轻量网络库之libev封装

引言：

之所以想写这篇文章，是因为不久前有朋友给我反馈，说他们的后台开发人员在写服务程序的时候，经常花费大量时间在网络调试上。于是我拿了一部分代码研究，发现这个程序从创建套接口、到上层应用，都扁平化的实现在了一个CPP文件里，而且每次创建新项目，都是拷贝这个文件，在上面直接修改，代码冗余量较大，极易出错，朋友希望给予帮助。网络调试中的问题是由多方面原因造成的，一部分是代码编写不严谨，一部分是测试或数据不仔细。我说我可以提供轻量级的网络库，方便你们少写一些代码。

做为一名资深码农，我肯定是有相应的网络库的，我曾经在YY呆过一段时间，如YY的网络框架就是很优秀的作品（学之者生，用之者亦生），但是我不打算分享一款商业产品。另一个tinynet，轻量，支持多线程、支持YY、json、protobuf等协议格式，但还在完善中，我暂时不打算开源。

我准备借用第三方库，目前linux下比较流行的一些开源网络库有ACE、libevent、libev以及boost的asio，在这些库的基础上进行封装，不会带来任何版权问题。选哪个库呢，ACE是“学之者生，用之者死”，asio我不太熟悉，于是在libevent和libev中二选一，由于libev性能更高、代码更简洁，号称是libevent的替代产品，所以我便选它了。

libev用法说明:

参考：http://www.yeolar.com/note/2012/12/16/libev/

先看官方用例：

```
// a single header file is required
#include <ev.h>

#include <stdio.h> // for puts

// every watcher type has its own typedef'd struct
// with the name ev_TYPE
ev_io stdin_watcher;
ev_timer timeout_watcher;

// all watcher callbacks have a similar signature
// this callback is called when data is readable on stdin
static void
stdin_cb (EV_P_ ev_io *w, int revents)
{
  puts ("stdin ready");
  // for one-shot events, one must manually stop the watcher
  // with its corresponding stop function.
  ev_io_stop (EV_A_ w);

  // this causes all nested ev_run's to stop iterating
  ev_break (EV_A_ EVBREAK_ALL);
}

// another callback, this time for a time-out
static void
timeout_cb (EV_P_ ev_timer *w, int revents)
{
  puts ("timeout");
  // this causes the innermost ev_run to stop iterating
  ev_break (EV_A_ EVBREAK_ONE);
}

int
main (void)
{
  // use the default event loop unless you have special needs
  struct ev_loop *loop = EV_DEFAULT;

  // initialise an io watcher, then start it
  // this one will watch for stdin to become readable
  ev_io_init (&stdin_watcher, stdin_cb, /*STDIN_FILENO*/ 0, EV_READ);
  ev_io_start (loop, &stdin_watcher);

  // initialise a timer watcher, then start it
  // simple non-repeating 5.5 second timeout
  ev_timer_init (&timeout_watcher, timeout_cb, 5.5, 0.);
  ev_timer_start (loop, &timeout_watcher);

  // now wait for events to arrive
  ev_run (loop, 0);

  // break was called, so exit
  return 0;
}
```

写一个后台程序的流程：
1）初使化ev_loop对象；
2）创建侦听套接口，并关联ev_io对象，加入到ev_loop事件循环中；
3）接收客户端连接，为每个客户端套接字关联一个ev_io对象，也加入到ev_loop事件循环中；
4）接收客户端事件消息，根据事件类型，调用操作系统IO获取网络数据流；
5）解析协议格式，交给应用层逻辑处理；
6）发回响应，需要找到连接，再次调用操作系统IO发送数据；

libev对高并发和多种事件处理得很好，但libev对使用者的要求仍旧比较高，程序员需要对网络底层比较熟悉，否则比较容易出现或这或那的问题。

我们希望屏蔽底层复杂的细节，任何一位程序员都能轻松地编写网络程序，封装后的libev库取名libevwork。

libevwork用法说明：

关于libevwork的用法，下面提供helloword样例，给大家一个大致的了解：

```
#include "EVWork.h"

using namespace evwork;

class CDataEvent
    : public IDataEvent
{
public:

    virtual int onData(IConn* pConn, const char* pData, size_t uSize)
    {
        pConn->sendBin(pData, uSize);
        return uSize;
    }
};

int main(int argc, char* argv[])
{
    //-------------------------------------------------------------------
    // libevwork初使化

    CSyslogReport LG;
    CEVLoop LP;
    CConnManager CM;
    CWriter WR;

    CEnv::setLogger(&LG);
    CEnv::setEVLoop(&LP);
    CEnv::setLinkEvent(&CM);
    CEnv::setConnManager(&CM);
    CEnv::setWriter(&WR);

    LP.init();

    //-------------------------------------------------------------------
    // 应用程序初使化

    CDataEvent DE;
    CEnv::setDataEvent(&DE);

    CListenConn listenConn(1981);

    //-------------------------------------------------------------------
    // 启动事件循环

    LP.runLoop();

    return 0;
}
```

样例侦听1981端口，数据处理CDataEvent类简单的回显消息，实际应用在这里应解析协议并进行分派。

关于消息分派这块，需要应用层根据协议类型来进行编写，形态可以是MFC风格，也可以是IF/ELSE风络，这个就看使用者的习惯了。

最后，请有需要的朋友准备好您的原创作品，并留言给微码库，只要作品验证合格，就能交换到这款源代码。