# SmartIOSservicePorxyBase.h/cpp文件

### 1 SmartPhoneservice代码结构分析

Src：

Include ---包含的头文件

Proxy ---代理，在编译时，封装成库，供外部其他模块调用该服务时使用

Stub ---接受proxy中来自外部的调用请求，并调用内部service逻辑处理该请求

----将sub中的广播发送到外部

Sub----广播作用，向其他模块服务发送广播。告知自身变化。

### 2 在SmartIosServiceProxyBase.h中定义了三个枚举，分别代表着三种不同类型的方法

|  |
| --- |
| enum Method\_Code  {  METHOD\_SMT\_GET\_DEV\_CONNECT\_INFO = SERVICE\_FIRST\_TRANSACTION,//在该位置初始了一个枚举中的第一数值。  METHOD\_SMT\_GET\_DEV\_CONNECT\_INFO\_ANDROID,  METHOD\_MAX  };  enum Command\_Code  {  COMMAND\_SMT\_LAUNCH\_SOURCE = METHOD\_MAX,  COMMAND\_SMT\_SET\_POP\_STATUS,  COMMAND\_SMT\_SET\_POP\_STATUS\_ANDROID,  COMMAND\_SMT\_GET\_DEVICE\_INFO,  COMMAND\_SMT\_GET\_DEVICE\_INFO\_ANDROID,  COMMAND\_SMT\_DEL\_SINGLE\_DEVICE,  COMMAND\_SMT\_DEL\_ALL\_DEVICES,  COMMAND\_SMT\_LAUNCH\_SOURCE\_ANDROID,  COMMAND\_MAX  };  enum Notify\_Code  {  NOTIFY\_SMT\_POP\_STATUS\_IND = COMMAND\_MAX,  NOTIFY\_SMT\_POP\_STATUS\_IND\_ANDROID,  NOTIFY\_SMT\_DEVICE\_INFO\_IND,  NOTIFY\_SMT\_DEVICE\_INFO\_IND\_ANDROID,  NOTIFY\_SMT\_DEL\_SINGLE\_DEVICE\_IND,  NOTIFY\_SMT\_DEL\_ALL\_DEVICE\_IND,  NOTIFY\_SMT\_DEV\_INFO\_CMP\_IND,  NOTIFY\_MAX  }; |

### 3 smt\_getDeviceInfo ()

获取设备信息方法

android::status\_t SmartIosServiceProxyBase::smt\_getDeviceInfo(const SmtDevReqType& reqType, const uint8\_t& index)

参数：

const SmtDevReqType& reqType ：获取sartphone的请求类型，All/Single

const uint8\_t& index：设备index，1~5为已记录设备，6~15为未记录设备  
 发送异步请求，通过传入的枚举参数COMMAND\_SMT\_GET\_DEVICE\_INFO来区分

|  |
| --- |
| android::status\_t SmartIosServiceProxyBase::smt\_getDeviceInfo(const SmtDevReqType& reqType, const uint8\_t& index) /\* \_\_0x101009\_DTFUNCID\_\_ \*/  {  android::Parcel \_data;//声明一个Parcel数据包  prepareAsyncData(\_data);//调用platform层中的ServiceProxyBase中的方法，传入数据包//参数，准备数据  \_data.writeInt32(static\_cast<SmtDevReqType>(reqType));//向数据包中写入请求类型 \_data.writeInt32(index);//向数据包写入设备index  return sendAsyncRequest(COMMAND\_SMT\_GET\_DEVICE\_INFO, \_data);//返回调用platform层的发送异步请求的方法，传入枚举command,和数据包\_data的结果  } |

### 4 smt\_getDevConnectInfo ()

获得设备的连接信息方法

Android::status\_t SmartIosServiceProxyBase::smt\_getDevConnectInfo(uint8\_t& connState, uint8\_t& index, uint8\_t& popStatus)

获取设备连接信息

参数：

int8\_t& connState---连接状态

uint8\_t& index----设备索引号

uint8\_t& popStatus-----排他状态值

|  |
| --- |
| android::status\_t SmartIosServiceProxyBase::smt\_getDevConnectInfo(uint8\_t& connState, uint8\_t& index, uint8\_t& popStatus) /\* \_\_0x10100D\_DTFUNCID\_\_ \*/  {  android::Parcel data, \_data;//定义两个数据包  int ret = sendSyncRequest(METHOD\_SMT\_GET\_DEV\_CONNECT\_INFO, \_data, &data);//调用底层的发送同步请求方法，返回值为-1表示错误  if (ret == BS\_NO\_ERROR) {// BS\_NO\_ERROR -1  connState = data.readInt32();  index = data.readInt32();  popStatus = data.readInt32();  DTLOG\_INFO(0x10100D, "smt\_getDevConnectInfo connState = %d index = %d popStatus = %d", connState, index, popStatus);  }  return ret;  } |

### 5 onConnected()

获取设备连接

|  |
| --- |
| void SmartIosServiceProxyBase::onConnected() /\* \_\_0x101003\_DTFUNCID\_\_ \*/  {  DTLOG\_INFO(0x101003, "SmartIosServiceProxyBase::onConnected");//打log  ServiceProxyBase::onConnected();//调用plantform层的onConnect()连接方法  m\_replier->onConnected();//调用ISmartIosServiceProxyReplier接口中的一个连接方法  在该接口中，该方法为空实现，在这里不调用为报错。  } |

### 6 void SmartIosServiceProxyBase::onDisconnected()

断开连接

|  |
| --- |
| void SmartIosServiceProxyBase::onDisconnected() /\* \_\_0x101004\_DTFUNCID\_\_ \*/  {  DTLOG\_INFO(0x101004, "SmartIosServiceProxyBase::onDisconnected");//打log  ServiceProxyBase::onDisconnected();//调用plantform层的disConfonnect()连接方法  m\_replier->onDisconnected();  } |

### 7smt\_launchSource(const uint8\_t& index, const SmtIosSource& source)

IOS设备Carplay/Weblink/iPod机能使用，启动资源

**参数：**

const uint8\_t& index---设备index

const SmtIosSource& source----要使用的机能Carplay/Weblink/iPod。

SmtIosSource 是定义的一个设备类型枚举

|  |
| --- |
| android::status\_t SmartIosServiceProxyBase::smt\_launchSource(const uint8\_t& index, const SmtIosSource& source) /\* \_\_0x101005\_DTFUNCID\_\_ \*/  {  DTLOG\_INFO(0x101005, "smt\_launchSource index = %d source = %d", index, source);//打log, 0x101005,该方法的ID  android::Parcel \_data;  prepareAsyncData(\_data);  \_data.writeInt32(index);  \_data.writeInt32(static\_cast<SmtIosSource>(source));  return sendAsyncRequest(COMMAND\_SMT\_LAUNCH\_SOURCE, \_data);//将数据包和命令序号作为参数，发送一个异步请求  } |

### 8 smt\_setPopStatus(const SmtSetType& setType, const uint8\_t& index, const SmtPopStatus& pop) /\* \_\_0x101007\_DTFUNCID\_\_ \*/

设置弹出状态

参数：

const SmtSetType& setType—设置显示方式。SmtSetType枚举类型，表示设置的样式是弹出还是菜单显示

const uint8\_t& index-----设备index

const SmtPopStatus& pop---- 弹出方式。SmtPopStatus枚举，表示弹出时是无效还是AUTOCARPLA，还是其他

|  |
| --- |
| android::status\_t SmartIosServiceProxyBase::smt\_setPopStatus(const SmtSetType& setType, const uint8\_t& index, const SmtPopStatus& pop) /\* \_\_0x101007\_DTFUNCID\_\_ \*/  {  DTLOG\_INFO(0x101007, "smt\_setPopStatus setType = %d index = %d pop = %d", setType, index, pop);  android::Parcel \_data;  prepareAsyncData(\_data);  \_data.writeInt32(static\_cast<SmtSetType>(setType));  \_data.writeInt32(index);  \_data.writeInt32(static\_cast<SmtPopStatus>(pop));  return sendAsyncRequest(COMMAND\_SMT\_SET\_POP\_STATUS, \_data);//发送异步请求  } |

### 9SmartIosServiceProxyBase::smt\_delSingleDevice(const uint8\_t& index) /\* \_\_0x10100B\_DTFUNCID\_\_ \*/

删除单个设备

参数：

const uint8\_t& index---设备ID

|  |
| --- |
| android::status\_t SmartIosServiceProxyBase::smt\_delSingleDevice(const uint8\_t& index) /\* \_\_0x10100B\_DTFUNCID\_\_ \*/  {  android::Parcel \_data;  prepareAsyncData(\_data);  \_data.writeInt32(index);  return sendAsyncRequest(COMMAND\_SMT\_DEL\_SINGLE\_DEVICE, \_data);  } |

### 10 SmartIosServiceProxyBase::smt\_delAllDevices()

删除所有设备

|  |
| --- |
| android::status\_t SmartIosServiceProxyBase::smt\_delAllDevices() /\* \_\_0x10100C\_DTFUNCID\_\_ \*/  {  android::Parcel \_data;  prepareAsyncData(\_data);  return sendAsyncRequest(COMMAND\_SMT\_DEL\_ALL\_DEVICES, \_data);  } |

## SmartIosServiceStubBase.h/cpp文件

## 1 onSyncRequest(unsigned int code, const android::Parcel &data, android::Parcel& \_data) /\* \_\_0x202003\_DTFUNCID\_\_ \*/

处理同步请求

参数：

unsigned int code---请求code

const android::Parcel &data--同步请求传过来的数据包1

android::Parcel& \_data---同步请求传过来的数据包2

|  |
| --- |
| int SmartIosServiceStubBase::onSyncRequest(unsigned int code, const android::Parcel &data, android::Parcel& \_data) /\* \_\_0x202003\_DTFUNCID\_\_ \*/  {  UNUSED(data);  switch (code) {//通过case处理不同code的请求  case METHOD\_SMT\_GET\_DEV\_CONNECT\_INFO://proxy中的getDevConnectInfo请求  {  uint8\_t connState;  uint8\_t index;  uint8\_t popStatus;//定义三个变量。  smt\_getDevConnectInfo(connState, index, popStatus);//通过三个变量调用Smt\_getDevConnectInfo()方法  \_data.writeInt32(connState);  \_data.writeInt32(index);  \_data.writeInt32(popStatus);  break;  }  case METHOD\_SMT\_GET\_DEV\_CONNECT\_INFO\_ANDROID:  {  uint8\_t connState;  uint8\_t index;  uint8\_t popStatus;  smt\_getDevConnectInfoAndroid(connState, index, popStatus);  \_data.writeInt32(connState);  \_data.writeInt32(index);  \_data.writeInt32(popStatus);  break;  }  default:  break;  } |

## ===========================================、

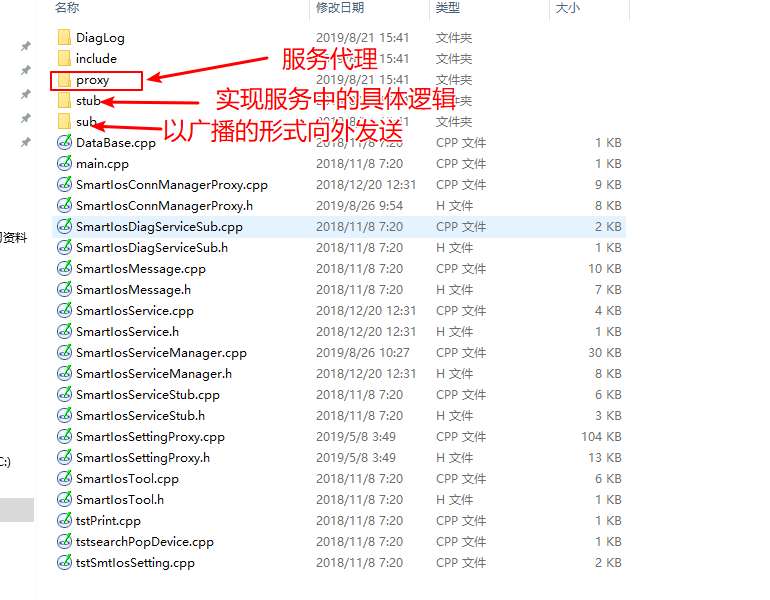
## Platform层ServiceProxyBase.h/cpp文件

### 1. sendSyncRequest（unsigned int code, const Parcel &data）发送同步请求

在platform层的baseService中定义

# Smartphone总结

代码结构



## Proxy-**Stub**模式分析

Stub 跟 Proxy 是一对，俗称“代理-桩”，一般用在远程方法调用。

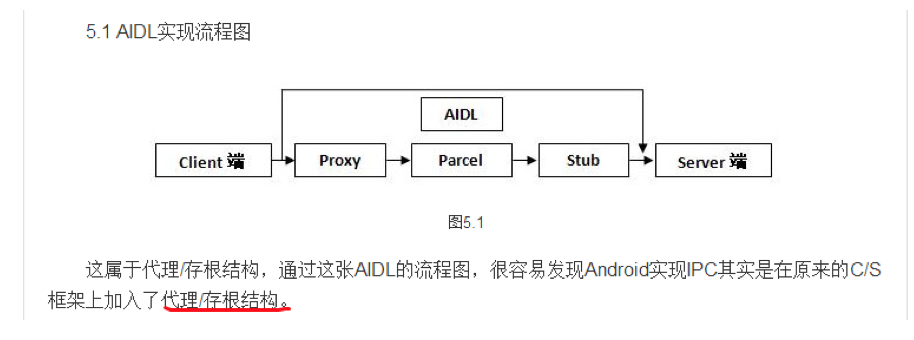
Proxy 的接口供客户端程序调用，然后它内部会把信息包装好，以某种方式（比如 RMI）传递给 Stub，而后者通过对应的接口作用于服务端系统，从而完成了“远程调用”。

一般不同进程间通信的时候都会用到这种模式。

应用场景：

android ipc方式aidl

AIDL属于Android的IPC机制，常用于跨进程通信，主要实现原理基于底层Binder机制。



## Service层的流程

在proxy层中定义了对外的接口，供其他service使用该服务。

在stub中实现具体的service功能，其实还是调用的serviceManager中的方法。

在manager中调用LoopThread类中的postMessage方法发送请求的message到消息队列中。

具体谁从消息队列中取走了消息，暂时不清楚

Proxy层

Replie类，应答类，当服务调用方需要请求的响应信息时，通过REplie类中的方法来实现service返回给调用方消息。该类在调用方实现。

ProxyBase类，服务的代理类，是smartPhone模块和其他service建立连接的一种方式

在proxyBase类中，该服务对外的结口，通过将请求的数据封装在Pacle数据包中，以发布同步或者异步请求的方式，通过sendAsyncRequest/sendSyncRequest方法，将该请求转发到stub，在stub中通过onAsyncResponse/ondSyncResponse方法，将proxy传过来的数据进行处理，处理的方式一般是通过调用serviceManager类中的方法实现。

在proxy中通过发送请求的方式，将数据传到stub中

|  |
| --- |
| android::status\_t SmartIosServiceProxyBase::smt\_getDeviceInfo(const SmtDevReqType& reqType, const uint8\_t& index) /\* \_\_0x101009\_DTFUNCID\_\_ \*/  {  android::Parcel \_data;  prepareAsyncData(\_data);  \_data.writeInt32(static\_cast<SmtDevReqType>(reqType));  \_data.writeInt32(index);  return sendAsyncRequest(COMMAND\_SMT\_GET\_DEVICE\_INFO, \_data);//发送一个异步请求  } |

在stub中，通过onAsyncRequest方法接受请求，根据请求状态值调用不同的case.

|  |
| --- |
| int SmartIosServiceStubBase::onAsyncRequest(SenderId &id, unsigned int code, const android::Parcel &data) /\* \_\_0x202004\_DTFUNCID\_\_ \*/  {  switch (code) {  case COMMAND\_SMT\_GET\_DEVICE\_INFO:  {  SmtDevReqType reqType = static\_cast<SmtDevReqType>(data.readInt32());  uint8\_t index = data.readInt32();  smt\_getDeviceInfo(id, reqType, index);  break;  }  ｝  } |

通过在serviceManager中调用 mThread.postMessage()的方法，实现将消息发送到消息队列中，实现不同进程之间的通信

|  |
| --- |
| android::status\_t SmartIosServiceManager::smt\_getDeviceInfo(SenderId id, const SmtDevReqType& reqType, const uint8\_t& index) /\* \_\_0x406013\_DTFUNCID\_\_ \*/  {  SmartIosMessage::GetDevInfoReq devInfoReq;  devInfoReq.id = id;  devInfoReq.index = index;  devInfoReq.reqType = reqType;  android::sp<SmartIosMessage>msg = new SmartIosMessage(devInfoReq);  if (msg != NULL) {  mThread.postMessage(reinterpret\_cast<Message\*>(msg.get()), 0);//将消息post到消息队列中，由该线程将此消息处理  }  return 0;  } |

问题：postMessage方法执行后，将消息添加到消息队列中后，消息最后怎么处理的不清楚。

有handle类专门处理消息对列中的消息。