在 [《POSA2》](http://www.douban.com/subject/1137259/) 一书中，关于这两个模式有两个很形象的比喻：   
  
**半同步/半异步（half-sync/half-async）：**   
许多餐厅使用 半同步/半异步 模式的变体。例如，餐厅常常雇佣一个领班负责迎接顾客，并在餐厅繁忙时留意给顾客安排桌位，为等待就餐的顾客按序排队是必要的。领班由所有顾客“共享”，不能被任何特定顾客占用太多时间。当顾客在一张桌子入坐后，有一个侍应生专门为这张桌子服务。   
  
**领导者/追随者（Leader/Followers）：**   
在日常生活中，领导者/追随者模式用于管理许多飞机场出租车候车台。在该用例中，出租车扮演“线程”角色，排在第一辆的出租车成为*领导者*，剩下的出租车成为*追随者*。同样，到达出租车候车台的乘客构成了必须被多路分解给出租车的事件，一般以先进先出排序。一般来说，如果任何出租车可以为任何顾客服务，该场景就主要相当于*非绑定*句柄/线程关联。然而，如果仅仅是某些出租车可以为某些乘客服务，该场景就相当于*绑定*句柄/线程关联。   
  
在 《POSA2》 书中列举的例子都比较复杂，并且书上没有列出完整的代码。但是这两个模式其实都可以在[《unix网络编程》](http://www.douban.com/subject/1231567/)一书中找到对应的完整的代码和相关的讨论。   
  
在 半同步/半异步 模式中，需要由模式实现者显示构造一个队列，以便同步层和异步层可以通信。   
在 [《unix网络编程》](http://www.douban.com/subject/1231567/) 一书的 “27.12 TCP预先创建线程服务器程序，主线程统一 accept” 的例子中，**如果只是从处理 accept 这个事件上看，可以认为这是一个使用了 半同步/半异步 模式的例子。但是从具体的业务处理（即web\_child的处理上），仍然可以认为是一个ThreadPerConnection模型，因为在 thread\_main 中直接读取请求和发送响应。**在这个例子中，就有一个队列：

**Java代码  [收藏代码](javascript:void())**

1. [b]//这就是一个典型的循环队列的定义，iget 是队列头，iput 是队列尾[/b]
2. **int** clifd[MAXNCLI], iget, iput;
4. **int** main( **int** argc, **char** \* argv[] )
5. {
6. ......
7. **int** listenfd = Tcp\_listen( NULL, argv[ 1 ], &addrlen );
8. ......
10. iget = iput = 0;
12. **for**( **int** i = 0; i < nthreads; i++ ) {
13. pthread\_create( &tptr[i].thread\_tid, NULL, &thread\_main, (**void**\*)i );
15. **for**( ; ; ) {
16. connfd = accept( listenfd, cliaddr,, &clilen );
17. clifd[ iput ] = connfd;     [b]// 接受到的连接句柄放入队列[/b]
18. **if**( ++iput == MAXNCLI ) iput = 0;
19. }
20. }
22. **void** \* thread\_main( **void** \* arg )
23. {
24. **for**( ; ; ) {
25. **while**( iget == iput ) pthread\_cond\_wait( ...... );
26. connfd = clifd[ iget ];     [b]// 从队列中获得连接句柄[/b]
27. **if**( ++iget == MAXNCLI ) iget = 0;
28. ......
29. web\_child( connfd );
30. close( connfd );
31. }
32. }

而在 领导者/追随者 模式中，同样是有一个队列的，不过不需要模式实现者显示构造，而是直接使用了操作系统底层的队列。   
  
在 [《unix网络编程》](http://www.douban.com/subject/1231567/) 一书的 “27.11 TCP 预先创建服务器线程，每个线程各自 accept ” 的例子中，就是直接使用了操作系统中关于 accept 的队列。这个例子可以认为是 领导者/追随者 模式的一个例子。

**Java代码  [收藏代码](javascript:void())**

1. **int** listenfd;
3. **int** main( **int** argc, **char** \* argv[] )
4. {
5. ......
6. listenfd = Tcp\_listen( NULL, argv[ 1 ], &addrlen );
7. ......
8. **for**( **int** i = 0; i < nthreads; i++ ){
9. pthread\_create( &tptr[i].thread\_tid, NULL, &thread\_main, (**void**\*)i );
10. }
11. ......
12. }
14. **void** \* thread\_main( **void** \* arg )
15. {
16. **for**( ; ; ){
17. ......
18. [b]// 多个线程同时阻塞在这个 accept 调用上，依靠操作系统的队列[/b]
19. connfd = accept( listenfd, cliaddr, &clilen );
20. ......
21. web\_child( connfd );
22. close( connfd );
23. ......
24. }
25. }

当然，这里提到的操作系统的队列，在 半同步/半异步 模式中虽然没有明显地指出来，但只要是通过操作系统来做 accept ，那么在 半同步/半异步 模式中仍然会隐式地用到。   
  
在[《POSA2》](http://www.douban.com/subject/1137259/)中，作者的评价：   
因为半同步/半异步设计在 web 服务器虚拟内存而不是操作系统内核内排队请求，所以它更具伸缩性。   
  
  
看了上面的代码之后，明白了为何 ACE 的作者在 [《C++网络编程2》](http://www.douban.com/subject/1141548/) 中特意引用了一首诗来“表达我们对 Richard 之持久影响的看法”：   
 **不是在悲哀的冥河之滨，也不是在遥远的   
乐土般的平原的清辉中，我们将在死者中间   
遇见那些我们一直是其学生的人 ... ...   
我们还将相遇，分离，再相遇，   
在死者们相遇的地方，在活着的人的唇上**  
  
关于不同的客户-服务器编程模型，在[《unix网络编程》](http://www.douban.com/subject/1231567/)的 “第27章 客户-服务器程序的其他设计方法”中讨论得很充分，对每种模型的性能也做了很好的分析。