总结三

一、 项目里方法

1、 监听设备连接

首先通过 IMBDeviceConnection 单例方法获取一个单例对象
之后通过其对象方法- startListen 进行监听
在初始化方法中添加监听相应的通知，分别是:
DeviceConnectedNotification（设备连接成功通知）
DeviceDisConnectedNotification（设备断开连接通知）
DeviceChangeNotification（连接设备改变通知）
DeviceNeedPasswordNotification（连接的设备需要密码通知）
DeviceIpodLoadCompleteNotification（连接设备数据加载完成后通知）

2、 设备连接成功后获取设备内 Itunes 数据

通过 DeviceIpodLoadCompleteNotification 通知传递过来的 userInfo 数据获取 IMBBaseInfo 对象，之后通过此对象获取对应的 IMBiPod 对象,之后通过 iPod 对象获取对应的 IMBInformation 对象，最后通过 information 对象的对象方法’-getTrackArrayByMediaTypes:’获取到设备 Itunes 数据信息

```
- (void)deviceIpodLoadComplete:(NSNotification *)notification
{
 NSDictionary *userInfo = [notification userInfo];
 IMBBaseInfo *baseInfo = userInfo[@"DeviceInfo"];

  self.sizeLabel.stringValue = [NSString
stringWithFormat:@"%@: %.01f GB Free/%.01f GB
Total",baseInfo.deviceName,baseInfo.kyDeviceSize/1024.0f/1024
.0f/1024.0f,baseInfo.allDeviceSize/1024.0f/1024.0f/1024.0f];


  IMBiPod *ipod = [[IMBDeviceConnection singleton]
getIPodByKey:baseInfo.uniqueKey];
if (ipod) {
    _information = [[IMBInformation alloc] initWithiPod:ipod];
    _dataArray = [[NSMutableArray alloc]
    initWithArray:[_information
    getTrackArrayByMediaTypes:[IMBCommonEnum
    categoryNodeToMediaTyps:Category_Music]]];
    [_tableView reloadData];

}
```

3、 显示获取到的数据信息

通过 tableView 将获取到的设备 Itunes 数据信息显示到界面之上

二、 实际调用的底层 C 代码

1、 监听设备连接

有关连接设备相关的 API 几乎都来自于 MobileDevice.framework 这个苹果系统自带的私有框架，由于此框架属于系统私有框架，所以不对外提供调用的接口，一切有关此框架的使用接口均来自于对苹果自带相应功能软件的反编译获得。

首先：对设备进行监听，用到的是 MobileDevice.framework 框架里的下面这个函数

```
mach_error_t AMDeviceNotificationSubscribe(
                                am_device_notification_callback
callback,
                                uint32_t unused0,
                                uint32_t unused1,
                                void *callback_data,
                                        am_device_notification
*notification);
```

使用：

```
AMDeviceNotificationSubscribe(notify_callback,  0,  0,  self,
&_notification);
```

其中 notify_callback 为自己写好的一个回调函数，用于响应设备连接的相关改变。传入的_notification 主要用于此后监听的取消

取消监听则使用框架内的下面这个函数

```
mach_error_t AMDeviceNotificationUnsubscribe(

am_device_notification subscription);
```

2、 设备连接成功获取设备信息

设备连接相对应的所有消息将由下面这个函数对其作出相应和接受相应的消息

```
static                  void                  notify_callback(struct
am_device_notification_callback_info *info, void* arg)
```

此函数中的 info 是一个此框架内的结构体

```
struct am_device_notification_callback_info {
am_device  dev;                 // 0    device
uint32_t   msg;                 // 4    one of adnci_msg
} __attribute__ ((packed));
```

这个结构体包含了获取设备相关信息的 am_device 参数和一个连接设备状态的 msg 消息参数

这里的 am_device 是一个框架内自带的结构体句柄

```
typedef struct _am_device           *am_device;
```

根据回调函数里得到的 am_device_notification_callback_info 句柄中的 msg 判断此刻连接设备的状态及相关的一系列信息
msg 里对应的是一个 int32 类型的数值，起对应的数值是下面 enum 中的一个

```
  typedef enum {
   ADNCI_MSG_CONNECTED       = 1,
   ADNCI_MSG_DISCONNECTED    = 2,
   ADNCI_MSG_UNSUBSCRIBED    = 3
} adnci_msg;
```

1 则为设备连接，2 则为设备断开连接，3 则为取消对设备的监听
设备连接成功后，可以通过回调参数 info 里的 dev 获取到一系列设备相关信息，主要通过过框架内的下面函数进行获取

```
CFStringRef       AMDeviceCopyValue(am_device  device,  CFStringRef domain,CFStringRef key);
```

其中的要获取设备内对应的设备信息就必须得通过对应的 domain 和 key 值来进行获取，其中 domain 和 key 都唯一对应着一条相应的设备信息。传入不同的 device 就将获取对应的设备上的数据信息
通过这个函数可以获取例如设备名、udid、产品名、磁盘容量和磁盘可用容量等信息。

3、 复制设备内 ItunesCDB 文件至项目内(非 ios 系统的名字为 ItunesDB)
问题:NSFileManager 也有一个对象方法，专门用于 copy 文件，为什么不用，而要自己写一个 copy 方法呢？

Copy 方法主要用 NSFileManager 的 write 和 read 对象方法进行操作。
首先用下列框架内函数得到一个 afc_connection 的变量

```
afc_error_t      AFCConnectionOpen(am_service      handle,uint32_t io_timeout,afc_connection *conn);
```

在对文件进行操作之前，使用下面函数判断当下能否对文件进行操作，当返回值为 0 的时候，则可以进行操作

```
afc_error_t AFCFileRefOpen(afc_connection conn, const char *path, uint64_t mode,afc_file_ref *ref);
```

之后在 conn 有值的情况下才能保证对文件进行相应的操作

copy 的文件是位于设备内路径为/iTunes_Control/iTunes/iTunesCDB(此路径为定死的路径)名为 ItunesCDB 文件。Copy 到的本地路径为项目内的随意路径即可。复制成功后就进行下面解析文件数据的操作，操作结束之后，就将复制进本地项目内的 ItunesCDB 文件删除。

4、 解析 ItunesCDB 文件数据

这个确实比较难，也在网上百度了相关的知识，但是也是极少量的信息。Itunes 主要是一个二进制文件，如果解析的话，就得一个节点一个节点的进行解析，并且解析的过程步骤也必须得按照文件内的节点和每个节点对应的字节长度进行解析，有一点的出错就将导致后续的解析失败。

iTunesCDB 是一个二进制的文件，其中是用节点来组织的，每个节点都包含特定的信息，比如节点类型，节点头大小，子节点个数等等。有点像 xml，但是和 xml 又不一样，xml 是闭合的。

MHBD 头

二进制文件的头四个字节mhbd 是一个标识,根据特定的标识不同的解析，在解析的时候只要关心自己需要的字段，其他不明白的字段读出来之后保存起来就可以了。下面介绍一下我在解析的时候关注的字段

1) 4字节的头标识 mhbd
2) 4字节的头长度
3) 4字节的总长度

如果是解析的话只需要关注这三个字段就可以了。

这个文件的解析难度还是挺大的，我这周搞了好几天，还是没法解析成功，应该也只能用项目里写好的文件，或者周末再继续试着解析解析。

可以新建一个对应的模型类，对解析出来的数据进行存储。项目里用的是 IMBTrack 进行存储。

5、 显示数据

显示数据同上。

难点：

1、文件 copy 时的 AFCConnectionOpen 和 AFCFileRefOpen 两个函数总是会返回错误的数据，具体原因不明，自己猜测可能是传入的某些参数有误。

2、文件 copy 时如果突然中断设备连接时，文件 copy 的相应操作

3、ItunesCDB 文件解析：到现在没法解析成功，主要原因难度系数比较大，或者说是自己技术能力有限或者对这方面技术知识了解不够。

4、进行复制文件操作之前判断文件能否进行复制的函数 `afc_error_t AFCFileRefOpen(afc_connection conn, const char *path, uint64_t mode,afc_file_ref *ref);`，此函数最终会返回成功的条件是什么？我找了好久，之后在项目中发现，需要在此函数之前添加下面的几个函

数才能使其成功：

```
AMDeviceConnect(dev);
AMDeviceIsPaired(dev);
AMDeviceValidatePairing(dev);
AMDeviceStartSession(dev);
```

问题：

1、下面两个函数的有什么区别，什么时候用哪一个

```
mach_error_t   AMDeviceStartService(am_device   device,CFStringRef service_name,am_service *handle,uint32_t *unknown);

mach_error_t   AMDeviceSecureStartService(am_device   device, CFStringRef service_name, uint32_t *unknown, am_service *handle);
```

2、下面一个函数有什么作用，主要用在什么地方

```
mach_error_t AMDeviceStartSession(am_device device);
```

3、下面函数中的 am_service 的对应参数应该传入什么样的值才会返回对应正确的值

```
afc_error_t   AFCConnectionOpen(am_service   handle,uint32_t io_timeout,afc_connection *conn);
```

4 、框架内以 AFC 开头的函数中的这个 AFC 是什么意思，还有以 AM 开头的函数的 AM 是什么意思，挺想搞明白的

```
afc_error_t AFCDirectoryOpen(afc_connection conn,const char *path,afc_directory *dir);
afc_error_t AFCDirectoryRead(afc_connection conn,afc_directory dir,char **dirent);
afc_error_t AFCDirectoryClose(afc_connection conn,afc_directory dir);

afc_error_t AFCDirectoryCreate(afc_connection conn,const char *dirname);
afc_error_t AFCRemovePath(afc_connection conn,const char *dirname);
afc_error_t AFCRenamePath(afc_connection conn,const char *from,const char *to);
afc_error_t AFCLinkPath(afc_connection conn,uint64_t mode, const char *target,const char *link);
//  NSLog(@"linkpath returned %#lx",AFCLinkPath(_afc,(1=hard,2=sym)"/tmp/aaa","/tmp/bbb"));

// file i/o functions
afc_error_t AFCFileRefOpen(afc_connection conn, const char *path, uint64_t mode,afc_file_ref *ref);
afc_error_t AFCFileRefClose(afc_connection conn,afc_file_ref ref);
afc_error_t AFCFileRefSeek(afc_connection conn, afc_file_ref ref, int64_t offset, uint64_t mode);
afc_error_t AFCFileRefTell(afc_connection conn, afc_file_ref ref, uint64_t *offset);
afc_error_t AFCFileRefRead(afc_connection conn,afc_file_ref ref,void *buf, afc_long *len);
afc_error_t AFCFileRefSetFileSize(afc_connection conn,afc_file_ref ref, afc_long offset);
afc_error_t AFCFileRefWrite(afc_connection conn,afc_file_ref ref, const void *buf, afc_long len);
afc_error_t AFCFileRefLock(afc_connection conn, afc_file_ref ref);
afc_error_t AFCFileRefUnlock(afc_connection conn, afc_file_ref ref);
```

```
mach_error_t    AMDeviceConnect(am_device device);
mach_error_t    AMDeviceDisconnect(am_device device);
uint32_t        AMDeviceGetInterfaceType(am_device device);
//uint32_t      AMDeviceGetInterfaceSpeed(am_device device);
//uint32_t      AMDeviceGetConnectionID(am_device device);
//CFStringRef       AMDeviceCopyDeviceIdentifier(am_device device);
CFStringRef     AMDeviceCopyValue(am_device device, CFStringRef domain,CFStringRef key);
mach_error_t    AMDeviceSetValue(am_device device, CFStringRef domain, CFStringRef name, C
mach_error_t    AMDeviceRetain(am_device device);
mach_error_t    AMDeviceRelease(am_device device);
```

5、下面两个函数的作用（按字面意思能理解为匹配设备是否是同一个，但是看使用的地方，感觉并不是这个意思）

```
int AMDevicePair(am_device device);
int AMDeviceIsPaired(am_device device);
```

收获：

1、明白了很多功能的底层实现和MobilDevice里的部分API的功能作用和使用

2、明白了公司项目的很多功能上的实现大部分都是借助于系统的某些私有框架，而这些私有框架则不提供对外接口，需要自行根据系统自带的具有相应功能的软件进行反编译从而获取框架内相应的接口API

3、对搞明白这些框架的接口API产生了兴趣，挺想要弄清楚这些东西，也想有时间自己学学苹果方面反编译的知识

4、现在还困扰在CDB文件解析上，确实比较麻烦。正在努力搞定