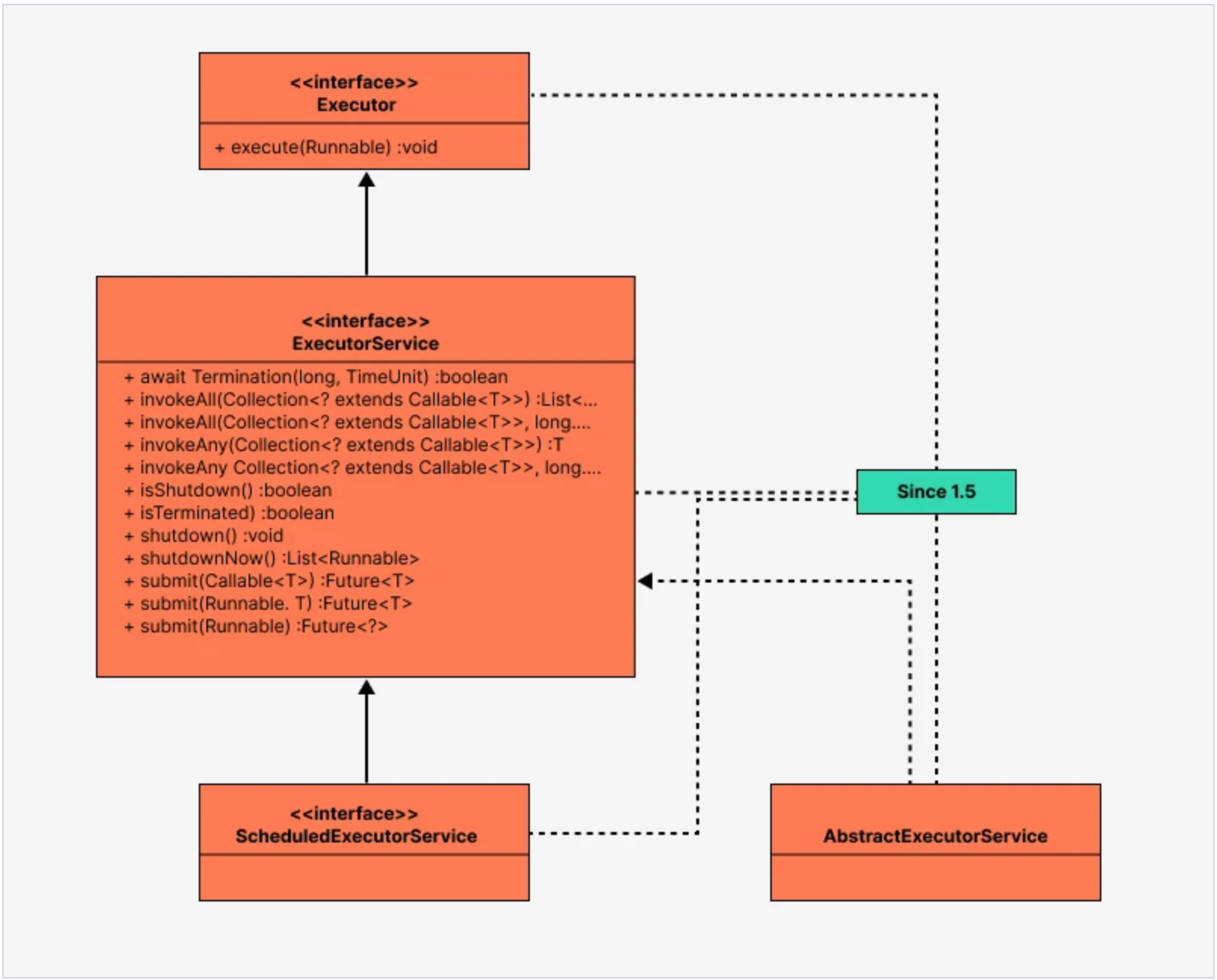
Executor Services

JSP & Servlets
19 уровень, 6 лекция

ОТКРЫТА

Executor



Executor — базовый интерфейс для классов, который реализует запуск ***Runnable*** задач. Тем самым обеспечивается помощь с добавлением задачи и способом ее запуска.

ExecutorService — интерфейс, который расширяет свойства ***Executor*** и который описывает сервис для запуска ***Runnable*** или ***Callable*** задач. Методы submit на вход принимают задачу в виде ***Callable*** или ***Runnable***, а в качестве возвращаемого значения идет Future, через который ты можешь получить результат.

Метод `invokeAll` отвечает за выполнение задач с возвращением списка задач с их статусом и результатами завершения.

Метод `invokeAny` отвечает за выполнение задач с возвращением результата успешно выполненной задачи (то есть без создания исключения), если таковые имеются.

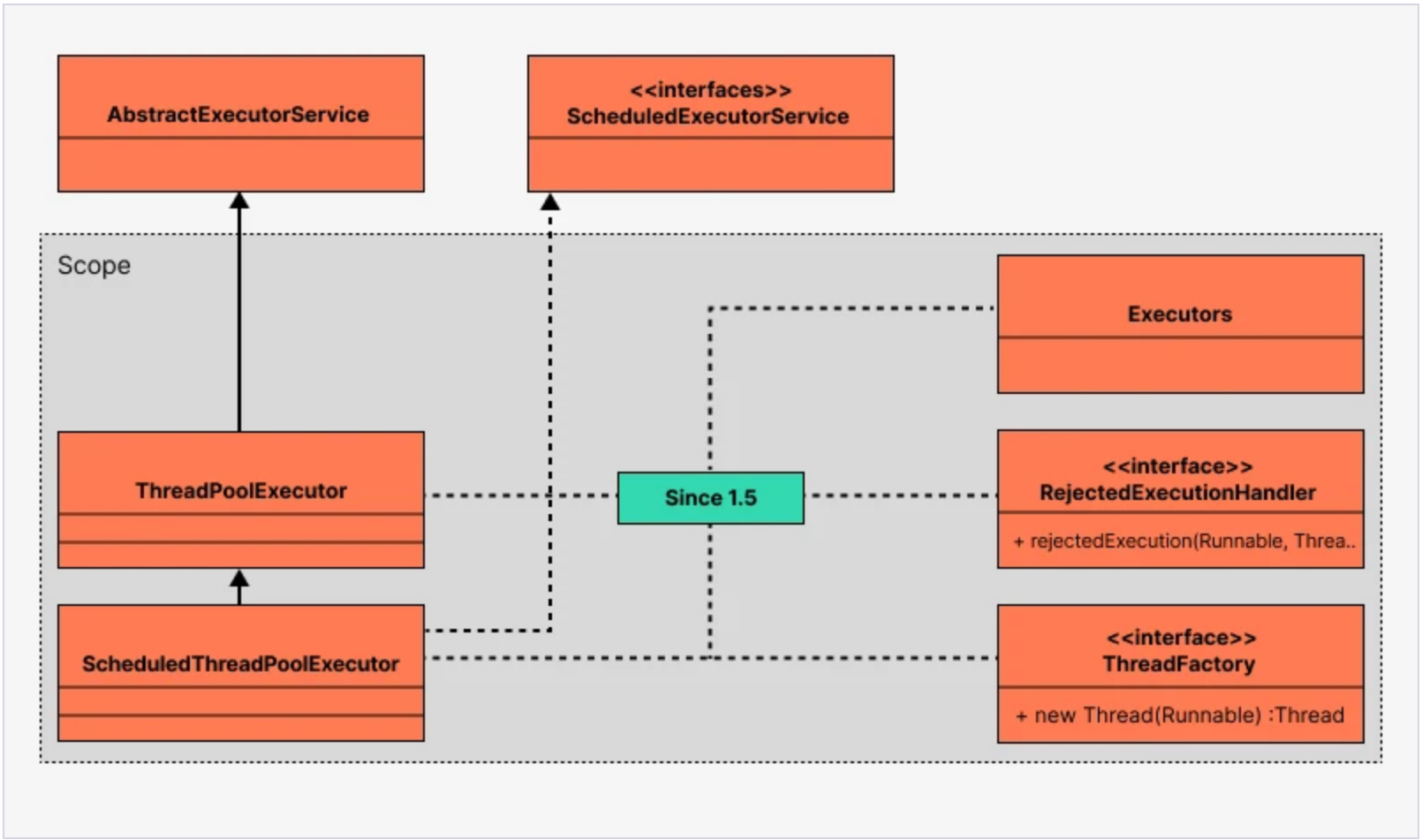
ScheduledExecutorService — данный интерфейс добавляет возможность запускать отложенные задачи с определенной задержкой или определенным периодом.

`AbstractExecutorService` — абстрактный класс для построения ***ExecutorService***'а. Внутри есть имплементация методов `submit`, `invokeAll`, `invokeAny`. От этого класса наследуются ***ThreadPoolExecutor***, ***ScheduledThreadPoolExecutor*** и ***ForkJoinPool***.

```
1  public static void main(String[] args) {
2     ScheduledExecutorService scheduledExecutorService = Executors.newScheduledThreadPool(1);
3     Callable<String> task = () -> {
```

```
        System.out.println(Thread.currentThread().getName());
        return Thread.currentThread().getName();
    };
    scheduledExecutorService.schedule(task, 10, TimeUnit.SECONDS);
    scheduledExecutorService.shutdown();
}
```

ThreadPoolExecutor



`Executors` — класс-фабрика для создания ***ThreadPoolExecutor***, ***ScheduledThreadPoolExecutor***. Если нужно создать один из этих пулов, то эта фабрика именно то, что нужно. Содержатся разные адаптеры Runnable-Callable, PrivilegedAction-Callable, PrivilegedExceptionAction-Callable и другие. Имеет статические методы для создания разных ***ThreadPool***.

ThreadPoolExecutor — реализует интерфейсы ***Executor*** и ***ExecutorService*** и разделяет создание задачи и ее выполнение. Нам необходимо реализовать объекты ***Runnable*** и отправить их исполнителю, а ***ThreadPoolExecutor*** отвечает за их исполнение, создание экземпляров и работу с потоками.

ScheduledThreadPoolExecutor — в дополнение к методам ***ThreadPoolExecutor*** создает пул потоков, который может планировать выполнение команд после заданной задержки или для периодического выполнения.

`ThreadFactory` — это объект, который создает новые потоки по требованию. Нам необходимо передать экземпляр в метод `Executors.newSingleThreadExecutor(ThreadFactory threadFactory)`.

```
ExecutorService executorService = Executors.newSingleThreadExecutor(new ThreadFactory() {
 @Override public Thread newThread(Runnable r) {
Thread thread = new Thread(r, "MyThread");
thread.setPriority(Thread.MAX_PRIORITY);
 return thread; }
 });
```

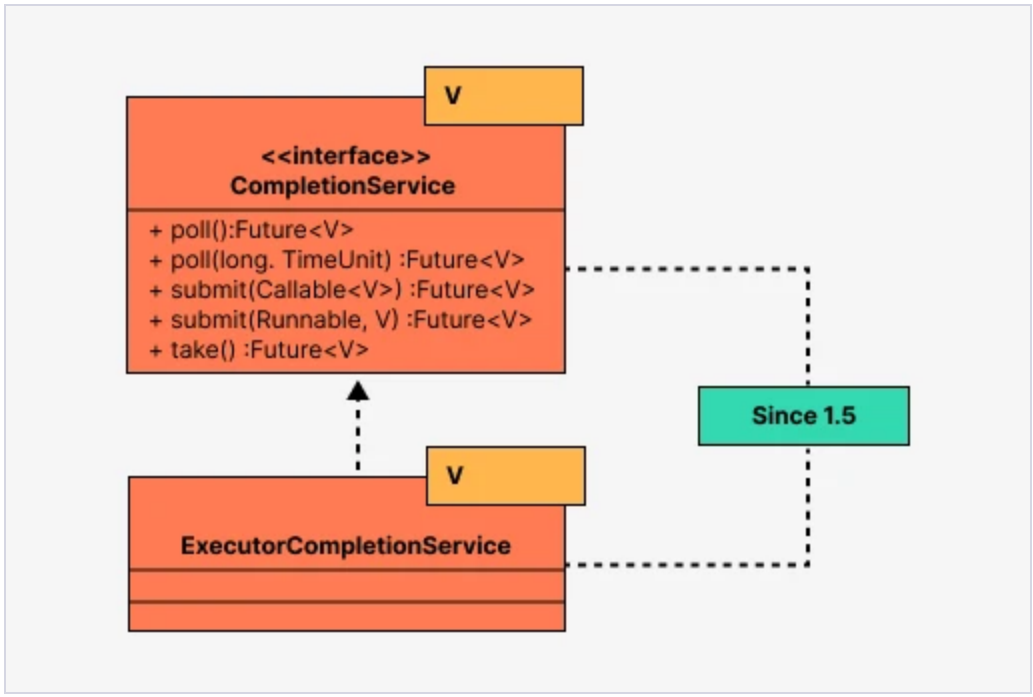
RejectedExecutionHandler — позволяет определить обработчик для задач, которые по каким-то причинам не могут быть выполнены через ***ThreadPoolExecutor***. Такое происходит, когда нет свободных потоков или сервис выключается или выключен (shutdown).

Несколько стандартных имплементаций находятся в классе `ThreadPoolExecutor`:

- CallerRunsPolicy — запускает задачу в вызывающем потоке;
- AbortPolicy — кидает эксепшен;
- DiscardPolicy — игнорирует задачу;

- DiscardOldestPolicy — удаляет самую старую незапущенную задачу из очереди, затем пытается добавить новую задачу еще раз.

Completion Service



CompletionService — интерфейс сервиса с развязкой запуска асинхронных задач и получением результатов. Для добавления задач есть метод `submit`, а для получения результатов уже завершенных задач используется блокирующий метод `take` и неблокирующий `poll`.

`ExecutorCompletionService` — является оберткой над любым классом, который реализует интерфейс ***Executor***, например, ***ThreadPoolExecutor*** или ***ForkJoinPool***. Используется, когда нужно абстрагироваться от способа запуска задач и контроля за их исполнением.

Если есть завершенные задачи, то вытаскиваем их. Если задач нет, то висим в take, пока что-нибудь не завершится. В основе сервиса используется LinkedBlockingQueue, но ты можешь передать любую реализацию BlockingQueue.

‹ Предыдущая лекция

Следующая лекция ›