



C o m m u n i t y   E x p e r i e n c e   D i s t i l l e d

# Apache Flume: Distributed Log Collection for Hadoop

Stream data to Hadoop using Apache Flume

Steve Hoffman

**[PACKT]** open source\*  
PUBLISHING community experience distilled

# Apache Flume: Distributed Log Collection for Hadoop

Stream data to Hadoop using Apache Flume

**Steve Hoffman**



BIRMINGHAM - MUMBAI

# Apache Flume: Distributed Log Collection for Hadoop

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: July 2013

Production Reference: 1090713

Published by Packt Publishing Ltd.  
Livery Place  
35 Livery Street  
Birmingham B3 2PB, UK.

ISBN 978-1-78216-791-4

[www.packtpub.com](http://www.packtpub.com)

Cover Image by Abhishek Pandey ([abhishek.pandey1210@gmail.com](mailto:abhishek.pandey1210@gmail.com))

# Credits

**Author**

Steve Hoffman

**Project Coordinator**

Sherin Padayatty

**Reviewers**

Subash D'Souza

Stefan Will

**Proofreader**

Aaron Nash

**Acquisition Editor**

Kunal Parikh

**Indexer**

Monica Ajmera Mehta

**Commissioning Editor**

Sharvari Tawde

**Graphics**

Valentina D'silva

Abhinash Sahu

**Technical Editors**

Jalasha D'costa

Mausam Kothari

**Production Coordinator**

Kirtee Shingan

**Cover Work**

Kirtee Shingan

# About the Author

**Steve Hoffman** has 30 years of software development experience and holds a B.S. in computer engineering from the University of Illinois Urbana-Champaign and a M.S. in computer science from the DePaul University. He is currently a Principal Engineer at Orbitz Worldwide.

More information on Steve can be found at <http://bit.ly/bacoboy> or on Twitter @bacoboy.

This is Steve's first book.

---

I'd like to dedicate this book to my loving wife Tracy. Her dedication to perusing what you love is unmatched and it inspires me to follow her excellent lead in all things.

I'd also like to thank Packt Publishing for the opportunity to write this book and my reviewers and editors for their hard work in making it a reality.

Finally, I want to wish a fond farewell to my brother Richard who passed away recently. No book has enough pages to describe in detail just how much we will all miss him. Good travels brother.

---

# About the Reviewers

**Subash D'Souza** is a professional software developer with strong expertise in crunching big data using Hadoop/HBase with Hive/Pig. He has worked with Perl/PHP/Python, primarily for coding and MySQL/Oracle as the backend, for several years prior to moving into Hadoop fulltime. He has worked on scaling for load, code development, and optimization for speed. He also has experience optimizing SQL queries for database interactions. His specialties include Hadoop, HBase, Hive, Pig, Sqoop, Flume, Oozie, Scaling, Web Data Mining, PHP, Perl, Python, Oracle, SQL Server, and MySQL Replication/Clustering.

---

I would like to thank my wife, Theresa for her kind words of support and encouragement.

---

**Stefan Will** is a computer scientist with a degree in machine learning and pattern recognition from the University of Bonn. For over a decade has worked for several startup companies in Silicon Valley and Raleigh, North Carolina, in the area of search and analytics. Presently, he leads the development of the search backend and the Hadoop-based product analytics platform at Zendesk, the customer service software provider.



# www.PacktPub.com

## Support files, eBooks, discount offers and more

You might want to visit [www.PacktPub.com](http://www.PacktPub.com) for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

## Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

## Free Access for Packt account holders

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

# 目录

---

前言 .....	11
这本书包含那些内容 .....	2
读这本书你需要准备些什么 .....	2
这本书是为谁准备的 .....	3
本书约定 .....	3
读者反馈 .....	4
用户支持 .....	4
勘误表 .....	4
概览 .....	8
<b>Flume 0.9</b> .....	8
<b>Flume 1.X (Flume-NG)</b> .....	8
HDFS 和流式数据之间的问题 .....	9
<b>Sources, channels 和 sinks</b> .....	10
<b>Flume events</b> .....	11
小结 .....	14
<b>Flume 快速入门</b> .....	15
下载 <b>Flume</b> .....	15
<b>Flume</b> 配置文件概览 .....	17
以 "Hello World" 开始 .....	18
小结 .....	23
<b>Channels</b> .....	25
内存 <b>channel</b> .....	26
文件 <b>channel</b> .....	27
小结 .....	31
<b>Sinks 和 Sink 处理器</b> .....	32



HDFS sink .....	32
数据压缩.....	38
Event 序列化.....	39
Sink 组.....	43
小结.....	45
Sources 和 Channel 选择器 .....	47
使用 tail 时的一些问题 .....	47
Exec source .....	49
Spooling directory source .....	51
Syslog sources .....	53
Channel 选择器 .....	58
小结.....	59
拦截器、ETL 和 Routing.....	61
拦截器 .....	61
分层数据流 .....	69
Routing .....	75
小结.....	76
Flume 监控 .....	77
监控 agent 进程 .....	77
监控性能指标 .....	78
小结.....	84
There Is No Spoon – 即时分布式数据收集系统的现实 .....	86
传输时间对比记录时间 .....	86
罪恶的时间戳.....	86
容量规划.....	86
多个数据中心时的注意事项 .....	87
合规和数据抹除 .....	88
小结.....	88
译者注 .....	89
关于这本书 .....	89
关于译者.....	89
欢迎加入 Flume 大家庭.....	89
索引 .....	90

---



# 前言

Hadoop 是一个伟大的开源工具，通过 Hadoop 过滤大量的非结构化数据形成可用的数据，从而使你的公司可以更好地洞察客户需求。它是廉价的（绝大多数的免费）、可水平扩展的只要你的数据中心有空间和电力，并且可以处理那些可以将你的传统数据仓库压垮的问题。据说一个鲜为人知的秘密就是您的 Hadoop 集群需要你给它数据来工作，否则，你只会拥有一个非常昂贵的高温发生器。你很快会发现，一旦你过了 Hadoop 的“玩耍”阶段，你将需要一个工具来自动的往你的集群中填充数据。在过去，你必须为这个问题想一个解决方案，但也仅此而已！Flume 开始是作为 Cloudera 的一个项目当他们的集成工程师必须不断地反复书写工具为客户自动导入数据。今天这个项目托管在 Apache 基金会项目，目前正在积极开发，拥有很多多年来一直在生产环境中使用它的用户。

在这本书中我希望通过概览 Flume 的架构和快速启动指南让你能够快速启动和运行 Flume。在这之后我们将深入到许多 Flume 有用的组件细节中去，包括非常重要的持久化存储流动中的数据记录的 File Channel、缓冲数据和写数据到 HDFS（Hadoop 分布式文件存储系统）中的 HDFS Sink。因为 Flume 有各种各样的模块，那么启动 Flume 你唯一需要的工具是用来编写配置文件的文本编辑器。

在这本书的最后，你应该对构建一个高可用性、高容错、流式数据通道有足够的了解，来为你的 Hadoop 集群提供数据。

## 这本书包含那些内容

第一章，*概览*，将 Flume 介绍给读者以及它试图解决的一些问题（特别对于 Hadoop 的）。以及对于在后面章节中会讲解到的各种组件的一些概述。

第二章，*Flume 快速入门*，本章节是为了让你能够快速启动和运行 Flume，包括下周 Flume、创建一个“Hello World”形式的配置文件，并运行它。

第三章，*Channels*，本章节涉及绝大多数人会使用到的两种主要的 channel，以及它们每个可用的配置选项。

第四章，*Sinks 和 Sink 处理器*，本章节将深入讲解使用 HDFS Flume 作为输出，包含压缩选项以及数据格式。故障转移选项同样涉及到从而创建一个更加健壮的数据传输通道。

第五章，*Sources 和 Channel 选择器*，本章节将介绍集中 Flume 输入机制以及他们的配置选项。根据数据内容来在各种 channel 中进行选择同样涉及到，而从能够创建复杂的数据流。

第六章，*拦截器、ETL 和 Routing*，本章介绍如何对飞转的数据进行格式转换以及从有效负载中提取信息来配合 channel 选择器进行路由决策的功能。本章还介绍了多层次的 Flume agent，以及使用 Flume 命令行作为一个独立的 Avro client 来测试和手动导入数据。

第七章，*Flume 监控*，讨论各种可以用来监控 Flume 的方案，不管是自带的还是外部的解决方案如 Monit，Nagios、Ganglia 和自定义的 hooks。

第八章，*There Is No Spoon – 即时分布式数据收集系统的现实*，包含了在上面介绍的如何配置和使用 Flume 之外的各种需要考虑的杂七杂八的事情。

## 读这本书你需要准备些什么

你需要一个安装了 Java 虚拟机的电脑，因为 Flume 是用 Java 程序编写的。如果你的电脑上还没有 Java 环境，你可以从这里下载：<http://java.com/>

你同样需要连接因特网，这样你就可以下载 Flume 来运行快速启动示例了。

这本数设计的 Flume 版本是 Apache Flume 1.3.0，包括少数后来移植到 Cloudera 的 Flume CDH4 版本中的项目。

## 这本书是为谁准备的

这本书是为了那些负责实现将来自不同系统的数据自动传输到 Hadoop 集群的人准备的。如果你的工作是定期将数据加载到 Hadoop 中，这本书将帮助你脱离手动的 monkey-work 或编写一个由你来做支持的自定义的工具，只要你在你的公司工作。

只需要基本的 Hadoop 的 HDFS 知识。一些自定义实现在你需要的时候实现它。对于这种程度的实现，您需要知道如何用 Java 进行编程。

最后，你需要您最喜欢的文本编辑器，因为这本书绝大多数都是介绍如何配置各种水槽组件通过 agent 的文本配置文件。

译者注：monkey-work，这里可不是“程序猿”的意思，Google 的解释是：We could hire a trained monkey to do your job，翻译过来就是“我们可以雇佣一只受过训练的猴子来做你的工作”

## 本书约定

在这本书中，你将会发现好几种文本格式来区分不同的信息。这里是一些文本格式的示例，以及这些文本格式所表示的含义。

代码片段在文本中以如下格式显示：“We can include other contexts through the use of the include directive.”

块状的代码以如下格式显示：

```
agent.sinks.k1.hdfs.path=/logs/apache/access
agent.sinks.k1.hdfs.filePrefix=access
agent.sinks.k1.hdfs.fileSuffix=.log
```


当我想让你的注意力集中到某部分代码块时，相关的行将以**粗体**进行显示：


```
agent.sources.s1.command=uptime
agent.sources.s1.restart=true
agent.sources.s1.restartThrottle=60000
```

一些命令行输入或者输出以如下格式显示（以“\$”开头）：

```
$ tar -zxf apache-flume-1.3.1.tar.gz
$ cd apache-flume-1.3.1
```

新的术语和重要词语将以粗体显示。

[  警告和重要提醒将放在这样的片段中。 ]

[  技巧和窍门将放在这样的片段中。 ]

## 读者反馈

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book title via the subject of your message. If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on [www.packtpub.com/authors](http://www.packtpub.com/authors).

## 用户支持

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## 勘误表

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from





## 盗版声明

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

## 问题

You can contact us at [questions@packtpub.com](mailto:questions@packtpub.com) if you are having a problem with any aspect of the book, and we will do our best to address it.



# 1 概览

如果你正在阅读这本书，你将有机会游弋在大数据的世界中。多亏了 Facebook、Twitter、亚马逊、数码相机和拍照手机、YouTube、谷歌和其他任何你能想到的连接在互联网上的东西，创建大数据变得非常容易。作为一个网站的内容提供商，10 年前，您的应用程序日志只是用来帮助你排除你网站的故障。今天，如果你知道如何从大量数据中筛选出有用的内容，同样的数据可以为你提供宝贵的数据来分析商务和客户。

此外，因为你正在阅读这本书，你也意识到 Hadoop 创造出来是用来解决(部分)处理大数据问题的。当然，这只工作在你能够正确加载数据到你的 Hadoop 集群中并对你的数据进行科学的筛选。

通过 Hadoop（在这个例子中是 Hadoop 文件系统（HDFS））放入数据和提取数据很简单——就是下面这个简单的命令：

```
% hadoop fs --put data.csv .
```

当你的数据都整齐的打包好并且准备上传时，这种方式很有效。

但你的网站总在不停的产生数据。你加载数据到 HDFS 的频率应该怎么定？一天？一小时？无论你选择何种处理频率，最终有人总是问：“你能让我更早的拿到数据吗？”你真正需要的是一个解决方案，它可以处理流式的日志。

在这个需求上看来你并不孤单。Cloudera, Hadoop 的专业服务提供商以及他们自己的 Hadoop 发行版本，在与他们的客户合作中遇到了这种需求。Flume 是为了满足这种需求而被创造出来的，并且建立了一个标准、简单、健壮、灵活、可扩展的机制用来摄取数据到 Hadoop 中。

## Flume 0.9

Flume 在 Cloudera 2011 年发布的 CDH3 中首次被引入。它由一组协同工作的守护进程（agents）组成，agents 的配置从由 zookeeper 管理的（联合配置和协调系统）中央 master（或多个 master）获取。通过 master，你可以在 Web 界面中检查 agent 的状态，以及集中的推送配置从 UI 或通过 shell 命令行（两者实际上都是通过 zookeeper 和 agent 进行通信）。

数据可以通过下面三种方式中的一种来发送：**best effort (BE)**、**disk failover (DFO)** 和 **end-to-end (E2E)**。Master 被用作 E2E 模式的确认，由于多 master 配置还未真正成熟，所以对于 E2E 数据流模式通常你只有一个 master 来让它作为失败的中心处理点。Best effort 模式正如它听起来的那样——agent 会尝试和发送数据，但如果没有成功，数据将会被丢弃。这种模式对于可以容许误差的度量计算来说非常好，因为数据只是排在第二位。Disk failover 模式存储无法发送的数据到本地硬盘中（有时候是本地的数据库），然后不停的重试直到数据能够被发送到数据流的下一个容器中。这对于那些意料之中（或者意料之外的）的中断运行很适用，只要你有足够的本地存储空间来缓存数据。

在 2011 年 6 月，Cloudera 将 Flume 项目控制权移交给了 Apache 基金会。一年后它以孵化项目的形式出现。在孵化的时间里，重构 Flume 的工作开始展开，即 Flume-NG（Flume 的下一代）。

## Flume 1.X (Flume-NG)

Flume 重构有很多的原因。如果你对细节有兴趣，你可以访问来进一步了解：<https://issues.apache.org/jira/browse/FLUME-728>。这个版本作为重构的分支，最终发展为 Flume 1.X 开发的主分支。

Flume 1.X 最明显的改动就是取消了集中管理配置的 master 和 Zookeeper。Flume 0.9 中的配置过于冗长，很容易产生错误。此外，集中的配置管理确实脱离了 Flume 的目标范围。集中的配置被本地磁盘上简单的配置文件所取代（配置文件是可插拔的，这样它就可以被随时替换）。这些配置文件可以轻松的用 cf-engine、chef 和 puppet 来进行分发到相应的 Flume 机器上。如果你正在使用 Cloudera 分布式系统，看一下 Cloudera 管理工具，用它来管理你的配置文件——他们的许可最近取消了节点数量的限制，所以这对你来说是一个有吸引力的选择。确保你不需要手动管理这些配置文件，否则你将永远需要手动来改动配置文件。

Flume 1.X 另一个主要的不同点是读入数据和写出数据现在由不同的工作线程处理（称为 Runner）。在 Flume 0.9 中，读入线程同样做写出工作（除了故障重试）。如果写出慢的话（而不是完全失败），它将阻塞 Flume 接收数据的能力。这种异步的设计使读入线程可以顺畅的工作而无需关注下游的任何问题。

本书涉及的 Flume 版本是 1.3.1 (在本书写作的时候)。

## HDFS 和流式数据之间的问题

HDFS 不是一个真正的文件系统，至少在传统的意义上来说不是，许多在普通文件系统里我们认为理所当然的东西在这里不适用，例如能够挂载。这使得让传输数据到 Hadoop 有点复杂。

在常规可移植操作系统接口(POSIX)类型的文件系统中，如果你打开了一个文件并且写入数据，在文件关闭之前它始终是存在于硬盘上的。这就是说如果另外一个程序打开了同样的文件，并且开始读取数据，它将可以获取已经写入到磁盘上的数据。此外，如果写操作过程中断，写入到磁盘上的那部分是可用的(可能是不完整的，但它的确存在)。

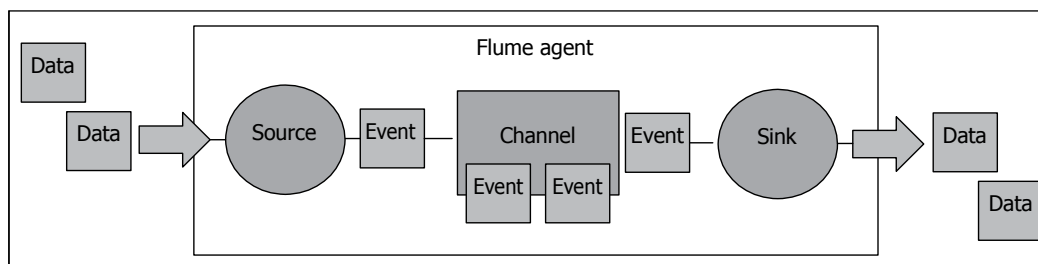
在 HDFS 文件只存在一个目录条目，直到文件关闭，它显示的大小都为零。这意味着如果数据写入的那个文件在较长时间内没有关闭，与客户端的网络断开会让你所有的努力除了得到一个空文件外一无所有。这可能会让你得出结论，写小文件将是明智的，这样你就能尽快关闭它们。

问题是 Hadoop 不喜欢大量的小文件。因为 HDFS 元数据保存在 NameNode 的内存中，您创建的文件越多，您就需要使用更多的 RAM。从 MapReduce 角度，小文件会导致效率低下。通常，每一个映射器分配一块文件作为输入(除非你使用某些压缩编解码器)。如果你有大量的小文件，启动工作进程的成本相对于它所需要处理的数据来说会很高。这种块碎片在整个任务运行时也导致增加了更多 mapper 任务。

这些因素在确定写 HDFS 旋转周期时需要考虑进去。如果你的计划是对数据进行短期保存，那么你可以倾向于较小的文件大小。然而如果你打算长期保存数据，你可以写大文件或定期清理紧小文件到更少数量的大文件中使它们有利于 MapReduce。毕竟，你只摄取数据一次，但你可能会成百上千次的运行 MapReduce 工作处理那些数据。

## Sources, channels 和 sinks

**Flume agent** 的体系结构可以从这个简单的图中看出一二。输入被称为 **source** 和输出称为 **sink**。**channel** 提供了 **source** 和 **sink** 之间的连接。所有这些运行在一个较 **agent** 的守护进程中。



每个人都应该记住以下事项：



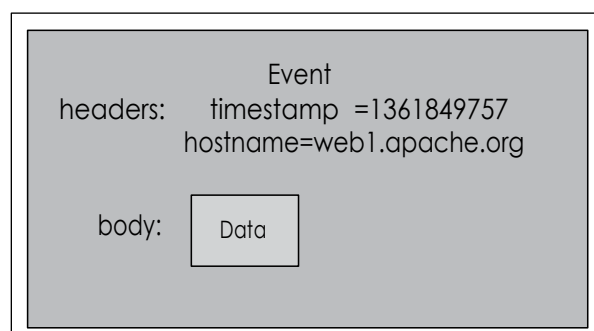
- 一个 **source** 写 **events** 到一或多个 **channels** 中；
- 一个 **channel** 是 **events** 从 **source** 传输到 **sink** 的等候区；
- 一个 **sink** 只可以从一个 **channel** 中接收 **events**；
- 一个 **agent** 可以有多个 **source**、**channel** 和 **sink**。

## Flume events

由 Flume 传输的原子数据称为 **event**。一个 **event** 是由零个或多个 **header** 和正文组成的。

**Header** 是 **key/value** 形式的，可以用来制造路由决策或携带其他结构化信息(如事件的时间戳或事件来源的服务器主机名)。你可以把它想象成和 HTTP 头一样提供相同的功能—通过该方法来传输正文意外的额外信息。

**Body** 是一个字节数组，包含了实际的内容。如果你的输入是 **tail** 命令得到的日志文件，这个数组最有可能是包含了一行数据的 UTF-8 编码的字符串。

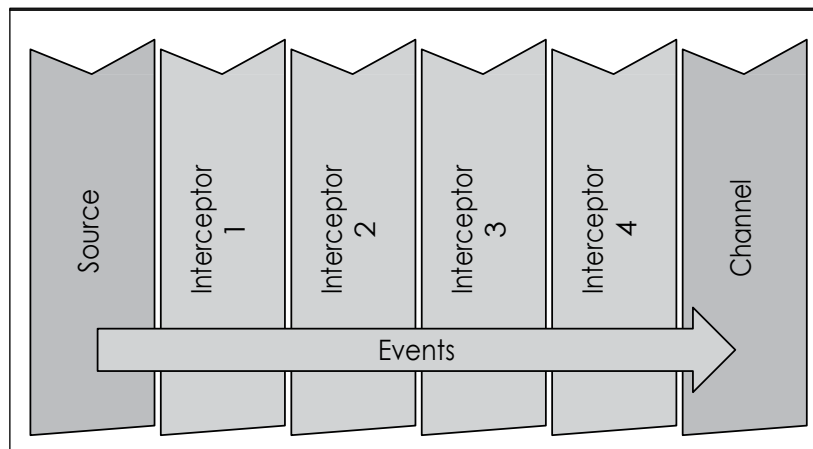


Flume 可以自动添加附加的 **header** 信息（例如：当一个 **source** 添加数据来源的主机名或创建一个事件的时间戳），但 **body** 绝大多数情况下是不会受影响的，除非你在传输时使用拦截器编辑它。



## 拦截器、channel 选择器和 sink 处理器

**interceptor** 是数据流中检查和修改 Flume **event** 的地方。你可以串联零到多个拦截器在 **source** 创建 **event** 后或者在 **sink** 发送 **event** 到目的地之前。如果您熟悉 Spring AOP 框架，它类似于 `MethodInterceptor`。在 Java servlet，它类似于 `ServletFilter`。这里有一个在 **source** 上使用四个串联拦截器的例子：



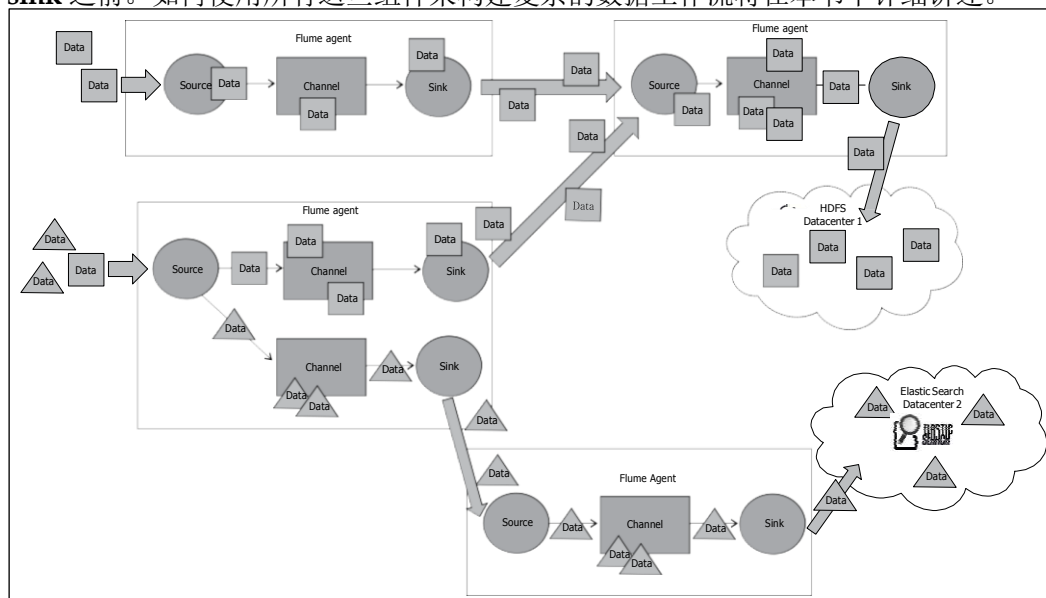
**Channel** 选择器负责如何将数据从 **source** 移动到一个或多个 **channel**。Flume 打包自带了两个 **channel** 选择器，包括了你可能用到的绝大多数场景，虽然您也可以在必要的时候实现自己的选择器。假设您已经配置了不止一个 **channel**，**replicating channel** 选择器(默认)仅仅把 **event** 复制到每个 **channel** 中。相比之下，**multiplexing channel** 选择器可以根据特定的 **event** 头信息写不同的 **channel**。结合拦截器逻辑，这种组合形成了路由 **event** 到不同 **channel** 中的基础。

最后，**sink** 处理器可以帮你的 **sink** 做到创建故障转移路径或者通过多个 **sink** 获取一个 **channel** 中的 **event** 来做到负载均衡。

## 分层数据收集(多重的流和 agents)

你可以根据您的特定使用场景串联 Flume agent。例如，您可能想要以分层的方式插入一个 agent 层来限制直接连接到您的 Hadoop 集群的客户端数量。更有可能你产生数据源的机器没有足够的磁盘空间来处理长期停机或维护窗口，所以你创建一个有大量的磁盘空间的层在数据源和 Hadoop 集群之间。

在下图中可以看到，有产生数据的地方(左边)和两个数据的终点(HDFS 和 ElasticSearch 在右边的云气泡中)。为了让事情更有趣，让我们说说机器生成的两种数据(我们称之为正方形和三角形数据)。你可以看到在左下方的 agent 中，我们使用 **multiplexing channel** 选择器将两种数据分离到不同的 **channel** 中去。矩形 **channel** 被路由到右上角的 agent 中(连同来自左上方 agent 中的数据)。合并后的 **event** 一同写到 **HDFS Datacenter 1** 中。与此同时三角形数据发送到写入 **ElasticSearch Datacenter 2** 的 agent 中。记住，数据转换可以发生在任何 **source** 之后或者任何 **sink** 之前。如何使用所有这些组件来构建复杂的数据工作流程将在本书中详细讲述。



## 小结

在这一章里，我们讨论了 **Flume** 试图解决的问题：通过简单的配置和可靠的方式传输数据到您的 **Hadoop** 集群中进行数据处理。我们还讨论了 **Flume agent** 及其逻辑组件包括: events、sources、channel selectors、channels、sink processors 和 sinks。

下面的章节将会详细、具体地讲述那些最常用的组件。像所有优秀的开源项目，几乎所有这些组件是可扩展的，如果自带的组件不能满足你的需求。

# 2

## Flume 快速入门

在前面的章节中，我们介绍一些基础知识，本章将帮助您开始使用水槽。所以，让我们从第一步开始，下载和配置 **Flume**。

### 下载 Flume

让我们从 <http://flume.apache.org/> 下载 Flume。寻找侧边导航栏的下载链接，您将看到两个压缩的 **tar** 存档，一个用于校验，一个主文件。如何验证下载文件的正确性网上有，这里我们就不谈了。检查校验码和文件内容得到的校验码是否相同来以此来检验下载文件是否损坏。检查签名文件确认你下载的所有文件(包括校验和签名)来自 **Apache** 而并不是其他非法的地方。你真的需要验证你的下载吗？总的来说这是一个好主意，**Apache** 建议您这样做。如果你选择不这么做,我不会说。

编译好的二进制文件在文件名中以 **bin** 标识，源文件在文件名中以 **src** 标识。源存档只包含了 **Flume** 的源代码。编译好的二进制发行版较源文件比大得多，因为它不仅包含了 **Flume** 源代码以及 **Flume** 编译后的组件 (**jar**、**javadoc** 等等)，而且包含了所有的依赖的 **Java** 库。二进制包包含相同的 **Maven POM** 文件归档，所以你总是可以重新编译源代码即使你从二进制发行版开始。

在我们开始时直接下载编译好的文件会为我们节省很多时间。

## Flume 在 Hadoop 发行版本中

Flume 在一些 Hadoop 发行版本中已经包含了。Hadoop 发行版可能绑定了依稀的 Hadoop 的核心组件和卫星项目(如 Flume)，诸如版本兼容性和额外的 bug 修复已经被考虑。这些发行版并没有好坏之分，只是各部相同。

这里是使用 Hadoop 发行版本的有好处。其他人已经做过了汇总所有版本兼容组件的工作。自从 Apache Bigtop(<http://bigtop.apache.org/>)项目开始后，这些问题已经不在是问题。然而,在预先构建的标准操作系统包如 RPMs 和 DEBs 简化了安装过程，同事提供了启动/停止脚本。每个发行版本都有不同程度的免费到付费的选项包括支付专业服务如果你真的遇到了你无法处理的情况。

当然发行版也有缺点。发行版本中捆绑的 Flume 的版本往往有点落后于 Apache 的版本。如果对一些新特性感兴趣，你要么等待你的发行版的提供者移植它，或者自己动手打补丁。此外，发行版的供应商做了大量的测试,如大部分的常用平台，你很可能会遇到一些他们的测试没有覆盖的地方。在这种情况下,你需要想出一个解决方案或深入代码修复它，或寄希望于开源社区中有人提交了补丁(在未来的某个时候会进入你的发行版的更新中或下一个版本中)。

所以问题在 hadoop 发行版本中进展的很慢。你可以认为这是好事或者坏事。大公司通常不喜欢不稳定的前沿技术，或经常改变版本，因为改变极有可能需要停机维护。为了稳定和兼容性，你很难找到使用最新的 Linux 内核而不是像 Red Hat Enterprise Linux(RHEL)、CentOS、Ubuntu LTS 或任何其他发行版的公司。如果你是一个启动建设下一代互联网的时尚范儿，您可能需要超前的特性来在竞争关系中分得一份蛋糕。

如果你正在考虑发行版本，做下调研，看看各个版本你能得到的(或必须舍弃的)。记住每一个产品都是希望你最终想要的或需要企业提供的，这些通常都不便宜。好好做下功课。



这里是一些发行版本的列表，可以进去看到更多信息：

- Cloudera: <http://cloudera.com/>
- Hortonworks: <http://hortonworks.com/>
- MapR: <http://mapr.com/>

## Flume 配置文件概览

现在我们下载好了 Flume，让我们花些时间来过一遍，看下如何配置一个 agent。

Flume agent 的默认配置提供者使用了一个简单的 Java 属性文件的键/值对，在启动时将它作为参数传递给 agent。因为你可以在一个文件中配置多个代理，您需要另外通过一个 agent 标识符(称为 agent 的名称)，这样它知道该使用哪个配置了。在我的例子中，我只指定一个 agent，我将使用“agent”作为其名称。

每个 agent 的配置以三个参数开始：

```
agent.sources=<source 列表>
agent.channels=<channel 列表>
agent.sinks=<sink 列表>
```

每一个 **source**、**channel** 和 **sink** 同样在该 **agent** 中有一个独一无二的名字。例如，如果我要运输 Apache 的访问日志，我可能会定义一个名叫“access”的 **channel**。这个通道的配置都从前缀 `agent.channels.access` 开始。每个配置项都有一个类型属性，告诉 Flume 是什么 **source**、**channel** 或 **sink**。在本例中，我们将使用一个内存 **channel**，类型为“memory”。该 **agent** 叫“agent”其中叫“access”的 **channel** 的完整配置如下：

```
agent.channels.access.type=memory
```

对于 source、channel、sink 的任何其他参数都用同样的前缀。任何参数来源,渠道,或添加额外的属性使用相同的前缀。Memory channel 有一个“capacity”参数参数来表示它所能容纳的 event 的最大数量。比方说我们不想使用默认值 100 个, 现在我们的配置如下:

```
agent.channels.access.type=memory
agent.channels.access.capacity=200
```

最后, 我们需要将“access” channel 的名称添加到“agent.channels”属性中, 这样 agent 就知道需要去加载它了:

```
agent.channels=access
```

让我们来看一个完整的示例, 就用典型的“Hello World”示例开始吧。

## 以"Hello World"开始

缺少了“Hello World”例子, 一本技术书籍肯定是不完整的。这是我们将使用的配置文件:

```
agent.sources=s1
agent.channels=c1
agent.sinks=k1
agent.sources.s1.type=netcat
agent.sources.s1.channels=c1
agent.sources.s1.bind=0.0.0.0
agent.sources.s1.port=12345
agent.channels.c1.type=memory
agent.sinks.k1.type=logger
agent.sinks.k1.channel=c1
```

这里我定义了一个 agent (名叫“agent”), 它有一个叫“s1”的 source, 一个叫“c1”的 channel, 一个叫“k1”的 sink。

s1 source 的类型是“netcat”, 它仅仅开启了一个端口来监听 event (每行数据就是一个 event)。它需要两个参数, 一个绑定 IP 和一个端口号。在本例中, 我们使用的是 0.0.0.0 绑定地址(在 Java 中它表示不确定地址或者任何地址, 它会告诉系统不管哪个网卡收到这个端口的数据都由我来处理)和端口 12345。source 的配置也有一个参数称为 channels (复数), 配置的内容是通道的名称, source 会把接收到的数据传输到这个 channel 中, 在本例中为“c1”。这是复数, 因为您可以配置一个源写入多个 channel, 在这个简单的例子中我们只是没有这么做。

c1 channel 是一个 memory channel, 采用了默认配置。



k1 sink 的类型是 logger。这种 sink 主要是用于调试和测试。它将使用 log4j 的 INFO 日志层级记录从它所配置的 channel 中获取到的所有 event 信息，在本例中为 c1 channel。这里的通道关键字是单数,因为一个 sink 只能从一个 channel 消费数据。

让我们用这个配置来运行 anget，并且通过 Linux 的 netcat 功能来连接 agent 并发送一个 event。

首先，解压我们先前下载好的编译过的 Flume tar 文件：

```
$ tar -zxvf apache-flume-1.3.1.tar.gz
$ cd apache-flume-1.3.1
```

然后，让我们简单的看下 Flume 的帮助命令中的内容。运行 flume-ng 命令，并带上 help 命令：

```
$ ./bin/flume-ng help
```

用法: ./bin/flume-ng<command> [options]...

commands:

help	查看帮助信息
agent	运行 Flume agent
avro-client	运行一个 Flume avro 客户端
version	查看版本信息

global options:

--conf, -c <conf>	配置目录
--classpath, -C <cp>	添加到类路径的目录
--dryrun, -d	预演，不真实启动 Flume，只输出命令行
-Dproperty=value	设置 Java 系统的属性

agent options:

--conf-file, -f <file>	指定的配置文件名称（必选） 这里面配置了 Flume 最核心的内容
--name, -n <name>	该 agent 的名称（必选）
--help, -h	显示帮助文本

**avro-client options:**

**--dirname<dir>** avro source 流产生的目录  
**--host, -H <host>** events 将要被发送去的 host 名称 (必选)  
**--port, -p <port>** events 将要被发送去的端口 (必选)  
**--filename, -F <file>** avro source 流产生的文件名 [默认: stdinput]  
**--headerFile, -R <file>** 在每一个新行上面包含的 event header 以 key/value 的形式存在  
**--help, -h** 显示帮助文本



注意如果<conf>目录被指定, 那么它将首先包含在 classpath 中

正如您可以看到的,有两种方法可以调用命令(除了不重要的帮助和版本命令)。我们将使用 agent 命令。avro-client 将以后的使用。

Agent 命令有两个必选的参数,一个是要使用的配置文件,另一个是 agent 的名称 (防止你的 agent 配置中包含多个 agent)。让我们用编辑器打开我们的示例配置文件 (我用的是 vi 编辑器,你可以用你自己喜欢的任何编辑器):

```
$ vi conf/hw.conf
```

接下来,用 vi 编辑器来输入我们的配置内容,保存,退出到命令窗口。

现在你可以开始运行 agent 了:

```
$ ./bin/flume-ng agent -n agent -c conf -f conf/hw.conf  
-Dflume.root.logger=INFO, console
```

-Dflume.root.logger 属性覆盖了 conf/log4j 中输出路径的配置项。如果我们不覆盖 root logger, 一切仍然运行, 但是数据将会输出到 log/flume.log 文件中。当然, 你也可以仅仅通过编辑 conf/log4j.properties 文件, 改变 flume.root.logger 属性 (或者其他任何你想做的) 来达到同样的目的。

你可能会问既然-f 参数包含完整的相对路径，为什么你还需要指定-c 参数。这样做的原因是 log4j 配置文件需要包含在 classpath 中。如果你忘记了-c 参数，在命令窗口中你会看到下面的错误：

```
log4j:WARN No appenders could be found for logger
(org.apache.flume.lifecycle.LifecycleSupervisor).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See
http://logging.apache.org/log4j/1.2/faq.html#noconfig for
more info.
```

但如果你没忘记那个参数，你将看到如下的日志：

```
2013-03-03 12:26:47, 437 (main) [INFO -
org.apache.flume.node.FlumeNode.start(FlumeNode.java:54)]
Flume node starting - agent
```

这一行日志告诉你名为“agent”的 agent 已经开始运行。当你多次修改配置文件时，通常看到这行你就可以肯定你用正确的配置启动：

```
2013-03-03 12:26:47, 448 (conf-file-poller-0) [INFO -
org.apache.flume.conf.file.
AbstractFileConfigurationProvider$FileWatcherRunnable.ru
n (AbstractFileConfigurationProvider.java:195)]
Reloading configuration file:conf/hw.conf
```

这是另一种明智的方法来检查确保你正确加载配置文件，在本例中是“hw.conf”文件：

```
2013-03-03 12:26:47, 516 (conf-file-poller-0) [INFO -
org.apache.flume.node.nodemanager.DefaultLogicalNodeManager.
startAllComponents(DefaultLogicalNodeManager.java:106)]
Starting new
configuration:{ sourceRunners:{s1=EventDrivenSourceRunner:
{ source:org.apache.flume.source.NetcatSource{name:s1,
state:IDLE}
}} sinkRunners:{k1=SinkRunner:
{ policy:org.apache.flume.sink
.DefaultSinkProcessor@42552ccounterGroup:{ name:null counters:{}
} }} channels:{c1=org.apache.flume.channel.MemoryChannel{name:
c1}} }
```

一旦所有的配置解析完成你会看到这个信息，它显示了所有配置。你可以看到 s1、c1 和 k1 和 Java 类正在工作。正如你可能猜到了，netcat 是 org.apache.flume.source.NetcatSource 的简写。如果我们想，我们也可以使用完整的类名。事实上，如果我有自定义 source 实现，我将使用其 classname 来作为 source 的 type 参数的值。除非你自己给 Flume 打补丁，否则你不可能有自定义组件的简写：

```
2013-03-03 12:26:48, 045 (lifecycleSupervisor-1-1) [INFO -
org.apache.flume.source.NetcatSource.start
(NetcatSource.java:164)] Created
serverSocket:sun.nio.ch.ServerSocketChannelImpl[/0.0.0.0:12345]
```

这里我们看到我们 source 正在监听 12345 端口的输入。那么让我们发送一些数据吧。

最后，打开另一个命令窗口。我们将用 nc 命令（你也可以使用 telnet 或者其他更简单的名称）来发送“Hello World”字符串，点击<RETURN>来结束 event：

```
% nc localhost 12345
Hello World<RETURN>
OK
```

“OK”在按回车后来自于 agent，表示其已经收到了这行作为单个 Flume event 的数据。如果你查看 agent 日志，你将看到如下内容：

```
2013-03-03 12:39:58, 582 (SinkRunner-PollingRunner-
DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:70)]
Event: { headers:{} body: 48 65 6C6C6F 20 57 6F 72 6C 64
Hello World }
```

这个日志显示了这个 Flume agent 不包含 header（netcat source 自身不会增加任何东西）。body 以 16 进制和字符串显示（人来读的话，在这个例子中就是“Hello World”消息）。

如果我发送像下面这行的话：

```
The quick brown fox jumped over the lazy dog.<RETURN>
OK
```

你会在 agent 日志中看到如下内容：

```
2013-03-03 12:45:08, 466 (SinkRunner-PollingRunner-
DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:70)]
Event: { headers:{} body: 54 68 65 20 71 75 69 63 6B 20 62 72
6F
77 6E 20 The quick brown }
```

Event 似乎被截断了。logger sink 的设计限制了 body 内容不能超过 16 byte，以避免你的屏幕被这些数据填满，超出你在调试环境所需要的数据量。如果你需要看调试的完整内容，你应该使用其他 sink，如 file\_rollsink，它会将数据写入本地的文件系统。

## 小结

在这一章里,我们讲述了下载 Flume 二进制发行版。我们创建了一个简单的配置文件,包括一个 source 写入一个 channel，该 channel 为一个 sink 提供数据。Source 监听了网络的一个端口，network 客户端可以连接该端口并向其发送 event 数据。这些 event 被写入到一个 memory channel 中，然后传输数据到 log4j sink 来进行输出。然后我们实验了与我们 agent 监听的端口进行通讯，我们使用 Linux netcat 实用工具和一些字符串发送 event 到我们 Flume agent 的 source 中。最后,我们见证了基于 log4j 的 sink 将 event 正确的进行了输出。

在下一章我们将详细看看在你的数据处理工作中可能会使用的两个主要 channel 类型：

- Memory channel
- File channel

对于每种类型的 channel，我们将讨论它们的所有的可用配置选项，什么时候、什么情况下你需要改变默认配置，最重要的是你为何使用这一种而非另一种。



# 3

## Channels

在 Flume 中，channel 是 source 和 sink 中间的组件。在你的数据处理流中，当 event 从 source 中读取到，它为你的 event 提供了一个缓冲区，直到他们可以被写到 sink 中。

我们接下来会讨论的两种类型的 channel 是以内存为基石/非持久化的 channel 和以本地文件系统为基石的/持久化的 channel。在向发送者发出确认收到 event 之前，持久化的文件 channel 会刷新所有的改动到硬盘上。这大大低于非持久化 channel 的处理速度，但在系统或 Flume 重启后提供了数据可恢复性。相反，内存通道要快得多，但故障会导致数据丢失，和文件通道 TB 级的磁盘存储容量相比要低得多。你选择哪个 channel 取决于你的特定的用例，失败场景，和风险承受能力。（PS: Flume 1.5 支持混合型 channel，但稳定性有待实验）

也就是说，不管你选择什么 channel，如果你从 source 到 channel 的摄取速度比 sink 消耗的速度快的话，你将超过 channel 的容量，并最终抛出一个 ChannelException 异常。你的 Source 对于 ChannelException 做或不做什么都是各个 source 自身的特性，但在某些情况下会丢失数据，所以你要避免填充大小不合适的 event。事实上，你总是希望你的 sink 写的速度能够比 sourced 读入的速度快。否则，你可能会进入一个状态，一旦你的 sink 比 source 慢你将永远无法赶上。如果你的数据量和网站的访问量有关，你可能白天有较高的数据量，在晚上有较低的数据量，给你的 channel 来消化。在实践中，你会想去尝试并保持 channel 的深度（event 在 channel 中的数量）尽可能的低，因为在最终到达目的地之前花在 channel 上的时间会转换为数据延迟。



## 内存 channel

Memory channel，正如预想的一样，是一种将流转中的数据存储在内存在中的 channel。由于内存（通常）提供了比硬盘快几个数量级的速度，event 能够被更快速的摄取从而减少硬件的需求。使用该 channel 缺点是 agent 故障(硬件问题、停电、JVM 崩溃、Flume 重启等等)会导致数据丢失。结合你的使用场景，这种 channel 可能非常好。系统指标通常属于这一类，丢失一些数据不会产生灾难性影响。然而，如果你的 event 记录了你网站上的交易记录，那 memory channel 将是一个非常糟糕的选择。

要使用 memory channel，设置你 channel 参数中的 type 值为 memory 即可。

```
agent.channels.c1.type=memory
```

这里在一个名叫“agent”的 agent 中定义了一个叫“c1”的 memory channel。

表格中是 memory channel 中可以调整的配置参数：

Key	Required	Type	Default
type	Yes	String	memory
capacity	No	int	100
transactionCapacity	No	int	100
byteCapacityBufferPercentage	No	int (percent)	20%
byteCapacity	No	long (bytes)	80% of JVM Heap
keep-alive	No	int	3 (seconds)

Memory channel 的默认容量是 100 个 event。通过像下面这样设置可以改变 capacity 的默认属性值：

```
agent.channels.c1.capacity=200
```

记住如果你增加了这个值，你可能同样需要增加你的 Java 堆栈大小，使用 -Xmx 参数，选用 -Xms 参数（译者注：这两个参数可以在配置文件目录下的 flume-env.sh 中指定，或者在启动命令中指定）。

另一个与容量相关的设置是 transactionCapacity。这是 source 的 ChannelProcessor 一次可以向 channel 中写入的最大 event 数量，该组件负责在一个事务中移动 source 中的 event 到 channel 中。这个值同样是 sink 的 SinkProcessor 一次可以从 channel 中去除的最大 event 数量，该组件负责从 channel 中取出 event 到 sink 中。你可能想将这个值设置的尽量大来减少事务包装带来的消耗，这种做法也许会提升速度。但随之而来的坏处就是，在发生故障时，source 和 sink 就必须回滚更多的数据。



Flume 只对各个 agent 中的各个 channel 提供了事务保障。在多 agent、多 channel 配置的情况下重复发送和乱序发送是可能的，但不应该视为正常情况。如果你在没有错误发生的情况下重复收到的数据，这意味着你需要继续调试你的配置。

如果你正在使用的 sink 会写数据到那些适合于大批次工作的地方（比如 HDFS），你也许想要设置 `transactionCapacity` 越大越好。就像很多其他事情一样，唯一用来确定最佳值的方法就是用不同的值来做性能测试。由 Flume 代码贡献者 Mike Percy 撰写的博客 <http://bit.ly/flumePerfPt1> 也许可以给你一些好的想法。

`byteCapacityBufferPercentage` 和 `byteCapacity` 参数在 <https://issues.apache.org/jira/browse/FLUME-1535> 中有详细介绍，作为一种度量 memory channel 容量的方法，使用 `byte` 比使用 `event` 的数量更合适，以试图避免内存溢出异常。如果你的 `event` 在大小上有很大的差异，你可能会使用到这些参数来调整容量大小，但要注意预估出来的值只能来自 `event` 的 `body` 部分。如果你的 `event` 的 `header` 部分有内容，你的实际内存消耗将会比配置的要大些。

最后，`keep-alive` 参数是线程写数据到 channel 中遇到 channel 爆满时在放弃之前的等待时长。由于数据在这个时候会被 sink 不断消费掉，如果在超时之前 channel 空间腾出来了，数据将会被写到 channel 中而不是对 source 抛出一个异常。你也许禁不住诱惑想把这个值设置得非常长，但请记住等待写入 channel 的时候将会阻塞随后进入你 source 中的数据，这可能会导致数据在上游的 agent 中发生回退。最后，这可能会导致 event 被丢弃。你需要为周期性的峰值流量和计划中（或计划外）的维护来定夺这个值的大小。

## 文件 channel

File channel 就是将 event 存储在本地文件系统上的 channel。虽然它的速度比 memory channel 慢，但它提供了持久化的存储方式可以解决绝大多数问题，这种 channel 应该被用在那些不容许数据流产生误差的场景下。

这种耐久性由预写日志系统（WAL）结合一个或多个文件存储目录来支持。WAL 用于以一种原子安全的方式跟踪 channel 中所有的输入和输出。通过这种方式，如果 agent 重新启动，WAL 可以回放来确保所有写入 channel 中的 event 在本地文件系统中被清空之前已经被读出了。在必要时可以重播，确保所有的事件，进入频道(使)写出来(需要)之前存储的数据可以被净化从本地文件系统。

此外,file channel 支持对写入文件系统的数据进行加密, 你的数据处理政策要求所有磁盘(即使是临时的)都是被加密的数据。我不会在这里讲述它, 但是如果你需要, Flume User Guide 中有一个例子

(<http://flume.apache.org/FlumeUserGuide.html>)。请记住,使用加密将减少您的 file channel 的吞吐量。

要使用 file channel, 设置你 channel 中 type 参数值为 “file”:

```
agent.channels.c1.type=file
```

这里为名叫 “agent” 的 agent 定义了一个名叫 “c1” 的 file channel。

表格中是 File channel 中可以调整的配置参数:

Key	Required	Type	Default
type	Yes	String	file
checkpointDir	No	String	~/.flume/file-channel/ checkpoint
dataDirs	No	String (comma-separated list)	~/.flume/file- channel/ data
capacity	No	int	1000000
keep-alive	No	int	3 (seconds)
transactionCapacity	No	int	1000
checkpointInterval	No	long	300000 (milliseconds - 5 min)
write-timeout	No	int	10 (seconds)
maxFileSize	No	long	2146435071 (bytes)
minimumRequiredSpace	No	long	524288000 (bytes)

为了指定 Flume agent 本地存储数据的位置, 你需要设置 checkpointDir 和 dataDirs 参数:

```
agent.channels.c1.checkpointDir=/flume/c1/checkpoint
agent.channels.c1.dataDirs=/flume/c1/data
```

从技术上讲，这些属性不需要，因为已经为开发人员设置了合理的默认值。然而，如果你有的 agent 中配置有多个 file channel，只有第一个 channel 可以启动。为了在生产环境部署和多 file channel 的开发工作，你应该为每个 file channel 的存储位置指定绝对路径，另外可以考虑在不同的硬盘上部署不同的 file channel 来避免 IO 竞争。此外，如果你想用一些大型的机器可以考虑使用一些磁盘阵列来获得高磁盘性能取代购买那些昂贵的 10k、15k 驱动器或者 SSD。如果你没有磁盘阵列，但有很多个硬盘，可以设置 dataDirs 属性，以逗号分隔来指定每个存储位置从而组成一个列表。使用多个磁盘可以将磁盘的速度提升到和条带状的 RAID 差不多，但是不计算 RAID 50/60 开销的话，和 RAID 10 相比 50% 的空间会被浪费掉。你需要测试你的系统来看看是否 RAID 的开销值得来换取速度的差异。因为硬盘故障是真实存在的，你可能对比单一磁盘更喜欢某些 RAID 配置，使你免受单点故障导致数据丢失的痛苦。

因为同样的理由，NFS 的存储方式同样应该避免。使用 JDBC channel 是一个很差的主意，因为它将引入一个瓶颈和单点故障问题而不是设计成为一个高度分布式的系统。



在使用 file channel 时一定要设置 HADOOP\_PREFIX 和 JAVA\_HOME 环境变量。虽然我们看似没有任何使用 Hadoop 特性的地方（比如写数据到 HDFS），但 file channel 使用了 Hadoop 序列化格式作为磁盘上的序列化格式。如果 Flume 无法找到 Hadoop 库，你也许会在你的启动命令中看到如下内容，所以请检查下你的环境变量：

```
java.lang.NoClassDefFoundError: org/apache/hadoop/
io/Writable
```

默认的 file channel 容量是一百万个 event，无论 event 的大小是多少。如果 channel 容量满了，source 将无法再接收数据。这个默认值在低容量的情况下应该是没有问题的。如果你的摄取量足够大，你将无法承受计划中或计划外的停机，这种情况下你也许想把这个值设的高一些。例如，你的 Hadoop 集群中有很多东西需要更改，这需要集群的重启。如果你的 Flume 需要写很重要的数据到 Hadoop 中，file channel 应该被设置到可以忍受 Hadoop 重启的时间（也许还应该为意想不到的情况留一些缓冲时间）。如果您的集群或其他系统是不可靠的，你可以将这个值设置得更大些来应对系统长时间不可用的情况。在某种程度上你会碰到这样一个事实：你的磁盘空间是一种有限的资源，所以你必须有个上线（或购买更大的磁盘）。

`keep-alive` 参数和 `memory channel` 中相似。它是 `source` 尝试写数据到一个爆满的 `channel` 中的最长等待时间。在超时之前如果空间可用了，就会写成功；否则就会向 `source` 抛出一个 `ChannelException` 异常。

`transactionCapacity` 参数是在一个事务中最大可以允许的 `event` 数量。这对于特定的 `source` 来说也许很重要，`source` 会将 `event` 批量打包，然后在一次请求中来传输它们。在大多数情况下你不需要改变这个值。将这个值设高将会在内部占用额外的资源，所以除非你遇到性能问题，否则你不应该增加它。

`checkpointInterval` 参数是执行 `checkpoint`（将日志文件写入到 `logDirs` 中）之间的毫秒数。你不能设置这个值低于 1000 毫秒。

`Checkpoint` 也同样根据 `maxFileSize` 属性中指定的数据量大小将数据写入文件中。你可以为低速度的 `channel` 来降低这个值以节省一些磁盘空间。比方说你的最大文件大小是 50000 字节，但 `channel` 每天只写 500 字节，这将需要 100 天来填满一个日志。比方说你在第 100 天的时候突然来了 2000 个字节的数据。一部分数据将被写入到旧的文件中，溢出的一部分将被写入到一个新文件中。在滚动文件后，`Flume` 尝试去删除一些不再需要的日志文件。由于满了的日志文件中有没有处理的数据，它将无法被清理。下一次可以去清理旧日志文件的机会可能需要再等上 100 天。也许 50000 个字节的数据存在更久并不重要，但由于默认值是 2GB，你的每个 `channel` 可能需要两倍（4GB）的硬盘空间。依据你有多少硬盘空间以及你 `agent` 的配置中有多少个 `channel`，这个配置可能是个问题，也可能根本不是问题。如果你有足够的磁盘空间，默认值也许不错。

最后，`minimumRequiredSpace` 属性是你不想用于写日志文件的空间大小。缺省配置将抛出一个异常，如果你尝试使用 `dataDir` 路径中的最后 500 MB 磁盘。这一限制适用于所有 `channel`，所以如果你配置了三个 `file channel`，上限仍然是 500 MB，而不是 1.5 GB。你可以设置这个值小到 1 MB，但一般来说，坏事往往会发生在你将磁盘利用空间到 100% 时。

## 小结

在这一章里,我们介绍了两种你最有可能在你的数据处理管道中用到的 `channel` 类型。

`Memory channel` 提供了速度保障, 但代价是故障时会丢失数据。

另外,`file channel` 提供了更可靠的运输保证, 它可以容忍 `agent` 失败和重新启动, 代价是性能损耗。

你需要依据你应用场景来决定使用哪种 `channel`。当试图确定 `memory channel` 是否合适时, 不妨问问自己如果数据丢失了会带来怎样的经济损失。当你最终决定使用持久化 `channel` 前, 先丈量下通过消耗更多硬件来弥补性能差异时的花费。另一个考虑是数据是否可以重复发送。并不是所有你摄取到 `Hadoop` 中的数据都来应用程序产生的流式日志。如果你收到的是按天获取的数据, 您可以侥幸使用 `memory channel`, 因为如果你遇到了问题, 你可以重新导入。



可能（或有意）重复的 `event` 是流式摄取数据中的一个现实。有些人会定期运行 `MapReduce` 工作了来清理数据（在找到重复数据时进行删除操作）。另外一些人只会在运行 `MapReduce` 工作时处理重复数据, 这样可以节省额外的后置处理工作。在实践中你可能会两者都做。

在下一章中, 我们将要看看 `sink`, 特别是写 `event` 数据到 `HDFS` 中的 `HDFS sink`。我们还将介绍 `event` 序列化器, 特别是 `Flume event` 是如何解析为更适合该 `sink` 输出的。最后, 我们将介绍 `sink` 处理器以及如何在分层的配置中为了更健壮的传输系统来建立负载均衡和故障转移的。

# 4

## Sinks 和 Sink 处理器

现在你对 sink 在 Flume 架构中所处的地位应该有了一个更好的思维想象了。在这章里我们将要学习与最常与 Hadoop 配合使用的 sink，即 HDFS sink。Flume 中还支持许多其他 sink，我们这本书里没法全部涉及。一些 Flume 中自带的 sink 可以写数据到 HBase、IRC、ElasticSearch 中，以及我们在第二章（Flume 快速入门）中用到的 log4j 和 file sink。其他在互联网上流传的 sink，如那些写数据到 MongoDB、Cassandra、RabbitMQ、Redis 以及其他你能想到的存储方式，都有各自的 sink。如果你找不到适合你需求的 sink，你可以自己写一个，继承 `org.apache.flume.sink.AbstractSink` 类即可。

### HDFS sink

HDFS sink 做的事情就是不断的在 HDFS 上打开文件，往里面不停的灌入数据，在某些时间点关闭文件并打开另一个新的文件继续灌数据。正如我们在第一章（概览）中讨论的，文件滚动之间的间隔应该多长必须权衡于 HDFS 文件关闭的速度如何，只有文件关闭后才能用于处理程序。我们已经讨论过，大量的小文件会让你的 MapReduce 工作效率低下。

要使用 HDFS sink，在你的 sink 中设置 `type` 的值为 “hdfs”：

```
agent.sinks.k1.type=hdfs
```

这里在 “agent” 的 agent 中定义了一个名叫 “k1” 的 HDFS sink。

有一些额外的必选参数需要指定，就从 “path” 开始，它是你在 HDFS 上写数据的目录：

```
agent.sinks.k1.hdfs.path=/path/in/hdfs
```

HDFS 路径，像 Hadoop 中的大多数文件路径一样，可以以三种不同方式来指定：绝对路径、带服务器名称的绝对路径、相对路径。他们都是等价的（假设你的 Flume Agent 是以 flume 用户来运行的）：

Type	Path
absolute	/Users/flume/mydata
absolute with server name	hdfs://namenode/Users/flume/mydata
relative	mydata

我喜欢在我安装的 Flume 机器上通过 `hadoop` 的命令来配置服务名称（上述第一种绝对路径），这可以通过设置 Hadoop `core-site.xml` 配置文件中的 `fs.default.name` 属性来做到。我不会在 HDFS 用户目录下面保存持久化数据，但更喜欢使用绝对路径与一些有意义的路径名（如：`/logs/apache/access`）。我唯一需要特别指定 `name node` 的情况是目标机器是完全不同的 Hadoop 集群。如果没有意想不到的结果，这样做允许将已经测试好的配置从一个环境移动到另一个环境，比如你的生产服务器写入数据到 STG 环境的 Hadoop 集群，因为有人忘了编辑配置中的目标集群。在外部指定环境变量是一个最佳的实践以避免这种情况。

HDFS sink 最后需要的一个参数，事实上是所有 sink 都需要的，是它将要获取数据的 channel。在这里，设置 `channel` 参数的值为需要获取数据的 channel 的名称：

```
agent.sinks.k1.channel=c1
```

这里的配置告诉了 `k1` sink 需要从 `c1` channel 中读取 event。

下面是一个几乎完整的可以调整默认值的参数列表：

Key	Required	Type	Default
<code>type</code>	Yes	String	<code>hdfs</code>
<code>channel</code>	Yes	String	
<code>hdfs.path</code>	Yes	String	
<code>hdfs.filePrefix</code>	No	String	<code>FlumeData</code>
<code>hdfs.fileSuffix</code>	No	String	
<code>hdfs.maxOpenFiles</code>	No	long	<code>5000</code>
<code>hdfs.round</code>	No	Boolean	<code>false</code>
<code>hdfs.roundValue</code>	No	int	<code>1</code>



Key	Required	Type	Default
hdfs.roundUnit	No	String (second, minute, or hour)	second
hdfs.timeZone	No	String	Local Time
hdfs.inUsePrefix	No	String	(CDH4.2.0 or Flume 1.4 only)
hdfs.inUseSuffix	No	String	.tmp (CDH4.2.0 or Flume 1.4 only)
hdfs.rollInterval	No	long (seconds)	30 Seconds (0=disable)
hdfs.rollSize	No	long (bytes)	1024 bytes (0=disable)
hdfs.rollCount	No	long	10 (0=disable)
hdfs.batchSize	No	long	100
hdfs.codeC	No	String	

记住为你的使用的 Flume 经常检查下 Flume User Guide: <http://flume.apache.org/>，在这本书发行后你实际使用的版本可能已经发生了改变。

## 目录和文件名

每次 Flume 在 HDFS 中的 `hdfs.path` 目录下开启一个新文件，文件名由 `hdfs.filePrefix` 指定的一段字符、文件开始的时间戳、以及由 `hdfs.fileSuffix` 指定的可选的文件结尾组成。例如，看下面这行代码：

```
agent.sinks.k1.hdfs.path=/logs/apache/access
```

这行代码将可能在目录 “/logs/apache/access/” 下产生文件 “FlumeData.1362945258”。

然而，再看看下面的配置：

```
agent.sinks.k1.hdfs.path=/logs/apache/access
agent.sinks.k1.hdfs.filePrefix=access
agent.sinks.k1.hdfs.fileSuffix=.log
```

在这个配置下，你最终的文件名称会像这样：/logs/apache/access/access.1362945258.log。

随着时间慢慢流逝，`hdfs.path` 目录下的文件将会变得非常多，那么你也许会想要在目录中加入一些时间属性来将文件划分到各个子目录中去。Flume 支持各种基于时间的转义序列，比如 `%Y` 会被转义为一个四位数的年份。我喜欢使用如下转义格式：`year/month/day/hour`（这样它们就可以由老到新的对文件进行排序），所以我经常使用如下的格式作为目录：

```
agent.sinks.k1.hdfs.path=/logs/apache/access/%Y/%m/%D/%H
```

这表示我想要一个类似这样的目录结构：`/logs/apache/access/2013/03/10/18/`。

基于时间的转义序列的完整列表，请参阅《Flume User Guide》。另一个方便的转义序列机制是在你的路径值中能够使用 `event` 中 `header` 的信息。例如，如果 `header` 中有一个 `key` 名为“`logType`”，通过用如下的方式转义，我可以在使用相同 `channel` 的情况下将 `access` 日志和 `error` 日志分别写到不同的目录中去：

```
agent.sinks.k1.hdfs.path=/logs/apache/{logType}/%Y/%m/%D/%H
```

这会导致将 `access` 日志写到 `/logs/apache/access/2013/03/10/18/` 目录中，同时将 `error` 日志写到 `/logs/apache/error/2013/03/10/18/` 目录中去。然而，如果我想要让两种日志都存在同一个目录下，我可以在 `hdfs.filePrefix` 中用 `logType` 来替代原有写法：

```
agent.sinks.k1.hdfs.path=/logs/apache/%Y/%m/%D/%H
agent.sinks.k1.hdfs.filePrefix={logType}
```

显然，Flume 可以同时写多个文件。`hdfs.maxOpenFiles` 属性设置了一次可以打开的最多文件个数，默认是 5000 个。如果你超出了这个限制，仍然打开的最早的文件将被关闭。记住每一个打开的文件都会在操作系统层面上和 HDFS 上（`NameNode` 和 `DataNode` 链接）增加开销。

另一组可能对你有用的属性允许 `event` 上的时间在小时、分钟或者秒的粒度上进行舍入，同时在文件路径上还保留这些元素。比方说你有一个路径的规范如下：

```
agent.sinks.k1.hdfs.path=/logs/apache/%Y/%m/%D/%H%M
```

但在这时你想要每天只有四子目录(在每个小时的 00、15、30、45，每个包含 15 分钟的数据)。通过设置以下值你可以做到这一点：

```
agent.sinks.k1.hdfs.round=true
agent.sinks.k1.hdfs.roundValue=15
agent.sinks.k1.hdfs.roundUnit=minute
```

这产生的效果是在 2013-03-10 这一天 01:15:00 到 01:29:59 之间的日志将会被包含在 /logs/apache/2013/03/10/0115/ 目录中。01:30:00 到 01:44:59 之间的日志将会被包含在 /logs/apache/2013/03/10/0130/ 目录中。

`hdfs.timeZone` 属性是用来指定你用来做转义序列的时间的时区的。默认值就是你电脑的本机时间。如果你的当地时间是受日照变化影响（即夏令时），你会有两倍的数据 %H == 02(秋季)，没有数据时 %H == 02(春季)。我认为引入时区是个坏主意，因为电脑必须要读取它。我认为时区是为人发明的，电脑之间应该只有一个统一的时间。因此我在 Flume agent 上通过如下的设置让时区消失了：

```
-Duser.timezone=UTC
```

如果你不同意我的观点也可以自由的去使用默认值（本地时间），或者设置 `hdfs.timeZone` 属性到你喜欢的时区。你所设置的值将会被传到 `java.util.Timezone.getTimeZone(..)` 中，所以可以看看 Java 文档来看看这里可以用哪些值。

最后，当文件正在被写到 HDFS 中时，会在末尾有一个 “.tmp”。文件关闭后，这个扩展名也会被去掉。这让你可以在运行 MapReduce 工作的时候很轻松地将这些 Flume 正在写的文件排除在外。这同样让你可以通过查询 HDFS 的文件目录来看到哪些文件正在 Flume 写。因为你通常指定一个目录作为你 MapReduce 工作的输入（或者因为你用的是 Hive），这些临时文件经常被错误的当作空输入或混乱的输入。Flume-1702 就是用来解决这个问题的，并且将在 Flume 1.4 中发布，但如果你正巧在用 Cloudera's CDH4.2.0 发行版，这点改动已经被移植到了 Flume 1.3 中。这里引入了两个新的属性来改变“使用中”的前缀和后缀。为了避免你的临时文件在关闭之前就被拿来使用，设置后缀为空（而不是默认的.tmp），前缀设置为点或者像下面这样的下划线：

```
agent.sinks.k1.hdfs.inUsePrefix=_
agent.sinks.k1.hdfs.inUseSuffix=
```

## 文件循环

默认情况下，Flume 每 30 秒钟或每 10 个 event 或者每 1024 个字节就主动触发文件的滚动。通过分别设置 `hdfs.rollInterval`、`hdfs.rollCount`、`hdfs.rollSize` 属性可以做到这些。可以通过将其中的一个或多个值设置为 0 来禁用这些特定的滚动条件。例如，如果你只是想要一个基于 1 分钟时长的滚动条件，你像下面这样设置参数：

```
agent.sinks.k1.hdfs.rollInterval=60
agent.sinks.k1.hdfs.rollCount=0
agent.sinks.k1.hdfs.rollSize=0
```

如果您的输出包含任意数量的头信息，HDFS 每个文件大小可能会比你想要的大，因为 `hdfs.rollSize` 滚动特性只计算 `body` 部分的大小。显然你可不想同时将三个滚动机制都禁用，否则你的数据文件将挤在一个文件中。

最后，一个相关的参数是 `hdfs.batchSize`。这是 `sink` 在一个事务中从 `channel` 中读取 `event` 的数量。如果你 `channel` 中有大量数据存在，将这个值较默认值调高些，你可能会见到性能的提升，因为这降低了每个活动事务的开销。

既然我们已经讨论了文件是如何管理的和在 HDFS 中滚动文件的方式，让我们来看看 `event` 的内容是如何写入的。

## 数据压缩

**Codecs**（编码/解码）是使用不同的压缩算法对数据进行压缩和解压的。Flume 支持 `gzip`、`bzip2`、`lzo` 和 `snappy`，尽快你可能需要自己安装 `lzo`，特别是如果你用的是发行版本，如 CDH，因为涉及到许可问题。

如果你想往 HDFS 中写压缩过的文件，且想为你的数据指定压缩方法，请设置 `hdfs.codeC` 属性。这个属性同样会被用在写到 HDFS 的后缀名上。例如，如果你像下面这样指定编码方式的话，那么你的所有文件后面都会有个 `.gzip` 的扩展名，所以在这个时候你不需要指定 `hdfs.fileSuffix` 属性：

```
agent.sinks.k1.hdfs.codeC=gzip
```

选用何种编码方式需要你自己研究研究。在花费更长时间压缩的情况下，你仍然有理由为了 `gzip` 和 `bzip2` 的高压缩比而使用他们，尤其是在你的数据只写一次但要读取成百上千次的时候。另一方面，使用 `snappy` 或者 `lzo` 会得到更快的压缩速度，但与之而来的是低压缩比。请记住文件的分裂型，尤其是你正在使用纯文本文件时，将极大地影响 MapReduce 的表现。去拿一份 Hadoop Beginner's Guide（<http://amzn.to/14Dh6TA>）或者 Hadoop: The Definitive Guide（<http://amzn.to/16Osflf>），如果你不明白我在说什么时。

## Event 序列化

Event 序列化就是一种将 Flume event 转换为另一种格式输出的机制。它有点类似于 log4j 中的 Layout 类。默认情况下是 text 序列化，它的输出就是 Flume event 的 body 中的内容。另一种 header\_and\_text 序列化，会将 event 中的 header 和 body 都输出。最后，还有一种 avro event 序列化可以用来创建 event Avro 化的表现形式。如果你实现自己的序列化机制，你需要使用实现了 `EventSerializer` 接口的类作为 `serializer` 属性的值（记住自定义的实现必须是全路径的）。

## 文本输出

正如前文所述，默认的序列化器就是 text 序列化。它只会输出 Flume event 的 body 部分，header 部分的数据会被丢弃。每个 event 都是独立一行，除非你通过将 `serializer.appendNewLine` 值设为 `false` 修改了默认的行为值。

Key	Required	Type	Default
<code>serializer</code>	No	String	text
<code>serializer.appendNewLine</code>	No	boolean	true

## 带有头信息的文本输出

`Text_with_headers` 序列化允许你保留 Flume event 的 header 中的数据而不是丢弃它们。输出的格式由如下部分组成：header 部分的内容、后面加一个空格、然后是 body 部分的内容、最后是一个可选的换行符。这里是由这个序列化器生成的输出内容：

```
{key1=value1, key2=value2} body text here
```

Key	Required	Type	Default
<code>serializer</code>	No	String	text_with_headers
<code>serializer.appendNewLine</code>	No	boolean	true

## Apache Avro

Apache Avro 项目(<http://avro.apache.org/>) 提供了一个类似于谷歌草拟的缓冲区的序列化格式，但基于 HadoopSequenceFiles 且包含了一些 MapReduce 集成特性对 Hadoop 来说更加友好。这种格式自身内部使用了 JSON，使其可以长远的存储数据，因为你的数据格式可能随着时间的迁移而发生变化。如果你的数据结构复杂，你肯定想要避免转换为 String 字符串，在你要解析这些字符串到 MapReduce 工作中去的时候，你应该去更多的了解下 Avro 来确定你是否想用它来作为你在 HDFS 中的存储格式。

Avro event 序列化器基于 Flume event 的模式来创建 Avro 数据。它没有格式化参数，由于 Avro 用如下参数规定数据格式和 Flume event 的结构：

Key	Required	Type	Default
serializer	No	String	avro_event
serializer.compressionCodec	No	String (gzip, bzip2, lzo,	
serializer.syncIntervalBytes	No	int (bytes)	2048000 (bytes)

如果你想使用 Avro，当想使用与 Flume event 不同的模式，你不得不实现一个自定义的序列化器。

如果你想要你的数据在写到 Avro 容器中之前被压缩，你应该为文件扩展名设置 `serializer.compressionCodec` 属性的值为一个安装的编码解码器。

`serializer.syncIntervalBytes` 属性决定了数据刷入 HDFS 之前的缓冲大小，因此，这个设置可以在使用编码器时影响你的压缩比。这里是一个使用 snappy 压缩 Avro 数据的例子，缓冲大小为 4MB：

```
agent.sinks.k1.serializer=avro_event
agent.sinks.k1.serializer.compressionCodec=snappy
agent.sinks.k1.serializer.syncIntervalBytes=4194304
agent.sinks.k1.hdfs.fileSuffix=.avro
```

对于工作在 Avro MapReduce 中的 Avro 文件来说，他们必须以 `.avro` 结束，否则他们会在输入中被忽略掉。因此，你需要显式的设置 `hdfs.fileSuffix` 属性。此外，你不需要在 Avro 文件中设置 `hdfs.codec` 属性。

## 文件类型

默认情况下 HDFS sink 以 Hadoop 序列化文件的形式写数据到 HDFS 中。这是一个常见的 Hadoop 包装器，由二进制字符和分隔符分隔的键和值域组成。通常，假设电脑上的文本文件一个换行符结束一行。那么假设你得文本文件包含换行符你该怎么做，像 XML 那样？使用序列化文件可以解决这个问题，因为它使用非输出字符作为分隔符。序列化文件同样是可剥离的，这使得对数据运行 MapReduce 工作时有更合适的位置和并行性，特别是在大文件上。

## 序列化文件

当使用 SequenceFile 文件类型，你需要指定在 SequenceFile 中你想要写的 key/value 的格式。在每个记录的 key 上永远都会包含一个 Long 型的当前时间戳，或者如果你在 header 中已经有了时间戳，这个时间戳会被用来取代当前时间生成的时间戳。默认情况下，value 值的格式是 org.apache.hadoop.io.BytesWritable，和 Flume body 部分的 byte[] 值相同：

Key	Required	Type	Default
hdfs.fileType	No	String	SequenceFile
hdfs.writeType	No	String	writable

然而，如果你想要有效数据以 String 的格式来编译，你可以修改 hdfs.writeType 属性值，这样 org.apache.hadoop.io.Text 将会被用在 value 编译上：

Key	Required	Type	Default
hdfs.fileType	No	String	SequenceFile
hdfs.writeType	No	String	text

## 数据流

如果你不想输出序列化的文件，因为你的数据没有天然的 key 值，那可以使用 DataStream 来单独输出未压缩的 value 值。简单的修改下 hdfs.fileType 属性即可：

```
agent.sinks.k1.hdfs.fileType=DataStream
```

这里是你要使用 Avro 序列化时需要的文件类型，因为任何压缩都应该在 event 序列化器中完成。如果需要使用 gzip 来压缩 Avro 序列化的文件，你需要像下面这样设置：

```
agent.sinks.k1.serializer=avro_event
agent.sinks.k1.serializer.compressionCodec=gzip
agent.sinks.k1.hdfs.fileType=DataStream
agent.sinks.k1.hdfs.fileSuffix=.avro
```

## 压缩流

CompressedStream 和 DataStream 类似，只是数据在写的时候已经压缩过。你可以把这个想象为在未压缩的文件上运行 gzip，只不过是一步到位而已。这不同于一个压缩过的 Avro 文件（它的内容是压缩过的），然后写到一个未压缩的 Avro 包装器中。

```
agent.sinks.k1.hdfs.fileType=CompressedStream
```

记住，只有特定特定的压缩格式在 MapReduce 中是可剥离的，你是否应该决定使用 CompressedStream。压缩算法的选择在 Flume 配置中没有提供，而是在由 Hadoop 的 `zlib.compress.strategy` 和 `zlib.compress.level` 属性决定的。

## 超时和工作线程数量

最后，还有两个混合使用的和超时相关的属性和两个和工作线程相关的属性：

Key	Required	Type	Default
<code>hdfs.callTimeout</code>	No	long (milliseconds)	10000
<code>hdfs.idleTimeout</code>	No	int (seconds)	0 (0 = disable)
<code>hdfs.threadsPoolSize</code>	No	int	10
<code>hdfs.rollTimerPoolSize</code>	No	int	1

`hdfs.callTimeout` 属性是 HDFS sink 等待 HDFS 返回成功（或者失败）的等待时间。如果你的 Hadoop 集群特别慢（例如开发集群或者虚拟集群），你可能需要将这个值设大些来避免错误。记住，如果你不能维持输出量速度高于输入量速度，那么你的 channel 会最终溢出。



`hdfs.idleTimeout` 属性如果设为非零，表示 Flume 在自动关闭空闲文件前等待的时长。我从来没有使用过这个属性，因为 `hdfs.fileRollInterval` 在每次滚动的时候都会处理去关闭这些文件，并且如果 `channel` 是空闲的，它是不会打开新文件的。这个选项看起来是创建了用来取代已经讨论过的大小、时间及 `event` 数量滚动机制的。你可能想要将尽可能多的数据写到一个文件中，直到没有数据可写的时候关闭文件。在这种情况下你可以使用 `hdfs.idleTimeout` 来达到这种文件滚动计划，如果你同时将 `hdfs.rollInterval`、`hdfs.rollSize` 和 `hdfs.rollCount` 都设为 0 的话。

第一个你可以调整工作线程数量的属性是 `hdfs.threadsPoolSize`，默认情况下是 10 个。这是在同一时间内可以同时写文件的最大数量。如果你使用 `event` 的头部信息来确定文件的路径，那么你可能同时会打开超过 10 个文件，但请注意当这个值设置的太大时可能会压垮 HDFS。

最后一个和工作线程数有关的属性是 `hdfs.rollTimerPoolSize`。这是处理由 `hdfs.idleTimeout` 属性定义的超时时间的工作线程数。处理关闭文件的工作量非常小，所以增加这个值（默认 1 个线程）没有太大的必要。如果你不使用基于 `hdfs.idleTimeout` 的文件滚动策略，你可以忽略这个属性，因为它根本用不到。

## Sink 组

为了消除数据处理管道中的单点故障，Flume 可以使用负载平衡或故障转移策略将 `event` 发送到不同的 sink。为了做到这一点，我们需要引入一个新的概念叫做 sink 组。sink 组是用来创建逻辑上的一组 sink。这个组的行为是由称为 sink 处理器来决定的，它决定了 `event` 的路由策略。

在你使用的 sink 不属于任何 sink 组时，这种情况下使用了默认的 sink 处理器，它只包含单个 sink。我们在第二章《Flume 快速入门》的 Hello World 示例中，使用了默认处理器。对使用单个的 sink 不需要任何特殊配置。

为了让 Flume 知道 sink 组的存在，有一个 agent 的顶级属性叫 `sinkgroups`。和 `source`、`channel`、`sink` 类似，你属性的前缀名称需要是 agent 的名称，这里叫“agent”：

```
agent.sinkgroups=sg1
```

这里我们为名叫“agent”的 agent 定义了一个 sink 组叫 sg1。

对于每个指定的 sink 组，你需要通过 sinks 属性指定它所包含的 sink，值为由逗号分隔的 sink 的名称：

```
agent.sinkgroups.sg1.sinks=k1, k2
```

这里定义了为“sg1”的 sink 组定义了 sink，由名叫 k1、k2 的 sink 组成。

通常 sink 组用于联合分层传输数据模式来绕过失败的情况。然而，他们也可以用来往不同的集群写数据，因为即使一个维护的很好的集群也需要定期检修。


## 负载均衡 Sink 组

继续前面的例子，比方说你想负载均衡 k1 和 k2 的流量。有一些额外的属性需要指定如下表所示：

Key	Type	Default
processor.type	String	load_balance
processor.selector	String (round_robin, random)	round_robin
processor.backoff	boolean	false

当你设置 processor.type 属性的值为 load\_balance 时，round robin 策略将会被使用，除非你特别通过 processor.selector 属性指定。这个值可以被设置为 round\_robin 或者 random。你还可以指定自己的负载均衡机制，在这里我们不涉及。如果你需要自己来控制负载均衡实现，你可以参照 Flume 的官方文档。

processor.backoff 属性指定了一个指数级的阻塞时间用于当重试一个 sink 抛出异常时。默认情况下是 false，这意味着 sink 抛出异常后，下次 Flume 将再次基于循环或随机选择机制进行尝试。如果设置为 true，那么对于每个失败后的等待时间将增加一倍，从一秒钟开始到大约 18 个小时的限制结束 ( $2^{16}$  秒)。

 在写这本书的时候，processor.backoff 属性在代码中默认是 false，但 Flume 官方文档中说的确是 true。自己麻烦点，指定你想要的值比依赖默认值要好。  
PS：翻译本书的时候，官方文档中已经改为 false 了

## 故障转移 Sink 组

如果你想要尝试使用一个 `sink`，然后在出现故障的时候尝试另一个，那么你可以设置 `processor.type` 为 `failover`。接下来你需要设置额外的属性来指定 `sink` 顺序，通过设置 `processor.priority` 属性附加 `sink` 的名称：

Key	Type	Default
<code>processor.type</code>	String	<code>failover</code>
<code>processor.priority.NAME</code>	int	
<code>processor.maxpenalty</code>	int (milliseconds)	30000

让我们看看这个示例：

```
agent.sinkgroups.sg1.sinks=k1, k2, k3
agent.sinkgroups.sg1.processor.type=failover
agent.sinkgroups.sg1.processor.priority.k1=10
agent.sinkgroups.sg1.processor.priority.k2=20
agent.sinkgroups.sg1.processor.priority.k3=20
```

优先级数字越小优先级越高，对于同样的数字，则是随机的顺序。你可以使用任何对你有意义的数字体系（如 1、5、10，怎样都行）。在这个例子中，`k1 sink` 将会第一个使用，如果抛出了异常，`k2`、`k3` 接下来将会被尝试。如果 `k3` 首先被选中并且失败了，`k2` 会继续尝试。如果 `sink` 组中的所有 `sink` 都失败了，`channel` 中的事务将会回滚。

最后，`processor.maxPenalty` 属性为 `sink` 组中失败的 `sink` 被关小黑屋的时限设置了上限值。在第一次失败后，在再次使用前它将会被禁用一秒。每次失败会加倍这个等待时间直到达到 `processor.maxPenalty` 设定的值。

## 小结

在这一章中，我们详细讲述了 Flume 输出数据到 HDFS 中的 HDFS `sink`。我们讲述了如何让 Flume 能够基于时间或者 `event` 的 `header` 的内容分离数据到不同的 HDFS 目录中去。下面几种文件滚动策略同样有所涉及：

- Time rotation
- Event count rotation
- Size rotation
- Rotation on idle only

我们讨论了压缩技术作为一种减少 HDFS 存储需求的方法，在适当的时候应当被使用。除了减少存储空间，通过读取压缩文件并在内存中解压缩它同样提供了更快的速度相比于读取一个未压缩的文件。这将使运行在这个数据上的 **MapReduce** 工作性能得到提升。在决定使用何种压缩算法的时候分割压缩文件同样作为一个因素被谈及。

Event 序列化被作为 Flume event 转换为外部存储格式的原理来介绍，包含如下内容：

- Text (只有 body)
- Text and Headers (header 和 body)
- Avro 序列化 (带可选的压缩功能)

下面是各种各样的文件格式：

- 序列化文件(Hadoop key/value 文件)
- 数据流(未压缩的数据文件，如 Avro 容器)
- 压缩数据流

最后，我们讨论了 sink 组，通过使用负载均衡或者故障转移策略路由 event 到不同 source 的一种方法，可以在将数据发送到目的地的时候消除单点故障。

在下一章，我们将要讨论各种各样的输入机制（Sources），它们会提供数据到你配置的 channel 中去（在第三章《Channels》中讨论的）。

# 5

## Sources 和 Channel 选择器

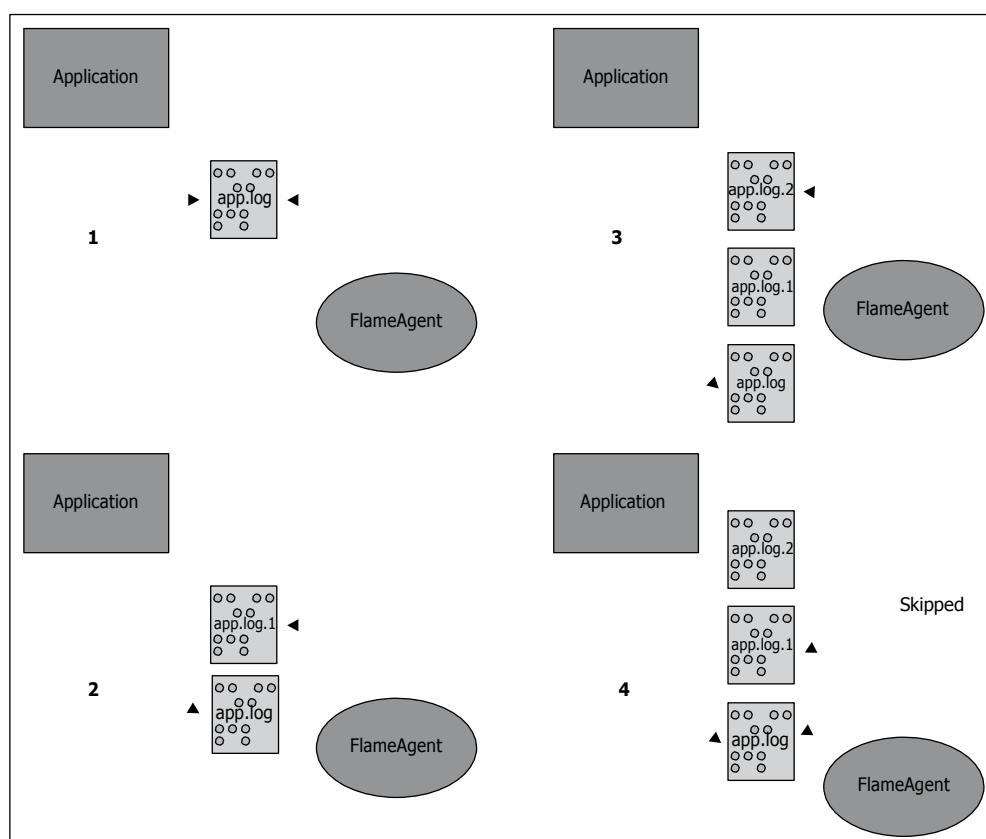
现在我们已经讲过了 channel 和 sink，我们在这里将涉及一些将数据加载到 Flume 中的常用方法。正如在第一章“概览”中讨论的，source 是将数据加载到 Flume 中的入口。在 Flume 发行版本中有很多自带的 source，同样也有很多开源版本的 source 可以供你使用。正如绝大多数开源软件一样，如果你无法找到合适的 source，你可以通过继承 `org.apache.flume.source.AbstractSource` 类来实现你自己的 source。因为本书的重点是摄取日志文件到 Hadoop 中，我们这里只会讲述一些更适合该场景的 source 组件。

### 使用 tail 时的一些问题

如果你以前使用过 Flume 0.9 release 版本，你会注意到 TailSource 不再是 Flume 的一部分。TailSource 提供了一个命令 `tail` ([http://en.wikipedia.org/wiki/Tail\\_\(Unix\)](http://en.wikipedia.org/wiki/Tail_(Unix))) 读取系统中的任何文件，并且将每一行作为数据创建一个 Flume event。许多人已经使用文件系统作为程序创建数据（例如：log4j）和移动这些文件的物理机制（例如：syslog）之间的转换点。所以，TailSource 可以完美的替代 syslog 传输，而不需要修改应用程序创建的数据。

同样在 channel 和 sink 的例子中，event 都是在传输系统中的 channel 中进行放入和取出的。当你在 `tail` 一个文件的时候，是没有办法真正参与到一个事务中去的。如果未能成功写入 channel 中或者 channel 满了的时候（比失败可能性更大），这些数据是无法以回滚的形式被“回放”回去的。

此外，如果数据写入文件的速度超过了 Flume 读取文件的速度，也有可能造成丢失一个到多个直接写入的数据文件。比如说你正在 `tail` 操作 `/var/log/app.log` 文件。当那个文件到了一定的大小，那个文件将会被旋转、重命名为 `/var/log/app.log.1`，同时开始写另一个新的文件 `/var/log/app.log`。假设你有一个良好的检测机制并且你程序产生的日志比通常多了很多。当另一个滚动发生将 `/var/log/app.log` 变为 `/var/log/app.log.1` 时，Flume 也许仍然在读取已经滚动了的文件 (`/var/log/app.log.1`)。Flume 正在读的文件被命名为 `/var/log/app.log.2`。当 Flume 读完了这个文件，它将会转移到它自认为的下一个文件 `var/log/app.log`，因此跳过了 `/var/log/app.log.1` 文件中的数据。这种类型的数据丢失是完全无法被察觉到的，如果可能的话要尽力避免。



由于这些因素的存在，决定了 `tail` 功能在重构时从 Flume 中移除了。一些应用到 `TailSource` 的场景已经被删除了，但是应该注意到没有一种场景可以消除这种场景下数据丢失的可能性。

## Exec source

Exec source 提供了一种在 Flume 外运行命令然后将内容作为 event 输出到 Flume 中。要使用 exec source，设置 type 属性的值为 exec：

```
agent.sources.s1.type=exec
```

所有 Flume 中的 source 都需要指定要写数据的 channel 列表，通过 channels（复数）属性来指定。值为用空格分隔的 channel 名称列表：

```
agent.sources.s1.channels=c1
```

其他需要的参数只有 `command` 属性，该属性告诉了 Flume 应该将什么样的命令传给操作系统。比如：

```
agent.sources=s1
agent.sources.s1.channels=c1
agent.sources.s1.type=exec
agent.sources.s1.command=tail -F /var/log/app.log
```

这里我为 agent 配置了一个 source 名为 s1。这个 source 是一个 exec source，将会对 `/var/log/app.log` 文件执行 `tail` 命令并遵循任何外部程序作用在该文件上的旋转规则。所有的 event 将会被写到 c1 中。这是一个在 Flume1.x 缺乏 TailSource 情况下的一个解决方案。



你是否真的应该使用 `tail -f` 命令结合 exec source？当 Flume agent 关闭或重启的时候 `tail` 进程可能不会 100% 的关闭。这会留下孤立的永远不会退出的 `tail` 进程。`Tail -F` 命令定义是永远没有结尾。即使你删除了已经 `tail` 出来的文件（至少在 Linux 中是这样），运行的 `tail` 进程不确定会在什么时候又让这个文件重新打开。这让该文件占用的空间一直无法回收利用直到 `tail` 进程退出——而退出是不可能的。我想你已经开始明白 Flume 开发者们为什么不喜欢 `tail` 文件了。

如果你走了这条路，请确保定期的扫描进程列表来搜寻“`tail -F`”命令，其父 PID 为 1。它们实际上是死掉的进程，需要手动的进行 kill 掉。

这里是使用 `exec source` 时可能会用到的一些属性：

Key	Required	Type	Default
type	Yes	String	exec
channels	Yes	String	Space-separated list of channels
command	Yes	String	
restart	No	boolean	false
restartThrottle	No	long (milliseconds)	10000 milliseconds
logStdErr	No	boolean	false
batchSize	No	int	20

不是每个命令都可以持续运行，不管是因为它失败了（如 `channel` 写满时）或者这个命令设计为立即退出的形式。在本例中，我们想通过 Linux “`uptime`” 命令记录系统负载命令，打印出一些系统信息并发送到 `stdout` 然后退出：

```
agent.sources.s1.command=uptime
```

这个命令会立即退出，所以你可以使用 `restart` 和 `restartThrottle` 属性来让它定期的执行：

```
agent.sources.s1.command=uptime
agent.sources.s1.restart=true
agent.sources.s1.restartThrottle=60000
```

这会在每分钟产生一个 `event`。在这个 `tail` 例子中，如果 `channel` 的填充导致了 `exec source` 的失败，你可以使用这些属性来重启 `exec source`。在这个例子中，设置 `restart` 属性将会在当前文件开始时启动 `tail` 文件，这就会产生重复数据。因为文件可能会滚动到 `Flume` 的范围之外你也许会丢失一些数据，这取决于你设置 `restartThrottle` 属性值的大小。此外，`channel` 也许仍然无法接受数据，这样仍然会导致 `source` 的失败。设置这个值过小意味着给予 `channel` 消化这些数据的时间更少，并且不像我们见过的一些 `sink`，没有一个指数级的黑名单选项。

有时候命令行将你想要捕获的内容输出到 `StdErr` 中。如果你想要同意包含这些错误数据，设置 `logStdErr` 属性的值为 `true` 即可。没有属性去关闭 `StdOut` 的内容（但你可以用我们在第六章讲到的“拦截器、ETL 和 `Routing`”将它们过滤掉）。



最后，你可以通过修改 `batchSize` 属性来指定每个事务中处理的 `event` 数量。如果你的输入内容很大并且你发现你往 `channel` 中写数据不够快，你也许需要将这个值设置得比默认 20 个更大一点。使用大一点的批处理大小将会减少每个 `event` 的平均事务开销。用不同的值来测试并且监控 `channel` 的放入速度是唯一可以得到合适值的方法。

## Spooling directory source

为了努力避免在 `tailing` 文件中所有假设的固有问题，一种新的 `source` 被设计用来跟踪哪些文件已经被转换为 Flume `event` 以及哪些文件仍然需要处理。`Spooling directory source` 被给予了一个目录来监视新文件的产生。它假设复制到该目录下面的文件是完结的；另一方面，`source` 可以尝试发送部分文件。它同意假设文件名称永远不会改变；另一方面，`source` 将会在重启后丢失文件的位置（哪些文件已经被发送过了、哪些还没有）。文件名的规则可以参考 `log4j`，请用 `DailyRollingFileAppender` 而非 `RollingFileAppender`，然而，当前打开的文件需要被写到一个目录中，在关闭后再复制到 `spool` 目录中。没有哪个 `log4j` 有这个功能。

这就是说，如果你在你的环境中使用 `Linux logrotate` 程序，这会很有趣。你可用使用 `postrotate` 脚本移动完结的文件到指定的文件夹中。

记住，你将需要一个单独的进程来清理 `spool` 目录中的所有被 Flume 标记为已发送的旧文件否则你的磁盘最终会被塞满。

要创建一个 `spooling directory source`，设置 `type` 属性值为 `spooldir`。你必须通过 `spoolDir` 属性设置你需要监控的目录：

```
agent.sources=s1
agent.sources.channels=c1
agent.sources.s1.type=spooldir
agent.sources.s1.spoolDir=/path/to/files
```

这里是 `spooling directory source` 所有需要用到的属性列表：

Key	Required	Type	Default
<code>type</code>	Yes	String	<code>spoolDir</code>
<code>channels</code>	Yes	String	Space-separated list of channels
<code>spoolDir</code>	Yes	String	Path to directory to spool
<code>fileSuffix</code>	No	String	<code>.COMPLETED</code>
<code>fileHeader</code>	No	boolean	<code>false</code>
<code>fileHeaderKey</code>	No	String	<code>file</code>
<code>batchSize</code>	No	int	10
<code>bufferMaxLines</code>	No	int	100
<code>maxBufferLineLength</code>	No	int	5000

当一个文件被传输完毕后它将会被重命名为一个以 “`.COMPLETED`” 结尾的文件，除非你修改 `fileSuffix` 属性，例如：

```
agent.source.sl.fileSuffix=.DONE
```

如果你想在每个 `event` 中带入文件的绝对路径，设置 `fileHeader` 属性为 `true`。这会在 `event` 的 `header` 中创建一个 `key` 为 “`file`” 的键值对，或者你可以设置 `fileHeaderKey` 属性来改变 `key` 值，例如：

```
agent.source.sl.fileHeader=true
agent.source.sl.fileHeaderKey=sourceFile
```

这会在 `header` 中添加 `{sourceFile=/path/to/files/foo.1234.log}` 如果这个 `event` 是从 `/path/to/files/foo.1234.log` 文件中读取的。`BatchSize` 属性允许你修改每个事务往 `channel` 中每次写 `event` 的数量。增加这个值可能会提供更好的吞吐量，代价是更大的事务（也可能是更大的回滚）。`BufferMaxLines` 属性是用来设置内存缓冲区大小的，用来在读取文件时将它和 `maxBufferLineLength` 属性的值相乘。如果你的数据非常短，你可以考虑在减少 `maxBufferLineLength` 大小的同时增加 `bufferMaxLines` 的大小。在这个例子中，它将产生更好的吞吐量而不需要增加你的内存开销。这就是说，如果你的 `event` 大小大于 5000 个字符，你需要将 `maxBufferLineLength` 设置的更大些。最后，你需要确保无论你采用何种机制来往你的 `spooling` 目录创建新的文件，文件名都必须是独特的，比如添加一个时间戳（或者其他更多的东西）。重用文件名会让 `source` 变得混乱，从而可能让你的文件被忽略掉。

不管怎么说，请记住重启和错误将会导致在 `spooling` 目录中任何没有被标记为已完成的文件创建重复的 `event`。

## Syslog sources

Syslog 已经存在了几十年，并且经常被用作操作系统级别的日志捕获和移动的机制。在很多方面它们和 Flume 提供的功能有很多重合之处。甚至有一个 Hadoop rsyslog 模块，是 syslog 的现代变体之一

([http://www.rsyslog.com/doc/rsyslog\\_conf\\_modules.html/omhdfs.html](http://www.rsyslog.com/doc/rsyslog_conf_modules.html/omhdfs.html))。一般来说，我不喜欢版本独立又相互之间有关联的技术。如果你使用 Hadoop 集成的 rsyslog，在你需要更新 Hadoop 集群到一个新版本时你同时需要更新编译到 rsyslog 中的 Hadoop 版本。如果你有大量的服务器或者环境按理说这会很困难。在 Hadoop 社区中 Hadoop 向下兼容协议是被积极应对的，但目前来说不是常态。我们将会在第七章“Flume 监控”讨论分层数据流时细说。

Syslog 有一个古老的 UDP 传输协议以及新的 TCP 协议，TCP 协议用于处理数据大于一个 UDP 数据包可以传输大小（约 64 k）的数据，以及处理网络相关的拥堵事件，此时可能需要重新传输数据。

最后，还有一些在 syslog source 中没有说明的属性允许为消息额外添加不符合 RPC 标准的正则表达式。我不会讨论这些额外的设置，但你在频繁遇到解析错误时应该知道他们的存在。在这个例子中，看一下

`org.apache.flume.source.SyslogUtils` 的源码实现来发现根本原因。

更多关于 syslog 条款的细节（比如机制是什么）和标准格式可以在 **RFC 3164**

（<http://tools.ietf.org/html/rfc3164>）和 **RFC 5424**

（<http://tools.ietf.org/html/rfc5424>）中找到。

## Syslog UDP source

当你从服务器的本地 syslog 进程接收数据，且数据足够小时（小于大约 64k），UDP 版本的 syslog 通常是可以安全使用的。



这个 source 的实现选择了 2500 个字节的最大有效荷载大小不管你的网络最大荷载是多少。所以，如果你的荷载会比这个大，请使用 TCP 源来代替。

要创建 syslog UDP source, 设置 `type` 属性为 `syslogudp`。你必须通过 `port` 属性设置要监听的端口号。可选属性 `host` 指定绑定的地址。如果没有 `host` 被指定, 服务器上的所有 IP 都会被使用-功能和指定为 `0.0.0.0` 一样。在这个例子中, 我们会监听本地 5140 端口的 UDP 连接:

```
agent.sources=s1
agent.sources.channels=c1
agent.sources.s1.type=syslogudp
agent.sources.s1.host=localhost
agent.sources.s1.port=5140
```

如果你想要 syslog 跟踪一个 tail 文件, 你可以在 syslog 中添加一行如下的配置文件:

```
*.err;*.alert;*.crit;*.emerg;kern.* @localhost:5140
```

这将把所有的错误、提醒、重要、紧急优先级和所有优先级的内核消息发送到 Flume 的 source。@符号指定了应该使用 UDP 协议。

这里是 syslog UDP source 所有需要用到的属性列表:

Key	Required	Type	Default
type	Yes	String	syslogudp
channels	Yes	String	Space-separated list of channels
port	Yes	int	
host	No	String	0.0.0.0

由 syslog UDP source 创建的 Flume header 内容列表如下:

Header key	Description
Facility	The syslog facility. See the syslog documentation.
Priority	The syslog priority. See the syslog documentation.
timestamp	The time of the syslog event translated into an epoch timestamp. Omitted if not parsed from one of the standard RFC formats.
hostname	The parsed hostname in the syslog message. Omitted if not parsed.
flume.syslog.status	There was a problem parsing the syslog message's headers. Set to <code>Invalid</code> if the payload didn't conform to the RFCs. Set to <code>Incomplete</code> if the message was longer than the <code>eventSize</code> value (for UDP this is set internally to 2500 bytes). Omitted if everything is fine.

## Syslog TCP source

如前所述，Syslog TCP source 通过 TCP 协议提供了一个终端消息，允许更大的有效荷载大小和 TCP 重试机制，可以被用于任何服务器之间的可靠通信。

要创建一个 syslog TCP source，设置 type 属性值为 syslogtcp。你同样必须设置需要监听的地址和端口号：

```
agent.sources=s1
agent.sources.s1.type=syslogtcp
agent.sources.s1.host=0.0.0.0
agent.sources.s1.port=12345
```

如果你的 syslog 实现支持 TCP 协议的 syslog，配置通常是相同的除了两个@符号是用来表示 TCP 传输。这里是一个用 TCP 传输的相同的例子，我用来转发数据到不同机器上的 Flume agent，该机器名叫“flume-1”：

```
*.err;*.alert;*.crit;*.emerg;kern.*      @@flume-1:12345
```

这里是 syslog TCP source 的一些可选属性：

Key	Required	Type	Default
type	Yes	String	syslogtcp
channels	Yes	String	Space-separated list of channels
port	Yes	int	
host	No	String	0.0.0.0
eventSize	No	int (bytes)	2500 bytes

由 syslog TCP source 创建的 Flume header 内容列表如下：

Header key	Description
Facility	The syslog facility. See the syslog documentation.
Priority	The syslog priority. See the syslog documentation.
timestamp	The time of the syslog event translated into an epoch timestamp. Omitted if not parsed from one of the standard RFC formats.
hostname	The parsed hostname in the syslog message. Omitted if not parsed.
flume.syslog.status	There was a problem parsing the syslog message's headers. Set to <i>Invalid</i> if the payload didn't conform to the RFCs. Set to <i>Incomplete</i> if the message was longer than the configured eventSize. Omitted if everything is fine.

## 多端口的 syslog TCP source

多端口 syslog TCP source 的功能和 syslog TCP source 几乎相同，但它可以监听多个端口的输入。您可能需要使用这种能力因为你可能无法改变 syslog 在其转发规则中使用的端口号（因为它可能不是你的服务器）。更有可能你将使用这种方式来通过一个 source 读取多种格式的数据写到多个 channel 中去。我们待会会在“Channel 选择器”章节中进行讨论。

要配置这种 source，设置 type 属性值为 multiport\_syslogtcp：

```
agent.sources.s1.type=multiport_syslogtcp
```

就像其他的 syslog source 一样，你需要指定端口号，但这种情况下它是一个空格分隔的端口号列表。只要你有有一个端口要指定你就可以使用这个。

这个属性就是“ports”（port 的复数形式）：

```
agent.sources.s1.type=multiport_syslogtcp
agent.sources.s1.channels=c1
agent.sources.s1.ports=33333 44444
agent.sources.s1.host=0.0.0.0
```

上面配置了多端口的 syslog TCP source，名叫“s1”，该 source 监听了所有连接到端口 33333 和 44444 的数据，并且将他们传输到 channel “c1”中。

为了能够知道每个 event 来自于哪个端口，你可以设置可选的 portHeader 属性为一个具体的值，这个值将作为 event header 中存储端口号的 key，value 则为具体的端口号。如果我添加来这个属性到配置中：

```
agent.sources.s1.portHeader=port
```

然后所有来自于 33333 端口的 event 将在 header 中有一个键值对{"port"="33333"}。正如你在第四章“Sink 和 Sink 处理器”中所看到的，你现在可以使用这个值（任何 header 中的值都可以使用）作为你 HDFS sink 文件路径的一部分，如下所示：

```
agent.sinks.k1.hdfs.path=/logs/{hostname}/{port}/{Y}/{m}/{D}/{H}
```

这里是多端口的 syslog TCP source 的所有属性列表：

Key	Required	Type	Default
type	Yes	String	syslogtcp
channels	Yes	String	Space-separated list of channels
ports	Yes	int	Space-separated list of port numbers

Key	Required	Type	Default
host	No	String	0.0.0.0
eventSize	No	int (bytes)	2500 bytes
portHeader	No	String	
batchSize	No	int	100
readBufferSize	No	int (bytes)	1024
numProcessors	No	int	Automatically detected
charset.default	No	String	UTF-8
charset.port.PORT#	No	String	

这种 TCP syslog source 较标准的 TCP syslog source 有一些额外的可调整的选项，你可能需要去调整他们。第一个是 `batchSize` 属性。这是每个和 `channel` 通讯的事务所处理的 `event` 数量。同样还有 `readBufferSize` 属性，它通过内置的 `Mina` 库指定了内部缓冲区大小。最后，`numProcessors` 属性用来指定 `Mina` 的工作线程池大小。在你修改这些参数前，你应该让你自己先熟悉下 `Mina` (<http://mina.apache.org/>) 并且看下这个 `source` 的源码再来修改默认值。

最后，你可以指定默认编码和具体端口号对应的编码用来在字符串和 `byte` 数组之间进行转换：

```
agent.sources.s1.charset.default=UTF-16
agent.sources.s1.charset.port.33333=UTF-8
```

这个例子展示了除了 33333 端口单独用 UTF-8 来编码外，其他所有的端口将使用 UTF-16 来编码。

由多端口的 syslog TCP source 创建的 Flume header 内容列表如下：

Header key	Description
Facility	The syslog facility. See the syslog documentation.
Priority	The syslog priority. See the syslog documentation.
timestamp	The time of the syslog event translated into an epoch timestamp. Omitted if not parsed from one of the standard RFC formats.
hostname	The parsed hostname in the syslog message. Omitted if not parsed.
flume.syslog.status	There was a problem parsing the syslog message's headers. Set to <code>Invalid</code> if the payload didn't conform to the RFCs. Set to <code>Incomplete</code> if the message was longer than the configured <code>eventSize</code> . Omitted if everything is fine.

## Channel 选择器

正如我们在第一章“概览”中讨论的，一个 source 可以将数据写入一个或多个 channel 中。这就是为什么这个属性是复数的原因（channels 替代了 channel）。有两种方式可以处理多路 channel。Event 可以写到所有的 channel 中去或者依据 Flume event header 中的值来决定往哪个 channel 中写。在 Flume 中这种内部机制被称为 channel 选择器。

对于任何 channel 可以通过 selector.type 属性来指定选择器类型。所有选择器指定属性的前缀格式都通常是 source 属性的前缀；agent 的名字，关键字“sources”，source 的名字：

```
agent.sources.s1.selector.type=replicating
```

## 复制

默认情况下，如果你不为 source 指定选择器，replicating 将会成为默认值。复制选择器将 event 写到所有在 source 的 channel 列表中 channel 中：

```
agent.sources.s1.channels=c1 c2 c3
agent.sources.s1.selector.type=replicating
```

在这个例子中，所有的 event 将会被写到三个 channel 中，分别为 c1、c2、c3。

这个选择器中有一个可选的属性 optional。它是一个空格分隔的可选 channel 列表。如下方这样：

```
agent.sources.s1.channels=c1 c2 c3
agent.sources.s1.selector.type=replicating
agent.sources.s1.selector.optional=c2 c3
```

任何往 c2 或者 c3 中写失败都不会导致事务的失败，只要往 c1 中写成功了事务都会提交。在更前面的例子中没有可选的 channel，任何一个 channel 的失败将会导致所有事务的回滚。

## 多路复用

如果你想将不同的 event 发送到不同的 channel 中，你可以通过 selector.type 设置值为 multiplexing 使用多路复用选择器。同样的你需要设置 selector.header 属性告诉选择器该使用哪个 header 值：

```
agent.sources.s1.selector.type=multiplexing
agent.sources.s1.selector.header=port
```



让我们来假定我们使用多端口的 syslog TCP source 来监听 4 个端口：11111、22222、33333 和 44444，同时设置 portHeader 为 “port”。来看下这个配置：

```
agent.sources.s1.selector.default=c2
agent.sources.s1.selector.mapping.11111=c1 c2
agent.sources.s1.selector.mapping.44444=c2
agent.sources.s1.selector.optional.44444=c3
```

这将会把 22222 和 33333 端口的数据传输到 c2 channel 中去。11111 端口中的数据将会传输到 c1 和 c2 channel 中去。任何通道的失败将会导致没有任何数据传输到另外的通道中。端口 44444 的数据将会被传输到 c2 和 c3 中去。然而当往 c3 中写数据失败时仍然会提交 c2（该 event 在 c3 上不会再被重试）。

## 小结

在这一章中我们深入学习了你往 Flume 传输数据时可能会用到的各种各样的 source，包括如下几个：

- Exec source
- Syslog sources（UDP，TCP，和多端口的 TCP source）

我们讨论了复制 Flume 0.9 中旧的 TailSource 的功能以及使用 tail 语法时通常会遇到的问题。

我们同样讨论了 channel 选择器以及如何发送 event 到一个或多个 channel 中去。确切的说有如下两个选择器：

- 复制 channel 选择器
- 多路复用 channel 选择器

可选的 channel 同样被讨论用来作为在存在多个 channel 时，在事务中该可选 channel 允许失败而不必进行事务回滚的存在。

在下一章中，我们将介绍拦截器，它允许动态检测和修改 event 数据。配合 channel 选择器来使用，拦截器提供了最后一个组件来在 Flume 中创建复杂的数据流。



# 6

## 拦截器、ETL 和 Routing

最后一个在你数据处理管道中的功能是在传输中检查和转换 **event** 的功能。使用拦截器可以完成这个功能。拦截器，正如我们在第一章《概览》中所讨论的，可以在 **source** 之后或者 **sink** 之前加入。

### 拦截器

一个拦截器的功能可以被概括为这个方法：

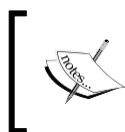
```
public Event intercept(Event event);
```

它传入一个 **event** 然后返回一个 **event**。它可以不做任何事情；也就是说没有任何改变的 **event** 将被返回。通常来说，它用一些有用的方式来改变 **event** 的内容。如果返回 **null** 值，这个 **event** 就表示被丢弃了。

要往 **source** 上添加拦截器，只需要添加 **interceptors** 属性到该 **source** 上。例如：

```
agent.sources.s1.interceptors=i1 i2 i3
```

这里在名为 **s1** 的 **source** 上定义了三个拦截器，**i1**、**i2** 和 **i3**



拦截器按他们的排列顺序运行。在前面这个例子中，**i2** 将接收 **i1** 的输出。**i3** 将接收 **i2** 的输出。最后 **channel** 选择器接收 **i3** 的输出。

现在我们已经定义了拦截器的名字，我们需要通过如下方式指定他们的类型：

```
agent.sources.s1.interceptors.i1.type=TYPE1
agent.sources.s1.interceptors.i1.additionalProperty1=VALUE
agent.sources.s1.interceptors.i2.type=TYPE2
agent.sources.s1.interceptors.i3.type=TYPE3
```

让我们看看它们中的一些拦截器，由 Flume 自带的，来看看有什么更好的方式来配置他们。

## Timestamp

Timestamp 拦截器，正如它名字暗示的，在 Flume event 的 header 中添加一个 timestamp 的 key，如果这个 key 值还不存在的话。要使用它，设置 type 属性值为 timestamp 即可。

如果 event 的 header 中已经包含了一个 timestamp 的 key，它将用当前时间来覆盖这个值，除非你设置 preserveExisting 属性的值为 true 来保护原有的值。

这里是 Timestamp 拦截器的所有属性列表：

Key	Required	Type	Default
type	Yes	String	timestamp
preserveExisting	No	Boolean	false

如果我们只想在 header 中不存在 timestamp 时添加该属性，那么这里有一个对 source 来说看起来相对完整的拦截器配置：

```
agent.sources.s1.interceptors=i1
agent.sources.s1.interceptors.i1.type=timestamp
agent.sources.s1.interceptors.i1.preserveExisting=true
```

回顾下第四章《Sink 和 Sink 处理器》中 HDFS Sink 的路径，李勇了 event 的时间：

```
agent.sinks.k1.hdfs.path=/logs/apache/%Y/%m/%D/%H
```

Header 中的 timestamp 决定了这个路径的最终值。如果它缺失了，可以确定 Flume 将不知道在何处创建文件，从而你将无法获得你想要的结果。

## Host

和 Timestamp 拦截器基本类似，Host 拦截器将在当前 Flume agent 的 event 的 header 中添加 IP 地址。要使用它，设置 type 属性为 host 即可。

```
agent.sources.s1.interceptors=i1
agent.sources.s1.interceptors.i1.type=host
```

除非使用 hostHeader 属性指定其他值，否则 header 中的 key 值将为 host。和之前的 Timestamp 拦截器类似，默认情况下会覆盖 header 中已有的值，除非你设置 preserveExisting 的值为 true。最后，如果你想要使用反向 DNS 查找得到 hostname 来单体 IP 作为值，设置 useIP 属性为 false 即可。记住反向查找将增加数据流的处理时间。

这里是 Host 拦截器的所有属性列表：

Key	Required	Type	Default
type	Yes	String	host
hostHeader	No	String	host
preserveExisting	No	Boolean	false
useIP	No	Boolean	true

如果我们只想在 header 中添加一个 relayHost 的 key，通过 DNS 反向查找得到的 hostname 作为 value 值，那么这里有一个对 source 来说看起来相对完整的拦截器配置：

```
agent.sources.s1.interceptors=i1
agent.sources.s1.interceptors.i1.type=host
agent.sources.s1.interceptors.i1.hostHeader=relayHost
agent.sources.s1.interceptors.i1.useIP=false
```

如果你想要在你的数据流中记录 event 的路径，这个拦截器还是很有用的。相比于 event 的路径，多数情况下你可能对 event 的来源更感兴趣，这也正是我使用它的原因。

## Static

Static 拦截器是用来往经过的所有 Flume event 的 header 中插入单个任何形式的 key/value 的。如果需要插入多个 key/value 值，你只需要添加更多的 static 拦截器即可。不像我们已经见过的其他拦截器，默认情况下该拦截器会保留已经存在的 key 的值。就像以往的一样，我建议你指定具体的值而非依赖默认值。我不知道为什么 key 和 value 属性不是必须的，因为默认值通常没什么用处。

这里是 Static 拦截器的所有属性列表：

Key	Required	Type	Default
type	Yes	String	static
key	No	String	key
value	No	String	value
preserveExisting	No	Boolean	true

最后，让我们看一下假设在先前不存在这两个值的情况下插入两个新的 header 值的例子：

```
agent.sources.s1.interceptors=pos env
agent.sources.s1.interceptors.pos.type=static
agent.sources.s1.interceptors.pos.key=pointOfSale
agent.sources.s1.interceptors.pos.value=US
agent.sources.s1.interceptors.env.type=static
agent.sources.s1.interceptors.env.key=environment
agent.sources.s1.interceptors.env.value=staging
```

## 正则表达式过滤拦截器

如果你想要依据 body 的内容来过滤 event，正则表达式过滤拦截器对你来说很有用。根据你提供的正则表达式，它将会筛选出符合条件的 event 或者只保留符合条件的 event。让我们从设置 type 属性值为 regex\_filter 开始。你想匹配的表达式需要使用 Java 格式的正则表达式来书写。具体的使用细节可以看看这个 Javadoc 文档：

<http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html>.

正则表达式字符串是在 regex 属性中设置的。最后，你需要通过设置 excludeEvents 属性为 true 告诉拦截器你想要排除匹配正则表达式的 event。默认值（false）说明你只想要可以匹配正则表达式的 event。

这里是正则表达式过滤拦截器的所有属性列表：

Key	Required	Type	Default
type	Yes	String	regex_filter
regex	No	String	.*
excludeEvents	No	Boolean	false

在这个例子中，任何包含“NullPointerException”字符串的 **event** 都会被丢弃掉：

```
agent.sources.s1.interceptors=npe
agent.sources.s1.interceptors.npe.type=regex_filter
agent.sources.s1.interceptors.npe.regex=NullPointerException
agent.sources.s1.interceptors.npe.excludeEvents=true
```

## 正则表达式提取拦截器

有时候你想要提取 **event body** 中的部分二进制数据到 **Flume header** 中，这样你就可以通过 **channel** 选择区来路由 **event** 了。你可以使用正则表达式提取拦截器来完成这个操作。我们通过设置 **type** 属性值为 **regex\_extractor** 开始：

```
agent.sources.s1.interceptors=e1
agent.sources.s1.interceptors.e1.type=regex_extractor
```

像正则表达式过滤拦截器一样，正则表达式提取拦截器也使用 **Java** 格式的正则表达式来书写。为了提取一到多个片段，你需要从设置 **regex** 属性值为一组匹配的括号开始。

让我们假设我们想要查找出 **event** 中的错误数量，格式为“**Error:N**”（**N** 为数字）：

```
agent.sources.s1.interceptors=e1
agent.sources.s1.interceptors.e1.type=regex_extractor
agent.sources.s1.interceptors.e1.regex=Error:\\s(\\d+)
```

正如你看到的我把捕获数字的正则放在括号中，可以包含一到多个数字。现在我可以匹配我想要的表达式，我需要告诉 **Flume** 我应该用这个表达式做什么。这里我们需要介绍下 **serializers**，它提供了一种可插拔的机制去解释每个表达式。在这个例子中我只需要一个匹配值，所以我用空格分隔的序列化列表只有一个值：

```
agent.sources.s1.interceptors=e1
agent.sources.s1.interceptors.e1.type=regex_extractor
agent.sources.s1.interceptors.e1.regex=Error:\\s(\\d+)
agent.sources.s1.interceptors.e1.serializers=ser1
agent.sources.s1.interceptors.e1.serializers.ser1.type=default
agent.sources.s1.interceptors.e1.serializers.ser1.name=error_no
```

**name** 属性指定了 **event** 的 **key** 值，而 **value** 的内容则用来匹配正则表达式。**Type** 属性的值为 **default**（如果没有指定默认也是 **default**）是一个简单的直接通过的序列化器。比如下面的 **event body**：

```
NullPointerException: A problem occurred. Error: 123. TxnID: 5X2T9E.
```

下面的 **header** 将会被添加到 **event** 中:

```
{ "error_no": "123" }
```

如果我想把 TxnID 的值添加到 **header** 中, 我仅仅需要增加另一组正则匹配和序列化器:

```
agent.sources.s1.interceptors=e1
agent.sources.s1.interceptors.e1.type=regex_extractor
agent.sources.s1.interceptors.e1.regex=Error:\\s(\\d+).*TxnID:\\s(\\w+)
agent.sources.s1.interceptors.e1.serializers=ser1 ser2
agent.sources.s1.interceptors.e1.serializers.ser1.type=default
agent.sources.s1.interceptors.e1.serializers.ser1.name=error_no
agent.sources.s1.interceptors.e1.serializers.ser2.type=default
agent.sources.s1.interceptors.e1.serializers.ser2.name=txnid
```

对于前面的输入内容, 将会加上如下的 **header** 值:

```
{ "error_no": "123", "txnid": "5x2T9E" }
```

然而, 如果内容颠倒了, 像这样:

```
NullPointerException: A problem occurred. TxnID: 5X2T9E. Error: 123.
```

我最终只会得到 TxnID 这一个 **header**。一个处理这种情形更好的方式就是使用多重拦截器这样命令就不会有问题了:

```
agent.sources.s1.interceptors=e1 e2
agent.sources.s1.interceptors.e1.type=regex_extractor
agent.sources.s1.interceptors.e1.regex=Error:\\s(\\d+)
agent.sources.s1.interceptors.e1.serializers=ser1
agent.sources.s1.interceptors.e1.serializers.ser1.type=default
agent.sources.s1.interceptors.e1.serializers.ser1.name=error_no
agent.sources.s1.interceptors.e2.type=regex_extractor
agent.sources.s1.interceptors.e2.regex=TxnID:\\s(\\w+)
agent.sources.s1.interceptors.e2.serializers=ser1
agent.sources.s1.interceptors.e2.serializers.ser1.type=default
agent.sources.s1.interceptors.e2.serializers.ser1.name=txnid
```

除了直接通过的 **default** 序列化器, Flume 系统自带的另外一个序列化实现就是需要指定全局限定类名的 `org.apache.flume.`

`interceptor.RegexExtractorInterceptorMillisSerializer`。这个序列化器用来将时间转化回毫秒的。你需要基于 `org.joda.time.format.DateTimeFormat` 正则来指定对应的正则表达式属性。



比如，我们假设你正在提取 Apache 网站服务器的日志。例如：

```
192.168.1.42 - - [29/Mar/2013:15:27:09 -0600] "GET
/index.html HTTP/1.1" 200 1037
```

对这个数据完整的正则表达式可能像这样（以 Java 字符串的形式，带有反斜杠和用反斜杠转义的引号）：

```
^([\\d.]+) \\S+ \\S+ \\[([\\w:/]+\\s[+\\-]\\d{4})\\]
\\"(.+?)\\" (\\d{3}) (\\d+)
```

这个时间正则表达式完全符合 `org.joda.time.format.DateTimeFormat` 正则的形式：

```
yyyy/MMM/dd:HH:mm:ss Z
```

这会让我们们的配置文件变得和下面这个片段类似：

```
agent.sources.s1.interceptors=e1
agent.sources.s1.interceptors.e1.type=regex_extractor
agent.sources.s1.interceptors.e1.regex=^([\\d.]+) \\S+ \\S+
\\ [([\\w:/]+\\s[+\\-]\\d{4})\\] \\"(.+?)\\" (\\d{3}) (\\d+)
agent.sources.s1.interceptors.e1.serializers=ip dt url sc bc
agent.sources.s1.interceptors.e1.serializers.ip.name=ip_address
agent.sources.s1.interceptors.e1.serializers.dt.type=org.apache.flume.
interceptor.RegexExtractorInterceptorMillisSerializer
agent.sources.s1.interceptors.e1.serializers.dt.pattern=yyyy/MMM/
dd:HH:mm:ss Z
agent.sources.s1.interceptors.e1.serializers.dt.name=timestamp
agent.sources.s1.interceptors.e1.serializers.url.name=http_request
agent.sources.s1.interceptors.e1.serializers.sc.name=status_code
agent.sources.s1.interceptors.e1.serializers.bc.name=bytes_xfered
```

这将对前文的内容产生如下的 header 信息：

```
{ "ip_address":"192.168.1.42", "timestamp":"1364588829",
"http_request":"GET /index.html HTTP/1.1", "status_code":"200",
"bytes_xfered":"1037" }
```

The body content is unaffected. You'll also notice I didn't specify default for the type of the other serializers as that is the default.



在拦截器中没有重写检查。例如，使用 `timestamp` 作为 key 值将会覆盖 event 之前存在的时间，如果存在的话。

你同样可以通过继承

`org.apache.flume.interceptor.RegexExtractorInterceptorSerializer` 接口来实现你自己的拦截器。然而，如果你的目的是将 **body** 中的数据移动到 **header** 中，你可能会想要实现一个自定义的拦截器，这样您就可以除了设置 **header** 值之外还可以修改 **body** 内容，否则数据就有可能重复。

让我们总结下这个拦截器的一些属性：

Key	Required	Type	Default
type	Yes	String	regex_extractor
regex	Yes	String	
serializers	Yes	Space-separated list of serializer names	
serializers.NAME.name	Yes	String	
serializers.NAME.type	No	Default or FQDN of implementation	default
serializers.NAME.PROP	No	Serializer-specific properties	

## 自定义拦截器

如果你有点想要添加进你 **Flume** 实现中的自定义代码，它很大可能是一个自定义拦截器。如之前提到的，你需要实现

`org.apache.flume.interceptor.Interceptor` 接口以及关联的 `org.apache.flume.interceptor.Interceptor.Builder` 接口。

假设我需要对我 **event** 的 **body** 的内容进行 **URL** 编码。代码可能如下方所示：

```
public class URLDecode implements Interceptor {

    public void initialize() {}

    public Event intercept(Event event)
    { try {
        byte[] decoded = URLDecoder.decode(new String(event.getBody()),
        "UTF-8").getBytes("UTF-8");
        event.setBody(decoded);
    } catch UnsupportedEncodingException e) {
        // This shouldn't happen. Fall through to unaltered event.
    }
}
```

```

        return event;
    }

    public List<Event> intercept(List<Event> events)
    { for (Event event:events) {
        intercept(event);
    }
    return events;
}

public void close() {}

public static class Builder implements Interceptor.Builder
{ public Interceptor build() {
    return new URLDecode();
}
    public void configure(Context context) {}
}
}

```

然后如果要配置我的新拦截器，对 **Builder** 类使用类似如下格式的 FQDN（完全限定域名）来作为 **type** 属性的值：

```

agent.sources.s1.interceptors=i1
agent.sources.s1.interceptors.i1.type=com.example.URLDecoder$Builder

```

对于如何传递和验证属性值的例子，可以去看 **Flume source** 部分已经实现的拦截器的源码来寻找灵感。

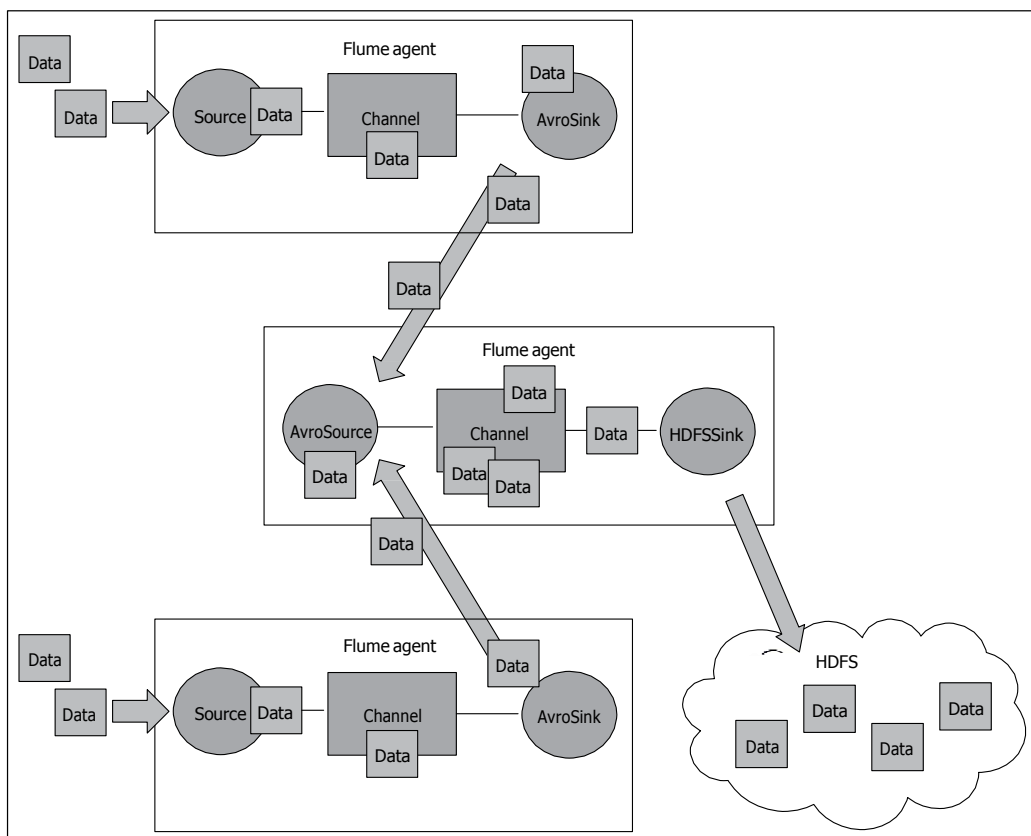
请记住自定义拦截器中任何繁重的处理过程都将影响整体的吞吐量，所以请注意自定义拦截器中对象的生成和密集的计算工作。

## 分层数据流

在第一章《概览》中，我们讨论了对数据分层。我们有很多理由需要这么做。你也许想要限制 **Flume agent** 直接连接到 **Hadoop** 集群的数量，从而间接限制并发请求数量。你也许同样因为在需要维护 **Hadoop** 集群的时候却发现应用服务器缺乏足够的存储空间来存储这些重要的数据。无论是什么原因或者场景，最常见的串联 **Flume agent** 的方式是配对使用 **Avro Source** 和 **Sink**。

## Avro Source/Sink

我们在第四章《Sink 和 Sink 处理器》中涉猎了一点点 Avro 知识，当时我们讨论了使用它作为存储在 HDFS 磁盘中的文件的序列化格式。这里我们用它在 Flume agent 之间进行通讯。下方是一种经典的配置方式：



要使用 Avro Source，你需要设置 `type` 属性值为 `avro`。你同样需要指定要监听的地址和端口号：

```
collector.sources=av1
collector.sources.av1.type=avro
collector.sources.av1.bind=0.0.0.0
collector.sources.av1.port=42424
collector.sources.av1.channels=ch1
collector.channels=ch1
```

```

collector.channels.ch1.type=memory
collector.sinks=k1
collector.sinks.k1.type=hdfs
collector.sinks.k1.channel=ch1
collector.sinks.k1.hdfs.path=/path/in/hdfs

```

右边的 agent 将监听 42424 端口，使用内存 channel，然后写数据到 HDFS 中。这里我使用内存通道是为了让配置看起来更加简洁。同样，注意这里我给这个 agent 起了一个不同的名字，叫 collector，目的是为了冲突。

左边的这个两 agent，将给 collector 供给数据，它的配置也许和下方这个类似。为了看起来更加简洁，我们省略了 source 的配置：

```

client.channels=ch1
client.channels.ch1.type=memory
client.sinks=k1
client.sinks.k1.type=avro
client.sinks.k1.channel=ch1
client.sinks.k1.hostname=collector.example.com
client.sinks.k1.port=42424

```

hostname 的值 “collector.example.com” 在部署这个 agent 的机器上没有任何作用，它是将要接收数据的目标机器的名称（或者 IP 地址）。这个配置，名叫 client，将被用于左侧的两个 agent，假设两个 agent 有相同的 source 源配置。

因为我不喜欢单点故障，我将用前面的配置来配置两个 collector agent，然后让 client agent 用 sink 组来在这两个之间轮询。同样为了简洁我略去 source 配置：

```

client.channels=ch1
client.channels.ch1.type=memory
client.sinks=k1 k2
client.sinks.k1.type=avro
client.sinks.k1.channel=ch1
client.sinks.k1.hostname=collectorA.example.com
client.sinks.k1.port=42424
client.sinks.k2.type=avro
client.sinks.k2.channel=ch1
client.sinks.k2.hostname=collectorB.example.com
client.sinks.k2.port=42424
client.sinkgroups=g1
client.sinkgroups.g1=k1 k2
client.sinkgroups.g1.processor.type=load_balance
client.sinkgroups.g1.processor.selector=round_robin
client.sinkgroups.g1.processor.backoff=true

```

## Command-line Avro

Avro Source 同样可以以命令行的方式来使用，你可能已经在第二章《Flume 快速入门》中见过了。不同于运行 `flume-ng` 以 `agent` 参数的形式，你可以通过 `avro-client` 参数来发送一个或多个文件到 Avro Source。关于 `avro-client` 的详细配置可以在 Flume 帮助文档中看到：

```
avro-client options:
  --dirname <dir>          directory to stream to avro source
  --host, -H <host>        hostname to which events will be sent
                           (required)
  --port, -p <port>        port of the avro source (required)
  --filename, -F <file>    text file to stream to avro source [default:
                           stdin]
  --headerFile, -R <file> headerFile containing headers as
                           key/value pairs on each new line
  --help, -h               display help text
```

这种变种的方式对于测试非常有用，如果错误可以手动重新发送数据，或者将旧数据导到其他地方。

就像 Avro Sink 一样，你需要指定发送数据的目标 `host` 和端口号。你可以使用 `-filename` 选项来发送单个文件或者使用 `-dirname` 选项来发送一个文件夹下的所有文件。如果这两个都没有指定，`stdin` 将会被使用。这里是如何发送一个叫 `foo.log` 的文件到我们之前配置的 Flume agent 的命令行：

```
$ ./flume-ng avro-client --filename foo.log --host collector.example.
com --port 42424
```

每一行的数据都将被转换成一个简单类型的 Flume event。

另外，你选择可以指定一个包含键值对的文件来为 Flume header 设置值。这个文件不要使用 Java 属性的语法。如果我有一个文件名叫 `headers.properties`：

```
pointOfSale=US
environment=staging
```

然后启动命令中包含 `-headerFile` 选项就会在每个 event 创建时往 header 中设置进去这两个键值对：

```
$ ./flume-ng avro-client --filename foo.log --headerFile headers.
properties --host collector.example.com --port 42424
```

## Log4J Appender

正如我们在第五章《Source 和 Channel 选择器》中讨论的，使用文件系统的文件作为数据源有可能会引起一些问题。一个避免这个问题的方法是在你的程序中使用 **Flume Log4j** 进行日志输出。在这种情况下，它使用和 **Avro Sink** 相同的 **Avro** 通讯技术，这样你就需要配置让它可以发送数据到 **Avro Source** 中。

这个输出也有两个属性，见的 XML 配置：

```
<appender name="FLUME" class="org.apache.flume.clients.log4jappender.
Log4jAppender">
  <param name="Hostname" value="collector.example.com"/>
  <param name="Port" value="42424"/>
</appender>
```

Body 内容的格式将取决于输出源的配置（这里未展示）。Log4j 可以添加到 Flume header 中的字段见下表：

Flume header key	Log4J LoggingEvent field
flume.client.log4j.logger.name	event.getLoggerName()
flume.client.log4j.log.level	event.getLevel() as a number. See org.apache.log4j.Level for mappings.
flume.client.log4j.timestamp	event.getTimeStamp()
flume.client.log4j.message. encoding	N/A. Always UTF8.
flume.client.log4j.logger.other	Will only see this if there was a problem mapping one of the previous fields—so normally this won't be present.

可以参考 <http://logging.apache.org/log4j/1.2/> 来了解 Log4j 的详细知识。你需要在你的 Java 程序中引入 **flume-ng-sdk jar** 包来使用 **Flume Log4j** 输出源。

请记住如果在发送数据到 **Avro Source** 的过程中出了问题，输出源将会抛出一个异常，并且这个日志也将被丢弃掉，因为根本没有地方来存储它。如果让它留在内存中将很快消耗尽你的 JVM 堆栈空间，这比丢弃掉数据来得更为严重得多。

## The Load Balancing Log4J Appender

我确信你注意到了 Log4j 输出源的配置文件中只包含了一个主机名/端口。如果你想要将负载分散到多个负责收集数据的 agent 上，无论是为了容量还是负载考虑，你可以使用 LoadBalancingLog4jAppender。这种日志输出源有一个必需的属性名叫 Hosts，它是一个用空格分隔的主机名和端口号的组合，就像下面这样：

```
<appender name="FLUME" class="org.apache.flume.clients.log4jappender.
LoadBalancingLog4jAppender">
  <param name="Hosts" value="server1:42424 server2:42424"/>
</appender>
```

还有一个可选属性 Selector，它指定了你负载均衡的实现方式。可选的属性值有 RANDOM 和 ROUND\_ROBIN。如果未指定，默认值是 RANDOM。你也可以实现你自己的选择器，但那部分超出了本书的讲解范围。如果你感兴趣，可以去看一下文档注释很全的 LoadBalancingLog4jAppender 类的源码就知道该怎么做了。



负载均衡 Log4j 输出源的默认的均衡选择机制是随机机制。你可能会至于到这和第四章《Sink 和 Sink 处理器》中 sink 组的默认均衡机制有所不同，sink 组的均衡机制默认是轮询机制。

可以说这是另一个举证，就是你千万要注意不要依赖于默认值，而应该去明确指定。

最后，还有另一个可选属性是黑名单的最大指数时间，这个是用来对那些无法连接的服务器进行暂时屏蔽使用的。首先，如果服务器无法连接，在这个服务可以再次被选中之前需要等待 1 秒钟时间。这个服务器每无法连接一次，这个等待时间就会翻倍，增长到默认的 30 秒的最大值。如果你想要增加这个值到 2 分钟时间，我们可以以毫秒数的形式像下面这样指定 MaxBackoff 属性的值：

```
<appender name="FLUME" class="org.apache.flume.clients.log4jappender.
LoadBalancingLog4jAppender">
  <param name="Hosts" value="server1:42424 server2:42424"/>
  <param name="Selector" value="ROUND_ROBIN"/>
  <param name="MaxBackoff" value="120000"/>
</appender>
```

在这个例子中，我们同样改写了默认的随机均衡机制转而使用轮询均衡机制。



## Routing

基于内容去路由不同的数据到不同的目的地相信现在你已经了解的很清楚了，我们已经给你介绍了 Flume 所有的路由机制了。

第一步就是将你要用于路由选择的数据写入到 Flume event 的 header 中，如果这个数据在 header 中还不存在的话。第二部就是使用多路复用 channel 选择器来基于 header 中的值将数据选择进入各种各样的 channel 中去。

例如，你想要捕获所有 HDFS 的异常。在这个配置中你可以发现 event 来自于 Avro Source s1 监听的端口 42424。Event 被检测来看是否内容中包含 “Exception”。如果包含了，它就在 event 的 header 中创建一个名为 “exception” 的 key（value 为 “Exception”）。这个 header 被用来将这些 event 选择进入 channel c1 中，最后写入 HDFS 中。

如果 event 没有匹配上这个正则表达式，它的 header 中将不包含 “exception” 这个 key，然后将会被传输到默认的 channel c2 中，它将通过 Avro 序列化传输到服务器 foo.example.com 的端口 12345 中去：

```
agent.sources=s1
agent.sources.s1.type=avro
agent.sources.s1.bind=0.0.0.0
agent.sources.s1.port=42424
agent.sources.s1.interceptors=i1
agent.sources.s1.interceptors.i1.type=regex_extractor
agent.sources.s1.interceptors.i1.regex=(Exception)
agent.sources.s1.interceptors.i1.serializers=ex
agent.sources.s1.interceptors.i1.serializers.ex.name=exception
agent.sources.s1.selector.type=multiplexing
agent.sources.s1.selector.header=exception
agent.sources.s1.selector.mapping.Exception=c1
agent.sources.s1.selector.default=c2
agent.channels=c1 c2
agent.channels.c1.type=memory
agent.channels.c2.type=memory
agent.sinks=k1 k2
agent.sinks.k1.type=hdfs
agent.sinks.k1.channel=c1
agent.sinks.k1.hdfs.path=/logs/exceptions/%y/%M/%d/%H
agent.sinks.k2.type=avro
agent.sinks.k2.channel=c2
agent.sinks.k2.hostname=foo.example.com
agent.sinks.k2.port=12345
```

## 小结

在这一章中我们覆盖了如下 Flume 自带的拦截器：

- **Timestamp:** 这个用来在 **header** 中添加一个时间戳，可能会覆盖已经存在的时间戳。
- **Host:** 这个用来在 **event** 的 **header** 中添加 Flume agent 的 **host** 名称或者 IP 地址。
- **Static:** 这个用来在 **header** 中添加一个静态的字符串。
- **Regular expression filtering:** 这个用来对符合正则表达式的 **event** 进行筛选或者剔除。
- **Regular expression extractor:** 这个用来在符合正则表达式的 **event** 中创建 **header** 信息。它同样用来和 **channel** 选择器配合使用。
- **Custom:** 这个用来创建一个自定义的拦截器来转换数据，如果你在这无法发现合适的拦截器。

我们同样讲解了使用 **Avro Source** 和 **Sink** 来对数据进行分层。

接下来，我们介绍两个 **Log4j** 日志输出源，一个单一路径和一个负载均衡版本的，来和 **Java** 程序直接进行整合。

最后，我们给出了一个使用拦截器结合 **channel** 选择器来提供路由决策逻辑的例子。

在下一章中，我们将讲述如何使用 **Ganglia** 监控 **Flume** 数据流。

# 7

## Flume 监控

Flume 的用户指南中这样写到:

*Monitoring in Flume is still a work in progress. Changes can happen very often.  
Several Flume components report metrics to the JMX platform MBean server.  
These metrics can be queried using JConsole.*

JMX 对浏览一些指标是很不错的, 当你有成百上千台在各处发着数据的时候用肉眼观察 JConsole 就不那么合适了。你需要的是一次能看到所有的数据。但你需要重点关注哪些东西? 那是个非常复杂的问题, 但我会在这一章中尽力覆盖几个我感觉比较重要的监控点。

## 监控 agent 进程

你想要做的一个最重要的监控是监控 Flume agent 的进程, 目的就是为了确保 agent 还在运行。有很多产品可以来做这种类型的监控, 所以这里我们没办法把它们都挨个讲一遍。如果你在任意一个规模大小的公司工作, 有很大可能已经有一个系统来做这个工作了。这种情况下, 不要自己再去造一个轮子了。最后一个运营想要做的事就是一个可以 7 天 24 小时可以看到的屏幕。

## Monit

如果你还没有一个适当的监控系统, 一个免费的选择就是 **Monit** (<http://mmonit.com/monit/>)。Monit 的开发者还有一个收费版本的可以提供更多的你也许想要的额外功能。尽管是免费的, 它还是可以提供给你一种检测 Flume agent 是否正常运行的方法, 如果没有正常运行可以重启它, 同时给你发送邮件, 这样你就可以立马去检查为何 agent 会死掉。

Monit 还可以做到更多的，但这些功能我们这里不会讲。如果你够聪明，除了我们本章讲述的，我知道你会第一时间增加对硬盘、CPU、内存使用情况的检测。

## Nagios

另一个可以选择用来监控 Flume agent 进程的工具是 **Nagios** (<http://www.nagios.org/>)。和 Monit 类似，你可以配置 Nagios 来监控你的 Flume agent，同时用 Web 网页、邮件或者 SNMP 陷阱对你进行告警。就是说，它没有重启功能。社区的强大使得有很多的插件可以提供给各种程序。我的公司使用这个来检查 Hadoop Web 网页的可用性。虽然不是对系统健康的完整监控，但它确实提供了更多对于我们 Hadoop 生态系统的整体监控。

再次重申，如果你的公司已经有了相关的工具，请在开始使用其他工具之前看看是否可以重用它们。

## 监控性能指标

既然我们已经讲解了一点点监控进程的工具，你是否想过你该如何知道你的程序是在真正的工作呢？在许多情况下我看到过被看住的 `syslog-ng` 进程表面看来是在正常运行的，但它就是无法发送数据。我没有特意针对 `syslog-ng`；如果没有设计来处理某种情况，在条件满足时所有的软件都会变成这样。

当讨论 Flume 数据流的时候，你需要监控下面这些东西：

- 输入的数据源速度在你的预期范围内
- 数据没有撑爆你的 `channel`
- 输出的数据源速度在你的预期范围内

Flume 有一个插件式的监控框架，但如我在这一章一开始就提到的，它还有许多的工作正在做。这并不意味着在它完成之前你就不能使用它。它意味着每当你升级的时候都需要准备额外的测试和集成。

虽然在 Flume 的官方文档中没有提及，通常情况下都支持在你的 Flume 的 JVM 中使用 JMX(<http://bit.ly/javajmx>)和使用 Nagios JMX 插件(<http://bit.ly/nagiosjmx>)来在 Flume agent 性能异常时进行告警。

## Ganglia

一种可以用来监控 Flume 内部参数的选择是 **Ganglia**。Ganglia (<http://ganglia.sourceforge.net/>) 是一款用于收集内部参数、图形化显示、且可以分布式部署来处理大量机器数据的开源监控工具。要将你的 Flume 内部参数发送给 Ganglia 集群，你需要在启动 agent 的时候传上一些参数：

Java property	Value	Description
flume.monitoring.type	ganglia	设置 ganglia
flume.monitoring.hosts	host1:port1, host2:port2	用空格分隔的 gmond 程序的主机名:端口列表。
flume.monitoring.pollInterval	60	发送数据之间的间隔（默认：60s）。
flume.monitoring.isGanglia3	false	如果使用旧的 ganglia3 的协议就设置为 true。默认使用 v3.1 版本的协议。

看看在相同网段 gmond 的实例（因为可达性是基于多路数据包的），你可以发现 gmond.conf 中 udp\_recv\_channel 的配置块。比方说我有两个相邻的服务器用下面两个相似的配置块：

```
udp_recv_channel
{ mcast_join =
  239.2.14.22
  port = 8649
  bind = 239.2.14.22
  retry_bind = true
}
udp_recv_channel
{ mcast_join =
  239.2.11.71
  port = 8649
  bind = 239.2.11.71
  retry_bind = true
}
```

在这个例子中我们在启动命令中指定 IP 和端口号 239.2.14.22/8649 对应第一个服务器，239.2.11.71/8649 对应第二个服务器：

```
-Dflume.monitoring.type=ganglia
-Dflume.monitoring.hosts=239.2.14.22:8649, 239.2.11.71:8649
```

这里，我使用默认的数据发送间隔以及使用新版的 ganglia 协议。



由于现在 Ganglia 支持 TCP 协议了，现在 Flume/Ganglia 的集成只支持使用多路 UDP 来发送数据。如果你有一个庞大复杂的网络结构，你可能在事与愿违的时候需要你的网络工程师的帮助。

## 内置 HTTP 服务

你配置开启 Flume agent 的 http 服务，可以输出 Json 格式的参数让外部的系统来查询。不像 Ganglia 集成推送那样，必须让一些外部的实体来主动拉取 Flume agent 的数据。

从理论上讲，你可以使用 Nagios 来拉取这个 JSON 数据然后在一定条件下发出告警，但我从来没有尝试过。同样的这个服务在开发和测试中非常有用，尤其是当你在写一个自定义的 Flume 组件来确保他们可以生成正确的参数。这里是你需要在 Flume agent 启动时需要配置的一些属性：

Java property	Value	Description
flume.monitoring.type	http	Set to http
flume.monitoring.port	The port number	The port number to bind the HTTP server

查看内部参数的 URL 如下：

```
http://SERVER_OR_IP_OF_AGENT:PORT/metrics
```

这个可以用在如下的 Flume 配置中：

```
agent.sources = s1
agent.channels = c1
agent.sinks = k1
agent.sources.s1.type=avro
agent.sources.s1.bind=0.0.0.0
agent.sources.s1.port=12345
agent.sources.s1.channels=c1
agent.channels.c1.type=memory
agent.sinks.k1.type=avro
agent.sinks.k1.hostname=192.168.33.33
agent.sinks.k1.port=9999
agent.sinks.k1.channel=c1
```

同样的，下面是启动参数：

```
-Dflume.monitoring.type=http  
-Dflume.monitoring.port=44444
```

访问 `http://SERVER_OR_IP:44444/metrics`，你也许可以看到如下的内容：

```
{  
  "SOURCE.s1": {  
    "OpenConnectionCount": "0",  
    "AppendBatchAcceptedCount": "0" ,  
    "AppendBatchReceivedCount": "0" ,  
    "Type": "SOURCE",  
    "EventAcceptedCount": "0",  
    "AppendReceivedCount": "0",  
    "StopTime": "0",  
    "EventReceivedCount": "0",  
    "StartTime": "1365128622891" ,  
    "AppendAcceptedCount": "0"} ,  
  "CHANNEL.c1": {  
    "EventPutSuccessCount": "0",  
    "ChannelFillPercentage": "0.0" ,  
    "Type": "CHANNEL",  
    "StopTime": "0",  
    "EventPutAttemptCount": "0",  
    "ChannelSize": "0",  
    "StartTime": "1365128621890" ,  
    "EventTakeSuccessCount": "0",  
    "ChannelCapacity": "100",  
    "EventTakeAttemptCount": "0"} ,  
  "SINK.k1": {  
    "BatchCompleteCount": "0",  
    "ConnectionFailedCount": "4",  
    "EventDrainAttemptCount": "0" ,  
    "ConnectionCreatedCount": "0" ,  
    "BatchEmptyCount": "0",  
    "Type": "SINK",  
    "ConnectionClosedCount": "0",  
    "EventDrainSuccessCount": "0" ,  
    "StopTime": "0",  
    "StartTime": "1365128622325",  
    "BatchUnderflowCount": "0"}  
}
```

}



正如你看到的，每个 `source`、`sink`、`channel` 都被用自身的参数区分开来。每一种 `source`、`channel` 和 `sink` 提供他们自身的内部参数，尽管有一些共性，所以一定要注意你感兴趣的数据。举例来说，`Avro Source` 有 `OpenConnectionCount` 参数，它是 `client` 的连接数量（最可能是正在发送数据的）。这可能会帮助你判断是否有你预期数量的客户端在给你发送数据，或者有过多的 `client` 端，这时就得考虑对 `agent` 进行分层了。

一般而言，`channel` 的 `ChannelSize` 和 `ChannelFillPercentage` 会告诉你是否数据来的比去的快。它同样会告诉你是否你有将它设置的足够大用以支撑后端维护和停机时的数据量。

再看看 `sink`，`EventDrainSuccessCount` 对应 `EventDrainAttemptCount` 会告诉你相比于尝试耗尽的次数成功耗尽的频率是多少。在这个例子中，我配置了一个 `Avro Sink` 往不存在的目标发送数据。正如你看到的 `ConnectionFailedCount` 的值在增长，这对于持续连接是一个很好的衡量指标。甚至增长的 `ConnectionCreatedCount` 可以衡量出连接被断掉了并且重连太频繁了。

现实中，对于看 `ChannelSize/ChannelFillPercentage` 并没有硬性的规定。每一个使用场景都有它自己的性能指标，所以从小事做起，学习和完成你自己的监控系统。

## 自定义监控 Hooks

如果你已经有了一个监控系统，你可能想要通过额外的工作来创建一个自定义的监控报警机制。你可能认为只要简单实现

`org.apache.flume.instrumentation.MonitorService` 接口就行了。你确实需要这么做，但看下这个接口，你只要看下 `start()` 方法和 `stop()` 方法。不像前面拦截器那样明显的实现，`agent` 期望你 `MonitorService` 的实现可以开启/关闭一个线程来定期发送数据到一个接收服务器中。如果你想操作一个服务，比如 `HTTP` 服务，然后开启/关闭方法将可以用来开启或关闭你的监听服务。内部的一些参数通过各种各样的 `source`、`sink`、`channel` 和使用 `org.apache.flume` 开头的对象的拦截器给到 `JMX`。你的实现需要读取 `MBeanServer` 来得到数据。我能给你的最好建议就是，当决定实现自己的监控时，先看下 `source` 的两个实现然后模仿他们的做法。要使用你的监控 `Hook`，设置 `flume.monitoring.type` 属性为你实现类的全局限定类名。可以预见的就是你在新版本 `Flume` 中重做自定义的 `hook` 直到这个框架成熟稳定。

PS: 本节作者水平有限，翻译可能无法表达原作的意思

## 小结

在这一章中我们同时讲述了从进程级别和内部参数级别（是否 Flume 真的在正常工作）来监控 Flume agent。

我们介绍了开源的 Monit 和 Nagios 来作为进程监控的选择。

然后我们讲述了使用 Ganglia 和 Flume 自带的 Http 请求返回 Json 数据来对 Flume agent 进行内部指标监控。Next we covered the Flume agent internal monitoring metrics with Ganglia and JSON over HTTP implementations that ship with Apache Flume.

最后，我们讲述了如何整合一个自定义的监控以防你需要直接整合其他 Flume 默认不支持的工具。

在最后一章中我们会讨论一些在 Flume 开发过程中的总体思想。



# 8

## There Is No Spoon — 即时分布式数据 收集系统的现实

在这最后一章中，我认为我们需要讨论一些不太具体的东西，我将数据收集到 Hadoop 中时更多的是一些随性思想。在这背后没有确切的依据，所以你不同意我看法时也无需感到不自在。

虽然 Hadoop 是一个消化海量数据的伟大工具，While Hadoop is a great tool for consuming vast quantities of data，我经常在思索发生在 1886 年明尼苏达州圣克罗伊河僵局的画面(<http://www.nps.gov/sacn/historyculture/stories.htm>)。当处理过量数据时你想要确保这不会阻塞你的河道。一定要认真对待前一章的监控，而不仅仅是作为一个更好的选择。

### 传输时间对比记录时间

我遇到过一个情况就是数据存储在一个文件名中使用日期的 HDFS 中，然而数据内容却和目录名称不匹配。期望的结果是在 2013/03/29 目录下面的数据包含了所有 2013 年 3 月 29 日的数据。但是事实是在传输过程中日期被改动了。原来我们使用的 syslog 版本在重写头部，包括日期部分，导致了从传输中获得的时间数据无法反映出记录的原始时间。通常情况下这种偏差很小，通常只有一两秒，所以没人会真正注意到这个。但如果有一天中继服务器死了，当数据被困在上游服务器中，最终发送的时候会用当前时间。在这个例子中，数据偏差了几天时间。太混乱了。

如果你使用日期来放置数据请确保这个不会发生在你的身上。检查一下日期边缘时刻的情况来看数据是否是你想要的，确保你测试你的故障场景在他们真实发生在生产环境之前。

正如我前面提到的，因为计划中或计划外的维护（甚至小小的网络波动）导致的数据重发很可能会造成数据重复和无序的数据传输，所以在数据传输前一定要考虑这一点。Flume 中没有单独的传输/排序的保障机制。如果你需要，使用一个带事务的数据库来替代。

## 罪恶的时间戳

如果你在第四章《Sink 和 Sink 处理器》中错过了我反对使用本地时间的偏见，我在这里要再强调一次——时间戳是罪恶的。就像邪恶博士 ([http://en.wikipedia.org/wiki/Dr.\\_Evil](http://en.wikipedia.org/wiki/Dr._Evil))那样邪恶——同时让我们不要忘记“迷你我” (<http://en.wikipedia.org/wiki/Mini-Me>)这个副本——DST（日光节约时间，我国叫夏令时）。

我们现在生活在一个全球化的世界中。你将所有地方的数据推送到你的 Hadoop 集群中。你甚至在一个国家的不同地方（或者整个世界）有多个数据中心。在分析你的数据的时候最后你想要做的事情就是处理脏数据。DST 至少在一年中数次改变了地球上一些地方的时间。看一下历史就知道了 (<ftp://ftp.iana.org/tz/releases/>)。为了避免头疼让他标准化为 UTC 时间就可以了。如果为了体验更好，你可以在被人查看时将它变为“当地时间”。当它存留在你的集群中时，请保持它为标准的 UTC 时间。如果考虑在所有地方都使用 UTC 时间可以在 Java 启动时设置参数（如果你不能修改系统时间的话）：

```
-Duser.timezone=UTC
```

我住在芝加哥并且我们的工作电脑都使用中央标准时间，就是考虑到 DST 问题。在我们的 Hadoop 集群中我们喜欢让数据以 YYYY/MM/DD/HH 的格式布局。一年中有两次会让事情变得很糟。在秋天，我们的早上 2 点的目录中有两倍的数据量。在春天就根本没有早上 2 点钟的目录。真疯狂！

## 容量规划

不管你认为你有多少数据，事情总是在不停的变化。新项目在不停的上线然后已有项目创建的数据也在变动（上升或者下降）。数据量也会随着每天的访问量起伏不定。最后，给你 Hadoop 集群提供数据的服务器数量也在不停的变化。

关于 Hadoop 集群中应该保留多少冗余有很多的流派思想（我们使用没有任何科学依据的 20%——这意味着我们通常将 80%作为满载可以订购硬件的标志，但不会惊慌知道我们到达了 85%-90%的利用率）。

你可能还需要在一个 agent 中设置多重数据流。由于 Source 和 sink 是单线程的所以在巨型数据量的情况下完成批量数据通信是有限制的。

给 Hadoop 提供数据的 Flume agent 的数量应该基于实际数据来调整。通过查看 channel 大小来看在正常速度下 source 的写状态情况。调整最大 channel 容量来让你无论在怎么的流量下都能保持完好。你可以将大小设置得比你需要的大，但即使一个长时间停机也可能超出你预估的最大大小。这个时候就需要你来判断那个数据对你更重要，然后调整你的 channel 大小来重点支持它。这样的话，如果达到了你的限制，不那么重要的数据将会第一个被丢弃掉。

很可能你的公司没有无限量的钱，在某一时刻继续扩展集群的耗费相对于数据的价值会收到质疑。这就是为什么要根据数据价值来设置收集限制的原因。任何程序发送到 Hadoop 集群的数据应该能说出数据的价值是什么，并且如果删除旧的数据会丧失什么。这是人们在写报告时唯一可以做出正确决定的方法。

## 多个数据中心时的注意事项

如果你在你的多个数据中心上运行业务并且有大量数据在收集的话，你可能想要考虑在每个数据中心中都设立 Hadoop 集群而不是把所有收集到的数据都放到一个数据中心的。这会相比于你可以运行一个 MapReduce 工作来应对所有数据来的困难的多。相反你需要并行运行工作然后通过第二轮计算合并结果。你可以用这个来查找和计算问题，但不可用用来做平均值运算——平均值的平均和一次平均是不一样的。

将所有数据拉到一个集群中也许同样会超出你的网络负载。这取决于你的数据中心之间是如何互相连接的，你可能无法发送你想要的数据量。最后，考虑到集群的整体故障或者沦陷可能会抹去一切，因为大多数集群太大而无法备份除了重要数据以外的其他数据。在这种情况下有一些旧数据总比一无所有好。对于多个 Hadoop 集群，如果你不想等待发送到本地的话，你需要有能力使用故障转移 sink 处理器来将数据发送到不同的集群。

如果你选择将所有数据发送到一个目的地，需要考虑添加一个大容量的机器来作为中继服务器的数据中心。这样如果有连接文件或者扩展集群需要维护，你可以让数据堆积在机器上而不是为你提供数据的客户机上。这是一个忠告即使在只有一个数据中心的情况下。

## 合规和数据抹除

记住你公司收集的顾客信息中可能包含敏感信息。你可能会受其他监管来限制访问数据，比如说 **Payment Card Industry (PCI)**—[http://en.wikipedia.org/wiki/PCI\\_DSS](http://en.wikipedia.org/wiki/PCI_DSS)) 或者 **Sarbanes Oxley (SOX)**—[http://en.wikipedia.org/wiki/Sarbanes%20%930xley\\_Act](http://en.wikipedia.org/wiki/Sarbanes%20%930xley_Act))。如果你在集群中没有妥善访问这些信息，政府可能会对你传讯甚至更糟，如果你的顾客觉得你没有保护好他们的权利和身份信息你将不会再有顾客上门。可以考虑对用户个人信息进行混淆、加密、隐藏等。业务上来说你通常对于分析诸如“有多少人早搜索一把锤子，但实际上买了的？”此类的问题，而不是“有多少顾客叫 Bob？”。正如你在第六章“拦截器”中看到的，写出一个可以在传输时加密 **Personally Identifiable Information (PII)**—[http://en.wikipedia.org/wiki/Personally\\_identifiable\\_information](http://en.wikipedia.org/wiki/Personally_identifiable_information)) 信息的拦截器是相当容易的。

你的公司可能有一个文档保留政策，很可能包含了你可以往 Hadoop 中放入的数据内容。请确保你删除掉那些你们公司政策不允许保留的数据。最后你可能想做的就是见一下你的律师。

## 小结

在这一章中我们讲述了几个现实中部署 Flume 时你需要考虑的事情，包含如下内容：

- 传输时间和记录时间有时候不匹配
- 相比于基准时间夏令时的危害
- 容量的规划
- 当有多个数据中心时需要考虑的一些事情
- 数据合规
- 数据抹除

我希望你喜欢这本书。希望你能够将这些信息直接用在你的程序和 Hadoop 集成工作中。

谢谢。

# 9

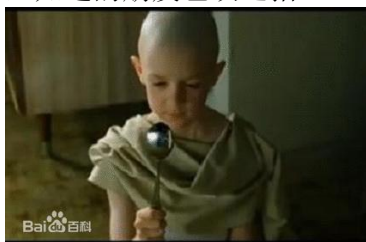
## 译者注

### 关于这本书

这本书是一位外国作者写的，可能是 Flume 系统使用太过简单，到本书翻译完成（2014 年 11 月 11 日）世界上也就仅此一本介绍 Flume 的书籍。当然这也从侧面说明了官方文档写的是相当的详细。

本书没有太多关于 Flume 代码层面的解读，如果想深入了解 Flume 的源码，请自行去官方下载 Flume 的源码，如果懒一点的可以参考玖疯的博客，可以省去自己不少的功夫：<http://www.cnblogs.com/lxf20061900/category/565688.html>。当然如果想精通 Flume 体系结构建议还是自己去翻阅源码，毕竟每个人的侧重点不同，文章中由于篇幅原因也无法太过详细的去解释每个细节，毕竟有些技术思想、设计模式不是三言两语就能说清楚的。

最后要说声抱歉的就是由于英文水平有限，加上对 Flume 的理解是随着时间慢慢加深的，文章中避免不了有一些翻译和理解错误，文字错误就更多了，欢迎大家及时纠正。还有由于文化差异这位老外引用了一些比喻，比如“*There Is No Spoon*”不懂的可以参考《黑客帝国》这部电影，“邪恶博士”可以参考《王牌大贱谍》这部电影，还有一些其他比喻就不得而知了，知道的朋友也欢迎指正。



### 关于译者

译者冯佳冬，初出茅庐，属于 90 后的先行者，就职于“1 号店”，推崇开源环境，爱倒腾源码，接触 Flume 源于偶然，也幸运的结识了一帮朋友，当然了译者不是大数据开发工程师，主要方向还是 Java 工程师，主攻 SOA 治理、系统优化、水平拆库等。



### 欢迎加入 Flume 大家庭

Flume 技术交流群：220278956，欢迎各位的加入，有新兵，有老鸟，当然老鸟有时候经常不在，可以直接@他们就行，本书完书时群内共有 421 人，欢迎加入。



# 索引

## Symbols

-c parameter 21  
-Dflume.root.logger property 20  
--dirname option 72  
--headerFile option 72

## A

agent 10  
agent.channels.access 17  
agent.channels property 18  
agent command 20  
agent process  
    monitoring 77  
Apache Avro serializer 40  
Apache Bigtop project  
    URL 16  
avro-client parameter 72  
avro\_event serializer 39  
Avro Sink. *See* Avro Source  
Avro Source  
    about 70  
    command-line 72  
    using 70, 71

## B

batchSize property 50, 52,  
57 best effort (BE) 8  
bufferMaxLines property 52  
byteCapacityBufferPercentage,  
    configura- tion parameter 26  
byteCapacity, configuration parameter  
26

## C

capacity, configuration parameter 26, 28  
channel 10  
ChannelException 25  
channel parameter 34  
channel selector  
    about 58  
    multiplexing channel selector 58  
    replicating channel selector 58  
channels parameter 55  
channels property 50, 52, 54, 56  
charset.default property 57  
charset.port.PORT# property 57  
checkpointDir, configuration parameter  
28 checkpointInterval, configuration  
parameter  
    28  
Cloudera  
    about 7  
    URL 17  
codecs 38  
command property 50  
CompressedStream file type 42

## D

data flows  
    tiering 69  
data  
    routing 75  
dataDir path 30  
dataDirs, configuration parameter  
28 disk failover (DFO) 8

## E

**Elastic Search** 13

**end-to-end (E2E)** 8

**event** 11, 61

**Event serializer**

about 39

Apache Avro 40

File type 41

Text output 39

Text with headers 39

timeouts and workers 42,  
43

**eventSize** parameter 55

**eventSize** property 57

**excludeEvents** property 64

**exec** source

about 49, 50

batchSize property 50, 51

channels property 50

command property 50

logStdErr property 50

restart property 50

restartThrottle property 50

type property 50

write-timeout, configuration parameter 28

**fileHeaderKey** property 52

**fileHeader** property 52

**fileSuffix** property 52

## F

**Facility, header key** 54, 55, 57

**failover** 45

**File Channel**

about 27

capacity, configuration parameter 28

checkpointDir, configuration parameter

28 checkpointInterval, configuration

param-

eter 28

configuration parameters 28

dataDirs, configuration parameter

28

keep-alive, configuration parameter 28

maxFileSize, configuration parameter

28 minimumRequiredSpace,

configuration

parameter 28

transactionCapacity, configuration

param- eter 28

using 28

## **File Type**

- about 41
- Compressed stream file
- type 42 Data stream file
- type 41 SequenceFile file
- type 41

## **Flume**

- configuration file, overview 17
- downloading 15
- event 11, 61
- in Hadoop
- distributions 16
- monitoring 77
- URL 15

## **Flume 0.9 8**

## **Flume 1.X 8, 9**

- flume.client.log4j.logger.name 73
- flume.client.log4j.logger.other 73
- flume.client.log4j.log.level 73
- flume.client.log4j.message.encoding 73
- flume.client.log4j.timestamp 73

## **Flume JVM**

- URL 78

- flume.monitoring.hosts property 79
- flume.monitoring.isGanglia3 property 79
- flume.monitoring.pollInterval
- property 79 flume.monitoring.port
- type property 80
- flume.monitoring.type property 79-82
- Flume-NG 8
- flume.syslog.status, header key 54-57

# **G**

## **Ganglia**

- about 79
- URL 79

# **H**

## **Hadoop distributions**

- Flume 16

## **Hadoop File System. *See***

## **HDFS HDFS**

- about 7, 13
- issues 9, 10

- hdfs.batchSize parameter 35
- hdfs.callTimeout property 42
- hdfs.codeC parameter 35
- hdfs.filePrefix parameter 34
- hdfs.fileSuffix parameter 34

**hdfs.fileSuffix** property 35, 38  
**hdfs.fileType** property 41  
**hdfs.idleTimeout** property 42, 43  
**hdfs.inUsePrefix** parameter 35  
**hdfs.inUseSuffix** parameter 35  
**hdfs.maxOpenFiles** parameter 34  
**hdfs.path** parameter 34  
**hdfs.rollCount** parameter 35  
**hdfs.rollInterval** parameter 35  
**hdfs.rollSize** parameter 35  
**hdfs.rollSize** rotation 38  
**hdfs.rollTimerPoolSize** property 42, 43  
**hdfs.round** parameter 34  
**hdfs.roundUnit** parameter 35  
**hdfs.roundValue** parameter 34  
**HDFS Sink**  
 about 33, 34  
 absolute 34  
 absolute with server name 34  
 channel parameter 34  
 file rotation 37, 38  
**hdfs.batchSize** parameter 35  
**hdfs.codeC** parameter 35  
**hdfs.filePrefix** parameter 34  
**hdfs.fileSuffix** parameter 34  
**hdfs.inUsePrefix** parameter 35  
**hdfs.inUseSuffix** parameter 35  
**hdfs.maxOpenFiles** parameter 34  
**hdfs.path** parameter 34  
**hdfs.rollCount** parameter 35  
**hdfs.rollInterval** parameter 35  
**hdfs.rollSize** parameter 35  
**hdfs.round** parameter 34  
**hdfs.roundUnit** parameter 35  
**hdfs.roundValue** parameter 34  
**hdfs.timeZone** parameter 35  
 path and filename 35-37  
 relative 34  
 type parameter 34  
 using 33  
**hdfs.threadsPoolSize** property 42  
**hdfs.timeZone** parameter 35  
**hdfs.timeZone** property 37  
**hdfs.writeType** property 41  
 Hello World example 18-22

**help** command 20

## **Hortonworks**

URL 17

**hostHeader property 63**

## **Host interceptor**

about 63

hostHeader property 63

preserveExisting property 63

type property 63

useIP property 63

**hostname, header key 54-57**

**host parameter 55**

**host property**

**54, 57 HTTP**

## **server**

about 80-82

flume.monitoring.port type

property 80

flume.monitoring.type

property 80

## **I**

## **interceptors**

about 12, 61, 62

custom interceptors 68, 69

Host interceptor 63

regular expression extractor

interceptor 65-67

regular expression filtering interceptor

64

Static interceptor 63, 64

Timestamp interceptor 62

**interceptors property 61**

## **J**

**JMX 77**

## **K**

**keep-alive, configuration parameter**

**26, 28**

**keep-alive parameter 27**

**key property 64**

## **L**

**load balancing 44**

**LoadBalancingLog4jAppend**

**er class 74 local filesystem**

**backed channel 25 Log4j**

**Appender**

about 73

flume.client.log4j.logger.name 73

- flume.client.log4j.logger.other 73
- flume.client.log4j.log.level 73
- flume.client.log4j.message.encoding 73
- flume.client.log4j.timestamp 73
- Flume headers 73
- load balancing 74
- properties 73
- logStdErr property** 50
- log time**
  - versus transport time 85, 86

## M

- MapR**
  - URL 17
- MaxBackoff property** 74
- maxBufferLineLength property** 52
- maxFileSize, configuration parameter**
- 28 memory-backed channel** 25
- Memory Channel**
  - about 18, 26
  - byteCapacityBufferPercentage, configuration parameter 26
  - byteCapacity, configuration parameter 26
  - capacity, configuration parameter 26
  - configuration parameters 26
  - keep-alive, configuration parameter 26
  - transactionCapacity, configuration parameter 26
  - type, configuration parameter 26
- minimumRequiredSpace, configuration parameter** 28
- Monit**
  - about 77, 78
  - URL 77
- multiple data centers**
  - considerations 87
- multiplexing channel selector** 58
- multiport syslog TCP source**
  - about 56
  - batchSize property 57
  - channels property 56
  - charset.default property 57
  - charset.port.PORT# property 57
  - eventSize property 57
  - Facility, header key 57

- flume.syslog.status, header key 57
- hostname, header key 57
- numProcessors property 57
- portHeader property 57
- ports property 56
- priority, header key 57
- readBufferSize property 57
- timestamp, header key 57
- type property 56

## N

- Nagios**
  - about 78
  - URL 78
- Nagios JMX**
  - flume.monitoring.hosts property 79
  - flume.monitoring.isGanglia3 property 79
  - flume.monitoring.pollInterval property 79
  - flume.monitoring.type property 79
  - URL 78
- name** 17
- netcat** 22
- non-durable channel** 25
- numProcessors property** 57

## O

- org.apache.flume.interceptor.Interceptor**
  - interface 68
- org.apache.flume.sink.AbstractSink** class 33
- org.apache.flume.source.AbstractSource**
  - class 47

## P

- Payment Card Industry.** *See* PCI
- PCI** 88
- Personally Identifiable Information.** *See* PII
- PII** 88
- Portable Operating System Interface.** *See* POSIX
- portHeader property** 57
- port parameter** 55
- port property** 54
- ports property** 56
- POSIX** 9

- preserveExisting** property 62-64
- priority**, header key 54-57
- processor.backoff** property 44
- processor.maxpenalty** property 45
- processor.priority.NAME** property 45
- processor.priority** property 45
- processor.selector** property 44
- processor.type** property 44, 45

## R

- readBufferSize** property 57
- Red Hat Enterprise Linux (RHEL)** 16
- regex** property 64, 65, 68
- regular expression**
  - filtering 65
- regular expression extractor interceptor**
  - about 65-67
  - properties 68
  - regex property 68
  - serializers.NAME.name property 68
  - serializers.NAME.PROP property 68
  - serializers.NAME.type property 68
  - serializers property 68
  - type property 68
- regular expression filtering interceptor**
  - about 64
  - cludeEvents property 64
  - properties 64
  - regex property 64
  - type property 64
- relayHost** header 63
- replicating channel** selector 58
- restart** property 50
- restartThrottle** property 50
- RFC 3164**
  - URL 53
- RFC 5424**
  - URL 53
- routing** 75
- rsyslog**
  - URL 53

## S

- Sarbanes Oxley**. *See* SOX
- selector.header** property 58

- selector.type** property 58
- serializer.appendNewLine** property 39
- serializer.compressionCodec** property 40
- serializer** property 39
- serializers** 65
  - serializers.NAME.name** property 68
  - serializers.NAME.PROP** property 68
  - serializers.NAME.type** property 68
  - serializers** property 68
  - serializer.syncIntervalBytes** property 40
- Sink** 10
- Sink groups**
  - about 43, 44
  - failover 45
  - load balancing 44
- source** 10
- SOX** 88
- spoolDir** property 51, 52
- spooling directory**
  - source**
    - about 51, 52
    - batchSize** property 52
    - bufferMaxLines** property 52
    - channels** property 52
    - fileHeaderKey** property 52
    - fileHeader** property 52
    - fileSuffix** property 52
    - maxBufferLineLength** property 52
    - spoolDir** property 52
    - type** property 52
- start() method** 82
- Static interceptor**
  - about 63
  - key** property 64
  - preserveExisting** property 64
  - properties 64
  - type** property 64
  - value** property 64
- stop() method** 82
- syslog sources**
  - about 53
  - multiport syslog TCP** source 56, 57
  - syslog TCP** source 55
  - syslog UDP** source 53, 54
- syslog TCP source**
  - about 55
  - channels** parameter 55
  - eventSize** parameter 55

- Facility, header key 55
- flume.syslog.status, header key 55
- hostname, header key 55
- host parameter 55
- port parameter 55
- priority, header key 55
- timestamp, header key 55
- type parameter 55

#### **syslog UDP source**

- about 53
- channels property 54
- Facility, header key 54
- flume.syslog.status, header key 54
- hostname, header key 54
- host property 54
- port property 54
- priority, header key 54
- timestamp, header key 54
- type property 54

## **T**

- tail 47
- tail -F command 49
- TailSource 47, 48
- Text output serializer 39
- text\_with\_headers serializer 39

- timestamp, header key 54-57
- Timestamp interceptor**
  - preserveExisting property 62
  - properties 62
  - type property 62
- time zones 86**
- transactionCapacity, configuration parameter 26, 28**
- transport time**
  - versus log time 85, 86
- type, configuration parameter 26**
- type parameter 34, 55**
- type property 50, 52, 54, 56, 63, 64, 68**

## **U**

- useIP property 63

## **V**

- value property 64
- version command 20

## **W**

- Write Ahead Log (WAL) 27
- write-timeout, configuration parameter 28





## Thank you for buying **Apache Flume: Distributed Log Collection for Hadoop**

### About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: [www.packtpub.com](http://www.packtpub.com).

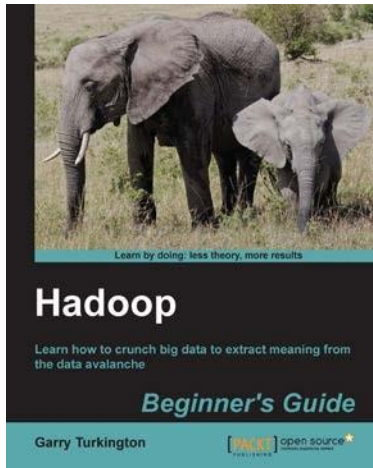
### About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

### Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to [author@packtpub.com](mailto:author@packtpub.com). If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



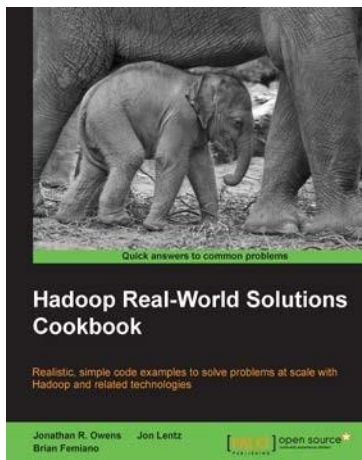
## Hadoop Beginner's Guide

ISBN: 978-1-84951-730-0

Paperback: 398 pages

Learn how to crunch big data to extract meaning from the data avalanche

1. Learn tools and techniques that let you approach big data with relish and not fear
2. Shows how to build a complete infrastructure to handle your needs as your data grows
3. Hands-on examples in each chapter give the big picture while also giving direct experience



## Hadoop Real-World Solutions Cookbook

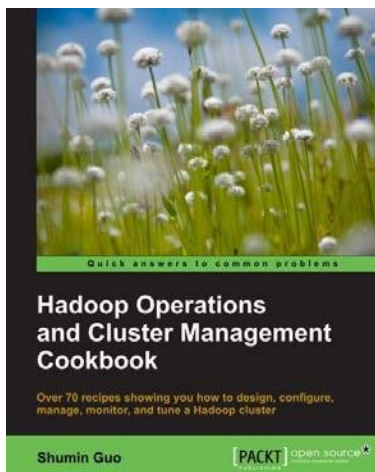
ISBN: 978-1-84951-912-0

Paperback: 316 pages

Realistic, simple code examples to solve problems at scale with Hadoop and related technologies

1. Solutions to common problems when working in the Hadoop environment
2. Recipes for (un)loading data, analytics, and troubleshooting
3. In depth code examples demonstrating various analytic models, analytic solutions, and common best practices

Please check [www.PacktPub.com](http://www.PacktPub.com) for information on our titles



## Hadoop Operations and Cluster Management Cookbook

ISBN: 978-1-78216-516-3      Paperback: 350 pages

Over 70 recipes showing you how to design, configure, manage, monitor, and tune a Hadoop cluster

1. Hands-on recipes to configure a Hadoop cluster from bare metal hardware nodes
2. Practical and in depth explanation of cluster management commands
3. Easy-to-understand recipes for securing and monitoring a Hadoop cluster, and design considerations



## HBase Administration Cookbook

ISBN: 978-1-84951-714-0      Paperback: 332 pages

Master HBase configuration and administration for optimum database performance

1. Move large amounts of data into HBase and learn how to manage it efficiently
2. Set up HBase on the cloud, get it ready for production, and run it smoothly with high performance
3. Maximize the ability of HBase with the Hadoop eco-system including HDFS, MapReduce, Zookeeper, and Hive

Please check [www.PacktPub.com](http://www.PacktPub.com) for information on our titles