# SmartIOSservicePorxyBase.h/cpp文件

### 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();  } |

### 8 smt\_setPopStatus(const SmtSetType& setType, const uint8\_t& index, const SmtPopStatus& pop)

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)

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);  } |

# =====================================

# 1 模块结构

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 主要接口

## 1 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

GetDeviceInfo方法中，通过传入的设备请求类型，经过调用，最终在SmtIosSettingProxy类的getDeviceInfo方法中进行逻辑处理。具体处理逻辑没整明白。

核心处理逻辑（smtIsoServiceSettingProxy）

|  |
| --- |
| bool SmartIosSettingProxy::getDeviceInfo(SmtDevReqType reqType, uint8\_t index, std::vector<IosDeviceName\*> \*mDevList, SmtDeviceType deviceType)  {  UNUSED(index);  char sql[SQL\_MAX\_LENGTH] = {0};  char \* errMsg = NULL;  char \*\*result = NULL;  int nRow = 0;  int nCol = 0;  enum SMT\_SQL\_RET resultRet;    if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  snprintf(sql,SQL\_MAX\_LENGTH,"select \* from popDevices order by priority");  }  else{  snprintf(sql,SQL\_MAX\_LENGTH,"select \* from popDevicesAndr order by priority");  }  resultRet = dbOperateRetJudgement(sqlite3\_get\_table(mdb,sql,&result,&nRow,&nCol,&errMsg));  if(SMT\_SQL\_RET\_SUCCESS == resultRet) {  SMTLOGD("get table success");  SMTLOGD("nrow %d,ncol %d",nRow,nCol);  int index = nCol;  int len = 0;  for(int i = 0; i < nRow; ++i) {  if(reqType == SMT\_IOS\_DEV\_ALL\_REQ) {  IosDeviceName\* device = new IosDeviceName;  if (NULL == device){  DTLOG\_PTRASSERT(device);  break;  }  else{  memset(device,0,sizeof(IosDeviceName));  SMTLOGD("seirnum %s,priority %s,popStatus %s,devname %s",result[index],result[index + 1],result[index + 2],result[index + 3]);  len = strlen(result[index]);  if(len > SMT\_IOS\_DEVICE\_SERIALNUM\_MAX) {  len = SMT\_IOS\_DEVICE\_SERIALNUM\_MAX;  }  memcpy(device->format.serinum, result[index], len);  device->format.serinum[SMT\_IOS\_DEVICE\_SERIALNUM\_MAX] = '\0';  device->format.priority = atoi(result[index + 1]);  device->format.popStatus = atoi(result[index + 2]);  len = strlen(result[index + 3]);  if(len > SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN) {  len = SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN;  }  memcpy(device->format.deviceName, result[index + 3], len);  mDevList->push\_back(device);  index += nCol;  }  }  else {  IosDeviceName\* device = new IosDeviceName;  if (NULL == device){  DTLOG\_PTRASSERT(device);  break;  }  else{  memset(device,0,sizeof(IosDeviceName));  len = strlen(result[index]);  if(len > SMT\_IOS\_DEVICE\_SERIALNUM\_MAX) {  len = SMT\_IOS\_DEVICE\_SERIALNUM\_MAX;  }  memcpy(device->format.serinum, result[index], len);  device->format.serinum[SMT\_IOS\_DEVICE\_SERIALNUM\_MAX] = '\0';  device->format.priority = atoi(result[index + 1]);  device->format.popStatus = atoi(result[index + 2]);  len = strlen(result[index + 3]);  if(len > SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN) {  len = SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN;  }  memcpy(device->format.deviceName, result[index + 3], len);  mDevList->push\_back(device);  break;  }  }  }  sqlite3\_free\_table(result);  }  else {  DTLOG\_ERR(0x408022, "get table faild");  SMTLOGD("get table faild,errMsg %s",errMsg);  cleanTablesWhenOpeErr(resultRet, SMT\_DB\_TABLE\_TYPE\_POP\_IOS);  return false;  }  if(mDevList->size() != 0){  return true;  }    DTLOG\_INFO(0x408022, "mDevList->size() == 0");  return false;  }  } /\* namespa |

涉及到smtIosService模块与Setting模块之间的项目调用，也涉及到了sqlite数据库的操作。

具体逻辑暂且不清晰。

## 2 smt\_launchSource()

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  } |

在smt\_launchSource方法中，接收调用方传过来的请求参数，通过proxy,stubBase,stub，ServiceManager，smtIosConnectManagerProxy,connectProxy等文件，将参数传到ConnectManagerService服务中，在该服务中完成资源的加载。

## 3 smt\_delAllDevices()

删除所有设备

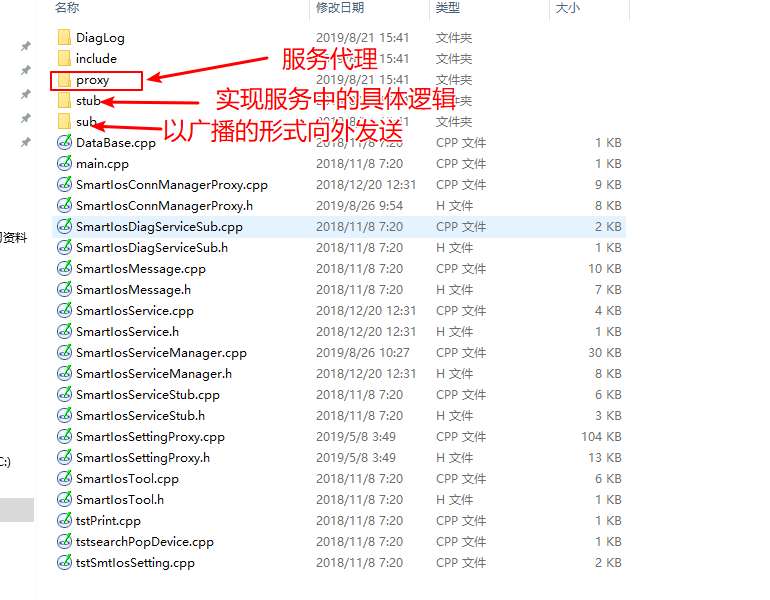
在SmtIosSettigProxy文件中，delAllDevices方法中调用了deleteTableFromDB()方法，执行具体删除所有设备的方法。

在该方法中，使用了sqlite事务。Begin …. commit

|  |
| --- |
| //在从数据库中执行删除表操作时，使用了事务。  void SmartIosSettingProxy::deleteTableFromDB(enum SMT\_DB\_TABLE\_TYPE tableType) /\* \_\_0x408008\_DTFUNCID\_\_ \*/{  char sql[SQL\_MAX\_LENGTH] = {0};    DTLOG\_INFO(0x408008, "SmartIosSettingProxy::deleteTableFromDB tableType = %d", tableType);  SMTLOGD("SmartIosSettingProxy::deleteTableFromDB");  if (NULL != mdb){  //开始事务  if (SMT\_SQL\_RET\_SUCCESS != dbOperateRetJudgement(sqlite3\_exec(mdb, "begin", NULL, NULL, NULL))){  DTLOG\_ERR(0x408008, "DB operate begin fail");  SMTLOGD("SmartIosSettingProxy::deleteTableFromDB, begin fail");  return;  }  //根据传入的参数判断删除哪张表  switch (tableType){  case SMT\_DB\_TABLE\_TYPE\_POP\_IOS:  snprintf(sql, SQL\_MAX\_LENGTH, "delete from popDevices");  break;  case SMT\_DB\_TABLE\_TYPE\_NO\_POP\_IOS:  snprintf(sql, SQL\_MAX\_LENGTH, "delete from noPopDevices");  break;  case SMT\_DB\_TABLE\_TYPE\_POP\_ANDROID:  snprintf(sql, SQL\_MAX\_LENGTH, "delete from popDevicesAndr");  break;  case SMT\_DB\_TABLE\_TYPE\_NO\_POP\_ANDROID:  snprintf(sql, SQL\_MAX\_LENGTH, "delete from noPopDevicesAndr");  break;  case SMT\_DB\_TABLE\_TYPE\_VERSION:  snprintf(sql, SQL\_MAX\_LENGTH, "delete from versionInfo");  break;  default:  return;  break;  }  //执行删除语句  if (SMT\_SQL\_RET\_SUCCESS != dbOperateRetJudgement(sqlite3\_exec(mdb, sql, NULL, NULL, NULL))){  DTLOG\_ERR(0x408008, "DB operate fail");  SMTLOGD("SmartIosSettingProxy::deleteTableFromDB, delete popDevices fail");  return;  }  //提交事务  if (SMT\_SQL\_RET\_SUCCESS != dbOperateRetJudgement(sqlite3\_exec(mdb, "commit", NULL, NULL, NULL))){  DTLOG\_ERR(0x408008, "DB operate commit fail");  SMTLOGD("SmartIosSettingProxy::deleteTableFromDB, commit fail");  return;  }  }  else{  DTLOG\_ERR(0x408008, "smartphone mdb is NULL");  SMTLOGD("SmartIosSettingProxy::deleteTableFromDB, smartphone mdb is NULL");  return;  }    return;  } |

# 3 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。

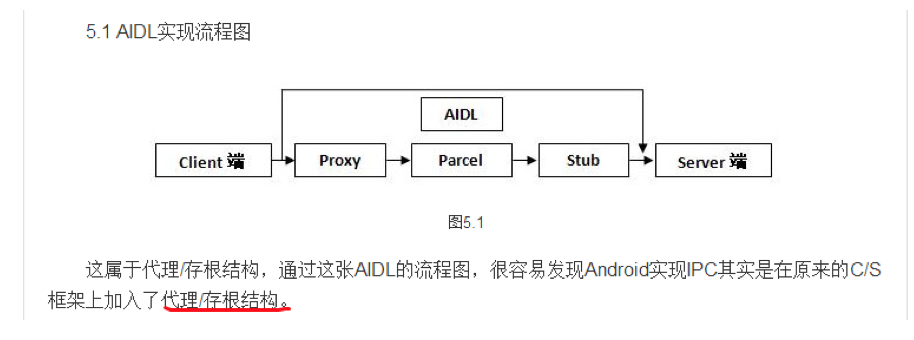
Proxy 的接口供客户端程序调用，然后它内部会把信息包装好，以某种方式（比如 RMI）传递给 Stub，而后者通过对应的接口作用于服务端系统，从而完成了“远程调用”。

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。