# 标题： 深入浅出-服务器高并发库libevent（二）

上一章，我们简单介绍了libevent的环境的安装，和简单的事例。

现在先不要着急分析他的代码，在这里我首先要介绍一个专业名词“Reactor

模式”。

## 2.1 Reactor的事件处理机制

我们应该很清楚函数的调用机制。

1. 程序调用函数
2. 函数执行
3. 程序等待函数将结果和控制权返回给程序
4. 程序继续处理和执行

Reactor 被翻译成 反应堆，或者反应器。 Re-actor 发音。

他是一种事件驱动机制。和普通函数调用的不同之处在于， 应用程序不是主动 的调用某刻API完成处理，而是恰恰相反，reactor逆置了事件的处理流程，应用程序需要提供相应的接口注册到reacotr上。如果相应的事件发生。Reacotr将主动调用应用程序注册的接口，这些接口就是我们常常说的“回调函数”。

我们使用libevent框架也就是想利用这个框架去注册相应的事件和回调函数。

当这些事件发生时，libevent会调用这些注册好的回调函数处理相应的事件(I/O读写、定时和信号)

通过reactor调用函数，不是你主动去调用函数，而是等着系统调用。

一句话：“不用打电话给我们，我么会打电话通知你”。

举个例子，你去应聘某xx公司，面试结束后。

“普通函数调用机制”公司的HR比较懒，不会记你的联系方式，那咋办，你只能面试完自己打电话问结果。有没有被录取啊，还是被拒绝了。

“Reacotr”公司的HR就记下了你的联系方式，结果出来后HR会主动打电话通知你。有没有被录取啊，还是悲剧了。你不用自己打电话去问，实际上你也不能，你没有HR的联系方式。

## 2.2 Reactor模式的优点

Reactor模式是编写高性能网络服务器的必备技术之一，它具有如下的优点：

    1）响应快，不必为单个同步时间所阻塞，虽然Reactor本身依然是同步的；

    2）编程相对简单，可以最大程度的避免复杂的多线程及同步问题，并且避免了多线程/进程的切换开销；

    3）可扩展性，可以方便的通过增加Reactor实例个数来充分利用CPU资源；

    4）可复用性，reactor框架本身与具体事件处理逻辑无关，具有很高的复用性；

## 2.3 Reactor模式的必备条件

1） 事件源  
Linux上是文件描述符，Windows上就是Socket或者Handle了，这里统一称为“句柄集”；程序在指定的句柄上注册关心的事件，比如I/O事件。

2） event demultiplexer——事件多路分发机制  
由操作系统提供的I/O多路复用机制，比如select和epoll。  
    程序首先将其关心的句柄（事件源）及其事件注册到event demultiplexer上；  
当有事件到达时，event demultiplexer会发出通知“在已经注册的句柄集中，一个或多个句柄的事件已经就绪”；  
    程序收到通知后，就可以在非阻塞的情况下对事件进行处理了。  
对应到libevent中，依然是select、poll、epoll等，但是libevent使用结构体eventop进行了封装，以统一的接口来支持这些I/O多路复用机制，达到了对外隐藏底层系统机制的目的。

3） Reactor——反应器  
    Reactor，是事件管理的接口，内部使用event demultiplexer注册、注销事件；并运行事件循环，当有事件进入“就绪”状态时，调用注册事件的回调函数处理事件。  
对应到libevent中，就是event\_base结构体。  
一个典型的Reactor声明方式：

class Reactor

{

public:

int register\_handler(Event\_Handler \*pHandler, int event);

int remove\_handler(Event\_Handler \*pHandler, int event);

void handle\_events(timeval \*ptv);

// ...

};

4） Event Handler——事件处理程序  
    事件处理程序提供了一组接口，每个接口对应了一种类型的事件，供Reactor在相应的事件发生时调用，执行相应的事件处理。通常它会绑定一个有效的句柄。  
对应到libevent中，就是event结构体。  
下面是两种典型的Event Handler类声明方式，二者互有优缺点。

class Event\_Handler

{

public:

virtual void handle\_read() = 0;

virtual void handle\_write() = 0;

virtual void handle\_timeout() = 0;

virtual void handle\_close() = 0;

virtual HANDLE get\_handle() = 0;

// ...

};

class Event\_Handler

{

public:

// events maybe read/write/timeout/close .etc

virtual void handle\_events(int events) = 0;

virtual HANDLE get\_handle() = 0;

// ...

};

上面讲到了Reactor的基本概念、框架和处理流程，对Reactor有个基本清晰的了解后，再来对比看libevent就会更容易理解了，接下来就正式进入到libevent的代码世界了，加油！