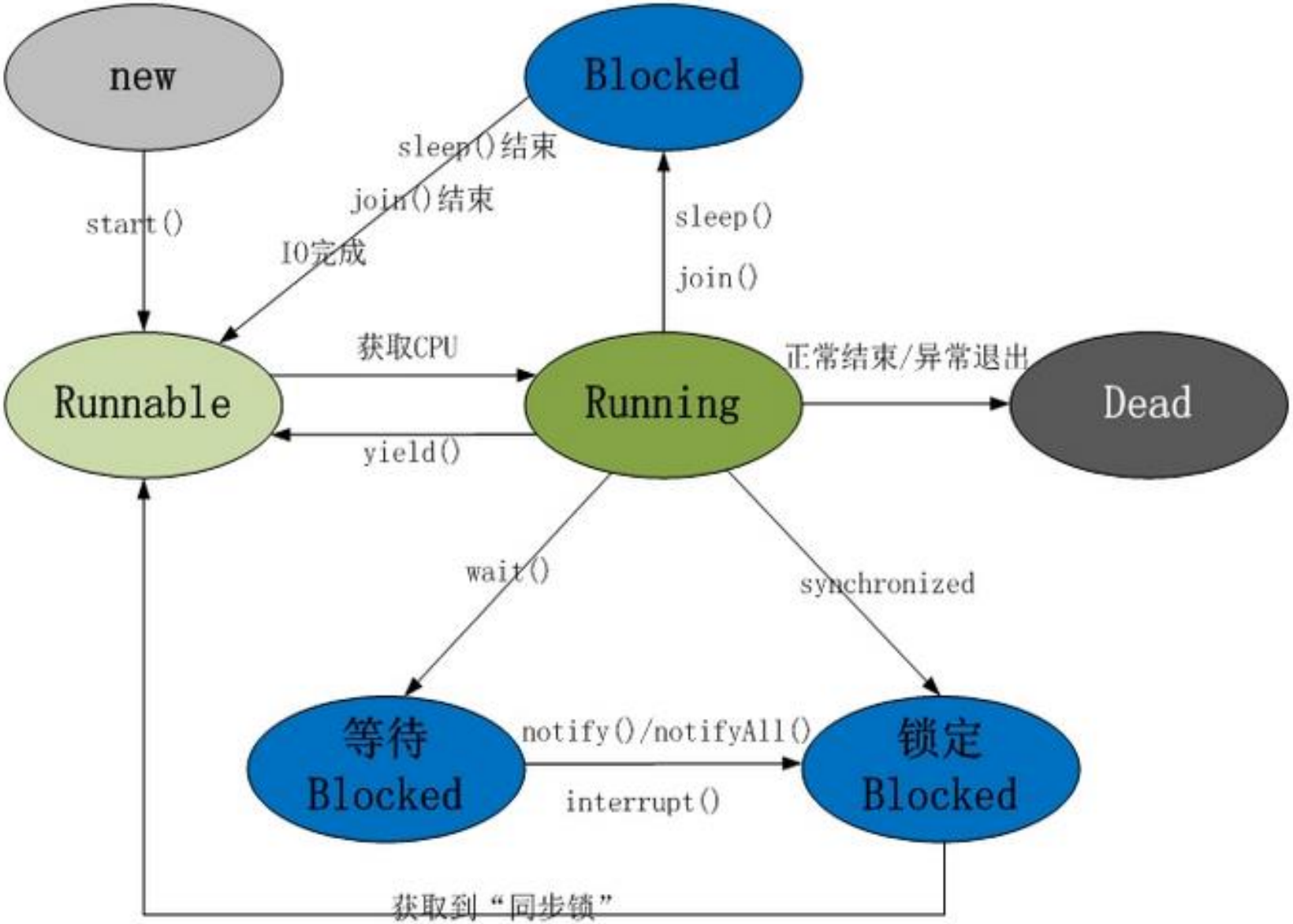
第十七章 多线程编程

线程的状态

- 线程可以分成以下五种状态
  - New（新建）
  - Runnable（就绪）
  - Running（运行）
  - Blocked（阻塞）
  - Dead（死亡）

线程图解


new
Blocked
sleep()结束
join()结束
IO完成
start()
sleep()
join()
获取CPU
Runnable
Running
正常结束/异常退出
Dead
yield()
wait()
synchronized
等待
Blocked
notify()/notifyAll()
interrupt()
锁定
Blocked
获取到“同步锁”

主线程

当Java程序启动时，一个线程立刻运行，该线程通常叫做程序的主线程（main thread），它是程序开始时就执行的

创建线程

- 创建线程
- 两种方式
  - 继承自Thread类
  - 实现Runnable 接口。

创建线程

希望主线程最后结束

1. sleep()
   暂停多长时间
1. isAlive()
   判断线程仍在运行，isAlive()方法返回true，如果不是则返回false。
2. join()
   等待一个线程终止

线程的优先级

- 理论上，优先级高的线程比优先级低的线程获得更多的CPU时间
- 实际上，线程获得的CPU时间通常由包括优先级在内的多个因素决定
- 理论上，等优先级线程有同等的权利使用CPU。

线程其它方法

- interrupt()线程中断
  - sleep()和join()
- yield() 线程让步
  - Thread.yield()

练习：

多线程实现程序。

- 每隔1秒显示当前线程的名字，共显示20次。
- 启动两个子线程实现。

16.2线程的同步

使用同步:

- **1. 同步方法**

  synchronized 方法{ }

- **2. 同步代码块**
  - 我们可以将对某个方法的调用放入一个synchronized块内

```
synchronized(对象){

    同步块;

}
```

Lock

- ReentrantLock锁
- 优势
- 语法：

```
try{
        加锁lock
}finally{
        释放unLock
}
```

线程的同步

- 关于同步
- 每个对象只有一个锁（lock）与之相关联
- 实现同步是要很大的系统开销作为代价的，甚至可能造成死锁，所以尽量避免无谓的同步控制

死锁

- **死锁**
- 死锁发生在当多个线程进入到了循环等待状态
- 死锁是很难调试的错误
  - 通常，它极少发生，只有到两线程的时间段刚好符合时才能发生。
- 我们在编写多线程并含有同步方法调用的程序中要格外小心，避免死锁的发生。

wait和sleep区别

- 区别：
  - 时间参数
    - ✓ wait()可以可定时间也可以不指定；
    - ✓ sleep()必须指定时间；
  - 同步状态：
    - ✓ sleep()释放执行权，不释放锁
    - ✓ wait释放执行权，释放锁