УДК 004.43

А.М. Луцків канд.техн.наук; доц., В.В. Худоба
Тернопільський національний технічний університет імені Івана Пулюя, Україна

КЛЮЧОВІ ОСОБЛИВОСТІ СТВОРЕННЯ БАГАТОПОТОКОВИХ ПРОГРАМ ДЛЯ ПЛАТФОРМИ JAVA

A.M. Lutskiv (Ph.D.; Assoc. Prof.), V.V. Khudoba

KEY FEATURES OF CREATING CONCURRENT PROGRAMS FOR JAVA PLATFORM

При створенні програмного забезпечення комп'ютерних систем для опрацювання великих масивів даних або великої кількості подій для платформи Java у більшості випадків використовуються засоби багатопотокового програмування (concurrent programming) та асинхронне опрацювання подій (Java NIO та NIO2). На сьогодні Java надає широкий спектр засобів для створення багатопотокового програмного забезпечення:

1. Низькорівневі засоби[1]: синхронізаційні примітиви, засоби створення та керування виконанням окремих потоків.

2. Високорівневі засоби, які входять в пакет java.util.concurrent[2,3]:

- засоби керування створенням та виконанням окремих потоків та пулів потоків (Executor Framework);
- високорівневі синхронізаційні засоби (Phaser, CyclicBarrier, CountDownLatch, Exchanger, Semaphore);
- блокуючі колекції для багатопотових програм (ArrayBlockingQueue, PriorityBlockingQueue, LinkedBlockingDeque та інші);
- неблокуючі колекції для багатопотокових програм (ConcurrentHashMap, ConcurrentLinkedDeque, CopyOnWriteArraySet та інші);
- фреймворк ForkJoin, який забезпечує інкрементальне розпаралелення виконання обчислювальної задачі.

При використанні наведених засобів необхідно дотримуватись низки важливих принципів[4]:

- намагатись створювати екземпляри об'єктів класів незмінними (immutable);
- намагатись проводити оптимізації на рівні алгоритмів, а не конструкцій мови, які якимось чином теоретично можуть впливати на роботу віртуальної машини Java;
- синхронізувати доступ до спільних змінних;
- уникати надмірної синхронізації, оскільки, це може суттєво знизити швидкодію роботи програми (наприклад, застарілі класи-колекції Vector, HashTable);
- надавати перевагу високорівневими засобами створення та виконання потоків (Executor Framework) та Stream API, а не низкорівневим (Thread, Runnable);
- надавати перевагу високорівневим синхронізаційним засобам, а не низькорівневим (wait, notify тощо) й за жодних умов не використовувати методи Java API, які вважаються застарілими: stop(), suspend() та інші;
- документувати багатопотоковість засобами JavaDoc;
- обережно й уважно використовувати відкладену («ліниву») ініціалізацію об'єктів;
- хід виконання програми не покладати на планувальник виконання потоків операційної системи.

При опрацювання великої кількості однотипних даних доцільним є використання функційної парадигми програмування, яка добре себе зарекомендувала в мові Haskell, Scala та низці інших. З версії 8 у Java з'явились відповідні можливості, а у версіях 9, 10

та 11 вони були вдосконалені. На жаль, невелика кількість розробників використовує Java Stream API коректно. Здебільшого ці можливості або ігноруються, або процедурний код бездумно перетворюється у псевдо-функційний. Водночас дотримання низки принципів дає змогу створювати ефективні програмні продукти в рамках функційного, об'єктно-орієнтованого та процедурного підходів на Java:

- віддавати перевагу використанню лямбд, а не анонімних класів;
- віддавати перевагу посиланням на метод замість лямбд (якщо це можливо);
- використовувати стандартні функційні інтерфейси (Function, Predicate, Supplier та інші);
- використовувати Stream API для тих задач, для яких воно підходить найкраще;
- надавати перевагу використанню функцій без побічних ефектів (side-effect) у Stream API, оскільки, потоки можуть виконуватися паралельно;
- надавати перевагу використанню колекції, як результату повернення з методу, а не потоку (Stream). У деяких випадках можливим є використання типу Iterable, оскільки, колекція та потік можуть бути приведені до цього типу.
- обережно й уважно використовувати паралельне виконання(виклик parallel()) в етапах потоку (stream pipeline).

Звичайно, крім зазначених рекомендацій доцільно звертатись й до загальних принципів функційного програмування. Зазначимо, що використання Stream API дає змогу здійснювати багатопотокове опрацювання даних без явної роботи з засобами багатопотоковості в коді програми.

Важливим етапом впровадження будь-яких програмних компонент комп'ютерних систем є їх тестування з метою оцінювання коректності та ефективності їх виконання. Ефективність виконання коду доцільно оцінювати спеціалізованими засобами, зокрема, в таких тестах потрібно унеможливлювати вплив оптимізацій віртуальної машини. Для цього варто скористатись фреймворком JMH (Java Microbenchmark Harness) [5], який дає змогу оцінити швидкодію Java-програми шляхом проведення цілої низки тестів, які дають змогу оцінити середній час виконання певного коду без JVM-оптимізацій.

Розробнику доцільно вивчати нові засоби Java API, які з'являються у нових версіях й читати офіційну документацію, оскільки, доволі часто це дає змогу підвищити ефективність роботи програми без суттєвих часозатрат — доволі часто певні колекції або утиліти можуть бути суттєво вдосконалені розробниками на алгоритмічному рівні та рівні структур даних. Яскравим прикладом є використання програмного генератора псевдовипадкових чисел: класичного java.util.Random та відносно нового java.util.concurrent.ThreadLocalRandom, який, як показали JMH-тести, працює приблизно в 6 разів швидше.

Урахування зазначених рекомендацій дає змогу створювати ефективні та коректні багатопотокові програми на мові Java.

Література

1.Lea D. Concurrent Programming in JavaTM: Design Principles and Patterns, Second Edition / Doug Lea // Addison Wesley, USA. 1999, 432p.

2.Goetz B. Java Concurrency in Practice 1st Edition / Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea// Addison-Wesley Professional, USA, 2006, 432p.

3.González J. F. Java 7 Concurrency Cookbook /Javier Fernández González// Packt Publishing, Birmingham - Mumbai.: 2012, 365p.

4.Bloch J. Effective Java (3rd Edition) / Joshua Bloch // Addison-Wesley Professional, 2018, USA, 416p.

5.OpenJDK. Code Tools: jmh [Електронний ресурс] Режим доступу: URL: https://openjdk.java.net/projects/code-tools/jmh/