



# Thread Pool

≡ 분류

Java

SpringBoot

## 문제점



단순히 Thread만 사용해서 동시에 여러 작업을 실행시킬 수 있는 프로그램을 만들 수 있을까?

- 요청이 올때마다 새로운 쓰레드를 생성해서 사용하는 프로세스
  - 문제 1 : Thread 생성 비용이 크기 때문에 요청에 대한 응답 시간이 늘어난다.



1. Java는 One-to-One Threading-Model로 Thread를 생성한다.
  2. User Thread(Process의 스레드) 생성 시 OS Thread(OS 레벨의 스레드) 와 연결해야한다.
  3. 새로운 Thread 를 생성할 때 마다 OS Kernal의 작업이 필요하다.
  4. Thread 생성 비용이 많이 든다.
  5. 작업 요청이 들어올 때 마다 Thread를 생성하면 최종적인 요청 처리 시간이 증가한다.
- 문제 2 : Thread가 너무 많으면 여러가지 문제를 발생시킨다.
1. Process 의 처리 속도보다 빠르게 대량의 요청이 들어올때
  2. 새로운 Thread가 무제한적으로 계속 생성된다.
  3. Thread가 많아 질수록 메모리를 차지하고 Context-Switching이 더 자주 발생한다.

4. 메모리 문제가 발생할 수 있고, CPU 오버헤드가 증가한다.

## 해결 방안



### Thread Pool

Thread 를 허용된 개수 안에서 사용하도록 제한하는 시스템



Thread Pool

- 스레드 풀은 매번 생성 및 수거 요청이 올 때 스레드를 생성하고 수거하는 것이 아닌, 스레드 사용자가 설정해둔 개수만큼 미리 생성해두는 것이다.




- 해결 1 : 미리 만들어 놓은 Thread를 재사용할 수 있기 때문에 새로운 Thread를 생성하는 비용을 줄일 수 있다.
- 해결 2 : 사용할 Thread 개수를 제한하기 때문에 무제한적으로 스레드가 생성되지 않아서 방지 가능하다.

## Tomcat의 Thread Pool

톰캣 튜닝 맛보기 - maxConnections, acceptCount, maxThreads

hudi.blog

 <https://hudi.blog/tomcat-tuning-exercise/>

- **Max-Connections**

Tomcat이 최대로 동시에 처리할 수 있는 Connection의 개수

Web 요청이 들어오면 Tomcat의 Connection을 생성하면서 요청된 작업을 Thread Pool의 Thread에 연결한다.

- **Accept-Count**

Max-Connections 이상의 요청이 들어왔을 때 사용하는 대기열 Queue의 사이즈

Max-Connections 와 Accept-Count 이상의 요청이 들어왔을 때 추가적으로 들어오는 요청은 거절될 수 있다.

## SpringBoot 설정을 통한 Tomcat Thread Pool 설정

- **server.tomcat.threads.max**

- Thread Pool 에서 사용할 최대 스레드 개수, 기본 값은 200
- 서버 어플리케이션이 동시에 처리할 수 있는 요청 개수와 관련 있다.
- 요청 수에 비해 너무 많게 설정 ➡ 놓고 있는 스레드가 많아져 비효율 발생
- 너무 적게 설정 ➡ 동시 처리 요청 수가 줄어든다. 평균 응답 시간, TPS 감소
- 기본적으로 Thread 가 많아지면 CPU 오버헤드와 메모리에서 문제가 생길 수 있다는 걸 고려해야 한다.

- **server.tomcat.threads.min-spare**

- Thread Pool 에서 최소한으로 유지할 Thread 개수, 기본 값은 10
- 너무 많이 설정 ➡ Thread Pool 이 항상 유지해야 할 Thread 수가 너무 많아진다.
- 적절하게 설정 ➡ 적은 수의 요청에서 새로운 스레드를 만들 필요없이 요청을 효과적으로 처리 할 수 있다.
- 잘못 설정했을 때 사용하지 않는 Thread 가 메모리를 차지하면서 비효율을 발생시킨다는 걸 고려해야 한다.

- **server.tomcat.threads.max-connections**

- 동시에 처리할 수 있는 최대 Connection의 개수, 기본 값은 8192
- 사실상 서버의 실질적인 동시 요청 처리 개수라고 생각할 수 있습니다.
- Blocking IO 는 1 Connection 1Thread 이다.
- Tomcat 8 이후 버전부터는 무조건 Non-Blocking IO를 사용한다.(N Connction 1 Thread)
- Non-Blocking에서는 Thread Pool의 최대 스레드 개수보다 많은 양의 Connction을 유지할 수 있다.
- Non-Blocking에서는 최대 Thread의 개수보다 적거나 같은 수의 max-connections를 설정하는 것은 비효율적인 설정이 될 수 있다.

- **server.tomcat.threads.accept-count**

- max-connections 이상의 요청이 들어왔을 때 사용하는 요청 대기열 Queue의 사이즈 기본 값은 100
- 너무 크게 설정 ➡ 대기열이 커지면서 메모리 문제를 유발할 수 있다.
- 너무 작게 설정 ➡ 요청이 몰렸을 때 들어오는 요청들을 거절해 버릴 수 있다.
- 이 설정을 하는 이유 중 하나는 부적절하거나 잘못된 요청이 한번에 너무 많이 들어와 서버에 장애를 발생시키는 것을 방지하기 위함도 있다.

## Java 에서의 Thread Pool

---

## Thread Pool 생성하기

- 자바는 `java.util.concurrent` 패키지에서 `Executors` 클래스를 제공한다. `Executors` 클래스의 정적 메소드를 사용하여 `ExecutorService` 라는 인터페이스의 구현체를 생성할 수 있다. 이 구현체가 바로 Thread Pool 이다.
- **초기 Thread 수** : Thread Pool 이 생성될 당시의 스레드 개수를 의미한다.
- **코어 Thread 수** : Thread Pool 내의 스레드가 제거되어도 남아있을 최소 스레드 개수를 의미한다.
- **최대 Thread 수** : Thread Pool 내 스레드의 최대 개수를 의미한다.

## CachedThreadPool

- 초기 Thread 수 : 0개
- 코어 Thread 수 : 0개
- 최대 Thread 수 : `Integer.MAX_VALUE`
- 작업이 들어올 때 마다 스레드를 생성
- 생성된 스레드가 60초 동안 아무일도 하지 않으면, 즉 유휴상태라면 해당 스레드를 제거한다.

```
// Thread Pool 생성
ExecutorService executorService = Executors.newCachedThreadPool();
```

## FixedThreadPool

- 초기 Thread 수 : 0

- 코어 Thread 수 : Thread Pool이 생성될 때 지정
- 최대 Thread 수 : Thread Pool이 생성될 때 지정
- 존재하는 쓰레드의 개수보다 작업량이 많으면 쓰레드를 생성
- 생성된 쓰레드는 유휴 상태가 오래되었다고 하더라도 제거하지 않는다.

```
// Thread Pool 생성 (Thread 수 200개)
ExecutorService executorService = Executors.newFixedThreadPool(200);
```

## ThreadPoolExecutor

- 사용자가 지정하여 Thread Pool을 생성할수도 있다. `ThreadPoolExecutor` 를 사용하는 것이다. 사실 위 두가지 방법도 내부적으로는 `ThreadPoolExecutor` 를 사용한다.

```
public static ExecutorService newFixedThreadPool(int nThreads) {
    return new ThreadPoolExecutor(nThreads, nThreads,
                                   TimeUnit.SECONDS, false,
                                   new SynchronousQueue<>());
}
```

```
ExecutorService myThreadPool = new ThreadPoolExecutor(
    3,    // 코어 Thread 수
    200,  // 최대 Thread 수
    120,  // 최대 유휴 시간
    TimeUnit.SECONDS, // 최대 유휴 시간 단위
    new SynchronousQueue<>() // 작업 큐
);
```



## Thread Pool 종료

```
threadPool.shutdown(); // (1)
List<Runnable> unprocessedTasks = threadPool.shutdownNow();
boolean complete = threadPool.awaitTermination(60, TimeUnit.SECONDS);
```

- **shutdown()**

- 현재 Thread가 처리중인 작업과 작업 큐에 대기하고 있는 작업을 모두 끝 마친 뒤 Thread Pool을 종료한다.

- **shutdownNow()**

- 현재 작업중인 Thread를 강제로 작업 중지하고, 작업 큐 대기열에 남아있는 작업을 `List<Runnable>` 로 반환한다.

- **awaitTermination()**

- `shutdown()` 을 먼저 호출하고, 전달된 timeout 시간 내로 모든 작업이 완료되면 true를, 완료하지 못하면 처리중인 Thread를 강제 종료하고 false를 반환한다.

## Thread Pool 에 작업 처리 요청

- **작업의 단위, Runnable와 Callable**

- Thread Pool에 작업 처리를 요청하기 위해서는 `Runnable` 또는 `Callable` 인터페이스의 구현체를 `ExecutorService` 에 전달해야한다. `Runnable` 은 작업 처리 완료 후 반환값이 없고, `Callable` 은 작업 처리 완료 후 반환값이 있다.

- **작업 처리 요청, execute()와 submit()**

- `ExecutorService` 는 작업 처리 요청을 위해 `execute()` 와 `submit()` 두가지 메소드를 제공한다.
- 차이점1. 처리 결과 반환
  - `execute()` 는 `Runnable` 만을 처리하며, 작업의 처리 결과를 반환받을 수 없다. 반면, `submit()` 은 `Runnable` 과 `Callable` 두가지를 처리하며 작업의 처리 결과를 `Future` 라는 인터페이스로 반환한다.
- 차이점2. 예외 발생 처리
  - `execute()` 는 Thread에서 작업 처리 도중 예외가 발생하면, Thread Pool에서 해당 예외를 제거하고 새로운 Thread를 생성한다. 반면, `submit()` 은 작업 처리 도중 예외가 발생해도 Thread를 제거하지 않고, 다음 작업에 재사용한다. `submit()` 이 오버헤드가 더 적으므로 따라서 `submit()` 을 사용하는 것이 일반적으로 권장된다.

## • 작업 처리 요청, `execute()`와 `submit()`

- `ExecutorService` 는 작업 처리 요청을 위해 `execute()` 와 `submit()` 두가지 메소드를 제공한다.
- 차이점1. 처리 결과 반환
  - `execute()` 는 `Runnable` 만을 처리하며, 작업의 처리 결과를 반환받을 수 없다. 반면, `submit()` 은 `Runnable` 과 `Callable` 두가지를 처리하며 작업의 처리 결과를 `Future` 라는 인터페이스로 반환한다.
- 차이점2. 예외 발생 처리
  - `execute()` 는 스레드에서 작업 처리 도중 예외가 발생하면, 스레드 풀에서 해당 예외를 제거하고 새로운 스레드를 생성

한다. 반면, `submit()` 은 작업 처리 도중 예외가 발생해도 쓰레드를 제거하지 않고, 다음 작업에 재사용한다. `submit()` 이 오버헤드가 더 적으므로 따라서 `submit()` 을 사용하는 것이 일반적으로 권장된다.

## Thread Pool 설정을 해야하는 이유



Thread Pool 은 응답시간과 TPS 에 영향을 주는 하나의 요소이다.



잘 조정된 Thread Pool 은 시스템의 성능을 끌어내고 안정적인 어플리케이션 운용을 가능하게 한다.



부적절하게 설정된 Thread Pool 은 병목 현상, CPU 오버헤드, 메모리 문제를 유발할 수 있다.

## 심화 내용

### 쓰레드풀 과 ForkJoinPool

쓰레드뚝뚝뚝!누구니?쓰레드 예요.. 프로그램(프로세스) 안에서 실행되는 하나의 흐름 단위예요. 내부에서 while 을 돌면 엄청 오랫동안 일을 할 수 도 있습니다.쓰레드 끼리는 값 (메모리) 을 공유 할 수 있습니다. 가끔

 <http://hamait.tistory.com/612>

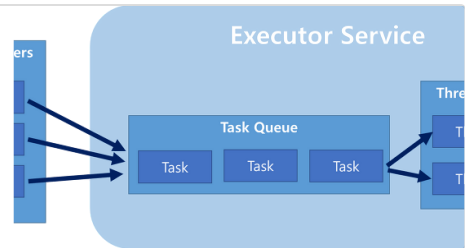


## 참고자료

### [Java] Thread Pool 이해하기

📖 함께 읽으면 좋은 글 프로세스와 스레드 Thread 이해하기 📄 목차  
Green Thread vs Native Thread Java는 어떤 스레드를 사용하는가  
Thread Pool ⚡ Green Thread vs Native Thread 💎 Green

🔗 <https://yeonyeon.tistory.com/271>



자바에서 스레드 풀 다뤄보기

🔗 <https://hudi.blog/java-thread-pool/>

**hudi.blog**

### [10분 테코톡] 조조그린의 Thread Pool

👉 우아한테크코스의 크루들이 진행하는 10분 테코토크입니다. 🧑‍🎓

'10분 테코톡'이란 우아한테크코스 과정을 진행하며 크루(수강생)들이

🔗 <https://www.youtube.com/watch?v=um4rYmQleRE>

**Thread Pool**

**조조그린**



### About the Thread Pool Example - Multithreaded Programming Guide

This book covers the POSIX and Oracle Solaris threads APIs, programming with synchronization objects, and compiling multithreaded programs. This guide is for developers who want to use multithreading to separate a process into independent execution threads, improving application

🔗 [https://docs.oracle.com/cd/E26502\\_01/html/E35303/ggfb.html#gggedp](https://docs.oracle.com/cd/E26502_01/html/E35303/ggfb.html#gggedp)