搜狐面试题

1、写个单例，什么是单例，单例怎么实现，怎么理解单例

最简单的单例就是一个模块：

```
class foo(object):
    def fin(self):
        pass
A = foo()
```

答：单例模式（SingletonPattern）是一种常用的软件设计模式，该模式的主要目的是确保某一个类只有一个实例存在。当你希望在整个系统中，某个类只能出现一个实例时，单例对象就能派上用场。

把你的单例的代码保存在一个py文件当中，要使用时，直接在本文件中导入其他文件的对象，这个对象即是单例模式的对象。

2、什么是websocket，socket、poll、epoll的区别，他们的机制是什么，有什么区别

websocket传输协议是基于tcp传输的一种网络协议，它实现了服务器和浏览器的全双工的通信，WebSocket协议之前，双工通信是通过多个http链接来实现，这导致了效率低下。WebSocket解决了这个问题

基于自己的理解：socket用的是1024个字节传输数据，poll超越了传输数据的限制，不过用的是轮询机制，每次需要就问一次，效率太低，而epoll就使用的是事务提交机制，就是你有什么问题，你先进行提交，

3、redis的连接方式

引入redis的原因：内存机的读写，加快读写的速度，所以引入了非关系型数据库 访问的人越来越多，重要的节点服务器的访问量急剧上升

1 加上 & 号使redis以后台程序方式运行 ./redis-server &
2 启动时指定配置文件
redis-server ./redis.conf
3 使用Redis启动脚本设置开机自启动
推荐在生产环境中使用启动脚本方式启动redis服务。启动脚本redis_init_script 位于位于Redis的 /utils/ 目录下。

1. 根据启动脚本要求，将修改好的配置文件以端口为名复制一份到指定目录。需使用root用户。
   mkdir /etc/redis cp redis.conf /etc/redis/6379.conf

2. 将启动脚本复制到/etc/init.d目录下，本例将启动脚本命名为redisd（通常都以d结尾表示是后台自启动服务）。 cp redis_init_script /etc/init.d/redisd
3. 设置为开机自启动 #设置为开机自启动服务器 chkconfig redisd on #打开服务 service redisd start

4、怎么理解事务，mysql中的锁都有哪些，各个锁的机制是什么

MySQL以及大多数关系型数据库都提供了一个叫做事务的技术。我们可以声明一个事务的开始，在确认提交或者指明放弃前的所有操作，都先在一个叫做事务日志的临时环境中进行操作。待操作完成，确保了数据一致性之后，那么我们可以手动确认提交，也可以选择放弃以上操作。

一旦选择了提交，那么便不能再利用放弃操作来撤销更改了。

数据库锁定机制简单来说，就是数据库为了保证数据的一致性，而使各种共享资源在被并发访问变得有序所设计的一种规则

MySQL各存储引擎使用了三种类型（级别）的锁定机制：表级锁定，行级锁定和页级锁定。

1.表级锁定（table-level）
表级别的锁定是MySQL各存储引擎中最大颗粒度的锁定机制。该锁定机制最大的特点是实现逻辑非常简单，带来的系统负面影响最小。所以获取锁和释放锁的速度很快。由于表级锁一次会将整个表锁定，所以可以很好的避免困扰我们的死锁问题。 当然，锁定颗粒度大所带来最大的负面影响就是出现锁定资源争用的概率也会最高，致使并大度大打折扣。 使用表级锁定的主要是MyISAM，MEMORY，CSV等一些非事务性存储引擎。

2.行级锁定（row-level） 行级锁定最大的特点就是锁定对象的颗粒度很小，也是目前各大数据库管理软件所实现的锁定颗粒度最小的。由于锁定颗粒度很小，所以发生锁定资源争用的概率也最小，能够给予应用程序尽可能大的并发处理能力而提高一些需要高并发应用系统的整体性能。 虽然能够在并发处理能力上面有较大的优势，但是行级锁定也因此带来了不少弊端。由于锁定资源的颗粒度很小，所以每次获取锁和释放锁需要做的事情也更多，带来的消耗自然也就更大了。此外，行级锁定也最容易发生死锁。 使用行级锁定的主要是InnoDB存储引擎。

3.页级锁定（page-level） 页级锁定是MySQL中比较独特的一种锁定级别，在其他数据库管理软件中也并不是太常见。页级锁定的特点是锁定颗粒度介于行级锁定与表级锁之间，所以获取锁定所需要的资源开销，以及所能提供的并发处理能力也同样是介于上面二者之间。另外，页级锁定和行级锁定一样，会发生死锁。

总的来说，MySQL这3种锁的特性可大致归纳如下： 表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低； 行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高；
页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。 适用：从锁的角度来说，表级锁更适合于以查询为主，只有少量按索引条件更新数据的应用，如Web应用；而行级锁则更适合于有大量按索引条件并发更新少量不同数据，同时又有并发查询的应用，如一些在线事务处理（OLTP）系统。

5、怎么优化数据库，你做过的数据库优化有哪些，还有哪些方式可以优化

1、对语句的优化

①用程序中，保证在实现功能的基础上，尽量减少对数据库的访问次数； 通过搜索参数，尽量减少对表的访问行数,最小化结果集，从而减轻网络负担； ②能够分开的操作尽量分开处理，提高每次的响应速度；在数据窗口使用SQL时，尽量把使用的索引放在选择的首列；算法的结构尽量简单； ③在查询时，不要过多地使用通配符如SELECT * FROM T1语句，要用到几列就选择几列如：SELECT COL1,COL2 FROM T1； ④在可能的情况下尽量限制尽量结果集行数如：SELECT TOP 300 COL1,COL2,COL3 FROM T1,因为某些情况下用户是不需要那么多的数据的。 ⑤不要在应用中使用数据库游标，游标是非常有用的工具，但比使用常规的、面向集的SQL语句需要更大的开销；按照特定顺序提取数据的查找。

2、 避免使用不兼容的数据类型。 例如float和int、char和varchar、binary和varbinary是不兼容的。 数据类型的不兼容可能使优化器无法执行一些本来可以进行的优化操作

3、 尽量避免在WHERE子句中对字段进行函数或表达式操作。 若进行函数或表达式操作，将导致引擎放弃使用索引而进行全表扫描

```
SELECT * FROM T1 WHERE F1/2=100
```

改为

```
SELECT * FROM T1 WHERE F1=100/2
```

4、 避免使用!=或<>、IS NULL或IS NOT NULL、IN ， NOT IN等这样的操作符
因为这会使系统无法使用索引,而只能直接搜索表中的数据,优化器将无法通过索引来确定将要命中的行数,因此需要搜索该表的所有行

5、 尽量使用数字型字段 一部分开发人员和数据库管理人员喜欢把包含数值信息的字段设计为字符型，这会降低查询和连接的性能， 并会增加存储开销。这是因为引擎在处理查询和连接回逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了

6、Django除了runserver，还有没有其他方式启动，Django怎么维护

除了这一种方式，还有另外一种方式，uWSGI+Nginx的方法是现在最常见的在生产环境中运行Django的方法 WSGI，全称Web Server Gateway Interface，或者Python Web Server Gateway Interface，是为Python语言定义的Web服务器和Web应用程序或框架之间的一种简单而通用的接口，基于现存的CGI标准而设计的。WSGI其实就是一个网关(Gateway)，其作用就是在协议之间进行转换。(PS: 这里只对WSGI做简单介绍，想要了解更多的内容可自行搜索)

uWSGI是一个Web服务器，它实现了WSGI协议、uwsgi、http等协议。注意uwsgi是一种通信协议，而uWSGI是实现uwsgi协议和WSGI协议的Web服务器。uWSGI具有超快的性能、低内存占用和多app管理等优点

7、并发量很大时，接口能抗多大压，测压测过吗，怎么实现

8、怎么处理用进程池处理大量用户同时访问

Pool类可以提供指定数量的进程供用户调用，当有新的请求提交到Pool中时，如果池还没有满，就会创

建一个新的进程来执行请求。如果池满，请求就会告知先等待，直到池中有进程结束，才会创建新的进程来执行这些请求。

```
import time
from multiprocessing import Pool
def run(fn):
  #fn: 函数参数是数据列表的一个元素
  time.sleep(1)
  return fn*fn

if __name__ == "__main__":
  testFL = [1,2,3,4,5,6]
  print 'shunxu:' #顺序执行(也就是串行执行，单进程)
  s = time.time()
  for fn in testFL:
    run(fn)

  e1 = time.time()
  print "顺序执行时间：", int(e1 - s)

  print 'concurrent:' #创建多个进程，并行执行
  pool = Pool(5)  #创建拥有5个进程数量的进程池
  #testFL:要处理的数据列表，run：处理testFL列表中数据的函数
  rl =pool.map(run, testFL)
  pool.close()#关闭进程池，不再接受新的进程
  pool.join()#主进程阻塞等待子进程的退出
  e2 = time.time()
  print "并行执行时间：", int(e2-e1)
  print rl
```

上例是一个创建多个进程并发处理与顺序执行处理同一数据，所用时间的差别。从结果可以看出，并发执行的时间明显比顺序执行要快很多，但是进程是要耗资源的，所以平时工作中，进程数也不能开太大。

9、订单价格，怎么保证前端传过来的价格是正确的，不是被抓包后修改的，怎么解决，有没有两三种方案

10、秒杀的实现过程，先接收用户请求，后端怎么异步处理，怎么告诉前端，如果有多个进程同时访问，怎么处理同时过来的请求，怎么避免并发，有几种办法，各种方法的实现的机制是什么

11、3.5和3.6版本的python有什么区别

主要是关于json模块的区别

```
>>> import json
>>> a = b'{"username": "xxx"}'
>>> c = json.loads(a)
```

3.5解决办法:

```
>>> a = b'123'
>>> c = json.loads(a)
>>> c = json.loads(a.decode('utf-8'))
```

无论bytes类型或者str类型都可以反序列化

3.6解决方法

```
>>> import json
>>> a = b'{"username": "xxx"}'
>>> c = json.loads(a)
>>> g = b'{"username": "xxx"}'
>>> h = json.loads(g.decode("utf-8"))
```