第五章 基础构建模块

#### 一．同步容器类

1. 早期类

Vector，HashTable。

实现方式：

将他们的状态封装起来，并对每个公有方法都进行同步，使得每次只有一个线程能访问容器的状态。

同步容器类的问题

1. 问题

同步容器类需要复合操作的时候会在多线程情况下差生问题。

2. 解决

客户端加锁

迭代器与ConcurrentModificationException

1. 及时失败的迭代器（fail-fast）

如果容器在迭代过程中被修改就会抛出一个ConcurrentModificationException异常

2. 解决方法

（1）对容器加锁

（2）复制容器

隐藏迭代器

1. 容器的hashCode和equals等方法也会间接的执行迭代操作，还有containsAll、removeAll、和retainAll等方法，以及把容器作为参数的构造函数，都会对容器进行迭代

2. 正如封装对象的状态有助于维持不变性条件一样，封装对象的同步机制同样有助于正确实施同步策略。

#### 二. 并发容器

1. 通过并发容器来代替同步容器，可以极大的提高伸缩性并降低风险。

2. CopyOnWriteArrayList、阻塞队列、ConcurrentHashMap、等

ConcurrentHashMap

1. 分段锁

2. 迭代器不会出错误

3. ConcurrentHashMap不能被加锁来执行独占访问，因此无法加锁来创建新的原子操作。

4. ConcurrentHashMap已经实现大部分原子操作。

CopyOnWriteArrayList

1. 用于替代同步List，在读多写少的情况下并发行更好。

2. 读取（包括迭代）的时候不需要加锁。

#### 三. 阻塞队列和生产者消费者模式

1. 优点

消除代码依赖性

把生产数据的过程和消费数据的过程解耦和

2. 常见的生产者和消费者

线程池与工作队列------>Executor

3. 分类

（1）有界队列

（2）无界队列

4. 在构建高可靠的应用程序时，有界队列是一种强大的资源管理工具：它们能抑制并防止产生过多的工作项，是应用程序在负荷过载的情况下变得更加健壮。

5. 常见API

LinkedBlockingQueue

ArrayBlockingQueue

二者类似于ArrayList于LinkedList但是拥有更好的并发性能

PriorityBlockingQueue

基于优先级的队列

SynchronousQueue

同步队列

6. 串行线程封闭

7. 双端队列与工作密取

Deque和BlockingDeque：

Deque是一个双端队列，实现了在队列头和队列尾的高效插入和移除。

工作密取：

在消费者-生产者模式中，多个消费者都有一个共享的工作队列。而在工作密取中，每个消费者都有各自的双端队列。如果一个消费者完成了自己双端队列的全部工作，那么它可以从其它消费者双端队列末尾秘密的获取工作。

优点：

具有更高的可伸缩性。

适用场景：

既是消费者也是生产者的问题。

#### 四. 阻塞方法与中断方法

1. 阻塞方法

当一个方法抛出InterruptedException时，表示该方法是一个阻塞方法，如果代码中调用了一个将抛出InterruptedException异常的方法时，你自己的方法也就变成了一个阻塞方法，如果这个方法被中断，那么它将努力提前结束阻塞状态。

2. 传递InterruptedException

避开这个异常通常时最明智的策略----只需要把InterruptedException传递给方法的调用者。例如，不捕获该异常，或者捕获然后在执行完简单的工作后再次抛出。

3. 恢复中断

有时不能抛出InterruptedException，例如当代码时Runnable的一部分时，必须捕获该异常，并通过当前线程上的interrupt方法恢复中断状态，这样在调用栈中更高层的代码将看到引发了一个中断。

#### 五. 同步工具类

1. 信号量（Semaphore）、栅栏（Barrier）、闭锁（Latch）

闭锁

1. 确保其他活动都完成后才执行。

2. CountDownLatch

有一个计数器被初始化为正数，表示需要等待的事件数量。countDown方法递减计数器，表示有一个事件发生了，而await方法等待计数器达到零，这表示所有需要等待的事件都已经发生。如果计数器非零，那么await会一直阻塞。

FutureTask

1. 表示一种抽象的可以生成结果的计算。

2. run()方法



（2）get()方法



信号量

1. 用来控制同时访问某个特定资源的操作数量，还可以用来实现某种资源池，或者对容器施加边界。

2. 原理

Semaphore管理着一组虚拟的许可，数量可以通过构造函数指定。执行操作的时候首先需要获得许可，并在使用以后释放许可。如果没有许可，那么acquire将一直阻塞知道有许可。Release方法将返回一个许可给信号量。

3. 用途

（1）实现资源池，例如数据库连接池。

（2）将任何容器（List，Set等）变成有界容器。

栅栏

1. 概念

等待其它线程全部到达栅栏位置才可以继续执行。

2. 与闭锁的区别

闭锁等待事件，而栅栏等待其它线程。

3. 分类

CyclicBarrier

Exchanger 交换数据，成对线程使用多个线程情况下，会随机交换，奇数个线程则有一个不能交换会一直等待。

#### 六. 构建高效且可伸缩的结果缓存

1. 重用之前的计算结果能降低延迟，提高吞吐量，但是会消耗更多内存。

2. 简单的缓存可能会将性能瓶颈转化为可伸缩性瓶颈。