There are [two ways for Java to traverse collections](https://www.java67.com/2012/07/how-to-iterate-loop-traverse-list-java.html). One is the most basic for-loop, and the other is the for-each introduced by jdk5. With this method, we can traverse arrays and collections more conveniently. But have you ever thought about these two methods? Which one is more efficient to traverse the collection?

**for-each Implementation Principle**

[Foreach](https://javarevisited.blogspot.com/2015/08/java-8-journey-of-for-loop-in-java.html)is not a new syntax, but a syntax sugar of Java. At compile time, the compiler will convert this code into an [iterator](https://www.java67.com/2013/02/java-iterator-example.html) implementation and compile it into bytecode. We can decompile the following compiled code by executing the commandjavap -verbose Testforeach

|  |
| --- |
| public class TestForeach { |
|  | List<Integer> integers; |
|  | public void testForeach(){ |
|  | for(Integer i : integers){ |
|  |  |
|  | } |
|  | } |
|  | } |

The detailed [bytecode](https://javarevisited.blogspot.com/2012/05/10-points-about-class-file-in-java.html) obtained is as follows:

|  |
| --- |
| public void testForeach(); |
|  | descriptor: ()V |
|  | flags: ACC\_PUBLIC |
|  | Code: |
|  | stack=1, locals=3, args\_size=1 |
|  | 0: aload\_0 |
|  | 1: getfield #2 // Field integers:Ljava/util/List; |
|  | 4: invokeinterface #3, 1 // InterfaceMethod java/util/List.iterator:()Ljava/util/Iterator; |
|  | 9: astore\_1 |
|  | 10: aload\_1 |
|  | 11: invokeinterface #4, 1 // InterfaceMethod java/util/Iterator.hasNext:()Z |
|  | 16: ifeq 32 |
|  | 19: aload\_1 |
|  | 20: invokeinterface #5, 1 // InterfaceMethod java/util/Iterator.next:()Ljava/lang/Object; |
|  | 25: checkcast #6 // class java/lang/Integer |
|  | 28: astore\_2 |
|  | 29: goto 10 |
|  | 32: return |
|  | LineNumberTable: |
|  | line 11: 0 |
|  | line 13: 29 |
|  | line 14: 32 |
|  | LocalVariableTable: |
|  | Start Length Slot Name Signature |
|  | 29 0 2 i Ljava/lang/Integer; |
|  | 0 33 0 this Ltest/TestForeach; |
|  | } |

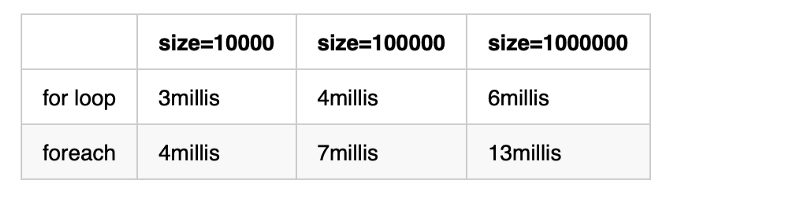
The general meaning of this bytecode is to use the getfield command to obtain the variable integers and invoke List.iterator to get the[iterator instance](https://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) and invoke iterator.hasNext. If it returns true, call iterator.next method. See, this is the implementation logic for iterators to traverse collections.

**Benchmark Test**

Now let’s use the for-loop method and the [for-each method](https://javarevisited.blogspot.com/2015/09/java-8-foreach-loop-example.html) to test.

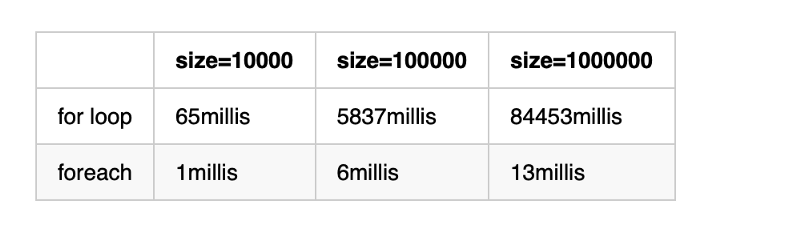
|  |
| --- |
| public class ForLoopTest { |
|  |  |
|  | public static void main(String[] args) { |
|  | List<Integer> arrayList = new ArrayList<>(); |
|  | for (int i = 0; i < 10000000; i++) { |
|  | arrayList.add(i); |
|  | } |
|  |  |
|  | long arrayListStartTime = System.currentTimeMillis(); |
|  | for (int i = 0; i < arrayList.size(); i++) { |
|  | arrayList.get(i); |
|  | } |
|  |  |
|  | long arrayListCost =System.currentTimeMillis()-arrayListStartTime; |
|  | System.out.println("ArrayList for loop traversal cost: "+ arrayListCost); |
|  |  |
|  | long arrayListForeachStartTime = System.currentTimeMillis(); |
|  | for (Integer integer : arrayList) { |
|  |  |
|  | } |
|  |  |
|  | long arrayListForeachCost =System.currentTimeMillis()-arrayListForeachStartTime; |
|  | System.out.println("ArrayList foreach traversal cost: "+ arrayListForeachCost); |

Here are the test results.

[[](https://javarevisited.blogspot.com/2011/12/how-to-traverse-or-loop-hashmap-in-java.html)](https://javarevisited.blogspot.com/2011/12/how-to-traverse-or-loop-hashmap-in-java.html)

As you can see, the result is obvious. For [ArrayList](https://www.java67.com/2015/06/20-java-arraylist-interview-questions.html), the performance of using the for-loop method is better than that of the for-each method. Can we declare the for-loop win? The answer is No. in the next benchmark, we change ArrayList to LinkedList.

Again, here are the test results.

[[](https://www.java67.com/2017/08/top-10-java-hashmap-interview-questions.html)](https://www.java67.com/2017/08/top-10-java-hashmap-interview-questions.html)

**Cause Analysis**

Some beginners may wonder why[ArrayList](https://www.java67.com/2022/03/10-examples-of-arraylist-in-java.html) traverses faster using the for loop method and [LinkedList](https://www.java67.com/2016/02/how-to-sort-linkedlist-in-java-example.html)is slower and very slow? This is determined by the [ArrayList and LinkedList data structures](https://javarevisited.blogspot.com/2012/02/difference-between-linkedlist-vs.html).

The ArrayList underlying layer uses arrays to store elements. Arrays are contiguous memory spaces. Data can be obtained through indexes. The time complexity is O(1), so it is fast.

The underlying layer of LinkedList is a two-way linked list. Use for-loop to implement traversal, you need to start from the [head node of the linked list](https://medium.com/javarevisited/top-20-linked-list-coding-problems-from-technical-interviews-90b64d2df093) every time. The time complexity is O (n\*n).

**Conclusion**

1. The for loop method is faster when using [ArrayList](https://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html) because the for-each is implemented by the iterator, and it needs to perform concurrent modification verification.
2. When using [LinkedList](https://javarevisited.blogspot.com/2012/12/how-to-initialize-list-with-array-in-java.html), the for-each is much faster than the for-loop, because LinkedList is implemented by using a two-way linked list. Each addressing needs to start from the header node. If we need to traverse LinkedList, we need to avoid using the for-loop.
3. Using the iterator pattern, for-each does not need to care about the specific implementation of the collection. If there is a need to replace the collection, it can be easily replaced without modifying the code.