# 1 smtIosService介绍

该模块主要用于当设备通过数据线与车机相连时，确定设备类型、将设备信息存入数据库，同时，确定显示默认设备的类型.

# 2 核心方法实现

## 1 smt\_getDeviceInfo ()

获取设备信息。

设备信息存放在数据库中数据库中有五张表，具体来说应该是四张。

PopDevice 选择了资源的苹果设备

NopopDevice 没有选择资源的苹果设备

AndroidPopDevice 选择了资源的android设备

AndroidNoPopDevice 没有选择资源的android设备

实现逻辑

传入：设备请求类型 和index

enum SmtDevReqType

{

SMT\_IOS\_DEV\_ALL\_REQ,

SMT\_IOS\_DEV\_SINGLE\_REQ,

};

传出：SmtDevice设备信息。

封装成类。

reqType(),

total(),

index(),

code(),

pop(),

popAndroid(),

nameLen(),

name()

在SmtIosServiceManager中的onGetDevInfoReqHandler方法中实现了获取设备信息的具体逻辑。

|  |
| --- |
| void SmartIosServiceManager::SmartIosManagerHandler::onGetDevInfoReqHandler(const SmartIosMessage::GetDevInfoReq\* req, SmtDeviceType deviceType) {  //定义一个设备列表  std::vector<IosDeviceName\*> mDeviceList;  mid = req->id;  //在调用了settingService中的getDeviceInfo方法后，在该方法中为mDeviceList赋值。  if(SmartIosSettingProxy::getInstance()->getDeviceInfo(req->reqType, (uint8\_t)req->index, &mDeviceList, deviceType))  {  if(mDeviceList.size() == 0)  {  SMTLOGD("onGetDevInfoReqHandler no pop device");  }  else  {  uint8\_t count = mDeviceList.size();  #if 1  if(count > SMT\_IOS\_POP\_DEVICE\_MAX) {  SMTLOGD("count is error");  count = SMT\_IOS\_POP\_DEVICE\_MAX;  }  #endif  if(req->reqType == SMT\_IOS\_DEV\_SINGLE\_REQ)  {  android::sp<SmtDeviceInfo> devInfo = new SmtDeviceInfo();  devInfo->setIndex(mDeviceList[0]->format.priority);  devInfo->setReqType(SMT\_IOS\_DEV\_SINGLE\_REQ);  devInfo->setTotal( 0XFF);  devInfo->setCode( 0x00);  devInfo->setNameLen(0);  //打印deviceInfo  if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  mOwner->getServiceReplier()->smtDeviceInfoInd(req->id, devInfo);  }  else{  mOwner->getServiceReplier()->smtDeviceInfoIndAndroid(req->id, devInfo);  }  }  else  {  uint8\_t i = 0;  for(i = 0; i < count; i ++)  {  android::sp<SmtDeviceInfo> devInfo = new SmtDeviceInfo();  devInfo->setIndex(mDeviceList[i]->format.priority);  devInfo->setReqType(SMT\_IOS\_DEV\_ALL\_REQ);  devInfo->setTotal( count);  devInfo->setCode( 0x00);  char devName[SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN + 1];  int len = strlen((char\*)mDeviceList[i]->format.deviceName);  if(len > SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN) {  len = SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN;  }  memcpy(devName,mDeviceList[i]->format.deviceName,len);  devName[len] = '\0';  devInfo->setNameLen(strlen(devName));  android::String8 name(devName);  devInfo->setName(name);  if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  devInfo->setPop((SmtPopStatus)mDeviceList[i]->format.popStatus);  //调用smtDeviceNfoInd方法将devinfo反馈给上层的调用方。  mOwner->getServiceReplier()->smtDeviceInfoInd(req->id, devInfo);  }  else{  devInfo->setPopAndroid((SmtPopStatusAndroid)mDeviceList[i]->format.popStatus);  mOwner->getServiceReplier()->smtDeviceInfoIndAndroid(req->id, devInfo);  }  }  }  uint8\_t i = 0;  for(i = 0; i < count; i ++)  {  IosDeviceName\* dev =mDeviceList[i];  if(dev != NULL)  delete dev;  }  }  }  //no pop device  else {  SMTLOGD("line %d,no pop devices",\_\_LINE\_\_);  android::sp<SmtDeviceInfo> devinfo = new SmtDeviceInfo();  devinfo->setTotal(0);  devinfo->setReqType(SMT\_IOS\_DEV\_ALL\_REQ);  if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  mOwner->getServiceReplier()->smtDeviceInfoInd(req->id, devinfo);  }  else{  mOwner->getServiceReplier()->smtDeviceInfoIndAndroid(req->id, devinfo);  }  }  } |

在上述方法中从数据库中查询信息的逻辑在SmartIosSettingProxy::getDeviceInfo方法中实现。

内容太长，这里不再赘述。

### 总结

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。

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

该方法的核心处理在SmartIosServiceManager::SmartIosManagerHandler::onLaunchSourceReqHandler方法中。掉用connectManager中的launchIosSource方法。

|  |
| --- |
| void SmartIosServiceManager::SmartIosManagerHandler::onLaunchSourceReqHandler(const SmartIosMessage::LaunchSourceReq\* req) {  //将请求通过smartIOSService服务转到connectManagerService 中。  if(mOwner != NULL)  {  mOwner->getConMngProxy()->launchIosSourceReq(req->source);  }  } |

因为该方法是调用的connectManager中的方法，这里不做详细解析。

## 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;  } |

## 4 smt\_setPopStatus

设置设备排他性

这里涉及到了smtIosService模块和SettingService的交互。

在代理类smtIosServiceproxy类中开始调用该方法，经过层层的调用，最终在SmartIosSettingProxy类中的setPopStatusReq方法中处理设备排他性的具体逻辑

业务逻辑分析；

//set priority logic

//1. when insert a new device，if device list is not full，get the max priortiy ，if is empty，set 0，

//then the new device priority is max priority + 1

//2. if device list is full，find the priority equal 1 position，in the position insert new serinum，then priority set max，other device

// priorit all need minus one

//3. when delete a device，other device waht priority more than deleted devices need minus one，because a device deleted，

//the after it into the device list device need minus one

//pop up set is divided into 4 situation

//1 the dev is pop device and setting is pop value,only need update popStatus

//2 the dev is pop device and setting is no pop value,need pop device -> no pop device

//3 the dev is no pop device and setting is no pop value,only need update popStatus

//4 the dev is no pop device and setting is pop value,need no pop device -> pop device

代码如下：

|  |
| --- |
| /\*\*  \*  \* index explain:  \* for pop device,index equal priority,value range is [1,5]  \* for no pop device,index > priority,and index = priority + 5(pop device max num),value range is [6,15]  \*/  int SmartIosSettingProxy::setPopStatusReq(SmtSetType setType, uint8\_t index, SmtPopStatus setting, SmtDeviceType deviceType) /\* \_\_0x40801B\_DTFUNCID\_\_ \*/  {  std::vector<SmartIosIosDevInfo\*> mDeviceList;  SmartIosIosDevInfo\* mSelectDev = NULL;  std::vector<uint16\_t> mIndexList;  int newIndex = index;  enum SMT\_SQL\_RET resultRet;    SMTLOGD("SmartIosSettingProxy setPopStatusReq setType = %d,index %d, setting %d", setType,index, setting);  //pop set,index = index  if (NULL != mdb){  if(setType == SMT\_IOS\_POPUP\_SET) {  SMTLOGD("line %d,pop set",\_\_LINE\_\_);  SMTLOGD("use sqlite3 db");  char sql[SQL\_MAX\_LENGTH] = {0};  char \*errMsg = NULL;  IosDeviceName devname;    //no pop device  if((index >= SMT\_IOS\_NOPOP\_INDEX\_START) && (index <= SMT\_IOS\_NOPOP\_INDEX\_END)) {  if(isPopStatusValidHigh(setting)) {  if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  snprintf(sql,SQL\_MAX\_LENGTH,"select \* from noPopDevices where priority = '%d'",index-SMT\_IOS\_POP\_DEVICE\_MAX);  }  else{  snprintf(sql,SQL\_MAX\_LENGTH,"select \* from noPopDevicesAndr where priority = '%d'",index-SMT\_IOS\_POP\_DEVICE\_MAX);  }  resultRet = dbOperateRetJudgement(sqlite3\_exec(mdb,sql,sqlite3\_callback,(void\*)&devname,&errMsg));  if(SMT\_SQL\_RET\_SUCCESS == resultRet) {  SMTLOGD("resultCountOpe = %d",resultCountOpe);  if(resultCountOpe == 0) {  DTLOG\_ERR(0x40801B, "not the no pop device");  SMTLOGD("not the no pop device");  return index;  }  else {  SMTLOGD("line %d,serinum %s,priority %d,pop %d,name %s,nameisValid %d,index %d",\_\_LINE\_\_,devname.format.serinum,devname.format.priority,devname.format.popStatus,devname.format.deviceName,devname.format.nameIsValid,devname.format.index);  }  }  else {  DTLOG\_ERR(0x40801B, "select faild");  SMTLOGD("select faild,errMsg %s",errMsg);  cleanTablesWhenOpeErr(resultRet, SMT\_DB\_TABLE\_TYPE\_NO\_POP\_IOS);  return index;  }    bool ret;  //Delete nopopdevice  ret = deleteNoPopDevice(index, deviceType);    if (true != ret){  DTLOG\_ERR(0x40801B, "deleteNoPopDevice return false");  return index;  }    //update no pop devices priority  updateNoPopStatusDevicesPriority(index - SMT\_IOS\_POP\_DEVICE\_MAX, deviceType);  //insert the device into popDevices table  int maxPriority = getPopDeviceMaxPriority(deviceType);  if(maxPriority < SMT\_IOS\_POP\_PRIORITY\_END) {  newIndex = maxPriority + 1;  DTLOG\_INFO(0x40801B, "pop device is not full");  SMTLOGD("pop device is not full");    if(devname.format.nameIsValid == NAME\_IS\_VALID) {  SMTLOGD("the devname is valid");  ret = insertPopDevice(&devname, setting, maxPriority, deviceType);    if (true != ret){  DTLOG\_ERR(0x40801B, "insertPopDevice return false");  newIndex = index;  return index;  }  }//name is invalid  else {  SMTLOGD("the name is invalid");  ret = insertPopDeviceNoName(&devname, setting, maxPriority, deviceType);    if (true != ret){  DTLOG\_ERR(0x40801B, "insertPopDeviceNoName return false");  newIndex = index;  return index;  }  }  }//pop device is full,replace priority = 1 device  else {  SMTLOGD("the pop device is full");  newIndex = SMT\_IOS\_POP\_PRIORITY\_END;  if(devname.format.nameIsValid == NAME\_IS\_VALID) {  SMTLOGD("the devname is valid");  replacePopDevice((char\*)devname.format.serinum,setting,(char\*)devname.format.deviceName,NAME\_IS\_VALID,0, deviceType);  }//name is invlaid  else {  SMTLOGD("the devname is invalid");  IosDeviceName name;  int maxIndex = 0;  memset(&name,0,sizeof(name));  if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  snprintf(sql,SQL\_MAX\_LENGTH,"select nonameindex from popDevices where priority = (select max(priority) from popDevices where nameIsValid = '%d')",NAME\_IS\_INVALID);  }  else{  snprintf(sql,SQL\_MAX\_LENGTH,"select nonameindex from popDevicesAndr where priority = (select max(priority) from popDevicesAndr where nameIsValid = '%d')",NAME\_IS\_INVALID);  }  resultRet = dbOperateRetJudgement(sqlite3\_exec(mdb,sql,sqlite3\_callback,(void\*)&name,&errMsg));  if(SMT\_SQL\_RET\_SUCCESS == resultRet) {  SMTLOGD("resultCountOpe = %d",resultCountOpe);  maxIndex = name.format.priority % SMT\_DEVICE\_NONAMEINDEX\_MAX;  SMTLOGD("select nameIsInvalid max index success,maxindex %d",maxIndex);  char invalidname[SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN] = {0};  if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  snprintf(invalidname,SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN,"%s%d","iPhone",maxIndex + 1);  }  else{  snprintf(invalidname,SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN,"%s%d","Android Phone",maxIndex + 1);  }  SMTLOGD("invalidname is %s",invalidname);  replacePopDevice((char\*)devname.format.serinum,setting,invalidname,NAME\_IS\_INVALID,maxIndex + 1, deviceType);  }  else {  DTLOG\_ERR(0x40801B, "sqlite3\_exec find the max noname index fail");  SMTLOGD("sqlite3\_exec find the max noname index fail");  cleanTablesWhenOpeErr(resultRet, SMT\_DB\_TABLE\_TYPE\_POP\_IOS);  return index;  }    }  }  }  else {  #if 0 //Unused in 18HPMDA  snprintf(sql,SQL\_MAX\_LENGTH,"update noPopDevices set popStatus = '%d' where priority = '%d'",setting,index-SMT\_IOS\_POP\_DEVICE\_MAX);  if(SQLITE\_OK == sqlite3\_exec(mdb,sql,NULL,NULL,&errMsg)) {  SMTLOGD("set nopopStatus %d success",setting);  }  else {  SMTLOGD("set nopopStatus %d faild,errMsg %s",setting,errMsg);  }  #endif  }  }  //the device is pop device  else {  if(isPopStatusValidHigh(setting)) {  if (SMT\_SQL\_RET\_SUCCESS != dbOperateRetJudgement(sqlite3\_exec(mdb, "begin", NULL, NULL, NULL))){  DTLOG\_ERR(0x40801B, "DB operate begin fail");  SMTLOGD("SmartIosSettingProxy::setPopStatusReq, begin fail");  return index;  }    if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  snprintf(sql,SQL\_MAX\_LENGTH,"update popDevices set popStatus = '%d' where priority = '%d'",setting,index);  }  else{  snprintf(sql,SQL\_MAX\_LENGTH,"update popDevicesAndr set popStatus = '%d' where priority = '%d'",setting,index);  }  resultRet = dbOperateRetJudgement(sqlite3\_exec(mdb,sql,NULL,NULL,&errMsg));  if(SMT\_SQL\_RET\_SUCCESS == resultRet) {  SMTLOGD("update pop value success,index %d,popValue %d",index,setting);  }  else {  DTLOG\_ERR(0x40801B, "update pop value faild");  SMTLOGD("update pop value faild,errMsg %s",errMsg);  if (SMT\_SQL\_RET\_SUCCESS != dbOperateRetJudgement(sqlite3\_exec(mdb, "rollback", NULL, NULL, NULL))){  DTLOG\_ERR(0x40801B, "DB operate rollback fail");  SMTLOGD("SmartIosSettingProxy::setPopStatusReq, rollback fail");  }  cleanTablesWhenOpeErr(resultRet, SMT\_DB\_TABLE\_TYPE\_POP\_IOS);  return index;  }    if (SMT\_SQL\_RET\_SUCCESS != dbOperateRetJudgement(sqlite3\_exec(mdb, "commit", NULL, NULL, NULL))){  DTLOG\_ERR(0x40801B, "DB operate commit fail");  SMTLOGD("SmartIosSettingProxy::setPopStatusReq, commit fail");  return index;  }  }  //pop device -> no pop device  //1 select pop device info,delete pop device ,update other pop device priority  //2 insert the device into no pop device table  else {  #if 0//Unused in 18HPMDA  snprintf(sql,SQL\_MAX\_LENGTH,"begin");  if(SQLITE\_OK == sqlite3\_exec(mdb,sql,NULL,NULL,&errMsg)) {  SMTLOGD("begin success");  }  else {  SMTLOGD("begin faild");  return newIndex;  }  snprintf(sql,SQL\_MAX\_LENGTH,"select \* from popDevices where priority = '%d'",index);  if(SQLITE\_OK == sqlite3\_exec(mdb,sql,sqlite3\_callback,&devname,&errMsg)) {  if(resultCountOpe != 0) {  SMTLOGD("this is pop device");  resultCountOpe = 0;  }  else {  SMTLOGD("is not pop device");  return newIndex;  }  }  else {  SMTLOGD("select faild,errMsg %s",errMsg);  return newIndex;  }  snprintf(sql,SQL\_MAX\_LENGTH,"delete from popDevices where priority = '%d'",index);  if(SQLITE\_OK == sqlite3\_exec(mdb,sql,NULL,NULL,&errMsg)) {  SMTLOGD("delete pop device success");  }  else {  SMTLOGD("delete faild");  }  updatePopStatusDevicesPriority(index);  int maxNoPopIndex = getNoPopStatusDeviceMaxPriority();  char \*sql2 = NULL;  if(maxNoPopIndex < SMT\_IOS\_NOPOP\_DEVICE\_MAX) {  newIndex = maxNoPopIndex + 1 + SMT\_IOS\_POP\_DEVICE\_MAX;  SMTLOGD("no pop device is not full");  sql2 = sqlite3\_mprintf("insert into noPopDevices values('%q','%d','%d','%q','%d','%d')",devname.format.serinum,maxNoPopIndex + 1,setting,devname.format.deviceName,devname.format.nameIsValid,devname.format.index);  if(SQLITE\_OK == sqlite3\_exec(mdb,sql2,NULL,NULL,&errMsg)) {  SMTLOGD("insert into no pop device success");  }  else {  SMTLOGD("insert into no pop device faild");  return newIndex;  }  }  else {  SMTLOGD("no pop device is full");  newIndex = SMT\_IOS\_NOPOP\_PRIORITY\_END + SMT\_IOS\_POP\_DEVICE\_MAX;  uint8\_t mIndex = 0;  replaceNoPopDevice((char\*)devname.format.serinum,setting,(char\*)devname.format.deviceName,devname.format.nameIsValid,devname.format.index,mIndex);  }  #endif  }  }  }  //manu set,index = priority  else {  SMTLOGD("line %d,manu set\n",\_\_LINE\_\_);  char sql[SQL\_MAX\_LENGTH] = {0};  char \*errMsg = NULL;    if (SMT\_SQL\_RET\_SUCCESS != dbOperateRetJudgement(sqlite3\_exec(mdb, "begin", NULL, NULL, NULL))){  DTLOG\_ERR(0x40801B, "DB operate begin fail");  SMTLOGD("SmartIosSettingProxy::setPopStatusReq, begin fail");  return index;  }    if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  snprintf(sql,SQL\_MAX\_LENGTH,"update popDevices set popStatus = '%d' where priority = '%d'",setting,index);  }  else{  snprintf(sql,SQL\_MAX\_LENGTH,"update popDevicesAndr set popStatus = '%d' where priority = '%d'",setting,index);  }  resultRet = dbOperateRetJudgement(sqlite3\_exec(mdb,sql,NULL,NULL,&errMsg));  if(SMT\_SQL\_RET\_SUCCESS == resultRet) {  SMTLOGD("update success");  }  else {  DTLOG\_ERR(0x40801B, "update faild");  SMTLOGD("update faild,errMsg %s",errMsg);  if (SMT\_SQL\_RET\_SUCCESS != dbOperateRetJudgement(sqlite3\_exec(mdb, "rollback", NULL, NULL, NULL))){  DTLOG\_ERR(0x40801B, "DB operate rollback fail");  SMTLOGD("SmartIosSettingProxy::setPopStatusReq, rollback fail");  }  cleanTablesWhenOpeErr(resultRet, SMT\_DB\_TABLE\_TYPE\_POP\_IOS);  return index;  }    if (SMT\_SQL\_RET\_SUCCESS != dbOperateRetJudgement(sqlite3\_exec(mdb, "commit", NULL, NULL, NULL))){  DTLOG\_ERR(0x40801B, "DB operate commit fail");  SMTLOGD("SmartIosSettingProxy::setPopStatusReq, commit fail");  return index;  }  }  }  else {  DTLOG\_ERR(0x40801B, "smartphone mdb is NULL");  SMTLOGD("smartphone mdb is NULL");  //reopen DB  initSqlite3DB();  }  return newIndex;  } |

## 5 smt\_getDevConnectInfo

获取设备连接信息。

该方法，通过参数，将smtIosService中的值传递给上层调用方。

核心代码如下

其中，mIndex，mPopStatus，mConnState作为全局变量，在其他方法中被赋值

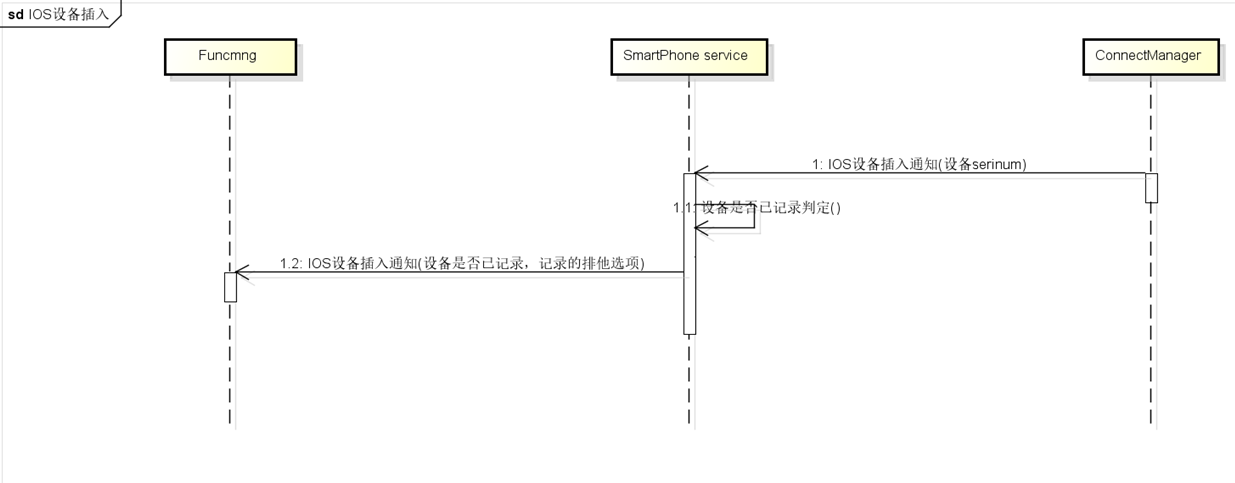
|  |
| --- |
| android::status\_t SmartIosServiceManager::getDevConnectInfo(uint8\_t& connState,uint8\_t& index,uint8\_t& popStatus) {  SMTLOGD("SmartIosServiceManager::getDevConnectInfo");  connState = mConnState;  index = mIndex;  popStatus = mPopStatus;  return 0;  } |

# 3 核心事件

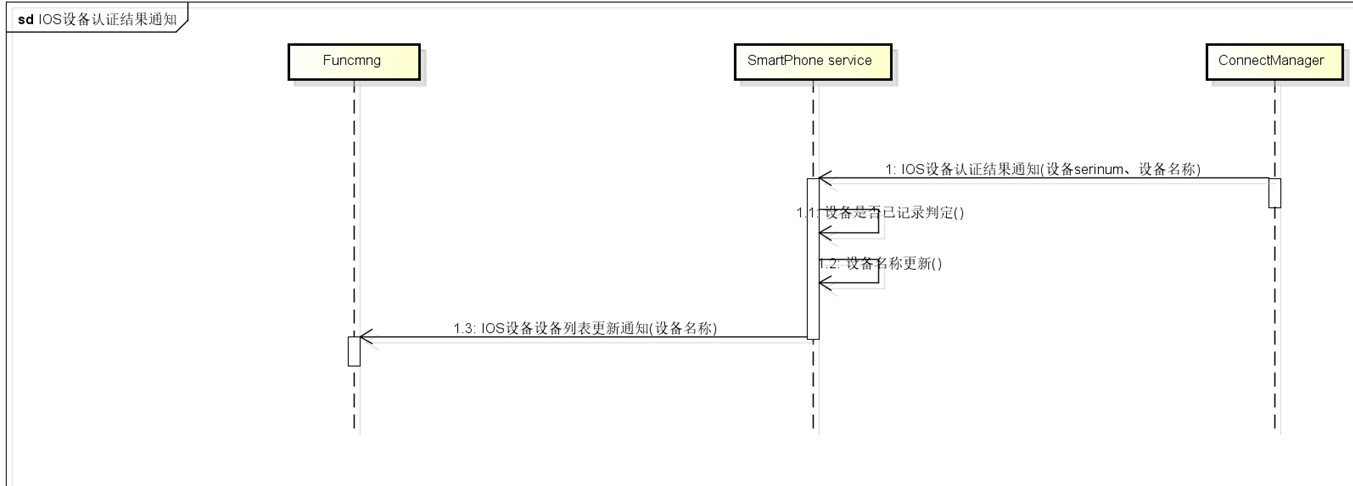
## 1 ios设备插入

### 时序

设备插入

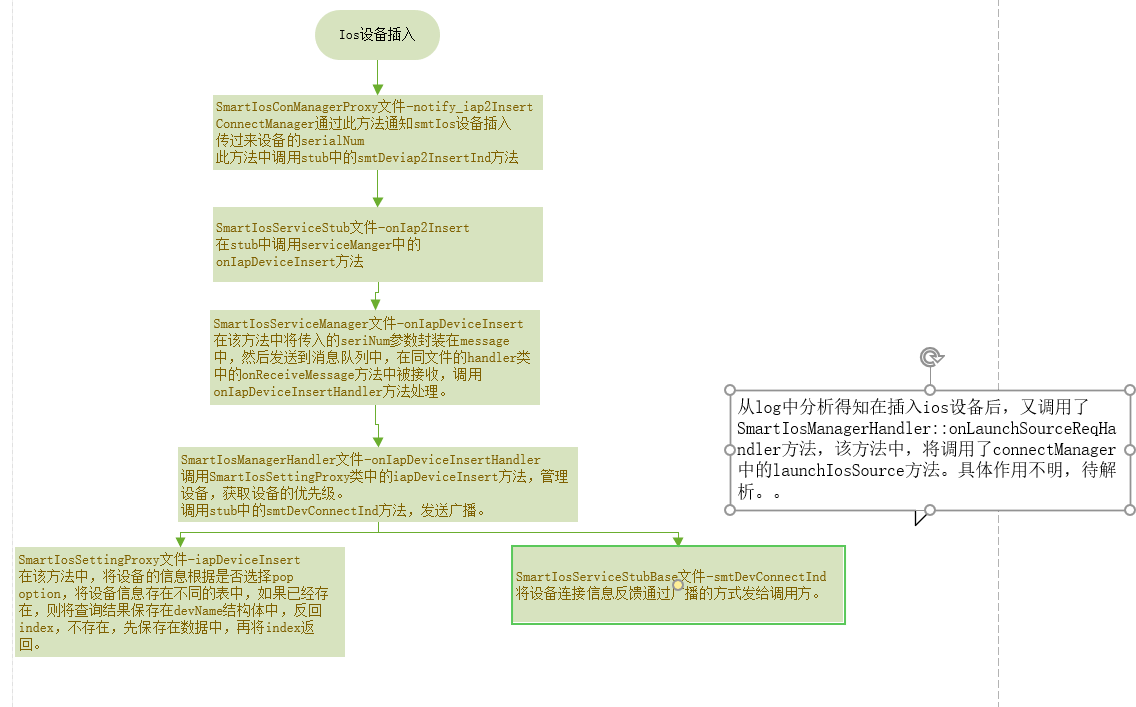


Iap2认证



### 执行流程

Ios设备插入流程图



Iap2认证流程略

### 涉及模块

connectManager

smtIosService

settingService

functionManager

### 实现

核心逻辑

当设备插入时，SmartIosManagerHandler文件-onIapDeviceInsertHandler方法逻辑具体逻辑，在方法中，调用SmartIosSettingProxy类中的iapDeviceInsert方法，管理设备，获取设备的优先级。调用stub中的smtDevConnectInd方法，发送广播。

|  |
| --- |
| void SmartIosServiceManager::SmartIosManagerHandler::onIapDeviceInsertHandler(const SmartIosMessage::IapDevInsertInd\* ind) {  uint8\_t popStatus;  //调用settingService中的方法，获取设备的优先权  uint8\_t index = SmartIosSettingProxy::getInstance()->iapDeviceInsert(ind->serialNum, &popStatus, SMT\_DEVICE\_TYPE\_IOS);  //新创建连接创建类  android::sp<SmtConState> con = new SmtConState;  if(index != SMT\_IOS\_DEVICE\_INDEX\_INVALID)  {  con->setIndex(index);  //mIndex：设备的优先级  mOwner->mIndex = index;  con->setPop(static\_cast<SmtPopStatus>(popStatus));  mOwner->mPopStatus = popStatus;  uint8\_t cState = SMT\_CONNECTED;  if(!(ind->isCarplaySupp)) {  cState |= (1 << 7);  }  mOwner->mConnState = cState;  con->setCState(SMT\_CONNECTED);  con->setIsCarplaySupp(ind->isCarplaySupp);  //发广播  mOwner->getServiceStub()->smtDevConnectInd(con);  }  } |

在smtIosServiceSettingproxy文件中的iapDeviceInsert方法中，通过connectManager传递过来的seriNum，在数据库的pop device和noPopDevice两个表中中进行设备查询，如果存在，则返回index，不存在则先保存在数据库中，再返回index。

|  |
| --- |
| 代码太多，不再粘贴。。。。 |

在smtIosServiceStub中发送广播的代码再这里不再演示。

在将设备的信息存在数据库中的后，通过log分析发现在某一层又调用了connectManage的iap2AuthInfo方法，这个方法具体干啥的，不太清除（个人理解，应该是苹果手机iap2协议认证相关。）

在connectManager层完成认证后，又通过notify\_iap2AuthInfo方法，通知给smtIosService.在smtIosService中经过connectManagerProxy、smtServiceStub、smtServiceManager文件中 各个方法的层层调用，最后在SmartIosServiceManager::SmartIosManagerHandler::onDevAutoInfoHandler方法中实现了具体逻辑。connectManager中传递给上层设备的seriNum和devName这两个参数，在该方法中，调用SmartIosSettingProxy类中的 setDeviceName方法，将传递上来的deviceName更新到数据库中。又调用了同类中的onGetDevInfoReqHandler方法，将根据seriNum从数据库中查询出来的设备详细信息传递给上层functionManager。

|  |
| --- |
| void SmartIosServiceManager::SmartIosManagerHandler::onDevAutoInfoHandler(const SmartIosMessage::devAutoInfo\* devinfo, SmtDeviceType deviceType){    //调用setting Service中setDeviceName方法，在settingService中设置deviceName  SmartIosSettingProxy::getInstance()->setDeviceName(devinfo->serinum, devinfo->devName, deviceType);  SmartIosMessage::GetDevInfoReq devInfoReq;  devInfoReq.id = mid;  devInfoReq.index = 0;  devInfoReq.reqType = SMT\_IOS\_DEV\_ALL\_REQ;  //调用onGetDevInfoReqHandler方法，逻辑转到getDeviceInfo的处理中，将获取到的device信息反馈给上层调用方。  onGetDevInfoReqHandler(&devInfoReq, deviceType);  } |

SmartIosSettingProxy类中的setDeviceName方法的具体代码这里不再列举。

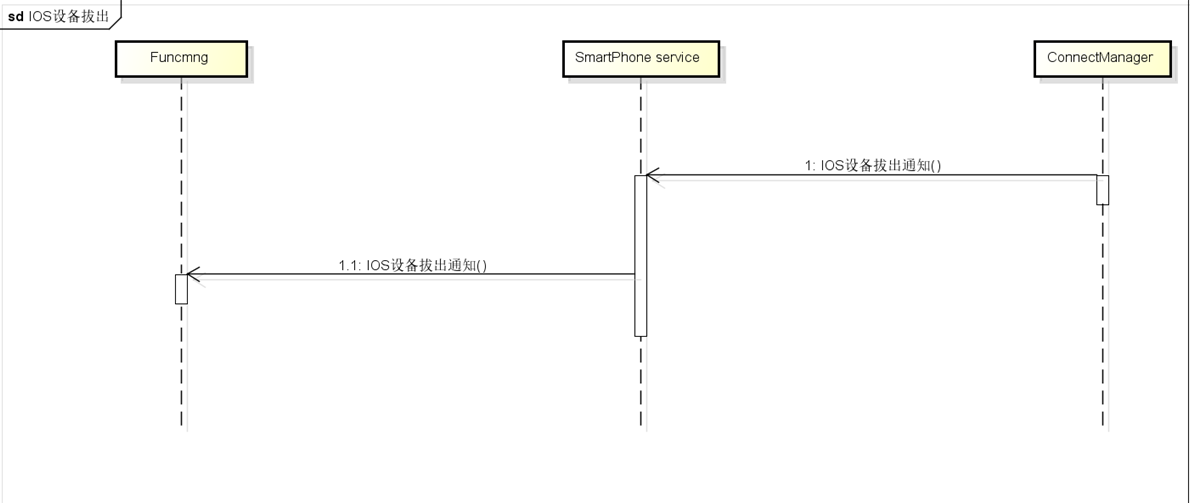
onGetDevInfoReqHandler方法代码

|  |
| --- |
| void SmartIosServiceManager::SmartIosManagerHandler::onGetDevInfoReqHandler(const SmartIosMessage::GetDevInfoReq\* req, SmtDeviceType deviceType) {  //定义一个设备列表  std::vector<IosDeviceName\*> mDeviceList;  SMTLOGD("get dev info,req->id %d",req->id);  mid = req->id;  //在调用了settingService中的getDeviceInfo方法后，在该方法中为mDeviceList赋值。  if(SmartIosSettingProxy::getInstance()->getDeviceInfo(req->reqType, (uint8\_t)req->index, &mDeviceList, deviceType))  {  if(mDeviceList.size() == 0)  {  SMTLOGD("onGetDevInfoReqHandler no pop device");  }  else  {  uint8\_t count = mDeviceList.size();  #if 1  if(count > SMT\_IOS\_POP\_DEVICE\_MAX) {  SMTLOGD("count is error");  count = SMT\_IOS\_POP\_DEVICE\_MAX;  }  #endif  if(req->reqType == SMT\_IOS\_DEV\_SINGLE\_REQ)  {  android::sp<SmtDeviceInfo> devInfo = new SmtDeviceInfo();  devInfo->setIndex(mDeviceList[0]->format.priority);  devInfo->setReqType(SMT\_IOS\_DEV\_SINGLE\_REQ);  devInfo->setTotal( 0XFF);  devInfo->setCode( 0x00);  devInfo->setNameLen(0);  if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  mOwner->getServiceReplier()->smtDeviceInfoInd(req->id, devInfo);  }  else{  mOwner->getServiceReplier()->smtDeviceInfoIndAndroid(req->id, devInfo);  }  }  else  {  uint8\_t i = 0;  for(i = 0; i < count; i ++)  {  android::sp<SmtDeviceInfo> devInfo = new SmtDeviceInfo();    devInfo->setIndex(mDeviceList[i]->format.priority);  devInfo->setReqType(SMT\_IOS\_DEV\_ALL\_REQ);  devInfo->setTotal( count);  devInfo->setCode( 0x00);  char devName[SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN + 1];  int len = strlen((char\*)mDeviceList[i]->format.deviceName);  if(len > SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN) {  len = SMT\_IOS\_DEVICE\_NAME\_MAX\_LEN;  }  memcpy(devName,mDeviceList[i]->format.deviceName,len);  devName[len] = '\0';  devInfo->setNameLen(strlen(devName));  android::String8 name(devName);  devInfo->setName(name);  if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  devInfo->setPop((SmtPopStatus)mDeviceList[i]->format.popStatus);  //调用smtDeviceNfoInd方法将devinfo反馈给上层的调用方。  mOwner->getServiceReplier()->smtDeviceInfoInd(req->id, devInfo);  }  else{  devInfo->setPopAndroid((SmtPopStatusAndroid)mDeviceList[i]->format.popStatus);  mOwner->getServiceReplier()->smtDeviceInfoIndAndroid(req->id, devInfo);  }  }  }  uint8\_t i = 0;  for(i = 0; i < count; i ++)  {  IosDeviceName\* dev =mDeviceList[i];  if(dev != NULL)  delete dev;  }  }  }  //no pop device  else {  android::sp<SmtDeviceInfo> devinfo = new SmtDeviceInfo();  devinfo->setTotal(0);  devinfo->setReqType(SMT\_IOS\_DEV\_ALL\_REQ);  if (SMT\_DEVICE\_TYPE\_IOS == deviceType){  mOwner->getServiceReplier()->smtDeviceInfoInd(req->id, devinfo);  }  else{  mOwner->getServiceReplier()->smtDeviceInfoIndAndroid(req->id, devinfo);  }  }  } |

### 总结

## 2 ios设备拔出

### 时序



### 执行流程

流程过于简单，不再画图

### 涉及模块

ConnectManager

SmtIosService

FunctionManager

### 实现

设备拔出后，connectManager通过回调函数notify\_iap2Extract通知给smtService,经过SmartIosConnManagerProxy、smtIosServiceStub、SmtIosServiceManager文件中的各个相关方法调用，最后在SmartIosServiceManager::SmartIosManagerHandler::onIapExactHandler方法中实现逻辑。

主要是将连接状态恢复为停止时的值，在stub中通过广播的方式发送给监听方。同时停止timer。

|  |
| --- |
| void SmartIosServiceManager::SmartIosManagerHandler::onIapExactHandler(void) {  android::sp<SmtConState> con = new SmtConState ;  con->setIndex (0xFF);  con->setPop(SMT\_POP\_INVALID);  con->setCState(SMT\_DISCONNECT);  //发广播  mOwner->getServiceStub()->smtDevConnectInd(con);  mOwner->mLaunchState = LAUNCH\_IDLE;  if(mOwner->mTimer.isActive()) {  SMTLOGD("---------------------------------stop timer");  mOwner->mTimer.stop();  }  } |

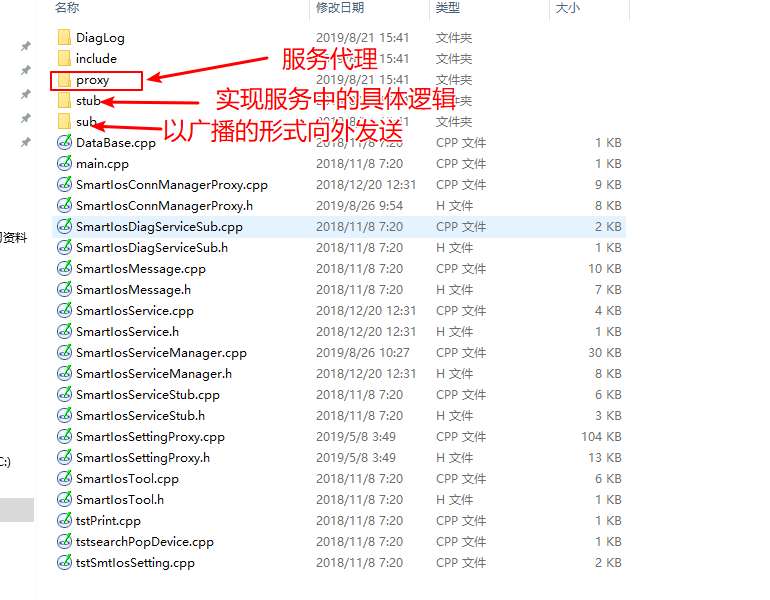
Stub中的发送连接信息的广播方法smtDevConnectInd

|  |
| --- |
| android::status\_t SmartIosServiceStubBase::smtDevConnectInd(const SmtConState\_Sp& conState) {  android::Parcel \_data;  \_data.writeCString("SmartIosService");  \_data.writeCString("multicast");  conState->writeToParcel(\_data);  return m\_owner->sendMulticast("multicast", MULTICAST\_SMT\_DEV\_CONNECT\_IND, \_data);  } |

### 总结

# 4 Smartphone架构

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。

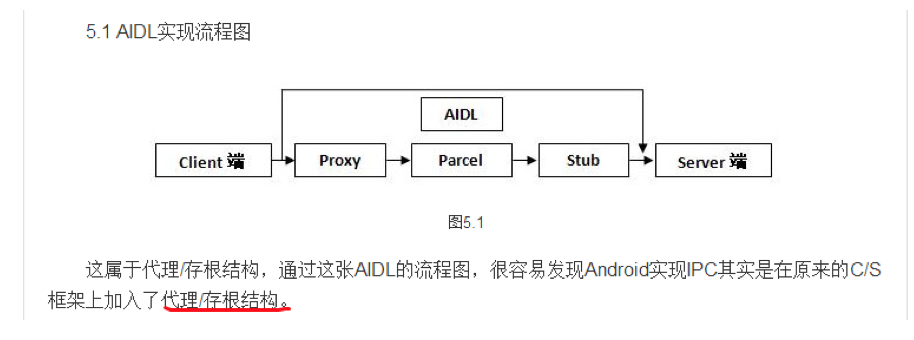
Proxy 的接口供客户端程序调用，然后它内部会把信息包装好，以某种方式（比如 RMI）传递给 Stub，而后者通过对应的接口作用于服务端系统，从而完成了“远程调用”。

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。

# 模版

### 时序

### 执行流程

### 涉及模块

### 实现

### 总结