boost::asio是一个高性能的网络开发库，Windows下使用IOCP，Linux下使用epoll。与ＡＣＥ不同的是，它并没有提供一个网络框架，而是采取组件的方式来提供应用接口。但是对于常见的情况，采用一个好用的框架还是能够简化开发过程，特别是asio的各个异步接口的用法都相当类似。  
　　受到　[SP Server](http://iunknown.javaeye.com/blog/59804) 框架的影响，我使用asio大致实现了一个多线程的半异步半同步服务器框架，以下是利用它来实现一个Echo服务器：  
  
1.　实现回调：

static void onSessionStarted(SessionPtr const& session)  
{  
       session->setReadMode(Session::READ\_LINE); // 设置为行读取  
}  
http://www.cppblog.com/Images/OutliningIndicators/None.gif  
static void onSession( SessionPtr const& session)  
{  
       print(session->getMsg()); //打印收到的消息  
       session->addReply(session->getMsg());　//回送消息  
       session->close();  
}  
http://www.cppblog.com/Images/OutliningIndicators/None.gif

说明：close()是一个关闭请求，它并不马上关闭Session，而是等待所有与该Session相关的异步操作全部结束后才关闭。  
  
2.　一个单线程的Echo服务器：

void server\_main()  
{  
    unsigned short port = 7;  
    AsioServer server(port);  
  
    server.sessionStarted = &onSessionStarted;  
    server.sessionHandle = &onSession;  
  
    server.run();  
}

3.　一个多线程的Echo服务器(半异步半同步：一个主线程，４个工作者线程）

void server\_main2()  
{  
    unsigned short port = 7;  
    AsioServer server(port);  
    AsioService worker;  
  
    server.sessionStarted = worker.wrap(&onSessionStarted);  
    server.sessionHandle = worker.wrap(&onSession);  
  
     int num\_threads = 4;  
     server.runWithWorkers(worker, num\_threads);  
}

　　其中主要用到了两个类，Session和AsioServer，大约有以下接口：

struct Session  
{  
    void setReadMode(int mode);  
    string getMsg();  
      
    void addReply(string const&);  
      
    void close();  
    void post();    // stop pending  
      
    string     getError();  
      
    Endpoint localEndpoint();  
    Endpoint remoteEndpoint();  
      
   shared\_ptr<void>    userData;  
};

struct AsioService  
{  
    void run();    
    void stop();   
    void post(function<void()>const&);   
      
    template <typename F> SessionWrapper<F> wrap(F const& f);  
};  
  
struct AsioServer : public AsioService  
{  
    typedef boost::function<void(SessionPtr const&)> callback\_t;  
  
    AsioServer(unsigned short port);  
      
    callback\_t   sessionStarted;  
    callback\_t   sessionHandle;  
    callback\_t   sessionClosed;  
    callback\_t   sessionError;  
    function<std::string>   error;  
  
    void runWithWorkers(AsioService& worker, size\_t num\_threads); // HS-HA pattern  
};

　　有了这样一个思路，实现起来就很容易了。重点是以下两点：  
　　１。缓冲区的管理与内存池的使用  
　　２。为了保证Session的线程安全，必须要设置一个挂起状态。  
           
  　　还有一个好处，就是完全隔绝了asio的应用接口，不用再忍受asio漫长的编译时间了。代码就不贴在这里了，有兴趣的可以通过email 探讨。