

congqingbin的专栏

目录视图

摘要视图

RSS 订阅

个人资料



nicGithub

访问：72292次

积分：930

等级：BLOG > 3

排名：千里之外

原创：17篇 转载：36篇

译文：0篇 评论：25条

[博客Markdown编辑器上线啦](#) [那些年我们追过的Wrox精品红皮计算机图书](#) [PMBOK第五版精讲视频教程](#) [火星人敏捷开发1001问](#)

深入分析C++引用

分类：c / c ++

2013-10-24 11:18

988人阅读

评论(1)

收藏

举报

c++

指针

引用

原文地址：<http://blog.csdn.net/webscaler/article/details/6577429>

关于引用和指针的区别的文章很多很多，但是总是找不到他们的根本区别，偶然在codeproject上看到这篇文章，觉得讲的挺好的，

文章搜索

文章分类

[Java](#) (22)[c / c ++](#) (7)[多线程与锁](#) (8)[设计模式](#) (3)[android 门外](#) (12)[android 跨门槛](#) (3)[ubuntu常用命令](#) (2)[算法](#) (1)[开发工具](#) (2)[语言之争](#) (2)[android入门 xml](#) (0)[java JNI](#) (1)[网络编程](#) (3)[upnp](#) (2)[ubuntu](#) (2)

文章存档

[2014年12月](#) (1)[2014年11月](#) (1)[2014年10月](#) (1)[2014年09月](#) (1)[2014年06月](#) (2)[展开](#)

所以翻译了下，希望对大家有帮助。

原文地址: http://www.codeproject.com/KB/cpp/References_in_c__.aspx

引言

我选择写 C++ 中的引用是因为我感觉大多数人误解了引用。而我之所以有这个感受是因为我主持过很多 C++ 的面试，并且我很少从面试者中得到关于 C++ 引用的正确答案。

那么 c++ 中引用到底意味这什么呢？通常一个引用让人想到是一个引用的变量的别名，而我讨厌将 c++ 中引用定义为变量的别名。这篇文章中，我将尽量解释清楚，c++ 中根本就没有什么叫做别名的东东。

背景

在 c/c++ 中，访问一个变量只能通过两种方式被访问，传递，或者查询。这两种方式是：

1. 通过值 访问 / 传递变量
2. 通过地址 访问 / 传递变量 – 这种方法就是指针

除此之外没有第三种访问和传递变量值的方法。引用变量也就是个指针变量，它也拥有内存空间。最关键的是引用是一种会被编译器自动解引用的指针。很难相信么？让我们来看看吧。。。

下面是一段使用引用的简单 c++ 代码

阅读排行

- java枚举类Enum方法简 (15024)
- RotateAnimation类：旋 (11268)
- android获取设备屏幕大 (9059)
- declare-styleable：自定 (8009)
- android中的LaunchMod (2608)
- ubuntu的ps -aux详细介绍 (1950)
- java 多线程基础--各种状 (1785)
- java wait()和sleep()方法 (1548)
- 关于MessageQuene, 关 (1472)
- Ubuntu 及windows 环境 (1462)

评论排行

- RotateAnimation类：旋 (9)
- declare-styleable：自定 (8)
- 关于MessageQuene, 关 (2)
- android中的LaunchMod (1)
- 可重入锁 ReentrantLock (1)
- 深入分析C++引用 (1)
- C中的静态存储区和动态 (1)
- 为什么基类的析构函数是 (1)
- android获取设备屏幕大 (1)
- 用easybcd在win7安装ut (0)

[html]

```
01. <pre name="code" class="cpp">[cpp] view plaincopyprint?
02. #include <iostream.h>
03. int main()
04. {
05.     int i = 10;    // A simple integer variable
06.     int &j = i;    // A Reference to the variable i
07.     j++;    // Incrementing j will increment both i and j.
08.     // check by printing values of i and j
09.     cout<< i << j <<endl; // should print 11 11
10.     // Now try to print the address of both variables i and j
11.     cout<< &i << &j <<endl;
12.     // surprisingly both print the same address and make us feel that they are
13.     // alias to the same memory location.
14.     // In example below we will see what is the reality
15.     return 0;
16. } </pre><br>
```

引用其实就是 c++ 中的常量指针。表达式 `int &i = j;` 将会被编译器转化成 `int *const i = &j;` 而引用之所以要初始化是因为 `const` 类型变量必须初始化，这个指针也必须有所指。下面我们再次聚焦到上面这段代码，并使用编译器的那套语法将引用替换掉。

[cpp]

```
01. [cpp] view plaincopyprint?
02. #include <iostream.h>
03. int main()
```

推荐文章

- * [纯CSS实现表单验证](#)
- * [段落文字彩条效果](#)
- * [Kepler性能分析之M2E调优](#)
- * [公共技术点之 Java反射 Reflection](#)
- * [QtAndroid详解\(2\): startActivity 和它的小伙伴们](#)
- * [Qt5官方demo解析集34——Concentric Circles Example](#)

最新评论

[RotateAnimation类: 旋转变化](#)
paddy: @wen944936:用 image.startAnimation(animation); 这个方法就可...

[深入分析C++引用](#)
莫利斯安: 博主你好, 我想咨询下博文中这句话. cout << &j << endl; 的语句, 编译器就会将其转...

[declare-styleable: 自定义控件](#)
eieihihi: 收藏收藏!!!

[android中的LaunchMode详解](#)
Alicedetears: 讲解的太详细, 非常感谢

[为什么基类的析构函数是虚函数](#)
有点文化的小流氓: 终于懂了~

[C中的静态存储区和动态存储区](#)
try15757125554: 学习了 很有帮助

```

04. {
05.     int i = 10;           // A simple integer variable
06.     int *const j = &i;    // A Reference to the variable i
07.     (*j)++;              // Incrementing j. Since reference variables are
08.                           // automatically dereferenced by compiler
09.     // check by printing values of i and j
10.     cout<< i << *j <<endl; // should print 11 11
11.     // A * is appended before j because it used to be reference variable
12.     // and it should get automatically dereferenced.
13.     return 0;
14. }
```

读者一定很奇怪为什么我上面这段代码会跳过打印地址这步。这里需要一些解释。因为引用变量时会被编译器自动解引用的, 那么一个诸如 cout << &j << endl; 的语句, 编译器就会将其转化成语句 cout << *j << endl; 现在 &* 会相互抵消, 这句话变的毫无意义, 而 cout 打印的 j 值就是 i 的地址, 因为其定义语句为 int *const j = &i;

所以语句 cout << &i << &j << endl; 变成了 cout << &i << *j << endl; 这两种情况都是打印输出 i 的地址。这就是当我们打印普通变量和引用变量的时候会输出相同地址的原因。

下面给出一段复杂一些的代码, 来看看引用在级联 (cascading) 中是如何运作的。

```

[cpp]
01. [cpp] view plaincopyprint?
02. #include <iostream.h>
```

可重入锁 ReentrantLock 源码解
memoryisking: 关于更多
ReentrantLock的内容可以去看
这里:

RotateAnimation类: 旋转变化云
wen944936: 放在oncreat里面一
起写了就可以动, 加到onclick里
面就不能动了

RotateAnimation类: 旋转变化云
代号evan: image= (ImageView)
findViewById(R.id.imageView1);
...

RotateAnimation类: 旋转变化云
代号evan: 没动静啊啊

```
03. int main()
04. {
05.     int i = 10; // A Simple Integer variable
06.     int &j = i; // A Reference to the variable
07.     // Now we can also create a reference to reference variable.
08.     int &k = j; // A reference to a reference variable
09.     // Similarly we can also create another reference to the reference variable k
10.     int &l = k; // A reference to a reference to a reference variable.
11.     // Now if we increment any one of them the effect will be visible on all the
12.     // variables.
13.     // First print original values
14.     // The print should be 10,10,10,10
15.     cout<< i << "," << j << "," << k << "," << l <<endl;
16.     // increment variable j
17.     j++;
18.     // The print should be 11,11,11,11
19.     cout<< i << "," << j << "," << k << "," << l <<endl;
20.     // increment variable k
21.     k++;
22.     // The print should be 12,12,12,12
23.     cout<< i << "," << j << "," << k << "," << l <<endl;
24.     // increment variable l
25.     l++;
26.     // The print should be 13,13,13,13
27.     cout<< i << "," << j << "," << k << "," << l <<endl;
28.     return 0;
29. }
```

下面这段代码是将上面代码中的引用替换之后代码, 也就是说明我们不依赖编译器的自动替换功能, 手动进行替换也能达到相同的目标。

[cpp]

```
01. [cpp] view plaincopyprint?
02. #include <iostream.h>
03. int main()
04. {
05.     int i = 10;           // A Simple Integer variable
06.     int *const j = &i;    // A Reference to the variable
07.     // The variable j will hold the address of i
08.     // Now we can also create a reference to reference variable.
09.     int *const k = &*j;   // A reference to a reference variable
10.     // The variable k will also hold the address of i because j
11.     // is a reference variable and
12.     // it gets auto dereferenced. After & and * cancels each other
13.     // k will hold the value of
14.     // j which is nothing but address of i
15.     // Similarly we can also create another reference to the reference variable k
16.     int *const l = &*k;   // A reference to a reference to a reference variable.
17.     // The variable l will also hold address of i because k holds address of i after
18.     // & and * cancels each other.
19.     // so we have seen that all the reference variable will actually hold the same
20.     // variable address.
21.     // Now if we increment any one of them the effect will be visible on all the
22.     // variables.
23.     // First print original values. The reference variables will have * prefixed because
24.     // these variables get automatically dereferenced.
25.     // The print should be 10,10,10,10
26.     cout<< i << ", " << *j << ", " << *k << ", " << *l <<endl;
27.     // increment variable j
28.     (*j)++;
29.     // The print should be 11,11,11,11
30.     cout<< i << ", " << *j << ", " << *k << ", " << *l <<endl;
31.     // increment variable k
32.     (*k)++;
33.     // The print should be 12,12,12,12
34.     cout<< i << ", " << *j << ", " << *k << ", " << *l <<endl;
35.     // increment variable l
```

```
36.     (*l)++;  
37.     // The print should be 13,13,13,13  
38.     cout << i << "," << *j << "," << *k << "," << *l << endl;  
39.     return 0;  
40. }
```

我们通过下面代码可以证明 c++ 的引用不是神马别名，它也会占用内存空间的。

```
[cpp]  
01. [cpp] view plaincopyprint?  
02. #include <iostream.h>  
03. class Test  
04. {  
05.     int &i;    // int *const i;  
06.     int &j;    // int *const j;  
07.     int &k;    // int *const k;  
08. };  
09. int main()  
10. {  
11.     // This will print 12 i.e. size of 3 pointers  
12.     cout<< "size of class Test = " << sizeof(class Test) << endl;  
13.     return 0;  
14. }
```

结论

我希望这篇文章能把 c++ 引用的所有东东都解释清楚，然而我要指出的是 c++ 标准并没有解释编译器如何实现引用的行为。所以实现取决于编译器，而大多数情况下就是将其实现为一个 const 指针。

引用支持 c++ 虚函数机制的代码

```
[cpp]
01. [cpp] view plaincopyprint?
02. #include <iostream.h>
03. class A
04. {
05. public:
06.     virtual void print() { cout<<"A.."<<endl; }
07. };
08. class B : public A
09. {
10. public:
11.     virtual void print() { cout<<"B.."<<endl; }
12. };
13.
14. class C : public B
15. {
16. public:
17.     virtual void print() { cout<<"C.."<<endl; }
18. };
19. int main()
20. {
```



```
21.         C c1;  
22.         A &a1 = c1;  
23.         a1.print(); // prints C  
24.         A a2 = c1;  
25.         a2.print(); // prints A  
26.         return 0;  
27.     }
```

上述代码使用引用支持虚函数机制。如果引用仅仅是一个别名，那如何实现虚函数机制，而虚函数机制所需要的动态信息只能通过指针才能实现，所以更加说明引用其实就是一个 const 指针。

[上一篇](#) [匿名内部类 参数用final修饰](#)

[下一篇](#) [C++ 虚函数表解析](#)

主题推荐

[c++](#)[cascading](#)[编译器](#)[内存](#)[面试](#)

猜你在找

[C++学习笔记31指向引用的指针3](#)[初识DLL及API](#)[nginx脚本引擎与变量设计一](#)[TCP标志位](#)[ImageView属性详解](#)[malloc与calloc的区别](#)[让CPU占用率曲线听你指挥zt](#)[GZIP文件格式简介](#)[Linux下端口复用SO_REUSEADDR与SO_REUSEPORT](#)[Learn Python The Hard Way学习42 - 继承包含对象和类](#)

准备好了么？跳吧

更多职位尽在 **CSDN JOB**

c++开发工程师

我要跳槽

C++服务器

我要跳槽

京品高科信息科技（北京）有限公司

| 7-15K/月

北京乐动卓越科技有限公司

| 15-20K/月

C++软件开发工程师

我要跳槽

软件工程师（C++/C#）

我要跳槽

天津市努思企业服务有限公司

| 7-10K/月

武汉海翼科技有限公司

| 4-6K/月



More Effective C++：35个改善
编程与设计的有效方法（中文

¥ 47.2 立即购买



Effective C++：改善程序与设计的
55个具体做法（第三版）（评

¥ 62 立即购买

查看评论

1楼 莫利斯安 2014-12-05 23:08发表



博主你好，我想咨询下博文中这句话。

cout << &j << endl; 的语句，编译器就会将其转化成语句

cout << &*j << endl; 这里的转化不等价嘛？我在VS2013中输出了一下发现两个是不一样的。想请教一下是不是我理解出问题了？

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity

[Splashtop](#) [UML](#) [components](#) [Windows Mobile](#) [Rails](#) [QEMU](#) [KDE](#) [Cassandra](#) [CloudStack](#)
[FTC](#) [coremail](#) [OPhone](#) [CouchBase](#) [云计算](#) [iOS6](#) [Rackspace](#) [Web App](#) [SpringSide](#) [Maemo](#)
[Compuware](#) [大数据](#) [aptech](#) [Perl](#) [Tornado](#) [Ruby](#) [Hibernate](#) [ThinkPHP](#) [HBase](#) [Pure](#) [Solr](#)
[Angular](#) [Cloud Foundry](#) [Redis](#) [Scala](#) [Django](#) [Bootstrap](#)

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [银行汇款帐号](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#) | [合作伙伴](#) | [论坛反馈](#)

[网站客服](#) [杂志客服](#) [微博客服](#) webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持

京 ICP 证 070598 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved 