**1.memcache redis mongodb 优缺点和他们之间的区别？**

**memcache**

免费开放源码，高性能的分布式内存对象缓存系统，在本质上是通用的，但它的目的是通过减轻数据库负载来加速动态web应用程序

**redis**

Redis是一个开源的、BSD许可的、高级的键值存储库。它通常被称为数据结构服务器，因为键可以包含字符串、散列、列表、集合和排序集

**mongoDB**

MongoDB是用C++语言编写的非关系型数据库。特点是高性能、易部署、易使用，存储数据十分方便。

**NoSql数据库优缺点**

在优势方面主要体现在下面几点：

简单的扩展

快速的读写

低廉的成本

灵活的数据模型

在不足方面主要有下面几点：

不提供对SQL的支持

支持的特性不够丰富

现有的产品不够成熟

memcache redis mongdb 优缺点：

-Redis

**优点**

支持多种数据结构，如 string（字符串）、 list (双向链表)、dict ( hash表)、set (集合）、zset (排序 set)、hyperloglog（基数估算）。

支持持久化操作，可以进行 aof 及 rdb 数据持久化到磁盘，从而进行数据备份或数据恢复等操作，较好的防止数据丢失的手段。

支持通过 Replication 进行数据复制，通过 master-slave 机制，可以实时进行数据的同步复制，支持多级复制和增量复制，master-slave 机制是Redis 进行 HA 的重要手段。单线程请求，所有命令串行执行，并发情况下不需要考虑数据一致性问题。

支持 pub/sub 消息订阅机制，可以用来进行消息订阅与通知。

支持简单的事务需求，但业界使用场景很少，并不成熟。

**缺点**

Redis 只能使用单线程，性能受限于 CPU 性能，故单实例 CPU 最高才可能达到 5-6wQPS 每秒（取决于数据结构，数据大小以及服务器硬件性能，日常环境中 QPS 高峰大约在1-2w左右）。

支持简单的事务需求，但业界使用场景很少，并不成熟，既是优点也是缺点。

Redis 在 string 类型上会消耗较多内存，可以使用 dict（hash 表）压缩存储以降低内存消耗。

-Memcache

**优点**

Memcached 可以利用多核优势，单实例吞吐量极高，可以达到几十万QPS（取决于 key、value 的字节大小以及服务器硬件性能，日常环境中QPS 高峰大约在4-6w左右）。适用于最大程度扛量。

支持直接配置为 session handle。

**缺点**

只支持简单的 key/value 数据结构，不像 Redis 可以支持丰富的数据类型。

无法进行持久化，数据不能备份，只能用于缓存使用，且重启后数据全部丢失。

无法进行数据同步，不能将 MC 中的数据迁移到其他 MC 实例中。

Memcached 内存分配采用 Slab Allocation机制管理内存，value 大小分布差异较大时会造成内存利用率降低，并引发低利用率时依然出现踢出等问题。需要用户注重 value 设计。

-MongoDB

**优点**

更高的写负载，MongoDB 拥有更高的插入速度。

处理很大的规模的单表，当数据表太大的时候可以很容易的分割表。

高可用性，设置M-S不仅方便而且很快，MongoDB 还可以快速、安全及自动化的实现节点（数据中心）故障转移。

快速的查询，MongoDB 支持二维空间索引，比如管道，因此可以快速及精确的从指定位置获取数据。MongoDB 在启动后会将数据库中的数据以文件映射的方式加载到内存中。如果内存资源相当丰富的话，这将极大地提高数据库的查询速度。

非结构化数据的爆发增长，增加列在有些情况下可能锁定整个数据库，或者增加负载从而导致性能下降，由于 MongoDB 的弱数据结构模式，添加1个新字段不会对旧表格有任何影响，整个过程会非常快速。

**缺点**

不支持事务。

MongoDB 占用空间缺点

过大 。

MongoDB 没有成熟的维护工具

Redis、Memcache 和、MongoDB 的区别

**1. 性能**

三者的性能都比较高，总的来讲：Memcache 和 Redis 差不多，要高于MongoDB。

**2. 便利性**

memcache 数据结构单一。

redis 丰富一些，数据操作方面，redis 更好一些，较少的网络 IO 次数。

mongodb 支持丰富的数据表达，索引，最类似关系型数据库，支持的查询语言非常丰富。

**3. 存储空间**

redis 在 2.0 版本后增加了自己的VM特性，突破物理内存的限制；可以对key value 设置过期时间（类似 memcache）。

memcache 可以修改最大可用内存,采用 LRU 算法。

mongoDB 适合大数据量的存储，依赖操作系统 VM 做内存管理，吃内存也比较厉害，服务不要和别的服务在一起。

**4. 可用性**

redis，依赖客户端来实现分布式读写；主从复制时，每次从节点重新连接主节点都要依赖整个快照,无增量复制，因性能和效率问题，所以单点问题比较复杂；不支持自动 sharding,需要依赖程序设定一致 hash 机制。一种替代方案是，不用 redis 本身的复制机制，采用自己做主动复制（多份存储），或者改成增量复制的方式（需要自己实现），一致性问题和性能的权衡。

Memcache 本身没有数据冗余机制，也没必要；对于故障预防，采用依赖成熟的 hash 或者环状的算法，解决单点故障引起的抖动问题。

mongoDB 支持 master-slave,replicaset（内部采用 paxos 选举算法，自动故障恢复）,auto sharding 机制，对客户端屏蔽了故障转移和切分机制。

**5. 可靠性**

redis 支持（快照、AOF）：依赖快照进行持久化，aof 增强了可靠性的同时，对性能有所影响。

memcache 不支持，通常用在做缓存,提升性能。

MongoDB 从1.8版本开始采用 binlog 方式支持持久化的可靠性。

**6. 一致性**

Memcache 在并发场景下，用 cas 保证一致性。

redis 事务支持比较弱，只能保证事务中的每个操作连续执行。

mongoDB 不支持事务。

**7. 数据分析**

mongoDB 内置了数据分析的功能( mapreduce),其他两者不支持。

**应用场景**

**三者应用场景完全不同：**

redis：是数据结构和算法的集合，快速的数据结构操作是他最大的特点，支持数据持久化；

memcache：单一键值对内存缓存的，做对象缓存无可替代的分布式缓存

MongoDB:是基于文档的数据库，所有数据是从参判进行读写，善长对无模式JSON数据查询。

2.mysql 主从数据不一致怎么办,如何发现？

导致mysql主从数据不一致的原因：

1.网络的延迟

由于mysql主从复制是基于binlog的一种异步复制，通过网络传送binlog文件，理所当然网络延迟是主从不同步的绝大多数的原因，特别是跨机房的数据同步出现这种几率非常的大，所以做读写分离，注意从业务层进行前期设计。

2.主从两台机器的负载不一致

由于mysql主从复制是主数据库上面启动1个io线程，而从上面启动1个sql线程和1个io线程，当中任何一台机器的负载很高，忙不过来，导致其中的任何一个线程出现资源不足，都将出现主从不一致的情况。

3.max\_allowed\_packet设置不一致

主数据库上面设置的max\_allowed\_packet比从数据库大，当一个大的sql语句，能在主数据库上面执行完毕，从数据库上面设置过小，无法执行，导致的主从不一致。

4.key自增键开始的键值跟自增步长设置不一致引起的主从不一致。

5.mysql异常宕机情况下，如果未设置sync\_binlog=1或者innodb\_flush\_log\_at\_trx\_commit=1很有可能出现binlog或者relaylog文件出现损坏，导致主从不一致。

6.mysql本身的bug引起的主从不同步。

7.版本不一致，特别是高版本是主，低版本为从的情况下，主数据库上面支持的功能，从数据库上面不支持该功能。

**mysql主从不同步、数据不一致解决办法：**

**第一种**:通过sql\_slave\_skip\_counter跳过同步错误，适用于一般异常如插入时主键冲突

这种情况一般是在从库进行了写操作，

错误提示

Last\_SQL\_Errno: 1062

Last\_SQL\_Error: Error 'Duplicate entry '14' for key 'PRIMARY'' on query. Defaultdatabase: 'hadoop'. Query: 'INSERT INTO `user` (`username`, `password`) VALUES ('b', 'b')'

1 row in set (0.00 sec)

如果提示

Last\_SQL\_Errno: 126

Last\_SQL\_Error: Error 'Incorrect key file for table './hadoop/user.MYI'; try to repair it' on query. Default database: 'hadoop'. Query: 'INSERT INTO `user` (`username`, `password`) VALUES ('c', 'c')'

则先修复下表repair table user;再继续下面的操作

从库执行:

#停止slave服务

mysql> stop slave;

Query OK, 0 rows affected, 1 warning (0.00 sec)

#跳过一步错误，后面的数字可变，如果还不行可执行多次

mysql> set global sql\_slave\_skip\_counter =1;

Query OK, 0 rows affected (0.00 sec)

#开始slave服务

mysql> start slave;

Query OK, 0 rows affected (0.00 sec)

#查看slave状态

mysql> show slave status\G

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

**第二种**：**使用第三方工具如pt-table-sync；**

**第三种**:**重新做主从，完全同步**

该方法适用于主从库数据相差较大，或者要求数据完全统一的情况，这种耗时长  
解决步骤如下：  
1.先进入主库，进行锁表，防止数据写入  
使用命令：  
mysql> flush tables with read lock;  
注意：该处是锁定为只读状态，语句不区分大小写  
2.进行数据备份  
#把数据备份到mysql.bak.sql文件  
[root@server01 mysql]#mysqldump -uroot -p -hlocalhost > mysql.bak.sql  
这里注意一点：数据库备份一定要定期进行，可以用shell脚本或者python脚本，都比较方便，确保数据万无一失  
3.查看master 状态  
mysql> show master status;  
+-------------------+----------+--------------+-------------------------------+  
| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |  
+-------------------+----------+--------------+-------------------------------+  
| mysqld-bin.000001 | 3260 | | mysql,test,information\_schema |  
+-------------------+----------+--------------+-------------------------------+  
1 row in set (0.00 sec)  
4.把mysql备份文件传到从库机器，进行数据恢复  
#使用scp命令  
[root@server01 mysql]# scp mysql.bak.sql root@192.168.128.101:/tmp/  
5.停止从库的状态  
mysql> stop slave;  
6.然后到从库执行mysql命令，导入数据备份  
mysql> source /tmp/mysql.bak.sql  
7.设置从库同步，注意该处的同步点，就是主库show master status信息里的| File| Position两项  
change master to master\_host = '192.168.128.100', master\_user = 'rsync', master\_port=3306, master\_password='', master\_log\_file = 'mysqld-bin.000001', master\_log\_pos=3260;  
8.重新开启从同步  
mysql> stop slave;  
9.查看同步状态  
mysql> show slave status\G 查看：  
Slave\_IO\_Running: Yes  
Slave\_SQL\_Running: Yes

10.验证。