分片是指分区吗  
分库一般是将不同业务涉及的表分在不同服务器上 减少单机io  
分表一般分为水平分表和垂直分表。水平分表用在大表上 减少单表总行数。垂直分表是将不必要的列分离出来 减少列数。两个都能在一定条件下提高性能。

分区在物理上和分表很像 分为好几个数据文件存储。但是在逻辑上还是显示一张表。他比水平分表好的地方是对应用透明，并且管理数据方便 可以直接对分区操作，而且做聚合查询比水平分表方便

http://www.cnblogs.com/zhongxinWang/p/4262650.html

Sharding 不是一个某个特定数据库软件附属的功能，而是在具体技术细节之上的抽象处理，是水平扩展(Scale Out，亦或横向扩展、向外扩展)的解决方案，其主要目的是为突破单节点数据库服务器的 I/O 能力限制，解决数据库扩展性问题。

通过一系列的切分规则将数据水平分布到不同的DB或table中，在通过相应的DB路由或者table路由规则找到需要查询的具体的DB或者table，以进行Query操作。

“sharding”通常是指“水平切分”

以此类推，利用分库的规则，反向的路由到具体的DB，这个过程我们称之为“DB路由”。

**数据切分可以是物理上的**，对数据通过一系列的切分规则将数据分布到不同的DB服务器上，通过路由规则路由访问特定的数据库，这样一来每次访问面对的就不是单台服务器了，而是N台服务器，这样就可以降低单台机器的负载压力。

**数据切分也可以是数据库内的**，对数据通过一系列的切分规则，将数据分布到一个数据库的不同表中，比如将article分为article\_001，article\_002等子表，若干个子表水平拼合有组成了逻辑上一个完整的article表，这样做的目的其实也是很简单的。举个例子说明，比如article表中现在有5000w条数据，此时我们需要在这个表中增加（insert）一条新的数据，insert完毕后，数据库会针对这张表重新建立索引，5000w行数据建立索引的系统开销还是不容忽视的。但是反过来，假如我们将这个表分成100 个table呢，从article\_001一直到article\_100，5000w行数据平均下来，每个子表里边就只有50万行数据，这时候我们向一张 只有50w行数据的table中insert数据后建立索引的时间就会呈数量级的下降，极大了提高了DB的运行时效率，提高了DB的并发量。当然分表的好处还不知这些，还有诸如写操作的锁操作等，都会带来很多显然的好处。

分表的好处:

insert数据后建立索引的时间就会呈数量级的下降,极大了提高了DB的运行时效率，提高了DB的并发量

还有诸如写操作的锁操作等，都会带来很多显然的好处

分库降低了单点机器的负载；分表，提高了数据操作的效率，尤其是Write操作的效率。

但是为什么要分离读和写呢？熟悉DB的研发人员都知道，写操作涉及到锁的问题，不管是行锁还是表锁还是块锁，都是比较降低系统执行效率的事情。我们这样的分离是把写操作集中在一个节点上，而读操作其其他 的N个节点上进行，从另一个方面有效的提高了读的效率，保证了系统的高可用性。

mongo和mysql比较

1. mongo
2. mongodb是一个内存数据库，数据都是放到内存里面的，所以速度上不比redis慢。mongodb对数据的操作大部分都在内存中。但mongodb并不是单纯的内存数据库。甚至个人认为不属于内存数据库。mongodb的所有数据实际上是存放在硬盘的，所有要操作的数据通过mmap的方式映射到内存某个区域内。然后，mongodb就在这块区域里面进行数据修改，避免了零碎的硬盘操作。至于mmap上的内容flush到硬盘就是操作系统的事情了，所以，如果，mongodb在内存中修改了数据，然后，mmap数据flush到硬盘之前，系统当机了，就会丢失数据了。mysql，无论数据还是索引都存放在硬盘中。到要使用的时候才交换到内存中。能够处理远超过内存总量的数据。数据量和性能，

当物理内存够用的时候，redis》mongodb》mysql

当物理内存不够用的时候

redis和mongodb都会使用虚拟内存。

实际上如果redis要开始虚拟内存，那很明显要么加内存条，要么你换个数据库了。

但是，mongodb不一样，只要，业务上能保证，冷热数据的读写比，使得热数据在物理内存中，mmap的交换较少。mongodb还是能够保证性能。有人使用mongodb存储了上T的数据。

mysql，mysql根本就不需要担心数据量跟内存下的关系。不过，内存的量跟热数据的关系会极大地影响性能表现。

当物理内存和虚拟内存都不够用的时候

估计除了mysql你没什么好选择了。

其实，从数据存储原理来看，我更倾向于将mongodb归类为硬盘数据库，但是使用了mmap作为加速的手段而已。

mmap简单来说就是：

mmap系统调用并不是完全为了用于共享内存而设计的。它本身提供了不同于一般对普通文件的访问方式，进程可以像读写内存一样对普通文件的操作。mmap系统调用使得进程之间通过映射同一个普通文件实现共享内存。普通文件被映射到进程地址空间后，进程可以像访问普通内存一样对文件进行访问，不必再调用read()，write（）等操作。mmap并不分配空间, 只是将文件映射到调用进程的地址空间里, 然后你就可以用memcpy等操作写文件, 而不用write()了.写完后用msync()同步一下, 你所写的内容就保存到文件里了. 不过这种方式没办法增加文件的长度, 因为要映射的长度在调用mmap()的时候就决定了.

* 1. 优点
     1. 高读写，尤其是写远大于mysql，适用于高写入系统
     2. 容易分片，分片允许I/O超预算时的线性横向扩展。Shards可以和复制结合，配合Replica sets能够实现Sharding+fail-over，不同的Shard之间可以负载均衡 没有join，容易水平扩展
     3. 基于位置的查询
     4. 修改表结构过程迅速，不会锁表
     5. 内置GridFS，支持大容量的存储。GridFS是一个出色的分布式文件系统，可以支持海量的数据存储。  
          内置了GridFS了MongoDB，能够满足对大数据集的快速范围查询。
     6. 第三方支持丰富。(这是与其他的NoSQL相比，MongoDB也具有的优势)
  2. 缺点
     1. 没有事务
     2. 占用空间大
        1. 空间预分配，64M、128M、256M那 样的指数递增，直到2G为单个文件的最大体积
        2. 字段名所占用的空间 每个字段的key-value都以BSON的形式存储
        3. 删除记录不释放空间

1. mysql
   1. 优点
   2. 缺点
      1. 一个COUNT类型的操作会锁定数据集

1. #将传入的数据都当成一个字符串，会对自动传入的数据加一个双引号。如：order by #user\_id#，如果传入的值是111,那么解析成sql时的值为order by "111", 如果传入的值是id，则解析成的sql为order by "id".

2. $将传入的数据直接显示生成在sql中。如：order by $user\_id$，如果传入的值是111,那么解析成sql时的值为order by user\_id,  如果传入的值是id，则解析成的sql为order by id.

3. #方式能够很大程度防止sql注入。

4.$方式无法防止Sql注入。

5.$方式一般用于传入数据库对象，例如传入表名.

6.一般能用#的就别用$.