ScheduledThreadPool的实现原理

**在**ScheduledThreadPoolExecutor中有两个比较重要的子类：

一是延迟队列**DelayedWorkQueue**；

一是定时任务**ScheduledFutureTask**；

ScheduledThreadPoolExecutor中没有直接使用DelayQueue，而是在内部实现了一个DelayedWorkQueue；DelayQueue是基于PriorityQueue实现的。

# Delay接口

java.util.concurrent.Delayed

public interface **Delayed** extends **Comparable<Delayed>**

A mix-in style interface for marking objects that should be acted upon after a given delay.

An implementation of this interface must define a **compareTo** method that provides **an ordering consistent with its getDelay method**.

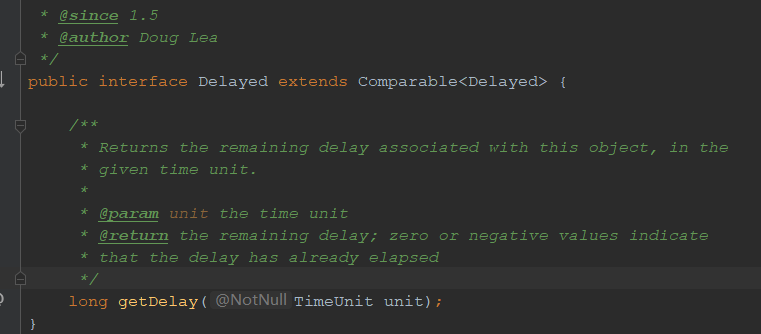
Since:1.5

long getDelay(TimeUnit unit)

Returns **the remaining delay** associated with this object, in the given time unit.

Delayed接口继承了Comparable<Delayed>接口，Comparable接口中定义了唯一的compareTo方法，Delayed定义了唯一的getDelay方法。因此要实现Delayed接口，需要实现compareTo和getDelay方法。

**getDelay方法**:返回剩下的延迟时间，单位通过参数TimeUnit传递进去。返回0或负值，表示延时时间已经过去。



# ScheduledThreadPoolExecutor

## getDelay实现：

public long **getDelay**(TimeUnit unit) {

return unit.convert(time - now(), **NANOSECONDS**);

}

预定的时间time减去当前时间，若大于0，表示时间未到，返回还需要延迟的时间长度；

若等于0或小于0，表示已经到达或已经超过。

## compareTo方法

public int compareTo(Delayed other) {

if (other == this) // **compare zero if same object**

return 0;

if (other instanceof ScheduledFutureTask) {

ScheduledFutureTask<?> x = (ScheduledFutureTask<?>)other;

**long diff = time - x.time;**

if (diff < 0)

return -1;

else if (diff > 0)

return 1;

**else if (sequenceNumber < x.sequenceNumber)**

**return -1;**

**else**

**return 1;**

}

**long diff = getDelay(NANOSECONDS) - other.getDelay(NANOSECONDS);**

**return (diff < 0) ? -1 : (diff > 0) ? 1 : 0;**

}

这里基于两个long参数比较大小：

首先基于time，当前对象的time大，则返回1；若小，则返回-1；

**若time相等，则比较sequenceNumber，若小返回-1，若大则返回1**；

如果此时time和sequenceNumber都相等，则通过getDelay方法返回纳秒long值，进行判断比较。

# ScheduledThreadPoolExecutor

ScheduledThreadPoolExecutor中定义了一个内部类：**ScheduledFutureTask**

private class **ScheduledFutureTask**<V>

extends FutureTask<V> implements RunnableScheduledFuture<V>

其中有3个long型变量比较重要：

private final long **sequenceNumber**;

private long **time**;

private final long **period**;

对应的构造方法：

*/\*\*  
 \* Creates a periodic action with given nano time and period.  
 \*/*ScheduledFutureTask(Runnable r, V result, long ns, long period) {  
 super(r, result);  
 this.time = ns;  
 this.period = period;  
 this.sequenceNumber = *sequencer*.getAndIncrement();  
}