第六章 数据库和缓存**

1.列举常见的关系型数据库与非关系型数据库.
2.MYSQL数据库常见的引擎与区别
3.简述事务及其特性
4.简述触发器.函数.视图,存储过程
5.Mysql的索引种类
6.索引在什么情况下遵循最左前缀的规则
7.Mysql中常见的函数
8.列举创建索引但是无法命中索引的情况
9.数据库导入导出的命令(结构和数据)
10.你了解那些数据库的优化方案.
11.char和varchar的区别?
12.简述mysql的执行计划的作用及使用方法
13.1000w条数据，使用limit offset分页时，为什么越往后翻越慢？如何解决?
14.什么是索引合并?
15.什么是覆盖索引
16.简述数据库读写分离
17.简述数据库分库分表（水平、垂直）
18.数据库锁的作用?
19.where 子句中有a,b,c三个查询条件，创建一个组合索引abc(a,b,c),以下哪种会命中索引?
(a) (b) (c) (a,b) (b,c) (a,c) (a,b,c)
20.mysql下面哪些查询不会使用索引?between,like,'c%',not in,not exists,!=, <,<=,=,>,>=,in
21.mysq中 varchar与char的区别以及 varchar(50)中的50代表的含义
22.请简述项目中优化sql语句执行效率的方法
23.从 delete语句中省略 where子句将产生什么后果?
24.叙述 mysql半同步复制原理
25.sql查询 存在的表有:
26.考虑如下表结构，写出建表语句
27.假设学生 Students和教师 Teachers关系模型如下所示
28.mysql中怎么创建索引
29.请简述sq1注入的攻击原理及如何在代码层面防止sq注入?
30.使用 Python实现将数据库的 student表中提取的数据写db.txt?
31.简述left join和right join的区别?
32.索引有什么作用,有哪些分类,有什么好处和坏处?
33.查询:
34.试列出至少三种目前流行的大型关系型数据库的名称. 其中你最熟悉的是?什么时候开始使用?
35.字段类型都是整数
36.用一条sql语句查询出每门课程都大于80分的学生姓名
37.设计表,关系如下:教师,班级,学生,科室.科室与教师为一对多关系,教师与班级为多对多关系,班级与学生为一对多关系,科室中需体现层级关系:
39. 什么是MySQL慢日志?
41.在对name做了唯一索引前提下，简述以下区别:
42.Redis 和 Memecached 的区别?
43.如何高效的找到 redis 中所有以 oldboy 开头的 key?
44.什么是一致性哈希？Python中是否有相应的模块及其代码实现?
45.redis 是单进程单线程的吗?
46.Redis 中数据库默认是多少个 db 及作用?
47.如果 redis 中的某个列表中的数据量非常大，如何实现循环显示每一个值?

1.列举常见的关系型数据库与非关系型数据库.

关系型数据库：
Oracle、DB2、Microsoft SQL Server、Microsoft Access、MySQL
非关系型数据库：
NoSql、Cloudant、MongoDb、redis、HBase

关系型数据库：
关系型数据库的特性
1、关系型数据库，是指采用了关系模型来组织数据的数据库；
2、关系型数据库的最大特点就是事务的一致性；要成功就一起成功,失败就一起失败；
3、简单来说，关系模型指的就是二维表格模型，而一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织。
关系型数据库的优点
1、容易理解：二维表结构是非常贴近逻辑世界一个概念，关系模型相对网状、层次等其他模型来说更容易理解；
2、使用方便：通用的SQL语言使得操作关系型数据库非常方便；
3、易于维护：丰富的完整性(实体完整性、参照完整性和用户定义的完整性)大大减低了数据冗余和数据不一致的概率；
4、支持SQL，可用于复杂的查询。
关系型数据库的缺点
为了维护一致性所付出的巨大代价就是其读写性能比较差；固定的表结构；高并发读写需求；海量数据的高效率读写；
非关系型数据库：
非关系型数据库的特性
1、使用键值对存储数据； 2、分布式； 3、一般不支持ACID特性； 4、非关系型数据库严格上不是一种数据库，应该是一种数据结构化存储方法的集合。
非关系型数据库的优点 ：
无需经过sql层的解析，读写性能很高；基于键值对，数据没有耦合性，容易扩展；存储数据的格式：nosql的存储格式是key,value形式、文档形式、图片形式等等，文档形式、图片形式等等，而关系型数据库则只支持基础类型。
非关系型数据库的缺点：
不提供sql支持，学习和使用成本较高；无事务处理，附加功能bi和报表等支持也不好；

2.MYSQL数据库常见的引擎与区别

```
InnoDB:mysql5.5版本后默认的存储引擎，支持事务，行级锁等。支持外键约束
MyISAM:mysql老版本的存储引擎  只有表级索
Memory：将数据存储在内存的引擎
BlackHole：不存储数据
```

3.简述事务及其特性

事务：数据库保证完成一系列操作的机制，中途遇到任意操作失败会导致事务内的所有操作失败，已完成的会发生回滚。
特性：事务具有4个属性：原子性、一致性、隔离性、持久性。这四个属性通常称为ACID特性。
原子性（atomicity）。一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。
一致性（consistency）。事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。
隔离性（isolation）。一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
持久性（durability）。持久性也称永久性（permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

4.简述触发器.函数.视图,存储过程

视图

视图是一个虚拟表（只包含表结构），其本质是根据SQL语句获取动态的数据集，并为其命名，用户使用时只需使用名称即可获取结果集，可以将该结果集当做表来使用。

```
#增
    create view show_all as select * from class inner join student on
student.cid=class.class_id;

#删
    drop view show_all;

#改
    alter view as ...;

#查
    select * from show_all;

#注意:
    1.当虚拟表中只有一张表的数据时可以对表数据进行修改，而多表关联后无法修改数据
    2.视图本身效率不高，且表结构发生变化时，视图需要自己去手动修改，扩展差，因此不建议使
用
```

触发器

使用触发器可以定制用户在对表进行增、删、改（没有查询）操作时前后的行为

基本语法:

```
# 插入前
CREATE TRIGGER tri_before_insert_tb1 BEFORE INSERT ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 插入后
CREATE TRIGGER tri_after_insert_tb1 AFTER INSERT ON tb1 FOR EACH ROW
BEGIN
    ...
END
```

举个例子:

```
delimiter //
create trigger tri_after_insert after insert on student for each row
begin
    if new.student_name="alex" then
        update class set class_name="全栈17期nb" where class_name="全栈17
期";
    end if;
end //
delimiter ;

#注意
1.触发器不能由用户主动执行，而是对表进行一定操作后（前）自动触发的
2.上述代码中 delimiter表示更换sql语句结束符， new表示新插入的数据，old表示旧的数据
3.触发器是在数据库层面做的操作，可以在业务逻辑中实现，建议在业务逻辑中完成

# 删除触发器
drop trigger tri_after_insert

# 查看触发器
show triggers
```

存储过程

存储过程包含了一系列可执行的sql语句，存储过程存放于MySQL中，通过调用它的名字可以执行其内部的一堆sql，也可以对外提供接口

```
# 优点
1.用于替代业务逻辑写的SQL语句，实现程序与sql解耦
2.基于网络传输，传别名的数据量小，而直接传sql数据量大
# 缺点
扩展功能不方便

# 创建存储过程
delimiter //
create procedure p1()
BEGIN
    select * from student;
```

```
    update class set class_name="Python全栈17期" where class_name="全栈17期"
;
END //
delimiter ;

#在mysql中调用
call p1()

#在python中基于pymysql调用
cursor.callproc('p1')
print(cursor.fetchall())
```

5.Mysql的索引种类

主键索引（聚簇索引）：主键索引的叶子节点存储的是整行数据
唯一索引
普通索引（index）（非主键索引）：也被称为二级索引，叶子节点存放的是主键的值，使用普通索引查询时需要先找到主键，再通过主键索引搜索一次，此过程称为回表。（普通索引又分出一种特殊情况，唯一索引，索引字段可以为空，但是必须唯一）
联合主键
联合唯一
联合索引：在多个字段上建立索引，只有在查询条件中使用了创建索引的最左N个字段或者最左N个字符，索引才会被使用，遵循最左前缀。

全文索引：MyISAM引擎特有的。

6.索引在什么情况下遵循最左前缀的规则

在联合索引中，在查询条件中使用了创建索引的最左N个字段或者最左N个字符时，会遵循最左原则。

7.Mysql中常见的函数

聚合函数：
    avg，sum，count，max，min
数学函数：
    绝对值函数：abs(x)，
    取模函数：mod(x,y)，
    随机数函数：rand()
    四舍五入函数：round(x,y)
字符串函数：
    合并字符串concat(str1,str2,str3…)，
    比较字符串大小：strcmp(str1,str2)，
    获取字符串字节数函数：length(str)
日期和时间函数：
    获取当前日期时间：now()
    从日期中选择出月份数：month(date),monthname(date)
    从日期中选择出周数：week(date)
    从时间中选择出小时数：hour(time)
    从时间中选择出分钟数：minute(time)

条件判断函数：
    if 处理双分支
    case 处理多分支

8.列举创建索引但是无法命中索引的情况

1.如果条件中有 or ，即使其中有条件带索引也不会命中(这也是为什么尽量少用or的原因）
2.like查询是以%开头，如果是int型索引不会命中，字符型的命中 'test%' 百分号只有在右边才可以命中。
3.如果列类型是字符串，那一定要在条件中将数据使用引号引用起来，否则不使用索引
4.没有查询条件，或者查询条件没有建立索引
5.查询条件中，在索引列上使用函数（ + ， - ， * ， / ），这种情况下需建立函数索引
    6.采用 not in, not exist
7.B-tree 索引 is null 不会走， is not null 会走
8.联合索引遵循最左原则，不满足的不会命中
参考： http://www.chinacion.cn/article/4907.html

9.数据库导入导出的命令(结构和数据)

MySQL命令行导出数据库
    1.进入MySQL目录下的bin文件夹：cd MySQL中到bin文件夹的目录，如我输入的命令行：cd C:\Program Files\MySQL\MySQL Server 4.1\bin(或者直接将windows的环境变量path中添加该目录)
    2.导出数据库：mysqldump -u 用户名 -p 数据库名 > 导出的文件名，如我输入的命令行:mysqldump -u root -p news > news.sql （输入后会让你输入进入MySQL的密码)（如果导出单张表的话在数据库名后面输入表名即可）
    3、会看到文件news.sql自动生成到bin文件下

命令行导入数据库
    1，将要导入的.sql文件移至bin文件下，这样的路径比较方便
    2，同上面导出的第1步
    3，进入MySQL：mysql -u 用户名 -p。如我输入的命令行:mysql -u root -p （输入同样后会让你输入MySQL的密码)
    4，在MySQL-Front中新建你要建的数据库，这时是空数据库，如新建一个名为news的目标数据库
    5，输入：mysql>use 目标数据库名，如我输入的命令行:mysql>use news;
    6，导入文件：mysql>source 导入的文件名;如我输入的命令行：mysql>source news.sql;MySQL备份和还原,都是利用mysqldump、mysql和source命令来完成的。
    参考： https://www.cnblogs.com/bluealine/p/7832210.html

10.你了解那些数据库的优化方案.

合理的建立索引
避免使用select *等
避免大批量的insert和delete
在数据库使用中尽量减少长事务
减少分布式事务的使用
用char代替varchar 定长的字段妨碍前面

11.char和varchar的区别?

char：固定长度，处理速度上要比varchar快速很多，但是占用更多的空间。如果数据长度小于固定长度，MySQL就会在它的右边用空格字符补足.（在检索操作中那些填补出来的空格字符将被去掉）非空CHAR的最大总长度是255【字节】；

varchar:可变长度，处理速度上要比char慢，但是占空间少。每个值只占用刚好够用的字节再加上一个用来记录其长度的字节（即总长度为L+1字节）非空VARCHAR的最大总长度是65533【字节】。如果数据长度超过定义的长度，非严格模式下会截断，严格模式下会报错

char的存储方式是，对英文字符（ASCII）占用1个字节，对一个汉字占用两个字节；而varchar的存储方式是，对每个英文字符占用2个字节，汉字也占用2个字节，两者的存储数据都非unicode的字符数据。

12.简述mysql的执行计划的作用及使用方法

参考：https://www.cnblogs.com/li1992/articles/9221829.html

13.1000w条数据，使用limit offset分页时，为什么越往后翻越慢？如何解决?

当一个数据库表过于庞大，LIMIT offset, length中的offset值过大，则SQL查询语句会非常缓慢，你需增加order by，并且order by字段需要建立索引。

如果使用子查询去优化LIMIT的话，则子查询必须是连续的，某种意义来讲，子查询不应该有where条件，where会过滤数据，使数据失去连续性。

如果你查询的记录比较大，并且数据传输量比较大，比如包含了text类型的field，则可以通过建立子查询。

```
SELECT id,title,content FROM items WHERE id IN (SELECT id FROM items ORDER BY id limit 900000, 10);
```

如果limit语句的offset较大，你可以通过传递pk键值来减小offset = 0，这个主键最好是int类型并且auto_increment

```
SELECT * FROM users WHERE uid > 456891 ORDER BY uid LIMIT 0, 10;
```

这条语句，大意如下：

```
SELECT * FROM users WHERE uid >=  (SELECT uid FROM users ORDER BY uid limit 895682, 1) limit 0, 10;
```

如果limit的offset值过大，用户也会翻页疲劳，你可以设置一个offset最大的，超过了可以另行处理，一般连续翻页过大，用户体验很差，则应该提供更优的用户体验给用户。

limit 分页优化方法

1.先找出第一条数据，然后大于等于这条数据的id就是要获取的数据

缺点：数据必须是连续的，可以说不能有where条件，where条件会筛选数据，导致数据失去连续性实验下

2.倒排表优化法

倒排表法类似建立索引，用一张表来维护页数，然后通过高效的连接得到数据

缺点：只适合数据数固定的情况，数据不能删除，维护页表困难

3.反向查找优化法

当偏移超过一半记录数的时候，先用排序，这样偏移就反转了

缺点：order by优化比较麻烦，要增加索引，索引影响数据的修改效率，并且要知道总记录数，偏移大于数据的一半

4.limit限制优化法

把limit偏移量限制低于某个数。。超过这个数等于没数据，我记得alibaba的dba说过他们是这样做的

5.只查索引法

```
① select * from table limit 2,1;
//含义是跳过2条取出1条数据，limit后面是从第2条开始读，读取1条信息，即读取第3条数据
② select * from table limit 2 offset 1;
//含义是从第1条（不包括）数据开始取出2条数据，limit后面跟的是2条数据，offset后面是从第1条开始读取，即读取第2,3条
```

14.什么是索引合并?

1、索引合并是把几个索引的范围扫描合并成一个索引。
2、索引合并的时候，会对索引进行并集，交集或者先交集再并集操作，以便合并成一个索引。
3、这些需要合并的索引只能是一个表的。不能对多表进行索引合并。
在使用explain对sql语句进行操作时，如果使用了索引合并，那么在输出内容的type列会显示index_merge，key列会显示出所有使用的索引。
参考：https://www.cnblogs.com/zhuifeng-mayi/p/9291503.html

15.什么是覆盖索引

覆盖索引又可以称为索引覆盖。
1： 就是select的数据列只用从索引中就能够取得，不必从数据表中读取，换句话说查询列要被所使用的索引覆盖。
2： 索引是高效找到行的一个方法，当能通过检索索引就可以读取想要的数据，那就不需要再到数据表中读取行了。如果一个索引包含了（或覆盖了）满足查询语句中字段与条件的数据就叫做覆盖索引。
3： 是非聚集组合索引的一种形式，它包括在查询里的Select、Join和Where子句用到的所有列（即建立索引的字段正好是覆盖查询语句[select子句]与查询条件[where子句]中所涉及的字段，也即，索引包含了查询正在查找的所有数据）。

16.简述数据库读写分离

读写分离，基本的原理是让主数据库处理事务性增、改、删操作（INSERT、UPDATE、DELETE），而从数据库处理SELECT查询操作。数据库复制被用来把事务性操作导致的变更同步到集群中的从数据库。
用一句话概括，读写分离是用来解决数据库的读性能瓶颈的。
其实就是将数据库分为了主从库，一个主库用于写数据，多个从库完成读数据的操作，主从库之间通过某种机制进行数据的同步，是一种常见的数据库架构。

17.简述数据库分库分表（水平、垂直）

1）数据库分表

把一张表按照一定的规则分解成不同的实体表。比如垂直划分和水平划分 垂直切分：把不同功能，不同模块的数据分别放到不同的表中，但是如果同一个模块的数据量太大就会存在性能瓶颈水平切分：垂直切分解决不了大表的瓶颈，如果同一个功能中表的数据量过大，就要对该表进行切分，为水平切分

通俗理解：垂直切分---分不同的模块表；水平切分---分同一个模块下的多个表

2）分库 将一堆数据放到不同的数据库中保存，上面说的都是在同一个数据库上，分库是分到不同的数据库上

https://www.jianshu.com/p/00bebc1c8441

18.数据库锁的作用?

数据库是一个多用户使用的共享资源。当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性。（类似于多线程并发的情况）

加锁是实现数据库[并发控制]的一个非常重要的技术。当事务在对某个数据对象进行操作前，先向系统发出请求，对其加锁。加锁后事务就对该数据对象有了一定的控制，在该事务释放锁之前，其他的事务不能对此数据对象进行更新操作。在数据库中有两种基本的锁类型：排它锁（Exclusive Locks，即X锁）和共享锁（Share Locks，即S锁）。当数据对象被加上排它锁时，其他的事务不能对它读取和修改。加了共享锁的数据对象可以被其他事务读取，但不能修改。数据库利用这两种基本的锁类型来对数据库的事务进行并发控制

19.where 子句中有a,b,c三个查询条件，创建一个组合索引abc(a,b,c),以下哪种会命中索引？(a) (b) (c) (a,b) (b,c) (a,c) (a,b,c)

按照最左原则：
命中：(a)(a,b) (a,c) (a,b,c)
不命中：(b) (c)(b,c)

20.mysql下面哪些查询不会使用索引?between,like,'c%',not in,not exists,!=, <,<=,=,>,>=,in

不使用索引：> / < / >= / <= / != / like / between and

21.mysq中 varchar与char的区别以及 varchar(50)中的50代表的含义

char：定长，效率高，一般用于固定长度的表单提交数据存储 ；例如：身份证号，手机号，电话，密码等
varchar：不定长，效率偏低，
总的来说：
长度的区别，char范围是0～255，varchar最长是64k，但是注意这里的64k是整个row的长度，要考虑到其它的 column，还有如果存在not null的时候也会占用一位，对不同的字符集，有效长度还不一样，比如utf8的，最多21845.

varchar 编码长度限制：
字符类型若为gbk，每个字符最多占2个字节，最大长度不能超过32766；
字符类型若为utf8，每个字符最多占3个字节，最大长度不能超过21845。
若定义的时候超过上述限制，则varchar字段会被强行转为text类型，并产生warning。
具体区别:https://www.cnblogs.com/Lance--blog/p/5193027.html
在mql4中，50代表的是该字段可以存储的数据的最大长度是50个字节，使用utf-8存汉字时，最大只能存储6个汉字.
在mysql5中，代表着50个字符，最大65535个字节，也就是说使用utf-8存汉字时，最大21845个字符

22.请简述项目中优化sql语句执行效率的方法

(1)优化查询过程中的数据访问；
    访问数据太多导致性能下降；
    不要使用 SELECT *。总是取出全部列，SELECT * 会让优化器无法完成索引覆盖扫描的优化
    确定应用程序是否在检索大量超过需要的数据，可能是太多行或列；
(2)优化长难的查询语句；
    切分查询；
    将一个大的查询分为多个小的相同的查询；
    一次性删除 1000万 的数据要比一次删除1万，暂停一会的方案更加损耗服务器开销。
(3)优化特定类型的查询语句；
    ③优化子查询（即嵌套查询）；
        (1)尽可能使用关联查询来替代。

23.从 delete语句中省略 where子句将产生什么后果?

(1)如果没有指定 WHERE 子句，MySQL 表中的所有记录将被删除 (B)
A. delete语句将失败因为没有记录可删除
B. delete语句将从表中删除所有的记录
C. delete语句将提示用户进入删除的标准
D. delete语句将失败，因为语法错误

24.叙述 mysql半同步复制原理

（1）默认情况下，MySQL的复制功能是异步的，异步复制可以提供最佳的性能， 主库把binlog日志发送给从库，这一动作就结束了，并不会验证从库是否接收完毕，这一过程，也就意味着有可能出现当主服务器或从服务器端发生故障的时候，有可能从服务器没有接收到主服务器发送过来的binlog日志，这就会造成主服务器和从服务器的数据不一致，甚至在恢复时造成数据的丢失。
注意：

半同步复制模式必须在主服务器和从服务器端同时开启，否则主服务器默认使用异步复制模式。
（2）为了解决上述可能发生的错误，MySQL 5.5 引入了一种半同步复制模式。该模式可以确保从服务器接收完主服务器发送的binlog日志文件并写入到自己的中继日志relay log里，然后会给主服务器一个反馈，告诉主服务器已经接收完毕，这时主服务线程才返回给当前session告知操作完成。
（3）当出现超时情况是，主服务器会暂时切换到异步复制模式，直到至少有一个从服务器从及时收到信息为止。

25.sql查询 存在的表有:

1. products(商品表) columns为id，name,price

2. orders(商城订单表) columns为id,reservation_id,product_id，quentity(购买数量)

3. reservations(酒店订单表) columns为id，user_id， price， created

需要查询的:
1. 各个商品的售卖情况，需要字段:商品名,购买总量,商品收入

2.所有用户在2018-01-01至2018-02-01下单次数，下单金额，商城下单次数，商城下单金额

3.历月下单用户数:下单十次用户数，下单两次用户数，下单三次及以上用户数

26.考虑如下表结构，写出建表语句

ID(自增主键)	NAME(非空)	Balance(非空)
1	A	19.59
2	A	29.59
3	A	198.90

```
Create table t1 (id int primary_key,auto_increment,name char(32) not null,Balance char(32)not null)
```

27.假设学生 Students和教师 Teachers关系模型如下所示

1. Student:(学号，姓名，性别，类别，身份证号)
2. Teacher:(教师号，姓名，性别，身份证号，工资)

其中，学生关系中的类别分别为”本科生”和"研究生两类"，性别分为男和"女”两类,

1.查询研究生教师平均工资(显示为平均工资),最高工资与最低工资之间的差值(显示为差值)的sql语句.

```
Select Avg(工资),Max(工资),Min(工资) from Teacher where 身份证号 in (select 身份证号 from Student 类型=研究生)
```

2.查询工资少于109元的女研究生教师的身份证号和姓名的SQL语句(非嵌套查询方式);

```
Select 身份证号，姓名 from Teacher inner join Student on Teacher.身份证号 = Student.身份证号 where 工资<109 and 性别=女 and 类别=研究生
```

28.mysql中怎么创建索引

在执行CREATE TABLE语句时可以创建索引，也可以单独用CREATE INDEX或ALTER TABLE来为表增加索引。
1. ALTER TABLE
ALTER TABLE用来创建普通索引、UNIQUE索引或PRIMARY KEY索引。
ALTER TABLE table_name ADD INDEX index_name (column_list)
ALTER TABLE table_name ADD UNIQUE (column_list)
ALTER TABLE table_name ADD PRIMARY KEY (column_list)
其中table_name是要增加索引的表名，column_list指出对哪些列进行索引，多列时各列之间用逗号分隔。索引名index_name可选，缺省时，MySQL将根据第一个索引列赋一个名称。另外，ALTER TABLE允许在单个语句中更改多个表，因此可以在同时创建多个索引。

2. CREATE INDEX
CREATE INDEX可对表增加普通索引或UNIQUE索引。

CREATE INDEX index_name ON table_name (column_list)

CREATE UNIQUE INDEX index_name ON table_name (column_list)
table_name、index_name和column_list具有与ALTER TABLE语句中相同的含义，索引名不可选。另外，不能用CREATE INDEX语句创建PRIMARY KEY索引。

29.请简述sq1注入的攻击原理及如何在代码层面防止sq注入?

通过把sql命令插入到web表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的sql命令。
1、对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双引号进行转换等。
2、不要使用动态拼装sql，可以使用参数化的sql或者直接使用存储过程进行数据查询存取。
3、不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。
4、不要把机密信息明文存放，一定要加密或者hash掉密码和敏感信息。
5、应用异常信息应该给出尽可能少的提示，最好使用自定义的错误信息对原始错误信息进行包装，把异常信息存放在独立的表中。

30.使用 Python实现将数据库的 student表中提取的数据写db.txt?

首先需要使用连接数据库的模块从表中读取数据,python3中使用的是pymysql连接数据库，

```
import pymysql
conn = pymysql.connect(
host = 'localhost',
port = 3306,
user = 'root',
```

```
password = '****',
database = 'db1',
charset='utf8'
)
cur = con.cursor()
```

关于游标cur的一些方法：
close():关闭此游标对象
fetchone():得到结果集的下一行
fetchmany([size = cursor.arraysize]):得到结果集的下几行
fetchall():得到结果集中剩下的所有行
excute(sql[, args]):执行一个数据库查询或命令
excutemany(sql, args):执行多个数据库查询或命令
注意事项：
对于数据增删改之后，一定要提交，
conn.commit()
在所有操作完成之后，一个好习惯是关闭数据库连接，关闭游标
cur.close()
conn.close()

```
select_sql = 'sql语句'
try:
    cur.execute(select_sql)
results = cursor.fetchall()       # 获取所有记录列表
except:
    print("Select is failed")
```

fetchone(): 该方法获取下一个查询结果集。结果集是一个对象
fetchall(): 接收全部的返回结果行.,是一个列表

最后通过
```
with open('db.txt','w') as f:
    f.write()
```
写入数据到文件

31.简述left join和right join的区别?

都是多表连接查询中的外连接方式
区别是:left join是左连接,优先显示左边中的全部记录
      right join是右连接,优先显示右表中的全部记录

补充:
1.语法:

```
SELECT 字段列表
    FROM 表1 INNER|LEFT|RIGHT JOIN 表2
    ON 表1.字段 = 表2.字段;
```

2.多表查询的连接方式有:内连接,左外连接,右外连接,全外连接.
3.全外连接:在内连接的基础上增加左边有右边没有的和左边没有右边有的记录,即,显示左右两张表的全部记录. 方式:左外连接 union 右外连接.

32.索引有什么作用,有哪些分类,有什么好处和坏处?

```
作用:加快查询,拖慢删除/添加的速度
分类:聚集索引,辅助索引
聚集索引:1.每张表只能有一个聚集索引;
         2.叶子节点直接对应数据,所以找到索引就是找到数据
         3.数据的存储物理地址是按照索引顺序来存的,所以按照聚集索引列排序非常快
辅助索引(非聚集索引):1.每张表可以有多个辅助索引,查询速度快,但占用更多磁盘空间,影响删除和添加的效率
                    2.叶子节点不直接存放数据,而是存放数据的地址,所以找到叶子节点后还需再做一次IO操作
                    3.数据的物理地址和索引顺序无关.

补充:
为了加快查询速度,根据二分法速度快的原理,产生可平衡二叉树,但是b树高度高,查询次数多,就增加了分叉,形成了b-树.
b-树:会把数据行存储在中间节点中,所以导致节点中能存储的数据太少
b+树:中间节点不存放数据.innodb,myisam都是基于b+树创建索引
什么决定树的高度?
    数据的量和数据的长度
什么是索引?
把数据的某个字段按照特殊的算法计算成一个树型结构,再根据树型结构提供的指针缩小范围,找到对应的磁盘块.通过这棵树,可以将我们每次的查询范围缩小1/3,加快了我们的查询速度.这棵树就是索引.
```

33.查询:

1.查询出每门课程都大于80分的学生姓名

```
select name from tablea group by(name) having min(fenshu) > 80;
```

2.查询语文成绩最大的学生姓名

```
select name from tablea where fenshu=(select MAX(fenshu) from tablea where kecheng='语文');
```

3.查询没有成绩的学生姓名

```
select name from tableb where name not in (select name from tablea);
```

34.试列出至少三种目前流行的大型关系型数据库的名称. 其中你最熟悉的是? 什么时候开始使用?

```
排名:https://db-engines.com/en/ranking
目前流行的大型关系型数据库:Oracle,MySQL,Microsoft SQL Server,IBM Db2,SQLite
```

35.字段类型都是整数

1.查询出b和c列的值,要求按照b列的升序排列

2.写入一条新的记录,值为(7,9,8)

3.查询c列,要求消除重复的值,按降序排列

```
1.  select b,c from t3 order by(b);
2.  insert into t3 (a,b,c) values(7,9,8);
3. select distinct c from t3 order by(c) desc;
```

36.用一条sql语句查询出每门课程都大于80分的学生姓名

同33题

37.设计表,关系如下:教师,班级,学生,科室.科室与教师为一对多关系,教师与班级为多对多关系,班级与学生为一对多关系,科室中需体现层级关系:

1. 写出各张表的逻辑字段
2. 根据上述关系表:
    1. 查询教师id=1的学生数
    2. 查询科室id=3的下级部门数
    3. 查询所带学生最多的教师的id

```
1. 教师
   id  name

   班级
   id  name    teacher_id

   学生
   id  name    class_name

   科室
   id  name    super_dep_id
2. 1. select count(id) from student having class_name in (select name
from class where  ppteacher_id = 1)
   2.select count(id) from department where super_dep_id = 3
```

39. 什么是MySQL慢日志?

MySQL的慢查询日志是MySQL提供的一种日志记录，它用来记录在MySQL中响应时间超过阀值的语句，具体指运行时间超过long_query_time值的SQL，则会被记录到慢查询日志中。long_query_time的默认值为10，意思是运行10S以上的语句。默认情况下，Mysql数据库并不启动慢查询日志，需要我们手动来设置这个参数，当然，如果不是调优需要的话，一般不建议启动该参数，因为开启慢查询日志会或多或少带来一定的性能影响。慢查询日志支持将日志记录写入文件，也支持将日志记录写入数据库表。

41.在对name做了唯一索引前提下，简述以下区别：

```
1.select * from tb where name = "Oldboy-Wupeiqi"
2.select * from tb where name = "Oldboy-Wupeiqi" limit 1
```

如果是唯一索引的话两者本质上没有什么区别，都是查询到一条数据后就不往下查询了，但是如果不是唯一索引的前提下，第二种加limit的当查询到一条数据后就不往下执行了，而第一种还是需要继续查询

42.Redis 和 Memecached 的区别?

共同点：Redis 和 Memcached 都是基于内存的数据存储系统。在内存中存储数据，防止高并发影响数据库性能，减少数据库压力，并提高查询数据速度。

1）数据类型支持不同

Memcached 只支持简单的 Key-Value 结构的数据记录 。

Redis 支持更多的数据结构和更丰富的数据操作。最为常用的数据类型主要有5种：String（字符串类型）、Hash（哈希类型）、List（列表类型）、Set（集合类型）和SortedSet（有序集合类型）。

2）存储方式

Memcached 只能将数据缓存到内存中，无法自动定期写入硬盘，一旦断电或重启，内存清空，数据丢失。（Memcached的应用场景适应于 缓存无需持久化的数据）

Redis 也是基于内存的存储系统，但是它本身支持内存数据的持久化（可以对Redis进行配置，周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，实现数据的持久化。） 提供两种主要的持久化策略：RDB 快照和 AOF 日志。

3）内存管理机制不同

在 Redis 中，并不是所有的数据都一直存储在内存中的。这是和 Memcached 相比一个最大的区别。当物理内存用完时，Redis 可以将一些很久没有用到的 Value 交换到磁盘。Redis 采用的是包装的 mallc/free，相较于 Memcached 的内存管理方法来说，要简单得多。

Memcached 默认使用 Slab Allocation 机制管理内存，其主要思想是按照预先规定的大小，将分配的内存分割成特定长度的块以存储相应长度的 key-value 数据记录，以完全解决内存碎片问题，但是它最大的缺点就是会导致空间浪费。

4）集群管理不同

Memcached是全内存的数据缓冲系统，Redis虽然支持数据的持久化，但是全内存毕竟才是其高性能的本质。作为基于内存的存储系统来说，机器物理内存的大小就是系统能够容纳的最大数据量。如果需要处理的数据量超过了单台机器的物理内存大小，就需要构建分布式集群来扩展存储能力。

Memcached本身并不支持分布式，因此只能在客户端通过像一致性哈希这样的分布式算法来实现Memcached的分布式存储。相较于Memcached只能采用客户端实现分布式存储，Redis更偏向于在服务器端构建分布式存储。

网络上有这么解释的：

1.存储容量：

memcached超过内存比例会抹掉前面的数据，而redis会存储在磁盘

2.支持数据类型：

memcached只支持string；

redis支持更多;如：hash、list、集合、有序集合

3.持久化：

redis支持数据持久化，可以将内存中的数据保持在磁盘中，memcached无

4.主从：

即master-slave模式的数据备份(主从)。

5.特性

Redis在很多方面具备数据库的特征，或者说就是一个数据库系统，Memcached只是简单的K/V缓存

43.如何高效的找到 redis 中所有以 oldboy 开头的 key?

```
# 命令模式
keys *           # *代表通配符
keys oldboy*     #  oldboy* 代表以所有以 olodboy 开头的 key

# redis模块
import redis
con = redis.Redis()
con.keys(pattern='oldboy*')     # *代表通配符
```

44.什么是一致性哈希？Python中是否有相应的模块及其代码实现?

一致性哈希： Distributed Hash Table

一致性hash算法（DHT）可以通过减少影响范围的方式，解决增减服务器导致的数据散列问题，从而解决了分布式环境下负载均衡问题；

如果存在热点数据，可以通过增添节点的方式，对热点区间进行划分，将压力分配至其他服务器，重新达到负载均衡的状态。

一致性哈希可以有效地解决分布式存储结构下动态增加和删除节点所带来的问题

一致性哈希的实现思路：

1.把机器按照某种hash算法（比如MD5）计算得到机器的hashcode值

2.对于存储的数据，根据数据的key，使用与机器相同的hash算法获取到相应的hashcode值，然后将key写入到顺时针最近的机器。

可以是hashcode(key) <= hashcode(machine)的机器

3.有新机器加入时，只需要把新加入机器影响到的数据进行重新分配；当删除机器时，只需要把被删除机器的数据重新分配一下，这样可以减小数据的迁移代价。

4.为了维持平衡性，防止雪崩效应，使用虚拟节点代替真实机器，一个真实机器对应多个虚拟节点，这样可以保证数据的分布均衡

模块： hash_ring

```
#! /usr/bin/env python
# -*- coding:utf8 -*-
```

参考：https://www.jianshu.com/p/49e3fbf41b9b

45.redis 是单进程单线程的吗?

Redis 是单进程单线程，线程安全。

Redis 采用的是基于内存的采用的是单进程单线程模型的 KV（键值） 数据库，由 C 语言编写。官方提供的数据是可以达到100000+的 qps（每秒查询率）。这个数据不比采用单进程多线程的同样基于内存的 KV 数据库 Memcached 差。

Redis可以能够快速执行的原因：

(1) 绝大部分请求是纯粹的内存操作（非常快速）

(2) 数据结构简单，对数据操作也简单

(3) 采用单线程,避免了不必要的上下文切换和竞争条件 减少开销

(4) 非阻塞IO - IO多路复用（IO 多路复用是什么意思？）

IO多路复用中有三种方式：select,poll,epoll。需要注意的是，select,poll是线程不安全的，epoll是线程安全的。

Redis内部实现采用epoll，采用了epoll+自己实现的简单的事件框架。epoll中的读、写、关闭、连接都转化成了事件，然后利用epoll的多路复用特性，绝不在io上浪费一点时间。

单线程指的是网络请求模块使用了一个线程（所以不需考虑并发安全性），即一个线程处理所有网络请求，其他模块仍用了多个线程。

多路 I/O 复用模型是利用 select、poll、epoll 可以同时监察多个流的 I/O 事件的能力，在空闲的时候，会把当前线程阻塞掉，当有一个或多个流有 I/O 事件时，就从阻塞态中唤醒，于是程序就会轮询一遍所有的流（epoll 是只轮询那些真正发出了事件的流），并且只依次顺序的处理就绪的流，这种做法就避免了大量的无用操作。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。采用多路 I/O 复用技术可以让单个线程高效的处理多个连接请求（尽量减少网络 IO 的时间消耗），且 Redis 在内存中操作数据的速度非常快（内存内的操作不会成为这里的性能瓶颈），主要以上两点造就了 Redis 具有很高的吞吐量。

提示：

Redis的网络模型是一个 单线程epoll模型 （网络请求模块使用了一个线程，即一个线程处理所有网络请求，其他模块仍使用了多个线程），但是不见得处理过程始终单进程哦，有些处理可能会阻塞服务器，还是会开进程处理的：比如在保存RDB文件时候，
https://github.com/antirez/redis/blob/unstable/src/rdb.c#L1059，还有sentinel，
https://github.com/antirez/redis/blob/unstable/src/sentinel.c#L754

因为CPU不是Redis的瓶颈。Redis的瓶颈最有可能是机器内存或者网络带宽，既然单线程容易实现，而且CPU不会成为瓶颈，那就顺理成章地采用单线程的方案了。关于redis的性能，官方网站也有，普通笔记本轻松处理每秒几十万的请求。

46.Redis 中数据库默认是多少个 db 及作用?

Redis 默认会创建 16 个 db。
• Redis 下，数据库是由一个整数索引标识（0-15），而不是由一个数据库名称。默认情况下，一个客户端连接到 0 号数据库。
•可以通过修改 redis 配置文件（/etc/redis/redis.conf）中的参数来数据库总数。配置项：
databases = 16 // 默认有 16 个数据库
•每个数据库的数据是隔离的不能共享
•可以随时使用 SELECT 命令来切换数据库。 redis> select 10 # 切换到 10 号 数据库
•注意： FLUSHALL 命令可以清空一个 Redis 示例中所有数据库中的数据

47.如果 redis 中的某个列表中的数据量非常大，如何实现循环显示每一个值?

```
def list_iter(name):
    """
    自定义redis列表增量迭代
    :param name:redis中的name，即：迭代name对应的列表
    :return: yield 返回列表元素
    """
    list_count =r.llen(name)
    for index in xrange(list_count):
        yield r.lindex(name, index)

# 通过scan_iter分片取，减少内存压力
scan_iter(match=None, count=None)增量式迭代获取redis里匹配的的值
# match，匹配指定key
# count，每次分片最少获取个数
    r =redis.Redis(connection_pool=pool)
    for key in r.scan_iter(match='PREFIX_*',count=100000):
        print(key)
```

48.redis 如何实现主从复制？ 以及数据同步机制?

实现主从复制：
        # 创建6379和6380配置文件
        redis.conf：6379为默认配置文件，作为Master服务配置 主
        redis_6380.conf：6380为同步配置，作为Slave服务配置 从
        # 配置slaveof同步指令
        在Slave对应的conf配置文件中，添加以下内容：
        slaveof 127.0.0.1 6379
数据同步步骤：
        (1) Slave 服务器连接到 Master 服务器
        (2) Slave 服务器发送同步(SYCN)命令
        (3) Master 服务器备份数据库到文件
        (4) Master 服务器把备份文件传输给 Slave 服务器
        (5) Slave 服务器把备份文件数据导入到数据库中

49.Redis 中的 sentinel （哨兵）的作用?

Redis Sentinel 的作用是 为 Redis 提供了高可用的实现。通俗来讲，Sentinel 能监控多个 Redis 的 master-slave 集群，发现 master 宕机后可以不用人为干预而自动进行故障转移 。

Redis-Sentinel是Redis官方推荐的高可用性(HA)解决方案，当用Redis做Master-slave的高可用方案时，假如 master宕机了，Redis本身(包括它的很多客户端)都没有实现自动进行主备切换，而Redis-sentinel本身也是一个独立运行的进程，它 能监控多个master-slave集群，发现master宕机后能进行自动切换。

它的主要功能有以下几点：
1.监控。不时地监控redis是否按照预期良好地运行；
2.通知。如果发现某个redis节点运行出现状况，能够通知另外一个进程(例如它的客户端)；
3.自动故障转移：能够进行自动切换。当一个master节点不可用时，能够选举出master的多个slave(如果有超过一个slave的话)中的一个来作为新的master,其它的slave节点会将它所追随的master的地址改为被提升为master的slave的新地址。
4.配置提供者。哨兵作为Redis客户端发现的权威来源：客户端连接到哨兵请求当前可靠的master的地址。如果发生故障，哨兵将报告新地址。

sentinel 的分布式特性
很显然，只使用单个sentinel进程来监控redis集群是不可靠的，当sentinel进程宕掉后(sentinel本身也有单点问题，single-point-of-failure)整个集群系统将无法按照预期的方式运行。所以有必要将sentinel集群，这样有几个好处：
•即使有一些sentinel进程宕掉了，依然可以进行redis集群的主备切换；
•如果只有一个sentinel进程，如果这个进程运行出错，或者是网络堵塞，那么将无法实现redis集群的主备切换（单点问题）；
•如果有多个sentinel，redis的客户端可以随意地连接任意一个sentinel来获得关于redis集群中的信息。

50.如何实现 redis 集群?

基于【分片】来完成。

- 集群是将你的数据拆分到多个Redis实例的过程
- 可以使用很多电脑的内存总和来支持更大的数据库。
- 没有分片，你就被局限于单机能支持的内存容量。

redis将所有能放置数据的地方创建了 16384 个哈希槽。
如果设置集群的话，就可以为每个实例分配哈希槽：
- 192.168.1.20【0-5000】
- 192.168.1.21【5001-10000】
- 192.168.1.22【10001-16384】

以后想要在redis中写值时：set k1 123
- 将k1通过crc16的算法转换成一个数字，然后再将该数字和16384求余，
- 如果得到的余数 3000，那么就将该值写入到 192.168.1.20 实例中。

集群方案：
- redis cluster：官方提供的集群方案。
- codis：豌豆荚技术团队。
- tweproxy：Twiter技术团队。

51.redis中默认有多少哈希槽

Redis 集群中内置了 16384 个哈希槽，当需要在 Redis 集群中放置一个 key-value时，redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节点。
Redis 集群没有使用一致性hash，而是引入了哈希槽的概念。
Redis 集群有16384个哈希槽,每个key通过CRC16校验后对16384取模来决定放置哪个槽.集群的每个节点负责一部分hash槽。这种结构很容易添加或者删除节点，并且无论是添加删除或者修改某一个节点，都不会造成集群不可用的状态。
使用哈希槽的好处就在于可以方便的添加或移除节点。
当需要增加节点时，只需要把其他节点的某些哈希槽挪到新节点就可以了；
当需要移除节点时，只需要把移除节点上的哈希槽挪到其他节点就行了；
在这一点上，我们以后新增或移除节点的时候不用先停掉所有的 redis 服务。
"用了哈希槽的概念，而没有用一致性哈希算法，不都是哈希么？这样做的原因是为什么呢？"
Redis Cluster是自己做的crc16的简单hash算法，没有用一致性hash。Redis的作者认为它的crc16(key) mod 16384的效果已经不错了，虽然没有一致性hash灵活，但实现很简单，节点增删时处理起来也很方便。
"为了动态增删节点的时候，不至于丢失数据么？"
节点增删时不丢失数据和hash算法没什么关系，不丢失数据要求的是一份数据有多个副本。
“还有集群总共有2的14次方，16384个哈希槽，那么每一个哈希槽中存的key 和 value是什么？”
当你往Redis Cluster中加入一个Key时，会根据crc16(key) mod 16384计算这个key应该分布到哪个hash slot中，一个hash slot中会有很多key和value。你可以理解成表的分区，使用单节点时的redis时只有一个表，所有的key都放在这个表里；改用Redis Cluster以后会自动为你生成16384个分区表，你insert数据时会根据上面的简单算法来决定你的key应该存在哪个分区，每个分区里有很多key。

52.redis 有哪几种持续化策略及比较

Redis 提供了多种不同级别的持久化方式：
RDB 持久化可以在指定的时间间隔内生成数据集的时间点快照(point-in-time snapshot)。

AOF 持久化记录服务器执行的所有写操作命令，并在服务器启动时，通过重新执行这些命令来还原数据集。 AOF 文件中的命令全部以 Redis 协议的格式来保存，新命令会被追加到文件的末尾。
Redis 还可以在后台对 AOF 文件进行重写(rewrite)，使得 AOF 文件的体积不会超出保存数据集状态所需的实际大小。
Redis 还可以同时使用 AOF 持久化和 RDB 持久化。 在这种情况下， 当 Redis 重启时， 它会优先使用 AOF 文件来还原数据集， 因为 AOF 文件保存的数据集通常比 RDB 文件所保存的数据集更完整。
你甚至可以关闭持久化功能，让数据只在服务器运行时存在。

RDB 的优点
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。
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。
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。
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

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。
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
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)。
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。
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。
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
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。
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)。
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。

53.redis 有哪几种过期策略?

maxmemory-policy 六种方式
volatile-lru：只对设置了过期时间的key进行LRU（默认值）
allkeys-lru ： 删除lru算法的key
volatile-random：随机删除即将过期key
allkeys-random：随机删除
volatile-ttl ： 删除即将过期的
noeviction ： 永不过期，返回错误

54.mysql里有2000w条数据,redis中只有20w条,如何保证redis中都是热点数据?

第一种：
比如用户数据。数据库有2000w条。
活跃用户：
redis sortSet里 放两天内(为方便取一天内活跃用户)登录过的用户，登录一次ZADD一次，如set已存在则覆盖其分数（登录时间）。键：login:users，值：分数 时间戳、value userid。
设置一个周期任务，比如每天03:00:00点删除sort set中前一天3点前的数据（保证set不无序增长、
留近一天内活跃用户）。
取时，拿到当前时间戳（int 10位），再减1天就可按分数范围取过去24h活跃用户。
==========
第二种：
提供一种简单实现缓存失效的思路： LRU(最近少用的淘汰)
即redis的缓存每命中一次,就给命中的缓存增加一定ttl(过期时间)(根据具体情况来设定，比如10分钟).
一段时间后，热数据的ttl都会较大，不会自动失效，而冷数据基本上过了设定的ttl就马上失效了.

55.基于redis的列表,实现先进先出,后进先出,优先队列?

56.基于redis实现消息队列

1.Redis实现消息队列原理
常用的消息队列有RabbitMQ,ActiveMQ，个人觉得这种消息队列太大太重，本文介绍下基于redis的轻量级消息队列服务。
一般来说，消息队列有两种模式，一种是发布者订阅模式，另外一种是生产者和消费者模式。Redis的消息队列，也是基于这2种原理的实现。
发布者和订阅者模式：发布者发送消息到队列，每个订阅者都能收到一样的消息。
生产者和消费者模式：生产者将消息放入队列，多个消费者共同监听，谁先抢到资源，谁就从队列中取走消息去处理。注意，每个消息只能最多被一个消费者接收。

2.Redis消息队列使用场景
在我们的项目中，使用消息队列来实现短信的服务化，任何需要发送短信的模块，都可以直接调用短信服务来完成短信的发送。比如用户系统登录注册短信，订单系统的下单成功的短信等。

3.SpringMVC中实现Redis消息队列
因为我们短信只需要发送一次，所以我们使用的是消息队列的生产者和消费者模式。

57.基于redis实现发布和订阅

一个基于消息订阅的模式，用来对非定时的的消息进行监听订阅。
Redis提供了发布订阅功能，可以用于消息的传输，Redis的发布订阅机制包括三个部分，发布者，订阅者和Channel。
https://blog.csdn.net/changhuzhao/article/details/79362140

58.什么是codis

https://blog.csdn.net/shmiluwei/article/details/51958359

59.什么是twemproxy

Twemproxy是一个代理服务器，可以通过它减少Memcached或Redis服务器所打开的连接数。

Twemproxy的用途如下：
通过代理的方式减少缓存服务器的连接数
自动在多台缓存服务器间共享数据
通过不同的策略与散列函数支持一致性散列
通过配置的方式禁用失败的结点
运行在多个实例上，客户端可以连接到首个可用的代理服务器
支持请求的流式与批处理，因而能够降低来回的消耗

60.redis如何实现事务

Redis 事务可以一次执行多个命令， 并且带有以下两个重要的保证：

批量操作在发送 EXEC 命令前被放入队列缓存。
收到 EXEC 命令后进入事务执行，事务中任意命令执行失败，其余的命令依然被执行。
在事务执行过程，其他客户端提交的命令请求不会插入到事务执行命令序列中。
一个事务从开始到执行会经历以下三个阶段：

开始事务。
命令入队。
执行事务。

61.redis中的watch命令的作用

watch 用于在进行事务操作的最后一步也就是在执行exec 之前对某个key进行监视
如果这个被监视的key被改动，那么事务就被取消，否则事务正常执行.
一般在MULTI 命令前就用watch命令对某个key进行监控.如果想让key取消被监控，可以用unwatch命令

62.redis分布式锁和redlock的实现机制

https://blog.csdn.net/andy86869/article/details/81668355

64.了解过hbase,DB2，SQLServer,Access吗?

HBASE是一个数据库----可以提供数据的实时随机读写
HBASE与mysql、oralce、db2、sqlserver等关系型数据库不同，它是一个NoSQL数据库（非关系型数据库）