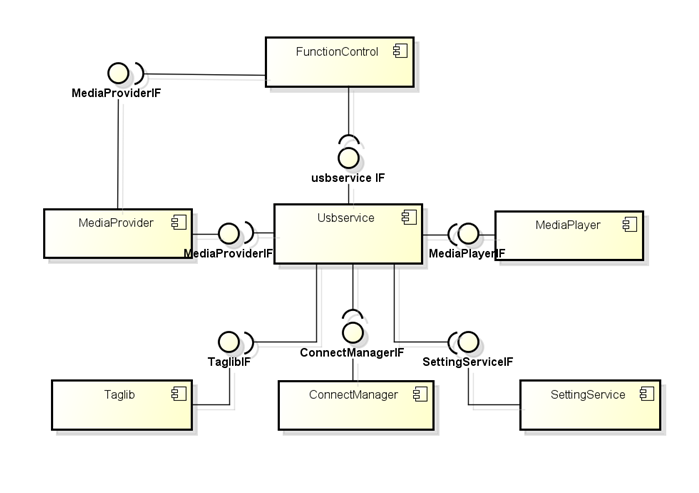
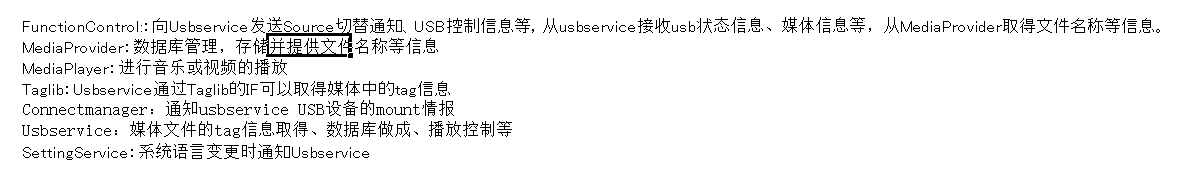
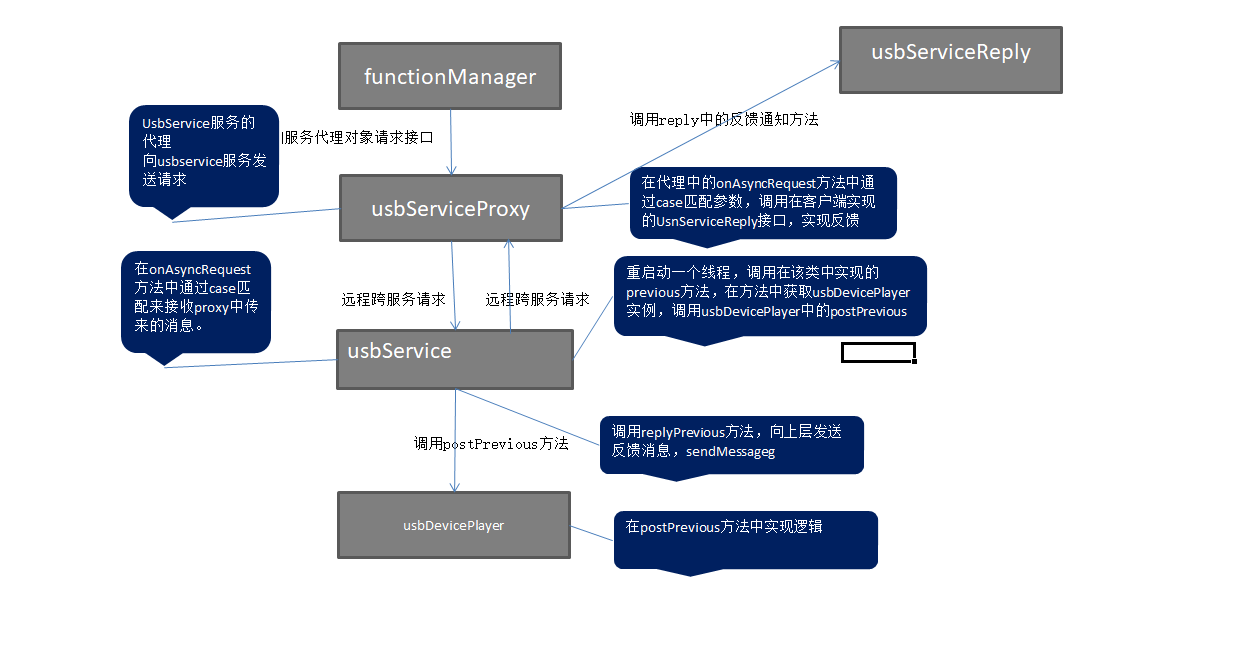
## 1 Usb与各个模块之间的调用关系





在usbservice中，上层与functionManager交互，下层与MediaService服务、DevicePlayerService服务、ConnectService服务交互。



## 2 Usbservice与各个模块交互的方式

### Usbservice与functionManager模块交互

向usbservice发送数据，和接受数据

在usbservice中，通过proxy与functionManager交互。

Proxy作为usbservice的代理接口，在proxy中的方法中sendMessage，该message会被GMUsbserviceBase中的onAsyncResponse方法中的case 接收。

而在UsbserviceBase中从serviceBasea中继承过来的sendAsyncRequest方法发送的消息，会被proxy中的onAsyncResponse方法中的case接收。

在usbService调用底层的逻辑处理请求前会将replyer应答反馈给上层。具体是，在Usbservice中发送sendMessage请求，在代理的onAsyncResponse方法中，以case匹配形式对应方法，实现将结果反馈给functionManager层。

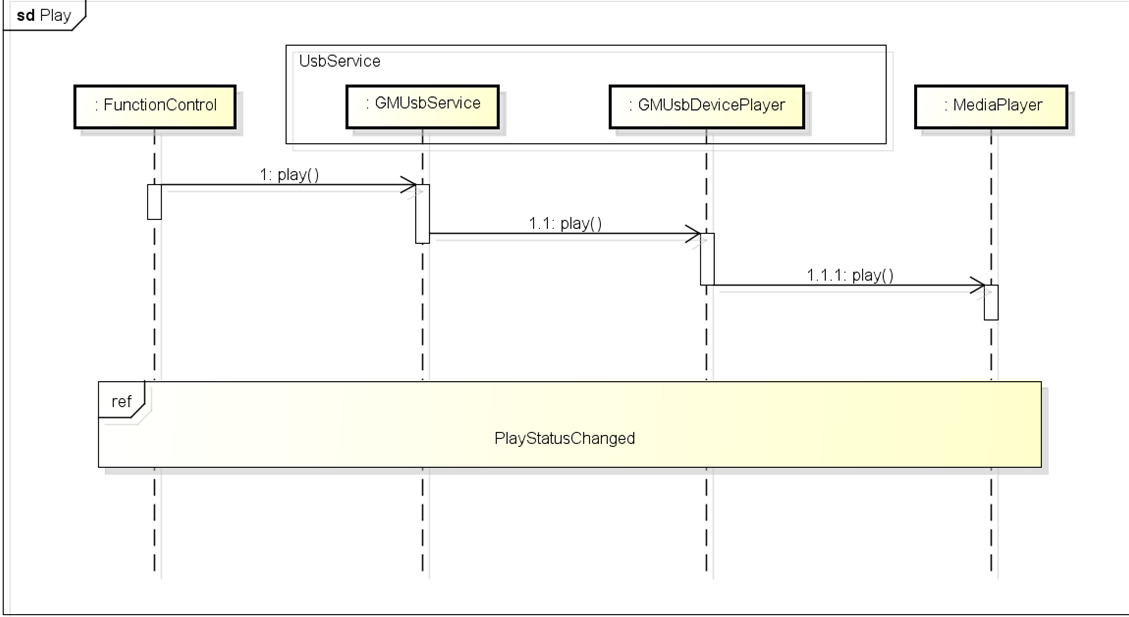
### UsbService与DevicePlayer模块的交互

在USbService中实现了plantform层的DevicePlayer的服务代理类NMMPClassicPlayerDelegation。在usbService中接受到message请求在方法onAsyncResponse中被case匹配处理。处理时，调用DevicePlayer中的方法，实现usbService服务与DevicePlayer服务的交互。

## 3 核心方法实现

### Play

#### 时序



#### 涉及的模块

FunctionManager

GMUsbService

DevicePlayer

#### 实现

functionManager 调用proxy代理中play方法

|  |
| --- |
| uint64\_t GMUsbProxy::play()  {  USBLOGD\_FUNCALL; //打log  //定义调用id，静态，unsigned long 型  static uint64\_t call\_id = 1;  unsigned int code = USBMethod\_Play; //方法id,通过枚举类型定义  //准备要发送的数据包  android::Parcel data;  prepareAsyncData(data);  // request service  //向usbservice发送异步请求,通过返回值判断请求是否成功。  if (BS\_NO\_CONNECTION == sendAsyncRequest(code, data)) {  return 0; // invalid call id '0' to indicate error  }  //调用完该方法后，返回call\_id+1  return call\_id++;  } |

proxy代理服务中发出的play请求，在usbService的onAsyncRequest方法中被接收，通过case匹配请求，选择对应的case进行处理。

|  |
| --- |
| int GMUsbServiceBase::onAsyncRequest(SenderId &id, unsigned int code, const android::Parcel &data) {  case USBMethod\_Play:  {  //定义一个task  GMTaskDes des(Runnable\_Pri\_N, 0, "USBMethod\_Play");  //启动主线程的looper方法，将task作为参数传入，调用该类中的play方法  (m\_pMainThread->looper()).postRunnable(GMTaskBind(des,this, &GMUsbServiceBase::play, id));  break;  } |

在UsbService类中play方法被调用

|  |
| --- |
| void GMUsbServiceBase::play(SenderId sid) /\* \_\_0xB02013\_DTFUNCID\_\_ \*/  {  //通过该类中的replyPlay方法，向上层以消息的形式发送一个反馈  replyPlay(sid);  //判断usb的状态  if (GMUsbSourceState\_SourceOn != m\_sourceState) {  USBLOGD("the source is not on, %d", m\_sourceState);  DTLOG\_INFO(0xB02013, "tthe source is not on, %d", m\_sourceState);  return ;  }  //调用该类中的getDPlayer方法，获取DevicePlayer对象，并调用其postPlay方法  if (NULL != getDPlayer(sid)) {  getDPlayer(sid)->postPlay();  }  } |

在DevicePlayer类中，postPlay方法被调用

|  |
| --- |
| void GMUsbDevicePlayer::postPlay() /\* \_\_0x500017\_DTFUNCID\_\_ \*/  {  //打log  USBLOGD\_FUNCALL;  //判断线程类指针是否为空  if (m\_pMPCbThread) {  //不为空时启动新的线程，并调用本类中的play方法  m\_pMPCbThread->looper().postRunnable(MRunnableTask(this, &GMUsbDevicePlayer::play));  }  //为空时，直接通过函数调用的方式调用play方法  else {  play();  }  } |

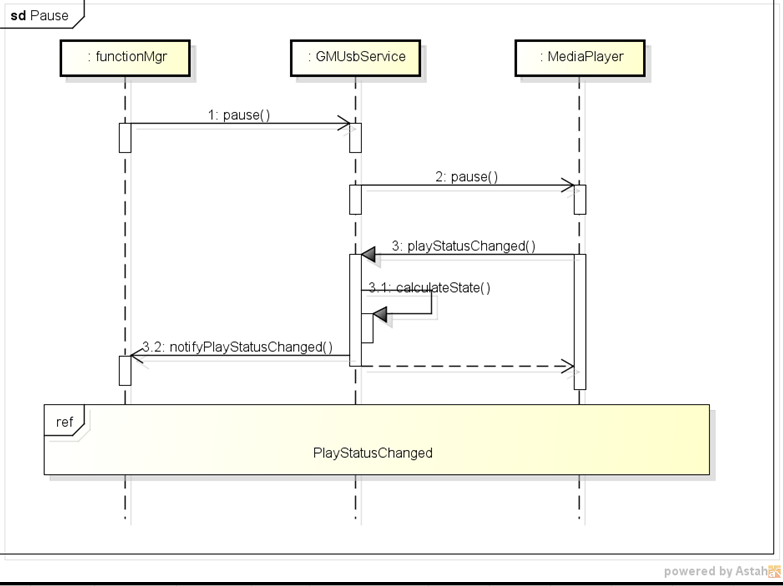
DevicePlayer类中的play方法

在该方法中实现了上层发出的play指令的具体逻辑，分析代码结构发现，最终还是调用的DevicePlayer的父类，也就是在plantform层中NMMpClassciPlayer中的定义的play方法

|  |
| --- |
| void GMUsbDevicePlayer::play()  {  USBLOGD\_FUNCALL;  if (NULL == m\_pMediaPlayer) {  USBLOGD("player not exist");  DTLOG\_INFO(0x500039, "player not exist");  return;  }  //宏  #ifdef GMUSB\_PROPERTY\_FAKESEEK  if (GIUsbItemType\_APE == m\_mediaStatus.itemInfo.eItemType) {  m\_ForwardRewind = PlaySpeed\_NormalPlay;  m\_isSeeking = true;  NMMP\_PropertyMap muteProperty;  NMMPPropSetAudioMute(NMMP\_DEVICE\_ID\_AUDIO\_FRONT, true, muteProperty);  m\_pMediaPlayer->setProperty(muteProperty);  if (GIUsbPlayStatus\_Playing != m\_mediaStatus.ePlayStatus) {  m\_pMediaPlayer->play();  }  MediaBackup::instance()->updatePlayTime(m\_deviceId, m\_mediaStatus.itemInfo.position, m\_mediaStatus.ePlayMode);  if (NULL != m\_pTimerHolder) {  m\_pTimerHolder->stopTimer();  }  m\_pMediaPlayer->seekTo(m\_mediaStatus.itemInfo.position, m\_ForwardRewind, false);  NMMP\_PropertyMap soundProperty;  NMMPPropSetAudioMute(NMMP\_DEVICE\_ID\_AUDIO\_FRONT, false, soundProperty);  m\_pMediaPlayer->setProperty(soundProperty);  }  #ifdef GMUSB\_PROPERTY\_FAKEOTHERSEEK  else if (PlaySpeed\_NormalPlay != m\_ForwardRewind) {  m\_ForwardRewind = PlaySpeed\_NormalPlay;  m\_isSeeking = true;  MediaBackup::instance()->updatePlayTime(m\_deviceId, m\_mediaStatus.itemInfo.position, m\_mediaStatus.ePlayMode);  if (NULL != m\_pTimerHolder) {  m\_pTimerHolder->stopTimer();  }  m\_pMediaPlayer->seekTo(m\_mediaStatus.itemInfo.position, m\_ForwardRewind, false);  }  #endif  else {  m\_ForwardRewind = PlaySpeed\_NormalPlay;  m\_pMediaPlayer->play();  }  #else  m\_ForwardRewind = PlaySpeed\_NormalPlay;  m\_pMediaPlayer->play();  #endif  } |

### Pause

#### 时序



#### 涉及模块

FunctionManager

GMUsbService

DevicePlayer

#### 实现

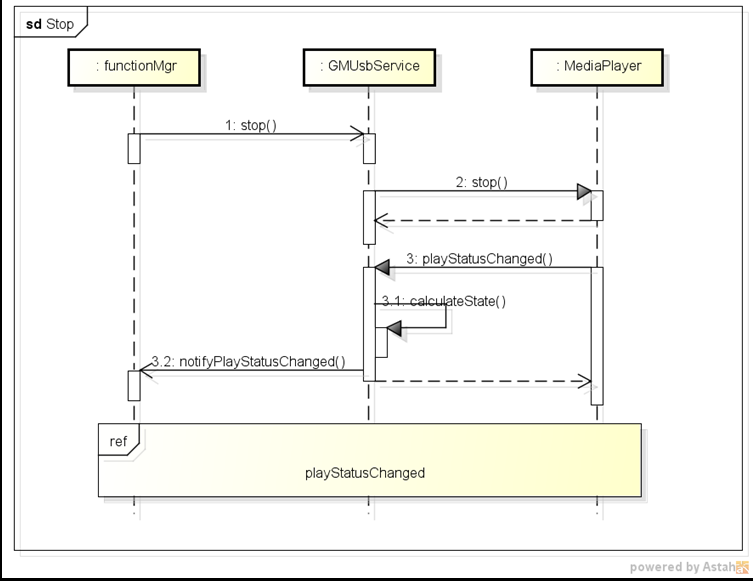
代码的实现逻辑和play一致，都是从functionManager发出请求的message,在service中做中转同时给上层发反馈，最终在GMUSbDevicePlayer（MediaPlayService代理）中实现逻辑。

这里列举在GMUsbDevicePlayer中的pause方法，在该类中的pause中，又调用了plantform层的NMMpClassicPlayer中的pause方法。

|  |
| --- |
| void GMUsbDevicePlayer::pause() /\* \_\_0x50003A\_DTFUNCID\_\_ \*/  {  USBLOGD\_FUNCALL;  if (NULL == m\_pMediaPlayer) {  USBLOGD("player not exist");  DTLOG\_INFO(0x50003A, "player not exist");  return;  }  m\_pMediaPlayer->pause();  } |

### Stop

#### 时序



#### 涉及模块

FunctionManager

GMUsbService

DevicePlayer

#### 实现

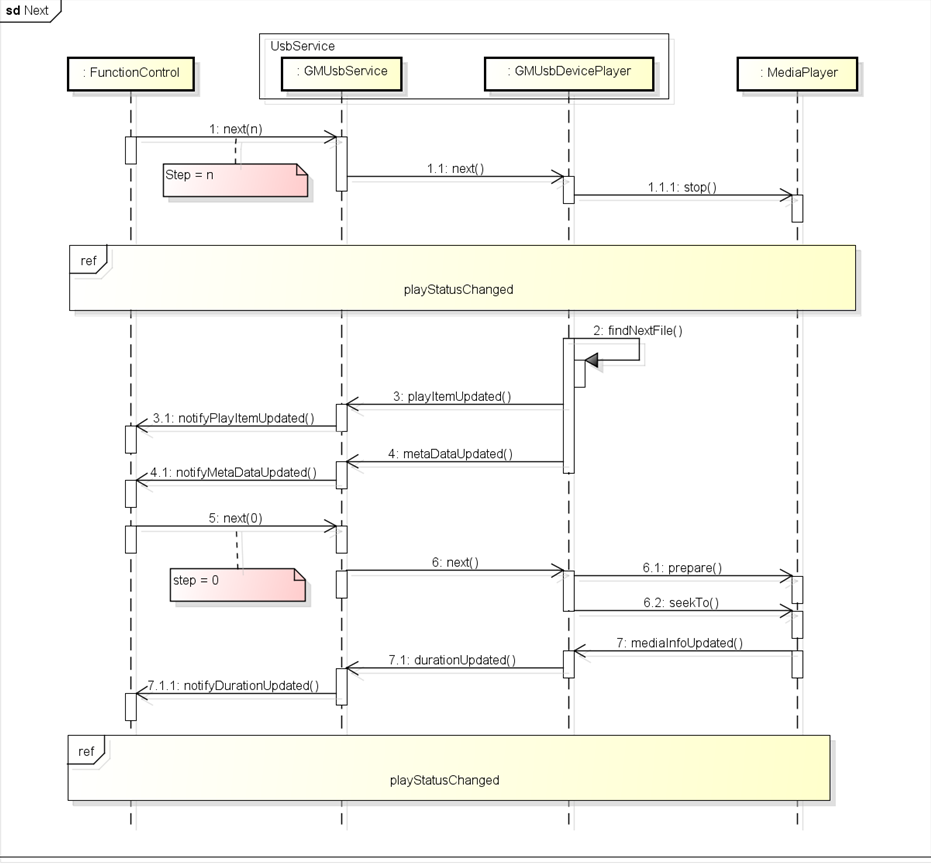
代码的实现逻辑和play一致，都是从functionManager发出请求的message,在service中做中转同时给上层发反馈，最终在GMUSbDevicePlayer（MediaPlayService代理）中实现逻辑。

这里列举在GMUsbDevicePlayer中的stop方法，在该类中的stop中，又调用了plantform层的NMMpClassicPlayer中的stop方法。

|  |
| --- |
| void GMUsbDevicePlayer::stop() /\* \_\_0x50003B\_DTFUNCID\_\_ \*/  {  USBLOGD\_FUNCALL;  //判断mediaPlayer对象是否为空。  if (NULL == m\_pMediaPlayer) {  USBLOGD("player not exist");  DTLOG\_INFO(0x50003B, "player not exist");  return;  }  //判断曲目的id是否为无穷大，不为则设置为无穷大。再调用下层的stop方法，同时设置playStatus为stoped状态。  if (m\_mediaStatus.itemInfo.uid != INFINITE) {  m\_mediaStatus.itemInfo.uid = INFINITE;  m\_pMediaPlayer->stop();  m\_mediaStatus.ePlayStatus = GIUsbPlayStatus\_Stopped;  }  } |

### Next

#### 时序



#### 涉及模块

FunctionManager

GMUsbService

MediaPlayer

#### 实现

在functionMaanger中调用usbService代理proxy中的next方法，向usbService发送异步请求。

|  |
| --- |
| uint64\_t GMUsbProxy::next(const uint32\_t step)  {  USBLOGD\_FUNCALL;  static uint64\_t call\_id = 1;  unsigned int code = USBMethod\_Next;  android::Parcel data;  prepareAsyncData(data);  data.writeInt32(step);  // request service  if (BS\_NO\_CONNECTION == sendAsyncRequest(code, data)) {  return 0; // invalid call id '0' to indicate error  }  return call\_id++;  } |

在usbService中通过case匹配的方式接受处理请求，并通过新启线程的方式，调用usbservice中的next方法。

|  |
| --- |
| 代码和以上模块类似。。。 |

在next方法中，向上层通知通知，同时调用GMUsbDevicePlayer类中的postNext方法。

|  |
| --- |
| 代码和以上模块类似。。。 |

在postNext方法中调用GMUsbDevicePlayer中next方法。通过主线程是否存在判断是通过线程调用还是函数调用。

|  |
| --- |
| 代码和以上模块类似。。。 |

在next方法中实现了具体的逻辑。在next中又调用了playingList中的next.。其实真正的next的逻辑实在playingList中实现的

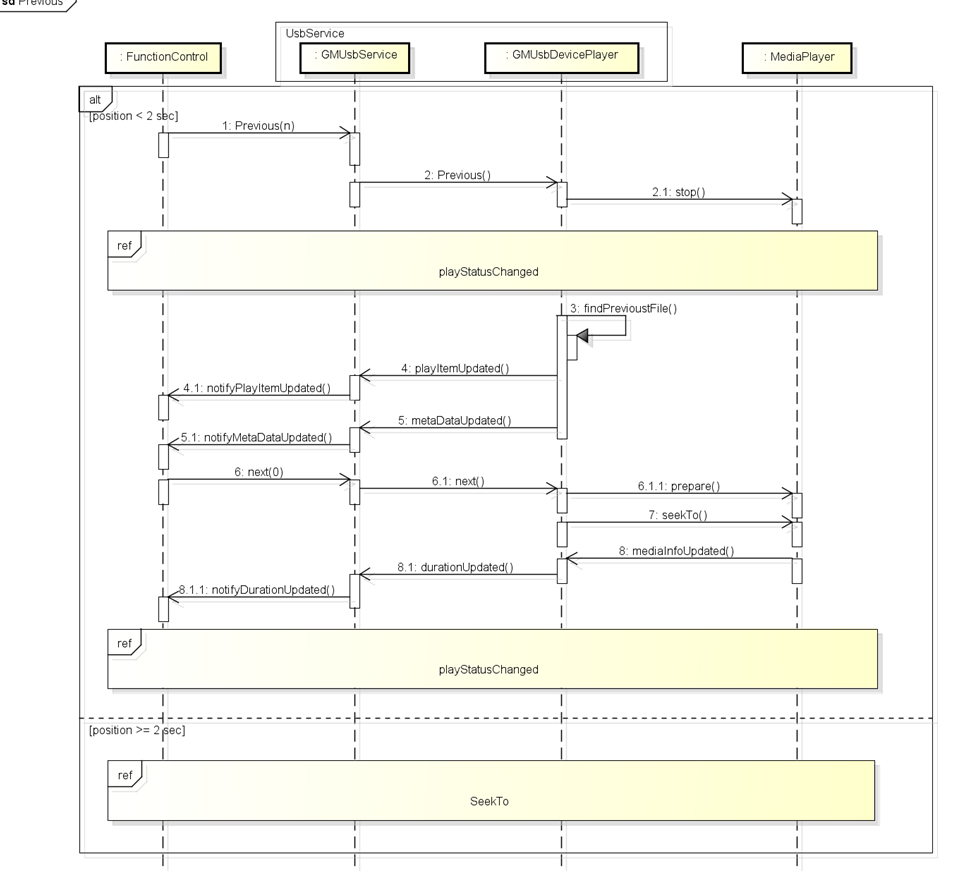
|  |
| --- |
| void GMUsbDevicePlayer::next(int step, bool forceSequence, bool autoPlay) /\* \_\_0x500028\_DTFUNCID\_\_ \*/  {  USBLOGD\_FUNCALL;  //判断播放列表是否为空。m\_pNowPlayingList在initialize方法中被初始化。  if (NULL == m\_pNowPlayingList) {  USBLOGD("m\_pNowPlayingList not exist");  DTLOG\_INFO(0x500028, "m\_pNowPlayingList not exist");  return;  }  //判断是否连接了mediaplayer  if (m\_isMpBlinkReset) {  USBLOGD("ignor this command, wait for mediaplayer connect");  DTLOG\_INFO(0x500028, "ignor this command, wait for mediaplayer connect");  return;  }  //当步数大于0时  if (0 < step) {  USBLOGD("continue to next");  m\_readyToPlay = false;  //调用stop先停止当前播放  stop();  //调用playingList中的next  m\_pNowPlayingList->next(step, forceSequence);  }  //当步数为0时且自动播放时  if (0 == step || true == autoPlay) {  USBLOGD("stop next and start to play, autoPlay = %d", autoPlay);  m\_readyToPlay = true;  //当前播放item的id不等于INFINITE（无限大）  if (INFINITE != m\_mediaStatus.itemInfo.uid) {  //设置播放准备，在setPrepare中，又调用了prepare方法，而在prepare中又调用了  //底层的prepare方法，真正的prepare逻辑是在plantform中实现。  setPrepare();  //设置播放，调用该类中的setPlay f方法。  setPlay();  }  else {  USBLOGE("ItemInfo uid is not available, trigger failed!!");  DTLOG\_INFO(0x500028, "ItemInfo uid is not available, trigger failed!!");  }  }  else {  USBLOGD("the step is invalid");  DTLOG\_INFO(0x500028, "the step is invalid");  }  } |

在GMUsbDevicePlayer中又调用了GMMediaPlayList类中的next方法。

|  |
| --- |
| bool GMMediaPlaylist::next(unsigned int step, bool forceSequence)  {  PLLOG\_FUNCALL;  if (NULL == m\_cb || DeviceType\_None >= m\_deviceInfo.type() || DeviceType\_Count <= m\_deviceInfo.type()) {  PLLOGW("next : param error, m\_cb = [%p], type = [%d]", m\_cb, m\_deviceInfo.type());  return false;  }  PLLOGD("next step = [%d], forceSequence = [%d]", step, forceSequence);  if (m\_totalError) {  PLLOGD("all error >> m\_errorCount = [%d]", m\_errorCount);  PLLOGD("[debug] : all error!!!,please choose another playlist");  m\_cb->onPlaylistEnded(EndedType\_AllError);  return false;  }  if (!forceSequence && (RandomMode\_All == m\_randomMode || RandomMode\_List == m\_randomMode)) {  if (m\_randomIndex >= m\_random.size()) {  PLLOGD("randomIndex error = [%d]", m\_randomIndex);  return false;  }  for (unsigned int i = 0, startIndex = 0; i < step; ++i) {  startIndex = m\_randomIndex;  do {  ++m\_randomIndex;  if (m\_randomIndex >= m\_random.size()) {  m\_randomIndex = 0;  }  if (startIndex == m\_randomIndex) {  // all file in this list has been played once,reset the list  random\_shuffle(m\_random.begin(), m\_random.end());  for (unsigned int j = 0; j < m\_random.size(); ++j) {  if (NULL != m\_random[j].get()) {  m\_random[j]->setPlayed(false);  }  }  m\_errorCountInList = 0;  }  } while (itemFilter());  m\_errorCountInList = 0;  m\_random[m\_randomIndex]->setPlayed(true);  }  // keep normal index point to the currentItem  m\_originIndex = m\_random[m\_randomIndex]->index();  if (RandomMode\_All == m\_randomMode) {  if (1 == amendListInfoIndex()) {  amendIndexAndCount();  }  }  updatePlaylistInfo();  if (m\_listError) {  PLLOGD("[debug] : list error!!!,please choose another playlist");  m\_listError = false;  m\_cb->onPlaylistEnded(EndedType\_ListError);  }  else {  m\_cb->onPlayItemChanged(m\_random[m\_randomIndex]);  }  }  else { // randomMode\_Off or forceSequence  if (m\_originIndex >= m\_origin.size()) {  PLLOGD("originIndex error = [%d]", m\_originIndex);  return false;  }  for (unsigned int i = 0; i < step; ++i) {  do {  ++m\_originIndex;  if (m\_originIndex >= m\_origin.size()) {  m\_originIndex = 0;  m\_errorCountInList = 0;  }  } while (itemFilter(forceSequence));  }  // keep the randomIndex point to the current item  if (forceSequence) {  for (unsigned int i = 0; i < m\_random.size(); ++i) {  if (m\_random[i]->index() == m\_originIndex) {  m\_randomIndex = i;  break;  }  }  }  if (RepeatMode\_All == m\_repeatMode || RepeatMode\_One == m\_repeatMode) {  if (1 == amendListInfoIndex()) {  amendIndexAndCount();  }  }  updatePlaylistInfo();  if (m\_listError) {  PLLOGD("[debug] : list error!!!,please choose another playlist");  m\_listError = false;  m\_cb->onPlaylistEnded(EndedType\_ListError);  }  else {  m\_cb->onPlayItemChanged(m\_origin[m\_originIndex]);  }  }  PLLOGD("operation next succeed");  return true;  } |

### Previous

#### 时序



#### 涉及模块

FunctionManager

GMUsbService

MediaPlayer

#### 实现

这里只列举在DMUsbDevicePlayer中的被调用的previous方法。Previous的逻辑和next的逻辑一致。

|  |
| --- |
| //在devicePlayer中，往线程队列中发送调用该方法的请求  void GMUsbDevicePlayer::previous(int step, bool autoPlay)  {  USBLOGD\_FUNCALL;  if (NULL == m\_pNowPlayingList) {  USBLOGD("m\_pNowPlayingList not exist");  DTLOG\_INFO(0x500029, "m\_pNowPlayingList not exist");  return;  }  if (m\_isMpBlinkReset) {  USBLOGD("ignor this command, wait for mediaplayer connect");  DTLOG\_INFO(0x500029, "ignor this command, wait for mediaplayer connect");  return;  }  if (0 < step) {  USBLOGD("continue to previous");  m\_readyToPlay = false;  //调用该类中的stop方法，停止播放  stop();  //调用下一曲方法，传入step  m\_pNowPlayingList->previous(step);  }  if (0 == step || true == autoPlay) {  USBLOGD("stop previous and start to play, autoPlay = %d", autoPlay);  m\_readyToPlay = true;  if (INFINITE != m\_mediaStatus.itemInfo.uid) {  setPrepare();  setPlay();  }  else {  USBLOGE("ItemInfo uid is not available, trigger failed!!");  DTLOG\_INFO(0x500029, "ItemInfo uid is not available, trigger failed!!");  }  }  else {  USBLOGD("the step is invalid");  DTLOG\_INFO(0x500029, "the step is invalid");  }  } |

在GMUsbDevicePlayer中调用了GMMediaPlayList的previous方法，实现真正的逻辑

|  |
| --- |
| bool GMMediaPlaylist::previous(unsigned int step)  {  PLLOG\_FUNCALL;  if (NULL == m\_cb || DeviceType\_None >= m\_deviceInfo.type() || DeviceType\_Count <= m\_deviceInfo.type()) {  PLLOGW("previous : param error, m\_cb = [%p], type = [%d]", m\_cb, m\_deviceInfo.type());  return false;  }  if (m\_totalError) {  PLLOGD("all error >> m\_errorCount = [%d]", m\_errorCount);  PLLOGD("[debug] : all error!!!,please choose another playlist");  m\_cb->onPlaylistEnded(EndedType\_AllError);  return false;  }  m\_errorCountInList = 0;  if (RandomMode\_All == m\_randomMode || RandomMode\_List == m\_randomMode) {  if (m\_randomIndex >= m\_random.size()) {  PLLOGD("randomIndex error = [%d]", m\_randomIndex);  return false;  }  m\_originIndex = m\_random[m\_randomIndex]->index();  for (unsigned int i = 0; i < step; ++i) {  do {  if (m\_originIndex == 0 && 0 != m\_origin.size()) {  m\_originIndex = m\_origin.size() - 1;  m\_errorCountInList = 0;  }  else {  --m\_originIndex;  }  } while (itemFilter(true));  m\_origin[m\_originIndex]->setPlayed(true);  }  // keep the randomIndex point to the current item  for (unsigned int i = 0; i < m\_random.size(); ++i) {  if (m\_random[i]->index() == m\_originIndex) {  m\_randomIndex = i;  break;  }  }  // mark played in randomArray  m\_random[m\_randomIndex]->setPlayed(true);  if (RandomMode\_All == m\_randomMode) {  if (1 == amendListInfoIndex()) {  amendIndexAndCount();  }  }  updatePlaylistInfo();  if (m\_listError) {  PLLOGD("[debug] : list error!!!,please choose another playlist");  m\_listError = false;  m\_cb->onPlaylistEnded(EndedType\_ListError);  }  else {  m\_cb->onPlayItemChanged(m\_random[m\_randomIndex]);  }  }  else { // randomMode\_Off  if (m\_originIndex >= m\_origin.size()) {  PLLOGD("originIndex error = [%d]", m\_originIndex);  return false;  }  for (unsigned int i = 0; i < step; ++i) {  do {  if (m\_originIndex == 0 && 0 != m\_origin.size()) {  m\_originIndex = m\_origin.size() - 1;  m\_errorCountInList = 0;  }  else {  --m\_originIndex;  }  } while (itemFilter());  }  if (RepeatMode\_All == m\_repeatMode || RepeatMode\_One == m\_repeatMode) {  if (1 == amendListInfoIndex()) {  amendIndexAndCount();  }  }  updatePlaylistInfo();  if (m\_listError) {  PLLOGD("[debug] : all error!!!,please choose another playlist");  m\_listError = false;  m\_cb->onPlaylistEnded(EndedType\_ListError);  }  else {  m\_cb->onPlayItemChanged(m\_origin[m\_originIndex]);  }  }  PLLOGD("operation previous succeed");  return true;  } |

### PlayAll

#### 时序

#### 涉及模块

FunctionManager

GMUsbService

DevicePlayer

#### 涉及文件

#### 实现

在GMUsbDevicePlayer中的playAll方法。

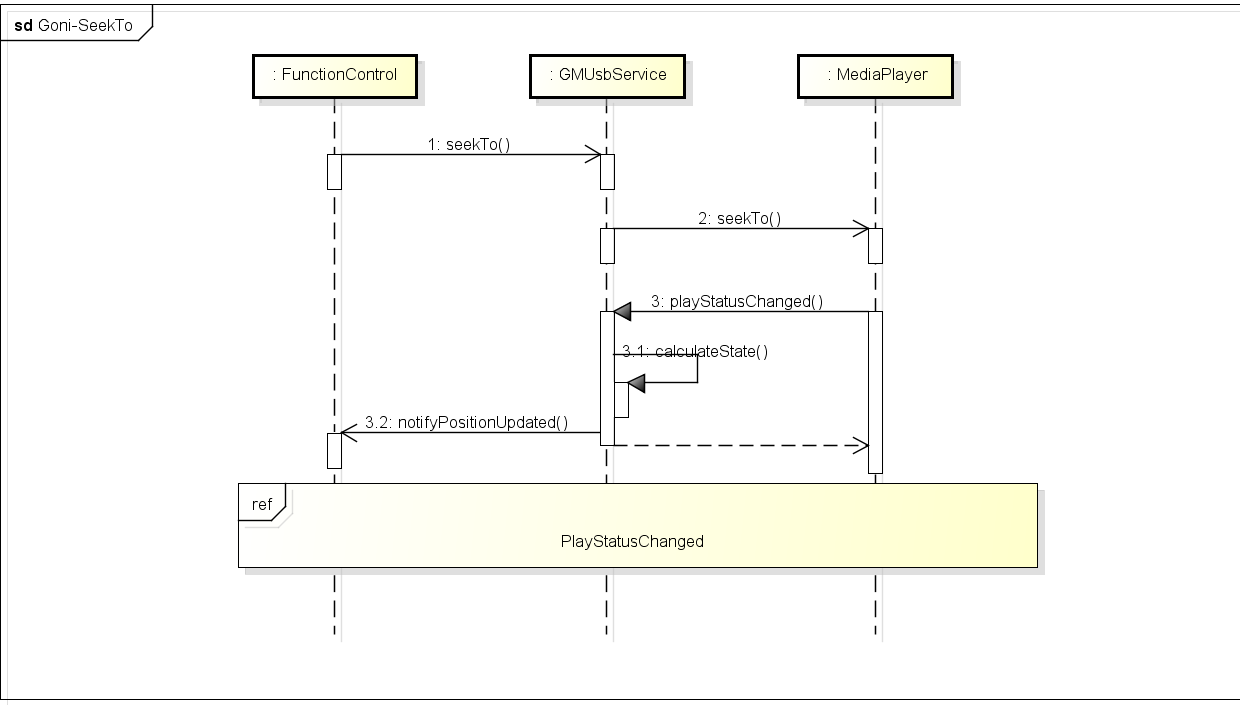
|  |
| --- |
| void GMUsbDevicePlayer::playAll(GMUsbAvMode mode, unsigned int itemid) /\* \_\_0x500022\_DTFUNCID\_\_ \*/  {  USBLOGD\_FUNCALL;  //判断playingList是否为空。  if (NULL == m\_pNowPlayingList) {  USBLOGD("m\_pNowPlayingList not exist");  DTLOG\_INFO(0x500022, "m\_pNowPlayingList not exis");  return;  }  //设置当前速度。  m\_ForwardRewind = PlaySpeed\_NormalPlay;  //调用暂停方法stop  stop();  //设置播放模式  GIUsbPlayMode playMode = static\_cast<GIUsbPlayMode>(mode);  //更新播放模式。  playModeUpdated(playMode);  //调用GMMediaPlaylist中的playAll方法  m\_pNowPlayingList->playAll(static\_cast<GMMediaPlaylist::AvMode>(mode), static\_cast<int>(itemid));  //调用GMMediaPlayList中save方法保存  m\_pNowPlayingList->save();  } |

在GMMediaDevicePlayer中调用了GMMediaPlayList类中的playAll方法，在其中实现 了真正的playAll逻辑

|  |
| --- |
| //AVMode mode参数：  bool GMMediaPlaylist::playAll(AvMode mode, int index)  {  PLLOG\_FUNCALL;  PLLOGD("playall mode = [%d], index = [%d]", mode, index);  int fileType = 0;  m\_playMode = mode;  if (AvMode\_Audio == mode) {  m\_playlistInfo.filePlayMode = FilePlayMode\_Audio\_All;  fileType = 2;  }  else if (AvMode\_Video == mode) {  m\_playlistInfo.filePlayMode = FilePlayMode\_Video\_All;  fileType = 4;  }  else {  return false;  }  m\_playlistInfo.folderId = USB\_INVALID\_ID;  m\_playlistInfo.genreId = USB\_INVALID\_ID;  m\_playlistInfo.artistId = USB\_INVALID\_ID;  m\_playlistInfo.composerId = USB\_INVALID\_ID;  m\_playlistInfo.albumId = USB\_INVALID\_ID;  MediaSrcType type = srcType();  if (MediaSrcType\_Any == type) {  PLLOGW("srcType error");  return false;  }  NCUri uri;  NCList<NCString> select;  NCString where("");  NCString mkey("");  NCList<NCString> whereArgs;  NCString order("SORTKEY");  int dbStatus = deviceDbInfo(DeviceDbInfo\_Status);  if (AvMode\_Video == mode) {  uri = GIMediaProviderUri::FileBase::getContentUri(type);  select<< new NCString("ID")<< new NCString("FOLDER\_ID")<< new NCString("NAME")<< new NCString("PATH");  where = "TYPE=?";  mkey.format("%d", fileType); // 2 refs audio , 4 refs video  whereArgs.append(&mkey);  }  else if (4 <= dbStatus) {  uri = GIMediaProviderUri::AudioBase::Songs::getContentUri(type);  select<< new NCString("FID")<< new NCString("FOLDER\_ID")<< new NCString("FILE\_NAME")<< new NCString("FILE\_PATH")  << new NCString("COMPOSER\_ID")<< new NCString("GENRE\_ID")<< new NCString("ARTIST\_ID")<< new NCString("ALBUM\_ID");  }  else {  uri = GIMediaProviderUri::FileBase::getContentUri(type);  select<< new NCString("ID")<< new NCString("FOLDER\_ID")<< new NCString("NAME")<< new NCString("PATH");  where = "TYPE=?";  mkey.format("%d", fileType); // 2 refs audio , 4 refs video  whereArgs.append(&mkey);  }  #ifndef \_\_LOCAL  NCCursor\* cursor = NULL;  cursor = m\_pProviderResolver->query(uri, select, where, whereArgs, order);  if (NULL == cursor) {  PLLOGD("query failed, no data received");  select.clearData();  return false;  }  int listCountTmp = cursor->getCount();  PLLOGD("list Count = [%d]", listCountTmp);  cursor->moveToFirst();  ncsp<NMDataTable>::sp listSp = new NMDataTable(listCountTmp, PlayItem::Column\_Count);  if (NULL != listSp.get()) {  INIT\_TBL(listSp);  for (int i = 0; i < listCountTmp; ++i) {  listSp->putLong(i, PlayItem::Column\_Fid, cursor->getInt64(PlayItem::Column\_Fid));  listSp->putLong(i, PlayItem::Column\_FolderId, cursor->getInt64(PlayItem::Column\_FolderId));  listSp->putString(i, PlayItem::Column\_Name, cursor->getString(PlayItem::Column\_Name).getString(),  cursor->getString(PlayItem::Column\_Name).getLength());  listSp->putString(i, PlayItem::Column\_Path, cursor->getString(PlayItem::Column\_Path).getString(),  cursor->getString(PlayItem::Column\_Path).getLength());  if (AvMode\_Video == mode) {  listSp->putLong(i, PlayItem::Column\_ComposerId, USB\_INVALID\_ID);  listSp->putLong(i, PlayItem::Column\_GenreId, USB\_INVALID\_ID);  listSp->putLong(i, PlayItem::Column\_ArtistId, USB\_INVALID\_ID);  listSp->putLong(i, PlayItem::Column\_AlbumId, USB\_INVALID\_ID);  }  else {  listSp->putLong(i, PlayItem::Column\_ComposerId, cursor->getInt64(PlayItem::Column\_ComposerId));  listSp->putLong(i, PlayItem::Column\_GenreId, cursor->getInt64(PlayItem::Column\_GenreId));  listSp->putLong(i, PlayItem::Column\_ArtistId, cursor->getInt64(PlayItem::Column\_ArtistId));  listSp->putLong(i, PlayItem::Column\_AlbumId, cursor->getInt64(PlayItem::Column\_AlbumId));  }  listSp->putLong(i, PlayItem::Column\_FileType, fileType);  cursor->moveToNext();  }  loadInfoList(listSp);  selectPlaylist(listSp, index);  // saveLastList();  }  FREEIF(cursor);  #else  NMDataTable\* tmp = GMUsbProvider::instance()->query(uri, select, where, whereArgs, order);  if (NULL != tmp) {  int listCountTmp = tmp->getRowCount();  ncsp<NMDataTable>::sp listSp = new NMDataTable(listCountTmp, PlayItem::Column\_Count);  INIT\_TBL(listSp);  for (int i = 0; i < listCountTmp; ++i) {  listSp->putLong(i, PlayItem::Column\_Fid, tmp->getLong(i, PlayItem::Column\_Fid));  listSp->putLong(i, PlayItem::Column\_FolderId, tmp->getLong(i, PlayItem::Column\_FolderId));  size\_t len;  listSp->putString(i, PlayItem::Column\_Name, tmp->getString(i, PlayItem::Column\_Name, len),  256);  listSp->putString(i, PlayItem::Column\_Path, tmp->getString(i, PlayItem::Column\_Path, len),  256);  if (AvMode\_Video == mode) {  listSp->putLong(i, PlayItem::Column\_ComposerId, USB\_INVALID\_ID);  listSp->putLong(i, PlayItem::Column\_GenreId, USB\_INVALID\_ID);  listSp->putLong(i, PlayItem::Column\_ArtistId, USB\_INVALID\_ID);  listSp->putLong(i, PlayItem::Column\_AlbumId, USB\_INVALID\_ID);  }  else {  listSp->putLong(i, PlayItem::Column\_ComposerId, tmp->getLong(i, PlayItem::Column\_ComposerId));  listSp->putLong(i, PlayItem::Column\_GenreId, tmp->getLong(i, PlayItem::Column\_GenreId));  listSp->putLong(i, PlayItem::Column\_ArtistId, tmp->getLong(i, PlayItem::Column\_ArtistId));  listSp->putLong(i, PlayItem::Column\_AlbumId, tmp->getLong(i, PlayItem::Column\_AlbumId));  }  listSp->putLong(i, PlayItem::Column\_FileType, fileType);  }  loadInfoList(listSp);  selectPlaylist(listSp, index);  }  FREEIF(tmp);  #endif  select.clearData();  return true;  } |

### SeekTo

#### 时序



#### 涉及模块

FunctionManager

GMUsbService

DevicePlayer

#### 实现

在GMUsbDevicePlayer中seekTo被调用。在seekTo中又调用了NMMPClasssicPlayer中的seekTo方法和PlaylistManager中的autoNext方法。

|  |
| --- |
| void GMUsbDevicePlayer::seekTo(uint32\_t secTimeIn) /\* \_\_0x50003C\_DTFUNCID\_\_ \*/  {  USBLOGD\_FUNCALL;  if (NULL == m\_pMediaPlayer) {  USBLOGD("player not exist");  DTLOG\_INFO(0x50003C, "player not exist");  return;  }  USBLOGD("the position need to seekto is %d", secTimeIn);  USBLOGD("the duration is %d", m\_mediaStatus.itemInfo.duration);  // because the operation of seekto is not finish,so the data is not be saved.  MediaBackup::instance()->updatePlayTime(m\_deviceId, secTimeIn, m\_mediaStatus.ePlayMode);  m\_mediaStatus.itemInfo.position = secTimeIn;  uint32\_t totalTime = m\_mediaStatus.itemInfo.duration;  if (-1 == static\_cast<int>(secTimeIn)) {  USBLOGD("play from tail of the item %d", secTimeIn);  m\_pMediaPlayer->seekTo(0, PlaySpeed\_NormalPlay, true);  }  else if ((totalTime > 0) && (((totalTime - secTimeIn) / 1000) < 1)) {  USBLOGD("stop seekto and next");  playCompletedNotify(0);  stop();  if (NULL != m\_pNowPlayingList) {  m\_pNowPlayingList->autoNext();  }  }  else {  if (!m\_isSeeking) {  #ifdef GMUSB\_PROPERTY\_FAKESEEK  if (GIUsbItemType\_APE == m\_mediaStatus.itemInfo.eItemType) {  m\_mediaStatus.itemInfo.position = secTimeIn;  play();  }  else {  m\_isSeeking = true;  m\_pMediaPlayer->seekTo(secTimeIn, PlaySpeed\_NormalPlay, false);  }  #else  m\_isSeeking = true;  m\_pMediaPlayer->seekTo(secTimeIn, PlaySpeed\_NormalPlay, false);  #endif  }  else {  USBLOGD("seekto is not completed ignore the command");  m\_ignorePosition = secTimeIn;  }  }  } |

PlaylistManager中的autoNext方法

|  |
| --- |
| bool GMMediaPlaylist::autoNext()  {  PLLOGD("autoNext");  if (NULL == m\_cb || DeviceType\_None >= m\_deviceInfo.type() || DeviceType\_Count <= m\_deviceInfo.type()) {  PLLOGW("autoNext : param error, m\_cb = [%p], type = [%d]", m\_cb, m\_deviceInfo.type());  return false;  }  if (m\_totalError) {  PLLOGW("warning : autoNext !!totalError!!");  m\_cb->onPlaylistEnded(EndedType\_AllError);  return true;  }  NCAutoSync autoSync(m\_syncObj);  if (RepeatMode\_One == m\_repeatMode) {  // need not switch to next playitem, play this again  if (NULL != currentPlayItem().get()  && currentPlayItem()->playable()) {  m\_cb->onNeedPlayAgain(currentPlayItem());  return true;  }  else {  return next(1);  }  }  else if (RepeatMode\_List == m\_repeatMode || RandomMode\_List == m\_randomMode) {  if (m\_infoList[m\_listIndex].isListError) {  return listUp(m\_playlistInfo.filePlayMode);  }  else {  return next(1);  }  }  else {  return next(1);  }  } |

### BindSource

#### 时序

无

#### 涉及模块

FunctionManager

GMUsbService

DevicePlayer

#### 实现

该方法主要是将usb与设备绑定

Proxy代理

|  |
| --- |
| // binding source usb1/usb2  uint64\_t GMUsbProxy::bindSource(const GIUsbSource& eSourceId)  {  USBLOGD\_FUNCALL;  static uint64\_t call\_id = 1;  unsigned int code = USBMethod\_BindSource;  android::Parcel data;  prepareAsyncData(data);  data.writeInt32(eSourceId);  data.setDataPosition(0);  // request service  if (BS\_NO\_CONNECTION == sendAsyncRequest(code, data)) {  return 0; // invalid call id '0' to indicate error  }  return call\_id++;  } |

UsbService中接收消息，启动线程，传入sourceId和senderId，调用该类中的bindSource方法

|  |
| --- |
| case USBMethod\_BindSource:  {  // define request param  GIUsbSource eSourceId;  // unpack request param  //从parse中获取sourceID  eSourceId = static\_cast<GIUsbSource>(data.readInt32());  // call request function  // bindSource(id, eSourceId);  GMTaskDes des(Runnable\_Pri\_N, 0, "USBMethod\_BindSource");  (m\_pMainThread->looper()).postRunnable(GMTaskBind(des, this, &GMUsbServiceBase::bindSource  , id, eSourceId));  break;  } |

UsbService 中的BindSource方法，在这个方法中，又调用了DevicePlayer中的initialize方法，在初始化时进行绑定操作。这也说明了Usbservice只是功能的中转站，最终的处理逻辑还是放在了各个其他服务的代理类中。

|  |
| --- |
| void GMUsbServiceBase::bindSource(SenderId sid, const GIUsbSource& eSourceId) /\* \_\_0xB0200B\_DTFUNCID\_\_ \*/  {  //往上层反馈信息  replyBindSource(sid, eSourceId);  //将source纳入到clientManager的管理中。  m\_spClientMgr->setLink(sid, static\_cast<int>(eSourceId));  if (NULL != getDPlayer(sid)) {  getDPlayer(sid)->initialize();  }  // TODO sync service status with proxy  // GIUsbServiceStatus statusInfo;  // if (NULL != getDPlayer(sid)) {  // statusInfo = getDPlayer(sid)->mediaStatus();  // }  // int deviceId = m\_spClientMgr->getDeviceId(sid);  // m\_usbExtInfo[deviceId].infoMask = mask;  // notifyUsbExtInfo(sid, m\_usbExtInfo[deviceId]);  } |

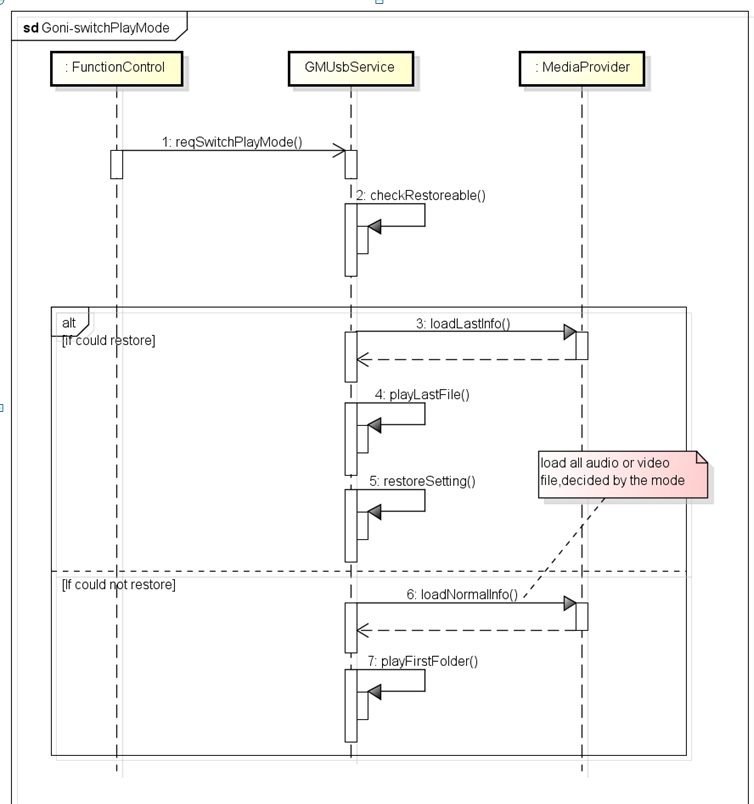
DevicePlayer中的initialize方法。

|  |
| --- |
| bool GMUsbDevicePlayer::initialize() /\* \_\_0x500007\_DTFUNCID\_\_ \*/  {  USBLOGD\_FUNCALL;  //创建新的线程类  if (NULL == m\_pMPCbThread) {  m\_pMPCbThread = new GMUsbRunnableThread();  if (NULL == m\_pMPCbThread) {  return false;  }  }  //创建mediaPlayer对象  //启动线程  if (NULL == m\_pMediaPlayer) {  m\_pMPCbThread->startThread(GONI\_USBSERVICE\_THREAD\_PLAYER\_CALLBACK);  NCRunnableLooper workLooper(m\_pMPCbThread->looper());  //创建mediaPlayer对象  m\_pMediaPlayer = new NMMPClassicPlayer(workLooper, this);  if (NULL == m\_pMediaPlayer) {  return false;  }  // initialize classic player // move to onConnected  // m\_pMediaPlayer->initialize();  // m\_pMediaPlayer->registerNotify(GM\_USB\_MP\_STATUS\_MASK, GM\_USB\_MP\_INFO\_MASK);  // Do not need to connect in this PF Framework  // m\_pMediaPlayer->connectToService();  }  //初始化playlistCallback  if (NULL == m\_pPlaylistCallback) {  m\_pPlaylistCallback = new GMUsbDevicePlayer::PlaylistCallback(\*this);  }  //初始化playlist  if (NULL == m\_pNowPlayingList && NULL != m\_pPlaylistCallback) {  m\_pNowPlayingList = new GMMediaPlaylist(m\_pPlaylistCallback);  //设备信息  GMMediaPlaylist::DeviceInfo devInfo;  devInfo.setRootPath(m\_rootPath);  if (GMUsbDeviceId\_USB1 == m\_deviceId) {  devInfo.setType(GMMediaPlaylist::DeviceType\_Usb1);  }  else if (GMUsbDeviceId\_USB2 == m\_deviceId) {  devInfo.setType(GMMediaPlaylist::DeviceType\_Usb2);  }  else {  devInfo.setType(GMMediaPlaylist::DeviceType\_None);  }  //初始化一系列相关对象后，调用playList中的bindDevice方法  //将devInfo来实例化DeviceInfo  m\_pNowPlayingList->bindDevice(devInfo);  }  #ifdef GMUSB\_PROPERTY\_FAKESEEK  m\_spTimer = new LocalTimer(this, 100);  m\_pTimerHolder = new GMUsbTimerHolder(m\_spTimer.get(), "deviceplayer timer");  #endif  return true;  } |

### SwitchPlayMode

这个方法的主要功能就是Audio和Vedio之间的切换。

#### 时序



#### 涉及模块

FunctionManager

GMUsbService

DevicePlayer

#### 涉及文件

#### 实现

Proxy中SwitchPlayMode方法

|  |
| --- |
| /// switch play mode  uint64\_t GMUsbProxy::switchPlayMode(const GIUsbPlayMode& ePlayMode)  {  USBLOGD\_FUNCALL;  static uint64\_t call\_id = 1;  unsigned int code = USBMethod\_SwitchPlayMode;  android::Parcel data;  prepareAsyncData(data);  data.writeInt32(ePlayMode);  data.setDataPosition(0);  // request service  if (BS\_NO\_CONNECTION == sendAsyncRequest(code, data)) {  return 0; // invalid call id '0' to indicate error  }  return call\_id++;  } |

UsbService中接收并响应proxy中的请求的case

|  |
| --- |
| case USBMethod\_SwitchPlayMode:  {  // define request param  GIUsbPlayMode ePlayMode;  // unpack request param  ePlayMode = static\_cast<GIUsbPlayMode>(data.readInt32());  // call request function  // switchPlayMode(id, ePlayMode);  GMTaskDes des(Runnable\_Pri\_N, 0, "USBMethod\_SwitchPlayMode");  (m\_pMainThread->looper()).postRunnable(GMTaskBind(des, this, &GMUsbServiceBase::switchPlayMode  , id, ePlayMode));  break;  } |

UsbService中在case中调用的该类中的SwitchPlayMode方法

|  |
| --- |
| void GMUsbServiceBase::switchPlayMode(SenderId sid, const GIUsbPlayMode& ePlayMode)  {  // reserved  replySwitchPlayMode(sid, ePlayMode);  //判断资源状态是否可用。  if (GMUsbSourceState\_SourceOn != m\_sourceState) {  USBLOGD("the source is not on, %d", m\_sourceState);  if (NULL != getDPlayer(sid)) {  USBLOGD("MARK DOWN THE PLAYMODE FOR MP BLINKRESET AND STUBBORN UI");  MediaBackup::instance()->updateLastAvMode(pathToDeviceId(getDPlayer(sid)->getRootPath()), static\_cast<int>(ePlayMode));  }  return ;  }  //调用DevicePlayer中的postSwitchPlayMode方法。  if (NULL != getDPlayer(sid)) {  getDPlayer(sid)->postSwitchPlayMode(static\_cast<GMUsbAvMode>(ePlayMode));  }  } |

调DevicePlayer中的postSwitchPlayMode方法。

|  |
| --- |
| void GMUsbDevicePlayer::postSwitchPlayMode(GMUsbAvMode mode)  {  USBLOGD\_FUNCALL;  if (m\_pMPCbThread) {  m\_pMPCbThread->looper().postRunnable(MRunnableTask(this, &GMUsbDevicePlayer::switchPlayMode, mode));  }  else {  switchPlayMode(mode);  }  } |

在devicePlayer类中，postSwitchPlayMode调用该类中的switchPlayMode方法

在该方法的逻辑实现过程中，调用了restorePlay方法，在切换播放模式后恢复播放，恢复失败，则再调用playFirstFolder方法，播放第一个文件夹中的文件

|  |
| --- |
| void GMUsbDevicePlayer::switchPlayMode(GMUsbAvMode mode)  {  USBLOGD\_FUNCALL;  USBLOGD("mode = [%d]", mode);  //调用恢复方法恢复播放。  if (!restorePlay(mode)) {  USBLOGD("restore failed, play first file if exist");  DTLOG\_INFO(0x50002D, "restore failed, play first file if exist");  // playAll(mode, 0);  //恢复播放失败，调用playFirstFolder播放第一个文件夹  playFirstFolder(mode);  }  } |

在switchPlayMode中调用该类中的的恢复播放的方法。目的是在切换模式后还可以恢复切换前的播放状态。

思路：

停止播放，从ＤＢ中取出存储的uid,pos等信息，重新赋给新的模式，然后调用seekTo方法，跳转到指定的位置。

|  |
| --- |
| bool GMUsbDevicePlayer::restorePlay(GMUsbAvMode mode) /\* \_\_0x500051\_DTFUNCID\_\_ \*/  {  USBLOGD\_FUNCALL;  m\_isSeeking = false;  m\_readyToPlay = true;  //判断playinglist是否为空  if (NULL == m\_pNowPlayingList) {  USBLOGD("now play list is null, return");  DTLOG\_INFO(0x500051, "now play list is null, return");  return false;  }  // Gypsophila  //打印DB中存储的备份数据  MediaBackup::instance()->printBackupData();  //设置播放速度为常规速度  m\_ForwardRewind = PlaySpeed\_NormalPlay;  //调用stop方法，停止播放  stop();  //mode为Audio  if (GMUsbAvMode\_Audio == mode) {  m\_mediaStatus.ePlayMode = static\_cast<GIUsbPlayMode>(mode);  //从DB中getPlayUid和播放时间  int *uid* = MediaBackup::instance()->getPlayUid(m\_deviceId, static\_cast<int>(GMUsbAvMode\_Audio));  int pos = MediaBackup::instance()->getPlayTime(m\_deviceId, static\_cast<int>(GMUsbAvMode\_Audio));  USBLOGD("get last audio uid [%d]", uid);  if (m\_pNowPlayingList->load(GMMediaPlaylist::AvMode\_Audio, true)) {  m\_restorePlayFlag = true;  GIUsbPlayMode playMode = static\_cast<GIUsbPlayMode>(mode);  //调用播放模式更新方法，一是在数据库中更新数据，二是通知上层播放模式的改变  playModeUpdated(playMode);  // int repeatMode = MediaBackup::instance()->getRepeatMode(m\_deviceId, static\_cast<int>(mode));  // int randomMode = MediaBackup::instance()->getRandomMode(m\_deviceId, static\_cast<int>(mode));  // m\_pNowPlayingList->setRepeatMode(static\_cast<GMMediaPlaylist::RepeatMode>(repeatMode));  // m\_pNowPlayingList->setRandomMode(static\_cast<GMMediaPlaylist::RandomMode>(randomMode));  从数据库中获取原来的播放模式。  int trickplayMode = MediaBackup::instance()->getTrickPlayMode(m\_deviceId, static\_cast<int>(mode));  //恢复设置  restoreSetting(static\_cast<TrickPlayMode>(trickplayMode));  m\_mediaStatus.itemInfo.position = pos;  m\_pNowPlayingList->selectUid(uid);  #ifdef GMUSB\_PROPERTY\_FAKESEEK  if (GIUsbItemType\_APE == m\_mediaStatus.itemInfo.eItemType) {  play();  }  else {  seekTo(pos);  }  #else  seekTo(pos);  #endif  }  else {  USBLOGW("load audio last info failed");  DTLOG\_INFO(0x500051, "load audio last info failed");  return false;  }  }  else if (GMUsbAvMode\_Video == mode) {  m\_mediaStatus.ePlayMode = static\_cast<GIUsbPlayMode>(mode);  int uid = MediaBackup::instance()->getPlayUid(m\_deviceId, static\_cast<int>(GMUsbAvMode\_Video));  int pos = MediaBackup::instance()->getPlayTime(m\_deviceId, static\_cast<int>(GMUsbAvMode\_Video));  USBLOGD("get last video uid [%d]", uid);  if (m\_pNowPlayingList->load(GMMediaPlaylist::AvMode\_Video, true)) {  m\_restorePlayFlag = true;  GIUsbPlayMode playMode = static\_cast<GIUsbPlayMode>(mode);  playModeUpdated(playMode);  // video do not have repeat&random mode  // int repeatMode = MediaBackup::instance()->getRepeatMode(m\_deviceId, static\_cast<int>(mode));  // int randomMode = MediaBackup::instance()->getRandomMode(m\_deviceId, static\_cast<int>(mode));  // m\_pNowPlayingList->setRepeatMode(static\_cast<GMMediaPlaylist::RepeatMode>(repeatMode));  // m\_pNowPlayingList->setRandomMode(static\_cast<GMMediaPlaylist::RandomMode>(randomMode));  int trickplayMode = MediaBackup::instance()->getTrickPlayMode(m\_deviceId, static\_cast<int>(mode));  restoreSetting(static\_cast<TrickPlayMode>(trickplayMode));  m\_mediaStatus.itemInfo.position = pos;  m\_pNowPlayingList->selectUid(uid);  seekTo(pos);  }  else {  USBLOGW("load video last info failed");  DTLOG\_INFO(0x500051, "load video last info failed");  return false;  }  }  else {  USBLOGD("restore mode error");  DTLOG\_INFO(0x500051, "restore mode error");  return false;  }  return true;  } |

playFirstFolder，先通过getFirstFolderId方法获取第一个folder的id,再通过这个ID调用该类中的playFolder方法，从第一个文件夹开始播放

|  |
| --- |
| void GMUsbDevicePlayer::playFirstFolder(GMUsbAvMode mode)  {  USBLOGD\_FUNCALL;  //得到第一个文件夹的id  unsigned int folderId = getFirstFolderId(mode);  if (0 != folderId) {  //调用该类中的playFolder方法,将第一个folder的ID作为参数传入即可  playFolder(mode, folderId, 0);  }  else {  USBLOGW("get first folder id failed");  DTLOG\_INFO(0x500023, "get first folder id failed");  }  } |

DevicePlayer 中的playFolder方法，在这里调用了PlayingList中的playFolder和save方法。

|  |
| --- |
| void GMUsbDevicePlayer::playFolder(GMUsbAvMode mode, unsigned int folderId,  unsigned int fileIndex) /\* \_\_0x500024\_DTFUNCID\_\_ \*/  {  USBLOGD\_FUNCALL;  //判断m\_pNowPlayingList对象是否创建成功  if (NULL == m\_pNowPlayingList) {  USBLOGD("m\_pNowPlayingList not exist");  DTLOG\_INFO(0x500024, "m\_pNowPlayingList not exis");  return;  }  //设置播放速度为正常。  m\_ForwardRewind = PlaySpeed\_NormalPlay;  //停止  stop();  GIUsbPlayMode playMode = static\_cast<GIUsbPlayMode>(mode);  playModeUpdated(playMode);  //调用m\_pNowPlayingList中的playFoleder方法，实现逻辑  m\_pNowPlayingList->playFolder(static\_cast<GMMediaPlaylist::AvMode>(mode), static\_cast<int>(folderId), static\_cast<int>(fileIndex));  //恢复设置  restoreSetting(TrickPlayMode\_RepAll\_RanOff); // restore default setting  //调用m\_pNowPlayingList保存save  m\_pNowPlayingList->save();  } |