16 线程 常用方法

JAVAEE高级

一：Thread类的有关方法

> 1、void start()
> 启动线程，并执行对象的run()方法
> 2、run()
> 线程在被调度时执行的操作
> 3、String getName()
> 返回线程的名称
> 4、void setName(String name)
> 设置该线程名称
> 5、static currentThread()
> 返回当前线程

1：线程设置名称

> 1、通过getName()/setName(String)方法获取(设置)线程对象的名字

```
@Test
    public void getVoid() {
        Thread t1 = new Thread() {
            public void run() {
                this.setName("张三");
                System.out.println(this.getName() + "AAAA");
            }
        };

        Thread t2 = new Thread() {
            public void run() {
                this.setName("李四");
                System.out.println(this.getName() + "BBBB");
```

```
                }
            };
            /*
             * t1.setName("张三"); t2.setName("李四");
             */
            t1.start();
            t2.start();
        }
```

> 2、通过构造方法进行设置名称

```
    @Test
public void getVoid() {
        new Thread("小陈") {
            public void run() {
                System.out.println(this.getName() + "AAAAA");
            }
        }.start();
    }
```

2：获取当前线程的名称Thread.currentThread()

```
@Test
    public void getVoid() {
        new Thread() {
            public void run() {
                System.out.println(getName() + "AAA");
            }
        }.start();


        new Thread(new Runnable() {
            public void run() {
                // Thread.currentThread()获取当前正在执行的线程
                System.out.println(Thread.currentThread().getName() + "
BBB");
            }
        }).start();

        Thread.currentThread().setName("我是主线程");
        System.out.println(Thread.currentThread().getName());
```

```
    }
```

3、线程休眠sleep(毫秒,纳秒)

> 1、Thread.sleep(毫秒,纳秒), 控制当前线程休眠若干毫秒1秒= 1000毫秒

```
public static void main(String[] args) {
        new Thread() {
            public void run() {
                for (int i = 0; i < 10; i++) {
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    System.out.println(getName() + "...线程1");
                }
            }
        }.start();
    }
```

4、守护线程setDaemon()

> 1、 setDaemon()、设置一个线程为守护线程,
>
> 2、该线程不会单独执行、 当其他非守护线程都执行结束后、 自动退出

```
public static void main(String[] args){
        Thread t1 = new Thread() {
            public void run() {
                for(int i = 0; i < 2; i++) {
                    System.out.println(getName() + "aaaa");
                }
            }
        };  t1.start();

        Thread t2 = new Thread() {
            public void run() {
                for(int i = 0; i <100; i++) {
```

```
                System.out.println(getName() + "bbbb");
            }
        }
    };
    t2.setDaemon(true);//设置为守护线程
    t2.start();
}
```

5、加入线程join()

> 1、join() ：
> 当某个程序执行流中调用其他线程的 join() 方法时，调用线程将被阻塞，等待指定的 join 线程执行完为止 ，当前线程再继续
>
> 2、join(int)：
> 可以等待指定的毫秒之后继续

```
public static void main(String[] args) {
    Thread t1 = new Thread() {
        public void run() {
            for (int i = 0; i < 10; i++) {
                System.out.println(getName() + "AAAA");
            }
        }
    };

    Thread t2 = new Thread() {
        public void run() {
            for (int i = 0; i < 10; i++) {
                if (i == 2) {
                    try {
                        // t1.join();
                        t1.join(); // 插队指定的时间,过了指定时间后,两条线程交替执行
                    } catch (InterruptedException e) {

                        e.printStackTrace();
                    }
                }
                System.out.println(getName() + "BBBB");
            }
```

```
                }
            };

            t1.start();
            t2.start();
        }
```

6、礼让线程yield()

> 1、在线程执行过程中让出CPU

```
class  MyThread extends   Thread{
    @Override
    public void run() {
       for (int i = 0; i < 12; i++) {
         if(i % 10==0){
             Thread.yield();
             System.out.println("礼让"+getName()+":"+i);
         }
         System.out.println(getName()+":"+i);
    }
    }
}
```

```
    public static void main(String[] args) {
        new MyThread().start();
        new MyThread().start();
    }
```

7、设置线程的优先级

> 1、setPriority()设置线程的优先级

```
public static void main(String[] args) {
        Thread t1 = new Thread() {
            public void run() {
                for (int i = 0; i < 30; i++) {
                    System.out.println(getName() + ":AAAA");
```

```
                }
            }
        };

        Thread t2 = new Thread() {
            public void run() {
                for (int i = 0; i < 30; i++) {
                    System.out.println(getName() + ":BB");
                }
            }
        };
         //范围大小最小1最大10默认5
        // t1.setPriority(1);
        // t2.setPriority(10);设置最大优先级
        t1.setPriority(Thread.MIN_PRIORITY); // 设置最小的线程优先级
        t2.setPriority(Thread.MAX_PRIORITY); // 设置最大的线程优先级
        t1.start();
        t2.start();
    }
```

线程优先级源码描述

TestJava.java | Thread.class

```
            synchronized (blockerLock) {
                blocker = b;
            }
        }

        /**
         * The minimum priority that a thread can have.
         */
        public final static int MIN_PRIORITY = 1;

       /**
         * The default priority that is assigned to a thread.
         */
        public final static int NORM_PRIORITY = 5;

        /**
         * The maximum priority that a thread can have.
         */
        public final static int MAX_PRIORITY = 10;

        /**
```