并发编程文档

一、线程的创建

1. 继承 Thread 类

```
class extThread extends Thread{
    @Override
    public void run() {
        LogUtil.info("Extend thread running ..");
    }
}

public class TestMain{
      public static void main(String[] args) {
        Thread extThread = new extThread();
        extThread.start();
      }
}
```

2. 实现 Runnable 接口

```
class implThread implements Runnable{
    @Override
    public void run() {
        LogUtil.info("Implement thread running ..");
    }
}

public class TestMain{
      public static void main(String[] args) {
        Thread t2 = new Thread(new implThread());
        t2.start();
      }
}
```

> 两种方式的区别：
>
> 因为线程类继承了 Thread 类，所以不能再继承其他的父类，因此，通过实现 Runnable 接口，可以实现多继承的功能。实现 Runnable 接口的方式来创建线程，比较灵活。

二、线程状态

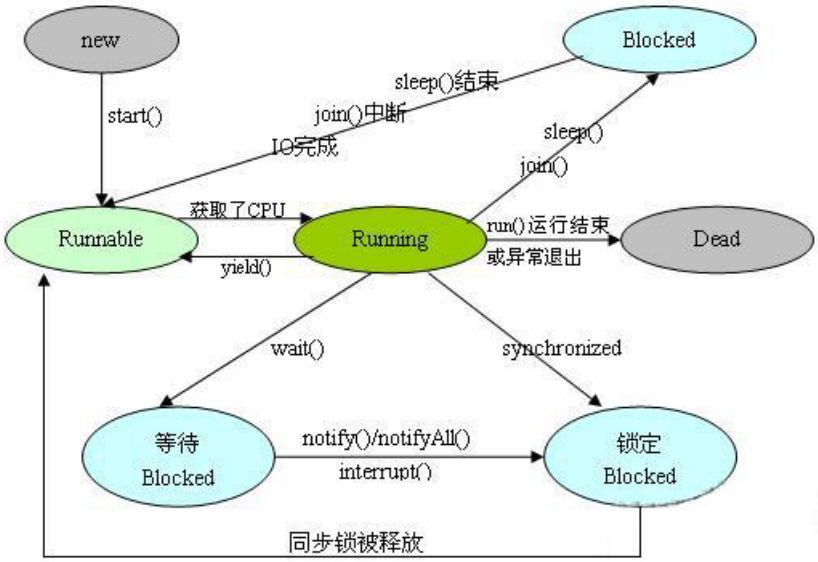
Java JDK 中，线程状态有六种：

- **NEW**：新建的线程，至今尚未启动的线程处于这种状态
- **RUNNABLE**：就绪状态，一个线程调用 start()方法后，正在 Java 虚拟机中执行的线程处于这种状态，等待 CPU 调度到之后开始执行（Running）
- **BLOCKED**：受阻塞并等待某个监视器锁的线程处于这种状态（如线程中调用 sleep()方法）
- **WAITING**: 无限期地等待另一个线程来执行某一特定操作的线程处于这种状态，当使用 wait() 方法后，线程会进入此状态
- **TIMED_WAITING**: 等待另一个线程来执行取决于指定等待时间的操作的线程处于这种状态，当线程调用 wait(long timeout) 方法后，会进入此状态
- **TERMINATED**:已退出的线程处于这种状态，线程 Dead

> 在给定时间点上，一个线程只能处于一种状态

三、线程状态变换

如图，状态变换：



> **疑问：**
>
> 1. 线程状态改变的时候，当 Runnable 状态，调用 join() 后，状态变成 Blocked , 该状态何时会再变得可执行？

四、线程同步

1.synchronized 同步方法

示例：

```
public synchronized  void add(float amt){
    ...
   }
```

1. 在方法上面添加同步关键字（synchronized），可以防止多个线程调用**同一个实例对象方法（不是同一个对象实例，无法实现同步效果）**的时候产生的数据安全问题。
2. 由于 Java 的每个对象都有一个内置锁，当用此关键字修饰方法时， 内置锁会保护整个方法。在调用该方法前，需要获得内置锁，否则就处于阻塞状态。

> synchronized 关键字也可以修饰静态方法，此时如果调用该静态方法，将会锁住整个类

2.synchronized 同步代码块

示例：

```
    synchronized(object){
    }
```

即有 synchronized 关键字修饰的语句块。 被该关键字修饰的语句块会自动被加上内置锁，从而实现同步。

3.使用特殊域变量(volatile)实现线程同步

> 多线程中的非同步问题主要出现在对域的读写上，如果让域自身避免这个问题，则就不需要修改操作该域的方法

1. volatile 关键字为域变量的访问提供了一种免锁机制
2. 使用 volatile 修饰域相当于告诉虚拟机该域可能会被其他线程更新
3. 每次使用该域就要重新计算，而不是使用寄存器中的值
4. volatile **不会提供任何原子操作**，它也不能用来修饰 final 类型的变量 (final 不可改变)

4. ThreadLocal 的方式

待完善...
