logcfg -c test:\*

logcat -s test &

adb push out/target/product/arm64/system/usr/bin/study\_tool /data/

service process name

配置文件中 hostprocess = .mk文件的模块名

服务名

自动生成的代理service name = 自动生成stub 返回服务名 = 注册在工厂中的服务名 = 自己写的stub 返回服务名

Idl自动生成

Include下面：

一、Interface :

* 接口定义头文件：NIBroad.h :会有方法method 函数 例如add
* 类型定义头文件： NIBroadTypeCollection.h 没啥用
* 接口回调定义头文件：NIBroadReplier.h 会有方法和广播对应的回调函数replyAdd（）， notify Timeotify() 。

二 proxy代理

NIBroadProxy.h 用于会有add函数，会有onReceiveMessage函数

有广播，则有registerTimenotify() 和unregisterTimenotify()函数

三、stub

NIBroadStub.h

onAsncCall 代理发生的回调

isAcceptable 检查信息是否可接受

serviceName 服务名字

add 加方法

NIBroadStubReplier.h模板类

实现了以下三个方法：

replyAsyncCall 返回结果

notifyChanged 通知变化对于整个的

offerChanged 通知改变对于特定的

定义没有实现三个返回方法

replyAdd()

notify Timenotify（一个广播参数）

notify Timenotify（一个上下文参数，一个广播参数）

src 下面：

一、pack

自动生成的数据打包、解包代码

* 打包： NIBroadPacker.h，NIBroadPacker.cpp
* 消息类型定义 ：NIBroadMessage.h
* 解包 ：NIBroadUnpacker.h, NIBroadUnpacker.cpp

二、proto

自动生成的数据类型、参数打包定义的 proto 文件

* NIBroadProto.proto

三、自动生成的proxy端代码

* proxy： 实现类NIBroadProxy.cpp

实现了会有add onReceiveMessage registerTimenotify() unregisterTimenotify()

四、动生成的stub端代码

* NIBroadStub.cpp 实现
* onAsncCall 代理发生的回调
* isAcceptable 检查信息是否可接受
* serviceName 服务名字
* add 加方法没有被实现，留给自己写

\*\*\*\*\*以上方法中多了一个NEContext，用于返回信息，记录了哪个proxy，以及那一次调用

* NBroadStubReplier.cpp

实现三个返回方法

replyAdd()

notify Timenotify（一个广播参数）

notify Timenotify（一个上下文参数，一个广播参数）

要改的地方，代理中的serviceName（）方法返回的Service名字 ，（代理和stub中有此方法）

iAuto3 Service 开发

0. 自动生成的servicename方法中的名字都要改成你的服务名字。

1.构建一个NI stub继承自动生成的stub。实现（除去广播的）自定义方法

2 构建一个（装有自定义stub）的模板，自动生成的replier对象，

使用replier中的方法返回消息。

3构建一个自己的NI service，继承NEService，在service 中的onInit方法中，注册一个自定义的Stub。

4 构建一个NC工厂类继承NCServiceFactory类，在createService方法中注册自定义的service

5.构建一个自己的进程NC，继承NCServerProcess。

6.在broad/commom/fw\_services中的service.poc 和service.poc-emu中，分别注册服务名，其中hostprocess名字与服务的nutshell.mk中的执行文件名字要一样。

6.在main()函数中，new一个工厂类，将工厂注册进进程中，然后进程初始化、启动、循环、结束，完成整个过程。

iAuto3客户端开发

1.新建一个replier类，继承自动生成的replier接口类（\*\*\*自动生成的接口\*\*\*\*），

2 在对应方法中实现replier中的方法，完成结果的回复

3 如果是广播，使用对应的方法进行回复。

4新建一个app 进程，此进程的名字要和nutshell.mk中的名字一样，然后app初始化，新建。

5 通过NIIpcServiceHelper帮手查找有没有指定的的服务

6，新建一个代理，新建一个replier,

7 将代理和服务绑定，

8 将代理和replier绑定

9.1 如果是方法，调用代理中对应的方法

9.2 如果是广播，调用代理中的register方法

10. 睡眠一会。等待服务完成任务，才继续，否则代理就会销毁了

11将代理与服务和replier分别解绑（先解绑服务，后解绑代理）

12. 设置代理和replier分别为空。

13．进程结束。

广播：service端：

在stub中单独写一个方法，然后用replier中对应的方法返回，

客户端：利用代理中的方法（\*\*\*\*\*标红的两个方法为同一个方法）直接调用，返回结果为replier中继承对应的方法（）。  
```

\*\*\*\*\*\*\*\*\*\*\*\*\*replier方法分别对应，proxy和服务中的方法分别对应

（1）服务测：stub继承的是自动生成的stub，重写父类方法，参数要对应  
（2）服务测： replier使用自动生成的replier，参数要对应

(3) 代理侧：replier继承的是自动生成的replier 接口，参数要对应

注意：

1.在Service的下一级新建一个自己的service.  
在这一目录下，mkdir .git 否则无法编译

2.单独编译一个.mk  
//编译  
mm  
3//adb设备  
adb shell  
adb push  /out/target/product/arm64/system/usr/bin/study\_tool /data/

单独push 动态库

adb push  /out/target/product/arm64/system/usr/lib/libservice /data/lib

4．动态库.so默认路径为/data/lib，

5.有单独的动态库，修改后要重新push到/data/lib 目录下，否则不会更新。

6 svrmgr\_tool ls 为查看服务状态，3为启动成功

find . –name \*.mk –exec grep –Hrn TARGET\_OUT\_DATA {} \;