Module06 – C++ ACE

C++ ACE - 课程目标

作为网络应用开发方面一个影响深远的项目， ACE 的功能和性能已经在业界诸多重要应用中得到了验证和高度的评价，尽管 ACE 以 C++ 语言的方式组织，但并不影响后续的网络通信框架或组件以它为范本。
在这个 Module 中，将从以下几个方面有限的内容中来领略并且应用这个久负盛名的网络通信项目：

- ACE 基础的网络 I/O 相关对象：
  - ACE_SOCK、 ACE_SOCK_Acceptor、 ACE_SOCK_Connector、 ACE_INET_Addr、 ...
- Reactor 框架：
  - ACE_Event_Handler、 Timer、 ACE_Reactor、 ...
- Service Configuration 框架

C++ ACE - 课程目标（续）

- Acceptor-Connector 框架
- Proactor 框架：
  - ACE_Service_Handler 、 ACE_Proactor 、 ...
- Task 框架：
  - ACE_Thread_Manager 、 ACE_Task
- 杂项，一些基础设施：
  - Logger 、 Singleton 、 ...

Module06-01
C++ ACE: I/O 相关对象

C++ ACE - ACE 简介

- ACE 简介
- I/O 相关对象
- Reactor 框架
- Service Configuration 框架
- Task 框架
- Acceptor-Connector 框架
- Proactor 框架
- 杂项

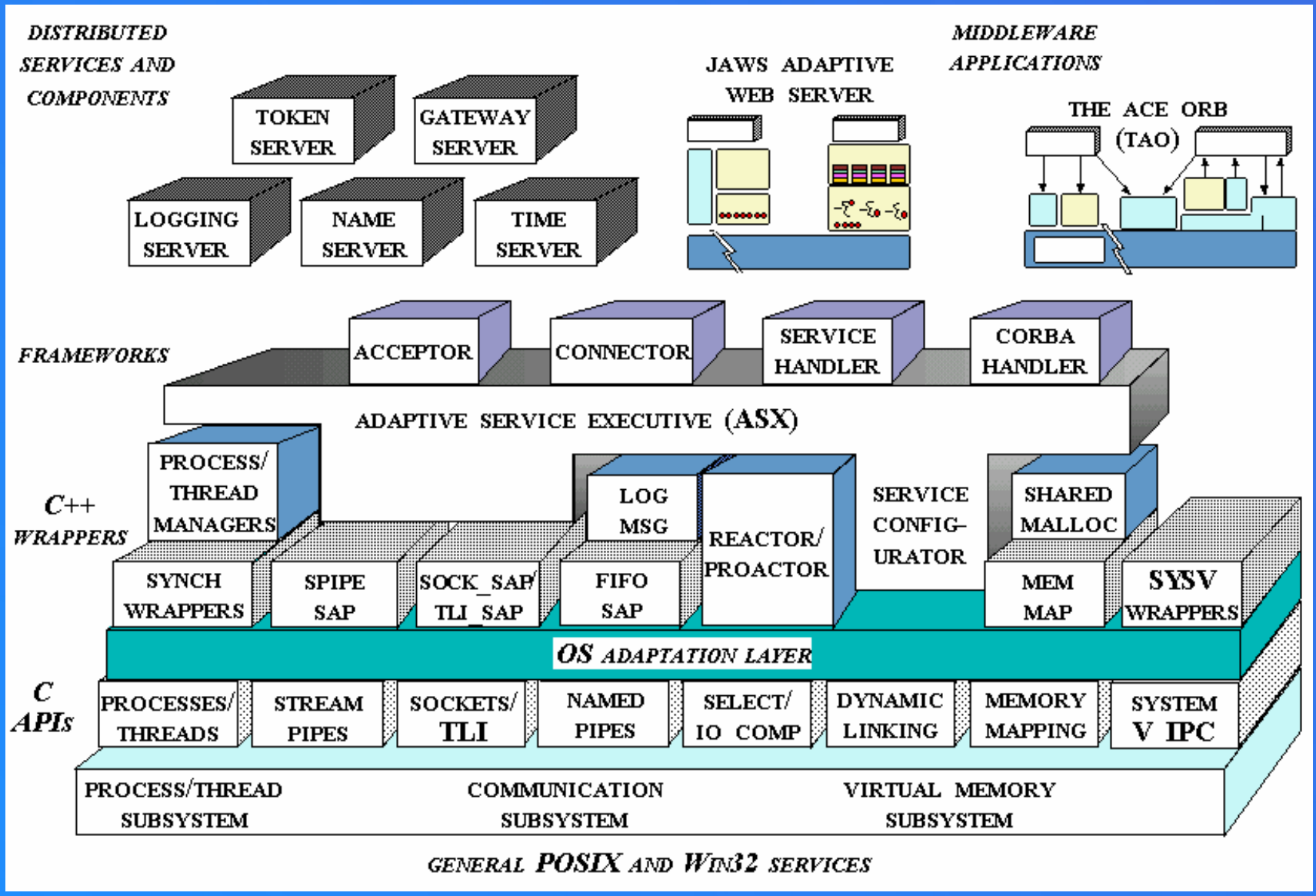
ACE 简介 - 关于 ACE

- 关于 ACE
  - ACE 自适配通信环境（Adaptive Communication Environment）是面向对象的构架和工具包，它为通信软件实现了核心的并发和分布式模式。 ACE 包含的多种组件可以帮助通信软件的开发获得更好的灵活性、效率、可靠性和可移植性。 ACE 中的组件可用于以下几种目的：
    - 并发和同步
    - 进程间通信（IPC）
    - 内存管理
    - 定时器
    - 信号
    - 文件系统管理

ACE 简介 - 关于 ACE

- 关于 ACE（续）
  - 线程管理
  - 事件多路分离和处理器分派
  - 连接建立和服务初始化
  - 软件的静态和动态配置、重配置
  - 分层协议构建和流式构架
  - 分布式通信服务：名字、日志、时间同步、事件路由和网络锁定，等等。

- ACE 分层体系架构图



ACE 简介 - ACE 体系架构

- ACE 层次
  - 操作系统适配层
  - C++ 包装层
  - 架构与模式层

ACE 简介 - ACE 体系架构

- ACE 操作系统适配层
  - OS 适配层是位于本地 OS API 和 ACE 之间的“瘦”代码层，它使 ACE 的较高层与平台依赖性屏蔽开来，从而使得通过 ACE 编写的代码保持了相对的平台无关性。只需要极少的努力，开发者就可以将 ACE 应用移植到任何平台上。
  - 目前 ACE 适用的 OS 平台包括：实时 OS（ VxWorks 、 Chorus 、 LynxOS 和 pSoS ）、大多数版本的 UNIX（ SunOS 4.x 和 5.x; SGI IRIX 5.x 和 6.x; HP-UX 9.x, 10.x 和 11.x; DEC UNIX 3.x 和 4.x; AIX 3.x 和 4.x; DG/UX; Linux; SCO; UnixWare; NetBSD 和 FreeBSD ）、 Win32/WIN64 以及 MVS OpenEdition 。

ACE 简介 - ACE 体系架构

- ACE C++ 包装层
  - C++ 包装层包括一些 C++ 包装类，它们可用于构建高度可移植的和类型安全的 C++ 应用。这是 ACE 工具包最大的一部分，大约包含了总源码的 50%。 C++ 包装类可用于：
    - 并发和同步： ACE 提供若干并发和同步包装类，对本地 OS 多线程和多进程 API 进行了抽象。这些包装类封装用于线程和进程的原语，比如信号量、锁、栅栏（ Barrier ）和条件变量。另外还有更高级的原语可用，比如守护（ Guard ）。所有这些原语共享类似的接口，因而很容易使用和相互替换。
    - IPC： ACE 提供若干 C++ 包装类，封装不同 OS 中不同的进程间通信（ IPC ）接口。例如， ACE 的包装类封装了以下 IPC 机制：BSD socket 、 TLI 、 UNIX FIFO 、流管道、 Win32 命名管道，等等。 ACE 还为消息队列提供包装类，包括特定的实时 OS 的消息队列

ACE 简介 - ACE 体系架构

- ACE C++ 包装层（续 1 ）
  - 内存管理组件： ACE 包含的一些类可用于内存动态分配和释放；其中包括允许预分配所有动态内存的类。这些预分配的内存随即通过 ACE 提供的管理类的帮助进行本地管理。在大多数实时和嵌入式系统中，这样的细粒度管理极为必要。另外还有一些类用于灵活地管理进程间共享内存。
  - 定时器类：有多种不同的类可用于处理定时器的调度和取消。 ACE 中不同种类的定时器使用不同的底层机制（堆、定时器轮（ timer wheel ）或简单列表）来提供不同的性能特性。但是，不管底层使用何种机制，这些类的接口都是一致的，从而使得开发者很容易使用任何一种定时器类。除了这些定时器类，还有封装高分辨率定时器（在部分平台上可用，比如 VxWorks, Win32/Pentium, AIX 和 Solaris ）和 Profile Timer 的包装类。
  - 容器类： ACE 还拥有若干可移植的 STL 风格的容器类，比如 Map 、 Hash_Map 、 Set 、 List ，等等

ACE 简介 - ACE 体系架构

- ACE C++ 包装层（续 2 ）
  - 信号处理： ACE 提供对特定 OS 的信号处理接口进行封装的包装类。这些类使得开发者能够很容易地安装和移除信号处理器，并且可以为一个信号安装若干处理器。另外还有信号守卫类，可用于在看守的作用域之内禁止所有信号。
  - 文件系统组件： ACE 含有包装文件系统 API 的类。这些类包括文件 I/O 、异步文件 I/O 、文件加锁、文件流、文件连接包装，等等。
  - 线程管理： ACE 提供包装类来创建和管理线程。这些包装还封装了针对特定 OS 的线程 API ，可被用于提供像线程专有存储这样的功能。

ACE 简介 - ACE 体系架构

- ACE 架构组件
  - ACE 含有以下一些组件：
    - 事件处理：大多数通信软件都含有大量处理各种类型事件（比如，基于 I/O 、基于定时器、基于信号和基于同步的事件）的代码。软件必须高效地多路分离、分派和处理这些事件。遗憾的是，大多数时间开发者们都在反复地编写这些代码，“重新发明轮子”。这是因为，事件多路分离、分派和处理代码全都紧密地耦合在一起，无法彼此独立地使用。 ACE 提供了被称为 Reactor（反应器）的构架组件来解决这一问题。反应器提供用于高效地进行事件多路分离和分派的代码，并极大地降低了它们与处理代码之间的耦合，从而改善了可复用性和灵活性。

ACE 简介 - ACE 体系架构

- ACE 架构组件（续 1 ）
  - 连接或服务初始化组件： ACE 提供 Connector （连接器）和 Acceptor （接受器）组件，用于降低连接初始化与连接建立后应用执行的实际服务之间的耦合。在接受大量连接请求的应用服务器中，该组件十分有用。连接首先以应用特有的方式初始化，然后每一连接被相应的处理例程分别处理。这样的去耦合使得开发者能够分别去考虑连接的处理和初始化。因此，如果在随后的阶段开发者发现连接请求数多于或是少于估算，它可以选择使用不同的初始化策略集（ ACE 提供了若干可供开发者挑选和选择的策略），以获得所要求的性能水平。

ACE 简介 - ACE 体系架构

- ACE 架构组件（续 2 ）
  - 流组件： ACE Stream 组件用于简化那些本质上是分层的（ layered ）或层次的（ hierarchic ）软件的开发。用户级协议栈的开发是一个好例子；这样的栈由若干互连的层次组成。这些层次或多或少可以相互独立地进行开发。当“数据”通过时，每一层都处理并改变它，并将它传递给下一层，以作进一步的处理。因为各层是相互独立的，它们很容易被复用或替换。
  - 服务配置组件：通信软件开发者面临的另一个问题是，很多时候，软件服务必须在安装时配置，或必须在运行时重配置。应用中特定服务的实现可能需要进行改动，因而应用可能必须用新改动的服务重新配置。 ACE Service Configurator （服务配置器）为应用的服务提供动态的启动、挂起和配置。

C++ ACE - I/O 相关对象

- ACE 简介
- → I/O 相关对象
- Reactor 框架
- Task 框架
- Acceptor-Connector 框架
- Proactor 框架
- 杂项

I/O 相关对象 - 概要

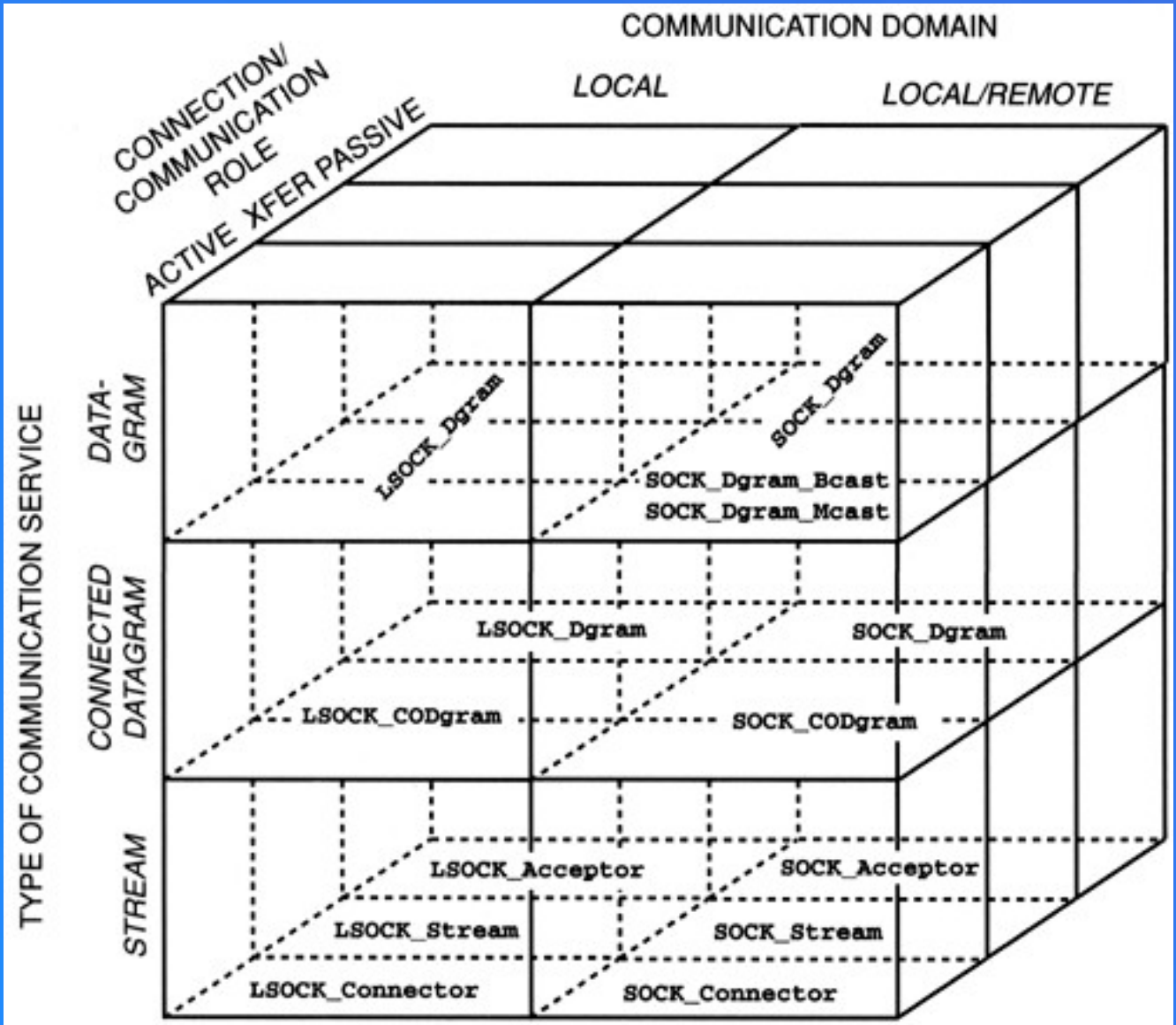
- I/O 相关对象：
  - 概要
  - ACE_INET_Addr
  - ACE_SOCK
  - ACE_SOCK_Connector
  - ACE_SOCK_Stream
  - ACE_SOCK_Acceptor
  - ACE_SOCK_Dgram

I/O 相关对象 - 概要

- ACE 的 Socket wrapper facade
  - ACE Socket wrapper facade 系列的 class，针对操作系统原生的 Socket API 进行了封装：
    - 提高类型安全
    - 保证了可移植性
    - 简化了常见的使用情况
  - 由于对关键函数做了 inline 处理，不会牺牲性能

I/O 相关对象 - 概要

- ACE 的 Socket wrapper facade 分类图



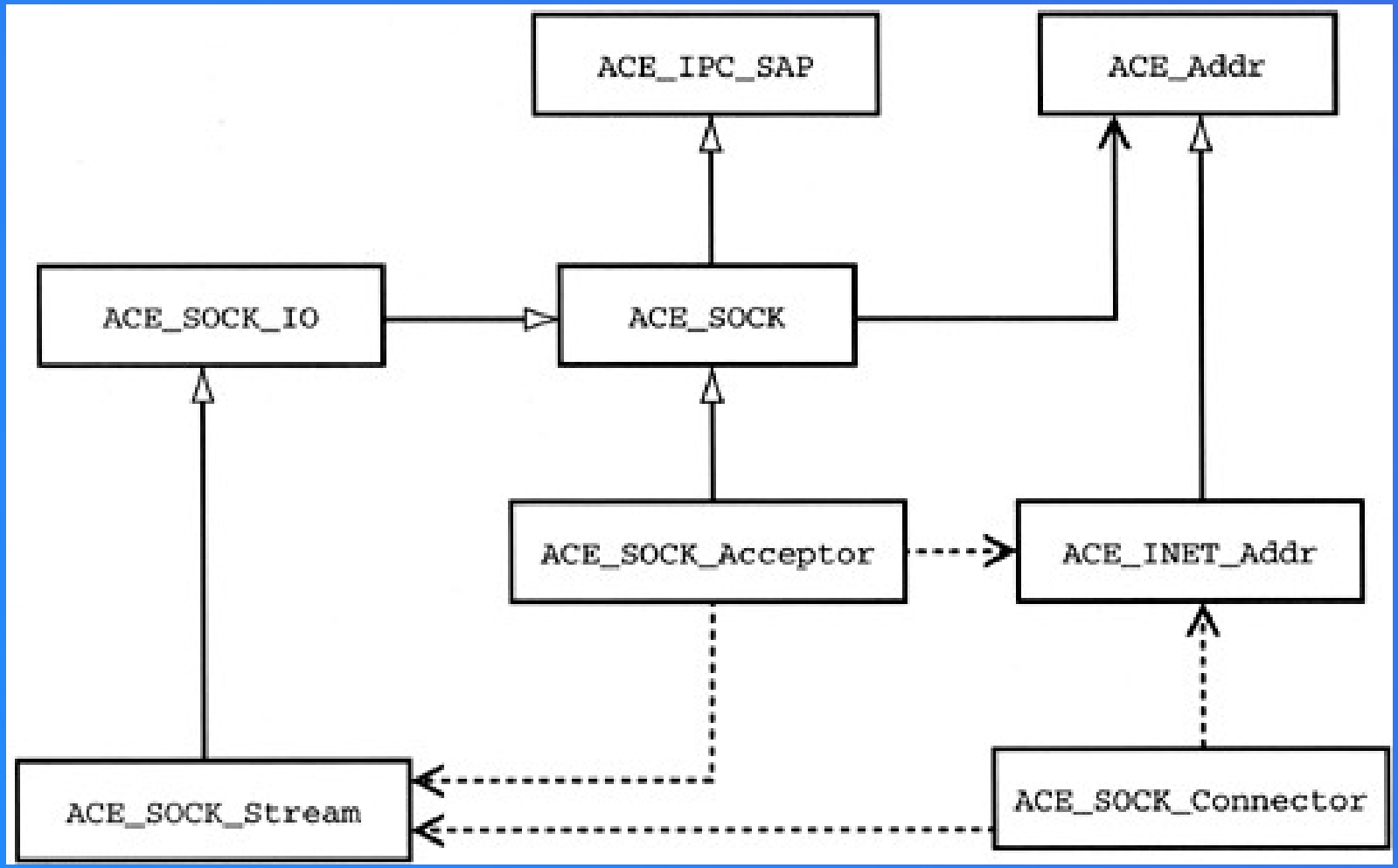
I/O 相关对象 - 概要

- ACE Internet 领域面向连接的 Socket wrapper facade 类简介
  - 以下仅讨论面向连接的 Internet 领域的部分 class（集中在 TCP 协议部分）

ACE 类	说明
ACE_Addr	ACE 网络地址的顶级基类
ACE_INET_Addr	Internet 地址族
ACE_IPC_SAP	ACE IPC wrapper facade 的顶级基类
ACE_SOCK	ACE Socket wrapper facade 的顶级基类
ACE_SOCK_Connector	一个工厂，连接到对端的接收者，然后在一个 ACE_SOCK_Stream 对象中初始化一个新的通信端点
ACE_SOCK_IO ACE_SOCK_Stream	二者封装了“数据模式（data-mode）”socket 支持的数据传输机制
ACE_SOCK_Acceptor	一个工厂，在一个 ACE_SOCK_Stream 对象中初始化一个新的通信端点，对来自对端连接者的连接请求做出回应

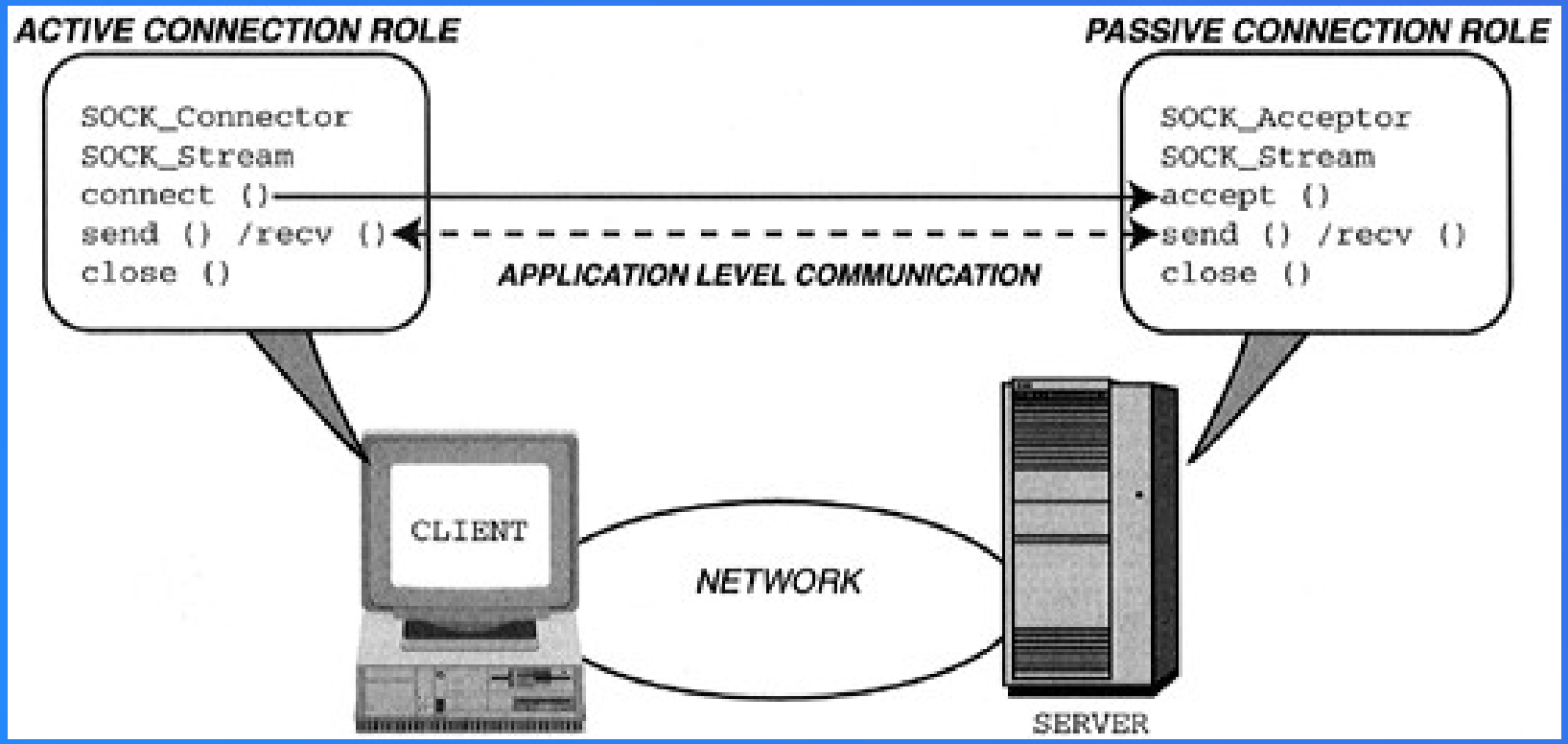
I/O 相关对象 - 概要

- ACE 的 Socket wrapper facade 类关系图示



I/O 相关对象 - 概要

- ACE 的 Socket wrapper facade 的角色



I/O 相关对象 - ACE_INET_Addr

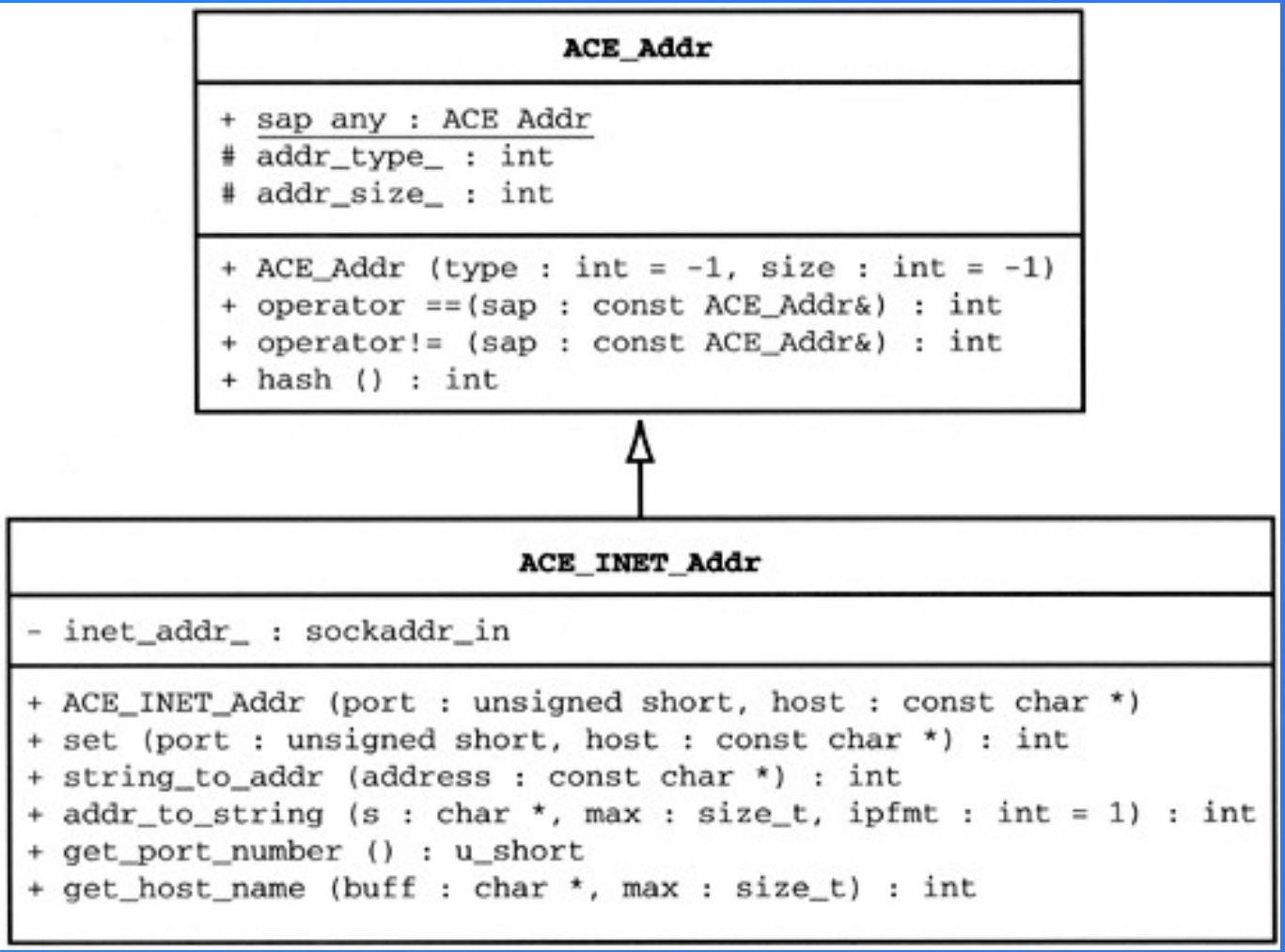
- I/O 相关对象：
  - 概要
  - ACE_INET_Addr
  - ACE_SOCK
  - ACE_SOCK_Connector
  - ACE_SOCK_Stream
  - ACE_SOCK_Acceptor
  - ACE_SOCK_Dgram

I/O 相关对象 - ACE_INET_Addr

- 关于 ACE_INET_Addr
  - ACE_INET_Addr 封装了操作系统原生的 sockaddr_in 地址结构，并且提供了众多方便的构造函数和 set() 方法，可以方便的构造一个网络地址对象，如不需要关心 port 的字节序等琐碎的问题

I/O 相关对象 - ACE_INET_Addr

- ACE_INET_Addr 类图



I/O 相关对象 - ACE_INET_Addr

- ACE_INET_Addr 示例

```
#include <iostream>
#include <ace/INET_Addr.h>
using namespace std;

int main() {
    ACE_INET_Addr addr(8868, "xuanyuan-soft.org.cn"/*,
PF_INET*/);
    char addrstr[32];
    addr.addr_to_string(addrstr, 64);
    cout << addrstr << endl;
}
```

I/O 相关对象 - ACE_SOCK

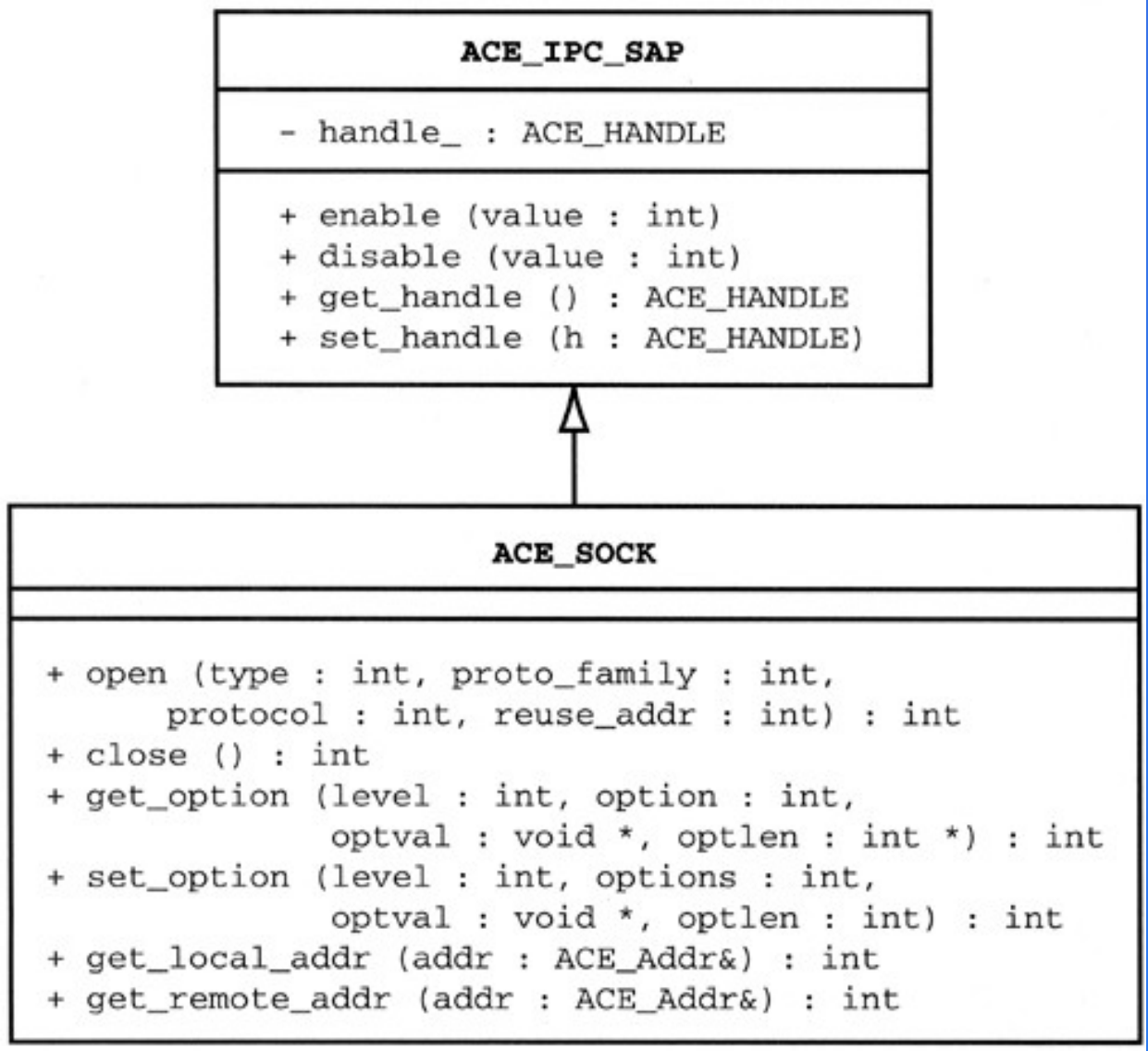
- I/O 相关对象：
  - 概要
  - ACE_INET_Addr
  - ACE_SOCK
  - ACE_SOCK_Connector
  - ACE_SOCK_Stream
  - ACE_SOCK_Acceptor
  - ACE_SOCK_Dgram

I/O 相关对象 - ACE_SOCK

- 关于 ACE_SOCK
  - ACE_SOCK 继承于 ACE_IPC_SAP，提供了：
    - 创建和销毁 socket handle
    - 获取 local 和 remote 对端的网络地址
    - 设置和读取 socket 选项，如 socket 队列的长度、提供广播或多播通信功能、禁用 Nagle 算法等
  - 该类是抽象类，不能直接实例化
  - 注意：该类没有在析构函数中关闭 socket，仅提供 close() 函数，socket 关闭的策略留给更高层次的类

I/O 相关对象 - ACE_SOCK

- ACE_SOCK 类图



I/O 相关对象 - ACE_SOCK_Connector

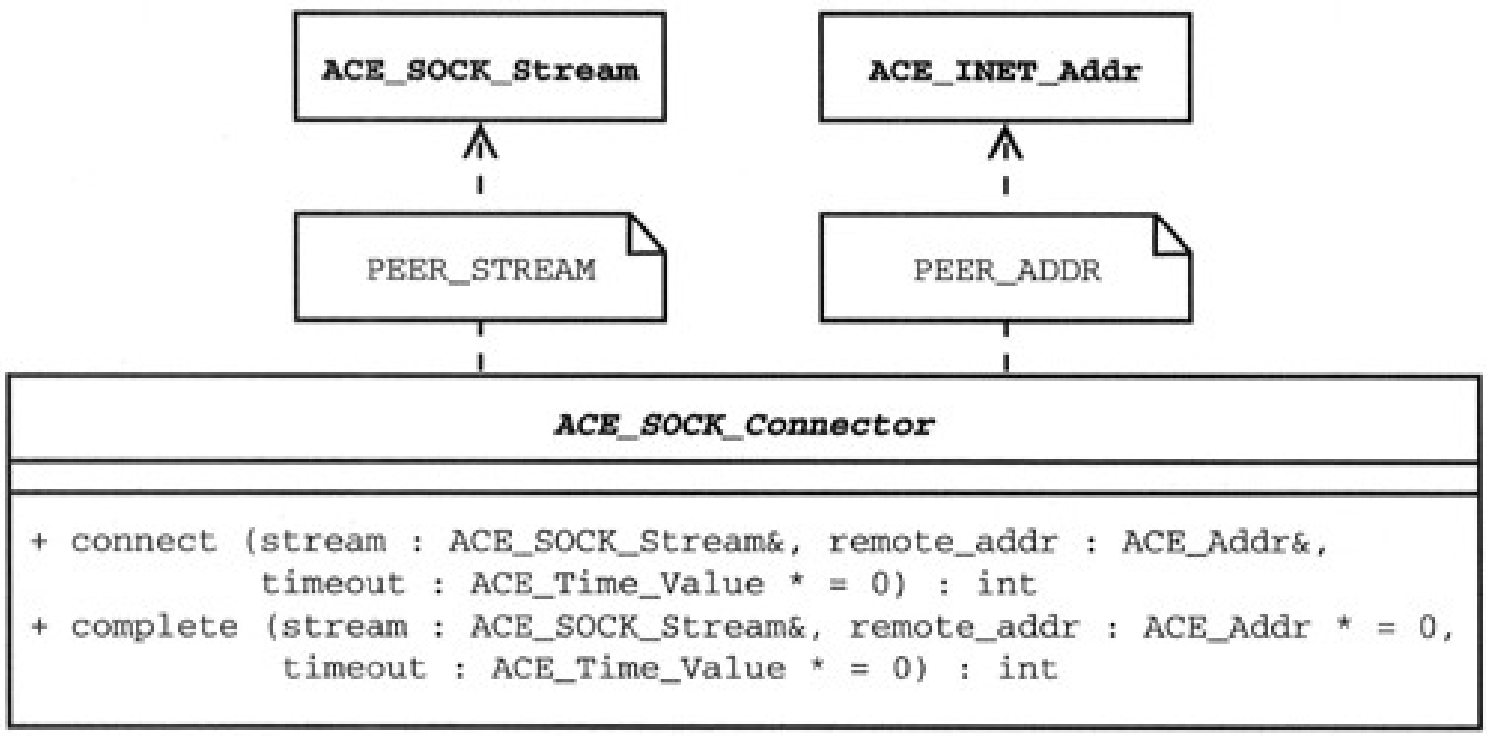
- I/O 相关对象：
  - 概要
  - ACE_INET_Addr
  - ACE_SOCK
  - ACE_SOCK_Connector
  - ACE_SOCK_Stream
  - ACE_SOCK_Acceptor
  - ACE_SOCK_Dgram

I/O 相关对象 - ACE_SOCK_Connector

- 关于 ACE_SOCK_Connector
  - ACE_SOCK_Connector 是一个模板类，提供以下功能：
    - 发起连接，且在连接建立后初始化一个 ACE_SOCK_Stream 对象
    - 连接可以通过“ blocking”、“ nonblocking”、“ time out” 方式发起

I/O 相关对象 - ACE_SOCK_Connector

- ACE_SOCK_Connector 类图



I/O 相关对象 - ACE_SOCK_Connector

- 关于 connect 操作的超时控制
  - connect() 函数支持 3 种方式：阻塞 (blocking) 、非阻塞 (nonblocking) 和定时 (time out)，默认为 blocking
  - connect() 函数的第 3 个参数的则是用于控制上述的连接方式：
    - 空 ACE_Time_Value 指针 (Null pointer)：表示阻塞调用 connect()，该操作会无限期等待连接操作成功，或直到操作系统认为无法连接而放弃
    - 非空 ACE_Time_Value 指针，但 sec() 和 usec() 均返回 0：表示 connect() 会执行非阻塞连接，如果没有立即连接成功，返回 -1，且将 errno 设为 EWOULDBLOCK
    - 非空 ACE_Time_Value 指针，但 sec() 和 usec() 返回值 >0：则表示 connect() 操作会等待指定的时间，如在指定的时间内无法建立连接，则返回 -1，且将 errno 设置为 ETIME

I/O 相关对象 - ACE_SOCK_Connector

- connect 操作示例
  - 阻塞式连接：

```
#include <ace/INET_Addr.h>
#include <ace/SOCK_Connector.h>
#include <ace/SOCK_Stream.h>

int main() {
    ACE_SOCK_Connector connector;
    ACE_SOCK_Stream peer;
    ACE_INET_Addr peerAddr;

    if (peerAddr.set(80, "xuanyuan-soft.cn") == -1)
        return 1;
    // 发起阻塞式连接操作
    else if (connector.connect(peer, peerAddr) == -1)
        return 1;
    // 接下来执行其它操作...
}
```

I/O 相关对象 - ACE_SOCK_Connector

- connect 操作示例（续 1 ）
  - 非阻塞式连接（仅描述 connect() 操作）：

```
    // 发起非阻塞式连接操作
    if (connector.connect(peer, peerAddr,
&ACE_Time_Value::zero) == -1) {
        if (errno == EWOULDBLOCK) {
            // Do some other work ...
            // Now, try to complete the connection
establishment,
            // but don't block if it isn't complete yet.
            if (connector.complete(peer, 0,
&ACE_Time_Value::zero) == -1) {
                // ...
            } else {
                // 接下来执行其它操作...
            }
        }
    } else {
        // 接下来执行其它操作...
    }
```

I/O 相关对象 - ACE_SOCK_Connector

- connect 操作示例（续 2 ）
  - 定时式连接（仅描述 connect() 操作）：

```
ACE_Time_Value timeout(10); // Set time-out to 10 seconds

    if (connector.connect(peer, peerAddr, &timeout) == -1) {
        if (errno == ETIME) {
            // Time-out, do something else...
        }
    }
```

I/O 相关对象 - ACE_SOCK_Stream

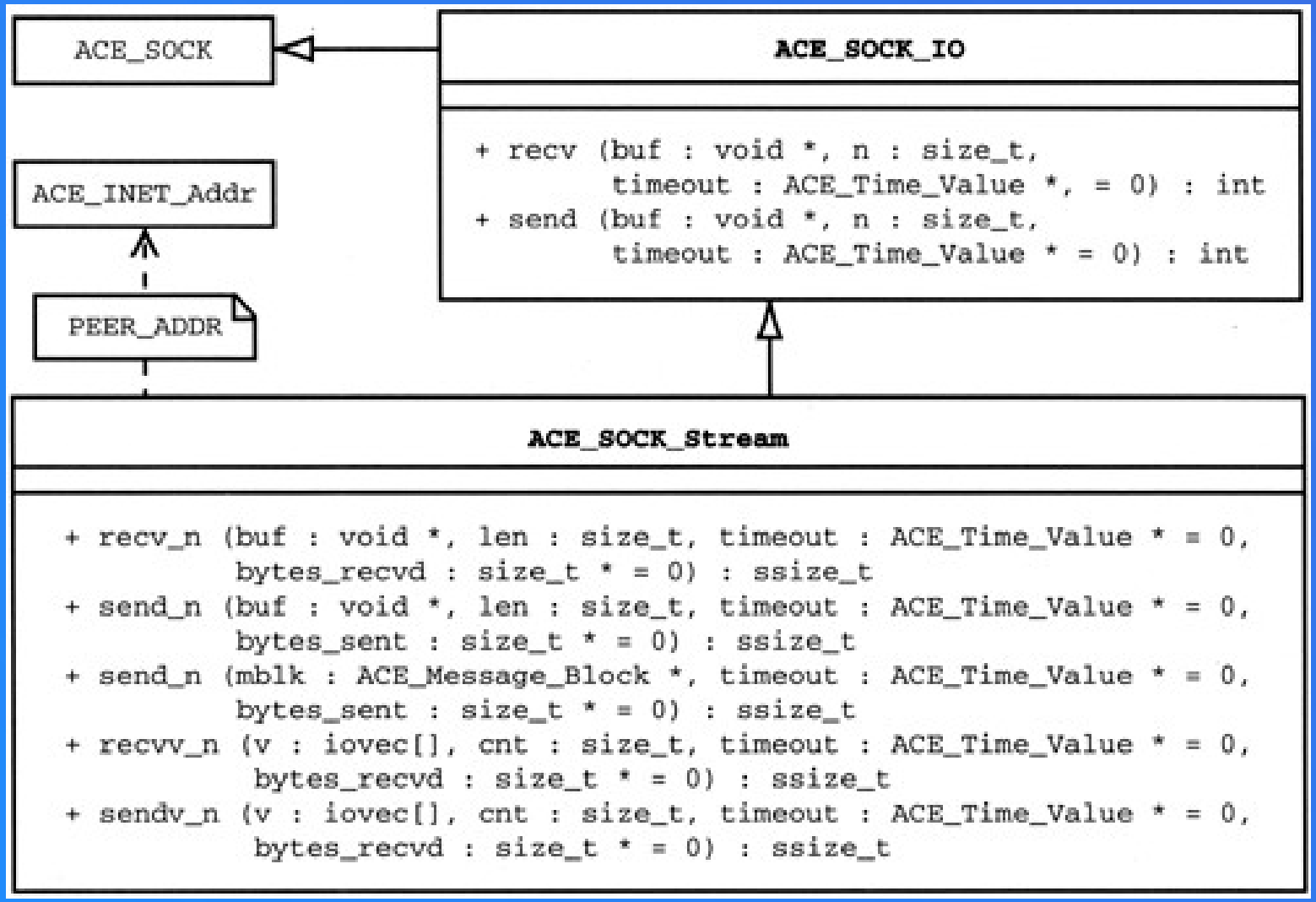
- I/O 相关对象：
  - 概要
  - ACE_INET_Addr
  - ACE_SOCK
  - ACE_SOCK_Connector
  - ACE_SOCK_Stream
  - ACE_SOCK_Acceptor
  - ACE_SOCK_Dgram

I/O 相关对象 - ACE_SOCK_Stream

- 关于 ACE_SOCK_Stream
  - ACE_SOCK_Stream 类封装了“数据模式” socket 支持的数据传输机制，提供以下功能：
    - 支持数据的发送和接收，发送的字节数最多 n 个或恰好 n 个
    - 支持“散读 (scatter-read)” 操作，这些操作涉及多个调用者提供的缓冲区，而不是单个连续的缓冲区
    - 支持“聚写 (gather-write)” 操作，在单个操作中传输多个非连续数据缓冲区的数据
    - 支持阻塞、非阻塞、定时 I/O 操作
    - 支持模板

I/O 相关对象 - ACE_SOCK_Stream

- ACE_SOCK_Stream 类图



I/O 相关对象 - ACE_SOCK_Stream

- ACE_SOCK_Stream 主要 I/O 函数

方法	说明
send() recv()	发送、接收缓冲区数据。读、写的字节数可能比请求的字节数少，因为要受 OS 的缓冲机制和传输协议中的流量控制的影响
send_n() recv_n()	发送、接收恰好 n 个字节的缓冲区数据，简化少量写入和少量读取时的应用程序处理
recvv_n()	使用 OS 的“散读”系统函数，高效、完整的接受多个缓冲区的数据
sendv_n()	使用 OS 的“聚写”系统函数，高效、完整的发送多个缓冲区的数据

I/O 相关对象 - ACE_SOCK_Stream

- ACE_SOCK_Stream 示例

```
#include <ace/INET_Addr.h>
#include <ace/SOCK_Connector.h>
#include <ace/SOCK_Stream.h>
#include <ace/Log_Msg.h>

int main() {
    ACE_SOCK_Connector connector;
    ACE_SOCK_Stream peer;
    ACE_INET_Addr peerAddr;

    if (peerAddr.set(80, "xuanyuan-soft.cn") == -1)
        ACE_ERROR_RETURN ((LM_ERROR, "%p\n", "set()"), 1);
    else if (connector.connect(peer, peerAddr) == -1)
        ACE_ERROR_RETURN ((LM_ERROR, "%p\n", "connect()"), 1);
```

I/O 相关对象 - ACE_SOCK_Stream

- ACE_SOCK_Stream 示例

```
    char buf[BUFSIZ];
    iovec iov[3];
    iov[0].iov_base = (char*)"GET ";
    iov[0].iov_len = 4; // Length of "GET ".
    iov[1].iov_base = (char*)"/index.php";
    iov[1].iov_len = 10;
    iov[2].iov_base = (char*)" HTTP/1.0\r\n\r\n";
    iov[2].iov_len = 13; // Length of " HTTP/1.0\r\n\r\n";

    if (peer.sendv_n(iov, 3) == -1)
        ACE_ERROR_RETURN ((LM_ERROR, "%p\n", "sendv_n()"), 1);

    for (ssize_t n; (n = peer.recv(buf, sizeof buf)) > 0;)
        ACE::write_n(ACE_STDOUT, buf, n);

    return peer.close();
}
```

I/O 相关对象 - ACE_SOCK_Acceptor

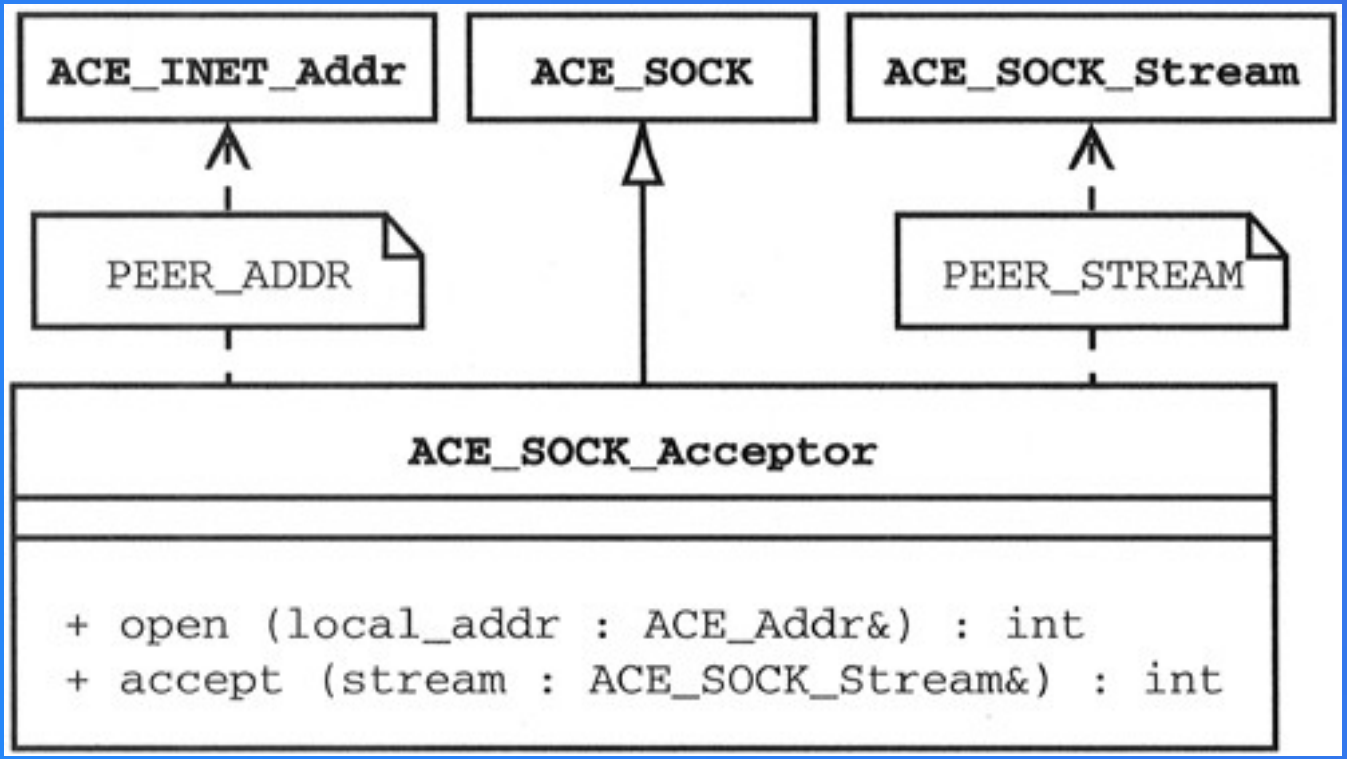
- I/O 相关对象：
  - 概要
  - ACE_INET_Addr
  - ACE_SOCK
  - ACE_SOCK_Connector
  - ACE_SOCK_Stream
  - ACE_SOCK_Acceptor
  - ACE_SOCK_Dgram

I/O 相关对象 - ACE_SOCK_Acceptor

- 关于 ACE_SOCK_Acceptor
  - ACE_SOCK_Acceptor 是一个工厂，用于被动的建立新的通信端点，提供以下功能：
    - 接受来自连接端的连接请求，并在连接建立后初始化 ACE_SOCK_Steam 对象
    - 支持阻塞、非阻塞、定时的方式接受
    - 支持模板

I/O 相关对象 - ACE_SOCK_Acceptor

- ACE_SOCK_Acceptor 类图



I/O 相关对象 - ACE_SOCK_Acceptor

- ACE_SOCK_Acceptor 示例

```
// 一个简易的 HTTP Server

#include <ace/INET_Addr.h>
#include <ace/SOCK_Acceptor.h>
#include <ace/SOCK_Stream.h>
#include <ace/Mem_Map.h>
#include <ace/Log_Msg.h>
// Return a dynamically allocated path name buffer.
extern char* get_url_pathname(ACE_SOCK_Stream *);

int main() {
    ACE_INET_Addr server_addr;
    ACE_SOCK_Acceptor acceptor;
    ACE_SOCK_Stream peer;

    if (server_addr.set(8868) == -1)
        ACE_ERROR_RETURN ((LM_ERROR, "%p\n", "set()"), 1);

    if (acceptor.open(server_addr) == -1)
        ACE_ERROR_RETURN ((LM_ERROR, "%p\n", "open()"), 1);
```

I/O 相关对象 - ACE_SOCK_Acceptor

- ACE_SOCK_Acceptor 示例（续）

```
    for (;;) {
        if (acceptor.accept(peer) == -1)
            ACE_ERROR_RETURN ((LM_ERROR, "%p\n", "accept()"), 1);
        peer.disable(ACE_NONBLOCK); // Ensure blocking <recv>s.

        char* pathname = get_url_pathname(&peer);
        // 检查 pathname 是否为NULL，略...
        ACE_Mem_Map mapped_file(pathname);

        if (peer.send_n(mapped_file.addr(), mapped_file.size())
== -1)
            ACE_ERROR_RETURN ((LM_ERROR, "%p\n", "send_n()"), 1);
        peer.close();
    }

    return acceptor.close() == -1 ? 1 : 0;
}
```

I/O 相关对象 - ACE_SOCK_Acceptor

- ACE_SOCK_Acceptor 示例（续： get_url_pathname ）

```
char* get_url_pathname(ACE_SOCK_Stream* peer) {
    static const char* docBase = "./web";
    char buf[512];
    if (peer->recv(buf, sizeof(buf)) != -1) {
        // request example:
        // GET /index.php HTTP/1.0\r\n
        const char* pos1 = strchr(buf, ' ');
        const char* pos2 = strchr(++pos1, ' ');
        const size_t len = pos2 - pos1;
        const size_t docLen = std::strlen(docBase);

        char* path = new char[len + docLen + 1];
        memcpy(path, docBase, docLen);
        memcpy(path + docLen, pos1, len);
        path[len + docLen] = '\0';
        return path;
    }
    return 0;
}
```

I/O 相关对象 - ACE_SOCK_Dgram

- I/O 相关对象：
  - 概要
  - ACE_INET_Addr
  - ACE_SOCK
  - ACE_SOCK_Connector
  - ACE_SOCK_Stream
  - ACE_SOCK_Acceptor
  - ACE_SOCK_Dgram

I/O 相关对象 - ACE_SOCK_Dgram

- 关于 ACE_SOCK_Dgram
  - ACE_SOCK_Dgram 一族 I/O 对象用于基于用户数据报 (udp) 的数据传输，为非面向连接的传输机制，按传输的端点分为：
    - 单点传输： unicast ， 一次传输的数据报仅有一个接收端
    - 多播 (组播)： multicast ， 数据报传输至一组注册在特定地址、端口的主机
    - 广播： broadcast ， 数据报的接收端为局域网中注册至某个特定端口的所有主机
  - 关于广播：
    - 数据报的广播容易引起整个局域网网络拥堵

I/O 相关对象 - ACE_SOCK_Dgram

- ACE_SOCK_Dgram 单点传播 (unicast)
  - ACE_SOCK_Dgram 和 ACE_LSOCK_Dgram
    - 这两个类为在运行在本地和 / 或远地主机上的进程间交换数据报提供机制。不像下面描述的有连接数据报，每个 send 和 recv 操作都必须为发送或接收数据报提供服务地址。
      ACE_LSOCK_Dgram 同时继承 ACE_SOCK_Dgram 和 ACE_LSOCK 的所有操作。它仅在同一主机上的进程间交换数据报。而 ACE_SOCK_Dgram 类可以在本地和 / 或远地主机上的进程间交换数据报。

I/O 相关对象 - ACE_SOCK_Dgram

- ACE_SOCK_Dgram 单点传播 (unicast) （续）
  - ACE_SOCK_CODgram 和 ACE_LSOCK_CODgram
    - 这两个类提供一种“有连接数据报”机制。不像上面所描述的无连接类，这两个类允许 send 和 recv 操作在交换数据报时省略服务地址。注意有连接数据报机制只是一种语法上的方便，因为没有其他的语义与数据传输相关联（也就是，数据递送还是不可靠的）。 ACE_SOCK_CODgram 的机制从 ACE_SOCK 基类继承。 ACE_LSOCK_CODgram 同时继承 ACE_SOCK_CODgram 和 ACE_LSOCK（它提供传递文件句柄的能力）的机制。

I/O 相关对象 - ACE_SOCK_Dgram

- ACE_SOCK_Dgram 简易 Echo Server 示例

```
#include <ace/SOCK_Dgram.h>
#include <ace/INET_Addr.h>
#include <ace/Time_Value.h>
#include <ace/Log_Msg.h>

int main(int argc, char *argv[]) {
    using namespace std;
    ACE_INET_Addr serverAddr(8868);
    ACE_SOCK_Dgram peer(serverAddr);
    char buf[512];
    for (;;) {
        memset(buf, 0, sizeof(buf));
        ACE_INET_Addr remoteAddr;
        ssize_t len = peer.recv(buf, sizeof(buf), remoteAddr);
        if (len > 0) {
            peer.send(buf, len, remoteAddr);
            ACE_DEBUG((LM_DEBUG, "%s %s\n", "recv", buf));
        }
    }
}
```

I/O 相关对象 - ACE_SOCK_Dgram

- ACE_SOCK_Dgram 简易 Echo Client 示例

```
#include <iostream>

#include <ace/SOCK_Dgram.h>
#include <ace/INET_Addr.h>
#include <ace/Time_Value.h>
#include <ace/Log_Msg.h>

int main() {
    ACE_INET_Addr remoteAddr(8868, "127.0.0.1");
    ACE_INET_Addr localAddr;
    ACE_SOCK_Dgram peer(localAddr);
```

I/O 相关对象 - ACE_SOCK_Dgram

- ACE_SOCK_Dgram 简易 Echo Client 示例

```
    char buf[128];
    for (;;) {
        memset(buf, 0, sizeof(buf));
        std::cin.getline(buf, sizeof(buf));
        if (0 == strcmp("quit", buf))
            break;
        ssize_t len = peer.send(buf, strlen(buf), remoteAddr);

        memset(buf, 0, sizeof(buf));
        len = peer.recv(buf, sizeof(buf), localAddr);
        if (len > 0)
            ACE_DEBUG((LM_DEBUG, "%s\n", buf));
    }
}
```

I/O 相关对象 - ACE_SOCK_Dgram

- ACE_SOCK_Dgram_Mcast 多播 (multicast)
  - ACE_SOCK_Dgram_Mcast
    - 该类提供的机制用于将 UDP 数据报多点传送给运行在本地子网中的本地和 / 或远地主机上的进程。该类的接口支持将数据报多点传送给特定的多点传送组。该类还将开发者与有效利用多点传送所需的低级细节屏蔽开来。

I/O 相关对象 - ACE_SOCK_Dgram

- ACE_SOCK_Dgram_Mcast 多播发送端示例

```
// 多播发送端
#include <iostream>
#include <ace/SOCK_Dgram_Mcast.h>
#include <ace/Log_Msg.h>
int main() {
    ACE_INET_Addr mcastAddr(9988, "224.0.0.16");
    ACE_INET_Addr localAddr((u_short) 0);
    ACE_SOCK_Dgram scok(localAddr);

    char buf[128];
    for (;;) {
        memset(buf, 0, sizeof(buf));
        std::cin.getline(buf, sizeof(buf));
        if (0 == strcmp("quit", buf))
            break;
        if (scok.send(buf, sizeof(buf), mcastAddr) == -1)
            ACE_ERROR_RETURN ((LM_ERROR,"%p\n", "send"), 1);
    }
}
```

I/O 相关对象 - ACE_SOCK_Dgram

- ACE_SOCK_Dgram_Mcast 多播接收端示例

```
// 多播接收端
#include <ace/SOCK_Dgram_Mcast.h>
#include <ace/Log_Msg.h>
int main() {
    ACE_INET_Addr mcastAddr(9988, "224.0.0.16");
    ACE_INET_Addr remoteAddr((u_short) 0);
    ACE_SOCK_Dgram_Mcast scok;
    if (scok.join(mcastAddr) == -1)
        ACE_ERROR_RETURN ((LM_ERROR,"%p\n", "join"), 1);
    char buf[512];
    for (;;) {
        memset(buf, 0, sizeof(buf));
        if (scok.recv(buf, sizeof(buf), remoteAddr) != -1)
            ACE_DEBUG((LM_DEBUG, "recv msg from %s:%d: %s\n",
                                remoteAddr.get_host_addr(),
                                remoteAddr.get_port_number(), buf));
    }
    if (scok.leave(mcastAddr) == -1)
        ACE_ERROR_RETURN ((LM_ERROR,"%p\n", "leave"), 1);
}
```

I/O 相关对象 - ACE_SOCK_Dgram

- 关于多播地址
  - 多播地址是一组保留的 D 类地址：
    - Link-local addresses: 224.0.0.0~224.0.0.255 由一些网络协议在本网段使用，路由器不转发这些报文
    - Administratively scoped addresses: 239.0.0.0~239.255.255.255 用于私有的多播域中，类似于私有 IP 地址
    - Globally scoped addresses: 224.0.1.0~238.255.255.255 可供任何实体使用，这些地址可以在一个组织内部或者 Internet 上路由，因此它们必须是全局唯一的
    - Source specific multicast: 232.0.0.0~232.255.255.255，用于指定源的多播（ Source specific Multicast ）， SSM 是 PIM 协议中的扩展
    - GLOP addresses: 233.0.0.0~233.255.255.255，保留给组织静态定义的多播地址

I/O 相关对象 - ACE_SOCK_Dgram

- ACE_SOCK_Dgram_Bcast 广播 (broadcast)
  - ACE_SOCK_Dgram_Bcast
    - 该类提供的机制用于将 UDP 数据报广播给本地子网中的本地和 / 或远地主机。该类的接口支持将数据报广播给（ 1 ）所有与主机相连的网络接口，或是（ 2 ）一个特定的网络接口。该类还将开发者与有效利用广播所需的低级细节屏蔽开来。

I/O 相关对象 - ACE_SOCK_Dgram

- ACE_SOCK_Dgram_Bcast 广播发送端示例

```
// 广播发送端
#include <iostream>
#include <ace/SOCK_Dgram_Bcast.h>
#include <ace/Log_Msg.h>

int main() {
    ACE_SOCK_Dgram_Bcast sock(ACE_Addr::sap_any);
    u_short port = 8868;

    char buf[128];
    for (;;) {
        memset(buf, 0, sizeof(buf));
        std::cin.getline(buf, sizeof(buf));
        if (0 == strcmp("quit", buf))
            break;
        if (sock.send(buf, strlen(buf), port) == -1)
            ACE_ERROR_RETURN ((LM_ERROR,"%p\n", "send"), 1);
    }
}
```

I/O 相关对象 - ACE_SOCK_Dgram

- ACE_SOCK_Dgram_Bcast 广播接收端示例

```
// 广播接收端
#include <ace/SOCK_Dgram_Bcast.h>
#include <ace/Log_Msg.h>

int main() {
    ACE_INET_Addr localAddr(8868);
    ACE_INET_Addr remoteAddr;
    ACE_SOCK_Dgram_Bcast scok(localAddr);

    char buf[512];
    for (;;) {
        memset(buf, 0, sizeof(buf));
        if (scok.recv(buf, sizeof(buf), remoteAddr) != -1)
            ACE_DEBUG((LM_DEBUG, "recv msg from %s:%d: %s\n",
                                 remoteAddr.get_host_addr(),
                                 remoteAddr.get_port_number(), buf));
    }
}
```

I/O 相关对象 - 小结

- 相对于操作系统中传统的网络 I/O 对象及相关的操作， ACE 提供了更安全、更统一的编程接口，同时基于模板的实现，更容易在不同 I/O 对象之间切换，实现不同的功能、而不需大量更改原有的代码。
- 本单元所介绍的 I/O 对象是构建后续众多框架的基础，如 Reactor 框架、 Acceptor-Connector 框架等