**学之者生，用之者死——ACE历史与简评**

陈硕 (giantchen@gmail.com)

Blog.csdn.net/Solstice

ACE 是现代面向对象网络编程的鼻祖，确立了许多重要模式，如 Reactor、Acceptor 等，重要到我们甚至觉得网络编程就应该是那样的。但为什么 ACE 叫好不叫座？大名鼎鼎却使用者寥寥？本文谈谈我的个人观点。

ACE 是一套重量级的 C++ 网络库，早期版本由 Douglas Schmidt 独自开发，后来有 [40 余名学生与工作人员](http://www.cs.wustl.edu/~schmidt/ACE-members.html)也贡献了大量代码。作者 Douglas Schmidt 凭借它发表了 [30 余篇学术论文](http://www.cs.wustl.edu/~schmidt/ACE-papers.html)。ACE 的一大特点是融合了 Douglas Schmidt 提出的很多面向对象[网络编程的设计模式](http://www.cs.wustl.edu/~schmidt/patterns-ace.html)，并且具有不可思议的[跨平台能力](http://www.cs.wustl.edu/~schmidt/ACE-versions-i.html)

**1 ACE 历史**

先说说 ACE 之父 Douglas Schmidt 的[个人经历](http://www.cs.wustl.edu/~schmidt/resume.pdf)：

* 1990 年在加州大学 Irvine 分校获计算机硕士学位；
* 1994 年在同一学校获计算机博士学位，论文《[An Object-Oriented Framework for Experimenting with Alternative Process Architectures for Parallelizing Communication Subsystems](http://www.cs.wustl.edu/~schmidt/PDF/dissertation.pdf)》。从论文内容看，主要工作就是后来大名鼎鼎的 ACE framework，文中叫 ASX framework。
* 1994 年博士毕业后前往华盛顿大学任助理教授，后升至副教授
* 2003 年起在 Vanderbilt 大学任正教授至今

我相信 ACE 是 Douglas 在读博期间的主要工作，ACE 这个名字最早出现在 1993 年 12 月的一篇会议论文上，Douglas 的这篇文章获得了“最佳学生论文”奖。在此之前，Douglas 已经用 ASX 等其他名字发表了内容相近的文章。

我能下载到的最早的 ACE 版本是 4.0.32，有大约 86,000 行 C++ 代码，代码的时间戳是 1998 年 10 月 22 日。早期 ACE 由 Douglas Schmidt 个人独立开发，从 ChangeLog 得知，1993 年 11 月 ACE 的版本号是 2.12。到了 1995 年 9 月，才有第一次出现其他开发者。在 1993~1996 年间的 684 次改动中，Douglas 一个人贡献了 529 次，另外几个主要开发者以及他们的修改次数分别是 Prashant Jain (58)、Tim Harrison (42)、David Levine (28)、Irfan Pyarali (20)、Jesper S. M|ller (5)。

从整个 ChangeLog 看，从 1993 年到 2010 年 3 月有 19,000 余次改动。有超过 200 人修改过代码，其中 23 个人的 check-in 次数大于 100，排名前 12 的代码修改者为：

   3635  Johnny Willemsen （活跃年份：2001~今）

   2586  Douglas C. Schmidt（原作者，活跃年份：1993~今）

   1861  Steve Huston （活跃年份：1997~今）

   1197  David L. Levine  （活跃年份：1996~2000）

    962  Nanbor Wang  （活跃年份：1998~2003）

    907  Ossama Othman （活跃年份：1999~2005）

    865  Chad Elliott （活跃年份：2000~今）

    823  Bala Natarajan （活跃年份：1999~2004）

    708  Carlos O'Ryan （活跃年份：1997~2001）

    544  J.T. Conklin （活跃年份：2004~2008）

    479  Irfan Pyarali （活跃年份：1996~2003）

    368  Darrell Brunsch （活跃年份：1997~2001）

看到这些“活跃年份”，你的第一反应是什么？我想到的是，这些人会不会多半是 Douglas 指导的研究生？我猜他们在读研期间参与改进 ACE，把工作内容写成论文发表，然后毕业走人。或许这能解释 ACE 代码风格的多样性。

在浏览代码历史的过程中，我还发现一个很有意思的现象，在 2008 年 3 月 4 日，某人不小心把整个 ACE 的源代码树删除了：

<https://svn.dre.vanderbilt.edu/viewvc/Middleware?view=revision&revision=80824>

随后又很快恢复：

<https://svn.dre.vanderbilt.edu/viewvc/Middleware?view=revision&revision=80826>

干这件事情的老兄在 2005~2009 这几年里一共 check in 了 120 余次。你对这件事情怎么看？你们的开发团队里有这样的人吗？

**2 事实与思考**

**1. 除了 Douglas Schmidt 和 Stephen Huston 写的三本书籍之外，没有其他专著讲 ACE。**

究竟是 ACE 太好用了，以至于无需其他书来讲解，还是太难用了，讲也讲不明白？抑或根本就没人在乎？

《[C++ 网络编程 第1卷](http://www.china-pub.com/34733)》《[C++ 网络编程 第2卷](http://www.china-pub.com/15709)》《[ACE 程序员指南](http://www.china-pub.com/22057)》这三本书先后于 2001、2002、2003 年出版，之后再无更新。在同一时期，同样在网络编程领域，尽管 W. Richard Stevens 在 1999 年去世，他的 UNP 和 APUE 仍然由别人续写了新版。讲 C 语言 Sockets API 的书尚且不断更新，上层封装的 C++ 居然无动于衷？真的是封装到位了，屏蔽了这些变化？

UNP 的可操作性很强，读前面几章，就能上手编写简单的网络程序，看完大半本书，网络编程基本就算入门了，能编写一般应用的网络程序。相反，读完 ACE 那几本书，对于简单的网络编程任务还是感觉无从下手，这是因为书写得不好，还是 ACE 本身不好用？

**2. ACE 很难用，非常容易用错**

我不止听到一个人对我说，他们在项目里尝试过 ACE，不是中途放弃，因为出了问题无法解决；就是勉强交差，并且从下一个项目起坚决不用。我听到的另一个说法是，ACE 教程的例子必须原封不动地抄下来，改一点点就会出漏子。不巧的是，ACE 的例子举来举去就是个 Logging 服务器，让人想照猫画虎也无从下手。在最近的《代码之美》一书中，Douglas Schmidt 再次拿它为例，说明他真的很喜欢这个例子。

用 ACE 编程如履薄冰，生怕在阴沟里翻船，不知道它背后玩了什么把戏。相反，用 10 来个 Sockets 系统调用就能搞定网络编程，我感觉比使用 ACE 难度要小。为什么“高级”工具反而没有低级工具顺手呢？

不好用的直接后果是少有人用，放眼望去，目前涉及网络的 C++ 开源项目里边，鲜有用 ACE 作为通信平台的（我知道的只有 Mangos）。相反，libevent 这个轻量级的 IO multiplexing 库有 memcached 这样的著名用户。

**3. ACE 代码质量不高，更像是一个研究项目，而不是工业界的产品**

读 ACE 现在的代码，一股学生气扑面而来，感觉像在读实习生写的代码。抛开编码风格不谈，这里举三个“硬伤”：

* sleep < 2ms

在某些早期的 Linux 内核上，如果 select/poll 的等待时间小于 2ms，内核会采用 [busy-waiting](http://lkml.indiana.edu/hypermail/linux/kernel/9912.3/0484.html)。这是极大的 CPU 资源浪费，而 ACE 似乎没有考虑避免这一点。

* Linux TCP self-connection

Linux 的 TCP 实现有一个特殊“行为”，在某些特殊情况下会发起[自连接](http://lkml.indiana.edu/hypermail/linux/kernel/9909.3/0510.html)。而 Linux 网络协议栈的维护者认为这是一个 feature，不是 bug，拒绝修复。通常网络应用程序不希望出现这种情况，我见过的好的网络库会有意识地检查并断开这种连接，然而 ACE 漠然视之。

* timeval on 64-bit

ACE\_Time\_Value 类直接以 struct timeval 为成员变量，保存从 Epoch 开始的微秒数。这在 32-bit 下没问题，对象大小是 8 字节。到了 LP64 模式的 64-bit 平台，比如 Linux，对象大小变为 16 字节，这么做就不够好了。我们可以直接用 int64\_t 来保存这个以微秒为单位的时间，64-bit 整数能存下上下 30 万年，足够用了。减小对象大小并不是为了节约几个字节的内存，而是方便函数参数传递。在 x86-64 上，这种 8 字节的结构体可以用 64-bit 寄存器直接传参，也就是说 pass by value 会比 pass by reference 更快。对于一般的应用程序而言，要不要这么做值得商榷。对于底层的 C++ 网络库，不加区分地使用 pass by reference 会让人怀疑作者知其然不知其所以然。

对于以上几点情况，我怀疑 ACE 根本没用在 Linux 大规模生产环境下使用过，我只能期望它在别的平台表现好一些了。ACE 的作者们似乎更注重验证新想法，然后发论文，而不是把它放到工业环境中反复锤炼，打造为靠得住的产品。（类似 Minix 与 Linux 的关系。）

**4. 移植性很好，支持我知道的和不知道的[很多平台](http://www.cs.wustl.edu/~schmidt/ACE-versions-i.html)**

ACE 的意义在于让我们明白了C++代码可以做到可移植，并展示了这么做会付出多么巨大的代价。不细说了，读过 ACE 代码的人都明白。

从代码质量上看，ACE 做到了能在这些平台上运行，但似乎没有在哪个平台占据主导地位。有没有哪个平台的网络编程首选 ACE？

出现这一状况的原因是，跨平台和高性能是矛盾的。跨平台意味着要抽象出多个平台的共性，以最 general 的方式编写上层代码。而高性能则要求充分发挥平台的特性，剑走偏锋，用尽平台能提供的一切加速手段，哪怕与其他平台不兼容。网络编程对此尤为敏感。

我不知道 ACE 的性能如何，因为在各项性能评测榜上基本看不到它的名字（c10k 里就没有 ACE 的身影）。另外，Buffer class 的好坏直接反应了网络库对性能的追求，ACE 提供了比 std::deque<uint8\_t> 更好的输入输出 Buffer 吗？（我不是说 deque 有多好，它基本是 fail-safe 的选择而已。）

**5. ACE 过于复杂，甚至比它试图封装的对象更复杂。**

（这里的代码行数均为 wc 命令的粗略估计。）

ACE 5.7 自身（不含 TAO 和 CIAO）有 30 万行 C++ 代码（Douglas [自己给出的数据是 25 万行](http://www.cs.wustl.edu/~schmidt/PDF/ACE-tutorial.pdf)，可能指的是略早的版本），这是一个什么概念呢？我们来看 TCP/IP 协议栈本身的实现有多少行：（均不含 IPv6）

* TCPv2 列出的 BSD4.4-Lite 完整 TCP/IP 协议栈代码有 15,000 行，其中 4,500 行 TCP 协议，800 行 UDP 协议，2,500 行 IP 协议
* Linux 1.2.13 完整的 TCP/IP 协议栈有 2 万多行 (net/inet)
* Linux 2.6.32.9 的 TCP/IP 协议栈有 6 万多行 (net/ipv4)
* FreeBSD 8.0 的 TCP/IP 协议栈有 5 万多行 (sys/netinet, 不含 sctp)

换句话说，ACE 用 30 万行 C++ 代码“封装”了不到 10 万行 C 代码（且不论 C++ 代码的信息密度比 C 大），这是不是头重脚轻呢？我理解的“封装”是把复杂的东西变简单，但 ACE 好像走向了另一个方向，把不那么复杂的东西变复杂了。

这个对比数字可能不太准确，因为 ACE 还封装了很多其他东西，请看。<http://www.dre.vanderbilt.edu/Doxygen/5.7.7/html/ace/inherits.html> 和 <http://www.dre.vanderbilt.edu/Doxygen/5.7.7/html/ace/hierarchy.html>

以下两张类的继承关系图片请在新窗口打开：

<http://www.dre.vanderbilt.edu/Doxygen/5.7.7/html/ace/a06178.png>

<http://www.dre.vanderbilt.edu/Doxygen/5.7.7/html/ace/a06347.png>

Douglas 说 ACE 包含了 [40 人年的工作量](http://www.cs.wustl.edu/~schmidt/PDF/ACE-tutorial.pdf)，对此我毫不怀疑。但是，网络编程真的需要这么复杂吗？TCP/IP 协议栈的实现也没这么多工作量嘛。或许只有 CORBA 这样的应用才会用到这么复杂的东西？那么为什么 ICE 在重新实现 CORBA 的功能时没有基于 ACE 来写呢？是不是因为 ACE 架子拉得大，底子并不牢？

**3 ACE 的意义**

ACE 对于面向对象、设计模式和网络编程具有重大历史和现实意义。

ACE 诞生之时，正是 90 年代初期面向对象技术的高速发展期，ACE 一定程度上是作为面向对象技术的成功案例来宣传的。

在 1994 年前后，Unix 分为两个阵营，AT&T 的 SVR4 与 BSD 的 BSD4.x，这两家的 IO multiplexing 不完全兼容。比如 SVR4 提供 poll 调用，而 BSD 提供 select 调用。ACE 当时的宣传点之一是用面向对象技术屏蔽了两个平台的差异，提供了统一的 Reactor 接口。

【接下来，poll 在 1996 年 9 月 7 号加入 NetBSD，并随 NetBSD 1.3 于 1998 年 1 月 4 号发布。随后 FreeBSD 3.0 也支持 poll，1998 年 10 月发布。Linux 很早就支持 select，从 2.1.23 内核起支持 poll，发布日期为 1997 年 1 月 26 号。也就是说，到了 1998 年，平台差异被暂时抹平了。随后 epoll、/dev/poll、kqueue 以性能为名，再次扩大了平台差异。当然，Windows 至今不支持 poll。】

ACE 的设计似乎过于强调面向对象的灵活性，一些不该使用虚函数的地方也提供了定制点，比如 ACE\_Timer\_Queue 就应是个具体类，而不是允许用户 override schedule/cancel/expire 之类的具体操作。面向对象中，“继承”的目的是为了被复用，而不是去复用基类的代码。

查其文献，Reactor 在 1993 年登上《C++ Report》杂志的时候，文章标题还比较朴素，挂着“面向对象”的旗号：

* 《[The Reactor: An Object-Oriented Interface for Event-Driven UNIX I/O Multiplexing (Part 1 of 2)](http://www.cs.wustl.edu/~schmidt/PDF/Reactor1-93.pdf)》
* 《[The Object-Oriented Design and Implementation of the Reactor: A C++ Wrapper for UNIX I/O Multiplexing (Part 2 of 2)](http://www.cs.wustl.edu/~schmidt/PDF/Reactor2-93.pdf)》

转眼到了 1994 年，也就是《设计模式》成书的那一年，Douglas 开始写文章言必称 pattern：

* Reactor 变成了 pattern，收录于《Pattern Languages of Program Design》一书（[An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events](http://www.cs.wustl.edu/~schmidt/PDF/Reactor.pdf)）。这篇文章比前面两篇难懂，如果直接阅读的话。
* Acceptor 是 pattern （[A Design Pattern for Passively Initializing Network Services](http://www.cs.wustl.edu/~schmidt/PDF/Acceptor.pdf)），
* Connector 也是 pattern（[A Design Pattern for Actively Initializing Network Services](http://www.cs.wustl.edu/~schmidt/PDF/Connector.pdf)），
* Proactor 还是 pattern（[An Object Behavioral Pattern for Demultiplexing and Dispatching Handlers for Asynchronous Events](http://www.cs.wustl.edu/~schmidt/PDF/proactor.pdf)），
* 居然连 Thread-Specific Storage 都成了 pattern（[An Object Behavioral Pattern for Accessing per-Thread State Efficiently](http://www.cs.wustl.edu/~schmidt/PDF/TSS-pattern.pdf)）。
* 还有 Non-blocking Buffered I/O，也是 pattern （[An Object Behavioral Pattern for Communication Gateways](http://www.cs.wustl.edu/~schmidt/PDF/TAPOS-00.pdf)）。

似乎 "pattern" 这个字样成了发文章的通行证，这股风气直到 2000 左右才刹住。之后这些论文集结出版，以《Pattern-Oriented Software Architecture》为名出了好几本书，ACE 的内容主要集中在第二卷。（请留意，原来的提法是 Object-Oriented，现在变成了 Pattern-Oriented，似乎软件开发就应该像糖果厂生产绿豆糕，用模子一个个印出来完事。）

ACE 就像一个 [pattern 大观园](http://www.cs.wustl.edu/~schmidt/patterns-ace.html)，保守估计有 10 来种 patterns 藏身其中，形成了一套模式语言（《[Applying a Pattern Language to Develop Application-level Gateways](http://www.cs.wustl.edu/~schmidt/PDF/TAPOS-00.pdf)》），这还不包括 GoF 定义的一般意义下的 OO pattern。

通过 ACE 来学习网络编程，那是本末倒置，因为它教不了你任何 UNP 以外的知识。（Windows 网络编程？）

然而，如果要用面向对象的方式来搞网络编程，那么 ACE 的思想（而不是代码）是值得效仿的，毕竟它饱含了 Douglas Schmidt 等学者的心血与智慧。学得好的例子有 Apache Mina、JBoss Netty、Python Twisted、Perl POE 等等。

这就是我说“学之者生，用之者死”的含义。

**4 ACE 文献导读**

Douglas Schmidt 写了很多 [ACE 的文章](http://www.cs.wustl.edu/~schmidt/ACE-papers.html)，其中不乏内容相近的作品。读他的文章，首选发表在技术杂志上的文章（比如 C++ Report），而不是发表在学术期刊或会议上的论文。前者的写作目的是教会读者技术，后者则往往是展示作者的新思路新想法，技术文章比学术论文要好读得多。

由于当时面向对象技术尚在发展，Douglas 文章里的图形很有特色，不是现在规范的 UML 图（那会儿 UML 还没定型呢），而是像变形虫一样的类图（经pinxue指出，这种图是 Grady Booch 发明的），放在一堆文献里也很容易认出来。

如果要用 ACE 的代码来验证文章的思路，我建议阅读和文章同时期的 4.0 版本代码，代码风格比较统一，代码量也不大，便于理解。

下面介绍几篇有代表性的论文。

* 1993 年 12 月第 11 届 SUG 会议，《The ADAPTIVE Communication Environment: Object-Oriented Network Programming Components for Developing Client/Server Applications》，获得最佳学生论文奖。这是我找到的最早一篇以 ACE 为题的论文。
* 1994 年 6 月第 12 届 SUG 会议，《[The ADAPTIVE Communication Environment: An Object-Oriented Network Programming Toolkit for Developing Communication Software](http://www.cs.wustl.edu/~schmidt/PDF/SUG-94.pdf)》，获得最佳学生论文奖。

以上两篇文章实际上内容基本相同，都是对 ACE 的概要介绍，看第二篇即可，第一次没看懂也没关系。

剩下要看的是一篇 Socket OO 封装、四篇 Reactor、三篇 Acceptor-Connector、一篇 Proactor。这些文章前面大多都给了链接，其余的这里补充一下：

* [IPC\_SAP: A Family of Object-Oriented Interfaces for Local and Remote Interprocess Communication](http://www.cs.wustl.edu/~schmidt/PDF/IPC_SAP-92.pdf)
* [The Design and Use of the ACE Reactor](http://www.cs.wustl.edu/~schmidt/PDF/reactor-rules.pdf)
* [Acceptor and Connector -- A Family of Object Creational Patterns for Initializing Communication Services](http://www.cs.wustl.edu/~schmidt/PDF/Acc-Con.pdf)  这篇论文其实可以不用看，因为它不过是把前面两篇发表在 C++ Report 上的文章合到了一起。

不想看这 10 篇论文的话，读中译本《[C++ 网络编程 第1卷](http://www.china-pub.com/34733)》《[C++ 网络编程 第2卷](http://www.china-pub.com/15709)》《[ACE 程序员指南](http://www.china-pub.com/22057)》也行，翻译质量都不错。

**5 设想中的 C++ 网络库**

与文章主旨无关，略。

我觉得网络库要解决现实的问题，满足现实的需要，而不是把 features/patterns 堆在那里等别人来用。应该先有应用，再提炼出库。而不是先造库，然后寻求应用。