# C++引用在本质上是什么，它和指针到底有什么区别?

< 上一页 下一页 >

通过上节的讲解，相信各位读者对引用都有了一个概念上的认识，能够简单地使用引用编程了，但又感觉糊里糊涂，不明白它到底是什么，它和指针有点相似，但又不是一个东西。

首先来回顾一下上节的例子：

```
#include <iostream>
using namespace std;

int main(){
    int a = 99;
    int &r = a;
    cout<<a<<", "<<r<<endl;
    cout<<&a<<", "<<&r<<endl;

    return 0;
}
```

运行结果：
99, 99
0x28ff44, 0x28ff44

我们知道，变量是要占用内存的，虽然我们称 r 为变量，但是通过 &r 获取到的却不是 r 的地址，而是 a 的地址，这会让我们觉得 r 这个变量不占用独立的内存，它和 a 指代的是同一份内存。

请读者再继续看下面的例子：

```
#include <iostream>
#include <iomanip>
using namespace std;

int num = 99;

class A{
public:
    A();
```

```
10.  private:
11.      int n;
12.      int &r;
13.  };
14.
15.  A::A(): n(0), r(num){}
16.
17.  int main (){
18.      A *a = new A();
19.      cout<<sizeof(A)<<endl;  //输出A类型的大小
20.      cout<<hex<<showbase<<*((int*)a + 1)<<endl;  //输出r本身的内容
21.      cout<<&num<<endl;  //输出num变量的地址
22.
23.      return 0;
24.  }
```

运行结果：
8
0x442000
0x442000

成员变量 r 是 private 属性的，不能直接通过对象来访问，但是借助强大的指针和类型转换，我们依然可以得到它的内容，只不过这种方法有点蹩脚，我们将在《突破访问权限的限制（C++ Hack）》一节中详细阐述，读者暂时不必理解，只要知道第 20 行代码是用来输出 r 本身的内容的即可。

第 20 行代码中，`hex` 表示以十六进制输出，`showbase` 表示添加十六进制前缀 `0x`。

从运行结果可以看出：

- 成员变量 r 是占用内存的，如果不占用的话，`sizeof(A)` 的结果应该为 4。
- r 存储的内容是 `0x442000`，也即变量 num 的地址。

这说明 r 的实现和指针非常类似。如果将 r 定义为 `int *` 类型的指针，并在构造函数中让它指向 num，那么 r 占用的内存也是 4 个字节，存储的内容也是 num 的地址。

其实引用只是对指针进行了简单的封装，它的底层依然是通过指针实现的，引用占用的内存和指针占用的内存长度一样，在 32 位环境下是 4 个字节，在 64 位环境下是 8 个字节，之所以不能获取引用的地址，是因为编译器进行了内部转换。以下面的语句为例：

```
01.  int a = 99;
02.  int &r = a;
03.  r = 18;
04.  cout<<&r<<endl;
```

编译时会被转换成如下的形式：

```
int a = 99;
int *r = &a;
*r = 18;
cout<<r<<endl;
```

使用 &r 取地址时，编译器会对代码进行隐式的转换，使得代码输出的是 r 的内容（a 的地址），而不是 r 的地址，这就是为什么获取不到引用变量的地址的原因。也就是说，不是变量 r 不占用内存，而是编译器不让获取它的地址。

当引用作为函数参数时，也会有类似的转换。以下面的代码为例：

```
//定义函数
void swap(int &r1, int &r2){
    int temp = r1;
    r1 = r2;
    r2 = temp;
}

//调用函数
int num1 = 10, num2 = 20;
swap(num1, num2);
```

编译时会被转换成如下的形式：

```
//定义函数
void swap(int *r1, int *r2){
    int temp = *r1;
    *r1 = *r2;
    *r2 = temp;
}

//调用函数
int num1 = 10, num2 = 20;
swap(&num1, &num2);
```

引用虽然是基于指针实现的，但它比指针更加易用，从上面的两个例子也可以看出来，通过指针获取数据时需要加 * ，书写麻烦，而引用不需要，它和普通变量的使用方式一样。

C++ 的发明人 Bjarne Stroustrup 也说过，他在 C++ 中引入引用的直接目的是为了让代码的书写更加漂亮，尤其是在运算符重载中，不借助引用有时候会使得运算符的使用很麻烦。

# 引用和指针的其他区别

1) 引用必须在定义时初始化，并且以后也要从一而终，不能再指向其他数据；而指针没有这个限制，指针在定义时不必赋值，以后也能指向任意数据。

2) 可以有 const 指针，但是没有 const 引用。也就是说，引用变量不能定义为下面的形式：

```
int a = 20;
int & const r = a;
```

因为 r 本来就不能改变指向，加上 const 是多此一举。

3) 指针可以有多级，但是引用只能有一级，例如，`int **p` 是合法的，而 `int &&r` 是不合法的。如果希望定义一个引用变量来指代另外一个引用变量，那么也只需要加一个 `&`，如下所示：

```
int a = 10;
int &r = a;
int &rr = r;
```

4) 指针和引用的自增（++）自减（--）运算意义不一样。对指针使用 ++ 表示指向下一份数据，对引用使用 ++ 表示它所指代的数据本身加 1；自减（--）也是类似的道理。请看下面的例子：

```
#include <iostream>
using namespace std;

int main (){
    int a = 10;
    int &r = a;
    r++;
    cout<<r<<endl;

    int arr[2] = { 27, 84 };
    int *p = arr;
    p++;
    cout<<*p<<endl;

    return 0;
}
```

运行结果：
11
84

< 上一页 下一页 >

**所有教程**

C语言入门 C语言编译器 C语言项目案例 数据结构 多线程 链接库
C++ STL C++11 socket GCC GDB Makefile OpenCV
Qt教程 Unity 3D UE4 游戏引擎 Python Python并发编程
TensorFlow Django NumPy Linux Shell Java教程 设计模式
Java Swing Servlet教程 JSP教程 Struts2 Maven Nexus Spring
Spring MVC Spring Boot Spring Cloud Hibernate Mybatis
MySQL教程 MySQL函数 NoSQL Redis常用命令手册 HBase MongoDB
Go语言 C# MATLAB JavaScript Bootstrap HTML CSS PHP
汇编语言 TCP/IP vi命令 Android教程 区块链 Docker 大数据
云计算 推荐阅读 编程笔记 资源下载 VIP视频 一对一答疑 关于我们