10.3 —通過引用傳遞參數

亞歷克斯（ALEX）在2007年7月24日| 由NASCARDRIVER於2021年1月23日最後修改

雖然按值傳遞在許多情況下都適用，但它有兩個限制。首先，將大型結構或類傳遞給函數時，按值傳遞會將參數的副本複製到函數參數中。在許多情況下，這是不必要的性能降低，因為原始參數就足夠了。其次，當按值傳遞參數時，將值返回給調用者的唯一方法是通過函數的返回值。儘管這通常很合適，但在某些情況下，使函數修改傳入的參數會更加清楚和有效。按引用傳遞解決了這兩個問題。

通過參考

要通過引用傳遞變量，我們只需將函數參數聲明為引用，而不是普通變量即可：

```
void addOne(int &ref) // ref is a reference variable
{
    ref = ref + 1;
}
```

調用該函數時，ref將成為對參數的引用。由於對變量的引用與變量本身完全一樣，因此對該引用所做的任何更改都將傳遞給參數！

下面的示例演示了這一操作：

```
void addOne(int &ref)
{
    ref = ref + 1;
}

int main()
{
    int value = 5;

    cout << "value = " << value << '\n';
    addOne(value);
    cout << "value = " << value << '\n';
    return 0;
}
```

該程序與我們用於傳遞值的示例的程序相同，只是foo的參數現在是引用而不是普通變量。當我們調用addOne（value）時，ref成為對main的value變量的引用。此代碼段產生輸出：

```
值= 5
值= 6
```

如您所見，該函數將參數的值從5更改為6！

通過out參數返回多個值

有時我們需要一個函數來返回多個值。但是，函數只能有一個返回值。返回多個值的一種方法是使用參考參數：

```
#include <iostream>
#include <cmath>    // for std::sin() and std::cos()

void getSinCos(double degrees, double &sinOut, double &cosOut)
{
    // sin() and cos() take radians, not degrees, so we need to convert
    static constexpr double pi { 3.14159265358979323846 }; // the value of pi
    double radians = degrees * pi / 180.0;
    sinOut = std::sin(radians);
    cosOut = std::cos(radians);
}

```

```
int main()
{
    double sin(0.0);
    double cos(0.0);

    // getSinCos will return the sin and cos in variables sin and cos
    getSinCos(30.0, sin, cos);

    std::cout << "The sin is " << sin << '\n';
    std::cout << "The cos is " << cos << '\n';
    return 0;
}
```

此函數將一個參數（按值）作為輸入，並“返回”兩個參數（按引用）作為輸出。僅用於將值返回給調用方的**參數**稱為**調出參數**。我們用後綴“ out”命名了這些out參數，以表示它們是out參數。這有助於提醒調用者，傳遞給這些參數的初始值無關緊要，我們應該期望它們會被重寫。按照慣例，輸出參數通常是最右邊的參數。

讓我們更詳細地探討它的工作原理。首先，主函數創建局部變量sin和cos。這些通過引用（而不是通過值）傳遞到getSinCos（）函數中。這意味著函數getSinCos（）可以訪問實際的sin和cos變量，而不僅僅是副本。getSinCos（）相應地為sin和cos分配了新值（分別通過引用sinOut和cosOut），這將覆蓋sin和cos中的舊值。Main然後打印這些更新的值。

如果sin和cos是通過值而不是引用傳遞的，則getSinCos（）將更改sin和cos的副本，從而導致任何更改在函數末尾被丟棄。但是由於sin和cos是通過引用傳遞的，因此對sin或cos（通過引用）所做的任何更改都將保留在功能之外。因此，我們可以使用這種機制將值返回給調用方。

這種方法雖然實用，但也有一些小缺點。首先，調用者必須傳入參數以保留更新的輸出，即使它不打算使用它們也是如此。更重要的是，語法有點不自然，輸入和輸出參數都放在函數調用中。從調用方的角度來看，sin和cos是輸出參數並且將被更改，這一點並不明顯。這可能是此方法中最危險的部分（因為它可能導致犯錯誤）。一些程序員和公司認為這是一個足夠大的問題，建議完全避免使用輸出參數，或者改用out by address作為輸出參數（語法更清晰，指示參數是否可修改）。

就個人而言，我們建議盡可能避免使用所有參數。如果確實使用它們，則用“ out”後綴（或前綴）命名參數（和輸出參數）可以幫助您清楚地知道該值可能會被修改。

通過參考的局限性

非const引用只能引用非const l值（例如，非const變量），因此引用參數不能接受為const l值或r值的參數（例如，文字和表達式的結果）。

通過const引用傳遞

如引言中所述，按值傳遞的主要缺點之一是將按值傳遞的所有參數都複製到函數參數中。當參數是大型結構或類時，這可能會花費很多時間。參考文獻提供了避免這種損失的方法。當參數通過引用傳遞時，將創建對實際參數的引用（這需要最少的時間），並且不會復制任何值。這使我們能夠以最小的性能損失傳遞大型結構和類。

但是，這也給我們帶來了潛在的麻煩。引用允許函數更改參數的值，當我們希望參數為只讀時，這是不可取的。如果我們知道函數不應更改參數的值，但又不想按值傳遞，最好的解決方案是按const引用傳遞。

您已經知道const引用是不允許通過引用更改被引用變量的引用。因此，如果我們使用const引用作為參數，則可以向調用方保證該函數不會更改參數！

以下函數將產生編譯器錯誤：

```
void foo(const std::string &x) // x is a const reference
{
    x = "hello";  // compile error: a const reference cannot have its value changed!
}
```

使用const很有用，原因如下：

- 它要求編譯器幫助確保不要更改不應更改的值（如上例所示，如果您嘗試這樣做，編譯器將拋出錯誤）。
- 它告訴程序員該函數不會更改參數的值。這可以幫助調試。

- 您不能將const參數傳遞給非const參考參數。使用const參數可確保您可以將非const和const參數都傳遞給函數。
- const引用可以接受任何類型的參數，包括非const l值，const l值和r值。

規則

通過引用傳遞參數時，除非需要更改參數的值，否則請始終使用const引用。

提醒

非常量引用不能綁定到r值。帶有非常量引用參數的函數不能使用文字或臨時變量來調用。

```
#include <string>

void foo(std::string& text) {}

int main()
{
  std::string text{ "hello" };

  foo(text); // ok
  foo(text + " world"); // illegal, non-const references can't bind to r-values.

  return 0;
}
```

指向指針

可以通過引用傳遞指針，並使函數完全更改指針的地址：

```
#include <iostream>

void foo(int *&ptr) // pass pointer by reference
{
    ptr = nullptr; // this changes the actual ptr argument passed in, not a copy
}

int main()
{
    int x = 5;
    int *ptr = &x;
    std::cout << "ptr is: " << (ptr ? "non-null" : "null") << '\n'; // prints non-null
    foo(ptr);
    std::cout << "ptr is: " << (ptr ? "non-null" : "null") << '\n'; // prints null

    return 0;
}

```

（我們將在下一課中顯示另一個示例）

提醒一下，您可以通過引用傳遞C樣式的數組。如果您需要該函數具有更改數組的功能（例如，用於排序函數），或者需要訪問固定數組的數組類型信息（執行sizeof（）或for-each循環），則此功能很有用。但是，請注意，為了使此功能起作用，您明確需要在參數中定義數組大小：

```
#include <iostream>

// Note: You need to specify the array size in the function declaration
void printElements(int (&arr)[4])
{
  int length{ sizeof(arr) / sizeof(arr[0]) }; // we can now do this since the array won't de
cay
```

```
7
8     for (int i{ 0 }; i < length; ++i)
9     {
10      std::cout << arr[i] << '\n';
11    }
12  }
13
14  int main()
15  {
16      int arr[]{ 99, 20, 14, 80 };
17
18      printElements(arr);
19
20      return 0;
21  }
```

這意味著這僅適用於一個特定長度的固定數組。如果希望此方法可用於任何長度的固定數組，則可以將數組長度設置為模板參數（在下一章中介紹）。

通過引用傳遞的利弊

通過引用傳遞的優點：

- 引用允許函數更改參數的值，這有時很有用。否則，可以使用const引用來確保函數不會更改參數。
- 因為未復制參數，所以即使與大型結構或類一起使用，按引用傳遞也很快。
- 引用可用於從函數（通過out參數）返回多個值。
- 引用必須初始化，因此不必擔心null值。

通過引用的缺點：

- 因為無法使用const l值或r值（例如，文字或表達式）初始化非常量引用，所以非常量引用參數的參數必須為普通變量。
- 很難判斷通過非常量引用傳遞的參數是輸入，輸出還是兩者都是。明智地使用const和out變量的命名後綴會有所幫助。
- 從函數調用中無法判斷參數是否可能更改。通過值傳遞和通過引用傳遞的參數看起來相同。我們只能通過查看函數聲明來判斷參數是通過值傳遞還是通過引用傳遞。這可能導致程序員無法意識到函數會改變參數值的情況。

何時使用通過引用傳遞：

- 傳遞結構或類時（如果為只讀，則使用const）。
- 當您需要函數來修改參數時。
- 當您需要訪問固定數組的類型信息時。

什麼時候不使用引用傳遞：

- 傳遞不需要修改的基本類型時（使用按值傳遞）。

規則

對於結構和類以及其他復制昂貴的類型，請使用“按（const）”引用而不是按值傳遞。

10.4-通過地址傳遞參數

指數