链滴

hyperledger fabric v0.6 pbft 源码分析（一）requeststore.go

作者：deshanxiao

原文链接：https://ld246.com/article/1511232762462

来源网站：链滴

阅读fabric源码的共识机制部分，感觉源码难度还是有的，所以先从最简单的requeststore开始吧。

在阅读了部分超级账本的源码后，有一个经验就是，在阅读源码特别是大项目的源码时，可能会感到所适从，其实这也是很正常的，我的经验是可以先从一条线开始理清代码的执行流。比如像 hyperledger 这样的平台，可以从链码的执行这条线来看源码，跟着调试一步步走，相信会简单不少。

但是对于那些不是很好调试的代码来说，还有一个简单的方法，就是看代码的单元测试的程序，体会是怎么使用的，这其实也是一个比较好的方法，下面分析pbft的实现源码，就是使用这种方法来分析。

pbft实现起来不容易，这里从它最简单的部分入手，话不多说，看代码吧：

```
// consensus/pbft/requeststore_test.go
func TestOrderedRequests(t *testing.T) {
    or := &orderedRequests{}
    or.empty()

    r1 := createPbftReq(2, 1)
    r2 := createPbftReq(2, 2)
    r3 := createPbftReq(19, 1)
    if or.has(or.wrapRequest(r1).key) {
        t.Errorf("should not have req")
    }
    or.add(r1)
    if !or.has(or.wrapRequest(r1).key) {
        t.Errorf("should have req")
    }
    if or.has(or.wrapRequest(r2).key) {
        t.Errorf("should not have req")
    }
    if or.remove(r2) {
        t.Errorf("should not have removed req")
    }
    if !or.remove(r1) {
        t.Errorf("should have removed req")
    }
    if or.remove(r1) {
        t.Errorf("should not have removed req")
    }
    if or.order.Len() != 0 || len(or.presence) != 0 {
        t.Errorf("should have 0 len")
    }
    or.adds([]*Request{r1, r2, r3})

    if or.order.Back().Value.(requestContainer).req != r3 {
        t.Errorf("incorrect order")
    }
}

func BenchmarkOrderedRequests(b *testing.B) {
    or := &orderedRequests{}
    or.empty()

    Nreq := 100000
```

```
    reqs := make(map[string]*Request)
    for i := 0; i < Nreq; i++ {
        rc := or.wrapRequest(createPbftReq(int64(i), 0))
        reqs[rc.key] = rc.req
    }
    b.ResetTimer()

    b.N = 100;
    fmt.Printf("N is %d\n", b.N)
    for i := 0; i < b.N; i++ {

        for _, r := range reqs {
            or.add(r)
        }

        for k := range reqs {
            _ = or.has(k)
        }

        for _, r := range reqs {
            or.remove(r)
        }
    }
}
```

从requeststore_test.go开始看，它测试了两个函数：

- TestOrderedRequests(t *testing.T)
- BenchmarkOrderedRequests(b *testing.B)

这里的第一个测试函数是普通的测试函数，第二个是benchmark测试函数（注意*testing.B）

先看createPbftReq这个函数：

```
// consensus/pbft/mock_utilities_test.go
func createPbftReq(tag int64, replica uint64) (req *Request) {
    tx := createTx(tag)
    txPacked := marshalTx(tx)
    req = &Request{
        Timestamp: tx.GetTimestamp(),
        ReplicaId: replica,
        Payload:   txPacked,
    }
    return
}
```

这里就是使用传过来的tag与replica构造了Request对象，其中tx的时间属性（Seconds）与tag有关在createTx还给定了tx的type，这些不是很重要，我们只要知道是通过tag和replica构造了一个请求行了。

继续看orderedRequests：

```
type orderedRequests struct {
```

```
    order    list.List
    presence map[string]*list.Element
}
```

它保存着一个列表，还有一个map，这里的map键是list元素的hash,值对应于list的元素。

继续看：wrapRequest函数

```
func (a *orderedRequests) wrapRequest(req *Request) requestContainer {
    return requestContainer{
        key: hash(req),
        req: req,
    }
}
```

就是把req变成 (hash(req), req)对，是不是很简单。。

后面的测试逻辑就很简单了，所以总的逻辑是：

创建空的orderedRequests并初始化

创建3个req

判断or有没有req1（此时为空，当然没有）

添加req1

判断or有没有req1（刚添加上，当然有）

判断or有没有req2（当然没有）

删掉r2（所以这个时候就可以得到源码里remove的作用，删除成功返回true，否则返回false）

删除r1（删除成功）

再删除r1（已为空，删除不成功）

检查or是否为空（为空）

添加r1,r2,r3到or

看最后一个是不是r3（这也体现出requeststore是顺序表）

第一个测试逻辑非常简单，但是可以让我们快速对源代码文件有了一定了解。

下一个测试函数是一个benchmark函数，本身非常的简单，我接触golang不久，要提的主要是b.N函数执行的次数，golang会使用不同的N来调用函数，多次测试取平均嘛，这个挺方便的。

另外测试结束后会提示：

```
100    1031034650 ns/op
```

它表示函数执行了100次，平次一次执行时间是1031034650纳秒。

测试到此就结束了，再看源码文件，测试没覆盖到的主要是：

```
type requestStore struct {
    outstandingRequests *orderedRequests
    pendingRequests     *orderedRequests
}
```

其他的函数基本上都是非常简单的，除了：

```
// getNextNonPending returns up to the next n outstanding, but not pending requests
func (rs *requestStore) getNextNonPending(n int) (result []*Request) {
    for oreqc := rs.outstandingRequests.order.Front(); oreqc != nil; oreqc = oreqc.Next() {
        oreq := oreqc.Value.(requestContainer)
        if rs.pendingRequests.has(oreq.key) {
            continue
        }
        result = append(result, oreq.req)
        if len(result) == n {
            break
        }
    }

    return result
}
```

这个函数主要是从outstandingRequests拿出不在pendingRequests的n个request。

上面就是这部分的内容，非常简单的代码，关键是看代码的一个思路，后面会对pbft其他部分进行分
。