链滴

Thread 源码详解

作者：roll

原文链接：https://ld246.com/article/1515723585516

来源网站：链滴

package java.lang;
/**thread 是程序中的一个执行程序，java 虚拟机允许一个应用有多个线程同时运行

A thread is a thread of execution in a program. The Java

Virtual Machine allows an application to have multiple threads of

execution running concurrently.

每个线程都有优先级，更高优先级的线程比低优先级优先运行。
每个线程都有可能被标记为守护线程 daemon。当代码运行时，一些线程差 UN 构建一个新的 Tread 对象，新的 thread 优先级初始设置等于创建该线程的优先级，当且精当创建线程的线程是一个护线程时候，才会创建一个守护线程（什么是守护线程：http://blog.csdn.net/shimiso/article/details/8964414）
只要还有任何非守护线程在运行，那么这个 Java 程序也在继续运行。当该程序中所有的非守护程都终止时，虚拟机实例将自动退出。

Every thread has a priority. Threads with higher priority are

executed in preference to threads with lower priority. Each thread

may or may not also be marked as a daemon. When code running in

some thread creates a new Thread object, the new

thread has its priority initially set equal to the priority of the

creating thread, and is a daemon thread if and only if the

creating thread is a daemon.

当 java 虚拟机启动的时候，有一个单非守护线程（通常被称作 main 方法）
java 虚拟机会一直运行这个线程知道下面一些情况发生：
1Runntime 方法的 exit 被调用并且安全管理通过了 exit 操作的发生
2 所有的非守护线程都死了，另外被调用的 fun 方法抛出了异常

When a Java Virtual Machine starts up, there is usually a single

non-daemon thread (which typically calls the method named

main of some designated class). The Java Virtual

Machine continues to execute threads until either of the following

occurs:

The exit method of class Runtime has been

*   called and the security manager has permitted the exit ope
ation
*   to take place.

All threads that are not daemon threads have died, either by

*   returning from the call to the run method or by
*   throwing an exception that propagates beyond the run</
>
*   method.

有两个新建执行线程的方法，第一个被申明为继承 Thread 方法，基类必须覆盖 Thread 的 run
法，
那么基类的一个示例就能被分配和开始线程了，例如 extends Thraed

There are two ways to create a new thread of execution. One is to

declare a class to be a subclass of Thread. This

subclass should override the run method of class

Thread. An instance of the subclass can then be

allocated and started. For example, a thread that computes primes

larger than a stated value could be written as follows:

*   class PrimeThread extends Thread {
*   long minPrime;
*   PrimeThread(long minPrime)

*
his.minPrime = minPrime;
*   }

*   public void run() {
*   /
 compute primes larger than minPrime
*
nbsp;. . .
*   }
*   }

The following code would then create a thread and start it running:

*   PrimeThread p = new PrimeThread(143);
*   p.start();

另一个方法是新建一个线程实现 Runable 接口，类去实现 run 方法，这个类的实例能被分配，
过作为参数传递到 THread 里面，并启动线程。

The other way to create a thread is to declare a class that

implements the Runnable interface. That class then

implements the run method. An instance of the class can

then be allocated, passed as an argument when creating

Thread, and started. The same example in this other

style looks like the following:

*   class PrimeRun implements Runnable {
*   long minPrime;
*   PrimeRun(long minPrime) {</
>
*
his.minPrime = minPrime;
*   }

*   public void run() {
*   /
 compute primes larger than minPrime
*
nbsp;. . .
*   }
*   }

The following code would then create a thread and start it running:

*   PrimeRun p = new PrimeRun(143);
*   new Thread(p).start();

每一个线程都有一个名字为了标记，不止一个线程拥有相同的名字，如果名字没有被指定的时候
 UN 构建了一个线程，那么会默认给一个新的名字。

Every thread has a name for identification purposes. More than

one thread may have the same name. If a name is not specified when

a thread is created, a new name is generated for it.

Unless otherwise noted, passing a {@<a href="https://ld246.com/member/code" aria-n
me="code" class="tooltipped__user" target="_blank">code null} argument to a construc
or

or method in this class will cause a {@<a href="https://ld246.com/member/link" aria-na
e="link" class="tooltipped__user" target="_blank">link NullPointerException} to be

thrown.

@author unascribed

@see   Runnable

@see   Runtime#exit(int)

@see   #run()

@see   #stop()

@since   JDK1.0

*/
public
class Thread implements Runnable {
类加载的时候，调用本地的注册本地方静态方法， 这个方法是本地方法
    /* Make sure registerNatives is the first thing does. */
    private static native void registerNatives();
    static {
        registerNatives();
    }
    private volatile char name[];
    private int   &nbs
;   priority;//优先级
    private Thread
nbsp; threadQ;
    private long   &n
sp;   eetop;
    /* Whether or not to single_step this thread. */
    private boolean   single_step;
    是否是守护线程
    /* Whether or not the thread is a daemon thread. */
    private boolean   daemon = false;
    /* JVM state */
    private boolean   stillborn = false;
要执行的 run 方法的对象
    /* What will be run. */
    private Runnable target;
这个线程的线程组
    /* The group of this thread */
    private ThreadGroup group;
这个线程的上下文类加载器
    /* The context ClassLoader for this thread */

    private ClassLoader contextClassLoader;
此线程继承的 AccessCOntrollerContext
上下文环境，指的是：继承父类的一些属性，比如优先级，是否是守护线程
    /* The inherited AccessControlContext of this thread */
    private AccessControlContext inheritedAccessControlContext;
对没有命名的线程自动编号
    /* For autonumbering anonymous threads. */
    private static int threadInitNumber;
    private static synchronized int nextThreadNum() {
       return threadInitNumber++;
    }
    /* ThreadLocal values pertaining to this thread. This map is maintaned
     * by the ThreadLocal class. */
    ThreadLocal.ThreadLocalMap threadLocals = null;
    /*
     * InheritableThreadLocal values pertaining to this thread. Ths map is
     * maintained by the InheritableThreadLocal class.
     */
    ThreadLocal.ThreadLocalMap inheritableThreadLocals = null;
    /*
    栈大小，默认是 0，具体分配由 jvm 决定
     * The requested stack size for this thread, or 0 if the creator did
     * not specify a stack size. It is up to the VM to do whatever it
     * likes with this number; some VMs will ignore it.
     */
    private long stackSize;
    /*
    本地线程终止后任然存在的 JVM 私有状态
     * JVM-private state that persists after native thread termination.
     */
    private long nativeParkEventPointer;
    /*
    线程 ID
     * Thread ID
     */
    private long tid;
    产生 ThreadID
    /* For generating thread ID */
    private static long threadSeqNumber;
    /* Java thread status for tools,
     * initialized to indicate thread 'not yet started'
     */
    private volatile int threadStatus = 0;
    private static synchronized long nextThreadID() {
        return ++threadSeqNumber;
    }
    /**
     * The argument supplied to the current call to
     * java.util.concurrent.locks.LockSupport.park.

     * Set by (private) java.util.concurrent.locks.LockSupport.setBocker
     * Accessed using java.util.concurrent.locks.LockSupport.getBocker
     */
    volatile Object parkBlocker;
    /* The object in which this thread is blocked in an interruptible I/O
     * operation, if any. The blocker's interrupt method should be invoked
     * after setting this thread's interrupt status.
     */
    private volatile Interruptible blocker;
    private final Object blockerLock = new Object();
    /* Set the blocker field; invoked via sun.misc.SharedSecrets from java.nio code
     */
    void blockedOn(Interruptible b) {
        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;
    /**
     * Returns a reference to the currently executing thread object.
     *返回一个实现当前在执行的线程
     * @return the currently executing thread.
     */
    public static native Thread currentThread();
    /**
    提示当前调度器愿意去 yield 目前使用的处理器。调度程序可以自由忽视这个示
     * A hint to the scheduler that the current thread is willing to yield
     * its current use of a processor. The scheduler is free to ignre this
     * hint.

   yield 是试图去改善相对的尝试去过度使用 cpu，它的使用应该与详细分析和肌醇结合起来，确保它实际上具有积极的效果

     * Yield is a heuristic attempt to improve relative progressio

     * between threads that would otherwise over-utilise a CPU.
ts use
     * should be combined with detailed profiling and benchma
king to
     * ensure that it actually has the desired effect.
    这个方法的使用场景很少，最可能去用于调试和测试目的，这个可以帮助去重现
bug 由于竞争资源，他也可以当设计并发结构的时候
     * It is rarely 难的 appropriate 合适的 to use this method. It
ay be useful
     * for debugging or testing purposes, where it may help to r
produce
     * bugs due to race conditions. It may also be useful when d
signing
     * concurrency control constructs such as the ones in the</
>
     * {@<a href="https://ld246.com/member/link" aria-name="
ink" class="tooltipped__user" target="_blank">link java.util.concurrent.locks} package.<
p>
     */
    public static native void yield();
    /**
    让当前的在运行的线程 sleep（暂定停止执行） 一定的时间（毫秒），以系统
当前为准，线程不会丢失任何监控的所有权
     * Causes the currently executing thread to sleep (temporaril
 cease
     * execution) for the specified number of milliseconds, subje
t to
     * the precision and accuracy of system timers and schedule
s. The thread
     * does not lose ownership of any monitors.
     *
     * @param millis
     *
he length of time to sleep in milliseconds
     *
     * @throws IllegalArgumentException
     *
nbsp; if the value of {@<a href="https://ld246.com/member/code" aria-name="code" class="
ooltipped__user" target="_blank">code millis} is negative
     *
     * @throws InterruptedException
     *
nbsp; if any thread has interrupted the current thread. The
     *
nbsp; interrupted status of the current thread is
     *
nbsp; cleared when this exception is thrown.
     */
    public static native void sleep(long millis) throws InterruptedExcept
on;
    /**
    导致当前正在执行的线程休眠，毫秒加上指定的纳秒数

     * Causes the currently executing thread to sleep (temporaril
cease
     * execution) for the specified number of milliseconds plus t
e specified
     * number of nanoseconds, subject to the precision and accu
acy of system
     * timers and schedulers. The thread does not lose ownershi
of any
     * monitors.
     *
     * @param millis
     *
he length of time to sleep in milliseconds
     *
     * @param nanos
     *
@<a href="https://ld246.com/member/code" aria-name="code" class="tooltipped__user" tar
et="_blank">code 0-999999} additional nanoseconds to sleep
     *
     * @throws IllegalArgumentException
     *
nbsp; if the value of {@<a href="https://ld246.com/member/code" aria-name="code" class="
ooltipped__user" target="_blank">code millis} is negative, or the value of
     *
nbsp; {@<a href="https://ld246.com/member/code" aria-name="code" class="tooltipped__u
er" target="_blank">code nanos} is not in the range {@<a href="https://ld246.com/me
ber/code" aria-name="code" class="tooltipped__user" target="_blank">code 0-999999}
/p>
     *
     * @throws InterruptedException
     *
nbsp; if any thread has interrupted the current thread. The
     *
nbsp; interrupted status of the current thread is
     *
nbsp; cleared when this exception is thrown.
     */
    public static void sleep(long millis, int nanos)
    throws InterruptedException {
        if (millis < 0) {
            throw n
w IllegalArgumentException("timeout value is negative");
        }
        if (nanos < 0 || nanos > 999999)

            throw n
w IllegalArgumentException(
           &
bsp;   &
bsp;   "nanosecond timeout value out of range");
        }
        if (nanos >= 500000 || (nanos != 0
&& millis == 0)) {
            millis++

       }
        sleep(millis);
    }
    /**
    初始化一个线程带有当前上下文

     * Initializes a Thread with the current AccessControlContext

     * @see #init(ThreadGroup,Runnable,String,long,AccessCont
olContext)
     */
    private void init(ThreadGroup g, Runnable target, String name,</p

          &
bsp;   long stackSize) {
        init(g, target, name, stackSize, null);</
>
    }
    /**
     * Initializes a Thread.
     *初始化一个线程 初始化的基本方法
ThreadGroup g：ThreadGroup 线程组 线程组是一个可以统一管理的线程集合。默认情况下，
没有指定线程组的话，创建的所有线程属于相同的线程组。先将 secuity 的线程组赋给新线程的线程组
如果还是为空，则设为父线程的线程组。
     * @param g the Thread group
Runnable target：Runnable 对象，即装配线程要执行的 run()方法，this.target = target;</p

     * @param target the object whose run() method gets calle

线程名字
     * @param name the name of the new Thread
栈大小
     * @param stackSize the desired stack size for the new thre
d, or
     *   zero to
ndicate that this parameter is to be ignored.
     * @param acc the AccessControlContext to inherit, or
     *   &
nbsp;   AccessController.getContext() if null
     */
    private void init(ThreadGroup g, Runnable target, String name,</p

          &
bsp;   long stackSize, AccessControl
ontext acc) {
        if (name == null) {
           throw n
w NullPointerException("name cannot be null");
        }
        this.name = name.toCharArray();
        Thread parent = currentThread();
        SecurityManager security = System.ge
SecurityManager();

        if (g == null) {
            /* Deter
ine if it's an applet or not */
            /* If ther
is a security manager, ask the security manager
           &
bsp; what to do. */
            if (securi
y != null) {
           &
bsp;   g = security.getThreadGroup();
            }
            /* If the
ecurity doesn't have a strong opinion of the matter
           &
bsp; use the parent thread group. */
            if (g ==
ull) {
           &
bsp;   g = parent.getThreadGroup();
            }
        }
        /* checkAccess regardless of whether
or not threadgroup is
           explicitly passed
in. */
        g.checkAccess();
        /*
         * Do we have the required pe
missions?
         */
        if (security != null) {
            if (isCCL
verridden(getClass())) {
           &
bsp;   security.checkPermission(SUBCLASS_IMPLEMENTATION_PERMISSION);<
>
            }
         }
         g.addUnstarted();
         this.group = g;
         this.daemon = parent.isDaemon();<
>
         this.priority = parent.getPriority();</p

         if (security == null || isCCLOverridden
parent.getClass()))
             this.cont
xtClassLoader = parent.getContextClassLoader();
         else
             this.cont
xtClassLoader = parent.contextClassLoader;
         this.inheritedAccessControlContext =
/p>

               acc != null ? acc : AccessController.getContext();
        this.target = target;
        setPriority(priority);
        if (parent.inheritableThreadLocals != ull)
             this.inheritableThreadLocals =
               ThreadLocal.createInheritedMap(parent.inheritableThreadLocals);
         /* Stash the specified stack size in cas the VM cares */
         this.stackSize = stackSize;
         /* Set thread ID */
         tid = nextThreadID();
     }
     /**
     抛出克隆不支持的异常并且新建一个新的线程来代替
      * Throws CloneNotSupportedException as a Thread can not be meaningfully
      * cloned. Construct a new Thread instead.
      *
      * @throws CloneNotSupportedException
      *   always
      */
     @Override
     protected Object clone() throws CloneNotSupportedException {
         throw new CloneNotSupportedException();
     }
     /**
分配一个新的线程对象，这个构造函数
      * Allocates a new {@code Thread} object. This constructor has the same
      * effect as {@linkplain #Thread(ThreadGroup,Runnable,String) Thread}
      * {@code (null, null, gname)}, where {@code gname} is a newly generated
      * name. Automatically generated names are of the form
      * {@code "Thread-"+}n, where n is an integer.
      */
     public Thread() {
         init(null, null, "Thread-" + nextThreadNum(), 0);
     }
     /**

     * Allocates a new {@code Thread} object. This constructor has the same
     * effect as {@linkplain #Thread(ThreadGroup,Runnable,String) Thread}
     * {@code (null, target, gname)}, where {@a href="https://ld246.com/member/code" aria-name="code" class="tooltipped__user" target "_blank">code gname} is a newly generated
     * name. Automatically generated names are of the form</>
     * {@code "Thread-"+}n, where n is an integer.
     *
     * @param target
     *   he object whose {@code run} method is invoked when this thread
     *   s started. If {@code null}, this classes {@code run} method does
     *   othing.
     */
    public Thread(Runnable target) {
        init(null, target, "Thread-" + nextThreadNum(), 0);
    }
    /**
     * Creates a new Thread that inherits the given AccessControContext.
     * This is not a public constructor.
     */
    Thread(Runnable target, AccessControlContext acc) {
        init(null, target, "Thread-" + nextThreadNum(), 0, acc);
    }
    /**
     * Allocates a new {@code Thread} object. This constructor has the same
     * effect as {@linkplain #Thread(ThreadGroup,Runnable,String) Thread}
     * {@code (group, target, gname)} ,where {code gname} is a newly generated
     * name. Automatically generated names are of the form</>
     * {@code "Thread-"+}n, where n is an inte

er.
     *
     * @param group
     *   he thread group. If {@code null} and there is a security
     *   manager, the group is determined by {@linkplain
     *   ecurityManager#getThreadGroup SecurityManager.getThreadGroup()}.
     *   f there is not a security manager or {@code
     *   ecurityManager.getThreadGroup()} returns {@code null}, the group
     *   s set to the current thread's thread group.
     *
     * @param target
     *   he object whose {@code run} method is invoked when this thread
     *   s started. If {@code null}, this thread's run method is invoked.
     *
     * @throws SecurityException
     *   nbsp; if the current thread cannot create a thread in the specified
     *   nbsp; thread group
     */
    public Thread(ThreadGroup group, Runnable target) {
        init(group, target, "Thread-" + nextTh eadNum(), 0);
    }
    /**
    新建一个线程指定名字
     * Allocates a new {@code Thread} object. T is constructor has the same
     * effect as {@linkplain #Thread(ThreadGroup,Runnable,Stri g) Thread}
     * {@code (null, null, name)}.
     *
     * @param   name
     *   nbsp; the name of the new thread
     */
    public Thread(String name) {
        init(null, null, name, 0);
    }

    /**
    * Allocates a new {@code Thread} object. This constructor has the same
    * effect as {@linkplain #Thread(ThreadGroup,Runnable,String) Thread}
    * {@code (group, null, name)}.
    *
    * @param group
    *   he thread group. If {@code null} and there is a security
    *   manager, the group is determined by {@linkplain
    *   ecurityManager#getThreadGroup SecurityManager.getThreadGroup()}.
    *   f there is not a security manager or {@code
    *   ecurityManager.getThreadGroup()} returns {@code null}, the group
    *   s set to the current thread's thread group.
    *
    * @param name
    *   he name of the new thread
    *
    * @throws SecurityException
    *   nbsp; if the current thread cannot create a thread in the specified
    *   nbsp; thread group
    */
    public Thread(ThreadGroup group, String name) {
        init(group, null, name, 0);
    }
    /**
     * Allocates a new {@code Thread} object. This constructor has the same
     * effect as {@linkplain #Thread(ThreadGroup,Runnable,String) Thread}
     * {@code (null, target, name)}.
     *
     * @param target
     *   he object whose {@code run} method is invoked when this thread
     *   s started. If {@<a href="https://ld246.com/member/code" aria-name="code" class="tooltipp

d__user" target="_blank">code null}, this thread's run method is invoked.
     *
     * @param name
     *   he name of the new thread
     */
    public Thread(Runnable target, String name) {
        init(null, target, name, 0);
    }
    /**
     * Allocates a new {@code Thread} object so that it has {@code target}
     * as its run object, has the specified {@code name} as its name,
     * and belongs to the thread group referred to by {@code group}.
     *
     * If there is a security manager, its
     * {@link SecurityManager#checkAccess(ThreaGroup) checkAccess}
     * method is invoked with the ThreadGroup as its argument
     *
     * In addition, its {@code checkPermission} method is invoked with
     * the {@code RuntimePermission("enableontextClassLoaderOverride")}
     * permission when invoked directly or indirectly by the consructor
     * of a subclass which overrides the {@code getContextClassLoader}
     * or {@code setContextClassLoader} methds.
     *
     * The priority of the newly created thread is set equal to th
     * priority of the thread creating it, that is, the currently runnng
     * thread. The method {@linkplain #setPriority setPriority} my be
     * used to change the priority to a new value.
     *
     * The newly created thread is initially marked as being a damon
     * thread if and only if the thread creating it is currently mar

ed
     * as a daemon thread. The method {@linkplain #setDaemon setDaemon}
     * may be used to change whether or not a thread is a daemon.
     *
     * @param group
     *   he thread group. If {@code null} and there is a security
     *   manager, the group is determined by {@linkplain
     *   ecurityManager#getThreadGroup SecurityManager.getThreadGroup()}.
     *   f there is not a security manager or {@code
     *   ecurityManager.getThreadGroup()} returns {@code null}, the group
     *   s set to the current thread's thread group.
     *
     * @param target
     *   he object whose {@code run} method is invoked when this thread
     *   s started. If {@code null}, this thread's run method is invoked.
     *
     * @param name
     *   he name of the new thread
     *
     * @throws SecurityException
     *   nbsp; if the current thread cannot create a thread in the specified
     *   nbsp; thread group or cannot override the context class loader methods.
     */
    public Thread(ThreadGroup group, Runnable target, String name)
        init(group, target, name, 0);
    }
    /**
     * Allocates a new {@code Thread} object so that it has {@code target}
     * as its run object, has the specified {@code name} as its name,
     * and belongs to the thread group referred to by {@<a hre

="https://ld246.com/member/code" aria-name="code" class="tooltipped__user" target="_bl
nk">code group}, and has
     * the specified stack size.
     *
     * This constructor is identical to {@<a href="https://ld246.c
m/member/link" aria-name="link" class="tooltipped__user" target="_blank">link
     * #Thread(ThreadGroup,Runnable,String)} with the exceptio
of the fact
     * that it allows the thread stack size to be specified.
he stack size
     * is the approximate number of bytes of address space that
he virtual
     * machine is to allocate for this thread's stack. The ef
ect of the
     * {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code stackSize} parameter, if any, is hi
hly platform dependent.
     *
     * On some platforms, specifying a higher value for the
     * {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code stackSize} parameter may allow a
thread to achieve greater
     * recursion depth before throwing a {@<a href="https://ld
46.com/member/link" aria-name="link" class="tooltipped__user" target="_blank">link S
ackOverflowError}.
     * Similarly, specifying a lower value may allow a greater nu
ber of
     * threads to exist concurrently without throwing an {@<a h
ef="https://ld246.com/member/link" aria-name="link" class="tooltipped__user" target="_bla
k">link
     * OutOfMemoryError} (or other internal error). The d
tails of
     * the relationship between the value of the stackSize param
ter
     * and the maximum recursion depth and concurrency level
re
     * platform-dependent. On some platforms, the value
of the
     * {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code stackSize} parameter may have n
effect whatsoever.
     *
     * The virtual machine is free to treat the {@<a href="https:/
ld246.com/member/code" aria-name="code" class="tooltipped__user" target="_blank">cod
 stackSize}
     * parameter as a suggestion. If the specified value is
unreasonably low
     * for the platform, the virtual machine may instead use som

     * platform-specific minimum value; if the specified value is
nreasonably
     * high, the virtual machine may instead use some platform-
pecific
     * maximum. Likewise, the virtual machine is free to r

und the specified
    * value up or down as it sees fit (or to ignore it completely)

    *
    * Specifying a value of zero for the {@<a href="https://ld24
.com/member/code" aria-name="code" class="tooltipped__user" target="_blank">code
tackSize} parameter will
    * cause this constructor to behave exactly like the
    * {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code Thread(ThreadGroup, Runnable,
tring)} constructor.
    *
    * Due to the platform-dependent nature of the behavior of
his
    * constructor, extreme care should be exercised in its use.<
p>
    * The thread stack size necessary to perform a given compu
ation will
    * likely vary from one JRE implementation to another.
In light of this
    * variation, careful tuning of the stack size parameter may b
required,
    * and the tuning may need to be repeated for each JRE imp
ementation on
    * which an application is to run.
    *
    * Implementation note: Java platform implementers are en
ouraged to
    * document their implementation's behavior with respect to
the
    * {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code stackSize} parameter.
    *
    *
    * @param group
    *
he thread group. If {@<a href="https://ld246.com/member/code" aria-name="code" class="t
oltipped__user" target="_blank">code null} and there is a security
    *
manager, the group is determined by {@linkplain
    *
ecurityManager#getThreadGroup SecurityManager.getThreadGroup()}.
    *
f there is not a security manager or {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code
    *
ecurityManager.getThreadGroup()} returns {@<a href="https://ld246.com/member/code" aria
name="code" class="tooltipped__user" target="_blank">code null}, the group
    *
s set to the current thread's thread group.
    *
    * @param target
    *
he object whose {@<a href="https://ld246.com/member/code" aria-name="code" class="too

tipped__user" target="_blank">code run} method is invoked when this thread
     *
s started. If {@<a href="https://ld246.com/member/code" aria-name="code" class="tooltipp
d__user" target="_blank">code null}, this thread's run method is invoked.
     *
     * @param name
     *
he name of the new thread
     *
     * @param stackSize
     *
he desired stack size for the new thread, or zero to indicate
     *
hat this parameter is to be ignored.
     *
     * @throws SecurityException
     *
nbsp; if the current thread cannot create a thread in the specified
     *
nbsp; thread group
     *
     * @since 1.4
     */
    public Thread(ThreadGroup group, Runnable target, String name,<
p>
            &
bsp;   long stackSize) {
        init(group, target, name, stackSize);</
>
    }
    /**
    导致线程去开始执行，java 虚拟机调用这个线程的 run 方法
     * Causes this thread to begin execution; the Java Virtual Ma
hine
     * calls the run method of this thread.
     *
    结果是两个线程同时运行，当前的线程（start 方法调用的的）和另一个（run
法）
     * The result is that two threads are running concurrently: th

     * current thread (which returns from the call to the
     * start method) and the other thread (which executes its</
>
     * run method).
     *

     * It is never legal to start a thread more than once.
     * In particular, a thread may not be restarted once it has c
mpleted
     * execution.
     *一个线程启动多次是不合法的，通常来说，一个线程不能被重
一旦他完成了执行
     * @exception IllegalThreadStateException if
he thread was already

     *   nbsp;   started.
     * @see   #un()
     * @see   #top()
     */
    public synchronized void start() {
        /**
    这个方法不是被 main 方法线程或者系统线程组通过 VM 创建，那就不会被 invke。每一个新的线程组在未来都会被加入 vm 中，threadStatus 等于 0 表示还未执行的线程，新建。
         * This method is not invoked for the main method thread or "system"
         * group threads created/set up by the VM. Any new functionality added
         * to this method in the future may have to also be added to the VM.
         *
         * A zero status value corresponds to state "NEW".
         */
        if (threadStatus != 0)
            throw new IllegalThreadStateException();
    加入线程组里面
         /* Notify the group that this thread is bout to be started
         * so that it can be added to the group's list of threads
         * and the group's unstarted count can be decremented. */
         group.add(this);
         boolean started = false;
         try {
             start0();
             started = true;
         } finally {
             try {
               if (!started) {
                   如果启动失败了，放入失败组
                  group.threadStartFailed(this);
               }
            } catch (hrowable ignore) {
              /* do nothing. If start0 threw a Throwable then
            &

bsp;   it will be passed up the call stack */
           }
        }
    }
    private native void start0();
    /**
    如果构造函数被用于接受一个 Runable，那么这个方法会被调用，否则不会返任何东西
     * If this thread was constructed using a separate
     * Runnable run object, then that
     * Runnable object's run method is called;
     * otherwise, this method does nothing and returns.
     *
     * Subclasses of Thread should override this method.
     *
     * @see   #start()
     * @see   #stop()
     * @see   #Thread(ThreadGroup, Runnable, String)
     */
    @Override
    public void run() {
        if (target != null) {
            target.ru();
        }
    }
    /**
    这个方法被系统调用，并给这个线程机会去清理在正在退出户之前，所有参数设置为 null
     * This method is called by the system to give a Thread
     * a chance to clean up before it actually exits.
     */
    private void exit() {
        if (group != null) {
            group.threadTerminated(this);
            group = null;
        }
        /* Aggressively null out all reference fields: see bug 4006245 */
        target = null;
        /* Speed the release of some of these esources */
        threadLocals = null;
        inheritableThreadLocals = null;
        inheritedAccessControlContext = null
        blocker = null;
        uncaughtExceptionHandler = null;</>
    }
    /**

    强制线程停止执行
     * Forces the thread to stop executing.
     *
    如果有安全管理被安装，则它的 checkAccess 调用方法 shis 调用，这可能会致 SecurityException 被引发，在当前线程中。
     * If there is a security manager installed, its checkAccess
     * method is called with this
     * as its argument. This may result in a
     * SecurityException being raised (in the current thread).
     *
    如果这个线程和当前线程不一样（这个是，当前的线程尝试去停止除本身以外线程）
    安全管理器的 checkPermission 方法被调动。另外这个可能被抛出 SecurityException 异常在当前线程中。
     * If this thread is different from the current thread (that is, te current
     * thread is trying to stop a thread other than itself), the
     * security manager's checkPermission method (with a
     * RuntimePermission("stopThread") argument) is called in
     * addition.
     * Again, this may result in throwing a
     * SecurityException (in the current thread).
     *
    这个线程相当于被这个线程强制停止，异常的和去新建一个 ThreadDeath 对作为异常
    允许停止一个线程如果一个线程还没开始，如果线程已经启动了，他立即终止
     * The thread represented by this thread is forced to stop watever
     * it is doing abnormally and to throw a newly created
     * ThreadDeath object as an exception.
     *
     * It is permitted to stop a thread that has not yet been started.
     * If the thread is eventually started, it immediately terminats.
     *
    一个应用不应该去尝试捕获 ThreadDeath 除非他必须做一些额外的清理操作
     * An application should not normally try to catch
     * ThreadDeath unless it must do some extraordinary
     * cleanup operation (note that the throwing of
     * ThreadDeath causes finally clauses of
     * try statements to be executed before the thread
     * officially dies). If a catch clause catches a
     * ThreadDeath object, it is important to rethrow the
     * object so that the thread actually dies.
     *
     * The top-level error handler that reacts to otherwise uncaht

     * exceptions does not print out a message or otherwise noti
y the
     * application if the uncaught exception is an instance of</
>
     * ThreadDeath.
     *
     * @exception SecurityException if the current
thread cannot
     *
nbsp;   modify this thread.
     * @see   #
nterrupt()
     * @see   #
heckAccess()
     * @see   #
un()
     * @see   #
tart()
     * @see
hreadDeath
     * @see
hreadGroup#uncaughtException(Thread,Throwable)
     * @see   S
curityManager#checkAccess(Thread)
     * @see   S
curityManager#checkPermission
    这个方法本质上是不安全的，停止线程的过程中会打开所有已经被锁的监视器

     * @deprecated This method is inherently unsafe. St
pping a thread with
     *   Thread.stop c
uses it to unlock all of the monitors that it
     *   has locked (as
a natural consequence of the unchecked
     *   ThreadDeath
xception propagating up the stack). If
     *   any of the obj
cts previously protected by these monitors were in
     *   an inconsistent
state, the damaged objects become visible to
     *   other threads,
potentially resulting in arbitrary behavior. Many
     *   uses of stop s
ould be replaced by code that simply
     *   modifies some
variable to indicate that the target thread should
     *   stop running.
nbsp; The target thread should check this variable
     *   regularly, and
eturn from its run method in an orderly fashion
     *   if the variable
ndicates that it is to stop running. If the
     *   target thread
aits for long periods (on a condition variable,

     *   for example), te interrupt method should be used to
     *   interrupt the ait.
     *   For more info mation, see
     *   {@docRoot}/../echnotes/guides/concurrency/threadPrimitiveDeprecation.html">Why
     *   are Thread.sto , Thread.suspend and Thread.resume Deprecated?.
     */
    @Deprecated
    public final void stop() {
        SecurityManager security = System.ge SecurityManager();
        if (security != null) {
            checkAc ess();
            if (this != Thread.currentThread()) {
              security.checkPermission(SecurityConstants.STOP_THREAD_PERMISSION); /p>
            }
        }
        // A zero status value corresponds to NEW", it can't change to
        // not-NEW because we hold the lock
        if (threadStatus != 0) {
           resume(); // Wake up thread if it was suspended; no-op otherwise
        }
        // The VM can handle all thread state
        stop0(new ThreadDeath());
    }
    /**
     * Throws {@code UnsupportedOperationE ception}.
     *
     * @param obj ignored
     *
     * @deprecated This method was originally designed to forc a thread to stop
     *   and th ow a given {@code Throwable} as an exception. It was
     *   inheren ly unsafe (see {@link #stop()} for details), and furthermore
     *   could e used to generate exceptions that the target thread was

     *   not prpared to handle.
     *   For moe information, see
     *   {@docoot}/../technotes/guides/concurrency/threadPrimitiveDeprecation.html">Why
     *   are Thread.stop, Thread.suspend and Thread.resume Deprecated?.
     */
    @Deprecated
    public final synchronized void stop(Throwable obj) {
        throw new UnsupportedOperationExcption();
    }
    /**中断这个线程
     * Interrupts this thread.
     *除非当前线程自己阻断了
     * Unless the current thread is interrupting itself, which is
     * always permitted, the {@link #checkAccess() checkAccess} method
     * of this thread is invoked, which may cause a {@link
     * SecurityException} to be thrown.
     *
     * If this thread is blocked in an invocation of the {@link
     * Object#wait() wait()}, {@link Object#wait(lon) wait(long)}, or {@link
     * Object#wait(long, int) wait(long, int)} methods of the {@link Object}
     * class, or of the {@link #join()}, {@link #join(long)}, {@link
     * #join(long, int)}, {@link #sleep(long)}, or {@link #sleep(long, int)},
     * methods of this class, then its interrupt status will be cleard and it
     * will receive an {@link InterruptedException}
     *
     * If this thread is blocked in an I/O operation upon an {@<a href="https://ld246.com/member/link" aria-name="link" class="tooltipped__user" target="_b

ank">link
     * java.nio.channels.InterruptibleChannel InterruptibleChann
l}
     * then the channel will be closed, the thread's interrupt</p

     * status will be set, and the thread will receive a {@<a href=
https://ld246.com/member/link" aria-name="link" class="tooltipped__user" target="_blank">l
nk
     * java.nio.channels.ClosedByInterruptException}.
     *
     * If this thread is blocked in a {@<a href="https://ld246.c
m/member/link" aria-name="link" class="tooltipped__user" target="_blank">link java.ni
.channels.Selector}
     * then the thread's interrupt status will be set and it will ret
rn
     * immediately from the selection operation, possibly with a
non-zero
     * value, just as if the selector's {@<a href="https://ld246.c
m/member/link" aria-name="link" class="tooltipped__user" target="_blank">link
     * java.nio.channels.Selector#wakeup wakeup} method were
nvoked.
     *
     * If none of the previous conditions hold then this thread's
nterrupt
     * status will be set.
     *
     * Interrupting a thread that is not alive need not have any e
fect.
     *
     * @throws SecurityException
     *
nbsp; if the current thread cannot modify this thread
     *
     * @revised 6.0
     * @spec JSR-51
     */
    public void interrupt() {
        if (this != Thread.currentThread())</p

            checkAc
ess();
        synchronized (blockerLock) {
            Interrupt
ble b = blocker;
            if (b != n
ll) {
            &
bsp;   interrupt0();   &nb
p; // Just to set the interrupt flag
            &
bsp;   b.interrupt(this);
            &
bsp;   return;
            }

       }
       interrupt0();
    }
    /**
    测试这个当前线程是否被阻塞了，阻塞状态
     * Tests whether the current thread has been interrupted.&n
sp; The
     * interrupted status of the thread is cleared by this method
  In
     * other words, if this method were to be called twice in suc
cession, the
     * second call would return false (unless the current thread
ere
     * interrupted again, after the first call had cleared its interru
ted
     * status and before the second call had examined it).
     *
     * A thread interruption ignored because a thread was not al
ve
     * at the time of the interrupt will be reflected by this metho

     * returning false.
     *
     * @return true if the current thread has been interru
ted;
     *   &
nbsp; false otherwise.
     * @see #isInterrupted()
     * @revised 6.0
     */
    public static boolean interrupted() {
        return currentThread().isInterrupted(t
ue);
    }
    /**
     * Tests whether this thread has been interrupted. Th
interrupted
     * status of the thread is unaffected by this method.
     *
     * A thread interruption ignored because a thread was not al
ve
     * at the time of the interrupt will be reflected by this metho

     * returning false.
     *
     * @return true if this thread has been interrupted;<
p>
     *   &
nbsp; false otherwise.
     * @see   #interrupted()
     * @revised 6.0
     */
    public boolean isInterrupted() {
        return isInterrupted(false);

    }
    /**
     * Tests if some Thread has been interrupted. The int
rrupted state
     * is reset or not based on the value of ClearInterrupted that
s
     * passed.
     */
    private native boolean isInterrupted(boolean ClearInterrupted);</
>
    /**
     * Throws {@<a href="https://ld246.com/member/link" aria
name="link" class="tooltipped__user" target="_blank">link NoSuchMethodError}.
     *这个方法本来是用于销毁一个线程而不做任何清理动作，每一
监视器都被帮助去被锁，他能产生思索，如果一个线程持有了一个保护关键系统资源的锁，被销毁了
但是没有释放资源，没有线程能再次拿到这个资源。如果另一个线程曾试图锁定这个资源会导致死锁
这种死锁表现通常会冻结自身，
     * @deprecated This method was originally designed to dest
oy this
     *   thread without any cleanup.
ny monitors it held would have
     *   remained locked. However, t
e method was never implemented.
     *   If if were to be implemented,
t would be deadlock-prone in
     *   much the manner of {@<a h
ef="https://ld246.com/member/link" aria-name="link" class="tooltipped__user" target="_bla
k">link #suspend}. If the target thread held
     *   a lock protecting a critical sys
em resource when it was
     *   destroyed, no thread could e
er access this resource again.
     *   If another thread ever attem
ted to lock this resource, deadlock
     *   would result. Such deadlocks
typically manifest themselves as
     *   "frozen" processes. For more
nformation, see
     *   {@docRoot}/../technotes/gui
es/concurrency/threadPrimitiveDeprecation.html">
     *   Why are Thread.stop, Thread
suspend and Thread.resume Deprecated?.
     * @throws NoSuchMethodError always
     */
    @Deprecated
    public void destroy() {
        throw new NoSuchMethodError();</p

    }
    /**
     * Tests if this thread is alive. A thread is alive if it has
     * been started and has not yet died.
     *
     * @return true if this thread is alive;

     *
nbsp; false otherwise.
     */
    public final native boolean isAlive();
    /**
     暂停这个线程
     * Suspends this thread.
     *
     * First, the checkAccess method of this thread is called
     * with no arguments. This may result in throwing a
     * SecurityException (in the current thread).
     *
         如果这个线程在活动中，他被暂停并且不会再有进一
的进展，指导被恢复
     * If the thread is alive, it is suspended and makes no furthe

     * progress unless and until it is resumed.
     *
     * @exception SecurityException if the current
thread cannot modify
     *
nbsp;   this thread.
     * @see #checkAccess
     会产生死锁问题
     * @deprecated   This method has been depre
ated, as it is
     *   inherently deadlock-prone. If the ta
get thread holds a lock on the
     *   monitor protecting a critical system resourc
when it is suspended, no
     *   thread can access this resource until the tar
et thread is resumed. If
     *   the thread that would resume the target th
ead attempts to lock this
     *   monitor prior to calling resume, deadlock r
sults. Such
     *   deadlocks typically manifest themselves as
frozen" processes.
     *   For more information, see
     *   {@docRoot}/../technotes/guides/concurren
y/threadPrimitiveDeprecation.html">Why
     *   are Thread.stop, Thread.suspend and Threa
.resume Deprecated?.
     */
    @Deprecated
    public final void suspend() {
        checkAccess();
        suspend0();
    }
    /**
     恢复一个被暂定的线程
     * Resumes a suspended thread.
     *
     * First, the checkAccess method of this thread is called

     * with no arguments. This may result in throwing a
     * SecurityException (in the current thread).
     *
     如果一个线程在活跃的时候被暂停了，被恢后他被允许继续在当前程序执行。
     * If the thread is alive but suspended, it is resumed and is
     * permitted to make progress in its execution.
     *
     * @exception SecurityException if the current thread cannot modify this
     *   nbsp;   thread.
     * @see   #heckAccess
     * @see   #uspend()
     容易出现线程死锁
     * @deprecated This method exists solely for use with {@link #suspend},
     *   which has been deprecated ecause it is deadlock-prone.
     *   For more information, see
     *   {@docRoot}/../technotes/guies/concurrency/threadPrimitiveDeprecation.html">Why
     *   are Thread.stop, Thread.suspnd and Thread.resume Deprecated?.
     */
    @Deprecated
    public final void resume() {
        checkAccess();
        resume0();
    }
    /**
     改变当前线程的优先级
     * Changes the priority of this thread.
     *
     * First the checkAccess method of this thread is called
     * with no arguments. This may result in throwing a
     * SecurityException.
     *
     否则，此线程的优先级设置为最小的值，制定的新优先级和最大的优先
     * Otherwise, the priority of this thread is set to the smaller o
     * the specified newPriority and the maximum permitted
     * priority of the thread's thread group.
     *
     * @param newPriority priority to set this thread to
     * @exception IllegalArgumentException If the priority is not in the

     *   nbsp;   range MIN_PRIORITY to
     *   nbsp;   MAX_PRIORITY.
     * @exception SecurityException if the current thread cannot modify
     *   nbsp;   this thread.
     * @see   getPriority
     * @see   #heckAccess()
     * @see   getThreadGroup()
     * @see   MAX_PRIORITY
     * @see   MIN_PRIORITY
     * @see   hreadGroup#getMaxPriority()
     */
    public final void setPriority(int newPriority) {
        ThreadGroup g;
        checkAccess();
        if (newPriority > MAX_PRIORITY || ewPriority < MIN_PRIORITY) {
            throw n w IllegalArgumentException();
        }
        if((g = getThreadGroup()) != null) {
            if (newPriority > g.getMaxPriority()) {
              newPriority = g.getMaxPriority();
            }
            setPriorit0(priority = newPriority);
        }
    }
    /**
     * Returns this thread's priority.
     *
     * @return this thread's priority.
     * @see   #setPriority
     */
    public final int getPriority() {
        return priority;
    }
    /**
     给线程一个名字，必须在线程启动之前
     * Changes the name of this thread to be equal to the argu ent
     * name.

     *
     * First the checkAccess method of this thread is called
     * with no arguments. This may result in throwing a
     * SecurityException.
     *
     * @param   name
  the new name for this thread.
     * @exception SecurityException if the current
thread cannot modify this
     *   &
nbsp;   thread.
     * @see
getName
     * @see   #
heckAccess()
     */
    public final synchronized void setName(String name) {
        checkAccess();
        this.name = name.toCharArray();
        if (threadStatus != 0) {
            setNativ
Name(name);
        }
    }
    /**
     * Returns this thread's name.
     *
     * @return this thread's name.
     * @see   #setName(String)
     */
    public final String getName() {
        return new String(name, true);
    }
    /**
     * Returns the thread group to which this thread belongs.</
>
     * This method returns null if this thread has died
     * (been stopped).
     *
     * @return this thread's thread group.
     */
    public final ThreadGroup getThreadGroup() {
        return group;
    }
    /**
     返回当前线程活动线程数的估计值，迭代递归当前所有子数组线程的线
组。
     * Returns an estimate of the number of active threads in the
current
     * thread's {@linkplain java.lang.ThreadGroup thread group}
and its
     * subgroups. Recursively iterates over all subgroups in the
urrent
     * thread's thread group.

     *返回的值仅仅是一个估计值，因为线程的数值会动态变化当方
，可能会受某些系统线程的影响，
     此方法主要用于吊事和监控目的
     * The value returned is only an estimate because the numb
r of
     * threads may change dynamically while this method traver
es internal
     * data structures, and might be affected by the presence of
ertain
     * system threads. This method is intended primarily for deb
gging
     * and monitoring purposes.
     *
     * @return an estimate of the number of active threa
s in the current
     *
nbsp; thread's thread group and in any other thread group that
     *
nbsp; has the current thread's thread group as an ancestor
     */
    public static int activeCount() {
        return currentThread().getThreadGro
p().activeCount();
    }
    /**
     * Copies into the specified array every active thread in the c
rrent
     * thread's thread group and its subgroups. This method si
ply
     * invokes the {@<a href="https://ld246.com/member/link"
ria-name="link" class="tooltipped__user" target="_blank">link java.lang.ThreadGroup#e
umerate(Thread[])}
     * method of the current thread's thread group.
     *
     * An application might use the {@linkplain #activeCount ac
iveCount}
     * method to get an estimate of how big the array should be
 however
     * if the array is too short to hold all the threads, the extra th
eads
     * are silently ignored. If it is critical to obtain every a
tive
     * thread in the current thread's thread group and its subgr
ups, the
     * invoker should verify that the returned int value is strictly
ess
     * than the length of {@<a href="https://ld246.com/membe
/code" aria-name="code" class="tooltipped__user" target="_blank">code tarray}.
     *
     * Due to the inherent race condition in this method, it is re
ommended
     * that the method only be used for debugging and monitor
ng purposes.
     *

     * @param tarray
     *
n array into which to put the list of threads
     *
     * @return the number of threads put into the array<
p>
     *
     * @throws SecurityException
     *
nbsp; if {@<a href="https://ld246.com/member/link" aria-name="link" class="tooltipped__us
r" target="_blank">link java.lang.ThreadGroup#checkAccess} determines that
     *
nbsp; the current thread cannot access its thread group
     */
    public static int enumerate(Thread tarray[]) {
        return currentThread().getThreadGro
p().enumerate(tarray);
    }
    /**
     * Counts the number of stack frames in this thread. The thr
ad must
     * be suspended.
     *
     * @return   the number of stack
frames in this thread.
     * @exception IllegalThreadStateException if
his thread is not
     *
nbsp;   suspended.
     * @deprecated The definition of this call depends on {@<a
href="https://ld246.com/member/link" aria-name="link" class="tooltipped__user" target="_b
ank">link #suspend},
     *
nbsp;   which is deprecated. Further, the results of this call
     *
nbsp;   were never well-defined.
     */
    @Deprecated
    public native int countStackFrames();
    /**
     等待时间这个线程结束，这个实现是一直循环看这个线程是否还存活，
线程结束的售后，其他线程被唤醒，被建议用 notifyall 代替，表示一个线程插入进来先运行
     * Waits at most {@<a href="https://ld246.com/member/co
e" aria-name="code" class="tooltipped__user" target="_blank">code millis} milliseconds
for this thread to
     * die. A timeout of {@<a href="https://ld246.com/member/
ode" aria-name="code" class="tooltipped__user" target="_blank">code 0} means to wai
 forever.
     *
     * This implementation uses a loop of {@<a href="https://l
246.com/member/code" aria-name="code" class="tooltipped__user" target="_blank">code<
a> this.wait} calls
     * conditioned on {@<a href="https://ld246.com/member/c
de" aria-name="code" class="tooltipped__user" target="_blank">code this.isAlive}. As a

hread terminates the
     * {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code this.notifyAll} method is invoked.
t is recommended that
     * applications not use {@<a href="https://ld246.com/mem
er/code" aria-name="code" class="tooltipped__user" target="_blank">code wait}, {@<a
href="https://ld246.com/member/code" aria-name="code" class="tooltipped__user" target="
blank">code notify}, or
     * {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code notifyAll} on {@<a href="https://
d246.com/member/code" aria-name="code" class="tooltipped__user" target="_blank">code
/a> Thread} instances.
     *
     * @param millis
     *
he time to wait in milliseconds
     *
     * @throws IllegalArgumentException
     *
nbsp; if the value of {@<a href="https://ld246.com/member/code" aria-name="code" class="
ooltipped__user" target="_blank">code millis} is negative
     *
     * @throws InterruptedException
     *
nbsp; if any thread has interrupted the current thread. The
     *
nbsp; interrupted status of the current thread is
     *
nbsp; cleared when this exception is thrown.
     */
    public final synchronized void join(long millis)
    throws InterruptedException {
         long base = System.currentTimeMillis(
;
         long now = 0;
         if (millis < 0) {
              throw n
w IllegalArgumentException("timeout value is negative");
         }
         if (millis == 0) {
              while (isA
ive()) {
              &
bsp;   wait(0);
             }
         } else {
              while (isA
ive()) {
              &
bsp;   long delay = millis - now;
              &
bsp;   if (delay <= 0) {
              &
bsp;   break;

               }
               wait(delay);
               now = System.currentTimeMillis() - base;
             }
        }
    }
    /**
     * Waits at most {@<a href="https://ld246.com/member/co
e" aria-name="code" class="tooltipped__user" target="_blank">code millis} milliseconds
plus
     * {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code nanos} nanoseconds for this thre
d to die.
     *
     * This implementation uses a loop of {@<a href="https://l
246.com/member/code" aria-name="code" class="tooltipped__user" target="_blank">code<
a> this.wait} calls
     * conditioned on {@<a href="https://ld246.com/member/c
de" aria-name="code" class="tooltipped__user" target="_blank">code this.isAlive}. As a
hread terminates the
     * {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code this.notifyAll} method is invoked.
t is recommended that
     * applications not use {@<a href="https://ld246.com/mem
er/code" aria-name="code" class="tooltipped__user" target="_blank">code wait}, {@<a
href="https://ld246.com/member/code" aria-name="code" class="tooltipped__user" target="
blank">code notify}, or
     * {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code notifyAll} on {@<a href="https://
d246.com/member/code" aria-name="code" class="tooltipped__user" target="_blank">code
/a> Thread} instances.
     *
     * @param millis
     *
he time to wait in milliseconds
     *
     * @param nanos
     *
@<a href="https://ld246.com/member/code" aria-name="code" class="tooltipped__user" tar
et="_blank">code 0-999999} additional nanoseconds to wait
     *
     * @throws IllegalArgumentException
     *
nbsp; if the value of {@<a href="https://ld246.com/member/code" aria-name="code" class="
ooltipped__user" target="_blank">code millis} is negative, or the value
     *
nbsp; of {@<a href="https://ld246.com/member/code" aria-name="code" class="tooltipped_
user" target="_blank">code nanos} is not in the range {@<a href="https://ld246.com/m
mber/code" aria-name="code" class="tooltipped__user" target="_blank">code 0-99999
}
     *

```
<p>     * @throws  InterruptedException</p>
<p>     *        
nbsp; if any thread has interrupted the current thread. The</p>
<p>     *        
nbsp; interrupted status of the current thread is</p>
<p>     *        
nbsp; cleared when this exception is thrown.</p>
<p>     */</p>
<p>    public final synchronized void join(long millis, int nanos)</p>
<p>    throws InterruptedException {</p>
<p>        if (millis &lt; 0) {</p>
<p>            throw n
w IllegalArgumentException("timeout value is negative");</p>
<p>        }</p>
<p>        if (nanos &lt; 0 || nanos &gt; 999999)
</p>
<p>            throw n
w IllegalArgumentException(</p>
<p>            &
bsp;            &
bsp;      "nanosecond timeout value out of range");</p>
<p>        }</p>
<p>        if (nanos &gt;= 500000 || (nanos != 0
&amp;&amp; millis == 0)) {</p>
<p>            millis++
</p>
<p>        }</p>
<p>        join(millis);</p>
<p>    }</p>
<p>    /**</p>
<p>     * Waits for this thread to die.</p>
<p>     *</p>
<p>     *  An invocation of this method behaves in exactly the sam
</p>
<p>     * way as the invocation</p>
<p>     *</p>
<p>     *</p>
<p>     * {@linkplain #join(long) join}{@<a href="https://ld246.com
member/code" aria-name="code" class="tooltipped__user" target="_blank">code</a> (0)}<
p>
<p>     *</p>
<p>     *</p>
<p>     * @throws  InterruptedException</p>
<p>     *        
nbsp; if any thread has interrupted the current thread. The</p>
<p>     *        
nbsp; interrupted status of the current thread is</p>
<p>     *        
nbsp; cleared when this exception is thrown.</p>
<p>     */</p>
<p>    public final void join() throws InterruptedException {</p>
<p>        join(0);</p>
<p>    }</p>
<p>    /**</p>
```

     * Prints a stack trace of the current thread to the standard e
ror stream.
     * This method is used only for debugging.
     *
     * @see   Throwable#printStackT
ace()
     */
    public static void dumpStack() {
        new Exception("Stack trace").printSta
kTrace();
    }
    /**
     * Marks this thread as either a {@linkplain #isDaemon dae
on} thread
     * or a user thread. The Java Virtual Machine exits when the
nly
     * threads running are all daemon threads.
     *
     * This method must be invoked before the thread is started

     *
     * @param on
     *   i
{@<a href="https://ld246.com/member/code" aria-name="code" class="tooltipped__user" ta
get="_blank">code true}, marks this thread as a daemon thread
     *
     * @throws IllegalThreadStateException
     *
nbsp; if this thread is {@linkplain #isAlive alive}
     *
     * @throws SecurityException
     *
nbsp; if {@<a href="https://ld246.com/member/link" aria-name="link" class="tooltipped__us
r" target="_blank">link #checkAccess} determines that the current
     *
nbsp; thread cannot modify this thread
     */
    public final void setDaemon(boolean on) {
        checkAccess();
        if (isAlive()) {
            throw n
w IllegalThreadStateException();
        }
        daemon = on;
    }
    /**
     * Tests if this thread is a daemon thread.
     *
     * @return true if this thread is a daemon thread;</p

     *
nbsp; false otherwise.
     * @see   #setDaemon(boolean)
/p>

<p>   */</p>
<p>   public final boolean isDaemon() {</p>
<p>   return daemon;</p>
<p>   }</p>
<p>   /**</p>
<p>   确认当前当前在运行的线程时候有权限修改这个线程</p>
<p>   * Determines if the currently running thread has permission to</p>
<p>   * modify this thread.</p>
<p>   *</p>
<p>   </p>
<p>   * If there is a security manager, its checkAccess method</p

<p>   * is called with this thread as its argument. This may result i
</p>
<p>   * throwing a SecurityException.</p>
<p>   *</p>
<p>   * @exception SecurityException if the current thread is not allowed to</p>
<p>   *
nbsp;   access this thread.</p>
<p>   * @see   S
curityManager#checkAccess(Thread)</p>
<p>   */</p>
<p>   public final void checkAccess() {</p>
<p>   SecurityManager security = System.ge
SecurityManager();</p>
<p>   if (security != null) {</p>
<p>   security.
heckAccess(this);</p>
<p>   }</p>
<p>   }</p>
<p>   /**</p>
<p>   * Returns a string representation of this thread, including t
e</p>
<p>   * thread's name, priority, and thread group.</p>
<p>   *</p>
<p>   * @return a string representation of this thread.</p

<p>   */</p>
<p>   public String toString() {</p>
<p>   ThreadGroup group = getThreadGro
p();</p>
<p>   if (group != null) {</p>
<p>   return "
hread[" + getName() + "," + getPriority() + "," +</p>
<p>   &
bsp;   &
bsp; group.getName() + "]";</p>
<p>   } else {</p>
<p>   return "
hread[" + getName() + "," + getPriority() + "," +</p>
<p>   &
bsp;   &

bsp; "" + "]";
       }
    }
    /**
     返回一个上下文环境信息
     * Returns the context ClassLoader for this Thread. The cont
xt
     * ClassLoader is provided by the creator of the thread for u
e
     * by code running in this thread when loading classes and r
sources.
     * If not {@linkplain #setContextClassLoader set}, the default
is the
     * ClassLoader context of the parent Thread. The context Cla
sLoader of the
     * primordial thread is typically set to the class loader used t
 load the
     * application.
     *
     * If a security manager is present, and the invoker's class lo
der is not
     * {@<a href="https://ld246.com/member/code" aria-name
"code" class="tooltipped__user" target="_blank">code null} and is not the same as or an
ancestor of the context class
     * loader, then this method invokes the security manager's
@<a href="https://ld246.com/member/link" aria-name="link" class="tooltipped__user" targe
="_blank">link
     * SecurityManager#checkPermission(java.security.Permissio
) checkPermission}
     * method with a {@<a href="https://ld246.com/member/li
k" aria-name="link" class="tooltipped__user" target="_blank">link RuntimePermission
untimePermission}{@<a href="https://ld246.com/member/code" aria-name="code" class="t
oltipped__user" target="_blank">code
     * ("getClassLoader")} permission to verify that retrieval of t
e context
     * class loader is permitted.
     *
     * @return the context ClassLoader for this Thread, or
{@<a href="https://ld246.com/member/code" aria-name="code" class="tooltipped__user" ta
get="_blank">code null}
     *   &
nbsp; indicating the system class loader (or, failing that, the
     *   &
nbsp; bootstrap class loader)
     *
     * @throws SecurityException
     *   &
nbsp; if the current thread cannot get the context ClassLoader
     *
     * @since 1.2
     */
    @CallerSensitive
    public ClassLoader getContextClassLoader() {
        if (contextClassLoader == null)

<p>   return nu
l;</p>
<p>   SecurityManager sm = System.getSec
rityManager();</p>
<p>   if (sm != null) {</p>
<p>   ClassLoa
er.checkClassLoaderPermission(contextClassLoader,</p>
<p>   &
bsp;   &
bsp;   &
bsp;   Reflectio
.getCallerClass());</p>
<p>   }</p>
<p>   return contextClassLoader;</p>
<p>   }</p>
<p>   /**</p>
<p>   * Sets the context ClassLoader for this Thread. The context<
p>
<p>   * ClassLoader can be set when a thread is created, and allo
s</p>
<p>   * the creator of the thread to provide the appropriate class
oader,</p>
<p>   * through {@<a href="https://ld246.com/member/code" ar
a-name="code" class="tooltipped__user" target="_blank">code getContextClassLoader},
to code running in the thread</p>
<p>   * when loading classes and resources.</p>
<p>   *</p>
<p>   * If a security manager is present, its {@<a href="https://ld
46.com/member/link" aria-name="link" class="tooltipped__user" target="_blank">link<
p>
<p>   * SecurityManager#checkPermission(java.security.Permissio
) checkPermission}</p>
<p>   * method is invoked with a {@<a href="https://ld246.com/
ember/link" aria-name="link" class="tooltipped__user" target="_blank">link RuntimePe
mission RuntimePermission}{@<a href="https://ld246.com/member/code" aria-name="code"
class="tooltipped__user" target="_blank">code</p>
<p>   * ("setContextClassLoader")} permission to see if setting the
context</p>
<p>   * ClassLoader is permitted.</p>
<p>   *</p>
<p>   * @param cl</p>
<p>   *
he context ClassLoader for this Thread, or null indicating the</p>
<p>   *
ystem class loader (or, failing that, the bootstrap class loader)</p>
<p>   *</p>
<p>   * @throws SecurityException</p>
<p>   *
nbsp; if the current thread cannot set the context ClassLoader</p>
<p>   *</p>
<p>   * @since 1.2</p>
<p>   */</p>
<p>   public void setContextClassLoader(ClassLoader cl) {</p>
<p>   SecurityManager sm = System.getSec

rityManager();
        if (sm != null) {
             sm.checPermission(new RuntimePermission("setContextClassLoader"));
        }
        contextClassLoader = cl;
    }
    /**
     仅当当前线程持有锁的时候，返回 true
     这个方法设计师为了云溪一个程序去断定当前线程已经持有了一个特殊锁
     * Returns true if and only if the current thread holds the
     * monitor lock on the specified object.
     *
     * This method is designed to allow a program to assert tha
     * the current thread already holds a specified lock:
     *
     *   assert Thread.holdsLock(obj)
     *
     *
     * @param obj the object on which to test lock ownership
     * @throws NullPointerException if obj is null
     * @return true if the current thread holds the monitor lock n
     *   he specified object.
     * @since 1.4
     */
    public static native boolean holdsLock(Object obj);
    private static final StackTraceElement[] EMPTY_STACK_TRACE
        = new StackTraceElement[0];
    /**
     返回堆栈元素代表栈堆数组的当前线程中，这个方法会返回长度为 0 的组如果这个线程没有被启动，已经被启动但是被预定运行时间，或者已经终止。如果返回的数组长度是 0 的话，那么数组的第一个元素表示堆栈的丁部，是最近使用的调用方法。数组的最后一个元素表推展的底部。这是最近使用的调用
     * Returns an array of stack trace elements representing the tack dump
     * of this thread. This method will return a zero-lengt array if
     * this thread has not started, has started but has not yet be n
     * scheduled to run by the system, or has terminated.
     * If the returned array is of non-zero length then the first el ment of
     * the array represents the top of the stack, which is the mos recent
     * method invocation in the sequence. The last elem nt of the array
     * represents the bottom of the stack, which is the least rece

t method
     * invocation in the sequence.
     *
     * If there is a security manager, and this thread is not
     * the current thread, then the security manager's
     * checkPermission method is called with a
     * RuntimePermission("getStackTrace") permission
     * to see if it's ok to get the stack trace.
     *
     * Some virtual machines may, under some circumstances,
mit one
     * or more stack frames from the stack trace. In the e
treme case,
     * a virtual machine that has no stack trace information conc
rning
     * this thread is permitted to return a zero-length array from
this
     * method.
     *
     * @return an array of StackTraceElement,
     * each represents one stack frame.
     *
     * @throws SecurityException
     *   if a sec
rity manager exists and its
     *   checkP
rmission method doesn't allow
     *   getting
the stack trace of thread.
     * @see SecurityManager#checkPermission
     * @see RuntimePermission
     * @see Throwable#getStackTrace
     *
     * @since 1.5
     */
    public StackTraceElement[] getStackTrace() {
         if (this != Thread.currentThread()) {</
>
             // check
or getStackTrace permission
             Security
anager security = System.getSecurityManager();
             if (securi
y != null) {
            &
bsp;   security.checkPermission(
            &
bsp;   SecurityConstants.GET_STACK_TRACE_PERMI
SION);
             }
             // optimi
ation so we do not call into the vm for threads that
             // have
ot yet started or have terminated

           if (!isAliv
()) {
           &
bsp;   return EMPTY_STACK_TRACE;
            }
            StackTra
eElement[][] stackTraceArray = dumpThreads(new Thread[] {this});
            StackTra
eElement[] stackTrace = stackTraceArray[0];
            // a thre
d that was alive during the previous isAlive call may have
             // since t
rminated, therefore not having a stacktrace.
             if (stackT
ace == null) {
            &
bsp;   stackTrace = EMPTY_STACK_TRACE;
             }
             return st
ckTrace;
        } else {
             // Don't
eed JVM help for current thread
             return (n
w Exception()).getStackTrace();
        }
    }
    /**
     * Returns a map of stack traces for all live threads.
     * The map keys are threads and each map value is an array
f
     * StackTraceElement that represents the stack dump
     * of the corresponding Thread.
     * The returned stack traces are in the format specified for<
p>
     * the {@<a href="https://ld246.com/member/link" aria-na
e="link" class="tooltipped__user" target="_blank">link #getStackTrace getStackTrace}
ethod.
     *
     * The threads may be executing while this method is called

     * The stack trace of each thread only represents a snapshot
and
     * each stack trace may be obtained at different time.
A zero-length
     * array will be returned in the map value if the virtual machi
e has
     * no stack trace information about a thread.
     *
     * If there is a security manager, then the security manager'

     * checkPermission method is called with a
     * RuntimePermission("getStackTrace") permission as well a

     * RuntimePermission("modifyThreadGroup") permission</
>
     * to see if it is ok to get the stack trace of all threads.
     *
     * @return a Map from Thread to an array of
     * StackTraceElement that represents the stack trace of
     * the corresponding thread.
     *
     * @throws SecurityException
     *   if a sec
rity manager exists and its
     *   checkP
rmission method doesn't allow
     *   getting
the stack trace of thread.
     * @see #getStackTrace
     * @see SecurityManager#checkPermission
     * @see RuntimePermission
     * @see Throwable#getStackTrace
     *
     * @since 1.5
     */
    public static Map, StackTraceElement[]> getAllStackTraces() {</
>
        // check for getStackTrace permissio

        SecurityManager security = System.ge
SecurityManager();
        if (security != null) {
            security.
heckPermission(
           &
bsp;   SecurityConstants.GET_STACK_TRACE_PERMISSION);
            security.
heckPermission(
           &
bsp;   SecurityConstants.MODIFY_THREADGROUP_PERMISSION);
        }
        // Get a snapshot of the list of all thre
ds
        Thread[] threads = getThreads();
        StackTraceElement[][] traces = dump
hreads(threads);
        Map, StackTraceElement[]> m = n
w HashMap<>(threads.length);
         for (int i = 0; i < threads.length; i++)
{
             StackTra
eElement[] stackTrace = traces[i];
             if (stackT
ace != null) {
            &
bsp;   m.put(threads[i], stackTrace);
             }

           // else te
minated so we don't put it in the map
        }
        return m;
    }
    private static final RuntimePermission SUBCLASS_IMPLEMENTATI
N_PERMISSION =
           &
bsp;   new RuntimePermission("enableContextClass
oaderOverride");
    /** cache of subclass security audit results */
    /* Replace with ConcurrentReferenceHashMap when/if it appears i
a future
     * release */
    private static class Caches {
         /** cache of subclass security audit re
ults */
         static final ConcurrentMap,Boolean&g
; subclassAudits =
            new Con
urrentHashMap<>();
         /** queue for WeakReferences to audi
ed subclasses */
         static final ReferenceQueue> subcl
ssAuditsQueue =
             new Refe
enceQueue<>();
    }
    /**
     * Verifies that this (possibly subclass) instance can be const
ucted
     * without violating security constraints: the subclass must n
t override
     * security-sensitive non-final methods, or else the
     * "enableContextClassLoaderOverride" RuntimePermission i
checked.
     */
    private static boolean isCCLOverridden(Class cl) {
         if (cl == Thread.class)
             return fa
se;
         processQueue(Caches.subclassAudits
ueue, Caches.subclassAudits);
         WeakClassKey key = new WeakClassK
y(cl, Caches.subclassAuditsQueue);
         Boolean result = Caches.subclassAudi
s.get(key);
         if (result == null) {
             result =
oolean.valueOf(auditSubclass(cl));
             Caches.s
bclassAudits.putIfAbsent(key, result);
         }
         return result.booleanValue();

```
<p>    }</p>
<p>    /**</p>
<p>     * Performs reflective checks on given subclass to verify that
t doesn't</p>
<p>     * override security-sensitive non-final methods.  Retu
ns true if the</p>
<p>     * subclass overrides any of the methods, false otherwise.</
>
<p>     */</p>
<p>    private static boolean auditSubclass(final Class subcl) {</p>
<p>        Boolean result = AccessController.doP
ivileged(</p>
<p>            new Privi
egedAction() {</p>
<p>            &
bsp;   public Boolean run() {</p>
<p>            &
bsp;       for (Class cl = subcl;</p>
<p>            &
bsp;            cl != Thr
ad.class;</p>
<p>            &
bsp;            cl = cl.ge
Superclass())</p>
<p>            &
bsp;       {</p>
<p>            &
bsp;           try {</p>
<p>            &
bsp;            &
bsp;  cl.getDeclaredMethod("getContextClassLoader", new Class[0]);</p>
<p>            &
bsp;            &
bsp;  return Boolean.TRUE;</p>
<p>            &
bsp;           } catch (NoSuc
MethodException ex) {</p>
<p>            &
bsp;           }</p>
<p>            &
bsp;           try {</p>
<p>            &
bsp;            &
bsp;  Class[] params = {ClassLoader.class};</p>
<p>            &
bsp;            &
bsp;  cl.getDeclaredMethod("setContextClassLoader", params);</p>
<p>            &
bsp;            &
bsp;  return Boolean.TRUE;</p>
<p>            &
bsp;           } catch (NoSuc
MethodException ex) {</p>
<p>            &
```

bsp;   }
                  }
                  return Boolean.FALSE;
                }
            }
        );
        return result.booleanValue();
    }
    private native static StackTraceElement[][] dumpThreads(Thread[] t
reads);
    private native static Thread[] getThreads();
    /**
     * Returns the identifier of this Thread. The thread ID
s a positive
     * long number generated when this thread was created.</
>
     * The thread ID is unique and remains unchanged during its
lifetime.
     * When a thread is terminated, this thread ID may be reused

     *
     * @return this thread's ID.
     * @since 1.5
     */
    public long getId() {
        return tid;
    }
    /**
     * A thread state. A thread can be in one of the follow
ng states:
     *
     * {@<a href="https://ld246.com/member/link" aria-name="
ink" class="tooltipped__user" target="_blank">link #NEW}
     *   A thread that has not yet star
ed is in this state.
     *
     * {@<a href="https://ld246.com/member/link" aria-name="
ink" class="tooltipped__user" target="_blank">link #RUNNABLE}
     *   A thread executing in the Jav
 virtual machine is in this state.
     *
     * {@<a href="https://ld246.com/member/link" aria-name="
ink" class="tooltipped__user" target="_blank">link #BLOCKED}
     *   A thread that is blocked wait
ng for a monitor lock
     *   is in this state.
     *
     * {@<a href="https://ld246.com/member/link" aria-name="
ink" class="tooltipped__user" target="_blank">link #WAITING}
     *   A thread that is waiting indef
nitely for another thread to

     *   perform a particular action is n this state.
     *
     * {@link #TIMED_WAITING}
     *   A thread that is waiting for a other thread to perform an action
     *   for up to a specified waiting t me is in this state.
     *
     * {@link #TERMINATED}
     *   A thread that has exited is in his state.
     *
     *
     *
     *
     * A thread can be in only one state at a given point in time. /p>
     * These states are virtual machine states which do not reflec
     * any operating system thread states.
     *
     * @since   1.5
     * @see #getState
     */
    public enum State {
        /**
          线程状态：一个线程还未启动的转台
         * Thread state for a thread whi h has not yet started.
         */
        NEW,
        /**
          正在运行的线程状态，一个线程在运行中在 java 虚机中但是可能等待资源
         * Thread state for a runnable t read. A thread in the runnable
         * state is executing in the Java irtual machine but it may
         * be waiting for other resource from the operating system
         * such as processor.
         */
        RUNNABLE,
        /**
          线程被阻塞了一个线程等待监视器锁的时候
         * Thread state for a thread blo ked waiting for a monitor lock.
         * A thread in the blocked state s waiting for a monitor lock
         * to enter a synchronized bloc /method or

         * reenter a synchronized block method after calling
          * {@link Object wait() Object.wait}.
          */
          BLOCKED,
          /**
          线程状态等待由于调用下面中的一个方法
          wait 超市
          其他线程的 join
          被锁
          * Thread state for a waiting thread.
          * A thread is in the waiting state due to calling one of the
          * following methods:
          *
          *   {@link Object#wait() Object.wait} with no timeout
          *   {@link #join() Thread.join} with no timeout
          *   {@link LockSupport#park() LockSupport.park}
          *
          *
          * A thread in the waiting state waiting for another thread to
          * perform a particular action.
          *例如，一个线程被调用了 wait 一个 object 上，指导其他线程调用 notify 或者 notifyAll 在这个 object 上，一个线程被调用了 join 为了登台这个线程被中断
          * For example, a thread that has called Object.wait()
          * on an object is waiting for another thread to call
          * Object.notify() or Object.notiyAll() on
          * that object. A thread that has called Thread.join()
          * is waiting for a specified thred to terminate.
          */
          WAITING,
          /**这个状态是进入等待
          * Thread state for a waiting thread with a specified waiting time.
          * A thread is in the timed waiting state due to calling one of
          * the following methods with a

specified positive waiting time:
         *
         *   {@<a href="ht
ps://ld246.com/member/link" aria-name="link" class="tooltipped__user" target="_blank">lin
 #sleep Thread.sleep}
         *   {@<a href="ht
ps://ld246.com/member/link" aria-name="link" class="tooltipped__user" target="_blank">lin
 Object#wait(long) Object.wait} with timeout
         *   {@<a href="ht
ps://ld246.com/member/link" aria-name="link" class="tooltipped__user" target="_blank">lin
 #join(long) Thread.join} with timeout
         *   {@<a href="ht
ps://ld246.com/member/link" aria-name="link" class="tooltipped__user" target="_blank">lin
 LockSupport#parkNanos LockSupport.parkNanos}
         *   {@<a href="ht
ps://ld246.com/member/link" aria-name="link" class="tooltipped__user" target="_blank">lin
 LockSupport#parkUntil LockSupport.parkUntil}
         *
         */
        TIMED_WAITING,
        /**线程被中断了，或者这个线程已经完
运行结束了
         * Thread state for a terminated
thread.
         * The thread has completed ex
cution.
         */
        TERMINATED;
    }
    /**
     * Returns the state of this thread.
     * This method is designed for use in monitoring of the syst
m state,
     * not for synchronization control.
     *
     * @return this thread's state.
     * @since 1.5
     */
    public State getState() {
        // get current thread state
        return sun.misc.VM.toThreadState(thr
adStatus);
    }
    // Added in JSR-166
    /**
     * Interface for handlers invoked when a Thread abruptly</
>
     * terminates due to an uncaught exception.
     * When a thread is about to terminate due to an uncaught
xception
     * the Java Virtual Machine will query the thread for its
     * UncaughtExceptionHandler using
     * {@<a href="https://ld246.com/member/link" aria-name="
ink" class="tooltipped__user" target="_blank">link #getUncaughtExceptionHandler} and

will invoke the handler's
    * uncaughtException method, passing the thread and the<
p>
    * exception as arguments.
    * If a thread has not had its UncaughtExceptionHandler</p

    * explicitly set, then its ThreadGroup object acts as its
    * UncaughtExceptionHandler. If the ThreadGroup object</
>
    * has no
    * special requirements for dealing with the exception, it can
orward
    * the invocation to the {@linkplain #getDefaultUncaughtEx
eptionHandler
    * default uncaught exception handler}.
    *
    * @see #setDefaultUncaughtExceptionHandler
    * @see #setUncaughtExceptionHandler
    * @see ThreadGroup#uncaughtException
    * @since 1.5
    */
   @FunctionalInterface
   public interface UncaughtExceptionHandler {
        /**
         * Method invoked when the gi
en thread terminates due to the
         * given uncaught exception.</
>
         * Any exception thrown by this
method will be ignored by the
         * Java Virtual Machine.
         * @param t the thread
         * @param e the exception</p

         */
        void uncaughtException(Thread t, Thr
wable e);
    }
    // null unless explicitly set
    private volatile UncaughtExceptionHandler uncaughtExceptionHan
ler;
    // null unless explicitly set
    private static volatile UncaughtExceptionHandler defaultUncaught
xceptionHandler;
    /**
     * Set the default handler invoked when a thread abruptly te
minates
     * due to an uncaught exception, and no other handler has
een defined
     * for that thread.
     *
     * Uncaught exception handling is controlled first by the thr
ad, then
     * by the thread's {@<a href="https://ld246.com/member/li

k" aria-name="link" class="tooltipped__user" target="_blank">link ThreadGroup} object and finally by the default
     * uncaught exception handler. If the thread does not have n explicit
     * uncaught exception handler set, and the thread's thread group
     * (including parent thread groups) does not specialie its
     * uncaughtException method, then the default handler's>
     * uncaughtException method will be invoked.
     * By setting the default uncaught exception handler, an application
     * can change the way in which uncaught exceptions are handled (such as
     * logging to a specific device, or file) for those threads that would
     * already accept whatever "default" behavior the system>
     * provided.
     *
     * Note that the default uncaught exception handler should ot usually
     * defer to the thread's ThreadGroup object, as that could cuse
     * infinite recursion.
     *
     * @param eh the object to use as the default uncaught exception handler.
     * If null then there is no default handler.
     *
     * @throws SecurityException if a security manager is present and it
     *   denies {@link RuntimePermission}
     *   "setDefaultUncaughtExceptionHandler")
     *
     * @see #setUncaughtExceptionHandler
     * @see #getUncaughtExceptionHandler
     * @see ThreadGroup#uncaughtException
     * @since 1.5
     */
    public static void setDefaultUncaughtExceptionHandler(UncaughtExceptionHandler eh) {
        SecurityManager sm = System.getSecurityManager();
        if (sm != null) {
            sm.checPermission(
              new RuntimePermission("setDefaultUncaughtExceptionHandler")
            &

bsp;   );
        }
         defaultUncaughtExceptionHa
dler = eh;
     }
    /**
     * Returns the default handler invoked when a thread abrupt
y terminates
     * due to an uncaught exception. If the returned value is null

     * there is no default.
     * @since 1.5
     * @see #setDefaultUncaughtExceptionHandler
     * @return the default uncaught exception handler for all th
eads
     */
    public static UncaughtExceptionHandler getDefaultUncaughtExcept
onHandler(){
        return defaultUncaughtExceptionHan
ler;
    }
    /**
     * Returns the handler invoked when this thread abruptly te
minates
     * due to an uncaught exception. If this thread has not had
n
     * uncaught exception handler explicitly set then this thread'

     * ThreadGroup object is returned, unless this thread
     * has terminated, in which case null is returned.
     * @since 1.5
     * @return the uncaught exception handler for this thread<
p>
     */
    public UncaughtExceptionHandler getUncaughtExceptionHandler()
{
        return uncaughtExceptionHandler !=
ull ?
            uncaugh
ExceptionHandler : group;
    }
    /**
     * Set the handler invoked when this thread abruptly termina
es
     * due to an uncaught exception.
     * A thread can take full control of how it responds to uncau
ht
     * exceptions by having its uncaught exception handler expli
itly set.
     * If no such handler is set then the thread's ThreadGroup</
>
     * object acts as its handler.
     * @param eh the object to use as this thread's uncaught ex
eption

     * handler. If null then this thread has no explicit handler.</
>
     * @throws SecurityException if the current th
ead is not allowed to
     *   &
nbsp; modify this thread.
     * @see #setDefaultUncaughtExceptionHandler
     * @see ThreadGroup#uncaughtException
     * @since 1.5
     */
    public void setUncaughtExceptionHandler(UncaughtExceptionHan
ler eh) {
        checkAccess();
       uncaughtExceptionHandler = eh;
    }
    /**
     * Dispatch an uncaught exception to the handler. This met
od is
     * intended to be called only by the JVM.
     */
    private void dispatchUncaughtException(Throwable e) {
        getUncaughtExceptionHandler().unca
ghtException(this, e);
    }
    /**
     * Removes from the specified map any keys that have been
enqueued
     * on the specified reference queue.
     */
    static void processQueue(ReferenceQueue> queue,
           &
bsp;   &
bsp;   ConcurrentMapextends
           &
bsp;   &
bsp;   WeakReference>, ?> map)
    {
        Referenceextends Class> ref;
        while((ref = queue.poll()) != null) {</
>
            map.re
ove(ref);
        }
    }
    /**
     * Weak key for Class objects.
     **/
    static class WeakClassKey extends WeakReference> {
        /**
         * saved value of the referent's
dentity hash code, to maintain
         * a consistent hash code after
he referent has been cleared
         */

        private final int hash;
        /**
         * Create a new WeakClassKey o the given object, registered
         * with a queue.
         */
         WeakClassKey(Class cl, ReferenceQue e> refQueue) {
             super(cl, efQueue);
             hash = S stem.identityHashCode(cl);
         }
         /**
          * Returns the identity hash co e of the original referent.
          */
         @Override
         public int hashCode() {
              return h sh;
         }
         /**
          * Returns true if the given obje t is this identical
          * WeakClassKey instance, or, if his object's referent has not
          * been cleared, if the given obj ct is another WeakClassKey
          * instance with the identical n n-null referent as this one.
          */
         @Override
         public boolean equals(Object obj) {</ >
             if (obj = this)
               return true;
              if (obj ins anceof WeakClassKey) {
               Object referent = get();
               return (referent != null) &
                     (referent == ((WeakCla sKey) obj).get());
             } else {< p>
               return false;
            }
        }

    }
    // The following three initially uninitialized fields are exclusively</
>
    // managed by class java.util.concurrent.ThreadLocalRandom. Thes

    // fields are used to build the high-performance PRNGs in the
    // concurrent code, and we can not risk accidental false sharing.</
>
    // Hence, the fields are isolated with @Contended.
    /** The current seed for a ThreadLocalRandom */
    @sun.misc.Contended("tlr")
    long threadLocalRandomSeed;
    /** Probe hash value; nonzero if threadLocalRandomSeed initialize
 */
    @sun.misc.Contended("tlr")
    int threadLocalRandomProbe;
    /** Secondary seed isolated from public ThreadLocalRandom sequ
nce */
    @sun.misc.Contended("tlr")
    int threadLocalRandomSecondarySeed;
    /* Some private helper methods */
    private native void setPriority0(int newPriority);
    private native void stop0(Object o);
    private native void suspend0();
    private native void resume0();
    private native void interrupt0();
    private native void setNativeName(String name);
}