Andersen

Многопоточность

Кожевников Константин

Andersen

План

1. Процесс и поток
2. Thread, Runnable, Callable, Object
3. Модель памяти в Java, Happens Before
4. Проблемы многопоточности
5. Synchronized, volatile
6. Concurrent
    a. Коллекции
    b. Атомики
    c. Локи
    d. Синхронизаторы

Andersen

Процесс и поток

1. Процесс - это совокупность кода и данных, разделяющих общее виртуальное адресное пространство.

1. Поток - это одна единица исполнения кода. Каждый поток последовательно выполняет инструкции процесса, которому он принадлежит, параллельно с другими потоками этого процесса.

https://habr.com/ru/post/164487/

Andersen

Thread, runnable, callable, object

Module java.base
Package java.lang

Interface Runnable

`void` `run()`

Module java.base
Package java.lang

Class Object

java.lang.Object

Module java.base
Package java.lang

Class Thread

java.lang.Object
    java.lang.Thread

All Implemented Interfaces:

Runnable

Module java.base
Package java.util.concurrent

Interface Callable<V>

Type Parameters:

V - the result type of method `call`

compact1, compact2, compact3
java.util.concurrent

Interface Future<V>

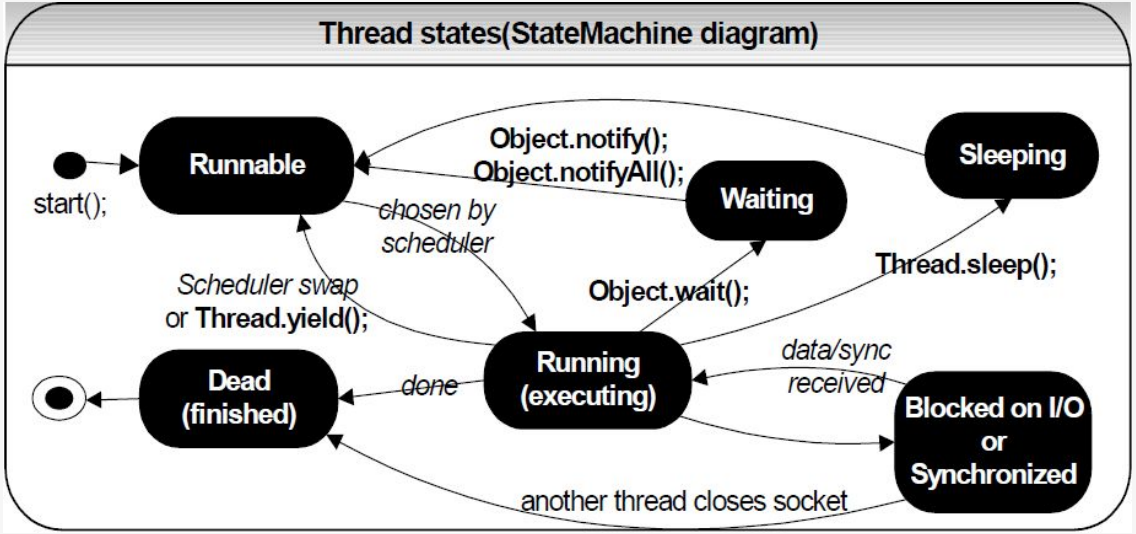
Type Parameters:

`V - The result type returned by this Future's get method`

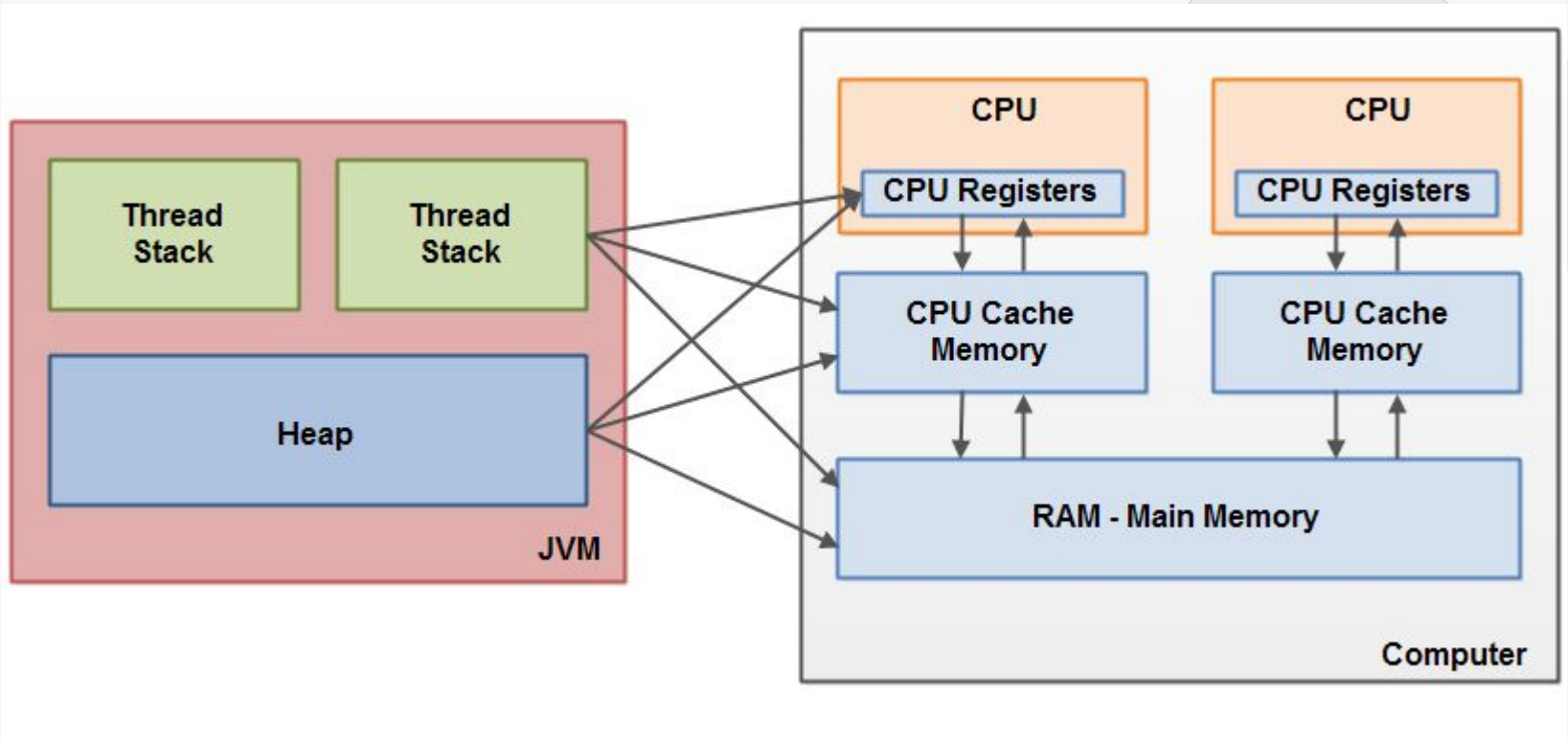
https://habr.com/ru/post/164487/

Andersen

Thread, runnable, callable, object



Andersen


Thread Stack
Thread Stack
Heap
JVM
CPU
CPU Registers
CPU
CPU Registers
CPU Cache Memory
CPU Cache Memory
RAM - Main Memory
Computer

Andersen

Модель памяти Java

Модель памяти Java (англ. Java Memory Model, JMM) описывает поведение потоков в среде исполнения Java. Модель памяти — часть семантики языка Java, и описывает, на что может и на что не должен рассчитывать программист, разрабатывающий ПО не для конкретной Java-машины, а для Java в целом.

Свойства:

1. Атомарность
2. Видимость
3. Изменение порядка

https://habr.com/ru/post/133981/

Andersen

Happens Before

1. В рамках одного поток любая операция happens-before любой операцией следующей за ней в исходном коде
2. Освобождение лока (unlock) happens-before захват того же лока (lock)
3. Выход из synhronized блока/метода happens-before вход в synhronized блок/метод на том же мониторе
4. Запись volatile поля happens-before чтение того же самого volatile поля
5. Запись значения в final-поле (и, если это поле — ссылка, то ещё и всех переменных, достижимых из этого поля (dereference-chain)) при конструировании объекта happens-before запись этого объекта в какую-либо переменную, происходящая вне этого конструктора.
6. Завершение метода run экземпляра класса Thread happens-before выход из метода join() или возвращение false методом isAlive() экземпляром того же треда
7. Вызов метода start() экземпляра класса Thread happens-before начало метода run() экземпляра того же треда
8. Завершение конструктора happens-before начало метода finalize() этого класса
9. Вызов метода interrupt() на потоке happens-before когда поток обнаружил, что данный метод был вызван либо путем выбрасывания исключения InterruptedException, либо с помощью методов isInterrupted() или interrupted()

http://www.javaspecialist.ru/2011/06/java-memory-model.html

Andersen

Проблемы многопоточности

- Состояние гонки
- Deadlock
- Livelock
- Starvation

https://habr.com/ru/company/otus/blog/549814/

Andersen

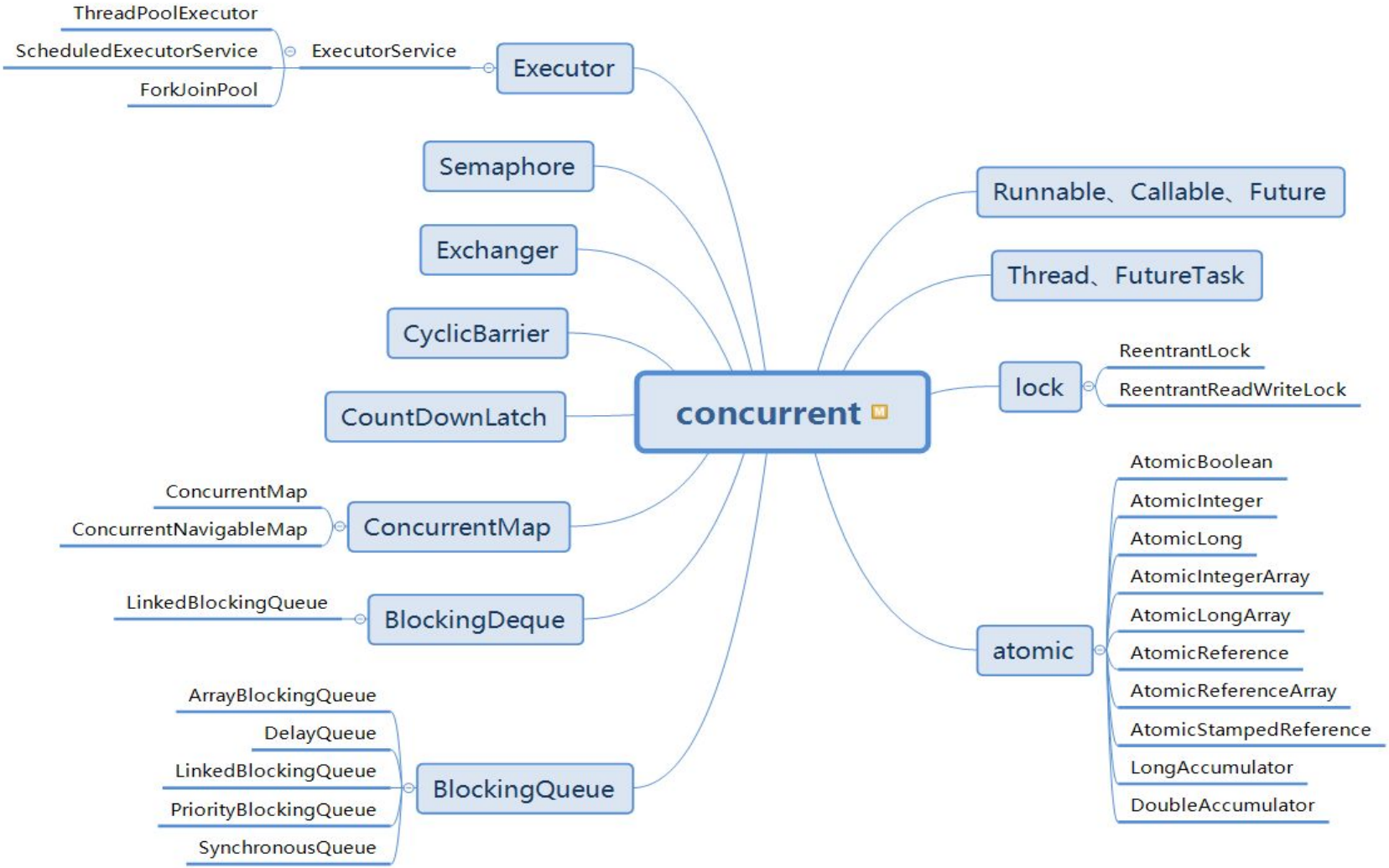
Synchronized, volatile

Монитор (мьютекс) - абстракция, используемая для синхронизации потоков.

Synchronized - ключевое слово обозначающее что для исполнения данного участка кода требуется монитор объекта.

Volatile - ключевое слово, которое добавляет атомарность при операциях чтения/записи, а так же дает гарантию что данные будут сохраняться и читаться в/из памяти, а не из кеша

https://habr.com/ru/post/108016/


concurrent
Executor
ExecutorService
ThreadPoolExecutor
ScheduledExecutorService
ForkJoinPool
Semaphore
Exchanger
CyclicBarrier
CountDownLatch
ConcurrentMap
ConcurrentMap
ConcurrentNavigableMap
BlockingDeque
LinkedBlockingQueue
BlockingQueue
ArrayBlockingQueue
DelayQueue
LinkedBlockingQueue
PriorityBlockingQueue
SynchronousQueue
Runnable、Callable、Future
Thread、FutureTask
lock
ReentrantLock
ReentrantReadWriteLock
atomic
AtomicBoolean
AtomicInteger
AtomicLong
AtomicIntegerArray
AtomicLongArray
AtomicReference
AtomicReferenceArray
AtomicStampedReference
LongAccumulator
DoubleAccumulator

Andersen

Пакет Concurrent

- ConcurrentHashMap коллекция типа HashMap, реализующая интерфейс ConcurrentMap;
- CopyOnWriteArrayList коллекция типа ArrayList с алгоритмом CopyOnWrite;
- CopyOnWriteArraySet реализация интерфейса Set, использующая за основу CopyOnWriteArrayList;
- ConcurrentNavigableMap расширяет интерфейс NavigableMap;
- ConcurrentSkipListMap аналог коллекции TreeMap с сортировкой данных по ключу и с поддержкой многопоточности;
- ConcurrentSkipListSet реализация интерфейса Set, выполненная на основе класса ConcurrentSkipListMap.

https://java-online.ru/concurrent-collections.xhtml

Andersen

Пакет Concurrent

- Неблокирующие очереди
  - ConcurrentLinkedQueue
  - ConcurrentLinkedDeque
- Блокирующие очереди
  - ArrayBlockingQueue
  - LinkedBlockingQueue
  - LinkedBlockingDeque
  - SynchronousQueue
  - LinkedTransferQueue
  - DelayQueue
  - PriorityBlockingQueue

https://java-online.ru/concurrent-queue-noblock.xhtml
https://java-online.ru/concurrent-queue-block.xhtml

Andersen

Пакет Concurrent

- AtomicBoolean
- AtomicInteger
- AtomicLong
- AtomicReference

https://java-online.ru/concurrent-atomic.xhtml

Andersen

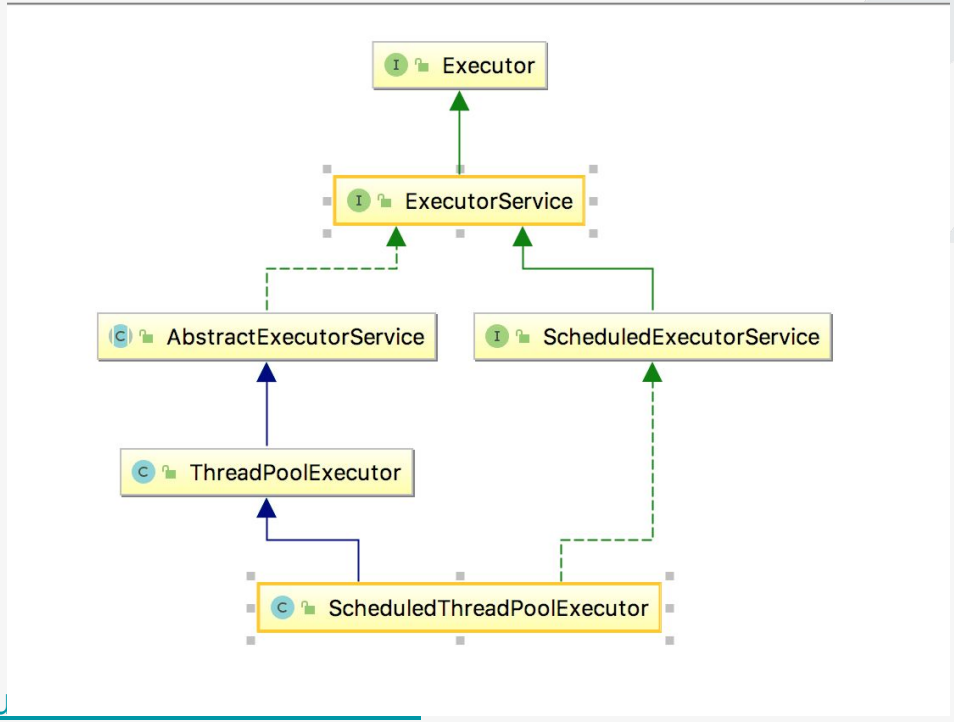
Пакет Concurrent

- ReentrantLock
- ReentrantReadWriteLock
- StampedLock

https://java-online.ru/concurrent-atomic.xhtml

Andersen

Пакет Concurrent



https://java-online.ru

Andersen

Пакет Concurrent

- **Semaphore** объект синхронизации, ограничивающий количество потоков, которые могут «войти» в заданный участок кода;
- **CountDownLatch** объект синхронизации, разрешающий вход в заданный участок кода при выполнении определенных условий;
- **CyclicBarrier** объект синхронизации типа «барьер», блокирующий выполнение определенного кода для заданного количества потоков;
- **Exchanger** объект синхронизации, позволяющий провести обмен данными между двумя потоками;
- **Phaser** объект синхронизации типа «барьер», но в отличие от CyclicBarrier, предоставляет больше гибкости.

https://java-online.ru/concurrent-atomic.xhtml

Бонус

https://habr.com/ru/company/luxoft/blog/157273/ - обзор на пакет Concurrent
https://youtu.be/t0dGLFtRR9c - ForkJoinPool
https://annimon.com/article/3462 - CompletableFuture

https://habr.com/ru/post/138533/ *