第 5 章

键值数据库实例： Redis 与 DynamoDB

5.1 Redis

5.1.1 Redis 介绍

1. Redis 简介

Redis(Remote Dictionary Server,远程字典服务)是一个由 Salvatore Sanfilippo 编写的键值存储系统,是跨平台的非关系数据库。

Redis 是一个开源的使用 ANSI C 语言编写、遵守 BSD(Berkeley Software Distribution,伯克利软件套件)协议、支持网络、可基于内存也可持久化的日志型键值数据库,并提供多种语言的 API。

目前,Redis 是最受欢迎的 NoSQL 数据库之一。它通常被称为数据结构服务器,因为值可以是字符串(String)、哈希(Hash)、列表(List)、集合(Set)和有序集合(Sorted Set)等类型。

2. Redis 的特点

(1) 读写性能极高。Redis 数据库读写速度非常快,因为把数据都读取到内存当中操作,而且 Redis 是用 C 语言编写的,是最“接近”操作系统的语言,所以执行速度相对较快。读取速度可高达 110 000 次/s,写速度高达 81 000 次/s。

(2) 支持数据持久化。虽然 Redis 数据的读取都存在内存当中,但是最终它支持数据持久化到磁盘当中。

(3) 数据结构丰富。Redis 支持二进制的字符串、哈希、列表、集合和有序集合数据类型操作。一方面满足存储各种数据结构体的需要;另一方面数据类型少,使得规则就少,需要的判断和逻辑就少,这样读写的速度就更快。

(4) 操作具有原子性,支持事务。Redis 的所有操作都是原子性的,同时 Redis 还支持对几个操作合并后的原子性执行。

(5) 支持数据备份。Redis 支持主从复制,主机会自动将数据同步到从机,读写可分离。

3. Redis 为什么这么快

(1) 完全基于内存,绝大部分请求是纯粹的内存操作,非常快速。数据存在内存中,类似于 HashMap,HashMap 的优势就是查找和操作的时间复杂度都是 $O(1)$。

(2) 数据结构简单,对数据操作也简单,Redis 中的数据结构是专门进行设计的。

(3) 采用单线程,避免了不必要的上下文切换和竞争条件,也不存在多进程或者多线程导致的切换而消耗 CPU,不用去考虑各种锁的问题,不存在加锁、释放锁操作,没有因为可能出现死锁而导致的性能消耗。

(4) 使用多路 I/O 复用模型、非阻塞 I/O,这里“多路”指的是多个网络连接,“复用”指的是复用同一个线程。

(5) VM 机制。VM(虚拟内存)机制就是暂时把不经常访问的数据(冷数据)从内存交换到磁盘中,从而腾出宝贵的内存空间用于其他需要访问的数据(热数据)。通过 VM 功能可以实现冷热数据分离,使热数据仍在内存中、冷数据保存到磁盘。这样就可以避免因为内存不足而造成访问速度下降的问题。

4. Redis 的数据类型

1) 字符串

字符串是 Redis 的最基本的数据类型。字符串类型是二进制安全的,意思是 Redis 的字符串可以包含任何数据(例如,图片或者序列化的对象)。一个 Redis 中字符串最多 512MB。

Redis 中的普通字符串采用原始编码(Raw Encoding)方式,该编码方式会动态扩容,并通过提前预分配冗余空间,来减少内存频繁分配的开销。

在字符串长度小于 1MB 时,按所需长度的 2 倍来分配; 超过 1MB,则按照每次额外增加 1MB 的容量来预分配。

Redis 中的数字也为字符串类型,但编码方式跟普通字符串不同,数字采用整型编码,字符串内容直接设为整数值的二进制字节序列。

在存储普通字符串、序列化对象,以及计数器等场景时,都可以使用 Redis 的字符串类型,字符串数据类型对应使用的指令包括 set、get、mset、incr、decr 等。

2) 哈希

Redis 中哈希是一个字符串类型的字段(Field)和值的映射表。哈希特别适合用于存储对象。

Redis 的哈希类型其实就是一个缩减版的 Redis。它存储的是键值对,将多个键值对存储到一个 Redis 键里面。

Redis 内部在实现哈希数据类型时使用了两种数据结构。一种是压缩列表(ziplist),另一种是哈希表。

压缩列表是一个经过特殊编码的双向链表,它不存储指向上一个链表节点和指向下一个链表节点的指针,而是存储上一个节点长度和当前节点长度,通过牺牲部分读写性能,来

换取高效的内存空间利用率，是一种时间换空间的思想。

当存储的数据量较小时，Redis 才使用压缩列表来实现字典类型。具体需要满足以下两个条件。

(1) 字典中保存的键和值的大小都要小于 64 字节。

(2) 字典中键值对的个数要小于 512。

当不能同时满足上面两个条件时，Redis 就使用哈希表来实现哈希类型。

哈希类型比较适合保存结构体信息的，不同于字符串一次序列化整个对象，哈希可以对用户结构中的每个字段单独存储。这样当需要获取结构体信息时可以进行部分获取，而不用序列化所有字段，将整个字符串保存的结构体信息一次性全部读取。

3) 列表

Redis 列表(List)是简单的字符串列表，按照插入顺序排序。可以添加一个元素到列表的头部(左边)或者尾部(右边)。

列表的数据结构为快速双向链表。所以列表类型的前后插入和删除速度是非常快的，但是随机定位速度非常慢，需要对列表进行遍历，时间复杂度是 $O(n)$。

列表最多可存储 4 294 967 295($2^{32}-1$)个元素，每个列表可存储 40 多亿元素。

有两种实现方法：一种是压缩列表(ziplist)；另一种是双向循环链表。

当列表中存储的数据量比较小时，列表就可以采用压缩列表的方式实现。需要满足以下条件。

(1) 列表中保存的单个数据(有可能是字符串类型的)小于 64 字节。

(2) 列表中数据少于 512 个。

不能同时满足上面两个条件时，Redis 就使用双向循环链表来实现列表类型。

列表结构常用作异步队列，将需要延后处理的任务结构体序列化成字符串插入 Redis 的列表，另一个线程从这个列表中轮询数据进行处理。常用于秒杀抢购场景，在秒杀前将本场秒杀的商品放到列表中，因为列表的 pop 操作是原子性的，所以即使有多个用户同时请求，也是依次进行 pop 操作，列表空了 pop 抛出异常就代表商品卖完了。

4) 集合

Redis 的集合(Set)是字符串类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据。

Redis 的集合类型底层实现主要是通过哈希表。不过 Redis 为了追求极致的性能，会根据存储的值是否满足一定的条件，选择 intset 数据结构。所满足的条件为：

(1) 存储的数据都是整数。

(2) 存储的数据元素个数不超过 512 个。

当不能满足上述条件，即存储的数据量较大时，Redis 就采用哈希表来存储集合中的数据。

Redis 中集合是通过哈希表实现的，所以添加、删除、查找的复杂度都是 $O(1)$。

其使用场景也是比较单一的，常用在一些去重的场景里，例如，每个用户只能参与一次活动、一个用户只能中奖一次等去重场景。

5) 有序集合

Redis 中的有序集合(Sorted Set)也称为 Zset，有序集合同集合类似，也是字符串类型

元素的集合,且所有元素不允许重复。

但有序集合中,每个元素都会关联一个 double 类型的分数值。有序集合通过这个分数值进行由小到大的排序。有序集合中,元素不允许重复,但分数值却允许重复。

Redis 有序集合的内部使用哈希映射(Hash Map)和跳跃表(Skip List)来保证数据的存储和有序,哈希映射里存放的是成员到分数值的映射,而跳跃表里存放的是所有的成员,排序依据是哈希映射里存放的分数值,使用跳跃表的结构可以获得比较高的查找效率。

当数据量比较小时,Redis 会用压缩列表来实现有序集合。使用压缩列表来实现有序集合,需要满足以下条件。

(1) 所有数据的大小都要小于 64 字节。

(2) 元素个数要小于 128。

有序集合常用于各类热门排序场景。例如,热门歌曲榜单列表,值是歌曲 ID,分数值是播放次数,这样就可以对歌曲列表按播放次数进行排序。

5.1.2 Redis 集群模式

1. 主从模式

主从模式包含一个主节点(Master)与一个或多个从节点(Slave),数据的复制是单向的,只能由主节点到从节点,从节点一般只读,如图 5-1 所示。

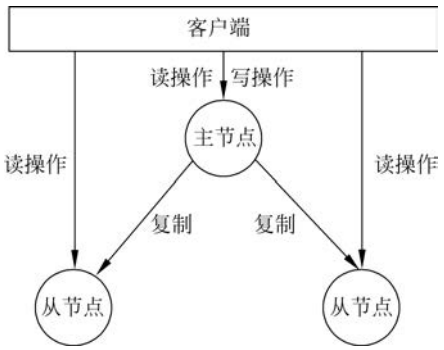

图 5-1 主从模式

1) 主从模式的特点

(1) 数据的复制是单向的,只能由主节点到从节点。

(2) 主节点可以进行读写操作,当数据变化时会自动将数据同步到从节点。

(3) 从节点一般都是只读的,并且接收主节点同步过来的数据。

(4) 一个主节点可以拥有多个从节点,但是一个从节点只能对应一个主节点。

(5) 从节点宕机不影响其他从节点的读和主节点的读和写,重新启动后会将数据从主节点同步过来。

(6) 主节点宕机后,不影响从节点的读,但 Redis 不再提供写服务,主节点重启后 Redis 将重新对外提供写服务。

(7) 主节点宕机后,不会在从节点中重新选一个主节点。

2) 主从模式的工作机制

当从节点启动后,主动向主节点发送 SYNC 命令。主节点接收到 SYNC 命令后在后台保存快照(RDB 持久化)和缓存保存快照这段时间的命令,将保存的快照文件和缓存的命令发送给从节点。从节点接收到快照文件和命令后加载快照文件和缓存的执行命令。

复制初始化后,主节点每次接收到的写命令都会同步发送给从节点,保证主从数据一致性。

3）主从模式的优点

（1）架构简单，部署方便。

（2）高可靠性。一方面，采用双机主备架构，能够在主节点出现故障时自动进行主备切换，从节点提升为主节点提供服务，保证服务平稳运行；另一方面，开启数据持久化功能和配置合理的备份策略，能有效地解决数据误操作和数据异常丢失的问题。

（3）读写分离策略。从节点可以扩展主节点的读能力，有效应对高并发量的读操作。

4）主从模式的缺点

（1）Redis 主从模式不具备自动容错和恢复功能，如果主节点宕机，Redis 集群将无法工作，此时需要人为干预，将从节点提升为主节点。

（2）如果主机宕机前有一部分数据未能及时同步到从机，即使切换主机后也会造成数据不一致的问题，从而降低了系统的可用性。

（3）因为只有一个主节点，所以其写入能力和存储能力都受到一定程度的限制。

（4）在进行数据全量同步时，若同步的数据量较大可能会造卡顿的现象。

5）主从同步原理

（1）全量复制。

Redis 全量复制一般发生在从节点初始化阶段，这时从节点需要将主节点上的所有数据都复制一份。具体过程如图 5-2 所示。

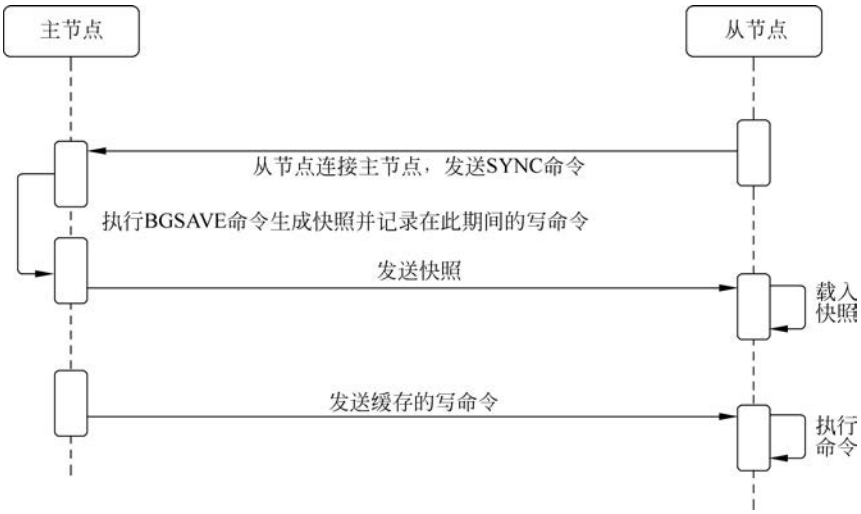

图 5-2　Redis 全量复制过程

① 从节点连接主节点，发送 SYNC 命令。

② 主节点接收到 SYNC 命令后，开始执行 BGSAVE 命令生成快照并使用缓冲区记录此后执行的所有写命令。

③ 主节点执行 BGSAVE 命令后，向所有从节点发送快照文件，并在发送期间继续记录被执行的写命令。

④ 从节点收到快照文件后丢弃所有旧数据，载入收到的快照。

⑤ 主节点快照发送完毕后开始向从节点发送缓冲区中的写命令。

⑥ 从节点完成对快照的载入，开始接收命令请求，并执行来自主节点缓冲区的写命令。

(2) 增量复制。

Redis 增量复制是指从节点初始化后开始正常工作时主节点发生的写操作同步到从节点的过程。

增量复制的过程主要是主节点每执行一个写命令就会向从节点发送相同的写命令,从节点接收并执行收到的写命令。

(3) 主从同步策略。

主从刚建立连接时,进行全量同步;全量同步结束后,进行增量同步。当然,如果有需要,从节点在任何时候都可以发起全量同步。Redis 策略是,无论如何,首先会尝试进行增量同步,如不成功,要求从节点进行全量同步。

2. 哨兵模式

自 Redis 2.8 版本开始,就有了哨兵(Sentinel)概念。哨兵模式(Redis Sentinel)是社区版本推出的原生高可用解决方案,其部署架构主要包括两部分:哨兵集群和数据集群,如图 5-3 所示。

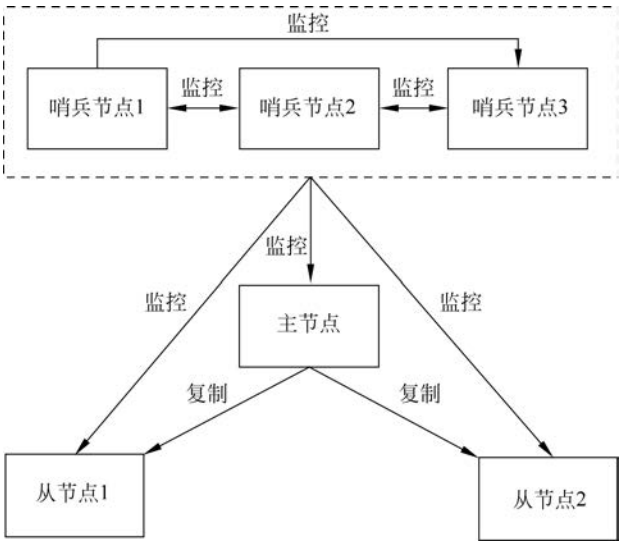

图 5-3 Redis 哨兵模式

哨兵集群是由若干哨兵节点组成的分布式集群,可以实现故障发现、故障自动转移、配置中心和客户端通知。哨兵节点数量必须是奇数个。

1) 哨兵模式的特点

(1) 当主节点宕机以后,哨兵节点会在从节点中选择一个作为主节点,并修改它们的配置文件,其他从节点的配置文件也会被修改,如 slaveof 属性会指向新的主节点。

(2) 当旧的主节点重新启动后,它将不再是主节点而是作为从节点,接收新的主节点的同步数据。

(3) 哨兵节点因为也是一个进程有宕机的可能,所以哨兵节点也会启动多个形成一个哨兵集群。

(4) 多哨兵节点配置时,哨兵节点之间也会自动监控。

(5) 当主从模式配置密码时,哨兵节点也会同步将配置信息修改到配置文件中。

(6) 一个哨兵节点或哨兵集群可以管理多个主从 Redis 实例，多个哨兵节点也可以监控同一个 Redis 实例。

(7) 哨兵节点尽量和 Redis 部署在不同的机器，否则 Redis 的服务器宕机以后，哨兵节点也宕机。

2) 哨兵模式的工作机制

(1) 每个哨兵节点以每秒一次的频率向它所知的主节点、从节点以及其他哨兵节点发送一个 ping 命令。

(2) 如果一个实例距离最后一次有效回复 ping 命令的时间超过 down-after-milliseconds 选项所指定的值，则这个实例会被哨兵节点标记为主观下线。

(3) 如果一个主节点被标记为主观下线，则正在监视这个主节点的所有哨兵节点要以每秒一次的频率确认主节点的确进入了主观下线状态。

(4) 当有足够数量的哨兵节点(大于或等于配置文件指定的值)在指定的时间范围内确认主节点的确进入了主观下线状态，则主节点会被标记为客观下线。

(5) 在一般情况下，每个哨兵节点会以每 10 秒一次的频率向它已知的所有主节点、从节点发送 info 命令。

(6) 当主节点被标记为客观下线时，哨兵节点向下线的主节点的所有从节点发送 info 命令的频率会从 10 秒一次改为 1 秒一次。

(7) 若没有足够数量的哨兵节点同意主节点已经下线，主节点的客观下线状态就会被移除。若主节点重新向哨兵节点的 ping 命令返回有效回复，则主节点的主观下线状态就会被移除。

3) 哨兵模式的优点

(1) 高可用。主从可以自动切换，系统更健壮，可用性更高。

(2) 实现多组节点监控。可以实现一套哨兵监控一组 Redis 数据节点或多组数据节点。

4) 哨兵模式的缺点

(1) 部署复杂。相对 Redis 主从模式部署更复杂，原理也更复杂。

(2) 资源利用率低。Redis 数据节点中从节点作为备份节点不提供服务。

(3) 默认不支持读写分离。不能解决读写分离问题，实现起来相对复杂。

(4) 维护成本大。需要多维护一套监控节点。

(5) Redis 较难支持在线扩容。对于集群，容量达到上限时在线扩容会变得很复杂。

3. 集群模式

上述两种模式的数据都是在一个节点上的，单节点存储是存在上限的。集群(Cluster)模式就是把数据进行分片存储，当一个分片数据达到上限时，就分成多个分片。Redis 3.0 加入了 Redis 的集群模式，对数据进行分片，将不同的数据存储在不同的主节点上面，从而实现了海量数据的分布式存储和在线扩容。

- 集群模式通常具有高可用、高可扩展性、分布式、高容错性等特性。
- 集群模式节点最小配置 6 节点(3 主 3 从)，其中主节点提供读写操作，从节点作为备用节点，不提供服务，只作为故障转移使用。

- 集群模式采用虚拟槽分区,所有的键根据哈希函数映射到 0～16 383 个整数槽内,每节点负责维护一部分槽以及槽所映射的键值数据。

如图 5-4 所示,Redis 集群模式可以看成多个主从架构组合起来的,每一个主从架构可以看成一个节点(其中,只有主节点具有处理请求的能力,从节点主要是用于保证节点的高可用)。

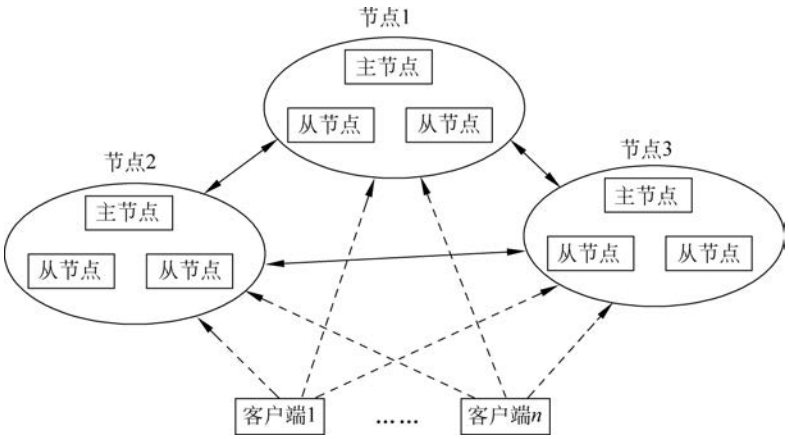

图 5-4　Redis 集群模式

1) 集群模式的特点

(1) 多个 Redis 节点网络互联,数据共享。

(2) 所有的节点都是一主一从(也可以是一主多从),其中从节点不提供服务,仅作为备用。

(3) 不支持同时处理多个键,如 MSET/MGET 操作,因为 Redis 需要把键均匀分布在各节点上,在并发量很高的情况下同时创建键值对会降低性能并导致不可预测的行为。

(4) 支持在线增加、删除节点。Redis 提供了进行重新分片的所有命令。集群的重新分片由内部的管理软件 redis-trib 负责执行,redis-trib 通过向节点发送命令来进行重新分片。

(5) 客户端可以连接任何一个主节点进行读写。

2) 集群模式的工作机制

(1) 选举。

集群启动后,主从已分配完成,经过了多轮的选举。当某一个主节点宕机,那么从节点需要经过选举成为主节点。下面简单介绍选举过程。

所有从节点向其他节点发送请求,请求自身成为主节点,其他节点收到请求后,返回投票信息,只有主节点有权投票,且只能投一次,当获取到的票数大于一半人数时(主节点个数),就当选主节点。

期间,所有从节点发送请求的时间有先后顺序,因此很少会出现票数相同的情况,如果相同,则重新选举,直到选出主节点为止。所以,需要至少 3 主 3 从,否则节点出现问题,将造成选举失败。

(2) 槽位。

在 Redis 集群中,定义了 16 384 个逻辑上的槽位。这些槽位均匀分配给多个节点(一

主一从为一节点)。例如,集群中存在3个节点,自动按序均匀分配,即0～5460个槽位分配给第一个节点。

当用户设置一个值时,除了计算键本身的哈希值之外,还会调用C语言的一个CRC16算法,将键当哈希值再计算出一个数字,然后与16 384取模,得到的数字落在哪个槽位,则会将数据放在对应的节点。

例如,计算出的数字为16 387,则取模16 384后,得到3,在0与5460之间,则放入对应的第一节点,依此类推。

(3) 跳转。

主从模式中,只有主节点可以写入数据,而从节点只能读取数据。在Redis集群中,设置值时,如果计算出的槽位在另一台服务器上,则集群连接会自动跳转至相应服务器。

3) 集群模式的优点

(1) 无中心架构。

(2) 数据共享。数据按照槽位存储分布在多节点,节点间数据共享,可动态调整数据分布。

(3) 可扩展性。可线性扩展到1000多个节点,节点可动态添加或删除。

(4) 高可用性。当部分节点不可用时,集群仍可用。通过增加从节点做备用数据副本,能够实现故障自动转移,节点之间通过Gossip协议交换状态信息,用投票机制完成从节点到主节点的角色提升。

(5) 任意节点读写。客户端可以连接任何一个主节点进行读写。

4) 集群模式的缺点

(1) 实现复杂,开发成本高。

(2) 需要建立配套的周边设施,如监控、域名服务、存储元数据信息的数据库等。

(3) 维护成本高。

5.1.3　Redis的持久化机制

1. RDB快照

RDB持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘,实际操作过程是创建一个子进程,先将数据集写入临时文件,写入成功后,再替换之前的文件,用二进制压缩存储,如图5-5所示。该持久化机制是Redis默认方式。

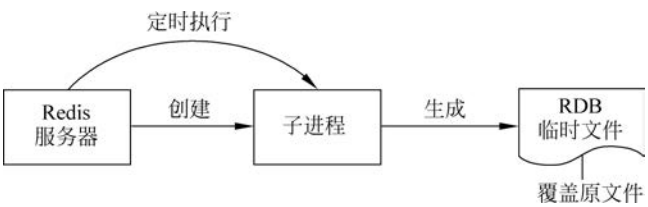

图5-5　RDB持久化过程

1) RDB的优点

(1) 利于备份。一旦采用该方式,那么整个Redis数据库将只包含一个文件,这对于文件备份而言是非常有利的。

(2) 数据恢复便捷。对于灾难恢复而言,RDB 是非常不错的选择。因为可以非常轻松地将一个单独的文件压缩后再转移到其他存储介质上。

(3) 性能最大化。对于 Redis 的服务进程而言,在开始持久化时,它唯一需要做的只是创建子进程,之后再由子进程完成这些持久化的工作,这样极大地避免服务进程执行 I/O 操作。

(4) 启动效率高。相比于 AOF 机制,如果数据集很大,RDB 的启动效率会更高。

2) RDB 的缺点

(1) 数据的完整性和一致性不高。系统一旦在定时持久化之前出现宕机现象,此前没有来得及写入磁盘的数据都将丢失。

(2) 备份时占用内存。Redis 在备份时会独立创建一个子进程,将数据写入一个临时文件(此时内存中的数据是原来的两倍),最后再将临时文件替换之前的备份文件。

3) RDB 持久化配置

Redis 会将数据集的快照转储(Dump)到 dump.rdb 文件中。可以通过配置文件来修改 Redis 服务器转储快照的频率,在打开 6379.conf 文件之后,搜索 save,可以看到下面的配置信息,如图 5-6 所示:

```
save 900 1              #在900秒(15分钟)之后, 如果至少有1个key发生变化, 则dump内存快照。
save 300 10             #在300秒(5分钟)之后, 如果至少有10个key发生变化, 则dump内存快照。
save 60 10000           #在60秒(1分钟)之后, 如果至少有10000个key发生变化, 则dump内存快照。
```

图 5-6 RDB 持久化配置信息

除了上面配置的会触发 RDB 持久化,还有以下几种默认方式。

(1) 执行 save(阻塞当前服务器,直到 RDB 完成为止)或者是 bgsave(异步)命令。

(2) 执行 flushall 命令,清空数据库所有数据。

(3) 执行 shutdown 命令,保证服务器正常关闭且不丢失任何数据。

2. AOF 日志

AOF 日志以文件的形式存在,是一种写后日志,其原理是将 Redis 的写操作以追加的方式写入文件。目前 AOF 是 Redis 持久化的主流方式,流程如图 5-7 所示。

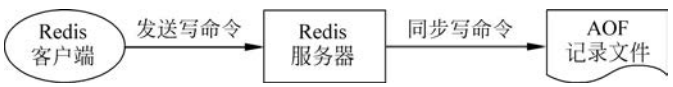

图 5-7 AOF 过程

1) AOF 的优点

(1) 数据的完整性和一致性更高,AOF 提供了 3 种同步策略,即每秒同步、每修改同步和从不同步。

(2) AOF 只是追加写日志文件,对服务器性能影响较小,速度比 RDB 要快,消耗的内存较少。

2) AOF 的缺点

(1) AOF 方式生成的日志文件太大,需要不断 AOF 重写,进行压缩。

(2) 即使经过 AOF 重写压缩,由于文件是文本文件,文件体积较大(相比于 RDB 的二

进制文件）。

（3）AOF 重演命令式的恢复数据，速度显然比 RDB 要慢。

3）AOF 持久化配置

在 Redis 的配置文件中存在 3 种同步方式，如图 5-8 所示。

```
appendfsync always      #每次有数据修改发生时都会写入AOF文件
appendfsync everysec    #每秒同步一次，该策略为AOF的默认策略
appendfsync no          #从不同步。高效但是数据不会被持久化
```

图 5-8　Redis 同步方式

3. Redis 4.0 混合持久化

若仅使用 RDB 快照方式恢复数据，由于快照时间粒度较大，因此会丢失大量数据。

若仅使用 AOF 重放方式恢复数据，日志性能相对 RDB 来说要慢。在 Redis 实例较大的情况下，启动需要花费很长的时间。

Redis 4.0 为了解决这个问题，带来了一个新的持久化选项——混合持久化，即将 RDB 文件的内容和增量的 AOF 日志文件存在一起。这里的 AOF 日志不再是全量的日志，而是自持久化开始到持久化结束的这段时间发生的增量 AOF 日志，通常这部分 AOF 日志相对较小，相当于：

（1）大量数据使用 RDB 快照方式。

（2）增量数据使用 AOF 日志方式。

在 Redis 重启时，可以先加载 RDB 的内容，然后再重放增量 AOF 日志就可以完全替代之前的 AOF 全量文件重放，重启效率因此大幅得到提升。

5.1.4　Redis 缓存穿透、击穿、雪崩

1. 缓存穿透

1）原因

当键对应的数据在数据源并不存在，每次针对此键的请求从缓存获取不到，请求都会直接访问数据源，从而可能因过大的访问压力而压垮数据源。例如，用一个不存在的用户 ID 获取用户信息，不论缓存还是数据库都没有，若黑客利用此漏洞进行攻击可能压垮数据库。

2）解决方案

（1）布隆过滤器。有多种方法可以有效地解决缓存穿透问题，最常见的是采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的基于位的映射（Bitmap）中，一个针对一定不存在数据的查询会被这个映射拦截掉，从而避免了对底层存储系统的查询压力。

（2）对空结果进行缓存，并缩短缓存时间设置。这是一个更为简单的方法，如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过 5 分钟。

2. 缓存击穿

1) 原因

虽然键对应的数据存在,但在 Redis 中过期,此时若有大量并发请求过来,这些请求发现缓存过期会从后端数据库加载数据并回写到缓存,这时大量并发的请求可能会瞬间把后端数据库压垮。针对的是一个热点键,如一个秒杀活动,并发量非常大。

2) 解决方案

使用互斥锁(Mutex Key):比较常用的做法。即在缓存失效时,不是立即导入数据,而是先使用缓存工具的某些带成功操作返回值的操作,如 Redis 的 SETNX 或者 Memcached 的 ADD,去设置一个互斥锁,当操作返回成功时,再进行导入数据的操作并回设缓存;否则,就重试整个获取缓存的方法。

3. 缓存雪崩

1) 原因

当缓存服务器重启或者大量缓存集中在某一个时间段失效,在失效时,也会给数据库带来很大压力。与缓存击穿的区别在于缓存雪崩针对多键缓存,缓存击穿则是针对某一个键。

2) 解决方案

(1) 使用锁或队列,可以在对数据库查询的地方进行加锁或队列控制,禁止所有请求同时访问数据库,以此缓解数据库的压力。

(2) 设置过期标志更新缓存,记录缓存数据是否过期,如果过期会触发通知另外的线程在后台去更新实际键的缓存。

(3) 为键设置不同的缓存失效时间,防止同一时间大量数据过期现象发生。

(4) 二次缓存机制,假设 C1 为原始缓存,C2 为复制的缓存,C1 失效时可以访问 C2,C1 缓存失效时间设置为短期,C2 缓存失效时间设置为长期。

5.1.5 Redis 的安装与使用

因为 Redis 官方不建议在 Windows 下使用 Redis,本书只介绍在 Linux 下 Redis 的安装方法。

在 Redis 官方网站选择下载所需要的 Redis 版本。下面以 6.0.8 版本为例,下载并安装 Redis。

```
#  wget http://download.redis.io/releases/redis-6.0.8.tar.gz
#  tar xzf redis-6.0.8.tar.gz
#  cd redis-6.0.8
#  make
```

执行完 make 命令后,redis-6.0.8 的 src 目录下会出现编译后的 Redis 服务程序 redis-server,还有用于测试的客户端程序 redis-cli。

启动 Redis 服务的命令如下:

```
#  cd src
#  ./redis-server
```

注意，这种方式启动 Redis 使用的是默认配置。也可以通过启动参数告诉 Redis 使用指定配置文件启动。使用下面命令启动：

```
#   cd src
#   ./redis-server ../redis.conf
```

redis.conf 是一个默认的配置文件。也可以根据需要使用自己的配置文件。

启动 Redis 服务进程后，就可以使用测试客户端程序 redis-cli 和 Redis 服务交互了。例如：

```
#   cd src
#   ./redis-cli
redis> set myk myval
OK
redis> get myk
"myval"
```

5.2 DynamoDB

5.2.1 DynamoDB 介绍

1. DynamoDB 简介

Amazon DynamoDB 是一种完全托管式、无服务器的 NoSQL 云数据库，支持键值对和文档数据模型，提供快速且可预测的性能，同时还能够无缝扩展。

它完全托管于 AWS(Amazon Web Services)，开发者只需要定义数据访问模式以及一些关键信息，就能通过 HTTP API 来使用。利用 DynamoDB，可以减轻操作和扩展分布式数据库的管理负担，这样就不必过多考虑硬件预置、设置和配置、复制、软件修补或集群扩展。同时，在任何规模的情况下都能针对应用提供毫秒级的响应。DynamoDB 还提供了静态加密，从而降低了在保护敏感数据时涉及的操作负担和复杂性。

2. DynamoDB 的特点

(1) 无缝扩展。DynamoDB 可以实现水平扩展，当表大小或访问量超过一定阈值时，DynamoDB 会自动且无缝地把一个表扩展到多个(最多几百个)服务器上以满足应用请求。

(2) 快速、可预期的性能。DynamoDB 服务端的平均延迟通常是几毫秒。运行在固态硬盘上面的 DynamoDB 服务，可以在任何扩展级别下维持一致性和低延迟。

(3) 易于管理。DynamoDB 是一个完全托管的数据库服务，用户只需要简单地创建一个数据库表，剩下的所有事情都由 AWS 服务来处理。不需要担心硬件和软件的配给、搭建、配置，也不要担心软件的安装和更新，更不必担心如何运行一个可靠的分布式数据库集群，或者把数据分区到多个实例。

(4) 内置容错性。DynamoDB 具有内在的容错能力，可以自动、同步地把数据复制到一个 Region 中的多个可用区中，即使遇到单个机器或设施的失效，数据也可以得到很好的保护。

(5) 数据高度灵活。DynamoDB 没有固定的模式(Schema)。相反,每个项目(Item)都具有不同数量的属性,可以支持多种数据类型。

(6) 强一致性、原子计数器。和许多 NoSQL 数据库不同,DynamoDB 使开发工作变得更加简单,它可以支持读操作的强一致性,从而保证可以总是获得最新的数据。读操作支持多个本地(Native)数据类型。这种服务也可以支持原子计数器(Atomic Counter),允许用户通过一个简单的 API 调用就可以实现数值属性自动增加和减少。

(7) 安全。DynamoDB 使用可靠的密钥方法,只允许授权用户访问数据,而不允许非授权用户的非法访问。DynamoDB 集成了 AWS Identity and Access Management(简称 AWS IAM),可以实现更细粒度的访问控制。

(8) 集成的监视功能。DynamoDB 可以在 AWS 管理控制台中,可视化关于表的关键性能指标。同时还集成了 Amazon Cloud Watch,可以让用户了解每个表的请求吞吐量和延迟,从而实现对资源的跟踪。

(9) 弹性的 MapReduce 集成。DynamoDB 同时集成了 Amazon EMR。Amazon EMR 可以支持对大型的数据集执行复杂的分析操作,并且采用 AWS 中按需付费的 Hadoop 框架。

3. DynamoDB 的数据类型

DynamoDB 对表中的属性支持很多不同的数据类型。可按以下方式为属性分类。

(1) 标量类型。标量类型可准确地表示一个值。标量类型包括数字、字符串、二进制、布尔值和 null。

(2) 文档类型。文档类型可表示具有嵌套属性的复杂结构。文档类型包括列表和映射。

(3) 集合类型。集合类型可表示多个标量值。集合类型包括字符串集、数字集和二进制集。

1) 标量类型

标量类型包括数字、字符串、二进制、布尔值和 null。

(1) 数字。

数字可为正数、负数或零。数字最多可精确到 38 位。超过此位数将导致异常。

在 DynamoDB 中,数字以可变长度形式表示。

所有数字将作为字符串通过网络发送到 DynamoDB,以最大限度地提高不同语言和库之间的兼容性。但是,DynamoDB 会将它们视为数字类型属性以方便数学运算。

可以使用数字数据类型表示日期或时间戳。执行此操作的一种方法是使用纪元时间,即自 1970 年 1 月 1 日 00:00:00 UTC 以来的秒数。

(2) 字符串。

字符串是使用 UTF-8 二进制编码的 Unicode。字符串受 DynamoDB 项目最大 400KB 的限制。此外,字符串属性如果未用作索引或表的键,那么其长度可以为 0。

以下附加约束将适用于定义为字符串类型的主键属性。

① 对于简单的主键,第一个属性值(分区键)的最大长度为 2048 字节。

② 对于复合主键,第二个属性值(排序键)的最大长度为 1024 字节。

使用字符串数据类型表示日期或时间戳。执行此操作的一种方法是使用 ISO 8601 字

符串。

(3) 二进制。

二进制(Binary)类型属性可以存储任意二进制数据，如压缩文本、加密数据或图像。DynamoDB 会将二进制数据的每字节视为无符号。

如果二进制属性未用作索引或表的键，且受到最大 DynamoDB 项目大小限制 400 KB 的约束，则该属性的长度可以为 0。

如果将主键属性定义为二进制类型属性，会有以下附加限制。

① 对于简单的主键，第一个属性值(分区键)的最大长度为 2048 字节。

② 对于复合主键，第二个属性值(排序键)的最大长度为 1024 字节。

在将二进制值发送到 DynamoDB 之前，应用程序必须采用 Base64 编码格式对其进行编码。收到值后，DynamoDB 会将数据解码为无符号字节数组。

(4) 布尔值。

布尔类型属性存储 true 或 false。

(5) null。

null 即空，代表属性具有未知或未定义状态。

2) 文档类型

文档类型包括列表和映射。这些数据类型可以互相嵌套，用来表示深度最多为 32 层的复杂数据结构。只要包含值的项目在 DynamoDB 项目大小限制(400KB)内，列表或映射中值的数量就没有限制。

如果属性未用于表或索引键，属性值可以是空字符串或空二进制值。属性值不能为空集(字符串集、数字集或二进制集)，但允许使用空的列表和映射。列表和映射中允许使用空的字符串和二进制值。

(1) 列表。

列表类型属性可存储值的有序集合。列表用方括号[]括起来。列表类似于 JSON 数组。列表元素中可以存储的数据类型没有限制，列表元素中的元素也不一定为相同类型。

以下示例显示了包含两个字符串和一个数字的列表。

```
FavoriteThings: ["Cookies","Coffee",3.14159]
```

(2) 映射。

映射类型属性可以存储名称/值对的无序集合。映射用大括号{}括起来。

映射类似于 JSON 对象。映射元素中可以存储的数据类型没有限制，映射中的元素也不一定为相同类型。

映射非常适合用来将 JSON 文档存储到 DynamoDB 中。以下示例显示了一个映射，该映射包含一个字符串、一个数字和一个含有另一个映射的嵌套列表。

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
```

```
            "Telephone",
            {
                Pens: { Quantity : 3},
                Pencils: { Quantity : 2},
                Erasers: { Quantity : 1}
            }
        ]
    }
```

3) 集合类型

DynamoDB 集合支持表示数字、字符串或二进制值集的类型。集合的所有元素必须为相同类型。例如,数字集类型的属性只能包含数字,字符串集只能包含字符串,以此类推。

只要包含值的项目大小在 DynamoDB 项目大小限制(400KB)内,集合的值的数量就没有限制。

集合的每个值必须是唯一的。集合的值的顺序不会保留。DynamoDB 不支持空集,但集合中允许使用空字符串和二进制值。

以下示例显示了一个字符串集:

```
["Black","Green","Red"]
```

4. 命名规则

下面是 DynamoDB 的命名规则。

(1) 所有名称都必须使用 UTF-8 进行编码,并且区分大小写。

(2) 表名称和索引名称的长度必须为 3～255 字符,而且只能包含以下字符。

- a～z/A～Z/0～9。
- _(下画线)。
- -(短线)。
- .(圆点)。

(3) 属性名称的长度必须至少为 1 个字符,但不得超过 64KB。存在以下例外,这些属性名称的长度不得超过 255 个字符。

- 二级索引分区键名称。
- 二级索引排序键名称。
- 任何用户指定的预测属性的名称。

5. AWS

AWS 即 Amazon Web Services(亚马逊网络服务),是亚马逊(Amazon)公司旗下的全球最全面、应用最广泛的云平台,从全球数据中心提供超过 200 项功能齐全的服务。

AWS 提供了一整套基础设施和应用程序服务,几乎能够在云中运行一切应用程序:从企业应用程序和大数据项目,到社交游戏和移动应用程序。

AWS 所提供的服务包括亚马逊弹性计算网云(Amazon EC2)、亚马逊简单存储服务(Amazon S3)、亚马逊简单数据库(Amazon SimpleDB)、亚马逊简单队列服务(Amazon Simple Queue Service)以及 Amazon CloudFront 等。

6. DynamoDB 与 Redis 的比较

1）数据结构层面

Redis 是单纯的键值对存储。

DynamoDB 虽然也是以键值对形式存储数据的，但是也引入了传统关系数据库中的表以及主键的概念，DynamoDB 中的分区键和排序键以及二级索引可以应对很多不同场景下的需求。

所以 DynamoDB 更像是处于 NoSQL 和传统关系数据库之间的一种数据库。

2）运行层面

Redis 是一种内存数据库，所有数据在内存中，支持高并发访问，使用单进程单线程＋I/O 多路复用机制保证了线程安全。

DynamoDB 是一项 Web 服务，数据的计算、存储均在 AWS 云中，用户所做的其实只是提交请求和接收响应，而不需要过多地考虑快速的数据量增长而导致的内存不足等问题。

3）应用层面

Redis 由于其性能更多地被用作缓存。

DynamoDB 则更接近于传统关系数据库的应用场景，如数据存储、增、删、查、改等。

4）持久化层面

Redis 的持久化机制是定期把内存中的数据写入磁盘，重新启动 Redis 时可以从磁盘中的转储文件加载至内存。

DynamoDB 的备份同样由 AWS 完成，利用 AWS 云计算框架在备份速度和安全性上都很优秀。并且由于其 Web 服务的性质，还给用户提供了自定义还原设置接口以及 35 天内可还原到任意一个时间上的功能。

5.2.2　DynamoDB 核心组件

在 DynamoDB 中，表、项目和属性是核心组件。表是项目的集合，而每个项目都是属性的集合。DynamoDB 使用主键来唯一标识表中的每个项目，并使用二级索引来提供更具灵活性的查询。用户可以使用 DynamoDB Streams 捕获 DynamoDB 表中的数据修改事件。

1. 表、项目和属性

1）表

与其他数据库系统类似，DynamoDB 将数据存储在表中。表是数据的集合。

2）项目

每个表包含 0 个或更多个项目。项目是一组属性，具有不同于所有其他项目的唯一标识。在 DynamoDB 中，对表中可存储的项目数没有限制。

3）属性

每个项目包含一个或多个属性。属性是基础的数据元素，无须进一步分解。DynamoDB 中的属性在很多方面都类似于其他数据库系统中的字段或列。

以图 5-9 中的 People 表为例。

(1) 表中的每个项目都有一个唯一的标识符或主键，用于将项目与表中的所有其他内容区分开来。在 People 表中，主键包含一个属性 (PersonID)。

(2) 与主键不同,People 表是无架构的,这表示属性及其数据类型都不需要预先定义。每个项目都能拥有其自己的独特属性。

(3) 大多数属性是标量类型的,这表示它们只能具有一个值。字符串和数字是标量的常见示例。

(4) 某些项目具有嵌套属性(Address)。DynamoDB 支持高达 32 级深度的嵌套属性。

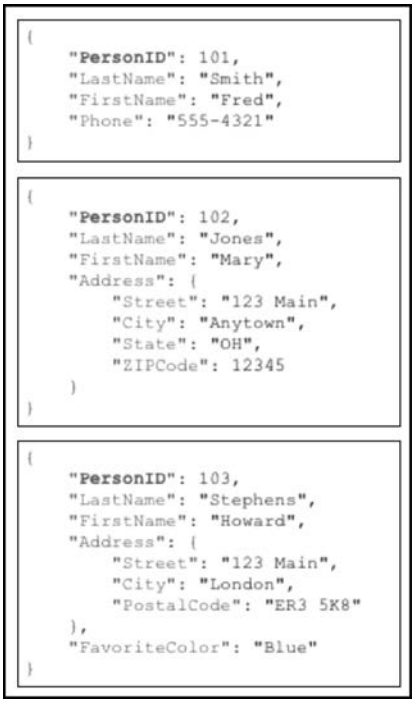

图 5-9 DynamoDB 表示例:People 表

2. 主键

创建表时,除表名称外,还必须指定表的主键。主键是表中每个项目的唯一标识,因此,任意两个项目的主键都不相同。

DynamoDB 支持两种类型的主键:分区键、分区键和排序键。

(1) 分区键是由一个称为分区键的属性构成的简单主键。DynamoDB 使用分区键的值作为内部哈希函数的输入。来自哈希函数的输出决定项目将存储到的分区。在只有分区键的表中,任何两个项目都不能有相同的分区键值。

图 5-10 显示了名为 Pets 的表,该表跨多个分区。表的主键为 AnimalType(仅显示此键属性)。在这种情况下,DynamoDB 会根据字符串 Dog 的哈希值,使用其哈希函数决定新项目的存储位置。注意,项目并非按排序顺序存储的。每个项目的位置由其分区键的哈希值决定。

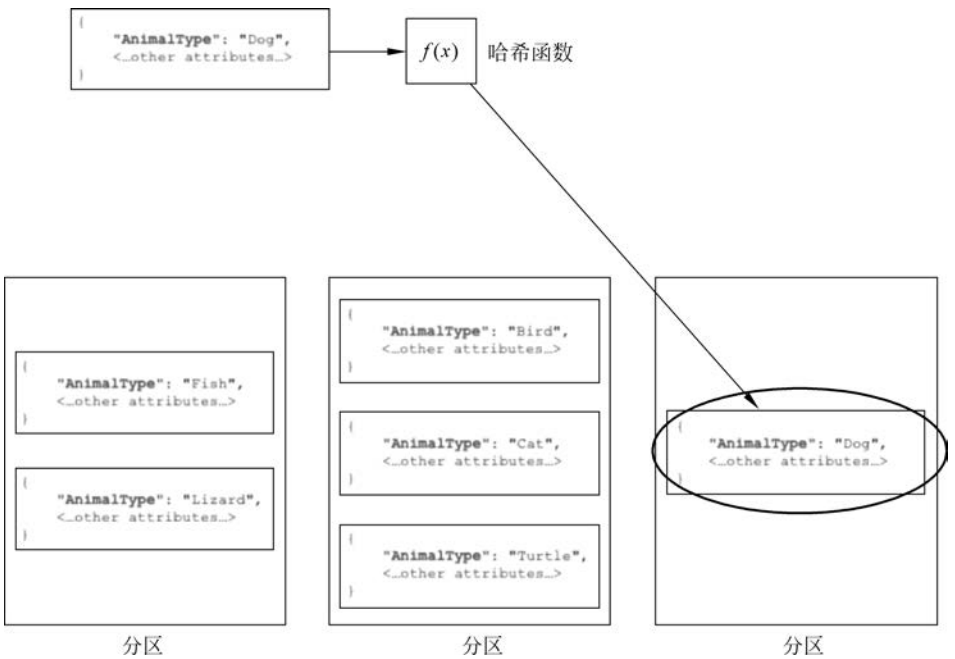

图 5-10 DynamoDB 分区示例:Pets 表分区

(2) 分区键和排序键称为复合主键。此类型的键由两个属性组成：分区键和排序键。DynamoDB 使用分区键值作为对内部哈希函数的输入。来自哈希函数的输出决定了项目将存储到的分区。具有相同分区键值的所有项目按排序键值的排序顺序存储在一起。在具有分区键和排序键的表中，多个项目可能具有相同的分区键值。但是，这些项目必须具有不同的排序键值。

假设 Pets 表具有由 AnimalType(分区键)和 Name(排序键)构成的复合主键。

图 5-11 显示了 DynamoDB 写入项目的过程，分区键值为 Dog，排序键值为 Fido。

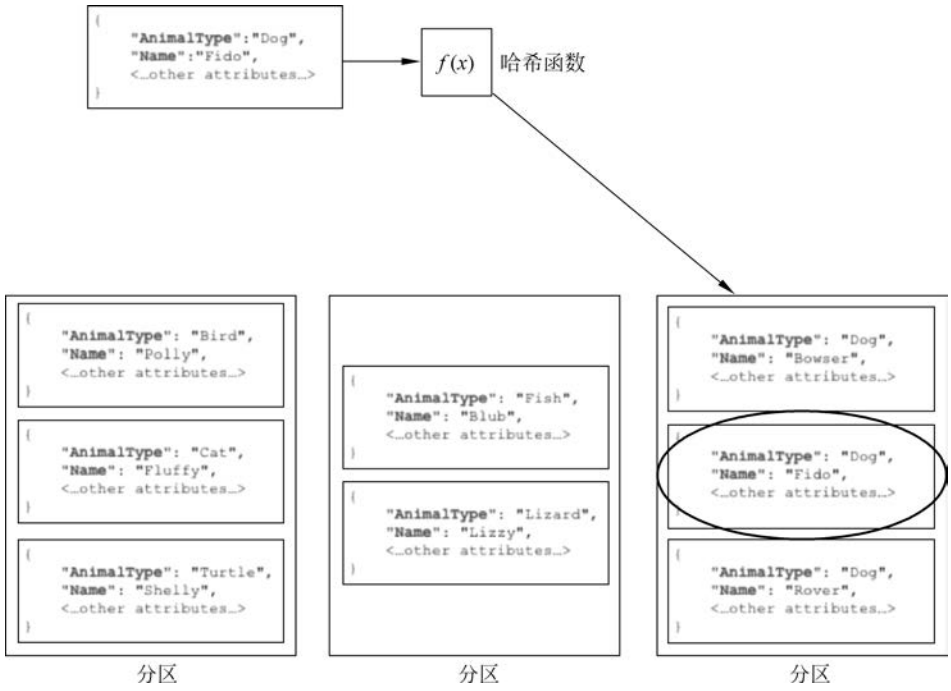

图 5-11　DynamoDB 写入项目过程

① 为读取 Pets 表中的同一项目，DynamoDB 会计算 Dog 的哈希值，从而生成这些项目的存储分区。然后，DynamoDB 会扫描这些排序键属性值，直至找到 Fido。

② 要读取 AnimalType 为 Dog 的所有项目，可以执行查询操作，无须指定排序键条件。默认情况下，这些项目会按存储顺序(即按排序键的升序)返回。或者，也可以请求以降序返回。

③ 要仅查询某些 Dog 项目，可以对排序键应用条件，例如，仅限 Name 在 A 至 K 范围内的 Dog 项目。

注意：

(1) 项目的分区键也称为其哈希属性。哈希属性一词源自 DynamoDB 中使用的内部哈希函数，以基于数据项目的分区键值实现跨多个分区的数据项目平均分布。

(2) 项目的排序键也称为其范围属性。范围属性一词源自 DynamoDB 存储项目的方式，它按照排序键值有序地将具有相同分区键的项目存储在互相紧邻的物理位置。

(3) 每个主键属性必须为标量(表示它只能具有一个值)。主键属性唯一允许的数据类型是字符串、数字和二进制。对于其他非键属性没有任何此类限制。

3. 二级索引

用户可以在一个表上创建一个或多个二级索引。利用二级索引,除了可对主键进行查询外,还可使用替代键查询表中的数据。DynamoDB 不要求用户使用索引,但它们为应用程序提供数据查询方面的更大的灵活性。在表中创建二级索引后,可以从索引中读取数据,方法与从表中读取数据大体相同。

DynamoDB 支持以下两种索引。

(1) 全局二级索引:分区键和排序键可与基表中的这些键不同。

(2) 本地二级索引:分区键可以和基表相同,排序键和基表保持不同。

DynamoDB 中的每个表具有 20 个全局二级索引(默认配额)和 5 个本地二级索引的配额。

图 5-12 显示了示例 Music 表,该表包含一个名为 GenreAlbumTitle 的新索引。在索引中,Genre 是分区键,AlbumTitle 是排序键。

图 5-12 DynamoDB 索引示例:Music 表及其索引

(1) 每个索引属于一个表(称为索引的基表)。在上述示例中,Music 是 GenreAlbumTitle 索引的基表。

(2) DynamoDB 将自动维护索引。当添加、更新或删除基表中的某个项目时,DynamoDB 会添加、更新或删除属于该表的任何索引中的对应项目。

(3) 当创建索引时,可指定哪些属性将从基表复制或投影到索引。DynamoDB 至少会将键属性从基表投影到索引中。如上面所示,此时 GenreAlbumTitle 索引中至少有 Music 表中的键属性(Artist 和 SongTitle)。

4. DynamoDB 流

DynamoDB 流(Stream)是一项可选功能，它用于捕获 DynamoDB 表中的数据修改事件。有关这些事件的数据将按事件发生的顺序近乎实时地出现在流中。

每个事件由一条流记录表示，若对表启用了流，每当以下事件发生时，DynamoDB 流都会写入一条流记录。

(1) 如果向表中添加了新项目，流将捕获整个项目的映像(包括其所有属性)。

(2) 如果更新了项目，流将捕获项目中任何已修改属性的“之前”和“之后”映像。

(3) 如果从表中删除了项目，流将在整个项目被删除前捕获其映像。

每条流记录还包含表名称、事件时间戳和其他元数据。流记录的有效时间为 24 小时，过此时间后记录将被自动删除。

此外，将 DynamoDB 流与 Amazon Lambda 结合使用以创建触发器——在流中有感兴趣的事件出现时自动执行的代码。如图 5-13 所示，假设有一个包含某公司客户信息的 Customers 表。假设希望向每位新客户发送一封“欢迎”电子邮件。可对该表启用一个流，然后将该流与 Lambda 函数关联。Lambda 函数将在新的流记录出现时执行，但只会处理添加到 Customers 表的新项目。对于具有 EmailAddress 属性的任何项目，Lambda 函数将调用 Amazon Simple Email Service(Amazon SES)以向该地址发送电子邮件。

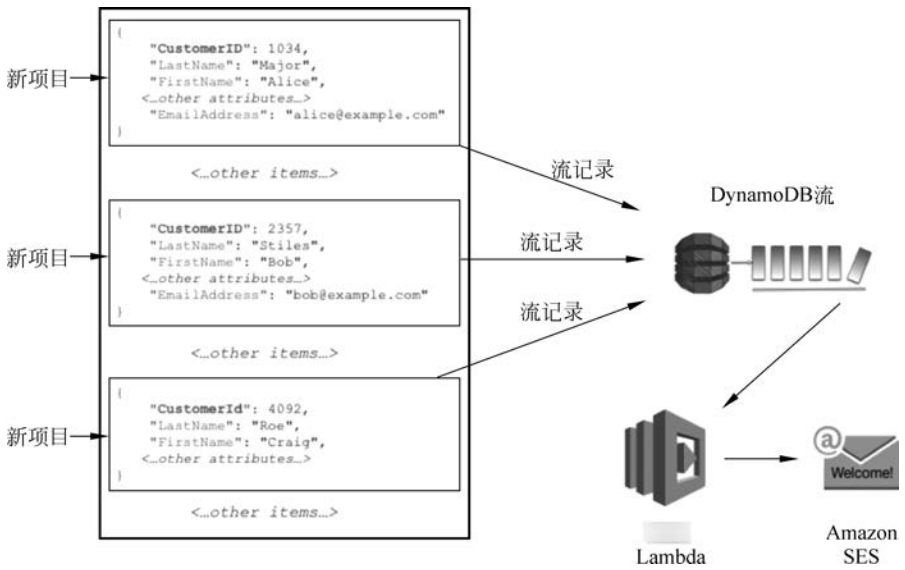

图 5-13　DynamoDB 触发器实例

在此示例中，最后一位客户 Craig Roe 将不会收到电子邮件，因为他没有 EmailAddress(电子邮件地址)。

除了触发器之外，DynamoDB 流还提供了强大的解决方案，例如，Amazon 区域内和区域之间的数据复制、DynamoDB 表中的数据具体化视图、使用 Kinesis 具体化视图的数据分析等。

5.2.3　DynamoDB API

1. 控制层面

控制层面操作可以用于创建和管理 DynamoDB 表，还支持用户使用依赖于表的索引、

流和其他对象。

- CreateTable——创建新表，或者可以创建一个或多个二级索引并为表启用DynamoDB流。
- DescribeTable——返回有关表的信息，例如，表的主键架构、吞吐量设置和索引信息。
- ListTable——返回列表中所有表的名称。
- UpdateTable——修改表或其索引的设置、创建或删除表上的新索引或修改表的DynamoDB流设置。
- DeleteTable——从DynamoDB中删除表及其所有依赖对象。

2. 数据层面

数据层面操作可对表中的数据执行创建、读取、更新和删除(也称为CRUD)操作。某些数据层面操作还可从二级索引读取数据。

可以使用PartiQL——Amazon DynamoDB的SQL兼容语言来执行这些CRUD操作，也可以使用DynamoDB的经典CRUD API，将每个操作分离为不同的API调用。

1) PartiQL

- ExecuteStatement——从表中读取多个项目。还可以写入或更新表的单个项目。当写入或更新单个项目时，必须指定主键属性。
- BatchExecuteStatement——写入、更新或读取表中的多个项目。这比ExecuteStatement更有效，因为应用程序只需一个网络往返行程即可写入或读取项目。

2) 经典API

(1) 创建数据。

- PutItem——将单个项目写入表中，必须指定主键属性，但不必指定其他属性。
- BatchWriteItem——将最多25个项目写入表，这比多次调用PutItem更有效，因为应用程序只需一个网络往返行程即可写入项目。还可以使用BatchWriteItem来从一个或多个表中删除多个项目。

(2) 读取数据。

- GetItem——从表中检索单个项目，必须为所需的项目指定主键。可以检索整个项目，也可以仅检索其属性的子集。
- BatchGetItem——从一个或多个表中检索最多100个项目，这比多次调用GetItem更有效，因为应用程序只需一个网络往返行程即可读取项目。
- Query——检索具有特定分区键的所有项目。必须指定分区键值。可以检索整个项目，也可以仅检索其属性的子集，还可以对排序键值应用条件，以便只检索具有相同分区键的数据子集。可以对表使用此操作，前提是该表同时具有分区键和排序键；还可以对索引使用此操作，前提是该索引同时具有分区键和排序键。
- Scan——检索指定表或索引中的所有项目。可以检索整个项目，也可以仅检索其属性的子集，还可以应用筛选条件以仅返回感兴趣的值并放弃剩余的值。

(3) 更新数据。

- UpdateItem——修改项目中的一个或多个属性，必须为要修改的项目指定主键。可

以添加新属性并修改或删除现有属性，也可以执行有条件更新，以便更新仅在满足用户定义的条件时成功，还可以实施一个原子计数器，该计数器可在不干预其他写入请求的情况下递增或递减数字属性。

(4) 删除数据。

- DeleteItem——从表中删除单个项目，必须为要删除的项目指定主键。
- BatchWriteItem——从一个或多个表中删除最多 25 个项目，这比多次调用 DeleteItem 更有效，因为应用程序只需一个网络往返行程即可删除项目。也可以使用 BatchWriteItem 来向一个或多个表添加多个项目。

3. DynamoDB 流

DynamoDB 流操作可对表启用或禁用流，并能允许对包含在流中的数据修改记录的访问。

- ListStreams——返回所有流的列表，或仅返回特定表的流。
- DescribeStreams——返回有关流的信息。
- GetShardIterator——返回一个分片迭代器，这是应用程序用来从流中检索记录的数据结构。
- GetRecords——使用给定分片迭代器检索一条或多条流记录。

4. 事务

事务提供原子性、一致性、隔离性和持久性 (ACID)，使用户能够更轻松地维护应用程序中的数据正确性。

用户可以使用 PartiQL 来执行事务操作，也可以使用 DynamoDB 的经典 CRUD API，将每个操作分离为不同的 API 调用。

1) PartiQL

ExecuteTransaction——一种批处理操作，用于在表内和跨表对多个项目执行 CRUD 操作，并保证得到全有或全无结果。

2) 经典 API

- TransactWriteItems——一种批处理操作，用于在表内和跨表对多个项目执行更新和删除等操作，并保证得到全有或全无结果。
- TransactGetItems——一种批处理操作，用于执行获取操作以从一个或多个表检索多个项目。

5.2.4　DynamoDB 工作原理

1. 读取一致性

DynamoDB 在全世界多个 Amazon 区域可用。每个区域均与其他 Amazon 区域独立和隔离。例如，如果一个 People 表在 us-east-2 区域，一个 People 表在 us-west-2 区域，则视为两个完全不同的表。

每个 Amazon 区域包含多个不同的称为“可用区”的位置。每个可用区都与其他可用区中的故障隔离，并提供与同一区域其他可用区的低成本、低延迟网络连接。这可以在某个区域的多个可用区之间快速复制数据。

当应用程序向 DynamoDB 表写入数据并收到 HTTP 200 响应（OK）时，该写入已发生并且持久。该数据最终将在所有存储位置中保持一致，通常只需 1 秒或更短时间。

DynamoDB 支持最终一致性和强一致性读取。

1）最终一致性读取

当从 DynamoDB 表中读取数据时，响应反映的可能不是刚刚完成的写入操作的结果。响应可能包含某些陈旧数据。如果在短时间后重复读取请求，响应将返回最新的数据。

2）强一致性读取

当请求强一致性读取时，DynamoDB 会返回具有最新数据的响应，从而反映来自所有已成功的之前写入操作的更新。但是，这种一致性有一些缺点。

（1）如果网络延迟或中断，可能会无法执行强一致性读取。在这种情况下，DynamoDB 可能会返回服务器错误（HTTP 500）。

（2）强一致性读取可能比最终一致性读取具有更高的延迟。

（3）全局二级索引不支持强一致性读取。

（4）强一致性读取可能比最终一致性读取使用更高的吞吐量。

注意：除非指定其他读取方式，否则 DynamoDB 将使用最终一致性读取。读取操作，如匹配项目（GetItem）、查询（Query）和扫描（Scan）提供了一个 ConsistentRead 参数。如果将此参数设置为 true，DynamoDB 将在操作过程中使用强一致性读取。

2. 读/写容量模式

DynamoDB 具有两个读/写容量模式来处理表的读写，即按需和预置（默认）。

读/写容量模式控制对读写吞吐量收费的方式以及管理容量的方式。可以在创建表时设置读/写容量模式，也可以稍后更改。

本地二级索引继承基表的读/写容量模式。

（1）按需模式指 DynamoDB 会随着工作负载的增加或减少，根据之前达到的任意流量水平即时调节工作负载。如果某个工作负载的流量级别达到一个新的峰值，DynamoDB 将快速调整以适应该工作负载。

如果满足以下任意条件，则按需模式是很好的选项。

- 创建工作负载未知的新表。
- 具有不可预测的应用程序流量。

（2）预置模式指定应用程序需要的每秒读取和写入次数。

如果满足以下任意条件，则预置模式是很好的选项。

- 具有可预测的应用程序流量。
- 运行流量比较稳定或逐渐增加的应用程序。

3. 分区与数据分配

DynamoDB 将数据存储在分区。分区是为表格分配的存储，由固态硬盘提供支持，并可在 Amazon 区域内的多个可用区中自动进行复制。分区管理由 DynamoDB 全权负责，用户不需要亲自管理分区。

在创建表时，表的初始状态为 CREATING。在此期间，DynamoDB 会向表分配足够的分区，以便满足预置吞吐量需求。表的状态变为 ACTIVE 后，便可开始读取和写入表数据。

在以下情况下，DynamoDB 会向表分配额外的分区。

- 增加的表的预置吞吐量设置超出了现有分区的支持能力。
- 现有分区填充已达到容量上限，并且需要更多的存储空间。
- 分区管理在后台自动进行，对程序是透明的。表将保留可用吞吐量并完全支持预置吞吐量需求。

DynamoDB 中的全局二级索引还包含分区。全局二级索引中的数据将与其基表中的数据分开存储，但索引分区与表分区的行为方式几乎相同。

如果表具有简单主键（只有分区键），DynamoDB 将根据其分区键值存储和检索各个项目。

如果表具有复合主键（分区键和排序键），DynamoDB 将采用与只有分区键相同的方式来计算分区键的哈希值。但是，它按排序键值有序地将具有相同分区键值的项目存储在互相紧邻的物理位置。

为将某个项目写入表中，DynamoDB 会计算分区键的哈希值以确定该项目的存储分区。在该分区中，可能有几个具有相同分区键值的项目。因此，DynamoDB 会按排序键的升序将该项目存储在具有相同分区键的其他项目中。

4. 表类别

DynamoDB 提供两个表类别，旨在帮助用户优化成本。“DynamoDB 标准”表类别是默认设置，建议用于绝大多数工作负载。“DynamoDB 标准-不经常访问（DynamoDB Standard-IA）”表类别针对存储占据主要成本的表进行优化，例如，存储不经常访问数据的表，如应用程序日志、旧的社交媒体帖子、电子商务订单历史记录以及过去的游戏成就适合使用该表类别。

思考题

1. 简述 Redis 与 DynamoDB 数据库的对比以及适用场景。
2. 查阅相关资料，说明 Redis 中 Zset 为什么不使用平衡树实现。
3. 查阅相关资料，说明 Redis 的内存淘汰机制。
4. 查阅相关资料，说明 Redis 6 之前使用单线程，为何 Redis 6 引入多线程以及其实现机制。
5. DynamoDB 在设计分区键时，需要考虑哪些因素？