文件打开之后会立即调用 afterOpen 方法，这个方法可以用来写文件级别的 header 信息，例如序列化器中的高级标签，可以在 XML 中写数据。

每次 Sink 读取事件时，调用 write 方法。该方法主要负责转换 Flume 事件为所需的格式并将它写到输出流。Sink 一旦完成了整个批处理，就调用 flush 方法，该方法必须刷新序列化器的任何内部缓冲区的数据到流中。如果流是被序列化器的 BufferedOutputStream 包装，序列化器必须刷新缓冲流，这样所有传递到序列化器的数据才能刷新到输出流中。序列化器不需要刷新传入流，因为这通过 HDS Sink 本身已经完成。

HDFS Sink 关闭文件前，调用 beforeClose 方法。该方法可以用来编写任何文件的追踪（如关闭顶级报头）。如果 supportsReopen 方法返回 true，HDFS Sink 可以附加到文件。所以，只有当文件可以关闭然后为写操作可以再次打开，那么该方法必须返回 true。

例 5-2 展示的序列化器，序列化 Flume 事件到一个文件中，文件中包含使用例 3-8 展示过的 Protobuf 定义序列化的 Protobuf。在 Protobuf 序列化事件之后，每个事件被表示为一个整数，表示事件的长度。

例5-2 Protobuf事件序列化器

```
package usingflume.ch05;

public class ProtobufSerializer implements EventSerializer {
  private final boolean writeHeaderAndFooter;
  private final BufferedOutputStream stream;
  private static final byte[] footer = ("End Using Flume protobuf " +
    "file").getBytes();
  private static final byte[] header = ("Begin Using Flume protobuf" +
    " file").getBytes();


  private ProtobufSerializer(Context ctx, OutputStream stream) {
    writeHeaderAndFooter = ctx.getBoolean("writeHeaderAndFooter",
      false);
    this.stream = new BufferedOutputStream(stream);
  }

  @Override
111 public void afterCreate() throws IOException {
    if(writeHeaderAndFooter) {
      stream.write(header);
    }
  }

  @Override
```