Release:2019-4-23

都是些知识点和功能，方便复制粘贴，解释性的东西很少...

jiyanglin@qq.com

第一章 C/C++

1.1. C

1.1.1. 基本数据类型

类型	符号
char	%c
unsigned char (0～255)	%c
short	%hd
unsigned short	%hu
int	%d
unsigned int	%u
long	%ld
unsigned long	%lu
long long	%lld
float	%f
double	%lf

整数格式：

%x 十六进制 %#x, %#X 带符号 uint64 对应%llx

%o 八进制 %#o 带符号 010 表示数字 8

%nd 位宽为 n，高位无数时空格 uint64 对应%lld

%0nd　　位宽为 n,高位无数时补 0

%nd 右对齐

%-nd 左对齐

浮点数格式：

%.nf　保留小数点后 n 位【四舍五入】(.nf 不能用于输入)

%e　　%E 代表科学计数法格式

%g　　%G　不显示无意义的 0，根据数据自动选择使用科学计数法

%p　地址(指针)格式

%zd　size_t 输出类型

%%代表%

Unicode 字符

wchar_t　c='x';

wchar_t* p=L"abcde";

isalpha(c)可以判断字符是否是字母（不区分大小写）

==========字面值常量

十进制：　　20

八进制：　　024

十六进制：　0x14

字面值整数常量类型默认为 int 或 long（其精度类型决定于字面值大小）。通过增加后缀，可以将其转换为某种精度。

长整形：1L　(long)　　8UL　(unsigned　long)

浮点型：3.14f　3.14L（浮点型默认是 double，后面加 f 代表单精度型，后面加 L 表示扩展精度型）

```
#include <inttypes.h>
uint16_t val = 12;
printf("val = %" PRId16 "\n",val);
```

```
#include <float.h>
FLT_DIG    float 支持的有效位个数
FLT_MIN    float 最小值
```

&取地址　　*从地址间接访问

“字符　“”字符串

转译字符：　\n , \t, \', \”

左值：可改变的值

右值：不可改变的值

（左值）a=5（右值）

0 代表假，其它数代表真

1.1.2. 运算符

不同数据类型进行计算时，以数据精度高的为主。

如(int)x/(float)y 结果为 float 型

1.1.2.1. 算术

+,-,*,/,%,++,--

++a 先加后使用

a++ 先使用后加

+=，-=，*=，/=

a+=1 等价于 a=a+1；

1.1.2.2. 关系

>,<,==,>=,<=,!=

1.1.2.3. 逻辑

&&（与） 有短路特性，前一个表达式为假，则不进行第二个表达式计算

||（或） 前一个为真，则不进行第二个表达式计算

!(非)

1.1.2.4. 位运算

~按位求反　一个二进制数

&按位与

取出某位(判断某位是否存在)　if(a&x)

1001　　1001

0010　得到 0　　0001　　得到 1

删除某位　　a&=~x;

1001　（删除末尾 1）使用 1 的反码 1110

1110　　得到 1000

| 按位或　合成数　　a|=x;

1010

0101　得到 1111

^ 异或　　不同时为 1 时得到 1　　a^=x;

1001　　1010

0011　得到 1010　　0011　得到 1001

1.1.2.5. 移位

移位运算的本质是进行了乘法操作，向左移动一位，就是乘一次 2。

<< 左移

>> 右移　　二进制

Exp：（9）1001<<2 得到 100100（36）

1001>>2 得到 0010（2）

1.1.2.6. sizeof

sizeof(int)　类型必须加括号

sizeof a　　变量可以不加括号

1.1.2.7. 三目运算符

a>b?a:b

1.1.3. 语句结构

if()…… 如果后面多条语句为一个整体，用{}括起来

if()……else……

if()……else if()……else　else 与最近的 if 配对

for(i=0;i<n;i++){……} 满足条件时，执行语句，然后 i++

break; 跳出循环

continue; 取消本次循环，进行下一次循环

while(i<10){……i++;……}

do(……)while();当语句中有 continue，则会停止下面语句，去执行 while()。

switch（整型）

{

case　常量 1：……；break;　不加 break 则会在满足 case 条件后执行下面的语句

case　常量 2：……；break;

default : ……;

}

switch 语句内部不能定义变量，需要使用变量只能在之前定义。

如果 switch 内要定义变量，需要使用块语句。

case 1:{　int a = 10; break; }

并行，语句之间相互顺序不影响

循环中使用 goto 可直接跳出循环

goto　p;　可直接跳到 p 的位置

p:　.....;

\ 可以进行折行

"abc\

def"

1.1.4. 函数

int ab （int a，int * p）{…… return x；}

返回值类型 函数名 形参 返回值

返回值为 void 类型时，只能用 return；不可返回值。

主函数默认返回类型为 int，可不写返回值类型和 return，主函数的参数为 argc，argv[],分别为命令行的字符串数量和这些字符串的指针组成的数组。

函数实现前至少要声明(若位置在调用前，则可不声明),声明后位置可任意放,声明时不可省略参数类型，可省略形参名

‘,’用来隔开参数,‘;’用来隔开语句，‘{}’用来划分语句块

函数是以形参的初始化开始的，形参就是在该函数中定义的局部变量，靠调用时传入的实参进行初始化。(传入参数为 int aa，函数内产生变量 int a = aa；传入为 int *pp，函数内产生变量 int *p = pp)，在一片新的内存空间运行，结束时释放这片内存空间。所以在这片空间

内运行的普通形参不能改变外部实参。在使用指针时，形参为指针，通过指针的*运算访问到了外部实参进行操作，所以可以改变外部实参。

1.1.5. 变量

声明变量时不分配空间

使用变量时，是把某块内存中存储的数字按照变量类型进行解释，如 int 就是直接把 4 字节的内存上存储的数拿出来用。

不能返回自动变量的地址，自动变量在函数中使用完后会自动释放

全局变量未初始化是 0,局部变量未初始化是随机数

auto int x = 10;

修饰符 数据类型 变量名 初始值

auto：自动变量,普通局部变量都是 auto,一般省略

static：被限制访问范围的全局变量。在全局区定义则只能在定义位置的 cpp 文件中访问（使用 extern 不能访问）。在函数中定义则只能在该函数内使用。在类内定义则只能在类及类的对象中访问。

register：被频繁使用的变量，建议编译器将其保存到寄存器中

volatile：变量可能会被未知的因素改变，建议编译器不对这个变量进行优化

使用场合：多进程，多线程，共享的数据

硬件寄存器

一个中断服务子程序中使用的非自动变量

const int a=10;　　// a 为常量，不可改变，只能在初始化时候赋值。

要在不同文件中访问全局变量：

int　x=10; //定义变量的文件

extern　int　x;// 告诉编译器，该变量在其他文件中定义

对于 const 变量

extern const int x=10;　//定义变量的文件

extern const int x;

在头文件中定义变量时，只能是 static 或者 const

1.1.6. 数组

1.1.6.1. 定义

a[]={…}　　a[]初始化后会跟据数据数量自动分配内存

a[10]

a[][10]={…}

a[10][10]

int a[n];使用时下标从 0 开始到 n-1 一共 n 个元素

sizeof(a)／sizeof(int) 可计算出数组中元素的个数,在函数调用中，退化为指针，无法计算出数组元素个数。

1.1.6.2. 初始化

==========一维数组

a[10]={1,2,3}　　数量不足时补 0

a[10]={[8]=5,[5]=3}　　不需要考虑顺序

a[10]={0}　　定义时进行清零

使用 memset(a,0,sizeof(a))也可将数组清零

==========二维数组

a[m][n]={{n 个数据},{…}…}

1.1.6.3. 数组名

一维数组：a+3 等价于 a[3]的指针

二维数组：a+3 等价于 a[3][0]的指针，*(*(a+i)+j)相当于 a[i][j]

数组名相当于一个地址，并不真实存在，二维数组名是一个特殊的二级指针(行指针)

数组名不可以使用++运算，因为数组名是常量

可将数组名当做一个指针传给函数

int a[10];

int *p=a 时,p 移动 1 位为 int 大小，即为数组下一元素的位置

(&a+1)　　&a 移动一位的大小为 sizeof(a)的大小

a 与&a 值相同，意义不一样

p=a 或者 p=&a[0],p 可以像 a 方式一样使用

1.1.7. 指针

指针存储的是内存地址，而内存地址是一个数字，所以*p 等价于 *0xffffff00（32 位系统的指针）

int a;　　int *p;

a 等价于*p, p 为一个 int 型的指针, *p 作为一个整体时为 int 型

可以使用 typedef　int*　INT　则 INT　p ;　p 为指针

typedef int　a;　(a 指代 int 类型)　int a;（a 表示变量）

指针移动一位与指针类型相关

int　a=10;

char　*p=&a; 编译会出现警告

p+1 移动一个字节(char)

在 32 位系统中，指针占四个字节，其中存放着指向的变量的地址值

指针包含:

自身所在的地址值

自身变量名

存储的变量的地址值

指针的类型

指向的数据的类型

void *p

通用指针，不可移动或取值，可赋值给任意类型的指针。c 中可直接赋值，c++中需要类型转换

int *p

一级指针 p+1 代表移动指针类型一块(一个单位)的大小，本质为一级指针中所存地址值发生变化。对*p 操作即为对 p 指向的变量的操作

int **pp

二级指针存着一级指针的地址,pp+1 代表移动四个字节(32 位系统中指针四个字节，在一级指针连续存储时+1 才有意义)。因为指向的是一级指针,对*pp 操作即为对 pp 指向的一级指针进行操作.对**p 操作即为对 pp 指向的一级指针指向的变量进行操作

pp 类型是 int **(int 型指针的指针),指向 int *。

int a = 10;

int *p = NULL;

int **pp = &p;

*pp = &a;

int *p[3]

指针数组，数组中元素为指针

```
char*  val1[100] = {0};  等价于
char** val2 = (int**)malloc(sizeof(int*)*100);
```

int (*p)[3]

数组指针，p 指向数组,p+1 代表移动数组大小

int *p(int)

函数，返回值为 int*,参数类型为 int

int (*p)(int)

函数指针，返回值 int，带一个参数 int

函数指针使用：

void fun1(int a,char b){cout<<a<<endl;}

void (*)(int,char) 是函数指针的类型

void (*p)(int,char) 是定义函数指针对象p

====方式 1

void (*p)(int,char) = NULL;//定义一个函数指针p，初始化为NULL

p = fun1;//指向函数

p(2,'a');//调用

====方式2

typedef void (*pfun)(int,char); //重定义类型

pfun p = NULL;

p = fun1;

p(2,'a');

const int *p 或者 int const *p 常量的指针

const 在*p 前，*p 为一个 int 型的值，表示 p 指向的值不可改变（不可通过 p 指针修改变量）

int *const p； 常量指针（形容指针是常量，不能改变）

const 在 p 前，表示 p 不可移动，指向的值可变（p 的内存地址值不可变）

void getMemory(char *p){ p=(char*)malloc(10);}

该函数不能给 p 分配内存，而且会造成内存泄漏。因为编译器要为每

个参数制作临时副本，p 的副本_p(_p 指向 p)，修改_p 指向的内容，也就是 p 指向的内容被修改。但是，为_p 分配内存相当于让_p 的指向另一片内存，而 p 指向的地方不变。

在函数中改变指针的指向无效。

在函数中分配的正确方法为：

1） void getMemoryA(char **p){*p=(char*)malloc(10);}

2） char *getMemoryB(){ return (char*)malloc(10);}

调用时：

char *p = NULL;

getMemoryA(&p); 或 p = getMemoryB();

野指针：指向垃圾内存，没有指向 null 的指针。

产生原因：1）指针变量没有被初始化为 null；2）free 内存后没有指向 null；3）指针操作超过了变量生存周期。

1.1.8. 字符串

字符串由一组连续字符变量表示

有三种表现形式

1）“abcd”，字面值常量，存于只读常量区

2）char a[]，数值可改变，数组字符串的本质是末尾带’\0’的数组

3）指针字符串，可指向字面值，也可以指向数组字符串

连续两个字面值可以合并成一个

printf(“%p”,”abc””de”);

输出“abcde”的首地址

char　*p = “abc”; //字面值字符串，不可变，长度 4

char　a[] = ”abc”；//数组字符串（末尾自动加’\0’），可变，长度 4

char　b[4] = {‘a’,’b’,’c’}；//数组字符串，可变，长度 4

char　c[] = {‘a’,’b’,’c’};//字符数组，可变，长度 3

char *p=”xxx”;　(字面值）

“xxx”为字面值（代表字母首地址），不可改变，只能用 p=”yyy”,改变指向

不可返回局部变量的地址(因为局部变量在栈中，被释放后有可能写入其他数据，这样局部变量的地址取值输出时会是垃圾数据)，但可以返回字面值：char*　fun(){return “xxxx”;}

char a[]=”abcd”;

数组字符串内容可以改变，初始化后只能用 strcpy 方式赋值

“=”改变地址，strcpy 改变内容

char *p=(char *)malloc(sizeof(char));

可写为 char *p=malloc（1）；

不可改变 p 的指向（内存泄漏），只能用 strcpy。

可向 p 输入字符串（指针必须指向某片空间）

p=”abc”； 堆分配后可以延长使用，若 p 指向的是栈空间，则不可以延长使用

1.1.9. 内存

程序：在硬盘上可以执行的文件

进程：在内存中运行起来的程序

一个进程的内存空间包括以下几个部分：

也可分为代码段，数据段，堆栈段

1）代码区：程序的代码（函数为基本单位）存入，只读

2）只读常量区：常量和字符串的字面值

3）全局区：存放全局变量和 static 变量，main 函数执行前分配全局区

4）bss 段：存放未初始化的全局变量，main 函数执行前清空 bss 段（清空为 0）

5）堆：自由分配和回收内存

6）栈：局部变量，包括函数的参数，自动分配和回收内存

在全局区定义 const int i=5； 存于只读常量区

在函数中定义 const int i=5； 存于栈

在函数内　　str[5]=”abc”;　栈（未制造常量）

*str=”abc”;　只读常量区(制造字面值)

内存的地址（虚拟内存地址）

每个进程都设定 0 到 4g 的虚拟内存地址（不是内存的真实物理地址，只是一个编号）。0 到 3g 是用户空间，3g 到 4g 是内核空间。虚拟内存地址开始时不对应任何物理内存，直接使用会引发段错误，虚拟内存地址必须映射物理内存（或硬盘上的文件[文件不能小于需要使用的大小]）才能存放数据

虚拟内存地址相当于为物理内存地址取名，存放的只是数字

内存分配其实指的是映射物理内存的过程，内存回收是解除映射的过

程

0

int 4

字节

进程 1 char

4g-1

0 虚拟内存 物理

内存

进程 2

4g-1 1

字节

内存管理的最小单位是一个内存页（4k，4096bit），虚拟内存地址连续时，物理内存地址可以不连续

内存分配<stdlib.h>

malloc:分配内存块，不会对分配内存进行初始化

```
void *malloc(size_t size);
```

分配成功返回指针，失败返回 NULL

exp：

```
int *p=(int *)malloc(3*sizeof(int));
```

可使用 p[0]=1,p[2]=3

calloc:分配内存块，对内存进行清零

void * calloc(size_t number ,size_t size);

为 number 个元素分配内存，每个大小 size，成功则返回指针，失败返回 null

exp：

```
int *p=(int *)calloc(3,sizeof(int));
```

realloc:调整先前已经分配的内存块大小

void* realloc(void *p , size_t size);

p 指向需要调整的内存块，size 为调整后的大小

1）内存扩张，realloc 不会初始化扩张的内存

2）size 为 0 时，释放原来分配的内存

3）失败返回 NULL，不影响原来的数据

释放： void free(void *p);

malloc 一次映射 33 个内存页，多余页留下次用，使用完后再次映射，除了数据区，还需额外保存一些信息（如分配内存大小，方便 free 释放时使用）。所以分配时，上一次 malloc 与下一次 malloc 不紧密连接。free 是把映射解除，相当于将使用状态切换到未使用状态（数据还在原内存位置，只是此时可以当作空白来覆盖）

如果一次申请页数超过 33 页，实际分配的内存页数会稍大于申请页

数。

getpagesize()可以获取当前页大小

malloc 不能返回动态内存：

void ab(char *p){ p=malloc(1);}错误

char* ab(){char *p=malloc(1);return p;} 正确

int *p=10;或者 int *p; int a; *p=a;不可以这样赋值（段错误）

p 有虚拟内存地址，但没有映射物理内存，不能存放数据

1.1.10. const

常用于定义常量，修饰函数的输入参数、返回值。

void ab(const int *p);

接收时可扩展接收范围：

const int *p1=const int *p2；

const int *p1=int *p2；

对于函数 void ab(int a); const 修饰无接收范围意义，因为函数中，a 接收一个值时，a=x 是复制一个变量，const int a=x 时，传入参数时是在做 const 型的 a 的初始化。

当一个变量为 const 型时，可通过指针修改

1）const int a=10;

int *p=&a;

*p=20;可修改 a 的值，编译时有警告，无错误

2）int a=10;

const int *p=&a;

int **pp=&p;

**pp=20;不可通过 p 修改变量，可以使用二级指针

3）int a；

const int b；

int *p=&a；

a，b 地址连续，p+1 可访问到 b 的位置

<c++不可修改成功 const 变量>

1.1.11. 结构体

1.1.11.1. 定义

struct A{……};

定义:struct A a;

初始化:struct A a={*,*,……}

访问或初始化或修改:a.*;

typedef:给类型取别名

exp: typedef int INT;

与宏的区别:#define INT int

typedef struct{…}A;

此时 A 为类名，struct A 等价于 A

结构体指针: struct A　*p=&a;

访问成员:(*p).i 或者 p->i

1.1.11.2. 结构体对齐

结构体成员在内存的边界上按照对齐量进行内存块对齐,连续多个成员合计内存小于对齐量则放一个内存块。

整个的结构体结构体长度为对齐量的整数倍

```
#pragma pack(1)//修改为单个字节对齐
struct INFO{}
#pragma pack()
```

1 字节对齐	2 字节对齐	4 字节	8 字节
char(1) double(8) short(2) char(1)	char(2) double(8) short(2) char(1) *(1)	char(4) double(8) short(2) char(1) *(1)	char(8) double(8) short(2) char(1) *(5)

1.1.11.3. 结构体位段

可以指定每个成员的大小，节约内存(大小指二进制数)

exp: int i:3;指定占三个二进制位

struct{int a:5;int b:2};

结构体大小 4 个字节，向结构体写入“1234”

内存中是对应 ASICII 码：00110000,00110001,00110010,00110011

a:5 b:2 则分别取：00110000,00110001,00110010,00110011

a:5 b:4 则分别取：00110000,00110001,00110010,00110011

b 取出为：1001

1.1.12. 联合

联合:和结构体用法相同(关键字用 union)

所有成员共用起始地址相同的一片空间

exp:union A{int m;char data[4];}

m=0x 44 43 42 41

输出 data 的四个元素为 4 个十六进制数的 ASCII 码代表的字符

int 占 4 个字节,1 个字节 8 位。8 个 1 代表数字 255

char 占 1 个字节,该内存大小可存 8 位二进制，2 位十六进制，int 相当于 4 个 char

m: 44(data[0]) 43 42 41(data[3])

大端模式：低字节存高位数据

小端模式: 低字节存低位数据

union { int i; char x[2]; }a; int main() { a.x[0]=10; a.x[1]=1; printf("%d\n",a.i); return 0; }

0x010A 266

a.x[0]=10; a.x[1]=1;a.x[2]=2;

Ox02010A

1.1.13. 枚举

枚举:为数值(整数)起名字，类似宏

enum 枚举名{枚举常量 1,常量 2,……};

enum color{R,G,B}; 分别代表 0,1,2

int x=G; 或使用 color c=G;

可以使用匿名枚举

枚举的默认值从 0 开始，可以在定义时赋 G=5,则其后的 B 变为 6,R 仍为 0

1.1.14. 预处理

#预处理命令　　在编译的第一个阶段被处理

宏:#define　A　　a

　　将 a 替换为 A,在写程序时，可用 A,编译时自动转换为 a

　宏定义时，\用来续行

宏是原样替换，无类型检查

宏只能用三目运算完成复杂运算

编写宏时，应在每个代表数字的参数及整个计算结果外都加括号，用来保证运算顺序

1.1.14.1. 宏函数

宏的参数都没有数据类型,个数任意

#define Max(x,y)　(x)>(y)?(x);(y)

不可使用 x++语法

#　将宏参数转义为字面值常量

#define　Txt(a,b)　#a"def"#b

int x = 10;

char *p = Txt(x,2.2);

得到字符串xdef2.2

##　　将两个标识符连在一起形成一个新的标识符

#define　M(type)　type##max　　变成 typemax

```
void FunA(){}
void FunB(){}

#define  Fun(X)   Fun##X()

Fun(A);   调用 FunA 函数
```

1.1.14.2. 条件编译

#if　和#endif (#elif,#else)

预处理遇到#if 会判断后面的宏目前的数值，如果为 0,则排除 if 与 endif 之间的内容

#if　0　　.....　　#endif　　可用来做注释，0 改为 1 取消注释

#if define 简写为#ifdef　　#ifndef

#undef　　删除宏

#error　　报告错误信息，挂起程序

#warning　警告信息

exp:#if (w>3)

#warning “w>3”

#endif

1.1.14.3. 可变参数

#define LOG(format,...) printf(format,__VA_ARGS__);

LOG("%s%d", "x=", 3);

1.1.14.4. NDEBUG

relase 时编译器选项

#ifndef NDEBUG 可以定义在 debuge 是才运行的代码

1.1.14.5. 函数声明

_declspec(deprecated("xxx")) void Fun()

该函数调用时，在编译会提示xxx

1.1.14.6. 头文件

自定义头文件: test.h

#ifndef TEST_H 防止重复定义

#define TEST_H

……

#endif

防止头文件重复包含还可以使用

#pragma once

#include <**.h> 到系统指定路径找

#include " **.h " 到当前目录找，适合自定义的头文件

1.2. C_Fun

1.2.1. 常用方法

1.2.1.1. 全局宏定义

__FILE__ 文件名

__LINE__ 行号

__TIME__ 代码编译生成的时间

__DATE__ 代码编译生成的日期

1.2.1.2. 字符判断

0 != isdigit(c) 满足条件则字符是一个数字

isalpha(c) 判断是否为字母

1.2.1.3. 字符转数字

‘5’-’0’　　5 转成数字 0～9

使用函数：

char　*x=”123_12”;

int b=atoi(x); //函数会自动在非数字的字符截断，结果为 123

char *x="23.34";

double b=atof(a);

char *x="-10"; //可以是正数也可以是负数

long l=strtol(x,&x,2);

//第二个参数用来接收指向函数结束后 char*的位置

//2 代表 x 是一个二进制的数字，函数将字符数字转换为 10 进制数

1.2.1.4. 数字转字符

5-0+’0’　　数字 5 转成字符

#include <stdlib.h>

_itoa(数字，存放数字的字符串或者字符变量，数字的进制)

int x=18;　char b[5];

_itoa(x,b,16); //可以将 x 转成 16 进制的字符串

1.2.1.5. 小写变大写

-'a'+'A'

1.2.1.6. 进制转换

十进制转二进制：

1）整数部分除 2 的余数倒排

2）小数部分乘 2 的积正排

1）余数为 0 或者 1

2）将小数点前的 0 或者 1 取走

0.a*2 得到 1.b（取 1）

0.b*2 ……

二进制转十进制：按位乘以 2 的 n 次方后，所有项相加。

……2^2,2^1,2^0,2^-1,2^-2……

二进制转十六进制：4 位一份，每份为一位十六进制

4 位 1111 为 15

简易方式：

16 的二进制是 10000（1 后面四个 0）

转十六进制快速法：

exp：28

减去一个 16 余 12

16 代表第二位为 1，12 代表 C

所以结果为 0x1C

1.2.2. 中文支持

```
#include <locale.h>


//char* old_locale = _strdup( setlocale(LC_CTYPE,NULL) );
setlocale( LC_CTYPE, "chs" );



//setlocale( LC_CTYPE, old_locale );
//free( old_locale );
```

1.2.3. math.h

```
z=pow(x,y);   //计算 x 的 y 次方，X 与 z 必须同时是 float 或 double。
z = pow((float)x,(float)1.0/3); //计算 x 的三次开方


int a=(int)sqrt((double)25);   //对 25 开平方，得到 5。


abs(x);        //求绝对值。


float   a=sin(3.14/2);   //参数是弧度，计算 sin90 度的值。
```

ceil 向上取整，floor 向下取整

1.2.4. 内存拷贝

1）void *memset(void *dest, int c, size_t count);

将 dest 前面 count 个字符设置为字符 c，返回 dest 的值。

2）void *memcpy(void *dest, const void *src, size_t count);

从 src 复制 count 字节的字符到 dest. 返回 dest 的值.

void *memmove(void *dest, const void *src, size_t count);

如果 src 和 dest 出现重叠，函数会自动处理.

```
char *a=malloc(1);      char *p = a;
memcpy(p,"hello",5);      可以正常输出(也可以写大于 5 的数字)

char      a;        char *p = &a;
memcpy(p,”hello”,5);
```

不可以正常输出，会多出乱码（且不能写大于 5 的数字，会出现段错误）

因为 p 只有指向堆对象时才可正常无限延伸

3）void *_memccpy(void *dest, const void *src, int c, size_t count);

从 src 复制 0 个或多个字节的字符到 dest. 当字符 c 被复制或者 count 个字符被复制时，复制停止.

如果字符 c 被复制，函数返回这个字符后面紧挨一个字符位置的指针. 否则返回 NULL.

4）void *memchr(const void *buf, int c, size_t count);

在 buf 前面 count 字节中查找首次出现字符 c 的位置. 找到了字符 c 或者已经搜寻了 count 个字节，查找即停止.

操作成功则返回 buf 中首次出现 c 的位置指针，否则返回 NULL.

5）int memcmp(const void *buf1, const void *buf2, size_t count);

比较 buf1 和 buf2 前面 count 个字节大小.

返回值< 0，表示 buf1 小于 buf2;

返回值为 0，表示 buf1 等于 buf2;

返回值> 0，表示 buf1 大于 buf2.

int memicmp(const void *buf1, const void *buf2, size_t count);

比较 buf1 和 buf2 前面 count 个字节. 与 memcmp 不同的是，它不区分大小写.

1.2.5. 字符串

1.2.5.1. 函数

char a[m];

scanf(“%s\n”,a);　无法读入字符串中的空格，遇到空格会截断

gets(s);可以读入输入字符串中的空格

fgets（a,n,stdin);用 n 限制输入字符的数量，如果输入的字符少于 n，可以将’\n’也读入，该函数比 gets 安全

c=getchar()　从键盘读入字符，可读入空格

格式化字符串函数需要格式精确匹配，如 float 一定是用%f，double 一定是用%lf。返回值为成功匹配的个数。

sprintf（str，“输出格式字符串”，输出列表）；

sscanf（str，“输入格式字符串”，输入列表）；

```
char *str="10,1.20576,3.1,abc";
int a = 0;
double b = 0;
float c = 0;
char s[100] = {0};
sscanf(str,"%d,%lf,%f,%s",&a,&b,&c,s);
```

1.2.5.2. <string.h>

函数	原型	使用
strlen	size_t strlen(const char *string)	获取字符串长度，字符串

		结束符 NULL 不计算在内，没有返回值指示操作错误. char s[10]=”abcd”; sizeof(s) 10 strlen(s) 4
strcpy	char *strcpy(char *dst, const char *src)	把 src 复制到 dst 中（清空 dst）
	char *strncpy(char *dst, const char *src, size_t count)	复制指定长度 不清空 dst（覆盖）
strcmp	int strcmp(const char *s1, const char *s2)	返回值< 0，表示 s1 小于 s2; 返回值为 0，表示 s1 等于 s2; 返回值> 0，表示 s1 大于 s2
	int strncmp(const char *s1, const char *s2, size_t count) 只比较 count 个字符	
strcat	char *strcat(char *dst, const char *strSource)	将 src 添加到目标串 dst 后面，并在得到的新串后面加上 NULL 结束符. src 会覆盖 dst 后面的结束符 NULL. 在字符串的复制或添加过程中没有溢出检查，所以要保证目标串空间足够大. 不能处理源串与目标串重叠的情况. 函数返回 dst 值
	char *strncat(char *dst, const char *src, size_t count)	将 src 开始的 count 个字符添加到 dst 后. 如果 count 大于 src 长度，则会用源串的长度值替换 count 值
strset	char *strset(char *str, int c)	将 str 的所有字符设置为字符 c，遇到 NULL 结束符停止. 函数返回内容调整后的 str 指针
	char *strnset(char *str, int c, size_t count)	将 str 开始 count 个字符设置为字符 c
strstr	char *strstr(const char *string, const char *strSearch)	在字符串 string 中查找 strSearch 子串. 返回子串 strSearch 在 string 中首次出现位置的指针. 如果没有找到子串 strSearch，则返回 NULL. 如果子串 strSearch 为空串，函数返回

		string 值
strchr	char *strchr(const char *string, int c)	查找字符 c 在字符串 string 中首次出现的位置, NULL 结束符也包含在查找中. 返回一个指针, 指向字符 c 在字符串 string 中首次出现的位置, 如果没有找到, 则返回 NULL
	char *strrchr(const char *string, int c)	查找字符 c 在字符串 string 中最后一次出现的位置, 也就是对 string 进行反序搜索
strpbrk	char *strpbrk(const char *string, const char *strCharSet)	从 str 的第一个字符开始查找，返回第一个出现在 strSet 中的字符位置指针. 如果两个字符串参数不含相同字符, 则返回 NULL 值.
strcspn	size_t strcspn(const char *string, const char *strCharSet)	从 str 的第一个字符开始查找，返回第一个出现在 strSet 中的字符位置下标 如果一直不满足条件则会返回字符串 NULL 的位置
strspn	size_t strspn(const char *str, const char *strSet)	从 str 的第一个字符开始查找，返回第一个没有出现在 strSet 中的字符位置下标
strrev	char * strrev(char *Str)	将字符串 string 中的字符顺序颠倒过来. NULL 结束符位置不变. 返回调整后的字符串的指针 (可以不使用返回值)
strupr	char *_strupr(char *string)	将 string 中所有小写字母替换成相应的大写字母, 其它字符保持不变. 返回调整后的字符串的指针
strlwr	char *_strlwr(char *string)	将 string 中所有大写字母替换成相应的小写字母, 其它字符保持不变. 返回调整后的字符串的指针

strdup	char *strdup(const char *strSource)	函数运行中会自己调用 malloc 函数为复制 strSource 字符串分配存储空间，然后再将 strSource 复制到分配到的空间中. 注意要及时释放这个分配的空间. 返回一个指针，指向为复制字符串分配的空间；如果分配空间失败，则返回 NULL 值.

.

char *strtok(char *strToken, const char *strDelimit);

在 strToken 串中查找下一个标记, strDelimit 字符集则指定了在当前查找调用中可能遇到的分界符.

返回一个指针，指向在 strToken 中找到的下一个标记. 如果找不到标记，就返回 NULL 值. 每次调用都会修改 strToken 内容，用 NULL 字符替换遇到的每个分界符，所以字符串不能是字面值。

```
char input[16]="abc,d";
char *p;
p=strtok(input,",");
if(p) printf("%s\n",p);
p=strtok(NULL,",");
if(p) printf("%s\n",p);
```

第一次要设定参数，第二次参数可以 NULL

函数第一次调用需设置两个参数，strtok(str,",")

str 需要分割的串，根据“,”分割

第一次分割的结果，返回串中第一个，之前的字串,也就是上面的程序第一次输出 abc

第二次调用该函数 strtok(NULL,"."),第一个参数设置为 NULL，第二个参数还是分割的依据

结果返回分割依据后面的字串，即上面的程序输出 d

====按照 Split 分割字符串

```
char  *_Context;
char *p = strtok_s(pBuf, Split,&_Context);
while (NULL != p)
{
      p = strtok_s(NULL, Split,&_Context);
}
```

1.2.6. I/O

<stdio.h>

printf("格式及内容",参数列表)

参数表达式多个时，从右向左计算

%后为参数输出格式,\后跟字符表示转译字符,其他内容原样输出

printf(“a=%d\n”,a);

可以直接输出格式字符串：char　*p=”abc”;　printf(p);

printf("%.*f",2,a);　　输出参数列表中定义输出宽度

puts(字符串)

scanf(“格式”，地址);

scanf 在按下回车时结束输入，在读到空格后会读入空格之前的内容，放入输入位置，剩余内容留在输入缓冲区（无空格就全读入，%c 可读入空格）

scanf("%*d %d",&a);　　忽略输入的第一个数字

清空输入缓冲区：

缓冲区内有内容时，会输入到下一次要输入的位置

scanf(“%*[^\n]”);

scanf(“%*c”); 读走所有非\n 的字符，然后读走\n。

当这两条语句前无输入时，会要求输入。

Exp：

scanf（“%d”,&a);

scanf(“%*c”);

scanf 不读入\n,会留在缓冲区，此句将\n 读走

输入缓冲区显示条件

1)遇到\n

2)函数结束时

3)输出缓冲区满

4)fflush(stdout)　强制刷新显示

获取单个字符：int c = getchar();

获取字符串（可以读取到空格）

char s[1024] = {0};

gets(s);

该函数不安全，使用 fgets 替代

char a[100] = {0};

```
 //fflush(stdin);
fgets(a,100,stdin);
a[strlen(a)-1] = 0;
```

1.2.7. 文件操作

文件就是存储介质上的数据的集合

操作系统都是以文件为单位对数据进行管理，每打开一个文件，都有一个结构体来保存，这个结构体是系统定义：FILE

文件指针就是指向结构体的指针

1.2.7.1. 打开关闭

FILE *fp　定义一个文件指针 fp

fopen(“文件名”，”打开方式”)

　　成功则返回文件指针，失败则返回 NULL

fclose(“文件指针”)　　文件指针=NULL；

　　成功则返回 0，失败返回-1（EOF）

文件打开方式：

r	只读	
w	只写	（文件存在时，打开会清空）
a	追加	（在原有内容上追加新内容）

r+ ， w+ ， a+ 都是读写

二进制读写

rb，wb, ab

rb+（r+b） wb+（w+b） ab+（a+b）

1.2.7.2. 读

fread(*p，psize,读取次数，文件指针） 二进制读取

fgets(str,n,fp)

返回值 str 首地址。

从 fp 中读出 n-1 个，最后加上'\0',放入 str。

如果读取到\n 或者遇到 EOF,则读入结束。因此读取的行内容中最后一个字符可能是'\n'。

```
char str[100] = { 0 };
while (fgets(str,100,pf))
{
    int posEnd = strlen(str) - 1;
    if(str[posEnd] == '\n') str[posEnd] = 0;
    puts(str);
}
```

fgetc(文件指针) 从文件中读出一个字符出来

失败返回 EOF，成功则返回读出的字符

fscanf（文件指针，格式字符串，输入列表）

返回从文件中格式化到输入的参数列表的个数

返回 0 表示没有格式化成功

如果返回值为-1 表示读取到末尾

1.2.7.3. 写

fwrite(p，psize，写入次数，文件指针)

fputs(pstr,fp) 成功返回非负数，失败返回 EOF

fputc('字符'，文件指针） 将字符写入文件

失败返回 EOF，成功则返回字符的 ASCII 码

fprintf(文件指针，格式字符串，输出列表)

将输出列表中的值按格式字符串变成字符串然后写到文件中。相当于 printf 不将结果输出到屏幕，而是输出到文件中。

1.2.7.4. 刷新

fflush(文件指针) 刷新缓冲区，也就是将缓冲区内容立即写入或者读取

1.2.7.5. 指针位置

文件读写时注意文件指针的位置

ftell（文件指针） 获得指针距离文件开头的距离

rewind(文件指针) 将指针移动到文件开头

fseek（文件指针，移动大小，基准位置）

SEEK_SET 0 以开头为基准

SEEK_CUR　1　　以当前位置为基准

SEEK_END　2　　以文件末尾为基准

fseek(fp,0 ,SEEK_END);

int fileLen = ftell(fp);　　将指针移动到文件尾，然后得到文件大小。

1.2.7.6.　文件操作

remove(“a.txt”);

rename(“a.txt”,”b.txt”);

1.2.8. 随机数

stdlib.h

//一个函数只能设置一个随机种子，不能写在循环内部重复设置

srand(time(0));

rand();//读取掉第一个数据，第一个产生的数据总是不断增大

int x=rand()%100;得到 100 以内的随机数

1.2.9. 系统命令

stdlib.h

相当于在 cmd 中运行命令行

system("pause");　　//让 cmd 窗口不关闭

system("cd d:");

system("dir"); //等价于在命令行连续执行两个命令

1.2.10. time.h

time_t t=time(0); 可获得当前时间(从 1970 年 1 月 1 日 0:0:0 开始的秒差)

显示时间：

while(1){

time_t t=time(0);

printf("…\r " ,t);

fflush(stdout);同一位置显示时间，不断刷新

while(t==time(0));等待 1 秒

}

s=t%60;

m=t%3600/60;

h=(t%(3600*24)/3600+8)%24

ctime(&t) 转成美式时间 %s

char buf[100]={0};

struct tm* cur=localtime(&t);

sprintf(buf,”%4d”,cur->tm_year+1900);

tm_year+1900

tm_mon+1

tm_mday

tm_hour

tm_min

tm_sec

sleep()

Exp: int a=sleep(3)

使程序休眠 3 秒，中断时返回剩余为休眠的秒数

1.2.11. <errno.h>

errno 为全局变量，存放错误编号，出错则 errno 的位置改变，不出错则不变化

perror(“str”) 自动找到错误编号，并打印错误信息，str 为额外的提示信息，可以不加

strerror()把错误编号转换成错误信息

printf(“%m\n”,errno); 打印错误信息

1.2.12. <stdarg.h>

```
int max(int cnt,...){
    int maxValue=0;
```

```
    int i=0; //在 va_list v 和 va_end(v)之间不可定义变量，所有变量的定义必须放在前面
    va_list v;    //定义一个变长参数列表
    va_start(v,cnt); //将参数列表定位到第二个参数位置（cnt 是对…包含几个参数进行说明】）
    maxValue=va_arg(v,int); //给 maxValue 赋的值为列表中当前位置的值，同时指针指向下一个值
    for(i=1;i<cnt;i++){
    int data=va_arg(v,int);
        if(data>maxValue)
            maxValue=data;
    }
    va_end(v); //释放变成参数列表
    return maxValue;
}
```

调用 max()函数时，可用 max(2,34,123,23),内部参数数量可以任意

1.2.13. <assert.h>

assert 函数在 NDEBUG 下自动屏蔽

1.2.14. 环境变量

每个程序都会收到一张环境表，是一个字符指针数组，存着各种数据的指针，以 null 结束。所有的环境变量在程序中可以通过环境表获取。

获取方式：

(1)extern char ** environ；

存放连续的一级指针（字符串），用二级指针可遍历

environ 就是环境表的首地址，是全局变量。

(2)主函数的第三个参数：

main(int argc, char **argv, char ** env)

env 用来接收 environ

遍历环境表：

```
for(;*env;env++)
  printf("%s\n",*env);
```

<stdlib.h>

putenv(“name=value”)

将形式为 name=value 的环境变量放入环境表

getenv(“name”)

返回 name 关系的 value 的指针（value 为字符串）

setenv(“name”,”value2”,1)

将 name 关系的内容设置为 value2，第三个参数决定是否替代已

有变量（0 代表不替代，1 替代）

unsetenv(“name”)

删除定义

clearenv()

删除环境表中所有项

1.3. c++

1.3.1. 特性

c++头文件无.h,需要使用 c 语言头文件时：

<stdio.h>或<cstdio>

std 命名空间有：cin，cout，endl，string，fstream 等

c++后缀为：.cpp .C .cxx

赋值可以用： int i(3);

c++中结构、联合、枚举

struct、union、enum 不再是类型的一部分，名字就是类型名

struct A{.....}; A a;

c++允许使用匿名联合

c++的 struct 可以像 class 那样用，不同的是结构体默认是 public，而 class 默认是 private。

没有任何字段的结构体，c++大小 1，c 中为 0

结构体封装数据，函数封装代码，结构体大小与定义的函数数量无关，只与变量有关。

1.3.2. 命名空间

一组相关的类型、变量、函数、对象等组织到一个逻辑结构中，按逻辑划分模块，可避免命名冲突

namespace　空间名{.....}

命名空间内可以定义函数，函数内不可定义函数。

同一命名空间可以分开写，定义后会整合成看作一个空间，在分开写时，如果要定义函数，则需在另外的空间声明

1）访问命名空间元素或者函数

空间名::a　　空间名::b()

2)若访问使不想使用::语法

using　namespace　n;

可以直接使用 n 中所有元素

using　n::i;

可以直接使用 n 中的变量 i

using 使用时，不能引用同名的变量，会产生冲突，使用::语法解决

无名命名空间：　namespace{int a；.....}

使用时： ::a 或者 a

全局变量与无名命名空间同名变量时会冲突，::a 不会和局部变量冲突，全局变量与局部变量重名时，局部优先。

匿名空间不可以跨文件访问

命名空间内套有命名空间时：

n1::n2::a 访问到 n1 内 n2 里的 a

命名空间的别名：

namespace n4=n1：：n2

将 n1 中的 n2 命名为 n4

嵌套的命名空间：

命名空间 A 中包含另一个命名空间 B，则 A 相当于 B 的全局区，所以 A 不能访问 B 中的元素，B 可以访问 A 中的元素。

重命名类：

```
class ABC{public:void fun() { cout << "xxx" << endl; }};

namespace NA {
    class AAA : public ABC {};
}
```

使用：

```
NA::AAA aa;
aa.fun();
```

1.3.3. 匿名联合

不需要定义联合名，全局区联合需要是 static

```
static union
{
    int  VAL;
    char CH;
};

VAL = 123;
int *p = (int*)&CH;
int x = *p;
```

1.3.4. 函数重载

函数重载：用同一作用域中，函数名相同，参数列表不同（个数，顺序，类型）的函数构成重载，返回值不构成重载。

因为 gcc 编译时不会给函数改名，g++编译时根据名字和参数改名

在 c++代码中的函数名前加 extern “C”可以在编译时不改变函数名，这样就可以让 c 与 c++互调函数

1）c 调用 c++函数：

g++ -c tc.cpp 生成 tc.o 文件

gcc t.c tc.o -lstd++

2）c++调用 c 的函数

gcc -c t.c 生成 t.o 文件

g++ tc.cpp t.o

1.3.5. 函数默认值

int ab(int x=1, int y=10){}

这样写可以在调用时不传入参数，如果传入参数，则使用传入的值替代默认值

靠右原则：默认一个参数值后，后面的参数都需要默认（左边的可以不默认）

当函数声明和实现分开时，参数的默认值在声明部分指定

ab(int x=1)与 ab()在使用 ab()调用时会产生冲突

1.3.6. bool

c++中 boo 是一个特定的类型

0 '\0' false NULL 代表 false

其他都是 true

1.3.7. 哑元

哑元：在实现函数时，函数的参数只有类型，无形参名

exp：

原函数： void ab(int a);

改变后： void ab(int);

使用后 ab()或者 ab(4)都可以，可以保存函数之前的使用方式

当 void ab()和 void ab(int)同时存在，ab(3)会使用 ab(int)函数，ab()会使用 ab()函数

1.3.8. 内联函数

inline　void　ab(){}

inline 关键字必须放在函数实现处，放在函数声明处无效。

如果类内函数在声明的同时实现，未分开写，就会变成内联函数。

头文件中只能放函数的声明，不能放实现。否则因为多个文件包含了头文件，会出现重定义。如果要在头文件中放函数的声明及实现，可以直接定义为内联函数。

直接把函数的二进制代码复制到调用位置，适合调用频繁，且代码量小的函数。主要是为了减少函数跳转调用的时间。内联只是一种优化策略，对编译器的权限只是一个建议，宏预处理时替换，内联编译时替换。

1.3.9. 内存分配

new——delete　　new[]——delete[]

new：

*p=new　类型；　　int　*p=new int；

*p=new　类；　　A　*a=new A();

*p=new　类型(初值)；　int *p=new　int(3);　*p 的值为 3

delete：

1）delete　指针

delete　p；

p=NULL；

2）int *p=new　int[5];　（可使用 p[1]=10）

delete[]　p;

int　(*p)[3]=new　int[4][3];

p 为()[3]数组的指针，分配 4 个数组大小的空间

定位内存分配：

char　data[100];

int　*p=new(data)　int[25];

直接从栈中分配内存（data[100]中），不需要释放内存

1.3.10. 引用

引用:变量的别名

c++中函数可以使用引用改变外部函数的值（类似指针）

void　fun(int&　a){}

不能返回局部变量的引用，加 static 可以让变量被返回

定义引用：　int　x=100;　引用必须初始化

int　&y=x；

一个变量可以有多个引用，引用绑定后不可作为其他变量的引用

引用前加 const 就可以为引用赋常数：

const　int　&ab=10；

或者 void ab(const　int　&a);　调用时 ab(10);

不可以有常量的引用：如&3

int　(&a)[5]　数组的引用

c++中引用和指针的区别：

1）引用代表变量本身，而不是变量地址

2）引用必须初始化，而指针不用

3）一个变量的引用建立联系后不可以关联到别的变量，而指针可以随时改变指向

4）有指针的指针，无引用的引用

5）有指针的引用，无引用的指针　　int *&b=p；

6）有指针的数组，无引用的数组

1.3.11. 类型转换

新类型　　变量名 = ***_cast<新类型>(原始变量)

1）隐式类型转换　　static_cast

避免自动类型转换

int a = 10;　　float b = static_cast<float>(a);

不能让指针类型互相转换，比如char*转换为int*。对于指针只能从void*转为其他指针。

2）const_cast　　去掉常属性

volatile const int y=100；

int *p=const_cast<int *>(&y);

*p=20;

如果不加 volatile，y 为 10，加了以后 y 才能被改变。

3）dynamic_cast

当 A 有虚函数时才能使用，否则编译报错。

A *pa = new B();　　B *pb = dynamic_cast<B*>(pa);

4）reinterpret_cast

允许任何类型指针类型的转换，包括指针变整数，整数变指针

1.3.12. 类

1.3.12.1. 概述

面向对象的三大特征

封装：把不需要用户知道的隐藏起来（private），然后提供公开的访问接口(public)。提供公开的接口一般是在 public 下写函数，通过函数访问到类内私有成员。可以保护数据，防止不必要的扩展。对于调用者而言，使用简单，便于协同开发。

继承：可以把一个类相关数据传到下一个类中，便于扩展（可以增加新特征和功能）。

多态：可以通过子类改写从基类中继承的方法。

类：是对象的抽象描述

对象：对类的实例化

A *a=new A();

A a=A();

A a; 定义一个类 A 的对象 a

A().fun(); 构造一个临时对象，并调用成员函数

对象就是实际内存中的内容，引用（指针）保存对象的地址，变量保存引用

1.3.12.2. 内存分布

一个空的类对象占 1 个字节
类对象的大小由非静态的成员变量大小决定，函数和静态成员不占用对象的大小，如果类中有 virtual 函数，则产生虚函数表，会增加一个指针大大小。

1.3.12.3. 与结构体区别

struct 包装数据，函数包装代码，类既包装数据，也包装代码

行为：类中的函数

特征：类中的变量

class 和 struct 都可以描述对象

struct 可以使用{.....}初始化对象，class 只有在最新的标准才可以使用这种初始化的方法（需要 public）

```
struct A
{
    void fun(){cout<<"xxx"<<endl;}
    int a;
    double b;
};
```

可以使用A a={1,1.2};

如果把struct换成class则可能无法使用该种赋值方式

1.3.12.4. 权限

struct 默认是完全公开的，class 默认是私有的

private：只有在类内可以使用

public：在类内外都可以使用

在 class 中加 public:则此语句下的内容都可以在类外和类内访问，直到遇到下一个权限的修饰语

class A{.....};

A a; 构造函数无参时，此行不可写成 A a()

A *p=new A();

new 后放的是 A 的构造函数

1.3.12.5. 类的基本函数

class A{.....

public: (如果定义构造函数，则不能省略)

构造函数重载
- A(){.....}
- A(int x， int y)：a(3),b(x),y(y)初始化参数{.....}
- A(int x=1){....}

拷贝构造
- A(const A&a){.....}

稀构函数 ~A(){.....}

}

声明和实现分开时：初始化参数列表放在实现部分，参数的默认值在

声明部分指定，函数前的 static 修饰放在声明处，函数的 const 修饰在声明和实现部分都写。

构造、析构都有系统默认的，如过要自定义，会覆盖系统的。此类函数无返回值

拷贝构造内的 A&a 可以不写 a，如果拷贝构造的操作中有需要使用 a，才必须写上

1.3.12.5.1. 构造函数

构造函数在对象创建时调用一次

A a； 创建一个栈对象，同时调用无参构造

A a(10); 创建一个栈对象，调用有参构造（传入参数 10）

A *p=new A(); 创建一个堆对象，同时调用无参构造（无参构造时也可用 A *p=new A）

A(x=1)和 A()在使用 A()调用时会产生冲突

const 变量只能在初始化时赋值（初始化参数列表中进行），初始化参数列表中写成 A(),如果 A 是基本类型,则 A 的值变为 0，如果 A 是类类型，则为无参构造（继承中用）

1.3.12.5.2. 拷贝函数

拷贝构造函数调用时间：在使用同类型的对象构建一个对象(复制)；

在函数参数值传递；函数返回对象一旦有拷贝操作就调用

使用时：A　a;　　A　b=a;

不可写成 A　a；　A　b；　　b=a；

1）A　get(){A　a；return a;}

　　A　　c=get();

　　不用拷贝构造，系统自动优化

2）A　get(){A　*a=new　A(); return *a;}

　　A　　c=get();

　　会调用拷贝构造，因为使用了堆

自定义拷贝构造时，必须同时定义构造函数，不可以只定义拷贝构造（可以只定义构造，不定义拷贝）

1.3.12.5.3. 析构函数

析构函数中执行释放内存可能会造成内存泄漏。

在栈中将 class 创建出对象后，会在程序某行之后不再出现该对象时，自动调用析构函数（在最后一次出现对象的行调用）

析构函数用来释放内存

可单独调用~A()　堆内层时必须单独调用析构，因为堆内存不会自动调用析构函数

exp：A　*a=new　A();

a->~A()

1.3.12.5.4. 虚析构函数

（在析构前加 virtual），基类有虚函数，基类对象的指针指向子类对象(多态条件时)，释放基类对象的该指针时，如果基类析构函数不是虚函数，则子类析构函数的调用行为未定（不调用），把基类中析构变为虚函数，则会调用子类构造，同时也会调用基类的析构。

构造函数不能为虚函数（拷贝构造，构造）

1.3.12.5.5. 深拷贝

当要拷贝的对象中有指针变量时，会使用拷贝前后两个对象中的指针指向同一块堆内存，在其中一个对象释放堆内存后，另一对象中指针指向的是一块已经释放的内存，再次释放会出错，解决此问题需要使用深拷贝方式（自己定义拷贝过程）

```
class　A{
        int　m;
        int　*n;
    A():m(10),n(new int(8)){}
    A(const A&a):n(new　int){
            m=a.m;　　*n=*(a.n);
     }
```

```
    ~A(){
        if(n！=NULL){delete    n; n=NULL;}
       }
};
```

1)构造函数为指针分配堆内存

2)拷贝构造函数为拷贝出的对象的指针分配新的堆内存，并进行其他数据的拷贝

3)自定义析构函数释放堆内存（也可以不使用 if 语句进行判断，因为有时候没有给类的指针成员分配内存，而是指向 null，可以不用释放,delete NULL 也不会出问题）

因为有了 1)和 3)所以才需要 2)的步骤。

1.3.12.5.6. 拷贝构造和赋值函数

编译器会自动生成缺省的拷贝构造和赋值函数，如果不想编写这两个函数，也不希望默认的被调用，则可以将这两个函数声明为私有的。

1.3.12.5.7. c++11 初始化

```
class ABC
{
    int x = 10;
    int y{2};
};
```

1.3.12.6. 声明与实现

类的声明和定义在开发时，是需要分开的，定义写在头文件中

返回类型　　类名：：函数名（参数）{.....}

==========声明

```
#ifndef    T_H
#define    T_H
class    Date{
     Date();
     void ShowDate();
};
#endif
```

==========实现

```
#include    “t.h”
Date::Date(){.....}
void Date::ShowDate(){.....}
```

1.3.12.7. this 指针

this 是指向当前对象的指针，*this 代表当前对象。构造对象时，指向正在构建的对象，成员函数中，指向这个函数的对象

当函数形参与成员变量重名时，可以用 this 区分：

```
class    A{
        int b;
     public:
        A(int b){this->b=b;}
```

};

可使用 this 返回数据，可作为参数传递

1.3.12.8. 静态成员、函数

静态成员：类和类的对象所共享的函数、数据（相当于全局变量、函数，存在于全局区，不属于对象，只是访问范围限制在类内），不需要对象，只需要类型就可以访问。

声明和实现分开时，函数前的 static 修饰放在声明处。

静态成员必须在类外（全局区）进行初始化操作，不能在类的初始化中进行赋值。

```
class    A{ public: static    int    x;};
int    A::x=0;
```

类内函数或类的对象，能直接访问类内的静态变量、函数。

类外调用静态成员、函数[需要是 public]

```
A::x;
A::fun();
```

静态函数只能直接访问静态成员，不能直接访问非静态成员，因为无 this 指针。如果想在静态函数中访问非静态成员，可以传入一个指针：

```
static void ab(A *a){    cout<<a->m<<endl;}
```

1.3.12.9. 成员指针

```
typedef void (A::* FUN)(int a);
```

类型是：void (A::*)(int);

//赋值

FUN f = &A::f1;

使用：

(a.*f)()

(this->*f)()

成员指针直接输出时为 1

用联合可以输出指针指向的地址

union{int A::*p; void *pr;};

用 pr 可以输出指向的地址

1.3.12.10.const 对象和函数

const 对象只能调用 const 函数，声明和实现分开时，函数的 const 修饰在声明和实现都需要写。

1)const A a; const 对象

2)void ab() const{.....} const 函数（类内）

void ab(){.....}

以上两个函数构成重载（非 const 函数优先调用非 const 函数，如果无非 const 函数，就用 const）

const 函数不能修改普通成员变量，如需修改，则在成员前加 mutable 修饰

exp： mutable int m；

void ab()const{m=10;}

1.3.12.11. 友元

友元函数：在全局函数中要调用类中私有成员，需要在类中用 friend 声明一下这个全局函数。则该全局函数可访问到类中私有成员（通过类的对象）

exp:

```
class A{int x;};
A a;
void ab(A a){cout<<a.x<<endl;}
```

把 class 改为：

```
class A{
    int x;
    friend void ab();
};
```

友元类：

friend　class　B；

在 B 继承 A 时，使用友元类，可以使 B 能访问到 A 的 private 成员

扩展用法：

```
template <typename T>
class AAA{friend T;int x;};
class BBB
{
    void fun()
    {
        AAA<BBB>  aa;
        aa.x = 10;
    }
};
```

1.3.12.12.其它

在类中调用其他函数，需要在类前声明要使用的函数。

对象创建过程，分配这个对象内存

对象没有任何成员，大小是 1

对象大小由数据成员决定，和函数无关（虚函数除外）

如果对象成员是 class 类型，调用这个对象的无参构造。基本类型不做任何操作，有初始化参数列表时，先执行初始化参数列表

不能在定义时对数据进行初始化

class　A{int a;}不能写成 int a=10；

当 class 中无数据需初始化，有时需要用 a={}或者写一个空参构造

A& b(){.....}

A a;

a.b().b().b(); 返回类型是 A&，所以可以让函数多次调用

explicit A(){} 构造函数

防止 A a=10; 语句将 10 转成 A 类型

class 中每个函数的第一个参数实际为类的指针

void ab(int *) 等价于 void ab(A a, int x)

因此使用 a.ab(y);

在全局区定义时： void (*p)(A,int)

1.3.13. 内存分配

new 与 malloc 区别：

1）new 会在创建对象时自动调用对象的构造函数

2）自动处理类型转换

delete 与 free 区别：

delete 释放内存前会自动调用析构函数

1.3.14. 运算符重载

1.3.14.1. 定义

运算符重载(一种特殊的函数)

双目运算符重载：

对象 1　运算符　对象 2

类内重载：

void　operator+(对象 2){.....}

void 为返回值类型，可任意

+为要重载的运算符，相当于函数名

在类内重载可以少写第一个参数，因为类内有 this 指针（this 指针指向当前对象 1）

对象 1 必须是类，对象 2 类型任意

全局区重载：

void　operator+(对象 1，对象 2){.....}

运算符重载后，即可用运算符对类进行运算，一般重载尽量定义在类内

输入输出重载只能在类外，不能打开输入输出类

istream&　operator>>(istream&is,　A &a){

return is>>a.x 或者 return cin>>a.x

}

is 等价于 cin，可任意名

A &a 不能加 const，输入是做改变，&a 引用 a，因为相当于在函数中改变外部的值

istream& 返回输入流，即返回的值后面可以用流的语法。

ostream& operator<<(ostream& os, const A &a){.....}

1.3.14.2. 单目运算符重载

运算符 对象 1 -, !, ~, ++, --

与双目运算符重载类似，只是少了一个对象 2

++，--重载时默认为前++、--：

类内定义： void operator++(){.....}

使用哑元 int 可变为后++、--

void operator++(int){.....}

1.3.14.3. ()运算符重载

```
class AA
{
    int x;
public:
    AA(){x=10;}
    int operator()(int a,int b)
    {//函数方式调用
        return a+b+x;
    }
```

```
        //类型转换
    operator double(){return x;}
};

AA  aa;
int x = aa(10,20);
double y = aa;
```

1.3.14.4. 特殊重载

有些运算符只能重载为类内的成员形式：

=， []

“=”重载时注意自赋值：

if(this==&a) return *this;

可以重载 new，delete，重载时至少有一个类类型

void* operator new(size_t t){.....}

void operator delete(void *p){.....}

*,->希望按照指针方式操作

*返回对象，->返回地址

A& operator*(){return *p;}

A* operator->(){return p;} 迭代器用

注意：(*p).m “.”优先级高

1.3.14.5. 不能进行的重载

不能对基本类型数据进行重载

不能发明新的运算符

不能被重载的运算符：

::

.

sizeof

.*　到成员的指针

？　：　三目运算符

typeid　　类型识别

1.3.15. 继承

1.3.15.1. 定义

B 继承自 A，则 B 中有 A 的一切，而且对 A 进行了拓展。所以 B 可以强制类型转换成 A，而 A 类的指针可以指向 B，反之却不可以。

1）组合继承：在子类中定义一个基类的对象作为成员(不能实现多态)，不可以在 B 的初始化参数列表调用 A 的构造：

```
class　B{A　a;.....};
```

2）class　B:public　A{.....};

或者 protect，private

可实现多态

在类中定义并实现另一个类时：

class A{class B{};};

A、B 不能直接互访成员，A 相当于 B 的类外全局区

1.3.15.2. 权限

public：可被任意实体访问到

protected：只允许本类成员及子类访问

private：只允许本类成员访问

1）public 公有继承：public---------->public

子类内和外都可以访问

protected---------->protected

子类可访问（子类的子类可访问）类外不能访问

private---------->无权

子类内外都不可访问

2）protected 保护继承：public、protected---------->protected

private---------->无权

3）private 私有继承：public、protected---------->private

子类可访问（子类的子类不可访问），类外不可访问到

private---------->无权

子类无权访问时，如果在基类中将子类定义为友元类，则可访问。

或者在基类提供公有的接口让子类可以通过接口访问到

子类继承基类后，基类相当于子类的一个成员，等同于把基类中的成

员写在子类中

子类中没有和基类相同名的数据时，可以直接使用，若有重名，则会隐藏基类数据，使用 A::ab 调用(有访问权限前提下)

1.3.15.3. 函数调用顺序

继承中，先调用基类的构造，后调用子类的构造（析构顺序相反），在子类定义后，不会影响基类的相应构造函数和析构函数

如果基类没有无参构造，则需要在子类构造函数的初始化参数列表中调用基类的构造。

子类默认先调用基类的拷贝构造，后调用子类的拷贝构造(赋值运算符)自定义后将不再调用基类的

子类自定义拷贝构造时：

```
class    B:public A{
        B(const    B&b):A(b){.............}
        调用基类的拷贝构造
        剩余内容（b 中特有内容）的拷贝
};
```

构造函数调用顺序：

虚基类构造--->普通基类构造---->子类中的子类---->子类

1.3.15.4. 多重继承

class B:public A1,public A2{.....};

class B:public A1,A2{.....}; //A2 前不加 public 表示 private 继承

基类有重名时，在子类定义此名，使基类的隐藏

1.3.15.5. 钻石继承

一个基类有多个子类，而这多个子类又是另一个类的基类

```
class A{public:int val;};

class B:public A{};
class C:public A{};

class D:public B,public C{};
```

访问 D 对象的 val 时，无法确定是 B 中的 val 还是 C 中的 val，需要类名作用域：

```
D d;
int x = d.B::val;
```

子类通过不同的基类访问到最高层的类时，会有歧义

为了消除歧义，引入虚继承，这样最高层代码只有一份，不会产生复制，最下层可直接访问到最上层

```
class A{public:int val;};

class B:virtual public A{};
class C:virtual public A{};

class D:public B,public C{};
```

使用时：

```
D d;
int x = d.val;
```

1.3.16. 多态

基类函数必须是 virtual（子类可以不加），使用基类的指针或者引用指向子类，实现基类对象调用子类的多态。

当子类调用基类的函数 a()，而这个函数又调用基类和子类都有的 b()时，会调用基类的 b()，只有重写时，才会调用子类的 b()。

1.3.16.1. 多态的应用

继承是多态的基础，虚函数重写是多态的关键

基类有虚函数，基类的指针或引用指向子类的对象时，基类掉调用这个虚函数就可以有不同表现（这个虚函数在子类中可以重写）

子类重写基类虚函数时，前面可以加 virtual，也可以不加

在子类定义和基类相同的函数，这种机制为名字隐藏，但如果基类函数是虚函数，则称为函数重写（overwrite）

```
class A{public: virtual void ab(){1}};
class B:public A{public:virtual void ab(){2}};
1)A *p=new B();
    p->ab();   结果为 2
```

2）B　b;

A& a　= b;

A 指向 B 后，通过指针 pA 只能访问 A 的成员内容，但是 pA 通过虚函数可以访问到 B 的函数，这样就相当于可以对 B 的内容进行操作。

1.3.16.2. 虚函数表

class 中数据占内存，函数不在 class 内存储，不占 class 内存，但在函数前加 virtual，就产生一个纯虚指针（指向虚函数表），该指针相当于 class 中数据，会占开头位置的内存

每个类只要有虚函数，则会有自己的虚函数表，如果没有重写基类的虚函数，则虚函数表中的函数地址是基类的，如果重写了虚函数，则虚函数表中的虚函数地址是自己的

调用虚函数表中的函数，需要得到虚函数表中函数的地址，虚函数表中都是虚函数的地址

多态时，基类的纯虚指针指向子类的相应虚函数表的位置

1.3.17. 抽象类

不能被实例化的类（类中有纯虚函数就不能被实例化）

纯虚函数：virtual void ab()=0;

除了不能被实例化，和其他类无区别

只能用于继承后，使用多态的方式实现出抽象类

A　*a=new　B();　　A 为抽象类

子类继承抽象类时，如果不实现纯虚函数（定义同名函数，将纯虚函数隐藏），则子类也是抽象类

1.3.18.　异常

当throw出现时，程序会直接跳转到catch处（一直向函数上层搜索catch块，如果没有catch则直接崩溃）

throw　处会创建或复制一个对象，throw上面所有变量进行了析构，然后在该处调用catch函数

```
class A{ public:　int x;　A(int a){x = a;}　};

try
{
    int condition = 2;　//根据condition，抛出不同类的对象
    if(1 == condition){ throw 1;}
    else if(2 == condition){throw　A(2); }
}catch(int i)//接收抛出的对象
{　　cout<<i<<endl;
```

```
}catch(A &a)
{       cout<<a.x<<endl;
}catch(...)//接收前面catch没有枚举出的异常类型
{       cout<<"other exception"<<endl;
}
```

1.3.19. C++11

__cplusplus　　c++编译器版本

__func__　　当前函数名,可用于初始化参数列表标志类名

```
constexpr int a = 10; //编译时确定值
```

```
static_assert(8 == sizeof(void*), "指针长度错误");//编译时确保指针长度为8
```

```
void fun()noexcept(true){}
```
表示函数不抛出异常，如果有异常，结束程序，阻止异常扩散

成员变量定义时可以初始化

```
template<typename T> class A
{
    friend T;
};
```

A<B> a;　　可以在B中访问A的私有成员

using INT = int;

class A final　　类不能被继承

virtual void fun()final　　虚函数不允许重载

void fun()override　　显示声明该函数是对基类函数重载

```
int a[] = {1,2,4,6,7,8};
for (auto i : a)
{
    if (i == 6) break;
    cout << i << endl;
}
```

1.4. c++StdLib

1.4.1. 流

1.4.1.1.　I/O 流

cin、cout 都属于<iostream>中类的对象

类	对象
istream	cin;
ostream	cout;

cout 可以被重定向，命令行启动程序时加入：

>>a.txt　可以将 cout 内容输出到 a.txt，不在屏幕显示

cerr不带缓冲，直接输出到屏幕

<iostream>

io 默认是格式化，会忽略空格和换行，>>对输入进行格式化，使之与特定数据类型匹配

流操作在两个数据之间不论多少空格或者换行，都会忽略

cout<<x<<endl;　等价于　cout<<x;cout<<endl;

因为<<流操作符返回的是左边的对象，即执行完后返回 cout。

cout<<cin<<endl; 流不出错时输出一个地址，出错会输出 0，拒绝 io，但不影响其他操作，输入流会出错的情况：

cin.getline(char *p, size_t　len);最多能接收 len-1 个字符，与*p 内存大小无关，输入个数超过时输入流会出错，此时无法再接受输入

.clear();纠正流状态，让出错的流变正常，如文件到末尾，不会清空流

.ignore(n,'\n')；清空缓冲区，最多清空 n 个字符，当遇到'\n'立即结束

(如果不加入'\n'，会一直等待输入区输入，直到清空够 n 个字节)

exp：

```
if(!cin){
   cin.clear();
   cin.ignore(200,'\n');
}
```

将结束符去掉，剩余内容留在缓冲区，以字符串形式输入到 data(格式化输入)（该函数型式只能用 cin）cin 相当于输入缓冲区

.get()　读一个字符　　exp:　c=cin.get();

.put()　写一个字符　　cout.put(c);

//输出时只保留小数点后两位

```
#include <iomanip>
double x = 21.2323;
cout<<fixed<<showpoint<<setprecision(2)<<x<<endl;
```

1.4.1.2.　文件流

<fstream>

只读 ifstream　　文件必须存在

只写 ofstream　　文件可以不存在，存在则清空

读写 fstream　　可以指定模式

fstream　iof("D:\\1.txt",fstream::out|fstream::app);每次写之前定位到末尾）

fstream iof("a.txt"); 构造函数直接打开文件，也可使用.open()函数打开，打开时也可指定模式，不指定模式时，文件必须存在

if(!iof) return; 判断文件未打开

.close() 关闭文件

==========读取所有数据

```
std::ifstream ifs("D:\\1.txt");
string strval;
std::getline(ifs, strval, (char)EOF);
```

==========逐行操作

iof<<"hello"<<endl<<"abc"<<endl;

向文件中写入两行内容（读/写指针移动）

iof>>str1>>str2; 从文件中读出两行内容

(空格或者换行自动分，两段内容之间不论多少空格或者换行)

string strline;

while (getline(ifs,strline))

char str[100] = { 0 };

while (ifs.getline(str, 100))

==========逐个操作

```
char    c;
while((c=iof.get()!=EOF))
        cout<<c<<endl;
```

逐个输出全部字符（读入前调整指针）

==========文件指针操作

.seekp(n,参考位置)　　从参考位置调整写指针位置(大小为 n)

参考位置：　ios::beg　　ios::cur　　ios::end　　.tellp()写指针当前位置

seekg、tellg 用于读指针，seekp、tellp 用于写指针。

fstream 中，读写指针一体，任意用一个即可，在 ifstream 中只能用读指针，ofstream 中只能用写指针

======获取文件大小

```
iof.seekg(0,ios::end);
fileLen = iof.tellg();
iof.seekg(0,ios::beg);
```

==========二进制读写

```
ifstream ifile(filePath,ifstream::binary);
```

.write　　iof.write(char*,n)

从 str 中向文件写入 n 个字符

.read　　iof.read(char*,n)

从文件中读出 n 个字符放到 str 中

.gcount

返回最近一次非格式化读取的字符的数量

如 read 函数时 n 为 10，但只有 6 个字节，则返回 6

如果没有读取到内容，返回 0

1.4.1.3. 字符流

<sstream>

istringstream　　从流中读出字符串

ostringstream　　向流中写入字符串，相当于 sprintf

wostringstream 对应 wsting

stringstream 同时具备 istringstream 和 ostringstream 功能

```
ostringstream  ostr;   //构造 ostringstream 类的对象
string  name;
int  age;
ostr<<name<< " "<<age;
string  str=ostr.str();    //构造出 string
```

istringstream istr(str); //构造 istringstream 类的对象

istr>>name>>age; //空格或者换行自动分

清空操作：ss.str("");

1.4.1.4. 流操作

>>操作是格式化按照字段之间的空格或换行来区分

getline(stream, string , '结束符')；

Steam 可以是 cin，也可以是 stringsream，也可以是 fstream

结束符代表读取结束的位置，可以不使用，默认为'\n'

1.4.2. <string>

using namespace std;

wstring 需要使用 wcout 和 wcin

使用前需要进行初始化：locale::global(locale(""));

或者 wcout.imbue(locale(""));wcin.imbue(locale(""));

1.4.2.1. wstring 之间转换

```
string str = "abc";
wstring wstr(str.length(),L' ');
std::copy(str.begin(), str.end(), wstr.begin());
```

1.4.2.2. 函数实现

```
class string
{
public:
	char *m_data;
	string(const char *p=NULL)
	{
		if(p!=NULL)
		{
			int len=strlen(p);
			m_data=new char[len+1];
			strcpy(m_data,p);
		}
		else
		{
			m_data=new char;
			*m_data='\0';
		}
	}
	string(const string& other)
	{
		int len=strlen(other.m_data);
		m_data=new char[len+1];
		strcpy(m_data,other.m_data);
	}
	~string(void)
	{
		delete[] m_data;
	}
	string& operator=(const string& other)
	{
		if(this==&other)
			return *this;
		else
		{
			delete[] m_data;
			int len=strlen(other.m_data);
			m_data=new char[len+1];
			strcpy(m_data,other.m_data);
		}
	}
};
```

1.4.2.3. 输入输出

string 类重载运算符 operator>>用于输入，同样重载运算符 operator<<用于输出操作。

getline(istream &in,string &s);用于从输入流 in 中读取一行字符串到 s 中，以换行符'\n'分开（可以读入空格，cin>>s 遇到空格会截断字符串）

输入 ab cd

getline(cin,str); //得到 str 为 ab cd

cin>>str1>>str2;//得到 str1 为 ab，str2 为 cd

1.4.2.4. 构造函数

当构造的 string 太长而无法表达时会抛出 length_error 异常

string(const char *s);　　//用 c 字符串 s 初始化

string(int n,char c);　　//用 n 个字符 c 初始化

string s1；

string s2="hello"；

1.4.2.5. 字符操作

const char &operator[](int n)const;

const char &at(int n)const;

operator[]和 at()均返回当前字符串中第 n 个字符的位置，但 at 函数提

供范围检查，当越界时会抛出 out_of_range 异常，下标运算符[]不提供检查访问。

const char *c_str()const;//返回一个以 null 终止的 c 字符串

const char *data()const;//返回一个非 null 终止的 c 字符数组

如果 string 类析构了，就会指向垃圾数据（必要时使用 strcpy）

int copy(char *s, int n, int pos = 0) const;//把当前串中以 pos 开始的 n 个字符拷贝到以 s 为起始位置的字符数组中，返回实际拷贝的数目

1.4.2.6. 特性描述

int size()const; int length()const; //返回当前字符串的长度

bool empty()const; //当前字符串是否为空

void resize(int len,char c);//把字符串当前大小置为 len，并用字符 c 填充不足的部分

int capacity()const; //返回当前容量（即 string 中不必增加内存即可存放的元素个数）

max_size()const; //返回 string 对象中可存放的最大字符串的长度，结果是一个很大的数值

1.4.2.7. 子串

string substr(int pos = 0,int n = npos) const;//返回 pos 开始的 n 个字符组成的字符串，n 默认值时代表子串是 pos 到末尾。

1.4.2.8. 查找

string::npos 可以指代-1，当未查找到时，可以用来判断返回值。

pos = s.find(str,int pos = 0);//在 s 中查找 str 第一次出现的位置。默认从下标为 0 的位置开始查找

s.rfind(str);//在 s 中查找 str 最后一次出现

s.find_first_of(str);//在 s 中查找 str 中任意字符第一次出现

s.find_last_of(str);//在 s 中查找 str 中任意字符最后一次出现

s.find_first_not_of(str);//在 s 中查找第一个不属于 str 的字符

s.find_last_not_of(str);//在 s 中查找最后一个不属于 str 的字符

1.4.2.9. 替换

s. replace(pos,len,str);//从 pos 位置开始的 len 个字符替换为 str

s.replace(itA,itB,str);//将迭代器 AB 之间的字符删除替换为 str

1.4.2.10. 插入

string &insert(int p0, const char *s);

string &insert(int p0, const char *s, int n);

string &insert(int p0,const string &s);

string &insert(int p0,const string &s, int pos, int n);

//前 4 个函数在 p0 位置插入字符串 s 中 pos 开始的前 n 个字符

string &insert(int p0, int n, char c);//此函数在 p0 处插入 n 个字符 c

iterator insert(iterator it, char c);//在 it 处插入字符 c，返回插入后迭代器的位置

void insert(iterator it, const_iterator first, const_iterator last);//在 it 处插入[first，last）之间的字符

void insert(iterator it, int n, char c);//在 it 处插入 n 个字符 c

1.4.2.11. 删除

iterator erase(iterator first, iterator last);//删除[first，last）之间的所有字符，返回删除后迭代器的位置

iterator erase(iterator it);//删除 it 指向的字符，返回删除后迭代器的位置

string &erase(int pos = 0, int n = npos);//删除 pos 开始的 n 个字符，返回修改后的字符串

1.4.3. <typeinfo>

可以进行运行时类型识别

typeid(变量或类型)

可以获得对象的类型信息

string typeName = typeid(int).name();

B *pb = new B();　　A *pa = pb;

cout<<typeid(*pa).name()<<endl;

if(typeid(*pa)==typeid(*pb)){cout<<"=="<<endl;}

typeid 和 dynamic_cast<>()在转换成相应的子类类型时，

有继承关系/需要父类有虚函数

1.4.4. <ctime>

std::clock_t timeStart = std::clock();//记录当前时间

。。。。。。。。

double t = double(std::clock() - timeStart)/CLOCKS_PER_SEC;

//计算程序运行到此处运行的时间，单位是秒

1.4.5. 线程

```
#include <thread>

void thr(int i,double cc){}//线程入口函数与参数个数任意，参数必须是可以拷贝的类型

thread t1(thr, 10,20);//创建并运行线程
t1.join();//如果在下一个thread定义前调用，可以确保下一个线程在这个线程运行结束后运行
```

```
atomic<A*> aa = new A();//定义原子类型
A* p = *pa;//线程中访问
```

1.5. DataStructure

1.5.1. 链表

由多个节点组成，每个节点中包括有效数据和至少一个节点指针

定义链表头结点时：

```
node *head=new node;
head->next=NULL;
```

判断时：

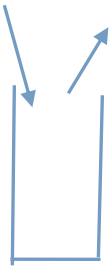
```
node *n=head->next;
while(n)
```

删除时：

```
void destroy(NODE *head){
    NODE *temp=head->next;
    while(temp){
        head->next=temp->next;
        delete temp;
        temp=head->next;
    }
}
```

1.5.2. 堆栈

先进后出，同一端进行插入、删除

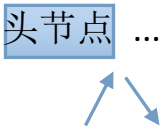


1）数组： 

void stack[N]；

int size; 有效数据个数，作为数组下标

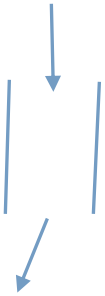
size=0 相当于清空栈

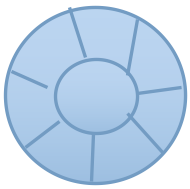
2）链表： 

头节点定位，在头结点后的位置进行删除和插入

1.5.3. 队列

先进先出，一端用来插入，一端用来取出



1）数组： 

void queue[n];

int head,tail,size;

从 tail 处插入，从 head 处取出，取出时先判断

head<tail,用 size 判断是否为空

push：queue[tail]=data;

tail=(tail+1)%n;

size++;

pop: head=(head+a)%n;

size--;

2)链表：头节点 .....

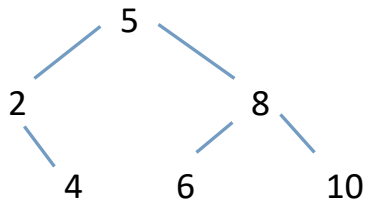
头节点定位，在头结点后取出，在链表尾插入

1.5.4. 树

通常包括节点声明和根指针说明，逻辑结构通常用链式存储方式实现

二叉树：每个节点最多有两个子节点，p_left,p_right

有序二叉树：左比根小，右比根大



二叉树遍历：

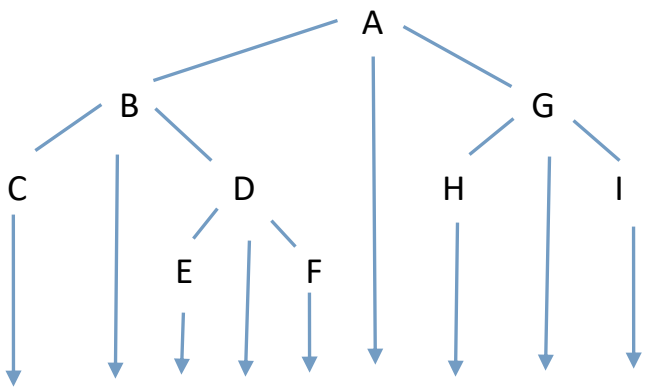
1）前序（根，左，右）

从根开始找，有左时，把左当成根再找，无左时把右当成根再找

2）中序（左，根，右）

3）后序（左，右，根）

中序可以用投影：



C　　B　E　D　F　A　　H　　G　　I

从 c 开始找，再找到 b,再到右边 d 时，把 def 当整体，所以先到左的 e，再到根 d

```
node ** pp_root;
node * p_root;
*pp_root=NULL;   初始值
*pp_root=p_root;    赋值
void   destroy(node **pp_root){
        if(*pp_root){
                destroy(&((*pp_root)->p_left));
                destroy(&((*pp_root)->p_right));
                free(*pp_root);
                *pp_root=NULL;
        }
}
void   show(node   *p_root){
        if(p_root){
                 show(p_root->p_left);
                 printf("%d\n",p_root->num);
                 show(p_root->p_right);
        }
```

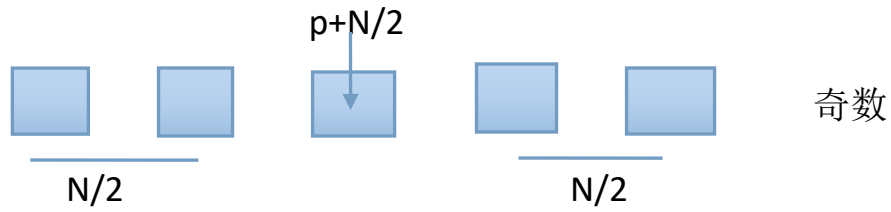
}

1.5.5. 查找

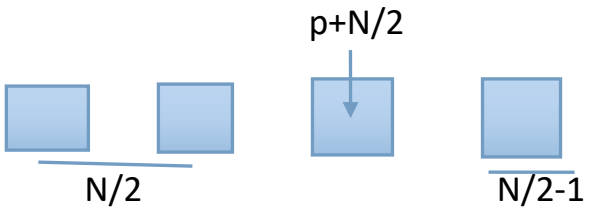
int a[N]

查找：顺序查找、折半查找(用已排序好的数组)

折半查找：每次与中间值进行比较，再决定到前一半或者或一半进行再次查找。



奇数



偶数

```
int    search(int    *a,    int    N, int    num){
   if(N>0){
            if(a[N/2]==num)          return 1;  表示找到
            else if(a[N/2]>num)    return search(a,N/2,num);
            else{
                  if(N%2)
                     return    search(a+N/2+1,N/2,num);
                  else
```

```
                return search(a+N/2+1,N/2-1,num);
            }
    }
    else      return -1;  未找到
}
```

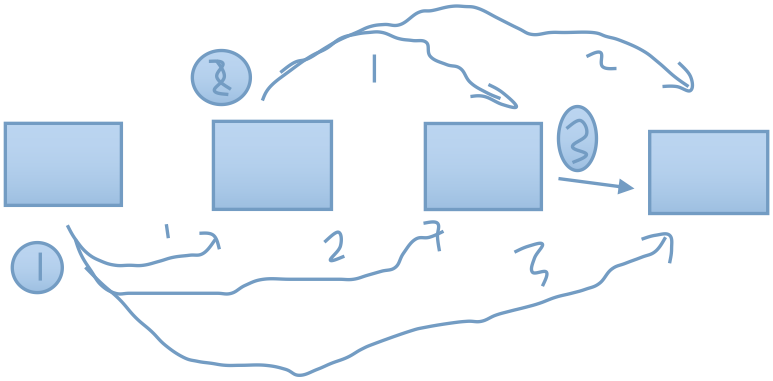
1.5.6. 排序

排序时候千万不要把循环内的 i，j 写错，否则会出现死循环

int a[N]

1.5.6.1. 冒泡排序

每次循环将最小值放到最前面的位置



```
for(i=0;i<N-1;i++){
        for(j=i+1;j<N;j++){       每次外循环把最小数放前面
                if(a[i]>a[j])        内循环将后面每个数与外循环数比较
                    swap(a[i],a[j]);
```

```
        }
}
```

1.5.6.2. 选择排序

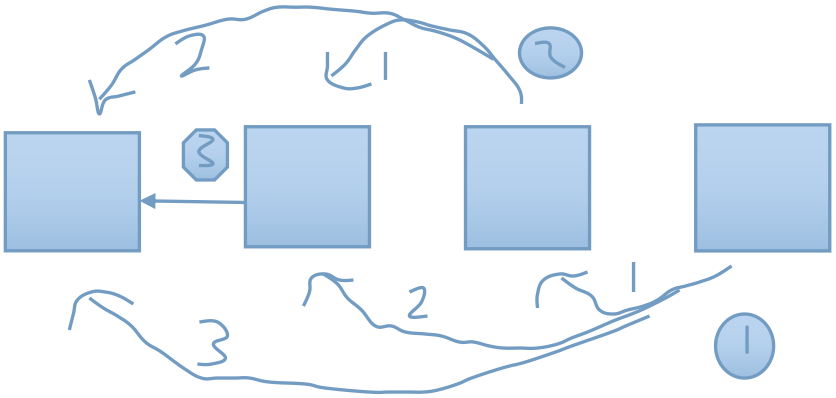
冒泡基础上加一个用来选择的下标变量，每次循环找出最小元素的下标，将这个元素与最前面的进行交换。

```
    for(int i=0;i<n-1;i++)
    {
        int min_index=i;    //用来存放每一圈里面的最小值的下标
        for(int j=i+1;j<n;j++)//每次 j 一圈时扫描选择出最小项
            if(arr[j]<arr[min_index])    min_index=j;
        if(min_index!=i)//用最小项交换，即将这一项移到列表中的正确位置
        {
                int temp;
                temp=arr[i]; arr[i]=arr[min_index]; arr[min_index]=temp;
            }
    }
```

1.5.6.3. 插入排序

把前面当成已排好的数组，进行插入。循环从下标 1 开始，每次循环

都是一位一位回退和插入，直到遇到前面大的（不需要插入的位置）。




```
for(j=1;j<N;j++){        每次外循环将当前数插入到前面
        num=a[j];  存储外循环上的数，用来插入到前面，同时用于比较
        for(i=j-1;i>=0;i--){  内循环找到插入位置
            if(a[i]>num) {
                a[i+1]=a[i];
                    如果前面大，则将其向后移动，留出插入位置
                a[i]=num;       插入到留出的位置
            }
            else    break;      如果前面没有比 num 大，退出循环
        }
}
```

1.5.6.4. 快速排序

每次循环先判断数组元素数量大于 0，使用一个基准数 num，一个头指针和一个尾指针，num 这个数在头指针和尾指针之间移动，头尾指针向中间移动。循环结束后 num 之前的都比 num 小，之后额都比 num 大

```
void  quick_sort(int  *a, int  N) {
    if (N < 2) return;

    int  *p_start = a;
    int  *p_end = a + N - 1;
    int  num = *p_start;   //基准数 num 为*p_start 的值
    while (p_end > p_start) {
        if (*p_start > *p_end)    swap(*p_start, *p_end);

        //num 在 p_start 和 p_end 的指向位
        //置上来回移动，两个指针向中间靠拢
        //移动那个没有指向 num 的指针
        if (*p_start != num)   p_start++;
        else   p_end--;
    }// 循环结束后，p_start 和 p_end 在同一位置上，且 num 位置排好

    quick_sort(a, p_start - a);   //排 num 之前的数组，计算出的数组中
    quick_sort(p_start + 1, (a + N - 1) - p_start); //排 num 之后的数组   数量也可写 n - (p_start - a + 1)
}
```

1.5.6.5. 库函数排序

```
int fun(const void *a,const void *b)
{
    return *(int*)a - *(int*)b; //升序
    return *(int*)b - *(int*)a;//降序
}
```

```
int a[10]={2,343,23,433,231,43,23,22,12,1};
qsort(a,10,sizeof(int),fun);
```

排序函数中也可以写：

```
if(*(int*)a > *(int*)b)
    return 1;     //返回 1 代表进行排序
return -1;
```

1.5.7. 递归

递归函数可以将函数中的数据分成若干分支，其中一定包含一种不需要调用递归函数的情况（结束深层次调用情况），剩余则是需要将数据进行深层次调用递归函数情况

Fun

结果 1　　结果 temp1（调用 fun）

结果 2　　结果 temp2（调用 fun）

结果 3　　结果 4

递归求阶乘

```
double fun(int i){
    if(i<=1) return 1.0;
    return i*fun(i-1);
```

}

1.5.8. 算法

二进制

1　2　4　8.....

可以相加组成 8 之前的所有数 1，2，3，4，5，6，7，8

得到数字没一位：取余，取整

exp: 532　　a=532%10

　　b=532%100/10

　　c=532/100

关于 10 取余可以得到 10 以内的数

判断一个整数二进制第三位置上是 0 还是 1

（data&4）==4　满足条件则是 1

　　与运算，4 的二进制为 100

取八进制的后四位

a%07777　　a%11　取 0 到 11 内的数

闰年

%4==0&&%100!=0||%400==0

奇数

1)%2!=0

2)等差数列　　1、3、5.....（i+=2）

3)通项公式　　i*2-1 或者 i*2+1

num=num*10+i

从高位数字开始，逐个加入数字组成数

统计任意输入数量数据的和

```
while(1){
    cin>>a;
    if(a==0)    break;
     sum+=a;
}
```

求整数位数

```
do{
    a[data%10]++;       数字 0 到 9，每出现一次，对应下标内的值加 1
     data/=10;
}while(data!=0);
```

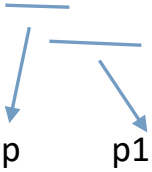
不断向一个数字加入尾数：

i=i*10+a;

判断一个字符是否是数字：

ch>='0'&&ch<='9'

1　2　3　5　7　13.....



second+=first

first=second-first　　p 变为 p1

p　　p1

a　(+7)%10　b　　加密

b　(+3)%10　a　　解密

计算一个数由两个数相加组成　　i<=a/2

计算一个数由两个数相乘组成　　i<=sqrt(a)

水仙花数：　n>=3　　153=1^3+5^3+3^3

a 取最后八位：　　a&0xFF

最大 int：　0x7F FF FF FF

0111　　1111

最高位为符号位

1.5.9. 原码/反码/补码

数字在存储时是二进制，对于有符号的类型，最高位 0 代表是正数，1 代表是负数

反码为原码逐位取反，补码为反码加 1

负数在计算和存储时，是使用补码进行的，-x 为~x+1，这样有利于在计算机中对两个数进行加法运算。

正数+负数 = 模　　（模就是该类型最大值）

-5+10 的运算过程（假设使用 8 位表示数）

0000 0101　　（5 的二进制）

1111 1011　　（-5 的二进制）

0000 1010　　（10 的二进制）

相加过程：

　　1111　1011　　（-5）

+　0000　1010　　（10）

= 1 0000　0101

去掉最高位溢出的 1，得到 0000 0101 也就是 5

1.5.10. 线性回归方程

```
void Line(double* pBufX,double* pBufY, const int nLen)
{
	double x,y,A=0.0,B=0.0,C=0.0,D=0.0,delta;

	for(int i=0;i<nLen;++i)
	{
	 x = pBufX[i];
	 y = pBufY[i];
	 A+=x*x;
	 B+=x;
	 C+=x*y;
	 D+=y;
	}
	delta = A*nLen-B*B;
	if(delta < PRECISION && delta > -PRECISION)
	{
	 TRACE("Divide Zero");
	 return ;
	}

	double k = (C*nLen-B*D)/delta;
```

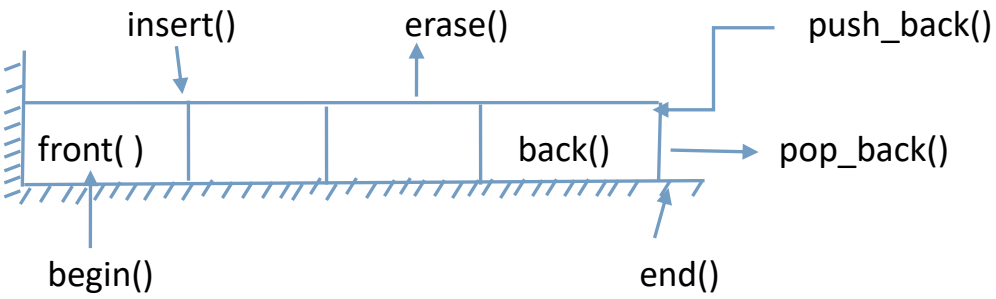
```
        double b = (A*D-C*B)/delta;
}
```

第二章 STL

2.1. STL

2.1.1. vector

支持对元素的下标访问，在尾部添加和删除元素效率高



======与数组转换

int a[5]={1,2,3,4,5}; v(a,a+5);

//a 代表 begin 迭代器 a+5 代表 end 迭代器（最后一个元素后面）

int *p = &a[0];

========赋值

vector<int> v;//定义一个空的 vector 对象

vector<int> v(5); //设置为 5 个 0，相当于调用 int 的默认构造

vector<int> v(5,10); 5 个 10

v.resize(5); //调整 vector 大小，如果原来有内容则保留原来内容，如果没有，则调用默认构造创建 （只对新分配的内存进行赋值）

v.resize(5,10);//调整大小，并对新分配的内容设置值

v.assign(5,3);//清空原有内容，并赋值为 3

list<int> l(3,5);

v.assign(l.begin(),l.end());//将 list 赋值给 vector

=======操作

v.reserve(100);//设置预留空间大小，与capacity相关

vector<int>::size_type size = v.capacity();

遍历取值

for(int i=0;i<v.size();i++) { cout<<v[i]<<endl; }

元素存取

v.push_back(x);

v.pop_back() ;

int end=v.back();

int begin = v.front();

try{x = v.at(100);}catch(...){}//比使用v[100]安全，可以抛出异常而不让

程序崩溃

v.clear();只清空内容，不释放分配的内存，内存到结束时释放，下标索引数量变为 0。因为内存内容还在，release 正常，debug 会崩溃。

v.swap(vector<int>());清空内容，释放内存

==========迭代器

任何可能导致容器结构发生变化的函数被调用后，先前获得的迭代器可能失效，需要重新初始化

erase 和 insert 操作单个元素时，都返回操作位置的迭代器。

删除[iterA – iterB)时，返回 iterB，vector 不可使用范围删除的迭代器返回值。

reverse_iterator　　rbegin()　　rend()

const_iterator

const_reverse_iterator

vector<int>::iterator　it;

定义了一个 vector 中的迭代器 it，迭代器是模版的共有内置类

通过迭代器遍历容器中元素：

```
for(it=v.begin();it!=v.end();it++)
        cout<<*it<<endl;
```

位置指针要用 it 迭代器

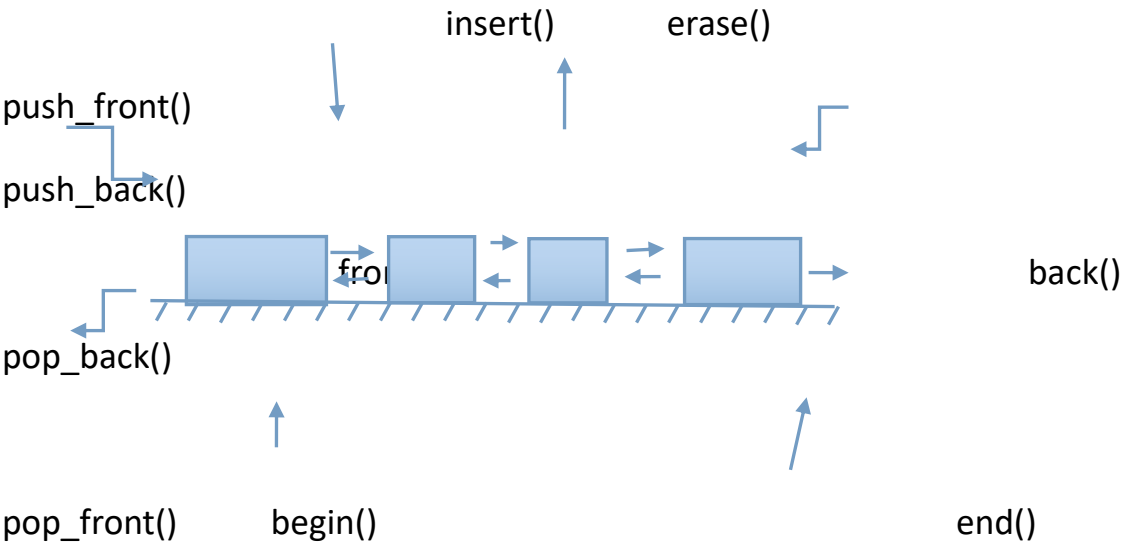
v.insert(位置指针，数量（可省略），数值)；

//在指针位置前插入，插入效率比 list 低

v.erase(位置指针)　　erase(开始位置，结束位置)

2.1.2. list

在任何位置插入和删除都很方便，不支持元素下标访问



除了不支持 vector 的下标操作外，和 vector 的初始化、插入、删除等函数的使用相同。

list.sort(); //可以让 list 中的元素从小到大自动排序，如果是元素是类类型，需要重载＜号

//删除满足某个条件的元素

bool equal(int &temp){return temp == 3;}

l.remove_if(equal);

```
l.remove_if([](int &temp)->bool{return temp == 3;});
```

2.1.3. map

以 key 的升序（插入时排序）存储 key_value 对的序列，每个 key 只能出现一次，根据 key 提取 value 的效率最高，排序根据 key 类型的‘<’操作符对其进行比较

map<string,int> m;

m["a"]=1;　　//m[**]可以访问到 key 对应的成员（[]操作符返回值相当于 value），同时也是进行检查然后插入 key 的操作。

m.insert(pair<string,int>("a",1));

m.insert(map<string,int>::value_type("b",2));

map<int,int> m;

pair<map<int,int>::const_iterator,bool>　　result=

m.insert(map<int,int>::value_type(1,2));//insert 的返回值第二个表示是否插入成功，第一个参数指向准备插入的内容。

如果要连续调用 map，可以使用引用，减少插入 key 的检查操作：

```
struct       A{int    x, int    y};

map<int    , map<int ,A>>        m;

map<int    , map<int ,A>> &mx = m[1][0];

mx.x = 1;

mx.y = 2;


map<string,int>::iterator    it;

==========遍历

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

{     cout<<it->first<<endl<<it->second<<endl;    }

==========查找

it=m.find("a");

if(it!=m.end())

{     cout<<m["a"]<<endl;    }


iterator erase(iterator it); //通过一个条目对象删除

iterator erase(iterator first, iterator last); //删除一个范围

size_type erase(const Key& key); //通过关键字删除，返回的是删除的个数，只会是 0 或者 1 。multimap 值可以是大于 1。


========== multimap

multimap<int,string> m;
```

```
int key = 1;
m.insert(pair<int,string>(key,"a"));
m.insert(pair<int,string>(key,"aa"));//相同键可以对应多个值
m.insert(pair<int,string>(2,"b"));
//查找并遍历某个键的值（通过 count 统计某个键对应的值的个数）
multimap<int,string>::const_iterator iter = m.find(key);
for(size_t count = 0; count != m.count(key); ++count,++iter)
{     cout<<iter->second<<endl; }
```

C11 初始化：

```
map<int, int> mm = {{1,2},{3,4} };
```

2.1.4. bitset

```
#include <bitset>
using namespace std;
```

左边存储序列低位，右边存储高位

```
bitset<10>    b(8);
 //能存储 10 个位数据，用 8 进行初始化。也可以用 b=8
//存储内容为 0001000000
for(int i=b.size()-1 ; i>=0 ; i--)
{     cout<<b[i]<<endl;     }
```

string s("1100");

bitset<10> b(s); //也可以使用 string 串形式的二进制数字初始化

可以进行位操作： bitset<10> b(5); bitset<10> a(2); b=a|b;

any() 判断是否存在 1

none() 是否全 0

count() 1 的个数

os<<b 输入到 os 流，流内会正向排序

flip() 逐位取反，flip(pos)只对某位进行取反

test(pos) 测试 pos 位置是否为 1

set() 全部置 1，可用 set(pos)

reset() 全部置 0，可用 reset(pos)

2.1.5. 其他容器

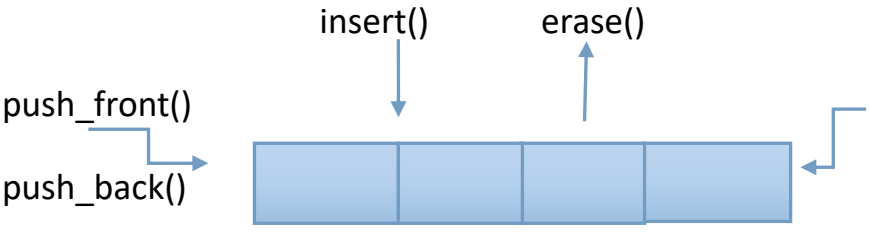
deque 和 stack 可以做适配器，stack 可以用 deque 初始化

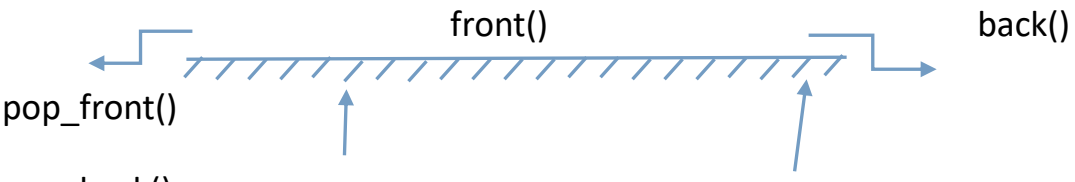
deque<int> d(v.begin(),v.end());

stack<int> s(d);

==========双端队列(deque)

支持对元素下标访问，支持在两端添加或者删除元素

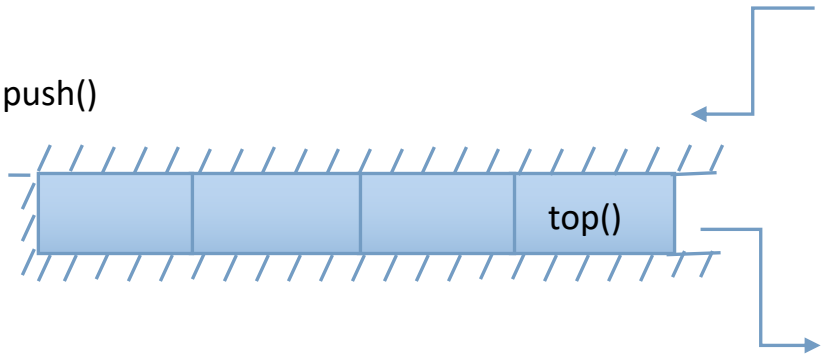




begin()　　end()

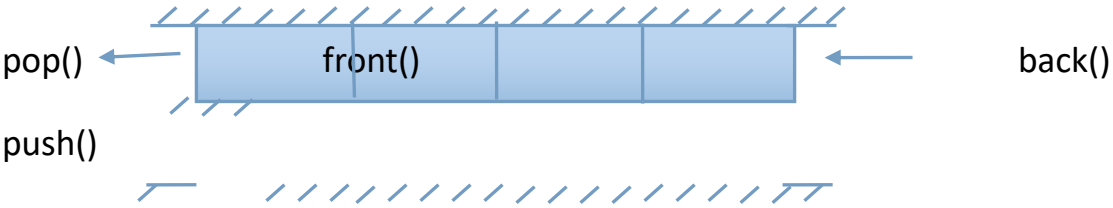
==========堆栈（stack）

只支持在一段存储和提取元素，无 iterator



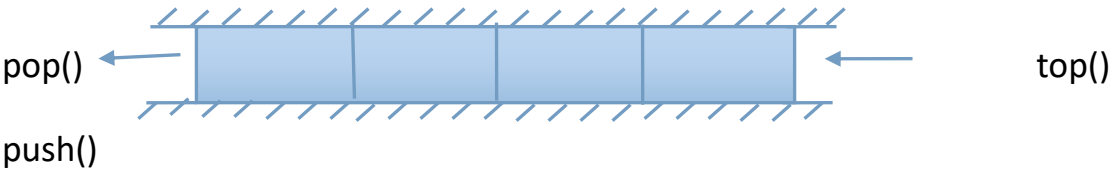
==========队列（queue）

支持从前端提取，从后端插入元素



==========优先队列（priority_queue）

类似于队列，但所提取的是具有最高优先级的元素



==========集合（set）

映射的低级形式，不需为每个 key 指定一个 value

s.insert(1);　　//直接放入元素

*it　　可以直接对迭代器取值

==========多重映射（multimap）

映射的通用形式，一个 key 可以拥有超过一个的 key_value 对

==========多重集合（multiset）

多重映射的低级形式，不需要为每个 key 指定一个 value

2.2. 循环队列

```
#define QueSize    11
class MyQue
{
public:
	MyQue()
	{
		dat.resize(QueSize,0);

		front = back = 0;
	}
	bool push(int* element)
	{//向队列插入元素
		if ((back + 1) % QueSize == front) //判断队列是否满
		{
			return false;
		}
		else
		{
			dat[back] = *element;
			back = (back+1) % QueSize;
			return true;
```

```
        }
    }
    int* pop()
    {//从队顶弹出元素
        int *pRetElement = NULL;

        if (front != back)//判断队列是否为空
        {
            pRetElement = &dat[front];

            front = (front + 1) % QueSize;
        }

        return pRetElement;
    }
public:
    int AvailableLength()
    {//返回队列的可以用空间长度
        return QueSize - 1 - (back - front + QueSize) % QueSize ;
    }
    int UsedLength()
    {//返回队列的已使用空间长度
            return (back - front + QueSize) % QueSize;
    }
private:
    int front; //对头指针
    int back;   //队尾指针
    vector<int> dat;
};
```

2.3. <algorithm>

2.3.1.1. 排序

如果 stl 的成员变量是类类型，需要重载类的<号

```
//假设要比较 A 内的 int   x;成员
bool operator<( const   A& other) const
{
```

```
        if(x<other.x)
                return    true;     //代表交换两个元素
         else
            return false;
}
```

对数据排序:std::sort(v.begin(),v.end());

插入时排序：v.insert(std::upper_bound(v.begin(),v.end(),a1),a1);

2.3.1.2. 距离

size_t i = std::distance(v.begin(),v.end());

可以返回两个迭代器之间的距离（此处返回了 v 的长度）

2.3.1.3. 去重

std::sort(v.begin(),v.end());

//将数据排成1,2,2,3,3,4,5

vector<int>::iterator iter = std::unique(v.begin(),v.end());

//将不重复的元素向前赋值，数据变成 1,2,3,4,5,4,5，并返回 4 的位置（返回不重复位置的末尾）

v.erase(iter,v.end());

或者：std::sort(v.begin(),v.end());

v.resize(std::distance(v.begin(),std::unique(v.begin(),v.end())));

2.3.1.4. 查找

vector<int>::iterator iter = std::find(v.begin(),v.end(),10);

查找到则 iter 执行找到的元素，查找不到则指向 v.end();

如果是类类型，则需要重载==符号。

2.3.1.5. 智能指针

std::auto_ptr<A> p(new A());

p->fun(); //使用指针

指针A在不需要使用时，会自动释放

p.reset(); //手动释放指针内存

p.reset(pb);//释放之前的指针，同时让p接管另一个指针

if(0 != p.get()) {p->fun();}

对智能指针进行非空判断

2.4. lambda 表达式

lambda 表达式的参数列表基本和函数的一致，不过有如下限制：

参数列表不能有默认参数

不能是可变参数列表

所有的参数必须有个变量名

遍历每个元素，让该元素*10；

struct A{ void operator()(int &x){x = x*10;} }; //需要重载()

std::for_each(v.begin(),v.end(),A()); // vector<int> v;

//使用 lambda

for_each(v.begin(), v.end(), [](int &a) { a *= 10;});

```
struct Data
{
    Data():a(1),b(1){}
    int a;
    int b;
};
vector<Data>  v(5);
```

for_each(v.begin(), v.end(), [](Data &dat) { dat.a *= 10;});//访问每个元素的引用

for_each(v.begin(), v.end(), [](Data dat) {cout << dat.a << endl;});//访问每个元素的值

//////////访问外部变量

int sum = 0;

for_each(v.begin(), v.end(), [&](Data &dat) { sum += dat.a ;});//按引用访

问外部元素

int temp = 100;

for_each(v.begin(), v.end(), [=](Data dat) {cout << dat.a * 100 << endl;});//按值访问外部元素

//作为参数传递

```
template<typename _Fun>
void fun(_Fun callback)
{
    int x = 10;
    callback(x);
}
```

调用：

```
    int factor = 10;
    fun([&] (int &x){
        cout << factor* x << endl;
    });
```

//map 访问方式

```
map<int, Data> m;
m[1] = Data();
m[2] = Data();
```

for_each(m.begin(), m.end(), [](pair<int, Data> dat) { cout << dat.first << " " << dat.second.a << endl;});

//返回值

```
int z = []()->int { return 10; }();
```

()->int 可以省略

//设置数组的值

```
const int SIZE = 20;
int array[SIZE];
generate_n(array, SIZE,[]() { return rand() % 10; });
```

2.5. <functional>

```
int fun1(int a,double b){return a + b;}
auto fun2 = [](int a, double b) {return a - b; };
class Functor
{
public:
    int operator() (int a,double b){return a*b;}
};
class ABC
{
    int x;
public:
    ABC() { x = 1000; }
    int fun3(int a, double b) { return a/b + x; }
};
void __fun(int a,  std::function<int(int, double)> fun)
{
    double  dd = 100.2;
    int ret = fun(a, dd);
    cout << ret << endl;
}


    __fun(1, fun1);
    __fun(1, fun2);
    int x = 5;
    __fun(1, [&](int a, double b) {return x + a + b; });
    __fun(1, Functor());
    ABC aa;
    std::function<int(int,double)>  obj = std::bind(&ABC::fun3, &aa,
std::placeholders::_1,std::placeholders::_2);
    __fun(1, obj);
```

2.6. <memory>

auto_ptr<A>　　pa(new　A);

A　*p = pa.get();

不能传入数组，数组要用 delete[]

2.7. 模板

template　<typename　　T>

此行下的是模版，且有未实例化的 T，<>中间的是类型形参表

当类型形参表多个时：<typename　T, typename　F>

模版不能像普通函数那样在.h 中只声明，.cpp 文件中实现。因为不是一个具体类型，所以只声明会出错，因此只能将声明和实现都放在头文件中

typename Element::ID_TYPE　id;

可以直接使用另一个模版类中的类型。

2.7.1. 函数模版

template　<typename　　T>

void　　ab(T　a, T　b){1}

函数必须要传入 T 才可以使用（不可以在内部定义 T）

1. ab(100,200)　　默认隐式推断（类模版不能用此法）

2. ab<int>(100,200)

特化：（相当于重载，函数名相同）

template<>　代表特化后的模版内无剩余的未实例化的参数

void　ab(int a,　int　b){2}　　用 int 特化

2.7.2. 类模版

```
template<typename T>
class　A{
　　T　x;
　　void　ab(){1}
};
```

使用时：　A<int>　　a;

```
template<>
class　A<int>{
　　int　x;
　　void　ab(){2}
};
```

类模版不是类，因为有未定义的 typename T，要实例化（编译时）T 才是类。编译时，无栈参与，栈在运行时参与，变量存于栈内，常量存代码段

2.7.3. 常量模板

非类型实参只能是常量性质

```
template<int    a，int    b>
class    A
{
        void fun(){int x = a;      x += b ; cout<<x<<endl;}
};
```

可以在类内使用未知的常量

2.7.4. 模板的派生

基模版派生子模版：

```
template    <typename    T>
class    A{T    t;.....};
template    <typename    T>
class    B:public    A<T>{.....};
```

子模版继承父类：

子类：template <typename   T，typename   Base>

```
        class    B:public    Base{....};
```

父类：class   A{....};

使用时：   B<int ,A>   b;

父类如果是模版类时：

子类改为：

```
template<typename    T,    template<typename F>Class    Base,
                    typename    BA>
class    B:public Base<BA>{.....};
```

用 BA 使 Base 变为类

模版中定义模版函数：

```
template <typename    T>
class    A{public:
              .....
              template<typename    T1>
              void ab(T1    b){.....}
};
```

使用时：

```
A<int>     ().ab<int>(3);
```

创建一个临时对象，在调用 ab 模版函数（用 int 特化）

可用 `A<int>     a;`

此时其中模版函数不可用

类外定义时：

```
template <typename    T>
    template<typename    T1>    模版下的模版
    void    A<T>::ab(T1    b){.....}
```

类中类：

A<int> a;

A<int>::B<long> b;

2.7.5. 特化

当类型参数表有多个时，进行多种特化后，在使用时选择特化程度最高的

针对指针的特化：

template <typename T>

class A<T*>{}

template <typename T>

class A<T[]>{}

2.7.6. 外部实现

可以内部定义，外部实现（同一个.h 文件中）

template <typename T>

void A<T>::ab(T t){}

类的成员特化：

template<>

void A<char>::ab(T t){}

2.7.7. 模版默认值

typename T=int 可以使用默认值

2.7.8. 模版中获取类型

```
template<typename T>
class A
{
    typename T::TY b;
};



class X
{
    int  a;
public:
    typedef int TY;
};
```

第三章 VC

3.1. 数据类型

3.1.1. Windows 数据类型

BYTE 是 1 个字节（char），8 位
（0[0x00]到 255[0xff],能指向 256 个 bit 位，也就是 256Bit 内存大小）
WORD 是 2 个字节（Unsigned short），16 位（0 到 65535）
DWORD 是 4 个字节（Unsigned long）,32 位（0 到 2^{32}-1)【可以指向 4GB 内存】
HANDLE 唯一标识一个对象（void *）

3.1.2. Unicode 和多字节

3.1.2.1. 多字节

```
char    c='A';
char *p="hello";
char a[10];
char a[]="hello";
```

字符串函数：
strlen

3.1.2.2. Unicode

```
typedef    unsigned short    wchar_t;
wchar_t    c='A';
wchar_t    *p=L"hello";
wchar_t     a[]=L"hello";
```

字符串函数:
wcslen

3.1.2.3. 综合方案

```
#ifndef    UNICODE
typedef    WCHAR    TCHAR, *PTCHAR;
```

typedef LPWSTR LPTCH,PTSTR,LPTSTR;
typedef LPCWSTR LPCTSTR;
#define TEXT(quote) L##quote //或者是_T
#else
typedef char TCHAR, *PTCHAR;
typedef LPSTR LPTCH,PTSTR,LPTSTR;
typedef LPCSTR LPCTSTR;
#define TEXT(quote) quote
#elseif
因为有了这样的定义，所以在使用时，可以直接用 TCHAR 作为基本的字符数据类型，在写字符串时可以写_T(“hello”)。

#ifdef UNICODE
#define MessageBox MessageBoxW
#else
#define MessageBox MessageBoxA
#endif
所有和文字处理相关的函数都有 W 和 A 版本，可以显示调用，也可以直接调用函数原型，通过以上的定义自动转换为需要的类型。

Windows 中定义的通用版基本字符串处理函数：
lstrlen, lstrcpy, lstrcpyn, lstrcat, lstrcmp, lstrcmpi
_stprintf

3.1.3. LPARAM

DWORD x=100;

DWORD y=200;

LPARAM lParam=MAKELPARAM(x,y);

3.2. WindowsThread

3.2.1. 线程

```
DWORD WINAPI fun(LPVOID param)
{
    int x = *(int*)param;
```

```
    cout <<  x << endl;
    return 0;
}
```

//创建并运行线程

```
int param = 10;
HANDLE handle = CreateThread(NULL, 0, fun, &param, 0, NULL);

//创建使不运行，程序挂起
HANDLE handle = CreateThread(NULL, 0, fun, &param, CREATE_SUSPENDED, NULL);
ResumeThread(handle);//恢复运行
```

//等待线程运行结束

```
WaitForSingleObject(handle, INFINITE);
```

//关闭线程

```
TerminateThread(handle, 0);
CloseHandle(handle);//句柄资源需要释放
```

//挂起、恢复线程

```
SuspendThread(handle);
ResumeThread(handle);
```

对于一个线程，挂起操作几次就要恢复操作几次
SuspendThread，ResumeThread 返回值是该线程剩余挂起次数，-1 代表线程为 NULL

```
//确保线程重启
while (true)  if (ResumeThread(thr) <= 0) break;


#include <process.h>
UINT threadId;
(HANDLE)_beginthreadex(NULL, 0, ListenThread, NULL, 0, &threadId);
UINT WINAPI ListenThread(void* pParam)
```

3.2.2. 线程同步

当对象被占用时 WAIT_TIMEOUT == WaitForSingleObject(h,0)

WaitForSingleObject(h, INFINITE)无限制等待，执行成功后占用对象

3.2.2.1. 互斥对象

互斥对象上几次锁就要调用几次 ReleaseMutex。上锁行为包括创建锁时的占用，以及执行 WaitForSingleObject 成功。

HANDLE hMutex; //要被多个线程调用

hMutex=CreateMutex(NULL,FALSE,NULL);//初始化，最后一个参数代表是无名的互斥对象，FALSE 代表当前线程创建锁时互斥对象没被占用（如果为 TRUE，表示创建的同时占用互斥对象），也就是其它线程可以使用。

::WaitForSingleObject(hMutex,INFINITE);//运行线程时，先运行这一句，每个线程前面都有这一句，就可以保证其中一个线程没结束，或者没有调用 ReleaseMutex 函数，其它线程代码不能运行，INFINITE 代表超时值为一直等待

ReleaseMutex(hMutex); //当线程运行时间久，不同线程交替运行，但只是保护循环中某处的代码时可以用这个，而不用等待线程结束自动释放互斥对象

可以使用有名互斥对象判断进程是否已经执行

if(CreateMutex(NULL,TRUE,"xx"))

if(ERROR_ALREADY_EXISTS==GetLastError()){已经执行}

3.2.2.2. 关键代码段

类似互斥对象和事件对象，这两个对象是内核对象，速度较慢。关键代码段工作在用户方式下，速度快，但无法设置超时值，容易造成死锁。

CRITICAL_SECTION cs;//要被多个线程调用，所以一般为全局变量

InitializeCriticalSection(&cs);//创建

DeleteCriticalSection(&cs);//释放

线程使用时:

EnterCriticalSection(&cs);

。。。。。。要保护的代码内容。。。。。

LeaveCriticalSection(&cs);

//线程结束不会自动释放，必须要用代码

3.2.2.3. hEvent

HANDLE hEvent;

hEvent=CreateEvent(NULL,TRUE,FALSE,NULL);//TRUE 代表创建的是手动事件，FALSE 代表创建时未占用（ResetEvent 状态,内核占用该对象）

SetEvent 将对象从内核拿出并占用

ResetEvent 将对象还给内核

多次调用 SetEvent(hEvent)不会有影响

3.2.2.3.1. 事件控制

```
DWORD WINAPI fun(LPVOID p)
{
    while (true)
    {
        WaitForSingleObject(hEvent, INFINITE);

        //要运行的代码

        ResetEvent(hEvent);
    }
    return 0;
}
```

判断处于 ResetEvent 状态（说明已经执行到线程末尾的 ResetEvent 处，线程处于空闲状态），然后再控制线程开始运行：

```
if (WAIT_TIMEOUT == WaitForSingleObject(hEvent, 0))
{  SetEvent(hEvent);   }
```

WAIT_OBJECT_0 ：核心对象已被激活

WAIT_TIMEOUT ：等待超时

WAIT_FAILED ：出现错误

3.2.2.3.2. 信号控制

可以被多个线程调用
相当于 bool 值,通过 HasSignal()判断有无信号

```
struct Signal
{
	Signal()
	{
		m_hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
	}
	bool HasSignal()
	{
		return WAIT_TIMEOUT != WaitForSingleObject(m_hEvent,0);
	}
	void SetSignal()
	{
		SetEvent(m_hEvent);
	}
	void DelSignal()
	{
		ResetEvent(m_hEvent);
	}
	HANDLE m_hEvent;
};
```

3.3. WindowsNetWork

3.3.1. Socket

Send等函数返回值为==SOCKET_ERROR，代表发送失败。

Socket是单向的，服务器和客户端各自维护一个文件。
该文件可以由本端写入，另一端读取。
本端是在内存区域不断放入，另一端是不断从内存区域拿走。

3.3.1.1. 初始化

```
#include <windows.h>//#include <Winsock2.h>
#pragma comment(lib,"ws2_32.lib")
```

```
WSADATA wsaData;
WORD wVersionRequested = MAKEWORD( 2, 2 );
if (0 != WSAStartup( wVersionRequested, &wsaData ))
{
        return;
}
if ( LOBYTE( wsaData.wVersion ) != 2 ||HIBYTE( wsaData.wVersion ) != 2 )
{
        WSACleanup( );//清理数据
        return;
}
```

3.3.1.2. TCP

3.3.1.2.1. Server

```
    //创建 socket
    SOCKET sockSrv=socket(AF_INET,SOCK_STREAM,0);
    SOCKADDR_IN addrSrv;
  addrSrv.sin_addr.S_un.S_addr=htonl(INADDR_ANY);
  addrSrv.sin_family=AF_INET;
  addrSrv.sin_port=htons(5555); //端口号

    bind(sockSrv,(SOCKADDR*)&addrSrv,sizeof(SOCKADDR));
    listen(sockSrv,5); //最多能接受 5 个客户端链接，如果是 SOMAXCONN，则自动设置为合理的最大值

                while(1)
    {
          SOCKADDR_IN addrClient;
        int len=sizeof(SOCKADDR);

        //接受链接
        SOCKET sockConn=accept(sockSrv,(SOCKADDR*)&addrClient,&len);
        //发送
        char sendBuf[100]={0};
        sprintf(sendBuf,"%s",inet_ntoa(addrClient.sin_addr));
        send(sockConn,sendBuf,strlen(sendBuf)+1,0);
        //接收
              const int bufSize = 100;
        char recvBuf[bufSize ] = {0};
        recv(sockConn,recvBuf,bufSize ,0);//返回值为接收到的实际大小,网络错误
```

```
时，返回 SOCKET_ERROR
        printf("%s\n",recvBuf);
        //关闭
        closesocket(sockConn);
    }
```

3.3.1.2.2. Client

客户端在第二次连接时，需要重新定义 SOCKET sockClient

```
    SOCKET sockClient=socket(AF_INET,SOCK_STREAM,0);
    SOCKADDR_IN addrSrv;
    addrSrv.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");
    addrSrv.sin_family=AF_INET;
    addrSrv.sin_port=htons(5555);
if(SOCKET_ERROR==connect(sockClient,(SOCKADDR*)&addrSrv,sizeof(SOCKADDR)))
{return ;}
  //接收
            const int bufSize = 100;
    char recvBuf[bufSize ]={0};
    recv(sockClient,recvBuf,bufSize ,0);
    printf("%s\n",recvBuf);
   //发送
   send(sockClient,"clientSend",strlen("client")+1,0);
  //关闭
    closesocket(sockClient);
    //WSACleanup();
```

3.3.1.3. UDP

3.3.1.3.1. Server

```
    SOCKET sockSrv=socket(AF_INET,SOCK_DGRAM,0);
    SOCKADDR_IN addrSrv;
    addrSrv.sin_addr.S_un.S_addr=htonl(INADDR_ANY);
    addrSrv.sin_family=AF_INET;
    addrSrv.sin_port=htons(5555);

    bind(sockSrv,(SOCKADDR*)&addrSrv,sizeof(SOCKADDR));
//不调用 listen 函数

    SOCKADDR_IN addrClient;
```

```
int len=sizeof(SOCKADDR);
char recvBuf[100];
 //接收
recvfrom(sockSrv,recvBuf,100,0,(SOCKADDR*)&addrClient,&len);
printf("%s\n",recvBuf);

closesocket(sockSrv);
WSACleanup();
```

3.3.1.3.2. Client

```
SOCKET sockClient=socket(AF_INET,SOCK_DGRAM,0);
SOCKADDR_IN addrSrv;
addrSrv.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");
addrSrv.sin_family=AF_INET;
addrSrv.sin_port=htons(5555);

//不调用 connect 函数
sendto(sockClient,"client",strlen("clinet")+1,0,(SOCKADDR*)&addrSrv,sizeof(SOCKADDR));

closesocket(sockClient);
WSACleanup();
```

3.3.1.4. 文件传输

3.3.1.4.1. 客户端（发送）

```
string fileName = "1.jpg";
string filePath = "D:\\";
ifstream ifile(filePath + fileName,ifstream::binary);
ifile.seekg(0, ios::end);
std::streamoff fileSize = ifile.tellg();
ifile.seekg(0,ios::beg);
if (!ifile)   return;

stringstream ss;
ss << fileName << " " << fileSize;
string fileInfo = ss.str();
send(sockClient, fileInfo.c_str(), fileInfo.size() + 1, 0);//发送文件名及文件大小
```

```
const int BufferSize = 1024;
int readSize = 0;
char SendBuf[BufferSize] = {0};
while (true)
{
    ifile.read(SendBuf, BufferSize);

    readSize = (int)ifile.gcount();
    if (readSize == 0)  break;
    send(sockClient, SendBuf, readSize, 0);
}
ifile.close();

//等待服务器接收成功
char EndMessage[1024];
recv(sockClient, EndMessage, 1024, 0);
cout << EndMessage << endl;
```

3.3.1.4.2. 服务器（接收）

```
int recSize = 0;

//获取文件名和大小
char FileInfo[1024] = { 0 };
recSize = recv(sockConn, FileInfo, 1024, 0);
if (recSize == 0) return;
stringstream ss(FileInfo);
string FileName;
std::streamoff fileSize = 0;
ss >> FileName >> fileSize;

//创建文件
string filePath = "E:\\";
filePath += FileName;
ofstream ofile(filePath,ofstream::binary);
if (!ofile) return;

//获取文件内容
std::streamoff RecvfileSize = 0;
const int BufferSize = 1024;
char RecvBuff[BufferSize] = {0};
while (true)
{
    recSize = recv(sockConn, RecvBuff, BufferSize, 0);
```

```
    if (recSize == SOCKET_ERROR) { cout << "erro " << endl;  return; }

    ofile.write(RecvBuff, recSize);
    RecvfileSize += recSize;

    if (RecvfileSize == fileSize) break;//接收完文件
}

send(sockConn, "suc", 4, 0);

ofile.close();
```

3.3.1.5. 检测 socket 是否可写

```
timeval tm = {0,100};
fd_set fd_write;
FD_ZERO(&fd_write);
FD_SET(*pSocket,&fd_write);
int iSelEro = select(*pSocket + 1 ,NULL,&fd_write,NULL,&tm);
```

3.3.2. 网络基本应用

3.3.2.1. 获取主机名/IP 地址

需要初始化 socket

```
char* name=new char[50];
gethostname(name,50);

//通过主机名称获取 ip 地址
char* ip=new char[50];
struct hostent    *pHostent=gethostbyname(name);
if(pHostent!=NULL)
{
    char* lpAddr=pHostent->h_addr_list[0];
    if(lpAddr!=NULL)
    {
        struct in_addr inAddr;
        memmove(&inAddr,lpAddr,4);
        ip=inet_ntoa(inAddr);//转换成标准的 ip 地址形式
    }
}
```

3.3.2.2. 获取 MAC

```
#include <IPHlpApi.h>
#pragma comment(lib,"Iphlpapi.lib")

DWORD AdapterInfoSize=0;
DWORD    Err=GetAdaptersInfo(NULL,&AdapterInfoSize);
if(Err!=0&&Err!=ERROR_BUFFER_OVERFLOW)
{
    TRACE("获取网卡信息失败");
    return ;
}
//分配网卡信息内存
PIP_ADAPTER_INFO pAdapterInfo=(PIP_ADAPTER_INFO)GlobalAlloc(GPTR,AdapterInfoSize);
if(pAdapterInfo==NULL)
{
    TRACE("分配网卡信息内存失败");
    return ;
}
if(GetAdaptersInfo(pAdapterInfo,&AdapterInfoSize)!=0)
{
    TRACE("获取网卡信息失败");
    GlobalFree(pAdapterInfo);
    return ;
}

CString mac;
mac.Format("%02X-%02X-%02X-%02X-%02X-%02X",pAdapterInfo->Address[0],pAdapterInfo->Ad
dress[1],pAdapterInfo->Address[2],pAdapterInfo->Address[3],pAdapterInfo->Address[4],pAdapte
rInfo->Address[5]);
```

3.3.2.3. 获取远程主机 MAC

```
#include <IPHlpApi.h>
#pragma comment(lib,"Iphlpapi.lib")

char *addr ="192.168.10.24";
u_char mac[6]={0xff,0xff,0xff,0xff,0xff,0xff};
ULONG ulen=6;
if(::SendARP(::inet_addr(addr),0,(ULONG*)mac,&ulen)==NO_ERROR)
{
    CString str;
    str.Format("%02X-%02X-%02X-%02X-%02X-%02X",mac[0],mac[1],mac[2],mac[3],mac[4],ma
```

c[5]);
}

3.4. 窗口

3.4.1. 程序结构

结构：
Winmain（.....）
{

……
注册窗口类
CreateWindowEx(……窗口类….窗口过程函数……..)

While(消息循环){……}
}

3.4.2. 对话框

3.4.2.1. 程序结构

```
INT_PTR CALLBACK DlgProc(HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam);

HINSTANCE hgInst;
int WINAPI WinMain(HINSTANCE hThisApp, HINSTANCE hPrevApp, LPSTR lpCmd, int nShow)
{
    hgInst = hThisApp;
    HWND hwnd = CreateDialog(hThisApp, MAKEINTRESOURCE(IDD_DLG), GetDesktopWindow(),
(DLGPROC)DlgProc);
    if (!hwnd) return 0;
    ShowWindow(hwnd, nShow);

    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return 0;
```

```
}
INT_PTR CALLBACK DlgProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch (msg)
    {
    case WM_INITDIALOG:
        SendMessage(hwnd, WM_SETICON, ICON_SMALL, (LPARAM)LoadIcon(hgInst,
MAKEINTRESOURCE(IDI_ICON)));
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
       default:
        return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}
```

3.4.2.2. 获取控件句柄

```
HWND wnd1 = GetDlgItem(hwnd, IDC_View0);
```

3.4.3. 资源加载

3.4.3.1. 对话框窗口

```
DialogBox(hInstance, MAKEINTRESOURCE(IDD_DLG), hWnd, msgProcFun);
```

3.4.3.2. 字符串资源

```
LoadStringW(hInstance, IDS_APP_TITLE, szTitle, 100);
```

3.4.3.3. 菜单

```
MAKEINTRESOURCEW(IDC_MENU);
```

对应：WM_COMMAND

```
int cmdId = LOWORD(wParam);
```

3.4.3.4. 快捷快捷键

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_ACC));

对应 TranslateAccelerator 函数

3.4.4. 外观

3.4.4.1. 去除边框

```
HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_VISIBLE,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);
```

```
SetWindowLong(hWnd, GWL_STYLE, GetWindowLong(hWnd, GWL_STYLE) & ~WS_CAPTION);
```

3.4.4.2. 全屏

方法一：

```
ShowWindow(hWnd, SW_MAXIMIZE);
```

方法二：

```
int cx = GetSystemMetrics(SM_CXSCREEN);
int cy = GetSystemMetrics(SM_CYSCREEN);
MoveWindow(hWnd, 0, 0, cx, cy, TRUE);
```

或:

```
HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_VISIBLE,
    CW_USEDEFAULT, 0, cx, cy,
    nullptr, nullptr, hInstance, nullptr);
```

3.4.5. 背景色

```
wcex.hbrBackground = CreateSolidBrush(RGB(0, 0, 0));
```

3.4.6. 双击功能

注册窗口类时加入属性 CS_DBLCLKS

3.4.7. 控件

3.4.7.1. 滚动条

```
HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW|WS_VSCROLL,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);
```

3.4.7.2. 按钮

```
HWND BtnWnd = 0;
BtnWnd = CreateWindow(TEXT("BUTTON"), TEXT(""), WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
            10, 10, 100, 30,
            hWnd, NULL, (HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE), NULL);


WM_COMMAND
if (lParam == (UINT)BtnWnd)
{
    MessageBox(hWnd, TEXT("Test"), TEXT("Test"), MB_OK);
}
```

3.4.7.3. 静态框

```
CreateWindow(TEXT("STATIC"), TEXT(""), WS_VISIBLE | WS_CHILD | SS_BLACKFRAME,
            10, 10, 100, 30,
            hWnd, NULL, (HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE), NULL);
```

3.4.8. 消息

3.4.8.1. WM_SIZE

```
int cx = LOWORD(lParam);
int cy = HIWORD(lParam);
```

3.4.8.2. WM_MOUSEWHEEL

```
int zDelta = GET_WHEEL_DELTA_WPARAM(wParam);
```

3.4.8.3. WM_VSCROLL

```
UINT nPos   = HIWORD(wParam);
UINT nSBCode   = LOWORD(wParam);
```

3.5. 绘图

3.5.1. 画刷

HBRUSH hbr = CreateSolidBrush(RGB(0, 0, 0));

HBRUSH hbr = (HBRUSH)GetStockObject(NULL_BRUSH);//空画刷

3.5.2. WM_PAINT

PAINTSTRUCT ps={0};

HDC hdc=BeginPaint(hWnd,&ps); //该函数让无效区域背景被擦除

TextOut(hdc,x,y,"paint",5);

EndPaint(hWnd,&ps);

//BeginPaint 和 EndPaint 函数成对结束后，窗口区域才变得有效

3.5.3. WM_PAINT 外

HDC hdc=GetDC(hwnd);//如果 hwnd 为 NULL，则代表桌面

TextOut(hdc,x,y,"paint",5);

ReleaseDC(hwnd,hdc);

//这对函数属于在某个指定区域绘图，不会让无效区域变得有效，也不一定是在无效区域绘图，不像 WM_PAINT 时需要收集足够多的绘图资讯。要显示地使无效区域变得有效。

3.5.4. 获取字体大小

```
TEXTMETRIC tm;

HDC hdc=::GetDC(hWnd);

GetTextMetrics(hdc,&tm);

ReleaseDC(hWnd,hdc);
```

可以从 tm 结构体中获取系统字体信息

3.6. GDI+

3.6.1. 初始化和释放

```
#include <comdef.h>
#include <GdiPlus.h>
#pragma comment(lib,"gdiplus.lib")
using namespace Gdiplus;


=======旧版本
#ifndef ULONG_PTR
#define ULONG_PTR unsigned long
#endif
```

初始化：

```
GdiplusStartupInput gdiplusStartupInput;
ULONG_PTR    pGdiToken;
GdiplusStartup(&pGdiToken,&gdiplusStartupInput,NULL);
```

释放：

```
if (pGdiToken>0)      GdiplusShutdown(pGdiToken);
```

3.6.2. Image

3.6.2.1. GetEncoderClsid

```
bool GetEncoderClsid(const WCHAR* format, CLSID* pClsid)
{
	UINT  num = 0;
	UINT  size = 0;
	ImageCodecInfo* pImageCodecInfo = NULL;
	GetImageEncodersSize(&num, &size);
	if(size == 0)  return false;

	pImageCodecInfo = (ImageCodecInfo*)(malloc(size));
	if(pImageCodecInfo == NULL)  return false;

	GetImageEncoders(num, size, pImageCodecInfo);
	for(UINT index = 0; index < num; ++index)
	{
		if( wcscmp(pImageCodecInfo[index].MimeType, format) == 0 )
		{
			*pClsid = pImageCodecInfo[index].Clsid;
			free(pImageCodecInfo);
			return true;
		}
	}
	free(pImageCodecInfo);
	return false;
}
```

用法：

```
CLSID encoderClsid
GetEncoderClsid(L"image/jpeg", &encoderClsid);
GetEncoderClsid(L"image/png", &encoderClsid);
GetEncoderClsid(L"image/bmp", &encoderClsid);
GetEncoderClsid(L"image/gif", &encoderClsid);
GetEncoderClsid(L"image/tiff", &encoderClsid);
```

3.6.2.2. 绘制图像

```
Image image(L"d:/1.jpg");
Graphics graphics(hdc);
```

graphics.DrawImage(&image,0,0,100,100);

3.6.2.3. 获取尺寸

int width=image.GetWidth();
int height=image.GetHeight();

3.6.2.4. 格式转换

Image img(L"D:\\b.bmp");
CLSID jpgClsid;
GetEncoderClsid(L"image/jpeg", &jpgClsid);
if(Ok != img.Save(L"D:\\xxx.jpg",&jpgClsid)){..}

3.6.2.5. 获取缩略图

Image *pImg = img.GetThumbnailImage(50,50);

delete pImg;

3.6.3. 颜色

第一个参数数透明度，0 代表全透明，255 代表不透明
Color drawClr(AlphaVal,GetRValue(clr),GetGValue(clr),GetBValue(clr));

3.6.4. 画笔

Pen(const Color& color, REAL width);
Pen(const Brush* brush, REAL width);
pen.SetDashStyle(DashStyleDash);//虚线

3.6.5. 画刷

//实心画刷
SolidBrush brush(drawClr);

//阴影画刷　　bkColor 为透明时可以看到效果

```
HatchBrush brush(HatchStyleHorizontal, forColor, bkColor);

//渐变画刷
LinearGradientBrush brush(pointA, pointB, Color(AlphaVal,255,0,0),Color(AlphaVal,255,255,0));

//图像画刷
Image image(L"img.jpg");
TextureBrush tBrush1(&image);
```

3.6.6. 绘制直线

```
#define    WIDTH    2

Graphics graphics(hDC);
Pen newPen(drawClr,WIDTH);
graphics.DrawLine(&newPen,10,30,100,100);
```

3.6.7. 绘制矩形

```
    SolidBrush brush(Color::BurlyWood);
    GetClientRect(rect);
        graphics.FillRectangle(&brush,rect.left,rect.top,rect.right,rect.bottom);
//FillRectangle 绘制实心矩形     DrawRectangle 绘制矩形边框
```

3.6.8. 绘制文字

```
SolidBrush brush(drawClr);
PointF pf(pt.x,pt.y);
Gdiplus::Font font(_T("宋?体?"), 10);
Graphics graphics(MemDC.m_hDC);
graphics.DrawString(name.GetBuffer(),name.GetLength(), &font,pf,&brush);
```

3.6.9. bmp 转成 jpg 流

```
    CLSID encoderClsid;
    GetEncoderClsid(L"image/jpeg", &encoderClsid);

    Bitmap Bmp(hDib, NULL);//dib 绑定到 bmp 后需要调用 DeleteObject(hDib);

    IStream *pStream = NULL;
    CreateStreamOnHGlobal(NULL, TRUE, &pStream);//创建一个 IStream
```

```
    Bmp.Save(pStream, &encoderClsid);//将 bmp 写入 IStream

    HGLOBAL hGlobal = NULL;
    GetHGlobalFromStream(pStream, &hGlobal);//获取 IStream 的内存句柄

    LPBYTE pBits = (LPBYTE)GlobalLock(hGlobal);//pBits 就是存储 jpg 内容的内存指针
    size_t size = GlobalSize(pBits);

    ////////////
      DeleteObject(hDib);
    pStream->Release();
      //GlobalFree(hGlobal);//与 pStream 指向同一个内存，同时释放会访问冲突
```

==========对 jpg 流进行使用

```
    //显示
    IStream *pImgStream = NULL;
    CreateStreamOnHGlobal(NULL, TRUE, &pImgStream);
    ULONG Written = 0;
    pImgStream->Write(pBits, size, &Written);
    Image img(pImgStream);
    CClientDC dc(this);
    Graphics graphics(dc.m_hDC);
    graphics.DrawImage(&img, 0, 0, 100, 100);
    pImgStream->Release();
```

========== hDib

```
    char           chBmpBuf[2048] = { 0 };
    BITMAPINFO     *pBmpInfo = (BITMAPINFO *)chBmpBuf;
    pBmpInfo->bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
    pBmpInfo->bmiHeader.biWidth = Width;//位图的宽度像素为单位
    pBmpInfo->bmiHeader.biHeight = Height;//位图的高度
    pBmpInfo->bmiHeader.biPlanes = 1;
    pBmpInfo->bmiHeader.biBitCount = biBitCount;//每个像素的位数(必须是8的倍数)
    pBmpInfo->bmiHeader.biCompression = BI_RGB;//压缩方式，一般为0或者BI_RGB（未压缩）
    for (int i = 0;i<256;i++)
    {
        pBmpInfo->bmiColors[i].rgbBlue = i;
        pBmpInfo->bmiColors[i].rgbGreen = i;
        pBmpInfo->bmiColors[i].rgbRed = i;
```

```
        pBmpInfo->bmiColors[i].rgbReserved = i;
    }

    BYTE* PDIBBuffer = NULL;
    HBITMAP hDib = CreateDIBSection(NULL, pBmpInfo, DIB_RGB_COLORS, (void**)&PDIBBuffer,
NULL, 0);//HDC仅在 usage参数为DIB_PAL_COLORS才使用
    memcpy_s(PDIBBuffer, bufSize, SrcBuffer, bufSize);
```

3.7. Dll

3.7.1. 动态库信息查看

Vs 命令提示符查看 dll 依赖项

dumpbin -dependents d:**.dll

-headers 查看 dll 信息

-exports 查看导出函数

3.7.2. 导出声明

3.7.2.1. 导出函数

```
extern "C" void _declspec(dllexport) fun(){}
```

3.7.2.2. 导出类

```
class _declspec(dllexport) CtestBase{};
```

3.7.2.3. stdcall

```
void __stdcall Run()
```

def 文件：

```
EXPORTS
```

Run

工程属性:
连接器->输入->模块定义文件-> xxx.def

使用：
int typedef (__stdcall *Fun)();

3.7.3. 共享数据段

不同进程加载同一 dll 时候，可以共享相同的数据。

#pragma data_seg ("shareData")//shareData 为自定义的段名

HWND g_hWnd=NULL;//共享数据必须初始化，否则会变成 bss 段的

#pragma data_seg()

#pragma comment(linker,"/section:shareData,rws")

3.7.4. 动态库调用

3.7.4.1. 隐式调用

需要 lib 文件，dll 文件

声明函数

void show();

库文件使用声明

#pragma comment(lib,"kk.lib")

直接调用

show()

3.7.4.2. 显示调用

只需要 dll 文件

```
typedef void (*pFun)();

HINSTANCE hInstanceLibrary=::LoadLibrary("kk.dll");

pFun fun=(pFun)GetProcAddress(hInstanceLibrary,"show");

fun();

::FreeLibrary(hInstanceLibrary);
```

3.7.5. 调用类型

3.7.5.1. _stdcall

Pascal 程序的缺省调用方式，参数采用从右到左的压栈方式，被调函数自身在返回前清空堆栈。

WIN32 Api 都采用_stdcall 调用方式

3.7.5.2. _cdecl

C/C++的缺省调用方式，参数采用从右到左的压栈方式，传送参数的内存栈由调用者维护。_cedcl 约定的函数只能被 C/C++调用，每一个调用它的函数都包含清空堆栈的代码，所以产生的可执行文件大小会

比调用_stdcall 函数的大。

由于_cdecl 调用方式的参数内存栈由调用者维护，所以变长参数的函数能使用这种调用约定

3.7.5.3. _fastcall

调用较快，它通过 CPU 内部寄存器传递参数。

按 C 编译方式，_fastcall 调用约定在输出函数名前面加“@”符号，后面加“@”符号和参数的字节数，形如@functionname@number。

3.7.6. 错误原因

头文件的作用是声明函数或者变量，如果缺少头文件，会出现找不到标识符的错误。

Lib 文件的作用是指明头文件中声明的函数存在，lib 文件包含实际代码的进入信息。如果没有引用 lib，会出现 unresolved external symbol（无法解析的外部符号）的错误。如果引用了 lib，但是 lib 不在系统目录或者当前目录，会出现缺少 lib 的错误。(lib 文件可以在连接器的附加依赖项中设置，也可以在代码中显式声明)

Dll 文件存储着函数的实现代码，如果缺少 dll，程序运行时会出错。

stl 类型的参数无法作为引用或者指针进行传递。

3.7.7. 调试

3.7.7.1. Dll 工程调试

配置属性-》调试-》命令　　输入调用 exe 所在位置

3.7.7.2. EXE 调试

dll 和 pdb 文件需要在一起

将 dll 的 cpp 文件直接在 exe 中打开即可

3.7.8. 窗口交互

3.7.8.1. 创建模式对话框

函数前添加宏

```
AFX_MANAGE_STATE(AfxGetStaticModuleState());
```

3.7.8.2. 获取外部窗口

在库文件中需要调用外部窗口对象时，可以使用

HWND hWnd=GetForegroundWindow();

3.8. SHELL

一般是指由操作系统提供的用于计算机用户向操作系统输入相关指令并得到结果的程序，shell 可以字符形式的，也可以是 GUI。Windows shell 最重要的组成部件是 explorer.exe，提供了开始菜单、任务栏、资源管理器等。

Shell API 都是在 shlobj.h 文件中声明，由 Shell32.dll 导出，链接时需要 Shell32.lib.

Shell 扩展程序是 COM(Component Object Model,组件对象模型)程序。COM 程序一般是 DLL 文件（ActiveX 程序是.ocx 扩展名的文件）

3.8.1. 遍历回收站

```
TCHAR pszPath[MAX_PATH];
IShellFolder *pisf=NULL;
IShellFolder *pisfRecBin=NULL;
SHGetDesktopFolder(&pisfRecBin);
IEnumIDList *peidl=NULL;
LPITEMIDLIST pidlBin=NULL;
LPITEMIDLIST idlCurrent=NULL;

SHGetSpecialFolderLocation(NULL,CSIDL_BITBUCKET,&pidlBin);
pisfRecBin->BindToObject(pidlBin,NULL,IID_IShellFolder,(void**)&pisf);

pisf->EnumObjects(NULL,SHCONTF_FOLDERS|SHCONTF_NONFOLDERS|SHCONTF_IN
CLUDEHIDDEN,&peidl);
STRRET strret;
ULONG uFetched;
while(1)
{
    if(peidl->Next(1,&idlCurrent,&uFetched)==S_FALSE)
        break;
    SHGetPathFromIDList(idlCurrent,pszPath);
    pisf->GetDisplayNameOf(idlCurrent,SHGDN_NORMAL,&strret);
    CStringW str;
    str.Format(L"%s",strret.pOleStr);
    ::MessageBoxW(m_hWnd,str,L"xx",MB_OK);
}
```

3.8.2. CLSID

GUID(Global unique identier),全局唯一标识符，用于在系统中唯一标识一个对象。CLSID 是 GUID 在注册表中的表示。

3.9. function

3.9.1. CString

#include <atlstr.h>

CString 以”\0”结尾，如果字符串中在结尾前包含”\0”,那么 CString 以

第一次出现”\0”为准。

LPCTSTR 指的就是 CString，LPSTR 指 char *

可用 CString str(_T("xxx"));构造

可用 str.LoadString(ID)加载字符串资源中的字符串

str.Format(_T("define %d %d"),wParam,lParam);

str.Empty(); 清空字符串

str+_T(’c’); 为字符串增加一个字符

========== 追加

str.Append(_T("x"));

str.AppendFormat(_T("%d"),10);

==========查找

if(-1!=str.Find(_T("clear")))//表示 str 中有 clear 字样,如果 find 返回-1 代表没有}

==========分割

CString str = _T("ab,cde");

int pos = str.Find(_T(","));

str = str.Left(pos);//得到 ab

str = str.Mid(pos+1);//得到 cde,第二个参数如果不设置，代表从 pos 位置截取到末尾（加 1 因为查找的字符串为个长度）

或者： str = str.Right(str.GetLength()-pos-1);

==========修剪

CString str=_T(" ab");

str.TrimLeft(); //可以将字符串左边多余的空格去掉

==========按条件截取

CString str=_T("123abcd456");

CString s1=str.SpanIncluding(_T("0123456789"));//s1 为 123

CString s2=str.SpanIncluding("a");//s2 为空

CString s3=str.SpanExcluding(_T("d"));//s3 为 123abc

==========比较

str1.Compare(str2); //如果str1与str2一样，返回0，否则返回1

str1.CompareNoCase(str2); //比较时忽略大小写

==========转换

char *p=strA.GetBuffer(0);//CStringA 中的 GetBuffer 就是转成 char *;GetBuffer 后必须要调用 ReleaseBuffer，否则 CStringA 的其他函数失效。

int i=_ttoi(p); //字符串变成 int

转成 wchar_t

```
int i=::MultiByteToWideChar(CP_ACP,0,(LPCTSTR)file,-1,NULL,0);
wchar_t *pPath=new wchar_t[i];
::MultiByteToWideChar(CP_ACP,0,(LPCTSTR)file,-1,pPath,i);
delete[] pPath;
```

==========使用 CString 直接进行转换

```
CStringW strw = _T("abc");
CStringA str(strw.GetBuffer());
string    s(str.GetBuffer());
```

3.9.2. 时间

```
SYSTEMTIME st;
GetLocalTime(&st);  得到系统时间，并放入到结构体中
CString str;
str.Format("%d:%d:%d",st.wHour,st.wMinute,st.wSecond);
```

计时：

```
LONGLONG    t1 = GetTickCount64();
Sleep(60);
LONGLONG val = GetTickCount64() - t1;
```

3.9.3. INI 文件

通过设置段名和键名可以确定一个键值。可以进行写入和读取。

Ini 文件写入情况如下（也可以用 section 的函数写没有键名的）

不能新建 txt 然后改成 ini，第一次文件需要用函数写。文件名必须是全路径，不能是相对路径。

[Section1]

1=x

2=y

[Section2]

abc

::WriteProfileString 等 SDK 函数可以在 C:\WINDOWS\win.ini 读写数据。

如果是 WriteProfileString 等 CWinApp 的成员函数，则会写入注册表中。

3.9.3.1. 写入

::WritePrivateProfileString(strSectionName, strKeyName, strKeyValue, strFileName);

如果 strKeyValue 是 NULL，则表示删除那一行

没有 strKeyName 时(debug 下会有冗余数据，只有 release 可以)：

::WritePrivateProfileSection(strSectionName,strKeyValue,strFileName);

3.9.3.2. 读取

CString strKeyValue = _T("");

::GetPrivateProfileString(strSectionName, strKeyName, _T(""),

strKeyValue.GetBuffer(1024), 1024,strFileName);

strKeyValue.ReleaseBuffer();

没有 strKeyName 时：

::GetPrivateProfileSection(strSectionName,strKeyValue.GetBuffer(1024),

1024,strFileName);

读取数字：（只用于带 key 的）

int x=GetPrivateProfileInt(strSectionName,strKeyValue,NULL,strFileName);

3.9.3.3. 遍历

```
void GetUnitINI(map<CString,CString> &Key_Value,CString strSection,CString iniFileName)
{
    while(true)
    {
        CString strKey;
        DWORD                                                nChar                          =
GetPrivateProfileString(strSection,NULL,_T(""),strKey.GetBuffer(1024),1024,iniFileName)
;
        if (0 == nChar) break;


        CString strValue;

    GetPrivateProfileString(strSection,strKey,_T(""),strValue.GetBuffer(1024),1024,iniF
```

```
ileName);
        strValue.ReleaseBuffer();

        Key_Value[strKey] = strValue;

        //删除
        WritePrivateProfileString(strSection,strKey,NULL,iniFileName);
    }

    //重新写入
    for (map<CString,CString>::const_iterator iter = Key_Value.begin(); iter != Key_Value.end(); ++iter)
    {
        WritePrivateProfileString(strSection,iter->first,iter->second,iniFileName);
    }
}
```

3.9.4. CImage

3.9.4.1. 绘图及图像转换

```
#include <atlimage.h>

CImage Img;
if (S_OK != Img.Load(L"D:\\1.jpg")) return;
HDC hdc = GetDC()->m_hDC;
::SetStretchBltMode(hdc, COLORONCOLOR);//缩放绘制时必须设置，否则出现失真
Img.Draw(hdc, 0, 0, 100, 100);

int nchannels = img.GetBPP()/8;

Img.Save(L".\\abc.jpg");
```

3.9.4.2. 获取 DIB

```
    CImage Img;
    Img.Attach(hDib);
    Img.Draw(GetDC()->m_hDC,0,0,Img.GetWidth(),Img.GetWidth());
```

可以代理hdib的销毁操作

3.9.5. 调试

TRACE("xx");//可以将 CString 输出到调试窗口，debug 有效

TRACE(_T("\r\n"));

3.9.6. 消息盒子

if(IDCANCEL== MessageBox(m_hWnd,_T("xx"),"标题",MB_OK)) return;

3.9.7. CMD

#include <io.h>

AllocConsole(); //FreeConsole();

HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);

CString str = _T("abc");

WriteConsole(handle, str.GetBuffer(), str.GetLength(), NULL, NULL);

========字体颜色

白底红字

system("color fc");

FOREGROUND_RED

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),0xfc);

//0xfa 绿色字白色底

3.9.8. 获取命令行参数

int num = __argc;

TCHAR **args = __targv;

3.9.9. 特殊目录获取

```
#include <ShlObj.h>

CString GetSpecialFolderPath(UINT CSIDL)
{
    TCHAR szMyDocument[MAX_PATH] = {0};
    SHGetSpecialFolderPath(NULL,szMyDocument,CSIDL,FALSE);
    return szMyDocument;
}
```

```
CSIDL_SYSTEM        系统dll放置位置    GetSystemDirectory
CSIDL_SYSTEMX86     32位dll放置位置
CSIDL_DESKTOP       桌面
CSIDL_STARTUP       启动目录
CSIDL_COMMON_STARTMENU      开始菜单（可以创建文件夹）
CSIDL_FONTS                 字体目录
```

3.9.10. 创建快捷方式

```
#include <ShlObj.h>


void  CreateShortCut(CString  filePathName,CString  destPath,CString
destName,CString description = _T(""))
{//destPath后面不带结束符
    if (!PathFileExists(filePathName)) throw filePathName + _T("  不存
在");
```

```
    CString  destPathName = destPath + _T("\\") + destName + _T(".lnk");

    IShellLink *plink = NULL;
    HRESULT                                hr                                =
CoCreateInstance(CLSID_ShellLink,NULL,CLSCTX_INPROC_SERVER,IID_IShell
Link,(void**)&plink);
    if(FAILED(hr))  throw CString(_T("创建快捷方式实例错误"));

    IPersistFile *ppf = NULL;
    hr  = plink->QueryInterface(IID_IPersistFile,(void**)&ppf);

    if (FAILED(hr))
    {
        plink->Release();
        throw CString(_T("创建快捷方式QueryInterface错误"));
    }

    plink->SetPath(filePathName);

    plink->SetDescription(description);

    plink->SetShowCmd(SW_NORMAL);

    hr = ppf->Save(destPathName,TRUE);

    ppf->Release();
    plink->Release();

    if(!SUCCEEDED(hr)) CString(_T("创建快捷方式失败"));
}
```

3.9.11. 获取窗口位置

```
RECT rect;
```

GetClientRect(hWnd,&rect);得到客户区坐标（大小），相对于父窗口的坐标

GetWindowRect(hWnd,&rect);得到当前窗口的坐标（大小），相对于屏幕坐标系的坐标

3.9.12. 窗口大小和位置

```
int cx=::GetSystemMetrics(SM_CXSCREEN);
int cy=::GetSystemMetrics(SM_CYSCREEN);
```

如果是窗口，以屏幕坐标系为基准（窗口属于屏幕）。如果是窗口内部控件，就以窗口为基准（控件属于窗口）。

SetWindowPos(hWnd,NULL, x, y, cx, cy,SWP_SHOWWINDOW);

第一个参数指 z 序对应的父窗口

如果最后一个参数是SWP_NOZORDER时，x，y同时为0，则该函数只能设置大小，位置会自动调节到屏幕中间。SWP_NOMOVE可以使窗口无法移动。

```
MoveWindow(hWnd, x, y, cx, cy,TRUE);
```

主窗口使用 MoveWindow 时如果 x，y 同时为 0，则会自动移到中间。

3.9.13. 顶层窗口

```
SetWindowPos(hWnd, HWND_TOPMOST, x, y, cx, cy, SWP_SHOWWINDOW);
```

3.9.14. 鼠标

if(GetAsyncKeyState(VK_LBUTTON)==0) //判断全局的鼠标左键是否弹起

ShowCursor(FALSE); //显示或隐藏鼠标

POINT pt;

GetCursorPos(&pt);//得到当前鼠标位置，相对于屏幕坐标系。

或者：

```
DWORD dwPos = GetMessagePos();
POINT pt = { LOWORD(dwPos), HIWORD(dwPos) };
```

ScreenToClient(hWnd,&pt);//转换为客户区坐标系

//判断鼠标是否在某个区域

PtInRect(rect,pt)

3.9.15. 获取鼠标滚动量

```
case  WM_MOUSEWHEEL:
{
    POINT pt;
    GetCursorPos(&pt);
    ScreenToClient(hWnd,&pt);//获取当前鼠标在对话框中的位置
    int delta = GET_WHEEL_DELTA_WPARAM(wParam);//获取滚动量
}
```

SetFocus(hWnd);//按钮点击时会改变鼠标焦点，此时需要将焦点返回到窗口，否则滚轮消息收不到

3.9.16. 模拟鼠标消息

SetCursorPos(x,y);

mouse_event(MOUSEEVENTF_LEFTDOWN,0,0,0,0);

mouse_event(MOUSEEVENTF_LEFTUP,0,0,0,0);

3.9.17. 播放声音

WAV 格式

```
#include <Mmsystem.h>
#pragma comment(lib, "winmm.lib")

==============播放文件
sndPlaySound(_T("D:\\2.wav"),SND_ASYNC);


==============播放资源文件
PlaySound(MAKEINTRESOURCE(IDR_WAVE1),AfxGetResourceHandle(),SND_ASYNC|SND_RESOURCE|SND_
NODEFAULT|SND_LOOP);


==============加载资源文件播放
HMODULE hmod=AfxGetResourceHandle();
HRSRC hSndResource=FindResource(hmod,MAKEINTRESOURCE(IDR_WAVE1),_T("WAVE"));
HGLOBAL m_hGlobalMem=LoadResource(hmod,hSndResource);
LPCTSTR m_lpMemSound=(LPCSTR)LockResource(m_hGlobalMem);

sndPlaySound(m_lpMemSound,SND_MEMORY);

//FreeResource(m_hGlobalMem);
```

3.9.18. 后台运行

```
WINDOWPLACEMENT wpOrg;
GetWindowPlacement(hWnd,&wpOrg);//恢复时用


WINDOWPLACEMENT wp;
wp.length = sizeof(WINDOWPLACEMENT);
wp.flags = WPF_RESTORETOMAXIMIZED;
wp.showCmd = SW_HIDE;
SetWindowPlacement(hWnd,&wp);
```

3.9.19. 只运行一个程序实例

```
const CString g_szPropName =_T("Your Prop Name");
const HANDLE  g_hValue = (HANDLE)1;
```

```
BOOL CALLBACK EnumWndProc(HWND hwnd,LPARAM lParam)
{
    HANDLE h = GetProp(hwnd,g_szPropName);
    if( h == g_hValue)
    {
        *(HWND*)lParam = hwnd;
        return false;
    }

    return true;
}
```

======程序开始运行时判断：

```
HWND oldHWnd = NULL;
EnumWindows(EnumWndProc,(LPARAM)&oldHWnd); if(oldHWnd != NULL)
{   ::ShowWindow(oldHWnd,SW_SHOW);
    ::SetForegroundWindow(oldHWnd);
      return false;
}
```

======程序启动后，设置程序标志：

```
BOOL    addNewHandle=SetProp(m_hWnd,g_szPropName,g_hValue);
//设置当前程序的标识，使下次运行时可以作为判断依据
```

======释放标记

```
RemoveProp(m_hWnd,g_szPropName);
```

3.9.20. 内存监控

```
MEMORYSTATUSEX mems;
mems.dwLength = sizeof(mems);
GlobalMemoryStatusEx(&mems);

DWORDLONG ullTotalPhys    = mems.ullTotalPhys/1024/1024;//计算机内存
DWORDLONG ullAvailPhys = mems.ullAvailPhys/1024/1024;//计算机可用内存    8g 内存 32 位系统可以用 4g

DWORDLONG ullTotalVirtual = mems.ullTotalVirtual/1024/1024;//m
```

DWORDLONG ullAvailVirtual = mems.ullAvailVirtual/1024/1024;//m

DWORDLONG usedMem = ullTotalVirtual - ullAvailVirtual; //当前已经使用的内存

3.9.21. 窗口去色

```
void SetAlpha(HWND wnd, COLORREF alphaClr = RGB(0, 0, 0))
{
#define LWA_COLORKEY    0x00000001
#define LWA_ALPHA       0x00000002
#define WS_EX_LAYERED   0x00080000

    typedef BOOL(WINAPI *LPFNSETLAYEREDWINDOWATTRIBUTES)(HWND hWnd, COLORREF crKey, BYTE bAlpha, DWORD dwFlags);
    LPFNSETLAYEREDWINDOWATTRIBUTES SetLayeredWindowAttributes;
    HMODULE hUser32 = GetModuleHandle(_T("user32.dll"));
    SetLayeredWindowAttributes = (LPFNSETLAYEREDWINDOWATTRIBUTES)GetProcAddress(hUser32, "SetLayeredWindowAttributes");
    SetWindowLong(wnd, GWL_EXSTYLE, GetWindowLong(wnd, GWL_EXSTYLE) | WS_EX_LAYERED);
    SetLayeredWindowAttributes(wnd, alphaClr, 0, LWA_COLORKEY);
}
```

3.9.22. 定时器

SetTimer(hWnd, 0, 1000, NULL);

第二个参数为定时器 id，可任意，第三个 1000 代表 1 秒，第四个代表使用 hWnd 中的处理函数,如果为 NULL，则使用 WM_TIMER（*wParam* 代表 id）

3.9.23. 显示或隐藏任务栏

```
//隐藏任务栏
HWND wnd = FindWindow(_T("Shell_TrayWnd"), NULL);
if (IsWindowVisible(wnd)) ShowWindow(wnd,SW_HIDE);


//显示任务栏
HWND wnd = FindWindow(_T("Shell_TrayWnd"), NULL);
if (!IsWindowVisible(wnd)) ShowWindow(wnd, SW_SHOW);
```

3.9.24. 注销、关闭、重启系统

3.9.24.1. 使用前相关操作

```
if (IDNO == MessageBox(_T("是否进行**操作？"), _T("对话框标题"), MB_YESNO)){
        return;
    }
    //打开进程令牌
    HANDLE hToken;
    if (!OpenProcessToken(GetCurrentProcess(),
            TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken))
    {
        AfxMessageBox(_T("OpenProcessToken Error."));
    }

    //获得 LUID
    TOKEN_PRIVILEGES tkp;
    LookupPrivilegeValue(NULL, SE_SHUTDOWN_NAME, &tkp.Privileges[0].Luid);
    tkp.PrivilegeCount = 1;
    tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

    //调整令牌权限
    AdjustTokenPrivileges(hToken, FALSE, &tkp, 0, (PTOKEN_PRIVILEGES)NULL, 0);
     if (GetLastError() != ERROR_SUCCESS) return;
```

3.9.24.2. 注销

```
if (!ExitWindowsEx(EWX_LOGOFF, 0))    return;
```

3.9.24.3. 关闭

```
if (!ExitWindowsEx(EWX_SHUTDOWN, 0)) return;
```

3.9.24.4. 重启

```
if (!ExitWindowsEx(EWX_REBOOT, 0)) return;
```

3.9.25. 窗口标题操作

```
SetWindowText(hWnd,_T("窗口标题"));
CString rString;
GetWindowText(hWnd, rString.GetBuffer(1024),1024);
```

3.9.26. 打开文件夹对话框

```
inline bool GetDirPath(CString &strFolderPath,HWND wndParent=NULL,CString Title = _T("请选择一个文件夹"))
{
	TCHAR 			szFolderPath[MAX_PATH] = {0};

	BROWSEINFO 		sInfo;
	::ZeroMemory(&sInfo, sizeof(BROWSEINFO));
	sInfo.pidlRoot 	= 0;
	sInfo.lpszTitle 	= 	Title;
	sInfo.ulFlags 		= 	BIF_DONTGOBELOWDOMAIN 	| 	BIF_RETURNONLYFSDIRS 	| BIF_NEWDIALOGSTYLE | BIF_EDITBOX;
	sInfo.lpfn 		= NULL;
	sInfo.hwndOwner = wndParent;

	// 显示文件夹选择对话框
	LPITEMIDLIST lpidlBrowse = ::SHBrowseForFolder(&sInfo);
	if (lpidlBrowse != NULL)
	{
		// 取得文件夹名
		if (::SHGetPathFromIDList(lpidlBrowse,szFolderPath))
		{
			strFolderPath = szFolderPath;
		}
		::CoTaskMemFree(lpidlBrowse);
	}

	if(lpidlBrowse == NULL) 	return false;
	return true;
}
```

3.9.27. 打开文件对话框

```
#include "commdlg.h"
#pragma comment(lib,"Comdlg32.lib")
```

```
bool ChoiceFile(CString &filePathName,const TCHAR* filter)
{
    //_T("xx(*.jpg)\0*.jpg\0所有文件（*.*）\0*.*\0");
    TCHAR szFileName[MAX_PATH] = {};
    OPENFILENAMEW openFileName = {};
    openFileName.lStructSize = sizeof(OPENFILENAMEW);
    openFileName.nMaxFile = MAX_PATH;//这个必须设置，不设置的话不会出现打开文件对话框
    openFileName.lpstrFilter = filter;
    openFileName.lpstrFile = szFileName;
    openFileName.nFilterIndex = 1;
    openFileName.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;
    // 如果保存则用GetSaveFileName()
    if (!GetOpenFileName(&openFileName)) return false;

    filePathName = openFileName.lpstrFile;
    return true;
}
```

3.9.28. 读写注册表

```
//打开注册表  用户环境变量
HKEY hKey;
if(RegOpenKey(HKEY_CURRENT_USER, _T("Environment"), &hKey) != ERROR_SUCCESS) return;

//读取 path 环境变量
LPBYTE  data[2048] = {0};
DWORD lValueSize;
if (RegQueryValueExA(hKey,"path",NULL,NULL,(LPBYTE)data,&lValueSize) == ERROR_SUCCESS)
{
    CStringA re;
    re.Format("%s",data);
}

//设置环境变量
if(RegSetValueExA(hKey,strName.c_str(),NULL,regType,(BYTE*)strValue.c_str(),strValue.le
ngth()) != ERROR_SUCCESS) return;
```

3.9.29. 开机启动

```
bool SetStartInfo(TCHAR *KeyName,bool remove = false)
```

```
{
    HKEY hKey;
    TCHAR* lpRun = _T("Software\\Microsoft\\Windows\\CurrentVersion\\Run");
    long lRet = RegOpenKeyEx(HKEY_CURRENT_USER, lpRun, 0, KEY_WRITE,
&hKey);

    if(ERROR_SUCCESS != lRet) return false;


    TCHAR pFileName[MAX_PATH] = {0};
    DWORD dwRet = GetModuleFileName(NULL, pFileName, MAX_PATH);
     DWORD strSize = dwRet*sizeof(TCHAR);

    if (remove)
    {//删除
        lRet = RegDeleteValue(hKey,KeyName);
    }
    else
    {//添加
        lRet = RegSetValueEx(hKey, KeyName, 0, REG_SZ, (BYTE *)pFileName,
strSize);
    }

    RegCloseKey(hKey);
    if(lRet != ERROR_SUCCESS)      return false;
    return true;
}
```

3.9.30. 注册 ActiveX 控件

```
BOOL DllRegister(LPCTSTR lpszDllName)
{
    ASSERT(lpszDllName != NULL);
    ASSERT(AfxIsValidString(lpszDllName));

    //加载 ActiveX 控件
    HINSTANCE hLib = LoadLibrary(lpszDllName);
    if (hLib == NULL)
    {
        TRACE(_T("%s 加载失败\n"), lpszDllName);
        return FALSE;
    }
```

```
    //获得注册函数 DllRegisterServer 地址
    FARPROC lpDllEntryPoint;
    lpDllEntryPoint = GetProcAddress(hLib, _T("DllRegisterServer"));

    //调用注册函数 DllRegisterServer
    if (lpDllEntryPoint != NULL)
    {
        if (FAILED((*lpDllEntryPoint)()))
        {
            TRACE(_T("调用 DllRegisterServer 失败\n"));
              FreeLibrary(hLib);
              return FALSE;
        }
        else
        {
            FreeLibrary(hLib);
            return TRUE;
        }
    }
    else
    {
        TRACE(_T("调用 DllRegisterServer 失败\n"));
        FreeLibrary(hLib);
        return FALSE;
    }
}
```

3.9.31. 模拟键盘按键

3.9.31.1. 模拟 NUM LOCK 按键

```
keybd_event(VK_NUMLOCK, 0, KEYEVENTF_EXTENDEDKEY | 0, 0);
keybd_event(VK_NUMLOCK, 0, KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP, 0);
```

3.9.31.2. 模拟 CAPS LOCK 按键

```
keybd_event(VK_CAPITAL, 0, KEYEVENTF_EXTENDEDKEY | 0, 0);
keybd_event(VK_CAPITAL, 0, KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP, 0);
```

3.9.31.3. 模拟 SCROLL LOCK 按键

```
keybd_event(VK_SCROLL, 0, KEYEVENTF_EXTENDEDKEY | 0, 0);
keybd_event(VK_SCROLL, 0, KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP, 0);
```

3.10. file

3.10.1. 基本概念

3.10.1.1. 磁盘分区

一个物理硬盘上划分的可以独立工作的一些逻辑磁盘。

3.10.1.2. 卷

也称为逻辑驱动器，是 ntfs 等文件系统的最高层。是存储设备（硬盘）上文件系统管理的一块区域，在逻辑上互相隔离的存储单元。一个磁盘分区至少含有一个卷。仅存在于一个分区上的卷称为“简单卷”，仅存在与多个分区上的卷称为“多分区卷”或“跨区卷”。一般情况下，一个分区只包含一个卷，一个卷也只存在于一个分区上。卷存在卷标，程序可以通过卷标访问卷。

3.10.2. 获取硬盘信息

```
DWORD VolumeSerialNumber;
GetVolumeInformation("C:\\",NULL,12,&VolumeSerialNumber,NULL,NULL,NULL,10);
//CString ss;
//ss.Format("%x",VolumeSerialNumber);
```

可以获取 c 盘的序列号，如果第一个参数不指定位置，则获取的是当前位置的号码

3.10.2.1. 遍历卷

```
DWORD len=1024;
CHAR ch[1024];
ZeroMemory(ch,1024);
GetLogicalDriveStrings(len,ch);

CHAR* p=ch;
while(*p!='\0')
{
```

```
        CString str;
        str.Format("%s",p);
        AfxMessageBox(str);//得到所有的卷标

        p+=lstrlen(p)+1;
    }
```

3.10.2.2. 读文件

```
    HANDLE hFile;
    hFile=CreateFile("d:\\a.txt",GENERIC_READ,0,NULL,OPEN_EXISTING,FILE_ATTRIB
UTE_NORMAL,NULL);
    char ch[100];
     DWORD dwReads;
    ReadFile(hFile,ch,100,&dwReads,NULL); //读取文件放到 ch 中
    ch[dwReads]=0;
    CloseHandle(hFile);
```

3.10.2.3. 写文件

```
    HANDLE hFile;
    hFile=CreateFile("d:\\a.txt",GENERIC_WRITE,0,NULL,CREATE_NEW,FILE_ATTRIBUTE_NORMA
L,NULL);
    DWORD dwWrites;
    WriteFile(hFile,"abcde",strlen("abcde"),&dwWrites,NULL);
    CloseHandle(hFile);
```

3.10.3. 目录操作

```
#include <shlwapi.h>
```

3.10.3.1. 获取当前目录

VS 中，当前目录指工程所在目录

一般直接写".\\"

有时不能使用相对路径，只能用绝对路径

GetCurrentDirectory(1024,str.GetBuffer(1024));

//获取exe所在目录

```
CString GetCurPath()
{
    CString path;
    GetModuleFileName(NULL, path.GetBuffer(MAX_PATH), MAX_PATH);
    path.ReleaseBuffer();

    path = path.Left(path.ReverseFind(_T('\\')));
    return path;
}
```

3.10.3.2. 获取模块名称

第一个参数为 null 时，获取当前模块（exe）的全路径（包括文件名）

GetModuleFileName(NULL,str.GetBuffer(1024),1024);

3.10.3.3. 设置进程当前目录

SetCurrentDirectory("d:\\");

设置后，程序中使用.\\即为 d:\\

3.10.3.4. 判断是否存在

也可以用于判断网络目录是否联通

CString strPath=_T("d:\\test");

if(PathIsDirectory(strPath)) {　//存在}

else{ //不存在}

3.10.3.5. 创建目录

#include <shlobj.h>

(void)SHCreateDirectoryEx(NULL,"D:\\abc\\efg",NULL);

3.10.4. 文件查找

```
void DirList(CString Path,vector<CString> &fileVec)
{
    WIN32_FIND_DATA FindData;
    HANDLE hError = FindFirstFile(Path + _T("\\*.*"), &FindData);
    if (hError == INVALID_HANDLE_VALUE) return;
    CString DotDir = _T(".");
    CString TowDotDir = _T("..");
    while (::FindNextFile(hError, &FindData))
    {
        if (DotDir == FindData.cFileName || TowDotDir == FindData.cFileName) continue;

        CString FullPathName = Path + _T("\\") + FindData.cFileName;
        if (FindData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
        {
            DirList(FullPathName, fileVec);
        }
        else
        {
            fileVec.push_back(FullPathName);
        }
    }
}

void GetFileList(CString Path, vector<CString> &fileVec,CString ext=_T("*.txt"))
{
    WIN32_FIND_DATA FindData;
    HANDLE hError = FindFirstFile(Path + _T("\\") + ext, &FindData);
    if (hError == INVALID_HANDLE_VALUE) return;

    CString FullPathName = Path + _T("\\") + FindData.cFileName;
    fileVec.push_back(FullPathName);
    while (::FindNextFile(hError, &FindData))
    {
        FullPathName = Path + _T("\\") + FindData.cFileName;
```

```
            fileVec.push_back(FullPathName);
        }
    }
```

3.10.5. 删除

3.10.5.1. 删除文件

DeleteFile("d:\\1.txt");

删除后回收站无信息

3.10.5.2. 删除目录

只能用于非空目录删除

RemoveDirectory(directory_path);

3.10.6. 文件操作

3.10.6.1. 文件是否存在

PathFileExists(strPath)；

3.10.6.2. 复制

CopyFile("d:\\1.jpg","d:\\11.jpg",TRUE);

将 1.jpg 复制为 11.jpg，如果已经存在，则覆盖（TRUE 指定）

3.10.6.3. 移动/重命名

可以移动文件或者目录

MoveFile("d:\\1.jpg","d:\\ATY\\1.jpg");

3.10.6.4. 文件另存为

CopyFile(_T(".\\a.txt"),_T("d:\\a.txt"),FALSE);//将当前目录下的 a.txt 保存到 d 盘

3.10.6.5. 内存方式读写文件

通过使用 Mapping File 对文件进行如同读写内存方式的操作

```
#define FILE_MAP_START 0x28904
#define    BUFFSIZE 1024

    SYSTEM_INFO SysInfo;
    ::GetSystemInfo(&SysInfo);
    DWORD dwSysGran=SysInfo.dwAllocationGranularity;//获取系统内存分配粒度
    DWORD dwMapViewSize=(FILE_MAP_START%dwSysGran)+BUFFSIZE;
    DWORD dwFileMapSize=FILE_MAP_START+BUFFSIZE;

    HANDLE
hFile=CreateFile("d:\\2.jpg",GENERIC_READ|GENERIC_WRITE,FILE_SHARE_READ|FIL
E_SHARE_WRITE,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
    HANDLE
hMapFile=CreateFileMapping(hFile,NULL,PAGE_READWRITE,0,dwFileMapSize,NULL);
    if(hMapFile==NULL)
    {
        return ;
    }
    LPVOID lpMapAddress=MapViewOfFile(hMapFile,FILE_MAP_ALL_ACCESS,0,
        0,dwMapViewSize);//代表从起始位置开始
    if(lpMapAddress==NULL)
    {
        return ;
    }

    char* pData=(char *)lpMapAddress;//char*指针（也可以是 int*等指针），相当
于直接指向文件，此时对通过该指针可以对文件进行类似指针的操作

    *pData=*pData^22; //可以对指定位置进行两次异或操作，即变化后再还原
```

FlushViewOfFile(lpMapAddress,dwMapViewSize);//操作文件视图后，写入文件

CloseHandle(hFile);

3.10.6.6. 中文支持

```
#include <locale>
setlocale(LC_CTYPE,"chs");
```

unicode 下无法输出中文，需要设置

3.11. IPC

3.11.1. 获取 PID

通过遍历操作系统进程获取进程名称和 PID

```
#include <tlhelp32.h>
bool getPid(DWORD &findPID, CString findExeName)
{//findExeName名称中包含.exe
    bool hasfind = false;

    PROCESSENTRY32 pe32;
    pe32.dwSize = sizeof(pe32);
    HANDLE hProcessSnap = ::CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (hProcessSnap == INVALID_HANDLE_VALUE)
    {
        MessageBox(NULL,TEXT("CreateToolhelp32Snapshot调用失败!"),TEXT(""),MB_OK);
        return hasfind;
    }

    BOOL bMore = ::Process32First(hProcessSnap, &pe32);
    while (bMore)
    {
        CString exeName = pe32.szExeFile;
        DWORD pid = pe32.th32ProcessID;
        if (exeName == findExeName)
```

```
        {
            findPID = pid;
            hasfind = true;
            break;
        }
        bMore = ::Process32Next(hProcessSnap, &pe32);
    }
    ::CloseHandle(hProcessSnap);

    return hasfind;
}
```

3.11.2. 获取 hProcess

```
HWND wnd = ::FindWindow(NULL,wndName);
if (NULL == wnd) return;
DWORD pid;
GetWindowThreadProcessId(wnd,&pid);

HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS,FALSE,pid);
```

3.11.3. DLL 注入

```
void InjectHookDLL(HANDLE hProcess,PCWSTR dllFilePathName)
{
    int cch = 1 + lstrlenW(dllFilePathName);
    int cb = cch * sizeof(wchar_t);
    PWSTR PszLibFileRemote = (PWSTR)VirtualAllocEx(hProcess, NULL, cb,
MEM_COMMIT, PAGE_READWRITE);
    WriteProcessMemory(hProcess, PszLibFileRemote,
(PVOID)dllFilePathName, cb, NULL);
    HANDLE hThread = CreateRemoteThread(hProcess, NULL, 0,
(PTHREAD_START_ROUTINE)LoadLibrary, PszLibFileRemote, 0, NULL);
```

```
    WaitForSingleObject(hThread, INFINITE);
}
```

3.11.4. 钩子

3.11.4.1. 进程内钩子

```
::SetWindowsHookEx(WH_MOUSE,MouseProc, NULL, GetCurrentThreadId());
```

3.11.4.2. 监听程序消息

动态库为所有被监听对象共享（每个被监听的程序触发监听事件后都会加载该动态库）
CreatorPID 为 0，则监听所有进程

3.11.4.2.1. 动态库

```
LRESULT CALLBACK KeyboardProc(int code, WPARAM wParam, LPARAM lParam)
{
        ********
    return 0;//返回 0，代表将消息传出去，[如果是 return 1，则拦截消息]
}
extern "C" _declspec(dllexport) void Run(DWORD CreatorPID,const char *dllName)
{
    SetWindowsHookEx(WH_KEYBOARD,
KeyboardProc, ::GetModuleHandle(CString(dllName).GetBuffer()), CreatorPID);
}
```

3.11.4.2.2. 使用程序激活动态库

```
HWND hWnd = ::FindWindow(0, _T("MFCApplication2"));
if (NULL == hWnd)return;
DWORD CreatorPID = GetWindowThreadProcessId(hWnd, 0);

HMODULE h = LoadLibrary(_T("*****\\Release\\hh.dll"));
typedef void  (*pfun)(DWORD CreatorPID, const char *dllName);
pfun fun = (pfun)GetProcAddress(h, "Run");
fun(CreatorPID,"hh.dll");
```

3.11.4.3. 卸载钩子

HHOOK hHook=SetWindowsHookEx(*******);

UnhookWindowsHookEx(hHook);

3.11.4.4. 钩子传递

钩子处理函数中调用

CallNextHookEx(0, nCode, wParam, lParam);

//将消息传递给钩子链中的下一个钩子函数

3.11.4.5. 键盘钩子

WH_KEYBOARD

```
LRESULT CALLBACK KeyboardProc(
  int code,          // 按键消息状态
  WPARAM wParam,   // 虚键码（VK_ESCAPE 这种），对应键盘 ASCII 码
  LPARAM lParam     // 按键信息
)
```

3.11.4.5.1. code

HC_ACTION 代表 lParam 中包含按键信息

HC_NOREMOVE 代表 lParam 中包含按键信息，但是消息没有从队列取出，比如程序调用的是 PeakMessage。

3.11.4.5.2. wParam

//获取键盘按键 a~z，并放入 str 中(使用 ANSI)

```
if(wParam>=65&&wParam<=90)
{
        TCHAR ch=wParam;
        CString str;
        str.Format(_T("%c"),ch);
}
```

3.11.4.5.3. lParam

31 位标识了按键是处于按下还是弹起，1 代表弹起。

处理键盘消息前需要判断，否则会出现两次按键消息。

```
if(lParam>>31&1)
```

0-15 位标识按键重复的次数

29 位标识 alt 键是否按下，1 代表按下

```
//屏蔽 Alt+F4 组合键
if(VK_F4==wParam && (lParam)>>29&1)
        return 1;
```

```
if(((DWORD)lParam&0x40000000) && (HC_ACTION==code))
{
    BOOL b_sft=::GetAsyncKeyState (VK_SHIFT)>>((sizeof(short)*8)-1);
    //代表是否同时按下 shift 键
}
```

3.11.4.6. 鼠标钩子

WH_MOUSE

鼠标钩子的回调函数 MouseProc 中一定要对 code,wParam 的状态进行判断，否则因为鼠标的状态一直变化，执行语句会变成死循环。

if (0 != code) return 0;

WM_MOUSEMOVE==wParam　　表示鼠标移动消息

```
LRESULT CALLBACK MouseProc(
    int nCode,          // hook code
    WPARAM wParam,    // message identifier
    LPARAM lParam      // mouse coordinates
)
```

在 MouseProc 中可以获取位置信息

```
PMOUSEHOOKSTRUCT pStruct=(PMOUSEHOOKSTRUCT)lParam;
WPARAM x = pStruct->pt.x;
WPARAM y = pStruct->pt.y;
```

LPARAM 是指针，所以不能用 postMessage 传递

3.11.5. 打开其他程序

3.11.5.1. WinExec

WinExec("*****.exe",SW_HIDE);

3.11.5.2. CreateProcess

```
PROCESS_INFORMATION processInfo;
STARTUPINFO startupInfo;
::ZeroMemory(&startupInfo, sizeof(startupInfo));
startupInfo.cb = sizeof(startupInfo);
startupInfo.wShowWindow = FALSE;
if (!::CreateProcess(_T("E:\\bbb.exe"), NULL,
        NULL,    // process security
        NULL,    // thread security
        FALSE, // no inheritance
        0,         // no startup flags
        NULL,    // no special environment
        NULL,    // default startup directory
        &startupInfo,
        &processInfo)) return;
```

包含程序的句柄和 PID
processInfo.hProcess
processInfo.dwProcessId

3.11.5.3. ShellExecuteEx

======打开并等待结束
文件路径和参数字符串必须是字面值常量

```
SHELLEXECUTEINFO ShExecInfo = { 0 };
ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);
ShExecInfo.fMask = SEE_MASK_NOCLOSEPROCESS;
ShExecInfo.hwnd = NULL;
ShExecInfo.lpVerb = NULL;
ShExecInfo.lpFile = _T("C:\\Python36\\python.exe");//exe 或文件
ShExecInfo.lpParameters = _T("D:\\t.py");//参数
ShExecInfo.lpDirectory = NULL;
ShExecInfo.nShow = SW_HIDE;//后台打开程序
ShExecInfo.hInstApp = NULL;
ShellExecuteEx(&ShExecInfo);
WaitForSingleObject(ShExecInfo.hProcess, INFINITE);//等待结束
CloseHandle(ShExecInfo.hProcess);
```

======结束程序
TerminateProcess(sei.hProcess,0);

3.11.5.4. 控制其他程序

使用 VS 下 工具->SPY++ 选择工具查找窗口，可找到对应程序窗口的类名和窗口标题

```
HWND    hWnd=::FindWindow(NULL,_T("name"));

if(hWnd==NULL){return;}

PostMessage (hWnd,WM_USER+1,0,0);

//只能接收PostMessage发的消息
```

如果接收消息的程序的border属性为none，则需要初始时：

```
SetWindowText(hWnd, TEXT("name"));
```

移动目标程序的窗口

::MoveWindow(hWnd,0,0,500,500,TRUE);

3.11.6. 读写目标程序内存

```
void WriteExeMemory(HANDLE hProcess,UINT64 addr,DWORD val)
{
    //DWORD readVal = 0;
    //ReadProcessMemory(hProcess,(LPVOID)addr,&readVal,sizeof(readVal),0);

    BOOL ret = WriteProcessMemory(hProcess,(LPVOID)addr,&val,sizeof(val),0);
}
```

3.11.7. WM_COPYDATA

发送 WM_COPYDATA 后，要等客户端执行该消息函数结束后
::SendMessage(hWnd,(UINT)WM_COPYDATA,0,(LPARAM)&cds);代码之后的才执行
WM_COPYDATA 传输只读数据

3.11.7.1. 发送端

```
HWND    hWnd=::FindWindow(NULL,_T("name")); //接收端窗口名
if(hWnd==NULL){return;}
COPYDATASTRUCT cds;
```

```
cds.dwData=1;//用户定义数据
cds.lpData="abcd";//数据指针
cds.cbData=5;//数据长度
::SendMessage(hWnd,(UINT)WM_COPYDATA,(WPARAM)m_hWnd,(LPARAM)&cds);
```

3.11.7.2. 接收端

处理 WM_COPYDATA 消息

```
BOOL C**Dlg::OnCopyData(CWnd* pWnd, COPYDATASTRUCT* pCopyDataStruct)
{
	CStringA	str = (LPSTR)pCopyDataStruct->lpData;

	return CDialog::OnCopyData(pWnd, pCopyDataStruct);
}


//PCOPYDATASTRUCT lpcds=(PCOPYDATASTRUCT)lParam;
//(LPSTR)lpcds->lpData
```

3.11.8. 共享内存

```
#define Share_Name	_T("sharename")
struct	Stu
{
	int acc;
	char ch[100];
};
const int ShareSize = sizeof(Stu);
```

3.11.8.1. 创建端

```
Stu* Create()
{
	HANDLE hMapFile
=CreateFileMapping(INVALID_HANDLE_VALUE,NULL,PAGE_READWRITE,0,ShareSize,Share_Name)
;
	if(hMapFile==NULL||hMapFile==INVALID_HANDLE_VALUE)	return NULL;
	return (Stu*)MapViewOfFile(hMapFile,FILE_MAP_ALL_ACCESS,0,0,ShareSize);
}
```

3.11.8.2. 读取端

```
Stu* Read()
{
    HANDLE hMapFile=OpenFileMapping(FILE_MAP_ALL_ACCESS,FALSE,Share_Name);
    if(hMapFile==NULL)      return NULL;

    Stu *ptr = (Stu*)MapViewOfFile(hMapFile,FILE_MAP_ALL_ACCESS,0,0,ShareSize);
    CloseHandle(hMapFile);
    return ptr;
}
```

3.11.9. 剪贴板

3.11.9.1. 写入

```
void WriteClipboard(char* pStr,int strLen)
{
    if (!OpenClipboard(NULL)) return;

    EmptyClipboard();

    HANDLE hClip = GlobalAlloc(GMEM_MOVEABLE, strLen);
    void *pBuf = GlobalLock(hClip);
    memcpy_s(pBuf, strLen, pStr, strLen);

    SetClipboardData(CF_TEXT, hClip);//CF_TEXT表示当前使用文本类型数据

    GlobalUnlock(hClip);
    CloseClipboard();
}
```

3.11.9.2. 获取

```
CString GetClipboard()
{
    if (!OpenClipboard(NULL)) return _T("");
    if (!IsClipboardFormatAvailable(CF_TEXT)) return _T("");

    HANDLE hClip = GetClipboardData(CF_TEXT);
    if (NULL == hClip) return _T("");
```

```
    char *pBuf = (char*)GlobalLock(hClip); //取出数据
    CString str(pBuf);

    GlobalUnlock(hClip);
    CloseClipboard();

    return str;
}
```

3.11.10. 有名管道

Windows 上的客户端-服务器模式

3.11.10.1.服务端

HANDLE hPipe=NULL;
析构时：
if(hPipe){CloseHandle(hPipe);}

初始化：

```
    hPipe=CreateNamedPipe(_T("\\\\.\\pipe\\Mypipename"),PIPE_ACCESS_DUPLEX
|FILE_FLAG_OVERLAPPED,0,1,1024,1024,0,NULL); //第一个参数格式固定，只有最后的名字 Mypipename 可以改变
     if(INVALID_HANDLE_VALUE==hPipe)
    {
        hPipe=NULL;
        return ;
    }
    HANDLE hEvent=CreateEvent(NULL,TRUE,FALSE,NULL);
    if(!hEvent)
    {
        CloseHandle(hPipe);
        hPipe=NULL;
        return ;
    }
    OVERLAPPED overlap;
    ZeroMemory(&overlap,sizeof(OVERLAPPED));
    overlap.hEvent=hEvent;
    if(!ConnectNamedPipe(hPipe,&overlap))//等待客户端连接
    {
```

```
        if(ERROR_IO_PENDING!=GetLastError())
        {
            CloseHandle(hPipe);
            CloseHandle(hEvent);
            hPipe=NULL;
            return ;
        }
    }
    if(WAIT_FAILED==WaitForSingleObject(hEvent,INFINITE))
    {
        CloseHandle(hPipe);
        CloseHandle(hEvent);
        hPipe=NULL;
        return ;
    }
    CloseHandle(hEvent);
```

读取：

```
    TCHAR buf[100];
    DWORD dwRead;
    if(!ReadFile(hPipe,buf,100,&dwRead,NULL))
    {return ;}

    CString str;
    str.Format(_T("%s"),buf);
```

写入：

```
    TCHAR buf[]=_T("服务端....."); //要写入的数据
    DWORD dwWrite;
    if(!WriteFile(hPipe,buf,lstrlen(buf)+1,&dwWrite,NULL))
    {return ;}
```

3.11.10.2. 客户端

```
HANDLE hPipe=NULL;
```

析构时：

```
if(hPipe){CloseHandle(hPipe);}
```

初始化：

```
    if(!WaitNamedPipe(_T("\\\\.\\pipe\\Mypipename"),NMPWAIT_WAIT_FOREVER)
```

```
)
    {return ;}
    hPipe=CreateFile(_T("\\\\.\\pipe\\Mypipename"),GENERIC_READ|GENERIC_WRI
TE,0,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
    if(INVALID_HANDLE_VALUE==hPipe)
    {
        hPipe=NULL;
        return ;
    }
```

读取：

```
    TCHAR buf[100];
    DWORD dwRead;
    if(!ReadFile(hPipe,buf,100,&dwRead,NULL))
    {return ;}

    CString str;
    str.Format(_T("%s"),buf);
```

写入：

```
    TCHAR buf[]=_T("客户端....."); //要写入的数据
    DWORD dwWrite;
    if(!WriteFile(hPipe,buf,lstrlen(buf)+1,&dwWrite,NULL))
    {return ;}
```

3.11.11. 匿名管道

只能在父子进程间通信

3.11.11.1. 父进程

```
//用于读写的句柄
HANDLE hRead=NULL;
HANDLE hWrite=NULL;
```

析构时：

```
if(hRead){CloseHandle(hRead);}
if(hWrite){CloseHandle(hWrite);}
```

初始化:

```
 SECURITY_ATTRIBUTES sa;
sa.bInheritHandle=TRUE;
sa.lpSecurityDescriptor=NULL;
sa.nLength=sizeof(SECURITY_ATTRIBUTES);
if(!CreatePipe(&hRead,&hWrite,&sa,0))//创建匿名管道
{ return ;}
 //准备相关结构体
STARTUPINFO sui;
PROCESS_INFORMATION pi;
ZeroMemory(&sui,sizeof(STARTUPINFO));
sui.cb=sizeof(STARTUPINFO);
sui.dwFlags=STARTF_USESTDHANDLES;
sui.hStdInput=hRead;
sui.hStdOutput=hWrite;
sui.hStdError=GetStdHandle(STD_ERROR_HANDLE);

if(!CreateProcess("d:\\child.exe",NULL,NULL,NULL,TRUE,0,NULL,NULL,&sui,&pi))
{//创建子进程     假设子进程放在 d 盘，名为 child.exe
    CloseHandle(hRead);
    CloseHandle(hWrite);
    hRead=NULL;
    hWrite=NULL;
    return ;
}
else
{
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

读取：

```
TCHAR buf[100];
DWORD dwRead;
if(!ReadFile(hRead,buf,100,&dwRead,NULL))
{return ;}

CString str;
str.Format(_T("%s"),buf);
```

写入：

```
TCHAR buf[]=_T("父进程....."); //要写入的数据
```

```
DWORD dwWrite;
if(!WriteFile(hWrite,buf,lstrlen(buf)+1,&dwWrite,NULL))
{return ;}
```

3.11.11.2. 子进程

```
//用于读写的句柄
HANDLE hRead=NULL;
HANDLE hWrite=NULL;
```

析构时：

```
if(hRead){CloseHandle(hRead);}
if(hWrite){CloseHandle(hWrite);}
```

初始化:

```
hRead=GetStdHandle(STD_INPUT_HANDLE);
hWrite=GetStdHandle(STD_OUTPUT_HANDLE);
```

读取：

```
TCHAR buf[100];
DWORD dwRead;
if(!ReadFile(hRead,buf,100,&dwRead,NULL))
{return ;}

CString str;
str.Format(_T("%s"),buf);
```

写入：

```
TCHAR buf[]=_T("子进程.....");
DWORD dwWrite;
if(!WriteFile(hWrite,buf,lstrlen(buf)+1,&dwWrite,NULL))
{return ;}
```

3.11.12. 邮槽

无连接，单向通信，服务器读取，客户端写入。

3.11.12.1. 服务端

```
HANDLE hMailslot;
hMailslot=CreateMailslot(_T("\\\\.\\mailslot\\MySlot"),0,MAILSLOT_WAIT_FOREVER,NULL);//第一个参数格式固定，只有最后的名字 MySlot 可以改变
if(INVALID_HANDLE_VALUE==hMailslot)
{return ;}

TCHAR buf[100];
DWORD dwRead;
if(!ReadFile(hMailslot,buf,100,&dwRead,NULL))
{
    CloseHandle(hMailslot);
    return ;
}

CString str;
str.Format(_T("%s"),buf);
CloseHandle(hMailslot);
```

3.11.12.2. 客户端

```
HANDLE hMailslot;
hMailslot=CreateFile(_T("\\\\.\\mailslot\\MySlot"),GENERIC_WRITE,FILE_SHARE_READ,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
if(INVALID_HANDLE_VALUE==hMailslot)
{return ;}

TCHAR buf[]=_T("客户端.....");//要发送的数据
DWORD dwWrite;
if(!WriteFile(hMailslot,buf,lstrlen(buf)+1,&dwWrite,NULL))
{
    CloseHandle(hMailslot);
    return ;
}
CloseHandle(hMailslot);
```

第四章 MFC

4.1. function

4.1.1. 获取窗口对象

CWnd *pWnd = CWnd::FromHandle(hwnd);

4.1.2. 时间

CTime time(2001,1,1,13,13,13);

//构造时间 2001 年 1 月 1 日 13：13：13

//也可以使用类的其他构造函数构造时间

CTime time=CTime::GetCurrentTime();

CString str=time.Format("%Y-%m-%d %H:%M:%S");

//也可以使用类中的其他函数单独获取年月日等，用作文件名时，不能带冒号

int year=time.GetYear();

//可以让两个时间相减，得到间隔时间

CTimeSpan t = ctime2 - ctime1;

__time64_t x = t.GetTimeSpan(); //得到两个时间的秒差

t.GetMinutes(); //通过该类函数可以获取两个时间之间年、月、小时

等直接的时差

4.1.3. 调试

TRACE("xx");//可以将 CString 输出到调试窗口，debug 有效

TRACE(_T("\r\n"));

4.1.4. 消息盒子

4.1.4.1. 使用

if(IDCANCEL==AfxMessageBox(_T("close"),MB_OKCANCEL)) return;

4.1.4.2. 修改标题

重写 App 类的虚函数 DoMessageBox

将函数中内容替换为：

LPCTSTR pOldAppName=m_pszAppName;

m_pszAppName=_T("新标题");

int i=CWinApp::DoMessageBox(lpszPrompt, nType, nIDPrompt);

m_pszAppName=pOldAppName;

return i;

4.1.5. 截获消息

在虚函数 PreTranslateMessage 中：

if(pMsg->message==WM_KEYDOWN){

//屏蔽关闭按键

```
if (pMsg->wParam == VK_ESCAPE || pMsg->wParam == VK_RETURN) return TRUE;
```

}

对于其他程序发的消息，只能接收 PostMessage 发送的。

```
if (pMsg->message == WM_SYSKEYDOWN && pMsg->wParam == VK_F4)  return TRUE;
```

4.1.6. 右键菜单

菜单资源 IDR_MENU1，使用第一列中的菜单内容

```
CMenu m_menu;
m_menu.LoadMenu(IDR_MENU1);
```

动态创建

```
CMenu menu;
menu.CreatePopupMenu();
menu.AppendMenu(MF_STRING, 消息id,_T("cmd1"));//消息id
```

```
POINT pt;
GetCursorPos(&pt);
m_menu.GetSubMenu(0)->TrackPopupMenu(TPM_LEFTALIGN,pt.x,pt.y,this);
```

4.1.7. 屏蔽 f10

```
if (pMsg->message == WM_SYSKEYDOWN) if (pMsg->wParam == VK_F10) return TRUE;
```

4.1.8. 屏蔽帮助

CXXXApp.cpp中去掉：

```
ON_COMMAND(ID_HELP,  &CWinApp::OnHelp)
```

4.2. 消息

4.2.1. 消息映射机制

4.2.1.1. 基本条件

1.类必须从 CCmdTarget 派生

2.类内必须声明宏函数 DECLARE_MESSAGE_MAP()

afx_msg .....处理函数原型

3.类外必须添加实现宏函数

BEGIN_MESSAGE_MAP(derived Class, base Class)

1） ON_MESSAGE(WM_xxx,处理函数名）

2） ON_EN(BN)_XXX(ID,处理函数名)

3) ON_COMMAND(ID,处理函数名)

4) ON_WM_XXXX()

END_MESSAGE_MAP()

4.类外实现消息处理函数

4.2.1.2. 处理顺序

Frame，app

当处理同一个消息时，如果 frame 中有处理函数，则使用 Frame 中的处理函数，如果没有则使用 app 中的处理函数。

4.2.2. 消息分类

4.2.2.1. windows 标准消息

键盘、鼠标、定时器……

ON_WM_XXXX()

Exp:

```
class CMyFrameWnd:public CFrameWnd{
DECLARE_MESSAGE_MAP()  类内声明宏
public:
    afx_msg int OnCreate( LPCREATESTRUCT    pcs);
```

可在 msdn 中查找到系统消息 ON_WM_CREATE( )对应的处理函数原型，其中的参数可以不写或者只写其中的个别需要用的参数，但是原来参数的位置顺序不可变

```
};
BEGIN_MESSAGE_MAP(CMyFrameWnd,CFrameWnd)
                参数分别为当前类，及其父类
ON_WM_CREATE( )
                系统消息声明
END_MESSAGE_MAP()
int CMyFrameWnd::OnCreate(LPCREATESTRUCT pcs ){
   处理函数的实现
AfxMessageBox(_T("xxxxxxxx"));
```

return CFrameWnd::OnCreate(pcs);

}

4.2.2.2. 自定义消息

#define WM_XXXX WM_USER+n

SendMessage/PostMessage

Exp:

A 类发消息，就在 A 类的头文件中定义消息。如果由 B 类响应这个消息，那么 B 类包含 A 类头文件

#define WM_MY WM_USER+10

任意一个函数中发送消息：

::SendMessage(this->m_hWnd,WM_MY,1,2);

1,2 代表附加信息

BEGIN_MESSAGE_MAP(****,****)

ON_MESSAGE(WM_MY,OnMyMessage)

OnMyMessage 为自定义处理函数

END_MESSAGE_MAP()

类内声明：

DECLARE_MESSAGE_MAP()

```
afx_msg    LRESULT OnMyMessage(WPARAM
                                wParam,    LPARAM lParam);
```

类外实现：

```
LRESULT    CMyFrameWnd::OnMyMessage(WPARAM
                                wParam,    LPARAM lParam){
        CString    str;
        str.Format("define %d %d",wParam,lParam);
        AfxMessageBox(str);
        return 0;
}
```

4.2.2.3. 命令消息

WM_COMMAND

一般是菜单项目的响应消息

4.2.2.4. 通知消息

ON_BN_CLICKED

ON_EN_CHANGE

ON_通知码

4.3. file

4.3.1. 删除目录下所有文件

```
void DeleteDirectory(CString directory_path)
{
	CFileFind finder;
	CString path;
	path.Format(_T("%s/*.*"),directory_path);
	BOOL bWorking = finder.FindFile(path);
	while(bWorking){
		bWorking = finder.FindNextFile();
		if(finder.IsDirectory() && !finder.IsDots()){//处理文件夹
			DeleteDirectory(finder.GetFilePath()); //递归删除文件夹
			RemoveDirectory(finder.GetFilePath());
		}
		else{//处理文件
			DeleteFile(finder.GetFilePath());
		}
	}
}
```

4.3.2. 文件状态

```
CFileStatus    status;

CFile::GetStatus("D:\\1.txt",status);

======用 CTimeSpan 修改时间

CTimeSpan      span(7,0,0,0);//天数，时，分，秒

status.m_mtime-=span;

//将文件修改时间状态信息提前七天

=======用 CTime 修改时间

CTime time(2001,1,1,13,13,13);

status.m_ctime=time;
```

CFile::SetStatus("D:\\1.txt",status);//确认修改

属性修改

status.m_attribute|=FILE_ATTRIBUTE_HIDDEN;

status.m_attribute|=FILE_ATTRIBUTE_READONLY;//设置为只读以后就不可用对 status 进行修改

CTimeSpan 的成员变量（CTime 类型）：

m_ctime （Create）

m_mtime (last modified)

m_atime (last accessed for reading)

4.3.3. 逐行读写

4.3.3.1. 读文件

//CStdioFile file(_T("D:\\test.csv"), CStdioFile::modeRead|CStdioFile::typeBinary);//二进制读写和文本读写对\n 的解释不同

```
CStdioFile file;
if(!file.Open(_T("D:\\test.csv"),CStdioFile::modeRead))
{return;}
```

```
CString strline;
while(file.ReadString(strline)) {AfxMessageBox(strline);}
```

4.3.3.2. 写文件

Unicode 下不能写中文

```
CStdioFile file;
if(!file.Open(_T("D:\\1.csv"),
CStdioFile::modeWrite|CStdioFile::modeCreate))    return;

CString line1(_T("a,b,c\n"));
file.WriteString(line1);
CString line2(_T("1,2,3\n"));
file.WriteString(line2);

file.Close();
```

4.3.4. 序列化

```
class Test:public CObject
{
    DECLARE_SERIAL(Test)
public:
```

```
    int x;
    double y;

    virtual void Serialize(CArchive& ar)
    {
        if (ar.IsStoring())
        {
            ar<<x<<y;
        }
        else
        {
            ar>>x>>y;
        }

    }
};
IMPLEMENT_SERIAL(Test,CObject,1);
```

写入：

```
    Test tt;
    CFile
```

```
file(_T("D:\\tt.archive"),CFile::modeCreate|CFile::modeWrite);
    CArchive oar(&file,CArchive::store);

    oar<<&tt;
    oar.Close();
    file.Close();
```

读取:

```
    Test *pTT = new Test();
    CFile file(_T("D:\\tt.archive"),CFile::modeRead);
    CArchive oar(&file,CArchive::load);

    oar>>pTT;
```

4.3.5. 文档类

4.3.5.1. 保存功能

视图类显示数据，文档类中的成员负责保存数据，自己定义的类是个中间变量，可以管理某类型的数据

文档类中使用一个数组保存自定义的类的多个对象

1）定义图形类 A(以画小圆圈为例)

成员变量:CRect m_rect;

2）在文档类中添加成员变量

CObArray　m_array;

数组类，可以存任何 CObject 类及其派生类的数据，和数组用法相似

3）在视图类的某个消息中确认保存数据到文档类的成员

如鼠标左键消息(鼠标点击一下则画一个小圆圈，同时 new 出一个 A 的对象，然后将 new 出的对象保存到文档类的 m_array 中)

OnLButtonDown(UINT nFlags, CPoint point) ：

CRect　rect(point.x-10,point.y-10,point.x+10,point.y+10);

CClientDC dc(this);

dc.Ellipse(rect);

A *pa=new A;

pa->m_rect=rect;

CADoc* pDoc = GetDocument();

在 onDraw 函数中就可以不用写这句，系统已经自动添加过

pDoc->m_array.Add(pa);

如果将 a 进行 delete 操作，则 m_array 中的成员消失

视图类 cpp 中需要包含自定义类 A 的头文件（包含的头文件必须写在 stdafx.h 下面一行，不能写在上面）

4）可在 OnDraw 函数中从 m_array 中取出变量，然后重新绘图，这样缩放窗口就不会出现问题

```
for(int i=0;i<pDoc->m_array.GetSize();i++){
        A *pa=(A*)pDoc->m_array[i];
```

```
        pDC->Ellipse(pa->m_rect);
}
```

5）在文档类中释放 new 出的对象

用一个函数 clear(文档类内的自定义函数)封装以下代码

```
for(int i=0;i<m_array.GetSize();i++)
        delete    m_array[i];
m_array.RemoveAll();
```

必须存在此句，否则会出错

文档类析构函数中调用 clear

C**Doc::OnNewDocument()中也需要调用 clear，因为是新建文件

6）在文档类 C**Doc::Serialize(CArchive &ar)虚函数中保存文件：

```
    if (ar.IsStoring())
    {
        int count=m_array.GetSize();
        ar<<count;
        for(int i=0;i<count;i++){
            A *pa=(A*)m_array[i];
            CRect    rect=pa->m_rect;
            ar<<rect.left<<rect.top<<rect.right<<rect.bottom;
        }
    }
    else
    {
```

```
        clear();
        int count;
        ar>>count;
        for(int i=0;i<count;i++){
            A *pa=new A;
            CRect    &rect=pa->m_rect;
            ar>>rect.left>>rect.top>>rect.right>>rect.bottom;
            m_array.Add(pa);
        }
    }
```

在存储数据时，一定要注意数据的存储顺序和变量名，还有流的方向不可写错，否则在打开时会出现空白或者错误数据。

文档类的序列化是在打开和保存文件时系统自动进行调用

自定义类时，不使用类向导(注意先包含 stdafx.h)，然后在自己定义的类中进行序列化，这样就可以使用不同版本号，而且因为自定义类已经有了序列化，所以可以使用 ar>>pa 的这种语法。但是从类向导中弄出的类没有版本号，而且序列化后无法使用 ar>>pa 语法。

4.3.5.2. 文件类型

字符串表资源第一个 IDR_MAINFRAME

有六个\n,七个成员

在 msdn 的 GetDocString,其中有对七个成员的解释

windowTitle， docName ， fileNewName， filterName， filterExt，

regFileTypeId，regFileTypeName

最后两个参数是将文件写入注册表

第四和第五个参数是保存类型和文件名后缀类型

如：/n name(*.jyl)/n .jyl　　第五个参数只能为空，或者写成文件后缀

显示时： 文件名　**.jyl

保存类型　name(*.jyl)

在新建工程第四步，高级选项中可以设置

4.3.5.3. 撤销/还原

```
void CDrawDoc::undo()
{
    int count=m_array.GetSize();
    if(count){
    CMyShap *shap=(CMyShap *)m_array[count-1];
    m_array.RemoveAt(count-1);
    m_temp.Add(shap);
    }
}
void CDrawDoc::redo()
{
    int count=m_temp.GetSize();
```

```
    if(count){
    CMyShap    *shap=(CMyShap *)m_temp[count-1];
    m_temp.RemoveAt(count-1);
    m_array.Add(shap);
    }
}
```

4.3.5.4. 双击打开文件

在 C**App 类中的 InitInstance 函数中加入：

```
m_pMainWnd->DragAcceptFiles();  //拖拽打开功能
EnableShellOpen();
RegisterShellFileTypes(TRUE);
```

需要加在 AddDocTemplate 后面，否则会报错

4.4. draw

4.4.1. 绘图处理位置

画图操作不能写在 Create 消息中

不能在 OnDraw 函数或者 WM_PAINT 中刷新窗口，否则会死循环

4.4.1.1. onDraw 函数

(视图类中的虚函数)中用

virtual void OnDraw(CDC *pDC);

pDC->TextOut(200,200,"xxxx");

4.4.1.2. OnPaint

（WM_PAINT 消息）函数中，常用于对话框

CPaintDC dc(this);

dc.TextOut(34,234,"ssf");

写了 WM_PAINT 消息会使 onDraw 函数失效，因为在 WM_PAINT 消息响应函数中封装了，将 dc 传到 onDraw 函数，所以如果重写了该消息响应函数，会使 onDraw 得不到调用。

4.4.1.3. 其他函数中

常使用 CClientDC 这类的 DC

CDC* pDC=GetDC();

pDC->TextOut(300,34,str);

ReleaseDC(pDC);

4.4.2. 刷新

4.4.2.1. 刷新无效问题

可以通过 PostMessage 调用函数刷新控件
刷新时先判断控件是否初始化
if(NULL == GetWindow(GW_CHILD)) return;

OnInitDialog()　初始化后刷新

使用 PostMessage

WM_SIZE 移动和最大化时刷新
消息处理时刷新 if (nType == SIZE_MAXIMIZED) 使用 PostMessage

WM_MOVE 控制移动时刷新
消息处理时刷新

WM_SYSCOMMAND 最小化还原刷新
使用 PostMessage

4.4.2.2. 刷新显示

InvalidateRect(NULL); //刷新整个范围

4.4.2.3. 减少闪烁

4.4.2.3.1. 刷新指定位置

CRect rect;

GetDlgItem(ID)->GetWindowRect(rect); //得到该控件在屏幕的位置

this->ScreenToClient(rect);//函数是对话框的，所以转换为以对话框为基准的坐标系

InvalidateRect(rect);

GetWindowRect 这种函数都是以屏幕坐标系为基准的

GetClientRect 这种函数都是以当前窗口为基准的

4.4.2.3.2. 擦除背景

如果在对话框类中找不到该消息，可以在类向导 class info 中设置消

息过滤为 window 则可以看到该消息。

在 WM_ERASEBKGND 消息处理中，返回值改为 return TRUE,则可以不擦除背景。在返回前进行位图等绘制，可以减少闪烁，但是如果有其他背景绘制，可能会有影响。

4.4.2.3.3. 内存 DC 缓冲

可以将绘图内容以位图形式保存到兼容 DC 中，需要的时候将这个兼容 DC 中的内容绘制出来（缓冲，减少闪烁）。

4.4.2.4. 内存缓冲

```
CDC *pDC= GetDlgItem(ID)->GetDC();
CRect rect;
GetDlgItem(ID)->GetClientRect(&rect);


CDC MemDC;
if (NULL == MemDC.m_hDC) return;
MemDC.CreateCompatibleDC(pDC);
CBitmap bitmap;
bitmap.CreateCompatibleBitmap(pDC,rect.Width(),rect.Height());
MemDC.SelectObject(&bitmap);
MemDC.BitBlt(0,0,rect.Width(),rect.Height(),pDC,0,0,SRCCOPY);

//////////此处对 MemDC 进行任意绘图
HDC hDC = MemDC.GetSafeHdc();
。。。。。。。。


///最终进行显示
pDC->BitBlt(0,0,rect.Width(),rect.Height(),&MemDC,0,0,SRCCOPY);
```

```
ReleaseDC(&MemDC);
ReleaseDC(pDC);
```

4.4.3. DC 分类

4.4.3.1. CDC 类的对象

父类 CObject，封装了 win32 下所有绘图的 API 函数，还封装了 m_hDC(保存了绘图设备句柄)

4.4.3.2. CClientDC

封装了在窗口的客户区中绘图的绘图设备(封装了 CDC 类，在构造时调用了 GetDC，析构时调用了 ReleseDC)

4.4.3.3. CWindowDC

可在标题栏绘图

CWindowDC dc(AfxGetApp()->m_pMainWnd);

dc.TextOut(0,0,"xxx"); 可以画到框架窗口栏，因为以框架为句柄

第三个参数可以直接是 CString 类的对象 str

或者用 GetParent()

如果使用 GetDesktopWindow()则可以在桌面绘图,即在应用程序窗口外部绘图

4.4.3.4. CPaintDC

封装了在 WM_PAINT 消息中绘图的绘图设备

只能用于 WM_PAINT 也就是 OnPaint 函数中（该函数也可以用 CClientDC），其他地方不管用

不能写在函数的 if 这类语句中，要写在函数最外层

4.4.3.5. CMetaFileDC

是一个文件 dc。在调用这个 dc 时，使用 Ellipse 等函数相当于给这个 dc 发绘图命令，让这个 dc 存储这些命令，使用时可以直接将所有的绘图命令执行出来。

4.4.3.5.1. wmf

1）创建

```
CMetaFileDC fileDC;
fileDC.Create();
```

2）使用

不需要调用 OnDraw 这类的函数进行重绘

```
fileDC.Ellipse(point.x-10,point.y-10,point.x+10,point.y+10);
CClientDC dc(this);
HMETAFILE hMetaFile=fileDC.Close();
dc.PlayMetaFile(hMetaFile);
fileDC.Create(); //再次创建
fileDC.PlayMetaFile(hMetaFile);//将之前的图形在 fileDC 上面再画一次，相当于创建 fileDC 后把之前的绘图命令放进去
DeleteMetaFile(hMetaFile);
```

3）保存

```
HMETAFILE hMetaFile=fileDC.Close();
CopyMetaFile(hMetaFile,_T("d:\\f.wmf"));
```

```
fileDC.Create();
DeleteMetaFile(hMetaFile);
```

4）打开

```
HMETAFILE hMetaFile=GetMetaFile(_T("d:\\f.wmf"));
fileDC.PlayMetaFile(hMetaFile);
CClientDC dc(this);
hMetaFile=fileDC.Close();
dc.PlayMetaFile(hMetaFile);
fileDC.Create();
fileDC.PlayMetaFile(hMetaFile);
DeleteMetaFile(hMetaFile);
```

4.4.3.5.2. emf

1）创建

```
CMetaFileDC metaDC;
RECT rectHimm,rectPixel;
```

Init 中

```
HDC hdcRef=GetDC()->m_hDC;
 int iWidthMM=GetDeviceCaps(hdcRef,HORZSIZE);
 int iHeightMM=GetDeviceCaps(hdcRef,VERTSIZE);
 int iWidthPels=GetDeviceCaps(hdcRef,HORZRES);
 int iHeightPels=GetDeviceCaps(hdcRef,VERTRES);
 rectHimm.left=0;
 rectHimm.top=0;
 rectHimm.right=iWidthMM*100;
 rectHimm.bottom=iHeightMM*100;
 rectPixel.left=0;
 rectPixel.top=0;
 rectPixel.right=iWidthPels;
 rectPixel.bottom=iHeightPels;
 metaDC.CreateEnhanced(GetDC(),NULL,&rectHimm,NULL);
```

2）使用

```
metaDC.Ellipse(point.x-10,point.y-10,point.x+10,point.y+10);

 HENHMETAFILE hemf=metaDC.CloseEnhanced();
 PlayEnhMetaFile(GetDC()->m_hDC,hemf,&rectPixel);
 metaDC.CreateEnhanced(GetDC(),NULL,&rectHimm,NULL);
 metaDC.PlayMetaFile(hemf,&rectPixel);
```

```
DeleteEnhMetaFile(hemf);
```

3）保存 //打开和保存后无法编辑

```
HENHMETAFILE hemf=metaDC.CloseEnhanced();
CopyEnhMetaFile(hemf,_T("d:\\x.emf"));
DeleteEnhMetaFile(hemf);
```

4）打开

```
HENHMETAFILE hemf=GetEnhMetaFile(_T("d:\\x.emf"));
UINT size=GetEnhMetaFileHeader(hemf,0,NULL);
ENHMETAHEADER *emHeader=(ENHMETAHEADER *)malloc(size);
GetEnhMetaFileHeader(hemf,size,emHeader);
RECTL rect1=emHeader->rclBounds;
RECT rect={rect1.left,rect1.top,rect1.right,rect1.bottom};
PlayEnhMetaFile(GetDC()->m_hDC,hemf,&rect);
DeleteEnhMetaFile(hemf);
```

4.4.3.6. DC 的属性

dc.GetBkColor()可以获得背景颜色

CSize size=dc.GetTextExtent(str); 可以获得 CString 类对象 str 的尺寸

TEXTMETRIC tm;

dc.GetTextMetrics(&tm); 可以获得 dc 中字体的格式（如字体高度）

4.4.3.7. GDI 绘图对象

CPaintDC，CClientDC,CWindowDC 不能使用 delete 函数删除。

自己定义的 CDC，需要用 DeleteDC 删除；定义的 pen 和 brush 要用 DeleteObject 进行删除。

CPen/CBrush/CFont/CBitmap

CGdiObject　父类 CObject

封装了 m_hObject(保存了和对象绑定在一起的 GDI 绘图设备句柄)

4.4.3.7.1. CPen

CClientDC　dc(this);

CPen　pen(PS_SOLID,10,RGB(255,0,0));//画刷类型非 PS_SOLID 时，画笔宽度 1 才有效

CPen　*oldpen=dc.SelectObject(&pen);

dc.SelectObject(oldpen);

pen.DeleteObject();

4.4.3.7.2. CBrush

CBrush　brush(RGB(255,0,0));

dc.SelectObject(&brush)

或者 CBrush　brush(HS_CROSS,RGB(255,0,0));

或者也可以用 CBrush　brush(*bitmap)　位图的画刷

CBrush *pbrush=

CBrush::FromHandle((HBRUSH)GetStockObject(NULL_BRUSH));空画刷

设置画刷背景色，阴影画刷时不设置背景色会是白色

dc.SetBkMode(TRANSPARENT);

4.4.3.7.3. CFont

dc.SetBkMode(TRANSPARENT);可以设置文字背景透明

```
CFont font;
font.CreatePointFont(120,"方正喵呜体"); //120 代表字体大小 12 号
dc.SelectObject(&font);//画文字时，没有使用 pen 和 brush
COLORREF clr=dc.SetTextColor(RGB(255,0,0));
COLORREF clBk=dc.SetBkColor(RGB(0,0,255));//设置文字背景色
dc.TextOut(0,0,"xxxx");       //用设置的红色绘制字体
dc.SetTextColor(clr);
dc.SetBkColor(clBk);
```

DrawText(str,rect,DT_LEFT)可在指定矩形区域用指定方式绘制文字

设置字体大小

```
void SetPdcFont(CDC * pDC, int fontHeight)
{
    CFont font;
    LOGFONT lf;
    memset(&lf, 0, sizeof(LOGFONT));
    lf.lfHeight = fontHeight;
    VERIFY(font.CreateFontIndirect(&lf));
```

```
    CFont* def_font = pDC->SelectObject(&font);
}
```

4.4.4. 文字处理

4.4.4.1. 获取系统所有字体名

```
int CALLBACK GetNameFun(
    _In_ ENUMLOGFONT   *lpelf,
    _In_ NEWTEXTMETRIC *lpntm,
    _In_ DWORD         FontType,
    _In_ LPARAM        lParam
    )
{

    list<CString>  *pfontName = (list<CString>*)lParam;
    pfontName->push_back(lpelf->elfFullName);

    return TRUE;
}


list<CString>  fontName;
    ::EnumFontFamilies(GetDC()->m_hDC, (LPTSTR) NULL, (FONTENUMPROC)GetNameFun,
(LPARAM)&(fontName));
```

4.4.4.2. 文字输出

```
dc.TextOut(x,y,str);
```

中文字符宽度

```
int widthText = tm.tmAveCharWidth*2
```

4.4.4.3. 创建插入符

在 Create 消息中

CClientDC dc(this);

TEXTMETRIC tm;

dc.GetTextMetrics(&tm);　　//获得 DC 中的字体的信息

CreateSolidCaret(tm.tmAveCharWidth/8,tm.tmHeight); //利用字体信息中的宽度和高度创建插入符

ShowCaret();　　//显示插入符

也可以使用位图插入符：

CBitmap bmp; //类内成员变量

bmp.LoadBitmap(IDB_BITMAP1);

CreateCaret(&bmp);

4.4.4.4. 制作输入文本

在输入文字时，每次都是从输入符（pt 的位置）最开始的位置重新刷新显示一下，用户输入的字母不断加入 CString 中，然后重新显示一遍

CString　m_str;　//类内定义一个存储字符串的变量，用来记录一行字符串的值，当换行时，m_str 才需要完全清空

CPoint　m_pt;　//类内定义一个存鼠标上一次点击时的位置的变量，相当于字符串开始绘制的位置，当换行时候，m_pt 才会发生变化，

变化的值是 y 值

类的构造函数中，进行初始化：

```
m_str="";
m_pt=0;
```

在 WM_LBUTTONDOWN 消息中：

```
SetCaretPos(point);     //改变插入符的位置
m_str.Empty();   //将存放字符串的变量内容清空
m_pt=point;      //将当前鼠标的位置保存下来，用来当输入点
```

在 WM_CHAR 消息中：

```
    CClientDC dc(this);
    TEXTMETRIC tm;
    dc.GetTextMetrics(&tm);
     if(13==nChar){   //13代表回车键（CR）,在 msdn 中 ASCII character
                                codes 可查到字母的 ascii 码
        m_str.Empty();              //字符串进行清空
        m_pt.y+=tm.tmHeight;   //按回车时，输入的坐标点变到下一行
    }
    else if(8==nChar){   //8代表退格键（BS）
        COLORREF clr=dc.SetTextColor(dc.GetBkColor());
```

dc.TextOut(m_pt.x,m_pt.y,m_str);//用背景色将字符串画出来一遍，相当于原来的字符串消失

m_str=m_str.Left(m_str.GetLength()-1);//如果是多字节，需要判断 ASCII 码，决定-1还是-2

dc.SetTextColor(clr);

}

else{

m_str+=nChar;　//将字符加入到字符串

}

CSize size=dc.GetTextExtent(m_str);

CPoint pt;　　//用来记录新的插入符的位置，此时不可改变 m_pt 的值

pt.x=m_pt.x+size.cx;

pt.y=m_pt.y;

SetCaretPos(pt);　　//将插入符设置到新的位置

dc.TextOut(m_pt.x,m_pt.y,m_str);　//绘制出字符串

4.4.4.5. 字体逐渐变色效果

在类内定义一个变量 m_width,用来存放渐变的宽度大小

已经用 TextOut 函数画出文字

```
CString str("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");
dc.TextOut(0,10,str);
```

设置一个定时器，在定时器消息处理中：

```
    CClientDC dc(this);
    TEXTMETRIC tm;
     dc.GetTextMetrics(&tm);
    dc.SetTextColor(RGB(255,0,0));
    m_width+=3;
    CRect    rect(0,10,m_width,10+tm.tmHeight);
CString str("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");
dc.DrawText(str,rect,DT_LEFT);
```

如果使用 DT_RIGHT 则会出现文字从最后一个字母开始出现，相当于文字从左向右移动

可以使用

```
CSize size=dc.GetTextExtent(str);
    if(m_width>size.cx)
        m_width=0;
```

进行重置

4.4.5. 图形绘制

4.4.5.1. 绘制点

```
dc.SetPixel(CPoint(100,100),RGB(255,0,0));
```

4.4.5.2. 直线绘制

CPoint pt(12,12);

pt 取代 MoveTo 等函数中 x,y 两个代表点的参数

dc.MoveTo(pt1);

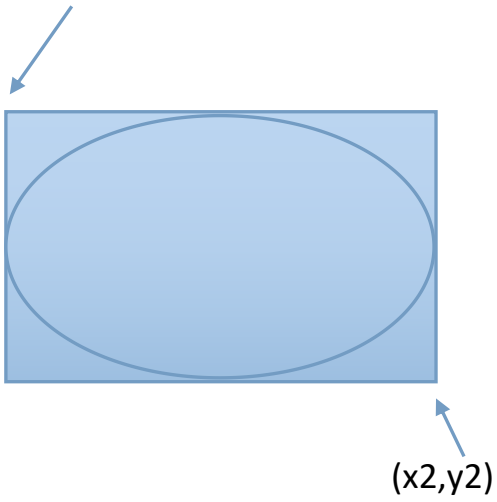
dc.LineTo(pt2);

4.4.5.3. 矩形/椭圆

画椭圆(Ellipse)或者矩形(Rectangle)时，都使用了四个坐标点

dc.Ellipse(rect);

(x1,y1)



(x2,y2)

CRect rect(x1,y1,x2,y2);

rect 可作为 LPCRECT 类型参数传入

4.4.5.4. 画刷/边框绘图

FillRect 函数相当于 Rectangle 函数没有画笔，只有画刷

dc.FillRect(CRect(20,20,100,100),&CBrush(RGB(100,0,0)));

FrameRect 相当于 Rectangle 函数只有画刷，没有画笔

dc.FrameRect(CRect(200,200,400,400),&CBrush(RGB(0,0,100)));

4.4.5.5. 限制 dc 输出区域

对 dc 的输出区域进行设置后，dc 输出时候只在指定区域绘图

```
CClientDC dc(this);
//设置输出区域为五角星
POINT point[5]={{0,200},{600,200},{100,600},{300,0},{500,600}};
HRGN hrgn=CreatePolygonRgn(point,5,WINDING);
::SelectClipRgn(dc.m_hDC,hrgn);
```

4.4.5.6. 曲线

贝塞尔曲线：至少需要四个点，第一个点和最后一个点是首尾，其他点是中间控制点。

```
CClientDC dc(this);
POINT pt[4]={{0,0},{100,50},{100,200},{0,100}};
dc.PolyBezier(pt,4);
```

Polyline(pt,num);　可以将 num 个点连接起来

绘制 sin 曲线：

```
#include <math.h>
#define PI2    2*3.14
#define NUM 1000

CClientDC dc(this);
POINT pt[NUM];
for(int i=0;i<NUM;i++)
{
        int width=400;
        int height=300;
        pt[i].x=i*width/NUM;
        pt[i].y=(int)(height/2*(1-sin(i*PI2/NUM)));//sin 的结果有正负，所以要用
1 去减，全部转化为正数
}
dc.Polyline(pt,NUM);
```

4.4.5.7. 不规则图形

1)利用 CRgn 对象调用一系列 CreateXXX 函数创建基本规则图形

2)调用 CRgn::CombineRng 合并规则图形（按照指定规则合并）

3)调用 CDC::FillRgn 给合并的图形填充颜色

4)调用 CDC::FrameRgn 给合并的图形设置线条颜色

```
CClientDC dc(this);
```

1)
```
CRgn    rgn1;

rgn1.CreateEllipticRgn(0,0,500,500);

CRgn    rgn2;

rgn2.CreateEllipticRgn(0,0,250,250);
```

2)
```
rgn1.CombineRgn(&rgn1,&rgn2,RGN_AND);
```

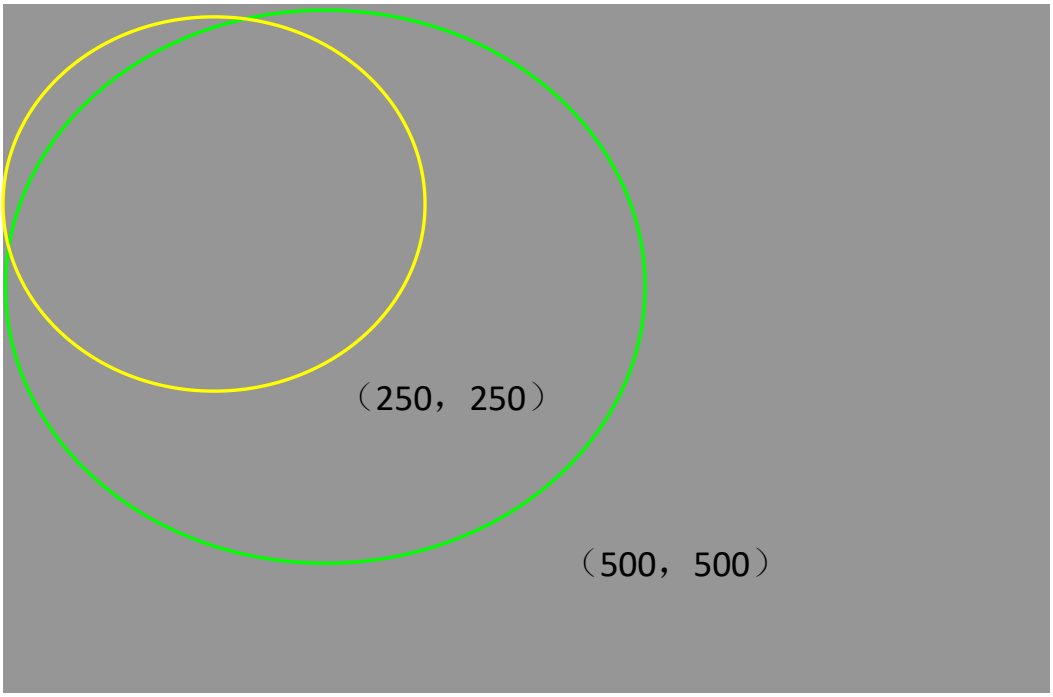
3)
```
CBrush      brushFill(RGB(255,0,0));

dc.FillRgn(&rgn1,&brushFill);
```

4)
```
CBrush      brushFrame(RGB(0,255,0));
```

dc.FrameRgn(&rgn1,&brushFrame,2,2);最后两个参数代表线条的水平方向和垂直方向粗细

两圆形区域的交界处即为合成后的图形



::SetROP2(R2_NOT)

4.4.5.8. 画区域路径

（让两个图在同一区域画图时进行运算）

```
CString str("xxxxxx");
pDC->TextOut(100,100,str);

CSize size=pDC->GetTextExtent(str);  //得到字体区域的大小
pDC->BeginPath();  //开始画选择区域
pDC->Rectangle(100,100,size.cx+100,size.cy+100);
                    //将文字区域放入路径中
```

```
pDC->EndPath();     //结束画选择区域


pDC->SelectClipPath(RGN_DIFF);
                //使下面画的圆圈不会覆盖上面字体的区域，也可用
                    RGN_AND，则圆圈只画区域相交部分
pDC->Ellipse(0,0,300,300);
```

4.4.5.9. 铅笔绘图

```
CPoint ptStart;
BOOL btnDown=FALSE;
void C**View::OnLButtonDown(UINT nFlags, CPoint point)
{
    ptStart=point;
    btnDown=TRUE;
    CView::OnLButtonDown(nFlags, point);
}
void C**View::OnMouseMove(UINT nFlags, CPoint point)
{
    CClientDC    dc(this);
    if(btnDown){
       dc.MoveTo(ptStart);
       dc.LineTo(point);
```

```
        ptStart=point;     //模仿铅笔绘图
    }
    CView::OnMouseMove(nFlags, point);
}
void void C**View::OnLButtonUp(UINT nFlags, CPoint point)
{
    btnDown=FALSE;
    CDialogEx::OnLButtonUp(nFlags, point);
}
```

4.4.5.10. 获取颜色

```
COLORREF ref=dc.GetPixel(CPoint(100,100));
```

4.4.6. 图像绘制

4.4.6.1. DIB

宽度可以非 4 的倍数,
点阵的内存必须是 4 的倍数

```
    int channel = 3;
int Width = 301;
int Height = 300;

int val = Width*channel;
int widstep = val%4 == 0 ? val: val + 4 - val%4;
int dibBufSize = widstep*Height;
```

```
char                chBmpBuf[2048] = { 0 };
BITMAPINFO          *pBmpInfo = (BITMAPINFO *)chBmpBuf;
pBmpInfo->bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
pBmpInfo->bmiHeader.biWidth = Width;//位图的宽度像素为单位
pBmpInfo->bmiHeader.biHeight = Height;//位图的高度
pBmpInfo->bmiHeader.biPlanes = 1;//必须为 1
pBmpInfo->bmiHeader.biBitCount = 8 * channel;//每个像素的位数(必须是 8 的倍数)
pBmpInfo->bmiHeader.biCompression = BI_RGB;//压缩方式，一般为 0 或者 BI_RGB（未压缩）
//info.bmiHeader.biSizeImage 以字节为单位的图像大小（仅用于压缩位图）
//info.bmiHeader.biXPelsPerMeter 以目标设备每米的像素来说明位图的水平分辨率
//info.bmiHeader.biYPelsPerMeter 以目标设备每米的像素来说明位图的垂直分辨率
//info.bmiHeader.biClrUsed 颜色表的颜色数，如果是 0,则使用最大颜色数
//info.bmiHeader.biClrImportant 重要颜色数目，如果是 0，表示所有颜色都重要
//// 黑白图像需要初始化调色板
for (int i = 0;i<256;i++)
{
    pBmpInfo->bmiColors[i].rgbBlue = i;
    pBmpInfo->bmiColors[i].rgbGreen = i;
    pBmpInfo->bmiColors[i].rgbRed = i;
    pBmpInfo->bmiColors[i].rgbReserved = i;
}

//BYTE* PDIBBuffer = NULL;
//HBITMAP hDib = CreateDIBSection(NULL, pBmpInfo, DIB_RGB_COLORS, (void**)&PDIBBuffer, NULL, 0);//HDC 仅在 usage 参数为 DIB_PAL_COLORS 才使用

int bufSize = pBmpInfo->bmiHeader.biBitCount*pBmpInfo->bmiHeader.biWidth*pBmpInfo->bmiHeader.biHeight;
BYTE* PDIBBuffer = new BYTE[bufSize];
memset(PDIBBuffer, bufSize, 0);

//从左下方开始，将图像左下角一半变成红色
for (int row = 0; row <= 150; ++row)
{
    for (int col = 0; col <= 150; ++col)
    {
        int pos = widstep * row + col * channel;
        PDIBBuffer[pos++] = 0;      //B
        PDIBBuffer[pos++] = 0;      //G
        PDIBBuffer[pos++] = 255;    //R
    }
}
```

```
CRect wrect;
GetClientRect(wrect);
CDC *pdc = GetDC();
HDC hdc = pdc->GetSafeHdc();
::SetStretchBltMode(hdc, COLORONCOLOR);
::StretchDIBits(hdc,
    0, wrect.Height(), wrect.Width(), -wrect.Height(),
    0, 0, pBmpInfo->bmiHeader.biWidth, pBmpInfo->bmiHeader.biHeight,
    PDIBBuffer, pBmpInfo,
    DIB_RGB_COLORS, SRCCOPY);//0, wrect.Height(), wrect.Width(), -wrect.Height()可以进行反转显示
ReleaseDC(pdc);
//DeleteObject(hDib);
```

4.4.6.2. 显示 BMP

```
CClientDC dc(this);
CBitmap bmp;

//bmp.LoadBitmap(IDB_BITMAP1);

HBITMAP     hBitmap = (HBITMAP)::LoadImage(
    AfxGetInstanceHandle(), _T("D:\\1.bmp"),
    IMAGE_BITMAP, 0, 0,
    LR_LOADFROMFILE | LR_CREATEDIBSECTION);
bmp.Attach(hBitmap);

//得到图片的尺寸信息
BITMAP BitInfo;
bmp.GetBitmap(&BitInfo);
int width = BitInfo.bmWidth;
int height = BitInfo.bmHeight;

CDC  memDC;
memDC.CreateCompatibleDC(&dc);
CBitmap  *oldbmp = memDC.SelectObject(&bmp);



//dc.BitBlt(100, 100, 800, 700, &memDC, 0, 0, SRCCOPY);
//100，100 代表图片左上角的坐标位置，800，700 代表显示图片的框架的大小(这四个坐标相当于进行图像裁剪)
```

```
//伸缩显示
//可以将图片从本来的 width,height 变成客户区大小显示
CRect rect;
GetClientRect(&rect);
dc.StretchBlt(0, 0, rect.right, rect.bottom, &memDC, 0, 0, width, height, SRCCOPY);


memDC.SelectObject(oldbmp);
memDC.DeleteDC();
bmp.DeleteObject();
```

4.4.6.3. 合成图

在画完位图后，用同一个 dc 设置位置再画一个位图，则会覆盖在前一个位图上面。

```
CClientDC dc(this);
CBitmap bmp;
bmp.LoadBitmap(IDB_BITMAP1);
CDC    memDC;
memDC.CreateCompatibleDC(&dc);
CBitmap    *oldbmp=memDC.SelectObject(&bmp);
dc.BitBlt(500,0,800,700,&memDC,0,0,SRCCOPY);
bmp.DeleteObject();
```

//位图 bmp 中的缓存需要删除；不能删除 memDC，后面需要用；oldbmp 一直保存着原来的内容，结束后在进行还原

```
bmp.LoadBitmap(IDB_BITMAP2);
memDC.SelectObject(&bmp);
dc.BitBlt(700,200,800,700,&memDC,0,0,SRCCOPY);
memDC.SelectObject(oldbmp);
memDC.DeleteDC();
bmp.DeleteObject();
```

4.4.6.4. 去图片底色

原图 255 255 0
掩码 255 0 0

掩码求反 0 255 255
与原图位与 255 255 0
结果 0 255 0

```
CClientDC dc(this);
        CBitmap    bmp;
bmp.LoadBitmap(IDB_BITMAP1);
CDC memDC;
memDC.CreateCompatibleDC(&dc);
memDC.SetBkColor(RGB(0,0,0));//代表要去掉的颜色
CBitmap *oldmap=memDC.SelectObject(&bmp);

CDC    maskDC;//掩码 DC
maskDC.CreateCompatibleDC(&memDC);
CBitmap      maskbmp;
maskbmp.CreateBitmap(400,400,0,0,NULL); //创建掩码位图，两个 400 代表掩码图的图像大小（设置成和位图一样的大小,假设位图为 400*400）
CBitmap *oldmask=maskDC.SelectObject(&maskbmp);
maskDC.BitBlt(0,0,400,400,&memDC,0,0,SRCCOPY);

dc.BitBlt(0,0,400,400,&memDC,0,0,SRCINVERT);
 dc.BitBlt(0,0,400,400,&maskDC,0,0,SRCAND);
dc.BitBlt(0,0,400,400,&memDC,0,0,SRCINVERT);

memDC.SelectObject(oldmap);
memDC.DeleteDC();
bmp.DeleteObject();

maskDC.SelectObject(&oldmask);
maskDC.DeleteDC();
maskbmp.DeleteObject();
```

4.4.6.5. 获取剪贴板图像

```
if (OpenClipboard())
{
    HBITMAP     handle     =     (HBITMAP) GetClipboardData(CF_BITMAP);
    if (handle!=NULL)
    {
        CBitmap* bmp     =     CBitmap::FromHandle(handle);

        CClientDC dc(this);
        CDC    memDC;
        memDC.CreateCompatibleDC(&dc);
        memDC.SelectObject(bmp);
        BITMAP BitInfo;
        bmp->GetBitmap(&BitInfo);
```

```
            int width=BitInfo.bmWidth;
            int height=BitInfo.bmHeight;
            dc.StretchBlt(0,0,width,height,&memDC,0,0,width,height,SRCCOPY);
        }
        CloseClipboard();
}
```

4.5. dialog

4.5.1. 外观

4.5.1.1. 全屏

ModifyStyle(WS_BORDER, WS_OVERLAPPED);或者 Border 属性设置为 None

```
//ShowWindow(SW_SHOWMAXIMIZED);

int cx = GetSystemMetrics(SM_CXSCREEN);
int cy = GetSystemMetrics(SM_CYSCREEN);
MoveWindow(0,0,cx,cy);
```

4.5.1.2. 背景

可以把背景色刷成和背景图片的背景颜色一样，然后把背景图片放到屏幕中间。

注意设置背景色时，是在 OnPaint 函数的 else 里面，必须将 CDialog::OnPaint();注释掉。

```
CBrush brush(RGB(0,0,22));
CPaintDC dc(this);
CRect rectfill;
GetClientRect(rectfill);
dc.FillRect(rectfill,&brush);
```

也可以使用 WM_CTLCOLOR 消息中返回画刷来改变背景色

4.5.1.3. 控件位置

OnSize 函数中调节控件位置要进行判断

```
if (GetWindow(GW_CHILD) != NULL)
{
    GetDlgItem(IDC_BUTTON1)->MoveWindow(0, 0, cx / 2, cy / 2);
```

```
}
```

```
int cx=::GetSystemMetrics(SM_CXSCREEN);
int cy=::GetSystemMetrics(SM_CYSCREEN);
CRect rect;
m_btn.GetClientRect(rect);//GetWindowRect 获取关于屏幕坐标位置
m_btn.MoveWindow(.....);
```

```
xxx.SetWindowPos(&m_ctl, 0, 0, rect.Width(), rect.Height(), 0);
```
第一个参数代表插入到哪个窗口位置后面，如果是 null，则放到顶层。

Ctrl+d　编辑时调整控件 z 序列，数字小控件越顶层，只能点击数字大的，让其变小

4.5.1.4.　颜色和透明背景

除了静态文本，此功能只能用于多字节
可用于对话框，也可用于对话框上的控件（只要是窗体就可以）

添加对话框的 WM_CTLCOLOR 消息响应函数
在 return hbr 上面加入语句进行控制
(设置字体时要注意大小)

```
CFont font;   //成员变量，不能在 CTLCOLOR 中反复创建对象

switch(pWnd->GetDlgCtrlID())
{
    case IDC_TEXT: //静态文本框的 ID，也可用于输入文本及其他控件
        {
        font.DeleteObject();
            font.CreatePointFont(180,"方正喵呜体"); //180 代表 18 号字体
            pDC->SelectObject(&font); //此处只选入，不能在 return 前选回旧的
              pDC->SetTextColor(RGB(255,0,0));   //设置文字颜色
              pDC->SetBkMode(TRANSPARENT);   //透明化文字背景
              return    (HBRUSH)::GetStockObject(NULL_BRUSH);
// HBRUSH b=CreateSolidBrush(RGB(135,206,235)); //成员变量，不能在 CTLCOLOR 中反复创建对象

        }
        break;
    default :
        break;
}
```

如果有某个函数调用时会改变文字（编辑框是 EN_CHANGE 里面）
那么在那个函数中加入

```
CRect rect;
GetDlgItem(ID)->GetWindowRect(rect);
ScreenToClient(rect);
InvalidateRect(rect);    //只刷新控件的显示
```

不能加入到 CTLCOLOR 中

Center Image 属性选择 True 文本则居中显示

4.5.1.5. 标题栏重绘制拖动

处理 NCHITTEST 消息

```
LRESULT CtestmfcDlg::OnNcHitTest(CPoint point)
{
                CRect rect;
                GetClientRect(rect);
                 ClientToScreen(rect);

    //使用某个控件的顶部位置限制标题栏检测底边
                    CRect rt;
    GetDlgItem(IDC_LIST1)->GetWindowRect(rt);
    rect.bottom = rt.top;

    return    rect.PtInRect(point) ? HTCAPTION: CDialog::OnNcHitTest(point);
}
```

OnPaint()函数 else 中设置标题栏颜色

```
    CWnd *pwnd = GetDlgItem(IDC_LIST1);
        if (NULL != pwnd)
        {
            CBrush brush(RGB(0,200,122));
            CPaintDC dc(this);
            CRect rectfill;
            GetClientRect(rectfill);

            CRect rt;
            pwnd->GetWindowRect(rt);
```

```
        ScreenToClient(rt);
        rectfill.bottom = rt.top;
        dc.FillRect(rectfill,&brush);
    }
```

4.5.1.6. 鼠标指针

```
HCURSOR    m_hCurSor;

m_hCurSor = LoadCursorFromFile(_T("skin\\curFile.cur"));
//LoadCursor( NULL   , IDC_CROSS ) ;获取系统定义的鼠标
```

SETCURSOR 消息处理

```
BOOL  Dlg::OnSetCursor(CWnd*  pWnd,  UINT  nHitTest,  UINT
message)
{
    ::SetCursor(m_hCurSor);
    return TRUE;
}
```

App 类最后一行

```
::DestroyCursor(dlg.m_hCurSor);
```

4.5.1.7. 窗口显示状态获取

```
    WINDOWPLACEMENT    lwndpl;
```

```
    GetWindowPlacement(&lwndpl);
  if (SW_SHOWMAXIMIZED == lwndpl.showCmd)
```

4.5.1.8. 获取控件在客户区位置

```
CRect rect;
GetDlgItem(IDC_PIC)->GetWindowRect(rect);

ScreenToClient(rect);
```

4.5.2. 子对话框

插入一个对话框资源，假设 ID 为：IDD_DIALOG1

双击对话框，创建一个类 Cdlg

类内变量

```
Cdlg   *pdlg;

pdlg=new CFL();
pdlg->Create(IDD_DIALOG1);
pdlg->ShowWindow(SW_SHOW);
```

4.5.3. 可变大小

设置勾选属性 Minimize box 和 maximize box
border 属性选择 resizing

4.5.4. 控件功能扩展

每个控件能响应的消息和其中的变量是固定的，如果需要扩展，可以继承这个控件类，然后增加函数和变量。

以按钮为例，增加一个新的类 class CBtnX : public CButton，在使用向导为按钮绑定控件型变量时，不选 CButton，选择 CBtnX 即可（注意在对话框类绑定变量，需要包含 CBtnX 的头文件）。

此时可以为 CBtnX 增加变量以及各种消息响应，这样按钮就可以响应别的消息以及拥有其他功能。

4.5.5. 数据交互

不能在线程中对控件进行调整，只能在线程中对控件状态进行判断。

对话框与外界数据交互：可以为对话框加入一个 GetData 函数，用指针变量成员（m_p）接收外界的变量的指针(m_pData)，然后在对话框中利用这个 m_p 就可以修改外界的数据

```
CDlgwidth dlg;
dlg.GetData(&m_penwidth);
dlg.DoModal();
```

4.5.6. 获取主窗口

```
AfxGetMainWnd()->m_hWnd
```

4.5.7. 消息传递

对话框的控件在收到点击消息时，会拦截消息，使消息在当前控件处理后不再被其他控件或者对话框响应，如果要增加其他的响应，需要在控件中向其他窗口发消息。

4.5.8. CtoolTipCtrl

可以为指定控件（窗口）增加提示，当鼠标移动到控件上时，弹出提示。
类内变量:
CToolTipCtrl m_tt;

初始化:
EnableToolTips(TRUE);
m_tt.Create(this);
m_tt.Activate(TRUE);
m_tt.AddTool(GetDlgItem(ID),"提示内容");

PreTranslateMessage:
m_tt.RelayEvent(pMsg);

4.5.9. 菜单

创建菜单资源
CMenu m_menu;

m_menu.LoadMenu(IDR_MENU1);
SetMenu(&m_menu);

选择属性添加事件处理函数

4.5.10. 按钮

双击可进入代码，编辑按钮点击的响应函数

GetDlgItem(IDC_BUTTON1)->EnableWindow(FALSE);

IsWindowEnabled()

判断按钮是否可用

4.5.11. 文本框

4.5.11.1. 基本操作

编辑框的文本属性样式可以使用密码模式

静态文本如果需要响应点击消息，需要将 style 中勾选 notify，同时修改原来的 IDC_STATIC

```
CString str=_T("");
//获取和设置编辑框内容
GetDlgItemText(IDC_EDIT1,str); //所有控件都可以使用这个获得控件上显示的文字
SetDlgItemText(IDC_EDIT1,str);  //也可用于设置其它控件的文本
//可以将控件属性设置为数字，则只可以输入数字
int i=GetDlgItemInt(IDC_EDIT1);
SetDlgItemInt(IDC_EDIT1,i);
```

4.5.11.2. 静态文本控制

静态文本绑定变量 control 型 m_text

类内变量：CFont m_Font;

初始化时：

```
m_Font.CreatePointFont(300, "宋体", NULL);
m_text.SetFont(&m_Font,true);
```

4.5.11.3. 粘连选择

//复选编辑框内容，相当于鼠标粘连选择，注意设置后需要设置焦点，否则焦点不在编辑框，而看不见效果。setSel 函数第一个参数代表起始位置，第二个代表结束位置（如果-1，代表到结尾）

```
((CEdit*)GetDlgItem(IDC_EDIT1))->SetSel(0,-1);
GetDlgItem(IDC_EDIT1)->SetFocus();
```

4.5.11.4. 回车键换行

Want return 属性 true，在编辑框中按下 Enter 键时即可换到下一行而不关闭对话框。

4.5.11.5. 多行显示

multiline 和 vertical scroll 属性（可以设置 Read Only 属性为 ture）

编辑框绑定变量：数值型 m_show，控件型 m_control

假设要加入的数据是 str

```
if(m_show==_T("")){ //如果是第一次添加，就不换行
        m_show+=str;
}
else{
        m_show+=_T("\r\n")+str;
}
this->UpdateData(FALSE);
```

4.5.11.6. 编辑框日志显示

4.5.11.6.1. 属性

Multiline, Want return,read only 勾选

vertical scroll

4.5.11.6.2. 加入数据

```
static CString log;
log += strLine;
log += _T("\r\n");

m_logCtl.SetWindowText(log);
m_logCtl.LineScroll(m_logCtl.GetLineCount());
UpdateData(FALSE);
```

4.5.11.6.3. 清空

```
m_logCtl.SetWindowText(_T(""));
m_logCtl.LineScroll(0);
UpdateData(FALSE);
```

4.5.12. Check Box

绑定变量或使用 IsDlgButtonChecked(m_hWnd, IDC_CHECK1)进行勾选

判断

方型，打勾选择

//GetCheck 判断勾选状态，SetCheck 设置勾选

```
if(1==((CButton*)GetDlgItem(IDC_CHECK1))->GetCheck()){
        ((CButton*)GetDlgItem(IDC_CHECK1))->SetCheck(0);
}
else{
        ((CButton*)GetDlgItem(IDC_CHECK1))->SetCheck(1);
}
```

======代码创建 CheckBox

```
    CButton *pBtn = new CButton();
    RECT rect = {x,y,x+width,y+height};
    pBtn->Create(_T("                    标                    题
"),WS_TABSTOP|WS_VISIBLE|WS_CHILD|BS_AUTOCHECKBOX,rect,this,id)
;
```

4.5.13. Radio Button

圆形，打点选择（互斥，只能选中一个）

默认以有组属性的单选按钮和该按钮之后连续的按钮为一组，直到遇到下一个组属性的按钮。

ctrl+d 调出 layout，改变先后编号

单选按钮多个时，利用 layout 排序，将顺序弄连续，然后将编号小的控件属性选择组属性，即可分组（可用分组框进行修饰）

分组后，类向导中会有组属性的控件 ID，绑定成员变量，通常使用 int 型。值为-1，代表没有选择任何选项。

4.5.14. Combo Box

支持用户的输入和选择，可下拉

（最好在加入时将高度弄高，修改时需要点击下拉按钮，然后把鼠标放到控件下方中间点才可以修改）

在属性中有数据项目，可以输入项目名称，按 ctrl+回车，切下一行（VS 中用;写表示换行）

如果属性里样式类型改为下拉列表（drop list），就只能选择，不能接受输入，可勾选分类，让其自动分类

Control,CcomboBox

m_combox.EnableWindow(BOOL) 设置组合框是否可用

int nSel=m_combox.GetCurSel();//得到当前选择的位置，如果没有选择则为-1

m_combox.GetLBText(nSel,str);//获取选择位置的数据

4.5.15. 图片控件

类型可以选择位图，图标等

先在资源中导入图片，然后在图片控件的图像中找到选择图片的 ID，则图片控件会显示图片

图片控件还可以选择框架类型，颜色选最后一种，则可做出一条直线（刻在窗口上的线条）

动态加载:

```
CBitmap bitmap;
bitmap.LoadBitmap(IDB_BITMAP1);
HBITMAP hBmp = (HBITMAP)bitmap.GetSafeHandle();
pic.SetBitmap(hBmp); //图片控件绑定变量
```

或者：

```
HBITMAP        hBmp=(HBITMAP)::LoadImage(
                 AfxGetInstanceHandle(),_T("d:\\1.bmp"),
                 IMAGE_BITMAP,0,0,
               LR_LOADFROMFILE|LR_CREATEDIBSECTION); pic.SetBitmap(hBmp);
```

4.5.16. 时间控件

```
CTime time;
DWORD      dwResult      =      m_dateCtl.GetTime(time);
if (dwResult != GDT_VALID)
{
    AfxMessageBox(_T("Time      not      set!"));
}
CString strDate =    time.Format(_T("%Y-%m-%d"));
```

4.5.17. List Box

绑 Control 类型

样式的选择里，可设置单个或者多个的选择，无法通过属性添加数据

```
listbox.SetItemHeight(index,30);//设置列表高度
if(LB_ERR==m_list.FindString(-1,cstring)){
 //搜索判断类表中所有项目名称，如果没有名称 cstring 则执行
        int nIndex=m_list.AddString(cstring);
        CString     *pPath=new    CString;
            //为 list 添加附加数据（比如添加地址，后面可以取出）
           //如果添加的数字是数字，就可以直接加入。。不用这个方
式
        *pPath=m_strFilePath;
         m_list.SetItemData(nIndex,(DWORD)pPath);
}
```

找到列表框 IDC_LIST 的 LBN_DBLCLK（双击消息），然后进入相应的处理函数取出 pPath：

```
int    nSel=m_list.GetCurSel();
if(LB_ERR==nSel) return;
else{
    CString    *pPath=(CString *)m_list.GetItemData(nSel);
}
```

new 出的 pPath 需要考虑释放，使用 m_list.GetCount()统计出数量，在 for 循环中 delete 掉从 GetItemData 中拿出的附加数据

```
int    count=m_list.GetCount();
```

```
for(int i=0;i<count;i++){
    CString *pPath=(CString *)m_list.GetItemData(i);
    delete    pPath;}
```

m_list.GetText(nSel,str);可获得列表框文本

m_list.DeleteString(nSel); 将列表框中某行删除

4.5.18. 调节数值的控件

（可以为控件增加相应的消息处理函数）

设置/获取控件表示的数值(位置)范围:（可以不设置，默认从 0 到 100）

SetRange(int nLower, int nUpper)/GetRange(&nLower, &nUpper)

获取/设置控件的当前位置:

GetPos() /SetPos(int)

响应事件使用对话框的 WM_HSCROLL

或者控件的NM_RELEASEDCAPTURE

4.5.18.1. 旋转按钮（Spin）

通常与编辑框控件一起使用(与编辑框序号连续，且编辑框序号在前，勾选旋转按钮样式 Auto Buddy【自动结伴】，SetBuddy integer【设置结伴整数】，左边排列方式选择靠右。这样在运行时两个控件就会合成一体)，用来完成某个数值的增加或者减少，对应控件类型 CSpinButtonCtrl

设置控件的增量/步长：

```
UDACCEL    accel;
accel.nSec=1;          //改变数值前所经过的秒数（可用于接收）
accel.nInc=2;          //每次数值改变的大小
m_spin.SetAccel(1,&accel);    //第一个参数用来表示有几个 accel，一般写 1
```

4.5.18.2. 进度条（Progress）

通常用于安装程序或者复制文件时进度显示，对应 CProgressCtrl

```
m_progress.SetStep(5);    //设置每次的增量（满值是 100，所以 100/5 就是需要多少次才到满值）
m_progress.StepIt();      //执行此语句时进度条增加
int nPos=m_progress.GetPos();      //得到当前进度值
```

4.5.18.3. 滑块（Slider）

通常用于滑动调节某些数值，对应 CSliderCtrl

```
m_slider.SetRange(0,255); //设置变化范围 0-255
NM_RELEASEDCAPTURE  鼠标拖动后弹起事件
int pos = m_slider.GetPos(); //获取位置
```

4.5.19. 滚动条

在创建窗口时包含窗口样式WS_VSCROLL和WS_HSCROLL即可拥有垂直滚动和水平滚动条。

SetScrollRang函数可以设置卷动列的范围，默认是0到100。如果在这个函数后又使用了其他影响绘制的函数，需要在这个函数的最后一个参数bRedraw设置为FALSE。

SetScrollPos函数可以设置卷动列上面的滑块的当前位置。

用鼠标拖动滑块或者点击卷动列时，会产生WM_VSCROLL和WM_HSCROLL消息。每个动作会产生两个消息，一个是按下时发生的，一个是释放时发生的。

其中的wParam表示了对卷动列的操作，低位是SB_THUMBTRACK时，高位代表拖动时目前的位置，低位是SB_THUMBPOSITION时，高位是释放后的最终位置。

对话框属性 Vertical ScrollBar 选择 TRUE，则可以产生垂直滚动条。在对话框的 WM_VSCROLL 消息中可以进行控制。

```
class ScrollBarCtl
    {
     int m_ScrollType;//类型 SB_VERT 或者 SB_HORZ

     int m_curScrol;//当前位置
```

```
    int m_ScrollMin;//最小值
    int m_ScrollMax;//最大值
    int m_Scrolldelta;//滚动变化量

    CWnd *m_pScroll;
   public:
    ScrollBarCtl(int
_ScrollType):m_ScrollType(_ScrollType),m_curScrol(0),m_ScrollMin(0),m
_ScrollMax(100),m_pScroll(NULL),m_Scrolldelta(1){}
    void Init(int _ScrollMin,int _ScrollMax,int _Scrolldelta,CWnd
*_pScroll)
    {
        m_Scrolldelta = _Scrolldelta;
        m_curScrol = 0;
        m_ScrollMin = _ScrollMin;
        m_ScrollMax = _ScrollMax;
        m_pScroll = _pScroll;

        if (NULL == m_pScroll) return;
        m_pScroll->SetScrollRange(m_ScrollType,0,m_ScrollMax);
    }
```

```
void OnScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    if (NULL == m_pScroll) return;



    SCROLLINFO vSI;
    m_pScroll->GetScrollInfo(m_ScrollType,&vSI);
    switch (nSBCode)
    {
    case SB_LINEUP: //点击滚动条最上面
        vSI.nPos = 0;
        break;
    case SB_LINEDOWN: //点击滚动条最下面
        vSI.nPos = m_pScroll->GetScrollLimit(m_ScrollType);
        break;
    case SB_PAGEUP:
        break;
    case SB_PAGEDOWN:
        break;
    case SB_THUMBPOSITION:
        break;
    case SB_THUMBTRACK: //拖动滚动条
```

```
            vSI.nPos = nPos;
            break;
                case SB_ENDSCROLL: //结束拖动滚动条
            break;
        case SB_TOP:
            vSI.nPos = 0;
            break;
        case SB_BOTTOM:
            vSI.nPos = INT_MAX;
            break;

    }

    m_curScrol = vSI.nPos;
    m_pScroll->SetScrollInfo(m_ScrollType,&vSI);
}

void ProcMOUSEWHEEL(int delta)
{
    if (NULL == m_pScroll) return;
```

```
        if (delta > 0 )
            m_curScrol -= m_Scrolldelta;
        else
            m_curScrol += m_Scrolldelta;



        if (m_curScrol<0) m_curScrol = 0;
        if (m_curScrol>m_ScrollMax) m_curScrol=m_ScrollMax;



        m_pScroll->SetScrollPos(m_ScrollType,m_curScrol);
    }
    int GetCurrentScrol()
    {
        return m_curScrol;
    }
};

ScrollCtl    m_ScrollVert(SB_VERT);
ScrollCtl    m_ScrollHORZ(SB_HORZ);
```

```
    void C**Dlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar*
pScrollBar)
    {
     m_ScrollHORZ.OnScroll(nSBCode,nPos,pScrollBar);

     CDialogEx::OnHScroll(nSBCode, nPos, pScrollBar);
    }
    void C**Dlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar*
pScrollBar)
    {
     m_ScrollVert.OnScroll(nSBCode,nPos,pScrollBar);
     m_ScrollImgCtl.Show(m_ScrollVert.GetCurrentScrol());
     CDialogEx::OnVScroll(nSBCode, nPos, pScrollBar);
    }
    BOOL C**Dlg::PreTranslateMessage(MSG* pMsg)
    {
     UINT msg = pMsg->message;
     if (msg == WM_MOUSEWHEEL)
     {
          int delta =GET_WHEEL_DELTA_WPARAM(pMsg->wParam);
```

```
        m_ScrollVert.ProcMOUSEWHEEL(delta);
        m_ScrollImgCtl.Show(m_ScrollVert.GetCurrentScrol());
    }
    return CDialogEx::PreTranslateMessage(pMsg);
}
```

4.5.20. 列表控件（List Control）

相关控件：CListCtrl 控件类

CListView 视图类，列表为视图

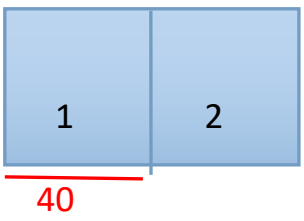
CLiseView=CView+CListCtrl

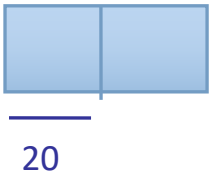
列表控件属性 View 中可以选图标、小图标、列表、报告。选择报告可以显示更多信息。

4.5.20.1. 图标设置

使用于 ICON，SMALL ICON 和 LIST 风格

加入两个位图资源：





IDB_NORMAL IDB_SMALL

类内定义：

CImageList　m_normal;

CImageList　m_small;

初始化：

m_normal.Create(IDB_BIG,40,1,RGB(255,255,0));

m_small.Create(IDB_SMALL,20,1,RGB(255,255,0));

40 代表一次移动的大小，1 代表图片使用时移动一格

RGB 代表不显示的颜色（需要去掉的颜色）

m_listCtl.SetImageList(&m_normal,LVSIL_NORMAL);

m_listCtl.SetImageList(&m_small,LVSIL_SMALL);

4.5.20.2. 修改样式

m_listCtl.ModifyStyle(LVS_REPORT,　LVS_SMALLICON);

　　//第一个参数代表需要移除的风格，第二个代表要加入的风格

4.5.20.3. 插入数据

ICON 与 SMALL ICON 风格

只有行的概念，没有列的概念，每个图像代表一行(排列的时候一行会排几个)

LIST 风格

只有行的概念，像图标一样，有图和对应文字，相比 List Box，只是多了一个图标

m_listCtl.InsertItem(0,"file1",1);

//加入图标（只要每次调用这个函数使用不同的数字[最好按顺序写]，就按照函数排列的位置进行顺序插入）。标题为”file1”，使用位图中第二个图标

m_listCtl.SetItemText(1,0,"x");

//修改图标对应的文字

//没有列概念，第二个参数总是 0,1 指代第二个图像，与图像排列位置相关

4.5.20.4. Report 风格

```
m_listOptKey.ModifyStyle(LVS_ICON, LVS_REPORT);
m_listOptKey.SetExtendedStyle(LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);
m_listOptKey.InsertColumn(0, _T("功能"), LVCFMT_LEFT, 120);
m_listOptKey.InsertColumn(1, _T("快捷键"), LVCFMT_LEFT, 100);
m_listOptKey.SetTextBkColor(RGB(10,10,10));
m_listOptKey.SetBkColor(RGB(100,100,100));
for (int i = 0; i < itemCount; i++)
{
    int icount = m_listOptKey.GetItemCount();
    m_listOptKey.InsertItem(icount, val_key[i][0]);
    m_listOptKey.SetItemText(icount, 1, val_key[i][1]);
}
```

4.5.20.4.1. 设置风格

```
ModifyStyle(LVS_ICON,  LVS_REPORT);
```

4.5.20.4.2. 插入字段

有行和列的概念，先设计好每列的条目，然后再加入行

```
m_CtrList.SetExtendedStyle(LVS_EX_FULLROWSELECT |
 LVS_EX_GRIDLINES); //设置为可以行选
```

添加最上层的条目说明

```
m_CtrList.InsertColumn(0, _T("序号"), LVCFMT_CENTER, 80);
m_CtrList.InsertColumn(1, _T("姓名"), LVCFMT_CENTER, 100);
m_CtrList.InsertColumn(1, _T("年龄"), LVCFMT_CENTER, 100);
```

第一个参数代表索引号，最后一个代表字段宽度。

4.5.20.4.3. 修改字段

```
//设置第一行第二列的字段
int    row = 0;
int    col = 1;

SetItemText(1,2,_T("xx"));
```

或者：

```
LVCOLUMN col;
m_CtrList.GetColumn(row ,&col);
col.pszText=_T("XX");
m_CtrList.SetColumn(col ,&col);
```

4.5.20.4.4. 插入数据

```
    int icount=m_CtrList.GetItemCount(); //得到当前条目数，作为新的
```

数据的插入位置

```
m_CtrList.InsertItem(icount,strNumber); //插入数据
m_CtrList.SetItemText(icount,1,strName); //和插入数据同一行的其他数据
m_CtrList.SetItemText(icount,2,strAge);
```

4.5.20.4.5. 显示最后一行

```
m_CtrList.EnsureVisible(m_CtrList.GetItemCount() -1, FALSE);
```

4.5.20.4.6. 选择控制

```
m_CListCtrl.Create(WS_CHILD | WS_VISIBLE|LVS_SHOWSELALWAYS, rectDummy, pParent, 1); //LVS_SHOWSELALWAYS, 只在创建时有效

m_list.SetItemState(-1,0,LVIS_SELECTED); //取消所有选择状态
m_CListCtrl.SetItemState(curIndex,LVNI_SELECTED|LVNI_FOCUSED,LVNI_SELECTED|LVNI_FOCUSED);
m_CListCtrl.EnsureVisible(curIndex,FALSE);
```

4.5.20.4.7. 设置显示选择行

```
SetExtendedStyle(LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES|LVS_SHOWSELALWAYS);
```

4.5.20.4.8. 获取选择行

```
int row = m_bzListCtl.GetSelectionMark();
```

4.5.20.4.9. 得到选中的多个位置

```
CListCtrl* pListCtrl = (CListCtrl*)GetDlgItem(IDC_LIST1);
POSITION pos = pListCtrl->GetFirstSelectedItemPosition();
if (pos != NULL)
```

```
{   while(pos){
      int nItem = pListCtrl->GetNextSelectedItem(pos);
      //利用 nitem 对选中数据进行操作，这里 pos 是引用
      CString str=pListCtrl->GetItemText(0,1);
   }
}
```

可以添加列表控件的 LBN_DBLCLK 消息处理，实现在双击某项获取数据并操作(数据为列表中的项目）

4.5.20.4.10. 右击消息取数据

```
NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
if(pNMListView->iItem != -1)
{
      DWORD dwPos = GetMessagePos();
      CPoint point( LOWORD(dwPos), HIWORD(dwPos) );
      m_CtrList.ScreenToClient(&point);
      LVHITTESTINFO lvinfo;
      lvinfo.pt = point;
      lvinfo.flags = LVHT_ABOVE;
      int nItem = m_CtrList.SubItemHitTest(&lvinfo);
      if(nItem != -1)
      {    //单击的是第 lvinfo.iItem 行第 lvinfo.iSubItem 列
        //取出数据
          CString str=m_CtrList.GetItemText(lvinfo.iItem,lvinfo.iSubItem);
      }
}
```

4.5.20.4.11. 清空数据

pListCtrl->DeleteAllItems();

4.5.20.4.12. 属性修改

```
LONG lStyle;
lStyle = GetWindowLong(m_CtrList.m_hWnd, GWL_STYLE);//获取当前窗口 style
lStyle &= ~LVS_TYPEMASK; //清除显示方式位
lStyle |= LVS_REPORT; //设置 style
SetWindowLong(m_CtrList.m_hWnd, GWL_STYLE, lStyle);//设置 style
```

DWORD dwStyle = m_CtrList.GetExtendedStyle();
dwStyle |= LVS_EX_FULLROWSELECT;//选中某行使整行高亮（只适用与 report 风格的 listctrl）
dwStyle |= LVS_EX_GRIDLINES;//网格线（只适用与 report 风格的 listctrl）
m_CtrList.SetExtendedStyle(dwStyle); //设置扩展风格

4.5.20.4.13. 设置列宽

m_CtrList.SetColumnWidth(列编号,列宽度);

4.5.20.4.14. 设置行不可选

```
void **::OnLvnItemchanging(NMHDR *pNMHDR, LRESULT *pResult)
{
    LPNMLISTVIEW pNMLV = reinterpret_cast<LPNMLISTVIEW>(pNMHDR);
    if ((pNMLV->uChanged & LVIF_STATE) && (pNMLV->uNewState & LVNI_SELECTED))  *pResult =
1;
    else *pResult = 0;
}
```

4.5.21. Tree

4.5.21.1. 操作说明

风格设置为 has Buttons 和 Lines at root

HTREEITEM item; //存储插入数据后返回的节点句柄
item=m_tree.InsertItem(“节点”); //只有一个数据代表插入节点数据
item=m_tree.InsertItem(“数据”,item); //代表在指代的节点下插入数据

m_tree.SetItemData(item,1234);//加入附加数据，需要数字，可以用(DWORD)pName 加入一个指向堆对象的 CString

m_tree.DeleteAllItems();//删除所有节点

m_tree.SelectItem(item);

HTREEITEM hChildItem = m_tree.GetChildItem(item); 树控件如果没有底层目录了，值就为空

4.5.21.2. 取出附加数据

为树控件增加 NM_CLICK 消息处理

```
HTREEITEM hSelectItemOld = m_tree.GetSelectedItem();
CPoint point;
GetCursorPos(&point);
m_tree.ScreenToClient(&point);

UINT flag = TVHT_ONITEM ;
HTREEITEM hItem = m_tree.HitTest(point);
if(hItem != NULL && hSelectItemOld != hItem)
m_tree.SelectItem(hItem);
HTREEITEM childTree= m_tree.GetChildItem(hItem);

if (!childTree)
{
    //附加数据是字符串时
     CString *str=(CString*)m_tree.GetItemData(hItem);
     //附加数据是数字时
    int  i=(int)m_tree.GetItemData(hItem);

}
```

4.5.21.3. 循环插入实例

4.5.21.3.1. 假设数据

假设有如下向量，每个数据带有两个 CString 的结构，且内部数据是按照根相同的数据连续放置。

```
#include <vector>
using namespace std;
typedef struct Data
{
      CString type;
      CString name;
}Data;
typedef vector<Data>  VectorData;


VectorData  v;
```

```
Data d1;
d1.type="蔬菜";
d1.name="白菜";
v.push_back(d1);

Data d2;
d2.type="蔬菜";
d2.name="土豆";
v.push_back(d2);

Data d3;
d3.type="水果";
d3.name="葡萄";
v.push_back(d3);

Data d4;
d4.type="零食";
d4.name="方便面";
v.push_back(d4);
```

4.5.21.3.2. 插入树

```
    HTREEITEM    root; //指代根节点，用来判断是否需要插入新的根
    CString rootRepeate=_T(""); //用来存储每次根节点数据，用来判断数据是否和刚才插入的根相同
    for(int i=0;i<v.size();i++)    //循环加入数据
    {
        //取出数据
        Data data=v[i];
        CString name=data.name;
        CString type=data.type;
        /////////为树插入节点和数据
        if (type!=rootRepeate)    //type 是每次取出的代表根的数据，如果和上次节点不同，就插入新的节点，如果相同，就在上次节点后继续插入（需要取出的数据按规律排）
        {
            root=m_tree.InsertItem(type); //如果插入新根节点，就将 root 指向新的节点
        }
        HTREEITEM leaf=m_tree.InsertItem(name,root); //在根节点下插入数据

        CString *pName=new CString;
        *pName=name;
```

```
        m_tree.SetItemData(leaf,(DWORD)pName);//附加数据，如果是数字，可以直接写

        rootRepeate=type;
    }
```

4.5.22. TABcontrol

4.5.22.1. 设置

控件需要的页面的对话框如：IDD_DIALOG1，IDD_DIALOG2

设置属性：Border none ， Style Child

建立对话框类

dlg1，dlg2

主对话框使用时注意包含页面对话框的头文件

绑定变量

CTabCtrl m_Table;

类内变量

dlg1 page1;

dlg2 page2;

4.5.22.2. 初始化

4.5.22.2.1. 设置页面文字

```
m_Table.InsertItem(0,_T("页面1"));
m_Table.InsertItem(1,_T("页面2"));
```

4.5.22.2.2. 设置控件属性（非必须）

m_Table.MoveWindow(30,30,200,200);

m_Table.SetMinTabWidth(30); // 设置标签项的最小宽度

m_Table.SetPadding(CSize(30,8)); // 设置标签项和图标周围的间隔

4.5.22.2.3. 创建页面

page1.Create(IDD_DIALOG1,&m_Table);

page2.Create(IDD_DIALOG2,&m_Table);

4.5.22.2.4. 页面显示位置

将其覆盖在Tab控件的相应位置上

只能在创建页面后使用

```
CRect rect;
m_Table.GetClientRect(&rect);
rect.top+=20;
rect.bottom-=5;
rect.left+=5;
rect.right-=5;
page1.MoveWindow(&rect);
page2.MoveWindow(&rect);
```

4.5.22.2.5. 设置选择

page1.ShowWindow(SW_SHOW); //初始化选择

m_Table.SetCurSel(1);

4.5.22.3. 选择消息处理

为Tab控件的TCN_SELCHANGING消息添加ON_NOTIFY的相应消息处理

在函数中为不同页签的选择添加相应显示

```
int CurSel=m_Table.GetCurSel();
switch(CurSel)
{
case 0:
    page1.ShowWindow(SW_SHOW);
    page2.ShowWindow(SW_HIDE);
    break;
case 1:
    page1.ShowWindow(SW_HIDE);
    page2.ShowWindow(SW_SHOW);
    break;
default: ;
}
```

4.5.23. CPropertySheet

4.5.23.1. 数据交互

可以在 sheet 中声明一些变量，在 page 中对这些变量进行赋值，在 sheet 的 DoModal 之后用 sheet 的对象取出这些数据。

4.5.23.2. 设置

class CMyPropertySheet : public CPropertySheet

//属性表单，用来联系属性页

建立两个对话框(可以选择IDD_PROPPAGE_LARGE这种类型的对话框资源[注意资源语言的选择]，而不使用默认的)

为两个对话框建立对话框类，注意继承自 CPropertyPage

class CPage1 : public CPropertyPage

class CPage2 : public CPropertyPage

4.5.23.3. 初始化

注意包含头文件

CMyPropertySheet sheet(_T("xx"),this);

CPage1 p1;

CPage2 p2;

sheet.AddPage(&p1);

sheet.AddPage(&p2); //按顺序加入，代表向导中顺序

sheet.SetWizardMode();//设置为向导模式

//如果不设置向导模式，可以和 TABcontrol 有相同显示效果

4.5.23.4. 显示

sheet.SetActivePage(&p2); //设置要激活显示的页面

sheet.DoModal();

///如果是 Wizard 模式，DoModal 的返回值为 ID_WIZFINISH 或者 IDCANCEL.

4.5.23.5. 修改属性

4.5.23.5.1. 修改整体属性

需要为 CMyPropertySheet 添加 OnInitDialog 等函数

//隐藏掉帮助按钮

GetDlgItem (IDHELP)->ShowWindow (FALSE);

//隐藏掉取消按钮

GetDlgItem(IDCANCEL)->ShowWindow(FALSE);

4.5.23.5.2. 修改单页面属性

每个页面包含的按钮以及按钮的可用性不同，需要在 CPage1 这种页面的类中（注意包含头文件）实现虚函数 OnSetActive。

第一个页面中：

((CMyPropertySheet*)GetParent())->SetWizardButtons(PSWIZB_NEXT);

最后一个页面中：

((CMyPropertySheet*)GetParent())->SetWizardButtons(PSWIZB_FINISH|PSWIZB_BACK);

中间步骤页面中：

((CMyPropertySheet*)GetParent())->SetWizardButtons(PSWIZB_BACK|PSWIZB_NEXT);

4.5.23.5.3. 阻止进入下一页

为需要的 page 页增加虚函数 OnWizardNext

if(***){ *** return -1;}

4.5.24. 文件打开/另存对话框

CFileDialog(打开或另存，文件扩展名，默认文件名，窗口风格，文件类型的过滤，父窗口)

第一个参数 TRUE 为打开对话框，FALSE 为另存对话框

文件类型过滤字符串的格式：

1）如果是多行，每行以 | 分隔，整个字符串以 || 结束

2）每行有两部分（显示和后缀）组成，这两部分以 | 分隔

```
CString  fliter = _T("图像(*.jpg)|*.jpg|所有文件(*.*)|*.*||");
CFileDialog  dlg(TRUE, NULL, NULL, OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, fliter);
if (IDCANCEL == dlg.DoModal()) return;
dlg.GetPathName();
dlg.GetFileName();
```

4.5.25. 颜色对话框

CColorDialog dlg; //可使用 dlg(colorref)，构造初始选择颜色

if(IDCANCEL==dlg.DoModal()) return;

(COLORREF) m_color=dlg.GetColor();

可通过变量 m_color 接收选择的颜色信息

颜色对话框的成员变量 m_cc 是一个 CHOOSECOLOR 结构体，可以通过修改该结构体而改变颜色对话框的属性。

dlg.m_cc.Flags|=CC_FULLOPEN;//可以将颜色对话框自定义颜色展开

dlg.m_cc.Flags|=CC_RGBINIT;

dlg.m_cc.rgbResult=RGB(0,255,0);//可以设置初始选择颜色（必须设置 CC_RGBINIT 属性才有效）

4.5.26. 字体对话框

CFontDialog dlg;

CFont font; //用来接收选择的字体

```
if(IDOK!=dlg.DoModal()) return;

CString fontName = dlg.m_lf.lfFaceName;
int fontSize = dlg.m_cf.iPointSize/10;
```

4.5.27. 键盘消息响应

对话框不直接相应键盘消息，要在对话框类的 PreTranslateMessage 中加入：

```
if(pMsg->message==WM_KEYDOWN){
    UINT iKey=(UINT)pMsg->wParam;
    if(iKey==13){//按了回车}
}
```

4.5.28. 模式和非模式对话框

模式对话框在弹出时会阻塞，也就是在对话框不关闭时，其他窗口无法通过点击激活

CDialog　父类 CWnd

CCommonDialog　　通用对话框类，包括字体对话框，颜色对话框，文件打开、保存对话框，打印对话框，打印设置对话框

4.5.28.1. 对话框资源使用

对话框资源中，双击对话框资源，可以创建对话框类（如类名 CDlg）

可以在菜单项处理函数中：（需包含 CDlg 的头文件 Dlg.h）

1）创建模式对话框

```
CDlg    dlg;
dlg.DoModal();
```

2）创建非模式对话框

```
CDlg    *dlg=new CDlg;
dlg->Create(IDD_DIALOG1);
dlg->ShowWindow(SW_SHOW);
dlg->UpdateWindow();
```

为对话框类添加 WM_CLOSE 的处理函数，用以非模式对话框的销毁

```
CDialog::OnClose();
DestroyWindow();
```

必须写在下面一行

右击对话框类，添加虚函数 PostNcDestroy：

```
CDialog::PostNcDestroy();
delete this;
```

必须要写在下面这一行

或者可以使用（同一对话框只能创建一个）：

在类内定义成员变量：CDlg   dlg;

在某个函数内创建非模式对话框：

```
dlg.DestroyWindow();    //先销毁对话框类，然后再 Create
dlg.Create(IDD_DIALOG1);
dlg.ShowWindow(SW_SHOW);
```

dlg.UpdateWindow();

4.5.28.2. 模式对话框

1）模式对话框的创建显示：

CDialog::DoModal();

2）对话框的初始化函数，在对话框显示前调用

CDialog::OnInitDialog()

3)对话框的关闭

CDialog::OnOK/OnCancel()

EXP:

CDialog dlg(IDD_DIALOG1);

第二个参数是 CWnd*默认是 NULL，可以写成 this

dlg.DoModal();

4.5.28.3. 非模式对话框

非模式对话框与一般窗口类似

非模式对话框需要用户自己处理关闭进程(因为要释放 new 出的对象)

1）需要重写 CDiglog::OnOK() 和 OnCancel(), 在函数中调用 DestroyWindow()函数销毁窗口（OK 为大写）

2）重写 PostNcDestroy()，在函数中 delete this；这个函数在 1）中的函数执行时被调用

exp:

void CMyDlg::OnOK(){

CDialog::OnCancle();

DestroyWindow();}

void CMyDlg::PostNcDestroy(){

CDialog::PostNcDestroy();

delete this;}

CMyDlg *pdlg=new CMyDlg(IDD_DIALOG1,m_pMainWnd);

m_pMainWnd 与 pdlg 是等价的

m_pMainWnd->ShowWindow(SW_SHOW);

m_pMainWnd->UpdateWindow();

4.5.29. DDX

不可用于线程中

将对话框的控件与对话框类的成员变量绑定，可以通过访问成员变量方式去操作对话框控件

virtual void DoDataExchange(CDataExchange *pDX)

包含一系列的绑定函数（使用参数 pDX），绑定过程可自动生成

绑定函数：以 DDX 开头

DDX_Text(.....) 值类型的绑定

DDX_Control(.....)　控件类型的绑定

UpdateData(BOOL)　在控件与成员变量进行数据交换时调用

不能在线程中用这个函数，线程中只能用 SendMessage 方式让主线程刷新。

TRUE　将用户在控件中输入的值传递给变量，得到数据（默认，一般可不写）

FALSE　将变量的值显示到控件上（编辑框用）

控件绑定的变量不能写在对话框类的初始化函数或者 WM_CREATE 里面，只能写在 InitDialog 函数或者之后的执行函数里面。

对话框类内：

控件类型的绑定

1）类内定义控件类

CButton　m_btnOK;　//或者写成 CWnd　m_btnOK;

2）在 DoDataExchange 函数中绑定

DDX_Control(pDX,IDOK,m_btnOK);

DoDataExchange 中传入　与 ok 按钮绑定　链接到类内成员

3）使用绑定的控件进行操作（比如在 InitDialog 函数中）

if(!CDialog::OnInitDialog())

return FALSE;

m_btnOK.SetWindowText(“xxx”);

m_btnOK.MoveWindow(0,0,100,100);通过变量修改控件

return TRUE;

值类型绑定：

1）类内定义变量 CString str; 也可以绑定 int 类型

2）DoDataExchange 函数中：

DDX_Text(pDX,IDC_EDIT1,str);

与编辑框绑定

【在 InitDialog 函数中可用：

str="xxxx";

UpdateData(FALSE); 显示到控件

或者运行时改变编辑框的内容，会自动改变 str 的值，使用前注意先更新数据】

3）类内重写 virtual void OnOK();

重写 ok 按钮的处理

UpdateData(TRUE);

可将编辑框输入的内容取出到 str 中。TRUE 为默认值，可以缺省

AfxMessageBox(str);

CDialog::OnOK();

4.6. MFC 线程

CWinThread* pThread=AfxBeginThread(ThreadProc,NULL);

UINT ThreadProc(LPVOID pParam){return 0;}/

::WaitForSingleObject(pThread->m_hThread, INFINITE);可用于析构函数中等待线程结束。

//启动线程，初始为挂起状态

```
pThread = AfxBeginThread(ThreadProc, &m_ThreadParam, THREAD_PRIORITY_ABOVE_NORMAL, 0, CREATE_SUSPENDED);
//线程结束时不自动撤销
pThread->m_bAutoDelete = FALSE;
//恢复线程运行
pThread->ResumeThread();
```

结束线程并不能析构线程中分配的堆对象。

```
if(pThread!=NULL){
        TerminateThread(pThread->m_hThread,0);
        pThread=NULL;
}
```

==========CEvent

对 API 函数进行了封装

```
CEvent event(FALSE,TRUE);

WaitForSingleObject(event.m_hObject, INFINITE);

event.ResetEvent();


event.SetEvent();
```

4.7. SDI

4.7.1. 绘图消息

ON_WM_PAINT()

```
void CMyFrameWnd::OnPaint(){
        CPaintDC dc(this);
        dc.TextOut(34,234,"ssf");
}
```

4.7.2. 鼠标消息

ON_WM_MOUSEMOVE()

```
void CMyFrameWnd::OnMouseMove(UINT nKey,CPoint pt){
.....................
    InvalidateRect(NULL,TRUE);
}
```

4.7.3. 键盘消息

ON_WM_CHAR

```
afx_msg void OnChar( UINT nChar, UINT nRepCnt, UINT nFlags ){
            if(97==nChar){
                    AfxMessageBox("a");}
}
```

4.7.4. 最近打开文档

4.7.4.1. 添加

theApp.AddToRecentFileList(filePathName);

4.7.4.2. 使用/删除

获取点击的最近文档，返回 NULL 删除该项目

```
CDocument* C***App::OpenDocumentFile(LPCTSTR lpszFileName)
{
    return NULL;
    return CWinAppEx::OpenDocumentFile(lpszFileName);
}
```

4.7.5. 视图类滚动条

将基类改为 CScrollView（修改其他消息对应代码基类）

OnInitialUpdate 中设置滚动条

CSize size(0,0);

SetScrollSizes(MM_TEXT,size);

该代码可以在其他位置调用进行设置

4.7.6. 菜单

1.相关问题

Win32 HMENU

MFC　　CMenu 类对象

2.相关类

CMenu　封装了 m_hMenu 和 API

4.7.6.1. 添加菜单资源

File，new，Resource Script，进入资源选项卡，右击资源文件夹，insert，可选择加入各种类型的资源。

添加菜单项目时，如使用 f&ile 则 i 就变成热键，即使用 alt+i 可以选中该项目。

文件中需包含头文件

#include “resource.h”（不要写在预编译头文件 StdAfx.h 中，否则会无法修改菜单 id）

4.7.6.2. 设置菜单

1.在 CFrameWnd：：Create 参数中设置菜单

pFrame->Create(NULL,”name”,WS_OVERLAPPEDWINDOW,CFrameWnd::rectDefault,NULL,MAKEINTRESOURCE(IDR_MENU1));

后面有缺省参数，可以不写，CFrameWnd：：rectDefault 代表四个参数。

2.在框架窗口的 ON_WM_CREATE()消息处理中：

```
CMenu　menu;
menu.LoadMenu(IDR_MENU1);
```

SetMenu(&menu);

menu.Detach();如果 menu 是局部变量，用这一句可以将句柄与 menu 对象断开，防止发生问题

4.7.6.3. 处理菜单项目

ON_COMMAND(ID,OnNew)

参数为菜单的 id 及自定义的处理函数

afx_msg void OnNew();

4.7.6.4. 设置菜单状态

在框架类的构造函数中改变变量，取消自动检测更新菜单可用性，这样才可以自由设置菜单项目的可用性

this->m_bAutoMenuEnable=FALSE;

ON_WM_INITMENUPOPUP()

菜单被激活，即将显示时（下拉出菜单的项目时)

afx_msg void OnInitMenuPopup(CMenu *pop,UINT

n,BOOL i);

pop->EnableMenuItem(id,

MF_DISABLED|MF_GRAYED);

pop->CheckMenuItem（id，MF_CHECKED）

或者使用;

::CheckMenuItem(pop->m_hMenu,id,MF_CHECKED);

MF_BYCOMMAND 代表第一个参数使用 ID，如果使用另一个参数 MF_BYPOSITION 则第一个参数使用从 0 开始的相对位置

pop->SetDefaultItem(ID);使菜单项目变粗体，但是只能设置一个菜单项目

4.7.6.4.1. 设置图像菜单

CBitmap bmp1,bmp2;　在 Frame 类内定义

bmp1.LoadBitmap(IDB_BITMAP1);

bmp2.LoadBitmap(IDB_BITMAP2);

CMenu *pop=GetMenu();

pop->GetSubMenu(0)->SetMenuItemBitmaps(1,MF_BYPOSITION, &bmp1,&bmp2);

第一个 bmp 代表 MF_UNCHECKED 时显示的位图，第二个代表 MF_CHECKED 时的显示位图

GetSystemMetrics(SM_CXMENUCHECK)可以获得标记菜单位图应该设置的大小

（注意菜单挂在 Frame 类中，则只能在 Frame 中的函数里用 GetMenu 获取菜单的指针）

SetMenu(NULL); 可以去掉菜单

4.7.6.5. 简易方式设置菜单项目状态

在类向导中，可以为菜单项目 id 添加 COMMAND 消息处理，还有一个就是 UPDATE_COMMAND_UI

这个消息发生在程序发生变化的时候，也就是有程序自动调用，实时根据情况刷新菜单项目

可以为类增加一个变量 x，用 x 的值进行判断，来决定怎么改变菜单项目：

```
if(x>0){pCmdUI->Enable();//就可以设置是否可用}
else{pCmdUI->Enable(FALSE);}
```

pCmdUI->m_nID 保存了当前菜单项目对应的 ID

4.7.6.6. 右键菜单

4.7.6.6.1. 使用方式

ON_WM_CONTEXTMENU()

(使用屏幕坐标系，不需要转换,ClientToScreen 可以把客户区坐标转换成屏幕坐标)

```
afx_msg void OnContextMenu(CWnd *pWnd,CPoint pt);
CMenu menu;
menu.LoadMenu(IDR_MENU1);
CMenu      *pPopup=menu.GetSubMenu(0);
```

菜单资源，取第一个顶层菜单项目下的子菜单

pPopup->TrackPopupMenu(TPM_LEFTALIGN,pt.x,pt.y,this);

this 也用右键菜单处理函数的第一个参数 pWnd

或者选择 project 里的 add to project 里的 components and controls 在里面 c++文件夹直接选择 popupmenu 加入

4.7.6.6.2. 常用方式

```
CMenu menu;
menu.CreatePopupMenu();
menu.AppendMenu(MF_STRING,IDM_MENU1,"名称");
        //需要为 IDM_MENU1 定义宏以及命令消息处理
CPoint pt;
GetCursorPos(&pt);//pt 为当前鼠标坐标点
menu.TrackPopupMenu(TPM_LEFTALIGN,pt.x,pt.y,this);
menu.DestroyMenu();
```

4.7.6.7. 动态添加菜单

```
CMenu menu;
menu.CreatePopupMenu(); //创建一个顶层菜单
GetMenu()->InsertMenu(1,MF_BYPOSITION|MF_POPUP,(UINT)menu.m_
hMenu,"顶层菜单");  //在菜单第 2 个位置插入一个顶层菜单
```

GetMenu()->AppendMenu(MF_POPUP,(UINT)menu.m_hMenu," 顶层菜单"); //为菜单栏添加一个顶层菜单

menu.AppendMenu(MF_STRING,100,"菜单项 2"); //添加一个菜单项目，100 为项目 id

menu.InsertMenu(0,MF_STRING|MF_BYPOSITION,101,"菜单项 1"); //在指定位置插入一个菜单项目

menu.Detach();//menu 为局部变量时使用

GetParent()->DrawMenuBar();在 view 类中动态添加菜单后，需要这句重绘菜单栏

DeleteMenu(0,MF_BYPOSITION）

可以删除菜单顶层菜单或者菜单项目

4.7.7. 工具栏

1.相关类

CToolBarCtrl

父类 CWnd，封装了对工具栏控件的操作

CToolBar

父类 CControlBar，封装了工具栏和框架窗口之间关系（停靠位置），另外还包括工具栏的创建

2.使用

1）添加资源

2）创建工具栏　CToolBar：：Create/CreateEx

3) 加载工具栏资源　　CToolBar：：LoadToolBar

工具栏资源创建时，将某项的 id 设置为菜单某项目的 id，可与其进行绑定，也可以不绑定菜单。和菜单的项目处理方式相同，删除某项时，用鼠标将小图直接拖出即可。

<afxext.h>

CToolBar　toolbar；在 CMyFrame 类内定义一个对象

ON_WM_CREATE()

afx_msg int OnCreate(LPCREATESTRUCT pcs);

toolbar.CreateEx(this,TBSTYLE_FLAT,

WS_CHILD|WS_VISIBLE|CBRS_ALIGN_TOP|CBRS_GRIPPER|CBRS_TOOLTIPS|CBRS_FLYBY|CBRS_SIZE_DYNAMIC);

this 之后的有缺省值，可以不写后面的

CBRS_GRIPPER 可以使工具栏有把手，可移动

CBRS_TOOLTIPS 使鼠标划过时显示提示信息

CBRS_FLYBY　　使鼠标划过时在状态栏显示相应提示信息

CBRS_SIZE_DYNAMIC　使工具栏移动后大小可变

CBRS_ALIGN_TOP 的作用：

1）后面可以不使用 DockControlBar 函数，但使用把手移动的风格失效

2）使用 DockControlBar 时第二个参数可使用默认值 0；

toolbar.LoadToolBar(IDR_TOOLBAR1)；

加载资源后即可显示

工具栏的停靠（船坞化）

1）工具栏准备停靠的位置

CToolBar：：EnableDocking

2)框架窗口允许停靠的位置

CFrameWnd：：EnableDocking

3）框架窗口确定停靠的位置

CFrameWnd：：DockControlBar

1），2）必须有交集

toolbar.EnableDocking(CBRS_ALIGN_ANY);

EnableDocking(CBRS_ALIGN_ANY);

DockControlBar(&toolbar)；设置开始时停靠的位置（第二个参数，可省略）写完此三句后才可拖拽工具栏。

AFX_IDW_DOCKBAR_TOP

AFX_IDW_DOCKBAR_BOTTOM

AFX_IDW_DOCKBAR_LEFT

AFX_IDW_DOCKBAR_RIGHT

toolbar.SetWindowText(“xx”);

在工具栏资源中，选中资源，按回车键可进入属性，在底下的提示内

可写提示信息(需要加入风格 CBRS_TOOLTIPS)

提示信息格式：

aaa\nbbb

aaa 会在状态栏显示

bbb 会在鼠标滑过工具栏项目时提示

工具栏的显示和隐藏

CFrameWnd：：ShowControlBar（&toolbar，TRUE，FALSE）

第二个参数为是否显示

第三个参数为是否延迟，一般用 FALSE

toolbar.IsWindowVisible()

判断是否处于显示状态

Mousemove 中如果有 invalidateRect 操作，会影响工具栏的显示

4.7.8. 状态栏

小格为指示器

CStatusBar　封装了关于状态栏的操作，以及包括创建状态栏

<afxext.h>

4.7.8.1. 创建

CStatusBar statusbar;　　CMyFrameWnd 类内定义

Create 消息处理中创建：

statusbar.CreateEx(this);

4.7.8.2. 设置指示器

CStatusBar：：SetIndicators

1)添加字符串资源（如 IDS_POS,IDS_TIME）

多文档中，指示器不能直接用系统定义好的字符串资源，将其删除，自己定义字符串资源

2)全局数组

UINT g_hIndicators[]={0,IDS_POS,***};

数组内第一个元素为 0，显示的是工具栏提示信息 aaa/nbbb 中的 aaa（会自动加入字符串资源）（需为工具栏加入风格 CBRS_FLYBY），第二个和第三个参数显示的是字符串资源中的内容。

3）statusbar.SetIndicators(g_hIndicators,

sizeof(g_hIndicators)/sizeof(UINT));

4.7.8.2.1. 设置指示器宽度和风格

statusbar.SetPaneInfo(1,IDS_POS,SBPS_NORMAL,200);

1 代表数组内的第二个元素,第二个参数貌似没啥用。第三个参数代表设置的风格，第四个参数为指示器宽度。

4.7.8.2.2. 设置/修改指示器的文本内容：

statusbar.SetPaneText(1,”文字”);

1,代表第一个位置。0 位置用于最左边动态显示工具栏提示。右边开始可以从 1 开始数指示器位置。

可用于动态设置指示器内容，如 mousemove 消息中：

```
CString str;
str.Format(“%d,%d”,pt.x,pt.y);
statusbar.SetPaneText(1,str);
```

（鼠标移动消息的处理和状态栏类的声明同在类 CMyFrameWnd 中）

4.7.8.3. 设置状态栏高度

StatusBar.GetStatusBarCtrl().SetMinHeight(30) ;

4.7.8.4. 指示器位置

CRect rect;

m_wndStatusBar.GetItemRect(1,rect);

可以得到状态栏中第 2 格的指示器的位置，该位置是以状态栏为基准的位置，rect.top 一般都是 2。

该函数不能在 OnCreate 函数中调用，要等到状态栏创建完成后才能调用。

4.7.8.5. 对话框中添加指示器

需要添加代码：

CRect rect;

GetClientRect(&rect); 参数也可以写成 rect

statusbar.MoveWindow(0,rect.bottom-20,rect.right,20);

4.7.9. 加速键

资源：id 加速键

id 可对应菜单项目的 id

HACCEL ：：LoadAccelerators

BOOL ：：TranslateAccelerator

CWinApp 类的虚函数

virtual BOOL PreTranslateMessage（MSG *pMsg）;

HACCEL hAccel=::LoadAccelerators(

AfxGetInstanceHandle(),

MAKEINTRESOURCE(IDR_ACCELERATOR1));

::TranslateAccelerator(this->m_pMainWnd->m_hWnd,

hAccel,pMsg);

return CWinApp::PreTranslateMessage(pMsg);

第一个参数得到的是 CMyFrameWnd 类窗口的窗口句柄，因为菜单消息在 CMyFrameWnd 中处理

4.7.10. 光标

光标资源

ON_WM_SETCURSOR()

鼠标移动过程中并且在没有捕获鼠标消息情况下连续出现的消息

在设置光标资源的消息处理函数中处理(或者在 mousemove 中):

afx_msg void OnSetCursor(CWnd *pWnd,UINT nHitTest,UINT message)

```
if (NULL != m_hcursor)
{
    ::SetCursor(m_hcursor);
    return TRUE;
}
return CView::OnSetCursor(pWnd, nHitTest, message);
```

```
m_hcursor= LoadCursorFromFile(_T("skin\\curFile.cur"));
m_hcursor = LoadCursor( NULL , IDC_CROSS );
```

4.7.11. ribbon

4.7.11.1. 隐藏右键菜单和系统功能

```
class CMFCRibbonBarEx:public CMFCRibbonBar
{
public:
    void RemoveQAT(){ m_QAToolbar.RemoveAll();}
    virtual BOOL OnShowRibbonContextMenu(CWnd* pWnd, int x, int y, CMFCRibbonBaseElement*
pHit){return TRUE;}
};
```

4.7.11.2. 移除面板

```
m_wndRibbonBar.GetActiveCategory()->RemovePanel(7);
```

4.7.12. 自定义窗口

::CreateWindowEx(0,"button","ok",WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,400,100,100,40,this->m_hWnd,(HMENU)1001,AfxGetInstanceHandle(),NULL);

创建一个按钮(在有视图时，创建的按钮应放在视图窗口上,且应该写在 WM_CREAT 消息中，写在 Create 函数中无效)

或者使用：

CButton *btn=new CButton; 也可以为类内增加一个 btn 对象成员

btn->Create("ok",WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,CRect(0,0,100,40),this,1001);

ON_BN_CLICKED(1001,OnClicked)当按钮被点击时，可对应自定义

处理函数 OnClicked，其中 1001 为按钮 id，虽然参数为菜单形式，但不是菜单。

::CreateWindowEx(0,"EDIT","",WS_CHILD|WS_VISIBLE|WS_BORDER, 100,100,200,200,this->m_hWnd,(HMENU)1002,AfxGetInstanceHandle(), NULL);

创建一个编辑框

ON_EN_CHANGE(1002,OnEnChange)当编辑框内容发生变化时，可对应自定义处理函数 OnEnChange，其中 1002 为编辑框 id。

Edit 和 button 是系统窗口类名，可以不区分大小写

4.7.13. 设置窗口样式

在 MainFrame 类的虚函数 PreCreateWindow 中可以修改结构体 CREATESTRUCT& cs，达到改变窗口样式的目的。

该结构体就是 CreateWindowEx 函数中 12 个参数组成的结构。

cs.cx=100;

cs.cy=100;

cs.style | =WS_***; //增加窗口风格

cs.style &=~WS_***;//去掉窗口风格

对话框不响应该函数，对话框经过 OnInitDialog()

在 View 类 WM_CREATE 的 OnCreate 函数中，可以通过代码，在窗口创建后改变窗口样式。

SetClassLong(m_hWnd,GCL_HBRBACKGROUND,(LONG)GetStockObject(BLACK_BRUSH));//设置黑色背景

SetClassLong(m_hWnd,GCL_HBRBACKGROUND,(LONG)CreateSolidBrush(RGB(255,0,0))); //设置红色背景

第二个参数可以是 GCL_HICON 等，通过定时器动态调用该函数，可以做到窗口样式的动画效果。

4.7.14. 标题改变

显示格式：文档名 —窗口名

4.7.14.1. 文档名

每次打开文件时，文档名变化，是自动进行的

在文档类中 OnNewDocument 函数中加入：

SetTitle("文档");

这样可以在新建文件时显示新的文字，而不是未命名

4.7.14.2. 窗口名

资源中修改 String Table 中的 IDR_MAINFRAME 的值。第一个就是窗口名（多文档中此资源只有 1 项），修改即可。

4.7.15. 修改窗口风格

可以在框架类的 PreCreateWindow 函数里修改风格

cs.style&=~FWS_ADDTOTITLE; //去除标题条中的文档名

4.7.16. 获取其它窗口

4.7.16.1. 对话框句柄获取

HWND hwnd=::FindWindow(NULL,"对话框标题");

可以获得对话框的句柄

4.7.16.2. 获取父级指针

GetParent(); 在视图类获得框架类的指针

4.7.16.3. 获取应用程序实例

AfxGetInstanceHandle（）

可以拿到 winMain 的第一个参数 g_hInstance

4.7.16.4. 获取视图类指针

C**View *p=(C**View *)GetActiveView(); 在框架类中获得视图类的指

针

4.7.17. 框架类处理命令消息

在框架类中，可以重写虚函数 OnCommand(WPARAM wParam, LPARAM lParam),可以使用，用来截获命令消息，：

```
int id=LOWORD(wParam);
if(ID_NEW==id){
.....
return TRUE;
}
```

4.7.18. 显示/隐藏工具栏/状态栏

```
void CMainFrame::OnShowToolBar()
{
    if (m_wndToolBar.GetStyle() & WS_VISIBLE)
    {
        //隐藏工具栏
        ShowControlBar(&m_wndToolBar, FALSE, FALSE);
    }
    else
    {
        //显示工具栏
        ShowControlBar(&m_wndToolBar, TRUE, FALSE);
    }
}
void CMainFrame::OnUpdateShowToolBar(CCmdUI* pCmdUI)
{
    if (m_wndToolBar.GetStyle() & WS_VISIBLE)
    {
        pCmdUI->SetCheck(TRUE);
    }
    else
    {
```

```
        pCmdUI->SetCheck(FALSE);
    }
}
```

4.7.19. 屏蔽拖动标题栏功能

```
WM_NCLBUTTONDOWN

void CMainFrame::OnNcLButtonDown(UINT nHitTest, CPoint point)
{
    if (HTCAPTION == nHitTest) return;
    CFrameWndEx::OnNcLButtonDown(nHitTest, point);
}
```

4.7.20. 单文档只有第一次打开时最大化问题

```
int CRoadProcApp::ExitInstance()
{
        AfxOleTerm(FALSE);
    CleanState();
    return CWinAppEx::ExitInstance();
}


最大化设置
ParseCommandLine(cmdInfo);
m_nCmdShow=SW_SHOWMAXIMIZED;
```

4.7.21. 去除标题

```
App::InitInstance
::SetWindowText(AfxGetMainWnd()->GetSafeHwnd(),_T(""));


CMainFrame::PreCreateWindow
cs.style &= ~ FWS_ADDTOTITLE;
```

4.8. ActiveX

4.8.1. 控件的导入

如果系统没有注册控件，而程序中存在控件，会出现点击程序后无任何反应。

控件为 dll 或者 ocx，安装了某些软件后会自动生成一些控件。有些控件依赖于软件的存在，有些控件只需要一个 dll 或者 ocx 注册一下即可。

在 VC6.0 中可以通过“project”-> “Add To Project”-> “Components and Controls”导入 ActiveX 控件。

VS 系列中可以通过“project”-> “Add Class”->“MFC Class From ActiveX ”，可以在下拉框中找到控件名，然后增加到“Generated classes”中，点完成即可。

4.8.2. 显示处理

OnDraw 函数中可以将原来绘图的代码注释掉，写入自定义的显示代码。

调用 Invalidate();或者 InvalidateControl();可以让窗口重新刷新显示。

4.8.3. 消息处理

可以为 C**Ctrl 类加入消息响应，如 WM_CREATE 中初始化，WM_TIMER 处理定时器消息（注意在写 SetTimer(1,1000,NULL);这种函数时候，第一个参数其实已经包含，可能会出现代码提示帮助错误）

4.8.4. 与外界交互

4.8.4.1. 属性页

在 C**Ctrl 的 cpp 中有关于属性页的绑定函数：

```
BEGIN_PROPPAGEIDS(C***Ctrl, 1)
    PROPPAGEID(C***PropPage::guid)
END_PROPPAGEIDS(C***Ctrl)
```

如果要增加属性页中的页面，可以在这里加入绑定，加入后数量从原来的 1，变为 2。

如增加颜色属性页：

```
PROPPAGEID(CLSID_CColorPropPage)
```

ClassWizard 内 Automation 页面中，选择 Add Property，这个页面中的 External name 可以输入一个名称定义一个属性，也可以选择增加一些常规的属性。

4.8.4.1.1. 常规属性

Implementation 选择 Stock 表示选择的属性是控件的标准常规属性。添加后可以在控件的属性页面中看到这个属性并且可以设置。程序中可以通过函数获取这个设置的属性，然后对控件进行响应属性改变。

增加常规属性 BackColor，后可以在属性设置页面中设置这个值，在程序中通过：

```
COLORREF ref=TranslateColor(GetBackColor());
CBrush brush(ref);
pdc->FillRect(rcBounds,&brush);
```

可以获取颜色，然后使用设置的颜色设置控件背景色。

pdc->SetBkMode(TRANSPARENT)可以使上面的文字背景色透明。;

4.8.4.1.2. 自定义变量/属性

Implementation 选择 Member variable，表示增加一个变量，同时会生成相关函数（如果选择 Get/Set Methods 只能生成函数，还需要自己加入成员变量来保存该属性变量）

External name 是提供给外部程序调用时使用的变量名。如 x;

Type 可以自定义选择变量类型，如 short。

Variable name 是控件内部程序对于这个变量使用的名字。如 m_x;

Notification function 中对应的是函数名，该函数在外部程序改变这个变量时自动调用。在该变量设置后，外部使用这个控件会看到关于该属性的设置和获取的函数。

4.8.4.1.3. 属性值保存

在对话框中通过属性页设置了属性后，可以保存下来，下次打开工程时，查看控件的属性时可以是修改后的值。

在 CXXXCtrl 的 DoPropExchange 函数中：

```
PX_Short(pPX,"x",m_x,100);    //初始值为 100
```

4.8.4.1.4. 属性值通知

自定义了属性值，在改变后，需要让容器也知道属性值改变了，以便让容器的属性面板中的相应值进行改变。

在属性值改变的响应函数中加入代码：

```
BoundPropertyChanged(0x01);
```

数字代表在_D**中自动生成的变量 id 号码。如 [id(1)] short x;

4.8.4.1.5. 运行状态

使用 if(AmbientUserMode()){...}可以判断控件是在运行时还是在容器中处于设计状态。可以使有些代码的运行只在程序运行后执行。

4.8.4.1.6. 属性页对话框

可以在属性页的对话框资源中加入编辑框等控件，然后绑定变量（和

C***PropPage 绑定的），这时有一个 Option Property name，可以关联到常规属性或者自定义属性。这样在属性页中可以修改自定义或者常规的属性。

4.8.4.2. 与外界连通的函数

ClassWizard 内 Automation 页面中选择 Add Message。可以设置函数的名称，返回值以及参数列表（参数的类型有限制）。确认后会在_D***中生成相应外部函数接口代码。在 C**Ctrl 类中，则是函数的声明和实现。

4.8.4.3. 事件

可以为控件增加某些消息事件，会在_D***Events 中生成相应的代码。增加后，控件就可以响应该消息，并将该消息发送给容器。比如鼠标点击控件，则对话框可以收到控件被点击的消息。

可以增加标准事件，也可以增加自定义事件，自定义事件后，会生成一个函数（而且会为控件生成一个消息名称，在外部可以响应这个消息）。在控件中调用该函数时候，就可以向外部发出消息。

第五章 QT

http://download.qt.io/archive/qt/

5.1. function

5.1.1. 消息响应

A 类中有 SIGNAL 函数 sendSigFun()

B 类中有 SLOT 函数 RevSigFun()

将 A 对象的信号函数与 B 对象的槽函数绑定

```
QObject::connect(&Aobj,SIGNAL(sendSigFun()),
        &Bobj,SLOT(RevSigFun()));
```

消息函数的声明宏必须放在头文件中

函数绑定时一定不能写形参名，只能写类型

```
class AA :public QObject {
    Q_OBJECT   //信号槽类需要声明宏
signals:       //信号函数以 signals:指示字符开始
    void sendSigFun(const char*);


public:
    void Proc(const char* str) {
        emit sendSigFun(str);//信号函数调用前加 emit
    }
};


class BB :public QObject
{
    Q_OBJECT            //信号槽类需要声明宏
    private slots:      //private slots: 指示字符开始
    void  RevSigFun(const char* str) {
        qDebug()<<str;
    }
};



QObject::connect(&Aobj,SIGNAL(sendSigFun(const char*)),
                 &Bobj,SLOT(RevSigFun(const char*)));
```

5.1.2. 线程

```
class WorkerThread : public QThread
{
    void run() override {
    }
};
```

```
WorkerThread m_WorkerThread;
m_WorkerThread.start();
```

5.1.3. <QDebug>

qDebug()<<*****

5.1.4. <QString>

函数	功能
QString() QString(const char* str) QString(const QChar *unicode, int size = -1)	构造函数
int a = str.toInt();	字符串转数字
QString str=QString::number(12)	数字转字符串
str = QString::asprintf("%d,%.3f",12,12.123);	格式化字符串
int indexof(QString str); int indexof(char ch);	未找到返回-1

5.1.5. <QList>

QList<int> list;

MyChar ch=list.first();

foreach(int, list){}遍历，每次将 list 中的成员放入 ch

list.count()

5.1.6. <QTimer>

```
QTimer    *timer=new QTimer(&obj);
timer->setInterval(1000);
QObject::connect(timer,SIGNAL(timeout()),&obj,SLOT(RevSigFun()));
timer->start();
```

5.2. 动态库

```
int Q_DECL_EXPORT sum(int a,int b)
{
    return a +b;
}
```

修改为控制台工程：

```
CONFIG += c++11 console
#TEMPLATE = lib
```

5.3. 网络

```
#include <QtNetwork>
#include <QTextCodec>
```

5.3.1. http

```
QNetworkAccessManager m_manager;

connect(&m_manager,SIGNAL(finished(QNetworkReply *)),this,SLOT(rev(QNetworkReply *)));
m_manager.get(QNetworkRequest(QUrl("http:/****")));


public slots:
    void    rev(QNetworkReply *ry)
    {

    QTextCodec *codec = QTextCodec::codecForName("utf-8");//QTextCodec::codecForLocale()
```

```
    QString all = codec->toUnicode(ry->readAll());
    qDebug()<<all;
    ry->deleteLater();
  }
```

5.3.2. udp

```
#include <QtNetwork>
QT += network
```

5.3.2.1. 服务器

```
QUdpSocket m_reciver;
void ***::revData()
{
    while(m_reciver.hasPendingDatagrams())
    {
        QByteArray dat;
        dat.resize(m_reciver.pendingDatagramSize());

        m_reciver.readDatagram(dat.data(),dat.size());

        qDebug()<<dat.data();
    }
}


m_reciver.bind(1234,QUdpSocket::ShareAddress);
connect(&m_reciver,SIGNAL(readyRead()),this,SLOT(revData()));
```

5.3.2.2. 客户端

```
QUdpSocket m_client;
QByteArray dat = "aaa,bbb,ccc";
QHostAddress addr("127.0.0.1");
m_client.writeDatagram(dat,addr,1234);
```

5.4. widget

5.4.1. 结构

```
#include    <QApplication>
#include    <QWidget>
int main(int argc, char **argv){

QApplication        app(argc,argv);
QWidget          w;     //继承  QWidget 进行扩展
w.show();

return    app.exec();
}
```

```
sudo    apt -get    install qt-sdk
```

1)创建工程目录

2)在该目录下编写代码

3)qmake -project   (a.pro)

4)qmake   Makefile

5)make

5.4.2. qtdesigner 消息绑定

qtcreator 中可以使用 qtdesigner 在界面里右键控件，菜单中选择转到槽，生成对应槽函数

在界面对应类的构造函数中写入绑定

```
QtGuiApplication1::QtGuiApplication1(QWidget *parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);

    QObject::connect(ui.butObj, SIGNAL(clicked()), this, SLOT(Onclick()));
}
```

5.4.3. 鼠标

<QMouseEvent>

void mousePressEvent(QMouseEvent*){}

容器窗口的消息响应，用虚函数重写了父类实现的

QPoint　pos();　获取鼠标的坐标

QT：：MouseButton button(); 获取键值

QT：：keyboardModifiers　　modifiers(); 伴随键

exp:

```
void    mouseMoveEvent(QMouseEvent *ev){
ev->pos()
}
```

5.4.4. 键盘

<QKeyEvent>

if(QT::shiftModifier|QT::ControlModifier)

判断是否同时按下 shift 和 ctrl

void keyPressEvent(QKeyEvent *ev){}

int key();

QString modifiers();

QString text(); 键的名字

bool isAutoRepeat();按下后不放

注意键名的大小写

exp：

if(ev->key()==QT::Key_8)

判断 8 键是否按下

5.4.5. 控件

没有父窗口的窗口句柄为主窗口

窗口之间是对象间的关系

5.4.5.1. 按钮

<QPushButton>

QPushButton button;

button.setText("Button");

button.setParent(&w); w 为 QWidget 类的对象

button.setGeometry(x,y,cx,cy);

QPushButton(QWidget *parent=0)

QPushButton(QString text,QWidget *parent=0)

QPushButton *button=new QPushButton(“Button”,&w)

父窗口接管内存管理，不用通过 button 释放内存

5.4.5.2. <QLabel>

QLabel(QString str, QWidget *parent=0)

void setText(QString str);

QString text(); 获取显示内容

5.4.5.3. <QLineEdit>

QLineEdit(QWidget *parent=0)

QLineEdit(QString str, QWidget *parent=0)

void setText(QString str)

QString text(); 获取显示内容

void setEchoMode(QLineEdit::EchoMode mode);设置显示模式

mode：

Normal

NoEcho 无显示

Password 输入时看见**

PasswordEchoOnEdit 输入时看不见内容

QString str=edit.text();

int value=str.toInt();

将输入字符串数字改为数字

connect(&edit1,SIGNAL(textEdited(QString)),&edit2,SLOT(setText(QStrin
g)));

edit1 变化，edit2 同步显示

5.4.5.4. <QTextCodec>

QTextCodec *codec=QTextCodec::codeForName(“UTF8”);

本机文本编码

修改部分文字：

codec->toUnicode(“要转换的文字”)

修改整个程序的文字：

QTextCodec::setCodecForCStrings(codec);

5.4.5.5. <QSlider>

<QSpinBox>

connect(slider,SIGNAL(valueChanged(int),spinbox,SLOT(setValue(int)))

slider 滑块滑动，spinbox 框内数字变化

slider：

QSlider slider(QT::Horizontal,parent)

QT:;Vertical

int value()

void setMaxinum(int max)

void setValue(int value)

void SetMininum(int min)

void SetRange(int min,int max)

QSpinBox:

QSpinBox(QWidget *parent=0)

其他与 QSlider 一样

5.4.5.6. <QLCDNumber>

显示 LED 格式字符和数字

QLCDNumber(QWidget *parent=0);

lcd=new QLCDNumber(&w); 如果在 w 类内的构造函数定义，可以用 this 指针

lcd->setNumDigits(8);

lcd->setGeometry(10,10,23,23);

QTime t=QTime::currentTime(); <QTime>

lcd->display(t.toString("hh::mm::ss"));

5.4.5.7. <QMenuBar>

QMenuBar(QWidget *parent=0)

<QMenu>菜单项目

按顺序加入：

QMenu *menu=menubar->addMenu("&file");

必须用指针方式

menuaddAction("&open",&w,SLOT(ab()));

menuaddSeparator()；加上分隔线

5.4.5.8. <QPainter>

调用时间：

1）窗口显示时（程序开始运行时）

2）调用 QWidget 的 update()成员函数

3）系统认为需要重绘时

void paintEvent(QPaintEvent *){}

QPainter p(&w); 一般定义在窗口类中，用 this

p.drawLine(QLine(...))

p.drawRect(QRect(...))

p.drawEllipse(QRect(...)) 外切矩形的椭圆

p.drawEllipse(QPoint(中心点 x，y，半径 1，半径 2))

p.drawText(QRect(....),QT::AlignCenter,"xx");

p.drawImage(x,y,QImage("带路径的文件名")) <QImage>

1)绘制动作：drawLine，drawRect.....

2)选择工具动作：

QPen 影响开放性几何图形和封闭性集合图形的边缘

QBrush 只影响封闭图形内部

<QPen>

<QBrush>

<QFont>

p.setPen(QPen(QT::red));

p.setBrush(QBrush(QT::yellow));

p.setFont(QFont(“字体”,”画笔大小”，高度，bool 表示是否斜体));

5.4.5.9. 坐标体系

QPoint(int x, int y)

QLine(int x1, int y1, int x2, int y2)

QLine(QPoint p1, QPoint p2)

QRect(int x,int y, int wide, int height)

QSize(int wide, int height)

5.4.5.10. <QHBoxLayout>

horizontal

QHBoxLayout layout(&w);

创建一个 layout 指向主窗口

layout.addWidget(&button);

layout.addWidget(&button1);

将需要被 layout 控制的控件放入，layout 绑在主窗口，控件绑在 layout 上

QHBoxLayout(QWidget *parent=0);

void addWidget(QWidget *widget, int stretch=0)

第二个参数用来放大窗口占的总水平线的比例

void addLayout(QLayout *layout, int stretch=0)

void addStrech(int stretch=0)

void addSpacing(int size)

layout.addStretch(1) 最左边，向右挤

layout..... 按顺序加入排列内容

layout.addStretch(1) 最右边，向左挤

5.4.5.11. <QVBoxLayout>

与水平排列类相同，QH 改为 QV 即可

QHBoxLayout layout(&w);

QVBoxLayout v;

可使用： layout.addLayout(&r)

5.4.5.12. <QGridLayout>

QGridLayout layout(&w);

void addLayout(QLayout *layout, int row, int col)

void setRowStretch(int row, int stretch)

void setColumnStretch(int col , int stretch)

exp:两行两列排

layout.addWidget(&b1,0,0);

layout.addWidget(&b2,0,1);

layout.addWidget(&b3,1,0);

layout.addWidget(&b4,0,0);

5.4.6. 设置窗体位置

setGeometry(0, 0, 400, 400);

5.4.7. 消息框

<QMessageBox>

QMessageBox::warning(&w,"warning","信息字符串");

5.5. QML

5.5.1. 结构

引用对象重命名必须大写字母开头
import QtQuick 2.9 as Mm
import "Aa.js" as Jobj //引用 JavaScript 代码

```
Window {//实例化一个 Window 类的对象
    visible: true
    width: 640;height: 480
    title: qsTr("title")
```

```
    color: "#FF0000FF"//ARGB

    Mm.Component.onCompleted: console.debug("finish")  //初始化完成时调用

    Mm.MouseArea{
        anchors.fill: parent
        onClicked: console.debug("颜色：" + color)  //调试输出
        onReleased: console.debug(Jobj.fun(10))
    }

    Mm.Rectangle{//窗口中创建一个矩形区域
        id: aB3C  //id 必须小写字母或者下划线开头
        visible: true
        color: "#FF00FF00"
        width: 600
        height: 200
    }

    Mylable{//默认可以直接访问到同目录下对象类型，需要其他对象类型时需要 import
      Mm.Text{ text:"xxxx"}  //为 Mylable 中的 Text 对象默认属性赋值
    }
}


Mylable.qml:
Text {
    default property var someText//默认属性
    readonly property int intVal: 10   //只读属性
    text: "val:" + intVal + someText.text
}


Aa.js:
function fun(val){
    return val * 12;
}
```

5.5.2. 与 c++交互

5.5.2.1. 对象注入

类中声明属性调用函数

```
Q_PROPERTY(QString name READ getName WRITE setName)

//代码中将对象加入上下文
#include <QQmlContext>
QQmlApplicationEngine engine;
MyData data;
QQmlContext *ptx = engine.rootContext();
ptx->setContextProperty("msg",&data);

//qml 调用
msg.name = "abc";//调用 setName 函数
var val = msg.name;//调用 getName 函数
```

5.5.2.2. 注册 qml 对象

```
class MyData:public QObject
{

    Q_OBJECT
    Q_PROPERTY(QString name    READ getName WRITE setName NOTIFY nameChanged)
public:
    void setName(const QString &name)
    {
        if(m_name != name)
        {
            m_name = name;
            emit nameChanged(name);
        }
    }
    QString getName()const{
        return m_name;
    }
signals:
    void nameChanged(const QString &name);
    //函数首字母必须小写，对应槽函数 onNameChanged

private:
    QString m_name;



    //注册枚举类
    Q_ENUMS(Status)
```

```
public:
    enum Status
    {
        Status_A,
        Status_B,
    };
    Q_INVOKABLE Status getStatus() //注册可以被 qml 调用的函数，函数首字母必须小写
    {
        return Status_A;
    }
};



//代码中注册
qmlRegisterType<MyData>("jyl.data", 1, 0, "MyData");

//qml 调用
import jyl.data  1.0
MyData{
    id:myd
    onNameChanged: function(name)
    {//可以不写 function(name)
        console.debug(name);
    }
}

myd.name = "abc";//调用 setName 函数
var val = myd.name;//调用 getName 函数

//调用 getStatus()函数
var ss = myd.getStatus();
if(ss === 0) console.log("Status_A");
```

5.5.3. 布局

类型	说明
anchors	anchors.centerIn: *parent* 将元素位置固定到父元素中间

	anchors.right: *rl*.left anchors.rightMargin: 30 将元素位置固定到父元素左侧（window 无 right，可以在 window 上覆盖一个 Rectangle），且设置间距为 30 像素
Column { spacing: 10 }	将所有 visible 属性为 true 的项目排成一列，数据列间距设置为 10
Row {}	排成一行
Grid { columns: 3 rows:3 }	定义一个 3 行 3 列的排列网格，如果其中的元素数量超出 9 个，则会从第一个格子开始覆盖。

5.5.4. 控件

控件类型	使用
Item	所有项目都继承自 Item，常用与项目分组 Item { opacity: 0.2//不透明度，1 表示完全不透明。该属性应用于该项目下所有项目 }
Rectangle	使用纯色或渐变色绘制矩形区域
Text	Text{ font.pointSize: 30 text:"123" color: "blue" }
Button	import QtQuick.Controls 1.1 Button{text: "开始";y:0;onClicked: fun()}

5.5.5. 粒子系统

```
import QtQuick.Particles 2.0
import QtQuick.Controls 1.1
```

如果将渲染器和发射器放入粒子系统类，则不需要定义 id

```
ParticleSystem{
    id:par//定义粒子系统 par
    running: false
}

ItemParticle{
     system: par
     delegate://为 par 定义渲染器
     Rectangle{
              id:rect;width: 10;height: 10;color: "red";radius: 10
     }
}


Emitter{////为 par 定义发射器
      system: par
      emitRate: 50//每秒发射粒子数，默认 10
      x:0;width: parent.width //从 0 开始宽度为父对象宽度的直线上发射粒子
      velocity: PointDirection{y:800;x:400;yVariation: 100;}//发射方向（x 与 y 组合的向量），yVariation 表示粒子间 y 方向不同但是差异不超过该值
}


//按钮控制 par 的开始
Button{text: "开始";y:0;onClicked: par.start()}
```

第六章 openCV

6.1. Mat

6.1.1. 显示

```
using namespace cv;
  Mat img = imread("d:\\1.png");
  if(!img.data)
          return ;
  namedWindow("window", CV_WINDOW_AUTOSIZE);
  imshow("window", img);
```

6.1.2. 构造

```
cv::Mat mat(height, width, CV_16UC1);
mat.data[pos] = val;
```

16UC1 表示 16 位深度 1 通道

6.1.3. 转换

```
cv::Mat image = cv::cvarrToMat(ipl);
imwrite("D:\\111.jpg",image);
IplImage result=I;
```

6.2. 绘制到 windows 窗口

```
class DrawImg
{
public:
    DrawImg()
    {
        m_hbrush = CreateSolidBrush(RGB(0,0,0));
    }
    void Init(CWnd *pWnd)
    {
        pBmpInfo = (BITMAPINFO *)chBmpBuf;
        pBmpInfo->bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
```

```
        pBmpInfo->bmiHeader.biPlanes = 1;
        pBmpInfo->bmiHeader.biCompression = BI_RGB;
        for (int i = 0;i<256;i++)
        {
            pBmpInfo->bmiColors[i].rgbBlue = i;
            pBmpInfo->bmiColors[i].rgbGreen = i;
            pBmpInfo->bmiColors[i].rgbRed = i;
            pBmpInfo->bmiColors[i].rgbReserved = i;
        }

        pWnd->GetClientRect(wrect);
        CDC *pdc = pWnd->GetDC();
        hdc = pdc->GetSafeHdc();
    }
    void Draw(IplImage *pImg)
    {
        pBmpInfo->bmiHeader.biWidth = pImg->width;
        pBmpInfo->bmiHeader.biHeight = pImg->height;
        pBmpInfo->bmiHeader.biBitCount = pImg->depth *  pImg->nChannels;


        int  x  =  0,  y  =  0,  width  =  pBmpInfo->bmiHeader.biWidth,  height  =
pBmpInfo->bmiHeader.biHeight;
        if (NULL != pImg->roi) { x = pImg->roi->xOffset; y = pImg->roi->yOffset; width =
pImg->roi->width; height = pImg->roi->height; }



        FillRect(hdc,wrect,m_hbrush);
        int DesX =0;
        int DesY =wrect.Height();
        int DesWidth = wrect.Width();
        int DesHeight = wrect.Height();
        double ratio = pImg->width/(double)pImg->height;
        int WidthRequest = ratio * wrect.Height();
        int HeightRequest = wrect.Width()/ratio;
        if (wrect.Width() < WidthRequest)
        {
            DesHeight = DesWidth / ratio;
            DesY = (wrect.Height() - DesHeight)/2 + DesHeight;
        }
        else if (wrect.Height() < HeightRequest)
        {
            DesWidth = DesHeight*ratio;
```

```
            DesX = (wrect.Width() - DesWidth)/2;
        }



        ::SetStretchBltMode(hdc, COLORONCOLOR);
        ::StretchDIBits(hdc,
            DesX, DesY, DesWidth, -DesHeight,
            x, y, width, height,
            pImg->imageData, pBmpInfo,
            DIB_RGB_COLORS, SRCCOPY);
    }
private:
    char           chBmpBuf[2048];
    BITMAPINFO     *pBmpInfo;
    CRect wrect;
    HDC hdc;
    HBRUSH m_hbrush;
};
```

==========使用

```
DrawImg m_DrawImg;
m_DrawImg.Init(this);
m_DrawImg.Draw(pImg);
```

```
        Mat img;
    int depth = img.depth();
    if(CV_8U == depth || CV_8S == depth) depth = 8;
    else if(CV_16U == depth || CV_16S == depth) depth = 16;
    else if(CV_32S == depth || CV_32F == depth) depth = 32;
    else if(CV_64F == depth) depth = 64;
```

6.3. 视频

6.3.1. 播放 avi

```
cvNamedWindow("aviWindow",CV_WINDOW_AUTOSIZE);
CvCapture* capture=cvCreateFileCapture("d:\\1.avi");
IplImage* frame;
```

```
while(1)
{
    frame=cvQueryFrame(capture);
    if(!frame)
        break;
    cvShowImage("aviWindow",frame);
    char c=cvWaitKey(33); //相当于设置帧速率
    if(c==27)break; //表示按esc退出
}
cvReleaseCapture(&capture);
cvDestroyWindow("aviWindow");
```

6.3.2. avi 设置

获取 avi 总帧数

```
double frameCount=cvGetCaptureProperty(capture,CV_CAP_PROP_FRAME_COUNT);
```

第二个参数可以是：

CV_CAP_PROP_POS_MSEC - 影片目前位置，为毫秒数或者视频获取时间戳
CV_CAP_PROP_POS_FRAMES - 将被下一步解压/获取的帧索引，以 0 为起点
CV_CAP_PROP_POS_AVI_RATIO - 视频文件的相对位置（0 - 影片的开始，1 - 影片的结尾)
CV_CAP_PROP_FRAME_WIDTH - 视频流中的帧宽度
CV_CAP_PROP_FRAME_HEIGHT - 视频流中的帧高度
CV_CAP_PROP_FPS - 帧率
CV_CAP_PROP_FOURCC - 表示 codec 的四个字符
CV_CAP_PROP_FRAME_COUNT - 视频文件中帧的总数

设置播放位置

```
cvSetCaptureProperty(capture,CV_CAP_PROP_POS_FRAMES,frameCount/2);
```

第二个参数可以是：

CV_CAP_PROP_POS_MSEC - 从文件开始的位置，单位为毫秒
CV_CAP_PROP_POS_FRAMES - 单位为帧数的位置（只对视频文件有效）

CV_CAP_PROP_POS_AVI_RATIO - 视频文件的相对位置（0 - 影片的开始，1 - 影片的结尾)
CV_CAP_PROP_FRAME_WIDTH - 视频流的帧宽度（只对摄像头有效）
CV_CAP_PROP_FRAME_HEIGHT - 视频流的帧高度（只对摄像头有效）
CV_CAP_PROP_FPS - 帧率（只对摄像头有效）
CV_CAP_PROP_FOURCC - 表示 codec 的四个字符（只对摄像头有效）

6.3.3. 捕获摄像头

```
cvNamedWindow("aviWindow",CV_WINDOW_AUTOSIZE);
```

```
//CvCapture* capture=cvCreateCameraCapture(0);
CvCapture* capture=cvCaptureFromCAM(0);//代表第一个摄像头,该函数必须在主线程中执行
if(!capture){exit(0);}
IplImage* frame;
while(1)
{
    frame=cvQueryFrame(capture);
    if(!frame)
        break;
    cvShowImage("aviWindow",frame);
    char c=cvWaitKey(33); //相当于设置帧速率
    if(c==27)break; //表示按esc退出
}
cvReleaseCapture(&capture);
cvDestroyWindow("aviWindow");
```

6.3.4. 保存为 avi

程序运行前确保 avi 文件名不存在，保存文件的路径存在，否则创建 writer 会失败。
如果要将保存avi和摄像头捕捉显示同时进行，注意避免同时访问frame的冲突。

(int)cvGetCaptureProperty(cap,CV_CAP_PROP_FOURCC)可以获取编码模式

全局变量：

```
CvVideoWriter *writer=NULL;
```

保存过程：

```
int fps     = 12;
CvSize size=cvSize((int)cvGetCaptureProperty(capture,CV_CAP_PROP_FRAME_WIDTH),(int)cvGetCaptureProperty(capture,CV_CAP_PROP_FRAME_HEIGHT));
CTime time=CTime::GetCurrentTime();
CString str=time.Format("%Y-%m-%d  %H-%M-%S");
CString file="d:\\avi\\"+str+".avi";//ANCI时
writer=cvCreateVideoWriter(file,CV_FOURCC('I', 'Y', 'U', 'V'),fps,size);//此参数传递-1，可以查看和选择已有的编码模式，也可以传入其他数字进行测试。
while((frame=cvQueryFrame(capture))!=NULL)
{
    Sleep(fps);
    cvWriteFrame(writer, frame);
}
```

析构：

```
cvReleaseVideoWriter(&writer);
```

6.4. IplImage

6.4.1. IplImage 构造

```
int depth = IPL_DEPTH_8U;
int channels = 1; //黑白 1， RGB 3
IplImage *pImg = cvCreateImage(cvSize(Width,Height),depth,channels);

for (int row = Height -1 ; row >= 0; --row)
{
    for (int col = 0; col < Width; ++col)
    {
    int pos = pImg->widthStep * row + col;// 如果 depth 为 IPL_DEPTH_16U   int pos = pImg->widthStep * row + col*2;
    pImg->imageData[pos] = dat;
    }

}
```

6.4.2. IplImage 结构体赋值

```
void InitIpl(IplImage &img,char* DibBuf,int width,int height,int depth,int nchannel)
{
        img.nSize = sizeof(IplImage);
        img.ID = 0;
        img.alphaChannel =0;
        img.depth = depth;
        strcpy_s(img.colorModel,4,"RGB");
        strcpy_s(img.channelSeq,4,"BGR");
        img.dataOrder = 0;
        img.origin = 0;
        img.align = 4;
        img.roi = NULL;
        img.maskROI = NULL;
        img.imageId = NULL;
        img.tileInfo = NULL;
        memset(img.BorderMode,4,0);
        memset(img.BorderConst,4,0);
```

```
        img.imageDataOrigin = 0;
        img.width = width;
        img.height = height;
        img.nChannels = nchannel;
        img.imageSize = img.width * img.height  * img.nChannels;
        img.widthStep = img.width  * img.nChannels;
        img.imageData = DibBuf;
}
```

6.4.3. 读取图像

IplImage *pImg = cvLoadImage("D:\\1.png",CV_LOAD_IMAGE_ANYCOLOR|CV_LOAD_IMAGE_ANYDEPTH);
第二个参数可以不写，默认是将图像以 8 位 3 通道载入。
CV_LOAD_IMAGE_GRAYSCALE 代表以单通道载入

第二个参数 iscolor
>0　强制以彩色（3 通道）
=0　灰度
<0　取决于图像本身

6.4.4. 弹窗显示

可以在 app 类 initInstance 创建对话框前调用函数，取代对话框。

```
IplImage* img=cvLoadImage("d:\\1.jpg"); //将图像加载到内存中


cvNamedWindow("picWindow",CV_WINDOW_AUTOSIZE);
//窗口名为picWindow,第二个参数默认0，表示任意拉伸。CV_WINDOW_AUTOSIZE表示窗口大小是图像的大小，不能拉伸

cvShowImage("picWindow",img);
//使用创建的窗口显示图片


cvWaitKey(0);//使程序暂停，等待用户按下键盘，参数0代表一直等待，如果参数为正数x，代表等待x毫秒
cvReleaseImage(&img);//释放内存中的图像
cvDestroyWindow("picWindow");//销毁创建的窗口
```

6.4.5. ROI

```
cvSetImageROI(img,cvRect(x,y,width,height));
cvResetImageROI(img);
```

在 cvSetImageROI和cvCopy函数时，一定要注意图像的大小（或ROI大小），避免越界和图像大小不一。

6.4.6. 保存图片

可以将img保存为任意格式图片，保存图片的目录必须存在

```
cvSaveImage("d:\\x.png",img);
```

6.4.7. 创建与销毁

```
IplImage* img = cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,3);
cvReleaseImage(&img);
```

创建时，也可以使用cvGetSize(img)作为参数

```
IplImage*                    img                    =
cvCreateImage(cvGetSize(src),src->depth,src->nChannels);
```

6.4.8. 归零

```
cvZero(img);
```

6.4.9. 设置颜色

```
cvSet(pImg, cvScalar(0x00,0xDB, 0x00));
```

6.4.10. 翻转

```
cvFlip(img,img,0);
```

IplImage中的origin成员为0，代表上下翻转，1代表左右翻转。

6.4.11. 复制

cvCopy(src, dst);//将src拷贝到dst中

IplImage* dst=cvCloneImage(img);//将img完整复制一份

6.4.12. 获取尺寸

```
int cx=cvGetSize(img).width;

int cy=cvGetSize(img).height;
```

或者

```
int cx=img->width;
int cy=img->height;
```

6.4.13. 调整大小

将src变为dst的大小，dst为缩放后的图像。

```
IplImage* dst = cvCreateImage(cvSize(200,60),src->depth,src->nChannels);
cvResize(src,dst);

cvReleaseImage(&dst);//放在函数最后（处理图像结束后）释放内存
```

6.4.14. 裁剪

将 src 进行裁剪，裁剪后的图像为 dst。

```
int x=0,y=0,width=200,height=200;
IplImage* dst = cvCreateImage(cvSize(width,height),src->depth,src->nChannels);
cvSetImageROI(src,cvRect(x,y,width,height));//设置感兴趣的区域，也就是用来操作的区域
cvCopy(src,dst);
```

6.4.15. 分离通道

cvSplit(src,r,g,b,NULL);

将src的rgb分别放到三个通道图中,r,g,b三个IplImage都是单通道的

6.4.16. 平滑处理

将src平滑处理后放到dst

cvSmooth(src,dst,CV_GAUSSIAN ,3,3);

第三个以后的参数都有默认值。

第三个参数可以是:

CV_BLUR_NO_SCALE (简单不带尺度变换的模糊) - 对每个象素的 param1×param2 领域求和。如果邻域大小是变化的，可以事先利用函数 cvIntegral 计算积分图像。
CV_BLUR 对每个象素param1×param2邻域 求和并做尺度变换 1/(param1.param2).
CV_GAUSSIAN对图像进行核大小为 param1×param2 的高斯卷积
CV_MEDIAN对图像进行核大小为param1×param1 的中值滤波(邻域是方的).
CV_BILATERAL (双向滤波) - 应用双向 3x3 滤波，彩色sigma=param1，空间 sigma=param2.
平滑操作的第一个参数.

6.4.17. 融合

两个图象可以是任何象素类型，只要它们的类型相同。它们可以是单通道或是三通道，只要它们相符。必须有相同的象素类型。这些图象可以是不同的尺寸，但它们的 ROI 必须有相同的大小。
融合时候，以图像本身大小为基准，而不是以显示界面的大小为基准
IplImage *dst, *src;
dst存储合成后的图

```
 int width =200,height =300; //设置ROI大小相同
 cvSetImageROI(dst, cvRect(0,0,width,height));
 cvSetImageROI(src, cvRect(0,0,width,height));
//前两个参数代表图像左上角位置

 double alpha =0.5;
 double beta  =1;//设置为0和1可以让上面的图盖在底图上
 double gamma=0;
 cvAddWeighted(dst, alpha, src,beta,gamma,dst);
//运算是dst=dst*alpha+src*beta+gamma;

 cvResetImageROI(dst); //因为ROI变小，所以重置回原来的大小
```

6.4.18. 去白色融合

去掉 src2 的白色，融合图放在 src3

```
IplImage* src3 = cvCreateImage(cvSize(width,height),src1->depth,src1->nChannels);
cvAnd(src1,src2,src3);


cvAndS(src,cvScalar(255,255,0),src3); //可以用指定颜色进行位与
```

6.4.19. 调色

为图片增加红色（红通道加了 100，如果是负数代表减少颜色），cvScalar 可以有四个参数 BGRA

```
cvAddS(img,cvScalar(0,0,100),img);
```

6.4.20. 反色

```
for(int i=0;i != height; ++ i)
{
	for(int j=0;j != width; ++ j)
	{
		for(int k=0;k != channels; ++ k)
		{
			data[i*step+j*channels+k]=255-data[i*step+j*channels+k];
		}
	}
}
```

6.4.21. 边缘检测

```
IplImage* img=cvLoadImage("d:\\1.jpg",CV_LOAD_IMAGE_GRAYSCALE); //灰度图像
IplImage* CannyImg=cvCreateImage(cvGetSize(img),IPL_DEPTH_8U,1);
cvCanny(img,CannyImg,50,150,3);//输出为 CannyImg
```

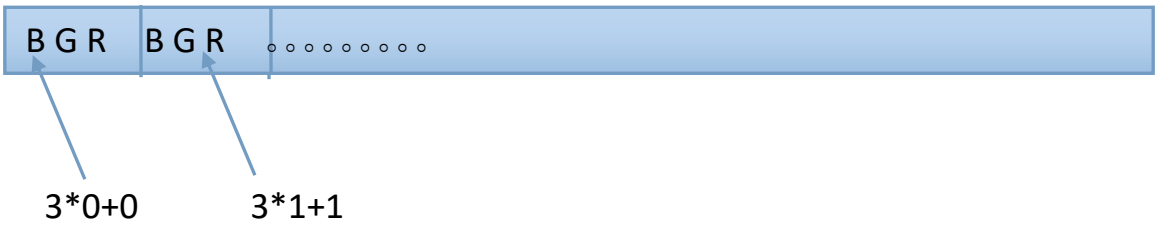
6.4.22. 遍历 IplImage 处理图像

将 img 中蓝色最大化

```
IplImage* img = cvCreateImage(cvSize(src->width,src->height),IPL_DEPTH_8U,3);
cvCopy(src, img);
```

```
for(int y=0;y<img->height;y++)
{
    uchar* ptr=(uchar*)(img->imageData+y*img->widthStep);//ptr 指向一行的数据长度，也就是指向的内存大小为一行的内存大小
    for(int x=0;x<img->width;x++)
    {
    ptr[3*x+0]=255;//0，1，2 分别代表蓝绿红，3*x 是因为这是 3 通道的图像
    }
}
```

ptr 指向的行内存排列应该是：



6.4.23. 替代 ROI 方式

使 img 的部分区域红色增加 100

```
int x=10,y=10,width=400,height=400;//感兴趣区域
IplImage* sub_img=cvCreateImageHeader(cvSize(width,height),img->depth,img->nChannels);
sub_img->origin=img->origin;
sub_img->widthStep=img->widthStep;
sub_img->imageData=img->imageData + img->widthStep*y + img->nChannels*x;
cvAddS(sub_img,cvScalar(0,0,100),sub_img);//让红通道增加 100
cvReleaseImageHeader(&sub_img);
```

ROI 只能串行处理，且不断设置和重置，该方式可以保持一副图像的多个子区域处于活动状态下。

6.4.24. 拉伸变换

不能对一幅图进行两次拉伸，需要多次拉伸时，用多个 dst 接收 src 的变换，然后设置 ROI 进行合成即可。

```
CvPoint2D32f srcTir[4],dstTir[4];
CvMat *warp_mat=cvCreateMat(3,3,CV_32FC1);
```

```
IplImage *src=cvLoadImage("d:\\1.jpg");
IplImage *dst=cvCloneImage(src);
dst->origin=src->origin;
cvZero(dst);

///src 中要进行变换的四个点坐标
srcTir[0].x=52;
srcTir[0].y=0;//LT
srcTir[1].x=src->width-52;
srcTir[1].y=0;//RT
srcTir[2].x=0;
srcTir[2].y=300;//LB
srcTir[3].x=src->width;
srcTir[3].y=300;//RB


//变换后的四个点坐标
dstTir[0].x=0;
dstTir[0].y=0;
dstTir[1].x=src->width;
dstTir[1].y=0;
dstTir[2].x=0;
dstTir[2].y=300;
dstTir[3].x=src->width;
dstTir[3].y=300;

cvGetPerspectiveTransform(srcTir,dstTir,warp_mat);
cvWarpPerspective(src,dst,warp_mat);


////对 dst 进行显示//////

cvReleaseImage(&dst);
cvReleaseMat(&warp_mat);
```

6.4.25. 转换颜色类型

```
#pragma comment(lib,"opencv_objdetect248d")
#pragma comment(lib,"opencv_imgproc248d")

IplImage* gray = cvCreateImage(cvSize(img->width, img->height), 8, 1);
cvCvtColor(img, gray, CV_BGR2GRAY);
```

6.5. 绘图

6.5.1. 矩形

在两个点定位处绘制一个白色正方形,最后一个参数是线条宽度，如果是-1，代表画一个填充矩形。

```
cvRectangle(img,cvPoint(10,10),cvPoint(50,50),cvScalar(255,255,255),-1);
```

6.5.2. 圆

```
cvCircle(pImg, cvPoint(x,y), 8,cvScalar(0,0,255), CV_FILLED);
//最后一个参数是线条宽度，cvScalar(0,0,255)  BGR
//8代表圆的半径，最后一个参数是线条宽度
```

6.5.3. 椭圆

```
cvEllipse(img,cvPoint(x,y),cvSize(width,height),0,0,360,cvScalar(255,255,255),1);
//第二个参数代表中心点，第三个参数代表了椭圆的两个半径，第五个是偏离主轴的角度，六七代表弧线开始和结束角度，一般都是 0 和 360。最后一个是线条宽度
```

6.5.4. 直线

```
cvLine(img,cvPoint(10,10),cvPoint(50,50),cvScalar(255,255,255),1);
```

最后一个参数是线宽，默认为 1

6.5.5. 文字

只能绘制英文

```
CvFont font;
cvInitFont(&font,CV_FONT_HERSHEY_SCRIPT_SIMPLEX,hscale,vscale,0,thick);
cvPutText(img,"3",cvPoint(100,200),&font,cvScalar(0,0,255));
```

hscale 和 vscale 代表字体宽和高的间距，thick 代表字体线条宽度,thick 的前一个参数代表是否字体倾斜

6.6. 计时

获取秒差:

```
double timeSpan = (GetTickCount() - time1 )/GetTickFrequency();
```

6.7. 图像处理

6.7.1. 腐蚀膨胀

cvErode()腐蚀后 cvDilate()膨胀，叫作开操作，那些离散点或游丝线、毛刺就被过滤
cvDilate()膨胀后 cvErode()腐蚀，叫作闭操作，那些断裂处就被缝合。
函数的输入和输出可以是一个指针

```
IplConvKernel *ConvKernel = cvCreateStructuringElementEx(3, 3, 1, 1, CV_SHAPE_RECT);
//表示 3*3 矩形进行操作


cvErode(pImg,pImg, ConvKernel, 1);//腐蚀,  最后一个参数代表腐蚀次数
cvDilate(pImg, pImg, ConvKernel, 4);//膨胀
```

6.7.2. 查找轮廓

cvFindContours
需要二值图像，非 0 作为 1，0 保持不变。识别 1
0 是背景，1 是图案
会按照边缘连续性，图案中空心则会识别两个轮廓

cvThreashold 对单通道操作，一般是对灰度图转换为 2 值图
cvThreshold(pImg,pImg, 128, 1, CV_THRESH_BINARY);//将大于 128 的灰度值（白色）变为 1，小于 128 的变成 0
第三个参数代表阈值，第四个代表转换后最大值(0,1)
CV_THRESH_BINARY: value > threshold ? max_value : 0
CV_THRESH_BINARY_INV 则相反处理

```
CvMemStorage* storage = cvCreateMemStorage(0);
CvSeq* contour = 0;
cvFindContours(gray, storage, &contour);//gray 为 2 值图像

cvZero(gray);

for (; contour != 0; contour = contour->h_next)
{
    double contourSize = fabs(cvContourArea(contour));//轮廓面积
    ///CvRect rect = cvBoundingRect(contour, 0);//轮廓外接矩形
```

```
            if(contourSize > 200) continue;//挑选面积小的



            cvDrawContours(pImg, contour,
                cvScalar(255,255,255),cvScalar(255,255,255),
                -1,CV_FILLED);
            //pImg 是原始图，gray 是二值图
            //CV_FILLED 代表完全填充，cvScalar 参数可以是 1 到 4 个，和图像位数相关

        }

            cvReleaseMemStorage(&storage);



int GetCountersNum(CvSeq* contour)
{//获取轮廓数量
    int total = 0;
    CvSeq* contour_bak = contour;
    for (; contour != 0; contour = contour->h_next)    total++;
    return total;
}
```

6.8. gamma 增亮

```
void Gamma(int width,int height,BYTE *pImg)
{
    static   const double gamma = 0.45;
    static   const double offset = 0.055;
    static   const double threshold = 0.0031308;
    static   const double maxgray = 255;
    static   const double s = ((1 + offset)*pow(threshold, gamma) - offset) / threshold;

    for (int row=0; row< height; ++row)
    {
        for (int col = 0 ; col < width;++col)
        {
            BYTE *pVal = pImg + row*width + col;
            double gray = (*pVal) / maxgray;

            double SI = 0;
            if (gray <= threshold)
                SI = s*gray*maxgray;
```

```
                else
                    SI = maxgray*((1 + offset)*pow(gray, gamma) - offset);

                *pVal = (BYTE)SI;
            }
        }
    }
```

6.9. OpenCVSharp

6.9.1. 图像显示

```
Mat src = new Mat("D:\\1.jpg", ImreadModes.AnyColor);
//Cv2.ImShow("im", src);
//Cv2.WaitKey();
Bitmap bitmap = OpenCvSharp.Extensions.BitmapConverter.ToBitmap(src);//using System.Drawing;
using (MemoryStream stream = new MemoryStream())
{
                    bitmap.Save(stream, System.Drawing.Imaging.ImageFormat.Jpeg); //不带透明度
                    stream.Position = 0;
                    BitmapImage result = new BitmapImage();
                    result.BeginInit();
                    result.CacheOption = BitmapCacheOption.OnLoad;
                    result.StreamSource = stream;
                    result.EndInit();
                    result.Freeze();
                    img.Source = result;
}
```

6.9.2. 获取点阵数据

```
IntPtr datpr = src.Data;
```

6.9.3. 创建图像

```
Mat.Zeros(row, col, OpenCvSharp.MatType.CV_8UC1);
```

6.9.4. 获取设置像素

```
int pix = src.At<byte>(i, j);

ret.SetArray(i, j, r);
```

6.9.5. 获取图像中最大最小值

```
double min, max;
src.MinMaxIdx(out min, out max);
```

6.9.6. 图像合并

```
Mat src = new Mat("D:\\a.jpg", ImreadModes.AnyColor | ImreadModes.AnyDepth);
Mat src2 = new Mat("D:\\b.jpg", ImreadModes.AnyColor | ImreadModes.AnyDepth);
if (src.Rows != src2.Rows) return;
Mat MatrixCom = new Mat(src.Rows, src.Cols + src2.Cols, src.Type());
Mat temp = MatrixCom.ColRange(0, src.Cols);
src.CopyTo(temp);
temp = MatrixCom.ColRange(src.Cols, src.Cols + src2.Cols);
src2.CopyTo(temp);
Cv2.ImWrite("D:\\c.jpg", MatrixCom);
```

第七章 boost

7.1. 编译

1.打开 vs 命令提示符（x86 本机工具命令提示符）

2.进入 boost 包目录

3.命令运行 bootstrap.bat

4.命令运行 b2.exe install

编译后会在 c 盘根目录下生成 boost 文件夹

b2.exe stage --toolset=msvc --stagedir="D:\boost\out\bin" link=static

threading=multi runtime-link=static --build-type=complete –

-build-dir="D:\boost\out\build"

7.2. 时间

<boost\timer.hpp>

timer t;

.....

t.elapsed();//获取从对象创建到当前位置的时间

<boost\progress.hpp>

progress_timer t;

可以像 timer 一样使用，还可以在退出作用域时自动输出时间。

<boost\date_time\gregorian\gregorian.hpp>

```
    gregorian::date d1;
    gregorian::date d2(2016, 4, 17);
    gregorian::date d3(d2);

    if (gregorian::date(boost::date_time::not_a_date_time) == d1)
    {
        cout << "not" << endl;
    }
    if (d2 == d3)
    {
        cout << "d2 == d3" << endl;
    }
```

获取当前时间

```
#include <boost/date_time/posix_time/posix_time.hpp>
```

```
std::string strTime =
boost::posix_time::to_iso_string(boost::posix_time::second_clock::local_time());
// strTime 里存放时间的格式是 YYYYMMDDTHHMMSS，日期和时间用大写字母 T 隔开
int pos = strTime.find('T');
strTime.replace(pos, 1, std::string("-"));
strTime.replace(pos + 3, 0, std::string(":"));
strTime.replace(pos + 6, 0, std::string(":"));
```

7.3. 内存管理

<boost\shared_ptr.hpp>
可以在多线程中安全使用

```
shared_ptr<int> p(new int(5));
cout << *p << endl;

//使用 make_shared 抵消 new 的使用
shared_ptr<int> p(make_shared<int>(5));
```

7.4. 文件操作

```
#include<boost/filesystem.hpp>

m_filePath = "d:\\filename";
boost::filesystem::path path(m_filePath);
if (boost::filesystem::exists(path))
{
    boost::filesystem::remove(path);
}
```

7.5. 序列化

```
boost::archive::binary_oarchive      ioa(stream);

//可以是文件流或者字符流

//包括 xml、text 和 binary 的 io 类型

//xml 序列化要用 oa & BOOST_SERIALIZATION_NVP(a)
```

ioa& Entity;//必须是支持序列化的类型

```
#include <fstream>

#include<boost\archive\binary_iarchive.hpp>
#include<boost\archive\binary_oarchive.hpp>
#include <boost\serialization\vector.hpp>
using namespace std;


class Test
{
protected:
    int id;
    vector<int> v;
public:
    Test() {};
    Test(int d) :id(d){}
    void Set(int x)
    {
        v.resize(10, x);
    }
private:
    friend class boost::serialization::access;
    // 如果类 Archive 是一个输出存档，则操作符& 被定义为<<.  同样，如果类 Archive
    // 是一个输入存档，则操作符& 被定义为>>.
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version)
    {
        ar & id;
        ar & v;
    }
};



// 保存数据到存档
    {
        // 创建类实例
```

```
    Test g(1);
    g.Set(5);
    // 创建并打开一个输出用的字符存档
    std::ofstream ofs("d:\\filename");

    boost::archive::binary_oarchive oa(ofs);
    // 将类实例写出到存档
    oa << g;
    // 在调用析构函数时将关闭存档和流
}

//将类实例恢复到原来的状态
{
    Test newg;

    // 创建并打开一个输入用的存档
    std::ifstream ifs("d:\\filename");
    boost::archive::binary_iarchive ia(ifs);
    // 从存档中读取类的状态
    ia >> newg;
    // 在调用析构函数时将关闭存档和流
}
```

7.6. 解析 json

```
#include <boost/property_tree/ptree.hpp>
#include <boost/property_tree/json_parser.hpp>

    std::string json = "{\"A\":1,\"B\":{\"C\":2,\"D\":3},\"E\":[{\"F\":4},{\"F\":5}]}";
    boost::property_tree::ptree pt, child1, child2;
    std::stringstream ss(json);
    boost::property_tree::read_json(ss, pt);
    child1 = pt.get_child("B");
    for (auto c : child1)
    {
        cout << c.first << c.second.data() << endl;
    }
```

第八章 C#

8.1. grammar

8.1.1. 命名空间

可以通过using重定义类型名
using NewMsgSizeType=UInt32;

可以在命名空间 A 内嵌套命名空间 B，相当于在 A 内定义一个变量 B，所以 A 和 B 不能互相访问元素。

命名空间内只能包含类，枚举，结构体，委托等，不能包含变量和函数。

默认情况下存在一个全局命名空间，所以在命名空间外定义的类可以直接进入到此全局命名空间。在全局命名空间中包含的类内写 static 变量，就相当于全局变量。

8.1.2. 基本数据类型

8.1.2.1. 基本类型

类型	字节数	描述
Byte	1	unsigned byte
Sbyte	1	signed byte
Short	2	signed short
Ushort	2	unsigned short
Int	4	signed integer
Uint	4	unsigned integer
Long	8	signed long
Ulong	8	unsigned long
Float	4	floating point number

Double	8	double precision number
Decimal	8	fixed precision number
String		Unicode string
Char	2	Unicode char
Bool		true，false，boolean

8.1.2.2. 可空类型

int? x = 5; //x 可以赋值为 null
与非可空类型运算时，需要显示转换
int y = (int)x* 2;或者 int y = x.Value * 2;

??运算

允许给值可能为 null 的表达式赋另外一个值
x??y　等价 x==null?y:x

使用时如：
int? x = null;
int result = x * 2 ?? 5;
结果为 5

8.1.2.3. Var

可以指代任何类型的变量
var x="***";
var a = new A();

8.1.3. 类

class 类型必须用 new，在函数中传递时是引用传递（可以不用 ref，但在函数中不能改变 class 对象的指向）。

8.1.3.1. 静态成员

因为 C#没有全局变量和函数。Static 成员不属于对象实例，所以可以在不实例化类的情况下

直接调用。

C#中，const 变量也可以被 static 函数调用。

静态变量可以用来在类与类之间进行数据交互和共享。

class A{ public static int x=10;}

在其他任何地方，都可以使用 A.x 进行访问。

8.1.3.2. 属性

属性比字段（变量）包含的内容多，在修改状态时还可以进行额外的操作。

包含两个函数块（get 和 set），一个用于获取属性，一个用于设置属性。这两个块也称为访问器。

```
class Data
{
        int month=10;//私有的，外部不能直接访问
        public int Month //指代变量
        {
            get { return month;}
            set { month = value;}
        }
}
```

使用时

```
Data data = new Data();
data.Month = 20;//设置
int x=data.Month;//获取
```

自动属性

```
public int x { set; get; }
```

```
class Cu
{
        public string Name { get; set; }
        public string City { get; set; }
}
```

```
Cu c = new Cu() { Name = "ab", City = "12" };
```

8.1.3.3. 权限

只能修饰单个成员，不能用 public:修饰下面的所有成员。

类可以用 internal 修饰，表示只能在当前项目访问。子类的可访问性不能高于父类（internal 可以继承 public，反之不可）。

可以用 public，private，protected 修饰

readonly

只能在初始化或者执行构造时赋值。仅用于修饰类的数据成员，和 const 不同之处在于 const 要求在声明时进行直接初始化，readonly 是指一旦进行了写操作或者初始化，就不可以修改。readonly 修饰变量不能在函数中定义，只能修饰类的成员变量。

sealed

可以用来修饰类，该类不允许被继承。

8.1.3.4. 构造

```
class A
{
        int x, y;
```

```
        public A(int x,int y)
        { this.x = x; this.y = y;}
}
```

使用时：A a = new A(1, 3); //带参数的构造函数只能用new调用

A a; //该方式和无参构造无关，在类内定义无参构造没有意义，即使定义为private也可以。

8.1.3.5. 多态

父类使用virtual关键字，子类实现虚函数使用override关键字。

```
class B
{
    public virtual void fun() {Console.WriteLine("B");}

}
class C : B
{
    public override void fun(){Console.WriteLine("C");}
}
```

使用时：

```
B b=new C();
b.fun(); //输出C
```

和C++概念相同，父类指向子类，本质还是父类（忽略子类中除了父类以外的内容），子类只是用来改写父类的虚函数。

如果父类和子类有重名的函数或者变量，使用base在子类中引用父类的成员：base.fun();

8.1.3.6. 继承

如果没有指定继承的父类，那么就是默认只继承自 System.Object。所以，每个类都是继承自 System.Object。

子类中有和父类相同的成员，就会隐藏父类的成员。使用 base 关键字就可以访问被隐藏或者重写的父类成员,如 base.fun()。

```
class A{A(int x){} }
class B : A {//没有protected和private继承
    B():base(5){} //初始化参数列表中用base关键字指定父类构造中需要的参数
}
```

8.1.3.7. Abstract

主要用作对象系列的基类，共享某些主要特性。

抽象类不能直接实例化，也可以和接口一样声明函数，由派生类实现。

```
abstract class A
{
    int x;
    public void fun1(){Console.WriteLine("aa");}
    public abstract void fun2();
}
class B : A
{
    override public void fun2(){Console.WriteLine("xx");}
}
```

8.1.3.8. 释放对象

```
using(x){.....}
```

在 using 的代码块中使用变量 x，代码块结束后，立即释放对象。

8.1.3.9. partial

使用 partial 关键字可以让一个类在不同的文件中定义

==========文件 1 内

```
namespace N
{
        partial class A
      {
              public void fun1() { Console.WriteLine("fun1"); }
              partial void fun3();//这种只声明，不实现的函数必须是 void，而且不能用 public 这种修饰。如果声明的这个函数没有被实现，编译器就会在编译时自动删除相关代码。
      }
}
```

==========文件 2 内

```
namespace N
{
      partial class A
      {
            public void fun2() { Console.WriteLine("fun2"); fun3(); }
            partial void fun3() { Console.WriteLine("fun3"); }
      }
}
```

==========使用时

```
using N;

A a = new A();
a.fun1();
a.fun2();
```

8.1.4. Object

可以指代任何对象

```
void fun(object[] obj)
 {
      foreach(object o in obj)
            Console.WriteLine(o);
 }
```

```
fun(new object[]{1,3.3,"adf"});
```

8.1.5. 接口

```
interface X1{ void fun1();    void fun2();}//声明接口
interface X2 { int a { get; } }
class X : X1, X2 //用多继承接口，并实现接口
{
    void X1.fun1() {Console.WriteLine("fun1");}//显示实现
    public void fun2(){Console.WriteLine("fun2");}//隐式实现
  public int a { get;protected set; } //实现时必须要get，set同时存在，如果接口中不存在其中一个，就要添加。添加时修饰符必须要比接口中的严格（接口是public，所以此处可以用protected）
}
```

调用时

```
X x = new X();
X1 x1 = (X1)x; //将类转换为某一接口的内容
x1.fun1();
x.fun2();
```

8.1.6. 委托

可以把函数引用保存在变量中，类似于函数指针。使用关键字delegate。

```
delegate void fun(int x);

void fun1(int x){}
void fun2(int x){}
```

```
//创建对象调用
fun ff = new fun(fun1);
ff(1);

//直接调用
fun ff = fun1;
ff(1);


//Lambda
fun ff = (int x) => { Console.WriteLine("{0}", x); };
ff(1);


//事件
fun ff = fun2;
ff += fun1;
ff += fun3;
ff -= fun1;
ff(1);
```

8.1.7. Attribute

```
using System.Diagnostics;
```

添加额外信息

8.1.7.1. 内置 attribute

函数只在 debuge 下有效

```
[Conditional("DEBUG")]
void fun(){}


[Obsolete("过时的方法",true)]     第二个参数为编译时是否报错
 void fun(){}
```

8.1.7.2. 自定义 attribute

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = false, Inherited = false)]
        //代表该 Attribute 只能用在 class 上一行，不能连续两行出现该 Attribute，类在继承时不继承该 Attribute
```

```
class MyAttrbute : Attribute
{
    public string info;
    public MyAttrbute(string msg)
    {
        info = msg;
    }
    public int vale = 0;
}
```

使用时：

```
[MyAttrbute("xxxxx", vale = 11)]
 class Test{}
```

通过反射获取信息：

```
foreach (var attr in typeof(Test).GetCustomAttributes(true))
  {//获取 Test 类的所有 Attribute，找到 MyAttrbute 进行处理
      MyAttrbute myAt = attr as MyAttrbute;
      if (null != myAt)
      {
          Console.WriteLine(myAt.vale);
      }
  }
```

8.1.8. 函数

8.1.8.1. 引用传递

void fun(ref int x){....}

调用时(ref 也是函数签名的一部分，所以调用时候也要写上。传入的参数必须有初始化[赋值])，class 类型如果只是在函数中改变值，不改变指向，可以直接传递，不用 ref。

int x=10;

fun(ref x);

8.1.8.2. 输出参数

类似于 ref，不同处在于传入的变量可以没有初始化（赋值）。在函数中，要把传入的变量作为未初始化的来处理，也就是在函数中必须要进行初始化（赋值）。

```
void fun(out int x){x = 10;....}

int x;
fun(out x);
```

8.1.8.3. params

可以声明为不确定参数个数(数组类型)，该参数必须总是函数最右边的参数，且只有一个参数可以是数组类型。

下面的函数调用时可以使用：int max=fun(1,3,243,434,12);

```
int fun(params int[] a)
```

8.1.9. 数据类型

8.1.9.1. 用户类型

包括类，结构体，枚举（和 C++完全一样），接口。

除了内存分配，类和结构与 C++完全相同。

类的对象在堆中分配，使用 new 创建（构造函数必须加上（）），当不再使用时，将自动进行垃圾清理。

结构是在栈中分配。属于轻量级快速数据类型。

8.1.9.2. 结构体

结构体是值类型，而类是引用类型。

结构体没有构造函数，也不能在定义变量时赋值。

```
struct data
```

```
{
        public int a ;
        public float b;
 }

data d;
d.a=10;
d.b=1.1f;
```

8.1.9.3. 枚举

```
enum  EE
{
   吃饭,
   睡觉,
}

int index = EE.吃饭.GetHashCode();
string val = EE.睡觉.ToString();
```

8.1.10. 字符串

```
 string s1 = "scca";//”scca”字面值本身就是 string 类型
 string s2 = new string('c', 5);
 char[] a = { 'a', 'b', 'c','\0'};//可以不加\0
 string s3 = new string(a);
========@转译
适合地址的转换
如@”d:\1.avi”
或者 string   addr=”d:\1.avi”;   string str=@addr;
========转换为其他类型
int x = Convert.ToInt32("12");
========其他类型转换为字符串
```

```
int    x=10;
string s=x.ToString();
```

========转换为字符串数组

```
char[] a = s.ToCharArray();
```

========去除开头和结尾多余的空格：

```
s=s.Trim();
```

========加入字符

```
string s = "abc";
```

s=s.PadLeft(5');//可以在 s 前面加入两个空格，因为 s 原长 3，要补足到 5，所以加入 2 个

s=s.PadLeft(5,'_');//加入两个_

========分割字符串

str = str.Substring(pos,length); //从 pos 截取 length 长度（无 length 代表到结尾）

```
string s = "abc|de|fg ";
string[] str = s.Split('|');//用 str.Length 可以求出数组长度
foreach (string i in str)
{
      Console.WriteLine(i);
}
```

========字符串查找

```
bool b=str.Contains("xxx");
```

8.1.11. 字符串格式化

```
string s = string.Format("{0:D3},{1:n2}", 10, 1.23234);
```

//:D3 表示整数至少 3 位，不足高位补 0

//:n2 表示小数点后保留 2 位

:c　货币形式

:x　十六进制

:000.00　　用 0 填充不足的位数，小数点后固定两位

8.1.12. StringBuilder

```
StringBuilder sb = new StringBuilder();//构造函数可以传入初始化需要分配的内存大小
sb.AppendFormat("{0},{1}:", 1, 2);
sb.Append('\n');
sb.ToString();
```

8.1.13. 数组

数组分配于堆中，是引用类型。

8.1.13.1. 定义

```
int[]   a=new int[10];
int[] a =new int[5]{ 34, 12, 32, 43, 32}
int[] a={1,2,3,4,5};
```

二维数组

```
int[,] a = new int[5, 10];//或者int[,] a={ { 3, 4 }, { 3, 2 } };
a[1,2]=5;// 赋值
```

8.1.13.2. 遍历

数组可以用 foreach 循环

```
foreach (int x in array){...}
```

8.1.13.3. 拷贝

将数组 a 的内容拷贝到数组 b，数组 b 接收拷贝的位置是从 b[2]开始

```
a.CopyTo(b, 2);
```

8.1.13.4. 功能函数

int len = a.GetLength(0);//得到数组中元素个数，0代表这是一维数组

或者使用int len=a.Length;

int index = a.GetUpperBound(0);//得到数组最大下标数

8.1.13.5. Linq

```
int[] numbers = {3,5,2,5,12,5,36,75,42};
var qur = numbers.Where((n) => n % 2 == 0).OrderBy((n) => n);
foreach (var i in qur) Console.WriteLine(i + "");

int[] ret = qur.ToArray<int>();
```

8.1.14. 预处理

8.1.14.1. region

当代码比较长，可以定义分组，这样就可以折叠这个区域

```
#region A
.....
#endregion
```

8.1.14.2. define

```
#define De                  //在文件最开头定义一个标志
#undef De   可以进行取消定义
```

8.1.14.3. 条件判断

如果有定义则代表会进行编译，否则会隐藏

```
#if De
            Console.WriteLine("de");
#endif
```

8.1.14.4. 编译消息

在文件中定义后，编译文件会弹出警告或者错误

```
#warning xxx
#error xxx
```

可以修改在编译时弹出的行号，字符串可以修改当前提示的文件名

```
#line 100  "flileName"
```

8.1.14.5. Pragma

```
#pragma warning disable 123

#pragma warning restore 123
```

8.1.15. 修饰符

8.1.15.1. unsafe

修饰在 C#中定义的不安全的上下文，如 C++的指针等。如：

```
public unsafe fun(int *p){    }


unsafe
{
    .....
}
```

属性-》生成-》允许不安全的代码

8.1.15.2. fixed

在不安全代码中固定内存位置

```
fixed (double* pa = &a[0],pb = &b[0]){}
```

8.1.15.3. checked

checked（….）可以检查表达式是否有溢出，如果有溢出，则程序会崩溃。

8.1.15.4. lock

类似于互斥锁。

将语句块标记为临界区，方法是获取给定对象的互斥锁，执行语句，然后释放该锁，lock确保当一个线程位于代码的临界区时，另一个线程不进入临界区。如果其他线程试图进入锁定的代码，将一直等待，直到该对象被释放。

```
object locker = new object();
lock(locker){ ..... }
```

8.1.15.5. case

可以用常数变量作为判断值。必须有break跳转，不支持执行完一个case执行别的case。

可以使用以下方法执行多个case

```
case 2:
case 3:    Console.WriteLine("3");        break;
```

8.1.16. 转换类型

8.1.16.1. is

用于检查操作数类型是否相等或可以转换，使用两个操作数，其结果是布尔值。适合于多态的情形。

```
if(a is ClassA){.....................}
```

8.1.16.2. as

检查操作数的类型是否可以转换或者相等（是由is完成的），如果是，则处理结果是已转换

或已装箱的对象。如果否，则返回null。

```
ClassA    a=new ClassA();
ClassB    b=a as ClassB; //返回值null，不可转换
ClassC    c=a;
ClassA    a2=c as ClassA; //将进行转换
```

8.1.16.3. 重载转换

==========定义

```
class A
{
        public int val=1;
        public static implicit operator B(A a)    //隐式转换
        {
                B b = new B();
                b.val = a.val;
                return b;
        }
}
class B
{
      public long val=1000;
      public static explicit operator A(B b)//显示转换
      {
              A a = new A();
              checked { a.val = (int)b.val; }
              return a;
       }
}
```

==========使用

A a = new A();

B b = a;

B b = new B();

A a = (A)b;

8.1.17. 装箱/拆箱

所有的数据类型都是从 System 命名空间的基类 Object 继承，所以基础或是原始的类型打包为一个对象，称为装箱。相反的处理称为拆箱。

int a1 = 10;　　object obj = a1; //装箱

int a2 = (int)obj;//拆箱

一个 int 值可以被转换为对象，并且能再次转换成 int。

当某种值类型的变量需要被转换为一个引用类型时，便会产生一个对象箱保存该值，拆箱则完全相反。当某个对象箱被转换会其原值类型时，该值从箱中拷贝至适当的存储空间。

因为是装箱到 obj，所以可以做一个 List<object>，在读取的时候，通过使用 is 判断类型再拆箱(或使用 as)，实现不同类型放在同一个 list 中。

8.1.18. 运算符重载

```
class A
{
        public int x =10;
        public static A operator+(A a1, A a2)
        {
                A a = new A();
                a.x = a1.x + a2.x;
                return a;
         }
}
```

可以重载的运算符

一元：+，-,!，~，++，true，false

二元：+，-，*，/,%,&,|,^,<<,>>

比较：==,！=，<,>,<=,>=

重载bool运算符就可以用　if(a1){}

8.1.19. 初始化器

```
class A
{
      public string x { get; set; }
      public int y { get; set; }
}
```

A a = new A { x="a", y=1 };//这种方法可以不使用其他构造函数或者对象来初始化成员
List<A> l=new List<A>{new A{x="a",y=1},new A{x="b",y=2}};//可以替代 l.add 以及赋值的语法

8.1.20. 异常

```
public void fun()
{
       throw new Exception("fun");
}
```

```
try
{
    fun();
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
finally
{
    Console.WriteLine("try 执行结束");
}
```

8.1.21. 索引器

使类可以通过[]语法访问和赋值。

```
class Test
{
        string Content1 = "aaa";
        string Content2 = "bbb";


       public  string this[int index]
        {
            get
            {
                if(index == 1) return Content1;
                if (index == 2) return Content2;
                return "erro";
            }
            set { }
        }
 }



Test test = new Test();
Console.WriteLine(test[1]);
```

8.1.22. 类型获取

```
string s = "";
Type t1 = s.GetType();
Type t2 = Type.GetType("system.string", false, true);
 Type t3 = typeof(string);
```

8.1.23. 反射

8.1.23.1. 类信息

获取类中包含的可以访问的方法

```
Type t = typeof(Cl);
```

```
MethodInfo[] mi = t.GetMethods();

foreach (MethodInfo info in mi)
    Console.WriteLine(info.Name);

//获取单个方法
MethodInfo mi = t.GetMethod("fun1");
```

8.1.23.2. 程序集

获取程序集中包含的类

```
 //Assembly obj = Assembly.Load("");
Assembly obj = Assembly.GetExecutingAssembly();
 Type[] types = obj.GetTypes();

 foreach (Type t in types)      Console.WriteLine(t.Name);
```

8.1.23.3. 加载使用

加载当前程序集中的 Cl 类，void fun1(int i,string s)函数

```
Assembly ass = Assembly.GetExecutingAssembly();
Type t = ass.GetType("ConsoleApplication1.Cl", false, true);
if (null == t) return;
```

```
object obj = Activator.CreateInstance(t);//创建类的实例
MethodInfo mi = t.GetMethod("fun1");
mi.Invoke(obj,new object[]{1,"a"});
```

8.2. file

8.2.1. 命名空间

```
using System.IO;
```

8.2.2. 获取当前路径

```
string path = System.AppDomain.CurrentDomain.BaseDirectory;
```
获取的路径末尾存在一个\

8.2.3. File

静态类

8.2.3.1. 文件复制

```
File.Copy(orgFile, newName);
```

8.2.3.2. 文件否存在

```
bool b=File.Exists(Path);
```

8.2.3.3. 文件删除

```
File.Delete(Path);
```

8.2.4. Directory

静态类

8.2.4.1. 目录是否存在

Directory.Exists(Path);

8.2.4.2. 删除目录

如果使用第二个参数，则只删除非空目录
Directory.Delete(Path,true);

8.2.4.3. 创建目录

可以创建多层次目录
Directory.CreateDirectory(newName);

8.2.5. FileInfo

FileInfo fin = new FileInfo(Path);

8.2.5.1. 文件是否存在

fi.Exists

8.2.5.2. 创建文件

FileStream fs = fi.Create();

8.2.5.3. 时间获取

```
string modiytime =
fin.LastWriteTime.ToLongDateString() + " " + fin.LastWriteTime.ToLongTimeString();
```

8.2.5.4. 删除文件

fin.Delete();

8.2.6. DirectoryInfo

string dir = @"D:\";
@可以将\转换为\\

DirectoryInfo TheFolder = new DirectoryInfo(dir);

属性
TheFolder.Name
TheFolder.FullName

//遍历 dir 中所有目录　　不包含点目录
foreach (DirectoryInfo NextFolder in TheFolder.GetDirectories())

///遍历dir中所有文件
foreach (FileInfo NextFile in TheFolder.GetFiles())
///遍历文件夹下所有jpg文件
foreach (FileInfo NextFile in TheFolder.GetFiles("*.jpg"))

//遍历目录及子目录下所有txt

```
void AddPath(string path, List<string> PathNameList)
{
      DirectoryInfo TheFolder = new DirectoryInfo(path);
      foreach (FileInfo NextFile in TheFolder.GetFiles("*.txt"))
      {
            PathNameList.Add(NextFile.FullName);
      }
      foreach (DirectoryInfo NextFolder in TheFolder.GetDirectories())
      {
           AddPath(path + "\\" + NextFolder.Name, PathNameList);
      }
}
```

8.2.7. FileStream

使用 try 判断文件是否打开或创建成功

8.2.7.1. 读取

```
FileStream file = new FileStream("e:\\1.txt", FileMode.OpenOrCreate);

byte[] data = new byte[10];
file.Read(data, 0, 10);
string s = Encoding.UTF8.GetString(data);
```

8.2.7.2. 写入

```
FileStream file = new FileStream("e:\\1.txt", FileMode.OpenOrCreate);

string s = "abcdefg";
byte[] data =Encoding.UTF8.GetBytes(s);

file.Write(data, 0, data.Length);
```

8.2.7.3. 文件指针移动

```
file.Seek(2, SeekOrigin.Begin);  //文件指针从头向后移动两个
file.Seek(-2, SeekOrigin.End);从末尾向前移动两个位置
```

8.2.8. 文件流操作

8.2.8.1. 文本流

========逐行读取

```
using (StreamReader sr = new StreamReader(new FileStream("D:\\tt.txt", FileMode.Open), Encoding.Default))
{        string strline;
         while((strline = sr.ReadLine() )!= null){}
 }
```

==========保存为指定编码

```
StreamWriter file = new StreamWriter("D:\\.txt", false,Encoding.GetEncoding("gbk"));
file.WriteLine(...);
```

==========追加

追加文本

```
using (StreamWriter sw = File.AppendText("D:\\tt.txt"))
{
      sw.WriteLine("\nxabc");
 }
```

追加字符串，并在一个字符串后加\n

```
  string[] val = new string[2];
  val[0] = "aaa";
  val[1] = "bbb";
  File.AppendAllLines("D:\\1.txt", val, Encoding.Default);
```

8.2.8.2. 二进制流

构造函数传入文件流

```
BinaryReader br = new BinaryReader(new FileStream("D:\\tt.txt",FileMode.Open));
BinaryWriter bw = new BinaryWriter(new FileStream("D:\\tt.txt", FileMode.Create));
```

8.2.9. 读写压缩数据

```
using System.IO;
using System.IO.Compression;


static void SaveCompressdFile(string filename, string data)
{
        FileStream fileStream = new FileStream(filename, FileMode.Create, FileAccess.Write);
        GZipStream compresssionStream = new GZipStream(fileStream,
CompressionMode.Compress);
        StreamWriter writer = new StreamWriter(compresssionStream);
        writer.Write(data);
        writer.Close();
}
static string LoadCompressedFile(string filename)
{
        FileStream fileStream = new FileStream(filename, FileMode.Open, FileAccess.Read);
        GZipStream compressionstream = new GZipStream(fileStream,
CompressionMode.Decompress);
        StreamReader reader = new StreamReader(compressionstream);
        string data = reader.ReadToEnd();
        return data;
```

```
}
```

使用时：

```
SaveCompressdFile("e:\\xx.txt", "abcdefghijklmn");//写入


string s = LoadCompressedFile("e:\\xx.txt");//读取
```

8.2.10. 序列化数据

```
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
==============可以序列化的类
[Serializable]
class A
{
    int x;
    [NonSerialized]
    string age;//不进行序列化

    public A(int a)
    {
        x = a;
    }
}
==============序列化
List<A> data = new List<A>();
data.Add(new A(1));
data.Add(new A(2));

FileStream file = new FileStream("e:\\x.bin", FileMode.Create, FileAccess.Write);
IFormatter serializer = new BinaryFormatter();
serializer.Serialize(file, data);
file.Close();
==============反序列化
FileStream file = new FileStream("e:\\x.bin", FileMode.Open, FileAccess.Read);
IFormatter serializer = new BinaryFormatter();
List<A> data = serializer.Deserialize(file) as List<A>;
file.Close();
```

8.2.11. 监控文件系统

FileSystemWatcher watcher = new FileSystemWatcher();
watcher.Path = "E:\\"; //监控目录为 e 盘，不能写成 e:\\
//watcher.Filter = "*.txt"; //设置监控的文件类型
watcher.EnableRaisingEvents = true;
watcher.Deleted += new ******Tab 键进行插入; //监控删除事件

8.2.12. XML

8.2.12.1. 文件说明

不同于关系型数据库的行列方式，采用类似目录的展开方式（只能有一个根元素）
可以在其中定义任意的标记对,如<x a=”**”>***</x>
在 vs 中，可以将文件另存为别的格式。
=========xml文件声明
<?xml version="1.0" encoding="utf-8"?>
=========引用命名空间
<abc xmlns="**********"> //表示abc及其子元素属于名称空间，名称空间一般是一个地址。
  <a>

</abc>
或者
<abc xmlns:x="urn:schemas-microsoft-com:xml-data">
  <x:a></x:a>
  <x:b></x:b>
</abc>

8.2.12.2. 读写

using System.Xml;

```
XmlDocument doc = new XmlDocument();
doc.Load("config.xml");
```

<config>
  <PicWidth>500</PicWidth>
</config>

```
XmlNode rootNode = doc.SelectSingleNode("config");
```

```
==========访问某节点
string s = rootNode.SelectSingleNode("a").InnerText; //获取节点 a 内的值
//string s = rootNode.InnerText;    //会将根节点中所有节点的的值合并为一个字符串

==========获取节点属性

rootNode.Attributes["属性名"].Value;


==========修改某节点
rootNode.SelectSingleNode("a").InnerText = "xx";
document.Save("D:\\x.xml");


==========删除某节点
rootNode.SelectSingleNode("a").RemoveAll();
document.Save("D:\\x.xml");

==========插入节点
XmlElement node = document.CreateElement("z");
rootNode.AppendChild(node);

rootNode.AppendChild(document.CreateComment("***"));//插入注释

==========遍历根节点下的所有节点
if (rootNode.HasChildNodes)
{
    foreach (XmlNode node in rootNode)
    {
        string str = node.InnerText;
    }
}
```

8.3. thread

using System.Threading;

8.3.1. 运行线程

Thread thread = new Thread(fun);//线程函数，fun 不能有参数

或者使用 lambda 传入函数：new Thread(() => {

Console.WriteLine("xx");

});

thread.IsBackground = true;//前台线程退出则后台线程也退出，如果使用默认的 false，则前台线程退出后，程序也不会退出

thread.Start();//开始运行

8.3.2. 结束线程

thread.Abort();

8.3.3. 线程名称

获取和设置当前线程的名称

Thread.CurrentThread.Name

8.3.4. 暂停当前线程

Thread.Sleep(1000);

8.3.5. 互斥对象

```
Mutex mutex = new Mutex();
```

mutex.WaitOne();//调用多次进行多次上锁，需要多次解锁

.....保护代码.....

mutex.ReleaseMutex();

8.3.6. 事件

public ManualResetEvent manualEvent = new ManualResetEvent(false);//true 代表终止状态，可以被使用,fasle 代表非终止，不可被使用

线程代码（无限循环中，执行一次就暂停，等待信号）

manualEvent.WaitOne();

.....

manualEvent.Reset();

让线程代码运行

manualEvent.Set();

判断线程是否正在运行

manualEvent.WaitOne(0)//返回值为 ture 代表正在被使用

8.4. Generics

//using System.Collections.Generic;

8.4.1. 线型

8.4.1.1. List

======定义

List<string> list=new List<string>();

可以直接用下标访问 list[3]

======遍历

foreach(string x in list)

======排序

```
class Dat
{
   public int a = 0;
}
```

======使用 lambda

```
ll.Sort((Dat x, Dat y) =>
{
    return x.a - y.a;
});
```

======实现接口 IComparable

（右键菜单，选择实现接口，可以自动生成代码）

基本类型已实现该接口

```
class Dat : IComparable
{
      int a = 0;

       int IComparable.CompareTo(Object obj)
       {
              Dat temp = (Dat)obj;
              return a - temp.a;
        }
}

ll.Sort();
```

=====只排序部分元素

实现接口 IComparer

```
class Cmp : IComparer<Stu>
{

    public int Compare(Stu x, Stu y)
    {
        return x.x - y.x;
    }
}
```

从下标 4 的元素开始，一共排序 3 个

```
Cmp cmp = new Cmp();
l.Sort(4, 3, cmp);
```

==========求最大最小值

实现接口 IComparable

```
Stu max = l.Max<Stu>();
Stu min = l.Min<Stu>();
```

========查找

//查找满足某种条件的元素，在函数中写查询条件

```
bool find(Stu st)
{
    return st.x == 10;
}

Stu st = l.Find(find);
if (null != st){}

//也可以使用委托
Stu st = l.Find((Stu temp)=> temp.x == 10);
```

8.4.1.2. ArrayList

存储空间连续
可以存储不同类型元素
当每个元素存储类型相同时，与 list 用法相同

存储元素类型不同时：

```
const int capacity = 100;
```

```
ArrayList ll = new ArrayList(capacity);

 ll.Add(1);
 ll.Add(1.5);
 ll.Add("abc");
ll.Add(new Dat(2));

Console.WriteLine(ll[0]);
Console.WriteLine(ll[1]);
Console.WriteLine(ll[2]);
Dat dd = (Dat)ll[3];
dd.Show();
```

8.4.2. HashSet

可以对数据进行去重，不进行排序
非基本类型需要重载函数

```
class Dat
{
            int a = 0;

            public override bool Equals(object obj)
            {
                return a == ((Dat)obj).a;
            }
            public override int GetHashCode()
            {
                return a;
            }
}
HashSet<Dat> set = new HashSet<Dat>();

set.Add(new Dat(0));
set.Add(new Dat(2));
set.Add(new Dat(3));
set.Add(new Dat(0));
```

8.4.3. queue

```
Queue<int> que = new Queue<int>();

que.Enqueue(1);//向尾部加入元素
```

```
que.Enqueue(2);
que.Enqueue(2);

int x = que.Peek();//查看顶部元素
int y = que.Dequeue();//查看并取出顶部元素
```

8.4.4. 键值对

8.4.4.1. Hashtable

可以存储不同元素，加入数据时，key 不能相同，否则异常

```
const int capacity = 100;
Hashtable ht = new Hashtable(capacity);

ht.Add(0, 1);
ht.Add(1, new Dat(111));
ht.Add(2, 1.5);

Console.WriteLine(ht[0]);
((Dat)ht[1]).Show();
Console.WriteLine(ht[2]);
```

8.4.4.2. Dictionary

```
Dictionary<string, int> dir=new Dictionary<string,int>();
dir.Add("a", 1);
dir.Add("b", 2);

//通过键值查找
if (dir.ContainsKey("a")){int x = dir["a"];}

//遍历
foreach (string key in dir.Keys)
foreach(int value in dir.Values)
foreach (KeyValuePair<string, int> d in dir)

//排序遍历
 foreach (var item in dic.OrderBy(dicItem=> dicItem.Key))
 {//用key进行排序
            MessageBox.Show(item.Value);
  }
```

8.4.4.3. SortedDictionary

8.4.4.4. 排序

对 key 进行排序，key 需要实现 IComparable 接口

```
struct Stu:IComparable
{
        public int a;
        public int b;
        int IComparable.CompareTo(object obj)
        {//从小到大排序
             return a - ((Stu)obj).a;
        }
}
```

8.4.4.5. 获取首部数据

```
using System.Linq;
string val = SortedDictionaryObj.Values.First<string>();
```

8.4.5. 定义泛型

```
class A<S, T>
{
          public S s;
          public T t;
       public void fun(){s = default(S);}//如果是引用类型，则赋值 null，如果是 int 就赋值 0。
 }

A<int, string> a = new A<int, string>();
a.s = 10;
a.t = "xx";
```

8.4.6. 泛型约束

```
class   A<S,T>where S:classX        S 约束为 classX 或者继承自 classX
class   A<S,T>where S:struct         S 被约束为 struct
class   A<S,T>where S:T                把一个类型参数用作另一个的约束
class   A<S,T>where S:classX,classyY
class   A<S,T>where S:classX where T:classY
```

8.4.7. 泛型继承

class B : A<int>
class B<T> : A<T>
将模板实例化后继承或继承模板的类型

8.4.8. 模板泛型

```
delegate void Fun<T>(T t);
void fun(int x) {}

Fun<int> df = new Fun<int>(fun);
df(12);
```

8.5. function

8.5.1. 基本数据类型转换

可以对基本类型做任意转换

```
string s = "12";
int i = Convert.ToInt32(s,10);
```

第二个参数默认 10，代表 10 进制

转为 2 位 16 进制字符串数字

```
int x = 10;
string val  = x.ToString("X2");
```

二进制数据转换为 base64 字符串

```
Convert.ToBase64String(Imgbytes);
```

8.5.2. 字节操作

8.5.2.1. 字节数组拷贝

```
Array.Copy(sourceArray, sourceIndex, destinationArray,
destinationIndex, length);
```

8.5.2.2. 字节流

```
MemoryStream ms = new MemoryStream();
        预先分配指定大小new MemoryStream(1024*1000);

ms.Position = 0;//再次写入时从起始位置开始

//将每次获取的字节放入字节流
byte[] data = new byte[10];
ms.Write(data, 0,len);//ms.Length 获取当前字节流长度

=======将字节流转出
拷贝方式转出
byte[] result = ms.ToArray();
不进行拷贝
sendByte = ms.GetBuffer();
```

8.5.2.3. 转换成数字

```
UInt32 datSize = BitConverter.ToUInt32(recvBytes, POS_DATA_SIZE);
```

转 float

```
BitConverter.ToSingle(bytes, pos);
```

8.5.2.4. 数字转成字节流

```
UInt32 datSize;
BitConverter.GetBytes(datSize);
```

8.5.2.5. 字符解码

```
byte[] data = new byte[1024];
string str = Encoding.Default.GetString(data);//代表系统编码
str = Encoding.UTF8.GetString(data);
```

8.5.2.6. 字符串转字节流

```
byte[] u = Encoding.ASCII.GetBytes(str);
```

8.5.3. 图像操作

```
FileStream fs = new FileStream("D:\\1.jpg", FileMode.Open);
Bitmap bmp = new Bitmap(fs);//new Bitmap("D:\\1.jpg")


Image image = Image.FromFile(imgPathName);
image = Image.FromStream(Stream);
```

8.5.4. 获取图像宽高

```
using (FileStream fs = new FileStream("D:\\1.tif", FileMode.Open, FileAccess.Read))
{
        System.Drawing.Image image = System.Drawing.Image.FromStream(fs);
        int    width = image.Width;
        int    height = image.Height;
}
```

8.5.5. 修改图像质量

```
        public byte[] ChangeImageQuality(byte[] img)
        {
            MemoryStream ms = new MemoryStream(img);
            byte[] buffer = null;

            Bitmap bmp = null;
            ImageCodecInfo ici = null;
            System.Drawing.Imaging.Encoder ecd = null;
            EncoderParameter ept = null;
            EncoderParameters eptS = null;
            try
            {
                bmp = new Bitmap(ms);
                ici = this.getImageCoderInfo("image/jpeg");
                ecd = System.Drawing.Imaging.Encoder.Quality;
                eptS = new EncoderParameters(1);
                ept = new EncoderParameter(ecd, 50L);
                eptS.Param[0] = ept;
                //bmp.Save("D:\\xxxx.jpg", ici, eptS);

                MemoryStream stream = new MemoryStream();
                bmp.Save(stream, ici, eptS);
```

```
                buffer = new byte[stream.Length];
                stream.Seek(0, SeekOrigin.Begin);
                int len = stream.Read(buffer, 0, buffer.Length);
                if (len != buffer.Length) buffer = null;
            }
            catch (Exception ex)
            {
                return buffer;
            }
            finally
            {
                bmp.Dispose();
                ept.Dispose();
                eptS.Dispose();
            }
            return buffer;
        }
        private ImageCodecInfo getImageCoderInfo(string coderType)
        {
            ImageCodecInfo[] iciS = ImageCodecInfo.GetImageEncoders();
            ImageCodecInfo retIci = null;
            foreach (ImageCodecInfo ici in iciS)
            {
                if (ici.MimeType.Equals(coderType))
                    retIci = ici;
            }
            return retIci;
        }
```

8.5.6. 定时器

```
System.Timers.Timer tt = new System.Timers.Timer(1000);
tt.Elapsed += *****
tt.Enabled = true;
```

8.5.7. 类库

使用.NET 中的任何类都是在使用外部程序集中的类，因为它们的处理方式是相同的。

创建类库项目，可以生成 dll，让其他 C#工程调用。

```
namespace ClassLibrary1   //该名称为生成的 dll 名
{
    public class AA
```

```
    {
        public void fun(){}
    }
}
```

在其他工程中，可以在解决方案中，右键引用，选择 dll 所在目录。
使用时：

```
using ClassLibrary1;

AA a = new AA();
a.fun();
```

8.5.8. 调试信息

```
MessageBox.Show(string, "名称");


using System.Diagnostics;

Debug.WriteLine("xx");
```

8.5.9. 时间

```
//using System.Data;

string s=DateTime.Now.ToString("yyyy-MM-dd-HH-mm-ss");


//设置一个 3s 的时间范围

TimeSpan span = new TimeSpan(0, 0, 3);

TimeSpan span = TimeSpan.FromSeconds(3);


int count = (int)span.TotalSeconds;//获取总的秒数
```

8.5.10. 打开/关闭程序

```
using System.Diagnostics;

Process[] process = Process.GetProcessesByName("a");
if (process.GetLength(0) > 0)
{
```

```
        process[0].CloseMainWindow();
        process[0].Close();
}
else
{
    Process.Start("a.exe");
}
```

8.5.11. 打开文件

```
Process.Start(DatSourcePathname);
```

8.5.12. 启动程序

```
Process.Start(new ProcessStartInfo("python.exe", "aa.py"));
```

8.5.13. 退出程序

```
Process.GetCurrentProcess().Kill();
```

8.5.14. Socket

客户端发送，服务端接收

```
//using System.Text;
using System.Net.Sockets;
using System.Net;
using System.Threading;
```

8.5.14.1. 客户端

```
private Thread threadSocket = null;
private Socket sokClient = null;
private string localIP = "127.0.0.1";//服务器地址
private int localPort = 5555;//端口
private string sendStr="";//发送内容

void StartSocket()
{
      IPAddress address = IPAddress.Parse(localIP);
      IPEndPoint endpoint = new IPEndPoint(address, localPort);//把 ip 和端口转化为 IPEndPoint
```

实例

```
    sokClient = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);//创建一个 Socket

    sokClient.Connect(endpoint);//连接到服务器

    byte[] bs = Encoding.ASCII.GetBytes(sendStr);
    sokClient.Send(bs, bs.Length, 0);//发送

}
```

8.5.14.2. 服务端

```
private Thread threadSocket = null;
private Socket socketServer = null;
private string ip = "127.0.0.1";
private int port = 5555;

void Start()
{
    try
    {
        socketServer = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        IPAddress address = IPAddress.Parse(ip);
        IPEndPoint endpoint = new IPEndPoint(address, port);
        socketServer.Bind(endpoint);

        threadSocket = new Thread(StartSocket);
        threadSocket.IsBackground = true;
        threadSocket.Start();
     }
     catch (Exception ex)
     {
     }
}

void StartSocket()
{
            socketServer.Listen(1);
            Socket temp = socketServer.Accept();
```

```
            byte[] recvBytes = new byte[1024];
            int revbytes = temp.Receive(recvBytes, recvBytes.Length, 0);
            string recvStr = Encoding.ASCII.GetString(recvBytes, 0, revbytes);
 }
```

调用时：

```
Start();
```

8.5.15. web

8.5.15.1. GET 请求

```
HttpWebRequest req = (HttpWebRequest)HttpWebRequest.Create(reqstStr);
req.Method = "GET";
req.Headers["diyKey"] = "val";//自定义报头中的 key-val
using (WebResponse wr = req.GetResponse())
{
        Stream responseStream = wr.GetResponseStream();
        StreamReader streamReader = new StreamReader(responseStream, Encoding.UTF8);
        string retString = streamReader.ReadToEnd();
}
```

8.5.15.2. POST 请求

```
string param = string.Format("userName={0}&passWord={1}", user, password);
byte[] bs = Encoding.ASCII.GetBytes(param);

HttpWebRequest req = (HttpWebRequest)HttpWebRequest.Create(reqstStr);
req.Method = "POST";
req.KeepAlive = false;
req.ContentType = "application/x-www-form-urlencoded";
req.ContentLength = bs.Length;
using (Stream reqStream = req.GetRequestStream())
{
      reqStream.Write(bs, 0, bs.Length);
}



using (WebResponse wr = req.GetResponse())
{
        Stream responseStream = wr.GetResponseStream();
        XmlDocument doc = new XmlDocument();
```

```
        doc.Load(responseStream);
}
```

8.5.15.3. HTTP 服务

```
class HttpSer
{
    HttpListener httpPostRequest = new HttpListener();
    public void Start()
    {
        httpPostRequest.Prefixes.Add("http://127.0.0.1:8080/abc/");
        httpPostRequest.Start();

        Thread ThrednHttpPostRequest = new Thread(httpPostRequestHandle);
        ThrednHttpPostRequest.IsBackground = true;
        ThrednHttpPostRequest.Start();
    }
    void httpPostRequestHandle()
    {
        while (true)
        {
            HttpListenerContext requestContext = httpPostRequest.GetContext();

            Thread threadsub = new Thread(() =>
            {
                HttpListenerContext request = (HttpListenerContext)requestContext;
                string requestMethod = request.Request.HttpMethod;

                if (requestMethod == "GET")
                {
                    string val = request.Request.QueryString["user"];
string requestKeyVal = request.Request.Headers["diyKey"];//获取自定义报头中的 key-val
                }
                else
                {
                    string requestDatType = request.Request.ContentType;
                    Stream SourceStream = request.Request.InputStream;
                    if (requestDatType == "text/plain")
                    {
                        StreamReader  streamReader  =  new  StreamReader(SourceStream,
Encoding.UTF8);
                        string retString = streamReader.ReadToEnd();
                    }
                    else if(requestDatType == "image/jpeg")
```

```
                    {
                        int requestDatLen = (int)request.Request.ContentLength64;

                        BinaryReader br = new BinaryReader(SourceStream);
                        byte[] img = new byte[requestDatLen];
                        int revLen = 0;
                        while (revLen < requestDatLen)
                        {
                            int read = br.Read(img, revLen, requestDatLen - revLen);
                            revLen += read;
                        }
                    }
                }


                request.Response.StatusCode = 200;
                request.Response.Headers.Add("Access-Control-Allow-Origin", "*");
                request.Response.ContentType = "application/json";
                request.Response.ContentEncoding = Encoding.UTF8;
                byte[] buffer = System.Text.Encoding.UTF8.GetBytes("success");
                request.Response.ContentLength64 = buffer.Length;
                var output = request.Response.OutputStream;
                output.Write(buffer, 0, buffer.Length);
                output.Close();
            });
            threadsub.IsBackground = true;
            threadsub.Start();
        }
    }
}
```

8.5.16. 发邮件

```
public void SendMail()
{
    System.Net.Mail.MailMessage msg = new System.Net.Mail.MailMessage();
    msg.To.Add("jiyanglin@roadmaint.com");
    // msg.To.Add("a@a.com"); 可以发送给多人
    //msg.CC.Add(c@c.com);  可以抄送给多人

    String sendMailAddr = "jiyanglin@qq.com";
    String sendName = "jiyanglin";
    msg.From = new System.Net.Mail.MailAddress(sendMailAddr, sendName,
System.Text.Encoding.UTF8);
```

```
            msg.Subject = "我是标题";
            msg.SubjectEncoding = System.Text.Encoding.UTF8;
            msg.Body = "邮件内容";
            msg.BodyEncoding = System.Text.Encoding.UTF8;
            msg.Attachments.Add(new System.Net.Mail.Attachment("D:\\123-纪阳林.xlsx"));
            msg.IsBodyHtml = false;
            msg.Priority = System.Net.Mail.MailPriority.High;//邮件优先级

            System.Net.Mail.SmtpClient client = new System.Net.Mail.SmtpClient();
            client.Credentials = new System.Net.NetworkCredential("jiyanglin@qq.com",
"**********");
            client.EnableSsl = true;
            client.Host = "smtp.qq.com";
            object userState = msg;
            try
            {
                //client.SendAsync(msg, userState);
                client.Send(msg);
                MessageBox.Show("发送成功");
            }
            catch (System.Net.Mail.SmtpException ex)
            {
                MessageBox.Show(ex.Message, "发送邮件出错");
            }
        }
```

8.5.17. 串口

```
using System.IO.Ports;
```

类内变量：

```
SerialPort serialPort1 = new SerialPort("COM2", 9600, Parity.None, 8, StopBits.One);
```

8.5.17.1. 打开

```
serialPort1.Open();
```

8.5.17.2. 关闭

```
serialPort1.Close();
```

8.5.17.3. 发送

======发送字符串

```
serialPort1.Write("xx");
```

======直接发送16进制byte数组

```
byte[] send = new byte[3];
send[0] = 0xae;
send[1] = 0x05;
send[2] = 0x12;

serialPort1.Write(send, 0, 3);
```

======将16进制字符串转换为byte数组发送

```
private static byte[] strToToHexByte(string hexString,ref int size)
{
        hexString = hexString.Replace(" ", "");
        if ((hexString.Length % 2) != 0)
                hexString += "";
        byte[] returnBytes = new byte[hexString.Length / 2];
        int i = 0;
        for (; i < returnBytes.Length; i++)
            returnBytes[i] = Convert.ToByte(hexString.Substring(i * 2, 2), 16);
        size = i;
        return returnBytes;
}

int size = 0;
byte[] send = strToToHexByte(str,ref size);//str为ae 01 03这种
serialPort1.Write(send, 0, size);
```

8.5.17.4. 接收

```
serialPort1.DataReceived+=new SerialDataReceivedEventHandler(serialPort1_DataReceived);
/////////对应函数
private void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
      string s = serialPort1.ReadExisting();//接收串口的字符串
    //throw new NotImplementedException();自动生成的函数需要将这句注释掉，否则运行该句后程序会退出
}
```

十六进制接收：

```
public string byteToHexStr(byte[] bytes)
{
        string returnStr = "";
        if (bytes != null)
        {
              for (int i = 0; i < bytes.Length; i++)
              {
                    returnStr += bytes[i].ToString("X2");
               }
         }
         return returnStr;
 }
```

使用时

```
int DataCount = serialPort1.BytesToRead;
byte[] ReCMD = new byte[DataCount];
serialPort1.Read(ReCMD, 0, DataCount);
string RecStr = byteToHexStr(ReCMD);
```

8.5.18. 正则表达式

```
using System.Text.RegularExpressions;

==========字符条件判断
 string str = "scca";

 Regex reg = new Regex("^s.*a$");
 if (reg.IsMatch(str))
 {
     Console.WriteLine("满足条件");
 }


                Regex r = new Regex("abc");
                MatchCollection matches = r.Matches("123abc45abcd");
                foreach (Match match in matches)
                {
                      Console.WriteLine("{0} found at pos {1}", match.Value, match.Index);
                      string str = match.Result("$&_r");//将匹配结果加上_r 返回字符串
                }
==========匹配内容
                var input = "this _is#  a _big _apple#";
                var pattern = @"_(\w+)#";//\w+表示一个单词(连续非特殊字符)      匹配句子中所
```

有以_开头#结尾的单词　结果为 is 和 apple

```
Match match = Regex.Match(input, pattern);

while(match.Success)
{
    Console.WriteLine(match.Groups[1].Value);

    match = match.NextMatch();
}
```

```
string input = "Born: July 12, 1010";
//有三个括号，所以匹配出来后有三组外加总体的匹配字符
string pattern = @"(\w+)\s(\d{1,2}),\s(\d{4})";

Match match = Regex.Match(input, pattern);
if (match.Success)
{
    for (int ctr = 0; ctr < match.Groups.Count; ++ctr )
    {
        Console.WriteLine("group {0}: {1}", ctr, match.Groups[ctr].Value);
    }
}
```

==========替换

```
//将匹配出的 12.34 前加上价格
string pattern = @"\b\d+\.\d{2}\b";
//\b 代表单词边界　\d+代表多个数字　\.代表.　d{2}代表两个数字
//匹配带小数点两位的数字

string replacement = "价格$&";//$&代表匹配上的内容 $是特殊转义字符 输出$用$$

string input = "apple: 12.34  banana: 6.5";
Console.WriteLine(Regex.Replace(input, pattern, replacement));
```

==========拆分

```
string input = "1.aa  2.bb  3.cc";
string pattern = @"\b\d+\.";//匹配序号加.

foreach(string item in Regex.Split(input,pattern))
{
    if (!String.IsNullOrEmpty(item))
        Console.WriteLine(item);
}
```

8.5.19. LINQ

实现 IEnumerable 的类都可以用 linq

==========LINQ 查询

```
string[] arry = { "a1","a2","b1","b2","c"};
var query = from n in arry where n.StartsWith("a") select n;//LINQ 语句
string str="";
foreach(var s in query){str += s;}
```

from n in arry 类似 foreach

where 后是限定选择的条件

select 字句是必须的，指定结果集中包含那些元素。

```
  int[] numbers = {3,5,2,5,12,5,36,75,42};
 var qur = from num in numbers where num % 2 == 0 orderby num descending select num;
//只有调用语句才会运行

//var qur = numbers.Where(n => n % 2 == 0).OrderBy(n => n);
   foreach (var i in qur) Console.WriteLine(i + "");
```

```
int[] ret = qur.ToArray<int>();
```

==========方法语句

```
var query = arry.Where(n=>n.StartsWith("a"));
var query = arry.Where(n=>n.Length==1);
```

==========排序

```
var query = from n in arry where n.StartsWith("a") orderby n select n;
var query = arry.OrderBy(n=>n).Where(n=>n.StartsWith("a"));

var query = arry.OrderBy(n => n.Substring(n.Length - 1)).Where(n=>n.StartsWith("a"));//排序时，
```

按照最后一个字符排

==========聚合运算符

可以对查询结果进行分析。

```
int count=query.Count();
```

==========let

```
                string[] strs = { "aA-11", "bb-22","cc" };

                var qur = from s in strs

                          let words = s.Split('-')

                          from word in words
```

```
                                let w = word.ToUpper()
                                select w;

                        foreach (var s in qur)
                                Console.WriteLine("{0}",s);
```

==========分组与表连接

```
    class Customer
     {
          public string Name { get; set; }
          public string City { get; set; }
     }
     class Employee
     {
          public string Name { get; set; }
          public int ID { get; set; }
 }

List<Customer> cust = new List<Customer>();
cust.Add(new Customer() { Name="aa", City = "beijing" });
cust.Add(new Customer() { Name = "bb", City = "beijing" });
cust.Add(new Customer() { Name = "cc", City = "xian" });

List<Employee> em = new List<Employee>();
em.Add(new Employee() { Name = "aa", ID = 1 });
em.Add(new Employee() { Name = "dd", ID = 2 });
```

==========分组查询

```
 var quer = from c in cust
 group c by c.City;
```

```
foreach (var cg in quer)
 {
            Console.WriteLine(cg.Key);
            foreach (var c in cg)       Console.WriteLine(" " + c.Name);
}
```

```
            var quer = from c in cust
                       group c by c.City into custGroup
                       where custGroup.Count() >= 2
                       select new { City = custGroup.Key, number = custGroup.Count() };

            foreach (var c in quer)
                Console.WriteLine("{0} {1}", c.City,c.number);
```

```
============表连接
  var queryJoin = from c in cust
                          join e in em on c.Name equals e.Name
                select new { PersonName = c.Name, PersonID = e.ID, PersonCity = c.City };

foreach (var p in queryJoin)
        Console.WriteLine("{0} {1} {2}", p.PersonName, p.PersonID, p.PersonCity);
```

8.5.20. 获取主板信息

```
System.Management;

      ManagementClass mc = new ManagementClass("Win32_BaseBoard");
```

```
ManagementObjectCollection moc = mc.GetInstances();
string strID = null;
foreach (ManagementObject mo in moc)
{
    strID = mo.Properties["SerialNumber"].Value.ToString();
    break;
}
```

8.5.21. 打开文件夹对话框

wpf 需要添加 dll 引用（解决方案文件浏览中引用目录）

```
System.Windows.Forms.FolderBrowserDialog fb = new System.Windows.Forms.FolderBrowserDialog();
fb.Description = "选择目录";
if (System.Windows.Forms.DialogResult.OK == fb.ShowDialog())
{
    string path = fb.SelectedPath;
}
```

8.5.22. 打开文件对话框

```
var of = new Microsoft.Win32.OpenFileDialog() {  Filter = "jpg|*.jpg||"  };
if (true == of.ShowDialog())
{
    string file = of.FileName;
}
```

8.5.23. 限制输入数字

```
lvds1.PreviewTextInput += new TextCompositionEventHandler(EditPreviewTextInput);

void  EditPreviewTextInput(object sender, TextCompositionEventArgs e)
{
    e.Handled = false;
```

```
        if (!isNumberic(e.Text))  e.Handled = true;
    }
    public bool isNumberic(string _string)
    {
        if (string.IsNullOrEmpty(_string))
            return false;
        foreach (char c in _string)
        {
            if (!char.IsDigit(c))
                return false;
        }
        return true;
    }
```

8.5.24. 嵌入资源

在项目 dirname 目录下存在 filename.txt，生成操作设置为嵌入的资源

```
string resourceName = MethodBase.GetCurrentMethod().DeclaringType.Namespace + ".dirname.filename.txt";
Stream stream = Assembly.GetExecutingAssembly().GetManifestResourceStream(resourceName);
StreamReader sr = new StreamReader(stream,Encoding.Default);
string strline = sr.ReadToEnd();
```

8.5.25. Resource

工程 WpfApp1 加入目录 dirname，复制到输出目录：不复制，生成操作：Resource

代码中可以"/WpfApp1;component/dirname/文件名"进行引用（用于图像内容的 url）

```
img.Source = new BitmapImage(new Uri("/prjName;component/dirname/1.jpg", UriKind.RelativeOrAbsolute));
```

8.5.26. 环境变量

加入临时环境变量

```
string dllDirectory = System.AppDomain.CurrentDomain.BaseDirectory + "dirname\\dir";
Environment.SetEnvironmentVariable("PATH", Environment.GetEnvironmentVariable("PATH") + ";" + dllDirectory);
```

8.6. c++mix

8.6.1. C++调用 C#

c++工程必须 clr 支持
导出类和函数必须 public

```
#using "Csharp.dll"
using namespace AddSpace;
Add^ add = gcnew Add();
add.fun();
```

捕获c#异常

```
try{ ....}
catch (System::Exception^ e){}
```

托管类不能作为类成员或全局变量
可以通过在类内创建类本身的全局变量进行全局使用

```
public class AA
{
        public static AA obj = new AA();

        public void run(string str){.....}
}
```

c++调用时：

```
AA::obj->run(str)
```

8.6.2. CLR 类库

建立 C++CLR 类库，该类库生成的 dll，可以直接在 c#中添加引用后使用该命名空间内的类。
与 C#交互的函数中不能包含 c#不支持的类型，否则在 C#中不会提示相应函数

字符串传递

```
String^ Fun()
{
     CString ss = L"吃饭睡觉";
     String^ str = gcnew String(ss.GetBuffer());
```

```
    return str;
}

void Fun(String^ str)
{
   CString ss(str);

char*                                    ch2                                    =
(char*)(void*)System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi(str);
```

8.6.3. 数据类型

C/C++	C#	长度
short	short	2
int	int	4
long	int	4
bool	bool	1
char	byte	1
wchar_t	char	2
float	float	4
double	double	8

8.6.4. 调用方式

```
[DllImport("**.dll", EntryPoint = "fun", CallingConvention = CallingConvention.StdCall,
CharSet = CharSet.Auto)]
public static extern void fun();
```

using System.Runtime.InteropServices;

导入的函数中，字符串类型都用 string，数字可以用 uint、long 等。

句柄可以用 uint 或者 IntPtr

using System.Windows.Interop;
IntPtr hwnd = new WindowInteropHelper(this).Handle;

8.6.5. c++回调 c#

```
typedef void (__stdcall *RR)(int x);
extern "C" __declspec(dllexport)  void  fun(RR rr)
```

```
public delegate void RR(int x);
[DllImport("Dll1.dll", EntryPoint = "fun") ]
public static extern void fun(RR rr);

 fun((int xx)=> { Console.WriteLine(xx); });
```

8.6.6. 字符串传递

```
extern "C" __declspec(dllexport) void __stdcall show(wchar_t *str)

public static extern unsafe void show(char* str);
unsafe
{
    string str = "abc 吃饭 def";
    fixed (char* p = &(str.ToCharArray()[0]))
    {
        show(p);
    }
}




extern "C" __declspec(dllexport) void __stdcall getStr(char *outStr)
{
     strcpy_s(outStr, MAX_PATH, "abc 吃饭 def");
}

public static extern unsafe void getStr(byte* outStr);
unsafe
{
    byte[] str = new byte[260];
    fixed (byte *p = &str[0])
    {
       getStr(p);
     }
     string ret = Encoding.Default.GetString(str);
     int pos = ret.IndexOf('\0');
     ret = ret.Substring(0, pos);
}
```

8.6.7. 数组传递

```
extern "C" __declspec(dllexport) void __stdcall showArry(float *pDat,int len)


public static extern unsafe void showArry(float* pDat, int len)
float[] dat = new float[3] { 1.1f, 2.2f, 3.3f };
fixed (float* parry = &dat[0])
{
     showArry(parry, dat.Length);
}
```

8.6.8. 结构体传递

```
struct INFO
{
    int    aa;   //结构体对齐 8
    double  bb;
    char  cc[100];
};
void __stdcall testrun(INFO *info)
{
    info->aa = 1;
    info->bb = 2.2;
    strcpy_s(info->cc, "吃饭");
}

=======调用方式 1
[StructLayout(LayoutKind.Sequential)]
public struct INFO
{
       public int aa;
       public double bb;
       [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 100)]
        public string cc;
}
 [DllImport("AA.dll", EntryPoint = "testrun")]
 public static extern void testrun(ref INFO info);


INFO info = new INFO();
testrun(ref info);
```

=======调用方式 2

```
[DllImport("AA.dll", EntryPoint = "testrun")]
public static extern void testrun(IntPtr pv);


IntPtr pv = Marshal.AllocHGlobal(8+8+100);
testrun(pv);
int aa = Marshal.ReadInt32(pv, 0);
byte[] b = new byte[8];
for(int i = 0; i < 8;++i)
      b[i] = Marshal.ReadByte(pv, 8+i);
double bb = (double)BitConverter.ToDouble(b,0);
string cc =  Marshal.PtrToStringAnsi(pv + 8+8);

Marshal.FreeHGlobal(pv);
```

8.6.9. 内存拷贝

```
IntPtr datpr = datptr;
Marshal.Copy(dat, 0, datpr,dat.Length);
```

8.6.10. WIN32DLL

8.6.10.1. 分配控制台

```
private const string Kernel32_DllName = "kernel32.dll";
[DllImport(Kernel32_DllName)]
private static extern bool AllocConsole();
```

8.6.10.2. 隐藏鼠标

```
[DllImport("user32.dll", EntryPoint = "ShowCursor", CharSet = CharSet.Auto)]
//或者 CharSet.Unicode
public static extern int ShowCursor(bool bShow);
```

使用时：

```
ShowCursor(false);
```

8.6.10.3. 读写 ini

```
[DllImport("kernel32")]
private static extern long WritePrivateProfileString(string section, string key, string val, string filePath);
[DllImport("kernel32")]
private static extern int GetPrivateProfileString(string section, string key, string def, StringBuilder retVal, int size, string filePath);
```

写入：

```
WritePrivateProfileString("section", "key", "xx", ".\\1.ini");
```

读取：

```
StringBuilder temp = new StringBuilder(1024);
string result;//接收用的字符串
if (0 == GetPrivateProfileString("section"," key", "", temp, 500, ".\\1.ini"))
{
        result = null;
}
else
{
        result = temp.ToString();
}
```

8.6.10.4. 发送消息

```
[DllImport("User32.dll")]
private static extern uint SendMessage(uint hWnd, uint Msg,  uint wParam,  uint lParam);
[DllImport("User32.dll")]
private static extern uint FindWindow(string className, string windowName);

uint  hWnd=FindWindow(null, "name");
if (hWnd >0)
{
    SendMessage(hWnd, 0x0400 + 1, 0, 0);//代表 WM_USER
}
```

8.6.10.5. 关机

```
[DllImport("user32.dll")]
static extern bool ExitWindowsEx(uint uFlags, uint dwReason);
```

ExitWindowsEx(0x01,0); //注销:0x00 关机：0x01 重启：0x02

8.7. openXML

8.7.1. Word

8.7.1.1. 安装

https://msdn.microsoft.com/en-us/library/office/cc850833.aspx
OpenXMLSDKv2.msi
OpenXMLSDKTool.msi

可以不安装，添加引用即可:
C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.0\WindowsBase.dll
C:\Program Files (x86)\Open XML SDK\V2.0\lib\DocumentFormat.OpenXml.dll

8.7.1.2. 创建

会和 System.Windows.Documents 重名

```
using DocumentFormat.OpenXml;
using DocumentFormat.OpenXml.Wordprocessing;
using DocumentFormat.OpenXml.Packaging;

using OpenXmlParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;
using OpenXmlWordRun = DocumentFormat.OpenXml.Wordprocessing.Run;
```

```
WordprocessingDocument doc = WordprocessingDocument.Create(pathName,
WordprocessingDocumentType.Document);
MainDocumentPart mainPart = doc.AddMainDocumentPart();
mainPart.Document = new Document();
Body body = mainPart.Document.AppendChild(new Body());

//....写入段落
OpenXmlWordRun run =  new OpenXmlWordRun(new Text(str));
```

```
OpenXmlParagraph paragraph = body.AppendChild(new OpenXmlParagraph());
paragraph.AppendChild(run);
//....写入

doc.Close();
```

8.7.1.3. 打开

```
WordprocessingDocument.Open(filePath, true)
```

8.7.1.4. 文字格式

```
 RunFonts fonts = new RunFonts() { EastAsia = "黑体" };
 FontSize size = new FontSize() { Val = "70" };
DocumentFormat.OpenXml.Wordprocessing.Color color = new
DocumentFormat.OpenXml.Wordprocessing.Color() { ThemeColor = ThemeColorValues.Accent2};


//或使用 runProperties.Append(color);  runProperties.Append(size);
RunProperties runProperties = new RunProperties(color,size,fonts);

Text txt = new Text("word 文字");
OpenXmlWordRun run = new OpenXmlWordRun(runProperties, txt);


OpenXmlParagraph paragraph = body.AppendChild(new OpenXmlParagraph());
paragraph.AppendChild(run);
```

8.7.1.5. 插入图像

```
using A = DocumentFormat.OpenXml.Drawing;
using DW = DocumentFormat.OpenXml.Drawing.Wordprocessing;
using PIC = DocumentFormat.OpenXml.Drawing.Pictures;

 public void AddPictureIntoWord(string docfilePath, string picturePath)
{
            using (WordprocessingDocument doc = WordprocessingDocument.Open(docfilePath, true))
            {

                ImagePartType imagePartType = ImagePartType.Jpeg;
```

```
                //string picType = picturePath.Split('.').Last();
                //if (!Enum.TryParse<ImagePartType>(picType, true, out imagePartType)) return;  // 通过后缀名判断图片类型, true 表示忽视大小写

                ImagePart imagePart = doc.MainDocumentPart.AddImagePart(imagePartType);
                imagePart.FeedData(File.Open(picturePath, FileMode.Open)); // 读取图片二进制流
                AddImageToBody(doc, doc.MainDocumentPart.GetIdOfPart(imagePart));
            }
        }



    private void AddImageToBody(WordprocessingDocument wordDoc, string relationshipId)
        {
            // Define the reference of the image.
            var element =
                 new Drawing(
                     new DW.Inline(
                         new DW.Extent() { Cx = 990000L, Cy = 792000L }, // 调节图片大小
                         new DW.EffectExtent()
                         {
                             LeftEdge = 0L,
                             TopEdge = 0L,
                             RightEdge = 0L,
                             BottomEdge = 0L
                         },
                         new DW.DocProperties()
                         {
                             Id = (UInt32Value)1U,
                             Name = "Picture 1"
                         },
                         new DW.NonVisualGraphicFrameDrawingProperties(
                             new A.GraphicFrameLocks() { NoChangeAspect = true }),
                         new A.Graphic(
                             new A.GraphicData(
                                 new PIC.Picture(
                                     new PIC.NonVisualPictureProperties(
                                         new PIC.NonVisualDrawingProperties()
                                         {
```

```
                                        Id = (UInt32Value)0U,
                                        Name = "New Bitmap Image.jpg"
                                    },
                                    new
PIC.NonVisualPictureDrawingProperties()),
                                new PIC.BlipFill(
                                    new A.Blip(
                                        new A.BlipExtensionList(
                                            new A.BlipExtension()
                                            {
                                                Uri =

"{28A0092B-C50C-407E-A947-70E740481C1C}"
                                            })
                                    )
                                    {
                                        Embed = relationshipId,
                                        CompressionState =
                                        A.BlipCompressionValues.Print
                                    },
                                    new A.Stretch(
                                        new A.FillRectangle())),
                                new PIC.ShapeProperties(
                                    new A.Transform2D(
                                        new A.Offset() { X = 0L, Y = 0L },
                                        new A.Extents() { Cx = 990000L, Cy =
792000L }), //与上面的对准
                                    new A.PresetGeometry(
                                        new A.AdjustValueList()
                                    ) { Preset = A.ShapeTypeValues.Rectangle }))
                            )                   {                   Uri                   =
"http://schemas.openxmlformats.org/drawingml/2006/picture" })
                        )
                        {
                            DistanceFromTop = (UInt32Value)0U,
                            DistanceFromBottom = (UInt32Value)0U,
                            DistanceFromLeft = (UInt32Value)0U,
                            DistanceFromRight = (UInt32Value)0U,
                            EditId = "50D07946"
                        });

            // Append the reference to body, the element should be in a Run.
            wordDoc.MainDocumentPart.Document.Body.AppendChild(new     Paragraph(new
Run(element)));
```

```
        }
```

8.7.2. Excel(EPPlus.dll)

8.7.2.1. 创建/打开

```
ExcelPackage _pakage = new ExcelPackage(new FileInfo(excelFilePathName));
```

8.7.2.2. 选择 sheet 页

```
ExcelWorksheet _curSheet = _pakage.Workbook.Worksheets[1];
_curSheet = m_pakage.Workbook.Worksheets[sheetName];
```

8.7.2.3. 创建 sheet 页

```
_curSheet = _pakage.Workbook.Worksheets.Add(sheetName);
```

8.7.2.4. 保存

```
_pakage.Save();
```

8.7.2.5. 数据操作

8.7.2.5.1. 获取行列数

```
int rowMax = _curSheet.Dimension.End.Row;
int colMax = _curSheet.Dimension.End.Column;
```

8.7.2.5.2. 获取单元格数据

```
string val = (string)_curSheet.GetValue(row, col);
if (null == val) val = "";
```

8.7.2.5.3. 设置单元格数据

```
_curSheet.SetValue(row, col, val);
```

8.7.2.6. 格式

8.7.2.6.1. 网格线条

_curSheet.View.ShowGridLines = false；

8.7.2.6.2. 合并单元格

_curSheet.Cells[row1, col1, row2, col2].Merge = true;

8.7.2.6.3. 对齐方式

_curSheet.Cells[row, col].Style.HorizontalAlignment = ExcelHorizontalAlignment.Left

8.7.2.6.4. 字体颜色

_curSheet.Cells[row, col].Style.Font.Color.SetColor(System.Drawing.Color.FromArgb(r, g, b));

8.7.2.6.5. 单元格背景色

```
 _curSheet.Cells[row, col].Style.Fill.PatternType = ExcelFillStyle.Solid;
 _curSheet.Cells[row, col].Style.Fill.BackgroundColor.SetColor(Color.FromArgb(r, g, b));
```

8.7.2.6.6. 超链接

```
                m_curSheet.Cells[row, col].Hyperlink = new ExcelHyperLink(path, UriKind.Relative);
                 m_curSheet.Cells[row,
col].Style.Font.Color.SetColor(System.Drawing.Color.FromArgb(0, 0, 255));
                 m_curSheet.Cells[row, col].Style.Font.UnderLine = true;
```

sheet 页链接：
curSheet.Cells[row, col].Hyperlink = new ExcelHyperLink("sheet 名!A1", "显示文字");

8.7.2.6.7. 单元格大小

_curSheet.Column(col).Width = 10；

第九章 winform

9.1. 窗体

9.1.1. 创建

右键工程，”添加”->”windows 窗体”。（假设 Name 为 form2）

```
Form2   f=new Form2();
f.Show(); //第一次调用该函数时，窗体才会创建
f.Hide();
```

窗口必须在 ui 线程创建

ff.Show();　　非模式对话框，需要在 ui 线程创建

ff.ShowDialog();　模式对话框，自带 ui 线程

9.1.2. 屏蔽关闭按钮

override 窗口的 OnFormClosing 事件

```
e.Cancel = true;
this.Hide();
```

可以使 alt+F4 与关闭按钮都失效，并隐藏窗体

9.1.3. 屏蔽最大化按钮

```
this.MaximizeBox = false;
```

9.1.4. 屏蔽所有系统按钮

```
this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.SizableToolWindow;
```

最大化、最小化、关闭按钮都消失，alt+f4 失效，上面出现一个红色×用来隐藏窗口

9.2. 定时器

资源中，所有 windows 窗体内，可以找到定时器控件，将控件加入到窗体。在定时器控件的属性设置中，Name 代表定时器的名字（如 timer1），Interval 代表定时器的间隔时间（1000 代表 1s），在属性的事件里，可以设置定时器响应函数的名字，并且回车生成函数，也可以直接点击定时器，生成并进入函数。使用时，timer1.Start();开启定时器，timer1.Stop();取消定时器。

9.3. 屏幕大小

```
//获取工作区大小
Rectangle rect = new Rectangle();
rect = Screen.GetWorkingArea(this);

///获取整个屏幕大小，如果需要全屏，设置 ResizeMode 为 NoResize;
Rectangle rect = System.Windows.Forms.SystemInformation.VirtualScreen;
```

9.4. 控件大小/位置

```
this.**.Left = 0;
this.**.Top = 0;
this.**.Width = 1000;
this.**.Height = 500;
```

9.5. 消息

9.5.1. 截获消息

```
protected override void DefWndProc(ref Message m)
{
        switch (m.Msg)
        {
                case 0x0400 + 1:

                        break;
                default:
```

```
                base.DefWndProc(ref m);
                break;
        }
}
```

9.5.2. 鼠标消息

```
string s = string.Format("{0},{1}", e.X, e.Y);
MessageBox.Show(s, "鼠标位置"); //弹出消息盒子
```

9.5.3. 键盘消息

```
if (e.KeyCode == Keys.A && e.Control){    同时按下 control 和 a    }
```

Keys 类定义了所有的按键，对于 control，alt 这类的键，只能用 e.Control 这种布尔值进行判断，不能用 KeyCode 和 Keys 进行比较。

9.6. 窗口设置

设置属性即可

9.6.1. 顶层窗口

TopMost

9.6.2. 去除标题和边框

WindowStyle none
ResizeMode NoResize

9.6.3. 全屏

WindowState Maximized

9.6.4. 设置窗口位置

this.Left = 0;
this.Top = 0;
this.Width = 1000;
this.Height = 500;

9.7. 控件

9.7.1. 线程中访问控件

```
delegate void SetTextBoxCallBack(string txt);
void SetTextBox(string txt){this.textBox1.Text = txt;}
```

线程中使用：

```
this.Invoke(new SetTextBoxCallBack(SetTextBox), new object[] { i.ToString() });
```

9.7.2. Anchor

该属性可以让控件在保持与边界的位置，即设置了四个方向，则窗口拉伸时，控件会按照比例进行拉伸。

9.7.3. 可见性

**.Visibility = System.Windows.Visibility.Hidden;

9.7.4. Button

可以在属性中设置图案

在消息响应函数中可以使用参数获取对象
((Button)sender).Text = "xx";

动态创建

```
Button btn = new Button();
btn.Text = "xx";
btn.Click += new EventHandler(btn_Click);
Controls.Add(btn);
```

9.7.5. Label

this.label1.Content= "xx";

label1 是在窗体中的 Label 控件的名称（可以通过属性 Name 进行修改）

9.7.6. TextBox

string s=this.textBox1.Text;

string s = "xx";

this.textBox1.AppendText(s);

9.7.7. ListBox

listBox1.Items.Add(s);

listBox1.SelectedIndex = 1;

9.7.8. listView

9.7.8.1. 创建

```
listView1.GridLines = true;
listView1.FullRowSelect = true;
listView1.View = View.Details;
listView1.Scrollable = true;
listView1.MultiSelect = true;
listView1.BackColor = Color.HotPink;//背景颜色
listView1.ForeColor = Color.Yellow;//文字颜色

//创建表头
listView1.Columns.Add("ID", 100, HorizontalAlignment.Center);
listView1.Columns.Add("Name", 100, HorizontalAlignment.Center);


//加入一行数据
```

```
ListViewItem item = new ListViewItem();
item.SubItems.Clear();
//ID 列
item.UseItemStyleForSubItems = false;
item.SubItems[0].Text = "0";
item.SubItems[0].BackColor = Color.Red;
//Name 列
item.SubItems.Add(new ListViewItem.ListViewSubItem());
item.SubItems[1].Text = "ABC";
item.SubItems[1].BackColor = Color.Green;
//数据加入控件
listView1.Items.Add(item);


//加入一行数据
listView1.Items.Add(new ListViewItem(new string[] { "1","DEF"}));
```

9.7.8.2. 删除

```
//删除第一行数据
listView1.Items[0].Remove();
//删除所有数据
while (listView1.Items.Count != 0)
{
    listView1.Items[0].Remove();
}
```

9.7.9. pictureBox

```
pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;

//pictureBox1.Load("D:\\1.jpg");
pictureBox1.Image = Bitmap ;
```

9.7.10. progress

```
progressBar1.Value = 100;//最大位置为 100
```

9.7.11. menuStrip

点击栏目，可以键入文字，下拉可以选择键入类型（如分割线等）。
在键入时，&可以增加 alt 的热键。
每个栏目都有属性，可以在属性中 ShortCutKeys 中增加快捷键，在事件中增加 click 属性可以添加回调函数。

9.7.12. 工具栏

9.7.12.1. ToolStrip

选择该控件，会在控件右上角出现一个小箭头，可以点击出一个菜单：选择插入标准项，可以插入打开，打印等标准的图标。选择编辑项，可以打开一个对话框，可以在其中为工具栏增加项目，包括按钮，文字等类型。
每个栏目都有属性，可以在属性中增加 click 事件。

9.7.12.2. toolStripContainer

dock 属性设置为 fill

this.toolStripContainer1.TopToolStripPanel.Controls.Add(this.toolStrip1);
Controls.Add(this.toolStripContainer1);

9.7.13. statusStrip

在加入项目的对话框中，增加项目后，为 Name 属性填写合适的名称如 abc。在程序中，则可以 abc.text=”xx”;

9.8. FLASH

工具栏，空白处右键，选择项，COM 组件，Shockwave flash Object

播放

this.axShockwaveFlash1.LoadMovie(0, "d:\\1.swf");//必须使用全路径

接收命令

控件属性，FSCOMMAND
string s = e.command;

9.9. MDI

假设主窗口 Name 属性为默认的 Form1
1）设置主窗口 IsMdiContain 属性为 true
2）创建新的 winform 窗口（form2），在代码中修改构造函数

```
public Form2(Form1 parent)
{
        InitializeComponent();
        this.MdiParent = parent;
}
```

3）使用

```
Form2 child = new Form2(this);
child.Show();
```

第十章 wpf

10.1. 工程转换

10.1.1. 属性

控制台程序属性输出类型可以选择 Windows 应用程序，就会去掉控制台

依赖的 dll：
PersentationCore WPF 的核心类库
PersentationFramework 封装了与 WPF 控件相关类型的类库
System.Xaml XAML 解析的类库
WindowBase Window 窗体相关的类库

10.1.2. 简单创建

```
[STAThread]
static void Main(string[] args)
{
    Window mainWindow = new Window();
    mainWindow.Title = "WPF应用";

    //Application类型用于创建一个消息循环
    Application app = new Application();
    app.Run(mainWindow);
}
```

10.1.3. APP 使用

```
[STAThread]
static void Main(string[] args)
{
      App app = new App();
      app.Run();
}
```

```
  public class App : Application
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);
            MainWindow mainWindow = new MainWindow();    //窗口可以创建类，也可以用创建的wpf窗体（控制台不能创建窗体可以创建UserControl1进行改造）
            mainWindow.Show();
        }
    }
    public class MainWindow : Window
    {
        public MainWindow()
        {
            this.Title = "MainWindow";
            this.MouseLeftButtonDown                    +=                    new System.Windows.Input.MouseButtonEventHandler(MainWindow_MouseLeftButtonDown);
        }
        void                MainWindow_MouseLeftButtonDown(object                sender, System.Windows.Input.MouseButtonEventArgs e)
        {
            MessageBox.Show(e.GetPosition(this).ToString(), this.Title);
        }
    }
```

10.2. 新建窗口

添加窗口(WPF)，Window1.xaml

```
Window1   w = new   Window1 ();
w.Show();   //非模式对话框，需要在 ui 线程创建
```

```
if (true != ff.ShowDialog()) return;  //模式对话框，自带 ui 线程
this.DialogResult = true;
```

10.3. UserControl

10.3.1. 用户控件创建

新建 UserControl1（命名空间 UserSpace）

代码中包含：

```
public int userVal { get; set; }
public void Fun()
{
    int y = userVal + 10;
}
```

10.3.2. 调用

```
<Window .....
        xmlns:NNN="clr-namespace:UserSpace"
        .....>

<NNN:UserControl1  x:Name="userCtl"   userVal="123"/>
```

调用用户控件中的方法

```
userCtl.Fun();
```

10.4. 显示

10.4.1. 全屏

WindowStyle="None" WindowState="Maximized"

10.4.2. 去除边框

```
WindowStyle="None"
AllowsTransparency="True"
```

10.4.3. 鼠标拖拽

```
MouseDown="Window_MouseDown"    this.DragMove();
```

10.4.4. 透明背景

```
WindowStyle="None"
Background="Transparent"
AllowsTransparency="True"
Opacity="1"
```

10.4.5. 背景颜色

```
 Background = Brushes.Green;
```

10.4.6. 窗口居中

```
WindowStartupLocation = WindowStartupLocation.CenterScreen;
```

10.4.7. 顶层窗口

```
Topmost="True"
```

10.4.8. 屏幕大小

```
double height = SystemParameters.PrimaryScreenHeight;
double width = SystemParameters.PrimaryScreenWidth;
```

10.4.9. 光标

```
Cursor="Hand"


Cursor cur = new Cursor("1.cur");
```

```
this.Cursor = cur;
```

10.4.10. 切分窗口

```
<Grid Grid.Column="1">
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="200" MaxHeight="400" MinHeight="100"/>
    </Grid.RowDefinitions>
    <Grid Grid.Row="0"    ></Grid>
        <GridSplitter Grid.Row="1"    ResizeDirection="Rows" Height="3" HorizontalAlignment="Stretch"/>
    <Grid Grid.Row="2"></Grid>
</Grid>
```

10.4.11. 程序单例

```
public App()
{
      this.Startup += App_Startup;
}

void App_Startup(object sender, StartupEventArgs e)
{
    bool ret = false;
    System.Threading.Mutex mutex = new System.Threading.Mutex(true, "xxxx", out ret);
    if(!ret)
    {
            MessageBox.Show("已经运行");
            Environment.Exit(0);
    }
}
```

10.4.12. grid 位置改变

```
***.SetValue(Grid.ColumnProperty, 2);
```

10.5. 控件

10.5.1. 线程访问控件

==========方法 1

```
delegate void FunCallBack(string s,int num);

void Fun(string s1, string s2)

{ textBox1.Text= s1+ s2;}
```

线程中调用：

```
Fun ff = new Fun(fun);

textBox1.Dispatcher.Invoke(ff, new object[] { "aa",

"bb" });
```

==========方法 2

.net4.5 以上

```
tx.Dispatcher.Invoke(()=> {
        tx.Text = "xxx";
});
```

10.5.2. 排版

10.5.2.1. Grid

```
<Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="3*"/>
            <RowDefinition Height="2*"/>
        </Grid.RowDefinitions>
        <Grid Grid.Row="0">
            <Grid.ColumnDefinitions>
```

```
                <ColumnDefinition Width="200" />
                <ColumnDefinition />
                <ColumnDefinition Width="200" />
            </Grid.ColumnDefinitions>
            <Button Content="btn1" Grid.Column="0" HorizontalAlignment="Left"
Margin="48,60,0,0" VerticalAlignment="Top" Width="75"/>
            <Button Content="btn2" Grid.Column="2" HorizontalAlignment="Left"
Margin="60,60,0,0" VerticalAlignment="Top" Width="75"/>
        </Grid>
        <Grid Grid.Row="1">
            <Grid.RowDefinitions>
                <RowDefinition/>
                <RowDefinition/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition />
                <ColumnDefinition />
                <ColumnDefinition />
            </Grid.ColumnDefinitions>
            <Border Grid.Row="0" Grid.Column="1">
                <Button Content="btn3" />
            </Border>
        </Grid>
</Grid>
```

10.5.2.2. StackPanel

默认垂直排列

```
<StackPanel Grid.Row="1" Orientation="Horizontal">
    <Button Content="btn1" Margin="5"    HorizontalAlignment="Left" FontSize="20"/>
    <Button Content="btn2" Margin="5" HorizontalAlignment="Left" FontSize="20"/>
</StackPanel>
```

10.5.2.3. Expander

```
<Expander IsExpanded="False" Header="标志" BorderBrush="{x:Null}">
。。。。。
</Expander>
```

10.5.3. 属性关联

```
<Image Grid.Row="0" Name="img"></Image>
<TextBox Grid.Row="1" Text="aaa"  Width="{Binding ElementName=img, Path=ActualWidth}"/>
```

将 TextBox 的宽度与 Image 实际显示的图像宽度进行绑定，宽度会随 Image 显示图像宽度进行变化

10.5.4. 事件

10.5.4.1. 双击

```
private void Img_MouseDown(object sender, MouseButtonEventArgs e)
{
    e.ClickCount >=2
}
```

10.5.4.2. 位置获取

```
private void Img_MouseDown(object sender, MouseButtonEventArgs e)
{
      Point pt =  e.GetPosition(img);
}
```

获取鼠标点击位置在 img 控件中相对位置（GetPosition 可以获取相对于某控件的位置）

10.5.4.3. 鼠标滚轮

```
private void UserControl_MouseWheel(object sender,
System.Windows.Input.MouseWheelEventArgs e)
{
        if(e.Delta > 0) {//向上}
        else{}
}
```

10.5.5. 属性绑定

```
<TextBox Grid.Row="1" Text="aaa" Width="{Binding WIDTH, Mode=TwoWay}" Name="tb"  />


    class ViewModel : INotifyPropertyChanged
```

```
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged(string propertyName = null)
        {
            if (PropertyChanged != null)
                PropertyChanged.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }

        private int width;
        public int WIDTH        {
            get { return width; }
            set
            {
                width = value;
                OnPropertyChanged("Width");            }
        }
    }


ViewModel viewModel = new ViewModel { WIDTH = 50};

tb.DataContext = viewModel;
```

设置属性：viewModel.WIDTH = 100;

10.5.6. Button

```
<Button Content="btnTxt" Name="btn"  HorizontalAlignment="Left" Margin="208,207,0,0"
VerticalAlignment="Top" Width="75"/>
```

10.5.7. CheckBox

```
<CheckBox Content="CheckBox"  Name="cb" IsChecked="True" HorizontalAlignment="Left"
Margin="210,189,0,0" VerticalAlignment="Top"/>
```

10.5.8. ComboBox

```
<ComboBox       Name="cbb"       HorizontalAlignment="Left"       Margin="270,185,0,0"
VerticalAlignment="Top" Width="120"/>

cbb.Items.Insert(0,"aa");
```

```
cbb.Items.Insert(1, "bb");
int index = cbb.SelectedIndex;
```

10.5.9. DataGrid

10.5.9.1. 数据绑定

========combox 相关资源定义

```
xmlns:core="clr-namespace:System;assembly=mscorlib"

    <Window.Resources>
        <ObjectDataProvider x:Key="SexEnumKey" MethodName="GetValues"
ObjectType="{x:Type core:Enum}">
            <ObjectDataProvider.MethodParameters>
                <x:Type Type="local:SexEnum"/>
            </ObjectDataProvider.MethodParameters>
        </ObjectDataProvider>
    </Window.Resources>
```

==========前台界面

```
<DataGrid Name="DataGrid_Name" ItemsSource="{Binding}" CanUserAddRows="False"
AutoGenerateColumns="False" HeadersVisibility="Column">
             <DataGrid.Columns>
                <DataGridTextColumn Header="姓名"  Binding="{Binding Name}"
Width="100"/>
                <DataGridComboBoxColumn Header="性别"  SelectedItemBinding="{Binding
Sex}" ItemsSource="{Binding Source={StaticResource SexEnumKey}}"  Width="100"/>
            </DataGrid.Columns>
</DataGrid>
```

==========后台后台数据

```
public enum SexEnum { Male,FeMale};
public class DataItem
{
    public string  Name { get; set; }
    public SexEnum Sex { get; set; }
}
```

```
ObservableCollection<DataItem> m_data = new ObservableCollection<DataItem>();

DataItem cm = new DataItem();
cm.Name = "AA";
cm.Sex = SexEnum.Male;
m_data.Add(cm);

DataGrid_Name.DataContext = m_data;
```

10.5.9.2. 设置不可编辑

```
IsReadOnly = true;
```

10.5.9.3. 禁用用户排序

```
CanUserSortColumns="False"
```

10.5.9.4. 背景色

```
<DataGrid Background="#363636" Foreground="White">
```

10.5.9.5. 隐藏表头

```
HeadersVisibility="Column"
```

10.5.9.6. 选择选择行样式

```
<DataGrid.RowStyle >
    <Style TargetType="DataGridRow">
        <Setter Property="Background" Value="White"/>
        <Style.Triggers>
            <Trigger Property="IsMouseOver" Value="True">
                <Setter Property="Background" Value="LightGray"/>
            </Trigger>
            <Trigger Property="IsSelected" Value="True">
                <Setter Property="Background" Value="#90F670"/>
                <Setter Property="Foreground" Value="White"/>
```

```
                </Trigger>
            </Style.Triggers>
        </Style>
</DataGrid.RowStyle>
<DataGrid.Resources>
        <SolidColorBrush    x:Key="{x:Static    SystemColors.InactiveSelectionHighlightBrushKey}"
Color="DodgerBlue"/>
</DataGrid.Resources>
```

10.5.9.7. 行跳转

```
dg.SelectedIndex = 123;
dg.ScrollIntoView(dg.SelectedItem);
```

10.5.9.8. 双击获取行数据

MouseDoubleClick

```
DataGrid dg = sender as DataGrid;
int index = dg.SelectedIndex;
dgItem item = dg.SelectedItem as dgItem;
if (null == item) return;
```

判断点击位置的列名：

```
if (dg.CurrentColumn.Header.ToString() != "列名") return;
```

10.5.9.9. 获取多行

```
List<PrjDat> itemList = new List<PrjDat>();
foreach(PrjDat item in PrjDatSouce.SelectedItems)
{
    itemList.Add(item);
}

foreach (PrjDat item in itemList)
{
     m_data.Remove(item);
}
```

10.5.10. Image

ActualWidth属性表示显示图像后，控件实际的宽度，五图像时该值为0

```
img.Source = new BitmapImage(new Uri(@"D:\1.jpg", UriKind.RelativeOrAbsolute));

//从文件的byte[]数据数据加载
MemoryStream ms = new MemoryStream(imgDat);
ImageSourceConverter imgSrcCvt = new ImageSourceConverter();
img.Source = imgSrcCvt.ConvertFrom(ms) as System.Windows.Media.Imaging.BitmapFrame;
```

10.5.11. Label

```
<Label Name="Label1" Content="xxxxx" Width="200" Height="80" FontSize="40" FontFamily="Georgia" FontWeight="Bold"/>
```

10.5.12. ListBox

```
<ListBox Name="listbox1" ></ListBox>
```

添加方法 1：

```
listbox1.Items.Add(1);
listbox1.Items.Add(2);
```

添加方法 2：

```
List<int> ll = new List<int>();

ll.Add(2);
ll.Add(3);
listbox1.ItemsSource = null;
listbox1.ItemsSource = ll;
```

添加方法 3：

```
<ListBox Name="listbox1" DisplayMemberPath="ss"></ListBox>

struct Inf
```

```
{
    public string ss { get; set; }
}

List<Inf> ll = new List<Inf>();

Inf inf1 = new Inf();
inf1.ss = "aa";
ll.Add(inf1);

listbox1.ItemsSource = null;
listbox1.ItemsSource = ll;
```

10.5.13. RadioButton

命名空间内定义数据：

```
    public enum EmunStr
    {
        XX,
        YY,
    }
    public class EnumToBooleanConverter : IValueConverter
    {
        object IValueConverter.Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return value == null ? false : value.Equals(parameter);
        }
        object IValueConverter.ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return value != null && value.Equals(true) ? parameter : Binding.DoNothing;
        }
    }
    public class DataModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        private void OnPropertyChanged(string p_propertyName)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(p_propertyName));
        }
        private EmunStr _Dat;
        public EmunStr Dat
        {
```

```
            get
            {
                return _Dat;
            }
            set
            {
                _Dat = value;
                OnPropertyChanged("Dat");
            }
        }
    }
```

Xaml 的 Window 中引入资源

```
<Window.Resources>
        <local:EnumToBooleanConverter x:Key="EnumToBooleanConverter" />
</Window.Resources>
```

```
<RadioButton      Content="A"   IsChecked="{Binding   Path=Dat,Converter={StaticResource
EnumToBooleanConverter},ConverterParameter={x:Static local:EmunStr.XX}}"></RadioButton>
<RadioButton      Content="B"   IsChecked="{Binding   Path=Dat,Converter={StaticResource
EnumToBooleanConverter},ConverterParameter={x:Static local:EmunStr.YY}}"></RadioButton>
<TextBox  Grid.Row="1"  Text="{Binding Path=Dat}"></TextBox>
```

```
DataModel m_dataModel;

m_dataModel = new DataModel();
DataContext = m_dataModel;//设置数据绑定内容
```

获取选择数据：

```
m_dataModel.SampleEnum.ToString()
```

10.5.14. Rectangle

```
<Rectangle  Width="200" Height="100"  Fill="Blue" Stroke="Black" StrokeThickness="4" />
```

10.5.15. StackPanel

默认竖向排布

```
<StackPanel Orientation="Horizontal">
```

```
</StackPanel>
```

10.5.16. TextBlock

```
<TextBlock Name="Promt_Str" VerticalAlignment="Center" HorizontalAlignment="Center" Text="未定义" FontSize="30" Foreground="#FFE20404" />
```

使数据在超出行时自动换行

```
TextWrapping="Wrap"
```

10.5.17. RichTextBox

```
<RichTextBox Name="rr" VerticalScrollBarVisibility="Auto" HorizontalAlignment="Left" Height="474" Margin="128,299,0,0" VerticalAlignment="Top" Width="543" Grid.ColumnSpan="2"/>

rr.Document.Blocks.Add(new Paragraph(new Run("bbbb") { Foreground = Brushes.Green }));
rr.ScrollToEnd();

rr.Document.Blocks.Clear();
```

10.5.18. TabControl

```
<TabControl Name="Tbctl">
    <TabItem Height="30" Width="75" Header="项目1">
        <Button Content="aaa"></Button>
    </TabItem>
    <TabItem Height="30" Width="75" Header="项目2" >
        <Button Content="bbb"></Button>
    </TabItem>
    <TabItem Height="30" Width="75" Header="项目3" >
        <Button Content="ccc"></Button>
    </TabItem>
</TabControl>
```

10.5.19. Slider

```
sl.Minimum= 0;
sl.Maximum= 255;

ValueChanged
```

```
Slider sl = sender as Slider;
int val = (int)sl.Value;
```

10.5.20. ProgressBar

```
progress.Maximum = 100;
progress.Value = 2;
```

10.5.21. mediaElement

```
<MediaElement Name="mediaElement1"    Stretch="Fill" LoadedBehavior="Manual" />
```

删除标签中的margin，可以实现填满整个程序

Stretch="Uniform"可以让mediaElement按比例拉伸

打开文件

```
this.mediaElement1.Source = new Uri("d:\\1.avi");
//可以用File.Exists("d:\\1.avi");验证地址合法性
```

播放

```
this.mediaElement1.Play();
```

暂停

```
this.mediaElement1.Pause();
```

停止事件

属性：MediaEnded="mediaElement1_MediaEnded"

音量

```
this.mediaElement1.Volume += 10;
```

媒体长度

```
TimeSpan span = this.mediaElement1.NaturalDuration.TimeSpan;
string s = string.Format("{0},{1}", span.Minutes, span.Seconds);
```

当前位置

```
TimeSpan span = this.mediaElement1.Position;
```

快进

```
mediaElement1.Position = mediaElement1.Position + TimeSpan.FromSeconds(10);
```

设置播放位置

```
TimeSpan span = new TimeSpan(0, 0, 3);//设置为第3秒
this.mediaElement1.Position = span;
```

10.5.22. webBrowser

打开网页

```
wb.Navigate(new Uri(address));
wb.NavigateToString("<html><body></body></html > ");
```

跳转

```
if (wb.CanGoBack){wb.GoBack();}
if (wb.CanGoForward){wb.GoForward();}
```

导航到新的页面时：

```
wb.Navigating
```

导航之后，在下载 web 页面之前

```
wb.Navigated
```

web 页面下载完成时

```
wb.LoadCompleted
```

10.6. InkCanvas

```
<InkCanvas Name="inkCanvas" EditingMode="None" Background="Transparent" Strokes="{Binding InkStrokes, Mode=TwoWay}" ></InkCanvas>
```

属性设置：

```
DrawingAttributes drawingAttributes = new DrawingAttributes
{
        Color = Colors.Red,
        Width = 2,
        Height = 2,
        StylusTip = StylusTip.Rectangle,
        IsHighlighter = false,
        IgnorePressure = true,
};
inkCanvas.DefaultDrawingAttributes = drawingAttributes;
```

```
List<System.Windows.Point> pointList = new List<System.Windows.Point>();
```

绘制椭圆：

```
double a = 0.5 * (pt2.X - pt1.X);
double b = 0.5 * (pt2.Y - pt1.Y);
for (double r = 0; r <= 2 * Math.PI; r = r + 0.01)
{
    pointList.Add(new System.Windows.Point(0.5 * (pt1.X + pt2.X) + a * Math.Cos(r), 0.5
* (pt1.Y + pt2.Y) + b * Math.Sin(r)));
}
```

绘制矩形：

```
pointList.Add(pt1);
pointList.Add(new System.Windows.Point(pt1.X, pt2.Y));
pointList.Add(pt2);
pointList.Add(new System.Windows.Point(pt2.X, pt1.Y));
pointList.Add(pt1);
```

```
StylusPointCollection point = new StylusPointCollection(ptlist);
Stroke stroke = new Stroke(point)
{
     DrawingAttributes = inkCanvasMeasure.DefaultDrawingAttributes.Clone()
};
inkCanvasMeasure.Strokes.Add(stroke);
```

10.7. COM 组件使用

右键解决方案增加 用户控件（非 wpf 那个）UserControl1
工具栏选择项增加 WMP 控件，将控件拖入 UserControl1 的窗口

UserControl1 的 cs 文件中加入接口函数

```
public void play(String mediaPathName)
{
    this.axWindowsMediaPlayer1.URL = mediaPathName;
}
```

调用：
xaml 中拖入 WindowsFormsHost 控件

标签加入 Name 属性为 host1

```
UserControl1 f = new UserControl1();
host1.Child = f;
f.play("D:\\xxx.wmv");
```

10.8. 定时器

```
using System.Windows.Threading;

DispatcherTimer m_timer = new DispatcherTimer();

m_timer.Interval = TimeSpan.FromMilliseconds(1000);
m_timer.Tick += new EventHandler(m_timer_Tick);
m_timer.Start();
```

10.9. 消息处理

```
this.KeyUp += MainWindow_KeyUp;
```

键盘消息

```
if (e.Key == Key.Up)

Key.OemPlus【+键】
Key.OemMinus【-键】
Key.Return【回车】
```

10.10. 接收/截获消息

```
using System.Windows.Interop;

///添加函数
IntPtr hwnd = new WindowInteropHelper(this).Handle;
HwndSource.FromHwnd(hwnd).AddHook(new HwndSourceHook(fun));

private IntPtr fun(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam, ref bool handled)
  {//  uint l = (uint)lParam;可以将lParam转成数字

            if (msg == 0x0400+1){ }
```

```
            return IntPtr.Zero;
}
```

10.11. 命令行参数

App.xaml 中加入属性 Startup="Application_Startup"

在 App.xaml.cs 生成的函数中

```
private void Application_Startup(object sender, StartupEventArgs e)
{
        if (e.Args.Length == 0)
            return;
        foreach (string arg in e.Args){}
}
```

10.12. 动画

```
using System.Windows.Media.Animation;

//////////////////设置控件在 1s 内从不透明变成半透明
Storyboard story = new Storyboard();
DoubleAnimation animation = new DoubleAnimation();
//form 和 to 是指代属性变化前后的值
animation.From = 1;
animation.To = 0.5;
animation.Duration = TimeSpan.FromMilliseconds(1000);
Storyboard.SetTarget(animation, this.button1);
//单一属性名可以通过控件的属性列表查找
Storyboard.SetTargetProperty(animation, new PropertyPath("Opacity"));
story.Children.Add(animation);

story.Completed += Story_Completed; //动画完成完成时的触发事件，在开始动画前设置
story.Begin();
```

10.13. 右键菜单

控件带 context 属性

```
<ListBox.ContextMenu>
        <ContextMenu Name="cm" StaysOpen="true">
                    <MenuItem Header="选择文件夹"/>
                    <MenuItem Header="一级菜单">
                        <MenuItem Header="二级菜单"/>
```

```
                </MenuItem>
            </ContextMenu>
        </ListBox.ContextMenu>
```

第十一章 U3D

11.1. UNITY

11.1.1. 配置

==========安装
Browse
C:\Program Files\Unity\Editor
点击 Patch

==========打开
edit->preference->General
alaways show project wizard 打勾

打开时，选择工程的文件夹就是打开工程，或者可以打开 Assets 文件中保存的后缀为 unity 的场景文件。

==========andriod 发布
Build settings

scene in build： add current

player settings （andriod）设置：com.test.test

==========编辑器设置
Layout 设为 tall

Edit-》preference-》external tools-》external script editor，可以选择 vs 的编辑器。

==========导入包
如果包中包含.unity 后缀的场景文件，可以打开一个场景。场景是使用各种元素搭建起来的工程。

========文件路径不能包含中文

11.1.2. 视图

Z 轴（蓝色）代表相机视角的方向。Y 代表空间高度（上下，可以用 0 为基准）。X 代表左右。
点击坐标轴，可以改变场景视图，用二维的视角进行调节。
按住 alt 和鼠标左键，可以任意移动场景视图角度。

Scene 上面有四个按钮（对应快捷键 QWER）。
第一个按钮手，代表移动场景视图。
第二个十字形，代表在鼠标场景中是移动模型的位置（拖动箭头可以单方向移动，选中的箭头会变成黄色）。
第三个代表旋转，选择了模型后，点击选中的轴线，沿着线移动（线平行方向移动，不是垂直方向）。
第四个代表缩放操作。

11.1.3. Scene

File->New Scene(Ctrl+N)可以创建不同场景。每个场景都是独立的，可以使用相同的资源，如模型，代码等。
Build Setting 中，可以将当前场景加入。在其中加入多个场景，就可以在代码中切换场景。
放在最上面的场景是游戏刚开始运行时出现的场景。
Application.LoadLevel("s1");//s1 代表某个场景的名称

11.1.4. Assets(project)

该菜单项目对应 Assets 视图中鼠标右键菜单。
选择 Create，可以加入各种资源。

11.1.4.1. Prefab

Prefab相当与一个将各种模型，代码等组织在一起的游戏物体（GameObject）。

将 hierarchy 中的模型直接拖入 Asset 文件夹中，可以直接生成 prefab。
也可以新建一个 prefab，将模型或者代码文件拖拽到 prefab 中。

使用时，可以直接拖入场景，也可以拖入 Hierarchy 视图中。

11.1.4.2. Material

可以为场景中的模型添加颜色与贴图，设置该资源的属性，拖入场景中的模型，即可改变模型的样子。

11.1.5. GameObject(hierarchy)

该菜单项目可以对应 hierarchy 视图中的 create。
可以直接加入物体到了场景中（不加入到 Assets 中）

所有游戏物体都有 Transform 属性，其他属性都可以删除或者添加。即所有游戏物体都是在 transform 属性基础上叠加其他属性。

11.1.5.1. Directional light

可以加入光源，否则场景会很暗。

11.1.5.2. Plane

默认出现的位置相当于地面。
直接将图片拖拽到该 plane 即可改变。
可以用来当墙面，或者 GUI 的背景。

11.1.5.3. GUI Texture

类似与 plane，默认出现位置相当于相机的对面。
可以向 Inspecror 中的 Texture 栏目拖入图片。
可以为这个 GUI Texture 增加脚本，按实际情况设置位置

```
void Start () {
    guiTexture.pixelInset = new Rect(0, 0, Screen.width, Screen.height);
}
```

11.1.6. Component(Inspector)

该菜单项目可以对应 inspector 视图中的 add Component。
Inspector 视图中包含了模型的各种属性。

11.1.6.1. 天空

选中相机， rendering-》Sky Box，然后可以在相机对应的 inspector 视图中看到 sky box，点击 custom skybox 那行的按钮，即可选择天空(需要先导入天空的包)

11.1.6.2. 刚体

Obj 是 GameObject，而且包含刚体属性

11.1.6.2.1. 重力

physicals-》rigibody，物体会自动往地面落下。

Constrains-》freeze rotation，勾选 x，y，z。则落下时不会改变旋转位置。也可以加入代码
if (rigidbody)　rigidbody.freezeRotation = true;

rigidbody.useGravity = false;

Physics.gravity.y = -2;

11.1.6.2.2. 加速度

//可以让刚体朝着前方运动，初速度是 200（速度的矢量方向合成）
Vector3 v = new Vector3(0,0,200);
Obj.rigidbody.velocity= transform.TransformDirection(v);

11.1.6.2.3. 力矩（自旋转）

Obj.rigidbody.AddTorque(Random.onUnitSphere * 0.1f,ForceMode.Impulse);

11.1.6.3. 碰撞

当两个模型接触后，如果都有碰撞体（且物体是刚体)，就会有碰撞效果。
component-》physicals 加入 Box Collider 等。
做碰撞效果时，刚体的属性不要约束位置的 x，y，z，否则会在碰撞后一直移动。

11.1.6.4. 隐藏

去掉 Mesh Renderer 属性，可以让模型看不见

11.1.6.5. 声音

音频监听器（Audio listener）通常附加在使用的相机上（在加入 Audio Source 会自动加入到

相机）。声音在 3D 空间内，所以如果包含声音的模型远离相机，声音就会减小。

勾选 play on Awake 可以在物体出现时自动播放，其他属性都可以在这个 Audio Source 中设置。

```
void Awake() {
    audio.volume = 0.2F;
    audio.Play();
}
```

播放 assets 中的声音

模型必须有 Audio Source

```
public AudioClip sound=new AudioClip();  //可以在 inspector 视图中选择声音

audio.PlayOneShot (sound, 1);//1 代表音量最大
```

11.1.6.6. Animation

模型中包含 Animation 组件

```
animation.Play("***", PlayMode.StopAll);
```

如果没有在播放任何动画，就播放

```
if (!animation.isPlaying)
            animation.Play("***");
```

11.1.7. 代码编辑

project 中 assets 区域，右键，Create，C#script（还可以导入 package，将其中的资源直接拖入左边的场景中，或拖拽到 hierarcy 视图中的物体上）

能拖入物体的代码必须继承自 MonoBehaviour，Update,FixedUpdate 函数是按照帧频自动调用。

11.1.8. 位置移动

移动当前模型的位置

```
if(Input.GetKey(KeyCode.W))
{//写成 Vector3.forward*0.1f，则可以减少移动的大小，不写*0.1f 会自动赋值一个移动大小
    transform.Translate(Vector3.forward);
}
if(Input.GetKey(KeyCode.S))
{
```

```
        transform.Translate(Vector3.back);
}
if (Input.GetKey (KeyCode.A))
{
        transform.Translate(Vector3.left);
}
if (Input.GetKey (KeyCode.D))
{
        transform.Translate( Vector3.right);
}
```

游戏开始时，将当前模型在 1s 内移动到目标模型的位置

transform.position = Vector3.Lerp(transform.position, target.transform.position, Time.time);//Time.time 指游戏开始到现在所用的时间

11.1.9. 输入设备

Edit->project Settings->input,在视图中可以看到默认的输入（Axes）
右键输入项目，可以选择复制或者删除输入项目

```
float WorS = 0.0f;//-1...1

WorS=Input.GetAxis("Vertical");//对应 Axes 中的名称
if (WorS > 0)
{
        transform.Translate(Vector3.forward*0.1f);
}
else if(WorS<0)
{
        transform.Translate(Vector3.back*0.1f);

}
```

----------THRUSTMASTER----------
(必要时删除多余输入设置，只使用手柄的输入设置，避免按键定义干扰)
=====左右
Type： Joystick Axis
Axis：X axis

```
float x = Input.GetAxis("Horizontal");
```

=====上下
Type： Joystick Axis
Axis: Y axies

float y = Input.GetAxis("Vertical");

=====滑动轴
Type： Joystick Axis
Dead：0.1
Sensitivity:1
Axis: 4th axis(Joysticks)
float x = Input.GetAxis("**");

=========按钮(使用 GetButtonUp 或 GetButton)
-----前按钮
if (Input.GetButton("**"))
Positive Button：joystick button 0
-----中按钮
Positive Button：joystick button 1
-----左按钮
Positive Button：joystick button 2
-----右按钮
Positive Button：joystick button 3

11.1.10. Camera

场景中必须有一个相机对象，且 camera 组件被勾选。

代码中 Camera.main 指当前使用的相机

相机切换
public Camera camera0，camera1;
camera0.enabled = true;
camera1.enabled = false;

11.1.11. 视角转动

可以移动视角（相机跟随物体），可以直接放到 update 中，相当于鼠标移动。
transform.Rotate(0, Input.GetAxis("Mouse X"), 0);
transform.Rotate(-Input.GetAxis("Mouse Y"), 0, 0);

转动位置设置
transform.rotation = new Quaternion(0, 0, 0, transform.rotation.w);

限制上下的视角转动（transform.rotation.x 总是小数）
if (transform.rotation.x < 0.2&&transform.rotation.x>-0.2)

```
{
        transform.Rotate(-Input.GetAxis("Mouse Y"), 0, 0);
}
if(transform.rotation.x > 0.2)
{
        transform.Rotate(-1, 0, 0);
}
else if (transform.rotation.x < -0.2)
{
        transform.Rotate(1, 0, 0);
}
```

11.1.12. 坐标转换

==========世界坐标系转屏幕坐标系
Vector3 Pos = Camera.main.WorldToScreenPoint(transform.position);

==========屏幕坐标系转世界坐标系
Vector3 v = Camera.main.ScreenToWorldPoint(new Vector3(Input.mousePosition.x, Input.mousePosition.y, Camera.main.nearClipPlane+40));
可以将屏幕坐标转换为世界坐标，Camera.main.nearClipPlane+40 可以让 z 的位置在相机面前的 40m 远处, Camera.main.nearClipPlane 代表相机所在的 Z 平面。

------物体移动到鼠标点击的位置（假设代码放在物体中）
float near = transform.position.z-Camera.main.transform.position.z;
Vector3 v = Camera.main.ScreenToWorldPoint(new Vector3(Input.mousePosition.x, Input.mousePosition.y, Camera.main.nearClipPlane+near));

transform.position = new Vector3(v.x, v.y,transform.position.z);

11.1.13. 模型控制

11.1.13.1. 模型变量

public GameObject[] obj ;//可以在 Inspector 视图将物体拖入，Inspector 上面关于该数组的 size 会自动变化。如果是 int 这种类型，可以在 size 中输入大小，回车确认，然后进行填写。
在 inspector 视图中找到该变量，选择物体
在 C#脚本中声明了 public 变量，将脚本加入 GameObject，则可以在该模型的 Inspector 中代码文件放置的位置看到该变量，而且可用在这里设置初始值的大小。

通过使用函数查找
public GameObject treeObj;//类内变量

treeObj=GameObject.Find ("tree");//放在 Start 函数中
tree 是要控制的模型的名字，通过 treeObj 可以对模型进行控制

11.1.13.2. 组件

A a= GetComponentsInChildren<A>();
可以获取子物体组件，即物体中包含的子物体中如果包含的代码中有类 A（或继承自类 A），就可以获取子物体。

Obj.GetComponent<AudioListener> ().enabled = false;
可以让模型中的声音侦听器不可用

11.1.13.3. 复制

GameObject clone = (GameObject)Instantiate(Obj, new Vector3(x, y,z), transform.rotation);
将 Obj 进行复制，位置为（x,y,z）,角度为当前的物件角度（可以使用 Quaternion.identity，让角度为默认的直线）。

11.1.13.4. 碰撞销毁

在两个物体碰撞时，将另一个物体（没有加入这个代码的物体）销毁。
将碰撞体的属性 Is Trigger 打勾。
void OnTriggerEnter (Collider other){Destroy(other.gameObject);}

在检测到碰撞后销毁当前物体
```
void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag == "***")//可以为被碰撞的物体设置标签，当前物体碰撞另一个物体时，通过标签判断是不是碰撞到了指定物体
            Destroy(gameObject);
}
```

11.1.13.5. 坐标位置

```
Obj.transform.position=new Vector3(
transform.position.x,
transform.position.y,
transform.position.z-1f);
//JS 写法  var direction = Vector3(0,5,0);
```
在 hierarchicy 视图中，将模型 B 拖入模型 A 下层，则可以让 B 随着 A 位置自动移动，此时 B

的坐标是相对与 A 的位置的坐标。

11.1.13.6. 旋转

默认的角度 Quaternion.identity

转动时遵守左手螺旋定则。
Obj.transform.Rotate(0, 100* Time.deltaTime, 0);
//自旋转，绕 y 轴每秒转 100 度

obj.transform.RotateAround (obj2.transform.position, Vector3.up, 80 * Time.deltaTime);
//围绕 obj2，每秒转 80 度

transform.rotation = Quaternion.identity;
//重置旋转角度

11.1.14. 外部命令

[RequireComponent(typeof(Rigidbody))]
可以在代码中加入模型属性，确保在视图中不能删除该属性

[HideInInspector]
代码中下面一行的 public 变量不会在视图中出现

[AddComponentMenu("***")]
可以在 Component 菜单中增加菜单项目。
在 Hierarchy 视图中选择中物体后，点击菜单，就可以将代码文件直接加入到物体。

[SerializeField]
private int x=0；
私有变量不在 Inspector 视图中显示，但是加了 SerializeField 就可以在视图中显示了。

11.1.15. EditorWindow

在项目中放入继承自 EditorWindow 的代码，写入 OnGUI 这样的函数，就可以在编辑窗口中增加一个自定义的菜单项目。

11.2. GUI

加入函数，将 GUI 内容写在该函数内

void OnGUI()

11.2.1. 层级

GUI.depth = -100;

11.2.2. 颜色

Color.clear;代表纯透明

Color color= Color.red;
color.a = 0.5f;
//自定义颜色，可以设置颜色的 GRBA，a 设置 0.5 代表半透明，0 代表纯透明

public Color color = new Color(0.2F, 0.3F, 0.4F，0.5F);//GRBA

可以改变控件的颜色,该类代码一定要放在 OnGUI() 函数最上面。
GUI.color = Color.yellow;// 将影响背景和文本颜色。

GUI.backgroundColor = Color.red;//背景颜色
GUI.contentColor = Color.yellow;//文本颜色

11.2.3. 字体大小

以设置 button 为例

方法一：（会影响整个程序的字体）
GUI.skin.button.fontSize=30;

方法二：（只设置单独的实例）
GUIStyle style=new GUIStyle();
style.normal.background = null;
style.normal.textColor=new Color(1,0,0);
style.fontSize = 40;

GUI.Button (new Rect (x, h , width, h), "nn",style)

11.2.4. 位置

GUI.skin.label.alignment = TextAnchor.UpperRight;

可以让 label 在右上角

11.2.5. 按钮

```
if (GUI.Button (new Rect (x, y, cx, cy), "name")) {
    Debug.Log("Clicked the button with an image");
}
```

11.2.6. 输入文本

```
private string str=”xxxx”;

str = GUI.TextArea(new Rect(x, y, cx, cy), str, 500);
```

11.2.7. 静态文本

```
GUI.Label(new Rect(x, y, cx, cy), "Hello World!");
```

11.2.8. 图片绘制

```
public Texture2D cur;//Inspector 中选择图片
GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), cur, ScaleMode.StretchToFill, true, 10.0F);//最后两个参数（进行通道融合，缩放比例 10：1），可以不写
```

11.2.9. 视频

```
public MovieTexture movTexture;//视图中拖入视频，需要安装 quicktime 解码

GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), movTexture, ScaleMode.StretchToFill);
movTexture.Play();//可以写在其他函数中
```

11.3. function

11.3.1. 定时器

```
Invoke("fun", 2);//2 秒后执行 fun 函数

void fun() { print("xx");}
```

11.3.2. 键盘/鼠标

if (Input.GetMouseButton(0))//判断鼠标左键是否按下，参数 1 时，代表右键

Screen.showCursor = false; //隐藏鼠标
Screen.lockCursor=true;
将鼠标隐藏并锁定在视图中间

鼠标位置
如果获取的位置是以左下角为原点，需要进行转换
public float x = Input.mousePosition.x;
public float y = Screen.height-Input.mousePosition.y;

11.3.3. 退出程序

Application.Quit();

11.3.4. 分辨率

int cx=Screen.width;
int cy=Screen.height;

11.3.5. 限制数值

限制 value 的值在 min 和 max 之间， 如果 value 小于 min，返回 min。 如果 value 大于 max， 返回 max， 否则返回 value

Clamp (*value* : float, *min* : float, *max* : float)

11.3.6. 两点间距离

float x = Vector3.Distance(a : Vector3, b : Vector3);

11.3.7. 调试

Debug.Log("xx");

print(Input.mousePosition);

11.3.8. 定时器

float timer=3.0f;

update 中：
timer -= Time.deltaTime;

11.3.9. 随机

Random.value //得到 0 到 1 之间的随机数

if (Random.Range(0,100)<30)//在范围内取随机数，概率为 30%

Random.rotation

11.3.10. 鼠标点击移动

会自动寻找鼠标点击的方向中的碰撞体，找到了碰撞体，就会往碰撞体定义一个射线

类内变量
public Vector3 end;

初始化
end = transform.position;

update 函数中
```
if (Input.GetMouseButtonDown(0))
{
	//定义一个当前相机到另一个物体之间的射线。
	Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
	RaycastHit hit;
	if (Physics.Raycast(ray, out hit))
	{
		end = hit.point;
	}
}
```

transform.position = Vector3.Lerp(transform.position, end, Time.deltaTime * 2.0f);

第十二章 JAVA

12.1. java

12.1.1. JDK 环境变量

JAVA_HOME
C:\Program Files\Java\jdk...

path
%JAVA_HOME%\bin

java5 以上不需要配置 CLASSPATH

运行环境：
JRE_HOME　　jre 的 bin 目录上级

12.1.2. 结构

```
public class A{
    public static void main(String Args[]){
        int    x=10;
        System.out.println("int"+x);
    }
}
```

编辑文本保存为　A.java。

使用命令　javac　A.java　（编译命令时可以不区分文件名大小写）

生成文件　A.class

运行命令　java A

保存时，一定要用类名作为文件名。单个 java 文件可以包含多个类，但最多只能有一个 public 类，如果某个类为 public，则该类名要作为文件名。

导出 jar 时，需要选择主类为应用程序主入口。

运行时 java -jar *.jar

label: ……..;

continue label;

12.1.3. Applet

```
<applet code="A.class"  width="100"  height="50">
```

```
import java.applet.Applet;
import java.awt.Graphics;
public class A extends Applet{
        public void paint(Graphics g){
                g.drawString("xxxx",20,30);
        }
}
```

12.1.4. 数据类型

包括基本数据类型和引用数据类型，引用数据类型包括 class 类型，枚举，数组。

对象在函数中传递时，基本类型都是值传递，引用类型传递的是引用，可以在函数中改变该对象的值，但不能改变该对象的指向。

12.1.5. 基本数据类型

数据类型	名称	位长	默认值
布尔型	boolean	1	False
字节型	byte	8	0
字符型	char	16	‘\u0000’
短整型	short	16	0
整型	int	32	0
长整型	long	64	0
浮点型	float	32	0.0f
双精度型	double	64	0.0

========整型常量

不含小数的整数值，十进制以非 0 开头，八进制以 0 开头，十六进制以 0x 开头。如果在数值后面加 L 或者 l 则表示 64 位长整型。

========浮点型常量

1）十进制形式：.123，0.123,123.0

2）科学计数法：123e3 或 123E-3。E 后面必须是整数。如果后面加 f 或 F，代表单精度，加上 d 或者 D，表示双精度。不加后缀默认为双精度，占 64 位。

========字符常量

可以用字符编码表示，如 A 的八进制表示'\101'，十六进制'\u0041'

无法通过键盘输入的用转译符号：

'\b' 退格　　'\r' 回车

'\n' 换行　　'\t' 水平制表符

'\''　　'\"'　　'\\'

12.1.6. 类型转换

a=(int)b

String str = "12.2";

double d = Double.parseDouble(str);

int i = Integer.parseInt(str);//str 不能是小数

12.1.7. Class

class 类型必须使用 new 来创建对象：B b=new B();

也可以创建临时对象调用函数：new B().fun();

this 可以指代当前对象

super 指代父类：super();super.fun();

super(xxx)　只能在构造函数第一行调用基类的构造

12.1.7.1. 修饰符

[public][abstract][final] class 类名

[extends 父类][implement　接口]{}

抽象类 abstract 不能实例化一个对象，只能被继承。如 Number 只能产生一个数的子类，比如 Integer 或者 Float。

final 类不能有子类。为了提高系统安全性和定义一个完全类。

接口是消息传递的通道，通过接口，消息才能传递到处理方法中进行处理。

12.1.7.2. 成员变量

成员变量描述了类和对象的状态，有时也称为属性、数据、域。

[public][private][protected][package]

[static][final][transient][volatile]　　类型　　名称

final 可以定义常量

protected 变量可以被声明它的类和子类以及同一个包中的类访问。如果子类在其他包，子类的对象可以访问，但子类中由父类产生的对象就不能访问。

package 在声明时常省略，即没有修饰的变量为 package 变量。

final 变量在程序运行过程中不能被改变。

transient 一般在对象序列化上使用。

volatile 用来防止编译器对变量进行优化。

12.1.7.3. 成员方法

[public][private][protected][package]

[static][final][abstract][native][synchronized]

返回值类型　　方法名(参数表)[throws 异常类型]

12.1.7.4. 语句块

```
static
{   //main 函数所在类，需要用 static 语句块，其他不需要
    System.out.println("xxxxxx");
}
```

在类中直接写的语句块，类似于类的构造函数中执行初始化语句，程序运行初始化时执行。

12.1.7.5. 继承和多态

继承使用关键字 extends，子类如果和父类有同名函数时，函数访问权限不能低于父类。此时将子类对象赋值给父类对象，父类调用该函数时为子类的函数。

```
class Base{public void fun(){System.out.println("Base");}}

class Der extends Base{public void fun(){System.out.println("Der");}}
```

使用时 Base b1 = new Dir();

或者 Dir d = new Dir();Base b2 = d;

此时调用 fun 函数，输出为 Dir。

将基类改为接口，可以使用接口实现多态

```
interface Base{public void fun();}
class Der implements Base{public void fun(){System.out.println("Der");}}
```

instanceof　用于测试一个对象是否是一个指定类的实例

12.1.7.6. 抽象类

不能被实例化的类，可以有静态方法，通过类名直接调用。

```
abstract class B//包含抽象方法的类必须声明为 abstract
```

{

abstract void fun();//抽象方法不能有实现，且声明为abstract

}

class C extends B

{

void fun(){}

}

12.1.7.7. 匿名对象

创建对象时就调用其中的函数，属于临时对象

```
new A().show();
```

12.1.7.8. 创建对象时加入代码

创建 A 对象时，为 A 类加入成员变量 y，重写 show 方法，在其中可以直接访问外部变量 z

```
int z = 1;
A a = new A(){int y =1; void show(){System.out.println(x+z+y);}};
```

```
(new A(){...}).show();
```

可以加入代码后创建临时变量使用

如果是直接创建接口的对象，则需要该语法实现接口的函数

12.1.7.9. 内部类

在类 A 内定义的类 B，依赖于 A 的存在，所以 B 中的代码可以直接访问到 A 中的成员。

12.1.8. 数组

int[] arr； 或者 int arr[];

Java 在数组定义时并不为数组元素分配内存，因此[]中不用指出数组的长度。

数组作为 Array 类的实例，可以引用属性和方法。

arr.length 直接获取长度属性

for(int x : arr){}遍历数组中每个元素

Arrays.sort(arr);//对数组进行排序，需要实现 Comparable 接口

========一维数组

1）int[] arr=new int[3];

2) int[] a={1,2,3,4,5};

========多维数组

int[][] arr=new int[3][4];

int arr[][]={{0,1,2,3,4},{3,4,5},{6,7,8}};

二维数组的第二维的长度可以不相等

12.1.9. 包装类型

将基本数据类型包装到类类型中，实现 toString 和 compareTo 等函数

String str = Integer.toHexString(17); //将整数转换为十六进制字符串

short 等基本类型包装类为 Short 这类将首字母大写，char 对应 Character， int 对应 Integer。

```
short st = 1;
Short s = new Short(st);//赋值装箱
s = 1;//自动装箱
```

12.1.10. 枚举

枚举是一种包含自身全局对象的特殊的类，且构造函数必须是 private。

```
enum   E
{
    e1(1),
    e2(2);     //两个 E 的全局对象，类似单例

    public int x = 0; //成员变量
    E(int a){x = a;}//构造函数，只能是 private
}
```

======使用

```
E e = E.e1;//获取枚举对象
if(E.e1 == e)
{
  String str = e.toString();
//将枚举的对象名 e1 转换为字符串“e1”。与之对应的是
E.valueOf("e1")，通过字符串“e1”获取对象 E.e1
}
========遍历
for(E temp : E.values())
{
      int value = temp.x;    //获取对象内部的成员变量
}
```

12.1.11. 接口

```
interface    I
{
      void fun();//不能有权限修饰符
}
class B implements I
{
    public void fun(){}//实现接口时必须是 public
}
```

12.1.12. 异常

```
class Ex extends Exception
{//自定义异常类
	String s = "Exx";
}

class B
{
	void fun(int x) throws Ex
	{//可以抛出异常的函数，不能直接被调用，需要在 try、catch 中调用
		if(10 == x)
		{
			throw new Ex();
		}
	}


	public void ProcessEx()
	{
		try{
				fun(10);
		}catch(Ex e)
		{
			System.out.println(e.s);
		}
	}
}
```

12.1.13. 包

javac A.java -d D:\

在 D：下生成 class，如果 A.java 有声明包路径，则会生成包目录

一般新建项目为一个目录 P，目录下有 abc 目录，则为一个 abc 的包。

假设 main 函数所在类为 P 目录下的 ABC 目录，则运行时，需要在 P 目录同级下运行：java P.ABC.A

package P.ABC;//包声明必须是第一句，目录间用.，表示 P 目录下的 abc 目录

import　P.ABC.*; //导入 ABC 包中的全部内容，也可以将*改为类名，只导入某个类。导入的内容访问权限如果是默认，则只能在同一个包下才能访问

12.1.14. 导入 jar

将 jar 放入工程目录，右键 jar 文件，选择 Build Path-》add to build path，则可以看到 reference libraries 中出现新的包和类。（不出现 build path 菜单说明工程配置损坏）

12.1.15. javaDoc

```
/**
 *说明
 *
 * @param  参数 1 说明
 * @param  参数 2  说明
 * @return
    */
```

参数说明中不能包含括号冒号等特殊字符

12.2. function

12.2.1. Scanner

```
Scanner scanner = new Scanner(System.in);
String str = scanner.nextLine();//获取字符串
char c = scanner.nextLine().charAt(0);//获取字符
int a = scanner.nextInt();//获取数字
scanner.close();
```

12.2.2. String

==========编码转换

byte[] send = str.getBytes("gbk");

String 不能改变自身长度和内容

赋值修改时是重新分配内存，指向了新的地址

==========构造

String str = "abcdef";

String str = new String();

String str = new String(char[]);

String str = new String(String);

==========操作

比较操作一般不使用==

if(str.indexOf("bc") >= 0) {//字符串或字符查找}

lastIndexOf(str)　最后一次出现的索引位置

str.startsWith(" ab")//是否以某个字符序列开始

str += "xx";//添加字符

String substr = str.substring(1,4);//从下标 1 截取到下标 3

str = str.toUpperCase();//转换为大写

str = str.replace("ab", "xx");//字符串替换

str = str.trim();//去掉首尾的空格

char[] s = str.toCharArray();//转换为字符数组

String[] s = str.split(" ");//字符串拆分

boolean b = str.contains("xx");//是否包含字符串

12.2.3. StringBuffer

StringBuffer 可以改变自身内容，属于线程安全

StringBuilder 适合单线程使用

```
StringBuffer strb = new StringBuffer("abc");
strb.append("def");
strb.delete(0, 1);//删除第一个字符
strb.insert(3, "xx");//在下标为 3 的位置插入
strb.setCharAt(1, 'z');//替换下标为 1 的字符
strb.reverse();//逆序
strb.replace(1, 3, "mmm");//将下标 1 到 2 的字符替换
String str = strb.toString();
strb.delete(0, strb.length());//清空数据
```

12.2.4. System

System.exit(0);//终止当前 java 虚拟机

long start = System.currentTimeMillis();//返回以毫秒为单位的当前时间

System.gc(); //清理垃圾

获取当前系统的属性

```
import java.util.*;

Properties p = System.getProperties();
Enumeration pNames = p.propertyNames();
while(pNames.hasMoreElements())
{
      String key = (String)pNames.nextElement();
    String value = System.getProperty(key);
    System.out.println(key+"   "+value);
}
```

12.2.4.1. 获取当前目录

```
System.getProperty("user.dir")
```

12.2.5. Runtime

打开和关闭其他程序

Runtime tr = Runtime.getRuntime();

Process p = tr.exec("notepad.exe");

p.destroy();

12.2.6. Thread

Thread()
Thread(实现 runnable 的对象)
Thread(实现 runnable 的对象，窗口名)

12.2.7. 创建线程

12.2.7.1. 后台线程

创建的线程默认是前台线程，即主线程结束后，线程未运行完也不会结束。在 start 前调用函数 setDaemon 可以让线程变成后台线程，当主线程运行结束后也停止。

```
Thread th = new Thread(new T());
th.setDaemon(true);
th.start();
```

12.2.7.2. 继承 Thread

```
class T extends Thread
{
    public void run(){}
}
```

调用：

```
new T().start();
```

```
new Thread(){
    @Override
```

```
    public void run() {

    }
}.start();
```

12.2.7.3. 实现 Runnable 接口

```
class T implements Runnable {
    public void run(){ }
}
```

调用：

```
new Thread(new T()).start();
```

可以在 Thread 类中传入同一个对象，让多个线程共享同一个对象的内容。

```
class T implements Runnable {
    int count = 10;
    public void run() {
        while (count > 0) {
            Thread t = Thread.currentThread();//获取运行该语句的线程
            System.out.println(t.getName() + " count:" + count--);
        }
    }
}
```

调用：

```
T t = new T();
new Thread(t,"window1").start();
new Thread(t,"window2").start();
```

12.2.7.4. 线程休眠

```
try
{
   Thread.sleep(2000);//当前线程休眠 2S
}catch(InterruptedException e){}
```

12.2.7.5. 线程优先级

可以设置某个线程的优先级，作为提高程序效率的一种手段。

th.setPriority(Thread.MAX_PRIORITY);

12.2.7.6. 线程让步

当某个线程执行到 yield 函数时，转换成就绪状态，只有与当前线程优先级相同或者更高的才能获得执行机会。

Thread.yield();

12.2.7.7. 线程插队

当某个线程中调用其他线程的 join 方法后，调用的线程被阻塞，直到被调用的线程结束后它才会执行。

th.join();

12.2.7.8. 线程同步

多个线程操作同一对象，在该对象类的同步代码块或同步方法中操作相同资源，可以进行互斥。

第一个线程进入 synchronized(lock)后占用了锁，第二个线程到 synchronized(lock) 时阻塞，直到第一个线程结束 synchronized 块内语句或者其他激活条件时第二个线程才执行 synchronized(lock)后一句

同步代码块：

```
static Object lock = new Object();
```

```
public void run()
{
    。。。。。
    synchronized(lock){ 同步块    }
    。。。。。
}
```

同步方法：

同步方法的锁就是 this 对象，相当于进入函数后进入 synchronized(this)语句块

synchronized void fun(){}

==========多线程通信

lock.wait() 使当前线程放弃同步锁 lock，并进入等待，直到其他线程进入此同步锁，并调用 notify 或者 notifyAll 方法唤醒该线程为止。

notify 唤醒此同步锁上等待的第一个调用 wait 方法的线程。

notify 和 wait 需要使用锁对象调用，且调用位置在该锁对象由 Syncoronized 触发的代码块内。

线程 1，2 公用 Stack 对象，1 线程调用 push()，2 线程调用 pop()。两个线程依靠互斥对象和 notiy、wait 方法交替执行

```
Stack stack = new Stack();
new Thread(new Th1(stack)).start();
new Thread(new Th2(stack)).start();
```

```
class Stack {
//可以不使用lock对象，直接把push和pop定义成synchronized方法，则两个方法使用进入时使用的是this作为锁
    static Object lock = new Object();
    void push() throws InterruptedException {
    synchronized(lock)
    {
       int i = 0;
       while (true) {
           System.out.println("push" + i++);
           Thread.sleep(1000);

           if (i == 5) {
               i = 0;
               lock.notify();
               lock.wait();
           }
          }
      }
    }

    void pop() throws InterruptedException {
       synchronized (lock) {
           int i = 0;
           while (true) {
               System.out.println("pop...." + i++);
               Thread.sleep(1000);

               if (i == 5) {
                   i = 0;
                    lock.notify();
                    lock.wait();
               }
           }
       }
   }
}
```

12.2.8. Math

Math.PI

Math.ceil(1.1)//向上取整 值为 2

Math.floor(1.9)//向下取整 值为 1

Math.abs(-1)//绝对值

Math.round(1.5)//四舍五入 值为 2

Math.random()//生成大于等于 0.1 小于 1 的随机数

产生 10 个[0-100]的随机数

```
import java.util.Random;
Random r = new Random();
for(int x = 0; x < 10 ; ++x)
{
      System.out.println(r.nextInt(100));
}
```

12.2.9. 日期

```
//获取日期差
Calendar c = Calendar.getInstance();
c.set(2018, 11, 3,21,39,12);
long time1 = c.getTimeInMillis();
c.set(2018, 11, 3,22,49,12);
long time2 = c.getTimeInMillis();
long between_minutes=(time2-time1)/(1000*60);

Date dateCur = new Date();//获取当前时间
System.currentTimeMillis();//获取当前时间


//格式化日期
SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String strCurTime  = sdf.format(dateCur);
String setTime = sdf.format(c.getTime());
```

```
    Date dateParse = sdf.parse("2018-11-3 22:49:12");
```

12.2.10. 文件操作

12.2.10.1.文件

```
File  ff = new File("D:\\1.txt");

ff.exists();//判断文件或者目录是否存在

ff.isDirectory();//判断是否是文件夹

ff.isFile();//判断是否是文件

ff.createNewFile();//不存在则创建新文件

ff.delete();//删除文件

ff.renameTo(new File("D:\\xx.txt"));//移动文件，也可以修改文件夹名,文件夹必须存在

ff.mkdirs();//创建目录，目录可以多层级创建
ff.mkdir();//创建目录，目录只能是一层新目录
ff.delete();//删除目录，不能删除有子目录的目录
```

12.2.10.2.遍历文件夹和子文件夹

```
void  ShowFile(File ff)
{
    if(!ff.isDirectory()) return;

    File[]  ffList = ff.listFiles();
    for(File f : ffList)
     {

if (f.isFile()){
    String fileName = f.getName();
    if(fileName.endsWith(".txt"))
    System.out.println(f.getName() + "    " + f.getPath());
}
```

```
else  ShowFile(f);
    }
}
```

12.2.10.3.读文件

```
FileInputStream fis = new FileInputStream("D:\\1.txt");
InputStreamReader isr = new InputStreamReader(fis,"UTF-8");
BufferedReader bfr = new BufferedReader(isr);

String line;
while((line = bfr.readLine()) != null)
{
    System.out.println(line);
}

bfr.close();
isr.close();
fis.close();
```

12.2.10.4.写文件

```
FileOutputStream fos = new FileOutputStream("D:\\2.txt");
OutputStreamWriter  osw = new OutputStreamWriter(fos, "utf-8");
BufferedWriter  bfw = new BufferedWriter(osw);

bfw.write("aaaa\r\n");

bfw.close();
osw.close();
fos.close();
```

12.2.10.5.追加内容

FileWriter 和 FileReader 只能是本地的 utf-8 编码，不能更改

```
boolean append = true;
FileWriter  fw = new FileWriter("D:\\1111.txt",append);
fw.write("aaaa\r\n");
fw.flush();
fw.close();
```

12.2.11. 序列化

```
//实现序列化接口Serializable（包含serialVersionUID）
//序列化类中的成员必须要支持序列化  static,transient后的变量不能被序列化
//当一个父类实现序列化，子类自动实现序列化，不需要显式实现Serializable接口
class People implements Serializable
{
    private static final long serialVersionUID = 1L;
    int id;
    String name;

    People(int id,String name)
    {
        this.id = id;
        this.name = name;
    }
}
class PeopleList implements Serializable
{
    private static final long serialVersionUID = 1L;//确保序列化和反序列化使用
相同id
    private ArrayList<People>  list = new ArrayList<People>();

    void Add(int id , String Name)
    {
        list.add(new People(id, Name));
    }
    void Show()
    {
        for(People p:list)   System.out.println(p.id + " " + p.name);
    }
}


============序列化
void Save() throws FileNotFoundException, IOException
    {
        PeopleList pList = new PeopleList();
        pList.Add(1, "aa");
        pList.Add(2, "bb");
        pList.Add(3, "cc");


                ObjectOutputStream   oos   =   new   ObjectOutputStream(new
```

```
FileOutputStream("D:\\temp.bin"));
        oos.writeObject(pList);
        oos.flush();
        oos.close();
    }

==========反序列化
void Read() throws FileNotFoundException, IOException, ClassNotFoundException
    {
        ObjectInputStream      oin      =      new      ObjectInputStream(new
FileInputStream("D:\\temp.bin"));
        PeopleList  list = (PeopleList ) oin.readObject();

        list.Show();
    }
```

12.3. Generics

12.3.1. 泛型

import java.util.*;

for(object x : arr) 不能改变其中内容，只能访问元素

不能在<>中写基本类型，只能是基本类型的包装类型

```
//迭代容遍历
Iterator<String> it = l.iterator();
while(it.hasNext()){
     System.out.println(it.next());
}

class Data<T>
{
    T temp;
    public void save(T temp)
    {
        this.temp = temp;
    }
    public Object get()
    {
```

```
        return temp;
    }
}
```

使用时

```
Data<String> d = new Data<String>();
d.save("abc");
System.out.println(d.get());
```

12.3.2. 去重/排序

12.3.2.1. Hash

需要重写方法 hashCode 和 equals

```
class Data
{
    public int i = 0;
    //重写方法
    public String toString()
    {
        Integer x = i;
        return x.toString();
    }
    public int hashCode()
    {
        return i;
    }
    public boolean equals(Object obj)
    {
        if(obj == this)
            return true;
        if(! (obj instanceof Data))
            return false;

        Data d = (Data)obj;

        return i == d.i;
    }
}
```

12.3.2.2. Tree

需要实现 Comparable

```
class Data implements Comparable
{
    public int i = 0;

    public int compareTo(Object obj)
    {//如果是包装类型，可以直接用其中的 compareTo 函数
        Data d = (Data)obj;
        return this.i - d.i; //等于返回 0，小于返回负数，大于返回正数
    }
}
```

可以自定义排序规则

```
    TreeMap<String,Integer>    m    =    new    TreeMap<String,Integer>(new
MyComparator());

class MyComparator implements Comparator
{
    public int compare(Object obj1,Object obj2)
    {
        String s1 = (String)obj1;
        String s2 = (String)obj2;
        return s2.compareTo(s1);
    }
}
```

12.3.3. Vector

```
Vector<String> v = new Vector<String>();

v.setSize(10);
v.trimToSize();//使容量和使用长度相等，充分利用空间
```

排序

```
vecStu.sort(new Comparator<Student>() {
    public int compare(Student o1, Student o2) {
```

```
        return o1.id - o2.id;}
});
```

12.3.4. 单列集合

12.3.4.1. ArrayList

封装了一个长度可变的数组，随机访问快

```
ArrayList<String>  l = new ArrayList<String>();
l.add("abc"); //添加元素
l.size()//获取元素个数
l.get(0)//获取下标为 0 的元素
l.set(0, "xx");//设置下标为 0 元素
l.remove(0);//移除第 0 号元素
List<String>  sub = l.subList(1, 2);//截取部分
l.addAll(1,sub);//添加 list 中从下标 1 开始的元素
l.toString();//转换为字符串包含所有元素
```

12.3.4.2. LinkedList

内部为双向循环链表，删除节点快

特有方法：getFirst(),removeFirst(),addFirst()

12.3.4.3. HashSet

存储的元素不保证有序且不重复

HashSet<Integer> s = new HashSet<Integer>();

12.3.4.4. TreeSet

存储的元素不重复，且保证有序

12.3.5. 双列集合

12.3.5.1. HashMap

12.3.5.1.1. 数据定义

```
class MyKey
{
    public MyKey(int _id) {
        id = _id;
    }
    int  id;

    @Override
    public int hashCode() {
    //hashMap 是数组加链表的结构
    //id%3 表示 hashMap 是 3 个位置的数组，每个位置中再根据 equals 判断是否相等
    //遍历输出时可以看到 id 顺序是根据%结果放置
        return id%3;
    }
    @Override
    public boolean equals(Object obj) {
        return id == ((MyKey)obj).id;
    }
    @Override
    public String toString() {
        return String.format("%d", id);
    }
}

HashMap<MyKey, Double> map= new HashMap<MyKey, Double>();
```

12.3.5.1.2. 放入元素

```
map.put(new MyKey(1), 5.1);
map.put(new MyKey(2), 7.3);
map.put(new MyKey(2), 2.1);//相同则替换
```

12.3.5.1.3. 合并

```
//返回值  参数...
BiFunction<Double, Double, Double> fun = (p1,p2)->{return p1 +p2;};
//      BiFunction<Integer, Integer, Integer> fun = new BiFunction<Integer,
Integer, Integer>()
//      {
//
//              @Override
//              public Integer apply(Integer t, Integer u) {
//                      // TODO Auto-generated method stub
//                      return t + u;
//              }
//
//      };

map.merge(new MyKey(2), 1.1, fun);//相同则合并
```

12.3.5.1.4. 遍历

```
        //遍历 key
        for(MyKey key : map.keySet())
        {
        System.out.println(key);
        }

        //遍历 value
        for(Double d : map.values())
        {
        System.out.println(d);
        }




        //遍历 key 和 value
        for (Map.Entry<MyKey,Double>  o : map.entrySet()) {
            System.out.println(o.getKey() + "  " + o.getValue());
        }
```

12.3.5.1.5. 查找

```
MyKey key = new MyKey(4);
if(map.containsKey(key))//查找
{
	Double b = map.get(key);
	map.put(key, 100.1);//替换
}
```

12.3.5.2. TreeMap

需要实现 Comparable

```
class MyKey implements Comparable<MyKey>
{
    public MyKey(int _id) {
        id = _id;
    }
    int  id;
    @Override
    public int compareTo(MyKey o) {
        return id - o.id;
    }

    @Override
    public String toString() {
        return String.format("%d", id);
    }
}
```

键值是按照顺序排列

基本的二叉树都需要满足一个基本性质，即树中的任何节点的值大于它的左子节点，且小于它的右子节点。按照这个基本性质使得树的检索效率大大提高。

TreeMap 的实现是红黑树算法的实现，是一颗自平衡的排序二叉树

12.3.6. Properties

```
Properties p = new Properties();
p.setProperty("color", "red");
p.setProperty("size", "12");
System.out.println(p.getProperty("color"));
```

12.3.7. Collections 工具类

ArrayList<Double> l = new ArrayList<Double>();

Collections.reverse(l);//对 list 进行反转

shuffle(list)//进行随机排序

sort(list)

swap(list,int i, int j)将制定 list 中 i 处和 j 处元素进行交换

addAll(Collection c, T elements)将所有元素添加到制定的 collection 中

binarySearch(list,object)使用二分法查找有序的 list 中的某个元素

Object max(Collection col)查找最大元素

Object min(Collection col)

replaceAll(list,oldobj,newobj)用新的 obj 替换 list 中所有旧的 obj

12.4. socket

12.4.1. URL

```
import java.net.InetAddress;
import java.net.UnknownHostException;

String s = "www.baidu.com";
InetAddress ts = null;
try
{
    ts = InetAddress.getByName(s);
}catch(UnknownHostException e){}

if(null != ts)
{
  System.out.println(s + " IP 地址是 " + ts.getHostAddress());
}
```

//构造和解析 URL

```
URL u = new URL("https://www.baidu.com/s?ie=utf-8&wd=baidu");


u.getProtocol() //获取协议 https
u.getHost()//获取主机名 www.baidu.com
u.getFile()//获取文件名 s?ie=utf-8&wd=baidu


import java.io.*;
```

//读取当前页面的内容，用字符串读出 html 文件中写的内容

```
BufferedReader r = new BufferedReader(new InputStreamReader(u.openStream()));
String s;
while(null != (s = r.readLine()))
{
    System.out.println(s);
}
r.close();
```

12.4.2. TCP

```
import java.io.*;
import java.net.*;
```

12.4.2.1. 服务端

```
try {
  ServerSocket server = new ServerSocket(1234);
  while (true) {
    Socket s = server.accept();

      DataInputStream data = new DataInputStream(s.getInputStream());
    System.out.println(data.readUTF());

    data.close();
    s.close();
    }
} catch (Exception e) {}
```

12.4.2.2. 客户端

```
try {
  Socket s = new Socket("localhost", 1234);

  DataOutputStream data = new DataOutputStream(s.getOutputStream());
  data.writeUTF("xx");

  data.close();
  s.close();
} catch (Exception e) {}
```

12.4.3. UDP

```
import java.net.*;
```

DatagramPacket 用于发送和接收时，构造函数不同

12.4.3.1. 服务端

```
try
```

```
{
   DatagramSocket dSocket = new DatagramSocket(1234);
   while(true){
    byte[]  inBuffer = new byte[100];
DatagramPacket inPacket = new DatagramPacket(inBuffer, inBuffer.length);
    dSocket.receive(inPacket);//接收信息

    InetAddress cAddr = inPacket.getAddress();
    int cPort = inPacket.getPort();
    String s = new String(inPacket.getData(),0,inPacket.getLength());
    System.out.println("recive: " + s);
    System.out.println("client name: " + cAddr.getHostName());
    System.out.println("client prot: " + cPort);
   }
}catch(Exception e){}
```

12.4.3.2. 客户端

```
try {
    DatagramSocket dSocket = new DatagramSocket();
    InetAddress sAddr = InetAddress.getByName("127.0.0.1");
    String s = "xxxx";
    byte[] outBuffer = s.getBytes();
DatagramPacket outPacket = new
DatagramPacket(outBuffer,outBuffer.length,sAddr,1234);
    dSocket.send(outPacket);

      dSocket.close();
} catch (Exception e) {}
```

12.5. c++Mix

12.5.1. C++调用 java

JNI(Java Native Interface)意为 JAVA 本地调用，它允许 Java 代码和其他语言写的代码进行交互
需要将 jre\bin\server\jvm.dll 加入环境变量 path，移动 dll 放入工程运行时出错
加载包时，需要包的全路径，如 `env->FindClass("java/lang/String");`

12.5.1.1. Java 文件

编译成 class 文件 `TestFun.class`

```
public class TestFun {
    public static int intMethod(int x,int y) {
        return x*y;
    }
    public void sayHelloFromJava(String str,int listLen,int[] listVal)
    {
       System.out.println(str);
       for(int i = 0 ; i <listLen;++i)
       {
         System.out.print(listVal[i] + " ");
       }
    }
}
```

12.5.1.2. C++调用

```
#include <jni.h>
#pragma comment(lib,"jvm.lib")
```

头文件目录

C:\Java\jdk1.8.0_60\include;C:\Java\jdk1.8.0_60\include\win32

Lib 目录

C:\Java\jdk1.8.0_60\lib

运行 dll 目录

C:\Java\jre1.8.0_60\bin\server

```
    //加载 jvm
    JavaVMInitArgs vm_args;
    memset(&vm_args, 0, sizeof(vm_args));
    vm_args.version = JNI_VERSION_1_8;//jdk 版本
    vm_args.nOptions = 1;
    JavaVMOption options[1];
    options[0].optionString = "-Djava.class.path=.";
    vm_args.options = options;

    JNIEnv *env;
    JavaVM *jvm;
    long status = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);
    if (status == JNI_ERR) throw(string("JNI_CreateJavaVM erro"));
```

```
    //加载类 TestFun
    jclass cls = env->FindClass("TestFun");
    if (0 ==  cls ) throw(string("FindClass erro"));


    //调用类中的静态方法 intMethod
    jmethodID mid = env->GetStaticMethodID(cls, "intMethod", "(II)I");//返回值 int，两个参数为 int 类型
    if (0 == mid) throw(string("intMethod erro"));
    jint square = env->CallStaticIntMethod(cls, mid, 5,5);
    printf("Result of intMethod: %d\n", square);




    //创建类并调用函数 sayHelloFromJava
    jmethodID ctor = env->GetMethodID(cls, "<init>", "()V");
    jobject obj = env->NewObject(cls, ctor);
    mid = env->GetMethodID(cls, "sayHelloFromJava", "(Ljava/lang/String;I[I)V");//无返回值，第一个参数 int，第二个 String
    if (0 == mid) throw(string("sayHelloFromJava erro"));

    jstring str1 = env->NewStringUTF("I am class Instance");
    const jint len = 3;
    jintArray testIntArray = env->NewIntArray(len);
    jint test[len] = {1,2,3};
    env->SetIntArrayRegion(testIntArray, 0, len, test);
    env->CallVoidMethod(obj, mid, str1, len, testIntArray);


    jvm->DestroyJavaVM();
```

域描述符

V	void
Z	boolean
B	byte
C	char
S	short
I	int
J	long

F	float
D	double
L..	引用类型 如 Ljava/lang/String;
[..	数组类型 如 [I
[[..	二维数组 如 [[I

12.6. extend

12.6.1. swing

======安装
http://www.eclipse.org/windowbuilder/download.php
页面找到 eclipse 对应版本，点击 link
复制下载地址粘贴在 eclipse 中 help-》install new software 中，下载插件

新建 other-》windowBuilder-》Swing designer-》JFrame

======控件设置
MVC（Model View Controller）：模型、视图和控制器

点击 design 进入可视化编辑（view）
点击控件，再点击窗口位置即可加入代码（Model）
右键控件，Add event hander 加入响应事件(Control)

12.6.2. json

=======依赖
```
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20180130</version>
</dependency>
```

=======生成
```
//{"name":"AA","do":["eat","sleep"],"age":10}
JSONObject obj = new JSONObject() ;
```

```
obj.put("name", "AA") ;
obj.put("age",10) ;

JSONArray arr = new JSONArray() ;
arr.put("eat") ;
arr.put("sleep") ;
obj.put("do", arr) ;

String str = obj.toString() ;
System.out.println(str);
```

=======解析

```
String str = "{\"dat\":[{\"inf\":\"abc\",\"lon\":\"116\",\"lat\":\"116\"},{\"inf\":\"def\",\"lon\":\"116.1\",\"lat\":\"116.1\"},{\"inf\":\"gg\",\"lon\":\"116.2\",\"lat\":\"116.2\"}]}";

JSONObject jsonObj = new JSONObject(str);
JSONArray arr = jsonObj.getJSONArray("dat");

for (Iterator<Object> tor = arr.iterator(); tor.hasNext();) {
                JSONObject obj = (JSONObject) tor.next();
        System.out.println(obj.get("inf"));
        System.out.println(obj.get("lon"));
        System.out.println(obj.get("lat"));
}
```

```
String json = '[["abc","116","39"],["def","116.1","39.2"],["gg","116.2","39.1"]]'
JSONArray arr = new JSONArray(json);
int len = arr.length();
for (int pos = 0 ; pos< len;++pos)
{
        JSONArray arrObj = (JSONArray)arr.get(pos);
        if(arrObj.length() != 3) continue;

        String infStr = (String) arrObj.get(0);
        String lonStr = (String) arrObj.get(1);
```

```
    String latStr = (String) arrObj.get(2);
}
```

12.6.3. log4j

12.6.3.1. 配置说明

```
log4j.rootLogger = [ level ] , appenderName, appenderName, …
```

level 是日志记录的优先级
优先级从高到低：【字母不区分大小写】
OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL、自定义的级别。
通过在这里定义 level，可以控制到应用程序中相应级别的日志信息的开关。比这里定义了 INFO 级别，则应用程序中所有 INFO 级别以下日志信息将不被输出。

appenderName 指带后面要配置的日志控制，可以同时编写多个日志控制。其中的 Threshold 字段定义了控制级别，表示控制该级别以上的日志。
如：log4j.appender.****.Threshold = DEBUG 表示控制 DEBUG 级别以上的日志的输出位置（在原有底层优先级日志输出的基础上，将 DEBUG 级别以上的日志再输出到另一个文件）
配置日志信息输出目的地 Appender，其语法为：
log4j.appender.appenderName = fully.qualified.name.of.appender.class
log4j.appender.appenderName.option1 = value1
log4j.appender.appenderName.option = valueN
其中，Log4j 提供的 appender 有以下几种：
org.apache.log4j.ConsoleAppender（控制台），
org.apache.log4j.FileAppender（文件），
org.apache.log4j.DailyRollingFileAppender（每天产生一个日志文件），
org.apache.log4j.RollingFileAppender（文件大小到达指定尺寸的时候产生一个新的文件），
org.apache.log4j.WriterAppender（将日志信息以流格式发送到任意指定的地方）

12.6.3.2. 依赖

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

12.6.3.3. 配置文件

log4j.properties 文件放在工程生成的 classpath 下
java 文件夹下放 java 代码，该目录中的文件就会放在 classpath 下
javaweb 中放在 WEB-INF/classes

```
### 设置###
log4j.rootLogger = debug,A,B,C

### 输出 ALL 级别以上日志到控制台###
log4j.appender.A = org.apache.log4j.ConsoleAppender
log4j.appender.A.Target = System.out
log4j.appender.A.layout = org.apache.log4j.PatternLayout
log4j.appender.A.layout.ConversionPattern = [%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS}
method:%l%n%m%n


### 输出 DEBUG 级别以上的日志到=./logs/a.log  javaweb 只能绝对路径###
log4j.appender.B = org.apache.log4j.DailyRollingFileAppender
log4j.appender.B.File = ./logs/a.log
log4j.appender.B.Append = true
log4j.appender.B.Threshold = DEBUG
log4j.appender.B.layout = org.apache.log4j.PatternLayout
log4j.appender.B.layout.ConversionPattern    =    %-d{yyyy-MM-dd    HH:mm:ss}
[ %t:%r ] - [ %p ]  %m%n

### 输出 ERROR 级别以上的日志到=./logs/b.log ###
log4j.appender.C = org.apache.log4j.DailyRollingFileAppender
log4j.appender.C.File =./logs/b.log
log4j.appender.C.Append = true
log4j.appender.C.Threshold = ERROR
log4j.appender.C.layout = org.apache.log4j.PatternLayout
log4j.appender.C.layout.ConversionPattern    =    %-d{yyyy-MM-dd    HH:mm:ss}
[ %t:%r ] - [ %p ]  %m%n
```

12.6.3.4. 调用

```
import org.apache.log4j.Logger;

private static Logger logger = Logger.getLogger(类名.class);

logger.debug("debug");
logger.info("info");
```

```
logger.warn("warning");
logger.error("error");
logger.fatal("fatal");
```

12.6.4. request

```
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.6</version>
</dependency>
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;


import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.StatusLine;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;



public static byte[] GetRequest(String url) throws URISyntaxException, ClientProtocolException,
IOException {
```

```
        RequestConfig config = RequestConfig.custom().setConnectTimeout(5000).setSocketTimeout(3000).build();
        CloseableHttpClient client = HttpClientBuilder.create().setDefaultRequestConfig(config).build();

        HttpGet httpGet = new HttpGet();
        httpGet.setURI(new URI(url));

        HttpResponse httpResponse = client.execute(httpGet);
        HttpEntity entity = httpResponse.getEntity();
        byte[] body = EntityUtils.toByteArray(entity);
        StatusLine sL = (StatusLine) httpResponse.getStatusLine();
        int statusCode = ((org.apache.http.StatusLine) sL).getStatusCode();
        if (statusCode != 200)
            return null;
        return body;
    }
    public static byte[] sendPost(String url, Map<String, Object> params) throws ClientProtocolException, IOException {

        List<NameValuePair> pairs = null;
        if (params != null && !params.isEmpty()) {
            pairs = new ArrayList<NameValuePair>(params.size());
            for (String key : params.keySet()) {
                pairs.add(new BasicNameValuePair(key, params.get(key).toString()));
            }
        }

        HttpPost httpPost = new HttpPost(url);
        if (pairs != null && pairs.size() > 0) {
            httpPost.setEntity(new UrlEncodedFormEntity(pairs));
        }

        RequestConfig config = RequestConfig.custom().setConnectTimeout(5000).setSocketTimeout(3000).build();
        CloseableHttpClient client = HttpClientBuilder.create().setDefaultRequestConfig(config).build();

        CloseableHttpResponse response = client.execute(httpPost);
        int statusCode = response.getStatusLine().getStatusCode();
        if (statusCode != 200) {
            httpPost.abort();
            throw new RuntimeException("HttpClient,error status code :" + statusCode);
        }
```

```
        HttpEntity entity = response.getEntity();
        if (entity != null) {
                return EntityUtils.toByteArray(entity);
        } else {
                return null;
        }
}
```

```
String url = "http://127.0.0.1:8080/DevSer/GetDevDat";

//byte[] body = GetRequest(url);

Map<String, Object> params = new HashMap<String, Object>();
params.put("IntVal", 12);
params.put("StrVal", "ss");
byte[] body = sendPost(url,params);

String json = new String(body, "UTF-8");
```

12.6.5. xml

```
<dependency>
        <groupId>jdom</groupId>
        <artifactId>jdom</artifactId>
        <version>1.1</version>
</dependency>
```

```
import java.io.*;
import org.jdom.*;
import org.jdom.input.*;

SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(new File("cfg.xml"));
Element root = doc.getRootElement();
String Ip = root.getChild("Ip").getValue();
String port = root.getChild("port").getText();
```

12.6.6. poi

```
import java.io.FileOutputStream;
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFCellStyle;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;


        //创建一个 webbook，对应一个 Excel 文件
         HSSFWorkbook wb = new HSSFWorkbook();

         //格式
         HSSFCellStyle style = wb.createCellStyle();

         //在 webbook 中添加一个 sheet,对应 Excel 文件中的 sheet
         HSSFSheet sheet = wb.createSheet("学生表一");



         //在 sheet 中第 1 行第一列加入数据 xx
         HSSFRow row = sheet.createRow(0);
         HSSFCell cell = row.createCell(0);
         cell.setCellValue("xxx");
         cell.setCellStyle(style); //设置单元格格式



         //保存文件
         try
         {
               FileOutputStream fout = new FileOutputStream("D:/students.xls");
               wb.write(fout);
               fout.close();
         }
         catch (Exception e)
         {
               e.printStackTrace();
         }
```

第十三章 JAVAWEB

13.1. web 路径

访问页面地址：http://127.0.0.1:8080/jyl/
页面中包含连接：resources/jyl.jpg，等价于 http://127.0.0.1:8080/jyl/resources/jyl.jpg

13.2. maven

13.2.1. 配置

13.2.1.1. 依赖获取

依赖查询地址：
http://mvnrepository.com/

13.2.1.2. 环境变量

M2_HOME，MAVEN_HOME （部分项目使用）：maven 所在目录
path *****\bin

13.2.1.3. 检查安装

命令行输入 mvn -v

13.2.1.4. 本地仓库目录

maven 的 conf 目录下：settings.xml
设置<localRepository>标签的仓库目录（默认为${user.home}/.m2/repository）：
<localRepository>C:\JaveWeb\mvn\repository</localRepository>

13.2.1.5. 中央仓库镜像

```
<mirror>
    <id>nexus-aliyun</id>
    <mirrorOf>central</mirrorOf>
```

```
        <name>Nexus aliyun</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    </mirror>
```

13.2.1.6. 本地依赖

部分依赖在中心仓库无法找到，需要本地配置

13.2.1.6.1. 安装方式

下载 kaptcha 并解压到 c:\kaptcha-3.2.5.jar，运行以下命令
mvn install:install-file -Dfile=c:\kaptcha-3.2.5.jar -DgroupId=com.google.code -DartifactId=kaptcha -Dversion=3.2.5 -Dpackaging=jar

即可使用：

```
<dependency>
        <groupId>com.google.code</groupId>
        <artifactId>kaptcha</artifactId>
        <version>2.3.2</version>
 </dependency>
```

表示仓库在/com/google/code/kaptcha/2.3 目录

13.2.1.6.2. 相对路径方式

直接放在了项目的 lib 文件夹

```
 <dependency>
            <groupId>com.abc</groupId>
            <artifactId>def</artifactId>
            <scope>system</scope>
            <version>1.0</version>
            <systemPath>${basedir}\src\lib\def.jar</systemPath>
 </dependency>
```

13.2.2. 错误处理

正常运行，xml 异常
相应包的_remote.repositories 修改：
json-20180130.pom>nexus-aliyun=
json-20180130.jar>nexus-aliyun=
json-20180130-sources.jar>nexus-aliyun=

变为：
json-20180130.pom>
json-20180130.jar>
json-20180130-sources.jar>

13.2.3. 版本定义

<properties> <spring.version>4.2.4.RELEASE</spring.version>

<version>${spring.version}</version>

13.2.4. eclipse 配置

window-》preferences-》Maven
Installations 设置 maven 所在目录
User Settings 设置仓库的 settings.xml 所在目录

All Catalogs 选择 org.apache.maven.archetypes maven-archetype-webapp 1.0
Grop Id: com.***
Artifact Id: PrjName

无法导入时：mvn eclipse:eclipse -Dwtpversion=2.0

eclipse 打开一个 workspace，对 eclipse 的设置是对应该 worksapce 的配置。

在导入 maven 项目前，设置仓库地址。
Window->Preferences->Maven->User settings
默认是用户目录下的.m2 目录下的 settings.xml，在该位置可以自定义 settings.xml
通过该设置可以定位本地的仓库目录

13.2.5. IDEA 配置导出 war

Mave Projects -> Lifecycle -> package

13.2.6. 导出 jar 运行包

13.2.6.1. POM 导出信息

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>1.2.1</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer

    implementation="org.apache.maven.plugins.shade.resource.ManifestResourc
eTransformer">
                                    <mainClass>com.test.MainClass</mainClass>
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

13.2.6.2. POM 属性版本

```
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
```

13.2.6.3. eclipse 导出运行

右键工程-》Run As -》Maven install

项目的 target 目录中运行：
java -jar ******-0.0.1-SNAPSHOT.jar

13.2.7. 私服

13.2.7.1. nexus 配置

https://www.sonatype.com/download-oss-sonatype

参数修改：
修改数据存储路径，文件目录：bin/nexus.vmoptions
修改 IP、端口、访问根目录，文件目录：etc/nexus-default.properties

13.2.7.2. 启动

运行 bin 目录下的 exe
./nexus.exe /run

默认端口 8081
http://127.0.0.1:8081

使用命令可以安装服务：
nexus.exe /install

13.2.7.3. 仓储设置

使用管理员账号登录，默认登录名及密码：admin admin123

maven-central：maven 中央库，默认从 https://repo1.maven.org/maven2/拉取 jar
maven-releases：私库发行版 jar
maven-snapshots：私库快照（调试版本）jar
maven-public：仓库分组，把上面三个仓库组合在一起对外提供服务，在本地 maven 基础配置 settings.xml 中使用。

13.2.7.3.1. maven 配置

settings.xml 文件内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository>C:\JaveWeb\mvn</localRepository>

  <pluginGroups>
    <pluginGroup>org.sonatype.plugins</pluginGroup>
  </pluginGroups>

  <proxies>
  </proxies>


  <servers>
    <server>
      <id>nexus</id>
      <username>admin</username>
      <password>admin123</password>
    </server>
  </servers>


  <mirrors>
    <mirror>
      <id>nexus</id>
      <mirrorOf>*</mirrorOf>
      <url>http://localhost:8081/repository/maven-public/</url>
    </mirror>
  </mirrors>

  <profiles>
    <profile>
      <id>nexus</id>
      <repositories>
        <repository>
          <id>central</id>
          <url>http://central</url>
```

```
            <releases><enabled>true</enabled></releases>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>
      </repositories>
     <pluginRepositories>
        <pluginRepository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>

  <activeProfiles>
    <activeProfile>nexus</activeProfile>
  </activeProfiles>
</settings>
```

13.2.7.3.2. 工程配置 pox.xml

```
  <distributionManagement>
  <repository>
      <id>nexus</id>
      <name>Releases</name>
      <url>http://localhost:8081/repository/maven-releases</url>
    </repository>
    <snapshotRepository>
      <id>nexus</id>
      <name>Snapshot</name>
      <url>http://localhost:8081/repository/maven-snapshots</url>
    </snapshotRepository>
  </distributionManagement>
  <build>
    <defaultGoal>compile</defaultGoal>
    <finalName>page</finalName>
    <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.4.2</version>
          <configuration>
```

```
            <skipTests>true</skipTests>
          </configuration>
        </plugin>
        <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
    </plugins>
  </build>
```

13.2.7.4. 编译项目

项目右单击->Run As->Maven build..
在界面中填写 Goals 为：deploy -e
POM 配置为<version>0.0.1-SNAPSHOT</version>，则编译到私服的 maven-snapshots 里。
<version>0.0.1</version>则编译到 maven-release 里。

13.2.7.5. 上传 jar 包

mvn deploy:deploy-file

-DgroupId=com.aliyun.oss -DartifactId=aliyun-sdk-oss -Dversion=2.2.3 -Dpackaging=jar -Dfile=D:\aliyun-sdk-oss-2.2.3.jar -Durl=http://127.0.0.1:8081/repository/maven-3rd/ -DrepositoryId=nexus-releases

13.3. Tomcat

13.3.1. 配置

webapps 中存放 web 项目文件

conf 下 server.xml 中 <Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>
定义连接端口,默认 8080

13.3.2. 运行

运行环境需要配置环境变量：JRE_HOME

可以通过 tomcat 下 bin 目录中 startup.bat 启动
默认打开 index 文件

http://127.0.0.1:8080/webPrjName/
项目的根目录就是该地址（项目的 webapp 目录）

导出 WAR 文件，放在 tomcat 的 webapps 目录下，使用 bin 目录下 startup.bat 运行

tomcat 运行环境的 conf 目录下 server.xml Host 标签可以设置项目文件目录:
<Context docBase="E:\apache-tomcat-9.0.0.M21\webapps\webTest" path="/webTest" reloadable="true" source="org.eclipse.jst.jee.server:webTest"/>

13.3.3. Eclipse 配置

==========加入 tomcat 环境
Window->Preferences->server->runtime environment

add... apache Tomcat v9.0
设置 tomcat installation directory(bin 目录上级)

==========web 项目配置
右键工程 build path libraries 页签

add libray...
Server Runtime 选择 tomcat 版本
JRE System Library 选择 jre 版本

13.3.4. IDEA 配置

不需要安装 tomcat，会自动自动下载 tomcat 插件相关相关内容

pom.xml:

```
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
        <port>8080</port>
```

```
            <path>/</path>
            <uriEncoding>UTF-8</uriEncoding>
            <server>tomcat7</server>
        </configuration>
</plugin>
```

Edit Configurations.. -> + ->Maven
Name: web
Command line:tomcat7:run

13.4. javaWeb

13.4.1. JSP 元素

========指令标识，java 代码，jsp 表达式使用

```
<%@ page import = "java.text.SimpleDateFormat" %>
<%@ page import = "java.util.Date" %>
<%
Date date = new  Date();
SimpleDateFormat   simpleDateFormat   =   new   SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
String  time = simpleDateFormat.format(date);
%>
```

body 中可以通过 jsp 表达式直接使用 java 代码：
curTime=<%=time %>

=======jsp 代码嵌套 html
body 中可以用 java 代码逻辑嵌套 html

```
<%  if(false){  %>
<p>true</p>
<%  }else{      %>
<p>false</p>
<%  }           %>
```

13.4.1.1. java 代码

<% **** %>
当前页面有效，页面关闭时销毁

13.4.1.2. 指令标识

<%@ 指令名 属性 1=“val1” %>

13.4.1.2.1. page

定义整个 jsp 页面相关属性，指令属性包括：
language： <%@ page language="java" %> 目前只包含 java
extends：所有 jsp 页面在执行之前都会被服务器解析成 servlet，而 servlet 是由 java 类定义的，所以 jsp 和 servlet 都可以继承指定的基类。该属性可以设置 jsp 页面继承的 java 类。
import：用于导入 java 代码使用的 java 类包 <%@page import="java.util.List" %>
pageEncoding：指定页面编码格式
contentType：设置 jsp 页面的 MIME 类型和字符编码 <%@ page contentType="text/html; charset=utf-8"%>

session：指定当前页面是否使用 session 会话对象，默认 true
buffer：设置 jsp 的 out 输出对象使用的缓冲区大小，默认 8kb，单位只能使用 kb。
autoFlush：设置 jsp 页面缓存满时是否自动刷新。默认值 true。如果使用 false，则缓冲满时抛出异常。
isErroPage：可以将当前 jsp 页面设置成错误处理页面来处理另一个 jsp 页面的错误。
errorPage：指定一个目标 jsp 页面来处理当前页面错误，目标 jsp 页面必须设置 isErroPage 属性为 true。该属性的值是一个 url。

13.4.1.2.2. include

可以引入另一个 jsp,将另一个 jsp 在当前页面展开合并为一个 jsp.
<%@ include file=”**.jsp” %>
被包含的 jsp 除了必要内容，只包含<%@ page pageEncoding="utf-8"%>即可

13.4.1.2.3. taglib

可以引入标签库

13.4.1.3. JSP 表达式

<%= 表达式 %>
表达式可以是任何完整的 java 表达式。结果被转为字符串

13.4.1.4. 注释

<%-- 注释 --%>
该注释方法在显示时是隐藏的

对代码片断的注释
<!-- <%= new Date() %> -->

动态注释
<!-- <%=new Date() %> -->

/** 说明内容
*/
在方法或变量上加入/**注释可以被 javadoc 获取，当鼠标移动到该方法或变量时会提示注释内容

13.4.1.5. 声明标识

<%! 全局变量或方法 %>
服务器执行 JSP 页面时，会将 JSP 页面转换为 Servlet 类，把页面中定义的全局变量和方法转换为类的成员变量和方法。

声明的标识一直有效，当前页面有效，服务器关闭时销毁

13.4.2. 动作标识

13.4.2.1. 包含文件

<jsp:include page="Def.jsp"></jsp:include> page 属性支持 jsp 表达式
<%@ include file="**.jsp" %>一样可以包含其它 jsp，file 属性不支持任何表达式，该指令是将文件原样插入，合成的最终文件再进行编译

不同点在于 jsp:include 是分开编译,是在不同页面执行进行返回.不存在变量同名问题.

13.4.2.2. 请求转发/参数传递

```
<jsp:forward page="login.jsp">
    <jsp:param value="admin" name="user"/>
</jsp:forward>
```

中转页面,直接跳转到 login.jsp ,将参数 user=admin 传递
等价于跳转时文件名后直接加 ?user=admin

13.4.3. 内置对象

Request 是客户端请求的封装，服务器用 reques 取出请求信息。服务器接受请求后添加信息到 response，用来返回给浏览器。

13.4.3.1. request

封装了由客户端传来的 html 请求

功能	函数
获取传递参数	String val = request.getParameter("x")
获取客户端地址	getRequestURL ()
获取客户端地址，不包括参数	getRequestURI()
获取请求方式(post/get)	getMethod()
获取 http 协议	getProtocol()
获取客户端 ip	getRemoteAddr()
获取客户端名称	getRemoteHost()
获取服务器名称	getServerName()
获取服务器端口	getServerPort()
获取客户端语言类型	java.util.Locale locale = request.getLocale(); String ch = ""; if(locale.equals(java.util.Locale.CHINA)) ch="china";

13.4.3.2. response

重定向网站
response.sendRedirect("https://www.baidu.com");
禁用缓存
response.setHeader("Cache-Control","no-store");
response.setDateHeader("Expires",0);
自动刷新
response.setHeader("refresh","1"); 每秒刷新一次
定时跳转
response.setHeader("refresh","2;https://www.baidu.com");
设置缓冲区 32kb

response.setBufferSize(32);

13.4.3.3. cookie

cookie 会放在客户机器，创建成功后每次打开网页都可以读取 cookie

======创建

```
<%
    Cookie cookie = new Cookie("name","jyl");
    cookie.setMaxAge(60*60*24*10);//有效期10天
    response.addCookie(cookie);
%>
```

======读取

```
<%
    Cookie[] cookies = request.getCookies();
    String val = "";
    if(cookies != null)
    {
        for(int i = 0 ; i < cookies.length;++i)
        {
            val = cookies[i].getValue();
            val = cookies[i].getName();
        }
    }
%>
```

13.4.3.4. session

session 可以保存用户状态，直到关闭浏览器，时间超时 session 对象会自动消失

设置属性：
session.setAttribute("name","xx");
获取属性（obj 对象）：
session.getAttribute("name")
移除属性
session.removeAttribute("name");
删除会话（删除后调用 session 对象会抛出异常）
session.invalidate();

13.4.3.5. application

保存所有应用程序中的公共数据，在服务器启动时自动创建，在服务器停止时销毁。

========读取初始化参数
web 应用目录下 WEB-INF 目录下 web.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                              http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
   version="3.0"
   metadata-complete="true">
<context-param>
     <param-name>val</param-name>
     <param-value>111</param-value>
</context-param>
</web-app>
```

====获取初始值

```
String    ret = application.getInitParameter("val");
```

====获取初始所有键值对

```
 java.util.Enumeration enema = application.getInitParameterNames();
 while(enema.hasMoreElements())
  {
      String name = (String)enema.nextElement();
      String value = application.getInitParameter(name);
 }
```

======设置和获取全局参数

```
application.setAttribute("a",1);//第二个参数是 obj
String str = application.getAttribute("a").toString();
```

13.4.3.6. out

向浏览器客户端输出信息

```
out.print(out.getBufferSize());
out.println("xxx");
```

13.4.3.7. pageContext

pageContext 容器内置对象,可以通过该对象获取 response/session/application/exception 等对象
包括 forward，setAttribute 等方法

13.4.3.8. config

通过 pageContext 对象的 getServletConfig()方法可以获取一个 config 对象,用来读取 web.xml 配置信息
getInitParameter()/getInitParameterNames()

13.4.3.9. page

page 对象代表 jsp 本身,只有在 jsp 页面内才合法.是包含当前 ServLet 接口引用的变量,this 关键字的别名.
page.getClass()返回当前 Object 类

13.4.3.10.exception

如果 jsp 页面中没有捕捉到异常，就会生成 exception 对象，并把 exception 传输到指定的错误页面。
指定的错误页面在 page 指令中设置 isErrorPage 属性为 true 的页面才可以被使用（一般的 jsp 页面中使用无法编译）。
<%@page errorPage="e.jsp" %>

<%@page isErrorPage="true" %>

13.4.4. javaBean

javaBean 类需要遵守规范：
提供一个默认的无参构造函数。
可能有一系列可读写属性,首字母必须小写
可能有一系列的"getter"或"setter"方法。

=====创建 bean 类

```
package com.test;
public class bb {
    private String val = null;
```

```
    public String getVal() {
        return val;
    }

    public void setVal(String val) {
        this.val = val;
    }
}
```

=====使用

```
<%--创建对象 objID --%>
<jsp:useBean id="ObjID" class="com.test.bb"/>

<%--设置对象 objID 的 val 属性 --%>
<jsp:setProperty name="ObjID"  property="val" value="aaabbbccc"/>

<%--显示 --%>
val = <jsp:getProperty name="ObjID"  property="val"/>
```

13.4.5. EL 表达式

开启 el 表达式（默认可能是关闭的）

```
<%@page isELIgnored="false" %>
```

${***}

等价于<%=request.getAttribute("***")%>

jsp 页面中输出$符号时:\${ 或者 ${"${"}

可以访问 pageContext 对象

除 0 后返回 Infinity,不返回错误

=======禁用

禁用后将原内容输出

禁用一个表达式:\${**}

禁用所有页面(web.xml):

```
    <jsp-config>
        <jsp-property-group>
           <url-pattern>*.jsp</url-pattern>
           <el-ignored>true</el-ignored>
```

```
        </jsp-property-group>
    </jsp-config>
```

====判空
判断对象是否是 null 或者””
```
<% request.setAttribute("nn", ""); %>
${empty nn}
```

====访问作用范围的隐含对象
```
<% request.setAttribute("nn", ""); %>
```
需要使用 requestScope.nn
不能通过 request.nn 访问到保存到 request 范围内的变量 nn

根据设置的访问范围进行获取
```
<jsp:useBean id="tt" scope="page" class="com.wt.tt"></jsp:useBean>
${ pageScope.tt.val}
```

====访问环境中的值
获取提交的标单中 txt 变量的值
```
${param.txt}
```

获取提交的标单中一组值
```
${paramValues.ck[0]}
${paramValues.ck[1]}
```

13.5. JSTL

13.5.1. 依赖

```
    <dependency>
       <groupId>javax.servlet</groupId>
       <artifactId>jstl</artifactId>
       <version>1.2</version>
    </dependency>
    <dependency>
       <groupId>taglibs</groupId>
       <artifactId>standard</artifactId>
       <version>1.1.2</version>
     </dependency>
```

13.5.2. 使用

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

<c:set var="pageurl" value="${pageContext.request.contextPath}"></c:set>	获取运行路径 使用： '${pageurl}/resources/icons/x.png'
<c:out value="xxx"/> <c:out value=”${aaa}”/> <c:out value=”${aaa}” default=”xxx”/>	输出字符串
<c:set var=”aaa” value=”xxx”/>	在 pageContext 中添加值
<c:set var=”a” value=”hello” scope=”session”/>	在 session 中添加值
<c:remove var="a" scope=”page”/>	删除 pageContext 中数据 无 scope 时删除所有域中的数据
<c:if test="${booleanVal}"> **** </c:if> <c:if test="${strval.equals('ss')}"> **** </c:if>	if 语句
<c:set var="score" value="${param.score }"/> <c:choose> <c:when test="${score > 100 || score < 0}">错误的分数：${score }</c:when> <c:when test="${score >= 90 }">A 级</c:when> <c:when test="${score >= 80 }">B 级</c:when> <c:when test="${score >= 70 }">C 级</c:when> <c:when test="${score >= 60 }">D 级</c:when> <c:otherwise>E 级</c:otherwise> </c:choose>	if/else if/else 结构
<c:forEach var="i" begin="1" end="10"> 。。。。 </c:forEach>	for(i =0 ; i < 10;++i)
<c:forEach items="${listObj}" var="obj"> </c:forEach>	for(obj : listObj) { var = obj.imageurl } 需要强转 url 为字符串时： src="'+url+'"

13.5.3. 自定义标签

13.5.3.1. Java 类

```
package com.test.domain;

import java.io.IOException;

import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class TeTag extends SimpleTagSupport {

    private String strParam;
    private int    intParam;

    public String getStrParam() {
        return strParam;
    }
    public void setStrParam(String strParam) {
        this.strParam = strParam;
    }
    public int getIntParam() {
        return intParam;
    }
    public void setIntParam(int intParam) {
        this.intParam = intParam;
    }



    @Override
    public void doTag() throws IOException
    {
        PageContext pc = (PageContext)getJspContext();
        JspWriter out = pc.getOut();
        out.print(strParam + "=====" + intParam);
    }
```

```
    public static String funEL(String val){
    return val + "===EL===";
    }
}
```

13.5.3.2. tld 文件

目录：/WEB-INFO/TeTag.tld

```
<?xml version="1.0" encoding="UTF-8" ?>

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
  version="2.0">

  <tlib-version>1.1</tlib-version>
  <short-name>ssname</short-name>
  <uri>*******</uri>


  <tag>
  <name>fun</name>
  <tag-class>com.test.domain.TeTag</tag-class>
  <body-content>empty</body-content>
  <attribute>
  <name>strParam</name>
  <required>true</required>
  <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
  <name>intParam</name>
  <required>true</required>
  <rtexprvalue>true</rtexprvalue>
  </attribute>
  </tag>



<function>
   <name>funEL</name>
   <function-class>com.test.domain.TeTag</function-class>
```

```
<function-signature>java.lang.String
funEL(java.lang.String)</function-signature>
</function>

</taglib>
```

13.5.3.3. Jsp 页面

```
<%@ taglib prefix="Te" uri="/WEB-INF/TeTag.tld" %>
<Te:fun strParam="aaaa" intParam="15"/>


<%@page isELIgnored="false" %>
${Te:funEL('abcd')}
```

13.6. Servlet

13.6.1. 创建

实现 Servlet 接口或者继承 HttpServlet 方法
init 方法和 destroy 方法为 Servlet 初始化与生命周期结束所调用。doGet 和 doPost 方法用于 http 的 get 与 post 请求。

==========创建 serverlet 类
创建 com.test.servletTe

```
package com.test;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class servletTe extends HttpServlet {
    private static final long serialVersionUID = 1L;
     private String message;

    /**
     * @see HttpServlet#HttpServlet()
     */
```

```
    public servletTe() {
        super();
        message = "servletTe";
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

      response.setContentType("text/html;charset=utf-8");
      response.setCharacterEncoding("utf-8");
      PrintWriter out = response.getWriter();
      String str = "我是网页内容";
      out.write(str);
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }

}
```

=========调用方式 1

web.xml 中加入以下配置：

```
  <servlet>
    <servlet-name>servletTe</servlet-name>
    <servlet-class>com.test.servletTe</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>servletTe</servlet-name>
    <url-pattern>/servletTe</url-pattern>
  </servlet-mapping>
```

url 访问/servletTe 时调用 serverlet

=========调用方式 2
直接在类上加入注解
@WebServlet("/servletTe")
或者@WebServlet(name="servletTe",urlPatterns="/servletTe")

13.6.2. 过滤器

Servlet 过滤器实现 Filter 接口，可以配置对指定页面进行过滤，当请求指定过滤页面时先调用过滤器的 doFilter

==========访问页面计数，每次请求页面时计数增加 1==========
==========配置

```
<filter>
       <filter-name>SF</filter-name>
       <filter-class>com.WebTest.SF</filter-class>
       <init-param>
          <param-name>count</param-name>
          <param-value>10</param-value>
       </init-param>
 </filter>
 <filter-mapping>
       <filter-name>SF</filter-name>
       <url-pattern>/index.jsp</url-pattern>
</filter-mapping>
```

在后面继续配置第二个<filter>同时定位到同一个<url-pattern>，可以做过滤链。

@WebFilter("/*") 指定所有页面都使用
@WebFilter("/index.jsp") 指定 index.jsp 使用过滤

```
import javax.servlet.annotation.WebInitParam;
@WebFilter(urlPatterns = "/index.jsp",initParams=
{@WebInitParam(name="aa",value="a")})
```

指定 index.jsp 使用过滤，设置初始参数 aa

==========类定义
包含 count 成员变量

init 函数中使用 fConfig 对象获取 web.xml 中<init-param>中配置的参数

```
String Count = fConfig.getInitParameter("count");
if(null == Count) return;
```

```
count = Integer.valueOf(count);
```

doFilter 函数中进行计数，并写入到 application 对象

```
++count;
HttpServletRequest req = (HttpServletRequest)request;
ServletContext context = req.getSession().getServletContext();
context.setAttribute("count", count);
```

==========index.jsp 中获取参数

```
<%=application.getAttribute("count")%>
```

13.6.3. 监听器

实现监听接口 HttpSessionBindingListener

public class SL implements HttpSessionBindingListener

java 文件中注解@WebListener 加入绑定

或者 web.xml 加入配置

```
  <listener>
    <listener-class>com.WebTest.SL</listener-class>
  </listener>
```

注解中可以加入描述

@WebListener("description")

在 jsp 页面中调用 session 对象的 setAttribute/removeAttribute 会触发 SL 的 valueBound/valueUnbound 方法

```
session.setAttribute("name", new SL());
session.removeAttribute("name");
```

13.6.4. 监听器上传文件

@MultipartConfig(location="D:/") //临时目录必须有访问权限

======上传界面

```
    <form action="SL" enctype="multipart/form-data" method="post">
        <br/>
        <input type="file" name="f1"/>
```

```
        <br />
        <input type="submit" name="upload" value="upload" />
    </form>
```

====== SL 类的 doPost

```
response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String path = this.getServletContext().getRealPath("/");
        javax.servlet.http.Part p = request.getPart("f1");
        if(p.getContentType().contains("image"))//只处理image文件
        {
            ApplicationPart ap = (ApplicationPart)p;
            String PathFileName = ap.getSubmittedFileName();
            int path_idx = PathFileName.lastIndexOf("\\") + 1;
            String                          fileName                          =
PathFileName.substring(path_idx,PathFileName.length());
            p.write("D:/workspace/" + fileName); //存储目录必须存在
            out.write("suc");
        }
        else
        {
          out.write("failed");
        }
```

13.7. spring

13.7.1. 地址

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>4.0.3.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>4.0.3.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>4.0.3.RELEASE</version>
</dependency>
```

13.7.2. 反向控制/依赖注入

反向控制 IoC（Inversion of Control），指实现必须依赖抽象，而不是抽象依赖实现。
依赖注入 DI（Dependency Injection），用来实现反向控制功能。包含三种实现方式：接口注入、Set 注入、构造注入。spring 实现 Set 注入和构造注入。

13.7.3. javabean 类的编写

定义一个接口（通过接口进行调用）

```
public interface AA {
	void show();
}
```

编写实现类 AAa

```
public class AAa implements AA{

	public void show() {
		System.out.println(msg);
		System.out.println(val);
		System.out.println(val2);
	}

	String val;
	String val2;
	public AAa(String co1,String co2)
	{
		val = co1;
		val2 = co2;
	}


	String msg;
	public String getMsg() {
		return msg;
	}
	public void setMsg(String msg) {
		this.msg = msg;
	}
}
```

编写实现类 AAb

```
public class AAb implements AA {
	public void show() {
		System.out.println(date);
	}

	Date date = null;
	public Date getDate() {
		return date;
	}
	public void setDate(Date date) {
		this.date = date;
	}
}
```

13.7.4. xml 编写

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
	http://www.springframework.org/schema/beans/spring-beans.xsd">

  <!--
  class 包含类的全路径，通过 property 使用 SetXX 方法设置成员变量 msg
  通过 constructor-arg 进行构造注入
  -->
  <bean id="idName1" class="inf.AAa" >
    <constructor-arg>
      <value>arg1</value>
    </constructor-arg>
    <constructor-arg>
      <value>arg2</value>
    </constructor-arg>
    <property name="msg">
      <value>xxx</value>
    </property>
  </bean>

  <!--
  Set 注入方式设置 AAb 类中的 date 对象
  可以使用 depends-on 强制初始化依赖的类 date
```

```
    -->
    <bean id="idName2" class="inf.AAb" depends-on="date">
        <property name="date">
            <ref bean="date"/>
        </property>
    </bean>
    <bean id="date" class="java.util.Date"></bean>


</beans>
```

13.7.5. 调用

```
import    org.springframework.context.support.FileSystemXmlApplicationContext;


FileSystemXmlApplicationContext ctx = new FileSystemXmlApplicationContext("setting.xml");
```

xml 放在项目根目录

```
AA a = (AA)ctx.getBean("idName1");
a.show();

AA b = (AA)ctx.getBean("idName2");
b.show();
```

13.8. springMVC

13.8.1. 框架搭建

13.8.1.1. 配置

13.8.1.1.1. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                                                xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd "
      version="2.5">
```

```
    <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

```
<mvc:resources mapping="/resources/**" location="/WEB-INF/classes/static/"/>
```

可以将 WEB-INF 下的 classes/static 目录映射到根目录的 resources 目录

web.xml 在 WEB-INF 下
定义了使用 springframework 进行监听和处理
默认使用当前目录下 applicationContext.xml 和 spring-servlet.xml

通过以下代码可以重定义 applicationContext.xml 位置
（classpath 表示 java 代码生成结果目录【WEB-INF\classes】）

```
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>
```

通过以下代码可以重定义 spring-servlet.xml 路径

```
<servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:spring-servlet.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
</servlet>
```

13.8.1.1.2. applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:p="http://www.springframework.org/schema/p"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:aop="http://www.springframework.org/schema/aop"
	xmlns:tx="http://www.springframework.org/schema/tx"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
		http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
		http://www.springframework.org/schema/context
		http://www.springframework.org/schema/context/spring-context-4.0.xsd
		http://www.springframework.org/schema/aop
		http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
		http://www.springframework.org/schema/tx
		http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">
</beans>
```

13.8.1.1.3. spring-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:p="http://www.springframework.org/schema/p"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:mvc="http://www.springframework.org/schema/mvc"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
		http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
		http://www.springframework.org/schema/context
		http://www.springframework.org/schema/context/spring-context-4.0.xsd
		http://www.springframework.org/schema/mvc
		http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

	<mvc:annotation-driven/>
	<context:component-scan base-package="com.test.controller"/>
	<mvc:default-servlet-handler/>
	<bean                                                    id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
		<property name="prefix" value="/WEB-INF/jsp/"/>
		<property name="suffix" value=".jsp"/>
```

```
        <property name="order" value="20"/>
    </bean>
</beans>
```

在 com.test.abc 包目录下找注解类进行对应（多包扫描用逗号分隔 com.abc,com.cde）

在/WEB-INF/jsp/下对应 Controller 控制跳转的 jsp【Controller 操作的根目录】

13.8.1.2. 使用

13.8.1.2.1. Controller

com.test.controller 包下建立类

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class mainController {
        @RequestMapping(value="/home",method=RequestMethod.GET)
        public ModelAndView home(){
                ModelAndView mv = new ModelAndView();
                mv.addObject("msg", "spring mvc");
                return mv;
        }
}
```

13.8.1.2.2. jsp 页面

WEB-INF/jsp/目录下建立 home.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<%@page isELIgnored="false" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Insert title here</title>
</head>
<body>
```

```
    ${msg}
</body>
</html>
```

13.8.2. Controller

13.8.2.1. 响应路径

```
 @RequestMapping(value="/home",method=RequestMethod.GET)
   public ModelAndView home(){
                ModelAndView mv = new ModelAndView();
                mv.addObject("msg", "spring mvc");
                return mv;
   }
```

函数无参数时，使用：jsp 目录/home.jsp,

页面请求为 ..../项目名/home 时，则访问 home.jsp

```
<jsp:forward page="/home"/>
```

通过框架跳转到 home.jsp

```
@RequestMapping(value="/run",method=RequestMethod.GET)
public String fun(){
      return "home/index";    //使用：jsp 目录/home/index.jsp
}
```

地址组合:(页面请求为 ..../项目名/abc/def)

```
@Controller
@RequestMapping(value="/abc")
public class mainController {
        @RequestMapping(value="/def",method=RequestMethod.GET)
        public ModelAndView home(){
        }
}
```

13.8.2.2. 参数

函数的参数个数和类型可以任意
spring 可以根据参数类型自动将请求中包含的内容传入函数。
可以添加一次请求中包含的任意对象，使用了反射，保持和界面中参数名称一致即可。

参数	说明
可以包含 HttpServletRequest request 和 HttpServletResponse response 及 HttpSession	<dependency> <groupId>javax.servlet</groupId> <artifactId>servlet-api</artifactId> <version>2.5</version> <scope>provided</scope> </dependency> ======获取 get 或 post 请求中包含的参数值 String title = request.getParameter("title"); ======设置变量值 HttpSession session = request.getSession(true); session.setAttribute("username",loginUser); 界面获取： <% String sessionUsername = (String)session.getAttribute("username"); %>
@RequestParam("valA") Integer val	将 jsp 界面的参数 count 或者 ajax 请求中的 json 直接解析出对应的值 val， 等价于 int Val = Integer.valueOf(request.getParameter("valA"));
@RequestMapping("/login/{username}") public String login(@PathVariable("username") String username)	从请求路径中获取参数
@RequestBody String param	获取请求对象整个字符串数据

13.8.2.3. 返回值

返回值类型	作用
String	表示返回 jsp 文件的名字

<table>
<tr><td></td><td>return "abc.jsp";</td></tr>
<tr><td rowspan="4">ModelAndView</td><td>将 jsp 文件组合数据进行返回</td></tr>
<tr><td>参数名-参数值传递
ModelAndView mv = new ModelAndView();
mv.addObject("msg", "spring mvc");
return mv;</td></tr>
<tr><td>实体类传递：
类的成员变量必须是小写开头，变量有 get 和 set 方法

ModelAndView mv = new ModelAndView("/run/index");
DataStu dd = new DataStu();
dd.setId(1234);
mv.addObject(dd);
return mv;

jsp 页面中使用[将类名前连续的大写全变为小写]
${dataStu.id}

<form method="post">
<input type="text" name="id"/>
<input type="submit" />
</form>
代码获取[表单内容中包含和实体类相同的变量名]
public ModelAndView fun(DataStu data){
Integer d = data.getId();
}</td></tr>
<tr><td>List 传递：
ModelAndView mv = new ModelAndView();
List<DataStu> datObj = UserService.fun();
mv.addObject("listDat", datObj);
return mv;

<%@taglib prefix="c"
uri="http://java.sun.com/jsp/jstl/core"%>
<c:forEach items="${listDat}" var="dat">
${dat.id}
"${dat.id}"
</c:forEach></td></tr>
<tr><td>void</td><td>通过以下代码直接创建内容：
response.setContentType("text/html;charset=utf-8");</td></tr>
</table>

	//response.setContentType("text/plain;charset=utf-8"); response.setCharacterEncoding("utf-8"); PrintWriter out = response.getWriter(); String str = "我是网页内容"; out.write(str);
@ResponseBody JavaBeanClass fun()	返回结果不会被解析为跳转路径，而是直接写入 HTTP response body 中 将 javaBean 对象组装成 json 返回，等价于用 response 直接写入 json 对象。 需要依赖： <dependency> <groupId>com.fasterxml.jackson.core</groupId> <artifactId>jackson-databind</artifactId> <version>2.4.3</version> </dependency> 可直接调用： ObjectMapper objectMapper = new ObjectMapper(); String json = objectMapper.writeValueAsString(javaBeanObj);

13.8.3. javabean

spring-servlet.xml 中构造全局对象

构造函数：

```
<bean id="devSer" class="*** ">
    <constructor-arg value="1"></constructor-arg>
    <constructor-arg value="2"></constructor-arg>
</bean>
```

13.8.4. WEB-INF

resources 目录下内容会拷贝到 WEB-INF\classes

WEB-INF 目录下的文件无法直接通过 URL 访问

通过在 web.xml 中配置，可以映射目录下的文件到根目录下，使其可以访问

spring-servlet.xml 中：
<mvc:resources mapping="/resources/**" location="/WEB-INF/classes/"/>
/resources/icon/a.png 映射为/WEB-INF/classes/icon/a.png

13.8.5. service

applicationContext.xml 中：
<context:component-scan base-package="com.abc.service"/>

扫描 com.abc.service 下所有的 service 类(@Service)，包装为 javabean

13.9. mybatis

13.9.1. msql 操作

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class dbOpt {
	Connection conn;
	Statement stmt;
	ResultSet result;


	public dbOpt(String DBName, String DBUser, String DBPass){
		try {
			Class.forName("com.mysql.cj.jdbc.Driver");//com.mysql.jdbc.Driver
			String connMsg = "jdbc:mysql://localhost:3306/" + DBName + "?user=" + DBUser +
"&password=" + DBPass;
			conn = DriverManager.getConnection(connMsg);
		}catch(Exception e){System.out.println(e.toString());}
	}


	public ResultSet Query(String SQLQuery){
		try {
			stmt = conn.createStatement();
			result = stmt.executeQuery( SQLQuery );
		}
		catch( Exception e ){System.out.println(e.toString());}
```

```
        return result;
    }
    public void exec(String SQLQuery){
        try {
            stmt = conn.createStatement();
            stmt.execute( SQLQuery );
        }
        catch( Exception e ){System.out.println(e.toString());}
    }


    public void close(){
        try {
            stmt.close();
            conn.close();
        }
        catch(Exception e){System.out.println(e.toString());}
    }
}
```

==========使用

```
dbOpt    db = new dbOpt("testdb","root","root");
ResultSet rs =db.Query("select * from testTable");
while(rs.next()){
    int id = rs.getInt("id");
    System.out.println(id);
}

db.exec("insert into testTable(id) values(23)");
```

13.9.2. 数据库表

数据库 testdb 中表 user 包含 id 和 name
用户名和密码都是 root

13.9.3. 使用

13.9.3.1. 依赖

```
    <dependency>
```

```
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.4.6</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.11</version>
        </dependency>
```

13.9.3.2. 实体类

创建包 com.test.domain

```
package com.test.domain;

public class User {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

13.9.3.3. mapper

创建包 com.test.mapper

13.9.3.3.1. userMapper 接口

```
package com.test.mapper;

import com.test.domain.User;

public interface userMapper {
    public User fun(int id) throws Exception;
}
```

13.9.3.3.2. userMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.test.mapper.userMapper">
    <select id="fun" parameterType="int"
        resultType="com.test.domain.User">
        select name from user where id=#{id}
    </select>
</mapper>
```

13.9.3.4. cfg.xml

将配置放在 com.test.mapper

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver" />
                <property name="url"
value="jdbc:mysql://localhost:3306/testdb?useUnicode=true&amp;characterEncoding=UTF
-8&amp;serverTimezone=GMT%2B8" />
                <property name="username" value="root" />
                <property name="password" value="root" />
            </dataSource>
        </environment>
    </environments>
  <mappers>
    <package name="com/test/mapper"/>
  </mappers>
</configuration>
```

告知映射文件方式如果使用：

```
<mappers>
```

```
        <mapper resource="com/test/mapper/userMapper.xml"/>
</mappers>
```
则可以不创建 userMapper 的接口类

13.9.3.5. 调用

```
import java.io.IOException;
import java.io.Reader;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import com.test.domain.User;

public class App{

    public static void main(String[] args) throws IOException {
        Reader reader = Resources.getResourceAsReader("com/test/mapper/cfg.xml");
        SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(reader);
        SqlSession session = sessionFactory.openSession();


//        String statement = "com.test.mapper.userMapper.fun";
//        int id = 0;
//         User user = session.selectOne(statement, id);
//        session.commit();
//        System.out.println(user.getName());

          int idParam = 0;
          userMapper mapper = session.getMapper(userMapper.class);
          User userRet =    mapper.fun(idParam);
          System.out.println(userRet.getName());
    }
}
```

13.9.3.6. MapperSql

使用<![CDATA[……]]> 可以在其中包含小于号

```
<select id="函数名"   resultType="输出参数类型"   parameterType="输入参数类型" >
```

```
        <![CDATA[   查询语句    ]]>
</select>
```

#{}将传入的参数当成一个字符串，会给传入的参数加一个双引号,能够很大程度上防止 sql 注入
${}将传入的参数直接显示生成在 sql 中，不会添加引号

```
<sql id="limit_sql">
      <where>
          <if test="id != null and   id != ''">
               TempTableName.id=#{id}
          </if>
     </where>
</sql>
<select id="getlimit" parameterType="map" resultType="map">
          select * from   TempTableName
          <include refid="limit_sql" />
          ORDER BY   id DESC LIMIT #{start}, #{limit}
</select>
```

13.9.3.7. parameterType

类型	用法
单个参数	可以不写 parameterType(嵌套语句中引用变量)，也可以限定参数类型 如： parameterType="int" ， parameterType="com.xxx.xxx" 如果是基本类型，直接引用变量：#{随便写} 如果是实体类，写成如#{id}，会自动在实体类中根据 id 字段拿出对应值 在嵌套语句中引用变量情况下，需要在函数声明时加@Param
多个参数	不用写 parameterType 或者写 parameterType="map" 需要在对应函数中为参数加@Param 注解，以标识 xml 引用的变量名 如： public User fun(@Param("name") String aa,@Param("age") int bb) throws Exception; xml 中：select name from user where name=#{name} AND age=#{age}

13.9.3.8. resultType

类型	resultType	记录条数	说明
对象/List 类型	resultType="Integer" resultType="com.**"	单条/多条	基本类型直接将结果赋值 实体类型时，将记录取出，并根据字段名匹配对象中的变量名进行赋值 多条数据时，返回值函数声明需要写成 List<ClassName>
Map 类型	resultType="map"	单条/多条	返回值是 json [{key1=val1, key2=value2}, {key1=val3, key2=val4}] 返回值可以用 List<任意包装类型>接收
Map 类型	Map 中 value 的类型，如：resultType="com.**"	多条	需要在 Mapper 的 java 文件中定义 map 中 key 的类型及对应的字段名 如以下代码定义使用 age 这个字段（数字类型）作为 key： @MapKey("age") public Map<Integer,User> fun() throws Exception;

13.9.4. 结合 springmvc

13.9.4.1. 依赖

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.0.3.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.0.3.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
```

```
        <artifactId>spring-webmvc</artifactId>
        <version>4.0.3.RELEASE</version>
</dependency>
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
        <version>4.0.3.RELEASE</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>4.0.3.RELEASE</version>
</dependency>
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>4.0.3.RELEASE</version>
</dependency>


<dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.4.6</version>
</dependency>
<dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>1.3.2</version>
</dependency>
<dependency>
        <groupId>org.mybatis.caches</groupId>
        <artifactId>mybatis-oscache</artifactId>
        <version>1.0.2</version>
</dependency>

<dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>1.1.10</version>
</dependency>
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
```

```
            <version>8.0.11</version>
        </dependency>
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.9.1</version>
        </dependency>
```

13.9.4.2. applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.0.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">


    <tx:annotation-driven />

    <bean id="dataSource"
        class="com.alibaba.druid.pool.DruidDataSource" init-method="init"
        destroy-method="close">
        <property name="url"

    value="jdbc:mysql://127.0.0.1:3306/testdb?useUnicode=true&amp;characterEncoding=UTF
-8&amp;serverTimezone=GMT%2B8" />
        <property name="username" value="root" />
        <property name="password" value="root" />


        <!-- 配置初始化大小、最小、最大 -->
```

```
        <property name="initialSize" value="10" />
        <property name="minIdle" value="5" />
        <property name="maxActive" value="50" />
        <!-- 配置获取连接等待超时的时间 -->
        <property name="maxWait" value="60000" />
        <!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
        <property name="timeBetweenEvictionRunsMillis" value="60000" />
        <!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->
        <property name="minEvictableIdleTimeMillis" value="300000" />
        <!-- 如果用 Oracle，则把 poolPreparedStatements 配置为 true，mysql 可以配置为 false。
分库分表较多的数据库，建议配置为 false。 -->
        <property name="poolPreparedStatements" value="false" />
    </bean>


    <context:component-scan base-package="com.test">
        <context:include-filter type="aspectj"
            expression="com.test.service..*" />
    </context:component-scan>

    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.test.mapper" />
    </bean>

    <bean id="sqlSessionFactory"
        class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="mapperLocations"
            value="classpath:com/test/mapper/*.xml" />
        <property name="typeAliasesPackage"
            value="com.test.domain" />
    </bean>


    <bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
        p:dataSource-ref="dataSource" />

    <bean id="jdbcTemplate"
        class="org.springframework.jdbc.core.JdbcTemplate"
        p:dataSource-ref="dataSource" />


</beans>
```

classpath 指:WEB-INFO/classes
java 类编译后会按照包的完整名称拷贝到这个目录

com.test.service..*表示扫描 com.test.service 或 com.test.serviceAb 这类名字的包

13.9.4.3. 实体类

创建包 com.test.domain

```
package com.test.domain;

public class User {
	private String name;
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

13.9.4.4. Mapper

创建包 com.test.mapper

13.9.4.4.1. userMapper 接口

```
package com.test.mapper;

import com.test.domain.User;

public interface userMapper {
	public User fun(@Param("id") int id) throws Exception;
}
```

可以使用@Param 标志从外部传来的参数为 Mapper.xml 使用

13.9.4.4.2. userMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.test.mapper.userMapper">
    <select id="fun" parameterType="int"
        resultType="com.test.domain.User">
        select name from user where id=#{id}
    </select>
</mapper>
```

13.9.4.5. Service

创建包 com.test.service

13.9.4.5.1. userService 接口

```
package com.test.service;
import com.test.domain.User;

public interface userService {
    public User fun(int id);
}
```

13.9.4.5.2. userServicempl

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.test.domain.User;
import com.test.mapper.userMapper;
import com.test.service.userService;

@Service
public class userServicempl implements  userService{
    @Autowired
```

```
    private userMapper UserMapper;

    public User fun(int id) {
        User user = null;
        try {
            user = UserMapper.fun(id);
        }catch(Exception e)
        {

        }
        return user;
    }
}
```

13.9.4.6. Controller 调用

```
    @Autowired
    private userService UserService;


 String str = UserService.fun(0).getName();
```

13.9.4.7. 日志输出

13.9.4.7.1. 依赖

```
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>
```

13.9.4.7.2. log4j.properties

classpath 目录下

```
log4j.rootLogger=off,Console,File
log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.Target=System.out
log4j.appender.Console.layout = org.apache.log4j.PatternLayout
log4j.appender.Console.layout.ConversionPattern = [%p] [%d{yyyy-MM-dd
```

HH\:mm\:ss}][%c]%m %l%n

log4j.appender.File = org.apache.log4j.RollingFileAppender
log4j.appender.File.File = C://logs/ssm.log
log4j.appender.File.MaxFileSize = 10MB
log4j.appender.File.Threshold = ALL
log4j.appender.File.layout = org.apache.log4j.PatternLayout
log4j.appender.File.layout.ConversionPattern = [%p] [%d{yyyy-MM-dd HH\:mm\:ss}][%c]%m %l%n

13.9.4.7.3. mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <settings>
        <setting name="logImpl" value="STDOUT_LOGGING" />
    </settings>
</configuration>
```

13.9.4.7.4. applicationContext 加入配置

```
<bean id="sqlSessionFactory"  加入属性

<property name="configLocation" value="WEB-INF/mybatis-config.xml"></property>
```

13.10. thymeleaf

```
public String index(ModelMap map) {
    map.addAttribute("abc", "吃饭睡觉");
    return "index";
}
```

```
<h1 th:text="${abc}">*****</h1>
```

```
<script th:inline="javascript">
```

```
    /*<![CDATA[*/
    var type = /*[[${abc}]]*/;
    alert(type);
    /*]]>*/
</script>
```

第十四章 JAVASCRIPT

14.1. js

14.1.1. 变量定义

a = 1.3；
var i=10;
var j="abc";

var string = String(100);//类型转换

数字与字符串相加为字符串

14.1.2. 基本数据类型

14.1.2.1. Number

-5, -5.5 , 0xd 都是数字类型
测试变量是非数字：
isNaN("4")会得到返回值 false，因为 4 是数字
isNaN("four")返回 ture，因为 four 不是数字

数学函数：
Math.PI 得到圆周率
Math.pow(x,y); 返回 x 的 y 次幂

常量：
Infinity
Number.MAX_VALUE
Number.MIN_VALUE
Number.NEGATIVE_INFINITY 负无穷大
Number.POSITIVE_INFINITY 正无穷大

14.1.2.2. 字符和字符串

‘a’　‘abc’　“abc”

14.1.2.3. Boolean

Boolean b=true;

14.1.2.4. null

代表没有值，不等价于空

14.1.2.5. undefined

是一种状态

14.1.2.6. 数据转换

parseInt("123", 10) 字符串转整数，第二个参数默认 10 进制
parseFloat() 字符串转浮点数
String(obj) 转换为字符串

14.1.3. 函数

function fun(*，*){.....}

参数无类型，只需要标识

返回值如果有需要直接写

一个函数返回值时，如果函数没有返回，则接收为 undefined

function fun(a,b)

{

```
        alert(a);
}
fun(12);   // 调用时可以直接缺省参数
```

14.1.3.1. 带参函数

function ab(txt){.....}
调用时：onclick=”ab(‘hello’)”
外部有双引号时，内部用单引号

14.1.3.2. 默认参数

定义函数时可以不定义参数，函数中使用 arguments 数组直接拿参数

```
a = arguments[0];
b = arguments[1];
```

14.1.3.3. 返回值

function ab(){ return “hello”;}
调用时：document.write(ab());

14.1.3.4. 匿名函数

也就是函数直接量

```
var i = function (x, y) { return x + y }
alert(i(1, 1));
```

14.1.3.5. 闭包

嵌套函数可以访问外部函数的变量

```
function f() {
    var num = 10;
    function show() {
        alert(num);
    }
    return show();
}
```

f();//调用

14.1.4. 事件

==========标签中调用

```
<script language = "javascript" type ="text/jscript">
        function check()
        {
            if (f1.user.value == "")
            {
                alert("请输入");
                f1.user.focus();
            }
            else
            {
                f1.submit();
            }
        }
</script>
```

Name 或 id 属性都可以用.语法访问到

```
<form name="f1" action="b.html" method="post" target="_blank">
 用户名:<input type="text" name="user" />
 <input type="button"     onclick = "check()" value="按钮名"/>
 </form>
```

==========放在元素下

Js 代码加载必须在相应标签加载完后执行

```
 document.getElementById('bt').onclick = function() { alert(1); };
```

或者

```
document.getElementById('bt').addEventListener('click', function() { alert(1); }, false);
```

14.1.5. 对象

14.1.5.1. 使用函数构造

```
    function People(name)
{
     this.name = name;
     this.show = function(age){alert(this.name + age);};
}


    var people = new People("nn");
people.show(10);
```

14.1.5.2. 声明后添加

```
    var people = new Object();
people.name = "xx";
people.show = function(age){alert(this.name + age);};

people.show(12);
```

14.1.5.3. json 构造

```
    var people = {name:"aa",show: function(age){alert(this.name + age)}};
people.show(10);
```

并列数据之间使用逗号（,）
映射用冒号（:）
并列数据集合（数组）用方括号（[]）
映射的集合（对象）用大括号({})

14.1.6. 操作符

14.1.6.1. in

用来鉴定一个给定的属性是否包含在一个对象内，in 搜索的是属性是否存在。
var obj =

```
{
	a: "aaa",
	b: "bbb"
};
if ("a" in obj) {
	alert("xx");
}
```

14.1.6.2. instanceof

用来测试一个给定的表达式（通常是一个变量）是否是类的一个对象。

```
var date = new Date();
if (date instanceof Date) {
	alert("xx");
}
```

14.1.6.3. delete

将对象的一个值删除或者变为未定义

```
delete N[0];    //删除数组中第一个元素
```

14.1.6.4. typeof

返回操作数的变量类型

```
var i = 10;
if (typeof (i) == "number") {
	alert("xx");
}
```

14.1.6.5. void

常用来提交一个表单或打开一个新的窗口

```
void(window.open());
```

14.1.7. 错误处理

```
try{...执行代码...
    throw(“ab”);
```

}
catch(err){..处理错误.
　　if(err==”ab”){.....}.
}
Throw 抛出的异常参数，由 err 变量接收

14.1.8. 执行代码

将字符串解释为 js 代码执行
eval("var i = 10; alert(i);")

14.1.9. 浏览器调用

14.1.9.1. 写入 HTML

<script>
document.write("<h1>xxxxxx</h1>");
</script>
包含函数的脚本位于文档的 head 部分，这样可以确保调用函数前，脚本已经载入。
包含执行内容的脚本一般放在 body 末尾，有利于载入，防止阻塞造成的页面载入慢。

在 xhtml 上，<和&被解释为 xml，为了避免干扰，在 xhtml 中使用
<script type="text/javascript">

14.1.9.2. JS 文件载入

将脚本保存为.js，可以被多个文件调用
JS 文件中：
document.write("<h1>xxxxxx</h1>");
function fun(){alert("xx");}

Html 文件中：

```
<script language = "javascript" type ="text/jscript" src = "a.js">
</script>

<button onclick = "fun();">btn</button>
```

14.1.9.3. 写入浏览器

在浏览器的地址栏直接写入代码
javascript:alert("hello");

14.1.9.4. JS 文件交互

通过<script>标签引入的两个 js 之间不能互相调用

a.js 需要调用 b.js 中的函数和变量时，body 中载入 b.js
在 body 节点后用 scrpit 标签引入 a.js

```
newElement = document.createElement("script");
newElement.setAttribute("src","b.js");
newElement.setAttribute("type","text/javascript");
document.body.appendChild(newElement);
```

14.2. js 对象

14.2.1. json

{"key1":obj1,"key2":obj2}
[obj1,obj2]
obj 可以是{}、[]，或其他类型数据

JSON.parse()　　将 JSON 字符串转换为 JavaScript 对象。
JSON.stringify()　　将 JavaScript 值转换为 JSON 字符串。

14.2.2. console

```
console.log("xxx");
```

14.2.3. 数组

14.2.3.1. 构造

1）var N = ["abc",2];
2）var N = new Array();
　　N[0] = "abc";
　　N[1] = 2;

3）var N = new Array("abc",2);

14.2.3.2. 数组操作

```
arr.pop();      在尾部删除元素，返回被删除元素
arr.push(7);    在尾部添加元素，返回新数组长度

arr.shift();      在头部删除元素
arr.unshift(7);   在头部添加元素

var str =  arr.join();    将数组内容变成字符串

var arr3 = arr.concat(arr2);   连接两个数组
```

14.2.3.3. 遍历

```
for in 对于数组对象属性进行循环
for(index in N){
    alert(N[index]);
}


var   CC = {a:1,b:2};
for(var x in CC)
{
    var key   = x;
    var val = CC[x];
}
```

14.2.4. String

```
Hello \
Abc          使用\可以对字符串进行折行处理
```

“abc”，”A”都是字符串类型
"***'str'***", '***"str"***' 字符串用单引号或者双引号包含，单引号内可以包含双引号，双引号内可以包含单引号。
转译字符：\”

功能	用法
length	"***".length 可以获取字符串长度
substr	substr(1,2) 返回下标 1 开始，长度 2 的字

	符串。第二个参数可省略
indexOf	indexOf("bc")　从字符串开头查找，返回 bc 所在的位置下标
lastIndexOf	lastIndexOf("bc")　从字符串末尾查找，返回 bc 所在的位置下标
split	将字符串根据-分割 var arr = str.split("-"); for(i = 0 ; i < arr.length; ++i){}
replace	匹配 str 中 xcc，全部替换成##。 /g 表示全部替换，如果不加则替换匹配的第一个 var reg = /xc{2}m/g; var ss = str.replace(reg,"##");
正则表达式	var str = "http://www.abc.org"; var Regex = /http:\/\/w+\.(.*)/g;　//正则表达式　　/g 表示匹配全部 var result = Regex.exec(str); //result 为一个字符串数组，第一个元素是源字符串，后面的元素为正则表达式中每个（）内的内容 String 的 match,search,replace 可以使用正则表达式

14.2.5. Date

date = new Date();　获取当前时间

new Date(2010,1,1);　构造，小时，分钟，秒，毫秒缺省

date.getSeconds()　获取秒数

date.getFullYear()获取年，getYear()获取的是 19**年差

显示时间

```
<p id = "clock"></p>
function show()
```

```
{
    var now = new Date();
    clock.innerHTML = now.getHours() + ":" + now.getMinutes() + ":" + now.getSeconds();
}
window.setInterval("show()",1000);
```

14.2.6. 浏览器对象

14.2.6.1. window

window 对象是一个全局对象，表示浏览器目前正打开的窗口，是其他对象的顶层对象，所以可以省略名称，直接调用

alert	弹出提示框 alert(....);
location	location.href = "page"
location.host	当前页面的地址，如：127.0.0.1:8080
confirm	弹出确认框,确认返回 true confirm(...);
prompt	弹出提示对话框，并要求输入一个字符串 defster 是要求输入位置的默认值 prompt("xx","defstr");
open	var windowVar = open("b.html","target","width=500,height=200"); 执行成功返回新窗口对象。第一个参数为 url 地址，空字符串则打开空窗口。第二个参数制定新窗口名称，该名称可作为<a>和<form>的 target 属性。第三个参数可选。
close	关闭浏览器窗口 close();

setInterval clearInterval	定时器，1s 调用 1 次 fun setInterval("fun",1000); 清除定时器，id 从 1 开始，是每个定时器设置的先后次序 clearInterval(1);
setTimeout clearTimeout	1s 后执行 fun 函数 setTimeout("fun()",1000); 取消代码延迟 clearTimeout(1);
print	调用浏览器打印机 print();
moveTo moveBy	移动浏览器到指定位置 moveTo(1,1); 移动偏移量 moveBy(-10,100);
scrollTo	移动右侧和下方滚动条
resizeTo	设置窗口宽高 resizeTo(500, 500);

14.2.6.2. DOM

文档对象模型（DOM）document 对象提供了一种访问和修改 HTML 文档内容的方法。
DOM 是树形，对于 html，树根是<html>标记，<head>和<body>是枝干

14.2.6.2.1. document 方法

write	document.write(" xxxxxx "); 直接写 html 文件，如果是在开始载入，则会写在文件头。如果是中间调用则会清空文件
getElementById	通过 id 获取一个标记对 innerHTML 表示标记对在页面显示的内容 通过标记对中 id 属性获取，如 <p

直接用 id 也可以代表一个标记对	id="id1"></p> var x = document.getElementById("id1"); x.innerHTML = "xfasdfsdf"; //修改标签内的内容//也可以使用 id1.innerHTML alert(x.tagName);
getElementsByTagName	通过标记对名获取一组标记对 获取所有 p 标签，遍历获得每个的内容 var content = document.getElementsByTagName("p"); for(i = 0 ; i < content.length; ++i) { alert(content[i].innerHTML); }
createElement	创建标记对 根据标签名 td 创建 td 标记对元素 td_con = document.createElement("td");

14.2.6.2.2. 节点方法

setAttribute	设置标记对属性值 **.setAttribute("bgcolor","red");
removeAttribute	删除属性 **.removeAttribute("bgcolor");
appendChild	将 con 加入到**节点下 **.appendChild(con);
insertBefore	在子节点 refChil 前面插入新节点 newElement **.insertBefore(newElement,refChil);
replaceChild	将子节点 odl 替换成 newElement **.replaceChild(newElement,old)
removeChild	删除子节点 ch **.removeChild(ch);
cloneNode	复制**节点，包括属性。参数为 true 会通过递归方法克隆子节点 newNode= **.cloneNode(true);
hasChildNodes	判断一个元素是否有子节点（标记对中没有任何内容，包括空白） 文本节点和属性节点不可能再包含子节点，返回 false nodeType

	1 元素节点类型 2 属性节点类型 3 文本节点类型

14.2.6.2.3. 遍历所有节点

```
var elemList = "";
function getElement(node)
{
    var total = 0;
    if (node.nodeType == 1)  //检查是否是 elment 对象
    {
        total++;
        elemList =  elemList + node.nodeName + "  ";
    }

    var chil = node.childNodes;
    for(var m = node.firstChild; m != null; m = m.nextSibling)
    {
         total += getElement(m);
    }


    return total;
}

var num = getElement(document);
alert("包含" + num + "个标记" + " " + elemList);
```

14.2.6.2.4. HTML 集合

```
document.anchors   包含所有 a 元素的组(具有 name 属性的参数)
document.forms     包含所有 form 元素的组
document.images    包含所有图像元素的组
document.links     包含所有具有 href 属性的 a 元素的值
```

14.3. function

14.3.1. 获取尺寸

BODY 对象宽度
document.body.clientWidth

可见区域宽度
document.documentElement.clientWidth

14.3.2. canvas 绘图

```
<canvas id="canv"></canvas>
```

```
var canvas = document.getElementById('canv');
var ctx = canvas.getContext("2d");
```

14.3.2.1. 铅笔绘图

```
        ctx.strokeStyle = "#f00";
    ctx.lineWidth = 3;

    function draw(sx, sy, ex, ey) {
        //ctx.clearRect(0, 0, 600, 600);
        //ctx.beginPath();//执行 clearRect 和 beginPath 则重绘

        ctx.moveTo(sx, sy);
        ctx.lineTo(ex, ey);
        ctx.stroke();
    }

    var oX, oY;
    var flag = false;
    canvas.onmousemove = function(e) {
        if(!flag) return;

        draw(oX, oY, e.pageX, e.pageY);
        oX = e.pageX;
```

```
        oY = e.pageY;
    }
    canvas.onmousedown = function(e) {
        flag = true;
        oX = e.pageX;
        oY = e.pageY;
    }
    canvas.onmouseup = function(e) {
        flag = false;
    }

    onmouseup = function(e)
    {
        flag = false;
    }
```

14.3.2.2. 绘制矩形

```
ctx.fillStyle = "red";
ctx.strokeStyle = "red";
ctx.lineWidth = 5;
ctx.fillRect(0,0, 300, 200);
ctx.strokeRect(0,0,300,200);
```

14.3.2.3. 绘制图片

```
    var img = new Image();
    img.src = "1.jpeg";
    img.onload = function()
    {
        ctx.drawImage(img,0,0);   //从左上角开始绘制图像
        ctx.drawImage(img,10,20,150,200);  //从指定坐标(10,20)开始绘制图像，
并设置图像宽和高 指定这些参数可以使得图像可以缩放

        //裁剪一部分图像放在左上角，并调整尺寸
        //ctx.drawImage(img,90,80,100,100,0,0,120,120);
    }
```

14.4. nodejs

14.4.1. 配置

安装包下载：http://nodejs.cn/download/

安装后检查
node --version
npm -v

14.4.2. npm

当不在 nodejs 的目录（node_modules 同级目录）运行 npm
就会安装在 C:\Users\用户名\node_modules 目录中

安装 npm 镜像，使用 cnpm 代替 npm
npm install -g cnpm --registry=https://registry.npm.taobao.org

安装 mysql 模块：
cnpm install mysql

14.4.3. 服务搭建

```
var http = require('http');
var url = require("url");
http.createServer(function (request, response) {
    var pathname = url.parse(request.url).pathname;
        response.writeHead(200, {'Content-Type': 'text/plain;charset=utf-8'});
        response.end('*********');
}).listen(8080);
console.log('Server running at http://127.0.0.1:8080/');
```

将内容写入 ser.js 文件
命令行启动服务： node ser.js

14.4.4. 参数获取

```
var parseObj = url.parse(request.url, true);
```

```
request.method          //请求方式 GET/POST
parseObj.pathname     //无参数的 url 路径
request.headers["content-type"]      //消息报头中的 key-val
parseObj.query['x']      //参数 x 的值
```

```
var qs = require('querystring');
//获取 post 参数
if("POST" == request.method.toUpperCase())
{
var postData = "";
request.addListener("data", function (data) {
postData += data;
});
request.addListener("end", function () {
var query = qs.parse(postData);
console.log(query['passWord']);
});
}
```

14.4.5. 读文件

```
var fs = require("fs");

fs.readFile(pathname, function (err, data) {
        if (err) {
            console.log(err);
            response.writeHead(404, { 'Content-Type': 'text/html' });
        } else {
            response.writeHead(200, { 'Content-Type': 'image/jpeg' });
            response.write(data);
        }
        response.end();
});
```

14.4.6. 模块调用

fileA.js 内容：

```
function run(param)
{
    console.log(param);
```

}
exports.run = run;

调用时：
var modeA = require("./fileA");
modeA.run("xxx");

14.4.7. mysql

cnpm install mysql

```
var mysql = require('mysql');

var connection = mysql.createConnection({
host : 'localhost',
user : 'root',
password : 'root',
database : 'testdb'
});
connection.connect(function (error)
{
if(null != error) console.log(error);
});
connection.query('SELECT * FROM user', function (error, rows) {
if(error){
console.log('[SELECT ERROR] - ',error.message);
return;
}
```

```
console.log(rows);
for(i in rows)
{
console.log(rows[i].age + " " + rows[i].name);
}
});
```

14.4.8. 事件

```
var events = require('events');
```

```
var eventEmitter = new events.EventEmitter();

var connectHandlerA = function connected() {
console.log('事件 A');
}
var connectHandlerB = function connected() {
console.log('事件 B');
}

// 绑定 事件处理程序
eventEmitter.on('eventA', connectHandlerA);
eventEmitter.on('eventB', connectHandlerB);


setTimeout(function() {
//触发事件
eventEmitter.emit('eventA');
}, 1000);
setTimeout(function() {
eventEmitter.emit('eventB');
}, 2000);
```

监听一次消息：
once(event, listener)

14.5. jQuery

14.5.1. 基础语法

$(selector).action()

$("p").hide() - 隐藏所有段落

14.5.1.1. 元素选择器

$("p") 选取 <p> 元素。
$("p.intro") 选取所有 class="intro" 的 <p> 元素。
$("p#demo") 选取所有 id="demo" 的 <p> 元素。
$("p[xx='#']"") 选取所有 xx="#" 的 <p> 元素。

14.5.1.2. 属性选择器

$("[xx]") 选取所有带有 xx 属性的元素。
$("[xx='#']") 选取所有带有 xx 值等于 "#" 的元素。
$("[xx!='#']") 选取所有带有 xx 值不等于 "#" 的元素。
$("[xx$='.jpg']") 选取所有 xx 值以 ".jpg" 结尾的元素。

14.5.1.3. CSS 选择器

$("p").css("background-color","red");
所有 p 元素的背景颜色更改为红色

14.5.2. 名称冲突

var jq=jQuery.noConflict();
使用自己的名称（比如 jq）来代替 $ 符号。

14.5.3. 初始化调用

```
$(document).ready(
function(){...} 或  fun()
);
```

文档加载完成时进行调用,该函数可以写多个

14.5.4. 按钮绑定函数

$("#btn1").click(function(){alert("xx");}); -设置按钮响应函数
只能使用以下方法绑定：

```
$(document).ready(
    function(){
        $("button").click(function(){
             btnFun();
             return false;//拦截原有的操作
         });
    }
);
```

14.5.5. 效果

Callback 参数是函数执行成功后调用，执行失败不调用

14.5.5.1. 隐藏显示

$("p").hide();　　隐藏所有 p 标签
$("p").show();　　显示所有 p 标签

14.5.5.2. 淡入淡出

参数可以是"slow"、"fast" 或毫秒
$("p").fadeOut(1000);　　所有 p 标签 1s 内逐渐消失
$("p").fadeIn(1000);　　所有 p 标签 1s 内逐渐出现

$("p").toggle(1000);　　如果处于隐藏则淡入，如果显示状态则淡出
$("p").fadeTo(1000,0.5);　　变成 0.5 的透明度

14.5.5.3. 滑动

参数可以是"slow"、"fast" 或毫秒
$("p").slideUp();　　向上滑动（消失）
$("p").slideDown();　　向下滑动（出现）

$("p").slideToggle();　　自动根据状态切换

14.5.5.4. 动画

第一个参数是 css 效果，第二个参数是变化时间

```
var v=$("p");
v.animate({height:'30px',opacity:'0.4'},"slow");
```

停止动画效果
`$(selector).stop(stopAll,goToEnd);`
可选的 `stopAll` 参数规定是否应该清除动画队列。默认是 `false`，即仅停止活动的动画，允许任何排入队列的动画向后执行。
可选的 `goToEnd` 参数规定是否立即完成当前动画。默认是 `false`。
因此，默认地，`stop()` 会清除在被选元素上指定的当前动画。

14.5.6. 连续操作

```
$("p").css("color","red").slideUp(2000).slideDown(2000);
```

14.5.7. HTML 元素操作

14.5.7.1. 设置获取

text() - 设置或返回所选元素的文本内容
html() - 设置或返回所选元素的内容（包括 HTML 标记）
val() - 设置或返回表单字段的值

14.5.7.2. 添加

$("p").append(" XXX"); 结尾加入文字
$("p").prepend(" XXX"); 前方加入文字

after() 和 before() 会加入换行

```
var txt1="<p>Text.</p>";                    // 以 HTML 创建新元素
var txt2=$("<p></p>").text("Text.");    // 以 jQuery 创建新元素
var txt3=document.createElement("p");  // 以 DOM 创建新元素
txt3.innerHTML="Text.";
$("p").append(txt1,txt2,txt3);              // 追加新元素
```

14.5.7.3. 删除

$("div").remove(); 删除元素及子元素
$("p").remove(".aa"); 选择 p 的 class 为 aa 的元素进行删除

$("div").empty(); 删除子元素

14.5.7.4. radiobutton

14.5.7.4.1. 选择值

```
var val = $('#radioGroupDiv input[name="radioItemClass"]:checked').val();
```

选择 id 为 radioGroupDiv 范围内的 class 为 radioItemClass 的 value 属性

14.5.7.4.2. 设置值

$("input[name=name 属性对应的值]").val(["value 属性对应值"]);

14.5.7.5. CSS

```
<style>
.ss{color:blue;}
</style>
```

$("p").addClass("ss"); //添加样式

$("p").css("color","red");//设置颜色

removeClass() - 从被选元素删除一个或多个类
toggleClass() - 对被选元素进行添加/删除类的切换操作
css() - 设置或返回样式属性

14.5.7.6. 尺寸

width()	设置或返回元素的宽度（不包括内边距、边框或外边距）
height()	设置或返回元素的高度（不包括内边距、边框或外边距）
innerWidth()	返回元素的宽度（包括内边距）
innerHeight()	方法返回元素的高度（包括内边距）
outerWidth()	方法返回元素的宽度（包括内边距和边框）
outerHeight()	方法返回元素的高度（包括内边距和边框）

14.5.7.7. 遍历

parent()	返回被选元素的直接父元素	
parents()	返回被选元素的所有祖先元素，它一路向上直到文档的根元素 (<html>)	
parentsUntil()	返回介于两个给定元素之间的所有祖先元素	$("span").parentsUntil("div")，span 处于 div 子节点下，选择 span 和 div 之间的关联元素。
children()	返回被选元素的所有直接子元素。该方法只会向下一级对 DOM 树进行遍历。	
find()	返回被选元素的后代元素，一路向下直到最后一个后代。	$("div").find("p").fadeToggle(); 将 div 下所有 p 元素隐藏

siblings()	返回被选元素的所有同胞元素
next()	返回被选元素的下一个同胞元素 prev()　反向
nextAll()	返回被选元素的所有跟随的同胞元素 prevAll()
nextUntil()	返回介于两个给定参数之间的所有跟随的同胞元素

first()	选择第一个元素	$("p").first()　选择第一个 p 元素 $("div p").first()选择 div 内第一个 p
last()	选择最后一个元素	
eq()	返回被选元素中带有指定索引号的元素，索引从 0 开始	$("p").eq(1) 选择第二个 p 元素
filter()	只选择满足筛选条件的元素	$("p").filter(".a") 选择 class 为 a 的 p 元素
not()	只选择不满足筛选条件的元素	

14.5.8. AJAX

$.ajax({　key:val,key2:val2,...　});	$.ajax({ type: "POST", url: ".....", data:{valA: "val1",valB: "val2"}, dataType: "json", success: function(data){} error:function(errorData){} //返回数据不为 dataType: "json"时调用 error 方法 });
$.get(URL,callback);	$.get("a.jsp", function(data, status) { alert("Data: " + data + "\nStatus: " + status); });
$.post(URL,data,callback);	$.post("a.jsp", { valA: "val1", valB: "val2" }, function(data, status) { console.log("Data: " + data + "\nStatus: " + status);

	});

14.6. bootstrap

14.6.1. 引用

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="stylesheet" href="bootstrap.min.css">
<script src="jquery-3.3.1.min.js"></script>
<script src="popper.min.js"></script>
<script src="bootstrap.min.js"></script>
```

14.6.2. 容器

```
<div class="container">
```
所有元素放在该容器内生效

14.6.3. 灰色区域

```
<div class="jumbotron">
```

14.6.4. 行列布局

```
<div class="row">
    <div class="col-sm-4">
        <h3>hhhh</h3>
        <p>ppppp</p>
    </div>
    <div class="col-sm-4">
        <h3>hhhh</h3>
        <p>ppppp</p>
    </div>
</div>
```

尺寸包括：

.col-xs- .col-sm- .col-md- .col-lg-

14.6.5. 表格

```
<table class="table table-striped table-bordered">
    <thead>
        <tr class="table-active">
            <th>名称</th>
            <th>城市</th>
        </tr>
    </thead>
    <tbody>
        <tr   class="table-warning">
            <td>Tanmay</td>
            <td>Bangalore</td>
        </tr>
        <tr class="table-success">
            <td>Sachin</td>
            <td>Mumbai</td>
        </tr>
    </tbody>
</table>
```

颜色包含 table-active、table-danger 等

14.6.6. 表单

```
<form role="form">
    <div class="form-group">
        <input type="text" class="form-control" placeholder="文本输入">
    </div>
</form>
```

14.6.7. 按钮

```
<button type="button" class="btn btn-default">默认按钮</button>
<button type="button" class="btn btn-primary">原始按钮</button>
<button type="button" class="btn btn-success">成功按钮</button>
<button type="button" class="btn btn-info">信息按钮</button>
<button type="button" class="btn btn-warning">警告按钮</button>
<button type="button" class="btn btn-danger">危险按钮</button>
<button type="button" class="btn btn-link">链接按钮</button>
```

.btn-lg　比较大
.btn-sm　比较小
.btn-xs　特别小
.btn-block　　横跨父元素的全部宽度

14.6.8. 按钮组

```
<div class="btn-group btn-group-justified">
  <div class="btn-group">
    <button type="button" class="btn btn-primary">A</button>
  </div>
  <div class="btn-group">
    <button type="button" class="btn btn-primary">B</button>
  </div>
</div>
```

14.6.9. 下拉按钮

```
<div class="btn-group">
  <button type="button" class="btn btn-primary dropdown-toggle" data-toggle="dropdown">
  Sony <span class="caret"></span></button>
  <ul class="dropdown-menu" role="menu">
    <li><a href="#">Tablet</a></li>
    <li><a href="#">Smartphone</a></li>
  </ul>
</div>
```

14.6.10. 分隔按钮

```
<div class="btn-group">
  <button type="button" class="btn btn-primary">Sony</button>
  <button type="button" class="btn btn-primary dropdown-toggle" data-toggle="dropdown">
    <span class="caret"></span>
  </button>
  <ul class="dropdown-menu" role="menu">
    <li><a href="#">Tablet</a></li>
    <li><a href="#">Smartphone</a></li>
  </ul>
</div>
```

14.6.11. 辅助类

文本样式：
.text-muted
.text-primary
.text-success
.text-info
.text-warning
.text-danger

背景颜色：
.bg-primary
.bg-success
.bg-info
.bg-warning
.bg-danger

其他：
.pull-left 元素浮动到左边
.center-block 设置元素为 display:block 并居中显示
.clearfix 清除浮动
.show 强制元素显示
.hidden 强制元素隐藏
.sr-only 除了屏幕阅读器外，其他设备上隐藏元素
.sr-only-focusable 与 .sr-only 类结合使用，在元素获取焦点时显示(如：键盘操作的用户)

.text-hide将页面元素所包含的文本内容替换为背景图
.close 显示关闭按钮
.caret 显示下拉式功能

14.6.12. bootstrap-table

14.6.12.1.中文编码

bootstraptable 刷新函数参数包含中文时，接收需要解码

String name = new String(userName.getBytes("iso-8859-1"),"utf-8");

14.6.12.2.前台

```
 <link rel="stylesheet" type="text/css" href="bootstrap.min.css" />
 <link rel="stylesheet" type="text/css" href="bootstrap-table.min.css">
 <script src="jquery.min.js"></script>
 <script src="bootstrap.min.js"></script>
 <!-- 一个页面中 js 引用多次会出错 -->
 <script src="bootstrap-table.min.js"></script>
 <script src="bootstrap-table-zh-CN.min.js"></script>


<table id="myTable" class="table table-hover"></table>


function queryParamsFun(params)
{
        var id = $("#barrelDevidQuery").val();
        var temp = {
            limit: params.pageSize,
            page: params.pageNumber,
            id:id,
          };
        return temp;
}

            //bootstrap-table 要求服务器返回的 json 数据包含：total 和 rows
             $('#myTable).bootstrapTable({
                url: 'GetPageData',
                method: "POST",
                queryParamsType: "",
                queryParams: queryParamsFun,
                contentType: "application/x-www-form-urlencoded",
                dataType:'json',
                toolbar: "#toolbar",
                striped: true,                              // 是否显示行间隔色
                uniqueId: "id",                     //每一行的唯一标识，一般为主键列
                sortable: false,                            //是否启用排序
                sortOrder: "asc",                           //排序方式
                pagination: true,                           //是否显示分页
                sidePagination: "server",                 //分页方式：client 客户端分页，server
服务端分页（*）
                pageNumber:1,                               //初始化加载第一页，默认第一
页
```

```
                pageSize: 5,                            //每页的记录行数
                pageList: [5, 10, 25, 50, 100],         //可供选择的每页的行数
                strictSearch: true,
                showColumns: true,                      //是否显示选择列按钮
                showRefresh: true,                      //是否显示刷新按钮
                minimumCountColumns: 2,                 //最少允许的列数
                clickToSelect: true,                    //是否启用点击选中行
                height: 500,                            //行高，如果没有设置 height 属性，
表格自动根据记录条数觉得表格高度
                showToggle:true,                        //是否显示详细视图和列表视图
的切换按钮
                cardView: true,                         //是否显示详细视图
                detailView: false,                      //是否显示父子表
                columns:[
                    {
                        field: 'none',
                        title: '操作',
                        width: 120,
                        align: 'center',
                        valign: 'middle',
                        formatter: actionFormatter,
                    },
                    {title: 'IP',field: 'ip',visible:true},
                    {title: 'port',field: 'port'},
                    {title: 'ID',field: 'id'},
                    ]
            });


        $("#quaryBtn").click(function() {
            $("#myTable").bootstrapTable('refresh',queryParamsFun);
        });




            function actionFormatter(value, row, index) {
            //在数据刷新时调用，将数值固定写入到了 result
            //value:当前单元格的 field
            //row:当前行的数据
            //index:当前行索引
                var val = JSON.stringify(row).replace(/\"/g,"'");
                var result = "";
```

```
                result += "<a href='javascript:;' class='btn btn-xs blue' onclick=\"EditView(" + value + "," + val + ")\" title='编辑'><span class='glyphicon glyphicon-pencil'></span></a>";
                return result;
            }
            function EditView(value,row){}
```

14.6.12.3.后端

14.6.12.3.1. 实体类

```
public class Page implements Serializable {
    private static final long serialVersionUID = 1L;
    private Integer total; //bootstrap-table 要求服务器返回的 json 数据包含：total 和 rows
    private List rows = new ArrayList();
    public Integer getTotal() {return total;}
    public void setTotal(Integer total) {this.total = total;}
    public List getRows() {return rows;}
    public void setRows(List rows) {this.rows = rows;}
}
```

14.6.12.3.2. contoller

```
    @RequestMapping(value = "/GetPageData", method = RequestMethod.POST)
    public @ResponseBody Page getBarrelByPage(@RequestParam("page") Integer startPage, @RequestParam("limit") Integer limit, @RequestParam(id") String Id) {
        return service.getByPage(startPage, limit, Id);
    }
```

14.6.12.3.3. service

```
    public Page getByPage(Integer startPage, Integer limit,String ID)
    {
        int start = (startPage - 1) * limit;
        Integer total = mapper.getcount(ID);
        List<SaveData> rows = mapper.getllimit(start, limit, devID);
        Page page = new Page();
        page.setTotal(total);
```

```
        page.setRows(rows);
        return page;
    }
```

14.6.12.3.4. mapper

```
<sql id="limit_sql">
    <where>
        <if test="id != null and id    != ''">
            <![CDATA[
                TempTableName.id=#{id}
            ]]>
        </if>
    </where>
</sql>
<select id="getcount" parameterType="map" resultType="int">
    <![CDATA[
        select count(id) from TableName TempTableName
    ]]>
    <include refid="limit_sql" />
</select>
<select id="getllimit" parameterType="map" resultType="map">
    <![CDATA[
        select * from TableName TempTableName
    ]]>
    <include refid="limit_sql" />
    <![CDATA[
        ORDER BY id DESC LIMIT #{start}, #{limit}
    ]]>
</select>
```

14.6.13. 模态对话框

14.6.13.1.按钮触发

```
<button class="btn btn-primary btn-lg" data-toggle="modal" data-target="#myModal">打开对话框</button>
```

14.6.13.2.js 中触发

```
$('#myModal').modal({key: val})
$('#myModal').modal('toggle')  手动切换
$('#myModal').modal('show')  手动打开
$('#myModal').modal('hide')  手动隐藏
```

14.6.13.3.对话框

```
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myModalLabel" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-hidden="true">&times;</button>
                <h4 class="modal-title" id="myModalLabel">模态框（Modal）标题</h4>
            </div>
            <div class="modal-body">在这里添加一些文本</div>
            <div class="modal-footer">
                <button type="button" class="btn btn-default" data-dismiss="modal">关闭</button>
                <button type="button" class="btn btn-primary">提交更改</button>
            </div>
        </div>
    </div>
</div>
```

14.7. vue

14.7.1. 数据绑定

```
<script src="vue.min.js"></script>

<div id="vue_det">
    <h1>{{aaa}}</h1>
    <h1>{{bbb}}</h1>
    <h1>{{funA()}}</h1>
    <h1>{{funB()}}</h1>
```

```
</div>
<script type="text/javascript">
	var vm = new Vue({
		el: '#vue_det',
		data: {
			aaa: "xxxx",
			bbb: "yyyy",
		},
		methods: {
			funA: function() {
				return this.aaa + "111";
			},
			funB: function() {
				return this.bbb + "222";
			}
		}
	})
</script>
<button onclick="vm.aaa='mmmm';vm.bbb='nnn';">modiyA</button>
```

第十五章 WEB

15.1. 协议

15.1.1. request

请求行	请求方法 空格 URL 协议版本 CRLF
消息报头 headers【可选】	Key：val CRLF …keyval… Key：val CRLF
CRLF	
消息正文（可选，用于 POST)	

```
GET /abc?name=xxx HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0
```

15.1.2. response

状态行	协议版本 状态码 状态说明 CRLF
消息报头 headers【可选】	Key：val CRLF …keyval… Key：val CRLF
CRLF	
消息正文（可选)	

HTTP/1.1 200 OK
Content-Length: 51
Content-Type: text/plain
空行
数据

15.1.3. 响应状态码

1xx：指示信息--表示请求已接收，继续处理
2xx：成功--表示请求已被成功接收、理解、接受
3xx：重定向--要完成请求必须进行更进一步的操作
4xx：客户端错误--请求有语法错误或请求无法实现
5xx：服务器端错误--服务器未能实现合法的请求

15.1.4. 数据类型

```
"txt": "text/plain",
"html": "text/html",
"png": "image/png",
"jpeg": "image/jpeg",
"jpg": "image/jpeg",
"xml": "text/xml",
"css": "text/css",
"gif": "image/gif",
"ico": "image/x-icon",
"js": "text/javascript",
"json": "application/json",
"pdf": "application/pdf",
"svg": "image/svg+xml",
"swf": "application/x-shockwave-flash",
"tiff": "image/tiff",
```

```
  "wav": "audio/x-wav",
  "wma": "audio/x-ms-wma",
  "wmv": "video/x-ms-wmv",
```

application/json;charset=utf8　　请求数据类型中包含字符编码

15.2. HTML

```
<html>
    <head>
        <title>网页标题</title>
   </head>
    <body>
            <h1>一级标题</h1>
段落文字
    </body>
</html>
```

注释：　　<!--　　***　　-->

15.2.1.　文本

```
  <meta charset="utf-8">
```
中文支持

只保留空格，标签内文字写换行不起作用

<p> 段落标记　在前后都加入空行

<h1> 1 级标题　一直到<h6>

　换行

 使内容居中

<b>粗体

<i>斜体

<sup>上标

<sub>下标

<big>大号字体

<small>小号字体

加强语气

<del>删除线　　~~内容~~

<ins>下划线　　内容

<blockquote>　　大段文字　　会自动插入换行和外边距

<bdo　dir="rtl">hello</bdo>　倒序输出：olleh

预格式文本

保留标签内文字格式

<pre>

```
for(int i=0;i < 10;i++)
{
    cout <<   i   ;
```

```
        }
</pre>
```

显示 xml 内容：

```
<xmp>
<!--这里放 HTML 代码-->
</xmp>
```

15.2.2. 实体

保留字符，如< 使用<

显示结果	描述	实体名称	实体编号
	空格		
<	小于号	<	<
>	大于号	>	>
&	和号	&	&
″	引号	"	"
’	撇号	' (IE 不支持)	'
¢	分（cent）	¢	¢
£	镑（pound）	£	£
¥	元（yen）	¥	¥
€	欧元（euro）	€	€
§	小节	§	§
©	版权（copyright）	©	©
®	注册商标	®	®
™	商标	™	™
×	乘号	×	×
÷	除号	÷	÷

15.2.3. 区域

定义一个区域

定义文本区域

```
<p>abcd<span>XX</span>efghi</p>
```

15.2.4. 列表

为无序列表　　为有序列表

```
<ul>
    <li type="disc">aaa</li>
    <li type="square">bb</li>
    <li type="circle">cc</li>
</ul>
```

Type 默认为 disc（实体圆）

自定义列表

```
<dl>
<dt>aaa</dt>
  <dd>scriptAA</dd>
<dt>bbb</dt>
  <dd>scriptBB</dd>
</dl>
```

15.2.5. 表格

```
<table border="1" width = 200 height = 100>
    <caption>表格标题</caption>
    <tr><th>1 号 </th><th>2 号 </th></tr>
    <tr><td>1,1 </td><td>1,2 </td></tr>
    <tr><td>2,1 </td><td>2,2</td></tr>
</table>
```

tr 创建一行，th 创建标题行内容，td 创建单元格内容

<td> </td>　空单元格

colspan="4"　合并 4 列

可以在界面设置一个无边框的表格，在表格中放置内容进行排版

js 中使用　**.deleteRow(x); 删除第 x 行

15.2.6. 属性

属性内容用　属性=”..”

如果属性值本身含有双引号，那么必须使用单引号。如：

name=’aaa”bbb”ccc’

15.2.6.1. 通用属性

id	唯一标识一个标记对，表示一个数据，对应 css 中 #*
name	可以重复 ， 可以表示一组数据
class	对应 css 中 .*
width height	元素宽高 "100%"表示占所在空间的比例(最外层元素则为浏览器宽度)，位置不够时会自动伸缩
title	当鼠标移动到标签时提示 title 内容
align	"center" img 居中需要包含在 div 中 <img >

15.2.6.2. 标签属性

bgcolor	用于 body <body bgcolor="red"></body> bgcolor=”#FFFFFF”bgcolor="red" 或者 rgb(255,255,255)或者 white 也可以用图片地址
background	设置背景图像，图像小的时候则重复铺入 <body background ="1.png" >
border	用于 table Border 属性为数字，0 表示无边框

	<table　border=”1”>
Bool 属性	checked="checked" 或 checked 都代表勾选，不写代表默认不勾选

15.2.7. 链接

address　标记中放置的文本显示为链接样式文字

<address>

网站名

</address>

15.2.7.1. 跳转属性

_blank 表示新页面打开

_self 当前页面打开

链接名

15.2.7.2. 文本连接

本网站的一个页面

15.2.7.3. 图像链接

15.2.7.4. 页面内连接

开头

跳转到开头

#代表当前页面

15.2.7.5. 页面嵌套

<embed src="#" width="600"></embed>

Src 使用#代表当前页面，url 可以显示其他网页

15.2.8. 表单

15.2.8.1. 定义

```
<form      id="f1">
        ======表单元素=====
        <input type="text" id="inputContent" />
</form>
```

f1.** 可以通过元素 id 或者 name 访问到元素

name 或 id 可以标识表单，通过 f1.inputContent.value 可以获取表单中输入数据，f1 调用方式对于 p 这类标签不起作用

也可以使用 In= document.getElementById("inputContent");

15.2.8.2. 表单的提交

```
<form name="f1" action="b.html" method="get"      target = "_blank">
        用户名:<input type="text" name="user"/>
      <input type="submit" value="提交">
</form>
```

submit 是提交按钮（特殊按钮），会调用 form 中的 action

数据会传送到 b.html 页面中

target 为打开新页面方式，默认为当前页面打开(_self)，包括_blank ，_parent,_top

method 为 post 代表把输入数据按照 http 协议中的 post 方式传送到服务器，get 将输入的数据追加到 action 指定的地址后面（在地址栏可以看到），并传送到服务器。

表单提交时会将 form 中所有表单元素内容进行提交（非表单元素或没有 name 属性不提交）

15.2.8.3. 按钮

<input type="button" onclick = "alert('xx');" value="按钮名"/>

<button onclick = "alert('xx');">btn</button>

15.2.8.4. 输入框

用户名:<input type="text" name="name"/>

密码：<input type="password" name="code"/>

15.2.8.5. 文件选择

<input type="file"/>

15.2.8.6. 单选按钮

<input type="radio" name="sex" value="male"/>Male

<input type="radio" name="sex" value="female"/>Female

获取选择内容

```
var sexval = document.getElementsByName("sex");
 for(var i = 0; i < sexval.length;++i )
 {
            if(sexval[i].checked)
                    alert("check   "+ sexval[i].value);
```

}

15.2.8.7. 组选按钮

<input type="checkbox" name="x"/>A

<input type="checkbox" name="x"/>B

15.2.8.8. 下拉列表

<select name="number" id=”nu”>

<option value="1">1</option>

<option value="2">2</option>

<option value="3" selected="selected">3</option>

</select>

获取数据：

var val = document.getElementById('nu').value;

或者

var val = document.getElementsByName(number)[0].value;

15.2.8.9. 标题框

<filedset>

<legend>信息</legend>

<form>

<lable>身高：<input type="text"/></lable>

</form>

</filedset>

Lable 可以确保其中的内容为一行

15.2.8.10.文本框

<textarea>多行文本，支持回车符　　cols 和 rows 设置区域大小

15.2.8.11.重置输入内容

<input type="reset" value="重置"/>

重置 form 中的输入内容

15.2.9.　图像

15.2.9.1.　图像显示

位置是相对位置，可以是当前目录相连的的目录，也可以是其它网页目录中的内容。

align=”bottom” 默认值

也可以是 middle,top,left,right

<img src=”***”　alt=”***”>

如果图像无法显示，则显示 alt 内的内容

15.2.9.2. 图像映射

为图像中的某些区域创建映射，使其可以点击跳转

```
<img src="x.jpg" usemap="#map1"/>
<map name="map1">
<area shap="rect"    coords="x1,y1,x2,y2"    href="b.html"/>
</map>
```

15.2.10. base64

```
<img src='data:image;base64,***base64string***'>
```

15.2.11. mate

刷新，content 中包含刷新等待时间（秒），分号后写跳转到的页面地址

```
<meta http-equiv="refresh" content="0;2.jsp"/>
```

15.2.12. 框架

可以在页面中嵌套其他网页

```
<iframe src="https://****"></iframe>
```

a 标签中的 target 属性的_parent 和_top 在框架中就可以体现出来。

15.2.13. 功能

15.2.13.1.刷新页面

1 秒刷新一次

```
<meta http-equiv="Refresh" content="1">
```

15.2.14. html5 标记

15.2.14.1.标签

区域（类似 div）	使用
header	一个内容区域或整个页面的标题
article	强调独立区域
footer	区域块的底部
section	强调同级别分类
nav	包含导航翻页等操纵链接
aside	用来表示当前页面或文章的附属信息
hgroup	对 h1-h6 的标签进行分组(将相关的放在一个 hgroup 下)
figure	

input 类型	使用
email	
url	
number	
range	

date	

标签	使用
canvas	
video	
audio	
embed	
mark	
progress	
meter	
time	<time datetime="2017-01-01T20:00+01:00">2017-01-01</time> datetime 属性可以确保显示正确，+01：00 表示时差多一个小时
ruby	
rt	
rp	
wbr	
command	
details	
datagrid	
keygen	
output	
source	
menu	

15.2.14.2.属性

属性	使用
contenteditable	xxx 标记后元素内容在页面可编辑
designMode	设置可编辑状态，只能在 js 中用 document.designMode="on"
hidden	bool 属性（不需要写内容），让元素隐藏
spellcheck	输入内容拼写检查

表单属性	使用

formaction	<input type="submit" value="b1" formaction="f2.html"/> 可以指定 submit 按钮提交的 action，而不用固定走外部 form 的 action
formmethod	指定 form 中提交按钮提交的方式(get,post)
formtarget	指定 form 中提交按钮提交的打开新页面方式(_blank 等)
autofocus	鼠标焦点设置，bool 属性(不需要写内容)
required	如果定义该属性的元素没有输入内容，提交时会提示，不允许提交

15.3. CSS

15.3.1. 行内样式

直接在标记对加属性

红色宋体 30px 居中显示

15.3.2. 内嵌式

style 标记对中定义

定义 p 标记对内容颜色为红色

```
<style>
        p{color:Red;}
</style>
```

15.3.3. 链接式

一般写在 head 中

```
<link rel="stylesheet" type="text/css" href="a.css" />
```

css 文件

```
 h1,h2
{
    color:Blue;
    font-size:small;
}
p{
    color:Red;
}
```

15.3.4. 选择器

15.3.4.1. class 选择器

class 在同一页面可以重复使用

```
.a{color:Red;}
.b{color:Blue;}
```

两个 p 使用不同样式

```
<p class="a">xxxx</p>
<p class="b">yyyy</p>
```

15.3.4.2. id 选择器

id 在同一页面不能重复使用

```
#first{color:Red;}
#second{color:Blue}
```

i d 不能使用数字

```
<p id="first">xxxx</p>
<p id="second">yyyy</p>
```

15.3.4.3. 多级选择

对于 id 为 div1 的区域中，选择 p 标签进行设置

```
#div1 p{
color: aqua;
}
```

对于 div 标签，选择 id 为 content_1 进行设置

```
div#content_1{
color: aqua;
}
```

15.3.5. 布局

15.3.5.1. Div 布局

```
    <body>
  <div id = "content">
      <div id = "content_head">标题</div>
      <div id = "content_1">板块 1</div>
      <div id = "content_2">板块 2</div>
      <div id = "content_bottom">底部</div>
  </div>
</body>
```

Float 让 content_1 和 content_2 从左开开始效果，最后 clear 属性取消浮动

Css 中可以不声明 div 标签

```
    body{
    margin: 0px;
}
div#content{
    width: 100%;
    height: 800px;
    background-color: grey;
}
div#content_head
{
    width: 100%;
    height: 10%;
    background-color: antiquewhite;
}
div#content_1
{
    width: 30%;
    height: 80%;
    background-color: blueviolet;
    float: left;
}
div#content_2
{
    width: 70%;
    height: 80%;
    background-color: aqua;
    float: left;
```

```
}
div#content_bottom {
    width: 100%;
    height: 10%;
    background-color: red;
    clear: both;
}
```

15.3.5.2. Table 布局

```
    <body marginheight="0px" marginwidth="0px">
<table width="100%" height="900px" style="background-color: gray">
    <tr>
        <td colspan="2" width="100%" height="10%" bgcolor="antiquewhite">标题</td>
    </tr>
    <tr>
        <td width="30%" height="80%" bgcolor= "blueviolet">板块 1</td>
        <td width="70%" height="80%" bgcolor="aqua">板块 2</td>
    </tr>
    <tr>
        <td colspan="2" width="100%" height="10%" bgcolor="red">底部</td>
    </tr>
</table>
</body>
```

15.4. xpath

方法	说明
nodename	选取此节点的所有子节点
/	从根节点选取 从匹配选择的当前节点选择文档中的节点，只选择当前节点的子节点
//	从匹配选择的当前节点选择文档中的节点，不考虑它们的位置，选择子节点及子节点的任意深度节点
.	选取当前节点。
..	选取当前节点的父节点。
@	选取属性
*	选择所有内容

@*	选择所有属性
//title/text()	获取 title 标签包围的文字
//title[@lang]	选取所有拥有名为 lang 的属性的 title 元素。
//title[@lang='eng']	选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。
//bookstore/book[price>35.00]	选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。

第十六章 ANDROID

16.1. Android Studio

16.1.1. 发布

buid -》generate sigend APK
签名类型选择 V1

16.1.2. 快捷键

Alt + Enter　　解析未导入包的类或未处理异常的代码段，将代码自动加入

Ctrl + Shift + 空格　　自动提示生成代码

Ctrl+B　跳转到函数定义（ctrl+鼠标左键）

F8　下一步
shift+F8　下一个断点

右键菜单“Column Selection Mode”就可以开启竖选模式

16.1.3. 错误处理

========解决 adb not responding if youd like to retry...错误

netstat -ano | findstr "5037"
通过线程 id，在任务管理器关闭相应的程序

========编译联网问题
在环境变量中增加_JAVA_OPTIONS 然后它的值为-Djava.net.preferIPv4Stack=true,之后打开 AS 就会更新 gradle

========More than one file was found with OS independent....
build.gradle 中：

```
android {
.....
    packagingOptions {
        exclude 'META-INF/DEPENDENCIES'
        exclude 'META-INF/NOTICE'
        exclude 'META-INF/LICENSE'
        exclude 'META-INF/LICENSE.txt'
        exclude 'META-INF/NOTICE.txt'
    }
}
```

此时编译报错无法删除某个文件，则手动删除

========控件不显示
修改 styles.xml
将 Theme 改成 Base.Theme

设计界面点击魔法棒

16.1.4. 文件分类

Android 目录下 app

manifests	工程的配置，包含所有 activity 的定义	
java	存放代码文件，通过右键创建 activity 可以在这里创建出类（同时会在 manifests 中添加定义，layout 中加入 xml）	
Res	drawable	存放图片，可以使用复制，选中文件夹进行粘贴
	Layout	Activity 文件，每个 activity 对应一个 content_** 。按钮 design 和 Text 可以切换成 xml 和可视化显示
	Values	其中定义的值，可以通过 name 访问。

		如"@string/ttt" 可以访问 string 标签中的 ttt 字符串

Java 文件夹下对应 activity 的 java 代码，res/layout 下 xml 对应界面（xml 可以直接编辑，也可以使用可视化）

16.2. controls

16.2.1. 布局

16.2.1.1. ConstraintLayout

android.support.constraint.ConstraintLayout

可以在编辑界面任意拖拽控件

控件四个边上的圆点可以拖拽到其他位置生成约束

拖拽控件中间可以移动位置

拖拽控件四个边可以改变控件大小

选中控件后，点击约束点可删除该条约束

通过编辑界面的按钮（Infer Constraint）可以自动设置控件在界面的相对位置

16.2.1.2. LinearLayout

不能在编辑界面任意拖拽控件

自动线性水平排列:

```
<LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
    android:orientation="horizontal">
```

16.2.1.3. FrameLayout

不能在编辑界面任意拖拽控件

子元素总是以屏幕的左上角层叠在一起

16.2.2. Fragment

不包含 view，没有 findViewById 函数

需要在 onCreateView 中通过 view 调用 findViewById 函数

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                        Bundle savedInstanceState) {
        View vv =    inflater.inflate(R.layout.****, container, false);

}
```

16.2.3. 消息框

```
import android.widget.Toast;

String str="xx";
Toast.makeText(getApplicationContext(), str,Toast.LENGTH_SHORT).show();
```

16.2.4. 按钮

activity_***.xml 中

修改对应控件的 id：

```
android:id="@+id/btnStart"
```

修改按钮显示文字（引用字符串资源）：

```
android:text="@string/btn_start"
```

按钮响应 1：

```
android:onClick="ClickStart"
```

```
public void ClickStart(View view) {
    Toast.makeText(this,"****",Toast.LENGTH_LONG).show();
}
```

按钮响应 2：
传入 View.OnClickListener 接口对象

```
btn.setOnClickListener(new OnClickListener() {
          @Override
          public void onClick(View v) {
               System.out.println(".......");
          }
});
```

内部类的函数可以直接访问外部类的成员变量

16.2.5. 背景图片

16.2.5.1. xml 定义方式

xml 标签中
android:background="@drawable/a"

表示链接到 drawable 系列文件夹下的 a 图像（图像格式任意）

16.2.5.2. 代码设置方式

```
btn.setImageDrawable(getResources().getDrawable(R.drawable.a));
```

16.2.6. 去掉标题

androidManifest.xml 文件，Application 页面最下层 Application Nodes，选择好当前的类，进入 Theme，选择 system Resources"，选 Theme.NoTitleBar。

16.2.7. 透明度

**.getBackground().setAlpha(0);
0 代表全透明

16.2.8. 可见性

setVisibility():
View.VISIBLE:0 可见
View.INVISIBLE:4 不可见，但还占着原来的空间
View.GONE:8 不可见，不占用原来的布局空间

16.2.9. 编辑框

控件名：Plain Text

修改 id 为 edit

导入包
import android.widget.EditText;

变量
private EditText editText=null;

OnCreate 中
editText=(EditText)super.findViewById(R.id.edit);

使用时
String str=editText.getText().toString();

editText.setText("xx");//设置初始显示

16.2.10. 编辑框样式

新建样式文件 editsharp.xml

```
<?xml version="1.0" encoding="utf-8"?>
 <shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

     <padding
         android:bottom="7dp"
         android:left="7dp"
         android:right="7dp"
         android:top="7dp" />
     <!-- 设置圆角矩形 -->
     <corners android:radius="3dp" />
```

```
    <stroke
        android:width="5px"
        android:color="#1199EF" />

    <solid android:color="#CCCCCC" />

</shape>
```

控件中设置：

```
android:background="@drawable/editsharp"
```

16.2.11. ImageView

```
<ImageView
    android:background="@drawable/img_11"
    tools:srcCompat="@tools:sample/backgrounds/scenic" />
```

```
ImageView backImg  = (ImageView)super.findViewById(R.id.imgback);
backImg.setBackground(getResources().getDrawable(R.drawable.img_1));
```

16.3. frame

16.3.1. Application

16.3.1.1. Activity 包含

application 标签中，如果某个 Activity 包含：

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

则这个 Activity 类就是 app 启动时加载的

如果一个程序有多个 Activity，且每个都包含以上标签，则会创建多个启动图标，每个图标启动对应的 Activity。

```
<activity android:name=".aty2" android:label="aty2">
```

Label 可以设置启动图标的说明文字

16.3.1.2. App 类使用

通过 app 类共享数据

```
public class App extends Application {
    public String strDat;
}
```

manifests 中 application 标签增加属性 android:name=".App"

此时程序启动加载不使用默认的 Application 类， 使用 App 类。
可以通过((App)getApplicationContext()).strDat 在 Activity 之间共享数据

App 类中的 onCreate()函数在每次 Activity 启动时都会先进行调用

16.3.2. Activity

16.3.2.1. 创建

鼠标右键工程文件夹可以 new 一个 activity，自动生成对应的 java 文件和 xml

1.Res/layout 文件夹右键 new-》layout resource file 创建 myaty.xml 文件，对应页面

2.java 文件夹中创建 Aty 类 extends Activity， 重写 onCreate 函数，setContentView(R.layout.myaty);

3.Manifests 文件夹中 xml 进行注册：<activity android:name=".Aty"></activity>

其中.Aty 表示和 manifest 中 package 属性中文件夹位置拼凑出的 Aty 对应 java 文件的路径

setContentView 是每个 Activity 类构造函数中需要调用的，用来绑定界面 xml。传入的参数也可以是控件（控件也是 Activity）。

16.3.2.2. 启动

16.3.2.2.1. 类名启动

假设在 MainActivity 类中启动一个 aty

```
Intent in = new Intent(MainActivity.this,aty.class);
startActivity(in); //启动 Activity
```

16.3.2.2.2. 字符串启动

```
<activity android:name=".Myaty" android:exported="true">
    <intent-filter>
        <category android:name="android.intent.category.DEFAULT"></category>
        <action android:name=" AtyName "></action>
    </intent-filter>
```

可以通过字符串跨程序启动

android:exported="true"　默认为 ture，表示可以被其他程序启动

<action android:name="AtyName"></action>　AtyName 表示 Activity 标志字符串

==========启动

```
try {
    startActivity(new Intent("AtyName "));
}catch (Exception e){}
```

==========名称相同

如果两个 Activity 字符串名称相同，则打开时会进行选择询问

<action android:name="AtyName"></action>　后面加入：

<data android:scheme="sss"></data>

```
startActivity( new Intent("AtyName ",Uri.parse("sss://")));
```

启动时就会自动筛选　　//后面是参数

16.3.2.2.3. 网页启动 Activity

```
<activity android:name=".Myaty" android:label="aaa">
    <intent-filter>
        <category android:name="android.intent.category.BROWSABLE"/>
        <category android:name="android.intent.category.DEFAULT" />

        <action android:name= "android.intent.action.VIEW" />
        <data android:scheme="app"></data>
    </intent-filter>
</activity>
```

网页标签

```
<a href="app://">open</a>
```

类中可以获取网页传递的启动参数

```
Uri uri = getIntent().getData();
```

16.3.2.2.4. 打开网页

```
startActivity(new Intent(Intent.ACTION_VIEW,Uri.parse("https://www.baidu.com/")));
```

16.3.2.3. 传递参数

16.3.2.3.1. 传递单个参数

启动 activity 之前加入

```
in.putExtra("dat","hello");
```

activity 中 oncreate 中获取

```
Intent i = getIntent();
String  info = i.getStringExtra("dat");
```

16.3.2.3.2. 传递数据包

启动 activity 之前加入

```
in.putExtra("name","aa");
in.putExtra("id",1);
```

或者

```
Bundle b = new Bundle();
b.putString("name","aa");
b.putInt("id",1);

in.putExtras(b);
//可以是 in.putExtra("dat",b);  通过 Bundle  b =  i.getBundleExtra("dat");获取
```

activity 中 oncreate 中获取

```
Intent i = getIntent();
Bundle  b =  i.getExtras();

String name = b.getString("name");
int   id = b.getInt("id");
int   age = b.getInt("age",20);//默认值 20
```

16.3.2.3.3. 传递自定义数据 1

使用 java 的序列化，操作简单，速度慢

```
public class UserDat implements Serializable {
    String name = "aa";
    UserDat(String name)
    {
        this.name = name;
    }
}
```

加入数据

```
in.putExtra("user",new UserDat("xxx"));
```

获取数据

```
UserDat  dat =  (UserDat) i.getSerializableExtra("user");
```

16.3.2.3.4. 传递自定义数据 2

使用 andorid 接口，操作复杂，速度快

```
public class UserDat implements Parcelable {
    String name = "aa";
    UserDat(String name) {
        this.name = name;
    }


//可以通过 Alt + Enter 自动实现（直接选择类名，Add Parcelable implement 可以之直接全部自动实现）
    @Override
    public int describeContents() {
        return 0;
    }
    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(name);
    }




    protected UserDat(Parcel in) {
        name = in.readString();
    }
    public static final Creator<UserDat> CREATOR = new Creator<UserDat>() {
        @Override
        public UserDat createFromParcel(Parcel in) {
            return new UserDat(in);
        }

        @Override
        public UserDat[] newArray(int size) {
            return new UserDat[size];
        }
    };


}
```

传递数据

```
in.putExtra("user",new UserDat("aaa"));
```

接收数据

```
UserDat  dat =  (UserDat) i.getParcelableExtra("user");
```

16.3.2.4. 接收 activity 返回值

16.3.2.4.1. 启动 activity

```
Intent in = new Intent(MainActivity.this,aty.class);
startActivityForResult(in,0);//0 代表请求码 requestCode
```

实现虚函数用于接收返回值

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    String info = data.getStringExtra("dat");
}
```

16.3.2.4.2. Activity 中退出并设置返回值

```
Intent in = new Intent();
in.putExtra("dat","xxxxx");
setResult(1,in);//1 为返回码 resultCode
finish();//关闭当前 activity
```

16.3.2.5. 属性

获取当前 Activity 的 id 和名称

```
int taskId = getTaskId();
String curActivity = toString();
```

16.3.2.6. 启动方式

Manifest.xml 中可以配置 Activity 启动方式，默认是 standard
<activity android:name=".MainActivity" android:launchMode="standard">

16.3.2.6.1. Standard

堆栈方式，新建的 Activity 向堆栈中放入（创建新实例），当点击后退时，退出当前 Activity，显示之前的一个 Activity。
taskId 相同，不同堆栈位置不同。

16.3.2.6.2. singleTop

如果当前 Activity 在栈顶，如当前 Activity 启动自己，则不会创建新的实例

16.3.2.6.3. singleTask

一个任务堆栈只包含 Activity 一个实例，可以包含多个 Activity

16.3.2.6.4. singleInstance

一个任务堆栈只包含一个 Activity 一个实例

16.3.2.7. 运行流程

onCreate()方法是启动 Activity 地默认调用的方法

```
setContentView(R.layout.activity_main);
```
设置启动的 activity

运行流程包括的函数：
onCreate onStart onResume onPause onStop onDestroy onRestart
启动时：onCreate onStart onResume
点击最小化或进入后台时：onPause onStop
再次进入：onRestart onStart onResume
退出 activity：onPause onStop onDestroy

在一个 activity1 中启动另一个 activity2 时，先执行 activity1 的 onPause ，再执行 activity2 的 onCreate onStart onResume，当 activity2 完全呈现时，再执行 activity1 的 onStop activity2 退出时，先执行 onPause，当 activity1 执行完 onRestart onStart onResume 之后，activity2 再执行 onStop onDestroy

16.3.3. Servece

16.3.3.1. 特性

程序在退出 Activity 界面时在系统运行任务中可以看到，如果停止服务，则退出界面时系统运行任务中看不到。
工程文件夹右键 new -》Serve 自动创建 Service 类，以及在 manifests 中加入 service 标签。与 Activity 相同。
onCreate 和 onDestroy 会在 server 被创建和被停止时调用

由 startService 启动则只能用 stopService 停止
由 bindService 启动则只能用 unbindService 停止
如果 Service 已经创建，则调用函数时只执行 onStartCommand（MyServer 中）或者 onServiceConnected（Activity 中）

16.3.3.2. 启动停止

启动：`startService(new Intent(MainActivity.this,MyService.class));`
停止：`stopService(new Intent(MainActivity.this,MyService.class));`

onStartCommand 函数在 startService 每次调用时执行

16.3.3.3. 绑定

```
bindService(new Intent(MainActivity.this, MyService.class), MainActivity.this, Context.BIND_AUTO_CREATE);
```

实现 ServiceConnection 接口，onServiceConnected 在第一次 bindService 时调用
`onServiceConnected(ComponentName name, IBinder service)`

MyService 类中：

```
public IBinder onBind(Intent intent) {
    return  new Binder();
}
```

解除绑定：unbindService(MainActivity.this);

如果 server 不存在或已经解绑，则 unbindService 会抛出异常

==========

可以通过 IBinder 接口实现外部直接操作 Myserver(在 Myserver 类中创建一个类 MyBinder,MyBinder 直接通过 Myserver.this 访问。外部通过将 IBinder 转换为 Myserver 进行访问)

=========MyService 类中

```
public IBinder onBind(Intent intent){ return new MyBind();}
public class MyBind extends  Binder
{
    public void Fun(){ MyService.this.Fun();}
}
void Fun(){System.out.println("fun");}
```

=========Activity 中

```
public void onServiceConnected(ComponentName name, IBinder service) {
    MyService.MyBind bind = (MyService.MyBind) service;
    bind.Fun();
}
```

16.3.3.4. 跨应用的 Servece

16.3.3.4.1. 启动参数

通过包名和类名地址，可以跨应用启动服务

com.***是包名（manifest 中 package 属性）

MyService 是类名

```
Intent intent = new Intent();
intent.setComponent(new ComponentName("com.*** ","com.***.MyService"));
```

16.3.3.4.2. 启动

startService(intent);

16.3.3.4.3. 绑定

生成或加入 AIDL 文件后先进行重新编译

New　AIDL 文件　自动生成 IMyAidlInterface 文件

MyServer 中

```
public IBinder onBind(Intent intent) {
    return new IMyAidlInterface.Stub() {
        @Override
        public void basicTypes(int anInt, long aLong, boolean aBoolean, float aFloat, double
aDouble, String aString) throws RemoteException {

        }
    };
}
```

绑定操作

```
bindService(intent,MainActivity.this, Context.BIND_AUTO_CREATE);
```

================ 通信

将其他程序的 AIDL 文件原样拷贝到当前工程

1. New-》Folder-》AIDL Folder
2. 为 AIDL Folder 中创建包，包名和其他程序的 AIDL 文件包名一样
3. 将 AIDL 文件拷贝

```
IMyAidlInterface bind = IMyAidlInterface.Stub.asInterface(iBinder);
```

16.4. function

16.4.1.　线程中操作 UI

```
Handler  handerGetDat =  new Handler() {
    @Override
    public void handleMessage(Message msg) {
        Bundle bd = msg.getData();
        String str = bd.getString("dat");

        tv.setText(str);
```

```
            super.handleMessage(msg);
        }
    };
```

线程中调用

```
Message msg = new Message();

String str = "abc" ;
Bundle bd = new Bundle();
bd.putString("dat",str);

msg.setData(bd);
handerGetDat.sendMessage(msg);
```

16.4.2. 网络

16.4.2.1. 权限配置

AndroidMenifest.xml 中，manifest 标签内添加权限。

```
<uses-permission android:name="android.permission.INTERNET" />
```

网络连接必须在线程中

16.4.2.2. http 支持

在 res 下新增一个目录(目录名可以叫 xml)，然后创建一个名为：network_security_config.xml

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config cleartextTrafficPermitted="true" />
</network-security-config>
```

在 APP 的 AndroidManifest.xml 文件下的 application 标签增加以下属性

```
<application
...
 android:networkSecurityConfig="@xml/network_security_config"
... />
```

16.4.2.3. http 请求

```
public static String sendHttpRequest(String address) {
    HttpURLConnection connection = null;
```

```
    try {
        URL url = new URL(address);
        connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("POST");//只能发出 post
        connection.setConnectTimeout(8000);
        connection.setReadTimeout(8000);
        connection.setDoInput(true);
        connection.setDoOutput(true);
        InputStream in = connection.getInputStream();
        BufferedReader reader = new BufferedReader(new InputStreamReader(in));
        StringBuilder builder = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            builder.append(line);// 一行行的读取内容并追加到 builder 中去
        }
        return builder.toString();
    } catch (Exception e) {
        e.printStackTrace();
        return e.getMessage();
    } finally {
        if (connection != null) {
            connection.disconnect();//连接不为空就关闭连接
        }
    }
}


new Thread(){
    @Override
    public void run() {
        sendHttpRequest("http://192.168.1.3:8080/");
    }
}.start();
```

16.4.3. 振动

```
<uses-permission android:name="android.permission.VIBRATE"/>

Vibrator vibrator = (Vibrator)this.getSystemService(this.VIBRATOR_SERVICE);
vibrator.vibrate(20);
```

16.4.4. 去除标题标题栏

styles.xml
修改：

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="windowNoTitle">true</item>
</style>
```

16.4.5. 地图

16.4.5.1. 调用

16.4.5.1.1. key 值获取

cmd 命令行中，cd .android

输入 keytool -list -v -keystore debug.keystore，

输入密钥库口令，通常是：android

然后就可以获得 SHA1 安全码:

12:3B:38:F4:AB:44:6D:C9:5C:8F:97:32:DA:F3:93:44:38:07:88:5C

工程中 build.gradle(Module:app)内的 applicationId

```
"com.example.jyl.myapplication"
```

得到高德 key：4bac2f198d6aa1c7d537c15bc698e48f

16.4.5.1.2. 加入开发包

将 jar 和 so 的文件夹全部放入 libs 目录

app/build.gradle 文件中：

```
android {
compileSdkVersion 28
*****。。。。。

    sourceSets{
```

```
        main{
            jniLibs.srcDirs = ['libs']
        }
    }
}
```

右键 jar，add as library

16.4.5.1.3. 配置 AndroidManifest.xml

```
    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.jyl.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <meta-data                                    android:name="com.amap.api.v2.apikey"
android:value="4bac2f198d6aa1c7d537c15bc698e48f">
            //value=开发者申请的 key
        </meta-data>
    </application>


    <!--允许程序打开网络套接字-->
    <uses-permission android:name="android.permission.INTERNET" />
    <!--允许程序设置内置 sd 卡的写权限-->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <!--允许程序获取网络状态-->
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <!--允许程序访问 WiFi 网络信息-->
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <!--允许程序读写手机状态和身份-->
```

```
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <!--允许程序访问 CellID 或 WiFi 热点来获取粗略的位置-->
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

</manifest>
```

16.4.5.1.4.配置界面容器

activity 对应的 xml 中直接加入

```
    <com.amap.api.maps.MapView
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

16.4.5.1.5.初始化代码

```
    import com.amap.api.maps.MapView;

public class MainActivity extends AppCompatActivity {

    MapView mMapView = null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //获取地图控件引用
        mMapView = (MapView) findViewById(R.id.map);
        //在 activity 执行 onCreate 时执行 mMapView.onCreate(savedInstanceState)，创建地图
        mMapView.onCreate(savedInstanceState);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        //在 activity 执行 onDestroy 时执行 mMapView.onDestroy()，销毁地图
        mMapView.onDestroy();
    }
    @Override
    protected void onResume() {
        super.onResume();
        //在 activity 执行 onResume 时执行 mMapView.onResume ()，重新绘制加载地图
        mMapView.onResume();
    }
```

```
    @Override
    protected void onPause() {
        super.onPause();
        //在 activity 执行 onPause 时执行 mMapView.onPause (), 暂停地图的绘制
        mMapView.onPause();
    }
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        //在 activity 执行 onSaveInstanceState 时执行 mMapView.onSaveInstanceState (outState),
保存地图当前的状态
        mMapView.onSaveInstanceState(outState);
    }
}
```

16.4.5.2. 操作

16.4.5.2.1. 获取控制器

在 mapView 初始化后可以获取控制器：

```
//初始化地图控制器对象
    AMap aMap = null; if (aMap == null) { aMap = mMapView.getMap(); }
```

16.4.5.2.2. 设置标记点

```
    LatLng latLng = new LatLng(39.906901,116.397972);
final Marker marker = aMap.addMarker(new MarkerOptions().position(latLng).title("北京
").snippet("DefaultMarker"));
```

16.4.5.2.3. 标记点窗口的显示和隐藏

```
======实现地图点击和标记点点击接口
    implements AMap.OnMapClickListener, AMap.OnMarkerClickListener

    private Marker curShowWindowMarker;
public boolean onMarkerClick(Marker marker) {
    curShowWindowMarker = marker;
    marker.showInfoWindow();
    return true;
}
```

```
public void onMapClick(LatLng latLng) {
    if(curShowWindowMarker!=null){
        curShowWindowMarker.hideInfoWindow();
    }
}
```

======设置监听

```
    aMap.setOnMapClickListener(this);
aMap.setOnMarkerClickListener(this);
```

16.4.5.2.4. 设置标记点图标

```
    aMap.addMarker(new
MarkerOptions().***.icon(BitmapDescriptorFactory.fromResource(R.drawable.xx)));
```

16.4.5.2.5. 设置地图缩放大小

```
aMap.moveCamera(CameraUpdateFactory.zoomTo(5));
```

第十七章 PYTHON

17.1. config

17.1.1. 安装

环境变量 path： python 所在路径 及 python 路径下\Scripts

文本文件 a.py
命令行 python a.py 执行

直接运行 phython 可以进入 phython 命令行，直接输入执行语句

Python 命令行参数

选项	描述
-i	python -i t.py 运行 t.py 文件后不立即退出 python shell

-h	python -h 显示帮助信息
-V	输出 Python 版本号

Lib\idlelib\idle.bat 可以启动 python 自带的 gui 编程环境

17.1.2. 编译

python -O -m py_compile a.py
编译成优化的字节码

17.1.3. 包管理

Scripts 文件夹下 pip.exe 可以进行包管理

可以将包安装到 \Lib\site-packages 下
pip install ** 安装***
pip list 列出已经安装的包

包文件夹中包含__init__.py，可以为空
该文件在 import 模块时执行

lib 下载地址:
https://www.lfd.uci.edu/~gohlke/pythonlibs/
（关键字 edu whl）
1.下载 whl 文件后缀改为 zip 解压后直接拿出即可
2.pip install ***.whl

17.1.4. 虚拟环境

在当前目录下生成 venv 文件夹，文件夹内包含虚拟环境，安装的模块都将放在该文件夹内
python -m venv venv

激活使用虚拟环境，此时命令提示符会显示(venv)，此时对 python 操作时会使用该虚拟环境下的安装模块（linux 使用. venv/bin/activate）
venv\Scripts\activate

使用虚拟环境中的 python.exe 运行程序，可以调用到虚拟环境下安装的模块
venv\Scripts\python a.py

17.2. grammar

17.2.1. 导入包

from abc import *
from aaa.def import * 从 aaa 包中导入其中 def.py 内的所有内容
import math
import math as mt 导入包并重新命名

17.2.2. 声明

#coding=utf-8 则支持中文
str.decode('utf-8').encode('gbk') //字符串输出时使用

#!/usr/bin/python
告诉操作系统执行这个脚本的时候，调用/usr/bin 下的 python 解释器
这种用法是为了防止操作系统用户没有将 python 装在默认的/usr/bin 路径里。当系统看到这一行的时候，首先会到 env 设置里查找 python 的安装路径，再调用对应路径下的解释器程序完成操作。
#!/usr/bin/env python 3.3

17.2.3. 注释

#注释

17.2.4. doc

```
def fun(x):
    """this is doc"""
    print(x)

print(fun.__doc__)
```

17.2.5. 语句格式

```
a = 1
b = 2.3
c = a+b
```

变量无类型
语句不需要分号结束
依靠排版区分语句
区分语句作用域用 Tab

两个语句在同一行使用分号隔开

行连接保留第二行空格

```
str = "a\
   def"
```

17.2.6. 数据类型

int,float,bool,complex

bool 对应 True 和 False

4+2j complex

type(*) 获取数据类型
type(*) == type(1) 可以判断*是否和 1 是相同的类型

17.2.7. 类型转换

int(x)	将 x 转换为一个整数
long(x)	将 x 转换为一个长整数
float(x)	将 x 转换到一个浮点数
complex(real [,imag])	创建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为表达式字符串
eval(str)	用来计算在字符串中的有效 Python 表达式,并返回一个对象
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
chr(x)	将一个整数转换为一个字符
unichr(x)	将一个整数转换为 Unicode 字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

数字转字符串
s = str(12)

字符串转数字【s.isdigit()】
f = float(s)
i = int(s)

字符转数字
ord(c)

数字转字符
chr(i)

17.2.8. 运算符

无自增运算

连续赋值
a,b = 3,3+1

a = 2**3 #2 的幂运算 2*2*2
a = 3//2.0 #取整运算
a = 3%2 #取余

逻辑运算符:
not/and/or(没有!,||,&&)
结果 True 和 False

17.2.9. 语句结构

17.2.9.1. 选择

```
# coding=utf-8

score = 75
if score>90:
    print("优秀")
elif score>60:
    print("合格")
else:
    print("不合格")
```

17.2.9.2. 循环

yyy 在循环结束后输出，range 最后一个参数可以定义 step

```
for i    in range(1,10):
        print("item{0}:{1}".format(i,i + 0.3))
print("yyy")
```

```
a = 0
while a < 10:
        print(a)
        a+=1
```

17.2.10. 函数

```
def fun(a,b=5):
    if a > b:
        return a
    else:
        return b

print(fun(2,4))
```

17.2.11. Class

_xxx　　保护成员
__xxx　　私有成员
__xxx__　　系统定义名字

dir(类名)　获取类中的属性和方法

17.2.11.1. 定义

```
class CC:
    val = 10
    def __init__(self,_name):
        self.name = _name

    def show(self):
        print(self.name)
```

```
        print(self.val)
```

使用：

```
c = CC("abc")
c.show()
```

类内函数第一个参数都是类的对象
类的成员变量可以通过 self.** 直接创造
__init__ 构造函数

17.2.11.2.继承

```
class DD(CC):
    def __init__(self,nn):
        CC.__init__(self,nn)
    def Sw(self):
        print(self._name)


d = DD("yy")
d.Sw()
```

17.2.12. 异常

17.2.12.1.异常处理

```
try:
    val = dic[7]
except (KeyError,TypeError) as erro:  #或者 except Exception as e:
    print(" : %s"%erro)
else:
    无任何异常时执行
```

17.2.12.2.抛出异常

```
raise Exception("抛出一个异常")
```

17.2.13. 字符串

17.2.13.1.编码转换

默认 utf-8
cstr=str.encode(“gbk”)

转成 byte 流
str.encode('utf-8')

byte 流转成字符串
by.decode(encoding = "utf-8")

17.2.13.2.引号使用

单引号中可以包含双引号，双引号中可以包含单引号
c = 'abc'
d = 'it is a "apple"'
e = "it's a dog"

三引号支持换行，保留编辑时的换行格式 （三个单引号或者三个双引号）
f = '''it
is
a
dog'''

17.2.13.3.转义

字符串前加 r，字符串中\n 之类的转义字符可以当作普通字符
路径可以使用
f = r"abc\ndef"

17.2.13.4.字符串重复

字符串重复 10 次
"abc\n"*10

17.2.13.5.截取字符串

```
c = "abcdefg"
f1 = c[1]          #取 1 位置，结果 b
f2 = c[1:]         #取 1 位置及之后所有，结果 bcdefg
f3 = c[:3]         #从 0 位置取到 3 位置结束，结果 abc
f4 = c[1:3]     #从 1 位置取到 3 位置，结果 bc
```

17.2.13.6.格式化字符串

```
s = "val:%d,%.2f,%s"% (12,1.523,"abc")

s = "val:{0},{1},{2}".format(12,1.5,"abc")
```

```
%5s     表示字符串至少 5 个字符，如果不够则前面空格补齐
%-5s    表示字符串至少 5 个字符，如果不够则后面空格补齐
```

17.2.13.7.拆分

```
str.split(",")    返回列表[]
```

17.2.13.8.判断是否是整数

```
str.isdigit()
```

17.2.13.9.查找

```
str.find("de",0,6)    后两个参数表示从查找范围，可以省略
str.index("de")    如果找不到会报错
返回查找到的下标，-1 表示没有找到
```

17.2.13.10. 连接字符串

```
str.join("def")
```

17.2.13.11. 替换

```
str = str.replace(".zip","")
```

17.2.14. yield

```
def fun(start,end):
    while start < end:
        yield start
        start = start + 1

for n in fun(0,10):
    print(n)
```

17.2.15. main

```
if __name__ == "__main__":
    print "xxx"
```

每个文件都由一个模块名__name__，如果是被 import__name__为文件名。如果是运行开始的文件则为__main__
使用 if 可以让内容只在该文件被启动运行时执行，而该模块被调用时不执行

17.2.16. pass

```
def fun():
    pass
    #to do
```

方法或类下不能为空，使用 pass 起占位作用

17.2.17. 装饰器

```
def TestFun(fun):
    fun(10,20)

@TestFun
def fun(a,b):
    print(a+b)
```

通过@引用一个调用当前函数的函数

17.2.18. 数据集

17.2.18.1.元祖

arry = (123,"abc",12.4)
arry = tuple((123,"abc",12.4))

arry[1]　　#元素“abc”

内容不能修改，只能读取

17.2.18.2.列表

arry = [123,"abc",12.4]
arry　= list([123,"abc",12.4])
Arry = list("abc")

str = arry[1]　　#访问元素“abc”

del li[1]　　#删除元素“abc”

arry[-1]　#访问倒数第一个元素 12.4

arry[1]=22 #修改元素

arr.append(3) #添加元素

l.sort() #排序

arry　　= arry * 4 #元素变四倍

123 in　arry　　#元素是否在集合中

遍历：

for obj in arry:

#生成顺序序列

arry = range(len(xNames))

17.2.18.3.集合

s1 = set("abccd")
s2 = {'b','c','x','c'}

#只保留序列中非重复内容,去重
构造函数只能加入序列

#添加元素
s.add(3)

#交集　　结果：b，c
x = s1 & s2

#并集　　合起来不重复的元素　　结果：a b c d x
y = s1 | s2

#差集　　s1 中去掉和 s2 的交集　　结果：a d
z = s1 - s2

17.2.18.4.字典

dic = {"str":1.2, 3.1:4}
dic =dict(　{"str":1.2, 3.1:4})

#修改元素，如果字典内有，则进行访问
dic['str'] += 5

#插入元素，如果字典内没有，则插入新值
dic[33] = 111
print(dic[33])

==========查找
```
if 5 in dic:
```

```
==========遍历
for key,value in dic.items():
    print key,value


for key in dic:
    print key,dic[key]


==========获取 set
s = set(dic.values());
s2 = set(dic.keys());


==========字符串格式化
dic = {"str1":1.2, "str2":"aaa"}
ss = "%(str1)f  %(str2)s"%dic
print(ss)
```

17.2.18.5.运算方法

元素个数　len(arr)

最小值　　min(arr)

最大值　　max(arr)

求和　　sum(arr)

17.2.19. lambda

结果要用*ret 进行取值

17.2.19.1.操作每个元素

为每个元素执行相同操作

```
li = [1,2,3,4,5,6]
ret = map(lambda x: x*100,li)
```

17.2.19.2.排序

```
class student:
    def __init__(self,age,name):
        self.age = age
        self.name = name
arry = (student(11,'a'),student(4,'b'),student(10,'x'))
```

对类中的 age 进行排序

```
ret = sorted(arry,key=lambda x:x.age)
for stu in ret:
    print(stu.age)
```

17.2.19.3.过滤数据

只保留满足条件（x%2==0）的数

```
li = [1,2,3,4,5,6]
ret = filter(lambda x: x%2==0,li)
```

```
arry = ('xxx','abcxxxdef','xyz','x.x','axxxxxxa')
```

完全匹配:

```
ret = filter((lambda x: re.match(r'xxx',x)),arry)
```

匹配包含字符串的内容:

```
ret = filter((lambda x: re.search(r'xxx',x)),arry)
```

只匹配 x.x

```
ret = filter((lambda x: re.search(r'x\.x',x)),arry)
```

17.2.19.4.列表解析

从列表中获取满足条件的元素

```
li = [1,2,3,4,5,6]
ret = [x for x in li if x%2==0 ]
```

17.2.20. Py 文件交互

在 b 文件中存在类 CC

```
class CC:
    def fun(self):
        print("cc")
```

其他文件使用时

====导入文件

```
import   b
c = b.CC()
c.fun()
```

====导入文件中的类

```
from b import CC
c = CC()
c.fun()
```

======函数调用

在 b 文件中只存在 fun 函数时

```
import b
b.fun()
```

======执行代码

如果 b 文件中有执行代码，被其他文件导入，因为导入在文件上面，所以 b 文件中的代码首先执行

17.2.21. 输入输出

```
a = input("输入 a:")
i = int(raw_input("input:"))

print(a)
print (a,b,c)
```

17.3. file

1 读写文件

17.3.1.1. 中文路径转换

path = "E:\\文件.txt"
uipath = unicode(path , "utf8")

17.3.1.2. 写文件

```
f = open("D:\\1.txt","w") #"a"追加，打开不清空
f.write("abcd\n")
f.close()
```

17.3.1.3. 读文件

```
for line in open("D:\\1.txt"):
    pos1,pos2 = line.split(",")
```

读取文本文件全部内容

```
f = open("D:\\1.txt")
dat = f.read()
```

17.3.1.4. 文件指针位置移动

f.seek(移动量,基准位置)　　f.seek(0,0)移动到开始位置

17.3.1.5. 读写 utf8

```
import codecs

ll = codecs.open(r"D:\decoderZip.pyw", 'r', 'utf-8').readlines()
of = codecs.open(r"D:\decoderZip1.pyw", 'w', encoding='utf8')
for line in ll:
```

```
    of.write(line)
of.close()
```

17.3.2. 遍历文件

os.sep windows 下是\\
os.path.join 连接字符串，文件夹路径和文件名

读取出的文件、路径名中包含中文时，字符串不能 print 但可以正常使用

17.3.2.1. 遍历文件夹

```
import os
dirPath= "d:" + os.sep

for name in os.listdir(dirPath):
    if os.path.isfile(os.path.join(dirPath,name)):
        print "file:" + name
    if os.path.isdir(os.path.join(dirPath,name)):
        print "dir:" + name
```

返回目录下所有 pdf 文件的全路径

```
ret = glob.glob("D:\\*.pdf")
```

17.3.2.2. 遍历所有文件

遍历 d:\\123 下所有 jpg 文件
walk 返回一个三元组：
dirpath 当前处理路径
dirnames 当前处理路径下包含的路径
filenames 当前处理路径下包含的所有文件名

```
#encoding=utf-8
import os
s = os.sep
root = "d:" + s + "123" + s
for dirpath, dirnames, filenames in os.walk(root):
    for fileName in filenames:#获取每一个文件名
        fileName_ext = os.path.splitext(fileName)#拆分文件名和类型
        if(fileName_ext[1] == ".jpg"):
```

print os.path.join(dirpath,fileName)#输出文件路径

17.3.3. 文件及目录操作

17.3.3.1. 目录拼接

```
ret = os.path.join("D:"+ os.path.sep + "abc","def")
```
得到 D:\abc\def

```
ret = os.path.split('D:\\abc.jpg')
```
得到('D:\\abc', 'def')

```
ret = os.path.splitext("D:\\abc.jpg")
```
得到('D:\\abc', '.jpg')

```
ret = os.path.abspath("..\\\\abc\\def")
```
将多余的分隔符去掉，将相对路径转换为绝对路径

17.3.3.2. 当前文件路径

```
curdir = os.path.abspath( os.curdir)
```
或者：`curdir = os.path.split(os.path.abspath(__file__))[0]`

17.3.3.3. 判断是否存在

判断目录或文件是否存在
os.path.exists(strPath)

17.3.3.4. 创建目录

makedirs 只能创建不存在的目录
```
if(not os.path.exists(strPath)):
    os.makedirs(strPath)
```

17.3.3.5. 删除目录

删除空目录：
```
os.rmdir(srcPath)
```

可以删除非空目录

```
shutil.rmtree(srcPath)
```

17.3.3.6. 移动/重命名

```
shutil.move(srcPath,dstPath)
```

17.3.3.7. 拷贝/重命名

```
shutil.copy(srcPath,dstPath)
```

17.3.3.8. 删除文件

```
os.remove(srcPath)
```

17.3.3.9. 目录/文件状态

```
os.stat(dir)
```

17.4. Thread

17.4.1. 注意问题

线程中不能修改全局变量

17.4.2. 导入模块

```
import threading
```

17.4.3. 开启线程

传入类的__thrRun 函数作为线程入口

```
_thread = threading.Thread(target=self.__thrRun)
_thread.setDaemon(True)
_thread.start()
```

setDaemon 默认为 False，如果使用默认值，则进程结束后，线程依然执行
如果设置为 True，则进程结束时线程停止

17.4.4. 线程等待

等待_thr1 线程执行完成，如果传递参数，则只等待传递的时间

```
self._thr1.join()
```

17.4.5. 暂停

```
import time

单位为 s
time.sleep(1)
```

17.4.6. 线程同步

```
self.lock = threading.Lock()

self.lock.acquire()
执行代码。。。
self.lock.release()
```

17.4.7. 结束线程

类中启动函数定义 startRun 状态为 True

```
def Start(self,begin,end):
     self.startRun = True
```

类中停止函数定义 startRun 状态为 False

```
def Stop(self):
    self.startRun = False
```

类中线程函数判断 startRun 进行停止

```
def  __thrRun(self):
        while (True):
            if self.startRun == False: return
```

```
....
if (..):
    self.startRun = False
    return
```

17.5. tkinter

```
#coding=utf-8
import tkinter as tk
import threading
import tkinter.messagebox as tkMB
import tkinter.filedialog as tkFD



class Application(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)

        self.pack()  #pack 布局
                     #可以将 pack 全部换成 grid  grid(row=1)   grid(row=0,column=1)
        self.create_widgets()
    def create_widgets(self):
        self.L = tk.Label(self,text="txt", fg="black", bg="white")
        self.L.pack()

        self.E = tk.Entry(self)
        self.E.pack()

        self.Etxt = tk.StringVar()
        self.Etxt.set("xxx")
        self.E["textvariable"] = self.Etxt



        self.BtnRun = tk.Button(self,width=15, height=5,text = "Run",command = self.__Run)
        self.BtnRun.pack(padx=20, side='left')
    def __Run(self):
        filePathName = tkFD.askopenfilename(filetypes=[('all files', '.*'), ('text files', '.txt')])
        self.L['text'] = filePathName

        _thread = threading.Thread(target=self.__thrRun)
        _thread.setDaemon(True)
        _thread.start()
    def __thrRun(self):
        ss = self.Etxt.get()
```

```
        tkMB.showinfo("提示", ss)
        self.BtnRun['state'] = tk.NORMAL

root = tk.Tk()
root.title('title')
root.geometry('400x200')
root.maxsize(400, 200)
root.minsize(400, 200)
app = Application(master=root)
app.mainloop()
```

17.5.1. 设置窗口大小和位置

root.title('窗口名')
root.geometry('600x400') *#设置了主窗口的初始大小 600x400*

root.maxsize(600, 400)
root.minsize(300, 240)

17.5.2. 设置窗口居中

```
def center_window(root, width, height):
    screenwidth = root.winfo_screenwidth()
    screenheight = root.winfo_screenheight()
    size = '%dx%d+%d+%d' % (width, height, (screenwidth - width)/2, (screenheight - height)/2)
    root.geometry(size)
```

center_window(root, 300, 240)

17.5.3. 控件

17.5.3.1. 按钮

属性名进行设置

```
tk.Button(text="运行", fg="red",command=self.fun,width = 20,height = 10)
```

17.5.3.2. 文本标签

```
self.l1 = tk.Label(text="txt", fg="black", bg="white")
```

```
self.l1["text"]="xxx"
```

17.5.3.3. 编辑框

```
self.editTxt = tk.Entry()          #创建文本控件
self.editTxt.pack()

self.contents = tk.StringVar()    #和 Entry 控件绑定的变量
self.contents.set("this is a variable")
self.editTxt["textvariable"] = self.contents

ss = self.contents.get()     #获取字符串

self.editTxt.bind('<Enter>',self.fun1)    #绑定 Enter 事件到函数 def fun1(self, event):
```

17.5.3.4. 进度条

```
scale=Scale(top,from_=10,to=40,orient=HORIZONTAL,command=fun)
scale.set(12)  #设置起始位置
scale.pack(fill=X,expand=1)


def fun(ev=None):
```

17.5.4. 控件可用

```
self.BtnRun['state'] = tk.DISABLED
self.BtnRun['state'] = tk.NORMAL
```

17.5.5. Ttk

```
from tkinter.ttk import *
```

可以取代（from Tkinter import *）tk 中的 button 等控件

```
self.l1 = Label(text="Test", style="BW.TLabel")
```

文本可以让底色透明

17.5.6. 弹出对话框

```
from tkinter import *

def dialog():
    win = Toplevel()
    Label(win,text="abc").pack()
    win.focus_set()
    win.grab_set()
    win.wait_window()

root = Tk()
Button(root,text="btn",command=dialog).pack()
root.mainloop()
```

17.5.7. 特殊对话框

Lib\tkinter\
filedialog.py
messagebox.py

17.5.7.1. 文件和文件夹

```
import tkinter.filedialog as tkFD
```

选择文件对话框

```
filePathName =tkFD.askopenfilename(filetypes=[("图像", ".jpg")])
[('all files', '.*'), ('text files', '.txt')]
```

文件夹选择对话框

```
dir = tkFD.askdirectory()
```

17.5.7.2. 消息框

```
import tkinter.messagebox as tkMB

tkMB.showinfo("提示","处理完成")
```

17.6. Lib

17.6.1. sys

17.6.1.1. 获取 python 环境路径

```
import sys
sys.path
```

17.6.1.2. 获取命令行参数

脚本名： sys.argv[0]
参数 1： sys.argv[1]
参数 2： sys.argv[2]

17.6.1.3. 获取已加载的模块名

```
sys.modules
```

可以通过该字典移除已经加载的模块

17.6.2. 获取环境变量

```
export CaffeHome=home/cpms/caffe

import os
env_dist = os.environ
print env_dist.get('CaffeHome')
```

17.6.3. 执行系统命令

仅仅在一个子终端运行系统命令，而不能获取命令执行后的返回信息
os.system("help")

17.6.4. string

遍历小写字母

```
for word in string.lowercase:
    print word
```

17.6.5. re

```
import re

pic_url = re.findall('img src="(.*?)"',html,re.S)
```

无 re.S 时，遇到字符串中的\n 则不往下一行匹配

match(pattern, string, flags = 0)	使用带有可选标记的正则表达式的模式来匹配字符串。如果匹配成功，返回匹配对象，否则返回 None
search(pattern, string ,flags = 0)	使用可选标记搜索字符串中第一次出现的正则表达式模式。如果匹配成功，则返回匹配对象，否则返回 None
findall(pattern, string[,flags])	查找字符串中所有(非重复)出现的正则表达式模式，并返回一个匹配列表
finditer(pattern, string[,flags])	与 findall()相同，但返回的是一个迭代器。对于每一次匹配，迭代器都能返回一个匹配对象
split(pattern, string, max = 0)	根据正则表达式的模式分隔符，split 函数将字符串分割为列表，返回匹配列表，分割最多操作 max 次

17.6.6. math

```
math.sqrt(9)
```

17.6.7. random

产生 1 到 10 之间的随机数

```
random.uniform(1,10)
```

17.6.8. time

暂停 1s

```
time.sleep(1)
```

获取当前时间

```
s = time.strftime('%Y-%m-%d %H:%M%S',time.localtime(time.time()))
```

17.6.9. datetime

```
start = datetime.datetime.now()
timeSpan = datetime.datetime.now() - start
```

17.6.10. pickle

```
=======序列化到内存
lsSave = pickle.dumps(obj)     #将对象序列化，序列化后的数据 lsSave 可以用来传递
l = pickle.loads(lsSave)          # 将对象恢复


=======序列化到文件
f1 = open('1.pk','wb')
pickle.dump(obj,f1)     #将对象放入文件
f1.close()


f2 = open('1.pk','rb')   # 将对象从文件读回
lf = pickle.load(f2)
f2.close()
```

17.6.11. dbm

key 和 val 必须是字符串

```
import dbm
db = dbm.open("D:\\people","c")
db["aa"] = "10"
db["bb"] = "20"
db.close()

C：打开 dbm 文件，如果不存在则创建
N：新建 dbm 文件，如果存在则覆盖
W:打开 dbm 文件，不存在则失败


删除 key-val
del db["aa"]

获取所有 key（返回 list）
li = db.keys()
```

17.6.12. sqlite

```
import sqlite3
conn = sqlite3.connect("D:\\abc.db")
cursor = conn.cursor()
cursor.execute("create table aa(id  int , name varchar(30),age int)")
cursor.execute("insert into aa values(1,'n1',11)")
cursor.execute("insert into aa values(2,'n1',11)")
conn.commit()
cursor.close()
conn.close()
```

获取数据：

```
cursor.execute("select * from aa")
for row in cursor.fetchall():
    print(row)
```

17.6.13. socket

17.6.13.1.udp

17.6.13.1.1. 服务器

```
from socket import *
HOST = '192.168.1.3'
PORT = 9999

s = socket(AF_INET, SOCK_DGRAM)
s.bind((HOST, PORT))
while True:
    data, address = s.recvfrom(1024)
    print(data, address)
    s.sendto('this is the UDP server',address)
s.close()
```

17.6.13.1.2. 客户端

```
from socket import *
HOST='192.168.1.3'
```

```
PORT=9999

s = socket(AF_INET,SOCK_DGRAM)
s.connect((HOST,PORT))
while True:
    message = raw_input('send message:>>')
    s.sendall(message)
    data = s.recv(1024)
s.close()
```

17.6.13.2.tcp

17.6.13.2.1. 服务器

```
from socket import *
address='127.0.0.1'        #127.0.0.1 是监听本机  0.0.0.0 是监听整个网络
port=12345

buffsize=1024
s = socket(AF_INET, SOCK_STREAM)
s.bind((address,port))
s.listen(1)        #最大连接数 1
while True:
    clientsock,clientaddress=s.accept()
    print('connect from:',clientaddress)
    while True:
        recvdata=clientsock.recv(buffsize).decode('utf-8')
        if    not recvdata:
            break
        senddata=recvdata+'from sever'
        clientsock.send(senddata.encode())
    clientsock.close()
s.close()
```

17.6.13.2.2. 客户端

```
from socket import *
address='127.0.0.1'
port=12345

buffsize=1024
s=socket(AF_INET, SOCK_STREAM)
```

```
s.connect((address,port))
while True:
    senddata=”xxxx”
    s.send(senddata.encode())

      recvdata=s.recv(buffsize).decode('utf-8')
      print(recvdata)
s.close()
```

17.6.14. 读写 ini

```
import configparser
cfg = configparser.ConfigParser()
cfg.read("F:\\cfg.ini")
val = cfg.get("section","key")

cfg.add_section("section")
cfg.set("section","a","b")
cfg.write(open("F:\\cfg.ini","w"))
```

17.6.15. 测试

```
class TesC(unittest.TestCase):
    def setUp(self):
        print('start')
    def tearDown(self):
        print('end')
    def runTest(self):
        self.failIf(1+1 != 2,'erro')
        self.failUnless(1+1 == 2,'erro')
        self.failUnlessEqual(1+1,2,'erro')


suite = unittest.TestSuite()
suite.addTest(TesC())
runner = unittest.TextTestRunner()
runner.run(suite)
```

17.6.16. Zipfile

```
import  zipfile
```

17.6.16.1.压缩

```
f = zipfile.ZipFile('E:/test.zip','w',zipfile.ZIP_DEFLATED)
f.write('D:/1.jpg')
f.write('D:/2.jpg')
f.close()
```

17.6.16.2.解压

```
f = zipfile.ZipFile('E:/test.zip')
f.extractall()        不传递参数则解压所有文件到当前目录
f.close()
```

第一个参数为解压后路径，路径必须存在
第二个参数为需要解压的文件名列表（zip.namelist()可以获取）
第三个参数为密码，函数必须传入 ascii 字符串

```
zip.extractall(r'D:\1',None,'123'.encode('ascii'))
```

17.6.17. smtp

```
import smtplib
from email.mime.text import MIMEText
from email.utils import formataddr

my_sender = '******@qq.com'  # 发件人邮箱账号
my_pass = '...'
my_user = '****@qq.com'  # 收件人邮箱账号
def mail():
    try:
        msg = MIMEText('填写邮件内容', 'plain', 'utf-8')
        msg['From'] = formataddr(["FromRunoob", my_sender])  # 括号里的对应发件人邮箱昵称、发件人邮箱账号
        msg['To'] = formataddr(["FK", my_user])  # 括号里的对应收件人邮箱昵称、收件人邮箱账号
        msg['Subject'] = "发送邮件测试"  # 邮件的主题，也可以说是标题

        server = smtplib.SMTP_SSL("smtp.qq.com", 465)  # 发件人邮箱中的 SMTP 服务器，端口是 25
        server.login(my_sender, my_pass)  # 括号中对应的是发件人邮箱账号、邮箱密码
        server.sendmail(my_sender, [my_user, ], msg.as_string())  # 括号中对应的是发件人邮箱账号、收件人邮箱账号、发送邮件
        server.quit()  # 关闭连接
    except Exception as e:  # 如果 try 中的语句没有执行，则会执行下面的 ret=False
        print(e)
```

```
            return False
        return True
```

17.6.18. tool

http 服务器:
python -m http.server 80
默认端口 8080

17.6.19. http

```
from urllib.parse import urlsplit, parse_qs
from http.server import HTTPServer, BaseHTTPRequestHandler


class ProHeader(BaseHTTPRequestHandler):
    def do_GET(self):
        url = urlsplit(self.path)
        qs = parse_qs(url.query)

        rsp = "get"
        rsp = rsp.encode("gb2312")
        self.send_response(200)
        self.send_header("Content-type", "text/html; charset=gb2312")
        self.send_header("Content-Length", str(len(rsp)))
        self.end_headers()
        self.wfile.write(rsp)

    def do_POST(self):
        datas = self.rfile.read(int(self.headers['content-length']))



if __name__ == '__main__':
    httpd = HTTPServer(('', 1234), ProHeader)
    httpd.serve_forever()
```

17.7. c++Mix

17.7.1. C 调用 python

文件名一定不能是 test，abc.....系统模块有。

不存在#include <inttypes.h>
屏蔽该行，包含#include <stdint.h>

没有 python36_d.lib
pyconfig.h 中：
pragma comment(lib,"python36_d.lib")
改为 python36.lib

17.7.1.1. Py 文件

文件名 MyTestMul

```
def add(i,j):
    return i+j
```

17.7.1.2. c 调用

```
Py_SetPythonHome(L"C:\\py363");
Py_Initialize();
if ( !Py_IsInitialized() )
{
    cout<<"Py_Initialize 错误"<<endl;
    return;
}


PyObject* model = PyImport_ImportModule("MyTestMul");
if (NULL == model)
{
    cout<<"PyImport_ImportModule 错误"<<endl;
    return;
}
```

```
PyObject* pfun = PyObject_GetAttrString(model,"add");

PyObject* pParm = PyTuple_New(2);
PyTuple_SetItem(pParm, 0, Py_BuildValue("i",3));
PyTuple_SetItem(pParm, 1, Py_BuildValue("i",4));

PyObject* pRetVal = PyEval_CallObject(pfun, pParm);
int retVal = -1;
int ret = PyArg_Parse(pRetVal, "i", &retVal);
if (0 == ret)
{
	cout<<"PyArg_Parse 错误"<<endl;
	return;
}

//Py_DECREF(pfun);
//Py_DECREF(pParm);
//Py_DECREF(pRetVal);

Py_Finalize();
```

=====构造参数

```
PyObject *param = Py_BuildValue("(i,i,i)", 123, 456, 789);//  构造元祖数据

PyObject * pList = PyList_New(0);
for (int i = 0; i < 3; i++)   PyList_Append(pList, Py_BuildValue("i", i));
```

字符	含义
s 或者 z	(string) [char *] 将 C 字符串转换成 Python 对象，如果 C 字符串为空，返回 NONE。
s#或者 z#	(string) [char *, int] :将 C 字符串和它的长度转换成 Python 对象，如果 C 字符串为空指针，长度忽略，返回 NONE。
z	(string or None) [char *] :作用同 s
i 或 b、l（long）	(integer) [int] :将一个 C 类型的 int 转换成 Python int 对象。
c	(string of length 1) [char] ：将 C 类型的 char 转换成长度为 1 的 Python 字符串对象。
d 或 f	(float) [double] :将 C 类型的 double 转换成 python 中的浮点型对象。
O&	(object) [converter, anything] ：将任何数据类型通过转换函数转换成 Python 对象，这些数据作为转换函数的参数被调用并且返回一个新的 Python 对象，如果发生错误返回 NULL。
(items)	(tuple) [matching-items] ：将一系列的 C 值转换成 Python 元组。

[items]	(list) [matching-items] ：将一系列的 C 值转换成 Python 列表。
{items}	(dictionary) [matching-items] ：将一系类的 C 值转换成 Python 的字典，每一对连续的 C 值将转换成一个键值对。

17.7.2. python 使用 dll

字符串传递需要转换格式
cstr=str.encode(“gbk”)

cdecl 调用方式：

ctypes.cdll.LoadLibrary

Stdcall 调用方式：

ctypes.windll.LoadLibrary

17.7.2.1. 函数调用

```
extern "C" __declspec(dllexport) char* fun(int *a,char *buf,int bufLen)
{
    *a = *a * 10;
    strcpy_s(buf, bufLen, "xxxx");
    return "abcdef";
}
```

```
#coding=utf-8
import ctypes
#加载 dll
dl = ctypes.cdll.LoadLibrary(r'C:\DT.dll')

#创建字符串缓冲区
bufLen = 100
strBuf = ctypes.create_string_buffer('\0',bufLen)

#创建 int*  byref 将数值转换为指针
a = ctypes.c_int(5)
aRef = ctypes.byref(a)
```

```
#执行 dll 中的函数
pchar = dl.fun(aRef,strBuf,bufLen)

#将 dll 返回的 char*进行转换
szbuffer = ctypes.c_char_p(pchar)


print a.value
print szbuffer.value
print strBuf.value

"""
sBuf = '123456789'
pStr = ctypes.c_char_p( )
pStr.value = sBuf
bufLen = len(pStr.value)
"""
```

17.7.2.2. 类型映射

ctypes 数据类型	C 数据类型
c_char	char
c_short	short
c_int	int
c_long	long
c_ulong	unsign long
c_float	float
c_double	double
c_void_p	void

对应的指针类型是在后面加上"_p"，如 int*是 c_int_p

17.7.2.3. 结构体交互

```
struct  DatStu
{
    int    num;
    char   val[512];
};
extern "C" __declspec(dllexport) void fun(DatStu *stu)
{
    stu->num = 10;
    strcpy_s(stu->val, 512, "xxx");
}
```

```
class DatStu(ctypes.Structure):
    _fields_ = [ ("num", ctypes.c_int),
                 ("val", ctypes.c_char * 512)]


dl = ctypes.cdll.LoadLibrary(r'C:\DT.dll')

dat = DatStu()
dl.fun(ctypes.byref(dat))

print dat.num
print dat.val
```

17.7.3. c++导出 pyd

========dll 编写

```
#include "Python.h"
static PyObject *ex_foo(PyObject *self, PyObject *Argvs)
{
    int Argv1(0), Argv2(0);
    if (!PyArg_ParseTuple(Argvs, "ii", &Argv1, &Argv2))
    {//ii 表示解析出来两个 int 型参数
        cout << "parse param failed" << endl;
        return NULL;
    }
```

```
    cout << Argv1 << "+ " << Argv2 << " = " << Argv1 + Argv2 << endl;

    Py_INCREF(Py_None);//Py_INCREF 增加 PyObject 对象的引用计数
            //Py_DECREF 减小 PyObject 对象的引用计数
           //为了让 Python 回收废弃的内存、或者防止 Python 过早地自动回收内存
    //必须用 Py_EDCREF 和 Py_INCREF 来控制 PyObject 的引用计数
    return Py_None;
}


static PyMethodDef ModulesMethods[] =
{
    { "fun", ex_foo, METH_VARARGS, "fun() doc string" },
    { NULL, NULL }
    //fun 是导出函数名
    //METH_VARARGS，则用 Tuple 来传递参数   METH_KEYWORDS，则用 Dictionary 的 Key（键值）来传递参数
    //"fun() doc string"  函数的说明字符串。在 Python 中就是函数的 DocString __doc__。
};
static struct PyModuleDef ModuleDesc =
{    PyModuleDef_HEAD_INIT,
     "Module", //内置的模块名   模块名.__name__来获取这个字符串
    "This module is created by C++. And it Add two Integer s!", //模块的 DocString，也可以用模块名.__doc__获得
    -1, //-1  模块在全局范围
    ModulesMethods
};


PyMODINIT_FUNC PyInit_DllName(void)//生成文件名必须 DllName.pyd
{
   //Py_InitModule("DllName", ModulesMethods);  py2
    return PyModule_Create(&ModuleDesc);
}
```

========使用

```
import DllName

DllName.fun(1,3)
```

17.8. extend

17.8.1. requests

pip install requests

访问本机网址时

```
url = 'http://localhost:8080/abc'
```

17.8.1.1. 发送请求

```
header = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.140 Safari/537.36 Edge/17.17134'}

    url = '******'
    response =  requests.Response()
    response = requests.get(url,headers = header)
    contentTxt = response.text
    contentBin = response.content
    contentJson = response.json()
```

17.8.1.2. 发送 post 请求

```
url = 'http://.......'
d = {'userName': 'abc', 'passWord': '123'}
r = requests.post(url, data=d)
```

17.8.2. html 解析

17.8.2.1. bs4

```
from bs4 import BeautifulSoup
```

方法	说明
soup = BeautifulSoup(txt,"html.parser")	读取 html 文件字符串生成对象，解析器可以用 html.parser 或者 lxml
obj = soup.tagName txt = obj .text	选择第一个 tagName 对象 返回值为一个新的 BeautifulSoup 对象 text 是标签包围的内容

	soup.tagName.tagName2 多级选择
val=soup.tagName['attr']	选择第一个 tagName 对象的 attr 属性，如果属性是 class，则返回列表，其他则返回字符串
.parent	选择父节点
obj = soup.find('tagName') obj = obj.find_next('tagName')	选择节点，用结果选择下一个
obj = soup.find(''tagName'',attr= 'aa')	选择''tagName''节点中 attr 属性为 aa 的节点
obj = soup.find(''tagName'',class_= 'bb')	选择''tagName''节点中 class 属性包含 bb 的节点
obj = soup.find(class_= 'bb')	选择 class 属性包含 bb 的节点
soup.find_all(''tagName')	选择所有 tagName 节点 返回 BeautifulSoup 对象列表

17.8.2.2. lxml

```
from lxml import etree
```

方法	说明
xpathObj=etree.HTML(text)	读取 html 文本字符串初始化生成一个 XPath 解析对象
xpathObj=etree.parse(obj,etree.HTMLParser())	obj 可以是文件路径，解析 html 文件并自动纠正 html 内容错误
result=etree.tostring(xpathObj,encoding='utf-8') txt = result.decode('utf-8')	转换出对应的字符串数据
obj = xpathObj.xpath('//li[@class="jyl"]') for val in obj: print(val.text)	xpath 返回 list 对象 text 是标签包围的字符(不包括子标签内容)

17.8.3. numpy

17.8.3.1. 数据转换

pythno list 转换成矩阵

```
X = [[-4,-2,0,2], [-4,-2,0,2],[-4,-2,0,2],[-4,-2,0,2]]
X=np.array(X)
```

17.8.3.2. 矩阵生成

类型	代码	结果
自定义生成矩阵	def fun(x,y): return y np.fromfunction(fun,(3,3))	[[0 1 2] [0 1 2] [0 1 2]]
空矩阵	np.empty((3,3))	[[0 0 0] [0 0 0] [0 0 0]]
赋值生成矩阵	np.array([[1,0],[0,1]])	[[1 0] [0 1]]
线性矩阵	np.arange(1,8,2)	[1 3 5 7]
矩阵纬度变化	V = np.arange(1,8,2) V.shape = 2,2	将[1 3 5 7]变成 [[1 3] [5 7]]

17.8.4. matplotlib

17.8.4.1. 曲线绘制

```
import matplotlib.pyplot as plt
from pylab import *    #中文支持
mpl.rcParams['font.sans-serif'] = ['SimHei']

x = [5,20,25,30,50]
y1 = [0.855, 0.84, 0.835, 0.815, 0.81]
y2=[0.86,0.85,0.853,0.849,0.83]
plt.plot(x, y1, marker='o', mec='r', mfc='w',label=u'曲线 1')
```

```
plt.plot(x, y2, marker='*', ms=10,label=u'y=曲线 2')
plt.legend()
#生成 X 的别名（X 实时显示失效）
#xNames = ['5', '20', '25', '30', '50']
#plt.xticks(x, xNames, rotation=45)
plt.margins(0)
plt.subplots_adjust(bottom=0.15)
plt.xlabel(u"X")
plt.ylabel("Y")
plt.title("title")
plt.show()
```

17.8.4.2. 三维绘制

```
from matplotlib import pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D


fig = plt.figure()
ax = Axes3D(fig)

#X = np.arange(-4, 4, 2)
#Y = np.arange(-4, 4, 2)
#X, Y = np.meshgrid(X, Y)
#R = np.sqrt(X**2 + Y**2)
#Z = np.sin(R)

X = [[-4,-2,0,2], [-4,-2,0,2],[-4,-2,0,2],[-4,-2,0,2]]
X=np.array(X)
Y = [[-4,-4,-4,-4],[-2,-2,-2,-2],[ 0,  0,  0,  0],[ 2,  2,  2,  2]]
Y=np.array(Y)
Z=[[-0.58617619,-0.9712778,-0.7568025,-0.9712778 ],[-0.9712778,0.30807174,0.90929743,0.
30807174],[-0.7568025,0.90929743,0.          ,0.90929743],[-0.9712778,0.30807174,0.909297
43,0.30807174]]
Z=np.array(Z)

#ax.scatter(X, Y, Z)          散点
#ax.plot_wireframe(X, Y, Z)  连线
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='rainbow')    曲面

plt.show()
```

17.8.5. pillow

pip install pillow

```
from PIL import Image
import  numpy as np

a = np.array(Image.open("D:\\1.jpg"));
b = [255,255,255] - a
im = Image.fromarray(b.astype("uint8"))
im.save("D:\\2.jpg")
```

```
im = Image.open(filePathName)
out = im.resize((200, 100), Image.ANTIALIAS)    # 图像缩放为 200*100
out.save(outfile)
```

17.8.6. function

17.8.6.1. 财经

pip install tushare
import tushare as ts
ts.get_today_all()

17.8.6.2. 调用 opencv

需要安装 numpy
使用 cv2.pyd

```
import cv2
img = cv2.imread("D:\\1.jpg")
cv2.namedWindow("Image")
cv2.imshow("Image", img)
cv2.waitKey (0)
cv2.destroyAllWindows()
```

17.8.6.3. 线性回归

```
from sklearn.linear_model import LinearRegression


x = [[1], [2], [3]]
y = [[2], [4], [6]]


model = LinearRegression()
model.fit(x, y)
x2 = [[9]]
y2 = model.predict(x2)
print(x2,y2)
```

17.8.6.4. 生成二维码

pip install Pillow
pip install qrcode

```
import qrcode
import PIL


url = 'http://....'

code_maker = qrcode.QRCode(error_correction=qrcode.constants.ERROR_CORRECT_H)
code_maker.add_data(url)
code_maker.make(fit=True)
code_image = code_maker.make_image()
code_image = code_image.convert('RGBA')

face_image = PIL.Image.open('xxxxxx.png')
face_image = face_image.resize((100,100), PIL.Image.ANTIALIAS)

code_width, code_height = code_image.size
code_image.paste(face_image, ((code_width-100)//2, (code_height-100)//2))
code_image.save(r'qrcode.png')
```

17.8.7. flask

```
from flask import Flask
app = Flask(__name__)
```

```
@app.route('/')
def hello_world():
    return 'Hello, World!'

app.run()
```

17.8.7.1. 启动程序

app.run('0.0.0.0',1234)　监听所有 ip，端口 1234

17.8.7.2. 路由

路由末尾如果包含/，则访问时不包含/则自动加上/
路由末尾如果不包含/，则行为表现与一个文件类似，如果访问加/则 404 错误

from flask import url_for with app.test_request_context(): 　　pathstr = url_for('static', filename='style.css') 　　print(pathstr) 　　url_for('index')	获取路由地址
@app.route('/jyl/<paramname>') def fun(paramname): 　　return 'p:%s'%(paramname)	路由中获取 url 末尾的变量 可以限制 paramname 类型(默认 string)： <int:paramname> <float:paramname> <path:paramname> 可以包含斜杠的字符串
@app.route('/login', methods=['GET', 'POST']) def login(): 　　if request.method == 'POST': 　　　　return do_the_login() 　　else: 　　　　return show_the_login_form()	使路由可以接受 post 和 get 方法

17.8.8. Django

17.8.8.1. 下载

git clone https://github.com/django/django.git

17.8.8.2. 安装

python setup.py install

使用 python 命令运行 django 下载目录中的 setup.py
安装到 python 目录下 Lib\site-packages

PYHOME C:\py363
Path：%PYHOME%;%PYHOME%\Scripts;
%PYHOME%\Lib\site-packages\Django-2.1-py3.6.egg

17.8.8.3. 创建项目

django-admin-script.py startproject HelloWorld

创建的目录：
HelloWorld: 项目的容器。
manage.py: 一个实用的命令行工具，可让你以各种方式与该 Django 项目进行交互。
HelloWorld/__init__.py: 一个空文件，告诉 Python 该目录是一个 Python 包。
HelloWorld/settings.py: 该 Django 项目的设置/配置。
HelloWorld/urls.py: 该 Django 项目的 URL 声明; 一份由 Django 驱动的网站"目录"。
HelloWorld/wsgi.py: 一个 WSGI 兼容的 Web 服务器的入口，以便运行你的项目。

17.8.8.4. 启动服务

python manage.py runserver 0.0.0.0:8000

http://127.0.0.1:8000/

17.8.8.5. MVC

=======HelloWorld 目录下创建 view.py
【创建运行文件】

```
from django.shortcuts import render


def hello(request):
    context = {}
    context['var'] = '****************'
    return render(request, 'view.html', context)
```

使用字典类型的参数改变页面值

========urls.py 修改

【运行文件绑定到启动页面】

```
from django.conf.urls import  url
from .  import  view

urlpatterns = [
    path('admin/', admin.site.urls),
    url(r'^$',view.hello)
]
```

或者 `url(r'^hello$',view.hello)`

每个 url 代表一个地址栏输入的地址

=======HelloWorld 目录同级（manage.py 所在目录）下创建 viewDir 目录

【创建 py 文件绑定的模板】

创建 view.html

包含内容 {{ var }}

var 为 py 文件可修改的值

=======settings.py 修改

【将模板加入设置】

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
 'DIRS': [os.path.join(BASE_DIR,'viewDir').replace('\\', '/')],  #修改
.....
.....
```

url() 可以接收四个参数，分别是两个必选参数：regex、view 和两个可选参数：kwargs、name
regex：正则表达式，与之匹配的 URL 会执行对应的第二个参数 view。
view：用于执行与正则表达式匹配的 URL 请求。
kwargs：视图使用的字典类型的参数。
name：用来反向获取 URL。

直接返回文字页面

```
from django.http import HttpResponse
return HttpResponse("hello world")
```

17.8.8.6. 静态文件使用

工程目录\static\img　　包含 1.jpg

settings.py 末尾

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static"),
    '/var/www/static/',
]
```

引用时

```
{% load static %}
<img src="{% static "img/1.jpg" %}"/>
```

17.8.8.7. 模板标签

17.8.8.7.1. if/else

基本语法格式如下：

```
{% if condition %}
     ... display
{% endif %}
```

或者：

```
{% if condition1 %}
```

```
  ... display 1
{% elif condition2 %}
  ... display 2
{% else %}
  ... display 3
{% endif %}
```

根据条件判断是否输出。if/else 支持嵌套。

{% if %} 标签接受 and ， or 或者 not 关键字来对多个变量做判断 ，或者对变量取反（ not ），例如：

```
{% if athlete_list and coach_list %}
    athletes 和 coaches 变量都是可用的。
{% endif %}
```

17.8.8.7.2. for

{% for %} 允许我们在一个序列上迭代。

与 Python 的 for 语句的情形类似，循环语法是 for X in Y ，Y 是要迭代的序列而 X 是在每一个特定的循环中使用的变量名称。

每一次循环中，模板系统会渲染在 {% for %} 和 {% endfor %} 之间的所有内容。

例如，给定一个运动员列表 athlete_list 变量，我们可以使用下面的代码来显示这个列表：

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% endfor %}
</ul>
```

给标签增加一个 reversed 使得该列表被反向迭代：

```
{% for athlete in athlete_list reversed %}
...
{% endfor %}
```

可以嵌套使用 {% for %} 标签：

```
{% for athlete in athlete_list %}
    <h1>{{ athlete.name }}</h1>
    <ul>
    {% for sport in athlete.sports_played %}
        <li>{{ sport }}</li>
    {% endfor %}
    </ul>
{% endfor %}
```

17.8.8.7.3. ifequal/ifnotequal

{% ifequal %} 标签比较两个值，当他们相等时，显示在 {% ifequal %} 和 {% endifequal %} 之中所有的值。

下面的例子比较两个模板变量 user 和 currentuser ：

```
{% ifequal user currentuser %}
    <h1>Welcome!</h1>
{% endifequal %}
```

和 {% if %} 类似， {% ifequal %} 支持可选的 {% else%} 标签：8

```
{% ifequal section 'sitenews' %}
    <h1>Site News</h1>
{% else %}
    <h1>No News Here</h1>
{% endifequal %}
```

17.8.8.7.4.注释

Django 注释使用 {# #}。

```
{# 这是一个注释 #}
```

17.8.8.7.5.过滤器

模板过滤器可以在变量被显示前修改它，过滤器使用管道字符，如下所示：

```
{{ name|lower }}
```

{{ name }} 变量被过滤器 lower 处理后，文档大写转换文本为小写。

过滤管道可以被* 套接* ，既是说，一个过滤器管道的输出又可以作为下一个管道的输入：

```
{{ my_list|first|upper }}
```

以上实例将第一个元素并将其转化为大写。

有些过滤器有参数。 过滤器的参数跟随冒号之后并且总是以双引号包含。 例如：

```
{{ bio|truncatewords:"30" }}
```

这个将显示变量 bio 的前 30 个词。

其他过滤器：

addslashes : 添加反斜杠到任何反斜杠、单引号或者双引号前面。

date : 按指定的格式字符串参数格式化 date 或者 datetime 对象，实例：

```
{{ pub_date|date:"F j, Y" }}
```

length : 返回变量的长度。

17.8.8.7.6.include

{% include %} 标签允许在模板中包含其它的模板的内容。

下面这个例子都包含了 nav.html 模板：

```
{% include "nav.html" %}
```

17.8.8.7.7.模板继承

模板可以用继承的方式来实现复用。

templates 目录中添加 base.html 文件，代码如下：

HelloWorld/templates/base.html 文件代码：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>tttt</title>
</head>
<body>
    <h1>Hello World!</h1>
    <p>aaaaaaa</p>
    {% block mainbody %}
       <p>original</p>
    {% endblock %}
</body>
</html>
```

以上代码中，名为 mainbody 的 block 标签是可以被继承者们替换掉的部分。

所有的 {% block %} 标签告诉模板引擎，子模板可以重载这些部分。

hello.html 中继承 base.html，并替换特定 block，hello.html 修改后的代码如下：

HelloWorld/templates/hello.html 文件代码：

```
{% extends "base.html" %}

{% block mainbody %}<p>继承了 base.html 文件</p>
{% endblock %}
```

17.8.9. mysql

pip install PyMySQL

```
import pymysql.cursors
connection = pymysql.connect(host='127.0.0.1', port=3306, user='root', password='root', db='testdb',
                             charset='utf8mb4', cursorclass=pymysql.cursors.DictCursor)
cursor = connection.cursor()

sql = "SELECT * FROM user"
cursor.execute(sql)
result = cursor.fetchall()
for data in result:
    print(data)

connection.commit()
```

17.8.10. pymongo

pip install pymongo

import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")	打开 mongodb
dblist = myclient.list_database_names()	获取数据库列表
选择数据操作集合	
mydb = myclient['jyl']	选择数据库
mycl = mydb["jylcl"]	选择数据库集合
增加	
mydict = { 'name':'aaa','age':11} x = mycl.insert_one(mydict) id = x.inserted_id	向集合中插入文档 插入时"_id"可以指定插入数据的 id，不指定则自动分配 insert_many()插入多个数据，输入参数为列表中包含多个集合
修改	
myquery = { "name": "aaa" } newvalues = { "$set": { "age": "100" } } mycl.update_one(myquery, newvalues)	设置集合中 name 为 aaa 的文档中 age 为 100
删除	
myquery = { 'name':'aaa' } mycl.delete_one(myquery)	从集合中删除指定文档
myquery = { "name": {"$regex": "^j"} } x = mycl.delete_many(myquery) count = x.deleted_count	从集合中删除所有 name 中首字母 j 开头的文档
x = mycl.delete_many({}) count = x.deleted_count	删除集合中所有文档
查询 取出的每个文档数据为字典类型	
x = mycl.find_one()	查询集合中第一个文档
for x in mycl.find(): print(x)	查询集合中所有文档
myquery = { "name": "aaa" } for x in mycl.find(myquery): print(x)	查询集合中所有 name 为 aaa 的文档
myquery = { "name": { "$regex": "^a"} }	查询集合中所有 name 为 a

for x in mycl.find(myquery): print(x)	开头的文档
for x in mycl.find().sort("age", -1): print(x)	将查询的结果降序排列，参数 2 不写或为 1 则为升序

17.8.11. pyqt

17.8.11.1. 安装

17.8.11.1.1. 模块安装

pip install pyqt5
pip install pyqt5-tools

17.8.11.1.2. pycharm 配置

settings->tools->external tools 进行添加
通过配置可以在 pycharm 中 tools->external tools 直接调用对应的 pyqt 程序

=======设置 qtdesigner 程序打开方式
Name:qtdesigner
Program：pyqt5-tools 安装目录中 designer.exe 的路径
Work directory : $FileDir$

=======设置 pyuic
Name: pyuic
Program： pythonEXE 路径
Parameters：
-m PyQt5.uic.pyuic $FileName$ -o $FileNameWithoutExtension$.py
Work directory: $FileDir$

=======使用
使用 qtdesigner 生成***.ui
选择该 ui 文件，执行 pyuic 生成对应的.py 文件
【运行时执行的是：python.exe -m PyQt5.uic.pyuic **.ui -o **.py】

17.8.11.2.ui 调用

```
app = QtWidgets.QApplication(sys.argv)
widget = QtWidgets.QWidget(None)
ui = Ui_Form()
ui.setupUi(widget)
widget.show()
sys.exit(app.exec_())
```

Ui_Form 为 ui 自动生成 py 文件中的类

17.8.11.3.消息响应

```
class MyApp:
    def __init__(self):
        self.widget = QtWidgets.QWidget(None)
        self.ui = Ui_Form()
        self.ui.setupUi(self.widget) #产生 btnObj 对象
        self.ui.btnObj.clicked.connect(self.fun)  # ui 中的 btnObj 对象
        self.widget.show()
    def fun(self):
        print("xxxx")

app = QtWidgets.QApplication(sys.argv)
obj = MyApp()
sys.exit(app.exec_())
```

17.8.12. Scrapy

17.8.12.1.安装

Twisted 需要 vc 编译工具，可以使用 Twisted 的 whl 进行安装。

```
pip install scrapy

pip install pypiwin32
```

17.8.12.2.创建项目

```
scrapy startproject 项目名
```

(Scripts 加入 path 环境变量)

17.8.12.3.建立爬虫

spiders 目录下建立文件
jyl_spider.py

```
import scrapy
class BlogSpider(scrapy.Spider):
    name = 'jyl' #爬虫类的 name 属性，用来标识爬虫，该名字在一个项目必须是唯一的。
    #allowed_domains = []   #限制只爬取这个域名下的网页
    start_urls = ['https://blog.scrapinghub.com']

    def parse(self, response):
        for title in response.css('h2.entry-title'):
            yield {'title': title.css('a ::text').extract_first()}
```

17.8.12.4.运行

项目目录（scrapy.cfg 同级）下运行
scrapy crawl jyl

文件目录下运行
scrapy runspider jyl_spider.py -o ret.json
将（yield ）输出 ret.json 文件[可以是 csv/XML]

17.8.12.5.调试

项目中建立新文件作为启动运行文件，文件内容：

```
from scrapy.cmdline import execute

#import os
#import sys
#添加当前项目的绝对地址   sys.path.append(os.path.dirname(os.path.abspath(__file__)))

#执行 scrapy 内置的函数方法 execute，使用 crawl 爬取并调试，最后一个参数是爬虫名
execute(['scrapy', 'crawl', 'jyl'])
```

17.8.12.6.设置

settings.py 设置

```
ROBOTSTXT_OBEY = False


DEFAULT_REQUEST_HEADERS = {
    'User-Agent':    'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36',
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
    'Accept-Language': 'en',
}
```

17.8.12.7.pipelines

```
======创建 piplelines
class JylPipeline(object):
    def process_item(self, item, spider):
        name = item['name']
        return item     #继续传递 item，下一个 Pipeline 可以接收
        #raise DropItem("drop %s" % item)     #扔掉 item，下一个 Pipeline 不再接收


======settings.py 设置加入 piplelines
ITEM_PIPELINES = {
    'jyl.pipelines.JylPipeline': 0,
}
```

0 表示优先级数字，数字越小优先级越高

```
======调用方式 1：
item 定义：
class JylItem(scrapy.Item):
    name = scrapy.Field()



from jyl.items import JylItem

ii = JylItem()
ii['name'] = imgSrc
yield ii
```

======调用方式 2：
yield {"name": imgSrc}

17.8.12.8.创建请求

爬虫类中，创建新请求，并使用当前类的回调函数

yield scrapy.Request(new_url,callback=self.parse)

17.8.12.9.response

属性	说明
url (string)	response 的 URL
status (integer)	
headers (dict)	response 头
body (bytes)	如果是字符串可以用 response.text
flags (list)	
request (Request object)	Response.request 初始属性
xpath	for imgSrc in response.xpath('//img/@src').extract() 遍历所有 img 标签的 src 属性
	.extract() 获取 xpath 对象内的所有数据，返回列表对象 .extract_first()返回列表第一个数据

17.8.12.10. shell

scrapy shell "http://www.budejie.com/text/"
打开 http://www.budejie.com/text/并进入 shell 命令

view(response)，打开默认浏览器并进入相应页面
response.css('title')，查看 title 的 css 样式
response.css('title::text').extract_first()，获取标题中内容

第十八章 DATABASE

18.1. SQL

18.1.1. 操作

18.1.1.1. 数据库选择

show databases;	显示所有数据库
create database abc;	创建数据库 abc
use abc;	使用数据库 abc

18.1.1.2. 表操作

create table aa(id int primary key, name varchar(30),age int);	创建表 aa id，name，age
desc aa；	显示表 aa 中的字段名
drop table aa;	删除表 aa
create table aa(id int primary key auto_increment,data int); insert into aa(data) values(33); insert into aa(data) values(22);	主键 id 自动递增
begin 。。。。。。大量操作	加快操作 commit； 修改数据后提交，否则

commit	其他终端无法操作该数据

18.1.1.3. 插入

insert into aa values(1,'n1',11);	向表 aa 中插入一条数据
insert into aa(id) values(4);	向表 aa 中插入一条数据

18.1.1.4. 删除

delete from aa where id=4 and name='n1';	删除表 aa 中所有 id 为 4,name 为 n1 的数据
truncate table aa;	清空表 aa 中所有数据，比 delete from aa；效率高。 delete 不释放空间，truncate 会彻底释放空间

18.1.1.5. 修改

update aa set age=55,name='xx' where id=1;	修改表 aa 中的字段

18.1.1.6. 查找

select * from aa;	选择 aa 中所有数据
select name from aa;	选择 aa 中所有 name 数据
select name,age from aa;	选择 aa 中所有 name 和 age 数据
条件查询	
select * from aa where id=1;	选择 aa 中所有 id 为 1 的数据
select min(id),name,age from aa where id>=5;	选择 aa 中 id>=5 的数据中 id 最小的数据
select * from aa where id between 5 and 30;	查找 id 在[5,30]范围内的数据
select * from aa where id in(5,10,15);	查找 id 为括号内列举值的数据
select distinct name from aa;	name 字段相同时排除
select * from aa order by id desc;	查找的数据按照 id 从大到小排序（不写 desc 默认从小到大） 排序时，null 作为最大值
select * from aa where name like '+%+';	查找所有 name 以+开头，+结尾的字段【% 任意数量字符，_ 单个字符】
子查询	

select name,age from (select * from aa where id>15) as temp;	as temp 作为()内查询结果的别名，虽然没有被使用但是必须
多表查询 自连接：一个表中放入了逻辑上两种不同的数据 内连接：符合 where 条件，数据就被选中，不符合 where 条件就被过滤 外连接：等于内连接加上匹配不上的记录（全部被选中）	
select aa.id,name from aa,cc where aa.id=bb.id; sql99 写法： select aa.id name from aa inner join cc on aa.id=cc.id;	从 aa 和 bb 两张表中进行查询，限制条件为两个表的 id 相同（显示字段时,如果该字段两个表都有，需要用 aa.id 这种语法区分）
select aa.id name from aa left outer join cc on aa.id=bb.id;	在满足 aa.id=cc.id 情况下，将 aa 里的数据全部也取出来
select aa.id name from aa right outer join cc on aa.id=cc.id;	在满足 aa.id=cc.id 情况下，将 cc 里的数据全部也取出来
select id from aa union	将两个表的查询结果合并起来

select id from cc;	

18.1.2. 功能

生成密码：password("abc")

获取密码字符：select password("abc") password

18.1.3. sql 执行顺序

from->where->group by->having->select->order by

在分组语句中，select 后的字段必须是分组标准或经过组函数处理过

18.1.4. 数据类型

18.1.4.1. 基本类型

number

number(7) 宽度 7 不加则不限制宽度

number(7,2) 小数点后占两位，总长为 7

char

char(7) 定长字符串 不够则补 0，省时间

varchar2(7) 变长，可以数量不足，省空间

18.1.4.2. date 类型

1）默认格式 ‘dd-mon-yy’

2) sysdate

exp: insert into 表名 values(... '10-dec-13');

或者将字符串写成 sysdate

取出日期：

to_char(要处理的日期，'日期格式')

日期格式：yyyy，mm，dd，hh24，mi，ss

exp：

select to_char(日期所在字段名，'yyyy-mm-dd') from 表名；

格式字符串中间的符号-可任意

day 星期

mon 三个字母的月的缩写

month 全写

pm 表示出 pm 或者 am

放入任意日期：

to_date(日期字符串，格式字符串)

exp：

to_date('2010-10-10 13;10:25','yyyy-mm-dd hh24:mi:ss')

格式字符串要与日期字符串保持一致

对日期进行调整：

按天进行调整：

select　to_char(日期所在字段名-1，’yyyy-mm-dd’) from 表名；

-1 代表减少一天

1/(24*60*60)　代表一秒

add_months(日期所在字段名，-1)

减少一个月

给定一个时间，得到这个时间对应月的最后一天的日期

select　to_char(last_day(sysdate),’yyyy-mm-dd　hh24:mi:ss’) from dual；

只有 dd 发生变化

next_day(日期，’星期二’)

跳到下一个星期二

round 默认以天为单位进行四舍五入，如果一天的时间过了一半，则变为新的一天（天后面的时间进行四舍五入，保留到天）

或者可以指定单位进行操作：

round(sysdate,’mm’)

trunc 默认以天为单位进行截取

如果以月为单位进行截取，则将小时，分秒清空，日期变为 1

得到一个月的最后一天的最后一秒：

trunc(last_day(sysdate)+1)-1/(24*60*60)

变成下个月的开始再减去一秒

18.1.5. 运算符

is null 判断空

and

or

not

18.1.6. 函数

单行函数：针对每一行数据都做处理，sql 语句影响多行，数据就返回多少个结果

组函数：针对每一组数据进行处理，无论影响多少行，只返回一个结果

upper()	
lower()	
initcap()	把字符串每个单词的首字母变大写
concat('s1','s2')	
length()	
substr(s1,i,n)	s1 为要截取的字符串，n 代表截取的长度，i 代表开始位置（-1 代表最后一个字符，-3 代表倒数第三个）

round	四舍五入 round(x,0) 代表保留小数点后 0 位，默认（可不写）
trunc	截取
to_char(字段，格式字符串)	格式显示函数： 格式字符串可以省略，代表把数据变成字符串类型 fm 格式字符串开始 9 任意数字 0 强制显示前导 0 $ L 本地货币符号 ， 分隔符号 . 小数点 to_char(salary,”fm$099,999,00”)
分组	常见组函数：count，max，min，sum，avg select count(id),max(salary) from s_emp; 显示 id 总量，salary 最大值 组函数中可以使用 distinct 关键字 sum(distinct salary)

	组函数对 null 的处理是忽略的
过滤	对组数据进行过滤（having） select dept_id,avg(salary) from s_emp group by dept_id having avg(salary)>2000; 只显示大于 2000 的

18.1.7. 事务操作

update.....;

savepoint a; 设置断点 a

update.....;

savepoint b; 设置断点 b

update.....;

rollback to b; 撤销到 b 点的操作结果

18.1.8. 数据库约束

对数据库中表的字段的值加一些限制和保护，是对数据保护的最后一道屏障。

主键约束：primary key 字段值不能为 null，且不重复

一个表中只能有一个主键

唯一性约束：unique　　字段值不能重复

非空约束：not　null　字段值不能为 null

检查约束：check　字段值必须符合检查条件

外键约束：references　foreign　key

on　delete　cascade

on　delete　set　null

列级约束：在定义表的每一列时，只直接对这一列加约束

create　table　abc(id number　primary　key);

表级约束：在定义完表的所有列之后，再选择某些列加约束

如果没有给约束起名，系统会自动分配一个约束名

id　number　constraint　xy　primary　key

自己给约束起名为 xy，不满足条件时，报错时会出现 xy

检查约束：　id　number　check(id>10)

id 加入时必须大于 10

表级约束：

not null 无表级约束

1）create　table　abc(id number, salary　number,　constraint　xy primary　key(id, salary));只要有一个满足约束即可

2）create　table　abc(id　number, salary　number, primary key(id),not　null(salary))

外键约束：外键字段的值引用自主表一个字段的值，受限于主表

1 2 主 键 字段	主表

1 2 null 1	从表

只能用主表中已存在的字段 1，2 和 null

先建主表：create table abc(id number primary key);

后建从表：create table def(salary number, sid number
references abc(id));

插入数据时，先插入主表，才能插入从表

删除数据时，先删子表的，才能删除主表的

删除表时，先删除从表，再删除主表，除非使用：

drop table 从表 cascade constraints；

加 s 代表断开多个关系

级联删除：

在建从表时：

sid number references abc(id) on delete cascade

当删除主表某个字段时：

delete from abc where id=1；

从表中所有 sid 等于 1 行的都随之删除

级联置空：

使用 on delete set null 时

从表中所有 sid 等于 1 的行都将 1 变为 null

18.1.9. 数据库其他对象

18.1.9.1. 序列

生成主键的值

1）创建序列

create sequence sq； sq 为序列名

2）测试序列

create sq.nextval from dual;

3)删除序列

drop sequence sq；

4）使用序列

create abc(id number primary key);

insert into abc values(sq.next.nextval);

sq.currval||’a’

可以将两个字符串拼接成一个字符串

没调用一次 sq.nextval,sq 的值就变化

5)复杂序列

create sequence sq MAXVALUE 9999;

增加额外限制信息

18.1.9.2. 索引

加速查询

1）建立索引

主键和唯一键上，系统会自动建立索引

create index 索引名 on 表名(字段名)；

2）删除索引

drop index 索引名；

18.1.9.3. 视图

本质对应的是一条 sql 语句，相对于视图对应的真实数据，视图的空间可以忽略不计

可以对同一份物理数据做不同表现

建立视图：

create or replace view 视图名 as select id，first_name from s_emp;

为一条 sql 语句建立一个视图

视图 select *from 视图名 where id=2；

删除 drop view 视图名；

18.1.9.4. 分页技术

rownum 伪列，用来计数

select rownum,id,first_name from s_emp where rownum<12;

不能使用 rownum<22 and rownum>12，因为从 rownum=1 开始运行，一开始不满足就退出

使用：

select * from(select rownum r, id from s_emp where rownum<22)where r>12;

18.1.10. sql 三范式

1）表中的字段不可分

<table>
<tr><td rowspan="3">字段 1</td><td colspan="2">字段 2</td></tr>
<tr><td>字段 3</td><td>字段 4</td></tr>
<tr><td colspan="2"></td></tr>
</table>

不能将字段 2 进行拆分

2）表中所有非空属性必须依赖主属性

3）在第二范式的基础上消除了传递依赖

如表中有员工和员工所属部门，如果放在同一张表，则会出现员工不重复，但是所属部门会重复，增加删除不方便

传递依赖：id 决定员工号，员工号决定员工名。三个数据或者更多的之间相互传递。

提高范式，就需要折表，减少传递。

18.2. MySql

18.2.1. 配置

18.2.1.1. 安装

环境变量：
path 加入 C:\Program Files\mysql-8.0.13-winx64\bin

将 mysql-8.0.13-winx64 目录放入 C:\Program Files
将 my.ini 放入 mysql-8.0.13-winx64 目录

==========my.ini==========
[mysql]

```
default-character-set=utf8
[client]
port=3306
default-character-set=utf8
[mysqld]
port=3306
basedir="C:\Program Files\mysql-8.0.13-winx64"
datadir="C:\Program Files\mysql-8.0.13-winx64\data"
max_connections=200
character-set-server=utf8
default-storage-engine=INNODB
default_authentication_plugin=mysql_native_password
```

运行 intall.bat：

```
cd C:\Program Files\mysql-8.0.13-winx64\bin
mysqld      --initialize-insecure
mysqld      --install MySQL --defaults-file="C:\Program Files\mysql-8.0.13-winx64\my.ini"
net start mysql
pause
```

登录：

```
mysql -u root -p
use mysql;
update user set authentication_string='' where user='root';

ALTER user 'root'@'localhost' IDENTIFIED BY '新密码'
【非 mysql_native_password 时：
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '新密码'
FLUSH PRIVILEGES;
】
```

卸载

```
net stop mysql
sc delete MySQL
```

18.2.1.2. 卸载

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Eventlog\Application\MySQL 文件夹删除

C:\ProgramData\MySQL 删除

删除 C:\Documents and Settings\All Users\Application Data\MySQL

18.2.2. 导入

进入 db
source d:\rmt.sql

18.2.2.1. 服务器运行

=======启动服务器
管理员模式运行 cmd
net start mysql

=======停止
net stop MySQL

18.2.2.2. 登录

mysql -u root -p（第一次登录没有密码，直接按回车过）

mysql -h30.158.59.78 -u 用户名 -p** //远程登录 **代表密码

18.2.2.3. 查看状态

mysql> status

18.2.2.4. 修改密码

mysql> set password =password('密码');
mysql> flush privileges;

18.2.3. 用户管理

以 root 用户登录

18.2.3.1. 创建用户

insert into mysql.user(Host,User,Password) values("localhost","abc",password("1234"));

insert into mysql.user(Host,User,Password) values("192.168.125.133","abc",password("1234"));
//为 192.168.125.133 创建用户 abc

flush privileges;

18.2.3.2. 删除用户

DELETE FROM mysql.user WHERE User="abc" and Host="localhost";

18.2.3.3. 授权

//为用户创建一个数据库
create database db;
//授权 phplamp 用户拥有 phplamp 数据库的所有权限。
grant all privileges on phplampDB.* to phplamp@localhost identified by '1234';
//刷新系统权限表
flush privileges;

//指定部分权限给一用户
mysql>grant select,update on db.* to abc@localhost identified by '1234';
//刷新系统权限表
mysql>flush privileges;

18.2.3.4. 修改用户密码

update mysql.user set password=password("new password") where User="abc" and Host="localhost";
mysql>flush privileges;

18.2.4. 创建/使用数据库

create database ***;

use ***;

18.2.5. 数据类型

int,integer,bigint
float,double
char,varchar

datetime: 2017-12-12 17:00:00
date:2017-12-12
time:17:00:00

18.2.6. 密码

md5(“password”) 根据字符串创建密码

18.3. mongodb

18.3.1. 环境配置

环境变量 path: C:\Program Files\MongoDB\Server\4.0\bin

安装启动服务：
mongod.exe --install
net start MongoDB

移除服务：
mongod.exe --remove

连接：
mongo.exe

mongod.cfg 文件（路径需要手动创建）：
storage:
dbPath: c:\data\db

18.3.2. 数据库操作

collection：数据库表/集合（table）
document：数据记录行/文档（row）
field ： 数据字段/域（column）

show dbs	显示所有数据库
db	显示当前使用的数据库名称
use jyl	创建/选择数据库 jyl
db.dropDatabase()	删除当前使用的数据库
show collections	显示当前使用的数据库中的所有集合名
db.createCollection("jylcl")	创建集合 jylcl
db.jylcl.drop()	删除集合 jylcl
db.jylcl.insert({'name':'jyl','age':1})	向集合 jylcl 插入数据，不存在则创建
db.jylcl.remove({'name':'jyl'})	按照条件删除集合中的文档
db.jylcl.update({'name':'jyl'},{$set:{'age':1}})	修改集合中的数据
db.jylcl.find()	查找集合中所有数据，find().pretty()以格式化的方式来显示所有文档
db.jylcl.find({'name':'jyl'}) db.jylcl.find({'name':'jyl','age':1})	按照条件查找集合中的文档

第十九章 REGULAR

1.1. 控制字符

字符	功能	示例
^	标记一行的开始	^cat 匹配所有以 cat 序列开头的行
$	标记一行的结尾	cat$ 匹配所有以 cat 序列结尾的行
|	将不同表达式进行或运算组合	AB|AC 匹配 AB 或者 AC
()	限制运算表达式运算范围，分	^(ab|cd)匹配以 ab 或 cd 开头

	隔表达式，可以限制|的作用范围		的行
[]	枚举取值内容，在字符组[]内外，元字符的含义会有不同。()是元字符，可以放在[]中，代表出现()字符		gr[eac]y 匹配出 grey、gray、grcy
	-	[]内字符的范围	[1-9] 匹配 1 到 9 的数字
			[1-9a-fC-E] 等价于 [123456789abcdefCDE]
	^	排除[]中的内容	gr[^ea] 匹配 gr 后面不是 e 或者 a 字符的内容【如果 gr 后是一行结束，则无法匹配，结束符不是字符】
.	代表单个任意字符		12.3 可以匹配出 12+3 12_3 12|3 等
?	表示?前面的一个字符不出现或出现 1 次		ab?c 匹配 abc 或者 ac
+	表示+前面的一个字符出现至少一次		3a+3 可以匹配 3aaa3 或者 3a3，不能匹配 33
*	表示*前面的一个字符不出现或出现任意次数		v3*v 可以匹配 v33v,v3v,v333v 等
{}	枚举前面的字符出现的次数		^a{1}$ 匹配 a 出现 1 次的行
			^a{1,3}$ 匹配 a 出现 1 次到 3 次的行

1.2. 转译字符

字符	功能	示例
\w	单个字母/数字/下划线(排除特殊字符，包括空格) \w+可以表示变量名	
\d	代表单个数字 (\d+)代表连续多个数字 \d{2,3}代表连续连个或三个数字	^\d$ 匹配出现单个数字的行
\D	非数字	
\< \>	分别代表单词起始和结束位置，可以匹配单词	\<is\> 匹配出 this is book, that is pen.中的单词 is
\c	与另一个字符一起使用时的控制字符。如\cA 是 control+A 的转义序列	
\n	换行	

\r	回车	
\s	单个空白字符，非空格或制表符	
\S	单个非空白字符	
\t	制表符	
\b	字边界	
\B	非字边界	

1.3. 特殊字符

字符	功能
$1 $99	匹配出了多个内容后，表示 1-99 个匹配的内容 __AAa_abc _+AAbef_def .*(AA.*)_.* 替换为 $1X 结果:AaX AAbefX
$&	匹配出的字符串
$`	匹配出的左侧的文本
$’	匹配出的右侧的文本
$$	$符号

第二十章 WINDOWS

20.1. Function

20.1.1. 登录不需要密码

netplwiz

20.1.2. 查看端口占用

netstat -aon|findstr "8080"

20.1.3. Win7 wifi

//创建和打开，ssid 为 wifi 名 key 是密码，至少 8 位
netsh wlan set hostednetwork mode=allow ssid=nnn key=1234abcd
netsh wlan start hostednetwork

//关闭
netsh wlan stop hostednetwork

网络和共享中心-》更改适配器设置
创建后多了一个适配器，将连接外网的适配器属性-共享，勾选允许其他计算机连接，下面选择多出的那个 wifi 适配器

无线网密码正确，处于分配 ip 状态，无法链接。可能是无线网络中 ip 地址重复，修改 ip 或者自动分配 ip 即可

20.1.4. IIS

----------安装配置----------
1.在添加和删除程序中，进入“打开或关闭 Windows 功能”(xp 下是打开组件)
2.将“Internet 信息服务”文件夹下的所有内容勾选。
最后面的两项：
万维网服务-》应用程序开发功能下，可以只勾选 ASP 和 ISAPI 扩展
运行状况和诊断下，可以只使用默认的 HTTP 日志和请求监视器
3.安装完成后，进入控制面板中的“管理工具”（将查看方式改为大图标，不要用类别）
点击“Internet 信息服务(IIS)管理器”，可进入管理页面（可以发送到桌面快捷方式）

----------IIS 设置----------
【====可以新建网站，在页面中一次性配置好网站名称，物理路径，端口号，IP 地址(相当于 2,3 两步)===】
1.左侧展开，找到网站文件夹，点击进入“Default Web Site”，进入页面
2.点击右侧的“高级设置”，在其中配置“物理路径”，该路径可以指定本地的一个文件夹。
3.继续在“Default Web Site”页面，点击右侧“绑定...”,可以在其中设置端口号和 IP 地址。
[4].在“Default Web Site”页面中的“目录浏览”中，点击右侧操作内的启用(此步骤相当于在文件夹内生成了一个 web.config 文件)

在“Default Web Site”页面右侧，“管理网站”中可以选择启动，重启，停止。

启动后，在浏览网站中，点击浏览***，即可浏览 ftp 网站，在其中看到指定的文件夹内容。

----------解决无法启动
左侧，最上层文件夹（比如 WIN7-20141205VV），右键-》启动

----------使用----------
在物理路径中指定的是一个文件夹，该文件夹下存储了和网站相关的所有文件（该文件夹下必须包括配置文件 web.config）

测试时可能会有缓存，显示之前内容，需要点 ctrl+F5 刷新

“默认文档”中可以设置默认要显示的内容，比如一个 html 网页，如果没有默认要显示的内容，或者没有找到要显示的内容，就会在页面中显示文件夹中所有内容。
“请求筛选”中可以设置不能被访问的文件的格式，如果文件夹中包含某种不能被访问的文件，访问就会出错

20.1.5. 防火墙

可以禁止或者允许某个程序使用网络（对应于程序所处的绝对路径，如果某个程序在不同的位置放置且运行过，则会为每个位置的程序都进行管理）

允许程序或功能通过 windows 防火墙：该功能内可以手动设置程序的联网权限。
打开或关闭 windows 防火墙：可以将防火墙关闭

无法打开防火墙解决方案:
控制面板-》管理工具-》服务-》Windows firewall-》右键属性，启动类型选择自动，之后进行开启

20.1.6. 关闭共享文件密码

网络和共享中心-》更改高级共享设置-》公用-》密码保护的共享 进行关闭

20.1.7. 注册表

注册表编辑器位置在 C:\Windows\regedit.exe

==========HKEY_CLASSES_ROOT 种类根键

包含了所有已装载的应用程序、OLE 或 DDE 信息，以及所有文件类型信息。每一个用圆点开始的子键表示一种文件类型。

==========*****注册右键菜单
*/shell/　　对所有文件起作用

===新建项 prjName
新建字符串值：icon　　...\prjName.exe,0
新建项:command　　...\prjName.exe %1

============HKEY_USERS　当前用户键
记录了有关登记计算机网络的特定用户的设置和配置信息，子键：

AppEvent：与 Windows 中特定事件相关连的声音及声音文件的路径。

Control Panel：包含了一些存储在 WIN.INI 及 SYSTEM.INI 文件中的数据，并包含了控制面板中的项目。

Install_Location_MRU：记录了装载应用程序的驱动器。

Keyboard Layout：识别普遍有效的键盘配置。

Network：描述固定网与临时网的连接。

RemoteAccess：描述了用户拨号连接的详细信息。

Software：记录了系统程序和用户应用程序的设置。

============HKEY_LOCAL_MACHINE　定位机器键
存储了 Windows 开始运行的全部信息。即插即用设备信息、设备驱动器信息等都通过应用程序存储在此键。子键有：

Config：记录了计算机的所有可能配置。

Driver：记录了辅助驱动器的信息。

Enum：记录了多种外设的硬件标识（ID）、生产厂家、驱动器字母等。

Hardware：列出了可用的串行口，描述了系统 CPU、数字协处理器等信息。

Network：描述了当前用户使用的网络及登录用户名。

Security：标识网络安全系统的提供者。

Software：微软公司的所有应用程序信息都存在该子键中，包括它们的配置、启动、默认数据。

System：记录了第一次启动 Windows 时的大部分部分信息。

==========HKEY_USER　私人用户键
描述了所有同当前计算机联网的用户简表。如果您独自使用该计算机，则仅 .Default 子键中列出了有关用户信息。该子键包括了控制面板的设置。

==========HKEY_CURRENT_CONFIG　当前配置键
该键包括字体、打印机和当前系统的有关信息。

20.1.8. 控件注册

注册：
regsvr32 MSCOMM32.OCX　//当前目录下存在控件

删除：
regsvr32　/u MSCOMM32.OCX

64 位机器注册：
C:\Windows\SysWOW64\regsvr32　MSCOMM32.OCX

20.1.9. 局域网远程桌面

被远程协助的计算机:
一，在“我的电脑”上右键“属性”，找到“远程设置”，在“允许远程协助连接到这台计算机”上打钩。
二，然后在“远程桌面”的三个选项中选择“允许任意版本的运行远程桌面的计算机连接到这台机器（较不安全）”。
三，点击右下角“选择用户”，出来的对话框里面点击“添加”，然后“选择用户”，在“输

入要选择的用户名”中输入你计算机上的一个用户名。然后点击“检查名称”。

进行远程协助的计算机：
打开远程协助：运行—输入 mstsc，然后再弹出的对话框里“计算机名”输入另外一台计算机的 IP 地址。

启用防火墙例外
运行---gpedit.msc--管理模板---组策略，进入计算机配置-管理模版-网络-网络连接-windows 防火墙-标准配置文件-windows 防火墙：允许入站远程桌面例外，双击，默认为未配置，选择已启用

20.1.10. 环境变量

我的电脑->属性->高级->环境变量
中可以设置环境变量

环境变量是一组系统或用户给应用程序设置的全局变量，通常用于保存各个应用程序都会用到的系统配置信息。应用程序可能会根据不同的环境变量配置产生不同的运行结果。操作系统启动后，会专门分配一块内存用于存储环境变量。类似于一种在整个操作系统范围内的全局变量。
在配置完环境变量后，需要使用环境变量的程序必须重新打开才可以使用环境变量。

在 cmd 或 ie 等程序中使用环境变量：%环境变量名%

path 环境变量可以作为可执行文件的路径（不包括文件名）
在寻找可执行文件的路径时，系统遵循以下顺序：
父进程所在目录，进程当前目录，系统目录，windows 安装目录，path 变量所指向的目录。

20.2. Command

1.1.1. CMD

命令	使用	说明
help	help **	显示所有命令的帮助文档
	命令 /?	显示命令的帮助文档
type	type 1.txt	将文本文件内容显示出来
echo	echo ***	显示内容***

	echo off	在此语句后所有运行的命令都不显示命令行本身
	echo *** >> 1.txt	将内容写入 txt
dir	/w	宽屏显示，一排显示 5 个文件名，而不会显示修改时间，文件大小等信息
	/p	分页显示,在当前屏最后有一个“press any key to continue . . .”提示，表示按任意键继续。
	/a	显示具有特殊属性的文件
	/s	显示当前目录及其子目录下所有的文件
cd	cd 路径	cd 命令只能进入当前盘符中的目录，其中“cd\”为返回到根目录,“cd..”为返回到上一层目录
md/mkdir	md abc\def	在当前目录下建立 abc 并在下面建立 def
rd	rd temp	表示删除当前路径下的 temp 目录,此命令只能删除空目录
	rd temp /s /q	/s 可以删除非空目录，/q 代表不需要选择确定
del	del *.dat	删除当前目录下所有扩展名为.dat 的文件
	del *.dat /s /f /q	可以删除包括子目录下的文件以及只读文件
copy	copy c:*.com d:\"	表示将 c 盘根目录下所有扩展名为 com 的文件拷贝到 d 盘根目录中
	copy *.txt ret.txt	将当前目录下所有 txt 内容放到 ret.txt 中
xcopy	xcopy /s /q /y /d c:\abc d:	执行此命令后，将把 c:\abc 目录及其目录中的文件全部拷贝到 d 盘根目录下。/s 对一个目录下的所有子目录进行拷贝。/y 覆盖不进行提示选择。/q 文件覆盖时不提示。/d 只复制比当前文件新的（data）
ren	ren *.txt *.csv	
format	format D: /s /q	此命令将格式化 D 盘，/q 表示进行快速格式化，/s 表示

		完成格式化后将系统引导文件拷贝到该磁盘，这样软件就可以作为 dos 系统启动盘了。格式化过程中，屏幕上会显示已经完成的百分比。格式化完成后，会提示为磁盘起 一个名字，最后还会报告磁盘的总空间和可利用空间等。
&	echo abc & echo def	让两条命令可以写一行

1.1.2. BAT

命令	使用	说明
call	call "***.bat"	执行某个 bat 文件
%[1-9]	分别表示启动 bat 时传入的第一个到第九个参数	
@	用于一行命令，不显示命令行本身	
pause	暂停批处理的执行并在屏幕上显示 Press any key to continue...的提示，等待用户按任意键后继续	
::或 rem	注释行	
if	if [not] "参数" == "字符串" 待执行的命令	
	if [not] exist [路径\]文件名 待执行的命令	
	if "abc"=="abcf" (echo "yes") else (echo "no")	
	if errorlevel <数字> 待执行的命令	
for	for %%v in (*.bat *.txt) do type %%v	将当前目录下所有 bat 和 txt 文件显示出来，%%v 代表执行过程中的变量
	for /r F:\xx %%v in (*.txt) do (type %%v)	遍历目录及子目录
	for /d %%v in (F:*.*) do echo %%v	只处理文件夹
	for /L %%v in (1,2,10) do echo %%v	1~10 步长 2
	for /f %%v in (a.txt) do echo %%v	逐行读取
	for /f "tokens=2 delims=," %%v in (a.txt) do echo %%iv	逐行读取,用,分割,取第二列 delims 默认为空格和 tab
	for /f "tokens=2,4 delims=," %%v in (a.txt) do echo %%v %%v	获取 2 和 4 列 必须使用%%i 和%%j 这种字母连续

	for /f "tokens=2-4 delims=," %%i in (a.txt) do echo %%i %%j %%k	获取 2 到 4 列
goto	goto mark :mark	跳转到 mark 处
set	设置环境变量，当前窗口有效	
	set /p val=****：	屏幕显示 ****：后等待输入，将输入字符串赋值给 val.
	"%val%"=="a" ,使用""将字符串包含起来进行比较	

```
echo off
cls

:Main
echo ************************************
echo *******        主菜单                *****
echo *******        1    fun1            *****
echo *******        2    fun2             *****
echo *******        X    exit             *****
echo ************************************
set/p    cc=     输入命令
if /i "%cc%"=="1"      goto FUN1
if       "%cc%"=="2"      goto FUN2
if       "%cc%"=="X"      goto EX
goto Main

:EX
exit

:FUN1
echo fun11111
goto Main

:FUN2
echo fun2222
goto Main
```

20.3. 注册表

用户启动项
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

系统启动项
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

操作系统启动项
HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run

20.4. VSCODE

shift+alt+左键　竖排文本选择
插件安装路径　　C:\Users\用户名\.vscode
ctrl +　放大
ctrl -　缩小

20.5. VS

20.5.1. 环境变量使用

存在环境变量 CODE D:\code

在工程属性配置时使用
$(CODE)
即可指代 D:\code

$(ConfigurationName) 当前项目配置的名称（例如，“Debug|Any CPU”）。

$(OutDir) 输出文件目录的路径，相对于项目目录。这解析为“输出目录”属性的值。它包括尾部的反斜杠“\”。

$(DevEnvDir) Visual Studio 2005 的安装目录（定义为驱动器 + 路径）；包括尾部的反斜杠“\”。

$(PlatformName) 当前目标平台的名称。例如“AnyCPU”。

$(ProjectDir) 项目的目录（定义为驱动器 + 路径）；包括尾部的反斜杠“\”。

$(ProjectPath) 项目的绝对路径名（定义为驱动器 + 路径 + 基本名称 + 文件扩展名）。

$(ProjectName) 项目的基本名称。

$(ProjectFileName) 项目的文件名（定义为基本名称 + 文件扩展名）。

$(ProjectExt) 项目的文件扩展名。它在文件扩展名的前面包括“.”。

$(SolutionDir) 解决方案的目录（定义为驱动器 + 路径）；包括尾部的反斜杠“\”。

$(SolutionPath) 解决方案的绝对路径名（定义为驱动器 + 路径 + 基本名称 + 文件扩展名）。

$(SolutionName) 解决方案的基本名称。

$(SolutionFileName) 解决方案的文件名（定义为基本名称 + 文件扩展名）。

$(SolutionExt) 解决方案的文件扩展名。它在文件扩展名的前面包括“.”。

$(TargetDir) 生成的主输出文件的目录（定义为驱动器 + 路径）。它包括尾部的反斜杠“\”。

$(TargetPath) 生成的主输出文件的绝对路径名（定义为驱动器 + 路径 + 基本名称 + 文件扩展名）。

$(TargetName) 生成的主输出文件的基本名称。

$(TargetFileName) 生成的主输出文件的文件名（定义为基本名称 + 文件扩展名）。

$(TargetExt) 生成的主输出文件的文件扩展名。它在文件扩展名的前面包括“.”。

20.5.2. cmd 编译

```
call    "******\vcvarsall.bat"

devenv    /out build_result.txt    /rebuild    "release|win32"      ******.sln
```

20.5.3. 程序管理员权限

工程属性： 链接器-〉清单文件->UAC 执行级别-〉requireAdministrator

20.5.4. 生成事件

文件名如果包含空格，需要用引号

=====预先生成事件

C++:
工程属性->生成事件->预生成事件
xcopy /s /q /y /d $(CODE)\发布 $(OutDir)

C#(不需要写生成的目标文件夹):
工程属性->生成事件->预先生成事件命令
xcopy /s /q /y /d $(CODE)\发布

可以对文件属性选择 复制到输出目录

=====后期生成事件
copy "$(TargetPath)" $(CODE)\发布\xx.exe

或者[C#无效] copy $(OutDir)$(TargetName)$(TargetExt) $(CODE)\发布\xx.exe
路径中有空格时需要用""

20.5.5. C#代码提示

重载函数:

类内写入 override 后空格，会自动提示可以重载的函数

事件处理:

this.button1.Click +=然后使用 tab 键，直接生成对应事件的处理函数

20.5.6. 调试

========变量显示
类型 值
int 10
struct {成员 1=** 成员 2=**}
map [成员数量]（元素 1，元素 2）

解决方案管理器：ctrl+alt+L
工具箱：ctrl+alt+X
类视图：ctrl+shift+C
资源视图：ctrl+shift+E
查找所有结果：ctrl + shift + F

F7 编译工程
Ctrl+F5 运行程序
Ctrl+F7 编译当前文件

F9 设置断点
F5 以调试状态运行/执行到下一断点
F11 跟踪时进入函数内
Shift+F11 跳到上一层调用栈
F10 执行一句
Ctrl+F10 执行到光标所在行
Shift+F9 QuickWatch,并显示关标所在处的变量值
Alt+3 Watch 查看窗口
Alt+4 Variables 监视变量(常用)
Alt+5 显示寄存器
Alt+6 显示内存(常用)
Alt+7 显示堆栈情况
Alt+8 显示汇编码

20.5.7. 快捷键

Alt+F7 工程设置对话框

Ctrl+k f 格式化代代码
Ctrl+c 复制一行
Ctrl+L 剪切一行
选择的粘滞键 按住 Shift+方向
Alt+F8 选中的代码书写格式自动排列
Ctrl+Z/Ctrl+Y Undo/Redo
Ctrl+K,C 注释选中的行
Ctrl+K,U 取消选中行的注释

Ctrl+U 字母转化为小写(有的 VC 没有设置)
Ctrl+Shift+U 字母转化为大写(有的 VC 没有设置)
Ctrl+F 查找

Ctrl+H 替换
Ctrl+G 跳到文件中第 n 行
Ctrl+} 匹配括号(),{}
Ctrl+Shift+G 光标在一个文件名上,直接跳到指定文件
Ctrl+J,K #ifdef...#endif 查找配对

20.5.8. 安装项目

解决方案属性，系统必备（从安装包路径下载）
解决方案属性页，TargetPlantform

文件系统编辑器，文件加入应用程序文件夹，程序文件夹，启动，桌面

自定义操作编辑器，添加程序，属性中加入参数

20.5.9. nuget

工具-》nuget 包管理器-》管理解决方案的 nuget 程序包
界面中程序源包选择设置，添加本地包位置
选择程序源包位置为添加的本地位置

20.5.10. 结构体对齐

c/c++ -> 代码生成 -> 结构体成员对齐

20.5.11. python

工具-》选项-》python-》调试-》使用旧版本调试程序
（或者安装升级 ptvsd 包）

工程中 python 环境，右键可以打开交互环境、安装 python 包

20.5.12. 问题

无法找到资源编译器 rcdll.dll	将 rcdll.dll 放到外层 x86、x64 文件夹
XXX 模块对于 SAFESEH 映像是不安全的	链接器->命令行->将 /SAFESEH:NO 键入附加选项框中

20.6. Vmware

安装时，需要关闭 vs

安装 VMware Tools（VM 菜单中），可以调节屏幕大小,设置分辨率即可改变大小

Ctrl+Alt　　退出鼠标
Ctrl+Alt+回车　　切换全屏

虚拟机 ip 地址一定要使用自动获取，主机上虚拟机的网络适配器 ip 地址也自动获取。
虚拟机只能用客户端程序，向主机发命令，不能做服务端。

主机可能需要先设置密码，在虚拟机中登录主机（需要输入密码）

//////主机共享路径
虚拟机关闭状态下
VM 虚拟机开启界面
Edit virtual machine settings
Options ->Shared folders
选择 always enabled
点击 ADD，选择主机需要共享的文件夹路径

打开虚拟机，安装 VMware Tools（VM 菜单中）
在计算机磁盘显示中，有可移动存储设备中，会出现 vmware tools，双击进行安装

右键菜单添加网络位置
浏览文件夹，在网络中添加 wmware-host

///虚拟机不能联网等问题
打开 windows 服务，开启 vm 相关服务

20.7. utf8

根据每个字节前面的数字决定后面使用几个字节表示一个 unicode 编码

0-0x7F　　0*******

0x80-0x07FF 110***** 10******
0x800-0xFFFF 1110**** 10****** 10******
0x10000-0x10FFFF 11110*** 10****** 10****** 10******

‘汉’的 unicode 编码
十六进制：6C49
二进制：01101100 01001001

二进制分割为： 0110 110001 001001
套入模板： 1110**** 10****** 10******
对应 utf 编码为 11100110 10110001 10001001

20.8. office

20.8.1. excel

==========条件筛选
选中某行或者其中某一列，ctrl+shift+L，可以下拉筛选当前列的内容

==========计算

在某单元格中输入 “=”点击选择其他单元格，会自动加上位置（如=A1），输入公式符号（如 = A1 +），再选择其他单元格（如 = A1 + A2）,按回车就可以计算出结果。
选中计算后的单元格，鼠标放在单元格右下角，变成“+”后，可以向下拉，这样整列都可以按照这个公式进行计算

20.8.2. word

=========多级标题设置
重新设置时，鼠标焦点要放在开头

点击工具栏中"多级列表"图标

选择"定义新的多级列表"

点击"更多"展开右边的选项

1.左上角"单击要修改的级别"
2.右边"将级别链接到样式",选择要修改的样式（如标题 1）
3."在库中显示的级别"，和 2 对应选择（如级别 1）

==========页码
设置前两页无页码，第三页开始有页码
1.插入页码（页面底端）
2.光标定位在第 2 页末，执行“页面布局/分隔符/下一页（分节符）”【此时页码排号分成两部分】
3.****点击第三页的页码，在上面出现的设计菜单中，将"链接到前一条页眉"去掉****
4.点击第一页或第二页页码，点击页码、删除页码（不能直接回退键取消页码）

=========合并文档
插入-》对象-》文件中的文字

20.9. git

========设置帐号和邮件
git config --global user.name "Your Name"
git config --global user.email "email@example.com"

========创建 repository
在需要创建的目录下执行初始化命令
git init

========提交
git add *　　添加当前所有文件，可以选择添加
git commit -m　..　　提交，-m 加入提交理由

=========状态
git status　　查看当前状态
git diff　　查看当前库中修改的内容

========记录

git log　　查看历史提交记录　--oneline 只显示简略信息
git reflog　　回退后，可以查看后面的版本号

==========回退
git reset　--hard　****

****可以是版本号，默认值是 head~1(上一次提交的版本，数字往前提几个版本)

========删除
git rm　a.txt

========分支
git　branch　　查看分支

git　branch　b1　　在当前分支状态创建分支
git　checkout b1　　切换到 b1 分支
git　checkout -b b1　　创建并切换到分支 b1
git　branch -m oldname newname　　重命名分支

git　merge b1　--squash　　当前分支合并 b1 分支　--squash 参数只进行合并，先不进行提交
git　cherry-pick　***　-n　　选择 reflog 中的提交号码进行合并，-n 可以只进行合并，先不提交

git　branch -d b1　　删除 b1 分支(如果 b1 分支没有被合并，则使用-D)

冲突
<<<<　HEAD　**** ======mergeTxt>>>>>branchName

HEAD 之后是开始冲突的内容
======mergeTxt>>>>>　mergeTxt 是合并过来的内容
branchName 是合并分支的名称

==========局域网
服务器：
git init --bare

[//192.168.111.2/aaa (aaa 为服务器 init --bare 的位置)]

客户端:
git config --global user.name "Your Name"
git config --global user.email "email@example.com"
git init

获取：
git pull //192.168.111.2/aaa

推送:
git add *
git push //192.168.111.2/aaa
git push --set-upstream //192.168.111.2/aaa master

tortoiseGit 设置
Remote: 名称任意
URL: //192.168.111.2/aaa (aaa 为服务器 init --bare 的位置)

==========github
git config --global user.name "jiyanglin"
git config --global user.email "760437565@qq.com"

ssh-keygen -t rsa -C "760437565@qq.com"
命令行需要输入验证码和确认
.pub 文件中包含 sshkey，github 中 SSH Keys 进行添加

使用名称 origin
git remote add origin git@github.com:JiYangLin/MySocket.git

下载
git clone https://github.com/JiYangLin/MySocket

提交到远程 master 分支，使用名称 origin

先本地提交
git push origin master

20.10. eclipse

==========界面布局
Open Perspective
java EE(default)　恢复默认布局

Window-》Show View-》Other　可以选择要显示的窗口

==========安装插件
help->Eclipse Marketplace

==========代码提示
window->preferences->java->editor->Content Assist->auto activation triggers for java
在其中可以把所有字母键入，则打所有字母都会有提示

==========编码格式
window->preferences->General->workspace　:utf-8

==========调试
Ctrl+Shift+B　在当前行设置断点或取消断点
F6　——单步调试
F8　——到下一个断点

F5　——进入函数内部

F7　——由函数内部返回到调用处
F3　——跳到声明或定义的地方

==========快捷功能 context 菜单
鼠标右键菜单
s-》r 产生成员变量的 get 和 set 方法

==========快捷操作
alt + shift + a　竖排文本选择模式

ctrl + shift + o 删除多余的 import 内容
ctrl + shift + f 自动调整格式
ctrl + 1 代码修复，可以导入包和添加异常处理
ctrl + d 删除行
ctrl + f11 快速运行项目
ctrl + M 放大工作区
ctrl + / 添加注释
alt + / 补全代码
alt + 上下 快速移动行
/** 回车 在函数上自动生成函数说明

第二十一章 LINUX

21.1. 系统

21.1.1. 命令

参数可以连续使用 ls －la
rmdir －help 显示帮助
|管道功能 ： ls|head ls 输出的内容为 head 的输入
bash 脚本(**.sh)可以包含一系列要执行的命令

命令	参数	说明
echo		再次显示 echo 后的内容到屏幕上 写文件：内容>>文件名 >加入内容 >>在原有内容上追加
ls	l 详细内容显示	(list)显示当前目录下所有内容
	a 显示全部内容，包括隐藏	
pwd		显示当前路径

clear		清屏
cd		change directory 绝对路径或者相对路径
mkdir	p (parents) 建立深层次目录	可以一次建立多个
rmdir		只能删除非空目录
rm	r 删除目录（可以非空）	删除文件
cp		copy 拷贝文件
mv		移动(剪切)
touch		创建新文件
cat		concatenate a file print it to the screen 打开并显示文件
head	n 根据 n 后的数字确定显示的行数	默认显示打开文件的前 10 行
tail		默认显示打开文件的后 10 行
find	size 指定查找的文件大小。数字加正负号则是文件大于或小于 nbyte。	默认当前目录查找
grep		基于内容查找

		要查找的内容紧跟命令后 grep a xx.txt 查找出带 a 的行显示出来
ln	s 软连接（原文件删除后，新文件消失）	ln 文件名 新文件名 默认硬连接（原文件删除后，不影响新文件）
chmod		修改文件权限 chmod u+x 文件名 u/g/o 用户类别 (+,-) r/w/x 权限 u/g/o/a 代表 user,group,other,all(不写前三个时) r/w/x 代表 read,write,execute chmod 755 文件名 r 4 w 2 x 1 r 100 w 010 x 001 755 三位数分别代表 u/g/o 的权限 由 r,w,x 代表的数位与组成

su		切换到 root 用户 带参数则切换到指定用户
useradd		创建用户
userdel	f 强制删除	删除用户
passwd		passwd username 创建/修改用户的密码
shutdown	r 重启 h now 立刻关机	
ps	aux linux 专用 ef linux,unix 通用	查看终端启动的进程的 id
time		time a.out 查看 a.out 各种运行时间
ifconfig		查看 ip

/ 根目录

..上级目录

～用户主目录

d 目录

-普通文件

|链接文件

21.1.2. 运行程序

./文件名　运行当前目录下文件

　　或　…/文件目录/文件名(其它位置时)

21.1.3. 脚本

***.sh
sh　***.sh

a=123
b=$a/456
得到 b 值为 123/456

21.1.4. 常用操作

右侧上键调出上一条命令

下键调出下一条命令

shift+pageup/down　翻页

ctrl+‘+’/‘-’　　调整字体大小,右侧小键盘的加减

tab 键可以自动补齐输入文件名

？,*,[]通配符　　对文件名进行书写时可用

　？代表一个字符，不可放末尾

*代表任意个字符，可放末尾

[]可写不同内容　exp:ab[3-7]

ctrl+c　终止进程

ctl + s 锁屏/ctl+q 解锁

21.1.5.　编译

c 语言用 gcc

c++用 g++

gcc　文件名　　预处理，编译，汇编，链接生成 a.out

　　通常所说的编译指前三步

-选项

-o　　新名称　　指定生成文件的名称

-c　　只编译不链接(.o)

-S 大写　　生成汇编文件（.s）

-E　　只进行预处理（.i）

-Wall　　显示更多警告信息

-V　　查看版本号

-g　　生成文件后，gdb　文件名　命令进入 gdb

将宏加上//,可用-D 定义宏:

//#define SIZE 8

gcc -DSIZE=5 a.c

#pragma 可额外增加一些功能

1)#pragma GCC dependency “fname”

“fname”文件如果比当前文件新，则产生警告

2)#pragma GCC poison ab

ab 不可用，如果使用则报错

3)#pragma pack(n)

指定对齐/补齐字节数(1,2)有意义

多源文件的编译

6)生成.o 文件

7)gcc *.o 链接所有的.o 文件

目录下提供相应的头文件，其中有对函数的声明，满足函数先声明，位置任意放

21.1.6. 库

静态库（.a）：使用是代码复制到目标文件，目标文件独立于库文件，运行速度稍快，目标

文件变大，代码修改不方便。

4）生成 t.o 文件

5）ar -r lib***.a t.o 创建名为***的静态库

调用库时：1）生成 test.c 的 test.o

2)gcc test.o libxx.a (当前目录时)

（常用）gcc test.o -l*** -l 库所在路径

3）配置环境变量 LIBRARY_PATH,把库文件路径加入，然后可用

gcc test.o -l***

共享库（.so）：使用是把函数地址放入目标文件，运行速度稍慢，目标文件小，修改方便

目标文件和共享库需要同时存在

3)生成 t.o 文件 （可不写-fpic）

4)gcc -shared t.o -o lib***.so

调用库时与静态库方式相同

ldd 名称 可查看包含的动态库，可检查哪些库文件未加入到运行环境变量中

21.1.7. gdb

L 查看源码

b 行数

或 b main 设置运行断点

r 运行

n	运行下一行
s	进入函数内运行
q	退出

21.1.8. Vi

操作	说明
vi 文件名	进入/建立文件
i	进入写入模式
大写 O	在光标上方插入一行，并进入写模式
小写 o	在光标下方插入一行，并进入写模式
esc	退出写入模式
:q	退出 ：q! 强行退出
:x **:wq** **shift+zz**	退出并保存
dd	剪切当前光标所在的行 ndd 剪切当前光标所在的行及之下的共 n 行

p	把剪切板中的内容粘贴到当前光标行
u	返回上一步
ctrl+r	恢复上一步被撤销的操做
yy	复制当前光标所在行 nyy 复制 n 行
v	进入视图模式，从当前光标开始，有高亮条，可选中内容， 使用 d 剪切，y 复制，p 粘贴

：36，51 w a.txt

把当前文件 36 到 51 行写到 a.txt 中

：r a.txt 把 a.txt 写到当前文件

：n 跳到第 n 行

/内容 跳到内容位置

21.1.9. 环境变量

vi .bashrc

在末尾加入一行 export PATH=$PATH:.

source .bashrc 当前窗口生效，或者重启生效

加入后，运行文件时不需要使用./文件名，可直接使用文件名运行。

21.1.10. 后台程序

ctl +z　退出程序界面，程序挂起

jobs　查看当前有多少在后台运行的命令

bg　将一个在后台暂停的命令，变成继续执行，bg %jobnumber 将选中的命令调出

fg　将后台中的命令调至前台继续运行，fg %jobnumber 将选中的命令调出

%jobnumber 是通过 jobs 命令查到的后台正在执行的命令的序号

21.1.11. ftp

gftp 进入 ftp 界面

sudo　dpkg　-i　安装包名　安装软件

ftp　********　登录 ftp 中：

！clear　清屏

bye　退出

put/mput　上传

get　文件名　（下载到当前目录）

mget　a*　下载名为 a*的所有文件到当前目录

21.1.12. 网络配置

ifconfig -a　查看所有网卡
route　-n　查看网络路由

dhclient -r　释放 ip 地址
dhclient　　获取 ip 地址

/etc/network/interfaces　网络配置文件

21.1.13. ubuntu

21.1.13.1.双系统安装

1.将.iso 中的 initrd.lz、vmlinuz.efi 解压出来与 iso 一同放在 C 盘 根目录。

2.添加新条目-》NeoGrub -》安装，编写
NeoSmart NeoGrub Bootloader Configuration File
#
This is the NeoGrub configuration file, and should be located at C:\NST\menu.lst
Please see the EasyBCD Documentation for information on how to create/modify entries:
http://neosmart.net/wiki/display/EBCD/

title Install Ubuntu
root (hd0,0)
kernel (hd0,0)/vmlinuz.efi boot=casper iso-scan/filename=/ios 名 ro quiet splash locale=zh_CN.UTF-8
initrd (hd0,0)/initrd.lz

5)在 EasyBCD 编辑引导菜单可看到 NeoGrub 引导加载器，记得勾选等待用户选择，保存设置

6)安装界面系统中用命令卸载盘符，sudo umount -l /isodevice

7)分配空间

交换空间　：逻辑分区　（RAM 大小）
/　　：逻辑分区
/home　　：逻辑分区
/boot　　：　主分区

6.选择/boot 所在分区进行安装

21.1.13.2.软件包管理

Ubuntu Software Center

21.1.13.3.更新软件

sudo apt-get update

21.1.13.4.浏览器

sudo apt-get update

sudo apt-get install google-chrome-stable

/usr/bin/google-chrome-stable

21.1.13.5.输入法

sudo apt install libopencc1 fcitx-libs fcitx-libs-qt fonts-droid-fallback
wget "http://pinyin.sogou.com/linux/download.php?f=linux&bit=64" -O "sougou_64.deb"
sudo dpkg -i sougou_64.deb

重启后
输入法设置中点击+，在弹出对话框中将 only show curent language 勾选掉，然后搜索到 sougou，然后进行添加
(系统设置>语言支持>键盘输入方式系统然后选择 fcitx 项)

安装不完整时移除
sudo apt remove sogoupinyin

21.1.13.6.快捷方式/搜索

dash 按钮（类似 windows 按钮）

21.1.13.7.菜单栏隐藏

System Settings->Appearance->Behavior->Auto-hide the Launcher

21.1.13.8.桌面工具

sudo apt-get install cairo-dock
cairo-dock 启动
cairo-dock -m 设置
sudo apt-get --purge remove cairo-dock 卸载

21.1.13.9.qtcreator

ctrl+i 自动排版
ctrl+b 编译
ctrl+r 运行
ctrl+空格 代码提示

==========cannot find lGL
locate libGL 或 find /usr -name libGL* 搜索。
搜索结果中发现/usr/lib/i386-linux-gnu/mesa/libGL.so.1 文件（这个文件也可能在另一个目录中）
使用 ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/libGL.so 命令给已存在的库文件创建一个链接到/usr/lib 目录。

21.1.13.10. LibreOffice

F5 文档结构

21.1.14. 设置

21.1.14.1.系统日志

tail -f /var/log/syslog

21.1.14.2.挂接 u 盘

vfat 代表 fat32
fdisk -l
mount -t vfat /dev/sdc1 /home/sign/usb
umount /media/usb

21.1.14.3.环境变量

$PATH="$PATH":/NEW_PATH

修改 /etc/profile

21.1.14.4.网络配置

/etc/network/interfaces

21.1.14.4.1. 静态 ip

interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 192.168.0.101
netmask 255.255.255.0

21.1.14.4.2. 动态 ip

dhclient 重启网卡
interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
pre-up ifconfig eth0 hw ether 40:8d:5c:e7:24:57
gateway 172.16.80.254
netmask 255.255.255.0
broadcast 172.16.80.255
dns-nameservers 114.114.114.114

21.2. Function

21.2.1. XSI IPC

XSI IPC 包括共享内存，消息队列，信号量集，隶属于一个规范，有着共同的特征
每个 XSI IPC 结构都是在内核中存储和维护的，外部到内核要用 key，内核中使用 id 标识
key_t key 有三种方式得到：
1）使用宏 IPC_PRIVATE 做 key，无法实现进程间通信（私有），极少使用
2）把所有的 key 定义在一个头文件中，用宏定义
3）使用 ftok()函数生成 key，参数：真实存在的路径和项目编号（0 到 255）
id 的生成：ipc 结构在内核中都用 id 做唯一标识，创建、获取 id 都有对应的函数
key_t key=ftok("/home/jyl",222);//生成一个 key

调用**get()时，都有一个 flags，创建时值为：
权限 | IPC_CREAT 或 IPC_EXCL
IPC_CREAT - 不存在则创建，存在则获取
IPC_EXCL - 如果存在则创建失败

IPC 结构都有一个特殊的操作函数，提供查询，修改和删除功能
函数名：**ctl() 如：shmct(),msgctl()
参数 cmd 提供功能：
IPC_STAT 查属性、状态
IPC_SET 修改权限
IPC_RMID 删除 ipc 结构，按 id 删除

21.2.1.1. 共享内存

ipcs -m 可以查看系统中共享内存的使用情况

每个进程内存独立，无法直接互访，共享内存就是内核管理一段内存（物理内存），这段内存允许每个进程进行映射，是速度最快的 IPC，效率高。
如果多个进程写数据，会产生覆盖

1）系统创建、获取共享内存（拿到物理内存）
shmget() 创建、获取共享内存，返回 id
2）挂接共享内存（映射）
shmat()

3）使用共享内存
4）脱接共享内存（解除映射）
    shmdt()
5）如果共享内存不再被使用，删除
    shmctl()

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
const int BUFSIZE = 100;
const key_t    key = (key_t)12345678;

======写入
 int    shmid=shmget(key,BUFSIZE,IPC_CREAT);
 char* pDat=(char*)shmat(shmid,0,0);

 char buf[BUFSIZE] = {0};
 while(1)
 {
            cin>>buf;
            memcpy(pDat,buf,BUFSIZE);
 }

 // 分离
shmdt(pDat);
// 删除共享内存
shmctl(shmid, IPC_RMID, 0);


======读取
int     shmid=shmget(key,0,0);
char* pDat=(char*)shmat(shmid,0,0);
while(1)
{
            sleep(1);
            cout<<pDat<<endl;
}
shmdt(pDat);



======状态
struct shmid_ds      ds;
```

shmctl(shmid,IPC_STAT,&ds); 取 shm 状态
挂接数 nattch %d ds.shm_nattch
ds.shm_perm.mode=0640; 权限可以修改
shmctl(shmid,IPC_SET,&ds);

21.2.1.2. 消息队列

队列中存放各种信息，每个进程可以把数据封存在消息中，再放入队列，每个进程都可以拿到消息队列，再从中取出、放入消息
1）创建、获取消息队列 msgget
2）发送、接收消息 msgsnd、msgrcv
3）如果所有进程不再被使用，删除 msgctl()

<sys/msg.h>

必须用结构体

```
struct DDD
{
    int   x;
};
const int BUFSIZE = sizeof(DDD);
const key_t   key = (key_t)6666;
```

======写入
程序结束时必须要删除消息队列，否则会造成读取错误

```
int msgid=msgget(key,IPC_EXCL);
if(msgid > 0)    msgctl(msgid,IPC_RMID,0); //删除消息队列

msgid=msgget(key,IPC_CREAT|0666);
while(1)
{
        int xx;
        cin>>xx;
        DDD dd;
        dd.x = xx;
        msgsnd(msgid,&dd,BUFSIZE,IPC_NOWAIT);
}
msgctl(msgid,IPC_RMID,0); //删除消息队列
```

```
======读取
int    msgid=msgget(key,IPC_EXCL);
 if(msgid < 0) return 0;

 while(1)
  {
          sleep(1);

          DDD dd;
          msgrcv(msgid,&dd,BUFSIZE,0,0);
          cout<<dd.x<<endl;
 }
```

21.2.1.3. 信号量集

由信号量组成数组，信号量其实就是一个计数器，用于控制同时访问共享资源的进程、线程的总数（IPC 中的信号只控制进程）
计数到 0 阻塞后续进程
信号量使用方法：
1）先赋初值，允许并行的进程最大数量
2）如果有进程访问，计数-1，到 0 就阻塞访问进程
3）如果有进程访问结束，计数加 1
信号量集就是一个计数器数组，只能控制进程数量而不能互发数据

```
<sys/sem.h>

int ArrayLen = 10;
//创建、获取信号量集的 id        0666 代表权限
int semid = semget((key_t) 1234567, ArrayLen, 0666 | IPC_CREAT);
//semctl(semid,0,IPC_RMID);   删除信号量集合？

int    res = semctl(semid, 0, SETVAL, 2);//下标为 0 的位置设置 2 个信号
res = semctl(semid, 1, SETVAL, 3);//下标为 1 的位置设置 3 个信号

for(int i=0;i<6;i++)
 {
                pid_t    pid=fork();
                 if(pid!=0) continue;

                 struct sembuf op;
```

```
        op.sem_num = 0;//对应下标为 0 的信号
        op.sem_op = -1;//对应信号量-1
        op.sem_flg = SEM_UNDO;
        if (semop(semid, &op, 1) == -1)
        {//进行操作信号量的个数，即 op 的个数，需大于或等于 1。常设置 1，只完成对一个信号量的操作
            cout<<"run failed"<<endl;
            exit(0);
        }


        cout<<i<<endl;
        string str;
        cin>>str;

        op.sem_op=1;//对应信号量+1
        semop(semid, &op, 1);
        exit(0);
}
```

21.2.2. 管道

管道分为有名管道和无名管道，有名管道自由建立管道文件（*.pipe），用于进程间通信。无名管道由内核建立管道文件，用于 fork 创建的父子进程之间的通信。管道数据是先进先出，只是数据中转站，不存储数据（只写不读会阻塞），和文件的操作相同。一个程序写，另一个程序读(阻塞)。

21.2.2.1. 有名管道

```
#include <fcntl.h>
#include <sys/stat.h>
const int BUFFER_SIZE = 100;
const char *pipName = "/home/jyl/NamedPip";

======写入
if (access(pipName, F_OK) == -1)
    {
        int res = mkfifo(pipName, 0777);
        if (res != 0) return 0;
    }


    int fd = open(pipName, O_WRONLY);
```

```
    char buf[BUFFER_SIZE] = {0};
    while(1)
    {
        cin>>buf;
        int res = write(fd, buf, BUFFER_SIZE);
        if(res == -1) break;
    }

close(fd);


======读取
const int BUFFER_SIZE = 100;
const char *pipName = "/home/jyl/NamedPip";

int fd = open(pipName, O_RDONLY);
char buf[BUFFER_SIZE] = {0};
while(1)
 {
        int res = read(fd, buf, BUFFER_SIZE);//阻塞
        if(res <= 0) break;
        cout<<buf<<endl;
  }

 close(fd);
```

21.2.2.2. 无名管道

```
const int bufLen = 100;
int    fd[2]={};
pipe(fd);

pid_t id    = fork();
if(id == 0)
{
    close(fd[1]);//只负责读，因此将写描述关闭(节约资源)
    char val[bufLen] = {0};
    while(1)
    {
        read(fd[0],val,bufLen);
        cout<<val<<endl;
    }
    exit(0);
```

```
}


 close(fd[0]);//只负责写，因此将读描述关闭(节约资源)
 while(1)
 {
        char str[bufLen] = {0};
        cin>>str;
        write(fd[1],str,bufLen);
 }
```

21.2.3. 信号

21.2.3.1. 定义

信号是程序中段的一种方式，属于软件中断。
中断就是终止当前的代码，转而执行其他代码，中断有软件中断和硬件中断。

信号的本质是一个非负整数
unix 常用信号 1 到 48，linux 常用信号 1 到 64，每个信号都有宏名，以 SIG 开头。
信号处理采用异步处理。

1 到 31 都是不可靠信号，这种信号不支持排队，有可能丢失，非实时信号。（同一信号时）
34 到 64 都是可靠信号，支持排队，不会丢失，是实时信号。

#include <signal.h>

21.2.3.2. 信号处理

1）默认处理　　80%的默认处理都是中断进程
2）忽略信号　　可以忽略信号，不做处理
3）自定义处理
信号 9 不可以被忽略，也不能被自定义
普通用户只能给自己的进程发信号，root 可以给所有用户发信号

fork 创建的子进程完全沿袭父进程对信号的处理。
vfork+exec 创建的子进程可以沿袭 SIG_IGN 和 SIG_DFL，不能沿袭自定义方式（改为默认）

signal(信号，信号处理函数指针)
函数指针：1）SIG_IGN　忽略

2）SIG_DFL　默认
3）自定义处理函数
错误返回 SIG_ERR

```
#define MSG_MY      _NSIG-1
======方式 1
void sigProc(int sig)
{
       cout<<"recv "<<sig<<endl;
}
signal(MSG_MY,sigProc);
======方式 2
void sigMsgProc(int    sig,    siginfo_t    *info,    void    *p)
{
       cout<<"procees["<<info->si_pid<<"]send ["<<sig<<"]:"<<endl;
       int sigValInt = info->si_int;
     cout<<sigValInt<<endl;
     //指针数据传递只适用当前程序
       //char *sigStr = (char*)info->si_ptr;
       //cout<<sigStr<<endl;
}

   struct    sigaction      action={};
     action.sa_flags=SA_SIGINFO;
     action.sa_sigaction=sigMsgProc;
     sigaction(MSG_MY,&action,NULL);
```

21.2.3.3. 信号发送

类别	信号
键盘发送（少部分信号）	ctrl+C　2　SIGINT
	ctrl+\　3　SIGQUIT　结束进程
出错（少部分信号）	段错误　11　SIGSEGV
	总线错误　7　SIGBUS
	整数除 0　8　SIGFPE
命令 kill（全部信号）	kill　-信号　进程 id
信号发送函数	int　raise(int sig) 只能给当前进程发信号，成功返回 0，失败返回

	非 0
	int kill（pid_t pid， int sig） pid>0 给 pid 对应的进程发信号 pid==0 发送给同组所有进程 pid==-1 发给所有有权限的进程 pid<-1 发给进程组（id 等于 pid 绝对值）中的所有进程
	<unistd.h> 闹钟信号 unsigned int alarm(unsigned int seconds); seconds 秒数后发送 SIGALRM 信号到当前进程 alarm（5） 5 秒后结束进程 alarm（0） 取消闹钟 每次使用 alarm 时，之前使用的 alarm 失效 调用 alarm 时，以前设置的闹钟还没有执行，会返回之前的闹钟剩余秒数，如果以前设置闹钟执行了或者没有设置闹钟，则返回 0

信号 0 用来测试是否有权限发信号，信号最大数值为_NSIG-1

======方式 1
//给当前进程发 MSG_MY 信号
raise(MSG_MY);
//给 pid 进程发信号
kill(pid,MSG_MY);
======方式 2
union sigval v;
v.sival_int=10;
v.sival_ptr = str;
sigqueue(pid,MSG_MY,v);

21.2.3.4. 信号集

用一个二进制位代表一个信号，用一个整数（sigset_t 类型）代表信号集
sigset_t set;
int sigemptyset(sigset_t *set); 将二进制位全部设置成 0
int sigfillset(sigset_t *set); 将二进制全设成 1
int sigaddset(sigset_t *set, int signum);
把信号 signum 加入信号集
int sigdelset(sigset_t *set, int signum);
把 signum 从信号集中删除

int sigismember(const sigset_t *set, int signum);

信号集中存储所有信号，每个信号都有相应二进制位置，且不重复
将信号加入，则将相应的位设成 1，每个信号的存储位置在信号集中都有预设

信号屏蔽：延后信号的处理，等屏蔽结束后执行发过的信号，可靠信号支持排队，会执行发过的次数，不可靠信号无论发几次，只执行一次
sigprocmask(int how, sigset_t *new, sigset_t *old)
how 是屏蔽方式：
SIG_BLOCK 在原有基础上加新信号进行屏蔽
SIG_UNBLOCK 减去原有信号屏蔽中的某些信号
SIG_SETMASK 重新设置 （常用）
new 是新的屏蔽信号集，old 可以把之前的屏蔽信号集保存起来，如果 old 参数为空就是不保存
可以用 old 作为新函数中的 new 来取消屏蔽

sigpending(sigset_t *set)
取信号屏蔽期间来过的信号放入信号集

21.2.4. 定时器

<sym/time.h>
linux 为每个进程维护了三个计时器
真实计时器：程序运行时间（常用） SIGALRM
虚拟计时器：用户态消耗时间 SIGVTALRM
实用计时器：用户态和内核态消耗时间 SIGPROF

settimer() gettimer()
struct itimerval timer;
timer.it_interval.tv_sec=1; 时间间隔秒数
timer.it_inerval.tv_usec=1000; 微秒（必须）
timer.it_value.tv_sec=3; 三秒后开始
timer.it_value.tv_usec=0;
settimer(ITIMER_REAL,&timer，0)
三个参数：真实计时器，新计时器，旧计时器
可用 signal(SIGALRM,p)为真实计时器的信号定义一个行为
timer 是为了设置计时器的属性，当使用 settimer 后，就可以使用真实计时器每隔 n 秒后发信号 SIGALRM

usleep(1 000 000)
暂停 1 秒=一百万 us

21.2.5. socket

使用 socket 文件做交互媒介，后缀是.sock,类型为 s
本地 PF_LOCAL 或 PF_UNIX
网络 PF_INET IPV4
PF_INET6 IPV6

UDP SOCK_DGRAM 打包发送
TCP SOCK_STREAM 数据流的方式发送

21.2.5.1. 服务器

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
	int sockSrv=socket(AF_INET,SOCK_STREAM,0);
	if (sockSrv == -1)
	{
		perror("socket 创建出错！");
		return 0;
	}

	struct sockaddr_in	addrSrv;
	addrSrv.sin_family=AF_INET;
	addrSrv.sin_port=htons(5555);//端口号
	addrSrv.sin_addr.s_addr = INADDR_ANY;
	bzero(&(addrSrv.sin_zero),8);

	if (bind(sockSrv, (struct sockaddr *)&addrSrv, sizeof(struct sockaddr)) == -1)
	{
		perror("bind 出错！");
		return 0;
	}
```

```
    if (listen(sockSrv, 5) == -1)//最多能接受 5 个客户端链接
    {
        perror("listen 出错！ ");
        return 0;
    }


    while(1)
    {
        struct sockaddr_in addrClient;
        socklen_t sin_size = sizeof(struct sockaddr_in);

        //接受链接
        int sockConn = accept(sockSrv, (struct sockaddr *)&addrClient, &sin_size);

        //发送
        char sendBuf[100]={0};
        sprintf(sendBuf,"%s",inet_ntoa(addrClient.sin_addr));
        send(sockConn,sendBuf,strlen(sendBuf)+1,0);

        //接收
        const int bufSize = 100;
        char recvBuf[bufSize ] = {0};
        recv(sockConn,recvBuf,bufSize ,0);//返回值为接收到的实际大小,网络错误时，返回
SOCKET_ERROR
        printf("%s\n",recvBuf);

        //关闭
        close(sockConn);
    }

    return 0;
}
```

21.2.5.2. 客户端

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
```

```
#include <sys/socket.h>

int main(int argc, char *argv[])
{

	int sockClient=socket(AF_INET,SOCK_STREAM,0);

	struct hostent *host = gethostbyname("127.0.0.1");
	struct sockaddr_in addrSrv;
	addrSrv.sin_family=AF_INET;
	addrSrv.sin_port=htons(5555);
	addrSrv.sin_addr = *((struct in_addr *)host->h_addr);
	bzero(&(addrSrv.sin_zero),8);

	//连接
	if (connect(sockClient, (struct sockaddr *)&addrSrv, sizeof(struct sockaddr)) == -1)
	{
		perror("connect 出错！ ");
		return 0;
	}

	//接收
	const int bufSize = 100;
	char recvBuf[bufSize ]={0};
	recv(sockClient,recvBuf,bufSize ,0);
	printf("%s\n",recvBuf);

	//发送
	send(sockClient,"clientSend",strlen("client")+1,0);

	close(sockClient);

	return 0;
}
```

21.2.6. 进程

21.2.6.1. 定义

linux 内核启动 0 进程，0 进程启动 1 进程和 2 进程（有些 linux 只启动 1 进程），1，2 再启动其他所有进程

进程常见的状态：

S 休眠状态
s 有子进程
o 可运行状态
R 运行状态
z 僵尸进程（已结束但资源未回收）
1）父进程启动子进程，父子进程同时运行
2）如果子进程先结束，子进程给父进程发信号，由父进程负责回收子进程的相关资源
3）如果父进程先结束，自进程会把 init（进程 1）作为新的父进程
4）如果子进程先结束，同时发出的信号父进程未收到或子进程未发信号，子进程就变成僵尸进程

linux 下一个进程在内存里有三部分：数据段，堆栈段，代码段
堆栈段存放的是子进程的返回地址，子进程的参数以及子进程的局部变量，而数据段则存放程序的全局变量，常数以及动态数据分配的空间，系统如果同时运行多个相同的程序，他们之间就不可能使用同一个堆栈段和数据段。

21.2.6.2. PID

<unistd.h>
进程 id（PID），非负整型，是唯一的，但也可以延迟重用
getpid() 取当前进程 id
getppid() 取当前进程的父进程 id
getuid()/geteuid() 取当前有效用户的 id

21.2.6.3. fork

<unistd.h>
pid_t fork() 失败返回-1

fork 通过复制自身（父进程）创建子进程。（复制代码区之外的内存区域，但和父进程共享代码区）fork 之前的代码，父进程执行一次，fork 之后的代码父子进程分别执行一次
fork 自身会有两次返回，在父进程返回子进程 id，在子进程返回 0.
fork 之后，父子进程同时运行，谁先运行不确定，子进程和父进程（虚拟内存地址相同），映射的物理内存不同
文件操作时，复制文件描述符，不复制文件表，所以父子进程相当于共用文件指针，不会相互覆盖
fork 函数执行后，系统就为新的进程准备数据段，堆栈段和代码段，首先让新的进程和旧的进程使用同一个代码段，因为他们的程序还是相同的，对于数据段和堆栈段，系统复制一份给新的进程，但实际上数据已经分开，相互之间不再有影

响，不再共享任何数据。fork 复制两个段实际上是逻辑上的，并非物理上的复制，实际执行 fork 时，物理空间上两个进程的数据段和代码段都还是共享的，当有一个进程写了某个数据时，这是两个进程之间的数据才有了区别，系统就将有区别的页从物理上分开。

父进程结束，子进程不结束。

```
pid_t   pid=fork();
if(pid==0)//子进程 pid 为 0
{
      .....子进程执行的代码
          exit(0);
}
.....
.....父进程执行的代码
```

21.2.6.4. vfork

<unistd.h>
pid_t vfork() 失败返回-1
vfork 不复制父进程任何内存空间，创建的子进程占用父进程的内存空间运行，父进程在此时被阻塞。vfork 和 exec 系列函数结合使用才有意义。vfork 负责创建子进程，exec 系列函数负责提供新的程序被执行。未使用 exec 系列函数则行为不确定。
一个进程一旦调用 exec 系列函数，它本身就终止了，系统把代码段替换成新的程序代码，废弃原来的数据段和代码段，并为新程序分配新的数据段和代码段，唯一留下的就是进程号。对系统而言，还是同一个进程，不过已经是另一个程序了。

exec 系列函数（6 种）不是新建一个进程，而是修改进程。用新的程序替换旧的，不改变 pid（占用父进程的空间运行）
execl(“exe 全路径”，“命令”，“选项”，“参数”,NULL）;
execlp 包含系统路径
execle 包含环境表
execl("/bin/ls","ls","-l","/home",NULL);//详细显示 home 目录下所有文件
execlp("ls","ls","-l","/home",NULL);

用于打开其他程序，父进程结束，子进程也结束。

```
pid_t pid = vfork();
if(pid == 0)//子进程 pid 为 0
 {
        子进程执行的代码打开 a 这个程序
          execl("/home/jyl/a",0);
 }
cout<<"baseRun"<<endl;
while(1){}
.....父进程执行的代码
```

21.2.6.5. 终止进程

进程终止的正常情况：
1）main 函数中执行 return
2）调用 exit 函数
3）调用.exit 或者_Exit
4）进程的最后一个线程执行了返回语句
5）进程的最后一个线程调用 pthread_exit 函数

异常终止：
1）调用 abort，产生 SIGABRT 信号
2）进程接收到某些信号
3）最后一个线程对“取消”请求做出响应

```
<unistd.h>
void    _exit(int status);
<stdlib.h>
void _Exit(int    status);
      status:-1    非正常         0 正常
<stdlib.h>
void    exit(int status);
   status:    -1 非正常，   0 正常
<stdlib.h>
void exit(int    status);
```

在退出之前会执行被 on_exit()和 atexit()注册的函数
atexit(函数指针) 注册放在 exit 之前

21.2.6.6. 进程等待

```
<sys/wait.h>
pid_t    wait(int *status)          status 作为返回值类型
```

成功返回子进程 id，失败返回-1

用于父进程等待子进程的结束，等待任意一个子进程结束都会返回
返回结束子进程的状态和退出码
exit（100），则返回的退出码为 100
wait（）可以回收僵尸子进程
宏函数：
WIFEXITED(status)可以判断是否正常结束
  wait if exit end
WEXITSTATUS(status)可以得到退出码
  （0 到 255 有效）
exp：
pid_t pid=fork();
if(pid==0){.....exit(100);}
int status;
int pid2=wait(&status);
  等待子进程结束后执行下面的代码
if(WIFEXITED(status))
  printf("%d\n",WEXITSTATUS(status));

pid_t waitpid(pid_t pid, int *status, int options)
pid: -1 等待任意一个子进程结束
  >0 等待指定子进程结束
  <-1 取绝对值，绝对值代表一个进程组，等待这个进程组中任意一个子进程结束
  0 等待任意和父进程同一个进程组的子进程结束
options： WNOHANG 没有子进程结束，立即返回 0
    wait no hang
    0 一直等待子进程结束

21.2.7. 线程

21.2.7.1. 定义

每个进程的内部，都支持多线程并行。线程不需要有自己独立的内存空间，只是拥有一个独立的栈，一个进程的所有线程共享进程的资源。
进程中必须有一个主线程，主线程结束，进程随之结束，进程结束导致所有线程结束。
多线程之间代码是乱序执行，每个线程内部代码是顺序执行
POSIX 规范中对线程做了完整定义，其中所有函数几乎都以 pthread_开头
#include<pthread.h>

21.2.7.2. 创建

int pthread_creat(线程 id，属性默认为 0，函数指针，函数参数的指针)
函数： void* (*pfun)(void *)
错误处理是通过返回错误码，不是使用 errono，失败返回非 0，成功返回 0

pthread_t id;
int res = pthread_creat(&id,0,pfun,0) 开启并运行线程
if(res) print("%s\n",strerror(res));

线程函数内部调用，参数同线程函数返回值
pthread_exit(void *returnval)

21.2.7.3. 终止

线程的取消就是给目标线程发 CANCEL 信号
目标线程可做出三种选择：忽略、立刻停止、延迟停止
pthread_cancel(id)给目标线程发取消信号

可以在线程内设置 CANCEL 信号的处理
//设置不支持 CANCEL 信号
pthread_setcancelstate(PTHREAD_CANCEL_DISABLE,0);
// 设置 CANCEL 信号的处理方式
pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS,
PTHREAD_CANCEL_DEFERRED);

21.2.7.4. 等待

pthread_t ptId=pthread_self(); 查看当前线程的 id
int pthread_equal(id1,id2); 相等返回非 0，否则返回 0

pthread_join(ptId,0); 等待 ptId 线程的结束

char *res;
pthread_join(id,(void**)&res);//线程的返回值为指针，所以用 pthread_join 第二个参数接收
printf("%s",res);

同步方式：被其他线程等待时，只有 pthread_join()函数返回时，才释放自己占用

的资源
异步方式：运行结束立即释放资源，未被其他线程等待
int pthread_detach(id)
执行后线程不能被 pthrea_join 同步

21.2.7.5. 锁

pthread_mutex_t　lock;//声明互斥变量
pthread_mutex_init(&lock,0);//初始化互斥量

pthread_mutex_lock(&lock);//加锁
.....访问
pthread_mutex_unlock(&lock);//解锁

pthread_mutex_destroy(&lock);//销毁锁

21.2.7.6. 信号量

信号量是一个计数器，用来控制访问临界资源的线程最大并行数量，当信号量的初始值为 1 时，效果等同于互斥量
信号量不是最早的 POSIX 线程相关规范，因此信号量相关定义是在 semaphore.h 中，其中定义的信号量用于线程（也可以用于进程，linux 不支持）。

sem_t　sem;
//0 代表线程计数（其他值代表进程计数，linux 不支持）
sem_init(&sem,0,计数初始值);

//获取信号量　计数减 1　　减到 0 时就代表上锁，此时阻塞
sem_wait(&sem);
.....访问资源
释放信号量（计数加 1）
sem_post(&sem);

删除信号量
sem_destroy(&sem)

21.2.8. 文件

<unistd.h>
void* sbrk(n)
n>0 分配 n 个字节的空间
n<0 释放 n 个字节的空间
n==0 返回当前的位置
sbrk 底层自动维护一个位置指针，通过位置的移动分配、释放内存
分配时，指针向后移动 n 个字节，分配结束后返回值为之前的指针的位置
释放时，指针向回移动 n 个字节，返回值为之前的指针位置（无意义）
sbrk 单向连续，先进后出（先分配的后释放），分配方便，释放不方便

int brk（void *）
失败返回-1
释放方便，分配不方便（brk 的指针不移动）

exp：
void *p=sbrk(0); 取当前位置
brk（p+4）； 分配 4 个字节的空间
brk（p+8）； 分配 4 个字节的空间（上一步已分配过 4 个）
brk（p+4）； 释放 4 个字节空间
brk（p）； 释放所有已分配的内存

<sys/mman.h> memory manage
void * mmap(0,4,
首地址，0 代表内核指定 分配 4 字节（映射 1 页）
PROT_READ|PROT_WRITE,
权限
MAP_PRIVATE|MAP_ANONYMOUS，
共有或私有 匿名（针对物理内存）
0,
目标处（如 fd），可从文件 fd 处分配内存，fd 必须有足够的大小，0 代表系统从硬盘分配
0)
偏移量

mmap 用来映射内存，可以是硬盘或文件

munmap(void* ,4) 释放 4 字节

内核中提供系统函数，如 sbrk（），可通过 sbrk 访问内核
time 命令可以测试程序在用户层和内核层的运行时间

linux 中，几乎一切都被看成了文件
<unistd.h>　读写
<fcntl.h>
int open(const char *pathname,int flags,权限)
新建文件，需要第三个参数，如果只打开则不需要
exp：
int　fd=open(“a.txt”,O_CREAT|O_RDWR,0666);
　　三位八进制权限，权限屏蔽写（2）
int　fd1=open(“a.txt”,O_RDONLY);
if(fd==-1) perror(“open”),exit(-1);

fd 为文件描述符
文件表会记录在硬盘上的文件信息（i 节点）。打开一个文件，用文件表记录文件，文件描述符在文件表中，指代文件，描述符为数字，从 3 开始，0，1，2 被系统占用

O_RDONLY　0
O_WRONLY　1
O_RDWR　2
O_APPEND　1024　追加方式打开
O_CREAT　不存在会新建，存在则打开

close（int fd） 关闭文件
ssize_t　read(int fd,　const　void　*buf,　size_t　count)
ssize_t　write(int fd,　const　void　*buf ,　size_t　count)
成功则返回读写的字节数，失败返回-1
读一般用：fd=open(“a.txt”,O_RDONLY);
写一般用：fd=open(“a.txt”,O_CREAT|O_TRUNC,0666);

标 c 都有输入、输出缓冲区，而 uc 函数在用户层无缓冲区，当频繁输入，输出时，uc 函数最好定于一个缓冲区。
char buf[]　先输入到 buf，然后一次性输入到位置

int dup(int oldfd)　复制一个文件描述符，新符号由内核指定
int dup(int oldfd, int newfd)　自定义新描述符，如果已存在，则关闭原来的，然后再使用
失败返回-1
off_t　lseek(int fd,偏移量，偏移起始位置)
SEEK_SET　SEEK_CUR　SEEK_END

fcntl(int fd, int cmd, ...)
1)可以复制文件描述符号（cmd 取 F_DUPFD）
　复制文件描述符时，和 dup 类似，不同点为当重名时，不会关闭已经使用的描述符，返回值是大于等于第三个参数

2)可以获取、设置文件的状态（cmd 取 F_SETFL,F_GETFL）
权限标识判断：
int flags=fcntl（fd，F_GETFL）;
1.if（（flags&3）==O_RDONLY）
3 代表二进制 11，O_RDONLY 为 0
判断只读时使用，因为只读为 0，任何值的二进制位与 0 都为 0，flags&3 取值只能是 0，1，2 中的一个
2.其他情况
if（flags&O_APPEND）
3)实现文件锁（cmd 取 F_SETLK,F_GETLK,F_SETKW）
fcntl（fd，cmd，&lock）;
读锁：共享锁，锁定其它写操作、允许读操作
写锁：锁定其它进程的操作
文件锁对应一个结构体：
struct flock{
short l_type; 锁的类型
short l_whence; 锁的开始位置
int l_start; 偏移量 和锁的开始位置同时使用
int l_len; 锁定长度，字节为单位
pid_t l_pid; 加锁的进程 id（一般为-1）
}
l_type: F_RDLCK,F_WRLCK,F_UNLCK
l_whence: SEEK_SET,SEEK_CUR,SEEK_END
l_pid: 只有在 F_GETLK 时有效，其他置-1 即可
exp：
int fd=open（.....）
描述锁: struct flock lock={F_RDLCK,SEEK_SET,0,20,-1}
加锁： int res=fcntl(fd,F_SETLK,&lock);
if(res==-1)perror(“lock”),exit(-1);
解锁：
lock.l_type=F_UNLCK;
res=fcntl(fd,F_SETLK,&lock);
if(res==-1)perror(“unlock”),exit(-1);
文件锁不能锁定硬盘上的文件，不能锁定 read，write 函数，只能阻止其他进程的上锁行为。

正确用法：在调用读函数前上读锁，调用写函数前上写锁。用 if 语句判断是否上锁成功，如果成功则执行读写
可使用 F_SETLKW 实现加不上锁，则等待上锁。参数为 F_GETLK 时，不是获取锁，而是测试某个锁是否能加上。如果已经有一个进程 1 中的锁 la 在运行，当前进程 2 中锁 lb 如果不可以加，则 lb 的参数都变成 la 的参数（其中 lb 的 l_pid 变成进程 1 的 id）如果 lb 可以加上，lb 中的（_type 变为 F_UNLCK）其他不变

<sys/stat.h>
struct stat st;
int res=stat("a.txt",&st);
if(res==-1)perror("stat"),exit(-1);
printf("size=%d\n",st.st_size);
通过 stat 可得到文件的很多信息，stat 结构体包括：st_vid,st_gid,
st_nlink,st_mode,st_size,st_atime,st_mtime,st_ctime.....

<unistd>
int access(const char *pathname,int mode);
mode: R_OK,W_OK,X_OK,F_OK(存在)
存在权限则返回 0，不存在则返回-1
mode 可写多个组合：R_OK|W_OK

<sys/stat.h>
umask(0022) 0 代表是八进制，后面三位对应用户
设置新屏蔽，返回旧屏蔽
对 u 不屏蔽，对 g，o 屏蔽写权限
exp：
mode_t m=umask（0022）;
int fd=open("a.txt",O_CREAT|O_RDWR,0666)
创建文件时被屏蔽权限（0022）
umask(m); 恢复之前的权限屏蔽

int chmod(const char *pathname,mode_t mode);
修改文件的权限

<sys/types.h>
truncate()/ftruncate() 可指定文件大小
exp： truncate("a.txt",100);
ftruncate(fd,100);

<sys/types.h>
目录操作 （目录中都包含.和..目录）
int mkdir(const char *pathname, mode_t mode);
rmdir(const char *pathname)
chdir(const char *pathname)
getcwd(char *buf, long size);
读入当前位置放入到 buf 中，当前目录大小值放入 size

<dirent.h>
DIR * dir=opendir("../ab");
打开上级目录下的 ab 目录
if(dir==NULL)perror("opendir"),exit(-1);

```
while(1){
struct dirent *     ent=readdir(dir);
        返回一个目录，并指向下一条目录
if(ent==NULL) break;   注意判断.目录
printf("%d,%s\n",ent->d_type,ent->d_name);
}
type：4 是目录
        8 是普通文件
rewinddir()
telldir()
seekdir()   类似文件指针的用法

动态调用共享库
<dlfcn.h>
代码在运行时才确定调用哪个函数
相关函数：dlopen()       dlclose()
                dlsym()         dlerror()
       dlsym()从一个打开的库文件中获取一个函数的指针
dlopen()第一个参数是共享库文件名，第二个是加载方式
   runtime    loading     RTLD_NOW       open 的同时加载
                          RTLD_LAZY       延迟加载

exp:     void * handle=dlopen("libxx.so",RTLD_NOW);
         char    *error=dlerror();
         int    (*p)(int,int);
         p=dlsym(handle,"ab");
               ab 为 libxx.so 中的函数名，ab 形式与 p 指向形式相同
         int    d=p(1,1);
         dlclose(handle);
gcc    dl.c    -ldl
```

21.3. cmake

编写 CMakeLists.txt 文件

cmake   目录

make

cmake 命令生成 makefile

make 命令进行编译

基本结构：

cmake_minimum_required (VERSION 2.6)

project (工程名)

add_executable(工程名 源文件.cpp)

21.3.1. 代码使用外部定义

21.3.1.1. 外部头文件

CMakeLists.txt 文件中：

```
set(EXT_INFO   123)

    configure_file (
          "${PROJECT_SOURCE_DIR}/cfg.h.in"
           "${PROJECT_BINARY_DIR}/cfg.h"
      )

    include_directories("${PROJECT_BINARY_DIR}")
```

编写 cfg.h.in

```
#define EXT_INFO @EXT_INFO@
```

cmake 会根据 cfg.h.in 在 PROJECT_BINARY_DIR 目录下自动生成 cfg.h（CMakeLists.txt 中将该目录加入头文件包含目录）

源代码中可以应用该头文件，并使用 EXT_INFO

21.3.1.2. 外部开关

＝＝＝＝＝＝定义开关

编写 cfg.h.in 时定义一个开关变量

```
#cmakedefine  ABCD
```

＝＝＝＝＝＝开关控制

CMakeLists.txt 文件中控制开关：

１．选项控制

option (ABCD

"..............." ON)

通过该 option 命令 ON 可以让 cmake 生成的 cfg.h 中包含#define ABCD

通过该 option 命令 OFF 则取消#define ABCD

２．函数检查

include (${CMAKE_ROOT}/Modules/CheckFunctionExists.cmake)

///////include (CheckFunctionExists)

```
check_function_exists (log ABCD)
```

如果函数存在则可以让 cmake 生成的 cfg.h 中包含#define ABCD

＝＝＝＝＝＝开关使用

１．CMakeLists.txt 文件中可以使用对应定义

if (ABCD)

。。。。

endif (ABCD)

２．代码中可以使用该 ABCD)宏

21.3.2. 子目录链接

代码目录下包含 AA 目录

AA 目录中包含 t1.cpp，其中实现 fun 函数

AA 目录中包含一个 CMakeLists.txt：

add_library(AA t1.cpp)

(可以对应生成 libAA.a)

代码目录中可以直接声明 AA 中的函数 fun，并调用

代码目录中 CMakeLists.txt:

```
cmake_minimum_required (VERSION 2.6)
project (tt)

include_directories ("${PROJECT_SOURCE_DIR}/AA")
add_subdirectory (AA)
set (EXTRA_LIBS ${EXTRA_LIBS} AA)

add_executable(tt main.cpp)
target_link_libraries (tt ${EXTRA_LIBS})
```

21.3.3. 安装

代码目录中 CMakeLists.txt:

```
install (TARGETS tt DESTINATION bin)
```

（放到文件末尾）

可以将生成的执行文件 tt 拷贝到 /usr/local/bin 目录下

```
install (FILES t1.h DESTINATION include)
```

可以将文件 t1.h 拷贝到 /usr/local/include 目录下

21.3.4. 测试

代码目录中 CMakeLists.txt 末尾：

include(CTest)

add_test (ttRun tt argsss)

set_tests_properties (ttRun PROPERTIES PASS_REGULAR_EXPRESSION "100")

测试用例名为 ttRun，argss 为启动参数

程序输出中是否包含字符串１００

使用 ctest 命令进行测试

可以使用宏进行用例编写

```
macro (do_test arg result)
  add_test (testName${arg} Tutorial ${arg})
  set_tests_properties (testName${arg}
    PROPERTIES PASS_REGULAR_EXPRESSION ${result})
endmacro (do_test)


do_test (25 "25 is 5")
do_test (-25 "-25 is 0")
```

21.3.5. 编译时运行程序

代码目录下包含 AA 目录

AA 目录中包含 t2.cpp:

```
int main(int argc,char** argv)
{
   if(argc != 2) cout<<"erro"<<endl;

   char *FilePathName = argv[1];
   FILE *pf = fopen(FilePathName,"w");
```

```
    fprintf(pf,"xxx");
    fclose(pf);
    return 0;
}
```

接收文件名参数，进行写文件操作。

AA 目录中对应的 CMakeLists.txt 文件：

```
add_executable(t2 t2.cpp)
add_custom_command (
    OUTPUT ${CMAKE_CURRENT_BINARY_DIR}/t2Ret.txt
    COMMAND t2 ${CMAKE_CURRENT_BINARY_DIR}/t2Ret.txt
    DEPENDS t2
    )
include_directories( ${CMAKE_CURRENT_BINARY_DIR} )

add_library(AA t1.cpp ${CMAKE_CURRENT_BINARY_DIR}/t2Ret.txt)
```

add_executable 编译该程序，再运行该程序，传入 t2Ret.txt 文件路径作为参数，让生成的 t2 程序生成该文件，并通过 add_library 加入项目输出

21.3.6. 项目打包

代码目录中 CMakeLists.txt 末尾：

```
include (InstallRequiredSystemLibraries)
set (CPACK_RESOURCE_FILE_LICENSE
    "${CMAKE_CURRENT_SOURCE_DIR}/License.txt")
set (CPACK_PACKAGE_VERSION_MAJOR "${×××_VERSION_MAJOR}")
set (CPACK_PACKAGE_VERSION_MINOR "${×××_VERSION_MINOR}")
include (CPack)
```

License.txt 源代码打包时使用
版本信息可以使用 CMakeLists 中的宏定义

打包程序命令：

```
cpack --config CPackConfig.cmake
```

打包源代码命令：

cpack --config CPackSourceConfig.cmake