目录

NIO的优点
创建线程有几种不同的方式？你喜欢哪一种？为什么？
线程池
竞态条件
概括的解释下线程的几种可用状态
用什么关键字修饰同步方法?
stop()和suspend()方法为何不推荐使用?
什么是ThreadLocal?
run()和start()区别
sleep() 和 wait() 有什么区别?
当一个线程进入一个对象的一个synchronized方法后，其它线程是否可进入此对象的其它方法?
请说出你所知道的线程同步的方法
线程调度和线程控制
JMM
同步和异步有何异同，在什么情况下分别使用他们？举例说明
什么是线程饿死，什么是活锁?
多线程中的忙循环是什么?
简述synchronized和java.util.concurrent.locks.Lock的异同？
同步方法和同步代码块区别：
如何确保N个线程可以访问N个资源同时又不导致死锁?
什么是原子操作
volatile 变量是什么？volatile 变量和 atomic 变量有什么不同
volatile 类型变量提供什么保证？能使得一个非原子操作变成原子操作吗
能创建 volatile 数组吗?
一张表，里面有ID自增主键，当insert了17条记录之后，删除了第15,16,17条记录，再把Mysql重启，再insert一条记录，这条记录的ID是18还是15？
Heap表是什么?
Mysql服务器默认端口是什么?
与Oracle相比，Mysql有什么优势?
区分CHAR_LENGTH和LENGTH?
请简洁描述Mysql中InnoDB支持的四种事务隔离级别名称，以及逐级之间的区别?
在Mysql中ENUM的用法是什么?
如何定义REGEXP?
CHAR和VARCHAR的区别?
列的字符串类型可以是什么?
如何获取当前的Mysql版本?
Mysql中使用什么存储引擎?
TIMESTAMP在UPDATE CURRENT_TIMESTAMP数据类型上做什么?
主键和候选键有什么区别?
如何使用Unix shell登录Mysql?
myisamchk是用来做什么的?
MYSQL数据库服务器性能分析的方法命令有哪些?
如何控制HEAP表的最大尺寸?
MyISAM Static和MyISAM Dynamic有什么区别?
federated表是什么?
如果一个表有一列定义为TIMESTAMP，将发生什么?
列设置为AUTO INCREMENT时，如果在表中达到最大值，会发生什么情况?
怎样才能找出最后一次插入时分配了哪个自动增量?
你怎么看到为表格定义的所有索引?

LIKE声明中的%和_是什么意思?
如何在Unix和Mysql时间戳之间进行转换?
列对比运算符是什么?
我们如何得到受查询影响的行数?
Mysql查询是否区分大小写?
LIKE和REGEXP操作有什么区别?
BLOB和TEXT有什么区别?
mysql_fetch_array和mysql_fetch_object的区别是什么?
数据库的三范式?
我们如何在mysql中运行批处理模式?
MyISAM表格将在哪里存储，并且还提供其存储格式?
Mysql中有哪些不同的表格?
ISAM是什么?
InnoDB是什么?
Mysql如何优化DISTINCT?
如何显示前50行?
可以使用多少列创建索引?
NOW（）和CURRENT_DATE（）有什么区别?
什么样的对象可以使用CREATE语句创建?
Mysql表中允许有多少个TRIGGERS?
什么是非标准字符串类型?
什么是通用SQL函数?
解释访问控制列表
MYSQL支持事务吗?
mysql里记录货币用什么字段类型好
MYSQL数据表在什么情况下容易损坏?
mysql有关权限的表都有哪几个?
Mysql中有哪几种锁?
Mysql数据优化
MySQL的关键字
存储引擎
数据库备份
如何显示创表语句以及给表中添加字段
SQL语言分类
truncate delete drop的区别:
说出ArrayList,Vector, LinkedList的存储性能和特性
HashMap和Hashtable的区别
快速失败(fail-fast)和安全失败(fail-safe)的区别是什么?
hashmap的数据结构
HashMap的工作原理是什么?
Hashmap什么时候进行扩容呢?
CorrentHashMap的工作原理?
Java集合类框架的基本接口有哪些?
HashSet和TreeSet有什么区别?
HashSet的底层实现是什么?
LinkedHashMap的实现原理?
为什么集合类没有实现Cloneable和Serializable接口?
什么是迭代器(Iterator)?
Iterator和ListIterator的区别是什么?

数组(Array)和列表(ArrayList)有什么区别？什么时候应该使用Array而不是 ArrayList?
Java集合类框架的最佳实践有哪些?
Set里的元素是不能重复的，那么用什么方法来区分重复与否呢? 是用==还是equals()? 它们有何区别
Comparable和Comparator接口是干什么的？列出它们的区别。
Collection和Collections的区别
java内存分配
Java堆的结构是什么样子的？什么是堆中的永久代(Perm Gen space)?
GC是什么? 为什么要有GC
简述java垃圾回收机制?
如何判断一个对象是否存活?(或者GC对象的判定方法)
垃圾回收的2种回收机制
垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收?
System.gc()和Runtime.gc()会做什么事情?
finalize()方法什么时候被调用？析构函数(finalization)的目的是什么?
如果对象的引用被置为null，垃圾收集器是否会立即释放对象占用的内存?
什么是分布式垃圾回收(DGC)？它是如何工作的?
串行(serial)收集器和吞吐量(throughput)收集器的区别是什么?
在Java中，对象什么时候可以被垃圾回收?
分代
Gc的基本概念
简述java内存分配与回收策率以及Minor GC和Major GC
JVM的永久代中会发生垃圾回收么?
java中垃圾收集的方法有哪些?
java中会存在内存泄漏吗，请简单描述。
java类加载过程?
简述java类加载机制?
什么是类加载器，类加载器有哪些?
类加载器双亲委派模型机制?
访问修饰符public,private,protected,以及不写（默认）时的区别?
如何阻止Java中的类型未检查的警告?
抽象类和接口
面向对象软件开发的优点有哪些?
封装的定义和好处有哪些?
多态的定义?
继承的定义?
抽象的定义？抽象和封装的不同点?
N层架构
3尾递归
Final和static
能否在运行时向static final类型的变量赋值
throws, throw, try, catch, finally分别代表什么意义
3HTTP请求的GET与POST方式的区别
JSP的常用指令
xml有哪些解析技术?区别是什么?
XML文档定义有几种形式？它们之间有何本质区别?
你在项目中用到了xml技术的哪些方面?如何实现的?
log4j日志

什么是RESTful架构：
Redis是什么？两句话做下概括
redis（管道，哈希）
redis实现原理或机制
Redis 有两种类型分区
Mongo DB
Mongo DB特点
Mongo DB功能
Mongo DB适用场景
redis、memcache、mongoDB 对比
Redis有什么用？只有了解了它有哪些特性，我们在用的时候才能扬长避短，为我们所用：
这里对Redis数据库做下小结

基本数据类型的长度

> char byte 1个字节 8位 -2^7--2^7-1
>
> Short 2个字节 16位 2^15--2^15-1
>
> Int float 4个字节 32位 2^31--2^31-1
>
> Double long 8个字节 64位 2^63--2^63-1

反射

> 可以在程序运行时修改或访问它本身方法的能力

ACID

> 原子性、一致性、事务性、持久性

MVC

> M:模型层，javabean、Hibernate、Mybatis
>
> V:视图层：JSP、HTML、 C:控制层：Struts、SpringMVC

RPC和RMI

> RPC：远程过程调用 面向过程 C++
>
> RMI：远程方法调用 面向对象 java

常见的RPC框架Dubbo

> 这种框架的内部一般维护了请求的协议和请求号，可以维护一个请求号为key，请求的结果为Result的future的map，结合NIO和长连接，获取非常不错的性能。
>
> - **RMI中使用RMI安全管理器(RMISecurityManager)的目的是什么?**
>
> RMISecurityManager使用下载好的代码提供可被RMI应用程序使用的安全管理器。如果没 有设置安全管理器，RMI的类加载器就不会从远程下载任何的类。
>
> - **解释下Marshalling和demarshalling。**
>
> 当应用程序希望把内存对象跨网络传递到另一台主机或者是持久化到存储的时候，就必须要 把对象在内存里面的表示转化成合适的格式。这个过程就叫做Marshalling，反之就是demarshalling。

JSWDL 开发包的介绍

> **JAXP**:(Java API for XML Parsing)定义了在Java中使用DOM, SAX, XSLT的通用的接口。
>
> **JAXM**:(Java API for XML Messaging) 是为SOAP通信提供访问方法和传输机制的API。
>
> **SOAP**:简单对象访问协议(Simple Object Access Protocol)，它是用于交换XML编码信息的轻量级协议。
>
> **UDDI**:基于Web的、分布式的、为Web Service提供的、信息注册中心的实现标准规范
>
> **WSDL**:是一种 XML 格式，用于将网络服务描述为一组端点，这些端点对包含面向文档信息或面向过程信息的消息进行操作。

WEB容器功能

> Tomcat、webLogic
>
> 通信支持、管理Servlet生命周期，多线程、将jsp转换成java等等

深拷贝和浅拷贝

> 简单来讲就是复制、克隆；
>
> Person p=new Person(“张三”);
>
> 浅拷贝就是对对象中的数据成员进行简单赋值，如果存在动态成员或者指针就会报错
>
> 深拷贝就是对对象中存在的动态成员或指针重新开辟内存空间

值传递和引用传递

> 值传递是针对基本数据类型而言，传递的是值得副本，对副本的改变不会影响到原变量
>
> 引用传递就是将一个堆内存空间的使用权交给多个栈内存空间，每一个栈内存空间都可以对堆内存空间进行修改

Ajax

> 原理：HTTP协议的异步通信
>
> 实现：创建一个XMLHttpRequest对象，调用open方法，设置回调函数

HTTP

> 互联网通信协议HTTP协议，是一个无状态协议。这意味着，所有的状态都保存在服务器端。HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。它们分别对应四种基本操作：

HTTP1和HTTP2

> Http1是文本传送，Http2是二进制传送
>
> Http2支持多路复用，流ID是一个Http请求完成多个Http请求传输变成可能
>
> Http2支持在客户端未经请求许可的情况下主动向客户端推送内容

常见的编程协议

> **TCP**：传输控制协议，三次握手和四次分手
>
> **UDP**：用户数据报协议，适用于一次只传输少量数据，稳定性要求不高
>
> **SPX**：顺序包交换协议，保证数据传输的完整性，一般用于大型网络和局域网游戏环境（反恐精英和星际争霸）
>
> **NetBIOS**：网络输入输出系统

TCP

> **三次握手**：（ACK和SYN（ACK起应答作用，而SYN起同步作用）放在一 个报文里来发送。）
>
> 客户端发送一个带有SYN(SYN-send）标志的报文到服务器（请求连接）

服务端收到这个报文后发回一个带有SYN(SYN-recived)和ACK标志的报文给客户端（可以连接，做好连接准备了？）

客户端又将带有ACK标志的报文再次发送给服务器（ok）

四次分手：（FIN报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以ACK报文 和FIN报文多数情况下都是分开发送的。）

客户端发送一个Fin标志给服务器；

服务器收到返回一个带有ACK标志的报文，并确认序号为收到序号 +1；

服务器关闭客户端的连接，并返回Fin给客户端

客户端返回ACK确认，并将确认序号设置为收到序号+1；

标志

Closed:初始状态

Listen:服务器的 某个Socket处于监听状态，表示可以连接

SYN-received:服务器收到报文

SYN-sent:客户端发送报文

ESTABLISHED:建立连接

FIN_WAIT_1:准备主动关闭连接

FIN_WAIT_2:主动关闭连接（但是表示还有点数据要传给你，等下关闭）

TIME_WAIT: 表示收到了对方的FIN报文，并发送出了ACK报文，就等2MSL后即可回到CLOSED可用状态了。（因为假象网络并不可靠，你无法保证你最后发送的ACK报文会一定被对方收到，所以TIME_WAIT状态的作用就是用来重发可能丢失的 ACK报文。）

CLOSING:双方都正在关闭Socket

CLOSE_WAIT:等待关闭

LAST_ACK: 被动关闭一方在发送FIN报文后，最后等待对方的ACK报文。当收到ACK报文后，也即可以进入到CLOSED可用状态了。

两个应用程序同时执行主动打开的情况是可能的，虽然发生的可能性较低。但多数伯克利版的tcp/ip实现并不支持同时打开。

3TCP（传输控制协议）和UDP（用户数据报协议）

这两种传输方式都是实际的网络编程中进行使用，重要的数据一般使用TCP方式进行数据传输，而大量的非核心数据则都通过UDP方式进行传递.

TCP/IP

> TCP/IP 意味着 TCP 和 IP 在一起协同工作。
>
> TCP 负责应用软件（比如你的浏览器）和网络软件之间的通信。
>
> IP 负责计算机之间的通信。
>
> TCP 负责将数据分割并装入 IP 包，然后在它们到达的时候重新组合它们。
>
> IP 负责将包发送至接受者。

Socket

> Java中基于TCP协议实现网络通信的类

线程的处理流程

> 线程的处理流程大概都是读取数据、解码、计算处理、编码、发送响应

阻塞I/O通讯模型

> 阻塞I/O在调用InputStream.read()方法时是阻塞的，它会一直等到数据到来时（或超时）才会返回；同样，在调用ServerSocket.accept()方法时，也会一直阻塞到有客户端连接才会返回.
>
> **两点缺点**
>
> - 当客户端多时，会创建大量的处理线程。且每个线程都要占用栈空间和一些CPU时间
> - 阻塞可能带来频繁的上下文切换，且大部分上下文切换可能是无意义的。

NIO(new IO)

> 
>
> ==java NIO采用了双向通道（channel）进行数据传输==
>
> NIO的服务端会在selector中添加一个读事件。服务端的处理线程会轮询地访问selector，如果访问selector时发现有感兴趣的事件到达，则处理这些事件，如果没有感兴趣的事件到达，则处理线程会一直阻塞直到感兴趣的事件到达为止

NIO的工作原理

> - 由一个专门的线程来处理所有的 IO 事件，并负责分发。
> - 事件驱动机制：事件到的时候触发，而不是同步的去监视事件。
> - 线程通讯：线程之间通过 wait,notify 等方式通讯。保证每次上下文切换都是有意义的。减少无谓的线程切换。

java.nio.中提供了

> - **Selector**：通过调用Selector的select方法可以从所有的Channel中找到需要服务的实例（例如 Accept，read等）
> - **Channel**：代表一个可以被用于Poll操作的对象（可以是文件流也可以使网络流），Channel能够被注册到一个Selector中。
> - **Buffer**：提供读写数据的缓存（人为控制缓存的大小以及具体的操作）

IO 和 NIO 的区别

> IO是面向流的，NIO是面向缓冲区的。
>
> IO是阻塞的，NIO是非阻塞的。
>
> NIO有选择器机制，可以让一个线程来监视多个IO通道。

NIO的优点

> 不需要使用 read() 或者 write() 就可以处理文件内容。
>
> NIO的处理效率很快。

创建线程有几种不同的方式？你喜欢哪一种？为什么？

> 继承Thread类
>
> 实现Runnable接口
>
> 应用程序可以使用Executor框架来创建线程池

```
public Executor taskExecutor() {

  ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();

  executor.setCorePoolSize(10);

  executor.setMaxPoolSize(20);

  executor.setQueueCapacity(200);

  executor.setKeepAliveSeconds(60);

  executor.setThreadNamePrefix("taskExecutor-");

  executor.setRejectedExecutionHandler(new
ThreadPoolExecutor.CallerRunsPolicy());
```

```
    return executor;

}
```

线程池

> 线程池是一种多线程处理形式，处理过程中将任务添加到队列，然后在创建线程后自动启动这些任务。线程池线程都是后台线程。每个线程都使用默认的堆栈大小，以默认的优先级运行，并处于多线程单元中。如果某个线程在托管代码中空闲（如正在等待某个事件）,则线程池将插入另一个辅助线程来使所有处理器保持繁忙。如果所有线程池线程都始终保持繁忙，但队列中包含挂起的工作，则线程池将在一段时间后创建另一个辅助线程但线程的数目永远不会超过最大值。超过最大值的线程可以排队，但他们要等到其他线程完成后才启动。

竞态条件

> 多个线程竞争同一资源时，如果存在顺序敏感，就称存在竞态条件
>
> 

概括的解释下线程的几种可用状态

> 1. 新建 new
> 2. 就绪 放在可运行线程池中，等待被线程调度选中，获取cpu
> 3. 运行 获得了cpu
> 4. 阻塞
>   1. 等待阻塞 执行wait()
>   2. 同步阻塞 获取对象的同步琐时，同步锁被别的线程占用
>   3. 其他阻塞 执行了sleep()或join()方法)
> 5. 死亡

用什么关键字修饰同步方法?

> ==**Synchronized**==

stop()和suspend()方法为何不推荐使用?

> 反对使用stop()，是因为它不安全。它会解除由线程获取的所有锁定，而且如果对象处于一种不连贯状态，那么其他线程能在那种状态下检查和修改它们。结果很难检查出真正的问题所在。

> suspend()方法容易发生死锁。调用suspend()的时候，目标线程会停下来，但却仍然持有在这之前获得的锁定。此时，其他任何线程都不能访问锁定的资源，除非被"挂起"的线程恢复运行。

什么是ThreadLocal?

> ThreadLocal用于创建线程的本地变量，我们知道一个对象的所有线程会共享它的全局变量，所以这些变量不是线程安全的，我们可以使用同步技术。但是当我们不想使用同步的时候，我们可以选择ThreadLocal变量。
>
> 每个线程都会拥有他们自己的Thread变量，它们可以使用get()\set()方法去获取他们的默认值或者在线程内部改变他们的值。ThreadLocal实例通常是希望它们同线程状态关联起来是private static属性。

run()和start()区别

> **run( )**：只是调用普通run方法
>
> **start( )**：启动了线程,由Jvm调用run方法
>
> 启动一个线程是调用start()方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由JVM调度并执行。这并不意味着线程就会立即运行。run()方法可以产生必须退出的标志来停止一个线程。

sleep() 和 wait() 有什么区别?

> sleep是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用sleep不会释放对象锁。
>
> wait是Object类的方法，对此对象调用wait方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出notify方法（或notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态。 启动一个线程是用run()还是start()?

当一个线程进入一个对象的一个synchronized方法后，其它线程是否可进入此对象的其它方法?

> 不能，一个对象的一个synchronized方法只能由一个线程访问。

请说出你所知道的线程同步的方法

> **wait()**：使一个线程处于等待状态，并且释放所持有的对象的lock。 **sleep()**：使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉InterruptedException异常。
> **notify()**：唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且不是按优先级。 **notityAll()**：唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争。

线程调度和线程控制

> **线程调度（优先级）：**
>
> 与线程休眠类似，线程的优先级仍然无法保障线程的执行次序。只不过，优先级高的线程获取CPU资源的概率较大，优先级低的并非没机会执行。 线程的优先级用1-10之间的整数表示，数值越大优先级越高，默认的优先级为5。 在一个线程中开启另外一个新线程，则新开线程称为该线程的子线程，子线程初始优先级与父线程相同。
>
> **线程控制**
>
> - **sleep( )** //线程休眠 **join( )** //线程加入 **yield( )** //线程礼让 **setDaemon( )** //线程守护
>
> **中断线程**
>
> - **stop( ) interrupt( )** ==(首先选用)==

JMM

> java内存模型(JMM)是线程间通信的控制机制.JMM定义了主内存和线程之间抽象关系。线程之间的共享变量存储在主内存（main memory）中，每个线程都有一个私有的本地内存（local memory），本地内存中存储了该线程以读/写共享变量的副本。本地内存是JMM的一个抽象概念，并不真实存在。它涵盖了缓存，写缓冲区，寄存器以及其他的硬件和编译器优化。Java内存模型的抽象示意图如下：
>
> 
>
> 从上图来看，线程A与线程B之间如要通信的话，必须要经历下面2个步骤：
>
> 1. 首先，线程A把本地内存A中更新过的共享变量刷新到主内存中去。
> 2. 然后，线程B到主内存中去读取线程A之前已更新过的共享变量。

同步和异步有何异同，在什么情况下分别使用他们？举例说明

> 如果数据将在线程间共享。银行存取钱；那么这些数据就是共享数据，必须进行同步存取。 当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。

什么是线程饿死，什么是活锁?

> 当所有线程阻塞，或者由于需要的资源无效而不能处理，不存在非阻塞线程使资源可用。JavaAPI中线程活锁可能发生在以下情形：
>
> 1. 当所有线程在序中执行Object.wait(0)，参数为0的wait方法。程序将发生活锁直到在相应的对象上有线程调用Object.notify()或者Object.notifyAll()。
> 2. 当所有线程卡在无限循环中。

多线程中的忙循环是什么?

> 忙循环就是程序员用循环让一个线程等待，不像传统方法wait(), sleep() 或 yield() 它们都放弃了CPU控制，而忙循环不会放弃CPU，它就是在运行一个空循环。这么做的目的是为了保留CPU缓存。
>
> 在多核系统中，一个等待线程醒来的时候可能会在另一个内核运行，这样会重建缓存。为了避免重建缓存和减少等待重建的时间就可以使用它了。

简述synchronized和java.util.concurrent.locks.Lock的异同？

> **主要相同点**：Lock能完成synchronized所实现的所有功能 **主要不同点**：Lock有比synchronized更精确的线程语义和更好的性能。synchronized会自动释放锁，而Lock一定要求程序员手工释放，并且必须在finally从句中释放。

同步方法和同步代码块区别：

> 同步方法默认用this或者当前类class对象作为锁；
>
> 同步代码块可以选择以什么来加锁，比同步方法要更细颗粒度，我们可以选择只同步会发生 同步问题的部分代码而不是整个方法；

如何确保N个线程可以访问N个资源同时又不导致死锁?

> 使用多线程的时候，一种非常简单的避免死锁的方式就是：指定获取锁的顺序，并强制线程按照指定的顺序获取锁。因此，如果所有的线程都是以同样的顺序加锁和释放锁，就不会出 现死锁了。

什么是原子操作

> 所谓原子操作是指不会被线程调度机制打断的操作；这种操作一旦开始，就一直运行到结束，中间不会有任何 context switch （切 [1] 换到另一个线程）。

volatile 变量是什么？volatile 变量和 atomic 变量有什么不同

> volatile则是保证了所修饰的变量的可见。因为volatile只是在保证了同一个变量在多线程中的可见性，所以它更多是用于修饰作为开关状态的变量，即Boolean类型的变量。
>
> volatile多用于修饰类似开关类型的变量、Atomic多用于类似计数器相关的变量、其它多线程并发操作用synchronized关键字修饰。
>
> **volatile 有两个功用**
>
> 1. 这个变量不会在多个线程中存在复本，直接从内存读取。
> 2. 这个关键字会禁止指令重排序优化。也就是说，在 volatile 变量的赋值操作后面会有一个内

> 存屏障（生成的汇编代码上），读操作不会被重排序到内存屏障之前。

volatile 类型变量提供什么保证？能使得一个非原子操作变成原子操作吗

> volatile 提供 happens-before 的保证，确保一个线程的修改能对其他线程是可见的。
>
> 在Java 中除了 long 和 double 之外的所有基本类型的读和赋值，都是原子性操作。而64位的long和 double 变量由于会被JVM当作两个分离的32位来进行操作，所以不具有原子性，会产生字撕裂问题。但是当你定义long或double变量时，如果使用 volatile关键字，就会获到（简单的赋值与返回操作的）原子性

能创建 volatile 数组吗?

> 可以，但是创建的对象或数组的地址具有可见性，里面的数据是不可见的

一张表，里面有ID自增主键，当insert了17条记录之后，删除了第15,16,17条记录，再把Mysql重启，再insert一条记录，这条记录的ID是18还是15？

> 要根据表的类型区分
>
> - **MyISAM型**：因为此类型的表会把自增主键的最大ID记录在数据文件中，重启MySQL自增主键的最大ID也不会丢失，所以是==18==
> - **InnoDB型**：此类型表只把自增主键的最大ID存在内存中，所以重启数据库或者对表进行Optimize操作，都会导致最大ID丢失，所以是==15==

Heap表是什么?

> HEAP表存在于内存中，用于临时高速存储。
>
> BLOB或TEXT字段是不允许的
>
> 只能使用比较运算符=，<，>，=>，= <
>
> HEAP表不支持AUTO_INCREMENT
>
> 索引不可为NULL

Mysql服务器默认端口是什么?

> Mysql服务器的默认端口是3306。

与Oracle相比，Mysql有什么优势?

> Mysql是开源软件，随时可用，无需付费。

> Mysql是便携式的

区分CHAR_LENGTH和LENGTH?

> CHAR_LENGTH是字符数，而LENGTH是字节数。Latin字符的这两个数据是相同的，但是对于Unicode和其他编码，它们是不同的。

请简洁描述Mysql中InnoDB支持的四种事务隔离级别名称，以及逐级之间的区别?

> SQL标准定义的四个隔离级别为：
>
> **read uncommited**：读到未提交数据
>
> **read committed**：脏读，不可重复读
>
> **repeatable read**：可重读
>
> **serializable**：串行事物

在Mysql中ENUM的用法是什么?

> ENUM是一个字符串对象，用于指定一组预定义的值，并可在创建表时使用。
>
> SQL语法如下：

```
Create table size(name ENUM('Smail,'Medium','Large');
```

如何定义REGEXP?

> REGEXP是模式匹配，其中匹配模式在搜索值的任何位置。

CHAR和VARCHAR的区别?

> CHAR和VARCHAR类型在存储和检索方面有所不同
>
> CHAR列长度固定为创建表时声明的长度，长度值范围是1到255
>
> 当CHAR值被存储时，它们被用空格填充到特定长度，检索CHAR值时需删除尾随空格。

列的字符串类型可以是什么?

> 字符串类型是：
>
> SET

> BLOB
>
> ENUM
>
> CHAR
>
> TEXT
>
> VARCHAR

如何获取当前的Mysql版本?

> **SELECT VERSION( )** 用于获取当前Mysql的版本。

Mysql中使用什么存储引擎?

> 存储引擎称为表类型，数据使用各种技术存储在文件中。
>
> 技术涉及：
>
> Storage mechanism
>
> Locking levels
>
> Indexing
>
> Capabilities and functions.

TIMESTAMP在UPDATE CURRENT_TIMESTAMP数据类型上做什么?

> 创建表时TIMESTAMP列用Zero更新。只要表中的其他字段发生更改，UPDATE CURRENT_TIMESTAMP修饰符就将时间戳字段更新为当前时间。

主键和候选键有什么区别?

> 表格的每一行都由主键唯一标识,一个表只有一个主键。
>
> 主键也是候选键。按照惯例，候选键可以被指定为主键，并且可以用于任何外键引用。

如何使用Unix shell登录Mysql?

> 我们可以通过以下命令登录：
>
> ```
> # [mysql dir]/bin/mysql -h hostname -u <UserName> -p <password>
> ```

myisamchk是用来做什么的?

> 它用来压缩MyISAM表，这减少了磁盘或内存使用。

MYSQL数据库服务器性能分析的方法命令有哪些?

> Show status 一些值得监控的变量值： Bytes_received和Bytes_sent 和服务器之间来往的流量。 Com*服务器正在执行的命令。 Created在查询执行期限间创建的临时表和文件。 Handler_存储引擎操作。 Select*不同类型的联接执行计划。 Sort*几种排序信息。 Show session status like 'Select'; Show profiles SET profiling=1; Show profiles\G Show profile;

如何控制HEAP表的最大尺寸?

> Heal表的大小可通过称为max_heap_table_size的Mysql配置变量来控制。

MyISAM Static和MyISAM Dynamic有什么区别?

> 在MyISAM Static上的所有字段有固定宽度。动态MyISAM表将具有像TEXT，BLOB等字段，以适应不同长度的数据类型。
>
> MyISAM Static在受损情况下更容易恢复。

federated表是什么?

> federated表，允许访问位于其他服务器数据库上的表。

如果一个表有一列定义为TIMESTAMP，将发生什么?

> 每当行被更改时，时间戳字段将获取当前时间戳。

列设置为AUTO INCREMENT时，如果在表中达到最大值，会发生什么情况?

> 它会停止递增，任何进一步的插入都将产生错误，因为密钥已被使用。

怎样才能找出最后一次插入时分配了哪个自动增量?

> LAST_INSERT_ID将返回由Auto_increment分配的最后一个值，并且不需要指定表名称。

你怎么看到为表格定义的所有索引?

> 通过以下语句：

```
SHOW INDEX FROM <tablename>;
```

LIKE声明中的%和_是什么意思?

> %对应于0个或更多字符，_只是LIKE语句中的一个字符。

如何在Unix和Mysql时间戳之间进行转换?

> UNIX_TIMESTAMP是从Mysql时间戳转换为Unix时间戳的命令
>
> FROM_UNIXTIME是从Unix时间戳转换为Mysql时间戳的命令

列对比运算符是什么?

> 在SELECT语句的列比较中使用=，<>，<=，<，> =，>，<<，>>，<=>，AND，OR或LIKE运算符。

我们如何得到受查询影响的行数?

> 通过以下语句

```
SELECT COUNT(user_id)FROM <tablename>;
```

Mysql查询是否区分大小写?

> 不区分

LIKE和REGEXP操作有什么区别?

> LIKE和REGEXP运算符用于表示^和%。

```
SELECT * FROM <tablename> WHERE * REGEXP "^b";
SELECT * FROM <tablename> WHERE * LIKE "%b";
```

BLOB和TEXT有什么区别?

> **BLOB**

> BLOB是一个二进制对象，可以容纳可变数量的数据。有四种类型的BLOB
>
> - TINYBLOB
> - BLOB
> - MEDIUMBLOB和
> - LONGBLOB
>
> 它们只能在所能容纳价值的最大长度上有所不同。
>
> **TEXT**
>
> TEXT是一个不区分大小写的BLOB。四种TEXT类型
>
> - TINYTEXT
> - TEXT
> - MEDIUMTEXT和
> - LONGTEXT
>
> 它们对应于四种BLOB类型，并具有相同的最大长度和存储要求。
>
> ==BLOB和TEXT类型之间的唯一区别在于对BLOB值进行排序和比较时区分大小写，对TEXT值不区分大小写。==

mysql_fetch_array和mysql_fetch_object的区别是什么?

> 以下是mysql_fetch_array和mysql_fetch_object的区别：
>
> mysql_fetch_array（） - 将结果行作为关联数组或来自数据库的常规数组返回。
>
> mysql_fetch_object - 从数据库返回结果行作为对象。

数据库的三范式?

> 第一范式：数据库表的每一个字段都是不可分割的
>
> 第二范式：数据库表中的非主属性只依赖于主键
>
> 第三范式：不存在非主属性对关键字的传递函数依赖关系

我们如何在mysql中运行批处理模式?

> 以下命令用于在批处理模式下运行：
>
> mysql;
>
> mysql mysql.out

MyISAM表格将在哪里存储，并且还提供其存储格式?

> 每个MyISAM表格以三种格式存储在磁盘上：
>
> ==“.frm”==文件存储表定义
>
> 数据文件具有==“.MYD”==（MYData）扩展名
>
> 索引文件具有==“.MYI”==（MYIndex）扩展名

Mysql中有哪些不同的表格?

> 共有5种类型的表格：
>
> MyISAM
>
> Heap
>
> Merge
>
> INNODB
>
> ISAM
>
> MyISAM是Mysql的默认存储引擎。

ISAM是什么?

> ISAM简称为索引顺序访问方法。它是由IBM开发的，用于在磁带等辅助存储系统上存储和检索数据。

InnoDB是什么?

> InnoDB是一个由Oracle公司开发的Innobase Oy事务安全存储引擎。

Mysql如何优化DISTINCT?

> DISTINCT在所有列上转换为GROUP BY，并与ORDER BY子句结合使用。
>
> ```
> SELECT DISTINCT t1.a FROM t1,t2 where t1.a=t2.a;
> ```

如何显示前50行?

> 在Mysql中，使用以下代码查询显示前50行：
>
> ```
> SELECT*FROM
> LIMIT 0,50;
> ```

可以使用多少列创建索引?

> 任何标准表最多可以创建16个索引列。

NOW（）和CURRENT_DATE（）有什么区别?

> **NOW**（）命令用于显示当前年份，月份，日期，小时，分钟和秒。
>
> **CURRENT_DATE**（）仅显示当前年份，月份和日期。

什么样的对象可以使用CREATE语句创建?

> 以下对象是使用CREATE语句创建的：
>
> DATABASE
>
> EVENT
>
> FUNCTION
>
> INDEX
>
> PROCEDURE
>
> TABLE
>
> TRIGGER
>
> USER
>
> VIEW

Mysql表中允许有多少个TRIGGERS?

> 在Mysql表中允许有六个触发器，如下：
>
> BEFORE INSERT
>
> AFTER INSERT
>
> BEFORE UPDATE
>
> AFTER UPDATE
>
> BEFORE DELETE and
>
> AFTER DELETE

什么是非标准字符串类型?

> 以下是非标准字符串类型：
>
> TINYTEXT
>
> TEXT
>
> MEDIUMTEXT
>
> LONGTEXT

什么是通用SQL函数?

> **数学函数**
>
> - Abs（num）求绝对值
> - floor（num）向下取整
> - ceil（num）向上取整
>
> ---
>
> **字符串函数**
>
> - insert (s1,index,length,s2) 替换函数
>   - S1表示被替换的字符串
>   - s2表示将要替换的字符串
>   - Index表示被替换的位置,从1开始
>   - Lebgth表示被替换的长度
> - upper（str），ucase（str）将字母改为大写
> - lower（str），lcase（str）将字母改为小写
> - left（str，length）返回str字符串的前length个字符
> - right（str，length）返回str字符串的后length个字符
> - substring（str，index，length）返回str字符串从index位开始长度为length个字符（index从1开始）
> - reverse（str）将str字符串倒序输出
>
> ---
>
> **日期函数**
>
> - curdate（）、current_date( )获取当前日期
> - curtime（）、current_time( )获取当前日期
> - now（）获取当前日期和时间
> - datediff（d1、d2）d1和d2之间的天数差
> - adddate（date，num）返回date日期开始，之后num天的日期
> - subdate（date，num）返回date日期开始，之前num天的日期
>
> ---
>
> **聚合函数**
>
> - Count（字段）根据某个字段统计总记录数（当前数据库保存到多少条数据）
> - sum（字段）计算某个字段的数值总和

- avg（字段）计算某个字段的数值的平均值
- Max（字段）、min（字段）求某个字段最大或最小值

解释访问控制列表

ACL（访问控制列表）是与对象关联的权限列表。这个列表是Mysql服务器安全模型的基础，它有助于排除用户无法连接的问题。

Mysql将ACL（也称为授权表）缓存在内存中。当用户尝试认证或运行命令时，Mysql会按照预定的顺序检查ACL的认证信息和权限。

MYSQL支持事务吗?

在缺省模式下，MYSQL是autocommit模式的，所有的数据库更新操作都会即时提交，所以在缺省情况下，mysql是不支持事务的。

但是如果你的MYSQL表类型是使用InnoDB Tables 或 BDB tables的话，你的MYSQL就可以使用事务处理,使用SET AUTOCOMMIT=0就可以使MYSQL允许在非autocommit模式，在非autocommit模式下，你必须使用COMMIT来提交你的更改，或者用ROLLBACK来回滚你的更改。

示例如下：

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summmary=@A WHERE type=1;
COMMIT;
```

mysql里记录货币用什么字段类型好

NUMERIC和DECIMAL类型被Mysql实现为同样的类型，这在SQL92标准允许。他们被用于保存值，该值的准确精度是极其重要的值，例如与金钱有关的数据。

DECIMAL和NUMERIC值作为字符串存储，而不是作为二进制浮点数，以便保存那些值的小数精度。

当一个DECIMAL或NUMERIC列被赋给了其大小超过指定(或缺省的）precision和scale隐含的范围的值，Mysql存储表示那个范围的相应的端点值。

MYSQL数据表在什么情况下容易损坏?

服务器突然断电导致数据文件损坏。

强制关机，没有先关闭mysql 服务等。

mysql有关权限的表都有哪几个?

> Mysql服务器通过权限表来控制用户对数据库的访问，权限表存放在mysql数据库里，由mysql_install_db脚本初始化。这些权限表分别user，db，table_priv，columns_priv和host。

Mysql中有哪几种锁?

> MyISAM支持表锁，InnoDB支持表锁和行锁，默认为行锁
>
> 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，并发量最低
>
> 行级锁：开销大，加锁慢，会出现死锁。锁力度小，发生锁冲突的概率小，并发度最高

Mysql数据优化

> 1. **优化数据类型**
>   1. 避免使用NULL，NULL需要特殊处理,大多数时候应该使用NOT NULL，或者使用一个特殊的值，如0，-1作为默认值。
>   2. 仅可能使用更小的字段，MySQL从磁盘读取数据后是存储到内存中的，然后使用cpu周期和磁盘I/O读取它，这意味着越小的数据类型占用的空间越小.
> 2. **小心字符集转换**
>
> 客户端或应用程序使用的字符集可能和表本身的字符集不一样，这需要MySQL在运行过程中隐含地进行转换，此外，要确定字符集如UTF-8是否支持多字节字符，因此它们需要更多的存储空间。
>
> 3. **优化count(my_col)和count(*)**
> 4. **优化子查询**
>
> 遇到子查询时，MySQL查询优化引擎并不是总是最有效的，这就是为什么经常将子查询转换为连接查询的原因了，优化器已经能够正确处理连接查询了，当然要注意的一点是，确保连接表(第二个表)的连接列是有索引的，在第一个表上MySQL通常会相对于第二个表的查询子集进行一次全表扫描，这是嵌套循环算法的一部分。
>
> 5. **优化UNION**
>
> 在跨多个不同的数据库时使用UNION是一个有趣的优化方法，UNION从两个互不关联的表中返回数据，这就意味着不会出现重复的行，同时也必须对数据进行排序，我们知道排序是非常耗费资源的，特别是对大表的排序。
>
> UNION ALL可以大大加快速度，如果你已经知道你的数据不会包括重复行，或者你不在乎是否会出现重复的行，在这两种情况下使用UNION ALL更适合。此外，还可以在应用程序逻辑中采用某些方法避免出现重复的行，这样UNION ALL和UNION返回的结果都是一样的，但UNION ALL不会进行排序。

MySQL的关键字

> 添加索引：

```
alter table tableName add 索引（索引字段）
```

主键：primary key

唯一：unique

全局：fulltext

普通：index

多列： index index_name

页级:引擎 BDB。次锁定相邻的一组记录。

表级:引擎 MyISAM ， 理解为锁住整个表，可以同时读，写不行。 行级:引擎 INNODB ， 单独的一行记录加锁，对指定的记录进行加锁，这样其它进程还是可以对同一个表中的其它记录进行操作。 表级锁速度快，但冲突多，行级冲突少，但速度慢。

存储引擎

存储引擎说白了就是如何存储数据、如何为存储的数据建立索引和如何更新、查询数据等技术的实现方法。

1. MyISAM：这种引擎是mysql最早提供的。这种引擎又可以分为静态MyISAM、动态MyISAM和压缩MyISAM三种：
    - 静态MyISAM：如果数据表中的各数据列的长度都是预先固定好的，服务器将自动选择这种表类型。因为数据表中每一条记录所占用的空间都是一样的，所以这种表存取和更新的效率非常高。当数据受损时，恢复工作也比较容易做。
    - 动态MyISAM：如果数据表中出现varchar、text或BLOB字段时，服务器将自动选择这种表类型。相对于静态MyISAM，这种表存储空间比较小，但由于每条记录的长度不一，所以多次修改数据后，数据表中的数据就可能离散的存储在内存中，进而导致执行效率下降。同时，内存中也可能会出现很多碎片。因此，这种类型的表要经常用 optimize table 命令或优化工具来进行碎片整理。
    - 压缩MyISAM：以上说到的两种类型的表都可以用myisamchk工具压缩。这种类型的表进一步减小了占用的存储，但是这种表压缩之后不能再被修改。另外，因为是压缩数据，所以这种表在读取的时候要先时行解压缩。

==但是，不管是何种MyISAM表，目前它都不支持事务，行级锁和外键约束的功能。==

2. MyISAM Merge引擎：这种类型是MyISAM类型的一种变种。合并表是将几个相同的MyISAM表合并为一个虚表。常应用于日志和数据仓库。
3. InnoDB：InnoDB表类型可以看作是对MyISAM的进一步更新产品，它提供了事务、行级锁机制和外键约束的功能。
4. memory(heap)：这种类型的数据表只存在于内存中。它使用散列索引，所以数据的存取速度非常快。因为是存在于内存中，所以这种类型常应用于临时表中。
5. **archive**：这种类型只支持select 和 insert语句，而且不支持索引。

Desc[ribe] tablename; //查看数据表的结构

show engines; 命令可以显示当前数据库支持的存储引擎情况

数据库备份

> 必须要在未登录状态下
>
> - 导出整个数据库

```
mysqldump -u 用户名 -p 数据库名 > 导出的文件名
```

> - 导出一个表

```
mysqldump -u 用户名 -p 数据库名 表名> 导出的文件名
```

> - 导出一个数据库结构

```
mysqldump -u dbuser -p -d --add-drop-table dbname >d:/dbname_db.sql
```

> -d 没有数据 --add-drop-table 在每个create语句之前增加一个drop table

如何显示创表语句以及给表中添加字段

> 显示创表语句

```
show create table <tableName>;
```

> 向表中添加字段

```
alter table t_user add column updateTime TIMESTAMP;
```

SQL语言分类

> SQL语言共分为四大类：数据查询语言DQL，数据操纵语言DML，数据定义语言DDL，数据控制语言DCL。

truncate delete drop的区别：

> **drop(DDL语句)**：是不可逆操作，会将表所占用空间全部释放掉；
>
> **truncate(DDL语句)**：只针对于删除表的操作，在删除过程中不会激活与表有关的删除触发器并且不会把删除记录放在日志中；当表被truncate后，这个表和索引会恢复到初始大小；
>
> **delete(DML语句)**：可以删除表也可以删除行，但是删除记录会被计入日志保存，而且表空间大小不会恢复到原来；

> ==执行速度：drop>truncate>delete==

说出ArrayList,Vector, LinkedList的存储性能和特性

> ArrayList和Vector都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，
>
> Vector由于使用了synchronized方法（线程安全），
>
> 通常性能上较ArrayList差，而LinkedList使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快
>
> ==ArrayList在查找时速度快，LinkedList在插入与删除时更具优势==

HashMap和Hashtable的区别

> HashMap是Hashtable的轻量级实现（非线程安全的实现），他们都完成了Map接口，主要区别在于HashMap允许空（null）键值（key）,由于非线程安全，效率上可能高于Hashtable。
>
> Hashtable继承自Dictionary类，而HashMap是Java1.2引进的Map interface的一个实现。 最大的不同是，Hashtable的方法是Synchronize的，而HashMap不是，在多个线程访问Hashtable时，不需要自己为它的方法实现同步，而HashMap 就必须为之提供外同步，因此，HashMap更适合于单线程环境，而Hashtable 适合于多线程环境。
>
> HashMap提供了可供应用迭代的键的集合，因此，HashMap是快速失败的。另一方面，Hashtable提供了对键的列举(Enumeration)。

快速失败(fail-fast)和安全失败(fail-safe)的区别是什么?

> Iterator的安全失败是基于对底层集合做拷贝，因此，它不受源集合上修改的影响。java.util 包下面的所有的集合类都是快速失败的，而java.util.concurrent包下面的所有的类都是安全失败的。快速失败的迭代器会抛出ConcurrentModificationException异常，而安全失败的迭代器永远不会抛出这样的异常。

hashmap的数据结构

> 在java编程语言中，最基本的结构就是两种，一个是数组，另外一个是模拟指针（引用），所有的数据结构都可以用这两个基本结构来构造的，hashmap也不例外。Hashmap实际上是一个数组和链表的结合体（在数据结构中，一般称之为“链表散列“）
>
> 

HashMap的工作原理是什么?

> Java中的HashMap是以键值对(key-value)的形式存储元素的。HashMap需要一个hash 函数，它使用hashCode()和equals()方法来向集合/从集合添加和检索元素。当调用put() 方法的时候，HashMap会计算key的hash值，然后把键值对存储在集合中合适的索引上。 如果key已经存在了，value会被更新成新值。HashMap的一些重要的特性是它的容量 (capacity)，负载因子(load factor)和扩容极限(threshold resizing)。

Hashmap什么时候进行扩容呢?

> 当hashmap中的元素个数超过数组大小*loadFactor时，就会进行数组扩容，loadFactor的默认值为0.75，也就是说，默认情况下，数组大小为16，那么当hashmap中元素个数超过16*0.75=12的时候，就把数组的大小扩展为2*16=32，即扩大一倍，然后重新计算每个元素在数组中的位置，而这是一个非常消耗性能的操作，所以如果我们已经预知hashmap中元素的个数，那么预设元素的个数能够有效的提高hashmap的性能。比如说，我们有1000个元素new HashMap(1000), 但是理论上来讲new HashMap(1024)更合适，不过上面annegu已经说过，即使是1000，hashmap也自动会将其设置为1024。 但是new HashMap(1024)还不是更合适的，因为0.75*1000 < 1000, 也就是说为了让0.75 * size > 1000, 我们必须这样new HashMap(2048)才最合适，既考虑了&的问题，也避免了resize的问题。

CorrentHashMap的工作原理?

> **参考答案:**在jdk 8中，ConcurrentHashMap不再使用Segment分离锁，而是采用一种乐观锁CAS算法来实现同步问题，但其底层还是“数组+链表->红黑树”的实现。

Java集合类框架的基本接口有哪些?

> 集合类接口指定了一组叫做元素的对象。集合类接口的每一种具体的实现类都可以选择以它 自己的方式对元素进行保存和排序。有的集合类允许重复的键，有些不允许。
>
> Java集合类提供了一套设计良好的支持对一组对象进行操作的接口和类。Java集合类里面 最基本的接口有：
>
> **Collection**：代表一组对象，每一个对象都是它的子元素。
>
> **Set**：不包含重复元素的Collection。
>
> **List**：有顺序的collection，并且可以包含重复元素。
>
> **Map**：可以把键(key)映射到值(value)的对象，键不能重复。

HashSet和TreeSet有什么区别?

> HashSet是由一个hash表来实现的，因此，它的元素是无序的。add()，remove()，contains()
>
> TreeSet是由一个树形的结构来实现的，它里面的元素是有序的。因此，add()， remove()，contains()方法的时间复杂度是O(logn)。

HashSet的底层实现是什么?

> 通过看源码知道HashSet的实现是依赖于HashMap的，HashSet的值都是存储在HashMap中的。在HashSet的构造法中会初始化一个HashMap对象，HashSet不允许值重复，因此，HashSet的值是作为HashMap的key存储在HashMap中的，当存储的值已经存在时返回false。

LinkedHashMap的实现原理?

> LinkedHashMap也是基于HashMap实现的，不同的是它定义了一个Entry header，这个header不是放在Table里，它是额外独立出来的。LinkedHashMap通过继承hashMap中的Entry,并添加两个属性Entry before,after,和header结合起来组成一个双向链表，来实现按插入顺序或访问顺序排序。LinkedHashMap定义了排序模式accessOrder，该属性为boolean型变量，对于访问顺序，为true；对于插入顺序，则为false。一般情况下，不必指定排序模式，其迭代顺序即为默认为插入顺序。

为什么集合类没有实现Cloneable和Serializable接口?

> 克隆(cloning)或者是序列化(serialization)的语义和含义是跟具体的实现相关的。因此，应该 由集合类的具体实现来决定如何被克隆或者是序列化。

什么是迭代器(Iterator)?

> Iterator接口提供了很多对集合元素进行迭代的方法。每一个集合类都包含了可以返回迭代 器实例的迭代方法。迭代器可以在迭代的过程中删除底层集合的元素,但是不可以直接调用集合的remove(Object Obj)删除，可以通过迭代器的remove()方法删除。

Iterator和ListIterator的区别是什么?

> 下面列出了他们的区别：
>
> Iterator可用来遍历Set和List集合，但是ListIterator只能用来遍历List。
>
> Iterator对集合只能是前向遍历，ListIterator既可以前向也可以后向。
>
> ListIterator实现了Iterator接口，并包含其他的功能，比如：增加元素，替换元素，获取前一个和后一个元素的索引，等等。

数组(Array)和列表(ArrayList)有什么区别？什么时候应该使用Array而不是ArrayList?

> Array可以包含基本类型和对象类型，ArrayList只能包含对象类型。
>
> Array大小是固定的，ArrayList的大小是动态变化的。

> ArrayList处理固定大小的基本数据类型的时候，这种方式相对比较慢。

Java集合类框架的最佳实践有哪些?

> - 假如元素的大小是固 定的，而且能事先知道，我们就应该用Array而不是ArrayList。
> - 有些集合类允许指定初始容量。因此，如果我们能估计出存储的元素的数目，我们可以设置初始容量来避免重新计算hash值或者是扩容。
> - 为了类型安全，可读性和健壮性的原因总是要使用泛型。同时，使用泛型还可以避免运行时的ClassCastException。
> - 使用JDK提供的不变类(immutable class)作为Map的键可以避免为我们自己的类实现hashCode()和equals()方法。
> - 编程的时候接口优于实现。
> - 底层的集合实际上是空的情况下，返回长度是0的集合或者是数组，不要返回null。

Set里的元素是不能重复的，那么用什么方法来区分重复与否呢? 是用==还是equals()? 它们有何区别

> Set里的元素是不能重复的，那么用iterator()方法来区分重复与否。equals()是判读两个Set是否相等
>
> equals()和==方法决定引用值是否指向同一对象equals()在类中被覆盖，为的是当两个分离的对象的内容和类型相配的话，返回真值

Comparable和Comparator接口是干什么的？列出它们的区别。

> Java提供了只包含一个compareTo()方法的Comparable接口。这个方法可以个给两个对象排序。具体来说，它返回负数，0，正数来表明输入对象小于，等于，大于已经存在的对象。
>
> Java提供了包含compare()和equals()两个方法的Comparator接口。compare()方法用来给两个输入参数排序，返回负数，0，正数表明第一个参数是小于，等于，大于第二个参数。equals()方法需要一个对象作为参数，它用来决定输入参数是否和comparator相等。只有当输入参数也是一个comparator并且输入参数和当前comparator的排序结果是相同的时 候，这个方法才返回true。

Collection和Collections的区别

> collection是集合类的上级接口,继承与它的接口主要是set和list
>
> collections类是针对集合类的一个帮助类.它提供一系列的静态方法对各种集合的搜索,排序,线程安全化等操作.

java内存分配

> **寄存器：**我们无法控制

> **静态域：**static定义的静态成员
>
> **常量池：**编译时被确定并保存在.class文件中的（final）常量值和一些文本修饰的符号引用（类和接口的全限定名，字段的名称和描述符，方法和名称和描述符）
>
> **非ram存储：**硬盘等永久存储空间
>
> **堆内存：**new创建的对象和数组，由java虚拟机自动垃圾回收器管理,存取速度慢
>
> **栈内存：**基本类型的变量和对象的引用变量（堆内存空间的访问地址），速度快，可以共享，但是大小与生存期必须确定，缺乏灵活性

Java堆的结构是什么样子的？什么是堆中的永久代(Perm Gen space)?

> JVM的堆是运行时数据区，所有类的实例和数组都是在堆上分配内存。它在JVM启动的时 候被创建。对象所占的堆内存是由自动内存管理系统也就是垃圾收集器回收。
>
> 堆内存是由存活和死亡的对象组成的。存活的对象是应用可以访问的，不会被垃圾回收。死亡的对象是应用不可访问尚且还没有被垃圾收集器回收掉的对象。一直到垃圾收集器把这些 对象回收掉之前，他们会一直占据堆内存空间。

GC是什么? 为什么要有GC

> GC是垃圾收集的意思（Gabage Collection）,忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃

简述java垃圾回收机制?

> 在java中，程序员是不需要显示的去释放一个对象的内存的，而是由虚拟机自行执行。在JVM中，有一个垃圾回收线程，它是低优先级的，在正常情况下是不会执行的，只有在虚拟机空闲或者当前堆内存不足时，才会触发执行，扫面那些没有被任何引用的对象，并将它们添加到要回收的集合中，进行回收。

如何判断一个对象是否存活?(或者GC对象的判定方法)

> 判断一个对象是否存活有两种方法:
>
> 1. 引用计数法
>
> 所谓引用计数法就是给每一个对象设置一个引用计数器，每当有一个地方引用这个对象时，就将计数器加一，引用失效时，计数器就减一。当一个对象的引用计数器为零时，说明此对象没有被引用，也就是“死对象”,将会被垃圾回收.
>
> 引用计数法有一个缺陷就是无法解决循环引用问题，也就是说当对象A引用对象B，对象B又引用者对象A，那么此时A,B对象的引用计数器都不为零，也就造成无法完成垃圾回收，所以主流的虚拟机都没有采用这种算法。

2. 可达性算法(引用链法)

该算法的思想是：从一个被称为GC Roots的对象开始向下搜索，如果一个对象到GC Roots没有任何引用链相连时，则说明此对象不可用。

在java中可以作为GC Roots的对象有以下几种:

- 虚拟机栈中引用的对象
- 方法区类静态属性引用的对象
- 方法区常量池引用的对象
- 本地方法栈JNI引用的对象

虽然这些算法可以判定一个对象是否能被回收，但是当满足上述条件时，一个对象比不一定会被回收。当一个对象不可达GC Root时，这个对象并不会立马被回收，而是出于一个死缓的阶段，若要被真正的回收需要经历两次标记.

如果对象在可达性分析中没有与GC Root的引用链，那么此时就会被第一次标记并且进行一次筛选，筛选的条件是是否有必要执行finalize()方法。当对象没有覆盖finalize()方法或者已被虚拟机调用过，那么就认为是没必要的。 如果该对象有必要执行finalize()方法，那么这个对象将会放在一个称为F-Queue的对队列中，虚拟机会触发一个Finalize()线程去执行，此线程是低优先级的，并且虚拟机不会承诺一直等待它运行完，这是因为如果finalize()执行缓慢或者发生了死锁，那么就会造成F-Queue队列一直等待，造成了内存回收系统的崩溃。GC对处于F-Queue中的对象进行第二次被标记，这时，该对象将被移除”即将回收”集合，等待回收。

垃圾回收的2种回收机制

回收机制有分代复制垃圾回收和标记垃圾回收

垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收?

对于GC来说，当程序员创建对象时，GC就开始监控这个对象的地址、大小以及使用情况。通常，GC采用有向图的方式记录和管理堆(heap)中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的"。当GC确定一些对象为"不可达"时，GC就有责任回收这些内存空间。可以。程序员可以手动执行System.gc()，通知GC运行，但是Java语言规范并不保证GC一定会执行。

System.gc()和Runtime.gc()会做什么事情?

这两个方法用来提示JVM要进行垃圾回收。但是，立即开始还是延迟进行垃圾回收是取决于JVM的。

finalize()方法什么时候被调用？析构函数(finalization)的目的是什么？

> 垃圾回收器(garbage colector)决定回收某对象时，就会运行该对象的finalize()方法 但是在Java中很不幸，如果内存总是充足的，那么垃圾回收可能永远不会进行，也就是说filalize() 可能永远不被执行，显然指望它做收尾工作是靠不住的。 那么finalize()究竟是做什么的呢？ 它最主要的用途是回收特殊渠道申请的内存。Java程序有垃圾回收器，所以一般情况下内存问题不用程序员操心。但有一种JNI(Java Native Interface)调用non-Java程序（C或C++）， finalize()的工作就是回收这部分的内存。

如果对象的引用被置为null，垃圾收集器是否会立即释放对象占用的内存?

> 不会，在下一个垃圾回收周期中，这个对象将是可被回收的。

什么是分布式垃圾回收(DGC)？它是如何工作的?

> DGC叫做分布式垃圾回收。RMI使用DGC来做自动垃圾回收。因为RMI包含了跨虚拟机的远程对象的引用，垃圾回收是很困难的。DGC使用引用计数算法来给远程对象提供自动 内存管理。

串行(serial)收集器和吞吐量(throughput)收集器的区别是什么?

> 吞吐量收集器使用并行版本的新生代垃圾收集器，它用于中等规模和大规模数据的应用程序。 而串行收集器对大多数的小应用(在现代处理器上需要大概100M左右的内存)就足够了。

在Java中，对象什么时候可以被垃圾回收?

> 当对象对当前使用这个对象的应用程序变得不可触及的时候，这个对象就可以被回收了。

分代

分代是Java垃圾收集的一大亮点，根据对象的生命周期长短，把堆分为3个代：Young，Old和Permanent，

> - **Young（年轻代）**
>
> 年轻代分三个区。一个Eden区，两个Survivor区。大部分对象在Eden区中生成。当Eden区满时，还存活的对象将被复制到Survivor区 （两个中的一个），当这个Survivor区满时，此区的存活对象将被复制到另外一个Survivor区，当这个Survivor去也满了的时候，从第一 个Survivor区复制过来的并且此时还存活的对象，将被复制“年老区(Tenured)”
>
> - **Tenured（年老代）**
>
> 年老代存放从年轻代存活的对象。一般来说年老代存放的都是生命期较长的对象。
>
> - **Perm（持久代）**

> 用于存放静态文件，如今Java类、方法等。持久代对垃圾回收没有显著影响，但是有些应用可能动态生成或者调用一些class，例如Hibernate等， 在这种时候需要设置一个比较大的持久代空间来存放这些运行过程中新增的类。持久代大小通过-XX:MaxPermSize=进行设置。

Gc的基本概念

> gc分为full gc 跟 minor gc，当每一块区满的时候都会引发gc。
>
> **Scavenge GC** 一般情况下，当新对象生成，并且在Eden申请空间失败时，就触发了Scavenge GC，堆Eden区域进行GC，清除非存活对象，并且把尚且存活的对象移动到Survivor区。然后整理Survivor的两个区。
>
> **Full GC** 对整个堆进行整理，包括Young、Tenured和Perm。Full GC比Scavenge GC要慢，因此应该尽可能减少Full GC。有如下原因可能导致Full GC：
>
> - Tenured被写满
> - Perm域被写满
> - System.gc()被显示调用
> - 上一次GC之后Heap的各域分配策略动态变化

简述java内存分配与回收策率以及Minor GC和Major GC

> - 对象优先在堆的Eden区分配。
> - 大对象直接进入老年代.
> - 长期存活的对象将直接进入老年代.
>
> 当Eden区没有足够的空间进行分配时，虚拟机会执行一次Minor GC.Minor Gc通常发生在新生代的Eden区，在这个区的对象生存期短，往往发生Gc的频率较高，回收速度比较快;Full Gc/Major GC 发生在老年代，一般情况下，触发老年代GC的时候不会触发Minor GC,但是通过配置，可以在Full GC之前进行一次Minor GC这样可以加快老年代的回收速度。

JVM的永久代中会发生垃圾回收么?

> 垃圾回收不会发生在永久代，如果永久代满了或者是超过了临界值，会触发完全垃圾回收(Full GC)。
>
> (注：Java8中已经移除了永久代，新加了一个叫做元数据区的native内存区)

java中垃圾收集的方法有哪些?

> 标记-清除:这是垃圾收集算法中最基础的，根据名字就可以知道，它的思想就是标记哪些要被回收的对象，然后统一回收。这种方法很简单，但是会有两个主要问题：1.效率不高，标记和清除的效率都很低；2.会产生大量不连续的内存碎片，导致以后程序在分配较大的对象时，由于没有充足的连续内存而提前触发一次GC动作。

> 复制算法:为了解决效率问题，复制算法将可用内存按容量划分为相等的两部分，然后每次只使用其中的一块，当一块内存用完时，就将还存活的对象复制到第二块内存上，然后一次性清楚完第一块内存，再将第二块上的对象复制到第一块。但是这种方式，内存的代价太高，每次基本上都要浪费一般的内存。 于是将该算法进行了改进，内存区域不再是按照1：1去划分，而是将内存划分为8:1:1三部分，较大那份内存交Eden区，其余是两块较小的内存区叫Survior区。每次都会优先使用Eden区，若Eden区满，就将对象复制到第二块内存区上，然后清除Eden区，如果此时存活的对象太多，以至于Survivor不够时，会将这些对象通过分配担保机制复制到老年代中。(java堆又分为新生代和老年代)
>
> 标记-整理:该算法主要是为了解决标记-清除，产生大量内存碎片的问题；当对象存活率较高时，也解决了复制算法的效率问题。它的不同之处就是在清除对象的时候现将可回收对象移动到一端，然后清除掉端边界以外的对象，这样就不会产生内存碎片了。
>
> 分代收集 :现在的虚拟机垃圾收集大多采用这种方式，它根据对象的生存周期，将堆分为新生代和老年代。在新生代中，由于对象生存期短，每次回收都会有大量对象死去，那么这时就采用复制算法。老年代里的对象存活率较高，没有额外的空间进行分配担保；

java中会存在内存泄漏吗，请简单描述。

> 会理论上Java因为有垃圾回收机制（GC）不会存在内存泄露问题（这也是Java被广泛使用于服务器端编程的一个重要原因）；然而在实际开发中，可能会存在无用但可达的对象，这些对象不能被GC回收，因此也会导致内存泄露的发生。例如。自己实现堆载的数据结构时有可能会出现内存泄露 或者hibernate的Session（一级缓存）中的对象属于持久态，垃圾回收器是不会回收这些对象的，然而这些对象中可能存在无用的垃圾对象，如果不及时关闭（close）或清空（flush）一级缓存就可能导致内存泄露。

java类加载过程?

> java类加载需要经历一下7个过程：
>
> 1. **加载**
>
> 加载是类加载的第一个过程，在这个阶段，将完成一下三件事情：
>
> - 通过一个类的全限定名获取该类的二进制流。
> - 将该二进制流中的静态存储结构转化为方法去运行时数据结构。
> - 在内存中生成该类的Class对象，作为该类的数据访问入口。
>
> 2. **验证**
>
> 验证的目的是为了确保Class文件的字节流中的信息不回危害到虚拟机.在该阶段主要完成以下四钟验证:
>
> - 文件格式验证：验证字节流是否符合Class文件的规范，如主次版本号是否在当前虚拟机范围内，常量池中的常量是否有不被支持的类型.
> - 元数据验证:对字节码描述的信息进行语义分析，如这个类是否有父类，是否集成了不被继承的类等。
> - 字节码验证：是整个验证过程中最复杂的一个阶段，通过验证数据流和控制流的分析，确定程序语义是否正确，主要针对方法体的验证。如：方法中的类型转换是否正确，跳转指令是

否正确等。
- 符号引用验证：这个动作在后面的解析过程中发生，主要是为了确保解析动作能正确执行。

3. **准备**

准备阶段是为类的静态变量分配内存并将其初始化为默认值，这些内存都将在方法区中进行分配。准备阶段不分配类中的实例变量的内存，实例变量将会在对象实例化时随着对象一起分配在Java堆中。

public static int value=123;//在准备阶段value初始值为0 。在初始化阶段才会变为123 。

4. **解析**

该阶段主要完成符号引用到直接引用的转换动作。解析动作并不一定在初始化动作完成之前，也有可能在初始化之后。

5. **初始化**

初始化时类加载的最后一步，前面的类加载过程，除了在加载阶段用户应用程序可以通过自定义类加载器参与之外，其余动作完全由虚拟机主导和控制。到了初始化阶段，才真正开始执行类中定义的Java程序代码。

6. **使用**
7. **卸载**

简述java类加载机制?

虚拟机把描述类的数据从Class文件加载到内存，并对数据进行校验，解析和初始化，最终形成可以被虚拟机直接使用的java类型。

什么是类加载器，类加载器有哪些?

实现通过类的权限定名获取该类的二进制字节流的代码块叫做类加载器。

主要有一下四种类加载器

1. 启动类加载器(Bootstrap ClassLoader)用来加载java核心类库，无法被java程序直接引用。
2. 扩展类加载器(extensions class loader):它用来加载 Java 的扩展库。Java 虚拟机的实现会提供一个扩展库目录。该类加载器在此目录里面查找并加载 Java 类。
3. 系统类加载器（system class loader）： 它根据 Java 应用的类路径（CLASSPATH）来加载 Java 类。一般来说，Java 应用的类都是由它来完成加载的。可以通过ClassLoader.getSystemClassLoader()来获取它。
4. 用户自定义类加载器，通过继承 java.lang.ClassLoader类的方式实现。

类加载器双亲委派模型机制?

当一个类收到了类加载请求时，不会自己先去加载这个类，而是将其委派给父类，由父类去加载，如果此时父类不能加载，反馈给子类，由子类去完成类的加载。

访问修饰符public,private,protected,以及不写（默认）时的区别?

修饰符	当前类	同 包	子 类	其他包
public	√	√	√	√
protected	√	√	√	×
Default	√	√	×	×
private	√	×	×	×

如何阻止Java中的类型未检查的警告?

```
@SuppressWarnings(“unchecked”)
```

抽象类和接口

抽象类可以在不提供接口方法实现的情况下实现接口。

Java接口中声明的变量默认都是final的。抽象类可以包含非final的变量。

Java接口中的成员函数默认是public的。抽象类的成员函数可以是private，protected或者是public。

接口是绝对抽象的，不可以被实例化。抽象类也不可以被实例化，但是，如果它包含main 方法的话是可以被调用的。

面向对象软件开发的优点有哪些?

代码开发模块化，更易维护和修改。

代码复用。

增强代码的可靠性和灵活性。

增加代码的可理解性。

面向对象编程有很多重要的特性，比如：封装，继承，多态和抽象。

封装的定义和好处有哪些?

封装给对象提供了隐藏内部特性和行为的能力。对象提供一些能被其他对象访问的方法来改

> 变它内部的数据。在Java当中，有3种修饰符：public，private和protected。每一种修饰符给其他的位于同一个包或者不同包下面对象赋予了不同的访问权限。
>
> 下面列出了使用封装的一些好处：
>
> 通过隐藏对象的属性来保护对象内部的状态。
>
> 提高了代码的可用性和可维护性，因为对象的行为可以被单独的改变或者是扩展。
>
> 禁止对象之间的不良交互提高模块化。
>
> 参考这个文档获取更多关于封装的细节和示例。

多态的定义?

> 多态是编程语言给不同的底层数据类型做相同的接口展示的一种能力。一个多态类型上的操 作可以应用到其他类型的值上面。

继承的定义?

> 继承给对象提供了从基类获取字段和方法的能力。继承提供了代码的重用行，也可以在不修 改类的情况下给现存的类添加新特性。

抽象的定义？抽象和封装的不同点?

> 抽象是把想法从具体的实例中分离出来的步骤，因此，要根据他们的功能而不是实现细节来 创建类。Java支持创建只暴漏接口而不包含方法实现的抽象的类。这种抽象技术的主要目把类的行为和实现细节分离开。
>
> 抽象和封装是互补的概念。一方面，抽象关注对象的行为。另一方面，封装关注对象行为的细节。一般是通过隐藏对象内部状态信息做到封装，因此，封装可以看成是用来提供抽象的 一种策略。

N层架构

> Pojo-->dao-->daoimpl-->service-->serviceimpl-->controller-->jsp

3尾递归

> 函数中如果所有递归形式的调用都出现在函数的末尾，我们称之为尾递归。
>
> 普通递归创建stack后内存减少，而尾递归只会占用恒量的内存。

Final和static

> **Final**:不能修饰构造方法，修饰的变量不能改，修饰的方法不能被覆盖，修饰的类不能被继承

> **Static**：修饰类或者变量，全局；修饰代码块在类加载的时候就加载；

能否在运行时向static final类型的变量赋值

> 对于final类型变量，我们必须在声明时或者在构造方法中完成对它赋值，但是static final类型变量，必须在声明时赋值；
>
> 使用final关键字修饰一个变量时，是引用变量不能变，引用的对象可变

throws, throw, try, catch, finally分别代表什么意义

> **throws:**获取异常 **throw:**抛出异常 **try:**将会发生异常的语句括起来，从而进行异常的处理，**catch:**如果有异常就会执行他里面的语句， **finally:**不论是否有异常都会进行执行的语句。

3HTTP请求的GET与POST方式的区别

> - Get方式在通过URL提交数据，数据在URL中可以看到；POST方式，数据放置在HTML HEADER内提交。
> - 对于get方式，服务器端用Request.QueryString获取变量的值，对于post方式，服务器端用Request.Form获取提交的数据。
> - GET方式提交的数据最多只能有1024字节，而POST则没有此限制。
> - 安全性问题。正如在（1）中提到，使用 Get 的时候，参数会显示在地址栏上，而 Post 不会。

JSP的常用指令

> isErrorPage(是否能使用Exception对象)，isELIgnored(是否忽略表达式)

xml有哪些解析技术?区别是什么?

> 有DOM,SAX,STAX等
>
> DOM:处理大型文件时其性能下降的非常厉害。这个问题是由DOM的树结构所造成的，这种结构占用的内存较多，而且DOM必须在解析文件之前把整个文档装入内存,适合对XML的随机访问SAX:不同于DOM,SAX是事件驱动型的XML解析方式。它顺序读取XML文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理XML文件，适合对XML的顺序访问

XML文档定义有几种形式？它们之间有何本质区别?

> 两种形式 dtd schema

> 本质区别:schema本身是xml的，可以被XML解析器解析(这也是从DTD上发展schema的根本目的)

你在项目中用到了xml技术的哪些方面?如何实现的?

> 用到了数据存贮，信息配置两方面。在做数据交换平台时，将不能数据源的数据组装成XML文件，然后将XML文件压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再同XML文件中还原相关信息进行处理。在做软件配置时，利用XML可以很方便的进行，软件的各种配置参数都存贮在XML文件中。

log4j日志

> Log4j由三个重要的组件构成：日志信息的优先级，日志信息的输出目的地，日志信息的输出格式。日志信息的优先级从高到低有ERROR、WARN、 INFO、DEBUG，分别用来指定这条日志信息的重要程度；日志信息的输出目的地指定了日志将打印到控制台还是文件中；而输出格式则控制了日志信息的显 示内容。
>
> 每个Logger都被了一个日志级别（log level），用来控制日志信息的输出。日志级别从高到低分为： A：off 最高等级，用于关闭所有日志记录。 B：fatal 指出每个严重的错误事件将会导致应用程序的退出。 C：error 指出虽然发生错误事件，但仍然不影响系统的继续运行。 D：warm 表明会出现潜在的错误情形。 E：info 一般和在粗粒度级别上，强调应用程序的运行全程。 F：debug 一般用于细粒度级别上，对调试应用程序非常有帮助。 G：all 最低等级，用于打开所有日志记录。

什么是RESTful架构：

> - 一种软件架构风格，设计风格而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。
>   1. 每一个URI代表一种资源；
>   2. 客户端和服务器之间，传递这种资源的某种表现层（Representation）；
>   3. 客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。
> - 通常被认为是一个数据结构服务器，主要是因为其有着丰富的数据结构 strings、map、list、sets、 sorted sets

Redis是什么？两句话做下概括

> 是一个完全开源免费的key-value内存数据库 2. 通常被认为是一个数据结构服务器，主要是因为其有着丰富的数据结构 strings、map、 list、sets、 sorted sets
>
> - Redis使用最佳方式是全部数据in-memory。
> - Redis更多场景是作为Memcached的替代者来使用。
> - 当需要除key/value之外的更多数据类型支持时，使用Redis更合适。
> - 当存储的数据不能被剔除时，使用Redis更合适。

redis（管道，哈希）

> - Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，zset，hash等数据结构的存储。
> - Redis支持数据的备份，即master-slave模式的数据备份。
> - Redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。

redis实现原理或机制

> redis是一个key-value存储系统。和Memcached类似，但是解决了断电后数据完全丢失的情况，而且她支持更多无化的value类型，除了和string外，还支持lists（链表）、sets（集合）和zsets（有序集合）几种数据类型。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。
>
> Redis是一种基于客户端-服务端模型以及请求/响应协议的TCP服务。这意味着通常情况下一个请求会遵循以下步骤：
>
> 客户端向服务端发送一个查询请求，并监听Socket返回，通常是以阻塞模式，等待服务端响应。服务端处理命令，并将结果返回给客户端。
>
> 在服务端未响应时，客户端可以继续向服务端发送请求，并最终一次性读取所有服务端的响应。
>
> Redis管道技术最显著的优势是提高了 redis 服务的性能。
>
> 分区是分割数据到多个Redis实例的处理过程，因此每个实例只保存key的一个子集。
>
> 通过利用多台计算机内存的和值，允许我们构造更大的数据库。
>
> 通过多核和多台计算机，允许我们扩展计算能力；通过多台计算机和网络适配器，允许我们扩展网络带宽。
>
> redis的一些特性在分区方面表现的不是很好：
>
> 涉及多个key的操作通常是不被支持的。举例来说，当两个set映射到不同的redis实例上时，你就不能对这两个set执行交集操作。
>
> 涉及多个key的redis事务不能使用。
>
> 当使用分区时，数据处理较为复杂，比如你需要处理多个rdb/aof文件，并且从多个实例和主机备份持久化文件。
>
> 增加或删除容量也比较复杂。redis集群大多数支持在运行时增加、删除节点的透明数据平衡的能力，但是类似于客户端分区、代理等其他系统则不支持这项特性。然而，一种叫做presharding的技术对此是有帮助的。

Redis 有两种类型分区

> 最简单的分区方式是按范围分区，就是映射一定范围的对象到特定的Redis实例。

比如，ID从0到10000的用户会保存到实例R0，ID从10001到 20000的用户会保存到R1，以此类推。

这种方式是可行的，并且在实际中使用，不足就是要有一个区间范围到实例的映射表。这个表要被管理，同时还需要各 种对象的映射表，通常对Redis来说并非是好的方法。

哈希分区：另外一种分区方法是hash分区。这对任何key都适用，也无需是object_name:这种形式，像下面描述的一样简单：

用一个hash函数将key转换为一个数字，比如使用crc32 hash函数。对key foobar执行crc32(foobar)会输出类似93024922的整数。

对这个整数取模，将其转化为0-3之间的数字，就可以将这个整数映射到4个Redis实例中的一个了。93024922 % 4 = 2，就是说key foobar应该被存到R2实例中。注意：取模操作是取除的余数，通常在多种编程语言中用%操作符实现。

实际上，上面的集群模式还存在两个问题：

1. 扩容问题：

因为使用了一致性哈稀进行分片，那么不同的key分布到不同的Redis-Server上，当我们需要扩容时，需要增加机器到分片列表中，这时候会使得同样的key算出来落到跟原来不同的机器上，这样如果要取某一个值，会出现取不到的情况，对于这种情况，Redis的作者提出了一种名为Pre-Sharding的方式：

Pre-Sharding方法是将每一个台物理机上，运行多个不同断口的Redis实例，假如有三个物理机，每个物理机运行三个Redis实际，那么我们的分片列表中实际有9个Redis实例，当我们需要扩容时，增加一台物理机，步骤如下：

1. 在新的物理机上运行Redis-Server；
2. 该Redis-Server从属于(slaveof)分片列表中的某一Redis-Server（假设叫RedisA）；
3. 等主从复制(Replication)完成后，将客户端分片列表中RedisA的IP和端口改为新物理机上Redis-Server的IP和端口；
4. 停止RedisA。

这样相当于将某一Redis-Server转移到了一台新机器上。Prd-Sharding实际上是一种在线扩容的办法，但还是很依赖Redis本身的复制功能的，如果主库快照数据文件过大，这个复制的过程也会很久，同时会给主库带来压力。所以做这个拆分的过程最好选择为业务访问低峰时段进行。

2. 单点故障问题：

还是用到Redis主从复制的功能，两台物理主机上分别都运行有Redis-Server，其中一个Redis-Server是另一个的从库，采用双机热备技术，客户端通过虚拟IP访问主库的物理IP，当主库宕机时，切换到从库的物理IP。只是事后修复主库时，应该将之前的从库改为主库（使用命令slaveof no one），主库变为其从库（使命令slaveof IP PORT），这样才能保证修复期间新增数据的一致性。

Mongo DB

非关系型数据库(NoSql),Mongo DB很好的实现了面向对象的思想(OO思想),在Mongo DB中 每一条记录都是一个Document对象。Mongo DB最大的优势在于所有的数据持久操作都无需开发人员手动编写SQL语句,直接调用方法就可以轻松的实现CRUD操作.

Mongo DB特点

> 高性能、易部署、易使用，存储数据非常方便。主要功能特性有：
>
> 面向集合存储，易存储对象类型的数据。
>
> 模式自由。
>
> 支持动态查询。
>
> 支持完全索引，包含内部对象。
>
> 支持查询。
>
> 支持复制和故障恢复。
>
> 使用高效的二进制数据存储，包括大型对象（如视频等）。
>
> 自动处理碎片，以支持云计算层次的扩展性
>
> 支持Python，PHP，Ruby，Java，C，C#，Javascript，Perl及C++语言的驱动程序，社区中也提供了对Erlang及.NET等平台的驱动程序。
>
> 文件存储格式为BSON（一种JSON的扩展）。
>
> 可通过网络访问。

Mongo DB功能

> 面向集合的存储：适合存储对象及JSON形式的数据。
>
> 动态查询：Mongo支持丰富的查询表达式。查询指令使用JSON形式的标记，可轻易查询文档中内嵌的对象及数组。
>
> 完整的索引支持：包括文档内嵌对象及数组。Mongo的查询优化器会分析查询表达式，并生成一个高效的查询计划。
>
> 查询监视：Mongo包含一个监视工具用于分析数据库操作的性能。
>
> 复制及自动故障转移：Mongo数据库支持服务器之间的数据复制，支持主-从模式及服务器之间的相互复制。复制的主要目标是提供冗余及自动故障转移。
>
> 高效的传统存储方式：支持二进制数据及大型对象（如照片或图片）
>
> 自动分片以支持云级别的伸缩性：自动分片功能支持水平的数据库集群，可动态添加额外的机器。

Mongo DB适用场景

> 网站数据：Mongo非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

缓存：由于性能很高，Mongo也适合作为信息基础设施的缓存层。在系统重启之后，由Mongo搭建的持久化缓存层可以避免下层的数据源 过载。

大尺寸，低价值的数据：使用传统的关系型数据库存储一些数据时可能会比较昂贵，在此之前，很多时候程序员往往会选择传统的文件进行存储。

高伸缩性的场景：Mongo非常适合由数十或数百台服务器组成的数据库。Mongo的路线图中已经包含对MapReduce引擎的内置支持。

用于对象及JSON数据的存储：Mongo的BSON数据格式非常适合文档化格式的存储及查询。

redis、memcache、mongoDB 对比

mongodb和memcached不是一个范畴内的东西。mongodb是文档型的非关系型数据库，其优势在于查询功能比较强大，能存储海量数据。

和memcached更为接近的是redis。它们都是内存型数据库，数据保存在内存中，通过tcp直接存取，优势是速度快，并发高，缺点是数据类型有限，查询功能不强，一般用作缓存。

1. 性能

redis和memcache差不多，要大于mongodb

2. 操作的便利性

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. 内存空间的大小和数据量的大小

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（单点问题）

redis对于单点问题，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（持久化）

对于数据持久化和数据恢复，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. 数据一致性（事务支持）

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),其他不支持

8. 应用场景

redis：数据量较小的更性能操作和运算上

memcache：用于在动态系统中减少数据库负载，提升性能;做缓存，提高性能（适合读多写少，对于数据量比较大，可以采用sharding）

MongoDB:主要解决海量数据的访问效率问题。

Redis有什么用？只有了解了它有哪些特性，我们在用的时候才能扬长避短，为我们所用：

1. **速度快**：使用标准C写，所有数据都在内存中完成，读写速度分别达到10万/20万
2. **持久化**：对数据的更新采用Copy-on-write技术，可以异步地保存到磁盘上，主要有两种策略，一是根据时间，更新次数的快照（save 300 10 ）二是基于语句追加方式(Append-only file，aof)
3. **自动操作**：对不同数据类型的操作都是自动的，很安全
4. 快速的主--从复制，官方提供了一个数据，Slave在21秒即完成了对Amazon网站10G key set的复制。
5. **Sharding技术**： 很容易将数据分布到多个Redis实例中，数据库的扩展是个永恒的话题，在关系型数据库中，主要是以添加硬件、以分区为主要技术形式的纵向扩展解决了很多的应用场景，但随着web2.0、移动互联网、云计算等应用的兴起，这种扩展模式已经不太适合了，所以近年来，像采用主从配置、数据库复制形式的，Sharding这种技术把负载分布到多个特理节点上去的横向扩展方式用处越来越多。

这里对Redis数据库做下小结

1. 提高了DB的可扩展性，只需要将新加的数据放到新加的服务器上就可以了
2. 提高了DB的可用性，只影响到需要访问的shard服务器上的数据的用户
3. 提高了DB的可维护性，对系统的升级和配置可以按shard一个个来搞，对服务产生的影响较小
4. 小的数据库存的查询压力小，查询更快，性能更好