Multithread

Runnable & Thread

可以通过implements前者的方式，更加灵活（因为还可以implements或者extends其他类）。而后者只能用extends（Java不支持多重继承。）

Interrupt

- try...catch(InterruptedException e)
- Thread.interrupted(Thread代表当前线程)

Joins

t.join()会让当前线程暂停并等待线程t执行完毕。

join和sleep一样依赖于操作系统，不能保证完全精确。

像 sleep 一样，join 通过以 InterruptedException 退出来响应中断。

Synchronized

Java中每个对象都有一把内在锁(Intrinsic lock)，Synchronized关键字作用在方法上，会对方法利用对象内在锁进行上锁。该锁是可重入的(Reentrant)。

对类的静态字段的访问由一个与该类的任何实例的锁不同的锁控制。

利用Synchronized关键字还可以上细粒度的锁：

粗粒度：

```
public synchronized void func() {
    doSomething();
    doSomething();
    doSomething();
}
```

细粒度：

```
public void func() {
    doSomething();
    synchronized(this) {
        doSomething();
    }
    doSomething();
}
```

可以利用Object自带内在锁的性质，多重上锁。

```
Object lock1, lock2;
synchronized(lock1) {
    doSomething();
}
synchronized(lock2) {
    doSomething();
}
```

Volatile

在一些32位的处理器上,如果要求对64位数据的写操作具有原子性,开销会比较大。

jvm不要求对64位long和double类型变量的写操作具有原子性。

当jvm在这种处理器上运行时候,可能会把一个64位的long/double类型变量的写操作拆分为两个32位的写操作来执行。

这两个操作可能会被分配到不同的总线事务中执行,此时对64位变量的写操作不具有原子性.

- 可见性: 对一个volatile的读,总是能看到任意线程对这个volatile最后的写
- 原子性: 对任意单个volatile变量的读/写具有原子性,但类似于volatile++这种复合操作不具有原子性

Wait & Notify

- using notifyAll() & wait().
- producer & consumer model.

Random

- Random 线程安全，但是消耗资源大
- LocalThreadRandom 线程间可能共享随机数：用current()解决该问题。

Immutable Object

不可变对象比较适合多线程环境，比如String就是不可变对象（不同于StringBuilder）

不可变对象：

- 没有public的setter；
- 成员对象为private & final;
- 不允许被继承，类应用final关键字修饰；亦可以让构造函数变为私有，用工厂模式获取对象实例。
- 不要暴露mutable对象（提供修改接口或者暴露引用）

不可变对象的gc其实没有看起来那么耗时。

Lock Objects

Lock 对象相对于隐式锁的最大优势是它们能够退出获取锁的尝试。

– 如果锁不立即可用或在超时到期之前（如果指定），则 tryLock 方法将退出。

– 如果另一个线程在获取锁之前发送中断，则 lockInterruptably 方法会退出

Advanced

- Executor:

  ExecutorService 接口用类似但更通用的提交方法补充了 execute。

  – 和 execute 一样，submit 接受 Runnable 对象，但也接受 Callable 对象，它允许任务返回一个值。

  – submit 方法返回一个 Future 对象，用于检索 Callable 返回值并管理 Callable 和 Runnable 任务的状态

- ThreadPool:

  线程池，控制线程数量（不断new线程出来可能会导致内存溢出。）

- Fork/Join:

  适合将任务分成很多小的子任务执行：比如图像处理。

- Atomic:

  AtomicInteger

  incrementAndGet()

  decrementAndGet()