evbuffer包含在event2/buffer.h中，evbuffer作为一个数据缓存对象，其具有很多操作方法

**创建和释放evbuffer**

struct evbuffer \*evbuffer\_new(void);

void evbuffer\_free(struct evbuffer \*buf);

**evbuffer与线程安全**

默认情况下，在多个线程中同时访问 evbuffer 是不安全的

int evbuffer\_enable\_locking(struct evbuffer \*buf, void \*lock);

void evbuffer\_lock(struct evbuffer \*buf);

void evbuffer\_unlock(struct evbuffer \*buf);

evbuffer\_enable\_locking：

为evbuffer启用锁，如果 lock 参数为 NULL，则使用默认锁

evbuffer\_lock：上锁

evbuffer\_unlock：解锁

**获取存储的数据长度**

size\_t evbuffer\_get\_length(const struct evbuffer \*buf);

**添加数据**

int evbuffer\_add(struct evbuffer \*buf, const void \*data, size\_t datlen);

//添加 data 指向的数据 datalen 字节到 buf 的末尾

//成功时返回0，失败时返回-1。

**移动数据**

将一个evbuffer的数据移到另一个evbuffer中

int evbuffer\_add\_buffer(struct evbuffer \*dst, struct evbuffer \*src);

**添加数据到evbuffer前面**

int evbuffer\_prepend(struct evbuffer \*buf, const void \*data, size\_t size);

**从evbuffer中取出数据**

该函数从evbuffer前面取走数据，取走的数据从evbuffer中删除

int evbuffer\_remove(struct evbuffer \*buf, void \*data, size\_t datlen);

**删除数据**

函数直接删除evbuffer前面len个数据

int evbuffer\_drain(struct evbuffer \*buf, size\_t len);

**拷贝数据**

有时候我们想观看数据但却不想从evbuffer中移除

ev\_ssize\_t evbuffer\_copyout(struct evbuffer \*buf, void \*data, size\_t datlen);

**基于socket的读写**

int evbuffer\_write(struct evbuffer \*buffer, evutil\_socket\_t fd);

int evbuffer\_read(struct evbuffer \*buffer, evutil\_socket\_t fd, int howmuch);

evbuffer\_write：将buffer的数据写到fd中

evbuffer\_read：从fd中读取至多howmuch个字节的数据到buffer中

如果使用bufferevent，则不需要调用这些函数，bufferevent的代码已经为你调用了

**向evbuffer添加文件**

正常情况要转输文件数据需要将文件数据读入用户态空间，然后再将用户态的数据复制到内核态，使用如下方法直接在内核态读取文件而不经过用户态

int evbuffer\_add\_file(struct evbuffer \*output, int fd, ev\_off\_t offset, size\_t length);

将指定文件描述符fd的offset位置开始的length个字节复制到evbuffer

**示例**#include <event2/event.h>

#include <iostream>

#include <event2/bufferevent.h>

#include <sys/socket.h>

#include <memory>

#include <sys/types.h>

#include <unistd.h>

#include <signal.h>

#include <string.h>

#include <event2/buffer.h>

// 如果缓存区有数据，将被调用

void readcb(struct bufferevent \*bev, void \*ptr)

{

    char text[100] = {0};

    // 获取 bufferevent 的 inputevbuffer

    auto inputEvBuffer = bufferevent\_get\_input(bev);

    // 获取缓存区的数据长度

    std::cout << "缓存区数据长度：" << evbuffer\_get\_length(inputEvBuffer) << std::endl;

    // 读取缓存区数据

    if(evbuffer\_remove(inputEvBuffer, text, sizeof(text)) < 0){

        std::cout << "读取数据错误" << std::endl;

        return;

    }

    std::cout << text << std::endl;

}

// 如果向缓存区写入数据，且可以写入，则被调用

void wirtecb(struct bufferevent \*bev, void \*ptr)

{

    std::cout << "wirtecb 被调用" << std::endl;

}

// 发生事件时被调用

void eventcb(struct bufferevent \*bev, short events, void \*ptr)

{

    if (events & BEV\_EVENT\_CONNECTED)

    {

        // 连接完成后进入

        std::string hello = "Hello World";

        // 获取 outputbuffer

        auto outputEvBuffer = bufferevent\_get\_output(bev);

        // 向缓存区添加数据

        if(evbuffer\_add(outputEvBuffer, hello.c\_str(), hello.size()) == -1){

            std::cout << "向 evbuffer 写入数据失败" << std::endl;

            return;

        }

    }

    else if (events & BEV\_EVENT\_ERROR)

    {

        // 发生了其他错误

    }

}

int main(void)

{

    signal(SIGCHLD, SIG\_IGN);

    // 设置Socket地址

    struct sockaddr\_in sin;

    memset(&sin, 0, sizeof(sin));

    sin.sin\_family = AF\_INET;

    sin.sin\_addr.s\_addr = htonl(0x7f000001); /\* 127.0.0.1 \*/

    sin.sin\_port = htons(8080);              /\* Port 8080 \*/

    if (fork() > 0)

    {

        // 父进程

        auto socketFd = socket(AF\_INET, SOCK\_STREAM, 0);

        // 绑定

        if (bind(socketFd, (sockaddr \*)(&sin), sizeof(sin)) < 0)

        {

            std::cout << "bind 失败" << std::endl;

            return 0;

        }

        // 设置监听套接字

        if (listen(socketFd, 5) != 0)

        {

            std::cout << "listen 失败" << std::endl;

            return 0;

        }

        while (true)

        {

            sockaddr\_in cliaddr;

            socklen\_t cliaddr\_len = sizeof(cliaddr);

            // 接收连接

            auto confd = accept(socketFd, (sockaddr \*)&cliaddr, &cliaddr\_len);

            if (confd < 0)

            {

                std::cout << "accept 失败" << std::endl;

                continue;

            }

            else

            {

                std::cout << "accept 成功" << std::endl;

            }

            // 接收数据

            char recvText[100] = {0};

            recv(confd, recvText, sizeof(recvText), 0);

            std::cout << recvText << std::endl;

            // 发送数据

            std::string hello = "Hello Text Too";

            send(confd, hello.c\_str(), hello.size(), 0);

            close(socketFd);

            close(confd);

            return 0;

        }

    }

    else

    {

        // 子进程

        sleep(2);

        struct event\_base \*base = event\_base\_new();

        struct bufferevent \*bev = bufferevent\_socket\_new(base, -1, BEV\_OPT\_CLOSE\_ON\_FREE);

        // 设置回调函数

        bufferevent\_setcb(bev, readcb, wirtecb, eventcb, NULL);

        // 默认设置了 EV\_WRITE，设置 EV\_READ（读取） | EV\_PERSIST（持久）

        bufferevent\_enable(bev, EV\_READ | EV\_PERSIST);

        // 连接服务器，连接成功 eventcb 将被调用

        if (bufferevent\_socket\_connect(bev,

                                       (struct sockaddr \*)&sin, sizeof(sin)) < 0)

        {

            std::cout << "bufferevent\_socket\_connect 失败" << std::endl;

            bufferevent\_free(bev);

            return -1;

        }

        // 启动 event\_base

        event\_base\_dispatch(base);

        std::cout << "子进程退出" << std::endl;

    }

    return 0;

}