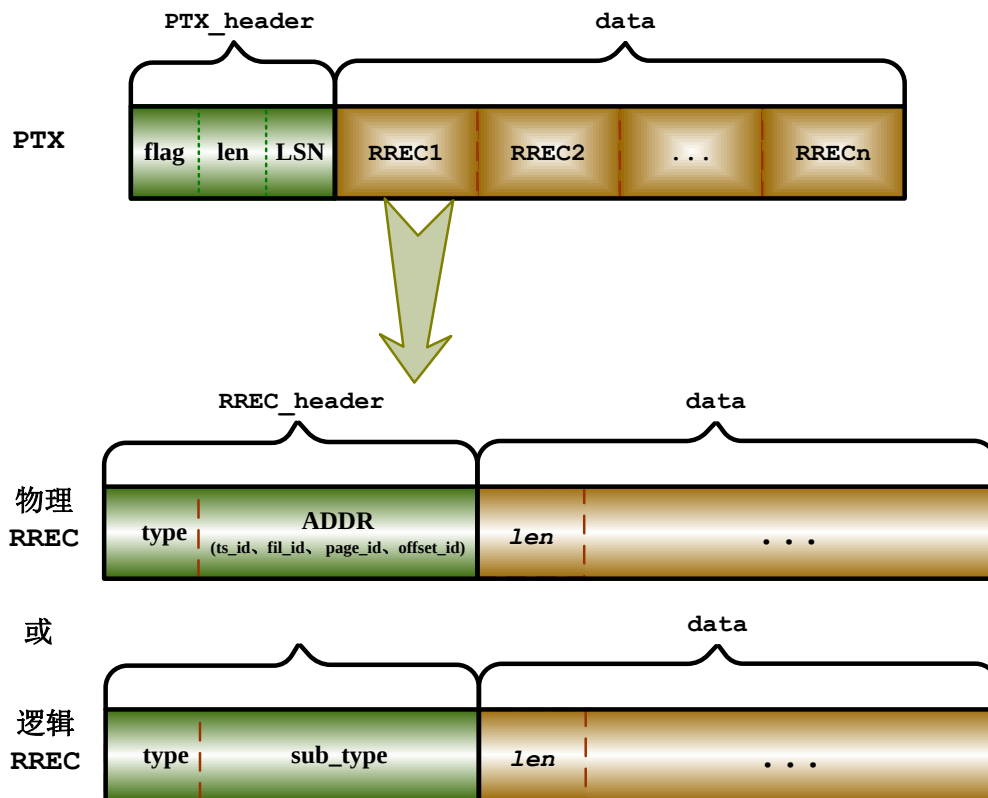


图 2.3 PTX/RREC 结构图

2.2.7 Redo 日志包 (RLOG_PKG)

Redo 日志包 (RLOG_PKG) 是 DM 数据库批量保存物理事务产生的 Redo 日志的数据单元，以物理事务 PTX 为单位保存日志，一个日志包内可连续保存一个或多个 PTX。日志包具有自描述的特性，日志包大小不固定，采用固定包头和可变包头结合的方式，包头记录日志的控制信息，包括类型、长度、包序号、LSN 信息、产生日志的节点号、加密压缩信息、日志并行数等内容。

日志包生成时按照序号连续递增，相邻日志包的 LSN 顺序是总体递增的，但是在多节点集群的 DSC 环境下不一定连续。如果未开启并行日志，RLOG_PKG 包内日志的 LSN 是递增的。如果开启并行日志，一个 RLOG_PKG 包内包含多路并行产生的日志，每一路并行日志的 LSN 是递增的，但是各路之间并不能保证 LSN 有序，因此并行日志包内 LSN 具有局部有序，整体无序的特点。

物理事务提交时将 Redo 日志写入到日志包中，在数据库事务提交或日志包被写满时触发日志刷盘动作。日志刷盘线程负责将日志包中的 Redo 日志写入联机日志文件，如果配置

了 Redo 日志归档，日志刷盘线程还将负责触发归档动作。DM 数据守护系统中，主库以 RLOG_PKG 为最小单位发送 Redo 日志到备库。

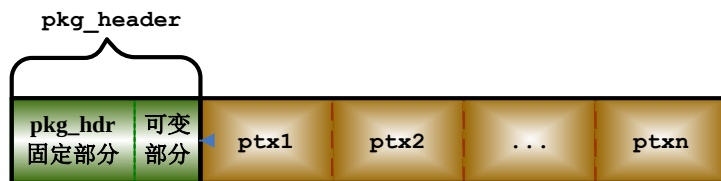


图 2.4 RLOG_PKG 结构图

2.2.8 包序号介绍

每个 RLOG_PKG 都有对应的序号属性，称之为包序号（PKG_SEQNO），日志包生成时按照序号连续递增。包序号包括本地包序号（LSEQ）和全局包序号（GSEQ），本地包序号是节点内唯一、连续递增的值，用于校验联机日志连续性；全局包序号由数据守护集群的主备库共同维护，具有全局唯一、连续、递增的特性，用于校验归档日志的连续性。

DM 数据库中与全局包序号相关的信息可以通过查询 V\$RLOG 表来获取，主要包括以下几种类型的全局包序号：

- **CUR_SEQ** 是系统已经分配的最大全局包序号。RLOG_PKG 写入联机日志文件前，系统会为其分配一个唯一的全局包序号。

- **FILE_SEQ** 是已经写入联机 Redo 日志文件的日志包的最大全局包序号。每次将 Redo 日志包 RLOG_PKG 写入联机 Redo 日志文件后，都要修改 FILE_SEQ 值。

- **APPLY_SEQ** 是备库已写入联机 Redo 日志文件的日志包的原始最大全局包序号，此序号取自主库对应的原始日志包的包序号。如果主库是 DSC 集群，备库分别为主库每一个节点维护一个 APPLY_SEQ。

- **RPKG_SEQ** 是备库重演全局包序号，表示备库已经重演完成的最大全局包序号。如果主库是 DSC 集群，备库分别为主库每一个节点维护一个 APPLY_SEQ。

DM 数据守护中也相应地定义了一批全局包序号：

- ✓ **CSEQ** 全局已分配包序号，标识系统已经分配的最大 GSEQ 值。
- ✓ **FSEQ** 全局文件包序号，标识已写入联机日志文件的最大 GSEQ 值。
- ✓ **ASEQ** 原始全局文件包序号，标识备库已写入联机日志文件的原始 GSEQ 值。
- ✓ **RSEQ** 全局重演包序号，标识备库已经重演的最大 GSEQ 值。
- ✓ **SSEQ** 全局备库包序号，标识备库明确可重演的最大 GSEQ 值。

- ✓ **KSEQ** 全局保留包序号，表示备库已经收到、未明确是否可以重演的最大 GSEQ 值。在读写分离集群中 $SSEQ == KSEQ$ 。

GSEQ 和 LSN 存在匹配关系，在判断归档连续性、备库归档恢复等场合一般都是同时使用。

2.2.9 KEEP_PKG 介绍

主库的 RLOG_PKG 日志通过实时归档机制发送到备库后，备库将最新收到的 RLOG_PKG 保存在内存中，不马上启动重演，这个 RLOG_PKG 我们称之为 KEEP_PKG。引入 KEEP_PKG 的主要目的是，避免下述场景中，主库故障重启后不必要的主备切换，减少用户干预。

场景说明（实时主备或 MPP 主备）：

1. 用户登录主库 A 执行

```
CREATE TABLE TX(C1 INT);  
  
INSERT INTO TX VALUES(1);  
  
COMMIT;
```

其中 COMMIT 操作将触发实时归档，发送 RLOG_PKG 到备库 B。

2. 备库 B 收到 RLOG_PKG，响应主库 A，并启动日志重演。

3. 主库 A 在 RLOG_PKG 写入联机日志文件之前故障。

4. 主库 A 重新启动后，由于 RLOG_PKG 没有写入联机日志文件，之前插入 TX 表的数据丢失；但此时备库 B 已经重演日志成功，TX 表中已经插入一行数据。

上述场景中，主备库数据不再保持一致，必须将备库 B 切换为主库，并重新从 B 同步数据到 A。如果配置的是手动切换模式，则必须要有用户干预，进行备库接管后，才能恢复数据库服务。

引入 KEEP_PKG 后，备库 B 收到主库 A 发送的 RLOG_PKG，并不会马上启动日志重演，主库 A 重启后，守护进程 A 检测到备库 B 存在 KEEP_PKG，通知备库 B 丢弃 KEEP_PKG 后，直接 Open 主库 A，就可以继续提供数据库服务。并且，这些操作是由守护进程自动完成，不需要用户干预。

如果备库自动接管、或者用户发起备库接管命令，那么备库的 KEEP_PKG 将会启动重

演，不管主库是否已经将 KEEP_PKG 对应的 Redo 日志写入联机日志文件中，备库接管时的 APPLY_LSN 一定是大于等于主库的 FILE_LSN。当故障主库重启后，仍然可以作为备库，自动重新加入数据守护系统。

备库 KEEP_PKG 日志重演的时机包括：

1. 备库收到新的 RLOG_PKG

备库收到新的 RLOG_PKG 时，会将当前保存的 KEEP_PKG 日志重演，并将新收到的 RLOG_PKG 再次放入 KEEP_PKG 中。

2. 收到主库的重演命令

主库会定时将 FILE_LSN 等信息发送到备库，当主库 FILE_LSN 等于备库 SLSN 时，表明主库已经将 KEEP_PKG 对应的 Redo 日志写入联机日志文件中，此时备库会启动 KEEP_PKG 的日志重演。

3. 备库切换为新主库

在监视器执行 SWITCHOVER 或 TAKEOVER 命令，或者确认监视器通知备库自动接管时，备库会在切换为 PRIMARY 模式之前，启动 KEEP_PKG 的日志重演。



即时归档在 RLOG_PKG 写入主库联机 Redo 日志文件后，再发送 RLOG_PKG

说明：到备库，因此即时备库没有 KEEP_PKG。

2.2.10 联机 Redo 日志文件

DM 数据库默认包含两个联机 Redo 日志文件（如 DAMENG01.log、DAMENG02.log），系统内部分别称为 0 号文件、1 号文件。RLOG_PKG 顺序写入联机 Redo 日志文件中，当一个日志文件写满后，自动切换到另一个文件。其中 0 号文件是 Redo 日志主文件，在日志主文件头中保存了诸如 CKPT_LSN，CKPT_FILE，CKPT_OFFSET，FILE_LSN 等信息。系统故障重启时，从 [CKPT_FILE，CKPT_OFFSET] 位置开始读取 Redo 日志，解析 RREC 记录，并重新修改对应数据页内容，确保将数据恢复到系统故障前状态。

随着检查点(Checkpoint)的推进，对应产生 Redo 日志的数据页从数据缓冲区(Data Buffer) 写入磁盘后，联机 Redo 日志文件可以覆盖重用、循环使用，确保 Redo 日志文件不会随着日志量的增加而增长。



任何数据页从 **Buffer** 缓冲区写入磁盘之前，必须确保修改数据页产生的

注意：Redo 日志已经写入到联机 Redo 日志文件中。

下图说明了联机日志文件、日志包 RLOG_PKG 以及相关各 LSN 之间的关系。

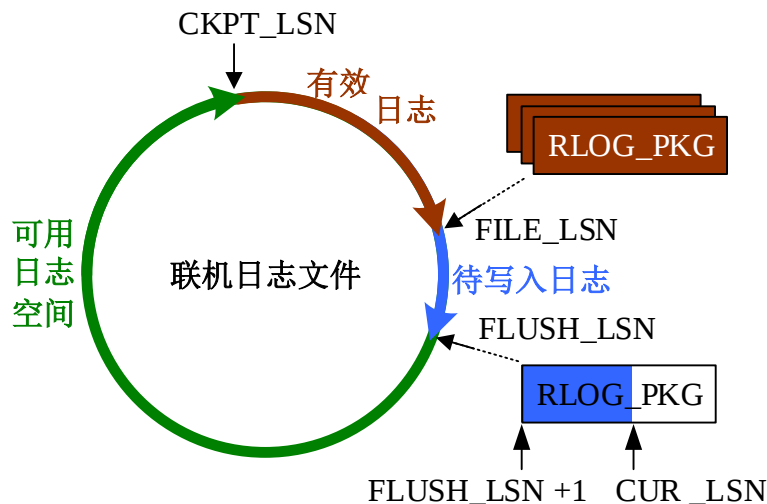


图 2.5 联机 Redo 日志文件与日志包

在联机日志文件中，可以覆盖写入 Redo 日志的文件长度为可用日志空间；不能被覆盖，系统故障重启需要重做部分，为有效 Redo 日志，有效 Redo 日志的 LSN 取值范围是 $(CKPT_LSN, FILE_LSN]$ ；已经被发起日志刷盘请求，但还没有真正写入联机 Redo 日志文件的区间为 $(FILE_LSN, FLUSH_LSN]$ ，称为待写入日志空间。

2.2.11 归档介绍

归档是实现数据守护系统的重要技术手段，根据功能与实现方式的不同，DM 数据库的归档可以分为 5 类：本地归档、远程归档、实时归档、即时归档和异步归档。其中，本地归档和远程归档日志的内容与写入时机与数据库模式相关；主库 Redo 日志写入联机日志文件后，再进行本地归档和远程归档；备库收到主库产生的 Redo 日志后，直接进行本地归档和远程归档，同时启动 Redo 日志重演。

2.2.11.1 本地归档

Redo 日志本地归档（Local），就是将 Redo 日志写入到本地归档日志文件的过程。配置本地归档情况下，Normal/Primary 模式库在 Redo 日志写入联机 Redo 日志文件后，

将对应的 RLOG_PKG 由专门的归档线程写入本地归档日志文件中。Standby 模式库收到主库产生的 Redo 日志后，直接进行本地归档，写入本地归档日志文件中，同时启动 Redo 日志重演。

Normal/Primary 模式库归档日志文件保存的是当前节点产生的 Redo 日志，归档日志文件内容与联机日志内容保持一致。Standby 模式库重演日志重新产生的 Redo 日志只写入联机日志文件，归档日志文件保存主库产生的 Redo 日志，因此备库联机日志文件内容和归档日志文件内容是不完全一致的。采用这种归档实现方式后，可以确保数据守护系统中所有节点的归档日志文件内容是完全一致的。

与联机 Redo 日志文件可以被覆盖重用不同，本地归档日志文件不能被覆盖，写入其中的 Redo 日志信息会一直保留，直到用户主动删除；如果配置了归档日志空间上限，系统会自动删除最早生成的归档 Redo 日志文件，腾出空间。本地归档文件在配置的归档目录下生成并保存，如果磁盘空间不足，且没有配置归档日志空间上限（或者配置的上限超过实际空间），系统将自动挂起，直到用户主动释放出足够的空间后继续运行。

DM 提供了按指定的时间或指定的 LSN 删除归档日志的系统函数，分别为 SF_ARCHIVELOG_DELETE_BEFORE_TIME 和 SF_ARCHIVELOG_DELETE_BEFORE_LSN，但用户需要谨慎使用。例如，在守护系统中，如果备库故障了，主库继续服务，主库的日志在继续增长。此时如果删除尚未同步到备库的主库归档日志，那么备库重启之后，会由于备库收到的日志缺失导致主备库无法正常同步数据。



注意：

为了最大限度地保护数据，当磁盘空间不足导致归档写入失败时，系统会挂起等待，直到用户释放出足够的磁盘空间。当磁盘损坏导致归档日志写入失败时，系统会强制 HALT。

2.2.11.2 远程归档

远程归档专门用于 DMDSC 环境中。

所谓远程归档（REMOTE ARCHIVE），顾名思义就是将归档目录配置在远程节点上。远程归档采用双向配置的方式，双向配置远程归档就是两个节点将自己的远程归档相互配置在对方机器上。集群中所有的节点，都拥有一套包括所有节点的，完整的归档日志文件。

具体有两种配置方式：一是共享本地归档的远程归档，即将远程归档目录配置为另一节

点的本地归档目录，以此来共享它的本地归档日志文件；二是通过 MAL 发送的远程归档，即将写入本地归档的 REDO 日志信息，通过 MAL 发送到远程节点，并写入远程节点的指定归档目录中，形成远程归档日志文件。

配置远程归档后，DMDSC 集群中每个实例完整保存了集群所有实例产生的 Redo 日志，任意实例都可以像访问本地归档一样，访问所有实例产生的归档日志。

远程归档的使用场景：

1. 执行数据库恢复时，恢复工具（如 DMRMAN）所在节点需要访问其他各节点归档日志。
2. DMDSC 守护系统中进行主、备库异步归档日志的同步或备库恢复时，DMDSC 控制节点作为发送端，需要访问其他从节点的归档日志。

2.2.11.3 归档文件

备库归档机制调整后，备库归档日志文件写入的并不是自己重演生成的 Redo 日志，而是直接将主库产生的 Redo 日志写入到本地归档日志文件中。为了区分生成 Redo 日志和写入 Redo 日志的库，归档日志文件头增加了几个 MAGIC 字段：

■ **PMNT_MAGIC** 永久魔数，用来唯一标识数据库，初始化数据库时生成并保持不变（DDL_CLONE 还原库除外），数据守护集群中所有主备库的 PMNT_MAGIC 是相同的。只有 DDL_CLONE 还原库的 PMNT_MAGIC 会发生改变，当一个库使用 DDL_CLONE 备份集还原并恢复之后，在执行 RECOVER DATABASE UPDATE DB_MAGIC 时，PMNT_MAGIC 会发生改变。

■ **DB_MAGIC** 数据库魔数，数据库初始化时生成，数据库还原后重新生成新的 DB_MAGIC，数据守护集群中所有主备库的 DB_MAGIC 是不同的。归档日志文件使用 DB_MAGIC 标识写入 Redo 日志的库。

■ **SRC_DB_MAGIC** 源库魔数，产生 Redo 日志数据库的 DB_MAGIC 值；主库归档日志文件中 SRC_DB_MAGIC 与 DB_MAGIC 相同；备库归档日志文件中 SRC_DB_MAGIC 与主库的 DB_MAGIC 值相同。

Primary/Normal 模式库本地归档日志文件的命名方式调整为：

```
"ARCH_NAME_DB_MAGIC[SEQNO]_日期时间.log"
```

其中 ARCH_NAME 是在 dmarch.ini 中配置的 LOCAL/REMOTE 归档名称，DB_MAGIC 是生成日志的数据库魔数，SEQNO 代表 DSC 节点号，日期时间是归档日志文件的创建时间。

例如：

```
"ARCHIVE_LOCAL1_0x567891[0]_2019-03-20_10-35-34.log"
```

Standby 模式库（备库）生成的归档日志文件的命名方式调整为：

```
"STANDBY_ARCHIVE_DB_MAGIC[SEQNO]_日期时间.log"
```

其中 STANDBY_ARCHIVE 表示备库生成的归档日志文件，DB_MAGIC 是生成日志的数据库（主库）魔数，SEQNO 代表主库对应的 DSC 节点号，日期时间是归档日志文件的创建时间。

例如：

```
"STANDBY_ARCHIVE_0x123456[0]_2019-03-20_10-35-34.log"
```



由于 STANDBY_ARCHIVE 用于表示备库生成的归档日志，不允许将归档名称

注意：配置为 STANDBY_ARCHIVE。

2.2.11.4 实时归档

与本地归档写入保存在磁盘中的日志文件不同，实时归档（Realtime）将主库产生的 Redo 日志通过 MAL 系统传递到备库，实时归档是实时主备和 MPP 主备的实现基础。实时归档只在主库生效，一个主库可以配置 1~8 个实时备库。

实时归档的执行流程是，主库在 Redo 日志(RLOG_PKG)写入联机日志文件前，将 Redo 日志发送到备库，备库收到 Redo 日志(RLOG_PKG)后标记为 KEEP_PKG，将原 KEEP_PKG 加入日志重演任务系统，并马上响应主库，不需要等待 Redo 日志重演结束后再响应主库。主库收到备库的响应消息，确认备库已经收到 Redo 日志后，再将 Redo 日志写入联机日志文件中。

另外，实时归档也可以支持读写分离集群，实时归档也分为两种模式：事务一致模式和高性能模式，这两种模式的具体含义和下一小节“2.2.11.5”中的说明完全相同，区别仅在于配置为实时归档时，dmarch.ini 中的 ARCH_WAIT_APPLY 配置项默认值为 0，即采用高性能模式，具体内容请参考“2.2.11.5”和“2.6.5”小节。

2.2.11.5 即时归档

即时归档 (Timely) 在主库将 Redo 日志写入联机日志文件后, 通过 MAL 系统将 Redo 日志发送到备库。即时归档与实时归档的主要区别是 Redo 日志的发送时机不同。一个主库可以配置 1~8 个即时备库。

根据备库重演 Redo 日志和响应主库时机的不同, 即时归档分为两种模式: 事务一致模式和高性能模式。即时归档模式根据配置文件 dmarch.ini 中的 ARCH_WAIT_APPLY 配置项 (默认值为 1) 来确定, 1 表示事务一致模式, 0 表示高性能模式。

■ **事务一致模式** 主库事务提交触发 Redo 日志刷盘和即时归档, 备库收到主库发送的 Redo 日志, 并重演完成后再响应主库。主库收到备库响应消息后, 再响应用户的提交请求。事务一致模式下, 同一个事务的 SELECT 语句无论是在主库执行, 还是在备库执行, 查询结果都满足 READ COMMIT 隔离级要求。

■ **高性能模式** 与实时归档一样, 备库收到主库发送的 Redo 日志后, 马上响应主库, 再启动日志重演。高性能模式下, 备库与主库的数据同步存在一定延时 (一般情况下延迟时间非常短暂, 用户几乎感觉不到), 不能严格保证事务一致性。

事务一致模式下, 主备库之间严格维护事务一致性, 但主库要等备库 Redo 日志重演完成后, 再响应用户的提交请求, 事务提交时间会变长, 存在一定的性能损失。高性能模式则通过牺牲事务一致性获得更高的性能和提升系统的吞吐量。用户应该根据实际情况, 选择合适的即时归档模式。

2.2.11.6 异步归档

异步归档 (Async) 由主、备库上配置的定时器触发, 根据异步备库的 KEEP_LSN 信息, 扫描本地归档目录获取 Redo 日志, 并通过 MAL 系统将 Redo 日志发送到异步备库。异步备库的 Redo 日志重演过程与实时归档等其他类型的归档完全一致。

每个 Primary 或 Standby 模式的数据库最多可以配置 8 个异步备库, Normal 模式下配置的异步备库会自动失效。异步备库可以级联配置, 异步备库本身也可以作为源库配置异步备库。

2.2.11.7 归档区别

表 2.1 归档类型比较

类型 比较	本地归档	实时归档	即时归档	异步归档
备库数量	0	1~8	1~8	1~8
通过 MAL 传递数据	否	是	是	是
归档时机	写入联机日志后，再写入本地归档日志文件	写入联机日志前，发送到备库	写入联机日志后，发送到备库	定时启动
归档写入（发送）	归档线程	日志刷盘线程	日志刷盘线程	异步归档线程
数据来源	RLOG_PKG	RLOG_PKG	RLOG_PKG	本地归档文件
失败处理	磁盘空间不足时，系统挂起等待用户释放出足够的磁盘空间。 磁盘损坏导致写入失败时，系统会强制 HALT	Suspend 数据库，保持归档状态不变，等待守护进程干预	Suspend 数据库，并设置归档为无效状态，等待守护进程干预	不做处理，等待下次触发继续发送
备库响应时机	无	事务一致模式：重演完成后响应； 高性能模式：收到立即响应	事务一致模式：重演完成后响应； 高性能模式：收到立即响应	收到立即响应
源库模式	Primary Standby Normal	Primary	Primary	Primary Standby

目标库模式	无	Standby	Standby	Standby
-------	---	---------	---------	---------



任意一个备库的实时归档/即时归档失败（即使其他备库归档成功了），主库说明：都会切换为 **Suspend** 状态。

2.2.11.8 归档状态

本地归档、实时归档和即时归档均包含两种状态：Valid 和 Invalid。异步归档只有一种归档状态：Valid。

- **Valid** 归档有效，正常执行各种数据库归档操作。
- **Invalid** 归档无效，主数据库不发送联机 Redo 日志到备数据库。

在不同的归档类型中，归档状态转换时机不同。具体转换时机描述如下：

1. 主备库启动后，主库到所有备库的归档默认为 Valid 状态，守护进程 Open 主库前，根据主备库日志同步情况，将数据不一致备库的归档修改为 Invalid 状态。
2. 备库故障恢复，从主库同步历史数据后，守护进程将主库修改为 Suspend 状态，并将主库到备库的归档状态从 Invalid 修改为 Valid。当守护进程再次 Open 主库后，主备库数据重新恢复为一致状态。
3. 主库发送日志到实时备库失败挂起，守护进程处理 Failover 过程中，将主库到备库的归档状态修改为 Invalid。
4. 主库发送即时归档失败后，直接将主库到备库的归档改为 Invalid 状态。
5. 异步归档始终保持 Valid 状态，一旦归档失败马上返回，等待下一次触发再继续发送。



实时归档、即时归档只对 **Primary** 模式的主库有效，备库上配置的实时归档、说明：即时归档状态没有实际意义，始终保持 **Valid** 状态。

2.2.12 MAL 系统

MAL 系统是基于 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。DM 通过 MAL 系统实现 Redo 日志传输，以及其他一些实例间的消息通讯。

2.2.13 OGUID

数据守护唯一标识码，配置数据守护时，需要由用户指定 OGUID 值。其中数据库的 OGUID 在 MOUNT 状态下由系统函数 SP_SET_OGUID 设置，守护进程和监视器的 OGUID 值在配置文件中设定。

同一守护进程组中的所有数据库、守护进程和监视器，都必须配置相同的 OGUID 值，取值范围为 0~2147483647。

OGUID 的查询方式：

```
SELECT OGUID FROM V$INSTANCE;
```

2.2.14 守护进程组

配置了相同 OGUID 的两个或多个守护进程，构成一个守护进程组。为方便管理，对每个守护进程组进行命名，守护进程组中的所有守护进程和监视器，必须配置相同的组名。

2.2.15 组分裂

同一守护进程组中，不同数据库实例的数据出现不一致，并且无法通过重演 Redo 日志重新同步数据的情况，我们称为组分裂。引发组分裂的主要原因包括：

1. 即时归档中，主库在将 Redo 日志写入本地联机 Redo 日志文件之后，发送 Redo 日志到备库之前出现故障，导致主备库数据不一致，为了继续提供服务，执行备库强制接管。此时，当故障主库重启后，就会引发组分裂。

2. 故障备库重新完成数据同步之前，主库硬件故障，并且长时间无法恢复；在用户接受丢失部分数据情况下，为了尽快恢复数据库服务，执行备库强制接管，将备库切换为主库。此时，如果故障主库重启，也会造成组分裂。

检测到组分裂后，守护进程会修改控制文件为分裂状态，被分裂出去的数据库需要通过备份还原等技术手段重新恢复。

2.2.16 脑裂

脑裂是同一个守护进程组中同时出现两个或者多个活动主库，并且这些主库都接收用户

请求，提供完整数据库服务。一旦发生脑裂，将无法保证数据一致性，对数据安全造成严重后果。

DM 数据守护系统为预防脑裂做了大量工作，例如故障自动切换模式的数据守护必须配置确认监视器。确认监视器启动故障切换之前，会进行严格的条件检查，避免脑裂发生。守护进程一旦检测到脑裂发生，会马上强制退出主库，等待用户干预，避免数据差异进一步扩大。

造成脑裂的主要原因有两个：网络不稳定或错误的人工干预。为了避免出现脑裂，我们建议：

1. 设置 `dm.ini` 参数 `ALTER_MODE_STATUS=0`，限制用户进行直接通过 SQL 修改数据库模式、状态以及 `OGUID`。
2. 提供稳定、可靠的网络环境。
3. 配置自动切换数据守护时，将确认监视器部署在独立的第三方机器上，不要与某一个数据库实例部署在一起，避免由于网络问题触发自动故障切换，导致脑裂发生。
4. 通过人工干预，将备库切换为主库之前，一定要确认主库已经发生故障，避免主库活动情况下，备库强制接管，人为造成脑裂。

2.3 术语定义

数据守护系统包含主库、备库、守护进程、监视器等诸多部件，在主备库切换、故障处理等场景下，仅以主库或者备库这些称谓，已经无法准确描述对应部件。为了更加清晰的描述数据守护系统，特别定义以下术语。

表 2.2 数据守护术语

序号	术语	含义
1	实时主备	配置实时归档的主备系统
2	MPP 主备	配置实时归档的 MPP 集群主备系统
3	读写分离集群	配置即时归档或实时归档的主备系统
4	实时主库[实例名]	实时主备系统中 Primary 模式的库
5	实时备库[实例名]	实时主备系统中 Standby 模式的库
6	MPP 主库[实例名]	MPP 主备系统中 Primary 模式的库
7	MPP 备库[实例名]	MPP 主备系统中 Standby 模式的库

8	即时主库[实例名]	采用即时归档的读写分离主备系统中 Primary 模式的库
9	即时备库[实例名]	采用即时归档的读写分离主备系统中 Standby 模式的库
10	异步备库[实例名]	异步归档目标库，Standby 模式
11	异步源库[实例名]	异步归档源库，Primary/Standby 模式都可，实时、即时归档的主备库作为异步源库时，各库都需要设置异步备库作为归档目标，支持多源的配置，但是只有当前主库才会同步数据。
12	故障主库[实例名]	发生故障的 Primary 模式实例
13	故障备库[实例名]	发生故障的 Standby 模式实例
14	数据一致备库[实例名]	主库到当前备库归档处于有效状态，备库与主库数据保持一致
15	可恢复备库[实例名]	主库到当前备库归档处于失效状态，备库与主库数据存在差异，但主库归档日志涵盖备库缺失的数据
16	分裂库[实例名]	与主库数据不一致，且无法通过重做归档日志将数据恢复到一致状态的库
17	守护进程组[组名]	配置了相同 OGUID 的两个或多个守护进程，构成一个守护进程组
18	DMDSC 数据守护	主备库中包含数据共享集群（DSC）的守护系统
19	控制守护进程	守护 DMDSC 集群数据库控制节点的守护进程
20	普通守护进程	守护 DMDSC 集群数据库普通节点的守护进程
21	重演节点个数	备库上记录的 n_apply 个数，和当前正在重演的日志所对应的主库节点个数（n_ep）一致
22	监视器	基于监视器接口实现的命令行工具，用于监控、管理数据守护系统
23	确认监视器	运行在确认模式下的监视器
24	网络故障	指主备库之间网络断开，消息无法传递
25	网络异常	指主备库之间网络未断开，消息可以传递，但出现速度变慢等情形
26	备库故障	指备库出现软、硬件故障，导致数据库实例关闭
27	备库异常	指备库的数据库实例正常，但响应速度出现异常



配置数据守护时，数据库实例名不建议配置为 **Primary/Standby**，守护进

注意：程组名不建议配置成和实例名相同，避免在描述对象时产生歧义。

2.4 实时主备

实时主备系统由主库、实时备库、守护进程和监视器组成。通过部署实时主备系统，可以及时检测并处理各种硬件故障、数据库实例异常，确保持续提供数据库服务。

2.4.1 主要功能

实时主备系统主要功能包括：

1. 实时数据同步

主备库通过实时归档完成数据同步，实时归档要求主库将 RLOG_PKG 发送到备库后，再将 RLOG_PKG 写入本地联机 Redo 日志文件。但要注意的是，备库确认收到主库发送的 Redo 日志，并不保证备库已经完成重演这些 Redo 日志，因此主备库之间的数据同步存在一定的时间差。

2. 主备库切换

主备库正常运行过程中，可以通过监视器的 Switchover 命令，一键完成主备库角色转换。主备库切换功能可以确保在软、硬件升级，或系统维护时，提供不间断的数据库服务。

3. 自动故障处理

备库故障，不影响主库正常提供数据库服务，守护进程自动通知主库修改实时归档为 Invalid 状态，将实时备库失效。

4. 自动数据同步

备库故障恢复后，守护进程自动通知主库发送归档 Redo 日志，重新进行主备库数据同步。并在历史数据同步后，修改主库的实时归档状态为 Valid，恢复实时备库功能。

备库接管后，原主库故障恢复，守护进程自动修改原主库的模式为 Standby，并重新作为备库加入主备系统。

5. 备库接管

主库发生故障后，可以通过监视器的 Takeover 命令，将备库切换为主库，继续对外提供服务。如果配置为自动切换模式，确认监视器可以自动检测主库故障，并通知备库接管，这个过程不需要人工干预。

6. 备库强制接管

如果执行 Takeover 命令不成功，但主库可能由于硬件损坏等原因无法马上恢复，为

了及时恢复数据库服务，DM 提供了 Takeover Force 命令，强制将备库切换为主库。但需要由用户确认主库故障前，主库与接管备库的数据是一致的（主库到备库的归档是 Valid 状态），避免引发守护进程组分裂。

7. 读写分离访问

在备库查询的实时性要求不高的条件下，实时主备也可以配置接口的读写分离属性访问，实现读写分离功能特性。详见 2.6 读写分离集群章节。



执行 Takeover Force 有可能引发组分裂，而 Takeover 命令是在确保不说明：会产生组分裂情况下才允许执行。

2.4.2 归档流程

实时归档是实时主备数据同步的基础，其流程如下图所示：

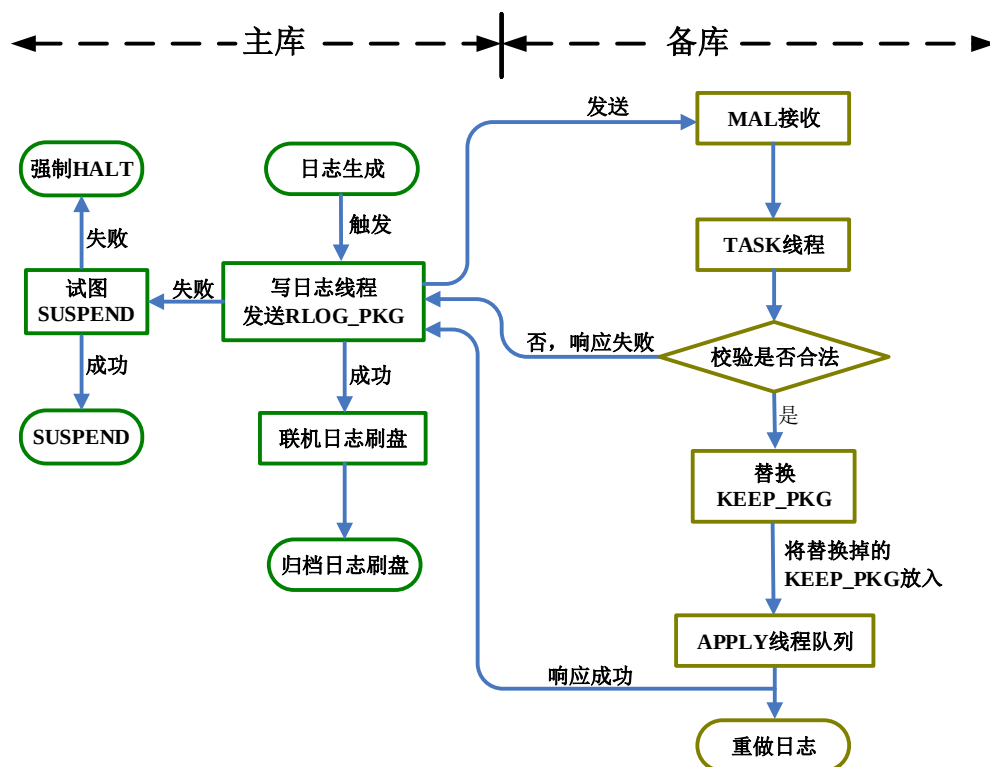


图 2.6 实时归档流程图

主库生成联机 Redo 日志，当触发日志写文件操作后，日志线程先将 RLOG_PKG 发送到备库，备库接收后进行合法性校验（包括日志是否连续、备库状态是否 Open 等），不合法则返回错误信息，合法则作为 KEEP_PKG 保留在内存中，原有 KEEP_PKG 的 Redo 日志加入 Apply 任务队列进行 Redo 日志重演，并响应主库日志接收成功。

2.5 MPP 主备

MPP 主备就是在 MPP 集群的基础上，为每一个 MPP 节点配置一套实时主备系统，这些实时主备系统一起构成了 MPP 主备系统。MPP 主备系统包含多个守护进程组，每个守护进程组都是一个相对独立的实时主备系统，具备实时主备的基本功能，可以进行主备切换、备库接管等操作。

MPP 主备的主要目的是为 DM MPP 集群提供数据可靠性保障，备库只做数据容灾、备份，MPP 备库并不是 MPP 集群的一部分，只是某个 MPP 节点（主库）的镜像。MPP 备库不参与 MPP 操作，与其他 MPP 备库之间没有任何关系，MPP 备库只能以单节点方式提供只读服务，但不提供全局的 MPP 只读服务。

MPP 主备系统中，一个守护进程 dmwatcher 可以监控、管理多个守护进程组的数据库实例。一般来说，一台物理机器上，可以部署 1 个 MPP 节点的主库和多个其他 MPP 节点的备库，充分利用硬件资源，节省投资。

Global 守护类型的 MPP 主备库需要在 dm.ini 中配置 MPP_INI 为 1，并且 MPP 主备库的本地数据文件目录下都需要有 dmmppctl 文件，如果 Global 守护类型的备库没有上述配置，守护进程和监视器无法正常使用，守护进程会切换到 Shutdown 状态，监视器上无法正常执行命令，会打印配置不一致的提示信息。

下图以三个 MPP 节点，每个节点配备两个备库为例，说明 MPP 主备系统的结构。

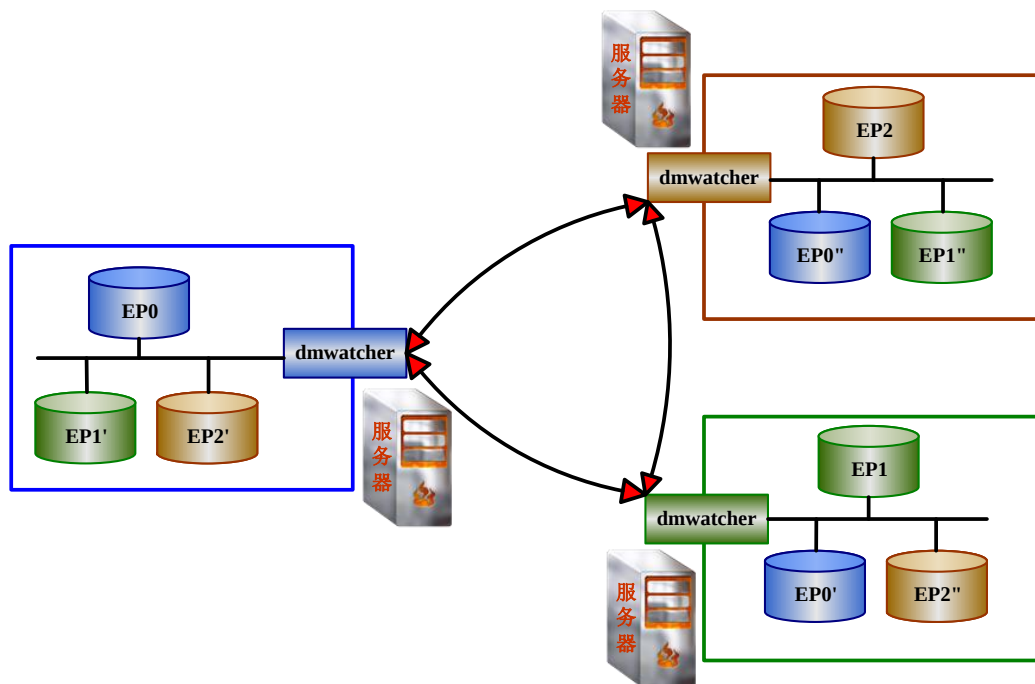


图 2.7 MPP 主备系统图

2.5.1 功能扩展

与实时主备系统相比，MPP 主备扩充了维护 MPP 控制文件功能，dmmp.ctl 控制文件记录了 MPP 集群的节点信息，在主备库切换或者备库接管后，必须将新的主库信息更新到 dmmp.ctl 文件中。

表 2.3 实时主备、MPP 主备比较

比较 \ 类型	实时主备	MPP 主备
数据库类型	单节点	MPP 集群
备库数量	1~8	1~8
归档类型	实时归档	实时归档
归档时机	写入联机日志前，发送到备库	写入联机日志前，发送到备库
数据来源	RLOG_PKG	RLOG_PKG
更新 dmmp.ctl	否	是（只更新 MPP 主节点上的 dmmp.ctl 文件）
是否支持 DMDSC 集群	支持	不支持

2.5.2 dmmp.ctl 维护

dmmp.ctl 控制文件的信息主要包括：系统状态、MPP 节点数、文件校验码、故障节点数、故障节点序号、配置项名、实例名、实例节点序号，以及根据节点数和实例序号生成的哈希映射数组等。MPP 集群中节点间的关联信息记录在 dmmp.ctl 控制文件中，因此，所有 MPP 主节点存放的 dmmp.ctl 文件内容要求完全一致。

MPP 主备系统中，要求所有节点（包括主库和备库）的 ctl_path 目录保存一份 dmmp.ctl 控制文件，并要求将所有节点的 dm.ini 参数 MPP_INI 配置为 1，以确保 MPP 主备系统正常运行。

当 MPP 主备系统中某个守护进程组的主节点发生变化（切换或接管），监视器会通知所有主节点更新本地 dmmp.ctl 文件，确保所有主节点 dmmp.ctl 控制文件始终保持一致。

当 MPP 主备系统中所有主节点都故障，需要进行强制接管时，守护进程会根据备库上的 dmmpp.ctl 文件，重新构造完整、有效的 dmmpp.ctl 内容，并最终更新新主库的 dmmpp.ctl 文件。

执行 MPP 主备库切换操作，需要修改所有节点的 dmmpp.ctl 文件，将新主库实例名替换原主库实例名。比如 0 号节点发生切换，只要将 0 号节点对应的 mpp_inst_name 修改为新主库的实例名。

下面根据图 2.7 MPP 主备系统图，举例说明一个节点主备切换时相应的控制文件是如何变化的。由于 dmmpp.ctl 是二进制文件，为了便于识别，下面列出的是对应转换的文本格式内容。原始配置如下：

```
[service_name1]                #配置项名

    mpp_seq_no      = 0        #节点序号

    mpp_inst_name   = EP0      #节点实例名

[service_name2]

    mpp_seq_no      = 1

    mpp_inst_name   = EP1

[service_name3]

    mpp_seq_no      = 2

    mpp_inst_name   = EP2
```

将 EP0 和 EP0' 切换后，dmmpp.ctl 中对应的实例信息如下，可以看出配置项中只有节点实例名 EP0 变化为 EP0' 了，节点序号以及其他配置项的内容都保持不变：

```
[service_name1]                #配置项名

    mpp_seq_no      = 0        #节点序号

    mpp_inst_name   = EP0'     #节点实例名

[service_name2]

    mpp_seq_no      = 1

    mpp_inst_name   = EP1

[service_name3]

    mpp_seq_no      = 2

    mpp_inst_name   = EP2
```

MPP 集群中用户登录某个节点创建一个会话 (session) 时，系统自动在其他 MPP 节

点上建立对应的镜像会话 (msession)，会话断开时，系统自动通知其他 MPP 节点释放对应会话（镜像会话）。MPP 主备切换时，整个 MPP 集群的所有连接被强制断开、所有会话被强制释放、并且切换过程中新的连接请求会被阻塞。主备切换完成后，连接请求会被分配到新的主节点上。

2.6 读写分离集群

读写分离集群是基于即时归档或实时归档实现的高性能数据库集群，不但提供数据保护、容灾等数据守护基本功能，还具有读写操作自动分离、负载均衡等特性。读写分离集群最多可以配置 8 个即时备库或 8 个实时备库，提供数据同步、备库故障自动处理、故障恢复自动数据同步等功能，也支持自动故障切换和手动故障切换两种守护模式。

一般情况下，应用系统中查询等只读操作的比例远大于 Insert/Delete/Update 等 DML 操作，修改对象定义等 DDL 操作的比例则更低。但是，这些操作往往混杂在一起，在高并发、高压情况下，会导致数据库性能下降，响应时间变长。借助读写分离集群，将只读操作自动分发到备库执行，可以充分利用备库的硬件资源，降低主库的并发访问压力，进而提升数据库的吞吐量。

读写分离集群不依赖额外的中间件，而是通过数据库接口与数据库之间的密切配合，实现读、写操作自动分离特性。DM 的 JDBC、DPI、DCI、ODBC、Provider 等接口都可以用来部署读写分离集群。

根据是否满足读提交事务隔离级特性，读写分离集群可以配置为事务一致模式和高性能两种模式。简单的说，事务一致模式下，不论一个 Select 语句是在备库执行、还是在主库执行，其查询结果集都是一样的。高性能模式则不能保证查询是一致的，备库的数据与主库的数据同步存在一定的延迟，当 Select 语句发送到备库执行时，返回的有可能是主库上一个时间点的数据。

2.6.1 归档流程

读写分离集群可以配置为即时归档，也可以配置为实时归档，这两种配置方式仅仅是归档流程上有差别，读写分离集群的特性仍然是一致的。即时归档流程与实时归档流程存在一定差异：

1. 主库先将日志写入本地联机 Redo 日志文件中，再发送 RLOG_PKG 到备库。

2. 备库日志重演时机有两种选择：

- 事务一致模式 要求备库在重演 Redo 日志完成后再响应主库。
- 高性能模式 与实时归档一样，收到 Redo 日志后，马上响应主库。

3. 即时归档的同步机制可以保证备库的 Redo 日志不会比主库的 Redo 日志多，因此即时备库不需要 KEEP_PKG，收到 RLOG_PKG 直接加入到 Apply 任务系统，启动 Redo 日志重演。

4. 备库故障或主备库之间网络故障，导致发送 RLOG_PKG 失败后，主库马上修改即时归档为 Invalid 状态，并切换数据库为 Suspend 状态。

5. 即时归档修改为 Invalid 状态后，会强制断开对应此备库上存在影子会话的用户会话，避免只读操作继续分发到该备库，导致查询数据不一致。

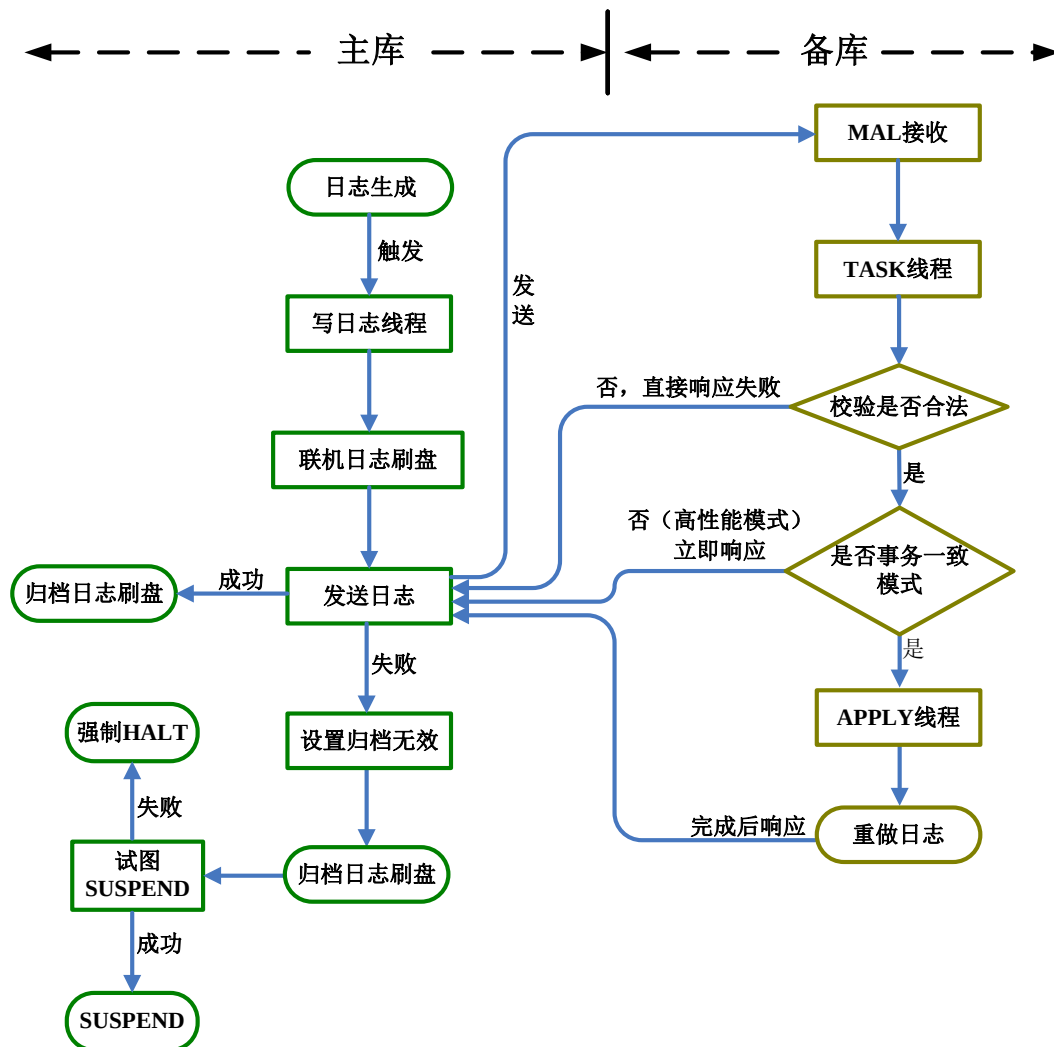


图 2.8 即时归档流程

2.6.2 实现原理

实现读写分离集群的基本思路是：利用备库提供只读服务、无法修改数据的特性，优先将所有操作发送到备库执行，一旦备库执行报错，则发送到主库重新执行。通过备库“试错”这么一个步骤，自然地将只读操作分流到备库执行。并且，备库“试错”由接口层自动完成，对应用透明。

读写分离集群数据库连接创建流程：

1. 用户发起数据库连接请求。
2. 接口（JDBC、DPI 等）根据服务名配置（在 `dm_svc.conf` 中进行配置）登录主库。
3. 主库挑选一个有效即时备库的 IP/Port 返回给接口。
4. 接口根据返回的备库 IP 和 Port 信息，向备库发起一个连接请求。
5. 备库返回连接成功信息。
6. 接口响应用户数据库连接创建成功。

接口在备库上创建的连接是读写分离集群自动创建的；对用户而言，就是在主库上创建了一个数据库连接。下图以配置了两个备库的读写集群为例，说明了读写分离集群的连接创建流程。

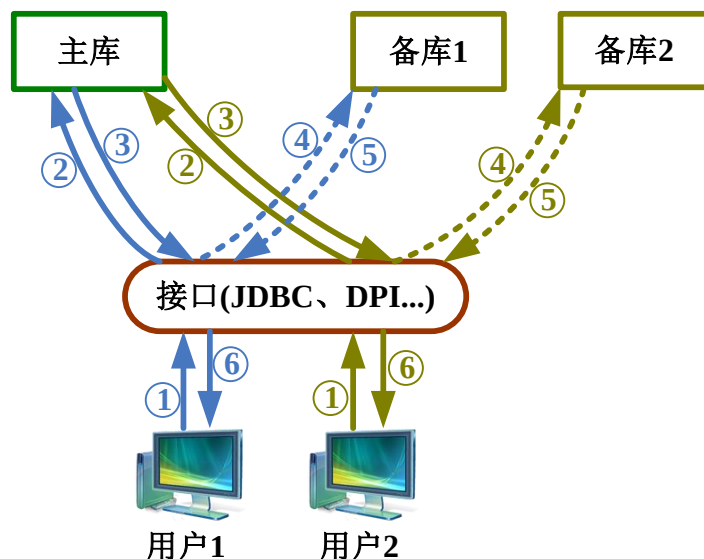


图 2.9 读写分离连接创建

读写分离集群语句分发流程：

1. 接口收到用户的请求。
2. 接口优先将 SQL 发送到备库执行。

3. 备库执行并返回执行结果。如果接口收到的是备库执行成功消息，则转到第 6 步，如果接口收到的是备库执行失败消息，则转到第 4 步。
4. 重新将执行失败的 SQL 发送到主库执行。只要第 3 步中的 SQL 在备库执行失败，则同一个事务后续的所有操作（包括只读操作）都会直接发送到主库执行。
5. 主库执行并返回执行结果给接口。一旦主库上执行的写事务提交，则下次继续从第 1 步开始执行。
6. 接口响应用户并将执行结果返回给用户。

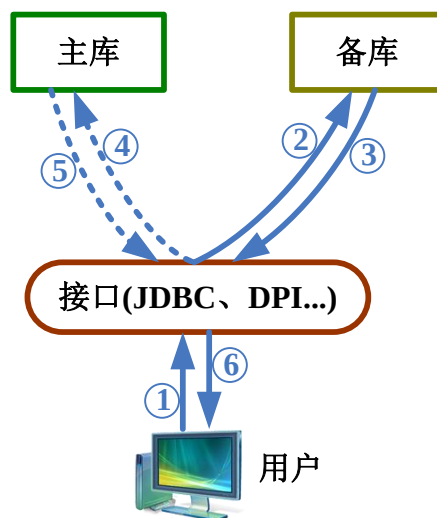


图 2.10 读写分离集群语句分发流程

执行举例说明：

```
--数据准备
CREATE TABLE T(C1 INT);

--事务开始
SELECT * FROM T;           --首先在备库上执行
INSERT INTO T VALUES(1);  --写操作转移到主库上执行
SELECT * FROM T;           --事务未提交，还在主库上执行
COMMIT;                    --事务提交
SELECT * FROM T;           --事务已提交，重新转移到备库上执行
```

2.6.3 事务一致性

读写分离集群通过 JDBC、DPI 等接口自动分发语句，一个事务包含的多个语句可能分

别在备库和主库上执行，但执行结果与单独在一个数据库实例上完全一致，满足读提交事务隔离级特性。

下面以两段伪代码 `trx1` 和 `trx2` 为例，说明读写分离的特性和如何实现事务隔离级。

`trx1:`

```
UPDATE T SET C1 = 2;  
  
COMMIT;
```

`trx2:`

```
RS = SELECT C1 FROM T;  
  
FETCH C1 FROM RS INTO VAR_X;  
  
INSERT INTO TX VALUES (VAR_X);  
  
COMMIT;
```

根据读写分离特性，`trx1` 的 `UPDATE` 在主库执行；`trx2` 的 `SELECT` 语句在备库执行，`INSERT` 语句转到主库执行，并且 `trx2` 的 `INSERT` 语句的插入值，是从之前执行的 `SELECT` 结果集中获取。

下面根据即时归档流程，结合 `trx2` 执行 `SELECT` 语句时机和以及 `trx1` 的不同状态进行讨论，详细地说明读写分离集群是如何实现提交事务隔离级别的。

1. 第一种情况，`trx2` 执行 `SELECT` 时，`trx1` 的 `COMMIT` 还未执行。

`trx2` 的 `SELECT` 语句，无论是在主库还是备库执行，查询结果都是 `trx1` 更新 `T` 表之前的值，`var_x = 1`。

2. 第二种情况，`trx2` 执行 `SELECT` 时，`trx1` 的 `COMMIT` 已经执行完成。

`trx2` 的 `SELECT` 语句，无论是在主库还是备库执行，查询结果都是 `trx1` 更新 `T` 表后的值，`var_x = 2`。

3. 第三种情况，`trx2` 执行 `SELECT` 时，`trx1` 正在执行 `COMMIT`。

`trx2` 的 `SELECT` 查询结果，与两个语句在系统内部的执行顺序有关，`var_x` 的值可能是 1 或者 2。但由于 `trx1` 的 `COMMIT` 并没有明确响应用户，`var_x` 的最终值取决于数据库管理系统的实现策略，无论返回 1 还是 2，都符合读提交事务隔离级。

为了保证主备库上的一致性，目前读写分离集群有下列一些类型的语句不会在备库上执行，都在主库上执行：

1. 设置会话、事务为串行化隔离级语句。
2. 表对象上锁语句（`LOCK TABLE XX IN EXCLUSIVE MODE`）。
3. 查询上锁语句（`SELECT FOR UPDATE`）。
4. 备份相关系统函数。

5. 自治事务操作。
6. 包操作。
7. 动态视图查询。
8. 设置自增列操作语句 (SET IDENTITY_INSERT TABLE ON)。
9. 临时表查询。
10. 访问 @@IDENTITY、@@ERROR 等全局变量。
11. SF_GET_PARA_STRING_VALUE、SF_GET_PARA_DOUBLE_VALUE 等函数。



说明:

读写分离集群中, 当一个 SQL 从备库切换到主库执行时, 主库会启动一个新的事务, 主库事务与备库事务没有任何联系, 事务 ID 也完全不同。备库事务 ID 与主库事务 ID 分配机制并不相同, 主库的事务 ID 取值范围是 [1 ~ 0x7FFFFFFFFFFFFFFF], 备库事务 ID 取值范围是 [0x800000000000 ~ 0xFFFFFFFFFFFFFFF]; 备库事务 ID 是一个内存值, 每次重启后都从 0x800000000000 开始重新分配; 主库事务 ID 是一个物理值, 一旦分配后, 就不会再重复分配。

2.6.4 性能调整

根据读写分离语句分发流程可以发现, 当一个应用系统中只读事务占绝大多数情况下, 可能出现备库高负载、高压, 主库反而比较空闲的情况。为了实现负载均衡, 更好地利用主备库的硬件资源, JDBC 等数据库接口提供了配置项, 允许将一定比例的只读事务分发到主库执行。因此, 用户应该根据主备库的负载情况, 灵活调整接口的分发比例 rwPercent 配置项, 以获得最佳的数据库性能。

备库数量是影响读写分离集群性能的一个重要因素, 备库越多则每个备库需要承担的任务越少, 有助于提升系统整体并发效率。另外, 事务一致模式下, 主库要等所有备库重演 Redo 日志完成后, 才能响应用户, 随着备库的增加, 即时归档时间会变长, 最终降低非只读事务的响应速度。因此, 部署多少备库, 也需要综合考虑硬件资源、系统性能等各种因素。

配置为高性能模式, 则是提升读写分离集群的另一个有效手段。如果应用系统对查询结果实时性要求并不太高, 并且事务中修改数据的操作也不依赖同一个事务中的查询结果。那么, 通过修改 dmarch.ini 中的 ARCH_WAIT_APPLY 配置项为 0, 将读写分离集群配置为高性能模式, 可以大幅提高系统整体性能。如果应用包含以下代码逻辑, 则不适合使用高性能模式,

能模式：

```
--tx1 事务开始

INSERT INTO T VALUES(1);           --写操作在主库上执行

COMMIT;                             --事务提交

--tx2 事务开始

SELECT TOP 1 C1 INTO VAR1 FROM T;    --tx1 事务已提交，SELECT 操作重新转移到备库上执行。高性能模式下备库可能还没有重做日志，查不出 tx1 中插入的结果

UPDATE T SET C1 = VAR1 + 1 WHERE C1 = VAR1;    --更新不到数据
```

此外，根据读写分离特性合理地规划应用的事务逻辑，也可以获得更佳的性能：

1. 尽可能将事务规划为只读事务和纯修改事务，避免无效的备库试错。
2. 读操作尽量放在写操作之前，用备库可读的特点来分摊系统压力。

2.6.5 实时归档的读写分离

实时主备也可以配置接口的读写分离属性进行访问，实现读写分离功能特性。

实时读写分离同样也支持事务一致模式和高性能模式，由配置文件 `dmarch.ini` 中的 `ARCH_WAIT_APPLY` 配置项来确定，1 表示事务一致模式，0 表示高性能模式，实时读写分离下，默认值为 0，即采用高性能模式。这个参数在实时归档中的用法和在即时归档中是相同的，只是默认值不同，事务一致模式和高性能模式的具体含义可参考“[2.2.11.5 即时归档](#)”小节。

和即时归档不同的是，实时归档先发送日志到备库，然后再写入本地联机日志，和即时归档相比，实时归档的读写分离可以有效避免备库自动接管后老主库的分裂，在对读写分离集群的可用性要求比较高的情况下，可以采用这种配置方式。



注意：

实时读写分离的事务一致模式仅在数据守护配置为自动切换模式下才会生效。

2.7 DMDSC 数据守护

DMDSC（数据共享集群）支持多个数据库实例同时访问、修改保存在共享存储中的数据，能够提供更高的数据库可用性和事务吞吐量。但由于数据是保存在共享存储上，当出现存储

失效等故障时，数据库服务将会中断。

DM 数据守护包含多个数据库，主库和备库部署在不同的机器上，数据分别保存在各自的存储上，主库传递 Redo 日志到备库，备库重演 Redo 日志实现数据同步。因此，DM 数据守护在容灾（特别是异地容灾）方面具有明显的优势。为了进一步提高 DMDSC 集群的数据安全性，以及系统的可用性，DM 提供了 DMDSC 集群数据守护功能。

DMDSC 集群数据守护功能与单节点数据守护保持一致，支持故障自动切换，支持实时归档与读写分离集群。支持 DMDSC 集群的守护，DMDSC（主）和单节点（备）、单节点（主）和 DMDSC（备）相互之间都可以作为主备库的数据守护。一般建议将 DMDSC 部署为主库，将单节点部署为备库。

2.7.1 总体结构

以下示例为 DMDSC 和单节点互为主备的守护系统结构简图：

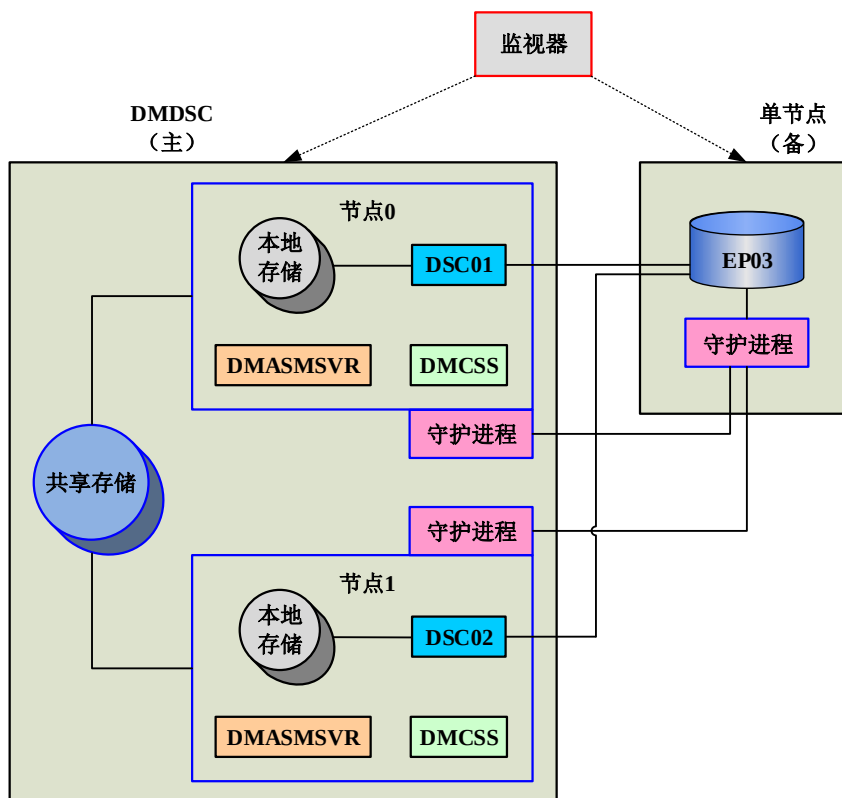


图 2.11 DMDSC 和单节点互为主备的守护系统结构简图

总体原则说明：

1. DMDSC 集群各个节点分别部署守护进程（dmwatcher）。
2. DMDSC 集群数据库控制节点的守护进程，称为控制守护进程，普通节点的守护进

程称为普通守护进程，如果控制节点发生变化，则控制守护进程也相应变化。

3. 守护进程会连接 DMDSC 集群所有实例，但只有控制守护进程会发起 OPEN、故障处理、故障恢复等各种命令。普通守护进程不处理用户命令，但接收其他库的控制守护进程消息。
4. 主备实时同步数据时，DMDSC 集群主库各个节点将各自产生的联机日志发送到备库进行重演。

2.7.2 系统连接

DMDSC 数据守护集群的守护进程和守护进程之间、监视器和守护进程之间、守护进程和 dmcss 之间都需要建立 TCP 连接，用于信息传递和命令执行。使用 TCP 连接的工具见下表：

表 2.4 工具的 TCP 连接说明

工具	TCP 连接说明
监视器 (dmmonitor)	连接所有的守护进程
守护进程 (dmwatcher)	1. 连接 DMDSC 集群内所有实例 2. 连接其他库的所有守护进程，但不连接 DMDSC 集群内其他实例的守护进程 3. 连接 DMDSC 集群内所有实例的 dmcss

TCP 连接的详细使用说明：

1. 控制守护进程连接所有 DMDSC 实例，发起 OPEN、故障处理、故障恢复等各种命令。普通守护进程不处理用户命令，但接收其他库的守护进程消息。
2. 控制守护进程定时发送消息给实例，普通守护进程不会发消息给实例。
3. 控制守护进程定时发送消息给其他库的守护进程以及监视器，普通守护进程只发送给监视器，不会发送给其他库的守护进程。
4. DMDSC 集群内部不同实例的守护进程之间不通信。
5. 普通守护进程连接 DMDSC 中所有实例，但只记录每个实例的信息，不做任何操作。同一个 DMDSC 集群内的所有守护进程之间不进行连接。
6. DMDSC 中每个实例存在多个守护进程的连接，向每个 dmwatcher 发送广播消息，但只可能接收控制守护进程的命令。

7. 不同库之间的守护进程都建立连接。
8. 守护进程和 dmcss 建立连接，部分监视器执行命令通过守护进程转发，由 dmcss 执行。

2.7.3 归档配置

1. DMDSC 集群必须配置远程归档，用于 DMDSC 节点故障后的数据同步。
2. 如果归档目标是 DMDSC 集群，则归档的目标节点需要同时配置 DMDSC 集群所有实例，一个 DMDSC 集群作为一个整体进行配置，`realtime/timely/async` 归档配置要求 `ARCH_DEST` 配置目标 DMDSC 集群所有节点信息，以 ‘/’ 分割。



注意：

必须保证 DMDSC 集群中所有节点的归档配置是相同的。

2.7.4 日志发送

1. 异步归档日志发送

主库是单节点时，单节点实例直接收集本地的归档日志发送到备库。

主库是 DMDSC 集群时，控制节点扫描本地归档和远程归档目录，收集所有节点的归档日志文件，并发送到备库，普通节点不发送归档日志。

2. 实时/即时归档日志发送

单节点和 DMDSC 集群采用相同的处理逻辑，各节点将本实例产生的 Redo 日志直接发送到备库重演实例。

2.7.5 重演实例

当 DMDSC 集群作为备库时，只由集群内的一个节点进行日志重演，主库的归档配置中包含了 DMDSC 集群内所有节点，主库发送归档之前需要确定其中一个节点作为归档目标，称为重演实例。非重演实例收到重做日志直接报错处理。

DM 规定将 DMDSC 备库的控制实例作为重演实例，并且要求 DMDSC 备库的控制实例必须是 0 号节点。主库启动时，对于单节点备库，默认归档目标就是备库实例；对于 DMDSC

备库，如果 0 号节点不是控制实例，则 DMDSC 备库将无法正常运行，具体限制策略请参考 2.7.10 小节描述。

2.7.6 备库日志重演

DMDSC 实现机制保证多个实例不能同时修改一个数据页，不同节点对同一个数据页修改产生的 LSN 一定是唯一、递增的。单个节点 Redo 日志的 LSN 可能不连续，但所有节点 Redo 日志归并在一起后，LSN 一定是连续的。但是，全局 Redo 日志归并后，可能存在 LSN 重复的非 PWR 日志。

为了降低备库重演不同节点日志时的 LSN 同步等待代价，备库收到不同节点的日志时，会按照日志包之间的依赖关系依次将不同节点的日志包按照顺序加入同一个重演任务系统，在开启并行重演的情况下，每个日志包会再交给多路并行重演线程进行并行重演，以此确保备库重演的性能。

2.7.7 守护控制文件

控制守护进程检测到本地 DMDSC 集群分裂时，会自动在 dm.ini 的 SYSTEM_PATH 路径下创建 dmwatcher.ctl 文件，记录 Split 分裂状态和分裂描述信息。如果数据库控制节点发生切换，控制守护进程也随之切换，新的控制守护进程从 SYSTEM_PATH 目录加载 dmwatcher.ctl 文件。因此，为了保证所有守护进程能访问 dmwatcher.ctl 文件，要求 SYSTEM_PATH 必须配置在共享存储上。

2.7.8 远程归档修复

远程归档（REMOTE）的实现机制允许某些场景下将远程归档失效，不保证远程归档始终处于有效状态，导致远程归档日志可能缺失，例如如下场景：

1. Open force 命令启动时会导致从节点的远程归档日志丢失
2. DMDSC 集群故障节点重加入后，可能会缺失部分节点的远程归档日志
3. 多个节点同时故障，DMDSC 全部节点强制退出，也可能会丢失远程归档日志
4. 节点间网络故障
5. 刷远程归档日志失败等

基于以上情况，需要增加远程归档修复处理，补齐远程归档日志。目前有两种方式，一种是服务器启动时根据联机日志自动修改远程归档文件，无需用户参与；一种是通过执行系统过程进行修复。

归档修复的系统过程包含如下：

1) SP_REMOTE_ARCHIVE_REPAIR

```
void  
  
SP_REMOTE_ARCHIVE_REPAIR (  
  
src_inst_name      varchar(256),  
  
dest_inst_name     varchar(256)  
  
)
```

功能说明：

在 dest_inst_name 以外的 DMDSC 实例上调用，修复 dest_inst_name 上的 src_inst_name 发到 dest_inst_name 上的远程归档。该函数自动收集 dest_inst_name 上缺失的来自 src_inst_name 的远程归档日志区间，src_inst_name 上根据每个区间发送对应的归档到 dest_inst_name 上，在 dest_inst_name 上生成归档文件，保存在对应的远程归档目录中。

参数说明：

src_inst_name: 远程归档源实例名
dest_inst_name: 待修复的远程归档目的实例名

返回值：

无

举例说明：

修复 DSC02 上来自 DSC01 的远程归档：

```
SP_REMOTE_ARCHIVE_REPAIR('DSC01','DSC02');
```

2) SF_REMOTE_ARCHIVE_CHECK

定义：

```
int  
  
SF_REMOTE_ARCHIVE_CHECK(  
  
inst_name          varchar(256)  
  
)
```

功能说明：

检查本地保存的源实例远程归档目录中归档是否完整

参数说明：

inst_name: 远程归档源实例名

返回值：

检查失败返回报错 code

0: 不完整, 需要修复; 1: 完整, 无需修复

举例说明:

登录 DSC01, 检查 DSC02 源实例的对应的远程归档是否完整

```
SELECT SF_REMOTE_ARCHIVE_CHECK('DSC02');
```

3) SF_REMOTE_ARCHIVE_TRUNCATE

定义:

int

SF_REMOTE_ARCHIVE_TRUNCATE (

inst_name varchar(256)

)

功能说明:

截断远程归档中不连续部分, 使远程归档连续。若指定 inst_name 为当前节点实例名, 则直接返回。建议在使用 SF_REMOTE_ARCHIVE_REPAIR 之后, 执行 SF_REMOTE_ARCHIVE_CHECK 仍不完整的情况下使用, 避免删除多余归档。但是即使误删除多余归档, 也可以使用其他节点修复。

参数说明:

inst_name: 远程归档源实例名

返回值:

检查失败返回报错 code

0: 截断完成 ; 1: 当前节点实例无需截断

举例说明:

登录 DSC01, 截断 DSC02 源实例的对应的远程归档

```
SELECT SF_REMOTE_ARCHIVE_TRUNCATE ('DSC02');
```

2.7.9 DMDSC（主）和单节点（备）

DMDSC 集群作为主库时, 不同节点可能修改同一个数据页并各自产生日志, 因此 DMDSC 集群的备库在重演同一个数据页的日志时, 必须严格按照各节点修改的先后顺序进行重演, 为了确保正确的重演顺序, DMDSC 集群的节点在修改数据页时, 会记录下修改此数据页的前序节点和对应的 LSN 值, 并将此信息写入日志包, 组成日志包的依赖信息。

备库重演主库的日志包时, 根据日志包上记录的对其他节点的依赖信息 (包括节点号和 LSN), 会等待依赖的节点 (即修改相同数据页的前序节点) 先重演到依赖信息中记录的 LSN 位置, 然后再启动当前日志包的重演, 确保重演顺序和主库上数据页的修改顺序完全一致。

主库通过异步方式发送日志时, 也会通过日志包的依赖信息判断是否存在日志缺失 (所

依赖的其他节点的日志包是否存在)，如果存在日志缺失则会终止本次日志发送，备库不会因为无法等到其他节点的日志出现重演卡住的情况。

2.7.10 单节点（主）和 DMDSC（备）

一般情况下，在部署 DMDSC 数据守护时，建议将 DMDSC 集群作为主库，将单节点作为备库来使用，但实际使用时，有可能会碰到主备切换的情况，出现单节点为主库，DMDSC 集群为备库的情况。DMDSC 集群作为备库运行时，有一些限制策略，具体说明如下。

- 1) DMDSC 备库的控制节点要求必须是 0 号节点，只能由 0 号节点重演主库的日志，如果 0 号节点故障，其他节点会被强制关闭。
- 2) 在 DMDSC 备库的 0 号节点故障恢复前，其他节点不允许正常启动运行。
- 3) DMDSC 备库的非 0 号节点生成的 PWR 日志会正常写入归档文件，以保障后续切换为主库时能够构建出有效的日志包依赖关系。
- 4) DMDSC 集群在主库模式下，如果 0 号节点是非控制节点，则不允许切换为备库模式。

2.7.11 DMDSC 集群的管理规则

2.7.11.1 控制守护进程的认定

控制节点本地的守护进程就是控制守护进程。

一旦控制节点故障，控制守护进程降级为普通守护进程。

普通守护进程一直保持在 Startup 状态下，控制守护进程可以进行各种状态切换。

监视器部分命令比如 Show 命令、Startup database 命令等执行时，可能 DMDSC 所有节点都不是活动状态，此时需要选出一个 dmwatcher 作为控制守护进程，对此制定如下规则：

1. 控制节点状态正常，并且本地的 dmwatcher 是活动的，则直接选择此节点的 dmwatcher 为控制守护进程。

2. 控制守护进程降级或者故障，系统中还存在其他活动的 dmwatcher，系统将按照下面的原则自动选择控制守护进程。

- 1) 控制节点本地的 dmwatcher 正常（控制节点发生故障导致降级），并且和其他活

动 dmwatcher 上记录的控制节点信息一致（DMDSC 集群还未选出新的控制节点），则仍然选择降级后的控制守护进程作为新的控制守护进程。

2) 控制节点本地的 dmwatcher 故障（控制节点可能故障也可能正常），其余活动 dmwatcher 上记录的控制节点信息一致，则在所有活动 dmwatcher 中，找出记录的 FSEQ/FLSN 最大的 dmwatcher 作为控制守护进程，如果所有 dmwatcher 上记录的 FSEQ/FLSN 信息相同，则返回本地 ep seqno 最小的 dmwatcher 作为控制守护进程。

3. 所有 dmwatcher 都发生故障（控制节点可能故障也可能正常），系统将按照下面的原则自动选择控制守护进程。

1) 这些故障 dmwatcher 上的控制节点信息一致，则从曾经收到的历史消息中取控制节点本地的故障 dmwatcher 作为控制守护进程（退出整个 DMDSC 集群及 dmwatcher 的情况）。

2) 这些故障 dmwatcher 上的控制节点信息不一致，则取最后一次收到过消息的 dmwatcher 作为控制守护进程，monitor 可以从这个 dmwatcher 上取出最新的 ep 信息。

2.7.11.2 守护进程名与实例名的对应关系

守护进程守护的本地实例名，称为守护进程名。

单节点的守护进程名，就是单节点的实例名。

对于 DMDSC 库的守护进程，由于收到的远程守护进程实例消息都是控制守护进程发送过来的，控制守护进程守护的是 DMDSC 控制节点，因此远程守护进程名也就是远程 DMDSC 库的控制节点实例名。

2.7.11.3 DMDSC 库模式和状态的认定

同一个 DMDSC 集群内，理论上所有节点实例的模式是相同的，因为修改模式的动作是同步执行的，登录任意一个节点就可以完成所有节点的模式修改，但修改状态的动作是异步的，DMDSC 集群允许不同的节点工作在不同的状态下，为了方便管理整个 DMDSC 集群，守护进程将 DMDSC 集群看作一个库，按照以下规则来认定 DMDSC 集群当前的模式和状态。

DMDSC 库获取模式和状态规则：

1. 主实例还未选出，或者不存在正常的实例，则无法判断模式状态。

2. 只根据 OK 数组中的节点来判断模式状态，在获取节点状态时要求 DMDSC 集群处于 Open 状态。
3. 如果存在模式不同的实例，则无法判断。
4. 如果存在非 Suspend/Mount/open 状态的节点实例，则直接返回此节点状态作为 DMDSC 集群状态。
5. 如果实例状态不一致，则按照优先级方式确定 DMDSC 集群当前的状态，Suspend 状态优先级最高，Mount 次之，Open 最低。

根据以上规则，如果无法认定 DMDSC 集群模式，则认为 DMDSC 集群为 Unknown 模式，如果无法认定 DMDSC 集群状态，则认为 DMDSC 集群当前为 Shutdown 状态。

如果节点状态不一致，守护进程按照优先级规则判定当前 DMDSC 集群应该处于什么状态（见下表），并将所有节点统一到这个状态。在各节点状态统一一致后，守护进程再根据本地和远程状态进一步处理，如执行命令、自动恢复、故障处理等。

表 2.5 不同的状态判定规则

判定状态 节点状态	节点状态	Open	Suspend	Mount
Open		Open	Suspend	Mount
Suspend		Suspend	Suspend	Open
Mount		Mount	Open	Mount

此表格第一行、第一列为节点状态，中间内容为不同行、列组合下守护进程经判定之后的状态，具体的场景介绍请参考 2.7.10 [场景说明](#) 小节。

需要注意的是，如果一个节点为 Open，一个节点为 Mount，并且 Mount 状态的节点是故障重加入的节点，则守护进程会直接通知此节点 Open，而不是先统一到 Mount 状态，再进行 Open。

2.7.11.4 DMDSC 故障检测时间与守护进程故障认定时间

DMDSC 集群出现节点故障，活动节点一旦检测到 MAL 链路出现异常，立即启动故障处理，进入 HPC_CRASH_RECV 状态。判断 MAL 链路异常，涉及到 MAL 配置中链路检测 MAL_CHECK_INTERVAL 参数。

守护进程根据 INST_ERROR_TIME 配置的时间超时检测实例是否故障，守护进程的故

障处理优先级低于 DMDSC 的故障处理，也就是 `INST_ERROR_TIME` 值至少要大于 DMDSC 故障检测时间和 MAL 链路检测时间中的最小值，否则守护进程启动时会强制调整 `INST_ERROR_TIME` 值 $> \min(\text{DMDSC 故障检测时间}, \text{MAL_CHECK_INTERVAL}) + 5$ ，避免守护进程早于 DMDSC 集群内部故障检测时间，过早认定实例故障。

2.7.11.5 DMDSC 库 OK/ERROR 的认定

认定原则如下：

1. 如果找不到控制节点，则认为 ERROR；
2. 如果在 DMDSC 的 `OK_EP` 数组中，存在非 OK 状态的实例，则认为 ERROR；
3. 其余情况认为 DMDSC 库是 OK 的。

2.7.11.6 接收 DMDSC 库消息超时

守护进程根据 `inst_error_time` 值判断接收本地库消息是否出现超时：若控制节点还未选出，认为 DMDSC 集群还未启动正常，则认定为消息超时；只要 DMDSC 库内有一个实例的消息未超时，就认为消息未超时。

2.7.11.7 接收远程守护进程消息超时

下面几种情况认为接收远程守护进程消息超时：

1. 如果守护进程的链路已经断开，认为超时；
2. 如果接收远程守护进程消息超过配置的 `DW_ERROR_TIME` 时间，则认为超时。

如果判断为消息超时，则设置远程守护为 ERROR 状态。

2.7.11.8 SSEQ/SLSN 和 KSEQ/KLSN 的获取

1. SSEQ/SLSN 的获取

对于主库，这个值取的是主库实例的 `FSEQ` 和 `FLSN`。如果主库是 DMDSC 集群，则取出的 `SSEQ/SLSN` 是一个数组，对应存放的是每一个节点实例的 `FSEQ` 和 `FLSN`，数组长度和主库节点个数一致。

对于备库，这个值取的是备库明确可重演到的最大 GSEQ 值和最大 LSN 值。如果主库是 DMDSC 集群，则对应取出的也是一个数组，数组个数和主库节点个数一致。

2. KSEQ/KLSN 的获取

对于主库，这个值取的是主库实例的 CSEQ 和 CLSN。如果主库是 DMDSC 集群，则取出的 CSEQ/CLSN 是一个数组，对应存放的是每一个节点实例的 CSEQ 和 CLSN，数组长度和主库节点个数一致。

对于备库，这个值取的是备库已经收到、未明确是否可以重演的最大 GSEQ 值和最大 LSN 值。如果主库是 DMDSC 集群，则对应取出的也是一个数组，数组个数和主库节点个数一致。

2.7.11.9 DMDSC 主库发送归档异常

守护进程可以通过配置参数 RLOG_SEND_THRESHOLD 监控主库到备库的归档发送情况。如果主库是 DMDSC 集群，则需要统计主库每个节点到备库控制节点的归档发送情况。DMDSC 集群中，主库任意一个节点归档发送异常，就认为出现异常，需要切换到 Standby_check 状态下对归档发送异常的备库进行处理。

异常判断前提：

1. RLOG_SEND_THRESHOLD 参数配置值大于 0。

2. 存在有归档处于有效状态的备库，对 DMDSC 集群，只需要主库任意一个节点实例到备库的归档有效即可。

异常判断规则：

主库最近 N 次（N 不超过主库 dm.ini 设置的 RLOG_SEND_APPLY_MON 值）到某个归档状态有效的备库发送归档的平均耗时超过设置的 RLOG_SEND_THRESHOLD 值。对 DMDSC 集群，如果当前守护进程处于 Recovery 状态，则只看控制节点到备库的归档发送情况。

2.7.11.10 DMDSC 备库重演异常

守护进程可以通过配置参数 RLOG_APPLY_THRESHOLD 监控备库的日志重演情况。当备库达到异常判断前提时，系统会对备库启动异常判断。如果备库达到异常判断规则中的情况，则认为备库异常。

异常判断前提：

1. RLOG_APPLY_THRESHOLD 参数配置值大于 0。
2. 备库上已经选出控制节点（重演实例，0 号节点），且存在重演的信息。

异常判断规则：

备库最近 N 次（N 不超过备库 dm.ini 设置的 RLOG_SEND_APPLY_MON 值）的平均等待时间加上真正的平均重演时间超过设置的 RLOG_APPLY_THRESHOLD 值，则认为备库异常。

2.7.11.11 DMDSC 备库重演相关判断

主库一定是单节点，备库重演实例（0 号节点）上只有该单节点对应的重演信息，如果重演信息的 KSEQ 大于 SSEQ，就认为存在 KEEP_PKG。

在故障备库恢复，守护进程判断其是否可加入主备系统时，守护进程会判断备库的重演实例是否重演完成，判断方法是根据主库在备库重演实例上重演的 ASEQ/ALSN 和 SSEQ/SLSN 判断，如果都相等，说明重演完成，这里不考虑是否存在 KEEP_PKG，备库的 KEEP_PKG 会在重加入前丢弃。

可以在备库重演实例上查询相关动态视图查看重演进度，包括 V\$KEEP_RLOG_PKG、V\$RAPPLY_SYS、V\$RAPPLY_LOG_TASK、V\$RAPPLY_PARALLEL_INFO 等。

2.7.12 场景说明

DMDSC 数据守护环境比单节点数据库更加复杂，增加了一些额外的处理逻辑，本小节针对 DMDSC 数据守护使用场景进行说明。

2.7.12.1 DMDSC 实例状态不一致

■ Mount/Open 状态

按照“2.7.9.3 DMDSC 库模式和状态的认定”的表格，这种情况的目标状态应该是 Mount，但是在 DMDSC 控制节点为 Open 状态，其他 Mount 状态的故障节点恢复，执行故障重加入时，直接将重加入的节点从 Mount 切换到 Open 状态，减少不必要的状态切换。

■ Suspend/Open 状态

DMDSC 主库的某个实例发送实时或即时归档失败切换为 Suspend 状态,通知其他节点切换为 Suspend 状态前,系统各个实例状态在短时间内不一致,最终会自动统一为 Suspend 状态。这种归档失败引起的 Suspend 状态切换,由 DMDSC 实例自主完成。

■ Open/Suspend 状态

DMDSC 主库同步历史数据到备库,转入 Suspend 状态同步所有归档日志后,修改备库归档状态为有效,并重新 Open 主库。控制节点先 Open 成功,普通节点 Open 失败(出现实时备库故障、网络异常、备库日志校验失败等原因),导致 DMDSC 主库各实例处于不一致状态。

守护进程会启动相应的处理流程,将所有实例切换为 Suspend 状态,失效归档以后,再重新 Open。

■ Mount/Suspend 状态

主备库 switchover 流程中,切换 DMDSC 主库为 Mount 状态过程中,当节点 DSC01 切换成 Mount, DSC02 还处于 Open 状态时,出现备库故障。DSC02 归档失败,切换为 Suspend 状态,导致出现 Mount/Suspend 状态组合。

由于服务器不支持 Suspend 和 Mount 状态间的切换,既无法将所有实例直接切换为 Mount 状态,也不能将所有实例直接切换为 Suspend 状态。守护进程的处理流程是,将所有 DMDSC 实例的归档失效后,先将 Suspend 状态实例重新 Open,再将 Mount 实例重新 Open,最终将所有实例状态统一为 Open 状态。

2.7.12.2 DMDSC 主库节点故障

由于实时归档先发送 Redo 日志到备库,再写入本地的联机日志文件,DMDSC 主库出现节点故障时,控制守护进程需要根据故障节点的 LSN 情况,通知备库丢弃或者应用 KEEP_PKG。而即时归档先将 Redo 日志写入本地的联机日志文件,再发送到备库,因此主库故障节点可能存在已写入日志文件,但未发送到备库的 Redo 日志。

为了简化处理流程,DMDSC 集群主库节点故障时,故障处理挂起工作线程后,强制所有节点的实时、即时归档失效。在主库 DMDSC 故障处理完成后,由主库守护进程重新启动恢复流程,同步主备库数据。

2.7.12.3 DMDSC 备库重演实例故障

当 DMDSC 备库控制节点（重演实例，0 号节点）故障，导致主库发送归档失败而挂起，主库守护进程进入 Failover 处理流程，设置归档失效后再 Open。

DMDSC 备库的普通节点会主动关闭，必须等到 0 号节点故障恢复后，其他节点才可以继续作为普通节点正常运行，具体限制策略请参考 2.7.10 小节。

2.7.12.4 DMDSC 备库非重演实例故障

如果 DMDSC 普通节点故障，重演实例可以正常接收主库日志，但 DMDSC 故障处理过程中，挂起工作线程等操作可能会导致日志重演挂起；备库重做 Redo 日志，可能需要访问故障节点的 GBS 等全局资源，也可能导致日志重演卡住。也就是说，虽然备库重演实例处于正常状态，但备库的日志重演仍然可能挂起。如果主库归档保持有效状态，继续发送 Redo 日志到备库，就有可能引发备库日志堆积，在 Redo 日志堆积过多情况下，还会导致主库日志延迟刷盘，进而影响主库的系统服务。

因此，在备库非重演实例故障时，守护进程也会进入 Failover 状态，启动故障处理，通知主库将实时、即时归档失效。在下述场景，守护进程会启动故障处理：

1. DMDSC 备库实例间 MAL 链路异常。
2. DMDSC 普通实例收到 Redo 日志，直接报错返回。
3. DMDSC 集群系统不是 DSC_OPEN 状态，直接报错返回。

2.7.12.5 主备库网络异常

主备库之间网络出现异常时，主库发送归档失败，导致主库节点挂起，如果主库是 DMDSC 集群，任何一个节点挂起，都会通知其他节点同步挂起（参考 2.7.10.1 DMDSC 实例状态不一致），守护进程会自动将主库转入 Failover 状态处理。

2.7.12.6 DMDSC、守护进程、监视器的并发处理

DMDSC 数据守护中，DMDSC 集群内部故障处理、故障节点重加入、守护进程 Failover 处理、Recovery 处理、监视器命令等可能会并发操作。为了确保在并发场景下各种命令能

正确地执行，DM 增加了命令执行的中断机制，并为每个命令分配了不同的执行优先级。

各种操作标记说明如下表：

表 2.6 操作标记说明

序号	标记名称	操作说明
1	DSC_CRASH_RECV	DSC 故障处理
2	DSC_ERR_EP_ADD	DSC 故障重加入
3	DW_FAILOVER	守护进程故障处理
4	DW_STDBY_CHECK	守护进程异常检测
5	DW_RECOVERY	守护进程备库恢复
6	MON_SWITCHOVER	监视器主备切换命令
7	MON_TAKEOVER	监视器接管命令
8	MON_OPEN_FORCE	监视器 OPEN 命令
9	MON_CLEAR_SEND_INFO	监视器清理主库归档发送信息命令
10	MON_CLEAR_RAPPLY_INFO	监视器清理备库重演信息命令
11	MON_LOGIN_CHECK	监视器登录命令
12	MON_MPPCTL_UPDATE	监视器更新 MPPCTL 命令
13	MON_CHANGE_ARCH	监视器设置归档命令
14	NONE	没有任何命令执行标记

这些操作的优先级由高到低排序为：

1. DMDSC 故障处理（CRASH_RECV）优先级最高。
2. ERR_EP_ADD /FAILOVER/ STANDBY_CHECK/监视器上需要和服务器交互的命令（可以被 CRASH_RECV 中断）。
3. RECOVERY （可以被以上操作中断）。

其他守护进程状态以及不需要和服务器交互的监视器命令，不需要进行并发控制。

2.7.12.7 DMDSC 主动宕机场景

以下一些场景会导致 DMDSC 实例主动宕机：

1. dmasmsvr 故障，对应节点实例会主动宕机。

2. 处于 Suspend 或 Mount 状态下，出现节点故障。
3. 启动过程中，检测到 INI 参数 TS_MAX_ID 配置不一致，可能引发强制宕机。
4. 故障重启后，没有将所有重做 Redo 日志修改的数据页刷盘前（控制节点的 CKPT_LSN 小于最大重做 LSN），控制节点故障，所有普通节点主动宕机。

2.8 异步备库

异步备库一般用于历史数据统计、周期报表等对数据实时性要求不高的业务场合。异步归档时机可以选择在源库空闲的时候，可避免源库的业务高峰期同步数据对性能的影响。

每个 Primary 或者 Standby 模式的库都可以配置最多 8 个异步备库。配置了异步备库的 Primary 或者 Standby 模式的库，统称为源库。可以在实时主备、MPP 主备和读写分离集群的主库和备库上配置异步备库，异步备库可级联配置，异步备库本身也可以作为源库配置异步备库。一个监视器最多可以同时监视 16 个异步备库。

配置了 Realtime 和 Timely 归档的主备库，允许配置同一个异步备库，也就是说一个异步备库允许有多个源库。系统自动识别，Primary 模式的源库负责向异步备库同步数据，主备库切换后，由新主库负责向异步备库同步数据。这种多源配置的异步备库，可以保证异步备库始终与当前的有效主库保持数据同步。但是，未配置 Realtime 和 Timely 归档的库，不能配置同一个异步备库。对于级联方式配置的异步备库，也就是异步备库自己作为源库的情况下，仍然需要由用户保证配置的正确性，避免数据同步异常。错误配置多个源库的情况下，多个源库可能同时、或交错向异步备库发送归档日志，导致备库日志连续性校验失败。

异步备库支持延时和定点重演功能，相关使用场景及使用说明请参考“6.25 异步备库延时和定点重演”小节。

配置异步备库十分简单，在源库的 dm.ini 中打开定时器 TIMER_INI 开关，同时配置文件 dmtimer.ini，在 dmarch.ini 中增加异步归档配置，在 dmmal.ini 中增加异步备库的 mal 配置，由定时器定时触发源库发送归档日志到异步备库即可。异步归档的最小触发间隔是 1 分钟，详细的配置示例请参考 7.6 配置异步备库小节。异步备库可以不配置本地归档。

下图中，“源库 P”表示 Primary 模式的库，“源库 S”表示 Standby 模式的库，“源库 S”上允许配置到同一个异步备库的异步归档，但只有“源库 P”会向异步备库同步数据，

对于级联方式配置的异步备库，不允许有多个源库。

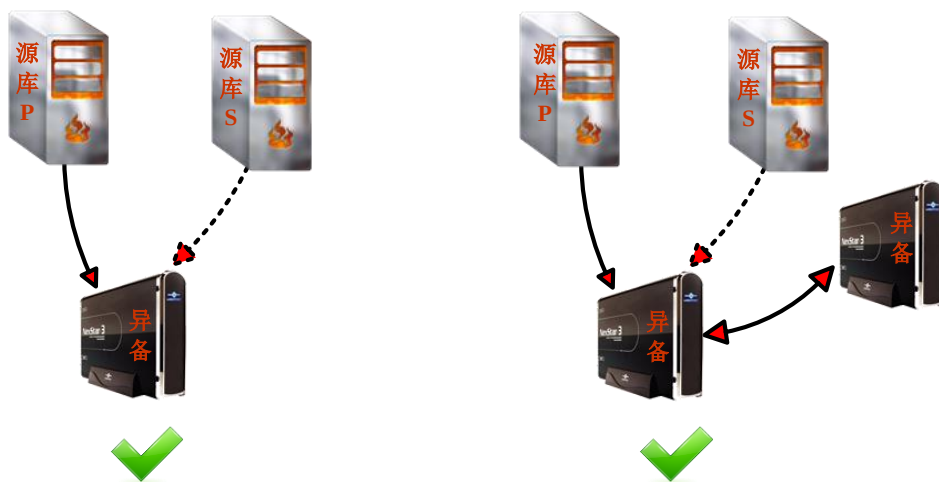


图 2.12 多源/级联异步备库配置

2.9 Huge 表

DM 的 Huge 表存储使用独立的存储机制，分别存储在数据文件和辅助表中。数据文件以数据区为单位进行管理，一个数据区可以存储单列数据的若干行，辅助表则是普通的行表，用于存放控制信息或者部分 Huge 表数据。

Huge 表又分为非事务型 Huge 表和事务型 Huge 表，其中非事务型 Huge 表不具备事务特性，因此数据守护 V4.0 不再对这种类型的 Huge 表进行支持，仅支持事务型 Huge 表的主备数据同步。

数据守护 V4.0 支持对事务型 Huge 表生成普通的 Redo 日志，和普通行表采用相同的数据同步机制，备库可以直接通过重演 Redo 日志来完成 Huge 表数据的同步，并且不区分实时主备、读写分离集群或者 MPP 主备，这三类数据守护系统都可以支持此功能，由于 Huge 表特殊的存储及管理方式，DMDSC 集群目前无法支持 Huge 表，也就是仅限于在单节点的数据守护系统中使用 Huge 表。

需要注意的是，在搭建数据守护系统时，需要将建库参数 `HUGE_WITH_DELTA` 和 `RLOG_GEN_FOR_HUGE` 都配置为 1，才能够正常生成 Huge 表 Redo 日志并发送给备库重演。对升级上来的老的数据守护系统，这两个建库参数则按照 0 值处理，Huge 表不会生成 Redo 日志并且不支持主备数据同步。

3 守护进程

守护进程（dmwatcher）是 DM 数据守护系统不可或缺的核心部件，是数据库实例和监视器之间信息流转的桥梁。数据库实例向本地守护进程发送信息，接收本地守护进程的消息和命令；监视器（dmmonitor）接收守护进程的消息，并向守护进程发送命令；数据库实例与监视器之间没有直接的消息交互；守护进程解析并执行监视器发起的各种命令（Switchover/Takeover/Open database 等），并在必要时通知数据库实例执行相应的操作。

3.1 主要功能

守护进程是管理数据守护系统的核心部件，监视器（dmmonitor）负责发起命令，守护进程负责解析、处理、转发命令。守护进程提供了数据库监控、故障检测、故障处理、故障恢复等各种功能。

3.1.1 监控数据库实例

守护进程负责监控数据库运行状态，必要时重启数据库服务。守护进程和实例链路建立成功后，数据库实例定时发送信息到守护进程，发送到守护进程的内容包括：实例进程 ID、实例名、数据库模式、数据库状态、FILE_SEQ、FILE_LSN、CUR_SEQ、CUR_LSN、MAL 链路状态、归档状态、公钥、MPP 控制文件等信息。

守护进程更新本地记录的实例信息后，同时记录该时间戳。当检测到实例进程 ID 已经不存在或者超过一段时间没有收到实例消息（INST_ERROR_TIME），则会认定实例故障。如果配置了自动重启，则会将实例重新拉起。

DMDSC 集群的自动重启由 `dmcss` 检测执行，单机的自动重启由守护进程检测执行。



注意：

守护进程采用超时机制判断实例是否故障，即当前时间和上次收到消息的时间差是否超过故障认定时间（`INST_ERROR_TIME`），因此不建议在数据守护系统运行过程中调整操作系统时间，避免导致这个差值很大，误判实例故障。

3.1.2 发送状态信息

守护进程将监控的数据库实例信息和守护进程自身的信息（包括守护类型、守护模式、守护状态、守护日志、监视器执行序列号、执行返回码等）捆绑在一起，定时发送给其他守护进程和所有监视器。

3.1.3 监控其他守护进程

接收并解析其他守护进程发送的消息，如果超过一段时间（`DW_ERROR_TIME`）没有收到远程守护进程消息，会将远程守护进程状态认定为 `ERROR` 状态。另外还会结合本地数据库信息和守护进程自身状态，切换数据库的运行模式和系统状态。

守护进程采用超时机制判断远程守护进程是否故障，即当前时间和上次收到消息的时间差是否超过故障认定时间（`DW_ERROR_TIME`），因此不建议在数



据守护系统运行过程中调整操作系统时间，避免导致这个差值很大，误判远程守护进程故障。

3.1.4 接收监视器消息

主备切换、备库接管等操作都是通过监视器命令进行，监视器将操作命令分解成多个步骤顺序执行。守护进程接收这些消息并通知实例进行相应操作，例如执行 `SQL` 语句修改实例模式、状态、`INI` 参数、设置归档状态等一系列动作，这些步骤依次执行完成后，即可完成主备库的切换或备库的接管等操作。

例如，主备切换操作，监视器首先通知待切换主备库的守护进程修改为 `Switchover` 状态，设置成功以后，其他监视器将不能再进行命令操作。守护进程收到监视器将实例

Mount 的命令，转发到本地实例执行，实例执行完成后返回执行结果。执行结果包含在实例向守护进程发送的消息中，守护进程根据消息中的执行码判断是否执行成功，并响应监视器。



监视器和守护进程之间也是采用超时机制判断对方是否故障，即当前时间和上次收到消息的时间差是否超过故障认定时间（守护进程配置的 **DW_ERROR_TIME**），因此不建议在数据守护系统运行过程中调整操作系统时间，避免导致这个差值很大，误判监视器故障。

3.1.5 主备库启动运行

数据守护系统刚启动时，所有实例处于 Mount 状态，守护进程处于 Startup 状态，启动时需要将实例转换到 Open 状态，守护进程也切换到 Open 状态，对外提供服务。

启动条件和流程详见数据守护使用说明章节中的数据守护启动流程 [6.2 数据守护的启动](#)。

3.1.6 备库故障处理

故障自动切换模式下，备库故障后，如果主备库之间的归档状态仍然有效，主库的守护进程会先切换为 Confirm 状态，等待确认监视器的确认消息，如果确认为符合故障处理条件，主库守护进程再切换至 Failover 状态，将故障备库的归档失效。

故障手动切换模式下，备库故障后，如果主备库之间的归档状态仍然有效，会直接切换到 Failover 状态，并将故障备库的归档失效，不需要备库故障确认。

备库故障后，备库的守护进程如果还处于活动状态且监控功能没有被关闭，则会切换到 Startup 状态下。

备库故障重启后，如果存在活动主库，主库守护进程根据备库实例的模式、状态、备库守护进程状态、守护进程控制文件状态、备库已经同步到的 Open 记录以及备库的恢复间隔等信息判断是否可以进行故障恢复，在满足故障恢复条件的情况下，主库守护进程启动 Recovery 流程，重新恢复主备库到一致状态。

如果一直没有观察到主库守护进程发起 Recovery 流程，可以借助监视器的 check recover 命令查找备库不满足条件的原因，并做对应的处理。

读写分离集群下，主库向即时备库发送归档失败后，会直接修改归档状态无效，并将数据库修改为 **Suspend** 状态。



说明：

如果主备库之间出现网络故障，并且在网络故障期间，主库没有修改操作触发归档日志发送，则主库会一直保持 **Open** 状态。

如果网络故障期间备库接管，网络恢复后，**dmwatcher** 会通知主库强制关闭。

3.1.7 备库异常处理

备库异常，指备库的数据库实例正常，但响应速度出现异常，这里的响应速度可能受主备库之间的网络影响，比如网络不稳定、网速大幅下降，也可能受备库自身的软硬件影响，比如操作系统原因或磁盘读写速度异常降低等异常情况导致响应主库的速度变慢，这种情况下会极大拖慢主库性能，影响主库正常处理对外的业务请求。

守护进程提供 **RLOG_SEND_THRESHOLD** 参数用于监控主备之间的日志发送速度，此参数应配置为大于 0 的值，否则守护进程不会打开监控功能。

主库守护进程在 **Open** 状态下对日志发送速度进行检测，一旦检测到异常，主库守护进程会切换到 **Standby check** 状态，并通知主库将异常备库的归档失效，暂停到此备库的数据同步，避免拖慢主库性能。

完整的备库异常处理流程如下（**Standby check** 状态处理）：

1. 收集所有的异常备库。
2. 将主库守护进程上记录的这些异常备库的最近一次恢复时间修改为当前时间。恢复间隔仍然为 **dmwatcher.ini** 中配置的 **INST_RECOVER_TIME** 值。这一步骤的目的是为了防止修改备库归档为 **Invalid** 无效状态后，主库立马启动 **recovery**，但是还未取到备库最新的 **LSN** 信息，导致 **recovery** 无法正确执行的情况发生。
3. 通知主库修改这些异常备库的归档为 **Invalid** 无效状态。
4. 守护进程切回 **Open** 状态。

下面情况会导致 **Standby check** 过程中断:



注意:

1. 在此状态下发现其他备库故障, 且符合 **failover** 条件, 则守护进程主动中断 **Standby check** 异常处理, 先做 **failover** 故障处理。
2. 主库是 **DSC** 集群, 在此期间启动故障处理或者故障重加入, 则守护进程会主动中断 **Standby check** 异常处理。

3.1.8 主库故障处理

故障自动切换模式下, 主库故障后, 确认监视器会捕获到故障信息, 自动选出可接管的备库, 并通知备库进行接管。备库接管由确认监视器自动触发, 无需用户干预。

故障手动切换模式下, 主库故障后, 需要人工干预, 通过监视器执行接管命令, 将可接管备库切换为主库。

故障自动切换模式下, 可以实时处理故障, 但对网络稳定性要求更高, 需要确保主备库之间, 主备库与守护进程、确认监视器之间的网络稳定可靠, 否则可能会误判主库故障, 备库自动接管后, 出现多个 **Open** 状态的主库, 引发脑裂。故障手动切换模式下, 备库不会自动接管, 出现节点故障或者网络故障时, 由用户根据各种故障情况, 进行人工干预。

主库故障重启后, 守护进程根据本地和远程库的 **Open** 记录、**LSN** 信息以及模式和状态信息来判定重启后的恢复策略, 可能继续作为主库加入守护系统, 也可能修改为 **Standby** 模式, 以备库身份重新加入守护系统, 如果出现分裂, 则需要用户干预。

3.1.9 故障恢复处理

故障恢复状态 (**Recovery**) 由守护进程自行判断是否切换, 和监视器无关。如果符合以下条件, 主库的守护进程可自动进入 **Recovery** 状态, 进行数据恢复:

1. 本地主库 [**Primary, Open**], 守护进程 **Open** 状态;
2. 远程备库 [**Standby, Open**], 归档状态 **Invalid**, 守护进程 **Open** 状态;
3. 远程备库 [**Standby, Open**] 的 **ASEQ/ALSN** 和 **SSEQ/SLSN** 相等, 没有待重做日志;
4. 根据 **Open** 记录等信息判断备库可加入;
5. 远程备库 [**Standby, Open**] 达到了设置的启动恢复的时间间隔。

对于前 4 点, 只是概括说明, 详细的条件比较多, 这里不再逐一点出, 如果某个备库

一直没有被恢复，可以借助监视器的 `check recover` 命令来查找备库无法进行恢复的原因。

对于第 5 点，可通过配置主库守护进程 `dmwatcher.ini` 的 `INST_RECOVER_TIME` 来控制，取值 (3s~ 86400 s)，该值对所有备库有效，可通过监视器的 `show arch send info` 命令查看当前设置的到指定备库的恢复时间间隔。

也可以使用监视器的 `set recover time` 命令来动态设置指定备库的恢复间隔，该命令只会修改命令中指定的备库的恢复间隔，并且只保存在主库的守护进程内存中，不会写入到 `dmwatcher.ini` 文件。

在主备库运行正常，不需要执行备库故障恢复的情况下，主库守护进程内存中对备库的恢复间隔值和 `dmwatcher.ini` 中配置的 `INST_RECOVER_TIME` 值是一致的，在某些场景下会被设置为不同的值，下面分别进行说明：

1. 主库守护进程主动设置恢复间隔为 3s

在以下场景中，主库的守护进程会重置内存中的 `INST_RECOVER_TIME` 为 3s，对满足前述前 4 个条件的备库在 3s 后会立即启动恢复流程：

- 1) 数据守护系统启动完成之后；
- 2) 守护系统运行过程中，主库手动重启或者守护进程自动启动 Open 之后；
- 3) 监视器执行 Switchover 主备切换/Takeover 备库接管/强制 Open 主库的操作之后。

2. 使用监视器命令动态设置恢复间隔

监视器提供有以下命令可动态修改 `INST_RECOVER_TIME` 值：

1) `set database [group_name.]db_name recover time time_value`

动态修改指定备库实例的恢复间隔，只修改对应主库守护进程内存中的 `INST_RECOVER_TIME` 值。

2) `set group [group_name] recover time time_value`

动态修改指定组或所有组中所有备库的恢复间隔，只修改对应主库守护进程内存中的 `INST_RECOVER_TIME` 值。

3) `set group [group_name] para_name para_value`

动态修改指定组或所有组中所有守护进程的配置参数值，`para_name` 允许指定为 `INST_RECOVER_TIME`，同时修改 `dmwatcher.ini` 文件和主库内存中的 `INST_RECOVER_TIME` 值。

以上 3 个命令都可以用来修改 `INST_RECOVER_TIME` 值，修改完成后，一旦主库对指定备库执行过恢复操作，不管恢复执行成功或失败，通过监视器动态修改的内存值都不再有效，主库守护进程在恢复完成后，会根据恢复结果重置内存中的恢复间隔值（对 `dmwatcher.ini` 中的值没有影响）。

3. 主库守护进程根据恢复结果设置恢复间隔

如果备库恢复成功，会重置此备库的恢复间隔为主库 `dmwatcher.ini` 中配置的值。

如果备库恢复失败，会根据失败 `code` 区分设置为不同的值，比如 1800s，一般是在主备库日志校验不连续的情况下设置，其他还可能设置为 3s、30s、300s 或者 `dmwatcher.ini` 中设置的值，这里不再详细说明，具体可通过服务器和守护进程的 `log` 日志查看详细的设置信息，也可以通过监视器的 `show arch send info` 命令查看相关 `code` 信息。

完整的故障恢复流程如下：

1. 通知备库丢弃 `KEEP_PKG`；
2. 通知主库发送归档日志，同步历史数据；
3. 通知主库切换为 `Suspend` 状态；
4. 再次通知主库发送归档日志；
5. 通知主库设置备库归档为 `Valid` 状态；
6. 通知主库切换为 `Open` 状态。

整个恢复过程中最耗时的是发送归档日志，当存在多个备库需要恢复时，为了提高恢复的效率，采用多备库并行发送归档的方式进行。守护进程每次搜集一个可恢复备库到恢复列表，按照上述故障恢复流程执行单个步骤，在等待发送归档日志的过程中，继续检测是否有备库可以恢复，如果有则加入恢复列表，也对其开始进行恢复流程处理；如果没有则检查恢复列表中是否有归档日志发送完成的备库，有则对其进行后续步骤处理，直至归档变成有效状态后从恢复列表中删除。对于恢复过程中出错的备库，也从恢复列表中删除。当恢复列表为空时，恢复流程执行结束，守护进程恢复为 `OPEN` 状态。

恢复过程中，守护进程会继续检测是否有恢复列表之外的备库需要恢复，如果有则可以动态加入恢复列表，实现动态并行恢复。

以下情况会导致 **Recovery** 过程中断：

1. **Recovery** 过程中收到监视器的命令。
2. 存在多个备库时，**Recovery** 过程中发现其他正常备库故障，且符合 **failover** 条件，则守护进程主动中断 **Recovery**，先做 **failover** 处理。
3. 存在多个备库时，**Recovery** 过程中发现到其他正常备库归档发送异常，则守护进程主动中断 **Recovery**，先做异常处理。
4. **Recovery** 过程中，当前正在被恢复的备库出现异常。
5. 正在执行 **Recovery** 的主库或备库是 **DMDSC** 集群，**Recovery** 过程中 **DMDSC** 集群启动故障处理或者故障重加入，也会中断当前的 **Recovery** 动作。



警告：常，则守护进程主动中断 **Recovery**，先做异常处理。

在守护进程打开备库异常监控的情况下，对于异常备库的恢复处理需要注意下面两点：

1. 如果主备库的 **LSN** 已经相等，但是根据统计出来的时间信息判断主库的归档发送时间或备库的日志重演时间仍然大于设置的阈值，则不会再进入 **Recovery** 状态，直到主库上有新的日志产生需要同步到备库，可以对统计的历史时间信息进行更新的情况下才会再进入 **Recovery** 状态尝试恢复。



注意：的历史时间信息进行更新的情况下才会再进入 **Recovery** 状态尝试恢复。

2. 在进入 **Recovery** 状态后，通知主库 **Suspend** 之前（主备库数据已经同步一致），会对主库的归档发送时间和备库的日志重演时间进行检查，在两者都小于或等于设置的阈值的情况下，才允许继续执行 **Suspend**，并恢复备库归档为有效状态，否则不允许再往下执行，本次 **Recovery** 执行失败。

3.2 守护类型

守护进程支持两种守护类型：

■ 本地守护

提供最基本的守护进程功能，监控本地数据库服务。如果实例使用 **Mount** 方式启动，守护进程会通知实例自动 **Open**，如果连续一段时间没有收到来自其监控数据库的消息，即认定数据库出现故障，根据配置（**INST_AUTO_RESTART**）确定是否使用配置的启动命令重启

数据库服务。异步备库也是采用这种方式配置。

■ 全局守护

实时主备、MPP 主备和读写分离集群系统中，需要将守护进程配置为全局守护类型；DMDSC 数据守护除了仅配置异步备库，也需要将守护进程配置为全局守护类型。守护进程根据数据库服务器配置的归档类型以及 MPP_INI 参数情况，自动识别具体的集群类型（实时主备、MPP 主备、读写分离集群或 DMDSC 数据守护），全局守护类型在本地守护类型的基础上，通过和远程守护进程的交互，增加了主备库切换、主备库故障检测、备库接管、数据库故障重加入等功能。



配置全局守护类型后，守护进程守护的数据库实例，必须配置实时或即时归档，

说明：否则 `dmwatcher` 会启动失败。

3.3 守护模式

守护进程支持两种故障切换模式：

■ 故障自动切换

主库发生故障时，确认监视器自动选择一个备库，切换为主库对外提供服务。故障自动切换模式，要求必须且只能配置一个确认监视器。具体的实现机制和使用注意事项可参考 4.3 自动接管小节。

■ 故障手动切换

主库发生故障时，由用户根据实际情况，通过监视器命令将备库切换为主库。在用户干预之前，备库可以继续提供只读服务和临时表的操作。

实时主备/MPP 主备/读写分离集群都可以配置为故障自动切换或故障手动切换模式，这两种模式下守护系统的启动流程、数据同步和故障处理机制存在一定的差异，主要差异参考表 3.1。

表 3.1 自动、手动切换比较

比较内容	故障自动切换	故障手动切换
硬件要求	>=3 台机器	>=2 台机器
主库故障需要人工干预	否	是
备库 KEEP_PKG	有（实时主备和 MPP 主备专用）	有（实时主备和 MPP 主备专用）

需要确认监视器	是	否
支持实时主备	是	是
支持 MPP 主备	是	是
支持读写分离集群	是	是
主库故障处理	备库自动接管	备库手动接管
备库故障处理	主库可能先进入 Confirm 状态，向确认监视器求证备库故障后，再进行 Failover 处理。 也可能直接 Failover 处理，具体参考 6.9 小节	主库直接 Failover 处理
主备库切换	支持	支持



一个数据守护集群，只能配置一个确认监视器。如果同时启动多个确认监视

说明：器，后启动的确认监视器将报错并自动退出。

3.4 守护状态

守护进程包括以下一些状态：

- **Startup** 守护进程启动状态，需要根据远程守护进程发送的状态信息，结合本地数据库的初始模式、状态和数据同步情况，确定本地数据库的启动模式和状态后，进入 Open 状态。

- **Open** 守护进程正常工作，监控数据库，并定时发送数据库的状态信息，接收其他守护进程发送的信息，接收监视器发送的用户请求。

- **Shutdown** 守护进程停止监控数据库状态，也不提供主备库切换功能。

- **Switchover** 主备库正常情况下，手动主备切换过程中设置为 Switchover 状态。

- **Failover** 远程备库故障后，本地主库执行故障处理时，守护进程设置为 Failover 状态。

- **Recovery** 故障恢复同步历史数据过程中设置为 Recovery 状态。

- **Confirm** 通过监视器确认远程主（备）库是否活动的过程中，守护进程设置为 Confirm 状态。

■ **Takeover** 主库确认故障后，备库手工接管或监视器通知自动接管过程中，守护进程设置为 Takeover 状态。

■ **Open force** 借助监视器命令强制 Open 主库或备库实例时，守护进程设置为 Open force 状态。

■ **Error** 超过一段时间 (DW_ERROR_TIME) 没有接收到远程守护进程消息，本地守护进程或监视器认定远程守护进程故障，修改远程守护进程为 Error 状态。

■ **Login check** 监视器执行登录命令时，守护进程所处的状态。

■ **Mppctl update** 修改主库 MPP 控制文件 (dmmpp.ctl) 时，守护进程所处的状态，只在 MPP 主备系统出现。

■ **Change arch** 监视器执行 set arch invalid 命令时守护进程所处的状态。

■ **Standby check** 主库守护进程监控到备库异常后，切换到此状态下通知主库修改此备库归档无效。

■ **Clear send info** 清理主库上的归档发送信息时，守护进程所处的状态。

■ **Clear rapply stat** 清理备库上的重演信息时，守护进程所处的状态。

■ **Unify ep** 统一 DMDSC 集群各节点实例状态，或者各实例状态已经一致时，守护进程在 Startup 或 Open 状态下通知实例执行相关操作，都进入 Unify_ep 状态执行。

■ **Css process** 监视器发起的对 DMDSC 集群的部分命令，比如启动、关闭、强杀 DMDSC 库，或者打开、关闭节点实例的自动拉起功能等命令，需要借助 dmcss 执行时，守护进程会切换到此状态下。

守护进程所有状态变换和它监控的数据库的状态变换都会生成相应的 LOG 信息，写入到 ../log 目录中以 'dm_dmwatcher_实例名_当前年月.log' 方式命名的日志文件中。用户可以通过查看日志文件，分析数据库和守护进程的运行状态、监控故障处理过程。

守护进程的状态转换如下图所示：

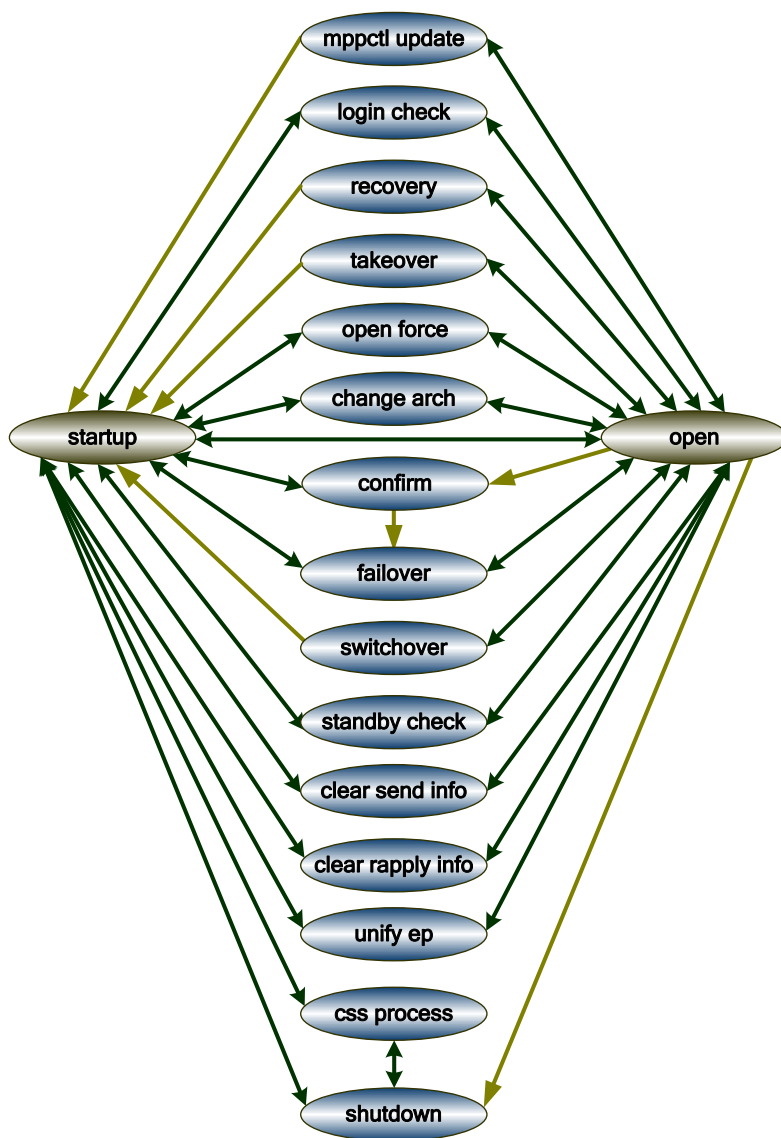


图 3.1 守护进程状态转换图

从图中可以看出，守护进程主要工作在 Startup 和 Open 状态，几乎任何状态都可以转到这两种状态，并且这两种状态之间也可以相互转换。

另外，当远程守护故障，任何状态都可转到 Error 状态。

3.5 控制文件

数据守护 V4.0 对守护进程控制文件（dmwatcher.ctl）进行了简化，仅用于记录本地数据库的分裂状态和分裂描述信息。守护进程在检测到本地库分裂时，自动创建 dmwatcher.ctl 文件，保存在本地库的 SYSTEM_PATH 路径下，并且文件中记录的状态一定是 Split 分裂状态。如果 dmwatcher 加载到 dmwatcher.ctl 文件，则认为对应的

库一定是分裂状态。如果需要对分裂库进行重建，则需要手动将 `dmwatcher.ctl` 文件删除，否则守护进程仍然会认定本地库为分裂库。

守护进程控制文件仅包含版本号、状态及分裂描述信息这三项内容。

状态字段包含以下两种：

- 有效 (VALID) 正常运行时状态。
- 分裂 (SPLIT) 数据和有效主库的数据不一致时设置。

3.6 可加入（分裂）判断规则

守护进程首先对本地和远程库的 Open 记录 (SYSOPENHISTORY) 进行比较，再结合数据库的模式、状态以及守护进程认定的 OK/ERROR 状态判断可加入或者分裂，这里仅对守护进程的判断规则进行大概说明，如果发现守护系统中有分裂库，可以从服务器和守护进程的 log 日志中找到具体的分裂原因。

Open 记录的比较结果分为四种：

1. 本地库和远程库相等

如果其中一个库是 Primary&Open 或者 Primary&Suspend 状态，则直接将其认定为主库，另一个库认定为备库。

否则需要进一步比较两个库的 APPLY 信息，比较结果可能会选出有效的主库和备库，也可能出现分裂，需要用户干预。

2. 远程库包含在本地库中

使用本地库包含关系之后的第一条 Open 记录项中的 APPLY 信息和远程库的 APPLY 信息比较大小，远程库小于或等于的情况下，远程库的守护进程再继续根据模式、状态信息确定下一步动作，最终可以将其作为备库重加入守护系统。

3. 本地库包含在远程库中

使用远程库包含关系之后的第一条 Open 记录项中的 APPLY 信息和本地库的 APPLY 信息比较大小，本地库小于或等于的情况下，本地库的守护进程再继续根据模式、状态信息确定下一步动作，最终可以将其作为备库重加入守护系统。

4. 本地库和远程库不相等，也没有包含关系

如果其中一个库处于 Primary&Open 状态，另一个库是备库，或者是 Mount 状态的主库，则另一个库的守护进程会将自己设置为 Split 分裂状态，创建 dmwatcher.ctl 文件，并通知本地库强制关闭。如果守护进程无法选出有效主库，则切换为 Startup 状态，并等待用户干预。

需要注意的是，SYSOPENHISTORY 表最多只维护 128 条 Open 记录，达到 128 条以后会从第一条记录开始自动覆盖，并依次往后循环使用。在主备库的数据差异很大的情况下，备库的最后一条 Open 记录有可能在主库上已经被覆盖掉，备库无法根据 Open 记录确定自己的包含关系，也无法重加入数据守护系统。这种情况下只能通过人工干预，使用备份还原的方式将备库重新恢复到和主库数据一致的状态并重新加入数据守护系统。

3.7 守护进程命令

守护进程（dmwatcher）既能以控制台方式启动，也可以配置为服务方式启动。可以在守护进程的控制台上输入命令，关闭守护进程，显示守护进程组的状态信息等，具体命令参考下表：

表 3.2 守护进程命令

命令	含义
help	显示帮助信息
exit	退出守护进程
status	显示守护进程运行状态信息
show	显示所有守护进程组中的本地库信息
show group group_name	显示指定守护进程组中的本地库信息
show version	显示守护进程自身版本信息
show monitor config	帮助监视器配置 ini 文件的信息
show link	显示守护进程上的 tcp 连接信息

下面对守护进程各命令含义进行详细介绍：

1. help

打印守护进程使用帮助信息。

2. exit

退出守护进程。

3. status

打印守护进程状态信息，各字段含义说明如下：

GROUP_NAME：守护进程组名。

DW_STATUS：守护进程状态。

DW_SUB_STATUS：守护进程子状态。

4. show

显示所有守护进程组中的本地库信息，不显示远程库信息。

各字段含义说明如下：

1) 组全局字段信息

GROUP_NAME：守护进程组名。

TYPE：配置的守护类型。

MODE：配置的切换模式，AUTO 表示故障自动切换模式，MANUAL 表示故障手动切换模式。

OGUID：守护进程组配置的唯一 OGUID 值。

MPP_FLAG：当前是否为 MPP 主备环境，值为 TRUE 或 FALSE。

AUTO_RESTART：守护进程是否配置有自动拉起，值为 TRUE 或 FALSE。

DW_STATUS：守护进程状态。

DW_SUB_STATUS：守护进程子状态。

DW_CTL_STATUS：守护进程控制文件状态。

2) 各节点实例字段信息

INST_OK：守护进程认定的节点实例状态，Ok 或 Error。

NAME：节点实例名称。

SVR_MODE：节点实例模式，包括 Normal/Primary/Standby 这三种模式。

SYS_STATUS：节点实例状态，包括 Startup/After Redo/Mount/Open/Suspend/Shutdown 这几种状态。

RTYPE：节点实例配置的归档类型，只统计 REALTIME/TIMELY 这两种归档类型，如

果这两种归档都没有配置，则显示为 NONE。

FSEQ: 节点实例已经写入联机日志的 RLOG_PKG 包序号。

FLSN: 节点实例的文件 LSN，指已经写入联机日志文件的最大 LSN 值。

CSEQ: 节点实例的系统当前 PKG_SEQNO，指当前数据库最新产生的 RLOG_PKG 包的序号。

CLSN: 节点实例的系统当前 LSN，指当前数据库最新产生的 LSN 值。

DW_STAT_FLAG: 节点当前的执行标记。

如果是备库模式，show 命令还会显示备库控制节点当前的重演信息，重演信息的行数和产生日志的主库节点个数一致，可以查看备库对主库不同节点日志的重演情况。

REDOS_PARALLEL_NUM: 备库并行重演线程数。

DSC_SEQNO: 主库节点序号，如果主库是单节点，则序号为 0。

SSEQ: 备库可重演到的最大日志包序号。

SLSN: 备库可重演到的最大 LSN，对应日志包序号为 STDBY_PKG_SEQNO。

KSEQ: 非自动切换模式下，备库保持不重演的日志包序号。

KLSN: 非自动切换模式下，备库保持不重演最大 LSN，对应的日志包序号为 KSEQ。

RSEQ: 备库针对主库此节点已经重演到的日志包序号。

RLSN: 备库针对主库此节点已经重演到的 LSN 值，对应的日志包序号为 RSEQ。

N_TSK: 备库针对主库此节点的待重演任务个数。

TSK_MEM_USE: 备库当前针对主库此节点的日志重演已经占用的内存大小（单位：字节）。

REDO_LSN_ARR: 备库各路并行重演线程已重演到的 LSN 值，数组中每个元素对应一路重演 LSN。

5. show group group_name

显示所有守护进程组中的本地库信息，不显示远程库信息。各字段含义同 show 命令。

6. show version

显示守护进程自身版本号。

守护进程版本信息格式为“DMWATCHER[数据守护版本号] 全局版本号”，例如：

“DMWATCHER[4.0] V8”。

同一套数据守护系统中，服务器、守护进程和监视器要求中括号内的数据守护版本号必须一致，否则不允许建立 TCP 连接，各自的控制台工具上和 log 日志中都会记录版本不匹配报错信息。

7. show monitor config

此命令为辅助用户配置 dmmonitor.ini 使用。

守护进程根据自己的组名、oguid、ip/port 等配置信息，生成一份 dmmonitor.ini 配置文件供用户参考使用。

此命令只会显示包含在守护进程守护的本地实例归档配置文件中的实例对应的守护进程信息，比如本地实例配置的实时归档、即时归档、异步归档的目标实例所在守护进程信息，对于未配置的目标实例守护进程信息则不会显示，比如其他实例上配置的异步备库守护进程信息不能显示。因此对监视器来说，单个守护进程上的显示结果配置信息可能是不完整的。只做配置参考使用，可以借助多个守护进程的显示结果，综合起来进行配置。

8. show link

此命令为辅助用户查看本地守护进程和本地各节点实例、远程守护进程、dmcss、监视器的 TCP 连接状态使用。

此命令只显示当前连接正常的链路信息，如果连接已经断开，则不会被显示出来。

各字段含义说明如下：

FROM_FLAG: 和本地守护进程建立 tcp 连接的对方程序名称，dmserver、dmcss、dmwatcher 或者 dmmonitor。

FROM_INAME: 和本地守护进程建立 TCP 连接的对方实例名或者 css 名称。

HANDLE: TCP 连接句柄。

MID: 只对监视器有效，监视器唯一标识 ID。

N_FIX: TCP 连接句柄当前被使用个数。

MON_ADDR: 连接对方 IP 地址。

MON_CONFIRM: 只对监视器有效，标识是否为确认监视器。

CONN_TIME: TCP 连接建立时间。

4 监视器

监视器（dmmonitor）是基于监视器接口（详见 10.2 监视器接口）实现的一个命令行工具，是 DM 数据守护系统的重要组成部分。

通过监视器，可以监控数据守护系统的运行情况，获取主备库状态、守护进程状态以及主备库数据同步情况等信息。同时，监视器（dmmonitor）还提供了一系列命令来管理数据守护系统。

监视器的基本作用如下：

■ 监控数据守护系统

接收守护进程发送的消息，显示主、备数据库状态变化，以及故障切换过程中，数据库模式、状态变化的完整过程。

■ 管理数据守护系统

用户可以在监视器上输入命令，启动、停止守护进程的监控功能，执行主备库切换、备库故障接管等操作。

■ 确认状态信息

用于故障自动切换的数据守护系统中，主、备库进行故障处理之前，需要通过监视器进行信息确认，确保对应的备库或者主库是真的产生异常了，避免主备库之间网络故障引发脑裂。

■ 发起故障自动接管命令

用于故障自动切换的数据守护系统中，主库发生故障时，挑选符合接管条件的备库，并通知备库执行接管操作。

对于实时主备和读写分离集群，监视器只允许配置一个守护进程组；MPP 主备允许配置多组，并且要求这些组的主库必须是同一套 MPP 系统；



注意：

对于本地守护类型，允许和实时主备/读写分离集群/MPP 主备配置到同一组作为异步备库，如果不作为某个库的异步备库，只是普通的单节点配置为本地守护类型，则需要单独成组，并且监视器也不允许配置多组。

4.1 监视器类型

监视器支持两种运行模式：监控模式和确认模式。监视器运行模式由配置文件（dmmonitor.ini）的 MON_DW_CONFIRM 参数来确定。MON_DW_CONFIRM 参数的默认值是 0，表示监控模式；MON_DW_CONFIRM 参数值为 1 时，表示监视器运行在确认模式下。为了区分监视器的两种模式，我们将运行在确认模式下的监视器称为确认监视器，将运行在监控模式下的监视器称为普通监视器。

1. 监控模式

一个数据守护系统中，最多允许同时启动 10 个监视器，所有监视器都可以接收守护进程消息，获取守护系统状态，也可以执行各种监控命令。所有监视器都可以发起 Switchover 等命令，但守护进程一次只能接收一个监视器的命令，在一个监视器命令执行完成之前，守护进程收到其他监视器发起的请求，会直接报错返回。

2. 确认模式

确认模式和监控模式的区别在于，除了具备监控模式监视器所有功能之外，确认监视器还具有状态确认和自动接管两个功能。故障自动切换模式的数据守护系统，必须部署一个确认监视器，否则在出现数据库故障时，会导致数据库服务中断。在使用主备多实例监视器的情况下，确认监视器可以配置多个，但是同时进行工作的确认监视器的 mid 必须配置为相同值，代表这些确认监视器属于同一个数据守护系统，同一个数据守护系统中的确认监视器在保证大多数工作正常的情况下发挥的功能与一个确认监视器相同。除使用主备多实例监视器的情况外，一个数据守护系统只能配置一个确认监视器。

4.2 状态确认

故障自动切换模式数据守护系统中，主库守护进程监测到备库故障时，需要向确认监视器求证，确认备库是真的故障了，再启动故障处理流程将归档失效，避免引发脑裂。比如，主库网络故障，主库直接将归档失效继续以 Primary 模式提供服务；同时，确认监视器认为主库故障，将备库切换为 Primary 模式，守护进程组中同时出现多个主库，引发脑裂。

状态确认只对故障自动切换数据守护系统有效，主库守护进程在满足一定条件时，会切换到 Confirm 状态，向确认监视器进行求证，具体的条件要求请参考备库故障处理小节。

主库守护进程切换到 Confirm 之后，根据不同的场景决定是否切换为 Failover 状态

并启动故障处理流程，具体的判断条件请参考备库故障处理小节，这里列举出几种常见的场景处理（注意前提是主库的守护进程已经处于 Confirm 状态）：

1. 主库网络故障，主库到备库、主库到确认监视器的连接异常。这种情况下主库守护进程一直保持 Confirm 状态，不会启动故障处理流程。
2. 备库故障或者备库网络故障，主库守护进程会切换为 Failover 状态，启动故障处理流程。
3. 主备库之间网络故障，但与确认监视器之间的网络正常，确认监视器确认主库满足 Failover 执行条件，主库守护进程会切换为 Failover 状态，启动故障处理流程，否则主库守护进程一直保持在 Confirm 状态。

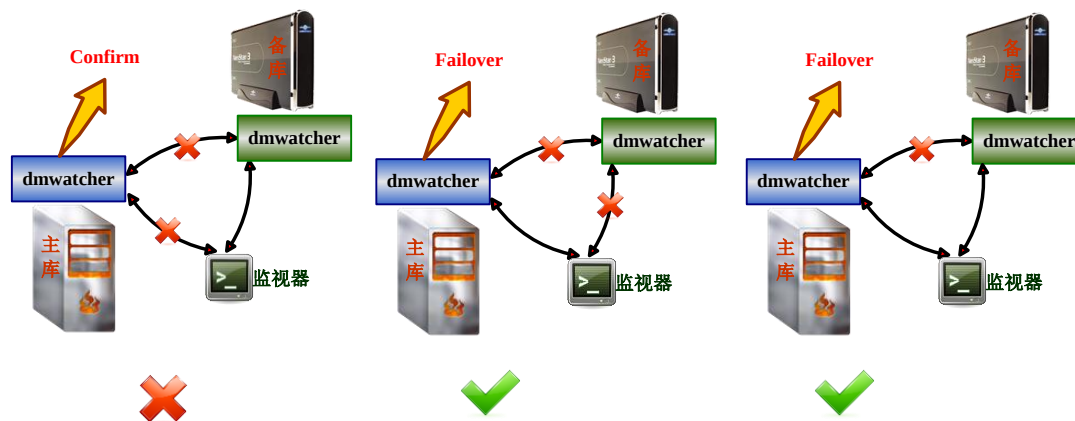


图 4.1 状态确认

4.3 自动接管

故障自动切换模式下，确认监视器检测到主库故障后，根据收到的主备库 LSN、归档状态、MAL 链路状态等信息，确定一个接管备库，并将其切换为主库。确认监视器启动自动接管流程的主要场景有三种，任何一种都会导致备库自动接管。场景如下：

1. 主库数据库实例异常终止，主库守护进程正常。
2. 主库硬件故障、或者数据库实例和守护进程同时故障。
3. 主库网络故障，主备库之间、主库与监视器之间连接异常。

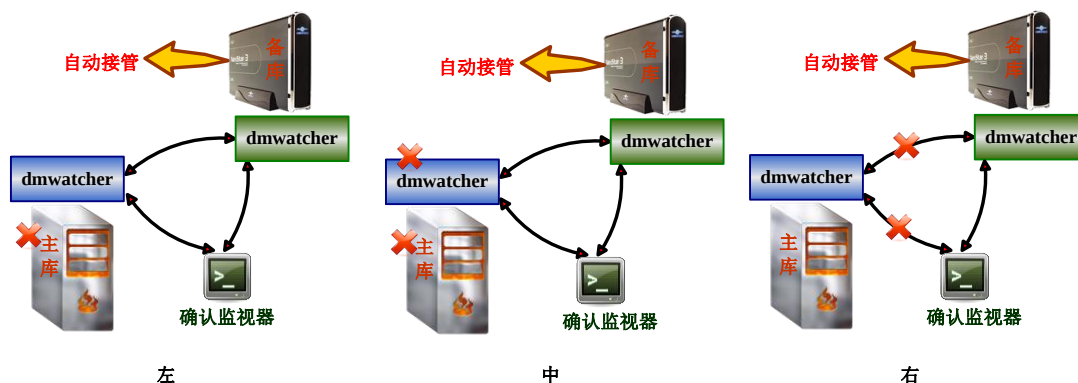


图 4.2 自动接管

若想实现备库自动接管，主库、归档状态、备库都必须符合一定条件才行。条件如下：

■ 主库

1. 主库是 Primary 模式、Open 状态时，产生故障。（如图 4.2 左、中）
2. 主库守护进程故障，故障前是 Open/Recovery 状态。（如图 4.2 中）
3. 故障主库与接管备库和确认监视器之间的 MAL 链路断开。（如图 4.2 右）

■ 归档状态

故障主库到接管备库的归档状态为 Valid。

■ 备库

接管备库是 Standby 模式、Open 状态。

- 如果需要在主库正常退出时备库进行自动接管，需要配置 **MON_TAKEOVER_SHUTDOWN** 参数。
- 如果主库故障前正在执行 **Switchover/Takeover** 等命令，则备库不会自动接管，需要人工干预。
- 确认监视器要求一开始就启动，保证出现故障情况时，确认监视器已经收到了故障主库或备库的历史消息，否则会因为条件不足无法自动处理，需要通过命令方式人工干预。
- 当由于一些原因导致自动接管失败时，系统可能不会再次尝试自动接管，需要人工干预。
- 确认监视器不要和主库部署在一台机器上，避免主库网络故障时，备库无法自动切换为主库。
- 确认监视器不要和备库部署在一台机器上，避免主备库之间网络异常时，确认监视器误认为主库故障，将备库切换为主库，引发双主问题。



注意：

4.4 监视器命令

数据守护 V4.0 对之前版本监视器的部分命令做了调整，新增、删除以及变更了一批监视器命令。

数据守护 V4.0 中监视器提供以下命令来管理守护系统：

表 4.1 监视器命令

命令	含义
系统全局命令	
help	显示帮助信息
exit	退出监视器
show version	显示监视器自身版本信息
show global info	显示所有组的全局信息
show database [group_name.]db_name	显示指定库的详细信息
show [group_name]	显示指定组的实例信息，如果未指定组名，则显示所有组信息
show i[nterval] n	每隔n秒自动显示所有组的实例信息
q	取消自动显示
list [[group_name.]db_name]	列出指定组的库对应的守护进程配置信息，如果都未指定，则列出所有守护进程配置信息
show open info [group_name.]db_name	显示指定库的Open历史信息
show arch send info [group_name.]db_name	查看源库到指定组的指定库的归档同步信息 (包含恢复间隔信息)
show apply stat [group_name.]db_name	查看指定组的指定库的日志重演信息
show monitor [group_name[.]] [db_name]	列出连接到指定守护进程的所有监视器信息
tip	查看系统当前运行状态
login	登录监视器
logout	退出登录
get takeover time	获取备库开始自动接管需要延迟等待的时间
show state	显示当前监视器所在监视器配置组的所有监视器的

	状态信息
以组为单位执行的命令	
startup dmwatcher [group_name]	启动指定组的守护进程监控功能
stop dmwatcher [group_name]	关闭指定组的守护进程监控功能
startup group [group_name]	启动指定组的库
stop group [group_name]	关闭指定组的库
kill group [group_name]	强制杀掉指定组中的活动库
choose switchover [group_name]	选择可切换为Primary库的备库列表
choose takeover [group_name]	选择可接管故障Primary库的备库列表
choose takeover force [group_name]	选择可强制接管故障Primary库的备库列表
set group [group_name] auto restart on	打开指定组中所有节点的自动拉起功能
set group [group_name] auto restart off	关闭指定组中所有节点的自动拉起功能
set group [group_name] para_name para_value	修改指定组的所有守护进程的指定配置参数(同时修改ini文件和内存值)，如果未指定组名，则通知所有组执行。 para_name指定参数名称，para_value指定参数值 支持修改参数： DW_ERROR_TIME/INST_RECOVER_TIME/ INST_ERROR_TIME/INST_AUTO_RESTART/ INST_SERVICE_IP_CHECK/RLOG_SEND_THRES HOLD/RLOG_APPLY_THRESHOLD
set group [group_name] recover time time_value	修改指定组中所有备库的恢复间隔为time_value指定的整数值(time_value取值：3~86400，单位为秒)(只修改守护进程内存值)，如果未指定组名，则通知所有组执行
set group [group_name] arch invalid	修改指定组中所有备库的归档为无效状态，如果未指定组名，则通知所有组执行
clear group [group_name] arch send info	清理指定组中源库到所有备库(包括异步备库)的最

	近N次归档发送信息（通知源库执行），没有指定组名则通知所有组执行，其中N值取源库dm.ini中配置的RLOG_SEND_APPLY_MON值和实际归档发送次数中的较小值。
clear group [group_name] apply stat	清理指定组中所有备库的最近N次重演信息（通知组中所有备库执行），没有指定组名则通知所有组执行，其中N值取备库dm.ini中配置的RLOG_SEND_APPLY_MON值和实际重演次数中的较小值。
以库为单位执行的命令	
check recover [group_name.]db_name	检查指定组的指定库是否满足自动恢复条件
check open [group_name.]db_name	检查指定组的指定库是否满足自动Open条件
open database [group_name.] db_name	强制Open指定组的指定库
switchover [group_name[.]] [db_name]	切换指定组的指定库为Primary库
takeover [group_name[.]] [db_name]	使用指定组的指定库接管故障Primary库
takeover force [group_name[.]] [db_name]	使用指定组的指定库强制接管故障Primary库
set database [group_name.]db_name recover time time_value	修改指定组的指定库的恢复间隔为time_value指定的整数值（time_value取值：3~86400，单位为秒）（只修改守护进程内存值）
set database [group_name.]db_name arch invalid	修改指定组的指定库的归档为无效状态
detach database [group_name.]db_name	将指定的备库分离出守护进程组
attach database [group_name.]db_name	将分离出去的备库重新加回到守护进程组
startup dmwatcher database [group_name.]db_name	打开指定库的守护进程监控功能
stop dmwatcher database[group_name.]db_name	关闭指定库的守护进程监控功能
startup database [group_name.]db_name	启动指定组的指定库

stop database [group_name.]db_name	关闭指定组的指定库
kill database [group_name.]db_name	强杀指定组的指定库
clear database [group_name.]db_name arch send info	清理指定组中主库到指定备库实例的最近N次归档发送信息（通知主库执行），其中N值取主库dm.ini中配置的RLOG_SEND_APPLY_MON值和实际归档发送次数中的较小值。
clear database [group_name.]db_name apply stat	清理指定备库的最近N次重演信息（通知备库执行），其中N值取备库dm.ini中配置的RLOG_SEND_APPLY_MON值和实际重演次数中的较小值。
set database [group_name.]db_name until time time_value	设置指定组的指定异步备库重演到的时间点为time_value指定的时间点
cancel database [group_name.]db_name until time	取消指定组的指定异步备库重演到的时间点
只允许在MPP主备环境下使用的命令	
show mpp	显示MPP节点信息
startup dmwatcher all	启动所有组的守护进程监控功能
stop dmwatcher all	关闭所有组的守护进程监控功能
startup group all	启动所有组的库
stop group all	关闭所有组的库
kill group all	强制杀掉所有组中的活动库
check mppctl	检查MPP控制文件是否处于一致状态
recover mppctl	恢复MPP控制文件到一致状态

下面对监视器命令进行详细的介绍：

命令中参数 `GROUP_NAME` 和 `DB_NAME` 的使用说明：

对于读写分离集群和实时主备，因为监视器只允许配置一组，所以命令中的

组名 `[group_name]` 都可以不指定；对于 MPP 主备，因为有多组，所以需要



注意：

但是 `set group` 相关的命令不受此限制，如果不指定 `[group_name]`，则所有组全部执行。

`[db_name]` 在有多个库时需要指定库名。

组名和库名均不唯一的情况下，需要全部指定。组名和库名用符号“.”分隔。

1. Help

显示帮助信息。

2. exit

退出监视器。

3. show version

显示监视器自身版本信息。

监视器版本格式为“`DMMONITOR[数据守护版本号] 全局版本号`”，例如：

```
"DMMONITOR[4.0] V8"
```

同一套数据守护系统中，服务器、守护进程和监视器要求中括号内的数据守护版本号必须一致，否则不能建立连接，各自的控制台工具上会打印版本不匹配信息。

4. show global info

数据守护系统中如果包含有多个 DMDSC 集群，通过监视器查看所有库的详细信息时，信息会过于庞杂，需要再逐个分割查找每个库的重点信息，比如状态是否 OK、是否发生分裂等信息。为了帮助用户更快捷地掌握整个守护系统的运行状态，此命令只打印出每个库的全局状态信息，不再详细打印每个节点实例信息，用户可借助 `show database` 命令查看指定库的详细状态信息。

此命令的显示结果中各字段含义说明如下：

1) 组全局字段信息

GROUP：守护进程组名。

OGUID：守护进程组配置的唯一 OGUID 值。

MON_CONFIRM: 监视器是否配置为确认模式, 值为 TRUE 或 FALSE。

MODE: 当前配置的切换模式, AUTO 表示故障自动切换模式, MANUAL 表示故障手动切换模式。

MPP_FLAG: 当前是否为 MPP 主备环境, 值为 TRUE 或 FALSE。

2) 库全局字段信息

守护进程和监视器是以库为单位进行管理的, 对有多个节点的 DMDSC 集群, 看作是一个完整的库, 以下信息属于库的全局信息, 部分守护进程相关的信息直接取自控制守护进程。

DW_IP: 守护进程的 IP 地址, 取自控制守护进程。

MAL_DW_PORT: 守护进程的监听端口, 取自控制守护进程。

WTIME: 守护进程本地的当前时间, 取自控制守护进程。

WTYPE: 守护进程配置的守护类型, 包括 LOCAL/GLOBAL 两种类型, 取自控制守护进程。

WCTLSTAT: 守护进程控制文件状态, 包括 Valid/Split/Invalid 三种状态, 对于配置为本地守护类型的实例, 由于不需要有守护进程控制文件, 该字段值为 NULL。

WSTATUS: 守护进程状态, 取自控制守护进程。

INAME: 库名称, 取自控制节点实例名。

INST_OK: 库的运行状态, 包括 Ok/Error 两种状态。

N_EP: 库的节点个数。

N_OK: 此字段对 DMDSC 集群内的节点有效, 表示 DMDSC 集群内 OK 节点个数。

ISTATUS: 数据库状态, 包括 Startup/After Redo/Mount/Open/Suspend/Shutdown 这几种状态, 由控制守护进程综合当前所有节点实例的状态得出。

IMODE: 数据库模式, 包括 Unknown/Normal/Primary/Standby 这四种模式, 由控制守护进程综合当前所有节点实例的模式得出。

DSC_STATUS: DSC 集群的运行状态, 该字段仅对 DMDSC 集群有效, 包括 DSC_STARTUP/DSC_OPEN/DSC_CRASH_RECV/DSC_ERR_EP_ADD 四种状态。

RTYPE: 数据库配置的归档类型, 只统计 REALTIME/TIMELY 这两种归档类型, 如果这两种归档都没有配置, 则显示为 NONE。

RSTAT: 此字段对备库有效, 表示主库到备库的归档状态, 可能为 Valid/Invalid/Unknown, 对于本地守护类型的备库, 此字段为 NULL, 对于主库本身, 此字段值为 Valid。该字段值要从备库对应的主库实例上取, 如果当前没有活动主库或者

备库无法确认对应的主库，则该字段显示为 Unknown。

5. show database [group_name.]db_name

此命令用于详细显示指定库的相关状态信息。

各字段含义说明如下：

1) 库全局字段信息

和 show global info 命令中的库全局字段含义相同。

2) 各节点实例字段信息

INST_PORT：节点实例的监听端口。

INST_OK：控制守护进程认定的节点实例状态，OK 或 ERROR。

INAME：节点实例名称。

ISTATUS：节点实例状态，包括 Startup/After Redo/Mount/Open/Suspend/Shutdown 这几种状态。

IMODE：节点实例模式，包括 Normal/Primary/Standby 这三种模式。

DSC_STATUS：此字段对单节点可忽略，只对 DMDSC 集群内的节点有意义，表示节点当前的 DMDSC 集群状态。

DSC_SEQNO：此字段对单节点可忽略，只对 DMDSC 集群内的节点有意义，表示节点在 DMDSC 集群内的序号。

DSC_CONTROL_NODE：此字段对单节点可忽略，只对 DMDSC 集群内的节点有意义，表示 DMDSC 集群内控制节点的序号。

RTYPE：节点实例配置的归档类型，只统计 REALTIME/TIMELY 这两种归档类型，如果这两种归档都没有配置，则显示为 NONE。

RSTAT：此字段对备库控制节点有效，表示主库到备库控制节点的归档状态，可能为 Valid/Invalid/Unknown，对于本地守护类型的备库，此字段为 NULL，对于主库本身，此字段值为 Valid。该字段值要从备库对应的主库实例上取，如果当前没有活动主库或者备库无法确认对应的主库，则该字段显示为 Unknown。

FSEQ：节点实例已经写入联机日志的 RLOG_PKG 包序号。

FLSN：节点实例的文件 LSN，指已经写入联机日志文件的最大 LSN 值。

CSEQ：节点实例的系统当前 PKG_SEQNO，指当前数据库最新产生的 RLOG_PKG 包的序号。

CLSN: 节点实例的系统当前 LSN, 指当前数据库最新产生的 LSN 值。

DW_STAT_FLAG: 节点当前的执行标记。

如果是备库模式, show 命令还会显示备库控制节点当前的重演信息, 重演信息的行数和产生日志的主库节点个数一致, 可以查看备库对主库不同节点日志的重演情况。

REDOS_PARALLEL_NUM: 备库并行重演线程数。

ARCH_SEND_UNTIL_TIME: 备库针对主库此节点重演到的截止时间。

APPLY_UNTIL_TIME: 此字段仅对异步备库有效, 标识异步备库是否重演到设置的 UNTIL_TIME。当未指定 UNTIL_TIME 或非异步备库时, 为 FALSE。

DSC_SEQNO: 主库节点序号。

SSEQ: 备库可重演到的最大日志包序号。

SLSN: 备库可重演到的最大 LSN, 对应日志包序号为 STDBY_PKG_SEQNO。

KSEQ: 非自动切换模式下, 备库保持不重演的日志包序号。

KLSN: 非自动切换模式下, 备库保持不重演最大 LSN, 对应的日志包序号为 KSEQ。

RSEQ: 备库针对主库此节点已经重演到的日志包序号。

RLSN: 备库针对主库此节点已经重演到的 LSN 值, 对应的日志包序号为 RSEQ。

N_TSK: 备库针对主库此节点的待重演任务个数。

TSK_MEM_USE: 备库当前针对主库此节点的日志重演已经占用的内存大小 (单位: 字节)。

REDO_LSN_ARR: 备库各路并行重演线程已重演到的 LSN 值, 数组中每个元素对应一路重演 LSN。

6. show [group_name]

如果指定 group_name, 则显示指定组中所有库的详细状态信息, 如果没有指定, 则显示所有组中所有库的详细状态信息。

show 命令由下面几种信息组成:

- 1) 组全局字段信息
- 2) 库全局字段信息
- 3) 各节点实例字段信息

也就是 show global info 和 show database 命令的整合, 各字段含义可参考这

两个命令中的含义说明。

7. show i[nterval] n

在 dmmonitor 工具界面上定时每隔 n 秒自动显示所有组中所有库的详细状态信息，相当于每隔 n 秒自动执行一次 show 命令，n 取值范围为 5~3600 (s)，相关字段含义可参考 show [group_name] 命令说明。

如果在工具界面上手动执行其他命令，则会自动取消定时显示功能。

8. q

取消 show i[nterval] n 命令设置的自动显示功能。

9. list [[group_name.]db_name]

此命令列出指定组中指定实例的守护进程配置信息，只有一组时允许不指定 group_name，如果组名和库名都没有指定，则显示所有守护进程的配置信息。

list 命令显示的字段和 dmwatcher.ini 的配置内容完全一致，可参考 dmwatcher.ini 配置表格的各个字段说明。

如果守护进程守护的是 DMDSC 集群，list 命令会额外显示 dmdcr.ini 中配置的 CSS_ASM_RESTART_INTERVAL 和 CSS_DB_RESTART_INTERVAL 参数值，方便用户查看 dmcss 的自动拉起配置情况。

10. show open info [group_name.]db_name

此命令显示指定库的 INST_NAME、N_ITEM、CTL_STATUS 以及 DESC。只有一组时，此命令允许不指定 group_name。

INST_NAME: 守护的数据库名称。

N_ITEM: Open 历史信息记录数。Open 历史信息记录最多只记录 128 项，写满 128 项后从第一项记录覆盖起，循环写入。Open 历史信息只有 Primary 和 Normal 模式的库 Open 时才主动写入。

CTL_STATUS: 控制文件状态，包括 Valid/Split/Invalid 三种状态。

DESC: 本地分裂库的描述信息，如果是有效状态，描述信息为空。

Open 历史信息包括 TGUID、ROWID、OPEN_TIME、SYS_MODE、PRI_INST_NAME、CUR_INST_NAME、PRI_DB_MAGIC、CUR_DB_MAGIC、N_EP、PKG_SEQNO_ARR 以及

APPLY_LSN_ARR, 具体含义如下:

TGUID: Open 记录唯一标识。

ROWID: 对应 SYSOPENHISTORY 中记录的 rowid。

OPEN_TIME: Open 的时间。

SYS_MODE: 系统 Open 时所处的模式。

PRI_INST_NAME: 对应主库的实例名。

CUR_INST_NAME: 当前主库的实例名。

N_EP: 数据库节点实例个数。非 DSC 环境, 实例个数为 1。

PKG_SEQNO_ARR: 已经写入联机日志的日志包序号, 数组长度为 N_EP。

APPLY_LSN_ARR: 已经重演的日志包序号, 数组长度为 N_EP。

11. show monitor [group_name[.]] [db_name]

列出连接到指定库控制守护进程的所有监视器信息。

对于 group_name, 在配置有多组时需要指定, 否则可以不指定。对于 db_name, 如果指定, 则显示指定实例守护进程上的连接信息, 如果没有指定, 则根据 dmmonitor.ini 对应组中的配置顺序, 显示第一个活动的守护进程上的监视器连接信息。如果 group_name 和 db_name 同时指定, 则需要用“.”符号分隔。

同一个守护系统中允许最多 10 个监视器同时连接, 可通过此命令获取指定守护进程上监视器连接信息, 包括连接时间、确认监视器配置、监视器 ID、监视器 IP 和版本信息。显示的第一行结果为本地监视器的连接信息。

命令显示结果的字段说明如下:

DW_CONN_TIME: 守护进程和监视器建立连接的时间。

MON_CONFIRM: 监视器是否配置为确认模式。

MID: 唯一标识监视器的 ID 值。监视器从启动到退出, MID 值保持不变。每次执行操作, 监视器会把 MID 设置到所有的守护进程上, 操作执行完, 会清理掉所有守护进程上的 MID 值。守护进程每隔 5s 会检查一次, 如果监视器已经退出, 会自行清理掉自己保存的 MID 值。

MON_IP: 监视器所在机器的 IP 地址。

MON_VERSION: 监视器版本信息, 如果监视器和守护进程版本不一致, 则不允许建立连接。

12. tip

打印当前守护系统的运行状态。

检查库状态、库的守护进程状态、库的分裂信息、库的可加入信息、是否存在多个主库以及 MPP 环境下 dmmppctl 是否处于一致状态等系统信息，并打印检查结果。

13. login

登录监视器，登录口令和服务器的用户登录口令一致，且必须有 SYSDBA 权限，默认使用 SYSDBA/SYSDBA。

监视器优先选择主库校验登录口令，如果没有活动主库，则选择一个活动备库校验登录口令，部分监视器命令需要先登录才允许执行。

14. logout

退出登录。

15. get takeover time

如果在主库正常退出时，配置了备库自动接管需要等待的时间（MON_TAKEOVER_SHUTDOWN），则可通过该命令打印备库自动接管仍需等待的剩余时间，若主备库状态正常，则打印无自动接管实例。

16. show state

显示当前监视器所在监视器配置组的所有监视器的状态（监视器角色）等信息。

相关字段含义如下：

MON_BRO_INTERVAL：监视器实际广播超时。

MON_VOTE_INTERVAL：监视器实际选举超时。

MON_NAME：监视器名。

MON_STATE：监视器状态（相对于当前监视器）。

ID：监视器在系统中的 ID。

MON_ROLE：监视器当前角色（显示当前监视器所知的内容）。

MON_IP：监视器 IP。

MON_PORT：监视器 PORT。

17. startup

dmwatcher

database

[group_name.]db_name

打开指定库的守护进程监控功能，只有一个组的情况下，可以不指定 group_name。

此命令是指在指定库的守护进程处于 Shutdown 状态时，将其切换至 Startup 状态，恢复守护进程的监控功能，而不是启动守护程序。如果守护进程不是处于 Shutdown 状态，则跳过不执行。

另外切换至 Startup 状态后，守护进程可能会根据本地或远程库状态信息自动做一些状态切换处理，不一定会一直保持在 Startup 状态。

18. startup dmwatcher [group_name]

启动指定组的所有守护进程监控功能，只有一组的情况下，可以不指定 group_name。

此命令是指在守护进程处于 Shutdown 状态时，将其切换至 Startup 状态，恢复守护进程的监控功能，而不是启动守护程序。如果守护进程不是处于 Shutdown 状态，则跳过不执行。

另外切换至 Startup 状态后，守护进程可能会根据本地或远程库状态信息自动做一些状态切换处理，不一定会一直保持在 Startup 状态。

19. stop dmwatcher database[group_name.]db_name

关闭指定库的守护进程监控功能，只有一组的情况下，可以不指定 group_name。执行此命令需要先登录监视器。

此命令只是将指定库的守护进程切换为 Shutdown 状态，并不会退出守护进程，守护进程切换至 Shutdown 状态后，不会再自动处理故障，也不会自动启动实例，但仍然能够正常接收、发送消息，监视器仍然能够监控到最新的实例状态。

如果本地守护的是 DMDSC 集群，此命令只会将控制守护进程切换为 Shutdown 状态，不会修改普通守护进程状态。

20. stop dmwatcher [group_name]

关闭指定组的所有守护进程监控功能，只有一组的情况下，可以不指定 group_name。执行此命令需要先登录监视器。

此命令只是将守护进程切换为 Shutdown 状态，并不会退出守护进程，守护进程切换至 Shutdown 状态后，不会再自动处理故障，也不会自动启动实例，但仍然能够正常接收、发送消息，监视器仍然能够监控到最新的实例状态。

如果本地守护的是 DMDSC 集群，此命令只会将控制守护进程切换为 Shutdown 状态，不会修改普通守护进程状态。

21. startup group [group_name]

启动指定组中的所有节点实例，只有一组的情况下，可以不指定 group_name。

如果守护进程的监控功能被关闭，处于 Shutdown 状态，在通知实例启动之前，会先打开守护进程的监控功能，再通知守护进程启动实例。对 DMDSC 集群，如果组内 dmcss 的自动拉起功能是关闭的，在各节点实例拉起成功后，会将其打开。

如果守护进程 dmwatcher.ini 中 INST_AUTO_RESTART 配置为 1，则对应实例不允许通过命令方式启动，需要等待守护进程自动拉起实例。



注意：

另外，如果守护进程守护的是 DMDSC 集群，控制守护进程收到命令后，会转发给每个 dmcss 执行，由 dmcss 拉起本地的节点实例，dmcss 执行完成后，再由控制守护进程将执行结果返回给守护监视器。

同样的，如果 dmcss 的自动拉起标记为 TRUE，也不允许通过此命令拉起 dmcss 本地的节点实例，需要等待 dmcss 自动拉起实例。

22. stop group [group_name]

退出指定组的所有节点实例，只有一组的情况下，可以不指定 group_name。执行此命令需要先登录监视器。

为了避免守护进程将实例再次启动，退出时会先通知关闭所有守护进程的监控、重启等功能，切换守护进程为 Shutdown 状态。如果已经是 Shutdown 状态，则跳过不通知。

退出时，如果组中存在 Open 状态的主库，则先通知主库退出，再依次通知其他备库退出，否则就按照 dmmonitor.ini 中的配置顺序，依次通知每个处于活动状态的实例退出。

如果守护进程守护的是 DMDSC 集群，监视器会通知控制守护进程切换为 Shutdown 状态。控制守护进程收到退出命令后，首先通知组内所有 dmcss 关闭自动拉起功能，然后再转发退出命令给 dmcss 控制节点，由 dmcss 控制节点通知所有节点实例执行正常退出，再将 dmcss 控制节点的执行结果返回给守护监视器。

执行完此命令后，守护进程会一直保持在 Shutdown 状态，如果是通过手动方式将所有库重启，则需要先通过 startup dmwatcher 命令将守护进程监控功能打开，整个守护系统才可以正常 Open。

23. open database [group_name.] db_name

强制 Open 指定组的指定库，只有一组的情况下，可以不指定 group_name，但数据库名 db_name 必须指定。如果指定有组名 group_name，则组名和实例名需要用“.”分隔。执行此命令需要先登录监视器。

正常情况下，守护进程会自动 Open 实例，不需要手动执行此命令，如果发现某个库一直无法自动 Open，可尝试使用此命令。

命令执行时，首先会检查守护进程是否可以执行自动 Open，如果不符合执行条件，则命令会报错返回，需要等待守护进程自动处理；否则允许命令执行，将指定库强制 Open。

24. choose switchover [group_name]

选出指定组中允许执行 Switchover 的备库列表，只有一组的情况下，可以不指定 group_name。

25. switchover [group_name[.]] [db_name]

将指定组中的指定备库切换为新的主库，将当前的主库切换为新的备库，此命令要求执行切换的主备库都是正常 Open 状态。执行此命令需要先登录监视器。

只有一组的情况下，可以不指定 group_name，组中只有一个备库的情况下，可以不指定 db_name，如果 group_name 和 db_name 同时指定，则需要用“.”符合分隔。

26. choose takeover [group_name]

指定组中主库出现故障时，可以通过此命令选出组中可以执行正常接管的备库列表，只有一组的情况下，可以不指定 group_name。

如果指定组中存在有活动主库，则不再选择可接管的备库列表。

27. takeover [group_name[.]] [db_name]

使用指定组中的指定备库接管组中的故障主库。执行此命令需要先登录监视器。

只有一组的情况下，可以不指定 group_name，组中只有一个备库的情况下，可以不指定 db_name，如果 group_name 和 db_name 同时指定，则需要用“.”符合分隔。

该命令要求监视器曾经收到过故障主库的历史信息，并且主库的历史信息必须满足被接管条件，可以先通过 choose takeover [group_name] 选出符合正常接管条件的备库列表，选择其中一个备库执行接管操作即可。

28. choose takeover force [group_name]

指定组中主库出现故障时，可以通过此命令选出组中可以执行强制接管的备库列表，只

有一组的情况下，可以不指定 `group_name`。

如果指定组中存在有活动主库，则不再选择可强制接管的备库列表。

出现主库故障时，需要优先使用 `choose takeover [group_name]` 选出可接管主库，并使用 `takeover [group_name[.]] [db_name]` 来执行正常接管。



注意：

如果不存在可以正常接管的备库，但又需要主库对外提供服务，可再考虑使用 `choose takeover force [group_name]` 选出可强制接管的备库，并使用 `takeover force [group_name[.]] [db_name]` 进行强制接管。但是强制接管可能会导致出现分裂情况。

29. `takeover force [group_name[.]] [db_name]`

使用指定组中的指定备库强制接管组中的故障主库。执行此命令需要先登录监视器。

只有一组的情况下，可以不指定 `group_name`，组中只有一个备库的情况下，可以不指定 `db_name`，如果 `group_name` 和 `db_name` 同时指定，则需要用“.”符合分隔。

30. `set group [group_name] auto restart on`

打开指定组内所有 DMDSC 集群节点实例的自动拉起标记。

DMDSC 集群的自动拉起是靠 `dmcass` 执行的，`dmcass` 只负责拉起和自己的 `dcr_seqno` 相同的 `db` 节点。

同一个守护系统内，可能有多个 DMDSC 集群，这些 DMDSC 集群有各自的 `dmcassm` 监视器，可以通过 `dmcassm` 命令分别控制每个集群内 `dmcass` 的自动拉起标记，但是操作起来比较麻烦，通过此命令，在守护监视器 `dmmonitor` 上，可以一次性打开所有 DMDSC 集群上 `dmcass` 的自动拉起标记，不需要再逐个去操作每个 `dmcassm`。

此命令执行时，有可能所有节点实例都还未启动，因此不要求校验登录口令。

此命令只修改 `dmcass` 内存中的自动拉起标记值，不会修改 `dmdcr.ini` 中的配置值，如果 `dmcass` 重启，此修改会丢失。

如果 `dmcass` 未配置自动拉起参数（`DMDCR_DB_RESTART_INTERVAL/`
`DMDCR_DB_STARTUP_CMD`），此命令会执行失败。

31. `set group [group_name] auto restart off`

关闭指定组内所有 DMDSC 集群节点实例的自动拉起标记。

和 `set group [group_name] auto restart on` 命令相对应，可以一次性关闭守护系统内所有 DMDSC 集群上 dmcss 的自动拉起标记。

此命令执行时，需要校验登录口令。

此命令只修改 dmcss 内存中的自动拉起标记值，如果 dmcss 重启，此修改会丢失。

32. `set database [group_name.]db_name recover time time_value`

设置主库到指定备库的 Recovery 时间间隔，取值范围 (3~ 86400)s。执行此命令需要先登录监视器。

只有一组时可以不指定 `group_name`，`db_name` 必须是备库实例名。命令执行时会查找 `db_name` 对应的主库，并通知主库的守护进程修改到备库的 `recover` 间隔时间为指定的 `value` 值，注意此命令只修改主库守护进程的内存值 (`INST_RECOVER_TIME`)，并且只对指定备库起作用，不会写入到 `dmwatcher.ini` 文件，如果主库是 DMDSC 集群，则会同时修改主库所有守护进程中的内存值。

命令执行时要求主库和主库的守护进程必须是活动的，否则无法查找 `db_name` 对应的主库信息，会导致命令执行失败。

设置成功后，在指定备库发生故障重启时，主库的守护进程要等待距离指定备库上次的恢复时间超过指定的时间间隔后，才再次启动 Recovery 检查，若满足 [3.1.9 故障恢复处理](#) 中列出的 Recovery 条件则进行 Recovery 流程，否则不能进行 Recovery。

对指定备库执行 Recovery 完成后，主库守护进程会根据执行结果重置内存中对指定备库的恢复间隔，本次命令设置的恢复间隔不再有效，具体请参考 [3.1.9 故障恢复处理](#)。

33. `set database [group_name.]db_name arch invalid`

设置主库到指定备库的归档状态无效。执行此命令需要先登录监视器。

只有一组时可以不指定 `group_name`，`db_name` 必须是备库实例名。

命令执行时会查找 `db_name` 对应的主库，并通知主库修改到备库的归档状态无效。

命令执行时要求主库和主库的守护进程必须是活动的，否则无法查找 `db_name` 对应的主库信息，会导致命令执行失败。

在主备库正常运行时，用户如果想人为退出备库（例如，需要修改备库配置），并且不想触发主库的 Failover 流程（避免影响应用执行），可先使用 `set recover time` 命令，将备库的 Recovery 间隔修改到一个较大的值 (3~86400s)，然后使用此命令通知主库修

改备库的归档状态无效，即可达到目的。

之所以先使用 `set recover time` 命令设置恢复间隔，是为了防止先设置归档无效后，在备库退出前，主库的守护进程立马启动 Recovery 流程，又重新将备库归档恢复到有效状态，避免出现备库退出后还会触发主库执行 Failover 的情况。

34. `set group [group_name] para_name para_value`

修改指定组的所有守护进程的指定参数值，此命令同时修改守护进程内存值和配置文件 `dmwatcher.ini` 中的参数值。执行此命令需要先登录监视器。

如果守护进程守护的是 DMDSC 集群，此命令会同时修改所有守护进程的内存值和配置文件值。

如果组中某个守护进程故障，则跳过不修改，如果要重启故障的守护进程，需要在重启前修改守护进程 `dmwatcher.ini` 中的 `DW_ERROR_TIME` 和远程一致，否则 `dmmonitor` 执行命令时会报错处理。

`group_name` 指定组名，如果没有指定，在 MPP 主备多组的环境下，会通知所有组全部执行。目前可修改的参数包括：`DW_ERROR_TIME`、`INST_RECOVER_TIME`、`INST_ERROR_TIME`、`INST_AUTO_RESTART`、`INST_SERVICE_IP_CHECK`、`RLOG_SEND_THRESHOLD`、`RLOG_APPLY_THRESHOLD`。

如果指定的参数是 `INST_RECOVER_TIME`，待主库对某个备库执行过恢复操作后，会根据恢复结果重置主库守护进程内存中对此备库的恢复间隔值（不会影响 `dmwatcher.ini` 中的值），具体请参考 [3.1.9 故障恢复处理](#)。

35. `set group [group_name] recover time time_value`

设置指定组中主库到所有备库的 Recovery 时间间隔，取值范围 (3~86400) s。执行此命令需要先登录监视器。

如果没有指定 `group_name`，在 MPP 主备多组的环境下，会通知所有组全部执行。

命令执行时，要求主库和主库的守护进程必须是活动的，监视器通知主库依次修改到每个备库的恢复间隔为指定的 `value` 值（不包括守护进程控制文件已经处于 `Invalid` 或者 `Split` 状态的备库）。

注意此命令只修改主库守护进程的内存值 (`INST_RECOVER_TIME`)，不会写入到 `dmwatcher.ini` 文件，如果主库是 DMDSC 集群，此命令会同时修改主库所有守护进程中的内存值。

设置成功后，在备库发生故障重启时，主库的守护进程要等待距离备库上次的恢复时间超过指定的时间间隔后，才再次启动 Recovery 检查，若满足 3.1.9 故障恢复处理中列出的 Recovery 条件则进行 Recovery 流程，否则不能进行 Recovery。

对指定备库执行 Recovery 完成后，主库守护进程会根据执行结果重置内存中对指定备库的恢复间隔，本次命令设置的恢复间隔不再有效，具体请参考 3.1.9 故障恢复处理。

36. set group [group_name] arch invalid

设置指定组中主库到所有备库的归档状态无效。执行此命令需要先登录监视器。

如果没有指定 group_name，在 MPP 主备多组的环境下，会通知所有组全部执行。

命令执行时要求主库和主库的守护进程必须是活动的，监视器通知主库依次修改到所有备库的归档状态无效（不包括守护进程控制文件已经处于 Invalid 或者 Split 状态的备库）。

在主库有大量操作产生大量 Redo 日志的情况下，为避免备库产生日志堆积影响主库性能，可结合 set group 的几个命令，设置备库为异步方式同步数据，待主库空闲后，再通知主库同步数据到备库，以避免备库拖慢主库性能。

设置步骤如下：

1) 通过 set group [group_name] recover time time_value 将所有备库的恢复间隔设置到一个比较大的值（3~86400s）。

2) 再通过 set group [group_name] arch invalid 将备库归档失效，主库就不会再发送联机日志到备库重做。

3) 待主库空闲时，可再通过 set group [group_name] recover time time_value 将备库的恢复间隔设置到一个比较小的值（3~86400s），启动备库的恢复流程，以达到备库数据的异步同步。

步骤 1) 在步骤 2) 之前做，是为了避免设置归档无效后，主库的守护进程立马启动 Recovery 流程，又重新将备库归档恢复到有效状态。

37. clear database [group_name.]db_name arch send info

清理指定组中主库到指定备库实例的最近 N 次归档发送信息（通知主库执行），N 值取主库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际归档发送次数中的较小值。可通过查询 V\$ARCH_SEND_INFO 查询实际发送次数。

如果主库是 DMDSC 集群，会清理主库所有节点到备库的最近 N 次归档发送信息。

主库守护进程判断备库归档发送是否异常，是根据（最近 N 次的总的发送耗时）/ N 得到的平均值来判断的，如果平均值大于主库守护进程配置的 RLOG_SEND_THRESHOLD 值，则认为到备库的归档发送出现异常，主库守护进程会启动对应的异常处理。

在异常处理完成后，异常备库归档处于无效状态，主库守护进程在达到恢复间隔后会启动到异常备库的 Recovery 恢复处理，在主备数据同步完成之后，主库 Suspend 之前，也会根据同样的方法判断归档发送是否异常，如果判断异常，则不允许恢复异常备库的归档为有效状态。

因此，在需要尽快恢复备库有效的场景下，可借助此命令清理掉最近 N 次的归档发送信息来达到目的，但这只是步骤之一，还需要清理备库的最近 N 次重演信息，具体请参考命令 `clear database [group_name.]db_name apply stat` 和 `clear group [group_name] apply stat` 的使用说明。

38. clear group [group_name] arch send info

清理指定组中源库到所有备库（包括异步备库）的最近 N 次归档发送信息，N 值取源库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际归档发送次数中的较小值。源库可能是主库，也可能是异步备库（级联配置方式中作为源库），可通过查询 V\$ARCH_SEND_INFO 查询实际发送次数。

如果没有指定组名则通知所有组执行。

如果主库是 DMDSC 集群，此命令会通知主库所有节点清理最近 N 次的归档发送信息。

此命令的使用场景可参考 `clear database [group_name.]db_name arch send info` 命令说明。

39. clear database [group_name.]db_name apply stat

清理指定备库的最近 N 次重演信息（通知指定备库执行），N 值取备库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际重演次数中的较小值，可通过查询 V\$RAPPLY_STAT 查询实际备库重演次数。

如果主库是 DMDSC 集群，备库会清理对主库所有节点的最近 N 次重演信息。主库守护进程判断备库重演日志是否正常，是根据（最近 N 次的等待时间+最近 N 次的重演时间）/ N 得到的平均值来判断的，如果平均值大于主库守护进程配置的 RLOG_APPLY_THRESHOLD 值，则认为备库重演异常。

在对异常备库执行 Recovery 恢复时,在主备数据同步完成之后,主库 Suspend 之前,会根据此方法判断备库重演是否异常,如果判断异常,则不允许恢复异常备库的归档为有效状态。

因此,在需要尽快恢复备库有效的场景下,可借助此命令清理掉指定备库最近 N 次的日志重演信息来达到目的,但这只是步骤之一,还需要清理主库到备库的最近 N 次归档发送信息,具体请参考命令 `clear database [group_name.]db_name arch send info` 和 `clear group [group_name] arch send info` 的使用说明。

40. clear group [group_name] apply stat

清理指定组中所有实时或即时备库实例的最近 N 次重演信息(通知所有备库执行),N 值取备库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际重演次数中的较小值,可通过查询 V\$RAPPLY_STAT 查询实际备库重演次数。

如果没有指定组名则通知所有组执行。

如果主库是 DMDSC 集群,备库会清理对主库所有节点的最近 N 次重演信息。

此命令的使用场景可参考 `clear database [group_name.]db_name apply stat` 命令说明。

41. show arch send info [group_name.]db_name

显示源库到指定备库的日志发送信息,以及源库的控制守护进程本地到指定备库的 Recovery 间隔信息。也支持异步备库的查询,其中 Recovery 间隔信息对异步备库无效。

查询结果中的字段说明如下:

■ 主库的控制守护进程字段信息

下面这些信息只对 Global 守护类型的备库有效。

LAST RECOVER TIME: 主库守护进程上次启动对指定备库的 Recovery 时间, NONE 表示未执行过恢复操作。

LAST_RECOVER CODE: 主库守护进程上次对指定备库执行 Recovery 的结果, Code 小于 0 表示执行失败,等于 0 表示执行成功,等于 100 表示实际未执行。

INST_RECOVER_TIME(IN DMWATCHER.INI): 主库的 dmwatcher.ini 中配置的 INST_RECOVER_TIME 值,对所有备库有效。

INST_RECOVER_TIME(IN DMWATCHER MEMORY): 主库的守护进程内存中的 INST_RECOVER_TIME 值,只对指定备库有效,可通过 SET RECOVER TIME 命令动态修

改。



如果主库是 DMDSC 集群，则会按照以下格式显示主库的每个节点到备库控制

注意：节点的归档发送相关信息。

■ 主库到指定备库的当前状态信息

MPP_FLAG: 是否为 MPP 主备环境，值为 TRUE 或 FALSE。

ARCH TYPE: 主库到指定备库的归档类型，包括 REALTIME/TIMELY 两种类型。

MAL STATUS: 主库到指定备库的 MAL 链路状态，包括 CONNECTED/ DISCONNECTED 两种状态。

ARCH STATUS: 主库到指定备库的归档状态，包括 VALID/INVALID 两种状态。

■ 主库最近一次向备库发送日志的历史信息

SEND TYPE: 主库到指定备库的日志发送类型。FOR REALTIME SEND/FOR TIMELY SEND 表示主库向备库正常的联机日志发送，FOR RECOVER SEND 表示主库的守护进程发起的 Recovery 流程中的归档日志发送，FOR ASYNC SEND 表示定时器定时触发的异步归档日志发送。

SEND START TIME: 主库到指定备库的日志发送起始时间，NONE 表示未执行过发送操作。

SEND END TIME: 主库到指定备库的日志发送结束时间，该字段为 NONE 时，如果 SEND START TIME 也是 NONE，则表示未执行过发送操作，如果 SEND START TIME 是具体的时间，则表示发送尚未结束。

SEND TIME USED: 主库到指定备库最近一次的日志发送耗时（微秒）。

SEND START LSN: 主库到指定备库的日志发送起始 LSN。

SEND END LSN: 主库到指定备库的日志发送结束 LSN。

SEND LOG LEN: 主库到指定备库最近一次发送的日志长度（字节）。

SEND PTX COUNT: 主库到指定备库最近一次发送的 PTX 个数。

SEND CODE: 主库到指定备库的日志发送结果，Code 小于 0 表示发送失败，等于 0 表示发送成功，等于 100 表示实际未发送。

SEND DESC INFO: 主库到指定备库的日志发送结果描述信息。

■ 主库最近 N 次向备库发送日志的历史信息

N 值取主库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际归档发送总次数中的

较小值。

SEND LOG LEN: 主库最近 N 次累计发送到指定备库的日志长度 (字节)。

SEND PTX COUNT: 主库最近 N 次累计发送到指定备库的 PTX 个数。

SEND TIME USED: 主库最近 N 次累计发送日志耗时 (微秒)。

主库守护进程判断到备库的日志发送是否出现异常, 就是判断 SEND TIME USED/N 得出的平均值是否大于主库 dmwatcher.ini 中配置的 RLOG_SEND_THRESHOLD 值, 如果大于, 则认为备库异常, 主库守护进程就会启动异常处理, 具体请参考 [3.1.7 备库异常处理](#) 小节。

■ 主库向备库发送日志的历史信息中的最大值

MAX SEND TIME USED: 主库向备库发送日志的最大耗时记录 (微秒)。

MAX END TIME: 主库向备库发送日志最大耗时的结束时间。

MAX PTX COUNT: 主库向备库发送日志的最大的 PTX 个数记录。

MAX SEND LOG LEN: 主库向备库发送日志的最大长度记录 (字节)。

MAX END LSN: 主库向备库发送日志的最大结束 LSN。

其中 MAX SEND TIME USED/ MAX END TIME 记录的是同一组信息, 其余的记录信息都是相互独立没有关联的。

■ 主库向备库发送日志的全部累计信息

TOTAL SEND COUNT: 主库向备库发送日志总次数

TOTAL SEND LOG LEN: 主库向备库发送的总日志长度 (字节)

TOTAL SEND PTX COUNT: 主库向备库发送的总 PTX 个数

TOTAL SEND TIME USED: 主库向备库发送日志总耗时 (微秒)

42. show apply stat [group_name.]db_name

显示指定备库的日志重演信息, 也支持异步备库的查询。



如果主库是 DMDSC 集群, 则会按照以下格式依次显示备库对主库每个节点的

注意: 重演信息。

■ 备库最近一次的日志重演信息

RECEIVED LOG LEN: 备库最近一次收到的日志长度 (字节)。

RESPONSE TIME USED: 备库最近一次响应主库耗时 (微秒)。

WAIT TIME USED: 备库最近一次重演日志的等待时间（微秒）。

APPLY LOG LEN: 备库最近一次重演日志长度（字节）。

APPLY TIME USED: 备库最近一次重演日志耗时（微秒）。

■ 备库最近 N 次的日志重演信息

N 值取备库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际重演总次数中的较小值。

RECEIVED LOG LEN: 备库最近 N 次收到的日志长度（字节）。

RESPONSE TIME USED: 备库最近 N 次响应主库耗时（微秒）。

WAIT TIME USED: 备库最近 N 次重演等待时间（微秒）。

APPLY LOG LEN: 备库最近 N 次重演日志长度（字节）。

APPLY TIME USED: 备库最近 N 次重演日志耗时（微秒）。

主库守护进程判断备库的日志重演是否出现异常，就是判断 $(WAIT TIME USED + APPLY TIME USED) / N$ 得出的平均值是否大于主库 dmwatcher.ini 中配置的 RLOG_APPLY_THRESHOLD 值，如果大于，则认为备库重演日志异常，主库守护进程不会恢复备库归档为有效状态，具体请参考 [3.1.9 故障恢复处理](#) 小节。

■ 备库日志重演历史信息中的最大记录值

MAX RESPONSE TIME: 备库响应主库的最大耗时记录（微秒）。

MAX WAIT TIME: 备库重演日志时的最大等待时间（微秒）。

MAX APPLY TIME: 备库的历史最大重演耗时（微秒）。

MAX APPLY LOG LEN: 备库最大重演耗时对应的日志长度（字节）。

其中 MAX APPLY TIME/ MAX APPLY LOG LEN 是同一组信息，其余的记录信息都是相互独立没有关联的。

■ 备库日志重演的全部累计信息

TOTAL RECEIVED NUM: 备库收到的总的日志缓存个数。

TOTAL RECEIVED LOG LEN: 备库收到的总的日志缓存长度（字节）。

TOTAL APPLY NUM: 备库已经重演的总的日志缓存个数。

TOTAL APPLY LOG LEN: 备库已经重演的总的日志长度（字节）。

TOTAL RESPONSE TIME: 备库响应主库的总耗时（微秒）。

TOTAL WAIT TIME: 备库上重演日志总的等待时间（微秒）。

TOTAL APPLY TIME: 备库上总的重演耗时（微秒）。

43. detach database [group_name.]db_name

此命令允许将全局守护类型的备库分离出守护进程组，执行此命令需要先登录监视器。

使用此命令不会触发主库的 Failover 故障处理流程。该操作不会修改相关的配置信息，只是为了需要主动退出维护备库时使用。

该命令包括下面两个操作步骤：

- 1) 通知主库设置到指定备库的恢复间隔内存值为 86400(s)。
- 2) 通知主库修改到指定备库的归档状态无效。

从上面步骤可以看出，此命令是前面几个命令的结合，步骤 1 相当于执行 set recover time 命令，步骤 2 相当于执行 set arch invalid 命令，使用此命令可简化手工操作步骤，一次完成将指定实例分离出守护进程组的所有操作。

使用此命令的前提是备库的守护进程控制文件状态必须是有效的。

如果需要将备库退出，可借助 stop database 命令完成此功能。

执行此命令需要先登录监视器，命令执行成功后，在分离出的备库维护工作完成之后，可通过下面方式将实例重加入守护进程组。

如果备库在维护时手动退出了，可先通过 startup database 命令或者手工方式将备库重启。

在备库处于活动状态并自动 Open 后，主库内存中对备库的恢复间隔时间仍然是 86400(s)，如果想尽快启动备库的恢复流程，可以使用 attach



注意：database [group_name.]db_name 命令将备库重加入，或者使用 set recover time 命令将备库的恢复间隔改小。

需要注意的是，此命令中的两个操作步骤都是通知当前主库执行的，如果主库在分离出去的备库重新加回之前发生故障，其他备库接管成为新主库，则之前的分离操作会失效，需要重新对此备库执行 detach 命令，通知当前的新主库执行以上操作。

44. attach database [group_name.]db_name

此命令和 detach database [group_name.]db_name 相对应，待 detach 分离出去的备库维护完成后，可通过此命令将其重加入守护进程组，执行此命令需要先登录监视器。

此命令的操作就是通知主库修改到指定备库的恢复间隔为 3s，如果主库是 DMDSC 集

群，则会通知主库所有守护进程修改到指定备库的恢复间隔内存值为 3s。

可以将此命令等同于 `set database [group_name.]db_name recover time time_value` 命令来使用。

45. kill group [group_name]

强制杀掉指定组中的活动实例。

守护进程收到监视器的 `kill group` 命令请求时，会先切换为 Shutdown 状态（如果已经处于 Shutdown 状态则不再切换），同时强制杀掉所守护的实例进程（如果实例已经故障处于 Error 状态，则跳过不执行）。

执行此命令时，实例当前可能正在执行守护进程发起的命令，无法校验监视器的登录口令，因此不要求用户先登录监视器。

如果守护进程守护的是 DMDSC 集群，监视器会通知控制守护进程切换为 Shutdown 状态。控制守护进程收到 `kill` 命令后，首先通知组内所有 `dmcss` 关闭自动拉起功能，然后再转发 `kill` 命令给 `dmcss` 控制节点，由 `dmcss` 控制节点通知各节点实例强制退出，再将 `dmcss` 控制节点的执行结果返回给守护监视器。



注意：

执行完此命令后，守护进程会一直保持在 Shutdown 状态，如果是通过手动方式将所有库重启，则需要先通过 `startup dmwatcher` 命令将守护进程监控功能打开，整个守护系统才可以正常 Open。

46. check recover [group_name.]db_name

检查指定组的指定备库是否满足自动恢复条件。

主库和守护进程都处于正常 Open 状态时，如果有备库发生故障重启，主库守护进程会自动检测备库是否满足自动恢复条件，如果满足，主库守护进程会发起 Recovery 流程执行故障恢复。

如果备库没有被自动恢复，可通过此命令获取备库不满足条件的原因。

47. check open [group_name.]db_name

检查指定组的指定库是否满足自动 Open 条件。

正常情况下，在库中所有 ok 节点启动到 Mount 状态后，守护进程会根据本地和远程守护进程发送的实例状态信息进行一系列条件判断，如果满足条件，守护进程会通知本地所有 ok 节点执行 `ALTER DATABASE OPEN FORCE` 语句，将实例启动到 Open 状态，守护

进程也会切换到 Open 状态，这一过程不需要人工干预。

如果守护进程判定本地库不符合自动 Open 条件，则不会通知其执行 OPEN FORCE 语句，此时可以借助此命令来获取不满足条件的原因。

48. startup database [group_name.]db_name

启动指定组的指定库实例。

支持单机和 DMDSC 集群的拉起。

如果指定的库是单机，则由守护进程执行拉起动作；如果指定的库是 DMDSC 集群，则由守护进程再去通知组内所有 dmcsc 各自执行拉起动作。

此命令执行前，如果守护进程处于 Shutdown 状态，会将其切换为 Startup 状态。对 DMDSC 集群，如果组内 dmcsc 的自动拉起功能是关闭的，在各节点实例拉起成功后，会将其打开。

如果守护进程 dmwatcher.ini 中 INST_AUTO_RESTART 配置为 1，则对应实例不允许通过命令方式启动，需要等待守护进程自动拉起实例。



注意：

同样的，如果守护进程守护的是 DMDSC 集群，dmcsc 的自动拉起标记为 TRUE，也不允许通过此命令拉起 dmcsc 本地的节点实例，需要等待 dmcsc 自动拉起实例。

49. stop database [group_name.]db_name

正常退出指定组的指定库实例。

支持单机和 DMDSC 集群的正常退出。

如果指定的库是单机，则由守护进程通知单机库正常退出；如果指定的库是 DMDSC 集群，则由守护进程再去通知 dmcsc 控制节点依次退出每个节点实例。



注意：会将每个 dmcsc 的自动拉起功能关闭。

此命令执行完成后，会将守护进程切换为 Shutdown 状态。对 DMDSC 集群，

如果要退出的是全局守护类型的备库，为了避免触发主库的 Failover 动作，要求此备库已经被 detach 分离出去，也就是备库归档要处于无效状态，才允许将其正常退出。

如果要退出的是主库，在自动切换模式下，为了避免触发备库的自动接管，要求所有备库都已经被 detach 分离出去，也就是所有备库归档都处于无效状态，才允许退出主库，手动切换模式下没有此要求。

50. kill database [group_name.]db_name

强杀指定组的指定库实例。

支持单机和 DMDSC 集群的强杀。

如果指定的库是单机，则由守护进程强制杀掉指定库的进程；如果指定的库是 DMDSC 集群，则由守护进程再去通知 dmcss 控制节点强杀所有节点实例。

此命令执行完成后，会将守护进程切换为 Shutdown 状态。对于 DMDSC 集群，会将所有 dmcss 的自动拉起功能关闭。

51. set database [group_name.]db_name until time time_value

设置指定组的指定异步备库重演到的时间点，指定的库必须为异步备库。

支持指定的时间值 time_value 精确到微秒。

命令执行时，首先找到异步备库对应的源库，监视器向守护进程发起重演到指定时间点的请求，守护进程根据请求修改源库的内存值，然后通知源库立即发送归档。其中，归档的发送是异步进行的，不会阻塞监视器命令窗口。

注意此命令只会通知当前有效的源库进行设置，如果源库是实时主库或即时主库，且在设置成功后出现故障，则用户需要对由备库接管后的新主库重新进行设置。

用户也可以在主备库上分别执行系统函数 SP_SET_ARCH_SEND_UNTIL_TIME 进行设置，详细请参考《DM8_SQL 语言使用手册》。

52. cancel database [group_name.]db_name until time

取消指定组的指定异步备库重演到的时间点，指定的库必须为异步备库。

命令执行时，首先找到异步备库对应的源库，监视器向守护进程发起取消重演到指定时间点的请求，守护进程根据请求修改源库的内存值。

注意此命令只会通知当前有效的源库进行设置，如果源库是实时主库或即时主库，且在设置成功后出现故障，则用户需要对由备库接管后的新主库重新进行设置。

如果用户通过系统函数 SP_SET_ARCH_SEND_UNTIL_TIME 在所有主备库上都设置了重演到的时间点，则需要在主备库上分别执行 SP_SET_ARCH_SEND_UNTIL_TIME 来取消设置，详细请参考《DM8_SQL 语言使用手册》。

53. **show mpp**

显示 MPP 站点信息，该命令仅支持 MPP 主备环境下使用。显示的信息和 `dmmppl.ctl` 中记录的节点信息一致。

相关字段含义如下：

INST_NAME：配置的 MPP 节点实例名。

EP_SEQNO：配置的 MPP 节点号。

EP_IP：MPP 节点 IP 地址，此信息不包含在 `dmmppl.ctl` 文件中，用户可在需要连接 MPP 节点时使用。

EP_PORT：MPP 节点端口号，此信息不包含在 `dmmppl.ctl` 文件中，用户可在需要连接 MPP 节点时使用。

54. **startup dmwatcher all**

启动所有组的守护进程监控功能，该命令仅支持 MPP 主备环境下使用。

此命令是指在守护进程处于 Shutdown 状态时，将其切换至 Startup 状态，恢复守护进程的监控功能，并不是启动守护程序，如果守护进程不是处于 Shutdown 状态，则跳过不再处理。

另外，切换至 Startup 状态后，守护进程可能会根据本地或远程实例状态信息自动做一些状态切换处理，不一定会一直保持在 Startup 状态。

55. **stop dmwatcher all**

关闭所有组的守护进程监控功能，该命令仅支持 MPP 主备环境下使用。执行此命令需要先登录监视器。

此命令只是将守护进程切换为 Shutdown 状态，并不会退出守护进程，守护进程切换到 Shutdown 状态后，不会再自动处理故障，也不会自动启动实例，但仍然能够正常接收、发送消息，监视器仍然能够监控到最新的实例状态。

56. **startup group all**

启动所有组的所有实例，该命令仅支持 MPP 主备环境下使用。

如果守护进程的监控功能被关闭，处于 Shutdown 状态，在通知实例启动之前，会先打开守护进程的监控功能，再通知守护进程启动实例。



如果守护进程 `dmwatcher.ini` 中 `INST_AUTO_RESTART` 配置为 1, 则对应

注意: 实例不允许通过命令方式启动, 需要等待守护进程自动拉起实例。

57. stop group all

退出所有组的所有实例, 该命令仅支持 MPP 主备环境下使用。执行此命令需要先登录监视器。

退出时会先通知关闭所有守护进程的监控功能, 切换守护进程为 Shutdown 状态, 避免守护进程将实例再次启动, 如果已经是 Shutdown 状态, 则跳过不再通知。

退出时, 如果组中存在 Open 状态的主库, 则先通知主库退出, 再依次通知其他备库退出, 否则就按照 `dmmonitor.ini` 中的配置顺序, 依次通知每个处于活动状态的实例退出。

58. kill group all

强制杀掉所有组中的活动实例, 该命令仅支持 MPP 主备环境下使用。

守护进程收到监视器的 kill 命令请求时, 会先切换为 Shutdown 状态, 同时强制杀掉所守护的实例进程 (如果实例已经故障处于 Error 状态, 则跳过不再执行)。

执行此命令时, 实例当前可能正在执行守护进程发起的其他命令, 无法校验监视器的登录口令, 因此不要求用户先登录监视器。

59. check mppctl

检查当前所有活动主库的 `dmmppctl` 文件内容是否一致, 仅支持 MPP 主备环境下使用, 如果某个组的主库出现故障, 则跳过不检查。

60. recover mppctl

恢复当前所有活动主库的 `dmmppctl` 到一致状态, 仅支持 MPP 主备环境下使用。执行此命令需要先登录监视器。

如果某个组的主库故障, 则跳过不恢复, 待主库恢复正常后, 可再通过 `check mppctl` 命令校验, 如果不一致, 可再尝试使用 `recover mppctl` 恢复到一致状态。

例如, MPP 系统中, 主备库切换, 备库接管等过程中, 主备信息发生变更, 新主库的信息将更新到 `dmmppctl` 中。如果此时, 另外一个主库发生故障, 因为还未来得及同步最新的 `dmmppctl`, 重启后, 可能造成主库的 `dmmppctl` 不一致的情况, 导致 MPP 系统不能正常工作。这种情况下, 可通过 `dmmonitor` 的检查 (`check mppctl`) 和修复 (`recover`

mppctl) 命令进行同步该控制文件。

4.5 监视器 LOG 日志

监视器 LOG 日志记录监视器自己的信息和守护进程的本地信息，在 LOG 日志中分别以 [monitor] 和 [守护进程本地的实例名] 开头。监视器 LOG 日志的命名格式为 “dmmonitor_年月日时分秒.log”，例如 “dmmonitor_20160418230523.log”。LOG 日志的路径通过 dmmonitor.ini 文件中 MON_LOG_PATH 来设置，如果没有设置则和 dmmonitor.ini 在同一个目录下。

5 配置文件说明

与 DM 数据守护相关的配置文件包括：

- 数据库配置文件 dm.ini
- 数据库控制文件 dm.ctl
- MAL 配置文件 dmmal.ini
- Redo 日志归档配置文件 dmarch.ini
- 守护进程配置文件 dmwatcher.ini
- 监视器配置文件 dmmonitor.ini
- 定时器配置文件 dmtimer.ini
- MPP 控制文件 dmmpp.ctl 等等

其中，dmmpp.ctl 在 [2.5.2 dmmpp.ctl 维护](#) 中介绍；dm.ctl 不需要用户修改，只要放在指定的目录即可。

各配置文件的存放路径：

1. dm.ini 存放目录没有限制，一般直接放在数据库目录中。
2. dmmal.ini、dmarch.ini、dmtimer.ini 存放目录由 dm.ini 的 CONFIG_PATH 配置项指定
3. dmwatcher.ini 存放目录没有限制，一般和 dm.ini 存放在同一个目录。
4. dmmonitor.ini 存放目录没有限制，一般和 dm.ini 存放在同一个目录。
5. dm.ctl 存放目录由 dm.ini 的 CTL_PATH 配置项指定。
6. dmmpp.ctl 存放目录由 dm.ini 的 SYSTEM_PATH 配置项指定。

下面分别介绍各配置文件中相关配置项的含义。

5.1 dm.ini

dm.ini 是 DM 数据库配置文件，下表介绍了 dm.ini 中与数据守护相关的配置参数。

表 5.1 dm.ini 数据守护相关配置项

配置项	配置含义
INSTANCE_NAME	数据库实例名（长度不超过 16 个字符），与 dmmal.ini 中的

	MAL_INST_NAME 对应。配置数据守护系统时，应该保持 INSTANCE_NAME 是全局唯一的。
PORT_NUM	服务器监听通讯端口号，服务器配置此参数，有效值范围（1024~65534），发起连接端的端口在 1024-65535 之间随机分配。此参数配置应与 dmmal.ini 中的 MAL_INST_PORT 对应。
DW_PORT	服务器监听守护进程连接请求的端口，服务器配置此参数，有效值范围（1024~65534），发起连接端的端口在 1024-65535 之间随机分配。 数据守护 V3.0 版本中，此参数仅作为单节点库的兼容参数保留。对于新配置的数据守护系统，如果是单节点库，建议改用 dmmal.ini 中的 MAL_INST_DW_PORT 参数进行配置，如果是 DMDSC 集群，此参数不再有用，必须使用 dmmal.ini 中的 MAL_INST_DW_PORT 参数进行配置。
DW_INACTIVE_INTERVAL	服务器认定守护进程未启动的时间，有效值范围（0~1800），单位为 s，默认 60s。 如果服务器距离上次收到守护进程消息的时间间隔在设定的时间范围内，则认为守护进程处于活动状态，此时，不允许手工执行修改服务器模式、状态的 SQL 语句； 如果超过设定时间仍没有收到守护进程消息，则认为守护进程未启动，此时如果 ALTER_MODE_STATUS 参数设为 1，则允许手工方式执行这类 SQL 语句。
ALTER_MODE_STATUS	是否允许手工修改数据库的模式和状态以及 OGUID, 1 表示允许，0 表示不允许，此参数可动态修改，默认为 1，数据守护环境下建议配置为 0，避免用户手工干预。
ENABLE_OFFLINE_TS	是否允许 offline 表空间，1 表示允许，0 表示不允许，2 表示禁止备库，其他放开。守护环境下建议配置为 2。
SESS_FREE_IN_SUSPEND	远程归档失败会导致系统挂起；为了防止主备库之间网络故障、备库强制接管后，应用连接一直挂住不切换到新主库，设置该参数，表示归档失败挂起后隔一段时间自动断开所有连接。默认值

	60s，取值范围 0-1800s，0 表示不断开。
MAL_INI	MAL 系统配置开关，0 表示不启用 MAL 系统，1 表示启用 MAL 系统。
ARCH_INI	Redo 日志归档配置开关，0 表示不启动 Redo 日志归档，1 表示启用 Redo 日志归档
TIMER_INI	是否启用定时器，0：不启用；1：启用
MPP_INI	是否启用 MPP 系统，0：不启用；1：启用 MPP 主备环境下，MPP 主库和全局守护类型的备库都需要配置为 1。
DW_MAX_SVR_WAIT_TIME	数据库等待守护进程启动的最大时间（范围 0~65534s）。如果设定时间内，守护进程没有启动，数据库实例强制退出。0 代表不检测，缺省为 0
REDOS_BUF_SIZE	待重演日志堆积的内存限制，堆积的日志缓冲区占用内存超过此限制，则新的日志将会被延迟加入重演队列，等待重演释放部分内存后再加入。以兆为单位，有效值范围（0~65536M），默认 1024。0 表示无内存限制。 REDOS_BUF_SIZE 和 REDOS_BUF_NUM 同时起作用，只要达到一个条件即会触发延迟处理。 此参数可用在备库重演、故障重启重做 Redo 日志、DSC 故障处理以及还原后的恢复场景中，如果是备库重演，由于新的日志被延迟加入重演队列，因此会延迟响应回主库。
REDOS_BUF_NUM	待重演日志缓冲区允许堆积的数目限制，超过限制则新的日志将会被延迟加入重演队列，等待堆积数减少后再加入。以个数为单位，有效值范围（0~9999），默认 4096。0 表示无限制。 REDOS_BUF_SIZE 和 REDOS_BUF_NUM 同时起作用，只要达到一个条件即会触发延迟处理。 此参数可用在备库重演、故障重启重做 Redo 日志、DSC 故障处理以及还原后的恢复场景中，如果是备库重演，由于新的日志被延迟加入重演队列，因此会延迟响应回主库。

REDOS_MAX_DELAY	备库重演日志缓冲区的时间限制，超过此限制则认为重演异常，服务器自动宕机，防止日志堆积、主库不能及时响应用户请求。 以秒(s)为单位，取值范围(0~7200)，默认1800s。0表示无重做时间限制。 此参数只在备库重演场景中起作用。
REDOS_PRE_LOAD	重演日志时是否开启预加载功能，在重演 Redo 日志的同时，根据参数设置提前解析 RLOG_PKG 中的 Redo 日志，并预加载数据页到缓存中。 取值(0、1)，0表示不开启，1表示开启，默认值为0，避免在某些场景下干扰到日志重演线程，反而影响备库重演性能。 此参数可用在备库重演、故障重启重做 Redo 日志、DSC 故障处理以及还原后的恢复场景中。
RLOG_SEND_APPLY_MON	此参数对主备库均有效。 对于主库，用于指定统计最近 N 次主库到每个备库的归档发送时间。对于备库，用于指定统计最近 N 次备库重演日志的时间。 N 为此参数设置的值，默认主备库均统计最近 64 次的时间信息。 取值范围(16~1024)，静态参数，默认值 64。
MESSAGE_CHECK	是否对通信消息启用消息体校验(只有当消息的发送端和接收端都配置为 1 才启用通信体校验)。0：不启用；1：启用。默认值 1。
REDOS_PARALLEL_NUM	指定日志并行重演的线程数，取值范围(1~64)，1表示不开启日志并行重演，默认值为 1。 此参数可用在备库重演、故障重启重做 Redo 日志、DSC 故障处理以及还原后的恢复场景中。
REDOS_ENABLE_SELECT	是否允许备库执行查询操作，取值(0、1)，默认值为 1。 0：仅允许备库查询 v\$ 动态视图，其他所有查询操作均不支持。 1：允许备库执行所有查询操作。 设置为 0 时，会在一定程度上提升备库的重演性能。
DW_CONSISTENCY_CHECK	备库是否检测主库到备库的连接丢失或归档失效，并在检测到上

	述情况时停止提供只读服务。0：不检测；1：检测。默认值为 0。
DW_ARCH_SPACE_CHECK	备库是否检测本地归档磁盘空间不足，并在磁盘空间不足时主动关闭服务，同时将本地守护进程切换为 SHUTDOWN 状态。0：不检测；1：检测。默认值为 0。 此参数目前仅支持单节点备库的检测。
RLOG_PKG_SEND_NUM	主库启动异步恢复时，连续发送设定值个日志包以后需要等待备库响应。有效值范围（1~10000）。缺省值 1 表示每发送一个日志包就要等待一次响应。如果主备库之间存在比较大的网络延迟，可以通过调大此参数以提升主库的日志发送速度。



注意：

数据守护环境下不允许修改 **TS_MAX_ID** 和 **TS_FIL_MAX_ID** 参数。

5.2 dmmal.ini

dmmal.ini 是 MAL 配置文件。需要用到 MAL 环境的实例，所有站点 dmmal.ini 需要保证严格一致。

表 5.2 dmmal.ini 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，数据守护环境不建议配置为 0，防止网络故障导致服务长时间阻塞
MAL_SEND_THRESHOLD	最大待发送邮件数，用于控制待发送邮件总数，防止发送邮件堆积。有效值范围（1024-10000），默认值 2048
MAL_CONN_FAIL_INTERVAL	判定 MAL 链路断开的时间，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，如果配置为 0，则表示不限制单个 MAL 缓存大小

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 lz 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 lz 算法，压缩率相对低
MAL_TEMP_PATH	指定临时文件的目录。当邮件使用的内存超过MAL_BUF_SIZE 或者MAL_SYS_BUF_SIZE时，将新产生的邮件保存到临时文件中。如果缺省，则新产生的邮件保存到temp.dbf文件中
MAL_COMBIN_BUF_SIZE	指定MAL消息合并发送的缓冲区大小，单位KB。MAL发送消息时进行合并，减少网络发送次数。有效值范围（0~131072）。0表示不进行消息合并。最小缓冲区大小128k，当设置值大于0小于128时，自动调整为128
MAL_MESSAGE_CHECK	是否对MAL通信消息启用消息体校验（只有当消息的发送端和接收端都配置为1才启用通信体校验）。0：不启用；1：启用。默认为1
[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。 对于 ipv4，直接使用 ip 地址即可，例如： MAL_HOST = 192.168.100.109 对于 ipv6，若当前环境存在多块网卡，需要用%号指定具体有效的网卡序号或网卡名称；若只有一块网卡或者已配置默认网卡，则可以不指定序号或名称。例如： MAL_HOST = fe80::6aa7:3f02:59b3:bcb4%3 其中，%3 为实际网卡序号
MAL_PORT	MAL 监听端口，用于数据守护、DSC、MPP 等环境中各节点实例之间 MAL 链路配置，监听端口配置此参数，范围 1024~65534，

	发起连接端的端口在 1024~65535 之间随机分配
MAL_INST_HOST	实例对外服务的 IP 地址。若 dm.ini 中配置了 LISTEN_IP，则 MAL_INST_HOST 必须与 LISTEN_IP 保持一致。 对于 ipv4，直接使用 ip 地址即可，例如： MAL_INST_HOST = 192.168.100.109 对于 ipv6，若当前环境存在多块网卡，需要用%号指定具体有效的网卡序号或网卡名称；若只有一块网卡或者已配置默认网卡，则可以不指定序号或名称。例如： MAL_INST_HOST = fe80::6aa7:3f02:59b3:bcb4%3 其中，%3 为实际网卡序号
MAL_INST_PORT	MAL_INST_NAME 实例服务器监听通讯端口号，服务器配置此参数，有效值范围（1024~65534），发起连接端的端口在 1024~65535 之间随机分配 此参数的配置应与 dm.ini 中的 PORT_NUM 保持一致
MAL_DW_PORT	MAL_INST_NAME 实例守护进程的监听端口，其他守护进程或监视器使用 MAL_HOST + MAL_DW_PORT 创建与该实例守护进程的 TCP 连接，监听端配置此参数，有效值范围（1024~65534），发起连接端的端口在 1024~65535 之间随机分配
MAL_INST_DW_PORT	节点实例监听守护进程的端口，同一个库上的各实例的守护进程使用 MAL_HOST + MAL_INST_DW_PORT 和各实例创建 TCP 连接，实例服务器配置此参数，有效值范围（1024~65534），发起连接端的端口在 1024~65535 之间随机分配
MAL_LINK_MAGIC	MAL 链路网段标识，有效值范围（0~65535），默认 0。设置此参数时，同一网段内的节点都设置相同，不同网段内的节点设置的值必须不一样
MAL_MESSAGE_CHECK	是否对 MAL 通信消息启用消息体校验（只有当消息的发送端和接收端都配置为 1 才启用通信体校验）。0：不启用；1：启用。 默认为 1



注意:

所有站点的 **MAL** 配置参数 **MAL_COMPRESS_LEVEL** 必须要一致, 否则节点间将不能建立链路, 导致系统无法运行 (在服务器 **log** 日志中可看到打印提示信息)。另外配置不为 0 时, 需要保证每个节点都能加载到对应的动态压缩库文件 (**snappy** 或 **zlib**), 如果未加载成功, 则默认变成 0, 也可能导致链路建立不成功。

5.3 dmarch.ini

dmarch.ini 是 Redo 日志归档配置文件。

表 5.3 dmarch.ini 配置项

配置项	配置含义
ARCH_WAIT_APPLY	备库收到 Redo 日志后, 是否需要重演完成后再响应主库。0 表示收到马上响应 (高性能模式), 1 表示重演完成后响应 (事务一致模式)。配置为即时归档的读写分离集群时, 默认值为 1; 配置为实时归档的读写分离集群时, 默认值为 0
ARCH_RESERVE_TIME	归档日志保留时间, 单位分钟, 取值范围 (0~4294967294)。只对远程归档和本地归档有效。服务器每隔 5 分钟检查是否存在超过保留时间的归档并删除。默认为 0, 表示不删除归档
ARCH_LOCAL_SHARE	DMDSC 集群本地归档是否共享给远程节点作为远程归档。0 表示不共享, 1 表示共享, 默认值为 0。此参数值在 DMDSC 集群各个节点的配置文件中需保持一致。 配置为 1 时, 本地归档路径和远程归档路径均需配置在 ASM 或共享磁盘上, 并且 dmarch.ini 中 ARCH_INCOMING_PATH 为对应节点的本地归档路径; 配置为 0 时, remote 归档路径与对应节点 local 归档路径不能指向共享存储的同一个目录
[ARCH_NAME]	Redo 日志归档名, 由于 “STANDBY_ARCHIVE” 用于表示备库生成的归档日志, 因此不允许将归档名称配置为 “STANDBY_ARCHIVE”

ARCH_TYPE	Redo 日志归档类型，LOCAL/REMOTE/REALTIME/TIMELY/ASYNC、分别表示本地归档/远程归档/实时归档/即时归档/异步归档
ARCH_DEST	归档目标，本地归档为归档文件存放路径，其他归档方式设置为目标数据库实例名，如果目标库为 DMDSC 库，则需要写上 DMDSC 每个实例名，以 '/' 分隔（如 DSC01/DSC02） 注：REMOTE 远程归档是 DMDSC 库内部实例相互配置，归档目标都是单个实例
ARCH_FILE_SIZE	单个 Redo 日志归档文件大小，取值范围（64M~2048M），对本地归档和远程归档有效，缺省为 1024MB，即 1G
ARCH_SPACE_LIMIT	Redo 日志归档空间限制，当同一节点的本地归档文件或远程归档文件达到限制值时，系统自动删除最早生成的归档日志文件。 0 表示无空间限制，取值范围（1024M~2147483647M），对本地归档和远程归档有效，缺省为 0
ARCH_FLUSH_BUF_SIZE	归档合并刷盘缓存大小，单位 MB，取值范围（0~128），缺省为 0，表示不使用归档合并刷盘
ARCH_TIMER_NAME	定时器名称，仅对异步归档有效
ARCH_INCOMING_PATH	对应远程归档目标 ARCH_DEST 在本地存放的归档路径
ARCH_SEND_DELAY	指定源库到异步备库的归档延时发送时间，单位为分钟，取值范围（0~1440），缺省为 0，表示不启用归档延时发送功能。 注：仅在异步归档类型中支持此配置项，其他归档类型不支持。 如果源库是 DSC 集群，建议用户配置时保证各节点上配置的值是一致的，并保证各节点所在机器的时钟一致，避免控制节点发生切换后计算出的归档延迟发送时间不一致

5.4 dmwatcher.ini

dmwatcher.ini 是守护进程配置文件。

表 5.4 dmwatcher.ini 配置项

配置项	配置含义
-----	------

[GROUP_NAME]	守护进程组名（长度不能超过 16）
DW_TYPE	守护类型，默认为 LOCAL LOCAL：本地守护 GLOBAL：全局守护
DW_MODE	切换模式，默认为 MANUAL MANUAL：故障手动切换模式 AUTO：故障自动切换模式
DW_ERROR_TIME	守护进程故障认定时间，取值范围为（3s~32767s），缺省 15 秒没有收到远程守护进程消息，即认定远程守护进程故障，对本地守护无效。 另外此参数也是监视器认定守护进程的故障时间，超过设置的时间间隔仍没有收到守护进程消息，监视器认为守护进程出现故障。
INST_ERROR_TIME	数据库故障认定时间，取值范围为（3s~32767s），缺省 15 秒没有收到数据库发送的状态信息，即认定其监控的数据库出现故障
INST_OGUID	数据守护唯一标识码，同一守护进程组中的所有数据库、守护进程和监视器，都必须配置相同的 OGUID 值，取值范围为 0-2147483647。
INST_INI	监控数据库 dm.ini 路径。dmwatcher 从 dm.ini 配置文件获取 DW_PORT 信息，并进一步从 dmmal.ini 中获取 MAL_HOST/MAL_DW_PORT 等信息。
DCR_INI	dmdcr.ini 文件路径。只有 DMDSC 集群的数据守护环境下才需要配置此参数。守护进程启动时从 DCR 磁盘上获取 DMCSS 和 DMASMSVR 的 IP/PORT 信息。
INST_AUTO_RESTART	是否自动重启数据库实例，0：不自动重启 1：自动重启。缺省为 0
INST_STARTUP_CMD	数据库启动命令。 1. linux 命令行方式启动（不能出现带有空格的路径）： INST_STARTUP_CMD = /opt/dm/bin/dmserver 2. linux 服务方式启动： INST_STARTUP_CMD = service dmserverd restart

	3. Windows 命令行启动： <pre>INST_STARTUP_CMD = c:\dm\bin\dmserver</pre> 4. Windows 服务方式启动： <pre>INST_STARTUP_CMD = net start 注册服务名（注册服务名，可通过 DM 服务查看器获取）</pre>
INST_RECOVER_TIME	备库故障恢复检测时间间隔，取值范围 0~86400s，缺省每 60 秒检查一下备库状态，满足故障恢复条件时，启动历史数据同步流程。 数据守护系统启动完成后、Switchover 主备切换后、Takeover 备库接管后以及强制 Open 主库后，主库守护进程 INST_RECOVER_TIME 内存值会强制设置成 0，确保尽快启动数据同步。另外，还可以通过监视器命令 set recover time 修改 INST_RECOVER_TIME 内存值。
INST_SERVICE_IP_CHECK	守护进程是否监控实例对外服务，取值范围：0、1，默认为 0。 配置为 1 时，守护进程会自动检测 Open 主库的公共网络是否故障，故障认定时间为 INST_ERROR_TIME 配置的时间值，如果认定公共网络故障，则会通知主库实例强制退出。 注意：配置为 1 时，只会对已经 Open 的主库实例进行网络故障检测，如果主库实例没有 Open 或者主库实例故障或者是备库实例，此参数无效。
RLOG_SEND_THRESHOLD	用于指定主库发送日志到备库的时间阈值。 如果主库守护进程检测到某个备库最近 N 次的平均日志发送时间大于此参数设置的值，则主库守护进程认为此备库出现异常，会启动异常处理，将此备库归档失效，N 值取主库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和主库实际发送归档次数中的较小值（可通过查询 V\$ARCH_SEND_INFO 获取实际发送归档次数）。 取值范围（0~86400），单位为秒，配置为 0 时此监控功能关闭，默认值为 0。 此参数对主库守护进程有效，建议主备库的守护进程都进行配置，以便备库切换为主库后使用。

RLOG_APPLY_THRESHOLD	用于指定备库重演日志的时间阈值。 如果某个备库最近 N 次的平均日志重演时间大于此参数设置的值，则主库守护进程不会将其归档恢复为有效状态，N 值取备库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和备库实际重演次数中的较小值（可通过查询 V\$RAPPLY_STAT 获取实际重演次数）。 取值范围（0~86400），单位为秒，配置为 0 时此监控功能关闭，默认值为 0。 此参数对主库守护进程有效，建议主备库的守护进程都进行配置，以便备库切换为主库后使用。
DW_RECONNECT	用于指定单节点主库或单节点备库在和本地守护进程的通信链路断开后，守护进程是否直接认定本地实例故障并将其 kill。0：在通信链路断开后，守护进程会直接认定本地实例故障并将其 kill，在需要快速启动自动接管的场景下，可以将此参数配置为 0；1：在通信链路断开后，守护进程会尝试自动重连本地实例。缺省为 1
DW_FAILOVER_FORCE	用于在自动切换模式下，主库向备库发送日志失败后挂起，但确认监视器未启动，无法返回确认消息给主库守护进程时，主库守护进程是否强制将主库重新 Open。 0：在此场景下，不支持将主库强制 Open，主库一直保持在挂起状态；1：在此场景下，主库守护进程经过 3 倍的 DW_ERROR_TIME 确认时长后，如果认定自己仍然满足重新 Open 条件，则将主库重新 Open（具体请参考 6.17 小节）。缺省为 1

5.5 dmmonitor.ini

dmmonitor.ini 是监视器配置文件。

表 5.5 dmmonitor.ini 配置项

配置项	配置含义
MON_DW_CONFIRM	是否配置为确认模式，缺省为 0。0：监控模式 1：确认模式
MON_LOG_PATH	日志文件路径，日志文件命名方式为“dmmonitor_年月日时分秒.log”，例如“dmmonitor_20150418230523.log”。

	其他请参考表后附加说明
MON_LOG_INTERVAL	自动记录系统状态信息到日志文件的时间间隔，取值 (0、1 或者 (5~3600))，单位秒，0 表示不记录任何日志，1 表示只记录监视器正常接收到的消息，5~3600 表示除了记录监视器正常接收消息外，每隔指定的间隔另外记录系统信息到日志文件中，默认值为 1
MON_LOG_FILE_SIZE	单个日志文件大小，范围 16~2048，单位为 M，默认值为 64，达到最大值后，会自动生成并切换到新的日志文件中，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
MON_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
MON_TAKEOVER_SHUTDOWN	主库正常退出时，备库自动接管需要等待的时间。范围 0~86400，单位为秒，默认值为 0，0 表示不启用主库正常退出，备库自动接管功能。非 0 表示主库正常退出后，超过设定的时间主库仍未重启，备库可以完成自动接管
[GROUP_NAME]	守护进程组名，与 dmwatcher.ini 中的守护进程组名保持一致
MON_INST_OGUID	数据守护唯一标识码，与 dmwatcher.ini 中的 INST_OGUID 保持一致
MON_DW_IP	守护进程 IP 地址和监听端口。 配置格式为：“守护进程 IP 地址:守护进程监听端口”。 其中 IP 地址和 dmmal.ini 中的 MAL_HOST 保持一致，端口和 dmmal.ini 中的 MAL_DW_PORT 保持一致。 如果需要监控的是 DMDSC 集群，作为一个整体的库，将 DMDSC 集群中所有守护进程的“IP:PORT”配置为一个 MON_DW_IP 项，每个守护进程的“IP:PORT”以“/”分隔。比如： MON_DW_IP=192.168.0.73:9236/192.168.0.73:9237。 如果使用 IPV6 地址，为了区分端口，需要用[]封闭 IP 地址。 对于 IPV6，若当前环境存在多块网卡，需要用%号指定具体有

	效的网卡序号或网卡名称；若只有一块网卡或者已配置默认网卡，则可以不指定序号或名称。 MON_DW_IP = [fe80::6aa7:3f02:59b3:bcb4%3]:52184
MON_MESSAGE_CHECK	是否对 MON 通信消息启用消息体校验（只有当消息的发送端和接收端都配置为 1 才启用通信体校验）。0：不启用；1：启用。 默认为 1
MON_INST_ID	监视器实例在监视器系统中的 ID，取值（1~2147483647）
MON_MID	当前监视器系统的唯一标识（监视器系统中若只有一个确认监视器则可以不配置该项），取值（1~2147483647）
MON_ID	当前监视器在监视器系统中的 ID，取值（1~2147483647）
MON_HOST	监视器实例在监视器系统中通信的 IP
MON_PORT	监视器实例在监视器系统中通信的 PORT
MON_INST_NUM	配置监视器实例的总个数，取值（1、3 或者 5）
MON_HB_INTERVAL	各监视器之间的通信心跳校验间隔，取值（5~600），单位秒
MON_BRO_INTERVAL	各监视器之间的通信心跳间隔，取值（5~600），单位毫秒，为避免用户输入不合理的数据，内部会对该值进行一定处理，实际参数值可通过 show state 命令进行查看
MON_VOTE_INTERVAL	监视器的基础投票间隔，取值（5~600），单位毫秒，为避免用户输入不合理的数据，内部会对该值进行一定处理，实际参数值可通过 show state 命令进行查看
[INST_NAME]	监视器实例名，与 GROUP_NAME 形式相同，根据后续配置项决定该内容属于 GROUP_NAME 还是 INST_NAME

MON_LOG_PATH 配置说明：

1. 对于 dmmonitor 命令行工具

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

2. 对于 dwmon_init 接口

接口说明可参考附录部分。

对于通过调用接口方式监控守护系统的情况，如果调用 `dwmon_init` 接口时指定有 `log_path` 参数，则将指定的 `log_path` 作为日志文件路径；如果没有指定 `log_path` 参数，则将 `dmmonitor.ini` 中配置的 `MON_LOG_PATH` 作为日志文件路径。如果 `dmmonitor.ini` 中没有配置 `MON_LOG_PATH`，则将 `dmmonitor.ini` 配置文件所在的路径作为日志文件路径。

只有在 `MON_LOG_INTERVAL` 配置大于 0 的情况下才会产生日志信息，并写入到日志文件中，日志文件路径请参考上述说明。

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



注意：（`MON_LOG_FILE_SIZE`）决定继续写入已有的日志文件或者创建新的日志文件写入。

创建新的日志文件时，根据设定的日志总空间大小（`MON_LOG_SPACE_LIMIT`）决定是否删除创建时间最早的日志文件。

5.6 dmtimer.ini

`dmtimer.ini` 用于配置定时器，可记录异步备库的定时器信息，具体使用请见异步备库配置章节。`dmtimer.ini` 的配置项见下表，其中项目名称就是定时器名称。表中参数与创建定时器的过程 `SP_ADD_TIMER()` 参数用法相同，更为详细的参数介绍，请参考《DM8_SQL 语言使用手册》中 `SP_ADD_TIMER()` 中参数介绍。

表 5.6 dmtimer.ini 的配置项

配置项	配置含义
[TIMER_NAME1]	定时器名称。例如，名为 RT 的定时器，名称书写方式为[RT]
TYPE	定时器调度类型： 1：执行一次。时间上只需要指定 <code>DURING_START_DATE</code> 即可，不需要指定 <code>START_TIME</code> 、 <code>END_TIME</code> 和 <code>DURING_END_DATE</code> 2：按日执行 3：按周执行 4：按月执行的第几天 5：按月执行的第一周 6：按月执行的第二周

	7: 按月执行的第三周 8: 按月执行的第四周 9: 按月执行的最后一周 10: 按 REPEAT_INTERVAL 指定的日历表达式的频率进行。不再需要通过 START_TIME、END_TIME、DURING_START_DATE 和 DURING_END_DATE 指定时间
FREQ_MONTH_WEEK_INTERVAL	间隔月或周数。这个列的可选值与上面的不同调度类型有关。表示执行的频率。说明如下: 当 TYPE=1 时, 这个值无效, 系统不会做检查。 当 TYPE=2 时, 表示每隔几天执行, 值的有效范围为 1 到 100。 当 TYPE=3 时, 表示的是每隔几个星期执行, 值的范围没有具体的限制。 当 TYPE=4 时, 表示每几个月中的某一天执行, 值的范围没有具体的限制。 当 TYPE 为 5、6、7、8 或 9 时, 都表示每几个月的某一周执行, 值的范围也没有具体的限制
FREQ_SUB_INTERVAL	间隔天数。这个列的可选值与上面的不同调度类型有关。表示执行的频率, 在 FREQ_MONTH_WEEK_INTERVAL 基础上, 继续指定更为精细的频率。说明如下: 当 TYPE=1 或 2 时, 这个值无效, 系统不会做检查。 当 TYPE=3 时, 表示的是某一个星期的星期几执行, 可以同时选中七天中的任意几天。取值范围 1~127。具体如何取值, 请用户参考如下规则。因为每周有七天, 所以 DM 用七位二进制来表示选中的日子。从最低位开始算起, 依次表示周日、周一...周五、周六。选中周几, 就将该位置 1, 否则 0。例如, 选中周二和周六, 7 位二进制就是 1000100, 转化成十进制就是 68, 所以 FREQ_SUB_INTERVAL 就取值 68。 当 TYPE=4 时, 表示将在一个月的第几天执行, 此时这个列的有效值范围为 1 到 31 当 TYPE 为 5、6、7、8 或 9 时, 都表示将在某一周内第几天执行, 有效值为 1 到 7, 分别表示从周一到周日
FREQ_MINUTE_INTERVAL	间隔分钟数。当 TYPE 为 2、3、4、5、6、7、8 或 9 时, 在 FREQ_MONTH_WEEK_INTERVAL 和 FREQ_SUB_INTERVAL 的基础上, 继续指定更为精细的频率。 表示一天内每隔多少分钟执行一次。取值范围 0~1439, 0 表示一天内执行一次
REPEAT_INTERVAL	当 TYPE 为 10 时使用, 用来指定的日历表达式。语法详见《DM8 系

	统包使用手册》手册 DBMS_SCHEDULER 包中关于 REPEAT_INTERVAL 的介绍
START_TIME	定时器开始时间，TIME 类型。例如 20:59:59.999999
END_TIME	定时器结束时间，TIME 类型
DURING_START_DATE	指定定时器的有效日期范围的起始日期，必须是有效的日期字符串，不可以为空。TIMESTAMP 类型。例如 2021-12-15 10:00:00
DURING_END_DATE	指定定时器的有效日期范围的结束日期，可以为空。如果不为空，必须是有效的日期字符串，同时必须是在 DURING_START_DATE 日期之后。TIMESTAMP 类型
NO_END_DATE_FLAG	是否结束标记。1 是，0 否
DESCRIBE	定时器描述
IS_VALID	定时器有效标记，默认为 0。 0：表示已关闭定时器 1：表示已启用定时器

例 1，配置名称为“TIMER_01”的定时器，TYPE = 1，开始时间为 2016-02-22 17:30:00，只执行一次。

TYPE = 1 时，只需要指定 DURING_START_DATE 即可。START_TIME、END_TIME 和 DURING_END_DATE 均不需要。

```
[TIMER_01]

TYPE                        = 1
FREQ_MONTH_WEEK_INTERVAL   = 0
FREQ_SUB_INTERVAL          = 0
FREQ_MINUTE_INTERVAL       = 0
START_TIME                 =
END_TIME                   =
DURING_START_DATE          = 2016-02-22 17:30:00
DURING_END_DATE            =
NO_END_DATE_FLAG           = 0
DESCRIBE                   = RT TIMER
IS_VALID                   = 1
```

例 2：配置名称为“TIMER_02”的定时器，TYPE = 2，每天 01:00:00 触发。

```
[TIMER_02]

TYPE                        = 2

FREQ_MONTH_WEEK_INTERVAL   = 1

FREQ_SUB_INTERVAL          = 0

FREQ_MINUTE_INTERVAL       = 0

START_TIME                 = 01:00:00

END_TIME                   = 01:00:10

DURING_START_DATE          = 2016-02-11 17:36:09

DURING_END_DATE            = 9999-12-31 23:59:59

NO_END_DATE_FLAG           = 1

DESCRIBE                   = RT TIMER

IS_VALID                   = 1
```

例 3：配置名称为“TIMER_03”的定时器，TYPE = 10，每秒触发一次。

```
[TIMER_03]

TYPE= 10

FREQ_MONTH_WEEK_INTERVAL=

FREQ_SUB_INTERVAL=

FREQ_MINUTE_INTERVAL=

REPEAT_INTERVAL= FREQ=SECONDLY;INTERVAL=1

START_TIME= 15:20:00

END_TIME=

DURING_START_DATE= 2021-12-15 10:00:00

DURING_END_DATE= 2021-12-20 10:00:00

NO_END_DATE_FLAG= 1

DESCRIBE= RT TIMER

IS_VALID= 1
```

5.7 端口配置关系说明

对上述 ini 文件中的各个端口之间的对应关系说明如下。

1. dm.ini

dm.ini 中端口相关的有两个配置项：PORT_NUM、DW_PORT。

■ PORT_NUM

是数据库实例的监听端口，监听用户的连接请求，dmmal.ini 中的 [MAL_INST_NAME: MAL_INST_PORT] 要与 dm.ini 中的 [INSTANCE_NAME: PORT_NUM] 保持一致。

■ DW_PORT

是守护系统中数据库监听守护进程连接请求的端口，由守护进程主动建立到数据库的 TCP 连接，这个端口只用于数据库和守护进程之间的消息交互使用。

DM8 版本支持 DMDSC 数据守护后，该参数转移到 dmmal.ini 的 MAL_INST_DW_PORT，dm.ini 中无需设置。此处保留是为了单节点库的向下兼容，对 DMDSC 集群，此参数不再起作用。

对于单节点库，如果同时配置了 DW_PORT 和 MAL_INST_DW_PORT，则要求两个配置值相同，如果只配置了其中一个配置项，则使用此配置项的值作为有效值。

对于新配置的数据守护系统，如果是单节点库，建议改用 dmmal.ini 中的 MAL_INST_DW_PORT 进行配置，不需要再配置 dm.ini 中的 DW_PORT。如果是 DMDSC 集群，则 dm.ini 中的 DW_PORT 不再有用，必须在 dmmal.ini 中配置 MAL_INST_DW_PORT。

例如，以主库 GRP1_RT_A，备库 GRP1_RT_B 为例，来说明如何使用 PORT_NUM、DW_PORT。

主库 GRP1_RT_A 的 dm.ini：

```
#这里只列举端口相关的参数，其他参数配置请参考后文的配置举例

INSTANCE_NAME          = GRP1_RT_A

PORT_NUM               = 5236   #数据库实例监听端口

DW_PORT               = 5237   #守护进程监听端口
```

备库 GRP1_RT_B 的 dm.ini：

```
#这里只列举端口相关的参数，其他参数配置请参考后文的配置举例

INSTANCE_NAME          = GRP1_RT_B

PORT_NUM               = 6236   #数据库实例监听端口

DW_PORT               = 6237   #守护进程监听端口
```

2. dmmal.ini

dmmal.ini 中每个实例都有一个单独的 mal 配置项，每个实例中有以下几个端口相关

的配置：MAL_PORT、MAL_INST_PORT、MAL_DW_PORT、MAL_INST_DW_PORT。

■ MAL_PORT

MAL 监听端口，用于创建 MAL 链路，同一个实例的 MAL 配置项中，MAL_PORT 不能和实例 dm.ini 中的两个端口相同，避免端口绑定冲突。

■ MAL_INST_PORT

MAL 配置项中，MAL_INST_NAME 实例的监听端口，和实例 dm.ini 的 PORT_NUM 值相同。

■ MAL_DW_PORT

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。监视器配置文件 dmmonitor.ini 中，MON_DW_IP 就是一组 MAL_HOST:MAL_DW_PORT。

■ MAL_INST_DW_PORT

实例对守护进程的监听端口，守护进程使用 MAL_HOST + MAL_INST_DW_PORT 创建到实例的 TCP 连接。

例如，这里以主库 GRP1_RT_A，备库 GRP1_RT_B 为例进行说明。

```
MAL_CHECK_INTERVAL      = 5
MAL_CONN_FAIL_INTERVAL  = 5

[MAL_INST1]

MAL_INST_NAME           = GRP1_RT_A

MAL_HOST                 = 192.168.0.141

MAL_PORT                 = 5238

MAL_INST_HOST            = 192.168.1.131

MAL_INST_PORT            = 5236      #数据库实例监听端口，对应 dm.ini 中的 PORT_NUM

MAL_DW_PORT              = 5239      #守护进程监听端口

MAL_INST_DW_PORT         = 5237

[MAL_INST2]

MAL_INST_NAME           = GRP1_RT_B

MAL_HOST                 = 192.168.0.142

MAL_PORT                 = 6238

MAL_INST_HOST            = 192.168.1.132
```

```

MAL_INST_PORT          = 6236    #数据库实例监听端口，对应 dm.ini 中的 PORT_NUM
MAL_DW_PORT            = 6239    #守护进程监听端口
MAL_INST_DW_PORT       = 6237

```

3. dmwatcher.ini

守护进程启动时，需要获取远程实例对应的配置项信息，创建到远程守护进程的 TCP 连接，这些信息通过 INST_INI 配置项来获取。

守护进程从 INST_INI 路径找到并读取 dm.ini、dmmal.ini、dmarch.ini，综合各个配置项得到远程守护进程的 IP、端口等信息并建立连接。

DMDSC 集群的守护进程启动后，需要连接到所有的 dmcss，如果 DMDSC 集群使用有 asm 文件系统，守护进程也需要连接到 asmsvr，守护进程根据 DCR_INI 路径从 DCR 磁盘获取到 dmcss 和 asmsvr 的 IP、端口等信息并建立连接。

4. dmmonitor.ini

监视器配置项中以守护进程组为单位，通过 MON_DW_IP 配置到每组的所有守护进程的 IP 和端口信息。

配置格式为：“守护进程 IP 地址:守护进程 TCP 端口”。

其中守护进程 IP 地址和 dmmal.ini 中的 MAL_HOST 保持一致，守护进程 TCP 端口和 dmmal.ini 中的 MAL_DW_PORT 保持一致。

dmmal.ini 中配置有多少个 IP 和 PORT，dmmonitor.ini 就需要配置多少个对应的“IP:PORT”信息，否则会无法接收到消息。

dmmonitor.ini 中的 MON_HOST 和 MON_PORT 是监视器实例在监视器系统中的监听端口，其他监视器向该 IP: PORT 发送相应消息，所有监视器的监视器实例部分的配置内容应该一致，其中 IP 和 PORT 不允许出现冲突。

对 DMDSC 集群的所有守护进程，作为一个整体配置在同一个 MON_DW_IP 项中，每个守护进程的“IP:PORT”以“/”分隔开。

这里以监控主库 GRP1_RT_A，备库 GRP1_RT_B 进行举例说明，假如 GRP1_RT_A 是单机，GRP1_RT_B 是 DMDSC 集群：

#这里只对 IP/端口信息举例说明，其他配置参数请参考后面的配置举例

```

[GRP1]

MON_INST_OGUID        = 82379

```

```
#IP 和 PORT 信息和 dmmal.ini 中的 MAL_HOST 和 MAL_DW_PORT 配置项一致

MON_DW_IP          = 192.168.0.141:5239          #GRP1_RT_A

MON_DW_IP          = 192.168.0.142:6239/192.168.0.142:7239    #GRP1_RT_B
```

5.8 服务名配置

配置 DM 数据守护，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` 文件中常用配置项目说明：

■ 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。

■ LOGIN_MODE

指定优先登录的服务器模式。0：优先连接 Primary 模式的库，Normal 模式次之，最后选择 Standby 模式；1：只连接主库；2：只连接备库；3：优先连接 Standby 模式的库，Primary 模式次之，最后选择 Normal 模式；4：优先连接 Normal 模式的库，Primary 模式次之，最后选择 Standby 模式。默认值为 4。

■ 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。

■ RW_SEPARATE

指定是否启用读写分离。0 表示不启用读写分离；1 表示启用读写分离，默认值为 0。

■ RW_PERCENT

启用读写分离时，读写分离的分发比例，有效值范围 0~100，默认值为 25。

例如，配置一个名为 dw_svc 的连接服务名，使用 dw_svc 连接数据守护中的数据库，即可实现故障自动重连。

```
dw_svc=(192.168.1.131:5236,192.168.1.132:5236)

LOGIN_MODE =(1)

SWITCH_TIME=(3)

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。

5.9 建库参数说明

数据守护 V4.0 支持 Huge 表的主备数据同步，需要结合下面两个建库参数使用：

表 5.7 dminit 建库参数

配置项	配置含义
HUGE_WITH_DELTA	是否仅允许创建事务型 HUGE 表。取值：1 是，0 否。默认值为 1。
RLOG_GEN_FOR_HUGE	是否生成 HUGE 表 REDO 日志。取值：1 是，0 否。默认值为 0。

如果要搭建支持 Huge 表的数据守护系统，需要将建库参数 HUGE_WITH_DELTA 和 RLOG_GEN_FOR_HUGE 都配置为 1。

6 数据守护使用说明

本章列举了一些常见的数据守护场景及其处理方法，并相应地介绍监视器命令的执行流程。

6.1 正常运行状态

守护系统正常运行时，同一个守护进程组中，只有一个主库，其他的都是备库。

主库处于 Open 状态，主库守护进程也处于 Open 状态，本地没有守护进程控制文件，其内存值是 Valid 有效状态。

所有备库也处于 Open 状态，所有备库守护进程处于 Open 状态，本地没有守护进程控制文件，其内存值是 Valid 有效状态。

主库到所有备库的归档也都处于 Valid 有效状态。

MPP 主备系统中，所有主库的 dmmpp.ctl 都处于一致状态。

6.2 数据守护的启动

Normal 模式的库默认以 Open 状态启动，也可以通过增加启动参数 Mount 将数据库启动到 Mount 状态。而 Primary/Standby 模式的库启动后，自动进入 Mount 状态，因此，数据守护系统启动时，所有数据库实例处于 Mount 状态。所有守护进程处于 Startup 状态。如果实例还未启动到 Mount 状态（比如还处于 After redo 状态），守护进程不会通知实例 Open。

Local 守护类型的守护进程，直接 Open 数据库实例，并修改守护进程状态为 Open。Global 守护类型的守护进程，需要相互协调信息，自动将数据库实例切换到 Open 状态，并将守护进程状态也切换为 Open 状态。

Global 守护类型的守护进程通知本地库 Open 的总体原则：

- 对于备库，如果可加入远程任意一个库，则允许将其 Open；
- 对于主库，如果远程所有库都可加入自己，则允许将其 Open。

还有一些细节条件这里不再具体列出，如果通过监视器没有观察到主库或备库 Open，可以借助监视器的 Check Open 命令查找原因，根据命令返回的原因考虑是否进行人工干

预，比如需要通过监视器命令强制 Open 主库或备库。

手动方式启动数据守护系统时，对于守护进程，数据库实例和监视器的启动顺序没有严格要求，也可以通过监视器命令启动守护系统（前提是所有守护进程已经启动）。

启动流程中，守护进程在通知主库 Open 之前，会先收集出和主库数据一致的备库（备库的 **ALSN** 信息和主库的 **FLSN** 信息相等），守护进程会将这些备库的归档设置为 **Valid** 有效状态，其他数据不一致的备库则设置为 **Invalid** 无效状态。**Primary** 模式数据库实例切换为 **Open** 状态时，需要回滚活动事务、**Purge** 已提交事务，并重构回滚段，会引发数据变化、**LSN** 增长。对归档无效的备库，在数据守护启动完成后，主备库数据肯定是处于不一致状态。



注意：

主库守护进程 Open 主库后，会修改 **INST_RECOVER_TIME** 内存值为 3 秒（默认 60 秒），确保归档状态无效的备库 Open 后，尽快启动故障恢复流程，同步主库数据完成后，重新将归档设置为 **Valid** 状态。

如果在故障恢复流程完成之前，主库故障，并且不存在归档状态有效的备库，则无法执行备库接管；备库强制接管会引发守护进程组分裂。

读写分离集群，在 **Timely** 或 **Realtime** 归档变为 **Valid** 之前，不会在备库上创建数据库连接，只读操作也无法分流到对应的备库。

6.3 强制 Open 数据库

正常情况下，守护进程 **dmwatcher** 可以自动 Open 数据库实例，但某些情况下（比如备库硬件故障无法启动），数据守护系统不满足 6.2 介绍的启动条件，我们可以通过监视器执行 **Open database** 命令强制 Open 数据库实例。主备库都可以强制 Open，其执行流程如下：

假设需要强制 Open 数据库 A，只需要启动一个监视器，登录后输入 **Open database A** 即可完成强制启动。

如果数据库 A 是 **Standby** 模式，强制 Open 的执行流程如下：

1. 通知 A 的守护进程切换为 Open Force 状态
2. 通知 A 执行 Open 操作
3. 通知守护进程切换 Open 状态

如果数据库 A 是 Primary 模式，强制 Open 的执行流程如下：

1. 通知 A 的守护进程切换为 Open Force 状态
2. 修改 A 到所有归档目标的实时归档/即时归档状态为无效
3. 通知 A 执行 Open 操作
4. 通知守护进程切换 Open 状态

Primary 模式数据库实例切换为 Open 状态时，需要回滚活动事务、Purge 已提交事务，并重构回滚段，会引发数据变化、LSN 增长。因此，这个操作可能会引发守护进程组分裂，比如下面的场景：

1. 主库 A 故障
2. 备库 B 接管，成为主库
3. B 故障
4. A 重启，并强制 Open
5. A 和 B 数据不一致，并且无法恢复到一致状态。此时，B 重启，就会产生守护进程组分裂。

强制 Open 主库前，会设置主库到所有归档目标的实时归档/即时归档为 Invalid 状态。



注意：

强制 Open 主库命令，会修改主库守护进程 `INST_RECOVER_TIME` 内存值为 3 秒（默认 60 秒），确保主库 Open 后，尽快启动故障恢复流程，同步主库数据完成后，重新将归档设置为 Valid 状态。

如果在故障恢复流程完成之前，主库故障，将无法执行备库接管；备库强制接管会引发守护进程组分裂。

6.4 关闭数据守护系统

关闭守护系统时，必须按照一定的顺序来关闭守护进程和数据库实例。特别是自动切换模式，如果退出守护进程或主备库的顺序不正确，可能会引起主备切换，甚至造成守护进程

组分裂。

通过监视器执行 Stop Group 命令关闭数据守护系统，是最简单、安全的方式。命令执行成功后，数据库实例正常关闭。但守护进程并没有真正退出，而是将状态切换为 Shutdown 状态。

Stop Group 命令内部流程如下：

1. 通知守护进程切换为 Shutdown 状态
2. 通知主库退出
3. 通知其他备库退出

如果使用手动方式关闭数据守护系统，请严格按照以下顺序执行：

1. 如果启动了确认监视器，先关闭确认监视器（防止自动接管）
2. 关闭备库守护进程（防止重启实例）
3. 关闭主库守护进程（防止重启实例）
4. Shutdown 主库
5. Shutdown 备库

如果是只关闭主库，并且不想引发备库自动接管，有以下两种方法：

方法一：

1. 通过 Detach database 命令将所有备库分离
2. 通过 Stop database 命令退出主库

方法二：严格按照以下顺序执行：

1. 通过 Stop dmwatcher 命令关闭所有守护进程监控
2. 手动正常退出主库

如果是只关闭备库，并且不想引发主库发送日志失败进入 Suspend 状态，请严格按照以下顺序执行：

1. 通过 Detach database 命令将备库分离出数据守护系统
2. 正常退出备库（手动退出或者通过 Stop database 命令退出）

关闭整个数据守护系统时，先关闭主库再关闭备库，顺序一定不能错。对于本地守护类型的库，在关闭数据守护系统时，不受此顺序限制。



注意：

因为主库 Shutdown 过程中，需要 Purge 所有已提交事务，会修改数据，并产生 Redo 日志。如果先 Shutdown 备库，会导致主库发送归档日志失败，并且由于主库已经处于 Shutdown 状态，会导致主库异常关闭。

■ 关闭后升级数据守护系统

当正常关闭数据守护系统后，可以将守护系统主备库都升级到新版本然后正常启动数据守护系统以达到升级整个数据守护系统版本的目的。如果需要不中断数据库服务升级数据守护系统，可使用滚动升级功能，具体可参看 6.19 节[滚动升级](#)。

升级版本后重新启动数据守护系统时，由于升级过程中可能涉及系统表、动态视图等的修改，需要在主库版本升级后将升级过程中的 Redo 日志发送到备库，备库通过重做这段 Redo 日志进行版本升级。在此过程中，不允许对备库进行访问，否则会报“备库版本升级时不允许访问”的错误。Redo 日志重做完成后，备库允许正常访问。

6.5 主备库切换

主库维护，滚动升级等场景，可以执行 Switchover 命令，实现主备库切换。如果存在多个备库，需要先执行 Choose Switchover 命令，选出守护进程组中可以切换的备库。

Choose Switchover 命令选择可切换备库的条件如下：

1. 主库守护进程是 Open 状态
2. 备库守护进程是 Open 状态
3. 主、备库的 OPEN 记录项内容相同，并且守护进程控制文件是 Valid 有效状态（内存值）
4. 主库正常运行
5. 备库正常运行
6. 主库处于 Open 状态
7. 备库处于 Open 状态
8. 主库到备库的归档是 Valid 状态

假定选出的可切换备库是 B，Switchover 切换流程如下：

1. 通知主备库守护进程，切换为 Switchover 状态
2. 通知主库 (A) Mount
3. 实时或 MPP 主备环境下，通知备库 (B) APPLY KEEP_RLOG_PKG
4. 通知备库 (B) Mount
5. 通知 (A) 切换为 Standby 模式
6. MPP 主备环境下，通知 (A) 修改 MPP_INI 内存值为 0
7. 通知 (B) 切换为 Primary 模式
8. 通知 (B) 修改所有归档目标的归档状态为无效
9. MPP 主备需要通知各组活动主库更新 dmmpp.ctl 文件，参考后文说明
10. 通知新的备库 (A) Open
11. 通知新的主库 (B) Open
12. 通知主备库守护进程切换为 Open 状态
13. 清理所有守护进程上记录的监视器命令执行信息

其中第 9 步，MPP 主备环境下更新 dmmpp.ctl 步骤说明如下：

1. 收集各组的活动主库信息，构造新的 dmmpp.ctl 文件。

主库收集条件：

- 1) 组中只有一个活动主库，不存在多个 OPEN 主库
- 2) 主库守护进程不是 Error 状态，且控制文件是有效状态
- 3) 主库实例不是 Error 状态

2. 通知步骤 1 中收集到的主库更新 dmmpp.ctl 文件

首先通知执行 Switchover 命令的守护进程组的主库执行更新操作，再通知其他组的主库执行更新操作。通知其他组执行更新操作时，会先将其主库的守护进程切换为 Mppctl update 状态，更新完成后，再切换回 Open 状态。

对步骤 1 中不符合收集条件的组，在步骤 2 中跳过不再处理，后续待其主库恢复正常后，可借助 recover mppctl 命令恢复 dmmpp.ctl 文件到一致状态。

更新 MPP 控制文件时，服务器自动断开当前所有的会话连接，回滚未提交事务，挂起所有工作线程，该过程可能会持续一段时间。

3. 根据步骤 2 的结果决定是否继续执行 Switchover 的后续步骤

步骤 2 中如果某个活动主库更新失败，则终止切换，执行失败。

主备库切换在实现逻辑上等同于主备库正常状态下用户主动发起的 **Takeover** 操作。**Switchover** 完成后，主备库之间数据是不完全同步的，要由新主库 B 的守护进程通过 **Recovery** 流程，重新同步数据到新备库 A。



Switchover 命令会修改切换后主库守护进程 **INST_RECOVER_TIME** 内存注意：值为 3 秒（默认 60 秒），确保尽快启动故障恢复流程，同步主库数据完成后，重新将归档设置为 **Valid** 状态。在故障恢复流程完成之前，再次执行 **Switchover** 命令会报错，如果主库故障，备库接管将会报错；备库强制接管会引发守护进程组分裂。

6.6 主库故障、备库接管

当出现硬件故障（掉电、存储损坏等）原因导致主库无法启动，或者是主库内部网卡故障导致主库短期不能恢复正常的情况下，可使用备库接管功能，将备库切换为主库继续对外服务。

故障自动切换模式下，确认监视器会自动选择符合条件的备库进行接管。

故障手动切换模式下，可以先在监视器上执行 **Choose Takeover** 命令，选出守护进程组中可以接管的备库。

为了避免备库接管后，造成守护进程组分裂，执行 **Takeover** 必须满足下列条件：

1. 主库是 **Primary** 模式、**Open** 状态时，发生故障
2. 主库守护进程故障，故障前是 **Startup/Open/Recovery** 状态；或者主库守护进程正常
3. 主库故障前到接管备库的归档状态为 **Valid**
4. 接管备库是 **Standby** 模式、**Open** 状态
5. 接管备库的守护进程控制文件状态为 **Valid**（内存值）
6. 故障主库和接管备库的 **Open** 记录项内容相同

假设主库 A 故障时，在故障自动切换模式下确认监视器自动选出待接管备库 B 并通知备库 B 自动接管，或者在故障手动切换模式下，通过监视器上的 **Choose Takeover** 命令，选出待接管备库 B，在监视器上输入 **Takeover** 命令通知备库 B 执行接管，这两种方式的

接管执行流程是一样的。

以备库 B 为例，接管的执行流程包括：

1. 监视器通知守护进程 (B) 切换为 Takeover 状态
2. 实时主备或 MPP 主备环境下，通知备库 (B) `APPLY KEEP_PKG`
3. 通知备库 (B) Mount
4. 通知 (B) 切换为 Primary 模式
5. 通知 (B) 修改到所有归档目标的归档状态为 Invalid
6. MPP 主备需要通知活动主库更新 `dmmppl.ctl` 文件 (步骤参考 6.5 [主备库切换](#))
7. 通知新的主库 (B) Open
8. 通知守护进程 (B) 切换为 Open 状态

例如，主库 C 故障，备库 B 接管。执行 `Takeover` 命令后，会修改守护进程 (B) 的 `INST_RECOVER_TIME` 内存值为 3 秒 (默认 60 秒)，确保尽快启动 B->C



注意：

的故障恢复流程，同步主库数据完成后，重新将 B->C 的归档设置为 Valid 状态。在故障恢复流程完成之前，主库 (B) 故障，备库 (C) 无法接管；强制备库 (C) 接管会引发守护进程组分裂。

6.7 备库强制接管

有些情况下，备库接管会失败，但主库不能启动或者及时恢复对外服务的情况下，可以使用 `Takeover Force` 命令，进行备库强制接管。强制接管具有一定的风险，可能导致备库和故障主库数据不一致，而造成部分数据的丢失，出现数据库分裂的情况，所以应该慎重使用。

例如主库和守护进程故障时，监视器未启动，用户启动监视器后，由于监视器并未收到故障主库任何信息，因此不满足 `Takeover` 条件，执行 `Takeover` 会报错。如果用户可确认主库故障时主备数据是一致的 (如故障时主库未执行操作，主备库归档有效的，并且两者的 LSN 一致)、或者丢失小部分数据的影响可忽略、或者丢失小部分数据的影响小于主库持续宕机造成的影响，则可以考虑执行 `takeover force` 强制接管。

强制接管，先通过 `Choose Takeover Force` 选出符合强制接管条件的备库，再执行 `Takeover Force` 命令。备库强制接管时，如果接管备库是处于 Mount 状态/Standby

模式的库，则会自动 Open 备库，其他执行流程与备库接管一致。强制接管的条件包括：

1. 不存在活动主库
2. 备库守护进程处于 Open 或 Startup 状态
3. 备库实例运行正常
4. 备库是 Standby 模式
5. 备库处于 Open 或 Mount 状态
6. 备库的 KLSN 必须是所有备库中最大的
7. 备库守护进程控制文件必须有效

与 **Takeover** 命令一样，**Takeover Force** 命令会修改主库守护进程的 **INST_RECOVER_TIME** 内存值为 3 秒（默认 60 秒），确保尽快启动故障恢复流程。

强制接管具有一定的风险，可能导致备库和故障主库数据不一致，从而造成部分数据的丢失、数据库分裂，所以应该综合考虑当时情况，慎重使用。



注意：对于强制接管并且引发分裂的场景，故障主库重启恢复后，只有当新接管的主库处于 **Primary & Open**，并且实例是活动情况下，才会主动设置原来的主库为 **Split** 分裂状态。如果新接管主库也被重启到 **Mount** 状态，由于两个主库互相不可加入，守护进程无法在两个 **Mount** 主库之间选出有效主库，需要用户干预。

6.8 主库故障重启（备库接管前重启）

主库故障后立即重启，此时主库的守护进程变成 Startup 状态，重新进入守护进程的启动流程，将数据一致的备库归档设置为有效状态，其余备库归档设置成无效状态，并重新 Open 主库。Open 成功后继续作为主库，当检测到归档状态无效的备库正常时会启动 Recovery 处理流程，重新同步主备库数据。

6.9 备库故障处理

备库产生故障（硬件故障或者内部网卡故障）时，主库的处理流程对手动切换、自动切换模式处理上有些差异。

■ 手动切换模式

对于手动切换模式，检测到备库故障，满足 Failover 条件时，主库的守护进程立即切换到 Failover 状态，执行对应的故障处理，如果不满足切换 Failover 条件，则保持当前状态不变。

手动切换模式下，主库守护进程切换 Failover 条件：

1. 备库实例故障，或者主备库之间出现网络故障，或者备库重演时校验 LSN 不匹配，这三种场景下引发主库同步日志到备库失败挂起，主库实例处于 Suspend 状态
2. 主库到此备库的归档状态是 Valid（读写分离集群没有此限制）
3. 主库的守护进程处于 Startup、Open 或 Recovery 状态
4. 当前没有监视器命令正在执行

■ 自动切换模式

对于自动切换模式，主库的守护进程会自动判断切换到 Failover 状态或者 Confirm 确认状态，如果两种状态切换条件都不满足，则保持当前状态不变。

自动切换模式下，主库守护进程不进入 Confirm 确认状态，直接切换到 Failover 条件：

1. 前四项条件，和上面列出的手动切换条件相同
2. 备库实例故障，备库守护进程正常

如果只满足条件 1，不满足条件 2，则主库守护进程会先进入 Confirm 确认状态，等待确认监视器的确认消息。主库的守护进程进入 Confirm 确认状态后，会有下面几种不同的处理：

1. 主库和确认监视器之间网络连接正常

主库的守护进程收到了确认监视器返回的确认消息，如果确认监视器认定可以执行 Failover，则主库的守护进程会切换为 Failover 状态并执行对应的处理；如果确认监视器认定不满足执行 Failover 条件，则主库的守护进程会一直保持在 Confirm 状态。确认监视器认定主库可以执行 Failover 条件：

- 1) 主库守护进程处于 Confirm 状态
- 2) 主库实例正常，处于 Suspend 状态
- 3) 主库没有被接管，不存在其他主库
- 4) 没有 takeover/switchover 命令在执行
- 5) 当前所有归档有效的备库均可以加入主库

2. 主库和确认监视器之间网络连接异常, 或者没有启动确认监视器。满足下面条件后主库允许切换至 Failover 状态执行故障处理:

- 1) 主库实例正常, 处于 Suspend 状态
- 2) 备库守护进程正常
- 3) 主库没有被接管, 不存在其他主库
- 4) 没有 takeover/switchover 命令正在执行
- 5) 备库故障前可以加入主库

3. 主库和确认监视器网络恢复正常后, 主库已经被接管。老主库的守护进程切换为 Startup 状态, 重新判断是否可加入新主库。

主库守护进程进入 Failover 状态后的执行流程 (自动或手动切换模式下执行流程相同):

- 1) 对实时主备或 MPP 主备, 通知主库修改发送归档失败的备库归档状态无效
- 2) 通知主库重新 Open。
- 3) 将主库的守护进程切换为 Open 状态

6.10 公共网络故障

主库公共网络故障, 主备库正常、内部网络正常情况下, 用户无法连接主库执行正常的数据库操作。这种情况下, 用户可以通过 Switchover 命令, 将备库切换为主库, 确保数据库服务不受影响。

读写分离集群中备库公共网络故障, 主备库正常、内部网络正常情况下, 用户的所有请求会自动分发到主库执行。

6.11 内部网络故障

DM 数据守护对内部网络的可靠性提出了很高的要求, 但是在实际应用中 (比如异地容灾), 存在很多不可控因素, 内部网络无法保证绝对可靠。守护进程和守护进程之间、守护进程和监视器之间通过超时机制来检测是否出现故障, 当内部网络出现故障时, 超过设置时间未收到远程消息, 会认定远程故障。

下面的表格说明在外部网络正常时, 内部网络故障的场景以及守护进程的处理策略:

表 6.1 故障场景和处理策略

故障类型	手动切换处理策略	自动切换处理策略
主库内部网卡故障	归档失败时，主库挂起，主库 Failover，归档设置无效，异步工作。	归档失败时，主库挂起，主库守护进程保持 Confirm 状态，确认监视器自动寻找备库自动接管。详见 4.2、4.3、6.6 节。 因为外网还是正常的，在经过 SESS_FREE_IN_SUSPEND 参数配置的时间后，老主库会自动断开所有连接，以便应用连接能够转移到新接管的主库上，详见下文说明。 注：此场景下主备库之间无法通信，但确认监视器仍然可以和备库的守护进程正常通信。
备库内部网卡故障	归档失败时，主库挂起，主库 Failover，归档设置无效，异步工作。	归档失败时，主库挂起，主库守护进程 Failover(经过 Confirm 状态确认后 Failover)，归档设置无效，异步工作。详见 4.2、4.3、6.9 节。 注：此场景下主备库之间无法通信，但确认监视器仍然可以和主库的守护进程正常通信。
主备库交换机故障	归档失败时，主库挂起，主库 Failover，归档设置无效，异步工作。	归档失败时，主库挂起，主库守护进程保持 Confirm 状态，备库不会自动接管。 注：此场景下主备库之间，以及和确认监视器之间都无法正常通信。
主备库网卡同时故障	归档失败时，主库挂起，主库 Failover，归档设置无效，异步工作。	归档失败时，主库挂起，主库守护进程保持 Confirm 状态，备库不会自动接管。 注：此场景下主备库之间，以及和确认监视器之间都无法正常通信。

自动切换模式下，当发生主库内部网络故障而其他网络均正常时，归档失败，主库 Suspend，会导致连接主库的会话被挂起，一直无法返回，备库自动接管后，这些会话连接无法自动切换到新主库上。为此，DM 采用超时机制断开由于归档失败导致挂起时的所有会话，通过配置 dm.ini 中的 SESS_FREE_IN_SUSPEND 实现，默认 60s。

网络故障恢复后，对于手动切换模式，没有强制接管的情况下，主库的守护进程会自动判断备库是否能加入，能加入则启动 Recovery 流程。对于自动切换模式已经被接管的情况，守护进程会将接管的老主库强制退出 (Shutdown)，老主库重启后，还需要守护进

程再去判断是否可作为备库重加入守护系统，如果可以，则启动恢复流程（切换为 Standby 模式，重新将实例 Open），如果不能，才修改为分裂状态。

当内部网络是由于拔掉网线、或者禁用网卡等原因导致的故障，MAL 系统检测到链路断开的 时间 由 dmmal.ini 中 配置 的 MAL_CHECK_INTERVAL 以及 MAL_CONN_FAIL_INTERVAL 决定。会话的中断时间会比杀掉实例、掉电等花费的时间长。下面举例说明：

dmmal.ini 配置

```
MAL_CHECK_INTERVAL      = 10  
  
MAL_CONN_FAIL_INTERVAL = 10
```

dmwatcher.ini 配置

```
DW_ERROR_TIME           = 10
```

拔掉备库网线后，主库上 MAL 检测线程根据 MAL_CHECK_INTERVAL 间隔检查，时间 $\leq 10s$ ，检查认定网络故障需要 10s，再加上守护进程认定远程网络故障时间 DW_ERROR_TIME 为 10s，在最坏的情况下，应用连接等待主库响应最长的时间可能会是三个时间的累加，也许能达到 30s。

6.12 备库异常处理

备库实例正常，只是内部网络出现异常，比如网络不稳定、网速急速下降，或者备库自身的软硬件出现问题，比如操作系统原因或磁盘读写速度异常降低等情况下，备库响应主库的时间变慢，这种情况下，主库守护进程会自动进行检测处理，具体的异常处理流程详见 [3.1.7 备库异常处理](#)。

6.13 故障库数据同步

当实例由于各种原因故障，重启或者网络恢复后，守护进程会自动判断能否加入当前主库系统，判断条件和数据恢复处理流程详见 [3.1.9 故障恢复处理](#) 章节。恢复的中间过程可通过附录中的动态视图进行查看。

6.14 备库重建

当主备库所在硬件出现故障、或者数据库文件损坏导致数据损毁时，可按照第 7 章 数据守护搭建中介绍的几种方式来重新准备数据。

重建库要选定一个当前有效库或者一个历史备份。下面以单节点备库 B 数据损坏，重新配置数据为例，说明备库重建流程：

1. 重新初始化数据库

```
./dmrman path=/dm/data
```

2. 启动 DISql 登录主库 A，执行联机备份

```
SQL> BACKUP DATABASE FULL BACKUPSET '/dm/data/BACKUP_FILE_01';
```

注意，如果主库 dm.ini 中的 USE_AP 设置为 1，则需要先启动 dmap 服务。

3. 将备份文件拷贝至 B 所在的机器上，执行脱机还原与恢复

```
./dmrman CTLSTMT="RESTORE DATABASE '/dm/data/DAMENG/dm.ini' FROM BACKUPSET
'/dm/data/BACKUP_FILE_01'"
./dmrman CTLSTMT="RECOVER DATABASE '/dm/data/DAMENG/dm.ini' FROM BACKUPSET
'/dm/data/BACKUP_FILE_01'"
./dmrman CTLSTMT="RECOVER DATABASE '/dm/data/DAMENG/dm.ini' UPDATE DB_MAGIC"
```

4. 重新配置 B 的 dm.ini、dmmal.ini、dmarch.ini 和 dmwatcher.ini 配置文件

5. 以 Mount 方式启动 B

```
./dmserver /dm/data/DAMENG/dm.ini mount
```

6. DISql 登录 B，设置 OGUID，修改备库模式

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);
SQL>sp_set_oguid(82379);
SQL>alter database standby;
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

7. 启动 B 的守护进程

```
./dmwatcher /dm/data/DAMENG/dmwatcher.ini
```

执行以上步骤后，恢复 B 的准备过程已经完成。接下来，数据守护系统会将 B 作为备库重加入数据守护系统，A 的守护进程会自动通知同步数据到 B，最终恢复主备库数据到一

致状态。

如果数据规模比较大、联机备份耗时较长、应用压力比较大的情况下，主库联机备份、备库脱机还原过程中，主库可能又新产生了大量归档日志。使用上述步骤重建备库后，主库向备库同步历史数据的时间会比较久，主备库数据会在比较长的一段时间内处于不一致状态。对这种情况，用户可以通过归档备份、还原和归档恢复功能，将备库数据恢复到更加接近主库的最新状态，有效减少备库加入主备系统后的历史数据同步时间。执行步骤如下：

1. 重新初始化数据库

```
./dminit path=/dm/data
```

2. 启动 DISql 登录主库 A，执行联机备份

```
SQL> BACKUP DATABASE FULL BACKUPSET '/dm/data/BACKUP_FILE_02' WITHOUT LOG;
```

注意，如果主库 dm.ini 中的 USE_AP 设置为 1，则需要先启动 dmap 服务。

3. 将备份文件拷贝至 B 所在的机器上，执行脱机还原

```
./dmrman CTLSTMT="RESTORE DATABASE '/dm/data/DAMENG/dm.ini' FROM BACKUPSET  
'/dm/data/BACKUP_FILE_02'"
```

4. 启动 DISql 登录主库 A，查看备份集的 BEGIN_LSN，并从 BEGIN_LSN 开始备份主库的归档日志。（这里假设查询到的备份集 BEGIN_LSN 是 38491）

如果备份集不是在默认备份路径下，则需要先将备份集所在目录添加进来，否则会查询不到备份集信息：

```
SQL> SELECT SF_BAKSET_BACKUP_DIR_ADD('DISK','/dm/data');
```

查询备份集信息并执行备份：

```
SQL> SELECT BACKUP_NAME,BEGIN_LSN FROM V$BACKUPSET;
```

```
SQL> BACKUP ARCHIVE LOG FROM LSN 38491 BACKUPSET '/dm/data/ARCH_BAK_02';
```

5. 将归档备份集拷贝至 B 所在的机器，还原归档后、利用这些归档日志进行数据库恢复

```
./dmrman CTLSTMT="RESTORE ARCHIVE LOG FROM BACKUPSET '/dm/data/ARCH_BAK_02' TO  
ARCHIVEDIR '/dm/data/arch'"
```

```
./dmrman CTLSTMT="RECOVER DATABASE '/dm/data/DAMENG/dm.ini' WITH ARCHIVEDIR  
'/dm/data/arch'"
```

```
./dmrman CTLSTMT="RECOVER DATABASE '/dm/data/DAMENG/dm.ini' UPDATE DB_MAGIC"
```

6. 重新配置 B 的 dm.ini、dmmal.ini、dmarch.ini 和 dmwatcher.ini 配置文

件

7. 以 Mount 方式启动 B

```
./dmserver /dm/data/DAMENG/dm.ini mount
```

8. DISql 登录 B, 设置 OGUID, 修改备库模式

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);
SQL>sp_set_oguid(82379);
SQL>alter database standby;
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

9. 启动 B 的守护进程

```
./dmwatcher /dm/data/DAMENG/dmwatcher.ini
```

6.15 守护进程组分裂

组分裂的概念和导致分裂的场景可参考 2.2.15 [组分裂](#)一节, 对于分裂的数据库需要进行人工干预, 重新准备数据后, 才允许重新加入守护系统。

借助监视器的 show 命令、或者 tip 命令可以查看守护系统中是否发生数据库的分裂, 对于已发生的分裂, 可以借助以下方法找出分裂产生的原因:

1. 查看分裂库的服务器和守护进程 log 日志, 查找带有 “[!!!” 和 “[!!!]” 标签的 log 信息, log 信息格式形如 “[!!! LOG_INFO !!!]”, 记录有数据库分裂的详细原因。

2. 如果分裂库的守护进程控制文件状态不是 Valid, 可借助监视器的 show open info 命令, 根据 DESC 字段找出原因。

3. 根据服务器、守护进程和监视器的 log 日志, 找出历史操作信息, 分析产生分裂的原因。

发生分裂后, 用户需要选择适当的主库作为最新主库, 备库采用重建的方法加入系统。

6.16 dmmppctl 不一致

MPP 主备环境下, 在某些故障场景中, 可能会出现 dmmppctl 控制文件不一致的情况, 可以借助监视器的 check mppctl 和 recover mppctl 命令来检测和恢复。

正常运行时, 所有 MPP 主节点的 dmmppctl 文件内容是完全一致的。如果某个组中

的主备库实例发生改变（如执行 `switchover`、`takeover` 等），导致 `dmmppctl` 变化，需要及时同步到所有组的主库节点上。

在执行数据同步过程中如果发生故障，导致 `dmmppctl` 未同步，就可能造成控制文件的不一致。例如，两节点 MPP，分成两个守护进程组（GRP1 和 GRP2），组 GRP1 包含两个数据库（主库 GRP1_MPP_EP01 和备库 GRP1_MPP_EP11），组 GRP2 包含两个数据库（主库 GRP2_MPP_EP02 和备库 GRP2_MPP_EP22）。在监视器上执行 `switchover` 命令，将 GRP1 的两个数据库进行切换。

当执行到 `switchover` 流程中更新 `dmmppctl` 文件时，可能会出现下列故障情况，导致 `dmmppctl` 处于不一致状态。

1. 更新 GRP1 的 `dmmppctl` 文件时，监视器故障或者链路断开等原因导致更新失败，GRP1 和 GRP2 的 `dmmppctl` 文件都没有更新，但 GRP1 的主库已经切换为 GRP1_MPP_EP11。

2. 更新 GRP1 的 `dmmppctl` 文件成功，但更新 GRP2 的 `dmmppctl` 失败，GRP1 和 GRP2 的 `dmmppctl` 文件不一致。

这两种不一致状态可以通过 `recover mppctl` 命令重新恢复到一致状态。

监视器执行 `recover mppctl` 的步骤如下：

1. 生成新文件

根据当前守护系统中所有组的活动主库信息，重新生成有效的 `dmmppctl` 文件。

2. 检查文件一致性

检查新生成的文件和每个组当前的 `dmmppctl` 文件是否一致，如果一致则无需恢复。

3. 文件发送

将新生成的 `dmmppctl` 文件内容依次发送给所有活动主库的守护进程，发送之前将守护进程修改为 `Mppctl update` 状态。再由守护进程转发至主库，主库收到后重新写入到本地的 `dmmppctl` 文件。

如果存在主库故障还没有处理时，`recover mppctl` 会跳过此故障组，只恢复存在有效主库组的 `dmmppctl` 文件。故障组主库恢复正常后，需要重新执



注意：行 `recover mppctl` 进行恢复。`dmmppctl` 文件不一致时，MPP 无法正常提供数据库功能。

6.17 确认监视器未启动

故障自动切换模式下，确认监视器必须一直处于启动状态，并且要确保确认监视器的配置正确（否则无法和守护进程通信，相当于确认监视器没有启动）。若确认监视器未启动或者配置错误的，在主库或备库发生故障时，无法通过确认监视器执行正常的故障处理。确认监视器的具体作用请参考状态确认和自动接管小节。

在没有确认监视器或确认监视器配置错误的情况下：

1. 如果主库故障，则备库无法自动接管为新主库。
2. 如果备库实例和备库的守护进程都出现故障，或者主备库之间出现网络故障，则主库的守护进程无法通过确认监视器来确认备库状态，主库守护进程会处于 Confirm 确认状态，实例处于 Suspend 挂起状态。

对于第 2 种情况，当备库实例和备库守护进程重新恢复正常，或者主备库之间的网络恢复正常后，并且主库守护进程 DW_FAILOVER_FORCE 配置为 1 时，即使没有确认监视器，主库的守护进程也会切换至 Failover 状态，将主库重新 Open，后续进行正常的备库恢复处理，而不会一直处于 Confirm 确认状态。

可以通过监视器的 show monitor 命令查看连接到指定守护进程的所有监视器信息，以此可以检查守护系统中是否启动有确认监视器以及确认监视器和守护进程的通信是否正常，也可以尝试再启动一个新的确认监视器，如果守护系统中已经启动有确认监视器，则不允许再启动第二个。

6.18 备库维护

守护系统正常运行时，如果需要主动维护备库，并且不启动主库的 Failover 故障处理流程（可参考 [6.9 备库故障处理](#)），可以通过监视器命令调整主库守护进程的恢复时间间隔，并主动将备库归档失效来达到目的。

监视器提供 detach database 命令支持备库的主动维护，在备库维护完成后，可再通过 attach database 命令将备库加回到守护系统。如果在维护备库时需要将备库退出，可借助监视器的 stop database 命令来完成，维护完成后再通过 startup database 命令将备库重新启动。

具体的命令用法说明请参考“[4.4 监视器命令](#)”小节说明。

假如需要动态维护的备库名称为 GRP1_RT_02，完整的维护步骤说明如下：

1. 在监视器上执行 login，输入登录口令
2. 在监视器上执行 detach database GRP1_RT_02，将备库分离出守护系统。
3. 在监视器上执行 stop database GRP1_RT_02，将备库正常退出，如果不需要退出备库，则需要执行 stop dmwatcher database GRP1_RT_02 关闭备库的守护进程监控功能，避免将备库分离成功后，主库又发生切换，导致之前的分离操作失效，又将备库重新加回守护系统。
4. 执行具体的备库维护操作。
5. 备库维护完成后，如果是关闭状态，在监视器上执行 startup database GRP1_RT_02 将备库重新启动，如果已经是运行状态，根据情况看是否需要执行 startup dmwatcher database GRP1_RT_02 打开备库的守护进程监控功能。
6. 待备库重新 Open 后，在监视器上执行 attach database GRP1_RT_02，将备库重新加回到守护系统中。

以上步骤中，在步骤 3 执行完成后，会将备库的守护进程切换到 Shutdown 状态，备库维护完成后，如果是用手工方式重启，需要先使用 startup dmwatcher 命令将备库的守护进程监控功能打开，备库才能自动 Open。如果是用 startup database 命令方式重启，此命令会自动将守护进程的监控功能打开。

需要注意的是，如果在分离出去的备库维护完成之前，主库故障，其他备库接管成为新主库，则之前的分离操作会失效，可以通过对备库的守护进程执行 stop dmwatcher database 关闭指定备库的监控，避免这种情况发生，或者重新对此备库执行 detach 命令，通知当前的新主库不去主动恢复正在维护中的备库。

6.19 滚动升级

数据守护 V4.0 可以确保在不中断数据库服务的情况下，实现滚动升级。

这里以一主一备的配置方式举例说明，如图 6.1 所示。

假设初始状态为：主库 DM1 部署在 DW_P 机器上，备库 DM2 部署在 DW_S 机器上，监视器部署在 DW_M 机器上，滚动升级操作步骤如下：

1. 关闭备库 DM2 的守护进程，正常关闭备库实例 DM2；这个过程中，主库 DM1 的守护进程会检测到备库故障，并进行故障处理，修改归档状态无效。

2. 升级 DW_S，重新安装新版本服务器，或者直接将新的执行程序 and 动态库替换旧版本。
3. 启动 DM2 实例，启动 DM2 的守护进程。
4. 等待历史数据同步完成，DM2 重新加入数据守护系统（可以通过监视器 DW_M 查看数据守护系统是否恢复正常）。
5. 关闭主库 DM1 的守护进程，正常关闭实例 DM1；使用监视器的接管命令将 DM2 接管为新的主库。如果配置了自动接管，因为老主库正常退出之后，默认配置（监视器的 MON_TAKEOVER_SHUTDOWN 配置为 0）无法触发自动接管，所以只能手动干预；如果 MON_TAKEOVER_SHUTDOWN 非 0，主库正常退出后，超过设定的时间主库仍未重启，备库可以完成自动接管。
6. 升级 DW_P，重新安装新版本服务器，或者直接将新的执行程序 and 动态库替换旧版本。
7. 启动 DM1 实例，启动 DM1 的守护进程。
8. 等待历史数据同步完成，DM1 作为备库重新加入数据守护系统（可以通过监视器 DW_M 查看数据守护系统是否恢复正常）。
9. 在 DW_M 监视器控制台上输入 Switchover 命令，重新将 DM1 切换为主库，至此滚动升级完成。

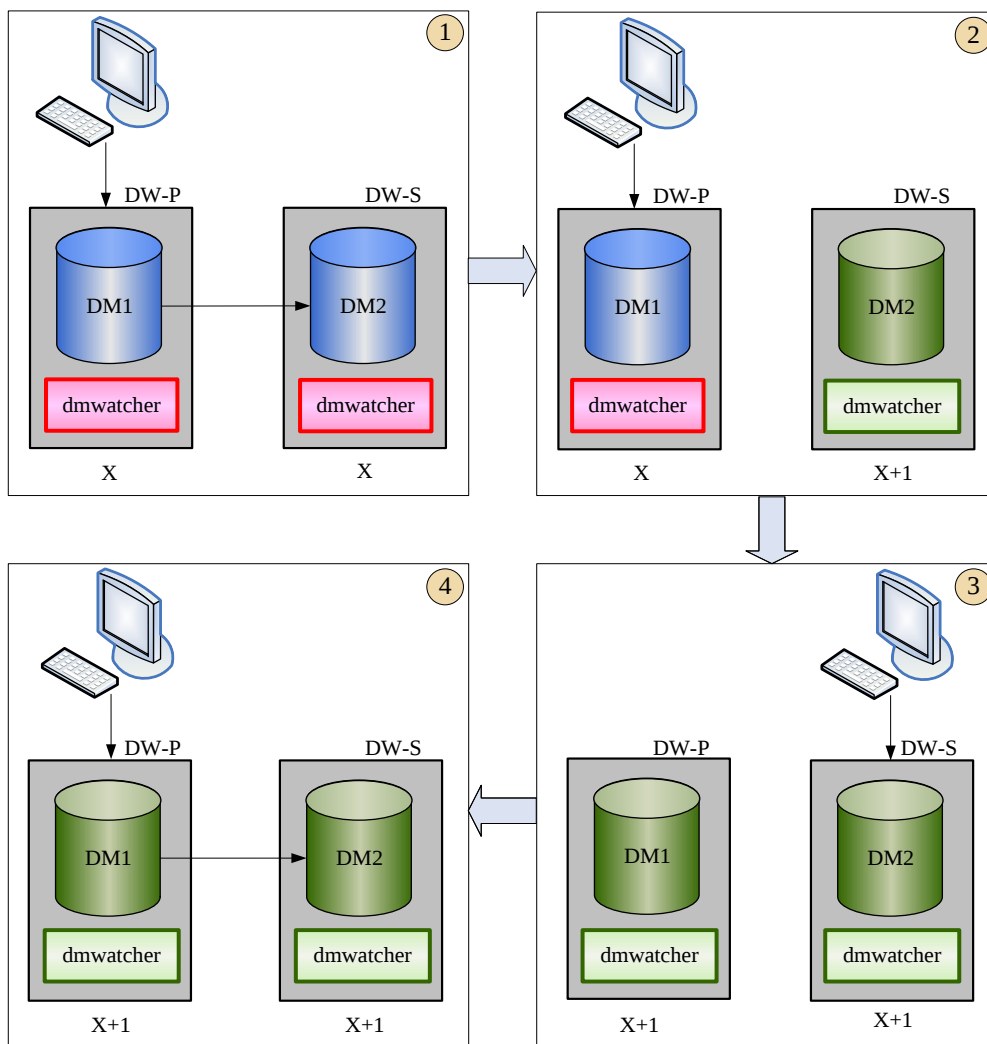


图 6.1 滚动升级

多个备库的情况，可同样参考上面的步骤 1 到步骤 4，先统一升级所有备库实例，步骤 5 中，可任意选择一个升级后的备库接管，最后升级主库完成后，再借助 `switchover` 命令切换主备库。

6.20 实时/读写分离/MPP 备库数据同步情况分析

正常情况下实时/读写分离/MPP 备库数据与主库数据基本一致，可以通过监视器的 `show` 命令观察主备库的 LSN 差距来大致了解数据同步情况，如果主备库的 LSN 值差距很小，说明主备库数据基本一致。当备库的 LSN 值远小于主库 LSN 值时，就需要引起重视，分析原因。

主备库 LSN 值相差很大的情况下，可以借助监视器的 `show` 命令查看状态来进行分析，下面的分析以守护系统中存在正常主库为前提，如果主库发生故障，是无法再向备库同步数

据的，需要等主库恢复正常后，再观察主备库的 LSN 同步情况。

1. 备库的守护进程状态是否正常（WSTATUS 字段）

如果 WSTATUS 是 Error 状态，说明监视器接收备库的守护进程消息超时，需要查看守护进程是否故障。如果发生故障，则需要根据守护进程日志分析故障原因，待守护进程恢复正常后再观察备库的 LSN 增长情况。

如果 WSTATUS 是 Shutdown 状态，说明备库的守护进程监控被关闭，需要借助监视器的 startup dmwatcher 命令打开监控，打开之后再观察备库的 LSN 增长情况。

2. 备库实例状态是否正常（INST_OK 字段）

如果 INST_OK 字段是 Error 状态，说明备库的守护进程接收备库实例消息超时，需要确认备库实例是否故障。如果发生故障，则需要根据服务器日志分析故障原因，待备库恢复正常后，再观察 LSN 增长情况。

3. 备库归档状态是否有效（RSTAT 字段）

1) 备库的 RSTAT 字段是 Invalid 状态

需要等主库的守护进程发起 Recovery 流程，同步主备库数据。

使用监视器的 show 命令，查看主库的守护进程状态，如果已经处于 Recovery 状态，可以使用 DISql 登录主库查询 V\$RECOVER_STATUS 视图，查看当前的恢复信息。

如果主库的守护进程不是 Recovery 状态或者 V\$RECOVER_STATUS 视图中没有查到备库的恢复信息，则需要根据 [3.1.9 故障恢复处理](#) 描述的 Recovery 条件来判断备库是否满足恢复条件或者借助监视器的 check recover 命令来查找原因，如果不满足条件，则需要根据不满足的情况来做不同的处理。

2) 备库的 RSTAT 字段是 Valid 状态

如果备库归档处于有效状态，并且备库实例是 Open 状态，主库也是正常 Open 状态，则说明主备库在正常同步数据，可以通过监视器的 show 命令查看备库的 LSN 是否在持续增长，以及备库重演信息中的 N_TASK/TASK_MEM_USED 字段值来判断备库上是否有较多的日志堆积。

可以登录备库查询 V\$RAPPLY_SYS/V\$RAPPLY_LOG_TASK 视图来查看日志的堆积情况，备库提供有一系列的参数来控制日志堆积，具体请参考 dm.ini 小节说明。

3) 备库的 RSTAT 字段是 Unknown 状态

备库需要从对应的主库上获取自己的归档状态，如果备库找不到自己的主库（没有收到过主库同步过来的日志，或者所在组中不存在活动主库，或者备库无法加入到活动主库中），

则无法得知主库到自己的归档状态信息，显示为 Unknown。此时需要用户去确认是否主库故障，或者备库不满足 Recovery 条件，或者是备库已经分裂出去的情况。

6.21 异步备库数据同步情况分析

源库（主库或者异步备库）通过配置的定时器来定时同步数据到异步备库，可以根据定时器的时间间隔，使用监视器的 show 命令来观察异步备库的 LSN 是否在定期增长，或者使用监视器的 show arch send info 命令来查看源库到异步备库的数据同步情况。

如果这两种方式都没有观察到源库到异步备库的数据同步，则需要从以下几个方面来分析原因：

1. 源库是否配置正确

- 1) 源库的 dmarch.ini 中是否有异步归档配置项，定时器名称是否配置 (ARCH_TIMER_NAME)，并且需要和 dmtimer.ini 中的定时器名称一致。
- 2) 源库的数据文件目录下是否有定时器配置文件 dmtimer.ini。
- 3) 源库的 dm.ini 中的 TIMER_INI 是否配置为 1。
- 4) 源库的 dmmal.ini 中是否有异步备库的配置项，并且守护系统中所有实例的 dmmal.ini 的内容完全一致。
- 5) 源库的 dmtimer.ini 中 IS_VALID 是否配置为 1。

可使用 DISql 登录源库，查询 V\$DM_ARCH_INI 和 V\$DM_TIMER_INI 视图来确认配置信息是否正确。

2. 源库和异步备库状态是否正常

通过监视器的 show 命令查看源库和异步备库状态，如果源库或异步备库发生故障，则不会再同步数据，需要等两者都恢复正常后，再观察异步备库的数据同步情况。

3. 源库到异步备库的 mal 链路是否正常

通过监视器的 show arch send info 观察源库到异步备库的 MAL 链路状态，如果处于 Disconnected 状态，也无法正常同步数据，需要等链路重建成功后，再观察异步备库的数据同步情况。

4. 定时器是否满足触发条件

查看源库的 dmtimer.ini 中的触发条件，或者 DISql 登录源库，通过 V\$DM_TIMER_INI 视图查看定时器的触发条件，包括起始时间、触发频率、定时器是否启

用等配置项，再根据操作系统当前时间来判断是否满足触发条件，可能会有人为修改操作系统时间或者定时器被关闭等导致定时器无法触发的情况，如果不满足触发条件，也无法同步数据到异步备库。

6.22 MPP 主备版本升级（从 V2.0 升级）

数据守护 V2.1 及以上版本，可以完全兼容实时主备和读写分离集群 V2.0 版本的配置文件，但是在 MPP 主备配置上与 V2.0 存在一定差异，因此从 V2.0 升级到更高版本时，需要修改所有 MPP 备库的 `dm.ini` 配置文件，将 `MPP_INI` 参数设置为 1，并从主库拷贝 MPP 控制文件 `dmmpp.ctl` 保存到备库的 `ctl_path` 目录。否则，数据守护从 V2.0 升级到 V2.1 及以上版本后，MPP 主备系统将无法正常运行。

6.23 MPP 主备限制登录说明

MPP 主备系统中可能会碰到登录受限的问题：[-607]:MPP 系统暂时限制用户登录。可能的场景如下：

- 1.配置顺序导致的。MPP 环境打开归档配置，`dmarch.ini` 配置实时归档，非 Mount 状态且非主备库模式（Normal），限制用户登录。原因是在 Normal 模式 Open 状态下登录后产生的日志不完整，有些 DW 类型日志只在 Primary 模式下才会生成。因此需要先配置库的主备模式后，再配置 MPP 实时归档。

- 2.MPP 主备系统处于切换、接管、修改 `MPP_INI` 参数等过程中会短暂限制用户登录。

- 3.MPP 主备系统中，如果切换过程中出现故障，可能导致各节点的 `dmmpp.ctl` 文件不一致，这种情况下会限制登录，此时需要通过监视器命令修复 `mppctl` 或者手动拷贝 `dmmpp.ctl` 文件保持一致，再对库进行重启操作。

6.24 备库查询限制说明

备库支持日志并行重演以后，各路并行重演线程分别负责重演不同的数据页，这些数据页在各路并行重演线程中重演的先后顺序是不确定的，也就是不能严格保证备库按照日志产生的顺序来修改数据页（这里指的是不同的数据页），由此会引发一些并发查询问题。

比如，主库总是先修改聚集索引，再修改二级索引，通过二级索引查询到记录，总是能

在聚集索引中定位到有效数据。但是，备库开启日志并行重演后，在二级索引中查询到的记录，不一定能在聚集索引中找到相应的数据。

再比如，如果聚集索引记录对当前事务不可见，需要通过记录中的回滚记录地址 RADDR 字段，从回滚页中读取回滚记录，来进一步判断事务可见性。但是，由于数据页与回滚页的重演进度是不一致的，通过 RADDR 定位到的回滚页可能还是其他事务修改的内容。即便 RADDR 定位到的回滚页是当前事务修改的，但回滚记录内容与物理记录也不一定匹配。

另外，事务的提交顺序与这个事务修改的数据页的重演进度，也不能严格保持匹配。利用事务的状态进行可见性判断时，也存在误判的可能。

因此，对于开启并行重演的备库，无法进行记录可见性判断，只能支持 READ UNCOMMITTED 隔离级，所有读取到的记录都认为是可见的。对于未开启并行重演的备库，则保持原有处理方式，支持类 READ COMMITTED 隔离级。

为了支持备库的查询功能，重演 ROOT 页封锁等类型的日志时，会进行局部串行化处理，对备库重演性能会造成一定的影响，在备库仅用于容灾的场景下，可以考虑将备库 dm.ini 中的 REDOS_ENABLE_SELECT 设置为 0，禁止在备库上查询所有非 vs 视图，配置为 0 时备库会取消部分串行化处理，能够进一步提升备库重演性能。

6.25 异步备库延时和定点重演

异步备库通过定时器定时和源库保持数据同步，每次同步数据时，源库会将本地新产生的日志全部发送给备库重演，在此基础上，还可以通过异步备库的延时和定点重演对异步备库的数据同步做进一步延时和定点限定。例如，延时重演设置为 1 小时，则主库仅会同步 1 小时前产生的日志到备库，比如重演到的时间点设置为 2021-6-4 10:30，则主库只会同步 2021-6-4 10:30 之前产生的日志到备库，之后产生的日志不会再发送到异步备库重演。

在这种模式下，异步备库始终和主库保持一段时间的日志差，此功能可在某些特定场景下发挥作用。例如，用户业务数据量很大，一旦发生了误 truncate 整张表的操作，通过备份还原方式恢复数据将会耗费很长时间，导致业务长时间中断，如果异步备库和主库之间存在一定的时间延时，在主库上数据被 truncate 以后，异步备库上还未重演，还能找到整张表数据，可以将表数据从异步备库上导出，再重新导入到主库，从而可以快速恢复主库数据，能够极大的缩短业务中断时长。

可以选择同时设置延时重演和定点重演，也可以只选择其中一项进行设置，其中延时重

演通过在源库的 dmarch.ini 文件中设置 ARCH_SEND_DELAY 生效，如果源库是主库，则需要所有备库上也进行设置，以便在主备切换后能够继续生效；定点重演功能则是通过在源库上执行系统函数或者通过执行监视器命令来动态进行设置。

下面以同时设置延时重演和定点重演功能进行举例说明：

1. 修改源库的 dmarch.ini 文件，设置异步备库的延时重演时间

如果源库是主库，则需要所有备库上也进行设置，以便在主备切换后能够继续生效。

例如，设置异步备库 GRP_LOCAL_01 延时时间为 30 分钟配置如下：

```
[ARCHIVE_ASYNC]
ARCH_TYPE           = ASYNC           #异步归档类型
ARCH_DEST           = GRP1_LOCAL_01   #异步归档目标实例名
ARCH_TIMER_NAME     = RT_TIMER        #定时器名称，和 dmtimer.ini 中的名称一致
ARCH_SEND_DELAY     = 30              #只发送 30 分钟之前产生的日志
```

延时配置存在以下限制：

- (1) 仅在异步归档配置项中支持此配置项，其他归档类型不支持。
- (2) 对 DMDSC 源库，建议用户在配置时保证各节点上配置的值是一致的，避免控制节点发生切换后延迟时间不一致（DMDSC 仅控制节点向异步备库发送数据）。
- (3) 如果 DMDSC 各节点配置在不同机器上，建议保证各机器时间一致，避免控制节点发生切换后延迟时间不一致。
- (4) 在实时备库或即时备库上需要同步进行配置，以便切换为主库后可以立即生效，对 DMDSC 备库，各节点也建议配置一致。

2. 动态设置重演到的时间点

可通过系统函数和监视器命令方式进行设置，下面分别举例说明。

(1) 通过系统函数设置

如设置异步备库 GRP_LOCAL_01 重演到 2021-6-4 10:30，则源库不再发送 2021-6-4 10:30 之后产生的日志，在源库上执行以下系统函数。

```
SQL>SP_SET_ARCH_SEND_UNTIL_TIME('GRP_LOCAL_01', '2020-04-28 10:30:00');
```

系统函数使用注意事项：

- 1) DSC 主库或备库修改时，仅允许在控制节点上修改。
- 2) 仅会修改内存值，源库重启即会失效，需要重新设置。
- 3) 指定时间串为空串时，会取消之前设置的时间点。

设置完成后，可使用系统函数查询设定的时间。

```
SQL>SF_GET_ARCH_SEND_UNTIL_TIME('GRP_LOCAL_01');
```

源库在到达定时器的定时间隔后，会根据配置的延时和定点时间信息向异步备库发送日志，如果想立即出发源库向异步备库发送日志，可通过执行以下系统函数来完成，如果源库是 DMDSC 集群，此系统函数只允许在控制节点上执行。

```
SQL>SF_NOTIFY_ARCH_SEND('GRP_LOCAL_01');
```

（2）通过监视器命令方式设置

在监视器上执行以下命令，指定异步备库 GRP_LOCAL_01 重演到 2021-6-4 10:30。

```
SET DATABASE GRP_LOCAL_01 UNTIL TIME 2021-6-4 10:30:00
```

监视器命令执行完成后，就会通知源库立即执行一次日志同步，可通过监视器 show 命令中的 APPLY_UNTIL_TIME 字段查看异步备库是否重演到了指定时间，同样的，此命令仅会修改源库中的内存值，源库重启即会失效，需要重新设置。

也可以通过监视器命令取消设置：

```
CANCEL DATABASE GRP_LOCAL_01 UNTIL TIME
```

以上全部设置完成后，在特定场景下一旦主库数据被误删除，即可借助异步备库达到快速恢复主库数据的目的。

表数据的导出及导入举例说明如下，以 TEST 表为例，使用时需要根据实际情况调整导出及导入的相关参数：

```
//从异步备库导出 TEST 表数据

dexp.exe    SYSDBA/SYSDBA@192.168.0.143:32143    file=test.txt    tables=(TEST)

TABLE_PARALLEL=40

//导入 TEST 表数据到源库

dimp.exe    SYSDBA/SYSDBA@192.168.0.141:32141    file=test.txt    tables=(TEST)

TABLE_EXISTS_ACTION=APPEND
```

6.26 备库归档磁盘空间不足

备库收到主库发送的日志包后，先将日志包写入归档日志文件，之后启动日志包的重演。在备库归档磁盘空间不足时（未设置归档空间上限，并且物理磁盘已经被写满），日志包写入归档日志文件的动作会被挂起，并等到人工清理出可用的磁盘空间后才会继续写入。这种

情况下，此日志包的重演动作也会被挂起，而主库还在不断地发送新的日志包过来，在前一个日志包进入重演任务系统之前，这些新收到的日志包都无法进入重演流程，如果备库一直未清理出可用的磁盘空间，最终就会达到日志包堆积上限，再收到新的日志包时，就会延迟响应主库，导致主库无法再执行写操作。

针对此问题，备库提供了一个 `dm.ini` 参数 `DW_ARCH_SPACE_CHECK` 来进行检测处理，此参数配置为 1 时（默认值为 0），会对上述场景进行检测并处理，确保不影响主库的写操作，此功能目前仅支持单节点备库，DSC 备库暂不支持。

`DW_ARCH_SPACE_CHECK` 置为 1 时，在备库检测到本地归档磁盘空间不足时，通知本地守护进程切换为 `SHUTDOWN` 状态，在检测到守护进程成功切换为 `SHUTDOWN` 状态后，备库主动将自己关闭，随后主库发送日志失败，会短暂的进入故障处理流程（先切换到 `SUSPEND` 状态，再重新 `OPEN`），然后继续正常运行。

之所以要将备库守护进程切换为 `SHUTDOWN` 状态，是为了避免在备库主动关闭后又被守护进程自动拉起，再次影响到主库。

在确认备库已清理出可用的磁盘空间后，可以在监视器上执行 `STARTUP DMWATCHER` 命令，打开备库守护进程的监控功能，备库守护进程会自动将备库实例拉起，数据守护集群会重新恢复正常。

6.27 注意事项

1. 数据库实例、守护进程、监视器启动后，如果收不到守护进程消息，或者守护进程收不到库中实例的消息（实例处于 `Error` 状态），请检查端口、`OGUID` 等配置，并确认是否消息被防火墙屏蔽。

2. 数据库实例、守护进程和监视器应用同一个用户启动，避免出现权限问题引发系统异常。

3. 手动方式启动数据守护系统时，对于守护进程、数据库实例和监视器的启动顺序没有严格要求，但手动方式退出数据守护系统时，一定要严格按照 [6.4 关闭数据守护系统](#) 小节中描述的顺序来操作，否则会导致主库异常关闭。

4. 如果 DMDSC 集群使用了 ASM 文件系统，则守护进程启动时必须等待 ASM 文件系统准备完成，即 `dmasmsvr` 服务启动完成，否则无法正常访问共享存储上的 `dmwatcher.ctl` 文件，在 `dmasmsvr` 服务启动完成之前，守护进程的启动会一直处于等

待状态。

5. DMDSC 集群如果作为备库配置在守护系统中,那么在配置源库的 dmarch.ini 时,要将 DMDSC 备库作为一个整体配置在同一个归档配置项中,DMDSC 备库的所有节点实例名以“/”分隔开作为一个 ARCH_DEST 进行配置,具体的配置示例请参考 7.5.1 配置说明小节。

6. DMDSC 集群存在一些特殊场景处理,比如在 Mount 或 Suspend 状态下启动故障处理或故障重加入时会主动宕机,守护进程在某些特定场景下也会主动宕机,具体的场景说明请参考 2.7.12 场景说明小节。

7. 备库故障重启,主库同步归档日志失败,可以通过监视器的 show arch send info 命令查看 Recovery 的历史同步信息,或者查看主库、备库的数据库日志信息、守护进程日志信息,检查失败原因。如果是主库归档日志不完整,或者主备库日志不连续等原因造成同步失败,则需要尽快重建备库。

8. 建议使用故障手动切换模式数据守护系统。如果配置为故障自动切换模式,则必须配置确认监视器。

9. 同时配置数据守护和高可靠集群(HIGH-AVAILABILITY,简称 HA)软件情况下,需要将 dmwatcher.ini 的 INST_AUTO_RESTART 设置为 0,避免 dmwatcher 与 HA 软件同时启动 dmserver 实例。同时,配置 dm.ini 中的 HA_INST_CHECK_IP/HA_INST_CHECK_PORT 参数,避免由于 HA 软件故障,引发多个 dmserver 实例同时操作一份数据,导致数据损坏。

10. 配置异步备库时,如果异步备库的源库是实时主库或即时主库,则需要在所有的实时备库或即时备库上配置相同的异步归档配置项、定时器等,以确保主备库发生切换后可以继续向异步备库同步数据。

11. 用户应该使用“正常接管”方式将备库切换为主库,在无法正常接管的情况下,再考虑使用“强制接管”方式,但“强制接管”前一定要确认主备库数据是一致的,避免引发组分裂。

12. 根据磁盘空间情况,合理设置 dmarch.ini 的 ARCH_SPACE_LIMIT 值,防止磁盘空间被归档日志文件占满。

13. 备库重做新建数据库文件操作时,缺省按照主库的目录进行创建,如果创建失败则会将文件创建到 dm.ini 中指定的 system_path 目录下。

14. DM 根据 dm.ini 参数 TS_MAX_ID/TS_FIL_MAX_ID 分配表空间和文件对象内存

存放空间，用户需要确保主备库参数一致。如果配置不一致，比如：主库 TS_MAX_ID > 备库 TS_MAX_ID、并且新建表空间 ID > 备库 TS_MAX_ID 情况下，主库创建表空间成功，备库重演创建表空间操作时检测到 ID 越界会强制退出系统。

15. 守护进程通过当前系统时间与最后一次接收消息时间的差值，来检测目标对象是否异常，因此，修改操作系统时间可能会导致误判系统出现故障，不建议在数据守护系统运行过程中调整操作系统时间。

16. 默认配置下，主备库都允许执行表空间脱机 (offline) 操作，并且主库的表空间脱机操作不会同步到备库。如果用户直接登录备库进行表空间脱机操作，重做 Redo 日志访问此表空间将导致备库强制退出。因此，针对备库的表空间脱机操作要十分谨慎。用户可以修改 dm.ini 配置参数 ENABLE_OFFLINE_TS = 2，限制备库表空间脱机操作，降低误操作概率。

17. 主库上新建表空间或增加文件等操作，当备库磁盘空间不足时，备库上重做日志会导致宕机。

18. MPP 主备交叉配置的环境中，创建表空间必须使用相对路径。例如，CREATE TABLESPACE TS4 DATAFILE 'TS4.DBF' SIZE 128。

19. 为了防止备库上重演日志堆积太多、占用大量内存、备库重演无响应导致主库挂住不能提供服务等情况的发生，可通过适当调整 BUFFER、REDOS_PRE_LOAD、REDOS_BUF_SIZE、REDOS_BUF_NUM、REDOS_MAX_DELAY 这几个参数达到加快备库重演速度、达到设置的堆积上限后延迟响应主机、备库自动宕机等目的。其中 REDOS_BUF_SIZE 和 REDOS_BUF_NUM 同时起作用，只要达到一个条件即延迟响应。

20. 备库支持索引监控，主备库可以分别设置不同的索引监控。备库还支持查询 v\$动态视图、执行获取和修改 INI 参数的系统函数以及执行备份相关的系统函数。需要注意的是，若是读写分离集群，以通常的登录名方式登录并执行上述操作可能会引发不确定性，操作可能在备库执行也可能在主库执行，建议使用 LOCAL 方式登录指定的数据库进行操作。

21. 主库使用 SP_SET_PARA_VALUE 系统过程修改 INI 参数时，不会生成 Redo 日志，因此此操作不会同步到备库。

22. 主库扩展日志文件后，也需要手动对备库扩展日志文件，以免备库出现日志环冲破的情况。

23. 主库执行 CREATE TABLESPACE 创建表空间、ALTER TABLESPACE ADD DATAFILE 表空间加文件、重命名文件等操作时，会将主库的文件路径信息写入到 Redo 日

志中。备库重做这些 Redo 日志时，在备库创建相应的文件。备库创建文件的规则如下：

- 1) 主库指定的是绝对路径，备库优先在相同目录下创建文件；
- 2) 主库指定是相对路径，备库使用 `dm.ini` 配置文件中的 `system_path` 和主库指定的文件名进行拼装，转换为绝对路径。这一步骤中，如果 `system_path + file_name` 的总长度超过 256 个字节，文件路径拼接失败，系统会强制退出。
- 3) 尝试创建文件，如果文件创建失败，则尝试在 `system_path` 目录下创建文件；如果仍然创建失败，则重命名文件，继续尝试在 `system_path` 目录下创建文件。如果连续重试 3 次仍然失败，则备库会强制退出。

7 数据守护搭建

在搭建数据守护系统前，应注意数据守护系统中各实例使用的 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，以及使用同一个用户启动 DM 服务器和守护进程 `dmwatcher`，以免系统在运行时出现意想不到的错误。

7.1 数据准备

配置数据守护 V4.0 之前，必须先通过备份还原方式同步各数据库的数据，确保各数据的数据保持完全一致。主库可以是新初始化的数据库，也可以是正在生产、使用中的数据库。备库在备份还原前，需要先准备初始化一个新库。

不能使用分别初始化库或者直接拷贝数据文件的方法来直接配置数据库守护，原因如下：

1. 每个库都有一个永久魔数 (`permanent_magic`)，一经生成，永远不会改变，主库传送日志时会判断这个值是否一样，确保是来自同一个数据守护环境中的库，否则传送不了日志。
2. 由于 `dminit` 初始化数据库时，会生成随机密钥用于加密，每次生成的密钥都不相同，备库无法解析采用主库密钥加密的数据。
3. 每个库都有一个数据库魔数 (`DB_MAGIC`)，每经过一次还原、恢复操作，`DB_MAGIC` 就会产生变化，需要通过这种方式来区分同一个数据守护环境中各个不同的库。

如果需要使用 **Huge** 表，在初始化库时需要将建库参数 `HUGE_WITH_DELTA` 和 `RLOG_GEN_FOR_HUGE` 都配置为 1。



对于新初始化的库，首次启动不允许使用 **Mount** 方式，需要先正常启动并正常退出，然后才允许 **Mount** 方式启动。

准备数据时，如果主库是新初始化的库，先正常启动并正常退出，然后再使用备份还原方式准备备库数据。

如果是初始搭建环境，可以通过对主库脱机备份、对备库脱机还原的方式来准备数据，如果主库已经处于运行状态，则可以对主库进行联机备份、对备库脱机还原的方式来准备数

据。

两种方式都需要服务器配置本地归档，本地归档配置方式如下：

1. 配置 dm.ini，打开 ARCH_INI 参数

```
ARCH_INI          = 1      #打开归档配置
```

2. 配置 dmarch.ini

```
[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL #本地归档类型

ARCH_DEST          = /dm/data/DAMENG/arch #本地归档文件存放路径

ARCH_FILE_SIZE     = 128   #单位 Mb，本地单个归档文件最大值

ARCH_SPACE_LIMIT   = 0     #单位 Mb，0 表示无限制，范围 1024~2147483647M
```

关于备份还原更详细的说明可以参考《DM8 备份与还原》。

7.1.1 脱机备份、脱机还原方式

1. 正常关闭主机数据库

2. 进行脱机备份

```
./dmrman CTLSTMT="BACKUP DATABASE '/dm/data/DAMENG/dm.ini' FULL TO BACKUP_FILE1
BACKUPSET '/dm/data/BACKUP_FILE_01'"
```

3. 拷贝备份文件到备库所在机器

4. 执行脱机数据库还原与恢复

数据库还原：

```
./dmrman CTLSTMT="RESTORE DATABASE '/dm/data/DAMENG/dm.ini' FROM BACKUPSET
'/dm/data/BACKUP_FILE_01'"
```

因为脱机备份没有产生任何 REDO 日志，所以此处省略恢复数据库的操作步骤。

数据库更新：

```
./dmrman CTLSTMT="RECOVER DATABASE '/dm/data/DAMENG/dm.ini' UPDATE DB_MAGIC"
```

7.1.2 联机备份、脱机还原方式

1. 对主库进行联机备份操作

```
SQL> BACKUP DATABASE BACKUPSET '/dm/data/BACKUP_FILE_01';
```

2. 拷贝备份文件到备库所在机器
3. 执行脱机数据库还原与恢复

```
./dmrman CTLSTMT="RESTORE DATABASE '/dm/data/DAMENG/dm.ini' FROM BACKUPSET
'/dm/data/BACKUP_FILE_01'"
./dmrman CTLSTMT="RECOVER DATABASE '/dm/data/DAMENG/dm.ini' FROM BACKUPSET
'/dm/data/BACKUP_FILE_01'"
./dmrman CTLSTMT="RECOVER DATABASE '/dm/data/DAMENG/dm.ini' UPDATE DB_MAGIC"
```

如果单节点系统已经上线，数据库服务不允许中断情况下，可以按照第“6.14 节备库重建”的步骤完成备库数据准备。

备份文件中会记录原备份库的模式和 OGUID 信息，使用备份文件还原成功后，需要再根据实际配置情况修改数据库的模式和 OGUID 值。如果执行备份时，待备份的库是 **Normal** 模式，并且不能确定这个库一定作为主库使



注意: 用，则对 **Normal** 模式的库必须使用脱机备份，不能使用联机备份方式，避免备份完成后，**Normal** 模式的库 LSN 有增长，又将其修改为 **Standby** 模式，而使用备份集还原后的库修改为 **Primary** 模式，备库数据比主库数据多，导致主备数据不一致的情况出现。因此，对执行联机备份的库，建议是已经修改为 **Primary** 模式的库。

7.2 配置实时主备

配置实时主备，有以下几种配置方案，可以根据实际情况部署：

1. 只配置主库和最多 8 个实时备库。
2. 只配置主库和最多 8 个异步备库。
3. 配置主库、最多 8 个实时备库，和最多 8 个异步备库。

在实际应用中，如果数据库规模很大，并且对数据的实时性要求不是很严格，则可以配置多个异步备库用于分担统计报表等任务。

异步备库的配置可以参考 [7.6 配置异步备库](#)，由于实时主备、读写分离集群、MPP 主备都支持配置异步备库，因此单独在 [7.6 配置异步备库](#) 进行配置说明。

为了帮助大家更加直观地理解 DM 数据守护方案，下面举例说明如何配置一个完整的实时主备的过程，配置方案为常见的一个主库和一个实时备库。



实际配置时，相关的端口配置和 **OGUID** 值建议不要和手册示例使用完全相同

注意: 的值，避免多个用户在同一个环境下搭建不同的数据守护系统，出现消息混乱或者端口冲突等问题。

7.2.1 环境说明

准备三台机器 DW_P、DW_S1、DW_M，DW_P 和 DW_S1 用来部署主备库，DW_M 用来部署确认监视器。其中 DW_P 和 DW_S1 配置两块网卡，一块接入内部网络交换模块，一块接入到外部交换机，DW_M 接入内部网络。

三台机器都事先安装了 DM，安装路径为 '/dm'，执行程序保存在 '/dm/bin' 目录中，数据存放路径为 '/dm/data'。

各主备库的实例名建议采用“组名_守护环境_序号”的方式命名，方便按组区分不同实例，注意总长度不能超过 16。本示例中组名为“GRP1”，配置为实时主备，主库命名为“GRP1_RT_01”，备库命名为“GRP1_RT_02”。

表 7.1 配置环境说明

机器名	IP 地址	初始状态	操作系统	备注
DW_P	192.168.1.131	主库	Linux rh6-141.test	192.168.1.131
	192.168.0.141	GRP1_RT_01	2.6.32-220.el6.x86_64 #1 SMP Wed Nov 9 08:03:13 EST 2011 x86_64 x86_64 x86_64 GNU/Linux	外部服务 IP; 192.168.0.141 内部通信 IP
DW_S1	192.168.1.132	备库	Linux rh6-142.localdomain	192.168.1.132
	192.168.0.142	GRP1_RT_02	2.6.32-220.el6.x86_64 #1 SMP Wed Nov 9 08:03:13 EST 2011	外部服务 IP; 192.168.0.142 内部通信 IP

			x86_64 x86_64 x86_64	
			GNU/Linux	
DW_M	192.168.0.73	确认监视器	Linux	
			rh6-73.localdomain	
			2.6.32-220.el6.x86_64	
			#1 SMP Wed Nov 9	
			08:03:13 EST 2011	
			x86_64 x86_64 x86_64	
			GNU/Linux	

表 7.2 端口规划

实例名	PORT_NUM	MAL_INST_DW_PORT	MAL_HOST	MAL_PORT	MAL_DW_PORT
GRP1_RT_01	32141	33141	192.168.0.141	61141	52141
GRP1_RT_02	32142	33142	192.168.0.142	61142	52142

7.2.2 数据准备

DW_P 机器上初始化主库至目录/dm/data：

```
./dminit path=/dm/data
```

DW_S1 机器上初始化备库至目录/dm/data：

```
./dminit path=/dm/data
```

然后按照 7.1 数据准备中的方法准备备库数据。

7.2.3 配置主库 GRP1_RT_01

7.2.3.1 配置 dm.ini

在 DW_P 机器上配置主库的实例名为 GRP1_RT_01，dm.ini 参数修改如下：

```
#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16
INSTANCE_NAME          = GRP1_RT_01
PORT_NUM                = 32141 #数据库实例监听端口
```

```

DW_INACTIVE_INTERVAL      = 60      #接收守护进程消息超时时间
ALTER_MODE_STATUS         = 0       #不允许手工方式修改实例模式/状态/OGUID
ENABLE_OFFLINE_TS         = 2       #不允许备库 OFFLINE 表空间
MAL_INI                   = 1       #打开 MAL 系统
ARCH_INI                  = 1       #打开归档配置
RLOG_SEND_APPLY_MON       = 64      #统计最近 64 次的日志发送信息

```

7.2.3.2 配置 dmmal.ini

配置 MAL 系统，各主备库的 dmmal.ini 配置必须完全一致，MAL_HOST 使用内部网络 IP，MAL_PORT 与 dm.ini 中 PORT_NUM 使用不同的端口值，MAL_DW_PORT 是各实例对应的守护进程之间，以及守护进程和监视器之间的通信端口，配置如下：

```

MAL_CHECK_INTERVAL        = 5       #MAL 链路检测时间间隔
MAL_CONN_FAIL_INTERVAL    = 5       #判定 MAL 链路断开的时间

[MAL_INST1]

MAL_INST_NAME             = GRP1_RT_01 #实例名，和 dm.ini 中的 INSTANCE_NAME 一致
MAL_HOST                  = 192.168.0.141 #MAL 系统监听 TCP 连接的 IP 地址
MAL_PORT                  = 61141      #MAL 系统监听 TCP 连接的端口
MAL_INST_HOST             = 192.168.1.131 #实例的对外服务 IP 地址
MAL_INST_PORT             = 32141 #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致
MAL_DW_PORT               = 52141 #实例本地的守护进程监听 TCP 连接的端口
MAL_INST_DW_PORT          = 33141 #实例监听守护进程 TCP 连接的端口

[MAL_INST2]

MAL_INST_NAME             = GRP1_RT_02
MAL_HOST                  = 192.168.0.142
MAL_PORT                  = 61142
MAL_INST_HOST             = 192.168.1.132
MAL_INST_PORT             = 32142
MAL_DW_PORT               = 52142

```

```
MAL_INST_DW_PORT      = 33142
```

7.2.3.3 配置 dmarch.ini

修改 dmarch.ini，配置本地归档和实时归档。

除了本地归档外，其他归档配置项中的 ARCH_DEST 表示实例是 Primary 模式时，需要同步归档数据的目标实例名。

当前实例 GRP1_RT_01 是主库，需要向 GRP1_RT_02（实时备库）同步数据，因此实时归档的 ARCH_DEST 配置为 GRP1_RT_02。

```
[ARCHIVE_REALTIME]

ARCH_TYPE      = REALTIME      #实时归档类型

ARCH_DEST      = GRP1_RT_02    #实时归档目标实例名

[ARCHIVE_LOCAL1]

ARCH_TYPE      = LOCAL        #本地归档类型

ARCH_DEST      = /dm/data/DAMENG/arch  #本地归档文件存放路径

ARCH_FILE_SIZE = 128          #单位 Mb，本地单个归档文件最大值

ARCH_SPACE_LIMIT = 0          #单位 Mb，0 表示无限制，范围 1024~2147483647M
```

7.2.3.4 配置 dmwatcher.ini

修改 dmwatcher.ini 配置守护进程，配置为全局守护类型，使用自动切换模式。

```
[GRP1]

DW_TYPE        = GLOBAL      #全局守护类型

DW_MODE        = AUTO        #自动切换模式

DW_ERROR_TIME  = 10          #远程守护进程故障认定时间

INST_RECOVER_TIME = 60        #主库守护进程启动恢复的间隔时间

INST_ERROR_TIME = 10          #本地实例故障认定时间

INST_OGUID     = 453331      #守护系统唯一 OGUID 值

INST_INI       = /dm/data/DAMENG/dm.ini  #dm.ini 配置文件路径

INST_AUTO_RESTART = 1        #打开实例的自动启动功能
```

```

INST_STARTUP_CMD    = /dm/bin/dmserver    #命令行方式启动

RLOG_SEND_THRESHOLD = 0                    #指定主库发送日志到备库的时间阈值，默认关闭

RLOG_APPLY_THRESHOLD = 0                  #指定备库重演日志的时间阈值，默认关闭

```

7.2.3.5 启动主库

以 Mount 方式启动主库

```
./dmserver /dm/data/DAMENG/dm.ini mount
```



注意：

一定要以 **Mount** 方式启动数据库实例，否则系统启动时会重构回滚表空间，生成 **Redo** 日志；并且，启动后应用可能连接到数据库实例进行操作，破坏主备库的数据一致性。数据守护配置结束后，守护进程会自动 **Open** 数据库。

7.2.3.6 设置 OGUID

启动命令行工具 DIsql，登录主库设置 OGUID 值。

```

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>sp_set_oguid(453331);

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);

```



注意：

系统通过 **OGUID** 值确定一个守护进程组，由用户保证 **OGUID** 值的唯一性，并确保数据守护系统中，数据库、守护进程和监视器配置相同的 **OGUID** 值。

7.2.3.7 修改数据库模式

启动命令行工具 DIsql，登录主库修改数据库为 Primary 模式

```
SQL>alter database primary;
```

7.2.4 配置备库 GRP1_RT_02

7.2.4.1 配置 dm.ini

在 DW_S1 机器上配置备库的实例名为 GRP1_RT_02，dm.ini 参数修改如下：

#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16

```

INSTANCE_NAME          = GRP1_RT_02

PORT_NUM               = 32142 #数据库实例监听端口

DW_INACTIVE_INTERVAL   = 60    #接收守护进程消息超时时间

ALTER_MODE_STATUS      = 0     #不允许手工方式修改实例模式/状态/OGUID

ENABLE_OFFLINE_TS      = 2     #不允许备库 OFFLINE 表空间

MAL_INI                = 1     #打开 MAL 系统

ARCH_INI               = 1     #打开归档配置

RLOG_SEND_APPLY_MON    = 64    #统计最近 64 次的日志重演信息

```

7.2.4.2 配置 dmmal.ini

配置 MAL 系统，各主备库的 dmmal.ini 配置必须完全一致，MAL_HOST 使用内部网络 IP，MAL_PORT 与 dm.ini 中 PORT_NUM 使用不同的端口值，MAL_DW_PORT 是各实例对应的守护进程之间，以及守护进程和监视器之间的通信端口，配置如下：

```

MAL_CHECK_INTERVAL     = 5      #MAL 链路检测时间间隔

MAL_CONN_FAIL_INTERVAL = 5      #判定 MAL 链路断开的的时间

[MAL_INST1]

MAL_INST_NAME          = GRP1_RT_01 #实例名，和 dm.ini 中的 INSTANCE_NAME 一致

MAL_HOST               = 192.168.0.141 #MAL 系统监听 TCP 连接的 IP 地址

MAL_PORT               = 61141      #MAL 系统监听 TCP 连接的端口

MAL_INST_HOST          = 192.168.1.131 #实例的对外服务 IP 地址

MAL_INST_PORT          = 32141 #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致

MAL_DW_PORT            = 52141 #实例对应的守护进程监听 TCP 连接的端口

MAL_INST_DW_PORT       = 33141 #实例监听守护进程 TCP 连接的端口

```

```
[MAL_INST2]

MAL_INST_NAME      = GRP1_RT_02

MAL_HOST           = 192.168.0.142

MAL_PORT          = 61142

MAL_INST_HOST      = 192.168.1.132

MAL_INST_PORT      = 32142

MAL_DW_PORT        = 52142

MAL_INST_DW_PORT   = 33142
```

7.2.4.3 配置 dmarch.ini

修改 dmarch.ini，配置本地归档和实时归档。

除了本地归档外，其他归档配置项中的 ARCH_DEST 表示实例是 Primary 模式时，需要同步归档数据的目标实例名。

当前实例 GRP1_RT_02 是备库，守护系统配置完成后，可能在各种故障处理中，GRP1_RT_02 切换为新的主库，正常情况下，GRP1_RT_01 会切换为新的备库，需要向 GRP1_RT_01 同步数据，因此实时归档的 ARCH_DEST 配置为 GRP1_RT_01。

```
[ARCHIVE_REALTIME]

ARCH_TYPE          = REALTIME      #实时归档类型

ARCH_DEST          = GRP1_RT_01    #实时归档目标实例名

[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL         #本地归档类型

ARCH_DEST          = /dm/data/DAMENG/arch #本地归档文件路径

ARCH_FILE_SIZE     = 128          #单位 Mb，本地单个归档文件最大值

ARCH_SPACE_LIMIT   = 0            #单位 Mb，0 表示无限制，范围 1024~2147483647M
```

7.2.4.4 配置 dmwatcher.ini

修改 dmwatcher.ini 配置守护进程，配置为全局守护类型，使用自动切换模式。

```
[GRP1]
```

DW_TYPE	= GLOBAL	#全局守护类型
DW_MODE	= AUTO	#自动切换模式
DW_ERROR_TIME	= 10	#远程守护进程故障认定时间
INST_RECOVER_TIME	= 60	#主库守护进程启动恢复的间隔时间
INST_ERROR_TIME	= 10	#本地实例故障认定时间
INST_OGUID	= 453331	#守护系统唯一 OGUID 值
INST_INI	= /dm/data/DAMENG/dm.ini	#dm.ini 配置文件路径
INST_AUTO_RESTART	= 1	#打开实例的自动启动功能
INST_STARTUP_CMD	= /dm/bin/dmserver	#命令行方式启动
RLOG_APPLY_THRESHOLD	= 0	#指定备库重演日志的时间阈值，默认关闭

7.2.4.5 启动备库

以 Mount 方式启动备库

```
./dmserver /dm/data/DAMENG/dm.ini mount
```



注意：

一定要以 **Mount** 方式启动数据库实例，否则系统启动时会重构回滚表空间，生成 **Redo** 日志；并且，启动后应用可能连接到数据库实例进行操作，破坏主备库的数据一致性。数据守护配置结束后，守护进程会自动 **Open** 数据库。

7.2.4.6 设置 OGUID

启动命令行工具 **Disql**，登录备库设置 OGUID 值为 453331

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>sp_set_oguid(453331);

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```



注意：

系统通过 **OGUID** 值确定一个守护进程组，由用户保证 **OGUID** 值的唯一性，并确保数据守护系统中，数据库、守护进程和监视器配置相同的 **OGUID** 值。

7.2.4.7 修改数据库模式

启动命令行工具 DIsql，登录备库修改数据库为 Standby 模式。

如果当前数据库不是 Normal 模式，需要先修改 dm.ini 中 ALTER_MODE_STATUS 值为 1，允许修改数据库模式，修改 Standby 模式成功后再改回为 0。如果是 Normal 模式，请忽略下面的第 1 步和第 3 步。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);      ----第 1 步
SQL>alter database standby;                               ----第 2 步
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);      ----第 3 步
```

7.2.5 配置监视器

配置监视器时可以选择配置单实例监视器或配置多实例监视器，下面将分别介绍配置单实例监视器以及多实例监视器时 dmmonitor.ini 配置文件中的配置信息。

7.2.5.1 配置单实例监视器

由于主库和实时备库的守护进程配置为自动切换模式，因此这里选择配置确认监视器。和普通监视器相比，确认监视器除了相同的命令支持外，在主库发生故障时，能够自动通知实时备库接管为新的主库，具有自动故障处理的功能。



故障自动切换模式下，必须配置确认监视器，且确认监视器最多只能配置一个。

修改 dmmonitor.ini 配置确认监视器，其中 MON_DW_IP 中的 IP 和 PORT 和 dmmal.ini 中的 MAL_HOST 和 MAL_DW_PORT 配置项保持一致。

```
MON_DW_CONFIRM      = 1      #确认监视器模式
MON_LOG_PATH        = /dm/data/log      #监视器日志文件存放路径
MON_LOG_INTERVAL    = 60      #每隔 60s 定时记录系统信息到日志文件
MON_LOG_FILE_SIZE    = 32      #每个日志文件最大 32M
MON_LOG_SPACE_LIMIT = 0      #不限定日志文件总占用空间

[GRP1]
```

```

MON_INST_OGUID          = 453331 #组 GRP1 的唯一 OGUID 值

#以下配置为监视器到组 GRP1 的守护进程的连接信息，以“IP:PORT”的形式配置

#IP 对应 dmmal.ini 中的 MAL_HOST，PORT 对应 dmmal.ini 中的 MAL_DW_PORT

MON_DW_IP                = 192.168.0.141:52141

MON_DW_IP                = 192.168.0.142:52142

```

7.2.5.2 配置多实例监视器

与配置单实例监视器相比，配置多实例监视器时 `dmmonitor.ini` 中需要添加整个监视器系统中的实例总数 `MON_INST_NUM`（取值为 1、3 或者 5），各实例通信的心跳校验间隔 `MON_HB_INTERVAL`（取值（5~600），单位秒），各实例通信的心跳间隔 `MON_BRO_INTERVAL`（取值（5~600），单位毫秒），各实例通信的基础投票间隔 `MON_VOTE_INTERVAL`（取值（5~600），单位毫秒），各实例通信的心跳间隔以及基础投票间隔会在真正使用时进行调整，各实例通信心跳间隔需要比基础投票间隔小（若配置中各实例通信心跳间隔比基础投票间隔大，内部会将实例通信心跳间隔调整为与基础投票间隔相同，再进行其他的调整处理），增加监视器 ID 即 `MON_ID`（用于区别所有监视器中的不同实例，每个监视器实例的 `MON_ID` 与 `MON_INST_ID` 相同，取值（1~2147483647）），增加监视器系统唯一标识 `MON_MID`（取值（1~2147483647）），多实例监视器系统中的所有监视器需配置相同的 `MON_MID`。

与配置单实例监视器相比，配置多实例监视器时组的配置信息不做更改，监视器实例的配置信息需要新增配置参数，通过组和实例的配置信息内容的不同来区分组和实例，如果用户将组和实例的配置信息混淆，则以配置信息中更加靠前的参数为准来决定当前配置属于组还是实例，不属于组或实例的配置项将会被忽略。

监视器实例的配置信息包括实例 IP 地址、监听端口号以及实例在系统中的唯一编号，实例的 IP 地址和监听端口号不应出现冲突，否则该监视器实例可能无法正常使用。所有监视器实例的 `dmmonitor.ini` 配置文件中除 `MON_ID` 参数项外应完全一致，每个监视器的 `MON_ID` 应互不相同，用于区分同一个监视器系统中的不同监视器实例，若配置相同的 `MON_ID` 将导致通信无法正常建立。监视器实例监听的 IP 及端口号的正确性由用户保证。

配置多实例监视器时 `dmmonitor.ini` 配置文件中的内容如下：

```

MON_LOG_PATH            = /dm/data/log

```

```

MON_LOG_INTERVAL          = 60

MON_LOG_FILE_SIZE         = 32

MON_LOG_SPACE_LIMIT       = 0

MON_DW_CONFIRM            = 1

MON_INST_NUM              = 3          #实例总个数

MON_HB_INTERVAL           = 60        #通信心跳校验间隔

MON_BRO_INTERVAL         = 100       #raft 协议中实例通信心跳间隔

MON_VOTE_INTERVAL        = 100       #raft 协议中基础投票间隔

MON_ID                    = 1         #当前监视器在监视器系统中的 ID

MON_MID                   = 45614     #当前监视器系统的唯一标识


[GRP1]

MON_INST_OGUID            = 453331

MON_DW_IP                 = 192.168.0.141:52141

MON_DW_IP                 = 192.168.0.142:52142


[MON1]

MON_HOST                  = 192.168.0.141    #系统监听 TCP 连接的 IP 地址

MON_PORT                  = 8339            #系统监听 TCP 连接的端口号

MON_INST_ID               = 1              #监视器实例在监视器系统中的 ID


[MON2]

MON_HOST                  = 192.168.0.141    #系统监听 TCP 连接的 IP 地址

MON_PORT                  = 8340            #系统监听 TCP 连接的端口号

MON_INST_ID               = 2              #监视器实例在监视器系统中的 ID


[MON3]

MON_HOST                  = 192.168.0.141    #系统监听 TCP 连接的 IP 地址

MON_PORT                  = 8341            #系统监听 TCP 连接的端口号

MON_INST_ID               = 3              #监视器实例在监视器系统中的 ID

```

7.2.6 启动守护进程

启动各个主备库上的守护进程：

```
./dmwatcher /dm/data/DAMENG/dmwatcher.ini
```

守护进程启动后，进入 Startup 状态，此时实例都处于 Mount 状态。守护进程开始广播自身和其监控实例的状态信息，结合自身信息和远程守护进程的广播信息，守护进程将本地实例 Open，并切换为 Open 状态。

7.2.7 启动监视器

启动监视器：

```
./dmmonitor /dm/data/dmmonitor.ini
```

监视器提供一系列命令，支持当前守护系统状态查看以及故障处理，可输入 help 命令，查看各种命令使用说明，结合实际情况选择使用。

至此一主一备的实时数据守护系统搭建完毕，在搭建步骤和各项配置都正确的情况下，在监视器上执行 show 命令，可以监控到所有实例都处于 Open 状态，所有守护进程也都处于 Open 状态，即为正常运行状态。

7.3 配置读写分离集群

配置读写分离集群，有以下几种配置方案，可以根据实际情况部署：

1. 只配置主库和最多 8 个即时备库。
2. 只配置主库和最多 8 个实时备库。
3. 只配置主库和最多 8 个异步备库。
4. 配置主库、最多 8 个即时备库和最多 8 个异步备库。
5. 配置主库、最多 8 个实时备库和最多 8 个异步备库。

在实际应用中，如果数据库规模很大，并且对数据的实时性要求不是很严格，则可以配置多个异步备库用于分担统计报表等任务。

异步备库的配置可以参考 [7.6 配置异步备库](#)，由于实时主备、读写分离集群、MPP 主备都支持配置异步备库，因此单独在 [7.6 配置异步备库](#) 进行配置说明。

为了帮助大家更加直观的理解 DM 数据守护方案，下面举例说明如何配置一个完整的读

写分离集群的过程，配置方案为一个主库和两个即时备库。



注意：

实际配置时，相关的端口配置和 **OGUID** 值建议不要和手册示例使用完全相同的值，避免多个用户在同一个环境下搭建不同的数据守护系统，出现消息混乱或者端口冲突等问题。

7.3.1 环境说明

准备四台机器 DW_P、DW_S1、DW_S2、DW_M，DW_P、DW_S1 和 DW_S2 用来部署主备库，DW_M 用来部署确认监视器。其中 DW_P、DW_S1 和 DW_S2 配置两块网卡，一块接入内部网络交换模块，一块接入到外部交换机，DW_M 接入内部网络。

四台机器事先都安装了 DM，安装路径为 `/dm`，执行程序保存在 `/dm/bin` 目录中，数据存放路径为 `/dm/data`。

各主备库的实例名建议采用“组名_守护环境_序号”的方式命名，方便按组区分不同实例，注意总长度不能超过 16。本示例中组名为“GRP1”，配置为读写分离集群，主库命名为“GRP1_RWW_01”，备库分别命名为“GRP1_RWW_02”和“GRP1_RWW_03”。

表 7.3 配置环境说明

机器名	IP 地址	初始状态	操作系统	备注
DW_P	192.168.1.13	主库	Linux rh6-141.test	192.168.1.13
	1	GRP1_RWW_01	2.6.32-220.el6.x86_	1 外部服务 IP;
	192.168.0.14		64 #1 SMP Wed Nov 9	192.168.0.14
	1		08:03:13 EST 2011	1 内部通信 IP
			x86_64 x86_64 x86_64	
			GNU/Linux	
DW_S1	192.168.1.13	备库	Linux	192.168.1.13
	2	GRP1_RWW_02	rh6-142.localdomain	2 外部服务 IP;
	192.168.0.14		2.6.32-220.el6.x86_	192.168.0.14
	2		64 #1 SMP Wed Nov 9	2 内部通信 IP
			08:03:13 EST 2011	
			x86_64 x86_64 x86_64	
			GNU/Linux	

DW_S2	192.168.1.13 3 192.168.0.14 3	备库 GRP1_RWW_03	Linux rh6-143.localdomain 2.6.32-220.el6.x86_ 64 #1 SMP Wed Nov 9 08:03:13 EST 2011 x86_64 x86_64 x86_64 GNU/Linux	192.168.1.13 3 外部服务 IP; 192.168.0.14 3 内部通信 IP
DW_M	192.168.0.73	确认监视器	Linux rh6-73.localdomain 2.6.32-220.el6.x86_ 64 #1 SMP Wed Nov 9 08:03:13 EST 2011 x86_64 x86_64 x86_64 GNU/Linux	

表 7.4 端口规划

实例名	PORT_NUM	MAL_INST_DW_PORT	MAL_HOST	MAL_PORT	MAL_DW_PORT
GRP1_RWW_01	32141	33141	192.168.0.141	61141	52141
GRP1_RWW_02	32142	33142	192.168.0.142	61142	52142
GRP1_RWW_03	32143	33143	192.168.0.143	61143	52143

7.3.2 数据准备

DW_P 机器上初始化库至目录/dm/data:

```
/dminit path=/dm/data
```

然后按照 7.1 数据准备中的方法分别准备备数据库数据。

7.3.3 配置主库 GRP1_RWW_01

7.3.3.1 配置 dm.ini

在 DW_P 机器上配置主库的实例名为 GRP1_RWW_01，dm.ini 参数修改如下：

```
#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16

INSTANCE_NAME          = GRP1_RWW_01

PORT_NUM               = 32141 #数据库实例监听端口

DW_INACTIVE_INTERVAL   = 60    #接收守护进程消息超时时间

ALTER_MODE_STATUS      = 0     #不允许手工方式修改实例模式/状态/OGUID

ENABLE_OFFLINE_TS      = 2     #不允许备库 OFFLINE 表空间

MAL_INI                = 1     #打开 MAL 系统

ARCH_INI               = 1     #打开归档配置

RLOG_SEND_APPLY_MON    = 64    #统计最近 64 次的日志发送信息
```

7.3.3.2 配置 dmmal.ini

配置 MAL 系统，各主备库的 dmmal.ini 配置必须完全一致，MAL_HOST 使用内部网络 IP，MAL_PORT 与 dm.ini 中 PORT_NUM 使用不同的端口值，MAL_DW_PORT 是各实例对应的守护进程之间，以及守护进程和监视器之间的通信端口，配置如下：

```
MAL_CHECK_INTERVAL     = 5      #MAL 链路检测时间间隔

MAL_CONN_FAIL_INTERVAL = 5      #判定 MAL 链路断开的的时间

[MAL_INST1]

MAL_INST_NAME          = GRP1_RWW_01 #实例名，和 dm.ini 中的 INSTANCE_NAME 一致

MAL_HOST               = 192.168.0.141 #MAL 系统监听 TCP 连接的 IP 地址

MAL_PORT               = 61141      #MAL 系统监听 TCP 连接的端口

MAL_INST_HOST          = 192.168.1.131 #实例的对外服务 IP 地址

MAL_INST_PORT          = 32141 #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致

MAL_DW_PORT            = 52141 #实例对应的守护进程监听 TCP 连接的端口

MAL_INST_DW_PORT       = 33141 #实例监听守护进程 TCP 连接的端口
```

```
[MAL_INST2]

MAL_INST_NAME      = GRP1_RWW_02
MAL_HOST           = 192.168.0.142
MAL_PORT           = 61142
MAL_INST_HOST      = 192.168.1.132
MAL_INST_PORT      = 32142
MAL_DW_PORT        = 52142
MAL_INST_DW_PORT   = 33142
```

```
[MAL_INST3]

MAL_INST_NAME      = GRP1_RWW_03
MAL_HOST           = 192.168.0.143
MAL_PORT           = 61143
MAL_INST_HOST      = 192.168.1.133
MAL_INST_PORT      = 32143
MAL_DW_PORT        = 52143
MAL_INST_DW_PORT   = 33143
```

7.3.3.3 配置 dmarch.ini

修改 dmarch.ini，配置本地归档和即时归档。

除了本地归档外，其他归档配置项中的 ARCH_DEST 表示实例是 Primary 模式时，需要同步归档数据的目标实例名。

当前实例 GRP1_RWW_01 是主库，需要向即时备库 GRP1_RWW_02/ GRP1_RWW_03 同步数据，因此即时归档的 ARCH_DEST 分别配置为 GRP1_RWW_02 和 GRP1_RWW_03。

```
[ARCHIVE_TIMELY1]

ARCH_TYPE          = TIMELY          #即时归档类型
ARCH_DEST          = GRP1_RWW_02     #即时归档目标实例名

[ARCHIVE_TIMELY2]

ARCH_TYPE          = TIMELY          #即时归档类型
ARCH_DEST          = GRP1_RWW_03     #即时归档目标实例名
```

```
[ARCHIVE_LOCAL1]

ARCH_TYPE           = LOCAL           #本地归档类型

ARCH_DEST           = /dm/data/DAMENG/arch #本地归档文件存放路径

ARCH_FILE_SIZE      = 128             #单位 Mb，本地单个归档文件最大值

ARCH_SPACE_LIMIT    = 0               #单位 Mb，0 表示无限制，范围 1024~2147483647M
```

7.3.3.4 配置 dmwatcher.ini

修改 dmwatcher.ini 配置守护进程，配置为全局守护类型，使用自动切换模式。

```
[GRP1]

DW_TYPE             = GLOBAL          #全局守护类型

DW_MODE             = AUTO           #自动切换模式

DW_ERROR_TIME       = 10              #远程守护进程故障认定时间

INST_RECOVER_TIME   = 60              #主库守护进程启动恢复的间隔时间

INST_ERROR_TIME     = 10              #本地实例故障认定时间

INST_OGUID          = 453332         #守护系统唯一 OGUID 值

INST_INI            = /dm/data/DAMENG/dm.ini #dm.ini 配置文件路径

INST_AUTO_RESTART   = 1              #打开实例的自动启动功能

INST_STARTUP_CMD    = /dm/bin/dmserver #命令行方式启动

RLOG_SEND_THRESHOLD = 0              #指定主库发送日志到备库的时间阈值，默认关闭

RLOG_APPLY_THRESHOLD = 0             #指定备库重演日志的时间阈值，默认关闭
```

7.3.3.5 启动主库

以 Mount 方式启动主库

```
./dmserver /dm/data/DAMENG/dm.ini mount
```



注意：

一定要以 **Mount** 方式启动数据库实例，否则系统启动时会重构回滚表空间，生成 **Redo** 日志；并且，启动后应用可能连接到数据库实例进行操作，破坏主备库的数据一致性。数据守护配置结束后，守护进程会自动 **Open** 数据库。

7.3.3.6 设置 OGUID

启动命令行工具 DIsql，登录主库设置 OGUID 值。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>sp_set_oguid(453332);

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```



系统通过 OGUID 值确定一个守护进程组，由用户保证 OGUID 值的唯一性，并

注意：确保数据守护系统中，数据库、守护进程和监视器配置相同的 OGUID 值。

7.3.3.7 修改数据库模式

启动命令行工具 DIsql，登录主库修改数据库为 Primary 模式

```
SQL>alter database primary;
```

7.3.4 配置备库 GRP1_RWW_02

7.3.4.1 配置 dm.ini

在 DW_S1 机器上配置备库的实例名为 GRP1_RWW_02，dm.ini 参数修改如下：

#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16

```
INSTANCE_NAME          = GRP1_RWW_02

PORT_NUM               = 32142 #数据库实例监听端口

DW_INACTIVE_INTERVAL   = 60    #接收守护进程消息超时时间

ALTER_MODE_STATUS      = 0     #不允许手工方式修改实例模式/状态/OGUID

ENABLE_OFFLINE_TS      = 2     #不允许备库 OFFLINE 表空间

MAL_INI                = 1     #打开 MAL 系统

ARCH_INI               = 1     #打开归档配置

RLOG_SEND_APPLY_MON    = 64    #统计最近 64 次的日志重演信息
```

7.3.4.2 配置 dmmal.ini

配置 MAL 系统，各主备库的 dmmal.ini 配置必须完全一致，MAL_HOST 使用内部网络 IP，MAL_PORT 与 dm.ini 中 PORT_NUM 使用不同的端口值，MAL_DW_PORT 是各实例对应的守护进程之间，以及守护进程和监视器之间的通信端口，配置如下：

```

MAL_CHECK_INTERVAL          = 5          #MAL 链路检测时间间隔
MAL_CONN_FAIL_INTERVAL      = 5          #判定 MAL 链路断开的的时间

[MAL_INST1]

MAL_INST_NAME               = GRP1_RWW_01 #实例名，和 dm.ini 中的 INSTANCE_NAME 一致
MAL_HOST                    = 192.168.0.141 #MAL 系统监听 TCP 连接的 IP 地址
MAL_PORT                    = 61141        #MAL 系统监听 TCP 连接的端口
MAL_INST_HOST               = 192.168.1.131 #实例的对外服务 IP 地址
MAL_INST_PORT               = 32141 #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致
MAL_DW_PORT                 = 52141 #实例对应的守护进程监听 TCP 连接的端口
MAL_INST_DW_PORT            = 33141 #实例监听守护进程 TCP 连接的端口

[MAL_INST2]

MAL_INST_NAME               = GRP1_RWW_02
MAL_HOST                    = 192.168.0.142
MAL_PORT                    = 61142
MAL_INST_HOST               = 192.168.1.132
MAL_INST_PORT               = 32142
MAL_DW_PORT                 = 52142
MAL_INST_DW_PORT            = 33142

[MAL_INST3]

MAL_INST_NAME               = GRP1_RWW_03
MAL_HOST                    = 192.168.0.143
MAL_PORT                    = 61143
MAL_INST_HOST               = 192.168.1.133
MAL_INST_PORT               = 32143
MAL_DW_PORT                 = 52143

```

```
MAL_INST_DW_PORT      = 33143
```

7.3.4.3 配置 dmarch.ini

修改 dmarch.ini，配置本地归档和即时归档。

除了本地归档外，其他归档配置项中的 ARCH_DEST 表示实例是 Primary 模式时，需要同步归档数据的目标实例名。

当前实例 GRP1_RWW_02 是备库，守护系统配置完成后，可能在各种故障处理中，GRP1_RWW_02 切换为新的主库，正常情况下，GRP1_RWW_01 会切换为新的备库，需要向 GRP1_RWW_01 和 GRP1_RWW_03 同步数据，因此即时归档的 ARCH_DEST 分别配置为 GRP1_RWW_01 和 GRP1_RWW_03。

```
[ARCHIVE_TIMELY1]

ARCH_TYPE          = TIMELY          #即时归档类型

ARCH_DEST          = GRP1_RWW_01     #即时归档目标实例名

[ARCHIVE_TIMELY2]

ARCH_TYPE          = TIMELY          #即时归档类型

ARCH_DEST          = GRP1_RWW_03     #即时归档目标实例名

[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL           #本地归档类型

ARCH_DEST          = /dm/data/DAMENG/arch #本地归档文件存放路径

ARCH_FILE_SIZE     = 128             #单位 Mb，本地单个归档文件最大值

ARCH_SPACE_LIMIT   = 0               #单位 Mb，0 表示无限制，范围 1024~2147483647M
```

7.3.4.4 配置 dmwatcher.ini

修改 dmwatcher.ini 配置守护进程，配置为全局守护类型，使用自动切换模式。

```
[GRP1]

DW_TYPE            =GLOBAL           #全局守护类型

DW_MODE            = AUTO            #自动切换模式

DW_ERROR_TIME      = 10              #远程守护进程故障认定时间
```

```

INST_RECOVER_TIME = 60          #主库守护进程启动恢复的间隔时间

INST_ERROR_TIME    = 10         #本地实例故障认定时间

INST_OGUID         = 453332     #守护系统唯一 OGUID 值

INST_INI           = /dm/data/DAMENG/dm.ini    #dm.ini 配置文件路径

INST_AUTO_RESTART  = 1          #打开实例的自动启动功能

INST_STARTUP_CMD   = /dm/bin/dmserver #命令行方式启动

RLOG_SEND_THRESHOLD = 0         #指定主库发送日志到备库的时间阈值，默认关闭

RLOG_APPLY_THRESHOLD = 0       #指定备库重演日志的时间阈值，默认关闭

```

7.3.4.5 启动备库

以 Mount 方式启动备库。

```
./dmserver /dm/data/DAMENG/dm.ini mount
```



注意：

一定要以 **Mount** 方式启动数据库实例，否则系统启动时会重构回滚表空间，生成 **Redo** 日志；并且，启动后应用可能连接到数据库实例进行操作，破坏主备库的数据一致性。数据守护配置结束后，守护进程会自动 **Open** 数据库。

7.3.4.6 设置 OGUID

启动命令行工具 `DISql`，登录备库设置 OGUID 值。

```

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>sp_set_oguid(453332);

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);

```



注意：

系统通过 OGUID 值确定一个守护进程组，由用户保证 OGUID 值的唯一性，并确保数据守护系统中，数据库、守护进程和监视器配置相同的 OGUID 值。

7.3.4.7 修改数据库模式

启动命令行工具 `DISql`，登录备库修改数据库为 Standby 模式。

如果当前数据库不是 Normal 模式，需要先修改 dm.ini 中 ALTER_MODE_STATUS 值为 1，允许修改数据库模式，修改 Standby 模式成功后再改回为 0。如果是 Normal 模式，请忽略下面的第 1 步和第 3 步。

```
SQL>SP_SET PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);          ----第 1 步
SQL>alter database standby;                                   ----第 2 步
SQL>SP_SET PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);          ----第 3 步
```

7.3.5 配置备库 GRP1_RWW_03

7.3.5.1 配置 dm.ini

在 DW_S2 机器上配置备库的实例名为 GRP1_RWW_03，dm.ini 参数修改如下：

```
#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16
INSTANCE_NAME          = GRP1_RWW_03
PORT_NUM               = 32143 #数据库实例监听端口
DW_INACTIVE_INTERVAL  = 60     #接收守护进程消息超时时间
ALTER_MODE_STATUS      = 0     #不允许手工方式修改实例模式/状态/OGUID
ENABLE_OFFLINE_TS      = 2     #不允许备库 OFFLINE 表空间
MAL_INI                = 1     #打开 MAL 系统
ARCH_INI               = 1     #打开归档配置
RLOG_SEND_APPLY_MON    = 64    #统计最近 64 次的日志重演信息
```

7.3.5.2 配置 dmmal.ini

配置 MAL 系统，各主备库的 dmmal.ini 配置必须完全一致，MAL_HOST 使用内部网络 IP，MAL_PORT 与 dm.ini 中 PORT_NUM 使用不同的端口值，MAL_DW_PORT 是各实例对应的守护进程之间，以及守护进程和监视器之间的通信端口，配置如下：

```
MAL_CHECK_INTERVAL     = 5      #MAL 链路检测时间间隔
MAL_CONN_FAIL_INTERVAL = 5      #判定 MAL 链路断开的的时间
[MAL_INST1]
MAL_INST_NAME          = GRP1_RWW_01 #实例名，和 dm.ini 中的 INSTANCE_NAME 一致
```

```

MAL_HOST                = 192.168.0.141 #MAL 系统监听 TCP 连接的 IP 地址
MAL_PORT                = 61141          #MAL 系统监听 TCP 连接的端口
MAL_INST_HOST           = 192.168.1.131 #实例的对外服务 IP 地址
MAL_INST_PORT           = 32141 #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致
MAL_DW_PORT             = 52141 #实例对应的守护进程监听 TCP 连接的端口
MAL_INST_DW_PORT        = 33141 #实例监听守护进程 TCP 连接的端口

[MAL_INST2]

MAL_INST_NAME           = GRP1_RWW_02
MAL_HOST                = 192.168.0.142
MAL_PORT                = 61142
MAL_INST_HOST           = 192.168.1.132
MAL_INST_PORT           = 32142
MAL_DW_PORT             = 52142
MAL_INST_DW_PORT        = 33142

[MAL_INST3]

MAL_INST_NAME           = GRP1_RWW_03
MAL_HOST                = 192.168.0.143
MAL_PORT                = 61143
MAL_INST_HOST           = 192.168.1.133
MAL_INST_PORT           = 32143
MAL_DW_PORT             = 52143
MAL_INST_DW_PORT        = 33143

```

7.3.5.3 配置 dmarch.ini

修改 dmarch.ini，配置本地归档和即时归档。

除了本地归档外，其他归档配置项中的 ARCH_DEST 表示实例是 Primary 模式时，需要同步归档数据的目标实例名。

当前实例 GRP1_RWW_03 是备库，守护系统配置完成后，可能在各种故障处理中，GRP1_RWW_03 切换为新的主库，正常情况下，GRP1_RWW_01 会切换为新的备库，需要向

GRP1_RWW_01 和 GRP1_RWW_02 同步数据，因此即时归档的 ARCH_DEST 分别配置为 GRP1_RWW_01 和 GRP1_RWW_02。

```
[ARCHIVE_TIMELY1]

    ARCH_TYPE          = TIMELY          #即时归档类型

    ARCH_DEST          = GRP1_RWW_01    #即时归档目标实例名

[ARCHIVE_TIMELY2]

    ARCH_TYPE          = TIMELY          #即时归档类型

    ARCH_DEST          = GRP1_RWW_02    #即时归档目标实例名

[ARCHIVE_LOCAL1]

    ARCH_TYPE          = LOCAL           #本地归档类型

    ARCH_DEST          = /dm/data/DAMENG/arch #本地归档文件存放路径

    ARCH_FILE_SIZE     = 128            #单位 Mb，本地单个归档文件最大值

    ARCH_SPACE_LIMIT   = 0              #单位 Mb，0 表示无限制，范围 1024~2147483647M
```

7.3.5.4 配置 dmwatcher.ini

修改 dmwatcher.ini 配置守护进程，配置为全局守护类型，使用自动切换模式。

```
[GRP1]

    DW_TYPE            = GLOBAL          #全局守护类型

    DW_MODE            = AUTO           #自动切换模式

    DW_ERROR_TIME      = 10              #远程守护进程故障认定时间

    INST_RECOVER_TIME  = 60              #主库守护进程启动恢复的间隔时间

    INST_ERROR_TIME    = 10              #本地实例故障认定时间

    INST_OGUID         = 453332         #守护系统唯一 OGUID 值

    INST_INI           = /dm/data/DAMENG/dm.ini #dm.ini 配置文件路径

    INST_AUTO_RESTART  = 1               #打开实例的自动启动功能

    INST_STARTUP_CMD    = /dm/bin/dmserver #命令行方式启动

    RLOG_SEND_THRESHOLD = 0              #指定主库发送日志到备库的时间阈值，默认关闭

    RLOG_APPLY_THRESHOLD = 0             #指定备库重演日志的时间阈值，默认关闭
```

7.3.5.5 启动备库

以 Mount 方式启动备库

```
./dmserver /dm/data/DAMENG/dm.ini mount
```



注意:

一定要以 **Mount** 方式启动数据库实例，否则系统启动时会重构回滚表空间，生成 Redo 日志；并且，启动后应用可能连接到数据库实例进行操作，破坏主备库的数据一致性。数据守护配置结束后，守护进程会自动 **Open** 数据库。

7.3.5.6 设置 OGUID

启动命令行工具 DIsql，登录备库设置 OGUID 值。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>sp_set_oguid(453332);

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```



注意: 确保数据守护系统中，数据库、守护进程和监视器配置相同的 OGUID 值。

7.3.5.7 修改数据库模式

启动命令行工具 DIsql，登录备库修改数据库为 Standby 模式

如果当前数据库不是 Normal 模式，需要先修改 dm.ini 中 ALTER_MODE_STATUS 值为 1，允许修改数据库模式，修改 Standby 模式成功后再改回为 0。如果是 Normal 模式，请忽略下面的第 1 步和第 3 步。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);          ----第 1 步

SQL>alter database standby;                                   ----第 2 步

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);          ----第 3 步
```

7.3.6 配置监视器

配置监视器时可以选择配置单实例监视器或配置多实例监视器，下面将分别介绍配置单实例监视器以及多实例监视器时 `dmmonitor.ini` 配置文件中的配置信息。

7.3.6.1 配置单实例监视器

由于主库和实时备库的守护进程配置为自动切换模式，因此这里选择配置确认监视器。和普通监视器相比，确认监视器除了相同的命令支持外，在主库发生故障时，能够自动通知实时备库接管为新的主库，具有自动故障处理的功能。



故障自动切换模式下，必须配置确认监视器，且确认监视器最多只能配置一

注意： 个。

修改 `dmmonitor.ini` 配置确认监视器，其中 `MON_DW_IP` 中的 IP 和 PORT 和 `dmmal.ini` 中的 `MAL_HOST` 和 `MAL_DW_PORT` 配置项保持一致。

```
MON_DW_CONFIRM          = 1      #确认监视器模式

MON_LOG_PATH            = /dm/data/log      #监视器日志文件存放路径

MON_LOG_INTERVAL        = 60      #每隔 60s 定时记录系统信息到日志文件

MON_LOG_FILE_SIZE       = 32      #每个日志文件最大 32M

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

[GRP1]

    MON_INST_OGUID       = 453332 #组 GRP1 的唯一 OGUID 值

#以下为监视器到组 GRP1 的守护进程的连接信息，以“IP:PORT”的形式配置

#IP 对应 dmmal.ini 中的 MAL_HOST，PORT 对应 dmmal.ini 中的 MAL_DW_PORT

    MON_DW_IP            = 192.168.0.141:52141

    MON_DW_IP            = 192.168.0.142:52142

    MON_DW_IP            = 192.168.0.143:52143
```

7.3.6.2 配置多实例监视器

与配置单实例监视器相比，配置多实例监视器时 `dmmonitor.ini` 中需要添加整个监

监视器系统中的实例总数 `MON_INST_NUM`（取值为 1、3 或者 5），各实例通信的心跳校验间隔 `MON_HB_INTERVAL`（取值（5~600），单位秒），各实例通信的心跳间隔 `MON_BRO_INTERVAL`（取值（5~600），单位毫秒），各实例通信的基础投票间隔 `MON_VOTE_INTERVAL`（取值（5~600），单位毫秒），各实例通信的心跳间隔以及基础投票间隔会在真正使用时进行调整，各实例通信心跳间隔需要比基础投票间隔小（若配置中各实例通信心跳间隔比基础投票间隔大，内部会将实例通信心跳间隔调整为与基础投票间隔相同，再进行其他的调整处理），增加监视器 ID 即 `MON_ID`（用于区别所有监视器中的不同实例，每个监视器实例的 `MON_ID` 与 `MON_INST_ID` 相同，取值（1~2147483647）），增加监视器系统唯一标识 `MON_MID`（取值（1~2147483647）），多实例监视器系统中的所有监视器需配置相同的 `MON_MID`。

与配置单实例监视器相比，配置多实例监视器时组的配置信息不做更改，监视器实例的配置信息需要新增配置参数，通过组和实例的配置信息内容的不同来区分组和实例，如果用户将组和实例的配置信息混淆，则以配置信息中更加靠前的参数为准来决定当前配置属于组还是实例，不属于组或实例的配置项将会被忽略。

监视器实例的配置信息包括实例 IP 地址、监听端口号以及实例在系统中的唯一编号，实例的 IP 地址和监听端口号不应出现冲突，否则该监视器实例可能无法正常使用。所有监视器实例的 `dmmonitor.ini` 配置文件中除 `MON_ID` 参数项外应完全一致，每个监视器的 `MON_ID` 应互不相同，用于区分同一个监视器系统中的不同监视器实例，若配置相同的 `MON_ID` 将导致通信无法正常建立。监视器实例监听的 IP 及端口号的正确性由用户保证。

配置多实例监视器时 `dmmonitor.ini` 配置文件中的内容如下：

```
MON_LOG_PATH           = /dm/data/log
MON_LOG_INTERVAL       = 60
MON_LOG_FILE_SIZE      = 32
MON_LOG_SPACE_LIMIT    = 0
MON_DW_CONFIRM         = 1
MON_INST_NUM           = 3           #实例总个数
MON_HB_INTERVAL        = 60         #通信心跳校验间隔
MON_BRO_INTERVAL       = 100        #raft 协议中实例通信心跳间隔
MON_VOTE_INTERVAL      = 100        #raft 协议中基础投票间隔
MON_ID                 = 1           #当前监视器在监视器系统中的 ID
```

```
MON_MID                = 45614      #当前监视器系统的唯一标识

[GRP1]

MON_INST_OGUID         = 453332

MON_DW_IP              = 192.168.0.141:52141

MON_DW_IP              = 192.168.0.142:52142

MON_DW_IP              = 192.168.0.143:52143

[GRP2]

...

[GRP3]

...

[MON1]

MON_HOST               = 192.168.0.141      #系统监听 TCP 连接的 IP 地址

MON_PORT               = 8339              #系统监听 TCP 连接的端口号

MON_INST_ID            = 1                 #监视器实例在监视器系统中的 ID

[MON2]

...

[MON3]
```

7.3.7 启动守护进程

启动各个主备库上的守护进程：

```
./dmwatcher /dm/data/DAMENG/dmwatcher.ini
```

守护进程启动后，进入 Startup 状态，此时实例都处于 Mount 状态。守护进程开始广播自身和其监控实例的状态信息，结合自身信息和远程守护进程的广播信息，守护进程将本地实例 Open，并切换为 Open 状态。

7.3.8 启动监视器

启动监视器：

```
./dmmonitor /dm/data/dmmonitor.ini
```

监视器提供一系列命令，支持当前守护系统状态查看以及故障处理，可输入 help 命令，查看各种命令使用说明，结合实际情况选择使用。

至此读写分离集群搭建完毕，在搭建步骤和各项配置都正确的情况下，在监视器上执行 show 命令，可以监控到所有实例都处于 Open 状态，所有守护进程也都处于 Open 状态，即为正常运行状态。

7.3.9 接口说明

DM 多种客户端接口都支持读写分离集群连接设置，以下说明客户端连接服务器时如何设置读写分离属性，详细可参考《DM8 程序员手册》。

7.3.9.1 JDBC 接口

在 JDBC 连接串中增加了两个连接属性：

- `rwSeparate` 是否使用读写分离系统，默认 0；取值（0 不使用，1 使用）。
- `rwPercent` 分发到主库的事务占主备库总事务的百分比，有效值 0~100，默认值 25。

```
<DRIVER>dm.jdbc.driver.DmDriver</DRIVER>

<URL>jdbc:dm://192.168.0.206:5236?rwSeparate=1&rwPercent=10</URL>
```

7.3.9.2 DPI 接口

DPI 接口的连接句柄上可设置读写分离属性：

- `DSQL_ATTR_RWSEPARATE`：读写分离（可读写）
- `DSQL_ATTR_RWSEPARATE_PERCENT`：读写分离比例（可读写）

属性设置举例：

```
dhenv          env;

dhcon          con;

dpi_alloc_env(&env);

dpi_alloc_con(env, &con);
```

```
dpi_set_con_attr(con, DSQL_ATTR_RWSEPARATE, (dpointer) DSQL_RWSEPARATE_ON, 0);  
dpi_set_con_attr(con, DSQL_ATTR_RWSEPARATE_PERCENT, (dpointer)25, 0);
```

Disql 工具可以直接设置读写分离的属性:

```
>disql /nolog  
SQL> login  
服务名:  
用户名:  
密码:  
端口号:  
SSL 路径:  
SSL 密码:  
UKEY 名称:  
UKEY PIN 码:  
MPP 类型:  
是否读写分离 (y/n):y  
读写分离百分比 (0-100):25
```

7.3.9.3 ODBC 接口

ODBC 接口中读写分离相关的连接关键字为:

- RW_SEPARATE: 是否配置读写分离: TRUE, FALSE
- RW_SEPARATE_PERCENT: 读写分离的比例: 0~100

连接串举例说明:

```
"DSN=DM8;DRIVER=DM ODBC  
DRIVER;UID=SYSDBA;PWD=SYSDBA;TCP_PORT=5236;RW_SEPARATE=TRUE;RW_SEPARATE_PERC  
ENT=25";
```

7.3.9.4 Provider 接口

DM .NET Provider 接口主要实现了 DmConnection，DmConnection 对象表示一个 DM 数据库打开的连接。其支持的读写分离属性包括：

- RW_Separate：是否读写分离，有效值为 true 或者 false；
- RW_Percent：表示分发到主库的事务占主备库总事务的百分比，有效值范围：0~100，默认值为 25。

连接串举例：

```
static DmConnection cnn = new DmConnection();

cnn.ConnectionString = "Server=localhost; User Id=SYSDBA; PWD=SYSDBA;
RW_Separate=true; RW_Percent=25";
```

7.3.9.5 DCI 接口

DCI 接口支持会话上的读写分离属性设置：

- OCI_ATTR_RW_SEPARATE 是否读写分离，有效值为 1 或者 0，默认 0；
- OCI_ATTR_RW_SEPARATE_PERCENT 表示分发到主库的事务占主备库总事务的百分比，有效值范围：0~100，默认值为 25。

例如，利用接口编程进行举例说明。

```
OCIEnv*      envhp;

OCISession*   authp;

OCIError*     errhp;

OCIInitialize(OCI_DEFAULT, NULL, NULL, NULL, NULL);

OCIEnvInit(&envhp, OCI_DEFAULT, 0, 0);

OCIHandleAlloc(envhp, (dvoid**)&authp, OCI_HTYPE_SESSION, 0, 0);

OCIAttrSet(authp, OCI_HTYPE_SESSION, (void
*)OCI_ATTR_RW_SEPARATE_ON, (ub4)sizeof(ub4), OCI_ATTR_RW_SEPARATE, errhp);

OCIAttrSet(authp, OCI_HTYPE_SESSION, (void
*)25, (ub4)sizeof(ub4), OCI_ATTR_RW_SEPARATE_PERCENT, errhp);
```

7.4 配置 MPP 主备

为了提高 MPP 系统可靠性，克服由于单节点故障导致整个系统不能继续正常工作，DM 在普通的 MPP 系统基础上，引入主备守护机制，将 MPP 节点作为主库节点，增加备库作为备份节点，必要时可切换为主库代替故障节点工作，提高系统的可靠性和可用性。

7.4.1 环境说明

本例配置 2 个 MPP 节点，每个节点作为主库，与其备库组成一个守护组，因此需要配置两个守护组，取名分别为 GRP1、GRP2，主库名为 GRP1_MPP_EP01/GRP2_MPP_EP02，对应的备库实例名分别为 GRP1_MPP_EP11/GRP2_MPP_EP22。

准备 3 台机器 A、B、C，A 和 B 用来交叉部署实例，C 用来部署监视器。其中 A 和 B 配置两块网卡，一块接入内部网络交换模块，一块接入到外部交换机，C 接入内部网络。

机器事先都安装了 DM，安装路径为 '/dm'，执行程序保存在 '/dm/bin' 目录中，数据存放路径为 '/dm/data/EP01'，'/dm/data/EP02'。



注意：

实际配置时，相关的端口配置和 OGUID 值建议不要和手册示例使用完全相同的值，避免多个用户在同一个环境下搭建不同的数据守护系统，出现消息混乱或者端口冲突等问题。

表 7.5 配置环境说明

机器名	IP 地址	初始状态	操作系统
A	192.168.1.131	主库 GRP1_MPP_EP01	Linux
	192.168.0.141	备库 GRP2_MPP_EP22	rh6-2.6.32-220.el6.x 86_64
B	192.168.1.132	主库 GRP2_MPP_EP02	Linux
	192.168.0.142	备库 GRP1_MPP_EP11	rh6-2.6.32-220.el6.x 86_64
C	192.168.0.144	监视器	Linux rh6-2.6.32-220.el6.x

			86_64
--	--	--	-------

表 7.6 端口规划-主库

实例名	PORT_NUM	MAL_INST_DW_PORT	MAL_HOST	MAL_PORT	MAL_DW_PORT	MPP 实例序号
GRP1_MPP_EP01	5236	5243	192.168.0.141	5337	5253	0
GRP2_MPP_EP02	5236	5243	192.168.0.142	5337	5253	1

表 7.7 端口规划-备库

实例名	PORT_NUM	MAL_INST_DW_PORT	MAL_HOST	MAL_PORT	MAL_DW_PORT	对应主库
	M	T		T	T	
GRP1_MPP_EP1	5237	5244	192.168.0.14	5338	5254	GRP1_MPP_EP0
1			2			1
GRP2_MPP_EP2	5237	5244	192.168.0.14	5338	5254	GRP2_MPP_EP0
2			1			2

表 7.8 守护进程规划

组名	实例名	所在机器
GRP1	GRP1_MPP_EP01	192.168.0.141
	GRP1_MPP_EP11	192.168.0.142
GRP2	GRP2_MPP_EP02	192.168.0.142
	GRP2_MPP_EP22	192.168.0.141

7.4.2 数据准备

A 机器上初始化库至目录 /dm/data/EP01:

```
./dmnit path=/dm/data/EP01
```

B 机器上初始化库至目录 /dm/data/EP02:

```
./dmnit path=/dm/data/EP02
```

即完成两个主数据库的初始化，然后按照 7.1 数据准备中的方法分别同步两个备数据库。

本例中采取机器交叉的方式配置两个备数据库，分别对应存放的目录为：

B 机器的 /dm/data/EP01

A 机器的 /dm/data/EP02

7.4.3 配置主库 GRP1_MPP_EP01

7.4.3.1 配置 dm.ini

在 A 机器上配置主库的实例名为 GRP1_MPP_EP01，dm.ini 参数修改如下：

```
#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16
INSTANCE_NAME          = GRP1_MPP_EP01

PORT_NUM               = 5236   #数据库实例监听端口

DW_INACTIVE_INTERVAL   = 60     #接收守护进程消息超时时间

ALTER_MODE_STATUS      = 0      #不允许手工方式修改实例模式/状态/OGUID

ENABLE_OFFLINE_TS      = 2      #不允许备库 OFFLINE 表空间

MAL_INI                = 1      #打开 MAL 系统

ARCH_INI               = 1      #打开归档配置

MPP_INI                = 1      #启用 MPP 配置

RLOG_SEND_APPLY_MON    = 64     #统计最近 64 次的日志发送信息
```

7.4.3.2 配置 dmmal.ini

配置 MAL 系统，各主备库的 dmmal.ini 配置必须完全一致，MAL_HOST 使用内部网络 IP，MAL_PORT 与 dm.ini 中 PORT_NUM 使用不同的端口值，MAL_DW_PORT 是各实例对应的守护进程之间，以及守护进程和监视器之间的通信端口，配置如下：

```
MAL_CHECK_INTERVAL     = 5       #MAL 链路检测时间间隔

MAL_CONN_FAIL_INTERVAL = 5       #判定 MAL 链路断开的时间

[MAL_INST1]

MAL_INST_NAME          = GRP1_MPP_EP01 #实例名，和 dm.ini 中的 INSTANCE_NAME 一致

MAL_HOST               = 192.168.0.141 #MAL 系统监听 TCP 连接的 IP 地址

MAL_PORT               = 5337       #MAL 系统监听 TCP 连接的端口

MAL_INST_HOST          = 192.168.1.131 #实例的对外服务 IP 地址
```

```
MAL_INST_PORT          = 5236  #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致

MAL_DW_PORT            = 5253  #实例对应的守护进程监听 TCP 连接的端口

MAL_INST_DW_PORT       = 5243  #实例监听守护进程 TCP 连接的端口

[MAL_INST2]

MAL_INST_NAME          = GRP2_MPP_EP02

MAL_HOST               = 192.168.0.142

MAL_PORT               = 5337

MAL_INST_HOST          = 192.168.1.132

MAL_INST_PORT          = 5236

MAL_DW_PORT            = 5253

MAL_INST_DW_PORT       = 5243

[MAL_INST3]

MAL_INST_NAME          = GRP1_MPP_EP11

MAL_HOST               = 192.168.0.142

MAL_PORT               = 5338

MAL_INST_HOST          = 192.168.1.132

MAL_INST_PORT          = 5237

MAL_DW_PORT            = 5254

MAL_INST_DW_PORT       = 5244

[MAL_INST4]

MAL_INST_NAME          = GRP2_MPP_EP22

MAL_HOST               = 192.168.0.141

MAL_PORT               = 5338

MAL_INST_HOST          = 192.168.1.131

MAL_INST_PORT          = 5237

MAL_DW_PORT            = 5254

MAL_INST_DW_PORT       = 5244
```

7.4.3.3 配置 dmarch.ini（实时归档）

修改 dmarch.ini，配置实时归档。

除了本地归档外，其他归档配置项中的 ARCH_DEST 表示实例是 Primary 模式时，需要同步归档数据的目标实例名。

当前实例 GRP1_MPP_EP01 是主库，需要向 MPP 备库 GRP1_MPP_EP11 同步数据，因此实时归档的 ARCH_DEST 配置为 GRP1_MPP_EP11。

```
[ARCHIVE_REALTIME1]

ARCH_TYPE          = REALTIME          #实时归档类型

ARCH_DEST           = GRP1_MPP_EP11     #实时归档目标实例名

[ARCHIVE_LOCAL1]

ARCH_TYPE           = LOCAL             #本地归档类型

ARCH_DEST           = /dm/data/EP01/DAMENG/arch #本地归档文件存放路径

ARCH_FILE_SIZE      = 128               #单位 Mb，本地单个归档文件最大值

ARCH_SPACE_LIMIT    = 0                 #单位 Mb，0 表示无限制，范围 1024~4294967294M
```

7.4.3.4 配置 dmmpp.ctl

dmmpp.ctl 是二进制文件，由 dmmpp.ini 文本通过 dmctlcvrt 工具转换而来，dmmpp.ini 配置项如下表：

表 7.9 dmmpp.ini 配置项

配置项	配置含义
[SERVICE_NAME]	标识每个实例的选项名
MPP_SEQ_NO	实例在 MPP 系统内的序号
MPP_INST_NAME	节点实例名

本例中两节点的 dmmpp.ini 配置如下：

```
[service_name1]

mpp_seq_no         = 0

mpp_inst_name      = GRP1_MPP_EP01

[service_name2]
```

```
mpp_seq_no      = 1
mpp_inst_name    = GRP2_MPP_EP02
```

转换命令如下：

```
./dmctlcvt          TYPE=2          SRC=/dm/data/EP01/DAMENG/dmmp.ini
DEST=/dm/data/EP01/DAMENG/dmmp.ct1
```

7.4.3.5 启动主库

以 Mount 方式启动主库。

```
./dmserver /dm/data/EP01/DAMENG/dm.ini mount
```



注意：

一定要以 **Mount** 方式启动数据库实例，否则系统启动时会重构回滚表空间，生成 Redo 日志；并且，启动后应用可能连接到数据库实例进行操作，破坏主备库的数据一致性。数据守护配置结束后，守护进程会自动 **Open** 数据库。

7.4.3.6 设置 OGUID

启动命令行工具 DIsql，使用 MPP 类型为 LOCAL 方式，登录主库设置 OGUID 值。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);
SQL>sp_set_oguid(45330);
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```



注意：

系统通过 OGUID 值确定一个守护进程组，由用户保证 OGUID 值的唯一性，并确保数据守护系统中，数据库、守护进程和监视器配置相同的 OGUID 值。

7.4.3.7 修改数据库模式

启动命令行工具 DIsql，使用 MPP 类型为 LOCAL 方式，登录主库修改数据库为 Primary 模式。

```
SQL>alter database primary;
```

7.4.4 配置主库 GRP2_MPP_EP02

7.4.4.1 配置 dm.ini

在 B 机器上配置主库的实例名为 GRP2_MPP_EP02，dm.ini 参数修改如下：

#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16

```

INSTANCE_NAME          = GRP2_MPP_EP02

PORT_NUM               = 5236   #数据库实例监听端口

DW_INACTIVE_INTERVAL   = 60     #接收守护进程消息超时时间

ALTER_MODE_STATUS      = 0      #不允许手工方式修改实例模式/状态/OGUID

ENABLE_OFFLINE_TS      = 2      #不允许备库 OFFLINE 表空间

MAL_INI                = 1      #打开 MAL 系统

ARCH_INI               = 1      #打开归档配置

MPP_INI                = 1      #启用 MPP 配置

RLOG_SEND_APPLY_MON    = 64     #统计最近 64 次的日志发送信息

```

7.4.4.2 配置 dmmal.ini

直接将 A 机器上实例 GRP1_MPP_EP01 配置的 dmmal.ini 拷贝到 /dm/data/EP02/DAMENG 目录中。

7.4.4.3 配置 dmarch.ini（实时归档）

修改 dmarch.ini，配置实时归档。

除了本地归档外，其他归档配置项中的 ARCH_DEST 表示实例是 Primary 模式时，需要同步归档数据的目标实例名。

当前实例 GRP2_MPP_EP02 是主库，需要向 MPP 备库 GRP2_MPP_EP22 同步数据，因此实时归档的 ARCH_DEST 配置为 GRP2_MPP_EP22。

```

[ARCHIVE_REALTIME1]

ARCH_TYPE              = REALTIME      #实时归档类型

ARCH_DEST              = GRP2_MPP_EP22 #实时归档目标实例名

```

```
[ARCHIVE_LOCAL1]
```

```

ARCH_TYPE           = LOCAL #本地归档类型

ARCH_DEST           = /dm/data/EP02/DAMENG/arch #本地归档文件存放路径

ARCH_FILE_SIZE      = 128   #单位 Mb，本地单个归档文件最大值

ARCH_SPACE_LIMIT    = 0     #单位 Mb，0 表示无限制，范围 1024~4294967294M

```

7.4.4.4 配置 dmmpp.ctl

同 7.4.3.4 配置 `dmmpp.ctl`，拷贝 `dmmpp.ctl` 到 `/dm/data/EP02/DAMENG` 目录即可。

7.4.4.5 启动主库

以 Mount 方式启动主库

```
./dmserver /dm/data/EP02/DAMENG/dm.ini mount
```



注意：

一定要以 **Mount** 方式启动数据库实例，否则系统启动时会重构回滚表空间，生成 Redo 日志；并且，启动后应用可能连接到数据库实例进行操作，破坏主备库的数据一致性。数据守护配置结束后，守护进程会自动 **Open** 数据库。

7.4.4.6 设置 OGUID

启动命令行工具 `DISql`，使用 MPP 类型为 `LOCAL` 方式，登录主库设置 `OGUID` 值。

```

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>sp_set_oguid(45331);

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);

```



注意：

系统通过 **OGUID** 值确定一个守护进程组，由用户保证 **OGUID** 值的唯一性，并确保数据守护系统中，数据库、守护进程和监视器配置相同的 **OGUID** 值。

7.4.4.7 修改数据库模式

启动命令行工具 DISql，使用 MPP 类型为 LOCAL 方式，登录主库修改数据库为 Primary 模式。

```
SQL>alter database primary;
```

7.4.5 配置备库 GRP1_MPP_EP11

7.4.5.1 配置 dm.ini

在 B 机器上配置备库的实例名为 GRP1_MPP_EP11，dm.ini 参数修改如下：

```
#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16
INSTANCE_NAME          = GRP1_MPP_EP11

PORT_NUM               = 5237  #数据库实例监听端口

DW_INACTIVE_INTERVAL   = 60    #接收守护进程消息超时时间

ALTER_MODE_STATUS      = 0      #不允许手工方式修改实例模式/状态/OGUID

ENABLE_OFFLINE_TS      = 2      #不允许备库 OFFLINE 表空间

MAL_INI                = 1      #打开 MAL 系统

ARCH_INI               = 1      #打开归档配置

MPP_INI                = 1      #打开 MPP 配置

RLOG_SEND_APPLY_MON    = 64     #统计最近 64 次的日志重演信息
```

7.4.5.2 配置 dmmal.ini

直接将 A 机器上实例 GRP1_MPP_EP01 配置的 dmmal.ini 拷贝到 /dm/data/EP01/DAMENG 目录中。

7.4.5.3 配置 dmarch.ini（实时归档）

修改 dmarch.ini，配置实时归档。

除了本地归档外，其他归档配置项中的 ARCH_DEST 表示实例是 Primary 模式时，需

要同步归档数据的目标实例名。

当前实例 GRP1_MPP_EP11 是备库，守护系统配置完成后，可能在各种故障处理中，GRP1_MPP_EP11 切换为新的主库，正常情况下，GRP1_MPP_EP01 会切换为新的备库，需要向 GRP1_MPP_EP01 同步数据，因此实时归档的 ARCH_DEST 配置为 GRP1_MPP_EP01。

```
[ARCHIVE_REALTIME1]

    ARCH_TYPE          = REALTIME          #实时归档类型

    ARCH_DEST           = GRP1_MPP_EP01     #实时归档目标实例名

[ARCHIVE_LOCAL1]

    ARCH_TYPE          = LOCAL              #本地归档类型

    ARCH_DEST           = /dm/data/EP01/DAMENG/arch #本地归档文件存放路径

    ARCH_FILE_SIZE      = 128              #单位 Mb，本地单个归档文件最大值

    ARCH_SPACE_LIMIT    = 0                #单位 Mb，0 表示无限制，范围 1024~4294967294M
```

7.4.5.4 配置 dmmpp.ctl

数据守护 V2.1 及以上版本中，MPP 备库同样需要配置 dmmpp.ctl 文件，可以直接从主库上拷贝。

本例中将 A 机器上实例 GRP1_MPP_EP01 配置的 dmmpp.ctl 拷贝到 /dm/data/EP01/DAMENG 目录中。

7.4.5.5 启动备库

以 Mount 方式启动备库

```
./dmserver /dm/data/EP01/DAMENG/dm.ini mount
```



注意：

一定要以 **Mount** 方式启动数据库实例，否则系统启动时会重构回滚表空间，生成 Redo 日志；并且，启动后应用可能连接到数据库实例进行操作，破坏主备库的数据一致性。数据守护配置结束后，守护进程会自动 **Open** 数据库。

7.4.5.6 设置 OGUID

启动命令行工具 DIsql，登录备库设置 OGUID 值。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>sp_set_oguid(45330);

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```



系统通过 OGUID 值确定一个守护进程组，由用户保证 OGUID 值的唯一性，并

注意：确保数据守护系统中，数据库、守护进程和监视器配置相同的 OGUID 值。

7.4.5.7 修改数据库模式

启动命令行工具 DIsql，登录实例修改数据库为 Standby 模式。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>ALTER DATABASE STANDBY;

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

7.4.6 配置备库 GRP2_MPP_EP22

7.4.6.1 配置 dm.ini

在 A 机器上配置备库的实例名为 GRP2_MPP_EP22，dm.ini 参数修改如下：

#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16

```
INSTANCE_NAME          = GRP2_MPP_EP22

PORT_NUM                = 5237  #数据库实例监听端口

DW_INACTIVE_INTERVAL   = 60    #接收守护进程消息超时时间

ALTER_MODE_STATUS      = 0     #不允许手工方式修改实例模式/状态/OGUID

ENABLE_OFFLINE_TS      = 2     #不允许备库 OFFLINE 表空间

MAL_INI                 = 1     #打开 MAL 系统

ARCH_INI                = 1     #打开归档配置
```

```

MPP_INI                = 1      #打开 MPP 配置
RLOG_SEND_APPLY_MON    = 64     #统计最近 64 次的日志重演信息

```

7.4.6.2 配置 dmmal.ini

直接将 A 机器上实例 GRP1_MPP_EP01 配置的 dmmal.ini 拷贝到 /dm/data/EP02/DAMENG 目录中。

7.4.6.3 配置 dmarch.ini

修改 dmarch.ini，配置实时归档。

除了本地归档外，其他归档配置项中的 ARCH_DEST 表示实例是 Primary 模式时，需要同步归档数据的目标实例名。

当前实例 GRP2_MPP_EP22 是备库，守护系统配置完成后，可能在各种故障处理中，GRP2_MPP_EP22 切换为新的主库，正常情况下，GRP2_MPP_EP02 会切换为新的备库，需要向 GRP2_MPP_EP02 同步数据，因此实时归档的 ARCH_DEST 配置为 GRP2_MPP_EP02。

```

[ARCHIVE_REALTIME1]

    ARCH_TYPE                = REALTIME          #实时归档类型

    ARCH_DEST                 = GRP2_MPP_EP02     #实时归档目标实例名

[ARCHIVE_LOCAL1]

    ARCH_TYPE                 = LOCAL             #本地归档类型

    ARCH_DEST                  = /dm/data/EP02/DAMENG/arch    #本地归档文件存放路径

    ARCH_FILE_SIZE             = 128             #单位 Mb，本地单个归档文件最大值

    ARCH_SPACE_LIMIT           = 0                #单位 Mb，0 表示无限制，范围 1024~4294967294M

```

7.4.6.4 配置 dmmppctl

数据守护 V2.1 及以上版本中，MPP 备库同样需要配置 dmmppctl 文件，可以直接从主库上拷贝。

本例中将 A 机器上实例 GRP1_MPP_EP01 配置的 dmmppctl 拷贝到

/dm/data/EP02/DAMENG 目录中。

7.4.6.5 启动备库

以 Mount 方式启动实例。

```
./dmserver /dm/data/EP02/DAMENG/dm.ini mount
```



注意：一定要以 **Mount** 方式启动数据库实例，否则系统启动时会重构回滚表空间，生成 Redo 日志；并且，启动后应用可能连接到数据库实例进行操作，破坏主备库的数据一致性。数据守护配置结束后，守护进程会自动 **Open** 数据库。

7.4.6.6 设置 OGUID

启动命令行工具 DIsql，登录实例设置 OGUID 值。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);  
  
SQL>sp_set_oguid(45331);  
  
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```



注意：系统通过 OGUID 值确定一个守护进程组，由用户保证 OGUID 值的唯一性，并确保数据守护系统中，数据库、守护进程和监视器配置相同的 OGUID 值。

7.4.6.7 修改数据库模式

启动命令行工具 DIsql，登录实例修改数据库为 Standby 模式：

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);  
  
SQL>ALTER DATABASE STANDBY;  
  
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

7.4.7 配置 dmwatcher.ini

一般来说，每个单独的实例都是使用一个单独的守护进程守护。比如前面的实时主备和

读写分离集群的配置。

本例中，由于同一台机器上有不同组的两个实例，我们可以只配置一个守护进程，同时守护两个实例。

A 机器上配置 `dmwatcher.ini`，配置为全局守护类型，使用自动切换模式。

```
[GRP1]

DW_TYPE           = GLOBAL  #全局守护类型
DW_MODE           = AUTO   #自动切换模式
DW_ERROR_TIME     = 10     #远程守护进程故障认定时间
INST_RECOVER_TIME = 60     #主库守护进程启动恢复的间隔时间
INST_ERROR_TIME   = 10     #本地实例故障认定时间
INST_OGUID        = 45330  #守护系统唯一 OGUID 值
INST_INI          = /dm/data/EP01/DAMENG/dm.ini #dm.ini 配置文件路径
INST_AUTO_RESTART = 1      #打开实例的自动启动功能
INST_STARTUP_CMD  = /dm/bin/dmserver  #命令行方式启动
RLOG_SEND_THRESHOLD = 0    #指定主库发送日志到备库的时间阈值，默认关闭
RLOG_APPLY_THRESHOLD = 0   #指定备库重演日志的时间阈值，默认关闭

[GRP2]

DW_TYPE           = GLOBAL  #全局守护类型
DW_MODE           = AUTO   #自动切换模式
DW_ERROR_TIME     = 10     #远程守护进程故障认定时间
INST_RECOVER_TIME = 60     #主库守护进程启动恢复的间隔时间
INST_ERROR_TIME   = 10     #本地实例故障认定时间
INST_OGUID        = 45331  #守护系统唯一 OGUID 值
INST_INI          = /dm/data/EP02/DAMENG/dm.ini #dm.ini 配置文件路径
INST_AUTO_RESTART = 1      #打开实例的自动启动功能
INST_STARTUP_CMD  = /dm/bin/dmserver #命令行方式启动
RLOG_SEND_THRESHOLD = 0    #指定主库发送日志到备库的时间阈值，默认关闭
RLOG_APPLY_THRESHOLD = 0   #指定备库重演日志的时间阈值，默认关闭
```

B 机器上配置 `dmwatcher.ini`，配置为全局守护类型，使用自动切换模式。

```
[GRP1]
```

```

DW_TYPE          = GLOBAL  #全局守护类型

DW_MODE          = AUTO    #自动切换模式

DW_ERROR_TIME    = 10      #远程守护进程故障认定时间

INST_RECOVER_TIME = 60      #主库守护进程启动恢复的间隔时间

INST_ERROR_TIME   = 10      #本地实例故障认定时间

INST_OGUID       = 45330   #守护系统唯一 OGUID 值

INST_INI         = /dm/data/EP01/DAMENG/dm.ini #dm.ini 配置文件路径

INST_AUTO_RESTART = 1       #打开实例的自动启动功能

INST_STARTUP_CMD  = /dm/bin/dmserver  #命令行方式启动

RLOG_SEND_THRESHOLD = 0      #指定主库发送日志到备库的时间阈值，默认关闭

RLOG_APPLY_THRESHOLD = 0     #指定备库重演日志的时间阈值，默认关闭

```

[GRP2]

```

DW_TYPE          = GLOBAL  #全局守护类型

DW_MODE          = AUTO    #自动切换模式

DW_ERROR_TIME    = 10      #远程守护进程故障认定时间

INST_RECOVER_TIME = 60      #主库守护进程启动恢复的间隔时间

INST_ERROR_TIME   = 10      #本地实例故障认定时间

INST_OGUID       = 45331   #守护系统唯一 OGUID 值

INST_INI         = /dm/data/EP02/DAMENG/dm.ini #dm.ini 配置文件路径

INST_AUTO_RESTART = 1       #打开实例的自动启动功能

INST_STARTUP_CMD  = /dm/bin/dmserver  #命令行方式启动

RLOG_SEND_THRESHOLD = 0      #指定主库发送日志到备库的时间阈值，默认关闭

RLOG_APPLY_THRESHOLD = 0     #指定备库重演日志的时间阈值，默认关闭

```

7.4.8 配置监视器

配置监视器时可以选择配置单实例监视器或配置多实例监视器，下面将分别介绍配置单实例监视器以及多实例监视器时 dmmonitor.ini 配置文件中的配置信息。

7.4.8.1 配置单实例监视器

由于主库和实时备库的守护进程配置为自动切换模式，因此这里选择配置确认监视器。和普通监视器相比，确认监视器除了相同的命令支持外，在主库发生故障时，能够自动通知实时备库接管为新的主库，具有自动故障处理的功能。



故障自动切换模式下，必须配置确认监视器，且确认监视器最多只能配置一个。

注意： 个。

修改 `dmmonitor.ini` 配置确认监视器，其中 `MON_DW_IP` 中的 IP 和 PORT 和 `dmmal.ini` 中的 `MAL_HOST` 和 `MAL_DW_PORT` 配置项保持一致。

```
MON_DW_CONFIRM          = 1      #确认监视器模式

MON_LOG_PATH            = /dm/data/log      #监视器日志文件存放路径

MON_LOG_INTERVAL        = 60      #每隔 60s 定时记录系统信息到日志文件

MON_LOG_FILE_SIZE       = 32      #每个日志文件最大 32M

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

[GRP1]

    MON_INST_OGUID       = 45330 #组 GRP1 的唯一 OGUID 值

#以下配置为监视器到组 GRP1 的守护进程的连接信息，以“IP:PORT”的形式配置

#IP 对应 dmmal.ini 中的 MAL_HOST，PORT 对应 dmmal.ini 中的 MAL_DW_PORT

    MON_DW_IP            = 192.168.0.141:5253

    MON_DW_IP            = 192.168.0.142:5254

[GRP2]

    MON_INST_OGUID       = 45331      #组 GRP2 的唯一 OGUID 值

#以下配置为监视器到组 GRP2 的守护进程的连接信息，以“IP:PORT”的形式配置

#IP 对应 dmmal.ini 中的 MAL_HOST，PORT 对应 dmmal.ini 中的 MAL_DW_PORT

    MON_DW_IP            = 192.168.0.142:5253

    MON_DW_IP            = 192.168.0.141:5254
```

7.4.8.2 配置多实例监视器

与配置单实例监视器相比，配置多实例监视器时 `dmmonitor.ini` 中需要添加整个监

监视器系统中的实例总数 `MON_INST_NUM`（取值为 1、3 或者 5），各实例通信的心跳校验间隔 `MON_HB_INTERVAL`（取值（5~600），单位秒），各实例通信的心跳间隔 `MON_BRO_INTERVAL`（取值（5~600），单位毫秒），各实例通信的基础投票间隔 `MON_VOTE_INTERVAL`（取值（5~600），单位毫秒），各实例通信的心跳间隔以及基础投票间隔会在真正使用时进行调整，各实例通信心跳间隔需要比基础投票间隔小（若配置中各实例通信心跳间隔比基础投票间隔大，内部会将实例通信心跳间隔调整为与基础投票间隔相同，再进行其他的调整处理），增加监视器 ID 即 `MON_ID`（用于区别所有监视器中的不同实例，每个监视器实例的 `MON_ID` 与 `MON_INST_ID` 相同，取值（1~2147483647）），增加监视器系统唯一标识 `MON_MID`（取值（1~2147483647）），多实例监视器系统中的所有监视器需配置相同的 `MON_MID`。

与配置单实例监视器相比，配置多实例监视器时组的配置信息不做更改，监视器实例的配置信息需要新增配置参数，通过组和实例的配置信息内容的不同来区分组和实例，如果用户将组和实例的配置信息混淆，则以配置信息中更加靠前的参数为准来决定当前配置属于组还是实例，不属于组或实例的配置项将会被忽略。

监视器实例的配置信息包括实例 IP 地址、监听端口号以及实例在系统中的唯一编号，实例的 IP 地址和监听端口号不应出现冲突，否则该监视器实例可能无法正常使用。所有监视器实例的 `dmmonitor.ini` 配置文件中除 `MON_ID` 参数项外应完全一致，每个监视器的 `MON_ID` 应互不相同，用于区分同一个监视器系统中的不同监视器实例，若配置相同的 `MON_ID` 将导致通信无法正常建立。监视器实例监听的 IP 及端口号的正确性由用户保证。

配置多实例监视器时 `dmmonitor.ini` 配置文件中的内容如下：

<code>MON_LOG_PATH</code>	<code>= /dm/data/log</code>	
<code>MON_LOG_INTERVAL</code>	<code>= 60</code>	
<code>MON_LOG_FILE_SIZE</code>	<code>= 32</code>	
<code>MON_LOG_SPACE_LIMIT</code>	<code>= 0</code>	
<code>MON_DW_CONFIRM</code>	<code>= 1</code>	
<code>MON_INST_NUM</code>	<code>= 3</code>	#实例总个数
<code>MON_HB_INTERVAL</code>	<code>= 60</code>	#通信心跳校验间隔
<code>MON_BRO_INTERVAL</code>	<code>= 100</code>	#raft 协议中实例通信心跳间隔
<code>MON_VOTE_INTERVAL</code>	<code>= 100</code>	#raft 协议中基础投票间隔
<code>MON_ID</code>	<code>= 1</code>	#当前监视器在监视器系统中的 ID

```

MON_MID                = 45614      #当前监视器系统的唯一标识

[GRP1]

MON_INST_OGUID         = 45330

MON_DW_IP              = 192.168.0.141:5253

MON_DW_IP              = 192.168.0.142:5254

[GRP2]

MON_INST_OGUID         = 45331

MON_DW_IP              = 192.168.0.142:5253

MON_DW_IP              = 192.168.0.141:5254

[GRP3]

...

[MON1]

MON_HOST               = 192.168.0.141      #系统监听 TCP 连接的 IP 地址

MON_PORT               = 8339              #系统监听 TCP 连接的端口号

MON_INST_ID            = 1                  #监视器实例在监视器系统中的 ID

[MON2]

...

[MON3]

```

7.4.9 启动守护进程

分别启动 A、B 机器上的守护进程，例如：

```
./dmwatcher /dm/data/EP01/DAMENG/dmwatcher.ini
```

守护进程启动后，进入 Startup 状态，此时实例都处于 Mount 状态。守护进程开始广播自身和其监控实例的状态信息，结合自身信息和远程守护进程的广播信息，守护进程将本地实例 Open，并切换为 Open 状态。

7.4.10 启动监视器

启动监视器：

```
./dmmonitor /dm/data/dmmonitor.ini
```

监视器提供一系列命令，支持当前守护系统状态查看以及故障处理，可输入 help 命令，查看各种命令使用说明，结合实际情况选择使用。

至此 MPP 实时主备搭建完毕，在搭建步骤和各项配置都正确的情况下，在监视器上执行 show 命令，可以监控到所有实例都处于 Open 状态，所有守护进程也都处于 Open 状态，即为正常运行状态。

7.5 配置 DMDSC 主备环境

DMDSC 主备环境搭建和单节点主备环境搭建步骤类似，区别主要在于首先要准备 DMDSC 集群环境，DM8 支持单节点和单节点、DMDSC 集群和单节点之间搭建主备环境。

7.5.1 配置说明

DMDSC 集群可以作为主库，也可以作为实时备库、即时备库或者异步备库，一般建议将 DMDSC 集群配置为主库，将单节点配置为备库。如果要将 DMDSC 集群作为备库配置在数据守护系统中，则要将其作为一个整体配置在源库的 dmarch.ini 中，也就是 DMDSC 集群所有节点要配置在同一个归档配置项中，每个节点实例名以 “/” 分隔开来。

假如 DMDSC 集群有两个节点 GRP1_RT_DSC01 和 GRP1_RT_DSC02，DMDSC 集群要作为备库进行配置，其源库为 A，则要在 A 的 dmarch.ini 文件中增加 DMDSC 集群的归档配置，这里以实时备库为例说明如下：

```
[ARCHIVE_REALTIME1]
ARCH_TYPE          = REALTIME
ARCH_DEST           = GRP1_RT_DSC01/GRP1_RT_DSC02
```

如果 DMDSC 集群要作为即时备库或者异步备库来配置，ARCH_DEST 的配置方式和示例中是相同的，ARCH_TYPE 则要分别替换为 TIMELY 或者 ASYNC，中括号内的配置项名称中包含的归档类型也建议修改和 ARCH_TYPE 一致。

而如果异步备库的源库为 DMDSC 集群，则要求 DMDSC 集群中的所有节点都将 INI 参数 TIMER_INI 置为 1 并配置 dmtimer.ini。

7.5.2 环境说明

下面以 DMDSC 集群和单节点之间搭建实时主备环境为例，对搭建步骤进行说明。

准备三台机器 DW_P、DW_S、DW_M，DW_P 和 DW_S 用来部署主备库，DW_M 用来部署确认监视器。其中 DW_P 和 DW_S 配置两块网卡，一块接入内部网络交换模块，一块接入到外部交换机，DW_M 接入内部网络。

三台机器事先都安装了 DM，安装路径为 '/dm'，执行程序保存在 '/dm/bin' 目录中，数据存放路径为 '/dm/data'。

各主备库的实例名建议采用“组名_守护环境_序号”的方式命名，方便按组区分不同实例，注意总长度不能超过 16。本示例中组名为“GRP1”，配置为实时主备，主库 DMDSC 集群的两个节点实例名分别命名为“GRP1_RT_DSC01”、“GRP1_RT_DSC02”，备库命名为“GRP1_RT_01”。

表 7.10 配置环境说明

机器名	IP 地址	初始状态	操作系统	备注
DW_P	192.168.1.131	主库	Linux rh6-141.test	192.168.1.131
	192.168.0.141	GRP1_RT_DSC01 GRP1_RT_DSC02	2.6.32-220.el6.x86_64 #1 SMP Wed Nov 9 08:03:13 EST 2011 x86_64 x86_64 x86_64 GNU/Linux	外部服务 IP; 192.168.0.141 内部通信 IP
DW_S	192.168.1.132	备库	Linux	192.168.1.132
	192.168.0.142	GRP1_RT_01	rh6-142.localdomain 2.6.32-220.el6.x86_64 #1 SMP Wed Nov 9 08:03:13 EST 2011 x86_64 x86_64 x86_64 GNU/Linux	外部服务 IP; 192.168.0.142 内部通信 IP
DW_M	192.168.0.73	确认监视器	Linux rh6-73.localdomain	

			2.6.32-220.el6.x86_64	
			#1 SMP Wed Nov 9	
			08:03:13 EST 2011	
			x86_64 x86_64 x86_64	
			GNU/Linux	

表 7.11 端口规划

实例名	PORT_NUM	MAL_INST_DW_PORT	MAL_HOST	MAL_PORT	MAL_DW_PORT
GRP1_RT_DSC01	8344	4567	192.168.0.141	8338	3567
GRP1_RT_DSC02	8346	4568	192.168.0.142	8339	3568
GRP1_RT_01	9344	4569	192.168.0.73	8738	3569

7.5.3 配置 DMDSC 主库环境

使用 DMDSC 集群手册《DM8 共享存储集群》或者相应脚本搭建好两节点 DMDSC 集群环境，DMDSC 实例名分别为 GRP1_RT_DSC01、GRP1_RT_DSC02。

注意 dmcss 对 dmserver 的自动拉起功能先不要打开，避免影响到配置过程。

搭建完成后，正常退出 DMDSC 集群的两个 dmserver 节点实例，不需要退出 dmcss 和 dmasmsvr。

7.5.3.1 配置 dmarch.ini

分别编辑两个 DMDSC 节点的 dmarch.ini 文件，增加本地归档和远程归档，并且设置本地归档共享参数 ARCH_LOCAL_SHARE 为 1。dmarch.ini 文件放在各自 dm.ini 中指定的 CONFIG_PATH 目录下。

注意，若 DMDSC 主库环境搭建时未打开本地归档共享参数 ARCH_LOCAL_SHARE 为 1，而是在后续搭建主备过程中打开该参数，并且本地归档路径发生了变化，需要保证 DMDSC 主库本地归档的完整性，例如将本地归档路径发生变化前的归档文件拷至变化后的路径下，否则 DMDSC 主备环境会由于主库归档日志不完整，将备库归档状态置为 INVALID，归档同步失败（dmmonitor 执行 show arch send info EP，系统报错）。

(1) 配置 GRP1_RT_DSC01 的 dmarch.ini 文件

```

ARCH_LOCAL_SHARE      = 1
[ARCHIVE_LOCAL1]
    ARCH_TYPE          = LOCAL
    ARCH_DEST          = +DMDATA/DSC/DSC01/arch
    ARCH_FILE_SIZE     = 128
    ARCH_SPACE_LIMIT   = 0
[ARCHIVE_REMOTE]
    ARCH_TYPE          = REMOTE
    ARCH_DEST          = GRP1_RT_DSC02
    ARCH_FILE_SIZE     = 128
    ARCH_SPACE_LIMIT   = 0
    ARCH_INCOMING_PATH = +DMDATA/DSC/DSC02/arch

```

(2) 配置 GRP1_RT_DSC02 的 dmarch.ini 文件

```

ARCH_LOCAL_SHARE      = 1
[ARCHIVE_LOCAL1]
    ARCH_TYPE          = LOCAL
    ARCH_DEST          = +DMDATA/DSC/DSC02/arch
    ARCH_FILE_SIZE     = 128
    ARCH_SPACE_LIMIT   = 0
[ARCHIVE_REMOTE]
    ARCH_TYPE          = REMOTE
    ARCH_DEST          = GRP1_RT_DSC01
    ARCH_FILE_SIZE     = 128
    ARCH_SPACE_LIMIT   = 0
    ARCH_INCOMING_PATH = +DMDATA/DSC/DSC01/arch

```

7.5.3.2 配置 dm.ini

分别编辑两个 DMDSC 节点的 dm.ini 文件，打开归档参数：

```
ARCH_INI      = 1
```

7.5.3.3 备份 DMDSC 库

重启 DMDSC 集群的两个 dmserver 实例，然后再正常退出。在 DMDSC 集群生成有归档日志的情况下进行脱机备份，以便后续校验日志连续性时使用。

```

--启动 dmrman
./dmrman use_ap=2 dcr_ini=/dm/data/DSC/conf/dmdcr1.ini

--脱机备份 DMDSC 集群

```

```

RMAN>BACKUP      DATABASE      '/dm/data/DSC/DSC01/dm.ini'      FULL      BACKUPSET
'/dm/data/DSC/DSC01/bak/db_full_bak_for_DSC';

```

7.5.4 配置单节点备库

按照以下步骤准备单节点备库环境：

```

--初始化备库
./dminit path=/dm/data/EP01/

--启动 dmrman
./dmrman use_ap=2

--使用 DMDSC 库的备份集还原恢复到单节点备库
RMAN>RESTORE      DATABASE      '/dm/data/EP01/DAMENG/dm.ini'      FROM      BACKUPSET
'/dm/data/DSC/DSC01/bak/db_full_bak_for_DSC';

RMAN>RECOVER      DATABASE      '/dm/data/EP01/DAMENG/dm.ini'      FROM      BACKUPSET
'/dm/data/DSC/DSC01/bak/db_full_bak_for_DSC';

RMAN>RECOVER DATABASE '/dm/data/EP01/DAMENG/dm.ini' UPDATE DB_MAGIC;

```

7.5.5 配置 dm.ini

依次配置 DMDSC 主库所有节点和单节点备库的 dm.ini 文件，修改数据守护相关的参数配置。

(1) 配置 GRP1_RT_DSC01 的 dm.ini 文件

#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16

```

INSTANCE_NAME      = GRP1_RT_DSC01

PORT_NUM           = 8344    #数据库实例监听端口

DW_INACTIVE_INTERVAL = 60    #接收守护进程消息超时时间

ALTER_MODE_STATUS  = 0      #不允许手工方式修改实例模式/状态/OGUID

ENABLE_OFFLINE_TS   = 2      #不允许备库 OFFLINE 表空间

MAL_INI            = 1      #打开 MAL 系统

ARCH_INI           = 1      #打开归档配置

RLOG_SEND_APPLY_MON = 64     #统计最近 64 次的日志发送信息

```

(2) 配置 GRP1_RT_DSC02 的 dm.ini 文件

```

INSTANCE_NAME      = GRP1_RT_DSC02
PORT_NUM           = 8346
DW_INACTIVE_INTERVAL = 60
ALTER_MODE_STATUS  = 0
ENABLE_OFFLINE_TS   = 2
MAL_INI            = 1
ARCH_INI           = 1
RLOG_SEND_APPLY_MON = 64

```

(3) 配置 GRP1_RT_01 的 dm.ini 文件

```

INSTANCE_NAME      = GRP1_RT_01
PORT_NUM           = 9344
DW_INACTIVE_INTERVAL = 60
ALTER_MODE_STATUS  = 0
ENABLE_OFFLINE_TS   = 2
MAL_INI            = 1
ARCH_INI           = 1
RLOG_SEND_APPLY_MON = 64

```

7.5.6 配置 dmmal.ini

在 DMDSC 集群的 dmmal.ini 文件基础上，增加备库 GRP1_RT_01 的配置项。

所有节点实例的 dmmal.ini 文件内容是一致的，配置完成后，拷贝到每个节点实例 dm.ini 中指定的 CONFIG_PATH 目录下。

```

MAL_CHECK_INTERVAL    = 30          #MAL 链路检测时间间隔
MAL_CONN_FAIL_INTERVAL = 10         #判定 MAL 链路断开的的时间

[MAL_INST0]

MAL_INST_NAME         = GRP1_RT_DSC01 #实例名，和 dm.ini 中的 INSTANCE_NAME 一致
MAL_HOST              = 192.168.0.141 #MAL 系统监听 TCP 连接的 IP 地址
MAL_PORT              = 8338          #MAL 系统监听 TCP 连接的端口

```

```
MAL_INST_HOST      = 192.168.1.131  #实例的对外服务 IP 地址

MAL_INST_PORT      = 8344  #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致

MAL_DW_PORT        = 3567  #实例本地的守护进程监听 TCP 连接的端口

MAL_INST_DW_PORT    = 4567  #实例监听守护进程 TCP 连接的端口


[MAL_INST1]

MAL_INST_NAME      = GRP1_RT_DSC02

MAL_HOST           = 192.168.0.141

MAL_PORT           = 8339

MAL_INST_HOST      = 192.168.1.131

MAL_INST_PORT      = 8346

MAL_DW_PORT        = 3568

MAL_INST_DW_PORT    = 4568


[MAL_INST2]

MAL_INST_NAME      = GRP1_RT_01

MAL_HOST           = 192.168.0.142

MAL_PORT           = 8738

MAL_INST_HOST      = 192.168.1.132

MAL_INST_PORT      = 9344

MAL_DW_PORT        = 3569

MAL_INST_DW_PORT    = 4569
```

7.5.7 配置 dmarch.ini

再次编辑各个节点的 dmarch.ini 文件，增加实时归档配置。

(1) 修改 GRP1_RT_DSC01 的 dmarch.ini 文件

```
ARCH_LOCAL_SHARE    = 1

[ARCHIVE_LOCAL1]

ARCH_TYPE           = LOCAL

ARCH_DEST            = +DMDATA/DSC/DSC01/arch
```

```
ARCH_FILE_SIZE      = 128

ARCH_SPACE_LIMIT    = 0

[ARCHIVE_REMOTE]

ARCH_TYPE            = REMOTE

ARCH_DEST            = GRP1_RT_DSC02

ARCH_FILE_SIZE      = 128

ARCH_SPACE_LIMIT    = 0

ARCH_INCOMING_PATH  = +DMDATA/DSC/DSC02/arch

[ARCHIVE_REALTIME]

ARCH_TYPE            = REALTIME

ARCH_DEST            = GRP1_RT_01
```

(2) 修改 GRP1_RT_DSC02 的 dmarch.ini 文件

```
ARCH_LOCAL_SHARE    = 1

[ARCHIVE_LOCAL1]

ARCH_TYPE            = LOCAL

ARCH_DEST            = +DMDATA/DSC/DSC02/arch

ARCH_FILE_SIZE      = 128

ARCH_SPACE_LIMIT    = 0

[ARCHIVE_REMOTE]

ARCH_TYPE            = REMOTE

ARCH_DEST            = GRP1_RT_DSC01

ARCH_FILE_SIZE      = 128

ARCH_SPACE_LIMIT    = 0

ARCH_INCOMING_PATH  = +DMDATA/DSC/DSC01/arch

[ARCHIVE_REALTIME]

ARCH_TYPE            = REALTIME
```

```
ARCH_DEST          = GRP1_RT_01
```

(3) 修改 GRP1_RT_01 的 dmarch.ini 文件

```
[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL

ARCH_DEST          = /dm/data/EP01/DAMENG/arch

ARCH_FILE_SIZE     = 128

ARCH_SPACE_LIMIT   = 0


[ARCHIVE_REALTIME1]

ARCH_TYPE          = REALTIME

ARCH_DEST          = GRP1_RT_DSC01/GRP1_RT_DSC02
```

7.5.8 配置 dmwatcher.ini

依次配置每个节点实例的 dmwatcher.ini 文件，放到各自 dm.ini 中指定的 CONFIG_PATH 目录下。

另外要注意，DMDSC 集群各节点实例的自动拉起是由各自本地的 dmcscs 执行的，不是由守护进程执行，如果要打开 DMDSC 集群的自动拉起，需要再去配置 dmdcr.ini 中的自动拉起参数，为避免 dmcscs 在所有配置步骤完成之前提前将 dmserver 自动拉起，这里先不修改 dmdcr.ini 配置，放到后面步骤中修改。

(1) 配置 GRP1_RT_DSC01 的 dmwatcher.ini 文件

```
[GRP1]

DW_TYPE            = GLOBAL    #全局守护类型

DW_MODE            = MANUAL    #手动切换模式

DW_ERROR_TIME      = 60        #远程守护进程故障认定时间

INST_RECOVER_TIME  = 60        #主库守护进程启动恢复的间隔时间

INST_ERROR_TIME     = 35       #本地实例故障认定时间

INST_INI            = /dm/data/DSC/DSC01/dm.ini #dm.ini 配置文件路径

DCR_INI            = /dm/data/DSC/conf/dmdcr1.ini #dmdcr.ini 配置文件路径

INST_OGUID         = 1000     #守护系统唯一 OGUID 值
```

```
INST_STARTUP_CMD      = /dm/bin/dmserver #命令行方式启动

INST_AUTO_RESTART     = 0      #关闭实例的自动启动功能

RLOG_SEND_THRESHOLD   = 0      #指定主库发送日志到备库的时间阈值，默认关闭

RLOG_APPLY_THRESHOLD  = 0      #指定备库重演日志的时间阈值，默认关闭
```

(2) 配置 GRP1_RT_DSC02 的 dmwatcher.ini 文件

```
[GRP1]

DW_TYPE               = GLOBAL

DW_MODE               = MANUAL

DW_ERROR_TIME         = 60

INST_RECOVER_TIME     = 60

INST_ERROR_TIME       = 35

INST_INI              = /dm/data/DSC/DSC02/dm.ini

DCR_INI               = /dm/data/DSC/conf/dmdcr2.ini

INST_OGUID            = 1000

INST_STARTUP_CMD      = /dm/bin/dmserver

INST_AUTO_RESTART     = 0

RLOG_SEND_THRESHOLD   = 0

RLOG_APPLY_THRESHOLD  = 0
```

(3) 配置 GRP1_RT_01 的 dmwatcher.ini 文件

```
[GRP1]

DW_TYPE               = GLOBAL

DW_MODE               = MANUAL

DW_ERROR_TIME         = 60

INST_RECOVER_TIME     = 60

INST_ERROR_TIME       = 35

INST_INI              = /dm/data/EP01/dm.ini

INST_OGUID            = 1000

INST_STARTUP_CMD      = /dm/bin/dmserver
```

```

INST_AUTO_RESTART    = 0 #对单节点，如果需要打开自动拉起功能，将此配置修改为 1 即可

RLOG_SEND_THRESHOLD  = 0

RLOG_APPLY_THRESHOLD = 0

```



实际配置时，相关的端口配置和 **OGUID** 值建议不要和手册示例使用完全相同
注意：的值，避免多个用户在同一个环境下搭建不同的数据守护系统，出现消息混乱
 或者端口冲突等问题。

7.5.9 配置 dmmonitor.ini

编辑 dmmonitor.ini 文件，放在 /dm/data 目录下：

```

MON_LOG_PATH          = /dm/data/log

MON_LOG_INTERVAL      = 60

MON_LOG_FILE_SIZE     = 64

MON_LOG_SPACE_LIMIT   = 0

MON_DW_CONFIRM        = 0


[GRP1]

MON_INST_OGUID        = 1000

MON_DW_IP              = 192.168.0.141:3567/192.168.0.141:3568

MON_DW_IP              = 192.168.0.142:3569

```

7.5.10 配置 dmdcr.ini

DMDSC 集群中 dmserver 的自动拉起是由 dmcsc 执行的，如果不需要打开 dmcsc 的自动拉起功能，则可以跳过此章节。

否则需要修改 dmdcr.ini 中的自动拉起配置参数，此参数修改完成后，需要重启 dmcsc 才可以生效，为了避免重启 dmcsc 引发 dmasmsvr 被强制关闭，这里先将 dmcsc 和 dmasmsvr 都正常退出。

另外要注意，本示例中是用命令行方式启动，启动参数中指定以 Mount 方式拉起 dmserver，如果是用服务方式启动，服务脚本中也一定要指定以 Mount 方式拉起

dmserver。

1. 打开 GRP1_RT_DSC01 的自动拉起参数

修改对应 dmcss 的 dmdcr.ini 文件，打开自动拉起参数。

```
#打开 DB 重启参数，命令行方式启动

DMDCR_DB_RESTART_INTERVAL = 60

DMDCR_DB_STARTUP_CMD = /dm/bin/dmserver path=/dm/data/DSC/DSC01/dm.ini

dcr_ini=/dm/data/DSC/conf/dmdcr1.ini mount
```

2. 打开 GRP1_RT_DSC02 的自动拉起参数

修改对应 dmcss 的 dmdcr.ini 文件，打开自动拉起参数。

```
#打开 DB 重启参数，命令行方式启动

DMDCR_DB_RESTART_INTERVAL = 60

DMDCR_DB_STARTUP_CMD = /dm/bin/dmserver path=/dm/data/DSC/DSC02/dm.ini

dcr_ini=/dm/data/DSC/conf/dmdcr2.ini mount
```

3. 重启 dmcss 和 dmasmsvr

```
./dmcss dcr_ini=/dm/data/DSC/conf/dmdcr1.ini

./dmcss dcr_ini=/dm/data/DSC/conf/dmdcr2.ini

./dmasmsvr dcr_ini=/dm/data/DSC/conf/dmdcr1.ini

./dmasmsvr dcr_ini=/dm/data/DSC/conf/dmdcr2.ini
```

7.5.11 启动主备库

注意实例都要使用 Mount 方式启动。

1. 启动 DMDSC 集群的两个 dmserver 实例

如果 dmcss 打开有自动拉起功能，也可以等待 dmcss 将本地的 dmserver 实例自动拉起（要保证以 Mount 方式拉起）。

手动启动命令如下：

```
./dmserver /dm/data/DSC/DSC01/dm.ini DCR_INI=/dm/data/DSC/conf/dmdcr1.ini
mount

./dmserver /dm/data/DSC/DSC02/dm.ini DCR_INI=/dm/data/DSC/conf/dmdcr2.ini
```

```
mount
```

2. 启动单节点备库

```
./dmserver /dm/data/EP01/dm.ini mount
```

7.5.12 设置 OGUID

启动命令行工具 DIsql，连接 DMDSC 集群中的任意一个节点，设置 DMDSC 主库的 OGUID 值。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);
SQL>SP_SET_OGUID(1000);
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

DIsql 连接单节点备库，设置备库 OGUID。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);
SQL>SP_SET_OGUID(1000);
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

7.5.13 修改主备库模式

启动命令行工具 DIsql，连接 DMDSC 集群中的任意一个节点，修改 DMDSC 库为 Primary 模式。

```
SQL>ALTER DATABASE PRIMARY;
```

DIsql 连接单节点备库，修改备库为 Standby 模式。

```
SQL>ALTER DATABASE STANDBY;
```

7.5.14 启动守护进程

启动 DMDSC 主库的所有守护进程：

```
./dmwatcher /dm/data/DSC/DSC01/dmwatcher.ini
./dmwatcher /dm/data/DSC/DSC02/dmwatcher.ini
```

启动备库的守护进程：

```
./dmwatcher /dm/data/EP01/dmwatcher.ini
```

7.5.15 启动监视器

启动数据守护监视器：

```
./dmmonitor path=/dm/data/dmmonitor.ini
```

监视器提供一系列命令，支持当前守护系统状态查看以及故障处理，可输入 help 命令，查看各种命令使用说明，结合实际情况选择使用。

另外，DMDSC 集群也有自己的监视器工具，需要时也可以启动：

```
./dmcsm ini_path=/dm/data/DSC/conf/dmcsm.ini
```

至此 DMDSC 集群的实时数据守护系统搭建完毕，在搭建步骤和各项配置都正确的情况下，在监视器上执行 show 命令，可以监控到所有实例都处于 Open 状态，所有守护进程也都处于 Open 状态，即为正常运行状态。

7.6 配置异步备库

在实际应用中，如果数据库规模很大，并且对数据的实时性要求不是很严格，则可以配置多个异步备库用于分担统计报表等任务。

异步备库支持多源配置，目的是在实时或即时主备环境中，当主备库切换或者主库故障时，备库接管以后可以继续向同一个异步备库同步数据，因此如果主库配置了异步备库，在所有备库上也需要配置相同的异步备库，备库只有在切换为主库时才会向其同步数据。

这里以实时主备环境为例，仍然采用 7.2 配置实时主备 小节的配置环境，对异步备库的搭建步骤做举例说明，读写分离集群和 MPP 主备的异步备库搭建步骤是相同的，只需要根据实际情况调整配置项的具体值即可。

这里以一个异步备库为例，配置在主库 GRP1_RT_01 上，异步备库实例名为 GRP1_LOCAL_01。如果需要配置多个异步备库，对应的源实例配置可以参考主库 GRP1_RT_01 的配置步骤，异步备库自身的配置可以参考 GRP1_LOCAL_01 的配置步骤。

7.6.1 环境说明

这里采用 7.2 配置实时主备 的实时主备环境，增加一台机器用来部署异步备库，其他环境不变。

新增的机器事先安装了 DM，安装路径为 '/dm'，执行程序保存在 '/dm/bin' 目录中，

数据存放路径为 '/dm/data'。

新增的异步备库实例名为“GRP1_LOCAL_01”，按照“组名_守护环境_序号”的建议方式命名，注意总长度不能超过 16。

表 7.12 配置环境说明

机器名	IP 地址	初始状态	操作系统	备注
DW_S2	192.168.1.133	异步备库	Linux	192.168.1.133
	192.168.0.143	GRP1_LOCAL_01	rh6-143.localdomain	外部服务 IP;
			2.6.32-220.el6.x86_64	192.168.0.143
			#1 SMP Wed Nov 9	内部通信 IP
			08:03:13 EST 2011	
			x86_64 x86_64 x86_64	
			GNU/Linux	

7.6.2 数据准备

按照 7.1 数据准备中的方法准备异步备库的数据。

如果实时主备守护环境已经处于运行状态，需要先正常退出主备库实例和守护进程。

7.6.3 配置主库 GRP1_RT_01

7.6.3.1 配置 dm.ini

在 7.2.3.1 配置 dm.ini 的基础上，打开定时器配置，其他配置不变。

#配置有异步归档时，打开定时器，定时同步归档到异备库

TIMER_INI = 1

7.6.3.2 配置 dmmal.ini

在 7.2.3.2 配置 dmmal.ini 的基础上，增加异步备库 GRP1_LOCAL_01 的 MAL 配置项。

[MAL_INST3]

```

MAL_INST_NAME      = GRP1_LOCAL_01 #实例名，和 dm.ini 中的 INSTANCE_NAME 一致

MAL_HOST           = 192.168.0.143 #MAL 系统监听 TCP 连接的 IP 地址

MAL_PORT           = 61143          #MAL 系统监听 TCP 连接的端口

MAL_INST_HOST      = 192.168.1.133 #实例的对外服务 IP 地址

MAL_INST_PORT      = 32143 #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致

MAL_DW_PORT        = 52143 #实例对应的守护进程监听 TCP 连接的端口

MAL_INST_DW_PORT   = 33143 #实例监听守护进程 TCP 连接的端口

```

7.6.3.3 配置 dmarch.ini

在 7.2.3.3 配置 dmarch.ini 的基础上，增加异步归档的配置项。

```

[ARCHIVE_ASYNC]

ARCH_TYPE          = ASYNC          #异步归档类型

ARCH_DEST          = GRP1_LOCAL_01 #异步归档目标实例名

ARCH_TIMER_NAME    = RT_TIMER       #定时器名称，和 dmtimer.ini 中的名称一致

```

7.6.3.4 配置 dmtimer.ini

配置 dmtimer.ini，用于定时触发实例发送归档日志到异步备库。

下面示例中定时器配置为每天 00:00:00 触发主库发送归档日志到异步备库，可以根据实际情况再做调整。

```

[RT_TIMER] #和 dmarch.ini 中的 ARCH_TIMER_NAME 一致

TYPE              = 2

FREQ_MONTH_WEEK_INTERVAL = 1

FREQ_SUB_INTERVAL  = 0

FREQ_MINUTE_INTERVAL = 0

START_TIME        = 00:00:00

END_TIME          = 00:00:00

DURING_START_DATE  = 2016-02-11 17:36:09

DURING_END_DATE    = 9999-12-31 23:59:59

NO_END_DATE_FLAG   = 1

```

```

DESCRIBE          = RT_TIMER
IS_VALID          = 1

```

7.6.3.5 其他配置说明

如果配置异步备库时，实时主备环境已经配置完成，这里对主库不需要再做其他配置，由于在配置异步备库之前实时主备环境已经被正常关闭，此处可以使用 Mount 方式将主库重启启动，并启动主库的守护进程。

如果同时也在搭建主库和实时备库，则主库其他的配置步骤请再参考 [7.2.3 配置主库 GRP1_RT_01](#) 的配置说明。

7.6.4 配置备库 GRP1_RT_02

主库上配置异步备库，备库上也需要增加相同的异步备库配置，保证备库在切换为主库后可以继续向同一个异步备库同步数据。

7.6.4.1 配置 dm.ini

在 [7.2.4.1 配置 dm.ini](#) 的基础上，打开定时器配置，其他配置不变。

```

#配置有异步归档时，打开定时器，定时同步归档到异步库
TIMER_INI          = 1

```

7.6.4.2 配置 dmmal.ini

dmmal.ini 中需要增加到异步备库 GRP1_LOCAL_01 的 MAL 配置项，可以直接拷贝 [7.6.3.2 配置 dmmal.ini](#) 修改后的 dmmal.ini 文件到本地 /dm/data/DAMENG/ 目录中。

7.6.4.3 配置 dmarch.ini

在 [7.2.4.3 配置 dmarch.ini](#) 的基础上，增加异步归档的配置项。

```

[ARCHIVE_ASYNC]

```

ARCH_TYPE	= ASYNC	#异步归档类型
ARCH_DEST	= GRP1_LOCAL_01	#异步归档目标实例名
ARCH_TIMER_NAME	= RT_TIMER	#定时器名称, 和 dmtimer.ini 中的名称一致

7.6.4.4 配置 dmtimer.ini

配置 dmtimer.ini, 用于备库切换为主库后, 定时触发实例发送归档日志到异步备库。

下面示例中定时器配置为每天 00:00:00 触发主库发送归档日志到异步备库, 可以根据实际情况再做调整。

```
[RT_TIMER] #和 dmarch.ini 中的 ARCH_TIMER_NAME 一致

TYPE                      = 2

FREQ_MONTH_WEEK_INTERVAL    = 1

FREQ_SUB_INTERVAL          = 0

FREQ_MINUTE_INTERVAL       = 0

START_TIME                 = 00:00:00

END_TIME                   = 00:00:00

DURING_START_DATE          = 2016-02-11 17:36:09

DURING_END_DATE            = 9999-12-31 23:59:59

NO_END_DATE_FLAG           = 1

DESCRIBE                   = RT_TIMER

IS_VALID                   = 1
```

7.6.4.5 其他配置说明

如果配置异步备库时, 实时主备环境已经配置完成, 这里对实时备库不需要再做其他配置, 由于在配置异步备库之前实时主备环境已被正常关闭, 此处可以使用 Mount 方式将实时备库重新启动, 并启动实时备库的守护进程。

如果同时也在搭建主库和实时备库, 则实时备库其他的配置步骤请再参考 [7.2.4 配置备库 GRP1_RT_02](#) 的配置说明。

7.6.5 配置异步备库 GRP1_LOCAL_01

7.6.5.1 配置 dm.ini

在 DW_S2 机器上配置备库的实例名为 GRP1_LOCAL_01，dm.ini 参数修改如下：

#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16

```

INSTANCE_NAME          = GRP1_LOCAL_01

PORT_NUM               = 32143 #数据库实例监听端口

DW_INACTIVE_INTERVAL  = 60      #接收守护进程消息超时时间

ALTER_MODE_STATUS     = 0       #不允许手工方式修改实例模式/状态/OGUID

ENABLE_OFFLINE_TS     = 2       #不允许备库 OFFLINE 表空间

MAL_INI               = 1       #打开 MAL 系统

ARCH_INI              = 1       #打开归档配置

RLOG_SEND_APPLY_MON   = 64      #统计最近 64 次的日志重演信息

```

7.6.5.2 配置 dmmal.ini

同一个守护进程组中，所有主备库必须使用相同的 dmmal.ini 配置文件，这里可以直接拷贝 7.6.3.2 配置 dmmal.ini 的 dmmal.ini 文件到本地的 /dm/data/DAMENG/ 目录中。

7.6.5.3 配置 dmarch.ini

本地守护类型的备库只需要配置本地归档。

```

[ARCHIVE_LOCAL1]

ARCH_TYPE              = LOCAL #本地归档类型

ARCH_DEST              = /dm/data/DAMENG/arch #本地归档文件路径

ARCH_FILE_SIZE         = 128   #单位 Mb，本地单个归档文件最大值

ARCH_SPACE_LIMIT       = 0     #单位 Mb，0 表示无限制，范围 1024~4294967294M

```

7.6.5.4 配置 dmwatcher.ini

修改 dmwatcher.ini 配置守护进程，配置为本地守护类型，异步备库不具备故障自动切换等功能，DW_MODE 配置并不起作用，此处配置为 MANUAL 即可。

```
[GRP1]

DW_TYPE           = LOCAL      #本地守护类型

DW_MODE           = MANUAL     #故障手动切换模式

DW_ERROR_TIME     = 10        #远程守护进程故障认定时间

INST_ERROR_TIME   = 10        #本地实例故障认定时间

INST_OGUID        = 453331    #守护系统唯一 OGUID 值

INST_INI          = /dm/data/DAMENG/dm.ini    #dm.ini 配置文件路径

INST_AUTO_RESTART = 1         #打开实例的自动启动功能

INST_STARTUP_CMD  = /dm/bin/dmserver  #命令行方式启动
```

7.6.5.5 启动异步备库

以 Mount 方式启动备库

```
./dmserver /dm/data/DAMENG/dm.ini mount
```

一定要以 Mount 方式启动数据库实例，否则系统启动时会重构回滚表空间，生成 Redo 日志；并且，启动后应用可能连接到数据库实例进行操作，破坏主备库的数据一致性。数据守护配置结束后，守护进程会自动 Open 数据库。

7.6.5.6 设置 OGUID

启动命令行工具 DISql，登录异步备库设置 OGUID 值。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>sp_set_oguid(453331);

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

系统通过 OGUID 值确定一个守护进程组，由用户保证 OGUID 值的唯一性，并确保数据守护系统中，数据库、守护进程和监视器配置相同的 OGUID 值。

7.6.5.7 修改数据库模式

启动命令行工具 DISql，登录异备库修改数据库为 Standby 模式。

```
SQL>SP_SET PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>ALTER DATABASE STANDBY;

SQL>SP_SET PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

7.6.6 配置监视器

在 7.2.5 配置监视器的配置基础上，需要增加到异步备库守护进程的“IP:PORT”信息，否则监视器无法收到异步备库消息，其他配置项不需要改动。

配置监视器时可以选择配置单实例监视器或配置多实例监视器，下面将分别介绍配置单实例监视器以及多实例监视器时 dmmonitor.ini 配置文件中的配置信息。

7.6.6.1 配置单实例监视器

修改 dmmonitor.ini 配置确认监视器，增加到 GRP1_LOCAL_01 守护进程的 MON_DW_IP 配置项，需要找到 dmmal.ini 中实例 GRP1_LOCAL_01 的配置项信息，MON_DW_IP 中的 IP 信息和 dmmal.ini 配置项中的 MAL_HOST 一致，PORT 信息和 dmmal.ini 配置项中的 MAL_DW_PORT 配置项保持一致。

```
MON_DW_CONFIRM          = 1      #确认监视器模式

MON_LOG_PATH            = /dm/data/log      #监视器日志文件存放路径

MON_LOG_INTERVAL        = 60      #每隔 60s 定时记录系统信息到日志文件

MON_LOG_FILE_SIZE       = 32      #每个日志文件最大 32M

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

[GRP1]

    MON_INST_OGUID       = 453331 #组 GRP1 的唯一 OGUID 值

#以下配置为监视器到组 GRP1 的守护进程的连接信息，以“IP:PORT”的形式配置

#IP 对应 dmmal.ini 中的 MAL_HOST，PORT 对应 dmmal.ini 中的 MAL_DW_PORT

    MON_DW_IP            = 192.168.0.141:52141

    MON_DW_IP            = 192.168.0.142:52142
```

```
#新增到异步备库 GRP1_LOCAL_01 守护进程的连接信息
```

```
MON_DW_IP = 192.168.0.143:52143
```

7.6.6.2 配置多实例监视器

与配置单实例监视器相比，配置多实例监视器时 `dmmonitor.ini` 中需要添加整个监视器系统中的实例总数 `MON_INST_NUM`（取值为 1、3 或者 5），各实例通信的心跳校验间隔 `MON_HB_INTERVAL`（取值（5~600），单位秒），各实例通信的心跳间隔 `MON_BRO_INTERVAL`（取值（5~600），单位毫秒），各实例通信的基础投票间隔 `MON_VOTE_INTERVAL`（取值（5~600），单位毫秒），各实例通信的心跳间隔以及基础投票间隔会在真正使用时进行调整，各实例通信心跳间隔需要比基础投票间隔小（若配置中各实例通信心跳间隔比基础投票间隔大，内部会将实例通信心跳间隔调整为与基础投票间隔相同，再进行其他的调整处理），增加监视器 ID 即 `MON_ID`（用于区别所有监视器中的不同实例，每个监视器实例的 `MON_ID` 与 `MON_INST_ID` 相同，取值（1~2147483647）），增加监视器系统唯一标识 `MON_MID`（取值（1~2147483647）），多实例监视器系统中的所有监视器需配置相同的 `MON_MID`。

与配置单实例监视器相比，配置多实例监视器时组的配置信息不做更改，监视器实例的配置信息需要新增配置参数，通过组和实例的配置信息内容的不同来区分组和实例，如果用户将组和实例的配置信息混淆，则以配置信息中更加靠前的参数为准来决定当前配置属于组还是实例，不属于组或实例的配置项将会被忽略。

监视器实例的配置信息包括实例 IP 地址、监听端口号以及实例在系统中的唯一编号，实例的 IP 地址和监听端口号不应出现冲突，否则该监视器实例可能无法正常使用。所有监视器实例的 `dmmonitor.ini` 配置文件中除 `MON_ID` 参数项外应完全一致，每个监视器的 `MON_ID` 应互不相同，用于区分同一个监视器系统中的不同监视器实例，若配置相同的 `MON_ID` 将导致通信无法正常建立。监视器实例监听的 IP 及端口号的正确性由用户保证。

配置多实例监视器时 `dmmonitor.ini` 配置文件中的内容如下：

```
MON_LOG_PATH = /dm/data/log
MON_LOG_INTERVAL = 60
MON_LOG_FILE_SIZE = 32
MON_LOG_SPACE_LIMIT = 0
```

```
MON_DW_CONFIRM          = 1

MON_INST_NUM            = 3          #实例总个数

MON_HB_INTERVAL         = 60        #通信心跳校验间隔

MON_BRO_INTERVAL        = 100       #raft 协议中实例通信心跳间隔

MON_VOTE_INTERVAL       = 100       #raft 协议中基础投票间隔

MON_ID                  = 1          #当前监视器在监视器系统中的 ID

MON_MID                 = 45614     #当前监视器系统的唯一标识


[GRP1]

MON_INST_OGUID          = 453331

MON_DW_IP               = 192.168.0.141:52141
MON_DW_IP               = 192.168.0.142:52142
#新增到异步备库 GRP1_LOCAL_01 守护进程的连接信息
MON_DW_IP               = 192.168.0.143:52143


[GRP2]

...

[GRP3]

...


[MON1]

MON_HOST                = 192.168.0.141    #系统监听 TCP 连接的 IP 地址
MON_PORT                = 8339            #系统监听 TCP 连接的端口号
MON_INST_ID             = 1                #监视器实例在监视器系统中的 ID

[MON2]

...

[MON3]
```

7.6.7 启动守护进程

启动异步备库的守护进程：

```
./dmwatcher /dm/data/DAMENG/dmwatcher.ini
```

守护进程启动后，进入 Startup 状态，此时实例处于 Mount 状态，异步备库的守护进程会将本地实例自动 Open，并切换守护进程自身为 Open 状态。

7.6.8 启动监视器

修改监视器配置后，需要重新启动监视器：

```
./dmmonitor /dm/data/dmmonitor.ini
```

监视器提供一系列命令，支持当前守护系统状态查看以及故障处理，可输入 help 命令，查看各种命令使用说明，结合实际情况选择使用。

至此实时主备环境下的异步备库搭建完成，可以根据实际情况考虑配置多个异步备库，每个异步备库源实例的配置方式都可以参考主库 GRP1_RT_01 的配置，在搭建步骤和各项配置都正确的情况下，在监视器上执行 show 命令，可以监控到异步备库实例和异步备库守护进程都处于 Open 状态。

7.7 注册服务

如果想让 dmserver、dmwatcher 和 dmmonitor 服务开机自启动，需要手动注册服务，注册步骤必须用 root 用户进行，注册完成后重启机器时，就会自动启动 dmserver、dmwatcher 和 dmmonitor 服务。其中，因为确认监视器一直处于工作状态，所以确认监视器需要注册 dmmonitor 服务，自动启动；非确认监视器是在查看信息时才使用，届时手动启动控制台 dmmonitor 服务即可，可以不用注册 dmmonitor 服务。详细的配置请参考《DM8_Linux 服务脚本使用手册》。

7.8 动态增加读写分离集群节点

当需要进行系统扩容，希望系统运行不中断，或者影响运行的时间尽可能短时，可通过动态增加集群节点的方式进行。下面举例对读写分离集群进行动态增加节点。

本例是在 7.3 配置读写分离集群的基础上，再增加一个备库，实例名为 GRP1_RWW_04。

表 7.11 配置环境说明

机器名	IP 地址	初始状态	操作系统	备注
DW_S3	192.168.1.134 192.168.0.144	备库 GRP1_RWW_04	Linux rh6 x86_64	192.168.1.134 外部服务 IP; 192.168.0.144 内部通信 IP

详细步骤如下：

7.8.1 数据准备

1. 对主库进行联机备份操作：

```
SQL> BACKUP DATABASE BACKUPSET 'BACKUP_FILE_01';
```

2. 初始化备机数据库

```
./dmunit path=/dm/data/
```

3. 还原恢复新增备库

拷贝生成的备份集目录 BACKUP_FILE_01 到 144 上 /dm/data/ 目录，使用 DMRMAN 工具脱机还原。

```
./dmrman CTLSTMT="RESTORE DATABASE '/dm/data/DAMENG/dm.ini' FROM BACKUPSET  
'/dm/data/BACKUP_FILE_01'"  
  
./dmrman CTLSTMT="RECOVER DATABASE '/dm/data/DAMENG/dm.ini' FROM BACKUPSET  
'/dm/data/BACKUP_FILE_01'"  
  
./dmrman CTLSTMT="RECOVER DATABASE '/dm/data/DAMENG/dm.ini' UPDATE DB_MAGIC"
```

7.8.2 配置新备库

7.8.2.1 配置 dm.ini

在 DW_S3 机器上配置备库的实例名为 GRP1_RWW_04，dm.ini 参数修改如下：

```
#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16  
  
INSTANCE_NAME          = GRP1_RWW_04  
  
PORT_NUM               = 32144 #数据库实例监听端口  
  
DW_INACTIVE_INTERVAL   = 60    #接收守护进程消息超时时间  
  
ALTER_MODE_STATUS      = 0     #不允许手工方式修改实例模式/状态/OGUID
```

```

ENABLE_OFFLINE_TS      = 2      #不允许备库 OFFLINE 表空间
MAL_INI                 = 1      #打开 MAL 系统
ARCH_INI                = 1      #打开归档配置
RLOG_SEND_APPLY_MON     = 64     #统计最近 64 次的日志重演信息

```

7.8.2.2 配置 dmmal.ini

拷贝一份原系统 dmmal.ini 文件，并加上自己一项，最终配置如下：

```

MAL_CHECK_INTERVAL      = 5      #MAL 链路检测时间间隔
MAL_CONN_FAIL_INTERVAL  = 5      #判定 MAL 链路断开的的时间

[MAL_INST1]

MAL_INST_NAME           = GRP1_RWW_01#实例名，和 dm.ini 中的 INSTANCE_NAME 一致
MAL_HOST                 = 192.168.0.141 #MAL 系统监听 TCP 连接的 IP 地址
MAL_PORT                 = 61141        #MAL 系统监听 TCP 连接的端口
MAL_INST_HOST            = 192.168.1.131 #实例的对外服务 IP 地址
MAL_INST_PORT            = 32141 #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致
MAL_DW_PORT              = 52141 #实例对应的守护进程监听 TCP 连接的端口
MAL_INST_DW_PORT         = 33141 #实例监听守护进程 TCP 连接的端口

[MAL_INST2]

MAL_INST_NAME           = GRP1_RWW_02
MAL_HOST                 = 192.168.0.142
MAL_PORT                 = 61142
MAL_INST_HOST            = 192.168.1.132
MAL_INST_PORT            = 32142
MAL_DW_PORT              = 52142
MAL_INST_DW_PORT         = 33142

[MAL_INST3]

```

```
MAL_INST_NAME      = GRP1_RWW_03

MAL_HOST            = 192.168.0.143

MAL_PORT            = 61143

MAL_INST_HOST       = 192.168.1.133

MAL_INST_PORT       = 32143

MAL_DW_PORT         = 52143

MAL_INST_DW_PORT    = 33143
```

```
[MAL_INST4]
```

```
MAL_INST_NAME      = GRP1_RWW_04

MAL_HOST            = 192.168.0.144

MAL_PORT            = 61144

MAL_INST_HOST       = 192.168.1.134

MAL_INST_PORT       = 32144

MAL_DW_PORT         = 52144

MAL_INST_DW_PORT    = 33144
```

7.8.2.3 配置 dmarch.ini

修改 dmarch.ini，配置本地归档和即时归档。

```
[ARCHIVE_TIMELY1]

ARCH_TYPE           = TIMELY           #即时归档类型

ARCH_DEST            = GRP1_RWW_01     #即时归档目标实例名

[ARCHIVE_TIMELY2]

ARCH_TYPE           = TIMELY           #即时归档类型

ARCH_DEST            = GRP1_RWW_02     #即时归档目标实例名

[ARCHIVE_TIMELY3]

ARCH_TYPE           = TIMELY           #即时归档类型

ARCH_DEST            = GRP1_RWW_03     #即时归档目标实例名

[ARCHIVE_LOCAL1]
```

ARCH_TYPE	= LOCAL	#本地归档类型
ARCH_DEST	= /dm/data/DAMENG/arch	#本地归档文件存放路径
ARCH_FILE_SIZE	= 128	#单位 Mb, 本地单个归档文件最大值
ARCH_SPACE_LIMIT	= 0	#单位 Mb, 0 表示无限制, 范围 1024~4294967294M

7.8.2.4 配置 dmwatcher.ini

修改 dmwatcher.ini 配置守护进程, 配置为全局守护类型, 使用自动切换模式。

```
[GRP1]

DW_TYPE           = GLOBAL      #全局守护类型
DW_MODE           = AUTO        #自动切换模式
DW_ERROR_TIME     = 10          #远程守护进程故障认定时间
INST_RECOVER_TIME = 60          #主库守护进程启动恢复的间隔时间
INST_ERROR_TIME   = 10          #本地实例故障认定时间
INST_OGUID        = 453332     #守护系统唯一 OGUID 值
INST_INI          = /dm/data/DAMENG/dm.ini  #dm.ini 配置文件路径
INST_AUTO_RESTART = 1          #打开实例的自动拉起功能
INST_STARTUP_CMD  = /dm/bin/dmserver #命令行方式启动
RLOG_SEND_THRESHOLD = 0        #指定主库发送日志到备库的时间阈值, 默认关闭
RLOG_APPLY_THRESHOLD = 0       #指定备库重演日志的时间阈值, 默认关闭
```

7.8.2.5 启动备库

以 Mount 方式启动备库

```
./dmserver /dm/data/DAMENG/dm.ini mount
```

7.8.2.6 设置 OGUID

启动命令行工具 DISql, 登录备库设置 OGUID 值。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>sp_set_oguid(453332);
```

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

7.8.2.7 修改数据库模式

启动命令行工具 DISql，登录备库修改数据库为 Standby 模式：

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);
```

```
SQL>ALTER DATABASE STANDBY;
```

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

7.8.3 动态添加 MAL 配置

本小节的步骤需要分别连接原系统中每个数据库单独执行，如果是 DMDSC 集群，则需要连接集群内的每一个节点执行：

动态增加 MAL 中 GRP1_RWW_04 的相关配置信息：

```
SF_MAL_CONFIG(1,0);
```

```
SF_MAL_INST_ADD('MAL_INST4','GRP1_RWW_04','192.168.0.144',61144,'192.168.1.134',32144,52144,0,33144);
```

```
SF_MAL_CONFIG_APPLY();
```

```
SF_MAL_CONFIG(0,0);
```

7.8.4 动态添加归档配置

动态添加归档配置步骤如下：

1. 关闭原系统中所有数据库的守护进程和监视器

动态添加归档要求数据库处于 MOUNT 状态，守护进程处于活动状态下不允许用户手动修改模式状态，因此需要先关闭守护进程，同时也是为了防止手动切换 MOUNT 状态后又被守护进程通知自动 OPEN，另外监视器配置也需要修改，因此在这里一并关闭。

关闭顺序为：

- 1) 关闭监视器
- 2) 关闭备库守护进程
- 3) 关闭主库守护进程

2. 手动修改原系统中所有数据库为 MOUNT 状态

必须先修改主库，再修改备库，否则可能会引发主备数据同步失败导致主库挂起。

```
--临时修改 ALTER_MODE_STATUS，允许用户手动修改模式状态，动态添加完后需要再改回原值
SQL> SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

--临时关闭守护进程活动状态检测，允许用户手动修改模式状态，动态添加完后需要再改回原值
SQL> SP_SET_PARA_VALUE(1, 'DW_INACTIVE_INTERVAL', 0);

--修改数据库为 MOUNT 状态
SQL> ALTER DATABASE MOUNT;
```

3. 动态添加归档节点

分别连接原系统中的所有数据库执行，动态添加 dmarch.ini 中归档节点，如果是 DMDCS 集群，则需要连接集群内的每一个节点执行。

```
SQL> ALTER DATABASE ADD ARCHIVELOG 'DEST=GRP1_RWW_04, TYPE=TIMELY';
```

4. 手动修改原系统中所有数据库为 OPEN 状态

必须先修改备库，再修改主库，否则可能会引发主备数据同步失败导致主库挂起。

```
--修改数据库为 OPEN 状态
SQL> ALTER DATABASE OPEN FORCE;

--将 ALTER_MODE_STATUS 改回原值
SQL> SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);

--将 DW_INACTIVE_INTERVAL 改回原值，本例中为 60
SQL> SP_SET_PARA_VALUE(1, 'DW_INACTIVE_INTERVAL', 60);
```

7.8.5 修改监视器 dmmonitor.ini

在 dmmonitor.ini 中添加新增的备库 GRP1_RWW_04:

```
MON_DW_IP          = 192.168.0.144:52144
```

7.8.6 启动所有守护进程以及监视器

分别启动主库和备库（包括 GRP1_RWW_04）的所有守护进程，最后启动监视器。

7.9 动态增加实时备库

当需要进行系统扩容，希望系统运行不中断，或者影响运行的时间尽可能短时，可通过动态增加节点的方式进行。下面举例对实时主备进行动态增加节点。

本例在 7.2 配置实时主备集群的基础上，再增加一个备库，实例名为 GRP1_RT_03。

表 7.14 配置环境说明

机器名	IP 地址	初始状态	操作系统	备注
DW_S2	192.168.1.133 192.168.0.143	备库 GRP1_RT_03	Linux rh6-141.test 2.6.32-220.el6.x86_64	192.168.1.133 外部服务 IP; 192.168.0.143 内部通信 IP

详细步骤如下：

7.9.1 数据准备

1. 对主库进行联机备份操作：

```
SQL> BACKUP DATABASE BACKUPSET 'BACKUP_FILE_01';
```

2. 初始化备机数据库

```
./dminit path=/dm/data/
```

3. 还原恢复新增备库

拷贝生成的备份集目录 BACKUP_FILE_01 到 143 上 /dm/data/ 目录，使用 DMRMAN 工具脱机还原。

```
./dmrman CTLSTMT="RESTORE DATABASE '/dm/data/DAMENG/dm.ini' FROM BACKUPSET  
'/dm/data/BACKUP_FILE_01'"  
  
./dmrman CTLSTMT="RECOVER DATABASE '/dm/data/DAMENG/dm.ini' FROM BACKUPSET  
'/dm/data/BACKUP_FILE_01'"  
  
./dmrman CTLSTMT="RECOVER DATABASE '/dm/data/DAMENG/dm.ini' UPDATE DB_MAGIC"
```

7.9.2 配置新备库

7.9.2.1 配置 dm.ini

在 DW_S3 机器上配置备库的实例名为 GRP1_RT_03，dm.ini 参数修改如下：

```
#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16

INSTANCE_NAME          = GRP1_RT_03

PORT_NUM               = 32143 #数据库实例监听端口

DW_INACTIVE_INTERVAL   = 60    #接收守护进程消息超时时间

ALTER_MODE_STATUS      = 0     #不允许手工方式修改实例模式/状态/OGUID

ENABLE_OFFLINE_TS      = 2     #不允许备库 OFFLINE 表空间

MAL_INI                = 1     #打开 MAL 系统

ARCH_INI               = 1     #打开归档配置

RLOG_SEND_APPLY_MON    = 64    #统计最近 64 次的日志重演信息
```

7.9.2.2 配置 dmmal.ini

拷贝一份原系统 dmmal.ini 文件，并加上自己一项，最终配置如下：

```
MAL_CHECK_INTERVAL     = 5      #MAL 链路检测时间间隔

MAL_CONN_FAIL_INTERVAL = 5      #判定 MAL 链路断开的的时间

[MAL_INST1]

MAL_INST_NAME          = GRP1_RT_01 #实例名，和 dm.ini 中的 INSTANCE_NAME 一致

MAL_HOST               = 192.168.0.141 #MAL 系统监听 TCP 连接的 IP 地址

MAL_PORT               = 61141      #MAL 系统监听 TCP 连接的端口

MAL_INST_HOST          = 192.168.1.131 #实例的对外服务 IP 地址

MAL_INST_PORT          = 32141 #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致

MAL_DW_PORT            = 52141 #实例本地的守护进程监听 TCP 连接的端口

MAL_INST_DW_PORT       = 33141 #实例监听守护进程 TCP 连接的端口

[MAL_INST2]
```

```
MAL_INST_NAME      = GRP1_RT_02
MAL_HOST            = 192.168.0.142
MAL_PORT            = 61142
MAL_INST_HOST       = 192.168.1.132
MAL_INST_PORT       = 32142
MAL_DW_PORT         = 52142
MAL_INST_DW_PORT    = 33142
```

```
[MAL_INST3]
```

```
MAL_INST_NAME      = GRP1_RT_03
MAL_HOST            = 192.168.0.143
MAL_PORT            = 61142
MAL_INST_HOST       = 192.168.1.133
MAL_INST_PORT       = 32143
MAL_DW_PORT         = 52143
MAL_INST_DW_PORT    = 33143
```

7.9.2.3 配置 dmarch.ini

修改 dmarch.ini，配置本地归档和即时归档。

```
[ARCHIVE_REALTIME]
ARCH_TYPE           = REALTIME      #实时归档类型
ARCH_DEST           = GRP1_RT_01    #实时归档目标实例名

[ARCHIVE_REALTIME2]
ARCH_TYPE           = REALTIME      #实时归档类型
ARCH_DEST           = GRP1_RT_02    #实时归档目标实例名

[ARCHIVE_LOCAL1]
ARCH_TYPE           = LOCAL         #本地归档类型
ARCH_DEST           = /dm/data/DAMENG/arch #本地归档文件存放路径
```

```

ARCH_FILE_SIZE      = 128    #单位 Mb, 本地单个归档文件最大值
ARCH_SPACE_LIMIT    = 0      #单位 Mb, 0 表示无限制, 范围 1024~4294967294M

```

7.9.2.4 配置 dmwatcher.ini

修改 dmwatcher.ini 配置守护进程，配置为全局守护类型，使用自动切换模式。

```

[GRP1]

DW_TYPE              = GLOBAL    #全局守护类型
DW_MODE              = AUTO      #自动切换模式
DW_ERROR_TIME        = 10        #远程守护进程故障认定时间
INST_RECOVER_TIME    = 60        #主库守护进程启动恢复的间隔时间
INST_ERROR_TIME       = 10        #本地实例故障认定时间
INST_OGUID            = 453331   #守护系统唯一 OGUID 值
INST_INI              = /dm/data/DAMENG/dm.ini    #dm.ini 配置文件路径
INST_AUTO_RESTART    = 1          #打开实例的自动拉起功能
INST_STARTUP_CMD      = /dm/bin/dmserver  #命令行方式启动
RLOG_SEND_THRESHOLD  = 0          #指定主库发送日志到备库的时间阈值，默认关闭
RLOG_APPLY_THRESHOLD = 0          #指定备库重演日志的时间阈值，默认关闭

```

7.9.2.5 启动备库

以 Mount 方式启动备库

```
./dmserver /dm/data/DAMENG/dm.ini mount
```

7.9.2.6 设置 OGUID

启动命令行工具 DIsql，登录备库设置 OGUID 值。

```

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>sp_set_oguid(453331);

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);

```

7.9.2.7 修改数据库模式

启动命令行工具 DISql，登录备库修改数据库为 Standby 模式：

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>ALTER DATABASE STANDBY;

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

7.9.3 动态添加 MAL 配置

本小节的步骤需要分别连接原系统中每个实例单独执行：

动态增加 MAL 中 GRP1_RT_03 的相关配置信息：

```
SF_MAL_CONFIG(1,0);

SF_MAL_INST_ADD('MAL_INST3','GRP1_RT_03','192.168.0.143',61142,'192.168.1.13
3',32143,52143,0,33143);

SF_MAL_CONFIG_APPLY();

SF_MAL_CONFIG(0,0);
```

7.9.4 动态添加归档配置

动态添加归档配置步骤如下：

1. 关闭原系统中所有实例的守护进程和监视器

动态添加归档要求数据库处于 MOUNT 状态，守护进程处于活动状态下不允许用户手动修改模式状态，因此需要先关闭守护进程，同时也是为了防止手动切换 MOUNT 状态后又被守护进程通知自动 Open，另外监视器配置也需要修改，因此在这里一并关闭。

关闭顺序为：

- 1) 关闭监视器
- 2) 关闭备库守护进程
- 3) 关闭主库守护进程

2. 手动修改原系统中所有实例为 MOUNT 状态

必须先修改主库，再修改备库，否则可能会引发主备数据同步失败导致主库挂起。

--临时修改 ALTER_MODE_STATUS，允许用户手动修改模式状态，动态添加完毕后需要再改回原值

```
SQL> SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

--临时关闭守护进程活动状态检测，允许用户手动修改模式状态，动态添加完毕后需要再改回原值

SQL> SP_SET_PARA_VALUE(1, 'DW_INACTIVE_INTERVAL', 0);

--修改数据库为 MOUNT 状态

SQL> ALTER DATABASE MOUNT;
```

3. 动态添加归档节点

分别连接原系统中的所有实例，动态添加 dmarch.ini 中归档节点。

```
SQL> ALTER DATABASE ADD ARCHIVELOG 'DEST= GRP1_RT_03, TYPE= REALTIME';
```

4. 手动修改原系统中所有实例为 OPEN 状态

必须先修改备库，再修改主库，否则可能会引发主备数据同步失败导致主库挂起。

```
--修改数据库为 OPEN 状态

SQL> ALTER DATABASE OPEN FORCE;

--将 ALTER_MODE_STATUS 改回原值

SQL> SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);

--将 DW_INACTIVE_INTERVAL 改回原值，本例中为 60

SQL> SP_SET_PARA_VALUE(1, 'DW_INACTIVE_INTERVAL', 60);
```

7.9.5 修改监视器 dmmonitor.ini

在 dmmonitor.ini 中添加新增的备库 GRP1_RT_03:

```
MON_DW_IP          = 192.168.0.143:52143
```

7.9.6 启动所有守护进程以及监视器

分别启动主库和备库（包括 GRP1_RT_03）的所有守护进程，最后启动监视器。

7.10 动态增加实时 DMDSC 备库

当需要进行系统扩容，希望系统运行不中断，或者影响运行的时间尽可能短时，可通过动态增加节点的方式进行。下面举例对实时主备进行动态增加 DSC 备库节点。

本例在 [7.2 配置实时主备集群](#) 的基础上，再增加一个 DMDSC 集群备库，DMDSC 集群

的两个节点实例名分别命名为“GRP1_RT_DSC01”、“GRP1_RT_DSC02”。

注意，本章节内容仅适用于原有的数据守护系统中不存在 DMDSC 集群的情况。

表 7.15 配置环境说明

机器名	IP 地址	初始状态	操作系统	备注
DW_S2	192.168.1.133 192.168.0.143	备库 GRP1_RT_DSC01 GRP1_RT_DSC02	Linux rh6-141.test 2.6.32-220.el6.x86_64	192.168.1.133 外部服务 IP; 192.168.0.143 内部通信 IP

详细步骤如下：

7.10.1 数据准备

1. 对主库进行联机备份操作：

```
SQL> BACKUP DATABASE BACKUPSET 'BACKUP_FILE_01';
```

2. 搭建 DMDSC 环境作为备库

使用 DMDSC 集群手册《DM8 共享存储集群》或者相应脚本搭建好两节点 DMDSC 集群环境，DMDSC 实例名分别为 GRP1_RT_DSC01、GRP1_RT_DSC02。

注意 dmcss 对 dmserver 的自动拉起功能先不要打开，避免影响到配置过程。搭建完成后，正常退出 DMDSC 集群的两个 dmserver 节点实例，不需要退出 dmcss 和 dmasmsvr。

3. 还原恢复新增备库

拷贝生成的备份集目录 BACKUP_FILE_01 到 143 上 /dm/data/ 目录，使用 DMRMAN 工具脱机还原。

```
./dmrman use_ap=2 dcr_ini=/dm/data/DSC/conf/dmdcr1.ini
RMAN>RESTORE DATABASE '/dm/data/DSC/DSC01/dm.ini' FROM BACKUPSET
'/dm/data/BACKUP_FILE_01'
RMAN>RECOVER DATABASE '/dm/data/DSC/DSC01/dm.ini' FROM BACKUPSET
'/dm/data/BACKUP_FILE_01'
RMAN>RECOVER DATABASE '/dm/data/DSC/DSC01/dm.ini' UPDATE DB_MAGIC
```

7.10.2 配置新备库 DMDSC 环境

7.10.2.1 配置 dm.ini

依次配置 DMDSC 备库所有节点的 dm.ini 文件，修改数据守护相关的参数配置。

(1) 配置 GRP1_RT_DSC01 的 dm.ini 文件

#实例名，建议使用“组名_守护环境_序号”的命名方式，总长度不能超过 16

```

INSTANCE_NAME          = GRP1_RT_DSC01

PORT_NUM               = 31431 #数据库实例监听端口

DW_INACTIVE_INTERVAL   = 60    #接收守护进程消息超时时间

ALTER_MODE_STATUS      = 0     #不允许手工方式修改实例模式/状态/OGUID

ENABLE_OFFLINE_TS      = 2     #不允许备库 OFFLINE 表空间

MAL_INI               = 1     #打开 MAL 系统

ARCH_INI               = 1     #打开归档配置

RLOG_SEND_APPLY_MON    = 64    #统计最近 64 次的日志发送信息

```

(2) 配置 GRP1_RT_DSC02 的 dm.ini 文件

```

INSTANCE_NAME          = GRP1_RT_DSC02

PORT_NUM               = 31432

DW_INACTIVE_INTERVAL   = 60

ALTER_MODE_STATUS      = 0

ENABLE_OFFLINE_TS      = 2

MAL_INI               = 1

ARCH_INI               = 1

RLOG_SEND_APPLY_MON    = 64

```

7.10.2.2 配置 dmmal.ini

在 DMDSC 集群的 dmmal.ini 文件基础上，增加主库 GRP1_RT_01、备库 GRP1_RT_02 的配置项。

```

MAL_CHECK_INTERVAL     = 5     #MAL 链路检测时间间隔

```

```
MAL_CONN_FAIL_INTERVAL = 5      #判定 MAL 链路断开的時間

[MAL_INST1]

MAL_INST_NAME           = GRP1_RT_01 #实例名，和 dm.ini 中的 INSTANCE_NAME 一致
MAL_HOST                = 192.168.0.141 #MAL 系统监听 TCP 连接的 IP 地址
MAL_PORT               = 61141         #MAL 系统监听 TCP 连接的端口
MAL_INST_HOST          = 192.168.1.131 #实例的对外服务 IP 地址
MAL_INST_PORT          = 32141 #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致
MAL_DW_PORT            = 52141 #实例本地的守护进程监听 TCP 连接的端口
MAL_INST_DW_PORT       = 33141 #实例监听守护进程 TCP 连接的端口

[MAL_INST2]

MAL_INST_NAME           = GRP1_RT_02
MAL_HOST                = 192.168.0.142
MAL_PORT               = 61142
MAL_INST_HOST          = 192.168.1.132
MAL_INST_PORT          = 32142
MAL_DW_PORT            = 52142
MAL_INST_DW_PORT       = 33142

[MAL_INST3]

MAL_INST_NAME           = GRP1_RT_DSC01 #实例名，和 dm.ini 中的 INSTANCE_NAME 一致
MAL_HOST                = 192.168.0.143 #MAL 系统监听 TCP 连接的 IP 地址
MAL_PORT               = 8338         #MAL 系统监听 TCP 连接的端口
MAL_INST_HOST          = 192.168.1.133 #实例的对外服务 IP 地址
MAL_INST_PORT          = 31431 #实例的对外服务端口，和 dm.ini 中的 PORT_NUM 一致
MAL_DW_PORT            = 3567 #实例本地的守护进程监听 TCP 连接的端口
MAL_INST_DW_PORT       = 4567 #实例监听守护进程 TCP 连接的端口

[MAL_INST4]

MAL_INST_NAME           = GRP1_RT_DSC02
```

```
MAL_HOST          = 192.168.0.143
MAL_PORT          = 8339
MAL_INST_HOST     = 192.168.1.133
MAL_INST_PORT     = 31432
MAL_DW_PORT       = 3568
MAL_INST_DW_PORT  = 4568
```

7.10.2.3 配置 dmarch.ini

编辑各个节点的 dmarch.ini 文件，增加本地归档、实时归档及远程归档配置。

(1) 修改 GRP1_RT_DSC01 的 dmarch.ini 文件

```
ARCH_LOCAL_SHARE  = 1

[ARCHIVE_LOCAL1]

    ARCH_TYPE      = LOCAL

    ARCH_DEST      = +DMDATA/DSC/DSC01/arch

    ARCH_FILE_SIZE = 128

    ARCH_SPACE_LIMIT = 0

[ARCHIVE_REMOTE]

    ARCH_TYPE      = REMOTE

    ARCH_DEST      = GRP1_RT_DSC02

    ARCH_FILE_SIZE = 128

    ARCH_SPACE_LIMIT = 0

    ARCH_INCOMING_PATH = +DMDATA/DSC/DSC02/arch

[ARCHIVE_REALTIME1]

    ARCH_TYPE      = REALTIME

    ARCH_DEST      = GRP1_RT_01

[ARCHIVE_REALTIME2]
```

```

ARCH_TYPE          = REALTIME
ARCH_DEST          = GRP1_RT_02

```

(2) 修改 GRP1_RT_DSC02 的 dmarch.ini 文件

```

ARCH_LOCAL_SHARE    = 1

[ARCHIVE_LOCAL1]

    ARCH_TYPE        = LOCAL
    ARCH_DEST        = +DMDATA/DSC/DSC02/arch
    ARCH_FILE_SIZE   = 128
    ARCH_SPACE_LIMIT = 0

[ARCHIVE_REMOTE]

    ARCH_TYPE        = REMOTE
    ARCH_DEST        = GRP1_RT_DSC01
    ARCH_FILE_SIZE   = 128
    ARCH_SPACE_LIMIT = 0
    ARCH_INCOMING_PATH = +DMDATA/DSC/DSC01/arch

[ARCHIVE_REALTIME1]

    ARCH_TYPE        = REALTIME
    ARCH_DEST        = GRP1_RT_01

[ARCHIVE_REALTIME2]

    ARCH_TYPE        = REALTIME
    ARCH_DEST        = GRP1_RT_02

```

7.10.2.4 配置 dmwatcher.ini

依次配置每个节点实例的 dmwatcher.ini 文件，放到各自 dm.ini 中指定的 CONFIG_PATH 目录下。

另外要注意，DMDSC 集群各节点实例的自动拉起是由各自本地的 dmcss 执行的，不是

由守护进程执行，如果要打开 DMDSC 集群的自动拉起，需要再去配置 dmdcr.ini 中的自动拉起参数，为避免 dmcsc 在所有配置步骤完成之前提前将 dmserver 自动拉起，这里先不修改 dmdcr.ini 配置，放到后面步骤中修改。

(1) 配置 GRP1_RT_DSC01 的 dmwatcher.ini 文件

```
[GRP1]

DW_TYPE           = GLOBAL   #全局守护类型

DW_MODE           = MANUAL   #手动切换模式

DW_ERROR_TIME     = 60       #远程守护进程故障认定时间

INST_RECOVER_TIME = 60       #主库守护进程启动恢复的间隔时间

INST_ERROR_TIME   = 30       #本地实例故障认定时间

INST_INI           = /dm/data/DSC/DSC01/dm.ini #dm.ini 配置文件路径

DCR_INI           = /dm/data/DSC/conf/dmdcr1.ini #dmdcr.ini 配置文件路径

INST_OGUID        = 453331   #守护系统唯一 OGUID 值

INST_STARTUP_CMD   = /dm/bin/dmserver #命令行方式启动

INST_AUTO_RESTART = 0        #关闭实例的自动启动功能

RLOG_SEND_THRESHOLD = 0      #指定主库发送日志到备库的时间阈值，默认关闭

RLOG_APPLY_THRESHOLD = 0     #指定备库重演日志的时间阈值，默认关闭
```

(2) 配置 GRP1_RT_DSC02 的 dmwatcher.ini 文件

```
[GRP1]

DW_TYPE           = GLOBAL

DW_MODE           = MANUAL

DW_ERROR_TIME     = 60

INST_RECOVER_TIME = 60

INST_ERROR_TIME   = 30

INST_INI           = /dm/data/DSC/DSC02/dm.ini

DCR_INI           = /dm/data/DSC/conf/dmdcr2.ini

INST_OGUID        = 453331

INST_STARTUP_CMD   = /dm/bin/dmserver

INST_AUTO_RESTART = 0

RLOG_SEND_THRESHOLD = 0
```

```
RLOG_APPLY_THRESHOLD = 0
```



实际配置时，相关的端口配置和 OGUID 值建议不要和手册示例使用完全相同
注意: 的值，避免多个用户在同一个环境下搭建不同的数据守护系统，出现消息混乱
或者端口冲突等问题。

7.10.2.5 配置 dmdcr.ini

DMDSC 集群中 dmserver 的自动拉起是由 dmcss 执行的，如果不需要打开 dmcss 的自动拉起功能，则可以跳过此章节。

否则需要修改 dmdcr.ini 中的自动拉起配置参数，此参数修改完成后，需要重启 dmcss 才可以生效，为了避免重启 dmcss 引发 dmasmsvr 被强制关闭，这里先将 dmcss 和 dmasmsvr 都正常退出。

另外要注意，本示例中是用命令行方式启动，启动参数中指定以 Mount 方式拉起 dmserver，如果是用服务方式启动，服务脚本中也一定要指定以 Mount 方式拉起 dmserver。

1. 打开 GRP1_RT_DSC01 的自动拉起参数

修改对应 dmcss 的 dmdcr.ini 文件，打开自动拉起参数。

```
#打开 DB 重启参数，命令行方式启动
DMDCR_DB_RESTART_INTERVAL = 60
DMDCR_DB_STARTUP_CMD = /dm/bin/dmserver path=/dm/data/DSC/DSC01/dm.ini
dcr_ini=/dm/data/DSC/conf/dmdcr1.ini mount
```

2. 打开 GRP1_RT_DSC02 的自动拉起参数

修改对应 dmcss 的 dmdcr.ini 文件，打开自动拉起参数。

```
#打开 DB 重启参数，命令行方式启动
DMDCR_DB_RESTART_INTERVAL = 60
DMDCR_DB_STARTUP_CMD = /dm/bin/dmserver path=/dm/data/DSC/DSC02/dm.ini
dcr_ini=/dm/data/DSC/conf/dmdcr2.ini mount
```

3. 重启 dmcss 和 dmasmsvr

```
./dmcss dcr_ini=/dm/data/DSC/conf/dmdcr1.ini
```

```
./dmcss dcr_ini=/dm/data/DSC/conf/dmdcr2.ini

./dmassvr dcr_ini=/dm/data/DSC/conf/dmdcr1.ini

./dmassvr dcr_ini=/dm/data/DSC/conf/dmdcr2.ini
```

7.10.2.6 启动 DMDSC 备库

启动 DMDSC 集群的两个 dmserver 实例，注意实例都要使用 Mount 方式启动。

如果 dmcss 打开有自动拉起功能，也可以等待 dmcss 将本地的 dmserver 实例自动拉起（要保证以 Mount 方式拉起）。

手动启动命令如下：

```
./dmserver /dm/data/DSC/DSC01/dm.ini DCR_INI=/dm/data/DSC/conf/dmdcr1.ini
mount

./dmserver /dm/data/DSC/DSC02/dm.ini DCR_INI=/dm/data/DSC/conf/dmdcr2.ini
mount
```

7.10.2.7 设置 OGUID

启动命令行工具 DIsql，连接 DMDSC 集群中的任意一个节点，设置 DMDSC 主库的 OGUID 值。

```
SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL>SP_SET_OGUID(453331);

SQL>SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

7.10.2.8 修改数据库模式

启动命令行工具 DIsql，连接 DMDSC 集群中的任意一个节点，修改 DMDSC 库为 Standby 模式。

```
SQL>ALTER DATABASE STANDBY;
```

7.10.3 动态添加 MAL 配置

本小节的步骤需要分别连接原系统中每个实例单独执行：

动态增加 MAL 中 GRP1_RT_DSC01、GRP1_RT_DSC02 的相关配置信息：

```
SF_MAL_CONFIG(1,0);

SF_MAL_INST_ADD('MAL_INST3','GRP1_RT_DSC01','192.168.0.143',8338,'192.168.1.133',31431,3567,0,4567);

SF_MAL_INST_ADD('MAL_INST4','GRP1_RT_DSC02','192.168.0.143',8339,'192.168.1.133',31432,3568,0,4568);

SF_MAL_CONFIG_APPLY();

SF_MAL_CONFIG(0,0);
```

7.10.4 动态添加归档配置

动态添加归档配置步骤如下：

1. 关闭原系统中所有实例的守护进程和监视器

动态添加归档要求数据库处于 MOUNT 状态，守护进程处于活动状态下不允许用户手动修改模式状态，因此需要先关闭守护进程，同时也是为了防止手动切换 MOUNT 状态后又被守护进程通知自动 Open，另外监视器配置也需要修改，因此在这里一并关闭。

关闭顺序为：

- 1) 关闭监视器
- 2) 关闭备库守护进程
- 3) 关闭主库守护进程

2. 手动修改原系统中所有实例为 MOUNT 状态

必须先修改主库，再修改备库，否则可能会引发主备数据同步失败导致主库挂起。

```
--临时修改 ALTER_MODE_STATUS，允许用户手动修改模式状态，动态添加完后需要再改回原值
SQL> SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

--临时关闭守护进程活动状态检测，允许用户手动修改模式状态，动态添加完后需要再改回原值
SQL> SP_SET_PARA_VALUE(1, 'DW_INACTIVE_INTERVAL', 0);

--修改数据库为 MOUNT 状态
```

```
SQL> ALTER DATABASE MOUNT;
```

3. 动态添加归档节点

分别连接原系统中的所有实例，动态添加 dmarch.ini 中归档配置项。

```
SQL> ALTER DATABASE ADD ARCHIVELOG 'DEST=GRP1_RT_DSC01/GRP1_RT_DSC02, TYPE=
REALTIME';
```

4. 手动修改原系统中所有实例为 OPEN 状态

必须先修改备库，再修改主库，否则可能会引发主备数据同步失败导致主库挂起。

--修改数据库为 OPEN 状态

```
SQL> ALTER DATABASE OPEN FORCE;
```

--将 ALTER_MODE_STATUS 改回原值

```
SQL> SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

--将 DW_INACTIVE_INTERVAL 改回原值，本例中为 60

```
SQL> SP_SET_PARA_VALUE(1, 'DW_INACTIVE_INTERVAL', 60);
```

7.10.5 修改监视器 dmmonitor.ini

在 dmmonitor.ini 中添加新增的备库 GRP1_RT_DSC01/GRP1_RT_DSC02:

```
MON_DW_IP          = 192.168.0.143:3567/192.168.0.143:3568
```

7.10.6 启动所有守护进程以及监视器

分别启动主库和备库（包括 DMDSC 集群 GRP1_RT_DSC01、GRP1_RT_DSC02）的所有守护进程，最后启动监视器。

8 利用 DEM 工具搭建数据守护

DM web 版数据库管理工具 (DEM) 提供数据守护的图形化搭建与管理功能。关于 DEM 工具的启动, 请查看 DEM 相关文档。DEM 启动后, 通过浏览器访问, 如下图所示:

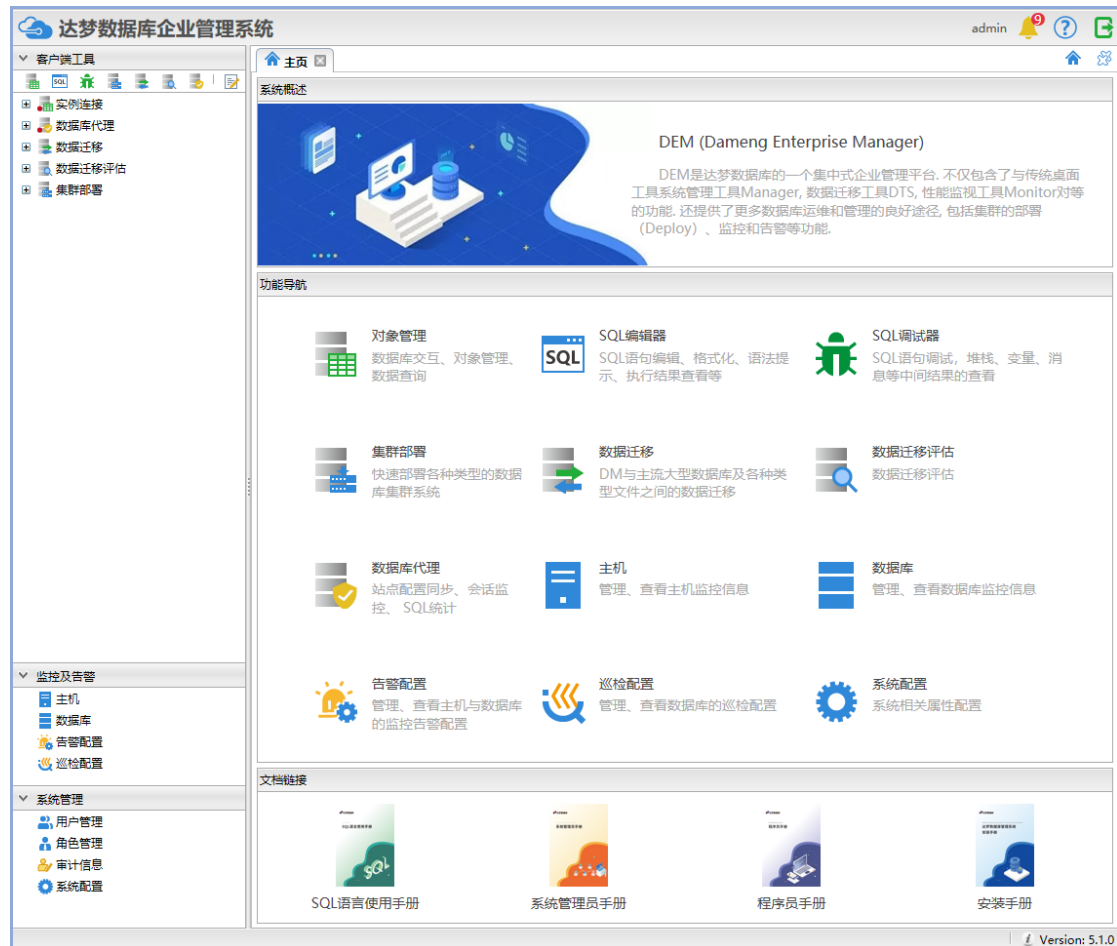


图 8.1 DEM 工具主页面

搭建与管理的集群类型有三种: MPP 主备、实时主备和读写分离集群。在 DEM 中, 对于集群的管理, 包括集群部署和集群监控。集群部署在“客户端工具”栏点击“部署集群”可以打开, 集群监控在“监控及告警”栏的“数据库”监控中添加对集群的监控即可打开。

本章以实时主备管理为例, 来介绍数据守护的部署与监控。MPP 管理、读写分离管理等更多详情, 请参考 DEM 工具帮助文档。

8.1 实时主备管理

下面将介绍实时主备管理是如何搭建的。实时主备管理分为两大步骤: 部署和监控。在

使用 DEM 对集群进行部署和监控之前需要在各节点所在机器部署 DM 数据库代理工具，关于 DM 数据库代理工具的部署与使用，请查看相关文档。

8.1.1 部署

在 DEM 的“客户端工具”栏，选择部署集群工具，打开新建集群部署的对话框，如下图：

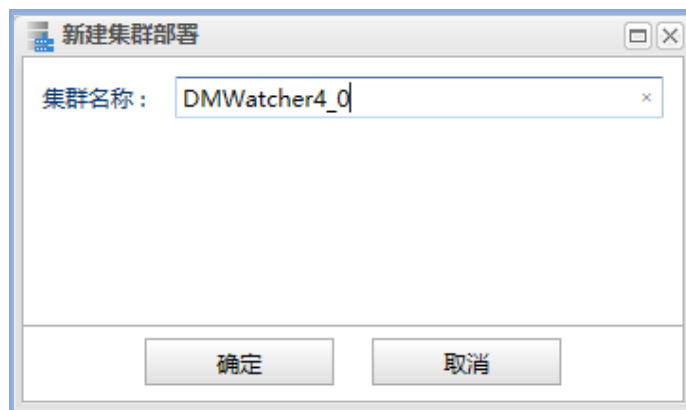


图 8.2 新建集群部署对话框

给定集群名称，点击“确定”以后，进入“选择部署集群类型”界面：

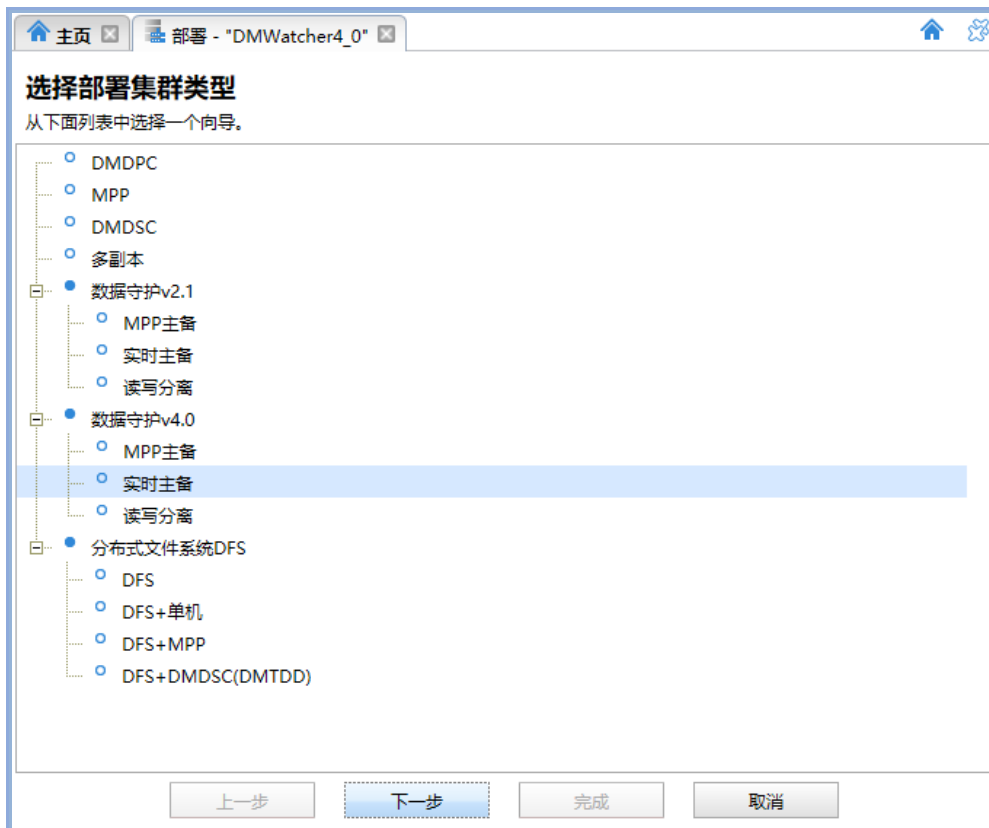


图 8.3 选择部署集群类型

选择集群类型为“数据守护 v4.0”下的“实时主备”，点击“下一步”后，进入“环境

准备”界面：

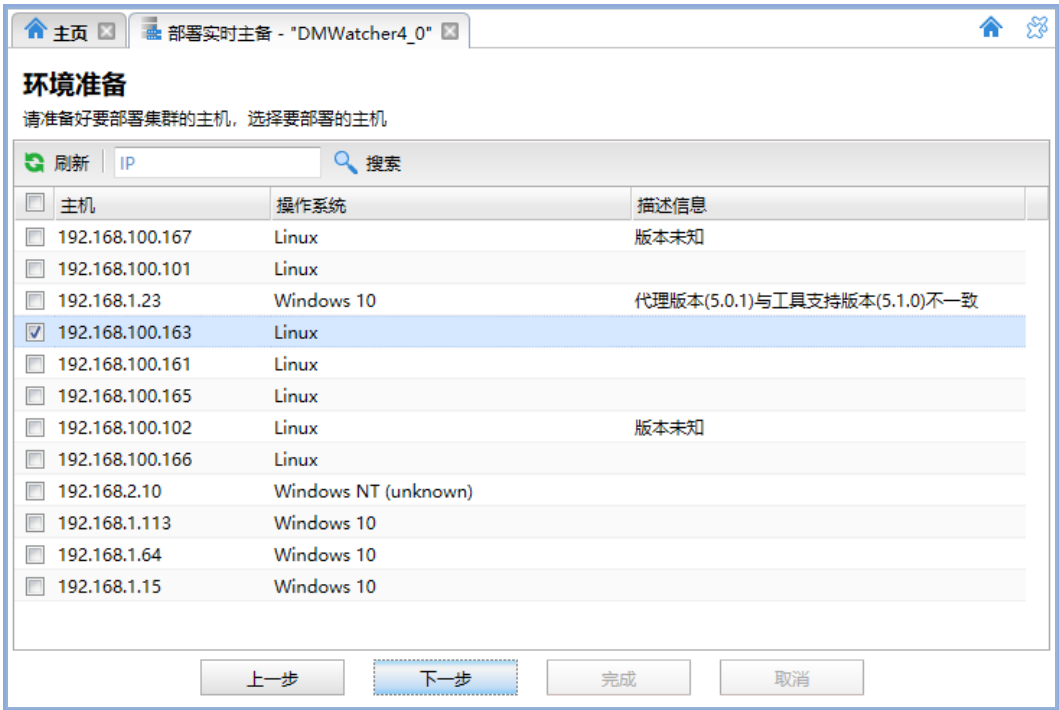


图 8.4 部署界面-环境准备

参数详解：

刷新：刷新主机列表。

搜索：输入 IP 地址，快速搜索指定主机。

选择要部署集群的主机，点击“下一步”，进入“实例规划”界面：

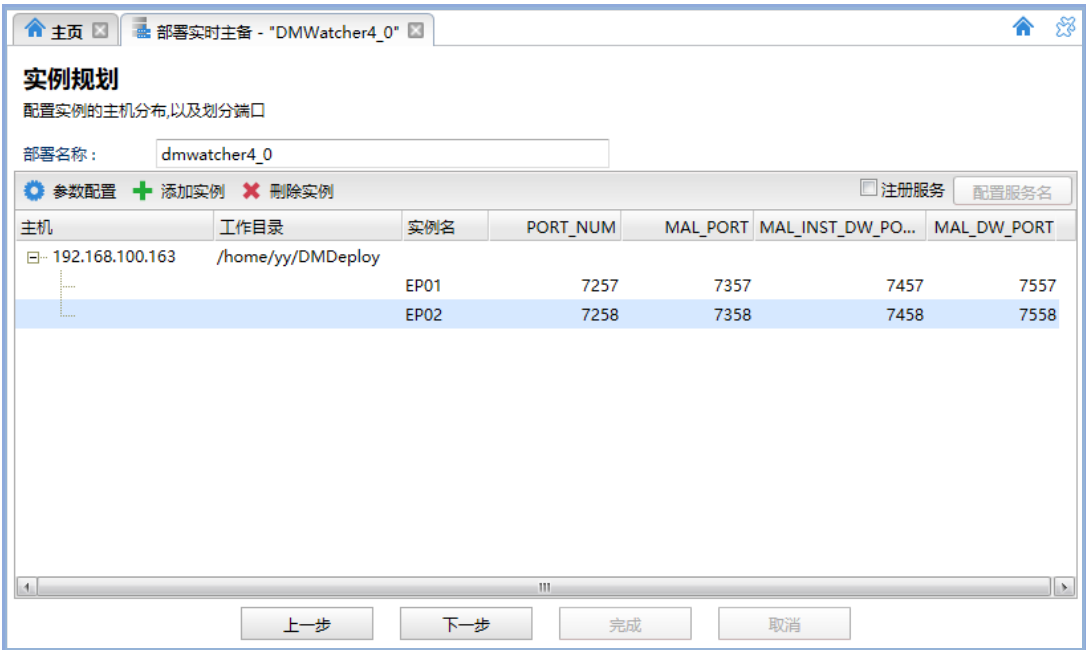


图 8.5 部署界面-实例规划

参数详解：

部署名称：部署名称，会在各部署主机的工作目录创建对应名称的目录来存储该集群。

参数配置：统一配置实例列表中所有实例的参数。另外也可以通过双击实例列表中某一个实例的某一个参数进行单独配置。

添加实例：添加实例，可以在选择的主机中添加实例，添加之后会出现在实例列表中。

删除实例：删除选中的实例。

注册服务：如果想让 dmserver、dmwatcher 和 dmmonitor 服务开机自启动，则需把 dmserver、dmwatcher 和 dmmonitor 注册成服务，注册完成后重启机器时，就会自动启动 dmserver、dmwatcher 和 dmmonitor 服务。如果选择了注册服务，该部署工具只会把 dmserver、dmwatcher、确认监视器注册成服务，普通监视器不注册服务。

配置服务名：配置注册的服务名，默认 dmserver 服务名为 DmService_实例名，dmwatcher 服务名为 DmWatcherService_实例名，确认监视器服务名为 DmMonitorService。

在配置好实例的分布，以及规划好端口后，点击“下一步”，进入“主备关系配置”界面：

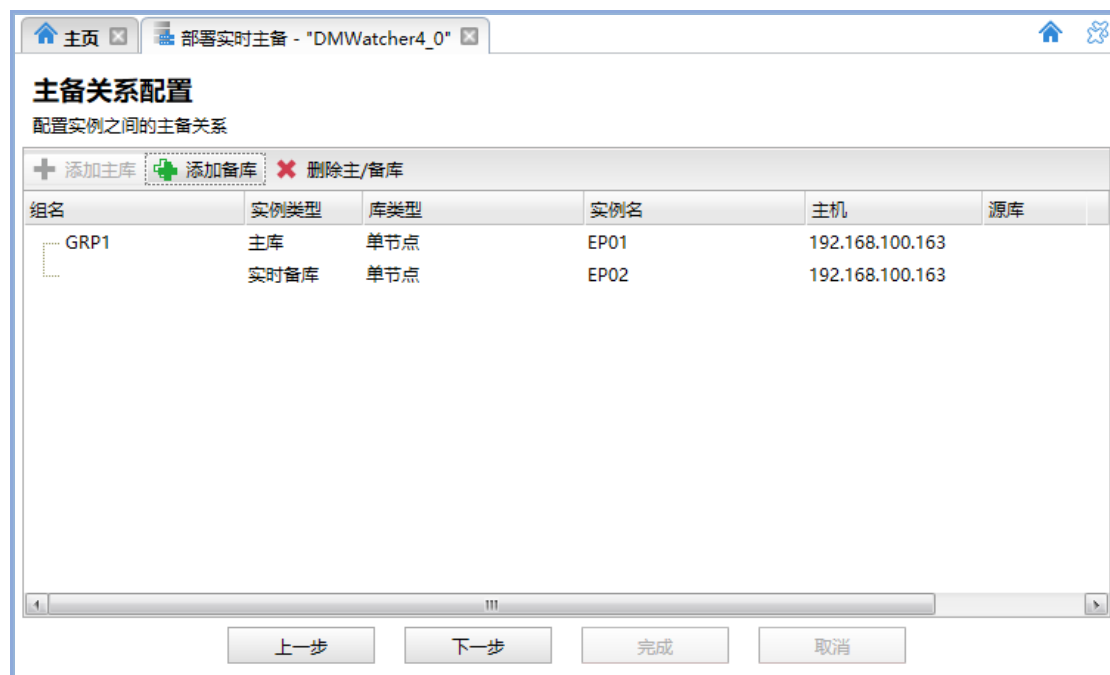


图 8.6 部署界面-主备关系配置

参数详解：

- 组名：可修改守护组名。
- 添加主库：从配置的实例中选择实例作为主库。
- 添加备库：从配置的实例中选择实例作为备库。
- 删除主/备库：删除实例的主/备库配置。

在配置好实例的分布，以及规划好端口后，点击“下一步”，进入“数据准备”界面：

主页部署实时主备 - "DMWatcher4_0"

数据准备

准备实例的数据,可以是新初始化的数据,也可以通过备份还原已有库的数据

初始化新库

使用已存在的库

初始化库参数配置

数据库名: DAMENG

簇大小: 16 页

页大小: 8 K

日志文件大小: 256 (M:64~2048)

时区设置: +08:00 (-12:59~+14:00)

页面检查: 不启用

字符集: GB18030

USBKEY-PIN: (长度: 1~48)

key文件: 浏览

☒ 字符串比较大小写敏感

☐ 空格填充模式

☐ VARCHAR类型以字符为单位

☐ 启用日志文件加密

☒ 改进的字符串HASH算法

☐ 四权分立

☐ 启用全库加密

加密算法: 请配置dll路径!

☐ 四权分立

☐ 单独配置口令(留空表示使用默认口令)

☐ 使用同一口令

SYSDBA: 口令: 确认口令: 默认口令: (SYSDBA)

SYSAUDITOR: 口令: 确认口令: (SYSAUDITOR)

上一步

下一步

完成

取消

图 8.7 部署界面-数据准备

参数详解：

- 初始化新库：初始化一个新库，作为守护系统的实例。
- 使用已存在的库：指定已经存在的库作为守护系统的实例。指定库所在的主机、工作空间、INI 路径、库的登录用户名和密码。如下所示：

270

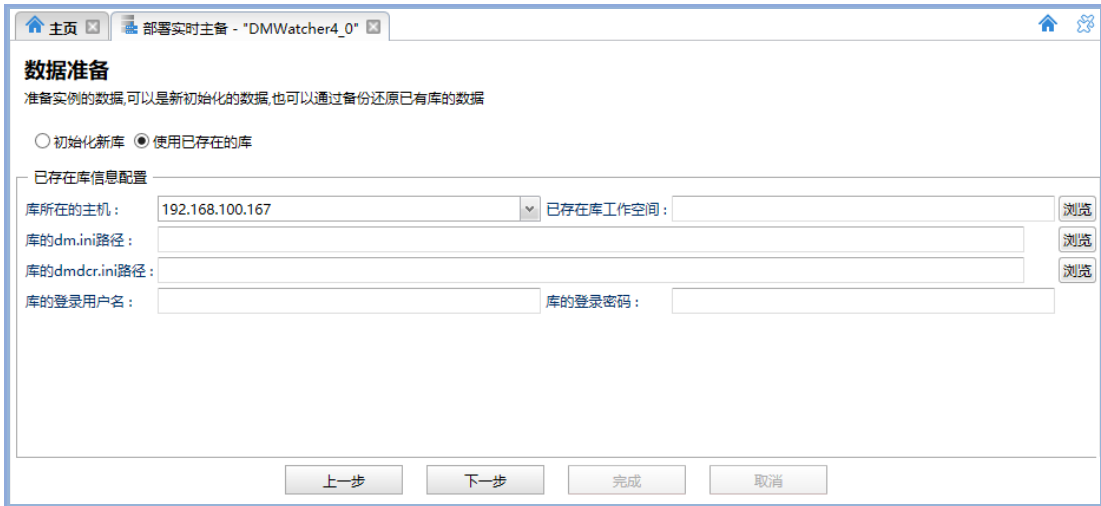


图 8.8 部署界面-数据准备 (使用已存在的库)

准备好数据之后，点击“下一步”，进入“dm.ini 配置”界面，配置 dm.ini 相关属性。

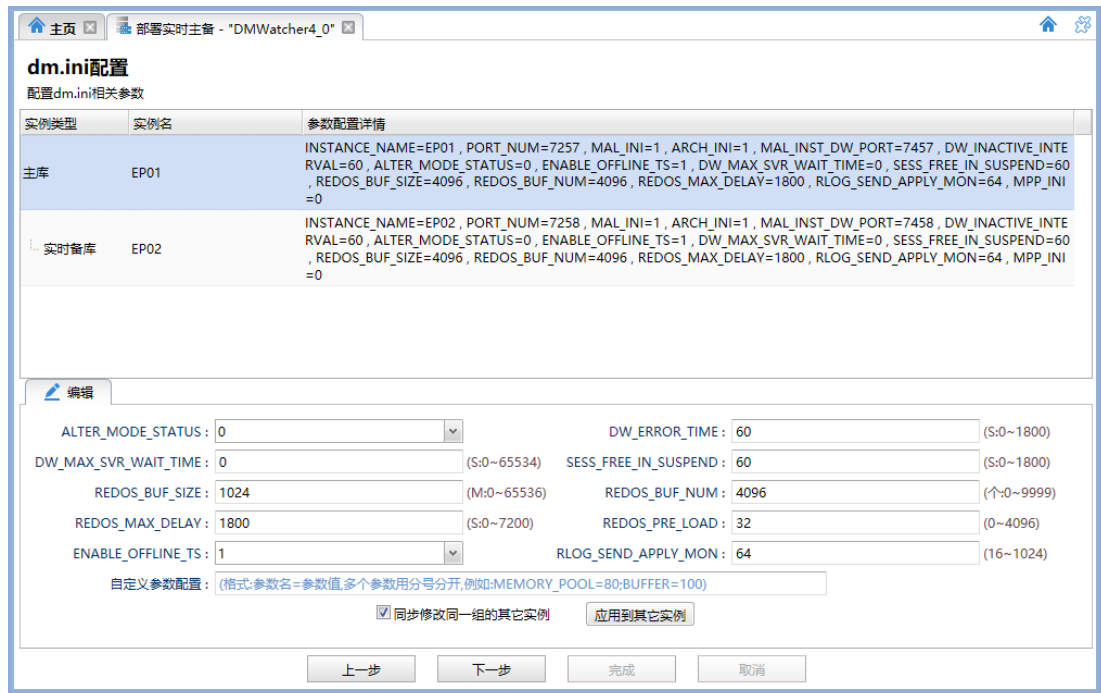


图 8.9 dm.ini 配置界面

参数说明：

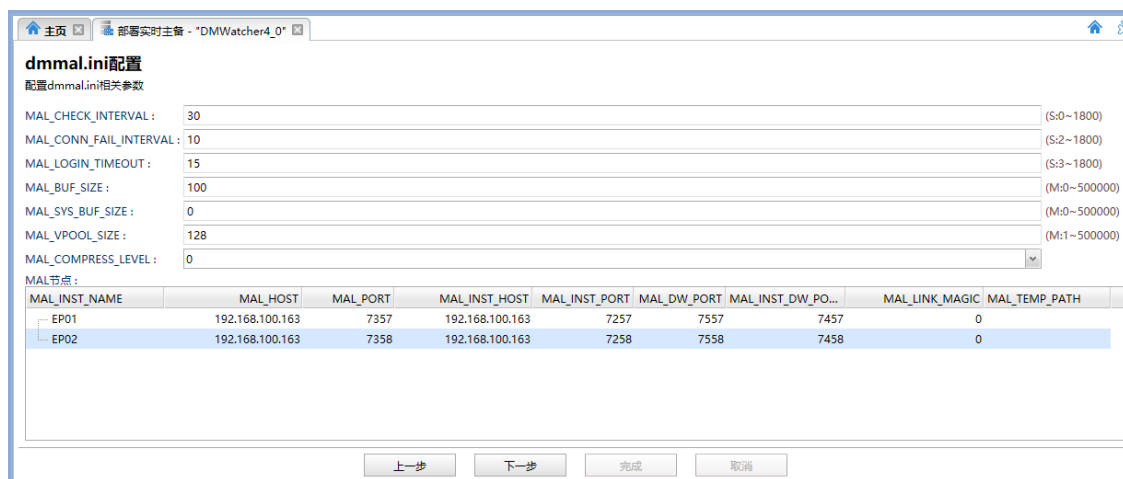
在面板的上面表格中显示各实例的 dm.ini 参数配置，选择每个实例可以在下面编辑面板中编辑对应实例的每一个 dm.ini 参数。

同步修改同一组的其它实例：修改的参数，同步应用到所选择的实例的同组的其它实例。

应用到其它实例：可以选择哪些参数的修改应用到哪些实例上。

具体 dm.ini 参数的配置参考 5.1 dm.ini。

点击“下一步”，进入“dmmal.ini 配置”界面，配置 dmmal.ini 相关属性。



该界面用于配置 dmmal.ini 参数。顶部有面包屑导航：主页 > 部署实时主备 - "DMWatcher4_0"。标题为“dmmal.ini配置”，副标题为“配置dmmal.ini相关参数”。

配置项包括：

- MAL_CHECK_INTERVAL: 30 (S:0~1800)
- MAL_CONN_FAIL_INTERVAL: 10 (S:2~1800)
- MAL_LOGIN_TIMEOUT: 15 (S:3~1800)
- MAL_BUF_SIZE: 100 (M:0~500000)
- MAL_SYS_BUF_SIZE: 0 (M:0~500000)
- MAL_VPOOL_SIZE: 128 (M:1~500000)
- MAL_COMPRESS_LEVEL: 0

MAL节点表：

MAL_INST_NAME	MAL_HOST	MAL_PORT	MAL_INST_HOST	MAL_INST_PORT	MAL_DW_PORT	MAL_INST_DW_PO...	MAL_LINK_MAGIC	MAL_TEMP_PATH
EP01	192.168.100.163	7357	192.168.100.163	7257	7557	7457	0	
EP02	192.168.100.163	7358	192.168.100.163	7258	7558	7458	0	

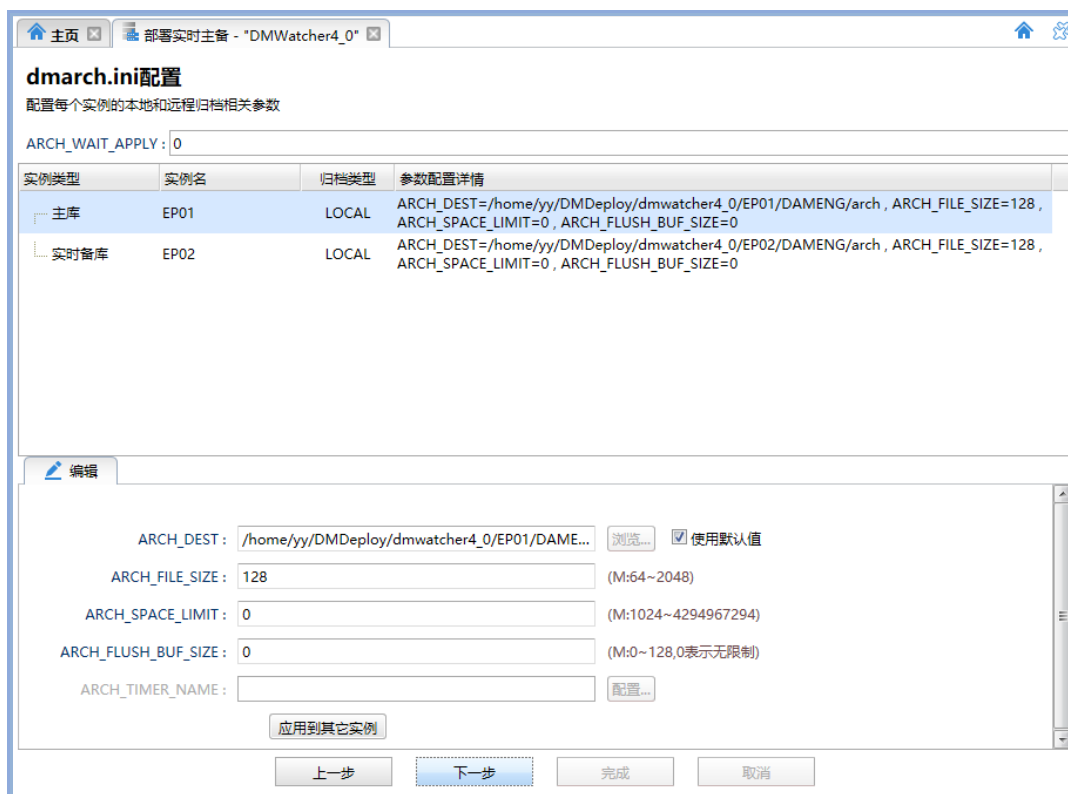
底部有“上一步”、“下一步”、“完成”、“取消”按钮。

图 8.10 dmmal.ini 配置界面

参数说明：

具体 dmmal.ini 参数的配置参考 [5.2 dmmal.ini](#)。

点击“下一步”，进入“dmarch.ini 配置”界面，配置 dmarch.ini 的属性，异步备库需要配置定时器信息。



该界面用于配置 dmarch.ini 参数。顶部有面包屑导航：主页 > 部署实时主备 - "DMWatcher4_0"。标题为“dmarch.ini配置”，副标题为“配置每个实例的本地和远程归档相关参数”。

ARCH_WAIT_APPLY: 0

实例配置表：

实例类型	实例名	归档类型	参数配置详情
主库	EP01	LOCAL	ARCH_DEST=/home/yy/DMDeploy/dmwatcher4_0/EP01/DAMENG/arch, ARCH_FILE_SIZE=128, ARCH_SPACE_LIMIT=0, ARCH_FLUSH_BUF_SIZE=0
实时备库	EP02	LOCAL	ARCH_DEST=/home/yy/DMDeploy/dmwatcher4_0/EP02/DAMENG/arch, ARCH_FILE_SIZE=128, ARCH_SPACE_LIMIT=0, ARCH_FLUSH_BUF_SIZE=0

下方有“编辑”按钮。

配置项包括：

- ARCH_DEST: /home/yy/DMDeploy/dmwatcher4_0/EP01/DAME... (浏览... 使用默认值)
- ARCH_FILE_SIZE: 128 (M:64~2048)
- ARCH_SPACE_LIMIT: 0 (M:1024~4294967294)
- ARCH_FLUSH_BUF_SIZE: 0 (M:0~128,0表示无限制)
- ARCH_TIMER_NAME: (配置...)

底部有“应用到其它实例”按钮。

底部有“上一步”、“下一步”、“完成”、“取消”按钮。

图 8.11 dmarch.ini 配置界面

参数说明：

在面板的上面表格中显示各实例的 dmarch.ini 参数配置，选择每个实例可以在下面编辑面板中编辑对应实例的每一个 dmarch.ini 参数。

应用到其它实例：可以选择哪些参数的修改应用到哪些实例上。

具体 dmarch.ini 参数的配置参考 5.3 dmarch.ini。

点击“下一步”，进入“dmwatcher.ini 配置”界面，配置 dmwatcher.ini 的属性。

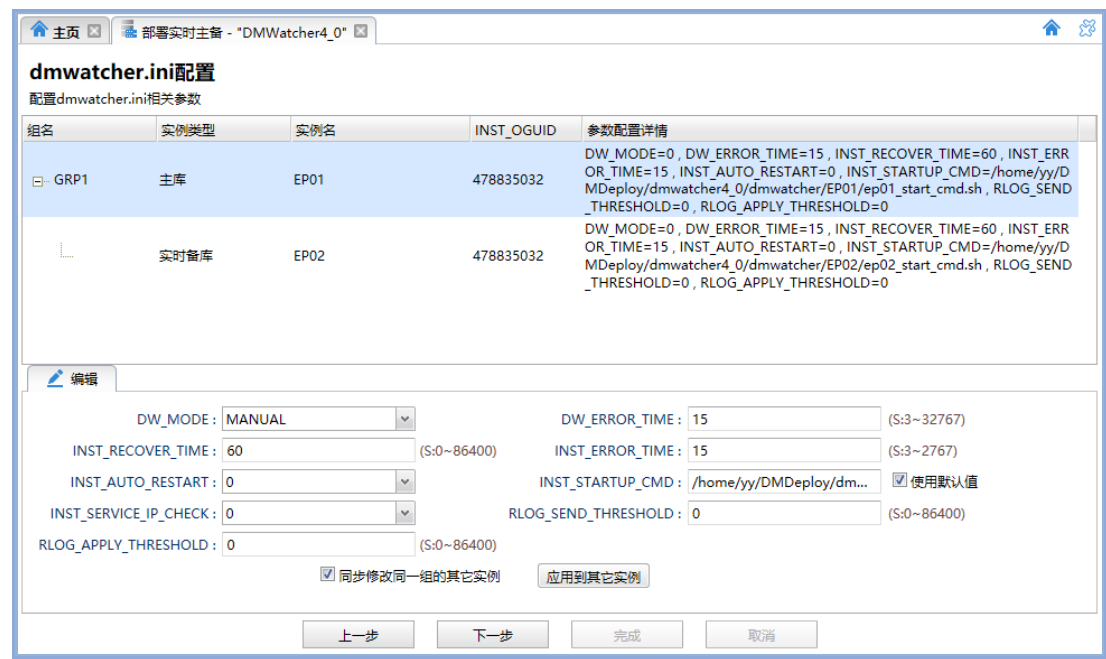


图 8.12 dmwatcher.ini 配置界面

参数说明：

在面板的上面表格中显示各实例的 dmwatcher.ini 参数配置，选择每个实例可以在下面编辑面板中编辑对应实例的每一个 dmwatcher.ini 参数。

同步修改同一组的其它实例：修改的参数，同步应用到所选择的实例的同组的其它实例。

应用到其它实例：可以选择哪些参数的修改应用到哪些实例上。

具体 dmwatcher.ini 参数的配置参考 5.4 dmwatcher.ini。

点击“下一步”，进入“监视器 dmmonitor 配置”界面，配置监视器相关属性。

监视器dmmonitor配置

配置监视器dmmonitor相关属性

☒ 是否部署监视器

监视器主机: 192.168.100.163 监视器工作空间: /home/yy/DMDeploy 浏览

监视器模式: 确认监视器

日志文件路径: 浏览

记录日志的时间间隔: 1 (S:5~3600,0,1)

单个日志文件大小: 64 (M:16~2048)

日志总空间大小: 0 (M:256~4096,0)

☒ 启动监视器

上一步 下一步 完成 取消

图 8.13 监视器配置界面

参数说明:

是否部署监视器: 选择是否部署监视器。

监视器主机: 选择要将监视器部署到那个主机上。

监视器工作空间: 配置监视器部署在主机的工作空间。

启动监视器: 选择部署完成后, 是否启动监视器。

其它监视器参数的配置参考 [5.5 dmmonitor.ini](#)。

点击“下一步”, 进入“上传服务器文件”界面, 请选择上传的服务器文件。

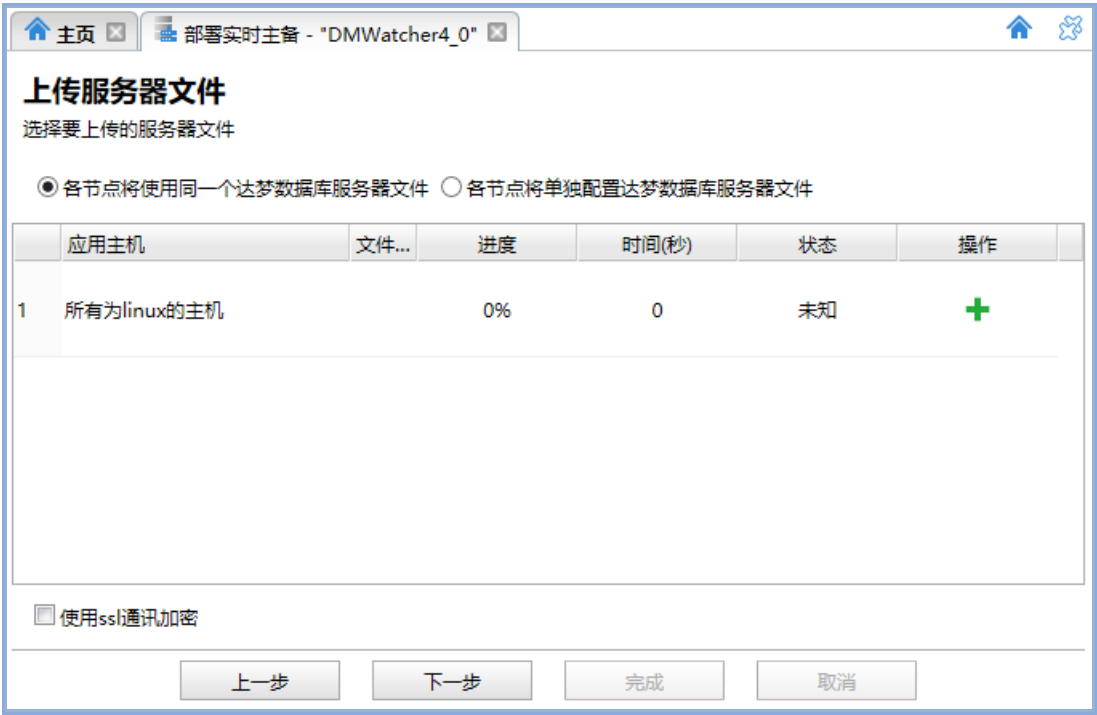


图 8.14 上传服务器文件

参数说明：

各节点将使用同一个达梦数据库服务器文件：选择该选项，则为所有为 Linux 的主机上传一份服务器文件，为所有为 Windows 的主机上传一份服务器文件。

各节点将单独配置达梦数据库服务器文件：为每一个主机单独上传服务器文件。

使用 ssl 通讯加密：如果上传的服务器是使用 ssl 通讯加密的，则需要上传客户端 SYSDBA 的 ssl 密钥文件，和输入 ssl 验证密码。

点击“下一步”，进入“详情总览”界面。列出将要部署的集群环境的所有配置信息：



图 8.15 详情总览

点击“下一步”，开始执行部署任务。执行过程如下：

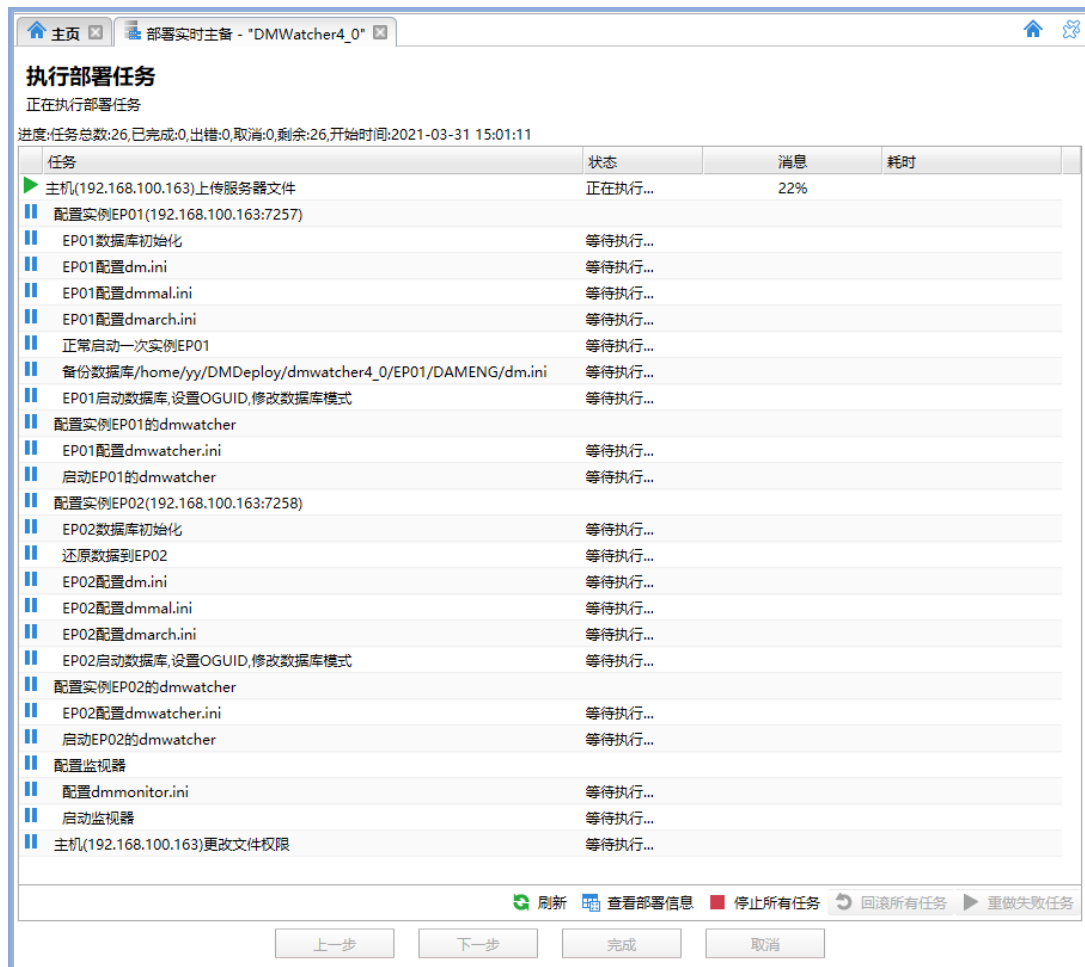


图 8.16 执行部署任务界面

参数说明：

刷新：刷新任务列表。

查看部署信息：对部署配置信息进行详细浏览。

停止所有任务：停止执行所有任务。

回滚所有任务：执行结束后，可以回滚所有执行的任务，清除环境。

重做失败任务：重新执行失败的任务。

部署任务执行完成后显示如下：



图 8.17 部署任务执行完成界面

参数说明：

完成：部署完成，关闭部署页面。

添加到监控：把该数据守护环境添加到监控中，并跳转到监控页面。

至此，数据守护搭建完成。

8.1.2 监控

在 DEM 的“监控与告警”栏，双击“数据库”， 在 DEM 右边面板打开数据库监控面板。

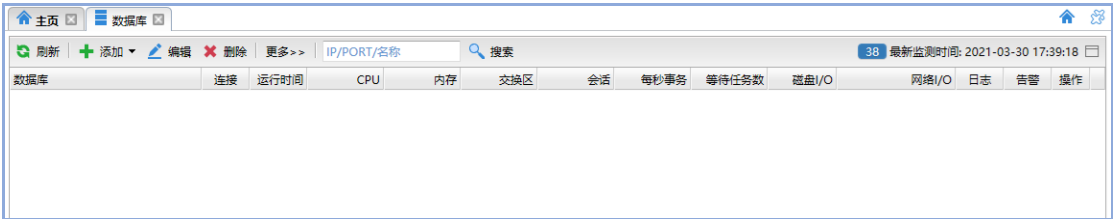


图 8.18 数据库监控页面

选择工具栏“添加”按钮，下拉选择“集群”。打开集群添加对话框：

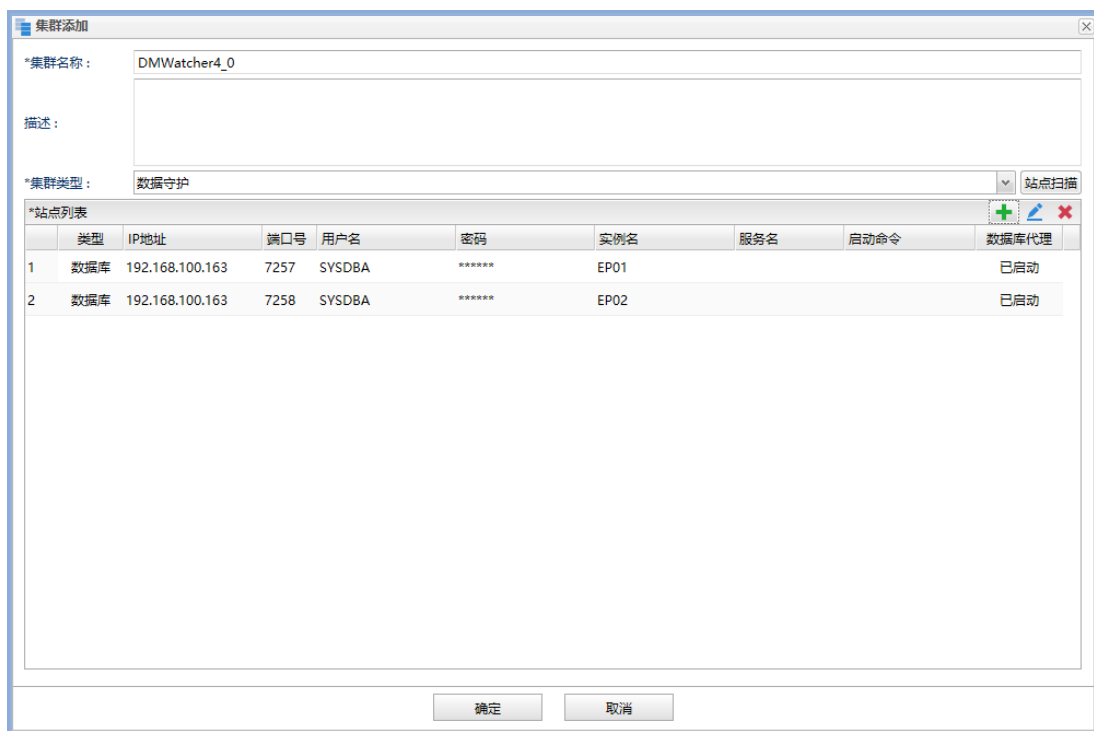


图 8.19 集群添加对话框

给定集群名称，选择集群类型“数据守护”，点击站点扫描，输入主库的连接信息，即可扫描出对应的集群节点，点击“确定”，把该集群添加到数据库监控页面，则可以对该集群进行监控管理。



图 8.20 集群监控页面-连接节点

对集群的监控与管理操作包括集群分析，集群管理，启动/停止集群，SQL 监控，表监控等，具体操作请查看 DEM 帮助手册。

9 版本升级

达梦数据库产品注重向下兼容性，虽然版本不断地更新升级，但在绝大多数情况下，使用高版本的执行码启动低版本的数据库时，会自动执行一系列更新升级动作。对于待升级的数据库，用户只需要保证它之前是正常退出的即可。

在数据库升降级操作前，建议先进行数据备份。

本章节中的升级步骤，是基于待升级的老库版本号等于或者高于 V7.1.6.11 的前提来写的。需要注意的是，DM 进行的三项较大的功能改进（V8.1.1.15 开始支持 RLOG_PKG 日志格式、V8.1.1.101 开始支持回滚管理段、V8.1.1.171 开始调整了数据页格式），使得待升级的库在跨这三个版本进行升级时需要进行额外的升级操作，下面进行详细的说明。

9.1 日志格式升级

从 V8.1.1.15 版本开始，DM 对 REDO 日志格式进行了升级，与老的日志格式不兼容，因此要求数据库在执行升级操作前，必须是使用之前版本的执行码执行正常退出后的库，以此来保证所有数据都已经刷盘，否则使用新版本的执行码启动数据库时，无法根据老的 REDO 日志对故障重启的数据库执行重做 REDO 日志、归档文件修复等动作，无法正常完成数据库升级。

9.1.1 版本说明

支持 DM7、DM8 老版本的库升级到 V8.1.1.15 或更高的数据库版本。

DM8 老版本的库正常退出后，允许直接启动升级到 V8.1.1.15 或者更高版本的 DM8，但必须要严格遵守 [9.1.2 升级步骤](#) 中的升级步骤。

DM7 老版本的库，针对不同版本有一些升级限制，具体说明如下：

- V7.1.6.11 之后的版本（包括 V7.1.6.11）

允许直接启动升级到 V8.1.1.15 或者更高版本的 DM8，同样也需要严格遵守升级步骤。

- 低于 V7.1.6.11 的版本

需要先升级到 V7.6.0.183 或者更高版本的 DM7，对日志版本号进行升级，然后

才允许升级到 V8.1.1.15 或者更高版本的 DM8。需要注意的是，V7.1.5.125 以及更低版本的库，如果打开了日志加密，则不再支持升级到 V8.1.1.15 或更高的数据库版本。

低于 V7.1.6.11 的版本需要升级两次，这两次版本升级的步骤是完全相同的，“升级到 V7.6.0.183 或者更高版本的 DM7”的步骤可以参考下文中描述的升级步骤，只是将升级步骤中的版本号替换为“V7.6.0.183 或者更高版本的 DM7”。

9.1.2 升级步骤

数据守护版本号从 V1.0 开始，目前已经支持到 V4.0，如果是只升级到 V3.0 或者 V3.0 之前的版本，配置文件和 dmwatcher.ctl 控制文件都可以向下兼容，REDO 日志格式也可以兼容，操作相对比较简单，这里不再对其进行说明。

此处只对升级到 V4.0 的操作步骤进行说明，V4.0 取消了 dmwatcher.ctl 控制文件的大部分功能，将其转移到 SYSOPENHISTORY 系统表上，守护进程的判断逻辑也随之做了较大改动，因此升级到 V4.0 时需要严格按照本文说明的操作步骤执行，否则升级过程中可能会出现守护进程或者监视器异常。

在执行升级步骤之前，需要特别注意以下几项说明：

1. V4.0 和 V3.0 的数据库 REDO 日志格式不兼容，因此要求数据库在执行升级操作之前，必须是使用之前版本的服务器执行码按照数据守护的退出顺序正常退出的库，包括主库和备库，对 DSC 集群，要求所有节点都处于 OK 状态并正常退出。

2. 升级后老的归档日志文件不再可用，因此需要确保备库在升级前和主库的数据处于一致状态，否则升级完成后，主库无法再从老的归档文件中收集数据同步到备库上，这种情况下只能对升级后的主库进行备份，再还原到备库上。

3. 升级后老的归档日志文件不再可用，因此必须在执行升级前先将归档日志文件全部从归档目录中移走，避免升级后再次降级时，误判归档日志文件不连续。

4. 主备库在升级过程中，需要重建 SYSOPENHISTORY 系统表，备库通过重演 REDO 日志进行重建，在此期间守护进程、监视器无法获取到主备库有效的 Open 历史信息，可能会出现强制退出，因此要求守护进程和监视器一定要放到最后启动，必须严格按照升级步骤执行。

5. V4.0 要求主备库的 DB_MAGIC 不能相同，升级前如果备库和主库的 DB_MAGIC

相同，则升级完成后主备库之间将无法同步数据，此时只能对升级后的主库进行备份，再还原到备库上完成升级。

6. V4.0 中异步备库的配置方式和 V3.0 不同，如果升级前有异步备库，升级后需要参考《DM8 数据守护与读写分离集群 V4.0》调整异步备库的配置方式。

7. 如果升级中遇到操作不当导致备库没有同步升级，则可以先单独升级主库，对升级后的主库进行备份，再还原到备库上，完成备库升级。

8. DSC 数据守护的升级，同样要求所有节点都是 OK 状态并正常退出，DM7 不支持 DSC 数据守护，因此不涉及 DCR Disk 和 Voting Disk 的重新初始化。

数据守护环境必须严格按照升级步骤执行，否则升级过程无法正常完成。数据守护环境升级步骤如下：

1. 确保主备库数据一致的情况下按照正确的退出顺序退出整个数据守护环境，包括主库、备库、守护进程和监视器。

2. 手动修改主备库 dm.ini 中的 ALTER_MODE_STATUS 为 1，如果是 DSC 集群，则每个节点的 dm.ini 都要修改。

3. 使用 V8.1.1.15 或者更高版本的执行码，Mount 方式启动所有主备库。

4. 使用 Disql 强制 Open 所有备库，如果备库是 DSC 集群，则需要依次连接每个节点执行。

```
SQL> Alter database open force;
```

注意：备库必须在主库之前 Open，否则备库无法重演主库升级产生的 REDO 日志，会导致备库无法完成升级。

5. 使用 Disql 登录主库，依次执行以下语句，如果主库是 DSC 集群，则需要依次连接每个节点执行。

1) 设置 DSC 备库的控制节点为归档目标节点。

如果不存在 DSC 备库，则可跳过此步骤，示例中的实例名是 DSC 备库的控制节点名称。

注意：DSC 备库只有控制节点允许接收主库的 REDO 日志，因此此处的归档目标只能设置为 DSC 备库的控制节点，如果设置成普通节点，会导致 DSC 备库无法完成升级，用 V8.1.1.15 或者更高版本的执行码在未升级的 DSC 备库上执行操作可能会引发异常。

```
SQL> SP_SET_ARCH_STATUS('实例名', 0);
```

2) 强制 Open 主库。

```
SQL> Alter database open force;
```

6. 主备库全部 Open 完成后，可分别连接主备库查询是否都存在 SYSOPENHISTORY 表，验证主备库是否正确升级。

备库重演 REDO 日志可能有一定延时，可通过分别查询主备库的 PKG_SEQ、LSN 信息确认主备库数据一致的情况下，再查询备库上是否存在 SYSOPENHISTORY 系统表。

主备库数据一致的判断方式：备库重演到的 PKG_SEQ 为主库的 GLOBAL_NEXT_SEQ 减 1，APPLY_LSN 和主库的 FILE_LSN 相同，DB_MAGIC 相同。如果不满足此条件，则需要分别在主备库上重复多次查询，主库可能 GLOBAL_NEXT_SEQ 和 FILE_LSN 还在增长，不一定第一次查询就能满足条件，如果查询结果确实无法匹配，只要能观察到备库的重演 PKG_SEQ 和 APPLY_LSN 在随主库增长，就可以尝试查询是否存在 SYSOPENHISTORY 表。

查询主库：

```
SQL> SELECT DB_MAGIC, GLOBAL_NEXT_SEQ, FILE_LSN FROM V$RLOG;
```

行号	DB_MAGIC	GLOBAL_NEXT_SEQ	FILE_LSN
1	460099052	4427	106610

```
SQL>SELECT * FROM SYSOPENHISTORY;
```

查询备库：

```
SQL> SELECT P_DB_MAGIC, PKG_SEQ_ARR, APPLY_LSN_ARR FROM V$RAPPLY_LSN_INFO;
```

行号	P_DB_MAGIC	PKG_SEQ_ARR	APPLY_LSN_ARR
1	460099052	(4426)	(106610)

```
SQL>SELECT * FROM SYSOPENHISTORY;
```

执行到这个步骤，主库已经升级完成，如果备库上没有查到 SYSOPENHISTORY 系统表信息，或者查询动态视图时备库出现异常退出情况，则说明备库没有升级成功，有可能是前面执行的升级步骤不对，或者备库 DB_MAGIC 和主库相同，或者备库升级前和主库数据不完全一致导致无法同步数据，这种情况下，只能正常退出主库和所有备库，对主库进行备份后再还原到备库上，完成备库升级，然后从步骤（9）开始继续执行。

7. 在步骤 6 查询结果正确的情况下，正常退出主库。

8. 正常退出所有备库。

9. 重新修改主备库 dm.ini 中的 ALTER_MODE_STATUS 为 0, 如果是 DSC 集群, 则每个节点的 dm.ini 都要修改。

10. 使用 V8.1.1.15 或者更高版本的执行码, Mount 方式启动所有主备库。

11. 启动所有主备库本地的守护进程。

12. 启动监视器。

至此, 整个数据守护环境升级完成, 可以通过执行监视器 show 命令观察守护系统是否运行正常, 主备库如果都运行在 Open 状态, 则说明升级成功。

如果升级过程中遇到服务器、守护进程或者监视器异常退出, 说明没有严格按照上述的升级步骤执行, 此时主备库数据可能已经不一致, 如果要继续完成升级, 只能先单独将主库升级完成, 再备份还原到备库上。

单独升级主库, 再备份还原到备库的执行步骤如下:

1. 使用 V8.1.1.15 或者更高版本的执行码, Mount 方式启动主库。

2. 依次执行以下语句 Open 主库。

如果主库是 DSC 集群, 则需要依次连接每个节点执行。

```
SQL> SP_SET_ALL_ARCH_STATUS(1);

SQL> SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 1);

SQL> ALTER DATABASE OPEN FORCE;

SQL> SP_SET_PARA_VALUE(1, 'ALTER_MODE_STATUS', 0);
```

3. 正常退出主库, 完成主库升级。

4. 对主库进行脱机备份, 再重新还原到备库上。

5. 使用 V8.1.1.15 或者更高版本的执行码, Mount 方式启动所有主备库。

6. 启动所有主备库本地的守护进程。

7. 启动监视器。

至此, 整个数据守护环境升级完成, 可以通过执行监视器 show 命令观察守护系统是否运行正常, 主备库如果都运行在 Open 状态, 则说明升级成功。

9.2 回滚管理段升级

DM 从 V8.1.1.101 版本开始支持回滚管理段, 用于存放事务信息, 在系统故障重启或者 DSC 故障处理时, 通过扫描回滚管理段收集事务信息, 以达到缩减事务收集时长的目的。

此版本增加了一个建库参数 (PSEG_MGR_FLAG) 表示是否仅使用回滚管理段记录事务信息, 取值 0、1, 默认为 0。取值含义说明如下:

- 0: 除了使用回滚管理段记录事务信息外, 在事务的首个回滚页上也记录事务信息。
- 1: 仅使用回滚管理段记录事务信息, 事务的首个回滚页上不再记录。

老版本的库升级时, 一律采用 PSEG_MGR_FLAG=0 的方式进行升级, 老版本库在升级时会自动创建回滚管理段, 升级成功后, 不允许再使用老执行码启动升级后的库。

9.2.1 版本说明

具体的升级版本限制说明如下:

- 升级前的老版本库是 DM8, 版本号等于或者高于 V8.1.1.15

老版本库已支持升级后的日志格式, 本次升级仅需要升级回滚管理段。

- 升级前的老版本库是 DM8, 版本号低于 V8.1.1.15; 或者升级前的老版本库是 DM7
老版本库不支持升级后的日志格式, 本次升级需要先升级日志格式, 再升级回滚管理段。

对这个版本范围的老版本库, 升级前首先需要满足“[9.1 日志格式升级](#)”的各项升级条件, 然后还要满足回滚管理段的升级条件才能升级。

9.2.2 升级步骤

回滚管理段的升级必须满足以下两个条件:

1. 要求升级前按照顺序正常退出整个数据守护环境, 退出前主备库必须都是 Open 状态。

2. 如果有 DSC 集群, 则要求升级前不存在故障节点, 并且所有节点均正常退出。

如果老库已支持升级后的日志格式 (版本号等于或者高于 V8.1.1.15), 在满足这两个条件之后, 直接使用 V8.1.1.101 或者更高版本的执行码启动整个数据守护环境 (启动顺序没有严格要求, 主备库的 dmserver、dmwatcher、dmmonitor 全部正常启动即可), 主备库都能够正常 Open、守护进程都处于 Open 状态、备库归档状态有效则说明升级成功。

需要注意的是, 如果老库升级前已支持升级后的日志格式 (版本号等于或者高于 V8.1.1.15), 则本次回滚管理段升级前后的日志格式是兼容的, 因此不要求升级前备库和主库数据一致, 也就是不要求备库归档状态有效, 回滚管理段升级完成后, 主库仍然可以正常同步日志到备库。

如果老库还未支持升级后的日志格式，则需要同时满足日志格式和回滚管理段的升级条件，具体执行升级时，要根据老库当前的版本号，严格按照 [9.1 日志格式升级](#) 中的方式进行升级，只需要将其中的 V8.1.1.15 替换为 V8.1.1.101 即可，V8.1.1.101 可以同时完成日志格式和回滚管理段的升级。

9.3 数据页格式升级

DM 从 V8.1.1.171 版本开始将修改数据页的节点信息 (FRESH_EP) 持久化到了数据页上，相应的对数据页上的页类型字段长度进行了缩减，同时将库版本号 (DB_VERSION) 由 0x0007000B 升级到了 0x0007000C。

新增的 FRESH_EP 字段主要用在 DMDSC 集群中，通过持久化的方式能够准确获取到修改数据页的前序节点，从而能够构建出正确的日志包依赖关系，以准确检测到是否存在日志缺失，解决备库重演卡住或者重演超时主动关闭等一系列问题。

特别要说明的是，对 DMDSC 集群、DMDSC 集群的备库、DMDSC 集群还原恢复后的库，升级成功后不支持再回退到升级前的版本。

9.3.1 版本说明

具体的升级版本限制说明如下：

- 升级前的老版本库是 DM8，版本号等于或者高于 V8.1.1.101

老版本库已支持升级后的日志格式和回滚管理段，本次升级仅需要升级数据页格式。

- 升级前的老版本库是 DM8，版本号等于或者高于 V8.1.1.15

老版本库已支持升级后的日志格式，本次升级需要同时完成回滚管理段和数据页格式两项升级内容，因此升级前需要同时满足“9.2 回滚管理段升级”和“9.3 数据页格式升级”的各项升级条件才能够升级成功。

- 升级前的老版本库是 DM8，版本号低于 V8.1.1.15；或者升级前的老版本库是 DM7

老版本库不支持升级后的日志格式，本次升级需要完成日志格式、回滚管理段和数据页格式三项升级内容。因此对这个版本范围的老版本库，升级前需要同时满足“[9.1 日志格式升级](#)”和“9.2 回滚管理段升级”以及“9.3 数据页格式升级”的各项升级条件才能够升级成功。

9.3.2 升级步骤

本小节内容仅适用于老库版本号等于或者高于 v8.1.1.101 的情况，如果是更低版本的老库升级，则需要同时参考 9.1 和 9.2 小节内容。

数据页格式的升级必须满足以下条件：

1. 要求升级前按照正常的退出顺序退出整个数据守护环境，退出前主备库必须都是 Open 状态。
2. 如果升级前的数据守护环境中没有 DMDSC 集群，则升级前不要求主备库数据完全一致（单节点环境下，日志包不存在依赖关系，不会影响到归档连续性校验），但是要求主备库都必须正常退出。
3. 如果升级前的数据守护环境中包含有 DMDSC 集群，则升级前要求主备库数据必须完全一致（避免升级完成后使用到升级前的归档日志，无法准确判断日志依赖关系，引发备库重演卡住等问题），同时要求 DMDSC 集群升级前不存在故障节点，并且所有节点均正常退出。

满足以上条件后，直接使用 v8.1.1.171 或者更高版本的执行码启动整个数据守护环境（主备库启动顺序没有严格要求），主备库都能够正常 Open、守护进程都处于 Open 状态、备库归档状态有效则说明升级成功。

注意，本次升级完成后，对 DMDSC 集群、DMDSC 集群的备库、DMDSC 集群还原恢复后的库，不允许再回退到升级前的版本。

9.4 HUGE 表空间升级

9.4.1 版本说明

从 v8.1.2.101 版本开始，DM 对 HUGE 表空间进行了升级改造。本节中低版本是指 v8.1.2.101 之前的库，高版本是指 v8.1.2.101 或更高的版本。

不使用 HUGE 表的用户可以不用升级 HUGE 表空间。需要特别注意的是，升级后将不支持继续使用低版本库下所创建的 HUGE 表。若用户库中已有正在使用的 HUGE 表，建议谨慎升级。自定义混合表空间由用户可根据自己需要创建。本章升级对象为 MAIN 系统混合表空间。

高版本与低版本之间的差异如下：

- 用户自定义混合表空间

在低版本中，表空间分为普通表空间和 HUGE 表空间。

从高版本中，对 HUGE 表空间进行了升级改造，废除了 HUGE 表空间的概念，取而代之的是混合表空间的概念。在高版本中，表空间分为普通表空间和混合表空间。

使用<HUGE 路径子句>创建的表空间为混合表空间，未使用<HUGE 路径子句>创建的表空间即为普通表空间。普通表空间只能存储普通表（非 HUGE 表）；而混合表空间既可以存储普通表又可以存储 HUGE 表。HUGE 数据文件存储在<HUGE 路径子句>指定的路径中，普通（非 HUGE）数据文件存储在<数据文件子句>指定的路径中。HUGE 表对应的辅助表为普通表，因此，HUGE 辅助表数据存在<数据文件子句>指定的路径中。

若创建 HUGE 表使用自定义的混合表空间，则需要用户先创建一个混合表空间。自定义混合表空间创建方法参考《DM8_SQL 语言使用手册》CREATE TABLESPACE 语法。

例 创建 HUGE 表 T1，使用自定义的混合表空间 TS1。

```
CREATE TABLESPACE TS1 DATAFILE 'd:\TS1.dbf' SIZE 128 WITH HUGE PATH
'D:\TS1\HUGE1';

CREATE HUGE TABLE T1(C1 INT, C2 INT) STORAGE(ON TS1);
```

- 系统 MAIN 表空间

在低版本中，普通表用户默认表空间 MIAN 为一个普通表空间。HUGE 表默认表空间 HMAIN 为一个 HUGE 表空间。

在高版本中，取消了 HMAIN 表空间。普通表和 HUGE 表用户默认表空间统一为 MAIN 表空间，MAIN 为一个混合表空间。

例 创建 HUGE 表 T2，缺省使用系统默认的 MAIN 混合表空间。

```
CREATE HUGE TABLE T2(C1 INT, C2 INT) ;
```

9.4.2 升级步骤

数据守护环境下 HUGE 表空间升级分为三种情况。

- 现有的主备库升级

对于数据守护环境中，对现有的低版本主备库升级需分别单独进行，各节点的升级方法和单节点一样。

- 在升级后的单节点基础上搭建主备或增加备库

需先将低版本单节点库进行升级。为了达到主备库完全一致的效果，先对升级后的库进行备份，通过还原恢复的方式搭建另外一个库，另搭建后无需再升级。

- 在未升级的单节点基础上搭建主备或增加备库

未升级的低版本数据库（简称 A）中的 MAIN 表空间为普通表空间。通过高版本的库软件对未升级的库进行备份还原后，新库（简称 B）中 MAIN 表空间会被初始化为高版本的混合表空间。此时，AB 两个库中的 MAIN 表空间就存在不一致的情况，A 库需要升级。例如，如果将 MAIN 混合表空间所在的 B 库作为主库时，主库可以成功创建 HUGE 表语句，但是备库 A 处于低版本而无法完成，可能会导致宕机。

此时，需对低版本 A 库采用脱机方式进行升级。高版本 B 库不用升级。

温馨提醒：当数据守护集群中节点数很多，DBA 无法区分哪个节点为带升级的低版本时，可通过查看控制文件来解决。如果控制文件中 MAIN 表空间的描述信息中包含 HUGE PATH 内容即为高版本库，无需升级；如果无 HUGE PATH 内容即为低版本库，需要升级。

10 附录

10.1 系统视图

用于数据守护环境下的一些系统信息查询。

1. V\$UTSK_INFO

查询守护进程向服务器发送请求的执行情况。

此视图仅用于查看当前服务器的命令执行情况，在 CMD 字段值不为空时，说明是有效的命令信息；此时如果 CODE 字段值是 100，说明命令还在执行中；如果 CODE 字段值是 0，说明命令已经执行成功；如果 CODE 字段值小于 0，则说明命令执行失败。

其中，RUNNING/RECOVER_BREAK/AUTO_SWITCH 为系统全局信息，为保持兼容在此视图中仍然保留这三个字段，但字段值统一置为 NULL，没有实际意义，可再通过查询 V\$UTSK_SYS2 视图来查看系统全局信息。

序号	列	数据类型	说明
1	RUNNING	VARCHAR(64)	当前系统是否正在处理守护进程请求，TRUE 表示正在处理，FALSE 表示没有。兼容保留字段，升级后直接显示 NULL 值
2	DSEQ	INTEGER	守护进程发送的报文序号，此字段为兼容保留字段，没有实际意义，字段值直接显示为 0，真正的报文序号可查询 DSEQ2 字段值获取
3	CODE	INTEGER	DSEQ 对应的执行结果
4	SEND_LSN	BIGINT	主库当前已发送给备库的 LSN 值
5	RECOVER_BREAK	VARCHAR(64)	用于显示主库的 Recovery 流程是否被中断，TRUE 表示被中断，FALSE 表示没有。兼容保留字段，升级后直接显示 NULL 值
6	AUTO_SWITCH	VARCHAR(64)	守护系统配置的切换模式，TRUE 表示故障自动切换，FALSE 表示故障手动切换。兼容保留

			字段，升级后直接显示 NULL 值
7	NTH	INTEGER	命令所在的数组序号
8	CMD	VARCHAR(64)	待执行或已经执行完成的命令
9	INST_NAME	VARCHAR(64)	需要恢复的备库实例名，只在主库启动恢复时会设置
10	MAX_SEND_LSN	BIGINT	主库向备库发送的最大 LSN 值
11	DSEQ2	BIGINT	守护进程发送的报文序号

2. V\$UTSK_SYS2

显示服务器当前的全局信息。

序号	列	数据类型	说明
1	RUNNING	VARCHAR(64)	当前系统是否正在处理守护进程请求，TRUE 表示正在处理，FALSE 表示没有
2	BREAK_FLAG	VARCHAR(64)	守护进程命令是否被中断，取值：TRUE/FALSE
3	AUTO_SWITCH	VARCHAR(64)	守护系统配置的切换模式，TRUE 表示故障自动切换，FALSE 表示故障手动切换
4	MAX_WAIT_TIME	INTEGER	服务器等待守护进程启动的最大时间，取本地 dm.ini 中的 DW_MAX_SVR_WAIT_TIME 值
5	LAST_MSG_TIME	TIMESTAMP(0)	服务器最近一次收到控制守护进程的时间
6	DW_VERSION	VARCHAR(64)	数据守护版本号

3. V\$RECOVER_STATUS

该视图需要在主库上查询（DMDSC 主库需要在控制节点上查询），用于查询备库的恢复进度，如果已恢复完成，查询结果为空。

注意这里显示的是主库向备库发送日志的进度，由于备库重做日志也需要时间，在最后一批日志发送完成后，KBYTES_TO_RECOVER 为 0，RECOVER_PERCENT 为 100%，表示主库已经完成所有日志发送，需要等待备库将最后一批日志重做完成，此时主库的守护进程可能仍然处于 Recovery 状态，待备库重做完成后，主库的守护进程会自动切换 Open 状

态。

$$\text{RECOVER_PERCENT} = (\text{KBYTES_TOTAL} - \text{KBYTES_TO_RECOVER}) / \text{KBYTES_TOTAL}$$

序号	列	数据类型	说明
1	PRIMARY_NAME	VARCHAR(256)	主库实例名
2	STANDBY_NAME	VARCHAR(256)	当前正在恢复的备库实例名
3	PRIM_LSN	BIGINT	主库当前的 FLSN
4	SEND_LSN	BIGINT	主库已发给备库的最大 LSN
5	KBYTES_TOTAL	BIGINT	主库需要发送给备库的日志量，单位为字节
6	KBYTES_TO_RECOVER	BIGINT	主库剩余需要发送给备库的日志量，单位为字节。
7	FILES_TO_RECOVER	INTEGER	主库需要发给备库重做的总的日志文件个数
8	RECOVER_PERCENT	VARCHAR(64)	主库向备库已发送日志的比例

4. V\$KEEP_RLOG_PKG

该视图需要在备库上查询（DMDSC 备库需要在控制节点（重演节点）上查询），用于查询备库上的 KEEP_RLOG_PKG 信息，在备库归档状态有效的情况下可以查到内容。

读写分离集群下备库没有 KEEP_RLOG_PKG 机制，该视图查询结果为空。

序号	列	数据类型	说明
1	TYPE	VARCHAR(64)	主库的日志同步类型，REALTIME 类型
2	MIN_LSN	BIGINT	KEEP_RLOG_PKG 中的最小 LSN
3	MAX_LSN	BIGINT	KEEP_RLOG_PKG 中的最大 LSN
4	PTX_OFF	SMALLINT	KEEP_RLOG_PKG 中的 PTX 偏移
5	LOG_SIZE	INTEGER	KEEP_RLOG_PKG 中的有效日志长度，单位为字节
6	DSC_SEQNO	INTEGER	标识对应的主库实例节点的序号 DSC_SEQNO。如果主库是单机，则显示为 0。

7	DB_MAGIC	BIGINT	产生 KEEP_RLOG_PKG 包的源数据库的 DB_MAGIC
8	PKG_TYPE	INTEGER	KEEP_RLOG_PKG 包的类型
9	PKG_SEQNO	BIGINT	KEEP_RLOG_PKG 包的全局序号
10	PREV_LSN	BIGINT	KEEP_RLOG_PKG 包的前一个包的最大 LSN 值
11	PKG_LEN	BIGINT	KEEP_RLOG_PKG 包的长度，单位为字节

5. V\$RAPPLY_SYS

该视图需要在备库上查询（DMDSC 备库需要在控制节点（重演节点）上查询），用于查询备库重做日志时的一些系统信息。

序号	列	数据类型	说明
1	APPLYING	VARCHAR(64)	备库的重做线程是否正在处理任务，查询结果为 TRUE 或 FALSE
2	SSEQ	BIGINT	备库可重演到的最大包序号值
3	SLSN	BIGINT	备库可重演到的最大 LSN 值
4	KSEQ	BIGINT	非自动切换模式下，备库保持不重演的 RLOG_PKG 的序号。
5	KLSN	BIGINT	非自动切换模式下，备库保持不重演的 RLOG_PKG 的最大 LSN。
6	TASK_NUM	INTEGER	备库上待重做的任务数
7	PRIMARY_NAME	VARCHAR(128)	备库对应的主库实例名
8	HAS_KEEP_PKG	VARCHAR(64)	备库上是否有 KEEP_RLOG_PKG，查询结果为 TRUE 或 FALSE
9	TASK_MEM_USED	BIGINT	备库上重做日志堆积占用的内存大小（字节）

10	TASK_START_TIME	TIMESTAMP	当前正在重做日志 PKG 的开始时间
11	LAST_REDO_TIME	BIGINT	最近一次日志 PKG 重做的耗时（毫秒）
12	TASK_NUM_APPLIED	BIGINT	备库上已经重做的日志 PKG 总个数
13	APPLIED_TOTAL_TIME	BIGINT	备库上重做的日志 PKG 的总时间（毫秒）
14	SEQNO	INTEGER	标识对应的主库实例节点的序号 DSC_SEQNO。如果主库是单机，则显示为 0
15	RREC_TYPE	VARCHAR(64)	当前重演的物理记录类型。取值如下： RTYPE_FILE_EXTEND：文件扩展； RTYPE_FILE_CREATE：文件创建； RTYPE_NONE：没有文件相关动作

6. V\$RAPPLY_LOG_TASK

该视图需要在备库上查询（DMDSC 备库需要在控制节点（重演节点）上查询），用于查询备库当前重做任务的日志信息。

序号	列	数据类型	说明
1	TYPE	VARCHAR(64)	主库的日志同步类型， REALTIME/TIMELY/ASync
2	MIN_LSN	BIGINT	日志 PKG 的最小 LSN 值
3	MAX_LSN	BIGINT	日志 PKG 的最大 LSN 值
4	PTX_OFF	SMALLINT	日志 PKG 的 PTX 偏移
5	LOG_SIZE	INTEGER	日志 PKG 的有效长度，单位为字节
6	DSC_SEQNO	INTEGER	标识对应的主库实例节点的序号 DSC_SEQNO。如果主库是单机，则显示为 0
7	DB_MAGIC	BIGINT	产生日志 PKG 的源数据库的 DB_MAGIC
8	PKG_TYPE	INTEGER	日志 PKG 的类型
9	PKG_SEQNO	BIGINT	日志 PKG 的序号
10	PREV_LSN	BIGINT	前一个 PKG 的最大 LSN
11	PKG_LEN	BIGINT	日志 PKG 的长度，单位为字节

7. V\$ARCH_FILE

查询本地已经归档的日志信息。对 DMDSC 集群，除了显示本地归档外，也显示远程归档信息。

序号	列	数据类型	说明
1	DB_MAGIC	BIGINT	写入日志的数据库的 MAGIC 值
2	STATUS	VARCHAR(64)	归档日志状态
3	LEN	BIGINT	日志长度，单位为字节
4	FREE	BIGINT	写日志偏移位置
5	ARCH_LSN	BIGINT	归档文件起始 LSN，仅对归档文件有意义
6	CLSN	BIGINT	已归档的最大 LSN
7	ARCH_SEQ	INTEGER	归档文件起始包序号，仅对归档文件有意义
8	NEXT_SEQ	INTEGER	已归档的最大包序号。
9	CREAT_TIME	TIMESTAMP	归档文件创建时间
10	CLOSE_TIME	TIMESTAMP	归档文件关闭时间
11	PATH	VARCHAR(256)	归档文件所在路径，包括归档文件名
12	PMNT_MAGIC	BIGINT	数据库的永久 MAGIC
13	DSC_SEQNO	INTEGER	DSC 节点序号，单节点为 0
14	CRC_CHECK	INTEGER	是否进行 CRC 校验
15	LAST_PKG_OFF	BIGINT	最后一个有效的日志包在文件中的偏移位置
16	PREV_LSN	BIGINT	前一个归档文件已归档的最大 LSN
17	GLOBAL_NEXT_SEQ	BIGINT	已归档的最大包序号
18	SRC_DB_MAGIC	BIGINT	产生日志的源数据库的 MAGIC 值

8. V\$ARCH_STATUS

查询归档状态信息，归档状态是由主库记录和维护的，此视图只在主库上查询有效，备库上的查询结果没有实际意义。

序号	列	数据类型	说明
1	ARCH_TYPE	VARCHAR(256)	归档类型
2	ARCH_DEST	VARCHAR(256)	归档目标, 本地归档和 REMOTE 归档为归档路径, 其他类型为归档目标实例名。
3	ARCH_STATUS	VARCHAR(256)	归档状态, Valid 为有效状态, Invalid 为无效状态。
4	ARCH_SRC	VARCHAR(256)	对 REMOTE 归档为源实例名, 对其他归档类型为本地实例名

9. V\$MAL_LINK_STATUS

查询本地实例到远程实例的 MAL 链路连接状态。

序号	列	数据类型	说明
1	SRC_SITE	VARCHAR(256)	源站点实例名
2	DEST_SITE	VARCHAR(256)	目的站点实例名。
3	CTL_LINK_STATUS	VARCHAR(256)	控制链路连接状态, CONNECTED 表示连接已建立, DISCONNECT 表示连接断开。
4	DATA_LINK_STATUS	VARCHAR(256)	数据链路连接状态, CONNECTED 表示连接已建立, DISCONNECT 表示连接断开。

10. V\$DM_ARCH_INI

归档参数信息。

序号	列	数据类型	说明
1	ARCH_NAME	VARCHAR(128)	归档名称
2	ARCH_TYPE	VARCHAR(128)	归档类型
3	ARCH_DEST	VARCHAR(512)	对于 LOCAL 归档, 表示归档路径; 对于 REMOTE 归档, 表示本节点归档要发送到的实例名; 对于其余类型的归档, 表示归档目标实例名
4	ARCH_FILE_SIZE	INTEGER	归档文件大小

5	ARCH_SPACE_LIMIT	INTEGER	归档文件的磁盘空间限制，单位是 MB
6	ARCH_HANG_FLAG	INTEGER	如果本地归档时磁盘空间不够，是否让服务器挂起。无实际意义，对本地归档类型和远程归档类型显示为1，其他归档类型显示为NULL
7	ARCH_TIMER_NAME	VARCHAR (128)	对于异步归档，表示定时器名称；其余类型归档显示 NULL
8	ARCH_IS_VALID	CHAR(1)	归档状态，是否有效
9	ARCH_WAIT_APPLY	INTEGER	性能模式，是否等待重演完成，取值 0：高性能模式， 1：数据一致模式。本地归档取值 NULL
10	ARCH_INCOMING_PATH	VARCHAR(256)	对 REMOTE 归档，表示远程节点发送过来的归档在本地的保存目录；其余类型归档显示 NULL
11	ARCH_CURR_DEST	VARCHAR(256)	当前归档目标实例名，如果备库是 DMDSC 集群，则归档目标是备库控制节点，该字段表示当前被选定为归档目标的节点实例名。备库是单机的情况下，就是指的备库实例名。
12	ARCH_SEND_DELAY	INTEGER	源库到异步备库的归档延时发送时间，单位为分钟，取值范围（0~1440），默认为 0，表示不启用归档延时发送功能

11. V\$DM_MAL_INI

MAL 参数信息。

序号	列	数据类型	说明
1	MAL_NAME	VARCHAR (128)	MAL 名称
2	MAL_INST_NAME	VARCHAR (256)	实例名称
3	MAL_HOST	VARCHAR (30)	IP 地址
4	MAL_PORT	INTEGER	端口号
5	MAL_INST_HOST	VARCHAR (256)	对应实例 IP 地址
6	MAL_INST_PORT	INTEGER	对应实例的端口号

7	MAL_DW_PORT	INTEGER	MAL_INST_NAME实例的守护进程监听远程守护进程或监视器连接请求的端口
8	MAL_LINK_MAGIC	INTEGER	MAL链路网段标识
9	MAL_INST_DW_PORT	INTEGER	MAL_INST_NAME实例监听守护进程连接请求的端口

12. V\$DM_TIMER_INI

定时器参数信息。

序号	列	数据类型	说明
1	TIMER_NAME	VARCHAR(128)	定时器名称
2	TYPE	TINYINT	类型
3	FREQ_MONTH_WEEK_INTERVAL	INTEGER	月或周的间隔
4	FREQ_SUB_INTERVAL	INTEGER	间隔天数
5	FREQ_MINUTE_INTERVAL	INTEGER	分钟间隔
6	START_TIME	TIME	开始时间
7	END_TIME	TIME	结束时间
8	DURING_START_DATE	DATETIME	有效时间段的开始日期时间
9	DURING_END_DATE	DATETIME	有效时间段的结束日期时间
10	NO_END_DATE_FLAG	CHAR(1)	无结束日期标识
11	DESCRIBE	VARCHAR(256)	描述
12	IS_VALID	CHAR(1)	是否有效

13. V\$DMWATCHER

查询当前登录实例所对应的守护进程信息,注意一个守护进程可以同时守护多个组的实例,因此查询结果中部分字段(N_GROUP、SWITCH_COUNT)为守护进程的全局信息,并不是当前登录实例自身的守护信息。在 DMDSC 集群环境中,只显示控制守护进程的信息。

另外 MPP 主备环境下，全局登录方式返回的是所有 MPP 站点上查询返回的守护进程信息，可以根据 INST_NAME 实例名字段来区分。

序号	列	数据类型	说明
1	N_GROUP	INTEGER	守护进程全局信息，指实例所在的守护进程守护的组个数
2	GROUP_NAME	VARCHAR(128)	实例所在的守护进程组名
3	INST_NAME	VARCHAR(128)	实例名
4	DW_TYPE	VARCHAR(32)	实例所在守护进程组的守护类型
5	DW_MODE	VARCHAR(32)	实例所在守护进程组的守护模式
6	AUTO_RESTART	INTEGER	守护进程对本实例是否配置有自动重启， 1：自动重启 0：不自动重启
7	DW_STATUS	VARCHAR(64)	实例的守护进程状态
8	DW_SUB_STATUS	VARCHAR(64)	实例的守护进程子状态
9	LAST_MSG_TIME	DATETIME	实例最近一次收到守护进程消息的时间
10	SWITCH_COUNT	INTEGER	守护进程全局信息，指守护进程组内主库的变迁次数（包括 SWITCHOVER 主备库切换，TAKEOVER 手动 / 自动接管，TAKEOVER 强制接管导致的主库变迁操作）。
11	CTL_NUM	INTEGER	实例的守护进程收到的远程实例控制文件信息个数
12	INST_NUM	INTEGER	实例的守护进程收到的远程实

			例信息个数
13	MAX_CONN_NUM	INTEGER	实例的守护进程当前最大的 TCP 连接数

14. V\$MAL_SYS

MAL 系统信息视图。如果是数据守护环境，则只显示主库的 MAL 系统信息。

序号	列	数据类型	说明
1	SYS_STATUS	INTEGER	mal 系 统 状 态 : 0:OPEN,1:PRE_SHUTDOWN,2:SHUTDOWN
2	STMT_ID	INTEGER	mal系统当前stmtid
3	NEXT_MAL_ID	BIGINT	下一个MAL_ID
4	MAL_PORT	INTEGER	mal监听端口
5	N_SITE	INTEGER	mal配置的站点数目
6	MAL_NUM	INTEGER	mal系统邮箱数目
7	MAL_SEQ_NO	INTEGER	站点本身的mal序号
8	EMPTY_LET_NUM	INTEGER	空邮件数目
9	MAL_CHECK_INTERVAL	INTEGER	链路检测间隔
10	MAL_CONN_FAIL_INTERVAL	INTEGER	认定链路断开的時間间隔
11	MAL_COMPRESS_LEVEL	INTEGER	邮件压缩级别
12	MAL_BUF_SIZE	INTEGER	单个MAL缓存大小限制，以M为单位。当MAL 的缓存邮件超过此大小，会将邮件存储到文 件中。有效值范围(0~500000)，默认为100， 配置为0表示无限制
13	MAL_SYS_BUF_SIZE	INTEGER	MAL系统总内存大小限制，以M为单位。有效 值范围 (0~500000)，默认为0，表示无限 制
14	MAL_VPOOL_SIZE	INTEGER	MAL配置的总的POOL大小，以M为单位。有 效值范围 (1~500000)，默认为128

15	MAL_TEMP_PATH	VARCHAR(256)	指定临时文件的目录。当邮件使用的内存超过 MAL_BUF_SIZE 或者 MAL_SYS_BUF_SIZE时,将新产生的邮件保存到临时文件中。如果缺省,则新产生的邮件保存到temp.dbf文件中
----	---------------	--------------	---

15. V\$MAL_SITE_INFO

MAL站点信息视图, MPP模式下, 自动收集MPP各个站点的信息。

序号	列	数据类型	说明
1	SRC_SITE_SEQ	INTEGER	发送邮件的源站点序号
2	DEST_SITE_SEQ	INTEGER	目标站点序号
3	MAL_PORT_NUM	INTEGER	兼容老版本而保留的字段, 当前取值0和1。 0表示数据链路已断开; 1表示数据链路处于连接状态。
4	BUILD_TIME	BIGINT	下一个邮件发送序号
5	LBTAPFDS	BIGINT	收到的邮件中已处理的最后一个邮件的序号
6	CUR_LETTER_NUM	INTEGER	当前保存不连续邮件的个数
7	MAX_LETTER_NUM	INTEGER	目前为止不连续邮件的最大个数
8	TOTAL_LINK_NUM	INTEGER	已创建到目标站点的mal_link数
9	FREE_LINK_NUM	INTEGER	当前空闲的mal_link数
10	SEND_LETTER_NUM	INTEGER	当前发送的邮件数

16. V\$ARCH_SEND_INFO

此视图用于在主库上查询各备库的日志发送统计信息。

如果主库是DMDSC集群, 并且主库当前正在执行Recovery恢复动作, 则只有控制节点上会有最新的归档发送信息, 如果主库在向备库正常同步数据, 则凡是有日志生成并发送的节点上都可以查询到最新的归档发送信息。

序号	列	数据类型	说明
----	---	------	----

1	ARCH_DEST	VARCHAR(256)	归档目标实例名
2	ARCH_TYPE	INTEGER	归档类型
3	FOR_RECOVERY	CHAR	是否为Recovery状态下的日志同步（Y/N）
4	RECNT_SEND_CNT	INTEGER	最近累计日志发送的次数N。
5	TOTAL_SEND_CNT	BIGINT	累计日志发送次数
6	TOTAL_SEND_LEN	BIGINT	累计日志发送长度（字节）
7	TOTAL_PTX_CNT	BIGINT	累计发送的PTX个数
8	TOTAL_SEND_TIME	BIGINT	累计日志发送耗时（微秒）
9	MAX_SEND_TIME	BIGINT	最大的日志发送耗时（微秒）
10	MAX_END_TIME	TIMESTAMP	最大耗时发送的结束时间
11	MAX_PTX_CNT	INTEGER	最大耗时发送的PTX个数
12	MAX_SEND_LEN	INTEGER	最大耗时发送的日志长度（字节）
13	MAX_SEND_LSN	BIGINT	最大耗时发送的结束LSN
14	LAST_SEND_LEN	INTEGER	最近一次发送的日志长度（字节）
15	LAST_SEND_PTX	INTEGER	最近一次日志发送的PTX个数
16	LAST_SEND_LSN	BIGINT	最近一次日志发送的结束LSN
17	LAST_SEND_TIME	BIGINT	最近一次日志发送耗时（微秒）
18	LAST_START_TIME	TIMESTAMP	最近一次日志发送的起始时间
19	LAST_END_TIME	TIMESTAMP	最近一次日志发送的结束时间
20	RECNT_SEND_LEN	BIGINT	最近RECNT_SEND_CNT 次发送的日志长度（字节）
21	RECNT_SEND_PTX	BIGINT	最近RECNT_SEND_CNT 次发送的PTX个数
22	RECNT_SEND_TIME	BIGINT	最近RECNT_SEND_CNT 次发送的耗时（微秒）

17. V\$RAPPLY_STAT

此视图用于在备库上查询备库重演日志的统计信息。

如果备库是DMDSC集群，则需要在控制节点（重演节点）上查询。

序号	列	数据类型	说明
1	RECNT_APPLY_NUM	INTEGER	备库最近重演的日志个数N。

2	TSK_MEM_USED	BIGINT	备库上重做日志堆积占用的内存大小（字节）
3	TSK_START_TIME	TIMESTAMP	当前正在重做日志BUF的开始时间
4	TOTAL_RECVED_NUM	BIGINT	累计收到日志包个数
5	TOTAL_RECVED_LEN	BIGINT	累计收到日志量（字节）
6	TOTAL_RECVED_TIME	BIGINT	累计消息响应时间（微秒）
7	TOTAL_APPLY_NUM	BIGINT	累计重演日志个数
8	TOTAL_APPLY_LEN	BIGINT	累计重演日志量（字节）
9	TOTAL_APPLY_TIME	BIGINT	累计日志重演时间（微秒）
10	TOTAL_WAIT_TIME	BIGINT	累计任务等待时间（微秒）
11	MAX_RECVED_TIME	BIGINT	最大消息响应时间（微秒）
12	MAX_WAIT_TIME	BIGINT	最大任务等待时间（微秒）
13	MAX_APPLY_TIME	BIGINT	最大任务重演时间（微秒）
14	MAX_APPLY_LEN	BIGINT	和MAX_APPLY_TIME对应的重演日志长度（字节）
15	LAST_RECVED_LEN	BIGINT	最近一次收到的日志长度（字节）
16	LAST_RECVED_TIME	BIGINT	最近一次响应时间（微秒）
17	RECNT_RECVED_LEN	BIGINT	最近RECNT_APPLY_NUM 次收到的日志长度（字节）
18	RECNT_RECVED_TIME	BIGINT	最近RECNT_APPLY_NUM 次响应时间（微秒）
19	LAST_WAIT_TIME	BIGINT	最近一次等待时间（微秒）
20	RECNT_WAIT_TIME	BIGINT	最近RECNT_APPLY_NUM 次等待时间（微秒）
21	LAST_APPLY_LEN	BIGINT	最后一次日志重演长度（字节）
22	LAST_APPLY_TIME	BIGINT	最近一次日志重演时间（微秒）
23	RECNT_APPLY_LEN	BIGINT	最近RECNT_APPLY_NUM 次日志重演长度（字节）
24	RECNT_APPLY_TIME	BIGINT	最近RECNT_APPLY_NUM 次日志重演时间（微秒）
25	SEQNO	INTEGER	标识对应的主库实例节点的序号DSC_SEQNO。如果主库是单机，则显示为0
26	LAST_CMT_TIME	DATETIME(6)	备库最新解析的主库最新事务提交记录对应的时间戳
27	APPLY_CMT_TIME	DATETIME(6)	备库最新重演的主库最新事务提交记录对应的时间戳

18. V\$RAPPLY_LSN_INFO

查询备库的重演信息，如果备库是DMDSC集群，则需要在控制节点（重演节点）上查询。如果在主库上查询此视图，则查到的是主库曾经作为备库时的历史重演信息。其中后面四个CKPT_XX字段，是在备库刷检查点时才调整，前面四个是随备库重演而动态调整的。

序号	列	数据类型	说明
1	P_DB_MAGIC	BIGINT	主库DB_MAGIC
2	N_EP	BIGINT	主库节点数
3	PKG_SEQ_ARR	VARCHAR(2048)	备库对应主库各节点已经写入到联机日志文件的原始包序号数组。
4	APPLY_LSN_ARR	VARCHAR(2048)	备库对应主库各节点已经写入到联机日志文件的原始LSN数组。
5	CKPT_P_DB_MAGIC	BIGINT	此字段无实际含义，是保留字段，和P_DB_MAGIC字段值相等。
6	CKPT_N_EP	INTEGER	此字段无实际含义，是保留字段，和N_EP字段值相等。
7	CKPT_PKG_SEQ_ARR	VARCHAR(2048)	此字段无实际含义，是保留字段，和PKG_SEQ_ARR字段值相等。
8	CKPT_APPLY_LSN_ARR	VARCHAR(2048)	此字段无实际含义，是保留字段，和APPLY_LSN_ARR字段值相等。

19. V\$RLOG_PKG

显示日志包信息。通过该视图可以查询日志系统中当前日志包的使用情况，如包的长度、最大LSN、最小LSN等。通过该视图还可以查询当前实例日志系统中等待刷盘的链表上的日志包信息。

序号	列	数据类型	说明
1	PKG_LEN	BIGINT	日志包的长度，单位为字节
2	DATA_OFF	BIGINT	日志包中有效数据长度
3	ENC_LEN	BIGINT	日志包加密后数据长度
4	CMPR_LEN	BIGINT	日志包压缩后数据长度
5	HDR_SIZE	INTEGER	日志包包头长度
6	N_MAGIC	BIGINT	日志包固定魔数

7	DB_MAGIC	BIGINT	数据库魔数
8	N_DSC	INTEGER	DSC集群节点数
9	DSC_SEQNO	INTEGER	DSC集群节点号
10	FROM_PRIMARY	INTEGER	是否为主库生成的日志
11	PKG_TYPE	INTEGER	日志包包类型
12	ENC_CMPR	INTEGER	加密压缩标记，0表示不加密不压缩，1表示只加密，2表示只压缩，3表示加密压缩
13	L_SEQNO	BIGINT	本地包序号
14	G_SEQNO	BIGINT	全局包序号
15	MIN_LSN	BIGINT	包内日志最小LSN
16	MAX_LSN	BIGINT	包内日志最大LSN
17	PREV_LSN	BIGINT	前一个包的日志最大LSN
18	N_FIXED	INTEGER	包的日志填充次数
19	N_FILLED	INTEGER	包的填充完成的次数
20	PKG_STAT	INTEGER	日志包状态，0表示正在填充的串行包，1表示正在填充的并行包，2表示已填充完成待刷盘的日志包

20. V\$RLOG_PKG_STAT

显示当前实例日志系统中日志包使用的统计信息。

序号	列	数据类型	说明
1	N_FLUSH_TO	BIGINT	请求刷盘的次数
2	N_FLUSH_WAIT	BIGINT	刷盘等待的次数
3	N_PUSHED	BIGINT	拷贝到 flush_lst 次数
4	N_COPY_WAIT	BIGINT	拷贝到 flush_lst 过程中产生等待的次数
5	N_FOR_CKPT	BIGINT	检查点类型的日志包个数
6	N_DISCARDED	BIGINT	日志刷盘线程中丢弃的日志包个数
7	N_FLUSHED	BIGINT	刷盘日志包个数
8	BYTES_FLUSHED	BIGINT	刷盘日志包长度总和

9	PKG_AVG_BYTES	INTEGER	刷盘日志包平均长度
10	NEXT_SEQNO	BIGINT	下一个日志包序号
11	N_ALLOC_WAIT	BIGINT	分配日志包过程中产生等待的次数
12	GLOBAL_NEXT_SEQNO	BIGINT	下一个全局日志包序号
13	PKG_AVG_PLI	INTEGER	日志包的平均并行度
14	N_FLUSH_LST	INTEGER	待刷盘的日志包个数

21. V\$RAPPLY_PARALLEL_INFO

查询备库的并行重演信息，如果备库是 DMDSC 集群，则需要在控制节点（重演节点）上查询。

序号	列	数据类型	说明
1	DSC_SEQNO	INTEGER	主库节点序号，如果主库是单节点，则序号为 0
2	APPLY_PKG_SEQ	BIGINT	备库对应主库各节点已经写入到联机日志文件的原始日志包序号
3	APPLY_LSN	BIGINT	备库对应主库各节点已经写入到联机日志文件的原始 LSN 值，和 APPLY_PKG_SEQ 相对应
4	REDOS_PARALLEL_NUM	INTEGER	备库并行重演线程数
5	REDO_LSN_ARR	VARCHAR(2048)	备库各路并行重演线程已重演到的 LSN 值
6	RPKG_SEQ	BIGINT	备库已重演到的日志包序号
7	RPKG_LSN	BIGINT	备库已重演到的 LSN 值，和 RPKG_SEQ 相对应

10.2 监视器接口

10.2.1 DLL 依赖

监视器 DLL 依赖了其他的一些动态链接库，编程时需要在加载动态链接库时指定 bin 目录。

10.2.2 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。对于启

动/关闭守护进程监控和实例的相关接口，返回值为100表示执行前状态已经和预期状态一致，接口实际并未执行。

相关的错误码说明请参考[10.2.8 错误码汇编](#)，也可借助dwmon_get_error_msg_by_code接口获取错误码对应的错误描述信息。

10.2.3 接口调整说明（V4.0）

数据守护V4.0版本对监视器接口变动较大，删除、新增、调整了部分接口，这里对涉及到的接口进行列举说明，具体的接口用法请参考[10.2.4 C接口说明](#)和[10.2.5 JNI接口说明](#)小节中的详细说明。

1. 删除的接口列表

1) dwmon_get_nth_watch_ctl_item

不再支持。

2. 调整过接口定义的列表

1) dwmon_get_split_sub_group_inst

更名为dwmon_get_split_sub_group_database

2) dwmon_get_nth_instance_info

更名为dwmon_get_nth_database_info，接口参数有调整。

3) dwmon_get_instance_info_by_name

更名为dwmon_get_database_info_by_name，接口参数有调整。

4) dwmon_get_instance_ep_info

更名为dwmon_get_database_info，接口参数有调整。

5) dwmon_get_nth_ep_info

接口参数有调整。

6) dwmon_get_ep_info_by_name

接口参数有调整。

7) dwmon_get_inst_arch_info

更名为dwmon_get_database_arch_info

8) dwmon_get_watch_ctl_info

接口参数有调整。

9) dwmon_get_monitor_info_with_version

更名为dwmon_get_monitor_info

10) dwmon_set_inst_recover_time

更名为dwmon_set_database_recover_time。

11) dwmon_set_inst_arch_invalid

更名为dwmon_set_database_arch_invalid。

12) dwmon_stop_instance

更名为dwmon_stop_database。

13) dwmon_startup_instance

更名为dwmon_startup_database。

14) dwmon_kill_instance

更名为dwmon_kill_database。

15) dwmon_open_instance

更名为dwmon_open_database。

16) dwmon_detach_instance

更名为dwmon_detach_database。

17) dwmon_attach_instance

更名为dwmon_attach_database。

18) dwmon_get_source_instance_name

更名为dwmon_get_source_database_name。

19) dwmon_get_ep_nth_apply_info

更名为dwmon_get_ep_nth_apply_stat，接口参数有调整。

20) dwmon_get_ep_nth_apply_lsn_info

更名为dwmon_get_ep_nth_apply_info。

21) dwmon_clear_inst_arch_send_info

更名为dwmon_clear_database_arch_send_info。

22) dwmon_clear_inst_rapply_info

更名为dwmon_clear_database_rapply_stat。

23) dwmon_clear_group_rapply_info

更名为dwmon_clear_group_rapply_stat。

对有参数调整的接口，新的参数列表和用法请参考[10.2.4 C接口说明](#)和[10.2.5 JNI接口说明](#)小节中的详细说明。

3. 新增的接口列表

- 1) `dwmon_get_database_open_history`
- 2) `dwmon_get_nth_open_history`
- 3) `dwmon_startup_database_watch`
- 4) `dwmon_stop_database_watch`
- 5) `dwmon_get_ep_nth_apply_with_parallel_info`

以上新增接口的定义和用法请参考[10.2.4 C接口说明](#)和[10.2.5 JNI接口说明](#)小节中的详细说明。

10.2.4 C 接口说明

1. `dwmon_alloc_handle`

函数原型:

```
DWMON_RETURN  
  
dwmon_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. dwmon_get_error_msg_by_code

函数原型:

DWMON_RETURN

```
dwmon_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:

mhdle: 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 mhdle 是无效的，否则要求 mhdle 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，-17 表示没有找到 code 对应的错误信息。

3. dwmon_init

函数原型:

DWMON_RETURN

```
dwmon_init(  

```

```

    mhandle      mhdle,

    mschar*      ini_path,

    mschar*      log_path

);

```

功能说明：

初始化监视器环境。

参数说明：

mhdle：输入参数，监视器操作句柄。

ini_path：输入参数，指定 dmmonitor.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 dmmonitor.ini 中配置的 MON_LOG_PATH 作为日志文件路径，如果 dmmonitor.ini 中没有配置 MON_LOG_PATH，则将 dmmonitor.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，-15 表示监视器尝试建立到所有守护进程的连接失败，可能守护进程都未启动，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 dwmon_deinit 接口），再释放操作句柄（调用 dwmon_free_handle 接口）。

4. dwmon_msg_event_wait**函数原型：**

```

void

dwmon_msg_event_wait(

    mhandle      mhdle

);

```

功能说明：

等待消息事件。监视器检测到守护进程或实例状态发生变化，或者接口命令执行过程中，产生中间输出消息时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 dwmon_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. dwmon_get_exec_msg

函数原型：

DWMON_RETURN

```
dwmon_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 dwmon_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. dwmon_get_msg_exit_flag

函数原型：

```
mbool  
  
dwmon_get_msg_exit_flag(  
  
    mhandle          mhdle  
  
);
```

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。

返回值：

1：允许线程退出。

0：不允许线程退出，需要继续等待获取输出消息。

7. dwmon_set_msg_exit_event

函数原型：

```
void  
  
dwmon_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 `dwmon_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：

`mhandle`：输入参数，监视器操作句柄。

返回值：

无

8. dwmon_msg_event_deinit

函数原型：

```
void  
  
dwmon_msg_event_deinit(  
  
    mhandle      mhandle  
  
);
```

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 `dwmon_msg_event_wait`、`dwmon_get_exec_msg`、`dwmon_get_msg_exit_flag`、`dwmon_set_msg_exit_event` 和 `dwmon_msg_event_deinit`，这几个接口需要配合使用。

参数说明：

`mhandle`：输入参数，监视器操作句柄。

返回值：

无

9. dwmon_get_version

函数原型：

```
DWMON_RETURN
```

```
dwmon_get_version(  
  
    mhandle          mhdle,  
  
    mschar*          version  
  
);
```

功能说明:

获取当前的监视器版本信息。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

version: 输出参数, 输出当前的监视器版本信息, version 的缓存长度不能小于 129, 避免溢出。

返回值:

0: 执行成功。

<0: 执行失败。

10. dwmon_get_mpp_flag

函数原型:

```
DWMON_RETURN  
  
dwmon_get_mpp_flag (  
  
    mhandle          mhdle,  
  
    mbool*           mpp_flag  
  
);
```

功能说明:

获取 MPP 主备环境标记, 用于判断当前是否为 MPP 主备环境。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

mpp_flag: 输出参数, MPP 主备环境下输出为 TRUE, 否则为 FALSE。

返回值:

0: 执行成功。

<0: 执行失败。

11. dwmon_get_n_group

函数原型:

DWMON_RETURN

```
dwmon_get_n_group(  
    mhandle          mhdle,  
    muint4*          n_group,  
    mschar**         group_name_arr  
);
```

功能说明:

获取当前监控的组个数和组名称。

只有 MPP 主备环境下允许配置多组。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

n_group: 输入输出参数, 监视器配置的监控组个数。输入参数: 指定可获取的最大组个数, 输出参数: 实际获取的组个数, 建议输入值不小于 16, 避免取不到完整信息。

group_name_arr: 输出参数, 输出监视器配置的监控组名, 参数是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。

12. dwmon_get_group_info

函数原型:

DWMON_RETURN

```
dwmon_get_group_info(  
    mhandle          mhdle,  
    mschar*          group_name,
```

```

    muint4*      oguid,

    muint4*      n_dw,

    mschar**     dw_ip_arr,

    msint2*      port_arr,

    mschar*      dw_type,

    mschar*      dw_mode,

    mschar*      primary_name,

    muint4*      n_split,

    mschar**     split_iname_arr,

    muint4*      n_db,

    mschar**     db_name_arr,

    muint4*      n_local,

    mschar**     local_name_arr

);

```

功能说明:

获取指定组的守护进程组信息。

参数说明:

`mhandle`: 输入参数, 监视器操作句柄。

`group_name`: 输入参数, 指定要获取信息的组名, 只有一组时允许不指定组名。

`oguid`: 输出参数, 输出指定守护进程组的 OGUID 值。

`n_dw`: 输入输出参数, 指定组配置的控制守护进程个数。输入参数: 可获取的最大守护进程个数, 输出参数: 实际获取的守护进程个数, 建议输入值不小于 25, 避免取不到完整信息。

`dw_ip_arr`: 输出参数, 输出指定组配置的控制守护进程 IP, 参数是字符串指针数组类型, 数组长度是 `n_dw` 的输入值, 数组中每个指针元素指向的缓存长度不能小于 65, 避免长度溢出。

`port_arr`: 输出参数, 输出指定组配置的控制守护进程端口, 参数是 `msint2` 数组类型, 数组长度是 `n_dw` 的输入值。

`dw_type`: 输出参数, 输出守护进程类型, 注意缓存长度不能小于 65, 避免长度溢出。

`dw_mode`: 输出参数, 输出守护进程切换模式, 注意缓存长度不能小于 65, 避免溢出。

`primary_name`: 输出参数, 输出当前的主库实例名, 如果组中有多个 Primary 实例, 则返回处于 Open 状态的 Primary 实例, 如果都是 Mount 状态, 则返回 LSN 最大的 Primary 实例, 如果有多个 Open 状态的 Primary 实例, 则接口会报错返回。注意缓存长度不能小于 65, 避免溢出。

`n_split`: 输入输出参数, 当前的子组个数, 如果组中有发生分裂, 每个分裂库是一个子组, 剩余没有分裂的正常库也是一个子组, 如果组中没有分裂情况, 则子组个数就是整个的组个数, 输出值为 1。输入参数: 可获取的最大子组个数, 输出参数: 实际获取的子组个数, 建议输入值不小于 9, 避免取不到完整信息。

`split_iname_arr`: 输出参数, 输出子组中其他库可加入的库名称, 参数是字符串指针数组类型, 数组长度是 `n_split` 的输入值, 数组中每个指针元素指向的缓存长度不能小于 65, 避免溢出。

`n_db`: 输入输出参数, 配置为 Global 守护类型的库个数。输入参数: 可获取的最大库个数, 输出参数: 实际获取的库个数, 建议输入值不小于 9, 避免取不到完整信息。

`db_name_arr`: 输出参数, 输出配置为 Global 守护类型的库名, 参数是字符串指针数组类型, 数组长度是 `n_db` 的输入值, 数组中每个指针元素指向的缓存长度不能小于 65, 避免溢出。

`n_local`: 输入输出参数, 配置为 local 守护类型的库个数。输入参数: 可获取的最大库个数, 输出参数: 实际获取的库个数, 建议输入值不小于 16, 避免取不到完整信息。

`local_name_arr`: 输出参数, 输出配置为 local 守护类型的库名, 参数是字符串指针数组类型, 数组长度是 `n_local` 的输入值, 数组中每个指针元素指向的缓存长度不能小于 65, 避免溢出。

返回值:

0: 执行成功。

<0: 执行失败。

13. `dwmon_get_split_sub_group_database`

函数原型:

DWMON_RETURN

```

dwmon_get_split_sub_group_database(

    mhandle          mhdle,

    mschar*          group_name,

    mschar*          db_name,

    muint4*          n_db,

    mschar**         db_name_arr

);

```

功能说明：

获取可加入指定库的库名信息，可以和 dwmon_get_group_info 接口配合使用，其中 db_name 是 dwmon_get_group_info 中 split_iname_arr 数组取出的某个库名。

参数说明：

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，指定库所在的组名，只有一组时可以不指定。

db_name：输入参数，指定要获取可加入信息的库名，此参数必须指定。

n_db：输入输出参数，可加入指定库的库个数。输入参数：指定可获取的最大库个数，输出参数：实际获取的库个数，建议输入值不小于 8，避免取不到完整信息。

db_name_arr：输出参数，输出可加入指定库的库名，参数是字符串指针数组类型，数组长度是 n_db 的输入值，数组中每个指针元素指向的缓存长度不能小于 65，避免溢出。

返回值：

0：执行成功。

<0：执行失败。

14. dwmon_get_nth_database_info

函数原型：

```

DWMON_RETURN

dwmon_get_nth_database_info(

    mhandle          mhdle,

    mschar*          group_name,

    muint4           nth,

```

```

mschar*      dw_time,
mschar*      dw_ip,
msint2*      dw_port,
mschar*      dw_type,
mschar*      dw_mode,
muint4*      dw_err_time,
mschar*      dw_sta,
mschar*      dw_sub_sta,
mschar*      dw_ctl_sta,
mschar*      db_name,
mschar*      db_ok_str,
mschar*      svr_mode,
mschar*      svr_status,
mschar*      arch_type,
mschar*      arch_status,
muint4*      pmnt_magic,
muint4*      db_magic,
msint2*      n_ep,
msint2*      ep_seqno_arr,
mschar**     ep_name_arr,
msint2*      dsc_ctl_node,
mschar*      dsc_status,
msint2*      n_ok_ep,
msint2*      ok_ep_arr,
msint2*      n_break_ep,
msint2*      break_ep_arr
);

```

功能说明:

获取指定组中第 nth 个库的全局信息, 这里的 nth 是根据 dmmonitor.ini 指定组中

所配置的守护进程先后顺序来排序的，每个组的 nth 都是从 0 开始，依次递增。

参数说明：



注意：

对字符串类型的输出参数，指向的缓存长度不能小于 65，避免溢出。

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，指定的守护进程组名，只有一组时可以不指定。

nth：输入参数，输入要获取的数据库实例下标。

dw_time：输出参数，输出控制守护进程本地的当前时间。

dw_ip：输出参数，输出控制守护进程配置的 IP 地址。

dw_port：输出参数，输出控制守护进程配置的端口。

dw_type：输出参数，输出配置的守护类型。

dw_mode：输出参数，输出配置的切换模式。

dw_err_time：输出参数，输出配置的守护进程故障认定时间。

dw_sta：输出参数，输出控制守护进程状态。

dw_sub_sta：输出参数，输出控制守护进程子状态。

dw_ctl_sta：输出参数，输出守护进程控制文件状态。

db_name：输出参数，输出守护的数据库名称。

db_ok_str：输出参数，输出守护进程认定的库状态 (OK/Error)。

svr_mode：输出参数，输出库模式 (Normal/Primary/Standby)。

svr_status：输出参数，输出库状态 (Startup/After Redo/Mount/Open/Suspend/Shutdown)。

arch_type：输出参数，输出库上配置的归档类型 (REALTIME/TIMELY)

arch_status：输出参数，指定待获取的库为实时备库或即时备库时，输出组中主库到此备库的归档状态，如果是异步备库，则输出为“NULL”，如果是主库，则输出“VALID”。

pmnt_magic：输出参数，输出数据库永久魔数。

db_magic：输出参数，输出数据库魔数。

n_ep：输入输出参数，指定库中包含的节点实例个数。输入参数：可获取的最大实例个数，输出参数：实际获取到的实例个数，建议输入值不小于 16，避免获取不到完整信息。

`ep_seqno_arr`: 输出参数, 输出指定库中所有实例的 `seqno` 数组, 参数是 `msint2` 数组类型, 数组长度是 `n_ep` 的输入值。

`ep_name_arr`: 输出参数, 输出指定库中所有实例的实例名数组, 参数是字符串指针数组类型, 数组长度是 `n_ep` 的输入值, 数组中每个指针元素指向的缓存长度不能小于 65, 避免长度溢出。

`dsc_ctl_node`: 输出参数, 输出指定库中控制节点的 `seqno`, 对单节点库, 输出值为 0。

`dsc_status`: 输出参数, 输出指定库的状态。

`n_ok_ep`: 输入输出参数, 指定库中包括的 `ok` 节点实例个数。输入参数: 可获取的最大实例个数, 输出参数: 实际获取到的实例个数, 建议输入值不小于 16, 避免获取不到完整信息。对单节点库, 输出值为 1。

`ok_ep_arr`: 输出参数, 输出指定库中所有 `ok` 节点实例的 `seqno` 数组, 参数是 `msint2` 数组类型, 数组长度是 `n_ok_ep` 的输入值。

`n_break_ep`: 输入输出参数, 指定库中当前正在故障处理的节点数。输入参数: 可获取的最大实例个数, 输出参数: 实际获取到的实例个数, 建议输入值不小于 16, 避免获取不到完整信息。

`break_ep_arr`: 输出参数, 输出指定库中当前正在故障处理的节点 `seqno` 数组, 参数是 `msint2` 数组类型, 数组长度是 `n_break_ep` 的输入值。

返回值:

0: 执行成功。

<0: 执行失败。

15. `dwmon_get_database_info_by_name`

函数原型:

DWMON_RETURN

`dwmon_get_database_info_by_name(`

<code>mhandle</code>	<code>mhdle,</code>
<code>mschar*</code>	<code>group_name,</code>
<code>mschar*</code>	<code>db_name,</code>

mschar*	dw_time,
mschar*	dw_ip,
msint2*	dw_port,
mschar*	dw_type,
mschar*	dw_mode,
muint4*	dw_err_time,
mschar*	dw_sta,
mschar*	dw_sub_sta,
mschar*	dw_ctl_sta,
mschar*	db_ok_str,
mschar*	svr_mode,
mschar*	svr_status,
mschar*	arch_type,
mschar*	arch_status,
muint4*	pmnt_magic,
muint4*	db_magic,
msint2*	n_ep,
msint2*	ep_seqno_arr,
mschar**	ep_name_arr,
msint2*	dsc_ctl_node,
mschar*	dsc_status,
msint2*	n_ok_ep,
msint2*	ok_ep_arr,
msint2*	n_break_ep,
msint2*	break_ep_arr

);

功能说明:

根据指定的数据库名称获取库的全局信息。

参数说明:

对字符串类型的输出参数，指向的缓存长度不能小于 65，避免溢出。

mhdle: 输入参数，监视器操作句柄。

group_name: 输入参数，指定的守护进程组名，只有一组时可以不指定。

db_name: 输入参数，指定要获取信息的数据库名称。

dw_time: 输出参数，输出控制守护进程本地的当前时间。

dw_ip: 输出参数，输出控制守护进程配置的 IP 地址。

dw_port: 输出参数，输出控制守护进程配置的端口。

dw_type: 输出参数，输出配置的守护类型。

dw_mode: 输出参数，输出配置的切换模式。

dw_err_time: 输出参数，输出配置的守护进程故障认定时间。

dw_sta: 输出参数，输出控制守护进程状态。

dw_sub_sta: 输出参数，输出控制守护进程子状态。

dw_ctl_sta: 输出参数，输出守护进程控制文件状态。

inst_ok_str: 输出参数，输出守护进程认定的库状态 (OK/Error)。

svr_mode: 输出参数，输出库模式 (NORMAL/Primary/Standby)。

svr_status: 输出参数，输出库状态 (Startup/AFTER Redo/Mount/Open/SUSPEND/Shutdown)。

arch_type: 输出参数，输出库上配置的归档类型 (REALTIME/TIMELY)

arch_status: 输出参数，指定待获取的库为实时备库或即时备库时，输出组中主库到此备库的归档状态，如果是异步备库，则输出为“NULL”，如果是主库，则输出“VALID”。

pmnt_magic: 输出参数，输出数据库永久魔数。

db_magic: 输出参数，输出数据库魔数。

n_ep: 输入输出参数，指定库中包含的节点实例个数。输入参数：可获取的最大实例个数，输出参数：实际获取到的实例个数，建议输入值不小于 16，避免获取不到完整信息。

ep_seqno_arr: 输出参数，输出指定库中所有实例的 seqno 数组，参数是 msint2 数组类型，数组长度是 n_ep 的输入值。

ep_name_arr: 输出参数，输出指定库中所有实例的实例名数组，参数是字符串指针数组类型，数组长度是 n_ep 的输入值，数组中每个指针元素指向的缓存长度不能小于 65，避免长度溢出。

dsc_ctl_node: 输出参数, 输出指定库中控制节点的 seqno, 对单节点库, 输出值为 0。

dsc_status: 输出参数, 输出指定库的状态。

n_ok_ep: 输入输出参数, 指定库中包括的 ok 节点实例个数。输入参数: 可获取的最大实例个数, 输出参数: 实际获取到的实例个数, 建议输入值不小于 16, 避免获取不到完整信息。对单节点库, 输出值为 1。

ok_ep_arr: 输出参数, 输出指定库中所有 ok 节点实例的 seqno 数组, 参数是 msint2 数组类型, 数组长度是 n_ok_ep 的输入值。

n_break_ep: 输入输出参数, 指定库中当前正在故障处理的节点数。输入参数: 可获取的最大实例个数, 输出参数: 实际获取到的实例个数, 建议输入值不小于 16, 避免获取不到完整信息。

break_ep_arr: 输出参数, 输出指定库中当前正在故障处理的节点 seqno 数组, 参数是 msint2 数组类型, 数组长度是 n_break_ep 的输入值。

返回值:

0: 执行成功。

<0: 执行失败。

16. dwmon_get_database_info

函数原型:

DWMON_RETURN

```
dwmon_get_database_info(  
    mhandle          mhdle,  
    mschar*          group_name,  
    mschar*          db_name,  
    msint2*          n_ep,  
    msint2*          ep_seqno_arr,  
    mschar**         ep_name_arr,  
    msint2*          dsc_ctl_node,  
    mschar*          dsc_status,
```

```

msint2*      n_ok_ep,

msint2*      ok_ep_arr

msint2*      n_break_ep,

msint2*      break_ep_arr

);

```

功能说明：

获取指定库上的实例信息。

参数说明：

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，输入指定库所在的守护进程组名，如果只有一组可以不指定。

db_name：输入参数，指定要获取实例信息的数据库名称。

n_ep：输入输出参数，指定库中包含的节点实例个数。输入参数：可获取的最大实例个数，输出参数：实际获取到的实例个数，建议输入值不小于 16，避免获取不到完整信息。

ep_seqno_arr：输出参数，输出指定库中所有实例的 seqno 数组，参数是 msint2 数组类型，数组长度是 n_ep 的输入值。

ep_name_arr：输出参数，输出指定库中所有实例的实例名数组，参数是字符串指针数组类型，数组长度是 n_ep 的输入值，数组中每个指针元素指向的缓存长度不能小于 65，避免长度溢出。

dsc_ctl_node：输出参数，输出指定库中控制节点的 seqno，对单节点库，输出值为 0。

dsc_status：输出参数，输出指定库的状态。

n_ok_ep：输入输出参数，指定库中包括的 ok 节点实例个数。输入参数：可获取的最大实例个数，输出参数：实际获取到的实例个数，建议输入值不小于 16，避免获取不到完整信息。对单节点库，输出值为 1。

ok_ep_arr：输出参数，输出指定库中所有 ok 节点实例的 seqno 数组，参数是 msint2 数组类型，数组长度是 n_ok_ep 的输入值。

n_break_ep：输入输出参数，指定库中当前正在故障处理的节点数。输入参数：可获取的最大实例个数，输出参数：实际获取到的实例个数，建议输入值不小于 16，避免获

取不到完整信息。

`break_ep_arr`: 输出参数, 输出指定库中当前正在故障处理的节点 `seqno` 数组, 参数是 `msint2` 数组类型, 数组长度是 `n_break_ep` 的输入值。

返回值:

0: 执行成功。

<0: 执行失败。

17. `dwmon_get_nth_ep_info`

函数原型:

DWMON_RETURN

```
dwmon_get_nth_ep_info(  
    mhandle          mhdle,  
    mschar*          group_name,  
    mschar*          db_name,  
    muint4           nth,  
    mschar*          ep_name,  
    muint4*          ep_port,  
    mschar*          ep_ok_str,  
    mschar*          svr_mode,  
    mschar*          svr_status,  
    mschar*          prim_inst,  
    mschar*          prim_mal_sta,  
    mschar*          prim_arch_type,  
    mschar*          prim_arch_sta,  
    muint8*          fseq,  
    muint8*          flsn,  
    muint8*          cseq,  
    muint8*          clsn,  
    mschar*          arch_type,
```

```

    muint4*      n_arch,

    muint4*      n_async,

    msint2*      dsc_seqno,

    mschar*      g_dw_status,

    mbyte*       mpp_flag_mem,

    mbyte*       mpp_flag_file

);

```

功能说明：

获取指定库上第 `nth` 个实例信息，可先通过 `dwmon_get_database_info` 接口获取到库上的节点实例个数 `n_ep`，`nth` 取值范围为 `[0, n_ep-1]`。

参数说明：

对字符串类型的输出参数，指向的缓存长度不能小于 65，避免溢出。

`mhdle`：输入参数，监视器操作句柄。

`group_name`：输入参数，输入指定库所在的守护进程组名，如果只有一组可以不指定。

`db_name`：输入参数，指定要获取实例信息的数据库名称，此参数必须指定。

`nth`：输入参数，指定要获取 `inst_name` 库上的第 `nth` 个实例信息。

`ep_name`：输出参数，输出第 `nth` 个实例名称。

`ep_port`：输出参数，输出实例端口号。

`ep_ok_str`：输出参数，输出守护进程认定的实例状态 (Ok/Error)。

`svr_mode`：输出参数，输出实例模式 (Normal/Primary/Standby)。

`svr_status`：输出参数，输出实例状态 (Startup/After Redo/Mount/Open/Suspend/Shutdown)。

`prim_inst`：输出参数，当前实例是 Standby 模式时该字段才有意义，输出对应的 Primary 实例名，如果是 Primary 模式，则输出为空。

`prim_mal_sta`：输出参数，当前实例是 Standby 模式时该字段才有意义，输出主库控制节点到当前库的控制节点之间的 mal 链路状态。

`prim_arch_type`：输出参数，当前实例是 Standby 模式时该字段才有意义，输出

主库到当前实例的归档类型，包括 REALTIME/TIMELY/ASYNC 这三种类型。

prim_arch_sta: 输出参数，当前实例是 Standby 模式时该字段才有意义，输出主库控制节点到当前库的控制节点的归档状态，local 守护类型的除外。

fseq: 输出参数，输出实例的 file_pkg_seq 值。

flsn: 输出参数，输出实例的 flsn 值。

cseq: 输出参数，输出实例的 cur_pkg_seq 值。

clsn: 输出参数，输出实例的 clsn 值。

arch_type: 输出参数，输出实例自身配置的归档类型，只统计 REALTIME/TIMELY 这两种归档类型，不考虑本地 LOCAL 归档类型。

n_arch: 输出参数，输出实例自身配置的归档个数，只统计 REALTIME/TIMELY 这两种归档类型下配置的归档个数。

n_async: 输出参数，输出实例自身配置的异步归档个数。

dsc_seqno: 输出参数，对 DMDSC 集群才有意义，输出实例在集群中的 seqno 序号。

g_dw_status: 输出参数，输出实例上设置的守护进程执行标记。

mpp_flag_mem: 输出参数，输出实例内存中的 mpp_ini 值。

mpp_flag_file: 输出参数，输出实例 dm.ini 中配置的 mpp_ini 值。

返回值:

0: 执行成功。

<0: 执行失败。

18. dwmon_get_ep_info_by_name

函数原型:

DWMON_RETURN

```
dwmon_get_ep_info_by_name(
    mhandle          mhdle,
    mschar*          group_name,
    mschar*          ep_name,
    muint4*          ep_port,
    mschar*          ep_ok_str,
```

mschar*	svr_mode,
mschar*	svr_status,
mschar*	prim_inst,
mschar*	prim_mal_sta,
mschar*	prim_arch_type,
mschar*	prim_arch_sta,
muint8*	fseq,
muint8*	flsn,
muint8*	cseq,
muint8*	clsn,
mschar*	arch_type,
muint4*	n_arch,
muint4*	n_async,
msint2*	dsc_seqno,
mschar*	g_dw_status,
mbyte*	mpp_flag_mem,
mbyte*	mpp_flag_file

```
);
```

功能说明：

根据指定的实例名获取实例的详细信息。

参数说明：

对字符串类型的输出参数，指向的缓存长度不能小于 65，避免溢出。

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，输入指定实例所在的守护进程组名，如果只有一组可以不指定。

ep_name：输入参数，输入要获取信息的实例名。

ep_port：输出参数，输出实例端口号。

ep_ok_str：输出参数，输出守护进程认定的实例状态 (Ok/Error)。

svr_mode：输出参数，输出实例模式 (Normal/Primary/Standby)。

svr_status: 输出参数, 输出实例状态 (Startup/After Redo/Mount/Open/Suspend/Shutdown)。

prim_inst: 输出参数, 当前实例是 Standby 模式时该字段才有意义, 输出对应的 Primary 实例名, 如果是 Primary 模式, 则输出为空。

prim_mal_sta: 输出参数, 当前实例是 Standby 模式时该字段才有意义, 输出主库控制节点到当前库的控制节点之间的 mal 链路状态。

prim_arch_type: 输出参数, 当前实例是 Standby 模式时该字段才有意义, 输出主库到当前实例的归档类型, 包括 REALTIME/TIMELY/ASYNC 这三种类型。

prim_arch_sta: 输出参数, 当前实例是 Standby 模式时该字段才有意义, 输出主库控制节点到当前库的控制节点的归档状态, local 守护类型的除外。

fseq: 输出参数, 输出实例的 file_pkg_seq 值。

flsn: 输出参数, 输出实例的 flsn 值。

cseq: 输出参数, 输出实例的 cur_pkg_seq 值。

clsn: 输出参数, 输出实例的 clsn 值。

arch_type: 输出参数, 输出实例自身配置的归档类型, 只统计 REALTIME/TIMELY 这两种归档类型, 不考虑本地 LOCAL 归档类型。

n_arch: 输出参数, 输出实例自身配置的归档个数, 只统计 REALTIME/TIMELY 这两种归档类型下配置的归档个数。

n_async: 输出参数, 输出实例自身配置的异步归档个数。

dsc_seqno: 输出参数, 对 DMDSC 集群才有意义, 输出实例在集群中的 seqno 序号。

g_dw_status: 输出参数, 输出实例上设置的守护进程执行标记。

mpp_flag_mem: 输出参数, 输出实例内存中的 mpp_ini 值。

mpp_flag_file: 输出参数, 输出实例 dm.ini 中配置的 mpp_ini 值。

返回值:

0: 执行成功。

<0: 执行失败。

19. dwmon_get_database_arch_info

函数原型:

DWMON_RETURN

```
dwmon_get_database_arch_info(
    mhandle      mhdle,
    mschar*      group_name,
    mschar*      db_name,
    mschar*      arch_type,
    muint4*      n_arch,
    mschar**     arch_dest,
    mschar**     mal_sta,
    mschar**     arch_sta,
    muint4*      n_async,
    mschar**     async_dest,
    mschar**     async_mal_sta
);
```

功能说明：

获取指定库配置的归档信息，包括归档目标实例名，到归档目标的 mal 链路状态和归档状态，以及异步归档信息。

参数说明：

arch_dest、mal_sta、arch_sta、async_dest 和 async_mal_sta 都是字符串指针数组类型，其中 arch_dest、mal_sta 和 arch_sta 的数组长度是 n_arch 的输入值，async_dest 和 async_mal_sta 的数组长度是 n_async 的输入值，建议 n_arch 和 n_async 的输入值不小于 8，避免取不到完整信息。

如果归档目标是 DMDSC 集群，则 arch_dest、async_dest 会输出完整的归档配置串（比如“DSC01/DSC02”），为避免发生溢出，arch_dest 和 async_dest 数组中的每个指针元素指向的缓存长度不能小于 272。除此之外，其他字符串指针数组中的指针元素指向的缓存长度不能小于 65，避免溢出。

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，输入指定库所在的守护进程组名，如果只有一组可以不指定。

db_name: 输入参数, 指定要获取归档信息的数据库名称。

arch_type: 输出参数, 输出指定库配置的归档类型, 只检查 REALTIME/TIMELY 这两种归档类型, 注意缓存长度不能小于 65, 避免溢出。

n_arch: 输入输出参数, 指定库配置的归档个数, 只检查 REALTIME/TIMELY 这两种归档类型。输入参数: 指定可获取的最大归档个数, 输出参数: 实际获取的归档个数, 建议输入值不小于 8, 避免取不到完整信息。

arch_dest: 输出参数, 输出指定库配置的归档目标名称, 只检查 REALTIME/TIMELY 这两种归档类型, 如果归档目标是 DMDSC 集群, 则输出完整的归档配置串 (形如 “DSC01/DSC02”), 为避免发生溢出, arch_dest 数组中的每个指针元素指向的缓存长度不能小于 272。

mal_sta: 输出参数, 输出指定库的控制节点到每个归档目标控制节点的 mal 链路状态数组, 只检查 REALTIME/TIMELY 这两种归档类型, 如果未找到控制节点, 则输出 “UNKNOWN”。

arch_sta: 输出参数, 输出指定库的控制节点到每个归档目标控制节点的归档状态数组, 指定实例是 Primary 模式时, 该参数输出的是有效值, 只检查 REALTIME/TIMELY 这两种归档类型。

n_async: 输入输出参数, 指定库配置的异步归档个数。输入参数: 指定可获取的最大归档个数, 输出参数: 实际获取的归档个数, 建议输入值不小于 8, 避免取不到完整信息。

async_dest: 输出参数, 输出指定库配置的异步归档目标名称, 如果归档目标是 DMDSC 集群, 则输出完整的归档配置串 (形如 “DSC01/DSC02”), 为避免发生溢出, async_dest 数组中的每个指针元素指向的缓存长度不能小于 272。

async_mal_sta: 输出参数, 输出指定库的控制节点到每个异步归档目标控制节点的 mal 链路状态数组。

返回值:

0: 执行成功。

<0: 执行失败。

20. dwmon_get_watch_ctl_info

函数原型:

DWMON_RETURN

```
dwmon_get_watch_ctl_info(
    mhandle      mhdle,
    mschar*      group_name,
    mschar*      db_name,
    mschar*      ctl_status,
    mschar*      desc
);
```

功能说明：

获取指定库的守护进程控制文件状态信息。

参数说明：

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，输入指定库所在的守护进程组名，如果只有一组可以不指定。

inst_name：输入参数，指定要获取控制文件信息的守护进程对应的数据库名称。

ctl_status：输出参数，输出守护进程控制文件状态，注意缓存长度不能小于 65，避免溢出。

desc：输出参数，输出本地分裂库的描述信息，如果是有效状态，描述信息为空。

返回值：

0：执行成功。

<0：执行失败。

21. dwmon_get_database_open_history

函数原型：

DWMON_RETURN

```
dwmon_get_database_open_history(
    mhandle      mhdle,
    mschar*      group_name,
```

```

    mschar*      db_name,

    muint4*      n_item

);

```

功能说明：

获取指定库的 OPEN 记录项个数。

参数说明：

mhandle：输入参数，监视器操作句柄。

group_name：输入参数，输入指定库所在的守护进程组名，如果只有一组可以不指定。

db_name：输入参数，指定要获取 OPEN 记录的守护进程对应的数据库名称。

n_item：输出参数，输出 db_name 指定的库的 OPEN 记录项个数。

返回值：

0：执行成功。

<0：执行失败。

22. dwmon_get_nth_open_history

函数原型：

DWMON_RETURN

```

dwmon_get_nth_open_history(

    mhandle      mhandle,

    mschar*      group_name,

    mschar*      db_name,

    muint4       nth,

    mschar*      tguid,

    muint8*      rowid,

    mschar*      time,

    mschar*      sys_mode,

    mschar*      p_db_name,

    mschar*      c_db_name,

```

```

muint4*      p_db_magic,

muint4*      c_db_magic,

msint2*      n_ep,

muint8*      aseq_arr,

muint8*      alsn_arr

);

```

功能说明：

获取指定组中指定库的第 *nth* 个 open 记录信息。

参数说明：

对字符串类型的输出参数，指向的缓存长度不能小于 65，避免溢出。

注意：

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，指定的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定要获取信息的数据库名称。

nth：输入参数，输入要获取指定第 *nth* 个 OPEN 记录项。

tguid：输出参数，输出 OPEN 记录唯一标识。

rowid：输出参数，输出系统表所在行的 *rowid* 值。

time：输出参数，输出 OPEN 的时间点。

sys_mode：输出参数，输出 OPEN 时所处的系统模式（PRIMARY/NORMAL）。

p_db_name：输出参数，输出对应主库的实例名。

c_db_name：输出参数，输出当前库的实例名。

p_db_magic：输出参数，输出对应主库的 *db_magic* 值。

c_db_magic：输出参数，输出当前库的 *db_magic* 值。

n_ep：输入输出参数，指定库中包含的节点实例个数。输入参数：可获取的最大实例个数，输出参数：实际获取到的实例个数，建议输入值不小于 16，避免获取不到完整信息。

aseq_arr：输出参数，输出 *pkg_seqno* 数组，长度不能小于 16。

alsn_arr：输出参数，输出 *apply_lsn* 数组，长度不能小于 16。

23. dwmon_get_watch_config_info

函数原型:

DWMON_RETURN

```
dwmon_get_watch_config_info(  
    mhandle          mhdle,  
    mschar*          group_name,  
    mschar*          inst_name,  
    mschar*          dw_type,  
    mschar*          dw_mode,  
    muint4*          dw_error_time,  
    muint4*          inst_oguid,  
    muint4*          inst_error_time,  
    muint4*          inst_recover_time,  
    mschar*          inst_ini,  
    mschar*          dcr_ini,  
    muint4*          inst_auto_restart,  
    mschar*          inst_startup_cmd,  
    muint4*          inst_host_check,  
    muint4*          rlog_send_threshold,  
    muint4*          rlog_apply_threshold,  
    mschar*          dcs_startup_cmd,  
    mschar*          dcs_ini,  
    muint4*          dcs_port  
);
```

功能说明:

获取指定实例本地的守护进程配置信息。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

`group_name`: 输入输出参数, 当前有多个组时, 作为输入参数, 指定实例所在的守护进程组名, 只有一组时允许不指定, 如果组名为空串, 则作为输出参数, 输出实例所在的守护进程组名, 输出缓存长度不能小于 65, 避免长度溢出。

`inst_name`: 输入参数, 指定要获取配置信息的守护进程本地的实例名。

`dw_type`: 输出参数, 输出 `dmwatcher.ini` 的 `dw_type` 配置值, 注意缓存长度不能小于 65, 避免溢出。

`dw_mode`: 输出参数, 输出 `dmwatcher.ini` 的 `dw_mode` 配置值, 注意缓存长度不能小于 65, 避免溢出。

`dw_error_time`: 输出参数, 输出 `dmwatcher.ini` 的 `dw_error_time` 配置值。

`inst_oguid`: 输出参数, 输出 `dmwatcher.ini` 的 `inst_oguid` 配置值。

`inst_error_time`: 输出参数, 输出 `dmwatcher.ini` 的 `inst_error_time` 配置值。

`inst_recover_time`: 输出参数, 输出 `dmwatcher.ini` 的 `inst_recover_time` 配置值。

`inst_ini`: 输出参数, 输出 `dmwatcher.ini` 的 `inst_ini` 配置值, 注意缓存长度不能小于 257, 避免溢出。

`dcr_ini`: 输出参数, 输出 `dmwatcher.ini` 的 `dcr_ini` 配置值 (守护 DMDSC 集群时需要配置), 注意缓存长度不能小于 257, 避免溢出。

`inst_auto_restart`: 输出参数, 输出 `dmwatcher.ini` 的 `inst_auto_restart` 配置值。

`inst_startup_cmd`: 输出参数, 输出 `dmwatcher.ini` 的 `inst_startup_cmd` 配置值, 注意缓存长度不能小于 257, 避免溢出。

`inst_host_check`: 输出参数, 输出守护进程是否检查实例对外服务 IP 的网卡故障, 0 表示不检查, 1 表示检查。

`rlog_send_threshold`: 输出参数, 输出主库发送日志到备库的时间阈值。

`rlog_apply_threshold`: 输出参数, 输出备库重演日志的时间阈值。

`dcs_startup_cmd`: 输出参数, 输出 `dmwatcher.ini` 的 `dcs_startup_cmd` 配置值, 注意缓存长度不能小于 257, 避免溢出。

`dcs_ini`: 输出参数, 输出 `dmwatcher.ini` 的 `dcs_ini` 配置值, 注意缓存长度不能小于 257, 避免溢出。

dcx_port: 输出参数, 输出 dmwatcher.ini 的 dcs_port 配置值。

返回值:

0: 执行成功。

<0: 执行失败。

24. dwmon_get_css_config_info

函数原型:

DWMON_RETURN

```
dwmon_get_css_config_info(
    mhandle          mhdle,
    mschar*          group_name,
    mschar*          inst_name,
    muint4*          asm_restart_itvl,
    muint4*          db_restart_itvl
);
```

功能说明:

获取和指定实例的 dcr_seqno 相同的 dmcsc 配置的自动拉起检测间隔 (DMDCR_ASM_RESTART_INTERVAL、DMDCR_DB_RESTART_INTERVAL)。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

group_name: 输入输出参数, 不能为空, 只有一组时允许不指定组名, 输入组名为空串的情况下, 接口会输出实例所在的守护进程组名, 输出缓存长度不能小于 65, 避免长度溢出。

inst_name: 输入参数, 指定要获取配置信息的实例名。

asm_restart_itvl: 输出参数, 输出实例本地的 dmcsc 配置文件 (dmdcr.ini) 中配置的 dmasmsvr 自动拉起检测间隔 (DMDCR_ASM_RESTART_INTERVAL)。

db_restart_itvl: 输出参数, 输出实例本地的 dmcsc 配置文件 (dmdcr.ini) 中配置的 dmserver 自动拉起检测间隔 (DMDCR_DB_RESTART_INTERVAL)。

返回值:

0: 执行成功。

<0: 执行失败。

25. dwmon_get_monitor_info

函数原型:

DWMON_RETURN

```
dwmon_get_monitor_info (
    mhandle          mhdle,
    mschar*          group_name,
    mschar*          db_name,
    muint4*          n_mon,
    mschar**         conn_time_arr,
    muint4*          confirm_arr,
    muint8*          mid_arr,
    mschar**         mon_ip_arr,
    mschar**         mon_version_arr
)
```

功能说明:

获取指定守护进程的监视器连接信息。同一个守护系统中允许最多 10 个监视器同时启动连接到守护进程, 可通过调用此接口获取指定守护进程上的监视器连接信息, 包括连接时间、确认监视器配置、监视器 ID、监视器 IP 和监视器版本信息。

其中 db_name 是输入输出参数, 表示指定守护进程对应的实例名, 如果此参数为空, 则根据 dmmonitor.ini 对应组中的配置顺序, 选择第一个活动的守护进程获取监视器连接信息, 并输出选中的守护进程本地库名称; 如果指定有合法值, 则返回指定库的控制守护进程上的监视器连接信息。

另外输出的数组参数中第一个数组元素存放的是本地监视器的连接信息。

参数说明:

conn_time_arr 和 mon_ip_arr 是字符串指针数组类型, 数组长度不能小于 10, 每个数组中的指针元素指向的缓存长度不能小于 65, 避免溢出。

`mon_version_arr` 也是字符串指针数组类型，数组长度不能小于 10，每个数组中的指针元素指向的缓存长度不能小于 129，避免溢出。

`mhandle`：输入参数，监视器操作句柄。

`group_name`：输入参数，输入指定库所在的守护进程组名，如果只有一组可以不指定。

`db_name`：输入输出参数，指定要获取信息的守护进程对应的实例名，在 `db_name` 为空串的情况下，会作为输出参数输出实际获取信息的实例名，此时缓存长度不能小于 65，避免输出时长度溢出。

`n_mon`：输入输出参数，指定守护进程上连接的监视器总个数。输入参数：可获取的最大个数，输出参数：实际获取的个数，建议输入值不小于 10，避免取不到完整信息。

`conn_time_arr`：输出参数，输出守护进程上建立连接的时间，数组长度是 `n_mon` 的输入值。

`confirm_arr`：输出参数，输出监视器的确认配置信息，参数为 `uint4` 数组类型，数组长度是 `n_mon` 的输入值。

`mid_arr`：输出参数，输出监视器的唯一 ID，参数为 `uint8` 数组类型，数组长度是 `n_mon` 的输入值。

`mon_ip_arr`：输出参数，输出监视器的 IP 地址，数组长度是 `n_mon` 的输入值。

`mon_version_arr`：输出参数，输出监视器的版本信息，数组长度是 `n_mon` 的输入值。

返回值：

0：执行成功。

<0：执行失败。

26. `dwmon_set_cmd_execing_with_command`

函数原型：

DWMON_RETURN

`dwmon_set_cmd_execing_with_command (`

<code>mhandle</code>	<code>mhandle,</code>
<code>mschar*</code>	<code>command</code>

```
);
```

功能说明：

如果上层有控制台工具界面，通过用户输入命令的方式调用接口，可通过此接口设置命令执行标记并传入用户执行的命令串信息，用于监视器 log 日志记录用户的操作信息。

在确认监视器模式下或者监视器设置有自动记录系统状态间隔（通过 `dwmon_set_show_interval` 或者 `dwmon_set_log_interval` 设置），调用命令接口之前，需要先调用此接口设置执行标记，避免和确认监视器自动接管故障主库或者监视器自动记录系统状态的操作发生并发冲突。

参数说明：

`mhandle`：输入参数，监视器操作句柄。

`command`：输入参数，如果上层有控制台工具界面，则表示用户手工输入的执行命令串，允许为 NULL 或空串。

返回值：

0：执行成功。

<0：执行失败。

27. `dwmon_reset_cmd_execing`

函数原型：

```
DWMON_RETURN  
  
dwmon_reset_cmd_execing(  
  
    mhandle      mhandle  
  
);
```

功能说明：

重置命令执行标记，和 `dwmon_set_cmd_execing_with_command` 接口配合使用，在调用命令接口执行完成后，需要调用此接口，重置执行标记，避免确认监视器无法执行自动接管故障主库的操作。

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。

28. dwmon_get_dw_confirm

函数原型:

DWMON_RETURN

```
dwmon_get_dw_confirm(  
    mhandle          mhdle,  
    mbool*           dw_confirm  
)
```

功能说明:

获取确认监视器标记，用于判断当前的监视器是否为确认监视器。

参数说明:

mhdle: 输入参数，监视器操作句柄

dw_confirm: 输出参数，输出确认监视器标记，1: 是确认监视器， 0: 不是确认监视器。

返回值:

0: 执行成功。

<0: 执行失败。

29. dwmon_set_show_interval

函数原型:

DWMON_RETURN

```
dwmon_set_show_interval(  
    mhandle          mhdle,  
    muint4           show_interval  
)
```

功能说明:

设置自动显示间隔。监视器根据指定的 `show_interval` 值，每隔 `show_interval` 个时间间隔，自动输出各实例的状态信息给用户，需要使用 `dwmon_get_exec_msg` 接口获取到输出消息，`show_interval` 单位为秒，默认值为 0。

参数说明：

`mhandle`：输入参数，监视器操作句柄

`show_interval`：输入参数，设置定时输出状态信息的时间间隔，单位为秒，取值为 0，或 5~3600，取值为 0 表示取消自动输出，默认值为 0。

返回值：

0：执行成功。

<0：执行失败。

30. `dwmon_get_show_interval`

函数原型：

DWMON_RETURN

```
dwmon_get_show_interval(  
    mhandle      mhandle,  
    muint4*      show_interval  
);
```

功能说明：

获取当前设置的自动显示时间间隔，单位为秒。

参数说明：

`mhandle`：输入参数，监视器操作句柄。

`show_interval`：输出参数，输出当前设置的自动显示时间间隔。

返回值：

0：获取成功。

<0：获取失败。

31. dwmon_set_log_interval

函数原型:

DWMON_RETURN

```
dwmon_set_log_interval(  
    mhandle      mhdle,  
    muint4       log_interval  
);
```

功能说明:

设置自动写入日志文件间隔。监视器按照设置的 `log_interval` 值，每隔 `log_interval` 个时间间隔，自动将各实例的状态信息写入到日志文件，`log_interval` 单位为秒，默认值为 1。

参数说明:

`mhdle`: 输入参数，监视器操作句柄。

`log_interval`: 输入参数，设置定时写入状态信息到日志文件的时间间隔，单位为秒，取值为 0、1 或 5~3600。

取值为 0 表示不写入任何信息到日志文件。

取值为 1 表示只写入普通的状态变化日志，例如收到新的信息或者检测到状态变化等。

取值为 5~3600 范围内的数值时，除了生成普通日志信息外，会按照设置的间隔定期写入站点状态信息到日志文件中。

返回值:

0: 执行成功。

<0: 执行失败。

32. dwmon_get_log_interval

函数原型:

DWMON_RETURN

```
dwmon_get_log_interval(  
    mhandle      mhdle,
```

```
    muint4*      log_interval
)

```

功能说明：

获取当前设置的自动写入日志的时间间隔，单位为秒。

参数说明：

mhandle：输入参数，监视器操作句柄。

log_interval：输出参数，输出当前设置的时间间隔。

返回值：

0：执行成功。

<0：执行失败。

33. dwmon_set_log_file_size

函数原型：

```
DWMON_RETURN
dwmon_set_log_file_size(
    mhandle      mhandle,
    muint4       log_file_size
)

```

功能说明：

设置监视器单个的日志文件大小，达到最大值后，会自动生成并切换到新的日志文件中，如果达到最大的日志总空间限制，会自动删除创建时间最早的日志文件，日志文件命名方式为“dmmonitor_年月日时分秒.log”，例如“dmmonitor_20160201155820.log”。

参数说明：

mhandle：输入参数，监视器操作句柄。

log_file_size：输入参数，设置单个日志文件大小，单位为 M，取值范围为 16~2048，默认值为 64。

返回值：

0：执行成功。

<0：执行失败。

34. dwmon_get_log_file_size

函数原型:

DWMON_RETURN

```
dwmon_get_log_file_size(  
    mhandle          mhdle,  
    muint4*          log_file_size  
);
```

功能说明:

获取当前设置的单个日志文件大小，单位为 M。

参数说明:

mhdle: 输入参数，监视器操作句柄。

log_file_size: 输出参数，输出当前设置的单个日志文件大小。

返回值:

0: 执行成功。

<0: 执行失败。

35. dwmon_get_log_path

函数原型:

DWMON_RETURN

```
dwmon_get_log_path(  
    mhandle          mhdle,  
    mschar*          log_path  
);
```

功能说明:

获取当前的日志文件路径。

参数说明:

mhdle: 输入参数，监视器操作句柄。

log_path: 输出参数，输出当前的日志文件路径，log_path 的缓存长度不能小于

257，避免溢出。

返回值：

0：执行成功。

<0：执行失败。

36. dwmon_set_log_space_limit

函数原型：

DWMON_RETURN

```
dwmon_set_log_space_limit(  
  
    mhandle          mhdle,  
  
    muint4           log_space_limit  
  
);
```

功能说明：

调用此接口，可以设置监视器的日志总空间大小，达到设置的总空间大小后，会自动删除创建时间最早的日志文件。

参数说明：

mhdle：输入参数，监视器操作句柄。

log_space_limit：输入参数，设置日志总空间大小，单位为 M，取值为 0 或 256~4096，默认值为 0，表示没有空间限制。

返回值：

0：执行成功。

<0：执行失败。

37. dwmon_get_log_space_limit

函数原型：

DWMON_RETURN

```
dwmon_get_log_space_limit(  
  
    mhandle          mhdle,
```

```
    muint4*      log_space_limit  
);
```

功能说明:

获取当前设置的日志总空间大小，单位为 M。

参数说明:

mhandle: 输入参数，监视器操作句柄。

log_space_limit: 输出参数，输出当前设置的日志总空间大小。

返回值:

0: 执行成功。

<0: 执行失败。

38. dwmon_login

函数原型:

```
DWMON_RETURN  
  
dwmon_login(  
  
    mhandle      mhandle,  
  
    mschar*      username,  
  
    mschar*      password  
  
);
```

功能说明:

登录监视器，这里的登录口令和服务器的登录口令是一致的，且必须有 DBA 权限。

校验登录信息时由守护进程转发给服务器校验，并返回校验结果给监视器，因此需要保证系统中至少有一个活动库，否则无法校验成功。

参数说明:

mhandle: 输入参数，监视器操作句柄。

username: 输入参数，登录用户名。

password: 输入参数，登录密码。

返回值:

0: 执行成功。

<0: 执行失败。

39. dwmon_logout

函数原型:

DWMON_RETURN

```
dwmon_logout(  
  
    mhandle      mhdle  
  
);
```

功能说明:

退出登录监视器。

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。

40. dwmon_set_database_recover_time

函数原型:

DWMON_RETURN

```
dwmon_set_database_recover_time(  
  
    mhandle      mhdle,  
  
    mschar*      group_name,  
  
    mschar*      db_name,  
  
    muint4       recover_time  
  
);
```

功能说明:

设置指定备库的恢复间隔时间, 设置成功后, recover_time 值保存在对应主库的守护进程内存中, 并且只对指定的备库起作用。

此接口只允许对指定备库执行修改操作,如果需要考虑对某个组或所有组中的备库执行修改操作,则可以使用 `dwmon_set_group_recover_time` 接口来实现。

参数说明:

`mhandle`: 输入参数,监视器操作句柄。

`group_name`: 输入参数,指定执行操作的守护进程组名,只有一组时可以不指定组名。

`db_name`: 输入参数,指定要设置恢复间隔时间的备库实例名。

`recover_time`: 输入参数,指定要设置的恢复间隔时间,取值 3~86400s,设置为 0 时,表示指定备库不需要有时间间隔限制,满足恢复条件后,主库的守护进程就会启动恢复流程,详细说明可参考 [3.1.9 故障恢复处理](#)。

返回值:

0: 执行成功。

<0: 执行失败。

41. `dwmon_set_database_arch_invalid`

函数原型:

DWMON_RETURN

```
dwmon_set_database_arch_invalid(
    mhandle      mhandle,
    mschar*      group_name,
    mschar*      db_name
);
```

功能说明:

设置主库到指定备库的归档状态无效。

此接口只允许对指定备库设置归档无效,如果需要考虑对某个组或所有组中的备库执行设置操作,则可以使用 `dwmon_set_group_arch_invalid` 接口来实现。

参数说明:

`mhandle`: 输入参数,监视器操作句柄。

`group_name`: 输入参数,指定执行操作的守护进程组名,只有一组时可以不指定组

名。

db_name: 输入参数, 指定要修改归档状态无效的备库实例名。

返回值:

0: 执行成功。

<0: 执行失败。

42. dwmon_startup_database_watch

函数原型:

DWMON_RETURN

```
dwmon_startup_database_watch(  
    mhandle          mhdle,  
    mschar*          group_name,  
    mschar*          db_name  
);
```

功能说明:

打开指定库的守护进程监控功能, 守护进程启动后, 默认监控功能处于打开状态。

对 DMDSC 集群, 此接口只通知控制守护进程执行。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

group_name: 输入参数, 指定要打开监控的守护进程组名, 只有一组时可以不指定组名。

db_name: 输入参数, 指定要打开监控的守护进程所在的本地实例名称。

返回值:

>=0: 执行成功, 执行前组中所有守护进程的监控已经处于打开状态时, 返回值为 100。

<0: 执行失败。

43. dwmon_startup_watch

函数原型:

DWMON_RETURN

```
dwmon_startup_watch(  
    mhandle      mhdle,  
    mschar*      group_name  
);
```

功能说明:

打开指定组的守护进程监控功能，守护进程启动后，默认监控功能处于打开状态。

对 DMDSC 集群，此接口只通知控制守护进程执行。

参数说明:

mhdle: 输入参数，监视器操作句柄。

group_name: 输入参数，指定要打开监控的守护进程组名，只有一组时可以不指定组名。

返回值:

>=0: 执行成功，执行前组中所有守护进程的监控已经处于打开状态时，返回值为 100。

<0: 执行失败。

44. dwmon_startup_watch_all

函数原型:

DWMON_RETURN

```
dwmon_startup_watch_all(  
    mhandle      mhdle  
);
```

功能说明:

打开所有组的守护进程监控功能，守护进程启动后，默认监控功能处于打开状态。

只有 MPP 主备环境允许调用此接口。

参数说明:

mhdle: 输入参数，监视器操作句柄。

返回值:

>=0: 执行成功，执行前执行库的守护进程的监控已经处于打开状态时，返回值为 100。

<0: 执行失败。

45. dwmon_stop_database_watch

函数原型:

DWMON_RETURN

```
dwmon_stop_database_watch(  
    mhandle          mhdle,  
    mschar*          group_name,  
    mschar*          db_name  
);
```

功能说明:

关闭指定库的守护进程的监控功能，监控功能被关闭后，守护进程还可以正常地接收和发送消息，但无法根据当前的实例状态来自动处理故障以及实例的自动拉起等，关闭后守护进程处于 Shutdown 状态。

如果需要正常地退出实例或者保持实例为 Mount 状态，则可以关闭监控功能，否则建议保持监控功能为打开状态，以保证能够及时处理系统的各种情况。

对 DMDSC 集群，此接口只通知控制守护进程执行。

执行此操作需要先登录监视器。

参数说明:

mhdle: 输入参数，监视器操作句柄。

group_name: 输入参数，指定要关闭监控的守护进程组名，只有一组时可以不指定组名。

db_name: 输入参数，指定要关闭监控的守护进程的本地实例名。

返回值:

>=0: 指定库的守护进程的监控关闭成功。执行前守护进程的监控已经处于关闭状态时，返回值为 100。

<0: 执行失败。

46. dwmon_stop_watch

函数原型:

DWMON_RETURN

```
dwmon_stop_watch(  
    mhandle      mhdle,  
    mschar*      group_name  
);
```

功能说明:

关闭指定组的所有守护进程的监控功能，监控功能被关闭后，守护进程还可以正常地接收和发送消息，但无法根据当前的实例状态来自动处理故障以及实例的自动拉起等，关闭后守护进程处于 Shutdown 状态。

如果需要正常地退出实例或者保持实例为 Mount 状态，则可以关闭监控功能，否则建议保持监控功能为打开状态，以保证能够及时处理系统的各种情况。

对 DMDSC 集群，此接口只通知控制守护进程执行。

执行此操作需要先登录监视器。

参数说明:

mhdle: 输入参数，监视器操作句柄。

group_name: 输入参数，指定要关闭监控的守护进程组名，只有一组时可以不指定组名。

返回值:

>=0: 所有守护进程的监控关闭成功。执行前组中所有守护进程的监控已经处于关闭状态时，返回值为 100。

<0: 执行失败

47. dwmon_stop_watch_all

函数原型:

DWMON_RETURN

```
dwmon_stop_watch_all(  

```

```

    mhandle      mhdle
);

```

功能说明：

关闭所有组的守护进程监控功能，监控功能被关闭后，守护进程还可以正常地接收和发送消息，但无法根据当前的实例状态来自动处理故障以及实例的自动拉起等，关闭后守护进程处于 Shutdown 状态。

如果需要正常地退出实例或者保持实例为 Mount 状态，则可以关闭监控功能，否则建议保持监控功能为打开状态，以保证能够及时处理系统的各种情况。

执行此操作需要先登录监视器，只有 MPP 主备环境下允许调用此接口。

参数说明：

mhdle：输入参数，监视器操作句柄。

返回值：

>=0：所有守护进程的监控关闭成功，执行前所有组守护进程的监控已经处于关闭状态时，返回值为 100。

<0：执行失败。

48. dwmon_startup_group

函数原型：

```

DWMON_RETURN

dwmon_startup_group(

    mhandle      mhdle,

    mschar*      group_name

);

```

功能说明：

启动指定组的所有实例，在 dmwatcher.ini 中 INST_AUTO_RESTART 配置为 0 时该接口可以成功执行，配置为 1 时在守护进程的监控功能打开的情况下，守护进程可以自动完成启动实例的功能，不需要再调用接口执行。

对 DMDSC 集群，由 dmcss 执行拉起动作，和 dmwatcher.ini 中的 INST_AUTO_RESTART 配置参数没有关系，由 dmdcr.ini 中的配置参数决定是否可以执

行自动拉起，如果某个实例可以被 dmcss 自动拉起，则 dmcss 会忽略收到的拉起命令，等待自动拉起。

参数说明：

mhandle：输入参数，监视器操作句柄。

group_name：输入参数，指定要启动实例的守护进程组名，只有一组时可以不指定。

返回值：

>=0：所有实例都启动成功，执行前组中所有实例已经处于活动状态时，返回值为 100。

<0：执行失败。

49. dwmon_startup_group_all

函数原型：

DWMON_RETURN

```
dwmon_startup_group_all(  
    mhandle      mhandle  
);
```

功能说明：

启动所有组的实例，在 dmwatcher.ini 中 INST_AUTO_RESTART 配置为 0 时该接口可以成功执行，配置为 1 时在守护进程的监控功能打开的情况下，守护进程可以自动完成启动实例的功能，不需要再调用接口执行。

只有 MPP 主备环境下允许调用此接口。

参数说明：

mhandle：输入参数，监视器操作句柄。

返回值：

>=0：所有实例都启动成功，执行前所有组的实例已经处于活动状态时，返回值为 100。

<0：执行失败。

50. dwmon_stop_group

函数原型：

DWMON_RETURN

```
dwmon_stop_group(
    mhandle      mhdle,
    mschar*      group_name
);
```

功能说明：

关闭指定组中的所有活动实例，关闭所有活动实例之前，会先关闭所有控制守护进程的监控功能，避免守护进程将实例再次拉起。

对 DMDSC 集群，是由 dmcss 执行退出操作，在 dmcss 通知所有实例退出之前，会先关闭 dmcss 的自动拉起功能。执行此操作需要先登录监视器，另外关闭守护进程只是将守护进程切换为 Shutdown 状态，并没有退出。

参数说明：

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，指定要关闭实例的守护进程组名，只有一组时可以不指定。

返回值：

>=0：所有实例都退出成功，执行前组中实例已经全部退出时，返回值为 100。

<0：执行失败。

51. dwmon_stop_group_all

函数原型：

DWMON_RETURN

```
dwmon_stop_group_all(
    mhandle      mhdle
);
```

功能说明：

关闭所有组的所有活动实例，关闭实例之前，会先关闭所有守护进程的监控功能，避免守护进程将实例再次拉起。

只有 MPP 主备环境下允许调用此接口，执行此操作需要先登录监视器，另外关闭守护进程只是将守护进程切换为 Shutdown 状态，并没有退出。

参数说明：

mhandle：输入参数，监视器操作句柄。

返回值：

>=0：所有实例都退出成功，执行前所有组的实例已经全部退出时，返回值为 100。

<0：执行失败。

52. dwmon_kill_group

函数原型：

DWMON_RETURN

```
dwmon_kill_group(  
    mhandle      mhandle,  
    mschar*      group_name  
);
```

功能说明：

强制杀掉指定组中的活动实例。

调用 dwmon_stop_group 关闭指定组的实例时，如果实例正在执行守护进程发起的命令，或者守护进程不是 STARTUP、OPEN 或 RECOVERY 状态，会导致接口执行失败，在这种情况下，如果需要强制关闭实例，则可以通过调用 dwmon_kill_group 接口来完成。

此接口执行时不要求先登录监视器，并且也会将守护进程切换为 Shutdown 状态（如果已经是 Shutdown 状态则跳过不再切换），避免守护进程将实例重新拉起。

参数说明：

mhandle：输入参数，监视器操作句柄。

group_name：输入参数，指定要强杀实例的守护进程组名，只有一组时可以不指定。

返回值：

>=0：所有实例都强杀成功，执行前组中实例已经全部退出并且守护进程都是 Shutdown 状态时返回值为 100。

<0：执行失败。

53. dwmon_kill_group_all

函数原型:

```
DWMON_RETURN  
  
dwmon_kill_group_all(  
  
    mhandle          mhdle  
  
);
```

功能说明:

强制杀掉所有组中的活动实例。

调用 dwmon_stop_group_all 关闭所有组的实例时，如果实例正在执行守护进程发起的命令，或者守护进程不是 STARTUP、OPEN 或 RECOVERY 状态，会导致接口执行失败，在这种情况下，如果需要强制关闭实例，则可以通过调用 dwmon_kill_group_all 接口来完成。

只有 MPP 主备环境下允许调用此接口，此接口执行时不要求先登录监视器，并且也会将守护进程切换为 Shutdown 状态（如果已经是 Shutdown 状态则跳过不再切换），避免守护进程将实例重新拉起。

参数说明:

mhdle: 输入参数，监视器操作句柄。

返回值:

>=0: 所有实例都强杀成功，执行前所有组的实例已经全部退出并且守护进程都是 Shutdown 状态则返回值为 100。

<0: 执行失败。

54. dwmon_startup_database

函数原型:

```
DWMON_RETURN  
  
dwmon_startup_database(  
  
    mhandle          mhdle,  
  
    mschar*          group_name,
```

```

    mschar*      db_name
);

```

功能说明：

启动指定组中指定的数据库实例。

如果指定的库是单机，则由守护进程执行拉起动作，如果指定的库是 DMDSC 集群，则由守护进程再去通知主从 css 各自执行拉起动作。

此命令执行前，如果守护进程处于 Shutdown 状态，会将其切换为 Startup 状态，如果主从 css 的自动拉起功能是关闭的，在各节点实例拉起成功后，会将其打开。

参数说明：

mhandle：输入参数，监视器操作句柄。

group_name：输入参数，指定要启动的库所在的组名，只有一组时可以不指定。

db_name：输入参数，指定要启动的数据库名称，此参数必须指定。

返回值：

>=0：执行成功，执行前库已经处于活动状态时返回值为 100。

<0：执行失败。

55. dwmon_stop_database

函数原型：

```

DWMON_RETURN

dwmon_stop_database(

    mhandle      mhandle,

    mschar*      group_name,

    mschar*      db_name

);

```

功能说明：

关闭指定组中指定的数据库实例。

如果指定的库是单机，则由守护进程通知单机库正常退出，如果指定的库是 DMDSC 集群，则由守护进程再去通知主 css 依次退出每个节点实例。

此命令执行完成后，会将守护进程切换为 Shutdown 状态，并将主从 css 的自动拉起

功能关闭。

注意：

如果要退出的是全局守护类型的备库，为了避免触发主库的 Failover 动作，要求此备库已经被 detach 分离出去，也就是备库归档要处于无效状态，才允许将其正常退出。

如果要退出的是主库，在自动切换模式下，为了避免触发备库的自动接管，要求所有备库都已经被 detach 分离出去，也就是所有备库归档都处于无效状态，才允许退出主库，手动切换模式下没有此要求。

参数说明：

mhdlc: 输入参数，监视器操作句柄。

group_name: 输入参数，指定要关闭的库所在的组名，只有一组时可以不指定。

db_name: 输入参数，指定要关闭的数据库名称，此参数必须指定。

返回值：

>=0: 执行成功，执行前库已经退出时返回值为 100。

<0: 执行失败。

56. dwmon_kill_database

函数原型：

DWMON_RETURN

```
dwmon_kill_database(
    mhandle      mhdle,
    mschar*      group_name,
    mschar*      db_name
);
```

功能说明：

强制关闭指定组中指定的数据库实例。

如果指定的库是单机，则由守护进程强制杀掉指定库的进程，如果指定的库是 DMDSC 集群，则由守护进程再去通知主 css 强杀所有节点实例。

注意：此命令执行完成后，会将守护进程切换为 Shutdown 状态。

参数说明：

mhdlc: 输入参数, 监视器操作句柄。

group_name: 输入参数, 指定要强制关闭的库所在的组名, 只有一组时可以不指定。

db_name: 输入参数, 指定要强制关闭的数据库名称, 此参数必须指定。

返回值:

≥ 0 : 执行成功, 执行前库已经不是活动状态时返回值为 100。

< 0 : 执行失败。

57. dwmon_open_database

函数原型:

DWMON_RETURN

```
dwmon_open_database(  
    mhandle      mhdle,  
    mschar*      group_name,  
    mschar*      db_name  
);
```

功能说明:

在系统中存在其他故障实例时, 守护进程无法自动 Open 主库, 或者系统中没有活动主库时, 守护进程无法自动 Open 备库, 这种情况下, 可以调用此接口强制 Open 指定的数据库。

执行此操作需要先登录监视器。

参数说明:

mhdlc: 输入参数, 监视器操作句柄。

group_name: 输入参数, 指定要强制 Open 库所在的组名, 只有一组时可以不指定。

db_name: 输入参数, 指定要强制 Open 的数据库名称, 此参数必须指定。

返回值:

≥ 0 : 执行成功, 执行前库已经处于 Open 状态时返回值为 100。

< 0 : 执行失败。

58. dwmon_get_switchover_standby

函数原型:

DWMON_RETURN

```
dwmon_get_switchover_standby(  
    mhandle          mhdle,  
    mschar*          group_name,  
    muint4*          n_db,  
    mschar**          db_name_arr  
);
```

功能说明:

获取可以切换为主库的备库信息,这里的切换是指主备库都正常,且处于 Open 状态时,交换主备库模式,将备库切换为新的主库,主库切换为备库的操作。

参数说明:

mhdle: 输入参数,监视器操作句柄。

group_name: 输入参数,指定要获取切换信息的组名,只有一组时可以不指定。

n_db: 输入输出参数,可以执行切换的备库个数。输入参数:指定可获取的最大备库个数,输出参数:实际获取的备库个数,建议输入值不小于 8,避免取不到完整信息。

db_name_arr: 输出参数,输出可以执行切换的备库名称,参数是字符串指针数组类型,数组长度是 n_db 的输入值,数组中每个指针元素指向的缓存长度不能小于 65,避免溢出。

返回值:

0: 执行成功。

<0: 执行失败。

59. dwmon_switchover

函数原型:

DWMON_RETURN

```
dwmon_switchover(  

```

```
mhandle      mhdle,  
  
mschar*      group_name,  
  
mschar*      db_name  
  
);
```

功能说明:

使用指定的备库和当前的主库执行切换操作，这里的切换是指主备库都正常，且处于 Open 状态时，交换主备库模式，将备库切换为新的主库，主库切换为备库的操作。

此接口只有在指定的备库满足切换条件时才能够执行成功，可以先调用 dwmon_get_switchover_standby 获取到满足条件的备库列表。

执行此操作需要先登录监视器。

参数说明:

mhdle: 输入参数，监视器操作句柄。

group_name: 输入参数，指定要执行切换的备库所在的组名，只有一组时可以不指定。

db_name: 输入参数，指定要执行切换的备库名称，只有一个备库时可以不指定。

返回值:

0: 执行成功。

<0: 执行失败。

60. dwmon_get_takeover_standby

函数原型:

```
DWMON_RETURN  
  
dwmon_get_takeover_standby(  
  
    mhandle      mhdle,  
  
    mschar*      group_name,  
  
    mbool        force_flag,  
  
    muint4*      n_db,  
  
    mschar**     db_name_arr  
  
);
```

功能说明：

获取可以接管为主库的备库信息，在主库发生故障时，可以通过调用此接口查找可以接管为新的主库的备库实例，其中包括正常接管和强制接管，通过输入参数 `force_flag` 可以指定接管方式。

参数说明：

`mhdl`: 输入参数，监视器操作句柄。

`group_name`: 输入参数，指定要获取接管信息的组名，只有一组时可以不指定。

`force_flag`: 输入参数，指定接管方式，`TRUE`: 强制接管，`FALSE`: 正常接管。

`n_db`: 输入输出参数，可以执行接管操作的备库个数。输入参数：可获取的最大备库个数，输出参数：实际获取的备库个数，建议输入值不小于 8，避免取不到完整信息。

`db_name_arr`: 输出参数，输出可以执行接管操作的备库实例名，参数是字符串指针数组类型，数组长度是 `n_db` 的输入值，数组中每个指针元素指向的缓存长度不能小于 65，避免溢出。

返回值：

0: 执行成功。

<0: 执行失败。

61. dwmon_takeover

函数原型：

DWMON_RETURN

```
dwmon_takeover(
    mhandle      mhdl,
    mschar*      group_name,
    mschar*      db_name
);
```

功能说明：

主库故障后，可以通过调用此接口执行正常地接管操作，注意，此接口只有在指定备库满足正常接管条件时才能够执行成功，可以先调用 `dwmon_get_takeover_standby` 获取到符合正常接管条件的备库实例，`force_flag` 指定为 `FALSE`。

执行此操作需要先登录监视器。

参数说明：

mhandle：输入参数，监视器操作句柄。

group_name：输入参数，指定要执行接管的备库所在的组名，只有一组时可以不指定。

db_name：输入参数，指定要执行接管的备库名称，只有一个备库时可以不指定。

返回值：

0：执行成功。

<0：执行失败。

62. dwmon_takeover_force

函数原型：

DWMON_RETURN

```
dwmon_takeover_force(  
  
    mhandle      mhandle,  
  
    mschar*      group_name,  
  
    mschar*      db_name  
  
);
```

功能说明：

主库故障后，可以通过调用此接口执行强制接管操作，注意，此接口只有在指定备库满足强制接管条件时才能够执行成功，可以先调用 dwmon_get_takeover_standby 获取到符合强制接管条件的备库实例，force_flag 指定为 TRUE。

执行此操作需要先登录监视器。

参数说明：

mhandle：输入参数，监视器操作句柄。

group_name：输入参数，指定要执行强制接管的备库所在的组名，只有一组时可以不指定。

db_name：输入参数，指定要执行强制接管的备库名称，只有一个备库时可以不指定。

返回值：

0：执行成功。

<0：执行失败。

63. dwmon_check_mppctl

函数原型：

```
DWMON_RETURN  
  
dwmon_check_mppctl(  
  
    mhandle          mhdle  
  
);
```

功能说明：

配置为 MPP 主备环境时，可通过调用此接口检查当前各个活动节点的 dmmppctl 是否处于一致状态，只有 MPP 主备环境下允许调用此接口。

参数说明：

mhdle：输入参数，监视器操作句柄。

返回值：

0：处于一致状态。

<0：检查失败，或者控制文件不一致（返回值为-12）。

64. dwmon_recover_mppctl

函数原型：

```
DWMON_RETURN  
  
dwmon_recover_mppctl(  
  
    mhandle          mhdle  
  
);
```

功能说明：

配置为 MPP 主备环境时，如果某个活动节点的 dmmppctl 不一致，可通过调用此接口恢复到一致状态。

只有 MPP 主备环境下允许调用此接口，并且需要先登录监视器。

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。

65. dwmon_get_mpp_site_info

函数原型：

DWMON_RETURN

```
dwmon_get_mpp_site_info(  
    mhandle      mhandle,  
    muint4*      n_site,  
    muint4*      ep_seqno,  
    mschar**     db_name,  
    mschar**     db_ip,  
    muint4*      db_port  
);
```

功能说明：

配置为 MPP 主备环境时，可通过调用此接口获取 MPP 节点信息，注意只有 MPP 主备环境下允许调用此接口。

参数说明：

mhandle：输入参数，监视器操作句柄。

n_site：输入输出参数，MPP 节点数。输入参数：可获取的最大节点数，输出参数：实际获取的节点数，建议输入值不小于 1024，避免取不到完整信息。

ep_seqno：输出参数，输出 MPP 节点号，参数为 muint4 数组类型，数组长度是 n_site 的输入值。

db_name：输出参数，输出 MPP 节点实例名，参数是字符串指针数组类型，数组长度是 n_site 的输入值，数组中每个指针元素指向的缓存长度不能小于 65，避免溢出。

db_ip：输出参数，输出 MPP 节点 IP 地址，参数是字符串指针数组类型，数组长度

是 `n_site` 的输入值，数组中每个指针元素指向的缓存长度不能小于 65，避免溢出。

`db_port`：输出参数，输出 MPP 节点端口号，参数为 `muint4` 数组类型，数组长度是 `n_site` 的输入值。

返回值：

0：执行成功。

<0：执行失败。

66. dwmon_tip

函数原型：

DWMON_RETURN

```
dwmon_tip(
    mhandle      mhdle
);
```

功能说明：

可调用此接口查看系统当前运行状态，输出的系统状态信息可通过消息相关接口获取到。

参数说明：

`mhdle`：输入参数，监视器操作句柄。

返回值：

0：系统运行正常，所有实例和守护进程都处于 OPEN 状态。

-25：所有组中都有正常的主库实例（实例和守护进程都是 OPEN 状态），不影响系统使用，但是存在有其他异常实例，具体描述信息可通过消息相关接口获取。

-26：系统运行异常，不是所有组都有正常的主库实例，守护进程正在自动处理，请稍后再次调用接口判断，具体描述信息可通过消息相关接口获取。

-27：系统运行异常，不是所有组都有正常的主库实例，需要用户根据描述信息确认是否进行人工干预，具体描述信息可通过消息相关接口获取。

其他<0 的值：表示接口执行失败。

67. dwmon_set_group_arch_invalid

函数原型:

DWMON_RETURN

```
dwmon_set_group_arch_invalid(  
    mhandle          mhdle,  
    mschar*          group_name  
);
```

功能说明:

修改指定组中所有备库的归档为无效状态，如果未指定组名，则通知所有组执行。

注意：执行此操作需要先登录监视器。

参数说明:

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，守护组名，不指定则对所有组执行。

返回值:

0：执行成功。

<0：执行失败。

68. dwmon_set_group_recover_time

函数原型:

DWMON_RETURN

```
dwmon_set_group_recover_time(  
    mhandle          mhdle,  
    mschar*          group_name,  
    muint4           recover_time  
);
```

功能说明:

修改指定组中所有备库的恢复间隔 (3~86400s)，只修改组中主库对应守护进程的内存值，如果未指定组名，则通知所有组执行。

注意：执行此操作需要先登录监视器。

参数说明：

mhandle：输入参数，监视器操作句柄。

group_name：输入参数，守护组名，不指定则对所有组执行。

recover_time：输入参数，指定恢复间隔 (3~86400s)，只修改组中主库守护进程的内存值。

返回值：

0：执行成功。

<0：执行失败。

69. dwmon_set_group_para_value

函数原型：

DWMON_RETURN

```
dwmon_set_group_para_value(
    mhandle      mhandle,
    mschar*      group_name,
    mschar*      para_name,
    mschar*      para_value
);
```

功能说明：

修改指定组的所有守护进程的指定配置参数，同时修改 dmwatcher.ini 配置文件和内存值，如果未指定组名，则通知所有组执行。

目前可修改的参数包括：DW_ERROR_TIME、INST_RECOVER_TIME、INST_ERROR_TIME、INST_AUTO_RESTART、INST_SERVICE_IP_CHECK、RLOG_SEND_THRESHOLD、RLOG_APPLY_THRESHOLD。

如果组中某个守护进程故障，则跳过不修改，如果要重启故障的守护进程，需要在重启前修改守护进程 dmwatcher.ini 中的 DW_ERROR_TIME 和远程一致，否则 dmmonitor 执行命令时会报错处理。

注意：执行此操作需要先登录监视器。

参数说明：

mhdlc: 输入参数，监视器操作句柄。

group_name: 输入参数，守护组名，不指定则对所有组执行。

para_name: 输入参数，指定要修改的参数名称，目前只允许是 DW_ERROR_TIME、INST_RECOVER_TIME、INST_ERROR_TIME、INST_AUTO_RESTART、INST_SERVICE_IP_CHECK、RLOG_SEND_THRESHOLD、RLOG_APPLY_THRESHOLD 中的一种。

para_value: 输入参数，指定要修改的参数值，和 para_name 对应。

返回值：

0: 执行成功。

<0: 执行失败。

70. dwmon_set_group_auto_restart_on**函数原型：**

DWMON_RETURN

```
dwmon_set_group_auto_restart_on(
    mhandle      mhandle,
    mschar*      group_name
);
```

功能说明：

打开指定组内所有 DMDSC 集群节点实例的自动拉起标记。

DMDSC 集群的自动拉起是靠 dmcsc 执行的，dmcsc 只负责拉起和自己的 dcr_seqno 相同的 db 节点。

此接口只修改 dmcsc 内存中的自动拉起标记值，不会修改 dmdcr.ini 中的配置值，如果 dmcsc 重启，此修改会丢失。

如果 dmcsc 未配置自动拉起参数（DMDCR_DB_RESTART_INTERVAL/DMDCR_DB_STARTUP_CMD），此命令会执行失败。

此接口执行时，有可能所有节点实例都还未启动，无法校验登录口令，因此不要求先登录监视器。

参数说明：

mhdlc: 输入参数，监视器操作句柄。

group_name: 输入参数，指定要执行操作的组名，只有一组时可以不指定。

返回值：

0: 执行成功。

<0: 执行失败。

71. dwmon_set_group_auto_restart_off

函数原型：

DWMON_RETURN

```
dwmon_set_group_auto_restart_off(  
  
    mhandle          mhdle,  
  
    mschar*          group_name  
  
);
```

功能说明：

关闭指定组内所有 DMDSC 集群节点实例的自动拉起标记。

和 dwmon_set_group_auto_restart_on 接口相对应，可以一次性关闭守护系统内所有 DMDSC 集群上 dmcsc 的自动拉起标记。

此命令只修改 dmcsc 内存中的自动拉起标记值，如果 dmcsc 重启，此修改会丢失。

执行此操作需要先登录监视器。

参数说明：

mhdlc: 输入参数，监视器操作句柄。

group_name: 输入参数，指定要执行操作的组名，只有一组时可以不指定。

返回值：

0: 执行成功。

<0: 执行失败。

72. dwmon_detach_database

函数原型:

DWMON_RETURN

```
dwmon_detach_database(  
    mhandle      mhdle,  
    mschar*      group_name,  
    mschar*      db_name  
);
```

功能说明:

此接口允许将指定的备库分离出守护进程组，执行此接口需要先登录监视器。

对于全局守护类型的备库，使用此命令不会触发主库的 Failover 故障处理流程。该接口不会修改相关的配置信息，只是为了需要主动维护备库时使用。

接口会依次执行下列操作：

1. 设置指定备库的恢复间隔为 86400 (s)
2. 修改主库到指定备库的归档状态无效

本接口不会主动退出备库实例，如果需要执行退出操作，可再通过调用 dwmon_stop_database 接口完成。

在备库维护结束后，如果备库是退出状态，可先调用 dwmon_startup_database 接口将备库启动，然后再调用 dwmon_attach_database 接口将备库重加入守护进程组。

参数说明:

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，指定要执行操作的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定要正常分离出的备库名称，此参数必须指定。

返回值:

0：执行成功。

<0：执行失败。

73. dwmon_attach_database

函数原型:

DWMON_RETURN

```
dwmon_attach_database(  
    mhandle          mhdle,  
    mschar*          group_name,  
    mschar*          db_name  
);
```

功能说明:

在使用 dwmon_detach_database 接口分离出去的备库维护完成后, 可通过调用此接口将备库重加入守护进程组, 执行此接口需要先登录监视器。

此接口执行时, 会通知主库将指定备库的恢复间隔调整为 3s, 3s 后主库守护进程会发起 Recovery 流程, 将备库重加入守护系统。

注意, 此接口不会将备库拉起, 如果备库处于退出状态, 需要先调用 dwmon_startup_database 接口将备库启动。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

group_name: 输入参数, 指定要执行操作的守护进程组名, 只有一组时可以不指定。

db_name: 输入参数, 指定要重加入的备库名称, 此参数必须指定。

返回值:

0: 执行成功。

<0: 执行失败。

74. dwmon_get_source_database_name

函数原型:

DWMON_RETURN

```
dwmon_get_source_database_name(  
    mhandle          mhdle,
```

```

mschar*      group_name,

mschar*      db_name,

mschar*      source_name

);

```

功能说明：

获取指定库的源库名称，指定库为备库时才有意义，可以是异步备库。

参数说明：

mhandle：输入参数，监视器操作句柄。

group_name：输入参数，指定备库所属的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定备库名称。

source_name：输出参数，输出备库的源库名称，注意缓存长度不能小于 65，避免发生溢出。

返回值：

0：执行成功。

<0：执行失败。

75. dwmon_get_ep_arch_send_info

函数原型：

DWMON_RETURN

```

dwmon_get_ep_arch_send_info(

    mhandle      mhandle,

    mschar*      group_name,

    mschar*      source_name,

    mschar*      db_name,

    muint4*      time_in_ini,

    muint4*      time_in_mem,

    mschar*      last_recover_time,

    mint4*       last_recover_code,

    mschar*      arch_type,

```

mschar*	mal_sta,
mschar*	arch_sta,
mschar*	send_type,
mschar*	last_start_time,
mschar*	last_end_time,
mint8*	last_send_time,
muint8*	last_start_lsn,
muint8*	last_end_lsn,
muint4*	last_send_len,
muint4*	last_send_ptx,
mint4*	code,
mschar*	desc_info,
mint4*	recnt_cnt,
mint8*	recnt_send_len,
mint8*	recnt_ptx_cnt,
mint8*	recnt_send_time,
mint8*	max_send_time,
mschar*	max_end_time,
muint4*	max_ptx_cnt,
muint4*	max_send_len,
mint8*	max_send_lsn,
mint8*	total_send_cnt,
mint8*	total_send_len,
mint8*	total_ptx_cnt,
mint8*	total_send_time

);

功能说明:

获取指定的源实例到指定备库的归档发送信息。

源库如果是 DMDSC 集群，则需要依次取出集群内每个源实例到指定备库的归档发送信

息，这个需要结合 `dwmon_get_source_database_name` 和 `dwmon_get_database_ep_info` 这两个接口来完成。

首先用 `dwmon_get_source_database_name` 获取到源库名称，再通过 `dwmon_get_database_ep_info` 获取到源库的所有实例名，再通过本接口依次取出源库每个实例到指定备库的归档发送信息，接口中的 `source_name` 字段就是指源库中的某个实例名。

如果备库是 DMDSC 集群，则备库控制节点为重演节点，以下参数说明中的备库均指备库控制节点，包括 `mal_sta` 和 `arch_sta` 这两个字段，如果备库是单节点，则就指备库自身。

参数说明：

`mhandle`：输入参数，监视器操作句柄。

`group_name`：输入参数，输入指定备库所在的守护进程组名，如果只有一组可以不指定。

`source_name`：输入参数，输入要获取归档发送信息的源库实例名。

`db_name`：输入参数，输入要获取归档发送信息的备库名称。

`time_in_ini`：输出参数，输出源库守护进程 `dmwatcher.ini` 中配置 `INST_RECOVER_TIME` 值。

`time_in_mem`：输出参数，输出源库守护进程内存中的 `INST_RECOVER_TIME` 值。

`last_recover_time`：输出参数，输出源实例上次对指定备库执行恢复操作的时间，对 `local` 守护类型的备库无效，`NONE` 表示未执行过恢复操作，注意缓存长度不能小于 65，避免溢出。

`last_recover_code`：输出参数，输出源实例上次对指定备库执行恢复操作的执行结果 `code`，对 `local` 守护类型无效，0 表示执行成功，100 表示实际未执行，小于 0 表示执行失败。

`arch_type`：输出参数，输出源实例 `source_name` 到备库的归档类型，包括 `REALTIME/TIMELY/ASYNC` 这三种类型，注意缓存长度不能小于 65，避免溢出。

`mal_sta`：输出参数，输出源实例 `source_name` 到备库的 `mal` 链路状态，注意缓存长度不能小于 65，避免溢出。

`arch_sta`：输出参数，输出源实例 `source_name` 到备库的归档状态，注意缓存长度不能小于 65，避免溢出。

`send_type`: 输出参数, 输出源实例 `source_name` 到备库的归档发送类型, FOR REALTIME SEND 和 FOR TIMELY SEND 表示正常的归档发送, FOR RECOVER SEND 表示守护进程发起的归档发送, FOR ASYNC SEND 表示定时器触发的异步归档发送, 注意缓存长度不能小于 65, 避免溢出。

`last_start_time`: 输出参数, 输出源实例 `source_name` 到备库的最近一次归档发送起始时间, NONE 表示未执行过发送操作, 注意缓存长度不能小于 65, 避免溢出。

`last_end_time`: 输出参数, 输出源实例 `source_name` 到备库的最近一次归档发送结束时间, `end_time` 为 NONE 时, 如果 `start_time` 也为 NONE, 则表示未执行过发送操作, 如果 `start_time` 为具体时间, 则表示发送尚未结束, 注意缓存长度不能小于 65, 避免溢出。

`last_send_time`: 输出参数, 输出源实例 `source_name` 到备库的最近一次归档发送耗时 (微秒)。

`last_start_lsn`: 输出参数, 输出源实例 `source_name` 到备库的最近一次归档发送起始 lsn。

`last_end_lsn`: 输出参数, 输出源实例 `source_name` 到备库的最近一次归档发送结束 lsn。

`last_send_len`: 输出参数, 输出源实例 `source_name` 到备库的最近一次归档发送长度 (字节)。

`last_send_ptx`: 输出参数, 输出源实例 `source_name` 到备库的最近一次发送的 PTX 个数。

`code`: 输出参数, 输出源实例 `source_name` 到备库的最近一次归档发送 `code`, 0 表示执行成功, 100 表示实际未执行, 小于 0 表示执行失败。

`desc_info`: 输出参数, 输出源实例 `source_name` 到备库的最近一次归档发送描述信息, 注意缓存长度不能小于 513, 避免溢出。

`recnt_cnt`: 输出参数, 输出源实例 `source_name` 到备库的最近日志发送次数 N。

`recnt_send_len`: 输出参数, 输出源实例 `source_name` 到备库的最近 N 次累计日志发送长度 (字节), N 值取 `recnt_cnt` 值, 下同。

`recnt_ptx_cnt`: 输出参数, 输出源实例 `source_name` 到备库的最近 N 次累计发送的 PTX 个数。

`recnt_send_time`: 输出参数, 输出源实例 `source_name` 到备库的最近 N 次累计

日志发送耗时（微秒）。

max_send_time: 输出参数, 输出源实例 source_name 到备库的最大发送耗时（微秒）。

max_end_time: 输出参数, 输出源实例 source_name 到备库最大发送耗时的结束时间, 注意缓存长度不能小于 65, 避免溢出。

max_ptx_cnt: 输出参数, 输出源实例 source_name 到备库发送的最大 PTX 个数。

max_send_len: 输出参数, 输出源实例 source_name 到备库发送的最大日志长度（字节）。

max_send_lsn: 输出参数, 输出源实例 source_name 到备库发送的最大日志 LSN。

total_send_cnt: 输出参数, 输出源实例 source_name 到备库总的归档发送次数。

total_send_len: 输出参数, 输出源实例 source_name 到备库总的归档发送长度（字节）。

total_ptx_cnt: 输出参数, 输出源实例 source_name 到备库总的 PTX 发送个数。

total_send_time: 输出参数, 输出源实例 source_name 到备库总的归档发送耗时（微秒）。

返回值:

0: 执行成功。

<0: 执行失败。

76. dwmon_get_n_apply

函数原型:

DWMON_RETURN

dwmon_get_n_apply(

mhandle	mhdle,
mschar*	group_name,
mschar*	db_name,
mschar*	ep_name,
msint2*	n_apply,
mschar*	prim_name

);

功能说明:

获取指定备库上的重演节点个数信息，也就是源库的节点个数。

参数说明:

mhdle: 输入参数，监视器操作句柄。

group_name: 输入参数，指定备库所属的守护进程组名，只有一组时可以不指定。

db_name: 输入参数，指定备库名称。

ep_name: 输出参数，输出获取重演信息的备库控制节点名称。

n_apply: 输出参数，输出备库控制节点上记录的重演节点个数。

prim_name: 输出参数，输出备库对应的源库名称。

返回值:

0: 执行成功。

<0: 执行失败。

77. dwmon_get_ep_nth_apply_info**函数原型:**

DWMON_RETURN

```
dwmon_get_ep_nth_apply_info(
    mhandle          mhdle,
    mschar*          group_name,
    mschar*          db_name,
    muint4           nth,
    mschar*          prim_name,
    msint2*          dsc_seqno,
    muint8*          sseq,
    muint8*          slsn,
    muint8*          kseq,
    muint8*          klsn,
    muint8*          aseq,
```

```

    muint8*      alsn,

    muint4*      n_task,

    muint8*      tsk_mem_used

);

```

功能说明：

获取指定备库上第 `nth` 组重演信息上记录的 `lsn` 等信息。

当备库对应的源库是 DMDSC 集群时，假如 DMDSC 集群有 `N` 个节点，那么备库上就会有 `N` 组重演信息依次和源库的每个节点相对应，`N` 值可以通过 `dwmon_get_n_apply` 接口中的 `n_apply` 字段值获取，本接口的输入参数 `nth` 的取值范围为 `[0, n_apply-1]`。

此接口只输出 `dsc_seqno`、`sseq`、`slsn`、`kseq`、`klsn`、`aseq`、`alsn` 等简略的重演信息，如果还需要获取详细的重演统计信息，请参考 `dwmon_get_ep_nth_apply_stat` 接口说明。

参数说明：

`mhandle`：输入参数，监视器操作句柄。

`group_name`：输入参数，指定备库所属的守护进程组名，只有一组时可以不指定。

`db_name`：输入参数，指定备库名称。

`nth`：输入参数，指定要获取备库上第 `nth` 组重演信息。

`prim_name`：输出参数，第 `nth` 组重演信息对应的源库实例名，注意缓存长度不能小于 65，避免溢出。

`dsc_seqno`：输出参数，第 `nth` 组重演信息对应的源库实例的 `seqno` 序号。

`sseq`：输出参数，备库可重演到的最大日志包序号。

`slsn`：输出参数，备库可重演到的最大 LSN，对应的日志包序号为 `sseq`。

`kseq`：输出参数，非自动切换模式下，备库保持不重演的日志包序号。

`klsn`：输出参数，非自动切换模式下，备库保持不重演的最大 LSN，对应的日志包序号为 `kseq`。

`aseq`：输出参数，备库已经写入联机日志文件的原始日志包序号。

`alsn`：输出参数，备库已经写入联机日志文件的原始 LSN 值，对应的日志包序号为 `aseq`。

`n_task`：输出参数，第 `nth` 组重演信息中当前还剩余的重演任务个数。

tsk_mem_used: 输出参数,第 nth 组重演信息中执行重演任务所占用的内存大小(字节)。

返回值:

0: 执行成功。

<0: 执行失败。

78. dwmon_get_ep_nth_apply_stat

函数原型:

DWMON_RETURN

```
dwmon_get_ep_nth_apply_stat(  
    mhandle          mhdle,  
    mschar*          group_name,  
    mschar*          db_name,  
    muint4           nth,  
    mschar*          prim_name,  
    msint2*          dsc_seqno,  
    muint8*          sseq,  
    muint8*          slsn,  
    muint8*          kseq,  
    muint8*          klsn,  
    muint8*          aseq,  
    muint8*          alsn,  
    muint4*          n_task,  
    muint8*          tsk_mem_used,  
    mint4*           recnt_apply_num,  
    muint4*          last_recved_len,  
    mint8*           last_res_time,  
    mint8*           last_wait_time,  
    muint4*          last_apply_len,  

```

```

    mint8*      last_apply_time,
    mint8*      recnt_recved_len,
    mint8*      recnt_res_time,
    mint8*      recnt_wait_time,
    mint8*      recnt_apply_len,
    mint8*      recnt_apply_time,
    mint8*      max_res_time,
    mint8*      max_wait_time,
    mint8*      max_apply_time,
    muint4*     max_apply_len,
    mint8*      total_recved_num,
    mint8*      total_recved_len,
    mint8*      total_apply_num,
    mint8*      total_apply_len,
    mint8*      total_res_time,
    mint8*      total_wait_time,
    mint8*      total_apply_time
);

```

功能说明：

获取指定备库上第 nth 组重演信息，包含详细的重演统计信息。

当备库对应的源库是 DMDSC 集群时，假如 DMDSC 集群有 N 个节点，那么备库上就会有 N 组重演信息依次和源库的每个节点相对应，N 值可以通过 dwmon_get_n_apply 接口中的 n_apply 字段值获取，本接口的输入参数 nth 的取值范围为 [0, n_apply-1]。

参数说明：

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，输入指定备库所在的守护进程组名，如果只有一组可以不指定。

db_name：输入参数，指定要获取重演信息的备库名称。

nth：输入参数，指定要获取备库上第 nth 组重演信息。

prim_name: 输出参数, 第 nth 组重演信息对应的源库实例名, 注意缓存长度不能小于 65, 避免溢出。

dsc_seqno: 输出参数, 第 nth 组重演信息对应的源库实例的 seqno 序号。

sseq: 输出参数, 备库可重演到的最大日志序号值。

slsn: 输出参数, 备库可重演到的最大 LSN 值, 对应的日志包序号为 sseq。

kseq: 输出参数, 非自动切换模式下, 备库保持不重演的日志的序号。

klsn: 输出参数, 非自动切换模式下, 备库保持不重演的最大 LSN 值, 对应的日志包序号为 kseq。

aseq: 输出参数, 备库已经写入联机日志文件的原始日志包序号。

alsn: 输出参数, 备库已经写入到联机日志文件的原始 LSN 值, 对应的日志包序号为 aseq。

n_task: 输出参数, 第 nth 组重演信息中当前还剩余的重演任务个数。

tsk_mem_used: 输出参数, 第 nth 组重演信息中执行重演任务所占用的内存大小(字节)。

/** 以下均为指定备库相对于主库第 nth 个节点的重演统计信息 **/

recnt_apply_num: 输出参数, 输出最近重演的日志包个数 N。

last_recved_len: 输出参数, 输出最近一次收到的日志长度 (字节)。

last_res_time: 输出参数, 输出最近一次消息响应时间 (微秒)。

last_wait_time: 输出参数, 输出最近一次任务等待时间 (微秒)。

last_apply_len: 输出参数, 输出最近一次重演的日志长度 (字节)。

last_apply_time: 输出参数, 输出最近一次重演耗时 (微秒)。

recnt_recved_len: 输出参数, 输出最近 N 次累计接收日志长度 (字节), N 值取自 recnt_apply_num, 下同。

recnt_res_time: 输出参数, 输出最近 N 次累计响应时间 (微秒)。

recnt_wait_time: 输出参数, 输出最近 N 次累计任务等待时间 (微秒)。

recnt_apply_len: 输出参数, 输出最近 N 次累计重演日志长度 (字节)。

recnt_apply_time: 输出参数, 输出最近 N 次累计重演耗时 (微秒)。

max_res_time: 输出参数, 输出最大的消息响应时间 (微秒)。

max_wait_time: 输出参数, 输出最大的任务等待时间 (微秒)。

max_apply_time: 输出参数, 输出最大的任务重演时间 (微秒)。

max_apply_len: 输出参数, 输出和 max_apply_time 对应的重演日志长度(字节)。

total_recved_num: 输出参数, 输出累计收到的日志包个数。

total_recved_len: 输出参数, 输出累计收到的日志量(字节)。

total_apply_num: 输出参数, 输出累计重演的日志缓存个数。

total_apply_len: 输出参数, 输出累计重演日志量(字节)。

total_res_time: 输出参数, 输出累计消息响应时间(微秒)。

total_wait_time: 输出参数, 输出累计任务等待时间(微秒)。

total_apply_time: 输出参数, 输出累计任务重演时间(微秒)。

返回值:

0: 执行成功。

<0: 执行失败。

79. dwmon_clear_database_arch_send_info

函数原型:

DWMON_RETURN

```
dwmon_clear_database_arch_send_info(
    mhandle          mhdle,
    mschar*          group_name,
    mschar*          db_name
);
```

功能说明:

清理指定组中主库到指定备库实例的最近 N 次归档发送信息(通知主库执行)。函数在使用前, 需要先登录监视器。

N 值取主库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际归档发送次数中的较小值, 实际归档发送次数可通过查询 V\$ARCH_SEND_INFO 获取。

如果主库是 DMDSC 集群, 则会通知集群内所有实例执行清理。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

group_name: 输入参数, 指定备库所属的守护进程组名, 只有一组时可以不指定。

db_name: 输入参数, 指定备库名称。

返回值:

0: 执行成功。

<0: 执行失败。

80. dwmon_clear_group_arch_send_info

函数原型:

DWMON_RETURN

```
dwmon_clear_group_arch_send_info(  
  
    mhandle          mhdle,  
  
    mschar*          group_name  
  
);
```

功能说明:

清理指定组中主库到所有实时或即时备库实例的最近 N 次归档发送信息（通知主库执行）。函数在使用前，需要先登录监视器。

N 值取主库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际归档发送次数中的较小值，实际归档发送次数可通过查询 V\$ARCH_SEND_INFO 获取。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

group_name: 输入参数, 指定守护进程组名, 如果不指定则通知所有组执行。

返回值:

0: 执行成功。

<0: 执行失败。

81. dwmon_clear_database_rapply_stat

函数原型:

DWMON_RETURN

```
dwmon_clear_database_rapply_stat(  
  

```

```

    mhandle      mhdle,

    mschar*      group_name,

    mschar*      db_name

);

```

功能说明：

清理指定备库实例的最近 N 次日志重演信息（通知备库执行）。函数在使用前，需要先登录监视器。

N 值取备库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际重演次数中的较小值，实际重演次数可通过查询 V\$RAPPLY_STAT 获取。

参数说明：

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，指定备库所属的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定备库名称。

返回值：

0：执行成功。

<0：执行失败。

82. dwmon_clear_group_rapply_stat

函数原型：

```

DWMON_RETURN

dwmon_clear_group_rapply_stat(

    mhandle      mhdle,

    mschar*      group_name

);

```

功能说明：

清理指定组中所有备库实例的最近 N 次日志重演信息（通知所有备库执行）。函数在使用前，需要先登录监视器。

N 值取备库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际重演次数中的较小值，实际重演次数可通过查询 V\$RAPPLY_STAT 获取。

参数说明：

mhdlc: 输入参数，监视器操作句柄。

group_name: 输入参数，指定守护进程组名，如果不指定则通知所有组执行。

返回值：

0: 执行成功。

<0: 执行失败。

83. dwmon_check_recover**函数原型：**

DWMON_RETURN

```
dwmon_check_recover(
    mhandle      mhdle,
    mschar*      group_name,
    mschar*      db_name
);
```

功能说明：

检查指定组的指定备库是否满足自动 Recover 条件，如果不满足条件，主库的守护进程不会对指定备库发起自动 Recovery 流程。

参数说明：

mhdlc: 输入参数，监视器操作句柄。

group_name: 输入参数，指定备库所属的守护进程组名，只有一组时可以不指定。

db_name: 输入参数，指定备库名称。

返回值：

0: 符合自动 Recover 条件，备库已经在待恢复列表或者主库的守护进程即将发起对指定备库的 Recover 操作。

-20: 当前无法判断是否符合自动 Recover 条件，比如有监视器命令正在执行、指定备库还未达到恢复间隔或者指定备库上的 RLOG_PKG 还未全部重演完成等情况，需要稍后再次调用接口判断，具体的描述信息可通过消息相关接口获取到。

-21: 表示备库不满足自动 Recover 条件，具体的描述信息可通过消息相关接口获取

到。

其他<0 的值：表示接口执行失败。

84. dwmon_check_open

函数原型：

DWMON_RETURN

```
dwmon_check_open(
    mhandle      mhdle,
    mschar*      group_name,
    mschar*      db_name
);
```

功能说明：

检查指定组的指定库是否满足自动 Open 条件，如果不满足条件，守护进程不会通知数据库实例执行 Open 操作。

参数说明：

mhdle：输入参数，监视器操作句柄。

group_name：输入参数，指定库所属的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定数据库名称。

返回值：

0：符合自动 Open 条件，库已经 Open 成功或者正在执行 Open，或者库的守护进程即将通知实例执行 Open 操作。

-22：当前无法判断库是否满足自动 Open 条件，比如有监视器命令正在执行，或者数据库实例还未启动到 MOUNT 状态等情况，需要稍后再次调用接口判断，具体的描述信息可通过消息相关接口获取到。

-23：表示库不满足自动 Open 条件，并且库自身不是 OPEN 状态，具体的描述信息可通过消息相关接口获取到。

-24：表示库不满足自动 Open 条件，其中库实例已经是 OPEN 状态，但库的守护进程不是 OPEN 状态，具体的描述信息可通过消息相关接口获取到。

其他<0 的值：表示接口执行失败。

85. dwmon_deinit

函数原型:

```
DWMON_RETURN  
  
dwmon_deinit(  
  
    mhandle      mhdle  
  
);
```

功能说明:

销毁监视器执行环境。

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。

86. dwmon_free_handle

函数原型:

```
DWMON_RETURN  
  
dwmon_free_handle(  
  
    mhandle      mhdle  
  
);
```

功能说明:

释放分配到的监视器句柄。dwmon_free_handle 后需要用户将 mhdle 置空, 避免重复执行 dwmon_free_handle 引发异常。推荐使用 dwmon_free_handle_low 释放句柄。

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。

87. dwmon_get_monitor_state

函数原型:

DWMON_RETURN

```
dwmon_get_monitor_state(  
  
    mhandle      mhdle,  
  
    muint4*      n_mon,  
  
    mschar**     mon_name_arr,  
  
    muint4*      mon_state_arr,  
  
    muint4*      mon_id_arr,  
  
    muint4*      mon_role_arr,  
  
    mschar**     mon_ip_arr,  
  
    muint4*      mon_port_arr,  
  
    muint4*      mon_bro,  
  
    muint4*      mon_vote  
  
);
```

功能说明:

配置为主备多监视器环境时，可通过调用此接口获取监视器信息。

参数说明:

mhdle: 输入参数，监视器操作句柄。

n_mon: 输出参数，监视器个数。

mon_name_arr: 输出参数，monitor 名字数组，数组长度和 n_mon 对应。

mon_state_arr: 输出参数，monitor 的监视器状态数组，和 n_mon 对应。

mon_id_arr: 输出参数，monitorID 数组，和 n_mon 对应。

mon_role_arr: 输出参数，monitor 角色数组，和 n_mon 对应。

mon_ip_arr: 输出参数，monitor 的 IP 地址数组，和 n_mon 对应。

mon_port_arr: 输出参数，monitor 的端口号信息，和 n_mon 对应。

mon_bro: 输出参数，monitor 的广播延时。

mon_vote: 输出参数，monitor 的投票延时。

返回值：

0：执行成功。

<0：执行失败。

88. dwmon_get_ep_nth_apply_with_parallel_info**函数原型：**

DWMON_RETURN

dwmon_get_ep_nth_apply_with_parallel_info(

 mhandle mhdle,

 mschar* group_name,

 mschar* db_name,

 muint4 nth,

 mschar* prim_name,

 msint2* dsc_seqno,

 muint8* sseq,

 muint8* slsn,

 muint8* kseq,

 muint8* klsn,

 muint8* rseq,

 muint8* rlsn,

 msint2* n_pll,

 muint8* redo_lsn_arr,

 muint4* n_task,

 muint8* tsk_mem_used

);

功能说明：

获取指定备库上第 nth 组并行重演信息。

当备库对应的源库是 DMDSC 集群时，假如 DMDSC 集群有 N 个节点，那么备库上就会有 N 组重演信息依次和源库的每个节点相对应，N 值可以通过 dwmon_get_n_apply 接口

中的 `n_apply` 字段值获取，本接口的输入参数 `nth` 的取值范围为 `[0, n_apply-1]`。

参数说明：

`mhandle`：输入参数，监视器操作句柄。

`group_name`：输入参数，守护组名，只有一组时可以不指定。

`db_name`：输入参数，需要获取日志同步信息的备库名。

`nth`：输入参数，获取指定备库的第 `nth` 个重演状态统计信息。

`prim_name`：输出参数，第 `nth` 个重演状态统计信息上记录的备库对应主库的实例名，可能为空。

`dsc_seqno`：输出参数，主库节点序号，主库是单节点时序号为 0。

`sseq`：输出参数，备库可重演到的最大日志包序号。

`slsn`：输出参数，备库可重演到的最大 LSN。

`kseq`：输出参数，备库保持不重演的日志包序号。

`klsn`：输出参数，备库保持不重演的最大 LSN。

`rseq`：输出参数，备库已经重演到的日志包序号，和 `dsc_seqno` 对应。

`rlsn`：输出参数，备库已经重演到的 LSN 值。

`n_pll`：输出参数，备库并行重演线程数。

`redo_lsn_arr`：输出参数，备库各路并行重演线程已重演到的 LSN 值。

`n_task`：输出参数，当前还剩余的重演任务数。

`tsk_mem_used`：输出参数，当前任务队列占用的总内存大小。

返回值：

0：执行成功。

<0：执行失败。

89. dwmon_set_database_until_time

函数原型：

DWMON_RETURN

`dwmon_set_database_until_time(`

`mhandle` `mhandle,`

`mschar*` `group_name,`

```
mschar*      db_name,  
  
mschar*      until_time  
  
);
```

功能说明:

设置指定组的指定异步备库重演到的时间点。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

group_name: 输入参数, 指定库所属的守护进程组名, 只有一组时可以不指定。

db_name: 输入参数, 指定异步备库名称。

until_time: 输入参数, 指定异步备库重演到的时间点, 可精确到微秒。

返回值:

0: 执行成功。

<0: 执行失败。

90. dwmon_get_database_until_time

函数原型:

```
DWMON_RETURN  
  
dwmon_get_database_until_time(  
  
    mhandle      mhdle,  
  
    mschar*      group_name,  
  
    mschar*      db_name,  
  
    mschar*      until_time,  
  
    mbool*       apply_over  
  
);
```

功能说明:

获取指定组的指定异步备库设置的重演到的时间点。

参数说明:

mhdle: 输入参数, 监视器操作句柄。

group_name: 输入参数, 指定库所属的守护进程组名, 只有一组时可以不指定。

db_name: 输入参数, 指定异步备库名称。

until_time: 输出参数, 异步备库设置的重演到的时间点。

apply_over: 输出参数, 表示异步备库是否已重演到指定的 until_time 时间点, 0: 否, 1: 是。

返回值:

0: 执行成功。

<0: 执行失败。

91. dwmon_free_handle_low

函数原型:

```
DWMON_RETURN  
  
dwmon_free_handle_low(  
  
    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。

92. dwmon_get_takeover_time_by_group_name

函数原型:

```
DWMON_RETURN  
  
dwmon_get_takeover_time_by_group_name(  
  
    mhandle      mhdle,  
  
    mschar*      group_name,
```

```
    muint4*      remain_time,  
  
    mschar*      inst_name  
  
);
```

功能说明：

获取指定组的自动接管时间。

参数说明：

mhdle: 输入参数, 监视器操作句柄。

group_name: 输入参数, 指定库所属的守护进程组名, 只有一组时可以不指定。

remain_time: 输出参数, 自动接管时间。

inst_name: 输出参数, 接管实例名。

返回值：

0: 执行成功。

<0: 执行失败。

10.2.5 JNI 接口说明

1. dwmon_init_handle

函数原型：

```
boolean  
  
dwmon_init_handle();
```

功能说明：

分配监视器操作句柄。

参数说明：

无

返回值：

true: 分配成功。

false: 分配失败。

2. dwmon_get_error_msg_by_code

函数原型:

```
DwMonMsg  
  
dwmon_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 DwMonMsg 对象，可通过对象成员 returnCode 获取执行结果。

0: 执行成功，可通过 DwMonMsg 的其他对象成员获取输出消息。

<0: 执行失败，-17 表示没有找到 code 对应的错误信息。

3. dwmon_init_env

函数原型:

```
int  
  
dwmon_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 dmmonitor.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 dmmonitor.ini 中配置的 MON_LOG_PATH 作为日志文件路径，如果 dmmonitor.ini

中没有配置 MON_LOG_PATH，则将 dmmonitor.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，-15 表示监视器尝试建立到所有守护进程的连接失败，可能守护进程都未启动，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 dwmon_deinit_env 接口），再释放操作句柄（调用 dwmon_free_handle 接口）。

4. dwmon_msg_event_wait

函数原型：

```
void  
  
dwmon_msg_event_wait();
```

功能说明：

监视器检测到守护进程或实例状态发生变化，或者接口命令执行过程中，产生中间输出消息时会把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 dwmon_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. dwmon_get_exec_msg

函数原型：

```
DwMonMsg  
  
dwmon_get_exec_msg();
```

功能说明：

获取输出信息，和 dwmon_msg_event_wait 配合使用。

参数说明：

无

返回值：

返回 DwMonMsg 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 DwMonMsg 的其他对象成员获取输出消息，其中 msgFlag 值用来标记是否继续读取消息，为 1 表示还有消息可读，为 0 表示已全部读完。

<0：执行失败。

6. dwmon_get_msg_exit_flag

函数原型：

boolean

dwmon_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. dwmon_set_msg_exit_event

函数原型：

void

dwmon_set_msg_exit_event();

功能说明：

如果上层应用程序中有单独的消息线程，在 dwmon_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. dwmon_msg_event_deinit

函数原型：

```
void
```

```
dwmon_msg_event_deinit();
```

功能说明：

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 dwmon_msg_event_wait、dwmon_get_exec_msg、dwmon_get_msg_exit_flag、dwmon_set_msg_exit_event 和 dwmon_msg_event_deinit，这几个接口需要配合使用。

参数说明：

无

返回值：

无

9. dwmon_get_version

函数原型：

```
DwMonData
```

```
dwmon_get_version();
```

功能说明：

获取当前的监视器版本信息。

参数说明：

无

返回值：

返回 DwMonData 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过对象成员 stringValue 获取监视器版本信息。

<0：执行失败。

10. dwmon_get_mpp_flag**函数原型：**

DwMonData

dwmon_get_mpp_flag();

功能说明：

获取 MPP 主备环境标记，用于判断当前是否为 MPP 主备环境。

参数说明：

无

返回值：

返回 DwMonData 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过对象成员 dataValue 获取确认标记，1：是 MPP 主备环境， 0：不是 MPP 主备环境。

<0：执行失败。

11. dwmon_get_n_group**函数原型：**

DwMonGrpNameArray

dwmon_get_n_group();

功能说明：

获取当前监控的组个数和组名称。只有 mpp 主备环境下允许配置多组。

参数说明：

无

返回值：

返回 DwMonGrpNameArray 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 DwMonGrpNameArray 的其他对象成员获取监控组信息。

<0：执行失败。

12. dwmon_get_group_info

函数原型：

```
DwMonGroup  
  
dwmon_get_group_info(  
  
    String          group_name  
  
);
```

功能说明：

获取指定组的守护进程组信息。

参数说明：

group_name：输入参数，指定要获取信息的组名，只有一组时可以不指定。

返回值：

返回 DwMonGroup 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 DwMonGroup 的其他对象成员获取监控组信息。

<0：执行失败。

13. dwmon_get_split_sub_group_database

函数原型：

```
DwMonDatabaseNameArray  
  
dwmon_get_split_sub_group_database(  
  
    String          group_name,  
  
    String          db_name  
  
);
```

功能说明：

获取可加入指定库的库名信息。

参数说明：

group_name：输入参数，指定实例所在的组名，只有一组时可以不指定。

db_name：输入参数，指定要获取可加入信息的实例名，此参数必须指定。

返回值：

返回 DwMonDatabaseNameArray 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 DwMonDatabaseNameArray 的其他对象成员获取实例信息。

<0：执行失败。

14. dwmon_get_nth_database_info

函数原型：

```
DwMonDatabase  
  
dwmon_get_nth_database_info(  
  
    String          group_name,  
  
    int             nth  
  
);
```

功能说明：

获取指定组中第 nth 个库的全局信息，这里的 nth 是根据 dmmonitor.ini 指定组中所配置的守护进程先后顺序来排序的，每个组的 nth 都是从 0 开始，依次递增。

参数说明：

group_name：输入参数，指定的守护进程组名，只有一组时可以不指定。

nth：输入参数，输入要获取的数据库实例下标。

返回值：

返回 DwMonDatabase 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 DwMonDatabase 的其他对象成员获取实例信息。

<0：执行失败。

15. dwmon_get_database_info_by_name

函数原型:

DwMonDatabase

```
dwmon_get_database_info_by_name(  
    String          group_name,  
    String          db_name  
);
```

功能说明:

根据指定的数据库名称获取库的全局信息。

参数说明:

group_name: 输入参数, 指定的守护进程组名, 只有一组时可以不指定。

db_name: 输入参数, 指定要获取信息的数据库名称。

返回值:

返回 DwMonDatabase 对象, 可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过 DwMonDatabase 的其他对象成员获取实例信息。

<0: 执行失败。

16. dwmon_get_database_info

函数原型:

DwDbInfo

```
dwmon_get_database_info(  
    String          group_name,  
    String          db_name  
);
```

功能说明:

获取指定库上的实例信息。

参数说明:

group_name: 输入参数, 输入指定库所在的守护进程组名, 如果只有一组可以不指

定。

db_name: 输入参数, 指定要获取实例信息的数据库名称。

返回值:

返回 DwDbInfo 对象, 可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过 DwDbInfo 的其他对象成员获取实例信息。

<0: 执行失败。

17. dwmon_get_nth_ep_info

函数原型:

```
DwEpInfo  
  
dwmon_get_nth_ep_info(  
    String      group_name,  
    String      db_name,  
    int         nth  
);
```

功能说明:

获取指定库上第 nth 个实例信息, 可先通过 dwmon_get_database_info 接口获取到库上的节点实例个数 n_ep, nth 取值范围为[0, n_ep-1]。

参数说明:

group_name: 输入参数, 输入指定库所在的守护进程组名, 如果只有一组可以不指定。

db_name: 输入参数, 指定要获取实例信息的数据库名称, 此参数必须指定。

nth: 输入参数, 指定要获取 db_name 库上的第 nth 个实例信息。

返回值:

返回 DwEpInfo 对象, 可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过 DwEpInfo 的其他对象成员获取实例信息。

<0: 执行失败。

18. dwmon_get_ep_info_by_name

函数原型:

```
DwEpInfo  
  
dwmon_get_ep_info_by_name(  
    String          group_name,  
    String          ep_name  
);
```

功能说明:

根据指定的实例名获取实例的详细信息。

参数说明:

group_name: 输入参数, 输入指定实例所在的守护进程组名, 如果只有一组可以不指定。

ep_name: 输入参数, 输入要获取信息的实例名。

返回值:

返回 DwEpInfo 对象, 可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过 DwEpInfo 的其他对象成员获取实例信息。

<0: 执行失败。

19. dwmon_get_database_arch_info

函数原型:

```
DwMonArch  
  
dwmon_get_database_arch_info(  
    String          group_name,  
    String          db_name  
);
```

功能说明:

获取指定库配置的归档信息, 包括归档目标实例名, 到归档目标的 mal 链路状态和归档状态, 以及异步归档信息。

参数说明：

group_name：输入参数，输入指定库所在的守护进程组名，如果只有一组可以不指定。

db_name：输入参数，指定要获取归档信息的数据库名称。

返回值：

返回 DwMonArch 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 DwMonArch 的其他对象成员获取相关归档信息。

<0：执行失败。

20. dwmon_get_watch_ctl_info

函数原型：

```
DwCtlInfo  
  
dwmon_get_watch_ctl_info(  
    String      group_name,  
    String      db_name  
);
```

功能说明：

获取指定库的守护进程控制文件信息，包括控制文件中的记录项个数，控制文件状态和当前 TGUID 值这些全局信息。

参数说明：

group_name：输入参数，输入指定库所在的守护进程组名，如果只有一组可以不指定。

db_name：输入参数，指定要获取控制文件信息的守护进程对应的数据库名称。

返回值：

返回 DwCtlInfo 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 DwCtlInfo 的其他对象成员获取相关控制文件信息。

<0：执行失败。

21. dwmon_get_database_open_history

函数原型:

```
DwOpenHistory  
  
dwmon_get_database_open_history(  
    String          group_name,  
    String          db_name  
);
```

功能说明:

获取指定库的 OPEN 记录项个数。

参数说明:

group_name: 输入参数, 输入指定库所在的守护进程组名, 如果只有一组可以不指定。

db_name: 输入参数, 指定要获取 OPEN 记录的守护进程对应的数据库名称。

返回值:

返回 DwOpenHistory 对象, 可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过 DwOpenHistory 的其他对象成员获取相关 OPEN 记录信息。

<0: 执行失败。

22. dwmon_get_nth_open_history

函数原型:

```
DwOpenHistory  
  
dwmon_get_nth_open_history(  
    String          group_name,  
    String          db_name,  
    int             nth  
);
```

功能说明:

获取指定组中指定库的第 nth 个 open 记录信息。

参数说明：

group_name：输入参数，输入指定库所在的守护进程组名，如果只有一组可以不指定。

db_name：输入参数，指定要获取 OPEN 记录的守护进程对应的数据库名称。

nth：输入参数，输入要获取指定第 nth 个 OPEN 记录项。

返回值：

返回 DwOpenHistory 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 DwOpenHistory 的其他对象成员获取相关 OPEN 记录信息。

<0：执行失败。

23. dwmon_get_watch_config_info

函数原型：

```
DwConfigInfo  
  
dwmon_get_watch_config_info(  
  
    String      group_name,  
  
    String      inst_name  
  
);
```

功能说明：

获取指定实例本地的守护进程配置信息。

参数说明：

group_name：输入参数，只有一组时允许不指定。

inst_name：输入参数，指定要获取配置信息的守护进程本地的实例名。

返回值：

返回 DwConfigInfo 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 DwConfigInfo 的其他对象成员获取配置信息。

<0：执行失败。

24. dwmon_get_css_config_info

函数原型:

```
DcrConfigInfo  
  
dwmon_get_css_config_info(  
    String      group_name,  
    String      inst_name  
);
```

功能说明:

获取和指定实例的 dcr_seqno 相同的 dmcss 配置的自动拉起检测间隔 (DMDCR_ASM_RESTART_INTERVAL、DMDCR_DB_RESTART_INTERVAL)。

参数说明:

group_name: 输入参数, 只有一组时允许不指定。

inst_name: 输入参数, 指定要获取配置信息的实例名。

返回值:

返回 DcrConfigInfo 对象, 可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过 DcrConfigInfo 的其他对象成员获取配置信息。

<0: 执行失败。

25. dwmon_get_monitor_info

函数原型:

```
DwMonInfo  
  
dwmon_get_monitor_info (  
    String      group_name,  
    String      db_name  
);
```

功能说明:

获取指定守护进程的监视器连接信息。同一个守护系统中允许最多 10 个监视器同时启

动连接到守护进程，可通过调用此接口获取指定守护进程上的监视器连接信息，包括连接时间、确认监视器配置、监视器 ID、监视器 IP 和监视器版本信息。

其中 `db_name` 是输入输出参数，表示指定守护进程对应的实例名，如果此参数为空，则根据 `dmmonitor.ini` 对应组中的配置顺序，选择第一个活动的守护进程获取监视器连接信息，并输出选中的守护进程本地库名称；如果指定有合法值，则返回指定库的控制守护进程上的监视器连接信息。

参数说明：

`group_name`：输入参数，输入指定库所在的守护进程组名，如果只有一组可以不指定。

`db_name`：输入参数，指定要获取信息的守护进程对应的实例名。

返回值：

返回 `DwMonInfo` 对象，可通过对象成员 `returnCode` 获取执行结果。

0：执行成功，可通过 `DwMonInfo` 的其他对象成员获取记录项信息，取出的第一个数组元素是当前 `monitor` 的连接信息。

<0：执行失败。

26. `dwmon_set_cmd_execing_with_command`

函数原型：

```
int  
  
dwmon_set_cmd_execing_with_command(  
  
    String          command  
  
);
```

功能说明：

如果上层有控制台工具界面，通过用户输入命令的方式调用接口，可通过此接口设置命令执行标记并传入用户执行的命令串信息，用于监视器 log 日志记录用户的操作信息。在确认监视器模式下或者监视器设置有自动记录系统状态间隔（通过 `dwmon_set_show_interval` 或者 `dwmon_set_log_interval` 设置），调用命令接口之前，需要先调用此接口设置执行标记，避免和确认监视器自动接管故障主库或者监视器自动记录系统状态的操作发生冲突。

参数说明：

command：输入参数，如果上层有控制台工具界面，则表示用户手工输入的执行命令串，允许为 NULL 或空串。

返回值：

0：执行成功。
<0：执行失败。

27. dwmon_reset_cmd_execing

函数原型：

```
int  
dwmon_reset_cmd_execing();
```

功能说明：

重置命令执行标记，和 dwmon_set_cmd_execing_with_command 接口配合使用，在调用命令接口执行完成后，需要调用此接口，重置执行标记，避免确认监视器无法执行自动接管故障主库的操作。

参数说明：

无

返回值：

0：执行成功。
<0：执行失败。

28. dwmon_get_dw_confirm

函数原型：

```
DwMonData  
dwmon_get_dw_confirm();
```

功能说明：

获取确认监视器标记，用于判断当前的监视器是否为确认监视器。

参数说明：

无

返回值：

返回 DwMonData 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过对象成员 dataValue 获取确认标记，1：是确认监视器， 0：不是确认监视器。

<0：执行失败。

29. dwmon_set_show_interval

函数原型：

```
int  
  
dwmon_set_show_interval(  
  
    int                show_interval  
  
);
```

功能说明：

监视器根据指定的 show_interval 值，每隔 show_interval 个时间间隔，自动输出各实例的状态信息给用户，需要使用 dwmon_get_exec_msg 接口获取到输出消息，show_interval 单位为秒，默认值为 0。

参数说明：

show_interval：输入参数，设置定时输出状态信息的时间间隔，单位为秒，取值为 0，或 5~3600，取值为 0 表示取消自动输出，默认值为 0。

返回值：

0：执行成功。

<0：执行失败。

30. dwmon_get_show_interval

函数原型：

```
DwMonData  
  
dwmon_get_show_interval();
```

功能说明:

获取当前设置的自动显示时间间隔，单位为秒。

参数说明:

无

返回值:

返回 DwMonData 对象，可通过对象成员 returnCode 获取执行结果。

0：获取成功，可通过对象成员 dataValue 获取自动显示间隔。

<0：获取失败。

31. dwmon_set_log_interval

函数原型:

```
int  
  
dwmon_set_log_interval(  
  
    int          log_interval  
  
);
```

功能说明:

监视器按照设置的 log_interval 值，每隔 log_interval 个时间间隔，自动将各实例的状态信息写入到日志文件，log_interval 单位为秒，默认值为 1。

参数说明:

log_interval：输入参数，设置定时写入状态信息到日志文件的时间间隔，单位为秒，取值为 0，1 或 5~3600。

取值为 0 表示不写入任何信息到日志文件。

取值为 1 表示只写入普通的状态变化日志，例如收到新的信息或者检测到状态变化等。

取值为 5~3600 范围内的数值时，除了生成普通日志信息外，会按照设置的间隔定期写入站点状态信息到日志文件中。

返回值:

0：执行成功。

<0：执行失败。

32. dwmon_get_log_interval

函数原型:

```
DwMonData  
  
dwmon_get_log_interval();
```

功能说明:

获取当前设置的自动写入日志的时间间隔，单位为秒。

参数说明:

无

返回值:

返回 DwMonData 对象，可通过对象成员 returnCode 获取执行结果。

0：获取成功，可通过对象成员 dataValue 获取自动写入日志文件间隔。

<0：获取失败。

33. dwmon_set_log_file_size

函数原型:

```
int  
  
dwmon_set_log_file_size(  
  
    int                log_file_size  
  
);
```

功能说明:

调用此接口，可以设置监视器单个的日志文件大小，达到最大值后，会自动生成并切换到新的日志文件中，如果达到最大的日志总空间限制，会自动删除创建时间最早的日志文件，日志文件命名方式为“dmmonitor_年月日时分秒.log”，例如“dmmonitor_20160201155820.log”。

参数说明:

log_file_size：输入参数，设置单个日志文件大小，单位为 M，取值范围为 16~2048，默认值为 64。

返回值:

0: 执行成功。

<0: 执行失败。

34. dwmon_get_log_file_size

函数原型:

DwMonData

```
dwmon_get_log_file_size();
```

功能说明:

获取当前设置的单个日志文件大小，单位为 M。

参数说明:

无

返回值:

返回 DwMonData 对象，可通过对象成员 returnCode 获取执行结果。

0: 获取成功，可通过对象成员 dataValue 获取单个日志文件大小。

<0: 获取失败。

35. dwmon_get_log_path

函数原型:

DwMonData

```
dwmon_get_log_path();
```

功能说明:

获取当前的日志文件路径。

参数说明:

无

返回值:

返回 DwMonData 对象，可通过对象成员 returnCode 获取执行结果。

0: 执行成功，可通过对象成员 stringValue 获取日志文件路径。

<0: 执行失败。

36. dwmon_set_log_space_limit

函数原型:

```
int  
  
dwmon_set_log_space_limit(  
  
    int                log_space_limit  
  
);
```

功能说明:

调用此接口，可以设置监视器的日志总空间大小，达到设置的总空间大小后，会自动删除创建时间最早的日志文件。

参数说明:

log_space_limit: 输入参数，设置日志总空间大小，单位为 M，取值为 0 或 256~4096，默认值为 0，表示没有空间限制。

返回值:

0: 执行成功。
<0: 执行失败。

37. dwmon_get_log_space_limit

函数原型:

```
DwMonData  
  
dwmon_get_log_space_limit();
```

功能说明:

获取当前设置的日志总空间大小，单位为 M。

参数说明:

无

无返回值:

返回 DwMonData 对象，可通过对象成员 returnCode 获取执行结果。

0: 执行成功，可通过对象成员 dataValue 获取当前设置的日志总空间大小。
<0: 执行失败。

38. dwmon_login

函数原型:

```
int  
  
dwmon_login(  
  
    String      username,  
  
    String      password  
  
);
```

功能说明:

登录监视器，这里的登录口令和服务器的登录口令是一致的，且必须有 DBA 权限。

校验登录信息时由守护进程转发给服务器校验，并返回校验结果给监视器，因此需要保证系统中至少有一个活动库，否则无法校验成功。

参数说明:

username: 输入参数，登录用户名。

password: 输入参数，登录密码。

返回值:

0: 执行成功。

<0: 执行失败。

39. dwmon_logout

函数原型:

```
int  
  
dwmon_logout();
```

功能说明:

退出登录监视器。

参数说明:

无

返回值:

0: 执行成功。

<0: 执行失败。

40. dwmon_set_database_recover_time

函数原型:

```
int  
  
dwmon_set_database_recover_time(  
  
    String      group_name,  
  
    String      db_name,  
  
    int         recover_time  
  
);
```

功能说明:

设置指定备库的恢复间隔时间, 设置成功后, `recover_time` 值保存在对应主库的守护进程内存中, 并且只对指定的备库起作用。

此接口只允许对指定备库执行修改操作, 如果需要考虑对某个组或所有组中的备库执行修改操作, 则可以使用 `dwmon_set_group_recover_time` 接口来实现。

参数说明:

`group_name`: 输入参数, 指定执行操作的守护进程组名, 只有一组时可以不指定组名。

`db_name`: 输入参数, 指定要设置恢复间隔时间的备库实例名。

`recover_time`: 输入参数, 指定要设置的恢复间隔时间, 取值 3~ 86400 s, 设置为 0 时, 表示指定备库不需要有时间间隔限制, 满足恢复条件后, 主库的守护进程就会启动恢复流程, 详细说明可参考 [3.1.9 故障恢复处理](#)。

返回值:

0: 执行成功。

<0: 执行失败。

41. dwmon_set_database_arch_invalid

函数原型:

```
int
dwmon_set_database_arch_invalid(
    String      group_name,
    String      db_name
);
```

功能说明：

设置主库到指定备库的归档状态无效。

此接口只允许对指定备库设置归档无效,如果需要考虑对某个组或所有组中的备库执行设置操作,则可以使用 dwmon_set_group_arch_invalid 接口来实现。

参数说明：

group_name: 输入参数,指定执行操作的守护进程组名,只有一组时可以不指定组名。

db_name: 输入参数,指定要修改归档状态无效的备库实例名。

返回值：

0: 执行成功。

<0: 执行失败。

42. dwmon_startup_database_watch

函数原型：

```
int
dwmon_startup_database_watch(
    String      group_name,
    String      db_name
);
```

功能说明：

打开指定库的守护进程监控功能,守护进程启动后,默认监控功能处于打开状态。

对 DMDSC 集群,此接口只通知控制守护进程执行。

参数说明：

group_name: 输入参数,指定要打开监控的守护进程组名,只有一组时可以不指定

组名。

db_name: 输入参数, 指定要打开监控的守护进程的本地实例名。

返回值:

>=0: 执行成功, 执行前守护进程的监控已经处于打开状态时, 返回值为 100。

<0: 执行失败。

43. dwmon_startup_watch

函数原型:

```
int  
  
dwmon_startup_watch(  
  
    String          group_name  
  
) ;
```

功能说明:

打开指定组的守护进程监控功能, 守护进程启动后, 默认监控功能处于打开状态。

对 DMDSC 集群, 此接口只通知控制守护进程执行。

参数说明:

group_name: 输入参数, 指定要打开监控的守护进程组名, 只有一组时可以不指定组名。

返回值:

>=0: 执行成功, 执行前组中所有守护进程的监控已经处于打开状态时, 返回值为 100。

<0: 执行失败。

44. dwmon_startup_watch_all

函数原型:

```
int  
  
dwmon_startup_watch_all();
```

功能说明:

打开所有组的守护进程监控功能, 守护进程启动后, 默认监控功能处于打开状态。

只有 MPP 主备环境下允许调用此接口。

参数说明：

无返回值：

>=0：执行成功，执行前所有组守护进程的监控已经处于打开状态时，返回值为 100。

<0：执行失败。

45. dwmon_stop_database_watch

函数原型：

```
int  
  
dwmon_stop_database_watch(  
  
    String          group_name,  
  
    String          db_name  
  
);
```

功能说明：

关闭指定库的守护进程的监控功能，监控功能被关闭后，守护进程还可以正常地接收和发送消息，但无法根据当前的实例状态来自动处理故障以及实例的自动拉起等，关闭后守护进程处于 Shutdown 状态。

如果需要正常地退出实例或者保持实例为 Mount 状态，则可以关闭监控功能，否则建议保持监控功能为打开状态，以保证能够及时处理系统的各种情况。

对 DMDSC 集群，此接口只通知控制守护进程执行。

执行此操作需要先登录监视器。

参数说明：

group_name：输入参数，指定要关闭监控的守护进程组名，只有一组时可以不指定组名。

db_name：输入参数，指定要关闭监控的守护进程的本地实例名。

返回值：

>=0：守护进程的监控关闭成功。执行前守护进程的监控已经处于关闭状态时，返回值为 100。

<0：执行失败

46. dwmon_stop_watch

函数原型:

```
int  
  
dwmon_stop_watch(  
  
    String          group_name  
  
);
```

功能说明:

关闭指定组的所有守护进程的监控功能，监控功能被关闭后，守护进程还可以正常地接收和发送消息，但无法根据当前的实例状态来自动处理故障以及实例的自动拉起等，关闭后守护进程处于 Shutdown 状态。

如果需要正常地退出实例或者保持实例为 Mount 状态，则可以关闭监控功能，否则建议保持监控功能为打开状态，以保证能够及时处理系统的各种情况。

对 DMDSC 集群，此接口只通知控制守护进程执行。

执行此操作需要先登录监视器。

参数说明:

group_name: 输入参数，指定要关闭监控的守护进程组名，只有一组时可以不指定组名。

返回值:

>=0: 所有守护进程的监控关闭成功。执行前组中所有守护进程的监控已经处于关闭状态时，返回值为 100。

<0: 执行失败

47. dwmon_stop_watch_all

函数原型:

```
int  
  
dwmon_stop_watch_all();
```

功能说明:

关闭所有组的守护进程监控功能，监控功能被关闭后，守护进程还可以正常地接收和发

送消息，但无法根据当前的实例状态来自动处理故障以及实例的自动拉起等，关闭后守护进程处于 Shutdown 状态。

如果需要正常地退出实例或者保持实例为 Mount 状态，则可以关闭监控功能，否则建议保持监控功能为打开状态，以保证能够及时处理系统的各种情况。

执行此操作需要先登录监视器，只有 MPP 主备环境下允许调用此接口。

参数说明：

无

返回值：

>=0：所有守护进程的监控关闭成功，执行前所有组守护进程的监控已经处于关闭状态时，返回值为 100。

<0：执行失败。

48. dwmon_startup_group

函数原型：

```
int  
  
dwmon_startup_group(  
  
    String          group_name  
  
);
```

功能说明：

启动指定组的所有实例，在 dmwatcher.ini 中 INST_AUTO_RESTART 配置为 0 时该接口可以成功执行，配置为 1 时在守护进程的监控功能打开的情况下，守护进程可以自动完成启动实例的功能，不需要再调用接口执行。

对 DMDSC 集群，由 dmcss 执行拉起动作，和 dmwatcher.ini 中的 INST_AUTO_RESTART 配置参数没有关系，由 dmdcr.ini 中的配置参数决定是否可以执行自动拉起，如果某个实例可以被 dmcss 自动拉起，则 dmcss 会忽略收到的拉起命令，等待自动拉起。

参数说明：

group_name：输入参数，指定要启动实例的守护进程组名，只有一组时可以不指定。

返回值：

≥ 0 : 所有实例都启动成功, 执行前组中所有实例已经处于活动状态时, 返回值为 100。

< 0 : 执行失败。

49. dwmon_startup_group_all

函数原型:

```
int  
  
dwmon_startup_group_all();
```

功能说明:

启动所有组的实例, 在 dmwatcher.ini 中 INST_AUTO_RESTART 配置为 0 时该接口可以成功执行, 配置为 1 时在守护进程的监控功能打开的情况下, 守护进程可以自动完成启动实例的功能, 不需要再调用接口执行。

只有 MPP 主备环境下允许调用此接口。

参数说明:

无返回值:

≥ 0 : 所有实例都启动成功, 执行前所有组的实例已经处于活动状态时, 返回值为 100。

< 0 : 执行失败。

50. dwmon_stop_group

函数原型:

```
int  
  
dwmon_stop_group(  
  
    String          group_name  
  
);
```

功能说明:

关闭指定组中的所有活动实例, 关闭所有活动实例之前, 会先关闭所有控制守护进程的监控功能, 避免守护进程将实例再次拉起。

对 DMDSC 集群, 是由 dmcss 执行退出操作, 在 dmcss 通知所有实例退出之前, 会先关闭 dmcss 的自动拉起功能。

执行此操作需要先登录监视器，另外关闭守护进程只是将守护进程切换为 Shutdown 状态，并没有退出。

参数说明：

group_name：输入参数，指定要关闭实例的守护进程组名，只有一组时可以不指定。

返回值：

>=0：所有实例都退出成功，执行前组中实例已经全部退出时，返回值为 100。

<0：执行失败。

51. dwmon_stop_group_all

函数原型：

```
int  
  
dwmon_stop_group_all();
```

功能说明：

关闭所有组的所有活动实例，关闭实例之前，会先关闭所有守护进程的监控功能，避免守护进程将实例再次拉起。

只有 MPP 主备环境下允许调用此接口，执行此操作需要先登录监视器，另外关闭守护进程只是将守护进程切换为 Shutdown 状态，并没有退出。

参数说明：

无

返回值：

>=0：所有实例都退出成功，执行前所有组的实例已经全部退出时，返回值为 100。

<0：执行失败。

52. dwmon_kill_group

函数原型：

```
int  
  
dwmon_kill_group(  
  
    String          group_name
```

```
);
```

功能说明：

强制杀掉指定组中的活动实例。

调用 `dwmon_stop_group` 关闭指定组的实例时，如果实例正在执行守护进程发起的命令，或者守护进程不是 STARTUP、OPEN 或 RECOVERY 状态，会导致接口执行失败，在这种情况下，如果需要强制关闭实例，则可以通过调用 `dwmon_kill_group` 接口来完成。

此接口执行时不要求先登录监视器，并且也会将守护进程切换为 Shutdown 状态（如果已经是 Shutdown 状态则跳过不再切换），避免守护进程将实例重新拉起。

参数说明：

`group_name`：输入参数，指定要强杀实例的守护进程组名，只有一组时可以不指定。

返回值：

≥ 0 ：所有实例都退出成功，执行前组中实例已经全部退出并且守护进程都是 Shutdown 状态时返回值为 100。

< 0 ：执行失败。

53. `dwmon_kill_group_all`

函数原型：

```
int  
  
dwmon_kill_group_all();
```

功能说明：

强制杀掉所有组中的活动实例。

调用 `dwmon_stop_group_all` 关闭所有组的实例时，如果实例正在执行守护进程发起的命令，或者守护进程不是 STARTUP、OPEN 或 RECOVERY 状态，会导致接口执行失败，在这种情况下，如果需要强制关闭实例，则可以通过调用 `dwmon_kill_group_all` 接口来完成。

只有 MPP 主备环境下允许调用此接口，此接口执行时不要求先登录监视器，并且也会将守护进程切换为 Shutdown 状态（如果已经是 Shutdown 状态则跳过不再切换），避免守护进程将实例重新拉起。

参数说明：

无

返回值:

≥ 0 : 所有实例都退出成功, 执行前所有组的实例已经全部退出并且守护进程都是 Shutdown 状态则返回值为 100。

< 0 : 执行失败。

54. dwmon_startup_database

函数原型:

```
int  
  
dwmon_startup_database(  
  
    String          group_name,  
  
    String          db_name  
  
);
```

功能说明:

启动指定组中指定的数据库实例。

如果指定的库是单机, 则由守护进程执行拉起动作, 如果指定的库是 DMDSC 集群, 则由守护进程再去通知主从 css 各自执行拉起动作。

此命令执行前, 如果守护进程处于 Shutdown 状态, 会将其切换为 Startup 状态, 如果主从 css 的自动拉起功能是关闭的, 在各节点实例拉起成功后, 会将其打开。

参数说明:

group_name: 输入参数, 指定要启动的库所在的组名, 只有一组时可以不指定。

db_name: 输入参数, 指定要启动的数据库名称, 此参数必须指定。

返回值:

≥ 0 : 执行成功, 执行前库已经处于活动状态时返回值为 100。

< 0 : 执行失败。

55. dwmon_stop_database

函数原型:

```
int  
  
dwmon_stop_database(  
  
    String          group_name,  
  
    String          db_name  
  
);
```

功能说明:

关闭指定组中指定的数据库实例。

如果指定的库是单机，则由守护进程通知单机库正常退出，如果指定的库是 DMDSC 集群，则由守护进程再去通知主 css 依次退出每个节点实例。

此命令执行完成后，会将守护进程切换为 Shutdown 状态，并将主从 css 的自动拉起功能关闭。

注意:

如果要退出的是全局守护类型的备库，为了避免触发主库的 Failover 动作，要求此备库已经被 detach 分离出去，也就是备库归档要处于无效状态，才允许将其正常退出。如果要退出的是主库，在自动切换模式下，为了避免触发备库的自动接管，要求所有备库都已经被 detach 分离出去，也就是所有备库归档都处于无效状态，才允许退出主库，手动切换模式下没有此要求。

参数说明:

group_name: 输入参数，指定要关闭的库所在的组名，只有一组时可以不指定。

db_name: 输入参数，指定要关闭的数据库名称，此参数必须指定。

返回值:

>=0: 执行成功，执行前库已经退出时返回值为 100。

<0: 执行失败。

56. dwmon_kill_database

函数原型:

```
int  
  
dwmon_kill_database(  
  
    String          group_name,
```

```
String      db_name
);
```

功能说明：

强制关闭指定组中指定的数据库实例。

如果指定的库是单机，则由守护进程强制杀掉指定库的进程，如果指定的库是 DMDSC 集群，则由守护进程再去通知主 css 强杀所有节点实例。

注意：此命令执行完成后，会将守护进程切换为 Shutdown 状态。

参数说明：

group_name：输入参数，指定要强制关闭的库所在的组名，只有一组时可以不指定。

db_name：输入参数，指定要强制关闭的数据库名称，此参数必须指定。

返回值：

>=0：执行成功，执行前库已经不是活动状态时返回值为 100。

<0：执行失败。

57. dwmon_open_database

函数原型：

```
int
dwmon_open_database(
    String      group_name,
    String      db_name
);
```

功能说明：

在系统中存在其他故障实例时，守护进程无法自动 Open 主库，或者系统中没有活动主库时，守护进程无法自动 Open 备库，这种情况下，可以调用此接口强制 Open 指定的数据库。

执行此操作需要先登录监视器。

参数说明：

group_name：输入参数，指定要强制 Open 库所在的组名，只有一组时可以不指定。

db_name：输入参数，指定要强制 Open 的数据库名称，此参数必须指定。

返回值：

>=0：执行成功，执行前库已经处于 Open 状态时返回值为 100。

<0：执行失败。

58. dwmon_get_switchover_standby

函数原型：

```
DwMonDatabaseNameArray  
  
dwmon_get_switchover_standby(  
  
    String          group_name  
  
);
```

功能说明：

获取可以切换为主库的备库信息，这里的切换是指主备库都正常，且处于 Open 状态时，交换主备库模式，将备库切换为新的主库，主库切换为备库的操作。

参数说明：

group_name：输入参数，指定要获取切换信息的组名，只有一组时可以不指定。

返回值：

返回 DwMonDatabaseNameArray 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 DwMonDatabaseNameArray 的其他对象成员获取备库实例信息。

<0：执行失败。

59. dwmon_switchover

函数原型：

```
int  
  
dwmon_switchover(  
  
    String          group_name,  
  
    String          db_name  
  
);
```

功能说明：

使用指定的备库和当前的主库执行切换操作，这里的切换是指主备库都正常，且处于 Open 状态时，交换主备库模式，将备库切换为新的主库，主库切换为备库的操作。

此接口只有在指定的备库满足切换条件时才能够执行成功，可以先调用 `dwmon_get_switchover_standby` 获取到满足条件的备库列表。

执行此操作需要先登录监视器。

参数说明：

`group_name`：输入参数，指定要执行切换的备库所在的组名，只有一组时可以不指定。

`db_name`：输入参数，指定要执行切换的备库名称，只有一个备库时可以不指定。

返回值：

0：执行成功。

<0：执行失败。

60. `dwmon_get_takeover_standby`

函数原型：

```
DwMonDatabaseNameArray  
  
dwmon_get_takeover_standby(  
  
    String          group_name,  
  
    boolean         force_flag  
  
);
```

功能说明：

获取可以接管为主库的备库信息，在主库发生故障时，可以通过调用此接口查找可以接管为新的主库的备库实例，其中包括正常接管和强制接管，通过输入参数 `force_flag` 可以指定接管方式。

参数说明：

`group_name`：输入参数，指定要获取接管信息的组名，只有一组时可以不指定。

`force_flag`：输入参数，指定接管方式，`true`：强制接管，`false`：正常接管。

返回值：

返回 `DwMonDatabaseNameArray` 对象，可通过对象成员 `returnCode` 获取执行结果。

0: 执行成功, 可通过 `DwMonDatabaseNameArray` 的其他对象成员获取备库实例信息。

<0: 执行失败。

61. dwmon_takeover

函数原型:

```
int  
  
dwmon_takeover(  
  
    String          group_name,  
  
    String          db_name  
  
);
```

功能说明:

主库故障后, 可以通过调用此接口执行正常的接管操作, 注意, 此接口只有在指定备库满足正常接管条件时才能够执行成功, 可以先调用 `dwmon_get_takeover_standby` 获取到符合正常接管条件的备库实例, `force_flag` 指定为 `false`。

执行此操作需要先登录监视器。

参数说明:

`group_name`: 输入参数, 指定要执行接管的备库所在的组名, 只有一组时可以不指定。

`db_name`: 输入参数, 指定要执行接管的备库名称, 只有一个备库时可以不指定。

返回值:

0: 执行成功。

<0: 执行失败。

62. dwmon_takeover_force

函数原型:

```
int  
  
dwmon_takeover_force(  
  
    String          group_name,
```

```
String      db_name
);
```

功能说明：

主库故障后，可以通过调用此接口执行强制接管操作。此接口只有在指定备库满足强制接管条件时才能够执行成功，可以先调用 `dwmon_get_takeover_standby` 获取到符合强制接管条件的备库实例，`force_flag` 指定为 `TRUE`。

执行此操作需要先登录监视器。

参数说明：

`group_name`：输入参数，指定要执行强制接管的备库所在的组名，只有一组时可以不指定。

`db_name`：输入参数，指定要执行强制接管的备库名称，只有一个备库时可以不指定。

返回值：

0：执行成功。

<0：执行失败。

63. `dwmon_check_mppctl`

函数原型：

```
int
dwmon_check_mppctl();
```

功能说明：

配置为 mpp 主备环境时，可通过调用此接口检查当前各个活动节点的 `dmmpp.ctl` 是否处于一致状态，注意只有 MPP 主备环境下允许调用此接口。

参数说明：

无

返回值：

0：处于一致状态。

<0：检查失败，或者控制文件不一致 (返回值为 -12)。

64. dwmon_recover_mppctl

函数原型:

```
int  
  
dwmon_recover_mppctl();
```

功能说明:

配置为 mpp 主备环境下时，如果某个活动节点的 dmmppctl 不一致，可通过调用此接口恢复到一致状态。

只有 MPP 主备环境下允许调用此接口，并且需要先登录监视器。

参数说明:

无

返回值:

0: 执行成功。

<0: 执行失败。

65. dwmon_get_mpp_site_info

函数原型:

```
DwMonMpp  
  
dwmon_get_mpp_site_info();
```

功能说明:

配置为 mpp 主备环境时，可通过调用此接口获取 mpp 节点信息，注意只有 MPP 主备环境下允许调用此接口。

参数说明:

无

返回值:

返回 DwMonMpp 对象，可通过对象成员 returnCode 获取执行结果。

0: 执行成功，可通过 DwMonMpp 其他对象成员获取 MPP 节点信息。

<0: 执行失败。

66. dwmon_tip

函数原型:

```
int  
  
dwmon_tip();
```

功能说明:

可调用此接口查看系统当前运行状态，输出的系统状态信息可通过消息相关接口获取到。

参数说明:

无

返回值:

0：系统运行正常，所有实例和守护进程都处于 OPEN 状态。

-25：所有组中都有正常的主库实例（实例和守护进程都是 OPEN 状态），不影响系统使用，但是存在有其他异常实例，具体描述信息可通过消息相关接口获取。

-26：系统运行异常，不是所有组都有正常的主库实例，守护进程正在自动处理，请稍后再次调用接口判断，具体描述信息可通过消息相关接口获取。

-27：系统运行异常，不是所有组都有正常的主库实例，需要用户根据描述信息确认是否进行人工干预，具体描述信息可通过消息相关接口获取。

其他<0 的值：表示接口执行失败。

67. dwmon_set_group_arch_invalid

函数原型:

```
int  
  
dwmon_set_group_arch_invalid(  
  
    String      group_name  
  
);
```

功能说明:

修改指定组中所有备库的归档为无效状态，如果未指定组名，则通知所有组执行。

注意：执行此操作需要先登录监视器。

参数说明：

group_name：输入参数，守护组名，不指定则对所有组执行。

返回值：

0：执行成功。

<0：执行失败。

68. dwmon_set_group_recover_time

函数原型：

```
int  
  
dwmon_set_group_recover_time(  
  
    String      group_name,  
  
    int         recover_time  
  
);
```

功能说明：

修改指定组中所有备库的恢复间隔 (3~86400s)，只修改组中主库对应守护进程的内存值，如果未指定组名，则通知所有组执行。

注意：执行此操作需要先登录监视器。

参数说明：

group_name：输入参数，守护组名，不指定则对所有组执行。

recover_time：输入参数，指定恢复间隔 (3~86400s)，只修改组中主库守护进程的内存值。

返回值：

0：执行成功。

<0：执行失败。

69. dwmon_set_group_para_value

函数原型：

```
int
```

```
dwmon_set_group_para_value(
    String      group_name,
    String      para_name,
    String      para_value
);
```

功能说明:

修改指定组的所有守护进程的指定配置参数，同时修改 dmwatcher.ini 配置文件和内存值，如果未指定组名，则通知所有组执行。

目前可修改的参数包括：DW_ERROR_TIME、INST_RECOVER_TIME、INST_ERROR_TIME、INST_AUTO_RESTART、INST_SERVICE_IP_CHECK、RLOG_SEND_THRESHOLD、RLOG_APPLY_THRESHOLD。

如果组中某个守护进程故障，则跳过不修改，如果要重启故障的守护进程，需要在重启前修改守护进程 dmwatcher.ini 中的 DW_ERROR_TIME 和远程一致，否则 dmmonitor 执行命令时会报错处理。

注意：执行此操作需要先登录监视器。

参数说明:

group_name: 输入参数，守护组名，不指定则对所有组执行。

para_name: 输入参数，指定要修改的参数名称，目前只允许是 DW_ERROR_TIME、INST_RECOVER_TIME、INST_ERROR_TIME、INST_AUTO_RESTART、INST_SERVICE_IP_CHECK、RLOG_SEND_THRESHOLD、RLOG_APPLY_THRESHOLD 中的一种。

para_value: 输入参数，指定要修改的参数值，和 para_name 对应。

返回值:

0: 执行成功。

<0: 执行失败。

70. dwmon_set_group_auto_restart_on**函数原型:**

```
int
```

```
dwmon_set_group_auto_restart_on(  
  
    String      group_name  
  
);
```

功能说明：

打开指定组内所有 DMDSC 集群节点实例的自动拉起标记。

DMDSC 集群的自动拉起是靠 dmcss 执行的，dmcss 只负责拉起和自己的 dcr_seqno 相同的 db 节点。

此接口只修改 dmcss 内存中的自动拉起标记值，不会修改 dmdcr.ini 中的配置值，如果 dmcss 重启，此修改会丢失。

如果 dmcss 未配置自动拉起参数（DMDCR_DB_RESTART_INTERVAL/DMDCR_DB_STARTUP_CMD），此命令会执行失败。

此接口执行时，有可能所有节点实例都还未启动，无法校验登录口令，因此不要求先登录监视器。

参数说明：

group_name：输入参数，指定要执行操作的组名，只有一组时可以不指定。

返回值：

0：执行成功。

<0：执行失败。

71. dwmon_set_group_auto_restart_off

函数原型：

```
int  
  
dwmon_set_group_auto_restart_off(  
  
    String      group_name  
  
);
```

功能说明：

关闭指定组内所有 DMDSC 集群节点实例的自动拉起标记。

和 dwmon_set_group_auto_restart_on 接口相对应，可以一次性关闭守护系统内所有 DMDSC 集群上 dmcss 的自动拉起标记。

此命令只修改 dmcss 内存中的自动拉起标记值，如果 dmcss 重启，此修改会丢失。

执行此操作需要先登录监视器。

参数说明：

group_name：输入参数，指定要执行操作的组名，只有一组时可以不指定。

返回值：

0：执行成功。

<0：执行失败。

72. dwmon_detach_database

函数原型：

```
int  
  
dwmon_detach_database(  
  
    String      group_name,  
  
    String      db_name  
  
);
```

功能说明：

此接口允许将指定的备库分离出守护进程组，执行此接口需要先登录监视器。

对于全局守护类型的备库，使用此命令不会触发主库的 Failover 故障处理流程。该接口不会修改相关的配置信息，只是为了需要主动维护备库时使用。

接口会依次执行下列操作：

1. 设置指定备库的恢复间隔为 86400(s)
2. 修改主库到指定备库的归档状态无效

本接口不会主动退出备库实例，如果需要执行退出操作，可再通过调用 dwmon_stop_database 接口完成。

在备库维护结束后，如果备库是退出状态，可先调用 dwmon_startup_database 接口将备库启动，然后再调用 dwmon_attach_database 接口将备库重加入守护进程组。

参数说明：

group_name：输入参数，指定要执行操作的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定要正常分离出的备库名称，此参数必须指定。

返回值：

- 0：执行成功。
- <0：执行失败。

73. dwmon_attach_database

函数原型：

```
int  
  
dwmon_attach_database(  
  
    String      group_name,  
  
    String      db_name  
  
);
```

功能说明：

在使用 dwmon_detach_database 接口分离出去的备库维护完成后，可通过调用此接口将备库重加入守护进程组，执行此接口需要先登录监视器。

此接口执行时，会通知主库将指定备库的恢复间隔调整为 3s，3s 后主库守护进程会发起 Recovery 流程，将备库重加入守护系统。

注意，此接口不会将备库拉起，如果备库处于退出状态，需要先调用 dwmon_startup_database 接口将备库启动。

参数说明：

- group_name：输入参数，指定要执行操作的守护进程组名，只有一组时可以不指定。
- db_name：输入参数，指定要重加入的备库名称，此参数必须指定。

返回值：

- 0：执行成功。
- <0：执行失败。

74. dwmon_get_source_database_name

函数原型：

```
DwMonData  
  
dwmon_get_source_database_name(  

```

```
String      group_name,  
  
String      db_name  
);
```

功能说明：

获取指定库的源库名称，指定库为备库时才有意义，可以是异步备库。

参数说明：

group_name：输入参数，指定备库所属的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定备库名称。

返回值：

返回 DwMonData 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过对象成员 stringValue 获取源库名称。

<0：执行失败。

75. dwmon_get_ep_arch_send_info

函数原型：

```
ArchSendInfo  
  
dwmon_get_ep_arch_send_info(  
  
String      group_name,  
  
String      source_name,  
  
String      db_name  
);
```

功能说明：

获取指定的源实例到指定备库的归档发送信息。

源库如果是 DMDSC 集群，则需要依次取出集群内每个源实例到指定备库的归档发送信息，这个需要结合 dwmon_get_source_database_name 和 dwmon_get_database_info 这两个接口来完成。

首先用 dwmon_get_source_database_name 获取到源库名称，再通过 dwmon_get_database_info 获取到源库的所有实例名，再通过本接口依次取出源库每个实例到指定备库的归档发送信息，接口中的 source_name 字段就是指源库中的某个实例

名。

参数说明：

group_name：输入参数，输入指定备库所在的守护进程组名，如果只有一组可以不指定。

source_name：输入参数，输入要获取归档发送信息的源库实例名。

db_name：输入参数，输入要获取归档发送信息的备库名称。

返回值：

返回 ArchSendInfo 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 ArchSendInfo 的其他对象成员获取归档发送信息。

<0：执行失败。

76. dwmon_get_n_apply

函数原型：

ApplyNumInfo

```
dwmon_get_n_apply(  
    String          group_name,  
    String          db_name  
);
```

功能说明：

获取指定备库上的重演节点个数信息，也就是源库的节点个数。

参数说明：

group_name：输入参数，指定备库所属的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定备库名称。

返回值：

返回 ApplyNumInfo 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 ApplyNumInfo 的其他对象成员获取详细信息。

<0：执行失败。

77. dwmon_get_ep_nth_apply_info

函数原型:

ApplyLsnInfo

```
dwmon_get_ep_nth_apply_info(  
    String          group_name,  
    String          db_name,  
    int             nth  
);
```

功能说明:

获取指定备库上第 nth 组重演信息上记录的 lsn 等信息。

当备库对应的源库是 DMDSC 集群时，假如 DMDSC 集群有 N 个节点，那么备库上就会有 N 组重演信息依次和源库的每个节点相对应，N 值可以通过 dwmon_get_n_apply 接口中的 applyNum 字段值获取，本接口的输入参数 nth 的取值范围为 [0, applyNum-1]。

此接口只输出简略的 lsn 重演信息，如果还需要获取详细的重演统计信息，请参考 dwmon_get_ep_nth_apply_stat 接口说明。。

参数说明:

group_name: 输入参数，指定备库所属的守护进程组名，只有一组时可以不指定。

db_name: 输入参数，指定备库名称。

nth: 输入参数，指定要获取备库上第 nth 组重演信息。

返回值:

返回 ApplyInfo 对象，可通过对象成员 returnCode 获取执行结果。

0: 执行成功，可通过 ApplyInfo 的其他对象成员获取重演 lsn 信息。

<0: 执行失败。

78. dwmon_get_ep_nth_apply_stat

函数原型:

ArchApplyInfo

```
dwmon_get_ep_nth_apply_stat(  

```

```

String      group_name,

String      db_name,

int         nth

);

```

功能说明：

获取指定备库上第 nth 组重演信息，包含详细的重演统计信息。

当备库对应的源库是 DMDSC 集群时，假如 DMDSC 集群有 N 个节点，那么备库上就会有 N 组重演信息依次和源库的每个节点相对应，N 值可以通过 dwmon_get_n_apply 接口中的 applyNum 字段值获取，本接口的输入参数 nth 的取值范围为 [0, applyNum - 1]。

参数说明：

group_name：输入参数，输入指定备库所在的守护进程组名，如果只有一组可以不指定。

db_name：输入参数，指定要获取重演信息的备库名称。

nth：输入参数，指定要获取备库上第 nth 组重演信息。

返回值：

返回 ArchApplyStat 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 ArchApplyStat 的其他对象成员获取详细的重演信息。

<0：执行失败。

79. dwmon_clear_database_arch_send_info

函数原型：

```

int

dwmon_clear_database_arch_send_info(

String      group_name,

String      db_name

);

```

功能说明：

清理指定组中主库到指定备库实例的最近 N 次归档发送信息（通知主库执行）。

N 值取主库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际归档发送次数中的较

小值，实际归档发送次数可通过查询 V\$ARCH_SEND_INFO 获取。

如果主库是 DMDSC 集群，则会通知集群内所有实例执行清理。

参数说明：

group_name：输入参数，指定备库所属的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定备库名称。

返回值：

0：执行成功。

<0：执行失败。

80. dwmon_clear_group_arch_send_info

函数原型：

```
int  
  
dwmon_clear_group_arch_send_info (  
  
    String          group_name  
  
);
```

功能说明：

清理指定组中主库到所有实时或即时备库实例的最近 N 次归档发送信息（通知主库执行）。

N 值取主库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际归档发送次数中的较小值，实际归档发送次数可通过查询 V\$ARCH_SEND_INFO 获取。

参数说明：

group_name：输入参数，指定守护进程组名，如果不指定则通知所有组执行。

返回值：

0：执行成功。

<0：执行失败。

81. dwmon_clear_database_rapply_stat

函数原型：

```
int
```

```
dwmon_clear_database_rapply_stat(
    String      group_name,
    String      db_name
);
```

功能说明：

清理指定备库实例的最近 N 次日志重演信息（通知备库执行）。

N 值取备库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际重演次数中的较小值，实际重演次数可通过查询 V\$RAPPLY_STAT 获取。

参数说明：

group_name：输入参数，指定备库所属的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定备库名称。

返回值：

0：执行成功。

<0：执行失败。

82. dwmon_clear_group_rapply_stat

函数原型：

```
int
dwmon_clear_group_rapply_stat(
    String      group_name
);
```

功能说明：

清理指定组中所有备库实例的最近 N 次日志重演信息（通知所有备库执行）。

N 值取备库 dm.ini 中配置的 RLOG_SEND_APPLY_MON 值和实际重演次数中的较小值，实际重演次数可通过查询 V\$RAPPLY_STAT 获取。

参数说明：

group_name：输入参数，指定守护进程组名，如果不指定则通知所有组执行。

返回值：

0: 执行成功。
<0: 执行失败。

83. dwmon_check_recover

函数原型:

```
int  
  
dwmon_check_recover(  
  
    String          group_name,  
  
    String          db_name  
  
);
```

功能说明:

检查指定组的指定备库是否满足自动 Recover 条件, 如果不满足条件, 主库的守护进程不会对指定备库发起自动 Recovery 流程。

参数说明:

group_name: 输入参数, 指定备库所属的守护进程组名, 只有一组时可以不指定。
inst_name: 输入参数, 指定备库名称。

返回值:

0: 符合自动 Recover 条件, 备库已经在待恢复列表或者主库的守护进程即将发起对指定备库的 Recover 操作。

-20: 当前无法判断是否符合自动 Recover 条件, 比如有监视器命令正在执行、指定备库还未达到恢复间隔或者指定备库上的 RLOG_PKG 还未全部重演完成等情况, 需要稍后再次调用接口判断, 具体的描述信息可通过消息相关接口获取到。

-21: 表示备库不满足自动 Recover 条件, 具体的描述信息可通过消息相关接口获取到。

其他<0 的值: 表示接口执行失败。

84. dwmon_check_open

函数原型:

```
int
```

```
dwmon_check_open(
    String      group_name,
    String      db_name
);
```

功能说明：

检查指定组的指定库是否满足自动 Open 条件，如果不满足条件，守护进程不会通知数据库实例执行 Open 操作。

参数说明：

group_name：输入参数，指定库所属的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定数据库名称。

返回值：

0：符合自动 Open 条件，库已经 Open 成功或者正在执行 Open，或者库的守护进程即将通知实例执行 Open 操作。

-22：当前无法判断库是否满足自动 Open 条件，比如有监视器命令正在执行，或者数据库实例还未启动到 MOUNT 状态等情况，需要稍后再次调用接口判断，具体的描述信息可通过消息相关接口获取到。

-23：表示库不满足自动 Open 条件，并且库自身不是 OPEN 状态，具体的描述信息可通过消息相关接口获取到。

-24：表示库不满足自动 Open 条件，其中库实例已经是 OPEN 状态，但库的守护进程不是 OPEN 状态，具体的描述信息可通过消息相关接口获取到。

其他<0 的值：表示接口执行失败。

85. dwmon_deinit_env**函数原型：**

```
int
dwmon_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。

86. dwmon_free_handle

函数原型：

int

dwmon_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。

87. dwmon_get_monitor_state

函数原型：

DwMonState

dwmon_get_monitor_state();

功能说明：

获取监视器信息。

参数说明：

无

返回值：

返回 DwMonState 对象，可通过对象成员 returnCode 获取执行结果。

0：执行成功，可通过 DwMonState 的其他对象成员获取监视器信息。

<0: 执行失败。

88. dwmon_get_ep_nth_apply_with_parallel_info

函数原型:

ApplyFullInfo

dwmon_get_ep_nth_apply_with_parallel_info(

String group_name,

String db_name,

int nth

);

功能说明:

获取指定备库上第 nth 组重演信息上记录的并行重演信息, 包括 lsn 值、包序号、并行重演线程数以及各路并行重演线程已重演到的 lsn 信息等。

当备库对应的源库是 DMDSC 集群时, 假如 DMDSC 集群有 N 个节点, 则备库上就会有 N 组重演信息依次和源库的每个节点相对应, N 值可以通过 dwmon_get_n_apply 接口中的 applyNum 字段获取, 本接口的输入参数 nth 的取值范围为 [0, applyNum-1]。

参数说明:

group_name: 输入参数, 指定备库所属的守护进程组名, 只有一组时可以不指定。

db_name: 输入参数, 指定备库名称。

nth: 输入参数, 指定要获取备库上第 nth 组重演信息。

返回值:

返回 ApplyParallelInfo 对象, 可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过 DwMonState 的其他对象成员获取监视器信息。

<0: 执行失败。

89. dwmon_set_database_until_time

函数原型:

Int

```
dwmon_set_database_until_time (  
  
    String  group_name,  
  
    String  db_name,  
  
    String  until_time  
  
);
```

功能说明：

设置指定组的指定异步备库重演到的时间点。

参数说明：

group_name：输入参数，指定库所属的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定异步备库名称。

until_time：输入参数，指定异步备库重演到的时间点，可精确到微秒。

返回值：

0：执行成功。

<0：执行失败。

90. dwmon_get_database_until_time

函数原型：

```
DwMonData  
  
dwmon_get_database_until_time (  
  
    String group_name,  
  
    String db_name  
  
);
```

功能说明：

获取指定组的指定异步备库设置的重演到的时间点。

参数说明：

group_name：输入参数，指定库所属的守护进程组名，只有一组时可以不指定。

db_name：输入参数，指定异步备库名称。

返回值：

返回 DwMonData 对象，可通过对象成员 returnCode 获取执行结果。

0: 执行成功, 可通过对象成员 stringValue 获取异步备库设置的重演到的时间点。

<0: 执行失败。

91. dwmon_get_group_takeover_time

函数原型:

```
DwMonTkovtime  
  
dwmon_get_group_takeover_time(  
  
    String group_name  
  
) ;
```

功能说明:

获取指定组的自动接管时间。

参数说明:

group_name: 输入参数, 指定库所属的守护进程组名, 只有一组时可以不指定。

返回值:

返回 DwMonTkovtime 对象, 可通过对象成员 returnCode 获取执行结果。

0: 执行成功。

<0: 执行失败。

10.2.6 C 编程示例

10.2.6.1 Windows 编程示例

这里给出 C 程序的编程示例, Windows 操作系统, 使用 VS2010, 通过配置项目属性, 使用 dwmon.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己的需求再做调整。本例中达梦安装路径为“D:\dmdbms”, 头文件 dwmon_dll.h 和 lib 文件 dwmon.lib 位于“D:\dmdbms\include”目录下, 动态链接库 dwmon.dll 位于“D:\dmdbms\bin”目录下。

本例中 VS2010 配置步骤如下:

1. 选择“文件”->“新建”->“项目”

2.新建“Win32 控制台应用程序”，项目名称为“test_dwmon”，位置为“E:\test”，点击“确定”->“下一步”，进入配置向导，应用程序类型选择“控制台应用程序”，附加选项“空项目”，点击“完成”。

3.在“头文件”目录上右键“添加”->“现有项”，选择“D:\dmdbms\include”目录下的 dwmon_dll.h 添加。

4.在“源文件”目录上右键“添加”->“新建项”，选择“C++文件(.cpp)”类型，名称为“test_dwmon.c”，点击“添加”，文件创建完成，文件内容可参考下面的源码。

5.配置项目属性，点击项目名称右键选择“属性”->“配置属性”->“调试”，“工作目录”中选择目录“D:\dmdbms\bin”。

6.配置项目属性，点击项目名称右键选择“属性”->“配置属性”->“C/C++”->“常规”，“附加包含目录”中选择目录“D:\dmdbms\include”。

7.配置项目属性，点击项目名称右键选择“属性”->“配置属性”->“C/C++”->“预处理器”，“预处理器定义”中增加“DM64;_CRT_SECURE_NO_WARNINGS;”，本例中使用 64 位平台，如果是 32 位平台，则不需要增加“DM64;”。

8.配置项目属性，点击项目名称右键选择“属性”->“配置属性”->“链接器”->“常规”，“附加库目录”中选择目录“D:\dmdbms\include”。

9.配置项目属性，点击项目名称右键选择“属性”->“配置属性”->“链接器”->“输入”，“附加依赖项”中增加“dwmon.lib;”。

至此，VS2010 环境配置完成，注意 dll 和 lib 文件需要和项目平台保持一致，都是 32 位或者都是 64 位，否则会导致程序无法正常运行。

test_dwmon.c 源码如下：

```
#include "dwmon_dll.h"

#include "stdio.h"

#include "stdlib.h"

#include "windows.h"

#include "process.h"


/* 消息线程 */

DWORD

WINAPI
```

```
dwmon_get_msg_thread(  
    mhandle    handle  
)  
{  
    mschar      buf_msg[4097];  
    muint4      msg_len_out;  
    muint4      get_flag;  
    DWMON_RETURN code;  
  
    while (TRUE)  
    {  
        /* 等待消息事件 */  
        dwmon_msg_event_wait(handle);  
  
        /* 获取退出标记 */  
        if (dwmon_get_msg_exit_flag(handle) == TRUE)  
        {  
            break;  
        }  
  
        do  
        {  
            /* 获取并打印消息 */  
            code = dwmon_get_exec_msg(handle, buf_msg, 4097, &msg_len_out,  
dwmon_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 */  
    dwmon_set_msg_exit_event(handle);  
}
```

```
    return 0;
}

/* 清理环境 */
void
dwmon_clear_env(
    mhandle    handle
)
{
    /* 通知并等待 dwmon_get_msg_thread 消息线程退出 */
    dwmon_msg_event_deinit(handle);

    /* 销毁监视器环境 */
    dwmon_deinit(handle);

    /* 释放句柄 */
    dwmon_free_handle(handle);
}

int
main()
{
    DWMON_RETURN    ret;

    mhandle          handle;

    HANDLE           thread_handle;

    char             msg[4097];

    int              msg_len;

    /* 分配操作句柄 */
    ret = dwmon_alloc_handle(&handle);

    if (ret < 0)
    {
        fprintf(stdout, "dwmon alloc handle failed!\n");

        /* 获取 ret 对应的错误描述信息 */
        dwmon_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, dwmon_get_msg_thread, handle,
0, NULL);

if (thread_handle == NULL)
{
    ret = GetLastError();

    fprintf(stderr, "_beginthreadex error! desc:%s, code:%d\n", strerror(ret),
ret);

    dwmon_free_handle(handle);

    return ret;
}

/* 初始化监视器环境，可以选择忽略-15 错误（建立到所有守护进程连接失败，可能当前守护进程
都还未启动） */

ret = dwmon_init(handle, "D:\\dwmon\\dmmonitor.ini", "D:\\dwmon\\log");

if (ret < 0 && ret != -15)
{
    fprintf(stdout, "dwmon init failed!\n");

    /* 获取 ret 对应的错误描述信息 */

    dwmon_get_error_msg_by_code(handle, ret, msg, 4097, &msg_len);

    if (msg_len > 0)
    {
        fprintf(stdout, "code:%d, error msg:%s", ret, msg);
    }

    dwmon_clear_env(handle);
}
```

```
        return ret;

    }

    /* 登录监视器 */

    ret = dwmon_login(handle, "SYSDBA", "SYSDBA");

    if (ret < 0)

    {

        fprintf(stdout, "dwmon login failed!\n");

        /* 获取 ret 对应的错误描述信息 */

        dwmon_get_error_msg_by_code(handle, ret, msg, 4097, &msg_len);

        if (msg_len > 0)

        {

            fprintf(stdout, "code:%d, error msg:%s", ret, msg);

        }

    }

    /* 执行其他命令 */

    /*...*/

    /* 退出登录 */

    dwmon_logout(handle);

    /* 清理环境 */

    dwmon_clear_env(handle);

    system("pause");

    return 0;

}
```

10.2.6.2 Linux 编程示例

这里给出 C 程序的编程示例，运行环境为 Linux 操作系统，使用时可根据实际情况调整代码。

本例中有两个源文件：test_dwmon_linux.h、test_dwmon_linux.c，存放在

/dm/test 目录下，此目录下还包含 dll 头文件 dwmon_dll.h 和监视器配置文件 dmmonitor.ini。

本例中动态链接库文件存放在 /opt/dm/bin 目录下，程序中使用相对路径方式加载 libdwmon.so 文件，需要先设置环境变量：

```
export LD_LIBRARY_PATH=/opt/dm/bin:$LD_LIBRARY_PATH
```

注意：此种方式只对当前会话有效，可考虑写入到 ~/.bashrc 或 ~/.bash_profile 文件中永久生效。

gcc 编译命令：

```
gcc -o test_dwmon_linux test_dwmon_linux.c -ldl -rdynamic -lpthread
```

编译成功后，执行程序：

```
./test_dwmon_linux
```

test_dwmon_linux.h 源码如下：

```
#include "dwmon_dll.h"

#ifndef _TEST_DWMON_H_
#define _TEST_DWMON_H_

typedef void (*dwmon_msg_event_wait_fun_t)(mhandle mhdle);

typedef mbool (*dwmon_get_msg_exit_flag_fun_t)(mhandle mhdle);

typedef void (*dwmon_set_msg_exit_event_fun_t)(mhandle mhdle);

typedef void (*dwmon_msg_event_deinit_fun_t)(mhandle mhdle);

typedef DWMON_RETURN (*dwmon_deinit_fun_t)(mhandle mhdle);

typedef DWMON_RETURN (*dwmon_free_handle_fun_t)(mhandle mhdle);

typedef DWMON_RETURN (*dwmon_alloc_handle_fun_t)(mhandle* mhdle);

typedef DWMON_RETURN (*dwmon_logout_fun_t)(mhandle mhdle);

typedef

DWMON_RETURN

(*dwmon_get_exec_msg_fun_t)(

    mhandle        mhdle,
```

```
    mschar*      buf_msg,  
    uint4        buf_len,  
    uint4*       msg_len_out,  
    uint4*       get_flag  
);
```

```
typedef
```

```
DWMON_RETURN
```

```
(*dwmon_get_error_msg_by_code_fun_t) (
```

```
    mhandle      mhdle,  
    int4         code,  
    mschar*      buf_msg,  
    uint4        buf_len,  
    uint4*       msg_len_out
```

```
);
```

```
typedef
```

```
DWMON_RETURN
```

```
(*dwmon_init_fun_t) (
```

```
    mhandle      mhdle,  
    mschar*      ini_path,  
    mschar*      log_path
```

```
);
```

```
typedef
```

```
DWMON_RETURN
```

```
(*dwmon_login_fun_t) (
```

```
    mhandle      mhdle,  
    mschar*      username,  
    mschar*      password
```

```
);
```

```
#endif
```

test_dwmon_linux.c源码如下:

```
#include "stdio.h"
```

```
#include "stdlib.h"
```

```
#include "dlfcn.h"
```

```
#include "pthread.h"
```

```
#include "test_dwmon_linux.h"
```

```
dwmon_alloc_handle_fun_t      dwmon_alloc_handle_fun;
```

```
dwmon_free_handle_fun_t      dwmon_free_handle_fun;
```

```
dwmon_init_fun_t              dwmon_init_fun;
```

```
dwmon_deinit_fun_t            dwmon_deinit_fun;
```

```
dwmon_msg_event_wait_fun_t    dwmon_msg_event_wait_fun;
```

```
dwmon_get_msg_exit_flag_fun_t dwmon_get_msg_exit_flag_fun;
```

```
dwmon_set_msg_exit_event_fun_t dwmon_set_msg_exit_event_fun;
```

```
dwmon_msg_event_deinit_fun_t   dwmon_msg_event_deinit_fun;
```

```
dwmon_get_exec_msg_fun_t       dwmon_get_exec_msg_fun;
```

```
dwmon_get_error_msg_by_code_fun_t dwmon_get_error_msg_by_code_fun;
```

```
dwmon_login_fun_t              dwmon_login_fun;
```

```
dwmon_logout_fun_t             dwmon_logout_fun;
```

```
/* 消息线程 */
```

```
void*
```

```
dwmon_get_msg_thread(  
    mhandle      handle  
)
```

```
{

    mschar          buf_msg[4097];

    muint4           msg_len_out;

    muint4           get_flag;

    DWMON_RETURN     code;

    while (1)
    {
        /* 等待消息事件 */

        dwmon_msg_event_wait_fun(handle);

        /* 获取退出标记 */

        if (dwmon_get_msg_exit_flag_fun(handle) == 1)
        {
            break;
        }

        do
        {
            /* 获取并打印消息 */

            code = dwmon_get_exec_msg_fun(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 */

    dwmon_set_msg_exit_event_fun(handle);

    return NULL;
}
```

```
}

/* 清理环境 */

void
dwmon_clear_env(
    mhandle    handle
)
{
    /* 通知并等待 dwmon_get_msg_thread 消息线程退出 */
    dwmon_msg_event_deinit_fun(handle);

    /* 销毁监视器环境 */
    dwmon_deinit_fun(handle);

    /* 释放句柄 */
    dwmon_free_handle_fun(handle);
}

int
dwmon_load_fun(
    void*       handle
)
{
    char*       err = NULL;

    dwmon_alloc_handle_fun = dlsym(handle, "dwmon_alloc_handle");

    if ((err = dlerror()) != NULL)
    {
        dlclose(handle);

        printf("dlsym dwmon_alloc_handle error! desc:%s\n", err);

        return -1;
    }
}
```

```
}

dwmon_free_handle_fun = dlsym(handle, "dwmon_free_handle");
if ((err = dlerror()) != NULL)
{
    dlclose(handle);

    printf("dlsym dwmon_free_handle error! desc:%s\n", err);
    return -1;
}

dwmon_init_fun = dlsym(handle, "dwmon_init");
if ((err = dlerror()) != NULL)
{
    dlclose(handle);

    printf("dlsym dwmon_init error! desc:%s\n", err);
    return -1;
}

dwmon_deinit_fun = dlsym(handle, "dwmon_deinit");
if ((err = dlerror()) != NULL)
{
    dlclose(handle);

    printf("dlsym dwmon_deinit error! desc:%s\n", err);
    return -1;
}

dwmon_msg_event_wait_fun = dlsym(handle, "dwmon_msg_event_wait");
```

```
if ((err = dlerror()) != NULL)
{
    dlclose(handle);

    printf("dlsym dwmon_msg_event_wait error! desc:%s\n", err);
    return -1;
}

dwmon_get_msg_exit_flag_fun = dlsym(handle, "dwmon_get_msg_exit_flag");
if ((err = dlerror()) != NULL)
{
    dlclose(handle);

    printf("dlsym dwmon_get_msg_exit_flag error! desc:%s\n", err);
    return -1;
}

dwmon_set_msg_exit_event_fun = dlsym(handle,
"dwmon_set_msg_exit_event");
if ((err = dlerror()) != NULL)
{
    dlclose(handle);

    printf("dlsym dwmon_set_msg_exit_event error! desc:%s\n", err);
    return -1;
}

dwmon_msg_event_deinit_fun = dlsym(handle, "dwmon_msg_event_deinit");
if ((err = dlerror()) != NULL)
{
```

```
        dlclose(handle);

        printf("dlsym dwmon_msg_event_deinit error! desc:%s\n", err);

        return -1;
    }

    dwmon_get_exec_msg_fun = dlsym(handle, "dwmon_get_exec_msg");
    if ((err = dlerror()) != NULL)
    {
        dlclose(handle);

        printf("dlsym dwmon_get_exec_msg error! desc:%s\n", err);

        return -1;
    }

    dwmon_get_error_msg_by_code_fun = dlsym(handle,
"dwmon_get_error_msg_by_code");
    if ((err = dlerror()) != NULL)
    {
        dlclose(handle);

        printf("dlsym dwmon_get_error_msg_by_code error! desc:%s\n", err);

        return -1;
    }

    dwmon_login_fun = dlsym(handle, "dwmon_login");
    if ((err = dlerror()) != NULL)
    {
        dlclose(handle);
```

```
        printf("dlsym dwmon_login error! desc:%s\n", err);

        return -1;
    }

    dwmon_logout_fun = dlsym(handle, "dwmon_logout");
    if ((err = dlerror()) != NULL)
    {
        dlclose(handle);

        printf("dlsym dwmon_logout error! desc:%s\n", err);

        return -1;
    }

    return 0;
}

int
main()
{
    DWMON_RETURN    ret;

    mhandle          handle;

    void*            dlhandle;

    char             msg[4097];

    int              msg_len;

    pthread_t        tid;

    /* 打开动态链接库 */

    dlhandle = dlopen("libdwmon.so", RTLD_NOW);

    if (dlhandle == NULL)
    {
```

```
        printf("dlopen error! desc: %s\n", dlerror());

        return -1;
    }

    /* 加载 dll 函数 */
    ret = dwmon_load_fun(dlhandle);

    if (ret < 0)
    {
        dlclose(dlhandle);

        return -1;
    }

    /* 分配操作句柄 */
    ret = dwmon_alloc_handle_fun(&handle);

    if (ret < 0)
    {
        fprintf(stdout, "dwmon alloc handle failed!\n");

        /* 获取 ret 对应的错误描述信息 */
        dwmon_get_error_msg_by_code_fun(handle, ret, msg, 4097, &msg_len);

        if (msg_len > 0)
        {
            fprintf(stdout, "code:%d, error msg:%s", ret, msg);
        }

        dlclose(dlhandle);

        return ret;
    }

    /* 创建消息线程 */
```

```
ret    = pthread_create(&tid, NULL, dwmon_get_msg_thread, handle);

if(ret != 0)

{

    fprintf(stdout, "pthread_create failed!\n");

    dwmon_free_handle_fun(handle);

    dlclose(dlhandle);

    return ret;

}

/* 初始化监视器环境，可以选择忽略-15 错误（建立到所有守护进程连接失败，可能当前守护
进程都还未启动） */

ret = dwmon_init_fun(handle, "/dm/test/dmmonitor.ini", "/dm/test/log");

if (ret < 0 && ret != -15)

{

    fprintf(stdout, "dwmon init failed!\n");

    /* 获取 ret 对应的错误描述信息 */

    dwmon_get_error_msg_by_code_fun(handle, ret, msg, 4097, &msg_len);

    if (msg_len > 0)

    {

        fprintf(stdout, "code:%d, error msg:%s", ret, msg);

    }

    dwmon_clear_env(handle);

    dlclose(dlhandle);

    return ret;

}
```

```
/* 登录监视器 */

ret = dwmon_login_fun(handle, "SYSDBA", "SYSDBA");

if (ret < 0)
{
    fprintf(stdout, "dwmon login failed!\n");

    /* 获取 ret 对应的错误描述信息 */
    dwmon_get_error_msg_by_code_fun(handle, ret, msg, 4097, &msg_len);
    if (msg_len > 0)
    {
        fprintf(stdout, "code:%d, error msg:%s", ret, msg);
    }
}

/* 执行其他命令 */

/*...*/

/* 退出登录 */
dwmon_logout_fun(handle);

/* 清理环境 */
dwmon_clear_env(handle);

dlclose(dlhandle);

return 0;
}
```

10.2.7 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_8.0.0.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。

Linux 环境下需要安装 linux 版本的 jdk 和 Eclipse，下面的源码示例为 windows 环境，linux 下需要修改源码中的路径格式，其他不需要改动。

```
package com.dameng.test;

import com.dameng.console.dwmon.*;

public class DwMonTest {

    public DwMonDLL monDll = null;

    public DwMonMsg monMsg = null;

    public Thread msgThread = null;

    public String ini_path = "D:\\dwmon\\dmmonitor.ini";

    public String log_path = "D:\\dwmon\\log";

    public String username = "SYSDBA";

    public String password = "SYSDBA";

    //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.dwmon_msg_event_wait();
                }
            }
        });
    }
}
```

```
//获取退出标记

if (monDll.dwmon_get_msg_exit_flag() == true)

    break;

do

{

    //获取并打印消息

    monMsg = monDll.dwmon_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.dwmon_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.dwmon_msg_event_deinit();

    }

}
```

```
    }

}

public void test() throws InterruptedException

{

    boolean    ret1 = false;

    int        ret2 = 0;

    DwMonMsg retMsg = null;

    monDll = new DwMonDLL();

    //分配句柄

    ret1 = monDll.dwmon_init_handle();

    if (ret1 == false)

    {

        System.out.println("dwmon alloc handle failed!");

        return;

    }

    else

    {

        System.out.println("dwmon alloc handle success!");

    }

    //启动消息接收线程

    startThread();

    //初始化监视器，可以选择忽略-15 错误（建立到所有守护进程连接失败，可能当前守护进程都还未启动）

    ret2 = monDll.dwmon_init_env(ini_path, log_path);

    if (ret2 < 0 && ret2 != -15)

    {

        System.out.println("dwmon init failed!");

        /* 获取 ret2 对应的错误描述信息 */

        retMsg = monDll.dwmon_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.dwmon_free_handle();

        return;

    }

else

    {

        System.out.println("dwmon init success!");

    }

    //登录监视器

    ret2 = monDll.dwmon_login(username, password);

    if (ret2 < 0)

    {

        System.out.println("login failed!");

        /* 获取 ret2 对应的错误描述信息 */

        retMsg = monDll.dwmon_get_error_msg_by_code(ret2);

        if (retMsg.getReturnCode() >= 0)

        {

            System.out.println("code:" + ret2 + "," + retMsg.getMsg());

        }

    }

    //退出登录

    monDll.dwmon_logout();

    //退出消息线程

    endThread();

    //销毁监视器环境
```

```
monDll.dwmon_deinit_env();

//销毁句柄

monDll.dwmon_free_handle();

}

/**
 * @param args
 * @throws InterruptedException
 **/

public static void main(String[] args) throws InterruptedException {

    DwMonTest test = new DwMonTest();

    test.test();

}

}
```

10.2.8 错误码汇编

10.2.8.1 监视器

错误码	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	未登录监视器或服务器公钥发生变化，请重新登录
-6	读取 dmmonitor.ini 文件失败（路径错误或配置错误或没有权限）
-7	无效的日志文件路径
-8	设置命令执行标记失败，此标记已被设置，在标记被重置后才允许再次设置
-9	重置命令执行标记失败，已经被重置过
-10	实例故障或者实例对应的守护进程故障
-11	系统运行异常

-12	MPP 控制文件不一致
-13	守护进程控制文件没有记录项
-14	同一个守护系统中有多个确认监视器
-15	建立到所有守护进程的连接失败，可能守护进程都未启动，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。
-16	非法的参数，参数为空或长度超长或值非法
-17	没有找到 code 对应的错误信息
-18	内存不足
-19	监视器已达到最大个数（同一个守护进程组中最多允许同时启动 10 个监视器）
-20	备库不满足自动 Recover 条件
-20	当前无法判断实例是否满足恢复条件，请稍后重试！
-21	检测实例不满足恢复条件！
-23	守护系统中已经有 OPEN 状态的主库实例，但仍然存在其他异常实例
-24	守护系统当前运行异常，守护进程正在自动处理中，请稍后再观察状态是否正常
-25	守护系统当前运行异常，需要用户确认是否进行手工干预
-26	没有正常主库（实例和 watch 都是 OPEN 状态），或者组中全部是 local 实例，但存在 有实例没有 OPEN，需要等待守护进程处理完成
-27	没有正常主库（实例和 watch 都是 OPEN 状态），或者组中全部是 local 实例，但存在 有实例没有 OPEN，需要用户确认是否手动干预
-28	没有 deinit，直接 free handle 报错
-8014	用户自定义异常或者未知错误
-8145	消息校验失败
-8013	接收守护进程消息超时
-8206	用户名或密码错误，或用户没有 DBA 权限
-8191	用户名加密失败
-8192	密码加密失败
-8197	用户名或密码为空或长度超长
-8204	当前没有活动实例或守护进程状态不匹配，登录信息校验失败

DM 数据守护与读写分离集群 V4.0

-8210	登录监视器失败
-8243	监视器还未收到操作组的任何一个守护进程消息 (不包括 local 守护类型)
-8342	监视器未收到操作组的所有守护进程消息 (不包括 local 守护类型)
-8345	尚未收到守护进程的控制文件消息
-8380	守护进程的控制文件不是有效状态
-8346	主备库的守护进程控制文件不一致
-8021	实例故障
-8037	实例不是 MOUNT 状态
-8038	有消息正在发送中, 您的命令当前不能执行
-8042	实例故障, 操作终止
-8043	接收守护进程消息超时, 操作终止
-8044	实例执行命令失败
-8045	实例名无效或尚未收到实例消息
-8496	组中当前没有可执行的备库
-8497	实例的守护进程是 local 守护类型
-8456	未指定组名
-8455	未指定实例名
-8146	无效的组名
-8046	守护进程不是 Startup 或 OPEN 状态
-8371	守护进程不是 Startup 状态
-8062	实例的模式不匹配, 预期为 Standby 模式
-8055	STOP 守护进程组失败
-8058	Startup 守护进程组失败
-8106	SWITCHOVER 所用的实例是 Primary 模式
-8083	使用实例接管失败
-8078	实例状态不匹配
-8079	组中存在多个 Primary 实例, 或不存在实例满足: 1.实例是 Primary&OPEN 状态 2. 守护进程是 OPEN 状态 3.实例状态 (INST_OK) OK
-8344	实例不满足条件: 1.实例是 Standby&OPEN 状态 2.守护进程是 OPEN 状态 3.实例归

	档状态为 VALID
-8082	实例切换失败
-8089	实例不是 MOUNT 或 SUSPEND 状态，不允许 OPEN
-8069	实例执行 SQL 语句失败
-8388	组中有活动 Primary 实例，不再查找可接管实例列表
-8099	修改所有实例归档为无效状态失败
-8100	组中存在多个 OPEN 状态的 Primary 实例
-8111	待切换实例归档不是有效状态
-8113	监视器操作冲突，终止操作
-8440	实例不是 OPEN 状态
-8439	守护进程不是 OPEN 状态
-8121	实例非 Primary 或 Standby 模式
-8123	守护进程状态不匹配，不是 Startup、OPEN、SHUTDOWN 或 RECOVERY 状态
-8441	守护进程状态为 ERROR
-8125	实例故障，实例不是 OK 状态
-8126	实例的归档状态无效
-8137	检测到故障实例已恢复，重新修改实例为 Standby&OPEN 状态
-8148	系统组内没有或有多个活动 Primary 实例
-8435	组中没有活动 Primary 实例
-8149	内存分配失败
-8155	实例的 MPPCTL 文件更新失败
-8157	MPPCTL 文件恢复失败
-8159	构造新的 MPPCTL 文件失败
-8163	通知实例更新 MPPCTL 文件失败
-8165	尚未收到组的 MPPCTL 消息或组中不存在活动的 Primary 实例
-8265	尚未收到任何组的 MPPCTL 消息
-8359	实例非 Primary 模式
-8178	实例 OPEN 失败
-8180	到守护进程的连接断开

-8182	根据实例名获取发送端口失败
-8183	MPP 控制文件不一致
-8199	监视器尚未收到公钥信息，请稍候重试
-8201	监视器生成密钥对失败，请重试登录
-8203	守护进程不是 Startup/OPEN/RECOVERY 状态，或实例非 OK 状态，校验登录信息失败
-8220	守护进程不是 Startup/OPEN/SHUTDOWN 状态，不能修改参数
-8234	STOP 所有守护进程组失败
-8236	Startup 所有守护进程组失败
-8219	守护类型未知或是 LOCAL 守护类型 (可使用 show 命令查看系统运行状态)
-8222	通知启动实例失败 (启动实例的守护进程失败)
-8223	通知启动实例失败，实例已处于运行状态，或 watch 已配置自动重启，或 watch 的启动参数配置有误
-8229	实例 SHUTDOWN 失败
-8231	退出组中所有活动实例失败
-8238	启动所有组的实例失败
-8240	退出所有组的活动实例失败
-8257	组中所有实例均已处于 SHUTDOWN 状态
-8431	组中守护进程故障或实例故障，或尚未收到消息
-8272	STOP 实例的守护进程失败
-8275	Startup 实例的守护进程失败
-8277	守护进程不是 Startup 或 SHUTDOWN 状态
-8280	启动组中的所有实例失败
-8299/-8427	切换守护进程状态失败
-8311	切换守护进程为 OPEN 状态失败
-8309	修改实例实时归档为无效状态失败
-8368	请等待守护进程自动 OPEN 实例，不允许手动 OPEN FORCE
-8333	不符合 OPEN FORCE 条件
-8424	守护进程配置不一致
-8340	通知守护进程增加 CTL 记录失败

-8360	组中已经存在 Primary&OPEN 状态的实例
-8387	守护类型未知或不是 MPP 主备环境，不支持此命令
-8425	非 MPP 主备环境，不支持配置多组
-8389	存在分裂实例，不允许执行命令
-8390	守护进程不是 OPEN 状态
-8458	守护进程不是 OPEN 或 Startup 状态
-8392	实例不是 Standby&OPEN 状态
-8459	实例不是 Standby 模式
-8460	实例不是 OPEN 或 MOUNT 状态
-8393	存在 SLSN 更大的活动实例，不允许接管
-8394	存在 Primary & OPEN 状态的实例，不允许接管
-8395	没有找到实例对应的 Primary 实例，不允许接管
-8396	被接管实例故障前不是 Primary&OPEN，不允许接管
-8397	被接管实例的 flsn 大于接管用实例的 slsn，不允许接管
-8398	守护进程故障前不是 OPEN 或 RECOVERY 状态，不允许接管
-8399	被接管实例仍然处于活动状态，不允许接管
-8420	被接管实例仍然处于活动状态，模式/状态或 LSN 值不符合接管条件
-8400	被接管实例故障前到接管用实例的归档状态无效，不允许接管
-8401	守护进程的控制文件不是有效状态
-8402	实例的 OPEN 记录不一致
-8403	存在 SLSN 更大的实例，不允许接管
-8404	没有收到守护进程的实例的 OPEN 记录消息，不允许接管
-8405	存在多个或没有 Primary&OPEN 状态的实例
-8419	未找到实例对应的 Primary 实例
-8406	备库对应的 Primary 实例不是 Primary&OPEN 状态
-8407	实例不是 Primary&OPEN 状态
-8408	实例的归档状态无效或 MAL 链路异常
-8416	组中存在多个 Primary&OPEN 状态的实例
-8428	设置 DMWATCHER 的 MID 命令执行失败

-8430	实例 [Standby, DW_TYPE:LOCAL] 配置错误, 被同时配置在多个实例, 只允许配置在一个源实例上
-8454	存在多个守护进程控制文件有效的实例, 且不可互相加入
-8436	实例发生分裂, 需要用户干预
-8442	实例可作为 Standby 模式重加入, 请等待守护进程自动处理
-8444	实例无法自动重加入, 需要等实例恢复正常, 或者用户干预
-8446	实例运行异常
-8445	尚未收到组的 MPPCTL 消息, 请检查组中有活动的 Primary 实例, 并且 MPP_INI 已配置, 且存在有 dmmpp.ctl 文件
-8457	组中没有活动的守护进程或尚未收到消息
-8461	实例属于其他组, 不在指定的组中
-8498	无效的参数名或者参数值
-8500	修改配置参数失败
-8503	修改组中实例对应的守护进程配置参数失败
-8504	监视器不支持监控不同守护类型的组, 且只有 MPP 主备环境支持多组
-8505	组中不存在 Primary&OPEN 状态的实例或者有多个 Primary 实例或者 Primary 实例故障
-8506	没有找到实例对应的 Primary 实例
-8507	查找实例的归档信息失败
-8508	实例对应的 Primary 实例的守护进程不是 STARTUP/OPEN/RECOVERY 状态, 不允许执行此命令
-8514	修改组中实例的恢复时间间隔失败
-8515	指定的恢复时间间隔非法, 必须在 3~86400 (s) 范围内
-8520	修改实例的归档为无效状态失败
-8521	实例是 Primary 模式, 不允许执行此命令
-8522	组中存在多个 Primary 实例
-8523	组中当前没有活动的 Primary 实例
-8524	实例的归档信息没有找到
-8525	没有找到实例的源实例或尚未收到消息

-8528	不符合设置归档失效条件
-8529	实例对应的主库不唯一
-8530	不符合退出实例条件
-8532	将实例分离出守护进程组失败
-8533	不符合 STARTUP 守护进程组执行条件
-8534	不符合启动组中的所有实例条件
-8535	不符合 STOP 守护进程组条件
-8536	修改组中所有备库归档无效失败
-8538	修改所有组的所有备库归档无效失败
-8540	修改组中所有备库的恢复间隔失败
-8542	修改所有组的所有备库的恢复间隔失败
-8545	修改所有组守护进程配置参数失败
-8547	实例加回到守护进程组失败
-8551	实例状态正常，归档有效，在守护进程组中运行正常，不需要再重加入
-8552	实例当前不是运行状态，不符合执行条件
-8553	守护进程不是 OPEN 或 RECOVERY 状态
-8556	尚未收到组中主库实例的 MPPCTL 消息，请确保组中有活动的 PRIMARY 实例 (PRIMARY 实例的 MPP_INI 配置为 1，且实例本地有 dmmppctl 文件)
-8557	尚未收到任何组中主库实例的 MPPCTL 消息
-8562	守护进程组执行 KILL INSTANCE 命令失败
-8564	强杀所有组当前的活动实例失败
-8565	实例正在执行守护进程发起的命令，请稍候重试
-8567	切换守护进程状态失败
-8568	守护进程组中存在多个 PRIMARY&OPEN 状态的实例，不满足 RECOVER 条件
-8569	守护进程组中没有 PRIMARY&OPEN 状态的实例，不满足 RECOVER 条件
-8570	指定实例不在主库实例的归档目标中，不满足 RECOVER 条件
-8571	指定实例是 PRIMARY 模式，不满足 RECOVER 条件
-8573	实例的守护进程 OPEN 记录不是有效状态，命令执行失败
-8574	接收守护进程消息超时，无法检测指定实例是否可以自动 OPEN

-8575	实例的模式不匹配，获取实例对应的主库信息失败
-8576	获取实例对应的主库信息失败（实例未收到过主库发送的归档信息）
-542	SUSPEND 工作线程超时
-8617	实例的守护进程不是 STARTUP/OPEN 状态，不允许执行此命令。
-8618	当前无法比较 LSN 大小，存在交叉情况。
-8619	守护进程执行失败。
-8623	修改库中所有正常的守护进程配置参数失败。
-8624	在归档目标中未找到指定实例。
-8625	此命令不支持启动 DSC 集群，请等待 CSS 自动拉起或者通过 DSC 监视器手动拉起。
-8626	服务器当前有操作正在执行，需要等当前操作完成，请稍候再试！
-8627	库的控制节点尚未选出，请等待 CSS 自动处理！
-8631	DSC 集群正在启动，或者正在执行故障处理或故障重加入，请等待 DSC 集群自动处理完成。
-8632	控制守护进程执行成功，从 dmwatcher 执行失败，不影响 DSC 集群使用。
-8633	修改组中主库的控制守护进程到备库的恢复时间间隔成功，从 dmwatcher 修改失败，不影响 DSC 集群使用。
-8634	DSC 集群实例故障或者 DSC 集群不是 DSC_OPEN 状态，操作终止。
-8635	主库实例的 flsn 不等于备库实例的 flsn，命令执行失败。
-8636	待执行清理的源库实例的守护进程不是 STARTUP/OPEN/RECOVERY 状态，不允许执行此命令。
-8637	组中不存在活动主库。
-8638	获取集群的控制守护进程信息失败，请检查控制守护进程是否发生过变化并且已经启动。
-8639	系统中存在有未选出控制节点的库，需要先等控制节点选出，请稍候再试！
-8649	实例还未选出控制节点，请先等 DSC 集群选出控制节点。
-8650	实例的控制守护进程故障，无法判断是否可以自动 OPEN。
-8651	实例已经处于 OPEN 状态，但是控制守护进程故障。
-8652	PRIMARY 实例的守护进程不是 STARTUP/OPEN/RECOVERY 状态，不允许执行此命令
-8653	被接管实例故障前没有控制守护进程，或者从 dmwatcher 还处于活动状态
-8654	实例的守护进程正在进行中断处理，请稍候重试！

DM 数据守护与读写分离集群 V4.0

-8655	实例已经处于 SHUTDOWN 状态
-8656	主库到备库实例的归档仍然是有效状态，不允许执行此命令（需要先 detach 备库）
-8657	实例模式未知，不允许执行此命令
-8659	强杀实例失败
-8661	实例已经不是活动状态并且守护进程已处于 SHUTDOWN 状态
-8664	退出实例失败
-8667	启动实例失败
-8668	通知启动实例失败，实例已处于运行状态，或者被 dmcss 强制 HALT，或者 dmcss 已配置自动重启，或者 dmcss 的启动参数配置有误
-8669	组中配置的 LOCAL 守护类型的库超过最大值，请减少 LOCAL 守护类型的库个数以避免出现异常
-8670	组中配置的 GLOBAL 守护类型的库超过最大值，请减少 GLOBAL 守护类型的库个数以避免出现异常
-8671	组中不存在 DSC 集群，不允许执行此命令
-8673	打开组中所有 DSC 节点的自动拉起功能失败
-8676	打开库中所有节点的自动拉起功能失败，请检查 dmcss 是否故障或者配置有自动拉起参数
-8679	关闭组中所有 DSC 节点的自动拉起功能失败
-8682	关闭库中所有节点的自动拉起功能失败
-8684	此命令需要转发 dmcss 执行，如果守护进程正常，请检查 dmcss 是否出现异常
-8685	守护进程接收监视器消息超时，命令执行信息已经被清理，操作终止
-8686	实例的守护进程配置有自动拉起，请等待守护进程将其自动拉起
-8687	DSC 集群中存在有可以执行自动拉起的 dmcss，请确保所有 dmcss 的自动拉起功能是打开状态，并等待 dmcss 将其自动拉起
-8688	实例所在 DSC 集群控制守护进程故障或该命令不能发送到普通守护进程上执行，操作终止
-8689	dmmonitor.ini 中存在有需要合并的 MON_DW_IP 配置项（属于同一个 DMDSC 集群），需要以“/”分隔开，修改完成后，需要重启监视器！
-8690	dmmonitor.ini 中存在有不完整的 MON_DW_IP 配置项（需要配置 DMDSC 集群内所有

	守护进程的连接信息), 以“/”分隔开, 修改完成后, 需要重启监视器!
-8691	指定库中的控制节点尚未选出!
-8047	DSC 集群 ckpt_lsn < max_redo_lsn, 操作被限制, 建议对 DSC 控制节点手动执行 checkpoint 系统函数主动推进检查点
-8699	守护进程到实例的连接断开, 命令执行失败
-8700	守护进程到服务器链路断开, 校验登录口令失败, 请稍后重试
-8701	服务器公钥发生变化, 请重新登录

10.2.8.2 守护进程

错误码	解释
-10194	存在有 ep 实例状态不一致, 守护进程需要转入 unify_ep 状态统一各 ep 实例状态。
-10195	守护进程当前处于 unify_ep 状态或者 login_check 状态, 需要等当前操作完成。
-10196	本地守护进程为 LOCAL 守护类型, 可以将实例自动 OPEN。
-10197	本地库不是 Primary 或 Standby 模式
-10201	主库或者被恢复备库校验日志失败
-10203	实例广播出来的 dw_stat_flag 值和预期值不匹配, 当前操作已经被中断。
-10204	主库上未找到被恢复备库需要的归档日志
-10205	守护进程收到监视器命令, 但是本地 server 的 dw_stat_flag 标记值已经被设置, 不允许执行监视器命令, 返回错误给监视器。
-10209	通知本地主库由 suspend 状态重新 open 失败
-10211	检测到远程实例模式、状态不匹配, 设置中断标记为 TRUE, 等待中断当前处理
-10212	控制节点出现故障
-10213	控制守护进程设置中断标记 5s 之后, dmserver 当前操作还没有被中断掉, 先返回错误 code 给监视器, 避免监视器命令卡住。
-10214	备库和主库的 PERMANENT_MAGIC 不匹配
-10215	备库系统状态正在切换中
-10216	需要等待远程 SUSPEND 实例处理完成
-10217	本地主库或备库 DSC 集群 ckpt_lsn < max_redo_lsn, 操作被限制, 建议对 DSC 控制

	节点手动执行 checkpoint 系统函数主动推进检查点
-10219	接收 CSS 消息超时
-10220	CSS 执行操作冲突，CSS 端当前的 mid 值和发起命令的监视器 mid 值不一致
-10221	到 dmcss 的连接断开
-10222	CSS 执行失败
-10223	检测到 CSS 控制节点发生变化，命令中断执行
-10226	获取 CSS 控制节点通信端口失败，CSS 控制节点尚未选出或 CSS 未启动
-10229	设置 CSS 的 MID 命令执行失败
-10230	通知 CSS 拉起本地 EP 失败
-10233	守护进程超时仍未等到 EP STARTUP 成功，命令执行失败
-10239	通知 CSS 退出所有 EP 失败
-10240	守护进程超时仍未等到组中的所有 EP STOP 或 KILL 成功，命令执行失败
-10242	通知 CSS 强制 HALT 所有 EP 失败
-10244	组中节点对应的 CSS 配置的自动拉起参数不匹配，不再通知 CSS 打开自动拉起功能，请检查是否需要修改配置
-10248	通知 CSS 打开节点的自动拉起功能失败
-10250	通知 CSS 关闭节点的自动拉起功能失败
-10251	OPEN 实例时实时归档 APPLY_LSN 校验失败，设置所有归档失效
-10252	没有找到可以执行当前命令的活动 CSS，可能 CSS 故障或者还未选出控制节点或者不符合执行条件
-10254	从 CSS 控制节点广播出来的信息中查找本地 EP 所属的 DB 组失败
-10255	在 DB 组中查找和当前 CSS 相同 dcr_seqno 的 EP 失败
-10256	MID 已经被清理，当前命令执行失败
-10263	收到监视器执行 sql 语句的请求，但是守护进程状态不匹配，当前状态下不允许执行此操作
-10264	本地实例不可加入远程实例，强制 shutdown 本地实例
-10265	正在处理监视器命令，同一时刻只能处理一个监视器命令
-10266	正在执行监视器命令
-10267	没有监视器命令在执行

-10268	已收到相同报文序号的监视器命令
-10269	收到监视器命令，守护进程当前状态不允许执行监视器命令，返回错误

10.2.8.3 服务器

show arch info 命令以及 dwmon_get_inst_arch_send_info 接口中输出的服务器错误 code 说明。

守护进程通知恢复 code:

错误码	解释
0	恢复执行成功
100	未执行恢复操作
-10050	恢复失败，远程实例的模式或状态不匹配

服务器发送归档 code:

错误码	解释
0	发送成功
100	未执行发送操作或正在等待归档目标响应
-6018	消息长度超出网络发送消息缓存区
-6010	MAL 连接丢失（源库到归档发送目标）
-6021	MAL 链路已断开（源库到归档发送目标）
-517	系统处于状态切换中
-703	服务器模式不匹配
-518	无效的系统状态
-533	数据库的 PERMANENT_MAGIC 不匹配
-701	日志复制 LSN 值错误（归档目标收到的日志和本地日志不连续）
-503	服务器内存不足
-9512	MPP 执行停止
-9513	MPP 清理执行
-9504	MPP 收到新的计划，终止当前执行
-715	（源库）归档日志不存在
-718	（源库）收集到的归档日志和起始 LSN 不连续

-721	守护进程发起的 recover 恢复操作被中断
-108	打开重做日志失败
-702	归档日志错，备库重做日志失败
-716	归档日志文件仅包含检查包
-730	归档日志缺失
-731	归档日志重叠
-732	包含多个数据库的归档日志
-733	包含多个 DSC 节点的归档日志
-6526	重演超时
-9713	数据库魔数校验失败
-737	备库上传重演未结束，需等待备库重演完成
-967	ARCH_SEND_DELAY 未配置
-968	实例对应归档配置项未找到
-969	普通节点不支持该操作
-970	UNTIL TIME 未配置
-971	ARCH_SEND_DELAY 未配置
-6025	获取备库信息失败，请检查备库是否正常

10.2.8.4 DSC 集群控制器（DMCSS）

错误码	解释
-1075	[CSS]：没有找到指定组的信息
-1012	只有 CSS 控制节点能执行该操作
-1078	[CSS]：CSS 监控处于关闭状态，不允许执行此命令
-1071	[CSS]：正在操作的组中没有活动 EP
-1098	[CSS]：正在操作的组中没有非活动 EP
-1059	[CSS]：正在操作组和预期状态不一致
-1073	[CSS]：CSS 对需要手动拉起的组已经配置有自动重启，请等待 CSS 自动检测重启
-1090	[CSS]：CSS 没有对需要执行拉起的组配置重启参数

-1074	[CSS]: EP 已经处于 active 状态
-1099	[CSS]: 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
-1091	[CSS]: 非法的 EP 序号
-1102	无效的组名

咨询热线：400-991-6599

技术支持：dmtech@dameng.com

官网网址：www.dameng.com



武汉达梦数据库股份有限公司
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
