1.进程:系统进行资源分配调用的独立单元叫进程.每个进程都有自己独立内存空间和系统资　　源.(正在运行的程序)

　　cpu时间片:指的是系统资源和内存空间.

2.线程:进程中的一条执行线路.每个线程要执行一个任务.进程中所有线程共享当前这个进程　　中系统资源和内存空间.

　　同一个进程中多个线程之间是互抢资源竞争关系.

3.进程与线程的关系:一个进程中可以一个到多个线程;一个线程只属于一个进程.

4.实现线程:有三种实现方式,前两种公司常用,第三种基本不用.

　　4.1:继承Thread类,来实现线程

　　eg:public class MyThead extends Thread{

```
    /**
     * 重写父类中线程的任务方法
     */
    @Override
    public void run() {
        for (int i = 1; i <=10; i++) {
            System.out.println(Thread.currentThread().getName()+":"+i);

        }
    }
}

public static void main(String[] args) {
        //创建子线程对象
        MyThead t1=new MyThead();
        MyThead t2=new MyThead();

        //启动线程
        t1.start();
```

```
        t2.start();
    }

    4.2:实现Runnable接口,实现线程.
        eg:public class MyRunable implements Runnable{
    /**
     * 实现父接口中任务方法
     */
    @Override
    public void run() {
        for (int i = 1; i <=10; i++) {
            System.out.println(Thread.currentThread().getName()+":"+i);
        }
    }
}

public static void main(String[] args) {
        //创建任务对象
        MyRunable r1=new MyRunable();
        MyRunable r2=new MyRunable();
        //用任务构建线程对象
        Thread t1=new Thread(r1, "线程A");
        Thread t2=new Thread(r2, "线程B");
        //启动线程
        t1.start();
        t2.start();
    }

    4.3:实现Callable接口,实现线程(现在基本不用).

5.继承Thread VS 实现Runnable接口的方式实现多线程
    5.1:代码简洁性不同:继承Thread的方式实现线程,代码较简洁;实现Runnable
接口的方式                    实现线程,代码较复杂.
    5.2:扩展性不同:继承Thread的方式实现线程,不能再继承其他类,只能实现其
```

他接口,所以扩展性较差;实现Runnable接口的方式实现线程,还可以再继承其他类实现其他接口,所以扩展性好.

5.3:资源共享性:继承Thread的方式实现线程,如果想让多个线程执行同一个任务,只能用静态的,静态比较耗资源,共享性不好;实现Runnable接口的方式实现线程,如果想让多个线程执行同一个任务,只需要让这多个线程共用同一个任务对象就可,共享较好.

6.给线程取名:

6.1:用线程对象调用setName("线程名");

6.2:用构造方法给线程取名.

6.3:用属性方式给线程取名.

6.4:用线程默认名称.

7.线程休眠:使当前正在执行的线程以指定的毫秒数暂停执行或暂停资源抢夺,等待时间到了后再重新参加资源抢夺.

语法:Thread.sleep(毫秒); 或 线程对象.sleep(毫秒);

```
eg:public static void main(String[] args) throws InterruptedException {
        for (int i = 6; i >=1; i--) {
            System.out.println(Thread.currentThread().getName()+":"+i);
            //每输出一个数字就让当前线程休眠一秒钟
            Thread.sleep(1000);
        }
  }
```

8.线程优先级:线程优先级由1-10,数字越大优先级越高.优先级越高的线程抢占资源的概率越高,但并不一定就能抢得到;优先级越低的线程抢占资源的概率越低,但并不一定抢不到.注意:一定要在线程启动之前设置线程优先级.

语法:线程对象.setPriority(int newPriority)

```
eg:public static void main(String[] args) {
//创建线程对象
MyThead t1=new MyThead();
MyThead t2=new MyThead();

//设置线程优先级
```

```
        t1.setPriority(Thread.MIN_PRIORITY);
        t2.setPriority(Thread.MAX_PRIORITY);

        //启动线程
        t1.start();
        t2.start();
}
```

9.线程合并:将两个及两个以上的线程合并为一个线程,合并过来的线程先执行完,再执行原来 线程.语法:线程对象.join()

9.1:子线程合并到主线中:合并之前,子线程和主线程互抢资源,当前子线程合并过来后就 变成一条执行线路,子线程要先执行完再执行主线程后面的内容.

```
        eg:public class MyThead extends Thread{
    /**
     * 重写父类中线程的任务方法
     */
    @Override
    public void run() {
        for (int i = 1; i <=100; i++) {
            System.out.println(Thread.currentThread().getName()+":"+i);
        }
    }
}

public static void main(String[] args) throws InterruptedException {
        //创建一个子线程
        MyThead t1=new MyThead();
        //设置线程名称
        t1.setName("子线程");

        //启动子线程
        t1.start();
```

```
            for (int i = 1; i <=100; i++) {
                System.out.println(Thread.currentThread().getName()+":"+i);
                //当主线程运行到10时,子线程合并过来,子线程先执行完再执行主
线程
                if (i==10) {
                    t1.join();
                }
            }
        }
```

9.2:子线程B合并到子线程A中:合并之前,两个线程互抢资源,子线程B合并子线程A后,

子线程B要先执行完,再执行子线程A.

```
    eg:public class MyThead2 extends Thread{
    //声明一个属性,存线程对象
    public MyThead2 t;

    //通过构造方法给线程取名
    public MyThead2(String name) {
        super(name);
    }

    /**
     * 重写父类中线程的任务方法
     */
    @Override
    public void run() {
        for (int i = 1; i <=100; i++) {
            System.out.println(Thread.currentThread().getName()+":"+i);
            //如果线程A运行到10,
            if ("线程A".equals(Thread.currentThread().getName())&&i==10)
{
                //让线程B合并过来,this指代当前线程A对象
                try {
```

```
                        this.t.join();
                    } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            }
        }
    }

    public static void main(String[] args) {
        //创建线程对象
        MyThead2 t1=new MyThead2("线程A");
        MyThead2 t2=new MyThead2("线程B");

        //将线程B对象作为线程A对象的属性传过来
        t1.t=t2;

        //启动线程
        t1.start();
        t2.start();
    }
```

10.线程礼让:暂停一下当前线程正在执行任务或资源的抢占,再接着执行任务或抢占资源.

语法:Thread.yield() 或 线程对象.yield()

```
eg:public static void main(String[] args) {
        //创建子线程
        MyThead t1=new MyThead();

        //启动线程
        t1.start();

        for (int i = 1; i <=100; i++) {
```

```
                System.out.println(Thread.currentThread().getName()+":"+i);
                //主线程每运行一次就要礼让一下
                Thread.yield();
            }
        }
```

11.线程中断

11.1:用中断方法改变中断状态,再根据中断状态来中断线程.

```
        eg:public class MyThead1 extends Thread{
    /**
     * 重写父类中线程的任务方法
     */
    @Override
    public void run() {
        for (int i = 1; i <=100; i++) {
            //判断子线程中断状态
            if (Thread.currentThread().isInterrupted()==true) {
                break;
            }
            System.out.println(Thread.currentThread().getName()+":"+i);
        }
    }
}

public static void main(String[] args) throws InterruptedException {
        //创建子线程
        MyThead1 t1=new MyThead1();
        //启动线程
        t1.start();

        for (int i = 1; i <=100; i++) {
            if (i==10) {
                t1.interrupt();//改变线程中断状态
                Thread.sleep(2000);
```

```
                }
                System.out.println(Thread.currentThread().getName()+":"+i);
            }
        }
```

11.2:在线程类中声明一个变量作标记,标记是否中断线程

```
        eg:public class MyThead2 extends Thread{
    //声明一个变量作标记,标记是否中断线程,默认不中断
    public boolean flag=false;

    /**
     * 重写父类中线程的任务方法
     */
    @Override
    public void run() {
        for (int i = 1; i <=100; i++) {
            //根据标记判断子线程是否中断
            if (flag) {
                break;
            }
            System.out.println(Thread.currentThread().getName()+":"+i);
        }
    }
}

        public static void main(String[] args) throws InterruptedException {
        //创建子线程
        MyThead2 t1=new MyThead2();
        //启动线程
        t1.start();

        for (int i = 1; i <=100; i++) {
            if (i==10) {
                //改变线程中断标记
                t1.flag=true;
```

```
                    Thread.sleep(2000);
                }
                System.out.println(Thread.currentThread().getName()+":"+i);
            }
        }
```

12.守护线程(后台线程,精灵线程):守护所有的非守护线程,当所有非守护线程全部死亡,那么　　守护线程无论有没有执行完都会死亡.

```
        前提:在线程启动之前设置线程是否为守护线程.
        语法:线程对象.setDaemon(boolean on)
        eg:public static void main(String[] args) {
        //创建一个子线程
        MyThead t1=new MyThead();
        //将子线程设置为守护线程
        t1.setDaemon(true);
        //启动线程
        t1.start();

        for (int i = 1; i <=100; i++) {
            System.out.println(Thread.currentThread().getName()+":"+i);
            //子线程把资源抢过去,还得运行一下
        }
    }
```

13.线程生命周期:

新建状态:刚New出来的线程对象就处于新建状态.

就绪状态:当线程调用start()或当阻塞状态的线程醒来后就处于就绪状态.

运行状态:当就绪状态的线程抢到资源运行run()时就处于运行状态.

阻塞状态:当线程调用sleep()或wait()时就处于阻塞状态.

死亡状态:当线程再也不执行了就处于死亡状态.

回顾:

1.作业.

2.随机流:能读能写的流,操作字节.(断点续传)

3.properties配置文件读写

4.jdbc结合流,向数据库的数据表中写入和读取文本文件和二进制文件.

5.装饰者模式.

6.进程和线程

7.实现线程有3种,前两种用得多,最后一种基本不用.

8.给线程取名4种