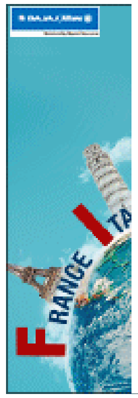
JavaMadeSoEasy.com (JMSE)

Search This Blog

Search



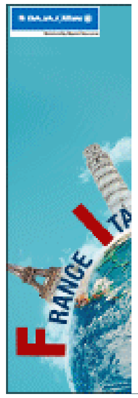
ConcurrentHashMap in java - with Segments formation in detail with diagram

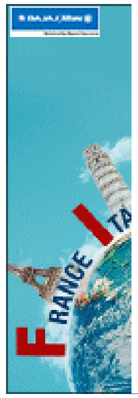
You are here : Home / Core Java Tutorials / Collection framework Tutorial in java

In this Collection framework tutorial we will learn what is java.util.concurrent.ConcurrentHashMap in Collection framework in java.

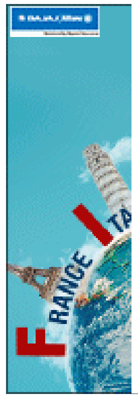
Contents of page :

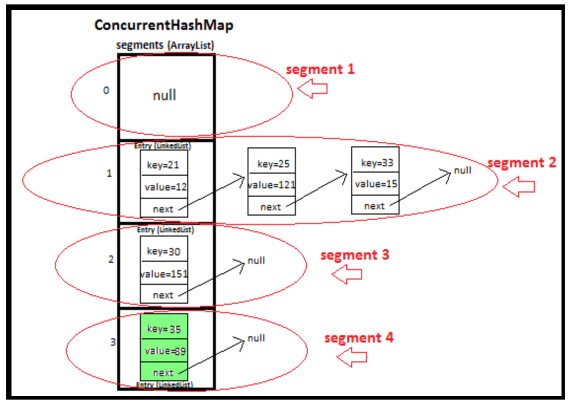
- Generics
- Hashing
- Interview questions
- iText Pdf tutorial
- Java 7
- Java 8
- Java 9
- Java QUIZ
- JavaScript
- JDBC
- JUNIT
- JVM
- Level1 programs (beginners)
- Level2 programs (intermediate)
- Level3 programs (advanced)
- Linked List
- Linux
- Matrix programs
- MongoDB
- MongoDB Java
- MsSql
- MultiThreading
- MySql
- Notepad++
- Oracle





- OutOfMemoryError
- Overriding equals and hashCode
- PostgreSQL
- Producer Consumer problem/pattern
- Pyramid generation
- RegEx
- Serialization
- Thread Concurrency
- Threads
- TUTORIALS
- Windows





1) java.util.concurrent.ConcurrentHashMap in java

java.util.concurrent.**ConcurrentHashMap** is implementation of the java.util.Map interface in java. java.util.concurrent.**ConcurrentHashMap** enables us to store data in key-value pair form. Insertion order of key-value pairs is not maintained. *ConcurrentHashMap* is synchronized in java.

2) What is hierarchy of ConcurrentHashMap in java?

```
-java.lang.Object
 -java.util.AbstractMap
  -java.util.concurrent.ConcurrentHashMap
```

For more detailed hierarchy information read : Map hierarchy in java

Subscribe-Our exclusive posts for FREE. Stay on top! Join 2693+ subscribers

Email address.. Submit

Join our Groups>

- FACEBOOK
- LINKED IN

3) Creating java.util.concurrent.ConcurrentHashMap (using constructor)

Constructs a new ConcurrentHashMap, Its **initial capacity** is **16**. And **load factor** is **0.75** (We'll discuss it later in post)

```
Map<Integer,String> concurrentHashMap=new ConcurrentHashMap<Integer,String>();
```

Defining **ConcurrentHashMap<Integer,String>** means key can of Integer type and value can be String type only, using any other type will cause compilation error.

4) What is ***concurrency level*** in java? What is default concurrency level of java.util.concurrent.ConcurrentHashMap?

Concurrency level tells how many threads can access ConcurrentHashMap concurrently, default **concurrency level** of ConcurrentHashMap is **16**.

```
new ConcurrentHashMap()
```

Creates a new ConcurrentHashMap with concurrency level of 16.

5) How ***ConcurrentHashMap works***? Can 2 threads on same ConcurrentHashMap object access it concurrently in java?

ConcurrentHashMap is divided into different **segments** based on concurrency level. So different threads can access different **segments** concurrently in java.

All Labels

Algorithm(34) Apache Tomcat Server(4) Arrays Java(30) Autoboxing(5) Basic java programs for beginners(15) Binary Trees(6) Collection Framework(68) Collection programs(105) Collections implementation(16) Comparable and Comparator program(22) Core Java(1032) core java Basics(38) Core java Conversions(21) Core Java Differences(11) Core Java Tutorials(12) CRUD MongoDB java(5) CRUD operations MongoDB(3) Custom implementation of Data Structures(11) Data Structure(26) Database(1) Database Tutorials(3) Date tutorials(11) DeSerialization(19) eclipse(27) Exceptions(71) File IO/File handling(71) Garbage collection in java(43) Generics(5) Hashing(8) Hibernate(1) Interview questions(14) iText Pdf tutorial(45) Java 4(1) java 5(3) Java 7(32) Java 8(80) Java 9(12)

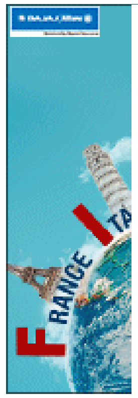
Can threads read the segment of *ConcurrentHashMap* locked by some other thread in java?
Yes. When thread locks one segment for updation it does not block it for retrieval (done by get method) hence some other thread can read the segment (by get method), but it will be able to read the data before locking.

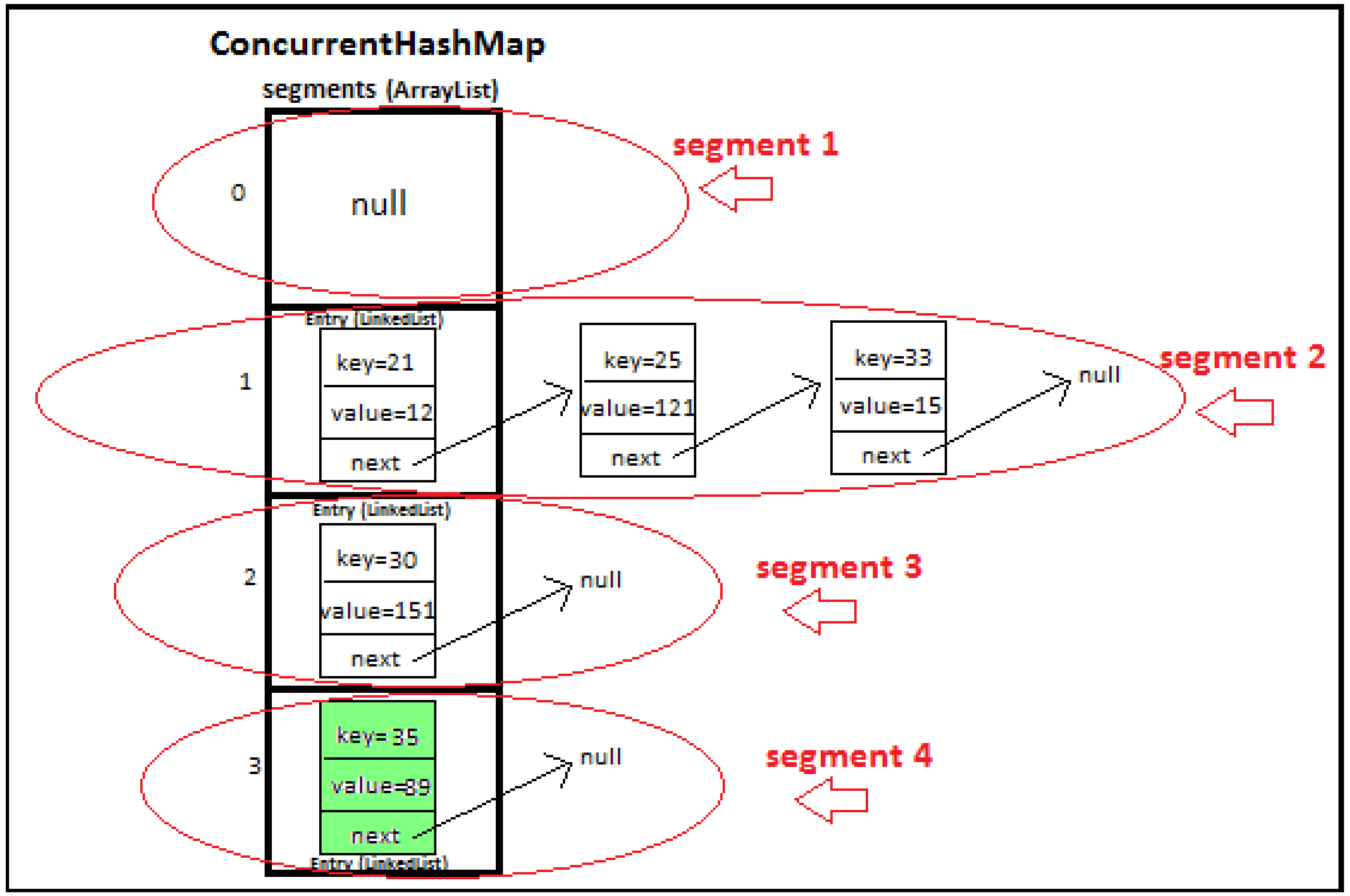
For operations such as putAll concurrent retrievals may reflect removal of only some entries.
For operations such as clear concurrent retrievals may reflect removal of only some entries.

6) **Segments** in ConcurrentHashMap with diagram in java >

we have ConcurrentHashMap with **4 segments -**
(Diagram shows how **segments** are formed in ConcurrentHashMap)

Java keywords(11) Java Mcq(Multiple choice questions)(5) Java Programs(8) Java QUIZ(7) JavaScript(2) JDBC(74) JUNIT(7) JVM(22) Level1 programs (beginners)(34) Level2 programs (intermediate)(23) Level3 programs (advanced)(13) Linux(6) Matrix programs(10) MongoDB(87) MongoDB Java(20) MsSql(1) MultiThreading(97) MySql(7) Notepad++(6) Oracle(18) OutOfMemoryError(13) Overriding equals and hashCode(11) Pattern generating(14) PostgreSQL(1) Producer Consumer problem/pattern(10) Pyramid generation(14) Recursion(10) RegEx(7) Serialization(20) Serialization top interview questions and answers in java(18) Thread Concurrency(35) Threads(73) TUTORIALS(22) Windows(8)





Now let's form few questions to clear your doubts (based on above diagram) in java >

ConcurrentHashMap Question 1 : What will happen **map.put(25,12)** is called and some other thread concurrently calls **map.get(25)**?

Answer : When **map.put(25,12)** is called **segment 2** will be locked,
key=25 also lies in **segment 2**, *When thread locks one segment for updation it does not block it for retrieval hence some other thread can read the same segment, but it will be able to read the data*

Popular Posts of JavaMadeSoEasy

- HashMap Custom implementation in java - How HashMap works internally with diagrams and full program
- Collection Quiz in Java - MCQ - Multiple choice questions
- CORE JAVA - Top 120 most interesting and important interview questions and answers in core java
- THREADS - Top 80 interview questions and answers in java for freshers and experienced(detailed explanation with programs) Set-1 > Q1- Q60
- Core Java Tutorial in detail with diagram and programs - BEST EXPLANATION EVER
- COLLECTION - Top 100 interview questions and answers in java for fresher and experienced in detail - Set-1 > Q1- Q50

before locking (hence **map.get(25)** will return **121**)

ConcurrentHashMap Question 2 : What will happen **map.put(25,12)** is called and some other thread concurrently calls **map.get(33)**?

Answer : When **map.put(25,12)** is called **segment 2** will be locked,
key=33 also lies in **segment 2**, *When thread locks one segment for updation it does not block it for retrieval hence some other thread can read the same segment, but it will be able to read the data before locking* (hence **map.get(33)** will return **15**)

ConcurrentHashMap Question 3 : What will happen **map.put(25,12)** is called and some other thread concurrently calls **map.put(33,24)**?

Answer : When **map.put(25,12)** is called **segment 2** will be locked,
key=33 also lies in **segment 2**, *When thread locks one segment for updation it does not allow any other thread to perform updations in same segment until lock is not released on segment*.
hence **map.put(33,24)** will have to wait for **map.put(25,12)** operation to release lock on segment.

ConcurrentHashMap Question 4 : What will happen **map.put(25,12)** is called and some other thread concurrently calls **map.put(30,29)**?

Answer : When **map.put(25,12)** is called **segment 2** will be locked,
but **key=30** lies in **segment 3**.
*Both the kays lies in different segments, **hence both operations can be performed concurrently.***

ConcurrentHashMap Question 5 : What will happen updations (put/remove) are in process in certain segments and new key-pair have to be put/remove in same segment ?

Answer : When updations are in process *thread locks the segment and it does not allow any other thread to perform updations (put/remove) in same segment until lock is not released on segment*.

*Let's **summarize** above section >*

What operations lock ConcurrentHashMap segment & what operations are allowed when ConcurrentHashMap segment is locked in java >

- Find sum of both diagonals in matrix - program in java
- LinkedHashMap Custom implementation in java - How LinkedHashMap works internally with diagrams and full program
- Thread/multi threading Quiz in Java - MCQ - Multiple choice questions
- EXCEPTIONS - Top 60 interview questions and answers in java for fresher and experienced - detailed explanation with diagrams Set-1 > Q1- Q25

- *thread locks one segment for updation (put/remove) & it does not block it for retrieval (get) hence some other thread can read the same segment, but it will be able to read the data before locking*
- *It's important to know get operations does not lock any segment.*

*7) ConcurrentHashMap **putIfAbsent** method in java*

*Definition of **putIfAbsent** method in java >*

```
public V putIfAbsent(K key, V value)
```

*What do **putIfAbsent** method do>*

If map does not contain specified **key**, put specified **key-value** pair in map and return null in java.
If map already contains specified **key**, return value corresponding to specified **key**.

***putIfAbsent** method is equivalent to writing following code in java >*

```
        synchronized (map){
    if (!map.containsKey(key))
      return map.put(key, value);
     else
      return map.get(key);
  }
```

***Program 1** to use java.util.concurrent.ConcurrentHashMap's putIfAbsent method in java >*

```
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;
```

JavaMadeSoeasy Archive

- ► 2018 (17)
- ► 2017 (211)
- ► 2016 (240)
- ► 2015 (722)
  - ► December (61)
  - ► November (96)
  - ► October (14)
  - ► September (13)
  - ► August (73)
  - ► July (54)
  - ► June (29)
  - ► May (53)
  - ► April (109)

```
/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class ConcurrentHashMapTest {
   public static void main(String args[]) {

      ConcurrentMap<Integer, String> concurrentHashMap =
                        new ConcurrentHashMap<Integer, String>();
      concurrentHashMap.put(1, "javaMadeSoEasy");
      System.out.println("concurrentHashMap : "+concurrentHashMap);

      System.out.println("\n putIfAbsent method >> "+
                  concurrentHashMap.putIfAbsent(1, "ankit"));
      System.out.println("concurrentHashMap : "+concurrentHashMap);

      System.out.println("\n putIfAbsent method >> "+
                  concurrentHashMap.putIfAbsent(2, "audi"));
      System.out.println("concurrentHashMap : "+concurrentHashMap);

   }
}

/*OUTPUT

concurrentHashMap : {1=javaMadeSoEasy}

 putIfAbsent method >> javaMadeSoEasy
concurrentHashMap : {1=javaMadeSoEasy}

 putIfAbsent method >> null
concurrentHashMap : {2=audi, 1=javaMadeSoEasy}

*/
```

concurrentHashMap.putIfAbsent(1, "ankit") > returned javaMadeSoEasy because map was already having that key in java.

- ▶ March (93)
- ▶ February (99)
- ▶ January (28)

concurrentHashMap.putIfAbsent(2, "audi") > putted specified key-value pair in map and returned null because map wasn't having that key in java.

Program 2 *to create method that provides* ***functionality similar to putIfAbsent method of ConcurrentHashMap*** *and to be used with java.util.HashMap in java >*

```
import java.util.HashMap;
import java.util.Map;

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class HashMapTest {

   static Map<Integer, String> map = new HashMap<Integer, String>();

   public static void main(String args[]) {

       map.put(1, "javaMadeSoEasy");
       System.out.println("hashMap : "+map);

       System.out.println("\n functionalityOfPutIfAbsent method >> "+
                    functionalityOfPutIfAbsent(1, "ankit"));
       System.out.println("hashMap : "+map);

       System.out.println("\n functionalityOfPutIfAbsent method >> "+
                    functionalityOfPutIfAbsent(2, "audi"));
       System.out.println("hashMap : "+map);

   }


   /**
    * Method is created to be used with HashMap, And
    * method provides functionality similar to putIfAbsent
    * method of ConcurrentHashMap.
    */
    public static synchronized String functionalityOfPutIfAbsent(Integer key,String value){
            if (!map.containsKey(key))
              return map.put(key, value);
             else
              return map.get(key);
    }

}

/*OUTPUT

hashMap : {1=javaMadeSoEasy}

 functionalityOfPutIfAbsent method >> javaMadeSoEasy
hashMap : {1=javaMadeSoEasy}

 functionalityOfPutIfAbsent method >> null
hashMap : {1=javaMadeSoEasy, 2=audi}

*/
```

Please note **functionalityOfPutIfAbsent** method is **synchronized,** because this method provides same functionality as that of **ConcurrentHashMap's putIfAbsent** method and all methods in **ConcurrentHashMap** are **synchronized**.

functionalityOfPutIfAbsent(1, "ankit") > returned javaMadeSoEasy because map was already having that key in java.

functionalityOfPutIfAbsent(2, "audi") > putted specified key-value pair in map and returned null because map wasn't having that key in java.

8) put element in java.util.concurrent.ConcurrentHashMap

put(K key, V value)

Method allows you put specified *key-value pair* in ConcurrentHashMap. If the map already contains a mapping for the *key*, the old *value* is replaced.

```
concurrentHashMap.put(11, "audi");
```

9) get elements from ConcurrentHashMap in java

get(Object key)

Method returns value corresponding to *key*.

Method returns null if map does not contain *key.*

```
concurrentHashMap.get(2);
```

Method returns element on 2nd index.

10) Remove element from ConcurrentHashMap in java

remove(Object key)

Method removes *key*-value pair from ConcurrentHashMap.

```
concurrentHashMap.remove(11);
```

11) contains element in ConcurrentHashMap

contains(Object object)

Method returns true if HAshmap contains specified on specified index.

```
concurrentHashMap.get(2);
```

Method returns element on 2nd index.

12) Size of java.util.concurrent.ConcurrentHashMap in java

size()

Method returns size of **ConcurrentHashMap.**

```
System.out.println(concurrentHashMap.size());
```

will print size of concurrentHashMap.

13) Iterate over java.util.concurrent.ConcurrentHashMap in java

Before iterating we will put 3 key-value pairs in concurrentHashMap.

```
    concurrentHashMap.put(11, "audi");
  concurrentHashMap.put(21, "bmw");
  concurrentHashMap.put(31, "ferrari");
```

13.1) Iterate over keys in java -

concurrentHashMap.keySet().iterator() method returns iterator to iterate over keys in ConcurrentHashMap.

```
Iterator<Integer> keyIterator=concurrentHashMap.keySet().iterator();
while(keyIterator.hasNext()){
 System.out.println(keyIterator.next());
}
```

```
/*OUTPUT
21
11
31
*/
```

Iteration using enhanced for loop in java.

concurrentHashMap.keySet() returns set of keys.

```
Set<Integer> keySet=concurrentHashMap.keySet();
for(Integer key :keySet){
  System.out.println(key);
}
```

iterator returned by ConcurrentHashMap over key is fail-safe. **Means any structural modification made to ConcurrentHashMap like adding or removing elements during Iteration will not throw any Exception**.

```
Iterator<String> iterator=concurrentHashMap.iterator();
while(iterator.hasNext()){
  System.out.println(iterator.next());
  concurrentHashMap.put(4, "d");
}
```

key-value has been added (map didn't contained this key previously) during iteration and no exception is thrown.

13.2) Iterate over values in java -

concurrentHashMap.values().iterator() method returns iterator to iterate over keys in ConcurrentHashMap.

```
Iterator<String> valueIterator=concurrentHashMap.values().iterator();
while(valueIterator.hasNext()){
 System.out.println(valueIterator.next());
}

/*OUTPUT
bmw
audi
ferrari
*/
```

Iteration using enhanced for loop.

concurrentHashMap.values() returns collection of values.

```
Collection<String> collection=concurrentHashMap.values();
for(String value :collection){
 System.out.println(value);
}
```

iterator returned by ConcurrentHashMap over values is fail-safe**. Means any structural modification made to ConcurrentHashMap like adding or removing elements during Iteration will not throw any Exception**.

```
Iterator<String> iterator=concurrentHashMap.iterator();
while(iterator.hasNext()){
 System.out.println(iterator.next());
 concurrentHashMap.put(5, "d");
}
```

key-value has been added (map didn't contained this key previously) during iteration and no exception is thrown.

13.3) Iterate over Entry in java-

concurrentHashMap.entrySet().iterator() method returns iterator to iterate over keys in ConcurrentHashMap in java.

```
Iterator<Entry<Integer, String>> entryIterator=concurrentHashMap.entrySet().iterator();
while(entryIterator.hasNext()){
   System.out.println(entryIterator.next());
}

/*OUTPUT
21=bmw
11=audi
31=ferrari
*/
```

Iteration using enhanced for loop.

concurrentHashMap.entrySet() returns collection of values.

```
Set<Entry<Integer, String>> entrySet=concurrentHashMap.entrySet();
for(Entry<Integer, String> entry:entrySet){
      System.out.println(entry);
}
```

iterator returned by ConcurrentHashMap over entry is fail-safe. **Means any structural modification made to ConcurrentHashMap like adding or removing elements during Iteration will not throw any Exception**.

14) Some other important methods of java.util.concurrent.ConcurrentHashMap

isEmpty() method returns true if this map contains any key-value pair in java.

clear() method removes all key-value pair from map in java.

15) Complexity of methods in ConcurrentHashMap in java

Operation/ method	**Worst case**	**Best case**
put(K key, V value)	O(n)	O(1)
get(Object key)	O(n)	O(1)

*16) **10 features** of java.util.concurrent.ConcurrentHashMap*

1. **ConcurrentHashMap** enables us to store data in key-value pair form in java.

2. **ConcurrentHashMap** is implementation of the java.util.**map** interface in java.

3. **Duplicate key**- ConcurrentHashMap does not allows to store duplicate keys. If the map already contains a mapping for the key, the old value is replaced in java.

4. **Null elements -** ConcurrentHashMap does **not allow to store null key or null value**. Any attempt to store null key or value in ConcurrentHashMap throws runtimeException (NullPointerException).

5. **Insertion order -** ConcurrentHashMap does not maintains insertion order in java.

   Example in java-

```
Let’s say we add 3 elements in concurrentHashMap
concurrentHashMap.put(1,"ind");
concurrentHashMap.put(2,"aus");
concurrentHashMap.put(3,"sa");

On displaying insertion order will not be maintained i.e.
3,sa
2,aus
1,ind
```

6. **synchronized -** ConcurrentHashMap is synchronized in java.

7. **Performance -** ConcurrentHashMap is synchronized, hence its operations are slower as compared to some unSynchronized implementation of map interface in java.

8. **Provides locking in segments -** *ConcurrentHashMap* is divided into different **segments** based on concurrency level. So different threads can access different **segments** concurrently in java.

9. ***iterator* are *fail-safe* -**
   1. *concurrentHashMap.keySet().iterator()*
   2. *concurrentHashMap.values().iterator()*

3. concurrentHashMap.entrySet().iterator()

all three iterators are ***fail-safe in java.***

10. ***putIfAbsent method is present in ConcurrentHashMap -*** If map does not contain specified **key**, put specified **key-value** pair in map and return null in java.

 If map already contains specified **key**, return value corresponding to specified **key**.

17) When to use java.util.concurrent.ConcurrentHashMap

1. ConcurrentHashMap can be used when we want to store data in key-value pair form in java.

2. ConcurrentHashMap can be used when we don't care about insertion order in java.

3. ConcurrentHashMap can be used when we are working in multithreading environment in java.

4. Hashtable is **obsolete in java 5 i.e. JDK 1.5**, hence it is better to use ConcurrentHashMap than using Hashtable in java.

18) Comparison of performance between HashMap and ConcurrentHashMap

We will **synchronize HashMap and then compare its performance with ConcurrentHashMap**.

We can synchronize HashMap by using Collections's class synchronizedList method in java.

```
Map synchronizedMap = Collections.synchronizedMap(hashMap);
```

Now, no 2 threads can access same instance of map concurrently.
Hence synchronized HashMap's performance is slower as compared to ConcurrentHashMap.

But why we didn't compared HashMap (unSynchronized) with ConcurrentHashMap?
Because performance of unSynchronized collection is always better than some synchronized collection. As, default (unSynchronized) hashMap didn't cause any locking.

19) Comparison of performance between Hashtable and ConcurrentHashMap in java

Hashtable is **obsolete in java 5 i.e. JDK 1.5**, it is better to use ConcurrentHashMap than using Hashtable, because of concurrency level ConcurrentHashMap's performance is better than Hashtable in java.

20) What is Load Factor in java?

Default load factor is 0.75
That means when set will be 75% filled, it's capacity will be doubled in java.

Example in java >
Initially when number of elements is 0, default capacity =16, Load Factor =0.75, ConcurrentHashMap is 0% full in java.

number of elements	***capacity of ConcurrentHashMap***	***Load factor***	***ConcurrentHashMap filled in %age***
0	16	0.75	0%
4	16	0.75	25%
8	16	0.75	50%
11	16	0.75	68.7%

When next element will be added (i.e. 12th element), concurrentHashMap will be 75% filled and capacity will be doubled i.e. from 16 to 32.

12	32	0.75	37.5%

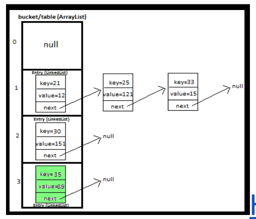
So in this Collection framework tutorial we learned what is java.util.concurrent.ConcurrentHashMap in Collection framework in java.

Having any doubt? or you you liked the tutorial! Please comment in below section.
*Please express your love by liking **JavaMadeSoEasy.com** (**JMSE**) on **facebook**, following on **google+** or Twitter.*

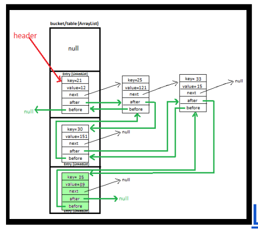
RELATED LINKS>

ArrayList in java

HashSet in java



HashMap Custom implementation



LinkedHashMap Custom implementation

ConcurrentHashMap in java - with Segments formation in detail with diagram

IdentityHashMap in java with program

WeakHashMap in java

EnumMap in java with program

Basic Collection - All properties in tabular form >

Collection - List, Set and Map all properties in tabular form

Collection hierarchy >

Map hierarchy in java - Detailed - HashMap, Hashtable, ConcurrentHashMap, LinkedHashMap, TreeMap, ConcurrentSkipListMap, IdentityHashMap, WeakHashMap, EnumMap classes

ConcurrentModificationException , Fail-fast and Fail-safe >

ConcurrentModificationException, Fail-fast and Fail-safe in detail in java

Important Similarity and Differences >

Map Differences >

HashMap and Hashtable - Similarity and Differences

HashMap and ConcurrentHashMap - Similarity and Differences

HashMap vs Hashtable vs LinkedHashMap vs TreeMap - Differences

HashMap vs IdentityHashMap - Similarity and Differences with program

Important Similarity and Differences Collection classes in concurrent and non-concurrent packages >

TreeSet vs ConcurrentSkipListSet - Similarity and Differences with program

TreeMap vs ConcurrentSkipListMap - Similarity and Differences with program

Top Collection Interviews question and answers >

COLLECTION - Top 50 interview questions and answers in java for fresher and experienced

HashMap Programs >

HashMap - put, get, containsKey , containsValue, remove, size methods

HashMap - Iterate on keys by obtaining keySet, Iterate on values by obtaining values, Iterate on entry by obtaining entrySet

HashMap - Iterator on keySet, values and entrySet is fail-safe or fail-fast?

HashMap - synchronizing map using Collections.synchronizedMap

HashMap - making map unmodifiable using Collections.unmodifiableMap

ConcurrentSkipListMap Programs >

ConcurrentSkipListMap - Iterate on keys by obtaining keySet, Iterate on values by obtaining values, Iterate on entry by obtaining entrySet

ConcurrentSkipListMap - Iterator on keySet, values and entrySet is fail-safe or fail-fast?

Different approaches/Programs to Sort Map by key >

Program to Sort Map by key in Ascending order by implementing Comparator interface and overriding its compare method

Program to Sort Map by key in Descending order by implementing Comparator interface and overriding its compare method

Different approaches/Programs to Sort Map by value >

Program to Sort Map by value in Ascending order by implementing Comparator interface and overriding its compare method

Collection hierarchy >

Map hierarchy in java - Detailed - HashMap, Hashtable, ConcurrentHashMap, LinkedHashMap, TreeMap, ConcurrentSkipListMap, IdentityHashMap, WeakHashMap, EnumMap classes

ConcurrentModificationException , Fail-fast and Fail-safe >

ConcurrentModificationException, Fail-fast and Fail-safe in detail in java

Important Similarity and Differences >

Map Differences >

HashMap and Hashtable - Similarity and Differences

HashMap and ConcurrentHashMap - Similarity and Differences

HashMap vs Hashtable vs LinkedHashMap vs TreeMap - Differences

HashMap vs IdentityHashMap - Similarity and Differences with program

Labels: Collection Framework Core Java

Must read for you :

4 Comments www.javamadesoeasy.com 1 Login

Recommend 1 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Dhirendra Kumar • a year ago

Awesome Tutorial

• Reply • Share ›

Saurabh Banerjee • 2 years ago

Dear Ankit , in the following code
synchronized (map){
if (!map.containsKey(key))
return map.put(key, value);
else
return map.get(key);
}

you are using mutable object, map as lock , Suppose after execution of the following statement << "return map.put(key, value); >> the current thread goes to sleep, another thread can also enter in the section synchronized (map) , ideally it should've .
This is because the lock is on mutable object, i.e if the content of the object changes in that case T1 and T2 are locking on two different objects , that's the reason we use synchronized with immutable object .

Reply • Share ›

Anonymous • 2 years ago

Can you explain how concurrent hashnap restrict multiple threads rehashing it, when it needs to grow in size?

Reply • Share ›

Sergii Poddyachiy • 2 years ago

Thank you so much! This is the only one clear explanation over whole internet.
But i have few questions:
1) Did I understood right that concurrency level is importat for simultaneously updating threads count. And doesn't matter for simultaneously reading threads.
For example cache: If there is only one refreshing thread that modifies ConcurrencyHashMap and 10000 users that ONLY READ data from it it in the same time. For this case concurrency level 1 is enought?
2)Could you please explain rehashing in ConcurrencyHashMap? What if one segment will be overloaded and others no? Will rehashing be done for single segment or for whole ConcurrencyHashMap ?

Thanks a lot one more time in advance.

Reply • Share ›

ALSO ON **WWW.JAVAMADESOEASY.COM**

JavaMadeSoEasy.com (JMSE): What is Covariant return type in java

1 comment • 2 years ago

Srinibash Mohanty — Before introducing covarient return type, how it was maintained ?Is there any other way to do same as this principle.?

JavaMadeSoEasy.com: How to convert String to int or Integer in java

1 comment • 2 years ago

Camellia Canan — All are saying the same thing repeatedly, but in your blog I had a chance to get some useful and unique information, I love ...

JavaMadeSoEasy: Difference between CLOB and CLOB data type in Oracle

1 comment • 3 years ago

Earl Bosch — Think the heading that reads "Difference between CLOB and CLOB" should be "Difference between CLOB and BLOB"

JavaMadeSoEasy.com: what is the default initial capacity of ARRAYLIST, how it is ...

2 comments • 3 years ago

Ankit Mittal — Hi @sudhir,Thanks for comment,When, new ArrayList<integer>() is executed, Size of ArrayList is 0.Internally, ...

Subscribe **Add Disqus to your siteAdd DisqusAdd** **Disqus' Privacy PolicyPrivacy PolicyPrivacy**

Newer Post Home Older Post