c++右值引用以及使用

前几天看了一篇文章《4行代码看看右值引用》 觉得写得不错，但是觉得右值引用的内容还有很多可以去挖掘学习，所以总结了一下，希望能对右值引用有一个更加深层次的认识

一、几个基本概念

1.1左值和右值

左值和右值的区分标准在于能否获取地址。

最早的c++中，左值的定义表示的是可以获取地址的表达式，它能出现在赋值语句的左边，对该表达式进行赋值。但是修饰符const的出现使得可以声明如下的标识符，它可以取得地址，但是没办法对其进行赋值：

```
const int& i = 10;
```

右值表示无法获取地址的对象，有常量值、函数返回值、Lambda表达式等。无法获取地址，但不表示其不可改变，当定义了右值的右值引用时就可以更改右值。

1.2 左值引用和右值引用

传统的c++引用被称为左值引用，用法如下：

```
int i = 10;
int & ii = I;
```

C++ Primer Plus 第6版18.1.9中说到，c++11中增加了右值引用，右值引用关联到右值时，右值被存储到特定位置，右值引用指向该特定位置，也就是说，右值虽然无法获取地址，但是右值引用是可以获取地址的，该地址表示临时对象的存储位置。语法如下：

```
int && iii = 10;
```

1.3 左值引用和右值引用的汇编代码

以下汇编都是x86汇编

写一段简单的语句，看其汇编

```
int i = 1;
int & ii = i;
```

```
0x080483f3  movl   $0x1,-0x10(%ebp)
0x080483fa  lea    -0x10(%ebp),%eax
0x080483fd  mov    %eax,-0x8(%ebp)
```

第一句是将1赋值给i，第二句将i的地址放入eax中，第三句将eax中的值传给ii。可见引用就是从一个变量处取得变量的地址，然后赋值给引用变量。

昵称：番茄汁汁
园龄：2年11个月
粉丝：13
关注：0
+加关注

< 2020年6月 >

日	一	二	三	四	五	六
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4
5	6	7	8	9	10	11

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

LeetCode(42)
算法设计与分析课程(10)
算法(6)
c++(4)
linux内核(3)
SELinux(2)
动态规划(2)
矩阵链乘法(1)
Volatile(1)
操作系统(1)
更多

随笔分类

algorithm(6)
angr(1)
C++(5)
CTF—pwn(1)
ELF(3)

再看一句右值引用的汇编

```
int && iii = 10;
```

```
0x08048400  mov    $0xa,%eax
0x08048405  mov    %eax,-0xc(%ebp)
0x08048408  lea    -0xc(%ebp),%eax
0x0804840b  mov    %eax,-0x4(%ebp)
```

第一句将10赋值给eax，第二句将eax放入-0xc(%ebp)处，前面说到"临时变量会引用关联到右值时，右值被存储到特定位置"，在这段程序中，-0xc(%ebp)便是该临时变量的地址，后两句通过eax将该地址存到iii处。

通过上述代码，我们还可以发现，在上述的程序中-0x4(%ebp)存放着右值引用iii，-0x8(%ebp)存放着左值引用，-0xc(%ebp)存放着10，而-0x10(%ebp)存放着1，左值引用和右值引用同int一样是四个字节（因为都是地址）

同时，我们可以深入理解下临时变量，在本程序中，有名字的1（名字为i）和没有名字的10（临时变量）的值实际是按同一方式处理的，也就是说，临时变量根本上来说就是一个没有名字的变量而已。它的生命周期和函数栈帧是一致的。也可以说临时变量和它的引用具有相同的生命周期。

1.4 const左值引用

如果写如下代码，定义一个左值引用，将其值置为一个常量值，则会报错：

```
int & i = 10;
```

原因很明显，左边是一个左值引用，而右边是一个右值，无法将左值引用绑定到一个右值上。

但是如果是一个const的左值引用，是可以绑定到右值上的。即如下写法是符合语法规范的：

```
const int & i = 10;
```

这段程序的汇编代码如下：

```
0x08048583  mov    $0xa,%eax
0x08048588  mov    %eax,-0x8(%ebp)
0x0804858b  lea    -0x8(%ebp),%eax
0x0804858e  mov    %eax,-0x4(%ebp)
```

易知-0x4(%ebp)处存放着i，-0x8(%ebp)处则存放着临时对象10，程序将10的地址存放到了i处。看到这里会发现const引用在绑定右值时和右值引用并没有什么区别。

1.5 左值引用和右值引用的相互赋值

能将右值引用赋值给左值引用，该左值引用绑定到右值引用指向的对象，在早期的c++中，引用没有左右之分，引入了右值引用之后才被称为左值引用，所以说左值引用其实可以绑定任何对象。这样也就能理解为什么const左值引用能赋予常量值。

```
int&& iii = 10;
int& ii = iii;       //ii等于10，对ii的改变同样会作用到iii
```

二、右值引用和移动语义

在旧的c++中，出现了很多的不必须要的拷贝，因为在某些情况下，对象拷贝完之后就下来就销毁了。新标准引入了移动操作，减少了很多的复制操作，而右值引用正式为了支持移动操作而引入的新的引用类型。

随笔档案

最新评论

1. Re:c++右值引用以及使用

楼主转发的例子，为啥要t1和t2反过来写呢，感觉增加了理解难度啊

--专给楼主点赞

2. Re:USBIP源码分析

@林清流 我都忘了，怕是帮不了你...

--番茄汁汁

3. Re:USBIP源码分析

2.1 标准库move函数

根据右值引用的语法规则可知，不能将右值引用绑定到一个左值上，c++11引入右值引用，并且提供了move函数，用来获得绑定到左值上的右值引用，此函数定义在头文件utility中。

```
Int &&iii = move(ii)
```

调用move之后，必须保证除了对ii复制或销毁它外，我们将不再使用它，在调用move之后，我们不能对移动源后对象做任何假设。

2.2 模板实参推断和引用

为了理解move函数的实现，首先需要理解模板实参推断和引用。

当左值引用作为参数时，看几个例子：

```
template<class T> void f1(T&) {}
f1(i)            //i是一个int，模板参数类型T是int
f1(ci)          //ci是一个const int，模板参数T是const int
f1(5)            //错误：传递给一个&参数的实参必须是一个左值
```

如果函数的参数是const的引用时：

```
template<class T> void f2(const T&) {}
f2(i)            //i是一个int，模板参数类型T是int，因为非const可以转化为const
f2(ci)          //ci是一个const int，模板参数T是int
f2(5)           //看前面，const的引用可以绑定右值，T是int
```

当参数是右值引用时，

```
template<class T> void f3(T&&) {}
f3(5)           // T是int
```

2.3 引用折叠和右值引用参数

按照道理来说，f3（i）是应该不正确的，因为无法将右值引用绑定到一个左值上，但是，c++中有两个正常绑定规则的例外，允许这种绑定。这两个例外规则是move正确工作的基础

例外1：右值引用的类型推断。当我们将一个左值传递给函数的右值引用作为参数时（函数参数为T&&），编译器推断模板类型参数为实参的左值引用类型，，因此，调用f3（i）时，T被推断为int&，而不是int。并且，模板函数中对参数的改变会反映到调用时传入的实参。

通常，我们不能直接定义一个引用的引用，但是同过类型别名（使用typedef）和模板间接定义是可以的。

例外2：引用折叠。当定义了引用的引用时，则这些引用形成了"折叠"，所有的情况下（除了一个例外），引用会折叠成一个普通的左值引用类型。这个例外就是右值引用的右值引用：

l X& &&、X& &&、X&& &都折叠成X&

l 类型X&& &&折叠成X&&

2.4 理解右值引用折叠和右值引用类型推断

对于函数f3而言，根据**右值引用类型推断规则**可以知道如下结果：

```
f3 (i)                  //实参是左值，模板参数T是int&
f3 (ci)                //实参是左值，模板参数T是一个const int&
```

要二次开发usbip，请问能否留下联系方式?

--林清流

4. Re:Unlink——2016 ZCTF note2解析

博主，此题源文件可以分享一个么602390443@qq.com

--WildBloom

5. Re:c++右值引用以及使用

学习了 感谢博主详细总结

--Jack_ff

阅读排行榜

1. c++右值引用以及使用(17248)
2. 从4行代码看右值引用(3275)
3. linux内核钩子--khook(2258)
4. USBIP源码分析(1221)
5. AddressSanitizer简介(1010)

评论排行榜

1. c++右值引用以及使用(5)
2. 南大算法设计与分析课程OJ答案代码（1）中位数附近2k+1个数、任意两数之和是否等于给定数(2)
3. USBIP源码分析(2)
4. Unlink——2016 ZCTF note2解析(1)

推荐排行榜

1. c++右值引用以及使用(2)
2. 南大算法设计与分析课程课后习题（1）(1)
3. 全排列问题全面解析(1)
4. CVE-2018-18955漏洞学习(1)
5. selinux学习(1)

但是当T被推断为int&时，函数f3会实例化成如下的样子：

```
void f3<int&> (int&  &&)
```

然后根据**右值引用折叠规则**可以知道，上述实例化方式应该被折叠成如下样子：

```
void f3<int&> (int&)
```

这两个规则导致了两个重要的结果：

l 如果一个函数参数是一个指向模板类型参数的右值引用，如T&&，则它能被绑定到一个左值，且

l 如果实参是一个左值，则推断出的模板实参类型将时一个左值引用，且函数参数被实例化为一个普通左值引用参数（T&）

值得注意，参数为T&&类型的函数可以接受所有类型的参数，左值右值均可。在前面，同样介绍过，const的左值引用做参数的函数同样也可以接受所有类型的参数。

2.5 当右值引用作为函数模板参数时

通过前面，我们了解到当右值引用作为函数模板参数时，类型T会被推断为一个引用类型。这一特性会影响模板函数内部的代码，看下面一段代码：

```
template<class T>
void f3(T&& val)
{
          T t = val;
          t = fcn(t);
          if(val == t){…}
}
```

假如以左值i来调用该函数，那么T被推断为int&，将t绑定到val之上，对t的更改就被应用到val，则if判断条件永远为true。

右值引用通常用于两种情况，模板转发其实参、模板被重载。下面都会介绍到

前面说到，const左值引用做参数和右值引用做参数一样，是可以匹配所有的参数类型，但当重载函数同时出现时，右值引用做参数的函数绑定非const右值，const左值引用做参数的函数绑定左值和const右值（非const右值就是通过右值引用来引用的右值，虽然无法获取右值的地址，但是可以通过定义右值引用来更改右值）：

```
Template<class T> void f(T&&)                              //绑定到非const右值
Template<class T> void f(const T&)                         //左值和const右值
```

2.6 move函数实现

vs2017中move函数的定义如下

```
using remove_reference_t = typename remove_reference<_Ty>::type;
template<class _Ty>
constexpr remove_reference_t<_Ty>&& move(_Ty&& _Arg) _NOEXCEPT
{
```

```
    return (static_cast<remove_reference_t<_Ty>&&>(_Arg));
}
template<class _Ty>
struct remove_reference
{    // remove reference
using type = _Ty;
};

template<class _Ty>
struct remove_reference<_Ty&>
{    // remove reference
    using type = _Ty;
};

template<class _Ty>
struct remove_reference<_Ty&&>
{    // remove rvalue reference
    using type = _Ty;
};
```

使用右值引用作为参数，前面说过，可以匹配所有类型。以下两种方式都是正确的：

```
string s1("s1"),s2
s2 = move(string("bye!"))         //_Ty推断为string
s2 = move(s1)                    //_Ty推断为string&
```

至于remove_reference就好理解了

综上，可以发现move函数不管传入什么类型参数，不管是左值还是右值，都会返回其右值引用类型

三、转发

某些函数需要将其中一个或多个实参连同类型不变地转发给其他函数，在这种情况下，我们需要保持被转发实参的所有性质，包括实参是否是const的、以及是左值还是右值。

有如下的两个函数，在flip中调用f：

```
void f(int v1, int &v2) {
    cout << v1 << " " << ++v2 << endl;
}
template <typename F, typename T1, typename T2>
void flip(F f, T1 t1, T2 t2) {
    f(t2, t1);
}
```

我们会发现f会改变第二个参数的值，但是通过flip调用f之后就不会改变

```
f(42, i)
flip(f, j, 42)
```

模板被实例化成如下：

```
void flip (void(*fcn)(int, int&), int t1, int t2);
```

j的值被拷贝到t1中，所以flip中的f只会改变t1，而不会改变j

3.1 定义能保持类型信息的函数参数

如果将flip的参数定义成右值引用，根据上面描述过的规则，当给flip传入引用时，T1被推断为int&，t1则被折叠成int&，完美保持了实参的类型。

```
template <typename F, typename T1, typename T2>
void flip(F f, T1&& t1, T2&& t2) {
    f(t2, t1);
}
```

但是当函数f接受右值引用作为参数的时候，flip就不能正常工作了

```
void f(int &&I, int j)
{
    cout << i << " " << j << endl;
}
flip(g,i,42)          //错误，不能从一个左值实例化int&&
```

注意，这里的f和g都不是模板函数，所以说前面提到的右值引用作为参数时的两个例外不能成立。所以这里是错误的

3.2 使用forward保持类型信息

Forward需要显示提供实参类型，返回该实参类型的右值引用（在前面可以看到，右值引用是可以赋值给左值引用的）

```
void flip(F f, T1&& t1, T2&& t2) {
    f(forward<T2>(t2), forward<T1>(t1));
}
```

当显式实参T是int&&时，forward返回int&& &&，折叠成int&&。当当显式实参T是int&时，forward返回int& &&，折叠成int&。所以说forward完美保持了参数的类型。

分类: C++

标签: c++, 右值引用

番茄汁汁
关注 - 0
粉丝 - 13
+加关注

2 0

« 上一篇： c++选择重载函数
» 下一篇： 南大算法设计与分析课程OJ答案代码（5）--割点与桥和任务调度问题

posted on 2018-05-17 17:27 番茄汁汁 阅读(17252) 评论(5) 编辑 收藏

FeedBack:

#1楼 2019-03-29 21:44 | philo&sophy

感觉好详细啊！一定要把您的这篇博文搞明白！

支持(0) 反对(0)

#2楼 2019-03-29 21:49 | philo&sophy

不是很理解X& &&是左值引用的右值引用吗？这样的话不是把一个右值引用绑定到左值上了？（我觉得左值引用和 右值引用都是右值）

支持(0) 反对(0)

#3楼 2019-03-29 21:50 | philo&sophy

哦说错了，我觉得左值引用和右值引用都是左值

支持(0) 反对(0)

#4楼 2019-07-06 09:03 | Jack_ff

学习了 感谢博主详细总结

支持(0) 反对(0)

#5楼 2020-05-20 01:15 | 专给楼主点赞

楼主转发的例子，为啥要t1和t2反过来写呢，感觉增加了理解难度啊

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册， 访问 网站首页。

相关博文：

· c++引用总结
· C++引用详解
· C++11的左值引用与右值引用总结

· C++右值引用浅析
· C++：引用的简单理解
» 更多推荐...

最新 IT 新闻:
· 京东：美国若无法检查审计情况 公司或从美股退市
· 医药史上最大并购！阿斯利康被曝洽购吉利德
· 抖音 头条 西瓜百亿流量扶持教育创作者 打造全民移动课堂
· 360全资收购大数据技术公司瀚思科技 共筑安全大脑
· 爱立信预计Q2或亏过亿美元 因中国5G业务初始成本高
» 更多新闻...