**Future模型**

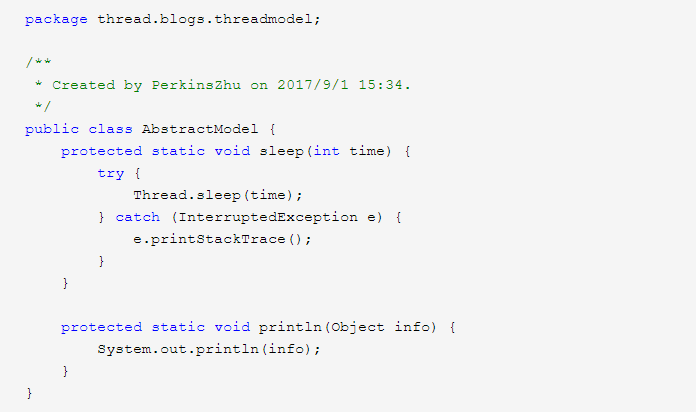
该模型通常在使用的时候需要结合Callable接口配合使用。Future：未来的、将来的，再结合Callable大概可以明白其功能。

　　Future是把结果放在将来获取，当前主线程并不急于获取处理结果。允许子线程先进行处理一段时间，处理结束之后就把结果保存下来，当主线程需要使用的时候再向子线程索取。

Callable是类似于Runnable的接口，其中call方法类似于run方法，所不同的是run方法不能抛出受检异常没有返回值，而call方法则可以抛出受检异常并可设置返回值。两者的方法体都是线程执行体。

无法抛出受检异常不等于无法捕获线程中throws的异常。run方法执行体中抛出异常是可以被捕获的，前提是使用Future来处理。

如果有一种场景需要一个线程处理一段业务，处理结束之后主线程将会使用处理结果进行后续处理。这样，按照普通逻辑，就需要使用到一个全局变量来保存子线程处理之后的结果。子线程处理结束之后，把结果保存在全局变量中供主线程进行调用。一旦涉及到全局能量便存在着多线程读写全局变量错误的风险。而使用Future模式便可以省去全局变量的使用，直接从线程中获取子线程处理结果。

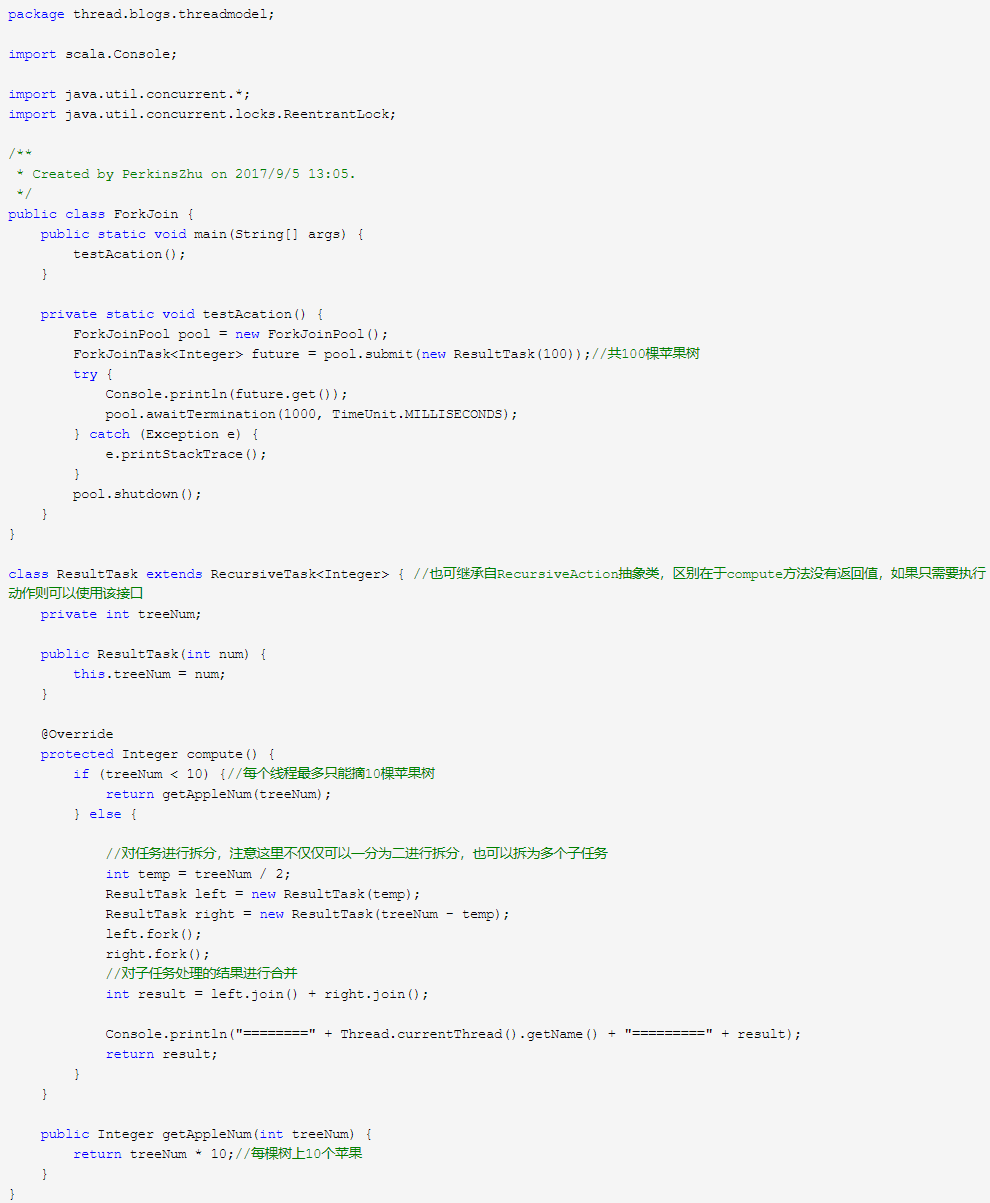




**fork&join 模型**

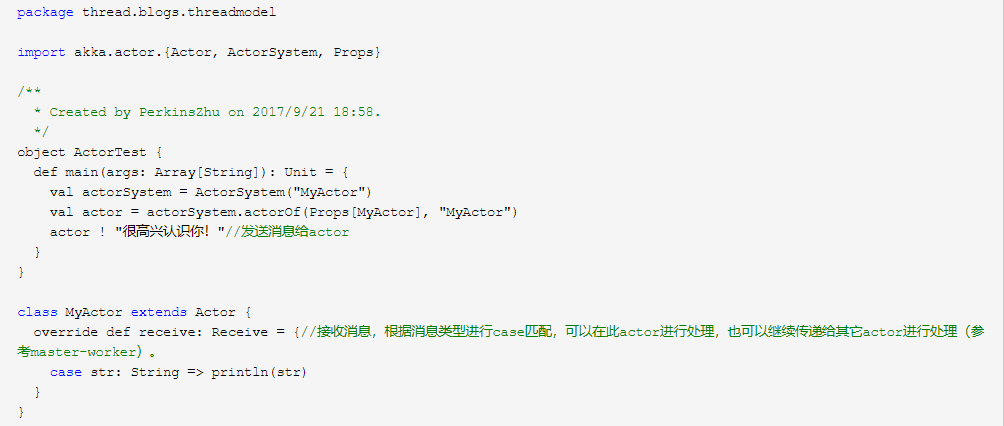
该模型是jdk中提供的线程模型。该模型包含递归思想和回溯思想，递归用来拆分任务，回溯用合并结果。 可以用来处理一些可以进行拆分的大任务。其主要是把一个大任务逐级拆分为多个子任务，然后分别在子线程中执行，当每个子线程执行结束之后逐级回溯，返回结果进行汇总合并，最终得出想要的结果。

在拆分子任务的时候并不是无限制开启线程，而是使用了线程池ForkJoinPool复用线程。



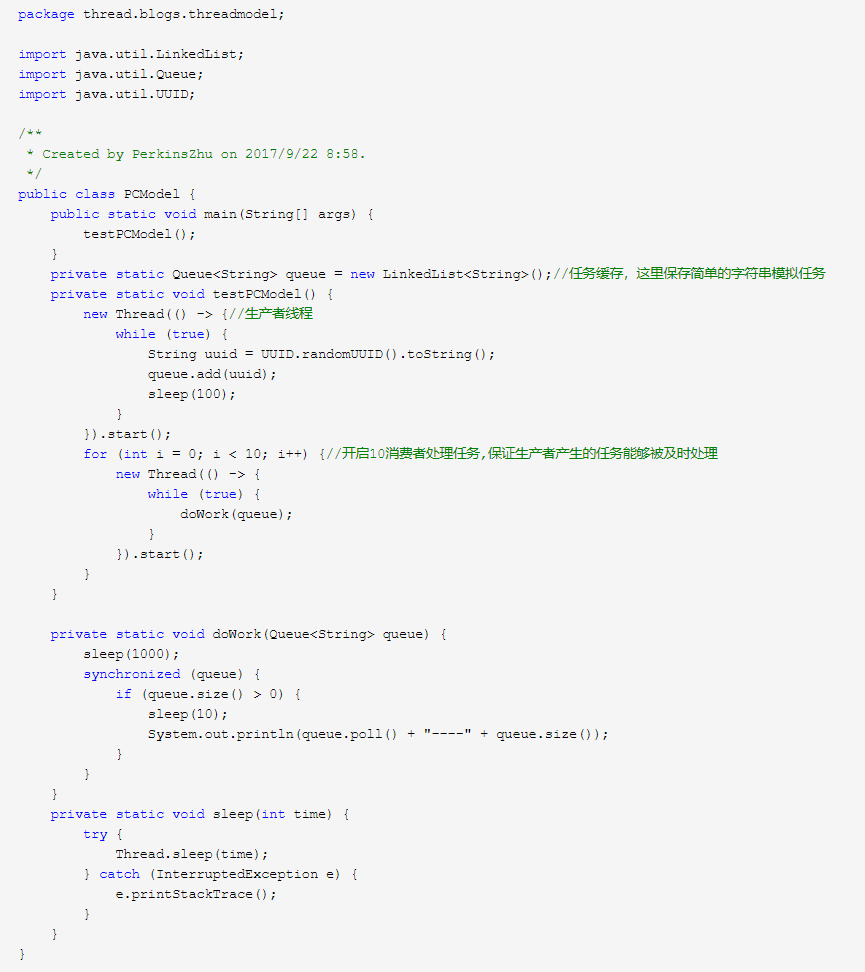
**actor消息模型**

actor模型属于一种基于消息传递机制并行任务处理思想，它以消息的形式来进行线程间数据传输，避免了全局变量的使用，进而避免了数据同步错误的隐患。actor在接受到消息之后可以自己进行处理，也可以继续传递（分发）给其它actor进行处理。在使用actor模型的时候需要使用第三方Akka提供的框架。



**生产者消费者模型**

生产者消费者模型都比较熟悉，其核心是使用一个缓存来保存任务。开启一个/多个线程来生产任务，然后再开启一个/多个来从缓存中取出任务进行处理。这样的好处是任务的生成和处理分隔开，生产者不需要处理任务，只负责向生成任务然后保存到缓存。而消费者只需要从缓存中取出任务进行处理。使用的时候可以根据任务的生成情况和处理情况开启不同的线程来处理。比如，生成的任务速度较快，那么就可以灵活的多开启几个消费者线程进行处理，这样就可以避免任务的处理响应缓慢的问题。



**master-worker模型**

master-worker模型类似于任务分发策略，开启一个master线程接收任务，然后在master中根据任务的具体情况进行分发给其它worker子线程，然后由子线程处理任务。如需返回结果，则worker处理结束之后把处理结果返回给master。下面的代码示例是使用akka actor for scala演示。使用的时候也可以使用java Thread来实现该模型。

