中数国科容灾与备份
技术白皮书

2023 年 8 月

目录

一、备份术语定义 3
1.1.灾备： 3
1.2.容灾： 3
1.3.CDP： 3
1.4.RPO： 5
1.5.RTO： 5
1.6 数据去重： 5
二、备份架构介绍 6
2.1.管理平台： 7
2.2.备份服务器： 7
2.3.备份代理服务器： 7
2.4.备份介质服务器： 7
2.5.备份客户端： 7
三、产品模块功能与技术介绍 9
3.1.虚拟机备份 9
3.2.数据库备份 10
3.3.文件备份 26
3.4.操作系统备份 29
3.4.卷备份 37
四、产品全局功能与技术介绍 46
4.1.数据压缩 46
4.2.数据重删 50
4.3.数据加密存储 52
五、断点续传 57
5.1.功能简介 57
5.2.技术实现 57
5.3.SpeedX 备份加速 58
5.4.虚拟化平台 58
5.5.RESETful API 61

一、备份术语定义

1.1.灾备：

灾难备援，它是指利用科学的技术手段和方法，提前建立系统化的数据应急方式，以应对灾难的发生。

1.2.容灾：

容灾系统是指在相隔较远的异地，建立两套或多套功能相同的 IT 系统，互相之间可以进行健康状态监视和功能切换，当一处系统因意外（如火灾、地震等）停止工作时，整个应用系统可以切换到另一处，使得该系统功能可以继续正常工作。

1.3.CDP：

持续数据保护，是一种在不影响主要数据运行的前提下，可以实现持续捕捉或跟踪目标数据所发生的任何改变，并且能够恢复到此前任意时间点的方法。CDP 系统能够提供块级、文件级和应用级的备份，以及恢复目标的无限的任意可变的恢复点。

1.3.1.定时备份：

是指有时间间隔的数据备份方式，比如一天一次，一周一次，或者一个月一次，因此定时备份是对数据进行周期性备份，存在备份时间窗口。

1.3.2.实时备份：

实时备份在任意时间间隔内对数据进行备份，无备份时间窗口，保障数据的零丢失。

1.3.3.完全备份：

用存储介质对整个数据及系统进行完全备份。这种备份方式的好处就是很直观， 容易被人理解，易恢复；缺点则是在备份数据中有大量重复数据，由于需要备份的数据量相当大， 因此备份所需时间较长。

1.3.4.增量备份：

每次备份的数据只是相当于上一次备份后增加的和修改过的数据。这种备份的优点很明显：重复数据少，即节省磁带空间，又缩短了备份时间；缺点在于当发生灾难时，数据恢复比较麻烦。

1.3.5.差异备份：

是备份所有新产生的或更新的数据，这些数据都是最近一次全量备份后产生的或更新的。差量备份与增量备份的区别是，增量备份判断数据更新标准是依据上一次备份检查点，而差量备份一定是依据全量备份检查点。如没有全量备份，则就没有差量备份。差量备份的主要目的是 限制完全恢复时使用的介质数量。

1.3.6.合成备份：

合成备份的含义就是将一个全备份和一些增量备份或者差分备份重新组成一个全备份，这样在恢复的时候，只用恢复一个备份即可，大大缩短了备份时间，提高了备份效率。

1.3.7.在线热备：

即在备份过程中，备份的目标对象如操作系统，数据库或虚拟机仍可以正常使用，无需关闭或做其他影响运行的配置。

1.3.8.有效数据备份：

即备份发生时，只需读取和传输目标对象必要的数据组成，例如备份分区时只传输了文件系统已分配的数据区域。

1.3.9.原机恢复：

将备份数据恢复到备份源，例如恢复到原虚拟机，恢复到原操作系统等。

1.3.10.异机恢复：

将备份数据恢复到非备份源所在的机器。

1.4.RPO：

所谓 RPO，Recovery Point Objective（恢复点目标），是指从系统和应用数据而言，要实现能够恢复至可以支持各部门业务运作，系统及生产数据应恢复到怎样的更新程度。这种更新程度可以是上一周的备份数据，也可以是上一次交易的实时数据。

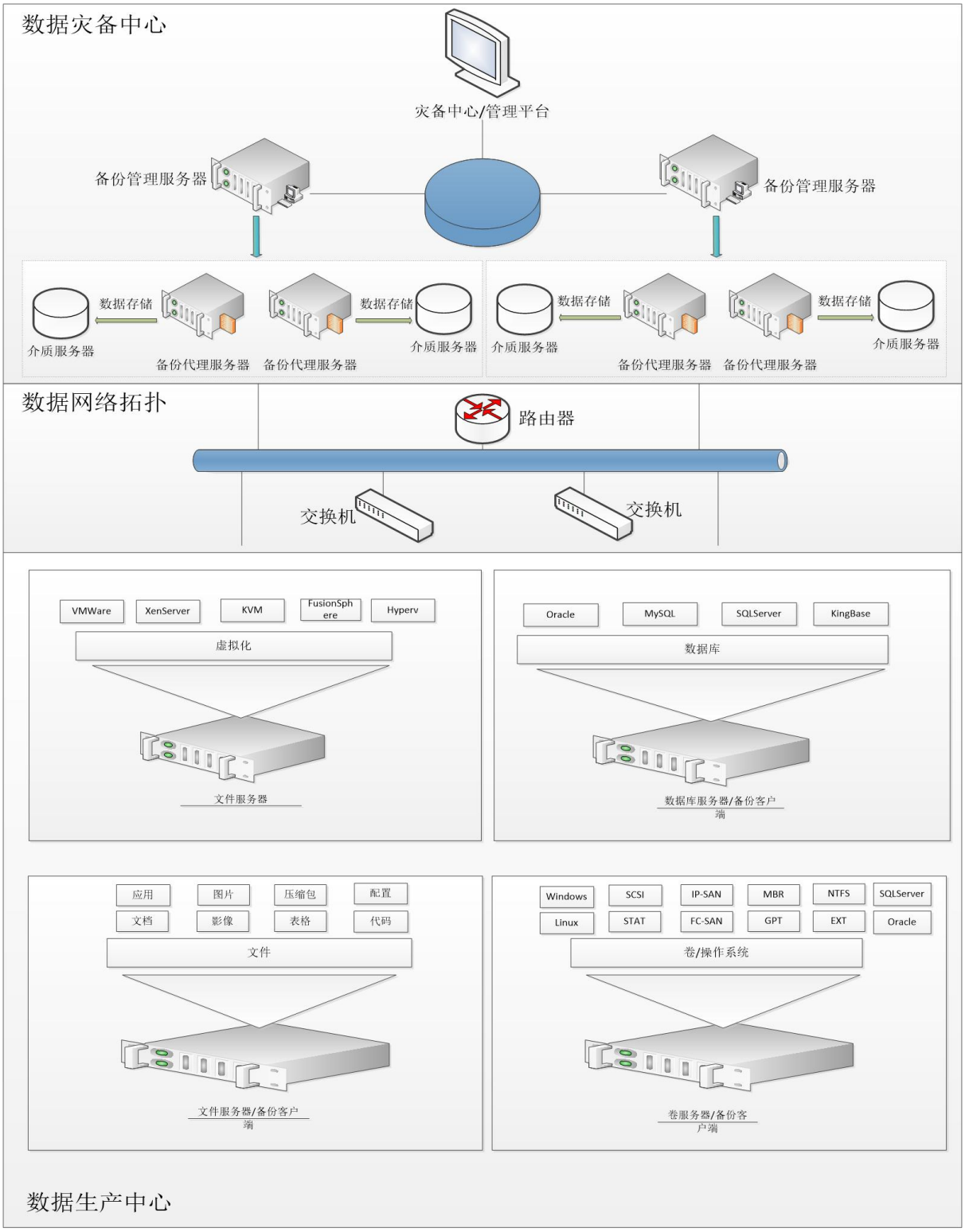
1.5.RTO：

所谓 RTO，Recovery Time Objective（恢复时间点目标），它是指灾难发生后，从 IT 系统当机导致业务停顿之时开始，到 IT 系统恢复至可以支持各部门运作、恢复运营之时，此两点之间的时间段称为 RTO。

1.6 数据去重：

一种数据缩减技术，通常用于基于磁盘的备份系统，旨在减少存储系统中使用的存储容量。它的工作方式是在某个时间周期内查找不同文件中不同位置的重复可变大小数据块。重复的数据块用指示符取代。

二、备份架构介绍



备份架构示意图

2.1.管理平台：

产品的管理结构属于 B/S 架构，管理平台是由 Apache 搭建的 WEB 服务器，在网络能够连通的任何终端上通过 WEB 浏览器便可以进行登录、访问、控制。

WEB 服务器可管理一个或多个备份服务器，在部署形态上，可独立部署，也可以内置在某个备份服务器中。

管理平台支持 HTTPS 访问，支持多用户管理，有超时锁定，会话锁定，可查看历史访问记录，密码过期时间设置等安全策略。

2.2.备份服务器：

备份服务器是备份代理与备份仓库的集合，内置备份控制程序，响应对 WEB 管理平台发起的操作，主要包括管理作业的控制运行，备份数据的生命周期，生产源端的注册与更新。

每个备份服务器默认内置一个备份代理，即在逻辑上默认包含一个 IO 管理单元，并可以添加多个数据存储仓库。因此一个备份服务器通常已经是一个备份架构的最小组成。

当然，每个备份服务器可以挂载更多的备份代理服务器与备份介质服务器，且每个管理平台可以挂载多个备份服务器，理论上该架构可以无限横向扩展。

2.3.备份代理服务器：

备份代理主要是指承载数据 IO 发生的逻辑控制单元。如果将控制程序的角色定义为备份代理，并装载在一个操作系统平台上，便组成了一个备份代理服务器。

独立的备份代理服务器必须被备份服务器管理，备份服务器将 IO 作业分发给备份代理服务器，由其完成数据的传输与存储任务。因此当生产源端具有较大的数据量和备份需求时，可通过添加多个备份代理服务器来分发 IO 负载，以达到更优的并发性能。

2.4.备份介质服务器：

备份仓库是指用来存储备份数据的介质。备份介质服务器则是指包括了物理存储单元与逻辑存储控制程序的组成。

备份介质支持硬盘、IP-SAN、FC-SAN、NAS、磁带库、对象存储等。

2.5.备份客户端：

备份客户端是指安装部署在生产源端的程序。备份客户端主要用于响应备份服务器/备份代理发起的控制请求，完成对源端信息与数据的检索与传输。

备份发生时，备份客户端将源端的数据读取并发送到备份服务端进行存储。

恢复发生时，备份客户端则接收备份服务端发送的数据写入目标对象。

根据备份模块的划分，备份客户端在功能上也有明确的分工，主要包括虚拟化、数据库、文件、操作系统与卷的备份。

三、产品模块功能与技术介绍

3.1.虚拟机备份

3.1.1.功能简介

3.1.1.1.定时备份

支持对市场上主流虚拟化平台的定时备份与恢复功能，为云平台的安全保驾护航。

■支持 VMware vSphere、Microsoft Hyper-V、KVM、Citrix Xenserver、华为 FusionSphere、华三 CAS、云宏 CNWare 等虚拟化平台。

■无代理备份，无需在虚拟化平台上部署程序。

■支持完全备份、差异备份、增量备份、合成备份四种备份模式。

■支持指定到虚拟磁盘级别的备份。

■支持原机、新建恢复，支持恢复到不同的存储池，宿主机。

■有效数据备份（即只备份虚拟磁盘实际使用的数据区域）。

■瞬时恢复（秒级将备份的虚拟机挂载到宿主机使用）。

■更小粒度恢复（可只恢复虚拟机的某个虚拟磁盘，或只恢复虚拟机内部的某个文件）。

■支持虚拟机备份等模块的 Server-FREE，即直接将数据从存储备份到灾备站点，无需消耗网络带宽及生产系统的性能。

3.1.2.技术实现

3.1.2.1.定时备份

是面向数据中心的虚拟化平台，是下一代云数据中心的基石。采用全新电信级虚拟化内核，虚拟化平台的稳定可靠性及处理性能达到电信级要求——秒级故障检测、微秒级内核时延等。

能够融合交付计算、存储、网络、安全虚拟化资源，高效整合数据中心基础架构资源，帮助客户迈向下一代云数据中心。采用裸金属架构（X86），无需绑定操作系统即可搭建虚拟化平台。

(1)系统架构：

虚拟化平台采用裸金属架构，实现对数据中心内的计算、网络和存储等硬件资源的虚拟化管理，对上层应用提供自动化服务。其业务范围包括：虚拟计算、虚拟网络、虚拟存储、高可用性（HA）、动态资源调度（DRS）、虚拟机容灾与备份、虚拟机模板管理、集群文件系统、虚拟网络策略管理等。

虚拟化平台对外提供 API 接口，可以与第三方平台进行对接。采用基于 KVM 技术的虚拟化解决方案，基于 OpenStack 架构的虚拟化管理解决方案。能够随着 Linux 版本的升级而升级，部署时无需绑定安装 OpenStack 相关组件。

(2)平台稳定可靠：

利用实时操作系统技术，将虚拟化底层内核系统升级为实时操作系统，确保计算逻辑性正确的同时，利用微秒级内核时延，秒级故障检测等技术保障虚拟化平台的稳定运行；支持数据中心级的容灾方案，并提供从物理机、虚拟机到应用软件层面的 HA 技术，防止物理机或虚拟机宕机影响业务系统的运行，全方位一体化支撑业务系统的稳定可靠运行。

(3)全面安全防护：

针对内核层、数据层、业务层和管理层都提供了相应的安全防护机制，满足安全合规要求，为虚拟化环境构建全方位的安全防护体系；内核深度集成安全防护引擎，支持无代理防病毒和深度包检测，为虚拟化内核及虚拟机提供高效的安全防护。

(4)运维简单高效：

基于 B/S 架构的图形化管理控制台，内置系统健康度模型，一屏直观掌握虚拟化系统运转情况，图形化实时展示监控到的 CPU、内存、磁盘 I/O、网络 I/O 等关键资源数据，并支持一键导出，为管理员合理规划资源提供详尽的数据；有丰富的告警策略，并支持短信或邮件通知到管理员，独创的一键运维和拓扑视图功能，为管理员提供一站式的运维操作。

(5)全融合虚拟化：

虚拟化融合了计算、网络、存储、安全资源的虚拟化，形成弹性的数据中心资源池，实现资源的自动化调度，更好地为上层应用服务。虚拟化后，虚拟机之间完全隔离，具有独立的 CPU、内存、磁盘 I/O、网络 I/O，任何一个虚拟机发生故障，同一物理机上的其他虚拟机不受影响，且不同虚拟机间操作系统可以异构。

(6)广泛兼容，合作开放：

广泛兼容第三方软硬件平台，包括市场上主流的 x86 服务器、存储阵列、原生操作系统、网卡、HBA 卡等,通过可定制的北向 REST API 可与第三方 OpenStack 架构云平台平滑对接，携手合作伙伴提供全方位一体化解决方案。

3.2.数据库备份

3.2.1 功能简介

3.2.1.1.定时备份

支持对主流数据库定时的热备与恢复。

(1)支持 SQLServer、Oracle、MySQL、KingBase、SyBase、达梦等数据库。
(2)支持 Oracle RAC。
(3)支持完全备份、差异备份、增量备份等三种备份模式。
(4)支持备份、恢复实例、库、表空间、数据文件、用户、表。
(5)支持单表恢复。
(6)支持 Oracle 应急保护方式，先恢复全备份数据库到备机，再实时复制最新的归档日志到备机并自动前滚，以实现数据库的应急保护。
(7)支持 AIX 上 Oracle 备份。

3.2.2.技术实现

3.2.2.1.定时备份

3.2.2.1.1.SQLServer 备份技术

SQL Server 是一个关系数据库管理系统。它最初是由 Microsoft Sybase 和 Ashton-Tate 三家公司共同开发的，于 1988 年推出了第一个 OS/2 版本。在 Windows NT 推出后，Microsoft 与 Sybase 在 SQL Server 的开发上就分道扬镳了，Microsoft 将 SQL Server 移植到 Windows NT 系统上，专注于开发推广 SQL Server 的 Windows NT 版本。Sybase 则较专注于 SQL Server 在 UNIX 操作系统上的应用。

(1)主要特性：

（1）高性能设计，可充分利用 WindowsNT 的优势。
（2）系统管理先进，支持 Windows 图形化管理工具，支持本地和远程的系统管理和配置。
（3）强壮的事务处理功能，采用各种方法保证数据的完整性。
（4）支持对称多处理器结构、存储过程、ODBC，并具有自主的 SQL 语言。
SQLServer 以其内置的数据复制功能、强大的管理工具、与 Internet 的紧密集成和开放的系统结构为广大的用户、开发人员和系统集成商提供了一个出众的数据库平台。

(2)备份数据库：

脚本分为两部分，一部分是批处理文件(dbbak.bat)，一部分为 SQL 文件(dbbak.sql)，两个文件要放在同一个目录下。

批处理文件中为主文件，真正的数据库备份操作是在 SQL 文件中完成的。

1.批处理文件内容:

sqlcmd -S 192.168.56.36 -Usa -Ppassword -i .\dbbak.sql -o .\dbbak.log

说明：

sqlcmd -S <数据库 IP 地址> -U<数据库用户> -P<数据库密码> -i <需执行的 SQL 文件名称> -o <执行结果日志文件>

sqlcmd：为 Sqlsever2005 后自带的命令行工具，可以执行 SQL 文件。

2.SQL 文件内容如下:

declare @date nvarchar(10) --定义日期变量

set @date = CONVERT(nvarchar(10),getdate(),112) --为日期变量赋当前日期，日期格式为 yyyymmdd 举例 20170830

declare @path nvarchar(250) -- 定义备份路径变量
set @path = 'D:\dbbak\' --赋值
declare @db_filename nvarchar(150) --定义文件名变量
set @db_filename = @path + 'db_'+@date+'.bak' --拼字符串，形成完整的备份文件路径
backup database DBNAME TO DISK=@db_filename --执行数据库备份操作，注意 DBNAME 为你实际要备份的数据库名

(3)还原数据库：

脚本分为两部分，一部分是批处理文件(dbrestore.bat)，一部分为 SQL 文件(dbrestore.sql)，两个文件要放在同一个目录下。
1、批处理文件内容
sqlcmd -S 192.168.56.36 -Usa -Ppassword -i .\dbrestore.sql -o .\dbrestore.log
说明：
sqlcmd -S <数据库 IP 地址> -U<数据库用户> -P<数据库密码> -i <需执行的 SQL 文件名称> -o <执行结果日志文件>
sqlcmd：为 Sqlsever2005 后自带的命令行工具，可以执行 SQL 文件。
2、SQL 文件内容如下
declare @date nvarchar(10) --定义日期变量
set @date = CONVERT(nvarchar(10),getdate(),112) --为日期变量赋当前日期，日期格式为 yyyymmdd 举例 20170830
declare @path nvarchar(250) -- 定义备份路径变量
set @path = 'D:\dbbak\' --赋值
declare @db_filename nvarchar(150) --定义文件名变量
set @db_filename = @path + 'db_'+@date+'.bak' --拼字符串，形成完整的备份文件路径
restore database DBNAME from DISK=@db_filename --执行数据库还原操作，注意 DBNAME 为你实际要备份的数据库名

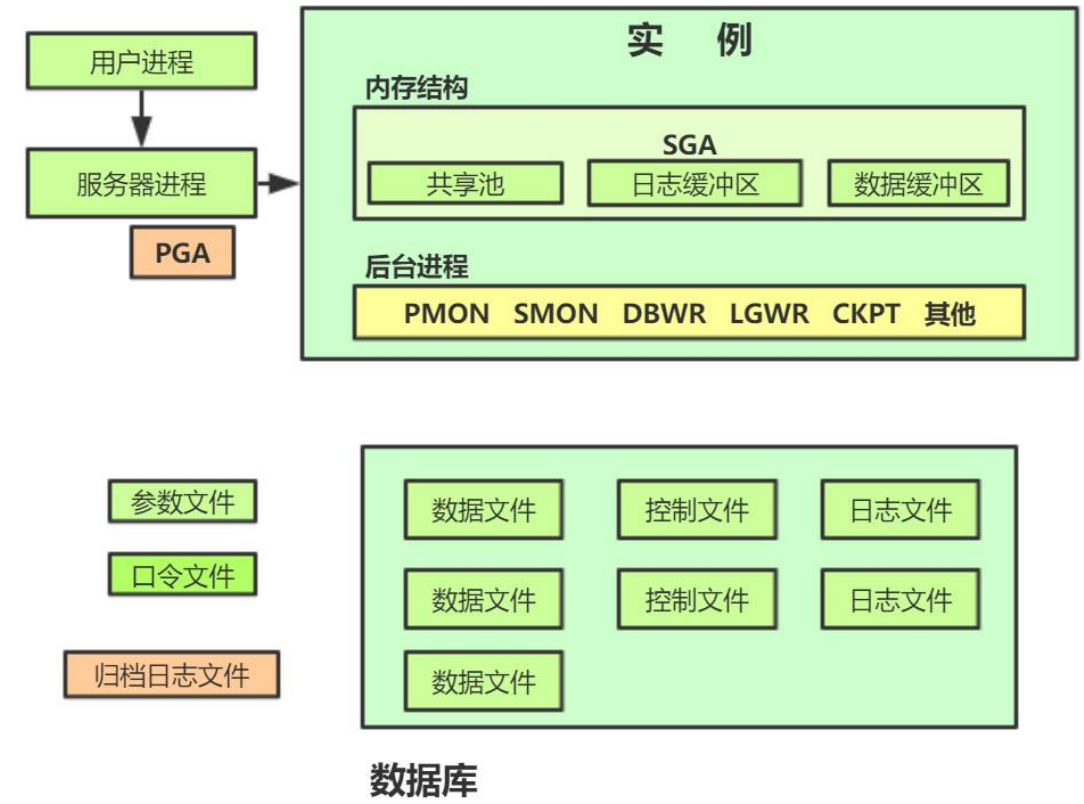
3.2.2.1.2.Oracle 备份技术

Oracle 备份主要通过封装 Oracle 提供的 OCCI 编程接口，并调用内置的 RMAN、EXP/IMP、EXPDP/IMPDP 等工具完成开发。

通过 OCCI 提供的接口，可以连接 Oracle 数据库并执行 SQL 语句，以获取数据库相关信息并进行操作，然后调用相关工具并搭配特定参数即可对数据库进行备份及恢复操作。

(1)Oracle 数据库基本结构概述

Oracle 数据库由实例、数据库组成:



(1)数据库：

由数据文件(包含 Oracle 数据、索引、表结构等数据)、控制文件(包括数据库名、数据文件位置等信息)、日志文件(数据操作 SQL 语句)、参数文件、口令文件、归档日志文件(归档模式下，服务器崩溃、硬盘损坏等情况下，通过日志恢复时使用)组成。

(2)实例：

由内存结构(memory strutct) 和后台进程(background processor)组成。

(3)内存结构组成：

a.PGA： Processor Global Area 程序全局区 ，每一个客户端接入到 oracle 服务器都有一个 PGA，用于保存该客户单的相关信息。

b.SGA:System Global Area 系统全局区，主要是给 oracle 实例使用，包括 shared pool 、 data buffer area , redo log buffer。

c.共享池(shared pool)：

包括 library cache 、data directory cache 组成，其中 library cache 主要保存最近的 sql 检查、编译、执行计划， 下次有同样语句过来的时候，可以重用这些，避免重复的检查编译执行计划。data directory cache 主要保存数据库数据表的字段定义、索引数据等，shared pool 的大小直接影响到数据库的性能。

d. data buffer area：

主要保存用户对数据的修改，查询操作。该内存区域的大小直接影响数据库的性能。

e. redo log buffer area：

主要保存最近用户对数据库的操作记录，该大小对数据库性能没有多大影响。

f. 关键后台进程：

●PMON：监控 PGA 的健康情况，释放已经死去的 PGA，回收资源，管理 PGA 的生命周期。

●SMON：监控 SGA 的健康情况，收集 SGA 碎片内存，监控实例健康情况。

●DBWR：维护 data buffer area 和物理表数据的一致性。

●LGWR：维护 redo log buffer area 内存数据和日志文件的一致性。

●CKPT：设置检查点，在 oracle 实例出现问题的时候，可以恢复到实例失败前的情况。

(2)Oracle 备份分类

Oracle 备份一般分为两种备份方式：

a. 物理备份

将 Oracle 物理文件(如数据文件、控制文件、归档日志文件等)转储到指定存储介质（通常为磁盘或磁带）的过程。当数据库发生故障时，便可利用这些文件进行还原；物理备份又分为冷备份和热备份，其中冷备份需要在数据库正常关闭后进行备份操作，而热备份可在数据库运行时进行备份操作，但需要数据库处于归档模式下。

b. 逻辑备份

通过 SQL 语言读取数据库一系列日志并抽取指定数据写入二进制文件的过程。这种备份不需要数据库运行在归档模式下，主要通过导入导出工具（EXP/IMP、EXPDP/IMPDP）进行操作，备份相对简单。是对物理备份的补充，多用于数据迁移。

(3)Oracle 备份方式比较

	逻辑备份	物理备份	
		归档模式	非归档模式

优点	1. 可对行对象进行备份。 2. 可跨平台进行数据备份和迁移。 3. 在数据库工作时也可进行备份。	1. 可在表空间或数据文件级备份，备份时间短。 2. 备份时数据库仍可使用。 3. 可达到秒级恢复（恢复到某一时间点上）。 4. 可对几乎所有数据库实体作恢复。 5. 恢复较快，在大多数情况下可在数据库仍工作时恢复。	1. 备份快速（只需拷文件） 2. 轻易归档（简单拷贝即可） 3. 轻易恢复到某个时间点上（只需将文件再拷贝回去） 4. 能与归档方法相结合，做数据库“最佳状态”的恢复。 5. 低度维护，高度安全。
缺点	1. 仅为逻辑备份，只能恢复逻辑错误。在数据库物理介质损坏的情况下，无法进行恢复 2. 只能恢复到备份点时刻。	1. 不能出错，否则后果严重。 2. 若热备份不成功，所得结果不可用于时间点的恢复。 3. 难于维护，要特别仔细小心，不允许“以失败而告终”。 4. 需要一定的空间存储归档日志。	1. 单独使用时，只能提供到“某一时间点上”的恢复。 2. 在备份的全过程中，数据库必须是关闭状态。 3. 若磁盘空间有限，只能拷贝到磁带等其他外部存储设备上，速度会很慢。 4. 不能进行表或用户恢复。
适用场景	通常用于较规律的日常备份	需要较高精确度时，如对表空间和数据文件进行备份	数据库可以暂时关闭

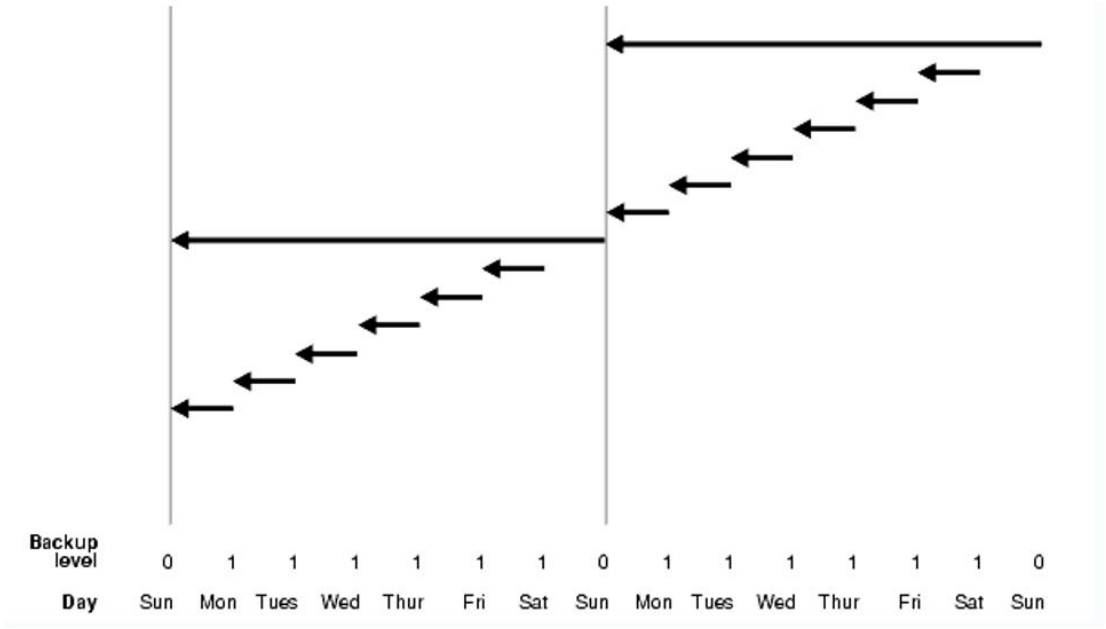
(4)对 Oracle 的三种备份模式

a.完全备份

一个或多个数据文件的一个完整副本，包含从备份开始处所有的数据块，对应 RMAN 的 0 级增量备份(backup incremental level 0 database；)，是所有备份的基础。

b.增量备份

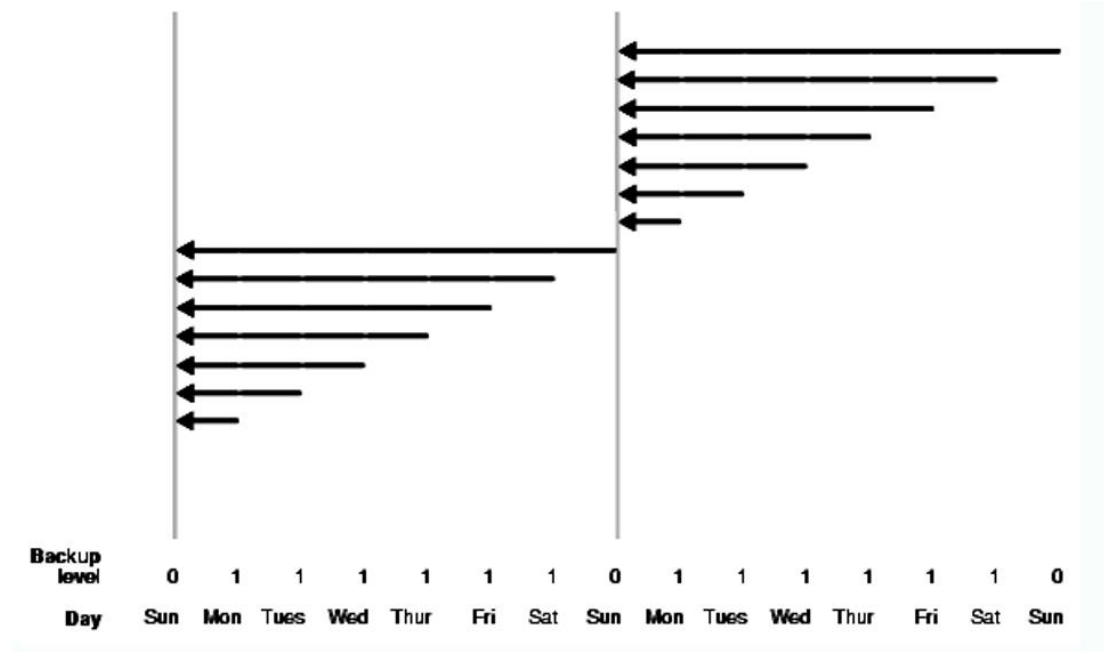
包含从最近一次备份以来被修改或添加的数据块，对应 RMAN 的 1 级差异增量备份(backup incremental level 1 database;)。



差异增量图示

c.差异备份

备份上一次完全备份以来所有变化的块，对应 RMAN 的 1 级累积增量备份(backup incremental level 1 cumulative database;)。



累积增量图示

(5)Oracle 备份流程概述

1.备份客户端程序通过 OCCI 与 Oracle 实例建立连接，获取实例状态与信息；

2.检查备份先决条件是否满足，如是否启用归档；

3.解析服务端发送过来的消息结构，获取备份目标并判断备份模式，构造对应的 RMAN 备份脚本，包含对控制文件、数据文件以及归档日志的备份；

4.通过 RMAN 连接对应实例，执行备份脚本，此时会创建快照控制文件，用以保

证数据一致性；
5.备份完成后校验生成的备份集有效性；
6.依次处理所有实例；
7.所有实例备份完毕后，清除备份产生的临时文件，断开与所有数据库实例的连接。

(6)Oracle 恢复流程概述

备份客户端程序通过 OCCI 与恢复目标实例建立连接；
1.解析服务端发送过来的消息结构，读取需要恢复的备份点及其依赖信息，构造对应的恢复脚本，包含对控制文件、数据文件和归档日志的还原；
2.通过 RMAN 连接恢复目标实例，执行恢复脚本；如果是实例级恢复，此时会停止数据库实例并将控制文件、数据文件以及归档日志还原，并通过日志重做还原数据库至备份点时刻的状态；
3.确认数据库实例已恢复至备份点时刻状态后，再将实例启动至运行状态；
4.依次处理其他恢复目标；
5.恢复操作完成后，清理恢复产生的临时文件，断开与所有数据库实例的连接。

3.2.2.1.3.MySQL 备份技术

MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，目前属于 Oracle 旗下公司。MySQL 最流行的关系型数据库管理系统，MySQL 是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。MySQL 所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策（本词条“授权政策”），它分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。由于其社区版的性能卓越，搭配 PHP 和 Apache 可组成良好的开发环境。

根据需求，要实现对承载业务系统数据的数据库进行备份，当数据库损坏后，对其进行恢复和还原，或新建恢复到其他数据库服务器，同时需要考虑可以容忍丢失多长时间的数据，恢复数据要在多长时间内完，恢复的时候是否需要持续提供服务，恢复的对象，是整个库，多个表，还是单个库，单个表等相关问题。

同时我们的备份需要保障在备份发生的同时，数据库业务不受到影响和干扰，即可以实现对数据库的热备功能。

基于以上要求，对于 MySQL 采用逻辑备份的思路，基于 mysqldump 工具实现完全备份的相关流程，基于 mysqlbinlog 完成增量、差异备份的相关流程。

(1)完全备份之 MysqlDump：

参考命令：

a)mysqldump - uuser_name -ppassword --single-transaction --master-data=2 database_name table_name > backup_file_name

注：--master-data=1 | 2，当该值为 1 或 2 时，会将当前 binlog 文

件，及文件偏移位置记录到导出文件中，为 1，不注释该记录，为 2 则要注释该记录。

--single-transaction，可实现存储引擎 innodb 的热备功能，当存储引擎为 MyISAM 时使用—lock-all-tables 或—lock-tables 命令。

--database-name 数据库名称

--table-name 表名称

--master-data 需在启用 log-bin 以后才有效

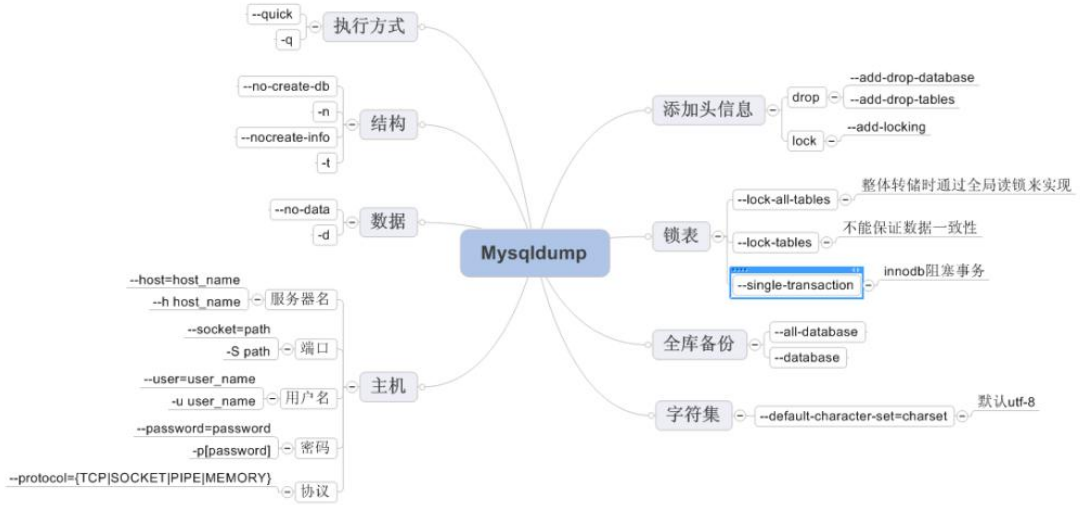
b)mysqldump - uuser_name -ppassword --single-transaction --master-data=2 --databases db1 db2 > backup_file_name

注：该命令为指定的一系列的数据库进行备份，如 db1，db2。

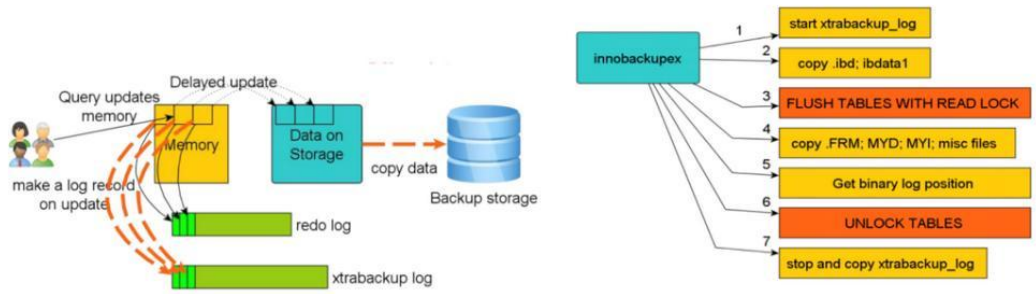
c)mysqldump - uuser_name -ppassword --single-transaction --master-data=2 --all-databases > backup_file_name

注：该命令为所有的数据库进行备份。

命令参数图：



流程图：



(2)增量备份：

MySQL 增量主要基于二进制日志及 mysqlbinlog 命令。

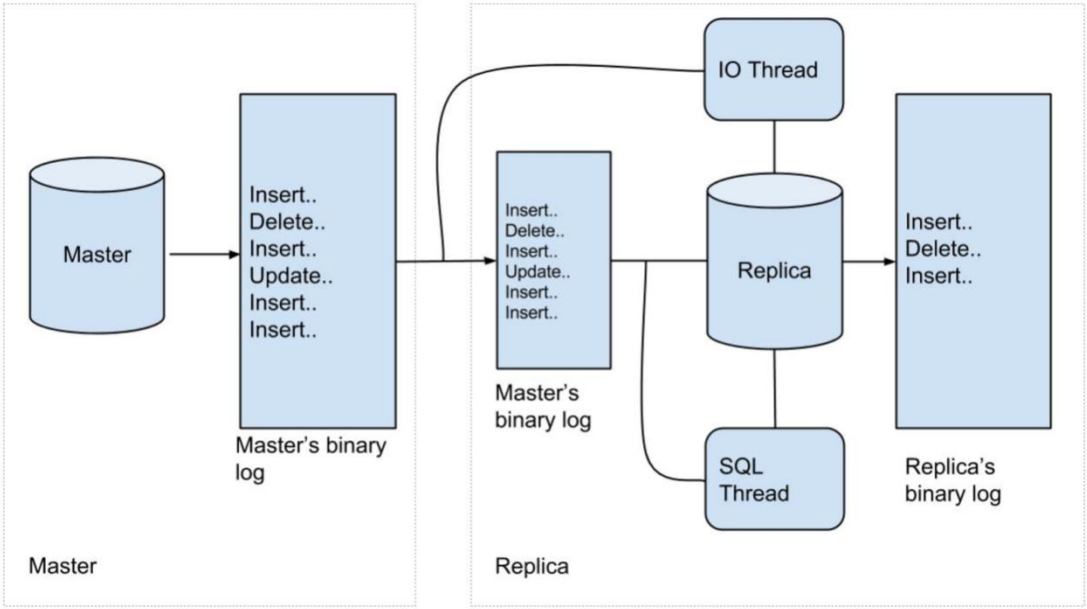
MySQL 二进制日志(Binlog)

binlog 是记录所有数据库表结构变更（例如 CREATE、ALTER TABLE…）以及表数据修改（INSERT、UPDATE、DELETE…）的二进制日志。

binlog 不会记录 SELECT 和 SHOW 这类操作，因为这类操作对数据本身并没有修改，但可以通过查询通用日志来查看 MySQL 执行过的所有语句。

二进制日志包括两类文件：二进制日志索引文件（文件名后缀为.index）用于记录所有的二进制文件，二进制日志文件（文件名后缀为.00000*）记录数据库所有的 DDL 和 DML(除了数据查询语句)语句事件。

MySQL 的主库（Master）对数据库的任何变化（创建表，更新数据库，对行数据进行增删改），都以二进制文件的方式记录与主库的 Binary Log（即 binlog）日志文件中。从库的 IO Thread 异步地同步 Binlog 文件并写入到本地的 Replay 文件。SQL Thread 再抽取 Replay 文件中的 SQL 语句在从库进行执行，实现数据更新。 需要注意的是，MySQL Binlog 支持多种数据更新格式 – 包括 Row，Statement，或者 mix（Row 和 Statement 的混合）。我们建议使用 Row 这种 Binlog 格式（MySQL5.7 之后的默认支持版本），可以更方便更加实时的反映行级别的数据变化。



基于 binlog 的主从复制架构

而我们的备份目标端模拟为 MySQL Slave，抽取出 Binlog 的日志文件，并把数据变化进行处理。

mysqlbinlog 参考命令：

1) show master status
   注：显示当前二进制日志的名称及位置
2) mysqlbinlog--start-position=n --stop-position=n /mydata/data/mysql-bin.000013 > /backup/hellodb
   注：备份二进制日志，--start-position 为要导出二进制日志的开始位置，--stop-position 为要导出二进制日志的结束位置
3) mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
   注：恢复二进制日志

4) show binary logs
注：显示二进制日志列表
5) flush logs
注：刷新二进制日志
6) reset master
注：删除所有二进制日志
7) SHOW BINLOG EVENTS[IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
注：查看二进制信息

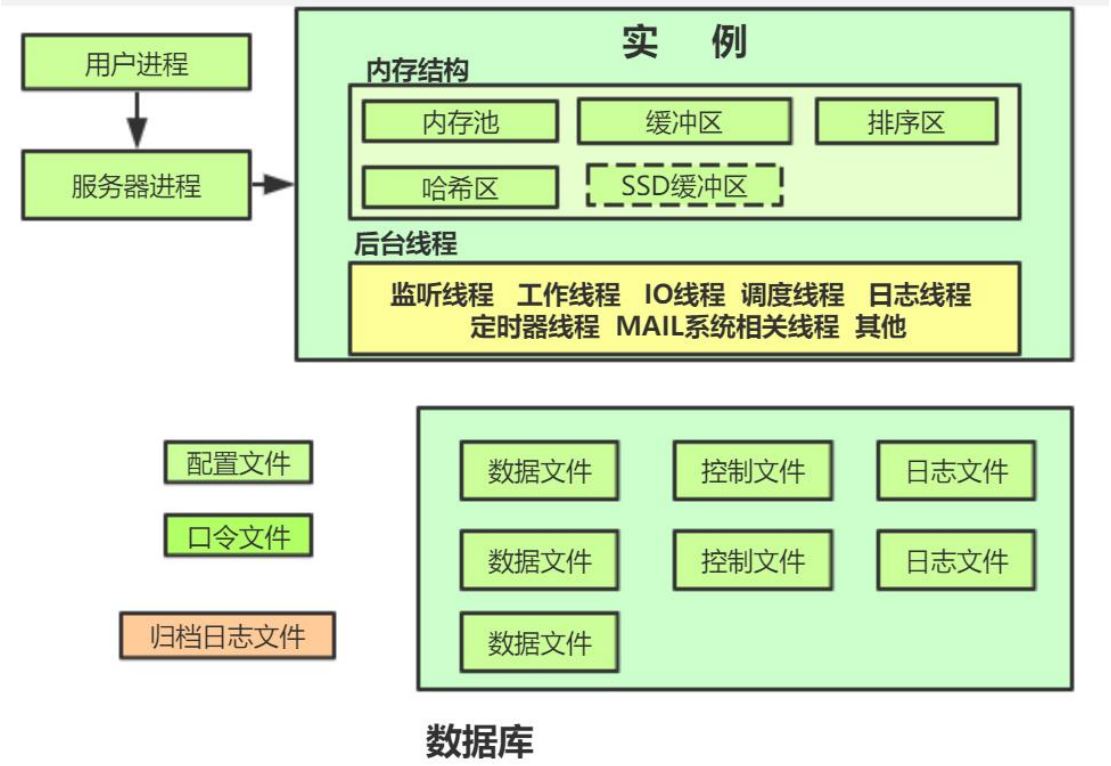
3.2.2.1.4.达梦备份技术

达梦数据库备份技术主要通过封装达梦提供的 DPI 编程接口，并调用内置的 DMRMAN、DEXP/DIMP 工具完成开发。

通过 DPI 提供的接口，可以连接达梦数据库并执行 SQL 语句，获取数据库相关信息并进行操作；执行特定 SQL 语句或调用相关工具并配置特定参数即可对数据库进行备份及恢复操作。

(1)达梦数据库基本结构概述

达梦数据库由实例、数据库组成:



a.数据库：

由数据文件(存储真实数据)、控制文件(记录了数据库必要的初始信息)、重做日志文件(存储数据库的事务日志)、逻辑日志文件(配置复制功能时启用，存

储着复制源端的各种逻辑操作)、归档日志文件(重做日志文件的连续备份)、配置文件(设置功能选项的一些文本文件的集合)、口令文件等组成。

b.实例：

由内存结构和后台线程组成。

内存池：DM Server 的内存池包括共享内存池和其他一些运行时内存池。

1.共享内存池

共享内存池是 DM Server 在启动时从操作系统申请的一大片内存。采用共享内存池则可一次向操作系统申请一片较大内存，即为内存池，当系统在运行过程中需要申请内存时，可在共享内存池内进行申请，当用完该内存时，再释放掉，即归还给共享内存池。

2.运行时内存池

运行时内存池是从操作系统申请一片内存作为本功能模块的内存池来使用，如会话内存池、虚拟机内存池等。

c.缓冲区

1.数据缓冲区

数据缓冲区是 DM Server 在将数据页写入磁盘之前以及从磁盘上读取数据页之后，数据页所存储的地方。

2.日志缓冲区

日志缓冲区是用于存放重做日志的内存缓冲区。

3.字典缓冲区

字典缓冲区主要存储一些数据字典信息，如模式信息、表信息、列信息、触发器信息等。

4.SQL 缓冲区

SQL 缓冲区提供在执行 SQL 语句过程中所需要的内存，包括计划、SQL 语句和结果集缓存。

d.排序区

排序缓冲区提供数据排序所需要的内存空间。当用户执行 SQL 语句时，常常需要进行排序，所使用的内存就是排序缓冲区提供的。

e.哈希区

虚拟缓冲区。系统没有真正创建特定属于哈希缓冲区的内存，而是在进行哈希连接时，对排序的数据量进行了计算。

f.SSD 缓存区

DM Server 将 SSD 文件作为内存缓存与普通磁盘之间的缓冲层，称为“SSD 缓存”。

关键后台线程：

1.监听线程：在服务器端口上进行循环监听，一旦有来自客户的连接请求，监听线程被唤醒并生成一个会话申请任务，加入工作线程的任务队列，等待工作线程进行处理。

2.工作线程： DM 服务器的核心线程，它从任务队列中取出任务，并根据任务的类型进行相应的处理，负责所有实际的数据相关操作。

3.IO 线程： 处理数据页不在缓冲区中等情况所需的额外 IO 操作

4.调度线程： 用于接管系统中所有需要定时调度的任务。

5.其他线程

(2)达梦数据库备份分类

达梦数据库备份一般分为两种备份方式：

a.物理备份

将达梦数据库的物理文件(如数据文件、控制文件、归档日志文件等)转储到指定存储介质（通常为磁盘或磁带）的过程。当数据库发生故障时，便可利用这些文件进行还原；物理备份又分为冷备份和热备份，其中冷备份需要在数据库正常关闭后进行备份操作，而热备份可在数据库运行时进行备份操作，但需要数据库处于归档模式下。

b.逻辑备份

通过 SQL 语言读取数据库一系列日志并抽取指定数据写入二进制文件的过程。这种备份不需要数据库运行在归档模式下，主要通过导入导出工具（DEXP/DIMP）进行操作，备份相对简单。是对物理备份的补充，多用于数据迁移。

(3)达梦数据库备份方式比较

	逻辑备份	物理备份	
		归档模式	非归档模式
优点	1. 可对行对象进行备份。 2. 可跨平台进行数据备份和迁移。 3. 在数据库工作时也可进行备份。	1. 可在表空间或数据文件级备份，备份时间短。 2. 备份时数据库仍可使用。 3. 可达到秒级恢复（恢复到某一时间点上）。 4. 可对几乎所有数据库实体作恢复。 5. 恢复较快，在大多数情况下可在数据库仍工作时恢复。	1. 备份快速（只需拷文件） 2. 轻易归档（简单拷贝即可） 3. 轻易恢复到某个时间点上（只需将文件再拷贝回去） 4. 能与归档方法相结合，做数据库“最佳状态”的恢复。 5. 低度维护，高度安全。
缺点	1. 仅为逻辑备份，只能恢复逻辑错误。在数据库物理介质损坏的情况下，无法进行恢复 2. 只能恢复到备份点时刻。	1. 不能出错，否则后果严重。 2. 若热备份不成功，所得结果不可用于时间点的恢复。 3. 难于维护，要特别仔细小心，不允许“以失败而告终”。 4. 需要一定的空间存	1. 单独使用时，只能提供到“某一时间点上”的恢复。 2. 在备份的全过程中，数据库必须是关闭状态。 3. 若磁盘空间有限，只能拷贝到磁带等其他外部存储设备上，速度会

		储归档日志。	很慢。 4. 不能进行表或用户恢复。
适用场景	通常用于较规律的日常备份	需要较高精确度时，如对表空间和数据文件进行备份	数据库可以暂时关闭

(4)对达梦数据库的三种备份模式

a.完全备份

完全备份生成的备份集包含了指定库（或者表空间）的全部有效数据页。当数据规模比较大的情况下，生成的完全备份集通常会比较大，而且备份时间也会比较长，是所有备份的基础。

示例：

```
BACKUP DATABASE FULL BACKUPSET '/home/dmbak/db_full_bak_01';
```

b.增量备份

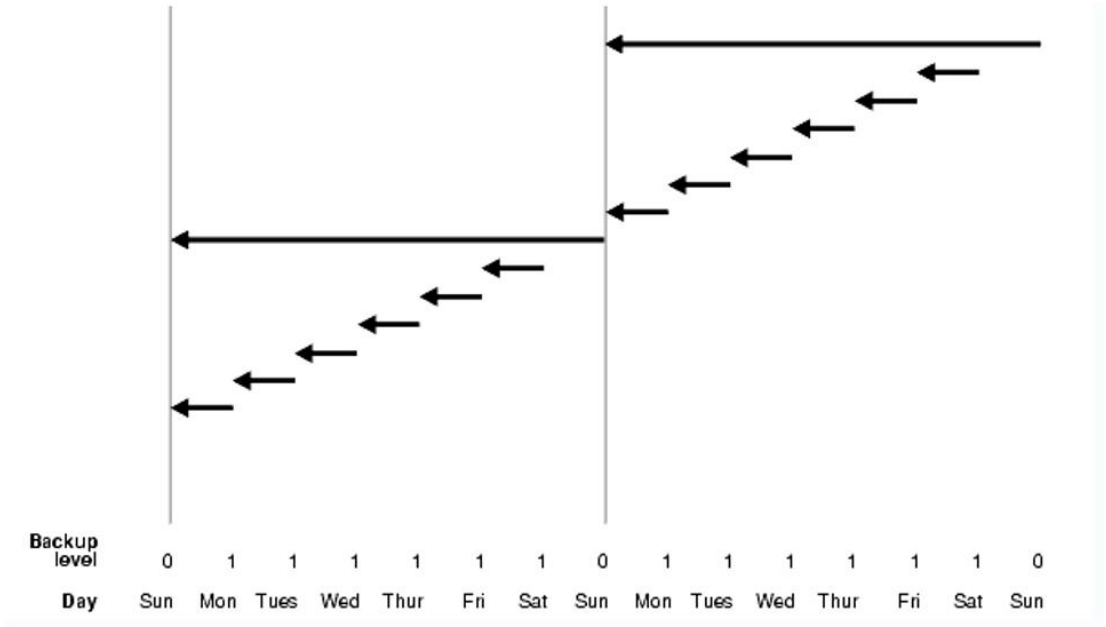
1.增量备份是在某个特定备份集基础上,收集数据库新修改的数据页进行备份，可以有效减少备份集的空间占用、提高备份速度。这个特定的、已经存在的备份集称为增量备份的基备份。

2.这里的增量备份特指达梦数据库中的差异增量备份,差异增量备份的基备份既可以是一个完全备份集，也可以是一个增量备份集。

3.利用增量备份进行还原操作时，要求其基备份必须是完整的；如果差异增量备份的基备份本身也是一个增量备份，那么同样要求其基备份是完整的；任何一个增量备份，最终都是以一个完全备份作为其基备份。因此，完全备份是增量备份的基础。

示例：

```
BACKUP DATABASE INCREMENT WITH BACKUPDIR '/home/dmbak' BACKUPSET
'/home/dmbak/db_increment_bak_02';
```



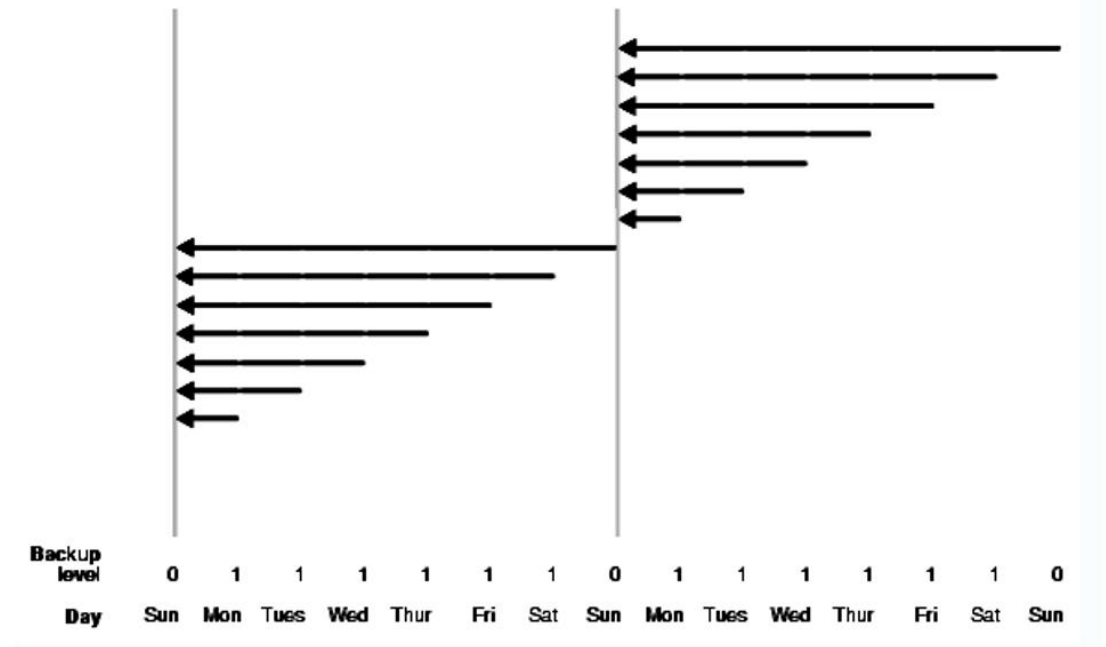
差异增量图示

注：0=FULL，1= INCREMENT

c.差异备份

备份上一次完全备份以来所有变化的块，对应达梦数据库中的累积增量备份，累积增量备份的基备份只能是完全备份集，而不能是增量备份集。

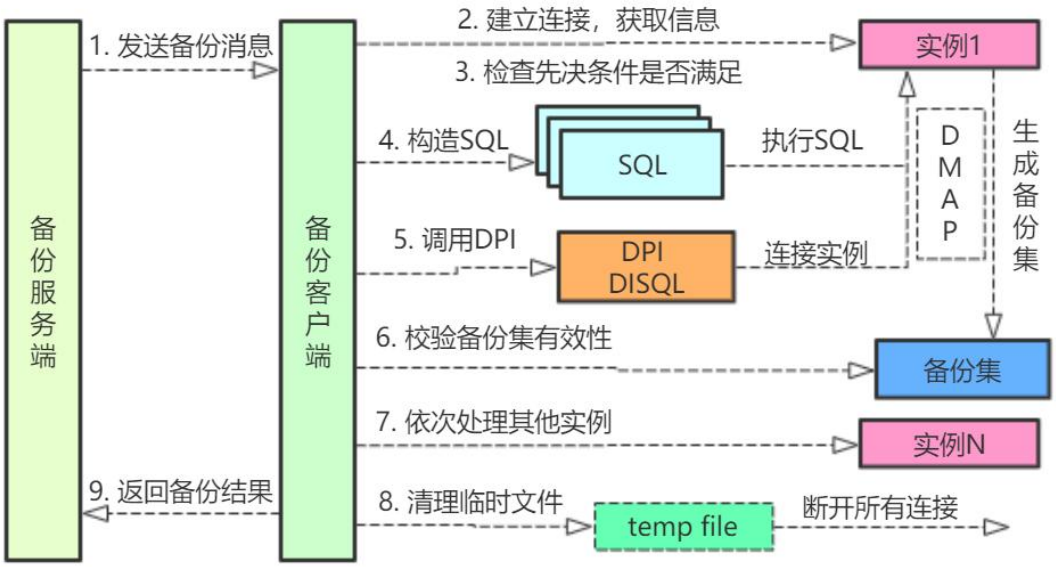
示例：BACKUP DATABASE INCREMENT CUMULATIVE WITH BACKUPDIR '/home/dmbak' BACKUPSET '/home/dmbak/db_increment_bak_03';



累积增量图示

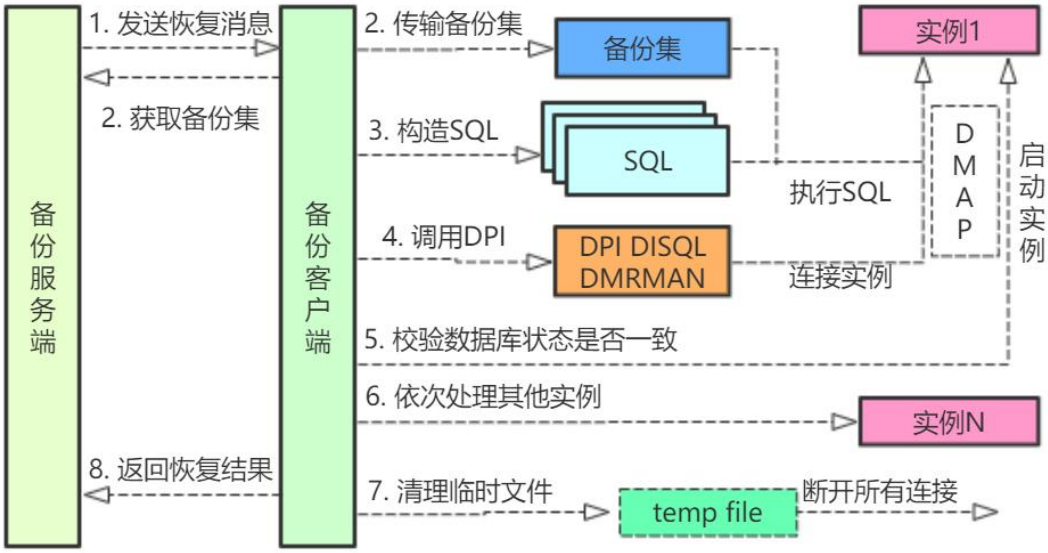
注：0=FULL，1= INCREMENT CUMULATIVE

(5)达梦数据库备份流程概述



1.备份客户端程序解析服务端发送过来的消息结构，获取备份目标并判断备份模式；
2.备份客户端程序通过 DPI 与 DM Server 建立连接，获取实例状态与信息；
3.检查备份先决条件是否满足，如是否启用归档，是否启用 DMAP 服务；
4.构造对应的 SQL 备份语句，对控制文件、数据文件以及归档日志备份；
5.通过 DPI 接口或 DISQL 工具连接对应实例，执行备份 SQL，此时会创建快照控制文件，用以保证数据一致性；
6.备份完成后校验生成的备份集有效性；
7.依次处理所有实例；
8.所有实例备份完毕后，清除临时文件，断开与所有数据库实例的连接；
9.向服务端返回备份结果。

(6)达梦数据库恢复流程概述



1.备份客户端程序解析服务端发送过来的消息结构，读取需要恢复的备份点及其依赖信息；

2.向服务端获取备份集信息，将存储介质中对应备份点的数据传输到客户端；
3.构造对应的 SQL 恢复脚本，包含对控制文件、数据文件和归档日志的还原；
4.通过 DPI 接口、DISQL 或 DMRAM 工具连接恢复目标实例，执行恢复 SQL 脚本；如果是实例级恢复，此时会停止数据库实例并将控制文件、数据文件以及归档日志还原，并通过日志重做还原数据库至备份点时刻的状态；
5.确认数据库实例已恢复至备份点时刻状态后，再将实例启动至运行状态；
6.依次处理其他恢复目标；
7.恢复操作完成后，清理恢复产生的临时文件，断开与所有数据库实例的连接；
8.向服务端返回恢复结果。

3.3.文件备份

3.3.1.功能简介

3.3.1.1.定时备份

(1)支持对主流操作系统上的文件备份。
(2)支持 Windows 全平台，支持 CentOS、RedHat 等 Linux 平台，AIX 等 UNIX 平台。
(3)支持完全备份、差异备份、增量备份三种备份模式。
(4)支持对海量文件的备份。
(5)支持对 NAS 设备，NFS、CIFS 等网络路径上的文件备份。

3.3.2.技术实现

3.3.2.1.定时备份

文件备份主要将用户选择的目标文件，以及目标文件产生的差异、增量的变化结果通过数据流传输协议传输、存储到备份仓库中。其中主要关注的技术点包括：

1.文件增量变化发现的机制：

Windows 系列：

Windows 系列的主要实现原理基于文件系统的日志记录 USN。

USN 是 Update Service Number Journal or Change Journal 的英文缩写，直译为“更新序列号”，是对 NTFS 卷里所修改过的信息进行相关记录的功能。当年微软发布 Windows 2000 时，建立 NTFS 5.0 的同时，加入了一些新功能和改进了旧版本的文件系统，为它请来了一位可靠的秘书，它可以在分区中设置监视更改的文件和目录的数量，记录下监视对象修改时间和修改内容。没错，它就是 USN 日志。当这个功能启用时，对于每一个 NTFS 卷，当发生有关添加、删除和修改文件的信息时，NTFS 都使用 USN 日志记录下来。事实上 Change Journal 是

标卷上一个特殊的文件，系统将其隐藏,所以用资源管理器或者 CMD Shell 都看不到，当文件系统中的文件或者目录发生改变时，就会向日志中追加记录。记录一般包括：文件名，变化时间，变化类型,而实际的数据不会记录，这样也可以保持记录文件足够小。

最开始的时候，日志文件时一个磁盘标卷上的空文件,随着改变的发生，记录不断被追写进日志。每条日志有个 64-bit 标识，即 USN（Update Sequence Number)，这个 USN 是自增的，所以你可以通过比较 USN 来,找到事件发生的顺序（号码越小，事件越早)，但不一定连续，有可能第一个 USN 是 0，而第二个是 128。

微软最开始构建 Change Journal 时，称其为 USN Journal，所以 winioctl.h 头文里的结构定义都是这个命名，写程序的时候也将大量使用这个名词,所以下面不区分,Change Journal=USN Journal。

由于总是向文件末端添加记录，所以采用文件偏移的形式来存储 USN,这样查询时只需要计算即可定位。但记录中的文件名是变长的，所以每条 USN 大小也不一定相同。考虑到性能问题，系统会将记录以 4KB（可以参看 winioctl.h 中的 USN_PAGE_SIZE 宏）为块大小存放,每块通常会包含三四十条记录。操作系统不允许单条记录横跨两个块页，所以有时候会发生 USN 为空，用来填充块间隙。

在 NTFS 标卷上,文件和目录信息存储于 Master File Table(MFT)中，其中的记录都描述了文件或目录名，位置，大小,属性等。NTFS 5.0 中，每个 MFT 记录项都保存了该文件或者目录最后的 USN 记录。当 Change Journal 记录时，文件系统更新被更改的 MFT 中最后的 USN 值。

如果日志文件过大（大于定义的 MaximumSize 参数)，系统将会清理掉文件开始部位较早的数据，通常截断开始数据需要大量的 I/O 操作，文件末端必须要被拷贝到新位置,这是一个耗时的过程。幸运的是，NTFS 5.0 支持稀疏文件，这种机制允许删除文件中不需要的部分,而保留其余数据的逻辑偏移。所以 Change Journal 就是一个稀疏文件,允许清除早期记录,而不会损失太多性能，也不影响原先的文件偏移访问。更多关于稀疏文件信息可以参考 A File System for the 21st Century: Previewing the Windows NT 5.0 File System。

标卷上的 Change Journal 功能可以关闭,这样系统就不会记录变化信息，默认情况下，NTFS 标卷上的 Change Journal 功能是关闭的,必须明确的开启才能使用,开启和关闭可以由任意程序,任意时间完成。问题来了,如果两个程序操作时发生冲突怎么办?当一个程序禁用标卷的 Change Journal，系统会清理所有先前的记录,以防止其他程序读取不可靠的数据。总的来说,Change Journal 启用时会创建日志文件，禁用时会删除日志文件。

每一个 Change Journal 会被分配一个唯一的 64-bit 标识(与 USN 标识不同),系统将会在禁用/启用之后改变这个标识,这样程序可以通过读取这个标识,来确定读取信息的可靠性。这个标识在重启后也不会变化，换句话说，如果标识不变，Change Journal 会记录开机后所有文件的变化。

Linux 系列：

Linux 系列的主要实现原理基于 Inotify。

Inotify 是一个 Linux 内核特性,它监控文件系统,并且及时向专门的应用程序发出相关的事件警告，比如删除、读、写和卸载操作等。您还可以跟踪活动的源头和目标等细节。

使用 inotify 很简单：创建一个文件描述符，附加一个或多个监视器（一个

监视器 是一个路径和一组事件),然后使用 read() 方法从描述符获取事件信息。read() 并不会用光整个周期，它在事件发生之前是被阻塞的。
更好的是，因为 inotify 通过传统的文件描述符工作，您可以利用传统的 select() 系统调用来被动地监控监视器和许多其他输入源。两种方法 — 阻塞文件描述符和使用 select()— 都避免了繁忙轮询。

Inotify 提供 3 个系统调用，它们可以构建各种各样的文件系统监控器：inotify_init() 在内核中创建 inotify 子系统的一个实例,成功的话将返回一个文件描述符，失败则返回 -1。就像其他系统调用一样，如果 inotify_init() 失败，请检查 errno 以获得诊断信息。顾名思义，inotify_add_watch() 用于添加监视器。每个监视器必须提供一个路径名和相关事件的列表（每个事件由一个常量指定，比如 IN_MODIFY)。要监控多个事件，只需在事件之间使用逻辑操作符或 — C 语言中的管道线（|）操作符。如果 inotify_add_watch() 成功，该调用会为已注册的监视器返回一个惟一的标识符；否则，返回 -1。使用这个标识符更改或删除相关的监视器。

inotify_rm_watch() 删除一个监视器。

此外,还需要 read() 和 close() 系统调用。如果描述符由 inotify_init() 生成，则调用 read() 等待警告。假设有一个典型的文件描述符，应用程序将阻塞对事件的接收，这些事件在流中表现为数据。文件描述符上的由 inotify_init() 生成的通用 close() 删除所有活动监视器,并释放与 inotify 实例相关联的所有内存（这里也用到典型的引用计数警告。与实例相关联的所有文件描述符必须在监视器和 inotify 消耗的内存被释放之前关闭。

2.文件传输的机制：

文件传输基于 socket 封装的数据流传输协议,将每个需要传输的目标文件进行可变长大小的分片、编号、校验，保证文件传输的可靠性与一致性。其主要流程与算法如下：

(1) α 主机告诉 β 主机文件 A 待传输。

(2) β 主机收到信息后，将文件 B 划分为一系列大小固定的数据块(建议大小在 500-1000 字节之间),并以 chunk 号码对数据块进行编号,同时还会记录数据块的起始偏移地址以及数据块长度。显然最后一个数据块的大小可能更小。

(3) β 主机对文件 B 的每个数据块根据其内容都计算两个校验码：32 位的弱滚动校验码(rolling checksum)和 128 位的 MD4 强校验码(现在版本的 rsync 使用的已经是 128 位的 MD5 强校验码)。并将文件 B 计算出的所有 rolling checksum 和强校验码跟随在对应数据块 chunk[N]后形成校验码集合，然后发送给主机 α 。

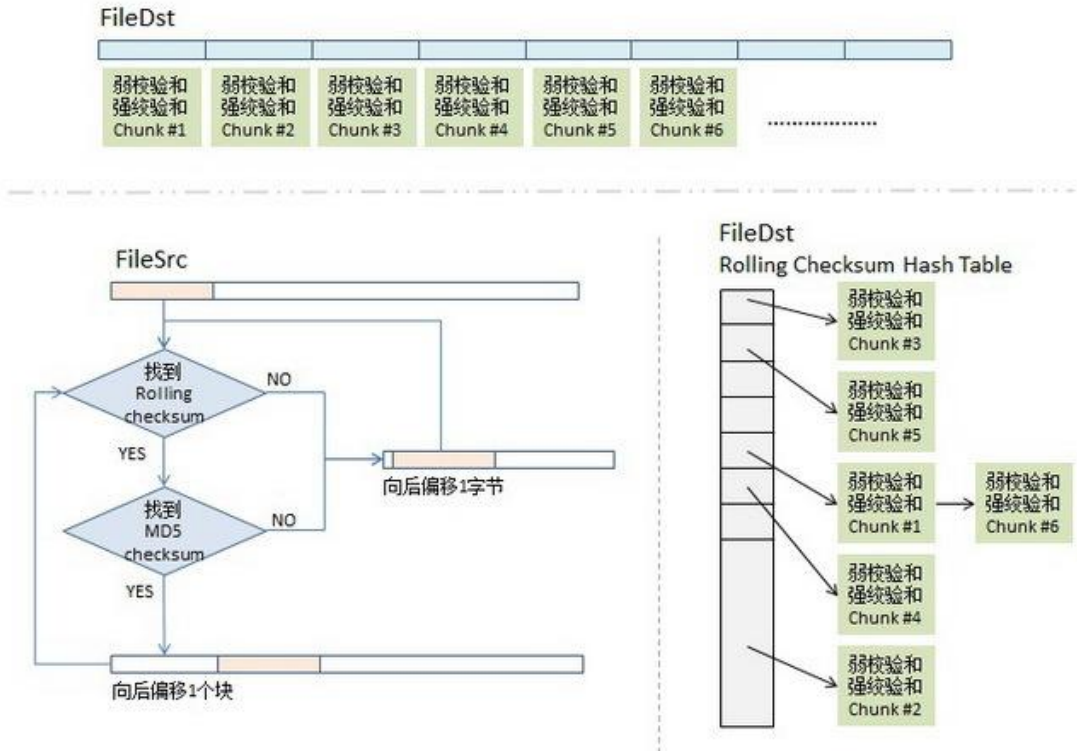
(4)当 α 主机接收到文件 B 的校验码集合后, α 主机将对此校验码集合中的每个 rolling checksum 计算 16 位长度的 hash 值,并将每 216 个 hash 值按照 hash 顺序放入一个 hash table 中,hash 表中的每一个 hash 条目都指向校验码集合中它所对应的 rolling checksum 的 chunk 号码,然后对校验码集合根据 hash 值进行排序,这样排序后的校验码集合中的顺序就能和 hash 表中的顺序对应起来。

(5)随后主机 α 将对文件 A 进行处理。处理的过程是从第 1 个字节开始取相同大小的数据块,并计算它的校验码和校验码集合中的校验码进行匹配。如果能匹配上校验码集合中的某个数据块条目，则表示该数据块和文件 B 中数据块相同,它不需要传输,于是主机 α 直接跳转到该数据块的结尾偏移地址,从此偏移处继续取数据块进行匹配。如果不能匹配校验码集合中的数据块条目,则表示该数据块是非匹配数据块,它需要传输给主机 β ,于是主机 α 将跳转到下一个字节,从此字

节处继续取数据块进行匹配。注意:匹配成功时跳过的是整个匹配数据块,匹配不成功时跳过的仅是一个字节。

(6)当α主机发现是匹配数据块时,将只发送这个匹配块的附加信息给β主机。同时,如果两个匹配数据块之间有非匹配数据，则还会发送这些非匹配数据。当β主机陆陆续续收到这些数据后，会创建一个临时文件,并通过这些数据重组这个临时文件，使其内容和A文件相同。临时文件重组完成后,修改该临时文件的属性信息(如权限、所有者、mtime等),然后重命名该临时文件替换掉B文件,这样B文件就和A文件保持了同步。

主要的比对算法：



3.文件断网续传的机制：

备份在检测到网络故障以后,会进入到断网续传的检测流程,周期性（间隔5秒）的检测网络是否恢复正常,如网络正常则继续数据传输,如网络异常且重连时间超过30分钟,则会退出断网续传的检测流程,并报告传输失败。

3.4.操作系统备份

3.4.1.功能简介

3.4.1.1.定时备份

支持对主流操作系统的系统备份。

（1）支持Windows全平台，支持CentOS、RedHat等Linux平台。

（2）保障系统备份数据的一致性。

（3）有效数据备份（即只备份分区上实际分配的数据区域）
（4）兼容 NTFS、FAT32、EXT2、EXT3、EXT4、XFS 等文件系统类型。
（5）兼容 MBR、GPT 分区表类型。
（6）兼容 BIOS、UEFI 引导模式。
（7）支持异构平台之间的恢复。
（8）支持 P2V。
（9）支持通过 PXE 网络引导，光盘，U 盘三种恢复引导模式。
（10）支持 SANBOOT，将备份的操作系统数据瞬时接管到运行到生产机器上。

3.4.2.技术实现

3.4.2.1.定时备份

操作系统备份主要涉及的技术包括卷快照技术，卷有效数据获取技术，数据传输技术，操作系统引导恢复技术，恢复的驱动兼容技术：

1.卷快照技术：

快照原理简介：

像照相机一样，机器快门一闪，很快就把刚刚的人像停留在了相纸上。存储系统中的数据“快照”与我们生活中所说的“照片”非常相似，所不同的是，照片的对象不是人，而是数据。如同照片留住了我们过去的摸样和岁月，快照把数据在某一时刻的映像也保留了下来。因此我们可以根据快照查找数据在过去某一时刻的映像，常常用来作为增强数据备份系统的一种技术，它可以很大的缩短 RTO 和 RPO 两个指标。

SNIA（存储网络行业协会）对快照（Snapshot）的定义是：关于指定数据集合的一个完全可用拷贝，该拷贝包括相应数据在某个时间点（拷贝开始的时间点）的映像。快照可以是其所表示的数据的一个副本，也可以是数据的一个复制品。而从具体的技术细节来讲，快照是指向保存在存储设备中的数据的引用标记或指针。

磁盘快照(Snapshot)是针对整个磁盘卷册进行快速的档案系统备份，与其它备份方式最主要的不同点在于「速度」。进行磁盘快照时，并不牵涉到任何档案复制动作。就算数据量再大，一般来说，通常可以在一秒之内完成备份动作。

磁盘快照的基本概念与磁带备份等机制有非常大的不同。在建立磁盘快照时，并不需要复制数据本身，它所作的只是通知 LX Series NAS 服务器将目前有数据的磁盘区块全部保留起来，不被覆写。这个通知动作只需花费极短的时间。接下来的档案修改或任何新增、删除动作，均不会覆写原本数据所在的磁盘区块，而是将修改部分写入其它可用的磁盘区块中。所以可以说，数据复制，或者说数据备份，是在平常档案存取时就做好了，而且对效能影响极低。LX Series NAS 档案系统内部会建立一份数据结构，纪录磁盘快照备份及目前作用中档案系统所使用到的磁盘区块及指针，让使用者可以同时存取到主要档案系统及过去的磁盘快照版本。

快照技术的作用：主要是能够进行在线数据恢复，当存储设备发生应用故障或者文件损坏时可以进行及时数据恢复，将数据恢复成快照产生时间点的状态。快照的另一个作用是为存储用户提供了另外一个数据访问通道，当原数据进行在

线应用处理时,用户可以访问快照数据,还可以利用快照进行测试等工作。 因此，所有存储系统，不论高中低端，只要应用于在线系统，那么快照就成为一个不可或缺的功能。创建一个快照不同的设备需要不同的命令，但对于系统来说，基本都包括如下几个步骤：

1.首先发起创建指令;
2.在发起时间点，指令通知操作系统暂停应用程序和文件系统的操作;
3.刷新文件系统缓存，结束所有的读写事务;
4.创建快照点;
5.创建完成之后，释放文件系统和应用程序，系统恢复正常运行。

而在我们的卷级别备份产品中，主要使用了 COW（写时拷贝）类型的快照，COW 快照需要消耗一些存储空间--建立快照卷。当我们为一个数据卷创建一个快照之后，这些预留的空间用来存放被变化数据更新的旧数据。COW 快照在初始化的过程中仅仅创建用来描述源数据块位置的指针信息(元数据),而不是完整的将源数据块拷贝过来。因此初始化的过程几乎可以在瞬间完成，对系统的影响也很小。

COW 快照会跟踪数据卷的写操作和数据块变化。当某个数据块发生改变时，在将旧的数据覆盖之前，首先将该块的旧数据复制到预留的快照卷，该步骤仅在数据卷相应数据块位置发生第一次写操作请求时进行。这个处理过程确保快照出来的数据与发起快照的那个精确时间点保持完全一致。这个过程也描述了“copy on write”这个名字的含义。

如果我们需要访问某个时间点的快照数据,对没有改变过的块直接从数据卷读取;对已经改变并被复制的块则从快照空间读取。从快照被创建那一刻开始，每个快照都会跟踪记录描述块改变的元数据信息。

COW 快照的主要优势在于空间的高效利用，因为快照卷只需要保留发生过变化的数据块，与数据卷相比要小得多。但是我们也知道 COW 快照有个缺点，它会引起数据卷性能的下降，这是因为创建快照之后，对数据卷的写操作会增加一个等待的过程 --即旧数据块复制到快照卷的过程。另外一个关键问题是每个快照卷必须依赖一个完整的数据卷。

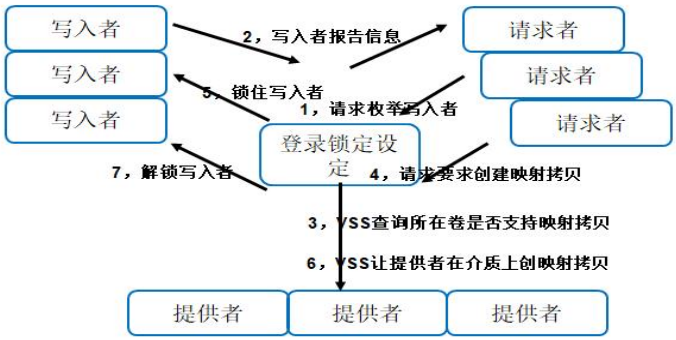
Windows 快照技术：

Windows 操作系统下主要使用了微软的 Windows NTFS 有 VSS 卷影拷贝服务 Volume Shadow Copy Services， Vista 称作 Shadow Copy。

VSS 整体框架包含了 VSS 核心模块、请求者(Requestor)、写入者(Writer)，以及提供者(Provider)。各个模块之间的关系如下图所示：

A) 请求者(Requestor)，其主要任务是初始化映射拷贝的创建;
B) 写入者(Writer)，其主要任务是保证数据的一致性;
C) 提供者(Provider)，其主要任务是创建映射拷贝;
D) 卷映射拷贝服务(VSS)核心模块，其主要任务是协调各个模块的协作运行，并提供创建卷映射拷贝的方法;

VSS 处理映射拷贝，需要协调各个模块来完成，从而保证创建出高保真的拷贝，进而实现数据的一致性，其主要流程如下：



VSS 处理流程图

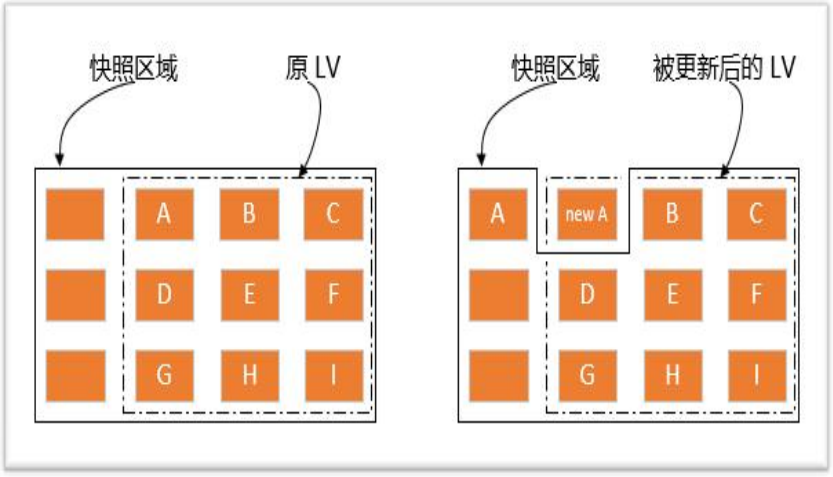
A) 第一步，请求者让 VSS 枚举所在卷上的写入者应用，并收集元数据(Metadata)。
B) 第二步，写入者可能通过 XML 文件来描述其组件(Components)，并定义其恢复(Restore)方法；其中，考虑到数据一致性，写入者需要一些相关处理，比如对于数据库应用来说，关闭所有打开的事务、回滚事务日志、以及将缓冲区中的数据写入等操作，直到所有数据准备好之后，通知 VSS 可以创建映射拷贝了。
C) 第三步，VSS 对于请求的卷，查询是否支持映射拷贝，并由那个提供者提供；因为在请求者管理应用中，会设置卷的映射拷贝属性以及策略等，所以需要进行查询和判断。
D) 第四步，请求者通知 VSS，要求在该卷上创建映射拷贝。
E) 第五步，VSS 锁住写入者应用，暂时不让写入新数据(在某些应用情况下，读操作请求是可以允许的)。
F) 第六步，VSS 让提供者在磁盘上创建当前状态的映射拷贝(创建映射拷贝的速度，和创建的方法以及提供者的实现相关)。
G) 第七步，创建映射拷贝完毕，VSS 解锁写入者应用；然后写入者就可以处理队列中的写请求，接着 VSS 会查询是否这些写请求在创建映射拷贝期间被保证在队列中，如果是，则说明数据是一致的，否则说明可能数据一致性有问题，并做相应处理。

Linux 快照技术：

LVM 提供了 LV 做快照的功能，也就是说可以给文件系统做一个备份，这也是设计 LVM 快照的主要目的。LVM 的快照功能采用写时复制技术(Copy-On-Write，COW)，这比传统的备份技术的效率要高很多。创建快照时不用停止服务，就可以对数据进行备份。

LVM 采用的写时复制，是指当 LVM 快照创建的时候，仅创建到实际数据的 inode 的硬链接(hark-link)而已。只要实际的数据没有改变，快照就只包含指向数据的 inode 的指针，而非数据本身。快照会跟踪原始卷中块的改变，一旦你更改了快照对应的文件或目录，这个时候原始卷上将要改变的数据会在改变之前拷贝到快照预留的空间。

其原理如下所示：



上图为最初创建的快照数据卷状况，LVM 会预留一个区域（比如左图的左侧三个 PE 区块）作为数据存放处。 此时快照数据卷内并没有任何数据，而快照数据卷与源数据卷共享所有的 PE 数据， 因此你会看到快照数据卷的内容与源数据卷中的内容是一模一样的。 等到系统运行一阵子后，假设 A 区域的数据被更新了(上面右图所示)，则更新前系统会将该区域的数据移动到快照数据卷中， 所以在右图的快照数据卷中被占用了一块 PE 成为 A，而其他 B 到 I 的区块则还是与源数据卷共享

由于快照区与原本的 LV 共享很多 PE 区块，因此快照区与被快照的 LV 必须要在同一个 VG 上头，下面两点非常重要：

✧VG 中需要预留存放快照本身的空间，不能全部被占满。

✧快照所在的 VG 必须与被备份的 LV 相同，否则创建快照会失败。

1.卷有效数据获取技术：

卷级别备份均采用有效数据备份，通过读取目标卷上的文件系统位图（bitmap）信息，可以遍历出文件系统已经分配的有效数据块，而我们备份程序只需要读取这些快，便能备份出一个有效的数据镜像。这里我们通过介绍 Linux 下文件系统的位图来对该项技术有更深入的认识。

2.位图的定义：

一种数据结构，代表了有限域中的稠集（dense set），每一个元素至少出现一次，没有其他的数据和元素相关联。在索引、数据压缩等方面有广泛应用。

而对于文件系统来说，位图就是按位(bit)来记录、索引某个对象状态的图表(map)，即用每一位（bit）来存放每一个对象的某种状态（例如分别用 0 和 1 表示的话，可以表示同一对象的两种状态）。

3.数据传输技术：

操作系统备份与恢复的传输主要采用了基于 tcp 的 socket 流传输，其主要流程如下:

（1）将需要备份的系统卷、数据卷的有效数据定位出来，理论上这将是一个个相对离散的块。

（2）如果两个块的位置是连续的，则将它们合并（即认为它们是一个块，只是块的长度会延长），判断的原则是 Block1_offset + Block1_len == Block2_offset。

（3）合并以后如果块的数据长度太长，则需要以最大长度为边界进行切割。

（4）在上面的工序完成以后，给这些块添加上数据头信息，信息包括数据块

的长度、偏移等。

（5）然后将完整的数据块再添加上数据包头，包头主要包含数据源，数据是否加密，包的序列等信息。这一步完成以后，数据块完成了打包，便丢入到收发流程。

（6）发送线程将打包以后的数据包通过 socket 协议发送到备份服务器，备份服务器收到以后需返回 ack 包，用来标识该数据包是否完整，有效，序列是否正确，接收队列是否拥塞，并根据接收到的情况置位 ack 的各种状态。

（7）客户端接收到 ack 以后，如果包需要重传，则再传输一次，如果备份服务端接收队列拥塞，则延迟再发。如果无其他异常，则继续发送下一个数据包，直到所有的包都成功发送。

4.操作系统引导恢复技术：

由于操作系统恢复是要对系统数据进行写入，所以必须要进入一个第三方的系统环境。Windows PE (WinPE) 是一个小型操作系统，用于安装、部署和修复 Windows 桌面版（家庭版、专业版、企业版和教育版）、Windows Server 和其他 Windows 操作系统。 通过 Windows PE，你可以：

（1）在安装 Windows 之前设置硬盘，或者对系统进行修改、擦除或覆盖。

（2）使用来自网络或本地驱动器的应用或脚本安装 Windows。

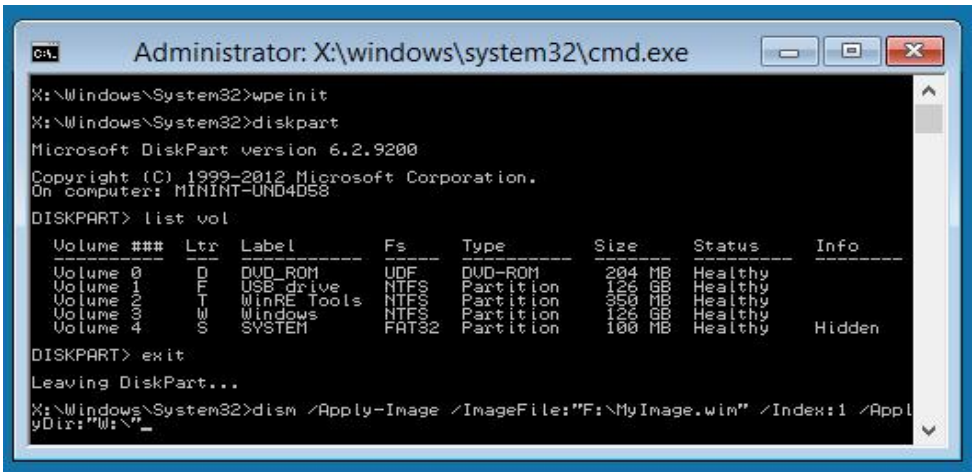
（3）捕获和应用 Windows 映像。

（4）在 Windows 操作系统未运行时，对其进行修改。

（5）设置自动恢复工具。

（6）从无法启动的设备中恢复数据。

（7）添加自己的自定义 shell 或 GUI 来使此类任务自动化。



Windows PE 是微软提供的一种迷你 Windows 系统的解决方案，只有不到 500MB，可以根据需要自己定制。Windows PE 运行 Windows 命令行环境，并支持以下 Windows 功能：

（1）批处理文件和脚本，包括对 Windows 脚本主机 (WSH) 和 ActiveX 数据对象 (ADO) 的支持，以及对 PowerShell 的可选支持。

（2）应用程序，包括 Win32 应用程序编程接口 (API)，以及对 HTML 应用程序 (HTA) 的可选支持。

（3）驱动程序，包括一组可运行网络、图形和大容量存储设备的通用驱动程序。

（4）映像捕获和服务，包括部署映像服务和管理 (DISM)。

（5）网络，包括通过 LAN 使用 TCP/IP 和 TCP/IP 上的 NetBIOS 连接到文件

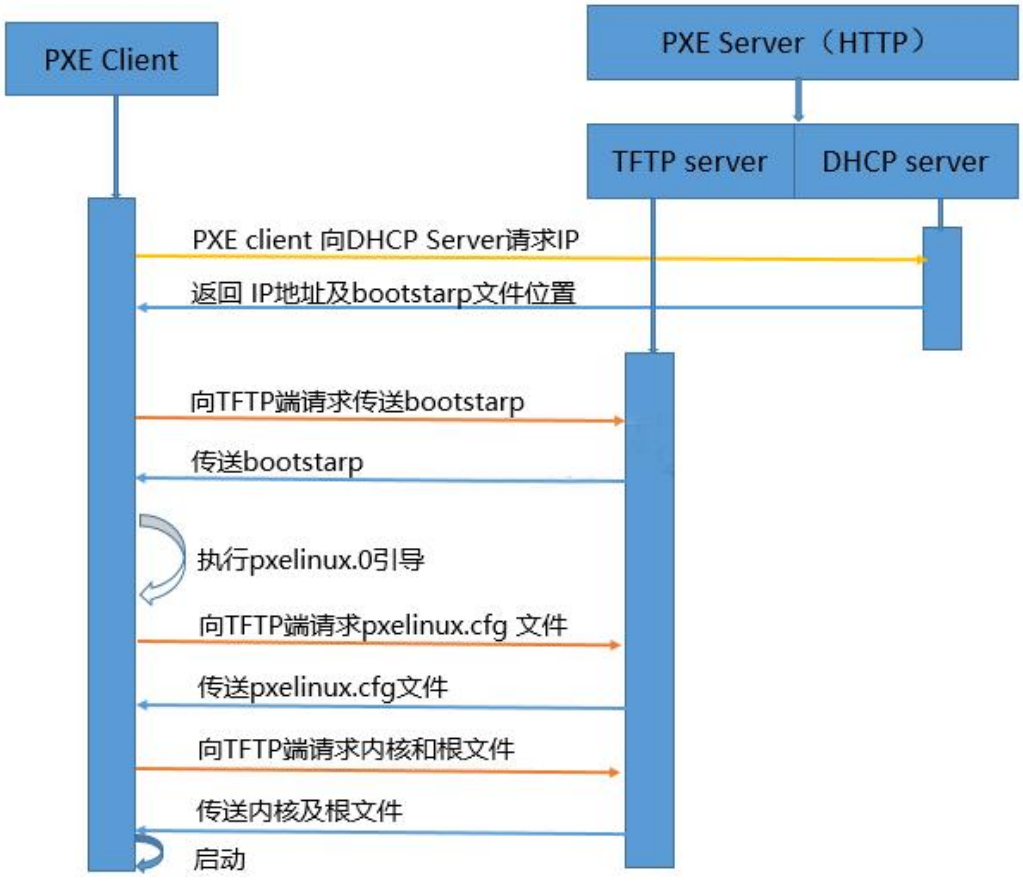
服务器。

（6）存储，包括 NTFS、DiskPart 和 BCDBoot，用于修复系统的引导问题。

（7）安全工具，包括对 BitLocker 和受信任的平台模块 (TPM)、安全启动以及其他工具的可选支持。

（8）Hyper-V，包括 VHD 文件、鼠标集成、大容量存储以及允许在虚拟机监控程序中运行 Windows PE 的网络驱动程序。

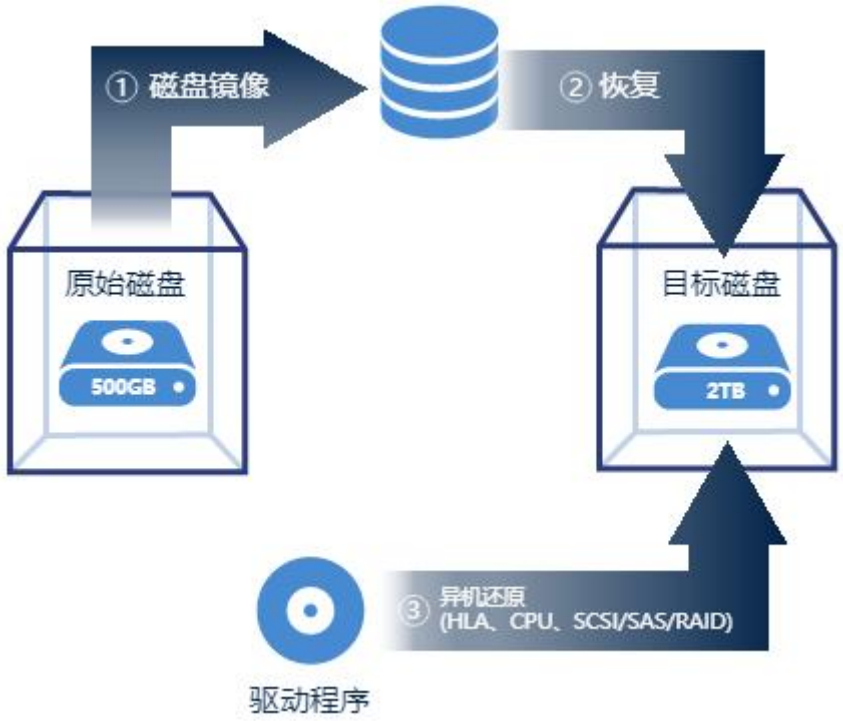
解决系统数据的写入问题，接下来就该解决如何把这个 RAM 系统引导起来。PXE 是一种成熟的网络引导系统的方案，无需额外的光盘、U 盘等存储介质，只需网络连通即可把镜像文件引导起来。PXE(Pre-boot Execution Environment，预启动执行环境)是由 Intel 公司开发的最新技术，工作于 Client/Server 的网络模式，支持工作站通过网络从远端服务器下载映像，并由此支持通过网络启动操作系统，在启动过程中，终端要求服务器分配 IP 地址，再用 TFTP(trivial file transfer protocol）或 MTFTP(multicast trivial file transfer protocol)协议下载一个启动软件包到本机内存中执行，由这个启动软件包完成终端基本软件设置，从而引导预先设置在服务器中的终端操作系统镜像。引导流程如下图所示：



就是在通过 PXE 引导的 Windows PE 中完成对操作系统的数据写入操作。

5.恢复的驱动兼容技术：

不同硬件恢复技术主要是解决硬件重要启动更改的问题。该神奇的技术无需启动就可以重新配置你的操作系统，它通过更改配置并且注入且激活驱动就可让操作系统启动。其主要流程如图中所示：



将磁盘镜像完整恢复后， 异机还原技术分析新的硬件平台,然后调整 Windows 的配置以匹配新的需求。主要涉及到如下硬件的驱动：

（1）CPU 驱动， 引擎分析 CPU 类型的更改（英特尔或者 AMD）、CPU 的数量（单核或者对称多处理 SMP）然后，为了匹配，它会更改操作系统的配置。这可以确保操作系统可以充分利用多个 CPU 或者不同数量的内核。

（2）HAL 驱动， 引擎分析每台计算机的类型，包括主板、芯片组、虚拟机配置，然后更改操作系统的 HAL（硬件抽象层）的配置。如果你的新的操作系统拥有不同数量的 CPU 或者不同的 CPU 品牌，这一点尤其重要。

（3）存储驱动， 异机还原技术分析目标硬件，注入启动操作系统所需的驱动。这些驱动包括 SATA、SAS、SCSI 和 RAID 驱动异机 SAN HBA 适配器。这是异机还原过程中最重要的部分。如果异机还原在 Windows 中没有找到合适的驱动器，将提示你提供标准 Microsoft 驱动。这些放到闪存驱动器或者软盘上的驱动和你通过使用 Windows F6 安装方式安装的驱动是一样的，这些驱动文件通常都带有 INF 和 SYS 后缀。另外，异机还原可以禁用新计算机上不需要的所有的启动硬件驱动。这样就消除了兼容的问题。

（4）网络驱动，为了确保网络的连接性，异机还原技术注入并且激活任何所需的网络驱动器。在 Windows 系统中，异机还原禁用并且移除旧网络适配器的所有配置，所以当配置网络时，你不需要删除隐藏或者丢失的设备。

3.4.卷备份

3.4.1.功能简介

3.4.1.1 定时备份

支持对主流操作系统上的卷定时备份。
(1)支持 Windows 全平台，支持 CentOS、RedHat 等 Linux 平台。
(2)支持只恢复卷上的某个文件。
(3)有效数据备份（即只备份分区上实际分配的数据区域）

3.4.1.2.实时备份

支持对主流操作系统上卷的实时备份，可保护卷上的数据库、应用、文件、图片、影像等类型数据，支持秒级的任意时间点回退，支持自动、手动接管，支持接管一键回切。
(1)支持 Windows 全平台，支持 CentOS、RedHat 等 Linux 平台。
(2)支持 NTFS、FAT32、EXT2、EXT3、EXT4、XFS 等文件系统类型。
(3)支持可回退数据存储空间的大小的灵活配置。
(4)支持备份数据加密传输。
(5)支持 IO 监控的同步及异步模式,可分别适应于主机内存资源是否充分的场景。

3.4.1.3.CDP

支持对主流操作系统上卷的实时备份，可保护卷上的数据库、应用、文件、图片、影像等类型数据，支持秒级的任意时间点回退，支持自动、手动接管，支持接管一键回切。
(1)通过标签点一致性技术保证数据库等结构化应用的一致性，完整性，可生成无限数量的 Oracle、SQLServer、MySQL、Sybase、ExchangeServer 等应用数据的一致性恢复点、接管点，最小间隔为 1 秒。
(2)支持恢复到任意时间点，最小间隔为 1 秒。

3.4.1.4.容灾

支持接管，当故障发生时可通过备机接管的方式完成业务的无缝切换运行，达到数据容灾的目标。
(1)主机级自动接管，支持在备份运行过程中，持续检测主机的运行状态，当宕机或网络故障时可在 60s 左右将故障前一刻的状态接管到备机,接管之后将主机

的 IP 附加到备机，将主机的数据挂载给备机，并将配置的接管应用还原。
(2)应用级自动接管，支持在备份运行过程中，持续检测应用的运行状态，当应用故障时，可在 30s 左右将应用故障前一刻的状态接管到备机。
(3)手动接管，支持手动选择接管任意时间点到备机，用于数据查验或使用，该接管方式不中断备份的正常运行。
(4)支持自定义接管到备机的挂载点。
(5)支持自定义应用故障检测脚本。
(6)支持自定义主机接管前执行脚本。
(7)支持自定义备机接管后支持脚本。
(8)支持自定义接管群组关系，良好的适应于集群环境。
(9)自定义多重心跳检测链路，防止单一条件触发误接管。
(10)支持一键回切，回切过程中，接管的 IP、数据以及应用不受影响，继续在备机上工作，直接停止接管以后，以上资源切换回主机运行，保障在回切过程中业务中断的时间极短（约在 30 秒左右）。
(11)支持实时查询备份的数据及应用状态，实时查询可实时的将主机最新的状态映射出来，用于用户数据查验，比较。
(12)支持定时查询，可查验备份时间链路中的某个时间点的数据及应用状态。

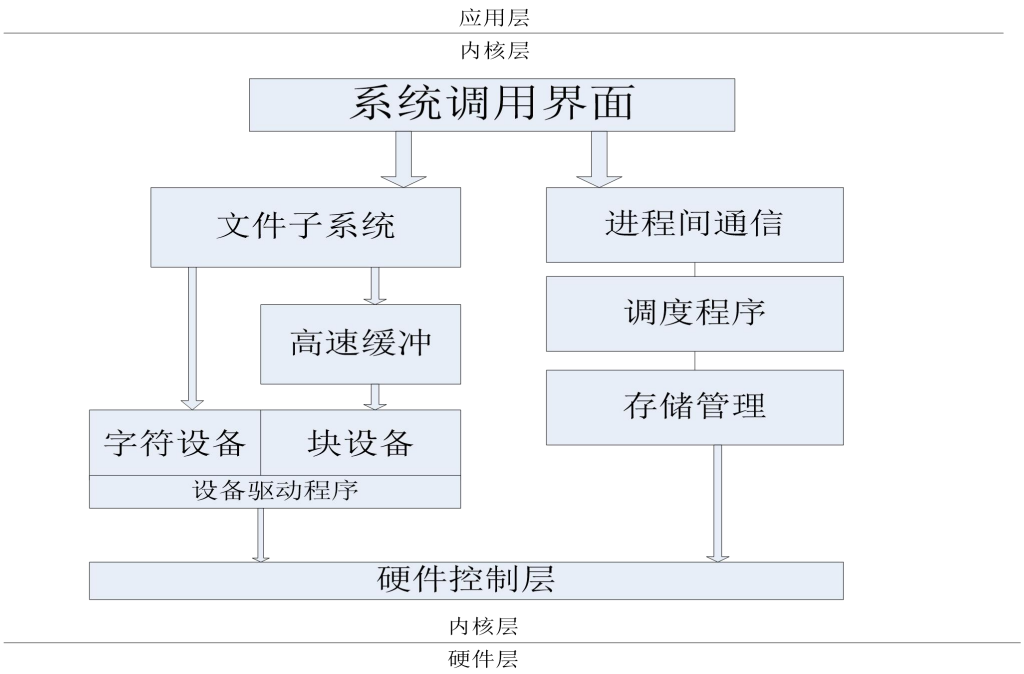
3.4.2.技术实现

3.4.2.1.定时备份

卷实时备份的技术核心点主要包括在源端通过过滤驱动程序对数据的过滤与复制，以及在目标端对数据的封装与存储，具体参见以下内容：

（1）数据的过滤与复制：

利用卷过滤驱动程序，设计一套稳定高效的数据分离机制，根据监控驱动程序所截获的，上层传递的写或者修改的数据信息，然后将相应信息添加应用标记，构造成相应的备份数据记录，并将记录拷贝至非生产数据的地方独立存储。该策略需跟踪记录生产数据每一次变化，使生产数据在遭到部分损毁时能够恢复到距灾难发生最近的正确状态，使容灾系统获得尽可能小的 RPO 指标。



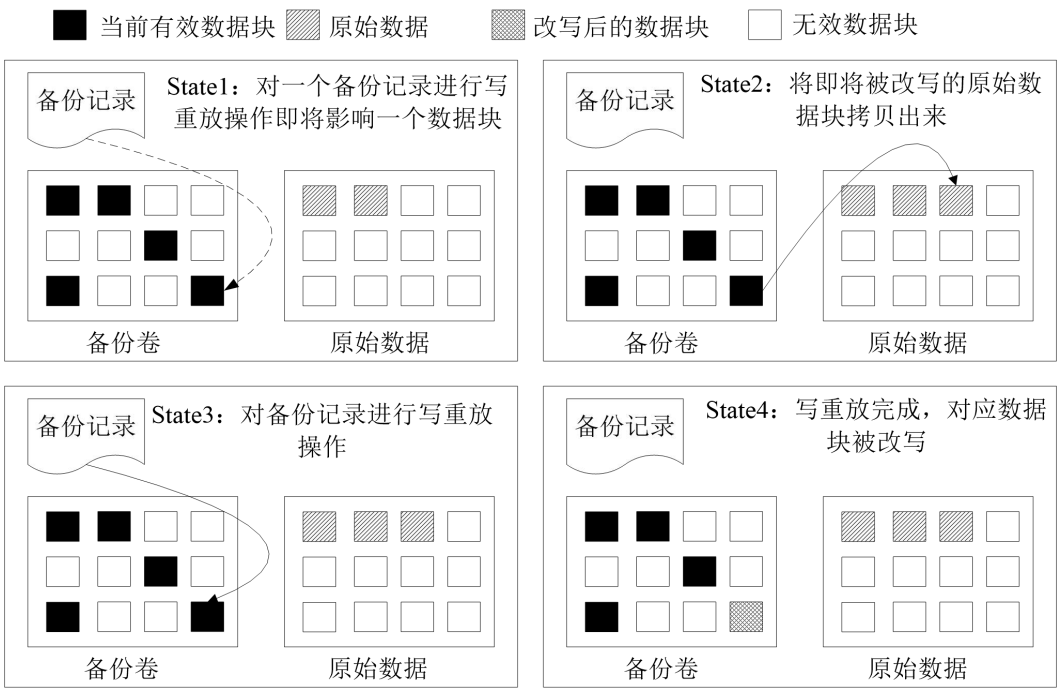
如上图所示，自上而下是操作系统内核的层次模型，该模块加载于块设备（卷）与硬件底层驱动之间，捕获上层调用所产生的写 IO 请求（Windows OS：IRP，Linux OS： BIO），对于每一个写 IO 请求，需创建对应的备份记录信息（包括时间戳、写入偏移量、写入数据长度以及写入数据等），然后需同时将元数据、变化数据写入到缓存文件进行记录。

（2）数据的封装与存储：

系统在对备份卷写入数据之前需执行写时拷贝的操作流程，并将备份卷上对应区域的原始数据进行封装及重放。

备份模块需要获得备份卷上每次写重放操作即将修改的原始数据，进而为每次数据变化形成日志记录。日志管理模块在存储服务器执行写重放操作时采用写时复制的方式读取备份卷上即将被覆盖的原始数据。由于需要连续执行写时复制操作。具体操作流程如下：

当一条备份记录到达存储服务器时，日志记录封装模块对备份记录中封装的写操作信息进行分析，按照 CDP 块大小，根据备份记录中写偏移 offset 和写入数据长度 len 将写操作映射到一个或多个 CDP 数据块上，将对应 CDP 块原始数据从备份卷上读出，同时通知存储子系统对备份记录进行写重放操作，日志封装模块将获得的原始数据按日志记录格式进行封装，封装好的日志记录被添加到网络发送队列等待转发。存储服务器上执行写时复制时数据状态迁移如下图所示。



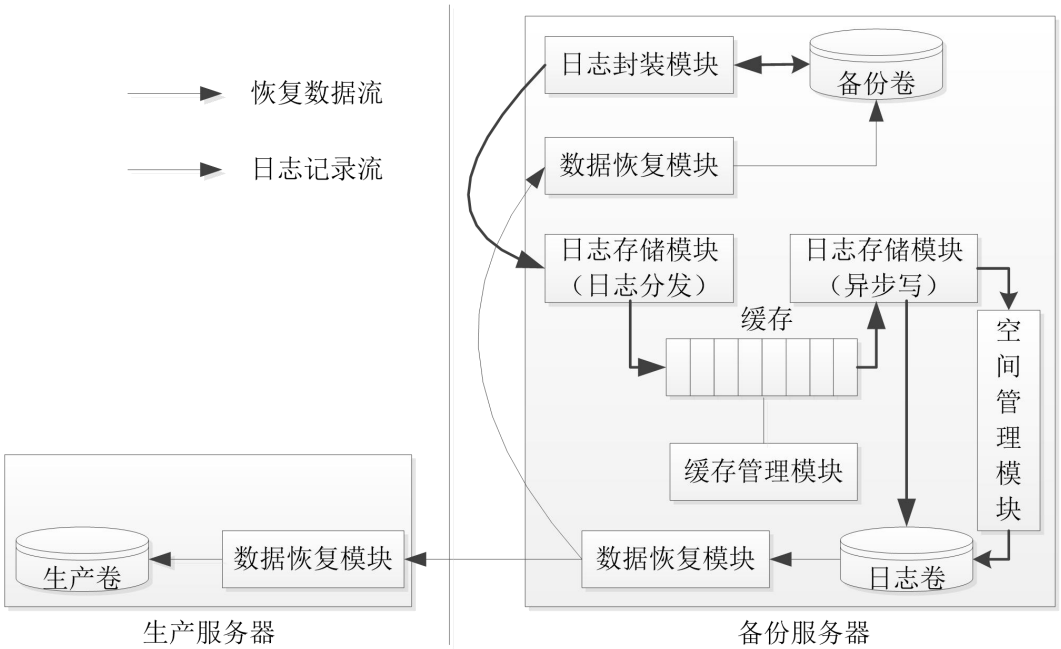
写时复制下数据状态迁移

3.4.2.2.CDP

CDP 意为持续数据保护，主要包括两个指标，实时备份与任意时间点回退。在上一节主要阐述实时备份的核心技术，这一节则主要描述任意时间点恢复的关键技术点。

（1）数据的恢复流：

基于数据块级的持续数据保护的解决方案，需要记录备份卷上的每一次数据变化，因此在存储端需消耗大量的存储空间，增加了底层物理存储的复杂性，而且在灾备任务生产卷数据因逻辑错误而遭到破坏时，需回滚备份卷到指定的时间点，线性增长的数据重组时间不能严格保证业务系统的连续性。针对以上问题，需通过合理规划的分段存储与快速查找的索引系统来完成任意点数据镜像的重组，以便于能快速启动恢复。



（2）数据的日志索引记录：

系统需要为灾备任务建立单独的日志记录索引，保证在数据回滚时日志记录的快速定位和读取。

要实现对生产数据的连续保护，提供数据保护窗口内任意时间点上的数据回滚功能，需要记录备份卷上的每一次数据变化，缺少中间任何一次变化记录都会造成备份数据的不连续，若以不连续的日志记录对生产卷进行数据回滚会导致生产数据的不可用。然而，连续记录通常会产生较大的日志记录量，如果仅仅顺序存储每条日志记录，那么在进行数据回滚时，需要依次从磁盘上读取日志记录，逐条遍历才能检索到回滚时间点上需要恢复的有效原始数据，涉及大量磁盘 I/O 操作，进而会导致较长的数据回滚时间。因此，需要在日志记录存储时为日志记录建立索引结构，减少数据回滚时日志记录检索耗时。

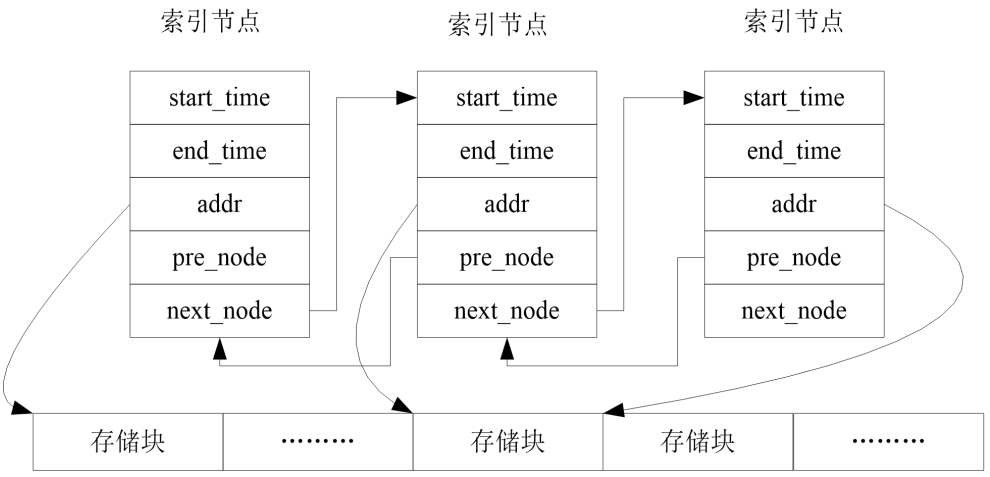
数据回滚时，日志记录检索的关键点在于找到与指定回滚时间点对应的日志记录在磁盘上的存储位置，其检索过程就是以时间戳为关键字的匹配过程，如果能够快速定位回滚时间点之后的第一条日志记录的位置，然后依次遍历处理需要处理的日志记录，就能大大减少对无关日志记录的遍历，从而提高日志记录检索的效率。

基于以上分析，我们采用时间戳为主键对日志记录建立索引。采用时间片的方式对数据建立时间索引，将数据保护时间窗口按等长的时间间隔划分为多个时间片，为每个时间片内的日志记录建立索引。这种实现方法虽然简单，但是考虑到每个灾备任务所对应的应用服务不同，各灾备任务在不同时间段上的日志记录量有所不同，如果采用固定的时间间隔建立索引，会造成日志记录在不同索引范

围内分配不均的情形。

为了使索引项之间分布的日志记录量相对均匀，本系统以一个基本单位的日志记录量来为日志记录建立基于时间范围的索引结构，使索引项在灾备任务相对繁忙，日志量较大的时间段内相对密集，而在生产数据产生速率较低，日志量较少的时间段内相对稀疏，索引的时间区间根据灾备任务的繁忙程度动态变化。

在系统中，采用一个磁盘存储块（例如 64M）为基本单位建立索引项，当日志记录累积满一个磁盘存储块大小时为该存储块建立一个索引项，索引项中记录该存储块内日志记录跨越的时间区间，存储块在磁盘上的起始位置。将同一个灾备任务的索引项按时间顺序组织成双向链表，整个索引结构如下图所示。



日志索引结构图

（3）数据的逆向回退机制：

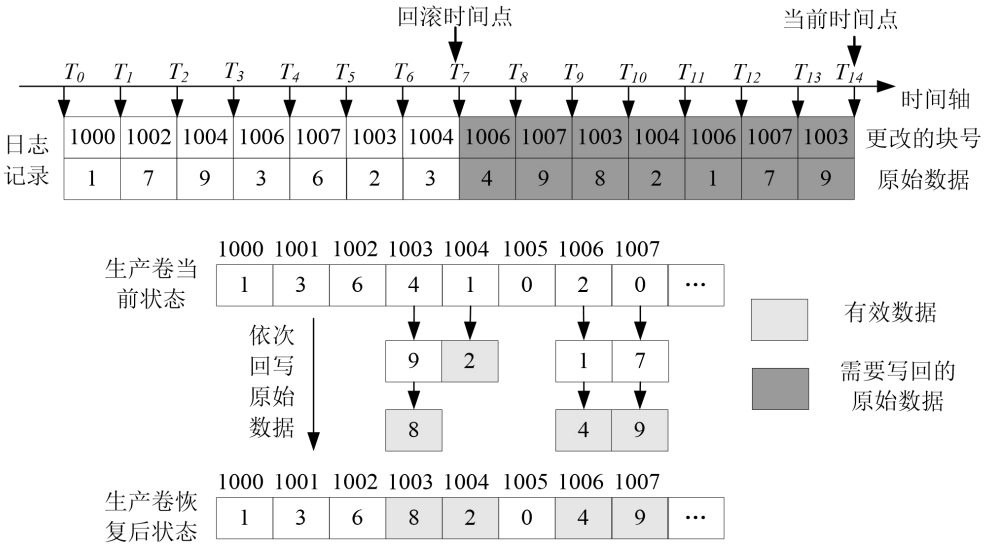
如果需要恢复的时间点在数据保护窗口内，数据恢复模块可以根据用户指定的回滚时间点读取日志记录对生产卷数据进行重建，使其恢复到一个正确的状态。

系统采用备份原数据的方式封装日志记录跟踪数据变化，其实质相当于记录了每次数据变化的回退操作，与数据库 undo 日志类似。数据库中通过逆序执行 undo 日志中的事务条目来撤销已经提交的事务，将数据库回滚到一个历史逻辑一致的状态。基于原始数据日志记录的卷数据回滚可以采用与数据库 undo 操作类似的逆向回滚恢复机制进行数据回滚，其主要方法是：将回滚时间点到当前时间点之间的日志记录中的原始数据按照时间逆序依次写回到生产卷上，以此对生产卷数据重构，使其重新回到回滚时间点上的状态。

如下图所示，当前时间点为T_{14}时刻，需要回滚的时间点为T_7时刻，按照逆向回滚恢复机制，需要逆序将T_{13}时刻到T_7时刻的原始数据依次写回到生产卷。根据局部性原理[31]，在短时间内应用程序会访问磁盘上连续相邻的磁盘块，在

我们的示例中，由于磁盘 I/O 的空间局部性，在回滚时间点到当前时间点之间对块 1006，块 1007，块 1003 分别执行了 2 次写操作，对块 1004 执行了一次写操作。

在执行数据回滚时，原始数据回写操作具有相同的局部性，在以下示例中一共需要执行 7 次原始数据回写操作，而对相同块的重复写操作中，前一次写操作会被后一次写操作覆盖，只有最后一次写入才是有效写入，在本例中，有效写入只有 4 次，有 3 次写操作无效。回滚时间点离当前时间点越远，执行数据回滚时产生的无效写操作越多。



逆向回滚恢复机制恢复过程示例

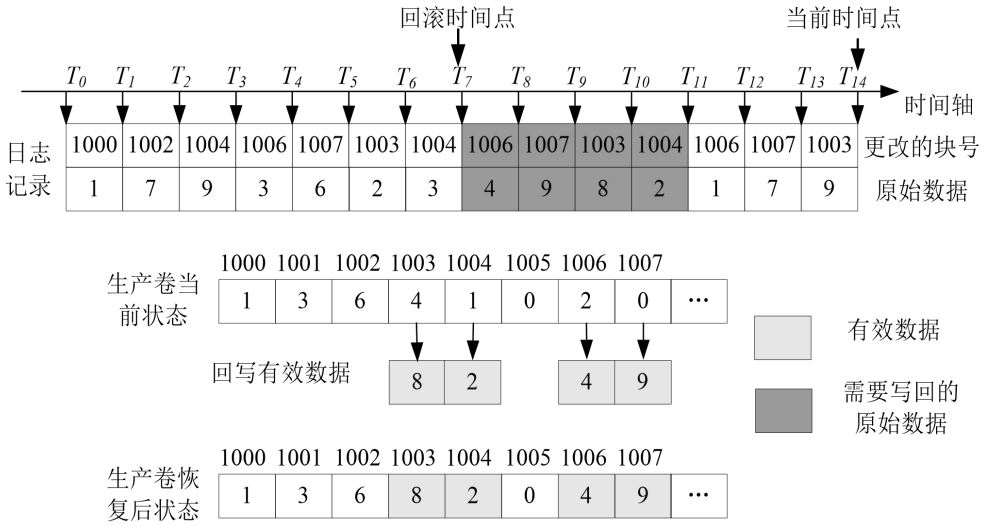
综上所述，在复制参考数据模型下采用逆向回滚恢复机制进行数据回滚会产生大量的无效写操作，由于原始数据需要通过网络传输到本地服务器上执行回写操作，大量无效数据的传输和磁盘写操作大大降低了数据回滚效率。

通过上例的分析，如果要提高数据回滚的效率，就需要减少无效写操作，一次性写入回滚时间点的有效数据能完全避免无效写操作，如果能快速提取回滚时间点到当前时间点之间发生数据变化的所有 CDP 块在回滚时间点上的有效数据，就能极大的提高数据回滚效率，完全避免对同一 CDP 块的重复无效写操作。

本系统设计的数据回滚方法采用正向数据回滚机制对生产卷进行数据回滚操作。本系统采用的日志记录模型以原始数据为备份对象，对一个特定的 CDP 块 N 来说，其在回滚时间点上的原始数据记录在回滚时间点后第一条对应的日志记录中，如果从回滚时间点正向遍历日志记录，第一次遇到块 N 的日志记录中所记录的原始数据就是块 N 需要恢复的有效数据，数据恢复模块只需要将其中的原始数据写回到生产卷上即完成了对块 N 的数据回滚操作，对于块 N 的后续日志记录，数据恢复模块可以直接忽略而不需要进行任何处理。在正向回滚机制中，每

一个在回滚时间点到当前时间点之间发生过数据改变的 CDP 块都只需要一次原始数据回写操作，完全避免了无效写操作，极大的减少了恢复数据的传输量和磁盘写操作量。

同样以前文中所用的示例为例，下图演示了正向回滚机制的回滚过程。从回滚时间点开始遍历日志记录，依次在 T_7，T_8，T_9，T_{10} 读取到块 1006、1007、1003、1004 的原始数据，由于它们都是距离回滚时间点最近的原始数据，因此都是需要恢复的有效数距，只需要将这些有效数据恢复到生产卷和备份卷上即可，而对于后面三个时间点上的日志记录均是对已恢复 CDP 块的记录，不需要进行回写处理。在该例中，需要恢复的数据块有 4 块，而恢复时对每一块各执行了一次回写操作，总共 4 次回写操作，完全避免了无效写操作，相对于逆向回滚机制中的 7 次回写操作，很大程度上提高了数据回滚效率。



正向回滚恢复机制恢复过程

3.4.2.3.容灾

基于卷 CDP 功能的容灾，主要的关键点包括以下内容：

（1）容灾的实现方式

在上面的章节中介绍了从源端捕获数据到目标端存储数据。而当用户的源端发生故障，例如磁盘损坏、病毒感染或其他不可抗力因素造成的更为严重层面的破坏，用户不得不面临生产停滞，业务无法正常继续的情况。

而容灾解决的问题便是当以上情况发生时，可在短时间内恢复用户的业务，让生产继续运行。系统提供的解决方案是在部署容灾系统时需独立部署一套与源端对应的备机环境，备机需预装好与源端主机相同版本的操作系统与应用程序。当故障发生时，备份服务端将已备份的数据通过 iSCSI 协议挂载给备机，然后将主机的 IP 地址漂移给备机，备机将应用程序与数据附加，将依赖的服务启动，

便可以通过漂移的 IP 继续提供服务了。

（2）接管的分类与触发

这里把（1）中所提到的容灾的解决方式称作“接管”，而接管的分类和触发如下：

a)自动接管：

自动接管需在创建备份作业的时候配置，当配置的接管条件成立时，便会触发接管。而自动接管的检测条件包括如下：

■检测主机故障，在备份运行期间，主机会通过长链接周期性向备份服务器发送心跳检测包，备份服务端收到心跳包以后会更新主机的存活时间，同样备份服务端会检测每个在备份的主机的存活时间是否在周期性更新，如果检测到存活时间超过一段时间（例如 60 秒）未更新便认为主机已经离线，相应的便会触发自动接管。

■检测主机上的指定应用，例如 SQLServer 服务，Oracle 服务，主机上运行的客户端程序周期性的检测这些服务的状态，如状态变成停止便认为应用故障，相应的便会触发自动接管。

■自定义检测脚本，在创建作业时配置脚本的路径，主机上运行的客户端程序周期性调用脚本，并判断脚本的执行结果，如执行结果出错时便触发自动接管。

b)手动接管：

与自动接管不同，手动接管不能自动触发，是用户需要接管时可创建一个接管作业来完成接管。另外还有其他几点区别：

■手动接管可接管任意时间点，用户在创建手动接管作业时便可指定接管时间。

■手动接管不会漂移主机的 IP。

■手动接管不会影响备份的运行，即在备份运行的过程中，用户可通过该功能来完成对备份数据可用性的校验。

c)实时查询：

实时查询在配置启动以后，可以将最新的备份数据实时的挂载给查询机器，实时查询可供用户随时查验最新的备份数据。

d)定时查询：

对比与实时查询，定时查询则是用户配置一个特定的时间点来查询备份数据的最新状态。

（3）接管数据的可靠性

系统通过标签点技术来保障这些特定时刻数据的一致性与可靠性。主要的实现技术包括数据库与文件系统的刷脏。

a)CheckPoint：

checkpoint 是一个内部事件，这个事件激活以后会触发数据库写进程(DBWR)将数据缓冲(DATABUFFER CACHE)中的脏数据块写出到数据文件中。

当 checkpoint 发生时，数据库会通知 DBWR 进程，把修改过的数据，也就是 Checkpoint SCN 之前的脏数据（Dirty Data）从 Buffer Cache 写入磁盘，当写入完成之后，CKPT 进程更新控制文件和数据文件头，记录检查点信息，标识变更。

因此数据库可以在性能允许的情况下，使得检查点的 SCN 主键逼近 Redo 的最新更新，那么最终可以获得一个最佳平衡点，使得数据库可以最大化地减少恢复时间。

b)文件系统缓存刷新：

在操作系统中，在文件或数据处理过程中一般先放到内存缓冲区中，等到适当的时候再写入磁盘，以提高系统的运行效率。而刷新缓存则可用来强制将内存缓冲区中的数据立即写入磁盘中。

（4）逆向回切

在发生接管以后，如果主机从故障中恢复，用户需要将业务回切到主机上运行，备份系统不仅完成了回切的过程且在该过程中实现了业务不停顿，具体的流程与技术点如下：

a)在接管状态下启动备机回切到主机。
b)备机对接管数据卷创建快照 SP，且同时启动对数据卷写 IO 的监控与复制。
c)备机将快照 SP 的数据传输到主机。
d)备机实时的将监控到的数据同步到主机，该过程业务仍然运行在备机之上。
e)停止接管，备机停止监控，将剩余的数据同步完，停止业务，卸载主机的 IP。
f)主机加载数据，启动业务，配置 IP，完成切换。

四、产品全局功能与技术介绍

4.1.数据压缩

4.1.1.功能简介

（1）支持通过 LZO、LZBJ、GZIP 等算法对备份数据的压缩存储。
（2）支持通过 LZO 等算法对数据传输过程中压缩。

数据压缩是指在关键不丢失信息的前提下，缩减数据量以减少存储空间，提高其传输、存储和处理效率的一种技术，或者指按照一定的算法对数据进行重新组织，减少数据的冗余和存储的空间。数据压缩包括有损压缩和无损压缩。它是一个数据编码的过程，可以理解为用不同的语言表达同样的含义，只不过有些语言很简练，而有些却很繁琐。我们的目的就是通过数据编码用最简洁的方式表达数据包含的信息，更确切地说是用最少的空间存储最多的信息。

数据压缩的方式有很多，一般来说可以分为无损压缩和有损压缩。

无损压缩是指使用压缩后的数据进行解压缩，得到的数据与原来的数据完全相同；无损压缩用于要求重构的信号与原始信号完全一致的场合。一个很常见的例子是磁盘文件的压缩。根据目前的技术水平，无损压缩算法一般可以把普通文件的数据压缩到原来的 1/2～1/4。一些常用的无损压缩算法如 LZO（）压缩算法等。

有损压缩是指使用压缩后的数据进行解压缩，得到的数据与原来的数据有所不同，但不影响人对原始资料表达的信息的理解。有损压缩适用于重构信号不一定非要和原始信号完全相同的场合。例如，图像和声音的压缩就可以采用有损压缩，因为其中包含的一些数据往往超过我们的视觉系统和听觉系统所能接收的范围，丢掉一些也不至于使人对声音或者图像所表达的意思产生误解，但可大大提

高压缩比。

4.1.2.技术实现

备份系统中主要是使用 LZO 无损压缩技术， LZO 算法是一种词典编码，它用的词典就是前面经过编码的信息序列。编码器利用一个滑动窗口查询输入的文本信息，如图所示。

滑动窗口包括两部分：查询缓冲区和预见缓冲区。查询缓冲区包含了最近刚经过编码的序列，预见缓冲区包含接下来需要编码的序列。在图 7-2 中的查询缓冲区包含了 8 个字符，预见缓冲区包含了 7 个字符。实际应用时，缓冲区的大小要比这个大得多，在这里只是为了方便说明而设置小一点。

为了对预见缓冲区中的序列进行编码，编码器在查询缓冲区中从后往前移动一个匹配指针，直到这个指针指向与预见缓冲区中第一个字符匹配的字符为止。从预见缓冲区到匹配指针之间的距离称为偏移量，记作 offset。接下来，编码器会继续检查匹配指针后面的连续字符是否与预见缓冲区中的连续字符相匹配。查询缓冲区和预见缓冲区中最长的匹配字符串长度称作匹配长度，记作 length。编码器要查找最长的匹配字符串，当编码器找到这个最长匹配串时，会用一个三元组<o, c="" 1,="" style="overflow-wrap: break-word;">对它进行编码。三元组中的 o 代表 offset，l 代表 length，c 代表预见缓冲区中最长匹配串后的第一个不匹配字符。例如，在图 7-2 中查询指针指向了最长匹配串的开头，offset 等于 7，匹配长度为 4，预见缓冲区中的第一个不匹配字符为 r。设置第三个元素 c 的原因是为了便于处理查询缓冲区和预见缓冲区中没有匹配字符串的情形。这种情况下，三元组中的 offset 和 length 都被设置为 0，c 被设置为不匹配字符本身。

如果设查询缓冲区的大小为 S，窗口（包括查询缓冲区和预见缓冲区）的大小为 W，源文件中的字符集大小为 A，那么用定长的三元组进行编码需要的比特数是[log2S]+[log2W]+[log2A]，其中[]表示上取整。注意，第二项是[log2W]而不是[log2S]，因为匹配长度可能会超出查询缓冲区，这种情况将在下面例子中说明。

在下面的例子中，我们将会考虑如下编码中可能遇到的三种情形：

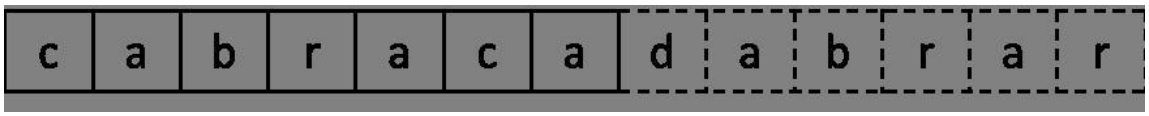
（1）没有匹配字符。

（2）有一个匹配字符。

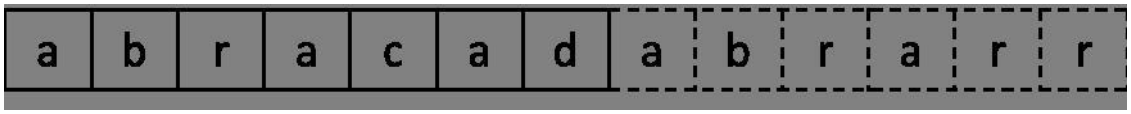
（3）匹配字符串超出了查询缓冲区的范围。

例如编码序列如下：... cabracadabrarrarrad ...

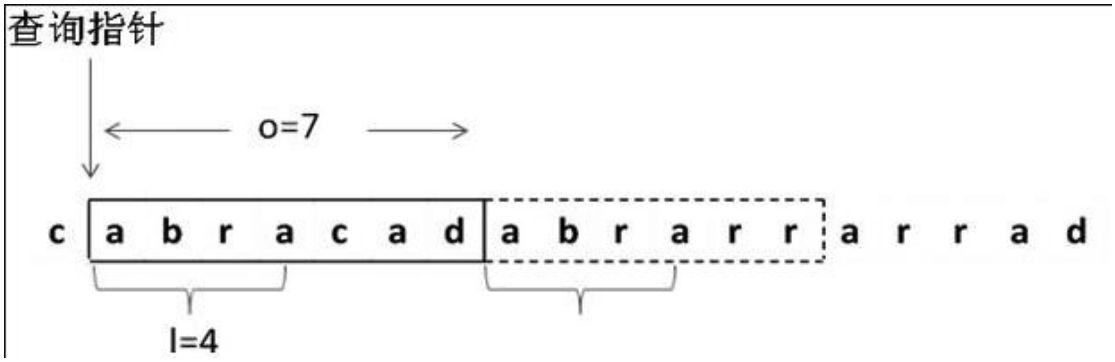
假设窗口的大小为 13，那么预见缓冲区的大小为 6，当前状态如图所示。



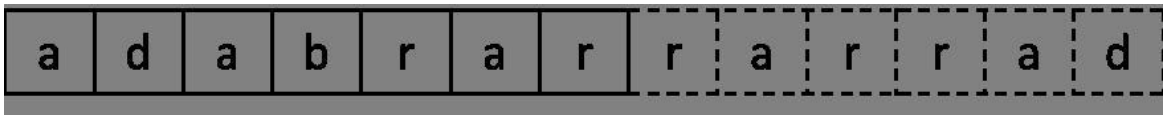
预见缓冲区中包含了 dabrar。我们在查询缓冲区中查找 d 的匹配字符，但是没有找到，所以输出<0, 0, C(d)>。前两个元素表示没有匹配字符，第三个元素 C(d)代表第一个 d 的编码。下面，我们继续编码的过程。因为前面对一个字符 d 进行了编码,所以把窗口向后移动一个字符。现在缓冲区中的情况如图所示。



预见缓冲区中包含字符串 abrarr。从当前位置往前在查找缓冲区中查找匹配字符，在偏移位置为 2 的地方找到了一个匹配的 a。这个匹配字符串的长度为 1。再往前查找，我们又在偏移位置为 4 的地方找到另一个匹配的 a，这个匹配字符串的长度仍然是 1。继续向前查找，我们在偏移位置为 7 的地方找到了第三个匹配的 a，但这次匹配字符串的长度为 4（见图 7-5），我们找到了匹配最长的字符串。于是，我们用三元组<7, 4, C(r)>代替字符串 abra，并且将窗口向后移动 5 个字符。

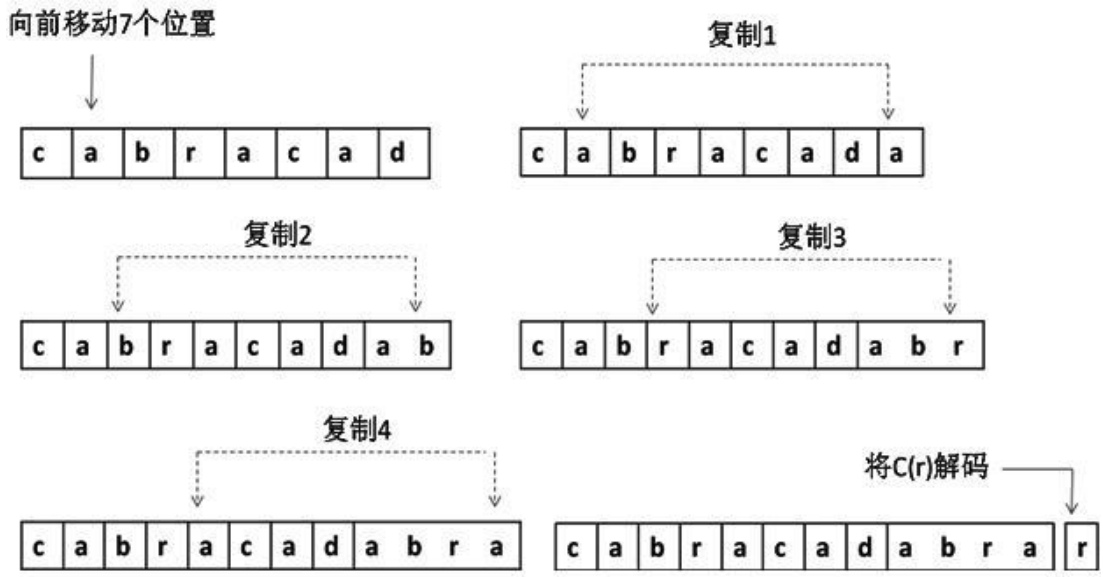


现在的窗口中包含如图所示的内容。

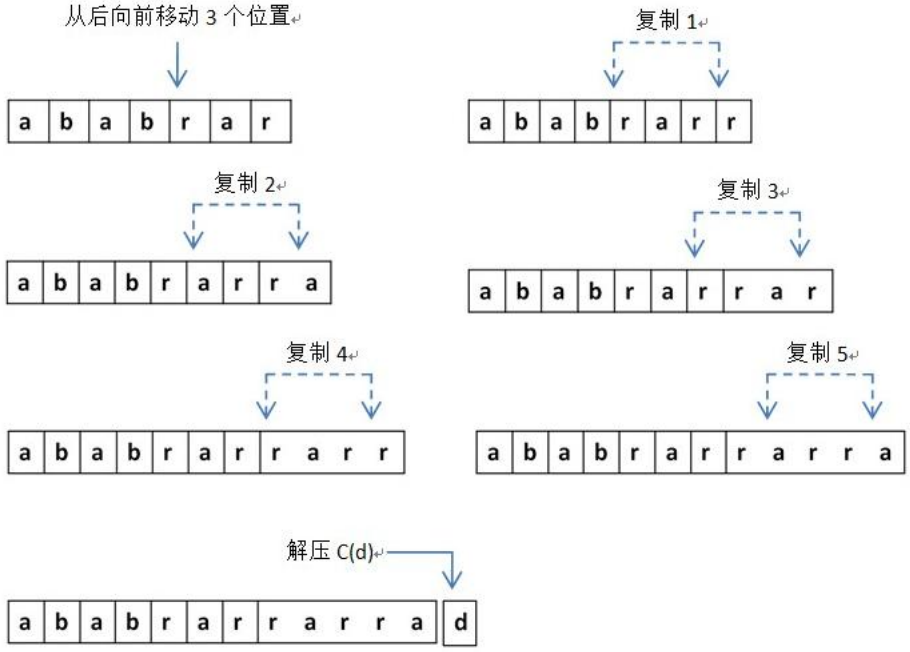


预见缓冲区中包含字符串 rarrad，向前寻找匹配字符串。第一个匹配字符串的偏移位置是 1，长度为 1；第二个匹配字符串偏移位置是 3，它的长度貌似是 3，但实际上我们可以越过查找缓冲区，将重复字符串扩展到预见缓冲区，这一点前面已经提到了。所以这里重复字符串的长度为 5，而不是 3。

这么做是可行的，通过下面解码的过程可以得到证实。假设已经解压完成的字符串为 cabraca，并且我们扫描到了三元组<0, 0, C(d)>，<7, 4, C(r)>，<3, 5, C(d)>。第一个三元组很好解码，它没有和已经解压的字符串重复的字符串，并且下一个字符为 d。现在得到的字符串为 cabracad。第二个三元组的第一个元素说明了重复字符串的偏移位置为 7，于是把指针向前移动 7 个字符，第二个元素说明重复长度为 4。于是连续输出后面的 4 个字符。具体解码过程如图 7-7 所示。



最后，让我们看看第三个三元组<3, 5, C(d)>是怎样解码的。根据第一个元素我们把指针向前移动 3 个字符，然后开始重复输出。前三个重复的字符是 rar，复制指针继续向后移动，如图所示，输出刚复制过的第一个字符 r，同样再输出第一个复制过的字符 a。这样一来，虽然在开始时只是复制了 3 个字符，但是最终我们解码出了 5 个字符。注意，匹配字符串必须起始于查找缓冲区，但是可以延伸到预见缓冲区。事实上，如果预见缓冲区中的最后一个字符是 r 而不是 d，即后面跟着一连串重复的 rar，那么整个 rar 重复序列可以用一个三元组进行编码压缩。



正如我们看到的，LZO 模式是一个非常简单的自适应模式，它不要求知道源文件的任何信息，并且几乎也不需要基于源文件的字符集，而且性能好，压缩速度快。

4.2.数据重删

4.2.1.功能简介

支持通过 SHA256 等算法对备份数据的重删存储。

备份设备中总是充斥着大量的冗余数据，为了解决这个问题，节省更多空间，“重复删除”技术便应运而生。采用“重复删除”技术可以将数据的实际使用空间大大减小，从而让出更多的备份空间，不仅可以使磁盘上的备份数据保存更长的时间，而且还可以节约离线存储时所需的大量的带宽。重复数据删除技术支持在已有的磁盘设备上存储更多的备份数据，因此采用“重复数据删除”技术可以增加您保存备份数据的时间，减少数据中心的消耗，降低成本。

该技术可以很大程度上减少对物理存储空间的需求，从而满足日益增长的数据存储需求。Dedupe 技术可以带许多实际的利益，主要包括以下诸多方面：

（1）满足 ROI(投资回报率,Return On Investment)/TCO(总持有成本,Total Cost of Ownership)需求；

（2）可以有效控制数据的急剧增长；

（3）增加有效存储空间，提高存储效率；

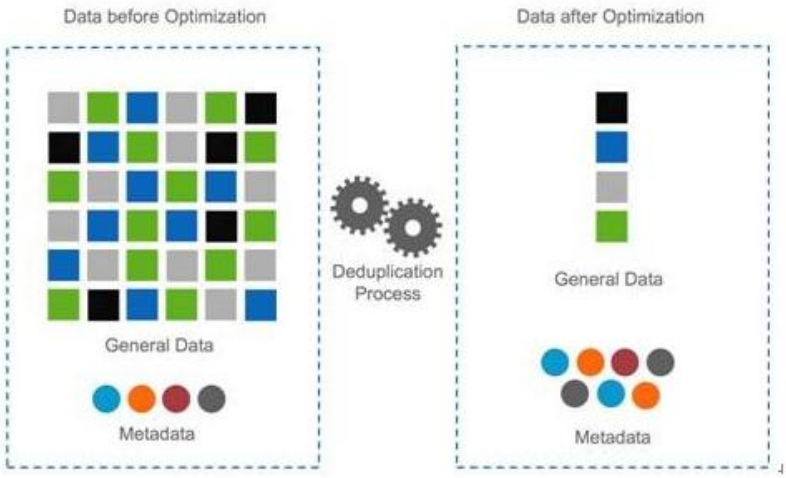
（4）节省存储总成本和管理成本；

（5）节省数据传输的网络带宽；

（6）节省空间、电力供应、冷却等运维成本。

4.2.2.技术实现

重复数据删除技术通过删除数据集中重复的数据，只保留其中一份，从而消除冗余数据。如下图所示。



存储系统的重复数据删除过程一般是这样的：首先将数据文件分割成一组数据块，为每个数据块计算指纹，然后以指纹为关键字进行 Hash 查找，匹配则表

示该数据块为重复数据块，仅存储数据块索引号，否则则表示该数据块是一个新的唯一块，对数据块进行存储并创建相关元信息。这样，一个物理文件在存储系统就对应一个逻辑表示，由一组 FP 组成的元数据。当进行读取文件时，先读取逻辑文件，然后根据 FP 序列，从存储系统中取出相应数据块，还原物理文件副本。从如上过程中可以看出，Dedupe 的关键技术主要包括文件数据块切分、数据块指纹计算和数据块检索。

主要实现流程：

（1）文件数据块切分：Dedupe 按照消重的粒度可以分为文件级和数据块级。文件级的 dedupe 技术也称为单一实例存储(SIS, Single Instance Store)，数据块级的重复数据删除其消重粒度更小，可以达到 4-24KB 之间。显然，数据块级的可以提供更高的数据消重率，因此目前主流的 dedupe 产品都是数据块级的。数据分块算法主要有三种，即定长切分(fixed-size partition)、CDC 切分(content-defined chunking)和滑动块(sliding block)切分。定长分块算法采用预先义好的块大小对文件进行切分，并进行弱校验值和 md5 强校验值。弱校验值主要是为了提升差异编码的性能，先计算弱校验值并进行 hash 查找，如果发现则计算 md5 强校验值并作进一步 hash 查找。由于弱校验值计算量要比 md5 小很多，因此可以有效提高编码性能。定长分块算法的优点是简单、性能高，但它对数据插入和删除非常敏感，处理十分低效，不能根据内容变化作调整和优化。

滑动块(sliding block)算法结合了定长切分和 CDC 切分的优点，块大小固定。它对定长数据块先计算弱校验值，如果匹配则再计算 md5 强校验值，两者都匹配则认为是一个数据块边界。该数据块前面的数据碎片也是一个数据块，它是不定长的。如果滑动窗口移过一个块大小的距离仍无法匹配，则也认定为一个数据块边界。滑动块算法对插入和删除问题处理非常高效，并且能够检测到比 CDC 更多的冗余数据，它的不足是容易产生数据碎片

（2）数据块指纹计算：数据指纹是数据块的本质特征，理想状态是每个唯一数据块具有唯一的数据指纹，不同的数据块具有不同的数据指纹。数据块本身往往较大，因此数据指纹的目标是期望以较小的数据表示(如 16、32、64、128 字节)来区别不同数据块。数据指纹通常是对数据块内容进行相关数学运算获得，从当前研究成果来看 Hash 函数比较接近与理想目标，比如 MD5、SHA1、SHA-256、SHA-512、为 one-Way、RabinHash 等。另外，还有许多字符串 Hash 函数也可以用来计算数据块指纹。然而，遗憾的是这些指纹函数都存在碰撞问题，即不同数据块可能会产生相同的数据指纹。相对来说，MD5 和 SHA 系列 HASH 函数具有非常低的碰撞发生概率，因此通常被采用作为指纹计算方法。其中，MD5 和 SHA1 是 128 位的，SHA-X(X 表示位数)具有更低的碰撞发生概率，但同时计算量也会大大增加。实际应用中，需要在性能和数据安全性方面作权衡。另外，还可以同时使用多种 Hash 算法来为数据块计算指纹。

（3）数据块检索：对于大存储容量的 Dedupe 系统来说，数据块数量非常庞大，尤其是数据块粒度细的情况下。因此，在这样一个大的数据指纹库中检索，性能就会成为瓶颈。信息检索方法有很多种，如动态数组、数据库、RB/B/B+/B*树、Hashtable 等。Hash 查找因为其 O(1)的查找性能而著称，被对查找性能要求高的应用所广泛采用，Dedupe 技术中也采用它。Hashtable 处于内存中，会消耗大量内存资源，在设计 Dedupe 前需要对内存需求作合理规划。根据数据块指纹长度、数据块数量(可以由存储容量和平均数据块大小估算)可以估算出内存需求量。

散列表(Hashtable，也叫哈希表)，是根据关键码值(Key value)而直接进行

访问的数据结构。也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表。散列表的查找过程基本上和造表过程相同，一些关键码可通过散列函数转换的地址直接找到，另一些关键码在散列函数得到的地址上产生了冲突，需要按处理冲突的方法进行查找。

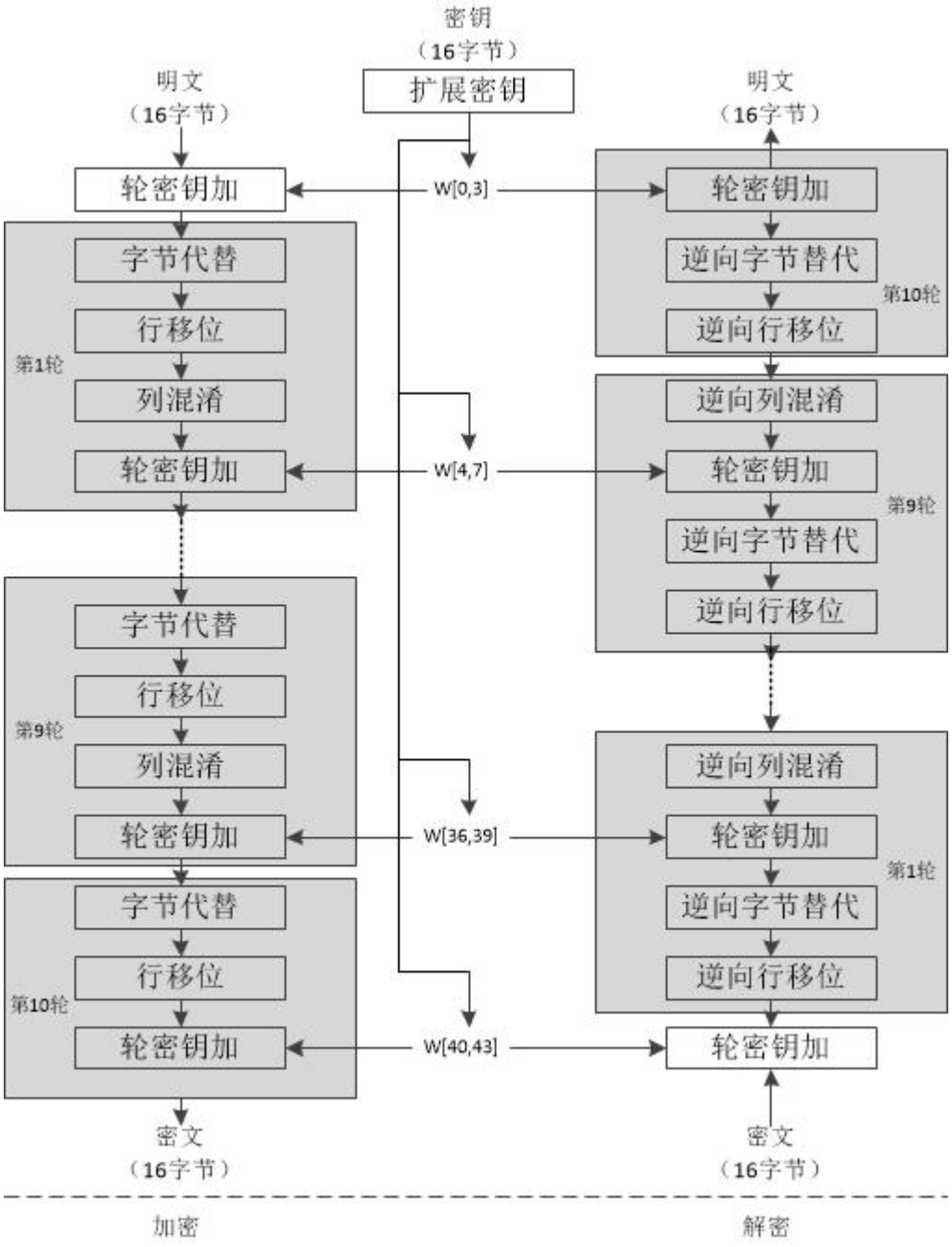
4.3.数据加密存储

4.3.1.功能简介

信息的保密性是信息安全性的一个重要方面。保密的目的是防止对手破译信息系统中的机密信息。加密是实现信息保密性的一种重要手段，就是使用数学方法来重新组织数据，使得除了合法的接收者外，任何其他人要想恢复原先的"消息"(将原先的消息称作"明文")或读懂变化后的"消息"(将变化后的消息称作"密文")是非常困难的，将密文变换成明文的过程称作解密。可见，加密技术可使一些重要数据存储在一台不安全的计算机上，或可以在一个不安全的信道上传送，只有持有合法密钥的一方才能获得"明文"。所谓加密算法就是对明文进行加密时所采用的一组规则，解密算法就是对密文进行解密时所采用的一组规则。加密算法和解密算法的操作通常都是在一组密钥控制下进行的，分别称为加密密钥和解密密钥。根据加密密钥和解密密钥是否相同，可将现有的加密体制分为两种:一种是私钥或对称加密体制、这种体制的加密密钥和解密密钥相同，其典型代表是美国的数据加密标准(DES):另一种是公钥或非对称加密体制，这种体制的加密密钥和解密密钥不相同并且从其中一个很难推出另一个。加密密钥可以公开，而解密密钥可由用户自己秘密保存，其典型代表是 RSA 体制。

4.3.2.技术实现

本系统主要是用的是国际公认比较安全的 AES 加密方案。AES 是一个分组密码，属于对称密码范畴，AES 算法的模块在对称密码领域特别是分组密码领域常有使用。AES 加密算法主要涉及 4 种操作：**字节替代**（SubBytes）、**行移位**（ShiftRows）、**列混淆**（MixColumns）和**轮密钥加**（AddRoundKey）。下图给出了 AES 加解密的流程，从图中可以看出：1）解密算法的每一步分别对应加密算法的逆操作，2）加解密所有操作的顺序正好是相反的。正是由于这几点（再加上加密算法与解密算法每步的操作互逆）保证了算法的正确性。加解密中每轮的密钥分别由种子密钥经过**密钥扩展算法**得到。算法中 16 字节的明文、密文和轮子密钥都以一个 4x4 的矩阵表示。



（1）字节替代

字节代替的主要功能是通过 S 盒完成一个字节到另外一个字节的映射。S 盒的详细构造方法可以参考文献[4]。这里直接给出构造好的结果，下图(a)为 S 盒，图(b)为 S^{-1}（S 盒的逆）。S 盒用于提供密码算法的**混淆性**。

		y														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

(b) Inverse S-box

S 和 S^{-1} 分别为 16x16 的矩阵，完成一个 8 比特输入到 8 比特输出的映射，输入的高 4-bit 对应的值作为行标，低 4-bit 对应的值作为列标。假设输入字节的值为 $a=a_7a_6a_5a_4a_3a_2a_1a_0$，则输出值为 $S[a_7a_6a_5a_4][a_3a_2a_1a_0]$，$S^{-1}$ 的变换也同理。

例如：字节 00000000_B 替换后的值为（S[0][0]=）63_H，再通过 S^{-1} 即可得到替换前的值，（S^{-1} [6][3]=）00_H。

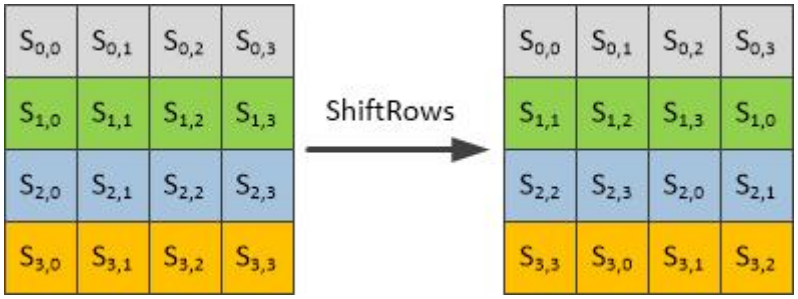
（2）行移位

行移位是一个 4x4 的矩阵内部字节之间的置换，用于提供算法的**扩散性**。

（3）正向行移位

正向行移位用于加密，其原理图如下。其中：第一行保持不变，第二行循环左移 8 比特，第三行循环左移 16 比特，第四行循环左移 24 比特。

假设矩阵的名字为 state，用公式表示如下：state’[i][j] = state[i][(j+i)%4];其中 i、j 属于[0,3]。



（4） 逆向行移位

逆向行移位即是相反的操作，即：第一行保持不变，第二行循环右移 8 比特，第三行循环右移 16 比特，第四行循环右移 24 比特。

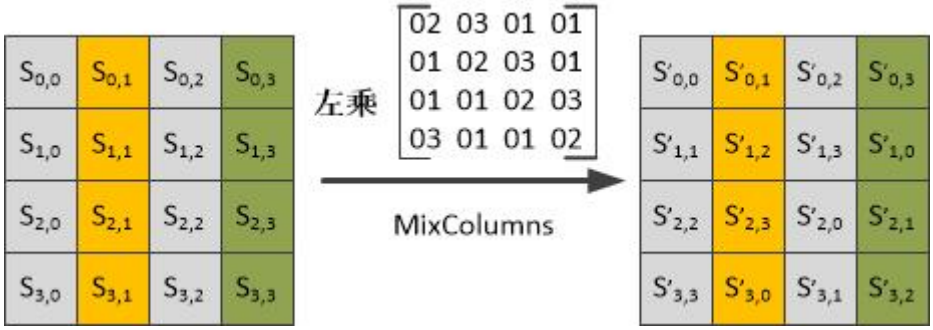
用公式表示如下：state’[i][j] = state[i][(4+j-i)%4];其中 i、j 属于[0,3]。

（5）列混淆

列混淆：利用 GF(2^8)域上算术特性的一个代替，同样用于提供算法的**扩散性**。

1.正向列混淆

正向列混淆的原理图如下：



根据矩阵的乘法可知，在列混淆的过程中，每个字节对应的值只与该列的 4 个值有关系。此处的乘法和加法都是定义在 GF(2^8)上的，需要注意如下几点：

a.将某个字节所对应的值乘以 2，其结果就是将该值的二进制位左移一位，如果原始值的最高位为 1，则还需要将移位后的结果异或 00011011；

b.乘法对加法满足分配率，例如：$07 \cdot S_{0,0}=(01 \oplus 02 \oplus 04) \cdot S_{0,0}= S_{0,0} \oplus (02 \cdot S_{0,0})(04 \cdot S_{0,0})$

c.此处的矩阵乘法与一般意义上矩阵的乘法有所不同，各个值在相加时使用的是模 2^8 加法（异或运算）。

下面举一个例子，假设某一列的值如下图，运算过程如下：

02	03	01	01	C9	=	DB
01	02	03	01	6E		37
01	01	02	03	46		94
03	01	01	02	A6		ED

$$S'_{0,0} = (02 \bullet C9) \oplus (03 \bullet 6E) \oplus (01 \bullet 46) \oplus (01 \bullet A6)$$

其中：

$$02 \bullet C9 = 02 \bullet 11001001_B = 10010010_B \oplus 00011011_B = 10001001_B$$

$$03 \bullet 6E = (01 \oplus 02) \bullet 6E = 01101110_B \oplus 11011100_B = 10110010_B$$

$$01 \bullet 46 = 01000110_B$$

$$01 \bullet A6 = 10100110_B$$

则：

$$S'_{0,0} = 10001001_B \oplus 10110010_B \oplus 01000110_B \oplus 10100110_B$$

$$= 11011011_B = DB$$

在计算 02 与 C9 的乘积时，由于 C9 对应最左边的比特为 1，因此需要将 C9 左移一位后的值与(0001 1011)求异或。同理可以求出另外几个值。

2.逆向列混淆

逆向列混淆的原理图如下：

由于：

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \\ 00 & 00 & 00 & 01 \end{bmatrix}$$

说明两个矩阵互逆，经过一次逆向列混淆后即可恢复原文。

3.轮密钥加

这个操作相对简单，其依据的原理是“任何数和自身的异或结果为0”。加密过程中，每轮的输入与轮子密钥异或一次；因此，解密时再异或上该轮的轮子密钥即可恢复。

五、断点续传

5.1.功能简介

持断点续传，以适应于网络在发生故障并恢复之后备份作业的不中断，当网络恢复正常后继续备份操作，备份从断点继续，而不是从头开始。

5.2.技术实现

5.2.1.虚拟化定时备份：

虚拟化备份通过调用 API 进行备份，当网络出现故障以后，API 会返回网络错误等错误码，若流程检测到网络错误便开始循环检测网络的连通性，若在一段时间之内网络恢复（例如 30 分钟），则恢复数据传输，否则退出并提示网络错误。

5.2.2.数据库/文件 定时备份：

这些模块在备份发生，由服务端调用数据流传输接口，当网络出现故障以后，数据流传输接口会返回对应的错误码，备份流程检测到网络错误便开始循环检测网络的连通性，若在一段时间之内网络恢复（例如 30 分钟），则恢复数据传输，否则退出并提示网络错误。

5.2.3.操作系统/卷 定时备份：

这些模块在备份发生时，是由客户端向服务端发送备份数据，若数据发送失败或接收 ACK 失败，则开始循环检测网络的连通性，若在一段时间之内网络恢复（例如 30 分钟），则恢复数据传输，否则退出并提示网络错误。

5.2.4.操作系统/卷 实时备份

这些模块在备份发生时，是由客户端向服务端发送备份数据，若数据发送失败或接收 ACK 失败，则开始循环检测网络的连通性，若网络恢复则恢复数据传输，否则无限重试重连，若在重连的过程中，检测到缓存溢出，则客户端停止整个备份流程，等待服务端网络恢复以后，会自动同步任务信息，停止当前任务状态。

5.3.SpeedX 备份加速

5.3.1.功能简介

通过在介质服务器上的缓存盘，来加速备份与恢复的处理，可以更快的完成数据的备份与恢复。

5.3.2.技术实现

在介质服务器中放置一块 SSD，将其作为备份加速的缓存盘。当备份发生时，先将目标数据存储到缓存盘中，缓存盘相较于机械硬盘本身就有更好的 IOPS，因此可以最大程度上减少备份的时间开销。

但是因为 SSD 的成本及容量限制，所以缓存盘只能用作临时加速，在备份完成之后，系统可以通过配置的迁移策略或者手动将数据迁移到大容量的备份仓库中。

当备份数据需要恢复时，首先在缓存盘中查找是否存在，若命中，则直接恢复，否则从备份仓库中读取并恢复。

迁移策略：

（1）当备份完成以后，自动迁移到备份仓库中。
（2）当缓存盘空间超过 n%以后，自动将最旧的备份点迁移到备份仓库中。
（3）手动迁移。

5.4.虚拟化平台

5.4.1.功能简介

系统内置基于 KVM 内核定制的虚拟化平台。

（1）支持定制虚拟化存储池，网络池
（2）支持通过模板、ISO 镜像、PXE 的方式创建虚拟机。
（3）支持模板管理功能，可上传和使用虚拟机模板。
（4）支持通过 VNC 的方式可视化操作虚拟机。
（5）支持编辑虚拟机，包括配置 CPU、内存、硬盘、网络、引导模式等信息。

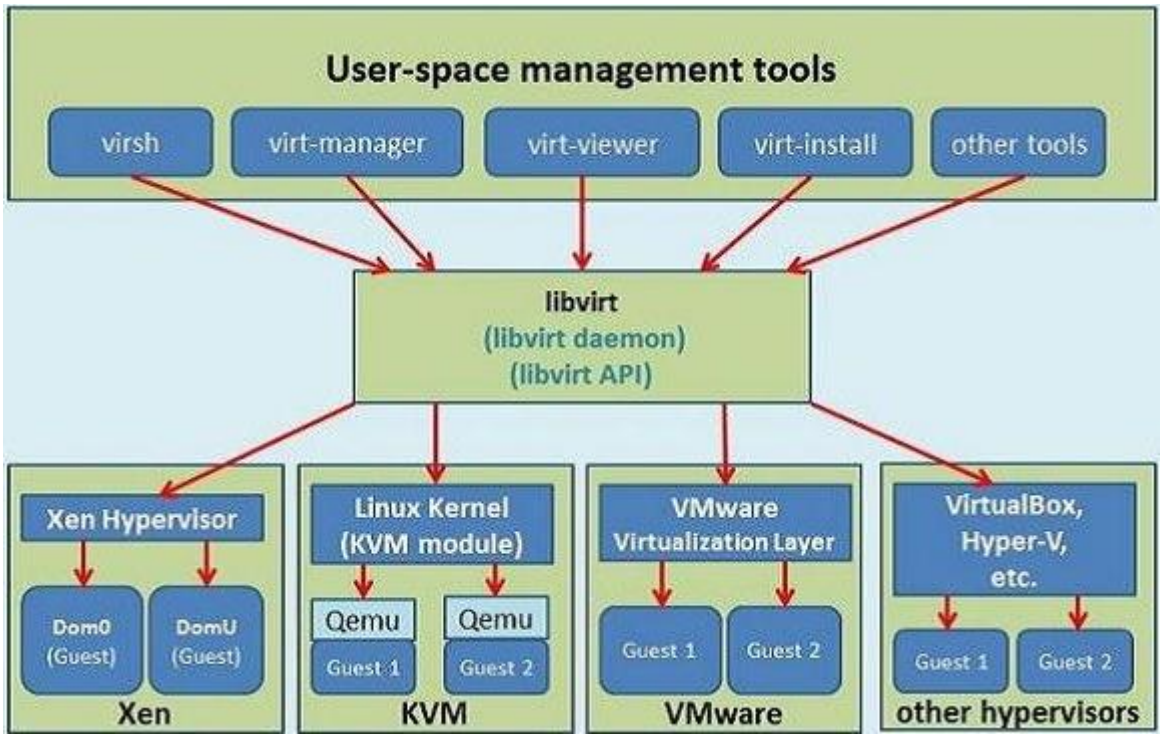
5.4.2.技术实现

虚拟化平台是使用 libvirt 来管理和操作 kvm 虚拟机。

（1）libvirt 介绍：

libvirt 是目前使用最为广泛的针对 KVM 虚拟机进行管理的工具和 API。Libvirtd 是一个 daemon 进程，可以被本地和远程的 virsh(命令行工具)调用，Libvirtd 通过调用 qemu-kvm 操作管理虚拟机。libvirt 由应用程序编程接口(API) 库、一个守护进程 (libvirtd)，和默认命令行实用工具 (virsh)等部分组成。

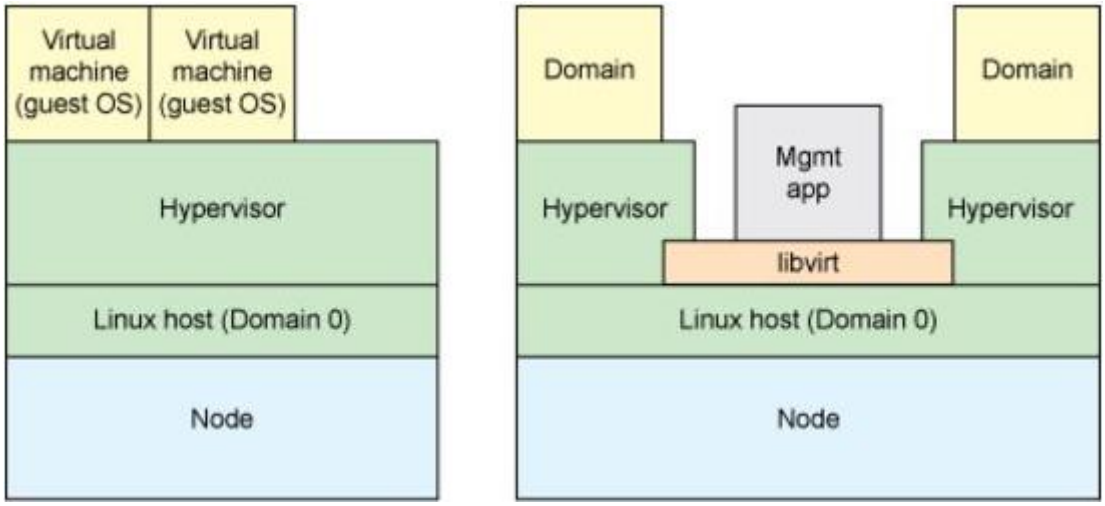
那么，libvirt 支持什么？ 一张图了解 libvirt 支持的 Hypervisor 和哪些管理工具支持 libvirt。



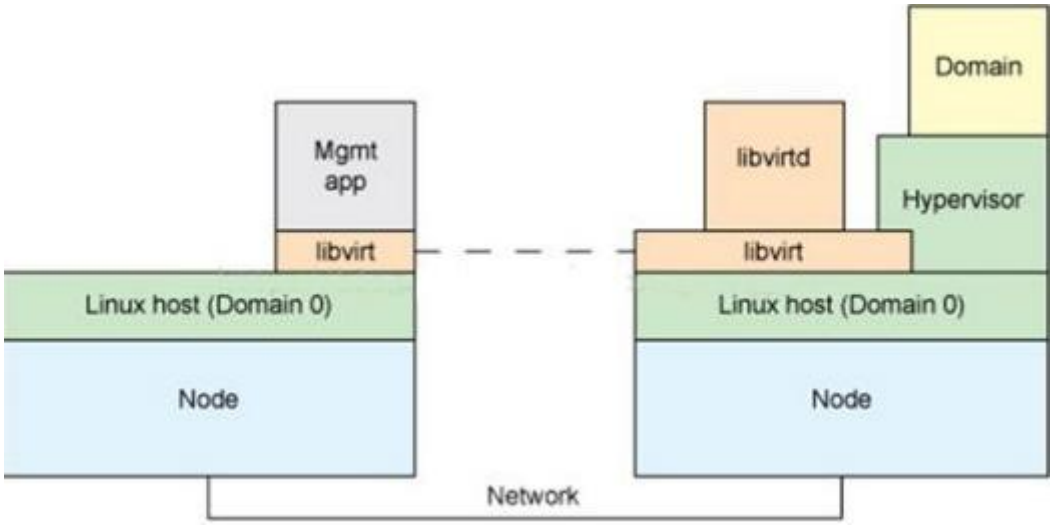
对上层 libvirt 是个 C 语言库，但同时它也提供了其他编程语言的封装，这些语言使用 libvirt 封装好的 libvirtmod。2014 年以前的 libvirt 代码中，包含 python 接口，在 2014 年某个版本以后，将 python 单独提出去了，不再和 libvirt 代码使用同一个 git 库，独立成为 libvirt-python 的 Git 库。Libvirt 目前也已经对 Ruby、Java 语言，Perl 和 OCaml 实施了绑定，支持最流行的系统编程语言(C 和 C++)、多种脚本语言。

（2）Libvirt 的架构：

libvirt 架构如下图，为支持各种 Hypervisor 的可扩展性，libvirt 实施一种基于驱动程序的架构，该架构允许一种通用的 API 以通用方式为不同的 Hyperviosr 供服务。右图展示了 libvirt API 与相关驱动程序的层次结构(Hypervisor 和 Domain 在同一个节点)。



Hypervisor 和 Domain 位于不同节点上时，管理应用程序通过一种通用协议从本地 libvirt 连接到远程 libvirtd，通过运行于远程节点的 libvirtd 的特殊守护进程来实现管理。libvirtd 提供从远程应用程序访问本地 Domain 的方式。



（3）Libvirt 主要支持的功能：

1.虚拟机管理：

包括不同的领域生命周期操作，比如：启动、停止、暂停、保存、恢复和迁移。支持多种设备类型的热插拔操作，包括：磁盘、网卡、内存和 CPU。

2.远程机器支持：

只要机器上运行了 libvirt daemon，包括远程机器，所有的 libvirt 功能就都可以访问和使用。支持多种网络远程传输，使用最简单的 SSH，不需要额外配置工作。

3.存储管理：

任何运行了 libvirt daemon 的主机都可以用来管理不同类型的存储：创建不同格式的文件镜像（qcow2、vmdk、raw 等）、挂接 NFS 共享、列出现有的 LVM 卷组、创建新的 LVM 卷组和逻辑卷、对未处理过的磁盘设备分区、挂接 iSCSI

共享等。因为 libvirt 可以远程工作，所有这些都可以通过远程主机使用。

4.网络接口管理：

任何运行了 libvirt daemon 的主机都可以用来管理物理和逻辑的网络接口。虚拟 NAT 和基于路由的网络：任何运行了 libvirt daemon 的主机都可以用来管理和创建虚拟网络。

（4）虚拟化平台具体实现：

1.配置磁盘 xml 字符串，通过 virStorageVolCreateXML 函数创建指定格式的虚拟磁盘。
2.配置虚拟机 xml 字符串，通过 virDomainDefineXML 函数创建虚拟机，虚拟机可以指定不同的引导模式，实现通过 ISO、磁盘和 PXE 来创建虚拟机。
3.通过 virDomainUndefineFlags 函数来删除虚拟机。
4.通过修改虚拟机 xml 字符串，来修改虚拟不同的磁盘和网卡模式。
5.虚拟机电源控制：使用 virDomainCreate 启动虚拟机，virDomainShutdown 关闭虚拟机，virDomainSuspend 挂起虚拟机，virDomainReboot 重启虚拟机。
6.通过 vncdisplay 命令获取 vnc 监听的端口，动态配置 novnc 信息，通过 novnc 在 web 上远程访问虚拟机。

5.5.RESETful API

5.5.1.功能简介

系统采用的目前最为流行的互联网软件架构 RESTFull，利用其清晰的结构、符合标准、易于理解、扩展方便的特性将系统中所涉及的所有功能接口化。通过开发和使用 restfull api 不仅降低开发的复杂性、提高系统的可伸缩性，更方便第三方软件对本系统功能调用，以达到与第三方系统的完美契合。

5.5.2.技术实现

RestfulAPI 是由后台(SERVER 端)来提供接口，前端来调用。前端调用 API 向后台发起 HTTP 请求，后台响应请求将处理结果反馈给前端。也就是说 Restful 是典型的基于 HTTP 的协议。RESTful API 具有如下特性：
（1）Resource 资源，首先是弄清楚资源的概念。资源就是网络上的一个实体、一段文本、一张图片或者一首歌曲。资源总是要通过一种载体来反应它的内容。文本可以用 TXT，也可以用 HTML 或者 XML、图片可以用 JPG 格式或者 PNG 格式，JSON 是现在最常用的资源表现形式；
（2）统一接口。Restful 风格的数据元操作 CRUD(create,read,update,delete)分别对应 HTTP 方法：GET 用来获取资源，POST 用来新建资源(也可以用于更新资源)，PUT 用来更新资源，DELETE 用来删除资源，这样就统一了数据操作的接口；
（3）无状态。所谓无状态即所有的资源都可以 URI 定位，而且这个定位与其他资源无关，也不会因为其他资源的变化而变化；
（4）请求的 URL 需包含版本号；如：GET:http://192.168.2.100/dse/v1/user

（5）规范返回数据。为了保障前后端的数据交互的顺畅，系统采用规范 json 数据格式封装进行封装；如：

```
{
    "result_code": 0,
    "result_message:": "success",
    "data":{
        "repo_uuid": "123"
    }
}
```

（6）状态码：使用 HTTP 状态码，HTTP 标准中提供了 70 多个状态来描述返回值，但是我们并不需要用到其中的全部，只需要了解其中一些比较常用的就可以了；

- ✧200 - OK - 一切正常 201 - OK - 新资源已经被创建 204 - OK - 资源删除成功；
- ✧304 - 没有变化，客户端可以使用缓存数据；
- ✧400 - Bad Request - 调用不合法，确切的错误应该在 error payload 中描述 401 - 未认证，调用需要用户通过认证 403 - 不允许的，服务端正常解析和请求，但是调用被回绝或者不被允许 404 - 未找到，指定的资源不存在 422 - 不可指定的请求体 - 只有服务器不能处理实体时使用，比如图像不能被格式化，或者重要字段丢失；
- ✧500 - Internal Server Error - 标准服务端错误，API 开发人员应该尽量避开这种错误；

通过利用 restfull 的相关特性设计本系统的接口，并通过 restful API 带 token 的方式验证请求的合法性，避免了非法请求对系统带来的潜在威胁。相关处理流程如示意图：