

# Etcd 分布式锁的演进

1. Etcd的lock实现
2. Jetcd的一些问题
3. 使用Txn命令实现tryLock
4. Gondor prod环境出现的问题
5. 一些优化

# 1. Etcd的lock实现

```
// master
key1 = "gondor.dev.dealer/1923199439283"
// slave
key2 = "gondor.dev.dealer/1823199439212"
// slave
key3 = "gondor.dev.dealer/2023199432122"
```

```
+-----+
|gondor.dev.dealer/1923199439283 | -> revision 6162122
+-----+
```

```
+-----+
|gondor.dev.dealer/1823199439212 | -> revision 7162122
+-----+
```

```
+-----+
|gondor.dev.dealer/2023199432122 | -> revision 8162122
+-----+
```

## 优点

1. 可重入
2. 公平锁

## 缺点

1. 范围查询以及多条语句查询， 速度较慢
2. 阻塞的， 没有tryLock机制

## 2. Jetcd的一些问题

```
CompletableFuture<LockResponse> lock(String lock, long lease)
```

```
CompletableFuture<LockResponse> r = lock("lock", 9123199213L)  
// slave会timeout, 但etcd server仍在执行等待锁的动作  
LockResponse res = r.get(6, SECONDS);
```

<https://github.com/nextopcn/gadget-etcd>

### 3. 使用Txn命令实现tryLock

#### Txn命令介绍

1. txn命令等同于 `if(condition) then success else failure`
2. txn命令类似CAS原子操作

## 实现tryLock

```
condition : "gondor.dev.dealer".revision == 0  
success  : put("gondor.dev.dealer", "dealer:21", lease)  
failure  : do nothing
```

## 4. Gondor prod环境出现的问题

1. etcd: request "xxxx" took too long (8.4s) to execute
2. wal: sync duration of 8.8s, expected less than 1s
3. grpc线程很多



## 5. 一些优化

1. 避免Txn命令包含太多指令
2. 避免创建并销毁太多lease
3. 避免Grpc创建太多线程
4. 将wal日志迁移到内存

## References

1. [mutex.go#L64](#)
2. [Util.java#L141](#)
3. [gadget-etcd](#)