Building Blocks

Yang Xu

Department of Automatic Control

Building blocks

- Synchronized collections
- Concurrent collections
- Blocking queues and the producer-consumer pattern
- Blocking and interruptible methods
- Synchronizers
- Building an efficient, scalable result cache

Problems with synchronized collections

- Common compound actions on collections: iteration, navigation, conditional operations
- Problem: They may not behave as expect.
- Solution: client-side locking

```
synchronized (list) {
        doSomething;
}
```

Iterators and ConcurrentModificationException

- To iterate a Collection by: explicitly Iterater, for-each loop syntax
- Problem: Fail-fast → ConcurrentModificationException
- Solution:

1. Locking: starvation, deadlock, hurting scalability
2. Clone the collection and iterate the copy instead

Hidden iterators

- Iteraion is indirectly invoked by:

1. string concatenation
2. hashCode
3. equals
4. the containsAll, removeAll, retainAll
5. the constructors that take the collections as arguments

ConcurrentHashMap

	Hashtable	synchronizedMap	ConcurrentHashMap
throughput	low	low	high
lock	yes	yes	no
size(), isEmpty()	exact count	exact count	approximation
lock the map	yes	yes	no
scalability	good	good	better

Additional atomic Map operations

- Atomic operations specified by the ConcurrentMap interface

```
public interfance ConcurrentMap<K,V> extends <K,V> {

        V putIfAbsent(K key, V value);

        boolean remove(K key, V value);

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

        V replace(K key, V newValue);

}
```

CopyOnWriteArrayList

- Better concurrency without the need to lock or copy the collection
- When an immutable object is properly published, no further synchronization is required.
- Copy-on-write collections (when iteration is far more common than modification)

Producer-consumer pattern example: desktop search

- An agent: scans local drives for documents → indexes them for later searching

- Code is more readable and reusable

- Better throughput

Serial thread confinement

- The blocking queue implementations contain internal synchronizaiton.
- Serial thread confinement: safe, visible
- Other methods: the atomic remove of ConcurrentMap, the compareAndSet of AtomicReference

Deques and work stealing

- Deque implementations: ArrayDeque and LinkedBlockingDeque

- Deques lend themselves to work stealing (more scalable)

- Is well suited to problems in which consumers are also producers

Blocking and interruptible methods

- Blocking methods: to wait for an event that is beyond its control before it can proceed
- Interrupt methods: to make an effort to stop blocking early
- Interruption: boolean property, cooperative mechanism
- Responses to interruption: propagate the InterruptedException, restore the interrupt

Latches

- Latches: to ensure that certain activities do not proceed until other one-time activities complete.
- Implementation: CountDownLatch
- Common uses for latches:

1. Staring gate: to release all the worker threads at once
2. Ending gate: to wait for the last thread to finish

FutureTask

- Three states: waiting to run, running, completed
- Once FutureTask enters the completed state, it stays in that state forever.
- Future.get depends on the state of the task
- Represents lengthy computation
- Reasons for ExecutionException: checked exception thrown by the Callable, RuntimeException, Error

Semaphores

- Counting semaphores: to control the number of activities that can access a certain resource or perform a given action at the same time.
- Implementation:

1. acquire() a permit to fetch a resource from a pool
2. release() the permit after putting a resource back in the pool

Barriers

- Barriers: block a group of threads until some event has occured. All the threads must come together at a barrier point at the same time.

- Implementation:

1. CyclicBarrier
2. Exchanger

Building an efficient, scalable resulet cache

- Memoizer1: HashMap, long computation time

- Memoizer2: ConcurrentHashMap, better concurrent behavior, but safty risk

- Memoizer3: FutureTask, perfect

Exercises

1. What are the characteristics of BlockingQueue, compared with other general queue?
2. What is the role of semaphore synchronizer?
3. Programming (using any method in this chapter)

The program contains three threads. First, these threads print out a message, and then sleep a random time. Finally print out the thread end message and exit.