



Пакет java.util.concurrent

Некоторые классы и примеры использования



Беркунский Е.Ю., кафедра ИУСТ, НУК
eugeny.berkunsky@gmail.com
<http://www.berkut.mk.ua>

Блокировки. ReentrantLock

- Для управления доступом к общему ресурсу в качестве альтернативы оператору `synchronized` мы можем использовать блокировки. Функциональность блокировок заключена в пакете `java.util.concurrent.locks`
- Вначале поток пытается получить доступ к общему ресурсу. Если он свободен, то на него накладывает блокировку. После завершения работы блокировка с общего ресурса снимается. Если же ресурс не свободен и на него уже наложена блокировка, то поток ожидает, пока эта блокировка не будет снята.

Блокировки. ReentrantLock

Классы блокировок реализуют интерфейс Lock, который определяет следующие методы:

- **void lock():** ожидает, пока не будет получена блокировка
- **void lockInterruptibly() throws InterruptedException:** ожидает, пока не будет получена блокировка, если поток не прерван
- **boolean tryLock():** пытается получить блокировку, если блокировка получена, то возвращает true. Если блокировка не получена, то возвращает false. В отличие от метода lock() не ожидает получения блокировки, если она недоступна
- **void unlock():** снимает блокировку
- **Condition newCondition():** возвращает объект Condition, который связан с текущей блокировкой

ReentrantLock

- Организация блокировки в общем случае довольно проста: для получения блокировки вызывается метод `lock()`, а после окончания работы с общими ресурсами вызывается метод `unlock()`, который снимает блокировку.
- Объект `Condition` позволяет управлять блокировкой.
- Как правило, для работы с блокировками используется класс `ReentrantLock` из пакета `java.util.concurrent.locks`. Данный класс реализует интерфейс `Lock`.

Блокировки. ReentrantLock

- Применение условий в блокировках позволяет добиться контроля над управлением доступом к потокам.
- Условие блокировки представляет собой объект интерфейса **Condition** из пакета *java.util.concurrent.locks*.
- Применение объектов Condition во многом аналогично использованию методов wait/notify/notifyAll класса Object, которые были рассмотрены на прошлом занятии

Методы интерфейса Condition

- **await:** поток ожидает, пока не будет выполнено некоторое условие и пока другой поток не вызовет методы `signal/signalAll`.
Во многом аналогичен методу `wait` класса `Object`
- **signal:** сигнализирует, что поток, у которого ранее был вызван метод `await()`, может продолжить работу.
Применение аналогично использованию методу `notify` класса `Object`
- **signalAll:** сигнализирует всем потокам, у которых ранее был вызван метод `await()`, что они могут продолжить работу.
Аналогичен методу `notifyAll()` класса `Object`

Применение

- Сначала, используя эту блокировку, нам надо получить объект Condition:

```
ReentrantLock locker = new ReentrantLock();  
Condition condition = locker.newCondition();
```


Применение

- Как правило, сначала проверяется условие доступа. Если соблюдается условие, то поток ожидает, пока условие не изменится:

```
while (условие)  
    condition.await();
```


Применение

- После выполнения всех действий другим потокам подается сигнал об изменении условия:

```
condition.signal();
```

или

```
condition.signalAll();
```



Пробуем



Исполнители

- Concurrency API вводит понятие сервиса-исполнителя (ExecutorService) — высокоуровневую замену работе с потоками напрямую. Исполнители выполняют задачи асинхронно и обычно используют пул потоков, так что нам не надо создавать их вручную.
- Все потоки из пула будут использованы повторно после выполнения задачи, а значит, мы можем создать в приложении столько задач, сколько хотим, используя один исполнитель.

Класс Executors

Класс Executors предоставляет удобные методы-фабрики для создания различных сервисов исполнителей

- `public static ExecutorService newFixedThreadPool(int nThreads)`
- `public static ExecutorService newSingleThreadExecutor()`
- `public static ExecutorService newCachedThreadPool()`
- `public static ExecutorService newWorkStealingPool() //1.8`

...



Callable и Future

- Кроме Runnable, исполнители могут принимать другой вид задач, который называется Callable. Callable — это также функциональный интерфейс, но, в отличие от Runnable, он может возвращать значение.
- Callable-задачи также могут быть переданы исполнителям.
- Поскольку метод `submit()` не ждет завершения задачи, исполнитель не может вернуть результат задачи напрямую.
- Исполнитель возвращает специальный объект Future, у которого мы сможем запросить результат задачи.

Callable и Future

```
ExecutorService executor = Executors.newFixedThreadPool(1);  
Future<Integer> future = executor.submit(task);  
  
System.out.println("future done? " + future.isDone());  
  
Integer result = future.get();  
  
System.out.println("future done? " + future.isDone());  
System.out.print("result: " + result);
```

Задачи жестко связаны с сервисом исполнителей, и, если вы его остановите, попытка получить результат задачи выбросит исключение:

```
executor.shutdownNow();  
future.get();
```


Таймауты

- Любой вызов метода `future.get()` блокирует поток до тех пор, пока задача не будет завершена. В наихудшем случае выполнение задачи не завершится никогда, блокируя ваше приложение. Избежать этого можно, передав таймаут:

```
ExecutorService executor = Executors.newFixedThreadPool(1);
Future<Integer> future = executor.submit(() -> {
    try {
        TimeUnit.SECONDS.sleep(2);
        return 123;
    }
    catch (InterruptedException e) {
        throw new IllegalStateException("task interrupted", e);
    }
});
future.get(1, TimeUnit.SECONDS);
```




Пробуем



InvokeAll

- Исполнители могут принимать список задач на выполнение с помощью метода `invokeAll()`, который принимает коллекцию callable-задач и возвращает список из `Future`

```
ExecutorService executor = Executors.newWorkStealingPool();
List<Callable<String>> callables = Arrays.asList(
    () -> "task1",
    () -> "task2",
    () -> "task3");
executor.invokeAll(callables).stream().map(future -> {
    try {
        return future.get();
    }
    catch (Exception e) {
        throw new IllegalStateException(e);
    }
}).forEach(System.out::println);
```



Пробуем





НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ
КОРАБЛЕБУДУВАННЯ
ІМЕНІ АДМІРАЛА МАКАРОВА



QUESTIONS
& ANSWERS