```
27  p.start()
```

运行结果如下：

```
16:29:19 A 放入:[冷饮 1]
16:29:19 B 取出 [冷饮 1]
16:29:20 A 放入:[冷饮 2]
16:29:21 A 放入:[冷饮 3]
16:29:22 A 放入:[冷饮 4]
16:29:23 A 放入:[冷饮 5]
16:29:24 B 取出 [冷饮 2]
16:29:24 A 放入:[冷饮 6]
16:29:25 A 放入:[冷饮 7]
16:29:29 B 取出 [冷饮 3]
16:29:29 A 放入:[冷饮 8]
16:29:34 B 取出 [冷饮 4]
16:29:34 A 放入:[冷饮 9]
```

以上代码是实现生产者和消费者模型的一个比较简单的例子。在并发编程中，使用生产者和消费者模式能够解决绝大多数并发问题。如果生产者处理速度很快，而消费者处理速度很慢，那么生产者就必须等待消费者处理完，才能继续生产数据。同样的道理，如果消费者的处理能力大于生产者，那么消费者就必须等待生产者。为了解决这个问题，于是引入了生产者和消费者模式，如图 4.2 所示。

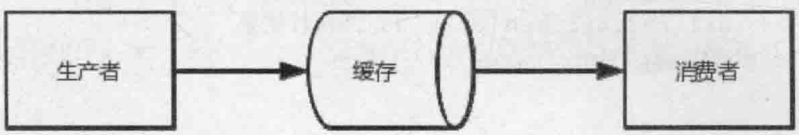

图 4.2　生产者和消费者模式

生产者和消费者模式是通过一个容器（队列）来解决生产者和消费者的强耦合问题。因为生产者和消费者彼此之间不直接通信，而是通过阻塞队列来进行通信，所以生产者生产完数据之后不用等待消费者处理，可直接扔给阻塞队列，消费者不找生产者要数据，而是直接从阻塞队列中取。阻塞队列就相当于一个缓冲区，平衡了生产者和消费者的处理能力。

4.8　多线程之线程池 pool

在面向对象编程中，创建和销毁对象是很费时间的，因为创建一个对象要获取内存资源或其他更多资源。虚拟机也将试图跟踪每一个对象，以便能够在对象销毁后进行垃圾回收。同样的道理，多任务情况下每次都会生成一个新线程，执行任务后资源再被回收就显得非常低效，