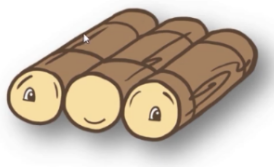
一、ETCD的一些基本理论

etcd与Raft的关系

- Raft是强一致的集群日志同步算法
- etcd是一个分布式KV存储
- etcd利用raft算法在集群中同步key-value

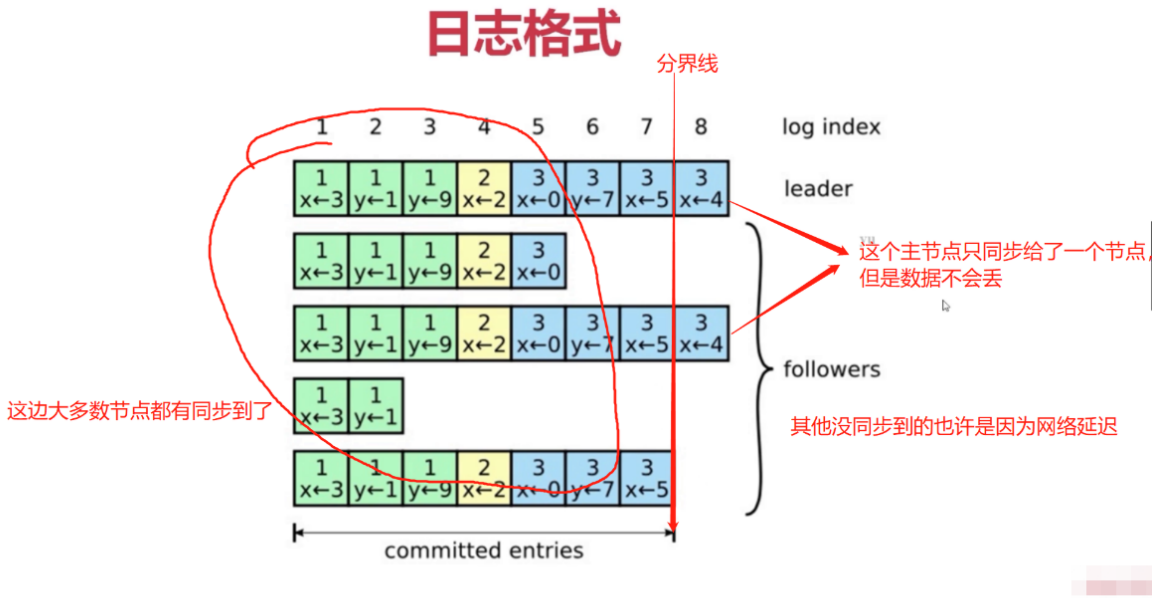


1.1、写入一个数据的阶段1

1.2、写入一个数据的阶段2

h5 1.3、日志同步原理

1.3、日志同步原理



1.4、Raft日志概念

1.4、Raft日志概念

Raft日志概念

- replication：日志在leader生成，向follower复制，达到各个节点的日志序列最终一致
- term：任期，重新选举产生的leader，其term单调递增
- log index：日志行在日志序列的下标

二、Golang代码操作etcd

2.1、etcd安装

```
# 官网：
https://github.com/etcd-io/etcd/tree/main/client/v3
https://pkg.go.dev/github.com/coreos/etcd/clientv3#pkg-index

# 安装依赖
go get go.etcd.io/etcd/client/v3

# 安装etcd
[root@node01 ~]# yum install -y etcd
# 设置开机自启动
systemctl enable etcd
# 启动etcd
systemctl start etcd
# 查看etcd运行状态
systemctl status etcd

# systemd配置
从systemctl status etcd命令的输出可以看到，etcd的 systemd配置文件位于/usr/lib/systemd/system/etcd.service，该配置文件的内容如下：

$ cat /usr/lib/systemd/system/etcd.service
[Unit]
Description=Etcd Server
After=network.target
After=network-online.target
Wants=network-online.target

[Service]
Type=notify
WorkingDirectory=/var/lib/etcd/
EnvironmentFile=-/etc/etcd/etcd.conf
User=etcd
# set GOMAXPROCS to number of processors
```

```
ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /usr/bin/etcd --
name=\"${ETCD_NAME}\" --data-dir=\"${ETCD_DATA_DIR}\" --listen-client-
urls=\"${ETCD_LISTEN_CLIENT_URLS}\""
Restart=on-failure
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target

# 从上面的配置中可以看到，etcd的配置文件位于/etc/etcd/etcd.conf，如果我们想要修改某些配置项，可以编辑该文件。

# 远程访问
etcd安装完成后，默认只能本地访问，如果需要开启远程访问，还需要修改/etc/etcd/etcd.conf中的配置。例如，本实例中我安装etcd的机器IP是10.103.18.41，我尝试通过自己的机器远程访问10.103.18.41上安装的etcd的2379端口，结果访问被拒绝：

# 修改/etc/etcd/etcd.conf配置：
ETCD_LISTEN_CLIENT_URLS="http://10.103.18.41:2379,http://localhost:2379"

# 然后重启
systemctl restart etcd
```

2.2、代码操作

- 连接etcd

```
package main

import (
    "fmt"
    "time"

    clientv3 "go.etcd.io/etcd/client/v3"
)

var (
    config clientv3.Config
    client *clientv3.Client
    err    error
)

func main() {

    // ETCD客户端连接信息
    config = clientv3.Config{
        Endpoints:   []string{"192.168.1.210:2379"}, // 节点信息
        DialTimeout: 5 * time.Second,                // 超时时间
    }

    // 建立连接
    if client, err = clientv3.New(config); err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(client)
}
```

- 操作etcd
- 相关理论

1. `Revision`
   : 作用域为集群，逻辑时间戳，全局单调递增，任何 key 修改都会使其自增
2. `CreateRevision`
   : 作用域为 key, 等于创建这个 key 时的 Revision, 直到删除前都保持不变
3. `ModRevision`
   : 作用域为 key, 等于修改这个 key 时的 Revision, 只要这个 key 更新都会改变
4. `Version`
   : 作用域为 key, 某一个 key 的修改次数(从创建到删除)，与以上三个 Revision 无关

关于 watch 哪个版本：

1. watch 某一个 key 时，想要从历史记录开始就用 CreateRevision，最新一条(这一条直接返回)开始就用 `ModRevision`
2. watch 某个前缀，就必须使用 `Revision`

- 增加一个key、查询一个key、删除一个key

```
package main

import (
    "context"
    "fmt"
    "time"

    clientv3 "go.etcd.io/etcd/client/v3"
)

var (
    config  clientv3.Config
    client  *clientv3.Client
    putResp *clientv3.PutResponse
    getResp *clientv3.GetResponse
    delResp *clientv3.DeleteResponse
    kv      clientv3.KV
    err     error
)

func main() {

    // ETCD客户端连接信息
    config = clientv3.Config{
        Endpoints:   []string{"192.168.1.210:2379"}, // 节点信息
        DialTimeout: 5 * time.Second,                // 超时时间
    }

    // 建立连接
    if client, err = clientv3.New(config); err != nil {
        fmt.Printf("connect to etcd failed, err:%v\n", err)
        return
    }
    fmt.Println("connect to etcd success")


    // 用于读写ETCD的键值对
```

```
    kv = clientv3.NewKV(client)
    // 操作etcd,context.TODO() 这是一个上下文,如果这上下文不知道选那种,就选这个万精
油;clientv3.WithPrevKV()加这参数获取前一个kv的值
    if putResp, err = kv.Put(context.TODO(), "/cron/jobs/job1", "1008611",
clientv3.WithPrevKV()); err != nil {
        fmt.Println(err)
        return
    }
    // Revision: 作用域为集群，逻辑时间戳，全局单调递增，任何 key 修改都会使其自增
    fmt.Println("Revision is:", putResp.Header.Revision)
    if putResp.PrevKv != nil {
        // 查看被更新的K V
        fmt.Println("更新的Key是：", string(putResp.PrevKv.Key))
        fmt.Println("被更新的Value是：", string(putResp.PrevKv.Value))
    }

    // 读取ETCD数据
    if getResp, err = kv.Get(context.TODO(), "/cron/jobs/job1"); err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(getResp.Kvs)

    // 读取ETCD数据，获取前缀相同的WithPrefix()
    if getResp, err = kv.Get(context.TODO(), "/cron/jobs/",
clientv3.WithPrefix()); err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(getResp.Kvs)

    // 删除ETCD数据;WithPrevKV--->赋值数据给delResp.PrevKvs,方便后续判断
    // 删除多个key：kv.Delete(context.TODO(), "/cron/jobs/",
clientv3.WithPrefix())
    if delResp, err = kv.Delete(context.TODO(), "/cron/jobs/job1",
clientv3.WithPrevKV()); err != nil {
        fmt.Println(err)
        return
    }
    // 打印被删除之前的kv
    if len(delResp.PrevKvs) != 0 {
        for _, kvpx := range delResp.PrevKvs {
            fmt.Println("被删除的数据是: ", string(kvpx.Key), string(kvpx.Value))
        }
    }
}
```

- 租约、自动租约、lease

```
package main

import (
    "context"
    "fmt"
    "time"

    clientv3 "go.etcd.io/etcd/client/v3"
```

```
)

var (
    config         clientv3.Config
    leaseID        clientv3.LeaseID
    client         *clientv3.Client
    LeaseGrantResp *clientv3.LeaseGrantResponse
    putResp        *clientv3.PutResponse
    getResp        *clientv3.GetResponse
    keepResp       *clientv3.LeaseKeepAliveResponse
    keepRespChan   <-chan *clientv3.LeaseKeepAliveResponse // 只读管道
    kv             clientv3.KV
    err            error
)

func main() {
    // 连接客户端配置文件
    config = clientv3.Config{
        Endpoints:   []string{"192.168.1.210:2379"},
        DialTimeout: 5 * time.Second,
    }

    // 建立连接
    if client, err = clientv3.New(config); err != nil {
        fmt.Printf("conect to etcd faild, err:%v\n", err)
        return
    } else {
        fmt.Println("connect to etcd success")
    }

    // 获取kv API子集
    kv = clientv3.NewKV(client)

    // 申请一个租约 lease
    lease := clientv3.Lease(client)

    // 申请一个10s的租约
    if LeaseGrantResp, err = lease.Grant(context.TODO(), 10); err != nil {
        fmt.Println("租约申请失败", err)
        return
    }

    // 租约ID
    leaseID = LeaseGrantResp.ID

    // 自动续租
    if keepRespChan, err = lease.KeepAlive(context.TODO(), leaseID); err != nil
{
        fmt.Println("自动续租失败", err)
        return
    }

    /*
      10s后自动过期
      ctx, canceFunc := context.WithCancel(context.TODO())
      // 自动续租
      if keepRespChan, err = lease.KeepAlive(ctx, leaseID); err != nil {
          fmt.Println("自动续租失败", err)
```

```
            return
        }
        canceFunc()

    */

    // 处理续约应答的协程  消费keepRespChan
    go func() {
        for {
            select {
            case keepResp = <-keepRespChan:
                if keepRespChan == nil {
                    fmt.Println("租约已经失效了")
                    goto END
                } else {
                    // KeepAlive每秒会续租一次,所以就会收到一次应答
                    fmt.Println("收到应答,租约ID是:", keepResp.ID)
                }
            }
        }
    END:
    }()

    // put一个kv,让他与租约关联起来,从而实现10s后自动过期,key就会被删除; 关联用的是
clientv3.WithLease(leaseID)
    if putResp, err = kv.Put(context.TODO(), "/cron/lock/job3", "3",
clientv3.WithLease(leaseID)); err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println("写入成功:", putResp.Header.Revision)

    // 判断key是否过期
    for {
        if getResp, err = kv.Get(context.TODO(), "/cron/lock/job3"); err != nil
{
            fmt.Println(err)
            return
        }
        // 如果等于0,说明过期了
        if getResp.Count == 0 {
            fmt.Println("kv过期了")
            break
        } else {
            fmt.Println("没过期", getResp.Kvs)
        }
        time.Sleep(2 * time.Second)
    }
}
```

- **watch操作**

```
package main

import (
    "context"
    "fmt"
```

```
    "time"

    "go.etcd.io/etcd/api/v3/mvccpb"
    clientv3 "go.etcd.io/etcd/client/v3"
)

var (
    config             clientv3.Config
    leaseID            clientv3.LeaseID
    watcher            clientv3.Watcher
    kv                 clientv3.KV
    watchResp          clientv3.WatchResponse
    event              *clientv3.Event
    client             *clientv3.Client
    LeaseGrantResp     *clientv3.LeaseGrantResponse
    putResp            *clientv3.PutResponse
    getResp            *clientv3.GetResponse
    keepResp           *clientv3.LeaseKeepAliveResponse
    keepRespChan       <-chan *clientv3.LeaseKeepAliveResponse // 只读管道
    watchRespChan      <-chan clientv3.WatchResponse
    watchStartRevision int64
    err                error
)

func main() {
    // 连接客户端配置文件
    config = clientv3.Config{
        Endpoints:   []string{"192.168.1.210:2379"},
        DialTimeout: 5 * time.Second,
    }

    // 建立连接
    if client, err = clientv3.New(config); err != nil {
        fmt.Printf("conect to etcd faild, err:%v\n", err)
        return
    } else {
        fmt.Println("connect to etcd success")
    }

    // 获取kv API子集
    kv = clientv3.NewKV(client)

    // 模拟etcd中数据的变化
    go func() {
        for {
            kv.Put(context.TODO(), "/cron/jobs/job18", "I am 18")

            kv.Delete(context.TODO(), "/cron/jobs/job18")

            time.Sleep(1 * time.Second)
        }
    }()

    if getResp, err = kv.Get(context.TODO(), "/cron/jobs/job18"); err != nil {
        fmt.Printf("getResp err:%v\n", err)
        return
    }
```

```
    if len(getResp.Kvs) != 0 {
        fmt.Println(getResp.Kvs[0].Value)
    }

    // 当前etcd集群事务ID,单调递增的
    watchStartRevision = getResp.Header.Revision + 1

    // 创建个 watcher
    watcher = clientv3.NewWatcher(client)

    // 启动监听
    fmt.Println("从该Revision版本向后监听:", watchStartRevision)

    // 一直监听
    // watchRespChan = watcher.Watch(context.TODO(), "/cron/jobs/job18",
clientv3.WithRev(watchStartRevision))

    // 自动关闭监听,调用canceFunc()函数即可取消
    xtc, canceFunc := context.WithCancel(context.TODO())

    // xx秒后干什么事--->time.AfterFunc,执行匿名函数
    time.AfterFunc(5*time.Second, func() {
        canceFunc()
    })

    //启动监听
    watchRespChan = watcher.Watch(xtc, "/cron/jobs/job18",
clientv3.WithRev(watchStartRevision))

    // 处理kv变化事件
    for watchResp = range watchRespChan {
        for _, event = range watchResp.Events {
            switch event.Type {
            case mvccpb.PUT:
                fmt.Println("修改为:", string(event.Kv.Value), "CreateRevision
is:", event.Kv.CreateRevision, "ModRevision is:", event.Kv.ModRevision)
            case mvccpb.DELETE:
                fmt.Println("删除了:", "Revision is", event.Kv.ModRevision)
            }
        }
    }
}
```

```
// xx秒后干什么事--->time.AfterFunc,执行匿名函数
    time.AfterFunc(5*time.Second, func() {
        fmt.Println("1")
    })
```

- **OP的方式PUT、GET数据**

```
package main

import (
    "context"
    "fmt"
    "time"
```

```
    clientv3 "go.etcd.io/etcd/client/v3"
)

var (
    config clientv3.Config
    kv     clientv3.KV
    putOp  clientv3.Op
    getOp  clientv3.Op
    opResp clientv3.OpResponse
    client *clientv3.Client
    err    error
)

func main() {
    // 连接客户端配置文件
    config = clientv3.Config{
        Endpoints:   []string{"192.168.1.210:2379"},
        DialTimeout: 5 * time.Second,
    }

    // 建立连接
    if client, err = clientv3.New(config); err != nil {
        fmt.Printf("conect to etcd faild, err:%v\n", err)
        return
    } else {
        fmt.Println("connect to etcd success")
    }

    // 获取kv API子集
    kv = clientv3.NewKV(client)

    // 创建OP---> k v 对象
    putOp = clientv3.OpPut("/cron/jobs/job19", "19")

    // 执行OP
    if opResp, err = kv.Do(context.TODO(), putOp); err != nil {
        fmt.Printf("执行OP faild, err:%v\n", err)
        return
    }

    fmt.Println("Revision is:", opResp.Put().Header.Revision)

    // 创建OP---> k v 对象
    getOp = clientv3.OpGet("/cron/jobs/job19")

    // 执行OP
    if opResp, err = kv.Do(context.TODO(), getOp); err != nil {
        fmt.Printf("执行OP faild, err:%v\n", err)
        return
    }

    // 打印数据
    fmt.Println("数据ModRevision", opResp.Get().Kvs[0].ModRevision)
    fmt.Println("数据Value", string(opResp.Get().Kvs[0].Value))
}
```

- **分布式锁**
  - **同时运行两次,验证代码**

```
package main

import (
    "context"
    "fmt"
    "time"

    clientv3 "go.etcd.io/etcd/client/v3"
)

var (
    config         clientv3.Config
    leaseID        clientv3.LeaseID
    ctx            context.Context
    canceFunc      context.CancelFunc
    txn            clientv3.Txn
    client         *clientv3.Client
    LeaseGrantResp *clientv3.LeaseGrantResponse
    keepResp       *clientv3.LeaseKeepAliveResponse
    txnResp        *clientv3.TxnResponse
    keepRespChan   <-chan *clientv3.LeaseKeepAliveResponse // 只读管道
    kv             clientv3.KV
    err            error
)

/*
    lease实现锁自动过期
    op操着
    txn事务：if else then
*/

func main() {
    // 连接客户端配置文件
    config = clientv3.Config{
        Endpoints:   []string{"192.168.1.210:2379"},
        DialTimeout: 5 * time.Second,
    }

    // 建立连接
    if client, err = clientv3.New(config); err != nil {
        fmt.Printf("conect to etcd faild, err:%v\n", err)
        return
    } else {
        fmt.Println("connect to etcd success")
    }

    // 1、上锁(创建租约、自动续租、拿着租约去抢占一个key)
    // 申请一个租约 lease
    lease := clientv3.Lease(client)

    // 申请一个5s的租约
    if LeaseGrantResp, err = lease.Grant(context.TODO(), 5); err != nil {
        fmt.Println("租约申请失败", err)
```

```
        return
    }

    // 租约ID
    leaseID = LeaseGrantResp.ID

    // 准备一个用于取消的自动续租的context; cancanceFunc 取消续租调用这个函数即可
    ctx, canceFunc = context.WithCancel(context.TODO())

    // 确保函数退出后，自动续约会停止
    defer canceFunc()
    defer lease.Revoke(context.TODO(), leaseID)

    // 自动续租
    if keepRespChan, err = lease.KeepAlive(ctx, leaseID); err != nil {
        fmt.Println("自动续租失败", err)
        return
    }

    // 判断续约应答的协程
    go func() {
        for {
            select {
            case keepResp = <-keepRespChan:
                if keepRespChan == nil {
                    fmt.Println("租约已经失效了")
                    goto END
                } else {
                    // KeepAlive每秒会续租一次,所以就会收到一次应答
                    fmt.Println("收到应答,租约ID是:", keepResp.ID)
                }
            }
        }
    END:
    }()

    // ***拿着租约去抢占一个key***
    // 获取kv API子集
    kv = clientv3.NewKV(client)

    // 创建事务
    txn = kv.Txn(context.TODO())

    // 定义事务
    // 如果key不存在;关联用的是clientv3.WithLease(leaseID)
    txn.If(clientv3.Compare(clientv3.CreateRevision("/cron/lock/job19"), "=",
0)).
        // 不存在就put一个key
        Then(clientv3.OpPut("/cron/lock/job19", "xxx",
clientv3.WithLease(leaseID))).
        // 否则枪锁失败
        Else(clientv3.OpGet("/cron/lock/job19"))

    // 提交事务
    if txnResp, err = txn.Commit(); err != nil {
        fmt.Println("txn err", err)
        return
    }
```

```
    // 判断释放抢到锁
    if !txnResp.Succeeded {
        fmt.Println("锁被占用",
string(txnResp.Responses[0].GetResponseRange().Kvs[0].Value))
        return
    }

    // 2、处理业务
    fmt.Println("处理任务")
    time.Sleep(50 * time.Second)

    // 3、释放锁(取消自动续租、释放租约)
    /*
        defer canceFunc()
        defer lease.Revoke(context.TODO(), leaseID)
        上面这个释放了租约,关联的kv会被删除,从而达到释放锁
    */
}
```