L01-线程池-手写线程池

JAVA并发编程

学习目标

- 掌握线程的本质、概念、API
- 掌握多线程的使用
- 掌握线程池的原理
- 掌握Callable的使用，Future的原理

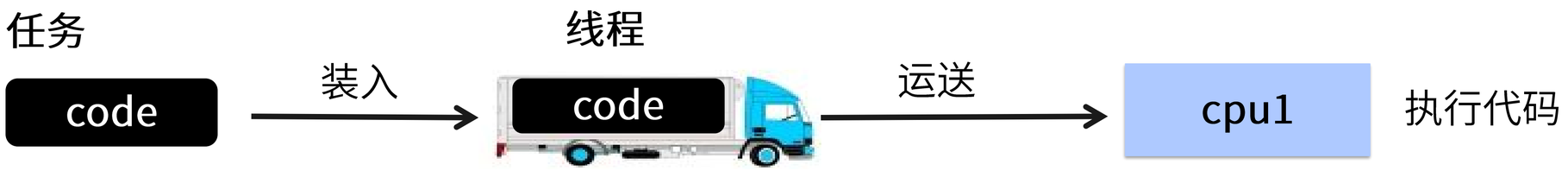
多线程面试-回顾基础

多线程面试-回顾基础

- **问题1、用多线程的目的是什么?**
  - 充分利用cpu资源，并发做多件事。
- **问题2、单核cpu机器上适不适合用多线程?**
  - 适合，如果是单线程，线程中需要等待IO时，此时CPU就空闲出来了。
- **问题3、线程什么时候会让出cpu?**
  - 阻塞时 wait await 等待 IO
  - Sleep
  - yield
  - 结束了

多线程面试-回顾基础

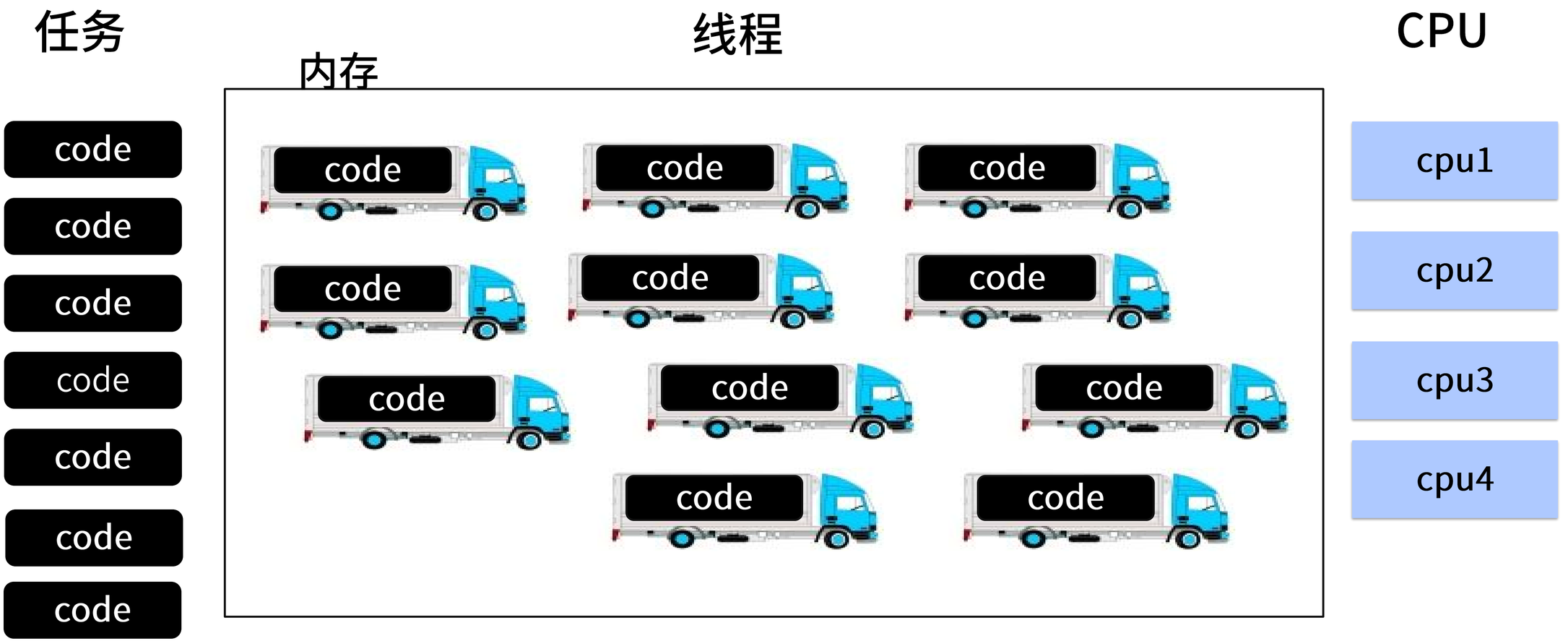
- 问题4、线程是什么?
  - 一条代码执行流，完成一组代码的执行。
  - 这一组代码，我们往往称为一个任务。
- 创建线程有几种方式?　　Runnable、Callable对象是线程吗?
- 问题5、cpu做的是什么工作?
  - 执行代码



多线程面试-多线程应用

- 问题6、线程是不是越多越好?

多线程面试-多线程应用



- 一定需要这么多卡车吗?
- 卡车循环运输，保证cpu不闲，可以吗?

多线程面试-多线程应用

问题6、线程是不是越多越好?

- 造卡车（线程）要不要时间？ 一次性使用，用完了得销毁，销毁要不要耗时间?

  - 1、线程在java中是一个对象，每一个java线程都需要一个操作系统线程支持。线程创建、销毁需要时间。如果 创建时间+ 销毁时间 > 执行任务时间 就很不合算。

- 造很多的卡车，得需要空间来放它们，会不会造成内存紧张?

  - 2、java对象占用堆内存，操作系统线程占用系统内存，根据jvm规范，一个线程默认最大栈大小1M，这个栈空间是需要从系统内存中分配的。

    线程过多，会消耗很多的内存。

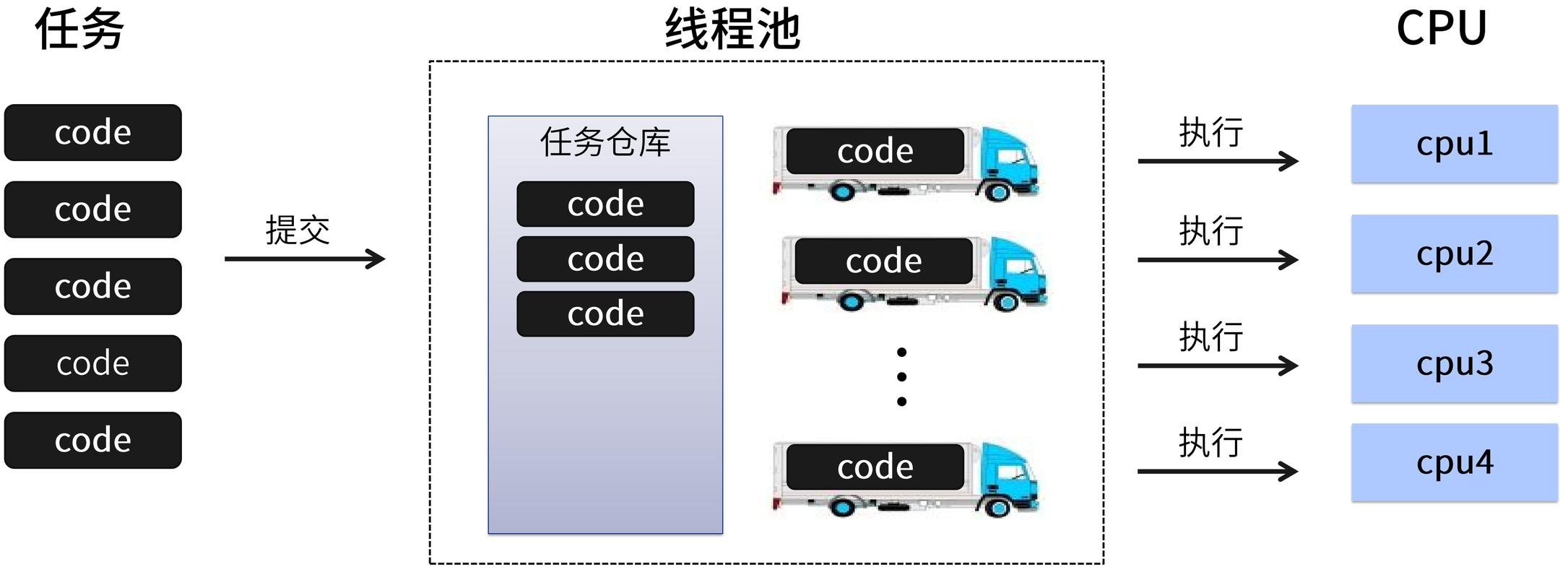
  - 3、操作系统需要频繁切换线程上下文（大家都想被运行），影响性能

多线程面试-多线程应用

- **问题7、该如何正确使用多线程?**
  - 多线程目的：充分利用cpu并发做事（多做事）
  - 线程的本质：将代码送给cpu执行
  - 用合适数量的卡车不断运送代码即可。
  - 这合适数量的线程就构成了一个池。
  - 有任务要执行，就放入池中，池中的一个线程将把任务运送到cpu执行。

线程池原理剖析

线程池原理剖析



线程池工作原理

- 接收任务，放入仓库
- 工作线程从仓库取任务，执行。
- 当没有任务时，线程阻塞，当有任务时唤醒线程执行

手写线程池-检验我们的编码能力

- 1、任务用什么表示？

  Runnable

  Callable

手写线程池-检验我们的编码能力

2、 仓库用什么?

- BlockingQueue 阻塞队列，线程安全的

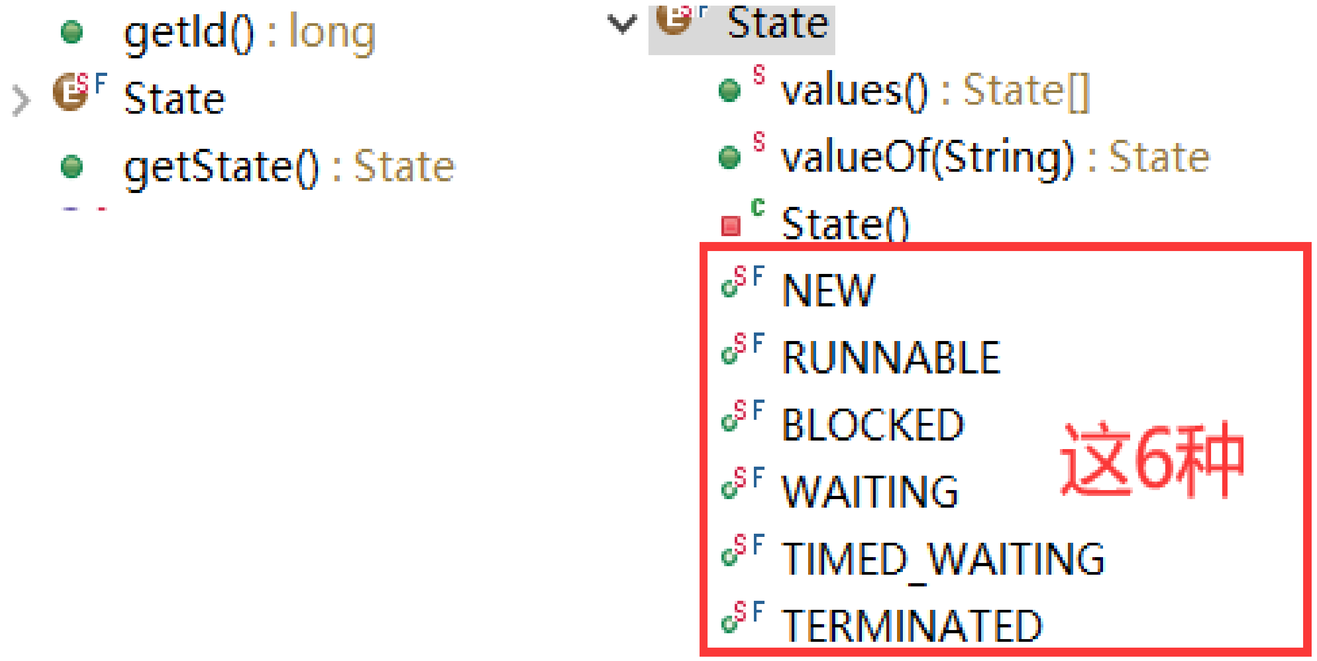
在队列为空时的获取阻塞，在队列满时的放入阻塞。
BlockingQueue 方法以四种形式出现，对于不能立即满足但可能在将来某一时刻可以满足的操作，这四种形式的处理方式不同：第一种是抛出一个异常，第二种是返回一个特殊值（null 或 false，具体取决于操作），第三种是在操作可以成功前，无限期地阻塞当前线程，第四种是在放弃前只在给定的最大时间限制内阻塞。下表中总结了这些方法：

	抛出异常	特殊值	阻塞	超时
插入	add(e)	offer(e)	put(e)	offer(e, time, unit)
移除	remove()	poll()	take()	poll(time, unit)
检查	element()	peek()	不可用	不可用

线程状态

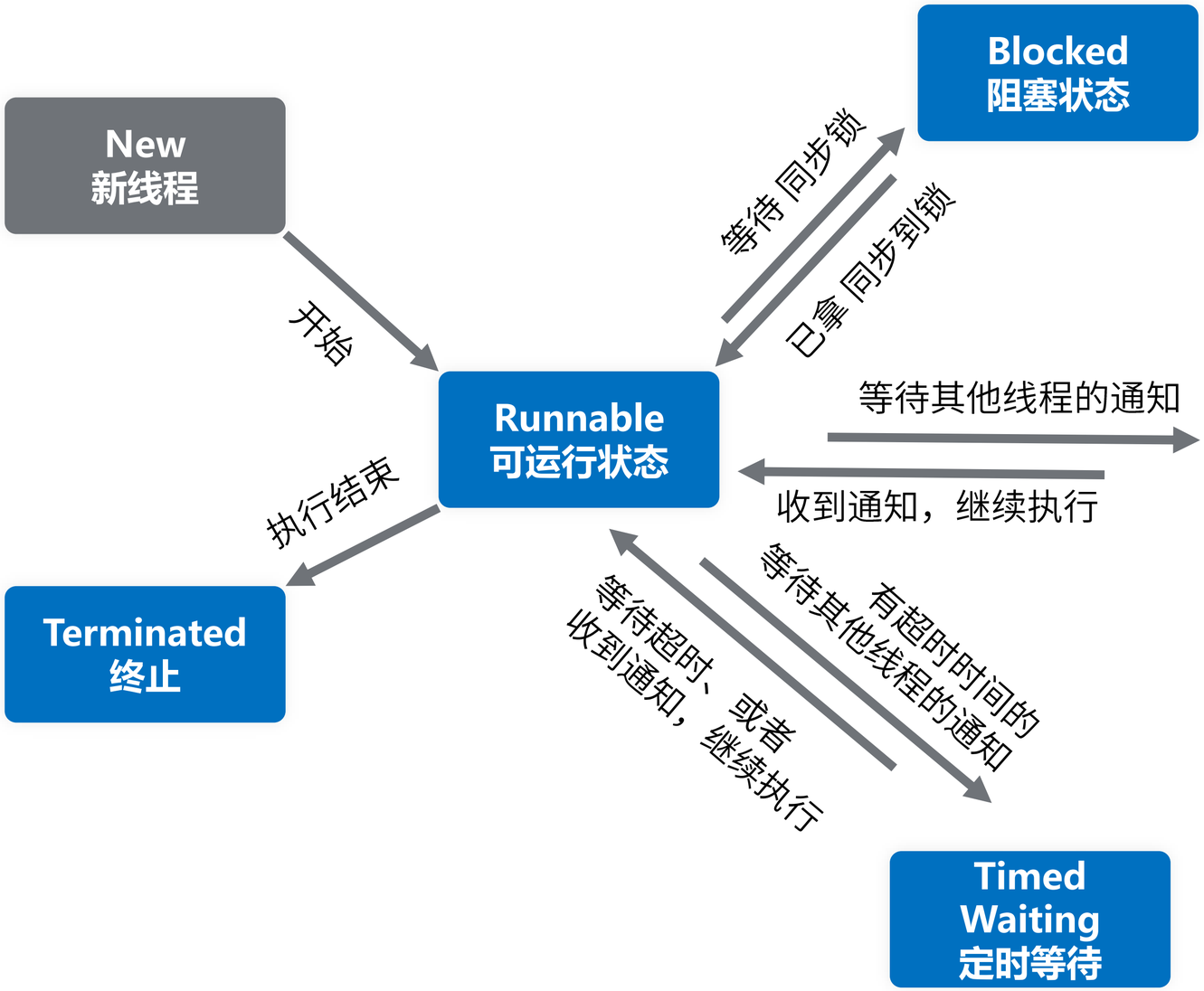
- 1 线程有几个状态？ 是哪几个？

线程到底有几个状态?



- 2 线程什么时候处于某状态?

线程6种状态的转换



- Sleep时处于什么状态?
- Join时处于什么状态?
- Wait时处于什么状态?
- 抢不到锁时处于什么状态?
- 等待IO时处于什么状态?

如何正确使用线程池

- **问题8、如何确定合适数量的线程?**
  - 如果是计算型任务?

    cpu数量的1-2倍
  - 如果是IO型任务?

    则需多一些线程，要根据具体的IO阻塞时长进行考量决定。

    如tomcat中默认的最大线程数为：200

    也可考虑根据需要在一个最小数量和最大数量间自动增减线程数

如何让线程终止

- 为什么不使用Thread.stop()?

  不安全

  立马释放所有持有的锁，会导致被保护资源不一致，使程序结果不确定

  官方说明：

  https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/doc-files/threadPrimitiveDeprecation.html

如何正确终止线程

1. 使用violate boolean变量来标识线程是否停止

2. 停止线程时，需要调用停止线程的interrupt()方法，因为线程有可能在wait()或sleep(), 提高停止线程的即时性

3. 对于blocking IO的处理，尽量使用InterruptibleChannel来代替blocking IO

如何正确终止线程-interrupt()

- interrupt()　InterruptedException　中断状态
- static booelean interrupted()　与　boolean isInterrupted()

Callable Future FutureTask

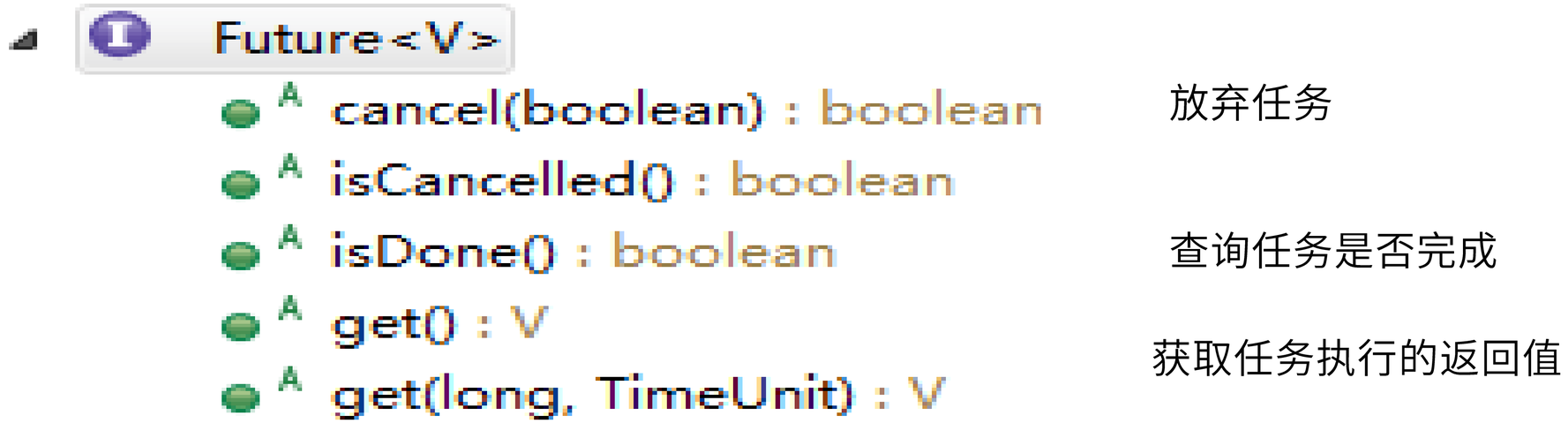
Callable

- Callable ： 对 Runnable的改进，可以返回值，可以抛出异常。

```
@FunctionalInterface
public interface Callable<V> {
    /**
     * Computes a result, or throws an exception if unable
     *
     * @return computed result
     * @throws Exception if unable to compute a result
     */
    V call() throws Exception;
}
```

Future

- Future 异步任务监视器，让提交者可以监控任务的执行了



- FutureTask 了解实现

FutureTask实现原理

- 思考：异步线程执行任务，怎样才能拿到它的任务返回值?
- FutureTask 是 Runnable，Callable是在Runnable中被调用执行
- Get方法解读：如何实现拿到返回值
- Cancel解读：如何实现放弃任务
- Callable执行异常怎么处理的

动手实践，下节课见