
C++
同一个终端启动的进程默认会在一个session里。例如图形界面的终端（比如GNOME按ctrl+atl+T呼出的命令行界面），都是虚拟终端（Virtual terminal），他们实质上只有一个终端在真正起作用，输入w命令，可以看到所有的control terminal。
运算符
&&，||，！
数组
不使用new创建的数组。采用的是静态联编，即数组的长度在编译时设置
int nums[10];
使用new创建数组时，使用动态联编（动态数组），这种数组使用完后应当delete释放。
int * nums = new int [3];
string
字符数组形式
//手动指定大小，主要要把\0也计算在内
char people[20] = "one , two, three";
//让编译器自行判断大小
char people[] = "one , two, three";
字符指针形式
char * name = "hello"; // 双引号是char * 类型
内置string
string newName = name; //char * 可以隐式转换为string类型
循环
for
for (int i = 0; i < 10 ; ++i) {}
遍历容器：int intlist[5] = {1,2,3,4,5};
for (int i: intlist) {
cout << i << endl;
}
while
流程控制
switch
break
continue
指针与引用
指针
引用
引用在实现的时候，传递的就是一个指针给函数！！。不仅仅是对于简单数据类型如此，对于复杂数据类型这也是同样的
并且因为引用不可能为空，所以不需要检查它的合法性，在一定程度上效率比指针都高。
区别
1. 指针是一个实体，而引用仅是个别名。所以我们使用 "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；
2. 引用使用时无需解引用（*），指针需要解引用；
3. 引用只能在定义时被初始化一次，之后不可变；指针可变；
4. 引用变量必须在声明时就初始化，而指针可以先声明，再赋值；
5. 引用没有 const，指针有 const；
6. 引用不能为空，指针可以为空；
用指针还是引用变量？
如果数据对象是类对象，则使用引用变量。其他用指针
类
属性类别
private、public、protected。默认为private
private：私有变量，只要本类方法跟友元函数可以访问。子类也不能访问
public
protected：派生类的成员可以直接访问基类的protected变量，但不能直接访问private变量
成员方法
成员方法必须实现
作用域解析运算符::来标识函数所属的类。比如，void Demo::changeAge(int num) {}表示Demo类的changeAge方法
成员变量
成员变量与成员函数是分开存储的，成员函数并不在类对象中
静态成员变量、静态成员函数也不在类对象中
提供的默认方法
默认构造函数
每次创建类的新对象实例时执行。构造函数的名称与类的名称是完全相同的，并且不会返回任何类型，也不会返回 void。构造函数可用于为成员变量（私有、公有都可以）设置初始值
使用列表为构造函数初始化字段
默认构造函数没有任何参数。如果实例化对象时没有显式的初始化，则会调用默认构造函数。如果程序中没有提供任何构造函数，则编译器会定义一个默认的构造函数
默认析构函数
同样不会返回任何类型，也不会返回 void
拷贝构造函数
它在创建对象时，是使用同一类中之前创建的对象来初始化新创建的对象
写法：Line(const Line & obj);
比如，Line line2 = line1。将lin1实例赋给了lin2实例，就会调用拷贝函数
赋值运算符
地址运算符
静态成员变量、函数
静态成员变量
与静态变量一样，编译阶段就分配了内存
即可以通过对象访问，也可以通过类名访问
可以类内声明，类外初始化（这样就不会执行构造函数）
也受权限约束
静态成员函数
所有对象都共享同一份函数
即可以通过对象访问，也可以通过类名访问
也受权限约束
静态成员函数只能访问静态成员变量
虚函数
实现用基类指针调用子类的虚成员函数
组成
虚函数指针
每个类的实例化对象都会拥有虚函数指针并且都排列在对象的地址首部
虚函数表
将虚函数指针按照一定的顺序组织起来的，从而构成了一种表状结构，称为虚函数表
虚函数的声明与定义要求非常严格，只有在子函数中的虚函数与父函数一模一样的时候（包括限定符）才会被认为是真正的虚函数，不然的话就只能是重载
友元
友元函数
通过将普通函数变为类的友元函数，可以赋予该函数与类成员函数相同的访问权限。
创建Demo类的友元：friend Demo add(int num, const Demo & d)
友元类
友元类可以访问基类的全部对象，包括私有对象。
声明友元类时，友元类中所有的函数变为了友元函数，而后面派生新加的函数则不为友元函数
友元成员函数
与派生类的关系
派生类不继承基类的友元函数，但是可以通过显式类型转换来访问基类的友元
继承
私有
公有
公有派生，基类的公有数据跟方法都可以被派生类使用。基类的私有对象不可使用
class BrithFromBase : public Base
protect
子类初始化
必须显式指明要使用的基类构造函数，否则会使用基类默认的构造函数
派生类构造函数通过成员初始化，将基类需要的信息传递给基类构造函数
初始化顺序：先调用父类的构造函数，再调用派生类的构造函数。析构函数与构造函数调用顺序相反
其他关系：基类指针、引用可以在不进行显式类型转换的情况下指向派生类对象。 Base & rt = bfm1;
Base * pt = &bfm1;
子类会隐藏父类中所有的同名函数（因为可能有函数重载），访问父类同名函数只需要加上父类作用域即可：Son::Base::func()
多继承
顺序
顺序从左到右。如果父类还有父类，先进行深度遍历，先构造父父类。
先往上找，找完再回来
python虽然也是先广度，再深度。但是python是将广度进行完，再进行深度。不会先往上找，找完再会回来
虚继承
解决：多个父类中有同名成员时，会有二义性，并且我们只需要一份
基类指针与父类指针的转换
基类指针或引用转为子类指针或引用称为向下强转，这是不允许的，因为基类比子类内容少，容易发生指针越界。相反可以
模版
函数模版
template <typename T>
T add(T &a, T &b) {
return a+b;
}
显式具体化模板(将参数定死)：res = add<int>(x,y);
函数模板不能进行隐式类型转换
函数模板与普通函数调用原则
如果函数模板与普通函数发生重载，那么普通函数优先执行
如果想强制使用函数模板，那么可以使用空模板参数列表
如果函数模板有更好的匹配，那么优先使用函数模板
模板也支持重载
template <typename T>
void swap(T *a, T *b, int n){};
类模板
template<class NAMETYPE, class AGETYPE = int >
类模板与函数模板的区别
类模板可以指定默认类型，函数模板不行
类模板不能使用自动类型推导
类模板中的成员函数，并不是一开始创建的，而是在运行阶段确定出T的数据类型才去创建的
多态
静态多态
运算符重载和函数重载就是编译时多态
动态多态
派生类和虚函数是运行时多态
const
方法被const修饰时，表示不允许该函数对类内成员做修改
void func() const
const在 * 号前面
const int *p1;
表示p1只有对变量的读权限，不能变量
子主题
const在 * 号后面
int * const p1;
表示p1只能指向这个变量，可以修改这个变量的值
const在 * 号两侧
const * int const p1
指针本身和它指向的数据都有可能是只读的
auto
子主题
auto声明的变量存放在栈中，栈是后进先出的
mmap
建立存储映射区
实现没有血缘关系的多进程通信
作用域
类的作用域
类中定义的变量、方法等对象的作用域为整个类
匿名类
特点，调用完后立即释放
子主题
函数
普通函数
参数
支持默认参数（但c不支持）
仿函数
仿函数不是函数，它是个重载了()运算符的类，使得它的对你可以像函数那样子调用
函数模板
重载
类的重载
以重载加法为例
Line Line::operator+(const Line & obj) const{}
重载限制
重载后的运算符必须至少有一个操作数是用户自定义的类型，即不能是c++的标准数据类型。目的是为了提醒用户
不能违反原来运算符的法则。比如变加法为减法
不能修改运算符的优先级
不能重载像. ? :: 等特殊运算符
C不支持重载。因为C在编译时会改变原来的函数名。而C++虽然也会改变，但是进行了名称优化
进程
进程组
进程组：每个进程都属于进程组。父进程创建子进程时，它们属于同一个进程组
因为父进程时该进程组里的第一个进程，所以组长ID==父进程ID
只要进程组中还有进程，那么就存在。与组长进程没关系
父进程将子进程启动后就退出了，让子进程在后台一直运行，就达到了守护进程的目的
kill -SIGKILL -进程组ID即可杀死整个进程组
会话
一个或多个进程组的集合
创建会话的不能是组长 ‘
线程
线程号
每个进程都有自己的PID，但是所有的子线程共用主线程的PID，但是它们有不同的线程号
栈不共享，堆共享
线程分离
线程主动与主线程断开关系。线程结束后，其状态不由其他线程获取，而是自己主动释放。网络、多线程服务器常用
取消线程
pthread_cancel调用并不等待线程终止，它只提出请求。线程在取消请求(pthread_cancel)发出后会继续运行，直到到达某个取消点才会真正取消
线程同步
生产者-消费者
锁机制
互斥锁
子主题
读写锁
可以同时读，但写只能有一个。用于读次数远大于写次数的场景
信号量
信号
信号集
未决信号集与阻塞信号
进程在接收到信号后，会先在未决信号集中置为1，然后去阻塞信号集中看是否被阻塞。如果进程被阻塞则等待。没阻塞则执行信号动作，然后将未决信号集中置为0，标示信号以被处理
信号的编号
信号的名称
产生信号的事件
信号的处理动作
相关函数有
kill
raise，给当前进程发送信号
abort，给自己发送异常终止信息
管道
管道其实是内核缓冲区。默认缓冲区大小为4k
数据被读取后就会清空缓冲区
管道的两端是阻塞的。
如果写端全部关闭了，那么read也会结束阻塞
读端全部关闭，则该进程立刻结束。（因为管道会溢出）
谓词、内建函数对象、适配器
谓词
函数或仿函数，返回值为bool类型的，叫做谓词。
内建函数对象
适配器(偏函数)
mmap
让用户程序直接访问内存，这种机制，相比较在用户空间和内核空间互相拷贝数据，效率更高
为了将文件映射到内存中。操作内存到变更会影响文件
流程：open打开文件描述符后。使用mmap映射文件内容到内存，修改的话用memcpy
异常
throw-try-catch
四大类型转换
static_cast<目标类型>(原对象)
允许基本类型转换
允许存在父子关系的类之间的指针或引用转换，即使转换不安全，也可以转
dynamic_cast
不允许基本类型转换
允许存在父子关系的类之间的指针或引用转换，子转父可以，父转子需要在有多态的情况下才会成功
常量转换 const_cast
只能对指针或引用使用
reinterpret_cast
最不安全的一种转换。两个毫无关系的类型可以通过编译
STL
子主题
编译
符号冲突问题
动态联编
编译器必须在程序运行时才能确定正确的虚函数的代码，因此这也称为动态联编
总之，编译器对非虚方法使用静态联编
内置指令
using
c++支持 xxx.h或者xxx的方式导入头文件，但需要注意的是，以xxx形式导入需要使用using namespace xxx指令
共同体
与c中的联合体一样，可以有多种类型，但只保存最后初始化的类型的值
分配内存
使用new来分配内存。在堆中（c使用的是malloc，跟这里new的作用一样）
new分配的内存要用delete释放掉。
int * ps =new int;
delete ps;
这将释放ps所指向的内存，但不会删除ps指针本身，可以把ps指向新的内存块。
不要创建两个指向同一内存块的指针，因为这可能造成这块内存释放不掉：
int * ps =new int;
int * pq = ps;
delete pq;
名称空间
使用多个库时，也可能导致名重复。
为了解决名称空间问题，C++引入了名称空间。
名称空间可以是全局的，也可以位于另一个名称空间中，但不能位于代码块中
除了用户定义的名称空间外，还存在全局名称空间，它是文件级声明区域。全局变量就在全局名称空间中
可以把现在的变量或函数等元素，加入到已有的命名空间中，还可以在多个文件中声明该名称空间，然后在一个文件中声明函数原型，然后在另一个文件中定义该函数
用法：
namespace jack {
int age;
int height;
void fetch();
}
using jack::height //将height添加到全局变量
//using namespace jack // 将jack命名空间的所有内容添加到全局变量
int main() {
using jack::age; // 将age添加到本地变量
age = 6;
return 0;
}