ExecutorService线程池 ExecutorService 建立多线程的步骤：

1。定义线程类

class Handler implements Runnable{

}

2。建立ExecutorService线程池

ExecutorService executorService = Executors.newCachedThreadPool();

或者

int cpuNums = Runtime.getRuntime().availableProcessors();

//获取当前系统的CPU 数目

ExecutorService executorService =Executors.newFixedThreadPool(cpuNums \* POOL\_SIZE);

//ExecutorService通常根据系统资源情况灵活定义线程池大小

3。调用线程池操作

循环操作，成为daemon,把新实例放入Executor池中

while(true){

executorService.execute(new Handler(socket));

// class Handler implements Runnable{

或者

executorService.execute(createTask(i));

//private static Runnable createTask(final int taskID)

}

execute(Runnable对象)方法

其实就是对Runnable对象调用start()方法

（当然还有一些其他后台动作，比如队列，优先级，IDLE timeout，active激活等）

几种不同的ExecutorService线程池对象

1.newCachedThreadPool()

-缓存型池子，先查看池中有没有以前建立的线程，如果有，就reuse.如果没有，就建一个新的线程加入池中

-缓存型池子通常用于执行一些生存期很短的异步型任务

因此在一些面向连接的daemon型SERVER中用得不多。

-能reuse的线程，必须是timeout IDLE内的池中线程，缺省timeout是60s,超过这个IDLE时长，线程实例将被终止及移出池。

注意，放入CachedThreadPool的线程不必担心其结束，超过TIMEOUT不活动，其会自动被终止。

2. newFixedThreadPool

-newFixedThreadPool与cacheThreadPool差不多，也是能reuse就用，但不能随时建新的线程

-其独特之处:任意时间点，最多只能有固定数目的活动线程存在，此时如果有新的线程要建立，只能放在另外的队列中等待，直到当前的线程中某个线程终止直接被移出池子

- 和cacheThreadPool不同，FixedThreadPool没有IDLE机制（可能也有，但既然文档没提，肯定非常长，类似依赖上层的TCP 或UDP IDLE机制之类的），所以FixedThreadPool多数针对一些很稳定很固定的正规并发线程，多用于服务器

-从方法的源代码看，cache池和fixed 池调用的是同一个底层池，只不过参数不同:

fixed池线程数固定，并且是0秒IDLE（无IDLE）

cache池线程数支持0-Integer.MAX\_VALUE(显然完全没考虑主机的资源承受能力），60秒IDLE

3.ScheduledThreadPool

-调度型线程池

-这个池子里的线程可以按schedule依次delay执行，或周期执行

4.SingleThreadExecutor

-单例线程，任意时间池中只能有一个线程

-用的是和cache池和fixed池相同的底层池，但线程数目是1-1,0秒IDLE（无IDLE）

上面四种线程池，都使用Executor的缺省线程工厂建立线程，也可单独定义自己的线程工厂

下面是缺省线程工厂代码:

static class DefaultThreadFactory implements ThreadFactory {

static final AtomicInteger poolNumber = new AtomicInteger(1);

final ThreadGroup group;

final AtomicInteger threadNumber = new AtomicInteger(1);

final String namePrefix;

DefaultThreadFactory() {

SecurityManager s = System.getSecurityManager();

group = (s != null) ? s.getThreadGroup() : Thread.currentThread().getThreadGroup();

namePrefix = "pool-" + poolNumber.getAndIncrement() + "-thread-";

}

public Thread newThread(Runnable r) {

Thread t = new Thread(group, r, namePrefix + threadNumber.getAndIncrement(), 0);

if (t.isDaemon()) t.setDaemon(false);

if (t.getPriority() != Thread.NORM\_PRIORITY) t.setPriority(Thread.NORM\_PRIORITY);

return t;

}

}

也可自己定义ThreadFactory，加入建立池的参数中

public static ExecutorService newCachedThreadPool(ThreadFactory threadFactory) {

Executor的execute()方法

execute() 方法将Runnable实例加入pool中,并进行一些pool size计算和优先级处理

execute() 方法本身在Executor接口中定义,有多个实现类都定义了不同的execute()方法

如ThreadPoolExecutor类（cache,fiexed,single三种池子都是调用它）的execute方法如下：

public void execute(Runnable command) {

if (command == null) throw new NullPointerException();

if (poolSize >= corePoolSize || !addIfUnderCorePoolSize(command)) {

if (runState == RUNNING && workQueue.offer(command)) {

if (runState != RUNNING || poolSize == 0) ensureQueuedTaskHandled(command);

} else if (!addIfUnderMaximumPoolSize(command)){

reject(command); // is shutdown or saturated

}

}

}