Java Refrences and Garbage Collection

# Different Java refrences

Many java developers are not aware that java has mainly 4 types of references.

1. Strong Reference
2. Weak Reference.
3. Soft Reference.
4. Phantom Reference.

But why there are different types of reference? What will be there usage?

To understand that I will take an Example,

Suppose in an Application, frequently one needs to fetch data from a MASTER TABLE. As a clever developer certainly he/she don’t want every time application will hit the database, it will degrade the performance.

Obviously choice is Cache. Cache is a class (Internally a Map) and first application will hit the cache to check data is available there else hit the database and put the entry in cache so next time it can be found in cache to skip database call.

**Is it going to improve the performance?**

It will depend on situation, If Master table has less entry this will work fine and certainly increase the performance. But if Master Table has huge entries it will create problem as time to time Cache map is growing and load entries from Master table. Now instead of providing better performance it will be the cause of Out of memory. Imagine a situation where all rows in this huge master table has been loaded to cache. Think about the size of cache it will take most of the JVM memory.

**So what should we do?**

One thing we can do restrict the number of entries in the Cache. And delete the old entries from the cache. I don’t go to the implementation but think is it going to solve the problem?

It will partially solve the problem but it will take constant memory in JVM. Although some of the Object in cache is not used for a long time.

**What will be the Ideal solution?**

The Ideal solution would be if we can make such type of cache which is dynamic in nature it will growing and shrinking as per need.

So we need some kind of technique where we can delete those entries which will not use for a long time and sits in cache.

To achieve it Java, provide different type of reference in ***java.lang.ref*** package.

**Strong Reference**: We use Strong reference in java everywhere. Crete an object then assigns it to a reference. Note that if object has a strong reference this object is never be garbage collected.

**Example:**

HelloWorld hello = new HelloWorld();

Here hello is strong reference to HelloWorld Object.

**Soft Reference**: If an Object has no Strong reference but has a Soft Reference then garbage collector reclaims this Object’s memory when there is not enough memory garbage collection needs to freed up some memory. In spite of soft reference garbage collector reclaims this Object’s memory. To get a Strong reference from soft reference one can invoke get() method. If the Object is not garbage collected it returns the object else return null.

**Weak Reference:** If an Object has no Strong reference but has a Weak Reference then garbage collector reclaims this Object’s memory although there is enough memory. To get a Strong reference from soft reference one can invoke get() method. If the Object is not garbage collected it return the object else return null.

**Phantom Reference**: If an Object does not have aforesaid references then may it has phantom references. Phantom references can’t be accessed directly. Using get() method it always return null. So what will be use of such reference? Think an object can referred by a strong reference in finalize block so to garbage collect this Object need second time to run the GC. But by phantom reference only one GC execution will suffice to GCed the same.

**Example of different type of reference.**

**package** com.example.reference;

**import** java.lang.ref.PhantomReference;

**import** java.lang.ref.ReferenceQueue;

**import** java.lang.ref.SoftReference;

**import** java.lang.ref.WeakReference;

**public** **class** ReferenceExample {

**private** String status ="Hi I am active";

**public** String getStatus() {

**return** status;

}

**public** **void** setStatus(String status) {

**this**.status = status;

}

@Override

**public** String toString() {

**return** "ReferenceExample [status=" + status + "]";

}

**public** **void** strongReference()

{

ReferenceExample ex = **new** ReferenceExample();

System.*out*.println(ex);

}

**public** **void** softReference()

{

SoftReference<ReferenceExample> ex = **new** SoftReference<ReferenceExample>(getRefrence());

System.*out*.println("Soft refrence :: " + ex.get());

}

**public** **void** weakReference()

{

**int** counter=0;

WeakReference<ReferenceExample> ex = **new** WeakReference<ReferenceExample>(getRefrence());

**while**(ex.get()!=**null**)

{

counter++;

System.*gc*();

System.*out*.println("Weak reference deleted after:: " + counter + ex.get());

}

}

**public** **void** phantomReference() **throws** InterruptedException

{

**final** ReferenceQueue queue = **new** ReferenceQueue();

PhantomReference<ReferenceExample> ex = **new** PhantomReference<ReferenceExample>(getRefrence(),queue);

System.*gc*();

queue.remove();

System.*out*.println("Phantom reference deleted after");

}

**private** ReferenceExample getRefrence()

{

**return** **new** ReferenceExample();

}

**public** **static** **void** main(String[] args) {

ReferenceExample ex = **new** ReferenceExample();

ex.strongReference();

ex.softReference();

ex.weakReference();

**try** {

ex.phantomReference();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

Output :

ReferenceExample [status=Hi I am active]

Soft refrence :: ReferenceExample [status=Hi I am active]

Weak reference deleted after:: 1null

Phantom reference deleted after

Look at , softReference() method here we create a soft reference as memory is available so reference is not Gced.

For weakRefrence(), reference is GCed immediately as no strong reference is there.

Same for Phantom reference.

**Comparison:**

****