**ThreadPoolExecutor và nguyên tắc quản lý pool size**

1. Khái niệm
2. Nguyên tắc vận hành
3. Code ví dụ

ThreadPoolExecutor là một class nâng cao hơn của các ThreadPool cơ bản trong gói java concurrent. Cụ thể các thể loại ThreadPool khác bạn xem ở đây:

1. [Khái niệm ThreadPool và Executor trong Java](https://loda.netlify.app/khai-niem-thread-pool-va-executor-trong-java-loda1554800053212)

Đặc điểm của các loại ThreadPool thông thường được cung cấp trong ExecutorService là không đủ linh động theo tình huống. điển hình là bị fix số lượng thread, hoặc cho phép tạo quá nhiều thread. Nó thực sự chưa phải phương án tối ưu.

ThreadPoolExecutor thì khác, một phiên bản nâng cấp hơn, cho phép chúng ta tùy biến số lượng Thread theo kịch bản. Giúp nó thông minh hơn mấy cái kia một chút.

Ngoài ra còn có ThreadPoolTaskExecutor do Spring Framework cung cấp cũng hoạt động tương tự

### [#](https://loda.netlify.app/thread-pool-executor-va-nguyen-tac-quan-ly-pool-size-loda1554816034602/#kh%C3%A1i-ni%E1%BB%87m)Khái niệm

ThreadPoolExecutor và ThreadPoolTaskExecutor cũng là Executor nhưng nó có thêm các tham số như sau:

* corePoolSize: Số lượng Thread mặc định trong Pool
* maxPoolSize: Số lượng tối đa Thread trong Pool
* queueCapacity: Số lượng tối da của BlockingQueue

### [#](https://loda.netlify.app/thread-pool-executor-va-nguyen-tac-quan-ly-pool-size-loda1554816034602/#nguy%C3%AAn-t%E1%BA%AFc-v%E1%BA%ADn-h%C3%A0nh)Nguyên tắc vận hành

Ví dụ với ThreadPoolExecutor có:

* corePoolSize: 5
* maxPoolSize: 15
* queueCapacity: 100

1. Khi có request, nó sẽ tạo trong Pool tối đa 5 thread (corePoolSize).
2. Khi số lượng thread vượt quá 5 thread. Nó sẽ cho vào hàng đợi.
3. Khi số lượng hàng đợi full 100 (queueCapacity). Lúc này mới bắt đầu tạo thêm Thread mới.
4. Số thread mới được tạo tối đa là 15 (maxPoolSize).
5. Khi Request vượt quá số lượng 15 thread. Request sẽ bị từ chối!

Với kịch bản như thế này, bạn sẽ luôn tiết kiệm được số lượng thread sử dụng là 5 trong trường hợp bình thường. Nhưng vẫn có thể handle lên tới 15 thread nếu server quá tải.

Điểm chúng ta hay nhầm lẫn là điều kiện để tạo thêm thread đó là khi **hàng đợi phải full**. Đúng vậy, nếu hàng đợi chưa full, thì có nghĩa chúng ta chưa quá tải.

### [#](https://loda.netlify.app/thread-pool-executor-va-nguyen-tac-quan-ly-pool-size-loda1554816034602/#code-v%C3%AD-d%E1%BB%A5)Code ví dụ

Tạo ra một Runnable để xử lý các nhiệm vụ.

public class RequestHandler implements Runnable {

String name;

public RequestHandler(String name){

this.name = name;

}

@Override

public void run() {

try {

System.out.println(Thread.currentThread().getName() + " Starting process " + name);

// Giả sử nhiệm vụ xử lý hết 0.5s

Thread.sleep(500);

System.out.println(Thread.currentThread().getName() + " Finished process " + name);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

Tạo ra ThreadPoolExecutor để xử lý 1000 request tới dồn dập.

public class ThreadPoolExecutorExample {

public static void main(String[] args) {

int corePoolSize = 5;

int maximumPoolSize = 10;

int queueCapacity = 100;

ThreadPoolExecutor executor = new ThreadPoolExecutor(corePoolSize, // Số corePoolSize

maximumPoolSize, // số maximumPoolSize

10, // thời gian một thread được sống nếu không làm gì

TimeUnit.SECONDS,

new ArrayBlockingQueue<>(queueCapacity)); // Blocking queue để cho request đợi

// 1000 request đến dồn dập, liền 1 phát, không nghỉ

for (int i = 0; i < 1000; i++) {

executor.execute(new RequestHandler("request-" + i));

}

executor.shutdown(); // Không cho threadpool nhận thêm nhiệm vụ nào nữa

while (!executor.isTerminated()) {

// Chờ xử lý hết các request còn chờ trong Queue ...

}

}

}

// OUTPUT

/\*

..

..

pool-1-thread-3 Finished process request-96

pool-1-thread-5 Finished process request-97

pool-1-thread-4 Finished process request-98

pool-1-thread-8 Finished process request-100

pool-1-thread-2 Finished process request-99

pool-1-thread-6 Finished process request-102

pool-1-thread-7 Finished process request-101

pool-1-thread-9 Finished process request-104

pool-1-thread-10 Finished process request-103

\*/

Bạn sẽ thấy là chương trình đã phải sử dụng tới 10 thread để xử lý hết 1000 request cùng 1 lúc. Nhớ là cùng 1 lúc nhé các bạn, thế là nhiều rồi đó. Và theo nguyên tắc. Nó đã tận dụng hết maxPoolSize rồi. Mà queue vẫn full. Nên các request không ở trong queue sẽ bị reject. Dẫn tới chỉ sử lý được 104 request mà thôi.

Bây giờ, vẫn là ví dụ này, nhưng mỗi request cách nhau 50 milliseconds thì sẽ như nào, dễ thở hơn k? chỉ 0.05s thôi.

for (int i = 0; i < 1000; i++) {

executor.execute(new RequestHandler("request-" + i));

Thread.sleep(50);

}

// OUTPUT:

/\*

..

..

pool-1-thread-2 Finished process request-993

pool-1-thread-1 Finished process request-994

pool-1-thread-3 Finished process request-995

pool-1-thread-4 Finished process request-996

pool-1-thread-5 Finished process request-997

pool-1-thread-9 Finished process request-998

pool-1-thread-10 Finished process request-999

\*/

Xử lý gọn gàng, sạch sẽ các bạn ạ. Sức mạnh của ThreadPoolExecutor phát huy rõ rệt hơn. Tận dụng được 10 thread và queue vẫn còn chỗ nên rất nhanh, khác biệt trong một hệ thống có thể đc tính bằng milliseconds như vậy đó. nếu mỗi request cách nhau 100 milliseconds thì nó chỉ cần sử dụng 5 thread thôi.

toàn bộ code mình để tại Github: [CODE](https://github.com/loda-kun/java-all)

Chúc các bạn học tập tốt! ohoho