What precisely is the algorithm used by java.lang.Object's hashCode

Ask Question

What is the algorithm used in the ***JVM*** to implement `java.lang.Object` 's implicit `hashCode()` method?

[***OpenJDK*** or ***Oracle JDK*** are preferred in the answers].

java openjdk

edited Jul 23 '14 at 23:47
user3667171

asked Jul 23 '14 at 22:59

djhaskin987
5,337 33 60

2 Just curious... Since Java is an open-source, what stopped you from looking yourself? – PM 77-1 Jul 23 '14 at 23:01

5 @PM77-1 probably fact that this is `native` method and not many people know where to look for its implementation. – Pshemo Jul 23 '14 at 23:03

2 See stackoverflow.com/questions/17977495/… - which has links to the native source. However, after reading the source I am even more confused. – user2864740 Jul 23 '14 at 23:15

It should be noted that the hashcode algorithms for other objects are all over the map. String, last I looked, uses a hash of the first N characters of the string data. Container objects will use some sort of sum of the hashes of the contained objects. – Hot Licks Jul 24 '14 at 0:16

Duly noted. I understand, but I'm still curious about object in particular. – djhaskin987 Jul 24 '14 at 1:29

1 Answer

It's implementation dependant (and heavily so, the algorithm is *entirely* up to the implementation, as long as it's consistent.) However, as per the answer here, you can see the native source file where the hash is generated in OpenJDK 7 (look at the `get_next_hash()` function), which actually specifies a number of possible algorithms in this particular release:

```
// Possibilities:
// * MD5Digest of {obj,stwRandom}
// * CRC32 of {obj,stwRandom} or any linear-feedback shift register function.
// * A DES- or AES-style SBox[] mechanism
// * One of the Phi-based schemes, such as:
//   2654435761 = 2^32 * Phi (golden ratio)
//   HashCodeValue = ((uintptr_t(obj) >> 3) * 2654435761) ^ GVars.stwRandom ;
// * A variation of Marsaglia's shift-xor RNG scheme.
// * (obj ^ stwRandom) is appealing, but can result
//   in undesirable regularity in the hashCode values of adjacent objects
//   (objects allocated back-to-back, in particular).  This could potentially
//   result in hashtable collisions and reduced hashtable efficiency.
//   There are simple ways to "diffuse" the middle address bits over the
//   generated hashCode values
//
```

As already stated, the default algorithm is to simply go with a random number, however an interesting comment further down states:

```
// Marsaglia's xor-shift scheme with thread-specific state
// This is probably the best overall implementation -- we'll
// likely make this the default in future releases.
```

This is similar to the `(obj ^ stwRandom)` approach mentioned in the above list of possibilities, but it gets around the unwanted regularities associated with objects allocated quickly in succession by tying "thread-specific state" information into the hash also - so if two objects happened to be allocated at a very similar time due to them being allocated on two separate threads executing simultaneously, the discrete threading information fed into the hash should still ensure discrete enough hashes will be generated.

edited May 23 '17 at 12:02

Community ♦

1 1

answered Jul 23 '14 at 23:23

Michael Berry

40.9k 16 106 161