```
agent.sources.spool.channels = memChannel

agent.sources.spool.spoolDir = /data/flume/spool
agent.sources.spool.batchSize = 250

agent.sources.spool.deletePolicy = immediate
agent.sources.spool.fileHeader = true
agent.sources.spool.fileHeaderKey = usingFlumeFiles
agent.sources.spool.deserializer = \
usingflume.ch03.ProtobufDeserializer$ProtobufDeserializerBuilder

agent.channels.memChannel.type = memory
agent.channels.memChannel.capacity = 10000
agent.channels.memChannel.transactionCapacity = 500
```

53

使用 Deserializers 读取自定义格式 *

Source 使用嵌入式的反序列化器转换目录中文件的数据，这允许 Source 以不同的方式从文件中读取数据到事件中。例如，一个“理解”Avro 的反序列化器能读取 Avro 格式的文件，并且转为每个 Avro 信息发送给 Flume 事件。否则，一次就要读取几行数据直到整个堆栈读取完并转换为 Flume 事件。文件中所有的数据一旦都读取完，Source 可以删除文件或者用新的扩展名重命名文件，这样相同的文件就不会被再次处理。

如果要使用自定义反序列化器，将 `deserializer` 参数的值设置为 `EventDeserializer$Builder` 的一个实现，`EventDeserializer$Builder` 可以构建 EventDeserializer 实现来使用。反序列化器可以通过传入参数 `deserializer.` 前缀进行配置。基于文本的反序列化器可以调用 `readChar` 方法读取字符。不同字符集的字符表示的方式不同。要告知 Source 用什么字符集，则设置 character 参数的值为输入字符集的名称，默认情况下是 UTF-8。

反序列化器实现 `EventDeserializer` 接口，还应该提供一个 `Builder` 类，它必须实现 `EventDeserializer$Builder` 接口。Builder 必须有一个无参数的公共构造函数，Flume 框架可以使用它来实例化构造器。Builder 的 `build` 方法必须创建并返回一个完全反序列化器的配置实例。

Context 实例和 `ResettableInputStream` 实例被传递给该方法。可以使用 Context 实例配置反序列化器。反序列化器将从输入流反序列化事件。`ResettableInputStream` 是一个从流读取数据的接口，而且还可以回滚到流以前的位置。`ResettableInputStream` 类的实例，调用 `reset` 方法时，不管上次 `mark` 方法调用之后，流使用 `read` 或 `readChar` 方法读取了多少字节的数据，能保证从上次 `mark` 方法调用发生时候的流的位置，重置从流