# 轻量级网络引擎——libevwork

**介绍：**

几个月前，为了帮助朋友公司提高后台程序的稳定性，我编写了libevwork库，当时还发表了一篇介绍性的文章，原文链接：http://mp.weixin.qq.com/s?__biz=MzIxOTEyOTc5Nw==&mid=405617216&idx=1&sn=790b0355b270daafd91ca29ef984df89#rd，或者关注“微码库”公众号，通过消息历史查看。

开发优秀的网络服务，是一门技术活，这要求程序员具备较强的技术功底，经验不足的程序员由于对系统知识缺乏全面了解，往往难以写出高质量的底层代码。因此，很多专业团队，为了降低风险，避免重复造轮子，都有自己通用的底层框架，如微信的serverkit，网易和YY的servercommon。

不过，你也可以选择网络上免费成熟的开源软件，如libev、libevent、ACE、boost和asio，经验表明用好这些库并不是那么轻松。libevwork基于libev事件库，屏蔽了底层复杂的逻辑环节，向开发者呈现MFC风格的调用方式，其主要特点如下：

- 高并发，单进程可承载数W客户端连接
- 高吞吐，单进程每秒可发送/接收数百MB数据
- 支持多线程模型，可用于开发IO等待型业务
- 支持yy、json、protobuf等多种传输协议
- 提供MFC风络的编程接口

**编程风络：**

很多手游产品采用json作为传输协议，我们现在来看一下基于libevwork如何编写代码：

声明：

```
#include "libevwork/FormDef.h"

class CDispatch
    : public evwork::PHClass
{
public:
    DECLARE_YY_FORM_MAP;

    void onSysRateInfo(evwork::Jpacket& packet, evwork::IConn* pConn);
    void onSysSetRate(evwork::Jpacket& packet, evwork::IConn* pConn);
    void onSysCtrlInfo(evwork::Jpacket& packet, evwork::IConn* pConn);
    void onSysSetCtrl(evwork::Jpacket& packet, evwork::IConn* pConn);
    void onSysRevCtrl(evwork::Jpacket& packet, evwork::IConn* pConn);
};
```

实现：

```
using namespace evwork;

BEGIN_YY_FORM_MAP(CDispatch)
    ON_YY_REQUEST_CONN(SYS_RATEINFO, &CDispatch::onSysRateInfo)
    ON_YY_REQUEST_CONN(SYS_SETRATE, &CDispatch::onSysSetRate)
    ON_YY_REQUEST_CONN(SYS_CTRLINFO, &CDispatch::onSysCtrlInfo)
    ON_YY_REQUEST_CONN(SYS_SETCTRL, &CDispatch::onSysSetCtrl)
    ON_YY_REQUEST_CONN(SYS_REVINFO, &CDispatch::onSysRevCtrl)
END_YY_FORM_MAP()

void CDispatch::onSysRateInfo(evwork::Jpacket& packet, evwork::IConn* pConn)
{
    std::string strPeerIp = "";
    uint16_t uPeerPort16 = 0;
```

```
    pConn->getPeerInfo(strPeerIp, uPeerPort16);

    LOG(Info, "[CDispatch::%s] from:[%s:%u] get rateinfo", __FUNCTION__, strPeerIp.c_str(), uPeerPort16);

    Jpacket packet_r;
    packet_r.val["cmd"] = SYS_RATEINFO;

    const MAP_TYPE_FISH_t& mapTypeFish = CAPPConfig::getFishConfigMap();

    for (MAP_TYPE_FISH_t::const_iterator c_iter = mapTypeFish.begin(); c_iter != mapTypeFish.end(); ++c_iter)
    {
        const SFishConf& fishConf = c_iter->second;

        Json::Value valRate;
        valRate["type"] = fishConf.type;
        valRate["rate"] = fishConf.capture;

        packet_r.val["fishs"].append(valRate);
    }

    packet_r.end();

    pConn->sendBin(packet_r.tostring().data(), packet_r.tostring().size());
}
```

protobuf作为谷歌的精品，近两年应用越来越普及，用法如下：

声明：

```
#include "pbProtocol.pb.h"
#include "libevwork/pbmfc/Sender.h"

enum
{
    MPubReq_CMD = 1,
    MPubRes_CMD = 2,
};

class CService
        : public pb::PHClass
{
public:
        DECLARE_PB_FORM_MAP;

        void onMPubReq(MPubReq* pObj, evwork::IConn* pConn);
};
```

实现：

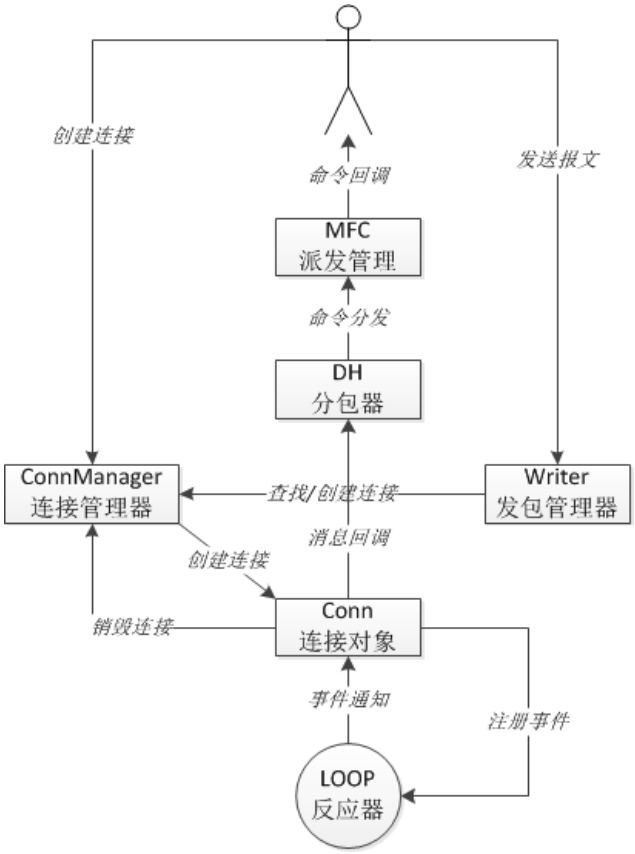
```
using namespace evwork;

BEGIN_PB_FORM_MAP(CService)
    ON_PB_REQUEST_CONN(MPubReq_CMD, MPubReq, &CService::onMPubReq)
END_PB_FORM_MAP()

void CService::onMPubReq(MPubReq* pObj, evwork::IConn* pConn)
{
    std::string strPeerIp = "";
    uint16_t uPeerPort16 = 0;
    pConn->getPeerInfo(strPeerIp, uPeerPort16);

    ETMCode emCode = E_TM_SUCCESS;

    // 遍历取消息，写入
    int pubCnt = 0;
    for ( ; pubCnt < pObj->binlogs_size(); )
    {
        const MBinLog& binlog = pObj->binlogs(pubCnt);
```

```
        emCode = CMSEnv::s_pTitleLayer->local_append(strTopic, binlog.type(), binlog.message(), getTimeNow());
        if (emCode != E_TM_SUCCESS)
            break;

        ++pubCnt;
    }

    // 发回响应

    MPubRes objRes;
    objRes.set_topic(strTopic);
    objRes.set_pubcount(pubCnt);

    if (emCode != E_TM_SUCCESS)
    {
        if (emCode == E_TM_PARAM_INVALID)
            objRes.set_rescode( MPubRes::BINLOG_EMPRY_OR_LIMIT );
        else
            objRes.set_rescode( MPubRes::UNKOWN );

        LOG(Error, "[CService::%s] from:[%s:%u] topic:[%s] binlog count:[%d] append count:[%d] last code:[%d]", __FUNCTION__,
            strPeerIp.c_str(), uPeerPort16, strTopic.c_str(), pObj->binlogs_size(), pubCnt, emCode);
    }
    else if (pubCnt < pObj->binlogs_size())
    {
        objRes.set_rescode( MPubRes::BINLOG_EMPRY_OR_LIMIT );

        LOG(Error, "[CService::%s] from:[%s:%u] topic:[%s] binlog count:[%d] append count:[%d] binlogsize is limit",
__FUNCTION__,
            strPeerIp.c_str(), uPeerPort16, strTopic.c_str(), pObj->binlogs_size(), pubCnt);
    }
    else
    {
        objRes.set_rescode( MPubRes::SUCCESS );

        LOG(Info, "[CService::%s] from:[%s:%u] topic:[%s] binlog count:[%d] append count:[%d] success", __FUNCTION__,
            strPeerIp.c_str(), uPeerPort16, strTopic.c_str(), pObj->binlogs_size(), pubCnt);
    }

    pb::Sender sdr;
    sdr.SerializePB(MPubRes_CMD, objRes);

    pConn->sendBin(sdr.Data(), sdr.Size());
}
```

留意上面红色的代码段部分，是不是跟MFC风格很像啊？

**设计思想：**

总的来说，libevwork继承了网易servercommon的OO设计，抽像了几个基本的接口对象，并且这些接口之间采用组合，非常方便程序员重载。我们看下主要的接口设计：




**关于性能：**

libevwork每一行代码都精心设计，秉着轻量高性能的原则，几乎没有冗余的代码，也不允许出现明显的性能牺牲。比如，在缓冲区设计上，我们用std::string、网易/YY的blockbuffer、和金磊同学的SocketBuffer、以及libevwork的CBuffer来做对比测试。先看下面的代码：

```
for (int i = 0; i < 10000; ++i)
{
    buffer.Append(pBuff, 1000000);

    for (int j = 0; j < 100; ++j)
        buffer.Erase(10000);
}
```

模拟向缓冲区写入10000次压包操作，并且每个包通过100次事件完成发送，统计时间消耗，对比如下：
std::string用时：约30S
blockbuffer用时：约30S
SocketBuffer用时：约30S
CBuffer用时：约18S
可以看到除了CBuffer外，其他3个性能都差不多，blockbuffer和SocketBuffer相对std::string的优点是按块分配，更节省内存。之所以CBuffer性能最高，是因为我们在设计上模拟了环形缓冲移位的特点，尽量减小收发包时内存搬移的消耗。
此外，如果对CBuffer的参数加以调整，模拟更大的环形缓冲，最优性能只需要0.6S!

下面是网络相关的性能测试指标，表列如下：

| 最大收包数量/S | 最大发包数量/S | 最大收包吞吐/S | 最大发包吞吐/S |
|---|---|---|---|
| 60K | 60K | 800M | 800M |

关于收发包数量性能，60K可能不是十分满意，然而这已经几乎接近单线程的系统极限了，因为每一次send/recv系统调用都需要耗时10~20us。实际应用中，我们通常都是将小包压入缓冲区，通过发大包的方法，减小send/recv的调用次数，每秒收发包数量可达数百K以上。

**结语：**
网络引擎是构建分布式的基础，然而分布式框架非常复杂，需要更强大的库包装。目前来说，利用libevwork编写网游后台、分布式组件还是比较合适的，有需要的朋友可以联系我。