**密级：**

**文档编号：**

**版本号：V1.0**

# Task模型说明

**北京华夏电通科技股份有限公司**

|  |  |
| --- | --- |
| 编制： | 生效日期： |
| 审核： | 批准： |

---------------------------------------------------------------------

北京华夏电通科技股份有限公司对本文件资料享受著作权及其它专属权利，未经书面许可，不得将该等文件资料（其全部或任何部分）披露予任何第三方，或进行修改后使用。

文件更改摘要：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **日期** | **版本号** | **修订说明** | **修订人** | **审核人** | **批准人** |
| 2013-02-26 |  |  | 范仁亮 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Task模型整体结构



图（一）Task模型整体结构

Task模型分两层，第一层为平台无关基础库，第二层为Task模型。平台无关基础库包含了编程过程中将用到的常用技术。目前有线程部分，容器部分，文件部分，网络部分。Task模型使用基础库提供的功能实现了任务模型。所谓任务模型就是当任意事件发生时，有个任务来处理它。

## 线程部分：

### ZOSThread类：

1 ZOSThread是对线程操作的一个封装，方便线程的使用。

2 ZOSThread的\_Entry函数是不断循环运行的，所以如果未调用Stop函数，则OnThreadEntry函数会被不断循环调用。

### 互斥相关类：

1 ZOSMutex是互斥对象的封装。

2 ZOSMutexLocker是帮助更方便的使用ZOSMutex，一定程度上降低锁未释放的风险。

### 信号量相关类：

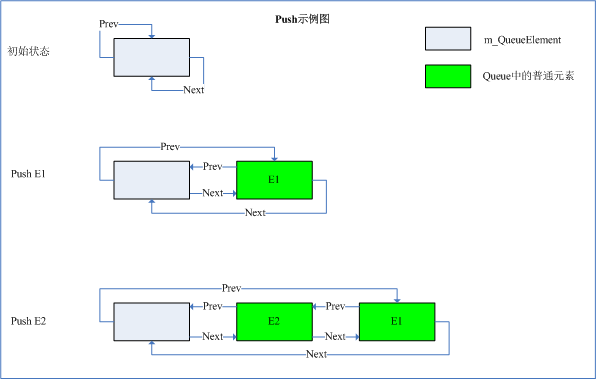
1 ZOSCond是一个事件信号量的封装。

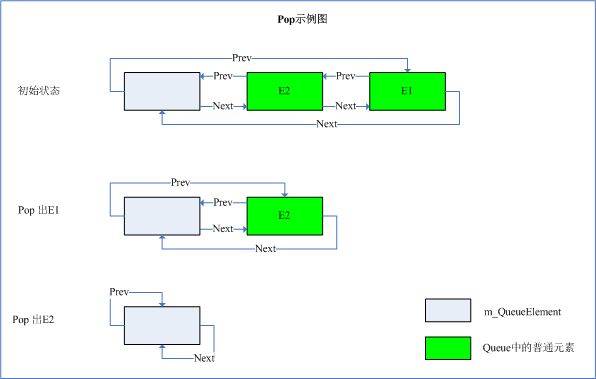
## 容器部分：

### Queue容器说明：

1 ZOSQueue是一个**先进先出**队列。Push函数实现往队列末尾放一个元素，Pop函数实现从队列头部取出一个元素。

如图：

图（二）Queue Push示例图



图（三）Queue Pop示例图

2 ZOSQueueElement是队列元素。m\_pObject用于存放ZOSQueueElement的实际内容。通过GetObject及SetObject函数可以设置及获取ZOSQueueElement的实际内容。

3 ZOSQueue及ZOSQueueElement在实现上采用双链表结构，有头结点。

4 ZOSqueue提供GetFirst，GetNext，GetCurrent函数用于遍历ZOSQueue中所有元素。

5 ZOSQueueBlocking继承自ZOSQueue，具备ZOSQueue的功能。同时结合信号量中的事件，完成在插入元素时触发信号量，取元素时等待信号量指定时间的功能。

### Heap容器说明：

1 ZOSHeap是一个最小堆的实现。

2 ZOSHeapElement是ZOSHeap的堆元素。m\_pObject用于存放ZOSHeapElement的实际内容。通过GetObject及SetObject函数可以设置及获取ZOSHeapElement的实际内容。m\_nValue是作为ZOSHeapElement被ZOSHeap进行堆排序的依据。m\_nValue值越小，则ZOSHeapElement位于ZOSHeap的层次就越高。

3 ZOSHeap在内存中用数组的结构来存放其所有的ZOSHeapElement，但逻辑上是一个最小堆。

如图：



图（四）堆的物理存储及逻辑结构

图中的15,18,27,24,23,40是各ZOSHeapElement的m\_nValue值。

4 执行ZOSHeap的Insert函数时，确保将最小值放到顶部。执行ZOSHeap的Extract函数时，重新排序。特别是ExtractMin函数后，确保将最小值依然放在树的顶部。

### ZOSHashTable容器说明：

1 ZOSHashTable是一个Hash表的实现。

内部结构如图：



图（五）Hash表结构图

2 ZOSHashTable执行Add函数将T元素放到m\_HashTable数组中，同时将原m\_HashTable数组中的内容接到新加入的T元素的后面。

3 ZOSObjct的实际内容存放于m\_pObject中。其成员变量m\_nHashValue,m\_nHashKey被ZOSHashTable的Add，Map，Remove函数使用。

4 ZOSObjectTable对ZOSHashTable进行实例化后封装，完成ZOSHashTable的功能。用Register，Unregister对应Add，Remove。

5 ZOSObjectTable提供GetFirst，GetNext，GetCurrent函数用于遍历ZOSObjectTable中所有元素。

6 ZOSTable与ZOSObjectTable的重要区别是，ZOSTable是一个模板实现。ZOSTable的DT，DK可以有更多选择。

### ZOSPool容器说明：

1 ZOSPool是一个池容器，其特点是预先分配好大小，使用上不能超过预先分配的大小。

2 ZOSPool有一个free和use的概念，free表示空闲，use表示使用。ZOSPool在逻辑上有一个空闲链表，链表中的所有元素都处于空闲状态，可被获取，然后使用。同时还有一个使用链表，链表中的所有元素都处于使用状态，可使用，然后置于空闲。虽然ZOSPool在逻辑上有空闲链表和使用链表，但物理实现上这些链表对应的所有值位于一个连续的内存中。通过m\_pFreeBegin,m\_pFreeNext,m\_pFreeEnd实现了逻辑上的空闲链表，通过m\_pUsedBegin, m\_pUsedNext, m\_pUsedEnd实现了逻辑上的使用链表。



图（六）ZOSPool逻辑结构图

3 GetFree与SetUsed函数必须配套使用。GetUsed与SetFree函数必须配套使用。

4 ZOSPool构造函数中确定了Pool的大小，但在实现时，先分配了指定大小的一半的空间，然后在使用过程中适情况不断分配，直到分配到Pool大小就不再分配。

### ZOSArray容器说明：

1 ZOSArray是一个数组。目前实现方式限制了ZOSArray内的T只能是基础数据类型。如：对象指针类型。

## 文件部分：

### ZOSFile类：

1 ZOSFile是一个二进制文件操作的封装。

### ZLog类：

1 封装日志操作。将日志等级分成4个等级，分别为ERROR，WARNING，INFO，DEBUG。

当设置等级为DEBUG时，日志文件中将打印所有等级的日志；

当设置等级为INFO时，日志文件中将打印INFO，WARNING，ERROR等级的日志；

当设置等级为WARNING时，日志文件中将打印WARNING，ERROR等级的日志；

当设置等级为ERROR时，日志文件中将只打印ERROR等级的日志；

2 通过宏LOG\_SET\_LEVEL,LOG\_ERROR等来实现日志的设置和打印。

3 由于在构造函数中将日志的实例赋值给pLog变量，所以这就决定了一个程序对应一个日志对象。

### XML文件相关：

1 ZXMLElement，ZXMLNode，ZXMLBase，ZXMLParser共同完成xml文件的读写操作。

## 网络部分：

### ZSocket类：

1 Initialize函数中初始化了socket库，同时创建出ZSocketEventThread。StartEventThread函数启动事件线程。

2 包含了通用的socket设置函数。如，设置（非）阻塞，发送缓存，接收缓存等。

### ZTCPSocket类：

1 继承自ZSocket，完成socket基本功能。

2 针对TCP的特殊性，实现Connect，Attach，Send，Recv

### ZTCPListenerSocket类：

1 继承自ZTCPSocket，对TCP监听类型的socket进行了封装。

2 重写ProcessEvent函数，改变socket事件的处理流程。

3 在收到accept事件后调用OnListenerEvent回调。

### ZUDPSocket类：

1 继承自ZSocket，完成socket基本功能。

2 针对UDP的特殊性，实现SendTo，RecvFrom，及组播相关函数。

### ZSocketEventThread类：

1 连接ZSocket的事件与Task的桥梁。

2 在WaitForEvent中，用select模型的方式监听事件。

## Task部分：

### 方法原理简述

基本目标：当某个事件发生时，此事件被处理。

内核任务调度模块采样了事件（ZEvent），事件线程（ZEventThread），任务（ZTask），任务线程（ZTaskThread），线程池（ZTaskThreadPool）五个类协调交互来完成上述目标。



图（1）

从图（1）可以看出：

1 当事件发生时，首先被ZEventThread检测到，ZEventThread从ZEvent集合中得到与此事件相关的ZEvent。

2 ZEventThread调用ZEvent的处理流程。

3 ZEvent的处理流程不做实际要处理的流程，而是将事件转交给与此ZEvent绑定的ZTask。ZTask保存着事件。

4 ZTask如果没有支持的线程，则从线程池中取出一个线程作为它的支持线程。ZTask将自己放到ZTaskThread的ZTask集合中。ZTaskThread循环从ZTask集合中取出ZTask进行处理。

5 ZTaskThread调用ZTask的处理流程来完成对事件的处理。

通过上述步骤1到5，当一个事件发生时，此事件得到了处理。

各个类职责

ZEventThread

1 循环不断检测事件是否发生。

2 维护ZEvent集合。

3 当事件发生时，从ZEvent集合中找到与此事件对应的ZEvent对象。调用ZEvent的处理流程。

4 每次从ZEvent集合中取出ZEvent时将ZEvent对象从ZEvent集合中删除。

ZEvent

1 维护对应的一个ZTask对象。

2 将事件转发给一个ZTask对象，或从ZTask对象中删除。

3 将自身添加到ZEventThread的ZEvent集合或从ZEvent集合中删除。

ZTask

1 维护一个ZTaskThread对象。

2 将事件保存，删除的维护。

3 从线程池中得到支持线程ZTaskThread。

4 将自身添加到ZTaskThread的ZTask集合或从ZEvent集合中删除。

ZTaskThread

1 循环从ZTask集合中取出ZTask对象，调用ZTask的Run函数。

2 维护ZTaskThread的任务负载。即维护ZTaskThread的任务数。

3 每次从ZTask集合中取出ZTask时将ZTask对象从ZTask集合中删除。

4 ZTask集合分成Heap和Queue。先处理Queue中的ZTask对象，再处理Heap中的ZTask对象。

ZTaskThreadPool

1 创建，开启，停止若干个线程。

2 维护线程池的总负载任务数及平均负载任务数。

3 被ZTask获得指定编号的ZTaskThread。如果此ZTaskThread负载的任务数超过一定值，则不让其被ZTask得到。通过这样的处理，保证线程池中的各个线程负载基本相同。

类具体说明

### ZEventThread类说明

1）ZEventThread继承自ZOSThread，所以ZEventThread为一个线程。自己会不断运行。

2）ZEventThread有m\_EventThreadTable成员变量，此变量保存着多个**句柄**与**事件对象**的组合对。也就是说通过一个句柄可以找到此句柄对应的事件。

3）AddEvent函数完成往m\_EventThreadTable插元素的操作，RemoveEvent完成从m\_EventThreadTable删除元素的操作。

4）OnEvent函数完成根据**句柄**得到**事件对象**，调用事件对象的ProcessEvent函数。执行OnEvent函数将**事件对象**从m\_EventThreadTable中删除。

注意：

类ZEventThread在OnThreadEntry中只调用了WaitForEvent，而ZEventThread的WaitForEvent函数只是在执行sleep操作。这样OnEvent就没有被调用，则事件对应的处理则不会被调用。

基于这样的问题，要使用ZEventThread，需要继承ZEventThread，并且重写WaitForEvent函数，在WaitForEvent函数中调用OnEvent。具体可阅读ZSocketEventThread的WaitForEvent函数。

### ZSocketEventThread类说明

1）继承自ZEventThread类，此类针对socket的读写事件。

2）AddEvent函数：调用ZEventThread函数的AddEvent，同时将socket句柄加到读写fd\_set集合中。

RemoveEvent函数：调用ZEventThread函数的RemoveEvent，同时将socket句柄从读写的fd\_set集合中删除。

3）WaitForEvent函数

总的思想：select等待读写事件，调用OnEvent执行读写事件。

具体步骤如下：

1 在556行执行select，等待读写事件的发生。

2 当有读或写事件发生，m\_nMonitorSetFromSelect返回事件的个数。

m\_nMonitorSetRead置为1。通过这两个标志，函数能进入482的FD\_ISSET的判断。

3 先检查可读事件，一旦发现bIsSet，则调用ZEventThread的OnEvent。

4 如果没有读事件或读事件处理完，则在495行m\_nMonitorSetRead=0，然后进入512行。

5 处理写事件，处理方法与读事件相同，不同点在ZEventThread传的事件类型为ZEvent：：EVENT\_WRITE。

6 如果没有写事件或写事件处理完，则进入525行，将m\_nMonitorSetProcessed = m\_nMonitorSetFromSelect。则不再进入读写事件的处理。

7 函数在539行到545行对各种标志变量置空。

8 循环进入了第1步。

SMS内核模块目前仅使用ZSocketEventThread类，用它来监听所有socket事件。

使用方法：

实例化，调用Start函数。

结束程序时调用Stop函数

### ZEvent类说明

1）成员变量包含一个m\_pEventTask，包含一个m\_pEventThread。即一个ZEvent对应一个ZTask和一个ZEventThread。在构造函数时设置m\_pEventThread值，在调用SetTask时设置了m\_pEventTask值。

2）成员变量包含m\_hHandle，此值在Create时设置，Close时清空。在本程序中此值即为Socket的句柄。

3）成员变量m\_nEventUID是一个事件的唯一标志。

4）RequestEvent函数将事件添加到EventThread中。

RemoveEvent函数将事件从EventThread中删除。

5）ProcessEvent函数完成功能是调用ZTask的AddEvent方法，ZTask的AddEvent方法完成将自己附着到一个支持线程上去执行。

**使用方法：**

初始化时调用Create，SetTask。

产生事件时调用RequestEvent。由于EventThread取完消息后将消息删除，所以如果还要继续监听事件，需要调用RequestEvent。

删除事件时调用RemoveEvent。

结束时调用Close。

### ZTask类说明

1）一个ZTask可以对应多个事件类型，对应的事件类型见TASK\_EVENT。

2）m\_nTaskEventFlag变量保存中所有事件，所有事件通过or共存。

3）支持线程由m\_UsedTaskThread指向。m\_DefalutTaskThread,m\_CurrentTaskThread在SMS内核中不使用。

4）m\_nTaskEventFlag, m\_nTaskThreadBegin, m\_nTaskThreadBound变量用于获得支持线程时使用。一些计数，用于循环从线程池中获得支持线程。

5）m\_OSQueueElement，m\_OSHeapElement变量中存放着task自己的指针

6）Create函数设置m\_OSQueueElement，m\_OSHeapElement变量的值。

7）AddEvent函数，通过or m\_nTaskEventFlag变量添加事件类型。如果本task没有绑定支持线程，则绑定一个支持线程。调用支持线程的AddTask函数

8）RemoveEvent函数调用支持线程的RemoveTask或调用m\_OSQueueElement，m\_OSHeapElement的Remove。

9）Run函数，是任务函数的最重要的函数，此函数会被ZTask的支持线程调用。重载此函数后检测自己关心的事件，然后处理事件处理函数。如ZAdmin的Run函数。

**使用方法：**

调用Create方法。

重写Run方法。在Run函数中进行处理。

关闭时调用Close方法。

### ZTaskThread类说明

1）继承自ZOSThread则ZTaskThread是一个线程会不断自动运行。

2）AddTask函数目的是将ZTask的m\_OSQueueElement放到m\_TaskThreadQueue中。所以如果原来在m\_TaskThreadHeap中则先删除，然后添加到m\_TaskThreadQueue中。

3）RemoveTask函数目的是将ZTask的m\_OSQueueElement及m\_OSHeapElement从m\_TaskThreadQueue，m\_TaskThreadHeap中删除。

4）WaitForTask函数目的是从Queue及Heap中获得ZTask。先从Queue中获得，再从Heap中获得。从Queue中取时也确保从Heap中取。这是因为位于Queue中的消息具有更高的优先级。如果从Heap中取时，超时时间还没到，则执行PopBlocking等待指定毫秒。

5）OnThreadEntry函数内调用Run函数。

如果Run函数返回>0则表示此ZTask是一个Update的事件，继续将这个ZTask放到heap中。调用m\_OSHeapElement的SetValue函数重新设置下次超时事件的时间点。

如果Run函数返回不为0.

如果task还有其他事件，则继续将ZTask的m\_OSHeapElement插入Heap。

如果task没有其他事件，则将ZTask从Heap中删除。ZTask的支持线程设置为NULL。下次ZTask可能选择其他的支持线程。

**使用方法：**

属于调度模块机制内部处理，已由ZTask的AddEvent函数完成。

### ZTaskThreadPool类说明

1）Initialize函数new出m\_nTaskThreadCounter个线程。

Uninitialize函数释放线程。

StartTaskThread函数启动线程池中的所有线程。

StopTaskThread函数停止线程池中的所有线程。

AddTaskEvent函数给线程池的任务数计数+1

SubTaskEvent函数给线程池的任务数计数-1

2）

GetTaskThread函数用于获得某个index的线程。这里有限制条件，只有当这个线程的附着的Task数小于线程池的平均负载才获得到。

**使用方法：**

在程序初始化时调用StartTaskThread，在程序结束时调用StopTaskThread。

### ZIdleTask类说明

1）Initialize函数完成给ZIdleTask指定ZIdleTaskThread。

Uninitialize函数完成将ZIdleTask指定的线程删除。

StartIdleTaskThread函数启动ZIdleTask指定的线程

StopIdleTaskThread函数停止ZidleTask指定的线程。

AddIdleTask函数调用m\_pIdleTaskThread的AddIdleTask

RemoveIdleTask函数调用m\_pIdleTaskThread的RemoveIdleTask

使用方法：

继承ZIdleTask类。

调用AddIdleTask函数将自身添加到ZIdleTaskThread中。

重载Run函数，处理TASK\_IDLE\_EVENT事件。

使用时参考ZTimeoutTask类。

### ZTimeoutTask类说明

ZIdleTask使用实例。

1）ZTimeoutTask本身是一个task，同时还包含了一个m\_pTimeoutTask变量。当超时事件发生时，调用m\_pTimeoutTask的AddEvent。

**使用方法：**

继承ZTimeoutTask类。

调用SetTimeoutTask

调用SetTimeout

重载Run函数，在Run函数运行中前运行ZTimeoutTask的Run函数。

使用时参考ZAdmin类

### ZIdleTaskThread类说明

1）循环检测heap中是否有idle任务，有，则给任务添加TASK\_IDLE\_EVENT事件。

2）AddIdleTask函数将下次超时时间值设置到m\_IdleTaskHeap中。

**使用方法：**

调用ZIdleTask的Initialize函数创建ZIdleTaskThread对象。

调用ZIdleTask的StartIdleTaskThread函数启动线程。

## 问与答

### ZTask与ZTaskThread关系：

ZTaskThread中有一个Queue和一个Heap，Queue存放多个QueueElement，Heap中存放着HeapElement。

ZTask中有一个QueueElement对象和一个HeapElement对象，这两个对象存放着ZTask的指针。

ZTask和ZTaskThread就是通过Queue，Heap容器保存QueueElement，HeapElement的方式从逻辑上使一个ZTaskThread对应处理多个ZTask。

一个ZTask对象可同时包含多个事件。

一个ZTask对象只对应一个ZTaskThread，即不存在一个ZTask对象被两个不同的ZTaskThread对象同时拥有。

不存在同一个Task对象指针在某个ZTaskThread对象的Queue中出现多次。

不存在同一个Task对象指针在某个ZTaskThread对象的Heap中出现多次。

存在同一个Task对象指针即在某个ZTaskThread对象的Queue中，又在Heap中。

### ZTaskThread如何做到ZTask的优先级处理：

在ZTaskThread的WaitForTask函数中，仅在Queue中没有了对象，才处理Heap中的对象。也就是说Queue中的Task的优先级比Heap中的Task的优先级高。

ZTask对象在AddEvent时，先将ZTask放到ZTaskThread的Queue中。具体执行的逻辑为，如果ZTask已经存在于Heap中，则现将Heap中的ZTask删除，然后再放入Queue中，如果Queue中已经存在此Task对象，则不再插入。

目前设计中只有TASK\_UPDATE\_EVENT事件放在了Heap内。

Queue中的ZTask是在ZTask执行AddEvent函数时加入的。

Heap中的ZTask是在ZTask::Run函数返回值非0时加入的。

### ZTask的事件的产生与销毁：

在ZTask的AddEvent函数被调用时事件被添加到ZTask中。实现的方式为ZTask使用m\_nTaskEventFlag变量存放着事件。每次事件的添加采用或的方式添加到m\_nTaskEventFlag变量中。

ZTaskThread执行ZTask的Run函数，在Run函数中通过调用ZTask的GetEvent函数将ZTask的**所有**事件取出。实现方式上采用减的方式。

由于ZTask的事件由一个变量来存放，所以当ZTask的某个事件未处理完，这个事件又再次发生，则这个事件只会被处理一次。如ZTask的TASK\_READ\_EVENT事件发生，但此事件还未被处理，此时此事件又再次发生，则只处理一次TASK\_READ\_EVENT事件。

从ZTask的事件的销毁方式（GetEvent函数）上可以看出，事件一次性就被全部取出来了。所以在编写自己的Task的Run函数时要注意怎么处理事件。

### ZEvent与ZTask关系：

ZEvent存放着一个ZTask对象指针m\_pEventTask，在ZEventThread的OnEvent函数中，执行ZEvent的ProcessEvent函数，函数中调用ZTask的AddEvent。也就是说ZEvent与ZTask绑定，ZEvent的事件由ZTask来处理。

### ZEvent与ZEventThread关系：

ZEvent存放着一个ZEventThread对象指针，ZEvent的RequestEvent函数及RemoveEvent函数都是通过ZEventThread的AddEvent函数及RemoveEvent来实现。

ZEventThread有一个ZOSObjectTable，可存放多个ZOSObject，ZEvent内包含一个ZOSObject对象。逻辑上ZEventThread中存放着多个ZEvent。

ZEventThread的OnEvent函数内部实际执行ZEvent的ProcessEvent函数。

### ZEvent的事件的产生与销毁：

ZEventThread类的WaitForEvent函数本身没有具体实现。

ZSocketEventThread是ZEventThread的一个具体实现，在WaitForEvent函数中，当socket句柄有读写事件时，触发了ZEvent的读写事件。

在ZEventThread类的OnEvent函数中调用Unregister，使事件销毁。

### socket事件的触发如何对应到ZTask继承类的Run函数中：

由于采用select的方式，所以当socket句柄存在读写事件，将被检测到。

由于ZSocket继承自ZEvent，所以ZSocket可绑定一个ZTask，同时对应一个ZEventThread的具体实现ZSocketEventThread。

在ZSocketEventThread的WaitForEvent函数中，select获得socket的读写事件，调用ZEventThread的OnEvent，接着调用ZEvent的ProcessEvent，再接着调用ZEvent绑定的ZTask类的AddEvent，此时socket事件就转移到了ZTask。ZTaskThread的OnThreadEntry函数执行ZTask的Run函数。

### ZTCPListenerSocket事件特殊处理：

由于ZTCPListenerSocket重写了ZEvent的ProcessEvent，所以未采用socket事件对应Task的Run函数处理的方法。ZTCPListenerSocket的读事件即为accept事件，在accept成功时，最终调用OnListenerEvent回调。

### ZTaskThreadPool的m\_nNoBlockTaskThreadCounter字段意义：

Task模型中的Task会附着于一个线程（线程A）上，当Task没有事件时，Task将解除与这个线程（线程A）的附着关系。当Task的事件再次发生时，Task会从线程池中再挑选出一个线程（线程B）附着上，然后由这个线程（线程B）来完成这个事件。系统运行过程中存在这样的应用，希望将线程池中的线程分区段。如线程池有100个线程，0-9号线程给Task（Task甲）用，10-19号线程给另一个Task（Task乙）用，剩余的20-99号线程给所有Task共用。也就是说，0-19号线程将用于特殊用途，那么此时的m\_nNoBlockTaskThreadCounter值为20。

阅读相关代码：

ZTask::AddEvent函数；完成Task附着线程。

ZTaskThread::OnThreadEntry函数； pTask->m\_CurrentTaskThread=NULL完成Task与线程解除附着关系。

ZTask::ZTask函数；完成使用线程区段。

ZTCPListenerSocket::ZTCPListenerSocket函数；完成任务使用特定区段线程。