10.4 —通過地址傳遞參數

亞歷克斯（ALEX）在2007年7月25日| 由ALEX於2020年12月21日最後修改

還有另一種將變量傳遞給函數的方法，即通過地址傳遞。 **通過地址傳遞參數**涉及傳遞參數變量的地址，而不是參數變量本身的地址。因為參數是地址，所以功能參數必須是指針。然後，該函數可以取消對指針的引用以訪問或更改所指向的值。

這是一個採用通過地址傳遞參數的函數的示例：

```
#include <iostream>

void foo(int *ptr)
{
    *ptr = 6;
}

int main()
{
    int value{ 5 };

    std::cout << "value = " << value << '\n';
    foo(&value);
    std::cout << "value = " << value << '\n';
    return 0;
}
```

上面的片段打印：

```
值= 5
值= 6
```

如您所見，函數foo（）通過指針參數ptr更改了參數的值（變量值）。

通過地址傳遞通常與指針一起使用，指針通常用於指向內置數組。例如，以下函數將打印數組中的所有值：

```
void printArray(int *array, int length)
{
    for (int index{ 0 }; index < length; ++index)
    {
        std::cout << array[index] << ' ';
    }
}
```

這是一個調用此函數的示例程序：

```
int main()
{
    int array[6]{ 6, 5, 4, 3, 2, 1 }; // remember, arrays decay into pointers
    printArray(array, 6); // so array evaluates to a pointer to the first element of the arra
y here, no & needed
}
```

該程序將打印以下內容：

```
6 5 4 3 2 1
```

請記住，固定數組在傳遞給函數時會衰減為指針，因此我們必須將長度作為單獨的參數傳遞。

在取消引用地址之前，確保通過地址傳遞的參數不是空指針始終是一個好主意。取消引用空指針通常會導致程序崩潰。這是我們的帶有空指針檢查的printArray（）函數：

```
void printArray(int *array, int length)
{
    // if user passed in a null pointer for array, bail out early!
```

```
    if (!array)
        return;

    for (int index{ 0 }; index < length; ++index)
        std::cout << array[index] << ' ';
}

int main()
{
    int array[6]{ 6, 5, 4, 3, 2, 1 };
    printArray(array, 6);
}
```

通過const地址傳遞

因為printArray（）不會修改其任何參數，所以使數組參數const成為一種很好的形式：

```
void printArray(const int *array, int length)
{
    // if user passed in a null pointer for array, bail out early!
    if (!array)
        return;

    for (int index{ 0 }; index < length; ++index)
        std::cout << array[index] << ' ';
}

int main()
{
    int array[6]{ 6, 5, 4, 3, 2, 1 };
    printArray(array, 6);
}
```

這使我們能夠一目了然地知道printArray（）不會修改傳入的數組參數，並確保我們不會偶然這樣做。

地址實際上是按值傳遞的

當您將指針傳遞給函數時，指針的值（其指向的地址）將從參數複製到函數的參數。換句話說，它是通過價值傳遞的！如果更改功能參數的值，則僅更改副本。因此，原始指針參數將不會更改。

這是一個說明此情況的示例程序。

```
#include <iostream>

void setToNull(int *tempPtr)
{
    // we're making tempPtr point at something else, not changing the value that tempPtr poi
nts to.
    tempPtr = nullptr; // use 0 instead if not C++11
}

int main()
{
    // First we set ptr to the address of five, which means *ptr = 5
    int five{ 5 };
    int *ptr{ &five };

    // This will print 5
    std::cout << *ptr;

    // tempPtr will receive a copy of ptr
    setToNull(ptr);

    // ptr is still set to the address of five!

    // This will print 5
    if (ptr)
        std::cout << *ptr;
    else
        std::cout << " ptr is null";
```

```
28
29      return 0;
30  }
```

tempPtr接收ptr持有的地址的副本。即使我們將tempPtr更改為指向其他內容（nullptr），這也不會更改ptr指向的值。因此，該程序將打印：

```
55
```

請注意，即使地址本身是通過值傳遞的，您仍然可以取消引用該地址以更改參數的值。這是一個常見的混淆點，因此讓我們澄清一下：

- 通過地址傳遞參數時，函數參數變量將從參數接收地址的副本。此時，函數參數和參數都指向相同的值。
- 如果隨後取消引用函數參數以更改所指向的值，則將影響參數所指向的值，因為函數參數和參數都指向相同的值！
- 如果為function參數分配了一個不同的地址，**則不會**影響參數，因為function參數是一個副本，並且更改副本不會影響原始參數。更改功能參數的地址後，功能參數和參數將指向不同的值，因此取消引用參數並更改值將不再影響參數所指向的值。

以下程序說明了這一點：

```
1   #include <iostream>
2
3   void setToSix(int *tempPtr)
4   {
5       *tempPtr = 6; // we're changing the value that tempPtr (and ptr) points to
6   }
7
8   int main()
9   {
10      // First we set ptr to the address of five, which means *ptr = 5
11      int five{ 5 };
12      int *ptr{ &five };
13
14      // This will print 5
15      std::cout << *ptr;
16
17      // tempPtr will receive a copy of ptr
18      setToSix(ptr);
19
20      // tempPtr changed the value being pointed to to 6, so ptr is now pointing to the value
     6
21
22      // This will print 6
23      if (ptr)
24          std::cout << *ptr;
25      else
26          std::cout << " ptr is null";
27
28      return 0;
29  }
```

打印：

```
56
```

通過引用傳遞地址

下一個邏輯問題是：“如果我們要更改函數中自變量指向的地址，該怎麼辦？”事實證明，這非常容易。您可以簡單地通過引用傳遞地址。引用指針的語法有點奇怪（而且很容易倒退）。但是，如果確實將其倒退，則編譯器將給您一個錯誤。

下面的程序說明瞭如何使用對指針的引用：

```
1   #include <iostream>
2
3
```

```
// tempPtr is now a reference to a pointer, so any changes made to tempPtr will change the argument as well!
void setToNull(int *&tempPtr)
{
    tempPtr = nullptr; // use 0 instead if not C++11
}

int main()
{
    // First we set ptr to the address of five, which means *ptr = 5
    int five{ 5 };
    int *ptr{ &five };

    // This will print 5
    std::cout << *ptr;

    // tempPtr is set as a reference to ptr
    setToNull(ptr);

    // ptr has now been changed to nullptr!

    if (ptr)
        std::cout << *ptr;
    else
        std::cout << " ptr is null";

    return 0;
}
```

當我們使用此版本的功能再次運行該程序時，將得到：

5點為空

這表明調用setToNull（）確實確實將ptr的值從&five更改為nullptr！

只有通過價值

現在您已經了解了按引用傳遞，地址傳遞和值傳遞之間的基本區別，讓我們暫時看一看簡化派。:)

在有關**引用**的課程中，我們簡要提到了引用通常由編譯器作為指針實現。這意味著在後台，按引用傳遞本質上只是一個按地址傳遞（通過對引用進行隱式取消引用）。

在上面，我們顯示了按地址傳遞實際上就是按值傳遞地址！

因此，我們可以得出結論，C ++確實通過價值傳遞了一切！通過地址（和引用）傳遞的屬性僅來自以下事實：我們可以取消引用傳遞的地址以更改參數，而使用常規值參數則無法做到這一點！

通過地址傳遞使可修改參數明確

考慮以下示例：

```
int foo1(int x); // pass by value
int foo2(int &x); // pass by reference
int foo3(int *x); // pass by address

int i {};

foo1(i);  // can't modify i
foo2(i);  // can modify i
foo3(&i); // can modify i
```

從對foo2（）的調用中還不明顯，該函數可以修改變量i，是嗎？

因此，一些指南建議按地址傳遞所有可修改的參數，以便從現有函數調用中可以明顯看出可以修改參數。

但是，這有其自身的缺點：調用者可能認為他們可以在不應該使用的時候傳遞nullptr，現在您必須嚴格檢查null指針。

我們傾向於通過引用傳遞非可選的可修改參數的建議。更好的是，完全避免修改參數。

通過地址傳遞的利弊

通過地址傳遞的優點：

- 通過地址傳遞允許函數更改參數的值，這有時很有用。否則，可以使用const來確保函數不會更改參數。（但是，如果要使用非指針執行此操作，則應改用按引用傳遞）。
- 由於未復制參數，因此即使與大型結構或類一起使用時，該參數也很快速。
- 我們可以通過out參數從一個函數返回多個值。

通過地址傳遞的缺點：

- 因為文字（C風格的字符串文字除外）和表達式沒有地址，所以指針參數必須是普通變量。
- 必須檢查所有值以查看它們是否為空。嘗試取消引用空值將導致崩潰。很容易忘記這樣做。
- 因為取消引用指針比直接訪問值要慢，所以訪問由地址傳遞的參數要比訪問由值傳遞的參數慢。

何時使用按地址傳遞：

- 當傳遞內置數組時（如果您可以接受它們會衰減為指針的事實）。
- 從邏輯上講，傳遞指針和nullptr時是有效的參數。

什麼時候不使用通過地址傳遞：

- 從邏輯上講，傳遞指針和nullptr時不是有效的參數（使用按引用傳遞）。
- 傳遞結構或類時（使用按引用傳遞）。
- 傳遞基本類型時（使用按值傳遞）。

如您所見，按地址傳遞和按引用傳遞具有幾乎相同的優點和缺點。由於按引用傳遞通常比按地址傳遞更安全，因此在大多數情況下應首選按引用傳遞。

規則

在適用的情況下，建議先通過引用傳遞通過地址傳遞。

10.5-按值，引用和地址返回值

指數

10.3-通過引用傳遞參數

C ++教程 | 打印這篇文章

對10.4的228條評論-通過地址傳遞參數

«較早的評論 1 2 3

名爵
2020年12月15日上午6:40 ·回复

我們傾向於通過引用傳遞非可選的可修改參數的建議。更好的是，完全避免修改參數。