MySQL 主从复制的原理：数据复制实际就是 Slave 从 Master 获得 Binary Log 文件，然后再本地镜像的执行日志中记录的操作，由于主从复制的过程是异步的，因此 Slave 和 Master 之间的数据可能存在延迟的现象，此时只能保持数据的最终一致性。

Binlog dump 线程操作的文件是 bin-log 日志文件，并且实现主从复制在主服务器上主要依靠 bin-log 日志文件。MySQL 的 Replication 是一个多 MySQL 数据库做主从同步的方案，特点是异步复制，广泛用在各种对，MySQL 有更高性能、更高可靠性要求的场合。与之对应的另一个同步技术是 MySQL Cluster 配置比较复杂。

在 Master 和 Slave 之间实现整个复制过程主要由三个线程来完成：（1）Slave SQL thread 线程，在 slave 端。（2）Slave I/O thread 线程，在 slave 端。（3）Binlog dump thread 线程，在，master 端。

如果一台主服务器配置两台从服务器，则主服务器上就存在两个 Binlog dump 线程，而每个从服务器鸽子有两个线程。要实现 MySQL 的 Replication，首先必须打开 master 端的 Binlog 日志功能，否则无法实现 MySQL 的主从复制，因为 MySQL 的整个主从复制过程实际上是 slave 端从 master 端获取 Binlog 日志，然后再在自己完全顺序执行该日志中记录的各种 SQL 操作。

MySQL 主从复制基本交互过程：（1）Slave 端的 IO 线程连接上 Master 端，并请求从指定的 Binlog 日志文件的指定 POS 节点位置开始复制之后的日志内容。（2）Master 端在接收到来自 Slave 端的 IO 线程请求后，通知负责复制进程的 IO 线程，根据 Slave 端 IO 线程的请求信息，读取指定 Binlog 日志指定 POS 节点位置之后的日志信息，然后返回给 Slave 端的 IO 线程。该返回信息中除了 Binlog 日志所包含的信息之外，还包含本次返回的信息在 Master 端的 Binlog 文件名以及在该 Binlog 日志中的 POS 节点位置。（3）Slave 端的 IO 线程在接收到 Master 端 IO 返回的信息后，将接收到的 Binlog 日志内容依次写入到 Slave 端的 relaylog 文件的最末端，并将读取到的 Master 端的 Binlog 文件名和 POS 节点位置记录到 Master-info 文件中，在下一次读取的时候能够清楚的告诉 Master 需要从哪个 Binlog 文件的哪个 POS 节点位置开始。（4）Slave 端的 SQL 线程在检测到 relaylog 文件中心增加的内容后，会马上解析 log 文件中的内容，然后还原成在 Master 端执行的那些 SQL 语句，并在自身按顺序依次执行这些 SQL 语句，实际上就是在 Master 端和 Slave 端执行了同样的 SQL 语句，保证了 Master 端和 Slave 端的数据完全一样。

MySQL 支持的复制类型及优缺点：（1）基于语句的复制，在主服务器上执行的 SQL 语句，在从服务器上执行同样的语句。MySQL 默认采用基于语句的复制，效率比较高。一旦发现不能精确复制时，会自动选择基于行的复制。（2）基于行的复制，把改变的内容复制过去，而不是把命令在从服务器上执行一遍。（3）混合类型的复制。

Statement-Based 优点：（1）Binlog 日志包含了描述数据库操作的事件，这些事件包含的情况只是对数据库进行改变的操作，如 insert、update、create、delete 等操作，而 select、desc

等类似的操作并不会去记录，并且它记录的是语句，相对于 Row-Based 来说占用的存储空间更小。（2）Binlog 日志文件记录了所有的改变数据库的语句，所以此文件可以作为以后数据库的审核依据。缺点：（1）不安全，并不是所有的改变数据的语句都会被记录复制。任何的非确定性的行为是很难被记录复制的。（2）对于没有索引条件的 update 语句，必须锁定更多的数据，降低了数据库的性能。

Row-Based 优点：（1）所有的改变都会被复制，这个是最安全的复制方法。（2）对于 update、insert、select 等语句锁定更少的行。缺点：（1）使用不方便，不能通过 Binlog 日志文件查看什么语句被执行了，也不知道从服务器接收到什么语句，只能看到什么数据改变了。（2）因为记录的是数据，所以 Binlog 日志文件占用的存储空间比 Statement-Based 大。

MySQL Replication 就是从服务器拉取主服务器上的二进制日志文件，然后再将日志文件解析成相应的 SQL 语句在从服务器上重新执行一遍主服务器的操作，通过这种方式来保证数据的一致性。MySQL Replication 是一个异步复制的过程，是一个 MySQL instance 复制到另一个 MySQL instance。

MySQL 主从复制的常用拓扑结构：（1）一主一从是最基础的复制结构，用来分担之前单台数据库服务器的压力，可以进行读写分离。（2）一主多从，一台 Slave 承受不住读请求压力时，可以添加多台，进行负载均衡，分散读的压力。还可以对多台 Slave 进行分工，服务于不同的系统，一部分负责网站前台的读请求，一部分负责后台统计系统的请求等。不同系统的查询需求不同，对 Slave 分工后，可以创建不同的索引，更好的服务于目标系统。（3）双主复制，Master 存在下线的可能，例如故障或者维护，需要把 Slave 切换为 Master。在原来的 Master 恢复可用后，由于其数据已经不是最新的了，不能再作为主节点，需要作为 Slave 添加进来。需要对其重新搭建复制环境，需要耗费一定的工作量。双主复制可以解决这个问题，互相将对方作为自己的 Master，自己作为对方的 Slave 进行复制，对外来说，还是一个主节点，一个从节点。（4）级联复制，当直接从属于 Master 的 Slave 过多时，连到 Master 的 Slave 的 IO 线程比较多，对 Master 的压力比较大。级联结构就是通过减少直接从属于 Master 的 Slave 数量，减少 Master 的压力，分散复制请求，从而提高整体的复制效率。（5）双主级联，级联复制结构解决了 Slave 多过导致的性能瓶颈问题，但是存在单主结构中切换主时的维护问题。复制的最大问题是数据延时，选择复制结构时需要根据自己的具体情况，并评估好目标结构的延时对系统的影响。

关系型数据库：A 特性：（1）关系型数据库是指采用了关系模型来组织数据的数据库。（2）关系型数据库的最大特点是事务的一致性。（3）关系模型指二维表格关系，一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织。B 优点：（1）容易理解，二维表结构是比较贴近逻辑世界的一个概念，关系模型相对网状、层次等其他模型来说更容易理解。（2）使用方便，通用的 SQL 语言使得操作关系型数据库非常方便。（3）易于维护，丰富的完整性降低了数据冗余和数据不一致的概率，有实体完整性、参照完整性和用户定义

的完整性。（4）支持 SQL，可用于复杂的查询。C 缺点：（1）为了维护一致性所付出的巨大代价就是读写性能比较差。（2）固定的表结构。（3）高并发读写需求。（4）海量数据的高效率读写。

非关系型数据库：A 特性：（1）使用键值对存储数据。（2）一般不具有 ACID 特性。（3）分布式。（4）非关系型数据库严格意义上不是一种数据库，应该是一种数据结构化存储方法的集合。B 优点：（1）不需要经过 SQL 层的解析，读写性能比较高。（2）基于键值对，数据没有耦合性，容易扩展。（3）存储数据的格式，NoSQL 的存储格式是 key，value 形式、文档形式、图片形式等。关系型数据库只支持基础类型。C 缺点：（1）不提供 SQL 支持，学习和使用成本比较高。（2）没有事务处理，BI 和报表不能很好的支持。

KMP 算法是一种改进的字符串匹配算法。是 D.E.Knuth，J.H.Morris 和 V.R.Pratt。KMP 算法的关键是利用匹配失败后的信息，尽量减少模式串和主串的匹配次数以达到快速匹配的目的。时间复杂度是 O(m+n)。在 KMP 算法中，对于每一个模式串都会先计算出模式串的内部匹配信息，在匹配失败时最大的移动模式串，以减少匹配次数。

程序：存储在磁盘上的二进制文件，是可以被内核所执行的代码。进程：当一个用户启动一个程序，将会在内存中开启一块空间，创建了一个进程，一个进程包含一个独一无二的 PID，和执行者的权限属性参数，以及程序所需代码和相关资料。进程是系统分配资源的最小单位。一个进程可以有多个子进程，子进程的相关权限会延用父进程的相关权限。线程：每个进程包含一个或者多个线程，线程是进程内的活动单元，是负责执行代码和管理进程运行状态的抽象。线程是独立运行和调度的基本单位。

子进程拥有一个指明其父进程 PID 的 PPID，子进程可以继承父进程的环境变量和权限参数，Linux 系统中有进程的层次结构-进程树。进程树的根是第一个进程，即 init 进程。进程调用的过程：fork 和 exec。一个进程生成一个子进程的过程是，系统首先会复制一份父进程，生成一个暂存进程，这个暂存进程和父进程的 PID 不同，而且拥有一个 PPID，这时再去执行这个暂存进程，让他加载实际要运行的程序，最终成为一个子进程存在。进程通信的方式：管道、信号量、共享内存、消息队列、快速用户控件互斥。服务（daemon）是常驻内存的进程，通常服务会在开机时通过 init.d 中的一段脚本被启动。

一个进程包括代码、数据和分配给进程的资源、fork 函数通过系统调用创建一个与原来进程几乎完全相同的进程，两个进程可以做相同的事情。如果初始参数或者传入的变量不同，两个进程也可以做不同的事情。一个进程调用 fork 函数后，系统先给新的进程分配资源，然后把原来的进程的所有值复制到新的进程中，只有少数值和原来的进程的值不同。Fork 函数被调用一次，却能够返回两次，它可能有三种不同的返回值。（1）在父进程中，fork 返回新创建子进程的进程 ID。（2）在子进程中，fork 返回 0。（3）如果出现错误，fork 返回一个负值。

几种关系：（1）父进程通过 fork 产生子进程。（2）孤儿进程：当子进程没有结束时，父

进程异常退出，原本需要由父进程进行处理的子进程变成了孤儿进程，init 系统进程会把这些进程领养，避免称为孤儿。（3）僵尸进程：当子进程结束时，会在内存中保留一部分数据结构等待父进程显式结束，如果父进程没有执行结束操作，则会导致子进程的剩余结构无法被释放，占用空间造成严重后果。（4）守护进程：守护进程用于监控其他进程，当发现大量僵尸进程时，会找到他们的父节点并杀死，同时让 init 认养他们以释放这些空间。僵尸进程时有害的，孤儿进程由于内核的认养不会造成危害。

进程组就是由一个或者多个为了实现作业控制而相互关联的进程组成，每个进程都属于某个进程组。一个进程组的 ID 是进程组首进程的 ID，如果一个进程组只有一个进程，则无区别，进程组的意义在于信号可以发送给进程组中的所有进程，可以实现对多个进程的同时操作。

守护进程运行在后台，不与任何控制终端关联，通常在系统启动时，通过 init 脚本被调用而开始运行。在 Linux 系统中，守护进程和服务没有区别。对于一个守护进程的两个基本要求：（1）必须作为 init 进程的子进程运行。（2）不和任何控制终端交互。

硬连接指的是不同的文件名指向同一个 inode 节点。软连接是指一个文件的 inode 节点不存数据，而是存储着另一个文件的绝对路径，访问文件内容时实际上是去访问对应路径下的文件 inode，即文件发生改动或者移动都会导致软连接失效。

线程是执行操作系统调度的最小执行单元，同一个进程内的所有线程共享该进程的内存空间。虚拟内存系统为每个进程分配独立的内存空间。

内核级线程模型，每个内核线程直接转换成用户空间的线程，即内核线程：用户空间线程=1:1。用户级线程，一个保护了 n 个线程的用户进程只会映射到一个内核进程，即 n:1。

文件描述符在 Linux 内核中，文件是用一个整数来表示的，称为文件描述符。当打开一个现有文件或创建一个新文件时，内核向进程返回一个文件描述符。当读或者写一个文件时，使用 open 或者 create 返回的文件描述符表示该文件，将其作为参数传递给 read 或者 write 函数。