Taf Server

2015年12月1日 19:49

1. 步骤
   a. initializeClient
      i. //初始化通信器
      _communicator = CommunicatorFactory::getInstance()->getCommunicator(_conf);
      设置代理网络线程数目和属性

```
54      string defaultValue = "dft";
55      if ((defaultValue == getProperty("enableset", defaultValue))
56              || (defaultValue == getProperty("setdivision", defaultValue)))
57      {
58          _properties["enableset"] = conf.get("/taf/application<enableset>", "n");
59          _properties["setdivision"] = conf.get("/taf/application<setdivision>", "NULL");
60      }
```

   b. initializeServer
      i. //初始化servert
      设置应用信息，初始化网络线程数目，建立与网络线程数目对应的epollServer；
      ii. 建立管理Servant，主控
   c. 绑定对象和端口
      i. 将配置文件中的具体业务的adapter添加到adapter的vector中（设置每个adapter的属性）；
   d. 业务逻辑的初始化
      i. 添加自己的Servant；
         1) 可以用Taf自带的Jce编译工具来通过接口生成我们自己的Servant的.h和.cpp文件，在IMP文件中实现我们的服务；
         2) 可以自己新建自定义Servant（继承Taf::servant），需要定义其中的doRequest，sendResponse，doResponse，doResponseException，doResponseNoRequest等相关函数
      ii. 调用addServant:
         1) 创建obj名字对应的servant对象；
   e. 设置handleGroup分组，启动线程
      i. 根据具体业务的adapters设置具体的servant组；
      setHandle(adapters[i]);

```
void Application::setHandle(TC_EpollServer::BindAdapterPtr& adapter)
{
    adapter->setHandle<ServantHandle>();
}
```

      创建相应个数的hanlde类（_iHandleNum）

```
template<typename T> void setHandle()
{
    _pEpollServer->setHandleGroup<T>(_handleGroupName, _iHandleNum, this);
}
```

      为每个servantHandle创建挂载的bindAdapter和handleGroup,

```
template<class T> void setHandleGroup(const string& groupName, int32_t handleNum, BindAdapterPtr adapter)
{
    map<string, HandleGroupPtr>::iterator it = _handleGroups.find(groupName);

    if (it == _handleGroups.end())
    {
        HandleGroupPtr hg = new HandleGroup();

        hg->name = groupName;

        adapter->_handleGroup = hg;

        for (int32_t i = 0; i < handleNum; ++i)
        {
            HandlePtr handle = new T();

            handle->setEpollServer(this);

            handle->setHandleGroup(hg);

            hg->handles.push_back(handle);
        }

        _handleGroups[groupName] = hg;

        it = _handleGroups.find(groupName);
    }
    it->second->adapters[adapter->getName()] = adapter;

    adapter->_handleGroup = it->second;
}
```

   f. epollServer启动handle

```
void TC_EpollServer::startHandle()
{
    if (!_handleStarted)
    {
        _handleStarted = true;

        map<string, TC_EpollServer::HandleGroupPtr>::iterator it;

        for (it = _handleGroups.begin(); it != _handleGroups.end(); ++it)
        {
            vector<TC_EpollServer::HandlePtr>& hds = it->second->handles;

            for (uint32_t i = 0; i < hds.size(); ++i)
            {
                if (!hds[i]->isAlive())
                {
                    hds[i]->start();
                }
            }
        }
    }
}
```

i. servant的run最终调用Handle的run方法，上行到servantHandle的run方法，servant未定义，上行到servant的run方法

```
void TC_EpollServer::Handle::run()
{
    initialize();

    handleImp();
}
```

ii. servant的initilize方法，在子类ServantHandle中被实现，这就是我们在addServant中添加的自己的Servant

```
for (adpit = adapters.begin(); adpit != adapters.end(); ++adpit)
{
    ServantPtr servant = ServantHelperManager::getInstance()->create(adpit->first);

    if (servant)
    {
        _servants[servant->getName()] = servant;
    }
```

对每一个Servant，调用自己的initialize函数进行初始化

```
while(it != _servants.end())
{
    try
    {
        it->second->setHandle(this);

        it->second->initialize();
```

调用void TC_EpollServer::Handle::handleImp()调用handle,

```
void ServantHandle::handle(const TC_EpollServer::tagRecvData &stRecvData)
{
    JceCurrentPtr current = createCurrent(stRecvData);

    if (!current) return;

    if (current->getBindAdapter()->isTafProtocol())
    {
        handleTafProtocol(current);
    }
    else
    {
        handleNoTafProtocol(current);
    }
}
try
{
    //业务逻辑处理
    sit->second->dispatch(current, buffer);
}
```

```
int Servant::dispatch(JceCurrentPtr current, vector<char> &buffer)
{
    int ret = JCESERVERUNKNOWNERR;

    if (current->getFuncName() == "taf_ping")
    {
        TLOGINFO("[TAF][Servant::dispatch] taf_ping ok from [" << current->getIp() << ":" << current->getPort() << "]" << end

        ret = JCESERVERSUCCESS;
    }
    else if (!current->getBindAdapter()->isTafProtocol())
    {
        TC_LockT<TC_ThreadRecMutex> lock(*this);

        ret = doRequest(current, buffer);
    }
    else
    {
        TC_LockT<TC_ThreadRecMutex> lock(*this);

        ret = onDispatch(current, buffer);
    }
    return ret;
}
```

对于非taf请求调用，下行到具体的Servant中的doRequest方法

```
int HttpHandleServant::doRequest(JceCurrentPtr current, vector<char> &response)
{
    vector<char> vRet;
    vector<char> vReqData = current->getRequestBuffer();
    std::string strReqData;
    strReqData.assign( current->getRequestBuffer().begin(), current->getRequestBuffer().end());

    if( m_pJava == 0  && g_app.getJavaVM() )
    {
```

如果有请求来，在TC_EpollServer::Handle::handleImp()中调用handleAsyncResponse处理对应的请求

```
  if (resp->response.iRet == JCESERVERSUCCESS)
  {
      it->second->doResponse(resp);
  }
  else if (resp->pObjectProxy == NULL)
  {
      it->second->doResponseNoRequest(resp);
  }
  else
  {
      it->second->doResponseException(resp);
  }
```

2. TC_EpollServer::waitForShutdown:

```
void TC_EpollServer::waitForShutdown()
{
    for (size_t i = 0; i < _netThreadNum; ++i)
    {
        _netThreads[i]->start();
    }
    debug("server netthread num : " + TC_Common::tostr(_netThreadNum));

    while(!_bTerminate)
    {
        {
            TC_ThreadLock::Lock sync(*this);
            timedWait(5000);
        }
    }
```

启动网络线程，监听端口

```
void TC_EpollServer::NetThread::run()
{
    //循环监听网路连接请求
    while(!_bTerminate)
    {
        _list.checkTimeout(TNOW);

        int iEvNum = _epoller.wait(2000);

        for(int i = 0; i < iEvNum; ++i)
        {
            try
            {
                const epoll_event &ev = _epoller.get(i);

                uint32_t h = ev.data.u64 >> 32;

                switch(h)
                {
                case ET_LISTEN:
                    {
                        //监听端口有请求
                        map<int, BindAdapterPtr>::const_iterator it = _listeners.find(ev.data.u32);
                        if( it != _listeners.end())
                        {
                            if(ev.events & EPOLLIN)
                            {
                                bool ret;
                                do
                                {
                                    ret = accept(ev.data.u32);
                                }while(ret);
                            }
                        }
                    }
                    break;
                case ET_CLOSE:
                    //关闭请求
                    break;
                case ET_NOTIFY:
                    //发送通知
                    processPipe();
                    break;
                case ET_NET:
                    //网络请求
                    processNet(ev);
                    break;
                default:
                    assert(true);
                }
```

```
void TC_EpollServer::NetThread::processNet(const epoll_event &ev)
{
    uint32_t uid = ev.data.u32;

    Connection *cPtr = getConnectionPtr(uid);

    if(!cPtr)
    {
        debug("TC_EpollServer::NetThread::processNet connection[" + TC_Common::tostr(uid) + "] not exist
        return;
    }

    if (ev.events & EPOLLERR || ev.events & EPOLLHUP)
    {
        delConnection(cPtr,true,EM_SERVER_CLOSE);

        return;
    }

    if(ev.events & EPOLLIN)               //有数据需要读取
    {
        recv_queue::queue_type vRecvData;

        int ret = recvBuffer(cPtr, vRecvData);

        if(ret < 0)
        {
            delConnection(cPtr,true,EM_CLIENT_CLOSE);

            return;
        }

        if(!vRecvData.empty())
        {
            cPtr->insertRecvQueue(vRecvData);
        }
    }
```