Broadview®
www.broadview.com.cn

etcd
技术内幕

百里燊 / 编著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

etcd
技术内幕

百里燊 / 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

etcd 是一个可靠的分布式 KV 存储产品，由 CoreOS 公司开发，其底层使用 Raft 算法保证一致性，主要用于共享配置和服务发现。

本书主要从源码角度深入剖析 etcd，首先介绍 etcd 的背景知识，如何搭建源码环境及相关的命令。接着从基本的 Raft 协议开始介绍，帮助读者了解 Raft 协议的背景、如何处理各种异常场景及相关扩展。然后分析 etcd-raft 模块对 Raft 协议的实现，同时介绍 etcd 自带的 raftexample 示例，帮助读者了解 etcd-raft 模块的基本使用方法。本书还介绍 HTTP 编程基础和 etcd-rafthttp 模块的工作原理及具体实现，以及 etcd 中如何处理 WAL 日志文件及快照数据文件，并且详细分析 etcd 的底层存储，对 v2 版本和 v3 版本的存储都做了详细的介绍。最后重点介绍 etcd 服务端和客户端的相关内容，etcd 服务端会组装并协调前面介绍的各个组件，并且在它们的基础上扩展出了更多的功能，此外还详细分析 v2 和 v3 两个版本客户端的具体实现。

本书适合 Go 语言开发者，以及对 etcd 技术感兴趣的读者阅读。

图书在版编目（CIP）数据

etcd 技术内幕 / 百里燊编著. —北京：电子工业出版社，2018.7
ISBN 978-7-121-34386-5

Ⅰ. ①e… Ⅱ. ①百… Ⅲ. ①分布式存贮器 Ⅳ. ① TP333.2

中国版本图书馆 CIP 数据核字（2018）第 122913 号

责任编辑：陈晓猛
印　　刷：三河市华成印务有限公司
装　　订：三河市华成印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　　印张：25.75　　字数：490 千字
版　　次：2018 年 7 月第 1 版
印　　次：2018 年 7 月第 1 次印刷
定　　价：89.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前　言

etcd 是一个可靠的分布式 KV 存储，其底层使用 Raft 算法保证一致性，主要用于共享配置和服务发现。etcd 是 CoreOS 公司发起的一个开源项目，其源代码地址为 https://github.com/coreos/etcd。

目前提供配置共享和服务发现功能的组件还是比较多的，其中应用最广泛、大家最熟悉的应该就是 ZooKeeper 了，很多开源项目都在不同程度上依赖了 ZooKeeper，例如，Dubbo、Kafka。在 Golang 社区中，etcd 是唯一一个可以媲美 ZooKeeper 的组件，在有些方面，etcd 甚至超越了 ZooKeeper，给开发者眼前一亮的感觉。

etcd 作为一个优秀的分布式 KV 存储产品，其底层的 etcd-raft 模块实现了 Raft 协议，可以帮助开发者快速实现最终一致性功能。etcd 以其高性能、易维护、Raft 实现等优点，受到越来越多的开发人员的青睐，在 Golang 社区中声名大噪。

etcd 的代码中有很多亮点，为了提高整体性能，其实现中使用了大量的 goroutine 和 Channel。etcd 3 中开始使用 BoltDB 作为底层的持久存储，使用 BTree 索引加速查询，还提供了可靠的 Watcher 功能，同时提供了基于 GRPC 的新版本客户端。当然，etcd 3 依然兼容 etcd 2 的内存存储和 HTTP API。

etcd 3 中有很多令人称赞的功能和优秀的设计，但至今还没有深入剖析 etcd 3 的内部设计和实现细节的图书，本书以 etcd 3 为基础，详细分析了 etcd 的架构设计和实现细节，其中穿插介绍了 etcd 源码中涉及的基础知识，以及笔者在实践中的思考。

如何阅读本书

由于篇幅限制，本书并没有详细介绍 Go 语言的基础知识，但为了便于理解读者理解 etcd 的设计思想和实现细节，笔者介绍了一些必需且重要的基础内容，例如，Go 语言提供的 HTTP 功能。

本书共 8 章，主要从源码角度深入剖析 etcd 的原理和实现。建议读者先阅读前两章，了解

Raft 协议之后，再开始学习 etcd 的工作原理和代码实现。

第 1 章简要介绍 etcd 的背景知识及其基本的数据模型，然后介绍如何搭建源码环境及相关的命令。

第 2 章从基本的 Raft 协议开始介绍，帮助读者了解 Raft 协议的背景知识、大致工作原理、如何处理各种异常场景，以及几个比较重要的扩展点。

第 3 章着重分析 etcd 中的 raft 模块对 Raft 协议的实现，etcd-raft 模块是 etcd 的核心模块之一，GitHub 上很多其他项目直接使用 etcd-raft 模块作为其 Raft 协议的底层实现。同时，本章也介绍了 etcd 自带的 raftexample 示例，帮助读者了解 etcd-raft 模块的基本使用方法，方便读者在实践中直接使用 etcd-raft 模块。

第 4 章简单介绍 HTTP 编程基础，以及 etcd-rafthttp 模块的工作原理和具体实现，其中涉及 RoundTripper 的基本原理、rafthttp.Transporter 的实现及多种 Handler 的实现。

第 5 章介绍 etcd 中如何处理 WAL 日志文件和快照数据文件，其中分析了 WAL 结构体和 Snapshotter 的具体实现。

第 6 章详细分析 etcd 的底层存储，对 etcd v2 和 etcd v3 两个版本的存储都做了详细的介绍，不仅分析了底层的 KV 存储，还分析了 watcher 机制和 Lessor 的实现原理。

第 7 章重点介绍 etcd 服务端的内容，etcd 服务端会组装并协调前面介绍的各个组件，并且在它们的基础上扩展出了更多的功能。

第 8 章主要介绍 etcd 客户端的相关内容，详细分析了 v2 和 v3 两个版本客户端的具体实现。

如果读者在阅读本书的过程中，发现任何不妥之处，请将您宝贵的意见和建议发送到邮箱 shen_baili @163.com，也欢迎读者朋友通过此邮箱与我进行交流。

致谢

感谢我的母亲，谢谢您的付出和牺牲！

感谢白酱陪我看过一盏盏亮起的路灯，感谢三十在技术上提供的帮助。

感谢电子工业出版社博文视点的陈晓猛老师，是您的辛勤工作让本书的出版成为可能。同时还要感谢许多我不知道名字的幕后工作人员为本书付出的努力。

百里燊

读者服务

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **提交勘误：**您对书中内容的修改意见可在提交勘误处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动：**在页面下方读者评论处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/34386

目　录

第 1 章 etcd 入门

1.1 etcd简介

etcd 是一个可靠的分布式 KV 存储，其底层使用 Raft 算法保证一致性，主要用于共享配置和服务发现。etcd 是 CoreOS 公司发起的一个开源项目，授权协议为 Apache，其源代码地址为 https://github.com/coreos/etcd。

目前提供配置共享和服务发现功能的组件还是比较多的，其中应用最为广泛、大家最为熟悉的应该就是 ZooKeeper 了，很多开源项目也都在不同程度上依赖了 ZooKeeper，例如，Dubbo、Kafka 等。

这里简单介绍一下“服务发现”和“共享配置”两个概念。随着一个系统的不断迭代，功能模块会不断增加，对整个系统进行服务的拆分是必然的，这就会出现多个服务之间的相互调用，而在出现服务发现组件之前，一般是通过读取配置文件中预先设置的 IP 来获取服务的地址，然后进行调用。这会导致很多问题，例如，某些服务已经不可用时，调用方不能及时感知，负载均衡比较复杂，等等。使用服务发现组件之后，我们可以将服务提供方的信息注册到服务发现组件，例如，注册到 ZooKeeper 中，然后定期发送心跳等信息，让服务发现组件知晓服务提供方是可用的。当调用方进行服务调用时，会先请求服务发现组件，由服务发现组件来保证返回可用的服务地址及负载均衡等功能。

在一个系统的不同模块中有很多配置信息，例如，数据库地址、连接配置信息等都差不多，如果使用静态配置文件的方式实现，则需要将相同的信息写多份，每次更新配置时也需要更新多次。如果将这些配置信息注册到共享配置组件中，则系统的不同模块在启动时可从共享配置

组件中获取配置，同时会监听配置信息的更改，当配置信息发生变更时，可以自动将配置值替换成新值。

在 Golang 社区中，etcd 则是唯一一个可以媲美 ZooKeeper 的组件，在有些方面，etcd 甚至超越了 ZooKeeper，给开发者眼前一亮的感觉。下面简单列举一下 etcd 相较于 ZooKeeper 的优势。

- **一致性协议：**一致性协议是配置共享和服务发现组件的核心，etcd 底层采用了 Raft 协议，而 ZooKeeper 使用 ZAB 协议，ZAB 协议是一种类 Paxos 的一致性协议。目前公认的是 Raft 比 Paxos 协议易于理解，工程化也较为容易。
- **API 接口：**etcd v2 版本中提供了 HTTP+JSON 的调用方式，在 etcd v3 版本的客户端中则使用 GRPC 与服务端进行交互，而 GRPC 本身就是跨平台的。
- **性能：**在官方提供的基准测试数据中，etcd 集群可以支持每秒 10000+次的写入，性能相当可观，优于 ZooKeeper。
- **安全性：**etcd 支持 TLS 访问，而 ZooKeeper 在权限控制这方面做得略显粗糙。

etcd 也有很多成功的案例，最为大家熟知的就是 Kubernetes，其底层就依赖于 etcd 实现集群状态和配置的管理。除了 Kubernetes，Cloud Foundry 等 500+个 GitHub 项目都使用了 etcd。etcd 在 GitHub 上也有 17000+的 Star 和近 4000 的 Fork。

1.2 数据模型

etcd 支持可靠的键值对存储并且提供了可靠的 Watcher 机制，其中的键值对存储支持多版本，并且具备能够“Watch”历史事件的功能。这里简单介绍多版本存储的含义，假设键 K1 对应的值为 V1，当我们将 K1 对应的值修改成 V2 时，etcd 并不会直接将 V1 修改成 V2，而是同时记录 V1 和 V2 两个值，并通过不同的版本号进行区分。另外，Watch 历史事件的含义是，我们可以向一个 Key 添加 Watcher，同时可以指定一个历史版本，从该版本开始的所有事件都会触发该 Watcher。

随着应用不断运行，键值对不断修改，每个 Key 都在 etcd 中保存了多个版本，数据量也会越来越大。为了缓解压力，etcd 会定期进行压缩，清理过旧的数据。

在很多现代数据库系统中，都用了 B 树索引加速查询，etcd 也是如此，其存储中会维护一个字段序的 B 树索引。在 B 树索引的每个索引项中，都存储了一个 Key 值，这样可以快速定位指定的 Key 或是进行范围查询。而每个 Key 值对应了多个版本号，etcd 中维护了一个全局自增的版本号，为每次事务分配一个全局唯一的版本号（main revision），事务中的每个操作也有唯一的编号（sub revision），通过这两部分可以确定一个唯一的 Value 值。

每个Key会对应多个generation，当Key首次创建时，会同时创建一个与之关联的generation实例，当该Key被修改时，会将对应的版本记录到generation中，当Key被删除时，会向generation中添加tombstone，并创建新的generation，会向新generation中写入后续的版本信息。

在查询时，先在内存索引中通过用户指定的Key值，查找到该Key值对应的全部版本号，然后根据用户指定的版本号，从底层存储中查找到具体的Value值。当然，如果指定的版本号已经被etcd压缩删除，则无法再查询到该版本的Value值。

在etcd v3版本中，底层存储使用的是BoltDB，其中的Key是版本信息（main revision+sub revision）。这样，在查询时先通过上述B树索引查找到对应的版本信息，然后在BoltDB中通过版本信息查找相应的Value值。

1.3 环境搭建

了解了etcd中基本的概念和数据模型之后，我们开始搭建阅读etcd源码的环境。这里推荐使用Goland，如果读者熟悉其他IDE，也可以按照下面的步骤进行搭建。

1.3.1 环境变量

首先需要从Golang官方网站上下载最新的Golang包，笔者在开始写作时的最新版本是go1.10.darwin-amd64.tar.gz，下载完成后进行解压。接下来我们需要配置相应的环境变量，打开~/.bash_profile文件，添加如下内容：

```
export GOROOT=/Users/xxx/Documents/go
export GOBIN=$GOROOT/bin
export GOARCH=amd64
export GOOS=darwin
export GOPATH=/Users/xxx/Documents/GoProject
export PATH=$PATH:$GOBIN:$GOPATH/bin
```

如果读者使用了其他终端，则需要在相应的终端配置文件中添加上述配置。最后执行一下“source .bash_profile”命令编译配置文件。我们可以通过“go env”命令验证环境变量配置是否成功。

1.3.2 代码结构

我们可以直接使用“go get”命令下载etcd的源码代码，具体命令如下：

```
go get github.com/coreos/etcd/cmd/etcd
```

下载过程可能比较长，请读者耐心等待，待下载完成之后，即可以将代码直接导入 Goland 中了。这里简单介绍其中各个模块的主要功能，如下所示。

- raft：Raft 协议的核心实现，其中只实现了基本的 Raft 协议，并未提供实现网络层相关的内容。
- raft-http：Raft 协议中各个节点发送消息时使用的网络层实现，该模块与 raft 模块紧密相关。
- wal 和 snap：WAL 日志和快照存储相关的实现。
- store：etcd 中的 v2 版本存储实现，v2 版本的存储是完全的内存实现。
- mvcc：etcd 中的 v3 版本存储实现，v3 版本的存储底层使用 BoltDB 实现持久化存储。
- lease：租约相关的实现。
- auth 和 alarm：权限和报警相关的实现。
- etcdserver：etcd 服务端实现，它会基于上述模块提供的功能，对外提供一个 etcd 节点的完整功能。
- client：v2 版本客户端的具体实现，v2 版本的客户端是通过 HTTP+JSON 的方式与服务端进行交互的。
- clientv3：v3 版本客户端的具体实现，v3 版本的客户端是通过 GRPC 的方式与服务端进行交互的。

1.3.3 运行

etcd 提供了单机模式和集群模式两种模式，单机模式比较简单，直接编译之前下载的源代码得到/bin/etcd 二进制文件并运行即可。默认配置运行时，etcd 服务端会监听本地的 2379 和 2380 两个端口，其中 2379 端口用于与客户端的交互，2380 端口则是用于 etcd 节点内部交互（主要是发送 Raft 协议相关的消息等）。当 etcd 服务端启动时，我们可以使用 etcdctl 工具进行测试，关于 etcdctl 命令的使用，这里不再展开介绍，请读者参考官方文档进行学习。

本小节重点介绍集群模式的搭建。为了方便测试和部署，在本小节的示例中将会在同一台机器上启动三个 etcd 节点，但每个节点监听不同的端口，形成伪分布式的集群。

首先介绍静态配置的启动方式，这种方式会预先将集群中各个节点的配置信息分配好，然后将集群中的节点逐个启动，这些节点将根据配置信息组成集群。在该示例中，三个节点启动

的代码分别如下所示。

```
./bin/etcd --name infra0 --initial-advertise-peer-urls http://127.0.0.1:2380 \
  --listen-peer-urls http://127.0.0.1:2380 \
  --listen-client-urls http://127.0.0.1:2379 \
  --advertise-client-urls http://127.0.0.1:2379 \
  --initial-cluster-token etcd-cluster-1 \
  --initial-cluster infra0=http://127.0.0.1:2380, \
    infra1=http://127.0.0.1:2382,infra2=http://127.0.0.1:2384 \
  --initial-cluster-state new

./bin/etcd --name infra1 --initial-advertise-peer-urls http://127.0.0.1:2382 \
  --listen-peer-urls http://127.0.0.1:2382 \
  --listen-client-urls http://127.0.0.1:2381 \
  --advertise-client-urls http://127.0.0.1:2381 \
  --initial-cluster-token etcd-cluster-1 \
  --initial-cluster infra0=http://127.0.0.1:2380,\
    infra1=http://127.0.0.1:2382,infra2=http://127.0.0.1:2384 \
  --initial-cluster-state new

./bin/etcd --name infra2 --initial-advertise-peer-urls http://127.0.0.1:2384 \
  --listen-peer-urls http://127.0.0.1:2384 \
  --listen-client-urls http://127.0.0.1:2383 \
  --advertise-client-urls http://127.0.0.1:2383 \
  --initial-cluster-token etcd-cluster-1 \
  --initial-cluster infra0=http://127.0.0.1:2380,\
    infra1=http://127.0.0.1:2382,infra2=http://127.0.0.1:2384 \
  --initial-cluster-state new
```

这里简单介绍一下上述命令使用的参数。

- --name：etcd 集群中的节点名称，在同一个集群中必须是唯一的。
- --listen-peer-urls：用于集群内各个节点之间通信的 URL 地址，每个节点可以监听多个 URL 地址，集群内部将通过这些 URL 地址进行数据交互，例如，Leader 节点的选举、Message 消息传输或是快照传输等。
- --initial-advertise-peer-urls：建议用于集群内部节点之间交互的 URL 地址，节点间将以该值进行通信。

- --listen-client-urls：用于当前节点与客户端交互的 URL 地址，每个节点同样可以向客户端提供多个 URL 地址。
- --advertise-client-urls：建议客户端使用的 URL 地址，该值用于 etcd 代理或 etcd 成员与 etcd 节点通信。
- --initial-cluster-token etcd-cluster-1：集群的唯一标识。
- --initial-cluster：集群中所有的 initial-advertise-peer-urls 的合集。
- --initial-cluster-state new：新建集群的标识。

当集群中各个节点按照上述配置分别启动后，集群会通过 Leader 选举选出一个 Leader 节点，另外两个节点将成为 Follower 节点。我们可以使用 etcdctl 命令检测当前集群的状态，etcdctl 的相关使用方式请读者参考官方文档进行学习。

除了静态配置的启动方式，etcd 还提供了服务发现和 DNS 发现两种启动方式，这两种启动方式并不复杂，这里不再一一展开介绍，读者可以参考官方文档进行测试。

本章小结

本章首先对 etcd 进行了简单的介绍，以及介绍 etcd 相较于 ZooKeeper 的优点。之后，我们简单介绍了 etcd 的数据模型，其中提到 etcd 中在内存中维护了 B 树索引，并且为每个 Key 维护了多个版本，在 etcd v3 中使用 BoltDB 作为持久存储，而 etcd v2 则是全内存实现。然后，我们简单介绍了 etcd 源码环境的搭建及代码结构，最后介绍了 etcd 集群中最基本的静态配置启动方式。希望通过本章的介绍，读者可以大致了解 etcd 的数据结构及代码结构，也希望读者能够自己亲自完成 etcd 的源码环境搭建，并使用静态配置的方式启动本地的测试集群。

第2章

Raft 协议

在正式开始介绍 Raft 协议之间，我们有必要简单介绍一下其相关概念。在分布式系统中，一致性是比较常见的概念，所谓一致性指的是集群中的多个节点在状态上达成一致。在程序和操作系统不会崩溃、硬件不会损坏、服务器不会掉电、网络绝对可靠且没有延迟的理想情况下，我们可以将集群中的多个节点看作一个整体，此时要保证它们的一致性并不困难。

但是在现实的场景中，很难保证上述极端的条件全部满足，节点之间的一致性也就很难保证，这样就需要 Paxos、Raft 等一致性协议。一致性协议可以保证在集群中**大部分节点可用**的情况下，集群依然可以工作并给出一个正确的结果，从而保证依赖于该集群的其他服务不受影响。这里的“**大部分节点可用**”指的是集群中超过半数以上的节点可用，例如，集群中共有 5 个节点，此时其中有 2 个节点出现故障宕机，剩余的可用节点数为 3，此时，集群中大多数节点处于可用的状态，从外部来看集群依然是可用的。

常见的一致性算法有 Paxos、Raft 等，Paxos 协议是 Leslie Lamport 于 1990 年提出的一种基于消息传递的、具有高度容错特性的一致性算法，Paxos 算法解决的主要问题是分布式系统内如何就某个值达成一致。在相当长的一段时间内，Paxos 算法几乎成为一致性算法的代名词，但是 Paxos 有两个明显的缺点：第一个也是最明显的缺点就是 Paxos 算法难以理解，Paxos 算法的论文本身就比较晦涩难懂，要完全理解 Paxos 协议需要付出较大的努力，很多经验丰富的开发者在看完 Paxos 论文之后，无法将其有效地应用到具体工程实践中，这明显增加了工程化的门槛，也正因如此，才出现了几次用更简单的术语来解释 Paxos 的尝试。Paxos 算法的第二个缺点就是它没有提供构建现实系统的良好基础，也有很多工程化 Paxos 算法的尝试，但是它们对 Paxos 算法本身做了比较大的改动，彼此之间的实现差距都比较大，实现的功能和目的都有所不同，同时与 Paxos 算法的描述有很多出入。例如，著名 Chubby，它实现了一个类 Paxos 的算法，但其中很多细节并未被明确。本章并不打算详细介绍 Paxos 协议的相关内容，如果读