事件处理

EPOLLIN = 0x001,
EPOLLPRI = 0x002,
EPOLLOUT = 0x004,
EPOLLRDNORM = 0x040,
EPOLLRDBAND = 0x080,
EPOLLWRNORM = 0x100,
EPOLLWRBAND = 0x200,
EPOLLMSG = 0x400,
EPOLLERR = 0x008,
EPOLLHUP = 0x010,
EPOLLRDHUP = 0x2000,
EPOLLWAKEUP = 1u << 29,
EPOLLONESHOT = 1u << 30,
EPOLLET = 1u << 31,

EPOLLIN 有新连接进来触发 对端普通数据进来触发 对端正常关闭连接触发（此时还可能触发EPOLLRDHUP）
EPOLLPRI 通过OOB方式来发送的数据触发
EPOLLOUT 水平触发模式下，只要发送缓冲区没满就一直触发，边沿触发模式下发送缓冲区从高水位进入低水位时触发（水位可以设置默认是1）
EPOLLERR socket能检测到对方出错吗？目前为止，好像我还不知道如何检测，但是，在给已经关闭的socket写时，会发生EPOLLERR，也就是说，
只有在采取行动（比如读一个已经关闭的socket，或者写一个已经关闭的socket）时候，才知道对方是否关闭了。这个时候如果对方关闭了，
则会出现EPOLLERR，EPOLLERR是服务器这边出错
EPOLLHUP 一般是收到RST包。
EPOLLERR|EPOLLHUP 这两个标记epoll_wait会默认检测，不需要设置
EPOLLRDHUP 这个在有些系统中表示对端已经关闭，在我们库里面完全可以由EPOLLIN事件来判断，所以没有用

epoll为什么高效有这么几点原因：
1. epoll是一个有状态的接口，内核维护了一个列表，表示哪些fd对哪些事件敢兴趣，这样在epoll_wait的时候不用收集fd然后再传入，省掉了收集的时间，并且这个列表为了更改事件足够高效采用了红黑树数据结构。
2. 内核维护了另外一个列表，已经发生事件的fd列表，在事件发生时把该fd挂到该列表中。这样在epoll_wait的时候直接取这个列表就行。
EPOLLET跟EPOLLLT实现区别 EPOLLET在epoll_wait时直接清空了上面2的列表了，EPOLLLT，如果此时fd上还有未处理的事件，就把fd重新放回列表中。

事件处理代码
```
uint32_t nRawEvent = this->m_vecEpollEvent[i].events;
uint32_t nEvent = 0;
if (nRawEvent & (EPOLLHUP | EPOLLERR))
    nEvent = eNET_Send|eNET_Recv;
if (nRawEvent & (EPOLLIN | POLLPRI))
 nEvent |= eNET_Recv;
if (nRawEvent & EPOLLOUT)
 nEvent |= eNET_Send;
// 这里不处理EPOLLRDHUP事件
pNetBase->onEvent(nEvent);
```

这里错误处理采用把错误转成READ|WRITE事件，然后在这些事件处理函数中调用send跟recv来再次触发错误，最后来释放连接。