Javaアプリケーションでの値による受け渡し

Peter Haggar

2000年 5月 01日

数か月ほど前になりますが、私の著書 Practical Java (Addison-Wesley 刊) からの抜粋が、developerWorks に掲載されました。このコラムでは、掲載された抜粋について皆様から寄せられた質問やコメントにお答えしたいと思います。

Java アプリケーションは参照によってパラメーターを渡すものと一般には考えられているようですが、この記事はそうした誤りを明らかにして、皆さんに正しい Java プログラミングの知識を持っていただくことを目的としていました。

記事の内容について、問題を複雑にしているだけだとか、考え方がまったくおかしいという意見もありました。 反論の多くは C++ 言語を例にあげていましたので、このコラムでは C++ や Java アプリケーションを用いて、事実を明らかにしていきたいと思います。

キー・ポイント

いただいたご意見をすべて読み終えて分かったのは、考えを混乱させる大きな原因が、少なくとも 1 つあるということです。このようなご意見がありました。「オブジェクトは参照によって受け渡されるので、記事の内容は誤りではないのか ?」。オブジェクト が参照によって受け渡されるのは事実です。記事でもこのことは否定していません。私が言いたかったのは、パラメーターは値、つまり異なる引数によって受け渡されるということです。確かに Java アプリケーションでは、オブジェクトではなく、オブジェクト参照のみを受け渡します。そのため、オブジェクトは参照により受け渡されることになります。ただし、パラメーターの受け渡し方法の違いを区別することは重要です。そして、それこそが抜粋記事の目的だったのです。Java アプリケーションは、オブジェクトを参照により受け渡します。しかし、パラメーターを参照により受け渡すわけではありません。パラメーターにはオブジェクト参照を指定することができるので、Java アプリケーションはオブジェクト参照を値によって受け渡すことができます。

C++ と Java アプリケーションでのパラメーターの受け渡し

Java アプリケーションの変数には、参照型とプリミティブ型の 2 つがあります。どちらの型も、引数としてメソッドに受け渡されると、同じ方法で処理されます。また、どちらの型も、値による受け渡しが行われます。参照によって受け渡されることはありません。これは重要な特徴です。後で示すサンプル・コードを見てもそれが分かります。

先に進む前に、値による受け渡し と参照による受け渡し という用語の意味をよく確認しておきましょう。値による受け渡し では、引数が関数に渡されるときに、関数は元の値のコピーを受け取ります。したがって、関数がパラメーターを変更しても、そのコピーが変更されるだけで、元の値はそのまま変わりません。参照による受け渡し では、引数が関数に渡されるときに、関数は元の値のコピーではなく、メモリー・アドレスを受け取ります。したがって、関数がパラメーターを変更すると、呼び出し側のコードの元の値が変更されます。

Java アプリケーションでのパラメーターの受け渡しについて誤解が生じるのは、多くのプログラマーが C++ プログラミングの考え方を Java プログラミングにも持ち込んでいるためです。C++ には、非参照型と参照型が含まれます。非参照型は値による受け渡しが行われ、参照型は参照による受け渡しが行われます。Java プログラム言語にはプリミティブ型とオブジェクト参照があります。そのため、Java アプリケーションでは、C++ と同様に、プリミティブ型については値による受け渡しが行われ、オブジェクト参照については参照による受け渡しが行われると考えるのが理にかなっています。こうした理由で、参照の受け渡し行うときは、参照による受け渡し にすべきだと考えてしまうのかもしれません。確かにこう考えるのが自然ですし、実際私もある時期そのように考えていました。しかし、この考えは正しくなかったのです。

関数に渡されるパラメーターが参照でない場合、C++ および Java のアプリケーションでは、ともに値のコピーが渡されます (値による受け渡し)。参照が渡される場合は、仕組みが異なります。C++ では、関数に渡されるパラメーターが参照の場合、参照またはメモリー・アドレスが渡されます (参照による受け渡し)。一方、Java アプリケーションでは、メソッドに渡されるパラメーターがオブジェクト参照の場合、参照そのものではなく、参照のコピーが渡されます (値による受け渡し)。ここで、呼び出し側のメソッドのオブジェクト参照とそのコピーが同じオブジェクトを指していることに注意してください。ここが、大きな違いなのです。各種のパラメーターを渡す場合、Java アプリケーションは、どれも同じように処理します。C++ とは、この点が異なります。Java アプリケーションは、どのパラメーターも値により渡すので、すべてのパラメーターのコピーをその型とは関係なく作成します。

例

これまでの説明をもとにして、以下の例を分析してみましょう。最初に、C++ コードを検討してみます。C++ 言語では、引数を受け渡す際に、値による受け渡しと参照による受け渡しの両方を使用します。以下の例をご覧ください。

リスト 1: C++ の例

```
#include
#include
void modify(int a, int *P, int &r);
int main (int argc, char** argv)
{
  int val, ref;
  int *pint;
  val = 10;
  ref = 50;
  pint = (int*)malloc(sizeof(int));
  *pint = 15;
  printf("val is %d\n", val);
  printf("pint is %d\n", pint);
  printf("*pint is %d\n", *pint);
  printf("ref is %d\n\n", ref);
  printf("calling modify\n");
```

```
  //val and pint passed by value, ref is passed by reference.
  modify(val, pint, ref);
  printf("returned from modify\n\n");
  printf("val is %d\n", val);
  printf("pint is %d\n", pint);
  printf("*pint is %d\n", *pint);
  printf("ref is %d\n", ref);
  return 0;
}
void modify(int a, int *p, int &r)
{
    printf("in modify...\n");
    a = 0;
    *p = 7;
    p = 0;
    r = 0;
    printf("a is %d\n", a);
    printf("p is %d\n", p);
    printf("r is %d\n", r);
}
```

このコードの出力結果は次のようになります。

リスト 2: C++ コードの出力

```
pint is 4262128
*pint is 15
ref is 50
calling modify
in modify...
a is 0
p is 0
r is 0
returned from modify
val is 10
pint is 4262128
*pint is 7
ref is 0
```

このコードでは、2 つの int と 1 つのポインターの 3 つの変数を宣言しています。それぞれの変数は初期値が設定され、出力されます。ポインター値は、ポイント先の内容とともに出力されます。3 つの変数はすべて、パラメーターとして `modify` 関数に渡されます。最初の 2 つのパラメーターは値によって渡され、最後のパラメーターは参照によって渡されます。 `modify` 関数の関数プロトタイプは、最後のパラメーターが参照として渡されていることを示します。C++ では、参照により受け渡される参照を除いて、すべてのパラメーターが値により受け渡されることを思い出してください。

`modify` 関数では、3 つのパラメーターの値がすべて変更されます。

- 最初のパラメーターは 0 に設定されます。
- 2 番目のパラメーターが指す値は 7 に設定され、2 番目のパラメーターは 0 に設定されます。
- 3 番目のパラメーターは 0 に設定されます。

新しい値が出力されると、関数はリターンします。制御が `main` に戻されると、ポインターが指す値とともに 3 つの変数の値が再度出力されます。1 番目のパラメーターおよび 2 番目のパラメーターとして渡された変数は、値により渡されているので、 `modify` 関数の影響を受けません。ただし、ポインターが指している値は変更されています。これは、最初の 2 つのパラメーターとは異

なり、最後のパラメーターとして渡された変数は参照により受け渡しされるため、 `modify` 関数によって変更されるからです。

次に、同様のコードを Java 言語で書いた場合を検討しましょう。

リスト 3: Java アプリケーション

```
{
  public static void main(String args[])
  {
    int val;
    StringBuffer sb1, sb2;
    val = 10;
    sb1 = new StringBuffer("apples");
    sb2 = new StringBuffer("pears");
    System.out.println("val is " + val);
    System.out.println("sb1 is " + sb1);
    System.out.println("sb2 is " + sb2);
    System.out.println("");
    System.out.println("calling modify");
    //All parameters passed by value
    modify(val, sb1, sb2);
    System.out.println("returned from modify");
    System.out.println("");
    System.out.println("val is " + val);
    System.out.println("sb1 is " + sb1);
    System.out.println("sb2 is " + sb2);
  }
  public static void modify(int a, StringBuffer r1,
                            StringBuffer r2)
  {
      System.out.println("in modify...");
      a = 0;
      r1 = null;  //1
      r2.append(" taste good");
      System.out.println("a is " + a);
      System.out.println("r1 is " + r1);
      System.out.println("r2 is " + r2);
  }
}
```

このコードの出力結果は、次のようになります。

リスト 4: Java アプリケーションの出力

```
val is 10
sb1 is apples
sb2 is pears
calling modify
in modify...
a is 0
r1 is null
r2 is pears taste good
returned from modify
val is 10
sb1 is apples
sb2 is pears taste good
```

このコードでは、1 つの int と 2 つのオブジェクト参照の 3 つの変数を宣言しています。それぞれの変数は初期値が設定され、出力されます。3 つの変数はすべて、パラメーターとして `modify` メソッドに渡されます。

`modify` メソッドは、3 つパラメーターの値をすべて変更します。

- 最初のパラメーター int は 0 に設定されます。
- 最初のオブジェクト参照 r1 はヌルに設定されます。
- 2 番目のオブジェクト参照 r2 の値は変更されませんが、 `append` メソッドの呼び出しにより、参照先が変更されます。(これは、前述の C++ の例でポインター p に関して行われる処理と似ています)

制御が `main` に戻されると、3 つの変数の値が再度出力されます。予想どおり、int `val` は変更されません。オブジェクト参照 `sb1` も変更されません。 `sb1` は、参照により受け渡される場合は、皆さんが言われるようにヌルになります。ただし、Java プログラム言語はすべてのパラメーターを値により受け渡すので、 `modify` メソッドには `sb1` の参照のコピーが渡されます。//1 で `modify` メソッドが `r1` をヌルに設定すると、 `sb1` 参照のコピーに対しても同様のことが行われます。C++ ではコピーではなく、元のものが処理の対象となる点が異なります。

また、2 番目のオブジェクト参照 `sb2` は、 `modify` メソッドで設定された新しい文字列を出力することに注意してください。 `modify` メソッドの変数 `r2` が参照 `sb2` のコピーである場合でも、同じオブジェクトが参照されます。したがって、コピーされた参照で呼び出されるメソッドが、同じオブジェクトの変更を行います。

スワップ・メソッドの作成

パラメーターの受け渡し方法が分かれば、別の手法を用いて、C++ でスワップ関数を作成することができます。値により渡される、ポインターを使用するスワップ関数は、以下のようになります。

リスト 5: ポインターを使用するスワップ関数

```
#include
#include
void swap(int *a, int *b);
int main (int argc, char** argv)
{
  int val1, val2;
  val1 = 10;
  val2 = 50;
  swap(&val1, &val2);
  return 0;
}
void swap(int *a, int *b)
{
  int temp = *b;
  *b = *a;
  *a = temp;
}
```

参照により受け渡される、参照を使用するスワップ関数の例を示します。

リスト 6: 参照を使用するスワップ関数

```
#include
#include
void swap(int &a, int &b);
int main (int argc, char** argv)
{
  int val1, val2;
  val1 = 10;
  val2 = 50;
  swap(val1, val2);
  return 0;
}
void swap(int &a, int &b)
{
  int temp = b;
  b = a;
  a = temp;
}
```

予想したとおり、いずれの C++ コードの例でも値がスワップされます。Java アプリケーションが参照による受け渡しを使用する場合、以下のスワップ・メソッドは C++ の例と同じように機能します。

リスト 7: C++ と同様の参照による受け渡しが行われる場合の Java スワップ関数

```
class Swap
{
  public static void main(String args[])
  {
    Integer a, b;
    a = new Integer(10);
    b = new Integer(50);
    System.out.println("before swap...");
    System.out.println("a is " + a);
    System.out.println("b is " + b);
    swap(a, b);
    System.out.println("after swap...");
    System.out.println("a is " + a);
    System.out.println("b is " + b);
  }
  public static void swap(Integer a, Integer b)
  {
    Integer temp = a;
    a = b;
    b = temp;
  }
}
```

Java アプリケーションはすべてのパラメーターを値により受け渡すので、このコードは正常に処理されず、以下の出力が生成されます。

リスト 8: リスト 7 からの出力

```
before swap...
a is 10
b is 50
after swap...
a is 10
b is 50
```

では、2 つのプリミティブ型または 2 つのオブジェクト参照の値をスワップするには、Java アプリケーションでどのようなメソッドを作成すればよいのでしょう。Java アプリケーションはすべてのパラメーターを値により受け渡すので、スワップを行うことはできません。値をスワップするには、メソッド呼び出しの外側で、インラインによる実行が必要です。

まとめ

私が自分の本の中でこうしたことを書いたのは、細かいことにこだわって、問題を複雑にしようと考えていたからではありません。「Java アプリケーションでは参照による受け渡しが使用される」という思い込みが危険であることを、プログラマーの皆さんに知ってもらうためだったのです。そうした考えを前提として持ってしまうと、前述のようにスワップ・メソッドを作成しても正しく機能せず、その原因がなかなか見付からないことになります。

実のところ、私も最初に、パラメーターがすべて値によって受け渡されるという事実を知ったとき、それを素直に受け入れることはできませんでした。Java アプリケーションには 2 つの型があるので、C++ と同じように、プリミティブ型は値により受け渡され、参照型は参照により受け渡されるはずである、そう思っていたのです。私は、数年間 C++ プログラミングの経験を積んでから、Java プログラミングの世界に入りました。当初は、すべてのものが直観的ではないように思われました。しかし、Java の構造を理解してからは、すべてのものを値によって受け渡すという Java の手法が、より直観的であるように思えるようになりました。Ken Arnold と James Gosling の両氏は、著書「Java プログラム言語、第 2 版 (Java Programming Language, Second Edition)」 の 2.6.1 節でこういっています。「Java にはパラメーターの受け渡しモードが 1 つしかありません。『値による受け渡し』。これのみが使用されるのです。そのため、プログラミングの考え方がよりシンプルになります」。まさにそのとおりだと思います。

関連トピック

- 「 Java プログラム言語、第 2 版 (Java Programming Language, Second Edition) 」(Ken Arnold、James Gosling 共著)。Java の構造およびライブラリー、そして Java 言語の細かなテクニックを活用するための手法が説明されています。