

## Использование библиотеки Executor для разработки многопоточных программ в Java

*Ервлева Регина Викторовна*

*Приамурский государственный университет имени Шолом-Алейхема*

*Студент*

*Ервлев Павел Андреевич*

*Приамурский государственный университет имени Шолом-Алейхема*

*Студент*

### Аннотация

В данной статье рассмотрен и описан процесс работы программы в многопоточном режиме. Произведена реализация данной библиотеки программы. Итоговым результатом будет являться рабочая программа, выполняющаяся в многопоточном режиме.

**Ключевые слова:** ExecutorService, Java, Spring

## Using the Executor library to develop multi -flow programs in Java

*Eroleva Regina Viktorovna*

*Sholom-Aleichem Priamursky State University*

*Student*

*Erolev Pavel Andreevich*

*Sholom-Aleichem Priamursky State University*

*Student*

### Abstract

This article will consider and describes the process of the program in multi -flow mode. This program will be implemented. The final result will be the work program performed in multi-wind mode.

**Keywords:** ExecutorService, Java, Spring

Многопоточная программа — это процесс разделения задач между несколькими потоками для повышения производительности. Когда происходит запуск любой программы, это выполняется одним потоком, называемым основным потоком.

Цель данной статьи – рассмотреть реализацию многопоточной программы.

А.А. Симаков описал разработку трассировки для JVM программ в использовании анализа ПО и обратного проектирования [1]. А.В. Колойдчук рассмотрела методы программирования кредитных рисков внедрения ИКТ с

помощью Java, Pascal и Basic [2]. А.Н. Иванов описал процесс реализации безопасного веб-приложения на Java [3]. В своей работе В.П. Великов, К.С. Добрева рассмотрели проблемы автоматизированной генерации ПО [4]. В своей статье А.А. Шейн описал процесс разработки программы, которая автоматически создает наборы классов в виде нативных объектов Java [5].

Чтобы сделать программу многопоточной, нужно породить потоки из основного потока. Есть два способа сделать это:

- Расширить класс «Thread»;
- Реализовать работающий интерфейс.

Внутренний класс «Thread» реализует интерфейс «Runnable» и имеет свои собственные методы, чтобы лучше управлять созданными потоками.

Начиная с Java 5, JDK предоставляет API «ExecutorService», упрощающий выполнение задач в асинхронном режиме. Он предоставляет пул потоков и API для назначения ему задач.

«ExecutorService» — это интерфейс, предоставляемый JDK. Его объект можно получить с помощью класса «Executors». Существует 5 типов пулов потоков, которые можно создать:

- `newSingleThreadExecutors` - Создаст объект, который будет использовать всего один поток;
- `newFixedThreadPool` - Создаст пул, который постоянно будет использовать заданное количество потоков;
- `newScheduledThreadPool` - Создаст пул, который будет планировать выполнение команд после определенной задержки;
- `newCachedThreadPool` - Создаст пул, которые будут создавать новые потоки, когда это необходимо, но при этом будет повторно использовать старые доступные потоки;
- `newWorkStealingPool` - Создаст пул для перевода работы с одних потоков на другие.

После создания пула потоков можно выполнить над ними необходимые действия:

- `execute` - выполняет данную команду в какой-то момент в будущем. Ничего не возвращает;
- `Submit` - выполняет переданную задачу и возвращает объект выполненной задачи;
- `invokeAll` - выполняет переданные задачи, возвращая список, содержащих их статус и результаты по завершению;
- `invokeAny` - выполняет переданные задачи, возвращая результат успешно выполненной задачи, без создания исключения, если такая имеется.
- `awaitTermination` - блокируется до тех пор, пока все задачи не завершат выполнение после запроса на завершение работы, или не истечет время ожидания, или текущий поток не будет прерван, в зависимости от того, что произойдет раньше;
- `isShutdown ()` – возвращает «true», если поток был закрыт;

- `isTerminated()` – возвращает «true», если задачи были выполнены после завершения работы;
- `shutdown()` - инициирует завершение работы, чтобы ранее отправленные задачи выполнялись, но новые задачи не принимались.;
- `shutdownNow()` - пытается остановить все активно выполняющиеся задачи, останавливает обработку ожидающих задач и возвращает список задач, ожидающих выполнения;

Класс «Future» используется для отслеживания хода выполнения одной или нескольких асинхронных задач. Он представляет собой результат асинхронного вычисления. Предоставляются методы для проверки завершения вычисления, ожидания его завершения и извлечения результата вычисления.

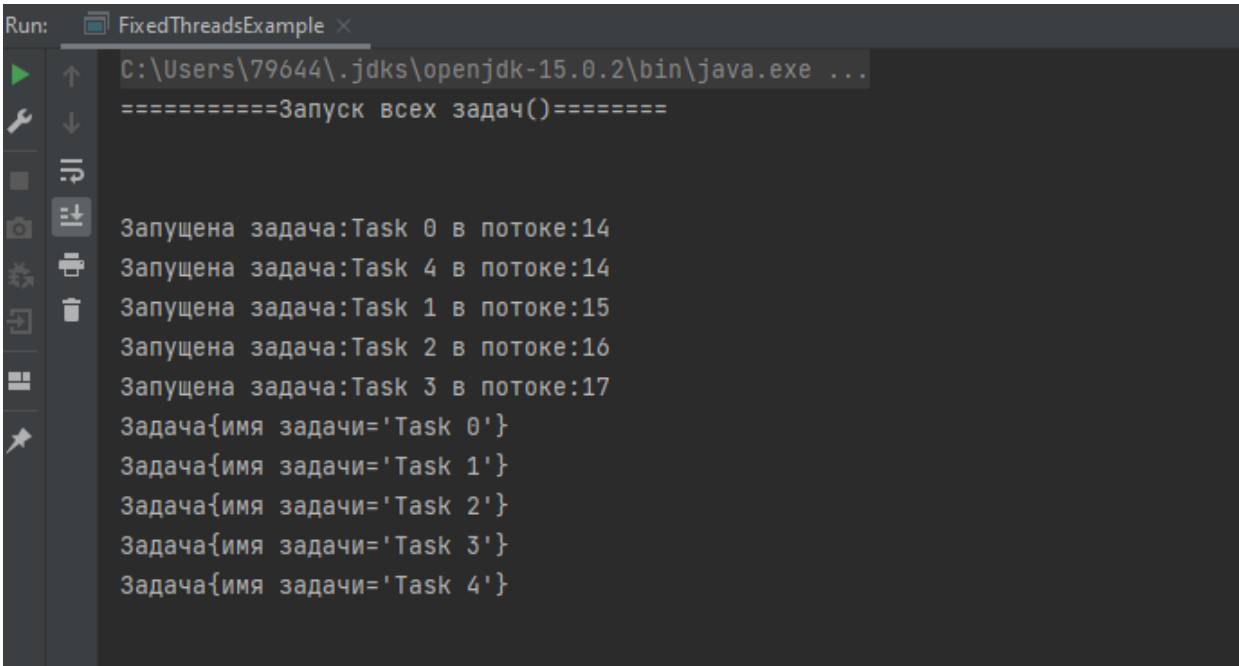
После завершения задач можно использовать метод «`get()`» для обработки всех данных.

Реализуем код для кэшированного пула потоков, учитывая, что класс `Task` реализует интерфейс `Runnable` (рис.1).

```
3 import com.example.executorserviceoverview.com.Task;
4 import java.util.ArrayList;
5 import java.util.List;
6 import java.util.concurrent.*;
7
8 public class CachedThreadsExample {
9     static ExecutorService executorService = Executors.newCachedThreadPool();
10
11     public CachedThreadsExample() {
12     }
13
14     public static void main(String[] args) throws InterruptedException, ExecutionException {
15         Future<String> future = null;
16         List<Future<Task>> futuresList = new ArrayList<>();
17         List<Callable<Task>> tasks = new ArrayList<>();
18         try {
19             for(int i =0;i<10;i++) {
20                 Task task = new Task("Task "+i);
21                 tasks.add(task);
22             }
23             futuresList = executorService.invokeAll(tasks);
24             executorService.shutdown();
25         }
26         catch(Exception ex) {
27             System.out.println("ERROR:"+ex);
28             executorService.shutdownNow();
29         }
30         finally {
31             executorService.awaitTermination(3000, TimeUnit.MILLISECONDS);
32             if(!executorService.isTerminated()) {
33                 executorService.shutdownNow();
34             }
35         }
36         for (Future<Task> futureRes : futuresList) {
37             String task = String.valueOf(futureRes.get());
38             System.out.println(task.toString());
39         }
40     }
41 }
42 }
```

Рисунок 1 – Реализация newCachedThreadPool

Теперь при запуске будут генерироваться задачи, которые будут распределены в потоки (рис.2).



```
Run: FixedThreadsExample x
C:\Users\79644\.jdk\openjdk-15.0.2\bin\java.exe ...
=====Запуск всех задач()=====
Запущена задача:Task 0 в потоке:14
Запущена задача:Task 4 в потоке:14
Запущена задача:Task 1 в потоке:15
Запущена задача:Task 2 в потоке:16
Запущена задача:Task 3 в потоке:17
Задача{имя задачи='Task 0'}
Задача{имя задачи='Task 1'}
Задача{имя задачи='Task 2'}
Задача{имя задачи='Task 3'}
Задача{имя задачи='Task 4'}
```

Рисунок 2 – Вывод программы

Как видно на рисунке выше, задача 0 и 4 переданы в один поток, но выполнение задачи 4 произошло только после выполнения всех остальных потоков.

В данной статье был рассмотрен класс «Future» и «Executors» для реализаций многопоточных программ, а также создан класс работающий в многопоточном режиме.

### Библиографический список

1. Симаков А.А. Java tracer. Программное средство для трассировки java программ // Заметки по информатике и математике. 2019. №3. С. 51-57.
2. Колойдчук А.В. Программирование инвестиционных и кредитных рисков введение икт с помощью pascal и visual basic // Формирование рыночных отношений в Украине. 2020 №4(227). С. 97-101.
3. Иванов А.Н. Разработка приложения с использованием веб-технологий java // Ученые заметки тогу. 2020. №4. С. 261-266.
4. Великов В.П., Добрева К.С. Генератор из диаграмм классов java в исходный код java // Информационные системы и технологии: управление и безопасность. 2014. №3. С. 14-23.
5. Шейн А.А. Генератор исходного кода на языке java по описанию бортовых компонентов decode // Современные проблемы науки и образования. 2016 №8. С. 18-25.