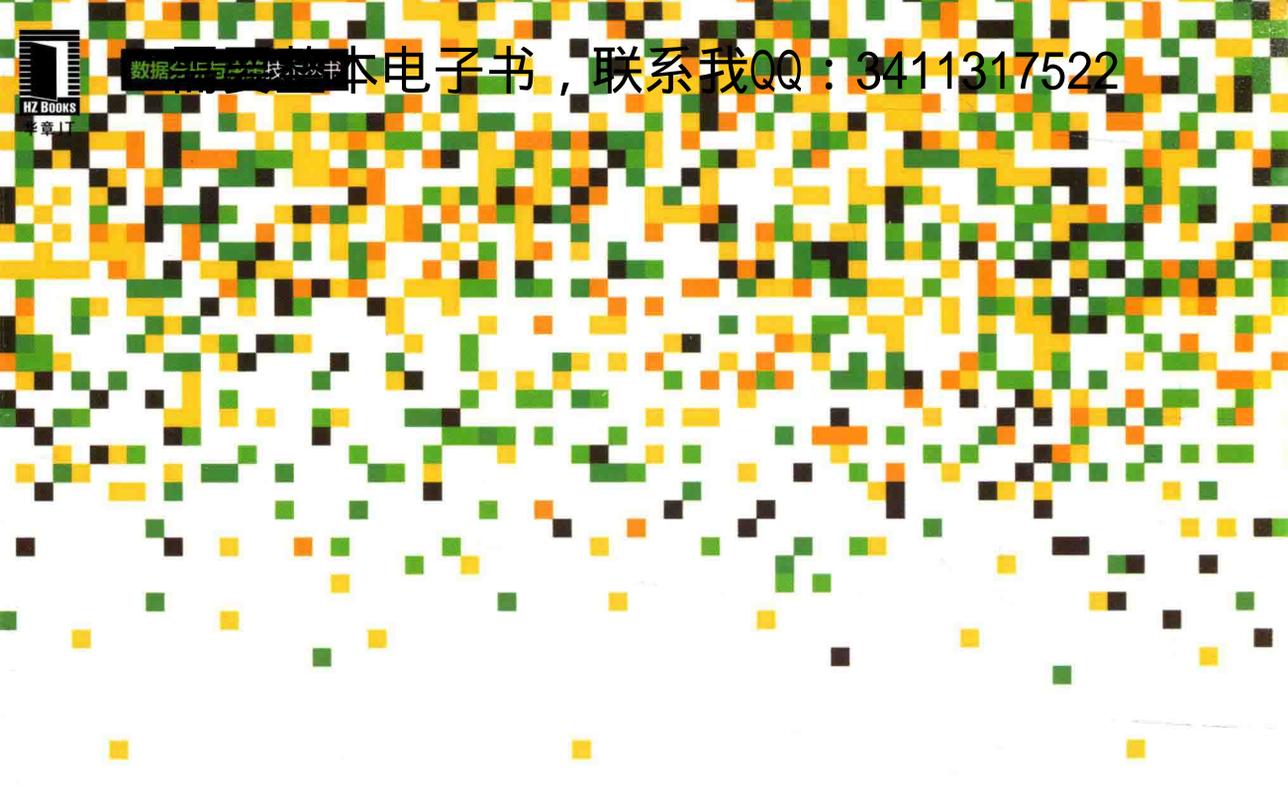




数据台与云技术丛书 本电子书，联系我QQ：3411317522



The Definitive Guide to Elasticsearch+Logstash+Kibana

# ELK stack 权威指南

饶琛琳◎编著

Elasticsearch+Logstash+Kibana一站式数据分析解决方案，快速应对大数据时代的数据收集、检索、可视化

从基础部署到千亿级扩展方案，从性能优化到插件开发，从数据模型到源码解析，全方位解析ELK，融入了作者多年大型网站运维开发的实战经验

需要整本电子书  联系我QQ：3411317522  
China Machine Press

需要整本电子书，联系我QQ：3411317522

数据分析与决策  
技术丛书

The Definitive Guide to Elasticsearch+Logstash+Kibana

# ELK stack 权威指南

饶琛琳◎编著

需要整本电子书，联系我QQ：3411317522

机械工业出版社  
China Machine Press

# 需要整本电子书，联系我QQ：3411317522

## 图书在版编目（CIP）数据

ELK stack 权威指南 / 饶琛琳编著. —北京：机械工业出版社，2015.9  
(数据分析与决策技术丛书)

ISBN 978-7-111-51634-7

I. E… II. 饶… III. 数据处理软件—指南 IV. TP274-62

中国版本图书馆 CIP 数据核字（2015）第 222006 号

ELK stack 是以 Elasticsearch、Logstash、Kibana 三个开源软件为主的数据处理工具链，是目前开源界最流行的实时数据分析解决方案，成为实时日志处理领域开源界的第一选择。然而，ELK stack 并不是实时数据分析的灵丹妙药，使用不恰当，反而会事倍功半。本书对 ELK stack 的原理进行了解剖，不仅分享了大量实战案例和实现效果，而且分析了部分源代码，使读者不仅知其然还知其所以然。读者可通过本书快速掌握实时日志处理方法，并搭建自己的数据分析系统。本书分为三大部分，共 19 章。第一部分“Logstash”介绍 Logstash 的安装与配置、场景示例、性能与测试、扩展方案、源码解析、插件开发等；第二部分“Elasticsearch”介绍 Elasticsearch 的架构原理、数据接口用例、性能优化、测试和扩展方案、映射与模板的定制、监控方案等；第三部分“Kibana”介绍 Kibana 3 和 Kibana 4 的特点对比，并分别讲解了 Kibana 3 和 Kibana 4 的配置、案例与源代码解析。

## ELK stack 权威指南

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：吴怡

责任校对：董纪丽

印刷：北京诚信伟业印刷有限公司

版次：2015 年 10 月第 1 版第 1 次印刷

开本：186mm×240mm 1/16

印张：25.5

书号：ISBN 978-7-111-51634-7

定价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88379426 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书责任编辑：李洪波 郭晓东

# 需要整本电子书，联系我QQ：3411317522

需要整本电子书，联系我QQ：3411317522

华章IT  
HZBOOKS | Information Technology



需要整本电子书，联系我QQ：3411317522

操作系统、应用服务和业务逻辑，都在不停地产生日志数据。过去，日志数据基本都存在单机磁盘上，只能用来做临时的事后分析和审计；有 Hadoop 以后，大家渐渐习惯将日志收集到 HDFS 中，然后每天运行 MapReduce 任务做统计报表。但是，面对诸如“新上线的版本过去几分钟在各地反馈如何”“昨天 23:40 左右这个投诉用户有没有异常”这种即时的开放性问题，传统的日志处理方案显得非常笨拙和低效，因为解答没有唯一套路，需要尝试下钻挖掘才能得出答案。复杂多变的实时数据分析需求，需要的是灵活快捷的响应处理。Splunk 公司正是凭借着自己在这个大数据细分领域的一枝独秀，成为百亿美元级的明星公司。但是 Splunk 每 GB 高达 4500 美元的报价，又让人望而却步。直到 ELK stack 出现后，大家才有了可选择的开源产品。

ELK stack 是以 Elasticsearch、Logstash、Kibana 三个开源软件为主的数据处理工具链，在实时数据检索和分析场合，三者通常是配合使用，而且又先后归于 Elastic.co 公司名下，故有此简称。

ELK stack 具有如下几个优点：

- 处理方式灵活。Elasticsearch 是实时全文索引，不需要像 Storm 那样预先编程才能使用。
- 配置简易上手。Elasticsearch 全部采用 JSON 接口，Logstash 是 Ruby DSL 设计，都是目前业界最通用的配置语法设计。
- 检索性能高效。虽然每次查询都是实时计算，但是优秀的设计和实现基本可以达到百亿级数据查询的秒级响应。
- 集群线性扩展。不管是 Elasticsearch 集群还是 Logstash 集群都是可以线性扩展的。
- 前端操作炫丽。Kibana 界面上，只需点击鼠标，就可以完成搜索、聚合功能，生成炫丽的仪表盘。

# 需要整本电子书，联系我QQ：3411317522

IV

2014年年初开QQ群交流ELK stack，发现网友们对ELK stack的原理常有误解误用，对实现的效果也多有不能理解或者因过多期望而失望之处。更令我惊奇的是，网友们分布之广，遍及传统企业和互联网公司、开发和运维领域、Linux和Windows平台，大家对非专精领域的知识，一般都缺乏了解，这也成为使用ELK stack的一个障碍。

为此，我决定写一本ELK stack技术指南，帮助大家厘清技术细节，分享一些实战案例。本书并不会逐一介绍ELK stack的全部聚合语法或者分词特性，而是从日志数据处理的角度介绍数据的解析、导入、可视化方式，讲解集群的稳定性和性能优化原理，剖析代码要点并提供ELK stack二次开发实例。

本书包括三大部分共19章，各部分可以独立阅读。但对于还没有大规模应用经验的新手，建议按顺序阅读全文。

## 第一部分 Logstash

第1章：入门示例。该章介绍Logstash及其插件的配置安装方法，自定义配置语言的设计用途，并为不熟悉Linux系统管理的开发人员介绍了多种后台运行方式。

第2章：插件配置。该章列举Logstash最常用的几十种插件，通过实际示例和效果，讲解各插件的配置细节和用途。

第3章：场景示例。该章以最常见的运维、网络、开发和数据库场景，介绍Logstash处理Nginx、Postfix、Ossec、Log4J、MySQL、Docker等日志的最佳实践。

第4章：性能与监控。了解Logstash的性能情况一直是个难题，该章从Logstash设计原理和JVM平台本质出发，介绍几种行之有效的检测和监控方案。

第5章：扩展方案。该章介绍采用Redis和Kafka完成Logstash水平扩展的方案，同时也介绍其他几种日志收集系统与Logstash的配合方式。

第6章：Logstash源码解析。该章解析Logstash源码中最重要的Pipeline设计，以及Logstash::Event的来龙去脉。

第7章：插件开发。该章以最常见的用户登录记录和地址库解析、Consul数据更新等需求，实际演示Logstash的自定义Filter、Input和Output插件的编写，同时还涉及了插件打包的RubyGems规范共有HttpClient功能项等细节。

## 第二部分 Elasticsearch

第8章：架构原理。该章从更高级的架构层面，介绍Elasticsearch分布式设计中涉及稳定性和高性能的部分原理，并由此引发相关的优化配置介绍。另外，还提供了一种针对时序数据索引的读写分离方案，适用于拥有少部分SSD设备的用户。

第9章：数据接口用例。该章介绍Elasticsearch的RESTful接口的基础知识，并针对常见的重建索引需求提供两种快速实现方案，对有Spark经验的读者介绍通过Spark Streaming

需要整本电子书，联系我QQ：3411317522

接口读写 Elasticsearch 的方法。

第 10 章：性能优化。该章介绍 Elasticsearch 在日志处理场景下的读写优化知识和官方推荐的 curator 工具，其中重点介绍了 Elasticsearch 中几种不同的 cache 的区别和有效场景。

第 11 章：测试和扩展方案。该章介绍 Elasticsearch 在生产环境中需要的一些周边工具，比如 Puppet 配置管理、Shield 权限管理、版本升级操作、别名切换流程设计等。

第 12 章：映射与模板的定制。该章详细介绍 Elasticsearch 中的核心类型及其对应的常见映射设置，以及如何通过动态模板简化映射定制操作的复杂度。

第 13 章：监控方案。Elasticsearch 作为一个分布式系统，也是有一定的运维难度的，因此其本身的监控也相当重要。该章介绍 Elasticsearch 自带的一系列监控接口，以及由此衍生的多种实时或长期的监控方案。

第 14 章：Elasticsearch 在运维监控领域的其他应用。该章介绍 Elasticsearch 在运维方面的其他运用方式，包括实时过滤接口、定时报警系统设计、时序数据存储和相关性排序等。

### 第三部分 Kibana

第 15 章：Kibana 的产品对比。该章介绍 Kibana 3 与 Kibana 4 之间，以及它们与 Hadoop、Splunk 之间的差异，方便读者在不同场景需求下选择更正确的工具。

第 16 章：Kibana 3。该章介绍 Kibana 3 的界面操作方式、面板的配置细节及其效果、动态仪表盘的高级用法，并提供了几种额外权限控制的部署方案。

第 17 章：Kibana 3 源码解析。该章以 index.html 为入口，介绍 Kibana 3 如何利用 angular.js、elastic.js 和 jquery.flot.js 三大框架实现单页应用。重点解析面板的实现过程，并分别演示了采用 facet 和 agg 接口开发一个 Kibana 3 面板的过程。

第 18 章：Kibana 4。该章介绍 Kibana 4 的安装部署和界面操作方式，重点介绍 Kibana 4 提供的几种可视化图表的配置细节和效果，并以几种场景的日志分析需求演示了 Kibana 4 全新的子聚合功能的效果。最后还介绍了一种采用 phantom.js 截图方式记录长期报表数据的方案。

第 19 章：Kibana 4 源码解析。该章介绍 Kibana 4 的界面实现，重点包括其内部 ORM 实现的 Courier 类、可视化绘图的 Vislib 类等。

## 致谢

我本人虽然接触 ELK stack 较早，但本身专于 Web 和 App 应用数据方面，动笔以来得到诸多朋友的帮助，详见后面“贡献者名单”。此外，还要特别感谢 Elastic.co 公司的曾勇 (medcl) 和吴晓刚 (Wood)，曾勇完成 Elasticsearch 在国内的启蒙式分享，并主办 Elasticsearch 中国

## 贡献者名单

- 感谢 crazw 完成 collectd 插件介绍章节。
- 感谢 松涛 完成 nxlog 场景介绍章节。
- 感谢 LeiTu 完成 logstash-forwarder 介绍章节。
- 感谢 jingbli 完成 kafka 插件介绍章节。
- 感谢 林鹏 完成 ossec 场景介绍章节。
- 感谢 cameluo 完成 shield 介绍章节。
- 感谢 tuxknight 指出 hdfs 章节笔误。
- 感谢 lemontree 完成 marvel 介绍章节。
- 感谢 childe 完成 Kibana 的 CAS 认证介绍、Elasticsearch 的读写分离和别名应用章节。
- 感谢 gnuhpc 完善 logstash-output-elasticsearch 配置细节和 logstash-forwarder-java 在 AIX 系统上的介绍。

前 言

第一部分 Logstash

第 1 章 入门示例 ..... 3

- 1.1 下载安装 ..... 3
- 1.2 Hello World ..... 5
- 1.3 配置语法 ..... 7
  - 1.3.1 语法 ..... 8
  - 1.3.2 命令行参数 ..... 10
- 1.4 插件安装 ..... 11
- 1.5 长期运行方式 ..... 12

第 2 章 插件配置 ..... 15

- 2.1 输入插件 ..... 15
  - 2.1.1 标准输入 ..... 16
  - 2.1.2 文件输入 ..... 17
  - 2.1.3 TCP 输入 ..... 18
  - 2.1.4 syslog 输入 ..... 19
  - 2.1.5 collectd 输入 ..... 21

- 2.2.1 JSON 编解码 ..... 24
- 2.2.2 多行事件编码 ..... 25
- 2.2.3 网络流编码 ..... 26
- 2.3 过滤器配置 ..... 28
  - 2.3.1 date 时间处理 ..... 28
  - 2.3.2 grok 正则捕获 ..... 30
  - 2.3.3 GeoIP 地址查询 ..... 33
  - 2.3.4 JSON 编解码 ..... 34
  - 2.3.5 key-value 切分 ..... 35
  - 2.3.6 metrics 数值统计 ..... 36
  - 2.3.7 mutate 数据修改 ..... 37
  - 2.3.8 随心所欲的 Ruby 处理 ..... 42
  - 2.3.9 split 拆分事件 ..... 43
  - 2.3.10 elapsed ..... 43
- 2.4 输出插件 ..... 44
  - 2.4.1 输出到 Elasticsearch ..... 44
  - 2.4.2 发送 email ..... 49
  - 2.4.3 调用系统命令执行 ..... 50
  - 2.4.4 保存成文件 ..... 50
  - 2.4.5 报警发送到 Nagios ..... 51
  - 2.4.6 statsd ..... 52
  - 2.4.7 标准输出 stdout ..... 54

2.4.8 TCP 发送数据	55	4.1.1 配置示例	77
2.4.9 输出到 HDFS	55	4.1.2 使用方式	78
<b>第 3 章 场景示例</b>	<b>57</b>	4.1.3 额外的话	79
3.1 Nginx 访问日志	57	4.2 监控方案	79
3.1.1 grok 处理方式	57	4.2.1 logstash-input-heartbeat 心跳	
3.1.2 split 处理方式	58	检测方式	80
3.1.3 json 格式	61	4.2.2 JMX 启动参数方式	81
3.1.4 syslog 方式发送	62	<b>第 5 章 扩展方案</b>	<b>83</b>
3.2 Nginx 错误日志	62	5.1 通过 Redis 队列扩展	84
3.3 Postfix 日志	63	5.1.1 读取 Redis 数据	84
3.4 Ossec 日志	64	5.1.2 采用 list 类型扩展 Logstash	85
3.4.1 配置所有 Ossec agent 采用		5.1.3 输出到 Redis	86
syslog 输出	64	5.2 通过 Kafka 队列扩展	87
3.4.2 配置 Logstash	65	5.2.1 Logstash1.4 版本插件的安装	88
3.4.3 推荐 Kibana 仪表盘	65	5.2.2 Input 配置	88
3.5 Windows 系统日志	67	5.2.3 Output 配置	90
3.5.1 采集端配置	67	5.3 logstash-forwarder	91
3.5.2 接收解析端配置	68	5.3.1 Indexer 端配置	91
3.6 Java 日志	69	5.3.2 Shipper 端配置	92
3.6.1 Log4J 配置	70	5.3.3 AIX 上的 logstash-forwarder-	
3.6.2 Logstash 配置	70	java	93
3.6.3 异常堆栈测试验证	70	5.4 Rsyslog	95
3.6.4 JSON Event layout	71	5.4.1 常用模块介绍	95
3.7 MySQL 慢查询日志	73	5.4.2 与 Logstash 合作	96
3.8 Docker 日志	74	5.4.3 Mmexternal 模块	97
3.8.1 记录到主机磁盘	75	5.5 Nxlog	99
3.8.2 通过 logspout 收集	75	5.6 Heka	101
<b>第 4 章 性能与监控</b>	<b>77</b>	5.7 Fluentd	102
4.1 需要整本电子书，联系我QQ：3411317522	77	5.7.1 配置示例	103
		Fluentd 插件	104

5.8	Message::Passing	104	8.2	segment merge 的影响	132
<b>第 6 章 Logstash 源码解析</b>		107	8.2.1	归并线程配置	133
6.1	Pipeline	108	8.2.2	归并策略	134
6.2	Plugins	109	8.2.3	optimize 接口	134
6.2.1	Input 中的 Codec	110	8.3	routing 和 replica 的读写过程	134
6.2.2	Output 中的 Worker	111	8.3.1	路由计算	134
<b>第 7 章 插件开发</b>		113	8.3.2	副本一致性	135
7.1	插件格式	113	8.4	shard 的 allocate 控制	136
7.2	插件的关键方法	114	8.4.1	reroute 接口	138
7.3	插件打包	115	8.4.2	冷热数据的读写分离	138
7.4	Filter 插件开发示例	116	8.5	自动发现的配置	139
7.4.1	mmdb 数据库的生成方法	116	8.5.1	multicast 方式	140
7.4.2	LogStash::Filters::Mmdb 实现	117	8.5.2	unicast 方式	140
7.4.3	logstash-filter-mmdb 打包	119	<b>第 9 章 数据接口用例</b>		141
7.5	Input 插件开发示例	119	9.1	增删改查操作	141
7.5.1	FileWatch 模块原理	120	9.2	搜索请求	143
7.5.2	LogStash::Inputs::Utmp 实现	121	9.2.1	全文搜索	143
7.6	Output 插件开发示例	124	9.2.2	聚合请求	145
<b>第二部分 Elasticsearch</b>			9.3	脚本	147
<b>第 8 章 架构原理</b>		129	9.3.1	动态提交	147
8.1	准实时索引的实现	129	9.3.2	固定文件	147
8.1.1	动态更新的 Lucene 索引	129	9.3.3	其他语言	148
8.1.2	利用磁盘缓存实现的 准实时检索	130	9.4	重建索引	148
8.1.3	translog 提供的磁盘同步		9.4.1	Perl 客户端	149
			9.4.2	用 Logstash 重建索引	149
			9.5	Spark Streaming 交互	150
<b>第 10 章 性能优化</b>		153	<b>第 10 章 性能优化</b>		153
			10.1	bulk 提交	153
<b>需要整本电子书，联系我QQ：3411317522</b>		154	<b>需要整本电子书，联系我QQ：3411317522</b>		154

10.1.2	UDP 方式	154	12.3	自定义字段映射	184
10.2	gateway 配置	155	12.3.1	精确索引	184
10.3	集群状态维护	156	12.3.2	时间格式	185
10.4	缓存	160	12.3.3	多重索引	185
10.4.1	filter 缓存	160	12.4	特殊字段	186
10.4.2	shard query 缓存	161	12.5	动态模板映射	186
10.5	字段数据	162	12.6	索引模板	187
10.5.1	Circuit Breaker	162			
10.5.2	doc-values	163	<b>第 13 章 监控方案</b>		189
10.6	curator 工具	163	13.1	监控相关接口	189
10.6.1	参数介绍	163	13.1.1	集群健康状态	189
10.6.2	常用示例	165	13.1.2	节点状态	191
			13.1.3	索引状态	199
<b>第 11 章 测试和扩展方案</b>		167	13.1.4	等待执行的任务	200
11.1	测试方案	167	13.1.5	cat 接口的命令行使用	201
11.2	多集群互联	168	13.2	日志记录	204
11.3	puppet-elasticsearch 模块的 使用	171	13.3	实时 bigdesk 方案	205
11.3.1	安装和配置示例	171	13.4	官方 marvel 方案	207
11.3.2	配置解释	171	13.4.1	安装和卸载	208
11.4	计划内停机升级的操作流程	172	13.4.2	配置	208
11.5	Shield 权限管理	174	13.4.3	访问	209
11.5.1	Shield 架构	174	13.4.4	面板定制示例	209
11.5.2	安装部署	175	13.5	Zabbix trapper 方案	212
11.6	别名的应用	176	13.5.1	安装配置	212
11.6.1	索引更名时的无缝切换	177	13.5.2	模板应用	213
11.6.2	限制索引数据部分可读	178	<b>第 14 章 Elasticsearch 在运维监控 领域的其他应用</b>		215
<b>第 12 章 映射与模板的定制</b>		181	14.1	Percolator 接口	215
12.1	映射的增删改查	181	14.2	Watcher 报警	217
12.2	需要整本电子书，联系我QQ：3411317522	220			

- 14.3.1 安装部署 ..... 220
- 14.3.2 配置示例 ..... 220
- 14.3.3 dashboard 效果 ..... 221
- 14.3.4 Kibana 3 拓扑图 ..... 223
- 14.4 时序数据库 ..... 224
- 14.5 Etsy 的 Kale 异常检测 ..... 226

第三部分 Kibana

第 15 章 Kibana 的产品对比 ..... 231

- 15.1 Kibana 3 的设计思路和功能 ..... 231
- 15.2 Kibana 4 的设计思路和功能 ..... 232
- 15.3 与 Hadoop 体系的区别 ..... 232
- 15.4 Splunk 场景参考 ..... 233

第 16 章 Kibana 3 ..... 235

- 16.1 Kibana 3 入门 ..... 235
  - 16.1.1 准备工作 ..... 236
  - 16.1.2 界面介绍 ..... 236
  - 16.1.3 跨域访问注意事项 ..... 239
- 16.2 config.js 配置 ..... 240
- 16.3 页面布局 ..... 240
  - 16.3.1 请求和过滤 ..... 241
  - 16.3.2 行和面板 ..... 244
- 16.4 各面板功能 ..... 251
  - 16.4.1 histogram ..... 252
  - 16.4.2 table ..... 262
  - 16.4.3 map ..... 266
  - 16.4.4 bettermap ..... 267

- 16.4.6 column ..... 274
- 16.4.7 stats ..... 274
- 16.4.8 query ..... 275
- 16.4.9 trend ..... 276
- 16.4.10 text ..... 277
- 16.4.11 sparklines ..... 278
- 16.4.12 hits ..... 279
- 16.4.13 goal ..... 279

16.5 仪表盘的保存和载入 ..... 279

- 16.5.1 保存仪表盘 ..... 280
- 16.5.2 加载仪表盘 ..... 280
- 16.5.3 分享仪表盘 ..... 280
- 16.5.4 保存成静态仪表盘 ..... 281

16.6 自定义仪表盘功能 ..... 281

- 16.6.1 schema 简介 ..... 281
- 16.6.2 模板化 template 仪表盘 ..... 288
- 16.6.3 脚本化 scripted 仪表盘 ..... 289

16.7 认证授权 ..... 290

- 16.7.1 用 Nginx 实现基础的认证 ..... 290
- 16.7.2 用 Node.js 实现基于 CAS 的认证 ..... 292
- 16.7.3 用 Perl 实现认证和用户授权 ..... 293

第 17 章 Kibana 3 源码解析 ..... 297

- 17.1 源码目录结构 ..... 297
- 17.2 入口和模块依赖 ..... 300
- 17.3 控制器和服务 ..... 302
  - 17.3.1 dashboard ..... 303
  - 17.3.2 querySrv ..... 303
  - 17.3.3 Facets ..... 304

17.3.4	fields	304	18.3.5	查看文档数据	330
17.3.5	esVersion	304	18.4	各种可视化功能	332
17.4	面板指令	304	18.4.1	area	333
17.4.1	添加面板	304	18.4.2	table	336
17.4.2	展示面板	305	18.4.3	line	337
17.5	面板实现	306	18.4.4	Markdown	338
17.5.1	module.js	306	18.4.5	metric	338
17.5.2	module.html	308	18.4.6	pie	339
17.5.3	editor.html	309	18.4.7	tile map	339
17.6	用 facet 接口开发一个 range panel	309	18.4.8	vertical bar	340
17.6.1	代码实现	310	18.5	仪表盘功能	341
17.6.2	面板效果	313	18.5.1	开始	342
17.7	用 agg 接口开发一个 percentile panel	313	18.5.2	容器功能	343
17.7.1	代码实现要点	316	18.5.3	修改可视化	346
17.7.2	面板效果	318	18.6	Setting 功能	346
<b>第 18 章 Kibana 4</b> 319			18.6.1	创建一个连接到 Elasticsearch 的索引模式	346
18.1	安装、配置和运行	320	18.6.2	创建一个脚本化字段	349
18.2	生产环境部署	323	18.6.3	设置高级参数	350
18.2.1	Nginx 代理配置	323	18.6.4	管理已保存的搜索、可视化 和仪表盘	350
18.2.2	配置 Kibana 和 shield 一起 工作	324	18.7	设置 Kibana 服务器属性	351
18.2.3	开启 SSL	325	18.8	常用 sub agg 示例	352
18.2.4	控制访问权限	325	18.8.1	函数堆栈链分析	352
18.3	Discover 功能	326	18.8.2	分图统计	355
18.3.1	设置时间过滤器	326	18.8.3	TopN 的时序趋势图	356
18.3.2	搜索数据	327	18.8.4	响应时间的百分占比 趋势图	358
18.3.3	按字段过滤	328	18.8.5	响应时间的概率分布在不同 时段的相似度对比	359
18.3.4	按时间过滤	328	18.8.6	响应时间的分布	360
18.3.5	查看文档数据	330	<b>需要整本电子书，联系我QQ：3411317522</b>		

<b>第 19 章 Kibana 4 源码解析</b> .....	363	19.4 Visualize 解析	377
19.1 Kibana 索引的数据结构	364	19.4.1 vis_types 实现	378
19.2 主页入口	365	19.4.2 savedVisualizations 实现	384
19.2.1 index.js 解析	365	19.4.3 Visualize 实现	384
19.2.2 Courier 类	367	19.4.4 VisEditorSidebar 实现	385
19.2.3 路径记忆功能的实现	370	19.5 Dashboard 解析	387
19.2.4 标签页应用的加载	371	19.6 Setting 解析	389
19.3 Discover 解析	374		

需要整本电子书，联系我QQ：3411317522



第一部分 *Part 1*

# Logstash

- 第 1 章 入门示例
- 第 2 章 插件配置
- 第 3 章 场景示例
- 第 4 章 性能与监控
- 第 5 章 扩展方案
- 第 6 章 Logstash 源码解析
- 第 7 章 插件开发

需要整本电子书，联系我QQ：3411317522

Logstash is a tool for managing events and logs. You can use it to collect logs, parse them, and store them for later use (like, for searching). -- <http://logstash.net>

Logstash 项目诞生于 2009 年 8 月 2 日。其作者是世界著名的运维工程师 乔丹·西塞 (Jordan Sissel)，乔丹·西塞当时是著名虚拟主机托管商 DreamHost 的员工，还发布过非常棒的软件打包工具 fpm，并主办着一年年度的 Sysadmin Advent Calendar (advent calendar 文化源自基督教氛围浓厚的 Perl 社区，在每年圣诞来临的 12 月举办，从 12 月 1 日起至 12 月 24 日止，每天发布一篇小短文介绍主题相关技术)。

Logstash 动手很早，对比一下，Scribed 诞生于 2008 年，Flume 诞生于 2010 年，Graylog2 诞生于 2010 年，Fluentd 诞生于 2011 年。Scribed 在 2011 年进入半死不活的状态，大大激发了其他各种开源日志收集处理框架的蓬勃发展，Logstash 也从 2011 年开始进入 commit 密集期并延续至今。

作为一个系出名门的产品，Logstash 的身影多次出现在 Sysadmin Weekly 上，它和小伙伴们 Elasticsearch、Kibana 直接成为了和商业产品 Splunk 做比较的开源项目 (乔丹·西塞曾经在博客上承认设计想法来自 AWS 平台上最大的第三方日志服务商 Loggly，而 Loggly 两位创始人都是 Splunk 员工)。

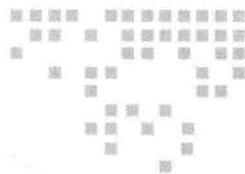
2013 年，Logstash 被 Elasticsearch 公司收购，ELK stack 正式成为官方用语。Elasticsearch 本身也是近两年最受关注的大数据项目之一，三次融资已经超过一亿美元。在 Elasticsearch 开发人员的共同努力下，Logstash 的发布机制、插件架构也愈发科学和合理。

## 社区文化

日志收集处理框架很多，如 Scribe 是 Facebook 出品，Flume 是 Apache 基金会项目，都算声名赫赫。但 Logstash 因乔丹·西塞的个人性格，形成了一套独特的社区文化。每一个在 Google Groups 的 Logstash-users 组里问答的人都会看到这么一句话：

Remember: if a new user has a bad time, it's a bug in Logstash.

所以，Logstash 是一个开放的、极其互助和友好的大家庭。如有问题，仅管在 Github Issue、Google Groups、Freenode#logstash Channel 上发问就好！



## 入门示例

什么是 Logstash？为什么要用 Logstash？怎么用 Logstash？这是本章将要介绍的内容。本章从最基础的知识着手，从以下几步介绍 Logstash 的必备知识。1) 下载安装。介绍 Logstash 软件的多种安装部署方式，并给出推荐的方式。2) 初次运行。通过 Hello World 示例，演示 Logstash 最简单的运用，解释其逻辑上的基础原理。3) 配置语法。介绍 Logstash 的 DSL 设计，Logstash 命令的运行参数。4) 插件安装。灵活和丰富的插件是 Logstash 最重要的优势。本节会介绍 Logstash 插件的安装方式。5) 长期运行方式。从初次终端测试到长期后台稳定运行，本节会介绍几种不同方案，供读者根据实际场景选择。

### 1.1 下载安装

#### 1. 下载

Logstash 从 1.5 版本开始，将核心代码和插件代码完全剥离，并重构了插件架构逻辑，所有插件都以标准的 Ruby Gem 包形式发布。不过，为了方便大家从 1.4 版本过渡，目前，官方依然发布打包有所有官方维护插件在内的软件包。只是不再发布类似原先 1.4 时代的 logstash-contrib.tar.gz 这样的软件包了。依然要使用社区插件的读者，请阅读稍后 1.4 节“插件安装”。

下载官方软件包的方式有以下几种：

压缩包方式



### ❑ Debian 平台

```
wget https://download.elastic.co/logstash/logstash/packages/debian/
logstash_1.5.1-1_all.deb
```

### ❑ Redhat 平台

```
wget https://download.elastic.co/logstash/logstash/packages/centos/
logstash-1.5.1-1.noarch.rpm
```

## 2. 安装

在上面这些包中，你可能更偏向使用 rpm、dpkg 等软件包管理工具来安装 Logstash，开发者在软件包里预定义了一些依赖。比如，logstash-1.5.1-1.narch 就依赖于 jre 包。

另外，软件包里还包含有一些很有用的脚本程序，比如 /etc/init.d/logstash。

如果你必须在一些很老的操作系统上运行 Logstash，那你只能用源代码包部署了，记住要自己提前安装好 Java：

```
yum install openjdk-jre
export JAVA_HOME=/usr/java
tar zxvf logstash-1.5.1.tar.gz
```

## 3. 最佳实践

但是真正的建议是：如果可以，请用 Elasticsearch 官方仓库来直接安装 Logstash！

### ❑ Debian 平台

```
wget -O - http://packages.elasticsearch.org/GPG-KEY-elasticsearch | apt-key add -
cat >> /etc/apt/sources.list <<EOF
deb http://packages.elasticsearch.org/logstash/1.5/debian stable main
EOF
apt-get update
apt-get install logstash
```

### ❑ Redhat 平台

```
rpm --import http://packages.elasticsearch.org/GPG-KEY-elasticsearch
cat > /etc/yum.repos.d/logstash.repo <<EOF
[logstash-1.5]
name=logstash repository for 1.5.x packages
baseurl=http://packages.elasticsearch.org/logstash/1.5/centos
gpgcheck=1
gpgkey=http://packages.elasticsearch.org/GPG-KEY-elasticsearch
enabled=1
EOF
yum clean all
```

## 1.2 Hello World

与绝大多数 IT 技术介绍一样，我们也以一个输出“Hello World”的形式开始学习 Logstash。

### 1. 命令行运行

在终端中，像下面这样运行命令来启动 Logstash 进程：

```
# bin/logstash -e 'input{stdin{}}output{stdout{codec=>rubydebug}}'
```

然后你会发现终端在等待你的输入。没问题，敲入 Hello World，回车，看看会返回什么结果！

```
{
  "message" =>"Hello World",
  "@version" =>"1",
  "@timestamp" =>"2014-08-07T10:30:59.937Z",
  "host" =>"raochenlindeMacBook-Air.local",
}
```

没错！就是这么简单。

### 2. 完整示例

命令行运行当然不是什么特别方便的使用法，所以绝大多数情况下，我们都是采用额外定义一个 logstash.conf 配置文件的方式来启动 Logstash。下面是我们的第一个完整版 logstash.conf 的示例：

```
input {
  stdin { }
}
output {
  stdout {
    codec => rubydebug {}
  }
  elasticsearch {
    embedded => true
  }
}
```

然后在终端上这样运行：

```
# bin/logstash -f logstash.conf
```

同样，还是输入一次 Hello World。你会看到和上一次一样的一段 Ruby 对象输出。但事实上，这个完整示例可不止如此。打开另一个终端，输入下面一行命令：

**需要整本电子书，联系我QQ：3411317522**

你会看到终端上输出下面这么一段内容：

```
{
  "took":15,
  "timed_out":false,
  "_shards":{
    "total":27,
    "successful":27,
    "failed":0
  },
  "hits":{
    "total":1,
    "max_score":0.095891505,
    "hits":[
      {
        "_index":"logstash-2015.08.22",
        "_type":"logs",
        "_id":"AU90s1eNgg_P5-w7SB32",
        "_score":0.095891505,
        "_source":{
          "message":"Hello World",
          "@version":"1",
          "@timestamp":"2014-08-07T10:30:59.937Z",
          "host":"raochenlindeMacBook-Air.local"
        }
      }
    ]
  }
}
```

如果你使用的是 Logstash-1.5 以下的版本，你还可以直接通过 bin/logstash web 命令，启动 Kibana 应用，然后在浏览器上访问 http://127.0.0.1:9292/ 地址，可以看到如图 1-1 所示的效果。我们在终端上输入的数据，就可以从页面上任意搜索了。

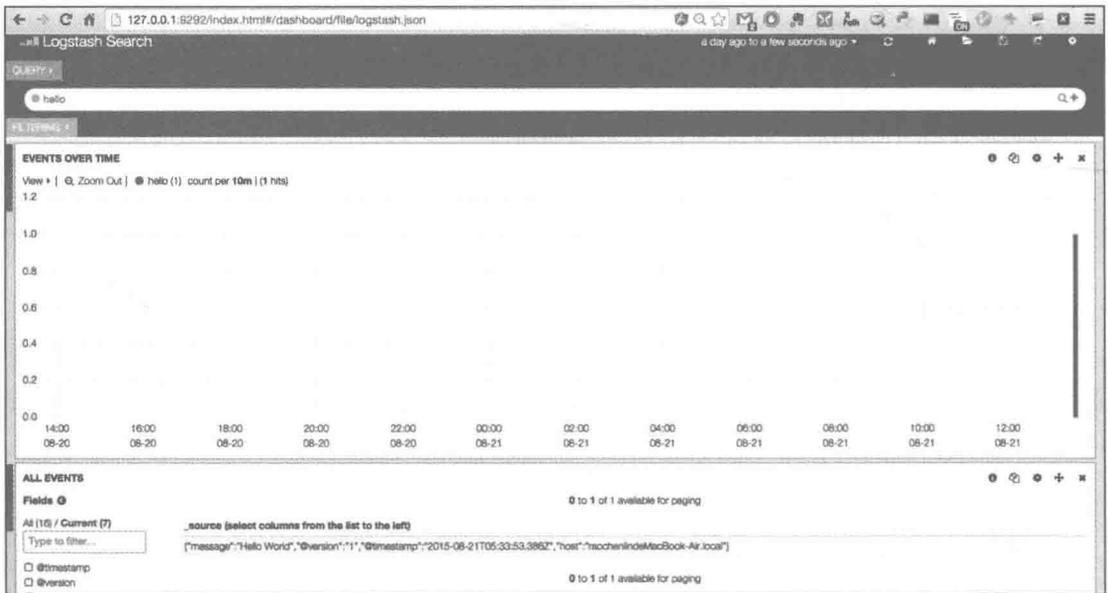


图 1-1 Kibana 上搜索的 hello world

如果你使用的是 Logstash-1.5 以上版本，Logstash 已经不再自带 Kibana 代码，你就只能自己安装部署一套 Kibana 程序了，相关内容，可以稍后阅读本书第三部分的介绍。

### 3. 解释

每位系统管理员都肯定写过很多类似这样的命令：cat randdata | awk '{print \$2}' | sort | uniq -c | tee sortdata。这个管道符 | 可以算是 Linux 世界最伟大的发明之一（另一个是“一切皆文件”）。

Logstash 就像管道符一样！

你输入（就像命令行的 cat）数据，然后处理过滤（就像 awk 或者 uniq 之类）数据，最后输出（就像 ~~到~~）到其他地方。

当然实际上，Logstash 是用不同的线程来实现这些的。如果你运行 `top` 命令然后按下 H 键，你就可以看到下面这样的输出：

```
PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
21401 root        16   0 1249m 303m 10m  S  18.6  0.2  866:25.46 |worker
21467 root        15   0 1249m 303m 10m  S   3.7  0.2  129:25.59 >elasticsearch.
21468 root        15   0 1249m 303m 10m  S   3.7  0.2  128:53.39 >elasticsearch.
21400 root        15   0 1249m 303m 10m  S   2.7  0.2  108:35.80 <file
21403 root        15   0 1249m 303m 10m  S   1.3  0.2   49:31.89 >output
21470 root        15   0 1249m 303m 10m  S   1.0  0.2   56:24.24 >elasticsearch.
```

如上例所示，Logstash 很温馨地给每类线程都取了名字，输入的叫 `<xx`，过滤的叫 `|xx`，输出的叫 `>xx`。

数据在线程之间以事件的形式流传。不要叫行，因为 Logstash 可以处理多行事件。

Logstash 会给事件添加一些额外信息。最重要的就是 `@timestamp`，用来标记事件的发生时间。因为这个字段涉及 Logstash 的内部流转，所以必须是一个 `joda` 对象，如果你尝试自己给一个字符串字段重命名为 `@timestamp` 的话，Logstash 会直接报错。所以，请使用 `logstash-filter-date` 插件来管理这个特殊字段。

此外，大多数时候，还可以见到另外几个：

- `host` 标记事件发生在哪里。
- `type` 标记事件的唯一类型。
- `tags` 标记事件的某方面属性。这是一个数组，一个事件可以有多个标签。

你可以随意给事件添加字段或者从事件里删除字段。事实上事件就是一个 Ruby 对象，或者更简单地理解为就是一个哈希也行。

每个 Logstash 过滤插件，都会有四个方法叫 `add_tag`、`remove_tag`、`add_field` 和 `remove_field`，它们在插件过滤匹配成功时生效。

---

## 推荐阅读

- 官网上“the life of an event”文档：<http://logstash.net/docs/1.4.2/life-of-an-event>
- Elastic{ON} 上《life of a logstash event》演讲：<https://speakerdeck.com/elastic/life-of-a-logstash-event>

---

## 1.3 配置语法

Logstash 社区通常习惯用 `Shipper`、`Broker` 和 `Indexer` 来描述数据流中不同进程各自的角色。如

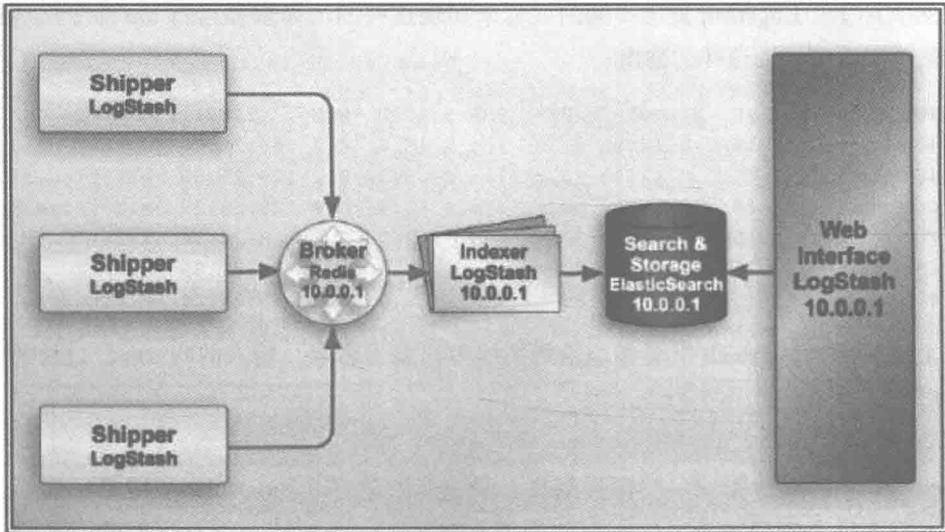


图 1-2 Logstash 角色说明

不过我见过很多运用场景里都没有用 Logstash 作为 Shipper，或者说没有用 Elasticsearch 作为数据存储，也就是说也没有 Indexer。所以，我们其实不需要这些概念。只需要学好怎么使用和配置 Logstash 进程，然后把它运用到你的日志管理架构中最合适它的位置就够了。

### 1.3.1 语法

Logstash 设计了自己的 DSL——有点像 Puppet 的 DSL，或许因为都是用 Ruby 语言写的吧——包括区域、注释、数据类型（布尔值、字符串数值、数组、哈希），条件判断、字段引用等。

#### 1. 区段 (section)

Logstash 用 {} 来定义区域。区域内可以包括插件区域定义，你可以在一个区域内定义多个插件。插件区域内则可以定义键值对设置。示例如下：

```
input {
  stdin {}
  syslog {}
}
```

#### 2. 数据类型

Logstash 支持少量的数据值类型：

- 布尔值 (bool)

## □ 字符串 (string)

```
host =>"hostname"
```

## □ 数值 (number)

```
port => 514
```

## □ 数组 (array)

```
match => ["datetime", "UNIX", "ISO8601"]
```

## □ 哈希 (hash)

```
options => {  
  key1 =>"value1",  
  key2 =>"value2"  
}
```

如果你用的版本低于 1.2.0，哈希的语法跟数组是一样的，像下面这样写：

```
match => [ "field1", "pattern1", "field2", "pattern2" ]
```

## 3. 字段引用 (field reference)

字段是 Logstash::Event 对象的属性。我们之前提过事件就像一个哈希一样，所以你可以想象字段就像一个键值对。

如果你想在 Logstash 配置中使用字段的值，只需把字段的名称写在中括号 [ ] 里就行了，这就叫“字段引用”。

对于“嵌套字段”（也就是多维哈希表，或者叫哈希的哈希），每层的字段名都写在 [ ] 里就可以了。比如，你可以从 geoip 里这样获取 longitude 值（是的，这是个笨办法，实际上有单独的字段专门存这个数据的）：

```
[geoip][location][0]
```

---

## 小知识

Logstash 的数组也支持倒序下标，即 [geoip][location][-1] 可以获取数组最后一个元素的值。

---

Logstash 还支持变量内插，在字符串里使用字段引用的方法是这样：

```
"the longitude is %[geoip][location][0]"
```

## 4. 条件判断 (condition)

Logstash 从 3.0 版开始支持条件判断的表达式。

表达式支持下面这些操作符：

- equality, etc: ==, !=, <, >, <=, >=
- regexp: =~, !~
- inclusion: in, not in
- boolean: and, or, nand, xor
- unary: !()

通常来说，你都会在表达式里用到字段引用。比如：

```
if "_grokparsefailure" not in [tags] {
} else if [status] !~ /^2\d\d/ and [url] == "/noc.gif" {
} else {
}
```

## 1.3.2 命令行参数

Logstash 提供了一个 shell 脚本叫 logstash 方便快速运行，下面介绍它支持的参数：

### 1. -e

意即“执行”。我们在“Hello World”的时候已经用过这个参数了。事实上你可以不写任何具体配置，直接运行 bin/logstash -e " 可达到相同效果。这个参数的默认值是下面这样：

```
input {
  stdin { }
}
output {
  stdout { }
}
```

### 2. --config 或 -f

意即“文件”。真实运用中，我们会写很长的配置，甚至可能超过 shell 所能支持的 1024 个字符长度。所以我们必把配置固化到文件里，然后通过 bin/logstash -f agent.conf 这样的形式来运行。

此外，Logstash 还提供一个方便我们规划和书写配置的小功能。你可以直接用 bin/logstash -f /etc/logstash.d/ 来运行。Logstash 会自动读取 /etc/logstash.d/ 目录下所有的文本文件，然后在自己内存里拼接成一个完整的大配置文件，再去执行。

### 3. --configtest 或 -t

意即“测试”。用来测试 Logstash 读取到的配置文件语法是否能正常解析。Logstash 配置语法是用 grammar.treetop 定义的。尤其是使用了上一条提到的读取目录方式的读者，尤其要提前测试。

### 4. --log 或 -l

意即“日志”。默认输出日志到 stderr。在生产环境中，你可以通过 bin/logstash

-l logs/logstash.log 命令来统一存储日志。

## 5. --filterworkers 或 -w

意即“工作线程”。Logstash 会运行多个线程。你可以用 `bin/logstash -w 5` 这样的方式强制 Logstash 为“过滤插件”（Logstash 目前还不支持输入插件的多线程，输出插件的多线程则是在配置内设置）运行 5 个线程。



**警告** Logstash 目前不支持对过滤器线程的监测管理。如果 `filterworker` 挂掉，Logstash 会处于一个无 `filter` 的僵死状态。这种情况在使用 `filter/ruby` 自己写代码时非常需要，很容易碰上 `NoMethodError: undefined method '*' for nil:NilClass` 错误。需要妥善处理，提前判断。

## 6. --pluginpath 或 -P

可以写自己的插件，然后用 `bin/logstash --pluginpath /path/to/own/plugins` 加载它们。



**警告** 如果你使用的 Logstash 版本在 1.5.0-rc3 到 1.5.3 之间，该参数一度被取消，请阅读本书稍后插件开发章节，改用本地 `gem` 插件安装形式。

## 7. --verbose

输出一定的调试日志。如果你使用的 Logstash 版本低于 1.3.0，则用 `bin/logstash -v` 来代替。

## 8. --debug

输出更多的调试日志。如果你使用的 Logstash 版本低于 1.3.0，则用 `bin/logstash -vv` 来代替。

## 1.4 插件安装

从 Logstash 1.5.0 版本开始，Logstash 将所有的插件都独立拆分成 `gem` 包。这样，每个插件都可以独立更新，不用等待 Logstash 自身做整体更新的时候才能使用了。

为了达到这个目标，Logstash 配置了专门的 `plugin` 管理命令。

`plugin` 命令用法说明如下：

Usage:

`bin/logstash [OPTIONS] SUBCOMMAND [ARG]`

```
Parameters:
  SUBCOMMAND      subcommand
  [ARG] ...       subcommand arguments

Subcommands:
  install          Install a plugin
  uninstall        Uninstall a plugin
  update           Install a plugin
  list             List all installed plugins

Options:
  -h, --help      print help
```

首先，你可以通过 `bin/plugin list` 查看本机现在有多少插件可用。（其实就在 `vendor/bundle/jruby/1.9/gems/` 目录下。）

然后，假如你看到 <https://github.com/logstash-plugins/> 下新发布了一个 `logstash-output-webhdfs` 模块（当然目前还没有）。打算试试，就只需运行如下命令：

```
bin/plugin install logstash-output-webhdfs
```

同样，假如是升级，只需运行如下命令即可：

```
bin/plugin update logstash-input-tcp
```

`bin/plugin` 不但可以通过 `rubygems` 平台安装插件，还可以读取本地路径的 `gem` 文件，这对自定义插件或者无外接网络的环境都非常有效：

```
bin/plugin install /path/to/logstash-filter-crash.gem
```

执行成功以后。你会发现，`logstash-1.5.0` 目录下的 `Gemfile` 文件最后会多出一段内容：

```
gem "logstash-filter-crash", "1.1.0", :path =>"vendor/local_gems/d354312c/logstash-filter-mweibocrash-1.1.0"
```

同时 `Gemfile.jruby-1.9.lock` 文件开头也会多出一段内容，如下所示：

```
PATH
remote: vendor/local_gems/d354312c/logstash-filter-crash-1.1.0
specs:
  logstash-filter-crash (1.1.0)
  logstash-core (>= 1.4.0, < 2.0.0)
```

## 1.5 长期运行方式

完成上一节的初次运行后，你肯定会发现一点：一旦你按下 `Ctrl+C`，停下标准输入输出，

**提示** 本章节问题对于一个运维来说应该属于基础知识，鉴于 ELK 用户很多其实不是运维，添加这段内容。

办法有很多种，下面介绍四种最常用的办法。

## 1. 标准的 service 方式

采用 RPM、DEB 发行包安装的读者，推荐采用这种方式。发行包内，都自带有 sysV 或者 systemd 风格的启动程序 / 配置，你只需要直接使用即可。以 RPM 为例，`/etc/init.d/logstash` 脚本中，会加载 `/etc/init.d/functions` 库文件，利用其中的 `daemon` 函数，将 Logstash 进程作为后台程序运行。

所以，你只需把自己写好的配置文件统一放在 `/etc/logstash/` 目录下（注意目录下所有配置文件都应该是 `.conf` 结尾，且不能有其他文本文件存在，因为 `logstash agent` 启动的时候是读取全文件夹的），然后运行 `service logstash start` 命令即可。

## 2. 最基础的 nohup 方式

这是最简单的方式，也是 Linux 新手们很容易搞混淆的一个经典问题：

```
command
command > /dev/null
command > /dev/null 2>&1
command &
command > /dev/null &
command > /dev/null 2>&1 &
command &> /dev/null
nohup command &> /dev/null
```

请回答以上命令的异同……

具体不一一解释了。直接说答案，想要维持一个长期后台运行的 Logstash，你需要同时在命令前面加 `nohup`，后面加 `&`。

## 3. 更优雅的 screen 方式

`screen` 算是 Linux 运维一个中高级技巧。通过 `screen` 命令创建的环境下运行的终端命令，其父进程不是 `sshd` 登录会话，而是 `screen`。这样就可以既避免用户退出进程消失的问题，又随时能重新接管回终端继续操作。

创建独立的 `screen` 命令如下：

```
screen -dmS elkscreen_1
```

连接进入已创建的 `elkscreen_1` 的命令如下：

```
screen -r elkscreen_1
```

然而你并不会得到一个一模一样的终端，在运行 `logstash` 之后，不要按 `Ctrl-C`，而是按

Ctrl+A+D 键，断开环境。想重新接管，依然用 `screen -r elkscreen_1` 即可。

如果创建了多个 `screen`，查看列表命令如下：

```
screen -list
```

#### 4. 最推荐的 daemontools 方式

不管是 `nohup` 还是 `screen`，都不是可以很方便管理的方式，在运维管理一个 ELK 集群的时候，必须寻找一种尽可能简洁的办法。所以，对于需要长期后台运行的大量程序（注意大量，如果就一个进程，还是学习一下怎么写 `init` 脚本吧），推荐大家使用一款 `daemontools` 工具。

`daemontools` 是一个软件名称，不过配置略复杂。所以这里我其实是用其名称来指代整个同类产品，包括但不限于 Python 实现的 `supervisord`，Perl 实现的 `ubic`，Ruby 实现的 `god` 等。

以 `supervisord` 为例，因为这个出来得比较早，可以直接通过 EPEL 仓库安装。

```
yum -y install supervisord --enablerepo=epel
```

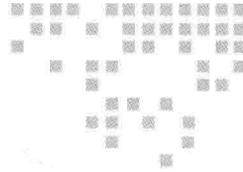
在 `/etc/supervisord.conf` 配置文件里添加内容，定义你要启动的程序，如下所示：

```
[program:elkpro_1]
environment=LS_HEAP_SIZE=5000m
directory=/opt/logstash
command=/opt/logstash/bin/logstash -f /etc/logstash/pro1.conf -w 10 -l /var/
    log/logstash/pro1.log
[program:elkpro_2]
environment=LS_HEAP_SIZE=5000m
directory=/opt/logstash
command=/opt/logstash/bin/logstash -f /etc/logstash/pro2.conf -w 10 -l /var/
    log/logstash/pro2.log
```

然后启动 `service supervisord start` 即可。

Logstash 会以 `supervisord` 子进程的身份运行，你还可以使用 `supervisorctl` 命令，单独控制一系列 Logstash 子进程中某一个进程的启停操作：

```
supervisorctl stop elkpro_2
```



## 插件配置

插件是 Logstash 最大的特色。各种不同的插件源源不断地被创造出来，发布到社区中供大家使用。本章将按照插件的类别，对一般场景下的一些常用插件做详细的配置和用例介绍。本章介绍的插件包括：1) 输入插件。基于 shipper 端场景，主要介绍 STDIN、TCP、File 等插件。2) 编解码插件。编解码通常是会被遗忘的环节，但是运用好了，会大大提高工作效率，本节介绍最常用的 JSON 和 multiline 插件。3) 过滤器插件。名为过滤器，其实各种数据裁剪和计算都可以在这类插件里完成，是 Logstash 最强大的一环。本节会详细介绍 grok、date、mutate、ruby、metrics 等插件的妙用。4) 输出插件。Logstash 虽然经常跟 Elasticsearch 并称，但是作为一个日志传输框架，它其实可以输出数据到各种不同的地方。比如 Graphite、HDFS、Nagios 等等。本节会介绍这些常用的输出插件用法。

### 2.1 输入插件

在“Hello World”示例中，我们已经见到并介绍了 Logstash 的运行流程和配置的基础语法。从这章开始，我们就要逐一介绍 Logstash 流程中比较常用的一些插件，并在介绍中针对其主要适用的场景、推荐的配置，作一些说明。

限于篇幅，接下来内容中，配置示例不一定能贴完整。请记住一个原则：Logstash 配置一定要有一个 input 和一个 output。在演示过程中，如果没有写明 input，默认就会使用“Hello World”里我们已经演示过的 logstash-input-stdin，同理，没有写明的 output 就是 logstash-output-stdout。

## 2.1.1 标准输入

我们已经见过好几个示例使用 `stdin` 了。这也应该是 Logstash 里最简单和基础的插件了。所以，在这段中，我们先介绍一些未来每个插件都会有的一些方法。

配置示例如下：

```
input {
  stdin {
    add_field => {"key" =>"value"}
    codec =>"plain"
    tags => ["add"]
    type =>"std"
  }
}
```

用上面的新 `stdin` 设置重新运行一次最开始的 Hello World 示例。我建议大家把整段配置都写入一个文本文件，然后运行命令：`bin/logstash -f stdin.conf`。输入“Hello World”并回车后，你会在终端看到如下输出：

```
{
  "message" =>"hello world",
  "@version" =>"1",
  "@timestamp" =>"2014-08-08T06:48:47.789Z",
  "type" =>"std",
  "tags" => [
    [0] "add"
  ],
  "key" =>"value",
  "host" =>"raochenlindeMacBook-Air.local"
}
```

`type` 和 `tags` 是 Logstash 事件中两个特殊的字段。通常来说，我们会在“输入区段”中通过 `type` 来标记事件类型——我们肯定是提前能知道这个事件属于什么类型的。而 `tags` 则是在数据处理过程中，由具体的插件来添加或者删除的。

最常见的用法是像下面这样：

```
input {
  stdin {
    type =>"web"
  }
}
filter {
  if [type] == "web" {
    grok {
      match => ["message", %{COMBINEDAPACHELOG}]
    }
  }
}
```

```
output {
  if "_grokparsefailure" in [tags] {
    nagios_nsca {
      nagios_status =>"1"
    }
  } else {
    elasticsearch {
    }
  }
}
```

看起来蛮复杂的，对吧？

继续学习，你也可以写出来的。

### 2.1.2 文件输入

分析网站访问日志应该是一个运维工程师最常见的工作了。所以我们先学习一下怎么用 Logstash 来处理日志文件。

Logstash 使用一个名叫 FileWatch 的 Ruby Gem 库来监听文件变化。这个库支持 glob 展开文件路径，而且会记录一个叫 `.sincedb` 的数据库文件来跟踪被监听日志文件的当前读取位置。所以，不要担心 Logstash 会漏过你的数据。



**提示** `sincedb` 文件中记录了每个被监听的文件的 `inode`, `major number`, `minor number` 和 `pos`。

配置示例如下：

```
input {
  file {
    path => ["/var/log/*.log", "/var/log/message"]
    type =>"system"
    start_position =>"beginning"
  }
}
```

有一些比较有用的配置项，可以用来指定 FileWatch 库的行为：

- ❑ `discover_interval`: Logstash 每隔多久去检查一次被监听的 `path` 下是否有新文件，默认值是 15 秒。
- ❑ `exclude`: 不想被监听的文件可以排除出去，这里跟 `path` 一样支持 glob 展开。
- ❑ `sincedb_path`: 如果你不想用默认的 `$HOME/.sincedb` (Windows 平台上为 `%USERPROFILE%\sincedb`，该变量默认值是: `C:\Windows\System32\config\systemprofile`)，可以通过这个配置定义 `sincedb` 文件到其他位置。

❑ `read_timeout`: Logstash 每隔多久去读 `sincedb` 文件，默认是 5 秒。

- ❑ `stat_interval`：Logstash 每隔多久检查一次被监听文件状态（是否有更新），默认是 1 秒。
- ❑ `start_position`：Logstash 从什么位置开始读取文件数据，默认是结束位置，也就是说，Logstash 进程会以类似 `tail-F` 的形式运行。如果你是要导入原有数据，把这个设定改成 `"beginning"`，Logstash 进程就从头开始读取，有点类似于 `cat`，但是读到最后一行不会终止，而是继续变成 `tail -F`。

注意事项如下：

1) 通常你要导入原有数据进 Elasticsearch 的话，你还需要 `filter/date` 插件来修改默认的 `"@timestamp"` 字段值。稍后会学习这方面的知识。

2) `FileWatch` 只支持文件的绝对路径，而且会不自动递归目录。所以有需要的话，请用数组方式都写明具体哪些文件。

3) `LogStash::Inputs::File` 只是在进程运行的注册阶段初始化一个 `FileWatch` 对象。所以它不能支持类似 `fluentd` 那样的 `path => "/path/to/%{+yyyy/MM/dd/hh}.log"` 写法。达到相同目的，你只能写成 `path => "/path/to/**/*/*/*.log"`。`FileWatch` 模块提供了一个稍微简单一点的写法：`/path/to/**/*log`，用 `**` 来缩写表示递归全部子目录。

4) `start_position` 仅在该文件从未被监听过的时候起作用。如果 `sincedb` 文件中已经有这个文件的 `inode` 记录了，那么 Logstash 依然会从记录过的 `pos` 开始读取数据。所以重复测试的时候每回需要删除 `sincedb` 文件。

5) 因为 Windows 平台上没有 `inode` 的概念，Logstash 某些版本在 Windows 平台上监听文件不是很靠谱。Windows 平台上，推荐考虑使用 `nxlog` 作为收集端，参阅本书稍后章节。

## 2.1.3 TCP 输入

未来你可能会用 Redis 服务器或者其他的消息队列系统来作为 Logstash Broker 的角色。不过 Logstash 其实也有自己的 TCP/UDP 插件，在临时任务的时候，也算能用，尤其是测试环境。



**注意** 虽然 `LogStash::Inputs::TCP` 用 Ruby 的 `Socket` 和 `OpenSSL` 库实现了高级的 SSL 功能，但 Logstash 本身只能在 `SizedQueue` 中缓存 20 个事件。这就是我们建议在生产环境中换用其他消息队列的原因。

配置示例如下：

```
input {
  tcp {
    port => 8888
```

```
        ssl_enable => false
    }
}
```

目前来看，Logstash::Inputs::TCP 最常见的用法就是配合 nc 命令导入旧数据。在启动 Logstash 进程后，在另一个终端运行如下命令即可导入数据：

```
# nc 127.0.0.1 8888 < olddata
```

这种做法比用 Logstash::Inputs::File 好，因为当 nc 命令结束，我们就知道数据导入完毕了。而用 input/file 方式，Logstash 进程还会一直等待新数据输入被监听的文件，不能直接看出是否任务完成了。

## 2.1.4 syslog 输入

syslog 可能是运维领域最流行的数据传输协议了。当你想从设备上收集系统日志的时候，syslog 应该会是你的第一选择。尤其是网络设备，比如思科中 syslog 几乎是唯一可行的办法。

我们这里不解释如何配置你的 syslog.conf、rsyslog.conf 或者 syslog-ng.conf 来发送数据，而只讲如何把 Logstash 配置成一个 syslog 服务器来接收数据。

有关 rsyslog 的用法，稍后的类型项目一节中，会有更详细的介绍。

配置示例如下：

```
input {
  syslog {
    port =>"514"
  }
}
```

作为最简单的测试，我们先暂停一下本机的 syslogd (或 rsyslogd) 进程，然后启动 Logstash 进程 (这样就不会有端口冲突问题)。现在，本机的 syslog 就会默认发送到 Logstash 里了。我们可以用自带的 logger 命令行工具发送一条“Hello World”信息到 syslog 里 (即 Logstash 里)。看到的 Logstash 输出像下面这样：

```
{
  "message" =>"Hello World",
  "@version" =>"1",
  "@timestamp" =>"2014-08-08T09:01:15.911Z",
  "host" =>"127.0.0.1",
  "priority" =>31,
  "timestamp" =>"Aug  8 17:01:15",
  "logsource" =>"raochenlindeMacBook-Air.local",
  "program" =>"com.apple.metadata.mdflagwriter",
  "pid" =>"381",
  "s
```

```
"facility" =>3,
"facility_label" =>"system",
"severity_label" =>"Debug"
}
```

Logstash 是用 UDPSocket、TCPSTServer 和 LogStash::Filters::Grok 来实现 LogStash::Inputs::Syslog 的。所以你其实可以直接用 Logstash 配置实现一样的效果，如下所示：

```
input {
  tcp {
    port =>"8514"
  }
}
filter {
  grok {
    match => ["message", "%{SYSLOGLINE}"]
  }
  syslog_pri { }
}
```

建议在使用 LogStash::Inputs::Syslog 的时候走 TCP 协议来传输数据。

因为具体实现中，UDP 监听器只用了一个线程，而 TCP 监听器会在接收每个连接的时候都启动新的线程来处理后续步骤。

如果你已经在使用 UDP 监听器收集日志，用下行命令检查你的 UDP 接收队列大小：

```
# netstat -plnu | awk 'NR==1 || $4~/:514$/ {print $2}'
Recv-Q
228096
```

228096 是 UDP 接收队列的默认最大大小，这时候 linux 内核开始丢弃数据包了！

强烈建议使用 LogStash::Inputs::TCP 和 LogStash::Filters::Grok 配合实现同样的 syslog 功能！

虽然 LogStash::Inputs::Syslog 在使用 TCPSTServer 的时候可以采用多线程处理数据的接收，但是在同一个客户端数据的处理中，其 grok 和 date 是一直在该线程中完成的，这会导致总体上的处理性能几何级的下降——经过测试，TCPSTServer 每秒可以接收 50000 条数据，而在同一线程中启用 grok 后每秒只能处理 5000 条，再加上 date 只能达到 500 条！

才将这两步拆分到 filters 阶段后，Logstash 支持对该阶段插件单独设置多线程运行，大大提高了总体处理性能。在相同环境下，logstash -f tcp.conf -w 20 的测试中，总体处理性能可以达到每秒 30 000 条数据！



注意 测试采用 Logstash 作者提供的命令：

```
yes "<44>May 19 18:30:17 snack jls: foo bar 32" | nc localhost 3000
```

出处见：<https://github.com/jordansissel/experiments/blob/master/ruby/jruby-netty/syslog->

如果你实在没法切换到 TCP 协议，可以自己写程序，或者使用其他基于异步 I/O 框架（比如 libev）的项目。下面是一个简单的异步 I/O 实现 UDP 监听数据输入 Elasticsearch 的示例：<https://gist.github.com/chenryn/7c922ac424324ee0d695>。

## 2.1.5 collectd 输入

collectd 是一个守护（daemon）进程，用来收集系统性能和提供各种存储方式来存储不同值的机制。它会在系统运行和存储信息时周期性的统计系统的相关统计信息。利用这些信息有助于查找当前系统性能瓶颈（如作为性能分析 performance analysis）和预测系统未来的 load（如能力部署 capacity planning）等

下面简单介绍一下 collectd 的部署以及与 Logstash 对接的相关配置实例。

### 1. collectd 的安装

解决依赖如下：

```
rpm -ivh "http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm"
yum -y install libcurl libcurl-devel rrdtool rrdtool-devel perl-rrdtool
        rrdtool-prel libgcrypt-devel gcc make gcc-c++ liboping liboping-devel perl-
        CPAN net-snmp net-snmp-devel
```

源码安装 collectd 如下：

```
wget http://collectd.org/files/collectd-5.4.1.tar.gz
tar zxvf collectd-5.4.1.tar.gz
cd collectd-5.4.1
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var --libdir=/usr/lib
        --mandir=/usr/share/man --enable-all-plugins
make && make install
```

安装启动脚本如下：

```
cp contrib/redhat/init.d-collectd /etc/init.d/collectd
chmod +x /etc/init.d/collectd
```

启动 collectd 如下：

```
service collectd start
```

### 2. collectd 的配置

以下配置可以实现对服务器基本的 CPU、内存、网卡流量、磁盘 I/O 以及磁盘空间占用情况的监控：

```
Hostname "host.example.com"
LoadPlugin interface
LoadPlugin cpu
```

```
LoadPlugin network
LoadPlugin df
LoadPlugin disk
<Plugin interface>
  Interface "eth0"
  IgnoreSelected false
</Plugin>
<Plugin network>
<Server "10.0.0.1"25826"> ## Logstash 的 IP 地址和 collectd 的数据接收端口号
</Server>
</Plugin>
```

### 3. Logstash 的配置

以下配置实现通过 Logstash 监听 25826 端口，接收从 collectd 发送过来的各项检测数据。

简单配置示例如下：

```
input {
  collectd {
    port => 25826 ## 端口号与发送端对应
    type => collectd
  }
}
```

推荐配置示例如下：

```
udp {
  port => 25826
  buffer_size => 1452
  workers => 3 # Default is 2
  queue_size => 30000 # Default is 2000
  codec => collectd { }
  type =>"collectd"
}
```

下面是简单的一个输出结果：

```
{
  "_index": "logstash-2014.12.11",
  "_type": "collectd",
  "_id": "dS6vVz4aRtK5xS86kwjZnw",
  "_score": null,
  "_source": {
    "host": "host.example.com",
    "@timestamp": "2014-12-11T06:28:52.118Z",
    "plugin": "interface",
    "plugin_instance": "eth0",
    "collectd_type": "if_packets",
```

```
"@version": "1",
"type": "collectd",
"tags": [
  "_grokparsefailure"
],
"sort": [
  1418279332118
]
}
```

---

## 参考资料

- ❑ collectd 支持收集的数据类型：<http://git.verplant.org/?p=collectd.git;a=blob;hb=master;f=README>
  - ❑ collectd 收集各数据类型配置参考资料：<http://collectd.org/documentation/manpages/collectd.conf.5.shtml>
  - ❑ collectd 简单配置文件示例：<https://gist.github.com/untergeek/ab85cb86a9bf39f1fc6d>
- 

## 2.2 编解码配置

Codec 是 Logstash 从 1.3.0 版开始新引入的概念（Codec 来自 Coder/decoder 两个单词的首字母缩写）。

在此之前，Logstash 只支持纯文本形式输入，然后以过滤器处理它。但现在，我们可以在输入期处理不同类型的数据，这全是因为有了 Codec 设置。

所以，这里需要纠正之前的一个概念。Logstash 不只是一个 input | filter | output 的数据流，而是一个 input | decode | filter | encode | output 的数据流！Codec 就是用来 decode、encode 事件的。

Codec 的引入，使得 Logstash 可以更好、更方便地与其他有自定义数据格式的运维产品共存，比如 graphite、fluent、netflow、collectd，以及使用 msgpack、json、edn 等通用数据格式的其他产品等。

事实上，我们在第一个“Hello World”用例中就已经用过 Codec 了——rubydebug 就是一种 Codec！虽然它一般只会用在 stdout 插件中，作为配置测试或者调试的工具。



这个五段式的流程说明源自 Perl 版的 Logstash（后来改名叫 Message::Passing 模块）的设计，本书稍后 5.8 节会对该模块稍作介绍。