# TODO

一、    jdk源码

本文为转载：http://sunboyyyl.blog.163.com/blog/static/2247381201211531712330/

在Java中每一个线程都归属于某个线程组管理的一员，例如在主函数main()主工作流程中产生一个线程，则产生的线程属于main这个线程组管理的一员。简单地说，线程组就是由线程组成的管理线程的类，这个类是java.lang.ThreadGroup类。

可以通过使用如下代码获取此线程所属线程组的名称。

Thread.currentThread().getThreadGroup().getName();

currentThread()：取得当前线程。

getThreadGroup()：取得当前线程所在的组。

getName()：取得组的名称。

定义一个线程组，通过以下代码可以实现。

ThreadGroup group=new ThreadGroup("group");

Thread thread=new Thread(group,"the first thread of group");

ThreadGroup类中的某些方法，可以对线程组中的线程产生作用。例如，setMaxPriority()方法可以设定线程组中的所有线程拥有最大的优先权。

ThreadGroup

线程组表示一个线程的集合。此外，线程组也可以包含其他线程组。线程组构成一棵树，在树中，除了初始线程组外，每个线程组都有一个父线程组。

允许线程访问有关自己的线程组的信息，但是不允许它访问有关其线程组的父线程组或其他任何线程组的信息。

构造方法摘要

ThreadGroup(String name) // 构造一个新线程组。

ThreadGroup(ThreadGroup parent, String name) // 创建一个新线程组。

如：

ThreadGroup g=new ThreadGroup("g1");

Thread t=new Thread(g,new TestThread(1000,"AAA"));

其中的g即线程所属的线程组

使用线程组的好处是可以对这一组的线程进行整体操作。

其中部分的方法

int activeCount()

返回此线程组中活动线程的估计数。

int activeGroupCount()

返回此线程组中活动线程组的估计数。

String getName()

返回此线程组的名称。

ThreadGroup getParent()

返回此线程组的父线程组。

void resume()

已过时。 此方法只用于联合 Thread.suspend 和 ThreadGroup.suspend 时，因为它们所固有的容易导致死锁的特性，所以两者都已废弃。有关详细信息，请参阅 Thread.suspend()。

void setDaemon(boolean daemon)

更改此线程组的后台程序状态。

void setMaxPriority(int pri)

设置线程组的最高优先级。

void stop()

已过时。

（一）代码

|  |
| --- |
| package java**.**lang**;**    public  class ThreadGroup **implements** Thread**.**UncaughtExceptionHandler **{**      private final ThreadGroup parent**;**      String name**;**      int maxPriority**;**      boolean destroyed**;**      boolean daemon**;**      boolean vmAllowSuspension**;**        int nUnstartedThreads **=** 0**;**      int nthreads**;**      Thread threads**[];**        int ngroups**;**      ThreadGroup groups**[];**        /\*\*这个是系统线程组 ，私有不允许用户声明   \*/      private ThreadGroup**()** **{**     // called from C code  **this.**name **=** "system"**;**  **this.**maxPriority **=** Thread**.**MAX\_PRIORITY**;**  **this.**parent **=** **null;**  **}**        public ThreadGroup**(**String name**)** **{**  **this(**Thread**.**currentThread**().**getThreadGroup**(),** name**);**  **}**          public ThreadGroup**(**ThreadGroup parent**,** String name**)** **{**  **this(**checkParentAccess**(**parent**),** parent**,** name**);**  **}**        private ThreadGroup**(**Void unused**,** ThreadGroup parent**,** String name**)** **{**  **this.**name **=** name**;**  **this.**maxPriority **=** parent**.**maxPriority**;**  **this.**daemon **=** parent**.**daemon**;**  **this.**vmAllowSuspension **=** parent**.**vmAllowSuspension**;**  **this.**parent **=** parent**;**          parent**.**add**(this);**  **}**        /\*       \* @throws  NullPointerException  if the parent argument is {@code null}       \* @throws  SecurityException     if the current thread cannot create a       \*                                thread in the specified thread group.       \*/      private static Void checkParentAccess**(**ThreadGroup parent**)** **{**          parent**.**checkAccess**();**  **return** **null;**  **}**        public final String getName**()** **{**  **return** name**;**  **}**        public final ThreadGroup getParent**()** **{**  **if** **(**parent **!=** **null)**              parent**.**checkAccess**();**  **return** parent**;**  **}**        public final int getMaxPriority**()** **{**  **return** maxPriority**;**  **}**        public final boolean isDaemon**()** **{**  **return** daemon**;**  **}**        public synchronized boolean isDestroyed**()** **{**  **return** destroyed**;**  **}**        public final void setDaemon**(**boolean daemon**)** **{**          checkAccess**();**  **this.**daemon **=** daemon**;**  **}**        public final void setMaxPriority**(**int pri**)** **{**          int ngroupsSnapshot**;**          ThreadGroup**[]** groupsSnapshot**;**          synchronized **(this)** **{**              checkAccess**();**  **if** **(**pri **<** Thread**.**MIN\_PRIORITY **||** pri **>** Thread**.**MAX\_PRIORITY**)** **{**  **return;**  **}**              maxPriority **=** **(**parent **!=** **null)** **?** Math**.**min**(**pri**,** parent**.**maxPriority**)** **:** pri**;**              ngroupsSnapshot **=** ngroups**;**  **if** **(**groups **!=** **null)** **{**                  groupsSnapshot **=** Arrays**.**copyOf**(**groups**,** ngroupsSnapshot**);**  **}** **else** **{**                  groupsSnapshot **=** **null;**  **}**  **}**  **for** **(**int i **=** 0 **;** i **<** ngroupsSnapshot **;** i**++)** **{**              groupsSnapshot**[**i**].**setMaxPriority**(**pri**);**  **}**  **}**        public final boolean parentOf**(**ThreadGroup g**)** **{**  **for** **(;** g **!=** **null** **;** g **=** g**.**parent**)** **{**  **if** **(**g **==** **this)** **{**  **return** **true;**  **}**  **}**  **return** **false;**  **}**        public final void checkAccess**()** **{**          SecurityManager security **=** System**.**getSecurityManager**();**  **if** **(**security **!=** **null)** **{**              security**.**checkAccess**(this);**  **}**  **}**        public int activeCount**()** **{**          int result**;**          // Snapshot sub-group data so we don't hold this lock          // while our children are computing.          int ngroupsSnapshot**;**          ThreadGroup**[]** groupsSnapshot**;**          synchronized **(this)** **{**  **if** **(**destroyed**)** **{**  **return** 0**;**  **}**              result **=** nthreads**;**              ngroupsSnapshot **=** ngroups**;**  **if** **(**groups **!=** **null)** **{**                  groupsSnapshot **=** Arrays**.**copyOf**(**groups**,** ngroupsSnapshot**);**  **}** **else** **{**                  groupsSnapshot **=** **null;**  **}**  **}**  **for** **(**int i **=** 0 **;** i **<** ngroupsSnapshot **;** i**++)** **{**              result **+=** groupsSnapshot**[**i**].**activeCount**();**  **}**  **return** result**;**  **}**          public int enumerate**(**Thread list**[])** **{**          checkAccess**();**  **return** enumerate**(**list**,** 0**,** **true);**  **}**          public int enumerate**(**Thread list**[],** boolean recurse**)** **{**          checkAccess**();**  **return** enumerate**(**list**,** 0**,** recurse**);**  **}**        private int enumerate**(**Thread list**[],** int n**,** boolean recurse**)** **{**          int ngroupsSnapshot **=** 0**;**          ThreadGroup**[]** groupsSnapshot **=** **null;**          synchronized **(this)** **{**  **if** **(**destroyed**)** **{**  **return** 0**;**  **}**              int nt **=** nthreads**;**  **if** **(**nt **>** list**.**length **-** n**)** **{**                  nt **=** list**.**length **-** n**;**  **}**  **for** **(**int i **=** 0**;** i **<** nt**;** i**++)** **{**  **if** **(**threads**[**i**].**isAlive**())** **{**                      list**[**n**++]** **=** threads**[**i**];**  **}**  **}**  **if** **(**recurse**)** **{**                  ngroupsSnapshot **=** ngroups**;**  **if** **(**groups **!=** **null)** **{**                      groupsSnapshot **=** Arrays**.**copyOf**(**groups**,** ngroupsSnapshot**);**  **}** **else** **{**                      groupsSnapshot **=** **null;**  **}**  **}**  **}**  **if** **(**recurse**)** **{**  **for** **(**int i **=** 0 **;** i **<** ngroupsSnapshot **;** i**++)** **{**                  n **=** groupsSnapshot**[**i**].**enumerate**(**list**,** n**,** **true);**  **}**  **}**  **return** n**;**  **}**          public int activeGroupCount**()** **{**          int ngroupsSnapshot**;**          ThreadGroup**[]** groupsSnapshot**;**          synchronized **(this)** **{**  **if** **(**destroyed**)** **{**  **return** 0**;**  **}**              ngroupsSnapshot **=** ngroups**;**  **if** **(**groups **!=** **null)** **{**                  groupsSnapshot **=** Arrays**.**copyOf**(**groups**,** ngroupsSnapshot**);**  **}** **else** **{**                  groupsSnapshot **=** **null;**  **}**  **}**          int n **=** ngroupsSnapshot**;**  **for** **(**int i **=** 0 **;** i **<** ngroupsSnapshot **;** i**++)** **{**              n **+=** groupsSnapshot**[**i**].**activeGroupCount**();**  **}**  **return** n**;**  **}**          public int enumerate**(**ThreadGroup list**[])** **{**          checkAccess**();**  **return** enumerate**(**list**,** 0**,** **true);**  **}**          public int enumerate**(**ThreadGroup list**[],** boolean recurse**)** **{**          checkAccess**();**  **return** enumerate**(**list**,** 0**,** recurse**);**  **}**        private int enumerate**(**ThreadGroup list**[],** int n**,** boolean recurse**)** **{**          int ngroupsSnapshot **=** 0**;**          ThreadGroup**[]** groupsSnapshot **=** **null;**          synchronized **(this)** **{**  **if** **(**destroyed**)** **{**  **return** 0**;**  **}**              int ng **=** ngroups**;**  **if** **(**ng **>** list**.**length **-** n**)** **{**                  ng **=** list**.**length **-** n**;**  **}**  **if** **(**ng **>** 0**)** **{**                  System**.**arraycopy**(**groups**,** 0**,** list**,** n**,** ng**);**                  n **+=** ng**;**  **}**  **if** **(**recurse**)** **{**                  ngroupsSnapshot **=** ngroups**;**  **if** **(**groups **!=** **null)** **{**                      groupsSnapshot **=** Arrays**.**copyOf**(**groups**,** ngroupsSnapshot**);**  **}** **else** **{**                      groupsSnapshot **=** **null;**  **}**  **}**  **}**  **if** **(**recurse**)** **{**  **for** **(**int i **=** 0 **;** i **<** ngroupsSnapshot **;** i**++)** **{**                  n **=** groupsSnapshot**[**i**].**enumerate**(**list**,** n**,** **true);**  **}**  **}**  **return** n**;**  **}**          public final void interrupt**()** **{**          int ngroupsSnapshot**;**          ThreadGroup**[]** groupsSnapshot**;**          synchronized **(this)** **{**              checkAccess**();**  **for** **(**int i **=** 0 **;** i **<** nthreads **;** i**++)** **{**                  threads**[**i**].**interrupt**();**  **}**              ngroupsSnapshot **=** ngroups**;**  **if** **(**groups **!=** **null)** **{**                  groupsSnapshot **=** Arrays**.**copyOf**(**groups**,** ngroupsSnapshot**);**  **}** **else** **{**                  groupsSnapshot **=** **null;**  **}**  **}**  **for** **(**int i **=** 0 **;** i **<** ngroupsSnapshot **;** i**++)** **{**              groupsSnapshot**[**i**].**interrupt**();**  **}**  **}**            private boolean stopOrSuspend**(**boolean suspend**)** **{**          boolean suicide **=** **false;**          Thread us **=** Thread**.**currentThread**();**          int ngroupsSnapshot**;**          ThreadGroup**[]** groupsSnapshot **=** **null;**          synchronized **(this)** **{**              checkAccess**();**  **for** **(**int i **=** 0 **;** i **<** nthreads **;** i**++)** **{**  **if** **(**threads**[**i**]==**us**)**                      suicide **=** **true;**  **else** **if** **(**suspend**)**                      threads**[**i**].**suspend**();**  **else**                      threads**[**i**].**stop**();**  **}**                ngroupsSnapshot **=** ngroups**;**  **if** **(**groups **!=** **null)** **{**                  groupsSnapshot **=** Arrays**.**copyOf**(**groups**,** ngroupsSnapshot**);**  **}**  **}**  **for** **(**int i **=** 0 **;** i **<** ngroupsSnapshot **;** i**++)**              suicide **=** groupsSnapshot**[**i**].**stopOrSuspend**(**suspend**)** **||** suicide**;**    **return** suicide**;**  **}**        public final void destroy**()** **{**          int ngroupsSnapshot**;**          ThreadGroup**[]** groupsSnapshot**;**          synchronized **(this)** **{**              checkAccess**();**  **if** **(**destroyed **||** **(**nthreads **>** 0**))** **{**  **throw** **new** IllegalThreadStateException**();**  **}**              ngroupsSnapshot **=** ngroups**;**  **if** **(**groups **!=** **null)** **{**                  groupsSnapshot **=** Arrays**.**copyOf**(**groups**,** ngroupsSnapshot**);**  **}** **else** **{**                  groupsSnapshot **=** **null;**  **}**  **if** **(**parent **!=** **null)** **{**                  destroyed **=** **true;**                  ngroups **=** 0**;**                  groups **=** **null;**                  nthreads **=** 0**;**                  threads **=** **null;**  **}**  **}**  **for** **(**int i **=** 0 **;** i **<** ngroupsSnapshot **;** i **+=** 1**)** **{**              groupsSnapshot**[**i**].**destroy**();**  **}**  **if** **(**parent **!=** **null)** **{**              parent**.**remove**(this);**  **}**  **}**        /\*\*       \* Adds the specified Thread group to this group.       \* @param g the specified Thread group to be added       \* @exception IllegalThreadStateException If the Thread group has been destroyed.       \*/      private final void add**(**ThreadGroup g**){**          synchronized **(this)** **{**  **if** **(**destroyed**)** **{**  **throw** **new** IllegalThreadStateException**();**  **}**  **if** **(**groups **==** **null)** **{**                  groups **=** **new** ThreadGroup**[**4**];**  **}** **else** **if** **(**ngroups **==** groups**.**length**)** **{**                  groups **=** Arrays**.**copyOf**(**groups**,** ngroups **\*** 2**);**  **}**              groups**[**ngroups**]** **=** g**;**                // This is done last so it doesn't matter in case the              // thread is killed              ngroups**++;**  **}**  **}**        private void remove**(**ThreadGroup g**)** **{**          synchronized **(this)** **{**  **if** **(**destroyed**)** **{**  **return;**  **}**  **for** **(**int i **=** 0 **;** i **<** ngroups **;** i**++)** **{**  **if** **(**groups**[**i**]** **==** g**)** **{**                      ngroups **-=** 1**;**                      System**.**arraycopy**(**groups**,** i **+** 1**,** groups**,** i**,** ngroups **-** i**);**                      // Zap dangling reference to the dead group so that                      // the garbage collector will collect it.                      groups**[**ngroups**]** **=** **null;**  **break;**  **}**  **}**  **if** **(**nthreads **==** 0**)** **{**                  notifyAll**();**  **}**  **if** **(**daemon **&&** **(**nthreads **==** 0**)** **&&**  **(**nUnstartedThreads **==** 0**)** **&&** **(**ngroups **==** 0**))**  **{**                  destroy**();**  **}**  **}**  **}**        void addUnstarted**()** **{**          synchronized**(this)** **{**  **if** **(**destroyed**)** **{**  **throw** **new** IllegalThreadStateException**();**  **}**              nUnstartedThreads**++;**  **}**  **}**        void add**(**Thread t**)** **{**          synchronized **(this)** **{**  **if** **(**destroyed**)** **{**  **throw** **new** IllegalThreadStateException**();**  **}**  **if** **(**threads **==** **null)** **{**                  threads **=** **new** Thread**[**4**];**  **}** **else** **if** **(**nthreads **==** threads**.**length**)** **{**                  threads **=** Arrays**.**copyOf**(**threads**,** nthreads **\*** 2**);**  **}**              threads**[**nthreads**]** **=** t**;**                // This is done last so it doesn't matter in case the              // thread is killed              nthreads**++;**                nUnstartedThreads**--;**  **}**  **}**        void threadStartFailed**(**Thread t**)** **{**          synchronized**(this)** **{**              remove**(**t**);**              nUnstartedThreads**++;**  **}**  **}**        void threadTerminated**(**Thread t**)** **{**          synchronized **(this)** **{**              remove**(**t**);**    **if** **(**nthreads **==** 0**)** **{**                  notifyAll**();**  **}**  **if** **(**daemon **&&** **(**nthreads **==** 0**)** **&&**  **(**nUnstartedThreads **==** 0**)** **&&** **(**ngroups **==** 0**))**  **{**                  destroy**();**  **}**  **}**  **}**      private void remove**(**Thread t**)** **{**          synchronized **(this)** **{**  **if** **(**destroyed**)** **{**  **return;**  **}**  **for** **(**int i **=** 0 **;** i **<** nthreads **;** i**++)** **{**  **if** **(**threads**[**i**]** **==** t**)** **{**                      System**.**arraycopy**(**threads**,** i **+** 1**,** threads**,** i**,** **--**nthreads **-** i**);**                      // Zap dangling reference to the dead thread so that                      // the garbage collector will collect it.                      threads**[**nthreads**]** **=** **null;**  **break;**  **}**  **}**  **}**  **}**        public void list**()** **{**          list**(**System**.**out**,** 0**);**  **}**      void list**(**PrintStream out**,** int indent**)** **{**          int ngroupsSnapshot**;**          ThreadGroup**[]** groupsSnapshot**;**          synchronized **(this)** **{**  **for** **(**int j **=** 0 **;** j **<** indent **;** j**++)** **{**                  out**.**print**(**" "**);**  **}**              out**.**println**(this);**              indent **+=** 4**;**  **for** **(**int i **=** 0 **;** i **<** nthreads **;** i**++)** **{**  **for** **(**int j **=** 0 **;** j **<** indent **;** j**++)** **{**                      out**.**print**(**" "**);**  **}**                  out**.**println**(**threads**[**i**]);**  **}**              ngroupsSnapshot **=** ngroups**;**  **if** **(**groups **!=** **null)** **{**                  groupsSnapshot **=** Arrays**.**copyOf**(**groups**,** ngroupsSnapshot**);**  **}** **else** **{**                  groupsSnapshot **=** **null;**  **}**  **}**  **for** **(**int i **=** 0 **;** i **<** ngroupsSnapshot **;** i**++)** **{**              groupsSnapshot**[**i**].**list**(**out**,** indent**);**  **}**  **}**          public void uncaughtException**(**Thread t**,** Throwable e**)** **{**  **if** **(**parent **!=** **null)** **{**              parent**.**uncaughtException**(**t**,** e**);**  **}** **else** **{**              Thread**.**UncaughtExceptionHandler ueh **=**                  Thread**.**getDefaultUncaughtExceptionHandler**();**  **if** **(**ueh **!=** **null)** **{**                  ueh**.**uncaughtException**(**t**,** e**);**  **}** **else** **if** **(!(**e **instanceof** ThreadDeath**))** **{**                  System**.**err**.**print**(**"Exception in thread \""  **+** t**.**getName**()** **+** "\" "**);**                  e**.**printStackTrace**(**System**.**err**);**  **}**  **}**  **}**        public String toString**()** **{**  **return** getClass**().**getName**()** **+** "[name=" **+** getName**()** **+** ",maxpri=" **+** maxPriority **+** "]"**;**  **}**  **}**  a |

二、测试代码

（一）源码

|  |
| --- |
| package myjava**.**lang**;**    **import** java**.**util**.**Arrays**;**  /\*  main.thread.name:{},main.threadGroup.name:{}mainmain  thread1.name:{},thread1.threadGroup.name:{}Thread-1main  threa1.name:{},threa1.isDaemon:{},threa1.priority:{}Thread-1false5  curThread.name:{},curThread.isDaemon:{},curThread.priority:{}mainfalse5  threa2.name:{}Thread-0  group1.parent.name:{}main  mainGroup.parentOf(group1):{},group1.parentOf(group1),{}truetrue  mainGroup.parent.name:{}system  mainGroup.activeCount:{}2  MyThread[MyThread-1] task begin.  mainGroupActiveThreads:{}[Thread[main,5,main], Thread[MyThread-1,5,main]]  group1.activeCount:{}1  mainGroup.activeCount:{}3  MyThread[MyThread-2] task begin.  mainGroupActiveThreads2:{}[Thread[main,5,main], Thread[MyThread-1,5,main], null]  mainGroupActiveThreads3:{}[Thread[main,5,main], Thread[MyThread-1,5,main], Thread[MyThread-2,5,ThreadGroup-1]]  get1 == get1Ref:{}true  MyThread[MyThread-1] was interrutped  MyThread[MyThread-1] task end.  threadName:{},state:{},isAlive:{}mainRUNNABLEtrue  threadName:{},state:{},isAlive:{}MyThread-1TERMINATEDfalse  threadName:{},state:{},isAlive:{}MyThread-2TIMED\_WAITINGtrue  MyThread[MyThread-2] was interrutped  MyThread[MyThread-2] task end.  myjava.lang.ThreadGroupTest$MyThreadGroup[name=MyThreadGroup,maxpri=10]      Thread[A-Exception-Thread,5,MyThreadGroup]  uncaughtException.thread.name.{}A-Exception-Threadjava.lang.NullPointerException  java.lang.ThreadGroup[name=main,maxpri=10]      uncaughtException.thread.state.{}RUNNABLE  Thread[main,5,main]      java.lang.ThreadGroup[name=ThreadGroup-1,maxpri=10]      myjava.lang.ThreadGroupTest$MyThreadGroup[name=MyThreadGroup,maxpri=10]          Thread[A-Exception-Thread,5,MyThreadGroup]  MyThread[A-Exception-Thread] task begin.  MyThread[A-Exception-Thread] task end.    \*/        /\*\*   \*   \* {@link java.lang.ThreadGroup}示例   \*   \* <pre>   \* 1.线程组表示一个线程的结合.此外线程组也可以包含其他线程组.线程组构成一棵树,在树中,除了初始线程组外,每个线程组都有一个父线程组   \* 2.每个线程产生时,都会被归入某个线程组(Java中每个线程都是属于某个线程组),视线程是在那个线程组中产生而定.如果没有指定,则归入产生该子线程的线程的线程组中.(如在main中初始化一个线程,未指定线程组,则线程所属线程组为main)   \* 3.线程一旦归入某个组就无法更换组   \* 4.main线程组的parent是system线程组,而system线程组的parent为null.(参考ThreadGroup的私有构造方法).也就是说初始线程组为system.以system/main衍生出一颗树   \* 5.其activeCount/activeCount/enumerate方法均为不精确的统计,建议仅用于信息目的   \* 6.可通过enumerate获得活动线程的引用并对其进行操作   \* 7.允许线程访问有关自己的线程组的信息{@link Thread#getThreadGroup()},但是不允许它访问有关其线程组的父线程组或其他任何线程组的信息   \* 8.线程组的某些方法,将对线程组机器所有子组的所有线程执行,如{@link ThreadGroup#interrupt()}   \* 7.public class ThreadGroup implements Thread.UncaughtExceptionHandler,即其实现了UncaughtExceptionHandler方法.即   \*     ->当Thread因未捕获的异常而突然中止时,调用处理程序的接口.当某一线程因捕获的异常而即将中止时,JVM将使用UncaughtExceptionHandler查询该线程以获得其   \*     UncaughtExceptionHandler的线程并调用处理程序的uncaughtException方法,将线程和异常作为参数传递.如果某一线程为明确设置其UncaughtExceptionHandler,   \*     则将它的ThreadGroup对象作为UncaughtExceptionHandler.如果ThreadGroup对象对处理异常没有特殊要求,可以将调用转发给   \*  {@link Thread#getDefaultUncaughtExceptionHandler()}   \* </pre>   \*   \* <pre>   \* 1."线程是独立执行的代码片断，线程的问题应该由线程自己来解决，而不要委托到外部".基于这样的设计理念，在Java中，线程方法的异常   \* （无论是checked还是unchecked exception），都应该在线程代码边界之内（run方法内）进行try catch并处理掉   \* .换句话说，我们不能捕获从线程中逃逸的异常   \*   \* 2.参考{@link Thread#dispatchUncaughtException}，该方法是一个私有方法，在异常逃逸时调用.判断线程自身是否设置了uncaughtExceptionHandler.   \* 如果没有则直接返回group，即自己的所在的线程组,而线程组实现了UncaughtExceptionHandler接口.{@link Thread#getUncaughtExceptionHandler()}   \* </pre>   \*   \* @author landon   \*   \*/  public class ThreadGroupTest **{**     // private static final Logger LOGGER = new Logger(ThreadGroupTest.class);        public static void main**(**String**[]** args**)** **{**          // main.thread.name:main,main.threadGroup.name:main          // 即当前线程为main，其所属线程组          // Thread#public final ThreadGroup getThreadGroup()          // 返回该线程所属的线程组.如果该线程已经停止运行则返回null          System**.**out**.**println**(**"main.thread.name:{},main.threadGroup.name:{}" **+**  Thread  **.**currentThread**().**getName**()+** Thread**.**currentThread**()**  **.**getThreadGroup**().**getName**());**            Thread thread1 **=** **new** Thread**(**"Thread-1"**);**          // 从输出可以看到          // 1.thread1为指定线程组,则其输出的线程组为main，即产生thread1的main线程所属的线程组main          // 2.thread1并未启动.其所属线程组就已经指定了,即在线程初始化的时候就已经拥有了          // 从源码看:          // Thread#private void init(ThreadGroup g, Runnable target,String          // name,long stackSize)          // 1.Thread parent = currentThread(),获取调用线程,即产生thread的线程          // 2.判断SecurityManager是否为空,如果不为空,则用SecurityManager#getThreadGroup().          // 3.如果SecurityManager#getThreadGroup()为空或者不存在SecurityManager，则线程组赋值为parent.getThreadGroup()，即调用线程所属的线程组          // 另外从Thread的初始化方法中可以看到,线程是否是守护线程以及线程的优先级均是由parent指定          System**.**out**.**println**(**"thread1.name:{},thread1.threadGroup.name:{}" **+**                  thread1**.**getName**()+**thread1**.**getThreadGroup**().**getName**());**          // 从是否是守护线程和线程的优先级的输出来看,threa1和其parent线程main的是一样的          System**.**out**.**println**(**"threa1.name:{},threa1.isDaemon:{},threa1.priority:{}" **+**                  thread1**.**getName**()+** thread1**.**isDaemon**()+**thread1**.**getPriority**());**          Thread curThread **=** Thread**.**currentThread**();**          System**.**out**.**println**(**                  "curThread.name:{},curThread.isDaemon:{},curThread.priority:{}" **+**                  curThread**.**getName**()** **+** curThread**.**isDaemon**()** **+**                  curThread**.**getPriority**());**            // 该线程未执行名字,从输出看:其名字是Thread-0          // 从Thread的初始化方法可以看到,当名字为null的时候（即匿名线程）,会指定一个名字"Thread-" +          // nextThreadNum()          // 而nextThreadNum是一个静态同步方法,对threadInitNumber这个静态计数器执行++          Thread thread2 **=** **new** Thread**();**          System**.**out**.**println**(**"threa2.name:{}" **+**  thread2**.**getName**());**            // public ThreadGroup(String name) 构造一个新的线程组,该新线程组的父线程组是目前调用线程的线程组          // 从源码上看:          // 1.this(Thread.currentThread().getThreadGroup(),          // name),即将当前调用线程所属的线程组作为其父parent线程组传入          // 2.将parent.maxPriority/daemon赋值传入          // 3.parent.add(this),将将该线程组加入到父线程组          // 4.另外ThreadGroup有一个私有的空参数的构造方法,其指定线程组名字为system,priority为最大优先级,parent为null，即没有父线程组.从该代码的注释来看,          // ->该私有构造方法通常用来创建一个系统线程组,C代码调用          ThreadGroup group1 **=** **new** ThreadGroup**(**"ThreadGroup-1"**);**          // 输出:group1.parent.name:main,即group1的父线程组为main          // ThreadGroup#public final ThreadGroup getParent() 返回线程组的父线程组          System**.**out**.**println**(**"group1.parent.name:{}" **+**  group1**.**getParent**().**getName**());**          ThreadGroup mainGroup **=** Thread**.**currentThread**().**getThreadGroup**();**          // 输出:mainGroup.parentOf(group1):true,group1.parentOf(group1),true          // ThreadGroup#public final boolean parentOf(ThreadGroup g)          // 判断该线程组是否为线程组参数或者是其祖先线程组          // 从输出看,mainGroup是group1的parent,而传入原组,方法也返回true          System**.**out**.**println**(**                  "mainGroup.parentOf(group1):{},group1.parentOf(group1),{}" **+**                  mainGroup**.**parentOf**(**group1**)** **+** group1**.**parentOf**(**group1**));**          // 从输出看mainGroup的父线程组为system.参考ThreadGroup的私有构造方法,可知system是一个系统线程组，其parent为null.即为初始线程组          System**.**out**.**println**(**"mainGroup.parent.name:{}" **+**  mainGroup**.**getParent**()**  **.**getName**());**            MyThread get1 **=** **new** MyThread**(**"MyThread-1"**);**          // 从Thread#start源码看，在启动线程后->会调用group.add(this),即只有在线程启动后,将才会将线程加入其所属线程组          // 另外注意ThreadGroup#void add(Thread          // t),即该方法只有默认访问权限,即包访问权限.所以应用程序无法调用该方法除非jdk lang包内的库          get1**.**start**();**          // ThreadGroup#public int activeCount()          // 返回此线程组中活动线程的估计数,结果并不能反映并发活动(因为多线程并发运行,所以不是很精确.多线程的不确定性,如add(某一新增线程启动)/remove(某一现有线程销毁)).          // ->固有的不精确性->建议只用于信息          // 从源码的实现看,其计算数目只是取了一个groupsSnapshot(syncrhonized)，即当前的快照          System**.**out**.**println**(**"mainGroup.activeCount:{}" **+**  get1**.**getThreadGroup**()**  **.**activeCount**());**          Thread**[]** mainGroupActiveThreads **=** **new** Thread**[**mainGroup**.**activeCount**()];**          // ThreadGroup#public int enumerate(Thread list[])          // 将此线程组即其子组中的所有活动线程复制到指定数组中.注应用程序可用使用activeCount方法获取数组大小的估计数.          // 如果数组太小而无法保持所有线程,则忽略额外的线程          // 可额外校验该方法的返回值是否严格小于参数list的长度          // 因为此方法固有的竞争条件(源码实现也是取了一个Snapshot(syncrhonized)),建议仅用于信息目的          mainGroup**.**enumerate**(**mainGroupActiveThreads**);**          // 从输出看:1.主线程组包括两个活动线程,main和MyThread-1          // 2.Thread#toString方法返回的信息是Thread[threadName,priority,groupName(如果其group不为null)]          System**.**out**.**println**(**"mainGroupActiveThreads:{}" **+**                  Arrays**.**toString**(**mainGroupActiveThreads**));**            // Thread#public Thread(ThreadGroup group, String name) 指定线程组          MyThread get2 **=** **new** MyThread**(**group1**,**                  "MyThread-2"**);**          // 通过调用start将get2加入group1          get2**.**start**();**            // 输出:group1.activeCount:1          System**.**out**.**println**(**"group1.activeCount:{}" **+**  group1**.**activeCount**());**          // 输出:mainGroup.activeCount:3,即其统计包括子线程的活动线程数目,因为group1为其子组          System**.**out**.**println**(**"mainGroup.activeCount:{}" **+**  mainGroup**.**activeCount**());**            Thread**[]** mainGroupActiveThreads2 **=** **new** Thread**[**mainGroup**.**activeCount**()];**          // ThreadGroup#public int enumerate(Thread list[], boolean recurse)          // recurse为递归的意思          // 如果recurse为true表示要复制遍历子线程组的活动线程,否则只是复制当前线程组的活动线程          // enumerate(Thread list[])方法默认传true          mainGroup**.**enumerate**(**mainGroupActiveThreads2**,** **false);**          // 输出:[Thread[main,5,main], Thread[MyThread-1,5,main], null]          // 从输出可以看,只包括主线程组自身的活动线程          System**.**out**.**println**(**"mainGroupActiveThreads2:{}" **+**                  Arrays**.**toString**(**mainGroupActiveThreads2**));**            Thread**[]** mainGroupActiveThreads3 **=** **new** Thread**[**mainGroup**.**activeCount**()];**          mainGroup**.**enumerate**(**mainGroupActiveThreads3**,** **true);**          // 输出:Thread[MyThread-1,5,main],          // Thread[MyThread-2,5,ThreadGroup-1]]          // 从输出可以看,包含了主线程组的子线程组group1的活动线程          System**.**out**.**println**(**"mainGroupActiveThreads3:{}" **+**                  Arrays**.**toString**(**mainGroupActiveThreads3**));**            // 通过enumerate方法得到活动线程的引用,我们可以对其进行操作          Thread get1Ref **=** mainGroupActiveThreads3**[**1**];**          System**.**out**.**println**(**"get1 == get1Ref:{}" **+**  **(**get1 **==** get1Ref**));**          // 中断get1,从输出看,get1线程任务确实被打断了          get1Ref**.**interrupt**();**            // 可以遍历活动线程列表，进行操作如获取线程状态，判断是否处于活动状态等  **for** **(**Thread tRef **:** mainGroupActiveThreads3**)** **{**              System**.**out**.**println**(**"threadName:{},state:{},isAlive:{}" **+**  tRef**.**getName**()** **+**                      tRef**.**getState**()+**tRef**.**isAlive**());**  **}**            // ThreadGroup#public final void interrupt() 对线程组及其子组中的所有线程调用interrupt方法          // 从输出可以看到get2也被中断了          mainGroup**.**interrupt**();**            // 自定义的一个group,覆写了uncaughtException方法i          MyThreadGroup etg **=** **new** MyThreadGroup**(**"MyThreadGroup"**);**          AExceptionThread aet **=** **new** AExceptionThread**(**etg**,** "A-Exception-Thread"**);**          aet**.**start**();**            // ThreadGroup#public void list() 将有关此线程组的信息打印的标准输出.此方法只对调试有用          // 包括此线程组的信息:className[name=groupName,maxPri=],还包括当前组内线程的信息,{@link          // Thread#toString()}.{@link ThreadGroup#toString()}          // 然后迭代子线程组进行输出          etg**.**list**();**            mainGroup**.**list**();**  **}**        /\*\*       \*       \* MyThread       \*       \* @author landon       \*       \*/      private static class MyThread **extends** Thread **{**          public MyThread**(**String name**)** **{**  **super(**name**);**  **}**            // 此构造方法指定了线程组          public MyThread**(**ThreadGroup tg**,** String name**)** **{**  **super(**tg**,** name**);**  **}**            @Override          public void run**()** **{**              System**.**out**.**println**(**"MyThread" **+** "[" **+** getName**()** **+** "]"  **+** " task begin."**);**                // 用sleep模拟任务耗时  **try** **{**                  sleep**(**5 **\*** 1000**);**  **}** **catch** **(**InterruptedException e**)** **{**                  System**.**out**.**println**(**"MyThread" **+** "[" **+** getName**()** **+** "]"  **+** " was interrutped"**);**  **}**                System**.**out**.**println**(**"MyThread" **+** "[" **+** getName**()** **+** "]"  **+** " task end."**);**  **}**  **}**        // 自实现的一个线程组,覆写了uncaughtException      private static class MyThreadGroup **extends** ThreadGroup **{**            public MyThreadGroup**(**String name**)** **{**  **super(**name**);**  **}**            @Override          public void uncaughtException**(**Thread t**,** Throwable e**)** **{**              // 注.该方法调用是在t线程,参考Thread#dispatchUncaughtException,是一个私有方法              System**.**out**.**println**(**"uncaughtException.thread.name.{}"**+**t**.**getName**()+** e**);**              System**.**out**.**println**(**"uncaughtException.thread.state.{}"**+** t**.**getState**());**                // 这里再重新启动一个新线程              MyThread thread **=** **new** MyThread**(this,**                      t**.**getName**());**              thread**.**start**();**  **}**  **}**        private static class AExceptionThread **extends** Thread **{**          public AExceptionThread**(**ThreadGroup group**,** String name**)** **{**  **super(**group**,** name**);**  **}**            @Override          public void run**()** **{**              // 直接抛出一个空指针异常  **throw** **new** NullPointerException**();**  **}**  **}**  **}** |

三、参考

1.  ThreadGroup自带示例代码

<http://www.blogjava.net/landon/archive/2013/12/18/407744.html>

# REF