32 (1)加餐 练习题详解（六）

今天我会带你把《模块六：文件系统》中涉及的课后练习题，逐一讲解，并给出每个课时练习题的解题思路和答案。

练习题详解

29 | Linux下各个目录有什么作用?

【问题】socket 文件都存在哪里?

【解析】socket 没有实体文件，只有 inode，所以 socket 是没有名字的文件。

你可以在 /proc/net/tcp 目录下找到所有的 TCP 连接，在 /proc/[pid]/fd 下也可以找到这些 socket 文件，都是数字代号，数字就是 socket 文件的 fd，如下图所示：

```
ramroll@u1:net$ cat /proc/net/tcp
  sl  local_address rem_address   st tx_queue rx_queue tr tm->when retrnsmt
  uid  timeout inode
   0: 00000000:0016 00000000:0000 0A 00000000:00000000 00:00000000 00000000
    0        0 24951 1 0000000000000000 100 0 0 10 0
   1: 0100007F:0277 00000000:0000 0A 00000000:00000000 00:00000000 00000000
    0        0 274437 1 0000000000000000 100 0 0 10 0
   2: 0100007F:9103 00000000:0000 0A 00000000:00000000 00:00000000 00000000
    0        0 48369 1 0000000000000000 100 0 0 10 0
   3: 3500007F:0035 00000000:0000 0A 00000000:00000000 00:00000000 00000000
 101        0 12865 1 0000000000000000 100 0 0 10 0
```

@拉勾教育

你也可以用 `lsof -i -a -p [pid` 查找某个进程的 socket 使用情况。下面结果和你用 `ls /proc/[pid]/fd` 看到的 fd 是一致的，如下图所示：

```
ramroll@u1:net$ lsof -i -a -p 2377
COMMAND  PID    USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
java    2377 ramroll   99u  IPv6 317130      0t0  TCP localhost:46261 (LISTE
N)
java    2377 ramroll  100u  IPv6 314877      0t0  TCP localhost:46261->local
host:57322 (ESTABLISHED)
java    2377 ramroll  108u  IPv6 382374      0t0  TCP localhost:39671 (LISTE
N)
java    2377 ramroll  110u  IPv6 384666      0t0  TCP localhost:39671->local
host:56794 (ESTABLISHED)
```

```
java    2377 ramroll  131u  IPv6 387682      0t0  TCP localhost:45115->local
host:35500 (ESTABLISHED)
```
@拉勾教育

30 | 文件系统的底层实现：FAT、NTFS 和 Ext3 有什么区别?

【问题】思考日志文件系统的数据冗余如何处理?

【解析】日志系统产生冗余几乎是必然发生的。 只要发生了修改、删除，肯定就会有数据冗余。日志系统通常不会主动压缩，但是日志文件系统通常会对磁盘碎片进行整理，这种机制和内存的管理非常相似。

首先我们把这个磁盘切割成很多等大的小块，大文件可能需要分配多个小块，多个小文件共用一个小块。而当很多文件被删除之后，磁盘中的小块会产生碎片，文件系统会进行碎片整理，比如把多个有很多碎片的区域拷贝到一起，使存储空间更加紧凑。

回到正题，最终的答案就是不压缩、不处理冗余，空间换时间，提升写入速度。

31 | 数据库文件系统实例：MySQL 中 B 树和 B+ 树有什么区别?

【问题】按照应该尽量减少磁盘读写操作的原则，是不是哈希表的索引更有优势?

【解析】哈希表是一种稀疏的离散结构，通常使用键查找值。给定一个键，哈希表会通过数学计算的方式找到值的内存地址。因此，从这个角度去分析，哈希表的查询速度非常快。单独查找某一个数据速度超过了 B+ 树（比如根据姓名查找用户）。因此，包括 MySQL 在内的很多数据库，在支持 B+ 树索引的同时，也支持哈希表索引。

这两种索引最大的区别是：B+ 树是对范围的划分，其中的数据还保持着连续性；而哈希表是一种离散的查询结构，数据已经分散到不同的空间中去了。所以当数据要进行**范围查找**时，比如查找某个区间内的订单，或者进行聚合运算，这个时候哈希表的性能就非常低了。

哈希表有一个设计约束，如果我们用了 m 个桶（Bucket，比如链表）去存储哈希表中的数据，再假设总共需要存储 N 个数据。那么平均查询次数 k = N/m。为了让 k 不会太大，当数据增长到一定规模时，哈希表需要增加桶的数目，这个时候就需要重新计算所有节点的哈希值（重新分配所有节点属于哪个桶）。

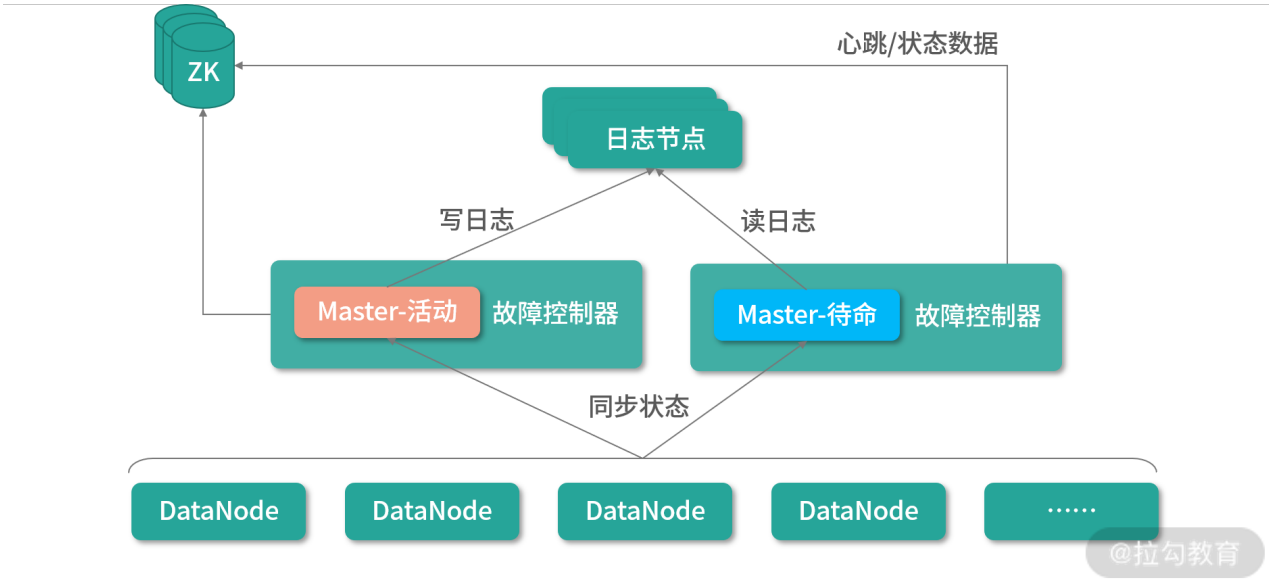
综上，对于大部分的操作 B+ 树都有较好的性能，比如说 >,<, =,BETWEEN，LIKE 等，哈希表只能用于等于的情况。

32 | HDFS 介绍：分布式文件系统是怎么回事?

【问题】Master 节点如果宕机了，影响有多大，如何恢复?

【解析】在早期的设计中，Master 是一个单点（Single Point），如果发生故障，系统就会停止运转，这就是所谓的单点故障（Single Point of Failure）。由此带来的后果会非常严重。发生故障后，虽然我们可以设置第二节点不断备份还原点，通过还原点加快系统恢复的速度，但是在数据的恢复期间，整个系统是不可用的。

在一个高可用的设计当中，我们不希望发生任何的单点故障（SPoF），因此所有的节点都至少有两份。于是在 Hadoop 后来的设计当中，增加了一种主从结构。



如上图所示，我们同时维护两个 Master 节点（在 Hadoop 中称为 NameNode，NN）——一个活动（Active）的 NN 节点，一个待命（StandBy）的 NN 节点。

为了保证在系统出现故障的时候，可以迅速切换节点，我们需要一个故障控制单元。因为是分布式的设计，控制单元在每个 NN 中都必须有一个，这个单元可以考虑 NN 节点进程中的一个线程。控制单元不断地检测节点的状态，并且不断探测其他 NN 节点的状态。一旦检测到故障，控制单元随时准备切换节点。

一方面，因为我们不能信任任何的 NN 节点不出现故障，所以不能将节点的状态存在任何一个 NN 节点中。并且节点的状态也不适合存在数据节点中，因为大数据集群的数据节点实时性不够，它是用来存储大文件的。因此，可以考虑将节点的状态放入一个第三方的存储当中，通常就是 ZooKeeper。

另一方面，因为活动 NN 节点和待命 NN 节点数据需要完全一致，所以数据节点也会把自己的状态同时发送给活动节点和待命节点（比如命名空间变动等）。最后客户端会把请求发送给活动节点，因此活动节点会产生操作日志。不可以把活动节点的操作日志直接发送给待命节点，是因为我们不确定待命节点是否可用。

而且，为了保证日志数据不丢失，它们应该存储至少 3 份。即使其中一份数据发生损坏，也可以通过对比半数以上的节点（2 个）恢复数据。因此，这里需要设计专门的日志节点

（Journal Node）存储日志。至少需要 3 个日志节点，而且必须是奇数。活动节点将自己的日志发送给日志节点，待命节点则从日志节点中读取日志，同步自己的状态。

我们再来回顾一下这个高可用的设计。**为了保证可用性，我们增加了备用节点待命，随时替代活动节点**。为了达成这个目标。有 3 类数据需要同步。

- **数据节点同步给主节点的日志**。这类数据由数据节点同时同步给活动、待命节点。
- **活动节点同步给待命节点的操作记录**。这类数据由活动节点同步给日志节点，再由日志节点同步给待命节点。日志又至少有 3 态机器的集群保管，每个上放一个日志节点。
- **记录节点本身状态的数据**（比如节点有没有心跳）。这类数据存储在分布式应用协作引擎上，比如 ZooKeeper。

有了这样的设计，当活动节点发生故障的时候，只需要迅速切换节点即可修复故障。

总结

这个模块我们对文件系统进行了系统的学习，下面我来总结一下文件系统的几块核心要点。

- 理解虚拟文件系统的设计，理解在一个目录树结构当中，可以拥有不同的文件系统——一切皆文件的设计。基于这种结构，设备、磁盘、分布式文件系、网络请求都可以是文件。
- 将空间分块管理是一种高效的常规手段。方便分配、方便回收、方便整理——除了文件系统，内存管理和分布式文件系统也会用到这种手段。
- 日志文件系统的设计是重中之重，日志文件系统通过空间换时间，牺牲少量的读取性能，提升整体的写入效率。除了单机文件系统，这种设计在分布式文件系统和很多数据库当中也都存在。
- 分层架构：将数据库系统、分布式文件系搭建在单机文件管理之上——知识是死的、思路是活的。希望你能将这部分知识运用到日常开发中，提升自己系统的性能。

上一页 下一页