Module06-05
C++ ACE: Acceptor-Connector 框架

C++ ACE - Acceptor-Connector 框架

- 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

Acceptor-Connector 框架 - 概要

- Acceptor-Connector 框架：
  - 概要
  - ACE_Svc_Handler
  - ACE_Acceptor
  - ACE_Connector

Acceptor-Connector 框架 - 概要

- 关于 Acceptor-Connector 框架
  - ACE Acceptor-Connector 框架实现了 Acceptor-Connector 模式，这种模式通过解除：
    - 网络化应用中相互协作的对等服务的连接和初始化所需的活动
    - 以及它们一旦连接和初始化后所执行的处理

    的耦合，增强了软件复用和可扩展性。

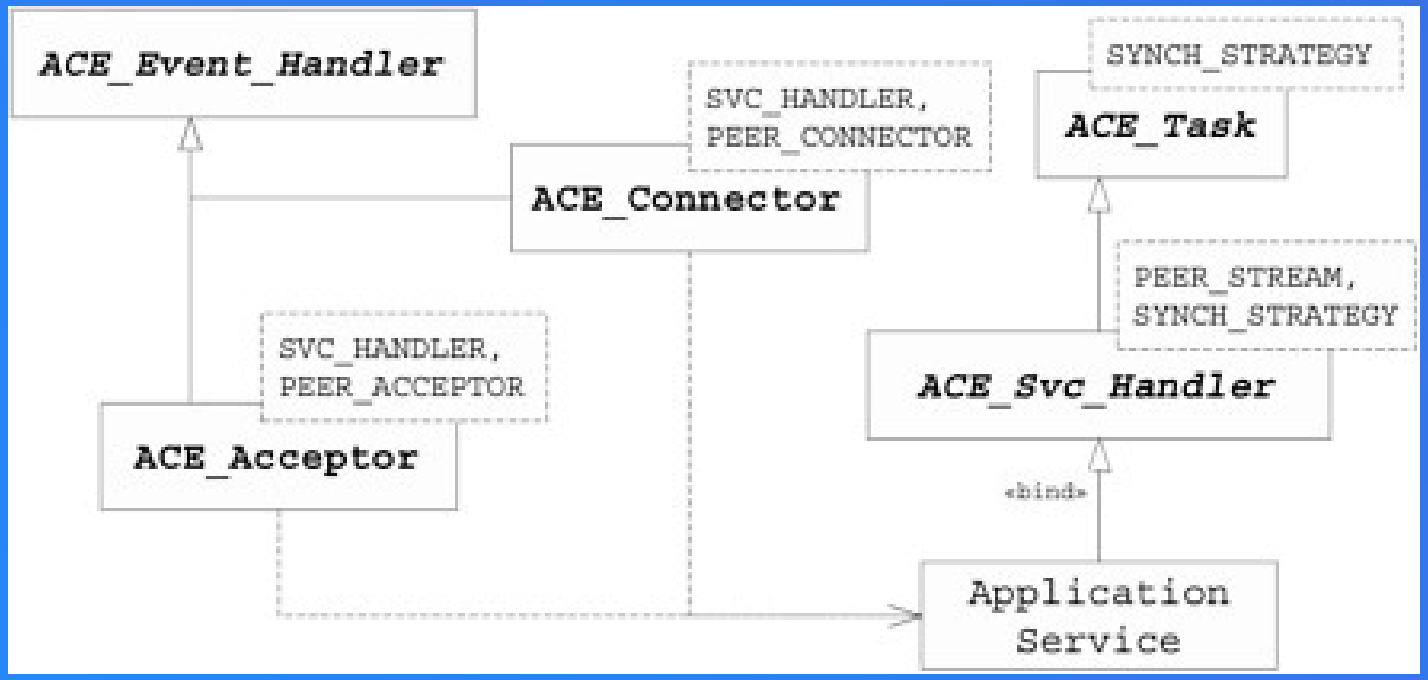
Acceptor-Connector 框架 - 概要

- Acceptor-Connector 框架的主要参与者

ACE 类	说明
ACE_Svc_Handler	表示某个已连接服务的本地端，其中含有一个用于与连接对端通信的 IPC 端点
ACE_Acceptor	该工厂被动的等待接受连接，并随即初始化一个 ACE_Svc_Handler 来响应来自对端的主动连接请求
ACE_Connector	该工厂主动的连接带对端接受器，并随即初始化一个 ACE_Svc_Handler 来与其相连的对端通信

Acceptor-Connector 框架 - 概要

- Acceptor-Connector 框架 类关系图



Acceptor-Connector 框架 - ACE_Svc_Handler

- Acceptor-Connector 框架：
  - 概要
  - ACE_Svc_Handler
  - ACE_Acceptor
  - ACE_Connector

Acceptor-Connector 框架 - ACE_Svc_Handler

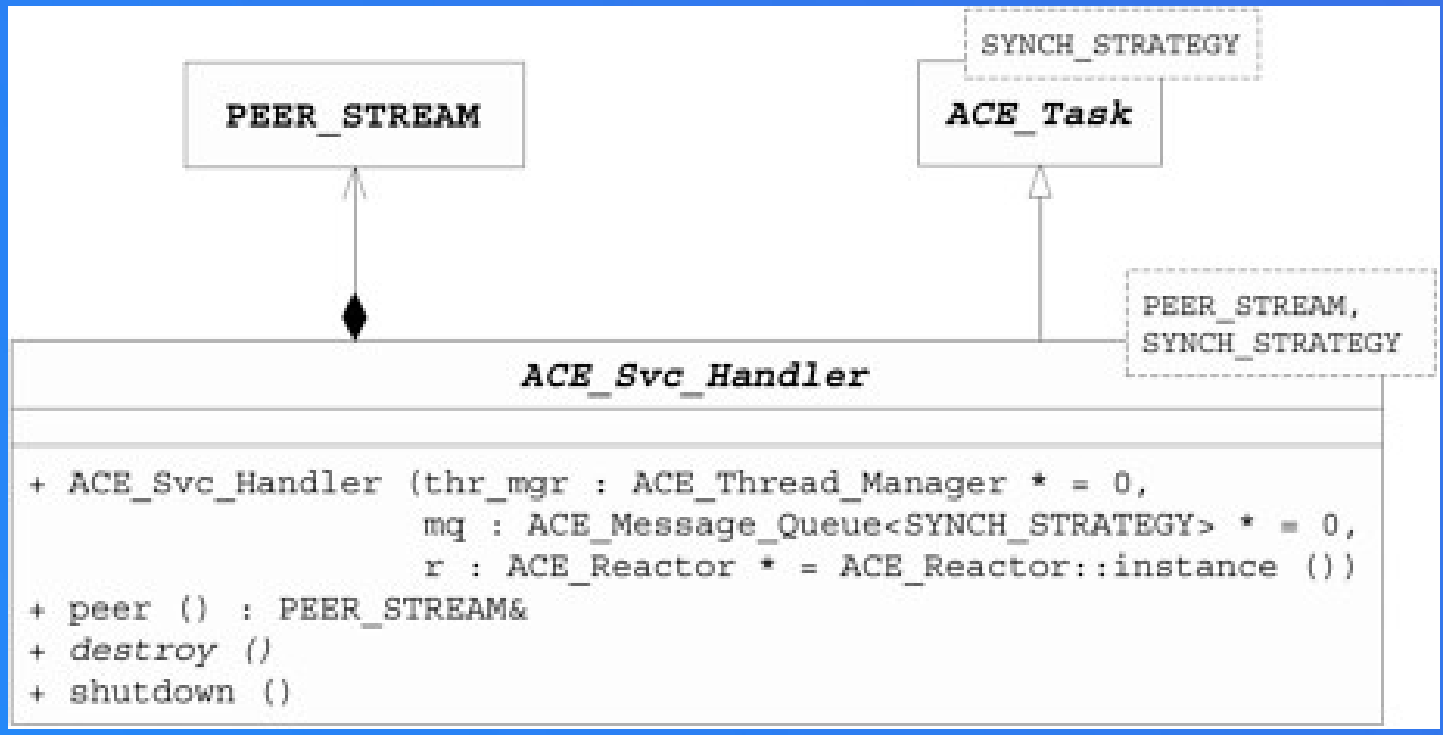
- 关于 ACE_Svc_Handler
  - ACE_Svc_Handler 是 ACE 的同步和反应式数据传输以及服务处理机制的基础，这个类提供了以下能力：
    - 为同步和 / 或反应式网络化应用中的服务初始化和实现提供了基础，并充当 ACE_Acceptor 和 ACE_Connector 连接工厂的目标
    - 为服务处理器提供了用于与其对端服务处理器进行通信的 IPC 端点。这种 IPC 端点的类型可以通过 ACE 的多种 IPC 封装类来参数化，从而将低级的通信机制与应用级服务处理策略分离开来
    - 由于派生自 ACE_Task(而 ACE_Task 又派生自 ACE_Event_Handler)，它继承了并发、同步、动态配置和事件处理能力

Acceptor-Connector 框架 - ACE_Svc_Handler

- 关于 ACE_Svc_Handler（续）
  - 使最为常见的反应式网络服务活动代码化，如在服务被打开时登记到反应器，在服务解除反应器上的登记时关闭 IPC 端点

Acceptor-Connector 框架 - ACE_Svc_Handler

- ACE_Svc_Handler 类图



Acceptor-Connector 框架 - ACE_Svc_Handler

- ACE_Svc_Handler 服务创建和激活方法

方法	说明
ACE_Svc_Handler()	connector 或 acceptor 创建 service handler 后调用
open()	挂钩方法，自动被 connector 或 acceptor 调用来初始化 service handler

  - 与 ACE_Event_Handler 不同， ACE_Svc_Handler 的默认 open() 定义中实现了几个常用的操作，如注册到 Reactor 等

```
template<class PEER_STREAM, class SYNCH_STRATEGY>
int ACE_Svc_Handler<PEER_STREAM, SYNCH_STRATEGY>::open(void
*factory) {
    if (reactor() && reactor()->register_handler(this,
            ACE_Event_Handler::READ_MASK) == -1)
        return -1;
    else
        return 0;
}
```

Acceptor-Connector 框架 - ACE_Svc_Handler

- ACE_Svc_Handler 服务处理方法

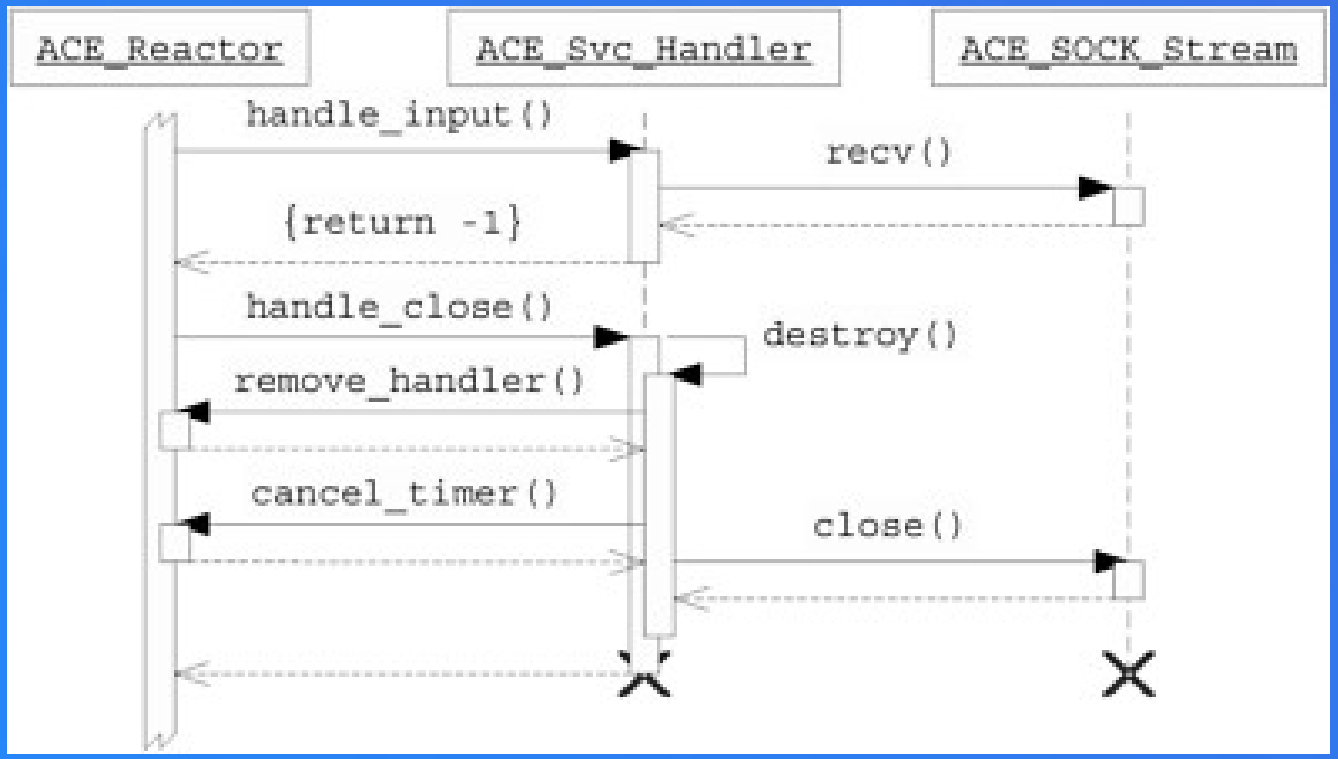
方法	说明
svc()	从 ACE_Task 继承而来
handle_*()	从 ACE_Event_Handler 继承而来
peer()	返回指向底层 PERR_STREAM 的引用， ACE_Svc_Handler 的 PEER_STREAM 在其 open() 挂钩方法被调用时就绪。任何服务处理方法都可使用这个方法来获取指向已连接的 IPC 机制的引用

- ACE_Svc_Handler 服务关闭方法

方法	说明
destroy()	可以直接用于关闭 ACE_Svc_Handler 实例 注意：如果该 Svc_Handler 向 Reactor 登记过，则不要在 Reactors 运行事件循环之外的线程中调用该方法关闭 Svc_Handler，而应该采用在某个 handle_*() 方法中返回 -1（比如 handle_input()），指示 Reactor 调用 handle_close() 来关闭该 Svc_Handler
handle_close()	通过 Reactor 的回调调用 destroy()
close()	在 ACE_Svc_Handler 实例退出时调用 handle_close()

Acceptor-Connector 框架 - ACE_Svc_Handler

- ACE_Svc_Handler 反应式关闭方法示意



Acceptor-Connector 框架 - ACE_Acceptor

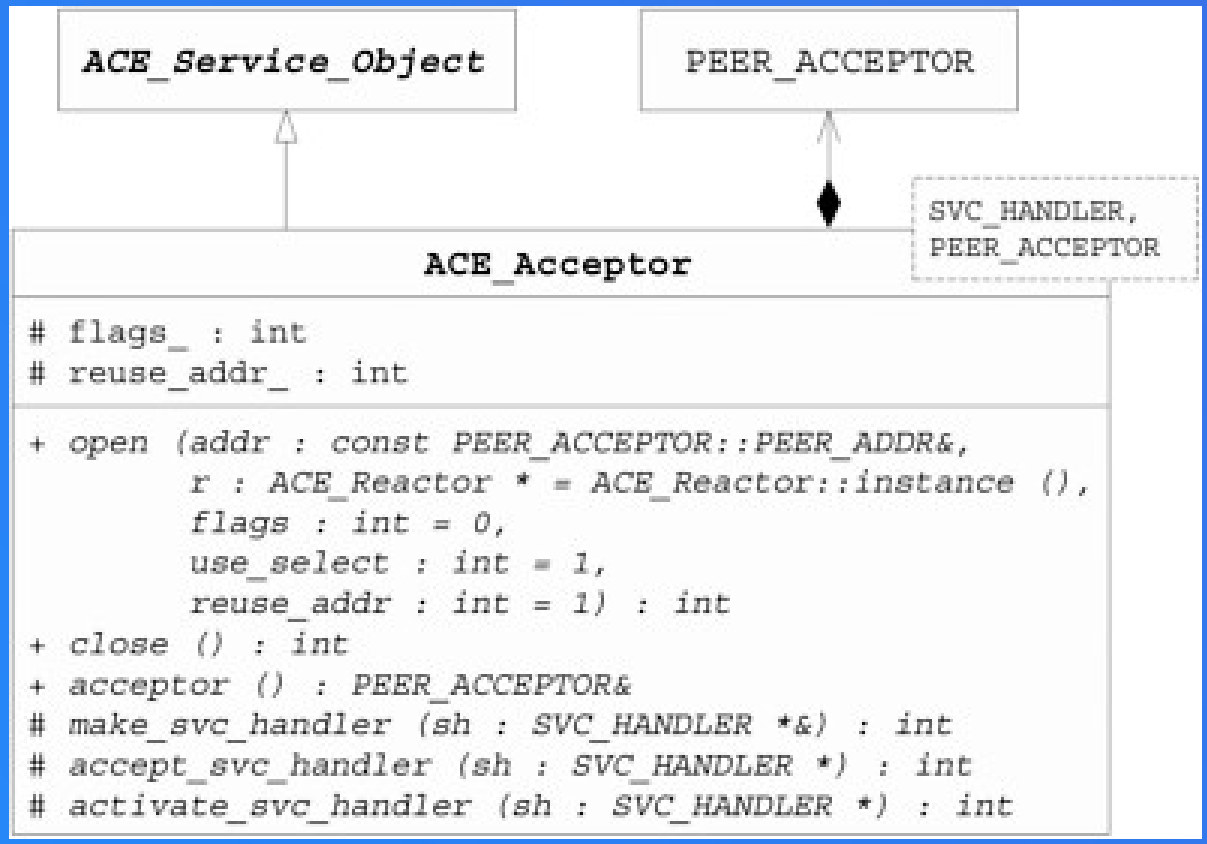
- Acceptor-Connector 框架：
  - 概要
  - ACE_Svc_Handler
  - ACE_Acceptor
  - ACE_Connector

Acceptor-Connector 框架 - ACE_Acceptor

- 关于 ACE_Acceptor
  - ACE_Acceptor 是一个工厂，这个类提供了以下能力：
    - 解除了 被动连接建立和服务初始化逻辑 与 服务处理器和初始化之后所执行的处理 的耦合
    - 它提供了一个被动模式的 IPC 端点，用于侦听和接受来自对端的连接，这个 IPC 端点的类型可以通过 ACE 的多种 IPC Wrapper facade 类来参数化，从而使较低级的连接机制与应用级服务初始化策略分离
    - 使 被动的连接 IPC 端点、创建 / 激活与其相关联的服务处理器 所必需的各个步骤得以自动化
    - 由于 ACE_Acceptor 派生自 ACE_Service_Object ， 继承了事件处理和配置能力

Acceptor-Connector 框架 - ACE_Acceptor

- ACE_Acceptor 类图



Acceptor-Connector 框架 - ACE_Acceptor

- ACE_Acceptor 构造、析构、初始化方法

方法	说明
ACE_Acceptor() open()	将接受器的被动模式 IPC 端点绑定到特定的地址，如 TCP 端口号和 IPC 主机地址，然后对连接请求的到达进行侦听
~ACE_Acceptor() close()	关闭接受器的 IPC 端点，释放其资源
acceptor()	返回指向底层 PERR_ACCEPTOR 的引用

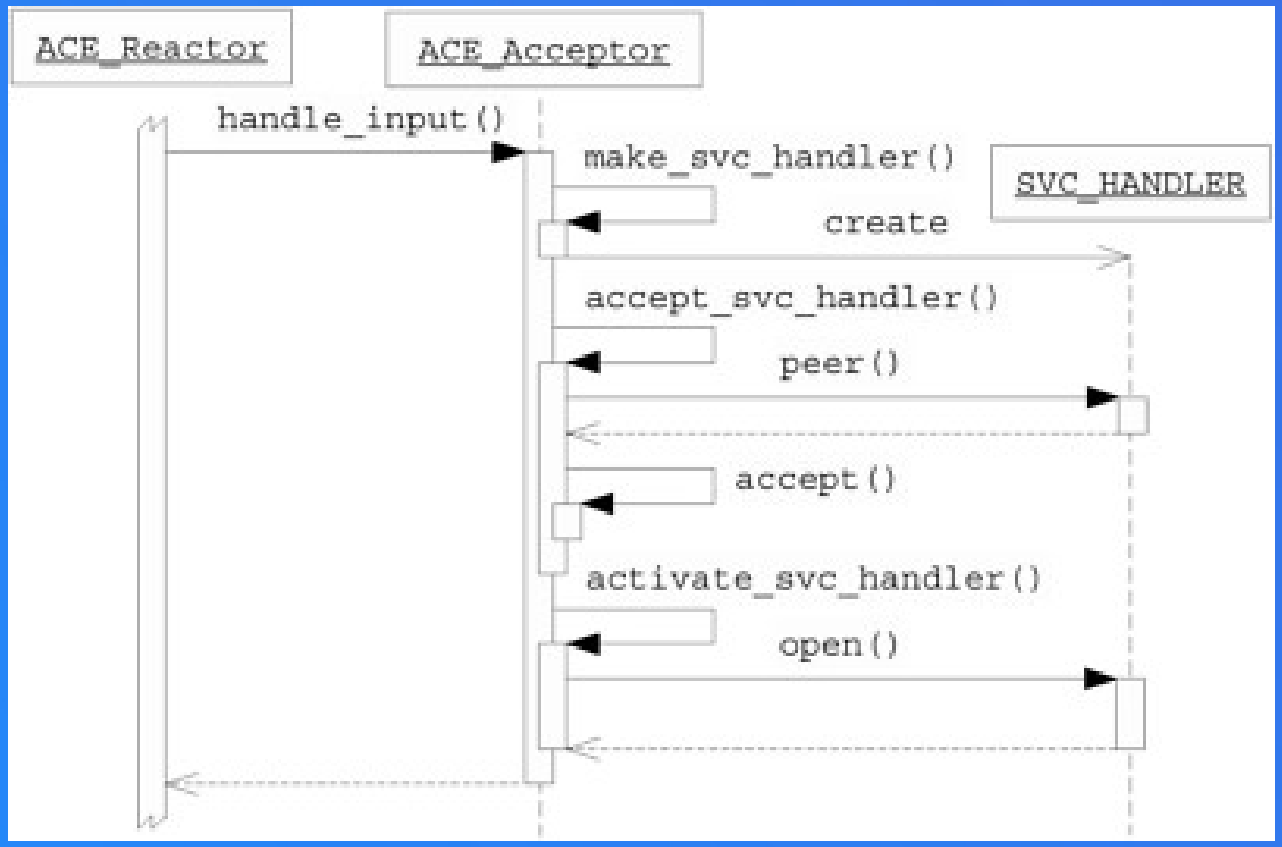
Acceptor-Connector 框架 - ACE_Acceptor

- ACE_Acceptor 连接建立和服务处理器初始化方法

方法	说明
handle_input()	当连接请求从对端连接器到达时，反应器会调用该模板方法，它可以使用下面的 3 个方法来使 被动的连接 IPC 端点并创建和激活与其相关联的服务处理器 所必需的步骤分离
make_svc_handler()	这个工厂方法创建服务处理器来处理通过其已连接的 IPC 端点、从对端服务发出的数据请求
accept_svc_handler()	这个挂钩方法使用接受器的被动模式 IPC 端点来创建已连接的 IPC 端点，并将此端点与服务处理器相关联的一个 I/O 句柄关联在一起
activate_svc_handler()	这个挂钩方法调用服务处理器的 open() 方法，让服务处理器完成对自己的初始化

Acceptor-Connector 框架 - ACE_Acceptor

- handle_input() 方法三个变化点图示



Acceptor-Connector 框架 - ACE_Acceptor

- handle_input() 方法三个变化点
  - 1，服务处理器的创建
    handle_input() 调用 make_svc_handler() 工厂挂钩方法创建新的服务处理器， make_svc_handler() 方法的默认实现：

```
template<class SVC_HANDLER, class PEER_ACCEPTOR>
int ACE_Acceptor<SVC_HANDLER,
PEER_ACCEPTOR>::make_svc_handler(SVC_HANDLER *&sh) {
    ACE_NEW_RETURN(sh, SVC_HANDLER, -1);
    sh->reactor(reactor());
    return 0;
}
```

- handle_input() 方法三个变化点（续 1 ）
  - 2，连接建立
    handle_input() 调用 accept_svc_handler() 挂钩方法被动的接受来自对端连接器的连接，该方法的默认实现将处理委托给 PEER_ACCEPTOR::accecpt()， accept_svc_handler() 方法的默认实现：

```
template<class SVC_HANDLER, class PEER_ACCEPTOR>
int ACE_Acceptor<SVC_HANDLER,
PEER_ACCEPTOR>::accept_svc_handler(SVC_HANDLER *sh) {
    if (acceptor().accept(sh->peer()) == -1) {
        sh->close(0);
        return -1;
    }
    return 0;
}
```

- handle_input() 方法三个变化点（续 2 ）
  - 3，服务处理器激活
    handle_input() 调用 activate_svc_handler() 激活代表新连接的服务处理器：

```
template<class SVC_HANDLER, class PEER_ACCEPTOR>
int ACE_Acceptor<SVC_HANDLER,
PEER_ACCEPTOR>::activate_svc_handler(
        SVC_HANDLER *sh) {
    int result = 0;
    if (ACE_BIT_ENABLED(flags_, ACE_NONBLOCK)) {
        if (sh->peer().enable(ACE_NONBLOCK) == -1)
            result = -1;
    } else if (sh->peer().disable(ACE_NONBLOCK) == -1)
        result = -1;
    if (result == 0 && sh->open(this) == -1)
        result = -1;
    if (result == -1) sh->close(0);
    return result;
}
```

Acceptor-Connector 框架 - ACE_Acceptor

- 使用 ACE_Acceptor （ Handler ）

```
#include <ace/Svc_Handler.h>
#include <ace/SOCK_Stream.h>
class AcceptorHandler:
    public ACE_Svc_Handler<ACE_SOCK_Stream, ACE_NULL_SYNCH> {
public:
  typedef ACE_Svc_Handler<ACE_SOCK_Stream,ACE_NULL_SYNCH> Parent;
  enum { BUF_SIME = 512 };

  virtual int AcceptorHandler::handle_input(ACE_HANDLE h) {
     ssize_t n = peer().recv(buf, BUF_SIME);
     if (n <= 0)
       ACE_ERROR_RETURN((LM_ERROR, "%p\n", "peer().recv()"), -1);
     if (peer().send(buf, n) == -1)
       ACE_ERROR_RETURN((LM_ERROR, "%p\n", "peer().send()"), -1);
     return 0;
  }
private:
  char buf[BUF_SIME];
};
```

Acceptor-Connector 框架 - ACE_Acceptor

- 使用 ACE_Acceptor （ Acceptor 和 main() ）

```
typedef ACE_Acceptor<AcceptorHandler, ACE_SOCK_Acceptor>
MyAcceptor;

int main() {
    ACE_INET_Addr addr(8868);
    MyAcceptor acceptor(addr, ACE_Reactor::instance());

    ACE_Reactor::instance()->run_reactor_event_loop();
}
```

Acceptor-Connector 框架 - ACE_Connector

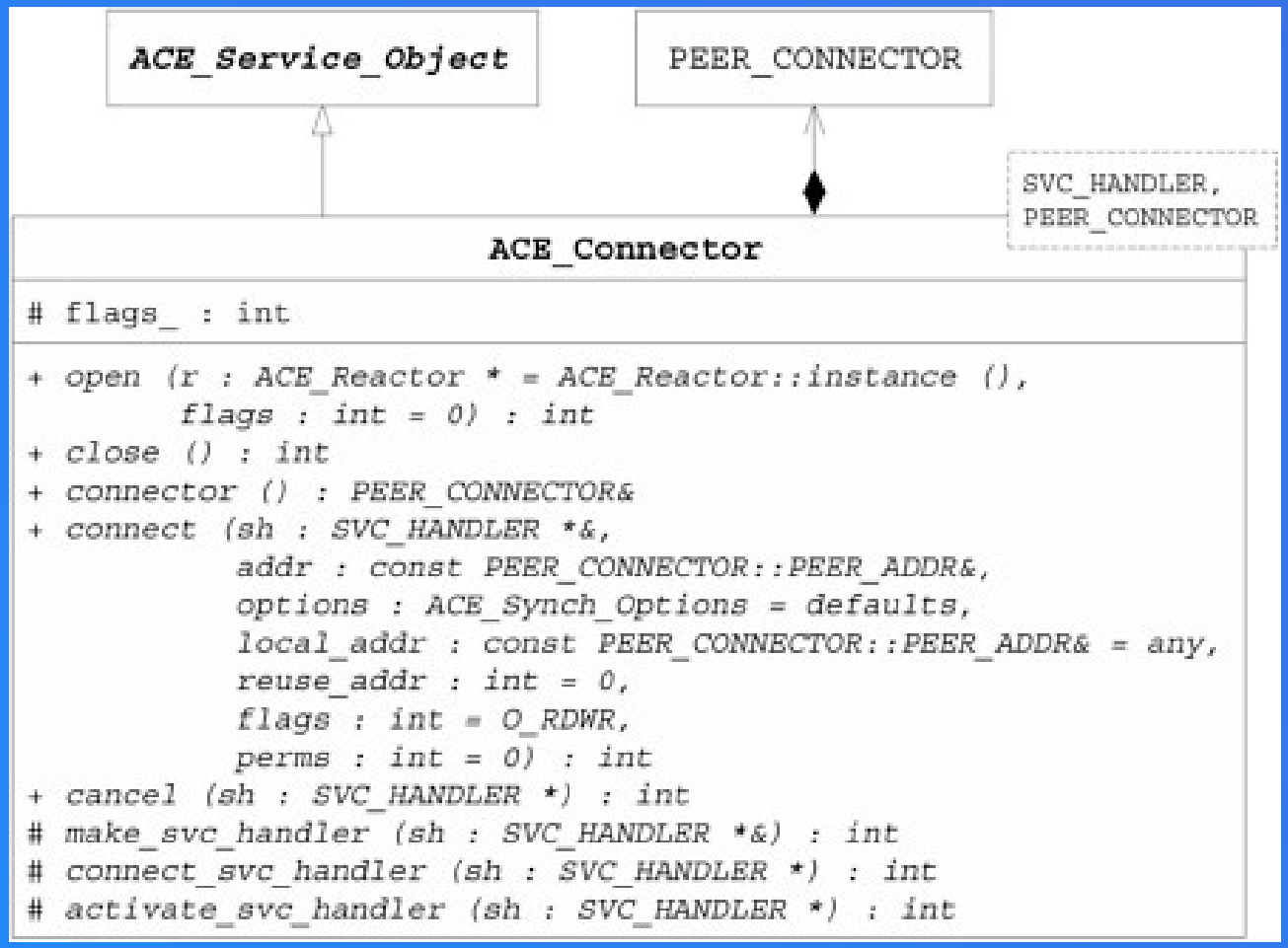
- Acceptor-Connector 框架：
  - 概要
  - ACE_Svc_Handler
  - ACE_Acceptor
  - ACE_Connector

Acceptor-Connector 框架 - ACE_Connector

- 关于 ACE_Connector
  - ACE_Connector 是一个工厂，这个类提供了以下能力：
    - 解除了 主动连接建立和服务初始化逻辑 与 服务处理器和初始化之后所执行的处理 的耦合
    - 它提供了一个 IPC 工厂，可以同步或反应式的主动与对端接受器建立连接，可以通过 ACE 多种 IPC wrapper facede 类类参数化这个 IPC 端点的类型，从而将较低级的连接机制与应用级服务初始化策略分离
    - 使 主动的连接 IPC 端点、创建 / 激活与其相关联的服务处理器 所必需的各个步骤得以自动化
    - 由于 ACE_Connector 派生自 ACE_Service_Object，继承了事件处理和配置能力

Acceptor-Connector 框架 - ACE_Connector

- ACE_Connector 类图



Acceptor-Connector 框架 - ACE_Connector

- ACE_Connector 初始化、析构方法

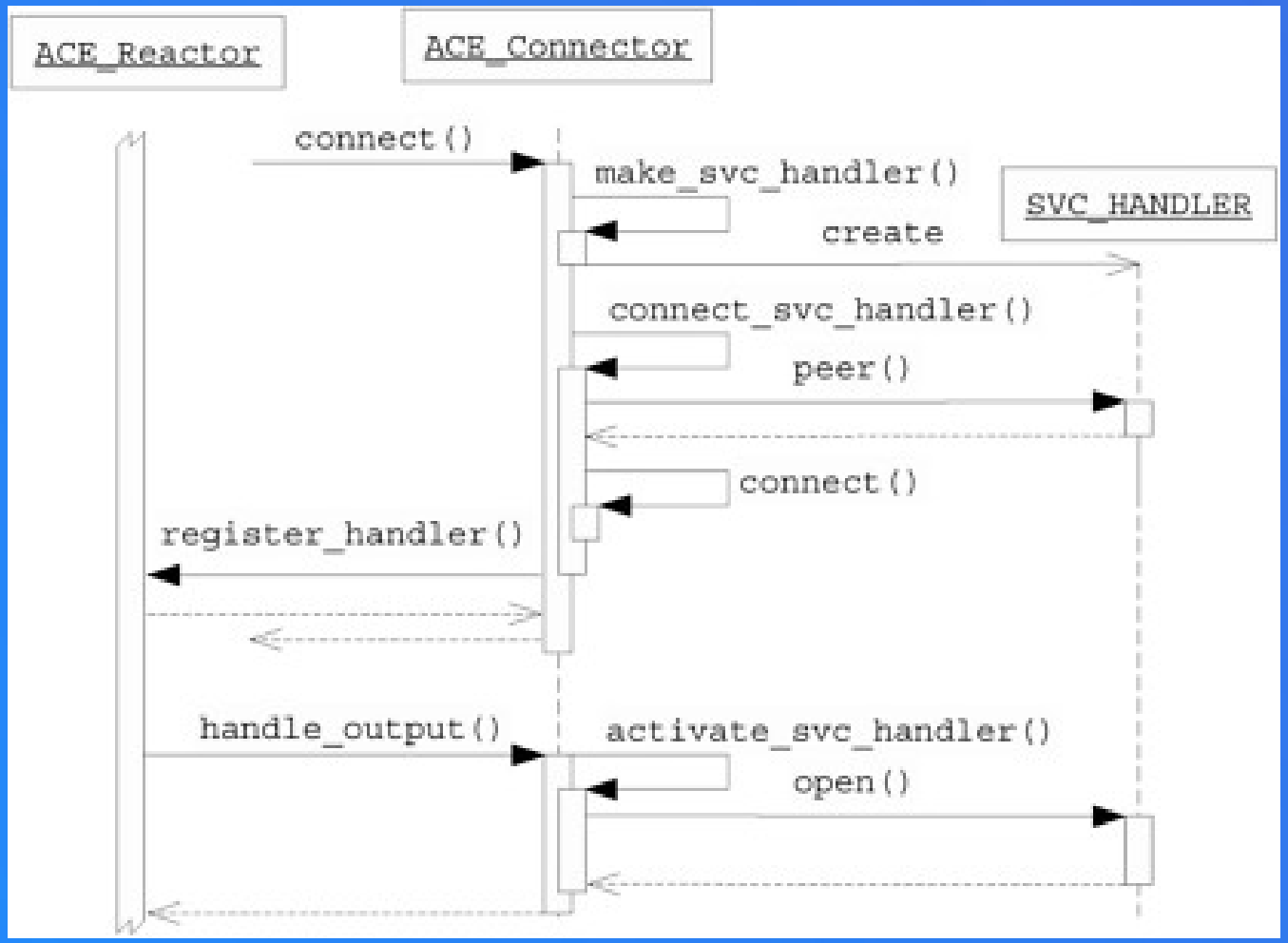
方法	说明
ACE_Connector() open()	
~ACE_Connector() close()	关闭接受器的 IPC 端点，释放其资源
connector()	返回指向底层 PERR_CONNECTOR 的引用

- ACE_Connector 连接建立和服务处理器初始化方法

方法	说明
connect()	当应用要将某个服务处理器连接带正在侦听的对端时，会调用该模板方法，它可以使用下面的 3 个方法来使 主动的连接 IPC 端点并创建和激活与其相关联的服务处理器 所必需的步骤分离
make_svc_handler()	这个工厂方法创建服务处理器，服务处理器则会使用已连接的 IPC 端点
connect_svc_handler()	这个挂钩方法使用服务处理器的 IPC 端点来同步或异步的主动连接端点
activate_svc_handler()	这个挂钩方法调用服务处理器的 open() 方法，让服务处理器完成对自己的初始化
handle_output()	在异步发起的连接请求完成之后，反应器调用该方法，它调用 activate_svc_handler()，让服务处理器对其自身进行初始化
cancel()	取消某个服务处理器，其连接是被异步发起的。调用者（不是连接器）负责关闭服务处理器

Acceptor-Connector 框架 - ACE_Connector

- ACE_Connector 异步连接建立的各步骤



Acceptor-Connector 框架 - ACE_Connector

- connect() 方法的部分参数
  - SVC_HANDLER 指针的引用
    - 如果传入 NULL，则 make_svc_handler() 方法被调用
  - ACE_Sync_Options 对象的引用
    - 该参数将影响：
      - 是否使用 ACE_Reactor 框架来检测连接完成
      - 为连接完成进行多久的等待

Acceptor-Connector 框架 - ACE_Connector

- 使用 ACE_Connector （ Handler ）

```
#include <ace/Svc_Handler.h>
#include <ace/SOCK_Stream.h>
class InputHandler:
    public ACE_Svc_Handler<ACE_SOCK_Stream, ACE_NULL_SYNCH> {
public:
  typedef ACE_Svc_Handler<ACE_SOCK_Stream,ACE_NULL_SYNCH> Parent;
  enum { BUF_SIME = 512 };

    virtual int open(void* a) {
        if (Parent::open(a) == -1) return -1;
        return this->activate(THR_NEW_LWP | THR_DETACHED);
    }
    virtual int handle_input(ACE_HANDLE) {
      ssize_t n = peer().recv(buf, BUF_SIME);
      if (n <= 0)
        ACE_ERROR_RETURN((LM_ERROR, "%p\n","peer().recv()"), -1);
        buf[n] = 0;
        ACE_DEBUG((LM_DEBUG, "%s\n", buf));
        return 0;
    }
```

Acceptor-Connector 框架 - ACE_Connector

- 使用 ACE_Connector （ Handler ）

```
    virtual int svc() {
        char inBuf[BUF_SIME] = "";
        while (std::cin.getline(inBuf, BUF_SIME)) {
            if (peer().send(inBuf, strlen(inBuf)) == -1) {
                ACE_ERROR((LM_ERROR, "%p\n", "peer().send()"));
                break;
            }
        }
        return 0;
    }

private:
    char buf[BUF_SIME];
};
```

Acceptor-Connector 框架 - ACE_Connector

- 使用 ACE_Connector （ Connector 和 main() ）

```
typedef ACE_Connector<InputHandler, ACE_SOCK_Connector>
MyConnector;

int main() {
    ACE_INET_Addr addr(8868, "127.0.0.1");
    MyConnector connector;

    InputHandler* p = 0;//由connector 创建 InputHandler
    if (connector.connect(p, addr) == -1)
        ACE_ERROR_RETURN((LM_ERROR, "%p\n", "connect()"), -1);

    ACE_Reactor::instance()->run_reactor_event_loop();
}
```

Acceptor-Connector 框架 - 小结

- Acceptor-Connector 框架一个重要的能力：
  - 解除 服务的连接和初始化策略 与 它的服务处理策略 的耦合
- 该框架将连接和初始化策略分解到 ACE_Acceptor 和 ACE_Connector 两个类模板
- 将服务处理策略分解到 ACE_Svc_Handler 类模板中