Thread-Safe Data Structures

—

NTQ
Julian Bui

Collection nào có thể giúp đỡ việc viết những chương trình multi-thread?

Background

Vấn đề bị mất sự cập nhật (Lost update problem)

Giả sử muốn 2 threads đọc và thay đổi một biến động thời

Thread 1	Thread 2		Integer value
			0
read value		←	0
increase value			0
write back		→	1
	read value	←	1
	increase value		1
	write back	→	2

Good

Thread 1	Thread 2		Integer value
			0
read value		←	0
	read value	←	0
increase value			0
	increase value		0
write back		→	1
	write back	→	1

Bad

Ví dụ cụ thể

```
stackPop() {
    If (size != 0) {
        Throw exception
    } else {
        //***
        Remove and return element
    }
}
```

***Nếu stack này chỉ có một phần tử rồi 2 thread vào cái khối-else thì có vấn đề xảy ra là remove 2 lần mặc dù nó chỉ có 1 phần tử.

“trạng thải không ổn định” (inconsistent state)

Java 1.0 thread-safe collections

Thread-safe Collections:

1. Hashtable
2. Vector
3. Stack

Những cái vấn đề chính:

- Ít người dùng các collection trên.
- Chậm, phải lock các hàm trong class.

Synchronized Collections (Java 1.2)

Vocabulary

1. Synchronized - Đồng bộ - Java cung cấp một cách để đồng bộ hóa tác vụ của chúng bằng cách sử dụng các khối synchronized. Trong java cái từ này có nghĩa là hàm synchronized sẽ không được phép hoạt động khi hàm/khối synchronized nào khác đang chạy.

Collections wrappers

`static <T> Collection<T>`	**`synchronizedCollection(Collection<T> c)`** Returns a synchronized (thread-safe) collection backed by the specified collection.
`static <T> List<T>`	**`synchronizedList(List<T> list)`** Returns a synchronized (thread-safe) list backed by the specified list.
`static <K,V> Map<K,V>`	**`synchronizedMap(Map<K,V> m)`** Returns a synchronized (thread-safe) map backed by the specified map.
`static <K,V> NavigableMap<K,V>`	**`synchronizedNavigableMap(NavigableMap<K,V> m)`** Returns a synchronized (thread-safe) navigable map backed by the specified navigable map.
`static <T> NavigableSet<T>`	**`synchronizedNavigableSet(NavigableSet<T> s)`** Returns a synchronized (thread-safe) navigable set backed by the specified navigable set.
`static <T> Set<T>`	**`synchronizedSet(Set<T> s)`** Returns a synchronized (thread-safe) set backed by the specified set.
`static <K,V> SortedMap<K,V>`	**`synchronizedSortedMap(SortedMap<K,V> m)`** Returns a synchronized (thread-safe) sorted map backed by the specified sorted map.
`static <T> SortedSet<T>`	**`synchronizedSortedSet(SortedSet<T> s)`** Returns a synchronized (thread-safe) sorted set backed by the specified sorted set.

Synchronized Collection là gì?

Thread-safe wrapper để xử lý việc synchronization bằng cách dùng decorator pattern để lock mọi hàm.

Vocabulary

1. Synchronized - Đồng bộ - Java cung cấp một cách để đồng bộ hóa tác vụ của chúng bằng cách sử dụng các khối synchronized. Trong java cái từ này có nghĩa là hàm synchronized sẽ không được phép hoạt động khi hàm/khối synchronized nào khác đang chạy.

2. Thread-safe - Khi nhiều threads chạy động thời để xử lý một việc, cái kết quả phải đúng.

Code

```java
List<Integer> unsynchronizedList = new ArrayList<Integer>();
List<Integer> synchronizedList = Collections.synchronizedList(unsynchronizedList);

synchronizedList.add(25);
synchronizedList.get(0);

synchronized(synchronizedList) {
    for (Integer i : synchronizedList) {
        System.out.println(i);
    }
}

synchronized(synchronizedList) {
    Iterator<Integer> iter = synchronizedList.iterator();
    while(iter.hasNext()) {
        Integer i = iter.next();
        System.out.println(i);
    }
}
```

Tính chất của Synchronized Wrappers

+) Thread-safe

-) Chậm: 2 hàm của 1 lớp không chạy cùng một lúc được vì cấu trúc dữ liệu phải được lock.

-) Không có cách để dùng các hàm của class cụ thể (e.g., ArrayList -> List), chỉ dùng được interface thôi

ConcurrentModificationException

“The iterators returned by this class's iterator method are fail-fast: if the set is modified at any time after the iterator is created, in any way except through the iterator's own remove method, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.”

Fail-fast iterators: Khi đang duyệt collection mà nó bị cập nhật, iterator phải bắn exception liền.

Nói chung là khi duyệt mọi cấu trúc dữ liệu với iterator trong java mà add/remove phần tử thì sẽ gặp ConcurrentModificationException. Chỉ khi remove bằng iterator thì sẽ không bị exception.

Note: Có thể gặp vấn đề này khi không dùng threads.

Concurrent* Collections (Java 1.5)

Tính chất của các cấu trúc Concurrent*

1. Thread-safe
2. Scalable vì nó được thực thi bằng cách dùng nhiều lock (synchronized collections chỉ có một lock thôi) nên có thể một function sẽ không block các function khác.
3. Không có cách để iterate snapshot - “Weakly consistent” (fail-safe) iteration.
4. Concurrent* collections sẽ không nhận NULL keys và NULL values

ConcurrentHashMap

ConcurrentHashMap

Gióng như HashMap,

mà các hàm của nó thread-safe

và có vài hàm đã thành Atomic

Va co nhung ham co the chay dong thoi

Vocabulary

1. Synchronized - Đồng bộ - Java cung cấp một cách để đồng bộ hóa tác vụ của chúng bằng cách sử dụng các khối synchronized. Trong java cái từ này có nghĩa là hàm synchronized sẽ không được phép hoạt động khi hàm/khối synchronized nào khác đang chạy.

2. Thread-safe - Khi nhiều threads chạy động thời để xử lý một việc, cái kết quả phải đúng.

3. Atomic - Thành phần nguyên tử - Đối tượng nhỏ nhất. Không tách nó ra thành từng thành phần nào nhỏ hơn được.

Atomic Functions

- `computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)`
- `computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)`
- `putIfAbsent(K key, V value)`
- `replace(K key, V value)`
- `replace(K key, V oldValue, V newValue)`

Atomic Functions

Tại sao cần nó? Race conditions.

public boolean replace(K key, V oldValue, V newValue)

```
if (map.containsKey(key) &&
Objects.equals(map.get(key), oldValue)) {
   map.put(key, newValue);
   return true;
 } else
   return false;
```

Parallel Functions

```
public void forEach(long parallelismThreshold,
                    BiConsumer<? super K,? super V> action)

public <U> U search(long parallelismThreshold,
                    BiFunction<? super K,? super V,? extends U> searchFunction)

public <U> U reduce(long parallelismThreshold,
                    BiFunction<? super K,? super V,? extends U> transformer,
                    BiFunction<? super U,? super U,? extends U> reducer)
```

Special considerations

1. Không có cách để duyệt snapshot. Nếu phần tử nào được add khi đang duyệt, thì có thể thấy các phần tử mới.

2. ConcurrentHashMap không cung cấp cách để dùng NULL Key hoặc là NULL Value

Copy-on-Write (COW) Collections

Copy-on-Write Collection là gì?

Các COW collection được thực thi băng cách copy cả collection khi dùng một hàm gọi là “Mutative” (add, set, remove, vv...). Cái copy được coi như là một collection immutable, nên nó thread-safe.

Draw

Khi đọc, không cần phải lock

Khi thay đổi dữ liệu, không cần phải block ai đang đọc

Snapshot iterator - có nghĩa là gì?

> Nếu chụp hình cái COW collection thì cái hình sẽ không thay đổi. Coi như là nó immutable nên khi duyệt nó, không ai có thể add/remove. Nên không bao giờ gặp ConcurrentModificationException.

Khi nào dùng Copy-on-Write Collection?

1. Đọc nhiều mà ít khi thay đổi cái collection
2. Không có quá nhiều dữ liệu trong collection
3. Cần snapshot iterator

Chú Ý

1. Nếu có nhiều dữ liệu trong cái COW collection thì, khi ghi vào nó, sẽ mất nhiều thời gian vì nó phải copy tất

Unmodifiable and Immutable Collections

Vocabulary

1. Synchronized - Đồng bộ - Java cung cấp một cách để đồng bộ hóa tác vụ của chúng bằng cách sử dụng các khối synchronized. Trong java cái từ này có nghĩa là hàm synchronized sẽ không được phép hoạt động khi hàm/khối synchronized nào khác đang chạy.

2. Thread-safe - Khi nhiều threads chạy động thời để xử lý một việc, cái kết quả phải đúng.

3. Atomic - Thành phần nguyên tử - Đối tượng nhỏ nhất. Không tách nó ra thành từng thành phần nào nhỏ hơn được.

4. Mutability - Sự thay đổi - Khi thay đổi dữ liệu, chương trình sẽ phức tạp hơn và phải biết synchronize nó.

Unmodifiable Collections

Collections wrappers

`static <T> Collection<T>`	**`unmodifiableCollection(Collection<? extends T> c)`** Returns an unmodifiable view of the specified collection.
`static <T> List<T>`	**`unmodifiableList(List<? extends T> list)`** Returns an unmodifiable view of the specified list.
`static <K,V> Map<K,V>`	**`unmodifiableMap(Map<? extends K,? extends V> m)`** Returns an unmodifiable view of the specified map.
`static <K,V> NavigableMap<K,V>`	**`unmodifiableNavigableMap(NavigableMap<K,? extends V> m)`** Returns an unmodifiable view of the specified navigable map.
`static <T> NavigableSet<T>`	**`unmodifiableNavigableSet(NavigableSet<T> s)`** Returns an unmodifiable view of the specified navigable set.
`static <T> Set<T>`	**`unmodifiableSet(Set<? extends T> s)`** Returns an unmodifiable view of the specified set.
`static <K,V> SortedMap<K,V>`	**`unmodifiableSortedMap(SortedMap<K,? extends V> m)`** Returns an unmodifiable view of the specified sorted map.
`static <T> SortedSet<T>`	**`unmodifiableSortedSet(SortedSet<T> s)`** Returns an unmodifiable view of the specified sorted set.

Unmodifiable Collections

Các interface trên sẽ bắn exception khi add/remove/modify collection

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
for (int i = 0; i < INITIAL_LIST_SIZE; i++) {
    arrayList.add(1);
}
List<Integer> unmodifiableList = Collections.unmodifiableList(arrayList);
arrayList = null; // throw away the reference
```

Chú Ý

Nếu không vứt cái reference đi, thì một ai vẫn còn thay đổi cái collection góc. Unmodifiable collection chỉ là cái Wrapper thôi.

Immutable Collections

com.google.guava Immutable Collections

ImmutableCollection, ImmutableList, ImmutableSet, ImmutableMap, ...

About all Immutable- collections

The remainder of this documentation applies to every public `Immutable-` type in this package, whether it is a subtype of `ImmutableCollection` or not.

Guarantees

Each makes the following guarantees:

- **Shallow immutability.** Elements can never be added, removed or replaced in this collection. This is a stronger guarantee than that of `Collections.unmodifiableCollection(java.util.Collection<? extends T>)`, whose contents change whenever the wrapped collection is modified.
- **Null-hostility.** This collection will never contain a null element.
- **Deterministic iteration.** The iteration order is always well-defined, depending on how the collection was created (see the appropriate factory method for details). View collections such as `ImmutableMultiset.elementSet()` iterate in the same order as the parent, except as noted.
- **Thread safety.** It is safe to access this collection concurrently from multiple threads.
- **Integrity.** This type cannot be subclassed outside this package (which would allow these guarantees to be violated).

When to use Guava Immutables over Unmodifiables?

The JDK provides `Collections.unmodifiableXXX` methods, but in our opinion, these can be

- unwieldy and verbose; unpleasant to use everywhere you want to make defensive copies
- unsafe: the returned collections are only truly immutable if nobody holds a reference to the original collection
- inefficient: the data structures still have all the overhead of mutable collections, including concurrent modification checks, extra space in hash tables, etc.
- `Collections.unmodifiableXXX` methods returns Set, List, etc., not “UnmodifiableSet”, “UnmodifiableList”, etc., so it’s not clear what’s the variables are.

Conclusion

Conclusion

Phải dùng các collection thread-safe khi có nhiều thread

Không nên dùng các wrapper của Collections.unmodifiable*

Không nên dùng các wrapper của Collections.synchronized*

Nên dùng Immutable collections của Guava nếu dữ liệu sẽ không thay đổi

Nên dùng các CopyOnWrite collection nếu dữ liệu sẽ không thay đổi thường xuyên

ConcurrentHashMap có nhiều lợi ích

Chưa kịp trình bay về:

- Các Concurrent* khác (BlockingQueue, BlockingDeque, ConcurrentSkip*, ConcurrentNavigable*, văn văn)
- Atomic collections (java.util.concurrent.atomic.*)