


Home

PUBLIC


 Stack Overflow

Tags

Users

Jobs

TEAMS

 Create Team

Why does ConcurrentHashMap prevent null keys and values?

Ask Question

The JavaDoc of [ConcurrentHashMap](#) says this:

Like `Hashtable` but unlike `HashMap`, this class does *not* allow `null` to be used as a key or value.

My question: why?

2nd question: why doesn't `Hashtable` allow `null`?

I've used a lot of `HashMap`s for storing data. But when changing to `ConcurrentHashMap` I got several times into trouble because of `NullPointerException`s.

[java](#) [concurrenthashmap](#)

edited Mar 30 '09 at 19:40

asked Mar 30 '09 at 19:12



Marcel

1,553 5 20 33

- 1 +1 for the great discussions and the fact it made some of us(O.K. Me) reevaluate our thoughts. – [WolfmanDragon](#) Mar 30 '09 at 20:46
- 1 I think it's an extremely annoying inconsistency. `EnumMap` doesn't allow `null` either. There's obviously no technical limitation that disallows `null` keys. for a `Map<K, V>`, simply a `V`-typed field will provide support for `null` keys (probably another boolean field if you want to differentiate between `null` value and no value). – [RAY](#) May 13 '11 at 15:41

From the author of [ConcurrentHashMap](#) himself (Doug Lea):

The main reason that nulls aren't allowed in ConcurrentMaps (ConcurrentHashMaps, ConcurrentSkipListMaps) is that ambiguities that may be just barely tolerable in non-concurrent maps can't be accommodated. The main one is that if `map.get(key)` returns `null`, you can't detect whether the key explicitly maps to `null` vs the key isn't mapped. In a non-concurrent map, you can check this via `map.containsKey(key)`, but in a concurrent one, the map might have changed between calls.

answered Feb 15 '12 at 17:23



[Bruno](#)

89.8k 9 204 289

4 Nice Explanation. – [kanaparthikiran](#) Apr 5 '15 at 6:42

Thank you, but what about having null as the key? – [AmitW](#) Sep 25 '17 at 8:34

I believe it is, at least in part, to allow you to combine `containsKey` and `get` into a single call. If the map can hold nulls, there is no way to tell if `get` is returning a null because there was no key for that value, or just because the value was null.

Why is that a problem? Because there is no safe way to do that yourself. Take the following code:

```
if (m.containsKey(k)) {  
    return m.get(k);  
}
```

Normally you would solve that by synchronizing, but with a concurrent map that of course won't work. Hence the signature for `get` had to change, and the only way to do that in a backwards-compatible way was to prevent the user inserting null values in the first place, and continue using that as a placeholder for "key not found".

answered Jan 21 '11 at 13:26



[Alice Purcell](#)

8,479 4 35 50

Josh Bloch designed `HashMap` ; Doug Lea designed `ConcurrentHashMap` . I hope that isn't libelous. Actually I think the problem is that nulls often require wrapping so that the real null can stand for uninitialized. If client code requires nulls then it can pay the (admittedly small) cost of wrapping nulls itself.

edited Jul 29 '16 at 2:21



[SkyWalker](#)

15.9k 3 32 70

answered Mar 30 '09 at 20:01



[Tom Hawtin - tackline](#)

121k 25 177 264

You can't synchronize on a null.

Edit: This isn't exactly why in this case. I initially thought there was something fancy going on with locking things against concurrent updates or otherwise using the Object monitor to detect if something was modified, but upon examining [the source code](#) it appears I was wrong - they lock using a "segment" based on a bitmask of the hash.

In that case, I suspect they did it to copy Hashtable, and I suspect Hashtable did it because in the relational database world, null != null, so using a null as a key has no meaning.

edited Mar 30 '09 at 20:43

answered Mar 30 '09 at 19:18



[Paul Tomblin](#)

133k 47 275 366

on. – [Paul Tomblin](#) Mar 30 '09 at 19:26

1 Why isn't there a special Object internally that could be used for synchronizing null values? e.g. "private Object NULL = new Object();" I think I've seen this before... – [Marcel](#) Mar 30 '09 at 19:41

What other sorts of locking do you mean? – [Tobias Müller](#) Mar 30 '09 at 19:50

1 I have wondered this myself, @Paul your explanation opened a light from heaven for me. Thanks. +1 – [WolfmanDragon](#) Mar 30 '09 at 19:51

ConcurrentHashMap is thread-safe. I believe that not allowing null keys and values was a part of making sure that it is thread-safe.

answered Mar 30 '09 at 19:17



[Kevin Crowell](#)

7,949 4 29 50

I guess that the following snippet of the API documentation gives a good hint: "This class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details."

They probably just wanted to make ConcurrentHashMap fully compatible/interchangeable to Hashtable . And as Hashtable does not allow null keys and values..

answered Mar 30 '09 at 19:29



[Tobias Müller](#)

198 1 7

1 And why doesn't Hashtable support null? – [Marcel](#) Mar 30 '09 at 19:39

From looking at its code, I don't see an obvious reason why Hashtable does not allow null values. Maybe it was just an API decision from back when the class was created?! HashMap has some special handling for the null-case internally which Hashtable does not. (It always throws NullPointerException.) – [Tobias Müller](#) Mar 30 '09 at 19:49