**JSoup와 thread**

**학습목표**

JSoup을 사용하여 웹페이지들을 가져오는 작업을 멀티 스레드로 구현해서 속도 개선

**목차**

[1. 배경 지식 2](#_Toc469946464)

[1) thread start & join 2](#_Toc469946465)

[2) thread를 활용한 속도 개선 2](#_Toc469946466)

[2. Example12 3](#_Toc469946467)

[1) GetArticleBody.java 3](#_Toc469946468)

[2) Example12.java 5](#_Toc469946469)

[3. Example13 7](#_Toc469946470)

[1) GetArticleList.java 7](#_Toc469946471)

[2) Example13.java 9](#_Toc469946472)

[3) 문제점 10](#_Toc469946473)

[4. pool1 11](#_Toc469946474)

[1) ExecutorService 클래스 11](#_Toc469946475)

[2) GetArticleBody.java 11](#_Toc469946476)

[3) GetArticleList.java 12](#_Toc469946477)

[4) Example01.java 14](#_Toc469946478)

# 배경 지식

## thread start & join

Thread 객체는 그냥 Java 객체일 뿐이고, 스레드(실행흐름)이 아니다.

Thread 객체의 start 메소드를 호출할 때, 새 스레드(새 실행흐름)이 생성되어

run 메소드를 실행하기 시작한다.

객체는 메모리(RAM)에 저장된 데이터이다.

Java 소스코드가 컴파일되어 생성된 바이트코드(bytecode)는 메모리(RAM)에 저장된 실행 명령이다.

CPU 코어가 바이트코드를 하나씩 읽어서 순서대로 실행하는 것이 실행흐름(thread) 이다.

|  |
| --- |
| main() {  // main 메소드부터 실행을 시작한 메인 스레드가 아래의 코드들을 실행하기 시작한다.    Thread objA = new Thread(); // Thread 객체가 하나 생성됨. 이 객체의 이름을 objA 라고 하자.  objA.start(); // 새 스레드(실행흐름)이 생성되어 run 메소드를 실행하기 시작한다.  // 현재 메인 스레드는 main 메소드를 실행하고 있고,  // 새 스레드는 run 메소드를 실행하고 있다.  objA.join(); // 이 문장을 실행하는 것은 메인 스레드이다. 메인 스레드는  // objA.start()를 호출하여 생성된 새 스레드가 run 메소드에서 리턴하여  // 종료될 때까지 대기(sleep) 상태가 된다.  } |

## 공유되는 객체 보호

멀티 스레드(여러 실행 흐름)에 의해서 공유되는 객체에 대한 작업은 보호되어야 한다.

객체에 대한 작업이란 그 객체의 메소드를 호출하는 것을 말한다.

아래와 같이 구현되어야 한다.

|  |
| --- |
| synchronized (공유되는 객체) {  공유되는 객체의 메소드 호출1  공유되는 객체의 메소드 호출2  } |

## 클래스 멤버 변수 보호

멀티 스레드(여러 실행 흐름)에 의해서 공유되는 클래스 멤버 변수에 대한 읽기 쓰기 작업도 보호되어야 한다.

그런데 synchronized 키워드로 보호할 수 있는 것은 객체만 가능하고, 변수를 보호할 수는 없다.

따라서 변수 하나만 보호할 수는 없고, 그 변수가 들어있는 객체를 보호해야 한다.

클래스 멤버 변수는 클래스 객체에 들어있다.

예를 들어 Worker 클래스의 count 클래스 멤버 변수를 보호하는 코드는, 아래와 같이 구현되어야 한다.

|  |
| --- |
| class Worker implements Runnable {  static int count = 0;  @Override  void run() {  synchronized (Worker.class) {  ++count;  }  }  } |

## AtomicInteger 클래스

스레드 세이프(thread safe)하게 구현되어 있는 Integer 클래스이다.

스레드 세이프하다는 말은, 멀티 스레드(여러 실행 흐름)에서 공유되어도 안전하기 때문에 따로 보호할 필요 없다는 얘기이다.

유용한 메소드는 다음과 같다.

### int incrementAndGet()

this 정수 값에 1을 더하고 그 결과 값을 리턴한다.

### int decrementAndGet()

this 정수 값에 1을 빼고 그 결과 값을 리턴한다.

## thread를 활용한 속도 개선

|  |
| --- |
| Document document = Jsoup.connect(url).get(); |

위 코드는, 웹서버에 url을 요청하고, 그 결과로 웹서버가 전송하는 html 문서를 다 받은 다음에 리턴한다.

get 메소드가 리턴하는 Document 객체가 바로 그 전송된 html 문서이다.

게시판의 전체 글을 다 받으려면, 웹 서버에 몇 백개의 요청을 해야 한다.

44 페이지 x 15 게시글 = 600

Example11.java 에서는 이 요청을 하나씩 순서대로 실행했다.

요청을 하나씩 순서대로 하는 것 보다, 동시에 여러 요청을 하면 좀 더 빨리 작업을 끝낼 수 있을 것이다.

게시글 본문 가져오기, 게시글 목록 가져오기 작업을 멀티 스레드로 구현하자.

# Example12

## GetArticleBody.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package jsoup1;  import java.io.IOException;  import java.util.ArrayList;  import java.util.List;  import org.jsoup.Jsoup;  import org.jsoup.nodes.Document;  public class GetArticleBody extends Thread {  static final String articleUrl =  "http://www.skhu.ac.kr/board/boardread.aspx?idx=999&bsid=10004&searchBun=51";  static List<GetArticleBody> objectList = new ArrayList<GetArticleBody>();  Article article;    public GetArticleBody(Article article2) {  this.article = article2;  objectList.add(this);  }    @Override  public void run() {  try {  String url = articleUrl.replace("999", String.valueOf(article.getIdx()));  Document document = Jsoup.connect(url).get();  String body = document.select("span#ctl00\_ContentPlaceHolder1\_ctl00\_lbContents").get(0).html();  article.setBody(body);  } catch (IOException e) {  e.printStackTrace();  }  }    public static void joinAll() throws InterruptedException {  for (GetArticleBody obj : objectList)  obj.join();  }  } |

### GetArticleBody 클래스

Thread 클래스를 상속하여 게시글 본문을 받아오는 코드를 구현하였다.

게시글 본문을 받아오는 코드가 run 메소드에 구현되어 있다.

### run 메소드

Thread 객체의 start 메소드를 호출하면, 새 스레드(multi-thread)가 생성되어 run 메소드를 실행한다.

run 메소드가 리턴할 때, 스레드가 종료한다.

### objectList 멤버 변수

objectList 멤버 변수가 static 임에 주목하자.

objectList 변수가 참조하는 ArrayList에 GetArticleBody 객체 전체를 보관한다.

GetArticleBody 클래스 생성자에서, objectList 변수가 참조하는 ArrayList에 this 객체를 add 한다.

이렇게 하는 이유는, joinAll 메소드에서 필요하기 때문이다.

### joinAll 메소드

joinAll 메소드를 호출하면,

GetArticleBody 클래스 객체의 start 메소드를 호출하여 생성된 새 스레드들이 모두 종료할 때까지,

joinAll 메소드를 호출한 메인 스레드가 대기(sleep) 상태가 된다.

모든 새 스레드가 종료한 후에 메인 스레드가 리턴한다.

## Example12.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92 | /\*  \* get article body를 스레드로 실행한 덕분에 빨리 종료함  \*/  package jsoup1;  import java.io.IOException;  import java.util.Comparator;  import java.util.TreeSet;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  import org.jsoup.Jsoup;  import org.jsoup.nodes.Document;  import org.jsoup.nodes.Element;  import org.jsoup.select.Elements;  public class Example12 {    static final String listUrl =  "http://www.skhu.ac.kr/board/boardlist.aspx?curpage=999&bsid=10004&searchBun=51";  static class ArticleComparator implements Comparator<Article> {  @Override  public int compare(Article a1, Article a2) {  return a1.getIdx() - a2.getIdx();  }  }    public static void main(String[] args) throws Exception {  Article[] articles = getArticles();  for (Article article : articles) {  int length = Math.min(30, article.getBody().length());  System.out.printf("%d %s %s %s %s %s\n",  article.getIdx(),  article.getTitle(),  article.getAuthor(),  article.getDate(),  article.getCount(),  article.getBody().substring(0, length));  }  }  static Article[] getArticles() throws Exception {  String regex = "boardread.aspx\\?idx=([0-9]+)&";  Pattern pattern = Pattern.compile(regex);  TreeSet<Article> set = new TreeSet<Article>(new ArticleComparator());  int pageCount = getPageCount();    for (int i=1; i <= pageCount; ++i) {  System.out.printf("%d page\n", i);    String url = listUrl.replace("999", String.valueOf(i));  Document document = Jsoup.connect(url).get();  Elements tagList = document.select("table.board\_list tbody tr");    for (Element trTag : tagList ) {  Element aTag = trTag.select("td span a").get(0);  Matcher matcher = pattern.matcher(aTag.attr("href"));  String s = matcher.find() ? matcher.group(1) : null;  int idx = Integer.valueOf(s);  String title = aTag.text();  String author = trTag.select("td:nth-child(4)").text();  String date = trTag.select("td:nth-child(5)").text();  String count = trTag.select("td:nth-child(6)").text();    Article article = new Article();  article.setIdx(idx);  article.setTitle(title);  article.setAuthor(author);  article.setDate(date);  article.setCount(count);  new GetArticleBody(article).start();  set.add(article);  }  }  GetArticleBody.joinAll();  return set.toArray(new Article[0]);  }    static int getPageCount() throws IOException {  String url = listUrl.replace("999", "1");  Document document = Jsoup.connect(url).get();  Element aTag = document.select("a.nextL").get(0);  String regex = "boardlist.aspx\\?curpage=([0-9]+)&";  Pattern pattern = Pattern.compile(regex);  Matcher matcher = pattern.matcher(aTag.attr("href"));  String s = matcher.find() ? matcher.group(1) : "0";  int pageCount = Integer.valueOf(s);  return pageCount;  }  } |

(줄53)

게시글 목록 페이지를 가져오는 작업을 메인 스레드가 실행한다.

페이지 가져오기가 완료될 때까지 메인 스레드는 get 메소드에서 리턴하지 않는다.

페이지 가져오기가 완료된 후에 메인 스레드는 줄54로 넘어간다.

(줄72)

게시글 본문 가져오는 작업을 메인 스레드가 아닌 새 스레드로 실행한다.

따라서 메인 스레드는 게시글 본문 가져오기 작업이 끝날 때까지 대기하지 않기 때문에,

줄56~74의 for 루프를 빨리 실행하게 된다.

for 루프가 반복할 때 마다 줄72에서 새 스레드가 만들어져서 본문 가져오는 작업을 진행한다.

줄72에서 생성된 여러 스레드가 작업을 동시에 진행하게 된다.

# Example13

## GetArticleList.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68 | package jsoup1;  import java.io.IOException;  import java.util.ArrayList;  import java.util.List;  import java.util.TreeSet;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  import org.jsoup.Jsoup;  import org.jsoup.nodes.Document;  import org.jsoup.nodes.Element;  import org.jsoup.select.Elements;  public class GetArticleList extends Thread {  static final String listUrl =  "http://www.skhu.ac.kr/board/boardlist.aspx?curpage=999&bsid=10004&searchBun=51";  static List<GetArticleList> objectList = new ArrayList<GetArticleList>();  TreeSet<Article> set;  int page;    public GetArticleList(int page, TreeSet<Article> set) {  this.page = page;  this.set = set;  objectList.add(this);  }    @Override  public void run() {  try {  String regex = "boardread.aspx\\?idx=([0-9]+)&";  Pattern pattern = Pattern.compile(regex);    String url = listUrl.replace("999", String.valueOf(page));  Document document = Jsoup.connect(url).get();  Elements tagList = document.select("table.board\_list tbody tr");    for (Element trTag : tagList ) {  Element aTag = trTag.select("td span a").get(0);  Matcher matcher = pattern.matcher(aTag.attr("href"));  String s = matcher.find() ? matcher.group(1) : null;  int idx = Integer.valueOf(s);  String title = aTag.text();  String author = trTag.select("td:nth-child(4)").text();  String date = trTag.select("td:nth-child(5)").text();  String count = trTag.select("td:nth-child(6)").text();    Article article = new Article();  article.setIdx(idx);  article.setTitle(title);  article.setAuthor(author);  article.setDate(date);  article.setCount(count);  new GetArticleBody(article).start();  synchronized (set) {  set.add(article);  }  }  } catch (IOException e) {  e.printStackTrace();  }  }    public static void joinAll() throws InterruptedException {  for (GetArticleList obj : objectList)  obj.join();  }  } |

### GetArticleList 클래스

Thread 클래스를 상속하여 게시글 목록을 받아오는 코드를 구현하였다.

게시글 목록을 받아오는 코드가 run 메소드에 구현되어 있다.

### run 메소드

Thread 객체의 start 메소드를 호출하면, 새 스레드(multi-thread)가 생성되어 run 메소드를 실행한다.

run 메소드가 리턴할 때, 스레드가 종료한다.

### objectList 멤버 변수

objectList 멤버 변수가 static 임에 주목하자.

objectList 변수가 참조하는 ArrayList에 GetArticleList 객체 전체를 보관한다.

GetArticleList 클래스 생성자에서, objectList 변수가 참조하는 ArrayList에 this 객체를 add 한다.

이렇게 하는 이유는, joinAll 메소드에서 필요하기 때문이다.

### joinAll 메소드

joinAll 메소드를 호출하면,

GetArticleList 클래스 객체의 start 메소드를 호출하여 생성된 새 스레드들이 모두 종료할 때까지,

joinAll 메소드를 호출한 메인 스레드가 대기(sleep) 상태가 된다.

모든 새 스레드가 종료한 후에 메인 스레드가 리턴한다.

### 줄55~57

게시글들을 저장하는 TreeSet<TreeSet> 객체는 메인 스레드에 의해서 한 개만 생성된다.

GetArticleList 객체에 의해서 생성된 스레드들은 TreeSet<TreeSet> 객체 하나를 공유한다.

여러 스레드들 사이에 공유되는 객체의 메소드 호출은 synchronized 키워드로 보호되어야 한다.

## Example13.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | /\*  \* get articlle list로 스레드로 실행했더니, 동시에 너무 많은 스레드가 생성되어  \* 소켓 에러 발생함.  \*/  package jsoup1;  import java.io.IOException;  import java.util.Comparator;  import java.util.TreeSet;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  import org.jsoup.Jsoup;  import org.jsoup.nodes.Document;  import org.jsoup.nodes.Element;  public class Example13 {    static class ArticleComparator implements Comparator<Article> {  @Override  public int compare(Article a1, Article a2) {  return a1.getIdx() - a2.getIdx();  }  }    public static void main(String[] args) throws Exception {  Article[] articles = getArticles();  for (Article article : articles) {  int length = Math.min(30, article.getBody().length());  System.out.printf("%d %s %s %s %s %s\n",  article.getIdx(),  article.getTitle(),  article.getAuthor(),  article.getDate(),  article.getCount(),  article.getBody().substring(0, length));  }  }  static Article[] getArticles() throws Exception {  TreeSet<Article> set = new TreeSet<Article>(new ArticleComparator());  int pageCount = getPageCount();    for (int i=1; i <= pageCount; ++i) {  System.out.printf("%d page\n", i);  new GetArticleList(i, set).start();  }  GetArticleList.joinAll();  GetArticleBody.joinAll();  return set.toArray(new Article[0]);  }    static int getPageCount() throws IOException {  String url = "http://www.skhu.ac.kr/board/boardlist.aspx?curpage=1&bsid=10004&searchBun=51";  Document document = Jsoup.connect(url).get();  Element aTag = document.select("a.nextL").get(0);  String regex = "boardlist.aspx\\?curpage=([0-9]+)&";  Pattern pattern = Pattern.compile(regex);  Matcher matcher = pattern.matcher(aTag.attr("href"));  String s = matcher.find() ? matcher.group(1) : "0";  int pageCount = Integer.valueOf(s);  return pageCount;  }  } |

(줄46)

게시글 목록 페이지 가져오는 작업을 메인 스레드가 아닌 새 스레드로 실행한다.

따라서 메인 스레드는 게시글 목록 페이지 가져오기 작업이 끝날 때까지 대기하지 않기 때문에,

줄44~47의 for 루프를 빨리 실행하게 된다.

for 루프가 반복할 때 마다 줄46에서 새 스레드가 만들어져서 게시글 목록 페이지 가져오는 작업을 진행한다.

줄46에서 생성된 여러 스레드가 작업을 동시에 진행하게 된다.

## 문제점

스레드를 무한히 많이 만든다고 해서 작업이 빨라지는 것이 아니다.

적정 숫자 이상으로 스레드가 많아지면, 오히려 느려질 수 있다.

그리고 너무 많은 스레드들이 한정된 자원을 동시에 사용하면,

자원 부족 에러가 발생할 수 있다.

인터넷에 연결하기 위한 Socket 객체도 한정된 자원들 중 하나이다.

Example13에서 너무 많은 스레드들이 만들어지고, 이들이 인터넷에 연결하려고 해서

Socket 부족 에러가 발생한다.

스레드 생성 수를 통제해야 한다.

Java 표준 라이브러리의 ExecutorService 클래스를 이용하면 쉽게 구현할 수 있다.

# pool1

## ExecutorService 클래스

정해진 숫자 만큼의 스레드만 생성되고, 이 스레드들이 동시에 작업을 진행한다.

정해진 숫자 만큼 생성된 스레드들이 대기하고 있는 곳을 스레드 풀(thread pool)이라고 부른다.

작업은 Runnable 인터페이스를 implements 하여 구현해야 한다.

작업 객체를 생성하고, ExecutorService.execute(작업객체) 메소드를 호출하여 작업을 제출한다.

제출된 작업은 작업 큐(job queue)에 넣어져서 대기하고 있게 된다.

작업 큐에 작업이 추가되면, 스레드 풀에 대기하고 있던 스레드가 스레드 풀에서 나와서,

작업 큐에서 작업을 꺼내서 실행한다.

스레드가 작업을 마치면, 작업 큐에서 그 다음 작업을 꺼내서 실행한다.

작업 큐에 작업이 남아 있지 않으면, 스레드는 스레드 풀에 들어가 다시 대기 상태가 된다.

## GetArticleBody.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | package pool1;  import java.io.IOException;  import org.jsoup.Jsoup;  import org.jsoup.nodes.Document;  import jsoup1.Article;  public class GetArticleBody implements Runnable {  static int count = 0;  static final String articleUrl =  "http://www.skhu.ac.kr/board/boardread.aspx?idx=999&bsid=10004&searchBun=51";  Article article;    public GetArticleBody(Article article) {  this.article = article;  synchronized (GetArticleBody.class) {  ++count;  }  }    @Override  public void run() {  try {  String url = articleUrl.replace("999", String.valueOf(article.getIdx()));  Document document = Jsoup.connect(url).get();  String body = document.select("span#ctl00\_ContentPlaceHolder1\_ctl00\_lbContents").get(0).html();  article.setBody(body);  } catch (IOException e) {  e.printStackTrace();  }  synchronized (GetArticleBody.class) {  --count;  }  }    public static void joinAll() throws InterruptedException {  while (count > 0)  Thread.sleep(100);  }  } |

Thread 클래스를 상속하지 않고, Runnable 인터페이스를 implements 했음에 주목하자.

GetArticleBody 객체는 Thread 클래스의 객체가 아니기 때문에, join 메소드를 사용할 수 없다.

그래서 static 멤버 변수인 count 멤버 변수를 사용해서,

GetArticleBody 작업이 모두 끝났는지 확인하기 위한 joinAll 메소드를 구현하였다.

생성된 GetArticleBody 객체의 수가 count 변수에 누적된다.

GetArticleBody 객체의 run 작업이 완료할 때 마다 count 변수 값이 1씩 감소한다.

count 값이 0 이 되면, GetArticleBody 객체들의 작업이 모두 완료된 것이다.

count 멤버 변수는 static 멤버 변수이므로, 멀티 스레드에 의해서 공유된다.

## GetArticleList.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | package pool1;  import java.io.IOException;  import java.util.TreeSet;  import java.util.concurrent.ExecutorService;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  import org.jsoup.Jsoup;  import org.jsoup.nodes.Document;  import org.jsoup.nodes.Element;  import org.jsoup.select.Elements;  import jsoup1.Article;  public class GetArticleList implements Runnable {  static int count = 0;  static final String listUrl =  "http://www.skhu.ac.kr/board/boardlist.aspx?curpage=999&bsid=10004&searchBun=51";  ExecutorService executorService;  TreeSet<Article> set;  int page;    public GetArticleList(int page, TreeSet<Article> set, ExecutorService executorService) {  this.page = page;  this.set = set;  this.executorService = executorService;  synchronized (GetArticleList.class) {  ++count;  }  }    @Override  public void run() {  try {  String regex = "boardread.aspx\\?idx=([0-9]+)&";  Pattern pattern = Pattern.compile(regex);    String url = listUrl.replace("999", String.valueOf(page));  Document document = Jsoup.connect(url).get();  Elements tagList = document.select("table.board\_list tbody tr");    for (Element trTag : tagList ) {  Element aTag = trTag.select("td span a").get(0);  Matcher matcher = pattern.matcher(aTag.attr("href"));  String s = matcher.find() ? matcher.group(1) : null;  int idx = Integer.valueOf(s);  String title = aTag.text();  String author = trTag.select("td:nth-child(4)").text();  String date = trTag.select("td:nth-child(5)").text();  String count = trTag.select("td:nth-child(6)").text();    Article article = new Article();  article.setIdx(idx);  article.setTitle(title);  article.setAuthor(author);  article.setDate(date);  article.setCount(count);  executorService.execute(new GetArticleBody(article));  synchronized (set) {  set.add(article);  }  }  } catch (IOException e) {  e.printStackTrace();  }  synchronized (GetArticleList.class) {  --count;  }  }    public static void joinAll() throws InterruptedException {  while (count > 0)  Thread.sleep(100);  }  } |

(줄19,23,26)

GetArticleList 클래스는 생성자 파라미터로 ExecutorService 클래스 객체를 받아서

executorService 멤버 변수에 대입한다.

(줄58)

기사 본문을 서버에서 가져오는 작업을 ㄷxecutorService에 제출한다.

제출된 작업은 executorService 내부의 작업 큐에 등록된다.

executorService 내부의 스레드들 중의 하나에 의해 이 작업이 실행될 것이다.

### Example3 예제와 다른 점.

Example3 예제의 코드

|  |
| --- |
| new GetArticleBody(article).start(); |

게시글 본문 가져오는 작업을, 새 스레드(실행 흐름)가 생성되어 실행한다.

스레드가 생성되는 수를 통제하기 어렵다.

pool1 예제의 코드

|  |
| --- |
| executorService.execute(new GetArticleBody(article)); |

스레드 풀에 대기하고 있던 스레드들 중 하나가, 게시글 본문 가져오는 작업을 실행한다.

스레드가 정해진 수 만큼만 생성된다.

## Example01.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | package pool1;  import java.io.IOException;  import java.util.Comparator;  import java.util.TreeSet;  import java.util.concurrent.ExecutorService;  import java.util.concurrent.Executors;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  import org.jsoup.Jsoup;  import org.jsoup.nodes.Document;  import org.jsoup.nodes.Element;  import jsoup1.Article;  public class Example01 {  static ExecutorService executorService = Executors.newFixedThreadPool(50);  static class ArticleComparator implements Comparator<Article> {  @Override  public int compare(Article a1, Article a2) {  return a1.getIdx() - a2.getIdx();  }  }    public static void main(String[] args) throws Exception {  Article[] articles = getArticles();  for (Article article : articles) {  int length = Math.min(30, article.getBody().length());  System.out.printf("%d %s %s %s %s %s\n",  article.getIdx(),  article.getTitle(),  article.getAuthor(),  article.getDate(),  article.getCount(),  article.getBody().substring(0, length));  }  }  static Article[] getArticles() throws Exception {  TreeSet<Article> set = new TreeSet<Article>(new ArticleComparator());  int pageCount = getPageCount();    for (int i=1; i <= pageCount; ++i) {  System.out.printf("%d page\n", i);  executorService.execute(new GetArticleList(i, set, executorService));  }  GetArticleList.joinAll();  GetArticleBody.joinAll();  return set.toArray(new Article[0]);  }    static int getPageCount() throws IOException {  String url = "http://www.skhu.ac.kr/board/boardlist.aspx?curpage=1&bsid=10004&searchBun=51";  Document document = Jsoup.connect(url).get();  Element aTag = document.select("a.nextL").get(0);  String regex = "boardlist.aspx\\?curpage=([0-9]+)&";  Pattern pattern = Pattern.compile(regex);  Matcher matcher = pattern.matcher(aTag.attr("href"));  String s = matcher.find() ? matcher.group(1) : "0";  int pageCount = Integer.valueOf(s);  return pageCount;  }  } |

|  |
| --- |
| ExecutorService executorService = Executors.newFixedThreadPool(50) |

50개의 스레드만 생성한다.

CPU 계산이 많은 작업일 수록 스레드 수가 적어야 한다.

하드디스크, 네트웍, 프린터 등 느린 장치를 사용할 일이 많은 작업일 수록 스레드 수가 많아야 한다.

|  |
| --- |
| executorService.execute(new GetArticleList(i, set, executorService)); |

이 코드에 의해서 새 스레드(실행 흐름)가 생성되지 않는다.

스레드 풀에 대기하고 있던 스레드들 중 하나가, 게시글 목록 가져오는 작업을 실행한다.

## 개선할 점

웹서버로부터 웹페이지를 받아오는 작업에서, 드물지만 에러가 발생할 수 있다.

에러가 발생한 경우, 조금 있다가 그 웹페이지를 다시 받아와야 한다.

# pool2

## GetArticleBody.jaa

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | package pool2;  import java.io.IOException;  import java.util.concurrent.atomic.AtomicInteger;  import org.jsoup.Jsoup;  import org.jsoup.nodes.Document;  public class GetArticleBody implements Runnable {  static AtomicInteger count = new AtomicInteger(0);  static final String articleUrl =  "http://www.skhu.ac.kr/board/boardread.aspx?idx=999&bsid=10004&searchBun=51";  Article article;    public GetArticleBody(Article article) {  this.article = article;  count.incrementAndGet();  }    @Override  public void run() {  boolean success = false;  while (success == false) {  try {  String url = articleUrl.replace("999", String.valueOf(article.getIdx()));  Document document = Jsoup.connect(url).get();  String body = document.select("span#ctl00\_ContentPlaceHolder1\_ctl00\_lbContents").get(0).html();  article.setBody(body);  success = true;  } catch (IOException e) {  e.printStackTrace();  }  }  count.decrementAndGet();  }      public static void joinAll() throws InterruptedException {  while (count.intValue() > 0)  Thread.sleep(100);  }  } |

AtomicInteger 클래스를 사용하여 count를 구현하였다.

AtomicInteger 클래스는 스레드 세이프하기 때문에, synchronized 키워드를 사용하여 보호할 필요가 없다.

(줄21~32)

서버로부터 웹페이지를 받아오는 작업에서 에러가 발생하면, 그 작업을 다시 반복한다.

만약 서버가 죽었으면, 이 코드는 무한히 반복되는 문제가 있다.

반복 횟수를 제한하도록 개선할 필요가 있다.

## GetArticleList.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73 | package pool2;  import java.io.IOException;  import java.util.Set;  import java.util.concurrent.ExecutorService;  import java.util.concurrent.atomic.AtomicInteger;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  import org.jsoup.Jsoup;  import org.jsoup.nodes.Document;  import org.jsoup.nodes.Element;  import org.jsoup.select.Elements;  public class GetArticleList implements Runnable {  static AtomicInteger count = new AtomicInteger(0);  static final String listUrl =  "http://www.skhu.ac.kr/board/boardlist.aspx?curpage=999&bsid=10004&searchBun=51";  ExecutorService executorService;  Set<Article> set;  int page;    public GetArticleList(int page, Set<Article> set, ExecutorService executorService) {  this.page = page;  this.set = set;  this.executorService = executorService;  count.incrementAndGet();  }    @Override  public void run() {  boolean success = false;  while (success == false) {  try {  String regex = "boardread.aspx\\?idx=([0-9]+)&";  Pattern pattern = Pattern.compile(regex);    String url = listUrl.replace("999", String.valueOf(page));  Document document = Jsoup.connect(url).get();  Elements tagList = document.select("table.board\_list tbody tr");    for (Element trTag : tagList ) {  Element aTag = trTag.select("td span a").get(0);  Matcher matcher = pattern.matcher(aTag.attr("href"));  String s = matcher.find() ? matcher.group(1) : null;  int idx = Integer.valueOf(s);  String title = aTag.text();  String author = trTag.select("td:nth-child(4)").text();  String date = trTag.select("td:nth-child(5)").text();  String count = trTag.select("td:nth-child(6)").text();    Article article = new Article();  article.setIdx(idx);  article.setTitle(title);  article.setAuthor(author);  article.setDate(date);  article.setCount(count);  executorService.execute(new GetArticleBody(article));  set.add(article);  success = true;  }  } catch (IOException e) {  e.printStackTrace();  }  }  count.decrementAndGet();  }    public static void joinAll() throws InterruptedException {  while (count.intValue() > 0)  Thread.sleep(100);  }  } |

(줄59)

스레드 세이프한 객체이므로 synchronized 키워드를 사용하여 보호하지 않았다.

## Example1

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | package pool2;  import java.io.IOException;  import java.util.Collections;  import java.util.Comparator;  import java.util.Set;  import java.util.TreeSet;  import java.util.concurrent.ExecutorService;  import java.util.concurrent.Executors;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  import org.jsoup.Jsoup;  import org.jsoup.nodes.Document;  import org.jsoup.nodes.Element;  public class Example01 {  static ExecutorService executorService = Executors.newFixedThreadPool(50);  static class ArticleComparator implements Comparator<Article> {  @Override  public int compare(Article a1, Article a2) {  return a1.getIdx() - a2.getIdx();  }  }    public static void main(String[] args) throws Exception {  Article[] articles = getArticles();  for (Article article : articles) {  int length = Math.min(30, article.getBody().length());  System.out.printf("%d %s %s %s %s %s\n",  article.getIdx(),  article.getTitle(),  article.getAuthor(),  article.getDate(),  article.getCount(),  article.getBody().substring(0, length));  }  }  static Article[] getArticles() throws Exception {  TreeSet<Article> set = new TreeSet<Article>(new ArticleComparator());  Set<Article> threadSafeSet = Collections.synchronizedSet(set);  int pageCount = getPageCount();    for (int i=1; i <= pageCount; ++i) {  System.out.printf("%d page\n", i);  executorService.execute(new GetArticleList(i, threadSafeSet, executorService));  }  GetArticleList.joinAll();  GetArticleBody.joinAll();  return threadSafeSet.toArray(new Article[0]);  }    static int getPageCount() {  while (true) {  try {  String url =  "http://www.skhu.ac.kr/board/boardlist.aspx?curpage=1&bsid=10004&searchBun=51";  Document document = Jsoup.connect(url).get();  Element aTag = document.select("a.nextL").get(0);  String regex = "boardlist.aspx\\?curpage=([0-9]+)&";  Pattern pattern = Pattern.compile(regex);  Matcher matcher = pattern.matcher(aTag.attr("href"));  String s = matcher.find() ? matcher.group(1) : "0";  int pageCount = Integer.valueOf(s);  return pageCount;  } catch (IOException e) {  e.printStackTrace();  }  }  }  } |

(줄42~43)

스레드 세이프한 set 객체를 생성한다.

스레드 세이프한 객체이므로 synchronized 키워드를 사용하여 보호할 필요가 없다.