#include<stdio.h>

#include<errno.h>

#include<string.h>

**errno.h** 是[C语言](http://baike.haosou.com/doc/5347427.html" \t "_blank)C标准函式库里的标准头文档，定义了通过[错误码](http://baike.haosou.com/doc/1617275.html)来回报错误资讯的宏：

errno宏定义为一个int型态的[左值](http://baike.haosou.com/doc/6981151.html" \t "_blank), 包含任何函式使用errno功能所产生的上一个错误码。

int main()

{

///这一句注释掉也行

extern int errno;

FILE\*fp = NULL;

if(NULL ==(fp = fopen("text.t","r")))

{

printf("errno = %d\n",errno);

printf("Mesg:%s\n",strerror(errno));

}

return 0;

}

libs := libtinyxml.a libzlib.a

如果该libtinyxml.a库在 /lib/tinyxml下，应写为

$(CXX) –L /lib/tinyxml –ltinyxml –c $@

#链接额外库 –L.表示在当前路径下

LDFLAGS = -L. -ltinyxml -L. –lzlib

#加上SDL2链接库

SDL2lib := -lSDL2\_image -lSDL2 -ljpeg –lpng

-l后面的自动扩展

如libtinyxml.so 或

Libtinyxml.a

#以c++11 的形式编译

cpp\_flags := -std=c++11

#files

CPP\_FILES := $(wildcard \*.cpp)

OBJ\_FILES := $(filter-out main.o ,$(subst .cpp,.o ,$(CPP\_FILES)))

all:$(libs) main

.PHONY:all

main : main.o $(OBJ\_FILES)

所谓预编译头，就是把头文件事先编译成一种二进制的中间格式，供后续的编译过程使用。Gcc编译头文件后的中间文件是\*.gch。g++把.h文件当成.cpp文件一样来编译。如果需要控制编译细节。比如常量定义之类的，可加上其他选项。运行之后，会在每个目录里生成一个名叫xxx.h.gch的文件。注意 在编译.gch的过程中，gcc并没有使用环境变量或-I选项来查找被编译的头文件。

为提高效率，应该把不常修改的（或系统提供的）的头文件放在预编译头里，而那些属于你的程序一部分的头文件，一般并不放在预编译头里，因为他们可能随时被修改。

$(CXX) $(SDL2lib) $^ -o $@ $(LDFLAGS)

main.o : main.cpp

$(CXX) $(cpp\_flags) -c $<

$(OBJ\_FILES) : %.o : %.h %.cpp

$(CXX) $(cpp\_flags) -c $^

$(libs):

cd tinyxml;$(MAKE)

cd tinyxml;mv -f libtinyxml.a ../

cd zlib;$(MAKE)

cd zlib;mv -f libzlib.a ../

**vector**

back() 返回vector的最后一个元素，如果vector是 空 则返回值没有定义

pop\_back() 移除最后一个元素，并减少容器size大小

push\_back()

at(index) 传回 index所指向的数据，如果index越界，抛出out\_of\_range

begin() 返回迭代器的第一个数据地址

end() 返回迭代器末端元素的下一个

capacity() 返回容器中的数据个数

empty()

erase(pos) 删除pos位置的数据，并返回下一数据的位置（不知道是指针会怎样）

erase(beg,end) 删除(ben,end)区间的数据，传回下一数据的位置

front() 传回第一个数据

insert(pos,elem) 在pos位置插入一个elem考呗，传回新数据位置

insert(pos,n,elem) 在pos位置插入n个elem数据，无返回值

max\_size() 返回容器中最大数据的数量

insert(pos,beg,end) 在pos位置插入在[beg,end]区间的数据，无返回值

swap(c1,c2)同上

c1.swap(c2) 将c1和c2元素互换

clear() 删除储存在vector的所有元素，若是类则会调用类的析构函数，如果是指向对象的指针，则并不会调用对象的析构函数。这时可以

**eg：**for(int i = 0;i < aValue.size();i++)

delete aValue[i];

aValue[].clear();

注：*调用clear后，vector的尺寸将变为0，但它的容量(capacity)并不发生变化。Vector本身并不释放内存。*

实际上是vector的capacity默认值为0，用push\_back()向vector加入新的object。当size大于capacity的时候，vector申请更大的存储空间，并将全部的元素重新拷贝了一份，就有了生成远远多于vector元素的object个数。

*在vector中 size() capacity() 相对应有两个函数*

*resize(size\_type) reserve(size\_type)*

*size()指目前实际有多少元素。*

*capacity()指至少多少元素才会使重新分配 ？？*

**eg：**

删除vector中满足条件的项可以这样：

std::vector<int>::iterator it;

for(it = count.begin();it != count.end();)

{

if(\*it == 0)

{

count.erase(it);

it = count.begin();

}

else

由end返回的迭代器并不指向vector中的任何实际元素，它只是起一个哨兵（sentinel）的作用，表示我们已经处理完所有元素

it++;

}

**迭代器**（类似指针）

\*操作符（解引用操作符）。取出迭代器指向的内容

自增/自减运算符 指向另一个

*注：任何改变vector长度的操作都会使已存在的迭代器失效。*

eg：1.遍历vector，删除不满足条件的节点

for(it = vec.begin();it != vec.end();++it)

{

if(\*it >2)

vec.erase(it);

}

出错：result：

当vec调用了rease后，it就变成了野指针。对一个野指针++肯定会出错的。

for(it = vec.begin();it != vec.end();)

{

if(\*it >2)

{

it = vec.erase(it);

}

else

{

++it;//注。++it 要比it++效率高一点。

}

}

**C++11。unique\_ptr**

1. 无法进行复制构造和赋值操作。

eg：std::unique\_ptr<int> ap(new int(88));//ok

unique\_ptr<int> one(ap); //error

unique\_ptr<int> two = ap; //error

1. 可以进行移动构造和移动赋值操作。

就像上面的一般意义的复制构造函数和赋值会出错，但在函数中作为返回值却可以使用

unique\_ptr<int> GetVal()

{

unique\_ptr<int> up(new int(88));

return up;

}

unique\_ptr<int> pt = GetVal(); //ok

上面的这几步类似于 unique\_ptr<int> pt =std::move(up);，不过这是显示的所有权转移

1. 可作为容器元素

我们知道auto\_ptr不能作为容器元素，而unique\_ptr也同样不能直接作为容器元素。

eg：

unique\_ptr<int> sp(new int(88));

这一步操作后，sp指针报废

vector<unique\_ptr<int> > vec;

vec.push\_back(std::move(sp)); //ok

vec.push\_back(sp); //error

1. 自定义删除操作

unique\_ptr默认的删除操作为delete/delete[]，若需要，可以自定义。

//资源清理函数

void end\_connection(connection\*p)

{

disconnect(\*p);//实现清理操作

}

//资源清理器的类型

unique\_ptr<connection,decltype(end\_connection)\*>

//传入函数名，会自动转换为函数指针

p(&c,end\_connection);

reset函数

ptr.reset(); //释放ptr指向的内存 等价于 ptr = nullptr

ptr.get(); //从ptr中获得该指针

static std::unique\_ptr<SDL\_Window,void(\*)(SDL\_Window\*)> m\_pWindow;

//..

std::unique<SDL\_Window,void(\*)(SDL\_Window\*)>Window::m\_pWindow = std::unique\_ptr<SDL\_Window,void(\*)(SDL\_Window\*)>(nullptr,SDL\_DestroyWindow);

//...

//init

m\_pWindow.reset(SDL\_CreateWindow(title.c\_str(),SDL\_WINDOWPOS\_CENTERED,SDL\_WINDOWPOS\_CENTERED,H,W,SDL\_WINDOW\_SHOWN));

**SDL，横竖屏切换**

SDL\_RenderCopyEx(SDL\_Renderer\*ren,SDL\_Texture\*tex,const SDL\_Rect\*src,SDL\_Rect\*dest,const double angle,SDL\_Point center,flip);

估计center为锚点

从梦维找到了窗口事件的枚举常量

窗口事件枚举值

可能值

SDL\_WINDOWEVENT\_NONE（never used）

SDL\_WINDOWEVENT\_SHOWN

窗口被显示

SDL\_WINDOWEVENT\_HIDDEN

窗口被隐藏

SDL\_WINDOWEVENT\_EXPOSED

窗口被显露出来且必须被重画（梦维：这种情况一般是窗口被其他窗口挡住，然后被遮挡部分又显示出来时）

SDL\_WINDOWEVENT\_MOVED

window has been moved to data1, data2

SDL\_WINDOWEVENT\_RESIZED

window has been resized to data1xdata2

SDL\_WINDOWEVENT\_SIZE\_CHANGED

窗口大小改变, 不论是用户拖动改变大小还是调用了改变窗口大小的API

SDL\_WINDOWEVENT\_MINIMIZED

SDL\_Event event;

while(SDL\_PollEvent(&event))

{

if(SDL\_WINDOWEVENT\_SIZE\_CHANGED)

{

//改变坐标

}

}

窗口被最小化

SDL\_WINDOWEVENT\_MAXIMIZED

窗口被最大化

SDL\_WINDOWEVENT\_RESTORED

窗口返回普通大小并返回原来位置

SDL\_WINDOWEVENT\_ENTER

窗口得到鼠标焦点

SDL\_WINDOWEVENT\_LEAVE

窗口失去鼠标焦点

SDL\_WINDOWEVENT\_FOCUS\_GAINED

窗口得到键盘焦点

SDL\_WINDOWEVENT\_FOCUS\_LOST

窗口失去键盘焦点

SDL\_WINDOWEVENT\_CLOSE

窗口管理器请求窗口关闭

当横竖屏切换时，就会发生SDL\_WINDOWEVENT\_SIZE\_CHANGED。

SDL\_Rect rect = {m\_gameWidth/2,m\_gameHeight/2,50,100};

SDL\_RenderFillRect时，SDL\_SetRenderDrawColor会改变其颜色，而SDL\_RenderPresent是使清除上一帧，画出下一帧

SDL\_RenderClear是使屏幕都显现該颜色

SDL\_RenderFillRect(m\_pRenderer,&rect);

///draw to the screen

SDL\_RenderPresent(m\_pRenderer);

**SDL\_Surface创建**

SDL\_Surface\*CreateRGBSurface(Uint32 flags,int width,int height,

flags一般为0

int depth,Uint32 Rmask,Uint32 Gmask,

Uint32 Bmask);

创建一个surface。这在有些情况下尤其有用。

SDL\_BlitSurface(SDL\_Surface\*src,SDL\_Rect srcrect,

SDL\_Surface\*dest,SDL\_Rect destrect);

这个是SDL1.2 时主要的绘图函数。类似SDL2的SDL\_RenderCopy()。但也有所不同。

srcrect是剪切src的图片，但destrect则只是用到了x，y坐标而已。这也是SDL2进步的地方

Uint32 SDL\_MapRGB(SDL\_PixelFormat\*,Uint8 r,g,b); 用来赋值不知道颜色格式的情况下用Format

**eg:**

*SDL\_Surface\*sur = SDL\_CreateRGBSurface(0,480,640,32,0,0,0,255);*

注，该format为sur->format

*SDL\_FillRect(sur,&rect,SDL\_MapRGB(sur->format,0xff,0x00,0x00));*

*SDL\_Texture\*tex = SDL\_CreateTextureFromSurface(ren,sur);*

SDL\_SetColorKey(sur,*SDL\_TRUE*,SDL\_MapRGB(sur->format,0x00,0x00,0x00));

这个函数较以前有所不同，现在黑白色都可以直接设置透明色。还有就是SDL\_TRUE,SDL\_FALSE

eg:

//新建一个surface，为黑色（貌似只能为黑色）

*SDL\_Surface\*sur = SDL\_CreateRGBSurface(0,480,640,32,0,0,0,255);*

*//把黑色转换成Uint32位*

Uint32 colorKey = SDL\_MapRGB(sur->format,0x00,0x00,0x00);

//设置黑色为透明色

SDL\_SetColorKey(sur,SDL\_TRUE,colorKey);

SDL强制横屏

处理方法

1:在清单文件中改。

2:在java里的setContentView之前

setRequestedOrientation(ActivityInfo.SCREEN\_ORENTATION\_LANDSCAPE)

*SDL\_RenderClear()配合SDL\_RenderPresent还是很有好处的。不过如果用到系统所给的SDL\_RenderDraw系列的话就不太好了。。。当然,也不算太麻烦,因为SDL\_RenderClear的颜色和SDL\_RenderDraw系列的颜色相同。只需要让他们不同就行，或者干脆画上背景图片也是可以的。*

**IMG\_Init()** Loads dynamic libraries and prepares them for use.Flag should be one or more flags from IMG\_InitFlags.itreturns the flags successfully initialized,or 0 on failure.

初涉TTF

*最近用了TTF\_SetFontStyle，倒是还好。*

*对于字体应有的宽度。SDL2给出了好些种应该所处面积。我尝试用了TTF\_SizeText(m\_font,str,&rect.w,&rect.h)。和SDL\_QueryTexture()作比较。发现后者要好于前者。难道是我姿势不对吗？？？？？*

**SDL\_mixer**

*int Mix\_OpenAudio(int frequency,Uint16 format,int channels,int chunksize);*

*播放器的播放频率。音频播放格式。播放的声道。音效的默认大小。*

*据 吧友 的描述。“第一个参数为播放频率，如果你对音频处理播放很了解，那么你可以填上你自己认为最合适的值，如果不了解，仅仅是想使用，和我一样小白，那么填上Mixer为我们准备的MIX\_DEFAULT\_FREQUENCY,第二个参数为音频播放格式，默认值为MIX\_DEF AULLT\_FORMAT,第三个是播放的声道，默认为MIX\_DEFAULT\_CHANNELS,第四个是音效的默认大小，这个不能乱填，看你的音频文件，自己尝试，填上一个值，如果播放有延迟就适当调小些”*

*“必须在使用所有函数前使用Mix\_OpenAudio，不得将Free函数与加载函数放在同一函数内，即使包含了播放函数” “由于解码器的问题，导致有些格式的音频暂时无法播放。目前android平台无法播放MP3，在使用音频时最好使用wav格式音频，android基本为ogg格式”*

*还有个退出函数Mix\_Quit()*

好像是Mix\_CloseAudio

*在“SDL中文教程”中，有“在初始化函数中，我们调用Mix\_OpenAudio()来初始化SDL\_mixer中的声音函数”，“对于那些使用OGG MOD或其他非WAV格式的音频文件时，请使用Mix\_Init()初始化解码器，并使用Mix\_Quit来关闭解码器。”*

*Mix\_Music结构体 音乐*

函数：

int Mix\_OpenAudio(int frequency,Uint16 format,int channels,int chunksize);

1. Mix\_Music\*Mix\_LoadMUS(const char\*path);
2. Mix\_FreeMusic(Mix\_Music\*);

//是否有音乐在播放 0表示没有，

1. Mix\_PlayingMusic(void);
2. Mix\_PlayMusic(Mix\_Music\*,int loop);//播放音乐，播放次数
3. Mix\_PausedMusic(void)//如果音乐被暂停 1，表示有
4. Mix\_ResumeMusic(void)//恢复音乐的播放

**SDL音效的播放不受SDL\_Delay的影响**

1. Mix\_HaltMusic(void);//停止音乐的播放
2. Mix\_RewindMusic(void);//倒带

Mix\_Chunk结构体 音效

1. Mix\_Chunk\*Mix\_LoadWAV(const char\*path);
2. Mix\_FreeChunk(Mix\_Chunk\*);

//第一个为-1，自动寻找可用频道，第三个循环次数，

失败返回-1

1. int Mix\_PlayChannel(-1,Mix\_Chunk\*,int loop);

匈牙利命名法

基本规则是：变量名 = 属性前缀 + 类型 + 对象描述

属性前缀

|  |  |  |  |
| --- | --- | --- | --- |
| g\_ | 全局作用域 | g\_Servers |  |
| m\_ | 成员变量 | m\_pDoc |  |
| l\_ | 局部作用域 | l\_strName | 少用 |
| c\_ | 常量 |  |  |
| s\_ | 静态变量 |  |  |

类型部分

|  |  |  |  |
| --- | --- | --- | --- |
| a 数组 | p 指针 | fn 函数 | v 无效 |
| h 句柄 | l 长整型 | b 布尔型 | f 浮点型 |
| by 字节 | cnt 计数 | ch 字符 | n 整形 |
| sz 字符串 | d 双精度类型 |  |  |

描述部分

|  |  |  |
| --- | --- | --- |
| Max 最大 | Min 最小 | Init 初始化 |
| T(Temp) 临时变量 | Src 源对象 | Dest 目的对象 |

手机触屏

获取该手指的id

Int id = e.tfinger.fingerId;

typedef Sint64 SDL\_TouchID;

typedef Sint64 SDL\_FingerID;

typedef struct SDL\_Finger

{

SDL\_FingerID id;

float x;

float y;

float pressure;

}SDL\_Finger;

*听到一个吧友说，iostream后台运行代码超过两千行，建议在c++中使用printf，而不是cout。在这里我不推荐，尝试了用了用，莫名其妙的出现了bug，不知道什么原因。百度上“cstdio使c++从c语言的stdio.h继承来的，定义在命名空间std里”我发现加不加std::都可以使用。是codeblocks的问题，还是SDL的问题？我不清楚。。*

SDL设置图标

函数：SDL\_SetWindowIcon(SDL\_Window\*window,SDL\_Surface\*surface)

目前的SDL\_Surface好像可以32 位深也可以用的，（以前有人说只能用8位的）

内存碎片：

虚拟地址系统将极大地帮助降低这个问题出现的几率，虚拟地址系统是操作系统或者内存管理库提供的管理内存

Android shell

在linux中有busybox，对应的android中的toolbox。

system执行dos下面的一些命令，在linux/unix 下，system可以执行所有的shell命令，windows还可以运行可执行文件

**USB2.0的最大传输带宽为480Mbps(即60**[**MB**](http://baike.haosou.com/subview/35572/5142531.htm)**/s)**，**而USB3.0的最大传输带宽高达5.0Gbps(500MB/s)**。

请注意5Gb/s的带宽并不是5Gb/s除以8得到的640MB/s而是采用与SATA相同的10 Bit传输模式(在USB2.0的基础上新增了一对纠错码)，因此其全速只有500MB/s。

不过，大家要注意这是**理论传输值**，如果几台设备共用一个USB通道，主控制芯片会对每台设备可支配的[带宽](http://baike.haosou.com/doc/176719-186678.html)进行分配、控制。如在[USB1.1](http://baike.haosou.com/doc/6846871-7064271.html)中，所有设备只能共享1.5MB/s的带宽。如果单一的设备占用USB接口所有带宽的话，就会给其他设备的使用带来困难。