计划步骤

Godmotion还有什么需要优化的？现在急需进行一场测试，还有就是把接口再整理一下

在这个之后，最好再优化一下模拟器，最好做一个qt版本的

如果这个弄完了，qt也就复习完了

之后再优化一下那个变声软件，通过qt去调节参数，希望能变出理想的声音

如果这个都完成了的话，那么对音频方面的内容就有了一个基本的了解

GodMotion整合到导航中去步骤

第一步，在GodMotion中重载几个接口，符合iRobot

第二部，

# 常用网址

## 手册性网址

### Linux中文man在线手册

<http://os.51cto.com/linuxman/>

### [Ubuntu Manpage: Welcome](http://manpages.ubuntu.com/" \t "_blank)

<http://manpages.ubuntu.com/>

# Vi

网址：

<http://man.chinaunix.net/newsoft/vi/doc/quickref.html>

### 相关设置

export LANG=“zh\_CN.GB18030“

----设置字符的显示格式

:set showmatch

----设置匹配模式：

:set nu

----显示行号

:set nohls

----关闭匹配高亮显示

:set hlsearch

----使匹配的字符串以高亮显示

### 搜索技巧

N 向上搜索

\*

----向下跳到光标所在单词的匹配处

#

----向上跳到光标所在单词的匹配处

g\*

----向下跳到光标所在单词的局部匹配处

----应该是变量的全局定义或局部定处

g#

----向上跳到光标所在单词的局部匹配处

gd

----移动至光标下对象的局布声明

gD

----移动至光标下对象的全局声明

// ??

----重复前面的搜索或反向搜索, 适用于中途间断过搜索模式

----感觉没什么用，因为默认就有这种功能

:g/and 显

----示所有含有and的行

:g/and/p

----显示所有含有and的行

----:g/p1/命令 或 :g!/p1/命令 对包含/不包含p1的行执行命令

----:g/and/:number ----搜索含有and的行，并显示行号

### 替换技巧

总结：替换模式g后面可跟:i c 数字

S前可加% n1,n2 :g/p1

r

----进入替换模式，只能替换当前一个字符

R

----进入替换模式，直到按下ESC为止

----感觉没什么用，还不如用ctrl+v，选中需要替换的，然后按s

:%s/^/ /g 所有行行首加四个空格，若g后面加个3表示只对三行进行操作

----如果列模式不行还可以使用这个方法

:,+3/^/ /g ----这样才是表示对当前行及往下的3行进行操作（一共四行）

:%s/old-text/new-text/g 全文替换，其中%指全文，g指每行所有匹配

:s/old-text/new-text/g 当前行之内替换

----若要使匹配忽略大小写可在g后面加i

----若要每次替换前给出提示，可在g后面加c

----之后y---替换当前一个继续 n---不替换当前并继续 a----一次替换所有

:1,3s/old-text/new-text/g 在第1到3行间替换

:s/old-text/new-text/g 3 在当前行往后共三行间替换

----若要替换整个单词而不是部分替换，则需用\<old-text\>代替old-text

:s/\(old-text1\|old-text2\)/new-text/g

----两种单词替换成同一种单词

s/\(love\)able/\lrs/, 则 loveable变成了lovers ----替换单词的一部分，与用整个单词替换作用差不多，只不过显得更高效而已

:g/p1/s/p2/p3/g 将文本中含有p1的行中的p2用p3替换

:g!/p1/s/p2/p3/g 将文本中不含有p1的行中的p2用p3替换

5s

----对单个字符段的替换

----比如hello world hello world

----先搜索hello，此时光标置于第一个hello的h上， 输入5s

----然后输入abcde

----然后输入”.”

----重复上次命令

### 编辑技巧

D 或d$ ----删除光标右边所有字符

----d换成c表示操作完后进入插入模式

----C 删除光标右边的所有字符，并进入插入模式

d0 或d^ ----删除光标左边的所有字符

----d换成c表示操作完后进入插入模式

dG ----删除当前及之后所有行 d可换成c，表示之后进入插入模式

dw ----删除光标之后的字符，直到下个单词的词首

----db是删除光标之前的字符，直到词首

----de是删除之后的字符，直到词尾

列模式在删除的时候用起来也非常方便

cc ----剪切并进入插入模式

J ----删除行尾换行符，用于合并

y0、y^ ----拷贝光标左边的文本

y$ ----拷贝光标右边的文本

yw ----拷贝，直到词尾

----yb是直到词首

----yiw是拷贝整个单词

----也可以在列模式下来只拷贝中间的字符

----在 列模式下选中后

----y 拷贝

----p 粘贴

u ----撤销上一个命令所做的修改

ctrl + r ----恢复上一次撤销的东西

U ----撤销最近针对该行所做的全部修改 （再按一次，就会恢复修改）

. ----重复上一次命令

:e! 放弃修改，编辑区修改为文件原样

:e 另一文件名 打开指定文件，调入编辑区(但这里怎么切换是个问题 ctrl+6好像可以)

:open 另一文件名，打开指定文件，调入编辑区(但这里怎么切换是个问题 ctrl+6好像可以)

:sp 另一文件名 横向切割屏幕，打开另一文件， 按ctrl +ww 可进行文件间切换

:vsp 另一文件名 纵向切割屏幕，同上，可用pp等进行文件间复制等（但这个有些文件不支持）

:f 显示文件相关信息

:

----按冒号后，可以用上下键查看之前使用的命令（含冒号的命令）

:‘a,’b co ‘c

----将标记a与标记b (含标记a和标记b) 之间的行复制到标记c之下

----若co换成m，则是将那些行剪切到c之下

----也可 :’a,’b y

----然后到对应的地方按 p完成粘贴

----在qt里面，这种方法可以完成在不同文件间的复制与粘贴

----这个功能实在太有用了，相比在终端下直接敲有一些优势了

----相信这个功能在clang中也有，到时候试一试

:’a,’b y

----将会复制

:’a,’b >

----将会向右缩进

‘’

----这个命令在qt中也非常好用，它可以回到上一次光标所在之处

----这个功能在查找变量定义后回到原处时非常有用

### 滚屏及跳转技巧

Ctrl + e ----向上滚动行, 与j、k上下移动光标相比，各有各的优势

Ctrl + y ----向下滚动行

Ctrl +u 向上滚动半屏

ctrl+d 向下滚动半屏

ctrl + g 显示总行数及当前行在总行数中的占比信息

ma 为当前行新建一个标记a， 下次使用`a可以回到这个地方

z+enter 将当前行滚到屏幕顶部

H、M、L 将光标移动到屏幕的首行、中行或尾行

:number 移到第多少行

b、w ----移到上一单词的词首 下一单词的词首

e ----光标移到当前或下一个单词的词尾

0 移到行首

^ 移到行首第一个非空格字符

$ 移到行尾

gm ----移动到改行的屏幕正中间，不满半行则移动到行尾

( 移动到句首

) 移动到句尾 ----以空行为分隔符，认为两空行间是一个句子，用处不大

{ 移动到段首

} 移动到段尾

### vi 使用外部命令

:e filename 载入另一个文件

:file ----显示当前文件属性

:w filename 将当前正编辑的文件保存到另一个文件中

:!cmd

:sh ----不关闭vi，打开一个shell

K ----查询光标处的man

### Vim插件

<http://www.cnblogs.com/wangj08/archive/2013/02/28/2937285.html>

----上面这个网址中有详细介绍

ctags

----yum 可安装

----ctags –R 可生成tags文件

----:set tags=/\*\*/tags

----vim 中设置

----文件更新后，到相应目录下再运行一遍ctags -R

----ctrl + ] 和 ctrl + t 可以实现跳转

Taglists

----需要下载源码，解压：unzip –d taglist name.zip

----sudo cp doc/taglist.txt /usr/share/vim/vim74/doc

----sudo cp plugin/taglist.vim /usr/share/vim/vim74/plugin

----这上面两部应该不需要

应该是放到~/.vim/下对应的目录

----之后打开vim，:Tlist 即可调出该界面

<http://blog.chinaunix.net/uid-26696487-id-3082281.html>

----其实所有的插件安装方法可以试试上面的这种简单方法

# Ue 使用技巧

[ ^t]+$ 替换为空 就会去掉行尾空格

# posix函数

## 单个函数

### mmap 共享内存映射

char \* pShmBuf = (char \*)mmap(NULL, length, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

----返回一个指针

----一共六个参数，第2个：长度 第5个：文件描述符 第6个 偏移量，必须是页大小的倍数，一般填0

第1个：地址，指定映射开始的地方，但一般填NULL，由系统指定地址，第3个：保护模式，可以用或运算符，

第4个，映射是否可间标记，即该块内存是否可供其它内存映射

### shm\_open 创建共享内存对象

int fd = shm\_open(pName, O\_RDWR, 0);

----返回一个文件描述符

----第1个参数：共享内存对象的名称，可以随便指定

----第2个参数：遮盖隐私信息的面具，也相当于权限等标记，如可读可写等

----第3个参数：一般填0

### ftruncate 截断为特定大小

int iRet = ftruncate(fd, length);

----第一个参数：文件描述符

----第二个参数：新的长度

### open 打开文件

int open(const char \*pathname, int flags);

int open(const char \*pathname, int flags, mode\_t mode);

----当 flags中有O\_CREAT的时，必须指定第三个参数，这时需用三个参数的函数

### lseek 调整读写文件的偏移量

off\_t lseek(int fd, off\_t offset, int whence);

----返回最后位置距离文件开始位置的字节数

----第三个参数常用选项有：SEEK\_SET, SEEK\_CUR, SEEK\_END，分别表示设置偏位置，当前位置，结尾位置

----第二个参数随第三个参数的不同所代表的意思不同, 当第三个参数为SEEK\_SET时，整个函数表示的意思是设置位置为第二个参数的值, 当第三个参数为SEEK\_CUR或SEEK\_END时, 整个函数的意思是设置当前的位置为第三个参数的位置加上第二个参数的值

### perror 打印系统错误信息

void perror(const char \*s);

----需要包括头文件：errno.h

----s是自己输入的信息

### read 对文件描述符进行读操作

ssize\_t read(int fd, void \*buf, size\_t count);

----返回读出数据的字节数

----从文件描述符fd 中读count个字节，放到buf中， 但实际读出来的字节数要以返回值为准

### fstat 获得文件的统计信息

int fstat(int fd, struct stat \*buf);

----成功则返回0， 失败则返回-1

### FD\_ISSET 判断文件描述符是否在给定的文件描述符集（fdset）中

int FD\_ISSET(int fd, fd\_set \*set);

----是就返回非零，不是则返回零

----FD\_ZERO 清零文件描述符集

----FD\_SET 往文件描述符集中增加一个

----FD\_CLR 从文件描述符中减少一个

### setsockopt 设置套接字关联的选项

int setsockopt(int sock, int level, int optname, const void \*optval, socklen\_t optlen);

----返回值：成功则返回0， 不成功则返回-1

----第一个参数：将要设置的套接字

第二个参数：选项所在的协议层

第三个参数：需要设置的选项名

第四个参数：指向包含新选项值的缓冲

第五个参数：新选项的长度

### bind 给套接字绑定一个名字

int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

----成功则返回0，失败则返回-1

----第一个参数，需绑定的套接字

----第二个参数，绑定到套接字的名字

----第三个参数，套接字的长度

### listen 监听连接过来的套接字

int listen(int sockfd, int backlog);

----成功时返回0， 失败时返回-1

----第一个参数，用来监听的socket，暂时不做它用，只在这里监听

## 函数块

### select

<http://zhoulifa.bokee.com/6087775.html>

fd\_set rdfds; /\* 先申明一个 fd\_set 集合来保存我们要检测的 socket句柄 \*/  
struct timeval tv; /\* 申明一个时间变量来保存时间 \*/  
int ret; /\* 保存返回值 \*/  
FD\_ZERO(&rdfds); /\* 用select函数之前先把集合清零 \*/  
FD\_SET(socket, &rdfds); /\* 把要检测的句柄socket加入到集合里 \*/  
tv.tv\_sec = 1;  
tv.tv\_usec = 500; /\* 设置select等待的最大时间为1秒加500毫秒 \*/  
ret = select(socket + 1, &rdfds, NULL, NULL, &tv); /\* 检测我们上面设置到集合rdfds里的句柄是否有可读信息 \*/  
if(ret < 0) perror("select");/\* 这说明select函数出错 \*/  
else if(ret == 0) printf("超时\n"); /\* 说明在我们设定的时间值1秒加500毫秒的时间内，socket的状态没有发生变化 \*/  
else { /\* 说明等待时间还未到1秒加500毫秒，socket的状态发生了变化 \*/  
    printf("ret=%d\n", ret); /\* ret这个返回值记录了发生状态变化的句柄的数目，由于我们只监视了socket这一个句柄，所以这里一定ret=1，如果同时有多个句柄发生变化返回的就是句柄的总和了 \*/  
    /\* 这里我们就应该从socket这个句柄里读取数据了，因为select函数已经告诉我们这个句柄里有数据可读 \*/  
    if(FD\_ISSET(socket, &rdfds)) { /\* 先判断一下socket这外被监视的句柄是否真的变成可读的了 \*/  
        /\* 读取socket句柄里的数据 \*/  
        recv(...);  
    }  
}

# 网络

## 长连接

长连接 检测连接 参数 SO\_KEEPALIVE

长连接 若不使用setsockopt设置连接检测时间（连接检测时间可设置=n小时） 连接是不会自动断开

# Powershell

常用命令网址：<http://technet.microsoft.com/en-us/library/hh849902.aspx>

ii ----运行项目指令，相当于图形面的双，全名是invoke-item

Get-Help Get-ChildItem -Online ----获得在线帮助

别名相关指令

Set-Alias -Name uedit -Value C:\'Program Files (x86)'\'IDM Computer Solutions'\UltraEdit\Uedit32.exe ----设置别名

Export-alias alias.ps1 ----导出别名

Import-Alias -Force alias.ps1 ----导入别名

$alias:uedit ----显示别名

## 查找文件

ls \*.txt -R –Force ----当前目录下递归查找.txt文件，-R为–Recurse的简写，-Force不能被简写

ls -path . \*.txt ----在Path 指定的目录下查找，Path不能简写，可小写

ls –path ./\* -Include \*.txt –Exclude D\* ----包含及不包含

ls –n ----只列出文件名，不列出属性

## 查找文本

"Hello","HELLO" | sls "HELLO" –casesensitive

"Hello","HELLO" | sls -pattern "HELLO" –casesensitive ----sls是select-string的简写 –pattern可省略 –casesensitive表示大小写敏感

sls “the the” –path \*.xml ----下面的语法习惯这样写

sls -path \*.xml -pattern "the the" ----当前目录下的.xml文件中查找含the the的行, 当前目录下path可省略

ls c:\windows\system32\\*.txt -r | sls -pattern "Microsoft" –casesensitive ---- ls 与sls联合起来可以实现递归查找文本

sls -path process.txt -pattern idle, svchost –notmatch ----查找不包含idle和svchost的行

查找匹配的上下文：

$f = sls -path audit.log -pattern "logon failed" -context 2, 3

$f.count

($f)[0].context | format-list

# shell(linux)

## history

!500 ----执行序号为500的历史命令

## 进程后台及前台间的切换

在命令的最后加一个&, 这个命令会直接放到后台执行

Ctrl + z切换到后台并暂停，要且回到前台可以先找到进程号，然后用“fg 进程号”可以切回到前台

Jobs命令可查看有多少个任务当前shell的后台运行，并获得各个命令的编号，列如 “jobs”

fg可将后台运行的命令切回到前台运行，列如”fg %1”, 可将编号为1的后台命令切回到前台执行

bg可使后台暂停的命令取消暂停，使其继续运行

## 标准输出 标准错误输出与文件

command 1 >stdout.txt 2>&1 ----标准输出、标准错误输出同时定向到文件

command 1> stdout.txt 2>stderr.txt ----标准输出、标准错误输出定向到不同的文件

## 组合字符

touch file\_{A,B} 将生成file\_A, file\_B两个文件

echo {x1,x2,x3}{y1,y2,y3} 将生成9个字符串，以空格分开

## cut的两个参数 列序号 分割符

echo "abc def ghi" | cut -f1 -d ' ' ----将输出abc

## 类型声明符 declare

declare -i J

J=1+1

echo J #=2, 否则显示的是1+1

## 反引号及刀郎小括号用于存放命令结果

DATE\_01=`date`

DATE\_02=$(date)

## sed单词替换

sed 's/line/LINE/g' Sed.txt 去掉g则替换每行的第一个，g改成2则替换每行的前两个

## sed删除行

sed '/Empty/d' Sed.txt 删除包含Empty的行

sed '5!d' Sed.txt 删除除第5行之外的所有行

sed '/^$/d' Sed.txt 删除空行

## sed删除行并修改原文件

sed -i '1d' file 删除第一行并修改原文件

## sed定位匹配行的下一行

sed '/^$/{n;s/line/LINE/g}' Sed.txt

找到空行，然后再找到它的下一行，并将line替换成LINE 这里n的作用是移到下一行

## 从键盘输入

从键盘输入，为脚本中的变量赋值

read -p "How many box do you want:" N 或：

echo -n "How many box do you want:"

read N

## find exec用法

exec事实上相当于一个管道符

先举几个例字：

find / -name "\*.cont" -exec ls -l {} \; 对找出来的结果进行操作

find / -name "\*.tmp" -exec rm -f {} \;

find . -name "\*" -type f | wc –l

find . -name "\*.c" -type f | wc –l

find . -type f -name "\*" ! -name "\*.o"

find pathname -options [-print -exec -ok ...]

find ./ -name "\*.txt" -exec grep "bin" {} \; ----当前目录下递归查找txt文件，打印出文件中含有“bin”的行，不在前面显示文件名，直接grep会显示文件名

find ./ -name “\*.txt” –exec rm {} \; ----当前目录下递归删除所有txt类型文件

find /home/oracle –mtime 1 –exec cp {} /backup/{}.backup \; 找到文件并在特定的目录下进行备份

上面命令中的大括号表示前面选择出来的逗号

上面的反斜杠加分号，表名这里需要一个分号，不是指shell语句的结束

exec可作为为内建命令使用，这时后面的参数将替代当前的shell，本来当前shell中应该发生的事情，都会在参数中发生

exec >out01.txt 将标注输出重定向到一个文件

exec >/dev/tty 将标准输出重定向到显示器

## locate查找文件名的位置

locate httpd.conf 查找速度比find要快，它是直接在数据库中查,

updatedb

locate httpd.conf

## 脚本返回值

退出脚本时可设置返回值，如exit 0、exit 5，用echo $?可以查看

## eval

将字符串解释成命令执行

eval $cmd

eval echo "$"name$num ----相当于 echo $name1

后面如果是一个变量，则会解释并执行该变量，如果是一个命令并且其中包括变量，则先解释出命令中的变量，然后再执行该命令

## for循环

for J in `seq 10`

相当于J的取值从1开始，一直到10

## 跳出循环

break 跳出一层循环

break 2 跳出两层循环，以此类推

## egrep扩展正则表达式

egrep 'go+d' RegExp.txt “+”代表匹配前面的字符一次及以上

egrep 'go?d' RegExp.txt “？”代表匹配前面的字符一次及以下

## grep 不区分大小写

grep -i 'good' RegExp.txt

## grep与posix字符

grep ^[[:upper:]] RegExp.txt 以大写字母开头的行

## grep与尖括号

在方括号中使用"^"表示反选

grep '[^Gd]ood' RegExp.txt 搜索包含ood的行，但不能是Good或good

## 手机号匹配

"^1[38][0-9]\{9\}” ----匹配移动或联通手机号

## 查看打开的有哪些文件

lsof 查询所有进程打开的所有的文件 ----查不到vi打开的文件

## killall

killall httpd 杀掉叫httpd的进程 比kill 进程号安全

## passwd密码非键盘输入

echo $PASSWORD | passwd --stdin $USERNAME ----不能用于su

## while理解及举例

while 一次读取一行 for一次读取一个空格分割的段

重定向输入是在done后面使用<file的

cat student\_info.txt | while read LINE ----while从文件输入

do

NAME=`echo $LINE awk '{print $1}'`

AGE=`echo $LINE | awk '{print $2}'`

Sex='echo $LINE | awk '{print $3}'`

echo "My name is $NAME, I'm $AGE years old, I'm a $Sex"

done

while read LINE

do

NAME=`echo $LINE | awk '{print $1}'`

AGE=`echo $LINE | awk '{print $2}'`

Sex=echo $LINE| awk '{print $3}'`

echo "My name is $NAME, I'm $AGE years old, I'm a $Sex"

done < student\_info.txt

while [[ $CONTER -gt 0 ]] ----CONTER的值大于0的情况下继续循环

## for理解及举例

for一次读取一个词段，默认以空格为分割符

for LINES in `cat addusers.txt` LINES可以之前没有定义

for ((i=1; i<=10; i++)) ----shell中的for循环

for VAR in $(seq 1 100) ----1到100 步长为1

for VAR in $(seq 1 2 100) ----1到100 步长为2

for VAR in `seq 1 100`

for VAR in $(seq 1 100)

for VAR in {1..5}

for VAR in 1 2 3 4 5

fruits="apple orange banana pear"

for FRUIT in ${fruits}

for FRUIT in apple orange banana pear

## select语句举例

select DAY in Mon Tue Wed Thu Fri Sat Sun

do

case $DAY in

Mon) echo "Today is Monday";;

Tue) echo "Today is Tuesday";;

Wed) echo "Today is Wednesday";;

Thu) echo "Today is Thursday";;

Fri) echo "Today is Friday";;

Sat|Sun) echo "You can have a rest today";;

\*) echo "Unknown input, exit now" && break;;

esac

## 中括号的判断作用

if [ "1" -lt 2 ]; then echo "yes"; fi

if [ -e /var/log/messages ]; then echo "yes"; fi

## 中括号与逻辑与或非等

if [ -e /var/log/messages -a -e /var/log/messages ]; then echo "yes"; fi; ----逻辑与：这里messages存在，两个结果都是0，代表成功，输出yes

----如果messages不存在，两边结果会有1，代表失败，则不输出yes

----测试的是语句，[ 1 –a 1 ]这样的结果不行，返回的也是1

[ -e /var/log/messages ] && [ -e /var/log/messages ] ----这也是逻辑与，不过if不好放

if [ ! -e /var/log/messages1 ]; then echo "yes"; fi; ----逻辑非：输出yes,表明message1不存在

if [ -e /var/log/messages -o -e /var/log/messages1 ]; then echo "yes"; fi ----逻辑或，前一个为真，后一个为假，输出yes

[ -e /var/log/messages ] || [ -e /var/log/messages01 ] ----这也是逻辑或，不过if不好放

## 中括号与字符串比较

[ "$str1" = "$str2" ]

[ "$str1" \> "$str2" ] ----需要转义（但其它等于号，不等于号之类的不用），若不想用转义，则需要用两个中括号

## 中括号与判断字符串是否为空

[ -z "$str2" ] ----判断字符串为空

[ -n "$str2" ] ----判断字符串不为空

## 运算器bc

echo "1 + 1" | bc ----将输出2

cat cal.bc | bc ----也可以处理文件，如文件内容为1+1 2+2 则会输出2 4 一行只能有一个算术，也可有多个算术，但要用分号分开

## 运算器expr

expr 1 + 1 ----结果是2，算术符号需要用空格隔开，乘号等需要使用反斜杠，如expr expr 1+1的结果是1+1

## 双小括号用于算术

var = $((2\*i+1)) ----c语言中能用的表达式

## 刀郎中括号用于算术--$[]

echo $[1+1] 与$(())类似， 乘号不用反斜杠

## 数组的新建及元素的获取

set –A array 1 2 3 4 5; echo $array 显示1 ----ksh

echo $ {$array[3]}显示4

echo $array[1]显示1[1]

declare -a Name=('john' 'sue') ---新建数组并赋值

Name[2]='wang' ----给数组临时增加元素

Score=([3]=3 [5]=5 [7]=7) ----数组的跳号赋值

## 数组所有元素的两种表示方法

${Array[@]}，${Array[\*]}前者得到的是以空格隔开的元素值，后者得到的是一整个字符串

echo ${Array[\*]}

0 1 HelloWorld

## 获取数组元素、元素长度、元素部分字符、元素个数、

echo ${Array[@]:1:2} ----获取数组的第一、第二个元素

1 HelloWorld

echo ${#Array[2]} ----获取某个元素的长度

echo ${Array[2]:0:5} # Hello

echo $(#Array[@]) 或 echo $(#Array[\*]) ----显示数组元素个数

## 数组的拼接、元素值的替换

Conn=(${Array[@]} ${Name[@]}) ----拼接

Array=(${Array[@]/HelloWorld/HelloJohn}) ----替换

## 数组、数组元素的取消

unset Array ----取消一个数组

unset Array[1] ----取消数组中的一个元素

## $相关变量介绍

$? 最后运行的命令的返回值

$0 相当于第零个参数 脚本中则是脚本文件的名称 shell中则是shell文件的名称

$1~$n 保存对应编号的参数 第一个参数编号为1

$\* 存放所有参数的列表，参数与参数间以空格分开, 实际是作为一个整体存放的 必须以 “$\*

$@ 与$\*差不多，但各参数是独立存放的，不是一个整体 暂时还不知道怎么试验 待查看1

$# 保存参数的个数

$! Shell最后运行的后台进程pid, 可以用echo $! 打印出来

$$ shell本身的pid 如echo $$ 则打印出7848

$- 存放着set命令设定的Flag set用来设置shell的各种选项 待查看1

$\_保存之前执行的命令的最后一个参数 但试验结果表明保存的是第0个参数 待查看1

## 设置提示符

在家目录下加一行：PS1=’[$LOGNAME@`uname`:$PWD]’

注：变量还是用单引号括起来

## 设置locale

设置目的 使用户、系统支持中文，支持GB18030

设置方法：编辑/var/lib/locales/supported.d/local这个文件，加入下面这几种编码，然后运行locale-gen –purge，生成locale，然后就可以用export了

zh\_CN.UTF-8 UTF-8

zh\_CN.GB18030 GB18030

zh\_CN.GBK GBK

zh\_CN GB2312

en\_US.UTF-8 UTF-8

# DB2常用系统表

参考网址：<http://www.chinadmd.com/file/ittixwupvxxvoiozu363r6ao_1.html>

<http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp> ----配置

<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/r0011297.htm> ----所有系统表及其各字段英文解释

<http://blog.csdn.net/db2china/article/details/17310159> 常见的10个左右的表介绍（三）

<http://blog.csdn.net/db2china/article/details/9823983> （一）

### 查看有哪些系统表

select \* from syscat.tables t where t.TABSCHEMA='SYSCAT'  with UR ----UR 是一中隔离级别，未提交读，不加锁

----另外三种隔离级别：CS RS RR

sysibm.systables ----存放表名、视图名及别名等

----作用：系统为每一个表，视图和别名在该表中创建一行记录

----试验用列子：

select name,creator,type,ctime

from sysibm.systables

where creator = 'DB2ADMIN'

### 查看表字段

select \* from sysibm.columns where table\_schema = 'FRONT' and table\_name = 'PRODUCT'

### 查看表结构

sql 方式：

select \* from sysibm.syscolumns where tbname = '<table name>'(table name大写) ----表名的各字母需要大写

select NAME, COLTYPE,LENGTH from sysibm.syscolumns where tbname = '<table name>'

命令行方式：

describe select \* from <schema name>.<table name>

describe table <schema name>.<table name>

### 存放表的唯一索引的系统表

通过系统表SYSCAT.INDEXES查询索引

如查询表myschema.tbl的[唯一索引](http://zhidao.baidu.com/search?word=%E5%94%AF%E4%B8%80%E7%B4%A2%E5%BC%95&fr=qb_search_exp&ie=utf8" \t "_blank)的语句：  
select indschema, indname, colnames, uniquerule from syscat.indexes where tabschema='MYSCHEMA' and tabname='TBL1' and uniquerule in ('U', 'P');

其中uniquerule 为'U'表示Unique，'P'表示用于实现Primary Key的索引

### 系统表存放的地方

在 sysibm 下的是系统表  
在用户模式下的是非系统表

### 存放表字段注释的表

select t.Remarks as 注释,t.\* from syscat.COLUMNS t   
where tabschema='模式名' and tabname=upper('表名')

### syscat.tables

----数据库中对象的信息

对象包括：table，view，nickname 和 alias 的一些定义

SYSCAT.TABLES 视图

试用举例：

select \* from syscat.tables where tabname = 'TBL\_FOR\_TEST' --据说这个系统表存放着用户表的表名

--表名的各字母都变成了大写了

--当表名作为表名使用时，是不区分大小写的，当作为字段时，是区分大小写的

### syscat.views

----视图的定义信息

试用举例：

select viewschema,viewname,owner,ownertype,readonly,valid

from syscat.views

where viewschema = 'DB2ADMIN'

### SYSCAT.ROUTINES

----暂时不要写到ppt

DB2 UDF，系统方法（system-generated method），用户定义方法（user-defined method）和存储过程（SP）的定义。我们可以认为该视图包含了数据库中程序的定义

### syscat.dbauth \*\*\*

数据库级授权信息表 ----保存用户或用户组可对数据库进行的操作

试验用列子：select \* from syscat.dbauth

### sysibm.syschecks 或syscat.checks

----check约束信息表

----作用：存放系统中的check约束信息，每个表的每个check约束都对应一条记录

----试验用例子：select \* from sysibm.syschecks where tbname = 'DB2\_TEST\_TBL'

或

select tabschema,tabname,constname

from syscat.checks

where tabname = 'DB2\_TEST\_TBL'

### syscat.tabconst

----约束信息，主键，外键，check，unique等

select tabschema,tabname,constname,type

from syscat.tabconst

where tabschema = 'DB2ADMIN'

### sysibm.referential\_constraints 或syscat.references

----外键约束信息表

----作用:为每个foreign key约束建立一条记录

----试验用例子：

select constraint\_schema, constraint\_name, unique\_constraint\_name

from sysibm.referential\_constraints

或：

select tabschema,tabname,constname,reftabname,refkeyname

from syscat.references

where owner = 'DB2ADMIN'

### sysibm.syscolumns 或syscat.columns

----所有表中列的描述信息

----作用：存放系统中所有表的数据列的描述信息，系统为db2里定义的每个表的每一行建立一条记录

----试验用例子（挑取部分字段）：

select name,tbname,coltype,typename,length,scale,nulls

from sysibm.syscolumns where tbname = 'DB2\_TEST\_TBL'

或：

select tabname,colname,typename,length,scale,NULLs,remarks

from syscat.columns

where tabname='DB2\_TEST\_TBL'

### sysibm.sysindexes 或syscat.indexes

----所有表中索引的描述信息

----一条索引对应一条记录

----试验用例子（挑取部分字段）：

select name,tbname,colnames,uniquerule,remarks

from sysibm.sysindexes

where tbname = 'DB2\_TEST\_TBL'

或：

select tabname,indname,colnames,uniquerule

from syscat.indexes

where tabname = 'DB2\_TEST\_TBL'

# oracle 安装及常用操作

#### 安装过程资料参考

redhat 源更新参考： <http://www.myhack58.com/Article/sort099/sort0102/2013/39651.htm>

oracle安装参考： <http://blog.csdn.net/gyming/article/details/7799370>

#### 安装后启动时常报错误参考

Sql>startup时常会报一些错，这里先把解决问题的地址留在这：

ORA-01102和 ORA-00205: <http://hi.baidu.com/120001240/item/abeda41f8c3102e787ad4e01>

ORA-00845: <http://blog.sina.com.cn/s/blog_545bc35401013zgw.html>

ORA-01078: <http://www.2cto.com/database/201204/129508.html>

#### oracle监听、登录、实例的启动与关闭、用户的切换

$ lsnrctl stop

$ lsnrctl start

$ sqlplus /nolog 或sqlplus sys/123456 as sysdba; 估计后者权限更高  
SQL> connect / as sysdba; 以别的用户名登录为：SQL> conn atman/atman;

SQL> startup

SQL> shutdown 注：一般用shutdown immediate或shutdown normal

#### 创建及删除表空间：

create tablespace appdata default storage(maxextents 5000 pctincrease 5) datafile '/home/oracle/tablespace/data/dcmsdata01.dbf' size 200M;

drop tablespace appdata including contents and datafiles; ----再删对应的文件

假如文件不小心事先删除了，则执行如下操作：

Alter database datafile ‘/home/oracle/tablespace/data/upsdata01.dbf’ offline drop;

然后就可以删除对应的表空间了：drop tablespace appdata;

#### oracle用户的创建、赋权限及删除

SQL> create user atman identified by atman; ----创建数据库用户，这里指定了用户名与密码都是atman;

SQL> grant **create session** to atman; ----给用户atman赋权限，这时可以登录

grant **create procedure** to dcms;

SQL>grant **connect,resource,dba** to atman ----这时可以建表

grant **select on dba\_pending\_transactions** to dcms;

drop user dcms cascade; ----删除oracle用户

#### 脚本中的sqlplus写法

sqlplus -S dcms/dcms <<EOF

<sql>

EOF

#### 设置远程能登录

参考网址：<http://hi.baidu.com/thinkwa/item/e1b96119f8e0cffe746a842b>

#### 查看数据库名、实例名、当前用户、数据库服务名

两种方法： select对应表或使用show语句

SQL> select instance\_name from v$instance; ----查看数据库实例名

SQL> select name from v$database; ----查看数据库名

或SQL>show parameter db\_name

SQL>show user ----显示当前连接的用户

SQL> select value from v$parameter where name = 'service\_name'; ----查看数据库服务名

或SQL>show parameter service\_name

#### 当前用户所有表的表名查看

select table\_name from user\_tables; ----查看当前用户的表名

select table\_name from all\_tables; ----查看所有用户的表名

#### 查看表字段

SQL> desc HOST\_TXN\_POST ----优选方式

其次：

select \* from user\_tab\_columns where Table\_Name='用户表';

select \* from all\_tab\_columns where Table\_Name='用户表';

select \* from dba\_tab\_columns where Table\_Name='用户表';

#### 新建linux用户并指定shell、用户组、oracle的访问权

linux下oracle用户创建方法：useradd –g oinstall –G dba oracle

其他用户访问oracle权限设置：usermod -G oinstall,dba -g oinstall dcms

usermod -a -G groupA user ----将用户添加到某个用户组里面去,然后还要设置PATH, ORACLE\_SID

参考<http://blog.sina.com.cn/s/blog_3fad003a0100eyqd.html>

建用户时指定shell: -s /bin/csh 指定目录 –d /home/ups

举例：

useradd -s /bin/csh -G oinstall,dba -g oinstall ups

添加组用户: groupadd ups

改变用户 用户组: chown ups:ups ups

#### Sqlplus常用设置及操作

SQL> set lines 150

SQL> set wrap off

SQL> set pagesize 1000

SQL> set trimspool on

设置列的显示属性：<http://blog.csdn.net/huozhicheng/article/details/5842433>

Sqlplus与命令行的切换

sql>!  
　　$hostname  
　　$exit  
　　sql>

暂停与恢复输入sql语句（期间执行另一sql语句语句）

Sql>select \* from table\_name where

#desc table\_name

Colname > 4000

Sqlplus中支持快速粘贴

在当页的任何地方，选中后按右键，则自动粘贴到sqlplus的最后一行

Sqlplus中字打错的问题

可以在使用sqlplus之前执行stty erase ^H

也可以设置SecureCRT：会话属性->Terminal->Emulation->Mapped Keys中，选中Backspace sends delete或Delete sends backspace

Sqlplus中的历史命令

l 或list 可以显示上一条使用的语句

e或edit可以编辑上一条使用的语句

#### 数据库字符集修改

blog.csdn.net/xtdhqdhg/article/details/18797475

echo $NLS\_LANG ----查看当前客户端字符集

select userenv(‘language’) from dual; 查看当前数据库字符集

参考搜索：oracle数据库字符集转换规律全面剖析

#### 数据库数据导入(恢复)

imp ups/ups@ORA11 full=y file=/home/ups/sdb/schema/expdat\_atman.dmp buffer=81920

exp ups/ups@ORA11 full=y file=/home/ups/sdb/schema/expdat\_atman.dmp

# tuxedo常用操作

#### 编译服务端、客户端程序

proc test.pc include=%TUXDIR%/include ----服务端程序.pc

buildserver -o simpserv -f simpserver.c -r Oracle\_XA -s TEST ----编译服务端.c程序

buildclient -o testcli -f testcli.c ----编译客户端.c程序

# dcms 安装

### 建表空间及建表

步骤1：~/sdb/schema/database.sql 执行该脚本，创建oracle用户dcms （先要为建立一个tools的表空间）

错误：表：createtable.sql 很多名字冲突，这里面有很多表，怀疑其他.sql是否需要执行

成功：表：newtable.sql

成功：表：sys\_return\_code.sql

成功：表：a.sql

成功：表：dztable.sql 所有报错都是删除时未发现该表

错误：表：atmc\_chk\_rslt.sql 执行时，依赖于ATMC\_DTL表, 而整体搜索没发现该表的建表语句

错误：表：t1.sql 有语法错误

错误：表：t.sql 很多名字有冲突，也不知道这个表是否需要

成功：数：load\_sys\_return\_code.sql

成功：数：load\_sys\_recode\_map.sql 表不存在，实际建表语句在createtable.sql中，挑出来单独执行就行了

----注: 当前目录中还有一个dbload目录，是用来导表数据的，执行目录dbload目录中的tmpload脚本就可以了, 这个步骤需要在上面两个步骤都执行完了再执行

成功：执行成功

步骤2: sdb/data/ load 该命令也是用来导表数据的，脚本执行过程中需要输入参数

成功：三个文件load成功

sdb/data/init\_data.sql 该命令是用来初始化另外一些表数据的，与上面load无关

成功：sqlplus中执行

### Make步骤

一个模块一个模块的来编译

common

cm 编译通过

convert 编译通过

database 编译通过

enc 目录中有多个mak，没有makefile，不知道编译哪一个

evt 目录中有两个.mak，已经手动指定方式编译通过，但有很多警告，感觉不对

kernel 编译通过

mngevt 编译通过，但有很多警告

mngkernel 编译通过

mngmsg 编译通过

msg 编译通过

oorc 编译通过

util 编译通过

comcon 在/usr/include下面新建一个空的stropts.h文件后编译通过

comatm 编译通过

batch 编译通过

cvthost 编译出错 ----链接时失败

cvtin 编译出错 ----链接时失败

cvtout 编译出错 ----链接时失败

dcmscon 编译成功 ----ncurses-devel 这个库需要yum装一下

dcmsmngsvr 编译成功

dcmssvr 编译成功

logsrv 编译成功

monitor 执行mk那个脚本就行，当前目录有一个ap目录，还需要进入里面去编译

unihost 编译出错 tuxedo 装好后将所有文件权限改成777 ，dcms中LDPATHLIB路径追加一下就可以了

unimng 编译出错 tuxedo 装好后将所有文件权限改成777 ，dcms中LDPATHLIB路径追加一下就可以了

注：应参照/$home/sbin/bootall来看成功起起来多少:

结果：起来部分：dcmsmngsvr dcmssvr fr\_cvtin fr\_cvtout logsrv MonitorSend unihost server comatm comcon

未起来部分：

### 64位机器上编译生成32位程序及其对应问题

CFLAGS 后面加-m32, 可能需要修改的FLAGS包括：UP\_INCFLAGS UP\_LIBFLAGS UP\_CCFLAGS UP\_ARFLAGS

更简单的办法： CC 原来是 =cc $(UP\_CCFLAGS)

现在改成： CC = -m32 $(UP\_CCFLAGS)

由于数据库是64位的，因此这种方法解决不了问题

### 需要屏蔽的地方

由于源代码中缺少getmskey.c，所以在src/common/enc/dcmsenc.c中屏蔽了一个函数的调用。

# dcms代码分析

## 函数归类

### GetItemValue

----四个参数：sRootNode、sSubNode、sItem、sValue：前三个匹配后，将对应值填到第四个参数中

----原全局变量结构体中的sValue值中可能含有${abc}的组成部分

### AppInit

ReadSysCfg

getenv("DCMS\_CFG\_FILE")

fopen

fgets

PreProcLine

AppTraceInit

### connect\_database

getenv ----好想默认的用户名与密码都是scdcms,不知道怎么连接了数据库

connect\_db

### Config\_Init

GetItemValue

……

fopen(hostencfile ----在初始化的时候就涉及到秘钥方面了

fgets

SetHostTKKey

cvtInitial

### msqAllInit

----对各个消息队列进行初始化

### DSSysCtl\_s

----select \* into 入参对应的内存

### load\_host\_post

DBHostTxnPost\_Del1 ----清空表

GetItemValue

fopen

while

fread

DBHostTxnPost\_Init

DBHostTxnPost\_Ins

fclose

### load\_uni\_post

DBUniTxnPost\_Del1

DBAtmTxnLog\_Cur

while

DBAtmTxnLog\_Cur

DBUniTxnPost\_Init

DBUniTxnPost\_Ins

DBAtmTxnLog\_Cur

## 模块分析

## batch

----对账用的

AppInit

connect\_database

DSSysCtl\_s

load\_host\_post

load\_uni\_post

proc\_txn\_match

exp\_txn

commit\_work

disconnect\_database

## comatm

----连接过来后貌似没有看到对应的处理，可能是与监控

Config\_Init ---- 里面应该对全局变量Sthr\_Num进行操作才是

msqAllInit

Pthread\_Init

初始化连接表Connection\_Table

创建线程pthread\_attr\_init

Listen\_socket

创建读消息队列和套接字线程，处理报文并发送报文至内部处理模块 pthread\_create

创建读socket线程

pthread\_mutex\_unlock

socket连接循环 while

accept

检查是否存在CheckClientIP

重设socket ID

## comcon

Config\_Init

进入while循环

Listen\_socket

子while循环

accept

DoIt

## cvthost

AppInit

Config\_Init

Connect\_socket

while循环

fork子进程 ----用于发送

ProcSend

recvMsgAPI

cvtDCMS2HOST

fork子进程 ----用于接收

ProcRecv

cvtHOST2DCMS

sendMsgAPI

## cvtin

----三种结构体之间的转化：atmstruct –> dcmsstruct，hoststruct->dcmsstruct

AppInit

cvtInitial ----装载应答码配置（放到数组中）

msqAllInit ----获得配置文件中的消息队列，然后再调用msgget

GetItemValue ----获得配置文件中的密钥

SetHostTKKey ---- memcpy(TK,sHostTK,8)，将密钥放到TK中去

Initialize ----比较复杂，设及到密钥、信号量、共享内存、下载主密钥

while

cvtDCMSInitial ----平台初始化，只是把结构体memset而已

recvMsgAPI ----通讯，根据进程而定

msqAtmPosConInRcv ----case确定的

msgrcv(giMsqAtmPosConInNum, &tIPCMsg, sizeof(tIPCMsg)-sizeof(long), 0, MSG\_NOERROR);

cvtATM2DCMS ----通讯报文转化成内部报文

cvtFMT85832DCMS ----case 0的时候，具体实现竟然是return 0，说明还不支持8583报文

cvtNCR2DCMS ----case 1的时候

sendMsgAPI ----通讯，根据进程而定

内存走向：**Buf**（长度1024）声明 ----> recvMsgAPI(CI\_ATMPOSCONIN, **Buf**,&iMsgLen ) ----> msg\_def \***bufin**=(msg\_def \*)Buf ----> cvtATM2DCMS(cvttype,(char \*)**bufin**->mtext,&bufout)---->cvtNCR2DCMS(char \***ncrBuf**,IPCMsgBufDef \*dcmsbuf) ----> atm\_req\_head\_def \***head**=(atm\_req\_head\_def \*)ncrBuf ----> ncrChildRQK (char \***ncrBuf**,IPCMsgBufDef \*dcmsbuf,char \*caEncOrgidx) atm\_req\_head\_def \***head**=(atm\_req\_head\_def \*)ncrBuf;

atmc\_rqk\_def \***body**=(atmc\_rqk\_def \*)ncrBuf; ---->然后再用memcpy 等对head进行拷贝

IPCMsgBufDef bufout ----> cvtDCMSInitial(&bufout) ----> cvtATM2DCMS(cvttype,(char \*)bufin->mtext,&bufout) ----> sendMsgAPI(bufout.lTo, (char \*)&bufout, sizeof(IPCMsgBufDef))

## cvtout

AppInit

cvtInitial

msqAllInit

Initialize

while

recvMsgAPI

getCvtTypeByDeviceType

cvtDCMS2ATM

sendMsgAPIEx

## dcmscon

AppInit

GetItemValue

get\_param

do – while

----控制台交互方面

## dcmsmngsvr

----估计也是某种控制台相关的

AppInit

SetStartUp

try – catch

cfg.UseXml

GetItemValue

……

DaemonInit

AddObject

……

SetDatabase

Init

Run

## dcmssvr

AppInit

SetStartUp

try – catch

cfg.UseXml

GetItemValue

……

AddObject

……

SetDatabase

Init

Run

## logsrv

AppInit

msqAllInit

dblDaemonizeSelf

connect\_database

for

recvMsgAPI

syslog\_i\_0

commit\_work

## monitor

MonitorSend.c

MonitorReceive.c

MonitorAddOneBrh.c

TestSendQMsg.c

test1.c

## unihost

AppInit

msqAllInit

Initialize

while

recvMsgAPI

fork

CallHost

sendMsgAPI

## unimng

----没有main函数

## 日志跟踪

msg\_def 用于发往socket或从socket接收时

IPCMsgBufDef 内部模块间通信时用

IPCATMPOSMngDef 用于cvtout向外转化时

comatm 管理交易通讯模块

"./src/comcon/mainproc.c" [readonly] line 28 of 140 --20%-- col 16 ----comcon支持的几个管理交易

"./include/kernel/DCMSDef.h" [readonly] line 259 of 297 --87%-- col 14

## 待了解

ATMDESDecrypt ----解密函数，有时间可以了解，暂时将验密标记由1变成0， 方法是./src/common/convert/ncrchildcvt.c 文件中调用UniEncTransferPIN函数的最后一个参数由1改成0

./src/common/convert/softcrypto.c 中函数HostEncPin也需要屏蔽，用if 0 屏蔽

# 文件传输方法

scp 本机文件路径 远程机器用户@远程机器ip:远程机器存放路径

scp 远程用户名@远程地址:文件路径 本机存放路径

# C 常用函数

字符串处理函数

Int isalpha(int c) ----c是字母字符

int isupper(int c) ----c是大写字母

int islower(int c) ----c是小写字母

int tolower(int c) ----当c是大写字母是返回对应小写字母，否则返回c本身

int toupper(int c) ----当c是小写字母是返回对应大写字母，否则返回c本身

int isdigit(int c) ----c是数字字符

int isxdigit(int c) ----c是16进制数字字符

int isalnum(int c) ----c是字母或数字字符

int ispunct(int c) ----c是标点符号

int isgraph(int c) ----c是可打印字符，不包括空格

int isprint(int c) ----c是可打印字符，包括空格

int isspace(int c) ----c是空格、制表符、换行符

int iscntrl(int c) ----c是控制字符

字符串函数

size\_t srlen(const char\*) ----函数返回字符串长度

char \*strcpy(char \*dest, const char \*src) ----字符串拷贝函数

char \*strncpy(char \*dest, const char \*src, size\_t n) ----字符拷贝函数，如果src中字符不足n个，则在dest中填充对应个数的空字符

chat \*strcat(char \*dest, const char \*src) ----把src中的字符复制到dest中已有的字符串之后

char \*strncat(char \*dest,const char \*src, size\_t n) ----把src中最多n个字符复制到dest中已有的字符串之后

int strcmp(const char \*str1, const char \*str2) ----比较两字符串的大小，在前者大于、等于、小于后者时分别返回正值、0、负值

int strncmp(const char \*str1, const char \*str2, size\_t n) ----对两字符串的前n个字符进行比较，在前者大于、等于、小于后者时分别返回正值、0、负值

char \*strchr(const char \*str, char c) ----在str中查询c，返回c第一次出现的位置，没有c时返回NULL。

char \*strrchr(const char \*str, char c) ----在str中查询c最后一次出现的位置，没有时返回NULL。

size\_t strspn(const char \*str1, const char \*str2) ----从str1中寻找一段全由str2中字符组成的序列，返回其长度

size\_t strcspn(const char \*str1, const char \*str2) ----str1中寻找一段全由非str2中字符组成的序列，返回其长度

char \*strpbrk(const char \*str1, const char \*str2) ----在str1中查询str2的组成字符，返回第一个出现字符的位置

char \*strstr(const char \*str1, const char \*str2) ----在str1中查询子串str2， 返回第一个出现的位置，没有时返回NULL

char \*strerror(size\_t n) ----返回与错误编号n相关的错误信息

char \*strtok(char \*str1, const char \*str2) ----在str1中查询str2中字符作为分隔符而形成的单词

存储区操作

void \*memcpy(char \*dest, const char \*src, n) ----从src开始处复制n个字符到dest出，返回dest

void \*memmove（char \*dest, const char \*src, size\_t n） ----从src处复制n个字符到dest处，返回dest，其中dest和src所指字符串可以重叠

int memcmp(const char \*str1, const char \*str2,size\_t n) ----比较由str1，str2开始的n个字符，返回值定义同strcmp

void memchr(const char \*str1, char c, size\_t n) ----在str1中前n个字符范围内查询c，返回该第一次出现的位置，没有该字符则返回NULL

void \*memset(char \*str, char c, size\_t n) ----将str的前n个字符设置为c，返回str

动态存储分配函数

void \*calloc(sizet n, size\_t size) ----分配一个可以存放n个大小为size的对象的存储，并将所有字节用0字符填充,返回该存储块的地址，不能满足时返回NULL

void \*malloc(size\_t size) ----分配一块足以存放大小为size的存储，返回该存储的地址，不能满足时返回NULL

void \*realloc(void \*p, size\_t size) ----将p所指存储块大小调整为size, 返回新块的地址，若恩能满足要求，则新块的内容与原块的内容一致，若不能满足要求时，函数返回NULL， 原块内容部变

void free(void \*p) ----释放p所指向的动态存储块

数值转换

double atof(const char \*str) ----返回由字符串str构造的一个双精度型数值

int atoi(const char\*str) ----返回由字符串str构造的一个整数值

long atol(const char \*str) ----返回由字符串str构造的一个长整数值

char \*itoa(int value char \*string, int radix); 将整形值value以radix进制表示法写入string

执行控制

void abort(void) ----非正常终止函数

void exit(int status) ----正常终止函数

与执行环境交互的函数

Int system(const char \*str) ----把str传给程序的执行环境，使str作为系统命令执行，若str为NULL，则函数返回非0表示环境里有命令解释器

char \*getenv(const char \*str); ----取出执行环境中与字符串str相关联的环境变量的值，如果找不到就返回NULL

排序函数

void \*bsearch(const void \*key, const void \*base, size\_t n, size\_t size, int (\*cmp)(const void \*keyval, const void \*daum));

----函数指针cmp用来确定排序的方式

void qsort(void \*base, size\_t n, size\_t size, int(\*cmp)(const void \*, const void \*));

----函数指针同样式用来确定排序的方式

输入输出函数

FILE \*fopen(const char \*filename， const char \*mode); ----文件的打开

Int fclose(FILE \*stream); ----文件的关闭

int fgetc(FILE \*fp); ----字符输入

int fputc(int c, FILE \*fp); ----字符输出

getc putc ----与上面两函数相似，通常用在宏定义中，如#define getchar() getc(stdin) #define putchar© putc(c, stdout)

格式化输入输出

int scanf(const char \*format, …);

int printf(const char \*format, …);

int fscanf(FILE \*stream, const char \*format, …);

int fprint(FILE \*stream, const char \*format, …);

int sscanf(char \*s, const char \*format, …);

int sprint(char \*s, const char fomt, …);

按行输入输出:

char \*fgets(char \*buffer, int n, FILE \*stream);

int fputs(const char \*buffer, FILE \*stream);

char \*gets(char \*s);

int puts(const char \*s);

直接输入输出：

size\_t fread(void \*pointer, size\_t size, size\_t num, FILE \*strea);

size\_t fwrite(const void \*pointer, size\_t size, size\_t num, FILE \*stream);

# Android

### 窗口与视图

requestWindowFeature(Window.FEATURE\_NO\_TITLE) ---全屏、隐藏窗口标题栏

setFlags(WindowManager.LayoutParams.FLAG\_FULLSCREEN, WindowManager.LayoutParams.FLAG\_FULLSCREEN) ----隐藏系统状态栏

# ppt制作技巧

制作母版

视图----》 母版----》幻灯片制作

母版标题字体----》宋体 字号----》44号 也可设置为楷体24号

除了设置标题外，还可以设置内容 日期 页脚等样式

完成设置后可以关闭母版视图看看效果

为母版配置图片背景

进入母版编辑界面 点击插入菜单 选择图片----》来自文件， 在图片上右击，找到叠放次序 选择置于底层

使用母版

在幻灯片处右击，然后选择新幻灯片，便生成了一个与母版一样的幻灯片

保存母版为设计模式

为了方便下次使用这个母版，需点击文件----》另存为----》保存为pot类型（演示文稿设计类型），可输入自己便于记忆的类型

以后使用方法

以后想使用的时候只需点击设计 然后选择这个母版

Ppt自带模板使用

在ppt空白处右击，选择幻灯片板式

# CMake

## Mindmap

Cmake

Dir

Include

target

syntax

set env define

cmp

others

dir p h cc cF

inc com bin dir

target

exec ----arg1

----arg2

so ----ag1 ST\_or\_SH arg2\_1 ag2\_3 …

proj ----two dir

----no other use

Install ----FILE

----TARGET

Compile ----add\_compile\_option

----add\_definitions

Custom ----command

----target

----output

----target ----fake

Configure\_file

Install prefix

Syntax

----Var define

----fun define and call

----if if\_else foreach

----macro

----option

----and

Env define

----set and get env

----set

----append path

----file cmd

----Include path

Cmp

----strcmp

----int cmp

----match

----match and replace

{

CMAKE\_FIND\_ROOT\_PATH

----交叉编译时用到

----是一个re-rooted目录

}

{

cmake中与文件相关的无非就是：

编译时需要包括的头文件目录

源文件目录

链接库目录位置

链接库名字

----源文件目录和头文件目录之所以没有弄成一个，可能是因为在某些情况下，头文件和源文件是分别处于不同的目录的

----头文件目录和链接库目录关键字中的directories是个复数，而源文件不是复数，这是因为源文件是去除来存放某个变

\_量中的，所以其实际存放的不是目录，而是其中的所有文件，所以并不需要将目录携程复数

----不知道aux\_source\_directory中的aux到底代表什么意思

----最后一个关键字可以读作：目录链接库，其中的库是个复数，因为这个函数的参数里面可以填多个库，所以是复数

\_一定要记住目录链接库这个口头禅，希望到时候不要再来翻看手册了。

include\_directories

----只需要指定安装在系统中的头文件的路径

----当前目录是不需要指定的，因为默认就是当前目录

aux\_source\_directory(不带双引号的相对路径 变量名称)

----这时，该变量名称的值将是该路径下的所有文件

----这应该是FILE GLOB抓取对应目录下cpp文件方法的一种替换

link\_directories

----链接库目录

target\_link\_libraries

----目标链接库文件

cmake中变量未声明可以直接使用，最常用的例子是：为未定义的列表变量名追加成员，这变量就相当于已声明并赋值了

}

{

find\_package(Boost COMPONENTS system thread)

----用来检查某些库是否存在

----上面是检查Boost,组件中的系统和线程两个模块是否存在

----只有包含了某些包，某些包中的内置变量（如某些内置的目录名）才能使用

----所以使用这个函数的作用是：确保某个模块是否存在，以免后面报错时不方便定位

----另一个好处是引入包中的某些内置变量

}

Cmake

----cmake详解一已经看了

----正在看cmake详解二

<http://www.cnblogs.com/coderfenghc/archive/2012/06/17/2552899.html>

cmake

----debug

----release

## Dir

### PORJECT\_SOURCE\_DIR & PROJECT\_BINARY\_DIR

#the PORJECT\_SOURCE\_DIR in \*\*\*.cmake

cmake -D XIAOI\_PLATFORM=local ../cmake/install\_protobuf/

#then ${PROJECT\_SOURCE\_DIR} is the path equal to ../cmake/install\_protobuf/

#then ${PROJECT\_BINARY\_DIR} is the current dir

#actually where the CMakeLists.txt is, then where project\_source\_dir is

### CMAKE\_CURRENT\_BINARY\_DIR

#CMAKE\_CURRENT\_BINARY\_DIR in CMakeLists.txt

message\_info(" CMAKE\_CURRENRT\_BINARY\_DIR = ${CMAKE\_CURRENT\_BINARY\_DIR}")

#the value is the dir in which exec the cmake cmd

### CMAKE\_CURRENT\_SOURCE\_DIR

Is the dir CMakeLists.txt existed

### CMAKE\_CURRENT\_LIST\_FILE

#The inline variable that represent the CMakeLists.txt file

${CMAKE\_CURRENT\_LIST\_FILE}

#include the path and file

## Include

### common

include(./xiaoi\_common.cmake)

#like c, must specify the path

### buildin

include (CMakeParseArguments)

#it's need something to parse arguments of cmake

### c-header file

set(JSONCPP\_INCLUDE\_PATH "${CMAKE\_SOURCE\_DIR}/vendors/jsoncpp-src-0.5.0/include" CACHE PATH "jsoncpp include path")

include\_directories(${JSONCPP\_INCLUDE\_PATH})

#it’s a dir, include c-header files

## Target exec library

### Create a exec file

add\_executable(a.out main.cpp)

#create a "a.out" in build

### So file creation

add\_library(jsoncpp ${SRC\_FILES})

#argv[0] is so\_file name, without lib and so fix

#the argv[1] is the source used to create the target

#in the end, we get a library , not a exec file

add\_library(xiaoi-pre-static STATIC ${PRE\_SOURCES} ${PRE\_SOURCES\_C})

add\_library(xiaoi-pre-shared SHARED ${PRE\_SOURCES} ${PRE\_SOURCES\_C})

#this target will be a library

#use STATIC or SHARED to decides which kind of target it is

### target property

set\_target\_properties(xiaoi-pre-static PROPERTIES OUTPUT\_NAME xiaoi-pre CLEAN\_DIRECT\_OUTPUT 1)

set\_target\_properties(xiaoi-pre-shared PROPERTIES OUTPUT\_NAME xiaoi-pre CLEAN\_DIRECT\_OUTPUT 1)

#the keyword CLEAN\_DIRECT\_OUTPUT can make sure that two kind of library

# CLEAN\_DIRECT\_OUTPUT to decide existed in the same place

### Project

project(example)

#it will import two buildin variable example\_BINARY\_DIR and example\_SOURCE\_DIR

#it’s equal to PROJECT\_BINARY\_DIR and  PROJECT\_SOURCE\_DIR

#seems it has no active usage

### add\_custom\_command and add\_custom\_target

set(TEST\_FILE "log.txt")

add\_custom\_target(Test1 ALL DEPENDS ${TEST\_FILE})

add\_custom\_command(OUTPUT ${TEST\_FILE}

COMMAND echo "Generating log.txt file..."

COMMAND ${CMAKE\_COMMAND} -E copy ${CMAKE\_CURRENT\_LIST\_FILE} ${TEST\_FILE}

COMMENT "This is a test"

)

add\_custom\_command(TARGET Test1

PRE\_BUILD

COMMAND echo "executing a fake command"

COMMENT "This command will be executed before building target Test1"

)

#add\_custom\_command (use OUTPUT) to create a target

(use TARGET) to exec a command before other target is created

#add\_custom\_target used to creat a fake target

#${CMAKE\_COMMAND} –E equal to /usr/local/bin/cmake –E

#this method maybe more uniform

### Compile arguments

add\_compile\_options(-g3)

#just like gcc -g3 in cmd

#define something in coding way

add\_definitions(-D\_\_LINUX\_ALSA\_\_)

add\_definitions(-D\_\_MACOSX\_CORE\_\_)

add\_definitions(-D\_\_WINDOWS\_WASAPI\_\_)

#maybe it's the same with the set\_compile\_option

### installation

file(GLOB INCLUDE\_FILES dir1/\*.h)

install(FILES ${INCLUDE\_FILES} DESTINATION /home/my-ubuntu/Test/Tcmake/test1/dir2)

#copy .h file in dir1 to dir2

#dir2 cannot be relative dir

#dir1 could using relative dir

install(TARGETS my\_a.out DESTINATION /home/my-ubuntu/Test/Tcmake/test1/dir4)

#only we exec make install that it can be execulate

### Link dir

link\_directories("$ENV{DXSDK\_DIR}Lib/x86")

#set a path to linker

#it must be a absolute path

#we can also link many paths in one step

### Compile config file

#copy and convert file in CMakeLists.txt

configure\_file("CMAKE\_CURRENT\_SOURCE\_DIR/config.h.in" "${CMAKE\_CURRENT\_SOURCE\_DIR}/config.h")

#the ${} will be replaced by it's value

#${CMAKE\_CURRENT\_SOURCE\_DIR} is the dir in which cmakelists.txt existed

### Set Install\_prefix

execute\_process(COMMAND ./configure --prefix=${INSTALL\_PREFIX}

WORKING\_DIRECTORY ${LIBS\_PATH}/../tmp/${unzip\_dir\_name})

#the ${INSTALL\_PREFIX} used to be /usr/local

## Syntax

### Variable definition

#varible define in CMakeLists.txt

if(NOT INSTALL\_PREFIX)

set(INSTALL\_PREFIX "/usr/local")

endif()

#use set

### Define function in cmake

function(message\_error msg)

message(FATAL\_ERROR "${msg}")

return()

endfunction()

#endfunction() is needed

#return with bracket

function(get\_func RESULT)

    set(${RESULT} "Hello Function" PARENT\_SCOPE)

endfunction()

get\_func(V1)

message(${V1})

# return in function

# output is Hello Function

### If

if(NOT XIAOI\_PLATFORM)

message\_error("please define the XIAOI\_PLATFORM value.")

return()

endif()

#endif() needed

### If\_else

if(CROSS\_COMPILE\_FOR\_NAO)

...

else()

...

endif(CROSS\_COMPILE\_FOR\_NAO)

#comments in the endif

### Foreach

#foreach in \*\*\*.cmake

foreach(file IN LISTS local\_SRC\_FILES)

if(SRC\_FILES\_STRING\_LOCAL STREQUAL "")

set(SRC\_FILES\_STRING\_LOCAL ${file})

else()

set(SRC\_FILES\_STRING\_LOCAL "${SRC\_FILES\_STRING\_LOCAL} \\\n${file}")

endif()

endforeach()

#LISTS is a keyword

#need endforeach

foreach(loop\_var IN LISTS PRE\_SOURCES)

message(${loop\_var})

endforeach(loop\_var)

# loop\_var don't need to be defined in the before

foreach(ITEM IN LISTS VISION\_INCLUDE\_FILES)

list(REMOVE\_ITEM INCLUDE\_FILES ${ITEM})

endforeach()

#list in foreach

#this will remove all item which existed in VISION\_INCLUDE\_FILES from list INCLUDE\_FILE

### Call function

message\_error("please define the XIAOI\_PLATFORM value.")

#don't need semicolon

### Macro

macro(PRINT\_THIS\_IS\_A\_MACRO)

message\_info(" this is a macro")

endmacro(PRINT\_THIS\_IS\_A\_MACRO)

PRINT\_THIS\_IS\_A\_MACRO()

#this is a simple macro example

#call macro needed bracket

macro(get\_macro RESULT)

    set(${RESULT} "Hello Macro")

endmacro()

get\_macro(V2)

message(${V2})

#return in macro

#output is Hello Macro

### Option

message("in before my\_option, MY\_OPTION = ${MY\_OPTION}")

option(MY\_OPTION "this is my option" OFF)

if(MY\_OPTION)

message("in if my\_option1, MY\_OPTION1 = ${MY\_OPTION}")

else()

message("in else, MY\_OPTION = ${MY\_OPTION}")

endif(MY\_OPTION)

#every time we need to delete the build dir which exec cmake

#otherwise it will have effect on the next time to cmake ..

#if we don't define the MY\_OPTION use -D in cmd, it will behave as the set in

#code, just as the default value, before the code of option, it will be none

#if we defined the MY\_OPTION use -D in cmd

#if the value is not OFF, it will has the effect of ON, and will

#enter if branch, but the value of the option will be as the one in cmd

#if the value is OFF, it will enter the else branch

### And

if(NOT $ENV{BOOST\_INCLUDE\_PATH} STREQUAL "" AND NOT $ENV{BOOST\_LIBRARIES\_PATH} STREQUAL "")

# no bracket after keyword NOT

## Others in cmake

### Run cmake

cmake -D [XIAOI\_PLATFORM=local] ..

#need bracket after –D

### Exec a sub cmake file

ADD\_SUBDIRECTORY(src)

#we can see, it don't need to exec cmake again

### Help in cmake

Cmake –help-xxxxx xxxx

-commands

-command find\_library

-command-list

-command-list | grep find

-variable-list | grep CMAKE | grep HOST

-property-list | grep NAME

-property OUT\_NAME

http://www.cppblog.com/skyscribe/archive/2009/12/14/103208.aspx

cmake --help-commands

cmake --help-command find\_library

cmake --help-command-list

cmake --help-command-list | grep find

cmake --help-variable-list | grep CMAKE | grep HOST

CMAKE\_HOST\_APPLE

CMAKE\_HOST\_SYSTEM

CMAKE\_HOST\_SYSTEM\_NAME

CMAKE\_HOST\_SYSTEM\_PROCESSOR

CMAKE\_HOST\_SYSTEM\_VERSION

CMAKE\_HOST\_UNIX

CMAKE\_HOST\_WIN32

cmake --help-property-list | grep NAME

GENERATOR\_FILE\_NAME

IMPORTED\_SONAME

IMPORTED\_SONAME\_<CONFIG>

INSTALL\_NAME\_DIR

OUTPUT\_NAME

VS\_SCC\_PROJECTNAME

cmake --help-property OUTPUT\_NAME

### Version

cmake\_minimum\_required(VERSION 2.8.8)

### Shell

execute\_process(COMMAND test -e ${file\_name}

WORKING\_DIRECTORY ${LIBS\_PATH}

RESULT\_VARIABLE SHELL\_RET)

#COMMAND to specify shell cmd

#WORKING\_DIRECTORY to specify directory

#RESULT\_VARIABLE to specify the return value

### Color log

message("[${Blue}Info${ColourReset}] ${Yellow}${msg}${ColourReset}")

#the color and ColourReset is a varible seted by man

## Set env define in cmake

### Set ENV

SET(ENV{var\_name} value)

#using set

if(DEFINED ENV{MY\_ENV\_VAL})

message\_info("find MY\_ENV\_VAL = $ENV{MY\_ENV\_VAL}")

else()

message("MY\_ENV\_VAL " "is not defined")

endif()

#use $ENV{} to get the value

#no $ when see if defined

if(DEFINED ENV{NAO\_2\_13\_CROSS\_TOOLCHAIN\_PATH})

message(find NAO toolchain.)

set(INSTALL\_PREFIX $ENV{NAO\_2\_13\_CROSS\_TOOLCHAIN\_PATH}/xiaoi\_libs)

else()

message(FATAL\_ERROR "can't find NAO\_2\_13\_CROSS\_TOOLCHAIN\_PATH")

endif()

#use DEFINED ENV{}, don’t need to have $ symbol

### Get ENV

copy\_reuslts\_to\_sysroot("nao" $ENV{NAO\_2\_13\_CROSS\_TOOLCHAIN\_PATH}/cross/i686-aldebaran-linux-gnu/sysroot/usr)

#$ENV{var\_name}

### Set

set(CMAKE\_BUILD\_TYPE "Debug")

if(NOT CMAKE\_BUILD\_TYPE)

set(CMAKE\_BUILD\_TYPE "Release")

message("default build,type is Release.")

endif()

set(INSTALL\_PREFIX /usr/local)

# str value could without ""

set(VAR\_NAME TRUE CACHE BOOL "this is the val name")

message\_info("VAR\_NAME = ${VAR\_NAME}")

#the output is TRUE

#don't know what's the CACHE BOOL mean

#additional knowndge of set in cmakelists.txt

set(WIN32\_SRC\_FILES ABCDEF GHIJKL)

#then the value of WIN32\_SRC\_FILES is ABCDEF;GHIJKL

### Append path

#set path varible in CMakeLists.txt

SET(CMAKE\_MODULE\_PATH ${PROJECT\_SOURCE\_DIR}/cmake/modules ${CMAKE\_MODULE\_PATH})

#this will append a path to group

#for example, in the first,${CMAKE\_MODULE\_PATH} is "path1;path2;path3", then after

#${CMAKE\_MODULE\_PATH} is "path4;path1;path2;path3"

### File command

file(GLOB\_RECURSE PRE\_SOURCES CMAKE\_SOURCE\_DIR \*.cpp | \*.c)

message\_info(" PRE\_SOURCES = ${PRE\_SOURCES}"

#this can find \*.cpp and \*.c recursely

### Include path in cmake

set(JSONCPP\_INCLUDE\_PATH

"${CMAKE\_SOURCE\_DIR}/vendors/jsoncpp-src-0.5.0/include"

CACHE PATH "jsoncpp include path")

include\_directories(${JSONCPP\_INCLUDE\_PATH})

## Cmp

### Strcmp

if(${XIAOI\_PLATFORM} STREQUAL "local")

string(COMPARE NOTEQUAL $ENV{DXSDK\_DIR} "" DXSDK\_DIR\_FOUND)

#if it's not equal ,then the DXSDK\_DIR\_FOUND is TRUE

### Int cmp

if(${SHELL\_RET} EQUAL 0)

#no prefix INT

### match

if(file\_name MATCHES "nettle")

execute\_process(COMMAND make -f peter\_Makefile all

WORKING\_DIRECTORY ${LIBS\_PATH}/../tmp/${unzip\_dir\_name})

endif()

#use keyword MATCHES

### Str match and replace

#find and replace of string in CMakeLists.txt

set(JSONCPP\_UNIX\_FORMAT\_BIN\_PATH "a/b/c/d/e")

string(REPLACE "/" "\\" JSONCPP\_WINDOWS\_FORMAT\_BIN\_PATH ${JSONCPP\_UNIX\_FORMAT\_BIN\_PATH})

message\_info(" JSONCPP\_WINDOWS\_FORMAT\_BIN\_PATH = ${JSONCPP\_WINDOWS\_FORMAT\_BIN\_PATH}")

#the out put is a\b\c\d\e

#as we can see ,a new var will be created

# #libxiaoi\_cmake

g++

libcurl4-gnutls-dev --make libxiaoi

dependence packages when opencv installed

libbz2-dev --for boost

libasound2-dev --for make libxiaoi

speex --for make libxiaoi

./configure

make

sudo make install

qt5-default libqt5svg5-dev qtcreator

--for examples/linux/QtCamera/build

--use sudo apt-get install

--then install qt, and see /etc/profile

qtmultimedia5-dev

--for run examples/linux/QtCamera/build

libgtest-dev

--for make tests/face\_reg

--sudo apt-get install

--{

cd /usr/src/gtest

sudo cmake -E make\_directory build #创建一个gtest的编译目录

sudo cmake -E chdir build cmake .. >> /dev/null #创建并生成Makefile文件

sudo cmake --build build >> /dev/null #编译

ls build/libgtest\*

build/libgtest.a build/libgtest\_main.a

sudo cp build/libgtest\* /usr/local/lib/

sudo rm -rf build

}

liblog4cplus-dev

--for make tests/testlogger

# #shell

### Comment many lines in shell

#commended contents

:<<'

commented contends

'

### Ungzip to another dir

#new usage of tar

tar zxvf ${LIBS\_PATH}/../tmp/${file\_name} -C ${LIBS\_PATH}/../tmp

#will get a tar package in the directory which -C specified

### xz

xz -dk ant\_1.9.3.orig.tar.xz

#unzip the file and keep the original

### If a dir or file existed

#if or not a dir or file is existed

test –d File

test –e File

#the first used to test file

#the second used to test dir

### Apt-get source

apt-get source cmake

#it will download cmake package to the current dir

sudo apt-cache search cmake

#search packages which have “cmake keywords”

apt-cache search package 搜索包

apt-cache show package 获取包的相关信息，如说明、大小、版本等

sudo apt-get install package 安装包

sudo apt-get install package - - reinstall 重新安装包

sudo apt-get -f install 修复安装"-f = ——fix-missing"

sudo apt-get remove package 删除包

sudo apt-get remove package - - purge 删除包，包括删除配置文件等

sudo apt-get update 更新源

sudo apt-get upgrade 更新已安装的包

sudo apt-get dist-upgrade 升级系统

sudo apt-get dselect-upgrade 使用 dselect 升级

apt-cache depends package 了解使用依赖

apt-cache rdepends package 是查看该包被哪些包依赖

sudo apt-get build-dep package 安装相关的编译环境

apt-get source package 下载该包的源代码

sudo apt-get clean && sudo apt-get autoclean 清理无用的包

sudo apt-get check 检查是否有损坏的依赖

### Ubuntu 设置时区

sudo tzconfig

----如没有，使用：dpkg-reconfigure tzdata

----然后：sudo cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

### Ssh & scp

ssh –l xiaoi 192.168.1.108

scp /home/my-ubuntu/mProject/goddess\_robot.tar.gz xiaoi@192.168.1.108:/home/xiaoi/xiaoi/projects

sudo reboot

----重启

sudo shutdown –h now

----立即关机

yawime ALL=(ALL:ALL) NOPASSWD: /sbin/shutdown, /sbin/halt, /sbin/reboot

----/etc/sudoers文件中加上这句后，sudo关机时就不需要密码了

# #jni

#compile c source file to a dynamic library

gcc -fPIC -shared HelloWorld.c -I/usr/lib/jvm/java-7-openjdk-amd64/include -o libHelloWorld.so

#use -fPIC, not -shared

#need to give the path of jni.h, which is used above with -I/usr/...

#compile java: javac HelloWorld.java

#create c header file: javah -jni HelloWorld

#to run, using cmd: java -Djava.library.path="./" <class name>

export LD\_LIBRARY\_PATH="./"

export LD\_LIBRARY\_PATH=${LD\_LIBRARY\_PATH}:/home/my-ubuntu/Downloads/soundtouch/source/soSoundStretch/build

#g++ -fPIC -shared SoundCvt.cpp -I/usr/lib/jvm/java-7-openjdk-amd64/include

#-I/home/my-ubuntu/Downloads/soundtouch/source/soSoundStretch

#-I/usr/local/include/soundtouch

#-L/home/my-ubuntu/Downloads/soundtouch/source/soSoundStretch/build

#-lxisdcnv

#-o libsoundcvt.so

#that's mean we could use many

#don't need the header of Java\_\*\*\* in cpp file while to implement the jni header file

//JNIEXPORT jlong JNICALL Java\_SoundCvt\_XiaoiSoundConvert

JNIEXPORT jlong JNICALL XiaoiSoundConvert

#the second one is correct

#extern "C" {}

#don't need to using extern "C" {} again in .cpp file while it's declared in .h file

SoundCvt.cpp: In function ‘jlong XiaoiSoundConvert(JNIEnv\*, jobject, jbyteArray, jbyteArray, jlong)’:

SoundCvt.cpp:7:79: error: declaration of C function ‘jlong XiaoiSoundConvert(JNIEnv\*, jobject, jbyteArray, jbyteArray, jlong)’ conflicts with

(JNIEnv \*env, jobject m\_this, jbyteArray bufIn, jbyteArray bufOut, jlong len)

#java ide

jetbrains idea

# #auto\_compile

Sudo sometimes need to input passwd many times

Their already a CMakeLists.txt in the top of dir

Need to get all the dependiences packages

Need to judge if a package is already installed

#sudo apt-get install build-essential qt5-default libvtk6-dev zlib1g-dev libjpeg-dev libwebp-dev libpng-dev libtiff5-dev libjasper-dev libopenexr-dev libgdal-dev libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev yasm libopencore-amrnb-dev libopencore-amrwb-dev libv4l-dev libxine2-dev libtbb-dev libeigen3-dev python-dev python-tk python-numpy python3-dev python3-tk python3-numpy ant default-jdk doxygen

Libpng need to install autoconf first

Then run: autoreconf

./configure

Make

Make install

libjpeg-dev

----still not find

build-essential

----used for build Debian packages, that’s mean in redhat, this package don’t needed

qt5-default

----can’t find the package

libvtk6-dev

----can’ find it

Libavcodec-dev

----can’t find it

# Serial

echo "hello, I'm yawime" > /dev/ttyS1

----client

cat < /dev/ttyS1

----server

----output “hello, I’m yawime”

----we can turn it

<http://blog.163.com/baosongliang@126/blog/static/194935702013613514729/>

----串口通讯 select

dmesg | grep ttys\* ----查找新所插入的串口究竟是哪个ttyS

sudo vim /etc/udev/rules.d/90-serials.rules

KERNEL=="ttyS[0-9]",NAME="%k",GROUP="tty",MODE="0666"

KERNEL=="ttyUSB[0-9]",NAME="%k",GROUP="tty",MODE="0666"

----串口没权限使用时，需要进行的操作

# 穿山甲机器人项目

get/response ----firmware version 0x02 / 0x40 0x00

get/response ----motor driver status 0x02 / 0x40 0x10

sync ----wheel speed / time 0x06 0x20

sync ----stop active 0x06 0x30

sensor event / Body board sync ----stop passive \* 0x07 / 0x83 0x30

set/response event ----position transmission 0x03 / 0x47 0xA0

set/response event ----obstacle distance transmission 0x03 / 0x47 0xA3

set/response event ----battery level 0x03 / 0x47 0xA1

set/response event ----ping 0x03 / 0x47 0xA2

sync ----shutdown active 0x06 0xB0

sensor event / Body board sync ----shutdown passive \* 0x07 / 0x83 0xB0

与人这端的接口

Firmware version 0x30 version

Motor driver status 0x31 status

Wheel speed / distance

Wheel stop active 0x32

Wheel stop passive ask 0x33

Shutdown active

Shutdown passive ask 0x34

Socket 过来消息的分析函数：analysisSocketMsg

----路由问题

----既有发往串口的

----也有重新发往socket的

----变量问题

----被动停止、被动关机都是用一个私有变量来保存是否请求过，以免没有请求却出现上层主动允许停止或关机的情况

----处理组合消息时，不需要对rp 进行加解锁吧？

----主要是读出来以后，放到一些分散的临时变量中，觉得需要对这些临时变量进行加锁处理

----对于这种锁，用互斥锁就可以了，因为不是一读一写的，这里可能是一对多的关系

停止

----如果是主动停止运动，怎检查当前是否在运动，当前不是运动状态，则不往board发送停止指令

----被动停止，即使不运动，也会往板子发送允许停止或禁止停止的应答

不需要写socket线程的原因

----因为写消息的message是触发机制的，也就是说消息一出现，在对应的情况下消息就已经发送了，不需要单独再开一个线程来处理这件事

为什么需要监听的线程

----如果不用监听线程，那万一人这端稍久一点没连上去，那么程序将一直阻塞在那

----但也许程序就应该是这样的

为什么需要读套接字线程

----因为读写套接字都是用一个成员类封装的，所以必须开一个线程不停的从这个成员类中取消息

为什么需要读串口线程

----因为我们必须不停的读，所以必须设一个线程

----凡是程序中有两个需要不停的做的任务，那么可能就要使用线程了

为什么需要写串口线程

----这也是message触发机制， 在对应需要往串口发送消息的情况下才会发送，理论上这里也是可以不使用线程的

----在这里只是程序的一种设计方式，方便管理而已

----但是在这里也会造成复杂的锁管理

----其实这里还好，这里带来的锁并不复杂，主要的锁管理的复杂因素是中间多个的临时变量

----后面优化的时候可能需要把这个线程去掉

所以最终可能需要的线程只有一个读socket的线程， 一个读串口的线程， 后面可能会省去一个监听socket的线程， 一个写串口的线程

读写串口对应的buff和读socket对应的buff也都好管理，锁机制不复杂

主要就是那些中间变量的管理

### 机器人转弯：

前提假设：

----转弯时间段内，任一轮子都不能出先速度增加的情况

----应尽量保持外圈轮子速度不变

----总是在最晚的时刻转弯，也就是说转弯不能出现反复，只能有一个圆弧

----两轮子的速度方向时刻要保持一致，且要保持大于0

----不考虑轮子漂移的情况

----转弯时应保证轴心线速度不变

情况分析：

----先确定最大最小转弯半径：

----最大转弯半径时，内圈轮子速度减小最少，这是理论上转弯半斤可以无穷大，因为内圈半斤可以无穷大

----最小转弯半斤时，内圈轮子速度减小最大，理论上转弯半斤可以无穷接近于两轮的间距

结论：转弯时转弯半斤可以通过设置两轮子的速度比而设定，这个速度比最好写到一个配置文件中

只能在静止时执行此命令，若是在机器人已经是处于运动状态，则不知道这个运动还能持续多久，因为停止的优先级高，所有再追加第二个指令没有意义

修改：

原来是保持线速度不变，这样的方式有一个缺点，那就是在原线速度最大情况下进行急转弯，会发生外轮线速度超过最大值的情况，所以这种情况必须避免

现在的方案：

保持外轮的速度与原线速度一致，这样就不会超过最大速度了

原来公式的基础：(v0 + v) / (v0 –v) = nr

现在公式的基础：v0 / (v0 –v) = nr

----上面公式中nr是两轮转弯半径之比，函数中的入参为实际的半径，所以外轮半径应该为实际半径，内轮半径应用它减去两轮间距

----一般函数传入一个实际半径后，先用它与两轮间距相比，得到一个比值n，然后两轮运转半径之比就是 n / n-1

----函数中实际输入的半径，指的是外轮的运转半径

### 停止运动

机器人前方探测到障碍物，需要判断当前是否是在向前运动，如果是，才停止

如果机器人当前正在原地旋转，或正在后退中，则不能停止

也就是说机器人探测到障碍物后，不能盲目的停止运动

停止的机制必须独立，不能依靠运动函数退出时的停止机制，因为有些运动函数并没有等待运动结束然后停止的机制，它只发送一个运动报文就结束了。但是在独立的前提下，也必须报出一个警告信息，要不然那些阻塞的运动就会一直等待时间到达，然后重新发送一个停止指令

如果机器人正在向后运动，后方发现障碍物，则也应该发出警报，并停止运动，也就是说，机器人发出障碍物警报是根据它的运动方向而定的。

### 距障碍物1m左右开始减速问题

这个大体上与停止运动相似，大意是需要先判断机器当前运行的方向：

假设机器当前正在向前运动，然后前方1m处有障碍物，判断当前速度，若大于阈值，则减速为阈值，没问题，在减速的过程中原来的运动指令可不必取消，但新设的这个速度可以设置在哪个保存速度的内存中

只有遇到停止情况时， 原指令才取消，即打断时间等待函数

假设机器正在向后运动，然后后方1m出有障碍物时，判断当前速度，使用另一个阈值，若大于这个阈值，则减速为阈值，同样，在减速过程中原运动指令不必退出

当机器正在原地旋转时，障碍物的情况当前大可不必理会

当机器正在向左转时，若发现前方或左前方1m处有障碍物时，这是该怎么减速

----这时的情况为：外内轮的速度可能相差很大，也许相差很小，也就是说要根据转弯半径和所设置的线速度而定

----若转弯半径小于0.8m，大可以不必理会

----若线速度小于阈值，也可不必理会

----若线速度大于阈值且转弯半径大于0.8m，这个时候就需要进行转速处理了，这时速度的减少量就要进行详系推敲了：

----若将两轮子的速度都设为一固定值，这显然是不合理的，因为这将会导致机器在转弯遇到障碍物，虽然减小了速度，但方向却变成了直线了的现象，这显然会看起来怪怪的。

----若值将外轮的速度减小为阈值，那么机器运转的方向就更不确定了，可见这种情况下，若将任一轮子的速度设为固定值，都是行不通的。

----有这么一种方法，那就是将两轮的速度减小相同的比例， 因为两轮的运转半径之与两轮的速度比相关，所以这样的话就不会改变机器的运转方向

----而且这么做也有一个优点，那就是与向前向后运动的减速处理情况相统一了，都通过将速度减小一定的比例来实现

结论：

----可以封装一个统一的减速函数，该函数传入的参数只有两轮子的当前速度，也可以不传参

----当然，也可以不传参，该速度可通过发送获取速度指令来获得，但是这样一来，时间会花费的太多，所以这一方案排除。

----不传参， 速度使用假速度，可能也只有这样了，因为获取真实速度是得不偿失的。

----在这种情况下，当获取到两轮的速度后，取得最大轮子的速度，将其与阈值比较，计算速度减小的比例，然后将此比例应用到另一轮子上

### 减速与停止的关系

如果机器已经减速了，那么后面怎么停止呢？怎么处理减速与停止的关系呢？

目前我设置的是：机器会在1m处减速，那么应该设在什么时候发送停止指令呢？

他们硬件指令的减速是发生在75cm处， 我软件层面在1m处减速后，他们就不应该在硬件层面再进行减速了，但是他们咋硬件层面还有一个急停的指令。

----所以，我在软件层面还是需要发送停止指令的，要不就每次必须得依赖他们硬件层面的急停。

----所以，我软件层面的急停放在40cm到50cm处比较合适，暂时先设置在50cm处吧。

步骤：软件减速后，一切操作就当什么都没发生，原先所有正在等待时间到达的指令仍然在继续。直到停止指令过来为止。所以我只要照原先的逻辑，在50cm左右发送停止指令就行了。

### 规划超声探障

首先超声信息保存在全局变量里

获得正前方 正后方障碍物距离， 可以封装成一个分析障碍物距离的函数，因为在别的地方，这个函数也可能用到，否则就只能多用几个全局变量了

发现封装成一个函数并不是好方法，因为后面运动并阻塞等待的情况就需要不停的调用这个函数，会严重影响程序的运行速度。

判断当前运动方向：若是向前，则分析前方障碍物距离；若是向后，则分析后方障碍物距离。

----如果发现距离太短，则首先停止运动，再设置某个全局变量

应该只需要一个方向的超声警报就可以了

----假设当前向前运动

----假设当前向后运动

----假设当前静止（运动前，调用一次函数检查障碍物距离）

----假设当前是旋转

新思路：

不运动，超声警报就不需要，直接跳过，只需要把原始数据放到内存中就可以了

第一次，超声危险，设置变量，保证不继续发第二个警报，那什么时候又可以发第二个警报了呢？

### 新指令过来旧指令撤销的问题

新指令过来旧指令必须撤销，要不会浪费处理器资源，还有会发生指令冲突的问题，比如前一个指令需要前行5秒，结果3秒后来了新指令，要再前行5秒，那么结果在第二次指令的第二秒就会停止，这是有问题的

解决方法是：一个是新指令过来后，都发一个停止命令，则原指令会break，新指令会不受干扰的执行，但有一个问题，就是，每每新指令过来时，机器人都会停一下，看起来就是卡顿，这不是一个好的解决方案

另一个解决方案就是，新指令过来后，不发停止指令，只发一个有新指令过来的标记，原指令发现有新指令过来后，break退出，新指令直接执行。这样一来，就不会出现卡顿的情况，但是也会有一些问题，比如原指令与新指令有冲突，这很有可能会损坏轮子或导致机器人摔倒。

相互冲突的指令可能会有：

原指令是两轮子向前，后指令是两轮子向后

原指令是向左旋转，后指令是向右旋转

原指令是向左转弯，后指令是向右转弯 ------这个可能并不会发生冲突

原指令是向左或向右转，后指令是向后退

原指令是自旋， 后指令是非自旋的运动

冲突规则就是，轮子不能突然向反方向旋转。

解决方案，对于每一种运动指令，都分配一个标记，再用一个全局变量标明上一个是什么样的运动指令，以判断是否与当前指令有冲突，若有冲突则先发一个停止指令，没有冲突则直接执行新的指令， 最后是写了一个函数，用来判断新老指令是否有冲突

### 红外问题

目前前方有远距、近距两个红外，远距探测距离为70-80cm， 近距探测为20cm，远距探测到障碍会减速，近距探测到障碍会刹车

目前在上位机层面：当只有向前运动并且红外探测到障碍时，才会发出对应警报

假如在任何情况下都发出警报，那么就只能在后续处理中对警报进行过滤了

----比如有一个等待函数，就要区分当前是前进还是后退，并进行相应的跳出操作

----现在还不知道哪种处理方式比较好，所以先用钱面一种

The system is going down for halt NOW!

\*\*\* Error in `./goddessrobot': double free or corruption (!prev): 0x08399170 \*\*\*

Aborted (core dumped)

xiaoi@xiaoi:~/xiaoi/projects/goddess\_robot/build$ Connection to 192.168.1.203 closed by remote host.

Connection to 192.168.1.203 closed.

### 关于电机状态新增一个刹车状态的问题

刹车状态，可用来获知当前底层的意图，但是上层的一些指令也会使刹车状态置1

底层的刹车意图上来后，会有一个通知过来

----通知过来后，用个变量保存这个消息

----如果组合消息

### 关于时间到却没有发刹车指令的问题

原因无非两种情况

----要么计算时间差的函数有问题

----要么时间到了，指令却没有发出去

----可能是被系统中其它的信号打断了

----可以多加日志，应该可以查出来

### 新架构思维

Socketcomm

----这个模块的start函数里面会放一个产生会话的循环，而构造函数里面会只放一个设置socket属性的函数

----有一个观点就是，start里面应该放起线程相关的函数，因为这是其他类似模块中常做的

----但是，这个socket模块是有所不同的，因为它要求用户退出后可以重新进入会话，也就必须有一个不断创建会话的循环

----在这里，前一个会话没有退出的时候，新的会话是不能够产生的，本来这是一个技术方面的短板，但是这个短板却恰好

满足了我们当前的需求

----当一个会话建立起来后，就会创建两个线程，一个用来读，一个用来写

----原本这个功能是要放在start里面的，现在这个功能要放到哪里去呢

----对于读线程

----可以另外新加一个成员函数，把循环放到这个函数中，并用boost：：thread将其作为一个线程来执行，这样似乎能满足要求

----对于写线程

----写线程好像是不需要循环的，因为有内容过来时，当下就写了，不需要一种类似守护过程的东西，所以，好像与串口模块一样

只有读线程，没有写线程。写线程仍然用一个通知代替，当有东西要写时，主动调用的地方调用被调用模块提供的一个通知接口

，通知被调用模块进行写操作。

----对于程序退出的问题

----还是统一用一个线程退出的成员变量标记

----这个标记同时作用于读线程和产生会话的循环

----理论上应该是读线程先意识到这个全局变量产生变化，然后退出，然后创建会话的循环再意识到这个变化，直接跳出创建

会话的循环，所以stop后另产生一个新的会话的这种担心是多余的。

----另外，以stop方式退出读线程与人这端主动废除会话是不一样的，前者会不再产生新的会话，后者只会废掉当前会话，但可以

继续产生新的会话

----所以，当前暂时先将上面写的这些实现先吧

----在实现过程中发现，pc与socket通讯的地方太多了，如果要进行发出信号这样的处理模式的改造，那么会增加很多行的代码，也会导致逻辑管理混乱

----现在的处理办法就是，要么封装一个函数，专门用来处理这种信号发送的事情，要么就是像单起一个线程一样，把发送任务放到一个线程中去

----如果把发送任务放到一个线程中去，那么整个函数模块就会因为这个东西而创建一个线程，其他的地方没有线程，这样看起来不是很好。

----其实可以参考下写串口是怎么实现的，这里写socket应该与写串口的处理机制是一样的，区别只有要用到的地方的多少而已

各个新函数接口

----原接口改成新接口应该不是很难吧，最简单的想法就是只要把函数名称修改一下，还有几个函数合并一下，还有函数传入的入参需要修改一下

----修改函数名称的情况

----当函数的入参全都一致时，应该只需要替换一下函数的名称就好了，但是函数入参全都一致的情况应该比较少

----将多个函数合并成一个的情况

----在这种情况下，应该会多出来一个入参，但是应该还是很好处理的，因为之前都已经封装了基本函数了，只要以某种巧妙的方法调用这个

基本函数就可以了

----修改参数单位的情况

----这个情况也应该会比较好处理，因为这只是一个纯计算的工作，也就是单位转换的计算

----设计成非阻塞函数

----听当时说好像是要用信号与槽的方式来实现，因为用回调函数的方法会有点繁琐，也就是说每一个函数都要有一个对应的回调函数，下面来分析一下

这其中的原因

----当用回调函数时，假如我要向前走5米

----那么回调函数的方式就是，先发一个指令，让机器人向前以某一速度运动，然后在函数内创建一个线程，这个线程的处理函数含有

一个参数，也就是回调函数，当线程快要退出时，执行这个回调函数。

当然，线程处理函数也可以不含参数，当线程快要退出时，都默认调用某一个全局函数或成员函数，也就是说，不用回调的方式实现回调的功能。

这种情况下，比如对于直走，需要调用一个回调函数，用于等待时间到达后主动发送停止指令， 也就是说，其实等待和发送停止指令可以整合

成一个通用的函数，也就是通用的线程处理函数，供各个指令调用，原来的情况下，只是对等待时间进行了封装。

----当用信号与槽时，比如同样是要求机器向前走5米

----那么当我发出向前走的指令后，并不创建一个线程，我只发出一个信号，这个信号通知某个槽，在多长时间后需要发送停止指令。

----这样看起来，代码似乎要优雅的多

----当机器前面遇到障碍物后，会接收到另外一个信号，在这种情况下做另外对应的处理。

----也就是说，一个函数可以注册多个信号。

----只所以要用注册信号而不用全局函数或最基本的信号量，是因为用全局函数需要不断的轮询，占用太多的处理器资源

用基本的信号量又会使程序卡在某个地方不动，这都不是好的办法。

----用注册信号的话，只要不退出对应的作用范围，信号一来，都会执行对应的处理

### 设置一个单位转换的类

----最底层的基本接口的公有函数的参数不用基本单位，用一个类。

----比如：原来长度为int longInMeter，现在替换为： iBotOS::IUnits long

----long 可以通过不同的构造函数得到，也就是说重载多个不同的构造函数

----然后获取long中的值的时候，可以提供多个get值的函数

待解决

旋转过程中没有限制最大速度， 已设置

### 运动指令发送后需返回信息

一共有三类运动指令，需要规划指令发送后，哪些信息需要返回

----按抽象方式：

指令id， 指令类型（一共就3类类型）， 两轮子速度，速度方向，等待时间

----按形象方式：

指令id，指令类型，

----直线走： 距离，实际所设速度， 实际将等待时间

目前比较急的一件事：

----自检程序

----还有目前程序退出，总会报一个崩溃的日志，这个问题必须要解决，要不然他们上层有一个类退出，造成整个应用崩溃，那就不好了

### 自检程序设计

自动测试

主要测试距离、角度控制，避障控制，硬件停止

----自检结果放到一个文件中

----文件类容为： 条件，结果， 只要结果匹配条件，就认为检测合格

----自检程序应该不受场地限制

----如果一个功能的当次测试条件没有达到，则应自动转一个角度，然后自动生成指令发送，直到条件不匹配的次数达到一定的数目为止

对于前进、后退检查项：

----打印原指令参数， 距离，速度

----打印函数结束时返回信息

----打印机器传输的传感器信息

----检测返回值信息与串感器信息是否匹配

----自检还是按照指令顺序来

----否则自检结果文件很难看，并且程序设计起来也很复杂

如果在测试一条指令时遇到电机状态错误，该怎么处理

----电机状态错误这种情况是不允许发生的，如果遇到这种情况，说明机器不正常，测试结束

如果在测试过程中发现指令发不出去，也就是说，在运动前就探测到前面有障碍物

----在这种情况下， 如果当前指令是向前运动

----那么可以试着旋转90度，然后再向前运动

----如果还是有障碍物，动不了，那么就再旋转90度

----如果还是动不了，那么再旋转90度

----也就是说，在测试一种指令时，那么可以使用的指令就只有该指令和旋转指令

如果在测试旋转时遇到障碍物

----在这种情况下就只有试试前进与后退

-----如果前进动不了，那么就试着后退

----如果旋转，前进，后退都因为障碍物而动不了，那么测试失败，需要人为移动位置后再测试

如果在测试转弯时遇到障碍物

----在这种情况下，可以同前进一样， 遇到障碍物时，逐个旋转90度

### 支持多实例化改造

现在的情况是：

首先类是一个普通的类，是非单例的

类的声明中有多个静态和非静态的变量

类的实现中有多个静态的全局变量

有些函数不能多次使用，比如启动服务，关闭服务

解决方法：

----\_isNeedQuitThread

----这个需要改造一下，不能因为一个类的实例发了一个停止指令，然后其它类的实例就不能工作了

#----改造方法就是：当一个类的实例析构时，先不真正析构，只是将类的引用计数减一，当发现当前类的引用计数已经是1的时候，就进行真正的析构

#----所以，在准备修改\_isNeedQuitThread这个值为true时，只有当发现当前类的引用计数为1时，才真正进行修改

#----所以构造函数也要进行一番修改：

#----每次构造，都将引用计数加1

#----启动服务函数也需要进行修改

#----如果发现当前服务已经为启动状态，那么就不要再启动了

#----这就需要增加一个记录当前启动状态的静态变量，这变量反正是bool型的，用不着加锁进行同步

经过反复思考，运动控制部分还是要声明明为单例

原因如下：

----为了很好的控制机器的运动，需要在类中建立很多中间的变量，以保存当前状态

----如果不是单列，那么这些状态就不好进行同步，除非将类中所有的成员变量都编程静态属性才行。

----但是静态变量有一个不好的点就是，在类中进行了声明后，还得在全局区进行一次初始化，这样看起来有点啰嗦，所以，还不如将类

----变成单例来的实在

----稍后会将只有变成单例才能解决问题的中间变量一一例举出来

中间变量的同步问题

----运动控制类中有许多中间变量，声明为单列已经保证了各个变量的唯一性，但是在多线程多个类中时，还是要考虑到线程的同步问题

----可以理解为，当某个东西是唯一的时候，就要考虑到同步的问题

----如果某种东西是拷贝型的，存在多个拷贝，这种状态是不需要考虑线程同步的，因为各线程都只在操作它自己的一份

----要解决这些同步问题，就需要加很多锁，这是不希望看到的。

----解决办法可能就是，尽量的减少中间变量的存在，只保留一份最原始的内存可能是一个解决办法

----也就是通过更多的计算来减少中间变量的数量

# 资源目录

https://git.oschina.net/jason-jiang/logsystem.git

kdiff3 ----目录级的比对

# 编程注释规则

所有星号对齐，注释内容从第二个星号开始，注释的内容与星号隔一个空格

最左边的斜杠与代码对齐

参数用@开头

参数与参数的解释各占一行

对文件名的注释

注明文件名

注明创建日期

注明作者

# Xcode

在项目中加一个playground

----只要拖一个控件到已有代码的目录树中就行

----右下角文件模板库中就有这个模板库

----这个模板库的名字叫playground

----有三种类型的模板，分别对应于ios osx ostv版本

----xcode会在代码同行的右边显示结果

最好常备几个课题，往深里死里研究，就不信不出成果

# Swift

### 闭包

闭包的表象形式是：一对花括号括起的一段代码

闭包可以有参数，也可以返回值，但是就是在定义时没有func关键字和函数名字，所以相当于一个匿名函数

闭包定义返回值的规则和函数定义返回值的规则是相同的

----闭包在参数和返回值定义完后会出现一个in关键字，这个关键字就是用来说明参数和返回值已近定义完了

----这些参数和返回值都是在下面的闭包中使用和生成。

闭包有一中简单形式，也就是单行表达式形式

----它可以省略参数的类型，当然也包括类型前的冒号

----它可以省略用于括住参数的括号，

----可以省略用于指向返回值类型的箭头，

----可以省略函数返回值类型

----可以省略返回函数值的return关键字

----总之，只需要排列参数，in 和返回值表达式就行了

----参数间以逗号隔开

----参数和in只需要空格

----in和返回值间只需要空格

闭包还有一种更简单的形式：

----它将参数和参数类型都省略了， 同时也将in省略，也就是说，只有一个花括号括住一个表达式

----一般这种情况下，参数常用$0 $1等之类的表示

最简单的闭包体现了闭包是函数的一种的本质

----String类中集成了字符串比较函数“>”， 因此”>”本身也就是一个闭包

闭包一般用来作为其它函数的参数

----比如上面最简单的闭包 “>”就是用来作为函数的参数了

----作为参数，是需要放在函数的括号里的

----但是trailling闭包可以放在函数小括号外

----也就是放在函数的花括号中

----如果函数只有这个闭包表达式一个参数，那么函数的小括号也可以省略了。

----这使得函数在调用时看起来像是函数在定义

闭包捕获

----说的是闭包可以捕获其所在的上下文中的产量 变量 和入参等。

----其实说的是，在定义函数时定义闭包，则这个闭包就可以使用函数的常量、变量和入参

----这个函数还可以单独拿出来使用

闭包是引用类型

如果将一个闭包赋值给两个变量，那么这两个变量将会指向同一个闭包

### 表达式

基本表达式

self

----相当于c++中的this

----用于对当前类型和类型实例的引用

----可以代表当前类型

----也可以代表当前类型的实例

----另外表达式后面也可以跟一个self

----如表达式.self， 表示返回表达式自身的值

----类型实例后跟一个self，表示返回目前实例所属的类型。

----还有另一种方法获取当前实例的类型

----那就是动态类型表达式

----格式为 “实例名.dynamicType”

----可以获得当前实例所属的类型名，并访问其中的方法

super

----作用与self类似，不过代表父类

----代表当前类或实例的父类或父实例

----也就是说通过它可以访问父类的成员和父类的方法

隐式成员表达式

----用于在可以推断出类型的环境下，引用这个类型的成员

----也就是说这个类型可以省略，直接使用它的成员

----使用方式是 点+成员

----如果这个类型或成员名写出来了，那么就叫显式成员表达式了

圆括号

----可用于创建元组

----在圆括号中，表达式可以加一个标签

----加标签的方式是：标签名加一个冒号

通配符表达式

----和正则表达式中的通配符不同

----这里用来忽略表达式中的某个参数

----如：(a,\_) = (1,2)

附属脚本表达式

----它是一种用来访问getter和setter函数的方法

----形式为:表达式1[index表达式2]

----中括号中的应该就是附属脚本表达式

----这里中括号中的index暂时还不知道是什么意思，它与getter setter函数有什么关联

强制取值表达式

----这里其实就是一个赋值语句，只不过对空加了一种预警机制

----形式：在值的后面加一个感叹号

----这里感叹号的意义就是表面，感叹号前面的这个东西它不能为空，为空了就会发生异常

可选链表达式

----可选类型变量或对象通过使用问号来声明

----当这个声明的对象不为nil时，就可以访问里面的方法或成员

----例如：var a = SomeClass?

Var a?.SomeFunc

----第一句是申明一个可选变量a， 它是使用可选表达式声明的

----第二句的var应该是可以去掉的，这里应该是印刷错误

----第二句应该是使用可选变量，在使用的过程中，又使用了可选表达式

----若一个可选表达式是另一个表达式的子表达式

----在这种情况下，只有最外层表达式返回的才是一个可选值

函数调用

----入参前面可包含标签

----格式是标签名+冒号

----有些函数可以没有括号

----如类型的初始化函数

### 函数

函数定义格式

----需要有关键字func

----入参中的参数类型放在参数名的后面，以冒号与参数名隔开

----返回值类型在括住参数的括号后面，被一个箭头指着

----然后花括号与c语言中的一样

----每个参数前面都可以加标签，标签与参数名以空格隔开

----标签的名字可以与参数的名字相同，这样就可以用#号代替，并且可以省掉中间的空格，但调用时还得按老规矩

----这样，该函数在调用时，也一样得加上标签，调用时标签名与参数名间以冒号隔开

返回值

----可以指定一个元组作为返回值

----这样就可以返回一个由多个值组成的元组

嵌套函数

----在swift中，函数可以嵌套

----可能只能在定义母函数时使用

### 控制流

If

----条件体不需要括号括住

----其他的与c语言一样

---- c一样，下面可以接else

----else if后的条件也不需要加括号

switch

----条件语句不需要加括号，其他的与c语言相同

----里面的case与c语言中的写法一样

----最后也是接default

----唯一的不同是，不需要在每个分支后面加一个break，因为语法会默认break出来

----也可以保持c的特性，不过要在case的结尾处加上fallthrough语句

----在这种规则下，每一个case后面至少需要接一条语句

----case后面可以接元组

----如果case已经包括所有可能，就不需要default。

----case中还支持值绑定，这样便可以使用该值

for in

----可以使用下划线作为变量，访问范围里的值，这是实际是只遍历次数，并不使用里面的值

for

----与switch一样，也是不需要括号的

----经常用到..< 和 …两个操作符

---- 1..<100 和 1…99两个表达式表示的范围是一样的

While 和 do - while

----条件也是不需要括号的

continue 和 break

----与c语言中的一样

标签语句

----可以在循环关键字之前加一个标签语句

----格式: label 标签名:正常循环语句

----这样，在break时，可在后面接一个标签变量

----这样就可以一次跳出多层循环

### 复杂数据类型

数组

----声明数组格式： [类型]()

----如var ermtyArray = [String] ()

----在声明一个有元素的数组时, 类型和括弧都可以省略

----是靠类型推断来知道类型的

----如var ExceptionTypes=[“none”, “warning”, “error”]

----可以用加法操作符来组合两个相同类型的数组

----新数组的数据类型可以从两个数组的数据类型中推断出来

----可以使用两个Array来声明一个二维数组

----如：var mutiArr=Array<Array<Int>>()

----swift中数组是值类型的，赋值和当做入参时，传的都是拷贝

字典

----用于存储无序的元素，可以存储任何类型的元素

----可以将数组作为字典的元素

----创建方式

----字典名称 = Dictionary<type,type>()

结构体

----有两种声明变量的方式

----第一种类似于c语言， 如：var instanceName:structType

----第二种看起来比c更方便

----var instanceName = structType(mem\_1:value\_1, mem\_2:value\_2 ….)

----结构体也是值类型的

枚举

----一种自定义的数据类型

----一个枚举定义相当于定义了一种新的数据类型，所以这个类型的首字母必须大写

----里面每一个成员前面都有case关键字

----不会默认0,1,2等值

----如果值类型都为int，就会有

----每一个没有被赋值的，都会分配一个自动递增的初始值

### 运算符

求余

----swift中可以对浮点数进行求余

----这是与c语言中不同的

字符连接符

----可以将两个字符串连接在一起成为一个新的字符串

=== 和 !==

----恒等运算符

----用来判断两个变量是否引用了同一个类实例

### 字符串

在swift中，字符串是可以相加的，就像在python中一样

### 打印相关

用println打印输出

# C & C++

typedef

----用typedef 声明别名时，用于变量和用于函数指针的方式是不一样得

----用于变量时：typedef 被define的变量 新产生的变量类型

----用于函数指针时： typedef 被define的变量

----不需要新产生的变量类型

----该变量类型默认为函数名

类对象初始化

两种简单的初始化方式，效果是一样得

----CTest TestInstance(20);

----CTest TestInstance = CTest(20);

C++ 中结构体对象可以相互赋值， 而在c中，编译会报错

编程风格

----varA =

varB =

varC = 1.0

----不用多个变量累加到一行了

## boost

bind

----http://www.cnblogs.com/sld666666/archive/2010/12/14/1905980.html

----上面网址是bind的基本用法

----总之bind网上有很多资料，需要找个时间整理一下

----http://blog.163.com/qianw\_ok/blog/static/4217590520084935050703/

----这个地方对bind工作原理进行了详细的介绍

Thread

----http://blog.chinaunix.net/uid-23093301-id-86385.html

----上面介绍了boost中的线程

### 关于boost中的锁

boost::mutex

----独占互斥类

----有lock和unlock方法

boost::shared\_mutex

----共享互斥类

----除lock和unlock方法，还有shared\_lock. Shared\_unlock方法

boost::unique\_lock<T>

----独占锁

----T可以为mutex或shard\_mutex

----会自动调用lock和unlock方法

----自动调用，暂不明白

boost::shared\_lock<T>

----共享锁，

----T只能为shared\_mutex

----自动调用shared\_lock和shared\_unlock方法

boost::mutex::scoped\_lock

----递归式的互斥量

----其实可以理解为含有某种范围的锁

----如：　　boost::mutex io\_mutex;  
　　 void foo( )  
　　 {  
        　　 {  
               　　 boost::mutex::scoped\_lock lock( io\_mutex );         /// 锁定  
        　　 } /// 解锁  
　　 }

----也就是说，将该锁声明为局部变量，当超出该局部变量的作用范围时，该锁自动解除

----但是有一个mutex要是某种全局的，要不然别的地方不知道你这个地方当前是锁住的

boost::unique\_lock<boost::timed\_mutex>

----设置锁超时

----如：boost::unique\_lock<boost::timed\_mutex> lk( io\_mutex , std::chrono::milliseconds(3) ); // 超时3秒

使用读写锁

boost::shared\_mutex rw\_mu //定义读写锁

boost::unique\_lock<boost::shared\_mutex> lock(rw\_mu); //写线程加唯一锁

boost::shared\_lock<boost::shared\_mutex> lock(rw\_mu); //读线程加共享锁

----在使用读写锁时，上面三行都要用到

----http://my.oschina.net/lingluonianhua/blog/217854

----上面网址是一个读写锁的简单例子

使用条件量

boost::mutex m\_mu;

boost::condition\_variable\_any m\_condPush;

----boost::mutex::scoped\_lock lock(m\_mu);

----首先在范围内加一个锁

----m\_condPush.wait(m\_mu);

----使用条件量可以在等待时暂时释放锁

----m\_condPush.notify\_one();

----使用通知，通知前面等待的条件为真，可以退出暂时等待

----实质上是相当于c中的信号量

----可以总结的就是，在全局需要有一个互斥锁

----http://my.oschina.net/lingluonianhua/blog/217854

----上面网址有一个条件量的简单例子

### Boost中的线程

线程群管理器

boost::thread\_group producer\_threads

producer\_threads.create\_thread(producer\_callback);

producer\_threads.join\_all();

----先创建组

----再用组创建线程

----再等待所有线程结束

----http://www.2cto.com/kf/201501/374244.html

----这是网上关于生产者和消费者的例子

----好像ios中也有这种线程组，也就是等待所有线程结束后再执行某种操作

### Boost中的一次实现

boost::once\_flag flag = BOOST\_ONCE\_INIT;

void thread()

{

boost::call\_once(&init, flag);

}

boost::thread thrd1(&thread);

boost::thread thrd2(&thread);

----这样之后，init函数就只会执行一次

----其实这种功能自己也可以通过算法实现

Boost中的图有关的封装

<http://blog.sina.com.cn/s/blog_69de213f0100l42p.html>

----上面网址简单介绍了boost中的图

{

boost::try\_to\_lock

这个方法返回 bool 型的值：如果能够获得互斥体则返回true，否则返回 false。

相比lock函数，try\_lock会立即返回，而且在获得互斥体之前不会被阻塞。

}

### C++内置函数索引

http://www.cplusplus.com/reference/array/array/

构造函数

返回值实际上市构造出来的对象，但在头文件和实现中都省略这个隐含的返回值，所以实现中不要再纠结有没有void了

一般函数

实现时返回值在类作用域名前面，说明类作用域名是修饰函数的，而不是修饰返回值的

带默认参数的函数

----声明时要加上默认参数，但在实现时可用可不用，反正只要在一个地方有说明就可以了

类的声明

----直接就是 class classname{ }; ， 没有别的什么关键字了

关于模板

----不支持分离编译，在包含模板头文件时，需要将对应实现文件也包含进来

宏定义规避长整形缺陷

// Patch for MinGW: on Win64 long is 32-bit

#ifdef \_WIN64

typedef unsigned long long ulongptr;

#else

typedef ulong ulongptr;

#endif

取消一个宏定义

#undef SOUNDTOUCH\_FLOAT\_SAMPLES

预防宏定义冲突

#ifdef SOUNDTOUCH\_FLOAT\_SAMPLES

// check that only one sample type is defined

#error "conflicting sample types defined"

单列、全局变量

在c++中，单列无论是静态成员还是非静态成员，多次实例化后，都只维持一份

非单列情况下，静态成员只有一份，非静态成员有多份

无论单列还是非单列，在cpp的实现文件中的全局变量都只有一份，也就是说，共多个类的实例共用

这是非常容易忽略的，也是致命的

c++ c中定义函数的区别

----在c中，函数参数的名字不能缺省，也就是说，在定义时，不能只有参数的类型

----在声明时当然可以只有参数类型

----在c++中，函数定义时也可以只有参数类型，如果该参数没有使用的话，就可以只用该参数的类型，不需要该参数的名字

Double型的最大最小正数值宏

----在<float.h>中定义了浮点类型的范围：  
----#define DBL\_MAX 1.7976931348623158e+308 /\* max value \*/  
----#define DBL\_MIN 2.2250738585072014e-308 /\* min positive value \*/

构造函数

----在c++中，结构体也可以写对应的构造函数，也有对应的冒号初始化方法

----这样的话，以后对结构体进行初始化就方便多了

### 单例的创建规则

----构造函数必须是私有的

----但是析构函数必须是公有的

----有一个静态的指针变量，指向一个该类的实例

----该指针也最好是私有的，以免别的地方将它乱指

----有一个静态函数，用来创建获取一个该类的实例

-----静态函数虽然不能使用类的非静态成员函数，但是却可以使用该类的够造函数

----想必构造函数也有某种静态属性

-----这样，类中的静态函数相比全局的静态函数就有了一个优势，那就是：类中的静态函数可以访问类的构造函数，而全局区的静态函数不能

----可以说，也正时有了类中的静态函数的这个特点，才有了单例的这一说法

----赋值操作符、括号操作符需要在类的私有区进行覆盖一下

----确保只能通过提供的那个静态函数来获取类的实例

----也就是说将其它构造实例的通道封闭

----目前发现在私有区覆盖括号操作符的时候，编译会报错，这方便后面还得花时间研究一下

# Automake

----http://blog.csdn.net/fd315063004/article/details/7785504

----automake 这个工具必须掌握，要不然没法参考第三方源代码

----<http://blog.csdn.net/shanshanpt/article/details/17200035>

----网上一个简单解释automake 与autoconfig一起生成makefile的例子

### Makefile生成顺序

Autoscan

----需要在源文件夹下执行这个命令

----将生成configure.scan文件

----之后将这个文件重命名为：configure.ac

----需要自己编辑修改这个文件

----这 个文件主要用来检测相关环境

----这个文件写好后，就该写一层一层的makefile.am了

aclocal -I .

----然后执行这个命令

----执行这个命令时需要带参数，目前还不知道要带什么参数，所以就指定当前目录为参数了

AC\_OUTPUT

----configure.ac 中的最后一行的这个是用来指定最终在哪些目录中生成makefile

# Qt

## Qml

----也有引入头文件的说法

----如：import QtQuick 2.0

----如果不引入这些头文件，里面有些关键字可能就不会识别

----http://www.cnblogs.com/csulennon/p/4485768.html

----这是一个更简单的列子及说明

----http://blog.csdn.net/foruok/article/details/32698603

----这是qml与c++混合编程的例子

----http://wenku.baidu.com/link?url=peMPaxLDdJI2qv5py61Y\_\_yEYaypFeBr5ZdBsNZs-9hYbxCFmXKYvBE3hCI6MqmyQTTFEySggXeWHC-y36NM9rE-q8mkj5NeuVbDb6DVH6W

----这是一个qml学习漫画，讲了一些特殊的数据类型

----http://blog.chinaunix.net/uid-374124-id-4661152.html

----比较详细的介绍，着重于属性特性方面的介绍

----里面也有信号处理方面的介绍

----<http://www.cnblogs.com/xiangxiaodong/archive/2011/07/30/2121691.html>

----qml学习文档，里面都是一些小的功能例子

### Qml理解

Item

----QtQuick模块中的Item可以用来编写视图中的父子关系

----他是qml中大多数对象中的基类型

----rectangle和text的基类都是item

----在这种对象中，parent值指的是它的视图父亲 ，而不是对象树中的父亲

----一个对象的视图父亲是可以修改的，但是对象树父亲是修改不了的

----其他对象只是对象树中的某种父子关系

Component

----组件类型通过加载才能使用

----加载方式

----Loader { sourceComponent: 组件id}

----在加载的过程中好像也可以指定x，y轴的值，也就是在视图中的位置

----Loader { sourceComponent: redSquare; x: 20 }

Id

----每一个qml对象都有一个id特性

----这个id特性有qml语言提供，无法重定义和覆盖

----小写字母或下划线开头，由字母，数字，下划线组成

----对象.id这种用法是不合法的

属性特性

----必须以下滑线或小写字母开头

----关键字default是可选的

----声明一个自定义属性的同时，系统会隐式的创建一个“值改变信号”和一个关联信号的处理器“onChanged”

----除枚举类型外， 所有基本类型都可以作为自定义属性的类型

----qml中的对象类型也可以作为属性的类型

基本类型

----有些基本类型是由QtQuick中的一些模块提供的，使用前需要将该模块导入

----var类型是通用类型，能够保存包括list和object在内的所有类型的值

----也就是说，其它类型的值都可以赋给var类型的变量

类型安全

----一个属性仅能够接受与它相同类型的值

----有些属性的类型没有自然值的表现形式，这是qml引擎会自动的执行字符串到该类型值的转换

特殊属性类型

----对象列表属性

----一个对象列表值是一个由中括号括起来，逗号分隔对象的列表

----如果列表中仅有一个对象，那么中括号可以省略

属性的别名

----别名仅能引用它所在名字空间中的对象或对象属性

----不能包含JavaScript

----别名只能在他所处对象全部实例化完后才能使用

只读属性

----只读属性值能在初始化时赋值

## 常用基本语句

(QMouseEvent \*event)

----鼠标事件参数用法如下：

----实现了判断是左键还是右键

----实现了接受事件（虽然还不知道这个功能用来做什么）

if (event->buttons() & Qt::LeftButton) {

move(event->globalPos() - dragPosition);

event->accept();

}

QPainter painter(this)

painter.rotate(6.0);

----旋转，应该是逆时针

----单位应该是度

painter.drawLine(92, 0, 96, 0);

----应该是画一条线段

槽

----类里面的槽函数多是声明为静态函数

# Soundtouch

# Valgrind

<http://www.cnblogs.com/wangkangluo1/archive/2011/07/20/2111273.html>

----这个网址写的比较详细，有时间可以看看

valgrind --tool=memcheck --leak-check=yes ./GodMotion\_Example

----检查是否有内存泄漏

----still reachable 指的是内存指针还在，还有机会使用或者释放

valgrind --tool=memcheck --leak-check=full ./a.out

----可以检查内存泄漏和数组越界

valgrind --tool=helgrind ./a.out

----用来检查是否有线程死锁

----也用来检查是否有竞争（race， 由于没有保证线程同步造成的，也能够定位到行数）

----能够定位到函数中含有lock的那一行（在编译时要加上-g才能够指定行数）

valgrind --tool=callgrind ./sec\_infod

----用来分析程序的性能，以及制约程序性能的瓶颈

----结束程序后，会在当前路径下生成callgrind.out.pid文件

----callgrind\_annotate --auto=yes callgrind.out.19689   >log

----将上面产生的文件转化成可读文件log

----进而可以用vim打开log进行查看，分析新能

其他参数

--log-file=filename将输出的信息写入到filename.PID的文件里

--log-file exactly=filename 这样明确指定，则输出文件没有pid后缀

# Gdb

----gdb加程序的运行方式

----或gdb先进去后，再用file指定可执行文件

----l

----列出文件

-----一般只会列出包含main函数的文件，显示行号

----break n

----在文件的第n行设置一个断点

----r

----正式运行程序

----自己想要的设置项设置好后再执行这个操作，要不后面就不能再设置了

----n

----一般在断点处停下后，就可以一行行的执行了

----c

-----如果又不想单步执行了，可以输入这个命令，一般可以执行到下一个断点

----p i

----打印变量i的值

----可能只有断点到当前行时才能够使用

<http://blog.csdn.net/haoel/article/details/2879>

----上面的总结个网址

# Google

<http://laod.cn/hosts/2016-google-hosts.html>

----2016 Google hosts 持续更新

----谷歌搜索host

# 室内导航

----导航的开始函数实现的功能是这样的：

----首先，可通过传感器获得当前的位置，也就是二维坐标

----可找出离当前位置最近的设定点

----计算出当前位置与最近设定点的距离

----如果过大>1.2，则将它当做一个目标，并使机器正对该目标

----过程是将一个函数连接一个信号，这个函数的作用就是始终使方向正对该目标

----当目标距离小于一定值<0.8的时候，再断开与该信号的连接

----如果障碍物距离太小，则返回，不进行旋转操作

----如果足够小，则就当当前已经在那个设置点上了，通过最短路径算法找出下一个点，并走上面那一套路径（距离过大的时，将它作为一个目标）

----一个特殊情况

----如果已经到达最终的点，则停下，否则再算出下一个点

----这函数的功能好像只能控制转向，并不能控制走多远

----也许这个模块的功能只是控制转向

----

----statecontrol

----应该这里是提供机器人向前走得功能

----这里是一个单列

----这里调用所有的运动函数

----首先，state类里面有一个statecontrol指针

----然后statecontrol里面有四个state指针

----其中三个是三个状态

----一个是一个变量，可以存放前面三个状态中的一个

----使用statecontrol控制运动时，要调用当前状态指针中的Event方法，该方法实际上是调用本类私有robot变量的公有成员函数实现的

----总感觉这里有点绕

----目前还不知道这样做有什么好处

伪陀螺

----这个类最终就是用来维护一个二维角度值的

----当要实现某种复杂的算法的时候还是新建一个类来写靠谱些

\_onStateChanged

----每改变一次状态，陀螺里面的角度传感器数据更新一次

----这个函数由别的类里面的onRunCommand触发

----每次执行指令时，statecontrol里面的onRunCommand将会发出一个信号供伪陀螺这边的\_onStateChanged处理

----如果当前是旋转或停止状态，要将点列表中的状态清空，因为这些点是用来拟合直线的，所以只能收集一次直线行走的点

----改变状态后，如果是旋转，就需要更新传感器的角度

\_onLocaltionChanged

----这个函数里面最后发出了角度改变的信号，这个信号触发了自身函数的再次调用

----但什么地方是第一次触发这个函数的，目前还没有找到

问题1

----导航和避障两者都会用到控制机器旋转，两者会不会发生冲突

问题2

----避障每次都向前走1m，会不会发生走过头的情况

问题3

----避障走直线，导航旋转，两者之间会不会发生冲突

----也就是说，会不会发生时刻都在旋转的问题

----这个不会，因为导航的旋转要在机器与固定方向形成一定夹角后才会旋转

# 神经网络算法

{

神经网络算法中间的参数究竟怎么确定？

方法一：

继续在网上搜文档

好像所有的神经网络算法都没有告诉怎么确定中间层各个参数是怎样生成的

好像找到了权值的计算方法了：下面网址文档中的第49页

----http://max.book118.com/html/2015/1123/29977580.shtm

BP神经网络算法：

----神经网络训练是根据对训练样本的学习来调整网络连接权值

----初始时刻，各节点的的联接权值是杂乱无章的

---- ----只能通过

----当一个测试样本通过网络，网络的权值就进行了一次调整

----于是将第二个样本输入到前一次调整过后的网络，又可以得到第二个实际输出值和期望值之间的误差

----继而继续调整网络，通过不断输入训练样本，网络权值也在不停的调整

----当输入任何一个样本，网络的输出误差都控制在一个期望的范畴字中之时，表示网络训练完成。

BP算法推导

----采用误差法相调节方式调价网络

算法推导过程：这研究论文还整U币好确定是什么问题：只能进行下一不分析：

首先算出一个期望值，

-----这应该是第一层里面可定有很多子：

----第二个参数是一个去去求和的值的一半，好像没有指定求和的

http://www.tuicool.com/articles/ba63eqB

----这个网页是讲MLP的

方法二：

继续猜

方法三：

通过看代码是否有文档、注释来确定

}

{

http://www.cnblogs.com/heaad/archive/2011/03/07/1976443.html

----神经网络算法简介

}

{

神经网络算法

设置输入数据：

----只是设置第0层的神经元的值，不是设置它的权值

----第0层也没有权值

GetOutputSignal

----取的是最后一层神经元的各x的值

Simulate

----触发函数

----读满第一层神经元数量的input数据和最后一层神经元数量的target数据后，才调用一次函数

PropagateSignal

----计算当前层各神经元的x值

----也就是产生一个下层神经元的输入信号

----所以根据函数名：繁殖信号是可以见名知意的

----上层各神经元x的值与本层特定一个神经元的加权值列表相乘，获得一个结果

----将该结果乘以以个负的系数[-dGain]作为一个自然指数，再将该值加1取倒数

----上面方法得到的值就是当层某神经元的x值

ComputeOutputError

----好像最终是用来计算两个标准差的，也就是dMSE和dMAE

----将传入的目标值数组中的对应元素与当层对应神经元的x值做差, 也就是获得距离目标的差值

----对所有神经元的该差值的平方求和再取平均数就是dMSE

----对该差值取绝对值再求平均数就是dMAE

----各神经元还有一个误差属性:e, 它的计算方法是：

----该差值再与对应神经元的x值的某一表达式：(x \* (1-x))相乘

----将获得的结果再乘以一个系数：dGain

----这个e实际上是用来做反馈的，也就是向后繁殖时会用到

BackPropagateError

----当层某个神经元的差值：e 等于下层各神经元差值与当层神经元权值列表的对应元素的乘积之和

----不对，得出的这个结果后还要乘以一个x表达式系数和一个dGain系数才是

----也就是说，反向反馈错误的时候，可能实际上计算的是这个e值

AdjustWeight

----先获得上层神经元对应的x

----再获得本层神经元对应的e

----再获得本层神经元的dw元素

----那么本层的w元素就等于上层的x乘以本层的e 在乘以以个系数：dEta

----还要再加上本层的dx 乘以一个系数：dw

----需要再重新计算本层的dw

----等于上层的x乘以本层的e再乘以一个系数：dEta

Run

----Run函数循环调用训练和测试函数：Train Test， 训练和测试函数循环调用触发函数：Simulate

----每一次触发函数都要

----设置输入信号

----繁殖信号

----获得输出信号

----计算输出误差

如果是处于训练状态，则需要

----向后繁殖错误值

----调整权重

}

# opencv中的坐标

{

opencv中x y轴方向问题

http://www.cnblogs.com/tornadomeet/archive/2012/12/12/2813939.html

}

# c++命名规则

{

命名规则：

http://www.cnblogs.com/ggjucheng/archive/2011/12/15/2289291.html

类的私有成员变量：以下划线开头

类的公有成员变量：以m加下划线开头

}

# 远程登录ubuntu图形界面

{

http://jingyan.baidu.com/article/8ebacdf0cdc64949f75cd555.html

----远程登录ubuntu图形界面

}

Ubuntu中的串口助手

{

cutecom

----ubuntu下的串口调试助手

----apt-get就可以安装了

----直接敲名字: cutecom就可以运行了

}

# 锁

{

http://zsxxsz.iteye.com/blog/2028452

----线程池中的惊群现象

}

{

GodUltraSensor.cpp中的116行，需要补充一下，还有这个地方可能需要加一个锁

}

{

GodMotion

电量信息结构体是否需要加锁问题 g\_stBattery

电量信息现在在运动控制部分还没用到，所以这个问题以后再说

真实速度记录及控制是否需要加锁问题 g\_stSpeedMsg

----这里是短整型，单个速度是原子操作，但是两个速度一起，可能会造成获得的两个轮子的速度不一样

----所以还是有需要同步一下的

----但是这里目前看起来还不是致命问题，是小概率事件

假速度记录及读取问题：m\_curWheelSpeed[2]

----现在用的是互斥锁，必要时需要换成读写锁

超声数据：m\_ultrasonicStatus

}

# socket通讯方面

{

----select 是用来检查是否有可以读的

----对于写socket，是不需要select的，直接写对应的文件描述符就可以了

----短连接似乎有以个弊端，那就是服务端貌似不能主动往客户端发起通讯，因为服务端根本就不知道客户端的地址信息

----所以对于那种需要长时间才能应答的东西就不能采用短连接

}

{

确定报文是否发往对方

确定报文是否传到下位机的最好确认办法并不是在最后发送之前打印一行日志

----而是采用抓取报文的方式，比如：myshock----可能拼错了。

----这个包应该最好是从文系统中产生的，要不然就不好办了

}

# boost

## boost中得字符串处理

{

http://blog.csdn.net/huang\_xw/article/details/8267500

----boost处理字符串相关举例

}

{

boost::to\_upper(str)

boost::to\_upper\_copy(str)

----将字符串所有字母变成大写

----前者，会改变原字符串

----后者，不会改变原字符串，会新生成一个字符串用于返回

}

## boost中的signal

{

收到一个消息后，就开辟一个线程去触发信号，等待别的程序处理完再删除线程，感觉这样是一个解决办法

----这样就不需要用全局变量然后再加一堆的锁了

----实际上这也用到了全局变量，只不过这个全局变量是用来管理线程的

----还有，这个全局变量是一个类的对象，这个类是自己定制并实现的

----所以也是需要加锁的只不过这个锁可以加在这个类自己里面，从而实现了它的封装

----所以现在悟出的一个规则就是，锁最好加在类的实现中，不要在调用它的上层类中去加锁，因为那样更复杂并且不好管理

}

{

signal作用范围

----signal触发回调函数，如果注册的地方在局部时，触发地方当超出局部作用范围时，是否还能触发回调函数

----结果证明超出作用范围后也能触发，所以后面写signal有关的函数时就需要注意了，要注意必要时需要断开连接

}

connect

----signal need to connect to a function

----when connected to, should specify param placeholder(that means we are using a bind)

----常将connect这个语句封装一下，成为一个函数

----这函数接收的参数为一个函数，参数可用bind一下，参数可以只有占位符

----函数参数的个数可以与信号发出的参数的个数不相等，但bind占位符的个数需要与它相等

----signal具体是boost::signal2::signal<type of callback func> signalName

----绑定的是一个函数的地址，不是函数

----when signal stricked, function were called

# 可重入函数

{

怎样写出可重入的函数

----对于使用单个的全局或成员变量

----封装一下对该变量的读，写，在封装中加入锁机制，可以用互斥锁，但最好用读写锁

----对于一个全局数组或成员数组：

----看样子也只能像上面一样，不能细分到数组中的元素去加锁，因为若是这样，就会需要加很多个锁

----所以全局或成员变量都应封装一下

----所以对于类的成员变量，都应该加一个get函数和set函数

----所以应该尽量少用全局变量，因为全局变量不方便像类一样加get和set函数，主要是容易混淆

----到时候总结了锁机制再来修改这些小类吧，现在先把功能实现先

}

## boost动态、静态编译问题

boost

----static

# 单例

{

http://www.cnblogs.com/weixliu/p/3900764.html

----单列，第一次创建时加锁

----单例一般最好配合shared\_ptr使用

}

# service

{

服务

----现在还是找不到一个方便的方法来实现一个服务，也许所有的东西真的需要自己去实现吧，所以首先做得就值 能事一点一滴的积累资料了

http://www.2cto.com/kf/201305/209861.html

----代理模式

http://www.cnblogs.com/marchtea/archive/2011/12/04/2275194.html

----这里似乎也介绍了代理，但是与上面介绍的好像不是一个东西

http://blog.csdn.net/lcl\_data/article/details/8989420

----各种代理简介

http://www.ibm.com/developerworks/cn/webservices/ws-soa-capisca1/

----c++构建服务

http://www.ibm.com/developerworks/cn/webservices/ws-soa-capisca2/index.html

----这是后续第二部分章节

http://blog.csdn.net/wowo1109/article/details/7226777

----安卓与c++创建简单services的例子

http://www.educity.cn/develop/479726.html

----用VC++建立Service服务应用程序

----这个比较简单，可以详细看看

http://www.xuebuyuan.com/1335726.html

----这也是一个，可以仔细看看

http://www.2cto.com/os/201204/127211.html

----这是脚本方面的，说不定到时候也要参考下

http://biancheng.dnbcw.info/255/255143.html

----后台服务程序学习笔记

http://www.cnblogs.com/niocai/archive/2012/07/12/2587736.html

----提示在linux什么目录下配置什么东西，也就是程序好了该怎么配置

----没有说明别的程序怎么调用这个服务程序中的接口

----搜：linux 将一个 程序 变成 一个 服务

http://www.vckbase.com/index.php/wv/1391

----vc 中 createservice

搜：

一个 services 多个进程 共用

写一个 linux service 程序

}

{

将GodMotion变成一个服务，是需要将它放在后台运行就行了，但是想要调用服务里面的接口，好像是没有现成的，看样子是需要自己实现的

----模式是：客户端进程通过include一个共用的接口模块（这个模块是不含有main函数的）与service进行通讯

----再由这个共用的接口模块与services与service进行通讯

----可见，这个共用的接口模块是有多份拷贝的，但是service只有一个

----也就是说接口与services间需要加一个协议

----协议就用函数与各参数名之间以下划线分隔就好了，参数也不用参数名标记了

----问题是，service返回函数调用结果时，怎么知道该返回到哪个客户端

----通讯机制先考虑短连接吧，其实用消息队列也是可以的，不过用短连接拓展性会好一点

----如果在报文中不加识别的话，看样子是没办法知道该返回到哪一个客户端的

----现在在报文中加一个ip和端口作为识别码

----那么现在的问题就是，在services中怎么保持这个识别码不混淆

----进程内motionid是唯一的，只要在入口处将motionid与ip、端口绑定，在出口处再通过motionid查找ip信息就可以处理一来一回的任务了

----那么对于signal这种机制在进程间怎么实现呢？

----如果通过广播，那么在刚开始时，广播也不知道有多少个对象需要广播

----接口模块刚连接上来时，统一需要加一个注册函数，也就是相当于握一次手

----接口模块需要断开时，统一再握一次手，也就是相当于解除注册，使服务下次广播时少一个对象

----幸好之前那个有socket的godmotion版本还没有完全删掉

----对于一个进程间共享的服务，必须实现一个代理功能，拿到这个代理就像拿到这个服务本身一样

----代理应该也是通过某种继承方式来实现的，因为只有继承才能拿到该类的函数

----所以最好是继承某种抽象类

----感觉可以搞一个用来代理的代理程序，由代理程序分配资源，如端口什么的

----但是各个客户端程序怎么协调，还是需要一个统一的东西，如文件什么的，但是这样又不太安全

----还是靠程序自己去获得端口什么的吧

----还有一个东西需要保证，那就是在每个客户端进程当中，代理的东西要弄成一个单例，要不然发消息到GodMotion时，有可能会被认为是两个客户端

}

服务的自己实现

{

对于轮询的群组消息，怎么将它依次发送到每个对应的会话当中去？

----对于这种需求，一种自然而然的想法就是各个会话自己过来取这个轮询的报文信息

----但是这是个socket，没法取，因为它只能发送报文，没有自动读取的功能

----另一种方法就是主动推送，把所有正在会话的id全找出来，一个一个的去推送消息

----但是这种方法也有一定的局限性，因为有些会话它不需要这个信号。

----也就是说有些会话比如我的伪终端是不需要这些群组消息的，发过来就会用不着，只会在终端上打印出来而已。

----当然也可以控制不在屏幕上面打印，加个条件屏蔽一下就可以了

需要新建一个类用来管理会话sessionId

----这个类可以在socketcomm这个类里面声明一个对象

现在该考虑群组消息传送给各客户端（也就是应用软件）时该采用什么样的格式

----其它指令的响应报文的格式是：cmd\_id + 对应信息， 现在还没有报文长度信息，校验码什么的，如果会出问题，后面还得加上

----现在决定群发指令的报文格式与一般指令保持一致，只不过cmd用一个固定值，因为前面预留了10个cmdid

运动状态应该反馈些什么东西呢？

----首先，cmdid是需要返回的，否则请求和应答报文就没法子对应上，这是进程间通讯的基础，没这个东西报文就没法对应上

----然后就跟一个buff就行了，根据不同的指令，去解释这个buff是什么意思

----还得另有一个标记，用来表示这个返回的指令是个什么类型的指令，根据这个标记就可以知道怎么解释报文了

----在实际实现层面

----对于立马返回的指令，sessionid可从入参中得到，cmdid是从报文中来的，motionid是返回值，所以这三个就直接知道了

----对于运动指令的反馈，因为前面生成了一个唯一的motionId，可根据motionId去找另外两个id

----对于群发指令，因为是群发的，sessionid不需要知道，motionid没有，可以写一个固定值，cmdid也是没有的，也可以写一个固定值

用一个类来存储motionId cmdId sessionId 这三者之间的关系

----在什么时候插入这个idmap呢？

----当运动指令就插入返回值非负时，就插入一个值

----究竟什么时候删除idMap里面的某个元素呢？

----现在的问题是：若是函数返回值为非负值就插入了一个id元素了，那么删除该选在什么时候呢？

----应该在运动被停止或打断时，也就是结束时再删掉，本应该是这样的

----那么有没有运动还没开始就被打断的呢？ 也就是没有运动结束这个信号

----理论上是不能出现这种情况的，即使出现了也要想办法避免

---------------------这个地方到时候还是要查一下#############################################

}

# Git

{

http://git.oschina.net/yeahren/libxiaoi\_cmake/issues/6

----交叉编译，为nao编译boost的时候出现问题，LIBS\_PATH未定义，实际上定义的是RESOURCE\_PATH，已经修改

----网址需记

----这样以后也可以提bug

}

# 常用简单技巧

{

sign = x>0 ? 1 : (x==0 ? 0 : -1)

}