STL 设计之 EBO(空基类优化)

发布于2019-10-24 20:37:21 阅读 817

STL 设计之 EBO(空基类优化)

0.导语

EBO 简称 Empty Base Optimization。

本节从空类开始，到 STL 内部，到测试，再到我们自己实现一个 EBO，对比性能，最后再测试，总结。

1.空类

定义一个空类：没有成员变量，没有继承，没有数据元素的类。

```
class Empty{
public:
    void print() {
        std::cout<<"I am Empty class"<<std::endl;
    }
};
```

由于它是空的，所以这个 sizeof 是多少呢?

```
std::cout<<sizeof(Empty)<<std::endl; //1
```

结果是 1，**它是空的怎么不是 0 呢?**

因为空类同样可以被实例化，每个实例在内存中都有一个独一无二的地址，为了达到这个目的，编译器往往会给一个空类隐含的加一个字节，这样空类在实例化后在内存得到了独一无二的地址．所以上述大小为 1.

根据上面的回答，估计大家或引出另一个问题：**为什么两个不同对象的地址应该不同?**

现在有下面这个例子：

```
class Empty{
public:
    void print() {
        std::cout<<"I am Empty class"<<std::endl;
    }
};
template < typename T >
bool isSame( T const & t1, T const & t2 )
{
    return &t1 == &t2;
}
```

我们来测试一下：

```
int main() {
    Empty a,b;
    assert(!isSame(a,b));  // 编译通过，a与b的地址不同

    Empty *p=new Empty;
    Empty *q=new Empty;
    assert(!isSame(p,q)); // 编译通过，a与b的地址不同
    return 0;
}
```

上面测试了，两个不同对象地址是不同的，考虑下面场景：

```
Empty a,b;
// 在同一地址具有两个对象将意味着在使用指针引用它们时将无法区分这两个对象。
Empty *p1=&a;
p1->print();
```

此时会发现，如果 a 的地址与 b 的地址一样，那么在同一地址具有两个对象将意味着在使用指针引用它们时将无法区分这两个对象。因此两个不同对象的地址不同。

2.空基类优化

现在对比一下下面两个用法，第一种，一个类中包含了两一个类作为成员，然后通过这个来获得被包含类的功能。

```
class notEbo  {
    int i;
    Empty e;
    // do other things
};
```

另一种直接采用继承的方式来获得基类的成员函数及其他功能等等。

```
class ebo:public Empty {
    int i;
    // do other things
};
```

接下来做个测试：

```
std::cout<<sizeof(notEbo)<<std::endl;
std::cout<<sizeof(ebo)<<std::endl;
```

输出：

```
8
4
```

第一种，会因为字节对齐，将其原来只占 1 字节，进行扩充到 4 的倍数，最后就是 8 字节。

对比这两个发现，第二种通过继承方式来获得基类的功能，**并没有产生额外大小的优化称之为 EBO(空基类优化)。**

接下来，我们回到 STL 源码中，看看其中的使用!

3.STL 中的 EBO 世界

不管是 deque、rb_tree、list 等容器，都离不开内存管理，在这几个容器中都包含了相应的内存管理，并通过 `_xx_impl` 来继承下面这几个类：

```
std::allocator<_Tp>
__gnu_cxx::bitmap_allocator<_Tp>
__gnu_cxx::bitmap_allocator<_Tp>
__gnu_cxx::__mt_alloc<_Tp>
__gnu_cxx::__pool_alloc<_Tp>
__gnu_cxx::malloc_allocator<_Tp>
```

那这和我们的 EBO 有啥关系呢?

实际上，上面所列出继承的基类都是内存管理的 EBO(空基类)。

在每个容器中的使用都是调用每个内存管理的 `rebind<_Tp>::other` 。

例如红黑树源码结构：

```
typedef typename __gnu_cxx::__alloc_traits<_Alloc>::template
       rebind<_Rb_tree_node<_Val> >::other _Node_allocator;
struct _Rb_tree_impl : public _Node_allocator
{
// do somethings
};
```

接下来我们看上面列出的内存管理类里面的源码结构：这里拿 `allocator` 举例：

```
template<typename _Tp>
class allocator: public __allocator_base<_Tp>
{
     template<typename _Tp1>
     struct rebind { typedef allocator<_Tp1> other; };
};
```

看到了没，通过 `rebind<_Tp>::other` 来获得传递进来的内存分配器，也就是前面提到的这些。

```
std::allocator<_Tp>
__gnu_cxx::bitmap_allocator<_Tp>
__gnu_cxx::bitmap_allocator<_Tp>
__gnu_cxx::__mt_alloc<_Tp>
__gnu_cxx::__pool_alloc<_Tp>
__gnu_cxx::malloc_allocator<_Tp>
```

搞懂了这些，来测试一波：

```
void print() {
    cout<<sizeof(std::allocator<int>)<<" "<<sizeof(std::allocator<int>::rebind<int>::othe
    cout<<sizeof(__gnu_cxx::bitmap_allocator<int>)<<" "<<sizeof(__gnu_cxx::bitmap_allocat
    cout<<sizeof(__gnu_cxx::new_allocator<int>)<<" "<<sizeof(__gnu_cxx::new_allocator<int
    cout<<sizeof(__gnu_cxx::__mt_alloc<int>)<<" "<<sizeof(__gnu_cxx::__mt_alloc<int>::reb
    cout<<sizeof(__gnu_cxx::__pool_alloc<int>)<<" "<<sizeof(__gnu_cxx::__pool_alloc<int>
    cout<<sizeof(__gnu_cxx::malloc_allocator<int>)<<" "<<sizeof(__gnu_cxx::malloc_allocat
}
```

我们来测试这些 sizeof 是不是 1，经过测试输出如下：

```
1 1
1 1
1 1
1 1
1 1
1 1
```

说明内存管理的实现就是通过采用继承的方式，使用空基类优化，来达到尽量降低容器所占的大小。

展开阅读全文

C++ 面向对象编程 测试服务 WeTest 容器 举报

点赞 2 分享

0 条评论

作者介绍

公众号guangcity

关注 专栏

文章 阅读量 获赞 作者排名

379 170.1K 927 711

目录

相关文章

快速阅读源码的能力培养

基于《C++那些事》交流群大众反馈，同时针对自己学习过程中的一些问题，做一些记录性的分享。

公众号guangcity

面试中经常被问到的 OpenGL ES 对象，你知道的有哪些?

VBO（Vertex Buffer Object）是指顶点缓冲区对象，而 EBO（Element Buffer Object）是指图元索引缓冲区对象，VBO和...

字节流动

OpenGL ES 对象

VBO（Vertex Buffer Object）是指顶点缓冲区对象，而 EBO（Element Buffer Object）是指图元索引缓冲区对象，VAO

ruochen

熟悉 OpenGL VAO、VBO、FBO、PBO 等对象，看这一篇就够了

VBO（Vertex Buffer Object）是指顶点缓冲区对象，而 EBO（Element Buffer Object）是指图元索引缓冲区对象，VAO 和...

字节流动

OpenGL学习笔记 （二）- 顶点与绘制指令

前一篇文章（OpenGL学习笔记 （一）- 综述、渲染管线）提到过，现代OpenGL不再推荐使用显示列表或者更古老的glVertex了。这篇笔记将详细探讨这个话...

KAAAsS

NDK OpenGL ES 3.0 开发（二十一）：3D 模型加载和渲染

上一节简单介绍了常用的 3D 模型文件 Obj 的数据结构和模型加载库 Assimp 的编译，本节主要介绍如何使用 Assimp 加载 3D 模型文件和渲染 3...

字节流动

OpenGL ES 3D 模型的加载和渲染

上一节简单介绍了常用的 3D 模型文件 Obj 的数据结构和模型加载库 Assimp 的编译，本节主要介绍如何使用 Assimp 加载 3D 模型文件和渲染 3...

字节流动

STL学习笔记（1）STL 概述

长久以来，软件界一直希望建立一种可重复利用的东西，以及一种得以制造出"可重复运用的东西"的方法,让程序 员的心血不止于随时间的迁移，人事异动而烟消云散，从函数(...

轻舞飞扬SR

从零开始学C++之STL（一）：STL六大组件简介

一、STL简介 （一）、泛型程序设计 泛型编程（generic programming） 将程序写得尽可能通用 将算法从数据结构中抽象出来，成为通用的 C...

s1mba

C++ STL编程轻松入门基础

C++ STL编程轻松入门基础 1 初识STL：解答一些疑问 1.1 一个最关心的问题：什么是STL 1.2 追根溯源：STL的历史 1.3 千丝万缕的联系 ...

猿人谷

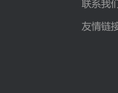
C++基础 STL简介

STL（Standard TemplateLibrary），即标准模板库，从根本上说，STL是一些"容器"的集合，这些"容器"有list、vector、set、...

xxpcb

揭秘3D打印的工业革命：你须知3D打印知识

什么是3D打印？ 3D打印，即快速成型技术的一种，它是一种以数字模型文件为基础，运用粉末状金属或塑料等可粘合材料，通过逐层打印的方式来构造物体的技术。 3D打印...



机器人网

华人团队打造：AutoML + GAN = AutoGAN！AI设计GAN模型比人类…

生成对抗网络（GAN）自其诞生以来一直盛行。它的一个最显著的成功在于是用各种各样的卷积结构生成逼真的自然图像。

新智元

记武汉2016年第一期学习力提升工作坊——MVP验证篇工作坊总体设计

当开始有了第一课的时候，剩下的课程我还没有准备好。只通过一些粗浅的想法形成了课程内容，主要根据学员的第一次的反馈来规划下一次课程的内容。并根据反馈对相应的课程进...

顾宇

OpenGL现代编程第二课——第一个多边形

https://learnopengl-cn.github.io/01%20Getting%20started/04%20Hello%20Triangle/

用户5908113

【笔记】C++标准库: 体系结构与内核分析(上)

这篇是这段时间看的侯捷关于C++标准模板库的课程《C++标准库: 体系结构与内核分析》的笔记, 课程内容大家自己找吧. 这个课程质量很高, 除了介绍STL的基础...

ZifengHuang

更多文章 >