博客： https://www.cnblogs.com/HOsystem/p/14116443.html

2、具体内容

多线程的主要操作方法都在Thread类中定义了。

■线程的命名与取得

多线程的运行状态是不确定的，那么在程序的开发之中为了可以获取到一些需要使用到线程就只能依靠线程名字来进行操作。所以线程的名字是一个至关重要的概念，这样在Thread类之中就提供有线程名称的处理：

·**构造方法**：public Thread(Runnable target,String name);

·**设置名字**：public final void setName(String name);

·**取得名字**：public final String getName();

对于线程对象的获得不可能只是依靠一个this来完成的，因为线程的状态不可控，但是有一点是明确的，所有的线程对象一定要执行run()方法，那么这个时候可以考虑获取当前线程，在Thread类里面提供有获取当前线程的方法：

·获取当前线程：public static Thread currentThread();

范例：观察线程的命名操作

```
class MyThread implements Runnable {
	@Override
	public void run() {
		System.out.println(Thread.currentThread().getName());
	}
}
public class ThreadDemo {
	public static void main(String[] args) throws Exception {
		MyThread mt = new MyThread() ;
		new Thread(mt,"线程A").start() ;	// 设置了线程的名字
		new Thread(mt).start() ;		// 未设置线程名字
```

```
            new Thread(mt,"线程B").start() ;     // 设置了线程的名字
        }
}
```

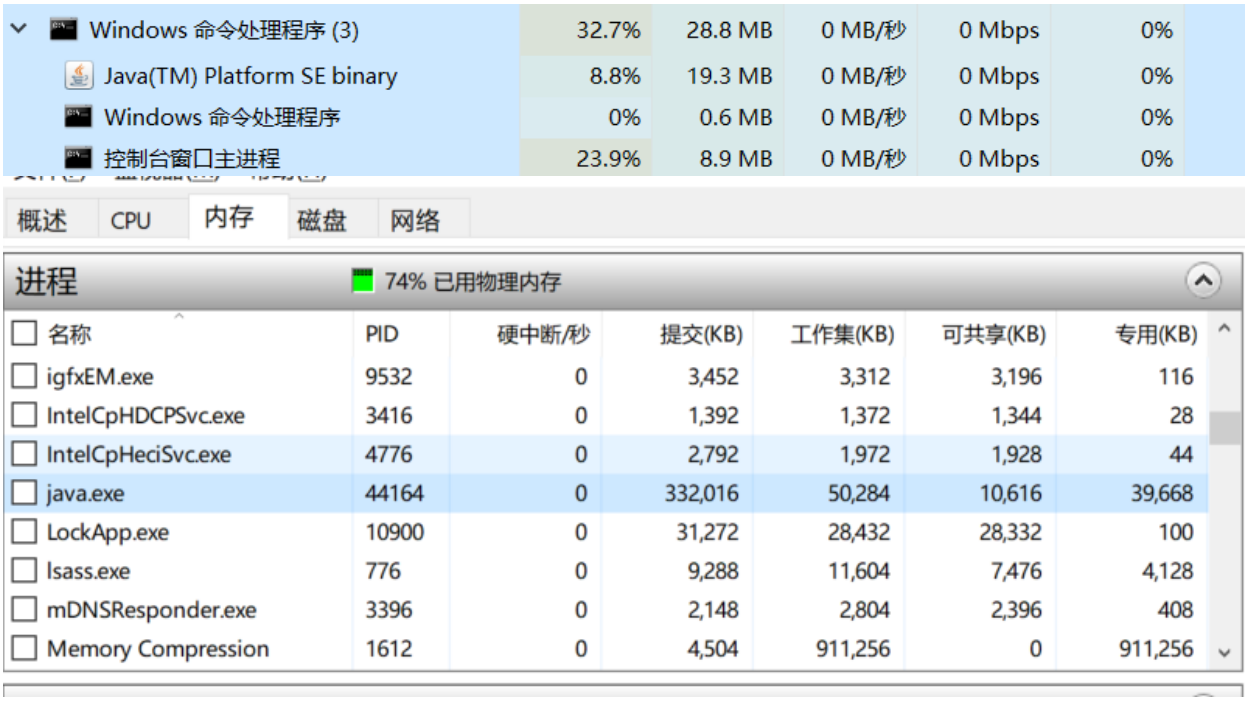
当开发者为线程设置名字的时候就使用设置的名字，而如果没有设置名字，则会自动生成一个不重复的名字，这种自动属性命名主要是依靠了static属性完成的，在Thread类里面定义有如下操作：

```
private static int threadInitNumber;
private static synchronized int nextThreadNum() {
        return threadInitNumber++;
    }
```

范例：观察一个程序

```
class MyThread implements Runnable {
        @Override
        public void run() {
                System.out.println(Thread.currentThread().getName());
        }
}
public class ThreadDemo {
        public static void main(String[] args) throws Exception {
                MyThread mt = new MyThread() ;
                new Thread(mt,"线程对象").start() ; // 设置了线程的名字
                mt.run(); // 对象直接调用run()方法
        }
}
```

通过此时的代码可以发现当使用了“mt.run()”直接在主方法之中调用线程类对象中的run()方法所获得的线程对象的名字为“main”，所以可以得出一个结论：主方法也是一个线程，那么现在的问题来了，所有的线程都是在进程上的划分，那么进程在哪里？每当使用java命令执行程序的时候就表示启动了一个JVM的进程，一台电脑上可以同时启动若干个JVM进程，所以每一个JVM进程都会有各自的线程。



在任何的开发之中，主线程可以创建若干个子线程，创建子线程的目的是可以将一些复杂逻辑或者比较耗时的逻辑交由子线程处理；

范例：子线程处理

```
package cn.mldn.demo;
public class ThreadDemo {
	public static void main(String[] args) throws Exception {
		System.out.println("1、执行操作任务一。");
		new Thread(()->{	// 子线程负责统计
			int temp = 0 ;
			for (int x = 0 ; x < Integer.MAX_VALUE ; x ++) {
				temp += x ;
			}
		}).start();
		System.out.println("2、执行操作任务二。");
		System.out.println("n、执行操作任务N。");
	}
}
```

主线程负责处理整体流程，而子线程负责处理耗时操作。

■线程的休眠

如果说现在希望某一个线程可以暂缓执行一次，那么就可以使用休眠的处理，在Thread类之中定义的休眠方法如下：

·休眠：public static void sleep(long millis)throws InterruptedException；

·休眠：public static void sleep(long millis,int nanos)throws InterruptedException；

在进行休眠的时候有可能会产生中断异常“InterruptedException”，中断异常属于Exception的子类，所以证明该异常必须进行处理。

范例：观察休眠处理

```
public class ThreadDemo {
	public static void main(String[] args) throws Exception {
		new Thread(()->{
			for (int x = 0 ; x < 10 ; x ++) {
				System.out.println(Thread.currentThread().getName() + "、x = " + x);
				try {
					Thread.sleep(1000);		// 暂缓执行
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		},"线程对象").start();
	}
}
```

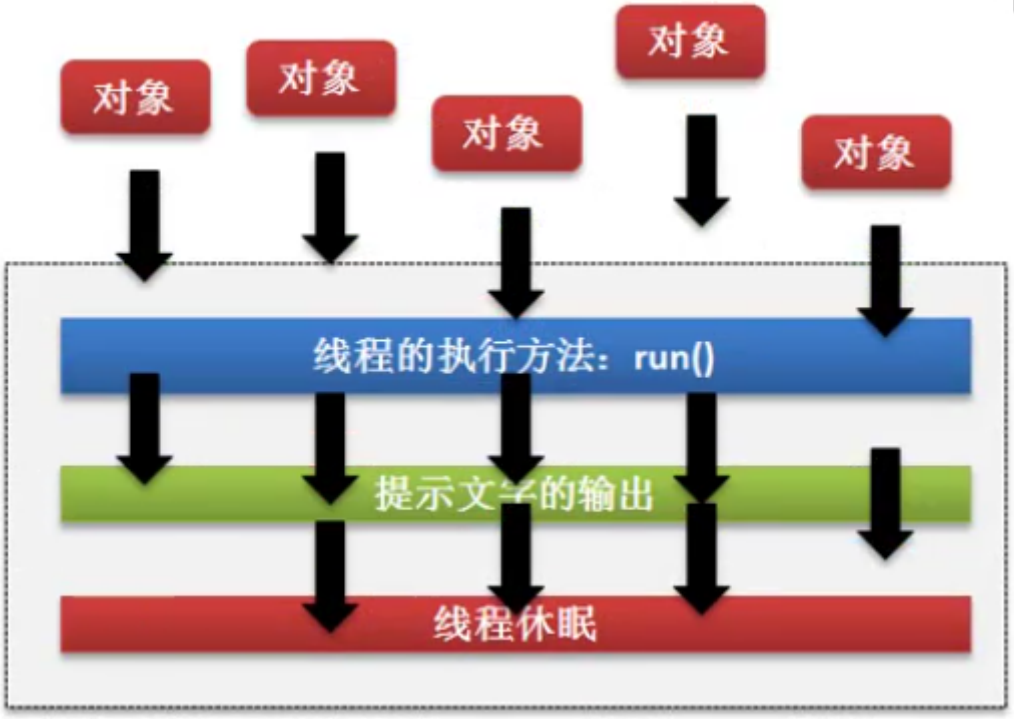
休眠的主要特点是可以自动实现线程的唤醒，以继续进行后续的处理。但是需要注意的是，如果现在有多个线程对象，那么休眠也是有先后顺序的。

范例：产生多个线程对象进行休眠处理

```
public class ThreadDemo {
	public static void main(String[] args) throws Exception {
		for (int num = 0; num < 5; num++) {
			new Thread(() -> {
				for (int x = 0; x < 10; x++) {
					System.out.println(Thread.currentThread().getName() + "、x = " +
x);
					try {
						Thread.sleep(1000); // 暂缓执行
					} catch (InterruptedException e) {
						e.printStackTrace();
					}
				}
			}, "线程对象 - " + num).start();
		}
	}
}
```

```
//两个代码效果是一致的
public class ThreadDemo {
	public static void main(String[] args) throws Exception {
		Runnable run = () -> {
			for (int x = 0; x < 10; x++) {
				System.out.println(Thread.currentThread().getName() + "、x = " + x);
				try {
					Thread.sleep(1000); // 暂缓执行
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		} ;
		for (int num = 0; num < 5; num++) {
			new Thread(run, "线程对象 - " + num).start();
		}
	}
}
```

此时将产生五个线程对象，并且这五个线程对象执行的方法体都是相同的。此时从程序执行的感觉上来讲好像是若干个线程一起进行了休眠，而后一起进行了自动唤醒，但是实际上是有差别的。



■线程中断

在之前发现线程的休眠里面提供有一个中断异常，实际上就证明线程的休眠是可以被打断的，而这种打断肯定是由其它线程完成的，在Thread类里面提供有这种中断执行的处理方法：

·判断线程是否被中断：public boolean isInterrupted();

·中断线程执行：public void interrupt();

范例：观察线程的中断处理操作

```
public class ThreadDemo {
    public static void main(String[] args) throws Exception {
        Thread thread = new Thread(()->{
            System.out.println("*** 72个小时的疯狂我需要睡觉补充精力。");
            try {
                Thread.sleep(10000);      // 预计准备休眠10秒
                System.out.println("*** 睡足了，可以出去继续祸害别人了。");
            } catch (InterruptedException e) {
                //e.printStackTrace();
                System.out.println("敢打扰我睡觉，老子宰了你。");
            }
        }) ;
        thread.start(); // 开始睡
        Thread.sleep(1000);
        if (!thread.isInterrupted()) {    // 该线程中断了吗?
            System.out.println("我偷偷的打扰一下你的睡眠。");
            thread.interrupt();  // 中断执行
```

```
            }
        }
}
```

所有正在执行的线程都是可以被中断的，中断线程必须进行异常的处理。

■线程的强制执行

所谓的线程的强制执行指的是当满足于某些条件之后，某一个线程对象将可以一直独占资源，一直到该线程的程序执行结束。

范例：观察一个没有强制执行的程序

```
public class ThreadDemo {
    public static void main(String[] args) throws Exception {
        Thread thread = new Thread(() -> {
            for (int x = 0 ; x < 100 ; x ++) {
                System.out.println(Thread.currentThread().getName() + "执行、x = " +
x);
            }
        },"玩耍的线程") ;
        thread.start();
        for (int x = 0 ; x < 100 ; x ++) {
            System.out.println("【霸道的main线程】number = " + x);
        }
    }
}
```

这个时候主线程和子线程都在交替执行着，但是如果说现在你希望主线程独占执行。那么就可以利用Thread类中的方法。

·强制执行：public final void join()throws InterruptedException；

```
public class ThreadDemo {
    public static void main(String[] args) throws Exception {
        Thread mainThread = Thread.currentThread() ; // 获得主线程
        Thread thread = new Thread(() -> {
            for (int x = 0 ; x < 100 ; x ++) {
                if (x == 3) {     // 现在霸道的线程来了
                    try {
                        mainThread.join();// 霸道线程要先执行
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println(Thread.currentThread().getName() + "执行、x = " +
x);
            }
```

```
            },"玩耍的线程") ;
            thread.start();
            for (int x = 0 ; x < 100 ; x ++) {
                Thread.sleep(100);
                System.out.println("【霸道的main线程】number = " + x);
            }
        }
}
```

在进行线程强制执行的时候一定要获取强制执行线程对象之后才可以执行join的()调用。

■线程的礼让

线程的礼让指的是先将资源让出去让别的线程先执行。线程的礼让可以使用Thread中提供的方法：

·礼让：public static void yield()；

范例：使用礼让操作

```
public class ThreadDemo {
    public static void main(String[] args) throws Exception {
        Thread thread = new Thread(() -> {
            for (int x = 0 ; x < 100 ; x ++) {
                if (x % 3 == 0) {
                    Thread.yield();  // 线程礼让
                    System.out.println("### 玩耍的线程礼让执行 ###");
                }
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println(Thread.currentThread().getName() + "执行、x = " +
x);
            }
        },"玩耍的线程") ;
        thread.start();
        for (int x = 0 ; x < 100 ; x ++) {
            Thread.sleep(100);
            System.out.println("【霸道的main线程】number = " + x);
        }
    }
}
```

礼让执行的时候每一次调用yield()方法都只会礼让一次当前的资源。

■线程优先级

从理论上来讲线程的优先级越高越有可能先执行（越有可能先抢占到资源）。在Thread类里面针对于优先级的操作提供有如下的两个处理方法：

·设置优先级：public final void setPriority(int newPriority)；

·获取优先级：public final int getPriority()；

在进行优先级定义的时候都是通过int型的数字来完成的，而对于此数字的选择在Thread类里面就定义有三个常量：

·最高优先级：public static final int MAX_PRIORITY、10；

·中等优先级：public static final int NORM_PRIORITY、5；

·最低优先级：public static final int MIN_PRIORITY、1；

范例：观察优先级

```
public class ThreadDemo {
	public static void main(String[] args) throws Exception {
		Runnable run = ()->{
			for (int x = 0 ; x < 10 ; x ++) {
				try {
					Thread.sleep(1000);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
				System.out.println(Thread.currentThread().getName() + "执行。");
			}
		} ;
		Thread threadA = new Thread(run,"线程对象A") ;
		Thread threadB = new Thread(run,"线程对象B") ;
		Thread threadC = new Thread(run,"线程对象C") ;
		threadA.setPriority(Thread.MIN_PRIORITY);
		threadB.setPriority(Thread.MIN_PRIORITY);
		threadC.setPriority(Thread.MAX_PRIORITY);
		threadA.start();
		threadB.start();
		threadC.start();
	}
}
```

主方法是一个主线程，那么主线程的优先级呢？

范例：获取优先级

```
public class ThreadDemo {
	public static void main(String[] args) throws Exception {
		System.out.println(new Thread().getPriority());
		System.out.println(Thread.currentThread().getPriority());
	}
}
```

主线程是属于中等优先级，而默认创建的线程也是中等优先级。