BlmComm模块的大体设计：先看使用的例子，一共有两个，后面列出了相关的例子

class MainTest {

BlmClient\* pcli;

public:

MainTest(BlmClient\* cli) { pcli = cli; }

int onConnect(void\*) {

//发送登录消息

string cont = formatString("{uinput:\"%s\",pwd:\"%s\",ver:\"%s\",mac:\"\",tp:\"%s\"}", "dongfuye", "yedongfu", "2", "1");

pcli->sendMsg(0x00, 0x03, cont.c\_str(), cont.size());

return 0;

}

int handleLogin(int mt, int st, const string& data, void\*) {

blmlog(3, "main", "login result: %s", data.c\_str());

return 0;

}

};

void test1() {

//轮询器

BlmPoller poller;

//BlmClient代表一个与服务器的连接

BlmClient blmclient(&poller);

MainTest mtest(&blmclient);

blmclient.addConnectCallback(&mtest, &MainTest::onConnect);

blmclient.addMsgCallback(0x00, &mtest, &MainTest::handleLogin);

blmclient.connect("222.73.34.216", 5224);

for (int i = 0; i<20; i++) {

poller.poll(50);

}

blmclient.removeAllCallback(&mtest);

}

//这个例子演示了如何连接最快的服务器，并且演示了定时器的使用

BlmClient\* bj, \*sh;

bool timerfired;

class Test2 {

public:

Test2() {}

int onConnect(void\* cli) {

BlmClient\* cl = (BlmClient\*)cli;

blmlog(3, "main", "%s is connected", cl == bj ? "bj" : "sh");

return -1; //关闭连接

}

int onTimer(void\* cli) {

timerfired = true;

return 0;

}

};

void test2() {

BlmPoller poller2;

Test2 t2;

bj = new BlmClient(&poller2);

sh = new BlmClient(&poller2);

bj->addConnectCallback(&t2, &Test2::onConnect, bj);

sh->addConnectCallback(&t2, &Test2::onConnect, sh);

poller2.addTimerCallback(3000, &t2, &Test2::onTimer);

bj->connect("123.125.17.116", 5224);

sh->connect("222.73.34.216", 5224);

while (!timerfired) {

poller2.poll(50);

}

//…省略了注销等调用

}

int main() {

blminit();

//设置日志，可以多次设置，

//对于detail\_level，为最低级别，错误、警告以级别输出，为普通信息输出，为最详细的输出

//modules指定输出那些模块的信息，例如：""全部输出，"blmcomm main"输出blmcomm与main模块的日志

//void blmSetLog(int detail\_level=3, const char\* modules="", const char\* logfile="");

blmSetLog(3);

test1();

test2();

}

//用户使用它与服务器通信，发送请求，处理数据

class BlmClient {

public:

BlmClient(BlmPoller\* poller1);

//与服务器建立连接

void connect(const char\* ip, short port);

//断开连接

void disconnect();

//发送请求

void sendMsg(int mt, int st, const char\* data, int sz);

//以下与回调相关，只支持成员函数的回调（这样可以保持接口的简洁性，又能满足绝大部分的需求）

//注册回调时，请注意下面的函数是以模板的方式给出，请确保成员函数与add\*Callback的声明相匹配

//成员函数的返回值为int，返回表示成功，返回其他表示出错，此时连接会被关闭

//成员函数的最后一个参数为void\*，进行回调时，会把注册回调时传递的arg参数传递给成员函数，与启动线程时传递的参数方式相同

//返回值类型为void\*，指向内部使用的回调对象，详见removeCallback

//注册回调，建立连接后，进行回调

template<class C> void\* addConnectCallback(C\* obj, int (C::\*memFn)(void\*), void\* arg=0);

//注册回调，断开连接时，进行回调

template<class C> void\* addCloseCallback(C\* obj, int (C::\*memFn)(void\*), void\* arg=0);

//注册回调，收到指定的mt消息包时，进行回调

template<class C> void\* addMsgCallback(int mt, C\* obj, int (C::\*memFn)(int, int, const string&, void\*), void\* arg=0);

//注销回调，有两种方式，通常在对象的析构中调用removeAllCallback，更细粒度的控制则调用removeCallback

//上面注册的所有回调都可用下面的函数注销

//注销回调，传入的参数为注册回调时返回的指针

//成功返回，失败返回-1

int removeCallback(void\* cb);

//注销与此对象相关的所有回调，传入的参数为注册回调时所使用的obj参数

//成功返回，失败返回-1

int removeAllCallback(void\* obj);

};

//轮询器，用户可以在此注册定时器函数

class BlmPoller {

public:

BlmPoller();

//轮询网络数据，调用数据处理的回调，轮询定时器，调用超时的定时器

void poll(int milliseconds);

//注册定时器的回调，参数详见BlmClient中，注册函数的说明

template<class T> void\* addTimerCallback(int interval, T\* obj, int (T::\*mfn)(void\*), void\* arg=0) ;

//注销回调

int removeTimer(void\* timer);

}