



Passing Parameters By Reference and By Value in Apex

http://res.cloudinary.com/hzxejch6p/image/upload/v1371104089/qbeb4hfwztpdxr5ujpin.pngWe will be making a change to the Apex documentation on the notion of “pass by reference”.  Here is a detailed explanation of why the change is being made.

The current Apex documentation says that non-primitive variables are passed by reference:

*“Also note that all primitive variables are passed by value, while all non-primitive data types are passed by reference.”*

Strictly speaking, this is not true.  The new text will read:

*“In Apex, all primitive data type arguments, such as Integer or String, are passed into methods by value. This means that any changes to the arguments exist only within the scope of the method. When the method returns, the changes to the arguments are lost.*

*Non-primitive data type arguments, such as sObjects, are also passed into methods by value. This means that when the method returns, the passed-in argument still references the same object as before the method call, and can’t be changed to point to another object. However, the values of the object’s fields can be changed in the method.”*

The behavior nearly always appears like non-primitives are passed by reference, and you would rarely notice that they are actually not passed by reference.  However, when you act on the variable itself in the method, like calling “new” on it, you will notice that things do not behave in pass-by-reference fashion.

We’ll illustrate this with some code samples.  But first, let’s step back and define these terms.

**“Pass by value”** means when a method is called, a second copy of the parameter variable is made in memory, and this copy is passed to the method as a parameter.  This copy can be modified in the method, but the original variable in the caller is unchanged, and the changes to the copy are lost upon return.

**“Pass by reference”** means when a method is called, that actual variable is passed to the method.

As described in the new developer’s guide text, Apex is not passing anything by reference.  It is passing the reference (i.e., the memory address) by value, which allows the developer to change fields on the object and call methods (like list.add()) on the object.

Let’s illustrate with some code.  First, an example where we get apparent pass-by-reference behavior:

List fillMe = new List();

PassByWhatNow.reference(fillMe);

System.assertEquals(fillMe.size(),5); //five items, as expected

/\* calls this method

public static void reference(List m) {

List la = [select id, name from Account limit 5];

for (Account a : la) m.add(a);

}

\*/

In this example, the parameter passed to the function, behind the scenes, is a pointer to the fillMe collection, passed by value.  That is to say, we took the memory address and copied it – but it is the same memory address number in both places.  When the add() function is called on the copy, that same memory location is modified by adding the specified items to the list.  So when we return to the calling program, the modifications made in the static function are visible.  So far, so good.

Now let’s look at an example where things break down.

Again, we pass a copy of the list’s memory address to the function.

List createMe = new List();

PassByWhatNow.referenceNew(createMe);

System.assertEquals(createMe.size(),0); //nothing in the list

/\* calls this method

public static void referenceNew(List m) {

m = new List([select id, name from Account limit 5]);

}

\*/

However, this time, the function is *initializing the variable* anew. When the “new” function is called, new memory is allocated.  This is the big difference.

The **copy** of the memory address passed as a parameter is overwritten with the address of the newly allocated memory.  The **original,** however, is not overwritten.  That change in address is **not known to the caller**!

Any subsequent actions done in the method are on that new memory address rather than the original object address.  These changes, including the addition of the Accounts to the list, can not be available to the caller, since it only knows of the original address.  No actions were done on this address; the list on the caller is therefore unchanged, still empty as it was when the call was made.

While Apex usually does behave as if we’re passing variables by reference, we’re not truly passing by reference.  There are a few situations where the distinction is important, as detailed here.  Now you know more, and you are a stronger developer.

Happy coding!

