java.util.concurrent: синхронизаторы, коллекции + исполнители

Помимо синхронизаторов и параллельных коллекций в java.util.concurrent можно встретить дополнительные инструменты для работы с многопоточным кодом - Thread Executors (исполнители потоков).

Thread executors - управляют исполнением потоков. На вершине иерархии располагается интерфейс Executor, предназначенный для запуска потока исполнения. Помимо Executor имеется интерфейс ExecutorService, который расширяет Executor и предоставляет дополнительные методы для управления исполнением. Помимо этого можно обнаружить 2 реализации интерфейса ExecutorService в таких классах как:

- **ThreadPoolExecutor** - обеспечивает поддержку управляемого пула потоков исполнения
- **ScheduledThreadPoolExecutor** - обеспечивает поддержку планирования пула потоков исполнения

А так же определяется класс Executors, в котором определены некоторые статические методы, упрощающие создание различных исполнителей (thread executor’ов)

Пул потоков

Пул потоков представляет собой совокупность потоков исполнения для тех или иных нужд. Вы можете создавать несколько потоков по отдельности, а можете создать целый ряд (пул), которые будут совместно решать ту или иную задачу. Плюс с такого подхода - сокращение времени, связанное с созданием множества отдельных, самостоятельных потоков.

Пул можно создавать через классы ThreadPoolExecutor или ScheduledThreadPoolExecutor напрямую, но чаще всего его создают через статические методы вспомогательного класса Executors, наиболее популярные из них

- static ExecutorService **newSingleThreadExecutor()** - метод, которые создаёт пул с одним рабочим потоком
- static ExecutorService **newFixedThreadPool**(int nThreads) - метод, который создаёт пул потоков исполнения, состоящий из указанного количества потоков
- static ExecutorService **newCachedThreadPool**() - метод, который формирует пул потоков, оптимизируя их за счет возможности повторного переиспользования их (кеширования)
- static ScheduledExecuteService **newScheduledThreadPool**(int corePoolSize) - метод, который создаёт пул потоков исполнения, в котором можно осуществлять планирования потоков исполнения

Пул потоков

наиболее популярные и часто применяемые методы у пулов потоков:

- **execute (Runnable command)** - работает по принципу “запустил и забыл”, принимает Runnable объект и выполняет его асинхронно
- **submit** - метод, который ставит задачу в очередь; метод возвращает объект интерфейса Future (про Future на следующем слайде) - отличная возможность контролировать состояние потока
  - **submit(Runnable task)** - можно передать уже привычный Runnable
  - **submit(Callable<V> task)** - а можно передать Callable (про Callable на следующем слайде)
- **shutdown()** - завершение потоков внутри ExecutorService

Future + Callable

Тесную связь с исполнителями имеют такие интерфейсы как Future и Callable.

Future - содержит значение, возвращаемое после выполнения потока, это такое значение, которое определяется как бы “на будущее” уже после того, как поток завершит своё выполнение.

Callable - определяет поток исполнения, который возвращает значение (довольно часто проводят условную связь между Runnable и Callable, где последний умеет возвращать значение, в отличие от первого)

Future + Callable

Future - обобщённый интерфейс, представляет значение, возвращаемое объектом типа Callable.

```
interface Future<V>
```

Чтобы получить значение, следует вызвать метод get() из интерфейса Future:

- V get() - ожидание получения результатов может длиться долго
- V get(long timeout, TimeUnit unit) - ожидание результатов результата длится только в течение определённого периода времени, определяемого параметром timeout, в единицах, обозначенных в параметре unit.

Future + Callable

Callable - механизм, который представляет поток исполнения, возвращающий значение. Объекты Callable используются, например, для вычисления чего-либо и возвращения результата вызывающему потоку исполнения. Это довольно популярный и эффективный способ, поскольку он облегчает написание кода для самых разных числовых расчётов, где промежуточные, частичные результаты вычисляются одновременно. Так же его можно использовать и для запуска потока исполнения, возвращающего код состояния, который может быть гарантией успешности (или не успешности) выполнения потока.

Callable - обобщённый интерфейс:

```
interface Callable<V>
```

У интерфейса Callable определяется единственный метод:

```
V call()
```

В методе call() определяется проблема, которую необходимо решить, когда функционал будет определён и выполнен, возвращается результат, но если результат нельзя выполнить, то будет сгенерировано исключение.

Рекомендации

- Каждая задача уникальна, поэтому стоит подходить внимательно к выбору того или иного пула потоков, который сможет помочь, а не навредить
- Пул потоков (любой) должен быть явно завершён с помощью метода shutdown(), забывчивость завершения пула потоков создаст неуместную ситуацию с тем, что ваш код будет работать бесконечно
- Не стоит ставить в очередь те задачи, которые в это же самое время ожидают результатов от других задач, это приводит к неразрешимости ситуации
- Стоит быть осторожным, применяя потоки для длительных операций, т.к. такой подход может создать бесконечно долгий процесс ожидания, как итог - утечка ресурсов на поддержание потоков, которые находятся в стадии ожидания. Лучше декомпозировать долгий процесс на более мелкие операции.