例 3-1 展示了一个实例，即为名为 avro 的 Source 配置两个拦截器。配置两个拦截器，*host* 和 *static*（分别命名为 i1 和 i2），是用来拦截 Source 收到的事件。如上所述，拦截器可以接受 interceptors.<interceptor_name>. 前缀的配置。

例3-1 配置拦截器

```
agent.sources.avro.interceptors = i1 i2
agent.sources.avro.interceptors.i1.type = host
agent.sources.avro.interceptors.i1.preserveExsiting = true
agent.sources.avro.interceptors.i2.type = static
agent.sources.avro.interceptors.i2.key = header
agent.sources.avro.interceptors.i2.value = staticValue
```

每个 Source 恰好有一个 Channel 选择器（这就是为什么它不是一个指定的组件，并且可以使用 selector 配置后缀进行配置）。尽管 Channel 选择器的配置看起来像 Source 的子组件的配置，但是 Source 不需要配置选择器——这是由配置系统完成的。在例 3-2 中，名为 avro 的 Source 的 Channel 处理器配置了一个多路复用 Channel 选择器，用来分流 Source 事件。我们将在第 6 章中讨论多路复用 Channel 选择器的具体配置参数，但是正如这个例子的示范，选择器可以接受配置参数，它们会返回给 Source 写入具体事件的 Channel。

例3-2 Channel选择器配置

```
agent.sources.avro.selector.type = multiplexing
agent.sources.avro.selector.header = priority
agent.sources.avro.selector.mapping.1 = channel1
agent.sources.avro.selector.mapping.2 = channel2
agent.sources.avro.selector.default = channel2
```

如果一个 Agent 被重新配置，相同的 Source 类的实例将不能被重用。因此，所有 Flume 自带的 Source 都是无状态的。我们也期望任何自定义的被嵌入到 Flume 的 Source 是无状态的，以避免数据丢失。

我们已经了解了 Source 的基础概念，现在我们来讨论 Flume 封装的各种 Source。

Sink-to-Source 通信

36

Flume 最重要的特性之一就是 Flume 部署水平扩展的简单性。可以很容易完成扩展的原因是，很容易为 Flume 调度添加新的 Agent，也很容易配置新的这些 Agent 发送数据给其他 Flume Agent。类似地，一旦添加了新的 Agent，仅仅通过更新配置文件，就能很简单地配置已经运行的 Agent 来写入这个新的 Agent。

Flume 灵活性的重点就是 Flume 的 RPC sink-source 的结合。在第 2 章的“Flume Agent