编译器会为const引用创建临时变量

< 上一页 下一页 >

上节我们讲到，引用不能绑定到临时数据，这在大多数情况下是正确的，但是当使用 const 关键字对引用加以限定后，引用就可以绑定到临时数据了。下面的代码演示了引用和 const 这一对神奇的组合：

```
01.  typedef struct{
02.      int a;
03.      int b;
04.  } S;
05.
06.  int func_int(){
07.      int n = 100;
08.      return n;
09.  }
10.
11.  S func_s(){
12.      S a;
13.      a.a = 100;
14.      a.b = 200;
15.      return a;
16.  }
17.
18.  S operator+(const S &A, const S &B){
19.      S C;
20.      C.a = A.a + B.a;
21.      C.b = A.b + B.b;
22.      return C;
23.  }
24.
25.  int main(){
26.      int m = 100, n = 36;
27.      const int &r1 = m + n;
28.      const int &r2 = m + 28;
29.      const int &r3 = 12 * 3;
30.      const int &r4 = 50;
31.      const int &r5 = func_int();
32.
```

```
    S s1 = {23, 45};
    S s2 = {90, 75};
    const S &r6 = func_s();
    const S &r7 = s1 + s2;

    return 0;
}
```

这段代码在 GCC 和 Visual C++ 下都能够编译通过，这是因为将常引用绑定到临时数据时，编译器采取了一种妥协机制：**编译器会为临时数据创建一个新的、无名的临时变量，并将临时数据放入该临时变量中，然后再将引用绑定到该临时变量。**注意，临时变量也是变量，所有的变量都会被分配内存。

为什么编译器为常引用创建临时变量是合理的，而为普通引用创建临时变量就不合理呢？

1) 我们知道，将引用绑定到一份数据后，就可以通过引用对这份数据进行操作了，包括读取和写入（修改）；尤其是写入操作，会改变数据的值。而临时数据往往无法寻址，是不能写入的，即使为临时数据创建了一个临时变量，那么修改的也仅仅是临时变量里面的数据，不会影响原来的数据，这样就使得引用所绑定到的数据和原来的数据不能同步更新，最终产生了两份不同的数据，失去了引用的意义。

以《C++引用10分钟入门教程》一节中讲到的 swap() 函数为例：

```
void swap(int &r1, int &r2){
    int temp = r1;
    r1 = r2;
    r2 = temp;
}
```

如果编译器会为 r1、r2 创建临时变量，那么函数调用 `swap(10, 20)` 就是正确的，但是 10 不会变成 20，20 也不会变成 10，所以这种调用是毫无意义的。

总起来说，不管是从“引用的语义”这个角度看，还是从“实际应用的效果”这个角度看，为普通引用创建临时变量都没有任何意义，所以编译器不会这么做。

2) const 引用和普通引用不一样，我们只能通过 const 引用读取数据的值，而不能修改它的值，所以不用考虑同步更新的问题，也不会产生两份不同的数据，为 const 引用创建临时变量反而会使得引用更加灵活和通用。

以上节的 isOdd() 函数为例：

```
bool isOdd(const int &n){  //改为常引用
    if(n/2 == 0){
```

```
        return false;
    }else{
        return true;
    }
}
```

由于在函数体中不会修改 n 的值，所以可以用 const 限制 n，这样一来，下面的函数调用就都是正确的了：

```
int a = 100;
isOdd(a);  //正确
isOdd(a + 9);  //正确
isOdd(27);  //正确
isOdd(23 + 55);  //正确
```

对于第 2 行代码，编译器不会创建临时变量，会直接绑定到变量 a；对于第 3~5 行代码，编译器会创建临时变量来存储临时数据。也就是说，编译器只有在必要时才会创建临时变量。

< 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