================================================================

## C++

===============================================================

### 1 多个文件时，要将多个文件一起编译才可使用，不然会出现 undefined reference to error得错误。

多文件编译方案：

a 使用g++ 几个文件一起编译 g++ a.cpp b.cpp -o ab

b 分别编译各个源文件，之后对编译后输出的目标文件链接。

用法：

#gcc -c a.cpp //a.cpp编译成a.o

#gcc -c b.cpp //b.cpp编译成b.o

#gcc -o a.o b.o -o test //将a.b.o链接成test

第一中方法编译时需要所有文件重新编译，而第二种方法可以只重新编译修改的文件，未修改的文件不用重新编译。

总结：其实最好的编译方案是通过Makefile文件进行编译

### 2 在.cpp文件中实现.h文件中的方法时，切记一定要把virtual方法实现。

一定要。。。

### 3 在.cpp文件中实现.h文件中的方法时，已经在方法名上做了类限定符号，在方法体中不需要再写

eg:

int Fun::add(){

return get();//get（）方法上不需要再写Fun::

}

### 4 error: ISO C forbids declaration of ‘add’ with no type [-fpermissive]

不让声明，说明函数写的有问题（这次是函数实现时忘记了返回值）

### 5 undefined reference to `vtable for B'

一般是基类的虚函数未被实现

### 6 c++中实例化对象的三种方式及其区别

class A{}

A a；在栈上分配内存，系统自动回收

A a()；在栈上分配内存，系统自动回收，但是不能用来实例化无参构造的类

A \*a =new A；在堆上分配内存，手动回收内存delete，可以实例化有参无参类。。

### 7 运算符重载。

在自定义的类或者结构体中，不能像基础类型那样可以直接使用等号赋值或者使用==判断两者是否相同。因为自定义的类中的一些成员涉及到引用类型。开辟空间在堆中，没法直接通过=赋值。

这时，就需要重载=、==运算符来实现运算符的重载。

### 8 友元类使用复习

友元函数：对于外部的私有的类或者方法，在其他类中无访问权限，但是可以在类中声明友元函数或者友元类来访问类中的私有成员。友元是一种特权访问其他私有对像、函数的方式。

1 全局函数做友元函数

全局函数中可以访问类中的私有成员

### 9 局部函数做友元函数

该类中的局部函数可以访问另一个类中的私有成员。

### 10 友元类

如果这个类被声明为友元类，则这个类中的所有方法，都可以访问其他类的私有成员。

### 11 random\_shuffle (begin,end)

c++STL内部函数。

通过random下的random\_shuffle()方法，将begin-end范围内的数据重置。

### 12 c++ 中内部类 以及内部类和友元类的区别

内部类可以有权限，当定义为private时，外部类不可以访问内部类的成员，但是内部类可以通过类对象的方式访问外部类

内部类是外部类的友元类，即内部类可以访问外部类的所有成员。

但是外部类没有办法访问内部类的私有成员，且访问普通成员得通过实例化对象的方式。

### 13 C++中结构体怎么初始化

https://blog.csdn.net/chenyulancn/article/details/80596339

在c++中，个人理解，struct,用来封装和组合一些数据，方便传输。

struct的初始化

1 通过memset和ZeroMemory等方法来对struct进行初始化。

2 在结构体中可以定义构造函数初始化结构体中的数据。（结构体和类很像，只是默认的作用域不同。）

3 定义一个模版基类，在基类中对数据进行初始化，该结构体继承模版基类。

4 在定义结构体时再对其进行初始化。

### 14在线程类中的run方法什么时候执行

一般线程都有一个start方法来启动线程，当线程启动时，会自动调用该方法，执行其中的逻辑。

### 15 c标准库和c++标准库

#### 由来

c标准库

每年，国际标准化组织(ISO）聚在一起开会制定、修改、定义语言规则，讨论出c、C++的标准。c标准库就是在这些标准中制定的。

C标准库也称为ISO C库，它是在C标准中（例如C11标准）中定义的。其内容分布在不同的头文件中，比如math.h。

C++标准库

和C标准库的概念类似，但仅针对C ++。C++标准库是一组C++模板类，它提供了通用的编程数据结构和函数，如链表、堆、数组、算法、迭代器和任何其他你可以想到的C++组件。C ++标准库也包含了C标准库，并在C++标准中进行了定义（例如C++ 11标准）。

#### 实现

从事于标准库实现的开发者阅读官方的ISO规范并将其转化为代码。他们必须依赖其操作系统所提供的功能（读/写文件，分配内存，创建线程，......所有这些被称为系统调用），

因此每个平台都有其自己的标准库实现。 有时它是系统内核的一部分，有时它是作为一个附加组件 - 编译器 - 必须单独下载。

标准库中调用了系统函数，所以每个平台中使用的c标准都不同，另外，同一平台根据平台需求，对ISO规范也有不同的实现，这就是为啥世面上有那么多c标准库产品的原因。

#### C++ 标准库

c++标准库分为两部分，标注函数库和面向对象类库。



================================================================

## Linux

===============================================================

### 1 在更改make的版本过程中，编译时遇到undefined reference to `\_\_alloca'错误

修改 make文件中的glob/glob.c

添加以下内容

// #if !defined \_\_alloca && !defined \_\_GNU\_LIBRARY\_\_

# ifdef \_\_GNUC\_\_

# undef alloca

# define alloca(n) \_\_builtin\_alloca (n)

# else /\* Not GCC. \*/

# ifdef HAVE\_ALLOCA\_H

# include <alloca.h>

# else /\* Not HAVE\_ALLOCA\_H. \*/

# ifndef \_AIX

# ifdef WINDOWS32

# include <malloc.h>

# else

extern char \*alloca ();

# endif /\* WINDOWS32 \*/

# endif /\* Not \_AIX. \*/

# endif /\* sparc or HAVE\_ALLOCA\_H. \*/

# endif /\* GCC. \*/

# define \_\_alloca alloca

// #endif

保存退出后重新编译即可

================================================================

## Git

===============================================================

1 在使用指令git push -u origin master 往远程服务器上上传文件时报错

To git@github.com:lvlianwei/FileServer.git

! [rejected] master -> master (fetch first)

error: failed to push some refs to 'git@github.com:lvlianwei/FileServer.git'

hint: Updates were rejected because the remote contains work that you do

hint: not have locally. This is usually caused by another repository pushing

hint: to the same ref. You may want to first integrate the remote changes

hint: (e.g., 'git pull ...') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.

原因是本地的仓库中没有README.MD文件

解决：

$ git pull --rebase origin master

$ git push -u origin master

================================================================

## 业务代码

===============================================================

### 1 usbService 模块中的mediaPlayList 文件的作用逻辑。

作用：

Manage the info of GMMediaPlaylist.

主要是维护几个list列表。通过操作这几个list。实现对播放曲目的控制。

### 2 UsbService 中origin、random 、infoList三个vector

在usbService的处理逻辑中维护了三个vector.origin和random还有infoList。其中random表中存储的是什么？是origin的index还是fid。infoList和origin和random有什么关系。

m\_origin.push\_back(itemSp);

m\_random.push\_back(itemSp);

在vector 和 origin列表中存储的是item实体对象。在selectList方法中，实现了将DB中取到的数据存放在list的逻辑。

在infoList 中存放的是PlayItemlist实体。

### 3 usbService中的next方法的核心逻辑总结

usbService的next的核心逻辑在GMMediaPlaylist文件中的next方法中实现

方法参数：

step:next的步数

forceSequence：默认播放模式下为false。

执行过程：

默认播放模式下（repeatAll/random OFF）,在next方法中，顺序播放状态下，将originIndex的值+step后，与originList的size比较。

如果大于了size，则说明需要从头开始计算播放位置。

如果不大于size，则将originIndex的值加step

在强制顺序播放模式下，需要将randomIndex值设置为当前播放的item的index.

repeatALl或者one时，需要修正listInfo的索引。同时 还要更新playlistInfo

最后调用回调类m\_call类中的onPlayItemChanged方法，将信息反馈给调用方。

最终在onPlayItemChanged方法中及之后执行的逻辑暂时还未解析。

### 4 usbService模块中的 GMUsbDataManager::mgrDb方法中的 NCAutoSync autoSync(m\_innerSync)是干啥的。

这是在多线程中的使用的一种锁的机制，通过加入m\_innerSync锁，避免多线程运行过程中产生死锁。

5 playlistInfo对象中的baseCount 和 fileCount这两个全局变量是干啥的

fileCount：在代码注释中解释： file count in this playlist, depend on file play mode, init in func: initCountInList() 即，fileCount是定义在playlist中的file的数量。

baseCount ：这个变量暂时不清楚。

### 6在hancock项目SmtIosService模块中的src下添加了一个MyAdd.h/cpp文件，并且在Gomi.mk中添加编译，在编译时

报以下错误：

No private recovery resources for TARGET\_DEVICE Hancock1S

No recovery.fstab for TARGET\_DEVICE Hancock1S

make: Entering directory `/home/lvlianwei/Hancock'

make: \*\*\* No rule to make target `frameworks/service/smartphoneservice/src/MyAdd.cpp', needed by `out/target/product/Hancock1S/obj/EXECUTABLES/smartphone\_intermediates/src/MyAdd.o'. Stop.

make: Leaving directory `/home/lvlianwei/Hancock'

解决方案:

MyAdd.cpp文件命令多了一个空格。。。。

### 7大编译问题no crti.o库的问题

问题描述： 在执行完单体手顺后，make clean 后，再次大编译就会报出no crti.o crt1.o库找不到的错误。

具体原因不详（个人感觉应该是x86和arm架构下的预编译的原因。）

解决方案：

将新的prebuild文件夹拷贝在工程中覆盖原来的文件夹。

原因 \*\*\*\*\*\*\*\*\*\*\*\*

在执行21MUT手顺的最后清除多余文件的命令时，不应该在项目的根目录下执行，这样会把项目中的所有.o的静态库全部清除，导致再次编译时找不到cri.o库。

该部分的命令应该在对应的UT模块中执行。这样可以避免删除项目中依赖的库

================================================================

## Android

===============================================================

1