# 事务

传播行为：

PROPAGATION\_REQUIRED：如果当前没有事务，就新建一个事务，如果已经存在一个事务中，加入到这个事务中。这是最常见的选择。

PROPAGATION\_SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行。

PROPAGATION\_MANDATORY：支持当前事务，如果当前没有事务，就抛出异常。

PROPAGATION\_REQUIRES\_NEW：新建事务，如果当前存在事务，把当前事务挂起。

PROPAGATION\_NOT\_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

PROPAGATION\_NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。

　　PROPAGATION\_NESTED：如果一个活动的事务存在，则运行在一个嵌套的事务中。 如果没有活动事务, 则按REQUIRED属性执行

虽然有7种，但是常用的就第一种REQUIRED和第四种REQUIRES\_NEW

五个隔离级别：

ISOLATION\_DEFAULT：这是一个PlatfromTransactionManager默认的隔离级别，使用数据库默认的事务隔离级别.

另外四个与JDBC的隔离级别相对应；

ISOLATION\_READ\_UNCOMMITTED：这是事务最低的隔离级别，它充许别外一个事务可以看到这个事务未提交的数据。

这种隔离级别会产生脏读，不可重复读和幻像读。

ISOLATION\_READ\_COMMITTED：保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据。

这种事务隔离级别可以避免脏读出现，但是可能会出现不可重复读和幻像读。

ISOLATION\_REPEATABLE\_READ：这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。

它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免下面的情况产生(不可重复读)。

ISOLATION\_SERIALIZABLE：这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。

除了防止脏读，不可重复读外，还避免了幻像读。

* **未提交时读 Read uncommitted**（最低级别，任何情况都无法保证。）
* **已提交后读 Read committed**（可避免脏读的发生。）
* **可重复读 Repeatable read**（可避免脏读、不可重复读的发生。）
* **序列化（串行化） Serializable**（可避免脏读、不可重复读、幻读的发生。）

# 抽象类和接口

区别：   
  
抽象类可以有构造函数，接口不可以有构造函数   
  
接口中变量是public static final

抽象类中的方法可以被static修饰，接口中的方法不可以被static修饰

抽象类中可以有普通方法和抽象方法，接口中的方法全是抽象方法  
一个类只能继承一个抽象类，接口可以被多实现，即一个类只能继承一个类，可以实现多个接口   
  
何时使用   
  
接口主要用于模块与模块之间的调用。主要用接口来实现多继承，因为java不支持类的多继承，只能用接口   
  
抽象类主要用于当做基础类使用，即基类。如果想拥有一些方法，并且这些方法有默认实现，那么使用抽象类

# 请简述final、finalize和finally的区别

# 为什么使用线程池

1.减少了创建和销毁线程的次数，每个工作线程都可以被重复利用，可执行多个任务。

2.可以根据系统的承受能力，调整线程池中工作线线程的数目，防止因为消耗过多的内存，而把服务器累趴下(每个线程需要大约1MB内存，线程开的越多，消耗的内存也就越大，最后死机)。

Java里面线程池的顶级接口是Executor，但是严格意义上讲Executor并不是一个线程池，而只是一个执行线程的工具。真正的线程池接口是ExecutorService。

①Executors.newSingleThreadExecutor

单个线程的线程池，即线程池中每次只有一个线程工作，单线程串行执行任务

②Executors.newFixedThreadExecutor(n)

固定数量的线程池，没提交一个任务就是一个线程，直到达到线程池的最大数量，然后后面进入等待队列直到前面的任务完成才继续执行

③Executors.newCacheThreadExecutor（推荐使用）

可缓存线程池，**当线程池大小超过了处理任务所需的线程，那么就会回收部分空闲（一般是60秒无执行）的线程**，当有任务来时，又智能的添加新线程来执行。

④newScheduleThreadExecutor

大小无限制的线程池，支持定时和周期性的执行线程

# String,StringBuffer与StringBuilder的区别

String的值是不可变的，这就导致每次对String的操作都会生成**新的String对象**，这样不仅效率低下，而且大量浪费有限的内存空间。

StringBuffer可变 线程安全

StringBuilder可变 线程不安全

和 String 类不同的是，StringBuffer 和 StringBuilder 类的对象能够被多次的修改，并且**不产生新的未使用对象**。

StringBuilder 类在 Java 5 中被提出，它和 StringBuffer 之间的最大不同在于 StringBuilder 的方法不是线程安全的（不能同步访问）。

由于 StringBuilder 相较于 StringBuffer 有速度优势，**所以多数情况下建议使用 StringBuilder 类**。然而在应用程序要求线程安全的情况下，则必须使用 StringBuffer 类。

# 请你说明一下int 和 Integer 有什么区别

对象引用实例变量  
的缺省值为null，而原始类型实例变量的缺省值与它们的类型有关。

# 关键字

## Volatitle

被volatitle修饰的变量能够保证可见性，不保证原子性，每个线程能够获取该变量的最新值。

实现的机制：在写volatitle变量写到主内存时，指令前会加上lock，该指令有两个影响：

将当前处理器缓存行的数据写回系统内存；

这个写回内存的操作会使得其他CPU里缓存了该内存地址的数据无效。

在多核处理器中，其他线程发现本地缓存失效，就会到主内存重读这个变量，因此在一个volatitle变量发生变化，会发生以下变化：

Lock前缀的指令会引起处理器缓存写回内存；

一个处理器的缓存回写到内存会导致其他处理器的缓存失效；

当处理器发现本地缓存失效后，就会从内存中重读该变量数据，即可以获取当前最新值；

这样针对volatile变量通过这样的机制就使得每个线程都能获得该变量的最新值。

# 缓存穿透，击穿，雪崩的解决方案

穿透：访问数据库null也设置null到redis

击穿：

1，使用互斥锁(mutex key): 这种解决方案思路比较简单，就是只让一个线程构建缓存，其他线程等待构建缓存的线程执行完，重新从缓存获取数据就可以了

2，用永不过期+随机过期时间

这里的“永远不过期”包含两层意思：

(1) 从redis上看，确实没有设置过期时间，这就保证了，不会出现热点key过期问题，也就是“物理”不过期。

(2) 从功能上看，如果不过期，那不就成静态的了吗？所以我们把过期时间存在key对应的value里，如果发现要过期了，通过一个后台的异步线程进行缓存的构建，也就是“逻辑”过期



# [java中public，private，protected和default的区别](https://www.cnblogs.com/heqiyoujing/p/11222083.html)

      priavte          本类可见

      public            所有类可见

      protected      本包和所有子类都可见（本包中的子类非子类均可访问，不同包中的子类可以访问，不是子类不能访问）

      default          本包可见（即默认的形式）（本包中的子类非子类均可访问，不同包中的类及子类均不能访问）