TAF RPC学习笔记

大纲

RPC 简介

1.1 RPC基础概念

TAF RPC实现简析

2.1 TAF路由

2.2 TAF IDL

2.3 TAF队列&连接管理

TAF服务运维& QA

3.1 TAF服务运维

RPC介绍

RPC基础概念

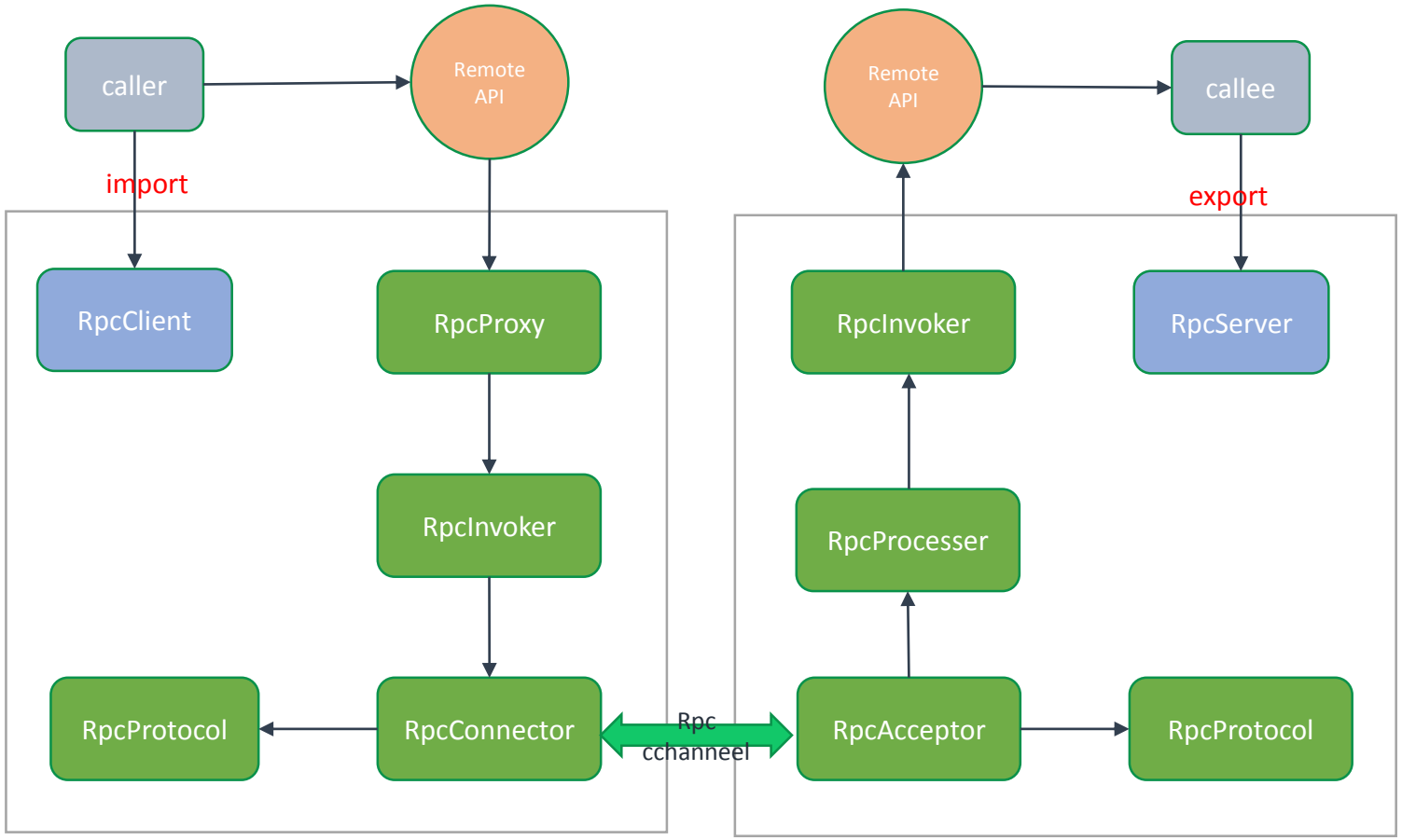
1. 定义：

RPC 全称是 Remote Procedure Call 是一种进程间通信方式。

它允许程序调用另一个地址空间（通常是共享网络的另一台机器上）的过程或函数，而不用程序员显式编码这个远程调用的细节。即程序员无论是调用本地的还是远程的，本质上编写的调用代码基本相同;

2.动机：分布式计算

3.实现：见右图



TAF RPC实现简析(路由)

```
module Test
{
interface A
{
   string test(long p);
};
};
```

```
class A : public taf::Servant
{

virtual string test(Int64 p,
taf::JceCurrentPtr current) = 0;

 }.
```

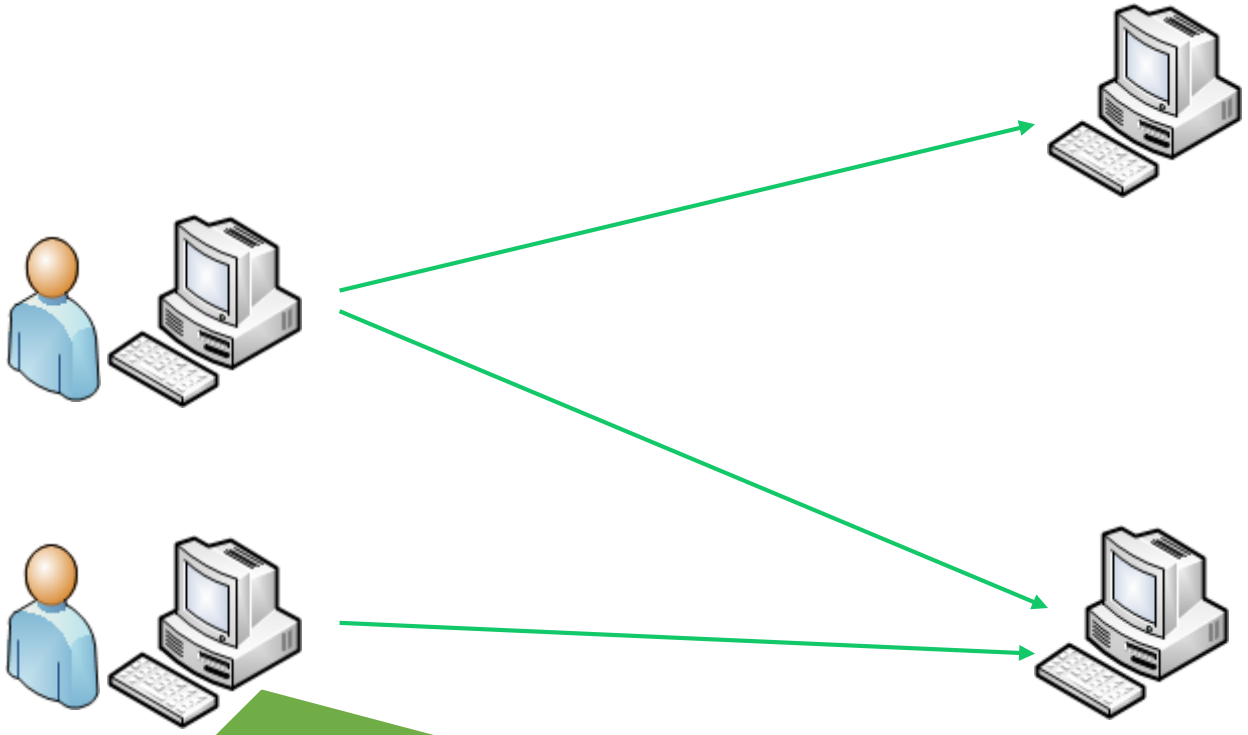
```
class AImp
{
virtual string test(int64 p,
taf::JceCurrentPtr current)
{
//TODO:业务逻辑写在这
```

- 客户端如何调用到服务端的test函数？
- 异步回来，如何进入callback_test函数？

```
class TT
{
public:
 virtual void callback_test (...)
 {
  //TODO:business logic
 }
};
```

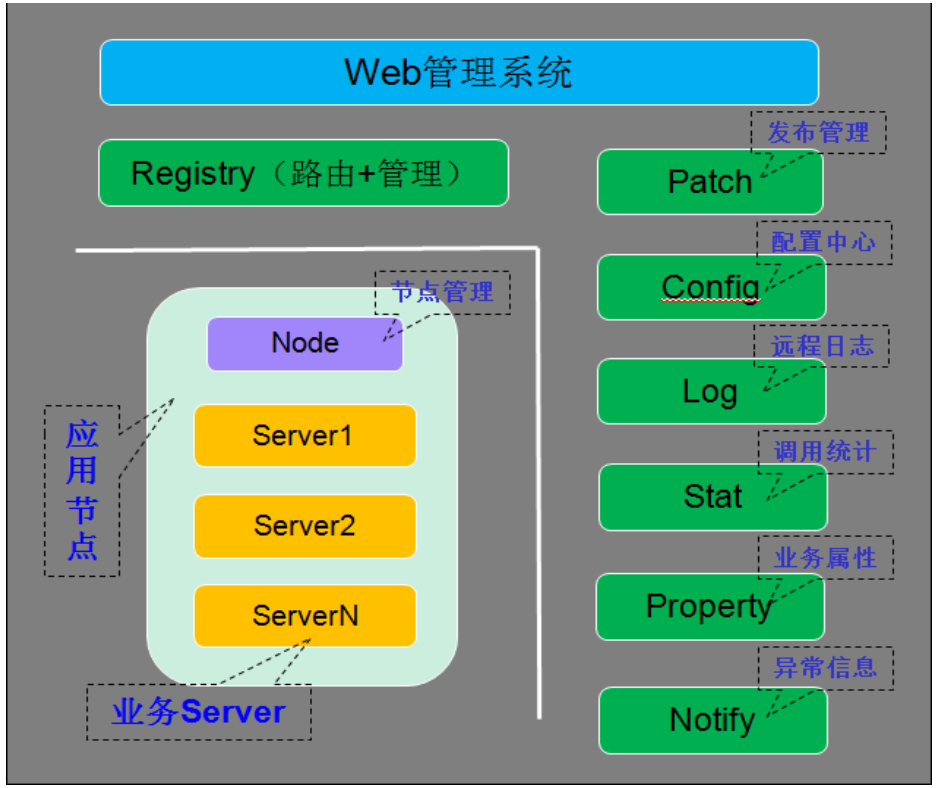
```
TestServantPrx  prx =
comm.stringToProxy<TestServantPrx> (“..TestServant");
APrxCallbackPtr cb=new TTCallback ();
int ret = prx >async_test(cb,...);
```

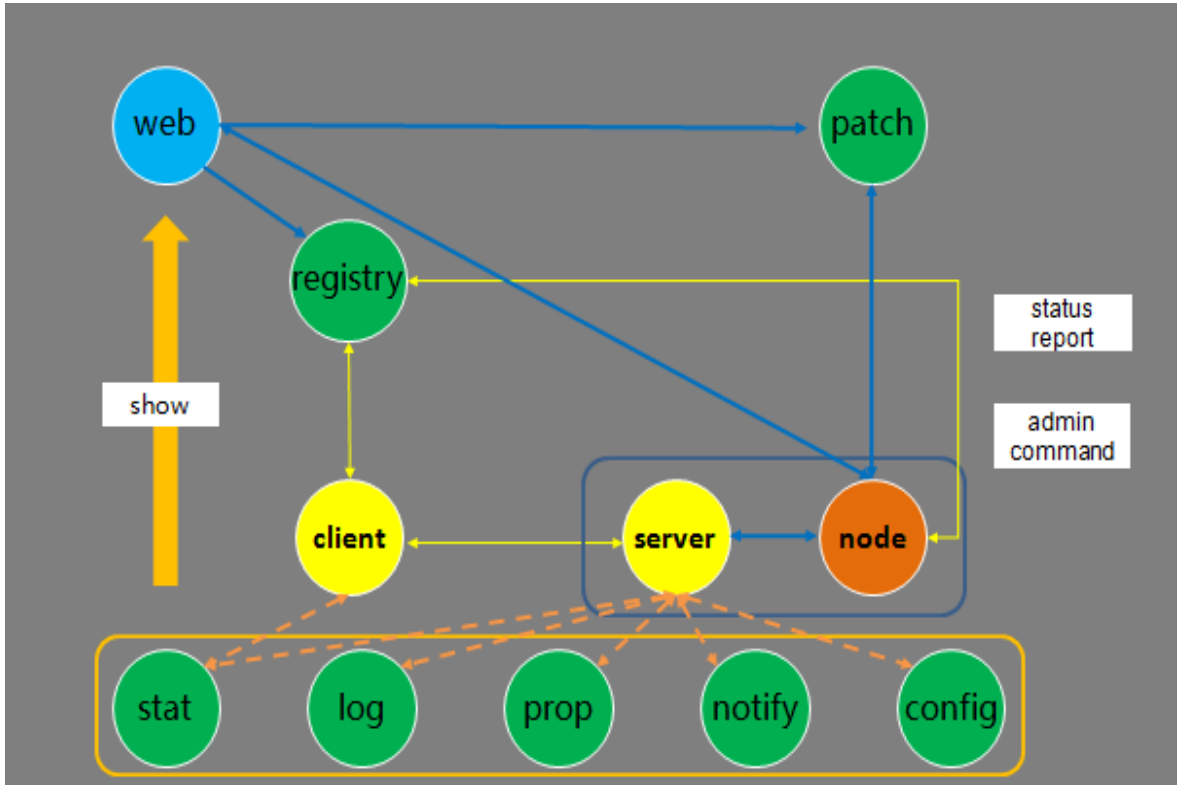
路由问题分解



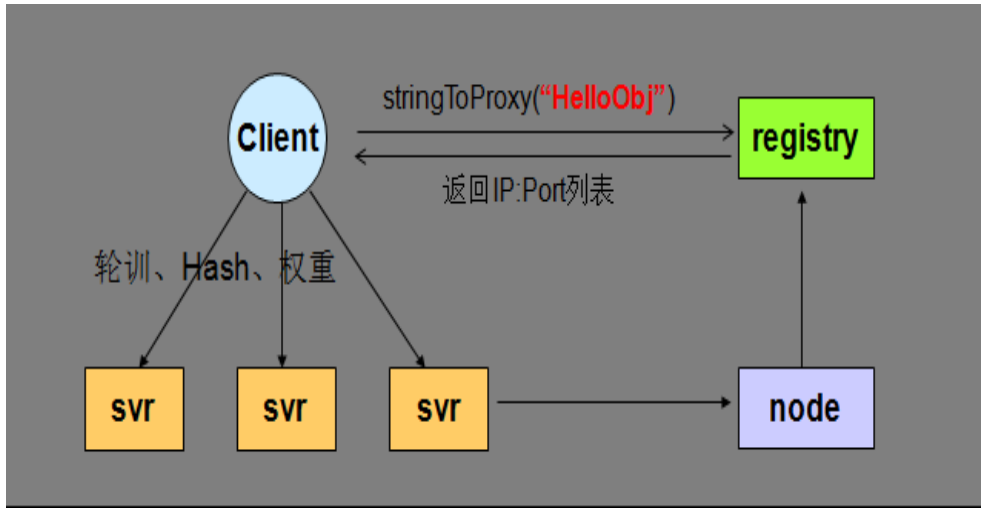
- 1.*如何定位服务节点？*----------client
- 2.如何定位要调用的对象？-----server
- 3.如何定位要调用的方法？-----server

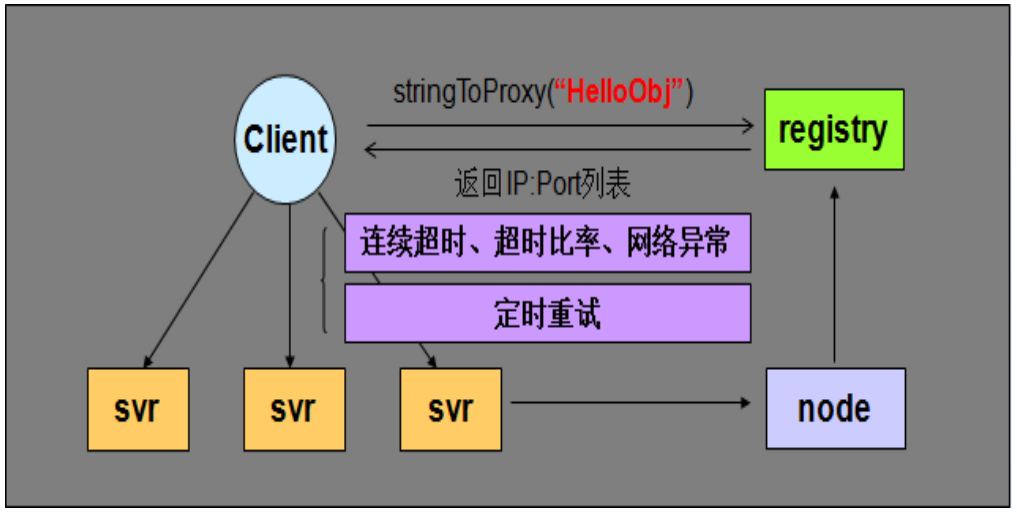
客户端定位服务端


Web管理系统
Registry（路由+管理）
发布管理
Patch
配置中心
Config
远程日志
Log
调用统计
Stat
业务属性
Property
异常信息
Notify
节点管理
Node
应用节点
Server1
Server2
ServerN
业务Server


web
patch
registry
show
status report
admin command
client
server
node
stat
log
prop
notify
config

客户端定位服务端





如何发现服务节点：

TAF框架通过名字服务来实现服务的注册与发现，Client通过访问名字服务获取到被调服务的地址信息列表；

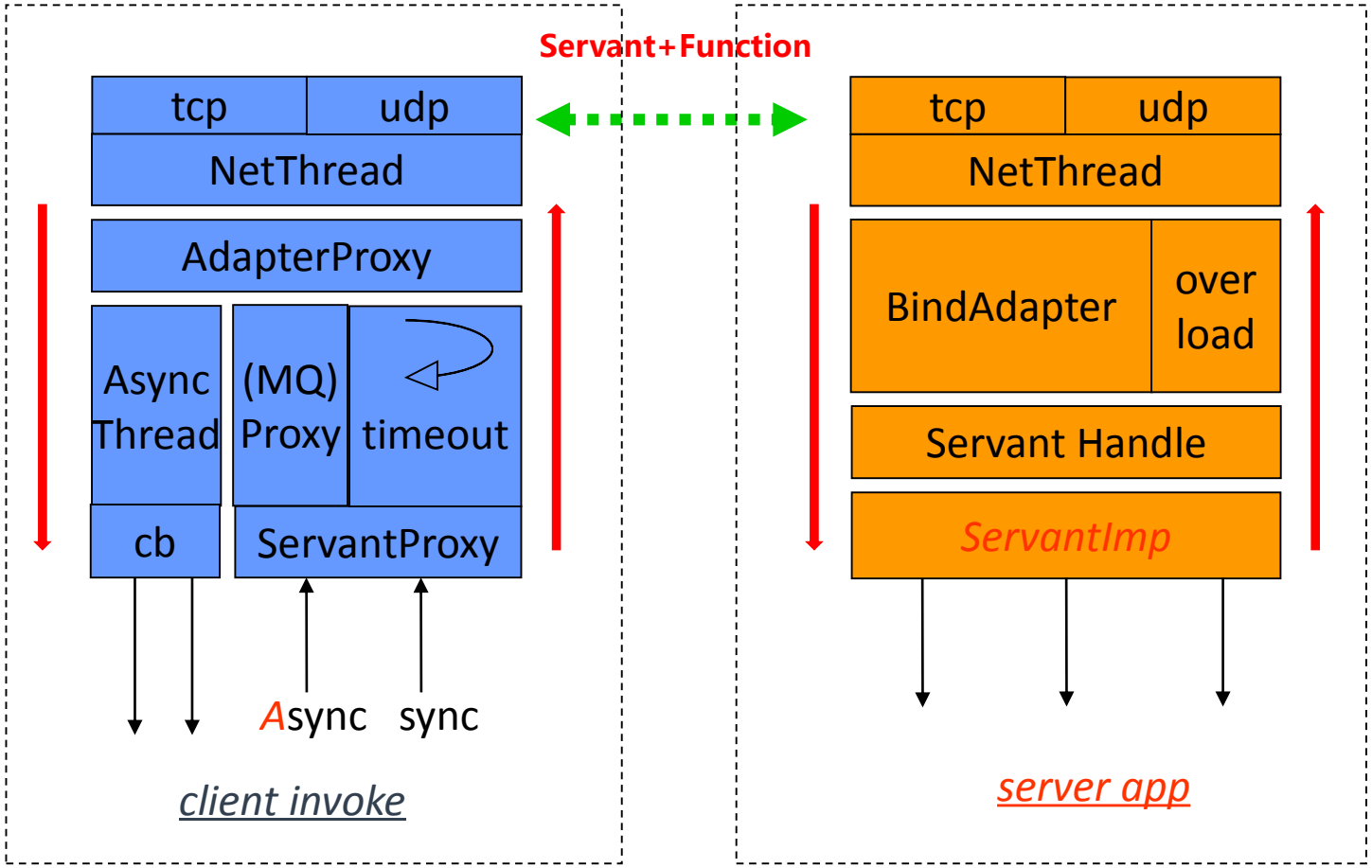
Client再根据需要选择合适的负载均衡方式来调用服务。

如何容错：

名字服务排除的策略：通过心跳

Client主动屏蔽：通过超时率屏蔽,重连恢复

对象方法路由


Servant+Function
tcp
udp
NetThread
AdapterProxy
Async Thread
(MQ) Proxy
timeout
cb
ServantProxy
Async
sync
client invoke
tcp
udp
NetThread
BindAdapter
over load
Servant Handle
ServantImp
server app

客户端路由

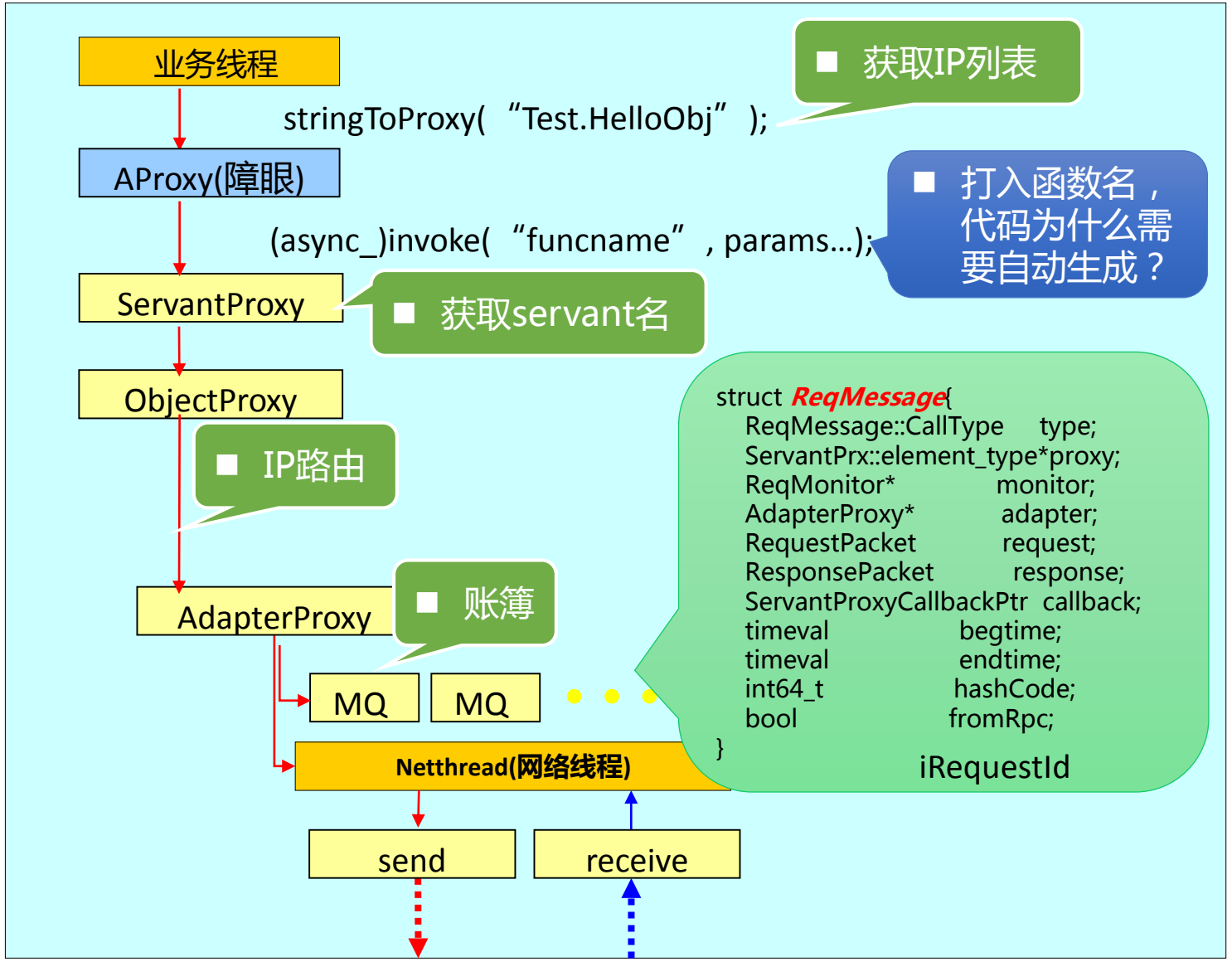
```
//采样信息需要透传
msg->sampleKey = pSptd->_sampleKey;
//调用广度要+1
pSptd->_sampleKey._width ++;

//设置超时时间
msg->request.iTimeout = (ReqMessage::SYNC_C

//判断是否针对接口级设置超时
if(pSptd->_bHasTimeout)
{
    msg->request.iTimeout = (pSptd->_iTimeo
    pSptd->_bHasTimeout = false;
}

ObjectProxy * pObjProxy = NULL;
ReqInfoQueue * pReqQ = NULL;
selectNetThreadInfo(pSptd,pObjProxy,pReqQ);

//调用发起时间
msg->iBeginTime = TNOWMS;
msg->pObjectProxy = pObjProxy;
```



路由-客户端关键代码

■ 生成代码

```
int AProxy::test(stReq,stRsp)
```

```
int ServantProxy::invoke(ReqMessagePtr&req)
```

```
selectNetThreadInfo(pSptd,pObjProxy,pReqQ);
```

```
pObjProxy->getCommunicatorEpoll()->notify(pSptd->_iReqQNo, pReqQ);
```

```
int ObjectProxy::invoke(ReqMessagePtr&req)
```

```
AdapterProxy*adp=selectAdapterProxy(req);
return adp->invoke(req);
```

```
int AdapterProxy::invoke(ReqMessagePtr&req)
```

```
_pTrans->sendRequest(msg->sReqData.c_str(),msg->sReqData.size()
```

```
if (events & EPOLLOUT)
{
        try
        {
            handleOutputImp(pTransceiver);
        }
}
```

路由-服务端关键代码

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

    handleImp();
}
```

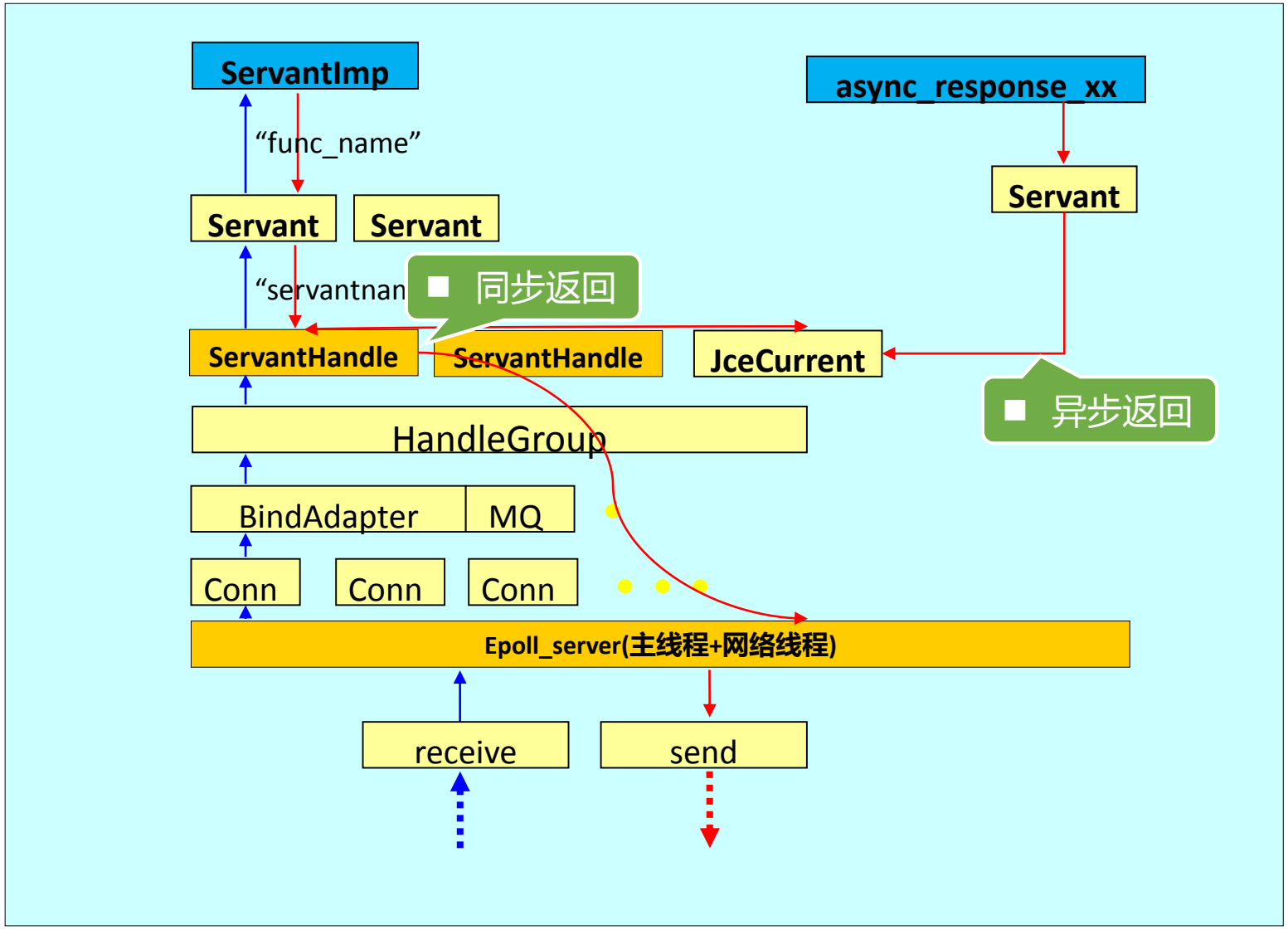
```
ServantHandle::initialize()

map<string, TC_EpollServer::BindAdapterPtr>::iterat

map<string, TC_EpollServer::BindAdapterPtr>& adapte
    = _handleGroup->adapters;

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

    if (servant)
    {
        _servants[servant->getName()] = servant;
    }
    else
    {
        TLOGERROR("[TAF]ServantHandle initialize cr
    }
}
```



路由-服务端关键代码

■ 生成代码

```
int XXServert::onDispatch(current,buffer)
    r=equal_range(,current->getFuncName())
```

```
int Servant::dispatch(current,buffer)
    ret=onDispatch(current,buffer)
```

```
map<string, ServantPtr>::iterator sit
    = _servants.find(current->getServantName());
ret = sit->second->dispatch(current, buffer);
```

XXImp::getSmallerOfTwoInt()

```
void ServantHandle::handle(stRecvData)
    handleTafProtocol(current)
```

```
int Handle::HandleImp()
handle(stRecvData)(virtual函数)
```

```
void BindAdapter::insertRecvQueue(vtRecvData)
{
    _rbuffer.push_front(vtRecvData);
    _handleGroup->monitor.notify();
}
```

```
_pBindAdapter->insertRecvQueue(vRecvData)
```

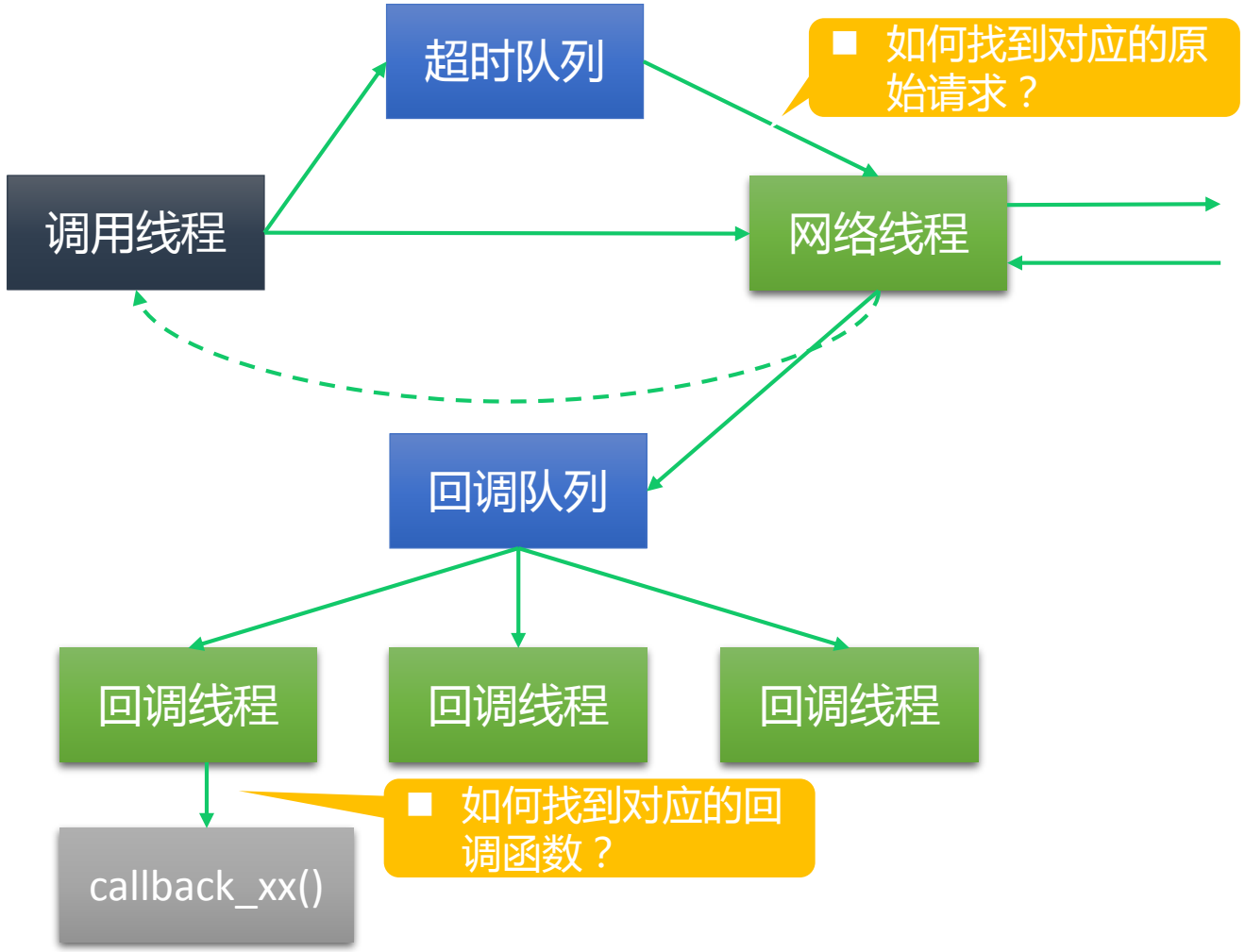
```
Void Connection::insertRecvQueue(vRecvData);
```

```
void TC_EpollServer::processNet(const epoll_event&ev)
    Connection*cPtr=getConnectionPtr(uid);
        cPtr->insertRecvQueue(vRecvData);
```

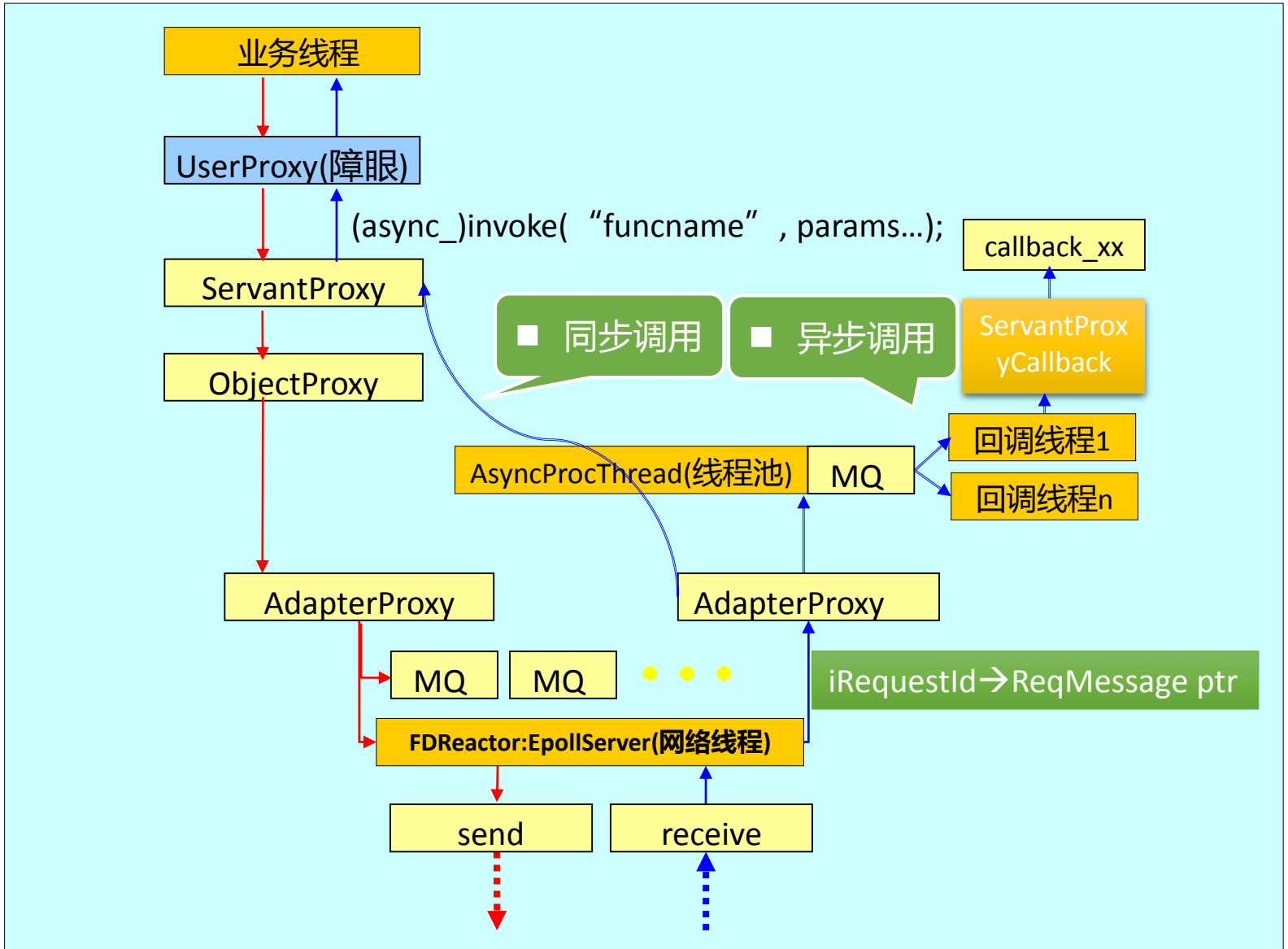
```
void TC_EpollServer::waitForShutdown(){
while(!_bTerminate){
    case ET_NET:
            processNet(ev);
  }}
```

回调问题解析

- 异步回来，如何进入callback_Test函数？



回调具体流程


业务线程
UserProxy(障眼)
(async_)invoke(“funcname” , params…);
callback_xx
ServantProxy
同步调用
异步调用
ServantProxyCallback
ObjectProxy
回调线程1
AsyncProcThread(线程池)
MQ
回调线程n
AdapterProxy
AdapterProxy
MQ
MQ
iRequestId→ReqMessage ptr
FDReactor:EpollServer(网络线程)
send
receive

回调关键代码

```
int XXCallback::onDispatch(taf::ReqMessagePtr msg)
msg->request.sFuncName
```

```
void AsyncProcThreadRunner::run()
req->callback->onDispatch(req);
```

```
void AsyncProcThread::put(ReqMessagePtr&req)
_queue.push_back(req);
```

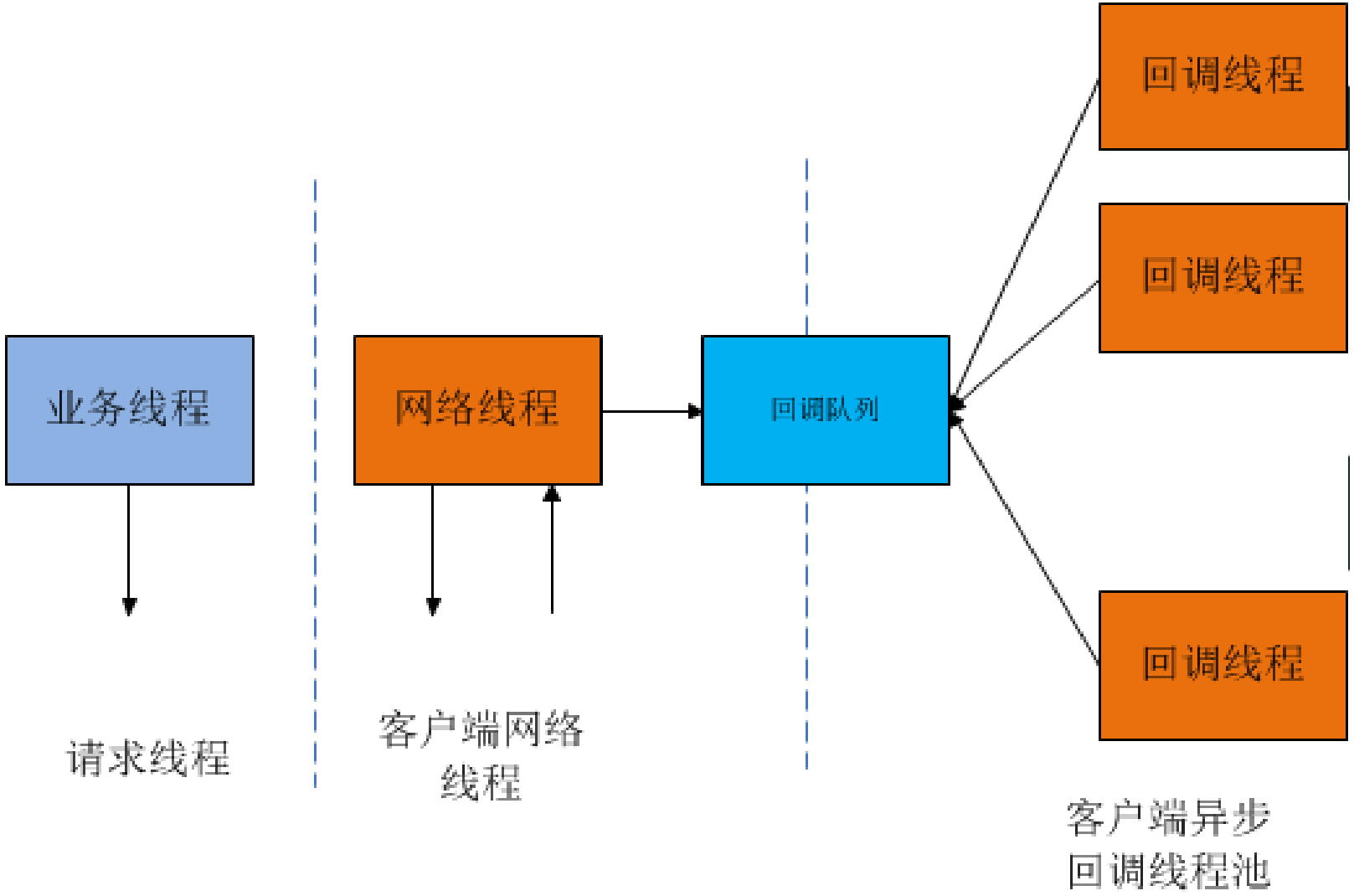
```
int AdapterPrxoy::finished(ResponsePacket&rsp)
ptr = _objectProxy->getTimeoutQueue()->get(rsp.iRequestId);
同步：ptr->proxy->finished(ptr);{req->monitor->notify();}
异步：_comm->asyncProcThread()->put(ptr);
```

```
void FDReactor::handleInputImp(adp)
adapter->finished(*lit);
```

```
void FDReactor::handle(adp,events)
handleInputImp(adapter);
```

```
void FDReactor::run()
{
    int num = _ep.wait(1000);
    for (int i = 0; i < num; ++i)
        {
            handle((AdapterProxy*)data, events);
        }
}
```

TAF线程与队列模型



3.1 TAF服务运维