1. **Redis**
2. **Redis 的特点**

Redis 是由意大利人 Salvatore Sanfilippo（网名：antirez）开发的一款内存高速缓存数据库。Redis 全称为： Remote Dictionary Server（远程数据服务），该软件使用 C 语言编写，典型的 NoSQL 数据库服务器，Redis 是一个 key-value 存储系统，它支持丰富的数据类型，如：string、list、set、zset(sorted set)、hash。

Redis 本质是一个 Key-Value 类型的内存数据库，很像 memcached，整个数据库全部加载在内存当中进行操作，定期通过异步操作把数库数据 flush 到硬盘上进行保存。因为是纯内存操作，Redis 的性能非常出色，每秒可以处理超过10万次读写操作，是已知性能最快的 Key-Value DB。

Redis 的出色之处不仅仅是性能，Redis 最大的魅力是支持保存多种数据结构，此外单个value 的最大限制是1GB，不像 memcached 只能保存 1MB 的数据，另外 Redis 也 可 以 对 存 入 的 Key-Value 设 置 expire 时 间 。

Redis 的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写，因此 Redis 适合的场景主要局限在较小数据量的高性能操作和运算上。

1. **Redis 有哪些功能？**

数据缓存功能

分布式锁的功能

支持数据持久化

支持事务

支持消息队列

1. **为什么 redis 需要把所有数据放到内存中**

不是什么地方都需要使用缓存，符合以下两个条件才需要使用缓存。

1. 数据访问的频率很高

2. 数据修改的频率低

另外在使用缓存的时候，要注意缓存是否有效利用，需要及时清理掉缓存中不常用的数据Redis 为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以 redis 具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘 I/O 速度会严重影响 redis 的性能。

1. **Redis 常见的性能问题都有哪些？如何解决？**

（1）、Master 写内存快照，save 命令调度 rdbSave 函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以 Master 最好不要写内存快照。

（2）、Master AOF 持久化，如果不重写 AOF 文件，这个持久化方式对性能的影响是最小的，但是 AOF 文件会不断增大，AOF 文件过大会影响 Master 重启的恢复速度。Master 最好不要做任何持久化工作，包括内存快照和 AOF 日志文件，特别是不要启用内存快照做持久化,如果数据比较关键，某个 Slave 开启 AOF 备份数据，策略为每秒同步一次。

（3）、Redis 主从复制的性能问题，为了主从复制的速度和连接的稳定性，Slave 和 Master 最好在同一个局域网内

（4）缓存穿透的问题

一般出现这样的问题，是因为当我们查询一条肯定不存在的数据的时候，缓存中没有，就会透过缓存来查询数据库，数据库也不存在，这样就不会将值保存在缓存中，最后还是缓存和数据库中都没有，如果一直访问这条数据。我们就对数据库进行频繁的查询，给数据库带来压力；

解决方法：当查询的时候，如果缓存和数据库中都没有，我们就将这个数据以空的形式存放在缓存中，（或者是给一个false的标示）这样就不用去数据库就可以知道不存在，减少对数据库查询的次数，当我们这个值发生改变的时候，我们在重新进行赋值；

（5）并发情况：当我们大量访问都是查询一个缓存中没有的数据时，这样就会都去数据库中进行查询，可能会造成数据库的宕机；

解决方法：在查询的时候，我给他添加了一个同步锁，只有第一条数据去数据库中查并且返回到redis中后才能查询，这是数据库中已近存在了值，这样也可以避免；

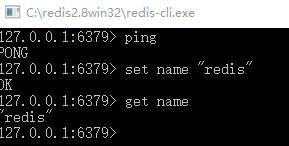
（6）雪崩：大量数据的访问时间失效，这样用户就会访问到数据库，第一台数据库崩溃了，访问就会到第二台数据库进行查询，这样会导致第二台的也崩溃；

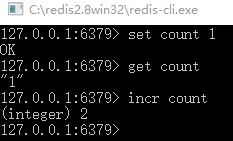
解决方法，就是设置失效时间时，不要一起失效，或者是设置在访问量少的时候，或者设置为永远不失效；

1. **Redis 最适合的场景有哪些**
2. 利用redis的过期策略,短信验证码存入redis,两分钟之内有效
3. 门户系统中的首页内容信息的展示。（商品类目、广告、热门商品等信息）门户系统的首页是用户访问量最大的，而且这些数据一般不会经常修改，因此为了提高用户的体验，我们选择将这些内容放在缓存中；
4. 单点登录系统中也用到了redis。因为我们是分布式系统，存在session之间的共享问题，因此在做单点登录的时候，我们利用redis来模拟了session的共享，来存储用户的信息，实现不同系统的session共享；
5. 我们项目中同时也将购物车的信息设计存储在redis中，购物车在数据库中没有对应的表，用户登录之后将商品添加到购物车后存储到redis中，key是用户id，value是购物车对象；
6. 因为针对评论这块，我们需要一个商品对应多个用户评论，并且按照时间顺序显示评论，为了提高查询效率，因此我们选择了redis的list类型将商品评论放在缓存中；
7. 在统计模块中，我们有个功能是做商品销售的排行榜，因此选择redis的zset结构来实现；
8. **Redis 有哪几种数据类型**

Redis 的数据结构有五种

**1.String**：最基本的数据类型，二进制安全（可以包含任何数据，比如jpg图片或序列化的对象）

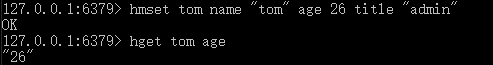




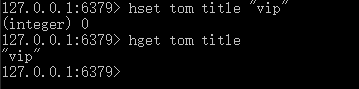
应用：统计每个用户每天访问网站的次数。用userid+时间戳（精确到日）

incr userId180901

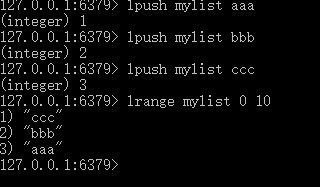
**2.Hash**:string元素组成的字典，适用于存储对象。



修改title为”vip”s

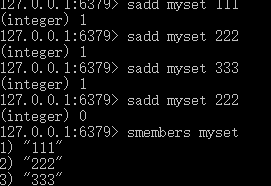


**3.List**列表，按String元素插入顺序排序



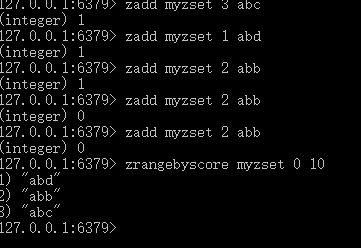
应用场景：实现最新消息排行榜

**4.set**:String元素组成的无序集合，通过哈希表实现，不允许重复。



应用场景：在微博中，所有的关注人存在一个集合中，redis提供了求交集，并集、差集，就可以实现共同爱好等功能。

**5.Sorted set**:通过分数来为集合中的成员进行从小到大的排序。（分数小排前）



应用场景：存储班级学生的成绩，score为分数，学号为value

1. **redis是单线程执行的,为什么性能这么高?**

因为 Redis 的瓶颈最有可能是机器内存或者网络带宽，而非单线程，既然单线程不是Redis 的性能瓶颈，并且单线程又比较容易实现，所以 Redis 就选择使用单线程来实现。

**单线程**并不代表运行速度就慢，比如，Nginx 和 NodeJs 都是单线程高性能的代表。

10万以上的QPS(QPS即每秒查询次数)

使用单线程，多个客户端对同一个key进行操作时，由单线程串行的处理，所以不会有并发的问题。避免频繁的上下文切换和锁竞争。

**完全基于内存**，绝大部分请求是纯粹的内存操作，不会受到硬盘IO的限制，执行效率高。

**多路IO复用模型(NIO)**： selector系统调用可以监听多个文件描述符（一个打开的文件通过唯一的描述符进行引用）的可读可写情况，当文件描述符可读可写时，selector就会返回可读可写的文件个数，这样Thread就可以去做别的事情而不会被阻塞。

Redis采用的多路复用函数有多个：epoll、kqueue、evport、select

1. **redis 设置过期时间**

Redis 通过 expire() 方法设置过期时间，语法：redis.expire(key, expiration)。当

expire 的过期时间设置为 -1 时，表示永不过期。

Redis中有个设置时间过期的功能，即对存储在redis 数据库中的值可以设置一个过期时间。作为一个缓存数据库， 这是非常实用的。如我们一般项目中的token 或者一些登录信息，尤其是短信验证码都是有时间限制的，按照传统的数据库处理方式，一般都是自己判断过期，这样无疑会严重影响项目性能。

我们set key 的时候，都可以给一个expire time，就是过期时间，通过过期时间我们可以指定这个key 可以存活的时间。

如果假设你设置了一批key 只能存活1个小时，那么接下来1小时后，redis是怎么对这批key进行删除的？ 定期删除+惰性删除。

**定期删除**：redis默认是每隔100ms 就随机抽取一些设置了过期时间的key，检查其是否过期，如果过期就删除。

注意这里是随机抽取的。为什么要随机呢？你想一想假如redis 存了几十万个key ，每隔100ms就遍历所有的

设置过期时间的key 的话，就会给CPU 带来很大的负载！

**惰性删除**：定期删除可能会导致很多过期key 到了时间并没有被删除掉。所以就有了惰性删除。假如你的过期key，靠定期删除没有被删除掉，还停留在内存里，除非你的系统去查一下那个key，才会被redis给删除掉。这就是所谓的惰性删除

但是仅仅通过设置过期时间还是有问题的。我们想一下：如果定期删除漏掉了很多过期key，然后你也没及时去查， 也就没走惰性删除，此时会怎么样？如果大量过期key堆积在内存里，导致redis内存块耗尽了。怎么解决这个问题呢？redise有数据淘汰策略。

1. **redis缓存淘汰机制**

LFU算法：Least Frequently Used，最不经常使用算法。

LRU算法：Least Recently Used，近期最少使用算法。LRU和LFU的区别。LFU算法是根据在一段时间里数据项被使用的次数选择出最少使用的数据项，即根据使用次数的差异来决定。而LRU是根据使用时间的差异来决定的。

ttl:从已设置过期时间中挑选将要过期的淘汰

random:从已设置过期时间的数据中随机淘汰

1. **如何实现缓存同步的**

只要使用了缓存就涉及到缓存同步的问题。缓存同步其实就是当缓存的信息发生变化，也就是后台对缓存的数据进行增、删、改操作后，数据库中的数据发生了变化同时要把缓存中的数据对应删除即可。当页面再次请求数据时，缓存中不能命中就会从数据库中查询并且添加到缓存中，即实现了缓存同步。

redis缓存的数据有一些是常驻缓存的，当数据库中数据有变化时做数据同步。  
有一些缓存是设置有效期的，当缓存到期后会自动删除。可以用ttl查看时间剩余。  
删除redis缓存使用del或者hdel命令。

也可以使用MQ,当数据库中的数据发生变化时发一个消息，通知redis更新缓存。

1. **Redis 的持久化**

在redis中的介绍中，我们知道了redis不光是一个基于内存的缓存，同时还支持数据的持久化。在redis的配置文件模块介绍中，我们可以设置redis的持久化方式。redis的持久化方式有两种，即RDB和AOF的方式，分别介绍如下：

**RDB（快照方式 snapshotting）（全量持久化）：**

将当前内存中的数据集快照写入磁盘，实现数据的持久化，恢复时可以将快照重新载入内存。

**AOF（append-only-file）（增量持久化）：**

在 redis配置文件的 APPEND ONLY MODE 中，可以设置AOF持久化。通过记录redis服务器所执行的写命令来记录数据库状态。恢复时可以将AOF文件载入内存，并且可以通过redis-check-aof --fix 进行修复AOF文件。

1. **redis集群**

用一个redis实例作为主机，其余的实例作为从机。主机和从机的数据完全一致，主机支持数据的写入和读取等各项操作，而从机则只支持与主机数据的同步和读取。因而可以将写入数据的命令发送给主机执行，而读取数据的命令发送给不同的从机执行，从而达到读写分离的目的。

问题是主从模式如果所连接的redis实例因为故障下线了，没有提供一定的手段通知客户端另外可连接的客户端地址，因而需要手动更改客户端配置重新连接。如果主节点由于故障下线了，那么从节点因为没有主节点而同步中断，因而需要人工进行故障转移工作。为了解决这两个问题，在2.8版本之后redis正式提供了sentinel（哨兵）架构。

1. **redis的哨兵机制**

Redis 提供的sentinel（哨兵）机制，通过sentinel 模式启动redis后，自动监控Master/Slave的运行状态，基本原理是：心跳机制+ 投票裁决。

简单来说，哨兵的作用就是监控Redis 系统的运行状况。它的功能包括以下两个：

1. 监控主数据库和从数据库是否正常运行；

2. 主数据库出现故障时自动将从数据库转换为主数据库。

哨兵模式主要有下面几个内容：

监控（ Monitoring ）：Sentinel 会定期检查主从服务器是否处于正常工作状态。

提醒（ Notification ）：当被监控的某个Redis 服务器出现异常时，Sentinel 可以通过API 向管理员或者其他应用程序发送通知。

自动故障迁移（Automatic failover）：当一个主服务器不能正常工作时，Sentinel 会开始一次自动故障迁移操作，它会将失效主服务器的其中一个从服务器升级为新的主服务器，并让失效主服务器的其他从服务器改为复制新的主服务器；当客户端试图连接失效的主服务器时，集群也会向客户端返回新主服务器的地址， 使得集群可以使用新主服务器代替失效服务器。

Redis Sentinel 是一个分布式系统，你可以在一个架构中运行多个Sentinel 进程（ progress ）。

优点

1. 哨兵模式主从可以切换，具备基本的故障转移能力；

2. 哨兵模式具备主从模式的所有优点。

缺点

1. 哨兵模式也很难支持在线扩容操作；

2. 集群的配置信息管理比较复杂。

1. **主从同步**

持久化保证了即使redis服务重启也不会丢失数据，因为redis服务重启后会将硬盘上持久化的数据恢复到内存中，但是当redis服务器的硬盘损坏了可能会导致数据丢失，如果通过redis的主从复制机制就可以避免这种单点故障。

集群的作用

1．主从备份防止主机宕机

2．读写分离,分担master的任务

3．任务分离,如从服分别分担备份工作与计算工作

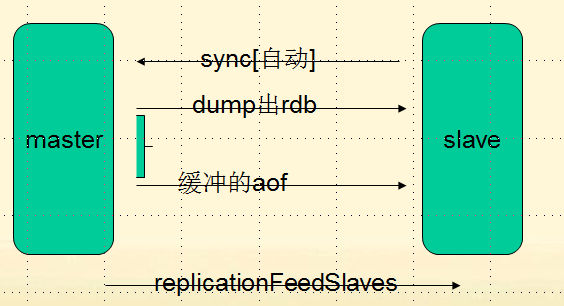
在Slave启动并连接到Master之后，它将主动发送一个SYNC命令。此后Master将启动后台存盘进程，同时收集所有接收到的用于修改数据集的命令，在后台进程执行完毕后，Master将传送整个数据库文件到Slave，以完成一次完全同步。而Slave服务器在接收到数据库文件数据之后将其存盘并加载到内存中。此后，Master继续将所有已经收集到的修改命令，和新的修改命令依次传送给Slaves，Slave将在本次执行这些数据修改命令，从而达到最终的数据同步。

1. slave先要求自动同步主机中的数据

2. master先通过dump将大块的rdb数据发送给slave

3. master再将缓存中的aof发送给slave

4. 以后如果master有新的数据，会通过replicationFeedSlaves进程通知slave



如果Master和Slave之间的链接出现断连现象，Slave可以自动重连Master，但是在连接成功之后，一次完全同步将被自动执行。

1. **redis 事务**

Redis是有事务的，redis中的事务是一组命令的集合，这组命令要么都执行，要不都不执行，redis事务的实现，需要用到MULTI（事务的开始）和EXEC（事务的结束）命令 ;



当输入MULTI命令后，服务器返回OK表示事务开始成功，然后依次输入需要在本次事务中执行的所有命令，每次输入一个命令服务器并不会马上执行，而是返回”QUEUED”，这表示命令已经被服务器接受并且暂时保存起来，最后输入EXEC命令后，本次事务中的所有命令才会被依次执行，可以看到最后服务器一次性返回了两个OK，这里返回的结果与发送的命令是按顺序一一对应的，这说明这次事务中的命令全都执行成功了。

Redis的事务除了保证所有命令要不全部执行，要不全部不执行外，还能保证一个事务中的命令依次执行而不被其他命令插入。同时，redis的事务是不支持回滚操作的。

1. **Redis实现分布式锁**

所谓Redis 的并发竞争Key 的问题也就是多个系统同时对一个key 进行操作，但是最后执行的顺序和我们期望的顺序不同，这样也就导致了结果的不同！

推荐一种方案：分布式锁（zookeeper 和redis 都可以实现分布式锁）。（如果不存在Redis的并发竞争Key 问题，不要使用分布式锁，这样会影响性能）

set key value[EX seconds][PX milliseconds] [NX|XX]

EX seconds ： 将键的过期时间设置为 seconds 秒

PX milliseconds ： 将键的过期时间设置为 milliseconds 毫秒

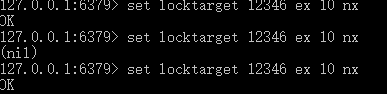
NX:只有键不存在时，才对键进行设置操作

XX: 只有键存在时，才对键进行设置操作

例如：获取一个名为target的锁。

如返回1，则该客户端获得锁，把lock.target的键值设置为时间值表示该键已被 锁定，该客户端最后可以通过DEL lock.target来释放该锁。

如返回0，表明该锁已被其它客户端取得，这时我们可以先返回或进行重试等 对方完成或等待锁超时。



大量的key同时过期时注意：当集中过期，由于清除大量的key很耗时，会出现卡顿现象，可以在设置key的过期时间时，给每个key加上随机值。

1. **多数据库测试**

一个Redis实例可以包括多个数据库，客户端可以指定连接某个redis实例的哪个数据库，就好比一个mysql中创建多个数据库，客户端连接时指定连接哪个数据库。一个redis实例最多可提供16个数据库，下标从0到15，客户端默认连接第0号数据库，也可以通过select选择连接哪个数据库

1. **Redis 常见异常及解决方案**

答： 在这里我们主要介绍Redis可能出现的三个问题，如下所示：

缓存雪崩：

举例：缓存同一时间大面积的失效，这个时候又来的一波请求都到数据库上，导致数据库连接异常。

解决办法：可以给缓存设置不同的缓存时间，更新数据使用互斥锁或者通过双缓存在避免缓存雪崩。

缓存穿透：

举例：故意的去请求缓存中不存在的数据，导致请求都打到了数据库上，导致数据库异常。

解决办法：可以使用互斥锁或者无论是否取到结果都将结果存入缓存，还可以使用有效的机制来拦截不合法的key值等。

**数据库和缓存的双写一致性问题：**

在高并发请求下很容易导致数据不一致的问题，如果你的业务需要保证数据的强一致性，那么建议不要使用缓存。在数据库中和缓存数据的删除或者写入过程中，如果有失败的情况，会导致数据的不一致。

解决办法：

（1）读的时候，先读缓存，缓存没有的话，那么就读数据库，然后取出数据后放入 缓存，同时返回响应

(2) 更新的时候，先删除缓存，然后再更新数据库，最后再隔固定的时间再次删除缓存。

1. **redis中对于生存时间的应用**

Redis中可以使用expire命令设置一个键的生存时间，到时间后redis会自动删除；

应用场景：

（1）分布式事务锁

（2）设置限制的优惠活动的信息；

（3）一些及时需要更新的数据，积分排行榜；

（4）手机验证码的时间；

（5）限制网站访客访问频率；

20.AOF和RDB的优点和缺点

### ＲＤＢ的优点

1. RDB 是一个非常紧凑（compact）的文件，它保存了 Redis 在某个时间点上的数据集。这种文件非常适合用于进行备份：比如说，你可以在最近的 24 小时内，每小时备份一次 RDB 文件，并且在每个月的每一天，也备份一个 RDB 文件。这样的话，即使遇上问题，也可以随时将数据集还原到不同的版本。
2. RDB 非常适用于灾难恢复（disaster recovery）：它只有一个文件，并且内容都非常紧凑，可以（在加密后）将它传送到别的数据中心，或者亚马逊 S3 中。
3. RDB 可以最大化 Redis 的性能：父进程在保存 RDB 文件时唯一要做的就是 fork 出一个子进程，然后这个子进程就会处理接下来的所有保存工作，父进程无须执行任何磁盘 I/O 操作。
4. RDB 在恢复大数据集时的速度比 AOF 的恢复速度要快

### ＲＤＢ的缺点

1. 如果你需要尽量避免在服务器故障时丢失数据，那么 RDB 不适合你。虽然 Redis 允许你设置不同的保存点（save point）来控制保存 RDB 文件的频率， 但是， 因为RDB 文件需要保存整个数据集的状态， 所以它并不是一个轻松的操作。因此你可能会至少 5 分钟才保存一次 RDB 文件。在这种情况下， 一旦发生故障停机， 你就可能会丢失好几分钟的数据。
2. 每次保存 RDB 的时候，Redis 都要 fork() 出一个子进程，并由子进程来进行实际的持久化工作。在数据集比较庞大时， fork()可能会非常耗时，造成服务器在某某毫秒内停止处理客户端；如果数据集非常巨大，并且 CPU 时间非常紧张的话，那么这种停止时间甚至可能会长达整整一秒。虽然 AOF 重写也需要进行 fork() ，但无论 AOF 重写的执行间隔有多长，数据的耐久性都不会有任何损失。

### AOF 的优点

1. 使用 AOF 持久化会让 Redis 变得非常耐久（much more durable）：你可以设置不同的 fsync 策略，比如无 fsync ，每秒钟一次 fsync ，或者每次执行写入命令时 fsync 。AOF 的默认策略为每秒钟 fsync 一次，在这种配置下，Redis 仍然可以保持良好的性能，并且就算发生故障停机，也最多只会丢失一秒钟的数据（ fsync 会在后台线程执行，所以主线程可以继续努力地处理命令请求）。
2. AOF 文件是一个只进行追加操作的日志文件（append only log）， 因此对 AOF 文件的写入不需要进行 seek ， 即使日志因为某些原因而包含了未写入完整的命令（比如写入时磁盘已满，写入中途停机，等等）， redis-check-aof 工具也可以轻易地修复这种问题。
3. Redis 可以在 AOF 文件体积变得过大时，自动地在后台对 AOF 进行重写：重写后的新 AOF 文件包含了恢复当前数据集所需的最小命令集合。整个重写操作是绝对安全的，因为 Redis 在创建新 AOF 文件的过程中，会继续将命令追加到现有的 AOF 文件里面，即使重写过程中发生停机，现有的 AOF 文件也不会丢失。而一旦新 AOF 文件创建完毕，Redis 就会从旧 AOF 文件切换到新 AOF 文件，并开始对新 AOF 文件进行追加操作。
4. AOF 文件有序地保存了对数据库执行的所有写入操作， 这些写入操作以 Redis 协议的格式保存， 因此 AOF 文件的内容非常容易被人读懂， 对文件进行分析（parse）也很轻松。导出（export） AOF 文件也非常简单：举个例子， 如果你不小心执行了 FLUSHALL 命令， 但只要 AOF 文件未被重写， 那么只要停止服务器， 移除 AOF 文件末尾的 FLUSHALL 命令， 并重启 Redis ， 就可以将数据集恢复到 FLUSHALL 执行之前的状态。

### AOF 的缺点

1. 对于相同的数据集来说，AOF 文件的体积通常要大于 RDB 文件的体积。
2. 根据所使用的 fsync 策略，AOF 的速度可能会慢于 RDB 。在一般情况下， 每秒 fsync 的性能依然非常高， 而关闭 fsync 可以让 AOF 的速度和 RDB 一样快， 即使在高负荷之下也是如此。不过在处理巨大的写入载入时，RDB 可以提供更有保证的最大延迟时间（latency）。
3. AOF 在过去曾经发生过这样的 bug ：因为个别命令的原因，导致 AOF 文件在重新载入时，无法将数据集恢复成保存时的原样。（举个例子，阻塞命令 BRPOPLPUSH 就曾经引起过这样的 bug 。） 测试套件里为这种情况添加了测试：它们会自动生成随机的、复杂的数据集， 并通过重新载入这些数据来确保一切正常。虽然这种 bug 在 AOF 文件中并不常见， 但是对比来说， RDB 几乎是不可能出现这种 bug 的