C++引用不能绑定到临时数据

< 上一页 下一页 >

我们知道，指针就是数据或代码在内存中的地址，指针变量指向的就是内存中的数据或代码。这里有一个关键词需要强调，就是 内存 ，指针只能指向内存，不能指向寄存器或者硬盘，因为寄存器和硬盘没法寻址。

其实 C++ 代码中的大部分内容都是放在内存中的，例如定义的变量、创建的对象、字符串常量、函数形参、函数体本身、 new 或 malloc() 分配的内存等，这些内容都可以用 & 来获取地址，进而用指针指向它们。除此之外，还有一些我们平时不太留意的临时数据，例如表达式的结果、函数的返回值等，它们可能会放在内存中，也可能会放在寄存器中。一旦它们被放到了寄存器中，就没法用 & 获取它们的地址了，也就没法用指针指向它们了。

下面的代码演示了表达式所产生的临时结果：

```
int n = 100, m = 200;
int *p1 = &(m + n);     //m + n 的结果为 300
int *p2 = &(n + 100);   //n + 100 的结果为 200
bool *p4 = &(m < n);    //m < n 的结果为 false
```

这些表达式的结果都会被放到寄存器中，尝试用 & 获取它们的地址都是错误的。

下面的代码演示了函数返回值所产生的临时结果：

```
int func(){
    int n = 100;
    return n;
}

int *p = &(func());
```

func() 的返回值 100 也会被放到寄存器中，也没法用 & 获取它的地址。

什么样的临时数据会放到寄存器中

寄存器离 CPU 近，并且速度比内存快，将临时数据放到寄存器是为了加快程序运行。但是寄存器的数量是非常有限的，容纳不下较大的数据，所以只能将较小的临时数据放在寄存器中。int、double、bool、char 等基本类型的数据往往不超过 8 个字节，用一两个寄存器就能存储，所以这些类型的临时数据通常会放到寄存器中；而对象、结构体变量是自定义类型的数据，大小不可预测，所以这些类型的临时数据通常会放到内存中。

下面的代码是正确的，它证明了结构体类型的临时数据会被放到内存中：

```
01.  #include <iostream>
02.  using namespace std;
03.
04.  typedef struct{
05.      int a;
06.      int b;
07.  } S;
08.
09.  //这里用到了一点新知识，叫做运算符重载，我们会在《运算符重载》一章中详细讲解
10.  S operator+(const S &A, const S &B){
11.      S C;
12.      C.a = A.a + B.a;
13.      C.b = A.b + B.b;
14.      return C;
15.  }
16.
17.  S func(){
18.      S a;
19.      a.a = 100;
20.      a.b = 200;
21.      return a;
22.  }
23.
24.  int main(){
25.      S s1 = {23, 45};
26.      S s2 = {90, 75};
27.      S *p1 = &(s1 + s2);
28.      S *p2 = &(func());
29.      cout<<p1<<", "<<p2<<endl;
30.
31.      return 0;
32.  }
```

运行结果：
0x28ff28, 0x28ff18

> 第10行代码用到了运算符重载，我们将在《C++运算符重载》一章中详细讲解。

关于常量表达式

诸如 100、200+34、34.5*23、3+7/3 等不包含变量的表达式称为常量表达式（Constant expression）。

常量表达式由于不包含变量，没有不稳定因素，所以在编译阶段就能求值。编译器不会分配单独的内存来存储常量表达式的值，而是将常量表达式的值和代码合并到一起，放到虚拟地址空间中的代码区。从汇编的角度看，常量表达式的值就是一个立即数，会被“硬编码”到指令中，不能寻址。

> 关于虚拟地址空间的分区，我们已在《Linux下C语言程序的内存布局》一节中讲到。

总起来说，常量表达式的值虽然在内存中，但是没有办法寻址，所以也不能使用 & 来获取它的地址，更不能用指针指向它。下面的代码是错误的，它证明了不能用 & 来获取常量表达式的地址：

```
int *p1 = &(100);
int *p2 = &(23 + 45 * 2);
```

引用也不能指代临时数据

引用和指针在本质上是一样的，引用仅仅是对指针进行了简单的封装。引用和指针都不能绑定到无法寻址的临时数据，并且 C++ 对引用的要求更加严格，在某些编译器下甚至连放在内存中的临时数据都不能指代。

下面的代码中，我们将引用绑定到了临时数据：

```
typedef struct{
    int a;
    int b;
} S;

int func_int(){
    int n = 100;
    return n;
}

S func_s(){
    S a;
    a.a = 100;
    a.b = 200;
    return a;
}

//这里用到了一点新知识，叫做运算符重载，我们会在《运算符重载》一章中详细讲解
S operator+(const S &A, const S &B){
```

```
    S C;
    C.a = A.a + B.a;
    C.b = A.b + B.b;
    return C;
}

int main(){
    //下面的代码在GCC和Visual C++下都是错误的
    int m = 100, n = 36;
    int &r1 = m + n;
    int &r2 = m + 28;
    int &r3 = 12 * 3;
    int &r4 = 50;
    int &r5 = func_int();

    //下面的代码在GCC下是错误的，在Visual C++下是正确的
    S s1 = {23, 45};
    S s2 = {90, 75};
    S &r6 = func_s();
    S &r7 = s1 + s2;

    return 0;
}
```

第 28~33 行代码在 GCC 和 Visual C++ 下都不能编译通过，第 38~39 行代码在 Visual C++ 下能够编译通过，但是在 GCC 下编译失败。这说明：

- 在 GCC 下，引用不能指代任何临时数据，不管它保存到哪里；
- 在 Visual C++ 下，引用只能指代位于内存中（非代码区）的临时数据，不能指代寄存器中的临时数据。

引用作为函数参数

当引用作为函数参数时，有时候很容易给它传递临时数据。下面的 isOdd() 函数用来判断一个数是否是奇数：

```
bool isOdd(int &n){
    if(n%2 == 0){
        return false;
    }else{
        return true;
    }
}

int main(){
    int a = 100;
    isOdd(a);  //正确
```

```
    isOdd(a + 9);  //错误
    isOdd(27);  //错误
    isOdd(23 + 55);  //错误

    return 0;
}
```

isOdd() 函数用来判断一个数是否为奇数，它的参数是引用类型，只能传递变量，不能传递常量或者表达式。但用来判断奇数的函数不能接受一个数字又让人感觉很奇怪，所以类似这样的函数应该坚持使用值传递，而不是引用传递。

下面是更改后的代码：

```
bool isOdd(int n){  //改为值传递
    if(n%2 == 0){
        return false;
    }else{
        return true;
    }
}

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

    return 0;
}
```

< 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