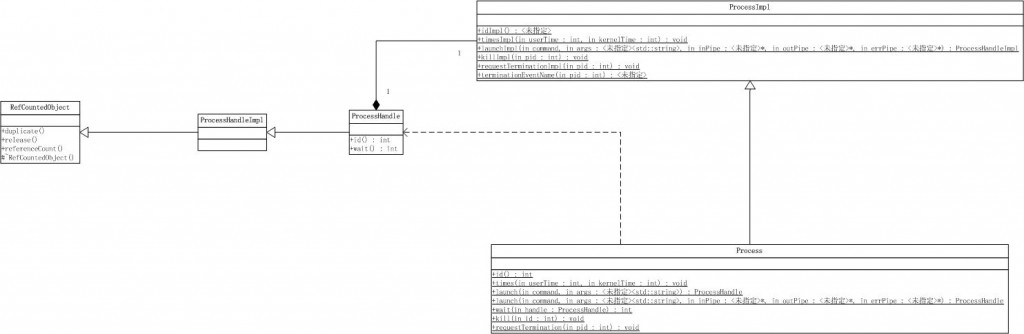
**POCO C++库学习和分析 -- 进程**

       Poco::Foundation库中涉及进程的内容主要包括了4个主题，分别是进程(Process)、进程间同步（inter-process synchronization）、管道（Pipes）、共享内存（Shared Memory）。我们都知道管道、共享内存、网络通讯是进程间数据交互的3种基本方式。由于网络通讯足够复杂，在Poco的结构划分里被单独分成了一个库Net，Foundation库中并没有涉及。下面一一介绍：

**1. 进程**

       关于中的进程其实没有什么可说的，不管是其内部实现还是外部使用都非常的简单。内部实现上只不过是不同操作系统进程API的封装，下面是它的类图：



       在Poco中进程类的所有成员函数都是静态函数。主要的功能函数覆盖3个方面：

      1. 创建新进程

      2. 销毁其他进程

      3. 获取当前进程信息

       值得注意的是，在Poco中进程创建时，可以对进程的I/O进程重定向。其函数如下：

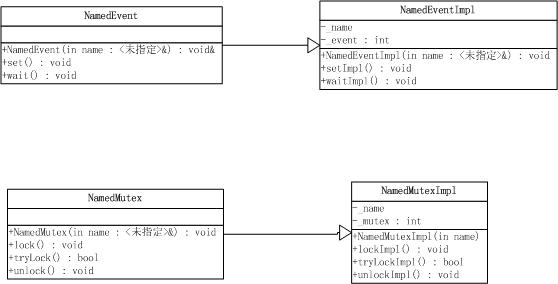
**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8648181)

1. ProcessHandle Process::launch( **const** std::string& path, **const** std::vector<std::string>& args, Pipe\* inPipe, Pipe\* outPipe, Pipe\* errPipe)

**2.  进程间同步**

       Poco库中提供了Poco::NamedMutex和Poco::NamedEvent类用于进程间的同步。同线程间同步的类Mutex，Event相比，进程间同步都是命名的，这毫无疑问是因为操作系统的底层函数的要求。

       其类图如下：



**3. 管道**

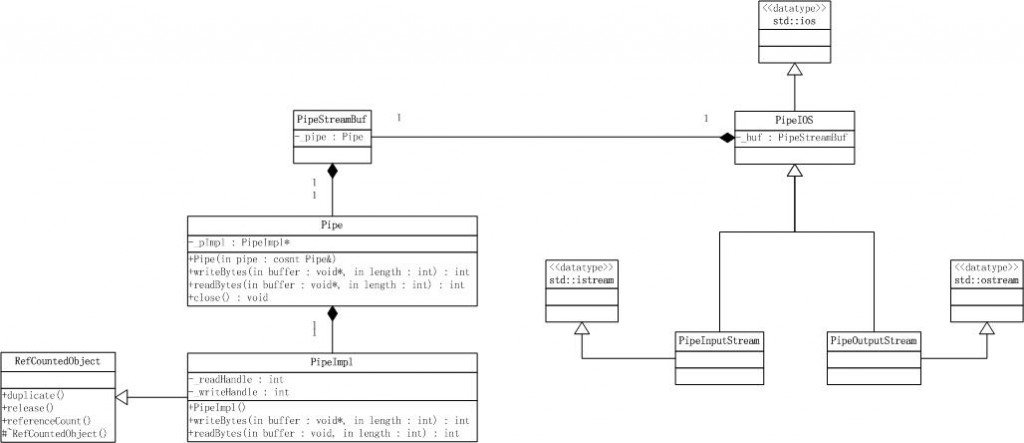
       我们都知道管道是一个单向的通讯通道，或者用来读或者用来写。如果两个进程间要实现双向的通讯，必须在进程之间创建两个管道。Poco库中也封装了管道方便进程通讯，但Poco库中对于管道的读写，却不是通过管道的本身，而是通过Poco::PipeOutputStream和Poco::PipeInputStream 两个类。这样的话，便可以实现和标准库流操作的无缝结合。

       下面是一个例子来说明这几者的关系：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8648181)

1. #include "Poco/Process.h"
2. #include "Poco/PipeStream.h"
3. #include "Poco/StreamCopier.h"
4. #include <fstream>
5. **using** Poco::Process;
6. **using** Poco::ProcessHandle;
7. **int** main(**int** argc, **char**\*\* argv)
8. {
9. std::string cmd("/bin/ps");
10. std::vector<std::string> args;
11. args.push\_back("-ax");
12. Poco::Pipe outPipe;
13. ProcessHandle ph = Process::launch(cmd, args, 0, &outPipe, 0);
14. Poco::PipeInputStream istr(outPipe);
15. std::ofstream ostr("processes.txt");
16. Poco::StreamCopier::copyStream(istr, ostr);
17. **return** 0;
18. }

       管道的类图如下：



**4. 共享内存**

       在Poco库中，Poco::SharedMemory类用于实现共享内存功能。它支持两种创建方式：

       1.从确定大小的内存区域

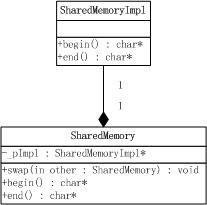
       2. 从文件（通过把文件映射入共享内存区域）

       而在接口上，Poco::SharedMemory只外露了两个接口：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8648181)

1. **char**\* begin() **const**;
2. **char**\* end() **const**;

      begin（）函数返回共享内存的起点，end（）函数则返回其终点。下面是它的类图和两个使用例子，并不复杂：



      例子一：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8648181)

1. // Map a file into memory
2. #include "Poco/SharedMemory.h"
3. #include "Poco/File.h"
4. **using** Poco::SharedMemory;
5. **using** Poco::File;
6. **int** main(**int** argc, **char**\*\* argv)
7. {
8. File f("MapIntoMemory.dat");
9. SharedMemory mem(f, SharedMemory::AM\_READ); // read-only access
10. **for** (**char**\* ptr = mem.begin(); ptr != mem.end(); ++ptr)
11. {
12. // ...
13. }
14. **return** 0;
15. }

      例子二：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8648181)

1. // Share a memory region of 1024 bytes
2. #include "Poco/SharedMemory.h"
3. **using** Poco::SharedMemory;
4. **int** main(**int** argc, **char**\*\* argv)
5. {
6. SharedMemory mem("MySharedMemory", 1024,
7. SharedMemory::AM\_READ | SharedMemory::AM\_WRITE);
8. **for** (**char**\* ptr = mem.begin(); ptr != mem.end(); ++ptr)
9. {
10. \*ptr = 0;
11. }
12. **return** 0;
13. }

（版权所有，转载时请注明作者和出处  <http://blog.csdn.net/arau_sh/article/details/8648181>）