# 前言

## 面向对象中间件体系结构

* 1. 主机基础设施中间件：封装socket,线程等不同主机的实现，形成统一的接口。如java,ACE
  2. 分布式中间件：连接管理，内存管理，整编，解编，端点和请求的多路分离，同步，多线程等，使程序员象开发独立应用程序一样开发分布式应用程序。分布式中间件的核心是ORB(Object Requests Broker对象请求代理)，如：COM+，JAVA RMI，CORBA

# 通信设计空间

## 面向连接协议

需要做出如下设计：

* 数据成帧策略
* 连接多路复用策略

1. 链接多路复用：多个线程复用同一个TCP链接
2. 非多路复用：多个进程使用多个链接。系统开销大

## 同步和异步消息交换

## 消息传递与共享内存

消息传递：消息中间件

共享内存：

* 本地共享内存：shmget(),shmat(). 内存映射文件
* 分布式共享内存：虚拟内存，是本地共享内存的一种抽象。

# SOCKET API 概述

# ACE Socket wrapper façade

## ACE\_Addr

运算符 == , !=

addr\_type, addr\_size

## ACE\_INET\_Add

## ACE\_IPC\_SAP

为其他ACE对象提供了基本的“I/O”句柄操作能力

enable()

disable()

get\_handle()

set\_handle()

## ACE\_SOCK

继承自ACE\_IPC\_SAP,

get \_local\_addr()

set\_local\_addr()

open()

close()

get\_remote\_addr()

set\_option()

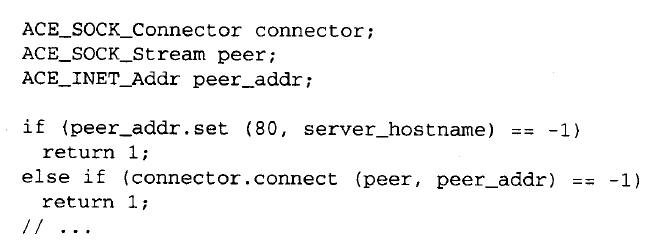
get\_option()

## ACE\_SOCK\_Connecter

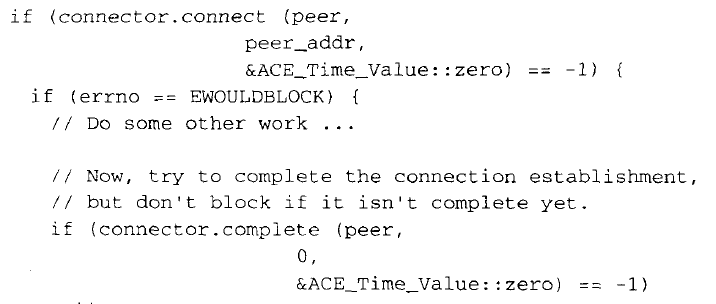
主动连接模式，是一个工厂类。发起一个连接，并在连接成功后初始化一个ACE\_SOCK\_Stream对象。

可以通过“阻塞”“非阻塞”“定时”方式发起。

以下是非阻塞模式



以下是非阻塞模式



## ACE\_SOCK\_Stream

数据传输角色的实现。

Send()

Recv()

Send\_n()

Recv\_n()

Recvv\_n() //使用分散读取系统函数，高效、完整读取多个缓冲区，参数 iovec[]

Sendv\_n() //使用集中写入系统函数，高效、完整写入多个缓冲区

Enable(ACE\_NOBLOCK)

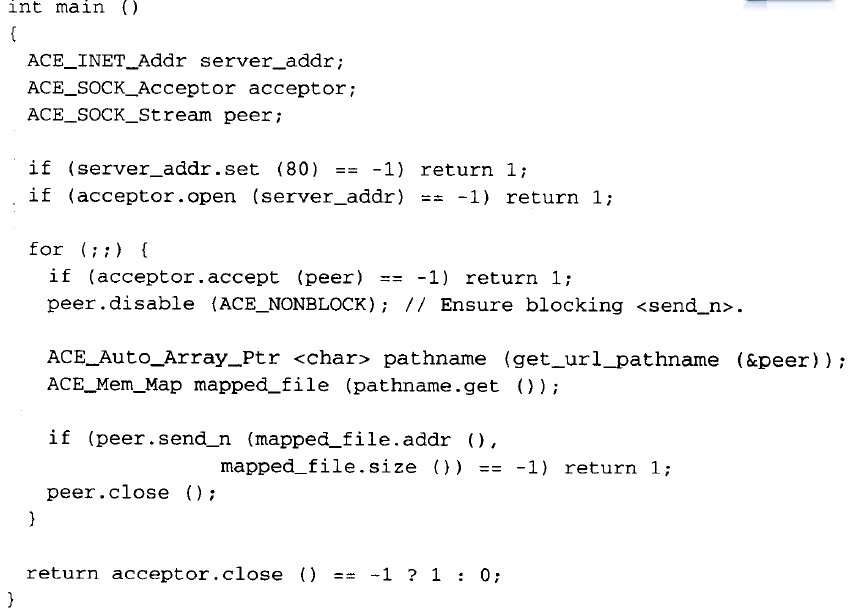
## ACE\_SOCK\_Acceptor

被动连接模式的实现。相当于accept()

是一个工厂类，初始化一个ACE\_SOCK\_Stream

Open()

Accept()



## ACE\_Mem\_Map

通过内存映射文件机制，将文件映射到内存中，以便及时读取，提高了效率。此外，内存映射文件还可以被多个进程共享。

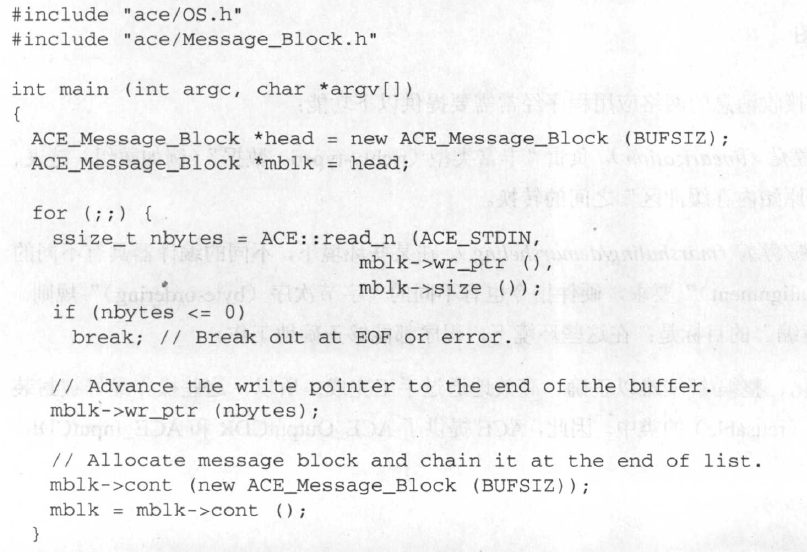
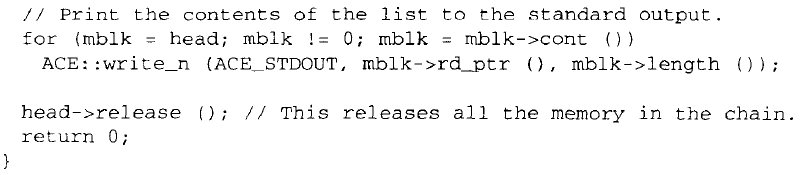
# 网络日志服务程序的实现

## ACE\_Message\_Block

标准的消息管理包括：

1. 收到消息后，保存到缓冲区
2. 添加和删除消息的头和尾
3. 将消息分段，或重新组装，以适应网络的最大传输单元MCU
4. 将消息保存到缓冲区中，以进行传输
5. 对“未按顺序”接收的消息进行记录

ACE\_Message\_Block实现了composite模式。避免频繁的内存复制操作。提高效率。

注：也可以用ACE::write\_n(ACE\_STDOUT,head)一次性写入

## ACE\_InputCDR与ACE\_OutputCDR

功能：整编，解编。避免不同操作系统对数据类型（如int）的不同实现方法。

ACE\_OutputCDR header(ACE\_CDR::MAX\_ALIGNMENT+userlength)

ACE\_OutputCDR::from\_boolean(ACE\_CDR\_BYTE\_ORDER)

# 并发设计空间

服务器可以分为循环式、并发式和反应式等几大类。

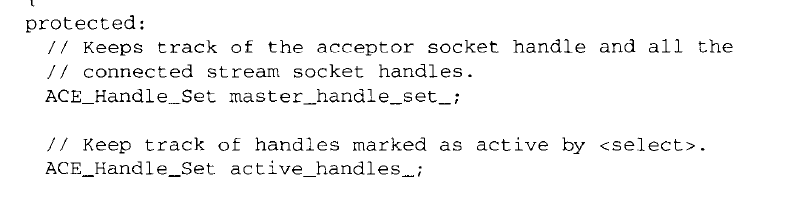
# 操作系统的并发机制

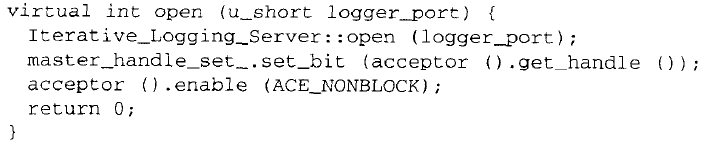
# ACE同步事件多路分离

## ACE\_Handle\_Set

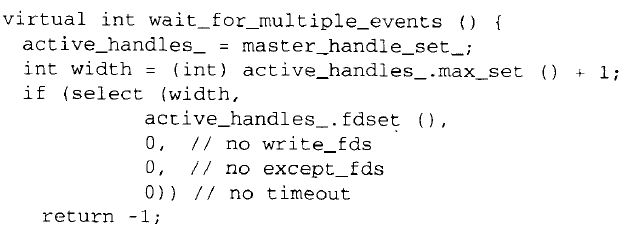
## ACE\_Handle\_Set\_Iterator

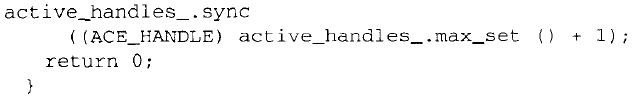
## ACE::select( )





Set\_bit() //将一个句柄放置在set中





领悟到的：

Logging\_handler 是一个工具对象。使用他时，先设置一个handle,然后再进行操作。就像螺丝刀一样。先指定给一个人，然后这个人才可以操作。

采用ACE\_Hash\_Map将socket句柄与log\_file句柄关联在一起。从而实现根据socket句柄写相应log\_file的功能。

# ACE进程Wrapper Façade

ACE\_OS::getenv(“fafaff”) 获得环境变量

ACE\_Process\_Options opt;

Opt.command\_line(“%s %d”,”hhafaf”,20);

ACE\_Process child;

Child.spawn(opt);

Child.wait();

Child.exit\_code();

# ACE线程Wrapper Façade

## ACE\_Thread\_Manager

创建，管理线程

ACE\_Thread\_Manager.instance()->spawn()

在执行线程时，为线程动态分配一个属于自己的对象，将对象传递给线程，线程结束时，由线程自身删除这个对象。

ACE\_Thread\_t me = ACE\_OS.thr\_self();

ACE\_Thread\_Manager.instance()->testCancle(me) == false 协同式删除

ACE\_Thread\_Manager.instance()->cancle\_all() 协同删除由ACE\_Thread\_Manager创建的所有线程。可以调用wait()等待。

## ACE\_Sched\_Params

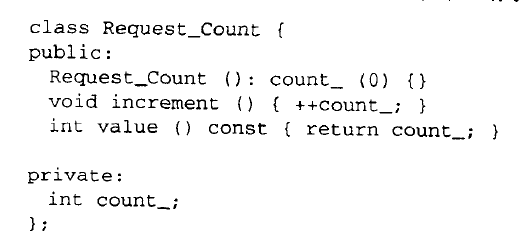
设置线程属性，如优先级，调度策略等

## ACE\_TSS

实现了“线程专有存储代理”，实现一个每个线程专有的对象。但这个对象看上去好像是一个全局变量。比如错误码，所有的线程都需要一个属于自己的错误码。

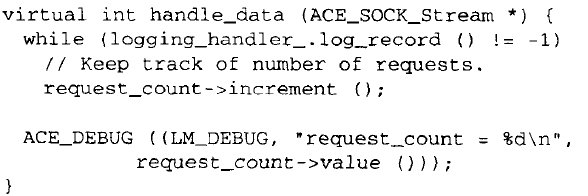
->操作符，得到和TSS键相关联的线程专有对象

Cleanup() 线程结束时，删除TSS对象

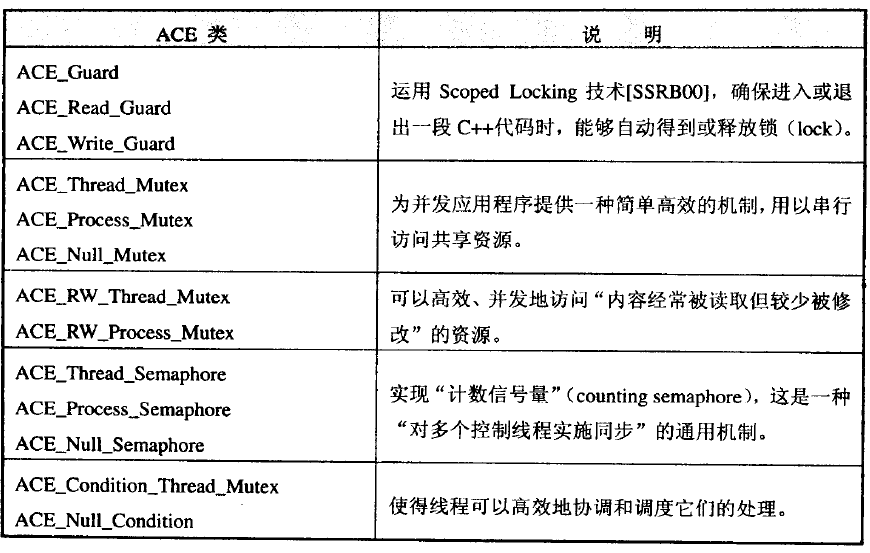




在线程内部：



# ACE同步wrapper façade



消息队列中信号量的使用：

多个生产者，多个消费者。

刚开始：notfull = 0,notempty =0. 不满，也不空。

生产者流程：

Notfull+1。

Notempty -1

消费者流程：

Notempty +1

Notfull-1

卷二

# ACE\_Service\_Configurator框架

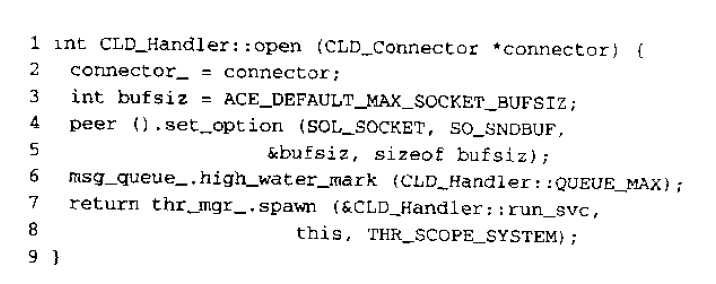
# ACE\_Message\_Queue

## 初始化和流控制方法

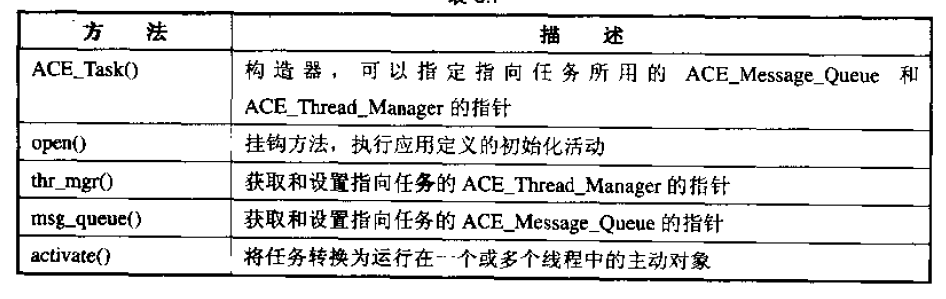
* Open() 初始化队列，设置水位标，设置通知策略
* Hight\_warter\_mark(),Low\_warter\_mark() 设置/获取流控制何时开始和结束的高低水位标
* Notification\_strategy() 设置，获取通知策略

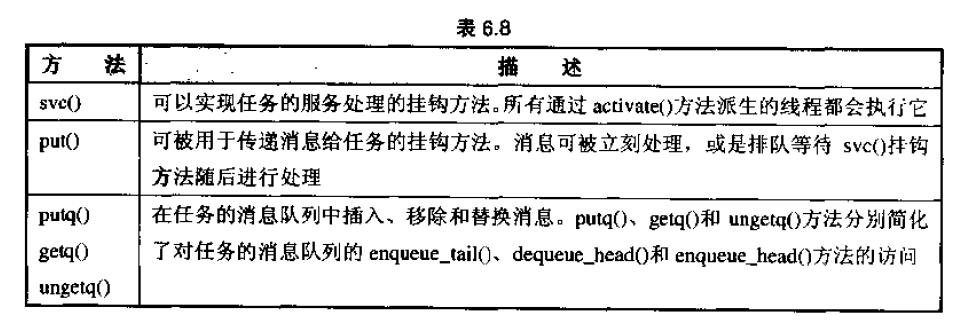
## 入队出对方法和消息缓冲

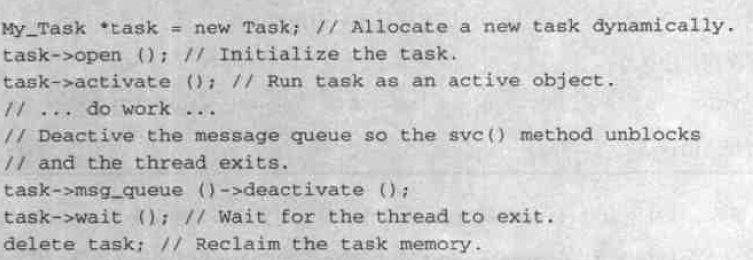
* Is\_empty()
* Is\_full()
* Enqueue\_head()
* Enqueue\_tail()
* Dequeue\_head()
* Dequeue\_tail()
* Enqueue\_prio() 根据其优先级来插入队列



# ACE\_Task



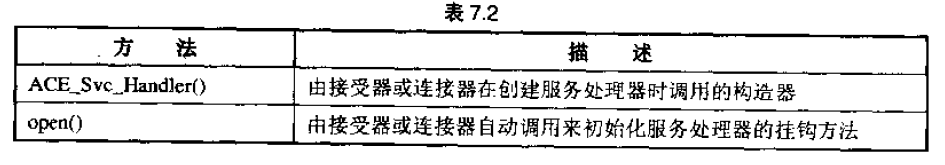


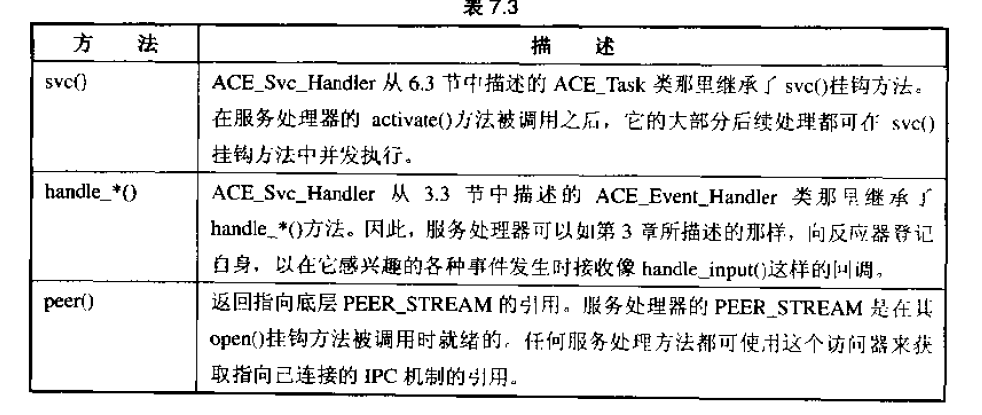


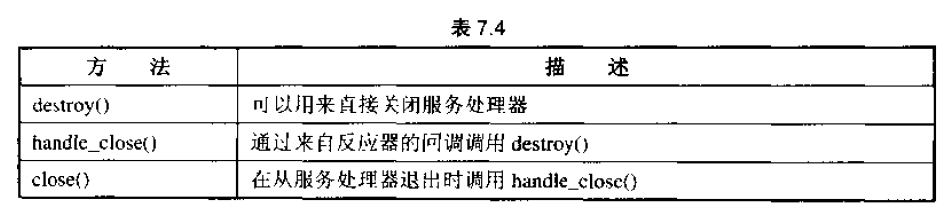
# Acceptor-Connector框架

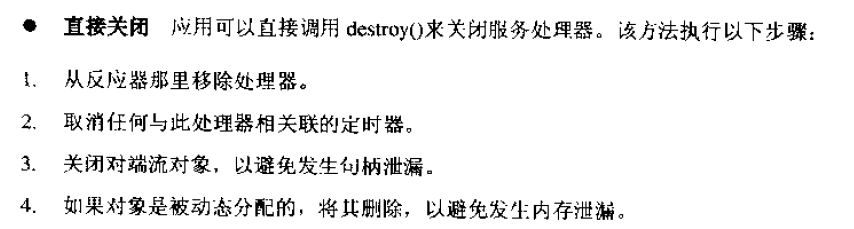
## ACE\_Svc\_Handler

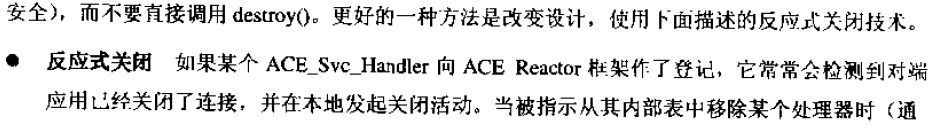
继承自ACE\_Task，而ACE\_Task又继承自ACE\_Event\_Handler

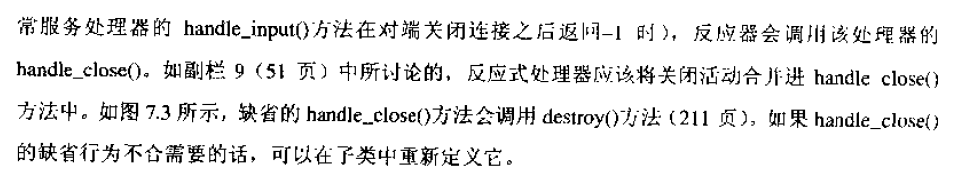


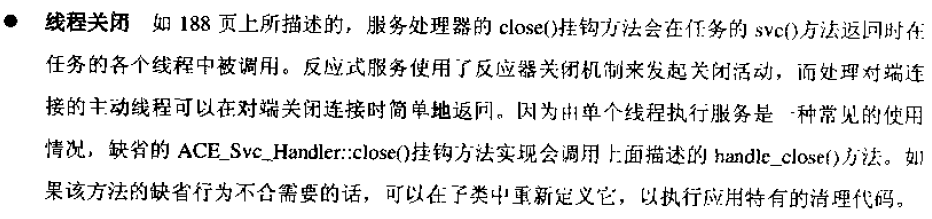




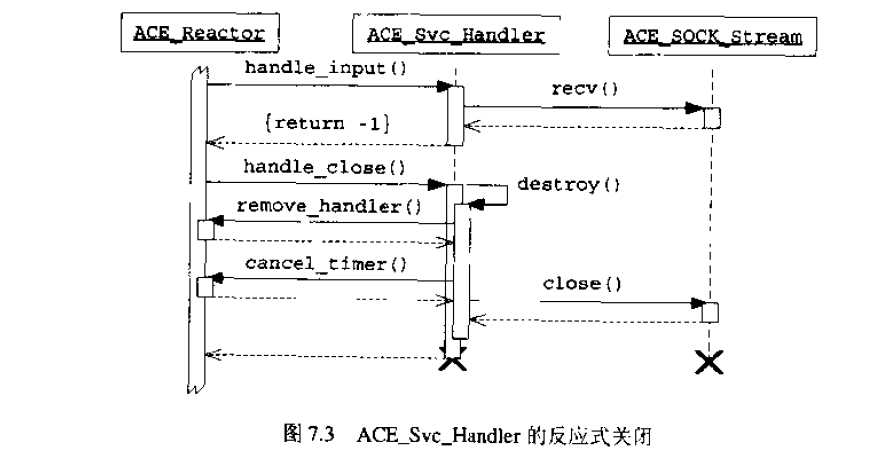






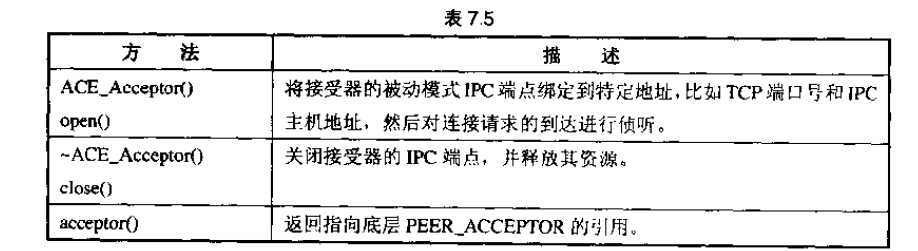


注意：close()方法是由每个线程都进行调用的。

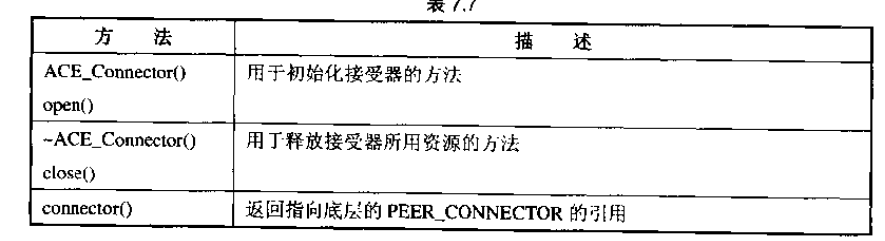


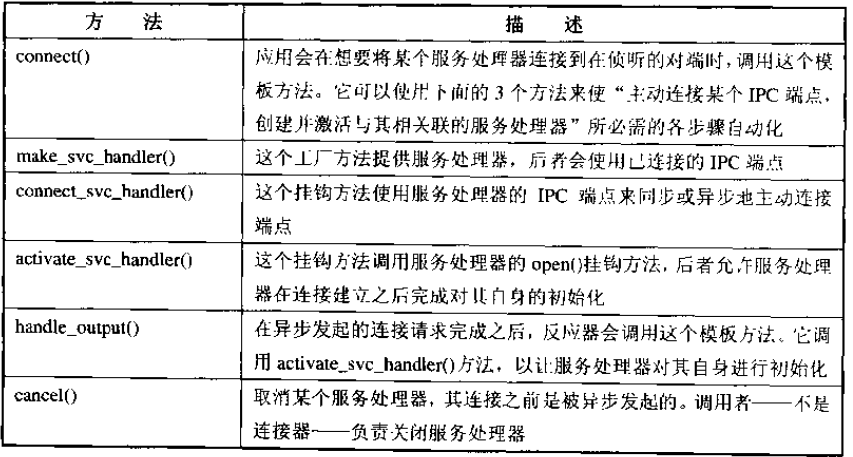
该类使用起来非常简单，因为他实现了Task框架和ACE\_Event\_Handle

## ACE\_Acceptor类

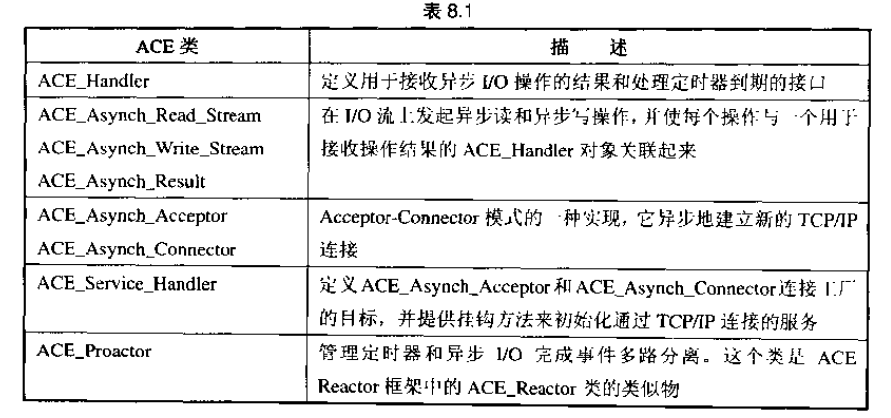


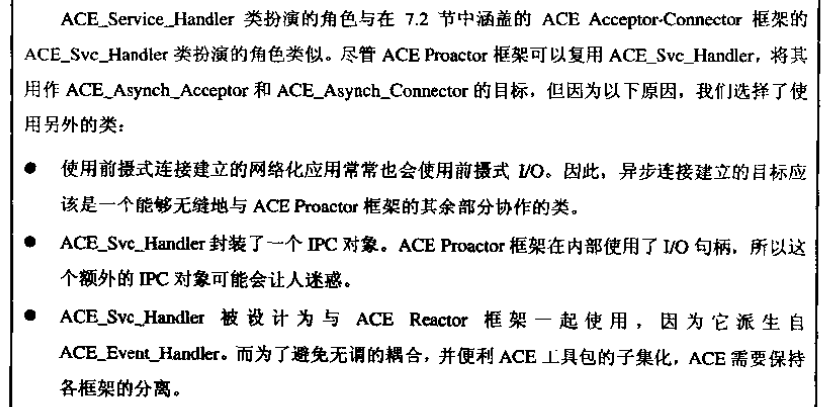
## ACE\_Connector类

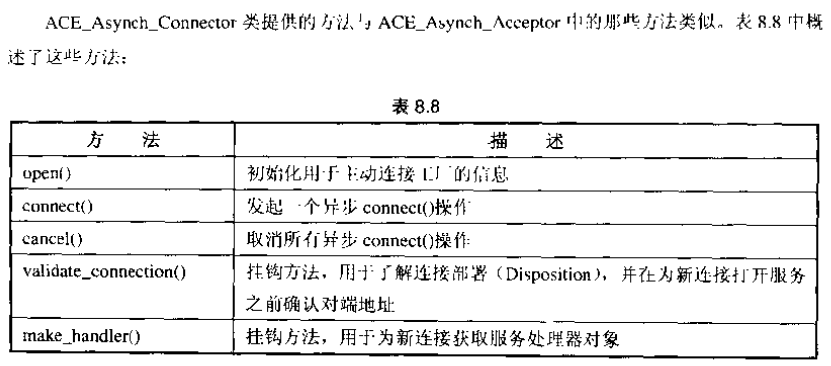


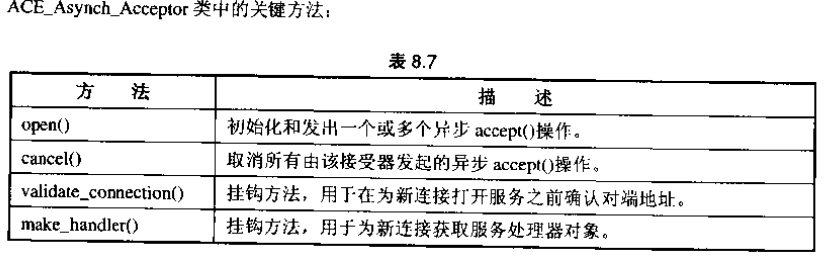


## ACE\_Proactor框架









## ACE\_Proactor类

