如果处理一个事件的方法被调用，那么相关的事务将只有一个事件，然后 Channel 处理器的 processEvent 方法将调用该事务，processEvent 方法是被 Source 调用来处理事件的。当第二个方法被调用，Channel 处理器的 processEventBatch 方法被 Source 调用，拦截器返回列表中的所有事件都写入一个单个事务中。请参见第 3 章“编写自定义 Source”一节理解 processEvent 方法和 rocessEventBatch 方法的不同之处。

例 6-2 展示了一个简单拦截器，并说明拦截器是如何工作的。Channel 处理器实例化 Builder 类，然后调用 Builder 对象的 configure 方法，该方法用于传递包含用于配置拦截器的配置参数的 Context 实例。然后，Channel 处理器调用 build 方法，该方法返回拦截器。Channel 处理器通过调用拦截器实例的 initialize 方法来初始化拦截器。通过构造方法传递配置参数给拦截器是一种非常好的方法，所以拦截器可以基于最终的配置完成所有的状态，正如 CounterInterceptor 类中实现的一样。

例6-2 简单拦截器

```
package usingflume.ch06;

public class CounterInterceptor implements Interceptor {
  private final String headerKey;
  private static final String CONF_HEADER_KEY = "header";
149  private static final String DEFAULT_HEADER = "count";
  private final AtomicLong currentCount;

  private CounterInterceptor(Context ctx) {
    headerKey = ctx.getString(CONF_HEADER_KEY, DEFAULT_HEADER);
    currentCount = new AtomicLong(0);
  }

  @Override
  public void initialize() {
    // No op
  }

  @Override
  public Event intercept(final Event event) {
    long count = currentCount.incrementAndGet();
    event.getHeaders().put(headerKey, String.valueOf(count));
    return event;
  }

  @Override
  public List<Event> intercept(final List<Event> events) {
    for (Event e : events) {
      intercept(
```