

# Nachdenkzettel Collections

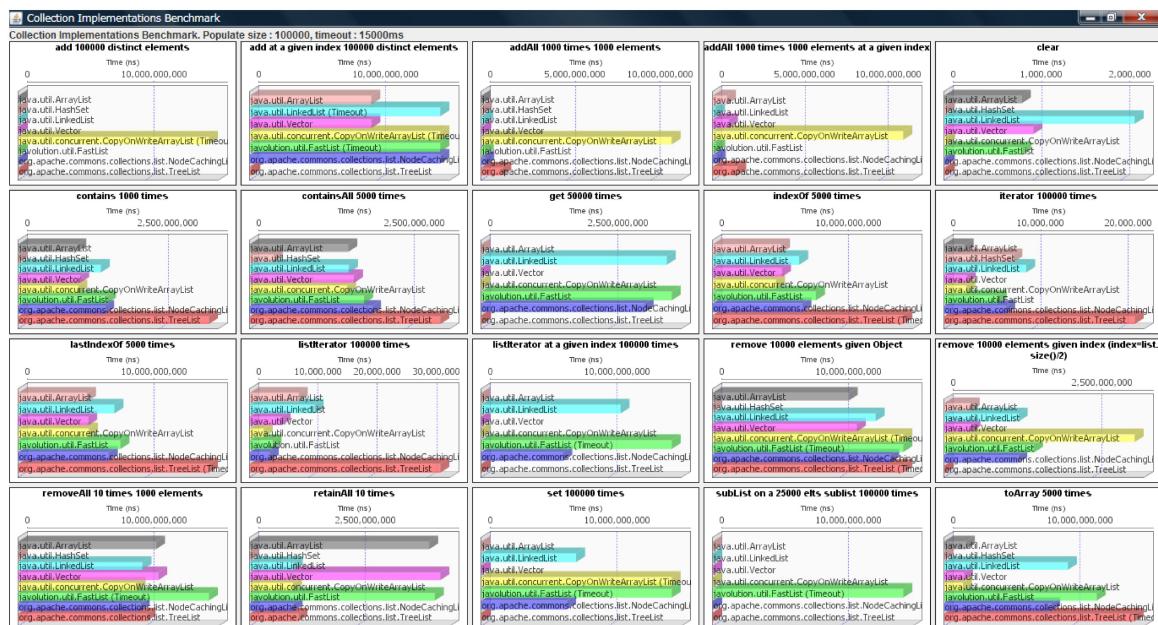
Mittwoch, 10. November 2021 14:12

## Nachdenkzettel: Collections

### 1. ArrayList oder LinkedList – wann nehmen Sie was?

Viel einfügen  
Viel get(index)

### 2. Interpretieren Sie die Benchmarkdaten von: <http://java.dzone.com/articles/java-collection-performance>. Fällt etwas auf?



LinkedList ist schlecht bei Indexanforderungen

ArrayList ist meistens ziemlich schnell

### 3. Wieso ist CopyOnWriteArrayList scheinbar so langsam?

Weil es immer wieder Elemente des Arrays kopiert.

4. Wie erzeugen Sie eine thread-safe Collection (die sicher bei Nebenläufigkeit ist) (WAS?? die ArrayLists, Linkedlists, Maps etc. sind NICHT sicher bei multithreading??? Wer macht denn so einen Mist???)

ParallelList

5. Achtung Falle!

```
List<Integer> list = new ArrayList<Integer>;  
Iterator<Integer> itr = list.iterator();  
while(itr.hasNext()) {  
    int i = itr.next();  
    if (i > 5) { // filter all ints bigger than 5  
        list.remove();  
    }  
}
```

Falls es nicht klickt: einfach ausprobieren...

Macht das Verhalten von Java hier Sinn?

Gibt es etwas ähnliches bei Datenbanken? (Stichwort: Cursor. Ist der ähnlich zu Iterator?)

Nachdem ein Element entfernt wurde, geht der Pointer eins weiter, aber an der Stelle ist am Ende dann gar kein Element mehr.

Cursor ist ähnlich wie Iterator.



6. Nochmal Achtung Falle: What is the difference between get() and remove() with respect to Garbage Collection?

get() kopiert eine Referenz und macht sonst nicht

remove() löscht eine Referenz, woraufhin der Garbage Collector das Objekt, auf welches die Referenz zeigte beseitigen muss.

7. Ihr neuer Laptop hat jetzt 8 cores! Ihr Code für die Verarbeitung der Elemente einer Collection sieht so aus:

```
Iterator<Integer> itr = list.iterator();  
while(itr.hasNext()) {  
    int i = itr.next();  
    //do something with i...  
}
```

} single threaded code

War der Laptop eine gute Investition?

Für die Mutigen: mal nach map/reduce googeln!