## redis讲解

另外为了保证高并发访问量大的情况，避免过多的连接一下子查询到数据库造成数据库的崩溃呢，我们采用了redis来实现数据的缓存，这样呢就会查询数据的时候，先从redis缓存中查询数据是否存在，如果缓存中存在我们就直接从缓存中取，这样可以减少多数据库的访问，如果缓存中不存在数据再去数据库查询，并且将查询出来的数据添加到缓存中，因为redis的查询速度是相当快的（11000次/s），另外为了保证redis服务器的安全通常会在redis.conf中绑定具体的ip地址这样只有该地址才能访问redis服务器，并且设置密码，为了保证redis不会因为占用内存过大而导致系统宕机，通常在将redis当做缓存服务器使用时，设置存储数据的过期时间，并且通过设置maxmemory【最大内存】和maxmemory-policy【数据清除策略】为allkeys-lru来达到预期的效果。我在项目中通常使用redis来充当缓存服务器来缓存分类列表，品牌列表，热销商品，推荐商品以及该商品的关联商品等等。使用了jedis作为客户端，并考虑到性能问题使用了jedis连接池。考虑到redis服务器的高可用性，我们做了redis的主从复制，并且通过加入哨兵来使redis主服务器宕机时，从服务器自动转换为主服务器继续提供服务。

## Redis可能出现的问题：

1：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；

2：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；

3：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；

## Redis持久化方面：

redis的一大特点就是可以将数据进行持久化，在一定程度上确保了数据的安全性，但不是绝对的；

首先持久化分为rdb（快照持久化）和aof（精细持久化）；

快照持久化，是默认开启的；会自动保存数据，当启动时会在文件夹中生成dump.rdb文件；存放持久化后的数据；

当然我们也可以设置持久化的频率，在redis.conf文件中通过save进行设置，默认有三种情况，每秒超过一万数据或每5分钟有10条数据的时候再或者每15分钟有1条记录，都会执行快照持久化，

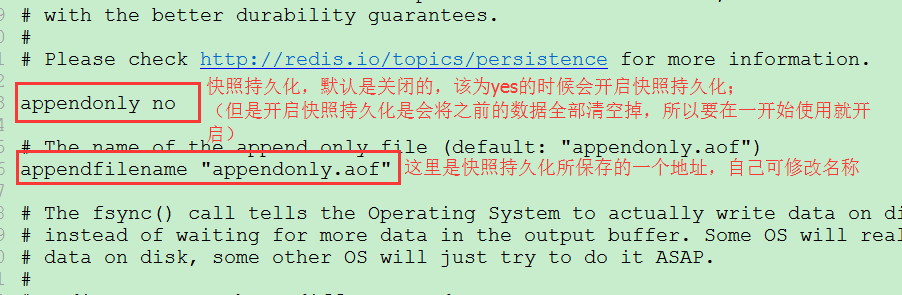
当然也可以通过bgsave的方法来手动进行一个快照持久化；（也可以通过ip和端口号就给别人进行手动持久化）；

如果频繁的快照持久化，会降低性能和效率，

但是这样也出现了一个问题，就是当一分钟内如果有一万条数据时，不会提交，但是在下一次提交之前，停电了，这样就会丢失掉这些数据；

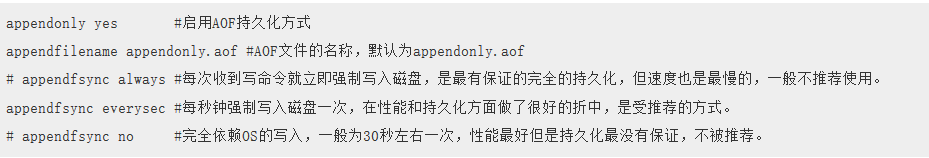
当时想到的解决方法呢就是和（AOF）精细持久化进行一个结合，达到一个秒级的持久化；

这个持久化需要我们手动进行开启，（注意，AOF开启之后，之前的数据都会丢失，所以要在一开始使用时就要配置好）开启的方法就是在配置redis.conf，将appendonly 改为yes；同时还可以更改文件名称；然后重新启动服务，这时精细化持久化就启动好了；



在快照持久化的同时，我们进行精细持久化，

比如，我们每隔一个小时进行一次快照持久化，这中间我们添加精细持久化；当55分的时候宕机了，我们还是可以通过RDB和AOF来恢复我们的数据，尽可能减少了损失；等到一个小时以后我们进行了快照持久化，数据就会保存在rdb的文件中，我们就可以将aof的持久化文件重新开始；



关于精细持久化也存在三个配置，

always只要发生改变就会发生持久化，这样是很浪费性能的，不推荐使用（我们的CPU不用干其他工作了，就在这里看着持久化）

everysec每秒进行一次快照持久化；推荐使用这种

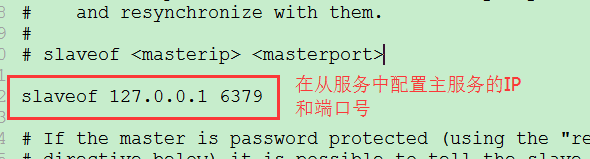
no 是不确定，看服务器的使用情况（心情）来定（不安全，不推荐）

我们还可以通过把rdb文件和aof文件复制到另外一个redis服务器上，就可以共享数据；

## Redis主从同步：

我们还可以对redis服务器配置主从同步，来减轻redis服务器的压力，

修改从服务对应的主服务地址（修改时可以直接搜索slaveof关键字）



这样访问时会读写分离，查询的时候会走从服务器，而更新操作的时候走主服务器；

与memcached的区别 （问到再说）

1，redis并不是把所有的数据都存放在内存，还可以将一些长期不使用的值存放在磁盘当中。而memcached只能存放在内存当中的；

2，redis的数据类型比memcached的数据类型丰富，除了string外，还有set,list,sortd set，hash等；

3：redis是可以持久化数据的，在一定程度上避免了数据的丢失，而memcached的安全性较低，一旦遇到停电，宕机的情况数据就会丢失；

4设置过期时间，memchched是在set key的时候直接设定时间（memcache.set(key,value,new Date(50)) 设置为50秒过期），而redis设置过期时间是通过expire来设置的；

## 如何在项目中使用redis的：

我们的项目采用的是maven来进行管理的，可以通过maven来进行包的依赖管理，保证jar包版本的一致性，所以只需要在pom.xml里添加对应的<dependency>信息即可自动下载jar包，

1：在项目中我们是通过jedis来操作redis的，jedis是java提供的用来操作redis的API

2：我们需要导入一些相关的jar包，目的是为了一些工具类的使用；

3：配置redis.xml文件,

1）我们配置了连接池；设置了最大链接数，最大空闲数；然后用静态语句块初始化连接池信息，在获得jedis实例

2）配置Jedis ConnectionFactory，通过IP和端口号来链接相对应的redis数据库；

3）配置了模板，并且在模板中引入了redistemplate的相关jar包；

4：在web.xml文件中引入了 redis.xml文件；

5：我们封装了相对应的工具类；通过操作模板来调用其中的方法，进行增删改查；在service层中获得jedis的实例，通过jedis实例来调用Redis中set类型对应的方法，就可以实现在Redis缓存中放入和取出查询到的商品的信息了，需要注意的是redis里只能存储字节，所以相对应的实体类要实现serializable这个接口；

## redis常见性能问题和解决方案：

(1) Master最好不要做任何持久化工作，如RDB内存快照和AOF日志文件

(2) 如果数据比较重要，某个Slave开启AOF备份数据，策略设置为每秒同步一次

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

(4) 尽量避免在压力很大的主库上增加从库

(5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3...

这样的结构方便解决单点故障问题，实现Slave对Master的替换。如果Master挂了，可以立刻启用Slave1做Master，其他不变。

还有就是Redis的秒杀场景

使用的是Redis的事务功能，先通过watch（）监控一个字段，且这个字段为0，之后对于秒杀数进行判断。当监控字段小于秒杀数时在IF条件内部内开启事务通过multi,当有人抢购成功的时候这个监控字段加1，且通过Exec执行事务，事务执行成功则抢购成功，否则抢购失败。抢购成功之后将相应的用户也存储到数据库中然后进行相应的订单处理。当监控字段大于秒杀数的时候也就是抢购完毕。无法进行抢购。

缓存的策略是每次数据在增删改之后就更新，主要是为了避免修改之后缓存更新不及时造成脏数据的出现