[三思笔记]一步一步学 DataGuard

本系列是一个一般初学者学习过程的记录，其中的概念可能解释的略显初级，其中的过程可能演示的略显生疏，其中的操作可能表现的略显繁琐，但，它记录了一种方法，一番热忱，一段历程，希望我曾经的崎岖能帮助你走的更平坦：[三思笔记]之一步一步学 DataGuard~~~~

另注：本系列中部分文章已发表至 itpub，如无授权请勿转载~~~

----君三思　junsansi@sain.com

第一部分 基础 1 之名词先混个脸熟 2007.11.23

它有无数个名字，有人叫它 dg，有人叫它数据卫士，有人叫它 data guard，在 oracle 的各项特性中它有着举足轻理的地位，它就是(掌声).....Oracle Data Guard。而对于我而言，我一定要亲切的叫它：DG(注：主要是因为打着方便)。

不少未实际接触过 dg 的初学者可能会下意识以为 dg 是一个备份恢复的工具。我要说的是，这种形容不完全错，dg 拥有备份的功能，某些情况下它甚至可以与 primary 数据库完全一模一样，但是它存在的目的并不仅仅是为了恢复数据，应该说它的存在是为了确保企业数据的高可用性，数据保护以及灾难恢复(注意这个字眼，灾难恢复)。dg 提供全面的服务包括：创建，维护，管理以及监控 standby 数据库，确保数据安全，管理员可以通过将一些操作转移到 standby 数据库执行的方式改善数据库性能。后面这一长串大家可以把它们理解成形容词，千万不要被其花哨的修饰所迷惑，要抓住重点，要拥有透明现象看本质的能力，如果没有那就要努力学习去拥有，下面我来举一个例子，比如我们夸人会说它聪明勇敢善良等等，这些就属于形容词，不重要，重点在于我们究竟想形容这个人是好人还是坏人。然后再回来看看 oracle 对 dg 功能上的形容，数据保护和灾难恢复应该都可以归结为高可用性，那么我们可以清晰的定位 dg 的用途了，就是构建高可用的企业数据库应用环境。

一、Data Guard 配置(Data Guard Configurations)

Data Guard 是一个集合，由一个 primary 数据库(生产数据库)及一个或多个 standby 数据库(最多 9 个)组成。组成 Data Guard 的数据库通过 Oracle Net 连接，并且有可能分布于不同地域。只要各库之间可以相互通信，它们的物理位置并没有什么限制，至于操作系统就更无所谓了(某些情况下)，只要支持 oracle 就行了。

你即可以通过命令行方式管理 primary 数据库或 standby 数据库，也可以通过 Data Guard broker 提供的专用命令行界面(DGMGRL)，或者通过 OEM 图形化界面管理。

1.Primary 数据库

前面提到，Data Guard 包含一个 primary 数据库即被大部分应用访问的生产数据库，该库即可以是单实例数据库，也可以是 RAC。

2.Standby 数据库

Standby 数据库是 primary 数据库的复制(事务上一致)。在同一个 Data Guard 中你可以最多创建 9 个 standby 数据库。一旦创建完成，Data Guard 通过应用 primary 数据库的 redo 自动维护每一个 standby 数据库。Standby 数据库同样即可以是单实例数据库，也可以是 RAC 结构。关于 standby 数据库，通常分两类：逻辑 standby 和物理 standby，如何区分，两类各有什么特点，如何搭建，这方面内容就是后面的章节主要介绍的，在这里呢三思先简单白话一下：

- 逻辑 standby

  就像你请人帮你素描画像，基本器官是都会有的，这点你放心，但是各器官位置啦大小啦肤色啦就不一定跟你本人一致了。

- 物理 standby

  就像拿相机拍照，你长什么样出来的照片就是什么样，眼睛绝对在鼻子上头。或者说就像你去照镜子，里外都是你，哇哈哈。具体到数据库就是不仅文件的物理结构相同，甚至连块在磁盘上的存储位置都是一模一样的(默认情况下)。

为什么会这样呢？这事就得从同步的机制说起了。逻辑 standby 是通过接收 primary 数据库的 redo log 并转换成 sql 语句，然后在 standby 数据库上执行 SQL 语句(SQL Apply)实现同步，物理 standby 是通过接收并应用 primary 数据库的 redo log 以介质恢复的方式(Redo Apply)实现同步。

另外，不知道大家是否注意到形容词上的细节：对于相机拍照而言，有种傻瓜相机功能强大而操作简便，而对于素描，即使是最简单的画法，也需要相当多的练习才能掌握。这个细节是不是也说明逻辑 standby 相比物理 standby 需要操作者拥有更多的操作技能呢？

二、Data Guard 服务(Data Guard Services)

- REDO 传输服务(Redo Transport Services)

  控制 redo 数据的传输到一个或多个归档目的地。
- Log 应用服务(Log Apply Services)

  应用 redo 数据到 standby 数据库，以保持与 primary 数据库的事务一致。redo 数据即可以从 standby 数据库的归档文件读取，也可直接应用 standby redo log 文件(如果实时应用打开了的话)。
- 角色转换服务(Role Transitions)

  Dg 中只有两种角色：primary 和 standby。所谓角色转换就是让数据库在这两个角色中切换，切换也分两种：switchover 和 failover

  switchover：转换 primary 数据库与 standby 数据库。switchover 可以确保不会丢失数据。

  failover：当 primary 数据库出现故障并且不能被及时恢复时，会调用 failover 将一个 standby 数据库转换为新的 primary 数据库。在最大保护模式或最高可用性模式下，failover 可以保证不会丢失数据。

注：上述各概念简要了解即可，这里写的太简单，不要咬文嚼字，不然你会越看越糊涂，相关服务在后面章节将会有详细介绍，不仅有直白的描述，还会有示例，再加上浅显的图片，就算你一看不懂，再看肯定懂：）

三、Data Guard 保护模式(Data Guard Protection Modes)

对于 Data Guard 而言，其生存逻辑非常简单，好好活，做有意义的事，做黑多黑多有意义的事：）

由于它提供了三种数据保护的模式，我们又亲切的叫它：有三模：

- **最大保护(Maximum protection)：**

这种模式能够确保绝无数据丢失。要实现这一步当然是有代价的，它要求所有的事务在提交前其 redo 不仅被写入到本地的 online redo log，还要同时提交到 standby 数据库的 standby redo log，并确认 redo 数据至少在一个 standby 数据库可用(如果有多个的话)，然后才会在 primary 数据库上提交。如果出现了什么故障导致 standby 数据库不可用的话，primary 数据库会被 shutdown。

- **最高性能(Maximum performance)：**

这种模式提供在不影响 primary 数据库性能前提下最高级别的数据保护策略。事务可以随时提交，当前 primary 数据库的 redo 数据也需要至少写入一个 standby 数据库，不过这种写入可以是不同步的。

如果网络条件理想的话，这种模式能够提供类似最高可用性的数据保护而仅对 primary 数据库有轻微的性能影响。

- **最高可用性(Maximum availability)：**

这种模式提供在不影响 primary 数据库可用前提下最高级别的数据保护策略。其实现方式与最大保护模式类

似，也是要求所有事务在提交前必须保障 redo 数据至少在一个 standby 数据库可用，不过与之不同的是，如果出现故障导入无法同时写入 standby 数据库 redo log，primary 数据库并不会 shutdown，而是自动转为最高性能模式，等 standby 数据库恢复正常之后，它又会再自动转换成最高可用性模式。

最大保护及最高可用性需要至少一个 standby 数据库 redo 数据被同步写入。三种模式都需要指定 LOG_ARCHIVE_DEST_n 初始化参数。LOG_ARCHIVE_DEST_n 很重要，你看着很眼熟是吧，我保证，如果你完完整整学完 dataguard，你会对它更熟。

四、Data Guard 优点总结

- 灾难恢复及高可用性
- 全面的数据保护
- 有效利用系统资源
- 在高可用及高性能之间更加灵活的平衡机制
- 故障自动检查及解决方案
- 集中的易用的管理模式
- 自动化的角色转换

经常开篇的灌输，相信大家已经看的出来，上面这几条都是形容词，看看就好，记住更好，跟人穷白活的时候通常能够用上:)

第一部分 基础 2 之术语再了解大概 2007.11.29

同一个 Data Guard 配置包含一个 Primary 数据库和最多九个 Standby 数据库。Primary 的创建就不说了，Standby 数据库初始可以通过 primary 数据库的备份创建。一旦创建并配置成 standby 后，dg 负责传输 primary 数据库 redo data 到 standby 数据库，standby 数据库通过应用接收到的 redo data 保持与 primary 数据库的事务一致。

一、Standby 数据库类型

前章我们简单介绍了 Standby 数据库，并且也知道其通常分为两类：物理 standby 和逻辑 standby，同时也简短的描述了其各自的特点，下面我们就相关方面进行一些稍深入的了解：

1. 物理 standby

我们知道物理 standby 与 primary 数据库完全一模一样(默认情况下，当然也可以不一样，事无绝对嘛)，Dg 通过 redo 应用维护物理 standby 数据库。通常在不应用恢复的时候，可以以 read-only 模式打开，如果数据库指定了 flashback area 的话，也可以被临时性的置为 read-write 模式。

- Redo 应用

  物理 standby 通过应用归档文件或直接从 standby 系统中通过 oracle 恢复机制应用 redo 文件。恢复操作属于块对块的应用(不理解？那就理解成块复制，将 redo 中发生了变化的块复制到 standby)。如果正在应用 redo，数据库不能被 open。

  Redo 应用是物理 standby 的核心，务必要搞清楚其概念和原理，后续将有专门章节介绍。

- Read-only 模式

  以 read-only 模式打开后，你可以在 standby 数据库执行查询，或者备份等操作(变相减轻 primary 数据库压力)。此时 standby 数据库仍然可以继续接收 redo 数据，不过并不会触发操作，直到数据库恢复 redo 应用。也就是说 read-only 模式时不能执行 redo 应用，redo 应用时数据库肯定处于未打开状态。如果需要的话，你可以在两种状态间转换，比如先应用 redo,然后 read-only，然后切换数据库状态再应用 redo，呵呵，人生就是循环，数据库也是一样。

- Read-write 模式

  如果以 read-write 模式打开，则 standby 数据库将暂停从 primary 数据库接收 redo 数据，并且暂时失去灾难保护的功能。当然，以 read-write 模式打开也并非一无是处，比如你可能需要临时调试一些数据，但是又不方便在正式库操作，那就可以临时将 standby 数据库置为 read-write 模式，操作完之后将数据库闪回到操作前的状态(闪回之后，Data Guard 会自动同步，不需要重建 standby)。

- 物理 standby 特点
  - 灾难恢复及高可用性

    物理 standby 提供了一个健全而且极高效的灾难恢复及高可用性的解决方案。更加易于管理的 switchover/failover 角色转换及最更短的计划内或计划外停机时间。
  - 数据保护

    应用物理 standby 数据库，Dg 能够确保即使面对无法预料的灾害也能够不丢失数据。前面也提到物理 standby 是基于块对块的复制，因此对象、语句统统无关，primary 数据库上有什么，物

理 standby 也会有什么。

- 分担 primary 数据库压力

  通过将一些备份任务、仅查询的需求转移到物理 standby，可以有效节省 primary 数据库的 cpu 以及 i/o 资源。

- 提升性能

  物理 standby 所使用的 redo 应用技术使用最底层的恢复机制，这种机制能够绕过 sql 级代码层，因此效率最高。

2. 逻辑 standby

逻辑 standby 是逻辑上与 primary 数据库相同，结构可以不一致。逻辑 standby 通过 sql 应用与 primary 数据库保持一致，也正因如此，逻辑 standby 可以以 read-write 模式打开，你可以在任何时候访问逻辑 standby 数据库。同样有利也有弊，逻辑 standby 对于某些数据类型以及一些 ddl,dml 会有操作上的限制。

- 逻辑 standby 的特点：

  除了上述物理 standby 中提到的类似灾难恢复，高可用性及数据保护等之外，还有下列一些特点：

  - 有效的利用 standby 的硬件资源

    除灾难恢复外，逻辑 standby 数据库还可用于其它业务需求。比如通过在 standby 数据库创建额外的索引、物化视图等提高查询性能并满足特定业务需要。又比如创建新的 schema(primary 数据库并不存在)然后在这些 schema 中执行 ddl 或者 dml 操作等。

  - 分担 primary 数据库压力

    逻辑 standby 数据库可以在更新表的时候仍然保持打开状态，此时这些表可同时用于只读访问。这使得逻辑 standby 数据库能够同时用于数据保护和报表操作，从而将主数据库从那些报表和查询任务中解脱出来，节约宝贵的 CPU 和 I/O 资源。

  - 平滑升级

    比如跨版本升级啦，打小补丁啦等等，应该说应用的空间很大，而带来的风险却很小(前提是如果你拥有足够的技术实力。另外虽然物理 standby 也能够实现一些升级操作，但如果跨平台的话恐怕就力不从心，所以此项就不做为物理 standby 的特点列出了)，我个人认为这是一种值的推荐的在线的滚动的平滑的升级方式。

二、Data Guard 操作界面(方式)

做为 oracle 环境中一项非常重要的特性，oracle 提供了多种方式搭建、操作、管理、维护 Data Guard 配置，比如：

- OEM(Oracle Enterprise Manager)

  Orcale EM 提供了一个窗口化的管理方式，基本上你只需要点点鼠标就能完全 dg 的配置管理维护等操作(当然三思仍然坚持一步一步学 rman 中的观点，在可能的情况下，尽可能不依赖视窗化的功能，所以这种操作方式不做详细介绍)，其实质是调用 oracle 为 dg 专门提供的一个管理器：Data Guard Broker 来实施管理操作。

- Sqlplus 命令行方式

  命令行方式的管理，本系列文章中主要采用的方式。不要一听到命令行就被吓倒，data guard 的管理命令并不多，你只需要在脑袋瓜里稍微挪出那么一小点地方用来记忆就可以了。

- DGMGRL(Data Guard broker 命令行方式)

  就是 Data Guard Broker，不过是命令行方式的操作。

- 初始化参数文件

我感觉不能把参数化参数视为一种操作方式，应该说，在这里，通过初始化参数，更多是提供更灵活的 Data Guard 配置。

三、Data Guard 的软硬件需求

1、硬件及操作系统需求

- 同一个 Data Gurid 配置中的所有 oracle 数据库必须运行于相同的平台。比如 inter 架构下的 32 位 linux 系统可以与 inter 架构下的 32 位 linux 系统组成一组 Data Guard。另外，如果服务器都运行于 32 位的话，64 位 HP-UX 也可以与 32 位 HP-UX 组成一组 Data Guard。
- 不同服务器的硬件配置可以不同，比如 cpu 啦，内存啦，存储设备啦，但是必须确保 standby 数据库服务器有足够的磁盘空间用来接收及应用 redo 数据。
- primary 数据库和 standby 数据库的操作系统必须一致，不过操作系统版本可以略有差异，比如(linux as4&linux as5)，primary 数据库和 standby 数据库的目录路径也可以不同。

2、软件需求

- Data Guard 是 Oracle 企业版的一个特性，明白了吧，标准版是不支持地。
- 通过 Data Guard 的 SQL 应用，可以实现滚动升级服务器数据库版本(要求升级前数据库版本不低于 10.1.0.3)。
- 同一个 Data Guard 配置中所有数据库初始化参数：COMPATIBLE 的值必须相同。
- Primary 数据库必须运行于归档模式，并且务必确保在 primary 数据库上打开 FORCE LOGGING，以避免用户通过 nologging 等方式避免写 redo 造成对应的操作无法传输到 standby 数据库。
- Primary 和 standby 数据库均可应用于单实例或 RAC 架构下，并且同一个 data guard 配置可以混合使用逻辑 standby 和物理 standby。
- Primary 和 standby 数据库可以在同一台服务器，但需要注意各自的数据文件存放目录，避免重写或覆盖。
- 使用具有 sysdba 系统权限的用户管理 primary 和 standby 数据库。
- 建议数据库必须采用相同的存储架构。比如存储采用 ASM/OMF 的话，那不分 primarty 或是 standby 也都需要采用 ASM/OMF。

另外还有很重要一点，注意各服务器的时间设置，不要因为时区/时间设置的不一置造成同步上的。

四、分清某某 REDO LOGS(Online Redo Logs, Archived Redo Logs, Standby Redo Logs)

黑多黑多的 redo，想必诸位早已晕头并吐过多次了吧。哎，说实话我描述的时候也很痛苦。这块涉及到中英文之间的意会。我又不能过度白话，不然看完我这篇文章再看其它相关文档的相关概念恐怕您都不知道人家在说什么，这种误人子弟的事情咱不能干(也许干过，但主观意愿上肯定是不想的)，更何况咱也是看各乱杂七杂八文档被误过 XXXXXXXXXXXXXXXXX 次(X=9)，深受其害，坚决不能再让跟俺一样受尽苦楚，历经磨难的 DDMM 们因为看俺的文档被再次一百遍啊一百遍。

但是已到关键时刻，此处不把 redo 混清楚，后头就得被 redo 混了，所以这里我要用尽我全部的口水+目前为止我所有已成体系的认识再给大家浅显的白话一回。注：基础概念仅一笔带过，水太大了也不好，要响应胡书记号召，书写节约型笔记。

REDO：中文直译是重做，与 UNDO 对应(天哪又扯出个概念，你看不见我看不见我看不见我)。重做什么？为什么要重做呢？首先重做是 oracle 对操作的处理机制，我们操作数据(增册改)并非直接反映到数据文件，而是

先被记录(就是 online redo log 喽)，等时机合适的时候，再由相应的进程操作提交到数据文件(详细可见：数据写过程中各项触发条件及逻辑)。你是不是想说如果把所有的 online redo logs 都保存下来，不就相当于拥有了数据库做过的所有操作了吗？en，我可以非常负责任的告诉你，你说的对，oracle 跟你想到一块去了并且也将其实现了，这就是 archived redo logs，简称 archive log 即归档日志。我们再回来看 Data Guard，由于 standby 数据库的数据通常都来自于 primary 数据库，怎么来的呢，通过 RFS 进程接收 primary 数据库的 redo，保存在本地，就是 Standby redo logs 喽(arch 模式的话不写 standby redo，直接保存归档)，然后 standby 数据库的相关进程读取接收到的 redo 数据，再将其写入 standby 数据库。保存之后数据又是怎么生成的呢，两种方式，物理 standby 通过 redo 应用，逻辑 standby 通过 sql 应用，不管是哪种应用，应用的是什么呢？是 redo log 中的内容(默认情况下应用 archived redo logs，如果打开了实时应用，则直接从 standby redo logs 中读取)，至于如何应用，那就是 redo 应用和 sql 应用机制的事情了(也许后头我们会深入聊一聊这个话题，很复杂也很有趣)。

针对上述内容我们试着总结一下，看看能否得出一些结论：

对于 primary 数据库和逻辑 standby 数据库，online redo log 文件肯定是必须的，而对于物理 standby 是否还需要 redo log 呢？毕竟物理 standby 通常不会有写操作，所以物理 standby 应该不会生成有 redo 数据。为保证数据库的事务一致性必然需要有归档，也就是说不管 primary 或 standby 都必须运行于归档模式。

standby redo logs 是 standby 数据库特有的文件(如果配置了的话)，就本身的特点比如文件存储特性，配置特性等等都与 online redo logs 非常类似，不过它存储的是接收自 primary 数据库的 redo 数据，而 online redo logs 中记录的是本机中的操作记录。

上面的描述大家尽可能意会，能够理解最好，理解不了也没关系，我始终认为，只要坚定不移的学习下去，总会水到渠成。下面进入实战章节，先来个简单的，创建物理 standby。

第二部分 物理 standby(1)创建步骤 2007.12.03

一、准备工作

不管物理 standby 还是逻辑 standby，其初始创建都是要依赖 primary 数据库，因为这个准备工作中最重要的一部分，就是对 primary 数据库的配置。

1、打开 Forced Logging 模式

将 primary 数据库置为 FORCE LOGGING 模式。通过下列语句：

```
SQL> alter database force logging;
```

提示：关于 FORCE LOGGING

想必大家知道有一些 DDL 语句可以通过指定 NOLOGGING 子句的方式避免写 redo log(目的是提高速度，某些时候确实有效)，指定数据库为 FORCE LOGGING 模式后，数据库将会记录除临时表空间或临时回滚段外所有的操作而忽略类似 NOLOGGING 之类的指定参数。如果在执行 force logging 时有 nologging 之类的语句在执行，则 force logging 会等待直到这类语句全部执行。FORCE LOGGING 是做为固定参数保存在控制文件中，因此其不受重启之类操作的影响(只执行一次即可)，如果想取消，可以通过 alter database no force logging 语句关闭强制记录。

2、创建密码文件(如果不存在的话)

需要注意的是，同一个 Data Guard 配置中所有数据库必须都拥有独立的密码文件，并且必须保证同一个 Data Guard 配置中所有数据库服务器的 SYS 用户拥有相同密码以保证 redo 数据的顺利传输，因为 redo 传输服务通过认证的网络会话来传输 redo 数据，而会话使用包含在密码文件中的 SYS 用户密码来认证。

3、配置 Standby Redo Log

对于最大保护和最高可用性模式，Standby 数据库必须配置 standby redo log，并且 oracle 推荐所有数据库都使用 LGWR ASYNC 模式传输，当然你现在可能还不知道 LGWR ASYNC 是什么问题，没关系，你很快就会知道了。

Oracle 建议你在创建 standby 时就考虑 standby redolog 配置的问题。standby redologs 与 online redologs 非常类似，应该说两者只是服务对象不同，其它参数属性甚至操作的命令格式几乎都一样，你在设计 standby redologs 的时候完全可以借鉴创建 online redologs 的思路，比如多个文件组啦，每组多个文件冗余之类的。除些之外呢，oracle 提供了一些标准的建议如下：

- 确保 standby redo log 的文件大小与 primary 数据库 online redo log 文件大小相同。

  这个很好理解的吧，就是为了接收和应用方便嘛。

- 创建适当的日志组

  一般而言，standby redo 日志文件组数要比 primary 数据库的 online redo 日志文件组数至少多一个。推荐 standby redo 日志组数量基于 primary 数据库的线程数(这里的线程数可以理解为 rac 结构中的 rac 节点数)。

  有一个推荐的公式可以做参考：(每线程的日志组数+1)*最大线程数

  例如 primary 数据库有两个线程，每个线程分配两组日志，则 standby 日志组数建议为 6 组，使用这个公式可以降低 primary 数据库实例 LGWR 进程锁住的可能性。

  提示：逻辑 standby 数据库有可能需要视工作量增加更多的 standby redo log 文件(或增加归档进程)，因为逻辑 standby 需要同时写 online redo log 文件。

Standby redo log 的操作方式与 online redo log 几乎一模一样，只不过在创建或删除时需要多指定一个 standby 关键字，例如添加：

```
SQL> alter database add standby logfile group 4 ('e:\ora10g\oradata\jsspdg\STANDBYRD01.LOG') size 20M;
```

删除也同样简单：

```
SQL> alter database drop standby logfile group 4;
```

另外，从可靠性方面考虑，建议在 primary 数据库也创建 standby redologs，这样一旦发生切换，不会影响 primary 做为 standby 的正常运行。

验证 standby redo log 文件组是否成功创建

例如：

```
SQL> SELECT GROUP#,THREAD#,SEQUENCE#,ARCHIVED,STATUS FROM V$STANDBY_LOG;
```

4、设置初始化参数

对于 primary 数据库，需要定义几个 primary 角色的初始化参数控制 redo 传输服务，还有几个附加的 standby 角色的参数需要添加以控制接收 redo 数据库并应用(switchover/failover 后 primary/standby 角色可能互换，所以建议对于两类角色相关的初始化参数都进行配置)。

下列参数为 primary 角色相关的初始化参数：	
DB_NAME	注意保持同一个 Data Guard 中所有数据库 DB_NAME 相同。 例如：DB_NAME=jssweb
DB_UNIQUE_NAME	为每一个数据库指定一个唯一的名称，该参数一经指定不会再发生变化，除非你主动修改它。 例如：DB_UNIQUE_NAME=jssweb
LOG_ARCHIVE_CONFIG	该参数通过 DG_CONFIG 属性罗列同一个 Data Guard 中所有 DB_UNIQUE_NAME(含 primary db 及 standby db)，以逗号分隔 例如：LOG_ARCHIVE_CONFIG='DB_CONFIG=(jssweb,jsspdg)'
CONTROL_FILES	没啥说的，控制文件所在路径。
LOG_ARCHIVE_DEST_n	归档文件的生成路径。该参数非常重要，并且属性和子参数也特别多(这里不一一列举，后面用到时单独讲解如果你黑好奇，建议直接查询 oracle 官方文档。Data guard 白皮书第 14 章专门介绍了该参数各属性及子参数的功能和设置)。例如： LOG_ARCHIVE_DEST_1= 'LOCATION=E:\ora10g\oradata\jssweb VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=jssweb'
LOG_ARCHIVE_DEST_STATE_n	指定参数值为 ENABLE，允许 redo 传输服务传输 redo 数据到指定的路径。该参数共拥有 4 个属性值，功能各不相同。
REMOTE_LOGIN_PASSWORDFILE	推荐设置参数值为 EXCLUSIVE 或者 SHARED，注意保证相同 Data Guard 配置中所有 db 服务器 sys 密码相同。
LOG_ARCHIVE_FORMAT	指定归档文件格式。
LOG_ARCHIVE_MAX_PRODUC	指定归档进程的数量(1-30)，默认值通常是 4。

ESSES	
以下参数为 standby 角色相关的参数，建议在 Primary 数据库的初始化参数中也进行设置，这样在 role transition 后(Primary 转为 Standby)也能正常运行：	
FAL_SERVER	指定一个数据库 SID，通常该库为 primary 角色。 例如：FAL_SERVER=jssweb
FAL_CLIENT	指定一个数据库 SID，通常该库为 standby 角色。 例如：FAL_CLIENT=jsspdg 提示：FAL 是 Fetch Archived Log 的缩写
DB_FILE_NAME_CONVERT	在做 duplicate 复制和传输表空间的时候这类参数讲过很多遍，该参数及上述内容中同名参数功能，格式等完全相同。
LOG_FILE_NAME_CONVERT	同上
STANDBY_FILE_MANAGEMENT	如果 primary 数据库数据文件发生修改（如新建，重命名等）则按照本参数的设置在 standby 中做相应修改。设为 AUTO 表示自动管理。设为 MANUAL 表示需要手工管理。 例如：STANDBY_FILE_MANAGEMENT=AUTO

注意：上面列举的这些参数仅只是对于 primary/standby 两角色可能会相关的参数，还有一些基础性参数比如*_dest,*_size 等数据库相关的参数在具体配置时也需要根据实际情况做出适当修改。

5、确保数据库处于归档模式

```
SQL> archive log list;
数据库日志模式                 存档模式
自动存档                       启用
.....
```

如果当前 primary 数据库并未处于归档模式，可通过下列命令将数据库置为归档模式：

```
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE ARCHIVELOG;
SQL> ALTER DATABASE OPEN;
```

二、手把手的创建物理 standby

1、创建备份(手工复制数据文件或通过 RMAN) ---primary 库操作

2、创建控制文件 --primary 库操作

通过下列语句为 standby 数据库创建控制文件

```
SQL> alter database create standby controlfile as 'd:\backup\jsspdg01.ctl';
```

注意哟，控制文件通常需要有多份，你要么手工将上述文件复制几份，要么用命令多创建几个出来。另外，创建完控制文件之后到 standby 数据库创建完成这段时间内，要保证 primary 数据库不再有结构性的变化(比如增加表空间等等)，不然 primary 和 standby 同步时会有问题。

3、创建初始化参数文件

- 创建客户端初始化参数文件

  例如：

```
SQL> create pfile='d:\backup\initjsspdg.ora' from spfile;
```

- 修改初始化参数文件中的参数
  根据实际情况修改吧，注意 primary 和 standby 不同角色的属性配置，注意文件路径。

4、复制文件到 standby 服务器

至少三部分：数据文件，控制文件，修改过的初始化参数文件，注意路径。

5、配置 standby 数据库

如果你看过三思之前"一步一步学 rman"系列，看过"duplicate 复制数据库"，或看过"传输表空间复制数据"系列，那么对于创建一个新的数据库应该非常熟悉了，下面再简单描述一下步骤：

1).创建新的 OracleService(windows 环境下需要)。

2).创建密码文件，注意保持密码与 primary 数据库一致。

3).配置监听并启动

4).修改 primary 和 standby 的 tnsnames.ora，各自增加对应的 Net Service Name。

5).创建服务器端的初始化文件

6、启动 standby

注意哟，咱们前面说过的，物理 standby 极少情况下可以以 read-write 模式打开，某些情况下可以以 read-only 模式打开，所以默认情况下，加载到 mount 状态即可。

```
SQL> STARTUP MOUNT;
```

启动 redo 应用

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT FROM SESSION;
```

启动实时应用

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE  DISCONNECT FROM SESSION;
```

提示：**disconnect from session** 子句并非必须，该子句用于指定启动完应用后自动退出到命令操作符前，如果不指定的话，当前 session 就会一直停留处理 redo 应用，如果想做其它操作，就只能新建一个连接。

7、停止 standby

正常情况下，我们停止也应该是先停止 redo 应用，可以通过下列语句：

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CALCEL;
```

然后再停止 standby 数据库

```
SQL> SHUTDOWN IMMEDIATE;
```

当然你非要直接 shutdown 也没问题，dg 本来就是用于容灾的，别说你生停 standby，就是直接拔电源也不怕。

基本步骤就是这样，下面我们进入实践环节.....

第二部分 物理 standby(2)创建示例 2007.12.06

为了最大的降低硬件需求，此处创建的 data guard 处于同一台机器，但其创建过程与多机并无区别。做为演示用的示例足够了，我们分两阶段配置，分别是配置 primary 数据库和配置 standby 数据库，如下：

一、Primary 数据库配置及相关操作

1、确认主库处于归档模式

```
SQL> archive log list;
数据库日志模式                 存档模式
自动存档                   启用
存档终点                   E:\ora10g\oradata\jssweb
最早的联机日志序列         148
下一个存档日志序列     150
当前日志序列                 150
SQL>
```

2、将 primary 数据库置为 FORCE LOGGING 模式。通过下列语句：

```
SQL> alter database force logging;

数据库已更改。
```

3、创建 standby 数据库控制文件

```
SQL> alter database create standby controlfile as 'd:\backup\jsspdg01.ctl';

数据库已更改。
```

4、创建 primary 数据库客户端初始化参数文件

注：主要此处修改项较多，为了方便，我们首先创建并修改 pfile，然后再通过 pfile 重建 spfile，你当然也可以通过 alter system set 命令直接修改 spfile 内容。

```
SQL> create pfile from spfile;

文件已创建。
```

将该初始化参数文件复制一份，做为 standby 数据库的客户端初始化参数文件

```
SQL> host copy e:\ora10g\product\10.2.0\db_1\database\initjssweb.ora d:\backup\initjsspdg.ora
已复制                1  个文件。

SQL>
```

修改客户端初始化参数文件，增加下列内容

```
DB_UNIQUE_NAME=jssweb
```

```
LOG_ARCHIVE_CONFIG='DG_CONFIG=(jssweb,jsspdg)'
LOG_ARCHIVE_DEST_1='LOCATION=E:\ora10g\oradata\jssweb\
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=jssweb'
LOG_ARCHIVE_DEST_2='SERVICE=jsspdg LGWR ASYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=jsspdg'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE

#--------配置 standby 角色的参数用于角色转换
FAL_SERVER=jsspdg
FAL_CLIENT=jssweb
DB_FILE_NAME_CONVERT='oradata\jsspdg','oradata\jssweb'
LOG_FILE_NAME_CONVERT='oradata\jsspdg','oradata\jssweb'
STANDBY_FILE_MANAGEMENT=AUTO
```

通过 pfile 重建 spfile

```
SQL> shutdown immediate
...
SQL> create spfile from pfile='initjssweb.ora';

文件已创建。
```

5、复制数据文件到 standby 服务器(方式多样，不详述)

注意需要复制所有数据文件，备份的控制文件及客户端初始化参数文件

6、配置 listener 及 net service names(方式多样，不详述)。

完之后重启 listener:

```
E:\ora10g>lsnrctl stop
E:\ora10g>lsnrctl start
```

通过 tnsping 测试 tnsnames 是否正确有效：

```
E:\ora10g>tnsping jssweb
...
Attempting to contact (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = jss)(PORT = 1521))
(CONNECT_
DATA = (SERVER = DEDICATED) (SERVICE_NAME = jssweb)))
OK (30 毫秒)

E:\ora10g>tnsping jsspdg
...
Attempting to contact (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = jss)(PORT = 1521))
(CONNECT_
DATA = (SERVER = DEDICATED) (SERVICE_NAME = jsspdg)))
OK (10 毫秒)
```

二、Standby 数据库配置及相关操作

1、通过 ORADIM 创建新的 OracleService

2、创建密码文件，注意保持 sys 密码与 primary 数据库一致。

```
E:\ora10g>orapwd file=e:\ora10g\product\10.2.0\db_1\database\PWDjsspdg
.ora password=verysafe entries=30
```

3、创建目录

```
E:\ora10g\product\10.2.0\admin\jsspdg>mkdir adump
```

4、复制文件，不做过多描述

5、修改初始化参数文件

增加下列参数

```
db_unique_name=jsspdg
LOG_ARCHIVE_CONFIG='DG_CONFIG=(jssweb,jsspdg)'
DB_FILE_NAME_CONVERT='oradata\jssweb','oradata\jsspdg'
LOG_FILE_NAME_CONVERT='oradata\jssweb','oradata\jsspdg'
LOG_ARCHIVE_FORMAT=log%t_%s_%r.arc
LOG_ARCHIVE_DEST_1='LOCATION=E:\ora10g\oradata\jsspdg\
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=jsspdg'
LOG_ARCHIVE_DEST_STATE_1=ENABLE

#---下列参数用于角色切换
LOG_ARCHIVE_DEST_2='SERVICE=jssweb                    LGWR                    ASYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)    DB_UNIQUE_NAME=jssweb'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
FAL_SERVER=jssweb
FAL_CLIENT=jsspdg
STANDBY_FILE_MANAGEMENT=AUTO
```

注意同时修改*_dest 的路径。

通过该 pfile 创建 spfile

```
SQL> create spfile from pfile='D:\backup\initjsspdg.ora';

文件已创建。
```

6、启动 standby 到 mount

```
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area    167772160 bytes
```

```
Fixed Size                  1289484 bytes
Variable Size              62915316 bytes
Database Buffers           96468992 bytes
Redo Buffers                7098368 bytes
数据库装载完毕。
```

7、启动 redo 应用

```
SQL> alter database recover managed standby database disconnect from session;

数据库已更改。
```

8、查看同步情况

首先连接到 primary 数据库

```
SQL> show parameter instance_name;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
instance_name                        string      jssweb
SQL> alter system switch logfile;

系统已更改。

SQL> select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
            51
```

连接到 standby 数据库

```
SQL> show parameter instance_name;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
instance_name                        string      jsspdg
SQL> select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
            51
```

9、暂停应用

通过下列语句暂停 redo 应用。

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;

数据库已更改。
```

注意，此时只是暂时 redo 应用，并不是停止 Standby 数据库，standby 仍会保持接收只不过不会再应用接收到的归档，直到你再次启动 redo 应用为止。

哈哈，成功鸟！现在你是不是想知道怎么把 standby 变成 primary 呢？接着往下看~~~~~~~~~~

第二部分 物理 standby(3)角色转换　2007.12.11

第 1 节的时候我们就提到了角色切换，我们也听说了其操作简单但用途广泛，同时我们也猜测其属于 primary 与 standby 之间的互动，那么在 primary 和 standby 数据库(之一)上都需要有操作，并且切换又分了：switchover 和 failover，前者是无损切换，不会丢失数据，而后者则有可能会丢失数据，并且切换后原 primary 数据库也不再是该 data guard 配置的一部分了.针对不同 standby(逻辑或物理)的处理方式也不尽相同。en,内容也挺多地。我们还是先大概了解下概念，然后再通过实战去印证。

角色转换前的准备工作

- 检查各数据库的初始化参数，主要确认对不同角色相关的初始化参数都进行了正确的配置。
- 确保可能成为 primary 数据库的 standby 服务器已经处于 archivelog 模式。
- 确保 standby 数据库的临时文件存在并匹配 primary 数据库的临时文件
- 确保 standby 数据库的 RAC 实例只有一个处于 open 状态。（对于 rac 结构的 standby 数据库，在角色转换时只能有一个实例 startup。其它 rac 实例必须统统 shutdown，待角色转换结束后再 startup）

Switchover：

无损转换，通常是用户手动触发或者有计划的让其自动触发，比如硬件升级啦，软件升级啦之类的。通常它给你带来的工作量非常小并且都是可预计的。其执行分两个阶段，第一步， primary 数据库转换为 standby 角色，第二步，standby 数据库(之一)转换为 primary 角色，primary 和 standby 只是简单的角色互换，这也印证了我们前面关于角色转换是 primary/standby 互动的猜测。

Failover：

不可预知原因导致 primary 数据库故障并且短期内不能恢复就需要 failover。如果是这种切换那你就要小心点了，有可能只是虚惊一场，甚至连你可能损失的脑细胞的数量都能预估，但如果运气不好又没有完备的备份恢复策略而且 primary 数据并非处于最大数据保护或最高可用性模式地话，黑黑，哭是没用地，表太伤心了，来，让三思 GG 安慰安慰你，这种情况下呢丢失数据有可能是难免的，并且如果其故障未能修复，那它甚至连快速修复成为 standby 的机会也都失去了呐，咦，你脑门怎么好像在往外冒水，难道是强效净肤液，你的脸也忽然好白皙哟~~~~

在执行 failover 之前，尽可能将原 primary 数据库的可用 redo 都复制到 standby 数据库。

注意，如果要转换角色的 standby 处于 maximum protection 模式，需要你首先将其切换为 maximum performance 模式（什么什么，你不知道怎么转换模式？oooo,对对，我们还没有操作过，这块并不复杂，接下来会通过专门章节讨论），这里先提供透露一下，转换 standby 数据库到 MAXIMIZE PERFORMANCE 执行下列 SQL 即可：

```
SQL> ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PERFORMANCE;
```

等 standby 切换为新的 primary 之后，你可以再随意更改数据库的保护模式。

你是不是有疑问关于为什么待切换角色的 standby 不能处于 maximum protection 模式呢？这个其实很好理解，我们在第一节学习三种保护模式的时候就介绍过其各自的特点，脑袋瓜好使的同学应该还有印象，maximum protection 模式需要确保绝无数据丢失，因此其对于提交事务对应的 redo 数据一致性要求非常高，另外，如果处于 maximum protection 模式的 primary 数据库仍然与 standby 数据库有数据传输，此时 alter database 语句更改 standby 数据库保护模式会失败，这也是由 maximum protection 模式特性决定的。

下面分别演示 switchover 和 failover 的过程：

一、物理 standby 的 Switchover

注意操作步骤的先后，很关键的哟。

1、检查是否支持 switchover 操作　　--primary 数据库操作

登陆 primary 数据库，查询 v$database 视图的 switchover_status 列。

```
E:\ora10g>set oracle_sid=jssweb

E:\ora10g>sqlplus "/ as sysdba"

SQL*Plus: Release 10.2.0.3.0 - Production on 星期四 12 月 13 09:41:29 2007

Copyright (c) 1982, 2006, Oracle.  All Rights Reserved.

已连接。

SQL> select switchover_status from v$database;

SWITCHOVER_STATUS
-------------------
TO STANDBY
```

如果该列值为"TO STANDBY"则表示 primary 数据库支持转换为 standby 角色，否则的话你就需要重新检查一下 Data Guard 配置，比如看看 LOG_ARCHIVE_DEST_n 之类参数值是否正确有效等等。

2、启动 switchover　--primary 数据库操作

首先将 primary 转换为 standby 的角色，通过下列语句：

```
SQL> alter database commit to switchover to physical standby;

数据库已更改。
```

语句执行完毕后，primary 数据库将会转换为 standby 数据库，并自动备份控制文件到 trace。

3、重启动到 mount　--原 primary 数据库操作

```
SQL> shutdown immediate
ORA-01507: 未装载数据库



ORACLE 例程已经关闭。
SQL> startup mount
ORACLE 例程已经启动。

Total System Global Area  167772160 bytes
Fixed Size                  1289484 bytes
```

```
Variable Size              104858356 bytes
Database Buffers            54525952 bytes
Redo Buffers                 7098368 bytes
数据库装载完毕。
```

4、检查是否支持 switchover 操作　--待转换 standby 数据库操作

待原 primary 切换为 standby 角色之后，检查待转换的 standby 数据库 switchover_status 列，看看是否支持角色转换。

```
E:\ora10g>set oracle_sid=jsspdg

E:\ora10g>sqlplus " / as sysdba"

SQL*Plus: Release 10.2.0.3.0 - Production on 星期四 12 月 13 10:08:15 2007

Copyright (c) 1982, 2006, Oracle.  All Rights Reserved.


已连接。

SQL> select switchover_status from v$database;

SWITCHOVER_STATUS
--------------------
TO PRIMARY

SQL>
```

此时待转换 standby 数据库 switchover_status 列值应该是"TO_PRIMARY"，如否则检查其初始化参数文件中的设置，提示一下，比着原 primary 数据库的初始化参数改改。

5、转换角色到 primary　--待转换 standby 数据库操作

通过下列语句转换 standby 到 primary 角色：

```
SQL> alter database commit to switchover to primary;

数据库已更改。
```

注意：待转换的物理 standby 可以处于 mount 模式或 open read only 模式，但不能处于 open read write 模式。

6、完成转换，打开新的 primary 数据库

```
SQL> alter database open;

数据库已更改。
```

注：如果数据库处于 open read-only 模式的话，需要先 shutdown 然后直接 startup 即可。

7、验证一下

新的 primary 数据库

```
SQL> show parameter db_unique

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_unique_name                       string      jsspdg
SQL> select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
            67

SQL> alter system switch logfile;

系统已更改。

SQL> select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
            68
```

新的 standby 数据库

```
SQL> show parameter db_unique

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_unique_name                       string      jssweb
SQL>    select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
            68
```

转换成功。

二、物理 standby 的 failover

注意几点：

- failover 之后，原 primary 数据库默认不再是 data guard 配置的一部分。
- 多数情况下，其它逻辑/物理 standby 数据库不直接参与 failover 的过程，因此这些数据库不需要做任何操作。
- 某些情况下，新的 primary 数据库配置之后，需要重新创建其它所有的 standby 数据库。

另外，如果待转换角色的 standby 处于 maximum protection 或 maximum availability 模式的话，归档日志应该是连续存在的，这种情况下你可以直接从第 3 步执行，否则建议你按照操作步骤从第 1 步开始执行。

一般情况下 failover 都是表示 primary 数据库瘫痪，最起码也是起不来了，因此这种类型的切换基本上不需要 primary 数据库做什么操作。所以下列步骤中如果有提到 primary 和 standby 执行的，只是建议你如果 primary 还可以用，那就执行一下，即使它能用你却不执行，也没关系，不影响 standby 数据库的切换:)

1、检查归档文件是否连续

查询待转换 standby 数据库的 V$ARCHIVE_GAP 视图，确认归档文件是否连接：

```
SQL> SELECT THREAD#, LOW_SEQUENCE#, HIGH_SEQUENCE# FROM V$ARCHIVE_GAP;

未选定行
```

如果返回的有记录，按照列出的记录号复制对应的归档文件到待转换的 standby 服务器。这一步非常重要，必须确保所有已生成的归档文件均已存在于 standby 服务器，不然可能会数据不一致造成转换时报错。文件复制之后，通过下列命令将其加入数据字典：

```
SQL> ALTER DATABASE REGISTER PHYSICAL LOGFILE 'filespec1';
```

2、检查归档文件是否完整

分别在 primary/standby 执行下列语句：

```
SQL> select distinct thread#,max(sequence#) over(partition by thread#) a from v$archived_log;
```

该语句取得当前数据库各线程已归档文件最大序号，如果 primary 与 standby 最大序号不相同，必须将多出的序号对应的归档文件复制到待转换的 standby 服务器。不过既然是 failover，有可能 primary 数据库此时已经无法打开，甚至无法访问，那你只好听天由命喽，三思在这里替你默念：苍天啊，大地啊，哪路的神仙大姐能来保佑俺们不丢数据呀！

3、启动 failover

执行下列语句：

```
SQL> alter database recover managed standby database finish force;

数据库已更改。
```

FORCE 关键字将会停止当前活动的 RFS 进程，以便立刻执行 failover。

剩下的步骤就与前面 switchover 很相似了

4、切换物理 standby 角色为 primary

```
SQL> alter database commit to switchover to primary;

数据库已更改。
```

5、启动新的 primary 数据库。

如果当前数据库已 mount,直接 open 即可，如果处于 read-only 模式，需要首先 shutdown immediate，然后再直接 startup。

```
SQL> alter database open;

数据库已更改。
```

角色转换工作完成。剩下的是补救措施(针对原 primary 数据库)，由于此时 primary 数据库已经不再是 data guard 配置的一部分，我们需要做的就是尝试看看能否恢复原 primary 数据库，将其改造为新的 standby 服务器。具体操作方式可以分为二类：1.重建　2.备份恢复。所涉及的技术前面的系列文章中均有涉及，此处不再赘述。

第二部分 物理 standby(4)高级管理　2008.1.13

世上没有永恒的主角，能够留住永恒的反是那些默默无闻的小角色，这一节出场的都是重量级选手，它们虽然不是主角，但他们比主角更重要(有时候)。

一、READ ONLY/WRITE 模式打开物理 STANDBY

前面提到关于物理 standby 可以有效分担 primary 数据库压力，提升资源利用，实际上说的就是这个。以 read only 或 read write 模式打开物理 standby，你可以转移一些查询任何啦，备份啦之类的操作到 standby 数据库，以这种方式来分担一些 primary 的压力。下面我们来演示一下，如何切换 standby 数据库的打开模式，其实，非常简单。例如，以 Read-only 模式打开物理 standby：

这里要分两种情况：

1).standby 数据库处于 shutdown 状态

直接 startup 即可。

```
SQL> startup
ORACLE 例程已经启动。
.....
```

2).standby 数据库处于 redo 应用状态。

首先取消 redo 应用：

```
SQL> alter database recover managed standby database cancel;

数据库已更改。
```

然后再打开数据库

```
SQL> alter database open ;

数据库已更改。
```

提示：open 的时候不需要附加 read only 子句，oracle 会根据控制文件判断是否是物理 standby，从而自动启动到 read only 模式，直接 startup 也是同理。

3).如果想从 open 状态再切换回 redo 应用状态，并不需要 shutdown，直接启用 redo 应用即可，例如：

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

STATUS
------------
OPEN

SQL> alter database recover managed standby database disconnect from session;

数据库已更改。
```

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

STATUS
------------
MOUNTED
```

正如演示中我们所看到的，操作有一点点复杂，并且由于只读打开时就不能应用，虽然我们能够查询，但是查询的结果确是与 primary 不同步的，这点大大降低了物理 standby 做报表服务分担主库压力的可能性，对于这点呢，我们有两个解决方案：

1、改用逻辑 standby，由于逻辑 standby 是打开状态下的实时应用，因此数据同步应该是没啥问题了（只要 primary 的数据类型和操作逻辑 standby 都能被支持），当然逻辑 standby 有逻辑 standby 的问题，这个看完后面的逻辑 standby 相关章节，您就明白了。

2、据称 oracle11g 全面改良了物理 standby，最突出的特点就是在 read only 打开模式下，可以边接收边应用了（这下不用担心查询的数据不及时的问题了），您可以考虑升级您的数据库到最新版本，当然新版本也有新版本的问题，比如各种尚未暴露出来的 bug，想想就担心是不是:)

所以你看，做技术其实并不困难，难的是做决择。这么引申过来看一看，老板们不容易啊，怪不得越大的领导脑袋上头发越少呢，为了保持我干净整洁浓密的发型，我我，我还是选择干技术吧~~~~

二、管理影响 standby 的 primary 数据库事件

为预防可能的错误，你必须知道 primary 数据库的某些事件可能影响 standby 数据库，并且了解如何处理。

某些情况下，primary 数据库的某些改动会自动通过 redo 数据传播到 standby 数据库，因此不需要在 standby 数据库做额外的操作，而某些情况，则需要你手工调整。

下列事件会由 redo 传输服务及 redo 应用自动处理，不需要 dba 的干预，分别是：

- ALTER DATABASE ENABLE|DISABLE THREAD 语句(主要针对 rac 环境，目前基本已废弃，因为 ENABLE|DISABLE INSTANCE 子句完全能够实现类似功能)
- 修改表空间状态(例如 read-write 到 read-only，online 到 offline)
- 创建修改删除表空间或数据文件(如果初始化参数 STANDBY_FILE_MANAGEMENT 被设置为 AUTO 的话，这点在前面第一章的时候提到过)

下列事情则需要 dba 手工干预：

1、添加修改删除数据文件或表空间

前面提到了，初始化参数 STANDBY_FILE_MANAGEMENT 可以控制是否自动将 primary 数据库增加删除表空间、数据文件的改动继承到 standby。

- 如果该参数值设置为 auto，则自动创建。
- 如果设置为 manual，则需要手工复制新创建的数据文件到 standby 服务器。

不过需要注意一点，如果数据文件是从其它数据库复制而来(比如通过 tts)，则不管 STANDBY_FILE_MANAGEMENT 参数值如何设置，都必须同时复制到 standby 数据库，并注意要修改 standby 数据库的控制文件。

下面分别通过示例演示 STANDBY_FILE_MANAGEMENT 参数值为 AUTO/MANUAL 值时增加及删除

数据文件时的情形：

1).STANDBY_FILE_MANAGEMENT 设置为 AUTO，增加及删除表空间和数据文件

我们先来看看初始化参数的设置：　----standby 数据库操作

```
SQL> show parameter standby_file

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
standby_file_management              string      AUTO
```

A).增加新的表空间　　--primary 数据库操作

```
SQL> create tablespace mytmp datafile 'e:\ora10g\oradata\jssweb\mytmp01.dbf' size 20m;

表空间已创建。
```

检查刚添加的数据文件

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

NAME
--------------------------------------------
E:\ORA10G\ORADATA\JSSWEB\SYSTEM01.DBF
E:\ORA10G\ORADATA\JSSWEB\UNDOTBS01.DBF
E:\ORA10G\ORADATA\JSSWEB\SYSAUX01.DBF
E:\ORA10G\ORADATA\JSSWEB\USERS01.DBF
E:\ORA10G\ORADATA\JSSWEB\WEBTBS01.DBF
E:\ORA10G\ORADATA\JSSWEB\MYTMP01.DBF

已选择 6 行。
```

切换日志

```
SQL> alter system switch logfile;

系统已更改。
```

B).验证 standby 库　　--standby 数据库操作

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

NAME
--------------------------------------------------
E:\ORA10G\ORADATA\JSSPDG\SYSTEM01.DBF
E:\ORA10G\ORADATA\JSSPDG\UNDOTBS01.DBF
E:\ORA10G\ORADATA\JSSPDG\SYSAUX01.DBF
E:\ORA10G\ORADATA\JSSPDG\USERS01.DBF
E:\ORA10G\ORADATA\JSSPDG\WEBTBS01.DBF
```

```
E:\ORA10G\ORADATA\JSSPDG\MYTMP01.DBF

已选择 6 行。

SQL> select name from v$tablespace;

NAME
------------------------------
SYSTEM
UNDOTBS1
SYSAUX
TEMP
USERS
WEBTBS
MYTMP

已选择 7 行。
```

可以看到，表空间和数据文件已经自动创建，你是不是奇怪为什么数据文件路径自动变成了 jsspdg，赫赫，因为我们设置了 db_file_name_convert 嘛。

C).删除表空间 --primary 数据库操作

```
SQL> drop tablespace mytmp including contents and datafiles;

表空间已删除。

SQL> select name from v$datafile;

NAME
------------------------------------------------
E:\ORA10G\ORADATA\JSSWEB\SYSTEM01.DBF
E:\ORA10G\ORADATA\JSSWEB\UNDOTBS01.DBF
E:\ORA10G\ORADATA\JSSWEB\SYSAUX01.DBF
E:\ORA10G\ORADATA\JSSWEB\USERS01.DBF
E:\ORA10G\ORADATA\JSSWEB\WEBTBS01.DBF

SQL> alter system switch logfile;

系统已更改。
```

提示：使用 including 子句删除表空间时，

D).验证 standby 数据库 --standby 数据库操作

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

NAME
```

```
----------------------------------------------
E:\ORA10G\ORADATA\JSSPDG\SYSTEM01.DBF
E:\ORA10G\ORADATA\JSSPDG\UNDOTBS01.DBF
E:\ORA10G\ORADATA\JSSPDG\SYSAUX01.DBF
E:\ORA10G\ORADATA\JSSPDG\USERS01.DBF
E:\ORA10G\ORADATA\JSSPDG\WEBTBS01.DBF

SQL> select name from v$tablespace;

NAME
------------------------------
SYSTEM
UNDOTBS1
SYSAUX
TEMP
USERS
WEBTBS

已选择 6 行。
```

得出结论，对于初始化参数 STANDBY_FILE_MANAGMENT 设置为 auto 的话，对于表空间和数据文件的操作完全无须 dba 手工干预，primary 和 standby 都能很好的处理。

2).STANDBY_FILE_MANAGEMENT 设置为 MANUAL，增加及删除表空间和数据文件

A).增加新的表空间　　--primary 数据库操作

```
SQL> create tablespace mytmp datafile 'e:\ora10g\oradata\jssweb\mytmp01.dbf' size 20m;

表空间已创建。
```

检查刚添加的数据文件

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

NAME
--------------------------------------------
E:\ORA10G\ORADATA\JSSWEB\SYSTEM01.DBF
E:\ORA10G\ORADATA\JSSWEB\UNDOTBS01.DBF
E:\ORA10G\ORADATA\JSSWEB\SYSAUX01.DBF
E:\ORA10G\ORADATA\JSSWEB\USERS01.DBF
E:\ORA10G\ORADATA\JSSWEB\WEBTBS01.DBF
E:\ORA10G\ORADATA\JSSWEB\MYTMP01.DBF

已选择 6 行。
```

切换日志

```
SQL> alter system switch logfile;

系统已更改。
```

B).验证 standby 库　　--standby 数据库操作

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

NAME
--------------------------------------------------
E:\ORA10G\ORADATA\JSSPDG\SYSTEM01.DBF
E:\ORA10G\ORADATA\JSSPDG\UNDOTBS01.DBF
E:\ORA10G\ORADATA\JSSPDG\SYSAUX01.DBF
E:\ORA10G\ORADATA\JSSPDG\USERS01.DBF
E:\ORA10G\ORADATA\JSSPDG\WEBTBS01.DBF
E:\ORA10G\PRODUCT\10.2.0\DB_1\DATABASE\UNNAMED00006

已选择 6 行。

SQL> select name from v$tablespace;

NAME
------------------------------
SYSTEM
UNDOTBS1
SYSAUX
TEMP
USERS
WEBTBS
MYTMP

已选择 7 行。
```

可以看到，表空间已经自动创建，但是，数据文件却被起了个怪名字，手工修改其与 primary 数据库保持一致，如下(注意执行命令之后手工复制数据文件到 standby)：

```
SQL> alter database create datafile
'E:\ORA10G\PRODUCT\10.2.0\DB_1\DATABASE\UNNAMED00006'
  2  as 'E:\ora10g\oradata\jsspdg\mytmp01.dbf';

数据库已更改。
```

C).删除表空间　　--primary 数据库操作

```
SQL> drop tablespace mytmp including contents and datafiles;

表空间已删除。
```

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

NAME
--------------------------------------------------
E:\ORA10G\ORADATA\JSSWEB\SYSTEM01.DBF
E:\ORA10G\ORADATA\JSSWEB\UNDOTBS01.DBF
E:\ORA10G\ORADATA\JSSWEB\SYSAUX01.DBF
E:\ORA10G\ORADATA\JSSWEB\USERS01.DBF
E:\ORA10G\ORADATA\JSSWEB\WEBTBS01.DBF

SQL> alter system switch logfile;

系统已更改。
```

D).验证 standby 数据库　　--standby 数据库操作

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

NAME
--------------------------------------------------
E:\ORA10G\ORADATA\JSSPDG\SYSTEM01.DBF
E:\ORA10G\ORADATA\JSSPDG\UNDOTBS01.DBF
E:\ORA10G\ORADATA\JSSPDG\SYSAUX01.DBF
E:\ORA10G\ORADATA\JSSPDG\USERS01.DBF
E:\ORA10G\ORADATA\JSSPDG\WEBTBS01.DBF
E:\ORA10G\PRODUCT\10.2.0\DB_1\DATABASE\UNNAMED00006

已选择 6 行。

SQL> select name from v$tablespace;

NAME
------------------------------
SYSTEM
UNDOTBS1
SYSAUX
TEMP
USERS
WEBTBS
MYTMP

已选择 7 行。
```

呀，数据还在啊。赶紧分析分析，查看 alert_jsspdg.log 文件，发现如下(特别注意粗体)：

```
File #6 added to control file as 'UNNAMED00006' because
```

```
the parameter STANDBY_FILE_MANAGEMENT is set to MANUAL
The file should be manually created to continue.
Errors with log E:\ORA10G\ORADATA\JSSPDG\LOG1_753_641301252.ARC
MRP0: Background Media Recovery terminated with error 1274
Fri Jan 18 09:48:45 2008
```

这下明白了，为什么有个 UNNAMED00006 的数据文件，也晓得为啥 standby 数据库没能删除新加的表空间了吧，原来是后台的 redo 应用被停掉了，重启 redo 应用再来看看：

```
SQL> alter database recover managed standby database disconnect from session;

数据库已更改。

SQL> select name from v$datafile;

NAME
--------------------------------------------
E:\ORA10G\ORADATA\JSSPDG\SYSTEM01.DBF
E:\ORA10G\ORADATA\JSSPDG\UNDOTBS01.DBF
E:\ORA10G\ORADATA\JSSPDG\SYSAUX01.DBF
E:\ORA10G\ORADATA\JSSPDG\USERS01.DBF
E:\ORA10G\ORADATA\JSSPDG\WEBTBS01.DBF
```

注意，既使你在 primary 数据库执行删除时加上了 including 子句，在 standby 数据库仍然只会将表空间和数据文件从数据字典中删除，你还需要手工删除表空间涉及的数据文件。

再次得出结论，初始化参数 STANDBY_FILE_MANAGMENT 设置为 manual 的话，对于表空间和数据文件的操作必须有 dba 手工介入，你肯定会问，这太麻烦了，那我干脆配置 dg 的时候直接把初始化参数设置为 auto 不就好了嘛，en，你想的很好，不过三思需要提醒你地是，如果你的存储采用文件系统，那当然没有问题，但是如果采用了裸设备，你就必须将该参数设置为 manual。

2、重命名数据文件

如果 primary 数据库重命令了一个或多个数据文件，该项修改并不会自动传播到 standby 数据库。因为此，如果你想让 standby 和数据文件与 primary 保持一致，那你也只能自己手工操作了。这会儿就算 STANDBY_FILE_MANAGEMENT 也帮不上忙啦，不管它是 auto 还是 manual。

下面通过示例做个演示：

A).将重命名的数据文件所在表空间 offline　--primary 数据库操作

```
SQL> alter tablespace webtbs offline;

表空间已更改。
```

B).手工将数据文件改名(操作系统) --primary 数据库操作

方式多样，不详述。

C).通过命令修改数据字典中的数据文件路径，并 online 表空间 --primary 数据库操作

```
SQL> alter tablespace webtbs rename datafile
  2  'E:\ORA10G\ORADATA\JSSWEB\WEBTBS01.DBF' to
  3  'E:\ORA10G\ORADATA\JSSWEB\TBSWEB01.DBF';

表空间已更改。

SQL> alter tablespace webtbs online;

表空间已更改。
```

D).暂停 redo 应用，并 shutdown --standby 数据库操作

```
SQL> alter database recover managed standby database cancel;

数据库已更改。

SQL> shutdown immediate
ORA-01109: 数据库未打开
.....
```

E).手工将数据文件改名(操作系统) --standby 数据库操作

方式多样，不详述。

F).重启 standby，修改数据文件路径(数据字典) --standby 数据库操作

```
SQL> startup mount
ORACLE 例程已经启动。

Total System Global Area  167772160 bytes
Fixed Size                  1289484 bytes
Variable Size             150995700 bytes
Database Buffers            8388608 bytes
Redo Buffers                7098368 bytes
数据库装载完毕。
SQL> alter database rename file
  2  'E:\ORA10G\ORADATA\JSSPDG\WEBTBS01.DBF' to
  3  'E:\ORA10G\ORADATA\JSSPDG\TBSWEB01.DBF';

数据库已更改。
```

G).重新启动 redo 应用。

```
SQL> alter database recover managed standby database disconnect from session;

数据库已更改。
```

H).切换日志　--primary 数据库操作

```
SQL> alter system switch logfile;

系统已更改。
```

峻工~~~~

3、添加或删除 Online redo logs

数据库调优时极有可能会涉及到重置日志文件大小或增加删除日志组等操作，基本上这种操作不会传播到 standby 数据库，也不会影响到 standby 数据库的运行，但是如果你不注意其中的关系，造成的影响可能会很深远，

比如，我们假设我们的一台 primary 数据库拥有 5 组 online redo 文件，standby 数据库拥有 2 组，当你执行 switch over 之后，新的 primary 执行归档的频率会比 standby 高的多，因此，当你在 primary 数据库增加或移除 online redologs 时，一定记的手工同步一相 standby 数据库中相关的设置。

这就是我们前面提到的 standby redologs 与 online redologs 之间的关系，即保证 standby redologs 比 online redologs 要至少多一组。

操作的过程很简单(总不会复杂过添加删除数据文件)，这里就不演示了，需要你注意的就是在 standby 做操作前务必将 STANDBY_FILE_MANAGEMENT 设置为 MANUAL。

三、对 open resetlogs 的 primary 数据库 standby 的恢复

当 primary 数据库被以 resetlogs 打开之后，dg 提供了一些方案，能够让你快速的恢复物理 standby，当然这是有条件的，不可能所有的情况都可以快速恢复。我们都知道 alter database open resetlogs 之后，数据库的 scn 被重置，也就是此时其 redo 数据也会从头开始。当物理 standby 接收到新的 redo 数据时，redo 应用会自动获取这部分 redo 数据。对于物理 standby 而言，只要数据库没有应用 resetlogs 之后的 redo 数据，那么这个过程是不需要 dba 手工参与的。

下表描述其它情况下如何同步 standby 与 primary 数据库。

Standby 数据库状态	Standby 服务器操作	解决方案
没有应用 resetlog 之前的 redo 数据	自动应用新的 redo 数据	无须手工介入
应用了 resetlog 之后的 redo 数据，不过 standby 打开了 flashback。	可以应用，不过需要 dba 手工介入	1.手工 flashback 到应用之前 2.重启 redo 应用，以重新接收新的 redo 数据。
应用了 resetlog 之后的 redo 数据，而且没有 flashback。	完全无法应用	重建物理 standby 是唯一的选择

很绕是吧，举个例子你就明白了：

假设 primary 数据库当前生成的 archive sequence#如下：...26,27,28，然后在 28 的时候执行了 resetlogs，又生成了新的 1,2,3.....，那么 standby 能够正常接收并应用 26,27,28 及新产生的 1,2,3....

如果 primary 数据库在 28 的时候发生数据出现故障，recover 到 27，然后 resetlogs，又生成了新的 1,2,3.....，这个时候(大家注意，招子放亮点)：如果 standby 还没有应用 28，刚刚应用到 27，则 standby 还可以继续接收新

的 redo 数据 1,2,3.....并应用。

如果此时不幸，standby 由于是实时应用，已经应用了 28 的 redo 数据，那么如果 standby 打开了 flashback，不幸中的万幸啊，这时候只需要 dba 手工介绍先 flashback 到 27,然后再接收并应用新的 1,2,3....

如果此时非常不幸，standby 由于是实时应用，已经应用了 28 的 redo 数据，并且 standby 也没有打开 flashback，那么，重建物理 standby 是你唯一的选择。

这下大家都明白了吧，赶紧起立鼓掌感谢 yangtingkun 大大的友情客串及形象示例，很通俗，很易懂:)。

四、监控 primary/standby 数据库

本节主要介绍一些监控 dg 配置的方式，先给大家提供一个表格(描述不同事件的不同信息监控途径)：

primary 数据库事件	primary 监控途径	standby 监控途径
带有 enable|disable thread 子句的 alter database 命令	➢ Alert.log ➢ V$THREAD	➢ Alert.log
当前数据库角色，保护模式，保护级别，switchover 状态，failover 快速启动信息等	➢ V$DATABASE	➢ V$DATABASE
Redo log 切换	➢ Alert.log ➢ V$LOG ➢ V$LOGFILE 的 status 列	➢ Alert.log
重建控制文件	➢ Alert log	➢ Alert log
手动执行恢复	➢ Alert log	➢ Alert log
表空间状态修改(read-write/read-only,online/offline)	➢ DBA_TABLESPACES ➢ Alert log	➢ V$RECOVER_FILE
创建删除表空间或数据文件	➢ DBA_DATA_FILES ➢ Alert log	➢ V$DATAFILE ➢ Alert log
表空间或数据文件 offline	➢ V$RECOVER_FILE ➢ Alert log ➢ DBA_TABLESPACES	➢ V$RECOVER_FILE ➢ DBA_TABLESPACES
重命名数据文件	➢ V$DATAFILE ➢ Alert log	➢ V$DATAFILE ➢ Alert log
未被日志记录或不可恢复的操作	➢ V$DATAFILE view ➢ V$DATABASE view	➢ Alert log
恢复的进程	➢ V$ARCHIVE_DEST_STATUS ➢ Alert log	➢ V$ARCHIVED_LOG ➢ V$LOG_HISTORY ➢ V$MANAGED_STANDBY ➢ Alert log
Redo 传输的状态和进度	➢ V$ARCHIVE_DEST_STATUS ➢ V$ARCHIVED_LOG ➢ V$ARCHIVE_DEST ➢ Alert log	➢ V$ARCHIVED_LOG ➢ Alert log

数据文件自动扩展	➢ Alert log	➢ Alert log
执行 OPEN RESETLOGS 或 CLEAR UNARCHIVED LOGFILES	➢ Alert log	➢ Alert log
修改初始化参数	➢ Alert log	➢ Alert log

概括起来主要通过二个方面：

1、Alert Log

一句话：一定要养成有事没事定期不定期随时查看 alert.log 的好习惯同时特别注意 alert 中的提示通常不经意间会发现它的提示能够让你的思路豁然开朗。

2、动态性能视图

先也是一句话：做为 oracle 自己自觉主动维护的一批虚拟表它的作用非常明显通过它可以及时获得当前数据库状态及处理进度总之好处多多也需特别关注后面示例也会多处用到大家要擦亮双眼。

- **先来点与恢复进度相关的 v$视图应用示例：**

A).查看进程的活动状况---v$managed_standby

该视图就是专为显示 standby 数据库相关进程的当前状态信息，例如：

```
SQL> select process,client_process,sequence#,status from v$managed_standby;

PROCESS   CLIENT_P  SEQUENCE# STATUS
--------- -------- ---------- ------------
ARCH      ARCH            763 CLOSING
ARCH      ARCH            762 CLOSING
MRP0      N/A             764 WAIT_FOR_LOG
RFS       LGWR            764 IDLE
RFS       N/A               0 IDLE
```

PROCESS 列显示进程信息

CLIENT_PROCESS 列显示对应的主数据库中的进程

SEQUENCE#列显示归档 redo 的序列号

STATUS 列显示的进程状态

通过上述查询可以得知 primary 开了两个归档进程，使用 lgwr 同步传输方式与 standby 通信，已经接收完 763 的日志，正等待 764。

B).确认 redo 应用进度---v$archive_dest_status

该视图显示归档文件路径配置信息及 redo 的应用情况等，例如：

```
SQL> select dest_name,archived_thread#,archived_seq#,applied_thread#,applied_seq#,db_unique_name
  2  from v$archive_dest_status where status='VALID';

DEST_NAME                 ARCHIVED_THREAD# ARCHIVED_SEQ# APPLIED_THREAD#
APPLIED_SEQ# DB_UNIQUE_
-------------------- ---------------- ------------- --------------- ------------ ----------
LOG_ARCHIVE_DEST_1                   1             765               0            0
jsspdg
```

```
    LOG_ARCHIVE_DEST_2                       0                 0                 0                 0
jssweb
    STANDBY_ARCHIVE_DEST                     1               764                 1
764 NONE
```

C).检查归档文件路径及创建信息---v$archived_log

该视图查询 standby 数据库归档文件的一些附加信息，比如文件创建时间啦，创建进程啦，归档序号啦，是否被应用啦之类，例如：

```
    SQL> select name,creator,sequence#,applied,completion_time from v$archived_log;

    NAME                                               CREATOR    SEQUENCE# APP COMPLETION_TIM
    -------------------------------------------------- ------- ---------- --- --------------
    E:\ORA10G\ORADATA\JSSPDG\LOG1_750_641301252.ARC     ARCH         750 YES 18-1 月  -08
    E:\ORA10G\ORADATA\JSSPDG\LOG1_749_641301252.ARC     ARCH         749 YES 18-1 月  -08
    E:\ORA10G\ORADATA\JSSPDG\LOG1_751_641301252.ARC     ARCH         751 YES 18-1 月  -08
    E:\ORA10G\ORADATA\JSSPDG\LOG1_752_641301252.ARC     ARCH         752 YES 18-1 月  -08
    E:\ORA10G\ORADATA\JSSPDG\LOG1_753_641301252.ARC     ARCH         753 YES 18-1 月  -08
    E:\ORA10G\ORADATA\JSSPDG\LOG1_754_641301252.ARC     ARCH         754 YES 18-1 月  -08
```

D).查询归档历史---v$log_history

该视图查询 standby 库中所有已被应用的归档文件信息(不论该归档文件是否还存在)，例如：

```
    SQL> select first_time,first_change#,next_change#,sequence# from v$log_history;

    FIRST_TIME          FIRST_CHANGE# NEXT_CHANGE#  SEQUENCE#
    ------------------- ------------- ------------ ----------
    2008-01-03 12:00:51        499709       528572         18
    2008-01-08 09:54:42        528572       539402         19
    2008-01-08 22:00:06        539402       547161         20
    2008-01-09 01:05:57        547161       560393         21
    2008-01-09 10:13:53        560393       561070         22
```

- **再来点与 log 应用相关的 v$视图应用示例：**

A).查询当前数据的基本信息---v$database 信息。

例如，查询数据库角色，保护模式，保护级别等：

```
    SQL>                                                                                select
database_role,db_unique_name,open_mode,protection_mode,protection_level,switchover_status
      2  from v$database;

    DATABASE_ROLE          DB_UNIQUE_NAME                                 OPEN_MODE
PROTECTION_MODE        PROTECTION_LEVEL       SWITCHOVER_STATUS
    ---------------- ------------------------------ ---------- -------------------- -------------------- --------------------
    PHYSICAL STANDBY jsspdg                                     MOUNTED         MAXIMUM
AVAILABILITY MAXIMUM AVAILABILITY SESSIONS ACTIVE
```

再比如，查询 failover 后快速启动的信息

```
SQL> select fs_failover_status,fs_failover_current_target,fs_failover_threshold,
  2  fs_failover_observer_present from v$database;

FS_FAILOVER_STATUS                                FS_FAILOVER_CURRENT_TARGET
FS_FAILOVER_THRESHOLD FS_FAIL
-------------------- ------------------------------ -------------------- -------
DISABLED                                                                  0
```

B).检查应用模式(是否启用了实时应用)---v$archive_dest_status

查询 v$archive_dest_status 视图，如果打开了实时应用，则 recovery_mode 会显示为：MANAGED REAL TIME APPLY，例如：

```
SQL> select recovery_mode from v$archive_dest_status where dest_id=2;

RECOVERY_MODE
----------------------
MANAGED REAL TIME APPLY
```

C).Data guard 事件---v$dataguard_status

该视图显示那些被自动触发写入 alert.log 或服务器 trace 文件的事件。通常是在你不便访问到服务器查询 alert.log 时，可以临时访问本视图查看一些与 dataguard 相关的信息，例如：

```
SQL> select message from v$dataguard_status;

MESSAGE
--------------------------------------------------------------------------------
ARC0: Archival started
ARC1: Archival started
ARC0: Becoming the 'no FAL' ARCH
ARC0: Becoming the 'no SRL' ARCH
ARC1: Becoming the heartbeat ARCH
Attempt to start background Managed Standby Recovery process
MRP0: Background Managed Standby Recovery process started
Managed Standby Recovery not using Real Time Apply
Media Recovery Waiting for thread 1 sequence 761
```

五、调整物理 standby log 应用频率

调整应用频率说白了就是调整 io 读取能力，所以通常我们可以从以下几个方面着手：

1、设置 recover 并行度

在介质恢复或 redo 应用期间，都需要读取重做日志文件，默认都是串行恢复，我们可以在执行 recover 的时候加上 parallel 子句来指定并行度，提高读取和应用的性能，例如：

```
SQL> alter database recover managed standby database parallel 2 disconnect from session;
```

推荐 parallel 的值是#CPUs*2;

2、加快 redo 应用频繁

设置初始化参数 DB_BLOCK_CHECKING=FALSE 能够提高 2 倍左右的应用效率，该参数是验证数据块是否有效，对于 standby 禁止验证基本上还是可以接受的，另外还有一个关联初始化参数 DB_BLOCK_CHECKSUM，建议该参数在 primary 和 standby 都设置为 true。

3、设置 PARALLEL_EXECUTION_MESSAGE_SIZE

如果打开了并行恢复，适当提高初始化参数：PARALLEL_EXECUTION_MESSAGE_SIZE 的参数值，比如 4096 也能提高大概 20%左右的性能，不过需要注意增大这个参数的参数值可能会占用更多内存。

4、优化磁盘 I/O

在恢复期间最大瓶颈就是 I/O 读写，要缓解这个瓶颈，使用本地异步 I/O 并设置初始化参数 DISK_ASYNCH_IO=TRUE 会有所帮助。DISK_ASYNCH_IO 参数控制到数据文件的磁盘 I/O 是否异步。某些情况下异步 I/O 能降低数据库文件并行读取，提高整个恢复时间。

第三部分 逻辑 standby(1)创建步骤 2008.02.03

一、准备工作

正如我们打小就被叮嘱饭前一定要洗手，在创建逻辑 standby 之前，准备工作同样必不可少。

在创建逻辑 standby 之前，首先检查 primary 数据库的状态，确保 primary 数据库已经为创建逻辑 standby 做好了全部准备工作，比如说是否启动了归档，是否启用了 forced logging 等，这部分可以参考创建物理 standby 时的准备工作。

除此之外呢，由于逻辑 standby 是通过 sql 应用来保持与 primary 数据库的同步。sql 应用与 redo 应用是有很大区别地，这事儿咱们前面提到过，redo 应用实际上是物理 standby 端进行 recover，sql 应用则是分析 redo 文件，将其转换为 sql 语句在逻辑 standby 端执行，因此，需要注意：

- **并非所有的数据类型都能被逻辑 standby 支持；**

  支持的数据类型有：

  BINARY_DOUBLE、BINARY_FLOAT、BLOB、CHAR、CLOB and NCLOB、DATE、INTERVAL YEAR TO MONTH、INTERVAL DAY TO SECOND、LONG、LONG RAW、NCHAR、NUMBER、NVARCHAR2、RAW、TIMESTAMP、TIMESTAMP WITH LOCAL TIMEZONE、TIMESTAMP WITH TIMEZONE、VARCHAR2 and VARCHAR

  提示：

  下列类型在获取 standby 支持时需要注意兼容性：

  - clob，需要 primary 数据库的兼容级别运行于 10.1 或更高
  - 含 lob 字段的索引组织表(IOT)，需要 primary 数据库的兼容级别运行于 10.2 或更高
  - 不含 lob 字段的索引组织表(IOT)，需要 primary 数据库的兼容级别运行于 10.1 或更高

  不支持的数据类型有：

  BFILE、Encrypted columns、ROWID, UROWID、XMLType、对象类型、VARRAYS、嵌套表、自定义类型。

洗手杀菌可以用肥皂或洗手液，检查数据库是否有不被逻辑 standby 支持的对象也同样有简单方式，我们可以通过查询视图 DBA_LOGSTDBY_UNSUPPORTED 来确定主数据库中是否含有不支持的对象：

```
SQL> SELECT * FROM DBA_LOGSTDBY_UNSUPPORTED;
```

提示：关于 DBA_LOGSTDBY_UNSUPPORTED

该视图显示包含不被支持的数据类型的表的列名及该列的数据类型。注意该视图的 ATTRIBUTES 列，列值会显示表不被 sql 应用支持的原因。

- **并非所有的存储类型都能被逻辑 standby 支持；**

  支持簇表(Cluster tables)、索引组织表(Index-organized tables)、堆组织表(Heap-organized tables)，不支持段压缩(segment compression)存储类型

- **并非所有的 pl/sql 包都能被 SQL 应用支持。**

  那些可能修改系统元数据的包不会被 sql 应用支持，因此即使它们在 primary 执行过，并且被成功传输到逻辑 standby 端，也不会执行，例如：DBMS_JAVA, DBMS_REGISTRY, DBMS_ALERT, DBMS_SPACE_ADMIN, DBMS_REFRESH, DBMS_REDEFINITION, DBMS_SCHEDULER, and DBMS_AQ

等。

只有 dbms_job 例外，primary 数据库的 jobs 会被复制到逻辑 standby，不过在 standby 数据库不会执行这些 job。

- **并非所有的 sql 语句都能在逻辑 standby 执行；**

  默认情况下，下列 sql 语句在逻辑 standby 会被 sql 应用自动跳过：

  ALTER DATABASE
  ALTER MATERIALIZED VIEW
  ALTER MATERIALIZED VIEW LOG
  ALTER SESSION
  ALTER SYSTEM
  CREATE CONTROL FILE
  CREATE DATABASE
  CREATE DATABASE LINK
  CREATE PFILE FROM SPFILE
  CREATE MATERIALIZED VIEW
  CREATE MATERIALIZED VIEW LOG
  CREATE SCHEMA AUTHORIZATION
  CREATE SPFILE FROM PFILE
  DROP DATABASE LINK
  DROP MATERIALIZED VIEW
  DROP MATERIALIZED VIEW LOG
  EXPLAIN
  LOCK TABLE
  SET CONSTRAINTS
  SET ROLE
  SET TRANSACTION

  另外，还有一大批 ddl 操作，同样也不会在逻辑 standby 端执行，由于数目较重，此处不再一一列举，感兴趣的话请 google 查看官方文档。

- **并非所有的 dml 操作都能在逻辑 standby 端 SQL 应用；**

  维护逻辑 standby 与 primary 的数据库同步是通过 sql 应用实现，SQL 应用转换的 SQL 语句在执行时，对于 insert 还好说，对于 update,delete 操作则必须能够唯一定位到数据库待更新的那条记录。问题就在这里，如果 primary 库中表设置不当，可能就无法确认唯一条件。

  你可能会说可以通过 rowid 唯一嘛!! 同学，千万要谨记啊，逻辑 standby，为啥叫逻辑 standby 呢，它跟物理 standby 有啥区别呢，就是因为它只是逻辑上与 primary 数据库相同，物理上可能与 primary 数据库存在相当大差异，一定要认识到，逻辑 standby 的物理结构与 primary 是不相同的(即使初始逻辑 standby 是通过 primary 的备份创建)，因此想通过 rowid 更新显然是不好使的，就不能再将其做为唯一条件。那怎么办泥，OK，话题被引入，下面请听三思向您一一道来：

如何确保 primary 库中各表的行可被唯一标识

Oracle 通过主键、唯一索引/约束补充日志(supplemental logging)来确定待更新逻辑 standby 库中的行。当数据库启用了补充日志(supplemental logging)，每一条 update 语句写 redo 的时候会附加列值唯一信息，

比如：

- 如果表定义了主键，则主键值会随同被更新列一起做为 update 语句的一部分，以便执行时区别哪些列应该被更新。
- 如果没有主键，则非空的唯一索引/约束会随同被更新列做为 update 语句的一部分，以便执行时区分哪些列应该被更新，如果该表有多个唯一索引/约束，则 oracle 自动选择最短的那个。
- 如果表即无主键，也没有定义唯一索引/约束，所有可定长度的列连同被更新列作为 update 语句的一部分。更明确的话，可定长度的列是指那些除:long,lob,long raw,object type,collection 类型外的列。

确定在主数据库上，补充日志是否被启用，可以查询 v$database，如下：

```
SQL> select supplemental_log_data_pk,supplemental_log_data_ui from v$database;

SUP SUP
--- ---
YES YES
```

因此，Oracle 建议你为表创建一个主键或非空的唯一索引/约束，以尽可能确保 sql 应用能够有效应用 redo 数据，更新逻辑 standby 数据库。

执行下列语句检查 sql 应用能否唯一识别表列，找出不被支持的表

```
SQL> SELECT OWNER, TABLE_NAME FROM DBA_LOGSTDBY_NOT_UNIQUE
  2> WHERE (OWNER, TABLE_NAME) NOT IN
  3> (SELECT DISTINCT OWNER, TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED)
  4> AND BAD_COLLUMN = 'Y';
```

提示：

关于 DBA_LOGSTDBY_NOT_UNIQUE

该视图显示所有即没主键也没唯一索引的表。如果表中的列包括足够多的信息通常也可支持在逻辑 standby 的更新，不被支持的表通常是由于列的定义包含了不支持的数据类型。

注意 BAD_COLUMN 列值，该列有两个值：

Y：表示该表中有采用大数据类型的字段，比如 LONG 啦，CLOB 啦之类。如果表中除 log 列某些行记录完全匹配，则该表无法成功应用于逻辑 standby。standby 会尝试维护这些表，不过你必须保证应用不允许

N：表示该表拥有足够的信息，能够支持在逻辑 standby 的更新，不过仍然建议你为该表创建一个主键或者唯一索引/约束以提高 log 应用效率。

假设某张表你可以确认数据是唯一的，但是因为效率方面的考虑，不想为其创建主键或唯一约束，，怎么办呢，没关系，oracle 想到了这一点，你可以创建一个 disable 的 primary-key rely 约束：

关于 primary-key RELY 约束：

如果你能够确认表中的行是唯一的，那么可以为该表创建 rely 的主键，RELY 约束并不会造成系统维护主键的开销，主你对一个表创建了 rely 约束，系统则会假定该表中的行是唯一，这样能够提供 sql 应用时的性能。但是需要注意，由于 rely 的主键约束只是假定唯一，如果实际并不唯一的话，有可能会造成错误的更新哟。

创建 rely 的主键约束非常简单，只要在标准的创建语句后加上 RELY DISABLE 即可，例如：

```
SQL> alter table jss.b add primary key (id) rely disable;

表已更改。
```

二、创建步骤

1、创建物理 standby

最方便的创建逻辑 standby 的方式就是先创建一个物理 standby，然后再将其转换成逻辑 standby，因此第一步就是先创建一个物理 standby。注意，在将其转换为逻辑 standby 前，可以随时启动和应用 redo，但是如果你决定将其转换为逻辑 standby，就必须先停止该物理 standby 的 redo 应用，以避免提前应用含 LogMiner 字典的 redo 数据，造成转换为逻辑 standby 后，sql 应用时 logMiner 字典数据不足而影响到逻辑 standby 与 primary 的正常同步。

2、设置 primary 数据库

在前面创建 standby 时我们曾经设置过无数个初始化参数用于 primary 与物理 standby 的角色切换，我想说的是，对于逻辑 standby 的角色切换，那些参数同样好使。

不过注意，如果希望 primary 数据库能够正常切换为逻辑 standby 角色的话，那么你还需要设置相应的 log_archive_dest_N，并且 valid_for 属性，需要更改成：(STANDBY_LOGFILES,STANDBY_ROLE)。

然后需要生成 LogMiner 字典信息，通过执行下列语句生成(**务必执行**)：

```
SQL> EXECUTE DBMS_LOGSTDBY.BUILD;
```

该过程专门用于生成记录的元数据信息到 redo log，这样改动才会被传输到逻辑 standby，然后才会被逻辑 standby 进行 SQL 应用。

提示：

- 该过程会自动启用 primary 数据库的补充日志(supplemental logging)功能(如果未启用的话)。
- 该过程执行需要等待当前所有事务完成，因此如果当前有较长的事务运行，可能该过程执行也需要多花一些等待时间。
- 该过程是通过闪回查询的方式来获取数据字典的一致性，因此 oracle 初始化参数 UNDO_RETENTION 值需要设置的足够大。

3、转换物理 standby 为逻辑 standby

执行下列语句，转换物理 standby 为逻辑 standby：

```
SQL> alter database recover to logical standby NEW_DBNAME;
```

关于 db_name(注意哟，这可不是 db_unique_name，不同于物理 standby，逻辑 standby 是一个全新的数据库，因此建议你指定一个唯一的，与 primary 不同的数据库名)，如果当前使用 spfile，则数据库会自动修改其中的相关信息，如果使用的 pfile，在下次执行 shutdown 的时候 oracle 会提示你去修改 db_name 初始化参数的值。

提示：执行该语句前务必确保已经暂停了 redo 应用，另外转换是单向的，即只能由物理 standby 向逻辑 standby 转换，而不能由逻辑 standby 转成物理 standby。这并不仅仅是因为 dbname 发生了修改，更主要

的原因是逻辑 standby 仅是数据与 primary 一致，其它如存储结构，scn 等基于 dbid 都不一相同。

另外，该语句执行过程中，需要应用全部的 LogMiner 字典相关的 redo 数据。这部分操作完全依赖于 primary 数据库 DBMS_LOGSTDBY.BUILD 的执行以及传输到 standby 后有多少数据需要被应用。如果 primary 数据库执行 DBMS_LOGSTDBY.BUILD 失败，则转换操作也不会有结果，这时候你恐怕不得不先 cancel 掉它，解决 primary 数据库的问题之后再尝试执行转换。

4、重建逻辑 standby 的密码文件

主要是由于转换操作修改了数据库名，因此密码文件也需要重建，这个操作我们做的比较多，这里就不详述了。

5、调整逻辑 standby 初始化参数

之所以要调整初始化参数，一方面是由于此处我们的逻辑 standby 是从物理 standby 转换来的，某些参数并不适合甚至可能造成错误，比如 log_archive_dest_n 参数的设置。另一方面，由于逻辑 standby 会有读写操作，因此需要读写本地 online redologs 及并产生 archivelogs，务必需要注意本地的 archivelogs 路径不要与应用接收自 primary 数据库的 redo 数据生成的 archivelogs 路径冲突。当然归根结底是因为逻辑 standby 是从物理 standby 转换而来，因此 standby 的初始化参数就需要第二次调整(第一次是创建物理 standby)，这里为什么要选择从物理 standby 转换呢？很简单，因为前面测试过程中创建了两个 standby，所以我觉着直接转换一个当成逻辑 standby，操作更省事儿：）

当然我相信，看完这个系列，如果你对于创建的流程能够非常清晰，完全可以跳过先创建物理 standby 的过程，直接创建逻辑 standby。

关于修改初始化参数的方式有多种，通过 alter system set 也可以，或者先生成 pfile 修改相关参数，然后再根据修改过的 pfile 生成 spfile 也可以。

6、打开逻辑 standby

由于逻辑 standby 与 primary 数据库事务并不一致，因此第一次打开时必须指定 resetlogs 选择，如下：

```
SQL> alter database open resetlogs;
```

然后可以通过执行下列 sql 命令应用 redo 数据：

```
SQL> alter database start logical standby apply immediate;
```

如果想停止逻辑 standby 的 sql 应用，可以通过下列命令：

```
SQL> alter database stop logical standby apply immediate;
```

第三部分 逻辑 standby(2)创建示例 2008.02.18

假设当前架构为一个 primary+二个物理 standby，我们转换其中一个物理 standby 成为逻辑 standby，专用于查询服务，另一个物理 standby 用于执行备份操作及提供灾备。这里我们直接借用之前创建的物理 standby，只演示创建过程，我们假设当前 primary 数据库状态良好，没有任何不被逻辑 standby 支持的对象或类型。

为了方便区分当前操作的数据库，我们设置一下操作符：

SQL> set sqlprompt JSSWEB> --表示 primary 数据库

SQL> set sqlprompt JSSPDG> --表示物理 standby

SQL> set sqlprompt JSSLDG> --表示逻辑 standby

一、创建物理 standby

此步跳过，如有不明，具体可参考第二部分。

提示：表忘记暂停该 standby 的 redo 应用

```
JSSLDG>alter database recover managed standby database cancel;

数据库已更改。
```

二、设置 primary 数据库

由于有前期创建物理 standby 时的基础，此处 primary 数据库的初始化参数可以不做修改，最重要的是不要忘记生成 LogMiner 字典信息。

```
JSSWEB>execute dbms_logstdby.build;

PL/SQL 过程已成功完成。
```

三、转换物理 standby 为逻辑 standby

执行下列语句，转换物理 standby 为逻辑 standby：

```
JSSLDG>show parameter db_name;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_name                              string      jssweb

JSSLDG>alter database recover to logical standby jssldg;

数据库已更改。

JSSLDG>shutdown immediate
ORA-01507: 未装载数据库


ORACLE 例程已经关闭。
JSSLDG>startup mount;
```

```
ORACLE 例程已经启动。

Total System Global Area    167772160 bytes
Fixed Size                    1289484 bytes
Variable Size                79692532 bytes
Database Buffers             79691776 bytes
Redo Buffers                  7098368 bytes
数据库装载完毕。
JSSLDG>show parameter db_name;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_name                              string      JSSLDG
JSSLDG>select database_role from v$database;

DATABASE_ROLE
----------------
LOGICAL STANDBY
```

四、重建逻辑 standby 的密码文件

```
E:\ora10g>orapwd file=e:\ora10g\product\10.2.0\db_1\database\PWDjssldg.ora password=verysafe entries=30
```

注意保持 sys 密码与 primary 数据库一致。

五、调整逻辑 standby 初始化参数

注意归档文件路径不要冲突：

```
JSSLDG>alter system set log_archive_dest_1='location=E:\ora10g\oradata\JSSLDG\arc\
valid_for=(online_logfiles,all_roles) db_unique_name=JSSLDG';

系统已更改。

JSSLDG>alter system set log_archive_dest_2='location=E:\ora10g\oradata\JSSLDG\std\
valid_for=(standby_logfiles,standby_role) db_unique_name=JSSLDG';

系统已更改。
```

另外，由于之前我们创建 JSSLDG 时并未创建 standby redologs，但对于逻辑 standby 的 sql 应用，standby redologs 是必须的，因此我们在此处也要为该 standby 创建几组 standby redologs：

```
JSSLDG>alter database add standby logfile group 4 ('E:\ora10g\oradata\JSSLDG\standbyrd01.log') size 20m;

数据库已更改。

JSSLDG>alter database add standby logfile group 5 ('E:\ora10g\oradata\JSSLDG\standbyrd02.log') size 20m;
```

```
数据库已更改。

JSSLDG>alter database add standby logfile group 6 ('E:\ora10g\oradata\JSSLDG\standbyrd03.log') size 20m;

数据库已更改。

JSSLDG>select member from v$logfile;

MEMBER
------------------------------------------------------------
E:\ORA10G\ORADATA\JSSLDG\REDO01.LOG
E:\ORA10G\ORADATA\JSSLDG\REDO02.LOG
E:\ORA10G\ORADATA\JSSLDG\REDO03.LOG
E:\ORA10G\ORADATA\JSSLDG\STANDBYRD01.LOG
E:\ORA10G\ORADATA\JSSLDG\STANDBYRD02.LOG
E:\ORA10G\ORADATA\JSSLDG\STANDBYRD03.LOG

已选择 6 行。
```

六、打开逻辑 standby

由于逻辑 standby 与 primary 数据库事务并不一致，因此第一次打开时必须指定 resetlogs 选择，如下：

```
SQL> alter database open resetlogs;

数据库已更改。
```

然后执行下列 sql 命令应用 redo 数据：

```
SQL> alter database start logical standby apply immediate;

数据库已更改。
```

七、检查一下

首先在 primary 数据库执行：

```
JSSWEB> select *from jss.b;

        ID
----------
         1
         2
         3

已选择 3 行。

JSSWEB> insert into jss.b values (4);
```

```
已创建 1 行。

JSSWEB> insert into b values (5);

已创建 1 行。

JSSWEB> insert into b values (6);

已创建 1 行。

JSSWEB> commit;

提交完成。

JSSWEB> alter system switch logfile;

系统已更改。
```

查询物理 standby 的同步情况，由于物理 standby 处于 mount 状态，无法直接查询，因此我们需要先暂停 redo 应用，然后以 read only 模式打开数据库再执行查询：

```
JSSPDG>alter database recover managed standby database cancel;

数据库已更改。

JSSPDG>alter database open read only;

数据库已更改。

JSSPDG>select * from jss.b;

        ID
----------
         1
         2
         3
         4
         5
         6

已选择 6 行。
```

查询逻辑 standby 的同步情况：

```
JSSLDG>select * from jss.b;
```

```
        ID
----------
         1
         2
         3
         4
         5
         6

已选择 6 行。
```

提示：细心观察，发现逻辑 standby 有一点很好，从 primary 接收到的 redo 文件，应用过之后会自动删除，节省磁盘空间。

Ok,逻辑 standby 也创建完成了，我们再回过头来回忆回忆我们最开始的假设：

对于相机拍照而言，有种傻瓜相机功能强大而操作简便，而对于素描，即使是最简单的画法，也需要相当多的练习才能掌握。这个细节是不是也说明逻辑 standby 相比物理 standby 需要操作者拥有更多的操作技能呢？

现在看起来，操作呢相比物理 standby 是稍稍复杂了一点点，但机理呢与物理 standby 大同小异，功能呢也不见的就比物理 standby 强到哪里，主要是前期准备工作略嫌繁琐(尤其你的数据库系统比较宏大时，毕竟有那么多支持和不支持的数据类型/操作/语句需要 dba 手工处理)，这么看来，画画的仿佛是要比搞摄影的更讲究基本功啊，不过事物要辩证着看，爱好摄影的朋友千万莫因此而感\到沮丧，从实用角度看，搞摄影不知要比画画强多少倍啊，效率在那摆着呢，要出片子按下快门就成啦！对于 standby 也是如此，你究竟是想要物理的，还是想要逻辑的呢，这是个问题~~~~~

第三部分 逻辑 standby(3)角色转换 2008.02.22

关于角色转换的一些概念在物理 standby 章节的时候已经讲了很多，在概念和操作方式上二者基本一致，不过如果你真正深刻理解了物理 standby 和逻辑 standby，你会意识到，对于逻辑 standby 而言，不管是 switchover 还是 failover，怎么操作起来，都这么怪怪的呢~~~

逻辑 standby 之 switchover

要在 primary 和逻辑 standby 之间切换角色，一般是从操作 primary 开始。

提示：

如果 primary 或逻辑 standby 是 rac 结构，切记只保留一个实例启动，其它实例全部 shutdown。等角色转换操作完成之后再启动其它实例，角色转换的操作会自动传播到这些实例上，并不需要你再对这些实例单独做处理。

一、准备工作

1、检查 primary 和逻辑 standby 的初始化参数设置，常规的检查包括：

- 确保 fal_server,fal_client 值设置正确
- 确保 log_archive_dest_n 参数设置正确

更多可能涉及的初始化参数可以参考 2.1 中的第 4 小章

首先来看当前的 primary 数据库：

```
JSSWEB> show parameter fal

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
fal_client                           string      jssweb
fal_server                           string      jsspdg

JSSWEB> show parameter name_convert

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_file_name_convert                 string      oradata\jsspdg, oradata\jssweb
log_file_name_convert                string      oradata\jsspdg, oradata\jssweb

JSSWEB> show parameter log_archive_dest

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
log_archive_dest                     string
log_archive_dest_1                   string      LOCATION=E:\ora10g\oradata\jss
                                                 web\arc  VALID_FOR=(ALL_LOGFIL
```

```
                                                    ES,ALL_ROLES) DB_UNIQUE_NAME=j
                                                    ssweb
log_archive_dest_2                  string      service=jsspdg
                                                    OPTIONAL LGWR SYNC AFFIRM VALI
                                                    D_FOR=(ONLINE_LOGFILES,PRIMARY
                                                    _ROLE) DB_UNIQUE_NAME=jsspdg
.................
.................
.................
log_archive_dest_state_1            string      ENABLE
log_archive_dest_state_2            string      defer
```

由于此处 primary 的初始化参数并不合适，为了避免其转换之后发生错误，我们需要提前做些修改：

```
JSSWEB> alter system set log_archive_dest_2='location=e:\ora10g\oradata\jssweb\std\
valid_for=(standby_logfiles,standby_role) db_unique_name
=jssweb';

系统已更改。

JSSWEB> alter system set log_archive_dest_1='location=e:\ora10g\oradata\jssweb\arc\
valid_for=(online_logfiles,all_roles) db_unique_name=jss
web';

系统已更改。

JSSWEB> alter system set log_archive_dest_state_2='enable';

系统已更改。

JSSWEB> alter system set fal_server='jssldg';

系统已更改。

--xx_file_name_convert 这两个参数无法动态修改，因此我们首先修改 spfile，然后再重启一下数据库
JSSWEB> alter system set db_file_name_convert='oradata\jssldg','oradata\jssweb' scope=spfile;

系统已更改。

JSSWEB> alter system set log_file_name_convert='oradata\jssldg','oradata\jssweb' scope=spfile;

系统已更改。

JSSWEB> startup force
```

然后再看看待转换的逻辑 **standby**

```
JSSLDG> show parameter fal

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
fal_client                           string
fal_server                           string
JSSLDG> show parameter file_name

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_file_name_convert                 string      oradata\jssweb, oradata\jssldg
log_file_name_convert                string      oradata\jssweb, oradata\jssldg
JSSLDG> show parameter log_archive

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
log_archive_config                   string      DG_CONFIG=(jssweb,jsspdg,jssld
                                                 g)
log_archive_dest                     string
log_archive_dest_1                   string      location=E:\ora10g\oradata\jss
                                                 ldg\arc\ valid_for=(online_log
                                                 files,all_roles) db_unique_nam
                                                 e=jssldg
log_archive_dest_10                  string
log_archive_dest_2                   string      location=E:\ora10g\oradata\JSS
                                                 LDG\std\ valid_for=(standby_lo
                                                 gfiles,standby_role) db_unique
                                                 _name=JSSLDG
.....
.....
```

对于待转换的逻辑 standby 中，某些初始化参数也可以不设置，不过走到这一步了，顺手全设置一遍。

```
JSSLDG> alter system set fal_server='jssweb';

系统已更改。

JSSLDG> alter system set fal_client='jssldg';

系统已更改。

JSSLDG> alter system set log_archive_dest_3='service=jssweb lgwr async
valid_for=(online_logfiles,primary_role) db_unique_name=jssweb';
```

```
系统已更改。
```

2、检查 primary 数据库是否配置了 standby redologs

```
JSSWEB> select * from v$standby_log;

未选定行
```

对于逻辑 standby 数据库，standby redologs 是必须的，因此我们需要为当前的 primary 创建几个 standby redologs。

```
JSSWEB> alter database add standby logfile group 4 ('e:\ora10g\oradata\jssweb\standbyrd01.log') size 20m;

数据库已更改。
.....
.....
.....
JSSWEB> alter database add standby logfile group 8 ('e:\ora10g\oradata\jssweb\standbyrd05.log') size 20m;

数据库已更改。
```

二、检查 primary 数据库状态

在当前的 primary 数据库查询 v$database 视图中的 switchover_status 列，查看当前 primary 数据库状态。

```
JSSWEB> select switchover_status from v$database;

SWITCHOVER_STATUS
--------------------
TO STANDBY
```

如果该查询返回 TO STANDBY 或 SESSIONS ACTIVE 则表示状态正常，可以执行转换操作，如果否的话，就需要你先检查一下当前的 dataguard 配置，看看是否

三、准备转换 primary 为逻辑 standby

执行下列语句，将 primary 置为准备转换的状态：

```
JSSWEB> alter database prepare to switchover to logical standby;

数据库已更改。
```

查看一下 switchover_status 的状态，哟，果然变成准备 ing 啦~~

```
JSSWEB> select switchover_status from v$database;

SWITCHOVER_STATUS
--------------------
PREPARING SWITCHOVER
```

四、准备转换逻辑 standby 为 primary

我们一定要学习 oracle 这种逻辑，甭管想做什么，都得先有个准备的过程~

```
JSSLDG> alter database prepare to switchover to primary;

数据库已更改。

JSSLDG> select switchover_status from v$database;

SWITCHOVER_STATUS
--------------------
PREPARING SWITCHOVER
```

五、再次检查 primary 数据库状态

```
JSSWEB> select switchover_status from v$database;

SWITCHOVER_STATUS
--------------------
TO LOGICAL STANDBY
```

注意：这步虽然不做什么操作，但检查结果却非常重要，它直接关系到 switchover 转换是否能够成功。逻辑 standby 执行完 prepare 命令之后，就会生成相应的 LogMiner 字典数据(就像我们前面创建逻辑 standby 时，primary 会生成 LogMiner 字典数据一样)，只有它正常生成并发送至当前的 primary，转换操作才能够继续下去。不然当前的 primary 数据库在转换完之后，可能就失去了从新的 primary 接收 redo 数据的能力了。

因此，如果上述查询的返回结果不是：TO LOGICAL STANDBY 的话，你可能就需要取消此次转换，检查原因，然后再重新操作了。

提示：
取消转换可以通过下列语句：
ALTER DATABASE PREPARE TO SWITCHOVER CANCEL;
需要分别在 primary 和逻辑 standby 执行。

六、转换 primary 为逻辑 standby

执行下列语句：

```
JSSWEB> alter database commit to switchover to logical standby;

数据库已更改。
```

该语句需要等待当前 primary 所有事务全部结束。同时该语句也会自动拒绝用户发布的新事务或修改需求。为确保该操作尽可能快的执行，最好自开始切换操作起就禁止所有用户的操作。

该命令执行完之后，这个 primary 就已经成为新的逻辑 standby 了。不过在新 primary 执行完转换之前，不要关闭当前这个数据库。

七、再次检查逻辑 standby 状态

逻辑 standby 在接收到前 primary 的转换消息，并应用完相关的 redo 数据之后，会自动暂停 sql 应用，然后

查询 switchover_status 的状态，应该为：TO PRIMARY

```
JSSLDG> select switchover_status from v$database;

SWITCHOVER_STATUS
--------------------
TO PRIMARY
```

八、转换逻辑 standby 为 primary

最后的工作总会在逻辑 standby 上操作，通过上列语句，将该逻辑 standby 转换为新的 primary。

```
JSSLDG> alter database commit to switchover to primary;

数据库已更改。
```

转换完成

九、启动新逻辑 standby 的 sql 应用

最后启动新逻辑 standby 的 sql 应用。

```
JSSWEB> alter database start logical standby apply;

数据库已更改。
```

提示：还记的我们的 jsspdg 吗，虽然它也是 standby(物理)，不过它现在也并非这个 dataguard 配置中的一员了，这也是由于逻辑 standby 自身特性决定的，每一个逻辑 standby 都相当于是一个不同于 primary 的数据库(DBID 都不同)，因此在逻辑 standby 完成了转换之后，相当于原 primary 已经消失，因此原 primary 配置的物理 standby 也失去了主从参照，不过原 primary 配置的逻辑 standby 不会有影响。

逻辑 standby 之 failover

前面学习物理 standby 的 failover 时我们提到过，failover 有可能会丢失数据(视当前的数据库保护模式而定)，对于逻辑 standby 也一样；物理 standby 在做 failover 演示时还提到过，所有的操作都会在 standby 端执行，对于逻辑 standby 这也一样，甚至对于明确提及在前 primary 执行的，你不执行，也没关系，毕竟对于 failover，我们假设的就是，primary 已经 over 了:)

一、准备工作要充分

准备工作可以从以下几个方面着手：

1、检查及处理丢失的归档

虽然本步不是必须的，但如果希望尽可能少丢失数据，除了数据保护模式之外，本步操作也非常重要。如果此时 primary 仍可被访问，首先检查当前的归档日志序号与逻辑 standby 是否相同：

```
JSSLDG> select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
```

```
JSSWEB> select sequence#,applied from dba_logstdby_log;

  SEQUENCE# APPLIED
---------- --------
        23 YES
        24 YES

已选择 2 行。
```

提示：如果 primary 的数据库已经无法打开，您就只好直接到磁盘上查看归档目录中的序号来与 standby 端做比较了。

如果不同序号，则将 primary 尚未发送至 standby 的归档文件手工复制到待转换的逻辑 standby 服务器，然后在 standby 端通过 ALTER DATABASE REGISTER LOGICAL LOGFILE ''; 命令将文件手工注册

如果 standby 与 primary 的归档序号相同，但某些序号的 applied 状态为 no，建议你检查一下当前 standby 是否启动了 SQL 应用：）。

2、检查待转换逻辑 standby 的日志应用情况

可以通过查询 v$logstdby_progress 视图：

```
JSSWEB> select applied_scn,latest_scn from v$logstdby_progress;

APPLIED_SCN LATEST_SCN
----------- ----------
    1259449    1259453
```

如果两值一致，表示所有接收到的归档都已经应用了。

3、检查及修正待转换逻辑 standby 的初始化参数配置

确认待转换的逻辑 standby 配置了正确的归档路径，不仅是写本地的归档，还要有写远程的归档，不然转换完之后，这台新的 primary 就成了光杆司令了。

```
JSSWEB> show parameter log_archive_dest
.....
```

当然一般来说，我们都是推荐在创建 standby 的同时将一些用于角色切换的初始化参数也配置好(primary 和 standby 端都应如此)，以减小切换时操作的时间，提高切换效率。

二、激活新的 primary 数据库

首先查看当前操作的角色

```
JSSWEB> select database_role,force_logging from v$database;

DATABASE_ROLE      FOR
---------------- ---
LOGICAL STANDBY    YES
```

注意，如果当前 force_logging 为 no，务必执行：Alter database force logging;

转换 standby 角色为 primary

```
JSSWEB> alter database activate logical standby database finish apply;

数据库已更改。
```

该语句主要是停止待转换的逻辑 standby 中 RFS 进程，并应用完当前所有已接收但并未应用的 redo 数据，然后停止 sql 应用，将数据库转换成 primary 角色。

```
JSSWEB> select database_role,force_logging from v$database;

DATABASE_ROLE      FOR
---------------- ---
PRIMARY            YES
```

基本上到这一步，我们可以说角色转换的工作已经完成了，但是注意，活还没有干完！

此处与逻辑 standby 的 switchover 同理，切换完之后，原 dg 配置就失效了(不仅原物理 standby 没了，原逻辑 standby 也失去了参照，看看，逻辑 standby 的 failover 确实威力巨大呀，怪不得逻辑 standby 用的人这么人呢，环境脆弱肯定是原因之一啊)，因此我们需要做些设置，重新将原来的 standby 再加入到新的 dg 配置环境中。

三、修复其它 standby

注意哟，逻辑 standby 的修复可并不像物理 standby 那样简单，每个逻辑 standby 都相当于是独立的数据库，如果你不希望重建逻辑 standby 的话呢，oracle 倒是也提供了其它解决方案(虽然不一定好使)：

1、在各个原逻辑 standby 中创建数据库链，连接到新的 primary

注意，数据库链中用于连接新 primary 的用户必须拥有 SELECT_CATALOG_ROLE 角色。

```
JSSLDG2> alter session disable guard;

会话已更改。

JSSLDG2> create database link getjssweb connect to jss identified by jss using 'jssweb';

数据库链接已创建。

JSSLDG2> alter session enable guard;

会话已更改。
```

验证一下数据链是否能够正常访问：

```
JSSLDG2> select sysdate from dual@getjssweb;

SYSDATE
--------------
23-2 月  -08
```

提示：关于 alter session enable|disable guard 语句，用于允许或禁止用户修改逻辑 standby 中的结构。例如：

```
JSSLDG2> conn jss/jss
已连接。
JSSLDG2> select *from b;

        ID
----------
         1
         2
         3
         4
         5
         6
         7
         8

已选择 8 行。


JSSLDG2> alter table b rename to a;
alter table b rename to a
*
第 1 行出现错误:
ORA-16224: Database Guard 已启用


JSSLDG2> alter session disable guard;

会话已更改。

JSSLDG2> alter table b rename to a;

表已更改。
```

2、重新启动 SQL 应用

在各个逻辑 standby 执行下列语句启动 sql 应用(注意更新 dblinkName)：

```
JSSLDG2> alter database start logical standby apply new primary getjssweb;

数据库已更改。
```

如果你运气好，等语句执行完之后，恢复过程就完成了。如果你非常不幸的碰到了 ORA-16109 错误，那么我不得不告诉你，恐怕你得重建逻辑 standby 了。所以，祝你好运吧:)

语句顺利执行完之后，我们来验证一下：

```
JSSWEB> alter system switch logfile;

系统已更改。

JSSWEB> select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
           862

JSSLDG2> select sequence#,applied from dba_logstdby_log;

 SEQUENCE# APPLIED
---------- --------
       862 NO
```

注意：出现问题了!!

日志是传输过去了，但是逻辑 standby 并没有开始应用，怎么回事？

我们先来确认一下 standby 的各进程状态：

```
JSSLDG2> select process,status,group#,thread#,sequence#,block#,blocks from v$managed_standby;

PROCESS   STATUS        GROUP#                                   THREAD#
SEQUENCE#     BLOCK#     BLOCKS
--------- ------------ ---------- ---------- ---------- ---------- ----------
ARCH      CLOSING            2                                    1          4
16385       1836
ARCH      CLOSING            6                                    1        862
1          18
RFS       IDLE             N/A                                    0          0
0          0
RFS       IDLE               3                                    1        863
2          1
```

看起来也是正常的，接收完了 862,正在等待 863，但是，为什么不应用呢。

手工查询一下新 primary 生成的归档日志情况：

```
JSSWEB> select sequence#,name,COMPLETION_TIME from v$archived_log where sequence#>855;

 SEQUENCE#                                                        NAME
COMPLETION_TIME
---------- ------------------------------------------------------------ -------------------
       856                          E:\ORA10G\ORADATA\JSSWEB\ARC\LOG1_856_641301252.ARC
2008-02-21 10:15:42
```

```
         857                              E:\ORA10G\ORADATA\JSSWEB\ARC\LOG1_857_641301252.ARC
2008-02-21 10:16:46
         858                              E:\ORA10G\ORADATA\JSSWEB\ARC\LOG1_858_641301252.ARC
2008-02-23 14:15:18
         859                              E:\ORA10G\ORADATA\JSSWEB\ARC\LOG1_859_641301252.ARC
2008-02-23 14:56:55
         860                              E:\ORA10G\ORADATA\JSSWEB\ARC\LOG1_860_641301252.ARC
2008-02-23 14:57:03
         861                              E:\ORA10G\ORADATA\JSSWEB\ARC\LOG1_861_641301252.ARC
2008-02-23 16:58:14
         861 jssldg2                                                              2008-02-
23 16:58:16
         862                              E:\ORA10G\ORADATA\JSSWEB\ARC\LOG1_862_641301252.ARC
2008-02-23 17:08:57
         862 jssldg2                                                              2008-02-
23 17:08:57
         863                              E:\ORA10G\ORADATA\JSSWEB\ARC\LOG1_863_641301252.ARC
2008-02-23 17:19:48
         863 jssldg2                                                              2008-02-
23 17:20:59
         864                              E:\ORA10G\ORADATA\JSSWEB\ARC\LOG1_864_641301252.ARC
2008-02-23 17:21:11
         864 jssldg2                                                              2008-02-
23 17:21:13

已选择 13 行。
```

发现了一点点痕迹，我们的切换操作是下午 3 点左右进行的，期间还产生了序列号为 860,861 之类的归档文件，但并未传输至 standby，是不是因为这些文件中包含了一部分应被应用的数据，因此造成 standby 接收到的新 primary 传输过来的归档 scn 与最后应用的 scn 不连续，所以无法应用？再来验证一下：

```
JSSLDG2> select applied_scn,latest_scn from v$logstdby_progress;

APPLIED_SCN LATEST_SCN
----------- ----------
    1259449    1284126
```

果然如此，应用的 scn 与最后的 scn 确实不匹配，剩下的就好解决了，把所有可疑的应传输到 standby 的归档文件手工复制到 standby，然后通过 alter 命令注册一下：

```
JSSLDG2> alter database register logical logfile 'E:\ora10g\oradata\jssldg2\std\LOG1_859_641301252.ARC';

数据库已更改。

JSSLDG2> alter database register logical logfile 'E:\ora10g\oradata\jssldg2\std\LOG1_860_641301252.ARC';
```

```
数据库已更改。

JSSLDG2> alter database register logical logfile 'E:\ora10g\oradata\jssldg2\std\LOG1_861_641301252.ARC';

数据库已更改。
```

提示：复制文件的时候尽可能把相近时间段的归档文件都拷过来，不用担心无用归档会不会影响到应用，oracle 会自动判断归档中的 scn，对于已经应用过的正常情况下是不会重复应用的，因此我们把 859,860,861 全部复制过来。

再查看一下应用状态：

```
JSSLDG2> select sequence#,applied from dba_logstdby_log;

 SEQUENCE# APPLIED
---------- --------
       862 CURRENT
       863 CURRENT
```

哈哈，已经开始应用了。逻辑 standby 恢复成功！想起官方文档中有一句提示，说的是在打开新的 primary 数据库，生成数据字典之前，不要执行任何 DDL，不然就只能重建逻辑 standby 了，估计就是担心执行 ddl 后不幸触发日志切换，造成逻辑 standby 接收新 primary 传来的归档文件不连续，无法顺利应用。

切换完成之后，在修复逻辑 standby 的同时，顺手打扫一下战场，比如设置新 primary 数据库的备份策略，以及考虑如何修复前故障的 primary 等等，dba 这份工作，人前看起来光鲜，如果你已经下定决心要从事这行，那对于人后的辛苦一定要有深刻心理准备哟，你看看像上面这种情况，primary 只要随随便便宕个机，引之而来的工作量就够我们忙活的。

也许这个时候 dba 需要的不仅是保持清晰的大脑，还要能打开思路，此时我们更不妨考虑在做角色切换和修复损坏的 primary 之间做个选择，究竟哪个更快，哪个更简便一些呢，

你看，干 dba 这行，不仅压力大，不仅要本领过硬，不仅要耐心细致，关键时刻还要保持清醒的头脑，额地神耶，太刺激啦，太有挑战啦~~~~~~

第三部分 逻辑 standby(4)高级管理　2008.03.04

一、监控逻辑 standby

与物理 standby 的管理一样，oracle 提供了一系列动态性能视图来查看逻辑 standby 的状态，有一些我们前面已经接触过，而有一些，我们还从未用过。。。。。

1、DBA_LOGSTDBY_EVENTS

可以把该视图看成逻辑 standby 操作日志，因此如果发生了错误，可以通过该视图查看近期逻辑 standby 都做了些什么。默认情况下，该视图保留 100 条事件的记录，不过你可以通过 DBMS_LOGSTDBY.APPLY_SET()过程修改该参数。

例如：

```
JSSLDG2> select event_time,status,event from dba_logstdby_events;

EVENT_TIME          STATUS                                                    EVENT
------------------- --------------------------------------------------- ----------------------------------------
2008-03-06 08:58:11 ORA-16112: 日志挖掘和应用正在停止
2008-03-06 09:02:00 ORA-16111: 日志挖掘和应用正在启动
2008-03-06 09:52:53 ORA-16128: 已成功完成用户启动的停止应用操作
2008-03-12 15:52:53 ORA-16111: 日志挖掘和应用正在启动
2008-03-12 16:09:17 ORA-16226: 由于不支持而跳过 DDL                          ALTER DATABASE
OPEN
2008-03-05 17:21:46 ORA-16111: 日志挖掘和应用正在启动
.....
```

2、DBA_LOGSTDBY_LOG

该视图用来记录当前的归档日志应用情况，等同于物理 standby 中的 v$archived_log，多数情况下，你只需要关注 SEQUENCE#,APPLIED，即查看日志序号和是否应用，当然该视图还能提供更多信息，比如应用的 scn，应用时间等，例如：

```
JSSLDG2> select sequence#,first_change#,next_change#,timestamp,applied from dba_logstdby_log;

 SEQUENCE# FIRST_CHANGE# NEXT_CHANGE# TIMESTAMP           APPLIED
---------- ------------- ------------ ------------------- --------
       869       1319212      1319811 2008-03-12 16:09:15 CURRENT
```

通常情况下，该查询只会返回几条记录，如果说你的数据库操作非常频繁，可能记录数会稍多一些，但如果记录数非常多，那你可能就需要关注一下，是不是出了什么问题，难道 sql 应用没有启动？

3、V$LOGSTDBY_STATS

从名字就大致猜的出来，该视图显示的是状态信息，没错，你猜对了，该视图就是用来显示 LogMiner 的统计信息及状态。

```
JSSLDG2> select *from v$logstdby_stats;
```

```
NAME                                                             VALUE
---------------------------------------------------------------- ----------
number of preparers                                                       1
number of appliers                                                        5
maximum SGA for LCR cache                                                30
parallel servers in use                                                   9
maximum events recorded                                                 100
preserve commit order                                                  TRUE
transaction consistency                                                FULL
record skip errors                                                     Y
record skip DDL                                                          Y
record applied DDL                                                       N
.....................
```

4、V$LOGSTDBY_PROCESS

该视图显示当前 log 应用服务的相关信息。常用于诊断归档日志逻辑应用的性能问题(后面优化部分会有涉及)，包含的信息也很广：

- 身份信息：SID,SERIAL#,SPID
- SQL 应用进程：COORDINATOR, READER, BUILDER, PREPARER, ANALYZER, 或 APPLIER
- 进程当前的状态：见 status_code 或 status 列
- 该进程当前操作 redo 记录最大 SCN：high_scn 列

例如：

```
JSSLDG2> select sid,serial#,spid,type,status,high_scn from v$logstdby_process;

       SID    SERIAL#   SPID                 TYPE                                                             STATUS
HIGH_SCN
---------- ---------- ------------ --------------- ------------------------------------------------------------ ----------
       145          1  508                  COORDINATOR          ORA-16116: 无可用工作
1319811
       146          2 2464               READER               ORA-16240: 正在等待日志文件 (线程号
1, 序列号 870)                1319811
       143          1  1512                 BUILDER              ORA-16116: 无可用工作
1319742
       142          1  4000                 PREPARER             ORA-16116: 无可用工作
1319741
       139          1  2980                 ANALYZER             ORA-16116: 无可用工作
1319707
       135          1  1648                 APPLIER              ORA-16116: 无可用工作
1319430
       138          1  2332                 APPLIER              ORA-16116: 无可用工作
1319439
       132          1  2200                 APPLIER              ORA-16116: 无可用工作
1319443
```

```
        134          1 4020          APPLIER          ORA-16116: 无可用工作
.....
```

5、V$LOGSTDBY_PROGRESS

该视图显示 log 应用服务当前进展状况，比如当前应用到逻辑 standby 的 scn 及时间，sql 应用开始应用的 scn 及时间，最后接收及应用的 scn 和时间等等。

例如：

```
JSSLDG2> select * from v$Logstdby_progress;

APPLIED_SCN APPLIED_TIME              RESTART_SCN RESTART_TIME              LATEST_SCN
LATEST_TIME              MINING_SCN MINING_TIME
----------- ------------------- ----------- ------------------- ---------- ------------------- ---------- -------------------
      1319810 2008-03-12 16:06:51       1319662 2008-03-12 16:03:22       1319810 2008-03-12 16:45:33
1319811 2008-03-12 16:06:51
```

6、V$LOGSTDBY_STATE

该视图就最简单了，就是显示 sql 应用的大致状态，比如 primary 库的 dbid 啦，是否启动了实时应用啦，当前 sql 应用的状态啦之类。

注意 state 列，该列可能有下述的几种状态：

- INITIALIZING: LogMiner session 已创建并初始化
- LOADING DICTIONARY: SQL 应用调用 LogMiner 字典
- WAITING ON GAP: SQL 应用正等待日志文件，可能有中断
- APPLYING: SQL 应用正在工作
- WAITING FOR DICTIONARY LOGS: SQL 应用等待 LogMiner 字典信息
- IDLE: SQL 应用工作非常出色，已经干的没什么可干了:)

例如：

```
JSSLDG2> select * from v$Logstdby_state;

PRIMARY_DBID SESSION_ID REALTIME_APPLY            STATE
------------ ---------- -------------------- ------------------------------
  3408827880         42 Y                    APPLYING
```

二、管理逻辑 standby

1、接收到的归档文件

前章曾经提到，逻辑 standby 应用完归档后会自动删除该归档文件，该特性你如果觉着不爽，没关系，执行下面这个过程，屏蔽掉它：

```
JSSLDG2> EXECUTE DBMS_LOGSTDBY.APPLY_SET('LOG_AUTO_DELETE', FALSE);
```

提示：这种操作并非毫无意义，比如说逻辑 standby 打开了 flashback database，那如果你想恢复到之前的某个时间点，然后再接着应用，就必须要有该时间点后对应的归档，假如 LOG_AUTO_DELETE 为 TRUE 的话，显然应用过的归档就不存在了，想回都回不去。

2、启动实时应用

默认情况下，log 应用服务会等待单个归档文件全部接收之后再启动应用(在前面 redo 传输服务中我们介绍了不同形式的传输方式)，如果 standby 端使用了 standby redologs，就可以打开实时应用(real-time apply)，这样 dg 就不需要再等待接收完归档文件，只要 rfs 将 redo 数据写入 standby redologs，即可通过 MRP/LSP 实时写向 standby，这样就可以尽可能保持 standby 与 primary 的同步。

要启动逻辑 standby 的实时应用，只需要在启动逻辑 standby 应用时加上 immediate 子句即可，前面我们已经无数次的演练过，例如：

```
JSSLDG2> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

3、定义 DBA_LOGSTDBY_EVENTS 视图中事件记录的相关参数。

Dba_logstdby_events 视图前面刚讲过，里面记录了逻辑 standby 的一些操作事件，如果你希望修改该视图中记录的事件信息的话，可以通过下列的方式：

例如，希望该视图能够保留最近 999 条事件，可以通过执行下列语句：

```
JSSLDG2> select *from v$logstdby_stats where name='maximum events recorded';

NAME                                                             VALUE
---------------------------------------------------------------- ----------
maximum events recorded                                          100

JSSLDG2> alter database stop logical standby apply;

数据库已更改。

JSSLDG2> execute dbms_logstdby.apply_set('max_events_recorded','999');

PL/SQL 过程已成功完成。

JSSLDG2> alter database start logical standby apply immediate;

数据库已更改。

JSSLDG2> select *from v$logstdby_stats where name='maximum events recorded';

NAME                                                             VALUE
---------------------------------------------------------------- ----------
maximum events recorded                                          999
```

再比如，你如果想在视图中记录 ddl 操作的信息，可以通过执行下列语句：

```
JSSLDG2> execute dbms_logstdby.apply_set('RECORD_APPLIED_DDL','TRUE');
```

4、指定对象跳过应用，请用 DBMS_LOGSTDBY.SKIP

默认情况下，接收自 primary 的 redo 数据中，所有能够被 standby 支持的操作都会在逻辑 standby 端执行，如果你希望跳过对某些对象的某些操作的话，DBMS_LOGSTDBY.SKIP 就能被派上用场了。

先来看看 dbms_logstdby.skip 的语法：

```
DBMS_LOGSTDBY.SKIP (
    stmt                    IN VARCHAR2,
    schema_name             IN VARCHAR2 DEFAULT NULL,
    object_name             IN VARCHAR2 DEFAULT NULL,
    proc_name               IN VARCHAR2 DEFAULT NULL,
    use_like                IN BOOLEAN DEFAULT TRUE,
    esc                     IN CHAR1 DEFAULT NULL);
```

除 stmt 外，其它都是可选参数，并且看字面意义就能明白其所指，下面简单描述一下 stmt 参数调用的关键字都是指定值，详细见下列：

STMT关键字	包含的操作
NON_SCHEMA_DDL	不属于模式对象的所有其它ddl操作
	提示：使用该关键字时，SCHEMA_NAME和OBJECT_NAME两参数也必须指定。
SCHEMA_DDL	创建修改删除模式对象的所有ddl操作 (例如: tables, indexes, and columns)
	提示：使用该关键字时，SCHEMA_NAME和OBJECT_NAME两参数也必须指定。
DML	Includes DML statements on a table (for example: INSERT, UPDATE, and DELETE)
CLUSTER	AUDIT CLUSTER CREATE CLUSTER DROP CLUSTER TRUNCATE CLUSTER
CONTEXT	CREATE CONTEXT DROP CONTEXT
DATABASE LINK	CREATE DATABASE LINK CREATE PUBLIC DATABASE LINK DROP DATABASE LINK DROP PUBLIC DATABASE LINK
DIMENSION	ALTER DIMENSION CREATE DIMENSION DROP DIMENSION
DIRECTORY	CREATE DIRECTORY DROP DIRECTORY
INDEX	ALTER INDEX CREATE INDEX DROP INDEX
PROCEDURE	ALTER FUNCTION ALTER PACKAGE ALTER PACKAGE BODY ALTER PROCEDURE CREATE FUNCTION CREATE LIBRARY CREATE PACKAGE CREATE PACKAGE BODY CREATE PROCEDURE

	DROP FUNCTION DROP LIBRARY DROP PACKAGE DROP PACKAGE BODY DROP PROCEDURE
PROFILE	ALTER PROFILE CREATE PROFILE DROP PROFILE
ROLE	ALTER ROLE CREATE ROLE DROP ROLE SET ROLE
ROLLBACK STATEMENT	ALTER ROLLBACK SEGMENT CREATE ROLLBACK SEGMENT DROP ROLLBACK SEGMENT
SEQUENCE	ALTER SEQUENCE CREATE SEQUENCE DROP SEQUENCE
SYNONYM	CREATE PUBLIC SYNONYM CREATE SYNONYM DROP PUBLIC SYNONYM DROP SYNONYM
TABLE	ALTER TABLE CREATE TABLE DROP TABLE
TABLESPACE	CREATE TABLESPACE DROP TABLESPACE TRUNCATE TABLESPACE
TRIGGER	ALTER TRIGGER CREATE TRIGGER DISABLE ALL TRIGGERS DISABLE TRIGGER DROP TRIGGER ENABLE ALL TRIGGERS ENABLE TRIGGER
TYPE	ALTER TYPE ALTER TYPE BODY CREATE TYPE CREATE TYPE BODY DROP TYPE DROP TYPE BODY
USER	ALTER USER CREATE USER DROP USER

VIEW	CREATE VIEW DROP VIEW

例如，你想跳过 jss 用户下对 tmp1 表的 dml 操作，可以通过执行下列语句实现(执行该过程前需要先停止 redo 应用)：

```
JSSLDG2> alter database stop logical standby apply;

数据库已更改。

JSSLDG2> execute dbms_logstdby.skip('DML','JSS','TMP1');

PL/SQL 过程已成功完成。

JSSLDG2> alter database start logical standby apply;

数据库已更改。
```

提示：DBMS_LOGSTDBY.SKIP 的功能非常强大，限于篇幅，这里仅举示例，而且由于其操作非常灵活，此篇俺也不可能就其用法做个一一列举，因此，更丰富的操作方式就留待看官们下头自行发现去吧:)

三、修改逻辑 standby 端数据

我们前面提到，逻辑 standby 一个极具实用价值的特性即是可以边查询边应用，因此将其做为报表服务器专供查询是个很不错的想法，而且逻辑 standby 相对于物理 standby 而言更具灵活性，比如我们可以在逻辑 standby 上，对一些表创建 primary 库上并不方便创建的索引，约束，甚至可以做 dml,ddl 操作(当然，需要注意不要破坏了与 primary 之间同步的逻辑关系)。不过由于此时 dg 仍然控制着对逻辑 standby 表的读写操作，因此，如果你想对逻辑 standby 中的数据做些什么的话，alter session database disable|enable guard 语句就必须牢记在心了，它拥有像“芝麻开门”一样神奇的能力，不信？下面我们就来感受一下吧。

1、逻辑 standby 端执行 ddl

在逻辑 standby 端开始了 redo 应用的情况下，执行 ddl 操作：

```
JSSLDG2> create table tmp55 as select * From b;
create table tmp55 as select * From b
                                     *
第 1 行出现错误:
ORA-01031: 权限不足
```

看看，出错了吧~~~

```
JSSLDG2> alter session disable guard;

会话已更改。

JSSLDG2> create table tmp55 as select * From b;

表已创建。
```

只有关闭了 guard 保护之后，才能操作数据，然后别忘了再启用 guard，以避免不经意的操作对逻辑 standby 的配置造成影响。

```
JSSLDG2> alter session enable guard;

会话已更改。
```

提示：oracle 建议还是尽可能不要在逻辑 standby 执行执行 dml 之类操作，以免破解其与 primary 之间同步的逻辑关系，当然，这只是个建议，如果你已经仔细看完了 3.1 章，并且对数据库表结构及存储结构了如指掌，那您就爱干嘛爱嘛。

2、取消对象同步

如果说，某些表或者数据不需要 dataguard 保护(比如一些在逻辑 standby 端生成的统计表)，这个时候就需要 DBMS_LOGSTDBY.SKIP，前头已经介绍过了 dbms_logstdby.skip 的基本用法，下面我们来具体演示一下！

下面我们假设 standby 端有一批表名为 tmp 开头的表，这张表不再需要保持与 primary 的同步，那么按照步骤执行下列语句，sql 应用即可跳过这些表：

老规矩，先停了 redo 应用

```
JSSLDG2> alter database stop logical standby apply;

数据库已更改。

JSSLDG2> execute dbms_logstdby.skip('SCHEMA_DDL','JSS','TMP%');
--跳过对象的 ddl 操作
PL/SQL 过程已成功完成。

JSSLDG2> execute dbms_logstdby.skip('DML','JSS','TMP%');
--跳过对象的 dml 操作
PL/SQL 过程已成功完成。

JSSLDG2> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;

数据库已更改。
```

注意其中的%，该符号为通配符，作用与在 sql 语句中的相同。

OK,下面来测试一下，先看看逻辑 standby 中表的信息，我们选择两张表，一张是我们前面已经指定了跳过的表 tmp1，另一张是普通表 b：

```
JSSLDG2> select max(aa) from jss.tmp1;

Max(aa)
--------------------
h

JSSLDG2> select max(id) from jss.b;
```

```
            Max(id)
----------
             9

JSSLDG2> select sequence#,applied from dba_logstdby_log;

  SEQUENCE# APPLIED
---------- --------
         872 YES
```

然后在 primary 数据库执行插入操作

```
JSSWEB> select max(aa) from jss.tmp1;

Max(aa)
--------------------
h

JSSWEB> insert into jss.tmp1 values ('i');

已创建 1 行。

JSSWEB> insert into jss.b values (10);

已创建 1 行。

JSSWEB> commit;

提交完成。

JSSWEB> alter system switch logfile;

系统已更改。

JSSWEB> select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
               873
```

再来看看逻辑 standby 端的同步情况：

```
JSSLDG2> select sequence#,applied from dba_logstdby_log;

  SEQUENCE# APPLIED
---------- --------
```

```
           873 YES

显然日志已经接收，再看看数据：
JSSLDG2> select max(id) from b;

        Max(id)
----------
           10

JSSLDG2> select max(aa) from jss.tmp1;

Max(aa)
--------------------
h
```

b 表已应用，而 tmp1 表则无变化。

3、恢复对象同步

如果说某些表某个时候取消了同步，现在希望再恢复同步，没问题，DBMS_LOGSTDBY 家大业大，它还有个叫 UNSKIP 的门生就是专干这个的。

我们来看一下 dbms_logstdby.unskip 的语法：

```
DBMS_LOGSTDBY.UNSKIP (
      stmt                              IN VARCHAR2,
      schema_name                       IN VARCHAR2,
      object_name                       IN VARCHAR2);
```

三项均为必选参数，各参数的定义与 skip 过程相同，这里不再复述。

此处我们来演示恢复 tmp%表的同步。

```
JSSLDG2> select *from dba_logstdby_skip;

ERROR STATEMENT_OPT     OWNER          NAME                  U E PROC
----- --------------- ---------- --------------- - - --------------------
N       SCHEMA_DDL        JSS            TMP%                  Y
N       DML               JSS            TMP%                  Y
N       DML               JSS            TMP1                  Y
........

JSSLDG2> alter database stop logical standby apply;

数据库已更改。

JSSLDG2> execute dbms_logstdby.unskip('DML','JSS','TMP1');
--本步操作是为解决历史遗留问题，不用关注
PL/SQL 过程已成功完成。
```

```
JSSLDG2> execute dbms_logstdby.unskip('DML','JSS','TMP%');

PL/SQL 过程已成功完成。

JSSLDG2> execute dbms_logstdby.unskip('SCHEMA_DDL','JSS','TMP%');

PL/SQL 过程已成功完成。
```

跳过同步已经取消了，紧接着我们需要再调用 dbms_logstdby.instantiate_table 过程重新同步一下跳地的对象，将 skip 这段时间，primary 对 tmp1 表所做的操作同步过来(就俺看来，instantiate_table 过程实际上是借助 dblink 重建了一遍对象)，以保持与 primary 的一致。Dbms_logstdby.instantiate_table 的语法如下：

```
DBMS_LOGSTDBY.INSTANTIATE_TABLE(
    schema_name     IN VARCHAR2,
    table_name      IN VARCHAR2,
    dblink          IN VARCHAR2);
```

使用 DBMS_LOGSTDBY.INSTANTIATE_TABLE 过程重新执行一下同步(执行前别忘了暂停 redo 应用)：

```
JSSLDG2> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE('JSS','TMP1','GETJSSWEB');

PL/SQL 过程已成功完成。

JSSLDG2> select *from jss.tmp1;

AA
--------------------
a
b
c
d
e
f
g
h
i

已选择 9 行。
```

数据已重建，下面测试一下该表的 redo 应用是否恢复了。

```
JSSWEB> insert into jss.tmp1 values ('j');

已创建 1 行。
```

```
JSSWEB> insert into jss.tmp1 values ('k');

已创建 1 行。

JSSWEB> commit;

提交完成。

JSSWEB> alter system switch logfile;

系统已更改。

JSSWEB> select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
           877
```

启动逻辑 standby 端的 redo 应用，看看对象的应用情况：

```
JSSLDG2> alter database start logical standby apply immediate;

数据库已更改。

JSSLDG2> select sequence#,applied from dba_logstdby_log;

  SEQUENCE# APPLIED
---------- --------
       875 YES
       876 YES
       877 YES

JSSLDG2> select *from jss.tmp1;

AA
--------------------
a
b
c
d
e
f
g
h
```

```
i
j
k

已选择 11 行。
```

OK,恢复正常啦！

注意哟，此处我们清楚明白的知道我们之前只操作了 tmp1 一张表，如果是正式应用的话，那你恐怕有必要将所有 tmp 开头的表都同步一下，不然有可能会造成数据丢失的哟。

四、特殊事件的控制

时间紧任务急，呵呵，这里三思就只描述流程，过程就不做演示了，相信你的智力，你一定能看懂。

1、导入传输表空间

- 第一步：屏蔽 guard 保护，逻辑 standby 端操作

```
SQL> ALTER SESSION DISABLE GUARD;
```

- 第二步：导入传输表空间，逻辑 standby 端操作

具体操作步骤可参考三思之前的笔记：使用可传输表空间的特性复制数据！

- 第三步：恢复 guard 保护(或者直接退出本 session 也成)，逻辑 standby 端操作

```
SQL> ALTER SESSION ENABLE GUARD;
```

- 第四步：导入传输表空间，primary 端操作

同第二步。

2、使用物化视图

SQL 应用不支持下列对物化视图的 ddl 操作：

- create/alter/drop materialized view
- create/alter/drop materialized view log

因此，对于现有逻辑 standby，primary 端对物化视图的操作不会传播到 standby 端。不过，对于 primary 创建物化视图之后创建逻辑 standby，则物理视图也会存在于逻辑 standby 端。

- 对于同时存在于 primary 和逻辑 standby 的 ON-COMMIT 物化视图，逻辑 standby 会在事务提交时自动刷新，而对于 ON-DEMAND 的物化视图不会自动刷新，需要手动调用 dbms_mview.refresh 过程刷新。
- 对于逻辑 standby 端建立的 ON-COMMIT 物化视图会自动维护，ON-DEMAND 物化视图也还是需要手工调用 dbms_mview.refresh 过程刷新。

3、触发器及约束的运作方式

默认情况下，约束和触发器同样会在逻辑 standby 端正常工作。

对于有 sql 应用维护的约束和触发器：

- 约束：由于约束在 primary 已经检查过，因此 standby 端不需要再次检查

- 触发器：primary 端操作时结果被记录，在 standby 端直接被应用。

没有 sql 应用维护的约束和触发器：

- 约束有效
- 触发器有效

五、优化逻辑 standby

1、创建 Primary Key RELY 约束

某些情况下能够有效提高 sql 应用效率，具体可参见第三部分第一章。

2、生成统计信息

这个很容易理解嘛，因为 cbo 要用，生成方式即可用 analyze，也可以用 dbms_stats 包。看你个人喜好了。

3、调整进程数

A).调整 APPLIER 进程数

首先查看当前空闲的 applier 进程数：

```
JSSLDG2> SELECT COUNT(*) AS IDLE_APPLIER FROM V$LOGSTDBY_PROCESS
  2    WHERE TYPE = 'APPLIER' and status_code = 16166;

IDLE_APPLIER
------------
           0
```

提示：

status_code = 16166 表示进程是空闲状态，可以看到"STATS"为"ORA-16116: no work available"，当然空闲的 applier 进程数为 0 不一定代表应用应用非常繁忙，也有可能是因为当前没什么需要应用的日志，因此甚至应用进程都没启动:)

检查事务的应用情况：

```
JSSLDG2> select name,value from v$logstdby_stats where name like 'TRANSACTION%';

NAME                                VALUE
---------------------          -------
transactions ready                  896
transactions applied                871
```

如果 ready-applied 的值比 applier 进程数的两倍还要大，则说明你有必要考虑增加 applier 进程的数目了，反之如果 applied 与 ready 的值差不多大，或者其差比 applier 进程数还小，则说明 applier 进程数偏多，你有必要考虑适当减小进程的数目。

如果确认当前 applier 进程都非常繁忙，要增加 applier 进程，可按如下步骤操作：

停止 sql 应用

```
ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

调整 applier 进程数为 20，默认是 5 个

```
EXECUTE DBMS_LOGSTDBY.APPLY_SET('APPLY_SERVERS', 20);
```

重启 sql 应用

```
ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

B).调整 PREPARER 进程数

需要调整 preparer 进程数的机会不多，通常只有一种情况：applier 进程有空闲，transactions ready 还很多，但没有空闲的 preparer 进程，这时候你可能需要增加一些 preparer 进程。

要检查系统是否存在这种情况，可以通过下列的 sql 语句：

首先检查空闲 preparer 进程数量：

```
SELECT COUNT(*) AS IDLE_PREPARER FROM V$LOGSTDBY_PROCESS WHERE TYPE = 'PREPARER' and status_code = 16166;
```

检查事务的应用情况：

```
select name,value from v$logstdby_stats where name like 'TRANSACTION%';
```

查看当前空闲的 applier 进程数：

```
SELECT COUNT(*) AS IDLE_APPLIER FROM V$LOGSTDBY_PROCESS WHERE TYPE = 'APPLIER' and status_code = 16166;
```

如果确实需要调整 preparer 进程数量，可以按照下列步骤，例如：

停止 sql 应用

```
ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

调整 preparer 进程数量为 4(默认只有 1 个 preparer 进程)

```
EXECUTE DBMS_LOGSTDBY.APPLY_SET('PREPARE_SERVERS', 4);
```

重启 sql 应用

```
ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

4、调整 LCR 使用的内存

执行下列语句，查询当前 LCR 可用的最大内存：

```
JSSLDG2> select * from v$logstdby_stats where name='maximum SGA for LCR cache';

NAME                                     VALUE
---------------------------------- --------------------
maximum SGA for LCR cache                30
```

要增加 LCR 可用的内存，按照下列步骤操作：

停止 sql 应用：

```
JSSLDG2> alter database stop logical standby apply;

数据库已更改。
```

调整内存大小，注意默认单位是 M：

```
JSSLDG2> execute dbms_logstdby.apply_set('MAX_SGA',100);

PL/SQL 过程已成功完成。
```

重启 sql 应用

```
JSSLDG2> alter database start logical standby apply immediate;

数据库已更改。
```

5、调整事务应用方式

默认情况下逻辑 standby 端事务应用顺序与 primary 端提交顺序相同。

如果你希望逻辑 standby 端的事务应用不要按照顺序的话，可以按照下列的步骤操作：

```
①  停止 sql 应用:
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;

②  允许事务不按照 primary 的提交顺序应用
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('PRESERVE_COMMIT_ORDER', 'FALSE');

③  重新启动 sql 应用
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

恢复逻辑 standby 按照事务提交顺序应用的话，按照下列步骤：

```
①  还是先停止 sql 应用：
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;

②  重置参数 PRESERVE_COMMIT_ORDER 的初始值：
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_UNSET('PRESERVE_COMMIT_ORDER');

③  重新启动 sql 应用：
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

第四部分 Standby 之 REDO 传输服务　2007.12.18

很多人看电影或者电视剧时往往都以为，影响剧情发展的关键是主角的命运，那么，我不得不又说，你只看到了问题的表面，真正影响剧情发展的.....是导演。对于 data guard 的数据应用而言，幕后的导演是 LOG_ARCHIVE_DEST_n。本章节我们要学习的内容会很多，这一次，希望你能理清要学习的重点。

关于 redo 传输服务(Redo Transport Services)，它不仅控制着传输 redo 数据到其它数据库，同时还管理着解决由于网络中断造成的归档文件未接收的过程。

一、如何发送数据

在 primary 数据库，DataGuard 可以使用归档进程(ARCn)或者日志写进程(LGWR)收集 redo 数据并传输到 standby，不过不管你选择归档进程也好，日志写进程也好，都由一个核心参数来控制，它就是：LOG_ARCHIVE_DEST_n，所以，我们先来：

1、认识 LOG_ARCHIVE_DEST_n 参数

LOG_ARCHIVE_DEST_n(n 从 1 到 10)定义 redo 文件路径。该参数定义必须通过 location 或 service 指明归档文件路径。location 表示本地路径，service 通常是 net service name，即接收 redo 数据的 standby 数据库。

提示：

对于每一个 LOG_ARCHIVE_DEST_n 参数，还有一个对应的 LOG_ARCHIVE_DEST_STATE_n 参数。LOG_ARCHIVE_DEST_STATE_n 参数用来指定对应的 LOG_ARCHIVE_DEST_n 参数是否生效，拥有 4 个属性值：

- ENABLE：默认值，表示允许传输服务。
- DEFER：该属性指定对应的 log_archive_dest_n 参数有效，但暂不使用。
- ALTERNATE：禁止传输，但是如果其它相关的目的地的连接通通失败，则它将变成 enable。
- RESET：功能与 DEFER 属性类似，不过如果传输目的地之前有过错误，它会清除其所有错误信息。

例如：指定本地归档路径

```
LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago/'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

又比如，指定 redo 传输服务

```
LOG_ARCHIVE_DEST_2='SERVICE=jsspdg'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

当然，LOG_ARCHIVE_DEST_n 参数的属性远不止这些。

这些参数都可以通过 alter system set 语句直接联机修改，修改会在 primary 下次日志切换时生效，当然你直接 shutdown 再重启数据库的话也会即时生效:)比如：

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='SERVICE=jsspdg
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)';
```

除 show parameter log_archive_dest 外，还可以通过查询 V$ARCHIVE_DEST 视图的方式查看参数配置，并且 V$ARCHIVE_DEST 视图还可以看到更详细的同步信息。

2、使用 ARCn 归档 redo 数据

默认情况下，redo 传输服务使用 ARCn 进程归档 redo 日志。不过 ARCn 归档进程只支持最高性能的保护模式。如果 standby 数据库处于其它类型的保护模式，那你必须使用 LGWR 传输 redo 数据(为什么会这样呢，三思再来白话几句，我们知道对于最大保护和最高可用性两种模式而言，其实强调的都是一点，redo 数据必须实时应用于 standby 数据库，我们再看来归档，归档是做什么呢？是备份已完成切换的 redolog，完成切换的 redolog 代表着什么呢？说明该 redo 中所有数据均已提交至数据文件，那好我们再回过头来看，数据已完成提交的 redo 并且完成了切换还被复制了一份做为归档，这个时候才准备开始传输到 standby 数据库，这与最大保护和最高可用所要求的实时应用差的简直不是一点半点，现在，你是不是明白了为什么 ARCn 归档进程只能支持最高性能的保护模式)。

1).初始化参数控制 ARCn 归档行为

影响 ARCn 进程的初始化参数有：

LOG_ARCHIVE_DEST_n 及 LOG_ARCHIVE_MAX_PROCESSES。

关于 LOG_ARCHIVE_DEST_n 参数的配置前面介绍了一些，我们知道该参数的部分属性与 ARCn 进程相关，而 LOG_ARCHIVE_MAX_PROCESSES 初始化参数则可看做是专为 ARCn 进程量身打造，该参数用来指定最大可被调用的 ARCn 进程的数量，注意我们指定的是最大值，也就是说数据库在运行过程中是会根据归档的任务繁重程度自动调节归档进程数量的。当然如果说你觉着你的系统归档任务比较繁重，可以通过设置较多的归档进程数量提高归档并发度，但是这个数字也不是越高越好，过高的归档进程数量有可能反而影响系统性能(所谓物极必反就是这个意思，所以这中间是需要你来把握平衡的，当然这方面更多涉及到调优了，非本章所要讲解之重点，就不多说了)。调整该参数可以通过下列语句：

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES = n;
```

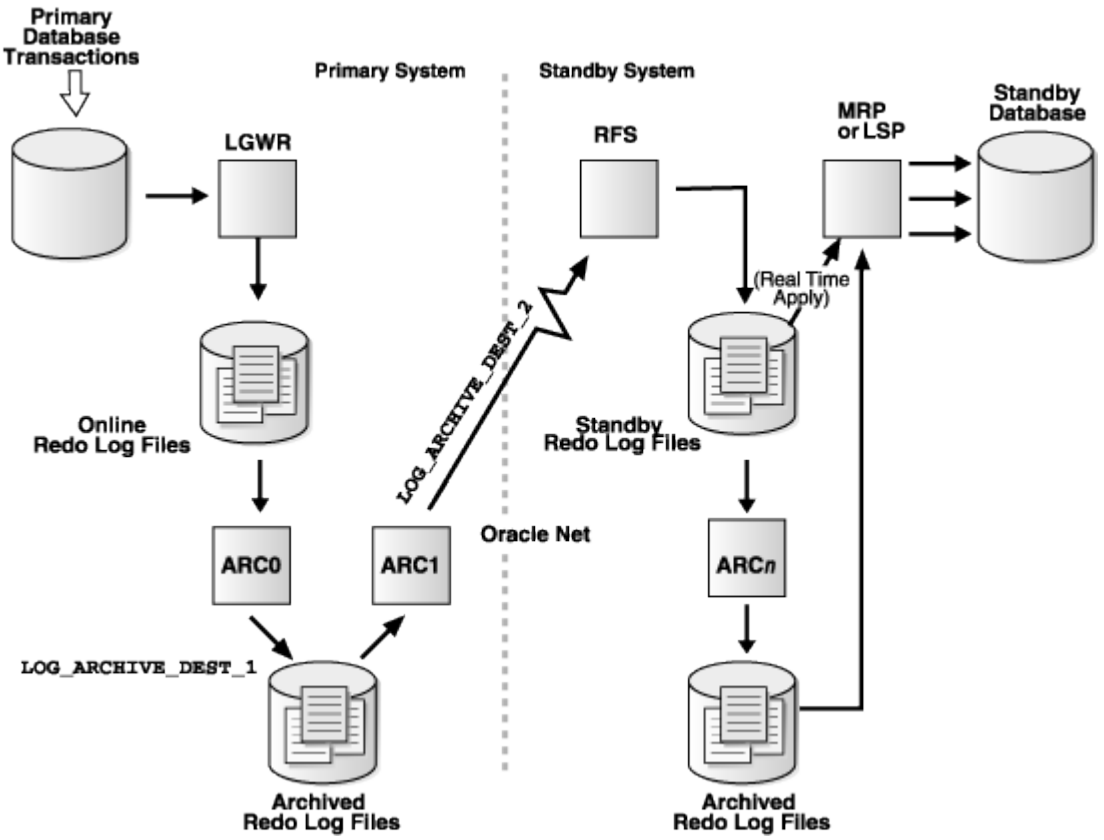
注：n>0 and n<=30

2).ARCn 的归档过程

primary 数据库日志发生切换时就会启动归档：

- 在 primary 数据库(假设有 2 个归档进程)，一旦 ARC0 进程完成 redolog 的归档，ARC1 进程即开始传输该归档到 standby 数据库的指定路径。
- 在 standby 数据库，RFS 进程轮流将 redo 数据写入 standby redo log，再由 standby 数据库中的 ARCn 进程将其写入归档，然后通过 REDO 应用或 SQL 应用将数据应用到 standby 数据库。

如图：[024_1.gif]



另外，因为本地的归档进程与远程归档进程间并无联系，注意如果本地存在删除完成备份的归档的策略，需要在删除之前首先确认这些归档已经被传输到 standby 数据库。

3、使用 LGWR 归档 redo 数据

使用 LGWR 进程与使用 ARCn 进程有明显不同，LGWR 无须等待日志切换及完成归档。

Standby 数据库的 LGWR 进程会先选择一个 standby redo log 文件映射 primary 数据库当前 redolog 的 sequence(以及文件大小)，一旦 primary 数据库有 redo 数据产生，视 LOG_ARCHIVE_DEST_n 初始化参数中 sync 或 async 属性设置，以同步或非同步方式传输到 standby 数据库。

要继续下面的内容，我们必须先了解与 LGWR 归档进程密切相关的几个 LOG_ARCHIVE_DEST_n 参数的属性，如果选择 LGWR 归档 redo 数据，那么在 LOG_ARCHIVE_DEST_n 中必须指定 SERVICE 和 LGWR 属性以允许日志传输服务使用 LGWR 进程来传送 redo 数据到远程归档目的地。我们还需要指定 SYNC(同步)还是 ASYNC(异步)的传输方式，如果指定 SYNC 属性(如果不明确指定的话，默认是 SYNC)，则 primary 数据库任何会产生 redo 操作都会同步触发网络 I/O，并且等到网络 I/O 全部完成才会继续下面的提交，而如果指定了 ASYNC 属性，则会 primary 数据库的操作会先记录 online redologs，然后再传输到 standby。下面详细看看其流程：

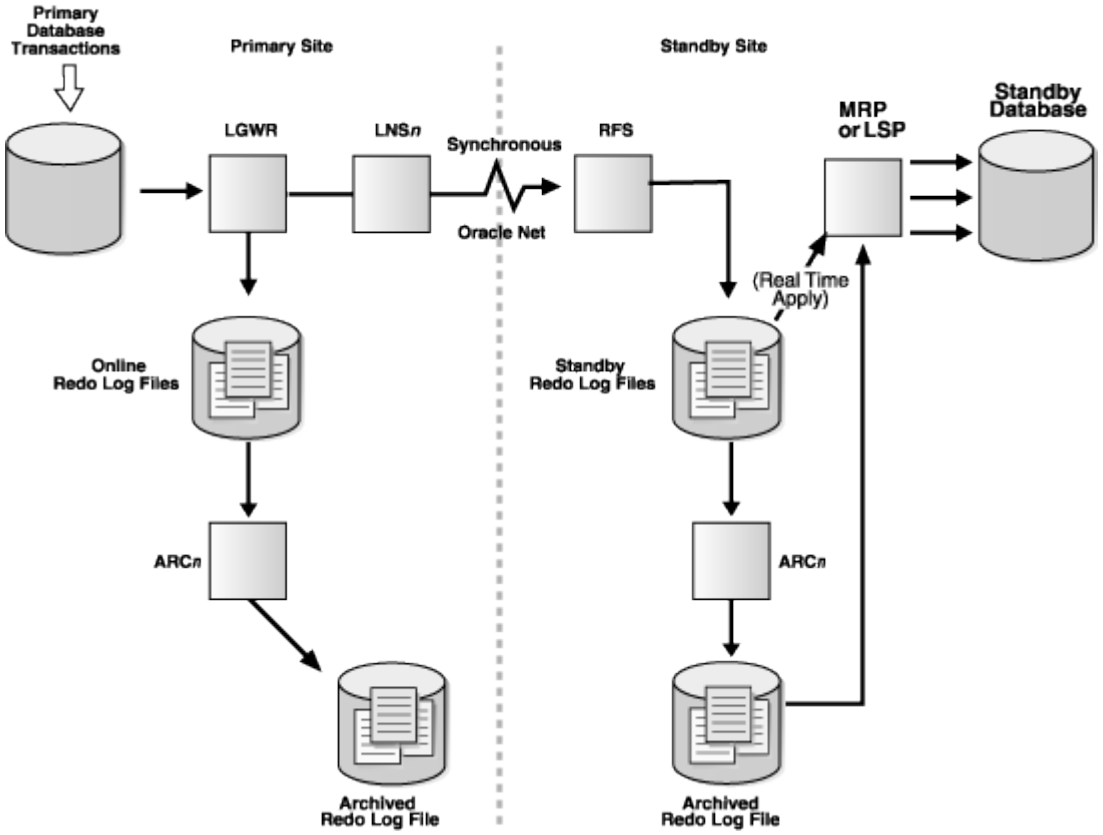
1).LGWR 同步归档的流程

例如初始化参数中有如下设置：

```
LOG_ARCHIVE_DEST_1='LOCATION=E:\ora10g\oradata\jssweb\'
LOG_ARCHIVE_DEST_2='SERVICE=jsspdg LGWR SYNC NET_TIMEOUT=30'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

如果设置 LOG_ARCHIVE_DEST_n 初始化参数 SYNC 属性，建议同时设置 NET_TIMEOUT 属性，该属性控制网络连接的超时时间，如果超时仍无响应，则会返回错误信息。

图：[024_2.gif]



展示了 primary 数据库 LGWR 写 online redologs 的同时，同步传输 redo 数据到 standby 数据库的过程。

我们仍然分两部分来解读：

- 在 primary 数据库，LGWR 提交 redo 数据到 LNSn(LGWR Network Server process)进程(n>0)，LNSn 启动网络传输。
- standby 数据库的 RFS(Remote File Server)将接收到的 redo 数据写入 standby redolog。特别注意，在此期间，primary 数据库的事务会一直保持，直到所有所有含 LGWR SYNC 属性的 LOG_ARCHIVE_DEST_n 指定路径均已完成接收。

一旦 primary 数据库执行日志切换，就会级联触发 standby 的 ARCn 将 standby redo 写入归档，然后通过 redo 应用(MRP 进程)或 sql 应用(LSP 进程)读取归档文件将数据应用至 standby 数据库。(如果启用了实时应用的话，MRP/LSP 会直接读取 standby redolog 并应用到 standby 数据库，无须再等待归档)。

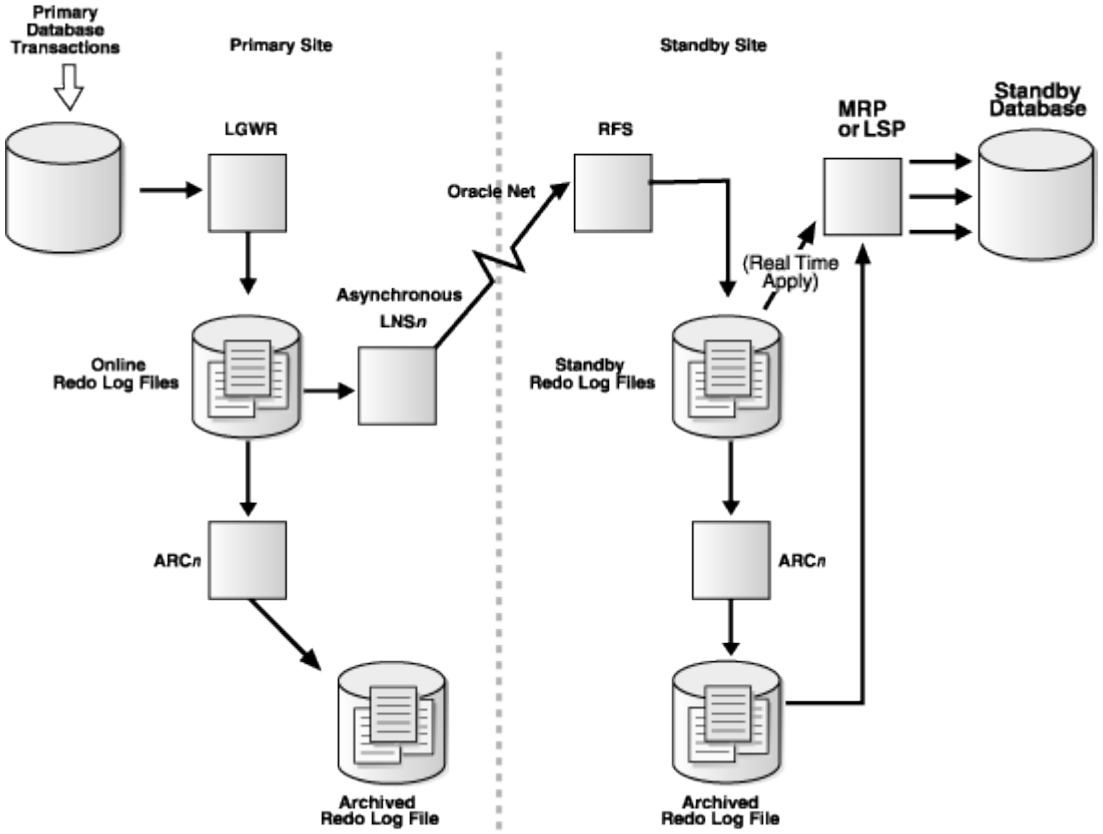
2).LGWR 不同步归档的流程

例如初始化参数中有如下设置：

```
LOG_ARCHIVE_DEST_1='LOCATION=E:\ora10g\oradata\jssweb\'
LOG_ARCHIVE_DEST_2='SERVICE=jsspdg LGWR ASYNC'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

ASYNC 方式归档就不需要再指定 NET_TIMEOUT 了，因为 LGWR 与 LNSn 之间已无关联，所以指定不指定 NET_TIMEOUT 就都没任何影响了，因此对于异步传输而言，即使网络出现故障造成 primary 与 standby 之间通信中断，也并不会影响到 primary 数据库的提交。

图：[024_3.gif]



展示了 LNSn 进程异步传输 redo 数据到 standby 数据库 RFS 进程的过程。

大致步骤与同步传输相同，差别只在 LNSn 进程这里，LGWR 写数据到 online redolog，LNSn 进程访问 online redolog 并传输数据到远程 standby 的 RFS 而不再与本地 LGWR 之间有联系。standby 数据库方面的处理逻辑仍然不变。

二、什么时候发送

这小节包含两个内容，发送什么，以及发送给谁：

1、通过 VALID_FOR 属性指定传输及接收对象

valid_for，字面理解就是基于 xx 有效，再配合其 redo_log_type,database_role 属性，我们基本上可以将其理解为：为指定角色设置日志文件的归档路径，主要目的是为了辅助一旦发生角色切换操作后数据库的正常运转。

redo_log_type 可设置为：ONLINE_LOGFILE，STANDBY_LOGFILE，ALL_LOGFILES

database_role 可设置为：PRIMARY_ROLE，STANDBY_ROLE，ALL_ROLES

注意 valid_for 参数是有默认值的，如果不设置的话，其默认值等同于：

valid_for=(ALL_LOGFILES,ALL_ROLES)

推荐主动设置该参数而不要使用默认值，某些情况下默认的参数值不一定合适，比如逻辑 standby 就不像物理 standby，逻辑 standby 处于 open 模式，不仅有 redo 数据而且还包含多种日志文件(online redologs,archived redologs 以及 standby redologs)。多数情况下，逻辑 standby 生成的 online redologs 与 standby redologs 生成在相同的目录内。因此，推荐你对每个*dest 设置合适的 valid_for 属性。

2、通过 DB_UNIQUE_NAME 属性指定数据库

DB_UNIQUE_NAME 属性主要是为某个数据库指定唯一的数据库名称，这就使得动态添加 standby 到包含 RAC 结构的 primary 数据库的 dg 配置成为可能，并且对于 log_archive_dest_n 中的 service 属性，其属

性值对应的也必然是 db_unique_name，也正因有了 db_unique_name，redo 数据在传输过程中才能确认传输到你希望被传输到的数据库上。当然要保障传输 redo 数据到指定服务器，除了 db_unique_name,log_archive_dest_n 之外，还有一个初始化参数：log_archive_config。

关于 log_archive_config 呢，我们前面有过一些接触，在第二部分第 1 节中也有一些简单的介绍，log_archive_config 初始化参数还包括几个属性，可以用过控制数据库的传输和接收，SEND,NOSEND,RECEIVE,NORECEIVE：

- SEND 允许数据库发送数据到远端
- RECEIVE 则允许 standby 接收来自其它数据库的数据
- NOSEND,NORECEIVE 自然就是禁止喽。

例如，设置 primary 数据库不接收任何归档数据，可以做如下的设置：

```
LOG_ARCHIVE_CONFIG='NORECEIVE,DG_CONFIG=(jssweb,jsspdg)'
```

当然，你同样也需要注意，一旦做了如上的设置，那么假设该服务器发生了角色切换，那它仍然也没有接收 redo 数据的能力哟。

三、出错了咋整

对于归档失败的处理，LOG_ARCHIVE_DEST_n 参数有几个属性可以用来控制一旦向归档过程中出现故障时应该采取什么措施，它们是：

1、REOPEN 指定时间后再次尝试归档

使用 REOPEN=seconds(默认=300)属性，在指定时间重复尝试向归档目的地进行归档操作，如果该参数值设置为 0，则一旦失败就不会再尝试重新连接并发送，直到下次 redo 数据再被归档时会重新尝试，不过并不会归档到已经失败的归档目的地，而是向替补的归档目的地发送。

2、ALTERNATE 指定替补的归档目的地

alternate 属性定义一个替补的归档目的地，所谓替补就是一旦主归档目的地因各种原因无法使用，则临时向 alternate 属性中指定的路径写。

需要注意一点，reopen 的属性会比 alternate 属性优先级要高，如果你指定 reopen 属性的值>0，则 lgwr/arch 会首先尝试向主归档目的地写入，直到达到最大重试次数，如果仍然写入失败，才会向 alternate 属性指定的路径写。

3、MAX_FAILURE 控制失败尝试次数

max_failure 属性指定一个最大失败尝试次数，大家应该能够联想到 reopen，没错两者通常是应该配合使用，reopen 指定失败后重新尝试的时间周期，max_failure 则控制失败尝试的次数，如例：

```
LOG_ARCHIVE_DEST_1='LOCATION=E:\ora10g\oradata\jsspdg\ REOPEN=60 MAX_FAILURE=3'
```

四、管理归档中断

如果 primary 数据库中的归档日志没能成功发送至 standby 数据库，就会出现归档中断。当然通常情况下你不需要担心这一点，因为 dg 会自动检测并尝试复制丢失的归档以解决中断问题，通过什么解决呢？FAL(Fetch Archive Log)。

FAL 分 server 和 client，前面创建步骤中讲初始化参数时想必大家也注意到了。FAL client 自动主动要求传

输归档文件，FAL server 则响应 FAL client 发送的请求。多好的两口子啊。

FAL 机制会自动处理下列类似的归档中断问题：

- 当创建物理或逻辑的 standby 数据库，FAL 机制会自动获取 primary 数据库热备时产生的归档文件。
- 当接收的归档文件出现下列的问题时，FAL 机会会自动获取归档文件解决：
  - 当接收到的归档在应用之后被删除时；
  - 当接收到的归档所在磁盘损坏时；
- 当存在多个物理 standby 时，FAL 机制会自动尝试从其它 standby 服务器获取丢失的归档文件。

让大家了解这个东西不是为了让你做什么，而是希望你放心，oracle 会管理好这些，如果它没有管理好，你明白这个原理，你也能分析一下它为什么没能管理好，通过什么方式能够促使它恢复有效管理的能力，当然你一定要自己动手，也可以，下面通过示例来说明手工处理归档中断（注意，俺只描述步骤，演示俺就不做了。因为三思现在用地最大保护模式，不会丢失归档地说，哇哈哈哈:)）

1、首先你要确认一下 standby 是否存在归档中断

可以通过查询 v$archive_gap 视图来确定，如果有记录就表示有中断。

```
SQL> select *from v$archive_gap;
```

这一步的目的是为了取出对应的 LOW_SEQUENCE#和 HIGH_SEQUENCE#，如果有 RAC 的话，还需要注意一下 THREAD#。

2、查找 sequence 对应的归档文件

```
SQL> SELECT NAME FROM V$ARCHIVED_LOG WHERE THREAD#=1 AND DEST_ID=1 AND SEQUENCE# BETWEEN 7 AND 10;
```

3、复制对应的归档文件到 standby

注意，如果是 RAC 的话，要找对机器哟:)

然后通过 alter database register logfile 命令将这些文件重新注册一下，例如：

```
SQL> ALTER DATABASE REGISTER LOGFILE 'e:\ora10g\jsspdg\xxxx.arc';
```

然后重启 redo 应用即可。

对于逻辑 standby 的丢失归档处理稍微复杂一点点，因为逻辑 standby 没有提供类型 v$archive_gap;之类的视图，因此我们需要自己想办法来识别是否存在丢失的情况，具体可以通过下列的语句：

```
SQL> select thread#,sequence#,file_name from dba_logstdby_log l
  2  where next_change# not in (
  3  select first_change# from dba_logstdby_log where l.thread# = thread#)
  4  order by thread#,sequence#;
```

找出丢失的归档文件后，接着的处理方式与物理 standby 相同。

第四部分 Standby 之选择数据保护模式　2007.12.18

关于有三模同学的光荣事迹大家应该都听说了，有不认识的请自觉重温"名词先混个脸熟"篇，下面让三思就有三模同学的高超本领为大家做个展示。

为了便于大家更好的理解，我们先画一个表，表中描述了不同保护模式下 LOG_ARCHIVE_DEST_n 参数应该设置的属性：

	最大保护	最高可能用	最高性能
REDO 写进程	LGWR	LGWR	LGWR 或 ARCH
网络传输模式	SYNC	SYNC	LGWR 进程时 SYNC 或 ASYNC，ARCH 进程时 SYNC
磁盘写操作	AFFIRM	AFFIRM	AFFIRM 或 NOAFFIRM
是否需要 standby redologs	YES	YES	可没有但推荐有

提示：

上面中的各项需求都是满足该保护模式的最低需求，这些需求都是据其特性而定的，同时你也一定要理解，这些需求仅只是保证你完成 data guard 保护模式的设置，如果你真正理解其特性的需求所希望满足的原因，你还需要做不少其它必要的工作。比如对于最高可用性而言，其根本目地是为了在某一台甚至多台主或备库瘫痪时，数据库仍能够在极短时间内恢复服务，这就不仅需要你将最高可用性保护模式的参数配置正确，还需要你拥有足够多的 standby 数据库。听明白了没？啥，木有？555555555，你在打击俺语言描述的能力~~~

另外再强调一遍：最大保护和最高可用性都要求 standby 数据库配置 standby redo logs(当然如果考虑角色切换的话，主库肯定也是需要配置的)，关于 standby redo logs 的故事你可以参考第二部分的第一章 1.3。

下面我们进入实践将一个 data guard 配置从最高性能模式改为最高可用性模式：

1、首先查看当前的保护模式　---primary 数据库操作

```
SQL> select protection_mode,protection_level from v$database;

PROTECTION_MODE          PROTECTION_LEVEL
-------------------- --------------------
MAXIMUM PERFORMANCE    MAXIMUM PERFORMANCE
```

2、修改初始化参数　--primary 数据库操作

```
SQL> alter system set log_archive_dest_2='SERVICE=jsspdg
  2    OPTIONAL LGWR SYNC AFFIRM VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
  3    DB_UNIQUE_NAME=jsspdg';

系统已更改。
```

3、设置新的数据保护模式并重启数据库　--primary 数据库操作

语句非常简单，如下：

```
SQL> alter database set standby database to maximize availability;
```

```
数据库已更改。
```

提示：maximize 后可跟{PROTECTION | AVAILABILITY | PERFORMANCE}，分别对应最大保护，最高可用性及最高性能。

Down 掉数据库，重新启动

```
SQL> shutdown immediate
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> startup
ORACLE 例程已经启动。

Total System Global Area    167772160 bytes
Fixed Size                    1289484 bytes
Variable Size               121635572 bytes
Database Buffers             37748736 bytes
Redo Buffers                  7098368 bytes
数据库装载完毕。
数据库已经打开。
```

4、看一下当前的保护模式　　--primary 数据库操作

```
SQL> select protection_mode,protection_level from v$database;

PROTECTION_MODE          PROTECTION_LEVEL
-------------------- --------------------
MAXIMUM AVAILABILITY MAXIMUM AVAILABILITY
```

5、修改 standby 初始化参数设置(主要考虑角色切换，如果只测试的话本步可跳过)　　---standby 数据库操作

```
SQL> alter system set log_archive_dest_2='SERVICE=jssweb OPTIONAL LGWR SYNC AFFIRM
  2    VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=jssweb';

系统已更改。
```

查看当前的保护模式

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

INSTANCE_NAME
----------------
jsspdg

SQL> select protection_mode,protection_level from v$database;
```

```
PROTECTION_MODE          PROTECTION_LEVEL
-------------------- --------------------
MAXIMUM AVAILABILITY MAXIMUM AVAILABILITY
```

配置成功，正面顺便再测试一下。

6、停掉 standby 数据库，再查看 primary 数据库状态

```
SQL> select protection_mode,protection_level from v$database;

PROTECTION_MODE          PROTECTION_LEVEL
-------------------- --------------------
MAXIMUM AVAILABILITY RESYNCHRONIZATION
```

Standby 数据库 shutdown 后，primary 数据库保护级别切换为待同步。

第四部分 Standby 之 Log 应用服务　2008.02.26

前面我们已经接触了很多相关的概念，我们都知道 DataGuard 通过应用 redo 维持 primary 与各 standby 之间的一致性，在后台默默无闻支撑着的就是传说中的 Log 应用服务。Log 应用服务呢，又分两种方式，一种是 redo 应用(物理 standby 使用，即介质恢复的形式)，另一种是 sql 应用(逻辑 standby 使用，通过 LogMiner 分析出 sql 语句在 standby 端执行)。

一、Log 应用服务配置选项

1、REDO 数据实时应用

默认情况下，log 应用服务会等待单个归档文件全部接收之后再启动应用(在前面 redo 传输服务中我们介绍了不同形式的传输方式)，如果 standby 端使用了 standby redologs，就可以打开实时应用(real-time apply)，这样 dg 就不需要再等待接收完归档文件，只要 rfs 将 redo 数据写入 standby redologs，即可通过 MRP/LSP 实时写向 standby。

物理 standby 启用实时应用通过下列语句：

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE ;
```

逻辑 standby 启用实时应用通过下列语句：

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

2、REDO 数据延迟应用

有实时就有延迟，某些情况下你可能不希望 standby 与 primary 太过同步:)，那就可以在 log_archive_dest_n 参数中指定 delay 属性(单位为分钟，如果指定了 delay 属性，但没有指定值，则默认是 30 分钟）。注意，该属性并不是说延迟发送 redo 数据到 standby，而是指明归档到 standby 后，开始应用的时候。

不过，即使在 log_archive_dest_n 中指定了 delay 属性，但如果你应用数据时指定了实时应用，则 standby 会忽略 delay 属性。另外，standby 端还可以通过下列的语句取消延迟应用。

物理 standby 取消延迟应用可以通过下列语句：

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE NODELAY;
```

逻辑 standby 取消延迟应用可以通过下列语句：

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NODELAY;
```

提示：flashback database 也可视为延迟应用的一种方式。

二、应用 redo 数据到物理 standby

注意：启动 redo 应用，物理 standby 需要首先启动到 mount 状态，然后再执行下列语句启动，或者停止 redo 应用。

1、启动 redo 应用

- 前台应用

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE;
```

- 后台应用

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT;
```

- 启动实时应用，附加 USING CURRENT LOGFILE 子句即可

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE;
```

2、停止 redo 应用

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

三、应用 redo 数据到逻辑 standby

注意：启用 sql 应用，逻辑 standby 需要启动至 open 状态。

1、启动 sql 应用

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
```

如果要启动实时应用，附加 immediate 子句即可：

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

2、停止 sql 应用

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

第四部分 附章(1)RMAN 备份来创建 2008.01.22

如果你看过三思之前的几个笔记系列，那么对于 rman 想必已经非常熟悉，操作这个必然也不成问题，如果你还没有看过，建议你先回去看看，然后再回来操作必然也没有问题，如果你一定不准备看，没关系，只要你严格按照实践部分的步骤操作，我相信你一定也可以创建成功，操作应该也没有问题，不过如果这样你也觉着没有问题，那么我要告诉你，可能就这是最大的问题：不知道做过什么，不知道该做什么，不知道为什么要做，一旦需求稍变，你甚至什么都不敢做。

你什么都不用说，我知道，你还有问题，下面，我们先来看你下意识的第一个问题，为什么要用 rman 创建物理 standby？Oracle 告诉我们，有三点：

- RMAN 创建 standby 是通过 primary 的备份，因此不会对 primary 有任何的影响
- RMAN 自动重命名 OMF 的文件及路径结构。
- RMAN 修复归档日志文件并执行恢复以尽可能保证 standby 与 primary 数据一致相同。

当然，我们也应该知道，上述这些都是形容词，它只是为了强化意识，说到这里再多白话几句，第一条呢还说的过去（虽然你不用 rman 备份，使用其它方式的备份创建 standby 也不会对 primary 造成影响），第二条第三条就完全不靠谱了，并不是说它实现不了，恰恰相反，是它描述的太基础了，形容手法有问题，我举个例子，比如你在鱼缸里看到一条鱼，你会不会形容说哇这条鱼能够在水里游耶(死鱼才不会在水里游呢)~~所以鉴别能力很重要，虽然这点我做的还很不够，但是请首长们放心，我一定会努力的，我一定会加强的，我一定会坚持的！！！

回到这个问题上来，为什么要用 rman 来创建物理 standby 呢，在我看来如果说有优势那么就一点：简单！

另外在这里三思更明确的指出，使用 RMAN 的 duplicate 命令只能直接创建物理 standby，幸还是不幸？

一、准备工作

注意，在做任何操作之前，需要确认以下几点：

- 拥有至少一份通过 rman 创建的备份；
- 已经在 primary 数据库设置了所有相关的初始化参数；
- 已经创建了 standby 的初始化参数文件并配置了所有相关的初始化参数；
- 已经配置了实例，NetService,Listener 等；
- 启动 standby 实例到 nomount 状态；

然后：

1、通过 rman 创建 standby 的控制文件

创建 standby 的控制文件前面我们提到通过 sql 命令，使用非常简单，使用 rman 命令创建与之同理，并且有两种方式创建演示如下：

```
1).
RMAN> backup current controlfile for standby;
2).
RMAN> copy current controlfile for standby to 'e:\ora10g\oradata\jssrman\JSSRMAN01.CTL';
```

2、定制 standby 数据(日志)文件重命名策略

为什么 oracle 要提供重命名策略呢？因为 dg 配置非常灵活，standby 甚至可以与 primary 在同一个数据库。

什么时候需要应用重命名策略呢？如果 standby 与 primary 在同一台服务器，或虽然不在同一台服务器，但 standby 的目录结构与 primary 不同，这两种情况下，都必须应用重命名策略。如果 standby 与 primary

不在同一台服务器，并且目录结构相同，那就不需要应用重命名策略。

如何应用重命名策略呢？多种方式，比如我们的老朋友初始化参数：db_file_name_convert,log_file_name_convert。还有 rman 命令 SET NEWNAME 或 CONFIGURE AUXNAME 等等，这些相关参数、命令的应用我们都在"Duplicate 复制数据库"系列中介绍并应用过，后面还会再次提及。

二、大致流程

通常情况下，rman 创建完 standby 之后不会自动执行 recover。

如果你执行 duplicate 命令时没有指定 dorecover 参数，则 rman 自动按照下面的步骤操作：

1、RMAN 连接 standby 与 primary，及 catalog(如果使用了的话)；

2、检索 catalog(nocatalog 的话是 primary 数据库的控制文件)，确定 primary 的备份以及 standby 控制文件；

3、如果使用介质恢复，RMAN 需要连接介质管理器以获取所需备份数据；

4、恢复 standby 控制文件到 standby 服务器；

5、恢复 primary 数据库备份集中相应数据文件到 standby 服务器；

6、rman 自动将 standby 数据库打开到 mount 状态，不过不会自动打开 redo 应用。

如果执行 duplicate 命令时指定了 dorecover 参数，则 rman 会在执行完第 5 步后，接着执行下列的操作：

7、在所有数据都 restored 之后，rman 自动执行 recovery，如果 recovery 过程需要归档文件，但是这些文件又不在本地盘，则 rman 会尝试从备份中获取。

8、rman 执行 recovery 之前，你可以通过指定 time,scn,logfile sequence 来确定 recovery 的内容，如果什么都不指定，则 rman 一直 recover 到最后一个归档文件。

9、rman 自动将 standby 数据库打开到 mount 状态，同样也不会自动打开 redo 应用。

三、方法及步骤

基本上，可以分成二类：

1、相同目录结构的创建

duplicate 不同服务器相同目录结构创建 standby 的操作极为简单，你即不需要动用 DB_FILE_NAME_CONVER/LOG_FILE_NAME_CONVERT之类参数，也不需要通过 set newname 之类命令，基本步骤如下：

1) 确保已设置 standby 服务器中所有相关的初始化参数。
2) 确认备份集中文件 scn 大于或等于控制文件中的 scn。
3) 如果需要，可以通过 set 命令指定 time,scn 或 log 序号以执行不完全恢复。
   例如：set until scn 152;
   提示：注意如果有 set，则 set 与 duplicate 必须在同一个 run 命令块中。
4) 如果没有配置自动分配通道的话，需要手工指定至少一条辅助通道。
5) 务必指定 nofilenamecheck 参数，我们之前"duplicate 复制数据库"系列中就曾提到过，异机操作路径相同还必需指定 NOFILENAMECHECK。因为此处 oracle 表现的很傻，它不知道你要恢复的路径是在另一台机器上，它只是认为要恢复到的路径怎么跟目标数据库表现的一样呢？会不会是要覆盖目标数据库啊，为了避免这种情形，于是它就报错。所以一旦异机恢复，并且路径相同，那么你必须通过指定 NOFILENAMECHECK 来避免 oracle 的自动识别。

例如脚本如下：

```
sql> duplicate target database for standby nofilenamecheck dorecover;
```

注意，dorecover 并非是必须参数，如果你不指定的话，则 duplicate 修复数据文件到服务器，并自动将 standby 启动到 mount 状态，不过并不会执行恢复操作。

2、不同目录结构的创建

对于不同目录结构创建 standby(与是否同一台服务器就基本无关了)，你需要对数据文件和日志文件路径重新定义，那你的选择可就多多了哟：

a. 使用初始化参数重定义数据文件及日志文件

关于 db_file_name_convert 和 log_file_name_convert 两个初始化参数的本领和套路大家已经都很熟悉了，所以呢这里就不多做介绍。duplicate 命令在此处执行的时候与相同目录结构执行也没什么不同，所以，你可以认为，这是不同路径下创建 standby 中，最简单的方式。

b. SET NEWNAME 命令重定义数据文件

步骤如下：

- 确保已设置 standby 服务器中所有相关的初始化参数。
- 确认备份集中文件 scn 大于或等于控制文件中的 scn。
- 如果需要，可以通过 set 命令指定 time,scn 或 log 序号以执行不完全恢复。
- 如果没有配置自动分配通道的话，需要手工指定至少一条辅助通道。
- 通过 set newname 命令为 standby 数据文件指定新路径
- 执行 duplicate 命令。

例如，脚本如下：

```
RUN
{
  # Set new file names for the datafiles
  SET NEWNAME FOR DATAFILE 1 TO '?/dbs/standby_data_01.f';
  SET NEWNAME FOR DATAFILE 2 TO '?/dbs/standby_data_02.f';
  . . .
  DUPLICATE TARGET DATABASE FOR STANDBY DORECOVER;
}
```

c. CONFIGURE AUXNAME 命令重定义数据文件

操作步骤皆与上同，不再详述，不过需要注意的是 CONFIGURE AUXNAME 命令的格式，并且 configure 命令不能在 run 块中执行，例如脚本如下：

```
# set auxiliary names for the datafiles
CONFIGURE AUXNAME FOR DATAFILE 1 TO '/oracle/auxfiles/aux_1.f';
CONFIGURE AUXNAME FOR DATAFILE 2 TO '/oracle/auxfiles/aux_2.f';
. . .
CONFIGURE AUXNAME FOR DATAFILE n TO '/oracle/auxfiles/aux_n.f';
DUPLICATE TARGET DATABASE FOR STANDBY;
```

最后，务必注意，configure auxname 命令执行是一直生效的，因此 duplicate 执行完之后，推荐清除 CONFIGURE AUXNAME。这样就不会对未来的类似操作造成影响。

例如：

```
CONFIGURE AUXNAME FOR DATAFILE 1 CLEAR;
CONFIGURE AUXNAME FOR DATAFILE 2 CLEAR;
. . .
CONFIGURE AUXNAME FOR DATAFILE n CLEAR;
```

步骤和方法介绍完了，下面实际操作一把~~~~~~~~~~~~~~

第四部分 附章(2)RMAN 备份来创建之实践 2008.01.28

目标：通过 rman 备份，在同一台服务器创建 standby，primary 数据库名为 jssweb，要创建的 standby，db_unique_name 命名为 jssrman，因为是在同一台服务器，所以需要对 standby 的数据文件和日志文件路径做下重定义，这里我们选择通过初始化参数的形式重定义文件路径。

一、做好准备工作

1、创建 standby 实例

需要首先通过 ORADIM 命令创建一个新的 OracleService，非 windows 环境可以跳过这一步。

```
E:\ora10g>oradim -new -sid jssrman
实例已创建。

E:\ora10g>set oracle_sid=jssrman

E:\ora10g>sqlplus "/ as sysdba"
.....
```

顺手配置一下监听，修改 tnsname.ora 之类，此处不做演示。

2、创建 standby 的初始化参数文件

没必要完全重建，首先根据 primary 的 spfile 生成一个 pfile 出来，然后比着改改就是了。例如：

在 primary 数据库执行，注意是 primary 数据库：

```
SQL> create pfile='E:\ora10g\product\10.2.0\db_1\database\INITjssrman.ora' from spfile;
```

用任意文本编辑器打开 INITjssrman.ora，注意修改相关的化参数，此例中主要修改下列属性：

```
*.audit_file_dest='e:\ora10g\product\10.2.0\admin\jssrman\adump'
*.background_dump_dest='e:\ora10g\product\10.2.0\admin\jssrman\bdump'
*.control_files='E:\ora10g\oradata\jssrman\control01.ctl','E:\ora10g\oradata\jssrman\control02.ctl','E:\ora10g\oradata\jssrman\control03.ctl'
*.core_dump_dest='e:\ora10g\product\10.2.0\admin\jssrman\cdump'
*.DB_FILE_NAME_CONVERT='oradata\jssweb','oradata\jssrman'
*.db_name='jssweb'
*.DB_UNIQUE_NAME='jssrman'
*.LOG_ARCHIVE_CONFIG='DG_CONFIG=(jssweb,jsspdg,jssrman)'
*.LOG_ARCHIVE_DEST_1='LOCATION=E:\ora10g\oradata\jssrman\
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=jssrman'
*.LOG_ARCHIVE_DEST_STATE_1='ENABLE'
*.LOG_FILE_NAME_CONVERT='oradata\jssweb','oradata\jssrman'
*.standby_file_management='AUTO'
*.user_dump_dest='e:\ora10g\product\10.2.0\admin\jssrman\udump'
```

注意，由于此 standby 数据库不考虑切换，因为不再配置 fal_*,等相关初始化参数。

然后通过该 pfile 创建 standby 的 spfile，注意是在 standby 数据库执行：

```
SQL> create spfile from pfile='E:\ora10g\product\10.2.0\db_1\database\initjssrman.ora';
```

3、启动 standby 到 nomount 状态

```
SQL> startup nomount
ORACLE 例程已经启动。

Total System Global Area    167772160 bytes
.....................
```

4、创建 standby 的密码文件

注意保持密码与 primary 数据库相同，非常重要。

```
E:\ora10g>orapwd file=e:\ora10g\product\10.2.0\db_1\database\PWDjssrman.ora password=verysafe entries=30
```

二、进行创建工作

1、连接 primary 和 standby

注意连接 standby 的时候前面指定 auxiliary

```
E:\ora10g>set oracle_sid=jssweb

E:\ora10g>rman target / auxiliary sys/tfad04@jssrman

恢复管理器: Release 10.2.0.3.0 - Production on 星期三 1 月 30 13:53:13 2008

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

已连接到目标数据库: JSSWEB (DBID=3408827880, 未打开)
已连接到辅助数据库: JSSWEB (未装载)
```

2、创建 standby 的控制文件。

我们之前都操作过通过 sql 命令创建 standby 的控制文件，这次换一换，在 rman 中创建 standby 的控制文件。

```
RMAN> copy current controlfile for standby to 'e:\ora10g\oradata\jssrman\control01.ctl';

启动 backup 于 29-1 月 -08
使用目标数据库控制文件替代恢复目录
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: sid=157 devtype=DISK
通道 ORA_DISK_1: 启动数据文件副本
复制备用控制文件
输出文件名 = E:\ORA10G\ORADATA\JSSRMAN\CONTROL01.CTL 标记 = TAG20080129T150422 recid = 6 时间戳 = 645289463
通道 ORA_DISK_1: 数据文件复制完毕, 经过时间: 00:00:03
```

```
完成 backup 于 29-1 月 -08

启动 Control File and SPFILE Autobackup 于 29-1 月 -08
段 handle=D:\BACKUP\C-3408827880-20080129-00 comment=NONE
完成 Control File and SPFILE Autobackup 于 29-1 月 -08
```

3、生成 standby 数据库

由于我们此时的备份集是刚刚的，所以在执行 duplicate 命令时并不需要指定 dorecover 子句，而且我们采用了通过 db_file_name_convert 等初始化参数的方式重定义数据文件和日志文件路径，因为执行的命令就非常简单，就这是我们前面说的，不同路径下创建 standby 中，最简单的方式：

```
RMAN> duplicate target database for standby;

启动 Duplicate Db 于 30-1 月 -08
使用目标数据库控制文件替代恢复目录
分配的通道: ORA_AUX_DISK_1
通道 ORA_AUX_DISK_1: sid=155 devtype=DISK

内存脚本的内容:
{
   restore clone standby controlfile;
   sql clone 'alter database mount standby database';
}
正在执行内存脚本

启动 restore 于 30-1 月 -08
使用通道 ORA_AUX_DISK_1

通道 ORA_AUX_DISK_1: 正在复原控制文件
通道 ORA_AUX_DISK_1: 已复制控制文件副本
输出文件名=E:\ORA10G\ORADATA\JSSRMAN\CONTROL01.CTL
输出文件名=E:\ORA10G\ORADATA\JSSRMAN\CONTROL01.CTL
输出文件名=E:\ORA10G\ORADATA\JSSRMAN\CONTROL02.CTL
输出文件名=E:\ORA10G\ORADATA\JSSRMAN\CONTROL03.CTL
完成 restore 于 30-1 月 -08

sql 语句: alter database mount standby database
释放的通道: ORA_AUX_DISK_1

内存脚本的内容:
{
   set newname for tempfile  1 to
 "E:\ORA10G\ORADATA\JSSRMAN\TEMP01.DBF";
   switch clone tempfile all;
   set newname for datafile  1 to
```

```
 "E:\ORA10G\ORADATA\JSSRMAN\SYSTEM01.DBF";
    set newname for datafile  2 to
 "E:\ORA10G\ORADATA\JSSRMAN\UNDOTBS01.DBF";
    set newname for datafile  3 to
 "E:\ORA10G\ORADATA\JSSRMAN\SYSAUX01.DBF";
    set newname for datafile  4 to
 "E:\ORA10G\ORADATA\JSSRMAN\USERS01.DBF";
    set newname for datafile  5 to
 "E:\ORA10G\ORADATA\JSSRMAN\TBSWEB01.DBF";
    restore
    check readonly
    clone database
    ;
}
正在执行内存脚本

正在执行命令: SET NEWNAME

临时文件 1 在控制文件中已重命名为 E:\ORA10G\ORADATA\JSSRMAN\TEMP01.DBF

正在执行命令: SET NEWNAME

正在执行命令: SET NEWNAME

正在执行命令: SET NEWNAME

正在执行命令: SET NEWNAME

正在执行命令: SET NEWNAME

启动 restore 于 30-1 月 -08
分配的通道: ORA_AUX_DISK_1
通道 ORA_AUX_DISK_1: sid=155 devtype=DISK

通道 ORA_AUX_DISK_1: 正在开始恢复数据文件备份集
通道 ORA_AUX_DISK_1: 正在指定从备份集恢复的数据文件
正将数据文件 00001 恢复到 E:\ORA10G\ORADATA\JSSRMAN\SYSTEM01.DBF
正将数据文件 00002 恢复到 E:\ORA10G\ORADATA\JSSRMAN\UNDOTBS01.DBF
正将数据文件 00003 恢复到 E:\ORA10G\ORADATA\JSSRMAN\SYSAUX01.DBF
正将数据文件 00004 恢复到 E:\ORA10G\ORADATA\JSSRMAN\USERS01.DBF
正将数据文件 00005 恢复到 E:\ORA10G\ORADATA\JSSRMAN\TBSWEB01.DBF
通道 ORA_AUX_DISK_1: 正在读取备份段 D:\BACKUP\04J6V1SJ_1_1
通道 ORA_AUX_DISK_1: 已恢复备份段 1
段句柄 = D:\BACKUP\04J6V1SJ_1_1 标记 = TAG20080124T111011
```

```
通道 ORA_AUX_DISK_1: 恢复完成, 用时: 00:00:57
完成 restore 于 30-1 月 -08

内存脚本的内容:
{
   switch clone datafile all;
}
正在执行内存脚本

数据文件 1 已转换成数据文件副本
输入数据文件副本 recid=11 stamp=645372185 文件名
=E:\ORA10G\ORADATA\JSSRMAN\SYSTEM01.DBF
数据文件 2 已转换成数据文件副本
输入数据文件副本 recid=12 stamp=645372185 文件名
=E:\ORA10G\ORADATA\JSSRMAN\UNDOTBS01.DBF
数据文件 3 已转换成数据文件副本
输入数据文件副本 recid=13 stamp=645372186 文件名
=E:\ORA10G\ORADATA\JSSRMAN\SYSAUX01.DBF
数据文件 4 已转换成数据文件副本
输入数据文件副本 recid=14 stamp=645372186 文件名
=E:\ORA10G\ORADATA\JSSRMAN\USERS01.DBF
数据文件 5 已转换成数据文件副本
输入数据文件副本 recid=15 stamp=645372186 文件名
=E:\ORA10G\ORADATA\JSSRMAN\TBSWEB01.DBF
完成 Duplicate Db 于 30-1 月 -08
```

4、重建 standby 临时表空间数据文件

提示：本步非必须，如果该 standby 不会切换为 primary 并且也不会有查询需求，就不必创建！

三、完成善后工作

善后工作通常很不起眼但是很重要，

1、修改 primary 数据库中的相关参数

```
SQL> show parameter db_unique

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_unique_name                       string      jssweb
SQL> set sqlprompt Jssweb>
Jssweb> alter system set log_archive_config='DG_CONFIG=(jssweb,jsspdg,jssrman)';

系统已更改。
```

```
Jssweb> alter system set log_archive_dest_3='SERVICE=jssrman lgwr async valid_for=(online_logfiles,primary_role) db_unique_name=jssrman';

系统已更改。

Jssweb> alter system set log_archive_dest_state_3=enable;

系统已更改。
```

2、考虑到为保证切换后，dg 仍能正常运转，同时修改待切换的 standby 数据库初始化参数

```
SQL> show parameter db_unique

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_unique_name                       string      jsspdg
SQL> set sqlprompt Jsspdg>
Jsspdg> alter system set log_archive_config='DG_CONFIG=(jssweb,jsspdg,jssrman)';

系统已更改。

Jsspdg> alter system set log_archive_dest_3='SERVICE=jssweb lgwr async valid_for=(online_logfiles,primary_role) db_unique_name=jssweb';

系统已更改。

Jsspdg> alter system set log_archive_dest_state_3=enable;

系统已更改。
```

3、打开 standby 的 redo 应用

```
SQL> show parameter db_unique

NAME                                 TYPE                 VALUE
------------------------------------ -------------------- ------------------------------
db_unique_name                       string               jssrman
SQL> set sqlprompt Jssrman>
Jssrman> alter database recover managed standby database disconnect from session;

数据库已更改。
```

4、Primary 切换日志，验证同步效果

```
Jssweb> alter system switch logfile;

系统已更改。
```

```
Jssweb>select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
           787

Jsspdg>select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
           787

Jssrman>select max(sequence#) from v$archived_log;

MAX(SEQUENCE#)
--------------
           787
```

与之前通过 primary 物理备份相比，通过 rman 的 duplicate 命令创建 standby，实际执行的步骤是不是更简单一些了呢，基本上你只需要记住 duplicate 的用法就好了，其它工作 rman 都自动帮你干。正象开篇中我说过的那样，为什么要选择通过 rman 来创建 standby 呢，因为简单:)

[三思笔记]使用可传输表空间的特性复制数据
http://www.itpub.net/926949.html

[三思笔记]日期时间及数字的格式化参数大全
http://www.itpub.net/913307.html

[三思笔记]**RMAN** 高级应用之 **Duplicate** 复制数据库
http://www.itpub.net/906598.html

[三思笔记]**RHEL AS4** 下升级 **oracle10g** 到 **10.2.0.3**
http://www.itpub.net/896394.html

[三思笔记]**RHEL AS4** 下安装 **32** 位 **oracle10g**
http://www.itpub.net/884137.html

[三思笔记]**Statspack** 初步学和用
http://www.itpub.net/857807.html

[三思笔记]**oracle** 著名及非著名函数介绍
http://www.itpub.net/843333.html

[三思笔记]一步一步学 **rman**
http://www.itpub.net/810100.html

[三思笔记]学习动态性能表
http://www.itpub.net/782892.html