本讲大纲：

1、线程简介

2、实习线程的两种方式
- 1、继承Thread类
- 2、实现Runnable接口

3、线程的生命周期

4、线程的操纵方法
- 1、线程的休眠
- 2、线程的加入
- 3、线程的中断
- 4、线程的礼让

5、线程的优先级

6、线程安全：　线程同步

7、线程池

线程简介

世间万物会同时完成很多工作，用户既可以使用计算机听歌的同时也可以使用它打印文件，而这些活动完以同时进行，这种思想在Java中被称为并发，而将并发完成的每一件事情称为线程。

Java语言提供并发机制，程序员可以在程序中执行多个线程，每一个线程完成一个功能，并与其他线程并发执行，这种机制被称为多线程。

实现线程方式1：继承Thread类

- 继承Thread类创建一个新的线程的语法如下：

```
public class ThreadTest extends Thread{
        pulbic void run()   {
            //这里编写线程要完成的任务
             }
}
```

- 启动线程的语法： 线程对象.start();

  如：New ThreadTest().start();

实现线程方式2：实现Runnable接口

通过扩展Thread类来创建非常方便，但如果程序员需要继承其他类（非Thread类）并且要使该类具备使用线程的功能，就需要通过实现Runnable接口来实现线程。

实现线程方式2：实现Runnable接口

- 实现Runnable接口的语法如下：

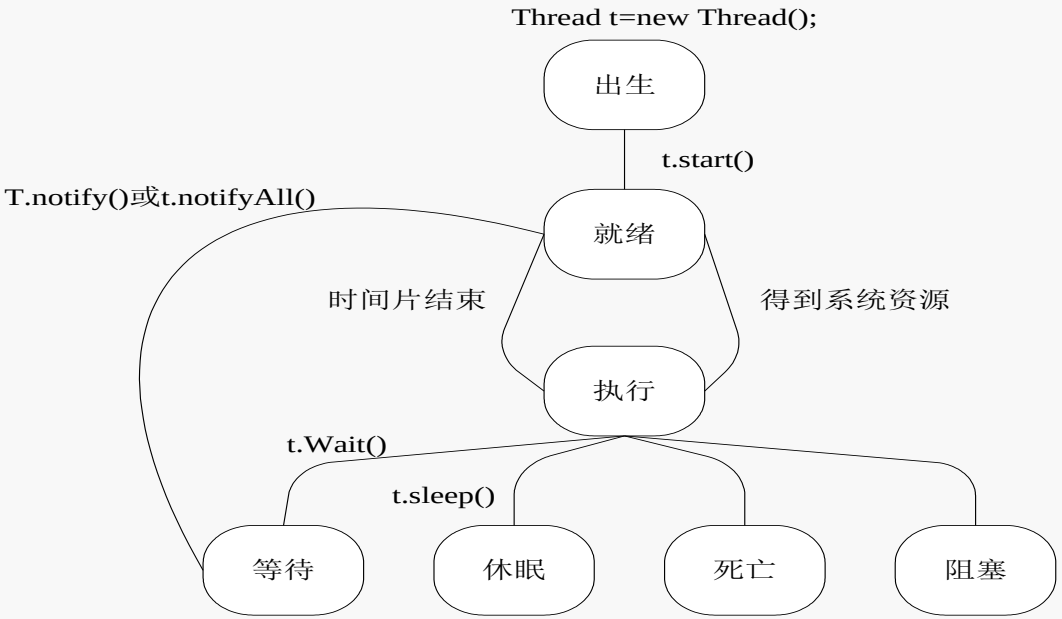
```
public class myRun implements Runnable
{
  publick void run()
   {
        //线程需要完成任务代码
   }
}
```

创建和启动线程：

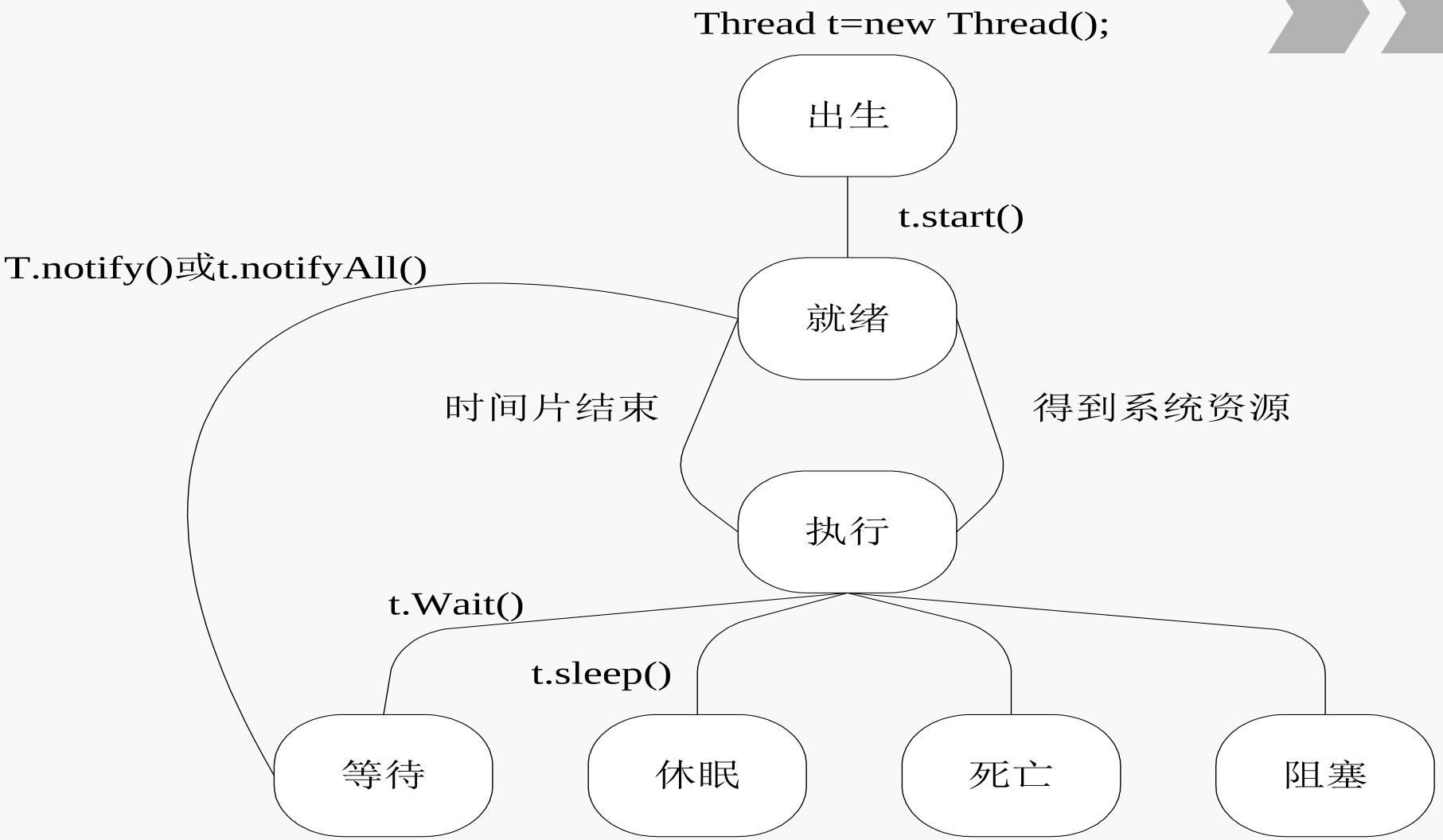
```
  Thread t = new  Thread(new myRun());
              t.start();
```

线程的生命周期

线程具有生命周期，其中包含7种状态，分别为出生状态、就绪状态、运行状态、等待状态、休眠状态、阻塞状态和死亡状态。



线程的生命周期


Thread t=new Thread();
出生
t.start()
T.notify()或t.notifyAll()
就绪
时间片结束
得到系统资源
执行
t.Wait()
t.sleep()
等待
休眠
死亡
阻塞

操纵线程：线程的休眠

sleep()方法需要一个参数用于指定该线程休眠的时间，该时间使用毫秒为单位。它通常是在run()方法内的循环中被使用。

sleep()方法的语法如下：

```
try{
        Thread.sleep(2000);
}catch(InterruptedException e){
        e.printStackTrace();
}
```

操纵线程：线程的加入

如果当前某程序为多线程程序，假如存在一个线程A，现在需要插入线程B，并要求线程B先执行完毕，然后再继续执行线程A，此时可以使用Thread类中的join()方法来完成。

当某个线程使用join()方法加入到另外一个线程时，另一个线程会等待该线程执行完毕再继续执行。

操纵线程：线程的中断

在以往时候会使用stop()方法停止线程，但也不建议使用stop()方法来停止一个线程的运行。现在提倡在run()方法中使用无限循环的形式，然后使用一个布尔型标记控制循环的停止。

注意：使用 Thread. **interrupt()**， **并不能保证线程马上中断。**

线程中断本质： http://www.jb51.net/article/32359.htm

操纵线程：线程的礼让

Thread类中提供了一种礼让方法，使用yield()方法表示，它只是给当前正处于运行状态下的线程一个提醒，告知它可以将资源礼让给其他线程，但这仅仅是一种暗示，没有任何一种机制保证当前线程会将资源礼让。

yield()方法使具有同样优先级的线程有进入可执行状态的机会，当当前线程放弃执行权时会再度回到就绪状态。对于支持多任务的操作系统来说，不需要调用yeild()方法，因为操作系统会为线程自动分配CPU时间片来执行。

线程的优先级

每个线程都具有各自的优先级，线程的优先级可以在程序中表明该线程的重要性，如果有很多线程处于就绪状态，系统会根据优先级来决定首先使哪个线程进入运行状态。但这并不意味着低优先级的线程得不到运行，而只是它运行的几率比较小，比如垃圾回收线程的优先级就较低。

设置优先级：Thread.setPriority(int n); // 1<= n<=10

Thread.MIN_PRIORITY（常数1）、

Thread.MAX_PRIORITY（常数10）

Thread.NORM_PRIORITY（常数5）

每个新产生的线程都继承了父线程的优先级。

多线程产生的安全问题

实际开发中，使用多线程程序的情况下，如银行排号系统、火车站售票系统等。这种多线程的程序处理不好，通常会发生问题。以火车站售票系统为例，当两个线程同时访问这段代码时（假如这时只剩下一张票），第一个线程将票售出，与此同时第二个线程也已经执行完成判断是否有票的操作，并得出结论票数大于0，于是它也执行售出操作，这样就会产生负数。所以在编写多线程程序时，应该考虑到线程安全问题。

线程安全问题来源于两个线程同时存取单一对象的数据。

线程安全问题的解决方法：线程同步机制

如何解决资源共享的问题？基本上所有解决多线程资源冲突问题都会采用给定时间只允许一个线程访问共享资源，这时就需要给共享资源上一道锁。这就好比一个人上洗手间，这个人进入洗手间后将门锁上，当他出来时再将锁打开，然后其他人才可以进入。Java中上锁的方式有：

同步块： synchronized(Obj obj){ // 同步代码块 }

同步方法：synchronized void myFun｛//同步方法｝

补充内容：线程池技术

1、为什么要用线程池： 减少创建和销毁线程的次数，每个工作线程可以多次使用 可根据系统情况调整执行的线程数量，防止消耗过多内存。根据系统自身的环境情况，有效的限制执行线程的数量，使得运行效果达到最佳。

2、线程池的实现机制：线程池主要是通过控制执行的线程的数量，超出数量的线程排队等候，等待有任务执行完毕，再从队列最前面取出任务执行。

3、官方建议的线程池实现方法 见 示例 demo2

补充内容：线程池技术

```
ExecutorService cachedThreadPool =
Executors.newCachedThreadPool();


ExecutorService fixedThreadPool =
 Executors.newFixedThreadPool(3);

ScheduledExecutorService scheduledThreadPool =
Executors.newScheduledThreadPool(5);

ExecutorService singleThreadExecutor =
Executors.newSingleThreadExecutor();
```