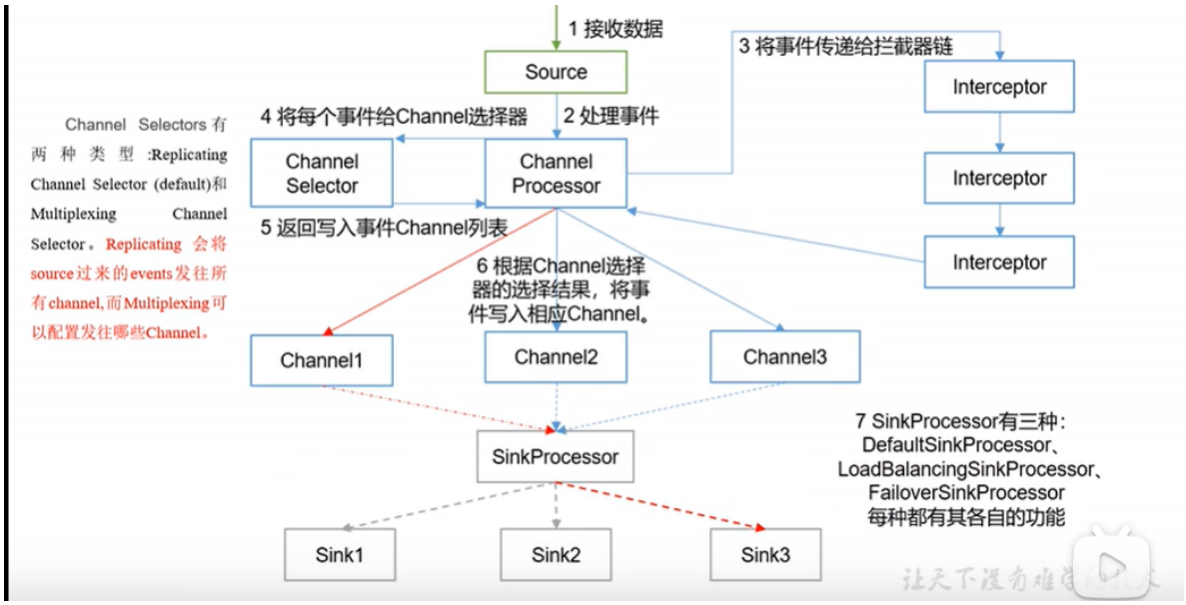
Flume8 Agent内部原理



拦截器

拦截器是简单的插件式组件，设置在source和channel之间。source接收到的时间，在写入channel之前，拦截器都可以进行转换或者删除这些事件。每个拦截器只处理同一个source接收到的事件。可以自定义拦截器。

flume内置了很多拦截器，并且会定期的添加一些拦截器，在这里列出一些flume内置的，经常使用的拦截器。

Timestamp Interceptor(时间戳拦截器)

flume中一个最经常使用的拦截器，该拦截器的作用是将时间戳插入到flume的事件报头中。如果不使用任何拦截器，flume接受到的只有message。时间戳拦截器的配置。 参数 默认值 描述 type 类型名称timestamp，也可以使用类名的全路径 preserveExisting false 如果设置为true，若事件中报头已经存在，不会替换时间戳报头的值

source连接到时间戳拦截器的配置：

```
a1.sources.r1.interceptors = timestamp
a1.sources.r1.interceptors.timestamp.type=timestamp
a1.sources.r1.interceptors.timestamp.preserveExisting=false
```

Host Interceptor(主机拦截器)

主机拦截器插入服务器的ip地址或者主机名，agent将这些内容插入到事件的报头中。时间报头中的key使用hostHeader配置，默认是host。主机拦截器的配置参数 默认值 描述 type 类型名称host hostHeader host 事件投的key useIP true 如果设置为false，host键插入主机名 preserveExisting false 如果设置为true，若事件中报头已经存在，不会替换host报头的值

source连接到主机拦截器的配置：

```
a1.sources.r1.interceptors = host
a1.sources.r1.interceptors.host.type=host
a1.sources.r1.interceptors.host.useIP=false
a1.sources.r1.interceptors.timestamp.preserveExisting=true
```

静态拦截器(Static Interceptor)

静态拦截器的作用是将k/v插入到事件的报头中。配置如下参数 默认值 描述 type 类型名称static key key 事件头的key value value key对应的value值 preserveExisting true 如果设置为true，若事件中报头已经存在该key，不会替换value的值source连接到静态拦截器的配置：

```
a1.sources.r1.interceptors = static
a1.sources.r1.interceptors.static.type=static
a1.sources.r1.interceptors.static.key=logs
a1.sources.r1.interceptors.static.value=logFlume
a1.sources.r1.interceptors.static.preserveExisting=false
```

正则过滤拦截器(Regex Filtering Interceptor)

在日志采集的时候，可能有一些数据是我们不需要的，这样添加过滤拦截器，可以过滤掉不需要的日志，也可以根据需要收集满足正则条件的日志。参数默认值描述 type 类型名称REGEX_FILTER regex .* 匹配除"\n"之外的任何个字符 excludeEvents false 默认收集匹配到的事件。如果为true，则会删除匹配到的event，收集未匹配到的。

source连接到正则过滤拦截器的配置：

```
a1.sources.r1.interceptors = regex
a1.sources.r1.interceptors.regex.type=REGEX_FILTER
a1.sources.r1.interceptors.regex.regex=(rm)|(kill)
a1.sources.r1.interceptors.regex.excludeEvents=false
```

这样配置的拦截器就只会接收日志消息中带有rm 或者kill的日志。

Channel Selectors

用于选择不同的Channel。

Replicating Channel Selector (default)

```
Replicating 会将source过来的events发往所有channel。
```

例如：这里有 channel1 和 channel2 ,当 even1 和 even2出现的时候，even1和even2会同时发往channel1 和channel2。

相关配置：

```
a1.sources = r1
a1.channels = c1 c2 c3
a1.sources.r1.selector.type = replicating  #可以不写
a1.sources.r1.channels = c1 c2 c3
a1.sources.r1.selector.optional = c3
```

Multiplexing Channel Selector

```
Multiplexing 可以选择该发往哪些channel
```

例如：这里有 channel1 和 channel2 ,当 even1 和 even2出现的时候，even1和even2可以选择发往channel1 和channel2。

相关配置：

```
a1.sources = r1
a1.channels = c1 c2 c3 c4
a1.sources.r1.selector.type = multiplexing
a1.sources.r1.selector.header = state
a1.sources.r1.selector.mapping.CZ = c1
a1.sources.r1.selector.mapping.US = c2 c3
a1.sources.r1.selector.default = c4
```

SinkProcessor sink组

在channel和sink之间，将channel的even根据不同的类型的sinkprocessor发送到sink中去。

DefaultSinkProcessor

只接收一个sink组

LoadBalancingSinkProcessor

负载均衡sink组

例子如下:

```
a1.sinkgroups = g1
a1.sinkgroups.g1.sinks = k1 k2
a1.sinkgroups.g1.processor.type = load_balance
a1.sinkgroups.g1.processor.backoff = true
a1.sinkgroups.g1.processor.selector = random
```

FailoverSinkProcessor

故障转移sink组

例子:

```
a1.sinkgroups = g1
a1.sinkgroups.g1.sinks = k1 k2
a1.sinkgroups.g1.processor.type = failover
a1.sinkgroups.g1.processor.priority.k1 = 5
a1.sinkgroups.g1.processor.priority.k1 = 10
a1.sinkgroups.g1.processor.maxpenalty = 10000
```