1、对象锁的同步和异步

同步的概念就是实现多线程下的数据的共享，如果资源不共享 就没必要实现同步 ，同步的目的就是实现线程安全，对于线程同步来说需要满足两个特征 ：原子性，可见性

异步的概念是独立的互相不受约束互相不影响的

1. threadLocal：线程的局部变量，是一种多线程间并发访问变量的解决方案（变量在当前线程内可见、有效）
2. **Volatile ：实现变量在多线程间 可见 （Volatile 会强制线程到主内存中 去读取变量 然后告诉线程工作区 主内存中的变量值是否发生变化 ）**
   1. **注意：Volatile 并不具备 同步的特性（原子性：当被Volatile 需要改的关键字已经发生变化时 ， 可能另一个线程还在操作 未发生变化的变量）**

**如果同步的话，当其中一个线程操作这个变量时 ，其他线程就不能操作此变量**

4、线程通信

使用**synchronized 、 wait、notify可以实现线程间的相互通信**

5、Iterable不是线程安全的， 如果使用迭代器迭代一个集合的过程中，对集合元素进行操作 会抛currentModificationException

6、在java1.5之前 ，并发类容器 都是使用 hashtable、vector实现并发类的容器 ，其内部的方法都是使用**synchronized 关键子修饰，只允许一个线程操作 ，但是性能交差**

1. 在java1.5 以后，提供了各种各样的并发类容器 且性能较高。 ConcurrentHashMap

代替hashtable 。copyOnWriteArrayList代替vector

* 1. ConcurrentHashMap 继承 AbstractMap 无排序功能
  2. ConcurrentSkipListMap继承AbstractMap 有排序功能 （treemap也具备排序）

1. 多线程的设计模式 ： future模式（jdk已经封装好future模式了）、master-work模式
   1. Future : 并行且异步的方式处理任务并返回线程的执行结果 （在处理任务时，首先返回处理数据的接口，并没有真正去执行任务 ，然后启动一个线程去执行真正的任务，并将结果返回代理对象 类似懒加载的感觉）

8、线程池

**AtomicInteger**是一个提供原子操作的**Integer**类，通过线程安全的方式操作加减。

1、为什么要用线程池:

（1）实现线程的重复利用，如果没有线程池 ，线程用完后会 被销毁

（2）根据系统自身的性能 合理的设置线程中的数量

2、线程池分类：固定尺寸的线程池、可变尺寸连接池

Executors

Executors 是一个工具类，类似于 Collections。提供工厂方法来创建不同类型的线程池，比如 FixedThreadPool 或 CachedThreadPool。

线程池的队列机制： 假如创建的线程池中有5个固定的线程， 但是有 6 个任务需要执行 ， 那么线程池有一个队列机制 哪一个线程先执行完，就会执行第六个任务

Executors 类的方法

newFixedTreadPool ： 创建一定数量的线程

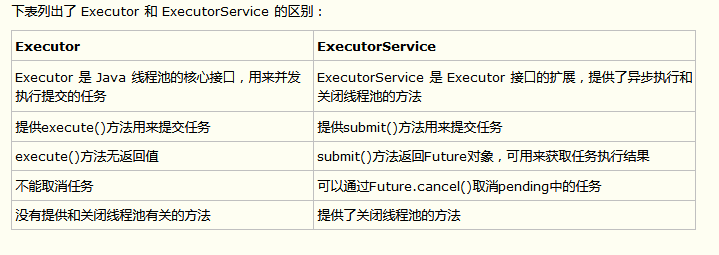
newSingleThreadExecutor： 创建一个线程池 ，内部只有一个线程 ，若有空闲线程则执行任务，没有线程则暂缓在任务对列中

newCachedThreadPool ：根据实际情况调整线程池中线程的数量，不是设置最大线程数量（内存可控），有任务创建线程，没有任务就不创建线程。如果线程被创建后没有任务则在60s后进行释放

newScheduleThreadPool：返回ScheduleExecutor对象可以设置线程池中线程数量 （可以实现定时任务）

Java提供四种线程池的创建， 底层都是创建threadPoolExecute实例 ， 当java提供的四种线程池不能满足特定的业务需求时 ，可以通过 threadPoolExecute 自定义线程池

Executor 、ExecutorService的区别



AbstractExecutorService抽象类实现ExecutorService

threadPoolExecutor继承AbstractExecutorService

自定义线程池 通过threadPoolExecutor来实现

有界队列 （arrayBlockQueue）：根据系统资源合理的创建线程数量，来执行任务 如果任务较多，则会将任务放入队列中 一旦有空闲的线程再去执行队列中的任务

无界队列（linkBlockQueue ） ：除非系统资源耗尽 ， 否则不会出现 任务入队失败的情况（只要来任务就放到无界队列中）

**New**ThreadPoolExecutor(nThreads,nThreads,0L, TimeUnit.***MILLISECONDS***,**new** LinkedBlockingQueue<Runnable>(),threadFactory);

参数1： 核心线程数量

参数2： 最大线程数量

参数3、参数4： 线程空闲时，缓存的时间

参数5：队列接口 ，可以是有界队列实现，也可以是无界队列实现

参数6：拒绝的接口

注意：通过threadpoolExecutor来实现线程池时 ， 参数的设置 跟 使用有界队列还是 无界队列有紧密的关系

有界队列参数： 当有新的任务需要执行时 ，如果当前线程池中线程的数量小于 coreaize数量，则创建线程 ，如果大于coresize的数量 会将此任务放入队列中 ，如果对列已满 ，在线程数不大于 maxSize的前提下创建线程 ，如果 线程数量大于 maxSize ，则执行撤回策略抛出异常（拒绝策略）

有界队列参数： 当任务个数小于线程池中线程coreSize的个数时 ， 有任务就创建线程 ， 当大于coreSize个数时 ，来的线程就一次放入对列中 直至系统资源耗尽（maxsize 一般都与coreSize的值相等， 如果大约coreSize 没有任何意义）

Jdk提供的撤回策略 （为了保证数据不丢失，都会自定义策略）

1. abortPolicy : 直接抛异常 不使用
2. callerRunsPolicy 不使用
3. DiscardOldRunsPolicy： 丢弃先进入队列的任务 不使用

（4）DiscardPolicy 不使用

自定义策略 ： 实现rejectedExecutionHandler

Threadlocal 类

1. 为什么有Threadlocal类
   1. 并发的根本原因 ，是多线程操作 同一个 资源 ，可能会导致数据的不一致

2、主要用来提供 线程局部变量 ，变量只对当前线程可见 ，如果一个变量或对象只对一个线程可见 ，就不会不限同一资源被多个线程共享的问题