UUID 拦截器生成的 UUID

UUID 拦截器生成版本 4 的 UUID，实际上是伪随机方式。对于那些需要很强的 UUID 唯一性保证的应用，最好编写一个自定义拦截器来保证强唯一性。

编写拦截器 *

拦截器是 Flume 中最容易编写的组件。编写拦截器的时候，实现者只需要写一个实现 `Interceptor` 接口的类。该接口本身非常简单，虽然 Flume 要求所有的拦截器必须有一个实现了 `Interceptor$Builder` 接口的 `Builder` 类。所有的 Builder 类必须有一个公共无参的构造方法，Flume 使用该方法来进行实例化。可以使用传递到 `Builder` 类的 `Context` 实例配置拦截器。所有需要的参数都要传递到 `Context` 实例。

拦截器一般用于分析事件以及在需要的时候丢弃事件。通常情况是，拦截器给事件插入事件报头，这些事件后续用于 HDFS Sink（用于时间戳或者用于基于报头的分桶）、HBase Sink（用于行键）等。这些事件报头也经常在复杂 Channel 处理器中用于将流分为多个流的分支，或者基于优先级将事件发送到不同的 Sink 中，这些事件报头是拦截器分析的内容。这种进行正则匹配和检测优先级（基于类似日志级别这样优先级）的字符串处理的方式，可以从创建数据的应用中卸载下来。

例 6-1 展示了所有拦截器必须实现的 Interceptor 接口。

148

例6-1 Interceptor接口

```
package org.apache.flume.interceptor;
public interface Interceptor {
  public void initialize();
  public Event intercept(Event event);
  public List<Event> intercept(List<Event> events);
  public void close();
  /** Builder implementations MUST have a no-arg constructor */
  public interface Builder extends Configurable {
    public Interceptor build();
  }
}
```

当实现拦截器的时候，有两种处理事件的方法，都命名为 `intercept`，它们具有不同的参数和返回类型。第一种方法只接受一个事件并返回一个事件（或者 `null`），第二种方法可接受一个事件列表并返回一个事件列表。这两种情况下，都包含伴随 Channel 的一个事务。这两种方法必须是线程安全的，因为如果 Source 运行在多线程的情况下，这些方法可能被多个线程调用。