

OOP讲稿

- 配置篇

for sublime text in OS X

1. 新建一个build-system
2. 编写配置文件如下
3. 保存，可以保存名字为C++11.sublime-build,保存在默认目录下。

由于配置文件里面已经通过select指定了文件类型，因此在Build System还是选择Automatic,当我们通过CMD + B，Sublime会使用我们保存的C++11.sublime-build配置进行build，非常方便，可以通过CMD + shift + B直接运行程序。

- Template

模板分为函数模板和类模板

函数模板可以描述一系列仅有参数类型或返回值不同的函数。

类模板则可以描述一系列仅有数据类型和成员函数的参数类型或返回值不同的类

1. 第一行的 `template<typename T>` 告诉编译器接下来要定义的函数是一个模板,并设定了模板参数。 `typename T` 说明 T 在定义中代表一个类型。

函数是比价两个T的大小

1. 键字class表示其后面的是类型参数。在本例中numtype就是一个类型参数名。这个名字是可以任意取的，只要是合法的标识符即可。这里取numtype只是表示“数据类型”的意思而已。
2. 在建立类对象时，如果将实际类型指定为int型，编译系统就会用int取代所有的numtype，如果指定为float型，就用float取代所有的numtype。这样就能实现“一类多用”。（面向对象的思想）

- STL

map

映射和多重映射基于某一类型Key的键集的存在，提供对T类型的数据进

行快速和高效的检索。对map而言，键只是指存储在容器中的某一成员。Map不支持副本键，multimap支持副本键。Map和multimap对象包涵了键和各个键有关的值，键和值的数据类型是不相同的，这与set不同。set中的key和value是Key类型的，而map中的key和value是一个pair结构中的两个分量。

example I

map对象是模板类，需要关键字和存储对象两个模板参数：

定义了一个用string作为索引,并拥有相关联的指向int的指针.

example II

map可以通过pair来进行嵌套使用。这样建立起来的映射关系十分清晰，有着强大的使用价值，并且体现了我们面向对象的思想。

使用typedef为现有类型创建同义字，定义易于记忆的类型名

homework说明

GraphNode.first:图

GraphNode.second.first:权值

GraphNode.second.second:hash

- smart point

普通指针

如果不恰当处理指针就会带来许多问题，所以人们总是避免使用它。这也是许多新手程序员不喜欢指针的原因。指针总是会扯上很多问题，例如指针所指向对象的生命周期，挂起引用以及内存泄露。

智能指针是一个RAII（Resource Acquisition is initialization）类模型，用来动态的分配内存。它提供所有普通指针提供的接口，却很少发生异常。在构造中，它分配内存，当离开作用域时，它会自动释放已分配的内存。这样的话，程序员就从手动管理动态内存的繁杂任务中解放出来了

auto_ptr

example I

上述代码会智能地释放与指针绑定的内存。作用的过程是这样的：我们申请了一块内存来放Test对象，并且把他绑定到auto_ptr p上。所以当p离开作用域时，它所指向的内存块也会被自动释放。

example II

所有权 (ownership)问题。当我在函数间传递auto_ptr时，这就是一个问题。话说，我在Foo()中有一个auto_ptr，然后在Foo()中我把指针传递给了Fun()函数，当Fun()函数执行完毕时，指针的所有权不会再返还给Foo()。

由于auto_ptr的野指针行为，上面的代码导致程序崩溃。在这期间发生了这些细节，p拥有一块内存，当Fun调用时，p把关联的内存块的所有权传给了auto_ptr p1, p1是p的copy（注：这里从Fun函数的定义式看出，函数参数时值传递，所以把p的值拷进了函数中），这时p1就拥有了之前p拥有的内存块。目前为止，一切安好。现在Fun函数执行完了，p1离开了作用域，所以p1关联的内存块也就释放了。那么p呢？p什么都没了，这就是crash的原因了，因为下一行代码还试图访问p。

example III

auto_ptr不能指向一组对象，就是说它不能和操作符new[]一起使用

上面的代码将产生一个运行时错误。因为当auto_ptr离开作用域时，delete被默认用来释放关联的内存空间。当auto_ptr只指向一个对象时，这当然是没问题的，但是在上面的代码里，我们在堆里创建了一组对象，应该使用delete[]来释放，而不是delete。

share_ptr

shared_ptr是一种智能指针（smart pointer）。shared_ptr的作用有如同指针，但会记录有多少个shared_ptr共同指向一个对象(即引用计数).shared_ptr最初实现于Boost库中，后来被C++标准委员会收录于TR1技术报告中，成为C++11标准的一部分。shared_ptr与scoped_ptr一样包装了new操作符在堆上分配的动态对象，但它实现的是引用计数型的智能指针，可以被自由地拷贝和赋值，在任意的地方共享它，当没有代码使用（引用计数为0）它时才删除

被包装的动态分配的对象。shared_ptr也可以安全地放到标准容器中，并弥补了auto_ptr因为转移语义而不能把指针作为STL容器元素的缺陷。

example 1

在此场景下，shared_ptr指向一组对象，但是当离开作用域时，默认的析构函数调用delete释放资源。实际上，我们应该调用delete[]来销毁这个数组。用户可以通过调用一个函数，例如一个lamda表达式，来指定一个通用的释放步骤。

interface

get(): 获取shared_ptr绑定的资源.

reset(): 释放关联内存块的所有权，如果是最后一个指向该资源的shared_ptr, 就释放这块内存。

unique: 判断是否是唯一指向当前内存的shared_ptr.

operator bool : 判断当前的shared_ptr是否指向一个内存块，可以用if 表达式判断。

- caffe

Blob类

Blob类存在namespace caffe 中, Blob是作为Caffe中数据流通的一个基本类，网络各层之间的数据是通过Blob来传递的。

类定义时使用了类模板

DISABLE_COPY_AND_ASSIGN(Blob) 为一个宏定义, 用来禁止一个类拷贝构造

在定义成员时，Blob使用了大量的share_ptr

- 参考网站

<http://blog.csdn.net/zhiganglet/article/details/38964531>

<http://blog.csdn.net/oqqquzi1234567/article/details/43489291>

<http://blog.csdn.net/yockie/article/details/6980692>

<http://baike.baidu.com/link?>

[url=xNO2KBZVpSvlgD91qTp5l5FqjfzdS959FdVqk0kDq7eAdzgo4r-2FbJoDU6Ksc7ksoKbgjEWQtSRBo3VQpt08K](http://baike.baidu.com/link?url=xNO2KBZVpSvlgD91qTp5l5FqjfzdS959FdVqk0kDq7eAdzgo4r-2FbJoDU6Ksc7ksoKbgjEWQtSRBo3VQpt08K)

<https://blog.blink.moe/2016/06/03/caffe1/?nsukey=z3tCMFCAWp3EaZMj%2Bnk%2F9mvP7mlvMD734javJFbxhDz8YgirsbOoYbzeeaShGrgRSc%2BX4QN1oVihbki4%2FLubPDP8JBOY0gw9ZcVK157%2FhJJfJKFdN5FwwMDpQkgYtjl2%2F0VOgjVNIN%2FK0deZrOPTAzswu34XMZFIfI4BnXsfo%2BkUUIYj17EsWrxqocOCC>

<http://www.jianshu.com/p/e4919f1c3a28>