**Linux：**

if [ str1 = str2 ] 当两个串有相同内容、长度时为真

if [ str1 != str2 ] 当串str1和str2不等时为真

if [ -n str1 ] 当串的长度大于0时为真(串非空)

if [ -z str1 ] 当串的长度为0时为真(空串)

if [ str1 ]　 当串str1为非空时为真

gcc工具链

objdump -S objfile

objdump -d objfile

objdump -h objfile

gdb调试

gdb 进程名字 coredump文件名字

gdb attach 进程号

bt ：打印堆栈

wget

wget http://www.linuxde.net/testfile.zip

linux个别常用命令

sed 's/要被取代的字串/新的字串/g'

例如 cat /etc/passwd | sed 's/root:x:/root::/g' > /3rd\_rw/passwd.txt

查看线程占用CPU使用率

1 先用ps + grep找出该死的进程pid，比如 30420

2 top -H -p pid(进程号)(top然后shift+H可以看出某个线程)所有该进程的线程都列出来了。看看哪个线程pid占用最多，然后将这个pid转换为16进制，如 44bf，注意要小写

3 jstack 30420 | less，然后查找 nid=0x44bf，哦，找到了

**c++学习：**

**Overload(重载)：**在C++程序中，可以将语义、功能相似的几个函数用同一个名字表示，但参数或返回值不同（包括类型、顺序不同），即函数重载。  
（1）相同的范围（在同一个类中）；  
（2）函数名字相同；  
（3）参数不同；  
（4）virtual 关键字可有可无。

**Override(覆盖)：**是指派生类函数覆盖基类函数，特征是：  
（1）不同的范围（分别位于派生类与基类）；  
（2**）**函数名字相同；  
（3）参数相同；  
（4）基类函数必须有virtual 关键字。

（5）**Override**函数的访问修饰符可以不同，尽管virtual函数是private的，在派生类中**Override**的函数可以是public或protected的

（6）覆盖的函数不能是static函数；即virtual关键字不需要和static关键字一起使用

**Overwrite(重写)：**是指派生类的函数屏蔽了与其同名的基类函数，规则如下：  
（1）如果派生类的函数与基类的函数同名，但是参数不同。此时，不论有无virtual关键字，基类的函数将被隐藏（注意别与重载混淆：类的范围不同）。  
（2）如果派生类的函数与基类的函数同名，并且参数也相同，但是基类函数没有virtual关键字。此时，基类的函数被隐藏（注意别与覆盖混淆：virtual关键字）

**C++中static**

• 出现在类体外的函数定义不能指定关键字static；

•虚函数不需要static关键字  
• 静态成员之间可以相互访问，包括静态成员函数访问静态数据成员和访问静态成员函数；   
• 非静态成员函数可以任意地访问静态成员函数和静态数据成员；   
• 静态成员函数不能访问非静态成员函数和非静态数据成员；   
• 由于没有this指针的额外开销，因此静态成员函数与类的全局函数相比速度上会有少许的增长；  
• 调用静态成员函数，可以用成员访问操作符(.)和(->)为一个类的对象或指向类对象的指针调用静态成员函数，也可以直接使用如下格式：  
＜类名＞::＜静态成员函数名＞（＜参数表＞）

•基类定义的static变量，派生类如果没有定义同名的变量，可以访问基类的静态变量，并且修改其内容，基类再查看这个变量的内容会发现内容也被修改。（如果不是static变量，则基类和派生类修改的都是各自变量的内容）

**STL（Standard Template Library）之map**

  1   头文件   
  #include   <map>     
  2   定义   
  map<string,   int>   my\_Map;   
  或者是typedef     map<string,   int>   MY\_MAP;   
  MY\_MAP   my\_Map;   
  3   插入数据   
  (1)   my\_Map["a"]   =   1;   
  (2)   my\_Map.insert(map<string,   int>::value\_type("b",2));   
  (3)   my\_Map.insert(pair<string,int>("c",3));   
  (4)   my\_Map.insert(make\_pair<string,int>("d",4));   
  4   查找数据和修改数据   
  (1)   int   i   =   my\_Map["a"];   
            my\_Map["a"]   =   i;   
  (2)   MY\_MAP::iterator   my\_Itr;   
            my\_Itr.find("b");   
            int   j   =   my\_Itr->second;   
            my\_Itr->second   =   j;   
  不过注意，键本身是不能被修改的，除非删除。   
  5   删除数据   
  (1)   my\_Map.erase(my\_Itr);   
  (2)   my\_Map.erase("c");   
  还是注意，第一种情况在迭代期间是不能被删除的，道理和foreach时不能删除元素一样。   
  6   迭代数据   
  for   (my\_Itr=my\_Map.begin();   my\_Itr!=my\_Map.end();   ++my\_Itr)   {}   
  7   其它方法   
  my\_Map.size()            返回元素数目   
  my\_Map.empty()       判断是否为空   
  my\_Map.clear()          清空所有元素

**C++中指针和引用的区别**

指针和引用在C++中很常用，但是对于它们之间的区别很多初学者都不是太熟悉，下面来谈谈他们2者之间的区别和用法。

1.指针和引用的定义和性质区别：

(1)指针：指针是一个变量，只不过这个变量存储的是一个地址，指向内存的一个存储单元；而引用跟原来的变量实质上是同一个东西，只不过是原变量的一个别名而已。如：

int a=1;int \*p=&a;

int a=1;int &b=a;

上面定义了一个整形变量和一个指针变量p，该指针变量指向a的存储单元，即p的值是a存储单元的地址。

而下面2句定义了一个整形变量a和这个整形a的引用b，事实上a和b是同一个东西，在内存占有同一个存储单元。

(2)可以有const指针，但是没有const引用；

(3)指针可以有多级，但是引用只能是一级（int \*\*p；合法 而 int &&a是不合法的）

(4)指针的值可以为空，但是引用的值不能为NULL，并且引用在定义的时候必须初始化；

(5)指针的值在初始化后可以改变，即指向其它的存储单元，而引用在进行初始化后就不会再改变了。

(6)"sizeof引用"得到的是所指向的变量(对象)的大小，而"sizeof指针"得到的是指针本身的大小；

(7)指针和引用的自增(++)运算意义不一样；

**2.指针和引用作为函数参数进行传递时的区别。**

**(1)指针作为参数进行传递**：

#include<iostream>

using namespace std;

void swap(int \*a,int \*b)

{

　　int temp=\*a;

　　\*a=\*b;

　　\*b=temp;

}

int main(void)

{

　　int a=1,b=2;

　　swap(&a,&b);

　　cout<<a<<" "<<b<<endl;

　　system("pause");

　　return 0;

}

**结果为2 1；**

用指针传递参数，可以实现对实参进行改变的目的，是因为传递过来的是实参的地址，因此使用\*a实际上是取存储实参的内存单元里的数据，即是对实参进行改变，因此可以达到目的。

再看一个程序;

#include<iostream>

using namespace std;

void test(int \*p)

{

　　int a=1;

　　p=&a;

　　cout<<p<<" "<<\*p<<endl;

}

int main(void)

{

int \*p=NULL;

test(p);

if(p==NULL)

cout<<"指针p为NULL"<<endl;

system("pause");

return 0;

}

运行结果为：

0x22ff44 1

指针p为NULL

大家可能会感到奇怪，怎么回事，不是传递的是地址么，怎么p会是NULL？事实上，在main函数中声明了一个指针p，并赋值为NULL，当调用test函数时，事实上传递的也是地址，只不过传递的是指地址。也就是说将指针作为参数进行传递时，事实上也是值传递，只不过传递的是地址。当把指针作为参数进行传递时，也是将实参的一个拷贝传递给形参，即上面程序main函数中的p何test函数中使用的p不是同一个变量，存储2个变量p的单元也不相同（只是2个p指向同一个存储单元），那么在test函数中对p进行修改，并不会影响到main函数中的p的值。

如果要想达到也同时修改的目的的话，就得使用引用了。

**（2）将引用作为函数的参数进行传递。**

在讲引用作为函数参数进行传递时，实质上传递的是实参本身，即传递进来的不是实参的一个拷贝，因此对形参的修改其实是对实参的修改，所以在用引用进行参数传递时，不仅节约时间，而且可以节约空间。

看下面这个程序：

#include<iostream>

using namespace std;

void test(int &a)

{

　　cout<<&a<<" "<<a<<endl;

}

int main(void)

{

int a=1;

cout<<&a<<" "<<a<<endl;

test(a);

system("pause");

return 0;

}

输出结果为： 0x22ff44 1

          0x22ff44 1

再看下这个程序：

这足以说明用引用进行参数传递时，事实上传递的是实参本身，而不是拷贝。

所以在上述要达到同时修改指针的目的的话，就得使用引用了。

#include<iostream>

using namespace std;

void test(int \*&p)

{

　　int a=1;

　　p=&a;

　　cout<<p<<" "<<\*p<<endl;

}

int main(void)

{

int \*p=NULL;

test(p);

if(p!=NULL)

cout<<"指针p不为NULL"<<endl;

system("pause");

return 0;

}

输出结果为：0x22ff44 1

         指针p不为NULL

**html**

1. 在js代码中更改css（样式）元素的属性

var div=document.getElementById('div');

div.style.setProperty('border','1px red solid');

<marquee behavior=alternate scrollamount =3>aaaa</marquee>